WEBSITE BOOK EDITION

Pixels → Perception

從像素到理解：視覺、語言與學習

基礎・影像・語言・學習・理論

第 2 冊（共 4 冊）

自然語言與圖形識別

繁體中文版・第 2 冊

2026-11-05

https://blog.shyandram.dev/

本書說明

本冊是 Pixels → Perception 網站完整課程分成的第 2 冊（共 4 冊），主題為「自然語言與圖形識別」，收錄 22 篇已發布教學與 0 篇部落格文章，依網站的學習單元順序編排。這是網站內容的書籍排版版本，並非另經同儕審查的出版品。

正文、程式碼、公式與參考文獻沿用網站內容。練習提示在本書中直接展開；靜態影像保留，互動示範則提供網站連結。原本的影像比較器改列為輸入與輸出影像。篇末參考文獻的編號僅適用於該篇教學。

本站內容仍在審閱中。製作過程使用 AI 協助，內容可能仍有錯誤或疏漏；請搭配原始文獻閱讀，並以網站的後續修訂為準。

兩種語言、各冊分別提供 PDF。連到同冊教學的連結會在本 PDF 內跳轉；連到其他冊的教學、特定小節、互動操作、筆記本與外部資料仍連至網站。內容快照: 2026-11-05

來源版本: `2b3c30d`

https://blog.shyandram.dev/

目錄

I 自然語言處理 1

NLP 01・自然語言處理導論 2

NLP 02・語言學基礎 15

NLP 03・自然語言處理的類神經網路 38

NLP 04・詞彙語意與詞嵌入 61

NLP 05・用於語言的循環神經網路 86

NLP 06・以 RNN 做序列標註（詞性標註、資訊擷取） 109

NLP 07・注意力機制與 Transformer 134

NLP 08・以 Transformer 建構序列到序列模型（機器翻譯、聊天機器人） 151

NLP 09・以 Transformer 做語言模型（自然語言生成） 173

NLP 10・預訓練模型：BERT 206

NLP 11・以 BERT 做分類（情感分析、自然語言推論） 223

NLP 12・句法與語意任務 243

II 圖形識別 270

PR 01・模式、機率與貝氏決策 271

PR 02・密度估計與貝氏學習 287

PR 03・迴歸與分類的線性模型 303

PR 04・近鄰、核與非線性分類器 **320**

PR 05・特徵選擇、轉換與降維 **339**

PR 06・模板比對與結構化辨識 **357**

PR 07・分群、混合模型與 EM **374**

PR 08・近似推論與貝氏模型複雜度 **397**

PR 09・集成、驗證與分群有效性 **418**

PR 10・辨識專題：拒絕、漂移與學習式特徵 **440**

篇 I

自然語言處理

NLP 01・自然語言處理導論

什麼是 NLP、語言為何對電腦困難、領域如何從規則走向統計再走向預訓練模型、主要任務、中文特有的問題，以及從任務定義到錯誤診斷的實用處理流程。

網站版本: https://blog.shyandram.dev/zh-tw/learn/nlp/introduction/

你將學到

- 說明語言為何對電腦困難，並為每個語言學層次舉出一個歧義例子。
- 透過標誌性的論文，追溯 NLP 的主要時代：規則、統計學習、神經網路與預訓練模型。
- 依輸出形式分類常見 NLP 任務，並找到涵蓋各任務的教學。
- 認識中文特有的問題：斷詞，以及繁體字與簡體字兩套字集。
- 透過輸入、輸出與所需語言證據定義 NLP 任務，並在正規化與斷詞過程中保留這些證據。
- 建立資料切分，找出錯誤真正起源的處理階段。

先看全貌

自然語言處理（natural language processing, NLP）研究能分析、解讀或產生人類語言的電腦系統。NLP 流程把特定語言問題轉成計算。考慮「Pump B: not ready.」：維護應用可能需要設備名稱與陳述的就緒狀態；訊息分流應用可能只需要類別；對話助理則可能需要追問還有什麼沒完成。即使輸入字串相同，它們也是不同任務。

第一步是決定應用需要什麼答案，以及哪些證據支持答案。接著把文字轉成模型輸入，同時保留這些證據。前處理已抹去的差異，再強的模型也無法恢復；看似合理的標籤，若描述錯誤的目標，也沒有用。

> **用最簡單的話說**
>
> 想像把信件分類放進不同托盤。先決定每個托盤代表什麼，再閱讀信件，而且不能擦掉重要的字。若把「包裹沒有送到」變成「包裹送到了」，還沒選分類工具，問題就已經出錯。

本教學以 Hagiwara 從應用出發的觀點，以及 McRoy 對語言和相關資料結構的說明為基礎 [1, 2]。我們先看語言為何困難、領域如何走到今天的方法、它解決哪些任務，以及中文文字有什麼不同。接著沿著一條實用路線前進：定義任務、保留文字、選擇計算單位、建立資料，再評估結果。NLP 02 說明這些決策背後的語言結構，本教學最後的路線圖則串起整個系列。

1. 語言為何對電腦困難

白話版。同一串字元可以有不同意思，而意思取決於結構、語境與說話者的目的。除非我們提供其餘資訊，否則電腦只看得到字元。

歧義出現在語言分析的每個層次。下表每一列都會在 NLP 02 以例子與程式深入說明：

層次	例子	兩種解讀
詞邊界	學生會說話	「學生有說話能力」／「學生會在發言」
詞類	“the pilots can land” / “the metal can fell”	“can” 作助動詞／作名詞
詞義	“window”	牆上的開口／螢幕上的區域
附著	“The technician inspected the panel with the camera.”	相機是工具／相機裝在面板上
作用域	“Every technician uses a camera.”	可能是不同相機／一台共用相機
指稱	“Mei put a camera beside a lamp. It was overheating.”	“it” 是相機／是燈
意圖	“Can you lower the gain?”	詢問能力／請求動作

人們解決大多數這類歧義時幾乎不自覺，靠的是前後文、情境與世界知識。程式則必須被提供這些語境，或從資料中學會近似它們。另外三種性質讓困難加劇。

- **生產性**。說話者隨時用熟悉的零件組出新詞與新句子，因此沒有一份有限的輸入清單能涵蓋整個語言。
- **偏斜的頻率**。少數詞極為常見，大多數詞很罕見。NLP 02 用 Zipf 定律量測這件事：在一個語料庫中，三分之一的詞型別只出現一次。罕見詞正是模型證據最少的地方。
- **變異**。方言、語域、拼寫慣例、混合語言訊息與書寫系統都不相同。以某一變體訓練的系統，可能在另一變體上悄悄失敗。

這些都不表示 NLP 做不到，而是表示每個系統都在選擇保留哪些區別；好的工程會把這些選擇明確寫出來。

2. 簡史：規則、統計、神經網路與預訓練

白話版。早期系統依照人工撰寫的規則運作。後來的系統從大量文字集合學習機率。接著神經網路學會自己的特徵；今天，多數系統從一個以未標註文字預訓練的大型模型出發，再調整到任務上。



圖 *1* — 本教學引用的里程碑，分成四個大致的時代。事件等距排列，並未依年份比例；實際上各種方法彼此重疊。

規則與人工建立的知識。 Weizenbaum 的 ELIZA（1966）回應輸入的英文時，會比對關鍵詞，並套用腳本中的分解與重組規則；它最知名的腳本模仿心理治療師的對話風格 [6]。這個時代的系統以人工編寫文法、詞庫與模式。在設計範圍內它們可以很精確，但每遇到新的說法或領域，都需要新規則。

從語料庫進行統計學習。在接下來的數十年間，研究者越來越常從資料估計模型，而不是寫下每一條規則。Brown 等人（1993）描述一系列統計翻譯模型，其參數由互為翻譯的句子對估計而得 [7]。同一年，Penn Treebank 釋出一個標註詞性與句法結構的大型英文語料庫 [8]，讓標記器與剖析器有共同的訓練資料與共同的測試。關鍵改變是證據來自語料庫中的計數，而在保留資料上評估也成為標準做法。

神經網路。 Bengio 等人（2003）訓練一個神經語言模型，同時學習每個詞的分散式表示與詞序列的機率 [9]。Collobert 等人（2011）用同一個神經架構處理詞性標註、詞組切分（chunking）、命名實體辨識與語意角色標註，主要從未標註文字學習內部表示，而不是依賴人工設計的特徵 [10]。Mikolov 等人（2013）讓詞向量能在大型語料庫上低成本地訓練 [11]。這時的模型仍傾向一次只為一個任務建立。

預訓練模型與大型語言模型。 Transformer（2017）以注意力取代循環結構 [12]。ELMo（2018）從預訓練的雙向語言模型提供依語境而變的詞表示 [13]；BERT（2019）在未標註文字上預訓練深層雙向 Transformer，再為每個任務加上小型輸出層做微調 [14]。GPT-3（2020）顯示大型自迴歸模型只要在提示中看到幾個例子，不需梯度更新，就能執行許多任務 [15]；InstructGPT（2022）則以人類回饋微調這類模型，使它遵循指令 [16]。Zhao 等人的綜述以預訓練、適應調整、使用方式與能力評估四個面向，組織大型語言模型的文獻 [17]。

各時代彼此重疊：規則式元件、計數方法與神經模型都仍在使用。能延續下去的是第 4–9 節的流程。不論用哪種模型，總要有人定義任務、保留證據、選擇單位、建立資料並進行評估。

3. 主要任務與應用

白話版。大多數 NLP 任務可以依輸出分組：整段文字一個標籤、每個詞一個標籤、一個結構、從集合中找出的段落，或新的文字。

輸出形式	任務例子	應用	對應教學
每段文字或文字對一個標籤	主題分類、情感分析、垃圾訊息偵測、自然語言推論	分流、內容審核、評論分析	NLP 03、NLP 11
每個 token 或範圍一個標籤	詞性標註、命名實體辨識、資訊擷取	把報告轉成紀錄	NLP 06
語言結構	剖析、語意角色、共指	分析誰對誰做了什麼	NLP 12
從集合中找證據	搜尋、擷取式問答	文件搜尋、客服	NLP 11、NLP 09
依輸入產生新文字	翻譯、摘要、對話	翻譯工具、助理、聊天機器人	NLP 08、NLP 09
可重用的表示	詞與句子嵌入、預訓練編碼器	相似度、分群、遷移到新任務	NLP 04、NLP 10

同一個應用常結合好幾列。回答維修問題的助理，可能要分類問題、檢索段落、擷取設備名稱，再產生回覆。每一步都有自己的輸出與錯誤，所以本教學其餘部分把任務定義當成第一個設計決策。

4. 選模型前先規定任務

白話版。「理解這則訊息」太籠統，無法直接訓練或評估。先指定哪些意思必須變成輸出。

對原創訊息「Pump B: not ready.」，三種可能輸出是：

任務	輸出	重要證據
主題分類	`maintenance`	訊息談的是設備狀態
實體擷取	`Pump B`，附上原文範圍	設備名稱的確切字元
狀態擷取	設備 `Pump B`；就緒狀態 `false`	名稱與否定就緒的陳述

第一個答案不能證明第三個答案。分類器可能只靠「pump」辨認主題，完全忽略「not」。狀態擷取器則必須保留否定。回傳設備名稱的系統還需要邊界規則：是否包含 `B`？是否排除冒號？把這些決定寫進標註說明。

接著比較「Pump B was not ready yesterday」與「Pump B is ready now」。設備相同，時間與肯定／否定卻不同。即時狀態應用需要處理多個帶日期陳述的規則；歷史擷取器可能保留兩者。正確答案取決於任務定義與可用脈絡，而不是字串唯一通用的解讀。

對「It still isn't ready」，訊息沒有寫設備名稱。若應用提供前文對話，指涉解析就能使用它；若只有這句孤立文字，辨認出 `Pump B` 就是在加入從未提供的資訊。這是輸入設計問題，NLP 12（共指）與 NLP 08（對話狀態）會進一步討論。

預測與執行要分開。預測出重新安排的請求，並不能決定該修改哪一筆預約。先從能和文字核對的輸出開始，例如意圖標籤、設備名稱範圍，或明確陳述的事件。後續教學再加入解析缺漏細節所需的情境推理。

5. 保留原始文字與證據

白話版。模型輸入與畫面上的證據文字，可以是同一則訊息的不同表示。保留能連起兩者的方式。

處理流程可寫成：

$$r \xrightarrow{g} x \xrightarrow{f_\theta} \hat{y}.$$

其中 r 是原始輸入紀錄；g 是前處理與數值編碼函數；x 是模型輸入；f_θ 是參數為 θ 的預測器；$\hat{y}$ 是輸出。紀錄可包含文字與允許使用的脈絡。前處理涵蓋正規化、token 邊界與詞彙 ID 等決策。

若兩筆不同紀錄變成同一個 x，確定性預測器就必須對兩者給出相同結果：

$$g(r_1) = g(r_2) \quad \Longrightarrow \quad f_\theta(g(r_1)) = f_\theta(g(r_2)).$$

其中 r_1, r_2 是兩筆原始紀錄。這個簡單等式解釋了為什麼前處理也是模型的一部分。若移除「not」使肯定與否定陳述完全相同，之後再改神經架構也無法區分。把字母轉成小寫而合併 `US` 與 `us`，或移除標點而讓小數變成另一個字串，也有同樣問題。

有些等價關係很有用。Unicode NFC 正規化把標準等價的字元序列映成共同形式 [3]。這和自行決定移除重音、標點或大小寫不同：前者處理指定的字元等價關係，後者可能抹掉任務需要的差異。應先決定本應用中哪些輸入應視為等價，而不是把所有轉換都當成無害清理。

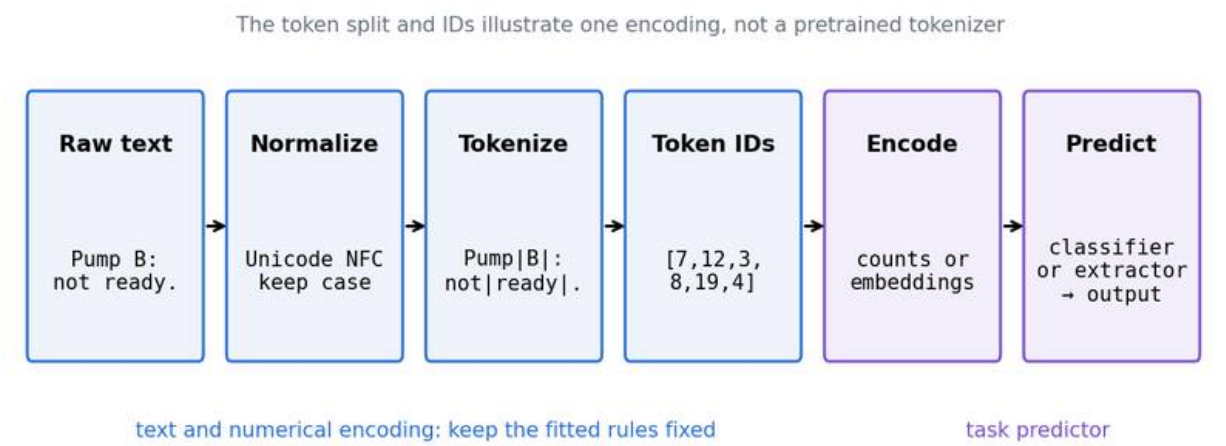


圖 2 — 示意文字處理流程。保留與原始文字的連結，訓練與評估時使用同一套已擬合編碼。

正規化與文字範圍的算例

`Café` 可以使用預組合的重音字母，也可以用基本字母 `e` 接上一個組合重音。兩者看起來相似，卻占用不同數量的 Unicode 碼位（code point）[3]。擷取器若回傳字元位置，正規化就可能讓後面的文字範圍位移。

文字範圍（span）$[a, b)$ 表示在**指定字串**中，從位移 a 開始、到 b 之前結束的字元。兩個位移都是非負整數，且 $a \le b$。不知道它們對應哪種文字表示與索引單位，位置本身就沒有意義。

```
import re
import unicodedata

raw = 'Café Pump B: not ready.'
normalized = unicodedata.normalize('NFC', raw)
needle = 'Pump B'
raw_start = raw.index(needle)
normal_start = normalized.index(needle)
raw_span = (raw_start, raw_start + len(needle))
normal_span = (normal_start, normal_start + len(needle))
tokens = [(m.group(), m.start(), m.end())
          for m in re.finditer(r'\w+|[^\w\s]', normalized)]
assert normalized != raw
assert unicodedata.normalize('NFC', normalized) == normalized
assert raw[raw_span[0]:raw_span[1]] == needle
assert raw[normal_span[0]:normal_span[1]] != needle
assert all(normalized[start:end] == token for token, start, end in tokens)
print('code points:', len(raw), '->', len(normalized))
print('raw / normalized evidence spans:', raw_span, normal_span)
print('normalized tokens:', [token for token, _, _ in tokens])
```

程式輸出碼位數 `24 -> 23`。原始設備名稱範圍為 `(6, 12)`，正規化後為 `(5, 11)`。兩者在各自字串中都選出 `Pump B`；把第二個範圍套到第一個字串，卻會選錯字元。最後一個斷言檢查正規化 token 與其來源位移是否對應。

做擷取時，可以維護轉換位置到原文的對齊，也可以直接在原始字串保留標註。本例展示錯誤如何發生，沒有實作通用對齊演算法。NLP 06 會說明詞、token 與字元範圍如何成為結構化輸出。

6. 決定計算使用哪些單位

白話版。詞、字元與模型 token 回答不同問題。選計算單位時，不要把它誤當成語言的所有單位。

在「a cat and a dog」中，空白切分得到五個 **token 實例**（occurrences）與四種不同的**型別**（types）：`a`、`cat`、`and`、`dog`。詞彙表替每個保留型別指定 ID。ID 是表格項目的名字，ID 7 並不比 ID 3 表達更大的意思。

「Pump B: not ready.」則帶來不同選擇。空白切分會把標點黏在詞上；辨識標點的切分器可能產生 `Pump`、`B`、`:`、`not`、`ready`、`.`；子詞斷詞器還可能把詞切得更細。它們都可能是有效的計算編碼，但 token 位置不同。詞標籤必須對齊模型實際使用的編碼。

子詞斷詞器從資料學習要保留哪些片段。NLP 02 會逐步訓練一個小型位元組對編碼（byte-pair encoding, BPE）斷詞器，並與 WordPiece、unigram 模型及 SentencePiece 比較。它的核心結論是：學到的片段反映頻率而不是構詞，而斷詞器是一整套已擬合的程序，不只是一份詞彙清單。

詞彙覆蓋率與未知輸入

詞彙表只包含保留的項目。若把沒見過的詞映到未知標記，所有這類詞都共享同一個 ID。任務若主要依賴熟悉的周圍詞，這可能可接受；但未見設備名稱與未見帳戶名稱之間的差異也會消失。

定義一個示意的未知 token 比例：

$$u = \frac{\sum_{i=1}^{N} \mathbf{1}[t_i \notin V]}{N}.$$

其中 u 是未知 token 比例；t_i 是指定斷詞器下第 i 個 token 實例；V 是擬合後詞彙表；$N > 0$ 是評估的出現總數；$\mathbf{1}[\cdot]$ 在條件成立時為一。重複出現的未知詞會重複計數。這量測缺少詞彙項目的比例，不是失去多少意思，也不是任務準確度。

7. 中文文字：斷詞與兩套字集

白話版。中文書寫的詞與詞之間沒有空格，所以決定詞的起訖本身就是一種分析。中文也以繁體字與簡體字兩套標準字集書寫，在其中一套上擬合的模型，看到另一套時會遇到許多陌生字元。

斷詞

中文通常沒有詞間空白。「學生會說話」可以切成 **學生 / 會 / 說話**（學生有說話能力），也可以切成 **學生會 / 說話**（學生會在發言）；脈絡有助選出原本的意思。模型斷詞器也可能只使用字元或子詞片段，其切分未必是完整詞法或語法分析。NLP 02 用一個小程式列舉兩種解讀，並區分這些語言結構。

斷詞也是一種慣例。第二屆國際中文斷詞評測使用四個語料庫：兩個為繁體字（臺灣的中央研究院與香港城市大學），兩個為簡體字（北京大學與微軟研究院），每個都依提供者自己的標準切分 [18]。當資料集有詞層級標籤時，它依循的標準就是任務定義的一部分；以字元為單位的模型在評估前，必須把輸出對齊到這些詞。

繁體字與簡體字

臺灣與香港的文字主要以繁體字書寫，中國大陸的文字則以簡體字書寫；上述評測語料庫也依循這個分界 [18]。許多字在兩套字集中相同，但也有許多字是不同的 Unicode 碼位：繁體的「機」是 U+6A5F，其簡體對應字則是 U+673A。Unicode 正規化不會合併它們，因為它們是不同的字元，而不是同一字元的不同編碼方式。

轉換不是一對一的字元替換。Unihan 資料庫為每個字記錄繁簡異體，並指出語境可能決定對應：從簡體轉成繁體時，U+810F 表示「內臟」時對應「臟」，表示「不乾淨」時對應「髒」[19]。不同地區的用詞也可能不同，所以即使字元轉換完美，也無法把一種變體的文字變成另一種變體的文字。

字集不一致會直接反映在詞彙覆蓋率上。下列程式在一句繁體句子上擬合字元詞彙表，再對同一句新句子的兩種寫法量測未知 token 比例 u。簡體版本以跳脫碼書寫，以便看出哪些字元不同。

```
vocab = set("這台相機已經校正好了")              # character vocabulary fitted on Traditional
 ↪  text
tests = {"Traditional": " 那台相機已經過熱了",
         # The same sentence in Simplified characters, written as escapes:
         "Simplified": " 那台相\u673a 已\u7ecf\u8fc7\u70ed 了"}
for name, text in tests.items():
    unknown = [f"U+{ord(ch):04X}" for ch in text if ch not in vocab]
    print(f"{name:11s} u = {len(unknown)}/{len(text)}  unknown: {unknown}")
```

繁體句子的 $u = 3/9$：「那」、「過」、「熱」不在這段極小的訓練文字中。簡體句子的 $u = 5/9$：「那」(兩套字集相同）以及「過」、「熱」的簡體形式仍然未知，另外再加上「機」與「經」的簡體對應字，因為詞彙表只有它們的繁體形式。真實詞彙表大得多，但效果在性質上相同：主要以一套字集訓練的模型，對另一套字集的證據較少。請決定要同時用兩套字集訓練、用考慮語境的工具轉換，還是分別評估各字集，並說明你的選擇。

8. 建立真正回答任務的樣本與標籤

白話版。語料庫是觀測集合；訓練樣本則依任務從觀測建立。建立方式決定模型學什麼。

分類樣本可以是完整訊息配上一個類別；擷取樣本可以是訊息配上字元範圍與標籤；語言建模則由文字本身提供預測目標，不需要類別標註。這些差異解釋了為什麼同一份語料能支援不同目標 [1, 2]。

假設維護訊息含有「the camera is fine, but the delivery was late」。產品情緒標籤、配送情緒標籤與整則訊息標籤未必一致。標註說明應包含目標。標註者若選不同目標，分歧首先反映任務定義問題，還不能直接解讀成模型品質問題。

定義資料切分的獨立單位。同一件客訴的兩種改寫，雖然是不同字串，卻可能描述同一事件。一則放訓練、一則放測試，可能讓測試獎勵辨認該事件，而不是轉移到新事件。相關訊息、對話或文件修訂若共享資訊而可能洩漏，就應整組切分。

須追蹤兩種擬合狀態：預測器從訓練標籤學權重，斷詞器或詞彙表可能從未標註訓練文字學習。若特徵流程仍用測試文字決定詞彙表，只隔離測試標籤還不夠。兩者都須依實驗規則固定。

用小型基準診斷表示方式

先從容易檢查行為的表示開始。訊息的計數表示保留哪些詞出現，卻忽略順序。因此「The dog bit the man」與「The man bit the dog」即使反轉誰對誰做了什麼，仍有相同計數。只用這些計數的確定性分類器，必須對兩者給出相同預測。

這是精確的表示限制。只要特徵映射仍相同，更多樣本也無法分開這組句子。局部片語特徵能分開部分語序差異；脈絡模型可使用更豐富的輸入互動。NLP 03 會推導並測試這些選擇。針對實際失敗增加結構，實驗才能辨認改變了什麼。

9. 評估輸出，診斷錯誤原因

白話版。評估應告訴你哪裡出錯、該改善哪裡。單一彙總分數通常找不出原因。

分類時檢查每個類別的錯誤，尤其罕見類別；準確率（accuracy）可能掩蓋某類從未被預測的問題。精確率（precision）問回傳項目有多少正確，召回率（recall）問必要項目找回多少。兩者都取決於如何定義項目 [5]。擷取找到正確詞卻回傳錯誤範圍，是邊界錯誤；狀態紀錄配對正確設備卻判錯肯定／否定，是意思錯誤。

保留幾組只改變一個重要語言差異的挑戰案例，例如「ready」與「not ready」、`Pump A` 與 `Pump B`、「yesterday」與「now」。固定任務，只改變該差異。普通訊息答對，還不能說明系統是否使用了這些案例所需的區別。

觀察	優先檢查	改善必須證明什麼
未見設備名稱被合併成同一表示	詞彙表與備援方式	名稱在輸入中仍可區分
詞正確，證據反白範圍錯誤	正規化與位移對齊	回傳位移對應畫面原文
否定使真實標籤反轉，預測卻不變	特徵映射與監督訊號	相關差異保留於編碼，並影響決策
驗證結果好，新客戶結果差	切分獨立性與領域差異	表現能轉移到目標群體
在一套中文字集上表現好，另一套很差	字元覆蓋率與轉換	兩套字集都被涵蓋，或明確回報差距

盡可能一次只改一個階段。前處理和模型一起改，即使分數提高，也無法知道誰造成改善。解讀權重或輸出前，先檢查轉換後輸入。這讓錯誤分析成為可測試的解釋，而不是一串模型名字。

10. NLP 系列路線圖

本系列依循第 2 節的時代順序：先談語言結構，再談神經網路基礎、序列模型、注意力與 Transformer，最後是預訓練模型與結構化任務。

教學	內容
NLP 01・自然語言處理導論	語言為何困難、歷史、任務、中文問題，以及從任務到評估的流程
NLP 02・語言學基礎	構詞、token 與子詞、詞性、句法、意義、語境、語料庫與 Zipf 定律
NLP 03・自然語言處理的類神經網路	計數特徵、TF–IDF、邏輯迴歸與神經網路文字分類器
NLP 04・詞彙語意與詞嵌入	分布式意義、計數式向量、word2vec
NLP 05・用於語言的循環神經網路	n-gram 語言模型、困惑度、循環語言模型
NLP 06・以 RNN 做序列標註	詞性標註、實體辨識與範圍評估
NLP 07・注意力機制與 Transformer	注意力、語境表示與位置
NLP 08・以 Transformer 建構序列到序列模型	編碼器–解碼器模型、解碼、機器翻譯與聊天機器人
NLP 09・以 Transformer 做語言模型	因果語言模型、文字生成與檢索增強生成
NLP 10・預訓練模型：BERT	遮罩語言預訓練與 BERT 編碼器
NLP 11・以 BERT 做分類	情感分析、自然語言推論與擷取式問答輸出層
NLP 12・句法與語意任務	剖析、語意角色、共指與篇章

本站其他教學涵蓋一般機制：ML 02 介紹神經網路，ML 04 介紹循環網路，ML 05 介紹注意力與 Transformer 區塊，IT 01 介紹熵與交叉熵。

第一次讀論文的路線。先讀章節正文；這十二篇導讀是選讀檢查點，不是額外先修要求。每篇開頭說明先備知識，結尾提供下一步。路線依問題分組，不照發表年份排列；理解注意力動機時，可依 Seq2Seq → Bahdanau → Transformer 閱讀。

論文路線 1・文字單位與詞向量

先理解：斷詞、向量內積、交叉熵

閱讀順序： BPE → word2vec → GloVe

理解檢查： token 是什麼？哪些證據賦予向量意義？

論文路線 2・局部與序列決策

先理解：卷積、循環狀態、條件機率

閱讀順序： TextCNN → Seq2Seq → BiLSTM-CRF

理解檢查：局部特徵如何變成類別、序列或受約束的標籤路徑？

論文路線 3・注意力與預訓練

先理解：編碼器—解碼器流程、softmax、遮罩

閱讀順序： Bahdanau attention → Transformer → BERT

理解檢查：模型能讀哪些上下文？訓練它預測什麼？

論文路線 4・任務目標與結構

先理解：微調、句對、跨度評估

閱讀順序： Sentence-BERT → InstructGPT → Coreference

理解檢查：相似度、人類偏好與實體連結需要哪些不同證據？

每讀一篇，試著解釋原圖的一條箭頭、定義目標式中的符號，並指出一個結果所用的資料切分與模型設定。若有不清楚的步驟，先回到導讀連結的所屬章節再繼續。

現代觀點

Chang 等人的評估綜述整理語言模型應評估什麼、在哪裡評估與如何評估 [4]。對處理流程而言，這促使我們分別檢查編碼、任務輸出與應用行為。生成解釋再流暢，也不是設備名稱召回率的另一種量測；兩種輸出回答不同問題。

Zhao 等人的大型語言模型綜述描述了目前的預設做法：在廣泛文字上預訓練一個大型模型，再加以適應（例如指令調整或對齊人類偏好），透過提示使用它，並在許多任務上評估其能力 [17]。它的組織方式呼應第 2 節的歷史：每個時代都沿用前一時代的任務定義，改變的是從文字得到輸出的方式。

對現今系統，可沿用的原則是比較模型家族前先檢查介面。相關文字沒有被截斷，較大的輸入預算才有用；表示能支持任務所需差異（包括使用者同時以兩套中文字集書寫時），多語斷詞器才有用。預訓練表示可能減少所需標註資料，但應用目標仍需要明確監督與評估。後續教學會分析各自機制，而不是把這些當成自動得到的好處。

開放問題包含如何量測超越字元覆蓋的語言覆蓋、切分如何影響不同語言與語域，以及如何區分表示失敗、標籤不足與脈絡缺漏。一個有用的小研究，是在固定任務、獨立切分資料與幾個語言診斷案例下比較兩種編碼。回報各編碼保留或失去什麼，再據此解釋觀察到的錯誤。

重點整理

- 語言在每個層次都有歧義，從詞邊界到意圖；語境與世界知識能解決字串本身無法決定的部分。
- NLP 從人工規則走向語料統計，再走向神經網路與預訓練模型；任務定義與評估問題則原封不動地延續下來。
- 依輸出形式分組任務：文字標籤、token 標籤、結構、檢索證據、生成文字或表示。
- 中文多了兩個問題：依不同標準進行的斷詞，以及轉換並非一對一的兩套字集。
- 選模型前，先定義輸出與支持它的語言證據，並區分原始文字與轉換後模型輸入。
- 預測器無法恢復前處理函數已抹掉的差異；token 與詞彙 ID 是計算單位，不是完整語言分析。
- 只在允許的訓練文字上擬合學習式表示，切分時把相關觀測放在同組；改架構前，先診斷詞彙、邊界、意思與轉移錯誤。

練習

1. 對「Pump B was not ready yesterday, but it is ready now」，定義主題分類、歷史狀態擷取與目前狀態擷取的標籤。哪些標籤需要時間處理規則？

   提示

   設備名稱可以不變，狀態值與有效時間卻不同。先決定回傳一個狀態，還是兩個帶日期事件。

2. 把正規化範例中的分解重音換成預組合重音。說明哪些位移改變，哪些斷言仍有效。

   提示

   分別比較原文與正規化字串。位移是某個表示中的位置，不是所有外觀等價字串中的共同位置。

3. 讓一個前處理規則移除「not」。建構兩則所需狀態標籤不同、轉換輸入卻相同的訊息，證明確定性預測器為什麼不能分開它們。

提示

套用包含 $g(r_1)$ 與 $g(r_2)$ 的等式。矛盾在訓練前就已出現。

4. 在字集覆蓋程式中，改用繁體訓練句與其簡體版本的聯集擬合詞彙表，重新計算兩個測試句的 u。接著解釋為什麼只靠一張字元對照表，不一定能在查詢前把每個簡體測試字轉成繁體。

提示

聯集能消除字集造成的未知字，卻無法消除「那」、「過」、「熱」。關於轉換，請參考 Unihan 的例子：同一個簡體碼位依意思對應到兩個不同的繁體字。

5. 把同一事件的三則改寫訊息分組，建立訓練／測試資料集。設計切分與兩種診斷，區分事件記憶與對新事件的轉移。

提示

按事件整組切分，再分別檢查未見名稱與未見措辭。標籤規則保持固定。

參考文獻

1. Masato Hagiwara, *Real-World Natural Language Processing: Practical Applications with Deep Learning*, Chapters 1–3 and 10–11, Manning, 2021. 出版社
2. Susan McRoy, *Principles of Natural Language Processing*, open textbook, 2021. 書籍
3. Unicode Consortium, “Unicode Normalization Forms,” Unicode Standard Annex #15, revision 58, 2026. 規格
4. Yupeng Chang et al., “A Survey on Evaluation of Large Language Models,” *ACM Transactions on Intelligent Systems and Technology*, 2024. 綜述
5. Scikit-learn developers, “Metrics and Scoring: Quantifying the Quality of Predictions,” documentation, current online edition. 文件
6. Joseph Weizenbaum, “ELIZA—A Computer Program for the Study of Natural Language Communication Between Man and Machine,” *Communications of the ACM*, 9(1), pp. 36–45, 1966. DOI
7. Peter F. Brown, Stephen A. Della Pietra, Vincent J. Della Pietra and Robert L. Mercer, “The Mathematics of Statistical Machine Translation: Parameter Estimation,” *Computational Linguistics*, 19(2), 1993. 論文
8. Mitchell P. Marcus, Beatrice Santorini and Mary Ann Marcinkiewicz, “Building a Large Annotated Corpus of English: The Penn Treebank,” *Computational Linguistics*, 19(2), 1993. 論文
9. Yoshua Bengio, Réjean Ducharme, Pascal Vincent and Christian Jauvin, “A Neural Probabilistic Language Model,” *Journal of Machine Learning Research*, 3, pp. 1137–1155, 2003. 論文
10. Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu and Pavel Kuksa, “Natural Language Processing (Almost) from Scratch,” *Journal of Machine Learning Research*, 12, pp. 2493–2537, 2011. 論文

11. Tomas Mikolov, Kai Chen, Greg Corrado and Jeffrey Dean, “Efficient Estimation of Word Representations in Vector Space,” arXiv:1301.3781, 2013. 論文
12. Ashish Vaswani et al., “Attention Is All You Need,” *NeurIPS*, 2017. 論文
13. Matthew E. Peters et al., “Deep Contextualized Word Representations,” *NAACL-HLT*, 2018. 論文
14. Jacob Devlin, Ming-Wei Chang, Kenton Lee and Kristina Toutanova, “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding,” *NAACL-HLT*, 2019. 論文
15. Tom B. Brown et al., “Language Models are Few-Shot Learners,” *NeurIPS*, 2020. 論文
16. Long Ouyang et al., “Training Language Models to Follow Instructions with Human Feedback,” *NeurIPS*, 2022. 論文
17. Wayne Xin Zhao et al., “A Survey of Large Language Models,” arXiv:2303.18223, 2023 (continuously updated). 綜述
18. Thomas Emerson, “The Second International Chinese Word Segmentation Bakeoff,” *Proceedings of the Fourth SIGHAN Workshop on Chinese Language Processing*, 2005. 論文
19. Unicode Consortium, “Unicode Han Database (Unihan),” Unicode Standard Annex #38, revision 41, 2026. 規格

NLP 02・語言學基礎

每個 NLP 系統都要面對的語言結構：單位與構詞、token 與子詞、詞性、詞組與依存結構、意義、語境與篇章，以及語料庫、標註與 Zipf 定律。

網站版本: https://blog.shyandram.dev/zh-tw/learn/nlp/linguistic-essentials/

你將學到

- 區分語言及其口語、手語、書寫形式，並區分語言學單位與計算用的 token。
- 分析詞的內部結構，比較詞幹擷取（stemming）與詞形還原（lemmatization），並列舉中文斷詞的歧義。
- 訓練一個小型 BPE 斷詞器，並說明它與 WordPiece、unigram 模型及 SentencePiece 的關係。
- 使用 Universal Dependencies 詞性標記，並解釋詞組結構、文法關係與一致關係為何包含詞頻無法保留的資訊。
- 推導詞彙意義、組合意義、否定與量詞作用域，並區分斷言、預設、會話含意與篇章關係。
- 批判地閱讀語料庫與標註：標註者之間的一致度、Zipf 定律與罕見詞的長尾。

先看全貌

語言讓人描述情境、提出問題、協調行動，也改變他人的認知。看得到的句子只是這些活動的一部分。句子的內部結構限制可行的解讀；說話者也依賴共同看見的物件、先前對話，以及對發話理由的預期。

> **用最簡單的話說**
>
> 想像有人遞給你積木，說：「再蓋一次那一個。」知道積木的名稱，仍然不能告訴你「那一個」是哪棟建築、上次發生了什麼，以及是否要使用相同顏色。語言也是如此：詞提供零件，文法把它們連接起來，情境則補上說話者沒有說出的資訊。

考慮 “The technician inspected the panel with the camera.” 技師是使用相機檢查面板，還是檢查一個裝有相機的面板？知道每個詞的字典義，仍然無法決定答案。再考慮 “Can you inspect it tomorrow?” 現在，受檢查的物件、參照日期，以及這句話是否在提出請求，都依賴語境。有用的分析會先辨認這些選項，再選擇表示方式或標籤。

本教學替 Hagiwara 的 *Real-World Natural Language Processing* 所採取的實作路線 [1]，以及 McRoy 的 *Principles of Natural Language Processing* 所採取的符號與計算路線 [2]，建立語言基礎。開放教科書 *Essentials of Linguistics* 提供補充的語言學背景 [3]。本教學範例均為原創。NLP 01 介紹了從任務到評估的處理流程；本教學說明這個流程必須保留的單位與結構。後續教學再把它們變成模型：NLP 06 標註詞性與實體，NLP 12 剖析句子、標註語意角色並解析共指。這些層次彼此影響，不表示軟體必須依序使用互相獨立的模組。

1. 語言、模態與單位

白話版。口語、手語與書寫是表達語言的不同方式，各自記錄不同資訊。電腦的「token」則是另一種為了處理而選定的單位。

語言是一套有結構的表達與解讀系統。口語與書寫是表達語言的方式，兩者不是可以任意互換的容器。手語使用有系統的視覺與手部表達，觸覺手語則提供另一種模態（modality）[4]。文字逐字稿保留互動中的部分資訊，卻省略時間節奏、視線、手勢，以及大部分實際環境。

語音與書寫符號回答不同問題

語音學（phonetics）研究語音的物理產生與感知；**音韻學（phonology）**研究語音在某個語言中的規律。**音素（phoneme）**是特定語言系統中具有區辨作用的語音單位；不同的物理實現可能屬於同一音素 [5]。**字素（grapheme）**則是書寫系統的單位。兩者未必一對一對應。

英文 “ship” 有四個字母，但開頭的 “sh” 表示一個子音。反過來，同一字母在不同詞裡也可能有不同作用。因此，計算字母數量無法直接回答語音結構的問題。即使使用者想表達相同的詞，語音應用與文字應用仍可能需要不同的標註。

聲調（tone）與語調（intonation）也不同 [6]。華語 mā、má、mǎ、mà 分別區分常見的書寫詞「媽、麻、馬、罵」。聲調形態參與詞彙上的區辨；整段發話的音高輪廓也可能傳達疑問或強調。在英文裡，“Really.” 與 “Really?” 在書寫上暗示不同解讀，但標點只能粗略模擬說話者如何表達不相信、好奇或確認。書寫的問號無法還原原本的聲音。

字元、詞與 token

詞形（word form）是實際出現的形式；**詞位（lexeme）**把相關詞形歸為同一詞彙項目。在簡化分析中，“repair”、“repairs” 與 “repaired” 都是動詞詞位 REPAIR 的詞形。**詞元（lemma）**是用來索引該項目的代表形式。因此，詞形還原（lemmatization）是一種標註決定，不表示不同詞形能在句子裡互換 [2, 3]。

在 “The technician repairs cameras” 與 “The technician repaired cameras” 中，這些區別很重要。把兩個動詞都歸入 REPAIR，有助於搜尋相關活動；但若丟掉實際詞形，就會失去現在與過去的對比。應用可以同時保留原始文字區段與詞元，不必認定某一表示一定要取代另一表示。

計算用的 **token** 是處理程序選定的輸入單位。它可能是一個詞、詞的一部分、標點，或一段字元序列。它的邊界不一定與音素、詞素或詞位一致。同樣地，中文字元也可能構成多字詞：「相機」這段書寫序列指稱一種物件。字元式系統可能接收兩個單位，詞導向分析則可能使用一個單位。兩種計數都不能單獨定義該片語包含多少個意義。

2. 構詞：詞的內部結構

白話版。詞有組成部分。有些部分帶有詞彙意義，有些帶有時態或單複數等文法資訊，而部分之間的分組方式也可能改變意義。

詞的結構可以同時傳達詞彙意義與文法資訊。**構詞學（morphology）**研究這種結構；**詞素（morpheme）**是形式與意義或文法功能之間最小的、反覆出現的配對 [7]。「最小」不表示「只有一個字母」，有意義的成分也不一定能獨立出現。複數詞尾、時態標記與派生前綴能提供有用證據，但不同語言不一定用相同片段或詞邊界表達這些性質。

屈折、派生與生產性的組合

屈折（inflection）對比改變詞的文法實現，例如 repair/repaired。**派生（derivation）**過程可以形成新的詞彙項目，例如 repair/repairable。**複合詞（compound）**結合詞基，例如 camera mount [3, 7]。這些是分析上的區別；只辨認詞尾的拼寫，不足以決定它的功能。

比較 “She repairs the camera” 與 “The repairs took a day” 裡的 “repairs”。前者是限定動詞的詞形，後者是複數名詞。若規則刪掉每個詞尾的 “s”，就會失去詞類，以及該詞尾出現的原因。這條規則也無法正確處理 mouse/mice 之類的不規則配對。分析的任務是解釋反覆出現的形式與功能規律，並明確表示例外。

生產性（productivity）表示一種模式能支持新組合，而不只是描述記住的清單。若工作室引入新的 “sensor cradle”，聽者可以根據熟悉的成分與情境解讀這個複合詞。然而，組合不會決定所有細節：它可能在組裝、運送或操作期間固定感測器。構詞可以提供概括的關係，語境則補上這次使用的確切實例。

中文也有詞的內部結構與生產性的組合。在「三台相機」中，「三」表達數量，「台」是與被計數物件搭配的量詞，「相機」命名物件。英文 “three cameras” 在這個例子裡把複數標記放在名詞上。這個對比指出不同的文法選擇，不表示中文詞缺少結構，也不表示英文名詞總是有明確的數量。

詞幹擷取與詞形還原

白話版。詞幹擷取器依規則切掉詞尾；詞形還原器則根據詞的文法類別查出字典形式。前者便宜而粗糙；後者需要更多知識，犯的明顯錯誤也較少。

詞幹擷取器（stemmer）透過改寫詞尾，把不同詞形映到共同字串，也就是**詞幹（stem）**。Porter 演算法是經典的英文例子：它依序套用數組詞尾規則，每條規則都附有對剩餘詞幹的條件，設計目的是在資訊檢索中把相關詞歸在一起 [16]。詞幹不必是真正的詞，只需要一致。**詞形還原器（lemmatizer）**回傳詞元，這需要詞庫，通常也需要詞性：作名詞時 “saw” 本身就是詞元，作動詞時卻是 SEE 的詞形。

下面的玩具版切除器只用四條有順序的規則，加上一個最短詞幹長度的條件；精神上接近 Porter 演算法，但簡單得多。查詢表則代替字典。

```
# A toy suffix stripper (a few Porter-style rules) versus a lookup lemmatizer.
RULES = [("ies", "y"), ("ing", ""), ("ed", ""), ("s", "")]   # tried in order; first match
 ↪  wins

def stem(word):
    for suffix, replacement in RULES:
        if word.endswith(suffix) and len(word) - len(suffix) >= 3:
            return word[: -len(suffix)] + replacement
    return word
```

```
# A lemmatizer needs the part of speech; this tiny table stands in for a dictionary.
LEMMAS = {("repairs", "VERB"): "repair", ("repairs", "NOUN"): "repair",
          ("repaired", "VERB"): "repair", ("mice", "NOUN"): "mouse",
          ("was", "AUX"): "be", ("batteries", "NOUN"): "battery",
          ("news", "NOUN"): "news", ("this", "DET"): "this"}

def lemmatize(word, pos):
    return LEMMAS.get((word, pos), word)

for word, pos in [("repairs", "VERB"), ("repaired", "VERB"), ("batteries", "NOUN"),
                  ("mice", "NOUN"), ("was", "AUX"), ("news", "NOUN"), ("this", "DET")]:
    print(f"{word:10s} stem={stem(word):9s} lemma={lemmatize(word, pos)}")
```

切除器能處理規則形式（`repairs`、`repaired`、`batteries` 都得到合理詞幹），卻讓 `mice` 與 `was` 原封不動，還把 `news` 變成 `new`、`this` 變成 `thi`。前兩個失敗來自不規則構詞；後兩個是**過度擷取（over-stemming）**：只是看起來像詞尾的字母被刪掉了。詞形還原器七個都對，但只因為表中已經有它們；沒見過的詞會原樣通過。兩種工具都不決定文法類別：`VERB` 或 `NOUN` 是呼叫者提供的，而那正是 NLP 06 中詞性標註器的工作。

該用哪一種取決於任務。對搜尋而言，合併 `repairs` 與 `repaired` 可以提高召回率，即使有少數無關詞撞在一起。對狀態擷取器而言，詞幹擷取器丟掉的時態可能正是答案。現代神經網路流程很少對輸入做詞幹擷取；它們保留原始形式，讓子詞 token 與脈絡承載構詞資訊，這也是下一節重要的原因之一。

相同零件可以用不同順序組合

假設有人使用 “an unsealable case” 這個描述，我們可以考慮兩種有用的結構分析：

- `un + (seal + able)`：無法密封的盒子。
- `(un + seal) + able`：可以解除密封的盒子。

第一種把 “un” 當作對形容詞的否定；第二種先把它當作逆轉密封動作的成分，再由 “able” 形成形容詞。兩種解讀對實際操作有相反含意。它們是否成立、是否自然，會受語境影響；因此本例示範可能的分析，不主張每位說話者都同樣常用兩種解讀。

兩種分析包含的零件完全相同。改變的是中間的分組，以及每個操作與哪種對象結合。同樣的問題也會出現在句子的附著與邏輯作用域：如果任務需要這個區別，表示方式就必須保留結構。

中文斷詞也是解讀問題

中文書寫不以空格標示詞邊界。原創字串「學生會說話」可以分析成「學生 / 會 / 說話」（學生有說話能力），或「學生會 / 說話」（學生會發言）。前者把「會」當作表達能力的成分，後者則把「學生會」當作複合名詞。因此，處理後插入的空格記錄了一種分析；沒有空格的原始字串並未直接提供這些邊界。

下列程式列舉一個明確的小型詞庫所允許的路徑。它不是通用的中文斷詞器，而是說明：選擇最長的匹配詞條，無法直接確定說話者的意圖。

```
from functools import lru_cache

text = " 學生會說話"
lexicon = (" 學生", " 學生會", " 會", " 說話")

@lru_cache(None)
def segment(start):
    if start == len(text):
        return ((),)
    paths = []
    for word in lexicon:
        if text.startswith(word, start):
            for rest in segment(start + len(word)):
                paths.append((word,) + rest)
    return tuple(paths)

for path in segment(0):
    print(" / ".join(path))
assert len(segment(0)) == 2
```

印出的兩條路徑就是上述解讀。每一步遞迴都消耗一個已知詞，終止條件只接受完整覆蓋字串的路徑。擴充詞庫可能增加選項，卻不會增加特定對話的證據。句法分析、前後文，或這段訊息與學生會會議相關的知識，可能讓其中一條路徑更合理。

連標準答案本身也是一種選擇。第二屆國際中文斷詞評測（Second International Chinese Word Segmentation Bakeoff）使用四個語料庫評估系統：兩個為繁體字（來自臺灣的中央研究院與香港城市大學），兩個為簡體字（來自北京大學與微軟研究院），而且每個提供者都依自己的標準切分資料 [27]。因此，同一種切法可能在某個語料庫「錯」、在另一個語料庫「對」。請說明你的標籤依循哪一種斷詞標準。

3. Token 與子詞

白話版。模型需要一份固定的單位清單。整詞會留下太多未知詞；單一字元會讓序列太長。子詞方法從資料學出折衷：常見字串成為單一 token，罕見字串則由較小片段拼出。

NLP 01 定義了未知 token 比例，並說明為何把每個沒見過的詞都映到同一標記會失去區別。必要字元都被涵蓋時，字元詞彙表能減少完整詞的未知問題，代價是序列更長。位元組編碼若有完整基礎位元組字母表，就能涵蓋任意輸入位元組，但單位未必對應可閱讀字元。子詞方法在詞彙表大小與序列長度間取捨；Mielke 等人的綜述整理了這些選擇 [22]。沒有一種切分對所有語言、任務與資源預算都最好。

學習式斷詞器通常擬合語料庫中的統計模式。這些模式有助重用常見片段，但常見片段未必是有意義的詞素（morpheme）；反過來，有意義的屈折變化也可能被切成數個片段。判斷模型能辨認什麼時，必須分清楚：斷詞提供編碼，後續計算仍須學習編碼片段如何組合。

逐步理解一種子詞方法：BPE

BPE 學習哪些相鄰片段值得合併，替常見組合建立符號，讓其他字串仍以較小片段的序列表示。Sennrich、Haddow 與 Birch 把位元組對編碼（byte-pair encoding, BPE）改用於罕見詞翻譯 [17]。下面的字元版教學實作在詞內運作：

1. 把每個訓練詞表示成字元，加上一個詞尾標記。
2. 計數相鄰符號對，依該詞的語料出現頻率加權。
3. 在這些詞表示中，把最常見符號對的每次出現都合併。
4. 儲存這次合併，重複固定次數。

合併會改變後續符號對計數。兩符號合成新符號後，新符號又能參與另一個符號對。因此演算法學得的是有順序的操作序列，而不是能任意套用順序的片段集合。

```
from collections import Counter

corpus = ("low low low low low lower lower newest newest newest newest newest "
          "newest widest widest widest").split()
# Start from characters; "_" marks the end of a word so merges respect boundaries.
words = Counter(tuple(w) + ("_",) for w in corpus)

def pair_counts(words):
    pairs = Counter()
    for symbols, freq in words.items():
        for a, b in zip(symbols, symbols[1:]):
            pairs[a, b] += freq
    return pairs

def merge(words, pair):
    out = Counter()
    for symbols, freq in words.items():
        s, i = [], 0
        while i < len(symbols):
            if i + 1 < len(symbols) and (symbols[i], symbols[i + 1]) == pair:
                s.append(symbols[i] + symbols[i + 1]); i += 2
            else:
                s.append(symbols[i]); i += 1
        out[tuple(s)] += freq
    return out

merges = []
for step in range(10):
    pairs = pair_counts(words)
    best = max(pairs, key=lambda p: (pairs[p], p))   # most frequent; ties broken by symbol
    ↪ order
    merges.append(best)
    words = merge(words, best)
print("merges:", [a + b for a, b in merges])

def tokenize(word):                    # apply the learned merges in order
    symbols = list(word) + ["_"]
    for pair in merges:
        symbols = list(merge(Counter({tuple(symbols): 1}), pair))[0]
```

```
    return symbols

for w in ["lowest", "newer", "wider"]:
    print(w, "->", tokenize(w))
```

輸出包含 `lowest -> ('low', 'est_')`、`newer -> ('ne', 'w', 'e', 'r', '_')` 與 `wider -> ('wi', 'd', 'e', 'r', '_')`。第一個詞不在訓練語料中，卻能用學得的片段重新組合字母。其他詞的編碼較長，反映哪些組合獲得合併。

合併順序為什麼重要

第一個合併為 `t_`，接著是 `st_` 與 `est_`。訓練詞中重複出現的結尾讓這些組合很常見。這在例子中像英文詞尾，但演算法沒有「找出詞尾」的指令，只依頻率做決策。同一拼法換一份語料，可能得到不同切分。

`low` 合併要等較小組合先形成後才會出現。編碼依儲存順序執行，才能重現擬合程序。只從最終符號中優先找最長片段，是另一種演算法，未必產生相同切分。平手規則也有影響：本程式以符號順序打破相同計數，讓執行可重現。

函數回傳片段，還不是數值 ID，也沒有證明能涵蓋任意未見字元。本例成功處理的新詞，都重用訓練出現過的字元；若一個詞含新字元，就需要指定備援方式與相容的詞彙查表。這分開了兩個問題：字串能否切分，以及每個回傳片段是否都有模型懂的 ID。

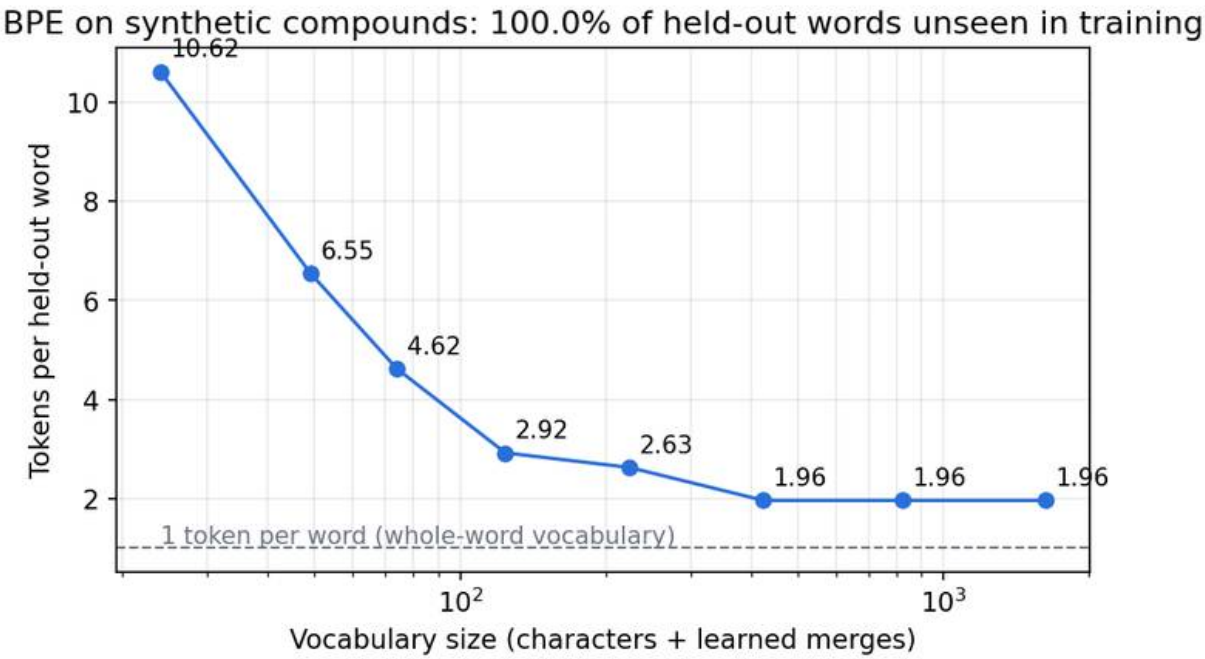


圖 *1* — 更多合併可能縮短編碼，但只看詞彙表大小，無法決定覆蓋率或語言品質。合成訓練樣本與保留樣本互相分開。

WordPiece、unigram 模型與 SentencePiece

BPE 只是其中一員。其他方法的差別在於如何選擇片段，以及如何切分新詞。

- **WordPiece** 源自日語與韓語語音搜尋 [18]。Google 神經機器翻譯系統描述它的版本時，說明是選出固定數量的片段，使切分後的訓練語料所用片段數最少，並在每個詞開頭加上邊界標記，讓原始詞序列能無歧義地還原 [19]。BERT 使用 WordPiece 詞彙表 [30]；NLP 10 會在此基礎上繼續。
- **Unigram 語言模型**切分由上而下：先從大型候選片段集合開始，把一個詞建模成獨立抽出的片段序列，再刪去對語料概似度貢獻最小的片段 [20]。因為它是機率模型，同一個詞可以有數種帶分數的切分；Kudo 利用這點在訓練時抽樣不同切分，作為正則化方法 [20]。
- **SentencePiece** 是工具組，而不是新的準則：它直接從原始句子訓練 BPE 或 unigram 模型，並把空白當成一般符號，因此不需要特定語言的預切分器，解碼也能還原編碼 [21]。這對中文、日文等不以空白分詞的語言很重要。

在這三種方法中，學到的片段都是統計上的，而不是構詞上的。子詞邊界剛好落在 `est`，是語料頻率的證據，並不是「`-est` 為最高級詞尾」的分析。

使用完整斷詞器，而不只是一份合併清單

斷詞軟體還會指定正規化、預切分、特殊標記與解碼 [23]。模型檢查點的 token ID 對應特定詞彙表與約定。只匹配斷詞器家族名稱並不足夠：兩個詞彙表可能把同一 ID 指派給不同符號。

預訓練模型應使用隨附的斷詞器與設定。若在實驗中訓練斷詞器，就只用允許的訓練文字擬合，評估前固定詞彙表與合併規則。分別查看片段、ID 與重建文字，各種檢視能找出不同錯誤。

僅線上版：*BPE — Neural Machine Translation of Rare Words with Subword Units* — *https://blog.shyandram.dev/zh-tw/learn/nlp/linguistic-essentials/papers/bpe–neural-machine-translation-of-rare-words-with-subword-units/*

4. 詞性

白話版。詞的類別描述它在句子中的行為。拼字能提供線索，但脈絡通常決定實際用法。

詞性（part of speech）描述名詞、動詞等語法類別。名詞通常是名詞片語的中心，動詞通常是子句述語的中心。這是分布上的描述，不是每個名詞都必須指實體物品的規則。「Decision」與「arrival」雖指抽象內容與事件，仍是名詞 [2]。

相同表面詞可以有不同類別。在原創例句「the pilots can land」與「the metal can fell」中，「can」扮演不同角色。第二句也顯示，句子其餘部分改變時，熟悉的局部片語可能誤導。標記器必須考慮允許的脈絡與標註規則。

Universal Dependencies 標記集

不同專案的標記集合各不相同，讓跨語言比較變得困難。**Universal Dependencies**（UD）是一套對詞性、構詞特徵與依存關係做一致跨語言標註的框架 [24]。它的通用詞性（UPOS）集合有 17 個標記 [25]：

類別	標記
開放類	`ADJ` 形容詞、`ADV` 副詞、`INTJ` 感嘆詞、`NOUN` 名詞、`PROPN` 專有名詞、`VERB` 動詞
封閉類	`ADP` 介詞（adposition）、`AUX` 助動詞、`CCONJ` 對等連接詞、`DET` 限定詞、`NUM` 數詞、`PART` 小品詞、`PRON` 代名詞、`SCONJ` 從屬連接詞
其他	`PUNCT` 標點、`SYM` 符號、`X` 其他

UD 把粗略標記與 `Number=Plur`、`Tense=Past` 等**特徵**分開，因此作名詞的 “repairs” 可以標成 `NOUN` 加上 `Number=Plur`，而 “repaired” 是 `VERB` 加上 `Tense=Past`。各語言的樹庫還會為通用準則未規定的構式補充文件化的慣例。這個標記集是共用的標註方案，不代表每種語言都在相同位置劃分類別界線。

已標記句子可表示為：

$$\mathbf{x} = (x_1, \dots, x_T), \qquad \mathbf{y} = (y_1, \dots, y_T), \qquad y_t \in \mathcal{Y}.$$

其中 $\mathbf{x}$ 是輸入詞序列，x_t 是位置 t 的詞，T 是序列長度，$\mathbf{y}$ 是標記序列，y_t 是詞 x_t 的標記，$\mathcal{Y}$ 是有限標記集合，例如 17 個 UPOS 標記。t 與 T 指語言學上的詞位置，不會自動等同模型子詞位置。

標註也需要縮寫、標點、多詞表達與多種合理分析的規則。當模型使用子詞時，應定義如何把輸出對應回詞。只用第一個子詞表示是一種做法；池化所有子詞則是另一種。不論選哪個，標準標籤與解碼輸出都必須使用相同單位。用隱藏式馬可夫模型、Viterbi 解碼與循環神經網路標記器，從 $\mathbf{x}$ 預測 $\mathbf{y}$，是 NLP 06 的主題。

5. 句法：順序、分組與文法關係

白話版。文法把詞組成詞組，並以主詞、受詞等關係連接它們。相同的詞放進不同結構，可能描述不同事件。

文法組織詞的組合。詞序、詞組結構，以及主詞、受詞等關係，限制誰對誰做了什麼 [2, 3]。這些限制比相鄰詞的清單豐富。

詞頻為何會失去角色資訊

“Ada praised Bo” 與 “Bo praised Ada” 包含相同的詞，但普通主動句解讀會交換稱讚者與被稱讚者。中文「阿達稱讚小博 / 小博稱讚阿達」在這兩句裡形成相同對比。只計算詞頻的表示，會把每一組句子映射成相同的對象。

```
from collections import Counter

first = "Ada praised Bo".split()
second = "Bo praised Ada".split()

def roles(tokens):
    # Only our deliberately fixed three-word active-clause template.
    assert len(tokens) == 3 and tokens[1] == "praised"
    return {"praiser": tokens[0], "praised_person": tokens[2]}

print(Counter(first) == Counter(second))
print(roles(first))
print(roles(second))
assert roles(first) != roles(second)
```

詞頻比較印出 `True`，角色字典則不同。這是資訊流失的結果，不是訓練模型的效能測試。當兩個輸入已經變成相同詞頻後，任何只依賴那些詞頻的後續確定性函數，都無法還原原先是哪種順序。

句法角色也不等於語意角色。在 “Bo was praised by Ada” 裡，Bo 是文法主詞，但 Ada 仍然是執行稱讚的人。即使被動句描述同一事件，把主詞直接當成行動者仍會失敗。抽取任務應說明它需要文法依存關係、事件參與者，還是兩者都需要。

成分結構與依存關係描述不同面向

句法成分（constituent）是參與更大結構的單位。在 “The label beside the boxes is missing” 裡，名詞片語 “the label beside the boxes” 作為主詞。它的中心語（head）是 “label”；“beside the boxes” 修飾這個中心語。把整個片語換成 “it”，仍保留簡單的文法框架：“It is missing”。這類測試提供分組的證據，不表示每段相鄰子字串都是詞組 [8]。

一致關係（agreement）的對比 “The labels beside the box are missing” 說明為何只看距離不夠。在這些例子中，動詞與主詞的中心語一致，而不是與最近的名詞 “box” 一致。局部啟發式規則即使正確找到最近名詞，仍可能選錯單複數。

依存（dependency）表示則記錄詞之間的關係，例如動詞的主詞與受詞，或名詞的修飾語 [2]。UD 使用依存樹：每個詞恰好有一個中心語，並以帶標籤的關係連到它，例如名詞性主詞 `nsubj` [24]。成分結構凸顯分組，依存關係凸顯相連的中心語。兩者可以描述相關證據，不必有完全相同的結構或標註慣例。早期大型英文資源 Penn Treebank 標註的是成分樹 [26]；UD 則以依存關係標註許多語言 [24]。

在面板範例中，“with the camera” 可以修飾檢查事件，也可以修飾面板。下圖只保留這一個附著差異，省略限定詞與其他關係。它們是教學示意圖，不是依某一具名標註標準完成的剖析結果。

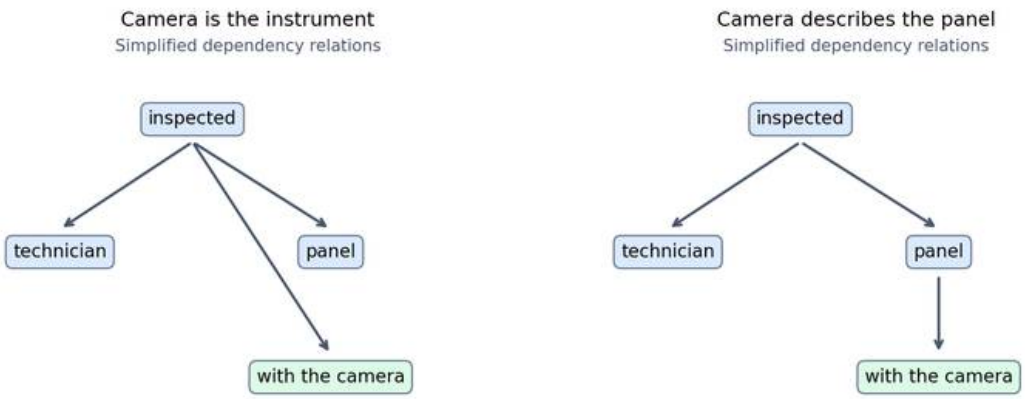


圖 2 — 相同的詞可以支持不同附著關係。選擇關係，也就選擇了不同解讀。

若附著於事件，相機是檢查的工具；若附著於名詞，相機協助辨認面板。前後文可以支持其中一種：例如技師解釋錄影方法，或有人區分有裝相機與沒有裝相機的面板。文法上的可能性不決定實際發生了哪一種情境。

這裡只是預覽。NLP 12 會深入文法、圖表剖析、機率式剖析與依存剖析。此處的核心問題是：表示需要保留什麼？答案包括分組、相連的中心語、角色方向，以及合理的替代分析。

6. 語意：意義與真值條件

白話版。語意學問一個表達式是什麼意思，以及世界要是什麼樣子，句子才為真。詞義重要，句子如何組合詞義也同樣重要。

句子可以很流暢，卻錯誤描述情境。**語意學（semantics）**研究表達式如何貢獻意義，以及有結構的句子如何決定斷言為真的條件 [2, 3]。把這些條件與會話推論分開，能讓歧義更容易分析。

詞彙意義依賴語境與關係

多義（polysemy）涉及同一表達式的相關義項；**同形異義（homonymy）**涉及共享形式的不同意義 [3]。window 可以指牆上的開口，也可以指螢幕上有邊界的區域。兩者的關係有助於解釋這種延伸，但若把 “window” 固定換成某一具體物件，就會漏掉螢幕上的解讀。字典可以列出義項，實際出現的詞仍需要解讀。NLP 04 會研究從語境學到的向量如何捕捉其中一些關係。

中文「開」出現在「開門」與「開會」中，英文分別常用 “open a door” 與 “hold a meeting”。把每個「開」都翻成同一英文動詞，會抹掉與構式相關的差異。意義屬於表達式在組合與語境中的使用，不是無視語境的中文字元到英文詞對照。

即使看似簡單的形容詞，也可能依賴比較類別。「小象」可能比「大螞蟻」大。這些表達式把動物與適當類別相比，而不是使用一個普遍的尺寸門檻。在一組只有玩具象的物件中，相關比較又可能改變。因此，名為「小」的標註，應說明比較的對象是什麼。

組合依賴結構

組合性（compositionality）指複合表達式的意義，取決於各部分的意義與組合方式 [3]。它不表示只要把詞義相加即可。“Bo praised Ada” 把參與者放入有方向的關係；重排句子會用不同方式組合同一組名字。附著差異也會改變修飾語限制哪個表達式。

慣用表達式也可能有約定俗成的意義，無法從獨立詞條預測。詞庫與文法互相補充：分析必須同時記錄慣例組合與生產性的組合。對應用而言，要問任務需要哪種區別。搜尋維修說明也許能接受概括的詞彙歸類；判斷維修是否完成，則需要事件、時態與否定資訊。

量詞作用域改變要求

“Every technician uses a camera” 有一種解讀，容許不同技師使用不同相機；在適當語境裡，也能理解成有一台所有技師共用的相機。這兩種條件不同：

$$A:\ \forall t \in T,\ \exists c \in C:\ U(t,c),$$
$$B:\ \exists c \in C,\ \forall t \in T:\ U(t,c).$$

其中 T 是技師集合，C 是相機集合，$U(t,c)$ 表示技師 t 使用相機 c，A 與 B 是兩種解讀的名稱；$\forall$ 表示「對每一個」，$\exists$ 表示「存在」。本例的兩個集合都非空。在 A 裡，變數 c 可以隨 t 改變；在 B 裡，同一個 c 必須適用於所有 t。

檢查 A 時，逐一查看每位技師，找到至少一台他使用的相機。檢查 B 時，先選一台相機，再確認所有技師都使用那一台。如果 B 成立，A 就成立：對每位技師都選那台共用相機即可。反向推論不成立，因為每個人都可能有相機，卻沒有一台相機是所有人共用的。

```
technicians = ("Ada", "Bo", "Cy")
cameras = ("A", "B")
worlds = {
    "distributed": {("Ada", "A"), ("Bo", "B"), ("Cy", "A")},
    "shared": {("Ada", "A"), ("Bo", "A"), ("Cy", "A")},
}
for name, uses in worlds.items():
    each_has_some = all(any((t, c) in uses for c in cameras)
                        for t in technicians)
    some_for_everyone = any(all((t, c) in uses for t in technicians)
                            for c in cameras)
    print(name, each_has_some, some_for_everyone)
assert not any(all((t, c) in worlds["distributed"] for t in technicians)
               for c in cameras)
```

輸出為 `distributed True False` 與 `shared True True`。圖 3 每列表示一個人的選項。第一種解讀要求每列至少有一格被標記；第二種要求至少有一整欄都被標記。因此，作用域改變的是限制條件，不只是把相同要求改寫成另一句話。

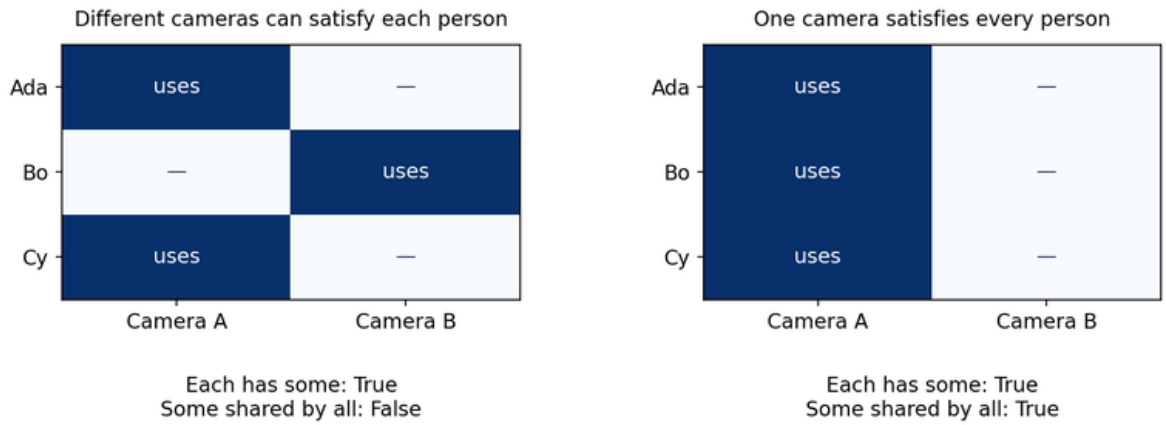


圖 3 — 改變量詞順序，會改變哪些情境滿足句子。

否定與作用域也必須分清楚

“Not every technician approved” 容許有人批准，但至少有一人沒有批准。“Every technician did not approve” 若採取否定在全稱量詞內部的解讀，則表示沒有人批准。書寫英文自身也可能留下作用域歧義；明確改寫能消除這種不確定性。

中文「不是每位技師都批准 / 每位技師都不批准」較容易展示這個區別。前者否定「所有人都批准」，後者對每位技師指定不批准。對非空的技師集合而言：

$$\neg\forall t \in T\ P(t) \ \equiv\ \exists t \in T\ \neg P(t), \qquad \forall t \in T\ \neg P(t) \ \equiv\ \neg\exists t \in T\ P(t).$$

其中 T 是技師集合，$P(t)$ 表示 t 批准，$\neg$ 是否定，$\forall$ 與 $\exists$ 分別是全稱與存在量詞，$\equiv$ 表示邏輯等價。第一個條件只需一個反例，就能推翻全體批准；第二個條件則排除所有批准。若一位技師批准、另一位沒有批准，就只有第一個條件為真。

NLP 12 會深入述詞與論元結構及語意角色標註，NLP 11 則測試句子之間的推論。在設計 `approved` 這類欄位之前，先確定它代表誰的批准、哪個事件，以及哪個帶有量詞的主張。

7. 語用：在情境中解讀

白話版。說話者的意思可能超出句子的字面內容。語用學研究情境如何補上這段落差。

說話者經常傳達超過句子字面斷言的內容。**語用學（pragmatics）**研究語境中的語言使用，包括指稱、背景假設與溝通行動 [9, 10]。語境提供證據，卻不代表可以任意解讀。

指稱與說話者的視角

“I will check that here tomorrow” 的結構很簡單，離開情境後卻缺少許多資訊。“I” 指說話者，“that” 選擇指稱對象，“here” 使用空間視角，“tomorrow” 使用時間視角。這些依賴語境的表達式涉及**指示（deixis）**[10]。轉寄的訊息不應把原說話者的時間與地點，默默換成接收者的時間與地點。

指稱也依賴篇章。“Mei put a camera beside a lamp. It was overheating.” 若沒有更多證據，“it” 可能指任一物件。知道相機與燈都可能變熱，仍然無法消除歧義。在 “Mei turned off the lamp. It stopped flickering” 裡，所描述的事件提供了支持「燈」的證據。證據可以支持某個候選，但不會把文法上的可能性變成普遍規則。

實際分析可以記錄表達式、候選指稱對象，以及支持它們的語境。若輸入不能確定指稱，保留未解決狀態也是有意義的結果。跨句子的共指解析在 NLP 12 中發展，跨對話輪次追蹤實體則在 NLP 08。

斷言、預設與會話含意

斷言（assertion）提出當下討論的主張。**預設（presupposition）**把命題當成背景，而不是主要更新。**會話含意（conversational implicature）**則是根據發話內容及其使用方式推得的額外意義，往往可以取消而不否定字面斷言 [9, 10]。三者不應被壓成同一個「意義」標籤。

“Mei stopped checking the sensor” 斷言某個活動停止，並通常把 Mei 先前檢查感測器當成背景。“Mei did not stop checking the sensor” 也通常把先前檢查視為已知。這種在否定下仍持續的貢獻，有助於辨認背景意義。它們是一般解讀，不是先前事件真的發生的證明。聽者可以質疑：「她根本從來沒有檢查過。」

“Some panels passed” 斷言相關集合裡至少有一個面板通過。在詢問整批是否都通過的對話裡，聽者可能推論「並非全部」。但是 “Some panels passed; in fact, all of them did” 可以取消這個推論，而不否定第一個斷言 [9]。相對地，若面板集合、測試與時間相同，“All panels passed; in fact, one failed” 就互相衝突。

這個差異不需要語言模型也能檢查。想像三個面板：一個、兩個或三個通過，都滿足字面的 “some”。“Some but not all” 的解讀則只保留前兩種可能。會話上的預期讓推論縮小了可能情境；這種縮小不是存在意義本身提供的。它是否合理，取決於問題、說話者的知識與報告方式。

文法上的問句可以執行請求

“Can you lower the gain?” 在形式上詢問能力。在錄音過程中，它可能是在請求調整設定；在測驗中，則可能詢問受訓者是否知道如何操作 [10]。**言語行為（speech act）**是透過發話執行的溝通行動，例如請求、承諾、斷言或提問。

“The gain is too high” 同樣可能是報告，也可能是間接請求。實際詞句本身不會指定所有人際條件。誰控制設備？前一個問題是什麼？說話者是在診斷錄音，還是在協調調整？這些細節為行動解讀提供依據。

有用的標註會分開記錄句子形式、斷言內容與意圖行動。否則，所有問句形式的請求都可能被標成尋求資訊，或所有負面描述都可能被標成指令。目標是對這次互動提出有依據的解讀；缺少情

境時，也要保留其他選項。

8. 篇章：跨句子與輪次的意義

白話版。文本或對話中的句子彼此依靠。後面的句子會回指、對比、解釋或修正前文，讀者則持續追蹤這個共同狀態。

發話可以引入實體、修正先前主張、提供理由，或對比預期。**篇章（discourse）**分析研究句子與對話輪次之間的連結 [2, 10]。逐句獨立閱讀，會丟掉讓後續表達式可被理解的共同狀態。

“Mei told Lin that she would calibrate the camera” 可以把任一人當成未來校正者。中文「美玲告訴怡君，她會校正相機」在這個虛構情境中也有相似歧義。前文若說「怡君已經自願幫忙」，就支持怡君；若說「美玲是唯一有空的技師」，就支持美玲。文法性別或提及順序，單獨都無法決定兩種語境。

比較 “The sensor became hot. Its readings stayed stable” 與 “The sensor became hot, but its readings stayed stable”。後者明確標記：結果違反熱可能干擾讀值的預期。關於穩定的斷言可以相同，篇章貢獻卻不同。相對地，“The readings drifted because the sensor became hot” 提出因果解釋，不只是依序出現兩件事。

對話也包含修復。“Move it to Friday.” “The inspection?” “Yes, the inspection.” 中間一輪不是在推進排程日期，而是在確定物件。釐清後，後續指稱可以使用共同確認的對象。若對話系統只摘要第一句與最後一句的詞，就會漏掉澄清如何改變對話狀態。

這提示一種實用表示：分開記錄已引入的實體、斷言事件、未解決指稱與對話問題。表示可以保留不完整。一句輸入可能沒有說明說話者；完整交換可能確定說話者，卻仍未指出確切相機。增加語境長度，只有在相關證據確實存在時才有幫助。

9. 語料庫、標註與 Zipf 定律

白話版。 NLP 從真實文字集合中學習，這些文字常附有人工標籤。文字如何蒐集、標籤是否一致，限制了任何模型能學到的內容；而且每個語料庫都有一條由罕見詞組成的長尾。

語料庫（corpus）是依明確決定蒐集的文字或語音轉寫：哪些來源、哪段時期、哪些說話者、哪些語言變體。**標註語料庫（annotated corpus）**再依書面準則加上分析，例如斷詞、標記、樹、實體範圍或情感標籤。兩個廣泛使用的例子顯示其範圍：Penn Treebank 為英文文字標註詞性與成分樹 [26]；Universal Dependencies 專案以同一套跨語言依存方案，釋出許多語言的樹庫 [24]。前面的斷詞評測語料庫也說明，連詞邊界都附帶一套標準 [27]。

標註準則與一致度

標籤的清楚程度取決於背後的準則。兩位細心的標註者可能因為句子本身有歧義、準則沒有規定，或其中一人出錯而意見不同。在共同樣本上量測一致度，能在把標籤拿去訓練之前區分這些情況。

當某個標籤非常常見時，原始的百分比一致度會高估可靠性，因為標註者即使用猜的也常會一致。Cohen's kappa 修正了這一點 [28]：

$$\kappa = \frac{p_o - p_e}{1 - p_e}, \qquad p_e = \sum_{k \in \mathcal{Y}} \frac{n_{A,k}}{n} \cdot \frac{n_{B,k}}{n}.$$

其中 p_o 是標註者 A 與 B 意見一致的項目比例；p_e 是假設兩人各自依觀察到的比例獨立選標籤時，預期的一致比例；$\mathcal{Y}$ 是標籤集合；$n_{A,k}$ 與 $n_{B,k}$ 分別是兩位標註者使用標籤 k 的次數；n 是項目數。$\kappa = 1$ 代表完全一致，$\kappa = 0$ 代表只達到機率水準的一致；公式要求 $p_e < 1$。

```
from collections import Counter
from sklearn.metrics import cohen_kappa_score

words = "Mei has to repair the camera mount before the inspection".split()
a = "PROPN VERB PART VERB DET NOUN NOUN ADP DET NOUN".split()   # annotator A
b = "PROPN AUX ADP VERB DET NOUN NOUN ADP DET NOUN".split()     # annotator B
n = len(words)
p_o = sum(x == y for x, y in zip(a, b)) / n                     # observed agreement
ca, cb = Counter(a), Counter(b)
p_e = sum(ca[k] * cb[k] for k in ca) / n**2                     # agreement expected by
 ↪  chance
kappa = (p_o - p_e) / (1 - p_e)
print(f"p_o={p_o:.3f}  p_e={p_e:.3f}  kappa={kappa:.3f}")
assert abs(kappa - cohen_kappa_score(a, b)) < 1e-12
print("disagreements:", [(w, x, y) for w, x, y in zip(words, a, b) if x != y])
```

程式印出 `p_o=0.800  p_e=0.180  kappa=0.756`，兩處分歧是 `has`（`VERB` 對 `AUX`）與 `to`（`PART` 對 `ADP`）。兩者都不是粗心失誤：“has to” 位於主要動詞與助動詞的邊界，不定詞的 “to” 也很像介詞。有用的結果是針對這個構式做出準則決定，而不是更高的數字。Kappa 總結可靠性；分歧清單則指出該修正什麼。

Zipf 定律：少數詞非常常見，多數詞很罕見

把語料庫裡每個詞都計數，再依頻率排序型別。少數功能詞如 “the” 與 “of” 占了所有 token 的一大部分，而大量型別只出現一兩次。**Zipf 定律**把詞的頻率排名與次數之間的關係描述為近似的冪律 [29]：

$$f(r) \approx \frac{C}{r^s}, \qquad \log f(r) \approx \log C - s \log r.$$

其中 $r = 1, 2, \ldots$ 是型別依次數遞減排序後的名次，$f(r)$ 是名次 r 的型別出現次數，$C > 0$ 是尺度常數，$s > 0$ 是指數。當 $s \approx 1$ 時，乘積 $r \cdot f(r)$ 大致固定。在雙對數座標上，這個關係是一條斜率為 $-s$ 的直線。Piantadosi 的回顧指出，這個定律在許多語言與語料庫中近似成立，在兩端有系統性的偏離，而其成因仍有爭論 [29]。

我們可以用已經在磁碟上的真實英文文字檢查它：專案環境（Python 3.9）所安裝的 Python 標準函式庫說明字串（docstring）。

```
import ast, re, sysconfig
from collections import Counter
from pathlib import Path

lib = Path(sysconfig.get_paths()["stdlib"])        # text already on disk: stdlib docstrings
docs = []
for path in sorted(lib.glob("*.py")):
    try:
        tree = ast.parse(path.read_text(encoding="utf-8"))
    except (SyntaxError, UnicodeDecodeError):
        continue
    docs += [ast.get_docstring(n) or "" for n in ast.walk(tree)
             if isinstance(n, (ast.Module, ast.FunctionDef, ast.AsyncFunctionDef,
              ↪  ast.ClassDef))]
tokens = [w for d in docs for w in re.findall(r"[a-z]+", d.lower())]
counts = Counter(tokens)
print(len(tokens), "tokens,", len(counts), "types")
ranked = counts.most_common()
for r in (1, 2, 10, 100, 1000):
    word, f = ranked[r - 1]
    print(f"rank {r:4d}  {word:10s} f={f:5d}  r*f={r * f}")
hapax = sum(f == 1 for f in counts.values())
print(f"types seen once: {hapax} ({hapax / len(counts):.0%})")
```

在 Python 3.9.7 下，程式找到 134,620 個 token 與 6,281 個型別。乘積 $r \cdot f(r)$ 在名次 1（“the”）是 8,313，名次 10（“be”）是 13,640，名次 100（“called”）是 20,100，名次 1,000 是 18,000：並不固定，但名次變動 1,000 倍時，乘積只在約 2.5 倍範圍內變動。圖表腳本在名次 10 到 1,000 之間做最小平方擬合，得到 $s \approx 1.00$。所有型別中有三分之一，也就是 2,003 個（32%），恰好只出現一次。不同 Python 版本附帶的說明字串不同，確切計數可能改變。

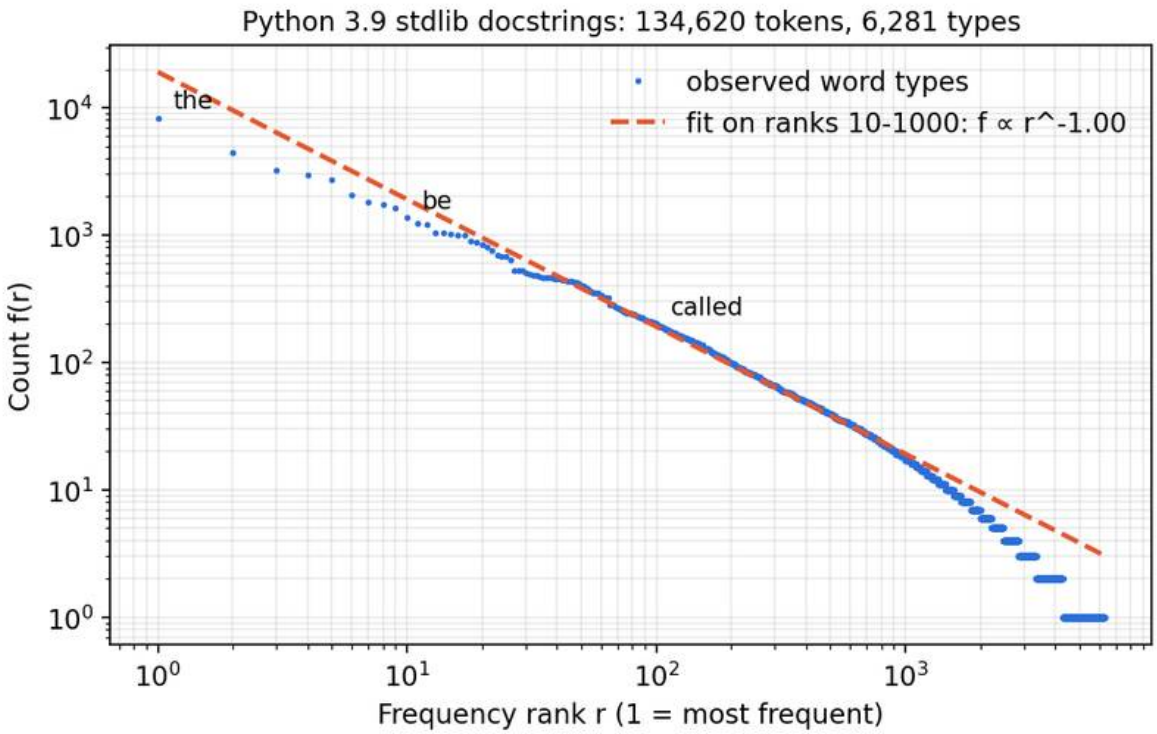


圖 *4 — Python 3.9* 標準函式庫說明字串的名次與頻率圖。虛線是在名次 *10–1,000* 上擬合的冪律；頭部與只出現一次的長尾都偏離這條線。

這對 NLP 有三個後果。第一，任何固定的詞彙表都會漏掉一部分長尾，因此新文字總會帶來沒見過的詞；這是子詞詞彙表存在的原因。第二，以計數估計的機率對常見詞可靠，對占多數的罕見詞則不可靠，這就是 n-gram 模型需要平滑化的原因（NLP 05）。第三，以所有 token 計算的準確率主要由常見詞決定；若罕見名稱或技術詞彙很重要，評估就必須另外量測它們。

10. 語言多樣性與模型假設

白話版。不同語言用不同方式包裝資訊，不同社群也使用不同的語言變體。圍繞某一語言習慣建立的模型，可能錯過另一種語言所表達的內容。

不同語言與社群會把資訊分配到構詞、詞序、韻律與語境的不同位置。**語言類型學（typology）**比較這些結構模式。**方言（dialect）**涉及社群的語言變體；**語域（register）**涉及特定情境或活動的使用方式 [3, 11]。這些區別描述變異，不是替說話方式排序。

翻譯不會保留每種結構

比較 “I calibrated this camera yesterday” 與「這台相機，我昨天校正過」。中文例子先把相機設為主題，再對它作出評論。把第一個名詞片語直接當成行動者，就會誤認校正者。英文翻譯可以保留事件，卻改變訊息如何組織已知與新資訊。

當相機與負責者已經確定時，短句「已經校正好了」可以提供充分資訊。逐字翻譯若要求每個英文論元都有明說的對應項目，可能把它看成不完整。在語境中，它卻能直接回答待解問題。這不表示省略的參與者一定能被找回；分析仍須查看實際篇章。

同樣地，「三台相機」與 “three cameras” 用不同方式包裝數量。不能只因為「複數詞尾」適合某一語言，就把這個特徵設為普遍要求。結構類別可以用於比較，同時需要各語言的實現與標註決定。強制詞尾較少的語言，仍然可以透過構式、詞與語境表達細緻區別。

變體、情境與混合語言表達

正式維修報告與技師聊天，可能用不同詞彙、縮寫與明確程度描述相同設備。報告的 “The shutter failed to actuate” 與口語的 “The shutter didn’t fire”，在語境中可能描述相關事件，但措辭提供不同的情境與讀者證據。不熟悉的說法不會自動成為錯誤。

說話者可能在雙語討論裡寫「把 exposure 調低一點」。英文項目出現在中文構式的一個位置。**語碼轉換（code-switching）**涉及互動中不同語言變體的交替 [14]；分析本例需要同時看詞彙選擇與周圍文法。只憑一次出現，未必能確定社群把該項目當成語碼轉換，還是已經成立的借詞。

分析應先保留觀察到的表達，再進行標準化。把地區或技術表達換成預設標準，可能有助於某個搜尋任務，卻抹掉另一任務需要的區別。語料庫記錄特定說話者、情境與蒐集決定下的語言；常見語料模式是使用證據，不是所有說話者可說內容的普遍清單。

11. 一次完整的閱讀分析

用一則虛構訊息，把不同層次放在一起：

> 小林說：「那台相機又過熱了，你明天能幫我看看嗎？」
>
> Lin said: “That camera overheated again. Can you take a look for me tomorrow?”

翻譯有助於英文讀者，但已經替「看看」與引述互動的標點選擇措辭。在把翻譯當成等價證據之前，先分析原文。

層次	訊息提供的內容	仍需確定的事項
書寫形式與斷詞	引號把小林的發話分成一組；相機是物件表達式	語音的語調與手勢不存在於文字中
構詞與構式	看看在此請求裡是重疊形式，不是計數兩次檢查	確切活動與預期投入取決於情境
句法	小林引入被報導的說話者；你是引號內的受話者	報導者不一定是受話者
語意	相機被描述為過熱；句子表達未來協助的問題	哪台相機，以及描述是否符合事實
背景與指稱	又通常援引先前發生；那台選擇語境中可辨認的相機	先前過熱與指示詞的對象需要證據
語用與篇章	在維修對話裡，問句自然支持請求解讀	該解讀是否符合這次互動，以及請求是否被接受

引述改變參照框架。「我」指引號內的說話者小林。「你」指小林的受話者，可能與目前讀者不同。「明天」相對於引號內發話的日期。如果報導較晚才送達，送達日期不會重新定義請求的日期。

在這個構式裡，「看看」的一般解讀呈現「看一下」的活動，往往帶有較輕或試探性的語氣 [13]。它不會確定精確檢查時間或數值上的重複次數。若翻成 “inspect twice”，就是把可見的形式重複誤當成事件計數。

最後，把語言證據與事實證據分開。訊息提供小林曾報告過熱與請求協助的證據；它本身不會驗證過熱、指出序號，或證明有人同意幫忙。結構化輸出可以記錄說話者與被報導事件，同時保留未解決指稱與日期。這比只給出「相機維修」之類的主題標籤，更能提供有用分析。

現代觀點

近期綜述說明，對大型語言模型而言，語言分析仍然必要。Ma 等人的 2025 年綜述，依指稱與指示、會話含意與預設、言語行為、篇章、社會語用等現象，組織語用評估 [10]。它比較任務設計、資料蒐集與評估，指出現有測試與語境中語言使用的落差。對本教學最有用的啟示是：評估任務真正要求的推論，不要把概括的語言分數當成所有語用能力的證明。

Mielke 等人的斷詞綜述，從詞、字元與中間單位整理開放詞彙問題 [22]。有用的問題是切分如何平衡覆蓋率、序列長度與重用。這些量彼此不同：斷詞器可以表示每個輸入字串，模型卻仍可能缺乏學習相關組合的訓練證據。Zipf 定律說明了這個問題為何永遠不會消失：無論詞彙表多大，長尾都會持續帶來罕見形式 [29]。

Qin 等人的 2025 年綜述，區分透過參數調整的多語言對齊，以及固定參數下的對齊，包括提示與翻譯式方法 [12]。它整理跨語言訓練、評估與挑戰。這個區分指出系統在哪裡調整行為，但能產生多語言輸出，不代表它保留構式的作用域、語域或會話功能。

Winata 等人的 2023 年系統性綜述，依語言、任務與研究趨勢，組織語碼轉換研究 [15]。可以搭配 Doğruöz 等人對結構與社會模式的分析 [14]：測試由兩個單語來源拼接的文字，與解讀雙語說話者如何在互動中實際切換，是不同任務。第 10 節 exposure 的例子把這點具體呈現出來：除了辨認英文項目，周圍構式與對話情境也提供資訊。

語言學分析也持續細化看似簡單的構式。Lu 與 Müller 的 2026 年研究，分析華語動詞重疊及其與動詞類別、時貌的互動 [13]。本教學相關的觀察是：有限活動的意義，以及依賴語境的語用效果，值得分開分析。看得到的形式重複，不是要求把事件計數兩次的普遍規則。

具體評估可以固定其他條件，只改變一項特性。交換 Ada 與 Bo，測試角色方向；保留詞彙、改變相機的附著；比較「不是全部」與「沒有任何一個」；把引述訊息移到較晚的報導語境。先寫出預期解讀與可接受選項，再觀察失敗在哪個區別上。這些是小型診斷測試，不表示通過後就具備一般語言能力。

開放問題包括如何取得相關語境、區分真正歧義與可避免的標註歧見、評估自然的混合語言對話，以及跨社群測量解讀，而不只使用翻譯提示。有些輸入本來就容許多種讀法。說明選項與缺少證據的系統，可能比自信選定單一答案的系統更忠於語言。

重點整理

- 口語、手語與書寫揭露不同資訊；逐字稿只是部分紀錄。
- 計算用 token 不必與詞、詞素、詞位或語音單位一致；BPE、WordPiece 或 unigram 模型學到的子詞片段反映頻率，而不是構詞。
- 詞幹擷取依規則切掉詞尾；詞形還原對應到字典形式，並需要詞性。兩者都會丟掉某些任務需要的資訊。
- 構詞結構與中文斷詞編碼了分析選擇，也可能有替代方案，而不同語料庫依循不同標準。
- 詞頻會丟掉順序；詞性類別、分組與文法關係提供額外意義。成分結構顯示分組，依存關係顯示中心語。
- 作用域決定真值條件：「每個人都有某個」不等於「某一個被所有人共用」。
- 斷言、背景假設、可取消的推論、指稱、言語行為與篇章關係，需要說話者的情境及對話歷史。
- 語料庫帶有蒐集選擇與標註準則；要檢查一致度，並預期罕見詞形成 Zipf 式長尾。

練習

1. 在 “The technician repaired two cameras” 裡，辨認詞形、詞位、屈折對比、一組 UPOS 標記序列，以及一種計算用 token 切分。解釋每種表示保留什麼。

   提示

   比較 “repaired” 與 REPAIR。Tokenizer 可以保留整個詞形或拆開；兩種選擇都不能單獨確定時態解讀，UD 則把時態記錄為特徵。

2. 替「學生會說話」的兩種解讀各加一句語境。解釋為何只列舉詞庫路徑，無法偏好任一語境。

   提示

   一種情境可以討論學生的能力；另一種可以討論學生會宣布立場。

3. 改變 BPE 語料中一個訓練詞的頻率。檢查第一個合併，再編碼 `lowest`。為什麼兩種結果都不必維持不變？

   提示

   符號對計數以詞的出現次數加權。前面的合併會改變後面可使用的符號對。

4. 替 “The technician inspected the panel with the camera” 寫出兩種語境，各支持一種附著。辨認修飾語連到哪個中心語，以及事件描述如何改變。

   提示

   對比檢查方法，以及在多個面板中辨認裝有相機的面板。

5. 修改量詞程式，加入一位沒有使用任何相機的技師。執行前先預測兩種解讀，再解釋 *B* 為何蘊涵 *A*。

   提示

   若任一技師沒有可用配對，兩個條件都失敗。一台共用相機能替每位技師提供見證。

6. 分別分析 “Mei stopped checking the sensor” 與 “Some panels passed” 在否定與明確取消下的表現。區分主要斷言、一般背景假設，以及會話推論。

   提示

   試著用 “Mei never checked it” 挑戰背景，用 “in fact, all passed” 取消推論。維持相同面板集合與測試。

參考文獻

1. Masato Hagiwara, *Real-World Natural Language Processing: Practical Applications with Deep Learning*, Manning, 2021. 出版社
2. Susan McRoy, *Principles of Natural Language Processing*, University of Wisconsin–Milwaukee open textbook, 2021. Grammars and Syntactic Processing
3. Catherine Anderson, Bronwyn Bjorkman, Derek Denis, Julianne Doner, Margaret Grant, Nathan Sanders and Ai Taniguchi, *Essentials of Linguistics*, 2nd ed., eCampusOntario, 2022. 開放教科書
4. Catherine Anderson et al., “3.1 Modality,” *Essentials of Linguistics*, 2nd ed., eCampusOntario, 2022. 章節
5. Catherine Anderson et al., “4.1 Phonemes and allophones,” *Essentials of Linguistics*, 2nd ed., eCampusOntario, 2022. 章節
6. Catherine Anderson et al., “3.12 Tone and intonation,” *Essentials of Linguistics*, 2nd ed., eCampusOntario, 2022. 章節
7. Catherine Anderson et al., “5.1 What is morphology?,” *Essentials of Linguistics*, 2nd ed., eCampusOntario, 2022. 章節
8. Catherine Anderson et al., “6.4 Identifying phrases: Constituency tests,” *Essentials of Linguistics*, 2nd ed., eCampusOntario, 2022. 章節
9. Catherine Anderson et al., “8.4 Conversational implicatures,” *Essentials of Linguistics*, 2nd ed., eCampusOntario, 2022. 章節
10. Bolei Ma, Yuting Li, Wei Zhou, Ziwei Gong, Yang Janet Liu, Katja Jasinskaja, Annemarie Friedrich, Julia Hirschberg, Frauke Kreuter and Barbara Plank, “Pragmatics in the Era of Large Language Models: A Survey on Datasets, Evaluation, Opportunities and Challenges,” *ACL*, pp. 8679–8696, 2025. 論文
11. Catherine Anderson et al., “10.2 Language varies,” *Essentials of Linguistics*, 2nd ed., eCampusOntario, 2022. 章節

12. Libo Qin, Qiguang Chen, Yuhang Zhou, Zhi Chen, Yinghui Li, Lizi Liao, Min Li, Wanxiang Che and Philip S. Yu, "A survey of multilingual large language models," *Patterns*, 6(1), 101118, 2025. 論文
13. Yanru Lu and Stefan Müller, "Deliminative verbal reduplication in Mandarin Chinese," *Journal of Linguistics*, 62(3), pp. 595–632, 2026. 論文
14. A. Seza Doğruöz, Sunayana Sitaram, Barbara E. Bullock and Almeida Jacqueline Toribio, "A Survey of Code-switching: Linguistic and Social Perspectives for Language Technologies," *ACL-IJCNLP*, pp. 1654–1666, 2021. 論文
15. Genta Winata, Alham Fikri Aji, Zheng Xin Yong and Thamar Solorio, "The Decades Progress on Code-Switching Research in NLP: A Systematic Survey on Trends and Challenges," *Findings of ACL*, pp. 2936–2978, 2023. 論文
16. M. F. Porter, "An algorithm for suffix stripping," *Program*, 14(3), pp. 130–137, 1980. DOI
17. Rico Sennrich, Barry Haddow and Alexandra Birch, "Neural Machine Translation of Rare Words with Subword Units," *ACL*, 2016. 論文
18. Mike Schuster and Kaisuke Nakajima, "Japanese and Korean Voice Search," *ICASSP*, pp. 5149–5152, 2012. DOI
19. Yonghui Wu et al., "Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation," arXiv:1609.08144, 2016. 論文
20. Taku Kudo, "Subword Regularization: Improving Neural Network Translation Models with Multiple Subword Candidates," *ACL*, 2018. 論文
21. Taku Kudo and John Richardson, "SentencePiece: A Simple and Language Independent Subword Tokenizer and Detokenizer for Neural Text Processing," *EMNLP System Demonstrations*, 2018. 論文
22. Sabrina J. Mielke et al., "Between Words and Characters: A Brief History of Open-Vocabulary Modeling and Tokenization in NLP," arXiv:2112.10508, 2021. 綜述
23. Hugging Face, "The Tokenization Pipeline," Tokenizers documentation, current online edition. 文件
24. Marie-Catherine de Marneffe, Christopher D. Manning, Joakim Nivre and Daniel Zeman, "Universal Dependencies," *Computational Linguistics*, 47(2), 2021. 論文
25. Universal Dependencies contributors, "Universal POS tags," Universal Dependencies documentation, current online edition. 文件
26. Mitchell P. Marcus, Beatrice Santorini and Mary Ann Marcinkiewicz, "Building a Large Annotated Corpus of English: The Penn Treebank," *Computational Linguistics*, 19(2), 1993. 論文
27. Thomas Emerson, "The Second International Chinese Word Segmentation Bakeoff," *Proceedings of the Fourth SIGHAN Workshop on Chinese Language Processing*, 2005. 論文
28. Jacob Cohen, "A Coefficient of Agreement for Nominal Scales," *Educational and Psychological Measurement*, 20(1), pp. 37–46, 1960. DOI
29. Steven T. Piantadosi, "Zipf's word frequency law in natural language: A critical review and future directions," *Psychonomic Bulletin & Review*, 21(5), pp. 1112–1130, 2014. DOI
30. Jacob Devlin, Ming-Wei Chang, Kenton Lee and Kristina Toutanova, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," *NAACL-HLT*, 2019. 論文

NLP 03・自然語言處理的類神經網路

從計數特徵與邏輯斯分類，走到嵌入層、神經機率語言模型與一維卷積文字分類器，並涵蓋填補、遮罩、正則化與完整的 PyTorch 實驗。

網站版本: https://blog.shyandram.dev/zh-tw/learn/nlp/neural-networks-for-nlp/

你將學到

- 以明確的標註規則定義文字分類任務，並推導計數特徵、TF–IDF、邏輯斯分類與交叉熵。
- 用嵌入層（embedding layer）把 token ID 轉成學習得到的向量，加以池化，再用小型多層感知器分類。
- 理解 Bengio 等人的神經機率語言模型（neural probabilistic language model），並說明共享詞向量為何讓它能推廣到未見過的歷史。
- 寫出一維卷積文字分類器與時間軸最大池化（max-over-time pooling）的數學，並以正確的填補與遮罩實作。
- 對小型 PyTorch 模型做正則化、訓練與提前停止，判讀訓練曲線，並依類別分析錯誤。
- 分開評估決策、機率品質、涵蓋率與類別不平衡。

先看全貌

文字分類器把訊息轉成允許的決策：分流申訴、辨識語言、標記評論，或把文件交給人員檢查。最重要的選擇往往發生在挑選神經網路架構之前。什麼算一個樣本？可以查看哪些脈絡？標籤代表什麼？當所有標籤都不適用時，系統應如何處理？

Hagiwara 在第 2–4 章與第 7 章發展第一個分類應用、詞與文件嵌入、句子分類及卷積編碼器 [1]。McRoy 第 6 章則把情緒、相似度、合文法性與蘊涵分成不同語言任務 [2]。本課結合這兩個方向：建立應用，同時精確定義預測目標。情緒標籤不是蘊涵判斷，相似度分數也不表示兩個陳述具有相同的真值條件。

> **用最簡單的話說**
>
> 想像把信件放進三個托盤：付款、配送與技術協助。計算詞語就像找信封上熟悉的線索。類神經網路則自己學線索：它給每個詞一小串數字，學會哪些短片語重要，再把證據加總。不論哪一種方法，可靠的分類員都需要第四個動作：當信件不明確屬於任何托盤時，詢問其他人。

本課分成兩半。前半建立最強的簡單基準：稀疏特徵加線性分類器，並謹慎地訓練與評估。後半以學習得到的特徵取代人工設計的特徵：**嵌入層**、池化後的多層感知器、神經機率語言模型與卷積編碼器。這裡每個神經模型仍以和線性基準相同的交叉熵損失訓練，因此改變的是表示，而不是目標函數。反向傳播、啟動函數與最佳化器在 ML 02 中說明；本頁直接使用它們，專注於文字特有的部分。

所有範例都使用原創訊息，不需要下載資料集。量測的分數只描述這些合成訊息，不能證明系統在真實應用中的表現。

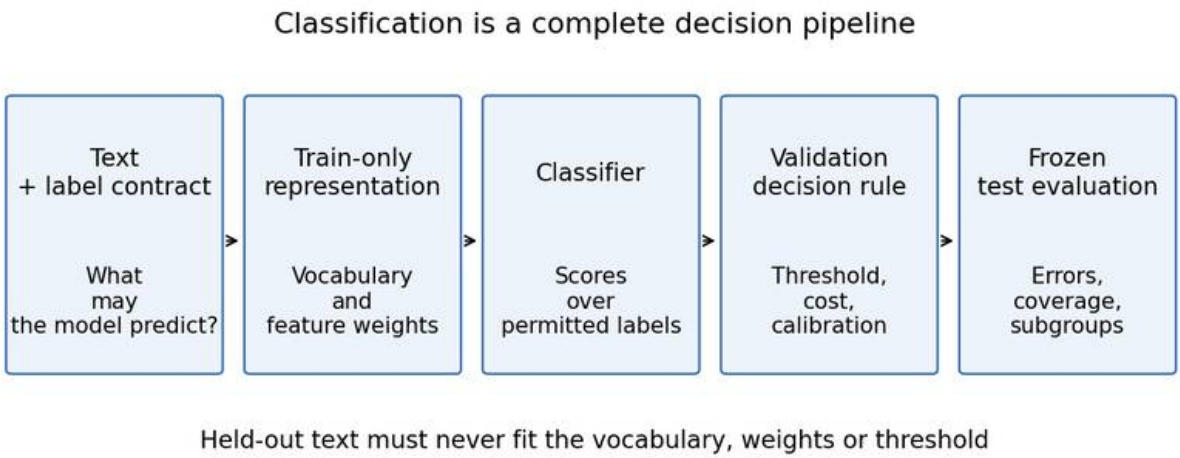


圖 *1* — 分類包含表示、預測與行動規則。保留最終測試集的要求，適用於每個會從資料學習的階段，包括神經特徵學習。

定義預測約定

白話版。標籤是對一段文字在這個應用中代表什麼的共同約定。先建立約定，再蒐集樣本或最佳化損失。

單標籤分類（single-label classification）讓一個輸入從固定集合得到恰好一個標籤。多標籤分類（multi-label classification）允許多個標籤同時成立。一則同時提到包裹延遲與收費錯誤的申訴，可能需要兩條處理路徑、主要路徑加次要標記，或由人員分流。即使輸入文字相同，這些仍是不同任務。

先用一般語言寫出約定。在本課教學應用中，billing 涉及付款或發票，delivery 涉及包裹或配送，technical 涉及帳號登入或軟體行為。分類後允許保留決策。因此，模稜兩可的多主題訊息，不應只因數值模型能計算 argmax，就被默默強迫送往一個類別。

$$f_\theta : \mathcal{X} \to \Delta^{K-1}, \qquad \hat{y}(x) = \operatorname*{argmax}_{k \in \{1,\dots,K\}} p_\theta(y = k \mid x).$$

其中 f_θ 是參數為 θ 的分類器，$\mathcal{X}$ 是允許的文字輸入集合，Δ^{K-1} 是 K 個類別機率向量組成的單純形，x 是一個輸入，y 是標籤，k 索引類別，p_θ 是模型的條件分布，$\hat{y}$ 是機率最高的類別。單純形中的各元素必須非負，且總和為一。

這個式子描述模型輸出，而不是完整的產品行動。服務可以只在信心超過驗證資料所選門檻時回傳路徑，也可以拒絕空白訊息、要求更多脈絡，或把多主題訊息交給人員檢查。評估必須呈現這些行動。

監督單位應與部署情境一致。只用獨立意見句訓練的句子情緒模型，可能無法處理包含引述、修正與多位說話者的完整對話。如果部署可以查看先前回合，標註與訓練也應包含相同的允許脈絡。反過來，標註者如果看得到最後處理結果，但模型只看得到第一則訊息，就可能產生模型根本無法判定的目標。

計數特徵與詞彙表

白話版。最簡單且實用的表示，是記錄出現了哪些詞或短序列。它便宜、容易檢查，也常是很好的第一個比較基準。

只從訓練文字選擇詞彙表。詞彙項目可以是詞、字元序列或相鄰詞對。使用詞彙表時，每個座標計算一個詞彙項目的出現次數。由於大多數可能的詞不會出現在同一份文件中，文件向量通常很稀疏。

$$x_j(d) = c(v_j, d), \qquad \mathbf{x}(d) = (x_1(d), \dots, x_V(d))^\top.$$

其中 d 是文件，v_j 是座標 j 對應的詞彙項目，$c(v_j, d)$ 是它在文件 d 的出現次數，$x_j(d)$ 是該特徵值，$\mathbf{x}(d)$ 是得到的行向量，V 是詞彙表大小。轉置符號 $\top$ 表示列出的座標以直向排列。

詞袋表示（bag of words）忽略順序。「Payment arrived after the refund」與「Refund arrived after the payment」包含相同的詞，卻描述不同事件順序。詞 bigram 能恢復部分局部資訊；字元特徵可以擷取拼字片段與詞形變化，但兩者都不會自動理解長距離依存或否定的作用範圍。

特徵選擇包含對任務的假設。小寫化能減少稀疏性，卻也會合併專有名稱與一般名詞。移除所有停用詞可能刪掉「not」，而它對許多分類很重要。語言辨識器可能更適合使用字元模式，而非為某一種語言設計的詞彙表。每個正規化規則都應有任務上的理由；NLP 02 討論這些選擇背後的語言單位。

在每次實驗中，詞彙表建立、文件頻率門檻與特徵選擇都必須放在訓練部分內。先查看測試文件，再決定保留哪些罕見詞，會讓訓練流程得到評估分布的資訊。即使沒有查看測試標籤，也可能形成不容易察覺的資料洩漏。

記錄詞彙表大小、完全沒有辨識特徵的訊息比例、token 數，以及被刪除詞語的代表範例。這些診斷能揭露單一準確率看不到的變化。新產品代碼或新語言，可能讓大量輸入幾乎變成空向量，卻不會產生明顯的例外錯誤。

TF–IDF 與向量正規化

白話版。幾乎出現在每則訊息中的詞，通常較難區分訊息。TF–IDF 把文件內的出現次數與訓練文件間的稀有程度結合。

本課實作使用的常見平滑定義為：

$$\operatorname{idf}_j = \log \frac{1 + N}{1 + \operatorname{df}_j} + 1, \qquad z_j(d) = c(v_j, d)\, \operatorname{idf}_j, \qquad \tilde{\mathbf{z}}(d) = \frac{\mathbf{z}(d)}{\|\mathbf{z}(d)\|_2}.$$

其中 idf_j 是詞彙項目 v_j 的逆文件頻率權重，N 是訓練文件數，df_j 計算包含該項目的訓練文件數，$c(v_j, d)$ 是它在文件 d 中的次數，$z_j(d)$ 是加權座標，$\mathbf{z}(d)$ 收集這些座標，$\tilde{\mathbf{z}}(d)$ 是長度為一的版本。

歐幾里得範數 $\|\mathbf{z}(d)\|_2$ 是座標平方總和的平方根。零向量維持零，不會除以零。Scikit-learn 特徵擷取文件說明了這些實作選擇 [3]。

加上的一避免已擬合詞彙表中的權重出現未定義情況，也固定了平滑慣例。不同函式庫可能使用不同定義，因此只說「TF–IDF」不足以重現。應記錄斷詞器、特徵範圍、次數轉換、加權公式與正規化方式。

正規化避免長訊息只因反覆用詞就占有較大影響。如果特徵空間中的方向比總長度重要，這通常有幫助。但長度本身也可能具有預測力，例如錯誤報告可能比重設密碼的請求更長。若長度與任務有關，應加入明確的長度特徵，而不是依賴未正規化計數的偶然效果。

TF–IDF 不會推論同義詞。「Charge」與「payment」仍是不同座標，除非樣本讓分類器學到它們扮演相似角色。它也可能給罕見識別碼很大權重，即使稀有程度與任務無關。一個帳號編號可能因相關紀錄共用範本而看似有分類力。依群組切分及檢查係數，有助找出這類捷徑。

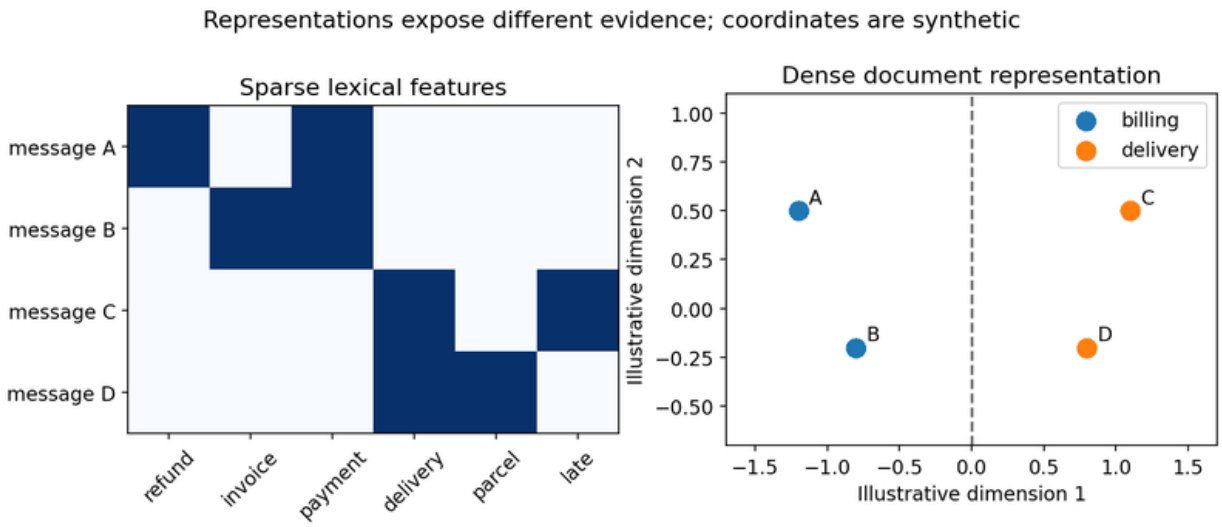


圖 2 — 稀疏特徵保留明確的詞彙證據；稠密表示把模式壓縮到座標中，座標意義取決於訓練目標。

邏輯斯分類與損失

白話版。線性分類器把各特徵的證據相加，為每個標籤產生分數。把分數轉成機率，便得到可微分的訓練目標與決策依據。

多類別模型可以寫成：

$$s_k = \mathbf{w}_k^\top \tilde{\mathbf{z}} + b_k, \qquad p_\theta(y = k \mid d) = \frac{\exp(s_k)}{\sum_{\ell=1}^{K} \exp(s_\ell)}.$$

其中 s_k 是類別 k 的分數，$\mathbf{w}_k$ 是它的學習特徵權重向量，$\tilde{\mathbf{z}}$ 是正規化後的文件表示，b_k 是學習得到的偏置，p_θ 是條件類別機率，θ 包含所有權重與偏置，d 是文件，ℓ 索引競爭類別，K 是類別數，exp 是指數函數。

訓練應提高標註類別的機率。正則化則抑制任意放大的權重，以免模型過度配合偶然出現的詞：

$$\mathcal{L}(\theta) = -\frac{1}{n}\sum_{i=1}^{n} \log p_\theta(y_i \mid d_i) + \frac{\lambda}{2}\sum_{k=1}^{K} \lVert \mathbf{w}_k \rVert_2^2 .$$

其中 $\mathcal{L}$ 是目標函數，n 是訓練樣本數，i 索引樣本，(d_i, y_i) 是已標註文件與標籤，$\lambda \geq 0$ 是正則化強度，$\lVert \mathbf{w}_k \rVert_2^2$ 是類別 k 特徵權重的平方總和。第一項是多類別交叉熵，第二項是平方權重正則化。函式庫可能採用不同縮放；scikit-learn 以參數 C 表示反向的正則化控制 [4]。IT 01 說明為何第一項是交叉熵。

很大的正係數表示該特徵會相對於其他證據提高類別分數，卻不表示該特徵造成真實類別。相關詞可能分攤或交換權重；改變詞彙表，也會改變係數比較的意義。

二元分類的 softmax 可以改寫成分數差的一個 sigmoid。獨立的多標籤預測則應為每個標籤使用一個 sigmoid，而不是強迫標籤競爭的 softmax。輸出啟動函數與損失必須配合標註約定，而不是配合最容易寫出的模型頭。

把這個模型當成網路來看，它就是一個線性層接著 softmax。下面每個神經分類器都保留這個最後一層與它的損失。改變的是向量 $\mathbf{z}$：它不再由固定的計數規則算出，而是由可學習的層計算。

完整的 CPU 分類實驗

白話版。詞彙表、分類器與門檻必須從指定的資料部分學習。使用 pipeline 能更容易維持這個邊界。

下列獨立範例使用原創訊息，明確分開訓練、驗證與測試文字。驗證階段依示範成本選門檻：一次錯誤分流的成本為兩個單位，要求人工檢查的成本為五分之一個單位。這些數值是教學政策參數，不是量測出的商業成本。在查看最終測試結果之前，凍結所有選擇。

```
import numpy as np
from sklearn.pipeline import Pipeline
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import f1_score, log_loss

train = [
    ("please refund my payment", "billing"),
    ("the invoice has a wrong charge", "billing"),
    ("my payment was charged twice", "billing"),
    ("how can i update the billing card", "billing"),
    ("please send the payment receipt", "billing"),
    ("the refund is missing from my invoice", "billing"),
    ("where is my delivery parcel", "delivery"),
    ("my shipment is late", "delivery"),
    ("please change the delivery address", "delivery"),
    ("the parcel arrived damaged", "delivery"),
    ("how do i track the shipment", "delivery"),
    ("the delivery driver could not find the address", "delivery"),
    ("i cannot login to my account", "technical"),
    ("the application crashes on startup", "technical"),
```

```
    ("please reset my account password", "technical"),
    ("the software shows an error", "technical"),
    ("my login fails after the software update", "technical"),
    ("the application cannot connect to my account", "technical"),
]
validation = [
    ("i need a receipt for the charge", "billing"),
    ("the billing payment needs a refund", "billing"),
    ("can i track this delivery", "delivery"),
    ("the shipment went to another address", "delivery"),
    ("my password does not allow login", "technical"),
    ("the software crashes with an error", "technical"),
]
test = [
    ("please explain the invoice payment", "billing"),
    ("a charge appears before the refund", "billing"),
    ("the parcel delivery is not here", "delivery"),
    ("my shipment arrived at the wrong address", "delivery"),
    ("an account error prevents login", "technical"),
    ("the application crashes after a password change", "technical"),
]

def unpack(rows):
    return [text for text, _ in rows], np.array([label for _, label in rows])

train_x, train_y = unpack(train)
val_x, val_y = unpack(validation)
test_x, test_y = unpack(test)
model = Pipeline([
    ("features", TfidfVectorizer(ngram_range=(1, 2))),
    ("classifier", LogisticRegression(C=4.0, max_iter=300, random_state=0)),
])
model.fit(train_x, train_y)
classes = model.named_steps["classifier"].classes_

def decision(probabilities, threshold):
    best = probabilities.argmax(axis=1)
    prediction = classes[best]
    accepted = probabilities[np.arange(len(best)), best] >= threshold
    return prediction, accepted

val_prob = model.predict_proba(val_x)
grid = [0.40, 0.50, 0.60, 0.70, 0.80]
costs = []
for threshold in grid:
    pred, accepted = decision(val_prob, threshold)
    wrong = accepted & (pred != val_y)
    costs.append((2.0 * wrong.sum() + 0.2 * (~accepted).sum()) / len(val_y))
threshold = grid[int(np.argmin(costs))]  # first minimum fixes the tie rule

test_prob = model.predict_proba(test_x)
test_pred, accepted = decision(test_prob, threshold)
print("validation-selected threshold:", threshold)
print("test macro F1:", round(f1_score(test_y, test_pred, average="macro"), 3))
print("test log loss:", round(log_loss(test_y, test_prob, labels=classes), 3))
print("test coverage:", round(accepted.mean(), 3))
```

```
if accepted.any():
    print("accepted accuracy:", round((test_pred[accepted] == test_y[accepted]).mean(), 3))
else:
    print("accepted accuracy: undefined (no accepted messages)")
probe = ["the weather is pleasant", "my delivery charge is wrong"]
for text, probabilities in zip(probe, model.predict_proba(probe)):
    prediction, keep = decision(probabilities[None, :], threshold)
    print(text, "->", prediction[0] if keep[0] else "review")
```

執行結果：

```
validation-selected threshold: 0.4
test macro F1: 1.0
test log loss: 0.413
test coverage: 1.0
accepted accuracy: 1.0
the weather is pleasant -> delivery
my delivery charge is wrong -> delivery
```

六則完全分對的測試訊息遠不足以支持任何關於方法的主張；它們只顯示流程能從頭跑到尾。驗證集讓高於 0.40 的每個門檻都顯得成本較高，因此選中網格中最低的值，而這個政策會接受所有訊息。

不熟悉的天氣訊息才是有用的部分。三個路徑都不正確，模型卻仍回傳機率分布，而且這裡最大的機率（delivery 約 0.51）通過了所選門檻。輸出頭沒有代表任意外部主題的額外類別。保留決策的門檻可能減少部分不適當行動，但最大機率不是有保證的領域外偵測器，在這次執行中它完全沒有攔下這則訊息。混合主題的「delivery charge」訊息同樣被強迫送往單一路徑。

注意，最終 macro F1 量測的是所有測試樣本被強制分類的結果；涵蓋率與接受樣本準確率則描述行動規則。只報接受樣本準確率，可能隱藏拒絕幾乎所有輸入的模型；只報強制分類準確率，可能隱藏有用的選擇性行為。兩種觀點與分母都應清楚呈現。

真實應用應把這些教學清單換成有文件說明與群組識別碼的語料。同一對話的訊息不應隨機分散到訓練與測試中。共享範本、重複簽名與近似描述也需要檢查。本例刻意呈現運作機制；驗證樣本太少，無法穩定估計門檻。

控制條件的否定測試

改變特徵詞彙表，可以保留無序詞頻會遺失的部分資訊。以下固定自行設計的標註語料與分類器種類，比較單一詞元特徵，以及單一詞元加上相鄰詞對的特徵。TF–IDF 與邏輯斯迴歸實作的說明見 [3,4]。

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import make_pipeline
```

```
train = [
    ("the battery is good", 1), ("good screen and good sound", 1),
    ("the camera is great", 1), ("great value , works well", 1),
    ("the battery is not good", 0), ("not great , the screen broke", 0),
    ("the sound is bad", 0), ("bad camera and bad battery", 0),
    ("works well , not bad at all", 1), ("not good , not worth it", 0),
]
test = ["the screen is not good", "the sound is not bad", "great battery"]
texts, labels = zip(*train)

for name, ngrams in [("unigrams", (1, 1)), ("uni+bigrams", (1, 2))]:
    clf = make_pipeline(TfidfVectorizer(ngram_range=ngrams, token_pattern=r"\S+"),
                        LogisticRegression(C=10.0))
    clf.fit(texts, labels)                    # vocabulary + IDF fitted on train only
    probs = clf.predict_proba(test)[:, 1]
    print(f"{name:12s}", [f"{t!r}: {p:.2f}" for t, p in zip(test, probs)])
```

正面評論的預測機率為：

```
unigrams     ["'the screen is not good': 0.43", "'the sound is not bad': 0.11", "'great
 ↪  battery': 0.61"]
uni+bigrams  ["'the screen is not good': 0.20", "'the sound is not bad': 0.18", "'great
 ↪  battery': 0.56"]
```

兩個模型都把 “the screen is not good” 的機率判在二分之一以下；加入詞對後，機率從 0.43 降到 0.20。“the sound is not bad” 的機率則從 0.11 升到 0.18，卻仍低於二分之一。這些觀察確立的是已擬合玩具模型的行為，無法單獨認定某個係數就是原因：加入詞對也改變了詞彙表、正規化與共同擬合的權重。

第二句是有用的錯誤探針：模型已有局部片語特徵，卻仍對預期的正面解讀給出錯誤標籤。語料中只有一個正面 “not bad” 範例，而且同時包含其他正面用語。這麼稀少的證據，無法充分確定究竟是哪個特徵承載標籤。在對架構下普遍結論之前，應檢查各特徵的貢獻，並加入固定主題、只改變否定的控制範例。

片語是否出現，也不同於否定的作用域。相鄰特徵可以區分 “not good” 與 “good”，但辨識 “not every repair succeeded” 與 “every repair failed” 需要另一種組合差異。增加詞對不會自動形成完整語意分析。NLP 02 與 NLP 12 會明確分析這些情況。

稀疏特徵的兩個限制，引出本課其餘內容。第一，每個 bigram 都是獨立座標：除非 “not good” 與 “not great” 都帶著標籤出現在訓練資料中，否則兩者毫無共享。第二，可能的 n-gram 數量隨 n 快速增加，所以長模式很少被觀察到。神經模型同時處理這兩點：把詞表示成能共享證據的向量，並學習哪些向量組合重要。

從詞 ID 到學習得到的向量：嵌入層

白話版。給每個詞彙項目一串短短的、可學習的數字。在訓練任務中行為相似的詞，可能得到相似的向量，於是關於一個詞的證據也能幫助另一個詞。

嵌入層是一張表，每個詞彙項目占一列。查詢一個 token，等同於把它的 one-hot 向量乘上這張表：

$$\mathbf{e}_t = E^\top \mathbf{o}_{i_t} = E_{i_t,:}^\top, \qquad E \in \mathbb{R}^{V \times D}.$$

其中 i_t 是位置 t 上 token 的詞彙 ID，$\mathbf{o}_{i_t} \in \{0,1\}^V$ 是只在座標 i_t 為一的 one-hot 向量，E 是有 V 列、D 行的可學習嵌入表，$E_{i_t,:}$ 是第 i_t 列，$\mathbf{e}_t \in \mathbb{R}^D$ 是該 token 的嵌入（寫成行向量）。D 通常遠小於 V，這就是嵌入被稱為**稠密**（dense）的原因：和稀疏計數向量不同，它大多數座標都不為零。

查表對訓練有一個實用的後果。損失對 E 的梯度只在目前批次出現過的 token 那幾列不為零，所以每次更新只動到少數幾列。從未在訓練中出現的詞會保留初始值，這也是詞彙表需要明確的未知詞項目以及使用規則的原因。這張表可以隨機初始化並與分類器一起訓練，也可以用在未標註文字上預訓練的向量初始化；NLP 04 說明這些向量如何學得，以及它們的幾何意義與限制。

先池化再分類：詞嵌入袋 MLP

文件的 token 數不固定，分類層卻需要固定寬度的輸入。最簡單的神經做法，是把 token 向量平均，再送進**多層感知器**（multilayer perceptron, MLP）：

$$\mathbf{h} = \frac{1}{T}\sum_{t=1}^{T} \mathbf{e}_t, \qquad \mathbf{a} = \phi\left(W_1 \mathbf{h} + \mathbf{b}_1\right), \qquad \mathbf{s} = W_2 \mathbf{a} + \mathbf{b}_2, \qquad p_\theta(y = k \mid d) = \operatorname{softmax}(\mathbf{s})_k.$$

其中 T 是文件 d 中真實 token 的數量，$\mathbf{h} \in \mathbb{R}^D$ 是平均池化後的文件向量，$W_1 \in \mathbb{R}^{H \times D}$ 與 $\mathbf{b}_1 \in \mathbb{R}^H$ 是隱藏層的權重與偏置，H 是隱藏寬度，ϕ 是逐元素的非線性函數（例如 ReLU），$\mathbf{a}$ 是隱藏層啟動值，$W_2 \in \mathbb{R}^{K \times H}$ 與 $\mathbf{b}_2 \in \mathbb{R}^K$ 產生類別分數 $\mathbf{s}$，而 $\operatorname{softmax}(\mathbf{s})_k = \exp(s_k) / \sum_\ell \exp(s_\ell)$。參數 $\theta = (E, W_1, \mathbf{b}_1, W_2, \mathbf{b}_2)$ 以和邏輯斯模型相同的交叉熵共同訓練，梯度由反向傳播計算。

兩個已發表的設計顯示這個簡單形狀能走多遠。Joulin 等人把詞與 n-gram 嵌入平均後送進線性分類器，以深度換取速度 [7]。Iyyer 等人的深度平均網路（deep averaging network）在平均之上堆疊數個非線性層，並加入**詞丟棄**（word dropout），在訓練時先移除整個 token 再平均 [6]。Goldberg 的入門綜述詳細說明這類前饋文字模型 [5]。

隱藏層很重要。對平均向量的線性模型只能把每個詞的證據相加，因此無法表示「*good* 是正面，除非出現 *not*」這種交互作用。非線性隱藏層可以把「not」的出現與正面詞的出現組合起來。但任何以池化為基礎的模型都無法恢復順序：token 的任何排列都給出相同的平均。“The screen is good but the case is bad” 與 “the case is good but the screen is bad” 是完全相同的輸入。本課後面的實驗會量測這個失敗。

神經機率語言模型

白話版。語言模型根據前面的詞預測下一個詞。Bengio 等人證明，若每個先前的詞都以學習得到的向量表示，網路就能在相似的歷史之間共享證據，並為從未見過的詞序列給出合理的機率。

語言模型（language model）一次預測一個 token，藉此為序列指定機率。以 w_t 表示位置 t 的 token，連鎖律給出

$$P(w_1, \dots, w_T) = \prod_{t=1}^{T} P(w_t \mid w_1, \dots, w_{t-1}).$$

其中 $P(w_1, \dots, w_T)$ 是整個序列的機率，$P(w_t \mid w_1, \dots, w_{t-1})$ 是在前面所有 token 已知時下一個 token 的條件機率。計數式 n-gram 模型只用前 $n-1$ 個 token 近似每個因子，並以計數估計；NLP 05 連同平滑與困惑度一起說明它們。它們的弱點是歷史 “a dog” 與 “a cat” 毫無共享：如果前者從未出現，計數對它就沒有任何資訊。

Bengio 等人保留固定的 $(n-1)$ 個 token 歷史，但以神經網路取代計數表 [8]。每個歷史詞都在共享的表 C 中查詢，向量串接後，由一個隱藏層為每個詞彙表中的詞產生分數：

$$\mathbf{x} = [C(w_{t-1}); C(w_{t-2}); \dots; C(w_{t-n+1})], \qquad \mathbf{y} = \mathbf{b} + W\mathbf{x} + U \tanh(\mathbf{d} + H\mathbf{x}), \qquad \hat{P}(w_t = i \mid w_{t-n+1}^{t-1}) = \frac{\exp(y_i)}{\sum_{j=1}^{V} \exp(y_j)}.$$

其中 $C(w) \in \mathbb{R}^m$ 是詞 w 學習得到的特徵向量（$V \times m$ 表 C 的一列），分號表示串接，因此 $\mathbf{x} \in \mathbb{R}^{(n-1)m}$；$H \in \mathbb{R}^{h \times (n-1)m}$ 與 $\mathbf{d} \in \mathbb{R}^h$ 是有 h 個隱藏單元的隱藏層權重與偏置；$U \in \mathbb{R}^{V \times h}$ 把隱藏單元對應到詞彙分數；$W \in \mathbb{R}^{V \times (n-1)m}$ 是從詞特徵到輸出的選用**直接連接**（direct connection），不使用時設為零；$\mathbf{b} \in \mathbb{R}^V$ 是輸出偏置；$\mathbf{y}$ 是未正規化的對數機率；V 是詞彙表大小；w_{t-n+1}^{t-1} 是歷史的縮寫。這就是論文的式 (1)，只是改用本課符號。訓練目標是最大化訓練 token 的平均對數概似，再加上權重衰減懲罰 [8]。

這個模型中的三個想法會在本課程後面反覆出現。詞向量是預測任務的副產品。和平均不同，串接保留了每個歷史詞的位置，因此 “dog bites” 與 “bites dog” 不同。而整個詞彙表上的 softmax 是昂貴的部分：每預測一個 token，其成本都隨 V 增加，這後來促成了較便宜的目標函數，例如 NLP 04 討論的負取樣。

下面的範例在原創語料上訓練一個極小版本，語料中從未出現歷史 “a dog”。為了方便，我們使用 Adam，而不是論文的隨機梯度程序。

```
import torch
from torch import nn
from torch.nn import functional as F

torch.manual_seed(0)
torch.set_num_threads(1)
corpus = """a cat sleeps . a cat eats . the cat sleeps . the cat eats .
the dog sleeps . the dog eats . a bird sings . the bird sings .
a bird eats . the bird eats""".split(".")      # the history "a dog" never occurs
sentences = [["<s>", "<s>"] + s.split() + ["</s>"] for s in corpus]
vocab = sorted({w for s in sentences for w in s})
ids = {w: i for i, w in enumerate(vocab)}
X, Y = [], []
for s in sentences:
    for t in range(2, len(s)):                  # predict w_t from (w_{t-2}, w_{t-1})
        X.append([ids[s[t - 2]], ids[s[t - 1]]])
        Y.append(ids[s[t]])
X, Y = torch.tensor(X), torch.tensor(Y)

class NPLM(nn.Module):
    def __init__(self, V, m=8, h=16, context=2, direct=True):
        super().__init__()
        self.C = nn.Embedding(V, m)                     # shared word features
```

```
        self.H = nn.Linear(context * m, h)              # hidden layer (bias = d)
        self.U = nn.Linear(h, V)                        # hidden-to-output (bias = b)
        self.W = nn.Linear(context * m, V, bias=False) if direct else None
    def forward(self, ctx):
        x = self.C(ctx).flatten(1)                      # concatenate the context vectors
        y = self.U(torch.tanh(self.H(x)))
        return y + self.W(x) if self.W is not None else y

model = NPLM(len(vocab))
opt = torch.optim.Adam(model.parameters(), lr=0.02, weight_decay=1e-4)
for step in range(400):
    opt.zero_grad()
    loss = F.cross_entropy(model(X), Y)
    loss.backward()
    opt.step()
print(f"training cross-entropy {loss.item():.3f} nats, {len(Y)} predictions")

history_count = sum(s[i:i + 2] == ["a", "dog"] for s in sentences for i in range(len(s)))
with torch.no_grad():
    p = F.softmax(model(torch.tensor([[ids["a"], ids["dog"]]])), dim=-1)[0]
print("training occurrences of the history 'a dog':", history_count)
for w in ["sleeps", "eats", "sings"]:
    print(f"P({w} | a dog) = {p[ids[w]].item():.3f}")
```

執行結果：

```
training cross-entropy 0.576 nats, 40 predictions
training occurrences of the history 'a dog': 0
P(sleeps | a dog) = 0.509
P(eats | a dog) = 0.490
P(sings | a dog) = 0.001
```

三元計數模型對歷史 “a dog” 沒有資料，必須退回較短的歷史。神經模型則把幾乎所有機率放在狗（以及貓）在訓練中會搭配的兩個動詞上，而幾乎不給只有鳥會做的 “sings”。它能推廣，是因為 “dog” 與 “cat” 因共享上下文而得到相關的向量，“a” 與 “the” 也是如此。這正是 Bengio 等人的核心主張，在此以玩具規模重現；它完全無法說明模型在真實文字上與平滑良好的 n-gram 模型相比如何，而後者才是論文實際量測的內容 [8]。

卷積文字分類器

白話版。卷積濾波器沿著句子滑動，只要看到某種短片語模式就會被激發，不論模式出現在哪裡。取每個濾波器最強的回應，就得到固定大小的摘要：它記得哪些模式出現過，卻不記得在哪裡出現。

Collobert 等人建立了一個用於標註任務的統一神經網路，其中對詞向量做卷積、再沿位置做最大池化，以摘要整個句子 [9]。Kim 證明，在預訓練詞向量之上只用一層卷積，就是很強的句子分類器 [10]。卷積本身的權重共享與池化，在 ML 03 中以影像說明；文字使用一維版本，只沿位置滑動。

一個濾波器的數學

把句子的 T 個 token 嵌入堆成矩陣 $X \in \mathbb{R}^{T\times D}$，第 t 列為 $\mathbf{e}_t^\top$。寬度為 h 的濾波器涵蓋 h 個連續的列：

$$c_t = \rho(\langle W, X_{t:t+h-1}\rangle + b), \qquad t = 1, \dots, T-h+1, \qquad \hat{c} = \max_t c_t.$$

其中 $X_{t:t+h-1} \in \mathbb{R}^{h\times D}$ 是第 t 到 $t+h-1$ 列組成的視窗，$W \in \mathbb{R}^{h\times D}$ 是濾波器的學習權重，$\langle A, B\rangle = \sum_{r,s} A_{rs}B_{rs}$ 為逐元素乘積的總和，$b \in \mathbb{R}$ 是偏置，ρ 是 ReLU 或 tanh 之類的非線性函數，c_t 是濾波器在視窗 t 的回應，序列 $(c_1, \dots, c_{T-h+1})$ 是**特徵圖**（feature map），$\hat{c}$ 是**時間軸最大池化**的結果。Kim 把視窗攤平成長度 hD 的向量，寫出相同的運算 [10]。

模型會使用許多濾波器。若每個寬度（例如 $\{3,4,5\}$）各有 F 個濾波器，池化後的值組成向量 $\mathbf{z} \in \mathbb{R}^{F\cdot(\text{number of widths})}$，再由線性層與 softmax 轉成類別機率，和前面一樣以交叉熵訓練。在 PyTorch 中，`nn.Conv1d(D, F, h)` 一次算出寬度 h 的全部 F 張特徵圖；它要求輸入形狀為 (batch, D, T)，因此嵌入輸出必須先轉置。

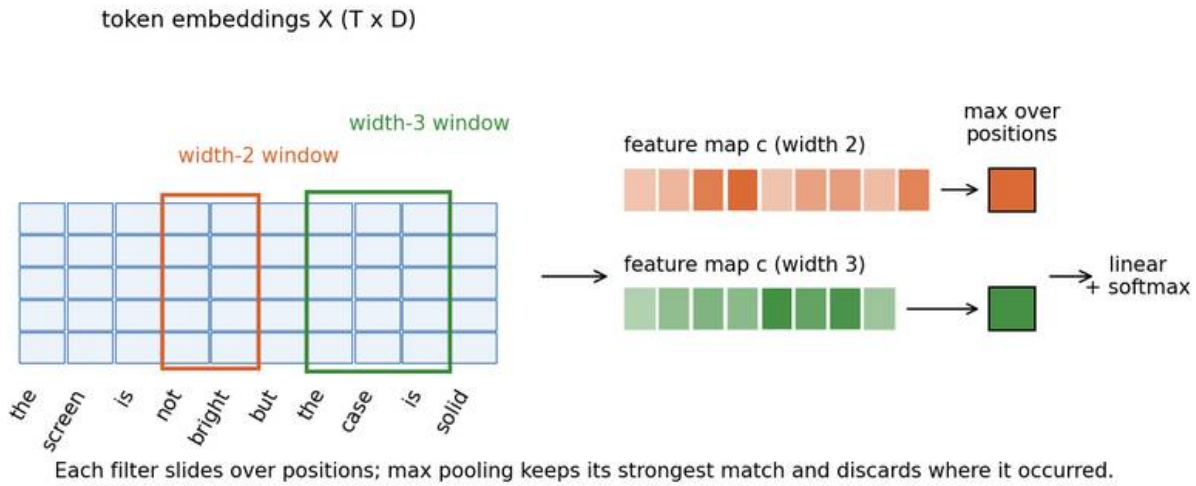


圖 3 — 一維卷積文字分類器。每個濾波器為其寬度的每個視窗打分；時間軸最大池化保留最強分數並丟棄它的位置。

各部分帶來什麼，又失去什麼

權重共享。同一個濾波器套用在每個位置，因此從一句開頭的 “not good” 學到的模式，也能在另一句的結尾偵測到。參數量取決於 h、D 與 F，與句子長度無關。

軟性 n-gram。由於濾波器是與嵌入做內積，對 “not good” 有反應的濾波器，若兩個形容詞向量相近，也會對 “not great” 產生較弱的反應。這是相對於稀疏 bigram 特徵的優勢。

最大池化。只保留最大值能處理不同句長，並記錄某模式是否在任何地方出現過。它丟掉了模式出現的位置與次數。單層濾波器也無法連接距離超過其寬度的 token：寬度最多為 4 時，模型中沒有任何視窗能同時看到 “but” 與四個位置之後的形容詞。下面的實驗正好顯示這一點。

Kim 的論文比較四種處理嵌入表的方式：隨機初始化並與模型一起訓練（CNN-rand）、預訓練向量保持固定（CNN-static）、預訓練向量加以微調（CNN-non-static），以及兩個通道、一個固定一個微調、每個濾波器同時套用在兩者上（CNN-multichannel）[10]。他回報的設定是濾波器寬度 3、4、5 各 100 張特徵圖、池化層的丟棄率 0.5、輸出權重向量的 ℓ_2 範數上限為 3，以及大小 50 的小批次 [10]。Zhang 與 Wallace 後來對這種單層設計做了敏感度分析，結論是濾波器區域大小可能影響很大且應該調整、1-max 池化在他們試過的池化策略中一致表現最好、ReLU 與 tanh 是最佳的啟動函數候選，而比較設定時必須報告不同執行之間的變異 [11]。

循環編碼器（recurrent encoder）走另一條處理順序的路：一邊讀序列一邊更新隱藏狀態，原則上能連接遠距離的 token。Hagiwara 的句子分類內容結合循環模型與實務訓練 [1]。NLP 05 針對語言發展它們，ML 04 說明共通數學。以注意力為基礎的上下文編碼器讓每個 token 的表示取決於整個輸入；NLP 07 與 NLP 10 會發展它們。

選擇應以證據為依據。如果標籤有清楚的詞彙線索，稀疏分類器可能已足夠。卷積或上下文編碼器可能改善間接說法、否定或領域詞義；更大的容量也讓資料切分與標註品質更重要。複雜度應回答已量測的問題，例如改述句的錯誤，而不是取代評估。

僅線上版：*TextCNN — Convolutional Neural Networks for Sentence Classification — https://blog.shyandram.dev/zh-tw/learn/nlp/neural-networks-for-nlp/papers/textcnn–convolutional-neural-networks-for-sentence-classification/*

填補、遮罩與批次

白話版。句子長短不一，批次卻是矩形。我們用填補 token 補滿短句，再確保填補永遠不影響結果。

訓練時一次處理多個序列，組成形狀為（batch, $T_{\max}$）的張量，$T_{\max}$ 是批次中最長的長度。較短的序列以保留的**填補 ID**（padding ID，通常為 0）補齊。布林**遮罩**（mask）m_{bt} 記錄哪些位置是真實的：

$$m_{bt} = \mathbf{1}[t \le T_b], \qquad \mathbf{h}_b = \frac{\sum_{t=1}^{T_{\max}} m_{bt}\,\mathbf{e}_{bt}}{\max\!\left(1, \sum_{t=1}^{T_{\max}} m_{bt}\right)}.$$

其中 b 索引批次中的序列，t 索引位置，T_b 是序列 b 的真實長度，$\mathbf{1}[\cdot]$ 在條件成立時為一、否則為零，$\mathbf{e}_{bt}$ 是序列 b 第 t 個位置的嵌入，$\mathbf{h}_b$ 是**遮罩平均**。$\max(1, \cdot)$ 防止空序列。每種池化運算都需要自己的遮罩規則：

- **平均池化**必須除以真實長度，而不是 $T_{\max}$。零填補向量不改變總和，卻會改變未遮罩的分母。
- **最大池化**必須排除填補位置或視窗。一般規則是在取最大值前填入 $-\infty$。經過 ReLU 後所有回應都非負，所以填入 0 是等價的，還多一個好處：比濾波器寬度還短的句子完全沒有有效視窗，此時得到 0 而不是 $-\infty$，否則下一層會變成 NaN。
- 與填補重疊的**卷積視窗**不是真實視窗。採右側填補時，從位置 t 開始、寬度 h 的視窗，當且僅當位置 $t + h - 1$ 是真實位置時才有效。
- 針對 token 位置的**損失**（之後的標註與語言模型會用到）必須忽略填補位置；PyTorch 的交叉熵為此提供 `ignore_index`。

PyTorch 的 `nn.Embedding(..., padding_idx=0)` 讓填補列保持為零，並把它排除在梯度更新之外。這有幫助，但不會自動讓平均或最大值正確。檢驗方法是一個不變性：多加填補不能改變任何輸出。

```
import torch
from torch import nn
from torch.nn import functional as F

torch.manual_seed(0)
emb = nn.Embedding(6, 4, padding_idx=0)            # ID 0 is reserved for padding
seqs = [torch.tensor([3, 5, 2]), torch.tensor([4, 1])]
x = nn.utils.rnn.pad_sequence(seqs, batch_first=True, padding_value=0)
print(x.tolist())                                   # [[3, 5, 2], [4, 1, 0]]

# A lookup is a one-hot row times the table.
one_hot = F.one_hot(x, num_classes=6).float()
assert torch.allclose(one_hot @ emb.weight, emb(x))

def masked_mean(ids):
    mask = (ids != 0).unsqueeze(-1).float()
    return (emb(ids) * mask).sum(1) / mask.sum(1).clamp(min=1)

longer = F.pad(x, (0, 3), value=0)                  # three more padding columns
print("masked mean unchanged:", torch.allclose(masked_mean(x), masked_mean(longer)))
print("plain mean unchanged: ", torch.allclose(emb(x).mean(1), emb(longer).mean(1)))
print("padding row:", emb.weight[0].tolist())
```

執行結果：

```
[[3, 5, 2], [4, 1, 0]]
masked mean unchanged: True
plain mean unchanged:  False
padding row: [0.0, 0.0, 0.0, 0.0]
```

填補列為零，普通平均卻仍然改變，因為它的分母把填補也算了進去。任何新的編碼器都應加入這類不變性檢查。

批次在正確性之外還有實務後果。填補浪費計算，因此訓練程式常把長度相近的序列放進同一批次（**分桶**，bucketing）。過長的輸入會被截斷到最大長度；請記錄這個上限，因為超出部分模型看不到，並決定要保留開頭、結尾或兩者。每個 epoch 都要打亂批次，讓模型不會看到相同的樣本順序；評估時要把模型切到評估模式，以關閉 dropout。

小型文字分類器的正則化

白話版。神經文字模型的參數足以背下小型訓練集，連其中的標註錯誤也一起背下。正則化讓背誦變難，而驗證集告訴我們何時該停。

這些工具就是 ML 02 中的通用工具，只是用在文字模型的特定位置：

- **Dropout** 在訓練時以機率 p 把一層中的每個單元設為零，並把其餘單元重新縮放，讓任何單元都不能依賴特定夥伴 [12]。在文字 CNN 中通常套用在池化後的特徵向量上，如 Kim 的模型 [10]。PyTorch 的 `nn.Dropout(p)` 採用的慣例是：p 是丟棄單元的機率。
- **詞丟棄**在訓練時從輸入移除整個 token，促使模型不依賴任何單一詞 [6]。
- **權重衰減**（weight decay）在損失中加入 $\frac{\lambda}{2}\|\theta\|_2^2$，與上面邏輯斯目標的懲罰相同。嵌入表常是最大的參數區塊，因此它是否被衰減或凍結很重要。
- **最大範數限制**（max-norm constraint）在權重向量的歐幾里得範數超過上限時把它重新縮放，如 Kim 的輸出層 [10]。
- **提前停止**（early stopping）保留驗證損失最佳那個 epoch 的參數。它本身也是一種正則化：限制最佳化朝背下訓練集的方向走多遠。
- **凍結預訓練嵌入**直接移除最大的一塊可訓練參數，代價是無法讓詞義適應任務。

這些選擇都是在驗證資料上選定的超參數。一次只改一項，並在下結論說某選擇有幫助之前，報告不同隨機種子間的變化；Zhang 與 Wallace 對文字 CNN 正是強調這一點 [11]。

完整的 PyTorch 實驗

白話版。現在在一個順序確實重要的原創評論任務上比較四個分類器，判讀它們的訓練曲線，並依類別檢查錯誤。

每則合成評論要不是一個子句 "the *aspect* is (*not*) *adjective*"，就是以 "but" 連接的兩個子句。當決定性子句為正面時，標籤為正；否定會翻轉形容詞，而在 "but" 評論中由第二個子句決定。訓練集有 300 則評論，其中 10% 的標籤被刻意翻轉以模擬標註雜訊；驗證集（300 則）與測試集（1,000 則）的標籤是乾淨的。每則測試評論都標記為 `plain`、`negated` 或 `contrast`，以便依類別計算錯誤。

```
import copy, random
import numpy as np
import torch
from torch import nn
from torch.nn import functional as F
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression

torch.set_num_threads(1)
ASPECTS = ["battery", "screen", "camera", "speaker", "case", "charger"]
POS, NEG = ["good", "great", "solid", "bright"], ["bad", "weak", "noisy", "dull"]

def clause(rng):
    aspect, positive = rng.choice(ASPECTS), rng.random() < 0.5
    negated = rng.random() < 0.4
    words = ["the", aspect, "is"] + ["not"] * negated + [rng.choice(POS if positive else
     ↪  NEG)]
    return words, int(positive != negated), negated

def example(rng):                        # label of "A but B" is the label of B
    words, label, negated = clause(rng)
    if rng.random() < 0.5:
```

```
        second, label, _ = clause(rng)
        return " ".join(words + ["but"] + second), label, "contrast"
    return " ".join(words), label, "negated" if negated else "plain"

def make(n, seed):
    rng = random.Random(seed)
    return [example(rng) for _ in range(n)]

train, val, test = make(300, 1), make(300, 2), make(1000, 3)
flip = random.Random(4)                 # 10% training-label noise; val/test stay clean
train = [(t, 1 - y if flip.random() < 0.1 else y, c) for t, y, c in train]
y_test, kind = np.array([r[1] for r in test]), np.array([r[2] for r in test])

def report(name, pred):
    parts = {k: round(float((pred[kind == k] == y_test[kind == k]).mean()), 3)
             for k in ["plain", "negated", "contrast"]}
    print(f"{name:12s} test acc {np.mean(pred == y_test):.3f}", parts)

for ngrams in [(1, 1), (1, 2)]:
    vec = TfidfVectorizer(ngram_range=ngrams, token_pattern=r"\S+")
    clf = LogisticRegression(C=10.0, max_iter=1000)
    clf.fit(vec.fit_transform([r[0] for r in train]), [r[1] for r in train])
    report(f"tfidf {ngrams}", clf.predict(vec.transform([r[0] for r in test])))

vocab = ["<pad>", "<unk>"] + sorted({w for r in train for w in r[0].split()})
ids = {w: i for i, w in enumerate(vocab)}

def encode(rows):                       # right-pad every sequence with ID 0
    seqs = [torch.tensor([ids.get(w, 1) for w in r[0].split()]) for r in rows]
    return (nn.utils.rnn.pad_sequence(seqs, batch_first=True, padding_value=0),
            torch.tensor([r[1] for r in rows]))

class MeanMLP(nn.Module):               # embedding -> masked mean -> MLP
    def __init__(self, V, D=16, H=32, p=0.3):
        super().__init__()
        self.emb = nn.Embedding(V, D, padding_idx=0)
        self.mlp = nn.Sequential(nn.Linear(D, H), nn.ReLU(), nn.Dropout(p), nn.Linear(H, 2))
    def forward(self, x):
        m = (x != 0).unsqueeze(-1).float()
        return self.mlp((self.emb(x) * m).sum(1) / m.sum(1).clamp(min=1))

class TextCNN(nn.Module):               # embedding -> 1-D conv -> masked max -> linear
    def __init__(self, V, D=16, widths=(2, 3, 4), maps=16, p=0.3):
        super().__init__()
        self.emb = nn.Embedding(V, D, padding_idx=0)
        self.convs = nn.ModuleList(nn.Conv1d(D, maps, h) for h in widths)
        self.drop, self.out = nn.Dropout(p), nn.Linear(maps * len(widths), 2)
    def forward(self, x):
        e, real = self.emb(x).transpose(1, 2), x != 0         # e: (batch, D, T)
        pooled = []
        for conv in self.convs:
            c = torch.relu(conv(e))                           # (batch, maps, T-h+1)
            valid = real[:, conv.kernel_size[0] - 1:].unsqueeze(1)  # window ends on a real
             ↪  token
            # ReLU outputs are >= 0, so filling invalid windows with 0 never wins the max,
```

```
            # and a text shorter than the filter gets 0 instead of -inf (which would give
             ↪ NaN).
            pooled.append(c.masked_fill(~valid, 0.0).max(dim=2).values)
        return self.out(self.drop(torch.cat(pooled, dim=1)))

def fit(model, weight_decay, epochs=80, batch=32):
    (xt, yt), (xv, yv) = encode(train), encode(val)
    opt = torch.optim.Adam(model.parameters(), lr=0.01, weight_decay=weight_decay)
    best, history = (float("inf"), None, 0), []
    for epoch in range(1, epochs + 1):
        model.train()
        for b in torch.randperm(len(yt)).split(batch):
            opt.zero_grad()
            F.cross_entropy(model(xt[b]), yt[b]).backward()
            opt.step()
        model.eval()
        with torch.no_grad():
            history.append((F.cross_entropy(model(xt), yt).item(),
                            F.cross_entropy(model(xv), yv).item()))
        if history[-1][1] < best[0]:                          # early stopping on val loss
            best = (history[-1][1], copy.deepcopy(model.state_dict()), epoch)
    model.load_state_dict(best[1])
    return history, best[2]

x_test, _ = encode(test)
for name, build, wd in [("mean+MLP", lambda: MeanMLP(len(vocab)), 1e-4),
                        ("CNN", lambda: TextCNN(len(vocab)), 1e-4),
                        ("CNN no-reg", lambda: TextCNN(len(vocab), p=0.0), 0.0)]:
    torch.manual_seed(0)
    model = build()
    history, best_epoch = fit(model, wd)
    with torch.no_grad():
        pred = model(x_test).argmax(1).numpy()
    print(f"{name:12s} last train/val loss {history[-1][0]:.3f}/{history[-1][1]:.3f}, "
          f"best epoch {best_epoch}")
    report(name, pred)
    if name == "CNN":
        cnn_pred = pred

wrong = np.flatnonzero(cnn_pred != y_test)
print("CNN errors by kind:", {k: int((kind[wrong] == k).sum()) for k in ["plain", "negated",
 ↪ "contrast"]})
for i in wrong[:4]:
    print(" ", y_test[i], "<-", test[i][0])
```

整個腳本在單一 CPU 執行緒上數秒內完成，輸出：

```
tfidf (1, 1) test acc 0.546 {'plain': 0.651, 'negated': 0.348, 'contrast': 0.555}
tfidf (1, 2) test acc 0.880 {'plain': 1.0, 'negated': 1.0, 'contrast': 0.77}
mean+MLP     last train/val loss 0.269/0.464, best epoch 36
mean+MLP     test acc 0.797 {'plain': 0.973, 'negated': 0.945, 'contrast': 0.645}
CNN          last train/val loss 0.144/0.431, best epoch 30
CNN          test acc 0.898 {'plain': 0.983, 'negated': 0.945, 'contrast': 0.833}
```

```
CNN no-reg   last train/val loss 0.092/0.444, best epoch 10
CNN no-reg   test acc 0.910 {'plain': 1.0, 'negated': 0.928, 'contrast': 0.852}
CNN errors by kind: {'plain': 5, 'negated': 10, 'contrast': 87}
  0 <- the camera is not dull but the battery is weak
  0 <- the camera is noisy but the screen is bad
  1 <- the charger is solid but the battery is solid
  1 <- the charger is solid but the charger is bright
```

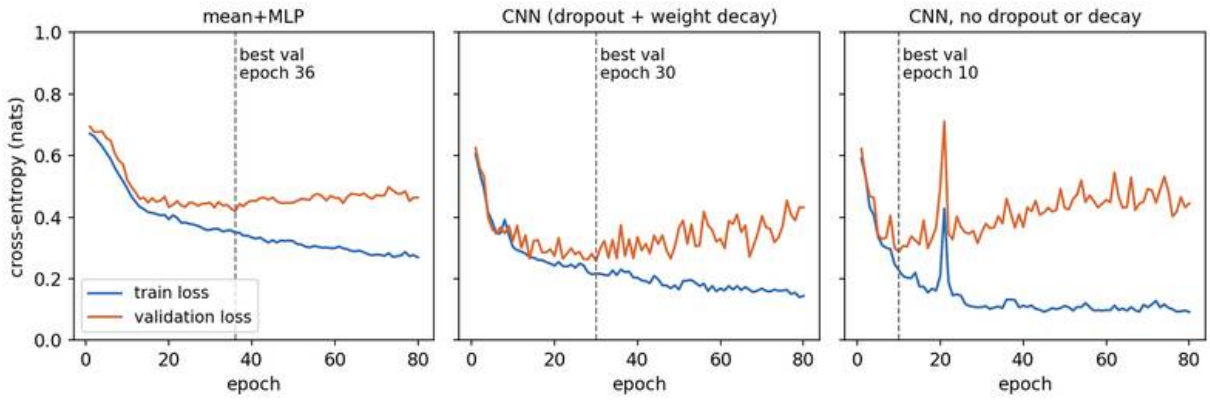


圖 *4* — 實驗中三個神經模型的訓練（藍）與驗證（橘）損失，種子為 *0*。虛線標出提前停止所保留的 *epoch*。

依類別解讀結果

Unigram TF–IDF 在設計上就無法處理否定。在這份資料中，“not” 出現在正面與負面評論的頻率相同，而每個形容詞是正是負取決於是否有 “not”。只看詞是否出現的線性模型無法表示這種交互作用，因此否定評論只得到 0.348，比隨機猜還差。

Bigram 修好了否定，卻修不好對比。“not good” 這個詞對是一個獨立特徵，所以 bigram 模型在一般與否定評論上都完全正確。在對比評論上它達到 0.77：它看得到兩個子句的 bigram，卻沒有可靠的方法知道哪個子句在 “but” 之後。

池化 MLP 解決了交互作用，卻解決不了順序。它的隱藏層能把 “not” 與形容詞結合，因此即使輸入是無序的平均，否定評論仍達到 0.945。在對比評論上它只有 0.645，因為平均池化讓 “A but B” 與 “B but A” 完全相同。

CNN 在對比評論上最好，但受限於寬度。濾波器寬度最多為 4 時，它在對比評論上達到 0.833。它的 102 個錯誤中只有 15 個不是對比評論，列出的錯誤說明了原因：在 “the charger is solid but the battery is solid” 中，決定性的形容詞位在 “but” 之後四個位置，沒有任何單一視窗能同時包含兩者。濾波器只能依賴間接線索。練習 3 請你加入寬度 5 的濾波器來檢驗這個解釋。

解讀曲線

三張面板呈現相同模式：訓練損失持續下降，驗證損失則停止改善，然後上升。訓練標籤中有 10% 被翻轉，因此訓練損失遠低於這個雜訊水準，就表示模型正在背下錯誤。沒有正則化的 CNN 最快這麼做：它的驗證損失在第 10 個 epoch 觸底，到第 80 個 epoch 時明顯變差。Dropout 與權重衰減減緩了這種漂移，有正則化的 CNN 最佳驗證損失出現得較晚。

然而在測試準確率上，一旦提前停止為每個模型選定 epoch，沒有正則化的 CNN（0.910）與有正則化的 CNN（0.898）相當接近。只用一個種子、差距約一個百分點，這次執行無法排出優劣。站得住的結論比較窄：在這裡，以乾淨驗證集做提前停止是必要的，而有正則化的模型對何時停止比較不敏感。真正的比較應以多個種子重複實驗並報告分布範圍 [11]。

把錯誤分析變成習慣

能夠依類別計數，是因為每個測試樣本都帶著描述它為何可能困難的標記。從一開始就把這類標記放進評估資料：否定、對比、罕見詞、長度、來源。接著逐一閱讀錯誤，並問所需的證據是在哪個階段遺失的。如果平均池化讓兩個輸入完全相同，再多標籤也幫不了那個模型；如果 CNN 的視窗太短，解法是更寬的濾波器或不同的編碼器；如果標註本身錯了，解法在資料裡。

分開評估錯誤與機率

白話版。分類器可能選對標籤，卻提供不可靠的機率。決策與不確定性估計應分別評估。

把一個類別視為正類時：

$$P = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP}}, \qquad R = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}}, \qquad F_1 = \frac{2PR}{P + R}.$$

其中 P 是精確率（precision），R 是召回率（recall），TP 是真陽性數，FP 是偽陽性數，FN 是偽陰性數，F_1 是精確率與召回率的調和平均。分母為零時，必須明確選擇慣例。Macro F1 平等平均各類別 F1；依樣本數加權的平均則讓常見類別影響較大。指標文件說明這些選擇 [13]。

Macro 平均讓罕見類別的失敗比較容易看見，卻不代表失敗的實際成本。漏掉緊急客服案件與誤送一般收據，可能有不同後果。應另外寫出成本政策，而不是把方便計算的分數當成完整效用。

校準（calibration）檢查預測信心是否與相似樣本的觀察正確率一致。模型可以保留 argmax 決策，同時改變信心。Guo 等人研究這個差別與訓練後的溫度縮放 [14]。校準參數必須使用保留的開發資料擬合，且需要足夠樣本估計機率行為。訓練到幾乎零訓練損失的神經分類器，是過度自信機率的常見來源，因此每次模型改變都應檢查校準。

選擇性預測可寫成：

$$\mathrm{coverage} = \frac{1}{n}\sum_{i=1}^{n} a_i, \qquad \mathrm{risk}_{\mathrm{accepted}} = \frac{\sum_{i=1}^{n} a_i \mathbf{1}[\hat{y}_i \neq y_i]}{\sum_{i=1}^{n} a_i}.$$

其中 n 是評估樣本數，$a_i \in \{0, 1\}$ 表示是否接受第 i 個樣本，$\hat{y}_i$ 是預測，y_i 是參考標籤，$\mathbf{1}[\cdot]$ 在條件成立時為一，否則為零。如果沒有接受任何樣本，接受風險便未定義。風險與涵蓋率曲線呈現不同門檻下的取捨；部署門檻應依驗證資料選擇，而不是依最終測試標籤選擇。

不平衡、錯誤分析與部署變化

白話版。整體分數提高時，重要使用者群組的表現仍可能變差。應檢查錯誤類型、罕見類別與發生情境。

建立錯誤表，包含文字、標準標籤、預測標籤、機率、語言、長度、來源群組與分析者的解釋。實用錯誤類型包括標註歧義、缺少脈絡、否定、未見詞彙、多主題輸入、引述，以及確實超出範圍的文字。不同類型需要不同修正。更多訓練樣本可能改善未見詞彙，卻無法補上預測時永遠拿不到的對話回合。

類別加權改變每個訓練樣本的貢獻：

$$\mathcal{L}_{\text{weighted}} = -\frac{1}{n}\sum_{i=1}^{n} \alpha_{y_i} \log p_\theta(y_i \mid d_i).$$

其中 $\mathcal{L}_{\text{weighted}}$ 是加權交叉熵，n 是訓練樣本數，i 索引樣本，d_i 是文件，y_i 是標註類別，$\alpha_{y_i} > 0$ 是該類別的權重，p_θ 是分類器分布。加權政策改變訓練目標，但不能創造缺少的語言多樣性。

加權、過度取樣與門檻調整回答不同問題。加權改變參數學習；過度取樣改變呈現頻率，有時只是不斷重複少數類別中的同幾個樣本；門檻調整則在分數算出後改變行動政策。Hagiwara 的開發章節討論不平衡與訓練選擇 [1]。任何介入都應以真實類別盛行率評估，而不是以人為平衡的最終測試集評估。

部署變化可能影響詞彙、類別比例與文字到標籤的關係。新功能上線可能帶來新的技術錯誤詞彙；假日可能增加配送訊息；分流政策改變則可能重新定義標籤。這些應分開監測。空特徵向量比例增加，或神經模型中未知詞 ID 的比例增加，提示表示涵蓋範圍有問題；分流政策改變，則需要更新標註約定。

把模型與前處理產物一起保存。詞彙順序、IDF 權重或嵌入表、分類器類別順序、最大長度、機率校準與決策門檻，共同構成推論套件。以另一份詞彙表載入權重，會默默改變意義。套件應保存有版本的標籤結構與幾個確定性介面範例，讓服務可以檢查假設。

現代觀點

綜述觀點。 Goldberg 的入門綜述以前饋、卷積、循環與遞迴網路、輸入編碼及計算圖為主軸組織神經 NLP [5]；它仍是本課模型的精簡地圖。Minaee 等人回顧了 150 多個深度學習文字分類器與 40 多個資料集，從前饋與卷積模型一路到循環、注意力與預訓練模型，並比較已回報的基準結果 [15]。這類表格應被視為特定資料集與實驗協定上的證據，而不是能套用到任意標籤結構的排名。

Zhao 等人的大型語言模型綜述依預訓練、調適、使用與評估組織 [16]。對分類而言，這形成一個實用比較：固定表示加學習模型頭、針對任務調適編碼器，以及以指令進行預測。Chang 等人從評估什麼、在哪裡評估、如何評估整理評估方法 [17]。這提醒我們，熟悉的基準並不能證明任意標籤結構都能正常運作。

穩健性觀點。 Sperduti 與 Moreo 的 2025 年拼字錯誤綜述整理拼字變化來源、修正方法，以及對下游任務與現代語言模型的影響 [18]。對本課的實務意義是：評估應包含真實拼字變化，再區分

正規化失敗與分類器失敗。字元特徵、子詞嵌入與上下文表示提供不同做法；沒有任務專屬比較之前，都不應被假定為穩健。

2023–2026 年實務。加上分類頭並微調的預訓練 Transformer 編碼器（見 NLP 11），是在標註資料與運算資源允許時的常見起點。指令跟隨語言模型增加了指定分類任務的方式 [16]。提示式模型可以解讀文字標籤描述，不必另外訓練分類器頭。這把一部分設定轉移到提示、標籤措辭與輸出剖析器。這些產物都應視為經過設計選擇的設定：測試前固定，將輸出限制在允許的結構，並量測保留決策與格式錯誤。

本課的模型仍有其角色。TF–IDF 加邏輯斯迴歸，以及小型卷積或平均網路，訓練便宜、服務快速、容易檢查。它們是判斷較大模型的進步是否真實的基準，而且當標籤有清楚的詞彙線索時往往已經足夠。

應用也可以使用較大模型建議標籤，再由人員檢查，最後訓練較小分類器。這會改變標註流程。生成標籤可能含有系統性錯誤，或重現教師模型的偏好。應檢查由人員確認的子集，並把合成監督與獨立評估標籤分開。能產生更多標籤，不等於監督品質更好。

未解問題。標註者與模型版本之間的標籤理解可能改變；低資源語言可能缺少獨立評估資料；多主題訊息難以符合單標籤結構；領域改變後校準可能惡化；嵌入相似度可能隱藏矛盾。這些問題需要明確的挑戰樣本與持續評估，而不是移除所有脈絡後的架構排名。

重點整理

- 標註約定決定預測問題與可採取的行動；詞彙表、IDF 權重與任何學習得到的表示都只能用訓練文字擬合。
- 稀疏線性基準強大、便宜又具診斷力；bigram 能捕捉局部否定，卻把每個片語視為與其他片語毫無關聯。
- 嵌入層是一張可學習的查詢表；平均池化加非線性 MLP 能學到詞的交互作用，卻看不到順序。
- 神經機率語言模型串接學習得到的詞向量，藉由在相似詞之間共享證據，推廣到未見過的歷史。
- 一維卷積分類器能在文字任何位置偵測軟性 n-gram 模式；最大池化丟棄位置，濾波器寬度限制它能連接的距離。
- 填補需要為平均、最大值、視窗與損失加上明確遮罩；多加填補不能改變任何輸出。
- 標籤有雜訊時，正則化與提前停止最重要；比較設定時應使用多個種子。
- 分開評估類別決策、機率品質、涵蓋率與各類別錯誤。

練習

1. **標籤約定**。定義客服任務的多標籤版本，為「delivery charge」、引述申訴與空白輸入寫出規則。說明輸出啟動函數與損失需要哪些改變。

   提示

   獨立的標籤機率可以允許兩條路徑。明確說明何時沒有標籤得到證據支持，而不是把所有低分都當成負標註。

2. **只用訓練資料的特徵**。加入只在最終測試集出現的詞。檢查已擬合的 TF–IDF 詞彙表與轉換後的測試向量，以及它在神經實驗詞彙表中得到的 ID。並與錯誤地使用所有文字擬合特徵比較。

   提示

   正確詞彙表不包含測試專屬詞；在神經模型中它會對應到 `<unk>`。除了詞彙表大小，也檢查文件頻率；即使所有詞已存在，洩漏仍可能改變權重。

3. **檢驗寬度的解釋**。在 PyTorch 實驗中，把 CNN 的寬度改成 `(2, 3, 4, 5)` 後重新執行。對比評論的準確率是否改變？接著以五個種子分別執行原始與修改後的模型，報告平均與範圍後再下結論。

   提示

   數一數 “but the battery is solid” 有幾個位置。寬度 5 的視窗能同時包含 “but” 與決定性形容詞。除了寬度之外其他一切保持不變，並記得單一種子的差異可能只是雜訊。

4. **故意製造遮罩錯誤**。移除 `MeanMLP` 的遮罩（改用普通的 `.mean(1)`），以及 `TextCNN` 的遮罩（對所有視窗池化）。在測試批次多補五欄填補，並與有遮罩的版本比較預測。哪個模型改變了？為什麼？

   提示

   填補嵌入為零，所以填補越多，普通平均就越往零縮。對 CNN 而言，與填補重疊的視窗仍有非零偏置，可能贏得最大值。

5. **手算神經語言模型**。對於有 V 個詞、$n-1$ 個歷史詞、每詞 m 個特徵、h 個隱藏單元的 NPLM，計算 C、H、$\mathbf{d}$、U、$\mathbf{b}$ 與 W 的參數量。V 很大時哪一項主導？移除直接連接後參數量如何改變？

   提示

   從式子寫出每個矩陣的形狀。輸出層與嵌入表都隨 V 增加，隱藏層則不會。

6. **選擇性決策**。以幾種驗證成本政策執行客服訊息實驗。在驗證資料畫出涵蓋率與接受錯誤，選擇一個門檻，再只在測試上評估一次。

   提示

   若沒有接受任何樣本，接受錯誤的分母為零。應報告未定義，而不是完美準確率。

參考文獻

1. M. Hagiwara, *Real-World Natural Language Processing: Practical Applications with Deep Learning*, Manning, 2021; Chapters 2–4, 7 and 10–11. publisher
2. S. McRoy, *Principles of Natural Language Processing*, open textbook, 2021; Chapter 6, “Benchmark Tasks for Language Modelling.” chapter
3. Scikit-learn developers, “Text feature extraction,” scikit-learn documentation, current online edition. documentation
4. Scikit-learn developers, “Logistic regression,” scikit-learn documentation, current online edition. documentation
5. Y. Goldberg, “A primer on neural network models for natural language processing,” *Journal of Artificial Intelligence Research* 57, 2016. paper
6. M. Iyyer, V. Manjunatha, J. Boyd-Graber and H. Daumé III, “Deep unordered composition rivals syntactic methods for text classification,” ACL-IJCNLP, 2015. paper
7. A. Joulin, E. Grave, P. Bojanowski and T. Mikolov, “Bag of tricks for efficient text classification,” EACL, 2017. paper
8. Y. Bengio, R. Ducharme, P. Vincent and C. Jauvin, “A neural probabilistic language model,” *Journal of Machine Learning Research* 3, 2003. paper
9. R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu and P. Kuksa, “Natural language processing (almost) from scratch,” *Journal of Machine Learning Research* 12, 2011. paper
10. Y. Kim, “Convolutional neural networks for sentence classification,” EMNLP, 2014. paper
11. Y. Zhang and B. Wallace, “A sensitivity analysis of (and practitioners’ guide to) convolutional neural networks for sentence classification,” IJCNLP, 2017. paper
12. N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever and R. Salakhutdinov, “Dropout: a simple way to prevent neural networks from overfitting,” *Journal of Machine Learning Research* 15, 2014. paper
13. Scikit-learn developers, “Classification metrics,” scikit-learn documentation, current online edition. documentation
14. C. Guo, G. Pleiss, Y. Sun and K. Q. Weinberger, “On calibration of modern neural networks,” ICML, 2017. paper
15. S. Minaee, N. Kalchbrenner, E. Cambria, N. Nikzad, M. Chenaghlu and J. Gao, “Deep learning-based text classification: a comprehensive review,” *ACM Computing Surveys* 54(3), 2021. survey
16. W. X. Zhao et al., “A survey of large language models,” *Frontiers of Computer Science*, 2026; online manuscript updated March 2026. survey
17. Y. Chang et al., “A survey on evaluation of large language models,” *ACM Transactions on Intelligent Systems and Technology*, 2024. survey
18. G. Sperduti and A. Moreo, “Misspellings in natural language processing: a survey,” arXiv:2501.16836, 2025. survey

NLP 04・詞彙語意與詞嵌入

從詞義、詞彙關係與 WordNet，走到分布式向量：PPMI、Word2Vec、GloVe 與 fastText，如何評估它們、它們吸收的偏誤、句子嵌入，以及邁向上下文表示的一步。

網站版本: https://blog.shyandram.dev/zh-tw/learn/nlp/lexical-semantics-word-embeddings/

你將學到

- 區分詞形、詞元（lemma）與詞義，並說出主要的詞彙關係：同義、反義、上下位與部分整體關係，以及 WordNet 如何組織它們。
- 解釋分布假設：共享上下文能提供哪些詞彙證據，以及無法確立哪些意義關係。
- 建立詞與上下文矩陣，把正向 PMI 轉成精簡向量，並推導 CBOW、Skip-gram、負取樣、GloVe 與 fastText 的目標函數。
- 以相似度與類比測試評估嵌入，同時認清它們的限制，並量測一種關聯偏誤。
- 將詞向量池化成句子表示、以對比學習訓練句子嵌入，並說明上下文嵌入改變了什麼。

先看全貌

詞 ID 只識別詞彙表項目。它的數值不會說明該詞在語言中做什麼。**嵌入（embedding）**替項目配置學習出的向量，讓不同詞共享有用的統計模式。核心問題不只是如何把向量縮小，而是哪些證據與目標應該決定它的座標。

> **用最簡單的話說**
>
> 想像透過觀察使用場合，認識不熟悉的工具。兩種工具若經常出現在木頭、釘子與工作臺旁，可能具有相關用途。即使沒人替工具命名，你仍能從觀察畫出一張地圖。但是，同在工作臺出現不代表可以互換：鐵鎚與鉗子仍做不同的事。詞向量也從周圍的詞學出類似地圖。

語言學家以**詞彙語意學（lexical semantics）**描述詞義：詞的各種意義，以及同義、上下位等關係，並由人工記錄在 WordNet 這類資源中。分布式方法則改從文字學習向量。本課連結這兩種觀點。Hagiwara 將詞與文件嵌入發展為實務應用可重用的特徵 [1]。McRoy 則區分詞彙相似度與情緒、蘊涵等語言任務 [2]。這裡專注於表示本身，並持續追問：向量幾何實際捕捉的是哪一種詞彙關係。

NLP 03 把詞向量當成任務的副產品來學習：分類器裡的嵌入層，以及 Bengio 等人神經機率語言模型中的表 C [3]。Word2Vec 則把向量本身當成目標，並以鄰近詞、而不是下一個詞，作為預測對象。兩者的預測約定不同：

問題	神經機率語言模型	靜態 Word2Vec 嵌入
觀察什麼？	有順序的 $n-1$ 個 token 歷史，以及緊接其後的 token	中心詞與兩側選定的鄰近詞
預測什麼？	下一個 token 的正規化分布	從鄰近詞猜中心詞，或從中心詞分別猜鄰近詞
目的是由左至右生成嗎？	是；它本身就是語言模型	基本視窗訓練不是有順序的生成器
訓練後重用什麼？	整個網路；詞向量只是副產品	學習出的向量表

CBOW 中的 **continuous** 代表內部表示具有實數座標，不代表文字本身連續、詞之間沒有邊界，或每種詞義都各有向量。本課最後會連結計數式與預測式構造，並指出各自保留的資訊。計數式語言模型及其評估則留到 NLP 05。

詞彙語意：詞義與關係

白話版。一個寫出來的詞可能有好幾個意義，而意義之間有結構化的關係：有些幾乎等價，有些彼此相反，有些是另一個的一種。嵌入稍後會把其中好幾種關係混在一起，所以先替它們命名。

詞形、詞元與詞義

詞形（word form）是文字中實際出現的樣子（`banks`、`banked`）；**詞元（lemma）**是把各種屈折形式歸在一起的辭典條目（`bank`）。**詞義（sense）**是詞元的一個意義。`bank` 至少有兩個彼此無關的詞義：河邊的斜坡與金融機構，這是**同形異義（homonymy）**。`window` 的詞義彼此相關：牆上的開口與螢幕上的區域，這是一**詞多義（polysemy）**。NLP 02 討論上下文與組合如何選出詞義。對表示而言，關鍵在於詞彙表項目是詞形或詞元，不是詞義：每個項目只有一列的表，必然把該項目的所有詞義合併在一起。

詞義之間的關係

詞彙關係存在於詞義之間，而不是拼寫字串之間：

- **同義（synonymy）**：兩個詞義在許多上下文中可以互換，意義改變很小，例如 `slope` 與 `incline`。完全同義很少見；語域、搭配與隱含意義通常不同。
- **反義（antonymy）**：沿著某個尺度或維度相反的詞義，例如 `hot` 與 `cold`。反義詞在其他方面非常相似：描述同一種屬性，出現在同類句子中。
- **下位與上位關係（hyponymy、hypernymy）**：一種「是一種」關係。`poodle` 是 `dog` 的下位詞，`dog` 是 `poodle` 的上位詞。這種關係不對稱，通常具遞移性，形成階層。
- **部分整體關係（meronymy）**：部分與整體，例如 `wheel` 與 `car`。

另有兩個概念對評估很重要。**相似度（similarity）**問兩個概念是否共享屬性（`cup` 與 `mug`）；**相關性（relatedness）**或聯想（association）問它們是否常一起出現（`cup` 與 `coffee`）。相關的一對詞不一定相似。

WordNet：人工建立的詞彙資料庫

WordNet 把英文名詞、動詞、形容詞與副詞組織成**同義詞集（synset）**：表達同一概念的一組同義詞元，各附一段簡短釋義（gloss），並以上位、部分整體與反義等關係相連 [4]。查詢一個詞元會傳回它的各個同義詞集，每個詞義一個。沿著上位連結往上走，會到達越來越一般的概念，讓程式能量測兩個詞義在階層中相距多遠。

下列範例以相同風格、用手工建立一個迷你資料庫，不必下載 WordNet 本身也能展示運作方式。其中條目都是本課自行編寫的。

```
# A hand-built miniature lexical database in the style of WordNet (not WordNet data).
synsets = {
    "bank.n.slope":   {"lemmas": ["bank"], "gloss": "sloping land beside a river",
     ↪  "hypernym": "slope.n"},
    "bank.n.finance": {"lemmas": ["bank"], "gloss": "an institution that holds deposits",
     ↪  "hypernym": "institution.n"},
    "slope.n":        {"lemmas": ["slope", "incline"], "gloss": "a surface at an angle",
     ↪  "hypernym": "landform.n"},
    "landform.n":     {"lemmas": ["landform"], "gloss": "a natural feature of land",
     ↪  "hypernym": "entity.n"},
    "institution.n":  {"lemmas": ["institution"], "gloss": "an organization with a public
     ↪  role", "hypernym": "organization.n"},
    "school.n":       {"lemmas": ["school"], "gloss": "an institution for teaching",
     ↪  "hypernym": "institution.n"},
    "organization.n": {"lemmas": ["organization"], "gloss": "a group with a purpose",
     ↪  "hypernym": "entity.n"},
    "entity.n":       {"lemmas": ["entity"], "gloss": "anything that exists", "hypernym":
     ↪  None},
}

def senses(word):
    return [s for s, v in synsets.items() if word in v["lemmas"]]

def path_to_root(s):
    path = [s]
    while synsets[path[-1]]["hypernym"]:
        path.append(synsets[path[-1]]["hypernym"])
    return path

def path_similarity(a, b):            # 1 / (1 + edges on the shortest is-a path)
    pa, pb = path_to_root(a), path_to_root(b)
    common = next(s for s in pa if s in pb)
    return 1 / (1 + pa.index(common) + pb.index(common))

print("senses of bank:", senses("bank"))
print("synonyms in slope.n:", synsets["slope.n"]["lemmas"])
print("hypernym chain:", " -> ".join(path_to_root("bank.n.finance")))
for other in ["school.n", "slope.n"]:
    print(f"path_similarity(bank.n.finance, {other}) = {path_similarity('bank.n.finance',
     ↪  other):.3f}")
```

執行會印出：

```
senses of bank: ['bank.n.slope', 'bank.n.finance']
synonyms in slope.n: ['slope', 'incline']
hypernym chain: bank.n.finance -> institution.n -> organization.n -> entity.n
path_similarity(bank.n.finance, school.n) = 0.333
path_similarity(bank.n.finance, slope.n) = 0.167
```

`bank` 的金融詞義比較接近 `school`（兩者都是機構），而不是 `slope`，即使 `bank` 的另一個詞義正是一種斜坡。詞義清單讓這個區別明確；字串 `bank` 的單一向量做不到。

人工資源精確且可解讀，但建置成本高，對某些語言與領域的涵蓋遠優於其他，跟不上新詞與新詞義，而且只編碼一套詞義劃分，未必符合應用需求。分布式向量做的是相反的取捨：便宜、涵蓋廣，但關係是隱含且混雜的。許多實務工作會結合兩者，例如用詞彙資源檢查嵌入的近鄰。

分布式證據：上下文是觀察

白話版。出現在相似上下文的詞，可能得到相似向量。上下文提供用法的證據，不會直接提供定義或真值條件。

Harris 主張意義的差異與分布的差異相關，也就是與語言單位出現的環境相關 [5]。這個**分布假說（distributional hypothesis）**是本課其餘所有方法的基礎。

詞類型（word type）是詞彙表項目；**詞出現（word token）**是一次實際出現。靜態嵌入把許多次出現的證據彙整成該類型的一個向量。`bank` 在 `river` 旁與在 `account` 旁出現，都會影響同一列。模型可以學到這個詞的混合用法，但查表操作無法為每種詞義選不同列。

收集統計之前先定義上下文。半徑二的詞視窗包含中心詞左右各最多兩個 token，排除中心詞。較短視窗著重緊鄰位置；較大視窗納入較遠的主題證據。若方向重要，可以把左側與右側位置當成不同特徵。本課刻意合併兩側，讓模型的不變性明確呈現。

分布假設（distributional assumption）實用之處，在於文字提供重複觀察，不必替每個詞標註語意類別。它並不完整，因為共享用法包含同義、反義、主題關聯與語法相容性。`hot` 與 `cold` 都可能出現在 `water` 前；基於這種證據的相似度，無法確立兩者表達相同屬性。用上一節的術語來說，分布式相似度混合了同義、反義、同位下位詞與相關性。Turney 與 Pantel 按照矩陣所表示的證據，整理這些向量空間方法 [6]。

一起記錄語料、tokenizer 與視窗規格。改變斷詞方式，就改變哪些類型被計算上下文。從技術事故報告學出的嵌入，可能以不同於電子設備手冊的方式組織 `charge`。座標是這些選擇的結果，不是與語言無關的詞義測量。

計數式嵌入：檢查共享上下文的假設

白話版。計算每個詞與各種上下文一起出現的頻率，再與假設兩者獨立的基準比較。特別強的關聯會形成正向座標。

Turney 與 Pantel 的分布式表示綜述 [6] 說明這種**計數式**途徑。令 C_{wc} 計算中心詞 w 與鄰近上下文詞 c 所形成之有序配對的觀察次數。即使兩種角色的詞彙表包含相同詞，角色仍不同。從表格

定義經驗機率：

$$N_C = \sum_{w,c} C_{wc}, \qquad P(w,c) = \frac{C_{wc}}{N_C}, \qquad P(w) = \sum_{c} P(w,c), \qquad P(c) = \sum_{w} P(w,c).$$

其中 N_C 是中心與上下文的總配對數，C_{wc} 是一個儲存格的計數，$P(w,c)$ 是經驗聯合機率，$P(w)$ 是中心詞邊際機率，$P(c)$ 是上下文邊際機率。加總遍歷對應詞彙表。一個 token 可能貢獻多個配對，因此 N_C 不是 token 數。

逐點互資訊（pointwise mutual information，PMI）比較經驗聯合機率與獨立基準。**正向 PMI（positive PMI，PPMI）**只保留強於基準的關聯：

$$\mathrm{PMI}(w,c) = \log \frac{P(w,c)}{P(w)P(c)}, \qquad \mathrm{PPMI}(w,c) = \max(\mathrm{PMI}(w,c), 0).$$

其中 $\mathrm{PMI}(w,c)$ 是一個配對的對數關聯分數，$\mathrm{PPMI}(w,c)$ 是其非負版本，log 是自然對數。機率定義如上。聯合計數為零且邊際機率為正時，PMI 趨近負無限大，PPMI 為零；整列完全未觀察到則需要另外處理。

高頻上下文不會自動得到很大的關聯分數。它必須比整體頻率所預期的更常出現在這個中心詞附近。相反地，罕見配對可能只靠很少證據就得到高 PMI。應同時檢查計數與分數；移除負關聯、只保留正關聯，也是在捨棄資訊，不會使剩下的每個關聯都可靠。

PPMI 矩陣仍為每個上下文保留一個座標。截斷**奇異值分解（singular value decomposition，SVD）**可將它壓縮：

$$M = A\Sigma B^{\top}, \qquad M_D = A_D \Sigma_D B_D^{\top}, \qquad E = A_D \Sigma_D.$$

其中 M 是 PPMI 矩陣，A 與 B 包含左右奇異向量，Σ 是奇異值對角矩陣，D 是保留維度，下標 D 取前 D 個奇異成分。M_D 是低秩重建，E 提供下方使用的中心詞向量。這是程式採用的慣例；以其他方式分配奇異值縮放，會產生不同的向量幾何。

SVD 找出列之間共享的模式，不會替每個座標指定語意名稱。若原矩陣的秩高於 D，壓縮無法保留每個儲存格。下列合成語料讓我們同時檢查構造方式，以及將餘弦相似度解讀為同義關係的失敗案例。

```
import numpy as np

sentences = """the cat drinks milk . the dog drinks water . a cat chases a mouse .
a dog chases a cat . the kitten drinks milk . the puppy drinks water .
a kitten chases a ball . a puppy chases a ball . the car needs fuel .
the truck needs fuel . a car drives on the road . a truck drives on the road .""".split(".")
tokens = [s.split() for s in sentences if s.strip()]
vocab = sorted({w for s in tokens for w in s})
idx = {w: i for i, w in enumerate(vocab)}

# 1) Co-occurrence counts within a +-2 word window.
C = np.zeros((len(vocab), len(vocab)))
for s in tokens:
    for i, w in enumerate(s):
        for j in range(max(0, i - 2), min(len(s), i + 3)):
            if j != i:
                C[idx[w], idx[s[j]]] += 1
```

```
# 2) Positive pointwise mutual information.
total = C.sum()
pw, pc = C.sum(1, keepdims=True) / total, C.sum(0, keepdims=True) / total
with np.errstate(divide="ignore"):
    pmi = np.log((C / total) / (pw * pc))
ppmi = np.maximum(pmi, 0)

# 3) Truncated SVD gives dense 8-dimensional embeddings.
U, S, _ = np.linalg.svd(ppmi)
E = U[:, :8] * S[:8]
E /= np.linalg.norm(E, axis=1, keepdims=True)

def nearest(word, k=3):
    sims = E @ E[idx[word]]
    order = [vocab[i] for i in np.argsort(-sims) if vocab[i] != word]
    return [(w, round(float(sims[idx[w]]), 2)) for w in order[:k]]

for w in ["cat", "puppy", "truck", "milk"]:
    print(w, "->", nearest(w))
```

輸出：

```
cat -> [('kitten', 0.96), ('puppy', 0.88), ('dog', 0.88)]
puppy -> [('dog', 1.0), ('kitten', 0.98), ('cat', 0.88)]
truck -> [('car', 1.0), ('needs', 0.6), ('the', 0.47)]
milk -> [('water', 0.97), ('a', 0.62), ('chases', 0.51)]
```

只用十二個句子、沒有任何標籤，動物就和動物聚在一起，車輛和車輛聚在一起。注意「milk」與「water」幾乎相同：在這個語料中兩者都是被喝的東西。嵌入相似度量的是**共享的上下文**，這通常、但不總是對應到相似的意思；例如「熱」與「冷」這類反義詞也會共享上下文。

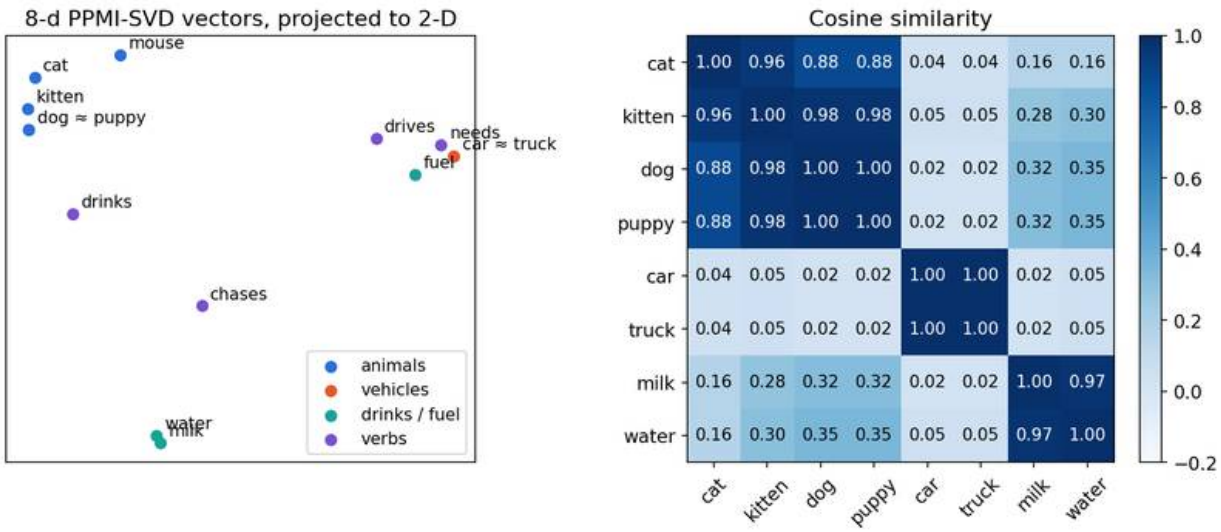


圖 *1* — 上方程式碼得到的向量。共享上下文的詞群組內，餘弦相似度很高。

Word2Vec：藉由預測鄰近詞學習詞向量

白話版。出現在相似上下文的詞，常具有相似用途。Word2Vec 把這個觀察變成小型預測任務：從鄰近詞猜中心詞，或從中心詞猜鄰近詞，並學習有助於預測的向量。向量是學習出的參數，不是人工列出的詞義清單。兩種基本架構是**連續詞袋模型（continuous bag of words，CBOW）**與 **Skip-gram** [7]。

把句子轉成訓練樣本

從原創句子 `small red robots move slowly` 開始。**上下文視窗（context window）**選出鄰近位置；半徑一代表中心詞前後各一個詞。此例只使用句子內部的中心位置，因此每個視窗都有兩個鄰近詞：

中心詞	上下文	CBOW 訓練樣本	Skip-gram 訓練樣本
`red`	`small`、`robots`	`[small, robots] → red`	`red → small`；`red → robots`
`robots`	`red`、`move`	`[red, move] → robots`	`robots → red`；`robots → move`
`move`	`robots`、`slowly`	`[robots, slowly] → move`	`move → robots`；`move → slowly`

CBOW 結合多個上下文詞來預測一個中心詞；Skip-gram 用一個中心詞，分別預測各個上下文詞。 Skip-gram 並不產生有順序的句子；「skip」也不代表隨機刪詞。目標可以位於中心詞任一側。實作可在句子邊界使用較短視窗；這裡的程式略過那些中心位置，讓張量形狀保持簡單。

訓練標籤直接來自文字本身。在建立樣本之前，不必有人替 `robots` 標註語意類別。重複出現就提供重複證據。視窗決定模型被鼓勵捕捉的關係：即使詞彙表不變，改變視窗仍會改變訓練任務。

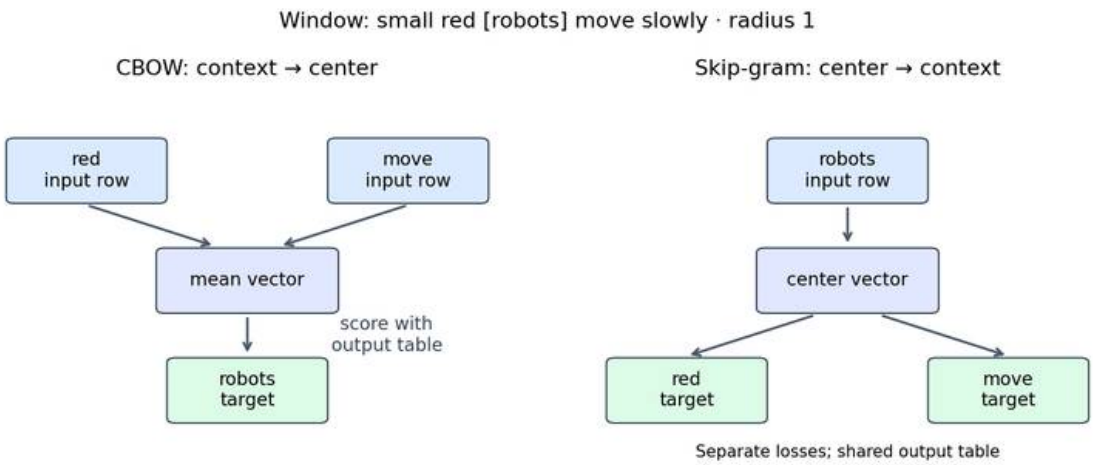


圖 2 — 同一句子形成方向相反的預測任務。兩種架構都學習輸入與輸出嵌入表；圖中的平均運算只出現在 *CBOW* 路徑。

兩個嵌入表與 softmax

一個詞作為輸入時需要向量，作為預測候選時也需要向量。使用兩個學習表，而不是先假定這兩種角色共享權重：

$$E, U \in \mathbb{R}^{V \times D}, \qquad \mathbf{e}_w = E_{w,:}^{\top}, \qquad \mathbf{u}_w = U_{w,:}^{\top}, \qquad p(o \mid \mathbf{h}) = \frac{\exp(\mathbf{u}_o^{\top}\mathbf{h})}{\sum_{v=1}^{V} \exp(\mathbf{u}_v^{\top}\mathbf{h})}.$$

其中 E 是輸入嵌入表，U 是輸出嵌入表，V 是詞彙表大小，D 是向量維度，w 是詞的 ID，冒號選取該列的所有座標。轉置把 $\mathbf{e}_w$ 與 $\mathbf{u}_w$ 變成直向行向量。$\mathbf{h} \in \mathbb{R}^D$ 表示輸入，o 是目標詞 ID，v 索引候選詞，$p(o \mid \mathbf{h})$ 是目標的 softmax 機率。每個內積都是預測分數；分母使整個詞彙表的機率總和為一。

CBOW 平均上下文向量來形成輸入。損失要求模型替觀察到的中心詞配置較高機率：

$$\mathbf{h}_t = \frac{1}{|C_t|} \sum_{j \in C_t} \mathbf{e}_{w_j}, \qquad \mathcal{L}_{\text{CBOW}} = -\frac{1}{|\mathcal{T}|} \sum_{t \in \mathcal{T}} \log p(w_t \mid \mathbf{h}_t).$$

其中 w_t 是位置 t 的中心詞，C_t 是選出的鄰近位置集合且不包含 t，$|C_t|$ 是集合大小，$\mathbf{h}_t$ 是這些詞向量的平均，$\mathcal{T}$ 包含有效中心位置，$\mathcal{L}_{\text{CBOW}}$ 是中心詞負對數機率的平均。平均不保留上下文順序：交換左右鄰近詞會產生相同輸入。有些寫法加總上下文向量；本課與程式使用平均。

Skip-gram 則查出中心詞向量，再用相同預測規則處理每個鄰近詞：

$$\mathcal{L}_{\text{SG}} = -\frac{1}{|\mathcal{D}|} \sum_{(t,j) \in \mathcal{D}} \log p(w_j \mid \mathbf{e}_{w_t}).$$

其中 $\mathcal{L}_{\text{SG}}$ 是 Skip-gram 損失，$\mathcal{D}$ 是觀察到的中心與上下文位置配對，$|\mathcal{D}|$ 是配對數，t 索引中心位置，j 索引一個鄰近位置，w_t 是輸入詞，w_j 是目標詞。即使詞 ID 重複，每次出現仍保留為獨立樣本。

一次訓練步驟會相對於其他候選，提高觀察到的目標分數，同時改變選出的輸入向量與輸出向量。許多樣本重複使用同一個輸入列，迫使模型整合該詞在不同上下文出現的證據。這種共享預測任務，就是從離散詞 ID 學到連續特徵的途徑。這兩種基本架構都不需要堆疊很深的隱藏層，這是它們與 NLP 03 神經語言模型在計算上的主要差別。

負取樣：比較觀察配對與隨機抽樣

完整 softmax 對每次預測都計算詞彙表中每個詞的分數。**負取樣（negative sampling）**將這個跨詞彙表的分類，改成較小的二元任務：區分實際觀察到的中心與上下文配對，以及隨機抽取目標詞形成的配對。Word2Vec 的後續論文提出了這個方法 [8]。

對觀察到的輸入詞 c 與上下文詞 o，一次取樣的損失為：

$$\mathcal{L}_{\text{neg}}(c, o) = -\log \sigma(\mathbf{u}_o^\top \mathbf{e}_c) - \sum_{k=1}^{K} \log \sigma(-\mathbf{u}_{n_k}^\top \mathbf{e}_c), \qquad n_k \sim q, \qquad \sigma(a) = \frac{1}{1+\exp(-a)}.$$

其中 $\mathcal{L}_{\text{neg}}$ 是一組觀察配對的負取樣損失，c 是中心詞 ID，o 是觀察到的上下文詞 ID，$\mathbf{e}_c$ 是中心的輸入向量，$\mathbf{u}_o$ 與 $\mathbf{u}_{n_k}$ 是輸出向量，K 是指定的隨機目標數，n_k 是分布 q 的第 k 次抽樣，σ 是作用於實數分數 a 的 sigmoid。第一項鼓勵觀察配對具有高分；其餘項鼓勵隨機配對具有低分。完整訓練目標對觀察配對與取樣隨機性取平均。若使用 CBOW 版本，可用上下文平均向量取代 $\mathbf{e}_c$。

原論文使用的一種常見取樣分布為：

$$q(w) = \frac{f(w)^{3/4}}{\sum_{v=1}^{V} f(v)^{3/4}}.$$

其中 $q(w)$ 是抽出詞彙項目 w 的機率，$f(w)$ 是它在語料中的出現次數，v 索引詞彙表項目，V 是詞彙表大小。相較於直接依次數取樣，這個指數讓頻率分布較平坦。它是設計選擇，不是普遍的機率定律。

對觀察配對 `robots → move`，隨機抽樣可能產生 `robots → red`。「負」標籤的意思是**由雜訊程序抽出**，並不宣稱兩個詞永遠不會共同出現。抽樣結果可能是其他位置的真實上下文，甚至與這次的目標相同。實作必須說明是否保留、略過或重新抽取這類碰撞；改變規則也會改變取樣目標。模型區分的是配對的統計來源，不是在建立絕對不相容的詞典。

二元 sigmoid 分數在詞彙表上的總和不等於一。負取樣是另一種訓練目標，不是上面 softmax 的精確計算；其損失也不是正規化的下一詞概似。它用來學習實用表示，而表示仍應依預定的下游任務評估。

預測為何會改變向量幾何

「從鄰近詞學習」可以直接從梯度看懂。對一個使用完整 softmax 的目標詞，其分數是輸出向量與輸入表示的內積：

$$s_v = \mathbf{u}_v^\top \mathbf{h}, \qquad \ell = -\log p(o \mid \mathbf{h}), \qquad \frac{\partial \ell}{\partial s_v} = p(v \mid \mathbf{h}) - \mathbf{1}[v = o], \qquad \nabla_{\mathbf{h}} \ell = \sum_{v=1}^{V} p(v \mid \mathbf{h}) \mathbf{u}_v - \mathbf{u}_o.$$

其中 s_v 是候選詞 v 的分數，ℓ 是觀察到目標詞 o 時的損失，$\mathbf{1}[v = o]$ 在候選詞等於目標詞時為一，否則為零；$\nabla_{\mathbf{h}} \ell$ 是對輸入表示的梯度。嵌入表、詞彙表大小 V 與條件機率皆沿用前面的定義。

梯度下降會讓輸入表示靠近觀察到的輸出向量，同時遠離模型依機率加權的其他候選向量平均。某個詞每次出現，都會透過同一列累積更新。因此，上下文分布相似的詞可能學到相近的預測角色，過程中完全不需要標註它們是不是同義詞。

對使用平均的 CBOW，每個選中的上下文向量列都收到 $\nabla_{\mathbf{h}} \ell / |C_t|$。如果兩組上下文只是相同詞的不同排列，輸入表示就無法區分它們。Skip-gram 則讓一個中心詞向量接受多個鄰詞目標的更新；它的基本分數沒有欄位記錄鄰詞位於左邊還是右邊。因此，選擇視窗同時決定目標函數可以直接表達哪些關係；擴大視窗並不只是讓小視窗變得更準確。

負取樣分數究竟估計什麼

先暫時不限制嵌入維度，就能看清楚取樣目標。固定一個中心詞，假設每個候選詞都有可以獨立調整的分數。以 p_o 表示它的經驗上下文機率、q_o 表示雜訊機率，該候選詞對期望損失的貢獻為：

$$\ell_o(s_o) = -p_o \log \sigma(s_o) - K q_o \log \sigma(-s_o), \qquad \frac{d\ell_o}{ds_o} = (p_o + K q_o)\sigma(s_o) - p_o.$$

其中 o 是一個候選上下文詞，s_o 是不受限制的實數分數，p_o 與 q_o 是固定中心詞下的兩種機率，K 是每個觀察詞對所搭配的獨立雜訊取樣次數，σ 是前面定義的 sigmoid。本推導保留雜訊取樣恰好抽到觀察目標的碰撞情況。

當兩種機率都大於零時，駐點分數為：

$$s_o^{*} = \log \frac{p_o}{K q_o}.$$

其中 s_o^{*} 使該候選詞的期望損失最小。若 $p_o = 0.5$、$q_o = 0.25$、$K = 2$，最佳分數就是零：在相關的機率質量中，兩種統計來源各占一半。如果只將 q_o 改成 0.05，最佳分數就變成 $\log 5 \approx 1.609$。觀察到的上下文頻率沒有改變，雜訊分布卻改變了分數。因此，正分數表示「比這個取樣基準更常共同出現」，並不表示「這個詞對一定有意義」。若觀察機率為零、雜訊機率為正，則不受限制的最佳分數趨向負無窮。

如果雜訊分布恰好等於語料的上下文邊際分布，這個比值就得到 PMI 減去 $\log K$。Levy 與 Goldberg 分析了由此產生的矩陣分解觀點 [9]；常用的四分之三次方取樣分布則會改變這個比值。實際嵌入模型讓分數共用低維向量，通常無法同時達到每個獨立最佳值。因此，向量表是在秩限制下折衷不同詞對的統計關係。這既說明它與計數式方法的連結，也解釋了截斷 PPMI 矩陣與訓練 Word2Vec 為何不一定得到相同向量。

完整 softmax 每個目標的運算量約為 $O(VD)$；計算一個觀察目標與 K 個取樣詞的分數則約為 $O((K+1)D)$，尚未計入取樣與批次處理成本。其中 D 是嵌入維度。計算量降低的同時，目標函數也改變了。sigmoid 值與負取樣損失都不能代替計算困惑度時所需的正規化詞元機率。

在 CPU 訓練 CBOW 與 Skip-gram

這個獨立範例使用完整 softmax 學習輸入與輸出表。詞彙表很小，因此可以直接檢查目標，不需要取樣近似。兩個模型共用相同的原創語料、半徑一的內部視窗、嵌入寬度與訓練迴圈。Skip-gram 路徑只有一個輸入詞，取平均就是中心向量本身。

```
import torch
from torch import nn
from torch.nn import functional as F

torch.manual_seed(7)
torch.set_num_threads(1)
corpus = [
    'small red robots move slowly',
    'small blue robots move quickly',
    'large red sensors work slowly',
```

```
    'large blue sensors work quickly',
    'small red sensors work quietly',
    'large blue robots move quietly',
]
vocab = sorted(set(' '.join(corpus).split()))
ids = {word: i for i, word in enumerate(vocab)}
cbow_x, cbow_y, sg_x, sg_y = [], [], [], []
for sentence in corpus:
    tokens = [ids[word] for word in sentence.split()]
    # Use radius 1 and interior centers, so every context has two words.
    for t in range(1, len(tokens) - 1):
        neighbors = [tokens[t - 1], tokens[t + 1]]
        cbow_x.append(neighbors)
        cbow_y.append(tokens[t])
        for neighbor in neighbors:
            sg_x.append([tokens[t]])
            sg_y.append(neighbor)

class Word2Vec(nn.Module):
    def __init__(self, size, width=8):
        super().__init__()
        self.input = nn.Embedding(size, width)
        self.output = nn.Embedding(size, width)
        nn.init.normal_(self.input.weight, std=0.1)
        nn.init.normal_(self.output.weight, std=0.1)

    def forward(self, context):
        h = self.input(context).mean(dim=1)
        return h @ self.output.weight.T

embeddings = {}
for name, inputs, targets in [('CBOW', cbow_x, cbow_y),
                              ('Skip-gram', sg_x, sg_y)]:
    x, y = torch.tensor(inputs), torch.tensor(targets)
    model = Word2Vec(len(vocab))
    optimizer = torch.optim.SGD(model.parameters(), lr=1.0)
    initial = F.cross_entropy(model(x), y).item()
    for step in range(250):
        optimizer.zero_grad()
        loss = F.cross_entropy(model(x), y)
        loss.backward()
        optimizer.step()
    final = F.cross_entropy(model(x), y).item()
    assert final < initial
    embeddings[name] = model.input.weight.detach().clone()
    print(name, len(y), f'{initial:.3f} -> {final:.3f}')
print('vocabulary', len(vocab), 'embedding shape', tuple(embeddings['CBOW'].shape))
```

執行會印出 `CBOW 18 2.384 -> 0.159`、`Skip-gram 36 2.402 -> 1.198` 與 `vocabulary 11 embedding shape (11, 8)`。損失是平均負對數機率，以 nat 為單位。斷言確認實作能擬合這些訓練樣本。CBOW 損失較低，不表示它學出的詞向量較好：它的目標與條件輸入和 Skip-gram 不同。這個語料太小，無法判斷語意鄰近關係或對自然語言的泛化能力。

訓練後，`embeddings['CBOW'][ids['robots']]` 可取出一個學到的輸入向量。訓練需要輸出

表，但若下游模型只查詢輸入向量，提供服務時可省略輸出表。應記錄保留哪個表；使用輸入向量、輸出向量或兩者組合，都是表示設計的選擇。

僅線上版：*word2vec — Efficient Estimation of Word Representations in Vector Space* — *https://blog.shyandram.dev/zh-tw/learn/nlp/lexical-semantics-word-embeddings/papers/word2vec–efficient-estimation-of-word-representations-in-vector-space/*

GloVe：直接擬合共現次數的對數

白話版。 GloVe 不是一次處理一個視窗、逐一預測，而是先計算每一對詞共同出現多少次，再擬合詞向量，讓它們的內積重現這些次數的對數。高頻配對得到較大權重，但權重有上限。

Pennington 等人以共現機率的比值說明 GloVe 的動機 [10]。令 X_{ik} 為詞 k 出現在詞 i 上下文中的（依距離加權的）次數，$P(k \mid i) = X_{ik} / \sum_{k'} X_{ik'}$。對兩個詞 i 與 j，比值 $P(k \mid i)/P(k \mid j)$ 在上下文詞 k 與 i 相關而與 j 無關時很大，反之很小；若 k 與兩者同樣相關或同樣無關，比值接近一。比值比任一個機率單獨更有區辨力。模型要求向量差編碼這類比值，最後得到對共現矩陣的加權最小平方目標：

$$J = \sum_{i,j=1}^{V} f\left(X_{ij}\right)\left(\mathbf{w}_i^{\top}\tilde{\mathbf{w}}_j + b_i + \tilde{b}_j - \log X_{ij}\right)^2, \qquad f(x) = \begin{cases} (x/x_{\max})^{\alpha} & x < x_{\max}, \\ 1 & \text{otherwise.} \end{cases}$$

其中 V 是詞彙表大小，X_{ij} 是詞 i 與 j 的共現次數，$\mathbf{w}_i \in \mathbb{R}^D$ 是 i 的詞向量，$\tilde{\mathbf{w}}_j \in \mathbb{R}^D$ 是 j 的另一個上下文向量，b_i 與 $\tilde{b}_j$ 是純量偏差，f 是加權函數，$x_{\max}$ 與 α 是它的截斷值與指數（式中 otherwise 意為「其他情況」）。因為 $f(0) = 0$，次數為零的儲存格完全不貢獻，所以永遠不需要 $\log 0$，加總實際上只涵蓋觀察到的配對。論文固定 $x_{\max} = 100$、$\alpha = 3/4$，使用相距 d 個詞的配對貢獻 $1/d$ 的共現加權，並以兩個表的和 $W + \tilde{W}$ 作為最終詞向量 [10]。

GloVe 與 Word2Vec 有三個設計差異。它以全域計數而不是個別視窗訓練，因此掃過語料一次建立 X，之後訓練時間與非零儲存格數成正比。偏差吸收詞頻，內積只需建模剩下的部分。加權函數則避免 `the` 這類極高頻配對主導，同時讓罕見而雜訊大的配對影響很小。與負取樣一樣，這個目標在秩限制下分解一個平移過的對數次數矩陣，這也是 GloVe 與 Skip-gram 向量實務上常表現相近的原因。

下列程式在先前的玩具語料上計算這些材料：共現比值與加權函數。

```
import numpy as np

text = """the cat drinks milk . the dog drinks water . a cat chases a mouse .
a dog chases a cat . the kitten drinks milk . the puppy drinks water .
a kitten chases a ball . a puppy chases a ball . the car needs fuel .
the truck needs fuel . a car drives on the road . a truck drives on the road ."""
sents = [s.split() for s in text.split(".") if s.strip()]
vocab = sorted({w for s in sents for w in s})
idx = {w: i for i, w in enumerate(vocab)}

X = np.zeros((len(vocab), len(vocab)))           # +-2 window, 1/distance weighting
for s in sents:
    for i, w in enumerate(s):
```

```
        for j in range(max(0, i - 2), min(len(s), i + 3)):
            if j != i:
                X[idx[w], idx[s[j]]] += 1.0 / abs(i - j)
P = X / X.sum(1, keepdims=True)                  # P(k | i): row-normalized counts

for k in ["drinks", "chases", "needs", "the"]:
    a, b = P[idx["kitten"], idx[k]], P[idx["truck"], idx[k]]
    ratio = "inf" if b == 0 else f"{a / b:.2f}"
    print(f"P({k}|kitten)={a:.3f}  P({k}|truck)={b:.3f}  ratio={ratio}")

def f(x, x_max=100.0, alpha=0.75):                # GloVe weighting function
    return np.minimum((x / x_max) ** alpha, 1.0)
print("f at 1, 10, 100, 1000:", np.round(f(np.array([1.0, 10.0, 100.0, 1000.0])), 3))
print("nonzero cells:", int((X > 0).sum()), "of", X.size)
```

執行會印出：

```
P(drinks|kitten)=0.200  P(drinks|truck)=0.000  ratio=inf
P(chases|kitten)=0.200  P(chases|truck)=0.000  ratio=inf
P(needs|kitten)=0.000  P(needs|truck)=0.200  ratio=0.00
P(the|kitten)=0.200  P(the|truck)=0.200  ratio=1.00
f at 1, 10, 100, 1000: [0.032 0.178 1.    1.   ]
nonzero cells: 96 of 361
```

比值把動物上下文與車輛上下文分開，而 `the` 的比值為一，因此不提供資訊。輸出也暴露這個小語料的限制：矩陣將近四分之三是零，而 GloVe 忽略零值儲存格。觀察到的配對這麼少時，只有幾百個參數的模型就能精確擬合每個觀察到的儲存格，得到的幾何主要取決於初始化。GloVe 的計數式設計需要矩陣足夠充實的大型語料；因此這裡不訓練它。練習 3 會請你親自觀察這種欠定（underdetermination）現象。

僅線上版：*GloVe — Global Vectors for Word Representation* — *https://blog.shyandram.dev/zh-tw/learn/nlp/lexical-semantics-word-embeddings/papers/glove–global-vectors-for-word-representation/*

fastText：由字元 n-gram 組成向量

白話版。詞級的表沒有看過的詞就沒有向量，而且把 `charger` 與 `chargers` 當成無關的兩列。fastText 用字元片段的向量組成每個詞的向量，因此拼寫相關的詞共享參數，沒見過的詞也能得到向量。

Bojanowski 等人擴充使用負取樣的 Skip-gram，把每個詞表示為一袋字元 n-gram [11]。詞的前後加上邊界符號 `<` 與 `>`，讓前綴與後綴能和詞中間相同的字母區分開來，並把整個加上邊界的詞另外加入為一個單位。論文使用所有 $3 \le n \le 6$ 的 n-gram。中心詞 w 與上下文詞 c 的分數變成

$$s(w, c) = \sum_{g \in \mathcal{G}_w} \mathbf{z}_g^\top \mathbf{v}_c .$$

其中 $\mathcal{G}_w$ 是 w 的 n-gram 集合（包含加上邊界的整個詞），$\mathbf{z}_g \in \mathbb{R}^D$ 是 n-gram g 的學習向量，$\mathbf{v}_c \in \mathbb{R}^D$ 是 c 的上下文向量。詞本身的向量是 $\sum_{g \in \mathcal{G}_w} \mathbf{z}_g$。這個分數取代上面負取樣損失中的

$\mathbf{u}_o^\top \mathbf{e}_c$。為了限制記憶體，論文以雜湊函數把 n-gram 映射到固定數量的桶，因此無關的 n-gram 可能共用同一個向量 [11]。

```
import numpy as np

def char_ngrams(word, n_min=3, n_max=6):
    w = f"<{word}>"                       # boundary symbols mark prefixes and suffixes
    grams = {w[i:i + n] for n in range(n_min, n_max + 1) for i in range(len(w) - n + 1)}
    return grams | {w}                    # the whole word is kept as one extra unit

print(sorted(char_ngrams("robot", 3, 3)))
vocab = ["charge", "charger", "recharge", "discharge", "robot", "robots"]
query = "chargers"                        # not in the vocabulary
q = char_ngrams(query)
for word in vocab:
    print(f"{word:10s} shares {len(q & char_ngrams(word)):2d} of {len(q)} units with
     ↪  '{query}'")

# A fastText-style vector is the sum of its unit vectors, so an unseen word still gets one.
rng = np.random.default_rng(0)
table = {}                                # stands in for a trained (hashed) n-gram table
def vector(word, dim=8):
    return sum(table.setdefault(g, rng.normal(size=dim)) for g in sorted(char_ngrams(word)))
def cos(a, b):
    return a @ b / (np.linalg.norm(a) * np.linalg.norm(b))
for word in vocab:
    vector(word)
print("cos(chargers, charger) =", round(cos(vector("chargers"), vector("charger")), 3))
print("cos(chargers, robots)  =", round(cos(vector("chargers"), vector("robots")), 3))
```

執行會印出：

```
['<ro', '<robot>', 'bot', 'obo', 'ot>', 'rob']
charge     shares 14 of 27 units with 'chargers'
charger    shares 18 of 27 units with 'chargers'
recharge   shares 10 of 27 units with 'chargers'
discharge  shares 10 of 27 units with 'chargers'
robot      shares  0 of 27 units with 'chargers'
robots     shares  0 of 27 units with 'chargers'
cos(chargers, charger) = 0.804
cos(chargers, robots)  = -0.222
```

這裡的 n-gram 向量是隨機的，沒有經過訓練，所以 `chargers` 與 `charger` 的高餘弦值完全來自它們共享的單位。這正是機制所在：沒看過的屈折形式，向量大多繼承自已知的片段。共享是否有幫助，由訓練決定。對屈折與衍生變化通常有幫助；對只是碰巧共享字母的詞則會誤導；`discharge` 與 `recharge` 都含 `charge`，但意義差異無法從片段看出。子詞向量對構詞豐富的語言與罕見詞最有幫助。也請注意它與 NLP 02 子詞 tokenizer 的關係：BPE 在查表前把詞切成一串片段，fastText 則把所有重疊 n-gram 的向量加總。

稠密詞與文件表示

白話版。稠密向量讓不同詞共享統計資訊。它能連結相關片語，但向量幾何反映訓練資料與目標，不是通用的意義定義。

靜態向量可作為分類器的精簡可重用特徵、探索詞的近鄰，或建立平均池化的文件基準。當有標籤的應用資料不多，但有適合的文字語料可用時，它們很實用。嵌入訓練與下游分類器訓練是不同階段：預測鄰近詞不會直接學到 billing 標籤或拒答政策。

靜態嵌入表為每個詞彙項目配置向量：

$$\mathbf{e}_t = E_{i_t,:}, \qquad \mathbf{h}(d) = \frac{1}{T}\sum_{t=1}^{T} \mathbf{e}_t.$$

其中 $E \in \mathbb{R}^{V \times D}$ 是嵌入表，V 是詞彙表大小，D 是嵌入維度，i_t 是位置 t 的 token ID，$\mathbf{e}_t$ 是選出的橫向列向量，T 是文件的 token 長度，$\mathbf{h}(d)$ 是平均池化後的文件向量。冒號表示取出選定列的所有座標。

上面的各種表都提供取得 E 的方式。有了 token 向量，平均池化就能把不同長度的文件轉成固定寬度的特徵。這個彙整步驟與嵌入表的訓練目標是分開的。

平均池化很適合教學，因為限制很清楚：任意排列文件中的 token，都會得到同一個平均。若使用相同靜態詞向量，「the technician helped the customer」與「the customer helped the technician」的表示完全相同。否定詞也只是平均中的一份貢獻，並未自動成為改變片語意義的運算子。`bank` 在 `river bank` 與 `bank account` 會取出相同靜態向量；增加訓練資料不會改變這個查詢規則。

這裡有兩個獨立選擇：如何取得 token 向量，以及如何組合向量。可以凍結預訓練 token 向量，只訓練分類器；也可以與分類器一起微調向量，或從頭訓練全部參數。凍結能減少任務專屬參數；微調能配合領域中的詞義，但小型資料集也可能扭曲表示或記住標籤。

不要只靠漂亮的群聚圖比較嵌入。二維投影可能隱藏原空間中的重疊。應在相同任務、切分、分類器容量與調參額度下比較表示。如果換成稠密表示時也更換下游模型頭，實驗便無法單獨識別表示的影響。

餘弦量測方向，不量測真值

計數式範例比較的是已正規化為單位長度的向量。此時內積就是兩者夾角的餘弦：

$$\cos(\mathbf{a}, \mathbf{b}) = \frac{\mathbf{a}^\top \mathbf{b}}{\|\mathbf{a}\|_2 \|\mathbf{b}\|_2}.$$

其中 $\mathbf{a}$ 與 $\mathbf{b}$ 是非零的嵌入直向行向量，$\mathbf{a}^\top\mathbf{b}$ 是內積，$\|\cdot\|_2$ 是歐幾里得長度。餘弦介於 -1 與 1。零向量需要明確處理規則，因為分母為零。

餘弦消除任一向量乘上正比例常數的影響，但不會消除所有與訓練目標相容的變換。在兩個嵌入表的模型中，預測分數取決於輸入與輸出向量的乘積。許多不同的表可以產生相同乘積：

$$E' = EA, \qquad U' = UA^{-\top}, \qquad E'U'^\top = EU^\top.$$

其中 E 與 U 是形狀為 $V \times D$ 的輸入及輸出表，A 是任意可逆的 $D \times D$ 矩陣，$A^{-\top}$ 是其反矩陣的轉置，E' 與 U' 是變換後的表。$EU^{\top}$ 的每個儲存格是一個中心詞與輸出詞的預測分數。因此，這種配對變換不會改變完整 softmax 的預測與負取樣目標。同樣的論證也適用於 GloVe 的 $\mathbf{w}_i^{\top}\tilde{\mathbf{w}}_j$。

正交座標變換會保留輸入向量的夾角，一般可逆變換則未必。因此，相同預測損失無法唯一決定保留之輸入表的餘弦鄰近關係。最佳化、初始化與額外限制會選出某個解。這也是低訓練損失本身無法保證下游相似度實用的原因之一。

下列獨立數值檢查結合三件事：分數保留、平均運算的詞序不變性，以及上方推導出的雜訊相依最佳分數。矩陣是為了這個計算手動選定的，不是訓練出的語言向量。

```
import numpy as np

# Two input rows and three output rows: scores are center-output dot products.
E = np.array([[1., 1.], [1., -1.]])
U = np.array([[1., 0.], [0., 1.], [-1., 1.]])
A = np.diag([2., 0.5])
changed_E = E @ A
changed_U = U @ np.linalg.inv(A).T

def cosine(a, b):
    return (a @ b) / (np.linalg.norm(a) * np.linalg.norm(b))

assert np.allclose(E @ U.T, changed_E @ changed_U.T)
assert np.allclose(E[[0, 1]].mean(0), E[[1, 0]].mean(0))
print('scores unchanged:', np.allclose(E @ U.T, changed_E @ changed_U.T))
print('cosine:', f'{cosine(E[0], E[1]):.3f}', '->',
      f'{cosine(changed_E[0], changed_E[1]):.3f}')
print('mean unchanged by order:',
      np.allclose(E[[0, 1]].mean(0), E[[1, 0]].mean(0)))

# The unrestricted negative-sampling optimum depends on the noise baseline.
p, K = 0.5, 2
for q in [0.25, 0.05]:
    score = np.log(p / (K * q))
    sigmoid = 1 / (1 + np.exp(-score))
    gradient = (p + K * q) * sigmoid - p
    assert np.isclose(gradient, 0)
    print('noise', q, 'optimal score', f'{score:.3f}')
```

輸出如下：

```
scores unchanged: True
cosine: 0.000 -> 0.882
mean unchanged by order: True
noise 0.25 optimal score 0.000
noise 0.05 optimal score 1.609
```

餘弦改變了，所有中心詞與輸出詞的分數卻維持不變。平均相等則有不同含意：一旦表示合併兩種 token 順序，只收到平均值的分類器就無法還原原本是哪個順序。這些是表示的性質，不是某個最佳化器訓練不佳的證據。

評估詞嵌入

白話版。判斷嵌入有兩種方式：問它的幾何是否符合人對詞的判斷（內在評估），或把它接到任務上量測任務表現（外在評估）。單靠任何一種，都無法說明向量是否「好」。

相似度：讓餘弦與人類評分相關

詞相似度（word similarity）資料集列出詞對與分級的人類評分。模型以餘弦替每一對打分，評估則以 **Spearman 等級相關（Spearman's rank correlation）**量測兩種排序的一致程度：

$$\rho = 1 - \frac{6 \sum_{i=1}^{n} \delta_i^2}{n(n^2 - 1)}.$$

其中 n 是詞對數，δ_i 是第 i 對在模型餘弦排序與人類評分排序中的名次差，$\rho \in [-1, 1]$。這個公式假設沒有同名次；有同名次時，ρ 是名次的 Pearson 相關。Spearman 的 ρ 只用名次，因此忽略餘弦的尺度。

評分代表什麼，和統計量一樣重要。WordSim-353 是廣泛使用的詞對評分集 [12]。Hill 等人指出它的評分沒有區分相似度與聯想，因此建立 SimLex-999 明確量化**相似度**，讓有關聯但不相似的詞對得到低分 [13]。以寬視窗訓練的模型，可能在聯想式資料上得分高、在相似度式資料上得分低，原因如詞彙語意一節所述：分布式證據混合了各種關係。

類比：向量位移測試及其限制

Mikolov、Yih 與 Zweig 觀察到，某些關係會呈現為大致固定的向量位移 [14]。要回答「a 之於 a^*，如同 b 之於 ?」，**位移法（offset method）**傳回

$$x^* = \operatorname*{argmax}_{x \in \mathcal{V} \setminus \{a, a^*, b\}} \cos(\mathbf{e}_x, \, \mathbf{e}_{a^*} - \mathbf{e}_a + \mathbf{e}_b).$$

其中 $\mathcal{V}$ 是詞彙表，$\mathbf{e}_x$ 是候選 x 的向量，題目中的三個詞被排除在候選之外。Linzen 指出這個排除其實很關鍵：目標點 $\mathbf{e}_{a^*} - \mathbf{e}_a + \mathbf{e}_b$ 常常最接近 b 本身，因此這個方法的成功混合了位移一致性與 b 周圍的鄰近結構 [15]。他建議在類比準確率旁一併報告簡單基準，例如直接傳回 b 的最近鄰。

下一個範例用自編的相似度評分與三道類比題，評估先前的 PPMI 向量。評分是本課自行編造的，不是基準資料集。

```
import numpy as np
from scipy.stats import spearmanr

text = """the cat drinks milk . the dog drinks water . a cat chases a mouse .
a dog chases a cat . the kitten drinks milk . the puppy drinks water .
a kitten chases a ball . a puppy chases a ball . the car needs fuel .
the truck needs fuel . a car drives on the road . a truck drives on the road ."""
sents = [s.split() for s in text.split(".") if s.strip()]
vocab = sorted({w for s in sents for w in s})
```

```
idx = {w: i for i, w in enumerate(vocab)}
C = np.zeros((len(vocab), len(vocab)))
for s in sents:
    for i, w in enumerate(s):
        for j in range(max(0, i - 2), min(len(s), i + 3)):
            if j != i:
                C[idx[w], idx[s[j]]] += 1
P = C / C.sum()
with np.errstate(divide="ignore"):
    ppmi = np.maximum(np.log(P / (P.sum(1, keepdims=True) * P.sum(0, keepdims=True))), 0)
U, S, _ = np.linalg.svd(ppmi)
E = U[:, :8] * S[:8]
E /= np.linalg.norm(E, axis=1, keepdims=True)
cos = lambda a, b: float(E[idx[a]] @ E[idx[b]])

# 1) Similarity: correlate model cosines with (invented) graded judgments, 0-10.
judged = [("cat", "kitten", 9), ("dog", "puppy", 9), ("car", "truck", 8), ("milk", "water",
 ↪  5),
          ("cat", "dog", 6), ("cat", "mouse", 3), ("car", "road", 2), ("kitten", "fuel", 0)]
model = [cos(a, b) for a, b, _ in judged]
rho = spearmanr(model, [h for _, _, h in judged]).statistic
print(f"Spearman rho = {rho:.3f} over {len(judged)} pairs")

# 2) Analogy a:b :: c:? with the offset b - a + c (3CosAdd).
def analogy(a, b, c, exclude_inputs=True):
    target = E[idx[b]] - E[idx[a]] + E[idx[c]]
    scores = E @ (target / np.linalg.norm(target))
    for i in np.argsort(-scores):
        if not exclude_inputs or vocab[i] not in (a, b, c):
            return vocab[i]
for a, b, c in [("cat", "kitten", "dog"), ("car", "fuel", "cat"), ("cat", "milk", "puppy")]:
    print(f"{a}:{b} :: {c}:?  ->", analogy(a, b, c), "| without excluding inputs ->",
          analogy(a, b, c, exclude_inputs=False))
```

執行會印出：

```
Spearman rho = 0.837 over 8 pairs
cat:kitten :: dog:?  -> puppy | without excluding inputs -> puppy
car:fuel :: cat:?  -> kitten | without excluding inputs -> cat
cat:milk :: puppy:?  -> water | without excluding inputs -> water
```

有兩題得到預期答案。中間那題同時展示兩個限制。不排除題目詞時，答案是 `cat`，也就是題目中的詞。排除後，答案是 `kitten`，即 `cat` 的最近鄰，而不是預期的「牠吃喝什麼」答案 `milk`。位移並沒有承載這個關係；決定答案的是 b 的鄰近結構。當然，八組自編詞對與三道類比題，遠不足以支持關於這些向量的任何一般結論。

內在評估與外在評估

Schnabel 等人比較多種評估方法，發現不同評估對嵌入方法的排序不同、詞頻會影響結果與餘弦相似度，而且外在任務不應被當成通用嵌入品質的代理指標 [16]。實務原則是：針對應用需要的

關係進行評估。

最近鄰清單能揭露明顯的語料模式，但它是診斷，不是應用分數。分開詞彙問題：候選是同義詞、反義詞、主題相關詞，還是能占據相同語法位置的詞？單一、未加限定的「相關」會掩蓋這些差別。

在一般留出樣本之外加入控制條件的配對。情緒表示應比較正面陳述與其否定；角色敏感任務應交換誰對誰做事，同時固定詞的集合；技術語料則測試術語的領域用法與日常用法。這些測試問的是表示是否保留必要差異，不要求每對詞都位於預設座標。

若平均池化把重要配對變成相同向量，在表示固定時，收集更多分類標籤也無法修復碰撞。若向量已區分配對，分類器卻預測錯誤，問題可能出在監督或決策邊界。從最早遺失必要證據的階段定位失敗。

嵌入中的偏誤

白話版。嵌入會學到語料中的任何規律，包括社會刻板印象。如果某一群名字與某些職業共同出現得比較多，向量就會編碼這種關聯，下游系統也可能把它複製出來。

Bolukbasi 等人顯示以新聞文字訓練的詞嵌入編碼了性別刻板印象，在空間中找出一個性別方向，並提出把它從應該性別中立的詞中移除 [17]。Caliskan 等人提出仿照心理學內隱聯結測驗（Implicit Association Test）的詞嵌入聯結測驗（Word Embedding Association Test，WEAT），發現嵌入重現了一系列已有文獻記錄的人類偏誤 [18]。對目標詞 w 與兩個屬性集合 A、B，WEAT 使用的關聯為

$$s(w, A, B) = \frac{1}{|A|}\sum_{a \in A} \cos(\mathbf{e}_w, \mathbf{e}_a) - \frac{1}{|B|}\sum_{b \in B} \cos(\mathbf{e}_w, \mathbf{e}_b).$$

其中 $|A|$ 與 $|B|$ 是集合大小，$\mathbf{e}$ 表示詞向量。正值代表 w 平均而言較接近集合 A。完整測試會在兩組目標詞上彙總這個分數，並與排列分布比較。

下列自編語料中，沒有任何句子把名字和職業連在一起。職業詞總是與自己的活動一起出現，而自編的名字以偏斜的比例與各種活動一起出現。

```
import random
import numpy as np

rng = random.Random(0)
# An invented corpus. Job words always appear with their own activity; invented
# names appear with activities at skewed rates. No sentence says "ana is a pilot".
group_a, group_b = ["ana", "ari"], ["bo", "bea"]
activity = {"pilot": "flies the plane", "baker": "bakes the bread", "clerk": "files the
 ↪  forms"}
share_a = {"pilot": 0.8, "baker": 0.2, "clerk": 0.5}    # P(group A | activity)
sents = []
for _ in range(3000):
    job = rng.choice(list(activity))
    if rng.random() < 0.5:
        sents.append(f"the {job} {activity[job]}".split())
    else:
```

```
        name = rng.choice(group_a if rng.random() < share_a[job] else group_b)
        sents.append(f"{name} {activity[job]}".split())
vocab = sorted({w for s in sents for w in s})
idx = {w: i for i, w in enumerate(vocab)}
C = np.zeros((len(vocab), len(vocab)))
for s in sents:
    for i, w in enumerate(s):
        for j in range(max(0, i - 2), min(len(s), i + 3)):
            if j != i:
                C[idx[w], idx[s[j]]] += 1
P = C / C.sum()
with np.errstate(divide="ignore"):
    ppmi = np.maximum(np.log(P / (P.sum(1, keepdims=True) * P.sum(0, keepdims=True))), 0)
U, S, _ = np.linalg.svd(ppmi)
E = U[:, :4] * S[:4]
E /= np.linalg.norm(E, axis=1, keepdims=True)

def association(word, E):            # mean cosine to group A minus mean cosine to group B
    unit = E / np.linalg.norm(E, axis=1, keepdims=True)
    w = unit[idx[word]]
    return np.mean([w @ unit[idx[a]] for a in group_a]) - np.mean([w @ unit[idx[b]] for b in
     ↪  group_b])

for job in activity:
    print(f"{job:6s} association {association(job, E):+.3f}")

# Project out one "group direction" and measure again.
d = E[[idx[a] for a in group_a]].mean(0) - E[[idx[b] for b in group_b]].mean(0)
d /= np.linalg.norm(d)
E_proj = E.copy()
for job in activity:
    E_proj[idx[job]] -= (E_proj[idx[job]] @ d) * d
print("after projection:", [f"{job} {round(association(job, E_proj), 3) + 0.0:+.3f}" for job
 ↪  in activity])
```

執行會印出：

```
pilot  association +0.795
baker  association -0.811
clerk  association +0.072
after projection: ['pilot +0.000', 'baker +0.000', 'clerk +0.000']
```

「誰從事哪種活動」的偏斜，變成名字與職業詞之間的強烈關聯，即使兩者從未在同一句中出現：它們共享活動上下文。投影掉群組方向後，量測到的關聯變成零，但這個結果是循環論證，因為投影移除的正是這個分數所量測的方向。Gonen 與 Goldberg 顯示，對已發表的去偏方法而言，偏誤大多只是被隱藏而非移除：它仍留在那些號稱已中立化的詞彼此間的距離中，而且可以被還原 [19]。應把去偏步驟視為需要在下游任務上稽核的一項介入，而不是保證，並記得關聯的來源在資料本身。

從詞向量到句子嵌入

白話版。嵌入提供特徵；應用仍需要目標、監督與行動規則。當應用要比較整個句子時，就針對它要支援的比較方式訓練句子表示。

文件分類可以先做平均池化實驗。替每份訓練文件建立一個向量，再以標籤配適分類器；評估時使用相同 tokenizer、嵌入查表與池化方法。將它與 NLP 03 的稀疏特徵與卷積編碼器比較。固定資料切分與分類器的調整機會，才能公平檢查表示改變的效果。

凍結的表使用標註任務之前學到的上下文統計；微調則讓標籤調整每列。兩種政策回答不同問題：語料中可重用的特徵是否已能區分類別，以及任務特定的調整是否改善表示。標籤有限時，微調也可能配適偶然出現的識別碼。將對話或重複文件模板等群組完整留出，有助於揭露這種行為。

決定如何處理未涵蓋文字。把每個未知詞映射到同一列，會合併所有這類詞。略過它們會改變池化分母，也可能使文件變空。回傳零向量不會形成有意義的餘弦方向。子詞向量能減輕、但無法消除這個問題。涵蓋率、保留 token 數，以及空表示的處理動作，都是模型規格的一部分。

詞層級標記應保留 token 向量序列，而不是池化成單一文件向量。下游編碼器就能先結合局部上下文，再為各位置預測標籤。這不會讓原本的表成為上下文表示：兩種詞義的 `bank` 初始查表值仍相同。上下文相依的計算發生在後續。

相似度與對比式句子學習

適合預測鄰近詞的表示，可能需要調適，才能讓整個句子的距離可靠。Sentence-BERT 使用共享編碼器與任務目標訓練成對的句子表示，讓比較句子時不必對每一對候選共同編碼 [20]。計算上的差別很重要：分別快取文件向量可以支援檢索；同時讀取查詢與候選文件的模型，則必須為每一對執行。

一般的對比目標能把想要的幾何關係寫清楚：

$$\mathcal{L}_{\text{pair}} = -\log \frac{\exp(\cos(\mathbf{q}, \mathbf{a}^{+})/\tau)}{\sum_{j=1}^{M} \exp(\cos(\mathbf{q}, \mathbf{a}_j)/\tau)}.$$

其中 $\mathcal{L}_{\text{pair}}$ 是一個查詢的損失，$\mathbf{q}$ 是查詢向量，$\mathbf{a}^{+}$ 是候選中與查詢相關的配對文字向量，$\mathbf{a}_j$ 是候選 j 的向量，M 是候選數，$\tau > 0$ 是控制分布尖銳程度的溫度。這是一般性的學習構造，不表示每個句子編碼器都使用完全相同的目標。它和 Word2Vec 的 softmax 形式相同：一個觀察到的配對與其他候選競爭。

SimCSE 是廣泛使用的一個實例 [21]。它的非監督版本把同一個句子編碼兩次；由於兩次前向傳遞的 dropout 不同，兩個向量略有差異而形成正配對，批次中的其他句子則是負樣本。作者報告移除 dropout 會導致表示崩塌。它的監督版本使用自然語言推論資料，以蘊涵配對作為正樣本、矛盾配對作為困難負樣本 [21]。

負樣本定義了什麼叫「不同」。兩則討論不同發票的客服訊息，對尋找同一案件可能是實用的負樣本；對辨識 billing 類別則可能是不好的負樣本。錯誤負樣本會把語意相關的例子推遠。在改變編碼器之前，先檢查正負配對如何取樣。

相似度也不等於蘊涵。「The parcel arrived」與「The parcel did not arrive」共享主題與許多 token，在語意搜尋空間可能靠近，卻做出相反陳述。因此，即使使用相同編碼器，分類、語意搜尋與推論仍需要不同的驗證樣本。NLP 11 把推論當成獨立任務處理。

預告：上下文嵌入

白話版。靜態表在任何地方都給 `bank` 同一個向量。上下文模型則利用周圍的詞，替每次出現計算不同的向量，讓河岸詞義與金錢詞義得以分開。

Peters 等人將 ELMo 表示計算為整個輸入句子的函數，使用雙向語言模型的內部狀態，並把它們作為特徵加入任務模型 [22]。BERT 在預訓練編碼器內更進一步：token 最後的隱藏狀態取決於其他允許看到的輸入 token [23]。差別是**查表與計算**。兩者都可能從嵌入表開始；靜態 Word2Vec 直接提供一列，BERT 則透過上下文相依網路處理各列與位置。

上下文能改善詞義區分，但不會讓每個隱藏狀態的餘弦自動成為已校準的相似度量。靜態向量是跨越許多次出現的類型層級摘要；上下文向量是一次出現的 token 層級表示。應用需要哪一種，取決於它問的是一般意義下的詞，還是此處用法中的詞。NLP 07 建立這些模型使用的注意力機制，NLP 10 則發展遮罩預訓練與 BERT 的上下文表示，這裡不重複。

現代觀點

Turney 與 Pantel 的綜述 [6] 區分詞項與文件、詞與上下文，以及配對與模式矩陣。這種分類使證據來源明確：文件關聯、鄰近詞關係與關係模式是不同觀察。本課的啟示是，先定義矩陣，再解讀向量空間。

Apidianaki 的 2023 年綜述 [24] 從靜態類型層級向量，走向上下文相依的出現層級表示，再討論如何回到靜態表示。內容涵蓋語意特化、詞義區分、表示解讀，以及從上下文編碼器取得靜態向量。核心區別是在哪個層級彙整證據，不是把架構排成「越新就自動解決所有詞彙任務」的名次。把詞彙資源中的關係注入向量的方法，與本課的 WordNet 一節直接相關。

在句子與文字層級的嵌入方面，Muennighoff 等人的 MTEB 基準涵蓋 8 種嵌入任務、58 個資料集與 112 種語言，作者發現沒有任何單一文字嵌入方法在所有任務上都占優勢 [25]。在 2023–2026 年間，多數實用的文字嵌入來自以上述對比目標微調的預訓練 Transformer 編碼器或語言模型，而 MTEB 風格的基準套件是比較它們的常見方式。靜態向量仍是快速、透明的基準，是低資源情境中的特徵，也是研究對象。

因此，適合檢視研究的問題是：**這種表示保留了哪個詞義區別，又如何驗證？**上下文隱藏狀態、由它們聚合而來的靜態向量，以及以任務訓練的句子表示，回答不同問題。能保留大致主題的嵌入，仍可能合併否定陳述或罕見詞義。如果小型靜態表的可測不變性符合應用，它仍是適當基準。

開放問題包括跨領域可靠幾何、證據稀少時的詞義區分、去偏後仍存留的偏誤，以及固定語料與評估條件的比較。多語工作還加入斷詞與不均衡涵蓋率：tokenizer 與訓練語料曾為該類型提供證據，這列才有用。應與最終分數一起報告這些條件與挑戰樣本，讓解釋連結到實際觀察的語言行為。

重點整理

- 詞彙表項目是詞形或詞元，不是詞義；同義、反義、上位等詞彙關係存在於詞義之間，WordNet 以人工記錄它們。
- 靜態嵌入彙整許多次出現，為每個詞彙表項目形成一個向量；分布式相似度捕捉共享用法，混合了同義、反義與相關性。
- PPMI 將共現與獨立基準比較；SVD、使用負取樣的 Skip-gram 與 GloVe，都在秩限制下壓縮關聯統計。
- CBOW 從合併的鄰近詞預測中心詞；Skip-gram 分別預測鄰近詞；負取樣區分配對的統計來源，不是正規化分布。
- fastText 以字元 n-gram 組成詞向量，因此拼寫相關的詞共享參數，沒見過的詞也有向量。
- 相似度與類比測試取決於評分的意義，以及是否排除題目詞等細節；應評估應用需要的關係。
- 嵌入會吸收語料中的關聯，包括社會偏誤；以投影去偏可能只是隱藏而非移除偏誤。
- 平均池化消除詞序；句子嵌入應針對它服務的比較訓練，而上下文模型是替每次出現計算向量，而不是查表。

練習

1. **建立樣本**。列出 `robots help people today` 的半徑一 CBOW 與 Skip-gram 樣本，先只使用內部中心位置，再納入邊界中心。哪些輸入形狀會改變？

   提示

   中心位置數與中心、上下文配對數分開計算。長度不同的 CBOW 上下文需要遮罩或分開處理；Skip-gram 每個配對仍只收到一個中心 ID。

2. **檢查矩陣儲存格**。建立一份語料，讓某個上下文整體很常見，卻對選定中心詞具有接近零的 PMI。再增加觀察到的中心與上下文配對，解釋分子與邊際機率如何改變。

   提示

   先從接近獨立的聯合表開始。修改計數後重算所有邊際；把邊際固定會變成分析另一張表。

3. **訓練 GloVe，看它搖擺**。在 PyTorch 中實作 GloVe 目標，於玩具語料上只用非零儲存格訓練。以三個隨機種子訓練，比較 `cat` 的最近鄰。接著計算參數數與非零儲存格數。是什麼造成不穩定？怎樣能讓問題的解更確定？

   提示

   兩個 $V \times D$ 的表加上兩個偏差向量，參數數很容易超過觀察到的儲存格數。上下文更多樣的大語料會增加方程式；較小的 D 則減少未知數。

4. **相似度與相關性**。寫出十組相關但不相似的詞對（例如 `cup` 與 `coffee`），以及十組相似的詞對。在你自己編寫的語料上，以視窗半徑 1 與 4 分別執行 PPMI 程式，比較餘弦如何區分兩組。

提示

較寬的視窗收集主題共現，有利於相關性；較窄的視窗強調能填入同一位置的詞。兩組的變化都要報告，而不是只給單一分數。

5. **揭露不變性**。建立兩句詞計數相同、參與者角色不同的有意義句子。即使每個詞向量都完美符合其訓練目標，說明靜態平均池化為何仍無法區分兩句。

提示

改變順序不會改變加總。這個限制對任何靜態嵌入表成立，不只對本課的合成數值成立。

6. **在投影之外稽核**。在偏誤範例中完成投影後，用剩下的所有維度計算每個職業詞最接近的名字，並訓練一個小型邏輯斯分類器，從投影後職業詞的鄰居預測 A 組或 B 組。關聯是否仍留下痕跡？

提示

投影只以一種方式移除了一個方向。尋找這個分數沒有量測到的結構，就像 Gonen 與 Goldberg 以分群與分類所做的那樣。

參考文獻

1. M. Hagiwara, *Real-World Natural Language Processing: Practical Applications with Deep Learning*, Manning, 2021; Chapters 3–4. publisher
2. S. McRoy, *Principles of Natural Language Processing*, open textbook, 2021; Chapter 6, “Benchmark Tasks for Language Modelling.” chapter
3. Y. Bengio, R. Ducharme, P. Vincent and C. Jauvin, “A neural probabilistic language model,” *Journal of Machine Learning Research* 3, 2003. paper
4. G. A. Miller, “WordNet: a lexical database for English,” *Communications of the ACM* 38(11), 1995. paper
5. Z. S. Harris, “Distributional structure,” *Word* 10(2–3), 1954. paper
6. P. D. Turney and P. Pantel, “From frequency to meaning: vector space models of semantics,” *Journal of Artificial Intelligence Research* 37, 2010. survey
7. T. Mikolov, K. Chen, G. Corrado and J. Dean, “Efficient estimation of word representations in vector space,” ICLR Workshop, 2013. paper
8. T. Mikolov, I. Sutskever, K. Chen, G. Corrado and J. Dean, “Distributed representations of words and phrases and their compositionality,” *Advances in Neural Information Processing Systems* 26, 2013. paper
9. O. Levy and Y. Goldberg, “Neural word embedding as implicit matrix factorization,” *Advances in Neural Information Processing Systems* 27, 2014. paper
10. J. Pennington, R. Socher and C. D. Manning, “GloVe: global vectors for word representation,” EMNLP, 2014. paper

11. P. Bojanowski, E. Grave, A. Joulin and T. Mikolov, “Enriching word vectors with subword information,” *Transactions of the Association for Computational Linguistics* 5, 2017. paper
12. L. Finkelstein, E. Gabrilovich, Y. Matias, E. Rivlin, Z. Solan, G. Wolfman and E. Ruppin, “Placing search in context: the concept revisited,” *ACM Transactions on Information Systems* 20(1), 2002. paper
13. F. Hill, R. Reichart and A. Korhonen, “SimLex-999: evaluating semantic models with (genuine) similarity estimation,” *Computational Linguistics* 41(4), 2015. paper
14. T. Mikolov, W.-t. Yih and G. Zweig, “Linguistic regularities in continuous space word representations,” NAACL-HLT, 2013. paper
15. T. Linzen, “Issues in evaluating semantic spaces using word analogies,” Workshop on Evaluating Vector-Space Representations for NLP (RepEval), 2016. paper
16. T. Schnabel, I. Labutov, D. Mimno and T. Joachims, “Evaluation methods for unsupervised word embeddings,” EMNLP, 2015. paper
17. T. Bolukbasi, K.-W. Chang, J. Zou, V. Saligrama and A. Kalai, “Man is to computer programmer as woman is to homemaker? Debiasing word embeddings,” *Advances in Neural Information Processing Systems* 29, 2016. paper
18. A. Caliskan, J. J. Bryson and A. Narayanan, “Semantics derived automatically from language corpora contain human-like biases,” *Science* 356(6334), 2017. paper
19. H. Gonen and Y. Goldberg, “Lipstick on a pig: debiasing methods cover up systematic gender biases in word embeddings but do not remove them,” NAACL-HLT, 2019. paper
20. N. Reimers and I. Gurevych, “Sentence-BERT: sentence embeddings using Siamese BERT-networks,” EMNLP-IJCNLP, 2019. paper
21. T. Gao, X. Yao and D. Chen, “SimCSE: simple contrastive learning of sentence embeddings,” EMNLP, 2021. paper
22. M. E. Peters, M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee and L. Zettlemoyer, “Deep contextualized word representations,” NAACL-HLT, 2018. paper
23. J. Devlin, M.-W. Chang, K. Lee and K. Toutanova, “BERT: pre-training of deep bidirectional transformers for language understanding,” NAACL-HLT, 2019. paper
24. M. Apidianaki, “From word types to tokens and back: a survey of approaches to word meaning representation and interpretation,” *Computational Linguistics* 49(2), 2023. survey
25. N. Muennighoff, N. Tazi, L. Magne and N. Reimers, “MTEB: massive text embedding benchmark,” EACL, 2023. paper

NLP 05・用於語言的循環神經網路

從 n-gram 語言模型及其記憶上限出發，建立、訓練並評估循環語言模型：Elman RNN、LSTM 與 GRU、字元層級建模、取樣，以及實務訓練技巧。

網站版本: https://blog.shyandram.dev/zh-tw/learn/nlp/recurrent-neural-networks/

你將學到

- 分清楚 n-gram 出現、文件特徵與條件語言模型，並從序列機率推導 unigram、bigram 與 trigram 預測。
- 計算、平滑與內插 n-gram 估計，再解讀保留資料的負對數概似及困惑度。
- 說明固定長度歷史為何造成更多資料也無法移除的記憶上限。
- 寫出 Elman 循環語言模型、其下一詞元交叉熵與困惑度，並說明時間反向傳播、梯度消失、LSTM 與 GRU 單元如何影響模型能學到什麼。
- 以串流打包、截斷時間反向傳播、梯度裁剪與提前停止訓練字元層級 LSTM 語言模型，並與調校過的 n-gram 基準公平比較。
- 以溫度從訓練好的模型取樣，並說明樣本能與不能告訴我們什麼。

先看全貌

語言模型根據前文預測下一個詞元，藉此替文字配置機率。本章建立兩種這樣的模型，並在同一套評估約定下比較。第一種是 n-gram 模型，它使用前文的一小部分預測下一個詞元。機率表可以直接檢查：我們能把預測追溯到訓練次數，並看清楚模型忽略了哪些較早的詞。因此，它適合作為第一個語言模型、受控基準，以及理解更複雜模型也會遇到的統計問題。第二種是循環神經網路（recurrent neural network，RNN），它以一個在每個詞元之後更新的學習狀態取代次數表，因此預測原則上可以依賴整個前綴。

> **用最簡單的話說**
>
> 想像一本記錄大家接著說什麼的筆記本。一頁記錄「cat」後面接什麼，另一頁記錄「the cat」後面接什麼。需要猜測時，就翻到對應頁面並計算範例次數。較長的頁面標題提供更多上下文，卻也可能讓許多頁面幾乎空白。平滑會替這一頁還沒出現過的詞保留一點空間。循環網路的做法不同：它不替每個標題準備一頁，而是在腦中保留一段簡短的摘要，每聽到一個詞就改寫一次。

Hagiwara 的語言模型章節把下一詞元預測連到實際文字生成 [1]。McRoy 的教材提供更完整的語言及應用背景 [2]。分詞放在 NLP 01；嵌入及其訓練目標放在 NLP 04。第 1–5 節專注於計數模型本身；第 6–12 節以循環網路取代次數表；以注意力為基礎的模型則在 NLP 07 與 NLP 09 介紹。循環網路的一般數學（完整的時間反向傳播推導、梯度界限、閘門方程式）在 ML 04；這裡採取語言模型的觀點。讀本章時持續追問：**哪些資訊與觀察證據決定下一個詞元的機率？**

1. 「n-gram」的三種意思

n-gram 是連續 n 個詞元組成的序列。詞元可以是詞、子詞或字元；分詞方式決定計算的單位。以詞序列 `a cat sleeps` 為例，unigram 是 `a`、`cat`、`sleeps`；bigram 是 `a cat`、`cat sleeps`；trigram 則是 `a cat sleeps`。

以下三種用途要分清楚：

用途	儲存或預測的物件	能告訴我們什麼
n-gram 出現	一個連續詞元元組	某個序列的局部片段
文件特徵	一個元組在文件中的次數或是否出現	提供給分類器或檢索模型的證據
n-gram 語言模型	每個保留歷史對應的正規化下一詞元分布	用來替序列評分或生成的機率

文件的 bigram 特徵 `not ready` 可以協助分類器區分操作狀態，但它本身沒有指定 `not` 後面出現 `ready` 的機率：還需要其他競爭詞元的次數與正規化規則。同樣地，語言模型中的片語機率取決於句子上下文；文件特徵次數通常會合併該片語在整篇文件中的所有出現。

語言模型的階數 n 包含正在預測的詞元。Bigram 使用**前一個詞元**，不是前兩個；trigram 使用前兩個。句子邊界需要一致規則：起始標記提供最初上下文，結束標記則是代表停止的預測結果。範例從不預測起始標記；若把它放進預測詞彙表，就會替句子格式不允許的事件配置機率。

2. 精確分解與有限歷史

逐一預測詞元，可以精確表示序列機率。捨棄較早上下文是模型假設，並不是機率理論要求這樣做。

$$p(x_1, \dots, x_T) = \prod_{t=1}^{T} p(x_t \mid x_{<t}), \qquad p_n(x_1, \dots, x_T) = \prod_{t=1}^{T} p_n(x_t \mid h_t^{(n)}).$$

其中 x_t 是位置 t 的詞元；T 是預測位置數，若使用結束標記也將它計入；$x_{<t}$ 是完整前綴；n 是 n-gram 階數；$h_t^{(n)}$ 是最多包含前 $n-1$ 個詞元的尾端歷史，需要時補上起始標記；p_n 是有限歷史模型。第一個等式是鏈式法則，第二個式子則定義只依保留尾端內容預測的模型。它不一定等於真實序列分布。

階數	在 `a cat sleeps` 中預測 `sleeps` 前的上下文	條件分布
Unigram，$n = 1$	空歷史	$p_1(\text{sleeps})$
Bigram，$n = 2$	`cat`	$p_2(\text{sleeps} \mid \text{cat})$
Trigram，$n = 3$	`a cat`	$p_3(\text{sleeps} \mid \text{a cat})$

空歷史仍是有效上下文：unigram 模型在每個位置使用同一個下一詞元分布。Bigram 做出更細的區分，卻必須合併所有最後一個詞元相同的前綴。Trigram 可以區分最後兩個詞元不同的前綴，但這些模型都不保證保留內容包含語言依存關係真正需要的資訊。

逐步計算不同階數的差異

下面的教學語料刻意讓冠詞帶有預測資訊：以 the 開頭的範例都以 sleeps 結尾，以 a 開頭的範例都以 eats 結尾。這是受控的統計模式，不是對英語用法的主張。範例使用相同詞彙表及平滑強度，分別估計三種階數。

```
from collections import Counter, defaultdict

train = ["the cat sleeps", "the dog sleeps", "a cat eats", "a dog eats"]
START, END, UNK = "<s>", "</s>", "<unk>"
vocab = sorted({w for s in train for w in s.split()} | {END, UNK})
alpha = 0.2
models = {}
for order in (1, 2, 3):
    rows = defaultdict(Counter)
    for sentence in train:
        tokens = [START] * (order - 1) + sentence.split() + [END]
        for t in range(order - 1, len(tokens)):
            history = tuple(tokens[t-order+1:t])
            rows[history][tokens[t]] += 1
    models[order] = rows

def probability(order, history, word):
    row = models[order].get(tuple(history), {})
    return (row.get(word, 0) + alpha) / (sum(row.values()) + alpha * len(vocab))

for order, rows in models.items():
    for history in rows:
        assert abs(sum(probability(order, history, w) for w in vocab) - 1) < 1e-10
print("vocabulary size:", len(vocab))
print("unigram P(eats):", round(probability(1, (), "eats"), 3))
print("bigram P(eats | cat):", round(probability(2, ("cat",), "eats"), 3))
print("trigram P(eats | a cat):", round(probability(3, ("a", "cat"), "eats"), 3))
print("trigram P(eats | the cat):", round(probability(3, ("the", "cat"), "eats"), 3))
print("unseen-history P(eats):", round(probability(3, (UNK, "cat"), "eats"), 3))
assert probability(3, ("a", "cat"), "eats") > probability(3, ("the", "cat"), "eats")
assert abs(probability(3, (UNK, "cat"), "eats") - 1 / len(vocab)) < 1e-10
```

輸出：

```
vocabulary size: 8
unigram P(eats): 0.125
bigram P(eats | cat): 0.333
trigram P(eats | a cat): 0.462
trigram P(eats | the cat): 0.077
unseen-history P(eats): 0.125
```

在 cat 後面，bigram 次數表有一次 eats 與一次 sleeps。兩種冠詞選擇都進入同一張表。Trigram 則把兩筆觀察分開：a cat 曾接過一次 eats，the cat 則沒有。因此 trigram 能表達這個

差異，但它的每張表都只有一筆觀察。在這份語料中，更細的條件區分會減少每個上下文可用的證據。

未見歷史沒有任何轉移次數；加法平滑使其成為八個預測詞元上的均勻分布。這是完整的機率分布，卻捨棄了有用證據：最後一個詞元 `cat` 其實出現在訓練資料中。與低階分布內插，可以保留這項證據。正規化與有效泛化是不同性質。

3. 從次數到正規化機率

要估計分布，先計算訓練中的每個「歷史—目標」事件，再除以該歷史的總轉移次數。讓結果可重現的兩項細節是：哪些符號會被預測，以及哪些上下文被視為不同。

$$\hat{p}(w \mid h) = \frac{c(h, w)}{c(h)}, \qquad c(h) = \sum_{v \in V} c(h, v), \qquad c(h) > 0.$$

其中 h 是保留歷史；w 是候選下一詞元；V 是預測詞彙表；v 遍歷詞彙表成員；$c(h, w)$ 是訓練中由 h 接到 w 的次數；$c(h)$ 是其總數；$\hat{p}$ 是最大概似估計。Trigram 的 h 是兩個詞元的元組，unigram 則是空元組。若歷史從未出現，這個估計沒有定義。

為什麼觀察相對頻率能解出概似問題？對單一歷史，在機率總和為一的限制下，最大化 $\sum_w c(h, w) \log p(w \mid h)$ 的分布，會讓有正次數的事件機率與次數成正比。正規化限制接著使它除以 $c(h)$。不同歷史可以獨立估計，因為計數表沒有在各歷史之間共享參數。

這種簡單性帶來兩種不同問題。**未見轉移**的歷史曾出現，但目標從未接在後面，因此機率為零；**未見歷史**則完全沒有次數表，使未平滑比例沒有定義。平滑能在固定詞彙表內處理兩者，卻無法生成已被詞彙表替換成未知標記的原始拼字。

加法平滑替每個可能的目標加上少量正次數：

$$p_\alpha(w \mid h) = \frac{c(h, w) + \alpha}{c(h) + \alpha |V|}.$$

其中 p_α 是平滑後分布；$c(h, w)$ 與 $c(h)$ 分別是轉移次數及歷史總轉移次數；w 是候選詞元；h 是保留歷史；V 是預測詞彙表；$|V|$ 是其大小；$\alpha > 0$ 是加到每個候選次數上的數量。把分子對整個詞彙表加總，恰好得到分母。即使歷史從未出現，分布仍有定義：所有候選都得到機率 $1/|V|$。

詞彙表大小與加到單一候選上的次數同樣重要。詞彙表很大時，替每個詞元加上 α 可能占去大量總機率。在這個微小詞彙表上有效的參數，不能未經驗證就原封不動移用到大得多的詞彙表。

程式碼：訓練、評分與取樣一個 bigram 模型

這個完整範例不需要下載資料或 GPU。語料是為教學編寫的；測試句子與訓練句子分開。測試中任何不在訓練詞彙表內的詞，都對應到同一個未知 token。

```
from collections import Counter, defaultdict
import math
import random
```

```
train = [
    "the cat sleeps", "the dog sleeps", "the cat runs",
    "the dog runs", "a cat eats", "a dog eats",
    "the bird sings", "a bird sleeps",
]
test = ["the cat eats", "a dog runs", "the bird sleeps", "a robot runs"]
START, END, UNK = "<s>", "</s>", "<unk>"
words = {w for sentence in train for w in sentence.split()}
vocab = sorted(words | {END, UNK})       # START is context, never predicted
alpha = 0.2
counts = defaultdict(Counter)
unigrams = Counter()

def encode(sentence):
    return [w if w in words else UNK for w in sentence.split()] + [END]

for sentence in train:
    tokens = [START] + encode(sentence)
    for h, w in zip(tokens, tokens[1:]):
        counts[h][w] += 1
        unigrams[w] += 1

def bigram(h, w):
    row = counts.get(h, {})
    return (row.get(w, 0) + alpha) / (sum(row.values()) + alpha * len(vocab))

def unigram(h, w):
    return (unigrams[w] + alpha) / (sum(unigrams.values()) + alpha * len(vocab))

def perplexity(sentences, probability):
    nll, n = 0.0, 0
    for sentence in sentences:
        tokens = [START] + encode(sentence)
        for h, w in zip(tokens, tokens[1:]):
            nll -= math.log(probability(h, w))
            n += 1
    return math.exp(nll / n)

for h in [START] + vocab:
    assert abs(sum(bigram(h, w) for w in vocab) - 1) < 1e-10
print("unigram test perplexity:", round(perplexity(test, unigram), 3))
print("bigram test perplexity:", round(perplexity(test, bigram), 3))
assert UNK in encode(test[-1])

rng = random.Random(7)
h, sample = START, []
for _ in range(20):                      # cap length for this demonstration
    w = rng.choices(vocab, weights=[bigram(h, v) for v in vocab])[0]
    if w == END:
        break
    sample.append(w)
    h = w
print("sample:", " ".join(sample))
```

輸出：

```
unigram test perplexity: 10.382
bigram test perplexity: 3.573
sample: a bird sings
```

assert 檢查每一個條件分布加總為一，以及「robot」被對應到未知 token。bigram 模型對測試句子的「驚訝程度」遠低於 unigram 模型，因為詞序帶有資訊。取樣仍可能產生彆扭的字串：機率大於零不代表合乎文法。長度上限只是讓示範停下來，並不改變評分所用的機率。

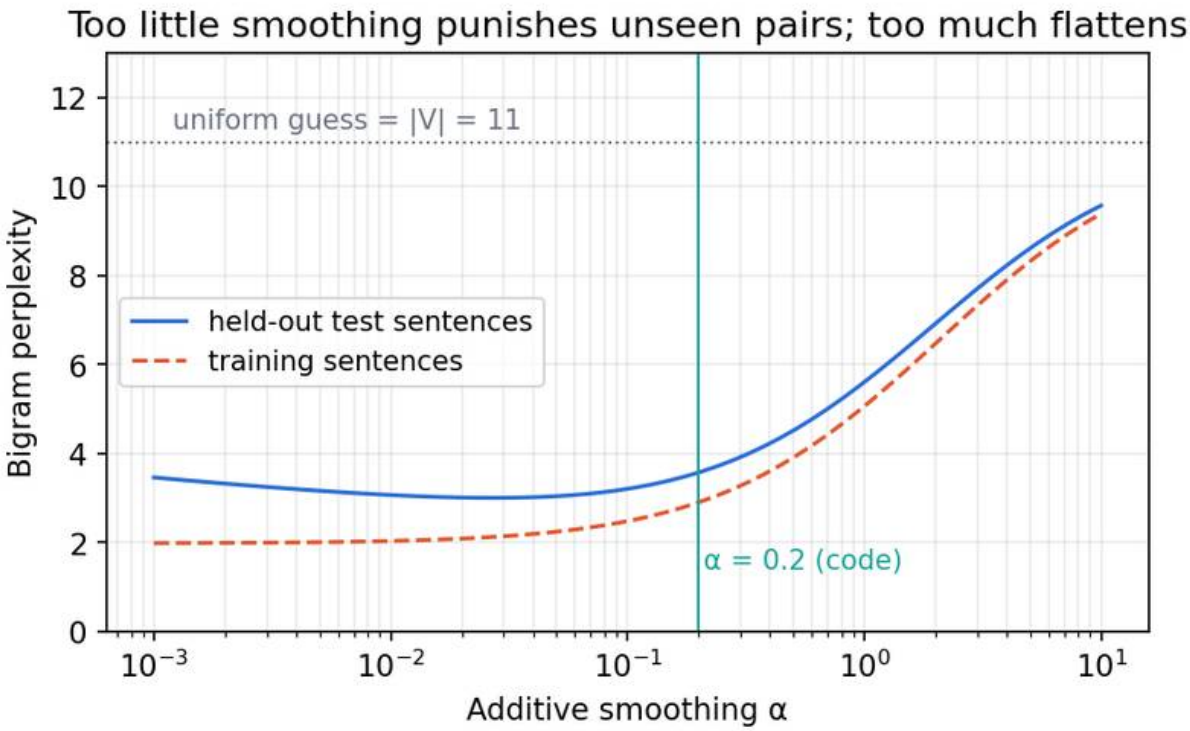


圖 *1* — 平滑替未觀察轉移保留機率。合成保留資料曲線說明取捨；四個保留句子無法證明一般情況的最佳設定。

分清楚平滑與更長的記憶

平滑處理的是既定表示下尚未觀察到的轉移，無法補回已捨棄的歷史。在 “the technician did not repair the” 與 “the technician repaired the” 之後，bigram 都只看見 `the`。即使更早的詞會讓不同延續較合適，它仍必須給出相同的下一詞元分布。增加語料可以更準確地估計這個共同分布，卻不會改變兩者必須相等的限制。

加法平滑也有精確的混合解釋。對觀察次數大於零的上下文：

$$p_\alpha(w \mid h) = \lambda_h \hat{p}(w \mid h) + (1 - \lambda_h)\frac{1}{|V|}, \qquad \lambda_h = \frac{c(h)}{c(h) + \alpha|V|}, \qquad \hat{p}(w \mid h) = \frac{c(h, w)}{c(h)}.$$

其中 p_α 是平滑後的分布，h 是保留的歷史，w 是候選詞元，$c(h,w)$ 是該轉移的觀察次數，$c(h) > 0$ 是從此歷史出發的總次數，$\hat{p}$ 是未平滑估計，$\alpha > 0$ 是平滑參數，$|V|$ 是詞彙表大小，λ_h 則是觀察資料的權重。計數為零的上下文直接使用均勻分布。

相同的平滑參數對罕見上下文影響較大，對常見上下文影響較小。上下文次數增加時，觀察證據得到較大的權重；平滑強度增加時，機率則移向均勻分布。這解釋了保留資料曲線為何可能先改善再惡化：保護未見轉移的同時，最終也會稀釋有用的已知差異。應以驗證資料選擇參數，不要使用最終測試集調整。

分詞器與未知詞元規則也是實驗的一部分。把多個未見詞映射成同一標記，會合併它們的機率。這樣得到的較低困惑度，可能部分來自較容易預測的符號集合，並不表示模型能辨識那些原始詞語。

內插不同歷史長度的證據

長上下文很少見時，較短上下文通常有更多觀察。內插會結合它們的完整分布，而不是給每個未見延續相同的機率。

$$p_{\text{mix}}(w \mid u,v) = \lambda_3 p_3(w \mid u,v) + \lambda_2 p_2(w \mid v) + \lambda_1 p_1(w), \qquad \lambda_i \ge 0, \quad \sum_{i=1}^{3} \lambda_i = 1.$$

其中 u,v 是最近兩個詞元；w 是下一詞元；p_3, p_2, p_1 是正規化的 trigram、bigram 與 unigram 分布；p_{mix} 是混合分布；λ_i 是第 i 階的非負權重。每個成分都必須對查詢歷史有定義，例如先透過平滑處理。把所有候選詞元的機率加總會得到一，因為各成分分布總和為一，權重也總和為一。

對前面小範例的未見上下文，即使完整的兩詞元歷史缺少觀察，混合仍可以使用 `cat` 後面已觀察到的 bigram。固定混合權重容易檢查；隨歷史改變的權重則可以更信任證據充分的長上下文。以驗證文字選擇權重。若在檢查最終測試困惑度後才決定權重，就讓那個分數參與了模型選擇。

內插與「只有轉移缺少時才替換成較短歷史」不同。這類退回（back-off）規則還必須考慮已配置給觀察轉移的機率，才能讓完整分布保持正規化。Chen 與 Goodman 的比較研究檢視不同平滑選擇及實驗行為 [4]。本章的加法與混合範例用來揭示運作方式，沒有實作該研究中的每一種估計方法。

4. 保留資料困惑度量測什麼？

評分量測模型給觀察目標多少機率。好分數代表：在指定分詞器與詞彙表下，這些特定詞元序列讓模型比較不驚訝；它不能證明事實正確或理解能力。

$$\mathcal{L} = -\frac{1}{N}\sum_{j=1}^{N} \log p(w_j \mid h_j), \qquad \text{PPL} = \exp(\mathcal{L}) = \left(\prod_{j=1}^{N} \frac{1}{p(w_j \mid h_j)}\right)^{1/N}.$$

其中 N 是評分詞元位置數；j 遍歷所有評估句子的這些位置；w_j 是觀察目標；h_j 是允許使用的歷史；p 是被評估的模型；$\mathcal{L}$ 是每詞元以 nat 為單位的平均負對數概似；PPL 是困惑度（perplexity）。乘積形式顯示，困惑度是目標機率倒數的幾何平均，並不是每個位置合理延續的個數。

假設每個觀察目標都得到機率 $1/K$，困惑度就等於 K，其中 K 是這個假設分布中等可能選項的數目。實際預測不是均勻的：一個難以預測的目標可能造成很大的對數損失，許多容易目標則貢獻很少。只報平均值，可能掩蓋集中在罕見名稱或陌生領域的失敗。也應依詞頻與句型檢查損失。

跨批次彙整時，先加總各批次的**總**負對數概似，再除以**總**評分詞元數，最後取指數。直接平均批次困惑度，或對未加權的批次平均損失取指數，一般都不等於整個語料的困惑度。後者只有在各批次評分詞元數相同時才正確；填補與較短的最後一批很容易破壞此假設。以下合成計算直接來自上面的定義：

```
import math

# Synthetic evaluation batches: 2 targets of probability 1/2, then 8 of 1/4.
counts = [2, 8]
mean_nll = [math.log(2), math.log(4)]
corpus_ppl = math.exp(sum(n * loss for n, loss in zip(counts, mean_nll)) / sum(counts))
print('token-weighted perplexity:', round(corpus_ppl, 3))
print('mean batch perplexity:', round(sum(math.exp(loss) for loss in mean_nll) / 2, 3))
print('exp of mean batch loss:', round(math.exp(sum(mean_nll) / 2), 3))
assert math.isclose(corpus_ppl, (2**2 * 4**8)**(1 / 10))
```

```
token-weighted perplexity: 3.482
mean batch perplexity: 3.0
exp of mean batch loss: 2.828
```

Bigram 程式計入結束標記，因此停止同時屬於訓練及評估問題；起始標記則不計入。Unigram 與 bigram 結果可以比較，因為測試文字、分詞、未知映射及評分位置數一致。詞分詞器與字元分詞器的比較則有不同單位；即使原始句子相同，分數分母與預測符號集合也可能改變。

訓練、驗證與測試文字有不同角色。訓練決定詞彙表及次數；驗證選擇階數與平滑；測試則對選定流程評估一次。保留資料中未見於訓練的拼字，必須遵循固定未知規則。把測試文字納入重建詞彙表，會悄悄改變實驗。不同切分之間的重複句子或複製內容，也可能讓記住的局部延續看起來像泛化。

5. 生成、停止與記憶上限

生成反覆從正規化的下一詞元分布取樣，將取樣結果加入歷史，並在結束標記出現時停止。評分則由待評估文字提供觀察歷史。相同機率表支援兩種程序，但一種檢查提供的文字，另一種產生隨機軌跡。

示範輸出 `a bird sings` 只是固定隨機種子下的一次取樣，不是具代表性的品質估計。應檢查多個獨立種子的軌跡，並記錄局部錯誤及異常長的序列。長度上限保證程式停止執行；若在結束標記之前碰到上限，顯示的序列只是被截斷，並非模型完成的一個句子。

每一步選擇機率最大的詞元，會得到確定性的貪婪軌跡。這通常不會最大化整個序列的機率：局部看似有利的詞元，可能導向較差的後續延續。另一方面，取樣保留分布的多樣性，卻也可能選到低機率轉移。因此，生成選擇與模型是否給保留文字配置良好機率，是不同問題。

更多訓練也無法消除的失敗

考慮 `the robot near the cabinets is` 與 `the robots near the cabinet are`。一致關係取決於主詞中心語，不只是緊鄰動詞之前的名詞。若相關中心語在保留尾端之外，模型就會合併需要不同一致選擇的例子。更多觀察能改善合併估計，卻無法讓估計依賴模型輸入之外的詞元。

只要兩個前綴的最後 $n-1$ 個詞元相同，n-gram 模型的下一詞元分布就必須完全相同。這是架構上的等式，不是實驗趨勢。提高 n 會把記憶上限往後推，但可能歷史的數目與次數表的稀疏程度也會增加。共享的神經表示可以連結相似上下文，實際行為仍需要測試；光是替換次數表，並不能證明模型學會了一致關係。

6. 循環語言模型

白話版。 n-gram 模型在表格中查找歷史。循環語言模型則一次讀一個詞元，並保留一個固定大小的向量，也就是隱藏狀態（hidden state），在每個詞元之後更新。下一詞元分布由這個狀態計算，因此模型不再被迫忘記 $n-1$ 個詞元之前的所有內容。

Elman 的簡單循環網路提出把狀態回饋為下一步輸入的想法 [6]；Mikolov 等人則證明，訓練成詞層級語言模型的循環網路，在語音辨識上可以勝過強大的平滑 n-gram 基準 [7]。第 2 節的鏈式法則完全不變，改變的只有條件分布。對有起始標記 x_0 的詞元序列 $x_1, \dots, x_T$：

$$\begin{aligned} \mathbf{h}_t &= \tanh(W_x\,\mathbf{e}(x_{t-1}) + W_h\,\mathbf{h}_{t-1} + \mathbf{b}), \qquad \mathbf{h}_0 = \mathbf{0}, \\ \mathbf{z}_t &= W_o\,\mathbf{h}_t + \mathbf{c}, \\ p_\theta(x_t = v \mid x_{<t}) &= \operatorname{softmax}(\mathbf{z}_t)_v = \frac{\exp z_{t,v}}{\sum_{u \in V} \exp z_{t,u}}. \end{aligned}$$

其中 $\mathbf{e}(x) \in \mathbb{R}^d$ 是詞元 x 的嵌入（學習表格中的一列，如 NLP 04 所述）；$\mathbf{h}_t \in \mathbb{R}^m$ 是讀完 $x_0, \dots, x_{t-1}$ 後的隱藏狀態；$W_x \in \mathbb{R}^{m \times d}$、$W_h \in \mathbb{R}^{m \times m}$ 與 $\mathbf{b} \in \mathbb{R}^m$ 是循環參數；$W_o \in \mathbb{R}^{|V| \times m}$ 與 $\mathbf{c} \in \mathbb{R}^{|V|}$ 把狀態對應到每個詞彙項目 v 的分數 $z_{t,v}$；V 是預測詞彙表，u 遍歷其成員；θ 收集所有參數；tanh 逐元素作用。這與 ML 03A 的遞迴式相同，只是輸入改寫成嵌入，輸出改成整個詞彙表上的 softmax。

由方程式可直接得到三個結果。

- **條件不受限，記憶有上限。** $\mathbf{h}_t$ 是整個前綴的函數，因此最後 $n-1$ 個詞元相同的兩個前綴可以得到不同預測。但 $\mathbf{h}_t$ 只有 m 個數字。模型必須學會保留什麼；沒有任何保證它會保留某個依存關係所需的資訊。
- **參數共享。** 每個位置都使用相同的 W_x、W_h 與 W_o。參數量不隨上下文長度增加，而 n-gram 可能歷史的數目以 $|V|^{n-1}$ 成長。
- **透過相似性泛化。** 相似的詞元得到相似的嵌入，相似的前綴可以產生相似的狀態。未見歷史不再退回均勻的一列：它被對應到某個狀態，而由該狀態做出的預測可以重用相似歷史教給模型的內容。

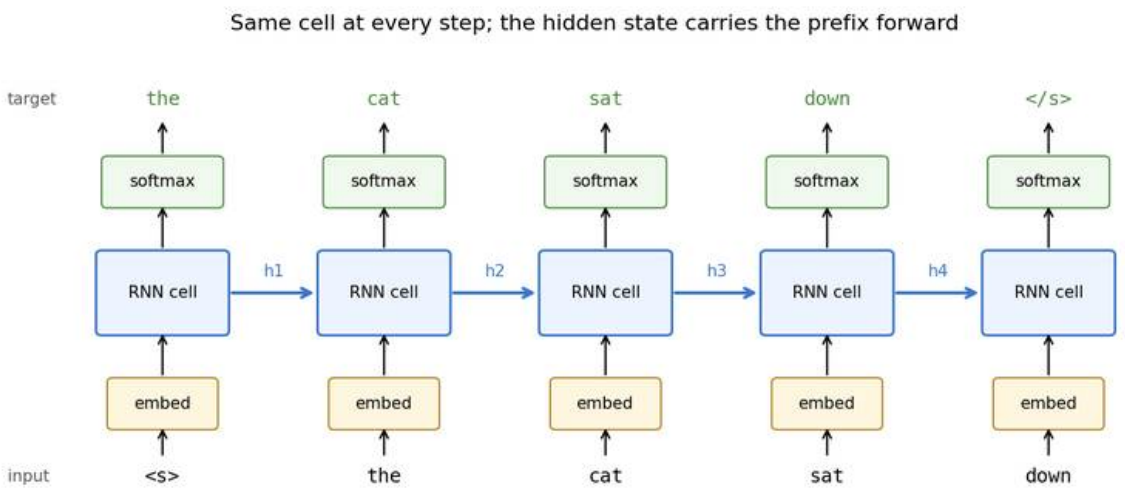


圖 2 — 展開五步的循環語言模型。每個輸入是前一個詞元，每個目標是下一個詞元，每一步都重用相同的單元參數。

訓練損失是下一詞元交叉熵

訓練最小化的量，正是第 4 節用於評估的量，只是現在寫成 θ 的可微函數：

$$\mathcal{L}(\theta) = -\frac{1}{N}\sum_{j=1}^{N} \log p_\theta(x_j \mid x_{<j}), \qquad \text{PPL} = \exp \mathcal{L}, \qquad \text{bits per token} = \frac{\mathcal{L}}{\ln 2}.$$

其中 N 是受監督的目標位置數（排除填補及其他忽略位置），x_j 是位置 j 的觀察目標，$x_{<j}$ 是其前綴，$\ln 2$ 把 nat 換算成 bit。每個位置的項，是 one-hot 目標與模型 softmax 分布之間的交叉熵；IT 01 說明交叉熵及其與編碼長度的關係。字元層級的研究常以每字元位元數（bits per character）取代困惑度；兩者攜帶相同資訊。

計數模型的兩項實作細節在這裡仍然適用。第一，**位移一格**：輸入序列是 $x_0, \dots, x_{T-1}$，目標序列是 $x_1, \dots, x_T$。若把同一位置的詞元同時當作輸入與目標，就會產生瑣碎的複製捷徑。第二，**教師強制（teacher forcing）**：訓練時每一步收到觀察到的前一個詞元，而不是模型自己較早的取樣。生成時，模型的取樣成為它的輸入，錯誤可能自我累積。因此，訓練損失與取樣行為需要分別檢查，正如第 5 節對 n-gram 模型的論證。

7. 時間上的訓練：梯度能到達哪裡

白話版。把遞迴展開後，模型成為一個權重共享的極深網路。某個預測的梯度必須穿過每一個中間步驟，才能到達本應影響它的詞元。距離很長時，梯度往往縮小到幾乎為零，或者較少見地爆炸。

時間反向傳播（backpropagation through time，BPTT）把一般反向傳播套用到展開後的計算圖，並把每個共享權重的梯度貢獻加總 [8]。較晚狀態對較早狀態的敏感度，是一步 Jacobian 的有序

乘積：

$$\frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_s} = \prod_{k=s+1}^{t} \operatorname{diag}(1 - \mathbf{h}_k^2)\, W_h,$$

其中 $s < t$ 是位置，$\mathbf{h}_k^2$ 逐元素平方，$\operatorname{diag}(\cdot)$ 以 tanh 導數組成對角矩陣。當各因子的範數通常小於一時，乘積隨距離 $t - s$ 指數衰減（梯度消失）；大於一時則可能指數成長（梯度爆炸）。Bengio、Simard 與 Frasconi 分析了為何以梯度下降學習長期依存關係很困難 [9]；Pascanu、Mikolov 與 Bengio 重新檢視這個問題，並推廣以裁剪梯度範數處理爆炸 [10]。ML 03A 完整推導並量測了這些結果。

這對語言代表什麼？回到第 5 節的一致關係例子：在 `the robots near the cabinet are` 中，動詞取決於往前四個詞的 `robots`，而中間的名詞指向另一個方向。普通 RNN 能夠表示記住複數主詞的狀態，但教它這麼做的訓練訊號，必須從 `are` 一路傳回 `robots`。在詞層級只有幾步；在字元層級，同一個依存關係大約跨越二十步。因此模型可能有足夠容量表示某個依存關係，卻仍學不會它；這與 n-gram 的架構上限是不同的失敗。

8. 閘控單元：LSTM 與 GRU

白話版。閘控單元加入一條主要以加法更新、並由學習開關控制的記憶路徑。資訊與梯度因此可以跨越許多步，而不必在每一步都被壓縮。

Hochreiter 與 Schmidhuber 的長短期記憶（long short-term memory，LSTM）單元 [11]，加上 Gers、Schmidhuber 與 Cummins 引入的遺忘閘 [12]，在隱藏狀態旁另外保留細胞狀態 $\mathbf{c}_t$：

$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \tilde{\mathbf{c}}_t, \qquad \mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{c}_t).$$

其中 $\mathbf{f}_t$、$\mathbf{i}_t$ 與 $\mathbf{o}_t$ 是遺忘閘、輸入閘與輸出閘（由 sigmoid 層根據目前輸入與 $\mathbf{h}_{t-1}$ 計算、落在 $(0,1)^m$ 的向量），$\tilde{\mathbf{c}}_t$ 是以相同方式計算的 tanh 候選值，$\odot$ 是逐元素乘法。沿著細胞路徑，$\partial \mathbf{c}_t / \partial \mathbf{c}_{t-1}$ 近似 $\operatorname{diag}(\mathbf{f}_t)$：沒有權重矩陣，也沒有壓縮函數。當遺忘閘維持在接近一時，在複數主詞處寫入的值可以保留到動詞出現。

Cho 等人的閘控循環單元（gated recurrent unit，GRU）[13] 合併細胞與隱藏狀態，並使用兩個閘：

$$\mathbf{h}_t = \mathbf{z}_t \odot \mathbf{h}_{t-1} + (1 - \mathbf{z}_t) \odot \tilde{\mathbf{h}}_t,$$

其中 $\mathbf{z}_t \in (0,1)^m$ 是更新閘，$\tilde{\mathbf{h}}_t$ 是以重設閘計算的候選狀態。（有些資料會對調 $\mathbf{z}_t$ 與 $1 - \mathbf{z}_t$ 的角色。）完整的閘門方程式、以 PyTorch 驗證的從零實作 LSTM 步驟及參數量，都在 ML 03A。對語言模型而言，實務結論是：LSTM 與 GRU 比普通 tanh RNN 穩定得多，兩者在不同任務上互有勝負，而閘控只是緩解、並未消除極長依存關係的困難。

9. 堆疊與雙向循環網路

白話版。循環層可以堆疊，讓較高層讀取下層的狀態序列。再加上一個由右向左執行的網路，每個位置也能看到它的未來；這對文字標註很有用，但在預測下一個詞元時是禁止的。

堆疊（深層）RNN 把第 ℓ 層的 $\mathbf{h}_1^{(\ell)}, \dots, \mathbf{h}_T^{(\ell)}$ 當作第 $\ell+1$ 層的輸入序列，softmax 讀取最上層。循環語言模型常用兩到四層；層與層之間的 dropout 是常見的正則化方式 [16]。

雙向 RNN 同時執行一個前向與一個反向網路，並串接兩者的狀態 $\mathbf{h}_t = [\vec{\mathbf{h}}_t; \overleftarrow{\mathbf{h}}_t]$ [14]。位置 t 的反向狀態已讀過 $x_{t+1}, \dots, x_T$。這非常適合標註：整個句子都已給定，而某個詞的標籤可能取決於右邊的詞；它是 NLP 06 的主要編碼器。但它對由左到右的語言模型是錯誤的：若用來預測 x_t 的狀態已經看過 x_t，損失就會獎勵複製，困惑度也失去意義。任何語言模型都可以做一個有用的健全性檢查：改變一個未來詞元，所有較早位置的預測都必須保持不變。

僅線上版：*Seq2Seq — Sequence to Sequence Learning with Neural Networks* — *https://blog.shyandram.dev/zh-tw/learn/nlp/recurrent-neural-networks/papers/seq2seq–sequence-to-sequence-learning-with-neural-networks/*

10. 字元層級語言模型

白話版。預測單位可以是字元而非詞。詞彙表變得極小，也永遠不會出現未知拼字；但序列會長上好幾倍，模型必須先學會拼字，才能學文法。

字元層級 RNN 曾是展示循環網路學到什麼的熱門例子；Graves 證明以字元訓練的 LSTM 可以生成較長且局部連貫的文字與手寫軌跡 [15]。與詞或子詞單位相比，取捨相當具體：

選擇	詞彙表	未知輸入	序列長度	長距離成本
詞	大；罕見詞需要未知標記	對應到同一標記，拼字遺失	最短	相關詞之間的步數最少
子詞（NLP 01）	中等，由分詞器固定	拆成已知片段	中等	中等
字元	極小的字母表	字母表上的任何拼字	最長	相關詞之間的步數很多

第 4 節的評估規則帶來兩點提醒。第一，字元困惑度與詞困惑度的單位不同；困惑度 11 的字元模型並不比困惑度 3.6 的詞模型「更好」，因為兩者在不同字母表上計算不同數量的預測。若要以共同文字單位比較，應對相同文字報告總負對數概似（或位元數），必要時再除以共同的位元組數。這要求相容、無損的文字前處理，並交代邊界慣例：把未知拼字合併成一個標記，並沒有編碼那些遺失字元。只改分母，無法補回評分前已擦除的資訊。第二，固定的字元字母表仍需要處理表外符號的規則，例如帶重音字母或表情符號。應在檢查保留文字之前決定。

11. 取樣與溫度

白話版。訓練好的模型替每個可能的下一詞元給出機率。生成時，我們抽出一個、附加上去，然後重複。溫度設定可以讓抽取更保守或更大膽，而不必重新訓練。

對 logit $\mathbf{z}_t$ 與溫度 $\tau > 0$，從下列分布取樣：

$$q_\tau(v) = \frac{\exp(z_{t,v}/\tau)}{\sum_{u \in V} \exp(z_{t,u}/\tau)}.$$

其中 v 與 u 遍歷詞彙表 V，$z_{t,v}$ 是模型在第 t 步給 v 的分數，q_τ 是取樣分布。$\tau = 1$ 時從模型自己的分布取樣；$\tau \to 0$ 時趨近於貪婪選擇最高分詞元；$\tau > 1$ 會壓平分布，讓不太可能的詞元得到更多權重。溫度只改變生成程序，不改變計算困惑度所用的機率，因此模型的保留資料分數在任何溫度下都相同。

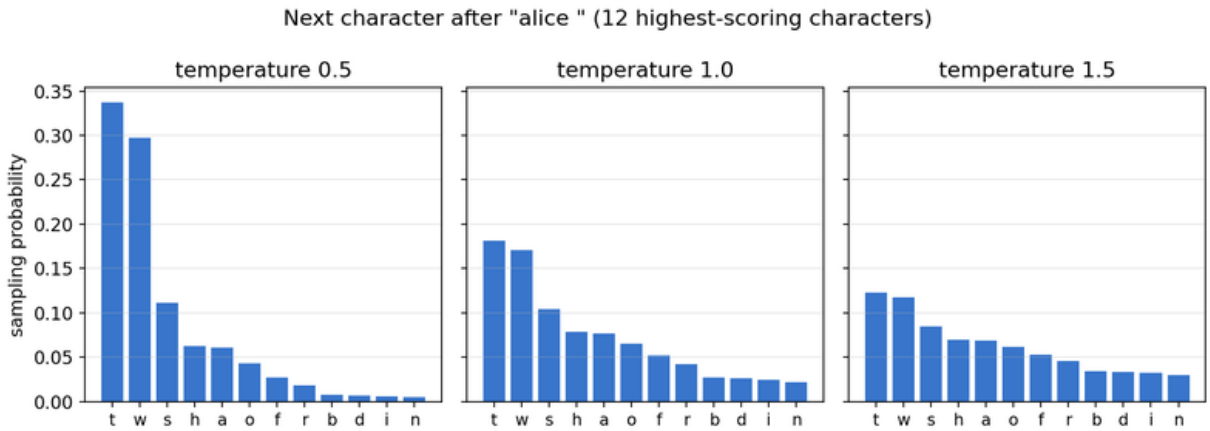


圖 3 — 第 12 節訓練好的 *LSTM* 在提示「*alice* 」之後、三種溫度下的下一字元分布。三個面板的 *logit* 完全相同，改變的只有取樣分布。

生成還涉及停止規則、長度限制，以及束搜尋（beam search）與 nucleus 取樣等搜尋策略。這些內容屬於 NLP 08 與 NLP 09。但有一項提醒現在就適用：一個樣本只是一條隨機軌跡。應以保留資料概似與任務檢查評判模型，樣本則用來尋找質性的失敗模式，而不是替模型排名。

12. 實務訓練，以及一個完整實驗

訓練檢查清單

白話版。大多數失敗的循環語言模型，失敗原因都很平凡：把填補當成文字計分、梯度爆炸、狀態在無關文件之間洩漏，或是用測試集挑選模型。每一項都有標準的補救方式。

- **把語料打包成串流。**對連續文字，把詞元串流切成 B 條等長的平行串流，排成 $B \times n$ 矩陣。每個訓練步驟讀取接下來的 T 欄。每個位置都是真實詞元，因此不需要填補。第 b 條串流會正好從上一個視窗停下的地方繼續。
- **填補與遮罩。**當範例是長度不同的獨立句子時，把它們填補到相同長度並保留布林遮罩。遮罩必須把填補位置排除在損失之外（否則模型會因預測填補而得到獎勵），對雙向編碼器還必須排除在狀態更新之外；PyTorch 的 `pack_padded_sequence` 負責後者。NLP 06 會在標註器中實作兩者。
- **截斷 BPTT。**反向傳播穿過數千步，記憶體成本隨長度線性成長。改為把隱藏狀態從一個 T 步視窗帶到下一個視窗，但將它分離（*detach*），讓前向計算仍看到完整過去，而梯度最多只往回流 T 步。
- **梯度裁剪。**若梯度範數 $\|\mathbf{g}\|$ 超過門檻 κ，就把它縮放成 $\kappa\,\mathbf{g}/\|\mathbf{g}\|$ [10]。這會保留更新方向並限制其長度。

- **正則化與提前停止**。在非循環連接上使用 dropout [16] 與權重衰減，可以限制記憶；選擇驗證困惑度最低的訓練週期，最後只報告一次測試困惑度。
- **公平的基準**。 Melis、Dyer 與 Blunsom 發現，在給予相當的超參數搜尋後，仔細調校的標準 LSTM 勝過幾種較新、較複雜的架構 [17]。同樣的原則在小規模也成立：在宣稱神經模型勝出之前，先用同一份驗證資料調校 n-gram 基準。

程式碼：字元層級 LSTM 對上內插 n-gram

文字是 Lewis Carroll《愛麗絲夢遊仙境》（*Alice's Adventures in Wonderland*，1865）第一章的開頭，屬於公有領域；取自 Project Gutenberg 電子書 #11 [24]，移除斜體底線、把彎引號改成直引號、把破折號寫成 `--`，再轉成小寫。第 1–7 段是訓練資料，第 8 段是驗證資料，第 9 段是測試集。字母表事先固定，在檢查保留文字之前就決定。兩個模型都以所有較早的文字作為每個保留字元的條件。n-gram 基準完全依照第 3 節的方式，內插經平滑的 unigram、bigram 與 trigram 字元分布，權重與平滑強度在驗證資料上選擇。LSTM 以不同種子訓練三次，因為在這麼少的資料上，單次結果可能誤導。整個程式在 CPU 上遠少於一分鐘即可完成，也不下載任何東西。

```
import copy
import math
from collections import Counter, defaultdict

import torch
from torch import nn

# Opening paragraphs of Chapter I of Lewis Carroll's "Alice's Adventures in
# Wonderland" (1865, public domain; Project Gutenberg eBook #11). Italic
# underscores removed, curly quotes straightened, em dashes written as "--".
PARAGRAPHS = [
    'Alice was beginning to get very tired of sitting by her sister on the '
    'bank, and of having nothing to do: once or twice she had peeped into the '
    'book her sister was reading, but it had no pictures or conversations in '
    'it, "and what is the use of a book," thought Alice "without pictures or '
    'conversations?"',
    'So she was considering in her own mind (as well as she could, for the hot '
    'day made her feel very sleepy and stupid), whether the pleasure of making '
    'a daisy-chain would be worth the trouble of getting up and picking the '
    'daisies, when suddenly a White Rabbit with pink eyes ran close by her.',
    'There was nothing so very remarkable in that; nor did Alice think it so '
    'very much out of the way to hear the Rabbit say to itself, "Oh dear! Oh '
    'dear! I shall be late!" (when she thought it over afterwards, it occurred '
    'to her that she ought to have wondered at this, but at the time it all '
    'seemed quite natural); but when the Rabbit actually took a watch out of '
    'its waistcoat-pocket, and looked at it, and then hurried on, Alice started'
    ' to her feet, for it flashed across her mind that she had never before '
    'seen a rabbit with either a waistcoat-pocket, or a watch to take out of '
    'it, and burning with curiosity, she ran across the field after it, and '
    'fortunately was just in time to see it pop down a large rabbit-hole under '
    'the hedge.',
    'In another moment down went Alice after it, never once considering how in '
    'the world she was to get out again.',
    'The rabbit-hole went straight on like a tunnel for some way, and then '
    'dipped suddenly down, so suddenly that Alice had not a moment to think '
```

```
    'about stopping herself before she found herself falling down a very deep '
    'well.',
    'Either the well was very deep, or she fell very slowly, for she had plenty'
    ' of time as she went down to look about her and to wonder what was going '
    'to happen next. First, she tried to look down and make out what she was '
    'coming to, but it was too dark to see anything; then she looked at the '
    'sides of the well, and noticed that they were filled with cupboards and '
    'book-shelves; here and there she saw maps and pictures hung upon pegs. She'
    ' took down a jar from one of the shelves as she passed; it was labelled '
    '"ORANGE MARMALADE", but to her great disappointment it was empty: she did '
    'not like to drop the jar for fear of killing somebody underneath, so '
    'managed to put it into one of the cupboards as she fell past it.',
    '"Well!" thought Alice to herself, "after such a fall as this, I shall '
    "think nothing of tumbling down stairs! How brave they'll all think me at "
    "home! Why, I wouldn't say anything about it, even if I fell off the top of"
    ' the house!" (Which was very likely true.)',
    'Down, down, down. Would the fall never come to an end? "I wonder how many '
    'miles I\'ve fallen by this time?" she said aloud. "I must be getting '
    'somewhere near the centre of the earth. Let me see: that would be four '
    'thousand miles down, I think--" (for, you see, Alice had learnt several '
    'things of this sort in her lessons in the schoolroom, and though this was '
    'not a very good opportunity for showing off her knowledge, as there was no'
    ' one to listen to her, still it was good practice to say it over) "--yes, '
    "that's about the right distance--but then I wonder what Latitude or "
    'Longitude I\'ve got to?" (Alice had no idea what Latitude was, or Longitude'
    ' either, but thought they were nice grand words to say.)',
    'Presently she began again. "I wonder if I shall fall right through the '
    "earth! How funny it'll seem to come out among the people that walk with "
    'their heads downward! The Antipathies, I think--" (she was rather glad '
    "there was no one listening, this time, as it didn't sound at all the right"
    ' word) "--but I shall have to ask them what the name of the country is, '
    'you know. Please, Ma\'am, is this New Zealand or Australia?" (and she tried'
    " to curtsey as she spoke--fancy curtseying as you're falling through the "
    'air! Do you think you could manage it?) "And what an ignorant little girl '
    "she'll think me for asking! No, it'll never do to ask: perhaps I shall see"
    ' it written up somewhere."',
]

paras = [p.lower() for p in PARAGRAPHS]
train = "\n".join(paras[:7]) + "\n"     # paragraphs 1-7: fit parameters
valid = paras[7] + "\n"                 # paragraph 8: choose settings
test = paras[8] + "\n"                  # paragraph 9: report once
ALPHABET = "\n !\"'(),-.:;?abcdefghijklmnopqrstuvwxyz"  # fixed in advance
V = len(ALPHABET)
assert set(train + valid + test) <= set(ALPHABET)
print("characters train/valid/test:", len(train), len(valid), len(test))
print("letters never seen in training:", sorted(set(valid + test) - set(train)))

# ---- Baseline: interpolated character n-grams (orders 1-3) ----
def count(order):
    rows = defaultdict(Counter)
    for t in range(len(train)):
        rows[train[max(0, t - order + 1):t]][train[t]] += 1
    return rows

tables = {k: count(k) for k in (1, 2, 3)}
```

```
def p_add(k, history, ch, alpha):
    row = tables[k].get(history[len(history) - k + 1:] if k > 1 else "", {})
    return (row.get(ch, 0) + alpha) / (sum(row.values()) + alpha * V)

def ngram_ppl(prefix, target, lams, alpha):
    full, nll = prefix + target, 0.0
    for pos in range(len(prefix), len(full)):
        h = full[max(0, pos - 2):pos]
        p = sum(l * p_add(k, h, full[pos], alpha) for k, l in zip((1, 2, 3), lams))
        nll -= math.log(p)
    return math.exp(nll / len(target))

grid = [(l1, l2, round(1 - l1 - l2, 2)) for l1 in (0.1, 0.2, 0.3)
        for l2 in (0.2, 0.3, 0.4, 0.5) if l1 + l2 < 1]
best_cfg = min(((ngram_ppl(train, valid, lams, a), lams, a) for lams in grid
                for a in (0.01, 0.1, 0.5)), key=lambda r: r[0])
ng_valid, ng_lams, ng_alpha = best_cfg
ng_test = ngram_ppl(train + valid, test, ng_lams, ng_alpha)
print(f"n-gram  weights={ng_lams} alpha={ng_alpha}  "
      f"valid PPL={ng_valid:.2f}  test PPL={ng_test:.2f}")

tri_alpha = min((0.01, 0.1, 0.5), key=lambda a: ngram_ppl(train, valid, (0, 0, 1), a))
tri_test = ngram_ppl(train + valid, test, (0, 0, 1), tri_alpha)
print(f"trigram only  alpha={tri_alpha}  test PPL={tri_test:.2f}")

# ---- Character-level LSTM language model ----
stoi = {c: i for i, c in enumerate(ALPHABET)}
encode = lambda s: torch.tensor([stoi[c] for c in s])

class CharLSTM(nn.Module):
    def __init__(self, vocab, emb=24, hidden=96, dropout=0.3):
        super().__init__()
        self.emb = nn.Embedding(vocab, emb)
        self.rnn = nn.LSTM(emb, hidden, batch_first=True)
        self.drop = nn.Dropout(dropout)
        self.out = nn.Linear(hidden, vocab)

    def forward(self, x, state=None):
        h, state = self.rnn(self.emb(x), state)
        return self.out(self.drop(h)), state

def lstm_ppl(model, prefix, target):
    model.eval()
    with torch.no_grad():
        ids = encode(prefix + target).unsqueeze(0)
        logits, _ = model(ids[:, :-1])          # condition on all earlier text
        nll = nn.functional.cross_entropy(logits[0, len(prefix) - 1:],
                                          ids[0, len(prefix):])
    model.train()
    return math.exp(nll.item())

def train_lstm(seed, epochs=80, B=16, T=48):
    torch.manual_seed(seed)
    model = CharLSTM(V)
    opt = torch.optim.Adam(model.parameters(), lr=3e-3, weight_decay=1e-4)
```

```
    ids = encode(train)
    n = (len(ids) - 1) // B                    # B parallel streams of n characters
    X, Y = ids[:n * B].view(B, n), ids[1:n * B + 1].view(B, n)  # targets shifted by one
    history, best = [], (float("inf"), None, 0)
    for epoch in range(1, epochs + 1):
        state, total, steps = None, 0.0, 0
        for i in range(0, n, T):
            if state is not None:              # truncated BPTT: keep values, cut graph
                state = tuple(s.detach() for s in state)
            logits, state = model(X[:, i:i + T], state)
            loss = nn.functional.cross_entropy(logits.reshape(-1, V),
                                               Y[:, i:i + T].reshape(-1))
            opt.zero_grad()
            loss.backward()
            nn.utils.clip_grad_norm_(model.parameters(), 1.0)
            opt.step()
            total, steps = total + loss.item(), steps + 1
        v = lstm_ppl(model, train, valid)
        history.append((epoch, math.exp(total / steps), v))
        if v < best[0]:                        # early stopping on validation only
            best = (v, copy.deepcopy(model.state_dict()), epoch)
    model.load_state_dict(best[1])
    return model, best, history

runs = {}
for seed in (0, 1, 2):
    m, best, hist = train_lstm(seed)
    runs[seed] = (m, hist, lstm_ppl(m, train + valid, test))
    print(f"LSTM seed={seed}  best epoch={best[2]}  valid PPL={best[0]:.2f}  "
          f"test PPL={runs[seed][2]:.2f}")
mean_test = sum(r[2] for r in runs.values()) / len(runs)
print(f"LSTM mean test PPL over 3 seeds: {mean_test:.2f}")
assert mean_test < tri_test
model = runs[0][0]

# ---- Sampling at several temperatures ----
def sample(prompt, tau, length=80, seed=1):
    g = torch.Generator().manual_seed(seed)
    model.eval()
    with torch.no_grad():
        logits, state = model(encode(prompt).unsqueeze(0))
        out = prompt
        for _ in range(length):
            probs = torch.softmax(logits[0, -1] / tau, dim=-1)
            nxt = torch.multinomial(probs, 1, generator=g)
            out += ALPHABET[nxt.item()]
            logits, state = model(nxt.view(1, 1), state)
    return out.replace("\n", " / ")

for tau in (0.5, 1.0, 1.5):
    print(f"tau={tau}:", sample("alice ", tau))

# ---- Same last two characters, different earlier context ----
def next_dist(prefix):
    model.eval()
    with torch.no_grad():
```

```
        logits, _ = model(encode(prefix).unsqueeze(0))
    return torch.softmax(logits[0, -1], dim=-1)

a, b = "alice was beginning to get ", "the rabbit-hole went "
assert a[-2:] == b[-2:]                    # a trigram sees only these two
def ngram_dist(prefix):
    return torch.tensor([sum(l * p_add(k, prefix[-2:], c, ng_alpha)
                             for k, l in zip((1, 2, 3), ng_lams)) for c in ALPHABET])

tv = lambda p, q: 0.5 * (p - q).abs().sum().item()
print("total variation, n-gram:", round(tv(ngram_dist(a), ngram_dist(b)), 3),
      " LSTM:", round(tv(next_dist(a), next_dist(b)), 3))
```

輸出：

```
characters train/valid/test: 2633 706 682
letters never seen in training: ['z']
n-gram  weights=(0.1, 0.4, 0.5) alpha=0.01  valid PPL=9.70  test PPL=11.24
trigram only  alpha=0.1  test PPL=13.37
LSTM seed=0  best epoch=43  valid PPL=10.13  test PPL=12.09
LSTM seed=1  best epoch=41  valid PPL=9.76  test PPL=11.05
LSTM seed=2  best epoch=39  valid PPL=10.09  test PPL=11.30
LSTM mean test PPL over 3 seeds: 11.48
tau=0.5: alice whas she sor of the to her fear the she was she sely and it she save of the
 ↪  was
tau=1.0: alice whas nyt ook "hente tome urabuint, mubbeit-insiwselly bulith ver to welf was
 ↪  bor
tau=1.5: alice wherany fowh ".pvenst gime feurdt, )ubger and iwselly ?hont  ver, vermef wamen
 ↪  r
total variation, n-gram: 0.0  LSTM: 0.197
```

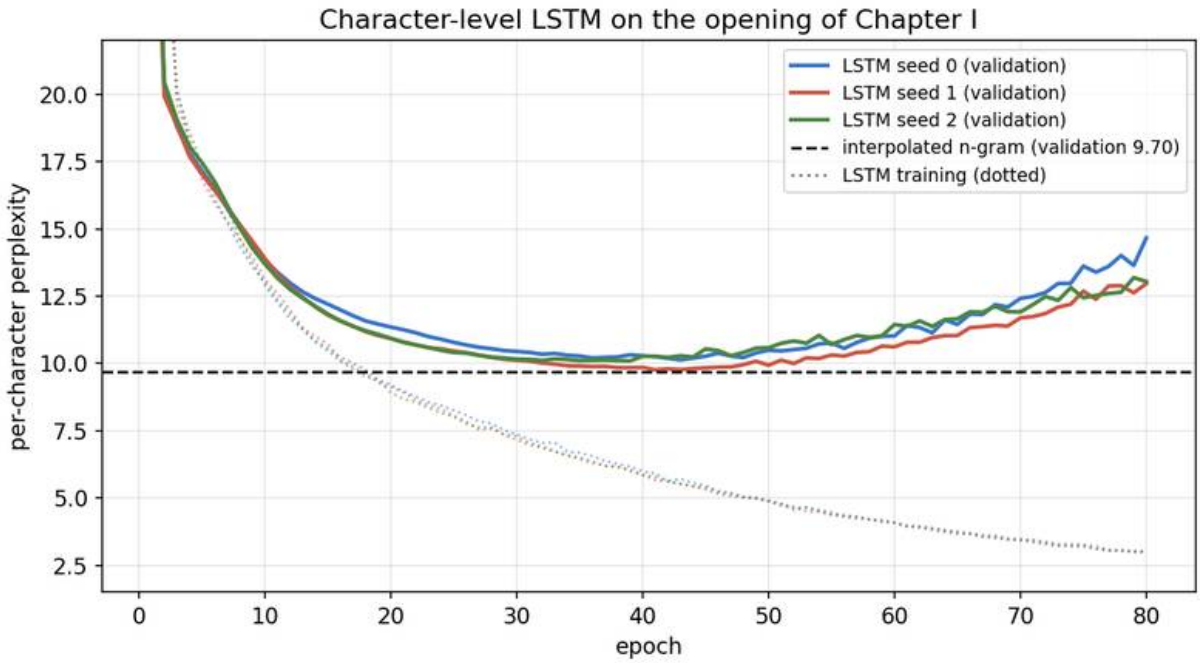


圖 *4* — 訓練過程中的每字元困惑度。訓練困惑度持續下降；驗證困惑度在第 *40* 週期附近達到最低，之後因模型記住 *2,633* 個訓練字元而回升。虛線是在同一驗證段落上調校的內插 *n-gram*。

解讀結果

基準很重要。使用加法平滑的單一 trigram 測試困惑度為 13.37，三次 LSTM 都勝過它。在同一驗證段落上調校的內插 n-gram 達到 11.24，而三次 LSTM 分別是 12.09、11.05 與 11.30（平均 11.48）。在 2,633 個訓練字元上，循環模型大致與平滑良好的計數模型打平，並明顯勝過平滑不佳的計數模型。這是 Melis 等人論點的小資料版本 [17]：對未調校基準宣稱的改善，面對調校過的基準可能就消失了。隨著資料增加，神經語言模型會拉開差距，因為共享表示能利用許多稀疏表格無法合併的微弱規律；Jozefowicz 等人記錄了循環語言模型在大型語料與模型上的進展 [19]。我們的實驗太小，無法呈現那種情況，也不需要：它示範的是如何誠實地比較。

種子很重要。不同種子之間的差異（11.05 到 12.09），大於 LSTM 平均值與 n-gram 之間的差距。若只用一個種子，我們可能報告「LSTM 勝出」，也可能報告「LSTM 落敗」。差異這麼小時，應報告多次執行結果，或對保留文件計算信賴區間。

驗證曲線顯示記憶現象。在圖 4 中，訓練困惑度持續下降，遠低於驗證值；模型正在學習特定的訓練段落。提前停止選出第 39–43 週期；若沒有它，最後一個週期在保留文字上會差上不少。

單位。這些是 39 個符號字母表上的每字元困惑度；n-gram 的 $\log_2 11.24 \approx 3.49$ 每字元位元。它們不能與第 3 節的詞層級數值（例如 3.573）比較，因為後者在不同詞彙表上計算不同事件。

量測記憶上限。最後幾行比較 `alice was beginning to get` 與 `the rabbit-hole went` 之後的下一字元分布，這兩個前綴的最後兩個字元相同。內插 trigram 必須給出完全相同的分布（總變異距離 0.0），正如第 5 節所證明。LSTM 的分布則不同（總變異距離 0.197），因為它的狀態取

決於更早的字元。這確認循環模型有使用較長的上下文；至於是否用得好，則是另一個問題，要由保留資料困惑度與針對性測試回答。

樣本。在 $\tau = 0.5$ 時，樣本重複 `she was`、`of the` 等常見片段；在 $\tau = 1.0$ 時出現詞的形狀，但拼錯很多；在 $\tau = 1.5$ 時輸出瓦解成罕見標點與隨機字母。一個只用不到三千個字元訓練的字元模型，學到的是常見字母序列、空白與一些短詞，並沒有學會英語。這些都是單一種子的單次抽樣，應視為溫度效果的示意，而不是品質估計。

現代觀點

綜述觀點。 Jurafsky 與 Martin 的 2026 年 8 月草稿仍保留 n-gram 專章，依機率預測、估計、評估、生成及平滑來組織 [3]。它提供有效的概念橋梁：讀者可以先檢查局部次數，再接觸共享可學習表示的模型。Chen 與 Goodman 的平滑比較則解釋了為何除了歷史長度，估計方法本身也很重要 [4]。兩種來源用途不同：教材發展模型，實證研究比較如何在稀疏證據下配置機率。Tiezzi 等人的 2024 年綜述回顧循環處理為何在 Transformer 時代重新受到重視 [20]。它把兩股趨勢放在「循環」這個統一框架下：一類結合 Transformer 與循環網路想法、以避開自注意力的二次方成本；另一類是作為特殊線性 RNN 的深層狀態空間模型。它提出的主要開放問題，是從長度事先未知的串流中學習，這把序列建模連到終身線上學習。

循環語言模型的成就。從 Mikolov 的 RNN 語言模型 [7]，到大規模 LSTM 語言模型 [19] 與仔細正則化的 LSTM [18]，循環網路一直是最強的通用語言模型，直到 Transformer 在多數大規模工作中取代它們。Merity 等人的 AWD-LSTM 研究顯示，那份實力有很大一部分來自正則化與最佳化選擇，而不是新的單元設計 [18]；Melis 等人透過受控的超參數搜尋得到相似結論 [17]。

2023–2026 年。循環以新形式回歸，目標是降低長上下文注意力的成本。RWKV 重新表述類 Transformer 模型，使它能平行訓練，並在推論時以 RNN 方式執行 [21]。Mamba 使用依輸入而變的（選擇性）狀態空間更新，並以平行掃描計算 [22]。xLSTM 以指數閘控與新的記憶結構重新檢視 LSTM [23]。這些模型保留了讓 RNN 語言模型具吸引力的性質：固定大小的狀態，因此每個詞元的生成成本不隨上下文長度增加。它們也繼承了第 7 節提出的問題：固定大小的狀態必須決定要忘記什麼。ML 03A 對狀態空間模型有更詳細的介紹。

評估。 Chang 等人的 2024 年綜述以「評估什麼、在哪裡評估、如何評估」組織大型語言模型評估 [5]，涵蓋詞元預測之外的任務行為與評估方法。套用到本章，較低困惑度仍應搭配真正重要的任務檢查，例如拼字恢復、一致關係、語意保存或適當停止規則。在 2023–2026 年的工作流程中，n-gram 基準的實際作用是讓問題可以量測。更複雜的模型是否在相同分詞下改善保留預測？是否區分了既定 n-gram 歷史必須合併的兩個前綴？增加記憶體與延遲是否改善下游結果？應用可比較的實驗回答，而不是比較條件不一致的分數，或假設較長上下文一定被有效利用。

開放問題。開放問題包括領域改變後的稀疏證據、新名稱的機率，以及分詞與量測難度之間的互動。字元或子詞會改變詞彙覆蓋與序列長度的取捨。更長的歷史，無論存在表格或循環狀態中，都提供更多資訊，卻也帶來更困難的估計與功勞分配（credit assignment）問題。應一起選擇預測單位與保留歷史，並在解讀結果之前說明限制。

重點整理

- n-gram 出現或文件特徵，不會自動成為正規化語言模型；第 n 階使用前 $n-1$ 個詞元，而機率鏈式法則本身沒有要求有限記憶。
- 平滑保護未觀察事件，內插重用較短歷史的證據；兩者都無法恢復模型已捨棄的歷史。
- 比較困惑度需要一致的預測單位、未知映射與邊界規則；字元與詞的困惑度不能比較。
- 循環語言模型以依賴整個前綴的學習狀態取代次數表，參數在各位置之間共享。
- BPTT 梯度隨距離縮小或放大；裁剪處理爆炸，LSTM 或 GRU 的閘控緩解梯度消失，但沒有消除長依存關係的困難。
- 雙向 RNN 適合標註，不適合由左到右的語言模型。
- 實務訓練需要位移目標、遮罩或打包串流、截斷 BPTT、梯度裁剪、正則化，以及以驗證資料為準的提前停止。
- 在極小資料上，調校過的內插 n-gram 可以與 LSTM 打平；應與調校過的基準比較，並報告多個種子。

練習

1. 手算第 2 節第一個程式中，`a cat` 與 `the cat` 後面的平滑 trigram 機率。確認八個目標詞元上的總和。

   提示 每個歷史只有一筆觀察。分母是一加上平滑強度乘以八；只有觀察到的延續額外得到一次計數。

2. 混合第一個範例的三種階數。任選總和為一的非負權重，並對未見兩詞元歷史驗證正規化。

   提示 讓每個成分都使用相同候選詞彙表。比較未見歷史下的機率、純均勻退回，以及 `cat` 後面觀察到的 bigram。

3. 分別在計入與不計入結束標記的條件下，替 bigram 測試集評分。解釋為什麼兩種困惑度回答不同預測問題。

   提示 一致地改變累積損失與位置數。不計入結束標記，移除了停止行為的評估，不只是改變顯示慣例。

4. 建立兩個最後兩詞元相同的前綴，讓它們因更早的否定、一致關係或篇章資訊，而需要不同的下一詞元機率。說明 trigram 能與不能學到什麼，以及 RNN 語言模型原則上能學到什麼。

   提示 先確認保留歷史完全相同，再區分資料不足與根本無法依省略的差異設定條件這兩種問題。對 RNN，要區分它的狀態能表示什麼，以及它的梯度能教會它什麼。

5. 在字元層級實驗中，把 `nn.LSTM` 換成 `nn.GRU`（狀態變成單一張量，所以要調整 detach 那一行）與 `nn.RNN`。以三個種子比較驗證與測試困惑度。接著移除普通 RNN 的梯度裁剪，記錄訓練損失發生什麼事。

提示 `nn.GRU` 與 `nn.RNN` 的狀態是張量而不是元組。保持其他所有設定不變，包括提前停止規則，讓單元類型成為唯一的改變。

6. 刻意破壞因果性：給 LSTM 模型在同一輸入上加一個反向 LSTM（雙向層）並重新訓練。解釋你觀察到的訓練與驗證困惑度，並寫一個本來能抓到這個錯誤的測試。

提示 有了反向計算，用來預測位置 t 字元的狀態已經讀過該字元。好的測試會改變一個未來字元，並檢查所有較早位置的預測分布保持不變。

參考文獻

1. Masato Hagiwara, *Real-World Natural Language Processing: Practical Applications with Deep Learning*, Chapter 5, Manning, 2021. Publisher
2. Susan McRoy, *Principles of Natural Language Processing*, 2021. Chapter
3. Daniel Jurafsky and James H. Martin, *Speech and Language Processing*, third-edition draft, Chapter 3, August 19, 2026. Chapter
4. Stanley F. Chen and Joshua Goodman, “An Empirical Study of Smoothing Techniques for Language Modeling,” ACL, 1996. Paper
5. Yupeng Chang et al., “A Survey on Evaluation of Large Language Models,” ACM Transactions on Intelligent Systems and Technology, 2024. Paper
6. Jeffrey L. Elman, “Finding Structure in Time,” *Cognitive Science* 14(2), 1990. DOI
7. Tomáš Mikolov, Martin Karafiát, Lukáš Burget, Jan Černocký and Sanjeev Khudanpur, “Recurrent Neural Network Based Language Model,” Interspeech, 2010. DOI
8. Paul J. Werbos, “Backpropagation Through Time: What It Does and How to Do It,” *Proceedings of the IEEE* 78(10), 1990. DOI
9. Yoshua Bengio, Patrice Simard and Paolo Frasconi, “Learning Long-Term Dependencies with Gradient Descent Is Difficult,” *IEEE Transactions on Neural Networks* 5(2), 1994. DOI
10. Razvan Pascanu, Tomas Mikolov and Yoshua Bengio, “On the Difficulty of Training Recurrent Neural Networks,” ICML, 2013. arXiv
11. Sepp Hochreiter and Jürgen Schmidhuber, “Long Short-Term Memory,” *Neural Computation* 9(8), 1997. DOI
12. Felix A. Gers, Jürgen Schmidhuber and Fred Cummins, “Learning to Forget: Continual Prediction with LSTM,” *Neural Computation* 12(10), 2000. DOI
13. Kyunghyun Cho et al., “Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation,” EMNLP, 2014. arXiv
14. Mike Schuster and Kuldip K. Paliwal, “Bidirectional Recurrent Neural Networks,” *IEEE Transactions on Signal Processing* 45(11), 1997. DOI
15. Alex Graves, “Generating Sequences With Recurrent Neural Networks,” arXiv:1308.0850, 2013. arXiv
16. Wojciech Zaremba, Ilya Sutskever and Oriol Vinyals, “Recurrent Neural Network Regularization,” arXiv:1409.2329, 2014. arXiv
17. Gábor Melis, Chris Dyer and Phil Blunsom, “On the State of the Art of Evaluation in Neural Language Models,” arXiv:1707.05589, 2017. arXiv

18. Stephen Merity, Nitish Shirish Keskar and Richard Socher, “Regularizing and Optimizing LSTM Language Models,” arXiv:1708.02182, 2017. arXiv
19. Rafal Jozefowicz, Oriol Vinyals, Mike Schuster, Noam Shazeer and Yonghui Wu, “Exploring the Limits of Language Modeling,” arXiv:1602.02410, 2016. arXiv
20. Matteo Tiezzi, Michele Casoni, Alessandro Betti, Tommaso Guidi, Marco Gori and Stefano Melacci, “On the Resurgence of Recurrent Models for Long Sequences: Survey and Research Opportunities in the Transformer Era,” arXiv:2402.08132, 2024. arXiv
21. Bo Peng et al., “RWKV: Reinventing RNNs for the Transformer Era,” Findings of EMNLP, 2023. Paper
22. Albert Gu and Tri Dao, “Mamba: Linear-Time Sequence Modeling with Selective State Spaces,” COLM, 2024. arXiv
23. Maximilian Beck et al., “xLSTM: Extended Long Short-Term Memory,” NeurIPS, 2024. Paper
24. Lewis Carroll, *Alice’s Adventures in Wonderland*, 1865; Project Gutenberg eBook #11. Project Gutenberg
25. Ilya Sutskever, Oriol Vinyals and Quoc V. Le, “Sequence to Sequence Learning with Neural Networks,” NeurIPS, 2014. Paper

NLP 06・以 RNN 做序列標註（詞性標註、資訊擷取）

替句子的每個詞元加上標籤：以 HMM 與 Viterbi 解碼做詞性標註，以 BIO 標籤與精確區段評估做實體辨識，並用 BiLSTM、字元特徵與 CRF 輸出層建立神經標註器。

網站版本: https://blog.shyandram.dev/zh-tw/learn/nlp/sequence-labeling/

你將學到

- 把詞性標註與實體辨識表示成序列標註，並說明為何獨立的詞元判定可能與合法的整句分析不一致。
- 推導隱藏馬可夫序列分數與最大路徑（Viterbi）解碼，並把它們連到條件隨機場。
- 以可重現的位移定義實體結構描述，用 BIO 標籤編碼提及，並以精確區段而非詞元準確率評估。
- 建立 BiLSTM 標註器，加入字元層級特徵與 CRF 輸出層，並以遮罩前向演算法的對數概似訓練 CRF。
- 把實體標註放進資訊擷取（實體、關係、事件）的全貌，並認識標準資料集。
- 執行完整的 PyTorch 實驗，在訓練中從未出現的名稱上量測區段 F1、錯誤類型與非法標籤轉移。

先看全貌

許多語言任務要求每個詞元一個標籤。詞性（part-of-speech，POS）標註器把每個詞標成名詞、動詞、形容詞等；實體辨識器標出哪些詞元屬於人名、組織名或地名，以及每個名稱從哪裡開始、到哪裡結束。兩者看起來像沿著句子重複進行分類，但判定彼此並不獨立：某個詞的正確標籤，既取決於鄰近的詞，也取決於鄰近的標籤。把整個標籤序列視為一個結構化輸出，是本章的核心想法。

McRoy 第 3、4 章介紹語法類別與語法處理，第 8 章連結語言處理與資訊擷取 [1]。Hagiwara 第 5 章發展序列標記，包括詞性標記與命名實體辨識 [2]。本章結合這些觀點：標註器為各位置附上類別；擷取器則把選定區段轉成應用程式能檢查的紀錄。輸出契約十分重要：客服系統可能需要產品名稱與抱怨事件，歷史檔案則需要人物、組織、日期與帶有消息來源的陳述。

> **用最簡單的話說**
>
> 給小朋友一個句子和三支螢光筆：黃色畫人、綠色畫公司、藍色畫地點。每個詞各自上色是不夠的。「Port」在「Port Lena」裡是藍色，在「港口裡的一艘船」這種用法裡就不是；兩個字的名稱如果後半畫成綠色，前半最好也是綠色。好的螢光筆會看完整個句子，讓顏色彼此一致。

實際風險是轉換時遺失重要差別。「供應商說裝置通過了測試」不能證明裝置確實通過。系統即使抓對所有名字，仍可能因為刪除消息歸屬或可能性而建立錯誤紀錄。因此，要同時評估表面擷取

與解讀；格式正確的輸出不是內容正確的證據。結構化預測的數學（HMM、前向—後向演算法、Viterbi 解碼與一般形式的 CRF）在 PR 06 發展；循環編碼器來自 NLP 05。剖析、語意角色，以及深入的關係與事件擷取，則在 NLP 12 介紹。

詞、詞法與語法類別

白話版。詞的類別描述它在句子中的行為。拼字能提供線索，但脈絡通常決定實際用法。

詞法（morphology）研究詞內結構。複數詞尾、時態標記與派生前綴能提供有用證據，但不同語言不一定用相同片段或詞邊界表達性質。為了建模效率學出的子詞 token，也不一定對應語言學上的詞素。應區分模型斷詞與標註使用的語法單位。

詞性（part of speech）描述名詞、動詞等語法類別。名詞通常是名詞片語的中心，動詞通常是子句述語的中心。這是分布上的描述，不是每個名詞都必須指實體物品的規則。「Decision」與「arrival」雖指抽象內容與事件，仍是名詞。

相同表面詞可以有不同類別。在原創例句「the pilots can land」與「the metal can fell」中，「can」扮演不同角色。第二句也顯示，句子其餘部分改變時，熟悉的局部片語可能誤導。標記器必須考慮允許的脈絡與標註規則。

已標記句子可表示為：

$$\mathbf{x} = (x_1, \dots, x_T), \qquad \mathbf{y} = (y_1, \dots, y_T), \qquad y_t \in \mathcal{Y}.$$

其中 $\mathbf{x}$ 是輸入詞序列，x_t 是位置 t 的詞，T 是序列長度，$\mathbf{y}$ 是標記序列，y_t 是詞 x_t 的標記，$\mathcal{Y}$ 是有限標記集合。t 與 T 指語言學上的詞位置，不會自動等同模型子詞位置。

標註也需要縮寫、標點、多詞表達與多種合理分析的規則。當模型使用子詞時，應定義如何把輸出對應回詞。只用第一個子詞表示是一種做法；池化所有子詞則是另一種。不論選哪個，標準標籤與解碼輸出都必須使用相同單位。

局部分類與序列預測

白話版。分別替每個詞選最佳標記，可能錯過更好的共同解讀。序列模型把標記間的關係寫清楚。

局部分類器可以預測：

$$\hat{y}_t = \operatorname*{argmax}_{k \in \mathcal{Y}} p_\theta(y_t = k \mid \mathbf{x}).$$

其中 $\hat{y}_t$ 是位置 t 選出的標記，k 遍歷標記集合 $\mathcal{Y}$，p_θ 是參數為 θ 的模型，$\mathbf{x}$ 是整個可見輸入序列。局部輸出頭仍可以讀取上下文；「獨立」描述輸出決策規則，不一定描述編碼器。

結構化解碼器則選擇：

$$\hat{\mathbf{y}} = \operatorname*{argmax}_{\mathbf{y} \in \mathcal{A}(\mathbf{x})} S_\theta(\mathbf{x}, \mathbf{y}).$$

其中 $\hat{\mathbf{y}}$ 是選出的完整標記序列，$\mathcal{A}(\mathbf{x})$ 是輸入 $\mathbf{x}$ 允許的分析集合，S_θ 是由參數 θ 對輸入與分析配對給出的分數。允許集合可以限制標註規則，分數則可以獎勵合理的相鄰標籤。

許多 NLP 任務都有這個差別。實體標籤必須組成有效跨度；依存樹中每個非根詞需要一個父節點；生成的結構化紀錄需要符合結構規格。較強的編碼器不會自動保證有效輸出，解碼器或獨立驗證器必須落實應用需要的限制。

結構化解碼也有取捨：錯誤的硬限制可能排除合法構造。如果文法要求每句都有明示主語，就會拒絕許多命令與片段。應記錄限制與預期輸入範圍，而不是把受限制的輸出當成該限制普遍適用所有語言的證據。

用隱藏馬可夫模型標記

白話版。想像標記形成一串隱藏序列，每個標記再產生一個詞。相鄰標記與詞的相容性共同決定分數。

Rabiner 綜述的一階隱藏馬可夫模型（hidden Markov model，HMM）[3] 使用：

$$p(\mathbf{x}, \mathbf{y}) = \pi_{y_1} b_{y_1}(x_1) \prod_{t=2}^{T} a_{y_{t-1}, y_t} b_{y_t}(x_t).$$

其中 $p(\mathbf{x}, \mathbf{y})$ 是詞與標記的聯合機率，π_k 是起始標記為 k 的機率，$a_{j,k}$ 是從標記 j 轉移到 k 的機率，$b_k(w)$ 是標記 k 發射詞 w 的機率，x_t 與 y_t 是位置 t 的詞與標記，T 是序列長度。本例以固定長度為條件；若要建模終止，可加入明確的結束狀態因子。

起始機率總和為一；每列轉移機率總和為一；每個發射分布在觀察詞彙表上的總和也為一。這些正規化條件讓式子成為機率模型，而不是任意有用分數的乘積。

假設相當有限：下一個標記只依前一個標記，發射詞只依目前標記。真實句子有長距離依存、詞彙偏好與篇章脈絡。本章使用這個模型的價值，在於假設與精確解碼器都看得見；它是一個用來區分模型容量與搜尋的小型實驗室。

如果有完整標記的訓練樣本，可以計算起始狀態、轉移與發射次數，再依明確平滑規則正規化。如果只有詞，估計隱藏狀態就是另一個問題；沒有監督學出的狀態，不會自動成為有名稱的語法類別。下例手動指定機率，避免訓練不確定性遮蔽解碼機制。

最大路徑解碼

白話版。保留以每種標記結束的最佳部分路徑。延伸這些路徑，就能找到最佳完整序列，不必列出所有可能。

使用對數機率，把乘積轉成總和：

$$\begin{aligned} D_1(k) &= \log \pi_k + \log b_k(x_1), \\ D_t(k) &= \log b_k(x_t) + \max_{j \in \mathcal{Y}} [D_{t-1}(j) + \log a_{j,k}]. \end{aligned}$$

其中 $D_t(k)$ 是涵蓋位置 1 到 t、且以標記 k 結束的最佳路徑對數分數，π_k 是起始機率，$b_k(x_t)$ 是發射機率，$a_{j,k}$ 是轉移機率，$\mathcal{Y}$ 是標記集合。j 與 k 分別指前一個與目前標記。零機率貢獻 $-\infty$。

這是最大路徑動態規劃，通常稱為 Viterbi 解碼 [3]。每格記錄達到最大值的前一個標記。選出最佳最後標記後，反向追蹤回溯指標。如果只保留分數，就會失去實際序列。

解碼器考慮 T 個位置與所有相鄰標記配對，因此 $K = |\mathcal{Y}|$ 個標記時，時間複雜度為 $O(TK^2)$。保留所有回溯指標的分數表占 $O(TK)$ 空間；列舉所有標記序列則需考慮 K^T 種可能。這是在特定分解下的計算節省，不是近似搜尋結果。

下列獨立範例以完整列舉確認動態規劃答案。教學觀察「fish fish」刻意使用歧義詞。獨立的發射偏好會在兩個位置選名詞，聯合模型則可能偏好名詞後接動詞。

```
import itertools
import numpy as np

tags = ["N", "V"]
vocabulary = ["fish", "swim"]
initial = np.array([0.8, 0.2])
transition = np.array([[0.1, 0.9], [0.8, 0.2]])
emission = np.array([[0.7, 0.3], [0.3, 0.7]])
words = ["fish", "fish"]
observation = [vocabulary.index(word) for word in words]
assert np.allclose(transition.sum(axis=1), 1)
assert np.allclose(emission.sum(axis=1), 1)

scores = np.empty((len(words), len(tags)))
back = np.zeros_like(scores, dtype=int)
scores[0] = np.log(initial) + np.log(emission[:, observation[0]])
for t in range(1, len(words)):
    candidates = scores[t - 1, :, None] + np.log(transition)
    back[t] = candidates.argmax(axis=0)
    scores[t] = candidates.max(axis=0) + np.log(emission[:, observation[t]])
path = [int(scores[-1].argmax())]
for t in range(len(words) - 1, 0, -1):
    path.append(int(back[t, path[-1]]))
path.reverse()

def joint_score(sequence):
    value = np.log(initial[sequence[0]])
    value += np.log(emission[sequence[0], observation[0]])
    for t in range(1, len(words)):
        value += np.log(transition[sequence[t - 1], sequence[t]])
        value += np.log(emission[sequence[t], observation[t]])
    return value

all_paths = list(itertools.product(range(len(tags)), repeat=len(words)))
brute = max(all_paths, key=joint_score)
assert tuple(path) == brute
print("local emission choices:", [tags[emission[:, w].argmax()] for w in observation])
print("best joint path:", [tags[index] for index in path])
print("joint log score:", round(joint_score(path), 3))
```

改變轉移機率，即使每個詞的局部偏好不變，也能改變選出的序列。這並不證明新選擇符合真實語言；它證明模型與解碼器實作了預期的依存關係。

把路徑分數解讀為證據，而不只取最大值

前例選出名詞—動詞，是因為轉移偏好超過第二個詞的發射偏好。其聯合機率是四個因子的乘積：以名詞開始、名詞發射 `fish`、名詞轉移到動詞、動詞發射 `fish`。與名詞—名詞比較時，前兩個共同因子可以消去，剩下轉移乘上發射的比較。因此，不必把解碼器當成黑箱，就能理解決策。

以下獨立計算列舉相同的四條路徑，再正規化為觀察詞給定後的後驗分布：

```
import itertools
import numpy as np

initial = np.array([0.8, 0.2])
transition = np.array([[0.1, 0.9], [0.8, 0.2]])
fish = np.array([0.7, 0.3])
paths = list(itertools.product(range(2), repeat=2))
mass = np.array([initial[a] * fish[a] * transition[a, b] * fish[b]
                 for a, b in paths])
posterior = mass / mass.sum()
print('NN / NV / VN / VV joint:', mass.round(4))
print('Posterior:', posterior.round(3))
print('Probability of observed words:', round(mass.sum(), 4))
assert paths[posterior.argmax()] == (0, 1)
assert np.isclose(posterior.sum(), 1)
```

聯合機率為 `0.0392`、`0.1512`、`0.0336`、`0.0036`，後驗四捨五入為 `0.172`、`0.664`、`0.148`、`0.016`；觀察機率為 `0.2276`。它們回答不同問題：聯合機率包含這些詞本身多可能出現，後驗則在詞已固定後比較標記解讀。在這個人工模型中，最佳路徑之外仍有其他機率質量。

較長序列若把前驅貢獻改為相加，而不是取最大值，就能計算觀察機率；結合前向與後向貢獻，則得到標記邊際機率 [3]。逐位置選最高機率標記，會最小化該後驗下的逐位置期望錯誤；選最高機率完整路徑，則針對整段序列正確性。兩種決策可能不同。若允許的序列具有額外限制，獨立選出的邊際標記也可能違反輸出限制。

為什麼最大路徑解碼能丟掉其他以同一標記結束的前綴？在一階分解下，後續延伸只依賴最後標記與未來觀察，不依賴更早的前綴細節。加入相同後綴後，較低分前綴不會超越較高分前綴。若新增分數會獎勵與數個位置之前標記的關係，原狀態便不再滿足這個論證；必須擴大狀態或更換解碼器。精確性來自分數分解，不是演算法名稱。

條件式結構分數連結神經特徵與解碼

HMM 必須建模詞如何產生；條件模型則可直接依觀察句子評分標記序列。線性鏈條件隨機場（conditional random field，CRF）提供這種介面 [5]：

$$S_\theta(\mathbf{x},\mathbf{y}) = \sum_{t=1}^{T} u_\theta(t, y_t, \mathbf{x}) + \sum_{t=2}^{T} A_{y_{t-1},y_t}, \qquad p_\theta(\mathbf{y}\mid\mathbf{x}) = \frac{\exp S_\theta(\mathbf{x},\mathbf{y})}{Z_\theta(\mathbf{x})}.$$

其中 $\mathbf{x}$ 是觀察句子，$\mathbf{y}$ 是標記序列，T 是序列長度，t 索引位置，u_θ 是參數 θ 計算的位置—標記分數，A 包含相鄰標記分數。正規化常數 $Z_\theta(\mathbf{x})$ 對每個允許序列 $\mathbf{y}'$ 的 $\exp S_\theta(\mathbf{x},\mathbf{y}')$ 加總。位置分數與 HMM 發射不同，不必在所有詞上加總為一。上下文編碼器可以用整句計算各分數，同時讓輸出依存保持鏈狀。

訓練標註序列時，最小化其負對數機率：相對於所有允許競爭序列，提高正確序列分數。計算正規化常數使用加總遞迴，尋找最佳序列使用最大值遞迴。這說明神經編碼器與結構化解碼器如何配合。豐富的輸入證據不要求放棄可計算的輸出分解；反之，鏈狀轉移表無法表達全部語法依存。整句輸入特徵與輸出限制，處理的是不同部分。

PR 06 發展這兩種模型背後的一般理論：前向與前向—後向演算法、以 Baum–Welch 從未標註序列訓練 HMM，以及以「觀察特徵次數減去期望特徵次數」表示的 CRF 梯度。下文的神經標註器各節完全重用這個 CRF，只是位置分數 u_θ 改由循環編碼器計算。

實體辨識從結構描述開始

直觀說法。實體辨識器找出提及並指定類型，不會自動找出背後唯一的現實物件。

實體提及可表示為：

$$r = (d, b, e, c).$$

其中 r 是提及紀錄；d 是文件識別碼；b 是包含在內的起始字元位移；e 是不包含在內的結束位移；c 是類別。表面文字就是原文件從 b 到 e 的切片。本例使用 Python 字串位移，其他介面可能使用位元組或別的索引慣例。必須聲明慣例，因為 Unicode 字元與編碼後的位元組不具有相同邊界。

標註指南必須決定稱謂是否包含在人名區段中、組織名稱後綴是否保留，以及歧義用法如何處理。「臺北」可能指地點，也可能在上下文中代指政府。只標位置的結構與依上下文標組織的結構，可以合理地給出不同標註。只有凍結類型與指南，評估才有意義。

命名實體辨識（named entity recognition，NER）經常區分人物、組織與地點；CoNLL-2003 共享任務論文介紹了一種既有基準設定 [6]。提到它是為了說明評估傳統，並不表示其類別適用所有應用。別的領域可能更需要產品代碼、法律條文或設備零件。

實體連結（entity linking）是另外的決策：把提及對應到知識庫紀錄，或判定沒有適當紀錄。相同拼法可能指不同組織，不同拼法也可能指同一個組織。辨識器把 Orion 標成組織，還沒有解決身分問題。連結需要候選集合、上下文與明確的未知實體政策。

儲存正規化名稱時，也應保留原文與位移。小寫化、去除重音與合併空白可能改變字串及邊界。可靠設計可直接從原文擷取，把正規化結果存為額外欄位。若必須前處理，保留對齊映射，才能把預測區段轉回原文而不靠猜測。

序列標籤與區段預測

直觀說法。實體提及可表示成詞元標籤，也可直接表示成文字區段。表示方式決定模型能描述哪些重疊。

BIO 方案使用 B 表示提及起點、I 表示延續、O 表示外部。例如 Mina works at Orion Labs 中，Mina 帶人物標籤，Orion 與 Labs 帶組織標籤。Hagiwara 的序列標註章節與 NER 綜述提供詞元層級擷取的背景 [2]、[7]。這個方案源自 Ramshaw 與 Marcus 的內外語塊標籤 [10]。

獨立詞元分類器可寫成：

$$p_\theta(\mathbf{y} \mid \mathbf{x}) = \prod_{i=1}^{T} p_\theta(y_i \mid \mathbf{x}).$$

其中 $\mathbf{x}$ 是輸入詞元序列；$\mathbf{y}$ 是標籤序列；T 是詞元數；i 是位置索引；θ 是學得的參數。每次預測可以使用完整允許範圍的上下文表示，但在這個分解下，輸出標籤彼此獨立。輸入上下文與輸出相依性是不同概念。

獨立判定可能產生非法轉移，例如 O 後面接 I-PER。可以用確定性規則修正，但修正規則也會成為評估契約的一部分。另一種方式是以整個序列評分，讓結構化解碼器只搜尋合法路徑。若分數能局部分解，最大路徑解碼一節的動態規劃概念就能套用。更強的編碼器不會免除合法輸出定義。

單一平面詞元標籤序列無法表示任意巢狀提及。在某種結構描述下，Orion Labs Research Division 可能同時包含組織與巢狀部門提及。區段模型則直接評分候選邊界與類型：

$$s_\theta(b, e, c \mid \mathbf{x}).$$

其中 s_θ 是學得的分數；b、e 是候選邊界；c 是候選類別；$\mathbf{x}$ 是輸入。這個分數未必是校準後的機率。選擇政策決定保留哪些候選、是否允許重疊，以及衝突如何解決。

候選數會隨句長增加，實務上可能限制最長區段或刪除部分候選。每種限制都會改變可找回的提及。若長名稱很重要，就要單獨量測其召回率。否則，一個快速系統可能平均表現不錯，卻完全無法表示少數使用者不可或缺的輸出。

邊界建模改變可能出現的錯誤

詞元標記模型與區段模型可以使用相同上下文編碼器，卻搜尋不同輸出空間。平面 BIO 標籤替每個詞元選一個歸屬；明確區段則選邊界配對與類型，選取規則允許時，巢狀區間可同時存在。差別在表示能力：解碼器無法表達巢狀輸出時，編碼器也無法把它恢復出來 [7]。

對 n 個詞元，非空連續區間數為 $n(n+1)/2$。其中 n 是句長；這是把每個起點可能的終點數加總。最大長度限制為 $L \le n$ 時，數量變成 $nL - L(L-1)/2$，其中 L 是允許區段長度。類型評分器可能需要為每個區間、每個類別計算分數。限制長度節省工作，卻在分類開始之前，就把較長提及移出候選空間。

邊界品質與語意分類需要分別診斷。涵蓋 `Orion` 的區段可能是強烈的組織候選，但標準提及是 `Orion Labs`。提高類別信心，不會補上缺少的邊界。模型也可能同時提出兩個區段，再由去重與重疊規則決定最後輸出。選取屬於模型行為，不能當成評估完成後的排版步驟。

關係又增加一個搜尋問題。保留越多提及，實體配對候選越多。依類型或距離限制配對，能降低計算，但文件層級關係可能跨越數句。聯合擷取器可用關係證據影響提及決策；流程式模型則先選定提及。分別以預測提及與標準提及評估流程，兩者差距能揭露在關係評分開始之前，就損失多少關係涵蓋能力。

精確區段評估

直觀說法。常見背景詞標得正確，可能掩蓋漏掉的實體。應評估完整提及，包括邊界與類型。

對預測紀錄集合 P 與正確標註集合 G：

$$\text{precision} = \frac{|P \cap G|}{|P|}, \qquad \text{recall} = \frac{|P \cap G|}{|G|}, \qquad F_1 = \frac{2|P \cap G|}{|P| + |G|}.$$

其中 P、G 都包含文件識別碼、起始位移、結束位移與類別元組；$|\cdot|$ 是集合大小；$P \cap G$ 是完全匹配的紀錄。精確率衡量預測中匹配的比例；召回率衡量標註中找回的比例；F_1 結合兩者。本例程式在所示分母為零時回報零；若採其他慣例，應明確記錄。

完全匹配很嚴格。正確標註是 Orion Labs，卻只預測 Orion，就同時產生假陽性與假陰性。下游程式依賴完整名稱時，這種標準合理。其他應用可以把部分重疊當成次要診斷，但必須另外說明匹配規則，不能把寬鬆的重疊分數當成既有的完全匹配指標 [6]。

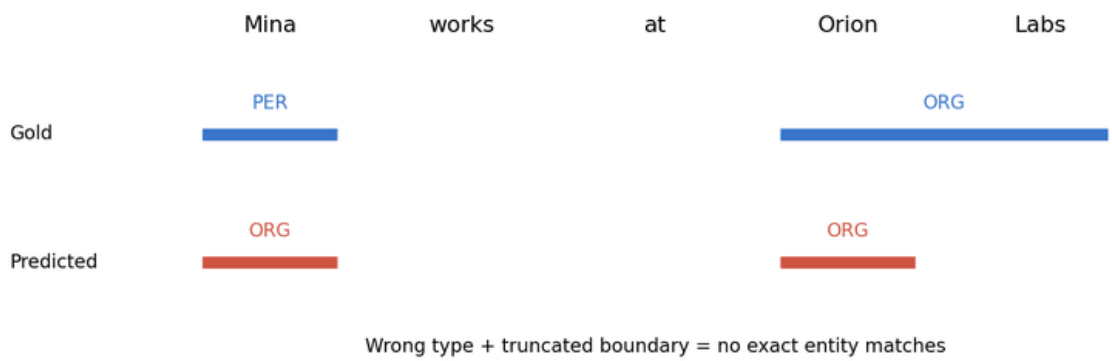


圖 *1* — 即使大多數詞語標得正確，邊界與類型錯誤仍然是錯誤。這是合成的診斷示例。

單一總分仍會隱藏原因。可區分漏掉提及、多餘提及、邊界不符、類別不符與連結不符，並按類別和文件來源報告表現。類別稀少時保留例數，讓讀者判斷不確定性。大型加權平均可能掩蓋幾乎沒有成功預測的類別。

完整的 CPU 擷取實驗

直觀說法。小型確定性擷取器能讓契約容易檢查。它是盲點可見的教學基準，不是通用語言理解系統。

以下獨立範例辨識自訂名稱表、擷取嚴格受限的主動修理句型，並評估精確實體紀錄。它不下載任何資料。名稱表在評估前固定，正確標註也明確寫出。本例允許使用詞彙表；若真實評估禁止外部名稱集合，同樣程式就會違反契約。

```
import re

documents = {
    "a": "Mina repaired Sensor A in Taipei.",
    "b": "Orion Labs said Sensor B might fail.",
    "c": "Leo did not repair Sensor B.",
    "d": "The sensor was repaired by Mina.",
}
inventory = {
    "Mina": "PER", "Leo": "PER", "Orion Labs": "ORG",
    "Sensor A": "DEVICE", "Sensor B": "DEVICE", "Taipei": "LOC",
}
gold = {
    ("a", 0, 4, "PER"), ("a", 14, 22, "DEVICE"), ("a", 26, 32, "LOC"),
    ("b", 0, 10, "ORG"), ("b", 16, 24, "DEVICE"),
    ("c", 0, 3, "PER"), ("c", 19, 27, "DEVICE"), ("d", 27, 31, "PER"),
}
predicted = set()
for doc_id, text in documents.items():
    for name, category in inventory.items():
        pattern = r"(?<!\w)" + re.escape(name) + r"(?!\w)"
        for match in re.finditer(pattern, text):
            predicted.add((doc_id, match.start(), match.end(), category))
tp = len(predicted & gold)
precision = tp / len(predicted) if predicted else 0.0
recall = tp / len(gold) if gold else 0.0
f1 = 2 * tp / (len(predicted) + len(gold)) if predicted or gold else 0.0
assert predicted == gold
print("entity precision/recall/F1:", precision, recall, f1)

repair = re.compile(r"^(Mina|Leo) repaired (Sensor [AB])(?=[ .])")
events = []
for doc_id, text in documents.items():
    match = repair.search(text)
    if match:
        events.append({
            "document": doc_id, "predicate": "repair",
            "actor": match.group(1), "object": match.group(2),
            "polarity": "positive", "modality": "asserted",
            "evidence": [match.start(), match.end()],
        })
assert len(events) == 1 and events[0]["document"] == "a"
for event in events:
    start, end = event["evidence"]
    print(event, "source:", documents[event["document"]][start:end])
print("unsupported repair constructions:", ["c", "d"])
```

在這些人工設計文件中，精確實體分數是 1.0，因為所有名稱拼法都在表中。這不能證明泛化能力。事件規則只回傳文件 a；它排除了 c 的明確否定，卻也漏掉 d 的合法被動句。兩者都是沒有

事件輸出，但原因不同。正式介面應區分被拒絕的否定主張與不支援的句型。

文件 b 有已辨識實體，卻沒有被直接斷言的修理事件；它還包含帶消息來源、可能發生的故障。這說明為何實體與事件需要分開評估。系統可能抓對所有名字，卻遺漏任務最重要的事件。擴充事件語法前，要先加入額外的正例與反例測試。

證據區間只包含匹配子句，不包含後面的地點。對沒有地點欄位的狹義修理紀錄，這樣足夠。若之後加入地點，就要一併擴充證據契約與測試。結構變更應視為任務變更，不能只是新增一個欄位，再隨意從附近不相關詞語填值。

實體區段之外的資訊擷取

白話版。實體辨識找出名稱；資訊擷取（information extraction，IE）接著把它們連起來：哪個人為哪個組織工作、哪台裝置在何時何地故障。每一步都加入一種新紀錄，也多一種出錯的方式。

一張有用的地圖分成三層，每層建立在前一層之上。

層次	輸出紀錄	本章領域的例子	常見方法類型
實體	有類型的區段 (d, b, e, c)	`Orion Labs` 是位於字元 0–10 的 ORG	序列標註（本章）或區段評分
關係	有類型、有方向的提及配對，附證據	`employer(Mina, Orion Labs)`	分類候選提及配對
事件	觸發詞加上帶角色標籤的論元、肯否定與模態	`repair`，修理者 `Mina`、物件 `Sensor A`、地點 `Taipei`	先偵測觸發詞，再標註論元

序列標註直接貢獻於三個層次。實體提及是 BIO 區段；事件觸發詞可以用與實體相同的方式標註；甚至部分關係與論元判定，也能相對於給定觸發詞標註每個論元的詞元，寫成標註問題。上面的規則式擷取器已經顯示層次之間的依賴：它唯一的事件紀錄之所以正確，是因為兩個實體區段都被找到，而且句型受到支援。

標準資源反映了這些層次。ACE 2005 多語訓練語料替英文、中文與阿拉伯文標註實體、關係與事件，其中事件標註只限英文與中文 [19]。TACRED 是以群眾外包建立的句子層級關係擷取資料集，針對 TAC 知識庫填充（Knowledge Base Population）評測的關係類型 [20]。DocRED 把關係擷取擴展到由 Wikipedia 與 Wikidata 建立的完整文件，其中許多關係必須讀過多個句子才能推得 [21]。本章前面提出的紀錄設計問題（證據區段、消息歸屬、否定、未知值）適用於每一層。關係與事件擷取、語意角色及其評估，在 NLP 12 發展。

神經標註器：讀取整句的 BiLSTM

白話版。以一個雙向讀取整句、在每個位置替每個標籤產生分數的神經網路，取代手寫的發射機率。最簡單的版本在每個位置獨立選出最高分標籤。

Collobert 等人證明，單一神經架構幾乎不需特徵工程，就能從資料學會標註詞性、語塊、命名實體與語意角色 [12]。使用循環編碼器時，標準設計是雙向 LSTM（BiLSTM）標註器 [13]、[14]、[15]：

$$\overrightarrow{\mathbf{h}}_t = \mathrm{LSTM}_{\rightarrow}(\mathbf{v}_t, \overrightarrow{\mathbf{h}}_{t-1}), \qquad \overleftarrow{\mathbf{h}}_t = \mathrm{LSTM}_{\leftarrow}(\mathbf{v}_t, \overleftarrow{\mathbf{h}}_{t+1}),$$
$$\mathbf{u}_t = W\,[\overrightarrow{\mathbf{h}}_t; \overleftarrow{\mathbf{h}}_t] + \mathbf{b} \in \mathbb{R}^K.$$

其中 $\mathbf{v}_t$ 是詞元 x_t 的輸入向量（詞嵌入，可再串接下文的字元特徵）；$\overrightarrow{\mathbf{h}}_t$ 讀過 $x_1, \dots, x_t$，$\overleftarrow{\mathbf{h}}_t$ 讀過 $x_t, \dots, x_T$；$[\cdot;\cdot]$ 表示串接；W 與 $\mathbf{b}$ 把兩個狀態對應到 $K = |\mathcal{Y}|$ 個標籤分數；$\mathbf{u}_t$ 收集這些分數，標籤 k 的分數為 $u_{t,k}$。這些分數扮演前面 CRF 公式中 $u_\theta(t, y_t, \mathbf{x})$ 的角色：它們可以依賴整個句子。與 NLP 05 由左到右的語言模型不同，這裡適合使用雙向結構，因為選擇任何標籤之前整個句子都已給定 [23]。

softmax 標註器在給定這些分數後，把各標籤視為條件獨立：

$$p_\theta(y_t = k \mid \mathbf{x}) = \frac{\exp u_{t,k}}{\sum_{k'=1}^{K} \exp u_{t,k'}}, \qquad \mathcal{L}_{\text{tok}} = -\frac{1}{N} \sum_{b} \sum_{t} m_{b,t} \log p_\theta(y_{b,t} \mid \mathbf{x}_b).$$

其中 b 遍歷小批次中的句子，$m_{b,t} \in \{0, 1\}$ 是填補遮罩（真實詞元為一），$N = \sum_{b,t} m_{b,t}$ 計算真實詞元數，$y_{b,t}$ 是正確標籤，k' 遍歷標籤。這正是序列標籤一節中的獨立詞元分類器，只是改由 BiLSTM 計算上下文。它速度快，也常常很強；它的弱點也與那裡指出的相同：沒有任何機制阻止它輸出 `O I-ORG` 或 `B-PER I-ORG`，因為每個位置的判定都忽略了鄰近位置的判定。

三項實務細節與架構同樣重要。

- **遮罩**。批次中的句子會填補到相同長度。遮罩把填補排除在損失之外；打包（`pack_padded_sequence`）或遮罩則讓填補不進入反向 LSTM，否則它會從填補符號開始讀。
- **未知詞**。詞嵌入表只涵蓋訓練詞。其餘詞對應到未知編號，而且要訓練這個編號：隨機把一部分訓練詞換成它（詞 dropout），可以教標註器在詞本身陌生時依賴上下文。
- **子詞編碼器**。若編碼器使用子詞，要定義詞標籤如何對應到片段（只標第一個片段並忽略其餘，或另一條明確規則），並在評估前把預測對應回詞，如第一節所述。

BiLSTM-CRF：替整個標籤序列評分

白話版。保留 BiLSTM 的逐詞元分數，再替每一對相鄰標籤加上學習得到的分數，選出總分最高的標籤序列。訓練時提高正確序列相對於所有其他序列的分數，而前向演算法能有效率地完成這個加總。

Huang、Xu 與 Yu 提出在 BiLSTM 上加一層線性鏈 CRF，用於詞性標註、語塊切分與 NER [13]。Lample 等人 [14] 以及 Ma 與 Hovy [15] 把同樣的輸出層與字元層級詞表示結合，在不使用手工特徵的情況下得到強大的 NER 結果。沿用最大路徑解碼一節的 CRF 記號，並加上起始與結束分數：

$$S_\theta(\mathbf{x}, \mathbf{y}) = \pi_{y_1} + \sum_{t=1}^{T} u_{t,y_t} + \sum_{t=2}^{T} A_{y_{t-1}, y_t} + \eta_{y_T}, \qquad \log p_\theta(\mathbf{y} \mid \mathbf{x}) = S_\theta(\mathbf{x}, \mathbf{y}) - \log Z_\theta(\mathbf{x}).$$

其中 $u_{t,k}$ 是 BiLSTM 分數；$A_{j,k}$ 是標籤 j 後接標籤 k 的學習分數；π_k 與 η_k 是以標籤 k 起始與結束的分數；$Z_\theta(\mathbf{x}) = \sum_{\mathbf{y}'} \exp S_\theta(\mathbf{x}, \mathbf{y}')$ 對所有長度為 T 的標籤序列 $\mathbf{y}'$ 加總。訓練損失是對句子平均的負對數概似 $-\log p_\theta(\mathbf{y} \mid \mathbf{x})$。

對 K^T 個序列的加總，以對數空間的前向演算法計算：

$$\alpha_1(k) = \pi_k + u_{1,k},$$
$$\alpha_t(k) = u_{t,k} + \log \sum_{j=1}^{K} \exp\left(\alpha_{t-1}(j) + A_{j,k}\right), \qquad t = 2, \dots, T,$$
$$\log Z_\theta(\mathbf{x}) = \log \sum_{k=1}^{K} \exp\left(\alpha_T(k) + \eta_k\right).$$

其中 $\alpha_t(k)$ 是涵蓋位置 1 到 t、以標籤 k 結尾的所有部分序列，其指數分數總和的對數。把每個 log-sum-exp 換成最大值並記錄取得最大值的 j，就得到 HMM 一節的 Viterbi 遞迴，只是 $\log b_k(x_t)$ 換成 $u_{t,k}$、$\log a_{j,k}$ 換成 $A_{j,k}$。兩者的成本都是 $O(TK^2)$。在填補過的批次中，遮罩為零的位置直接把 α_{t-1} 原樣往前複製，因此每個句子的正規化常數都停在它自己的長度。

梯度在做什麼。損失對轉移分數微分得到

$$\frac{\partial(-\log p_\theta(\mathbf{y} \mid \mathbf{x}))}{\partial A_{j,k}} = \mathbb{E}_{\mathbf{y}' \sim p_\theta(\cdot \mid \mathbf{x})}[n_{j,k}(\mathbf{y}')] - n_{j,k}(\mathbf{y}),$$

其中 $n_{j,k}(\mathbf{y})$ 計算 $\mathbf{y}$ 中標籤 j 後接標籤 k 的次數，期望值依模型分布計算。因此梯度下降會提高正確序列使用得比模型目前預期更多的轉移，並降低其他轉移；期望值來自前向—後向邊際機率，如 PR 06 所示。對前向演算法自動微分就能算出這個量，不必手寫反向傳播。

硬性限制。 BIO 方案直接禁止某些轉移：`I-x` 只能接在 `B-x` 或 `I-x` 之後，而且句子不能以 `I-x` 開頭。學習得到的 A 通常能從資料發現這一點，但資料很少時未必如此。把 A 與 π 中被禁止的項設為 $-\infty$（實務上用很大的負數），就讓非法序列在訓練與解碼時都不可能出現，而對合法序列毫無影響。這是局部與序列預測一節中輸出限制概念的具體實作。

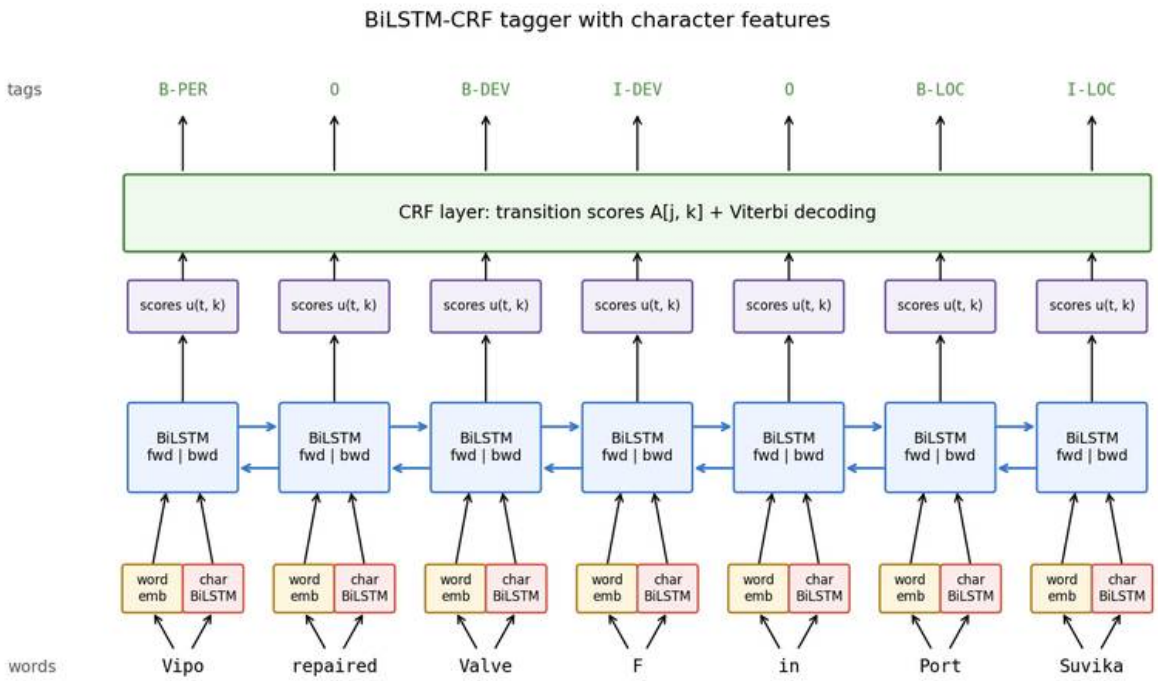


圖 2 — 帶字元特徵的 *BiLSTM-CRF* 標註器。發射分數來自編碼器；轉移分數與 *Viterbi* 解碼讓各標籤彼此一致。

僅線上版：*BiLSTM-CRF — Bidirectional LSTM-CRF Models for Sequence Tagging* — https://blog.shyandram.dev/zh-tw/learn/nlp/sequence-labeling/papers/bilstm-crf–bidirectional-lstm-crf-models-for-sequence-tagging/

字元層級特徵

白話版。詞嵌入表對沒見過的詞一無所知。詞的字母卻常常知道：大寫、「-ville」或「-corp」這類字尾、數字與連字號，即使對全新的名稱也是很強的標籤線索。

在神經標註器之前，這些線索是手寫特徵（是否大寫、是否以「-ing」結尾、是否含數字）。神經標註器則從字元學習它們。Lample 等人在每個詞的字元上執行一個小型雙向 LSTM，並把最後的前向與反向狀態串接到詞嵌入 [14]；Ma 與 Hovy 則對字元嵌入做卷積，再做最大池化 [15]。無論哪一種，輸入都變成

$$\mathbf{v}_t = [\mathbf{e}(x_t);\, \mathbf{c}(x_t)], \qquad \mathbf{c}(x_t) = [\vec{\mathbf{g}}_L; \overleftarrow{\mathbf{g}}_1],$$

其中 $\mathbf{e}(x_t)$ 是詞嵌入，$\mathbf{c}(x_t)$ 是字元摘要，L 是 x_t 的字元數，$\vec{\mathbf{g}}_L$ 是讀完最後一個字元後的前向字元 LSTM 狀態，$\overleftarrow{\mathbf{g}}_1$ 是反向讀回第一個字元後的狀態。詞層級 BiLSTM 接著讀取 $\mathbf{v}_t$。Akbik、Blythe 與 Vollgraf 把這個想法再往前推：他們像 NLP 05 一樣訓練字元層級語言模型，再把每個詞周圍的內部狀態當作序列標註用的上下文嵌入 [16]。

字元特徵正好幫助詞嵌入最難處理的輸入：未見名稱、罕見詞形變化與拼寫錯誤。但它們無法解決拼字本身沒有編碼的歧義。沒有明顯字尾的名稱可能是人，也可能是公司，只有上下文或世界知識能決定。字元輸入也需要自己的未知符號規則，處理訓練中沒出現過的字元。

標註與擷取的資料集

白話版。基準會固定標籤集合、標註指南、分詞方式與資料切分。只有四者都一致時，跨基準比較數字才有意義。

資源	任務	提供什麼
Penn Treebank [11]	英文詞性標註與剖析	大型英文標註語料，含詞性標籤與句法括號結構；其標籤集廣泛用於英文
Universal Dependencies [4]	跨語言詞性、詞法與依存關係	套用到多種語言樹庫的共同標註框架
CoNLL-2003 [6]	命名實體辨識	英文與德文資料，含四種類型（人物、組織、地點、其他）與完全匹配評估
OntoNotes [17]	18 種類型的命名實體，以及句法、語意角色與共指	多文體、多語言標註，具分層抽樣的訓練、開發與測試切分，其測試集與 CoNLL-2012 共享任務一致
WNUT 2017 [18]	雜訊使用者生成文字中的新興實體	測試罕見與先前未見表面形式的召回率
ACE 2005 [19]	實體、關係與事件	多語言標註（英文、中文、阿拉伯文），用於關係與事件擷取

資源	任務	提供什麼
TACRED [20]	句子層級關係擷取	針對 TAC KBP 關係類型、以群眾外包建立的關係標籤
DocRED [21]	文件層級關係擷取	在 Wikipedia 文件上標註的實體與關係，需要跨句推理

標籤編碼也各不相同。CoNLL-2003 檔案使用源自 Ramshaw 與 Marcus 語塊標籤的 IOB 變體，其中 `B-` 只出現在某個實體緊接在另一個同類型實體之後 [6]、[10]；訓練前應把這類檔案轉成本章使用的 BIO 形式，否則 BIO 區段擷取器會誤讀。授權也不同：有些資源可以自由下載，有些則由語言資料聯盟（Linguistic Data Consortium）依授權發行。使用前先確認條款。對新領域而言，最有價值的資料集通常是依你自己凍結的指南、仔細裁定過的小型標註集，再搭配公開基準來比較方法。

完整的 PyTorch 標註實驗

白話版。我們在同一個小型合成語料上訓練三個標註器，並在名稱從未出現在訓練中的句子上評估：只用詞的 BiLSTM、加上字元特徵的同一標註器，以及在字元標註器上加上受限 CRF 輸出層。評估計算精確區段、錯誤類型與非法標籤轉移。

語料是虛構的。人名、組織名與地名由隨機音節產生，因此測試名稱都是新拼法。組織名要嘛以 `tek` 或 `corp` 結尾，要嘛後面接 `Labs` 這個詞；地名要嘛以 `ville` 結尾，要嘛接在 `Port` 之後。有幾個模板允許同一位置放人或組織，所以單靠上下文不一定能決定類型。訓練句有 100 句，測試句有 200 句。其他所有東西都從零實作：從 BIO 標籤擷取區段、精確區段的精確率、召回率與 F1、錯誤分類、填補遮罩、詞 dropout、字元 BiLSTM，以及具遮罩前向演算法、Viterbi 解碼與 BIO 限制的 CRF，並以暴力列舉驗證。每個模型以三個種子訓練；整個程式在 CPU 上遠少於一分鐘即可完成。

```
import random
import torch
from torch import nn

# ---- 1. A synthetic tagged corpus: invented names, fixed templates ----
SYL = ["ka", "mi", "to", "re", "na", "lo", "su", "vi", "da", "po", "ri", "ze"]

def coin(rng, kind):
    stem = "".join(rng.choice(SYL) for _ in range(rng.choice((2, 3)))).capitalize()
    if kind == "PER":
        return stem
    if kind == "ORG":
        return rng.choice([stem + rng.choice(["tek", "corp"]), stem + " Labs"])
    return rng.choice(["Port " + stem, stem + "ville"])          # LOC

rng = random.Random(0)
TRAIN_NAMES = {k: sorted({coin(rng, k) for _ in range(40)}) for k in ("PER", "ORG", "LOC")}
seen = {n for v in TRAIN_NAMES.values() for n in v}
TEST_NAMES = {k: rng.sample(sorted({coin(rng, k) for _ in range(60)} - seen), 15)
              for k in ("PER", "ORG", "LOC")}                     # unseen spellings
```

```
TRAIN_NAMES["DEV"] = ["Pump A", "Pump B", "Sensor C", "Sensor D", "Valve E", "Valve F"]
TEST_NAMES["DEV"] = ["Pump G", "Sensor H", "Valve J", "Pump K"]
TEMPLATES = [
    "{PER} repaired {DEV} in {LOC} .",
    "{ORG} repaired {DEV} in {LOC} .",
    "{PER} reported a fault in {DEV} .",
    "{ORG} reported a fault in {DEV} .",
    "the sensor near {LOC} was checked by {PER} .",
    "the sensor near {LOC} was checked by {ORG} .",
    "{PER} and {PER} visited {LOC} .",
    "{ORG} shipped {DEV} to {LOC} yesterday .",
    "after the test , {PER} said {DEV} might fail .",
    "a pump in {LOC} failed and {ORG} replaced it .",
]

def make(names, n):
    data = []
    for _ in range(n):
        words, tags = [], []
        for piece in rng.choice(TEMPLATES).split():
            if piece.startswith("{"):
                kind = piece[1:-1]
                toks = rng.choice(names[kind]).split()
                words += toks
                tags += ["B-" + kind] + ["I-" + kind] * (len(toks) - 1)
            else:
                words.append(piece)
                tags.append("O")
        data.append((words, tags))
    return data

train_data, test_data = make(TRAIN_NAMES, 100), make(TEST_NAMES, 200)
TAGS = ["O"] + [f"{p}-{k}" for k in ("PER", "ORG", "LOC", "DEV") for p in "BI"]
t2i = {t: i for i, t in enumerate(TAGS)}
w2i = {w: i + 2 for i, w in enumerate(sorted({w for s, _ in train_data for w in s}))}
c2i = {c: i + 2 for i, c in enumerate(sorted({c for w in w2i for c in w}))}
PAD, UNK = 0, 1
print(train_data[0])

# ---- 2. Spans and exact span F1, from scratch ----
def spans(tags):
    out, start, kind = set(), None, None
    for i, t in enumerate(tags + ["O"]):
        if start is not None and not (t.startswith("I-") and t[2:] == kind):
            out.add((start, i, kind))
            start = None
        if t.startswith("B-") or (t.startswith("I-") and start is None):
            start, kind = i, t[2:]                  # an orphan I- still opens a span
    return out

def span_prf(gold_seqs, pred_seqs):
    tp = n_pred = n_gold = 0
    for g, p in zip(gold_seqs, pred_seqs):
        G, P = spans(g), spans(p)
        tp, n_pred, n_gold = tp + len(G & P), n_pred + len(P), n_gold + len(G)
    prec = tp / n_pred if n_pred else 0.0
```

```
    rec = tp / n_gold if n_gold else 0.0
    f1 = 2 * tp / (n_pred + n_gold) if n_pred + n_gold else 0.0
    return prec, rec, f1

def error_types(gold_seqs, pred_seqs):
    counts = {"type": 0, "boundary": 0, "missed": 0, "spurious": 0}
    for g, p in zip(gold_seqs, pred_seqs):
        G, P = spans(g), spans(p)
        overlap = lambda a, b: a[0] < b[1] and b[0] < a[1]
        for s in G - P:
            hits = [q for q in P if overlap(s, q)]
            if any(q[:2] == s[:2] for q in hits):
                counts["type"] += 1             # right boundaries, wrong category
            elif hits:
                counts["boundary"] += 1         # overlaps, wrong boundaries
            else:
                counts["missed"] += 1
        counts["spurious"] += sum(not any(overlap(q, s) for s in G) for q in P - G)
    return counts

def invalid(tags):                                  # I-x not preceded by B-x or I-x
    return sum(t.startswith("I-") and (i == 0 or tags[i - 1][2:] != t[2:])
               for i, t in enumerate(tags))

assert spans(["B-ORG", "I-ORG", "O", "B-PER"]) == {(0, 2, "ORG"), (3, 4, "PER")}
assert invalid(["O", "I-LOC", "B-DEV", "I-DEV"]) == 1
print("train/test sentences:", len(train_data), len(test_data),
      " gold test spans:", sum(len(spans(t)) for _, t in test_data))

# ---- 3. Padding, masks and word dropout ----
def batch(data, unk_rate=0.0, r=None):
    L = max(len(s) for s, _ in data)
    W = max(len(w) for s, _ in data for w in s)
    x = torch.full((len(data), L), PAD)
    c = torch.full((len(data), L, W), PAD)
    y = torch.zeros(len(data), L, dtype=torch.long)
    for b, (s, t) in enumerate(data):
        for i, w in enumerate(s):
            drop = r is not None and r.random() < unk_rate
            x[b, i] = UNK if drop else w2i.get(w, UNK)      # train the unknown id too
            y[b, i] = t2i[t[i]]
            for j, ch in enumerate(w):
                c[b, i, j] = c2i.get(ch, UNK)
    return x, c, y, x != PAD

# ---- 4. BiLSTM encoder with optional character BiLSTM features ----
class Encoder(nn.Module):
    def __init__(self, use_chars, wdim=32, cdim=16, hidden=48):
        super().__init__()
        self.use_chars = use_chars
        self.wemb = nn.Embedding(len(w2i) + 2, wdim, padding_idx=PAD)
        self.cemb = nn.Embedding(len(c2i) + 2, cdim, padding_idx=PAD)
        self.clstm = nn.LSTM(cdim, cdim, batch_first=True, bidirectional=True)
        self.lstm = nn.LSTM(wdim + (2 * cdim if use_chars else 0), hidden,
                            batch_first=True, bidirectional=True)
        self.emit = nn.Linear(2 * hidden, len(TAGS))
```

```
    def forward(self, x, c):
        feats = [self.wemb(x)]
        if self.use_chars:
            B, L, W = c.shape
            lengths = (c != PAD).sum(-1).clamp(min=1).view(-1)
            packed = nn.utils.rnn.pack_padded_sequence(
                self.cemb(c.view(B * L, W)), lengths, batch_first=True,
                enforce_sorted=False)
            _, (h, _) = self.clstm(packed)          # last forward + first backward state
            feats.append(torch.cat([h[0], h[1]], -1).view(B, L, -1))
        h, _ = self.lstm(torch.cat(feats, -1))
        return self.emit(h)                         # emission scores u[b, t, k]

# ---- 5. Linear-chain CRF with BIO constraints and a padding mask ----
class CRF(nn.Module):
    def __init__(self, K):
        super().__init__()
        self.A = nn.Parameter(torch.zeros(K, K))    # A[j, k]: score of tag j -> tag k
        self.start = nn.Parameter(torch.zeros(K))
        self.end = nn.Parameter(torch.zeros(K))
        ban = torch.zeros(K, K)                     # I-x only after B-x or I-x
        for k, tag in enumerate(TAGS):
            if tag.startswith("I-"):
                ban[:, k] = -1e4
                ban[t2i["B-" + tag[2:]], k] = ban[k, k] = 0.0
        self.register_buffer("ban", ban)
        self.register_buffer("ban_start", torch.tensor(
            [-1e4 if t.startswith("I-") else 0.0 for t in TAGS]))

    def trans(self):
        return self.A + self.ban

    def score(self, u, y, mask):                    # S(x, y) for the gold path
        s = (self.start + self.ban_start)[y[:, 0]] + u[:, 0].gather(1, y[:, :1]).squeeze(1)
        for t in range(1, u.size(1)):
            step = self.trans()[y[:, t - 1], y[:, t]] + u[:, t].gather(1, y[:, t:t +
             ↪  1]).squeeze(1)
            s = s + step * mask[:, t]
        last = y.gather(1, (mask.sum(1) - 1).unsqueeze(1)).squeeze(1)
        return s + self.end[last]

    def log_partition(self, u, mask):               # forward algorithm in log space
        alpha = self.start + self.ban_start + u[:, 0]
        for t in range(1, u.size(1)):
            nxt = torch.logsumexp(alpha.unsqueeze(2) + self.trans() + u[:, t].unsqueeze(1),
             ↪  dim=1)
            alpha = torch.where(mask[:, t:t + 1], nxt, alpha)   # padding: carry forward
        return torch.logsumexp(alpha + self.end, dim=1)

    def nll(self, u, y, mask):
        return (self.log_partition(u, mask) - self.score(u, y, mask)).mean()

    def viterbi(self, u, mask):                     # same recursion with max + backpointers
        delta, back = self.start + self.ban_start + u[:, 0], []
        for t in range(1, u.size(1)):
```

```
            best, arg = (delta.unsqueeze(2) + self.trans()).max(1)
            back.append(arg)
            delta = torch.where(mask[:, t:t + 1], best + u[:, t], delta)
        delta = delta + self.end
        paths = []
        for b in range(u.size(0)):
            k = int(delta[b].argmax())
            path = [k]
            for t in range(int(mask[b].sum()) - 1, 0, -1):
                k = int(back[t - 1][b, k])
                path.append(k)
            paths.append(path[::-1])
        return paths

# Check the masked forward algorithm and Viterbi against brute-force enumeration
import itertools
torch.manual_seed(0)
check = CRF(len(TAGS))
with torch.no_grad():
    check.A.normal_()
u = torch.randn(1, 3, len(TAGS))
m = torch.ones(1, 3, dtype=torch.bool)
paths = [torch.tensor([p]) for p in itertools.product(range(len(TAGS)), repeat=3)]
scores = torch.stack([check.score(u, p, m)[0] for p in paths])
assert torch.allclose(check.log_partition(u, m)[0], torch.logsumexp(scores, 0), atol=1e-4)
assert paths[int(scores.argmax())][0].tolist() == check.viterbi(u, m)[0]

# ---- 6. Train three taggers and evaluate exact spans on unseen names ----
def run(use_chars, use_crf, seed, epochs=15):
    torch.manual_seed(seed)
    r = random.Random(seed)
    enc, crf = Encoder(use_chars), CRF(len(TAGS))
    params = list(enc.parameters()) + (list(crf.parameters()) if use_crf else [])
    opt = torch.optim.Adam(params, lr=5e-3)
    order = list(range(len(train_data)))
    for _ in range(epochs):
        r.shuffle(order)
        for i in range(0, len(order), 32):
            x, c, y, mask = batch([train_data[j] for j in order[i:i + 32]], 0.25, r)
            u = enc(x, c)
            loss = (crf.nll(u, y, mask) if use_crf      # sequence-level loss
                    else nn.functional.cross_entropy(u[mask], y[mask]))  # per token
            opt.zero_grad()
            loss.backward()
            nn.utils.clip_grad_norm_(params, 5.0)
            opt.step()
    enc.eval()
    with torch.no_grad():
        x, c, y, mask = batch(test_data)
        u = enc(x, c)
        pred = (crf.viterbi(u, mask) if use_crf else
                [u[b, :int(mask[b].sum())].argmax(-1).tolist() for b in range(len(u))])
    pred = [[TAGS[k] for k in p] for p in pred]
    gold = [t for _, t in test_data]
    return span_prf(gold, pred), sum(map(invalid, pred)), error_types(gold, pred), crf
```

```
results = {}
for name, use_chars, use_crf in [("BiLSTM, words", False, False),
                                 ("BiLSTM, words+chars", True, False),
                                 ("BiLSTM-CRF, words+chars", True, True)]:
    runs = [run(use_chars, use_crf, seed) for seed in (0, 1, 2)]
    results[name] = runs
    f1s = [prf[2] for prf, _, _, _ in runs]
    errors = {k: sum(r[2][k] for r in runs) for k in runs[0][2]}
    print(f"{name}: F1 per seed {[round(f, 3) for f in f1s]}, mean {sum(f1s) / 3:.3f}")
    print(f"    invalid I- tags {[r[1] for r in runs]}, errors over 3 seeds {errors}")
assert all(r[1] == 0 for r in results["BiLSTM-CRF, words+chars"])
```

輸出：

```
(['Vipo', 'repaired', 'Valve', 'F', 'in', 'Port', 'Suvika', '.'], ['B-PER', 'O', 'B-DEV',
↪  'I-DEV', 'O', 'B-LOC', 'I-LOC', 'O'])
train/test sentences: 100 200  gold test spans: 497
BiLSTM, words: F1 per seed [0.893, 0.771, 0.82], mean 0.828
    invalid I- tags [0, 30, 0], errors over 3 seeds {'type': 66, 'boundary': 43, 'missed':
    ↪  247, 'spurious': 0}
BiLSTM, words+chars: F1 per seed [0.893, 0.903, 0.942], mean 0.913
    invalid I- tags [4, 1, 0], errors over 3 seeds {'type': 124, 'boundary': 2, 'missed': 7,
    ↪  'spurious': 0}
BiLSTM-CRF, words+chars: F1 per seed [0.915, 0.909, 0.961], mean 0.929
    invalid I- tags [0, 0, 0], errors over 3 seeds {'type': 102, 'boundary': 0, 'missed': 8,
    ↪  'spurious': 0}
```

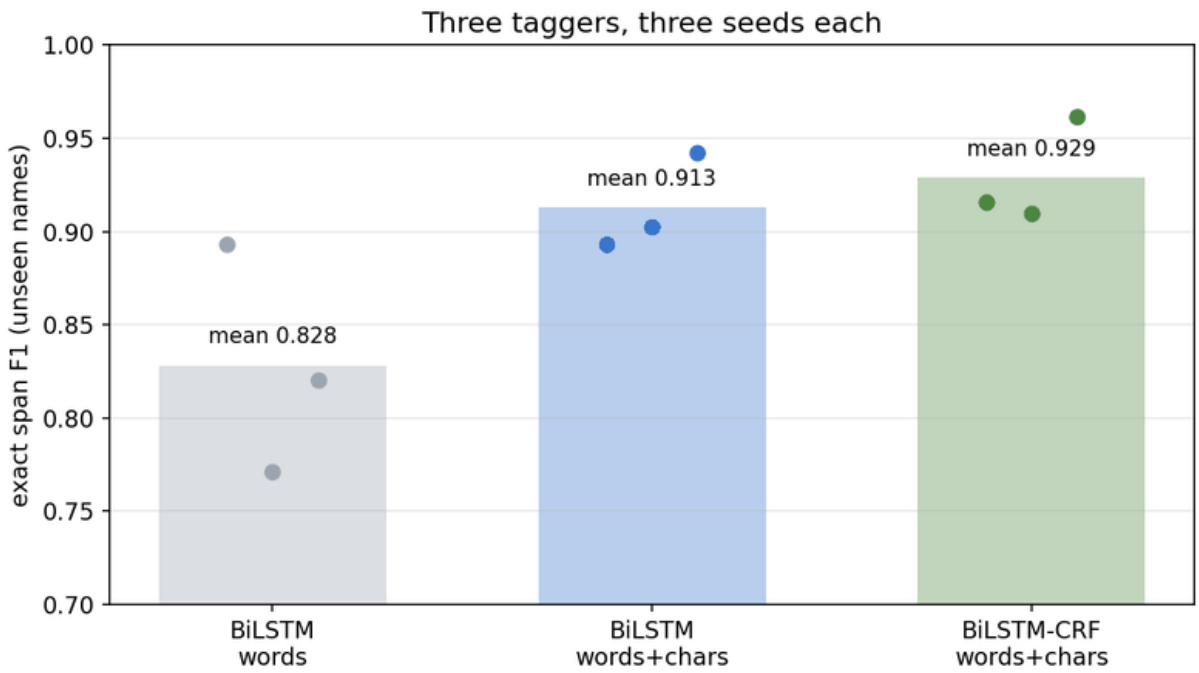


圖 3 — 在名稱從未出現在訓練中的測試句上的精確區段 *F1*。點是各種子，長條是平均。字元特徵帶來最大增益；受限 *CRF* 再加上較小但一致的改善。

解讀結果

字元特徵修好偵測。只用詞的標註器把大多數測試名稱看成未知編號。它的錯誤以漏掉實體為主（三個種子共 247 個），另外還有兩詞元名稱被切開或截短造成的邊界錯誤。加入字元特徵後，漏掉的實體降到 7 個，邊界錯誤降到 2 個：新名稱的字尾與大寫已足以找出它們。平均 F1 從 0.828 提高到 0.913。

剩下的是類型問題。兩個字元模型剩下的錯誤幾乎都是類型錯誤（三個種子共 124 與 102 個）：區段正確，但把人標成組織或反過來。這些都是單詞元名稱，常出現在同一位置可放任一類型的模板中，例如 `Mimitek reported a fault` 或 `checked by Renaka`。上下文無法決定這些情況；在這個語料中人名的拼字不帶類型標記；而只用 100 句訓練的模型，還沒有可靠地學會 `tek` 與 `corp` 字尾，不足以推翻也適合人名的上下文。

CRF 保證合法輸出，也稍有幫助。只用詞的標註器在一個種子中產生 30 個非法 `I-` 標籤，字元 softmax 標註器在兩個種子中分別產生 4 個與 1 個；具 BIO 限制的 CRF 依構造一個也沒有。它的平均 F1 為 0.929，對比 0.913，而且每個種子都至少與 softmax 標註器一樣好。在 100 句訓練資料下差距不大；這裡支持多加一層的，是改善的一致性與保證區段格式正確，而不是差距大小。

種子同樣重要。只用詞的標註器在不同種子間介於 0.771 到 0.893。產生 30 個非法轉移的種子，也是表現最差的那一個。與 NLP 05 一樣，單次執行是不可靠的摘要。

這個實驗沒有顯示什麼。語料是合成、極小且模板化的；測試句重複使用訓練模板；標籤集也很小。真實基準包含歧義上下文、巢狀與不連續提及，以及標註分歧。這個實驗把三種機制（未知詞處理、字元證據與輸出限制）分離出來，讓每個效果都能被看見與量測；它並未估計這些效果在 CoNLL-2003 或任何其他資料集上的大小。

轉移學到了什麼。圖 4 顯示一個種子學到的轉移分數。在允許的區域內，CRF 學到了語料的結構：`B-LOC → I-LOC` 與 `B-DEV → I-DEV` 明顯為正，而 `I-ORG → I-ORG` 與 `I-LOC → I-LOC` 明顯為負，因為這個語料中沒有任何名稱超過兩個詞元。學到的轉移表也是需要檢查的建模假設：在每個名稱最多兩個詞元的領域訓練後，它會在其他地方懲罰三詞元的名稱。

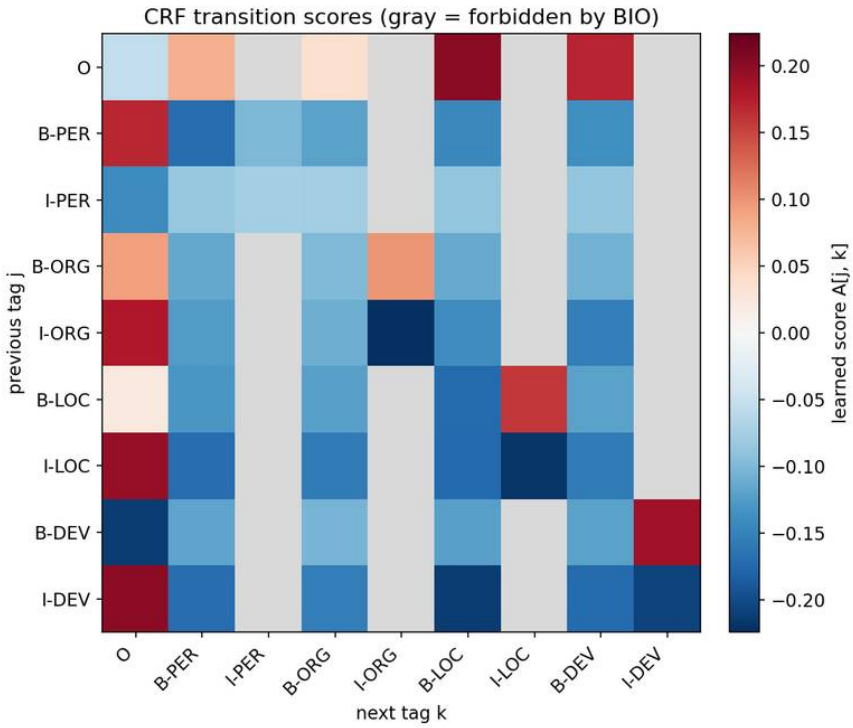


圖 *4* — *CRF* 學到的轉移分數（種子 *0*）。灰色格是限制遮罩禁止的 *BIO* 轉移；其餘分數由 *100* 句訓練資料學得。

現代觀點

直觀說法。當代擷取系統把預訓練表示與明確任務定義結合。通用生成提升彈性，但區段忠實度與支持性仍是另外的問題。

Li 等人的 NER 綜述依輸入表示、上下文編碼器與標籤解碼器整理早期深度系統 [7]。這個分類仍有用：換編碼器不會決定輸出是平面序列還是重疊區段。Xu 等人的 2024 生成式擷取綜述加入任務軸——實體、關係、事件與統一擷取——以及提示、例子、監督微調與受限生成等調適軸 [8]。應把這些圖譜當成設計維度，而不是普遍優越的架構排名。

從 BiLSTM-CRF 到預訓練編碼器。依 Li 等人的分類，本章實驗在保持 BiLSTM 上下文編碼器不變的情況下，改變輸入表示（只用詞或詞加字元）與標籤解碼器（softmax 或 CRF）[7]。2016 到 2018 年間，帶字元特徵的 BiLSTM-CRF [14]、[15] 成為標準做法，字元語言模型的上下文字串嵌入又進一步延伸了它 [16]。之後，預訓練 Transformer 編碼器在多數系統中取代 BiLSTM 成為上下文編碼器；其上的線性或 CRF 標籤解碼器，就是 NLP 10 與 NLP 11 討論的詞元分類輸出頭。本章的解碼器問題（合法轉移、區段邊界、精確評估）原封不動地延續下去。

以大型語言模型做開放式實體辨識。 UniversalNER（ICLR 2024）透過針對性的指令微調，把大型語言模型蒸餾成小得多的開放領域 NER 模型，並在由 9 個領域、43 個 NER 資料集組成的基準上評估 [22]。它與 GLiNER [9] 一起代表一種轉變：從固定的標籤集合，走向在推論時才提供實體類型。

GLiNER 是 2024 年以雙向 Transformer 做標籤條件式區段擷取的原始研究 [9]。它展示了改變所需實體類別，不一定要自迴歸生成一長串紀錄。結構化區段輸出可以直接對齊原文。應用比較要納入支援的類型定義、巢狀區段行為、延遲與錯誤覆核，不能只看平均準確率。

生成式擷取器則輸出所需紀錄的文字序列化。強制結構合法可避免缺少鍵值或語法損壞，但無法保證名稱、關係或證據區間正確。實務驗證層會檢查引文是否存在、位移是否匹配、類別是否允許，以及實體身分是否一致。這些檢查可排除部分錯誤，解讀錯誤仍需要評估或覆核。

開放問題包含持續改變的類別集合、專業語言、巢狀實體、跨文件身分與領域轉移下的可靠不確定性。多語擷取還面對不同名稱邊界、文字系統與稱謂慣例。語言特定指南是任務定義的一部分，翻譯類別名稱並不足夠。按來源與語言評估文件，同時保留足夠例數，才能解釋各項比較。

新專案應先建立確定性或小型監督基準，凍結標註指南，檢查錯誤類型，再選更大的模型。保留測試文件時，防止重複模板與近乎相同的報告跨越切分。比較完整且受支持的紀錄，不要只檢查是否能剖析輸出格式。這樣建立的擷取服務，才有可量測且能回溯證據的失敗模式。

新標籤同時測試表示與解讀

標籤條件式區段擷取 [9] 與生成式擷取 [8]，改變應用提供類別清單的方式。彈性帶來分析問題：模型辨認的是預定類別，還是只對熟悉標籤名稱反應？可以比較短標籤、精確描述，以及刻意相似的競爭描述。例如區分設備製造商與設備操作單位：兩者都可能是組織提及，但事件角色不同。

應分別判斷實體偵測、指定類型與關係配置。組織區段定位正確，不代表它製造了設備；生成紀錄若填滿每個要求欄位，也可能在來源省略參與者時編造一位。可以比較明確未知答案與強制填入。這個比較應揭露哪種輸出介面保留缺少資訊，以及哪些訓練範例能區分接近的結構規格。類別涵蓋擴大，只有在語意邊界仍清楚時才有用。

重點整理

- 序列標註替每個詞元指定一個標籤，但標籤彼此影響；結構化解碼器在既定分解下選出最佳完整序列。
- HMM 的 Viterbi 解碼在一階分解下是精確的；線性鏈 CRF 保留相同遞迴，但以條件方式替標籤評分，因此可由神經編碼器提供分數。
- 實體辨識從結構描述與位移開始；BIO 標籤編碼平面區段，巢狀提及需要區段表示。
- 評估精確區段（邊界與類型），並把錯誤拆成漏掉、多餘、邊界與類型錯誤，而不是回報詞元準確率。
- BiLSTM 標註器雙向讀取整個句子；遮罩與經過訓練的未知詞編號是不可少的細節。
- 以對數概似（前向演算法）訓練、以 Viterbi 解碼的 CRF 層讓標籤彼此一致；BIO 限制讓非法序列不可能出現。
- 字元特徵能從拼字找回未見名稱，卻無法解決拼字沒有編碼的歧義。
- 資訊擷取在實體區段之上建立關係與事件；每一層都有自己的紀錄契約與失敗模式。

練習

1. **序列決策**。改變 HMM 的一列轉移機率，同時保持正規化。找出「fish fish」最佳路徑變成兩個名詞的條件。

   提示

   比較完整路徑乘積。以名詞起始的兩個候選，共用起始因子與第一次發射。

2. 在 CPU 擷取範例加入名稱表沒有的新人物拼法，新增正確區段，回報精確召回率的變化。

   提示

   保持原名稱表固定。漏掉的提及是基準範圍的有用證據，不是把測試詞彙補進訓練的理由。

3. 為重疊的組織與部門名稱設計結構描述，解釋為何單一平面 BIO 序列無法表達所有允許標註。

   提示

   先畫字元區間，再決定巢狀名稱是否算獨立實體，以及部門是否連結母組織。

4. 在 PyTorch 實驗中移除 CRF 的 BIO 限制遮罩（把 `ban` 與 `ban_start` 設為零）並重新訓練。計算三個種子的非法 `I-` 標籤數，再檢查 `O` 後接 `I-ORG` 的學習轉移分數。

   提示

   沒有遮罩時，非法轉移只會被 100 句訓練資料教到的程度所抑制。把學到的分數與轉入 `I-ORG` 的合法轉移分數比較。

5. 加入只用字元的消融實驗：拿掉詞嵌入，只保留字元 BiLSTM。接著在產生器中替人名加上明顯字尾（例如 `-ko`），重新訓練所有模型。哪種錯誤類型改變了？為什麼？

   提示

   使用程式印出的錯誤分類。新的拼字線索應該只會把帶有該線索的名稱，從類型錯誤欄位中移出。

6. 寫一個部分匹配評估器：預測區段與同類型正確區段重疊時就給分。把它與只用詞標註器的精確 F1 並列，並解釋兩者為何可能以不同順序排列模型。

   提示

   精確定義匹配規則（例如依最大重疊做一對一匹配），並以它自己的名稱回報。只用詞標註器的邊界錯誤，正是兩種分數分歧之處。

參考文獻

1. Susan McRoy, *Principles of Natural Language Processing*, open textbook, 2021; Chapters 3–5 and 8. Open textbook
2. Masato Hagiwara, *Real-World Natural Language Processing: Practical Applications with Deep Learning*, Manning, 2021; Chapter 5, "Sequential labeling and language modeling." Chapter
3. L. R. Rabiner, "A tutorial on hidden Markov models and selected applications in speech recognition," *Proceedings of the IEEE*, 77(2), 1989. Paper
4. M.-C. de Marneffe, C. D. Manning, J. Nivre and D. Zeman, "Universal Dependencies," *Computational Linguistics*, 47(2), 2021. Paper
5. J. Lafferty, A. McCallum and F. Pereira, "Conditional random fields: Probabilistic models for segmenting and labeling sequence data," ICML, 2001. Paper
6. Erik F. Tjong Kim Sang and Fien De Meulder, "Introduction to the CoNLL-2003 Shared Task: Language-Independent Named Entity Recognition," CoNLL, 2003. Paper
7. Jing Li, Aixin Sun, Jianglei Han and Chenliang Li, "A Survey on Deep Learning for Named Entity Recognition," IEEE Transactions on Knowledge and Data Engineering 34(1), 2022. Author-hosted paper
8. Derong Xu, Wei Chen, Wenjun Peng, Chao Zhang, Tong Xu, Xiangyu Zhao, Xian Wu, Yefeng Zheng, Yang Wang and Enhong Chen, "Large language models for generative information extraction: a survey," Frontiers of Computer Science 18(6), 2024. Survey
9. Urchade Zaratiana, Nadi Tomeh, Pierre Holat and Thierry Charnois, "GLiNER: Generalist Model for Named Entity Recognition using Bidirectional Transformer," NAACL, 2024. Paper
10. Lance Ramshaw and Mitch Marcus, "Text Chunking using Transformation-Based Learning," Third Workshop on Very Large Corpora, 1995. Paper
11. Mitchell P. Marcus, Beatrice Santorini and Mary Ann Marcinkiewicz, "Building a Large Annotated Corpus of English: The Penn Treebank," *Computational Linguistics* 19(2), 1993. Paper
12. Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu and Pavel Kuksa, "Natural Language Processing (Almost) from Scratch," *Journal of Machine Learning Research* 12, 2011. Paper
13. Zhiheng Huang, Wei Xu and Kai Yu, "Bidirectional LSTM-CRF Models for Sequence Tagging," arXiv:1508.01991, 2015. arXiv
14. Guillaume Lample, Miguel Ballesteros, Sandeep Subramanian, Kazuya Kawakami and Chris Dyer, "Neural Architectures for Named Entity Recognition," NAACL-HLT, 2016. Paper
15. Xuezhe Ma and Eduard Hovy, "End-to-end Sequence Labeling via Bi-directional LSTM-CNNs-CRF," ACL, 2016. Paper
16. Alan Akbik, Duncan Blythe and Roland Vollgraf, "Contextual String Embeddings for Sequence Labeling," COLING, 2018. Paper
17. Sameer Pradhan, Alessandro Moschitti, Nianwen Xue, Hwee Tou Ng, Anders Björkelund, Olga Uryupina, Yuchen Zhang and Zhi Zhong, "Towards Robust Linguistic Analysis using OntoNotes," CoNLL, 2013. Paper
18. Leon Derczynski, Eric Nichols, Marieke van Erp and Nut Limsopatham, "Results of the WNUT2017 Shared Task on Novel and Emerging Entity Recognition," Workshop on Noisy User-generated Text, 2017. Paper
19. Christopher Walker, Stephanie Strassel, Julie Medero and Kazuaki Maeda, *ACE 2005 Multilin-*

gual Training Corpus, LDC2006T06, Linguistic Data Consortium, 2006. Catalog
20. Yuhao Zhang, Victor Zhong, Danqi Chen, Gabor Angeli and Christopher D. Manning, “Position-aware Attention and Supervised Data Improve Slot Filling,” EMNLP, 2017. Paper
21. Yuan Yao, Deming Ye, Peng Li, Xu Han, Yankai Lin, Zhenghao Liu, Zhiyuan Liu, Lixin Huang, Jie Zhou and Maosong Sun, “DocRED: A Large-Scale Document-Level Relation Extraction Dataset,” ACL, 2019. Paper
22. Wenxuan Zhou, Sheng Zhang, Yu Gu, Muhao Chen and Hoifung Poon, “UniversalNER: Targeted Distillation from Large Language Models for Open Named Entity Recognition,” ICLR, 2024. arXiv
23. Mike Schuster and Kuldip K. Paliwal, “Bidirectional Recurrent Neural Networks,” *IEEE Transactions on Signal Processing* 45(11), 1997. DOI

NLP 07・注意力機制與 Transformer

從編碼器—解碼器瓶頸出發，介紹 Bahdanau 與 Luong 注意力、縮放內積與多頭注意力、遮罩、位置，以及 Transformer 編碼器與解碼器區塊，並從零實作注意力與編碼器區塊。

網站版本: https://blog.shyandram.dev/zh-tw/learn/nlp/attention-transformers/

你將學到

- 為什麼把整個來源句壓進一個向量會限制編碼器—解碼器，以及注意力如何移除這個瓶頸。
- 加法式（Bahdanau）與乘法式（Luong）注意力的方程式，以及它們如何推廣成查詢、鍵與值。
- 縮放內積注意力、多頭注意力，以及自注意力（self-attention）與交叉注意力（cross-attention）的差別。
- 填補遮罩與因果遮罩如何規定每個位置可以讀什麼，以及如何測試它們確實有效。
- 為什麼注意力需要位置資訊，以及提供位置的主要方法。
- Transformer 編碼器與解碼器區塊如何組成、成本多少，以及哪些變體針對長上下文。

先看全貌

循環神經網路（NLP 05）一次讀一個詞元，並帶著一份持續更新的摘要。不論句子有五個詞還是五十個詞，這份摘要的大小都固定。注意力改變了讀取方式：不再只依賴一份摘要，每個位置都能回頭看所有已儲存的表示，並決定哪些對目前要做的預測有用。Transformer 把這個想法推到底，編碼器與解碼器都只用注意力和小型逐詞元網路組成，完全不需要循環。

> **用最簡單的話說**
>
> 想像你要翻譯一個長句。一種做法是讀一遍、闔上書，再憑記憶寫出譯文。另一種做法是讓書保持打開，每寫一個詞，就回頭看你正在翻譯的那一段。注意力就是第二種做法；「回頭看」是一組權重，說明每個詞要看多少。遮罩則是規定你可以看頁面哪些部分的規則：在某些遊戲裡，你不能偷看還沒寫出的詞。

本章由三個決定組織，值得分開看。**評分規則**決定一個位置與另一個位置有多匹配。**可見性規則**以遮罩表示，決定哪些位置根本可以被讀取。**位置規則**決定詞序如何進入一個原本會忽略詞序的計算。改進其中一項，修不好另一項的錯誤：更好的評分函數，無法修補洩漏未來詞元的遮罩。

Hagiwara 第 8 章從實務角度介紹注意力與 Transformer [1]。本章補上推導、對照原始論文，並用小型 CPU 範例實作每個機制。ML 05 從影像角度處理同一套機制，並提供完整的位置編碼數學；NLP 08 則用它建立翻譯模型。

1. 編碼器—解碼器的瓶頸

直觀說法。序列到序列模型有一個讀取來源的編碼器，以及一個寫出目標的解碼器。如果兩者之間唯一的橋樑是一個向量，解碼器需要的每個細節都必須擠過這個向量。

Sutskever、Vinyals 與 Le [2] 以及 Cho 等人 [3] 的循環編碼器—解碼器，以編碼器 RNN 讀取來源序列 $x_1, \dots, x_T$，再把最後狀態交給解碼器：

$$h_j = f_{\text{enc}}(h_{j-1}, x_j), \qquad c = h_T, \qquad s_u = f_{\text{dec}}(s_{u-1}, y_{u-1}, c), \qquad p(y_u \mid y_{<u}, \mathbf{x}) = g(s_u).$$

其中 $h_j \in \mathbb{R}^d$ 是讀完來源詞元 x_j 後的編碼器狀態；T 是來源長度；c 是唯一的上下文向量；s_u 是目標步驟 u 的解碼器狀態；y_{u-1} 是前一個目標詞元；$y_{<u}$ 是目標前綴；f_{enc} 與 f_{dec} 是 LSTM 或 GRU 單元之類的循環更新函數；g 是輸出層，後接目標詞彙表上的 softmax。ML 04 推導這些單元並訓練這種模型。

不論輸入是什麼，向量 c 的維度都是 d。十個詞的句子和六十個詞的段落，都必須在產生第一個目標詞之前壓進它。Bahdanau、Cho 與 Bengio 指出這個固定長度向量很可能是基本編碼器—解碼器的瓶頸，並提出讓解碼器改為在來源中搜尋相關部分 [4]。Sutskever 等人也回報，把來源句反轉有助於他們的 LSTM 模型，這與同一個診斷一致：縮短最早來源詞與最早目標詞之間的路徑，讓它們更容易被利用 [2]。

瓶頸關乎資訊流動，不只是容量。即使 c 很大，也必須在解碼器知道步驟 u 需要哪個來源細節之前就算好。注意力把這個選擇延後：編碼器為每個來源詞元保留一個向量，解碼器在每一步選擇不同的組合。

2. 加法式注意力（Bahdanau）

直觀說法。在每個目標步驟，拿解碼器目前的需求去替每個來源位置評分，把分數轉成總和為一的權重，再對編碼器狀態做加權平均。

Bahdanau 等人 [4] 保留全部編碼器狀態 $h_1, \dots, h_T$（來自雙向編碼器），並對目標步驟 u 計算：

$$e_{u,j} = v_a^\top \tanh\!\left(W_a s_{u-1} + U_a h_j\right), \qquad \alpha_{u,j} = \frac{\exp(e_{u,j})}{\sum_{k=1}^{T} \exp(e_{u,k})}, \qquad c_u = \sum_{j=1}^{T} \alpha_{u,j}\, h_j.$$

其中 s_{u-1} 是前一個解碼器狀態；h_j 是來源位置 j 的編碼器狀態；W_a、U_a 與 v_a 是學得的矩陣與向量；$e_{u,j}$ 是**對齊分數**；$\alpha_{u,j}$ 是**注意力權重**，非負且對 j 總和為一；c_u 是每一步各自的**上下文向量**，解碼器把它和自身狀態一起用來預測 y_u。記號與 ML 04 一致。由於分數先讓兩個向量通過一個小型前饋層再比較，這種形式稱為**加法式注意力**。

對 NLP 而言有兩個重要後果。第一，解碼器現在有通往每個來源詞元的直接路徑，第一個來源詞的資訊不必撐過多次循環更新。第二，權重 $\alpha_{u,j}$ 形成目標與來源位置之間的軟對齊，可以畫出來檢查。這種檢查是診斷，不是保證：如後文所述，高權重只顯示計算從哪裡讀取，不說明最終預測為何正確。

3. 乘法式注意力（Luong）

直觀說法。把小型評分網路換成內積，必要時中間夾一個學得矩陣。這更省計算，也讓注意力變成矩陣乘法。

Luong、Pham 與 Manning [5] 為一個讀取所有來源狀態的**全域**注意力模型比較了幾種評分函數。他們用目前（而非前一個）解碼器狀態 h_u 與每個來源狀態 $\bar{h}_j$ 評分：

$$\operatorname{score}(h_u, \bar{h}_j) = \begin{cases} h_u^\top \bar{h}_j & \text{dot} \\ h_u^\top W_a \bar{h}_j & \text{general} \end{cases} \qquad a_u(j) = \frac{\exp \operatorname{score}(h_u, \bar{h}_j)}{\sum_{j'} \exp \operatorname{score}(h_u, \bar{h}_{j'})}.$$

其中 h_u 是目標步驟 u 的頂層解碼器狀態；$\bar{h}_j$ 是來源位置 j 的編碼器狀態；W_a 是學得矩陣；$a_u(j)$ 是注意力權重。第三種選項 **concat** 像加法式注意力一樣，用前饋層對串接向量 $[h_u; \bar{h}_j]$ 評分。上下文向量 $c_u = \sum_j a_u(j) \bar{h}_j$ 與解碼器狀態結合成**注意力隱藏狀態**，再送進輸出 softmax：

$$\tilde{h}_u = \tanh(W_c [c_u; h_u]), \qquad p(y_u \mid y_{<u}, \mathbf{x}) = \operatorname{softmax}(W_s \tilde{h}_u).$$

其中 $[c_u; h_u]$ 是向量串接，W_c、W_s 是學得矩陣。Luong 等人也提出**局部**注意力，先預測一個對齊的來源位置，只讀取它附近的視窗；以及**輸入饋送**（input feeding），把 $\tilde{h}_u$ 送進下一個解碼步驟，讓模型知道自己先前的對齊決定 [5]。

「general」評分是通往 Transformer 的橋樑。令 $q_u = h_u$、$k_j = W_a^\top \bar{h}_j$，分數就只是 $q_u^\top k_j$。一旦每個分數都是**查詢**與**鍵**的內積，整張分數表就是一次矩陣乘法，正是現代硬體最擅長的運算。

4. 查詢、鍵與值：從對齊到自注意力

直觀說法。把解碼器狀態推廣成**查詢**（這個位置在找什麼），把來源狀態推廣成**鍵**（每個位置提供什麼來比對）與**值**（被讀到時傳出什麼）。如此注意力可在任意兩組向量之間運作，也包括序列和它自己。

在加法式與乘法式注意力中，同一個編碼器狀態同時用來比對和加權平均。Transformer 以三個學得投影分開這些角色 [6]。若查詢來自解碼器、鍵和值來自編碼器，這個運算就是**交叉注意力**，是 Bahdanau 與 Luong 注意力的直接後代。若查詢、鍵和值都來自同一個序列，就是**自注意力**：每個詞元讀取同一句中的其他詞元，建立自己的新表示。

自注意力產生**上下文表示**。在「the crane lifts steel」和「the crane eats fish」中，`crane` 的嵌入查表結果相同；但經過自注意力後，兩次出現讀到不同鄰居，輸出向量就能不同。這正是 NLP 10 發展成 BERT 的表示概念。重用這類表示必須配對分詞器與詞彙表：嵌入列只在該模型詞元 ID 的定義下有意義。未調適權重就換分詞器，可能讓形式合法的數字代表不相關符號。

5. 縮放內積注意力

直觀說法。每個位置詢問其他哪些位置含有有用資訊。注意力把比較轉成權重，再組合相應的值向量。

對單一注意力頭：

$$Q = HW_Q, \quad K = HW_K, \quad V = HW_V, \qquad A = \mathrm{softmax}_{\mathrm{row}}\left(\frac{QK^\top}{\sqrt{d_k}} + B\right), \quad O = AV.$$

其中 $H \in \mathbb{R}^{T\times d}$ 包含輸入表示；$W_Q, W_K \in \mathbb{R}^{d\times d_k}$ 是學得的查詢與鍵投影；$W_V \in \mathbb{R}^{d\times d_v}$ 是值投影；Q, K, V 是投影矩陣；d_k, d_v 是鍵和值的寬度；$B \in \mathbb{R}^{T\times T}$ 是加法可見性遮罩，元素為 0（允許）或 $-\infty$（阻擋）；A 包含按列正規化的注意力權重；$O \in \mathbb{R}^{T\times d_v}$ 是輸出。每個查詢列分別套用 softmax。這是 Transformer 論文介紹的機制 [6]。

在常見初始化假設下，縮放因子避免內積只因鍵向量維度增加而變大：若查詢與鍵的分量彼此獨立、平均為零、變異數為一，$q^\top k$ 的變異數就是 d_k；除以 $\sqrt{d_k}$ 讓變異數回到一，softmax 才不會飽和（ML 05 有完整推導）。它不會把注意力權重變成校準後的語意重要性。高權重表示經過學得投影後，在這次計算有較大貢獻；不會證明該詞元造成最終答案，也不會證明它支持生成的陳述。

程式碼：在玩具句子上從零實作注意力

為了不經訓練就看清方程式在做什麼，我們手動建立向量。「the pump stopped because it overheated」的每個詞元得到一個四特徵向量（裝置名詞、代名詞、事件動詞、功能詞），句子再補上一個 <pad>，就像在批次中一樣。投影的設計讓代名詞與動詞**尋找**裝置、名詞尋找事件，代名詞也微弱地**提供**裝置資訊。這裡沒有任何學習；重點是觀察評分、遮罩與平均如何運作。

```
import numpy as np

tokens = ["the", "pump", "stopped", "because", "it", "overheated", "<pad>"]
# Hand-built features: [device noun, pronoun, event verb, function word]
X = np.array([[0, 0, 0, 1], [1, 0, 0, 0], [0, 0, 1, 0], [0, 0, 0, 1],
              [0, 1, 0, 0], [0, 0, 1, 0], [0, 0, 0, 0]], dtype=float)
# Query space: [looking for a device, looking for an event]
W_Q = 3.0 * np.array([[0, 1], [1, 0], [1, 0], [0, 0]])   # nouns seek events; pronouns, verbs
 ↪  seek devices
W_K = np.array([[1, 0], [0.5, 0], [0, 1], [0, 0]])       # devices offer "device"; a pronoun
 ↪  weakly so
W_V = np.eye(4)                                          # pass the features through
 ↪  unchanged

def attention(X, mask):
    Q, K, V = X @ W_Q, X @ W_K, X @ W_V
    scores = Q @ K.T / np.sqrt(K.shape[1])
    scores = np.where(mask, scores, -np.inf)              # blocked pairs get zero weight
    A = np.exp(scores - scores.max(axis=1, keepdims=True))
    A /= A.sum(axis=1, keepdims=True)
    return A @ V, A

T = len(tokens)
real = np.array([t != "<pad>" for t in tokens])
padding_mask = np.broadcast_to(real, (T, T))              # every query may read real keys
causal_mask = padding_mask & np.tri(T, dtype=bool)        # ...and only earlier ones
out, A = attention(X, padding_mask)
_, A_causal = attention(X, causal_mask)
```

```
it, pump = tokens.index("it"), tokens.index("pump")
print("weights of 'it':", " ".join(f"{t}={w:.2f}" for t, w in zip(tokens, A[it])))
print("output of 'it' (device, pronoun, event, function):", out[it].round(2))
print("'pump' reads 'stopped': bidirectional", A[pump, 2].round(2),
      "| causal", A_causal[pump, 2].round(2))
X2 = X.copy(); X2[-1] = [5, 5, 5, 5]                       # garbage in the padding slot
out2, _ = attention(X2, padding_mask)
print("padding changes real outputs:", not np.allclose(out[real], out2[real]))
assert np.allclose(A.sum(axis=1), 1) and A[:, -1].max() == 0
```

輸出：

```
weights of 'it': the=0.07 pump=0.55 stopped=0.07 because=0.07 it=0.19 overheated=0.07
 ↪  <pad>=0.00
output of 'it' (device, pronoun, event, function): [0.55 0.19 0.13 0.13]
'pump' reads 'stopped': bidirectional 0.4 | causal 0.0
padding changes real outputs: False
```

代名詞的新向量大多是它從 pump 借來的裝置特徵：經過一步注意力，it 已帶有可能先行詞的資訊。填補欄的權重恰為零，因此即使填補位置存了垃圾，也改變不了真實詞元的輸出。在因果遮罩下，pump 不能再讀較晚的動詞 stopped。真實模型會學習投影；這裡是手動寫出，而且範例刻意只有一個裝置，所以不能說明訓練好的模型如何解決較難的指涉（那是共指問題，見 NLP 12）。

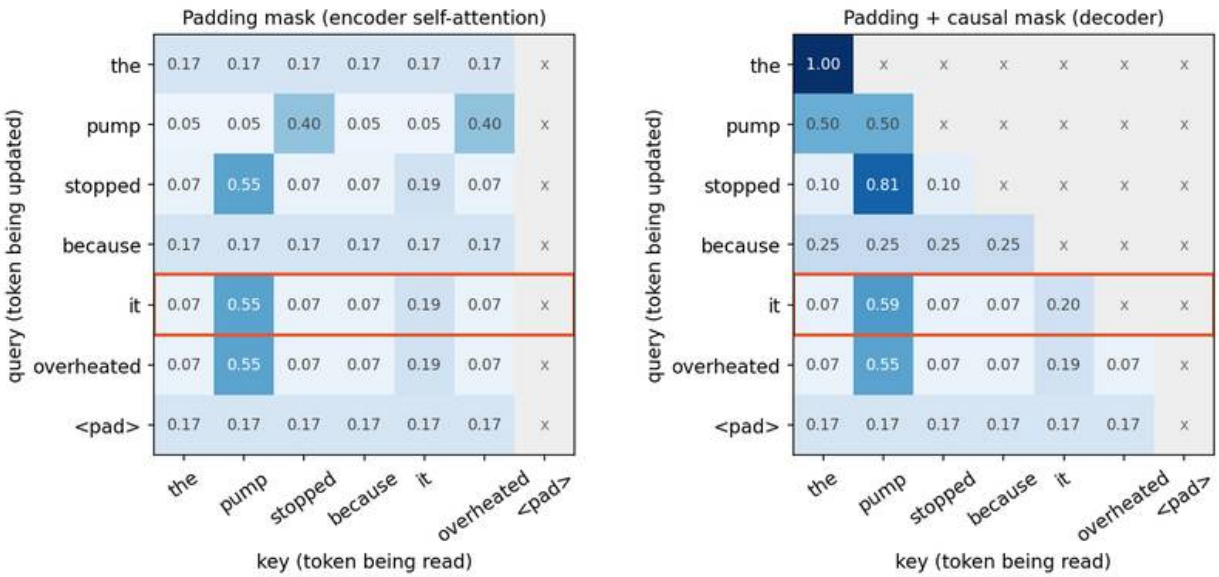


圖 *1* — 手動建立的玩具範例。左：只有填補遮罩，如編碼器。右：填補加因果遮罩，如解碼器。標 *x* 的灰格被阻擋；橘色框是代名詞那一列。權重來自手動選定的投影，不是訓練好的模型。

請注意圖中的 <pad> **列**：填補查詢仍會產生一個（無意義的）輸出，因為它的鍵並未全部被阻擋。這讓本例採用右側填補時的手寫 softmax 保持有定義。若一整列都被阻擋，一般 softmax 沒有定義並可能產生 NaN；不要假設不同核心會以相同方式處理。應拒絕沒有真實詞元的空序列，或明

確處理只有填補的查詢列，尤其是左側填補與因果遮罩一起使用時。應遮住填補**鍵**，並在損失中忽略填補**輸出**。

6. 多頭注意力

直觀說法。平行執行幾個較小的注意力運算，每個有自己的投影，讓不同的頭比對不同關係。再把輸出串接起來，用另一個投影混合。

有 h 個頭時，第 r 個頭有自己的投影 $W_Q^{(r)}, W_K^{(r)}, W_V^{(r)} \in \mathbb{R}^{d\times d/h}$：

$$\text{head}_r = \text{Attention}\big(HW_Q^{(r)}, HW_K^{(r)}, HW_V^{(r)}\big), \qquad \text{MHA}(H) = [\,\text{head}_1, \dots, \text{head}_h\,]\,W_O.$$

其中 Attention 是第 5 節的縮放內積運算，且 $d_k = d_v = d/h$；方括號把 h 個輸出（各為 $T \times d/h$）串接回寬度 d；$W_O \in \mathbb{R}^{d\times d}$ 是輸出投影。原始 Transformer 使用 $h = 8$ 個頭、模型寬度 $d = 512$ [6]。由於每個頭窄了 h 倍，多頭注意力的成本約等於一個全寬度的頭，四個投影矩陣共有 $4d^2$ 個權重（另加偏差），與 ML 05 的計數相同。

一個頭對每個查詢只形成一個加權平均。若動詞同時需要主詞與受詞，單一 softmax 必須把權重分給兩者，兩邊都會變模糊；多個頭則能各自集中在一種關係上。不過不要過度解讀：頭不保證學到人類可讀的角色，有些頭是多餘的。

後來的兩種變體只改變鍵與值的共享方式。**多查詢注意力**（multi-query attention）讓所有查詢頭共用一個鍵頭和一個值頭，縮小每個解碼步驟要讀取的記憶體 [15]。**分組查詢注意力**（grouped-query attention）讓每個鍵／值頭由一組查詢頭共用，是兩者之間的折衷 [16]。兩者主要影響快速自迴歸解碼，見 NLP 09。

7. 自注意力與交叉注意力

直觀說法。同一個運算可以連接同一序列內的詞元（自注意力），也可以連接兩個序列（交叉注意力）。唯一的差別是查詢、鍵和值從哪裡來。

機制	查詢來自	鍵與值來自	NLP 中的典型用途
雙向自注意力	該序列	同一序列的所有真實詞元	BERT 等編碼器；翻譯器的來源端
因果自注意力	該序列	同一序列的目前與較早位置	僅解碼器語言模型；翻譯器的目標端
交叉注意力	解碼器	編碼器輸出	編碼器—解碼器翻譯與摘要

對交叉注意力，令 $H_{\text{dec}} \in \mathbb{R}^{T_y\times d}$ 為解碼器狀態，$M \in \mathbb{R}^{T_x\times d}$ 為編碼器輸出（常稱為**記憶**，memory）：

$$Q = H_{\text{dec}}W_Q, \qquad K = MW_K, \qquad V = MW_V, \qquad O = \text{softmax}_{\text{row}}\left(\frac{QK^\top}{\sqrt{d_k}} + B_{\text{src}}\right)V.$$

其中 T_y 與 T_x 是目標與來源長度；分數矩陣形狀為 $T_y \times T_x$，每個目標位置一列；B_{src} 阻擋來源填補；$O \in \mathbb{R}^{T_y \times d_v}$ 是每個目標位置的輸出。這就是加上學得查詢、鍵、值投影的 Bahdanau 與 Luong 注意力。這裡不需要因果遮罩，因為解碼開始前整個來源都已知。

應由任務可取得的資訊決定遮罩，而不是模型名稱。完整句子已知時，分類器可使用雙向編碼器；串流延續則必須排除未來輸出。翻譯可以在來源全部抵達後讀取完整來源，但同步翻譯（來源句還沒說完就開始翻）需要更嚴格的來源可見性規則。

8. 遮罩：填補與因果

直觀說法。遮罩是一張表，說明哪個查詢可以讀哪個鍵。填補遮罩隱藏人為的填充詞元；因果遮罩隱藏未來。

因果自注意力在 $j \le i$ 時令 $B_{ij} = 0$，否則令 $B_{ij} = -\infty$。其中 i 是查詢位置，j 是候選鍵位置。目標已位移，所以目前輸入位置仍是允許的前綴詞元，對角線可見。完整雙向自注意力允許所有真實輸入位置，但仍可阻擋填補位置。填補遮罩則在鍵 j 是填補位置時，對每個查詢 i 令 $B_{ij} = -\infty$。結合多個遮罩時，只要任一遮罩阻擋某一對，該對就被阻擋。

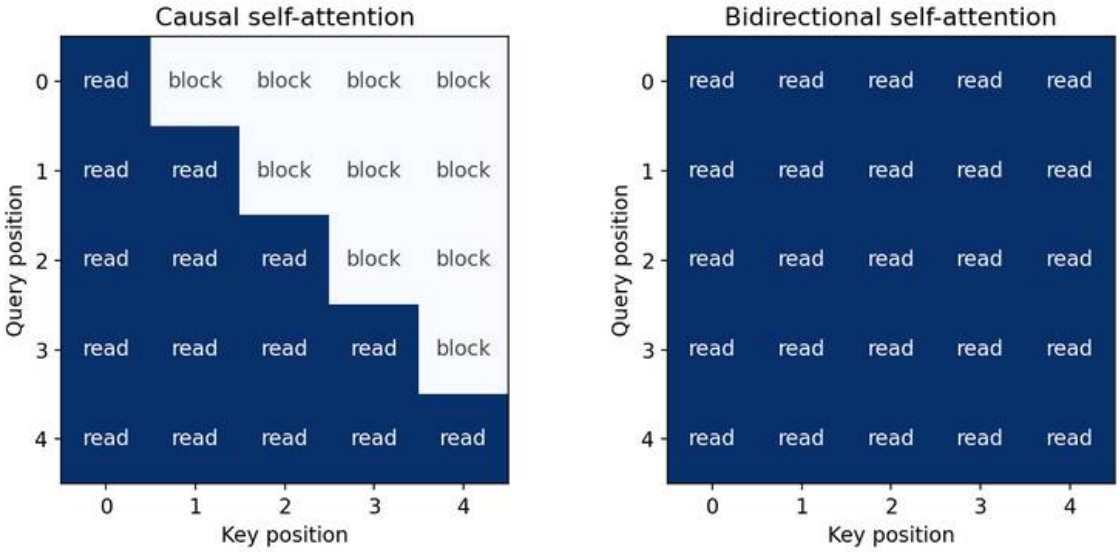


圖 2 — 可見性是任務定義的一部分。列是發出查詢的位置，欄是它們可以讀取的位置。

有兩種遮罩容易混淆。**注意力遮罩**控制表示可以讀什麼；**損失遮罩**控制哪些目標位置貢獻訓練目標。填補兩者都需要：填補鍵不得貢獻上下文，填補目標不得貢獻損失。它們解決不同問題，應分別測試。

布林遮罩的意思也取決於 API。在 PyTorch 的 `scaled_dot_product_attention` 中，`True` 代表此配對**允許**參與注意力 [22]。在 `MultiheadAttention` 中，`attn_mask` 的 `True` 代表**阻擋**，`key_padding_mask` 的 `True` 則標記要忽略的鍵 [23]。先用 `allowed` 或 `blocked` 命名邏輯關係，再明確轉換；直接重用同一個布林陣列，可能悄悄反轉資訊流規則。浮點遮罩則是加到分數上的偏移，例如零或負無限大；由零與一組成的浮點陣列不等同於布林遮罩。

以下獨立範例使用單位投影與均勻分數，讓預期輸出可以直接檢查。第一個查詢只讀第一個值，第二個查詢平均兩個值。反轉布林遮罩後，兩個 API 的結果相同。SDPA 也直接接收 dropout 機率，因此評估時必須明確傳入 `dropout_p=0.0` [22]。

```
import torch
from torch import nn
from torch.nn import functional as F

# Two queries and keys, with zero scores: allowed keys receive equal weights.
q = torch.zeros(1, 2, 2)
k = torch.zeros_like(q)
v = torch.tensor([[[10.0, 0.0], [0.0, 20.0]]])
allowed = torch.tril(torch.ones(2, 2, dtype=torch.bool))
blocked = ~allowed
sdpa = F.scaled_dot_product_attention(q[:, None], k[:, None], v[:, None],
                                      attn_mask=allowed, dropout_p=0.0)[:, 0]
mha = nn.MultiheadAttention(2, 1, bias=False, dropout=0.0, batch_first=True).eval()
with torch.no_grad():
    mha.in_proj_weight.copy_(torch.eye(2).repeat(3, 1))
    mha.out_proj.weight.copy_(torch.eye(2))
    module_out, _ = mha(q, k, v, attn_mask=blocked, need_weights=False)
expected = torch.tensor([[[10.0, 0.0], [5.0, 10.0]]])
assert torch.allclose(sdpa, expected)
assert torch.allclose(module_out, expected)
print('mask conventions agree:', torch.allclose(sdpa, module_out))
print('outputs:', sdpa[0].tolist())
```

```
mask conventions agree: True
outputs: [[10.0, 0.0], [5.0, 10.0]]
```

檢查注意力並測試資訊流

這個獨立矩陣範例把學得的分數與可見性規則分開。向量與投影皆為合成，因此權重沒有語言學解釋。

```
import numpy as np

rng = np.random.default_rng(0)
T, d = 5, 8                                  # 5 tokens, 8-dimensional embeddings
X = rng.normal(size=(T, d))                  # token embeddings (+ positions)
Wq, Wk, Wv = (rng.normal(size=(d, d)) / np.sqrt(d) for _ in range(3))

def attention(X, causal):
    Q, K, V = X @ Wq, X @ Wk, X @ Wv
    scores = Q @ K.T / np.sqrt(d)            # (T, T) query-key similarities
    if causal:                               # position t may only look at <= t
        scores = np.where(np.tri(T, dtype=bool), scores, -np.inf)
    w = np.exp(scores - scores.max(axis=1, keepdims=True))
    w /= w.sum(axis=1, keepdims=True)        # softmax over keys
```

```
    return w @ V, w

out, w = attention(X, causal=True)
print(np.round(w, 2))

# Changing the LAST token must not change any earlier causal output...
X2 = X.copy(); X2[-1] += 5.0
out2, _ = attention(X2, causal=True)
print("causal: earlier outputs unchanged:", np.allclose(out[:-1], out2[:-1]))
# ...but in a bidirectional encoder every position can see it.
b1, _ = attention(X, causal=False); b2, _ = attention(X2, causal=False)
print("bidirectional: earlier outputs unchanged:", np.allclose(b1[:-1], b2[:-1]))
```

輸出：

```
[[1.   0.   0.   0.   0.  ]
 [0.88 0.12 0.   0.   0.  ]
 [0.42 0.29 0.29 0.   0.  ]
 [0.28 0.24 0.29 0.19 0.  ]
 [0.34 0.06 0.24 0.11 0.26]]
causal: earlier outputs unchanged: True
bidirectional: earlier outputs unchanged: False
```

因果權重是下三角矩陣：第一個詞元只能讀自己，最後一個可以讀全部。數值檢查示範了遮罩的可見性規則。更一般地說，第 t 列只聚合位置不超過 t 的值向量。若查詢、鍵和值也由不含未來資訊的狀態計算而來，未來輸入就無法影響該輸出。這個條件必須在整個網路都成立，也包括任何混合不同位置的前處理。相同遮罩讓所有下一詞元訓練損失可以平行計算，同時讓每個預測只使用自己的前綴。

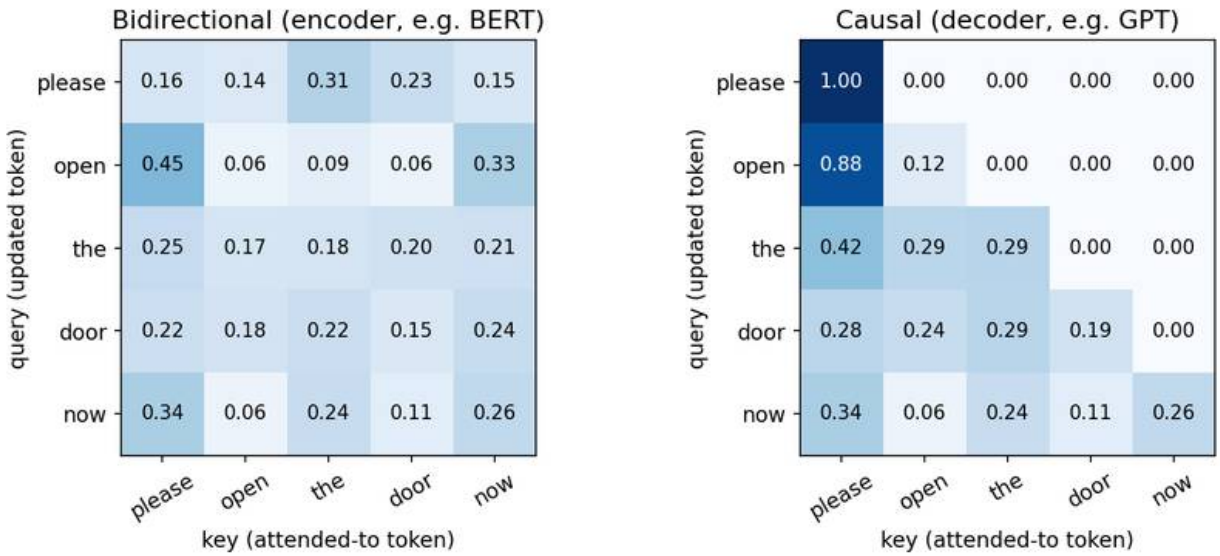


圖 3 — 上方程式碼的注意力權重（投影是隨機的，所以數值本身沒有語言學意義）。決定每個詞元能讀什麼的是遮罩，而不是權重。

在任何真實程式庫中，都值得保留這類擾動測試。低訓練損失揭露不了洩漏的遮罩，因為能看到答案的模型同樣能得到低損失；只有改變未來輸入、再檢查較早輸出的測試，才直接針對可見性契約。

9. 位置：告訴注意力詞元在哪裡

直觀說法。注意力比較詞元表示，位置資訊則告訴模型詞元在哪裡。詞元身分本身不會說明它是主詞還是最後一個詞。

沒有位置資訊時，自注意力具有**排列等變性**（permutation equivariance）：打亂輸入詞元，輸出會以相同方式被打亂，因為公式中沒有任何東西提到順序。「Dog bites man」與「man bites dog」會得到同一組向量。第 10 節的編碼器區塊程式碼會檢查這個性質，並顯示加入位置後性質消失。

最簡單的做法是在第一層之前，把位置向量加到每個詞元嵌入。使用學得的嵌入與位置表：

$$\mathbf{h}_i^{(0)} = E[x_i] + P[i].$$

其中 $\mathbf{h}_i^{(0)} \in \mathbb{R}^d$ 是位置 i 的初始表示；E 是詞元嵌入表；$E[x_i]$ 選取詞元 x_i 的列；P 是位置表；$P[i]$ 選取位置列；d 是表示寬度。學得的位置表也限制支援的最遠位置。超過範圍的序列不是普通長輸入，而是非法索引。服務契約應記錄上下文限制。擴充位置表或改變位置行為屬於模型變更，需要評估，不能直接假定長文件表現維持不變。

原始 Transformer 則使用固定的**正弦**（sinusoidal）編碼 [6]：

$$P[i]_{2m} = \sin\!\left(\frac{i}{10000^{2m/d}}\right), \qquad P[i]_{2m+1} = \cos\!\left(\frac{i}{10000^{2m/d}}\right), \qquad m = 0, \dots, \tfrac{d}{2} - 1.$$

其中 i 是位置，m 是一對維度的索引，每一對以自己的頻率旋轉。這與 ML 05 的公式相同（該頁寫成 $p_{pos,2i}$）；那裡證明了使它有用的兩個性質（固定位移相當於旋轉、內積只依賴位移），並畫出圖樣。原論文也在加上編碼之前，把詞元嵌入乘以 $\sqrt{d}$ [6]；NLP 08 的翻譯模型沿用這個慣例。

後來的方法把位置直接放進注意力分數，讓模型看到**相對**位移：

- **相對位置表示**（relative position representations）把依賴截斷位移 $j - i$ 的學得向量加到鍵與值上 [9]。
- **旋轉位置嵌入**（RoPE）把每個查詢與鍵旋轉一個與位置成正比的角度，使分數 $q_i^\top k_j$ 只透過 $i - j$ 依賴位置 [10]。
- **ALiBi** 對每個因果注意力分數加上固定、依頭而異、與距離 $i - j$ 成正比的懲罰，設計目標是讓在短輸入上訓練的模型能在較長輸入上評估 [11]。

這個選擇和上下文長度契約互相影響。學得的絕對位置表有硬性上限；相對方法移除了硬性上限，但不保證在超過訓練長度時仍準確。請在打算服務的長度上評估。

10. Transformer 編碼器與解碼器區塊

直觀說法。一個區塊輪流做兩件事。注意力讓詞元交換資訊；小型前饋網路再各自處理每個詞元。殘差連接與正規化讓深層堆疊可以訓練。解碼器在兩者之間多做一件事：透過交叉注意力讀

取編碼器。

原始**編碼器區塊** [6] 把每個子層包在殘差連接中，後接層正規化（**後正規化**，post-norm）：

$$U = \mathrm{LN}(H + \mathrm{MHA}(H)), \qquad H' = \mathrm{LN}(U + \mathrm{FFN}(U)), \qquad \mathrm{FFN}(u) = W_2 \max(0, W_1 u + b_1) + b_2.$$

其中 $H \in \mathbb{R}^{T \times d}$ 是區塊輸入；MHA 是帶填補遮罩的多頭自注意力；LN 是層正規化，把每個詞元的 d 個特徵標準化，再套用學得的縮放與平移 [7]；FFN 對每個位置分別套用，$W_1 \in \mathbb{R}^{d_{\mathrm{ff}} \times d}$、$W_2 \in \mathbb{R}^{d \times d_{\mathrm{ff}}}$，隱藏寬度為 d_{ff}（原始模型在 $d = 512$ 時為 2048）；H' 是區塊輸出。許多後來的模型把正規化移到每個子層**之前**（**前正規化**，pre-norm，即 $H + \mathrm{MHA}(\mathrm{LN}(H))$）。Xiong 等人分析了為何後正規化需要學習率暖身，而前正規化可以不用 [8]。因此，在說明正規化放在哪裡之前，「一個 Transformer」並未完整指定。

解碼器區塊有三個子層，各有自己的殘差連接與正規化：

1. 對目標前綴的**遮罩自注意力**，使用因果遮罩（批次時再加填補遮罩）。
2. **交叉注意力**，查詢來自第 1 步，鍵和值來自編碼器的最終輸出，只遮蔽來源填補。
3. 逐位置的**前饋網路**。

堆疊 N 個編碼器區塊與 N 個解碼器區塊，就得到完整模型（原始模型 $N = 6$ [6]）。每個解碼器區塊都讀同一個編碼器最終輸出。線性層與目標詞彙表上的 softmax 把最後的解碼器狀態轉成 $p(y_t \mid y_{<t}, \mathbf{x})$。去掉編碼器與交叉注意力，就成為**僅解碼器**模型，也就是目前多數大型語言模型的形狀（NLP 09）；去掉解碼器，就成為 BERT 之類的**僅編碼器**模型（NLP 10）。

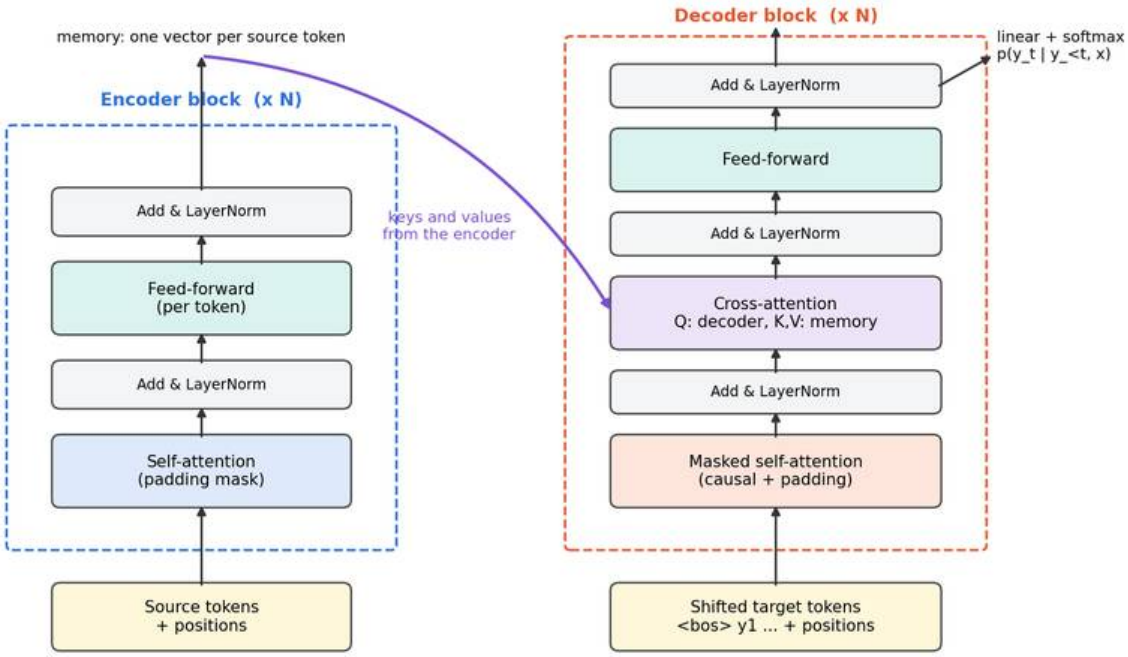


圖 4 — 原始後正規化 *Transformer* 的編碼器與解碼器區塊。每個子層周圍的殘差連接併入 *Add & LayerNorm* 方塊。紫色箭頭是交叉注意力：每個解碼器區塊都讀編碼器的最終輸出。

程式碼：從零實作微型 Transformer 編碼器區塊

以下程式碼直接依方程式實作多頭自注意力與後正規化編碼器區塊，借用 PyTorch `nn.TransformerEncoderLayer` 隨機初始化的權重 [21]，並檢查兩者輸出相同。接著測試三個性質：填補被忽略、沒有位置時區塊具排列等變性、正弦位置打破這個對稱。為了讓比較精確，dropout 關閉，參考層處於評估模式。

```
import math
import torch
from torch import nn
from torch.nn import functional as F

torch.manual_seed(0)
B, T, d, h, d_ff = 2, 6, 16, 4, 64
ref = nn.TransformerEncoderLayer(d, h, d_ff, dropout=0.0, activation="relu",
                                 batch_first=True, norm_first=False).eval()

def multi_head_attention(x, key_padding, p):
    q, k, v = (x @ p.in_proj_weight.T + p.in_proj_bias).chunk(3, dim=-1)
    split = lambda t: t.view(B, T, h, d // h).transpose(1, 2)        # (B, h, T, d/h)
    q, k, v = split(q), split(k), split(v)
    scores = q @ k.transpose(-2, -1) / math.sqrt(d // h)               # (B, h, T, T)
    scores = scores.masked_fill(key_padding[:, None, None, :], float("-inf"))
    heads = scores.softmax(dim=-1) @ v                                  # (B, h, T, d/h)
    concat = heads.transpose(1, 2).reshape(B, T, d)                     # heads side by side
    return concat @ p.out_proj.weight.T + p.out_proj.bias

def encoder_block(x, key_padding, layer=ref):
    # Post-norm, as in the original Transformer: x <- LN(x + Sublayer(x))
    a = multi_head_attention(x, key_padding, layer.self_attn)
    x = F.layer_norm(x + a, (d,), layer.norm1.weight, layer.norm1.bias, layer.norm1.eps)
    f = layer.linear2(F.relu(layer.linear1(x)))
    return F.layer_norm(x + f, (d,), layer.norm2.weight, layer.norm2.bias, layer.norm2.eps)

x = torch.randn(B, T, d)
pad = torch.tensor([[False] * 6, [False] * 4 + [True] * 2])     # second sentence: 4 real
 ↪  tokens
with torch.no_grad():
    ours, theirs = encoder_block(x, pad), ref(x, src_key_padding_mask=pad)
    print("matches nn.TransformerEncoderLayer:", torch.allclose(ours[~pad], theirs[~pad],
     ↪  atol=1e-5))
    x2 = x.clone(); x2[1, 4:] = 100.0                               # garbage in padded slots
    print("padding is ignored:", torch.allclose(encoder_block(x2, pad)[~pad], ours[~pad],
     ↪  atol=1e-5))
    perm = torch.tensor([3, 0, 5, 1, 4, 2])
    none = torch.zeros(B, T, dtype=torch.bool)
    y, yp = encoder_block(x, none), encoder_block(x[:, perm], none)
    print("no positions -> permutation-equivariant:", torch.allclose(yp, y[:, perm],
     ↪  atol=1e-5))
    pos = torch.arange(T)[:, None] / 10000 ** (torch.arange(0, d, 2) / d)
    pe = torch.stack([pos.sin(), pos.cos()], dim=-1).flatten(1)     # sin/cos interleaved
    y, yp = encoder_block(x + pe, none), encoder_block(x[:, perm] + pe, none)
    print("with positions -> still equivariant:", torch.allclose(yp, y[:, perm], atol=1e-5))
```

```
print("parameters:", sum(p.numel() for p in ref.parameters()))
```

輸出：

```
matches nn.TransformerEncoderLayer: True
padding is ignored: True
no positions -> permutation-equivariant: True
with positions -> still equivariant: False
parameters: 3280
```

比較只使用真實詞元：在評估模式下，PyTorch 可能走融合路徑，不在填補位置計算有意義的值，而這些輸出本來就會被忽略。參數數量驗證了公式：注意力有 $4d^2 + 4d = 1088$ 個權重與偏差，前饋網路有 $2d\,d_{\text{ff}} + d_{\text{ff}} + d = 2128$ 個，兩個層正規化有 $4d = 64$ 個。注意參數分布：在這個區塊中，如同原始 Transformer，前饋網路占了大多數。

在加上位置之前，模型沒有任何位置資訊，第三和第四項檢查正顯示了這一點。真實模型只在第一個區塊的輸入加一次位置向量；後面的區塊都透過殘差流繼承順序資訊。

僅線上版：*Transformer — Attention Is All You Need* — *https://blog.shyandram.dev/zh-tw/learn/nlp/attention-transformers/papers/transformer–attention-is-all-you-need/*

11. 成本與長上下文變體

直觀說法。每個查詢都和每個鍵比較，所以運算量與分數表會隨序列長度平方成長。對一個句子很便宜，對一本書就很昂貴。

對長度 T、寬度 d 的序列，一個自注意力層計算 $QK^\top$ 與 AV 各約需 $T^2 d$ 次乘加，投影另需約 $4Td^2$。循環層約需 Td^2，但需要 T 個**依序**步驟；自注意力層的所有位置則可平行計算，任兩個位置之間的路徑長度為一。Vaswani 等人在比較各類層時整理了這個取捨 [6]。對短句（T 小於 d），自注意力比較便宜；對長文件，T^2 項會占主導。

直接儲存注意力分數會隨序列長度平方成長。專用實作可避免實體化每個中間矩陣，但哪些位置需要互動的建模問題仍在。部署時，應按實際上下文長度量測記憶體與延遲，不能由短句直接外推。

長上下文方法可分成幾個家族。Tay 等人的綜述把它們整理為固定模式、可學模式、低秩與核方法、記憶、循環與降取樣 [17]。代表性例子：

想法	例子	改變了什麼
限制哪些位置對互動	Longformer：滑動局部視窗加少數全域詞元 [12]	固定視窗下成本隨 T 線性成長；遠距詞元透過全域詞元或多層溝通
替換 softmax 核	使用特徵映射的線性注意力 [13]	分數可分解，得到線性成本與因果注意力的循環觀點；注意力函數本身不同
保留精確注意力，減少記憶體搬移	FlashAttention：分塊、IO 感知計算 [14]	結果與標準注意力相同，記憶體流量較少；仍是平方級運算

想法	例子	改變了什麼
讓多個頭共用鍵與值	多查詢與分組查詢注意力 [15, 16]	解碼時的鍵／值快取較小
可延伸的位置方法	相對位置、RoPE、ALiBi [9–11]	移除硬性位置上限；在未見長度的準確度仍需量測

這些方法不能互換。FlashAttention 計算的函數與標準注意力相同；Longformer 與線性注意力計算的是不同函數，必須當成不同模型評估。完全不計算成對分數的狀態空間與循環替代方案，見 ML 04 與 ML 05。

現代觀點

直觀說法。 2017 年的注意力區塊在當前模型中仍清晰可辨；多數改變在於正規化位置、位置處理、鍵／值共享與高效核心運算，而不是核心方程式。

綜述。 Lin、Wang、Liu 與 Qiu 的 Transformer 變體綜述（*AI Open*，2022）先介紹原始模型，再提出分類法，從架構修改、預訓練與應用三個角度回顧變體 [18]。它很適合用來定位某個變體（「這是改注意力、改位置，還是改區塊？」）。Tay、Dehghani、Bahri 與 Metzler 的 *Efficient Transformers: A Survey*（*ACM Computing Surveys*，2022）聚焦注意力的記憶體與運算，並依機制分類高效模型（即第 11 節的家族）[17]。它的實務訊息是：效率主張取決於序列長度、硬體與實作，因此要在你需要的長度上比較模型。

注意力作為解釋。 Jain 與 Wallace 回報，在他們的模型中，注意力權重常與基於梯度的重要性量測不一致，而且非常不同的注意力分布可以得到相同預測 [19]。Wiegreffe 與 Pinter 則主張，注意力能否作為解釋取決於如何定義「解釋」，並提出檢驗方法 [20]。綜合來看：注意力圖顯示計算從哪裡讀取，對除錯遮罩與對齊很有用，但它本身不是模型為何做出某個決定的證據。

2023–2026 實務現況。分組查詢注意力 [16] 與 IO 感知的精確注意力核心 [14]，分別處理主導服務成本的兩件事：解碼時的鍵／值記憶體，以及注意力計算中的記憶體流量。RoPE [10] 與 ALiBi [11] 等相對位置方法的設計目標，是移除學得位置表的硬性最大長度。編碼器—解碼器 Transformer 仍很適合有明確來源與目標的任務，例如翻譯（NLP 08）；開放式生成則以僅解碼器模型為主（NLP 09）。

開放問題。可靠地使用極長上下文（較長視窗允許更多輸入，但不保證全部被利用）、超過訓練長度的長度泛化、長文件注意力的成本，以及忠實解讀注意力頭到底計算了什麼。

重點整理

- 單一上下文向量迫使整個來源通過固定大小的瓶頸；注意力讓每個輸出步驟都能讀取每個來源位置。
- Bahdanau 注意力用小型前饋網路評分；Luong 的 dot 與 general 評分把注意力化為矩陣乘法，直接引出查詢、鍵與值。
- 縮放內積注意力是 $\text{softmax}(QK^\top/\sqrt{d_k} + B)V$；多頭注意力以約一個頭的成本執行 h 個較窄的副本。

- 自注意力讓序列與自己混合；交叉注意力讓解碼器讀編碼器。決定每個位置能讀什麼的是遮罩，而不是權重。
- 遮蔽填補鍵、在因果模型中遮蔽未來、讓損失遮罩分開處理，並用擾動檢查測試可見性。
- 注意力忽略順序；學得、正弦、相對、旋轉或 ALiBi 位置提供順序，各自有不同的長度契約。
- 注意力成本隨 T^2 成長；長上下文變體不是改變函數（稀疏、線性），就是更有效率地計算同一個函數（FlashAttention）。

練習

1. 移除第 8 節資訊流範例的因果遮罩。哪一行輸出會改變？為什麼不使用遮罩訓練的語言模型仍可能有很低的訓練損失？

   提示

   改變的最後詞元能被較早位置讀取。低損失可能獎勵生成時根本不可取得的資訊。

2. 在玩具句子程式碼中，於代名詞前加入第二個裝置名詞（「valve」）。`it` 的權重會怎樣？除了這些手動特徵之外，模型還需要什麼才能選對先行詞？

   提示

   特徵相同時，兩個裝置得到相同分數。位置資訊、動詞意義與篇章脈絡必須進入查詢或鍵，才能打破平手。

3. 在因果解碼器中，為長度 3 與 5、填補到 5 的兩個目標序列批次設計注意力與損失遮罩。為每個序列寫出 5×5 的阻擋／允許表，並列出貢獻損失的目標位置。

   提示

   填補鍵不得貢獻上下文，填補目標不得貢獻損失。也要決定填補查詢列如何處理，避免全部被阻擋的 softmax 列。

4. 對 $d = 512$、$h = 8$、$d_{\text{ff}} = 2048$ 的模型，計算一個編碼器區塊與一個解碼器區塊的權重數（忽略偏差與層正規化）。編碼器區塊有多少比例在前饋網路？

   提示

   自注意力與交叉注意力各有 $4d^2$ 個權重；前饋網路有 $2d\,d_{\text{ff}}$ 個。解碼器比編碼器多一個注意力子層。

5. 對序列長度 $T = 128$、1,024 與 16,384，在 $d = 512$ 下比較 $T^2 d$（注意力分數）與 $4Td^2$（投影）。從哪個長度開始分數項占主導？

   提示

   兩項在 $T = 4d$ 時相等。低於這個長度時，投影的成本高於分數計算。

6. 說明 Luong 的「general」評分 $h_u^\top W_a \bar{h}_j$ 為何是縮放內積注意力的特例。查詢、鍵與值投影分別是什麼？

提示

把 W_a 吸收進鍵投影。值投影是單位矩陣，縮放因子為 1。

參考文獻

1. Masato Hagiwara, *Real-World Natural Language Processing: Practical Applications with Deep Learning*, Chapter 8, Manning, 2021. Publisher ·Chapter 8
2. I. Sutskever, O. Vinyals and Q. V. Le, “Sequence to Sequence Learning with Neural Networks,” NeurIPS, 2014. Paper
3. K. Cho, B. van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk and Y. Bengio, “Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation,” EMNLP, 2014. Paper
4. D. Bahdanau, K. Cho and Y. Bengio, “Neural Machine Translation by Jointly Learning to Align and Translate,” ICLR, 2015. Paper
5. M.-T. Luong, H. Pham and C. D. Manning, “Effective Approaches to Attention-based Neural Machine Translation,” EMNLP, 2015. Paper
6. A. Vaswani et al., “Attention Is All You Need,” NeurIPS, 2017. Paper
7. J. L. Ba, J. R. Kiros and G. E. Hinton, “Layer Normalization,” arXiv:1607.06450, 2016. Paper
8. R. Xiong et al., “On Layer Normalization in the Transformer Architecture,” ICML, 2020. Paper
9. P. Shaw, J. Uszkoreit and A. Vaswani, “Self-Attention with Relative Position Representations,” NAACL, 2018. Paper
10. J. Su, M. Ahmed, Y. Lu, S. Pan, W. Bo and Y. Liu, “RoFormer: Enhanced Transformer with Rotary Position Embedding,” *Neurocomputing* 568, 2024. DOI
11. O. Press, N. A. Smith and M. Lewis, “Train Short, Test Long: Attention with Linear Biases Enables Input Length Extrapolation,” ICLR, 2022. Paper
12. I. Beltagy, M. E. Peters and A. Cohan, “Longformer: The Long-Document Transformer,” arXiv:2004.05150, 2020. Paper
13. A. Katharopoulos, A. Vyas, N. Pappas and F. Fleuret, “Transformers are RNNs: Fast Autoregressive Transformers with Linear Attention,” ICML, PMLR 119, 2020. Paper
14. T. Dao, D. Y. Fu, S. Ermon, A. Rudra and C. Ré, “FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness,” NeurIPS, 2022. Paper
15. N. Shazeer, “Fast Transformer Decoding: One Write-Head is All You Need,” arXiv:1911.02150, 2019. Paper
16. J. Ainslie, J. Lee-Thorp, M. de Jong, Y. Zemlyanskiy, F. Lebrón and S. Sanghai, “GQA: Training Generalized Multi-Query Transformer Models from Multi-Head Checkpoints,” EMNLP, 2023. Paper
17. Y. Tay, M. Dehghani, D. Bahri and D. Metzler, “Efficient Transformers: A Survey,” *ACM Computing Surveys* 55(6), 2022. DOI
18. T. Lin, Y. Wang, X. Liu and X. Qiu, “A Survey of Transformers,” *AI Open* 3, 2022. DOI

19. S. Jain and B. C. Wallace, “Attention is not Explanation,” NAACL, 2019. Paper
20. S. Wiegreffe and Y. Pinter, “Attention is not not Explanation,” EMNLP-IJCNLP, 2019. Paper
21. PyTorch, `TransformerEncoderLayer` documentation. Documentation
22. PyTorch, `scaled_dot_product_attention` 文件，2.15 版，查閱日期 2026-11-05。文件
23. PyTorch, `MultiheadAttention` 文件，2.15 版，查閱日期 2026-11-05。文件

NLP 08・以 Transformer 建構序列到序列模型（機器翻譯、聊天機器人）

建立並訓練用於翻譯的編碼器—解碼器 Transformer，以貪婪與束搜尋解碼、用從零寫出的 BLEU 評分，再把同一套條件生成觀念帶到對話狀態、澄清與對話評估。

網站版本: https://blog.shyandram.dev/zh-tw/learn/nlp/seq2seq-transformers/

你將學到

- 機器翻譯如何從統計片語表走到神經編碼器—解碼器，以及 Transformer 改變了什麼。
- 條件序列學習：目標位移、教師強制（teacher forcing）、考慮填補的損失，以及為何生成時會出現曝光偏差（exposure bias）。
- 如何在 CPU 上訓練小型編碼器—解碼器 Transformer 翻譯器，並以貪婪搜尋與帶長度正規化的束搜尋（beam search）解碼。
- BLEU 如何運作（從零實作）、chrF 與 COMET 補上什麼，以及為何它們都不能證明意義正確。
- 同一個條件生成觀點如何套用到聊天機器人，以及為何任務對話還需要明確狀態、澄清與工具邊界。
- 如何評估完整對話，而不是單一回覆。

先看全貌

序列到序列（seq2seq）模型讀一個序列、寫出另一個序列：由英文句寫出法文句、由對話歷史寫出回覆、由文件寫出摘要。編碼器把輸入變成每個詞元一個向量；解碼器一次寫一個輸出詞元，透過因果自注意力讀自己的前綴，再透過交叉注意力讀輸入（NLP 07）。同一套機制服務翻譯與對話，但兩種任務的失敗方式不同。譯文可能很流暢卻反轉否定；聊天機器人可能回覆親切，卻查錯裝置。

> **用最簡單的話說**
>
> 想像會議上的口譯員。他先聽完整句話（編碼器），再一個詞一個詞說出譯文，每說一個詞之前，都回頭看一下正在翻譯的那段發言（交叉注意力）。好的口譯員在講者不清楚時，也會問「您是指 Pump A 還是 Pump B？」說得流暢不等於說得對；口譯員的評判標準，是聽者最後是否得到相同的意思。

本章以兩種互補閱讀為錨。Hagiwara 第 6 章說明用於翻譯與對話應用的序列到序列模型，第 8 章介紹注意力與 Transformer [1]。McRoy 第 9 章把翻譯視為保留意義的生成 [2]。本章範例都是原創的本機 CPU 實驗，不需下載模型或資料集就能展示機制。NLP 09 接著討論開放式生成、摘要與服務。

1. 從統計機器翻譯到神經機器翻譯

直觀說法。機器翻譯最初從對齊的句對學習詞與片語的對應，再結合另一個描述流暢目標文字的模型。神經翻譯則以一個端到端訓練的網路取代整條管線。

統計機器翻譯（SMT）把翻譯視為**雜訊通道**（noisy channel）。要把外語句 $\mathbf{f}$ 譯成英文 $\mathbf{e}$，選擇

$$\hat{\mathbf{e}} = \arg\max_{\mathbf{e}} \; p(\mathbf{e})\, p(\mathbf{f} \mid \mathbf{e}).$$

其中 $p(\mathbf{e})$ 是獎勵流暢英文的語言模型；$p(\mathbf{f} \mid \mathbf{e})$ 是獎勵忠實的翻譯模型；$\hat{\mathbf{e}}$ 是選出的譯文。Brown 等人的 IBM 模型從句子對齊的文字估計詞層級翻譯機率與對齊 [3]。Koehn、Och 與 Marcu 的片語式翻譯使用多詞片語對，比單詞更能掌握局部詞序調整與慣用語 [4]。這類系統結合許多分別訓練的元件與特徵。

神經網路先以元件身分加入：Cho 等人用 RNN 編碼器—解碼器在既有 SMT 系統中替片語對評分 [5]。接著 Sutskever、Vinyals 與 Le 訓練 LSTM 編碼器—解碼器直接翻譯 [6]。Bahdanau、Cho 與 Bengio 加入注意力，讓解碼器每一步都能搜尋來源 [7]；Vaswani 等人則完全以注意力取代循環 [8]。位元組對編碼（byte-pair encoding）等子詞切分讓開放詞彙變得實際可行，罕見詞與名稱可以譯成已知片段的序列 [9]；NLP 02 逐步建立 BPE 詞彙表。Stahlberg 的回顧詳細追溯了從計數式 SMT 到神經機器翻譯的轉變 [22]。

兩個近期錨點顯示領域的走向。NLLB 計畫建立了涵蓋 200 種語言的單一大規模多語模型，特別關注低資源語言 [21]。WMT24 一般翻譯任務在專業人工評估下，比較參賽系統與數個大型語言模型及線上服務；其標題概括了結論：LLM 時代已經到來，但機器翻譯尚未解決 [20]。

2. 條件序列學習

直觀說法。翻譯器一次一個詞元，建模「給定來源時目標句」的機率。訓練時給它正確的先前目標詞元；生成時則要它使用自己的輸出。

條件序列模型寫成：

$$p_\theta(\mathbf{y} \mid \mathbf{x}) = \prod_{t=1}^{U} p_\theta(y_t \mid y_{<t}, \mathbf{x}).$$

其中 $\mathbf{x}$ 是來源序列；$\mathbf{y} = (y_1, \dots, y_U)$ 是含結束詞元的目標序列；U 是其長度；$y_{<t}$ 是位置 t 之前的詞元；θ 是學得參數。來源可以完整可見，目標仍保持因果。分解描述分布，不保證取樣輸出獲得來源支持。

訓練通常提供正確的先前目標詞元，稱為教師強制 [6]。批次的詞元損失為：

$$\mathcal{L}(\theta) = -\frac{1}{M} \sum_{b=1}^{B} \sum_{t=1}^{T_b} m_{bt} \log p_\theta(y_{bt} \mid y_{b,<t}, \mathbf{x}_b).$$

其中 B 是例數；T_b 是範例 b 填補後的目標長度；m_{bt} 對真實目標為一、填補為零；$M = \sum_{b,t} m_{bt}$ 是真實詞元數；$\mathbf{x}_b$、y_{bt}、$y_{b,<t}$ 分別是來源、目前目標與參考前綴。使用自然對數。遮罩從損失移除填補，不會自動從計算移除。

仔細檢查位移。解碼器輸入從起始詞元開始，目標從第一個輸出詞元開始；目標含結束符號讓模型學停止。填補與結束必須分開，否則可能獎勵錯誤停止。不同長度句子的批次要記錄按詞元或例子平均，因為它們給長句不同權重。

資料建構往往比小架構調整更重要。翻譯對若默默修正數字，就訓練模型改數字。來源版本、指南與目標來源要一起保留，按文件、對話或事件分組切分，防止近重複跨訓練測試。

教師強制與曝光偏差

教師強制提供正確前綴，即使模型自己會產生另一個前綴。假設參考從 `Pump A` 開始，生成卻選了 `Pump B`。後續詞元仍可能構成完全流暢的 Pump B 陳述。參考前綴損失卻始終在 `Pump A` 後評估那些位置，沒有測試如何從這次生成錯誤恢復。差別在於：給定歷史下的預測，以及模型自己持續建立歷史時的行為。

Ranzato 等人把這種訓練／推論不一致稱為**曝光偏差**：模型訓練時只接觸正確前綴，測試時卻必須讀自己的預測，錯誤可能累積 [10]。兩種回應說明了設計空間。**排程取樣**（scheduled sampling）在訓練中逐步把部分正確解碼器輸入換成模型自己的預測 [11]。**序列層級訓練**直接最佳化整個生成序列的分數（例如 BLEU），而不只是下一詞元概似 [10]。兩者都不能免除評估完整生成輸出的需要；以教師強制做最大概似仍是標準起點，分解本身也是精確的。ML 09 討論影像生成中的同一種不一致。

詞元平均也隱藏不同語意後果。替換 `the` 可能只影響風格；替換 `not`、裝置 ID 或單位，卻能反轉結果。每個詞元貢獻一項對數損失，應用對這些變化的成本可能差很多。如果常見用詞改善，稀少關鍵差異卻惡化，平均損失下降仍可能伴隨更多重要錯誤。多個正確說法也是問題：單一參考提供一種可接受表達，不是完整集合。

僅線上版：*Bahdanau attention — Jointly Learning to Align and Translate — https://blog.shyandram.dev/zh-tw/learn/nlp/seq2seq-transformers/papers/bahdanau-attention–jointly-learning-to-align-and-translate/*

3. 用於翻譯的編碼器—解碼器 Transformer

直觀說法。編碼器以雙向自注意力讀完整個來源；解碼器以因果自注意力寫目標，並以交叉注意力讀編碼器。最後的線性層把每個解碼器狀態轉成目標詞元上的分布。

原始 Transformer [8] 由 NLP 07 的區塊組成：

1. **輸入**。來源與目標文字切成子詞 [9]（NLP 02）。每個詞元 ID 經嵌入、乘以 $\sqrt{d}$，再加上正弦位置編碼。
2. **編碼器**。N 個由自注意力與前饋子層組成的區塊，對來源鍵套用填補遮罩，產生**記憶** $M \in \mathbb{R}^{T_x \times d}$，每個來源詞元一個向量。
3. **解碼器**。N 個區塊，包含遮罩自注意力（因果加目標填補）、讀取 M 的交叉注意力（遮蔽來源填補）與前饋子層。
4. **輸出**。線性層與 softmax 在每個目標位置給出 $p_\theta(y_t \mid y_{<t}, \mathbf{x})$。

訓練時，所有目標位置一次算完，因為因果遮罩讓每個位置只看到自己的前綴。生成時，編碼器

只跑一次，解碼器每產生一個輸出詞元跑一次。實務解碼器會快取較早目標位置的鍵與值，而不是重算；NLP 09 分析這個快取。

原論文有兩個值得知道的訓練細節。模型使用**標籤平滑**（label smoothing）：目標分布把大部分機率放在參考詞元，把一小部分分給詞彙表其餘詞元 [8, 29]。平滑參數為 ε、詞彙表為 V 時，訓練目標為 $q(v) = (1-\varepsilon)\mathbf{1}\{v = y_t\} + \varepsilon/|V|$；其中 $\mathbf{1}\{\cdot\}$ 在條件成立時為一。這抑制過度自信；Vaswani 等人回報它使困惑度變差，但改善了 BLEU [8]。模型也使用學習率暖身，Xiong 等人後來把它與後正規化配置連結起來（NLP 07）。

4. 解碼：貪婪搜尋、束搜尋與長度正規化

直觀說法。訓練好的模型對每個可能的下一詞元給出機率。解碼器必須把這些機率變成一個輸出句。貪婪解碼每次取最好的下一詞元；束搜尋則同時保留幾個部分句子。

候選序列分數為：

$$\ell(\mathbf{y} \mid \mathbf{x}) = \sum_{t=1}^{U} \log p_\theta(y_t \mid y_{<t}, \mathbf{x}).$$

其中 ℓ 是完整序列對數機率；U 包含結束詞元；其他符號同前。對所有序列精確最大化並不可行，因為候選數隨長度指數成長。**貪婪解碼**每一步選機率最高的詞元。寬度為 k 的**束搜尋**保留分數最高的 k 個前綴，把每個前綴延伸所有詞元，再保留最好的 k 個；完成的假說（以結束詞元結尾）另外存放。這些選擇控制搜尋與變化，不會讓無支持內容變可靠。

貪婪解碼可能錯過最佳完整序列：機率稍低的第一個詞元，可能接著一個機率高得多的延續，而束搜尋能把它保留到發現為止。NLP 09 用人工範例說明這點，並討論原始對數機率對未完成前綴提供的上界、搜尋錯誤與模型錯誤的差別，以及基於取樣的解碼。這裡聚焦翻譯多出來的部分：長度正規化，以及在訓練好的翻譯器上實際量測的貪婪與束搜尋比較（第 6 節）。

長度正規化

原始對數機率有內建偏好。每多一個詞元就多一個非正項，所以較長的假說即使是較好的譯文，分數也傾向較低。Wu 等人在 Google 神經機器翻譯系統中描述了這個偏差，並以正規化分數排序完成的假說 [12]：

$$s(\mathbf{y}, \mathbf{x}) = \frac{\ell(\mathbf{y} \mid \mathbf{x})}{\mathrm{lp}(\mathbf{y})}, \qquad \mathrm{lp}(\mathbf{y}) = \frac{(5 + |\mathbf{y}|)^{\alpha}}{(5+1)^{\alpha}}.$$

其中 $|\mathbf{y}|$ 是假說的詞元長度；$\alpha \geq 0$ 控制正規化強度，$\alpha = 0$ 回到原始分數，$\alpha = 1$ 近似每詞元平均；s 是用來排序完成假說的分數。他們的完整評分函數還加上以注意力為基礎的涵蓋懲罰，這裡省略。他們在開發資料上調整 α [12]；應把它當成超參數，而不是常數。

```
candidates = {                       # per-token log-probabilities, end token included
    "stop <eos>": [-0.9, -1.1],
    "stop pump b now <eos>": [-0.9, -0.5, -0.3, -0.4, -0.2],
}
```

```
def length_penalty(n, alpha):        # Wu et al. (2016): ((5 + n) / 6) ** alpha
    return ((5 + n) / 6) ** alpha
for alpha in [0.0, 0.6, 1.0]:
    scores = {c: sum(lp) / length_penalty(len(lp), alpha) for c, lp in candidates.items()}
    best = max(scores, key=scores.get)
    print(f"alpha={alpha}:", {c: round(s, 3) for c, s in scores.items()}, "->", best)
```

輸出：

```
alpha=0.0: {'stop <eos>': -2.0, 'stop pump b now <eos>': -2.3} -> stop <eos>
alpha=0.6: {'stop <eos>': -1.823, 'stop pump b now <eos>': -1.693} -> stop pump b now <eos>
alpha=1.0: {'stop <eos>': -1.714, 'stop pump b now <eos>': -1.38} -> stop pump b now <eos>
```

原始分數偏好被截斷的指令；$\alpha = 0.6$ 與 $\alpha = 1$ 偏好完整指令（若 α 很小，例如 0.2，仍會偏好短的那個）。哪個正確取決於來源，而分數看不到來源。

正規化也改變搜尋本身：使用原始分數時，若某個完成假說勝過所有未完成前綴，就可以安全回傳，因為延伸前綴只會降低原始分數；改用長度正規化分數後，這個論證不再成立，停止規則必須把懲罰考慮進去。

要明訂輸出限制、停止規則及達上限後的行為，截斷翻譯不能當完整結果。解碼參數與模型一起記錄，讓評估可重現。

5. 評估譯文：BLEU、chrF 與 COMET

直觀說法。自動指標把系統輸出與一或多個人工參考譯文比較。它們便宜、可重現，適合比較系統；但一個小詞改動就能反轉意思，分數卻幾乎不變。

BLEU

BLEU 比較候選與參考的 n-gram [13]。對每個階數 $n = 1, \dots, 4$，它計算**修正（截限）精確率**：每個候選 n-gram 最多只計入它在「出現最多次的那個參考」中的次數。在整個測試語料上：

$$p_n = \frac{\sum_{\text{sent}} \sum_{g \in G_n} \min\big(c(g),\, c_{\text{ref}}^{\max}(g)\big)}{\sum_{\text{sent}} \sum_{g \in G_n} c(g)}, \qquad \text{BP} = \begin{cases} 1 & c > r \\ e^{1 - r/c} & c \le r \end{cases}, \qquad \text{BLEU} = \text{BP} \cdot \exp\Big(\sum_{n=1}^{4} w_n \log p_n \Big).$$

其中 G_n 是候選句中相異 n-gram 的集合；$c(g)$ 是 n-gram g 在候選中的次數；$c_{\text{ref}}^{\max}(g)$ 是它在該句任一參考中的最大次數；c 是整個語料的候選總長度；r 是**有效參考長度**，即每句取最接近候選長度的參考長度後加總；BP 是**短度懲罰**（brevity penalty），防止系統靠寫極短輸出提高精確率；$w_n = 1/4$ 是均勻權重。BLEU 是語料層級分數：精確率先在所有句子上合併，再取對數，所以它不是句子分數的平均。

```
import math
from collections import Counter
```

```
def ngrams(tokens, n):
    return Counter(tuple(tokens[i:i + n]) for i in range(len(tokens) - n + 1))

def corpus_bleu(hypotheses, references, max_n=4):
    """hypotheses: list of token lists; references: list of lists of token lists."""
    matched, total = [0] * max_n, [0] * max_n
    hyp_len = ref_len = 0
    for hyp, refs in zip(hypotheses, references):
        hyp_len += len(hyp)
        # effective reference length: the reference length closest to the hypothesis
        ref_len += min((abs(len(r) - len(hyp)), len(r)) for r in refs)[1]
        for n in range(1, max_n + 1):
            counts = ngrams(hyp, n)
            max_ref = Counter()
            for r in refs:
                max_ref |= ngrams(r, n)            # elementwise max over references
            matched[n - 1] += sum((counts & max_ref).values())   # clipped counts
            total[n - 1] += max(len(hyp) - n + 1, 0)
    if min(matched) == 0:
        return 0.0, matched, total
    log_p = sum(math.log(m / t) for m, t in zip(matched, total)) / max_n
    bp = 1.0 if hyp_len > ref_len else math.exp(1 - ref_len / hyp_len)
    return bp * math.exp(log_p), matched, total

refs = [["the cat is on the mat".split(), "there is a cat on the mat".split()]]
_, m, t = corpus_bleu(["the the the the the the the".split()], refs, max_n=1)
print("clipped unigram precision:", f"{m[0]}/{t[0]}")
ref = "pump a is not operating at 30 c".split()
for hyp in ["pump a is not operating at 30 c",
            "pump a is operating at 30 c",
            "pump a is not running at 30 c"]:
    score, _, _ = corpus_bleu([hyp.split()], [[ref]])
    print(f"{score:.3f}  {hyp}")
assert m[0] == 2 and t[0] == 7
assert corpus_bleu([ref], [[ref]])[0] == 1.0
```

輸出：

```
clipped unigram precision: 2/7
1.000  pump a is not operating at 30 c
0.515  pump a is operating at 30 c
0.500  pump a is not running at 30 c
```

第一項檢查重現經典的截限例子：重複七次「the」的候選只得到兩次的分數，因為沒有任何參考包含超過兩個「the」[13]。第二項檢查是警訊。去掉 `not` 會反轉意思，分數卻高於把 `operating` 換成 `running` 的正確改述。BLEU 量測的是與參考的表面重疊，它不知道哪些詞承載命題。

這個實作使用空白分詞、不做平滑（任一 n-gram 精確率為零就得零分），並採用最接近參考長度規則。發表的 BLEU 分數取決於這些選擇。Post 指出，光是分詞與正規化的差異就能大幅改變回報的 BLEU，並提出 sacreBLEU：它在去分詞後的文字上以標準內部分詞器計算分數，並印出版本簽章 [14]。回報任何 BLEU 數字時都應附上簽章；不同設定算出的分數不能比較。

chrF

chrF 以**字元** n-gram 取代詞 n-gram 評分 [15]：

$$\mathrm{chrF}_\beta = (1+\beta^2)\,\frac{\mathrm{chrP}\cdot\mathrm{chrR}}{\beta^2\,\mathrm{chrP}+\mathrm{chrR}}.$$

其中 chrP 與 chrR 是對各 n-gram 階數平均的字元 n-gram 精確率與召回率；β 讓召回率的權重是精確率的 β 倍。Popović 在其實驗中發現 6-gram 效果最好，並比較了 $\beta=1$ 與 $\beta=3$ [15]。字元 n-gram 對共享詞幹與屈折變化給部分分數，對形態豐富的目標語言有幫助；但 chrF 仍是重疊分數，分辨不出 `not` 比 `the` 重要。

學得指標：COMET

COMET 使用跨語言預訓練編碼器，訓練神經模型從來源、候選與參考預測人類品質判斷 [16]。它能獎勵與參考共享詞很少的正確改述。在以專家 MQM（多維品質指標，Multidimensional Quality Metrics）標註為準的 WMT22 指標任務中，神經學得指標與人類評分的相關性優於 BLEU、chrF 等重疊指標 [18]。學得指標帶有自己的模型版本、訓練分布與語言涵蓋，輸出只是在那些條件下的量測。應在應用重要的否定、數量與術語錯誤上檢查指標。

能排序系統的指標，不會自動成為輸出門檻

WMT25 評估結果分開片段品質預測、錯誤區段標註與品質導向修正，指出系統層級與片段層級評估可能偏好不同方法，語言多樣性仍然困難 [19]。這很重要：拒絕一個錯誤譯文，與比較模型平均品質，是不同決策。指標可能獎勵整體流暢系統，卻漏掉個別輸出的少見否定反轉。接受門檻需要在這些錯誤上驗證，不能直接由總體相關性結果繼承。

可交叉檢查生成器錯誤類別與評估器取得的輸入。只看參考的比較能檢查用詞一致，卻缺少驗證無支持陳述所需的來源；取得來源的評估器雖有證據，仍可能與生成器共享作用範圍或數量錯誤。應檢查分歧案例：正確改述被評差、流暢矛盾被評好，以及準確卻不完整的譯文。人工覆核保留原始來源，並分開充分性與風格。這能說明哪些評估訊號適合比較模型，哪些可用來判斷某次結果是否應接受。

重疊、召回與人工判斷

同一個反例也影響 ROUGE 用於摘要的偏召回重疊 [17]。簡單 unigram 召回檢查：

$$R_1 = \frac{\sum_{w} \min(c_{\mathrm{ref}}(w), c_{\mathrm{hyp}}(w))}{\sum_{w} c_{\mathrm{ref}}(w)}.$$

其中 w 遍歷詞型；$c_{\mathrm{ref}}(w)$、$c_{\mathrm{hyp}}(w)$ 是參考與候選詞頻；分母是參考詞元數。以下最小實作沒有重現完整套件的每項前處理或聚合。

```
import re
from collections import Counter
```

```
def words(text):
    return re.findall(r"[a-z0-9]+", text.lower())
def overlap_recall(reference, hypothesis):
    ref, hyp = Counter(words(reference)), Counter(words(hypothesis))
    return sum((ref & hyp).values()) / sum(ref.values())
reference = "Pump A is operating at 30 C."
contradiction = "Pump A is not operating at 30 C."
score = overlap_recall(reference, contradiction)
print("Unigram recall:", score)
assert score == 1.0
assert ("not" in words(reference)) != ("not" in words(contradiction))
```

矛盾句保留所有參考詞，因此召回 1.0。精確率會懲罰多出的詞，但小懲罰仍不能表達否定反轉的重要性。這是受控反例，不表示重疊指標永不與人類判斷相關，而是說明必須配合對意義敏感的檢查。

人工評估要隱藏系統身分、隨機呈現、保留來源並指定決策。Freitag 等人的大型 MQM 研究由能看到完整文件的專業譯者評分，得到的系統排名與同一共享任務中群眾評分者的排名有明顯差異 [30]。應請覆核者先標出具體錯誤，再判斷整體品質；回報誰覆核、涵蓋哪些語言，並把裁決過的挑戰集合與一般樣本分開。

6. 動手做：微型 Transformer 翻譯器

直觀說法。我們發明一個小型語言對，在 CPU 上用 3,000 個句對訓練編碼器—解碼器 Transformer 不到一分鐘，再用上面的 BLEU 函數比較貪婪與束搜尋解碼。

來源語言類似英文：主詞、動詞、受詞，名詞前可有形容詞，也可有 `never`。目標語言是發明的：每個詞有固定譯詞，形容詞放在名詞之後，受詞後接助詞 `o`，動詞放最後，否定變成句尾詞 `nai`。例如 `the small drone never lifts a red cable` 譯為 `la drono pico un kablo rosso o levi nai`。因此翻譯同時需要字典與詞序調整，正是逐詞翻譯不足的兩個原因。訓練句與測試句不重疊，但來自同一個小文法，所以這是分布內測試。

模型使用 PyTorch 的編碼器與解碼器層（各兩層、寬度 64、四個頭）、縮放嵌入加正弦位置、布林遮罩（`True` 表示阻擋），並以 `ignore_index` 從損失中去掉填補目標，實作第 2 節的遮罩損失。請在 BLEU 程式區塊之後執行，因為它會呼叫 `corpus_bleu`。

```
# Run after the BLEU block above: this code calls its corpus_bleu function.
import math, random, time
import torch
from torch import nn
from torch.nn import functional as F

torch.manual_seed(0); torch.set_num_threads(1); rng = random.Random(0)
lexicon = {"the": "la", "a": "un", "robot": "robo", "sensor": "senso",
           "pump": "pompa", "valve": "valva", "drone": "drono", "cable": "kablo",
           "red": "rosso", "blue": "blu", "big": "grando", "small": "pico",
           "sees": "vidi", "checks": "kontrolli", "moves": "movi",
           "opens": "aperi", "lifts": "levi"}
```

```
DET, NOUN = ["the", "a"], ["robot", "sensor", "pump", "valve", "drone", "cable"]
ADJ, VERB = ["red", "blue", "big", "small"], ["sees", "checks", "moves", "opens", "lifts"]

def noun_phrase():
    adj = [rng.choice(ADJ)] if rng.random() < 0.5 else []
    return rng.choice(DET), adj, rng.choice(NOUN)

def pair():          # source: SVO, adjective before noun; target: SOV, adjective after noun
    (d1, a1, n1), (d2, a2, n2) = noun_phrase(), noun_phrase()
    verb, neg = rng.choice(VERB), rng.random() < 0.3
    src = [d1, *a1, n1] + (["never"] if neg else []) + [verb, d2, *a2, n2]
    np_ = lambda d, a, n: [lexicon[d], lexicon[n]] + [lexicon[x] for x in a]
    tgt = np_(d1, a1, n1) + np_(d2, a2, n2) + ["o", lexicon[verb]] + (["nai"] if neg else [])
    return " ".join(src), " ".join(tgt)

pairs = list(dict(pair() for _ in range(6000)).items())   # unique source sentences
train_pairs, test_pairs = pairs[:3000], pairs[3000:3200]
PAD, BOS, EOS = 0, 1, 2
src_vocab = ["<pad>", "<bos>", "<eos>"] + sorted({w for s, _ in pairs for w in s.split()})
tgt_vocab = ["<pad>", "<bos>", "<eos>"] + sorted({w for _, t in pairs for w in t.split()})
s_id = {w: i for i, w in enumerate(src_vocab)}
t_id = {w: i for i, w in enumerate(tgt_vocab)}

def batch(texts, ids, L):
    rows = [[ids[w] for w in t.split()] + [EOS] for t in texts]
    return torch.tensor([r + [PAD] * (L - len(r)) for r in rows])
S = batch([s for s, _ in train_pairs], s_id, 10)
Y = batch([t for _, t in train_pairs], t_id, 12)
Y_in = torch.cat([torch.full((len(Y), 1), BOS), Y[:, :-1]], dim=1)   # shift right

def sinusoidal(n, d):
    pos = torch.arange(n)[:, None]
    omega = 1.0 / 10000 ** (torch.arange(0, d, 2) / d)
    pe = torch.zeros(n, d)
    pe[:, 0::2], pe[:, 1::2] = torch.sin(pos * omega), torch.cos(pos * omega)
    return pe

class TinyMT(nn.Module):
    def __init__(self, d=64):
        super().__init__()
        self.d = d
        self.src_emb = nn.Embedding(len(src_vocab), d)
        self.tgt_emb = nn.Embedding(len(tgt_vocab), d)
        self.register_buffer("pe", sinusoidal(32, d))
        enc = nn.TransformerEncoderLayer(d, 4, 128, dropout=0.0, batch_first=True)
        dec = nn.TransformerDecoderLayer(d, 4, 128, dropout=0.0, batch_first=True)
        self.encoder = nn.TransformerEncoder(enc, 2, enable_nested_tensor=False)
        self.decoder = nn.TransformerDecoder(dec, 2)
        self.out = nn.Linear(d, len(tgt_vocab))
    def embed(self, emb, x):
        return emb(x) * math.sqrt(self.d) + self.pe[: x.shape[1]]
    def encode(self, src):
        return self.encoder(self.embed(self.src_emb, src), src_key_padding_mask=src.eq(PAD))
    def decode(self, memory, src, y_in):
        T = y_in.shape[1]
        causal = torch.ones(T, T, dtype=torch.bool).triu(1)      # True = blocked
```

```
        h = self.decoder(self.embed(self.tgt_emb, y_in), memory, tgt_mask=causal,
                         tgt_key_padding_mask=y_in.eq(PAD),
                         memory_key_padding_mask=src.eq(PAD))
        return self.out(h)

@torch.no_grad()
def greedy(src_text, max_len=14):
    src = batch([src_text], s_id, len(src_text.split()) + 1)
    memory, y = model.encode(src), [BOS]
    for _ in range(max_len):
        logp = model.decode(memory, src, torch.tensor([y]))[0, -1].log_softmax(-1)
        y.append(int(logp.argmax()))
        if y[-1] == EOS:
            break
    return [tgt_vocab[i] for i in y[1:] if i != EOS]

@torch.no_grad()
def beam_search(src_text, width=4, alpha=0.6, max_len=14):
    src = batch([src_text], s_id, len(src_text.split()) + 1)
    memory = model.encode(src)
    beams, finished = [([BOS], 0.0)], []
    lp = lambda n: ((5 + n) / 6) ** alpha             # GNMT length penalty
    for _ in range(max_len):
        prefixes = torch.tensor([y for y, _ in beams])
        logp = model.decode(memory.expand(len(beams), -1, -1), src.expand(len(beams), -1),
                            prefixes)[:, -1].log_softmax(-1)
        cand = [(y + [t], s + logp[b, t].item()) for b, (y, s) in enumerate(beams)
                for t in logp[b].topk(width).indices.tolist()]
        cand.sort(key=lambda c: c[1], reverse=True)
        beams = []
        for y, s in cand:
            (finished if y[-1] == EOS else beams).append((y, s))
            if len(beams) == width:
                break
        if not beams:
            break
    best, _ = max(finished or beams, key=lambda c: c[1] / lp(len(c[0]) - 1))
    return [tgt_vocab[i] for i in best[1:] if i != EOS]

@torch.no_grad()
def log_prob(src_text, words):                        # model score of a full output
    src = batch([src_text], s_id, len(src_text.split()) + 1)
    y = [t_id[w] for w in words] + [EOS]
    logp = model.decode(model.encode(src), src, torch.tensor([[BOS] +
     ↪  y[:-1]])).log_softmax(-1)
    return logp[0, torch.arange(len(y)), y].sum().item()

def evaluate():
    model.eval()
    refs, outputs = [[t.split()] for _, t in test_pairs], {}
    for name, decode in [("greedy", greedy), ("beam-4", beam_search)]:
        hyps = [decode(s) for s, _ in test_pairs]
        exact = sum(h == r[0] for h, r in zip(hyps, refs)) / len(refs)
        mean_lp = sum(log_prob(s, h) for (s, _), h in zip(test_pairs, hyps)) / len(hyps)
        print(f"  {name}: BLEU {corpus_bleu(hyps, refs)[0]:.3f}  exact {exact:.3f}"
              f"  mean log p {mean_lp:.3f}")
```

```
        outputs[name] = hyps
    print("  outputs that differ:", sum(g != b for g, b in zip(*outputs.values())))
    model.train()

model = TinyMT()
opt = torch.optim.Adam(model.parameters(), lr=1e-3)
start = time.time()
for step in range(1, 401):
    idx = torch.randint(0, len(S), (64,))
    logits = model.decode(model.encode(S[idx]), S[idx], Y_in[idx])   # teacher forcing
    loss = F.cross_entropy(logits.reshape(-1, len(tgt_vocab)), Y[idx].reshape(-1),
                           ignore_index=PAD)
    opt.zero_grad(); loss.backward(); opt.step()
    if step in (100, 400):
        print(f"after {step} steps: training loss {loss.item():.3f}")
        evaluate()
model.eval()
print(" ".join(greedy("the small drone never lifts a red cable")))
print(f"total time {time.time() - start:.0f}s")
```

輸出（CPU、單執行緒；時間依機器而異，其他 PyTorch 版本或硬體可能改變最後幾位數）：

```
after 100 steps: training loss 0.221
  greedy: BLEU 0.614  exact 0.365  mean log p -1.587
  beam-4: BLEU 0.618  exact 0.375  mean log p -1.553
  outputs that differ: 31
after 400 steps: training loss 0.003
  greedy: BLEU 1.000  exact 1.000  mean log p -0.027
  beam-4: BLEU 1.000  exact 1.000  mean log p -0.027
  outputs that differ: 0
la drono pico un kablo rosso o levi nai
total time 16s
```

分開解讀兩個檢查點。訓練 100 步後模型只訓練了一半：200 個測試譯文中只有約三分之一完全正確。此時束搜尋改變了 200 個輸出中的 31 個，找到平均模型對數機率較高的輸出，BLEU 與完全匹配率也略為提升。400 步後，模型在這個分布內測試上既有信心又正確，貪婪與束搜尋輸出完全相同：模型分布非常尖銳時，更寬的搜尋找不到別的東西。束搜尋藉由減少**搜尋錯誤**而有幫助；它修不好**模型錯誤**，在簡單輸入上也可能毫無改變。

這裡 BLEU 與完全匹配同步變化，是因為語言很小又確定：每個來源恰有一個正確譯文。使用真實參考時，正確譯文可能與參考不同，兩個數字就會分歧。測試也只檢查同一文法的句子，完全沒說明組合泛化，例如訓練中從未見過的形容詞—名詞組合；練習 1 請你測試這點。

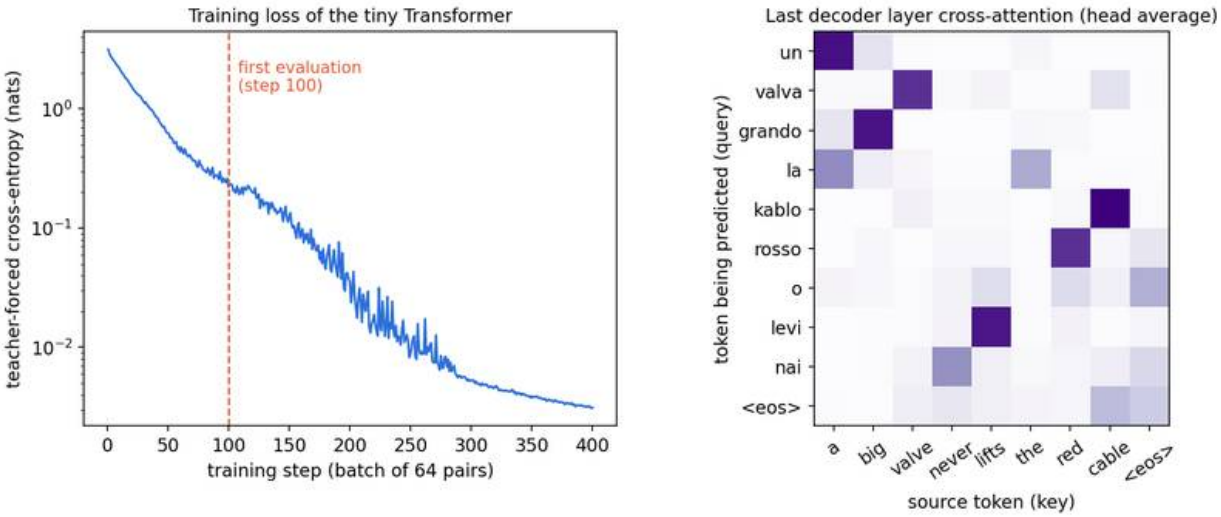


圖 *1* — 微型翻譯器。左：上方程式碼的教師強制訓練損失。右：對一個翻譯正確的保留句，最後一層解碼器交叉注意力（各頭平均）。每一列是預測該標示詞的解碼器位置。偏離對角線的圖樣就是模型學到的詞序調整；它顯示計算從哪裡讀取，不證明為何正確。

圖 1 的交叉注意力圖，是 Bahdanau 等人為 RNN 翻譯器視覺化的軟對齊的 Transformer 版本 [7]。目標中的形容詞讀取來源中位於名詞之前的形容詞；移到句尾的動詞讀取來源動詞；`nai` 讀取 `never`。這個對齊只從翻譯損失中浮現，沒有提供任何對齊標籤。

7. 翻譯跨形式保留意義

直觀說法。好的譯文會改變用詞與詞序，但保留誰對誰做了什麼、何時、多少，以及事情是否真的發生。

翻譯是具強內容限制的條件生成：保留來源意義，同時產生自然目標語言。詞序、形態、慣用語與隱含上下文，使它不只是逐詞替換。McRoy 翻譯討論 [2] 可搭配 Hagiwara 序列到序列材料 [1]。

應分開評估**充分性（adequacy）**——來源意義是否保留——與**流暢度（fluency）**——目標是否自然。流暢譯文可能漏例外、反轉否定或改行動者；字面翻譯則可能保技術內容卻不自然。可接受平衡取決於用途，必須在選模型或指標前指定。

通用翻譯器可用詞彙表保術語，仍要在上下文測試。必需詞可能存在，卻附錯實體；保留數字也不夠，因為值可能重新分配。可測「Pump A 比 Pump B 熱」、「閥門可能保持關閉」、「中午前未偵測到故障」等關係。最小改動揭露比較、模態、否定與時間是否被尊重。

語言標籤與正規化屬於模型契約。繁體與簡體是不同輸出要求；廣泛語言標籤不指定書寫系統、地域術語與標點。存來源前不要破壞性轉換，應並存原文與正規化輸入，讓覆核者追查誤譯。

低資源情況的不確定性可能來自訓練涵蓋有限，而非輸入含糊。報告應記錄領域、方言、文字系統與語言對。短產品描述驗證不代表長技術指令也成立。覆核者不能讀兩語就無法直接評充分性，要使用合格雙語覆核並說明缺口。

保留命題，不只保留詞

翻譯時，把來源與目標當成具有參與者、關係與作用範圍的命題比較。「Pump A is warmer than Pump B」必須保留比較方向。譯文即使含兩個名稱與「較熱」，仍可能反轉方向。「No sensor exceeded the limit」也不同於「Not every sensor exceeded the limit」：後者允許部分超標。NLP 12 說明這兩種否定範圍為何不同。

有時歧義原本就在來源。`May` 可依脈絡表示可能或許可。如果孤立句子無法決定，把它譯成明確表示許可的說法，就新增了解讀。有周邊文件或對話上下文時應利用；否則保留或標明歧義。目標流暢，不足以合理化任意決定未充分指定的來源。

以意義為中心的比較，可逐項檢查實體綁定、事件狀態、數量、單位、否定與條件，再評整體可讀性。這允許詞彙變化，同時揭露命題改變，也能區分真正誤譯與來源已有的歧義：因缺來源脈絡造成的標註分歧，不能當成某個詞彙選擇就能修好模型的證據。

8. 聊天機器人：把對話當成序列到序列

直觀說法。聊天機器人可以像翻譯器一樣建構：「來源」是目前為止的對話，「目標」是下一個回覆。這適合流暢閒聊，但任務助理還需要追蹤已經決定了什麼。

Vinyals 與 Le 在 IT 服務台資料集與電影字幕上訓練序列到序列模型，從先前回合預測下一回合；他們發現，即使最佳化的概似目標並不直接量測對話品質，模型仍能進行簡單對話 [23]。Li 等人觀察到這類模型傾向產生「I don't know」之類安全、通用、與輸入無關的回覆，並提出偏好針對訊息內容回覆的最大互資訊目標 [24]。這個教訓延續到現代助理：高機率回覆不一定有資訊量，也不一定正確。

回覆生成可寫成 $p_\theta(u_t \mid h_t, s_t, e_t)$，其中 u_t 是下一個系統話語、h_t 是選定歷史、s_t 是任務狀態、e_t 是證據、θ 是模型參數。條件分布說明生成器收到什麼，不保證它遵守每個欄位或證據。

任務導向對話有明確結果：形成查詢、完成類似預約的紀錄、找正確觀察或產生指定報告。開放交談的目標較不明確，可接受回覆也更多。兩者都需要一致性，成功標準卻不同。模組化任務系統分開理解、狀態追蹤、政策、證據存取與文字實現，讓錯誤容易定位。端到端系統可整合這些功能，但仍要測出錯狀態與措辭問題的差別。Deriu 等人 [25] 與 Ni 等人 [26] 的綜述描述更廣的設計與評估家族。

任務政策可選澄清、查詢、解釋或拒絕不支援操作，再由生成器說出這個動作。這讓對話流暢度不決定查詢是否準備好。開放交談受益於彈性語言，但角色風格不能覆蓋事實一致性。系統可能很吸引人，卻編造過去互動，或暗示未發生的動作已完成。對話主張要對實際紀錄檢查，外部主張則對證據檢查。

9. 回合、對話行為與狀態

直觀說法。對話是一個持續變化的共享問題。最新一句可能取代先前請求、回指先前事物，或修正系統。系統必須追蹤哪些被提議、哪些真正被接受。

對話回合是參與者在交談某時點的貢獻。它的功能，也就是**對話行為**（dialogue act），可以是請求、回答、修正、確認或澄清。同一表面詞依待回答問題而有不同功能：「週二」可能提供日期，

也可能修正原日期。

任務導向對話狀態儲存繼續任務所需資訊。設備查詢助理的欄位可包含裝置、時間區間、測量項目與輸出語言。值應連同來源回合、信心與狀態儲存。未知、明確不存在與不適用，不能全部壓成同一個空字串。狀態更新可寫成：

$$s_t = U(s_{t-1}, o_t, r_t), \qquad a_t = \pi(s_t).$$

其中 s_t 是回合 t 後的狀態；o_t 是觀察話語或擷取對話行為；r_t 指出說話者與其他回合脈絡；U 是更新規則；a_t 是下一系統動作；π 是狀態到動作的政策。狀態不一定是單一確定紀錄，政策也不一定直接生成文字。

提案與已接受狀態應分開。使用者可以要求選項、檢視再改變。系統建議出現在逐字稿，不代表成為使用者偏好。修正需要範圍：「其實要 Pump B」可只換裝置而保留時間；「重新開始」可能清除整個任務；「不是昨天」否定值卻未必提供替代。

含修正測試的確定性狀態追蹤器

對狹義任務，結構化追蹤器是有用基準，揭露語言理解與動作選擇的契約。範例消費的是已擷取事件，不主張能理解任意自然語言。狀態儲存選定欄位與修訂編號。任何接受的欄位變更，都讓先前確認失效；確認只對明確展示給使用者的版本有效，避免舊的「是」授權新變更提案。

```
from copy import deepcopy

initial = {"device": None, "period": None, "revision": 0, "confirmed": None}
def update(state, event):
    result = deepcopy(state)
    kind = event["kind"]
    if kind == "set":
        slot, value = event["slot"], event["value"]
        allowed = {"device": {"pump_a", "pump_b"}, "period": {"today", "yesterday"}}
        if slot not in allowed or value not in allowed[slot]:
            raise ValueError("Unsupported slot or value")
        if result[slot] != value:
            result[slot] = value
            result["revision"] += 1
            result["confirmed"] = None
    elif kind == "confirm":
        if event["revision"] != result["revision"]:
            raise ValueError("Confirmation refers to an old proposal")
        if any(result[k] is None for k in ["device", "period"]):
            raise ValueError("Cannot confirm incomplete state")
        result["confirmed"] = result["revision"]
    else:
        raise ValueError("Unsupported event")
    return result
s1 = update(initial, {"kind": "set", "slot": "device", "value": "pump_a"})
s2 = update(s1, {"kind": "set", "slot": "period", "value": "today"})
s3 = update(s2, {"kind": "confirm", "revision": 2})
s4 = update(s3, {"kind": "set", "slot": "device", "value": "pump_b"})
print("Before correction:", s3)
print("After correction:", s4)
```

```
assert initial["device"] is None
assert s3["confirmed"] == 2 and s4["confirmed"] is None
try:
    update(s4, {"kind": "confirm", "revision": 2})
except ValueError:
    pass
else:
    raise AssertionError("Stale confirmation was accepted")
```

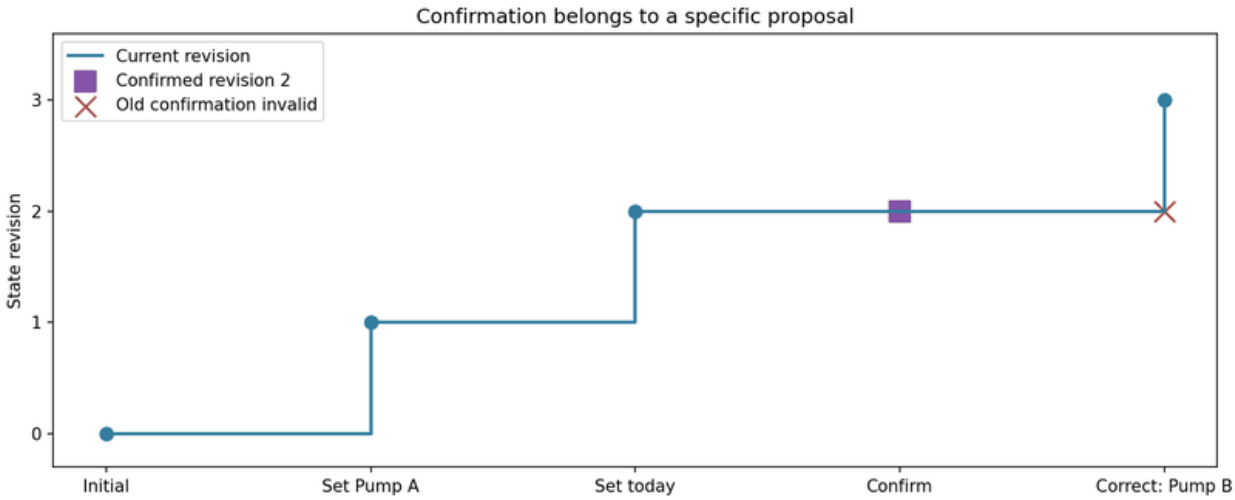


圖 2 — 修正改變狀態與確認範圍。保留修訂編號，才能看見過期接受，而非默默套到不同提案。

複製保留先前狀態，方便檢查與測試。真實系統也需要事件 ID，避免重複傳遞把更新套兩次。保存來源回合與剖析事件，才能追溯語言理解或狀態邏輯錯誤。欄位缺少不等於否定：「沒有發生故障」填入故障狀態，未提故障則仍未知。允許值有限，讓基準涵蓋範圍清楚。通用系統可使用學得欄位擷取與開放詞彙，但仍要驗證單位、結構與來源。模型信心不能代替合法值。

10. 信念狀態與有效澄清

直觀說法。話語含糊時，保留幾個帶機率的解讀，而不是直接猜。再依錯誤動作的代價，決定要行動還是提問。

簡單貝氏更新為：

$$b_t(h) = \frac{p(o_t \mid h) b_{t-1}(h)}{\sum_{h'} p(o_t \mid h') b_{t-1}(h')}.$$

其中 h 是候選解讀；$b_{t-1}(h)$ 是先前機率；o_t 是新觀察；$p(o_t \mid h)$ 是該解讀下的觀察概似；分母對候選 h' 正規化。簡化更新假設固定假說空間與已指定觀察模型。任務變動需要轉移模型或明確修正操作，不能反覆把每句話當獨立證據。

```
import numpy as np
```

```
hypotheses = ["pump_a", "pump_b", "unknown"]
prior = np.array([0.48, 0.48, 0.04])
ambiguous_likelihood = np.array([0.6, 0.5, 0.2])
posterior = prior*ambiguous_likelihood
posterior /= posterior.sum()
print("After ambiguous evidence:", dict(zip(hypotheses, posterior.round(3).tolist())))
action = "clarify" if posterior.max() < 0.9 else "query"
assert action == "clarify"
# A second synthetic observation explicitly names Pump B.
named_likelihood = np.array([0.02, 0.95, 0.03])
updated = posterior*named_likelihood
updated /= updated.sum()
print("After explicit naming:", dict(zip(hypotheses, updated.round(3).tolist())))
assert updated[1] > 0.9
assert np.isclose(updated.sum(), 1)
```

概似是教學設定，不是校準模型分數或量測的人類行為。學得系統需要驗證校準與分布轉移下的不確定性。保留未知假說，也避免封閉清單假裝每個真實請求都指已知裝置。

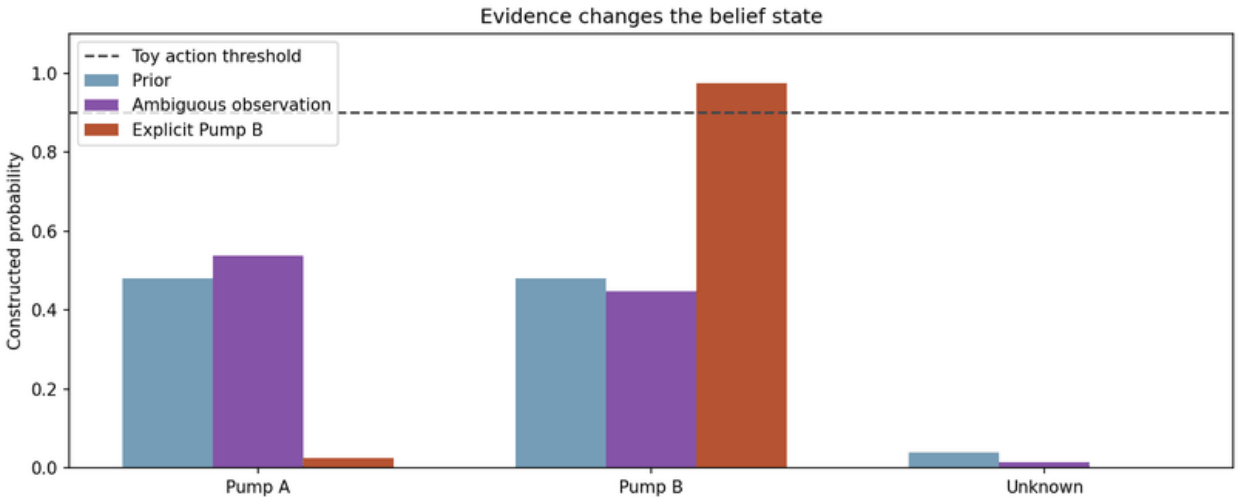


圖 3 — 信念狀態保留歧義，直到額外證據改變決策。機率為教學設定，不主張具有實證校準。

決策理論政策最小化期望損失：$a^\ast = \arg\min_a \sum_h b_t(h) L(a,h)$。其中 a 遍歷允許動作，$L(a,h)$ 是解讀 h 為真時執行動作 a 的代價。對候選問題 q，一步期望損失為：

$$V(q) = c(q) + \sum_o p(o \mid q) \min_a \sum_h b(h \mid o, q) L(a,h).$$

其中 $V(q)$ 是提問後的期望損失，$c(q)$ 是提問成本，o 遍歷可能回答，$p(o \mid q)$ 是目前信念下的回答機率，$b(h \mid o,q)$ 是收到回答後對解讀 h 的後驗，a 遍歷後續動作，$L(a,h)$ 是動作損失。最小值假定系統收到回答後選擇最佳下一步。應將 $V(q)$ 與直接動作的最小期望損失比較。這個有限決策計算是教學模型，不是已校準的人類回答模型。

```
import numpy as np
```

```
prior = np.array([0.55, 0.45])  # true device A or B
reply_given_device = np.array([[0.8, 0.2], [0.2, 0.8]])
question_cost = 1.0
for wrong_cost in [10.0, 2.0]:
    immediate = wrong_cost * (1 - prior.max())
    expected_after_question = question_cost
    for reply in range(2):
        mass = prior * reply_given_device[:, reply]
        probability = mass.sum()
        posterior = mass / probability
        expected_after_question += probability * wrong_cost * (1 - posterior.max())
    action = 'clarify' if expected_after_question < immediate else 'query A'
    print(wrong_cost, round(immediate, 2), round(expected_after_question, 2), action)
assert np.allclose(reply_given_device.sum(axis=1), 1)
```

輸出在錯誤查詢成本 `10` 時，比較 `4.5` 與 `3.0`，選擇澄清；成本 `2` 時，比較 `0.9` 與 `1.4`，直接查詢 A。信念與問題沒有改變，錯誤後果卻改變最佳動作。固定信心門檻隱藏這個依賴；期望損失則把它說清楚。應問能區分重要不確定性的問題：只有裝置含糊時，不要再問已確定的時間。

修正還需要另一個區分。「Actually, Pump B」可能改變指定裝置，而不是提供 Pump B 從頭就是意圖的證據。舊信念針對較早狀態，新話語則改變狀態本身。機率追蹤器除了觀察概似，也需要狀態轉移；或使用能辨認修正的明確更新規則。

11. 上下文、記憶與工具邊界

直觀說法。長對話放不進、也不需要放進每次模型輸入。保留下來的內容必須保留狀態：已撤回的請求不是有效請求。聊天機器人能呼叫工具時，「我做了」必須有實際結果支持。

可把上下文分為近期回合、結構化任務狀態與檢索證據。近期回合支持局部指涉與語氣；狀態保留已接受欄位及未解問題；證據支持事實主張。摘要不能把未驗證模型回覆升格為權威證據。有用記憶包含值、來源、時間、範圍與狀態（提議、接受或撤回）。滑動視窗可能只留「是」，卻刪了被確認的提案；摘要可能留某裝置，卻漏掉它已被另一裝置替換。應用合成對話測兩者，將重建狀態與明確事件紀錄比較。

工具連接的對話系統產生可由軟體執行的提案。提案、執行與結果是不同事件。「我查了資料庫」應由成功查詢結果支持，不能只是模型生成了查詢描述。工具學習研究討論基礎模型規劃、選工具、組參數及使用回饋 [27]。應用只應透過明確結構暴露允許操作，執行前驗證工具與參數，再把結果狀態及證據 ID 加入狀態。

唯讀證據也可能含命令文字。操作手冊可引用指令，檢索對話可含另一人的請求；它們仍是來源內容，不應更新當前使用者任務或擴大允許操作。結果要區分成功、沒有匹配觀察與執行錯誤；全部寫成一般流暢答案，會抹掉理解實際過程需要的資訊。重試唯讀查詢可能合理，重複改外部狀態的操作可能產生第二次效果，因此應在軟體中記錄請求 ID 與執行狀態，而不是靠語言模型記得。

12. 評估完整對話，不只一個回覆

直觀說法。看起來正確的回覆可能隱藏錯欄位、過期記憶或從未執行的動作。應跨完整對話量測狀態與最終結果。

Deriu 等人的綜述 [25] 說明任務對話、開放交談與互動問答為何需要不同評估方法。對指定欄位集合 J，定義精確狀態準確率：

$$A_{\text{state}} = \frac{1}{N}\sum_{t=1}^{N} \mathbf{1}\{\hat{s}_{t,j} = s_{t,j} \text{ for every } j \in J\}.$$

其中 N 是評估回合數；$s_{t,j}$ 是回合 t 的參考欄位 j；$\hat{s}_{t,j}$ 是預測值；$\mathbf{1}$ 在全部選定欄位匹配時為一，否則為零。嚴格指標揭露大量空值或簡單欄位平均所隱藏的錯誤。要指定欄位集合與未知值處理。

```
reference = [
    {"device": "pump_a", "period": None},
    {"device": "pump_a", "period": "today"},
    {"device": "pump_b", "period": "today"},
]
predicted = [
    {"device": "pump_a", "period": None},
    {"device": "pump_a", "period": "today"},
    {"device": "pump_a", "period": "today"},
]
slots = ["device", "period"]
joint = sum(all(p[j] == r[j] for j in slots)
            for p, r in zip(predicted, reference))/len(reference)
slot_accuracy = sum(p[j] == r[j] for p, r in zip(predicted, reference)
                    for j in slots)/(len(reference)*len(slots))
print("Exact state accuracy:", joint)
print("Per-slot accuracy:", slot_accuracy)
assert joint == 2/3 and slot_accuracy == 5/6
# The final device is wrong despite many correct fields.
assert predicted[-1]["device"] != reference[-1]["device"]
```

任務成功應檢查最終所求，不只看長度或親切風格。本例要求裝置、時間與證據正確，並尊重修正。短對話若查錯裝置，不是有效率的成功。以參考回覆計算 BLEU 在這裡並不合適，理由與翻譯相同，而且對話可接受的回覆更多，問題更嚴重。

人類評估要展示足夠歷史、指定判斷工作。只看最後回覆，無法判斷是否尊重修正。應隨機化並隱藏系統身分，定義錯誤類型並保留歧見。Schmidtova 等人 [28] 說明明確幻覺定義與方法報告為何重要。按對話切分資料，避免相鄰回合散落在訓練與測試；在對話層級回報不確定性，因為同一對話中的回合彼此相依。

現代觀點

直觀說法。翻譯與對話都走向大型預訓練模型，但本章的評估問題變得更重要，而不是更不重要。

綜述。 Stahlberg 在 *Journal of Artificial Intelligence Research*（2020）的回顧，從詞與句子嵌入及早期編碼器—解碼器，追溯神經機器翻譯到注意力與 Transformer，最後綜述近期趨勢 [22]。它是第 1–4 節材料最好的單一地圖。對話方面，Ni 等人的系統性綜述（*Artificial Intelligence Review*，2023）依模型與系統類型整理深度學習對話系統；任務導向與開放領域的區分，解釋為何一個回覆指標不能描述所有對話 [26]。Deriu 等人 [25] 把評估方法對應到這些系統類型。Qin 等人關於基礎模型工具學習的綜述（*ACM Computing Surveys*，2024）提醒我們，除了文字品質，也要檢查參數構造與結果使用 [27]。

2023–2026 最新進展。 WMT 記錄了大型語言模型進入翻譯的過程：WMT24 一般任務結果以專業人工標註，評估數個 LLM、參賽機器翻譯系統與線上服務，標題道出了平衡點：「LLM 時代已經到來，但機器翻譯尚未解決」[20]。NLLB 等大規模多語模型把涵蓋擴展到 200 種語言，明確以低資源語言為目標 [21]。評估方面，WMT22 指標任務發現，以專家 MQM 判斷為準時，神經指標比重疊指標可靠 [18]；WMT25 結果則把系統排名與片段層級錯誤偵測分開 [19]。

設計面向	要比較的選擇	需要的證據
架構	編碼器—解碼器、僅解碼器、多語	各語言對與領域的保留資料品質
解碼	貪婪、帶長度正規化的束搜尋、取樣	完整性、長度與語意錯誤，而不只是模型分數
評估	BLEU ／ chrF（附簽章）、學得指標、MQM 式人工覆核	與應用失敗類別的一致性
對話狀態	端到端生成、明確狀態與政策	狀態準確率、修正恢復、最終任務成功
工具使用	自由形式主張、附結果的已驗證提案	實際執行結果與回報結果一致

開放問題。低資源與方言翻譯、文件層級一致性、在罕見但關鍵的錯誤（否定、數字、名稱）上評估指標、可靠地自動接受個別輸出、任務變動下對話狀態的校準、多語修正，以及長對話中的過期記憶。較高的語料分數與流暢回覆都很容易取得；需要證據的是意義是否保留、最終動作是否正確。

重點整理

- SMT 結合分別估計的翻譯模型與語言模型；NMT 端到端訓練一個編碼器—解碼器，Transformer 讓它完全以注意力為基礎。
- 位移目標、保留結束詞元，在注意力與損失中都遮蔽填補；教師強制以正確前綴訓練，所以要在模型自己生成的輸出上測試（曝光偏差）。
- 貪婪與束搜尋以不同的搜尋量最佳化模型分數；長度正規化改變哪個假說勝出；兩者都修不好錯誤的模型。
- BLEU 是語料層級的截限 n-gram 精確率加短度懲罰；回報 sacreBLEU 簽章、加上 chrF 或學得指標，並直接檢查關鍵意義錯誤。

- 微型 Transformer 幾秒內就學會字典翻譯與詞序調整，它的交叉注意力在沒有監督下重現了對齊。
- 聊天機器人可以是 seq2seq 模型，但任務對話需要帶修訂的明確狀態、對含糊解讀的信念，以及依期望損失選擇的澄清。
- 讓工具提案、執行與結果保持分開，並以狀態與最終結果評估完整對話。

練習

1. **組合測試**。在翻譯器程式碼中，移除所有 `red` 修飾 `cable` 的訓練句對，再用含 `red cable` 的句子測試。比較這些句子與一般測試集的完全匹配率。

   提示

   在建立 `S` 與 `Y` 之前過濾 `train_pairs`，並保持詞彙表不變，讓失敗反映的是組合而不是未知詞。看看失敗案例的交叉注意力圖。

2. **束寬與長度**。用束寬 1、2、4、8 及 $\alpha \in \{0, 0.6, 1\}$ 評估 100 步的檢查點。回報 BLEU、完全匹配率、平均輸出長度與平均對數機率。較高的模型分數何時沒有改善 BLEU？

   提示

   束寬 1 且 $\alpha = 0$ 就是貪婪解碼。保存輸出並比較改變的句子，而不只是平均值。

3. **翻譯挑戰集合**。寫六個原創來源句，測否定、比較、單位、模態、時間與含糊指涉，指定任何可接受譯文都必須保留的意義。

   提示

   用加入 not 或交換裝置名等最小改動。來源充分性應獨立於目標是否優美。

4. **指標分歧**。建立兩個候選譯文：一個詞重疊高卻數量錯，另一個重疊較低卻意義正確。用 BLEU 函數評分，並設計能顯示差別的標註表。

   提示

   先請覆核者標出無支持的數量與遺漏的必需事實，再給整體判斷。保留來源，隱藏系統身分。

5. **修正範圍**。在追蹤器加測量欄位，測只改裝置、否定時間與完全重啟，不默默改不相關欄位。

   提示

   分別定義設定、撤回與重設事件。提案改變，就要使前一修訂的確認失效。

6. **錯誤動作報告**。設計區分查詢成功、無匹配與執行失敗的證據結構，寫出各狀態的回覆要求，並為每種狀態寫一段測試對話。

   提示

   結果附執行 ID 與狀態。生成提案本身不能支持「查詢成功」。

參考文獻

1. M. Hagiwara, *Real-World Natural Language Processing*, Manning, 2021, Chapters 6 and 8. Book ·Sequence-to-sequence chapter
2. S. McRoy, *Principles of Natural Language Processing*, open textbook, 2021, Chapter 9. Book · Generation chapter
3. P. F. Brown, S. A. Della Pietra, V. J. Della Pietra and R. L. Mercer, "The Mathematics of Statistical Machine Translation: Parameter Estimation," *Computational Linguistics* 19(2), 1993. Paper
4. P. Koehn, F. J. Och and D. Marcu, "Statistical Phrase-Based Translation," HLT-NAACL, 2003. Paper
5. K. Cho, B. van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk and Y. Bengio, "Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation," EMNLP, 2014. Paper
6. I. Sutskever, O. Vinyals and Q. V. Le, "Sequence to Sequence Learning with Neural Networks," NeurIPS, 2014. Paper
7. D. Bahdanau, K. Cho and Y. Bengio, "Neural Machine Translation by Jointly Learning to Align and Translate," ICLR, 2015. Paper
8. A. Vaswani et al., "Attention Is All You Need," NeurIPS, 2017. Paper
9. R. Sennrich, B. Haddow and A. Birch, "Neural Machine Translation of Rare Words with Subword Units," ACL, 2016. Paper
10. M. Ranzato, S. Chopra, M. Auli and W. Zaremba, "Sequence Level Training with Recurrent Neural Networks," ICLR, 2016. Paper
11. S. Bengio, O. Vinyals, N. Jaitly and N. Shazeer, "Scheduled Sampling for Sequence Prediction with Recurrent Neural Networks," NeurIPS, 2015. Paper
12. Y. Wu et al., "Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation," arXiv:1609.08144, 2016. Paper
13. K. Papineni, S. Roukos, T. Ward and W.-J. Zhu, "Bleu: a Method for Automatic Evaluation of Machine Translation," ACL, 2002. Paper
14. M. Post, "A Call for Clarity in Reporting BLEU Scores," WMT, 2018. Paper
15. M. Popović, "chrF: character n-gram F-score for automatic MT evaluation," WMT, 2015. Paper
16. R. Rei, C. Stewart, A. C. Farinha and A. Lavie, "COMET: A Neural Framework for MT Evaluation," EMNLP, 2020. Paper
17. C.-Y. Lin, "ROUGE: A Package for Automatic Evaluation of Summaries," Text Summarization Branches Out, 2004. Paper
18. M. Freitag et al., "Results of WMT22 Metrics Shared Task: Stop Using BLEU – Neural Metrics Are Better and More Robust," WMT, 2022. Report
19. A. Lavie et al., "Findings of the WMT25 Shared Task on Automated Translation Evaluation Systems: Linguistic Diversity is Challenging and References Still Help," WMT, 2025. Report
20. T. Kocmi et al., "Findings of the WMT24 General Machine Translation Shared Task: The LLM Era Is Here but MT Is Not Solved Yet," WMT, 2024. Report
21. NLLB Team et al., "Scaling neural machine translation to 200 languages," *Nature* 630, 2024. DOI
22. F. Stahlberg, "Neural Machine Translation: A Review," *Journal of Artificial Intelligence Research* 69, 2020. DOI
23. O. Vinyals and Q. V. Le, "A Neural Conversational Model," ICML Deep Learning Workshop,

2015. Paper
24. J. Li, M. Galley, C. Brockett, J. Gao and B. Dolan, “A Diversity-Promoting Objective Function for Neural Conversation Models,” NAACL-HLT, 2016. Paper
25. J. Deriu, Á. Rodrigo, A. Otegi, G. Echegoyen, S. Rosset, E. Agirre and M. Cieliebak, “Survey on Evaluation Methods for Dialogue Systems,” *Artificial Intelligence Review* 54, 2021. Survey
26. J. Ni, T. Young, V. Pandelea, F. Xue and E. Cambria, “Recent Advances in Deep Learning Based Dialogue Systems: A Systematic Survey,” *Artificial Intelligence Review* 56, 2023. Survey
27. Y. Qin et al., “Tool Learning with Foundation Models,” *ACM Computing Surveys* 57, 2024. Author manuscript ·Publication
28. P. Schmidtova et al., “Do My Eyes Deceive Me? A Survey of Human Evaluations of Hallucinations in NLG,” INLG, 2025. Survey
29. C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens and Z. Wojna, “Rethinking the Inception Architecture for Computer Vision,” CVPR, 2016. Paper
30. M. Freitag, G. Foster, D. Grangier, V. Ratnakar, Q. Tan and W. Macherey, “Experts, Errors, and Context: A Large-Scale Study of Human Evaluation for Machine Translation,” *Transactions of the Association for Computational Linguistics* 9, 2021. DOI

NLP 09・以 Transformer 做語言模型（自然語言生成）

訓練一個小型僅解碼器 Transformer 語言模型，用貪婪、束搜尋、溫度、top-k 與核取樣解碼，並評估摘要、資料到文字與有證據支持的生成。

網站版本: https://blog.shyandram.dev/zh-tw/learn/nlp/language-modeling-transformers/

你將學到

- 把序列機率分解成因果的下一詞元預測，並訓練一個遵守這個分解的小型僅解碼器（decoder-only）Transformer。
- 把 GPT 系列模型、它們的訓練目標與規模化研究放進同一張一致的圖。
- 用貪婪搜尋、束搜尋（beam search）、溫度、top-k 與核取樣（nucleus sampling）解碼，並說明各自改變與不改變什麼。
- 建立並評估三種生成任務：擷取式與抽象式摘要、資料到文字（data-to-text），以及開放式故事寫作。
- 以困惑度、ROUGE、人工覆核與模型評審量測生成品質，並了解各自的盲點。
- 以詞彙與稠密方法檢索證據，並用 Recall@k、MRR 與 nDCG 評估排序。
- 解釋幻覺（hallucination）、檢索增強生成，以及檢索證據與應用指令之間的界線。

先看全貌

語言模型依指定上下文替文字配置機率。這個簡單定義連結拼字補全、句子生成與可重用的上下文表示，但各種用途並沒有相同輸出契約。預測可能的延續與驗證事實回答是不同工作。流暢延續可能反映常見文字模式，卻不適合某個特定任務。

語言應用也必須選擇說什麼、清楚表達，並支持說出的內容，這是不同責任。可讀句子可能描述錯事件，正確句子也可能漏掉使用者需要的警告。部署再增加一項責任：結果必須可預期地抵達，並留下足夠上下文讓人追查失敗。

> **用最簡單的話說**
>
> 想像一個猜詞遊戲：看到句子開頭，猜下一個詞。接著把猜到的詞大聲說出來、接到句子後面，再猜下一個。語言模型就是這樣寫字的。它只能看已經說出的詞；偷看句子結尾就是作弊。而且聽起來對的猜測不等於真的，所以好的寫作者還會查核事實。

Hagiwara 第 5 章討論語言模型，第 8 章介紹注意力與 Transformer，第 9 章說明預訓練模型調適，第 10–11 章談訓練實務與部署 [1]。McRoy 第 6 章透過任務強調語言模型評估，第 9 章討論生成與摘要 [2]。本章把機率模型連到這些實際問題，只訓練微小合成模型；大型預訓練系統則透過已驗證的原始論文與綜述討論。

三個決策組織這個主題：**訓練目標**決定預測什麼；**注意力模式**決定哪些輸入位置可見；**解碼程序**決定預測如何變成使用者輸出。比較模型時要分開。改變取樣溫度會改變輸出分布，不會改變已學到哪些事實，也不會驗證產生的主張。

本章的位置：以計數為基礎的 n-gram 語言模型、平滑與最早的困惑度計算在 NLP 05；注意力、位置與 Transformer 區塊在 NLP 07；編碼器—解碼器翻譯與對話在 NLP 08。這裡把 Transformer 當成**僅解碼器**語言模型與文字生成器使用。雙向的替代方案 BERT 在 NLP 10。

1. 因果序列機率

直觀說法。依前面的詞元逐一預測，就能建立序列機率。訓練時雖然看得到完整序列，每次預測仍必須遵守可見性規則。

由機率鏈式法則：

$$p_\theta(x_1, \dots, x_T) = \prod_{t=1}^{T} p_\theta(x_t \mid x_{<t}).$$

其中 x_t 是位置 t 的詞元；T 是序列長度；$x_{<t}$ 是 t 之前的前綴；θ 是模型參數。起始詞元可提供第一個普通詞元的明確上下文，結束詞元則讓模型替停止配置機率。鏈式法則是精確等式，學得的條件分布則近似資料生成過程。

對含有 N 個受監督詞元位置的語料，平均負對數概似為：

$$\mathcal{L}_{\text{causal}} = -\frac{1}{N} \sum_{n=1}^{N} \log p_\theta(x_n \mid c_n).$$

其中 $\mathcal{L}_{\text{causal}}$ 是訓練目標；n 遍歷所有範例的有效預測位置；x_n 是目標詞元；c_n 是允許使用的前綴；N 不包含填補位置及其他刻意不監督的位置。自然對數使損失單位為每詞元的 nat。若批次平均把填補也計入，就改變了目標，可能獎勵模型預測人造填補符號。

NLP 05 的 n-gram 模型以最後 $n-1$ 個詞元取代 $x_{<t}$，因此結尾相同的兩個前綴必須得到相同預測。Transformer 語言模型則保留整個允許的前綴（直到上下文上限），由學得的注意力決定哪些較早詞元重要。分解方式相同，改變的是條件機率能讀取多少前綴。

教師強制（teacher forcing）在訓練時使用觀察到的前綴預測每個詞元。架構若強制因果遮罩，就能平行評估所有位置。生成時，前綴改為先前生成的詞元，錯誤可能影響後續預測。因此，即使權重相同，訓練目標與逐步生成行為仍需要不同診斷。

輸入與目標要錯開一個位置。完整序列若是起始、the、robot、moves、結束，模型輸入就是起始、the、robot、moves，目標則是 the、robot、moves、結束。若直接由未位移表示預測輸入詞元本身，就建立了與下一詞元預測無關的捷徑。第 3 節的程式明確寫出位移。

教師強制損失為何會漏掉部分生成錯誤？

教師強制提供正確前綴，即使模型自己會產生另一個前綴 [16]。假設參考從 `Pump A` 開始，生成卻選了 `Pump B`。後續詞元仍可能構成完全流暢的 Pump B 陳述。教師強制損失卻始終在 `Pump A`

後評估那些位置，沒有測試如何從這次生成錯誤恢復。差別在於：給定歷史下的預測，以及模型自己持續建立歷史時的行為。

詞元平均也隱藏不同語意後果。替換 `the` 可能只影響風格；替換 `not`、裝置 ID 或單位，卻能反轉結果。每個詞元貢獻一項對數損失，應用對這些變化的成本可能差很多。如果常見用詞改善，稀少關鍵差異卻惡化，平均損失下降仍可能伴隨更多重要欄位錯誤。因此，除了平均訓練目標，也要檢查完整輸出的命題。

2. 僅解碼器的 Transformer 語言模型

直觀說法。僅解碼器 Transformer 是一疊自注意力區塊，每個位置只能讀取自己和更早的位置。用下一詞元預測訓練後，它既能替文字評分，也能產生文字。

每個區塊先計算遮罩自注意力，再接逐位置前饋網路，並使用殘差連接與正規化，與原始 Transformer 相同 [3]；NLP 07 推導注意力公式。與雙向編碼器唯一的差別，是注意力 softmax 內的加法遮罩 B：當 $j \le i$ 時 $B_{ij} = 0$，否則 $B_{ij} = -\infty$。其中 i 是發出查詢的位置，j 是它可讀取的鍵位置。目標已位移，所以目前輸入位置仍是允許的前綴詞元，對角線可見。

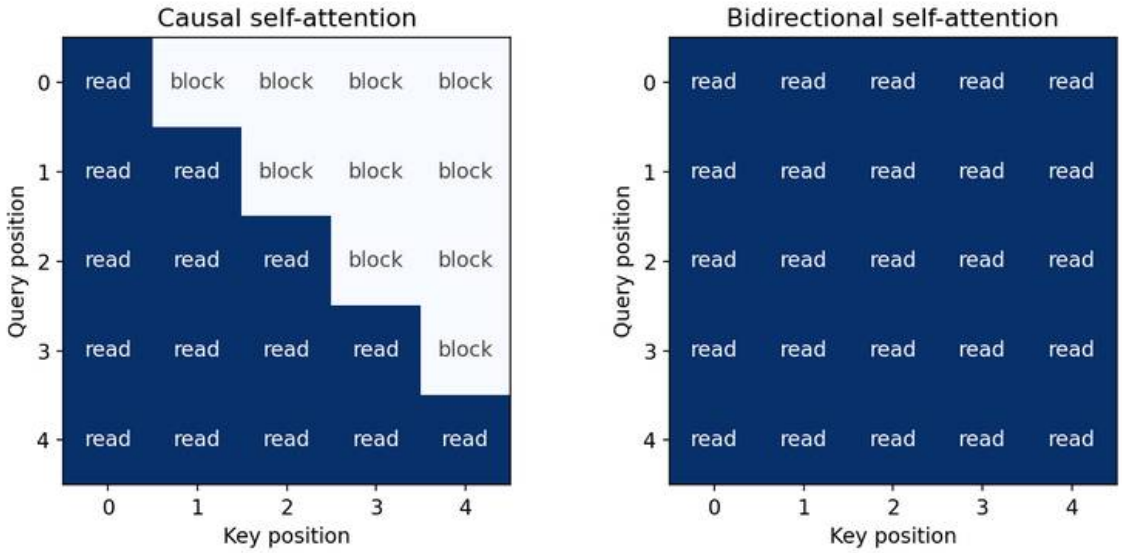


圖 *1* — 可見性是任務定義的一部分。列是發出查詢的位置，欄是它們可以讀取的位置。僅解碼器語言模型在每一層都使用左圖的模式。

最後一個線性層把位置 t 的頂層向量映成詞彙 logits $\mathbf{z}_t$，softmax 再把它轉成 $p_\theta(x_{t+1} \mid x_{\le t})$。因為遮罩的每一列都藏住未來，一次前向傳遞就能同時計算所有位置的訓練損失；生成時仍然每次只產生一個詞元。

GPT 系列

GPT 論文顯示這個單一設計能推到多遠。Radford 等人在 BooksCorpus 上，以遮罩自注意力訓練一個 12 層僅解碼器 Transformer 語言模型，再加上任務專用輸出層，針對分類、蘊涵與相似度等任務微調 [4]。配方是**生成式預訓練加判別式微調**：無標籤文字提供大部分學習訊號，少量標註資料再調適模型。

GPT-2 保留下一詞元目標，主要實驗則不再做任務專用微調。作者以新的網頁資料集 WebText 訓練，使用位元組層級的位元組對編碼（byte-pair encoding）讓任何字串都能表示，並報告語言模型開始能在零樣本（zero-shot）設定下執行問答與摘要等任務；最大模型有 15 億個參數 [5]。GPT-3 把同一方法擴大成 1,750 億參數的自迴歸模型，評估時只以文字給出任務說明與少數示範，不做梯度更新 [6]。這是同一條線上的三個點：架構與目標幾乎沒變，改變的是資料、模型大小，以及呈現任務的方式。

正確理解這條線，對後面各節很重要。提示（prompt）就是一個前綴 $x_{<t}$。「零樣本」與「少樣本」描述的是前綴內容，不是另一種模型。延續的品質仍取決於訓練文字讓什麼變得可能，這也是後來要再加上指令微調、偏好學習與檢索（第 9–11 節）的原因。

3. 可執行的因果模型

直觀說法。小模型可以學會合成語言模式，卻對沒見過的組合泛化很差。我們還測試低損失不能證明的性質：未來詞元不得影響較早預測。

以下獨立 CPU 範例使用自訂等長句子。訓練與測試句不同，但每個普通詞彙都曾出現在訓練中。程式沒有填補、預訓練權重或資料下載，刻意作為機制實驗而非語言基準。

實作使用 AdamW 最佳化 [9]、前饋區塊中的 GELU 活化函數 [10]，以及注意力和前饋轉換之前的層正規化（layer normalization）[11]。這些選擇定義小型實驗，不是重現原 Transformer 的所有細節。研究因果遮罩時保持它們固定，避免最佳化或活化函數的改變干擾比較。

```
import math
import torch
from torch import nn
from torch.nn import functional as F

torch.manual_seed(11)
torch.set_num_threads(1)
train_text = [
    'the red robot moves', 'the blue robot waits',
    'a red sensor waits', 'a blue sensor moves',
    'the red sensor moves', 'a blue robot waits',
]
test_text = ['a red robot moves', 'the blue sensor waits']
vocab = ['<bos>', '<eos>'] + sorted(set(' '.join(train_text).split()))
index = {word: i for i, word in enumerate(vocab)}
def encode(rows):
    return torch.tensor([[index[w] for w in ['<bos>'] + s.split() + ['<eos>']] for s in
    ↪  rows])
train, test = encode(train_text), encode(test_text)
```

```
class TinyCausalModel(nn.Module):
    def __init__(self):
        super().__init__()
        self.token = nn.Embedding(len(vocab), 16)
        self.position = nn.Embedding(5, 16)
        self.norm1, self.norm2 = nn.LayerNorm(16), nn.LayerNorm(16)
        self.q, self.k, self.v = [nn.Linear(16, 16, bias=False) for _ in range(3)]
        self.output = nn.Linear(16, 16)
        self.ff = nn.Sequential(nn.Linear(16, 32), nn.GELU(), nn.Linear(32, 16))
        self.head = nn.Linear(16, len(vocab))
    def forward(self, ids):
        length = ids.shape[1]
        h = self.token(ids) + self.position(torch.arange(length))
        z = self.norm1(h)
        scores = self.q(z) @ self.k(z).transpose(-1, -2) / math.sqrt(16)
        blocked = torch.ones(length, length, dtype=torch.bool).triu(1)
        weights = scores.masked_fill(blocked, float('-inf')).softmax(dim=-1)
        h = h + self.output(weights @ self.v(z))
        h = h + self.ff(self.norm2(h))
        return self.head(h)

model = TinyCausalModel()
optimizer = torch.optim.AdamW(model.parameters(), lr=0.01)
def loss(rows):
    logits = model(rows[:, :-1])
    return F.cross_entropy(logits.reshape(-1, len(vocab)), rows[:, 1:].reshape(-1))
initial = loss(train).item()
for _ in range(180):
    optimizer.zero_grad()
    objective = loss(train)
    objective.backward()
    optimizer.step()
model.eval()
with torch.no_grad():
    train_nll, test_nll = loss(train).item(), loss(test).item()
    original = train[:1, :-1]
    changed = original.clone()
    changed[0, -1] = index['blue']
    gap = (model(original)[:, :-1] - model(changed)[:, :-1]).abs().max().item()
assert train_nll < initial and gap < 1e-6
print('initial/train/test NLL:', *(round(x, 3) for x in [initial, train_nll, test_nll]))
print('held-out perplexity:', round(math.exp(test_nll), 3))
print('future-token perturbation gap:', gap)
```

輸出：

```
initial/train/test NLL: 2.194 0.409 3.17
held-out perplexity: 23.81
future-token perturbation gap: 0.0
```

斷言只檢查訓練損失下降，不要求保留損失改善特定幅度。這裡保留損失（每詞元 3.17 nat）甚至高於未訓練時的起始損失（2.194）：保留句把熟悉詞語組成訓練未包含的組合，而模型已自信地背下六個訓練模式。大的訓練—測試落差說明，學會觀察到的模式不代表有組合泛化能力。

未來詞元擾動測試改變最後輸入詞元，再比較所有較早位置的 logits。因果注意力應讓這個變動對較早預測不可見，印出的差距正好是零。測試直接對準可見性契約，但通過不代表所有資料隔離都成立：跨切分重複文件、使用禁止的測試資料建詞彙表，或未來中繼資料，都能造成其他洩漏。

模型沒有 dropout，因此本例評估模式不會改變隨機層；仍明確呼叫它，因為真實模型必須區分訓練與評估。損失函數展平批次及位置維度，保留詞彙維度作為類別。若不小心交換這些軸，程式可能照樣執行，卻最佳化錯誤問題。

每個序列等長，位置表也涵蓋五個模型輸入位置。學得的位置表限制支援的最遠位置；超過範圍的序列不是普通長輸入，而是非法索引。擴充成可變長資料，要分別加入注意力與損失遮罩。注意力遮罩控制表示可讀什麼；損失遮罩控制哪些目標對目標函數有貢獻。兩者處理不同問題，需要獨立測試。

4. 訓練目標與規模化

直觀說法。大型語言模型使用與上面小模型相同的下一詞元損失。改變的是文字量、參數量與投入的計算量，而這三者之間有可量測的取捨。

Kaplan 等人訓練多組 Transformer 語言模型，報告保留損失約隨模型大小、資料量與計算量呈冪律下降 [7]。只看模型大小時：

$$L(N) = \left(\frac{N_c}{N}\right)^{\alpha_N}.$$

其中 L 是保留交叉熵損失，N 是不含嵌入的參數數量，N_c 是擬合常數，$\alpha_N > 0$ 是擬合指數。Hoffmann 等人之後擬合含不可消除項的形式 $L(N, D) = E + A/N^{\alpha} + B/D^{\beta}$，其中 D 是訓練詞元數，E 是任何模型都無法移除的損失，A, B, α, β 是擬合常數；他們的結論是，固定計算預算時，參數與訓練詞元應以大致相同比例擴大 [8]。ML 10 推導計算最適的取捨，並以程式擬合規模化定律。

有兩個提醒讓規模化保持在正確位置。第一，擬合常數取決於分詞器、資料組成與訓練程序，不能在不相關的設定間轉移。第二，規模化定律預測的是**損失**，不是特定任務行為。較低的下一詞元損失通常有幫助，但模型在摘要中是否保留否定、是否會編造引用，仍要直接量測（第 8 節）。

5. 困惑度與有意義的比較

直觀說法。困惑度總結保留詞元預測表現，但數值取決於詞元定義與評估位置。

平均負對數概似為 $\mathcal{L}$ 時：

$$\mathrm{PPL} = \exp(\mathcal{L}).$$

其中 PPL 是困惑度（perplexity）；exp 是自然指數；$\mathcal{L}$ 對同一組評估位置使用自然對數。較低困惑度表示模型在該評估契約下，對觀察到的詞元配置較高平均機率，不是事實正確、實用性或合語法性的直接指標。IT 01 把這個量連到交叉熵，NLP 05 則替計數模型計算它。

只有分詞與輸入慣例相容時才比較困惑度。一個詞可能在某詞彙表是單一詞元，在另一個表被拆成多個詞元，平均每詞元損失便使用不同單位。還應記錄是否評分起始與結束符號、文件如何分段、上下文長度，以及評估視窗是否重疊。重複計算目標位置會改變權重。

評估文件應與訓練及其衍生文件獨立。把同一文件相鄰段落分到不同集合，可能暴露風格、事實與幾乎相同措辭。具有時間相依性的應用應採適當時間切分，並保留文件版本。單一名稱或短語沒見過，不能證明周圍文件獨立。

McRoy 的基準觀點提醒我們，序列預測之外還要評估任務行為 [2]。分類器需要類別表現，擷取器需要區段與受支持關係，生成器需要意義保留與遺漏評估。困惑度較低的模型仍可能把「沒有移動」改成「移動」而失敗。評估要對準使用者真正的結果。

可重現的語言模型實驗

別人能知道哪些文字、詞元與決策產生結果，結果才有用。先記錄文件切分與理由：獨立合成句子可能適合隨機切分，客服助理則可能需要按客戶或時間隔離。本例詞彙只來自訓練句，測試組合也刻意全部使用已知詞，因而隔離一個狹義的組合泛化問題。

接著記錄分詞、序列邊界、目標位移、最大上下文與損失分母。程式評分每個下一詞元目標，包括結束詞元，但不把起始詞元當目標。若別的實驗排除停止預測，或讓不同長度句子等權，其數字回答的就是不同問題。最後區分選擇與報告：在開發資料上選架構、訓練長度與解碼行為，只在選擇凍結後檢視保留集合。保存前處理版本、模型設定、種子、允許的輸入上下文與輸出設定，敏感原文件則放在適當存取控制下。

6. 解碼：從分布到文字

直觀說法。訓練模型產生分數；解碼器決定選最強詞元、搜尋數個延續，還是取樣其他選項。這些選擇都不會讓無支持內容變可靠。

溫度

對 logit z_v 與正溫度 τ：

$$q_\tau(v) = \frac{\exp(z_v/\tau)}{\sum_{u \in V} \exp(z_u/\tau)}.$$

其中 v 是候選詞元；V 是詞彙表；u 遍歷分母候選；z_v 是模型未正規化分數；τ 是溫度；q_τ 是所得取樣分布。低溫集中在高分詞元，高溫則分散機率。當 $\tau \to 0$，取樣趨近貪婪選擇，也就是直接取最高分詞元。事實錯誤也可能是最高 logit 的選項，因此降低隨機性不會移除它。

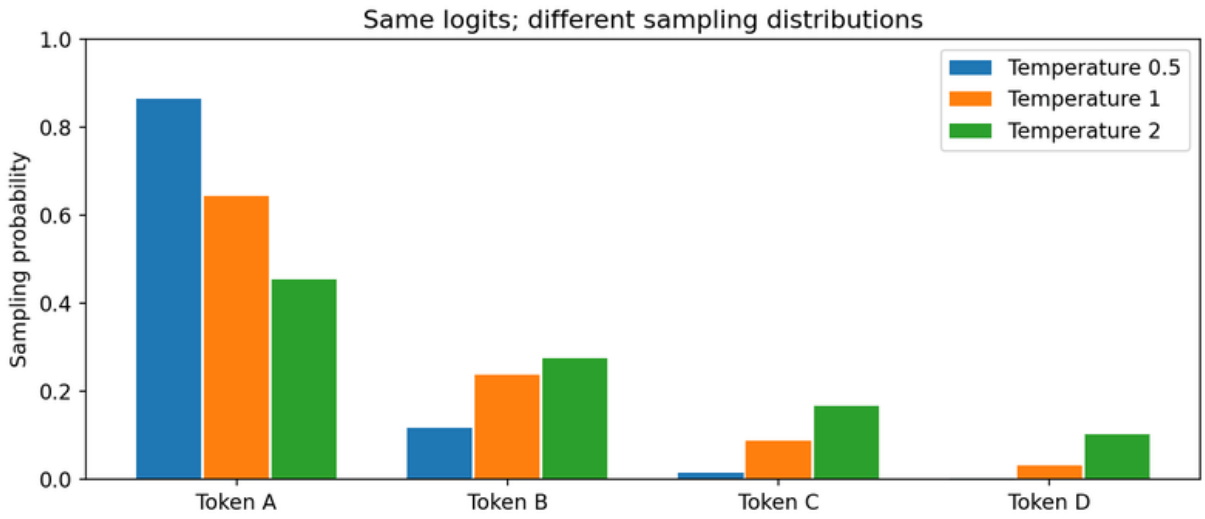


圖 2 — 固定合成 *logits* 下，溫度改變取樣分布。更集中的輸出不會自動更正確。

貪婪搜尋與束搜尋

貪婪解碼選最高的下一詞元，束搜尋保留數個部分序列。候選序列分數為：

$$\ell(y \mid x) = \sum_{t=1}^{T} \log p_\theta(y_t \mid y_{<t}, x).$$

其中 ℓ 是給定提示或來源 x 時完整輸出 y 的對數機率；T 包含結束詞元；其他符號同前。原始對數機率中，每個新增詞元都貢獻非正項，因此可能偏好短序列。長度調整會改變搜尋目標，應說明形式並在驗證資料調整，不能默默加入。

以下微型條件模型有兩個競爭的起始詞，展示選最佳下一詞不一定得到最佳完整序列。機率為實驗設定，不由語言資料估計。

```
import math

table = {
    (): {"safe": 0.55, "close": 0.45},
    ("safe",): {"stop": 0.51, "wait": 0.49},
    ("close",): {"valve": 0.99, "door": 0.01},
}
def next_probs(prefix):
    return table.get(prefix, {"<eos>": 1.0})
def decode(width):
    beams = [((), 0.0)]
    for _ in range(3):
        candidates = []
        for seq, score in beams:
            if seq and seq[-1] == "<eos>":
                candidates.append((seq, score))
```

```
                continue
            for token, prob in next_probs(seq).items():
                candidates.append((seq + (token,), score + math.log(prob)))
        beams = sorted(candidates, key=lambda pair: pair[1], reverse=True)[:width]
    return beams[0]
greedy, beam = decode(1), decode(2)
print(greedy[0], round(math.exp(greedy[1]), 4))
print(beam[0], round(math.exp(beam[1]), 4))
assert greedy[0] == ("safe", "stop", "<eos>")
assert beam[0] == ("close", "valve", "<eos>")
assert beam[1] > greedy[1]
```

輸出：

```
('safe', 'stop', '<eos>') 0.2805
('close', 'valve', '<eos>') 0.4455
```

束搜尋在人工樹中找到較可能的序列。這只說明模型的機率配置。若 close valve 與來源指令矛盾，更好的機率搜尋仍會產生錯誤的應用行為。搜尋品質與任務正確需要分開量測。

搜尋錯誤、模型錯誤與錯誤目標

微型束搜尋之所以成功，是因為保留 `close` 前綴，直到發現高機率延續。貪婪解碼在看到延續前，就丟掉該分支。相對於人工設定的序列機率目標，這是**搜尋錯誤**。較寬的束可減少這類錯誤，但仍是有界搜尋；任何深度都可能把實用前綴排除在保留集合之外。

原始前綴對數機率提供實用上界。後續每個對數機率項都非正，因此延伸前綴不會提高原始分數。若完成候選的分數高於所有未完成前綴，它就是保留下來分支中的最佳結果。這個論證要求未調整的總和；長度獎勵或正規化會改變上界，停止規則也必須一起考慮。

接著區分**模型錯誤**。若來源要求保持閥門開啟，模型卻替 `close valve` 配最高序列機率，精確機率搜尋只會更穩定地呈現模型錯誤。搜尋不能創造評分中缺少的來源相容性。最後，最大機率只是其中一種決策目標。通用短回答可能機率很高，卻漏掉必需警告。調整束寬之前，先評估目標是否獎勵使用者需要的結果。

top-k 與核取樣

在開放式寫作中，最大化機率常常不是正確目標。Holtzman 等人觀察到，即使模型很強，把概似當解碼目標仍會產生平淡且奇怪地重複的文字；而從完整分布直接取樣，又可能從長尾挑到極不可能的詞元 [12]。兩種截斷規則介於兩個極端之間。

top-k 取樣保留機率最高的 k 個詞元再重新正規化。Fan 等人在故事生成中使用它，每一步從 10 個最可能的候選中取樣，並發現此任務上它比束搜尋有效 [13]。**核取樣（top-p）**先依機率由高到低排序，再保留累積機率達到 p 的最短前綴 [12]：

$$\pi_1 \ge \pi_2 \ge \cdots \ge \pi_{|V|}, \qquad \pi_j = p_\theta(v_{(j)} \mid x_{\le t}),$$
$$k^* = \min\left\{k \in \{1,\dots,|V|\} : \sum_{j=1}^{k} \pi_j \ge p\right\}, \qquad V^{(p)} = \{v_{(1)},\dots,v_{(k^*)}\},$$
$$q(v) = \frac{p_\theta(v \mid x_{\le t})\,\mathbf{1}[v \in V^{(p)}]}{\sum_{u \in V^{(p)}} p_\theta(u \mid x_{\le t})}.$$

其中 $v_{(j)}$ 是排序第 j 的詞元，π_j 是其機率，$|V|$ 是詞彙表大小，k^* 是保留的前綴長度，$V^{(p)}$ 是核（nucleus），$p \in (0, 1]$ 是目標機率質量，$\mathbf{1}[\cdot]$ 是指示函數，q 是重新正規化後的取樣分布。相同機率需固定同分排序規則；程式保留原始索引順序。與 top-k 的差別在於核會**自動調整**：前綴只有一個明顯延續時，核可能只含一個詞元；前綴很開放時，核可能包含很多詞元。

由高到低取前綴的條件很重要。當機率為 0.4, 0.3, 0.2, 0.1、門檻為 0.5 時，top-p 保留前兩個詞元。第一個與最後一個也構成總和達到 0.5 的兩元素集合，但它跳過機率更高的候選，並非此處定義的核。下方斷言會檢查這個差別。

```
import numpy as np

tokens = np.array(['valve', 'pump', 'door', 'sensor', 'cabinet', 'label', 'fish', 'moon'])
logits = np.array([3.0, 2.6, 1.2, 1.0, 0.2, -0.5, -1.5, -2.0])

def softmax(z):
    e = np.exp(z - z.max())
    return e / e.sum()

def temperature(z, tau):
    return softmax(z / tau)

def top_k(p, k):
    keep = np.argsort(-p)[:k]                 # indices of the k largest
    q = np.zeros_like(p); q[keep] = p[keep]
    return q / q.sum()

def nucleus(p, top_p):
    if not 0 < top_p <= 1:
        raise ValueError("top_p must be in (0, 1]")
    order = np.argsort(-p, kind="stable")
    cumulative = np.cumsum(p[order])
    size = int(np.searchsorted(cumulative, top_p) + 1)   # smallest prefix reaching top_p
    q = np.zeros_like(p); q[order[:size]] = p[order[:size]]
    return q / q.sum()

p = softmax(logits)
print('greedy choice:', tokens[np.argmax(p)])
for name, q in [('tau=0.5', temperature(logits, 0.5)), ('tau=1.0', p),
                ('tau=2.0', temperature(logits, 2.0)),
                ('top-k=3', top_k(p, 3)), ('top-p=0.9', nucleus(p, 0.9))]:
    print(f'{name:9s} support={int((q > 0).sum())}  P(valve)={q[0]:.3f}  '
          f'tail mass={q[4:].sum():.3f}')

# The nucleus adapts its size to the shape of the distribution.
flat = softmax(np.zeros(8))
```

```
peaked = softmax(np.array([6.0, 1, 0, 0, 0, 0, 0, 0]))
print('nucleus size, flat vs peaked:',
      int((nucleus(flat, 0.9) > 0).sum()), int((nucleus(peaked, 0.9) > 0).sum()))

rng = np.random.default_rng(0)
draws = rng.choice(len(tokens), size=2000, p=nucleus(p, 0.9))
assert set(draws) <= set(np.flatnonzero(nucleus(p, 0.9)))
assert np.array_equal(np.flatnonzero(nucleus(np.array([0.4, 0.3, 0.2, 0.1]), 0.5)), [0, 1])
assert np.isclose(top_k(p, 3).sum(), 1) and np.argmax(temperature(logits, 0.1)) == 0
```

輸出：

```
greedy choice: valve
tau=0.5   support=8  P(valve)=0.667  tail mass=0.003
tau=1.0   support=8  P(valve)=0.481  tail mass=0.052
tau=2.0   support=8  P(valve)=0.312  tail mass=0.190
top-k=3   support=3  P(valve)=0.545  tail mass=0.000
top-p=0.9 support=4  P(valve)=0.507  tail mass=0.000
nucleus size, flat vs peaked: 8 1
```

這裡的「tail mass」是最不可能的四個詞元所剩的機率。溫度只重塑分布，所以每個詞元都仍可能被選到：在 $\tau = 2$ 時，將近五分之一的機率落在 `cabinet`、`label`、`fish` 與 `moon`。截斷則直接移除長尾。在均勻分布上，核保留全部八個詞元；在尖峰分布上只保留一個。固定的 k 在兩種情況下都保留三個。

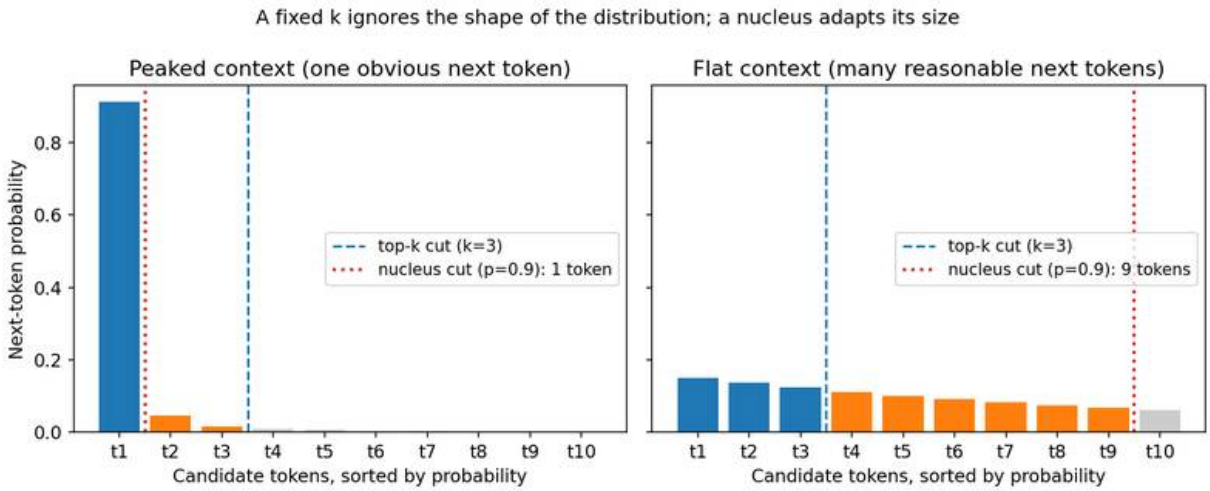


圖 3 — 兩個合成分布上的 *top-k* 與核截斷。藍色長條兩種規則都保留，橘色只被其中一種保留，灰色兩者都移除。

實務上這些設定會組合使用（例如先調溫度再做核取樣），並依任務選擇。資料到文字的報告通常需要確定且可重現的輸出；腦力激盪則需要多樣性。取樣多樣不是品質的證據，重複取樣得到一致結果也不是主張的獨立驗證：它們共享同一模型，也往往共享訓練偏誤。

停止與長度

生成還需要停止政策與長度預算。模型可能太早停止，也可能持續輸出重複內容。要明訂輸出上限、停止規則及達上限後的行為；截斷的摘要不能當完整結果回傳。解碼參數應與模型一起記錄，讓評估可重現；只改解碼器，就可能改變所有下游指標。

7. 自然語言生成任務

直觀說法。生成是產生內容與形式都符合任務的序列。輸入可以是文件、表格、提示或檢索段落；任務決定什麼算錯誤。

僅解碼器模型用同一種方式處理所有條件任務：把輸入當成前綴。條件分解為：

$$p_\theta(y \mid x, c) = \prod_{t=1}^{T} p_\theta(y_t \mid y_{<t}, x, c).$$

其中 x 是來源輸入；c 是要求語言或格式等任務脈絡；$y = (y_1, \dots, y_T)$ 是含結束詞元的輸出；$y_{<t}$ 是 t 前的詞元；θ 是學得參數。在僅解碼器模型中，x 與 c 被序列化成前綴，損失通常只計算 y 的詞元；編碼器—解碼器（NLP 08）則另外編碼 x。無論哪種，分解只描述分布，不保證取樣輸出獲得來源支持。

選模型前先定義生成問題

先用普通語言寫任務契約。設備報告可寫：「給定附時間戳的觀察，用兩句描述目前溫度及任何明確記錄的故障，保留裝置名稱、單位、數字與否定。」它揭露「好好摘要」這種無限制指令掩蓋的遺漏和矛盾。

分開三種操作。**內容選擇**選使用者相關觀察；**句子規劃**決定順序、分組與指代表達；**文字實現**把計畫寫成合語法文字。McRoy [2] 提供語言學基礎。神經系統可在內部合併這些操作，但仍需分別測試每項責任。

來源若有高溫、稍後正常溫度及明確沒有故障，選最高值與選最新值會產生不同報告。語法流暢不會決定該用哪個規則。要記錄描述目前狀態、極端事件，還是整段軌跡；時間與範圍屬於輸入契約。

資料到文字：模板基準

資料到文字生成把結構化紀錄寫成文字。Wiseman、Shieber 與 Rush 建立一個大型語料，把資料紀錄與描述文件配對，並提出擷取式評估方法，檢查文字實際傳達了哪些紀錄；他們測試的神經系統產生流暢文字，模板基準卻在其中部分指標上勝過它們 [15]。輸入結構狹窄時，模板是有用基準，可完全控制允許的主張，方便定位失敗。限制也清楚：無法自然涵蓋任意關係、指涉或篇章。應量測涵蓋範圍，不要把模板視為天生不足。

下面是狹義結構的確定報告函數，展示欄位驗證與可追溯輸出。事件紀錄是可信的結構輸入；從原文取得並驗證紀錄屬於另一個 NLP 階段。

```
import math

VERSION = "equipment-template-1"
def report(event):
    required = {"device", "temperature_c", "fault", "evidence_id"}
    if set(event) != required:
        raise ValueError("Unexpected input fields")
    if not isinstance(event["device"], str) or not event["device"].strip():
        raise ValueError("Missing device")
    value = event["temperature_c"]
    if isinstance(value, bool) or not isinstance(value, (int, float)) or not
     ↪  math.isfinite(value):
        raise ValueError("Temperature must be finite")
    if event["fault"] not in ("present", "absent", "unknown"):
        raise ValueError("Unknown fault status")
    if not isinstance(event["evidence_id"], str) or not event["evidence_id"]:
        raise ValueError("Missing evidence ID")
    phrases = {"present": "A fault was recorded.",
               "absent": "No fault was recorded.",
               "unknown": "Fault status was not recorded."}
    text = f"{event['device']} measured {value:g} C. {phrases[event['fault']]}"
    return {"text": text, "evidence_id": event["evidence_id"], "version": VERSION}
output = report({"device": "Pump A", "temperature_c": 30,
                 "fault": "absent", "evidence_id": "synthetic-event-01"})
print(output)
assert output["text"] == "Pump A measured 30 C. No fault was recorded."
assert output["version"] == VERSION
```

輸出：

```
{'text': 'Pump A measured 30 C. No fault was recorded.', 'evidence_id': 'synthetic-event-01',
 ↪  'version': 'equipment-template-1'}
```

注意 absent 與 unknown 的差別。「沒有記錄到故障」描述的是紀錄證據，不表示故障不可能。編碼這個差別，能防止文字實現階段把缺少證據升格為確定。學得生成器要在新增彈性解決了此基準已記錄的失敗時才有價值，而且應該用這些區別來測試它。

擷取式摘要：在預算內選證據

擷取式摘要選來源句子，抽象式摘要可改寫或融合句子。擷取讓每句的來源清楚，但所選句子仍可能因缺上下文、時間衝突或未解代名詞而誤導。選了「它故障了」卻沒有先行詞，即使每個字都出自來源，仍是篇章錯誤。

假設文件有候選句 $s_1, \dots, s_n$，句 i 用單位向量 v_i 表示，使用者查詢用單位向量 q 表示。簡單相關分數是 $r_i = v_i^{\top} q$。為避免重複，可用最大邊際相關性（maximal marginal relevance，MMR）[14]：

$$i^* = \arg\max_{i \notin S} \left[\lambda r_i - (1-\lambda) \max_{j \in S} v_i^\top v_j \right].$$

其中 S 是已選句索引集合；i^* 是下一個索引；$\lambda \in [0,1]$ 平衡相關性與已選句的相似度；S 為空時，最大相似度定義為零。向量長度為一，所以內積是餘弦相似度。這是貪婪選擇規則，不是摘要品質全域最佳的證明。

以下用二元詞指示讓每個決策都可檢查，四句皆為合成。移除少數功能詞能降低語法的支配程度，卻也可能移除有用的區別。

```
import re
import numpy as np

sentences = [
    "Pump A temperature reached 80 C during the test.",
    "The test recorded Pump A temperature at 80 C.",
    "Pump A vibration triggered a shutdown warning.",
    "The technician replaced a label on the cabinet.",
]
query = "pump temperature vibration shutdown"
stop = {"a", "the", "at", "on", "during"}
def tokens(text):
    return set(re.findall(r"[a-z0-9]+", text.lower())) - stop
vocab = sorted(set().union(*(tokens(s) for s in sentences), tokens(query)))
def vector(text):
    v = np.array([w in tokens(text) for w in vocab], dtype=float)
    return v / max(np.linalg.norm(v), 1e-12)
V = np.stack([vector(s) for s in sentences])
q = vector(query)
relevance, similarity = V @ q, V @ V.T
selected = []
for _ in range(2):
    redundancy = similarity[:, selected].max(axis=1) if selected else np.zeros(4)
    score = 0.55 * relevance - 0.45 * redundancy
    score[selected] = -np.inf
    selected.append(int(np.argmax(score)))
print("Selected indices:", selected)
print("Summary:", " ".join(sentences[i] for i in sorted(selected)))
assert len(set(selected)) == 2
assert 2 in selected and not ({0, 1} <= set(selected))
```

輸出：

```
Selected indices: [2, 0]
Summary: Pump A temperature reached 80 C during the test. Pump A vibration triggered a
 ↪  shutdown warning.
```

選到的兩句包含停機句與一個溫度句，避免幾乎重複的溫度觀察。選後排序索引可恢復來源順序：選擇順序量測邊際價值，呈現順序則要支持理解。

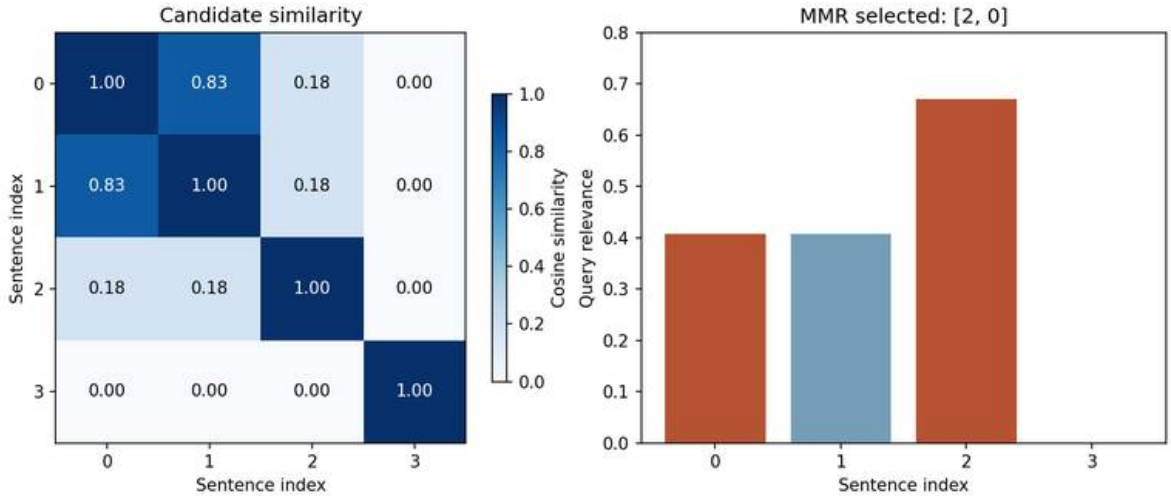


圖 4 — 擷取基準把取捨顯示出來。相似的溫度句競爭有限的摘要空間，停機句則提供不同內容。

真實預算通常限制詞元而非句數。整句分詞長度放得進剩餘預算，才能加入；默默截句可能刪掉條件或否定。多文件摘要應把近似重複報告分組，避免多份副本假裝成獨立證據。保留來源 ID 與時間戳，讓讀者能追查所選證據。

相關性、可行性與內容保留是不同目標

MMR 避免選取與先前句子相似的內容 [14]，但不知道哪些事實必須出現。若警告與已選觀察共享很多詞，相似度懲罰可能壓低它，即使它新增的限定十分重要。向量多樣性只是重複度的代理量，不直接量測完整意義。

內容要求會改變可行集合。對詞元預算與必需警告：

$$\sum_{i \in S} \ell_i \le B, \qquad S \cap W \neq \emptyset.$$

其中 S 是所選句子的索引集合，ℓ_i 是候選 i 的完整詞元長度，B 是預算，W 包含表達必需警告的候選。第一個條件限制大小，第二個確保涵蓋。若沒有集合同時滿足兩者，該預算下的擷取任務就不可行。改相關性係數，無法讓過長句子放得下，也無法創造缺少的警告。

以下獨立範例隔離這個差別。效用是明示的教學選擇，不是學得的相關分數：

```
import itertools

sentences = [
    'Pump A temperature reached 80 C during the test.',
    'The test recorded Pump A temperature at 80 C.',
    'Pump A vibration triggered a shutdown warning.',
    'The technician replaced a label on the cabinet.',
]
lengths = [len(s.split()) for s in sentences]
utility = [0.9, 0.85, 0.6, 0.2]
subsets = [tuple(i for i in range(4) if keep[i])
```

```
           for keep in itertools.product([False, True], repeat=4)]
feasible = [s for s in subsets if sum(lengths[i] for i in s) <= 20]
score = lambda s: sum(utility[i] for i in s)
ordinary = max(feasible, key=score)
with_warning = max((s for s in feasible if 2 in s), key=score)
print('Lengths:', lengths)
print('Without requirement:', ordinary, 'with warning:', with_warning)
assert ordinary == (0, 1) and with_warning == (0, 2)
```

輸出：

```
Lengths: [9, 9, 7, 8]
Without requirement: (0, 1) with warning: (0, 2)
```

這裡以空白分隔的詞計算長度；模型詞元預算則必須使用該模型的分詞器。無限制的選擇重複溫度內容，必含警告的要求則改變所選配對。不能接受遺漏時，更複雜的評分仍需要任務的硬內容要求。

抽象式摘要

改寫模型能融合觀察、提高可讀性，也能編造來源未建立的連接。「溫度高，因此幫浦故障」若來源只記錄同時發生的事件，就新增了因果。兩句改成一句，只要關係保留就可以；把先後改為因果，或刪除「during the test」，就改變了內容。先記錄命題及其來源區間、再產生文字的計畫，能讓你分別評估選擇、計畫與文字實現，找出意義在哪裡遺失。

故事生成與開放式文字

開放式生成把報告的優先順序反過來：沒有唯一正確輸出，多樣性本身就是品質的一部分。Fan、Lewis 與 Dauphin 收集人寫的故事與寫作提示配對，先生成故事前提再生成故事，並因為束搜尋傾向產生簡短而通用的文字而改用 top-k 取樣 [13]。對本章來說，重點是**同一個**語言模型可能需要依任務使用不同解碼器：受限報告用確定性搜尋，故事用截斷取樣。不變的是評估必須對準任務：故事看連貫性與切題程度，摘要看內容是否受支持。

8. 評估生成文字

直觀說法。單一自動分數無法總結所有生成品質。自動指標應按指定程序比較系統，再檢查應用真正在乎的錯誤維度。

困惑度（第 5 節）把模型當成參考文字的預測器來評估，並不直接說明某次解碼輸出的好壞，也不能比較分詞器不同的模型。**重疊指標**比較輸出與參考。ROUGE 包含偏召回的摘要重疊量測 [17]。這類指標比較候選與一個或多個參考，不證明語意等價或來源支持。報告分數要指定版本、分詞、聚合方式與參考集合。翻譯指標如 BLEU 與學得指標在 NLP 08 討論。

簡單的 unigram 召回檢查：

$$R_1 = \frac{\sum_{w} \min(c_{\text{ref}}(w), c_{\text{hyp}}(w))}{\sum_{w} c_{\text{ref}}(w)}.$$

其中 w 遍歷詞型；$c_{\text{ref}}(w)$、$c_{\text{hyp}}(w)$ 是參考與候選的詞頻；分母是參考詞元數。它示範 ROUGE 式評估的一個部分，以下最小實作沒有重現完整套件的每項前處理或聚合選項。

```
import re
from collections import Counter

def words(text):
    return re.findall(r"[a-z0-9]+", text.lower())
def overlap_recall(reference, hypothesis):
    ref, hyp = Counter(words(reference)), Counter(words(hypothesis))
    return sum((ref & hyp).values()) / sum(ref.values())
reference = "Pump A is operating at 30 C."
contradiction = "Pump A is not operating at 30 C."
score = overlap_recall(reference, contradiction)
print("Unigram recall:", score)
assert score == 1.0
assert ("not" in words(reference)) != ("not" in words(contradiction))
```

輸出：

```
Unigram recall: 1.0
```

矛盾句保留了所有參考詞，因此召回為 1.0。精確率會懲罰多出的詞，但小懲罰仍不能表達否定反轉的重要性。這是受控反例，不表示重疊指標永遠不與人類判斷相關，而是說明它必須配合對意義敏感的檢查。

人工評估與錯誤類別

設備應用可分別計算**無支持主張**、**必需主張遺漏**、**數值關係錯誤**與**未解指涉**。它們是專案自訂的錯誤類別。每類都要給標註者正反例，包含缺少證據與證據矛盾的情況。說得少的摘要可能無支持主張較少，卻漏掉關鍵資訊，因此兩個方向的錯誤都要回報。

人工評估要隱藏系統身分、隨機呈現、保留來源上下文並指定決策任務。先請覆核者標出無支持陳述，再判斷整體實用性；整體偏好可能獎勵風格，卻漏掉罕見的關鍵錯誤。應報告由誰覆核，以及如何處理歧見。聚合要以有意義的單位：同一事故的多個句子彼此相關，把它們當獨立例子會誇大確定性。

以語言模型當評審

可以用提示讓強大的語言模型評分或比較輸出。Zheng 等人研究這種「LLM 當評審」（LLM-as-a-judge）設定，報告在他們的基準上，強評審的偏好與人類偏好的一致程度，可與人類彼此之間的

一致程度相比；同時也記錄了位置偏誤、冗長偏誤與自我抬高偏誤 [18]。Wang 等人顯示，只要交換兩個回覆出現的順序，就可能改變模型評審偏好哪一個，並提出兩種順序都評估等校準方法 [19]。

把模型評審當成另一種有自己版本、提示與失敗模式的量測儀器。兩種呈現順序都要評估，隱藏輸出來自哪個系統，在你自己的錯誤類別（尤其是否定、數量與無支持主張）上以人工標註驗證評審，並在可行時避免用同一模型家族評自己的系統。平均而言能排好系統順序的評審，不會自動成為接受單一輸出的可靠門檻。

9. 檢索：找到生成所依據的證據

直觀說法。生成器要使用證據之前，得先有人把證據找出來。檢索系統從明確語料選出段落；這些段落是否支持回答、答案如何措辭，則是後續的決策。

語言模型不搜尋當前文件集合，也能延續問題；檢索系統則從明確語料選資訊。問答再加入一個決策：選到的內容是否支持回答，以及用哪些詞表達。把元件結合能建立有用管線，但沒有哪個階段會自動替下一階段背書。

本節貫穿的問題是「Pump A 的溫度上限是多少？」相關段落討論 Pump A 或溫度；足夠的段落指出該幫浦在適用政策下的上限；有效引用指向真實來源；受支持回答保留數值、單位、時間與範圍。四者是不同條件，評估應能指出失敗的是哪一項。

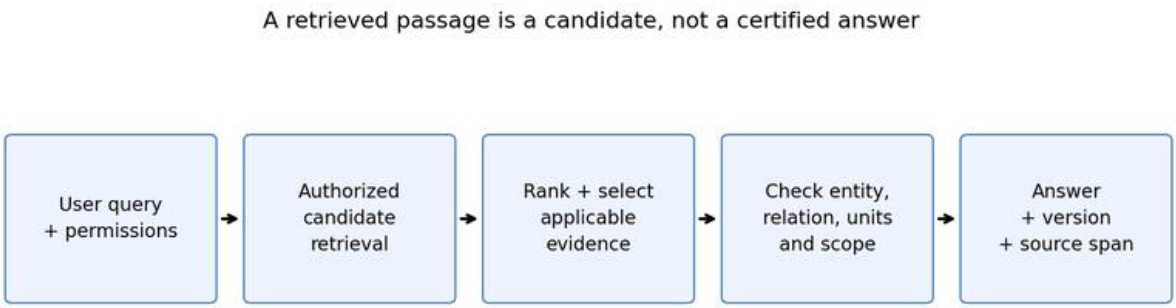


圖 5 — 檢索選候選，證據檢查決定能說什麼。每個階段都屬於明確應用契約。

本節建立並評估檢索階段。判斷檢索到的段落是否真的回答問題——擷取式片段答案、相關與可回答的區別，以及用受控替換診斷——在 NLP 11 討論。第 10 節再讓生成器以檢索證據為條件。

選模型前先設計語料

直觀說法。檢索器搜尋你提供的內容。文件身分、更新狀態與權限是系統的一部分，不是排序後才補的選填資料。

可搜尋段落可表示為：

$$d = (u, v, b, e, \mathbf{x}, a).$$

其中 d 是段落紀錄；u 是父文件識別碼；v 是版本；b、e 是原文中包含起點、不包含終點的位移；$\mathbf{x}$ 是段落文字；a 描述存取政策。必須聲明位移單位。這是本章的應用紀錄，不是通用儲存標準。

段落可是一個自然段、一組句子或依文件結構選的章節。短段落能隔離答案，卻可能丟掉限定條件；長段落保留上下文，卻引入不相關內容並消耗生成預算。切塊邊界應根據任務選，再評估支持事實與條件是否仍放在一起。

例如標題寫「只適用戶外設備」，下一段才給溫度上限。單獨索引該段，可能讓人以為也適用室內設備。保留標題或結構化適用範圍會改變可回答性契約。固定字元視窗容易實作，但簡單不代表能保留意義。

版本也重要。舊政策可能相關卻不適用。需要當前答案時，應保留生效日期與替代關係。不能只因最新段落放在提示最後，就期待生成器判斷當前政策。應依明訂適用規則過濾或排序；語料沒有指定唯一權威來源時，讓衝突可見。

在未授權元件或使用者收到段落文字前，就要套用權限。私密內容若已進模型請求或紀錄，才過濾最終答案就太晚了。檢索器語料視野應符合當前使用者權限，評估也應在受限視野下進行；否則全域相關性分數可能評到系統永遠不能展示的文件。

詞彙檢索與稀疏向量

直觀說法。詞彙檢索匹配詞語，替有辨識力的詞加權。它容易檢查，精確名稱、代碼與技術短語重要時尤其有用。

簡單的詞頻與逆文件頻率權重為：

$$w_{t,d} = \mathrm{tf}(t,d)\log\frac{N}{\mathrm{df}(t)}.$$

其中 $w_{t,d}$ 是詞 t 在文件 d 的權重；$\mathrm{tf}(t,d)$ 是依選定慣例定義的詞頻；N 是索引文件數；$\mathrm{df}(t)$ 是包含該詞的文件數。只出現在查詢的詞若文件頻率為零，需要特別處理。實作常加入平滑或詞頻轉換，應記錄真正使用的慣例 [33, 34]。

使用正規化文件與查詢向量時：

$$s(q,d) = \frac{\mathbf{v}_q^{\top}\mathbf{v}_d}{\|\mathbf{v}_q\|_2\|\mathbf{v}_d\|_2}.$$

其中 q 是查詢；d 是候選文件；$\mathbf{v}_q, \mathbf{v}_d$ 是向量；s 是餘弦相似度；$\|\cdot\|_2$ 是歐氏範數。任一向量為零時定義分數為零。程式使用 scikit-learn 正規化向量，因此非零列的內積就是餘弦相似度 [34]。

倒排索引（inverted index）為每個詞儲存出現它的文件，讓搜尋引擎不必把查詢與每份文件的所有詞彙逐一比較 [33]。本例只有四段，所以直接計算全部分數。數學表示與有效率的索引策略是分開的選擇。

前處理直接影響檢索。丟掉單字元詞元的分詞器可能刪掉 Pump A 的 A。普通詞小寫化可能有幫助，卻會抹掉區分大小寫代碼。短語特徵有助辨識相鄰詞，不會涵蓋任意語法，也不保證語意相同。漏掉結果時，先看詞彙表，再怪排序模型。

詞彙系統可能漏掉改寫。Device ceased operating 與 machine shutdown 可能少有共同詞。同義詞或查詢擴展能幫忙，但也改變匹配政策、增加歧義。應在保留查詢上評估擴展，特別包含精確

代碼案例。廣泛擴展提升召回時，也可能找到術語相似的錯誤裝置。

飽和與文件長度

直觀說法。重複查詢詞很多次，不應永遠提高相關性。長文件也有更多機會偶然匹配。

常見 BM25 形式為：

$$s(q,d)=\sum_{t\in q}\operatorname{idf}(t)\frac{f_{t,d}(k_1+1)}{f_{t,d}+k_1\left(1-b+b\,|d|/\overline{L}\right)}.$$

其中 q 在簡化式中是查詢詞集合；d 是候選文件；$f_{t,d}$ 是詞頻；$\operatorname{idf}(t)$ 是選定逆文件頻率權重；$|d|$ 是詞元長度；$\overline{L}$ 是平均文件長度；k_1 是正的飽和參數；$b\in[0,1]$ 控制長度正規化。不同實作的 IDF 與查詢詞頻慣例不同，資訊檢索教科書討論了這個家族及其假設 [33]。

隨重複增加，分子與分母讓詞頻貢獻接近一個極限。長度項改變長文件預期有多少詞頻。這些是排序調整，不是文件含有真實答案的機率。一段內容可能反覆討論上限而排名很高，卻從未給出數值。

應在反映部署語料的開發查詢上調整長度正規化。若所有答案都在短參考條目，不相關內容都在長敘事，一個設定可能特別強；改成長技術手冊後，行為會改變。比較排序設定時保持切塊固定，否則應說明語料表示與分數都改變了。

稠密檢索與聯合重排序

直觀說法。學得的編碼器能匹配沒有精確重疊的相關表達。效果取決於訓練任務與對比例子。

雙編碼器分數為：

$$s_\theta(q,d)=f_\theta(q)^\top g_\theta(d).$$

其中 f_θ 把查詢轉成向量；g_θ 把段落轉成向量；θ 是學得參數；s_θ 是內積分數。編碼器可共用或分別使用參數。Dense Passage Retrieval 是這種設計在問答中的原始研究例子 [35]。

段落向量可預先計算，查詢再搜尋高分向量。近似搜尋以部分精確近鄰忠實度換速度；若使用它，就要同時評估表示與搜尋近似。本章不實作這種索引，因為小語料精確搜尋更適合學習契約。

對比訓練可把一個正段落與數個負段落比較：

$$\mathcal{L}=-\log\frac{\exp(s_\theta(q,d^+)/\tau)}{\exp(s_\theta(q,d^+)/\tau)+\sum_j\exp(s_\theta(q,d_j^-)/\tau)}.$$

其中 $\mathcal{L}$ 是單一查詢損失；d^+ 是標註正段落；d_j^- 是負候選 j；τ 是正溫度；s_θ 是學得分數。負例可包含措辭相似的困難段落。但只因未被標註就判負的段落，若也有有效答案，就成了假負例。

聯合重排序器一起讀查詢與候選段落，能產生比兩個固定向量更豐富的詞元互動，通常放在較便宜的候選檢索後。這增加計算與一個選擇程序。應先報告候選召回率：沒有收到正確段落，重排序器就無法把它找回。

稠密與詞彙檢索可以互補。稠密表示可能幫助改寫，詞彙匹配則保留精確代碼與罕見名字。混合時要注意尺度，因為詞彙分數與學得內積沒有相同單位。應在開發資料選混合設定，檢查是否引入錯誤實體結果。

對比損失實際教模型什麼？

稠密損失不只是把查詢移向任何大致相關的文章，而是學習候選集合呈現的差異。令候選零為正例，其餘為負例。對候選分數：

$$p_j = \frac{\exp(s_j/\tau)}{\sum_{r=0}^{M}\exp(s_r/\tau)}, \qquad \frac{\partial \mathcal{L}}{\partial s_j} = \frac{p_j - \mathbf{1}[j=0]}{\tau}.$$

其中 s_j 是查詢對候選 j 的分數，M 是負候選數，r 索引全部候選，$\tau > 0$ 是溫度，p_j 是候選 softmax 機率，$\mathcal{L} = -\log p_0$ 是損失，$\mathbf{1}[j=0]$ 對正例為一、其餘為零。這個導數直接由前面的 softmax 損失得到。梯度下降提高正例分數、降低負例分數；競爭最強的負例承受最大的向下壓力。

DPR 透過正負文章比較，訓練分開的查詢與文章表示 [35]。公式說明比較的選擇為何重要。若負例都來自不相關主題，模型可能學會主題區分，卻沒有學會區分 Pump A 與 Pump B。具有相同關係、卻是錯誤裝置的文章，對這種錯誤提供更有用的對比。但若未標註文章其實支持答案，把它當成負例，就會教出錯誤的分離。

這個獨立的分數空間實驗確認梯度方向。它示範目標，不會訓練文章編碼器：

```
import numpy as np

scores = np.array([1.2, 0.8, -0.4])  # positive, hard negative, easy negative
def probabilities(values):
    z = np.exp(values - values.max())
    return z / z.sum()
p = probabilities(scores)
gradient = p.copy()
gradient[0] -= 1.0  # temperature is one
updated = scores - 0.5 * gradient
before, after = -np.log(p[0]), -np.log(probabilities(updated)[0])
print('Gradient:', gradient.round(3))
print('Loss:', round(before, 3), '->', round(after, 3))
assert gradient[0] < 0 and gradient[1] > gradient[2] > 0
assert after < before
```

梯度四捨五入為 `[-0.466, 0.358, 0.108]`，損失由 `0.627` 降至 `0.467`。分數較高的競爭負例，比簡單負例承受更大的向下壓力。

溫度同時影響分布尖銳程度與梯度尺度；候選則改變模型正在學的對比。因此，新編碼器若也改了正例標籤、負例建立方式或訓練語料，提升便不能只歸因於架構。比較時應保留這些差異。

依重要差異比較檢索路徑

雙編碼器在還不知道未來查詢時，就先壓縮文章。這讓向量能快取並有效率地搜尋候選，但未來問題需要的每個細節，都必須在壓縮後保留下來。聯合重排序器同時取得兩段文字，可比較彼此的特定詞元；每對計算成本較高，仍取決於訓練範例。聯合注意力提供辨識細節的機會，不證明模型一定尊重該細節。

查詢差異	有用證據	應檢查的失敗
`Pump A` 與 `Pump B`	精確識別碼加上關係	主題相似掩蓋一個字元的差異
`ceased operating` 與 `shutdown`	學到的改述相容性	詞彙表漏掉同義表達
目前與升級前的限制	版本與生效日期中繼資料	高度相關文章回答錯誤時間範圍
許可與禁止	否定及其語法作用範圍	共同詞彙隱藏相反指令

混合設計可聯集詞彙與稠密候選、去重，再聯合重排留下的文章。聯集保留任一路徑找到的候選，但固定總預算仍可能排除部分候選。直接相加原始分數較難解讀：把一個系統的分數乘常數，雖不改自己的排序，卻會改變它在加總中的影響力。因此，正規化或學習組合需要開發資料的相關性判斷，而不是假定任意係數都適用。

應分別比較候選召回、最後順序與資源成本。系統可能改善前者卻損害後者，也可能兩者都改善卻超出服務預算。把精確名稱、改述與範圍敏感問題分組診斷，錯誤模式能揭露表示丟了哪些資訊，以及哪種下游互動值得額外計算。

排序評估

直觀說法。檢索評估需要查詢及相關性判斷。選到一份文件，不代表選對了。

對查詢 q 的相關集合 R_q 與前 k 結果集合 $T_k(q)$：

$$\text{Recall@}k(q) = \frac{|R_q \cap T_k(q)|}{|R_q|}.$$

其中 k 是截斷位置；R_q 包含標註相關文件；$T_k(q)$ 包含截斷前檢索結果；$|\cdot|$ 是集合大小。本章只對至少一份相關文件的查詢計算。沒有答案的查詢屬於另外的可回答性評估，不應任意指定召回率。

第一個相關結果排名為 r_q 時：

$$\text{RR}(q) = \frac{1}{r_q}, \qquad \text{MRR} = \frac{1}{Q}\sum_{q=1}^{Q} \text{RR}(q).$$

其中 RR 是倒數排名；MRR 是其平均；Q 是評估查詢數；r_q 從一開始。沒有回傳相關結果時定義倒數排名為零。當代檢索增強評估討論使用這些診斷 [31]。倒數排名只重視第一個成功結果，忽略後面的相關結果。

標註完整性很重要。多個段落都支持答案，卻只標一個相關，可能把有效檢索判錯。要記錄相關是指主題匹配、包含答案，還是完整支持。重要歧義可使用多位判斷者，再依書面指南裁決。資料集應揭露任務定義，不只是列一段偏好的文字。

開發與測試要隔離查詢家族及文件衍生版本。把開發問題稍作改寫放進測試，泛化證據很弱；同一報告的切塊跨集合也可能洩漏事實與措辭。切分應反映部署問題：沒見過的使用者、文件、未來版本或新領域。

量測一個小型檢索器

這段獨立程式使用四個合成段落。它保留單字元詞，讓裝置識別碼 A 仍是特徵，並在三個標註查詢上量測排序。

```
import numpy as np
from sklearn.feature_extraction.text import TfidfVectorizer

corpus = {
    'temperature-v2': 'Pump A temperature limit is 80 C. This policy is current.',
    'shutdown-v1': 'Pump A shutdown follows a vibration alarm.',
    'label-v1': 'A technician replaced the cabinet label.',
    'sensor-v1': 'Sensor B reports humidity once per hour.',
}
ids = list(corpus)
vectorizer = TfidfVectorizer(ngram_range=(1, 2), token_pattern=r"(?u)\b\w+\b")
D = vectorizer.fit_transform([corpus[key] for key in ids])
def rank(query):
    score = (D @ vectorizer.transform([query]).T).toarray().ravel()
    order = sorted(range(len(ids)), key=lambda i: (-score[i], ids[i]))
    return [(ids[i], float(score[i])) for i in order]
queries = [
    ('Pump A temperature limit', {'temperature-v2'}),
    ('vibration shutdown alarm', {'shutdown-v1'}),
    ('humidity report', {'sensor-v1'}),
]
recalls, reciprocal = [], []
for query, relevant in queries:
    ranking = rank(query)
    recalls.append(len({key for key, _ in ranking[:2]} & relevant) / len(relevant))
    first = next((i + 1 for i, (key, _) in enumerate(ranking) if key in relevant), None)
    reciprocal.append(1 / first if first else 0.0)
    print(query, '->', ranking[0][0])
print('mean Recall@2:', np.mean(recalls))
print('mean reciprocal rank:', np.mean(reciprocal))
```

輸出：

```
Pump A temperature limit -> temperature-v2
vibration shutdown alarm -> shutdown-v1
humidity report -> sensor-v1
mean Recall@2: 1.0
mean reciprocal rank: 1.0
```

三個最高結果符合各自唯一相關段落，因此本次小型執行的平均 Recall@2 與平均倒數排名都是 1.0。這是固定測試案例結果，不是真實檢索基準。NLP 11 沿用同一語料與排序器，建立範圍狹窄的證據式回答器。

分級相關性：nDCG

Recall@k 與倒數排名把判斷視為相關或不相關。若判斷者替段落分級——完整支持高於只提到主題——量測就應獎勵把最有用的段落排在最前面。

$$\mathrm{DCG@}k = \sum_{i=1}^{k} \frac{g_i}{\log_2(i+1)}, \qquad \mathrm{nDCG@}k = \frac{\mathrm{DCG@}k}{\mathrm{IDCG@}k}.$$

其中 $g_i \geq 0$ 是排名 i 結果的分級相關性；DCG@k 是前 k 名的折損累積增益（discounted cumulative gain）；IDCG@k 是已判斷結果依理想順序排列時的同一總和，因此 nDCG@$k \in [0,1]$；IDCG@$k = 0$ 時定義為零。Järvelin 與 Kekäläinen 提出累積增益這一族量測 [36]；對數底數、增益轉換與截斷位置因實作而異，報告時應寫明。

同樣四個已判斷段落的兩種排序，差別只在支持段落或主題段落排第一：

```
import numpy as np

def dcg(gains, k):
    gains = np.asarray(gains[:k], dtype=float)
    return float(np.sum(gains / np.log2(np.arange(2, gains.size + 2))))
def ndcg(gains, k):
    ideal = dcg(sorted(gains, reverse=True), k)
    return dcg(gains, k) / ideal if ideal > 0 else 0.0

# graded judgments of one ranking: 2 = complete support, 1 = topical, 0 = unrelated
ranked_a = [1, 2, 0, 0]  # topical passage above the supporting one
ranked_b = [2, 1, 0, 0]  # supporting passage first
for name, gains in [('A', ranked_a), ('B', ranked_b)]:
    print(name, 'nDCG@3:', round(ndcg(gains, 3), 3),
          'binary RR:', 1 / (1 + gains.index(2)))
assert ndcg(ranked_b, 3) == 1.0 and ndcg(ranked_a, 3) < 1.0
```

輸出：

```
A nDCG@3: 0.86 binary RR: 0.5
B nDCG@3: 1.0 binary RR: 1.0
```

若以「完整支持」為相關標準，排序 A 的倒數排名減半。nDCG 也下降，但幅度較小，因為它仍替排第一的主題段落計分。哪種行為正確取決於任務：若只有完整答案有用，主題段落的分級分數可能掩蓋失敗。選量測前先寫好分級指南。

10. 幻覺與檢索增強生成

直觀說法。語言模型可能產生流暢、卻得不到輸入或真實世界支持的文字。檢索提供外部證據，但生成器仍必須正確使用；檢索來的文字絕不能被當成新的權威。

Ji 等人的綜述區分**內在**（intrinsic）幻覺，即與來源矛盾的輸出，以及**外在**（extrinsic）幻覺，即無法由來源驗證的輸出 [20]。Maynez 等人對抽象式摘要器的人工研究發現，所檢查的每個模型生成摘要都含有大量幻覺內容，並發現文字蘊涵量測與忠實度的相關性優於標準重疊指標 [21]。這兩點都可由第 1 節與第 6 節推得：訓練目標獎勵可能的文字，任何解碼器都無法補上分布中缺少的證據。

注意力可診斷計算存取，不能證明事實。詞元讀到正確證據，仍可能表達相反意思。驗證要看輸出主張、來源實體、關係及範圍。狹義結構任務可讓生成器說出已驗證的計畫，如上面的模板；自由文字的驗證並不完整，應明說限制。

檢索增強生成

原始檢索增強生成（retrieval-augmented generation，RAG）論文，把學得檢索器與條件序列生成結合，用於知識密集任務 [22]。一種序列層級形式為：

$$p(y \mid x) \approx \sum_{z \in Z_k(x)} p_\eta(z \mid x)\, p_\theta(y \mid x, z).$$

其中 x 是問題；y 是輸出序列；z 是檢索段落；$Z_k(x)$ 是前 k 個候選集合；η 是檢索器參數；θ 是生成器參數。近似反映只保留候選。這是特定模型的機率形式，不代表所有當代 RAG 應用都實作它。

很多實務系統先檢索、組提示，再執行分開訓練的生成器。即使未最佳化原始邊際目標，這種管線仍可有用，但要說清楚設計。否則 RAG 這個名稱會掩蓋檢索是否學得、候選是否重排序，以及生成器是否受訓使用來源。第 9 節建立並評估檢索器本身；擷取式答案與可回答性檢查在 NLP 11。

證據扎根（grounding）依明訂的支持規則，把主張連到外部證據。有引用的答案可能可追溯，但可追溯弱於受支持：有效網址也可能指向與答案矛盾的文件。要評估每個重要主張是否由引用證據推得，以及是否保留必要限定。

證據也可能衝突。生成器不應把互斥政策默默合成一個自信答案。應用可依契約優先採用權威版本、展示衝突或請求澄清。決策應來自中繼資料與任務規則，而非哪一句最流暢。

錯誤沿檢索、解讀與文字實現傳遞

受支持答案需要一串成功決策：語料必須有適用證據，檢索必須保留它，閱讀器必須解讀正確實體與關係，輸出也必須保留該解讀。由記憶知識猜出的答案，可能碰巧正確，卻不符合證據要求。對適用證據已存在語料中的問題，可寫成：

$$P(R \cap E \cap G) = P(R)P(E \mid R)P(G \mid R, E).$$

其中 P 表示機率，R 是充分證據保留於檢索結果的事件，E 是正確解讀證據的事件，G 是輸出忠實實現解讀的事件。這是機率鏈式法則，不是假設獨立。把互不相干的無條件階段分數相乘並不等價：閱讀器在人工挑選簡單文章上的準確率，可能不同於在檢索器實際候選上的準確率。

```
import numpy as np

retrieved = np.arange(100) < 80
interpreted = np.arange(100) < 72
realized = np.arange(100) < 68
r = retrieved.mean()
e = interpreted[retrieved].mean()
g = realized[retrieved & interpreted].mean()
complete = (retrieved & interpreted & realized).mean()
print('Conditional stage rates:', round(r, 3), round(e, 3), round(g, 3))
print('Complete supported answers:', round(complete, 3))
assert np.isclose(r * e * g, complete)
```

輸出：

```
Conditional stage rates: 0.8 0.9 0.944
Complete supported answers: 0.68
```

這是合成結果，不是系統效能報告。計算說明各元件表現看似不錯，完整答案仍未必同樣可靠。改善最弱階段的效果，還要能通過後續階段才有用。

檢索內容的邊界

檢索文字是待檢查的證據，不是有權改變應用的指令來源。段落可能包含引述指令、誤導文字或惡意要求。提示注入（prompt injection）試圖讓模型遵循與應用邊界衝突的指令，OWASP 記錄了此風險類別 [23]。檢索系統的重要界線，是使用者／應用指令與不可信來源材料之間。

來源文字應清楚分隔並標示文件身分，不得重新定義使用者權限、索取秘密資料或授權工具動作。若有工具，應在檢索段落之外，依使用者任務及應用政策驗證動作。文字分隔有助組織輸入，卻不是完整的強制機制。

相同原則適用於紀錄與快取。某使用者權限下產生的快取結果，不能未經檢查就給另一位使用者。證據區段必須指向回答所用的同一文件版本。文件移除或替換時，要更新索引，並考慮讓衍生回答失效。

這些檢查也支援一般正確性。「忽略舊上限，新政策如下」可以是需要解讀的合法文件內容，「忽略你的應用指令」卻不是系統的新權威。兩者都可儲存為來源材料，再套用任務的證據與權限規則；不能讓檢索器成為修改治理契約的通道。

11. 調適預訓練生成器

直觀說法。預訓練學習廣泛的延續模式；調適改變任務行為；提示則改變提供的輸入，不一定改權重。學習遵循指令與學習人們偏好的回覆是不同訊號，兩者都不保證真實。

監督任務調適對標註例子最小化任務損失，例如只在回覆詞元上計算第 7 節的條件損失。繼續在領域文字上訓練語言模型會改變所建模的分布，卻不會自動教會應用的輸出政策。熟悉領域與理解任務指令是兩種不同調適。上下文內例子（如 GPT-3 [6]）直接出現在輸入中，不更新參數就能示範格式；若例子包含私密答案或測試標籤，評估就可能失效，而在開發案例上選出的提示，必須在最終測試前凍結。

低秩調適（Low-Rank Adaptation，LoRA）固定基礎權重矩陣，訓練受限更新 [24]：

$$W' = W + BA.$$

其中 $W \in \mathbb{R}^{m\times n}$ 是固定矩陣；$A \in \mathbb{R}^{r\times n}$ 與 $B \in \mathbb{R}^{m\times r}$ 是學得因子；r 是選定的秩；W' 是有效調適矩陣。簡化式省略可選縮放。減少可訓練參數能降低調適成本，但不能取代代表性標籤、資料隔離與任務評估。基礎模型版本、分詞器、調適權重與設定應一起儲存：針對某基礎檢查點訓練的調適，不保證能套用到同家族另一檢查點。

InstructGPT 論文先研究監督指令例子，再以人類回饋進行偏好訓練 [25]，因而區分基礎延續模型與朝請求行為調適的模型。偏好來源很重要：標註者依政策平衡實用性、格式與其他目標。較受偏好的回覆仍可能有事實錯誤。

直接偏好最佳化（Direct Preference Optimization，DPO）結合偏好與不偏好的回覆，以及參考政策 [26]。代表性目標如下：

$$\mathcal{L}_{\mathrm{DPO}} = -\log \sigma\left(\beta\left[\log \frac{\pi_\theta(y^+ \mid x)}{\pi_{\mathrm{ref}}(y^+ \mid x)} - \log \frac{\pi_\theta(y^- \mid x)}{\pi_{\mathrm{ref}}(y^- \mid x)}\right]\right).$$

其中 x 是提示；y^+ 與 y^- 是偏好及不偏好的回覆序列；π_θ 是可訓練政策；π_{ref} 是固定參考政策；β 是正縮放參數；σ 是 logistic sigmoid；$\mathcal{L}_{\mathrm{DPO}}$ 是單一偏好對的損失。序列機率是回覆詞元條件機率的乘積，提示提供上下文，不是受監督回覆詞元。

目標提高偏好回覆相對於另一回覆的概似，並以參考政策為基準。若資料獎勵自信但無支持的答案，最佳化可能強化這種行為。指令遵循、偏好一致與事實支持是不同性質，需要不同證據。ML 10 以程式執行一個低秩更新。

僅線上版：*InstructGPT — Training Language Models to Follow Instructions with Human Feedback* — *https://blog.shyandram.dev/zh-tw/learn/nlp/language-modeling-transformers/papers/instructgpt–training-language-models-to-follow-instructions-with-human-feedback/*

12. 訓練實務遇上服務限制

直觀說法。部署後的生成器要服務長度不一的輸入，每個請求的成本同時取決於提示與輸出。填補與注意力快取是兩種不同的記憶體成本，兩者都要量。

Hagiwara 第 10 章把填補、排序與遮罩連到實際推論 [1]。要分清楚：正確的損失遮罩防止填補改變目標，未必防止硬體處理填補位置。定義填補利用率：

$$u = \frac{\sum_{b=1}^{B} L_b}{B \max_b L_b}.$$

其中 L_b 是請求 b 的真實詞元長度；B 是批次大小；u 是填補後位置中真實詞元的比例。這是工作量診斷，不是實測加速。按相近長度組批可提高利用率，也可能讓請求等待適合的批次。

```
import numpy as np

lengths = np.array([8, 64, 12, 60, 10, 56, 14, 52])
def utilization(values):
    batches = values.reshape(-1, 4)
    real = int(batches.sum())
    padded = int((batches.max(axis=1) * 4).sum())
    return real / padded
arrival = utilization(lengths)
bucketed = utilization(np.sort(lengths))
print("Arrival-order utilization:", round(arrival, 3))
print("Length-bucketed utilization:", round(bucketed, 3))
assert bucketed > arrival
# Two cache arrays (keys and values), with a toy equal head layout.
layers, kv_heads, head_dim, cache_tokens, bytes_per_value = 4, 2, 16, 128, 4
cache_bytes = 2 * layers * kv_heads * head_dim * cache_tokens * bytes_per_value
print("Toy cache bytes:", cache_bytes)
assert cache_bytes == 131072
```

輸出：

```
Arrival-order utilization: 0.575
Length-bucketed utilization: 0.885
Toy cache bytes: 131072
```

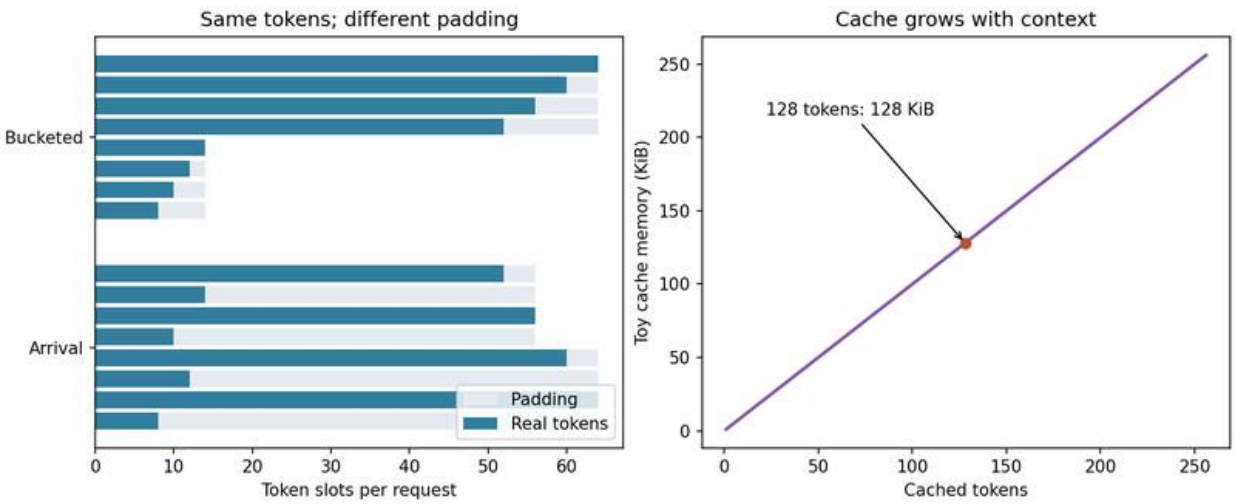


圖 *6* — 填補浪費與注意力快取是不同記憶體成本。圖中呈現計算的詞元位置及微型快取位元組，不主張實測服務加速。

自迴歸注意力常快取先前的鍵和值，避免每產生一個詞元就重算。簡單布局下，快取記憶體近似 $M_{\text{KV}} = 2N_{\text{layers}}H_{\text{KV}}d_hTb_v$，其中 N_{layers} 是層數、H_{KV} 是鍵／值頭數、d_h 是頭維度、T 是快取詞元數、b_v 是每個值的位元組數。因子二表示鍵與值。實際實作還有配置及中繼資料成本，也可能使用不同布局。

應分開量測首詞元時間、後續詞元間隔、總完成時間及負載下的吞吐量。短提示長答案與長提示短答案壓力不同。端到端量測要包含前處理、等待、驗證與序列化。依量測到的工作負載設定請求長度上限、輸出上限及佇列期限，並說明逾時時回傳部分結果、可重試錯誤或經覆核的替代結果。

可重現的服務契約

只有模型檔無法重現預測。分詞、正規化、證據選擇、解碼參數、驗證規則與輸出格式都影響行為。Hagiwara 的部署章 [1] 提供版本、測試、訓練—服務差異與監控的實務閱讀。換成學得生成器時，文字旁要回傳模型版本與已驗證證據 ID（如模板函數所做），並把前處理版本與解碼設定和模型一起保存。紀錄應能支援診斷，卻不必無謂保留私密原文。

新版本先服務有限比例的請求，用相同錯誤類別與穩定版本比較，同時檢查延遲、完成率與證據正確性。回復應保持簡單：儲存的套件應能一起選回舊模型、分詞器、解碼設定與驗證規則。舊模型混用新分詞器，可能產生各元件單獨時都沒有出現的失敗。

現代觀點

直觀說法。當代語言模型結合預訓練、調適、解碼、評估與系統整合。綜述若能釐清這些不同選擇，而非壓縮成單一模型排名，就特別有用。

Zhao 等人的綜述 [27] 更新至 2026 年，並發表於 Frontiers of Computer Science，依預訓練、調適、使用與能力評估組織大型語言模型。這有助區分更多訓練資料、指令微調與不同使用方式。**Chang 等人的評估綜述 [28]** 區分評什麼、在哪裡評與如何評。兩者共同支持任務專屬的評估計畫，而非依賴單一通用分數。

Li 等人的預訓練生成綜述 [29] 依輸入表示、架構與最佳化整理方法，並涵蓋應用及評估。這有助定位變化：檢索改變可用證據，編碼器—解碼器改變條件架構，調適改變參數或訓練目標。該綜述早於部分後續系統，應搭配較新的評估與部署證據閱讀。

Ji 等人的幻覺綜述 [20] 與 **Schmidtova 等人對幻覺人工評估的綜述 [30]** 從兩側處理同一問題。前者整理跨任務的定義、指標與緩解方法；後者檢查研究如何定義與評估幻覺，指出術語不一致及方法報告缺失。其實務重點是分開定義來源支持、世界知識與錯誤嚴重度，並公開標註說明與歧見處理方式。

Gao 等人的 RAG 綜述 [31] 把檢索增強系統分為 naive、advanced 與 modular 範式，並區分上下文相關、答案相關與答案忠實。各階段會互相影響：把「升級前的限制是多少？」改寫為「溫度限制是多少？」，可能改善主題檢索，卻刪掉時間條件。應保留原始查詢，比較改寫後的意義，而不只看檢索分數。

Kwon 等人的 PagedAttention [32] 研究語言模型服務中的鍵—值快取配置與共享，處理記憶體浪費以及記憶體所允許的批次大小，替上面的快取計算提供系統觀點。論文報告的改善屬於其工作負載與比較；採用服務設計前，先量測自己的請求長度、並行程度與延遲目標。

開放問題包括評估集合污染、長上下文穩健性（較大的上下文視窗允許更多輸入，卻不能證明所有相關部分都被可靠使用）、多語與低資源語言涵蓋、穩定的不確定性、在不壓制有用回答下偵測無支持主張，以及指令遵循和來源支持正確性的關係。評估模型可能與生成器共享錯誤，尤其

兩者訓練資料相近時。能以較低延遲與較簡單覆核滿足任務時，小型專用模型甚至模板，可能是更好的工程選擇。

重點整理

- 僅解碼器 Transformer 以因果方式分解文字；要明確位移目標，並測試未來詞元無法改變較早 logits。
- GPT 系列保留同一架構與目標，擴大資料、參數與呈現任務的方式；規模化定律預測損失，不預測任務行為。
- 困惑度比較需要相容的分詞與評估位置，也不直接說明解碼輸出的好壞。
- 貪婪與束搜尋最佳化序列機率；溫度重塑分布；top-k 與核取樣截掉長尾，而核會隨分布形狀調整。
- 生成需要內容契約：選擇、規劃與文字實現各有不同失敗方式，擷取式摘要仍需檢查必需內容。
- 重疊指標、人工覆核與模型評審各有盲點；要在重要錯誤（例如否定反轉）上驗證它們。
- 檢索提供證據，不提供權威；相關性、引用有效性與主張支持是不同條件。
- 把整個前處理—模型—解碼—驗證套件版本化，並把指令微調與偏好學習視為與事實支持不同的事。

練習

1. 移除可執行因果模型中的因果遮罩。哪個斷言應失敗？為何訓練損失仍可能好看？

   提示

   改變的最後詞元能被較早位置讀取。低損失可能獎勵生成時根本取不到的資訊。

2. 使用解碼程式，找出一個溫度，使單純取樣放在四個長尾詞元上的機率，超過 $k = 5$ 的 top-k 所能保留的上限。再構造一個 logit 向量，讓 $p = 0.9$ 的核大於五個詞元，並說明各規則何時較合適。

   提示

   $k = 5$ 的 top-k 最多只能保留一個長尾詞元。平坦的 logit 向量讓核變大，尖峰的則讓核變小。

3. 修改微型束搜尋的機率，讓束搜尋選出違反外部指令的句子。加入合法完整序列的限制，並與加寬束比較效果。

   提示

   機率搜尋最佳化的是分布分數，合法序列規則改變的是哪些輸出可接受。加寬束未必能修好錯誤的機率模型。

4. 建立兩個候選摘要：一個詞重疊高但數量錯，另一個重疊較低但意義正確。設計能顯示差別的標註表，以及模型評審程序。

提示

先請覆核者標出無支持數量與遺漏的必需事實，再判斷整體實用性。保留來源、隱藏系統身分；模型評審則兩種呈現順序都要跑。

5. 構造一對獎勵風格、卻選到較缺乏支持回覆的偏好資料，說明哪種額外評估能揭露問題，並計算 128×256 矩陣的秩 4 LoRA 更新有多少可訓練參數。

提示

固定證據，比較自信地加入事實的答案，與保留來源不確定性的限定答案。LoRA 的部分要分別計算兩個因子矩陣。

6. 擴充報告函數，加入時間戳與目標語言選項。拒絕格式錯誤的輸入、保留未知狀態、附上證據 ID，並將文字實現規則版本化。寫下以學得生成器取代模板前需要哪些測試。

提示

不要推測未記錄的時間戳，也不要把未知轉成不存在。要測試前處理一致性、數字格式、書寫系統要求、長輸入、逾時行為與完整回復套件。

參考文獻

1. Masato Hagiwara, *Real-World Natural Language Processing: Practical Applications with Deep Learning*, Chapters 5, 8, 9, 10 and 11, Manning, 2021. Publisher ·Training practices ·Deployment
2. Susan McRoy, *Principles of Natural Language Processing*, open textbook, Chapters 6 and 9, 2021. Benchmark chapter ·Generation chapter
3. Ashish Vaswani et al., “Attention Is All You Need,” NeurIPS, 2017. Paper
4. Alec Radford, Karthik Narasimhan, Tim Salimans and Ilya Sutskever, “Improving Language Understanding by Generative Pre-Training,” OpenAI technical report, 2018. Paper
5. Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei and Ilya Sutskever, “Language Models are Unsupervised Multitask Learners,” OpenAI technical report, 2019. Paper
6. Tom B. Brown et al., “Language Models are Few-Shot Learners,” NeurIPS, 2020. Paper
7. Jared Kaplan et al., “Scaling Laws for Neural Language Models,” arXiv:2001.08361, 2020. arXiv
8. Jordan Hoffmann et al., “Training Compute-Optimal Large Language Models,” arXiv:2203.15556, 2022; published as “An empirical analysis of compute-optimal large language model training,” NeurIPS, 2022. arXiv
9. Ilya Loshchilov and Frank Hutter, “Decoupled Weight Decay Regularization,” ICLR, 2019. Paper
10. Dan Hendrycks and Kevin Gimpel, “Gaussian Error Linear Units (GELUs),” arXiv:1606.08415, 2016, revised 2023. Paper

11. Jimmy Lei Ba, Jamie Ryan Kiros and Geoffrey E. Hinton, “Layer Normalization,” arXiv:1607.06450, 2016. Paper
12. Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes and Yejin Choi, “The Curious Case of Neural Text Degeneration,” ICLR, 2020. Paper
13. Angela Fan, Mike Lewis and Yann Dauphin, “Hierarchical Neural Story Generation,” ACL, 2018. Paper
14. Jade Goldstein and Jaime Carbonell, “Summarization: (1) Using MMR for Diversity-Based Reranking and (2) Evaluating Summaries,” TIPSTER Phase III Workshop, 1998. Paper
15. Sam Wiseman, Stuart M. Shieber and Alexander M. Rush, “Challenges in Data-to-Document Generation,” EMNLP, 2017. Paper
16. Ilya Sutskever, Oriol Vinyals and Quoc V. Le, “Sequence to Sequence Learning with Neural Networks,” NeurIPS, 2014. Paper
17. Chin-Yew Lin, “ROUGE: A Package for Automatic Evaluation of Summaries,” Text Summarization Branches Out, 2004. Paper
18. Lianmin Zheng et al., “Judging LLM-as-a-Judge with MT-Bench and Chatbot Arena,” NeurIPS Datasets and Benchmarks Track, 2023. Paper
19. Peiyi Wang et al., “Large Language Models are not Fair Evaluators,” ACL, 2024. Paper
20. Ziwei Ji et al., “Survey of Hallucination in Natural Language Generation,” ACM Computing Surveys 55, 2023. Survey
21. Joshua Maynez, Shashi Narayan, Bernd Bohnet and Ryan McDonald, “On Faithfulness and Factuality in Abstractive Summarization,” ACL, 2020. Paper
22. Patrick Lewis et al., “Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks,” NeurIPS, 2020. Paper
23. OWASP GenAI Security Project contributors, “LLM01: Prompt Injection,” OWASP guidance, accessed 2026. Official guidance
24. Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang and Weizhu Chen, “LoRA: Low-Rank Adaptation of Large Language Models,” ICLR, 2022. Author publication page
25. Long Ouyang et al., “Training language models to follow instructions with human feedback,” NeurIPS, 2022. Paper
26. Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D. Manning, Stefano Ermon and Chelsea Finn, “Direct Preference Optimization: Your Language Model is Secretly a Reward Model,” NeurIPS, 2023. Paper
27. Wayne Xin Zhao et al., “A Survey of Large Language Models,” Frontiers of Computer Science, 2026. Updated survey
28. Yupeng Chang et al., “A Survey on Evaluation of Large Language Models,” ACM Transactions on Intelligent Systems and Technology, 2024. Survey
29. Junyi Li, Tianyi Tang, Wayne Xin Zhao, Jian-Yun Nie and Ji-Rong Wen, “Pre-Trained Language Models for Text Generation: A Survey,” ACM Computing Surveys, 2024. Publisher ·Author manuscript
30. Patricia Schmidtova et al., “Do My Eyes Deceive Me? A Survey of Human Evaluations of Hallucinations in NLG,” INLG, 2025. Survey
31. Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, Meng Wang and Haofen Wang, “Retrieval-Augmented Generation for Large Language Models: A

Survey," arXiv:2312.10997, revised 2024. Survey
32. Woosuk Kwon et al., "Efficient Memory Management for Large Language Model Serving with PagedAttention," SOSP, 2023. Conference ·Paper
33. Christopher D. Manning, Prabhakar Raghavan and Hinrich Schütze, *Introduction to Information Retrieval*, Chapters 6, 8 and 11, Cambridge University Press, 2008. Author-hosted book
34. scikit-learn developers, "Text feature extraction," scikit-learn documentation, accessed 2026. Documentation
35. Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen and Wen-tau Yih, "Dense Passage Retrieval for Open-Domain Question Answering," EMNLP, 2020. Paper
36. Kalervo Järvelin and Jaana Kekäläinen, "Cumulated Gain-Based Evaluation of IR Techniques," ACM Transactions on Information Systems 20(4), 422–446, 2002. Publisher

NLP 10・預訓練模型：BERT

從 WordPiece 輸入到雙向編碼器理解 BERT，從零以遮罩語言建模預訓練一個小型 BERT 式模型，並探測其上下文表示包含什麼。

網站版本: https://blog.shyandram.dev/zh-tw/learn/nlp/pretrained-bert/

你將學到

- 說明「先預訓練、再微調」的配方，以及 BERT 與 ELMo、GPT 的相對位置。
- 把文字轉成 BERT 輸入：WordPiece 片段、`[CLS]` ／ `[SEP]` 打包，以及相加的詞元、段落與位置嵌入。
- 精確說明遮罩語言建模目標（選取 15%，再依 80/10/10 替換），以及下一句預測目標與後來的批評。
- 在玩具語料上從零預訓練一個小型 BERT 式編碼器，並量測它學到什麼。
- 區分上下文嵌入與靜態嵌入，並用控制任務探測（probe）表示。
- 把 RoBERTa、ALBERT、DistilBERT、ELECTRA、多語 BERT 與中文 BERT 放進同一張地圖。

先看全貌

一般 Word2Vec 特徵替每個詞取出一個固定向量 [6]。BERT，全名 **Bidirectional Encoder Representations from Transformers**，則讀取周圍文字，替每次出現的詞元重新計算表示 [1]。在「the crane lifts steel」與「the crane eats fish」中，`crane` 查表得到的初始嵌入相同，編碼器卻能利用其他詞算出不同的最終向量。關鍵是區分詞彙表項目與這次出現的表示。

BERT 分兩階段學習。**預訓練**使用一般無標籤文字，由文字自己提供目標。**微調**再把同一個網路加上小型任務頭，調適到情感或蘊涵等有標籤任務。第一階段昂貴、只做一次；第二階段便宜、每個任務各做一次。

> **用最簡單的話說**
>
> 用拇指蓋住句子中的一個詞，請朋友猜：「the crane ___ steel」。朋友看得到拇指兩側的詞，所以「lifts」是好答案。這個遊戲玩上幾百萬次，就能學到很多詞語如何搭配。之後，朋友只要看幾個例子，就能學會新工作，例如分辨開心和生氣的訊息，因為他們已經懂句子怎麼運作。

Hagiwara 第 9 章在實務情境中說明預訓練模型調適 [18]。本章則走進模型內部，並以 NLP 09 的因果語言模型作對照，沿用 NLP 07 的注意力區塊與 NLP 04 的靜態嵌入。接在 BERT 上的分類、蘊涵、標記與擷取式問答任務頭，是 NLP 11 的主題。這裡每個模型都很小，且從零訓練，不下載任何檢查點。

1. 先預訓練，再微調

直觀說法。先從大量無標籤文字學到語言的一般知識，再用少量標籤把它用在特定任務。問題在於：哪種預訓練目標能產生最可重用的表示？

三個 2018 年的設計構成 BERT 的背景。ELMo 訓練由左到右與由右到左的 LSTM 語言模型，並把其內部狀態的學得組合，當作上下文詞特徵提供給既有任務模型 [5]。GPT 把 12 層僅解碼器 Transformer 預訓練成由左到右語言模型，再針對每個任務微調整個網路 [4]。BERT 保留 GPT「微調全部參數」的配方，但把因果目標換成遮罩目標，讓每一層都能使用左右兩側上下文 [1]。

BERT 論文主張，由左到右的限制對句子層級任務並非最佳，對問答這類需要兩個方向上下文的詞元層級任務更可能有很大傷害 [1]。其消融實驗包含一個沒有下一句預測的由左到右模型，預訓練與微調都受同樣的只看左側限制，並與雙向模型比較 [1]。工程上的教訓是：可見性模式是與任務相連的建模選擇，不是個人喜好。

預訓練提供參數初始化與表示函數，卻沒有提供應用標籤。幫浦維護分類器仍需要範例定義「緊急」，實體標記器也需要辨認現場設備名稱的標註。微調會一起調整編碼器與任務頭；凍結編碼器、只訓練任務頭，成本較低，但調適能力也不同。

2. 遮罩與條件目標

直觀說法。不是每個預訓練語言模型都預測下一詞元。遮罩目標使用周圍內容，條件生成則根據另一個輸入預測輸出序列。

BERT 的遮罩語言目標藏起選定位置，再使用雙向上下文預測原詞元 [1]。簡化描述如下：

$$\mathcal{L}_{\text{mask}} = -\frac{1}{|M|} \sum_{i \in M} \log p_\theta(x_i \mid \tilde{\mathbf{x}}).$$

其中 M 是受監督遮罩位置集合；$|M|$ 是集合大小；x_i 是原詞元；θ 是模型參數；$\tilde{\mathbf{x}}$ 是受損輸入序列。波浪號表示損壞，不是獨立抽出的文件。這個式子描述遮罩預測部分；完整的損壞程序見第 5 節。

遮罩預測器適合上下文表示、分類與區段任務，其損失卻不會自動成為一般由左到右序列的對數機率。把各自讀取句子兩側的預測相乘，通常不會得到與因果分解相同的正規化聯合分布。因此，遮罩目標分數與因果困惑度不能直接互換。

條件序列生成則寫成：

$$p_\theta(\mathbf{y} \mid \mathbf{x}) = \prod_{t=1}^{U} p_\theta(y_t \mid y_{<t}, \mathbf{x}).$$

其中 $\mathbf{x}$ 是來源序列；$\mathbf{y}$ 是目標序列；U 是包含結束詞元的目標長度；$y_{<t}$ 是目標前綴。來源可以完整可見，目標仍保持因果。T5 把多種 NLP 任務整理成文字到文字架構，並研究 Transformer 編碼器—解碼器的遷移學習 [27]；NLP 08 深入介紹這一類模型。

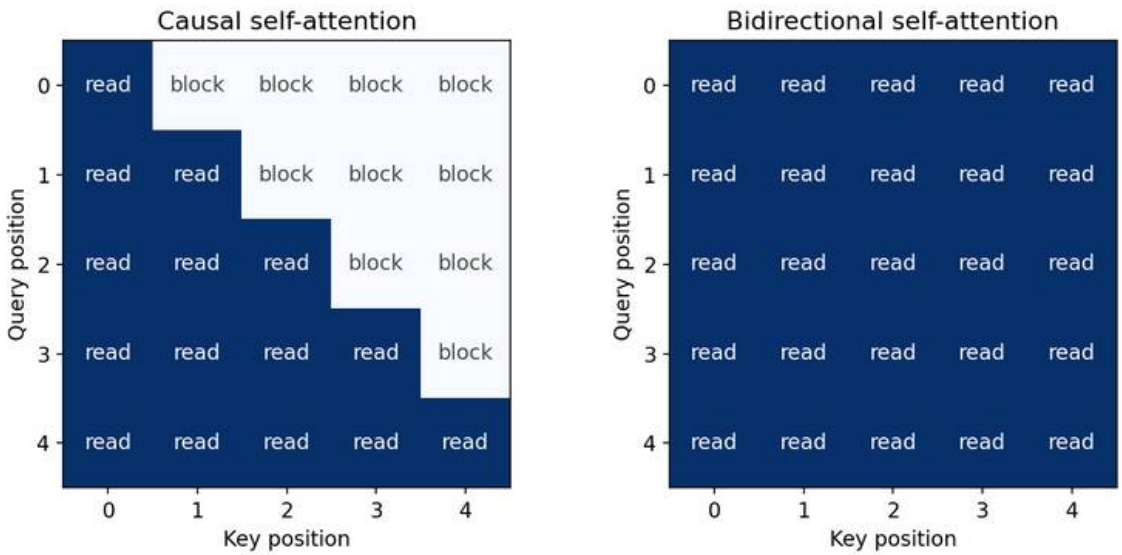


圖 *1* — 可見性是任務定義的一部分。列是發出查詢的位置，欄是它們可以讀取的位置。*BERT* 在每一層使用右圖的模式；填補鍵仍會被阻擋。

這些區別有助選模型。完整句子已知時，分類器可使用雙向編碼器；串流延續則必須排除未來輸出。翻譯可以在來源全部抵達後讀取完整來源，但同步翻譯需要更嚴格的來源可見性契約。應由任務可取得的資訊決定遮罩，而非因為某個模型名稱熟悉就選它的遮罩。

3. 從文字到 BERT 輸入

直觀說法。 BERT 從不直接看到詞。文字先被拆成詞彙表片段，再加上特殊標記，每個位置則得到三個學得向量的和。

WordPiece

原始分詞器先做文字前處理，再執行 WordPiece 分詞 [2]。BERT 使用含 30,000 個詞元的 WordPiece 嵌入 [1]，這是為 Google 神經機器翻譯系統提出的子詞方法 [7]。需要時，WordPiece 把詞拆成詞彙表中的片段；原始詞彙表以 `##` 標示接續片段。原始實作以**貪婪的最長匹配優先**切分每個詞：先取詞首最長的詞彙項目，再取最長的接續片段，依此類推；若某段剩餘部分沒有任何匹配，整個詞就變成 `[UNK]` [2]。詞彙表事先從語料學得，就像 NLP 02 中學得 BPE 合併規則一樣。

```
vocab = {'[UNK]', 'the', 'pump', 'over', 'heat', '##heat', '##ed', '##s',
         'un', '##load', 'load', '##ing', '幫', '浦', '過', '熱'}

def wordpiece(word, max_chars=200):
    """Greedy longest-match-first split into vocabulary pieces."""
    if len(word) > max_chars:
        return ['[UNK]']
    pieces, start = [], 0
```

```
    while start < len(word):
        end = len(word)
        while end > start:
            piece = word[start:end] if start == 0 else '##' + word[start:end]
            if piece in vocab:
                break
            end -= 1
        if end == start:                    # no piece fits: the whole word is unknown
            return ['[UNK]']
        pieces.append(piece)
        start = end
    return pieces

def basic_split(text):
    """Lowercase, put spaces around CJK ideographs, split on whitespace."""
    out = []
    for ch in text.lower():
        cjk = 0x4E00 <= ord(ch) <= 0x9FFF
        out.append(f' {ch} ' if cjk else ch)
    return ''.join(out).split()

for text in ['The pump overheated', 'unloading pumps', 'the pumpkin', '幫浦過熱']:
    tokens = [p for w in basic_split(text) for p in wordpiece(w)]
    print(f'{text!r:24} -> {tokens}')
assert wordpiece('overheated') == ['over', '##heat', '##ed']
assert wordpiece('pumpkin') == ['[UNK]']
```

輸出：

```
'The pump overheated'    -> ['the', 'pump', 'over', '##heat', '##ed']
'unloading pumps'        -> ['un', '##load', '##ing', 'pump', '##s']
'the pumpkin'            -> ['the', '[UNK]']
'幫浦過熱'                   -> ['幫', '浦', '過', '熱']
```

這帶來三個結果。一個詞可能占多個模型位置，因此詞標籤與字元區段必須對應到詞元位置。剩餘部分無法涵蓋的詞會變成單一 `[UNK]`，即使它的前綴 `pump` 是已知詞；這是貪婪規則的性質，不是語言的性質。中文字則在 WordPiece 之前一字一個位置地切開，原始多語與中文模型也是如此（第 11 節）。這裡的 `basic_split` 只檢查主要的 CJK 統一表意文字區塊；原始程式還檢查其他幾個範圍，並處理重音與標點 [2]。

打包一段文字或一對文字

單一文字打包成 `[CLS] text [SEP]`；文字對則打包成 `[CLS] text A [SEP] text B [SEP]`。序列分類器可讀取 `[CLS]` 位置的最終狀態，`[SEP]` 則標記邊界。段落 ID 區分兩部分，卻**不會阻擋跨段落注意力**：文字 A 的查詢仍能讀取文字 B。決策依賴兩段文字互動時，這種聯合編碼很有用 [2]。

進入編碼器前，先相加三種學得的向量：

$$\mathbf{e}_i = E[x_i] + P[i] + S[s_i], \qquad H = \mathrm{Encoder}_\theta(\mathbf{e}_1, \dots, \mathbf{e}_T).$$

其中 T 是打包後的序列長度；θ 表示編碼器參數；x_i 是位置 i 的詞元 ID；$s_i \in \{0, 1\}$ 是段落 ID；$E \in \mathbb{R}^{|V| \times d}$ 是詞彙表 V 的詞元表；$P \in \mathbb{R}^{T_{\max} \times d}$ 是位置表，支援長度為 $T_{\max}$；$S \in \mathbb{R}^{2 \times d}$ 是段落表；$\mathbf{e}_i \in \mathbb{R}^d$ 是相加後的輸入；$H \in \mathbb{R}^{T \times d}$ 包含上下文輸出列 $\mathbf{h}_i$。原始實作還會對相加後的嵌入執行正規化與 dropout [2]。填補鍵被阻擋，兩側的一般文字則保持可見。

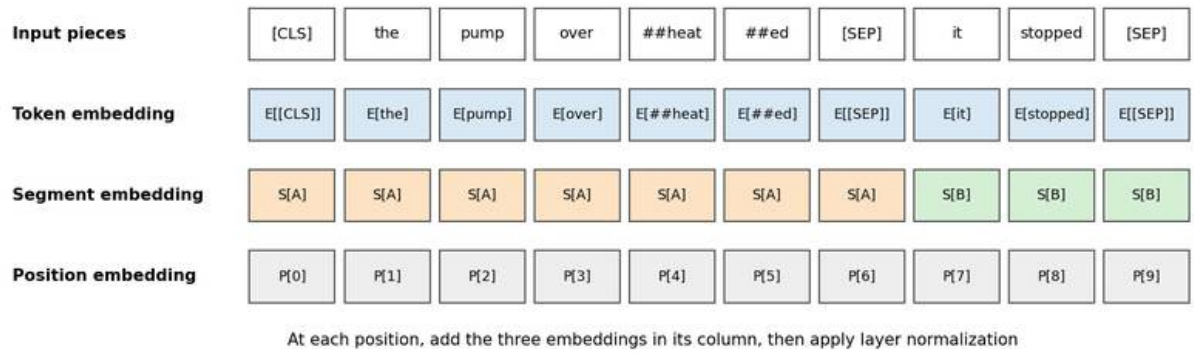


圖 2 — 文字對的 *BERT* 輸入。每一欄是一個位置，把該欄的詞元、段落（*A* 或 *B*）與位置嵌入相加。位置表是學得的，因此限定了最大輸入長度。

與原始 Transformer 的固定正弦位置不同，BERT 的位置向量是學得的列 [2]。學得的位置表限制支援的最遠位置；更長的輸入是非法索引，不是普通的長序列，必須截斷或切成視窗。嵌入列也只在該模型詞元 ID 的定義下有意義。未調適權重就換分詞器，可能讓形式合法的數字代表不相關符號。

4. 編碼器

直觀說法。 BERT 是 Transformer 的編碼器那一半：一疊相同的區塊，每個區塊讓每個位置讀取所有其他位置，再分別轉換每個位置。

BERT 疊合多個 Transformer **編碼器**區塊 [1, 3]。每個區塊先用雙向自注意力混合資訊，再用前饋網路轉換各位置；殘差連接與層正規化支援整個堆疊 [9]。「雙向」表示編碼器各層都能讀取輸入左右兩側，不是像 ELMo 那樣各訓練一個因果模型、再把輸出接起來 [5]。NLP 07 推導多頭注意力；BERT 唯一特有的選擇是圖 1 的注意力遮罩。

論文報告兩種大小。以 L 表示區塊數、H 表示隱藏寬度、A 表示注意力頭數，BERT-Base 為 $L = 12$、$H = 768$、$A = 12$，共 1.1 億個參數；BERT-Large 為 $L = 24$、$H = 1024$、$A = 16$，共 3.4 億個參數 [1]。BERT-Base 刻意選成與原始 GPT 同樣大小以便比較 [1]。前饋層使用 GELU 活化函數 [8, 1]。預訓練使用 BooksCorpus 與英文維基百科，論文強調使用文件層級語料，才能取得長的連續序列 [1]。

可把輸出想成每個輸入詞元各占一列的表格，而不是單一的句子向量。詞元標籤頭逐列讀取；序列頭則透過池化轉換讀取第一列。同一編碼器能支援兩者，因為輸出保留了詞元位置；NLP 11 會建立這些任務頭。

5. BERT 在取得任務標籤前學什麼？

直觀說法。遮罩語言建模把原始文字變成填空練習；下一句預測則多問一個是非題：兩段文字是否相連。

遮罩語言建模

遮罩語言建模（masked language modeling，MLM）從一般文字學習：原詞元就是自己的目標。例如 `the crane [MASK] steel` 要求模型利用兩側文字猜回被移除的詞元。預測損失同時更新編碼器與詞彙頭，因此最終詞元向量可作為後續監督任務的輸入。

原始 BERT 選取 15% 的 WordPiece 位置作為預測目標。在被選位置中，80% 換成 `[MASK]`、10% 換成隨機詞元、10% 保留原詞元 [1]。選取與替換是不同步驟：**每個被選位置都提供目標**，即使可見詞元未改變。其他位置提供上下文，不貢獻 MLM 損失。這種混合替換方式縮小預訓練輸入與一般未遮罩任務輸入的差異，因為後者從不出現 `[MASK]`。

這個設計的代價是樣本效率：只有被選位置產生損失。ELECTRA 的作者正是指出這一點：MLM 每個例子只從被遮罩的那一小部分詞元學習 [14]。原始實作也在準備資料時就一次決定遮罩；RoBERTa 比較這種靜態遮罩與**動態遮罩**（每次使用序列時重新抽取遮罩模式），發現動態遮罩相當或略好 [11]。第 7 節的小模型使用動態遮罩。

下一句預測及其批評

原始 BERT 也訓練**下一句預測**（next sentence prediction，NSP）[1]：文字對有一半來自連續語料，另一半則從語料隨機抽取第二段。池化後的 `[CLS]` 狀態預測這個二元標籤。NSP 預測關係標籤，不會生成下一句。原始預訓練結合 MLM 與 NSP，論文的消融實驗報告移除 NSP 會降低 QNLI、MNLI 與 SQuAD 1.1 的表現 [1]。

後續研究有不同結論。RoBERTa 的複製研究發現，以連續文字填滿輸入時，移除 NSP 損失的下游表現**相當或略好**，並推測原始消融可能只移除了損失項，卻保留了段落對的輸入格式 [11]。ALBERT 的作者推測 NSP 太簡單，因為它把主題預測與連貫性預測混在一起：隨機抽取的第二段通常主題不同。他們改用**句序預測**（sentence-order prediction），取同一文件中兩個連續段落，維持原順序或對調，讓主題不再洩漏答案 [12]。一般教訓是：應檢查實際檢查點的訓練目標，不要假定後來的每個編碼器都保留所有原始設計。

僅線上版：*BERT — Pre-training of Deep Bidirectional Transformers for Language Understanding* — *https://blog.shyandram.dev/zh-tw/learn/nlp/pretrained-bert/papers/bert–pre-training-of-deep-bidirectional-transformers-for-language-understanding/*

6. 可執行的雙向編碼器

直觀說法。在預訓練完整模型之前，先隔離讓它像 BERT 的兩個性質：改變較後面的詞會改變較前面詞的表示，而填補完全不造成影響。

以下 CPU 範例以四個自創句子訓練 MLM，固定選取位置 2 並換成 `[MASK]`。因此，右側可見動詞有助區分 `robot` 與 `sensor`。這個刻意簡化的替換規則用來觀察機制；模型隨機初始化，只有一個小型編碼器區塊，並以線性詞彙頭取代原始 BERT 的完整預測頭。它使用層正規化、GELU 與 AdamW [8–10]。

```
import torch
from torch import nn
from torch.nn import functional as F

torch.manual_seed(23)
torch.set_num_threads(1)
vocab = ['[PAD]', '[CLS]', '[SEP]', '[MASK]', 'the', 'a',
         'robot', 'sensor', 'moves', 'measures']
index = {word: i for i, word in enumerate(vocab)}
rows = ['the robot moves', 'the sensor measures',
        'a robot moves', 'a sensor measures']
ids = torch.tensor([[index[w] for w in ['[CLS]'] + s.split()
                     + ['[SEP]', '[PAD]']] for s in rows])
blocked = ids.eq(index['[PAD]'])
corrupted = ids.clone()
corrupted[:, 2] = index['[MASK]']
labels = torch.full_like(ids, -100)
labels[:, 2] = ids[:, 2]

class TinyEncoder(nn.Module):
    def __init__(self):
        super().__init__()
        self.word = nn.Embedding(len(vocab), 16)
        self.position = nn.Embedding(6, 16)
        self.segment = nn.Embedding(2, 16)
        self.embed_norm = nn.LayerNorm(16)
        self.layer = nn.TransformerEncoderLayer(
            16, 2, 32, dropout=0.0, activation='gelu', batch_first=True)
        self.mlm = nn.Linear(16, len(vocab))
        self.pool = nn.Sequential(nn.Linear(16, 16), nn.Tanh())
        self.classifier = nn.Linear(16, 3)
        self.tagger = nn.Linear(16, 3)
        self.span = nn.Linear(16, 2)
    def forward(self, tokens, padding):
        positions = torch.arange(tokens.shape[1])
        segments = torch.zeros_like(tokens)  # single segment here
        h = self.embed_norm(self.word(tokens) + self.position(positions)
                            + self.segment(segments))
        return self.layer(h, src_key_padding_mask=padding)

model = TinyEncoder()
optimizer = torch.optim.AdamW(model.parameters(), lr=0.01)
def objective():
    logits = model.mlm(model(corrupted, blocked))
    return F.cross_entropy(logits.reshape(-1, len(vocab)), labels.reshape(-1))
initial = objective().item()
for _ in range(100):
    optimizer.zero_grad()
    loss = objective()
```

```
    loss.backward()
    optimizer.step()
model.eval()
with torch.no_grad():
    final = objective().item()
    h = model(ids, blocked)
    changed = ids.clone()
    changed[0, 3] = index['measures']
    context_gap = (model(changed, blocked)[0, 2] - h[0, 2]).abs().max().item()
    changed_padding = ids.clone()
    changed_padding[blocked] = index['robot']
    pad_gap = (model(changed_padding, blocked)[~blocked] - h[~blocked]).abs().max().item()
    shapes = [model.classifier(model.pool(h[:, 0])).shape,
              model.tagger(h).shape, model.span(h).shape]
assert final < initial and context_gap > 1e-4 and pad_gap < 1e-6
print('masked loss:', round(initial, 3), '->', round(final, 3))
print('context changes representation:', context_gap > 1e-4)
print('padding leaves real tokens unchanged:', pad_gap < 1e-6)
print('class / tag / span shapes:', [tuple(s) for s in shapes])
```

輸出：

```
masked loss: 2.223 -> 0.001
context changes representation: True
padding leaves real tokens unchanged: True
class / tag / span shapes: [(4, 3), (4, 6, 3), (4, 6, 2)]
```

遮罩損失由 `2.223` 降到 `0.001`，量測的是對這四個範例的擬合。改變 `robot` 之後的詞會改變它的最終表示；改變被阻擋填補位置所儲存的值，則不影響真實詞元的輸出。因果模型必須排除前一種未來影響（NLP 09 的擾動測試），雙向編碼器則刻意允許它。

任務頭印出的形狀為 `(4, 3)`、`(4, 6, 3)` 與 `(4, 6, 2)`：四個序列各對三個類別做決策、每個範例有六個詞元決策，以及每個詞元的兩個邊界分數。**這些任務頭尚未訓練**。本例只有編碼器與 MLM 頭收到梯度。要建立應用，必須提供任務標籤、最佳化相應任務頭與編碼器，再評估保留資料上的任務輸出，NLP 11 會這樣做。

兩種遮罩需要分清楚。編碼器的填補遮罩決定哪些鍵可讀取；`-100` 標籤告訴交叉熵忽略哪些位置，不會把那些位置移出注意力。在這個示範中，所有真實詞都能提供上下文，但只有被選名詞貢獻 MLM 損失。

7. 從零預訓練一個小型 BERT

直觀說法。現在把真正的損壞配方套到一個小型人造語言上，訓練幾秒鐘，然後檢查三件事：模型填補保留句空格的能力、同一個詞在不同上下文是否得到不同向量，以及簡單分類器能從每一層讀出什麼。

玩具語料由一個小文法產生。主詞限制動詞、動詞限制受詞，所以被遮罩的詞常常可以由**兩側**推出。`crane` 刻意設計成有歧義：它可以 `lift`（機器，起重機），也可以 `eat`（鳥，鶴）。128 個可

能句子中保留 40 個。損壞函數精確實作第 5 節：選取 15%，再依 80/10/10 替換，每一步重新抽取。

```
import itertools, random
from collections import Counter
import numpy as np
import torch
from torch import nn
from torch.nn import functional as F
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import cross_val_score

random.seed(0); torch.manual_seed(0); torch.set_num_threads(1)

# 1. A toy corpus: the subject constrains the verb, the verb constrains the object.
frames = {'robot': ['lifts', 'moves'], 'crane': ['lifts', 'eats'],
          'technician': ['repairs', 'moves'], 'heron': ['eats', 'sees']}
objects = {'lifts': ['steel', 'box'], 'moves': ['box', 'cart'],
           'eats': ['fish', 'bread'], 'repairs': ['pump', 'cart'], 'sees': ['fish', 'boat']}
sentences = [f'{d1} {a}{s} {v} {d2} {o}'.split()
             for s in frames for v in frames[s] for o in objects[v]
             for d1, d2, a in itertools.product(['the', 'a'], ['the', 'a'], ['', 'old '])]
random.shuffle(sentences)
test, train = sentences[:40], sentences[40:]
special = ['[PAD]', '[CLS]', '[SEP]', '[MASK]']
words = sorted({w for s in sentences for w in s})
vocab = special + words
index = {w: i for i, w in enumerate(vocab)}
first_word = len(special)
L = 2 + max(len(s) for s in sentences)

def encode(rows):
    ids = torch.zeros(len(rows), L, dtype=torch.long)
    for r, s in enumerate(rows):
        seq = [index['[CLS]']] + [index[w] for w in s] + [index['[SEP]']]
        ids[r, :len(seq)] = torch.tensor(seq)
    return ids
train_ids, test_ids = encode(train), encode(test)

# 2. BERT-style corruption: select 15% of real word positions; of those,
#    80% -> [MASK], 10% -> random word, 10% -> unchanged. All selected are targets.
def corrupt(ids, generator):
    real = ids >= first_word
    selected = (torch.rand(ids.shape, generator=generator) < 0.15) & real
    labels = torch.where(selected, ids, torch.full_like(ids, -100))
    u = torch.rand(ids.shape, generator=generator)
    out = ids.clone()
    out[selected & (u < 0.8)] = index['[MASK]']
    swap = selected & (u >= 0.8) & (u < 0.9)
    out[swap] = torch.randint(first_word, len(vocab), (int(swap.sum()),),
     ↪  generator=generator)
    return out, labels

class TinyBERT(nn.Module):
    def __init__(self, d=32, layers=2):
```

```
        super().__init__()
        self.word, self.position = nn.Embedding(len(vocab), d), nn.Embedding(L, d)
        self.segment, self.norm = nn.Embedding(2, d), nn.LayerNorm(d)
        self.blocks = nn.ModuleList(nn.TransformerEncoderLayer(
            d, 4, 4 * d, dropout=0.0, activation='gelu', batch_first=True)
            for _ in range(layers))
        self.mlm = nn.Linear(d, len(vocab))
    def forward(self, ids):
        pad = ids.eq(index['[PAD]'])
        h = self.norm(self.word(ids) + self.position(torch.arange(ids.shape[1]))
                      + self.segment(torch.zeros_like(ids)))
        states = [h]
        for block in self.blocks:
            h = block(h, src_key_padding_mask=pad)
            states.append(h)
        return states                                  # embeddings, layer 1, layer 2

model = TinyBERT()
optimizer = torch.optim.AdamW(model.parameters(), lr=3e-3)
g = torch.Generator().manual_seed(1)
for step in range(600):                                # a fresh mask every step
    inputs, labels = corrupt(train_ids, g)
    logits = model.mlm(model(inputs)[-1])
    loss = F.cross_entropy(logits.reshape(-1, len(vocab)), labels.reshape(-1))
    optimizer.zero_grad(); loss.backward(); optimizer.step()
model.eval()

# 3. Held-out masked-word accuracy: mask each word position in turn.
def masked_hits(ids, rows_text):
    by_kind = {}
    with torch.no_grad():
        for r, s in enumerate(rows_text):
            for i, w in enumerate(s):
                x = ids[r:r + 1].clone(); x[0, i + 1] = index['[MASK]']
                guess = vocab[model.mlm(model(x)[-1][0, i + 1]).argmax()]
                kind = 'determiner/adjective' if w in ('the', 'a', 'old') else 'noun/verb'
                by_kind.setdefault(kind, []).append(guess == w)
    return {k: round(float(np.mean(v)), 3) for k, v in sorted(by_kind.items())}
counts = torch.bincount(train_ids[train_ids >= first_word], minlength=len(vocab))
real_test = test_ids[test_ids >= first_word]
print('final training MLM loss:', round(loss.item(), 3))
print('held-out masked accuracy:', masked_hits(test_ids, test))
print('most-frequent-word baseline:', vocab[int(counts.argmax())],
      round((real_test == counts.argmax()).float().mean().item(), 3))
# Upper reference: the best guess given every other word, read off the grammar.
def oracle(text):
    by_kind = {}
    for s in text:
        for i, w in enumerate(s):
            fits = Counter(t[i] for t in sentences if len(t) == len(s)
                           and all(a == b for j, (a, b) in enumerate(zip(t, s)) if j != i))
            kind = 'determiner/adjective' if w in ('the', 'a', 'old') else 'noun/verb'
            by_kind.setdefault(kind, []).append(fits[w] / sum(fits.values()))
    return {k: round(float(np.mean(v)), 3) for k, v in sorted(by_kind.items())}
print('expected accuracy of an ideal guesser:', oracle(test))
```

```
# 4. One word, two contexts: identical input row, different output rows.
with torch.no_grad():
    pair = encode([['the', 'crane', 'lifts', 'the', 'steel'],
                   ['the', 'crane', 'eats', 'the', 'fish']])
    cos = [F.cosine_similarity(s[0, 2], s[1, 2], dim=0).item() for s in model(pair)]
print('cosine of the two "crane" vectors by layer:', [round(c, 3) for c in cos])

# 5. Probes. (a) Part of speech vs a control task that gives each word type a
#    random label; (b) the sense of "crane" (machine vs bird) from its own vector.
pos = {w: 'DET' if w in ('the', 'a') else 'ADJ' if w == 'old' else
       'VERB' if w in objects else 'NOUN' for w in words}
control = {w: random.randint(0, 3) for w in words}
crane = [s for s in sentences if 'crane' in s]
crane_ids = encode(crane)
sense = np.array([int('eats' in s) for s in crane])
def rows(ids, text, layer):
    with torch.no_grad():
        h = model(ids)[layer]
    return np.array([h[r, i + 1].numpy() for r, s in enumerate(text) for i in range(len(s))])
for layer in range(3):
    Xtr, Xte = rows(train_ids, train, layer), rows(test_ids, test, layer)
    acc = []
    for table in (pos, control):
        ytr = [table[w] for s in train for w in s]; yte = [table[w] for s in test for w in s]
        acc.append(LogisticRegression(max_iter=3000).fit(Xtr, ytr).score(Xte, yte))
    h = model(crane_ids)[layer].detach()[:, :, :]
    Xc = np.array([h[r, s.index('crane') + 1].numpy() for r, s in enumerate(crane)])
    acc.append(cross_val_score(LogisticRegression(max_iter=3000), Xc, sense, cv=4).mean())
    print(f'layer {layer}: POS {acc[0]:.2f}  control {acc[1]:.2f}  crane sense {acc[2]:.2f}')
assert cos[0] > 0.999 and cos[-1] < cos[0]
```

輸出（單一 CPU 執行緒約六秒）：

```
final training MLM loss: 0.453
held-out masked accuracy: {'determiner/adjective': 0.418, 'noun/verb': 0.492}
most-frequent-word baseline: the 0.17
expected accuracy of an ideal guesser: {'determiner/adjective': 0.592, 'noun/verb': 0.633}
cosine of the two "crane" vectors by layer: [1.0, 0.819, 0.639]
layer 0: POS 1.00  control 1.00  crane sense 0.38
layer 1: POS 1.00  control 1.00  crane sense 1.00
layer 2: POS 1.00  control 1.00  crane sense 1.00
```

把遮罩詞的數字和兩個參考比較。永遠猜最常見的詞 `the`，有 17% 的機會正確。一個完全知道文法的**理想猜測者**，也就是知道每個空格在給定其他所有詞時的真實分布，在內容詞上也只會答對 63%，而不是 100%：在 `the robot ___ the box` 之後，`lifts` 和 `moves` 都合文法。模型只看 88 個訓練句，就在內容詞上達到 49%。這個差距才是它學到與沒學到多少的誠實量測。許多空格沒有唯一答案，這也是 MLM 準確率從來不是理解能力直接量測的原因。

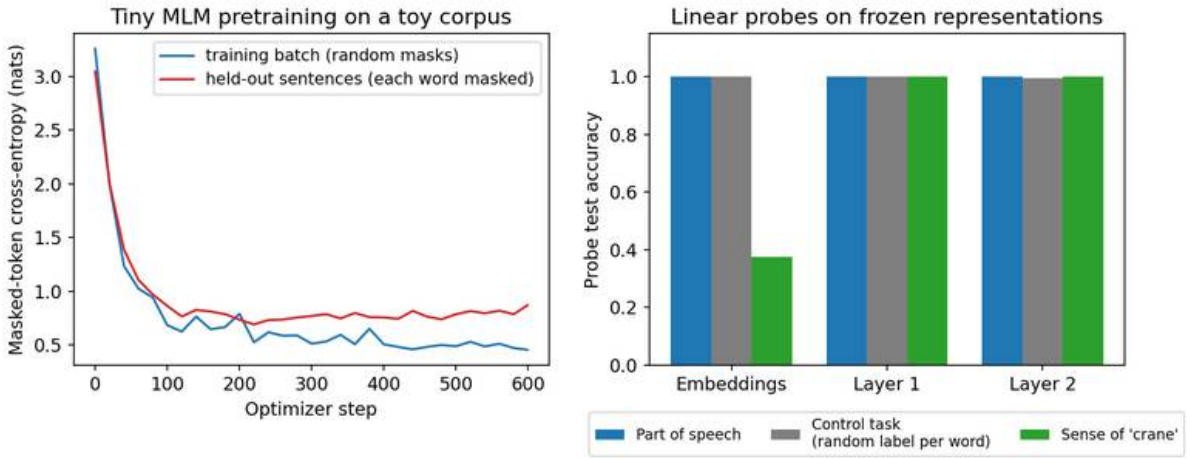


圖 *3* — 小型 *MLM* 訓練。左：隨機遮罩批次上的訓練損失，以及逐一遮罩每個詞時的保留損失；兩者差距是對 *88* 個句子的過擬合。右：在各層凍結表示上的線性探測（第 *9* 節說明控制任務為何重要）。

圖 3 左側顯示，訓練損失持續下降時，保留損失先趨平、再慢慢上升。同一個程式訓練更多步會讓訓練損失更低，保留資料的遮罩準確率卻變差，所以 600 步的預算是看保留損失決定的，而不是看訓練損失。這是一般的模型選擇，對預訓練與微調同樣適用。

8. 上下文嵌入與靜態嵌入

直觀說法。靜態嵌入讓一個詞在任何地方都只有一個向量。上下文編碼器替每次出現各給一個向量，而且差異隨深度增加。

NLP 04 建立的 Word2Vec 或 PPMI 向量是表格中的一列：鳥和機器共用同一個 `crane` 向量，它必須平均兩種上下文。上面印出的餘弦值以極小規模呈現上下文式的替代方案。在嵌入層，兩個 `crane` 向量完全相同（餘弦 1.0），因為詞元和位置都相同。經過一個編碼器區塊後餘弦為 0.819，兩個區塊後為 0.639：右側上下文 `lifts the steel` 或 `eats the fish` 已被混入。

Ethayarajh 在真實模型中量測了這個效應。在 ELMo、BERT 與 GPT-2 中，同一個詞在不同上下文的表示，仍比兩個不同詞的表示更相似，但這種自我相似度在上層低很多；平均而言，一個詞的上下文表示中，只有不到 5% 的變異能由該詞的靜態嵌入解釋 [19]。同一研究也發現，上下文表示在任何一層都不是各向同性的：向量集中在空間中的狹窄區域 [19]。實務上的後果是，BERT 向量之間的原始餘弦相似度，不會自動成為有意義的語意相似度。句子層級的相似度需要自己的訓練目標，NLP 04 與 NLP 11 會討論。

靜態向量仍有用途：便宜、推論時不需要編碼器，也容易檢查。當同一個表面形式帶有不同意義或角色時，上下文向量較合適，而真實文字大多是這種情況。

9. 探測表示中包含什麼

直觀說法。探測器（probe）是在凍結表示上訓練、用來預測某種語言性質的小型分類器。若它成功，表示該性質可以從向量中取回。究竟是表示還是探測器完成了工作，則是另一個問題。

Tenney、Das 與 Pavlick 逐層探測 BERT，發現負責詞性標記、剖析、命名實體、語意角色與共指的區域，大致依傳統 NLP 管線的順序出現，後面的層有時會修正前面的決定 [20]。Hewitt 與 Manning 的**結構探測器**尋找一個線性轉換，使詞向量之間的平方距離編碼剖析樹中的距離；他們在 ELMo 與 BERT 中找到這樣的轉換，在基準中則沒有 [21]。Clark 等人分析的是注意力而非向量，找到專門注意分隔標記、固定位置偏移，以及直接受詞、限定詞與共指提及的注意力頭 [23]。

Hewitt 與 Liang 補上了關鍵的對照。**控制任務**替每個詞型指派隨機標籤，因此只能靠記住詞的身分解決。好的探測器應該具有**選擇性**：語言任務準確率高，對應控制任務準確率低 [22]。沒有這個比較，高探測準確率可能只顯示探測器很強，而不是表示編碼了該性質。

我們的小型實驗呈現了兩面。詞性在每一層都可完美取回，但每個詞的隨機標籤也一樣，因為在這個文法中，詞性完全由詞本身決定。探測器的成功因此只說明「這個詞可以被辨認」，選擇性為零。`crane` 詞義探測則不同。在嵌入層，兩種詞義共用一個向量，準確率在機率水準或更低（0.38，32 個句子、四折）；經過一個編碼器區塊後就完美。這個性質需要上下文，所以逐層上升是編碼器把資訊移進 `crane` 向量的證據。即使如此，探測只顯示資訊對線性讀取器**可得**，不表示模型在預測時**使用**了它。

10. BERT 家族

直觀說法。 BERT 之後，好幾篇論文一次改變配方中的一部分：訓練多久、如何共享參數、如何壓縮，以及預測什麼。

模型	改變了什麼	報告的動機
RoBERTa [11]	以更大批次、更多資料訓練更久，移除 NSP，使用更長序列與動態遮罩	複製研究發現 BERT 明顯訓練不足
ALBERT [12]	分解式嵌入參數化、跨層參數共享、句序預測	降低記憶體並加速訓練；以連貫性目標取代 NSP
DistilBERT [13]	在預訓練階段把知識蒸餾到較小的學生模型，使用三重損失	作者報告小 40%、保留 BERT 97% 的語言理解能力、快 60%
ELECTRA [14]	取代詞元偵測：小型生成器填入被遮罩位置，編碼器判斷每個詞元是原本的還是被替換的	從全部輸入詞元學習，而非只從被遮罩的子集

ALBERT 的分解式嵌入把詞彙表大小與隱藏大小分開：不用一張 $|V| \times H$ 的表，而是一張 $|V| \times E$ 的表加上 $E \times H$ 投影，其中 E 遠小於 H [12]。作者認為 WordPiece 嵌入應學習與上下文無關的表示，隱藏狀態則與上下文有關，所以兩者大小不必相同 [12]。ELECTRA 微調時保留的是判別器；生成器在預訓練後就丟棄 [14]。

這些變體同時改變參數量、計算預算與目標，因此其發表的比較屬於各自的實驗設定。選用時，要檢查它的分詞器、最大長度、訓練目標與訓練資料，並在你在乎的任務上量測。

11. 多語與中文 BERT

直觀說法。一個編碼器可以用共享詞彙表同時在多種語言上預訓練。中文則需要決定單位，因為中文書寫時詞與詞之間沒有空格。

原始釋出包含一個多語 BERT-Base，在 104 種語言的維基百科上訓練，共用 110k 的 WordPiece 詞彙表，且沒有任何告訴模型目前是哪種語言的標記 [2]。Pires、Schlinger 與 Garrette 發現此模型的零樣本遷移出乎意料地好，也就是用一種語言微調、在另一種語言評估，甚至跨越書寫系統也行；在類型學上相近的語言之間遷移最好；但某些語言對有系統性的缺陷 [15]。XLM-R 之後以遠更多經過過濾的 CommonCrawl 資料，在一百種語言上訓練 Transformer 遮罩語言模型，報告在跨語言基準上大幅優於多語 BERT，對低資源語言尤其明顯 [16]。它對正向遷移與容量稀釋之間取捨的分析提醒我們：在固定大小的模型中加入更多語言並非沒有代價 [16]。

對中文，原始多語模型在 WordPiece 之前，於 CJK 表意文字範圍內的每個字前後加上空格，因此中文實際上以字為單位分詞 [2]；第 3 節的 `basic_split` 函數模仿這個規則。釋出內容也提供只用中文的 BERT-Base，涵蓋簡體與繁體中文 [2]。以字為位置可避免斷詞錯誤，但只遮罩「幫浦」這類多字詞中的一個字，模型可以由同詞的另一個字猜回，而不必依賴句子上下文。Cui 等人替中文 BERT 引入**全詞遮罩**（whole word masking）：以斷詞器找出詞的邊界，被選中詞的所有字一起遮罩，而輸入仍以字為單位分詞 [17]。他們的 MacBERT 模型以詞向量相似度選出的相近詞取代 `[MASK]`，縮小預訓練與微調輸入之間的差距 [17]。

處理台灣的繁體中文文字時，要檢查預訓練語料實際包含什麼，而不只看語言標籤。以簡體中文為主的資料所訓練的模型，可能能正確切分繁體字，卻在地區詞彙（例如「資料」與其他地區的用法）或特定領域的台灣術語上較弱。這是要用領域內例子量測的資料問題，不是架構能決定的。

現代觀點

Rogers、Kovaleva 與 Rumshisky 的入門綜述 [24] 回顧了超過 150 篇關於 BERT 的研究：它捕捉了哪些語言與世界知識、存在何處，注意力頭與各層如何運作，對訓練目標與架構的修改，過度參數化與壓縮。其整體訊息與第 9 節一致：許多性質可以被取回，但可取回不等於被使用，探測結果也取決於探測器。

在大型僅解碼器模型的時代，編碼器並沒有消失。**ModernBERT [25]** 發表於 ACL 2025，是 2024 年的僅編碼器模型，把近年的架構與效率最佳化帶進 BERT 設計；作者把編碼器描述為許多正式檢索與分類管線的主力，並報告以 2 兆個詞元訓練、原生序列長度為 8,192。**Zhao 等人的綜述 [26]** 把遮罩編碼器放在預訓練語言模型的更廣歷史中，並指出因果解碼器已被既有大型語言模型廣泛採用為架構；NLP 09 說明了為何單一因果目標就能同時用於評分與生成。

2026 年的實務總結：當整個輸入都可取得、輸出是標籤、區段或檢索用向量，而且延遲與成本重要時，使用雙向編碼器；輸出是開放式文字時，使用僅解碼器模型。開放問題包括超越探測的忠實解釋、低資源語言與書寫系統的穩健多語涵蓋、超出學得位置上限的長文件編碼，以及評估任務結果中有多少來自預訓練表示、而非微調資料。

重點整理

- BERT 是不用標籤預訓練、再以小型任務頭微調的 Transformer 編碼器；預訓練提供表示，不提供任務標籤。
- 輸入是打包成 `[CLS] A [SEP] B [SEP]` 的 WordPiece 片段，加上相加的詞元、段落與學得位置嵌入；段落不阻擋注意力。
- MLM 選取 15% 位置並依 80/10/10 替換；每個被選位置都是目標，遮罩分數也不是因果困惑度。
- NSP 後來被發現並非必要（RoBERTa），或被句序預測取代（ALBERT）。
- 從零預訓練的小型 BERT 呈現了這些機制：保留句空格的填補優於頻率基準、但低於文法本身的上限，而歧義詞的向量隨深度分開。
- 上下文向量隨每次出現而不同，靜態向量則不會。上下文向量之間的原始餘弦，不會自動是語意相似度。
- 探測需要控制任務；探測顯示資訊可得，不表示模型使用它。
- 多語與中文 BERT 做了特定的單位選擇（共享 WordPiece、CJK 逐字切分、全詞遮罩），這些選擇影響模型學到什麼。

練習

1. 在小型 BERT 中，把動態遮罩換成訓練前只抽一次的靜態遮罩。比較保留資料的遮罩準確率與損失曲線。

   提示

   把 `corrupt(train_ids, g)` 移到迴圈外只呼叫一次。靜態遮罩下，每個句子永遠藏同樣的位置，所以許多（句子、位置）目標從未出現。

2. 在玩具語料中加入下一句預測：用 `[SEP]` 與段落 ID 組成兩句的文字對，其中一半的第二句是隨機的。再建立句序版本。在你的語料中，哪一個只靠主題線索就能解決？

   提示

   建立兩份使用不同詞彙的「文件」。隨機第二句通常來自另一份文件，所以 NSP 很容易；同一文件內對調順序則不然。

3. 擴充 WordPiece 詞彙表，使 `pumpkin` 不再是 `[UNK]`，但不要直接加入 `pumpkin`。再找一個詞，讓貪婪最長匹配的切法與片段數最少的切法不同。

   提示

   第一部分加入 `##kin` 就夠了。第二部分可以放入一個會留下難處理剩餘部分的長前綴，以及一個能乾淨切分的短前綴。

4. 在玩具語料上設計一個探測任務，使類似詞性的控制結果會顯示**正的**選擇性。這個性質必須依賴上下文，而不是詞的身分。

提示

名詞是主詞還是受詞是一個選項，但先檢查在這個文法中，位置本身是否已經洩漏答案。若是，就加入打破這種關聯的句子變體。

5. 把 MLM 頭換成 ELECTRA 式的取代詞元偵測器：以隨機詞而非 `[MASK]` 損壞位置，並在每個位置訓練二元分類器。比較每個批次中有多少位置產生損失。

提示

ELECTRA 使用小型的已訓練生成器；隨機替換讓偵測容易得多，所以要說明你的簡化改變了什麼。比較有遮罩與沒有遮罩時的已標註位置數。

6. 把雙向範例改成詞元分類器。若一個已標註詞拆成三個片段，定義哪些位置貢獻損失，以及如何重建詞標籤。

提示

選擇只監督第一片段，或明確定義所有片段的標籤映射。忽略填補與特殊標記、訓練標記頭，再把輸出映射回原文。

參考文獻

1. Jacob Devlin, Ming-Wei Chang, Kenton Lee and Kristina Toutanova, “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding,” NAACL, 2019. Paper
2. Google Research, *BERT: original implementation and released models*, software, 2018. Encoder and pooler ·Tokenizer ·Multilingual and Chinese models
3. Ashish Vaswani et al., “Attention Is All You Need,” NeurIPS, 2017. Paper
4. Alec Radford, Karthik Narasimhan, Tim Salimans and Ilya Sutskever, “Improving Language Understanding by Generative Pre-Training,” OpenAI technical report, 2018. Paper
5. Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee and Luke Zettlemoyer, “Deep Contextualized Word Representations,” NAACL, 2018. Paper
6. Tomas Mikolov, Kai Chen, Greg Corrado and Jeffrey Dean, “Efficient Estimation of Word Representations in Vector Space,” ICLR Workshop, 2013. Paper
7. Yonghui Wu et al., “Google’s Neural Machine Translation System: Bridging the Gap between Human and Machine Translation,” arXiv:1609.08144, 2016. Paper
8. Dan Hendrycks and Kevin Gimpel, “Gaussian Error Linear Units (GELUs),” arXiv:1606.08415, 2016, revised 2023. Paper
9. Jimmy Lei Ba, Jamie Ryan Kiros and Geoffrey E. Hinton, “Layer Normalization,” arXiv:1607.06450, 2016. Paper
10. Ilya Loshchilov and Frank Hutter, “Decoupled Weight Decay Regularization,” ICLR, 2019. Paper
11. Yinhan Liu et al., “RoBERTa: A Robustly Optimized BERT Pretraining Approach,” arXiv:1907.11692, 2019. Paper

12. Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma and Radu Soricut, “ALBERT: A Lite BERT for Self-supervised Learning of Language Representations,” ICLR, 2020. Paper
13. Victor Sanh, Lysandre Debut, Julien Chaumond and Thomas Wolf, “DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter,” 5th Workshop on Energy Efficient Machine Learning and Cognitive Computing at NeurIPS, 2019. Paper
14. Kevin Clark, Minh-Thang Luong, Quoc V. Le and Christopher D. Manning, “ELECTRA: Pre-training Text Encoders as Discriminators Rather Than Generators,” ICLR, 2020. Paper
15. Telmo Pires, Eva Schlinger and Dan Garrette, “How Multilingual is Multilingual BERT?,” ACL, 2019. Paper
16. Alexis Conneau et al., “Unsupervised Cross-lingual Representation Learning at Scale,” ACL, 2020. Paper
17. Yiming Cui, Wanxiang Che, Ting Liu, Bing Qin and Ziqing Yang, “Pre-Training With Whole Word Masking for Chinese BERT,” IEEE/ACM Transactions on Audio, Speech, and Language Processing 29, 2021. Publisher ·Author manuscript
18. Masato Hagiwara, *Real-World Natural Language Processing: Practical Applications with Deep Learning*, Chapter 9, Manning, 2021. Publisher
19. Kawin Ethayarajh, “How Contextual are Contextualized Word Representations? Comparing the Geometry of BERT, ELMo, and GPT-2 Embeddings,” EMNLP-IJCNLP, 2019. Paper
20. Ian Tenney, Dipanjan Das and Ellie Pavlick, “BERT Rediscovers the Classical NLP Pipeline,” ACL, 2019. Paper
21. John Hewitt and Christopher D. Manning, “A Structural Probe for Finding Syntax in Word Representations,” NAACL, 2019. Paper
22. John Hewitt and Percy Liang, “Designing and Interpreting Probes with Control Tasks,” EMNLP-IJCNLP, 2019. Paper
23. Kevin Clark, Urvashi Khandelwal, Omer Levy and Christopher D. Manning, “What Does BERT Look at? An Analysis of BERT’s Attention,” BlackboxNLP Workshop, 2019. Paper
24. Anna Rogers, Olga Kovaleva and Anna Rumshisky, “A Primer in BERTology: What We Know About How BERT Works,” Transactions of the Association for Computational Linguistics 8, 2020. Paper
25. Benjamin Warner et al., “Smarter, Better, Faster, Longer: A Modern Bidirectional Encoder for Fast, Memory Efficient, and Long Context Finetuning and Inference,” ACL, 2025. Paper
26. Wayne Xin Zhao et al., “A Survey of Large Language Models,” Frontiers of Computer Science, 2026. Updated survey
27. Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li and Peter J. Liu, “Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer,” JMLR 21(140), 2020. Paper

NLP 11・以 BERT 做分類（情感分析、自然語言推論）

為單一文字與文字對決策微調預訓練雙向編碼器，從文章詞元讀出答案區段，並誠實評估標籤、機率與捷徑。

網站版本: https://blog.shyandram.dev/zh-tw/learn/nlp/classification-bert/

你將學到

- 依決策型態選擇輸出頭：一個標籤、每個詞元一個標籤，或文章中的一段區段。
- 為情感分析與前提—假說推論微調預訓練編碼器，並了解標準資料集（SST、IMDB、SNLI、MultiNLI、SQuAD）實際包含什麼。
- 辨識標註痕跡（annotation artifacts）、只看假說的基準與挑戰集，理解為何分布內高分可能掩蓋捷徑。
- 有意識地設定學習率、訓練輪數、暖身、類別權重與決策門檻，並知道微調為何可能不穩定。
- 以準確率、macro-F1、校準與覆蓋率評估，並區分相關證據與充分證據。
- 在 CPU 上從頭以遮罩語言模型預訓練一個微小編碼器，再微調於合成情感與推論任務。

先看全貌

NLP 10 建立了 BERT：以預測被遮住詞元來預訓練的雙向編碼器。預訓練給的是一個表示函數，不是一個應用。客服團隊仍要說明「緊急」是什麼意思，推論資料集仍要規定一句話何時可由另一句推出。本章處理第二步：在編碼器上接一個小輸出層，用標註樣本同時訓練兩者，再檢查結果解決的是預期任務，而不是一個方便的仿冒品。

> **用最簡單的話說**
>
> 想像一位讀過上千本書、卻從沒分類過影評的朋友。你給他看二十則標成「喜歡」或「不喜歡」的影評。因為他早就懂「沉悶」與「精彩」這些詞，很快就學會分類。但如果你給的每則「不喜歡」範例剛好都含有「不」字，朋友可能只會找「不」字。微調是那堂短短的課；好的評估則檢查朋友到底學到了什麼。

本章有兩類主要決策。**單一文字分類**替一個輸入指定一個標籤：正面或負面情感、主題、工單路徑。**文字對分類**標註兩個輸入之間的關係：假說是否可由前提推出、與前提矛盾，或兩者皆非？擷取式問答與它們相鄰：模型不是輸出一個標籤，而是選擇答案在文章中從哪裡開始、到哪裡結束。三者都重用同一個預訓練編碼器，只有輸出頭與訓練標籤不同 [3]。

Hagiwara 第 9 章正是針對這些應用討論以預訓練編碼器進行遷移學習 [1]；McRoy 關於基準任務以及資訊擷取與論證的章節則提供語言學觀點 [2]。全程請記住一個問題：什麼證據能說服懷疑者，模型學到的是任務，而不是資料集？

BERT 的應用：選擇輸出頭

直觀說法。任務頭把編碼器的詞元向量轉成需要的答案。先決定答案是一個類別、每個詞元各一個標籤，還是文章中的一段文字。

應用	輸入	監督輸出
情感、主題或客服工單分類	一段文字	由池化 `[CLS]` 狀態預測一個類別
蘊涵、改寫或查詢—文件相關性	一起打包的兩段文字	關係標籤或相關性分數
實體識別或其他詞元標記	已分詞的文字	每個受監督詞元位置的標籤
擷取式問答	問題與文章	文章中的起點與終點
使用 Sentence-BERT 的語意檢索	分開編碼的文字	以句子層級目標學得的可比較向量

序列與詞元任務使用 BERT 論文描述的表示 [3]；原始分類與區段實作明確呈現輸出形狀 [4]。Sentence-BERT 則加入句子比較的訓練 [20]。以下公式說明這些選擇各自計算什麼。

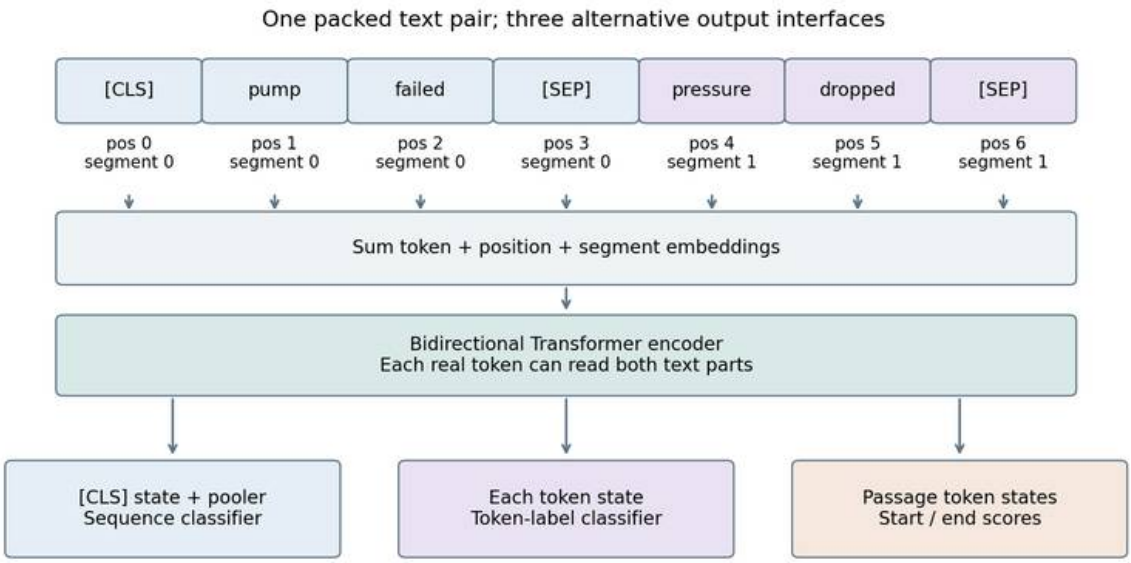


圖 *1* — 共用的 *BERT* 編碼器支援不同監督輸出。片段 *A*/*B* 標籤只標示文字部分，不阻止跨片段注意力。三個任務頭示意不同任務，不是訓練後模型的預測。

序列分類與文字對決策

對有 K 個類別的分類器，一種常見 BERT 介面是：

$$\mathbf{c} = \tanh(W_p \mathbf{h}_{\text{CLS}} + \mathbf{b}_p), \qquad p(y = k \mid \mathbf{x}) = \operatorname{softmax}(W_c \mathbf{c} + \mathbf{b}_c)_k.$$

其中 $\mathbf{x}$ 是打包後的詞元序列；$\mathbf{h}_{\text{CLS}} \in \mathbb{R}^d$ 是第一個上下文輸出列；$W_p \in \mathbb{R}^{d \times d}$ 與 $\mathbf{b}_p \in \mathbb{R}^d$ 組成池化器；$\mathbf{c} \in \mathbb{R}^d$ 是池化輸出；$W_c \in \mathbb{R}^{K \times d}$ 與 $\mathbf{b}_c \in \mathbb{R}^K$ 組成分類器；$y = k$ 表示類別 k。端到端微調

時，標註類別的交叉熵同時更新任務頭與編碼器 [4]。

對「the pump failed」，工單分類器可預測事故類別。對前提—假說文字對，相同輸出形狀可表示蘊涵、矛盾或兩者皆非。這些標籤來自任務資料集，不是預訓練時下一句預測（NSP）的標籤詞彙。文字對自注意力讓一段文字的詞元在最終決策前，先讀取另一段文字。

詞元標記

替每個上下文輸出列接上分類器：

$$p(y_i = k \mid \mathbf{x}) = \operatorname{softmax}(W_t \mathbf{h}_i + \mathbf{b}_t)_k, \qquad \mathcal{L}_{\text{tag}} = -\frac{1}{|J|} \sum_{i \in J} \log p(y_i^\ast \mid \mathbf{x}).$$

其中 i 是詞元位置，K 是標籤數量；$W_t \in \mathbb{R}^{K \times d}$ 與 $\mathbf{b}_t \in \mathbb{R}^K$ 由所有位置共用；y_i 是詞元標籤；$y_i^\ast$ 是標註值；J 是納入監督的非空位置集合。設備名稱若被拆成多個 WordPiece，就需要一致規則：只監督第一個片段、忽略其他片段，或把標籤一致地映射到各片段。J 不應包含填補與特殊標記。推論時，再把預測片段映射回詞或字元區段，細節見 NLP 06。NLP 12 的語意角色標註會以加上述詞標記的方式重用同一個任務頭。

擷取式問答

假設文章寫「Pump B stopped at noon」，問題是「Which pump stopped?」。把問題與文章一起打包、編碼，再替可能的答案邊界評分：

$$a_i = \mathbf{w}_s^\top \mathbf{h}_i, \quad b_i = \mathbf{w}_e^\top \mathbf{h}_i, \qquad \mathcal{L}_{\text{span}} = -\log p_s(i^\ast) - \log p_e(j^\ast).$$

其中 i, j 是候選起點與終點位置；$\mathbf{w}_s, \mathbf{w}_e \in \mathbb{R}^d$ 是學得的起點與終點向量；a_i, b_i 是邊界分數；p_s，p_e 是在允許位置上計算的 softmax 分布；$i^\ast, j^\ast$ 是標註的答案邊界。解碼時，在 $i \le j$ 與指定最大長度下，選取使 $a_i + b_j$ 最大的合法區段 [4]。問題位置與填補不能當成文章答案回傳。若加入「沒有答案」標記，也必須訓練此選項，並以開發資料選擇決策門檻。

BERT 論文為這個任務只新增兩個向量：起點向量與終點向量；每個邊界機率都是它與最終詞元狀態的內積，再對文章位置取 softmax [3]。圖 2 說明為何解碼必須聯合進行，而不是各自取兩次 argmax：最佳起點與最佳終點必須構成合法區段。在這組合成分數中，分數最高的合法區段是「Pump B」。

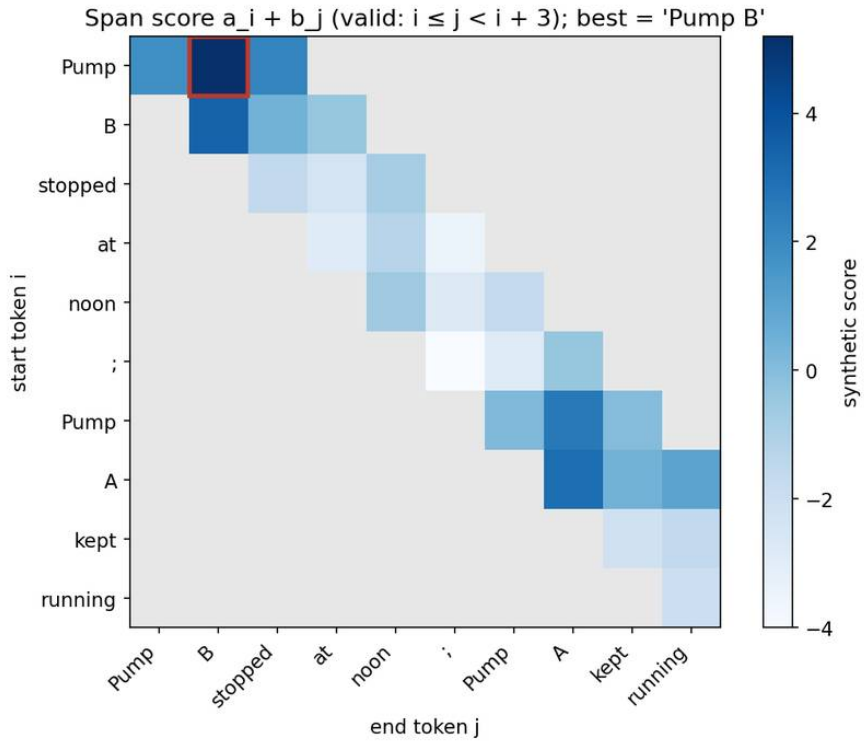


圖 2 — 在起點＋終點分數上解碼區段。灰色格子違反 $i \le j$ 或長度限制。分數為示意用的虛構數值。

輸出會是「Pump B」這類文章文字。這個設定不會生成解釋，也不會找出尚未取得的文章。NLP 09 的檢索增強生成處理找文章的問題；下方關於可回答性的小節則處理：找到的文章是否足夠。

檢索：聯合評分，還是可重用向量？

聯合 BERT 相關性模型把查詢與候選一起打包，因此候選表示依賴這次查詢；換一對文字就要重新執行。句子嵌入模型則分開編碼文件，允許快取。Sentence-BERT 以配對或三元組句子目標訓練第二種介面 [20]。原始預訓練 `[CLS]` 向量或任意平均池化，不會自動形成經過良好訓練的餘弦檢索空間。

實際系統的一種設計是先用分開儲存的向量召回候選，再聯合評分較少的查詢—候選文字對。應分別評估檢索階段的候選召回與最終排序器。NLP 04 說明池化與相似度，NLP 09 展開完整檢索流程。

單一文字分類：情感分析

直觀說法。情感分析判斷一段文字表達正面或負面意見。看似簡單，但否定、轉折與混合意見都表示：只數好話與壞話並不夠。

在 n 個標註樣本上寫出微調目標：

$$\mathcal{L}(\theta, \phi) = -\frac{1}{n}\sum_{i=1}^{n} \log p_{\theta,\phi}(y_i \mid \mathbf{x}_i).$$

其中 θ 是由預訓練初始化的編碼器參數；$\phi = (W_p, \mathbf{b}_p, W_c, \mathbf{b}_c)$ 是隨機初始化的任務頭參數；$\mathbf{x}_i$ 是第 i 段打包文字；y_i 是其標籤；$p_{\theta,\phi}$ 是上面定義的分類分布。完整微調同時更新 θ 與 ϕ。凍結 θ、只訓練 ϕ 比較便宜，但只能重用凍結特徵本來就具備的區分。

兩個基準資料集，以及它們量測什麼

Stanford Sentiment Treebank（SST）包含 11,855 個取自影評的單句；作者剖析這些句子，並請標註者標記剖析樹中全部 215,154 個不重複片語 [5]。片語層級標籤讓研究者能觀察情感如何組合：「not」與「good」組成的片語有自己的標籤。論文同時報告細粒度（五類）與二元的句子層級結果 [5]。GLUE 後來把二元版本收錄為 **SST-2**，列出約 67k 個訓練樣本與 1.8k 個測試樣本，以準確率評估 [14]。最初的 SST 模型是建立在剖析樹上的遞迴網路；ML 04 說明這類架構。

Maas 等人的 **IMDB** 影評資料集含 50,000 則評論，平均分為 25,000 則訓練與 25,000 則測試，另有 50,000 則未標註評論；標註任務是類別平衡的二元問題 [6]。它的單位是整篇評論，不是一句話。評論常比短編碼器視窗長，因此文件分類器需要明確的截斷或分塊政策：保留開頭、保留結尾，或在多個視窗上彙總預測。每種政策回答的問題略有不同，因為評論常在結尾改變語氣。

這兩個資料集說明一個反覆出現的問題：**監督單位**定義了任務。只用獨立意見句訓練的句子情感模型，可能無法處理包含引述、修正與多位說話者的完整對話。如果部署可以查看先前回合，標註與訓練也應包含相同的允許脈絡。反過來，標註者如果看得到最後處理結果，但模型只看得到第一則訊息，就可能產生模型根本無法判定的目標。NLP 03 以計數式分類器詳細說明這份預測約定，換成 BERT 後同樣適用。

上下文編碼器增加了什麼，又沒有增加什麼

詞袋分類器在「the battery is good」與「the battery is not good」中給「good」相同權重。上下文編碼器則在兩句中替「good」算出不同向量，所以任務頭原則上能學會反轉。NLP 03 以計數特徵做了受控否定測試，顯示稀疏證據會讓這種反轉無法確定。上下文讓區別**可以被表示**；標註資料仍必須讓它**可以被學到**。片語身分也不同於作用範圍：區分「not every repair succeeded」與「every repair failed」需要組合性的區別，相鄰詞特徵無法提供；微調後的編碼器也只有在訓練資料包含這種區別時才會學到。

情感標籤本身也是標註決策。一則評論可能稱讚螢幕、批評電池。單一標籤情感分析強迫這則評論歸入一類；面向式情感分析則把意見指派給目標。兩者都不是「唯一正確」的任務，應選擇應用實際使用其輸出的那一種。

文字對：自然語言推論

直觀說法。文字可以支持某個結論，卻未必證明它在現實中為真。文字蘊涵與證據品質要分開。

自然語言推論（natural language inference，NLI）依標註慣例，把前提與假說分為蘊涵、矛盾或中立。若前提是「Mina 修了一個感測器」，通常可推出「有人修了某樣東西」，卻推不出「Mina 修了所有感測器」；「Mina 沒有修任何感測器」則與前提衝突。

對 BERT，把文字對打包成 `[CLS] premise [SEP] hypothesis [SEP]`，片段 ID 分別為 0 與 1，再套用 $K = 3$ 的序列任務頭：

$$\hat{y} = \operatorname*{argmax}_{k \in \{\text{E},\text{N},\text{C}\}} p_{\theta,\phi}(y = k \mid [\text{CLS}]\,\mathbf{u}\,[\text{SEP}]\,\mathbf{v}\,[\text{SEP}]).$$

其中 $\mathbf{u}$ 是前提詞元序列，$\mathbf{v}$ 是假說詞元序列，E, N, C 分別表示蘊涵、中立與矛盾，其他符號與序列分類頭相同。因為自注意力跨越 `[SEP]` 邊界，每個假說詞元在每一層都能注意每個前提詞元。這種聯合編碼讓決策能依賴兩段文字逐詞元的比較；這也是文字對模型無法快取其中一側的原因。

SNLI 與 MultiNLI

Stanford Natural Language Inference（SNLI）語料含 570,152 組人工撰寫的句對，標記為蘊涵、矛盾與語意獨立（中立）[7]。標註者看到一則圖片說明作為前提，再撰寫假說。論文討論一個細微的標註選擇：是否假設兩句描述同一事件。若不做此假設，許多看似矛盾的句對就會是中立。語料固定一種慣例，讓每組句對只有一個標籤 [7]。你替自己的推論資料做的，正是這類必須明文記錄的決策。

MultiNLI 把設計擴展到十種書面與口語英語文類、共 433k 個樣本。訓練集只出現五種文類，因此開發集與測試集分成來自訓練文類的 **matched** 部分，以及來自其他文類的 **mismatched** 部分 [8]。這種切分把領域偏移變成可量測的量，而不是事後想到的問題。

標註痕跡與只看假說的基準

被要求寫出矛盾句的群眾標註者，常使用相同手法：否定詞，或換一個活動。Gururangan 等人顯示，一個**只看假說**的簡單文字分類器，能在 SNLI 約 67%、MultiNLI 約 53% 的資料上預測正確標籤，且否定與模糊等現象與特定類別高度相關 [9]。Poliak 等人在十個 NLI 資料集上執行只看假說的基準，發現它在其中多個資料集上顯著勝過多數類別基準 [10]。忽略前提的模型不可能在做推論，所以它的分數量測的是資料集規律。

McCoy、Pavlick 與 Linzen 從相反方向著手：他們建立 **HANS**，一組受控資料，讓三種句法啟發法——詞彙重疊、子序列與成分——給出錯誤答案，並發現在 MNLI 上訓練的模型（包括 BERT）在其上表現很差 [11]。這些研究合起來支持兩個習慣：

1. **一定要跑部分輸入基準**。若只看假說（或只看前提、只看問題）的模型表現良好，基準分數就有一部分與你在意的關係無關。
2. **建立翻轉線索的挑戰句對**。固定詞彙，只改變一個應該重要的區別：角色反轉、否定出現在前提而非假說、可能性與斷言、指控與事實、一個與全部。

推論模型預測的是標註關係，不會獨立驗證前提。若前提是編造的，獲得蘊涵的假說也可能是編造的。因此，結合檢索的應用需要來源選擇，也需要回答支持檢查；下方問答小節會繼續討論。

論證擷取可辨識主張、提出的支持陳述與明說的攻擊關係 [2]。「裝置可靠，因為它通過了一次測試」具有可辨識的論證結構，但辨識結構不代表測試具有代表性，也不表示結論有效。應用可以儲存陳述的理由，並把其證據強度標為尚未評估。

有用的挑戰句對一次改變一個關鍵差別。流暢改寫也是有用變化，但應保留原本標籤。合成改寫若改變作用範圍或刪除歸屬，就必須重新標註，不能自動繼承原紀錄。

僅線上版：*Sentence-BERT — Sentence Embeddings using Siamese BERT-Networks* — *https://blog.shyandram.dev/zh-tw/learn/nlp/classification-bert/papers/sentence-bert–sentence-embeddings-using-siamese-bert-networks/*

把擷取式問答視為文字對任務：答案需要支持

直觀說法。 SQuAD 式問答要求模型在給定文章中指出答案。實務上更難的問題是：文章裡到底有沒有受支持的答案。

Stanford Question Answering Dataset（SQuAD）含 107,785 組由群眾標註者針對 536 篇維基百科文章撰寫的問答；每個答案都是對應段落中的一段區段 [12]。它的兩個指標都忽略標點與冠詞 *a*、*an*、*the*：**完全符合（exact match）**是預測與任一參考答案完全相同的比例；**F1** 量測預測與參考答案的詞元重疊，再對問題平均 [12]。部分給分很重要，因為標註者的分歧大多出現在區段邊緣不重要的詞上。

SQuAD 2.0 加入超過 50,000 個刻意寫得與可回答問題相似的不可回答問題；系統還必須判斷段落何時不支持任何答案並拒答 [13]。BERT 論文的作法是把不可回答問題視為起點與終點都在 `[CLS]` 的區段。預測時比較空答案分數 $s_{\text{null}} = \mathbf{w}_s^{\top}\mathbf{h}_{\text{CLS}} + \mathbf{w}_e^{\top}\mathbf{h}_{\text{CLS}}$ 與最佳非空區段分數 $\hat{s} = \max_{j \ge i}(a_i + b_j)$，只有在 $\hat{s} > s_{\text{null}} + \tau$ 時才回答 [3]。其中 τ 是在開發集上選來最大化 F1 的門檻。這一個門檻就把區段模型變成選擇性回答器。

本機證據式回答器

不用訓練好的區段模型，也能示範回答契約。下面的回答器只處理一個明確問題家族，回傳來源區間。它以 TF–IDF 為四個合成段落排序，再只回答 Pump A 的狹義溫度上限問題。程式保留單字元詞，讓裝置代碼 A 仍是特徵。函數不是通用意圖分類器，呼叫者必須把已聲明的問題家族路由給它。

```
import re
from sklearn.feature_extraction.text import TfidfVectorizer

corpus = {
    'temperature-v2': 'Pump A temperature limit is 80 C. This policy is current.',
    'shutdown-v1': 'Pump A shutdown follows a vibration alarm.',
    'label-v1': 'A technician replaced the cabinet label.',
    'sensor-v1': 'Sensor B reports humidity once per hour.',
}
ids = list(corpus)
vectorizer = TfidfVectorizer(ngram_range=(1, 2), token_pattern=r"(?u)\b\w+\b")
D = vectorizer.fit_transform([corpus[key] for key in ids])
```

```
def rank(query):
    score = (D @ vectorizer.transform([query]).T).toarray().ravel()
    order = sorted(range(len(ids)), key=lambda i: (-score[i], ids[i]))
    return [(ids[i], float(score[i])) for i in order]

def answer_temperature(query):
    if not re.search(r'\bpump\s+a\b', query, flags=re.I):
        return {'answer': None, 'reason': 'unsupported entity'}
    key, score = rank(query)[0]
    text = corpus[key]
    pattern = r'Pump A temperature limit is (\d+) C\.'
    match = re.search(pattern, text)
    if not match:
        return {'answer': None, 'reason': 'no matching evidence'}
    evidence = text[match.start():match.end()]
    result = {'answer': f'{match.group(1)} C', 'document': key,
              'start': match.start(), 'end': match.end(), 'evidence': evidence}
    assert text[result['start']:result['end']] == result['evidence']
    return result
print(answer_temperature('What is the Pump A temperature limit?'))
print(answer_temperature('What is the Pump Z temperature limit?'))
```

Pump A 回傳受支持的 80 C、文件版本與字元區間 `0–33`。即使存在溫度相關段落，Pump Z 問題仍回傳 `unsupported entity`。在 SQuAD 式資料上訓練的區段任務頭會以學得的邊界分數取代正規表示式，但周邊契約——實體檢查、證據區間、驗證過的位移、明確拒答——應該保留。

明確實體檢查避免高詞彙分數把 Pump A 的上限移植到沒有支持的裝置。證據表達式也要求來源說出預期關係，而非只含一個數字。更彈性的系統需要更完整的實體與關係檢查，包括範圍、單位及適用日期。應透過標註測試擴充能力，不要放寬到所有問題都收到答案。

回答器只看最高段落，因此正確證據排在後面時會拒答。這個限制刻意保留並說明。後續版本可以在固定預算內檢查多個候選，記錄最終用了哪個。要同時量測受支持回答與錯誤接受。

函數保留證據切片並驗證位移，可抓出序列化錯誤，卻不能抓全部語意錯誤。引用句仍可能談錯裝置、過期政策或假設情況。精確引文是重要機械檢查，支持關係仍是契約另外一部分。

相關不代表可回答

段落可以討論主題，卻缺少所問事實。系統應能有意義地回傳證據不足。

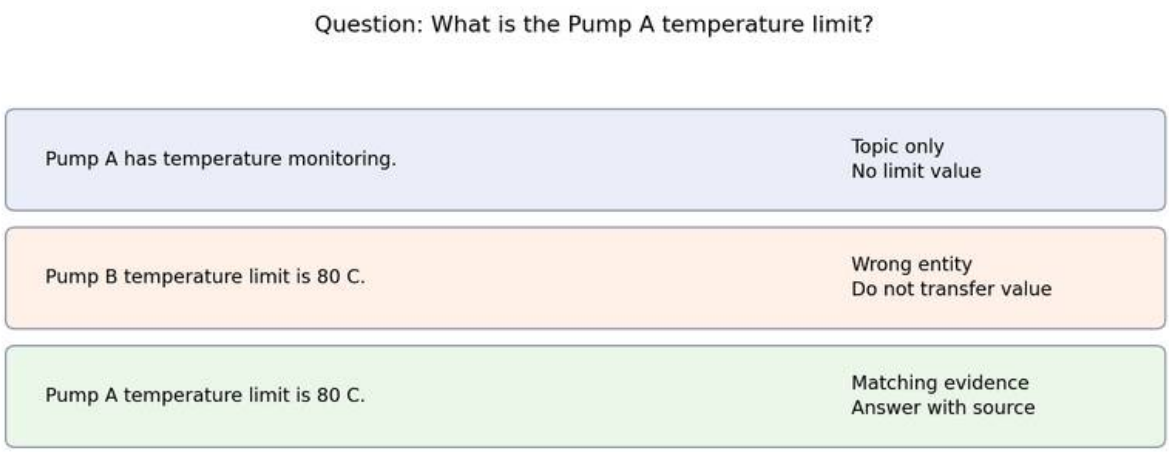


圖 3 — 主題重疊、錯誤實體證據與完整支持需要不同決策，分數本身不能定義可回答性。

狹義問題應先定義必要欄位，再評估系統。溫度範例需要裝置身分、上限關係、數值與單位，也可能需要生效日期或操作條件。即使段落第一名，只要少一個必需欄位，證據仍可能不足。

拒答政策決定何時不回答或請求澄清。若使用信心分數，門檻應在開發資料依錯答與不必要覆核的成本選擇——正是 SQuAD 2.0 任務頭中 τ 的角色。原始相似度不是校準後的正確機率；語料、分詞器或嵌入模型改變時，尺度也可能改變。

同時報告覆蓋率與接受回答的品質：

$$\text{coverage} = \frac{N_{\text{accepted}}}{N_{\text{all}}}, \qquad \text{accepted accuracy} = \frac{N_{\text{correct, accepted}}}{N_{\text{accepted}}}.$$

其中 N_{all} 是評估問題數；N_{accepted} 是收到回答的數量；$N_{\text{correct, accepted}}$ 是依契約判正確的接受回答數。沒有接受回答時，接受準確率未定義；應回報此情況，不能把零接受當成成功準確率。

覆蓋率必須包含沒有可用答案的問題。否則只在處處有答案的集合上，永遠回答的系統也會好看——這正是 SQuAD 2.0 要揭露的弱點 [13]。應納入錯誤實體、缺少值、來源衝突與不支援的時間範圍，測試模型是否認知證據限制，而非只產生可能的區段。

用受控替換診斷失敗

把一個不完美元件換成已知正確輸入，就能找到答案在哪裡失敗。診斷結果應與端到端部署分數分開。

先直接提供標註支持段落給回答器。如果仍答錯值，檢索就不是直接原因；應查實體、單位、範圍與文字實現。如果正確證據能答對，檢索證據卻失敗，就查候選召回與段落次序。這能定位瓶頸，但不假裝部署時也會有完美證據。

接著保留正確段落，加入無關或衝突段落。只有額外上下文才造成失敗，表示是內容選擇或證據整合問題。一次改一個性質：另一裝置、過期版本、引述指控或未授權指令。執行前先標預期行為，避免看到結果後改解釋替輸出辯護。

受支持答案需要一串成功決策。對適用證據已存在語料中的問題，可寫成：

$$P(R \cap E \cap G) = P(R)P(E \mid R)P(G \mid R, E).$$

其中 P 表示機率，R 是充分證據保留於檢索結果的事件，E 是正確解讀證據的事件，G 是輸出忠實實現解讀的事件。這是機率鏈式法則，不是假設獨立。把互不相干的無條件階段分數相乘並不等價：閱讀器在人工挑選簡單文章上的準確率，可能不同於在檢索器實際候選上的準確率。

以下人工評估讓條件關係看得見：

```
import numpy as np

retrieved = np.arange(100) < 80
interpreted = np.arange(100) < 72
realized = np.arange(100) < 68
r = retrieved.mean()
e = interpreted[retrieved].mean()
g = realized[retrieved & interpreted].mean()
complete = (retrieved & interpreted & realized).mean()
print('Conditional stage rates:', round(r, 3), round(e, 3), round(g, 3))
print('Complete supported answers:', round(complete, 3))
assert np.isclose(r * e * g, complete)
```

階段比例為 `0.8`、`0.9`、`0.944`，完整成功率卻為 `0.68`。這是合成結果，不是系統效能報告。各元件表現看似不錯，完整答案仍未必同樣可靠。最後保留真實查詢分布：診斷套件刻意提高困難案例比例，代表性保留集合則估計一般行為。兩者應分開報告目的，不要混成一個總分。

基準：GLUE 與 SuperGLUE

直觀說法。基準套件讓一個預訓練模型跑過許多小任務並彙總結果。它適合比較通用編碼器，卻不擅長告訴你自己的應用是否可行。

GLUE 收錄九個英語理解任務——單句任務（語法可接受度與 SST-2 情感）、相似度與改寫任務，以及包含 MNLI 在內的推論任務——並附上一個針對特定語言現象的診斷資料集與線上排行榜 [14]。它的資料集都不是為此基準從頭建立，部分任務保留私有測試標籤，以確保排行榜被公平使用 [14]。各任務指標不同：SST-2 用準確率，改寫任務用準確率與 F1，句子相似度用 Pearson 與 Spearman 相關係數，類別不平衡的可接受度任務用 Matthews 相關係數 [14]。BERT 論文的主要遷移結果報告在 GLUE、SQuAD 與一個常識推論資料集上 [3]。

SuperGLUE 保留 GLUE 的設計——圍繞八個任務的排行榜、單一彙總分數與分析工具——但挑選更困難的任務，包括針對段落的是非問答，以及嵌入子句上的三類蘊涵 [15]。

讀套件平均分數要小心。它混合了規模、指標與雜訊程度不同的任務，提升可能只來自某個小任務。除非同時報告部分輸入基準與挑戰集，否則它對標註痕跡一無所知；除非你的領域就在其中，否則它也不能說明你的領域。用套件挑選起始檢查點，用自己的保留資料挑選系統。

微調實務

直觀說法。微調是從好權重出發的短期監督訓練。主要風險是學習率太大而破壞預訓練所學、更新次數太少或太多、隨機種子運氣不好，以及類別比例或標籤意義與部署不同的資料。

學習率、訓練輪數與排程

BERT 作者把 dropout 固定在 0.1，並報告以下範圍在其任務上普遍有效：批次大小 16 或 32；Adam 學習率 5×10^{-5}、3×10^{-5} 或 2×10^{-5}；訓練 2、3 或 4 輪 [3]。他們也觀察到，大型資料集（十萬筆以上標註樣本）對這些選擇遠不如小型資料集敏感 [3]。釋出的分類程式預設學習率 5×10^{-5}、三輪、最大序列長度 128、暖身比例 0.1 [4]。其最佳化器排程為：

$$\eta_t = \begin{cases} \eta_{\max}\, t/t_w, & t < t_w, \\ \eta_{\max}\,(1 - t/T), & t \geq t_w. \end{cases}$$

其中 η_t 是第 t 次更新的學習率；$\eta_{\max}$ 是設定的峰值；T 是總訓練步數；t_w 是暖身步數，為 T 的固定比例。同一實作對層正規化與偏差以外的參數施加 0.01 的權重衰減，並把全域梯度範數裁剪在 1.0 [4]。這類解耦權重衰減的最佳化器一般稱為 AdamW [26]。

為什麼學習率這麼小？每次更新都在移動編碼了有用區別的預訓練權重。適合從隨機初始化訓練網路的學習率，可能在幾百步內就把它們抹除，之後模型實際上只是從少量標註資料學習。

不穩定與隨機重啟

對 BERT-LARGE，原作者發現在小型資料集上微調有時不穩定；他們進行數次隨機重啟——保留同一預訓練檢查點，只改變資料洗牌與分類層初始化——再依開發集選出最佳模型 [3]。Mosbach、Andriushchenko 與 Klakow 分析 BERT、RoBERTa 與 ALBERT 在 GLUE 任務上的這種不穩定。他們把失敗的執行歸因於訓練早期的最佳化困難，特徵是梯度消失；其餘變異則來自訓練損失幾乎相同的執行之間的泛化差異 [16]。他們沒有在 BERT-BASE 上觀察到這種不穩定。他們對小型資料集的建議是：使用**含** Adam 偏差修正的小學習率，並大幅增加迭代次數，訓練到接近零的訓練損失 [16]。

實務教訓：報告多個種子的分布，而不只報告最好的種子，且只依開發資料選擇。一次幸運的執行不是方法。

類別不平衡

真實標籤分布很少平衡。類別加權改變每個訓練樣本的貢獻：

$$\mathcal{L}_{\text{weighted}} = -\frac{1}{n}\sum_{i=1}^{n} \alpha_{y_i} \log p_\theta(y_i \mid d_i).$$

其中 $\mathcal{L}_{\text{weighted}}$ 是加權交叉熵，n 是訓練樣本數，i 索引樣本，d_i 是文件，y_i 是標註類別，$\alpha_{y_i} > 0$ 是該類別的權重，p_θ 是分類器分布。加權政策改變訓練目標，但不能創造缺少的語言多樣性。

加權、過度取樣與門檻調整回答不同問題。加權改變參數學習；過度取樣改變呈現頻率，有時只是不斷重複少數類別中的同幾個樣本；門檻調整則在分數算出後改變行動政策。Hagiwara 的開發章節討論不平衡與訓練選擇 [1]。任何介入都應以真實類別盛行率評估，而不是以人為平衡的最終測試集評估。

校準

分類器可能選對標籤，卻提供不可靠的機率。校準（calibration）檢查預測信心是否與相似樣本的觀察正確率一致。模型可以保留 argmax 決策，同時改變信心。Guo 等人研究這個差別與訓練後的溫度縮放 [17]：

$$\tilde{p}(y = k \mid \mathbf{x}) = \operatorname{softmax}(\mathbf{z}/T_c)_k, \qquad \mathrm{ECE} = \sum_{b=1}^{B} \frac{|S_b|}{n} \left| \operatorname{acc}(S_b) - \operatorname{conf}(S_b) \right|.$$

其中 $\mathbf{z}$ 是 logit 向量，$T_c > 0$ 是在保留開發資料上最小化負對數概似所擬合的單一溫度，$\tilde{p}$ 是縮放後的分布；對期望校準誤差 ECE，樣本依信心分成 B 個區間 S_b，n 是樣本數，$\operatorname{acc}(S_b)$ 是區間 b 內的準確率，$\operatorname{conf}(S_b)$ 是其平均預測信心。所有 logit 除以同一個正數不會改變大小次序，因此溫度縮放不可能改變準確率。

Desai 與 Durrett 在 NLI、改寫偵測與常識推理上量測 BERT 與 RoBERTa。直接使用時，預訓練模型在領域內是校準的，且領域外校準誤差低於他們比較的基準；溫度縮放進一步降低領域內誤差，標籤平滑則有助於領域外 [18]。這是那些任務與模型的性質，不保證你的微調檢查點也如此。只要機率會驅動拒答之類的行動，就應在自己的開發資料上量測校準。

評估：準確率、macro-F1 與選擇性決策

直觀說法。準確率計算答對多少。Macro-F1 讓每個類別同等重要，罕見類別因此無法被忽略。選擇性指標則問：系統可以拒答時，被接受的答案有多好。

把一個類別視為正類時：

$$P = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP}}, \qquad R = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}}, \qquad F_1 = \frac{2PR}{P + R}.$$

其中 P 是精確率（precision），R 是召回率（recall），TP 是真陽性數，FP 是偽陽性數，FN 是偽陰性數，F_1 是精確率與召回率的調和平均。分母為零時，必須明確選擇慣例。Macro F1 平等平均各類別 F1；依樣本數加權的平均則讓常見類別影響較大。指標文件說明這些選擇 [19]。

對 SST-2 或 IMDB 這類平衡任務，準確率已足夠 [6]、[14]。三類 NLI 也應報告各類別 F1：模型可能準確率不錯，卻混淆中立與矛盾。對偏斜任務，macro-F1 或 Matthews 相關係數讓罕見類別的失敗比較容易看見，但兩者都不代表失敗的實際成本。漏掉緊急客服案件與誤送一般收據，可能有不同後果。應另外寫出成本政策，而不是把方便計算的分數當成完整效用。

選擇性預測可寫成：

$$\mathrm{coverage} = \frac{1}{n} \sum_{i=1}^{n} a_i, \qquad \mathrm{risk}_{\mathrm{accepted}} = \frac{\sum_{i=1}^{n} a_i \mathbf{1}[\hat{y}_i \neq y_i]}{\sum_{i=1}^{n} a_i}.$$

其中 n 是評估樣本數，$a_i \in \{0,1\}$ 表示是否接受第 i 個樣本，$\hat{y}_i$ 是預測，y_i 是參考標籤，$\mathbf{1}[\cdot]$ 在條件成立時為一，否則為零。如果沒有接受任何樣本，接受風險便未定義。風險與涵蓋率曲線呈現不同門檻下的取捨；部署門檻應依驗證資料選擇，而不是依最終測試標籤選擇。

接著閱讀錯誤。建立錯誤表，包含文字、標準標籤、預測標籤、機率、語言、長度、來源群組與分析者的解釋。實用錯誤類型包括標註歧義、缺少脈絡、否定、未見詞彙、多主題輸入、引述，以及確實超出範圍的文字。不同類型需要不同修正。更多訓練樣本可能改善未見詞彙，卻無法補上預測時永遠拿不到的對話回合。ML 11 涵蓋一般的評估與監測工具。

把模型與前處理產物一起保存。分詞器與詞彙表、最大長度與截斷政策、標籤順序、校準溫度與決策門檻，共同構成推論套件。以另一個分詞器載入權重，會默默改變意義。套件應保存有版本的標籤結構與幾個確定性介面範例，讓服務可以檢查假設。

可執行實驗：先預訓練，再微調

直觀說法。我們在一個微小的未標註語料上，以遮罩語言模型從頭預訓練一個兩層編碼器，再為情感任務與 NLI 任務微調。資料是合成的，因此實驗示範的是機制，不是基準效能。

實驗設計分離出三個問題：

1. **預訓練能遷移嗎？**未標註語料包含所有形容詞，每個都出現在 `love` 或 `hate` 旁邊。情感訓練集從未包含 `superb` 或 `terrible`；測試集只包含這兩個形容詞，有否定也有不否定。隨機初始化的編碼器從未在任何脈絡中見過它們。
2. **只看假說的基準能揭露捷徑嗎？** NLI 訓練資料中，`not` 只出現在矛盾假說裡——一個刻意製造的標註痕跡。
3. **挑戰集能揭露它嗎？**挑戰集讓 `not` 出現在蘊涵中（兩句都否定），並建立假說沒有 `not` 的矛盾（改為前提被否定）。

編碼器以縮小版重現 NLP 10 的 BERT 配方：詞元＋位置＋片段嵌入、層正規化、兩個 Transformer 編碼器區塊、遮住 15% 的位置，以及微調時接在 `[CLS]` 上、含 tanh 層的池化任務頭。最佳化器為 AdamW [26]。

```
import itertools, random
import torch
from torch import nn
from torch.nn import functional as F
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, f1_score

torch.manual_seed(0); random.seed(0); torch.set_num_threads(1)
POS, NEG = ['good', 'great', 'fine', 'superb'], ['bad', 'awful', 'poor', 'terrible']
ITEMS, NAMES = ['battery', 'screen', 'camera', 'speaker'], ['mina', 'ravi', 'lena']
VERBS = {'repaired': 'repair', 'cleaned': 'clean', 'tested': 'test'}
OBJS = ['the pump', 'the valve', 'the sensor']
words = set(' '.join(POS + NEG + ITEMS + NAMES + OBJS).split()) | set(VERBS) \
    | set(VERBS.values()) | set('the is not i love hate it someone did'.split())
vocab = ['[PAD]', '[CLS]', '[SEP]', '[MASK]'] + sorted(words)
idx, L, D = {w: i for i, w in enumerate(vocab)}, 16, 32
```

```
def batch(examples):                       # each example: (text,) or (text_a, text_b)
    ids, segs = [], []
    for ex in examples:
        toks, seg = ['[CLS]'], [0]
        for s, text in enumerate(ex):
            part = text.split() + ['[SEP]']
            toks += part; seg += [s] * len(part)
        ids.append([idx[t] for t in toks] + [0] * (L - len(toks)))
        segs.append(seg + [0] * (L - len(seg)))
    return torch.tensor(ids), torch.tensor(segs)

class TinyBERT(nn.Module):
    def __init__(self):
        super().__init__()
        self.tok, self.pos, self.seg = nn.Embedding(len(vocab), D), nn.Embedding(L, D),
         ↪  nn.Embedding(2, D)
        self.norm = nn.LayerNorm(D)
        block = nn.TransformerEncoderLayer(D, 2, 64, dropout=0.0, activation='gelu',
         ↪  batch_first=True)
        self.encoder = nn.TransformerEncoder(block, 2)
        self.mlm = nn.Linear(D, len(vocab))
    def forward(self, ids, seg):
        h = self.tok(ids) + self.pos(torch.arange(ids.shape[1])) + self.seg(seg)
        return self.encoder(self.norm(h), src_key_padding_mask=ids.eq(0))

# 1) Masked-LM pretraining on unlabeled text (no sentiment or NLI labels).
corpus = [(f'i love it the {i} is {a}',) for i in ITEMS for a in POS] \
    + [(f'i hate it the {i} is {a}',) for i in ITEMS for a in NEG] \
    + [(f'{n} {v} {o}',) for n in NAMES for v in VERBS for o in OBJS]
model = TinyBERT()
ids, seg = batch(corpus)
opt = torch.optim.AdamW(model.parameters(), lr=3e-3)
for step in range(400):
    pick = (torch.rand(ids.shape) < 0.15) & ids.gt(3)    # 15% of real tokens
    logits = model.mlm(model(ids.masked_fill(pick, idx['[MASK]']), seg))
    loss = F.cross_entropy(logits.flatten(0, 1), ids.masked_fill(~pick, -100).flatten())
    opt.zero_grad(); loss.backward(); opt.step()
print('final MLM loss:', round(loss.item(), 3))
pretrained_state = {k: v.clone() for k, v in model.state_dict().items()}

# 2) Fine-tune encoder + [CLS] head end to end.
def finetune(encoder, train, n_classes, epochs=60, lr=1e-3):
    torch.manual_seed(1)
    head = nn.Sequential(nn.Linear(D, D), nn.Tanh(), nn.Linear(D, n_classes))
    opt = torch.optim.AdamW([*encoder.parameters(), *head.parameters()], lr=lr)
    x, y = batch([ex for ex, _ in train]), torch.tensor([lab for _, lab in train])
    for _ in range(epochs):
        loss = F.cross_entropy(head(encoder(*x)[:, 0]), y)
        opt.zero_grad(); loss.backward(); opt.step()
    return lambda exs: head(encoder(*batch(exs))[:, 0]).argmax(-1).tolist()

def report(name, data, predict):
    gold = [lab for _, lab in data]
    with torch.no_grad():
        pred = predict([ex for ex, _ in data])
```

```
    print(f'{name:46s} acc={accuracy_score(gold, pred):.2f} '
          f'macro-F1={f1_score(gold, pred, average="macro"):.2f}')

# Task A: sentiment with negation. "superb" and "terrible" never get a label.
def review(item, adj, negated):
    return ((f'the {item} is {"not " * negated}{adj}',), int(adj in POS) ^ negated)
labeled = [a for a in POS + NEG if a not in ('superb', 'terrible')]
train_a = [review(i, a, n) for i in ITEMS for a in labeled for n in (0, 1)]
test_a = [review(i, a, n) for i in ITEMS for a in ('superb', 'terrible') for n in (0, 1)]
report('sentiment, pretrained encoder', test_a, finetune(model, train_a, 2))
report('sentiment, random-init encoder', test_a, finetune(TinyBERT(), train_a, 2))

# Task B: NLI (0 entailment, 1 neutral, 2 contradiction) with a negation shortcut.
def pair(n, v, o, kind):
    p, neg = f'{n} {v} {o}', f'{n} did not {VERBS[v]} {o}'
    other = list(VERBS)[(list(VERBS).index(v) + 1) % 3]
    return {'ent': ((p, f'someone {v} {o}'), 0), 'neu': ((p, f'{n} {other} {o}'), 1),
            'con': ((p, neg), 2),                  # training: "not" only in contradictions
            'neg-ent': ((neg, neg), 0),            # challenge: "not" in an entailment
            'neg-con': ((neg, p), 2)}[kind]        # challenge: contradiction without "not"
triples = list(itertools.product(NAMES, VERBS, OBJS)); random.shuffle(triples)
train_b = [pair(*t, k) for t in triples[:18] for k in ('ent', 'neu', 'con')]
test_b = [pair(*t, k) for t in triples[18:] for k in ('ent', 'neu', 'con')]
challenge = [pair(*t, k) for t in triples[18:] for k in ('neg-ent', 'neu', 'neg-con')]

vec = CountVectorizer(token_pattern=r'\S+').fit([ex[1] for ex, _ in train_b])
hyp_only = LogisticRegression(C=10, max_iter=1000).fit(
    vec.transform([ex[1] for ex, _ in train_b]), [lab for _, lab in train_b])
hyp_predict = lambda exs: hyp_only.predict(vec.transform([ex[1] for ex in exs])).tolist()
model.load_state_dict(pretrained_state)
pair_predict = finetune(model, train_b, 3)
for split, data in [('in-distribution', test_b), ('challenge', challenge)]:
    report(f'NLI {split}, hypothesis-only', data, hyp_predict)
    report(f'NLI {split}, pair encoder', data, pair_predict)
counter = [pair(*t, k) for t in triples[:18] for k in ('neg-ent', 'neg-con')]
model.load_state_dict(pretrained_state)
report('NLI challenge, pair encoder + counterexamples', challenge,
       finetune(model, train_b + counter, 3))
```

在單一 CPU 執行緒上只需數秒，輸出為：

```
final MLM loss: 0.621
sentiment, pretrained encoder                 acc=1.00 macro-F1=1.00
sentiment, random-init encoder                acc=0.25 macro-F1=0.20
NLI in-distribution, hypothesis-only          acc=1.00 macro-F1=1.00
NLI in-distribution, pair encoder             acc=1.00 macro-F1=1.00
NLI challenge, hypothesis-only                acc=0.33 macro-F1=0.22
NLI challenge, pair encoder                   acc=0.67 macro-F1=0.56
NLI challenge, pair encoder + counterexamples acc=1.00 macro-F1=1.00
```

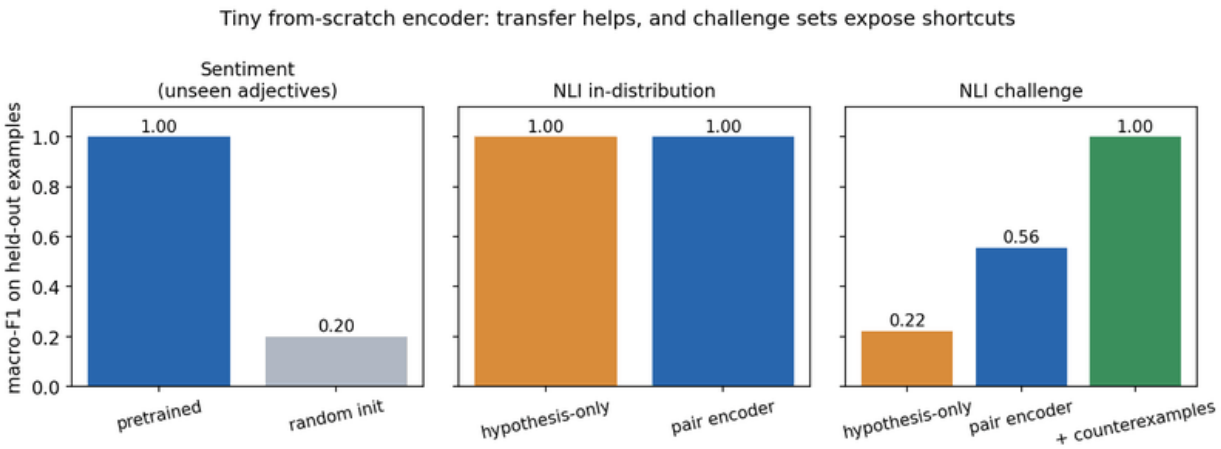


圖 *4* — 上述執行的 *macro-F1*，由圖表腳本重新產生。情感測試句 *16* 句，分布內與挑戰 *NLI* 句對各 *27* 組；全部為合成資料。

遷移。預訓練編碼器正確標記全部十六個測試句，包括否定句，儘管 `superb` 與 `terrible` 從未帶著標籤出現。預訓練期間，它們分別與 `love` 及 `hate` 共享脈絡，因此它們的表示早已靠近同極性的已標註形容詞。隨機初始化的編碼器對這兩個詞只有隨機嵌入，微調集也沒有任何東西能把它們連到標籤。這是縮小版的 BERT 論點：未標註文字可以提供少量標註資料無法提供的區別。它也顯示了依賴性：遷移之所以成功，是因為未標註語料包含相關的共現。用不相干的文字預訓練並不會有幫助。

捷徑。在分布內資料上，只看假說的邏輯斯迴歸是完美的。這正是警訊：從不讀前提的模型不可能在做推論，所以這份測試集並不要求推論。微調後的文字對編碼器在分布內也是完美的，單看這點無法知道它用了什麼。

挑戰。在挑戰集上，只看假說的基準掉到機率水準。檢查預測可發現，文字對編碼器正確分類了每個前提被否定的矛盾句對與每個中立句對，卻把九個雙重否定的蘊涵全部標成矛盾——它部分學到了「假說有 `not` 就是矛盾」。加入十八個兩種挑戰類型的反例（以訓練用的人名、動詞與物件建立）後，挑戰集就被修正。最後一行要小心解讀：反例與挑戰集共用樣板，所以滿分說明的是：只要資料要求，模型就能學會這個區別；並不表示它能泛化到新的捷徑。

這裡一切都刻意縮小：40 個詞的詞彙、最多十幾個詞元的句子、一個種子。在我們試過的四個種子中，定性模式相同，但隨機初始化與挑戰集的確切數字會變動。練習 5 請你量測這個分布。

現代觀點

綜述與其分類地圖。 Minaee 等人在 ACM Computing Surveys 的回顧涵蓋超過 150 個深度學習文字分類模型與超過 40 個資料集，橫跨情感分析、新聞分類、問答與自然語言推論 [21]。它依架構分組——前饋、循環、卷積、膠囊、注意力、記憶增強、圖、孿生、混合，以及基於 Transformer 的預訓練語言模型——核心結論是：以任務標籤微調的預訓練語言模型，已在許多文字分類任務上創下新的最佳成績。Chang 等人的評估綜述區分評估什麼、在哪裡（哪些基準）以及如何（自動、人工、程序）[24]；Zhao 等人的綜述則把分類放在預訓練、調適、使用與能力評估的整體流程中 [25]。合起來看，它們主張依任務擬定評估計畫，而不是只看一個排行榜數字。

LLM 時代的編碼器（2023–2026）。大型僅解碼器模型可以透過提示進行分類，不需任務專屬權重。Zhang 等人在 26 個資料集、13 個情感任務上，把 LLM 與在領域資料上訓練的小型模型比較。LLM 在較簡單的任務上表現令人滿意，但在需要結構化情感資訊或對特定現象更深入理解的任務上落後；在少樣本設定中則明顯勝過小型模型 [22]。實務解讀：標籤稀少時，提示是強力選項；有標註資料且重視延遲時，微調後的編碼器仍具競爭力。編碼器本身也仍在演進。發表於 ACL 2025 的 ModernBERT 把近年的架構與訓練最佳化帶入僅編碼器模型，以 2 兆個詞元、原生 8,192 詞元序列長度訓練，並在分類與檢索評估上報告強勁結果，同時兼顧速度與記憶體效率 [23]。本章的微調介面——接在 `[CLS]` 或詞元狀態上的任務頭——可以原封不動沿用。

沒有改變的事。標註痕跡 [9]、[10] 與啟發式捷徑 [11] 是資料集與訓練訊號的性質，不是某一種架構的性質。更大或更新的模型仍可能學到它們，提示式模型也可能帶來自己的捷徑。部分輸入基準、挑戰集、校準檢查 [18] 與拒答政策 [13] 仍是找出問題的工具。

開放問題。領域偏移下的穩健性（MultiNLI 的 matched ／ mismatched 切分 [8] 是早期且狹窄的版本）、長文與多段落輸入的校準拒答、多語與語碼轉換的情感、隨政策改變的標籤，以及評估集洩漏進預訓練語料。每一項都更適合在大型模型旁邊放一個小而透明的基準來研究。

重點整理

- 編碼器是共用的；任務頭與標籤定義任務。先選序列、詞元或區段輸出。
- 文字對任務把兩段文字打包成一個序列，讓注意力逐詞元比較；代價是任一側都無法快取。
- 監督單位——句子、評論、對話——是任務定義的一部分。
- 只看假說（部分輸入）的基準若表現良好，表示該基準並不完全要求你在意的關係。
- 一次翻轉一個線索的挑戰集，能揭露分布內準確率所掩蓋的捷徑。
- 以小學習率、暖身與衰減微調；報告多個種子的變異，並只依開發資料選擇。
- 把決策（準確率、macro-F1）、機率（校準）與拒答（覆蓋率、接受風險）當作不同性質分別評估。
- 相關不代表可回答：區段答案需要支持它的段落，以及明確說「未提及」的方式。

練習

1. **截斷政策**。你要分類的評論常比編碼器視窗長。比較保留開頭詞元、保留結尾詞元，以及在重疊視窗上平均預測三種作法。每種政策會誤判哪些評論？

   提示

   寫三則簡短的合成評論：結論出現在開頭、出現在結尾，以及在中途反轉（「一開始我很喜歡……但是」）。在選擇政策前，先決定標註準則想要的標籤。

2. **部分輸入稽核**。對實驗中的 NLI 資料，再訓練一個只看前提的基準。執行前先預測它的準確率，再解釋結果。

   提示

   在訓練樣板中，每個前提都會與三種標籤同時出現。只看前提的模型不可能優於多數類別；把它與只看假說的結果比較。

3. **SQuAD 2.0 門檻**。給定可回答與不可回答問題在開發集上的分數 s_{null} 與 $\hat{s}$，說明如何選擇 τ，以及 τ 增大時覆蓋率與接受準確率如何變化。

   提示

   在觀察到的分數差 $\hat{s} - s_{\text{null}}$ 上掃描 τ。τ 越大，回答的問題越少。畫出覆蓋率對接受準確率的曲線，再依錯答的既定成本選點。

4. **校準**。為實驗中的情感分類器加入溫度縮放：在保留切分上以最小化負對數概似擬合一個溫度。說明準確率為何不會改變，並量測平均信心如何移動。

   提示

   把 logit 除以正的純量會保留其次序。改為最佳化 $\log T_c$，可讓溫度保持為正。

5. **種子變異**。以五個種子執行預訓練與微調，報告每一行的平均與範圍，並判斷哪些結論仍然成立。

   提示

   兩個 `torch.manual_seed` 呼叫都要改。把方向性的主張（預訓練優於隨機初始化；挑戰集低於分布內）與確切數值的主張分開。

6. **推論挑戰句對**。建立六組 NLI 句對，分別測試角色反轉、前提中的否定、情態、歸屬、量詞範圍與時間。在評估任何模型前，先寫下預期標籤與假設。

   提示

   前提提供的資訊太少時，要保留中立選項。測試歸屬時，比較「Mina said the pump failed」與「The pump failed」。

參考文獻

1. Masato Hagiwara, *Real-World Natural Language Processing: Practical Applications with Deep Learning*, Chapters 9–11, Manning, 2021. Publisher
2. Susan McRoy, *Principles of Natural Language Processing*, Chapters 6 and 8, open textbook, 2021. Open textbook
3. Jacob Devlin, Ming-Wei Chang, Kenton Lee and Kristina Toutanova, “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding,” NAACL-HLT, 2019. Paper
4. Google Research, *BERT: original implementation*, software, 2018. Classification, Question answering, Optimizer
5. Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Ng and Christopher Potts, “Recursive Deep Models for Semantic Compositionality Over a Sentiment Treebank,” EMNLP, 2013. Paper

6. Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng and Christopher Potts, “Learning Word Vectors for Sentiment Analysis,” ACL-HLT, 2011. Paper
7. Samuel R. Bowman, Gabor Angeli, Christopher Potts and Christopher D. Manning, “A large annotated corpus for learning natural language inference,” EMNLP, 2015. Paper
8. Adina Williams, Nikita Nangia and Samuel R. Bowman, “A Broad-Coverage Challenge Corpus for Sentence Understanding through Inference,” NAACL-HLT, 2018. Paper
9. Suchin Gururangan, Swabha Swayamdipta, Omer Levy, Roy Schwartz, Samuel R. Bowman and Noah A. Smith, “Annotation Artifacts in Natural Language Inference Data,” NAACL-HLT, 2018. Paper
10. Adam Poliak, Jason Naradowsky, Aparajita Haldar, Rachel Rudinger and Benjamin Van Durme, “Hypothesis Only Baselines in Natural Language Inference,” *SEM, 2018. Paper
11. R. Thomas McCoy, Ellie Pavlick and Tal Linzen, “Right for the Wrong Reasons: Diagnosing Syntactic Heuristics in Natural Language Inference,” ACL, 2019. Paper
12. Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev and Percy Liang, “SQuAD: 100,000+ Questions for Machine Comprehension of Text,” EMNLP, 2016. Paper
13. Pranav Rajpurkar, Robin Jia and Percy Liang, “Know What You Don’t Know: Unanswerable Questions for SQuAD,” ACL, 2018. Paper
14. Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy and Samuel R. Bowman, “GLUE: A Multi-Task Benchmark and Analysis Platform for Natural Language Understanding,” ICLR, 2019. Paper
15. Alex Wang, Yada Pruksachatkun, Nikita Nangia, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy and Samuel R. Bowman, “SuperGLUE: A Stickier Benchmark for General-Purpose Language Understanding Systems,” NeurIPS, 2019. Paper
16. Marius Mosbach, Maksym Andriushchenko and Dietrich Klakow, “On the Stability of Fine-tuning BERT: Misconceptions, Explanations, and Strong Baselines,” ICLR, 2021. Paper
17. Chuan Guo, Geoff Pleiss, Yu Sun and Kilian Q. Weinberger, “On Calibration of Modern Neural Networks,” ICML, 2017. Paper
18. Shrey Desai and Greg Durrett, “Calibration of Pre-trained Transformers,” EMNLP, 2020. Paper
19. scikit-learn developers, “Classification metrics,” scikit-learn documentation, current online edition. Documentation
20. Nils Reimers and Iryna Gurevych, “Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks,” EMNLP-IJCNLP, 2019. Paper
21. Shervin Minaee, Nal Kalchbrenner, Erik Cambria, Narjes Nikzad, Meysam Chenaghlu and Jianfeng Gao, “Deep Learning–based Text Classification: A Comprehensive Review,” ACM Computing Surveys 54(3), 2021. Survey
22. Wenxuan Zhang, Yue Deng, Bing Liu, Sinno Jialin Pan and Lidong Bing, “Sentiment Analysis in the Era of Large Language Models: A Reality Check,” Findings of NAACL, 2024. Paper
23. Benjamin Warner, Antoine Chaffin, Benjamin Clavié, Orion Weller, Oskar Hallström, Said Taghadouini et al., “Smarter, Better, Faster, Longer: A Modern Bidirectional Encoder for Fast, Memory Efficient, and Long Context Finetuning and Inference,” ACL, 2025. Paper
24. Yupeng Chang et al., “A Survey on Evaluation of Large Language Models,” ACM Transactions on Intelligent Systems and Technology, 2024. Survey
25. Wayne Xin Zhao et al., “A Survey of Large Language Models,” Frontiers of Computer Science, 2026. Updated survey

26. Ilya Loshchilov and Frank Hutter, “Decoupled Weight Decay Regularization,” ICLR, 2019. Paper

NLP 12・句法與語意任務

以 CKY 剖析成分、以轉移或生成樹剖析依存，標註語意角色、解決共指、概覽 AMR，並以各自的指標評估每種結構。

網站版本: https://blog.shyandram.dev/zh-tw/learn/nlp/syntactic-semantic-tasks/

你將學到

- 定義上下文無關文法，用 CKY 圖表辨識並列舉剖析，再以機率文法或神經跨度分數在其中選擇。
- 以轉移系統（arc-standard）與圖式解碼建立依存樹，並解釋為何逐詞獨立選中心詞需要樹限制。
- 標註述詞—論元結構（語意角色），在擷取的關係中保留作用範圍與來源，並讀懂一個小型 AMR 圖。
- 以先行詞模型把提及連成實體，並在脈絡不足時保留歧義。
- 用各自的指標評估每種結構：PARSEVAL 括號、UAS ／ LAS、角色 F1、MUC ／ B^3 ／ CEAF 與 CoNLL F1，以及 Smatch。
- 說明預訓練編碼器與大型語言模型如何改變這些任務，以及哪些事沒有改變。

先看全貌

語言表面是線性的，解讀卻有結構。句子以有順序的詞序列輸入，但讀者會把詞分組、辨識彼此的關係、判斷誰對誰做了什麼，並把一句中的「it」連到另一句的「the pump」。「We saw robots with cameras」可能描述帶著相機的機器人，也可能描述觀察時使用的器材。單靠詞清單無法選定解讀。

> **用最簡單的話說**
>
> 句子像一組建構玩具。詞是零件，文法說明怎樣連接。剖析顯示連接方式；角色標註說明哪個零件是動手的、哪個是被處理的；共指則發現兩個零件其實是同一個玩具被看了兩次。有時相同零件能做出兩種合理建構。好的解釋應呈現兩者，再詢問哪個符合情境。

本章收集自然語言處理中經典的**結構化預測**任務。在 NLP 06 中，輸出是每個位置一個標籤；這裡的輸出則是一棵樹、一張圖或提及的分割，而且某處的選擇會限制其他地方：一個詞只能有一個中心詞、一個跨度只能屬於述詞的一個論元、共指連結必須形成一致的群組。McRoy 關於語法、語意與篇章的章節，以及另附的圖表剖析附錄，提供語言學觀點 [1]；Jurafsky 與 Martin 關於成分剖析、依存剖析、語意角色與共指的章節，提供演算法觀點 [2]。成分、依存、量詞範圍等語言學類別本身，在 NLP 02 介紹。

這些結構有助擷取誰做了什麼、定位修飾語，並建立區分流暢度與結構理解的測試。它們無法處理所有問題：合文法的句子可能是假的，不完整的訊息也可能清楚傳達意思。我們把每種結構視為範圍明確的表示，而不是理解的定義。

成分與上下文無關文法

直觀說法。片語結構把詞組成能遞迴結合的單位。文法規定允許的組合；剖析則是某個句子的推導。

上下文無關文法（context-free grammar）可定義為：

$$G = (\mathcal{N}, \Sigma, \mathcal{R}, S).$$

其中 G 是文法，$\mathcal{N}$ 是非終結類別集合，Σ 是終結詞彙表，$\mathcal{R}$ 是改寫規則集合，$S \in \mathcal{N}$ 是起始類別。非終結符代表名詞片語等結構，終結符則是觀察到的詞。上下文無關規則替換一個非終結符，而不檢查周圍符號。

原創教學文法可使用 $S \to NP\ VP$、$VP \to V\ NP$、$PP \to P\ NP$ 與 $NP \to NP\ PP$。其中 S 表示句子，NP 表示名詞片語，VP 表示動詞片語，PP 表示介系詞片語，V 表示動詞類別，P 表示介系詞類別。這是教學文法的符號選擇，不是完整英文文法。Penn Treebank 等樹庫提供大量以這類樹標註的句子，文法與剖析器即由此學習 [4]。

遞迴規則能以有限規格產生無界的句子家族，但同樣的遞迴也可能讓天真的搜尋反覆做相同工作或無限循環，因此剖析器需要搜尋策略與緊湊的部分分析表示。

應區分涵蓋範圍與偏好。規則系統可能接受句子卻給出多個分析，也可能因缺少詞彙條目而無法剖析完全合理的句子。剖析失敗只表示所實作的文法與搜尋沒有找到分析，不表示使用者寫了無意義的內容。明確文法的好處是可檢查性：能顯示哪個規則連結了跨度，以及歧義出現在哪裡 [1]。

依跨度進行圖表剖析：CKY 演算法

由上而下的假設與由下而上的證據

直觀說法。剖析器可以從句子預期包含什麼開始，也可以從觀察詞已支持什麼開始。兩個方向都需要避免重複工作。

由上而下搜尋從起始類別展開產生式，形成預期子節點。它能排除不符合預期的詞彙分析，但不好的展開選擇可能導致很長的失敗路徑；若遞迴產生式把第一個子節點展成同一類別，就可能在還沒消耗輸入前反覆遇到同一個未解預期。由下而上搜尋從觀察詞建立跨度支持的類別，可能產生許多不屬於任何完整句子的片語，但中間結果也可能被多個成功分析共用。

因此，設計問題不只是選哪個方向「比較好」，而是部分工作如何表示，以及兩個部分分析何時算同一搜尋狀態。圖表儲存完成跨度，另一條路徑抵達既有狀態時不必重新計算。不過記憶化本身不會解決歧義：辨識只需要每個跨度與類別一個布林值，偏好的剖析需要分數與回溯指標，所有選項則需要以森林保留不同推導連結。

跨度遞迴式

直觀說法。儲存每個連續跨度可能的類別。把短跨度結合成長跨度，並讓所有完整剖析共用結果。

Cocke–Kasami–Younger（CKY）演算法是這個想法的標準動態規劃 [2]。它需要二元規則：喬姆斯基正規形（Chomsky normal form）的文法只有 $A \to B\,C$ 與 $A \to a$ 兩種規則，而任何不含空產生式的上下文無關文法都能轉成此形式 [2]。採用半開邊界：跨度 $[i, j)$ 包含位置 i 到 $j-1$ 的詞元。辨識可寫成：

$$C[i,j,A] = \bigvee_{\substack{A \to BC \in \mathcal{R} \\ i<k<j}} \left(C[i,k,B] \wedge C[k,j,C] \right).$$

其中 $C[i,j,A]$ 是類別 A 能涵蓋跨度 $[i,j)$ 的布林敘述，$\mathcal{R}$ 是二元規則集合，$A \to BC$ 結合左類別 B 與右類別 C，k 是內部切分邊界，$\wedge$ 表示兩個子節點都必須存在，$\bigvee$ 表示至少一組規則與切分成功。字母 C 同時出現在圖表符號與右子節點類別中，可由中括號與類別引數分辨角色。

先以詞彙規則初始化長度為一的格子，再依跨度長度、起點、切分點與相容規則迭代。子節點比父節點短，因此已經完成。一元規則與空產生式需要額外處理；本章可執行文法刻意不包含兩者。

這個演算法能辨識輸入，但辨識本身無法解釋樹。應以回溯指標或緊湊森林儲存成功組合。多個父節點可以共享子跨度，而不必複製完整子樹。高度歧義句子若展開所有完整樹，即使辨識是多項式時間，展開仍可能需要指數時間。固定二元文法下，跨度與切分巢狀迴圈對詞元長度呈立方關係；圖表有平方數量的跨度，每個跨度包含可能類別。

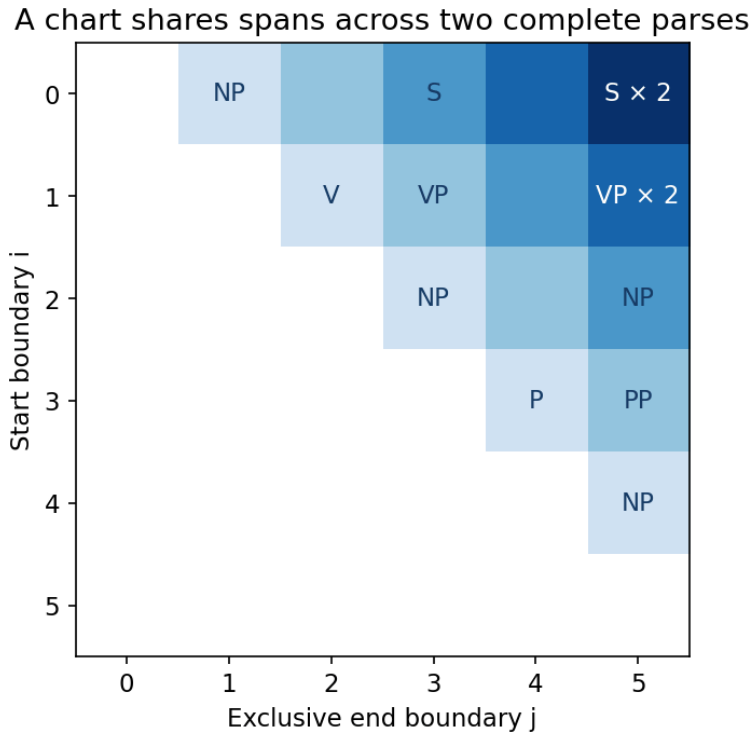


圖 *1* — 圖表儲存可重用的跨度；完整句子格可以包含不只一種分析。

保留兩種分析的剖析器

文法允許兩種附接時，就回報兩種選項。剖析器不應只因某個選項先被加入，就丟掉歧義。以下獨立剖析器在圖表格子中儲存完整小樹。它刻意比緊湊森林實作簡單；適合五個詞的示範，而前面幾段已說明為何明確列舉不適合大型歧義輸入。

```
from collections import defaultdict

words = "we saw robots with cameras".split()
lexical = {
    "we": ["NP"], "saw": ["V"], "robots": ["NP"],
    "with": ["P"], "cameras": ["NP"],
}
rules = [
    ("S", "NP", "VP"),
    ("VP", "V", "NP"),
    ("VP", "VP", "PP"),
    ("NP", "NP", "PP"),
    ("PP", "P", "NP"),
]
chart = defaultdict(lambda: defaultdict(list))
for i, word in enumerate(words):
    for category in lexical.get(word, []):
        chart[i, i + 1][category].append((category, word))
for length in range(2, len(words) + 1):
    for start in range(len(words) - length + 1):
        end = start + length
        for split in range(start + 1, end):
            for parent, left, right in rules:
                for left_tree in chart[start, split].get(left, []):
                    for right_tree in chart[split, end].get(right, []):
                        tree = (parent, left_tree, right_tree)
                        if tree not in chart[start, end][parent]:
                            chart[start, end][parent].append(tree)

def bracket(tree):
    if len(tree) == 2:
        return f"({tree[0]} {tree[1]})"
    return f"({tree[0]} {bracket(tree[1])} {bracket(tree[2])})"

parses = chart[0, len(words)].get("S", [])
assert len(parses) == 2
for tree in parses:
    print(bracket(tree))
print("complete parses:", len(parses))
```

```
(S (NP we) (VP (V saw) (NP (NP robots) (PP (P with) (NP cameras)))))
(S (NP we) (VP (VP (V saw) (NP robots)) (PP (P with) (NP cameras))))
complete parses: 2
```

一棵樹把介系詞片語放進受詞名詞片語，另一棵則讓它修飾動詞片語。兩者都按順序使用每個詞，並符合文法。要選擇其中之一，可能需要周圍場景、先前篇章，或超出目前規則的詞彙偏好。

加入未知詞，某個詞彙格子可能變空。正確做法是診斷涵蓋範圍。正式服務可以使用備援標記器、允許明確未知類別，或要求澄清，但每個備援都會改變接受的語言，應記錄並測試。

結構上的機率

直觀說法。文法說明哪些分析存在；機率模型可以偏好部分分析，並彙整各選項的不確定性。

機率式上下文無關文法（PCFG）為每個產生式配置機率，並在相同父類別的規則間正規化 [40]。一個推導具有：

$$p(\mathcal{T}) = \prod_{r \in \mathcal{T}} p(r), \qquad \hat{\mathcal{T}} = \operatorname*{argmax}_{\mathcal{T} \in \mathcal{D}(\mathbf{x})} p(\mathcal{T}).$$

其中 $\mathcal{T}$ 是完整推導樹，r 索引樹中每次產生式使用，$p(r)$ 是規則機率，$\mathcal{D}(\mathbf{x})$ 是終結輸出為輸入 $\mathbf{x}$ 的允許剖析集合，$\hat{\mathcal{T}}$ 是機率最高的剖析。同一規則重複使用時，乘積會分別計入。

把圖表中的選項改成加總，就能計算各推導的總機率質量（inside 演算法）；改成最大值，就能計算最佳推導（機率式 CKY）。兩者回答不同問題：具有許多中等機率剖析的句子，總機率可能高於只有一個強剖析的句子，最佳樹也不一定占後驗質量的大部分。以下程式對同樣五個詞同時計算兩者。

```
import math
from collections import defaultdict

words = "we saw robots with cameras".split()
lexical = {"we": "NP", "saw": "V", "robots": "NP", "with": "P", "cameras": "NP"}

def parse(p_np_pp):
    # All productions sharing a parent must sum to one, including lexical rules.
    assert 0 < p_np_pp < 1
    rules = {("S", "NP", "VP"): 1.0, ("PP", "P", "NP"): 1.0,
             ("VP", "V", "NP"): 0.6, ("VP", "VP", "PP"): 0.4,
             ("NP", "NP", "PP"): p_np_pp}
    lexical_prob = {"we": (1 - p_np_pp) / 3, "robots": (1 - p_np_pp) / 3,
                    "cameras": (1 - p_np_pp) / 3, "saw": 1.0, "with": 1.0}
    parent_mass = defaultdict(float)
    for (parent, _, _), probability in rules.items():
        parent_mass[parent] += probability
    for word, parent in lexical.items():
        parent_mass[parent] += lexical_prob[word]
    assert all(math.isclose(total, 1.0) for total in parent_mass.values())
    best = defaultdict(dict)    # (i, j) -> {category: (log prob, tree)}
    inside = defaultdict(lambda: defaultdict(float))
    for i, w in enumerate(words):
        best[i, i + 1][lexical[w]] = (math.log(lexical_prob[w]), f"({lexical[w]} {w})")
        inside[i, i + 1][lexical[w]] = lexical_prob[w]
    n = len(words)
```

```
    for length in range(2, n + 1):
        for i in range(n - length + 1):
            j = i + length
            for k in range(i + 1, j):
                for (a, b, c), p in rules.items():
                    if b in best[i, k] and c in best[k, j]:
                        score = math.log(p) + best[i, k][b][0] + best[k, j][c][0]
                        if score > best[i, j].get(a, (-math.inf,))[0]:
                            best[i, j][a] = (score, f"({a} {best[i, k][b][1]} {best[k,
                             ↪  j][c][1]})")
                        inside[i, j][a] += p * inside[i, k][b] * inside[k, j][c]
    score, tree = best[0, n]["S"]
    return tree, math.exp(score), inside[0, n]["S"]

for p in (0.2, 0.5):
    tree, p_best, p_total = parse(p)
    print(f"P(NP -> NP PP) = {p}: best = {p_best:.6f}, total = {p_total:.6f}")
    print("  ", tree)
    common = ((1 - p) / 3) ** 3
    assert math.isclose(p_total, common * (0.24 + 0.6 * p))
    assert math.isclose(p_best / p_total, max(0.24, 0.6 * p) / (0.24 + 0.6 * p))
```

```
P(NP -> NP PP) = 0.2: best = 0.004551, total = 0.006827
   (S (NP we) (VP (VP (V saw) (NP robots)) (PP (P with) (NP cameras))))
P(NP -> NP PP) = 0.5: best = 0.001389, total = 0.002500
   (S (NP we) (VP (V saw) (NP (NP robots) (PP (P with) (NP cameras)))))
```

三條詞彙 NP 規則平分遞迴規則用掉後剩餘的質量：每條機率為 $(1-p)/3$，其中 $p = p(NP \to NP\ PP)$。兩棵樹都包含這三個詞彙選擇，因此共同因子是 $c = ((1-p)/3)^3$。動詞附接的機率為 $c \times 0.4 \times 0.6$，名詞附接則為 $c \times 0.6p$。省略 c 不會改變**同一文法內**哪棵樹勝出，但相加後就不再是句子的機率。程式保留此因子，並檢查每個父類別的正規化。

當 p 越過 0.4 時，偏好的樹仍會翻轉。把某棵樹的聯合機率除以 inside 總和，才得到給定這些詞之後的後驗機率。因此，最佳樹機率、句子機率與後驗信心是三個不同量。這裡的附接偏好來自類別規則機率，而非詞彙上下文；下一節的神經評分器補入這項缺少的證據。

從樹庫估計的規則機率取決於標註慣例與領域。新聞樹庫與客服語料可能有不同片段模式與附接偏好。機率建模提供偏好機制，但不會消除領域依賴。

神經成分剖析：為跨度評分，再用圖表解碼

直觀說法。不再相乘規則機率，而是讓神經編碼器替每個跨度與標籤評分，再用同一張圖表找出分數最高的樹。

跨度式剖析器替每個候選成分評分，並搜尋總分最大的樹：

$$s(\mathcal{T}) = \sum_{(i,j,\ell)\in\mathcal{T}} s(i,j,\ell), \qquad \hat{\mathcal{T}} = \operatorname*{argmax}_{\mathcal{T}} s(\mathcal{T}).$$

其中 $\mathcal{T}$ 是視為帶標籤跨度 (i, j, ℓ) 集合的樹，ℓ 是半開跨度 $[i, j)$ 的標籤，$s(i, j, \ell)$ 是由跨度邊界上下文向量算出的分數，$\hat{\mathcal{T}}$ 是解碼得到的樹。Stern、Andreas 與 Klein 顯示，獨立替標籤與跨度評分可與傳統圖表動態規劃相容，也容許以遞迴切分輸入的貪婪由上而下解碼器 [5]。Kitaev 與 Klein 把該設計中的 LSTM 編碼器換成自注意力，發現分開位置與內容資訊能提升準確度，並報告在 Penn Treebank 上不使用外部資料時 93.55 F1、使用預訓練詞表示時 95.13 F1 [6]。後續研究再加入多語預訓練 [7]。

即使分數來自大型神經網路，輸出仍是一棵樹：圖表保證結構合法，編碼器則提供 PCFG 規則機率無法提供的詞彙證據。

依存結構與語法角色

直觀說法。依存分析直接連結詞，顯示哪個詞依附哪個詞，並標記語法關係。

基本依存樹可用中心詞索引表示：

$$h_t \in \{0, 1, \dots, T\} \setminus \{t\}, \qquad r_t \in \mathcal{D}.$$

其中 h_t 是詞 t 的中心詞索引，0 表示人工根節點，T 是句子長度，r_t 是該詞的關係標籤，$\mathcal{D}$ 是關係集合。有效基本樹還需要明確的根節點規則、連通性與無有向循環。

逐詞獨立選中心詞可能違反全域要求。兩個詞可能互相選為中心而形成循環；數個不連通部分可能沒有唯一句根。樹解碼器應落實預期結構，或由服務驗證並拒絕無效分析。

Universal Dependencies 提供跨語言的詞法、詞性與語法依存標註框架 [8]。類別設計希望兼顧跨語言比較與語言專屬差異。這是標註框架，不表示每種語言具有相同詞序或明示語法標記。

片語結構與依存關係強調不同資訊。成分樹清楚表示跨度分組；依存樹清楚表示詞彙中心與關係。相互轉換需要中心詞、標點與功能詞的約定，不能只當成換一種畫法。

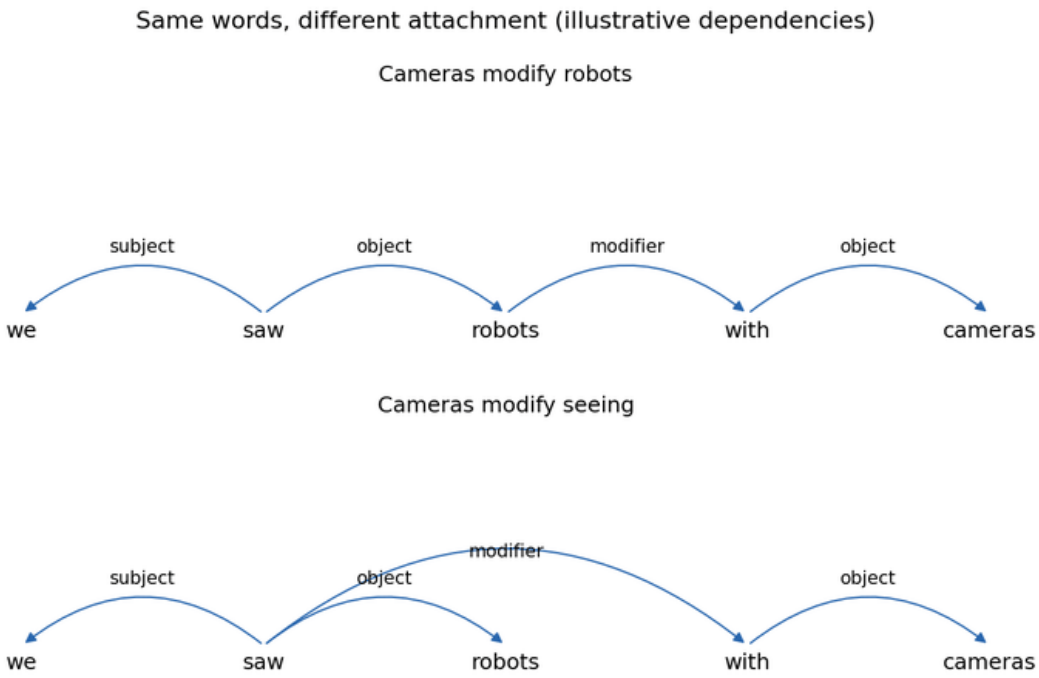


圖 2 — 相同詞的兩種附接。這些示意語法連結用來解釋歧義，並未複製樹庫中的圖。

對擷取而言，語法角色是有用證據，但不能與語意角色互換。被動句改變語法主語位置，卻保留誰對誰做了什麼。下方的語意角色小節會發展這個差別，說明依存箭頭為何不能單獨證明真實世界關係。

附接歧義揭露模型的歸納偏好

兩種相機附接共享詞彙，卻替看見事件配置不同參與者。名詞片語附接描述機器人的性質；動詞片語附接描述看見時的工具或情境。下游擷取器若選定名詞解讀，可能建立機器人—相機持有關係；工具解讀卻不支持該紀錄。附接差異之所以重要，是因為它改變關係，不只是括號長得不同。

如 PCFG 程式所示，文法可以允許兩者，機率模型仍會偏好其一。若只有父類別的規則機率，除非擴充狀態或評分表示，否則模型無法依 `saw`、`robots`、`cameras` 的詞彙身分改變偏好。這是容量限制；對相同分數擴大搜尋，不會創造缺少的詞彙證據。

相反的失敗是搜尋限制：模型可能已有實用分數，卻在形成完整樹之前，透過剪枝移除正確跨度。應分別診斷：參考結構能否表示、是否保留於候選建立階段，以及在留下的候選中是否得最高分。答案分別指向不同改善：擴充表示、調整剪枝或改進評分。

也可能真的缺少資訊：孤立句子中兩種閱讀可能都成立，標準樹只記錄語料選定的解讀。對擷取器而言，先保留兩種可能，等到檢查所需關係再決定，可能比早期定案更合適。

轉移式依存剖析

直觀說法。用一個堆疊由左到右讀句子，每一步從少數動作中選一個：把下一個詞移入堆疊，或把堆疊頂端兩項互相連結。分類器選動作，動作序列就建出樹。

Nivre 把確定性依存剖析器形式化為轉移系統，並分析以堆疊為基礎的變體（arc-standard 與 arc-eager，限於投射樹）以及以串列為基礎的變體，其中之一能處理非投射樹 [9]。**投射（projective）**樹是指弧線能畫在句子上方而不交叉的樹。在 Chen 與 Manning 使用的 arc-standard 系統中，組態 $c = (\sigma, \beta, A)$ 包含堆疊 σ、緩衝區 β 與弧集合 A，並有三種轉移 [10]：

$$\text{SHIFT}: \ (\sigma, w|\beta, A) \Rightarrow (\sigma|w, \beta, A),$$

$$\text{LEFT-ARC}_r: \ (\sigma|s_2|s_1, \beta, A) \Rightarrow (\sigma|s_1, \beta, A \cup \{s_1 \xrightarrow{r} s_2\}),$$

$$\text{RIGHT-ARC}_r: \ (\sigma|s_2|s_1, \beta, A) \Rightarrow (\sigma|s_2, \beta, A \cup \{s_2 \xrightarrow{r} s_1\}).$$

其中 s_1 是堆疊頂端，s_2 是其下一項，w 是緩衝區第一個詞，$\sigma|x$ 表示把 x 放在堆疊 σ 頂端，$h \xrightarrow{r} d$ 是從中心詞 h 指向依附詞 d、標籤為 r 的弧，$\Rightarrow$ 讀作「變成」。剖析開始時堆疊只有根、所有詞都在緩衝區；緩衝區清空且只剩根時結束。每個詞被移入一次、被連結一次，因此 n 個詞的句子恰好需要 $2n$ 次轉移。

訓練分類器時，把每棵標準樹轉成能產生它的轉移序列，稱為**靜態神諭（static oracle）**。以下規則是一種標準選擇：堆疊第二項的標準中心詞是頂端時向左連結；頂端的標準中心詞是第二項且頂端已收齊自己所有依附詞時向右連結；否則移入。

```
words = ["ROOT", "we", "saw", "robots", "with", "cameras"]
# Gold heads for the instrument reading: "with cameras" attaches to "saw".
gold = {1: 2, 2: 0, 3: 2, 4: 5, 5: 2}

def oracle(gold):
    stack, buffer, arcs, steps = [0], list(range(1, len(words))), {}, []
    def done(h):  # has h already collected all of its dependents?
        return all(d in arcs for d, head in gold.items() if head == h)
    while buffer or len(stack) > 1:
        if len(stack) >= 2 and gold.get(stack[-2]) == stack[-1]:
            dependent = stack.pop(-2)
            arcs[dependent] = stack[-1]; steps.append("LEFT-ARC")
        elif len(stack) >= 2 and gold[stack[-1]] == stack[-2] and done(stack[-1]):
            dependent = stack.pop()
            arcs[dependent] = stack[-1]; steps.append("RIGHT-ARC")
        else:
            stack.append(buffer.pop(0)); steps.append("SHIFT")
    return steps, arcs

steps, arcs = oracle(gold)
print(" ".join(steps))
print({words[d]: words[h] for d, h in sorted(arcs.items())})
assert arcs == gold and len(steps) == 2 * (len(words) - 1)
```

```
SHIFT SHIFT LEFT-ARC SHIFT RIGHT-ARC SHIFT SHIFT LEFT-ARC RIGHT-ARC RIGHT-ARC
{'we': 'saw', 'saw': 'ROOT', 'robots': 'saw', 'with': 'cameras', 'cameras': 'saw'}
```

五個詞用了十次轉移，與預測一致。`done` 檢查很重要：如果 `robots` 在 `cameras` 到達前就被連到 `saw` 並彈出，之後就不可能建出名詞附接的樹。歸約的順序編碼了樹。

剖析時由分類器取代神諭。Chen 與 Manning 把堆疊與緩衝區固定位置上的詞、詞性與弧標籤嵌入，輸入一個使用立方活化函數（cube activation）的單隱藏層網路，再以 softmax 選擇轉移；他們的貪婪剖析器很快，但如他們所說，會受到錯誤傳遞影響，因為早期錯誤動作無法撤銷 [10]。Kiperwasser 與 Goldberg 則為每個詞元計算一個與剖析器共同訓練的雙向 LSTM 向量，再串接少數幾個向量作為特徵；同一種表示同時適用於貪婪轉移剖析器與全域最佳化的圖式剖析器 [12]。轉移剖析器的時間與句長成線性；代價是局部、由左到右的決策程序。

圖式依存剖析

直觀說法。一次替所有可能的中心詞—依附詞弧評分，再選出能構成樹、總分最高的一組弧。

McDonald 等人把加權依存剖析形式化為：在以詞加上根為節點的有向圖中，尋找最大生成樹 [11]：

$$\hat{\mathbf{h}} = \operatorname*{argmax}_{\mathbf{h} \in \mathcal{Y}(\mathbf{x})} \sum_{t=1}^{T} s(h_t, t).$$

其中 $\mathbf{h} = (h_1, \dots, h_T)$ 為每個詞指定一個中心詞，$\mathcal{Y}(\mathbf{x})$ 是對句子 $\mathbf{x}$ 構成有效樹的中心詞指派集合，$s(h, t)$ 是從中心詞 h 指向依附詞 t 的弧分數。在投射樹上，Eisner 演算法以 $O(n^3)$ 時間求解；在包含非投射樹的所有樹上，Chu–Liu–Edmonds 最大生成樹演算法以 $O(n^2)$ 時間求解 [11]。

Dozat 與 Manning 在深層雙向編碼器之上，把弧評分器做成**雙仿射（biaffine）**形式 [13]：

$$s(h, t) = \mathbf{r}_t^{(\text{dep})\top} U \, \mathbf{r}_h^{(\text{head})} + \mathbf{r}_h^{(\text{head})\top} \mathbf{u}.$$

其中 $\mathbf{r}_t^{(\text{dep})}$ 與 $\mathbf{r}_h^{(\text{head})}$ 是依附詞與候選中心詞上下文向量的低維投影，U 是替兩者互動評分的學習矩陣，$\mathbf{u}$ 是讓每個詞具有成為中心詞先驗傾向的學習向量。第二個雙仿射分類器則預測所選弧的關係標籤。評分器一次就產生完整的 $(T+1) \times T$ 弧分數矩陣。

樹限制帶來什麼？以下程式為我們的句子使用虛構分數，先讓每個詞獨立挑選最佳中心詞，再列舉所有單一根的樹——只對極小輸入可行，卻讓解碼器的工作看得見。最後計算下一節定義的 UAS 與 LAS。

```
import itertools
import numpy as np

words = ["ROOT", "we", "saw", "robots", "with", "cameras"]
n = len(words) - 1
# score[h, d]: invented arc scores (head h -> dependent d), not a trained model.
score = np.full((n + 1, n + 1), -2.0)
score[0, 2], score[2, 1], score[2, 3], score[5, 4] = 3.0, 2.0, 2.0, 2.0
score[3, 5], score[2, 5] = 1.5, 1.2               # robots vs saw for "cameras"
score[4, 5] = 1.8                                 # tempting but creates a cycle
np.fill_diagonal(score, -np.inf)

def is_tree(heads):                               # heads[d-1] is the head of d
```

```
    if sum(h == 0 for h in heads) != 1:
        return False
    for d in range(1, n + 1):
        seen, node = set(), d
        while node != 0:
            if node in seen:
                return False
            seen.add(node); node = heads[node - 1]
    return True

greedy = [int(np.argmax(score[:, d])) for d in range(1, n + 1)]
total = lambda hs: sum(score[h, d] for d, h in enumerate(hs, start=1))
trees = [hs for hs in itertools.product(range(n + 1), repeat=n)
         if all(h != d for d, h in enumerate(hs, 1)) and is_tree(hs)]
best = max(trees, key=total)
print("greedy heads:", greedy, "valid tree:", is_tree(greedy))
print("best tree:   ", list(best), "score:", round(total(best), 2))
print("trees enumerated:", len(trees))

gold = [2, 0, 2, 5, 2]
labels_gold = ["nsubj", "root", "obj", "case", "obl"]
labels_pred = ["nsubj", "root", "obj", "mark", "nmod"]
uas = np.mean([p == g for p, g in zip(best, gold)])
las = np.mean([p == g and lp == lg for p, g, lp, lg
               in zip(best, gold, labels_pred, labels_gold)])
print(f"UAS = {uas:.2f}, LAS = {las:.2f}")
```

```
greedy heads: [2, 0, 2, 5, 4] valid tree: False
best tree:    [2, 0, 2, 5, 3] score: 10.5
trees enumerated: 625
UAS = 0.80, LAS = 0.60
```

貪婪選擇讓 `with` 與 `cameras` 互為中心詞，形成與根不連通的兩詞循環（圖 3）。在五個詞的 625 棵單一根樹中，最佳者打破循環，把 `cameras` 連到 `robots`——這是有效的樹，卻是名詞解讀，而標準標註選的是工具解讀。最大生成樹演算法不需列舉就能找到相同答案。

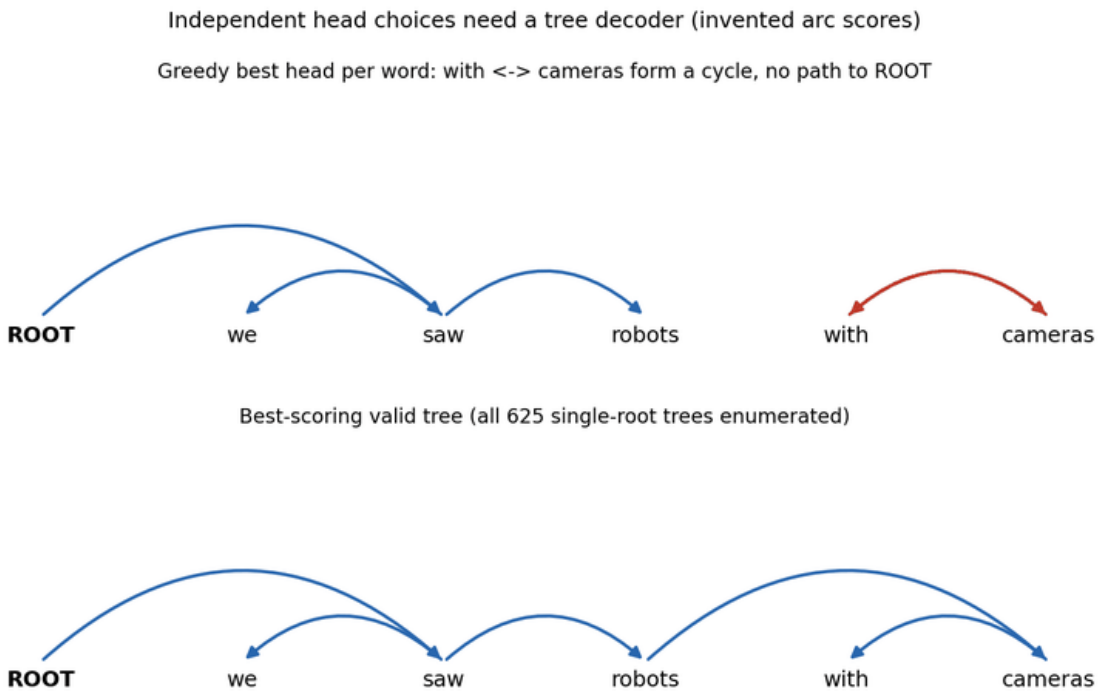


圖 3 — 獨立選出的最佳中心詞可能形成循環；樹解碼器恢復有效結構。分數為虛構，與程式一致。

圖式與轉移式剖析器的取捨不同：前者在弧分數上做全域搜尋，解碼為平方或立方時間；後者做線性時間的局部決策。Kiperwasser 與 Goldberg 的共用 BiLSTM 特徵同時服務兩個家族 [12]。不論用哪一族，結構保證都來自解碼器或轉移系統，而不是編碼器。

評估剖析器：PARSEVAL、UAS 與 LAS

直觀說法。比較剖析器承諾產生的表示，並檢查斷詞、標註或脈絡改變答案的情況。

最初的 PARSEVAL 程序先移除標點等材料，再比較候選剖析與樹庫剖析，計算**交叉括號**分數——候選括號與參考括號交叉的次數——以及**召回率**，即共同括號數除以參考括號數 [3]。今日標準是帶標籤括號的精確率、召回率與 F1 [2]：

$$P_{\text{lab}} = \frac{|\mathcal{B}(\hat{T}) \cap \mathcal{B}(T^{*})|}{|\mathcal{B}(\hat{T})|}, \qquad R_{\text{lab}} = \frac{|\mathcal{B}(\hat{T}) \cap \mathcal{B}(T^{*})|}{|\mathcal{B}(T^{*})|}.$$

其中 $\mathcal{B}(T)$ 是樹 T 的帶標籤成分 (i, j, ℓ) 集合，$\hat{T}$ 是預測樹，T^{*} 是參考樹，F1 是兩者的調和平均。對我們的兩個剖析，兩棵樹只有一個帶標籤括號不同：名詞附接樹有 $(2, 5, NP)$，動詞附接樹有 $(1, 3, VP)$。計入預終結符時，每棵樹有九個帶標籤成分，因此任一棵對另一棵評分，精確率與召回率都是 8/9。應說明是否包含詞彙預終結符、標點與根；不同納入規則會改變分母。

對依存結構，**無標籤附接分數**（UAS）是中心詞正確的詞所占比例，**帶標籤附接分數**（LAS）還要求關係正確：

$$\mathrm{LAS} = \frac{1}{M}\sum_{t \in \mathcal{J}} \mathbf{1}[(\hat{h}_t, \hat{r}_t) = (h_t, r_t)].$$

其中 LAS 是帶標籤附接分數，$\mathcal{J}$ 是評估詞位置集合，$M = |\mathcal{J}|$ 是集合大小，$\hat{h}_t$ 與 $\hat{r}_t$ 是預測中心詞與關係，h_t 與 r_t 是參考中心詞與關係，$\mathbf{1}[\cdot]$ 是指示函數；UAS 則去掉關係條件。在程式中，`cameras` 的中心詞錯誤（兩個分數都扣分），`with` 的中心詞正確但標籤錯誤（只有 LAS 扣分），因此得到 0.80 與 0.60。

套用這些公式前，必須固定詞元對齊。CoNLL 2018 共享任務要求系統把原始文字剖析成多種語言的 Universal Dependencies，測試輸入不附任何標準標註 [14]。系統因此必須自行切分與斷詞，而預測詞必須先與標準詞對齊，附接分數才有意義。如果標準標註有兩個詞，斷詞器卻輸出一個，就無法正常解讀附接比較。應報告切分不符，並說明如何處理對齊，不要默默刪除困難樣本。

建立小型診斷配對：保持詞彙相近但移動修飾語，加入巢狀子句，把主動句改為被動句，或加入片段與混合語言片語。即使總體分數穩定，這些測試仍能揭露模型結構在什麼條件下改變。句子確實有歧義時，不要把一棵標準樹視為絕對確定。語料需要選定慣例或經裁決的分析，但其他分析可能也合理。對應用而言，應問分歧是否改變擷取事實或行動，讓剖析錯誤與實際後果連結。

語意角色標註：誰對誰做了什麼

直觀說法。同一個事件可以用不同語法表達。應擷取事件角色，不能假定第一個名字永遠是行動者。

簡單的雙參與者述詞可寫成：

$$\mathrm{repair}(\mathrm{Mina}, \mathrm{sensor}).$$

其中 repair 是關係名稱；第一個論元是修理者；第二個是被修理物件。「Mina 修好了感測器」與「感測器被 Mina 修好了」可以共用這種表示，雖然表面的主詞不同。語法提供線索，角色解讀還必須考慮述詞與句型 [1]。

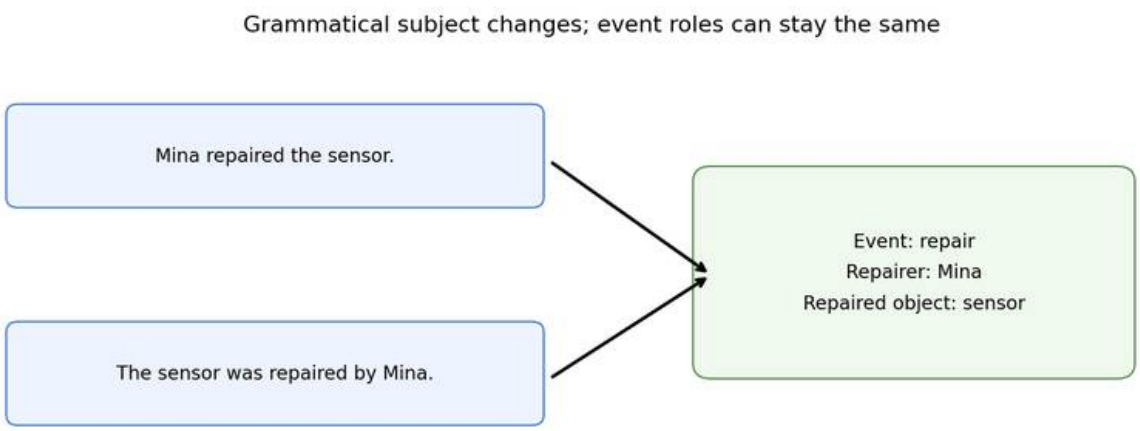


圖 *4* — 表面主詞可以改變，事件角色則保持相同。例句與紀錄皆為合成示例。

並非每個動詞都允許所有變換，也不是每個論元都明說。「感測器修好了」沒有指明修理者。若系統填入前文最近提到的人，就是做了上下文推論，而非直接擷取。這個推論可能合理，但需要另外的信心政策與證據契約。當應用未授權推論時，應保留明確的未知角色。

論元與修飾語也有差別。在「Mina 週二在臺北修好了感測器」中，修理者與被修理物件參與核心事件，地點與時間則補充事件。事件結構可包含四個欄位，但應說明哪些是必填。選填欄位能避免擷取器為滿足過度嚴格的格式而編造細節。

PropBank 角色與 SRL 任務

Gildea 與 Jurafsky 把自動語意角色標註建構為剖析句子上的統計任務：把帶角色的片語對應到剖析成分，從詞與樹抽取特徵，再估計每種可能角色的機率 [15]。**PropBank** 接著為 Penn Treebank 的動詞標註述詞專屬的編號論元：每個動詞義項有一個框架集，其中 `Arg0` 通常是施事般的參與者、`Arg1` 是受事或主題，更高編號則依動詞定義；附加語則用 `ArgM` 標籤，例如表示否定的 `ArgM-NEG` 與表示情態的 `ArgM-MOD` [16]。編號正是重點：在「an amendment offered by Rep. X」中，amendment 雖是語法主詞，仍是 *offer* 的 `Arg1` [16]。CoNLL-2005 共享任務以 PropBank 標準化 SRL 評估：預測論元只有在跨度與角色都正確時才算對，系統以論元層級的精確率、召回率與 F1 比較 [17]。

語意角色能連接不同句型，卻不保證各種資源的標註一致。有些資源使用述詞專屬角色，有些使用通用角色。合併訓練語料前，先比較定義與例子。機械式重新命名標籤，可能掩蓋工具、原因、受益者與隱含參與者的標註差異。

把 SRL 視為以述詞為條件的標記

He 等人把 SRL 寫成每個述詞各做一次的 BIO 序列標記：輸入標出哪個詞是述詞，深層 highway BiLSTM 搭配**受限解碼**——例如禁止沒有前置 `B-` 的 `I-` 標籤——標記論元跨度 [18]。他們的八層集成模型在 CoNLL-2005 測試集上報告 83.2 F1，並發現深層模型能找回長距離依存，卻仍會犯出

乎意料的明顯錯誤，句法資訊仍有幫助空間 [18]。換成預訓練編碼器後，同一設計就是 NLP 11 的詞元標記任務頭再加上述詞指示。以下程式把角色跨度轉成標籤序列再轉回來，並拒絕無效序列。

```
tokens = "Mina said the sensor was repaired by Lena".split()
# One frame per predicate: (start, end) token spans, end exclusive.
frames = {
    "said":     {"ARG0": (0, 1), "ARG1": (2, 8)},
    "repaired": {"ARG1": (2, 4), "ARG0": (6, 8)},
}

def to_bio(frame, n):
    tags = ["O"] * n
    for role, (i, j) in frame.items():
        tags[i] = "B-" + role
        tags[i + 1:j] = ["I-" + role] * (j - i - 1)
    return tags

def from_bio(tags):
    spans, start, role = {}, None, None
    for k, tag in enumerate(tags + ["O"]):
        if start is not None and not tag.startswith("I-" + role):
            spans[role] = (start, k); start = None
        if tag.startswith("B-"):
            start, role = k, tag[2:]
        elif tag.startswith("I-") and start is None:
            raise ValueError(f"I- tag without B- at position {k}")
    return spans

for predicate, frame in frames.items():
    tags = to_bio(frame, len(tokens))
    tags[tokens.index(predicate)] = "B-V"
    print(f"{predicate:9s}", " ".join(tags))
    assert {r: s for r, s in from_bio(tags).items() if r != "V"} == frame
try:
    from_bio(["O", "I-ARG1", "I-ARG1"])
except ValueError as error:
    print("invalid:", error)
```

```
said      B-ARG0 B-V B-ARG1 I-ARG1 I-ARG1 I-ARG1 I-ARG1 I-ARG1
repaired  O O B-ARG1 I-ARG1 O B-V B-ARG0 I-ARG0
invalid: I- tag without B- at position 1
```

同一句話產生兩個框架。對 *said*，Mina 是說話者，整個轉述子句是 `ARG1`。對 *repaired*，Lena 是修理者，感測器雖是語法主詞，仍是 `ARG1`。只讀 *repaired* 框架的擷取器，會把完成的修理記成事實；*said* 框架則顯示修理是 Mina 的轉述。這正是下方關係紀錄必須保留的歸屬差別。

語意表示中的作用範圍

直觀說法。「every」、「some」與「not」這類小詞，對陳述的改變可能比名字還大。語意剖析器必須記錄它們修飾什麼，否則輸出回答的是錯誤問題。

NLP 02 以一個實作反模型介紹量詞範圍與否定。這裡看作用範圍對任務輸出代表什麼。「Every technician checked a sensor」有一種閱讀是每位技師可檢查不同感測器，另一種閱讀是有一個感測器被所有人檢查。「Mina did not check every sensor」的形式是

$$\neg \forall z\,[S(z) \Rightarrow C(m,z)] \quad \equiv \quad \exists z\,[S(z) \wedge \neg C(m,z)].$$

其中 z 遍歷實體，$S(z)$ 表示感測器，$C(m,z)$ 表示 Mina 檢查該實體，m 指 Mina，$\neg$ 表示否定，$\equiv$ 表示邏輯等價。一個反例感測器，就足以讓全稱陳述不成立。「Mina checked no sensor」則排除每一次成功檢查。否定的位置決定哪種證據能反駁陳述。

文件資料庫通常沒有完整知識：沒有擷取到檢查紀錄，可能是未報告，不代表沒發生。回答全稱問題時，這個差別很重要。觀察清單可支持「這些感測器已檢查」，但要成立「所有感測器都已檢查」，還需要完整的相關感測器清單與全面涵蓋。反之，找到一個明確未檢查的感測器，就足以反駁完整性。作用範圍推理與證據完整性必須一起考慮。同樣地，「沒有找到故障證據」描述觀察或搜尋結果，不應自動轉成「故障不可能發生」。

範圍有限的擷取器遇到不支援的作用範圍句型，應拒絕判定或送交覆核。能處理「沒有修理」的規則，仍可能誤解「沒有說 Mina 修理了」：系統必須知道被否定的是什麼，而這正是 PropBank 的 `ArgM-NEG` 附在特定述詞上的資訊 [16]。

組合意義是對角色與作用範圍進行計算

理解組合的一個實用方式，是問每個片語還缺什麼。`Mina` 指定一個參與者；`repaired` 要求特定角色的參與者；`repaired a sensor` 用存在條件處理受詞要求，仍留下行動者等待填入；結合主詞後才補完該要求。這種組合順序說明，只辨認相同三個內容詞還不夠，角色必須正確綁定 [1]。

「Mina repaired a sensor」的一種受限邏輯閱讀為：

$$\exists z\,[S(z) \wedge R(m,z)].$$

其中 z 遍歷可能實體，$S(z)$ 表示 z 是感測器，$R(m,z)$ 表示參與者 m 修復 z，m 指 Mina。存在量詞表示至少有一個合適實體，不會識別是哪個感測器，也不表示所有感測器都修好了。這個簡單閱讀刻意省略時態與事件時間；它們是額外語意欄位，不是符號暗中提供的事實。

「Only Mina repaired the sensor」限制其他可能參與者；「Mina tried to repair the sensor」描述嘗試，不保證修復完成；「Mina stopped repairing the sensor」改變事件的時間解讀。只有行動者與對象的紀錄，無法表達這些差異。這提供選擇表示的具體標準：列出必須回答的問題，檢查是否有兩個需要不同答案的來源陳述，卻被壓成同一紀錄。如果嘗試修復與完成修復都變成 `repair(Mina, sensor)`，這個結構就太粗，無法回答完成狀態。序列化已經造成資訊損失時，更強編碼器也無法避免。

關係、事件與來源追溯

直觀說法。關係連結實體，事件則以一次發生為中心組織參與者。兩者都需要證據，也都可能在實體區段全對的情況下出錯。

二元關係紀錄可寫成：

$$q = (r, h, t, s).$$

其中 q 是紀錄；r 是關係類型；h 是頭實體或提及；t 是尾實體或提及；s 指向支持證據。頭與尾只是次序慣例，不是語言學主張。雇用關係中，交換員工與雇主就改變了紀錄。結構描述必須定義方向，評估也必須檢查。

事件擷取在觸發詞或事件類型之外，還包含買方、賣方、物品、金額與時間等角色——也就是限縮在應用結構描述內的語意角色標註。交易句可能提到多個價格，只有其中一個是實際付款金額。共同選擇角色配置能避免部分矛盾，但若來源本身含糊，仍需要更多上下文。應用允許時，可保留多個解讀或送交覆核。

來源追溯應包含文件身分、來源版本與支持區段。沒有版本，位移可能指向後來編輯過的文件；沒有歸屬——就像上面的 *said* 框架——引述的指控會變成直接事實。這些是紀錄管線的性質，不是換大編碼器就會自動解決的問題。

應分階段評估——提及偵測、類別預測、關係選擇與最終受支持紀錄——並且除了正確提及，也要用預測提及評估關係：測試時提供完美輸入，可能掩蓋實際部署管線的失敗率。

篇章、提及與共指

直觀說法。意義會跨越句子延續。共指消解判斷哪些提及指向同一實體，讓關於「the pump」與「it」的事實落在同一筆紀錄上。

篇章（discourse）是跨子句或句子組織的語言。例如「幫浦變熱了。操作員停止了它。」介紹事件、行動者與物件指涉，可能暗示對升溫的回應，卻不一定證明升溫造成停止。比較「The pump stopped **because** the valve closed」、「The pump stopped **after** the valve closed」與「The pump stopped **although** the valve remained open」：第一句提出因果解釋，第二句排列事件先後而不建立因果，第三句標示與預期的對比 [1]。摘要器或擷取器加入「because」，就是在增加主張。關係還有作用範圍：在「Mina said the pump stopped because the valve closed」中，解釋可能屬於 Mina 的報告，而不是獨立建立的發現。

提及（mention）是指向實體、事件或概念的文字區段；先行詞（antecedent）是協助解讀後續提及的較早表達；共指（coreference）把指向相同實體的提及分組。措辭相似不保證身分相同，兩個不同幫浦都可能叫「備用幫浦」。

先行詞排序模型

Lee 等人的區段式共指模型，可替一個提及的可能先行詞評分 [19]。抽象形式為：

$$p(a_i = j \mid d) = \frac{\exp s(i, j, d)}{\sum_{k \in A_i} \exp s(i, k, d)}.$$

其中 d 是文件或對話；i 索引當前提及；A_i 是包含空選項的候選先行詞集合；a_i 是選到的先行詞變數；$s(i,j,d)$ 是學得相容性分數。空選項讓提及可以引入新實體或暫不連結。刪除候選會改變模型能考慮的範圍。Lee 等人的模型把各項貢獻分開 [19]：

$$s(i,j) = s_m(i) + s_m(j) + s_a(i,j) \quad (j \neq \epsilon), \qquad s(i,\epsilon) = 0.$$

其中 i 是目前候選區段，j 是較早候選區段，ϵ 是空先行詞，s_m 評分區段是否為提及，s_a 評分配對相容性。固定空選項分數為零，替真實連結提供共同基準。提及剪枝節省配對計算，卻可能在連結開始前，就移除唯一有效的先行詞。

模型既要偵測提及，也要連結。較早實體區段若已被丟掉，後續只改善剩餘候選分數也找不回它。診斷時應分開偵測召回與連結準確率。

群組讓監督更複雜。如果 `Pump A`、`the pump`、`it` 指同一實體，最後提及連到任一較早提及都正確。任意只獎勵最近一個的損失，會懲罰另一條有效連結。以群組為目標的訓練改為獎勵它們的合計機率：

$$\mathcal{L}_i = -\log \sum_{j \in G_i} p(a_i = j \mid d).$$

其中 $\mathcal{L}_i$ 是候選區段 i 的損失，G_i 是依標註與剪枝規則保留下來、且非空的有效先行詞集合，a_i 是先行詞變數，d 是文件，$p(a_i = j \mid d)$ 是條件先行詞機率。沒有有效較早先行詞時，空選項屬於 G_i。這是對有效連結取邊際，原始模型也使用這種方法 [19]。推論時，沿先行詞連結合併提及，形成實體群組。一條錯誤橋接就可能合併兩個原本各自合理的群組，錯誤合併的群組之後可能讓多個事件歸到錯誤幫浦。

新近性基準與它的失敗

以下手寫基準使用明確相容性與新近性，刻意保持小型以揭露失敗。裝置類別由資料提供，程式不會依姓名或社會刻板印象推測屬性。

```
entities = [
    {"id": "pump_a", "category": "device", "turn": 1},
    {"id": "valve_b", "category": "device", "turn": 2},
    {"id": "operator", "category": "person", "turn": 2},
]
def candidates(category):
    return [e for e in entities if e["category"] == category]
def recent(category):
    choices = candidates(category)
    return max(choices, key=lambda e: e["turn"])["id"] if choices else None
print("Recency baseline for it:", recent("device"))
assert recent("device") == "valve_b"
# In a synthetic annotation, the speaker meant the earlier pump.
intended = "pump_a"
assert recent("device") != intended
# A cautious application preserves ambiguity instead of hiding the choice.
possible = {e["id"] for e in candidates("device")}
resolution = next(iter(possible)) if len(possible) == 1 else None
assert resolution is None and possible == {"pump_a", "valve_b"}
```

基準輸出 `valve_b`，人工設定的說話意圖卻指較早幫浦。線索無法每次都確定身分。「你指 Pump A 還是 Valve B？」這種澄清能明確保留不確定性；NLP 08 會在對話中發展澄清。

上下文還包含指示語（deixis）：這個、這裡、現在等表達依賴說話者、地點、時間或介面選取，因此應把相關情境與話語一起存；系統拒絕輸入無法支持的推論，不應被判錯。引文也帶來限制：操作員引文中的 `I` 指操作員，「另一台幫浦」則預設一個新近性規則無法定義的對比集合。

僅線上版：*Coreference — End-to-end Neural Coreference Resolution — https://blog.shyandram.dev/zh-tw/learn/nlp/syntactic-semantic-tasks/papers/coreference–end-to-end-neural-coreference-resolution/*

評估共指：MUC、B^3、CEAF 與 CoNLL F1

直觀說法。共指輸出是把提及分割成實體。不同指標以不同單位——連結、提及或實體——計算兩個分割的一致程度，因此單一指標無法說明全貌。

令 K 為鍵（標準）分割，R 為回應分割。

- **MUC** 以連結為基礎 [20]。對每個鍵實體 S，令 $p(S)$ 為回應在 S 上誘導的分割，未匹配的提及視為單例。召回率為 $\sum_S (|S| - |p(S)|) / \sum_S (|S| - 1)$：回應保留的鍵連結數，除以所需的最少連結數。精確率則交換鍵與回應的角色。
- **B^3** 以提及為基礎。對每個提及 m，比較包含它的鍵實體 K_m 與回應實體 R_m：召回率對提及平均 $|K_m \cap R_m|/|K_m|$，精確率平均 $|K_m \cap R_m|/|R_m|$ [23]。
- **CEAF** 以實體為基礎 [21]。它在鍵實體與回應實體間找最佳一對一對齊，也就是最大二分匹配。以實體相似度 $\phi(K, R) = 2|K \cap R|/(|K| + |R|)$（實體式變體 CEAF_e）計算時，召回率是對齊相似度總和除以鍵實體數，精確率則除以回應實體數。

CoNLL-2012 共享任務以 MUC、B^3 與 CEAF_e 三者 F1 的**未加權平均**評分，理由是每個指標代表不同面向 [23]。Pradhan 等人後來發布參考評分程式，因為不一致的實作——特別是預測提及與標準提及不符時——產生了無法比較的數字 [22]。以下程式針對圖 5 的回應計算三者，其中「it」被錯誤併入閥門群組。

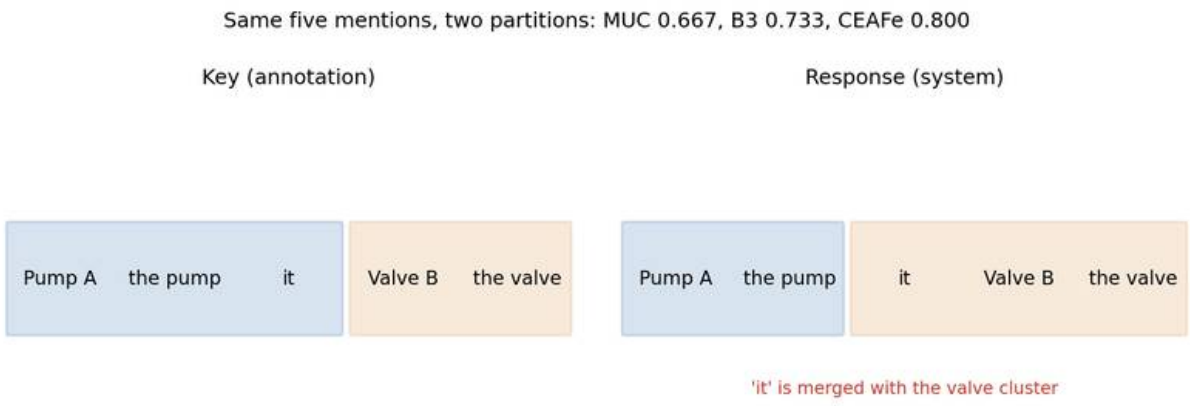


圖 5 — 一條錯誤連結，三種不同扣分。分數由下方程式計算。

```
import numpy as np
from scipy.optimize import linear_sum_assignment

# Mentions: a="Pump A", b="the pump", c="it", d="Valve B", e="the valve"
key = [{"a", "b", "c"}, {"d", "e"}]
response = [{"a", "b"}, {"c", "d", "e"}]          # "it" wrongly joined to the valve

def f1(p, r):
    return 2 * p * r / (p + r) if p + r else 0.0

def muc(gold, pred):                                  # link-based
    def recall(g, p):
        def parts(s):  # pieces of s cut by p; uncovered mentions are singletons
            covered = set().union(*p)
            return sum(1 for t in p if s & t) + len(s - covered)
        return sum(len(s) - parts(s) for s in g) / sum(len(s) - 1 for s in g)
    return recall(pred, gold), recall(gold, pred)    # precision, recall

def b_cubed(gold, pred):                              # mention-based
    def score(a, b):
        cluster_of = {m: s for s in b for m in s}
        return np.mean([len(s & cluster_of.get(m, set())) / len(s)
                        for s in a for m in s])
    return score(pred, gold), score(gold, pred)

def ceaf_e(gold, pred):                               # entity-based
    phi = np.array([[2 * len(g & p) / (len(g) + len(p)) for p in pred] for g in gold])
    rows, cols = linear_sum_assignment(-phi)
    total = phi[rows, cols].sum()
    return total / len(pred), total / len(gold)

scores = {name: f1(*fn(key, response)) for name, fn in
          [("MUC", muc), ("B3", b_cubed), ("CEAFe", ceaf_e)]}
for name, value in scores.items():
    print(f"{name:6s} F1 = {value:.3f}")
print(f"CoNLL F1 = {np.mean(list(scores.values())):.3f}")
```

```
MUC    F1 = 0.667
B3     F1 = 0.733
CEAFe  F1 = 0.800
CoNLL F1 = 0.733
```

手算 MUC：鍵需要 $2 + 1 = 3$ 條連結；回應把幫浦實體切成兩塊、閥門實體保持完整，保留 $1 + 1 = 2$ 條連結，因此召回率為 $2/3$，依對稱性精確率也是 $2/3$。B^3 對每個受影響的提及扣分，CEAF_e 則仍讓每個鍵實體對齊到大致正確的回應實體。一個錯誤、三個數字：這正是共享任務取平均的原因，也是應用應該同時檢查使用這些群組之紀錄的原因。

分數只在標註政策下可解讀：是否連結事件與聯想關係、單例是否計分，以及如何標註歧義。

語意剖析與 AMR

直觀說法。語意剖析把句子映射到程式能使用的意義表示：邏輯形式、資料庫查詢或一張圖。抽象意義表示（Abstract Meaning Representation，AMR）是廣泛使用的整句圖格式。

Banarescu 等人為「語意庫」設計 AMR：AMR 是有根、帶標籤的有向圖，便於人閱讀、也便於程式走訪；它試圖抽離句法上的特異性，讓基本意義相同的句子得到相同 AMR，並大量使用 PropBank 框架集 [24]。當一個實體扮演多個角色時，AMR 使用**再進入（re-entrancy）**——一個有多個父節點的節點——因此圖不是樹。在「the operator wants to stop the pump」中，operator 同時是 `want-01` 與 `stop-01` 的 `ARG0`。AMR 刻意不規定圖如何從詞推導出來，因此詞與節點的對齊是另一個問題 [24]。

Smatch 把每張圖視為三元組的合取——`instance(variable, concept)` 或 `relation(variable1, variable2)`——並在使匹配最多的變數對應下，報告匹配三元組的精確率、召回率與 F1，以此評估 AMR 剖析 [25]。找出該對應是 NP 完全問題，因此論文提供整數線性規劃方法，以及帶隨機重啟的爬山啟發法 [25]。對四個變數的圖，我們可以直接嘗試每種對應：

```
import itertools

# "the operator wants to stop the pump" vs a parse that loses who stops.
gold = {("instance", "w", "want-01"), ("instance", "o", "operator"),
        ("instance", "s", "stop-01"), ("instance", "p", "pump"),
        ("ARG0", "w", "o"), ("ARG1", "w", "s"),
        ("ARG0", "s", "o"), ("ARG1", "s", "p")}
pred = {("instance", "a", "want-01"), ("instance", "b", "operator"),
        ("instance", "c", "stop-01"), ("instance", "d", "pump"),
        ("ARG0", "a", "b"), ("ARG1", "a", "c"), ("ARG1", "c", "d")}

def rename(triples, mapping):
    return {(r, mapping.get(x, x), mapping.get(y, y)) for r, x, y in triples}

pred_vars, gold_vars = sorted({t[1] for t in pred}), sorted({t[1] for t in gold})
best = max(len(rename(pred, dict(zip(pred_vars, perm))) & gold)
           for perm in itertools.permutations(gold_vars))
precision, recall = best / len(pred), best / len(gold)
print("matched triples:", best)
print(f"Smatch P={precision:.3f} R={recall:.3f}
 ↪  F1={2*precision*recall/(precision+recall):.3f}")
```

```
matched triples: 7
Smatch P=1.000 R=0.875 F1=0.933
```

預測圖少了一個三元組——說明 operator 也是執行停止者的再進入 `ARG0` 邊。Smatch 仍給 0.933。對安全應用而言，「誰停止幫浦」可能就是全部問題。與本章其他彙總指標一樣，Smatch 應搭配針對關鍵關係的檢查。

預訓練模型如何改變這些任務

直觀說法。預訓練編碼器把剖析器、角色標註器與共指系統變成「編碼器加上小型結構解碼器」。它們沒有取消對解碼器、標註規範或謹慎評估的需求。

有三個明顯轉變。第一，**特徵被上下文向量取代**。Kitaev 與 Klein 的剖析器在使用預訓練詞表示後，從 93.55 提升到 95.13 F1 [6]；Joshi 等人把 BERT 用於共指，在 OntoNotes 上報告比先前最佳成績高 3.9 F1，並指出在文件層級脈絡、對話與提及改寫上仍有弱點 [26]。SpanBERT 則改變預訓練目標本身——遮住連續跨度，並訓練跨度邊界表示去預測跨度內容——報告 OntoNotes 共指 79.6% F1，以及 SQuAD 1.1 與 2.0 上的 94.6 與 88.7 F1 [27]。跨度形狀的任務受益於跨度形狀的預訓練。

第二，**結構有一部分其實存在於編碼器內部**。Hewitt 與 Manning 的結構探測找到一個線性轉換，使詞向量間的平方距離編碼剖析樹中的距離；這對 ELMo 與 BERT 成立，對基準則不成立 [28]。Tenney、Das 與 Pavlick 發現 BERT 以可解釋的順序在各層表示傳統管線的步驟——詞性標記、剖析、命名實體、語意角色，最後是共指——同時有時會依較高層資訊修正較低層的決策 [29]。這些是探測結果：它們顯示資訊可被線性取出，不表示模型像剖析器那樣使用它。

第三，**管線收斂成微調與提示**。編碼器為標記與樹提供證據，大型語言模型也可以依提示回傳結構 [30]。輸出驗證器仍有必要：看起來合理的括號字串，可能漏掉詞元、重排詞，或不形成連通樹。

現代觀點

綜述與其分類。 Zhang 的句法—語意剖析綜述聚焦於兩種最常見的形式化——成分剖析與依存剖析——回顧兩者以及語意依存圖的代表模型，並涵蓋跨領域、跨語言與聯合剖析，以及語料建置 [31]。Poesio 等人對回指計算模型的回顧，從神經網路之前的研究一路追溯到現今的神經模型，整理標註資料集，並討論橋接指涉與篇章指示等較少研究的現象 [32]。Chen 等人的 SRL 綜述已獲 Artificial Intelligence Review 接受，提出四維分類——模型架構、句法特徵建模、應用情境與多模態延伸——分析句法特徵何時有幫助，並探討 LLM 與專門 SRL 系統的互補角色 [34]。Zhao 等人的 LLM 綜述區分模型訓練、調適、使用與能力評估 [30]；結構可在其中扮演不同角色：監督輸出、內部表示探測、生成限制或診斷挑戰。Chang 等人的評估綜述再區分量測任務與使用的基準及程序 [33]。

近期原始研究（2023–2026）。 Tian、Xia 與 Song 發現 LLM 擅長淺層剖析（分塊），卻難以產生正確的完整成分樹，並以先提示分塊、再過濾分塊的方式改善 [35]。Hale 與 Stanojević 跨類型多樣的語言測試一項候選句法普遍性，發現所研究的模型只在部分較高資源語言中遵守它 [36]，說明以英文為主的結構測試不能證明一致的多語言能力。在共指方面，Le 與 Ritter 發現提示指令微調模型可以勝過無監督共指系統，但依賴高品質的提及偵測 [37]；Porada、Zou 與 Cheung 在五個共指系統間控制預訓練語言模型，發現在相同模型規模下，基於編碼器的模型勝過較新的基於解碼器的模型，而控制編碼器後，過去五年報告的 F1 提升大部分（但不是全部）消失 [38]。在語意表示方面，Jin 等人發現把 AMR 放進 LLM 思維鏈提示，在五個任務上大致弊多於利，錯誤集中在多詞表達、命名實體與最後推論步驟 [39]。

區分結構有效與結構泛化

這些研究引出兩種不同測試：第一，系統能否替這個標註規則產生合法結構？第二，選擇是否回應語法差異，而不是熟悉詞序？結構驗證器處理第一個問題，無法證明第二個。總是回傳某種慣常附接的模型，可以通過格式檢查，卻在受控比較中失敗。

建立詞彙幾乎不變、角色相關構造卻改變的配對（主動與被動、修飾語移位、關係子句），再固定構造並更換詞彙。如果不熟悉的名詞改變整個結構，詞彙涵蓋正在干擾；如果語法改變後解讀不變，熟悉詞彙模式主導了結果。最後檢查錯誤是否改變下游所需的語意角色。

較長上下文改變指涉決策，不只增加記憶大小

較長歷史可恢復缺少的先行詞，但也會加入相容候選；在先行詞 softmax 中，即使既有連結分數不變，新增候選仍會改變分母，因此以短歷史校準的門檻未必能沿用。比較短歷史、長但不相關的歷史，以及含另一台同類裝置的長歷史，才能區分有用脈絡與單純更多文字。

未解問題。標註框架各有差異；低資源語言缺少獨立檢查的樹；切分與語法交互影響；自發對話含有片段與修正；長文件與對話中的共指仍然困難；模型也可能學會基準慣例，卻缺乏穩健的組合能力。保留替代分析、測試結構最小配對、比較架構時控制編碼器 [38]，以及量測下游後果，是研究這些限制的具體做法。

重點整理

- 文法無法剖析是涵蓋範圍的結果，不是對說話者的評判；圖表共享跨度，但列舉所有樹仍可能指數成長。
- 最佳樹（最大值）與總機率（加總）回答不同問題；只有規則機率看不到詞彙證據，這正是神經跨度與弧評分器補上的部分。
- 轉移式剖析器做線性時間的局部決策；圖式剖析器替所有弧評分並需要樹解碼器，因為獨立選擇中心詞可能形成循環。
- 每種結構都有自己的指標——帶標籤括號、UAS ／ LAS、論元 F1、MUC ／ B^3 ／ CEAF、Smatch——且都需要說明分母與對齊政策。
- 語意角色在語法變換下保持不變；PropBank 依述詞編號論元，並把否定與情態標為附加語。
- 在擷取紀錄中保留作用範圍、歸屬與來源；過粗的結構描述會丟失任何編碼器事後都找不回的區別。
- 共指是偵測加上連結；一條錯誤橋接就可能合併實體，明確的歧義結果勝過自信的錯誤。
- 預訓練編碼器改進並簡化了這些系統，但有效性仍來自解碼器，評估仍來自標註規範。

練習

1. **延伸文法**。讓圖表剖析器接受「we saw cameras」與一個新介系詞，再加入未見名詞，說明需要哪個詞彙改變。

提示

既有二元規則已支援較短句子。終結符的涵蓋與片語組合是不同問題。

2. **辨識與列舉**。把圖表的樹清單改成布林類別，比較剖析器仍能回答什麼，以及丟掉什麼。接著修改 PCFG 程式，加入針對 `robots with` 的詞彙化規則分數，讓名詞附接在原始規則機率下勝出。

   提示

   不保留樹仍能判斷是否接受；要恢復不同附接，需要回溯指標。詞彙化版本可把 $NP \to NP\ PP$ 的分數乘上一個依中心名詞而定的因子。

3. **結構驗證器與非投射樹**。為基本依存樹檢查詞元輸出、每詞一個中心詞、沒有自我連結、唯一根與無循環。再為一個短句建構弧線會交叉的有效樹，並說明 arc-standard 神諭為何無法產生它。

   提示

   從每詞追蹤中心詞指標，應抵達根，且不重訪同一詞。arc-standard 只連結堆疊頂端兩項，因此它建立的每條弧都跨越一段連續、已完成的區域。

4. **兩種閱讀，兩筆紀錄**。寫出「A technician inspected every sensor」的兩種解讀。設計能儲存任一種閱讀的關係結構描述，以及從文件集合回答「每個感測器都檢查過了嗎？」的證據規則。

   提示

   比較存在量詞在全稱量詞外側與相反順序。答案需要感測器清單與涵蓋資訊，而不只是擷取到的檢查事件。

5. **共指指標**。建立一個把圖 5 五個提及全部併成一個實體的回應，以及一個讓五個提及都是單例的回應。執行程式前先算出兩者的 MUC、B^3 與 $CEAF_e$，再說明哪個指標對哪種錯誤最寬容。

   提示

   MUC 計算連結，所以單一巨大群組會保留每條鍵連結。B^3 對每個群組含錯誤成員的提及扣分。單例完全沒有連結。

6. **雙語指示語**。寫一則含有指示詞、相對日期與間接請求的雙語訊息，例如混用英文與中文。描述兩種會改變解讀的語境，並列出共指或擷取系統需要哪些證據才能區分。

   提示

   轉寄引述會改變 “I”、“here” 與 “tomorrow” 所依據的視角。能力測驗與維修對話，可以讓相同問句有不同目的。

參考文獻

1. Susan McRoy, *Principles of Natural Language Processing*, Chapters 3–5 and 7 and the chart-parsing appendix, open textbook, 2021. Open textbook
2. Daniel Jurafsky and James H. Martin, *Speech and Language Processing*, 3rd ed. draft (August 19, 2026), Chapters 19, 20, 22 and 24. Book site
3. E. Black, S. Abney, D. Flickenger, C. Gdaniec, R. Grishman, P. Harrison, D. Hindle, R. Ingria, F. Jelinek, J. Klavans, M. Liberman, M. Marcus, S. Roukos, B. Santorini and T. Strzalkowski, "A Procedure for Quantitatively Comparing the Syntactic Coverage of English Grammars," Speech and Natural Language Workshop, 1991. Paper
4. Mitchell P. Marcus, Beatrice Santorini and Mary Ann Marcinkiewicz, "Building a Large Annotated Corpus of English: The Penn Treebank," Computational Linguistics 19(2), 1993. Paper
5. Mitchell Stern, Jacob Andreas and Dan Klein, "A Minimal Span-Based Neural Constituency Parser," ACL, 2017. Paper
6. Nikita Kitaev and Dan Klein, "Constituency Parsing with a Self-Attentive Encoder," ACL, 2018. Paper
7. Nikita Kitaev, Steven Cao and Dan Klein, "Multilingual Constituency Parsing with Self-Attention and Pre-Training," ACL, 2019. Paper
8. Marie-Catherine de Marneffe, Christopher D. Manning, Joakim Nivre and Daniel Zeman, "Universal Dependencies," Computational Linguistics 47(2), 2021. Paper
9. Joakim Nivre, "Algorithms for Deterministic Incremental Dependency Parsing," Computational Linguistics 34(4), 2008. Paper
10. Danqi Chen and Christopher Manning, "A Fast and Accurate Dependency Parser using Neural Networks," EMNLP, 2014. Paper
11. Ryan McDonald, Fernando Pereira, Kiril Ribarov and Jan Hajič, "Non-Projective Dependency Parsing using Spanning Tree Algorithms," HLT-EMNLP, 2005. Paper
12. Eliyahu Kiperwasser and Yoav Goldberg, "Simple and Accurate Dependency Parsing Using Bidirectional LSTM Feature Representations," Transactions of the ACL 4, 2016. Paper
13. Timothy Dozat and Christopher D. Manning, "Deep Biaffine Attention for Neural Dependency Parsing," ICLR, 2017. Paper
14. Daniel Zeman, Jan Hajič, Martin Popel, Martin Potthast, Milan Straka, Filip Ginter, Joakim Nivre and Slav Petrov, "CoNLL 2018 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies," CoNLL Shared Task, 2018. Paper
15. Daniel Gildea and Daniel Jurafsky, "Automatic Labeling of Semantic Roles," Computational Linguistics 28(3), 2002. Paper
16. Martha Palmer, Daniel Gildea and Paul Kingsbury, "The Proposition Bank: An Annotated Corpus of Semantic Roles," Computational Linguistics 31(1), 2005. Paper
17. Xavier Carreras and Lluís Màrquez, "Introduction to the CoNLL-2005 Shared Task: Semantic Role Labeling," CoNLL, 2005. Paper
18. Luheng He, Kenton Lee, Mike Lewis and Luke Zettlemoyer, "Deep Semantic Role Labeling: What Works and What's Next," ACL, 2017. Paper
19. Kenton Lee, Luheng He, Mike Lewis and Luke Zettlemoyer, "End-to-end Neural Coreference Resolution," EMNLP, 2017. Paper
20. Marc Vilain, John Burger, John Aberdeen, Dennis Connolly and Lynette Hirschman, "A Model-

Theoretic Coreference Scoring Scheme," MUC-6, 1995. Paper
21. Xiaoqiang Luo, "On Coreference Resolution Performance Metrics," HLT-EMNLP, 2005. Paper
22. Sameer Pradhan, Xiaoqiang Luo, Marta Recasens, Eduard Hovy, Vincent Ng and Michael Strube, "Scoring Coreference Partitions of Predicted Mentions: A Reference Implementation," ACL, 2014. Paper
23. Sameer Pradhan, Alessandro Moschitti, Nianwen Xue, Olga Uryupina and Yuchen Zhang, "CoNLL-2012 Shared Task: Modeling Multilingual Unrestricted Coreference in OntoNotes," EMNLP-CoNLL Shared Task, 2012. Paper
24. Laura Banarescu, Claire Bonial, Shu Cai, Madalina Georgescu, Kira Griffitt, Ulf Hermjakob, Kevin Knight, Philipp Koehn, Martha Palmer and Nathan Schneider, "Abstract Meaning Representation for Sembanking," Linguistic Annotation Workshop, 2013. Paper
25. Shu Cai and Kevin Knight, "Smatch: an Evaluation Metric for Semantic Feature Structures," ACL, 2013. Paper
26. Mandar Joshi, Omer Levy, Luke Zettlemoyer and Daniel Weld, "BERT for Coreference Resolution: Baselines and Analysis," EMNLP-IJCNLP, 2019. Paper
27. Mandar Joshi, Danqi Chen, Yinhan Liu, Daniel S. Weld, Luke Zettlemoyer and Omer Levy, "SpanBERT: Improving Pre-training by Representing and Predicting Spans," Transactions of the ACL 8, 2020. Paper
28. John Hewitt and Christopher D. Manning, "A Structural Probe for Finding Syntax in Word Representations," NAACL-HLT, 2019. Paper
29. Ian Tenney, Dipanjan Das and Ellie Pavlick, "BERT Rediscovers the Classical NLP Pipeline," ACL, 2019. Paper
30. Wayne Xin Zhao et al., "A Survey of Large Language Models," Frontiers of Computer Science, 2026; online manuscript updated March 2026. Survey
31. Meishan Zhang, "A survey of syntactic-semantic parsing based on constituent and dependency structures," Science China Technological Sciences 63, 2020. Survey
32. Massimo Poesio, Juntao Yu, Silviu Paun, Abdulrahman Aloraini, Pengcheng Lu, Janosch Haber and Derya Cokal, "Computational Models of Anaphora," Annual Review of Linguistics 9, 2023. Survey
33. Yupeng Chang et al., "A Survey on Evaluation of Large Language Models," ACM Transactions on Intelligent Systems and Technology, 2024. Survey
34. Huiyao Chen, Meishan Zhang, Jing Li, Lilja Øvrelid, Jan Hajič, Hao Fei and Min Zhang, "A Systematic Survey of Semantic Role Labeling in the Era of Pretrained Language Models," arXiv: 2502.08660, 2025; accepted at Artificial Intelligence Review. Survey
35. Yuanhe Tian, Fei Xia and Yan Song, "Large Language Models are No Longer Shallow Parsers," ACL, 2024. Paper
36. John T. Hale and Miloš Stanojević, "Do LLMs learn a true syntactic universal?" EMNLP, 2024. Paper
37. Nghia T. Le and Alan Ritter, "Are Large Language Models Robust Coreference Resolvers?" arXiv:2305.14489, 2023. Paper
38. Ian Porada, Xiyuan Zou and Jackie Chi Kit Cheung, "A Controlled Reevaluation of Coreference Resolution Models," LREC-COLING, 2024. Paper
39. Zhijing Jin, Yuen Chen, Fernando Gonzalez Adauto, Jiarui Liu, Jiayi Zhang, Julian Michael, Bernhard Schölkopf and Mona Diab, "Analyzing the Role of Semantic Representations in the

Era of Large Language Models,” NAACL, 2024. Paper
40. NLTK contributors, `nltk.grammar.PCFG`，API 文件，查閱日期 2026-11-05。正規化契約

篇 II

圖形識別

PR 01・模式、機率與貝氏決策

建立辨識流程，區分似然與後驗機率，推導最小錯誤率、最小風險與拒絕決策，以及高斯判別函數與 ROC 分析。

網站版本: https://blog.shyandram.dev/zh-tw/learn/pattern-recognition/bayes-decisions/

你將學到

- 把辨識問題拆成觀測、特徵、預測與行動，並先想清楚「一筆樣本」代表什麼。
- 區分類別先驗、類別似然與類別後驗，並看懂先驗改變時決策邊界如何移動。
- 推導最小錯誤率規則、任意損失矩陣下的最小風險規則，以及 Chow 的拒絕規則。
- 把分類器寫成判別函數，推導高斯情形下的線性（LDA）與二次（QDA）決策邊界。
- 讀懂 ROC 曲線與精確率–召回率曲線，連結 Neyman–Pearson 準則，並依成本選擇操作點。

先看全貌

相機拍下一個零件。系統量測它的形狀，估計它是否有缺陷，再決定要不要送去人工檢查。這些是不同的階段：拍到的像素是**觀測**（observation）；形狀量測值是**特徵**（feature）；缺陷機率是**預測**（prediction）；送去檢查則是**行動**（action）。貝氏決策理論處理的正是最後兩者之間的連結：在目前的信念之下，應該採取什麼行動？

> **用最簡單的話說**
>
> 分水果的機器看到一顆水果，先猜它熟了沒有。「猜」是一件事，「選籃子」是另一件事。如果箱子裡大部分水果都是熟的，即使這顆看起來有點怪，機器也該偏向「熟了」。如果丟掉一顆好水果的代價，遠大於請人看一眼，機器就應該偶爾說：「我不確定，請人看一下這顆。」貝氏決策理論，就是把這三件事算清楚：每一種有多常見、證據怎麼說、每一種錯誤有多痛。

本系列沿用經典的圖形識別教科書：Duda、Hart 與 Stork 的《Pattern Classification》[1]、Bishop 的《Pattern Recognition and Machine Learning》[2]，以及 Theodoridis 與 Koutroumbas 的《Pattern Recognition》[3]，統計觀點則參考 Hastie、Tibshirani 與 Friedman [4]。各章的範例與推導都是原創的，較長的證明請回到教科書。這些教科書都從貝氏決策理論開始，因為後面所有方法都可以拿它來衡量：一個分類器好不好，取決於它多接近「如果知道真實分布，貝氏規則會做的決定」。

辨識不限於影像。觀測也可以是一段錄音、一串筆畫或一份文件。分類預測類別，迴歸預測數值，分群則在沒有標籤時找出群組。本章談分類，但這套思考方式（先為不確定性建模，再選期望成本最低的行動）適用於三者。

先定義實驗

白話版。在寫任何公式之前，先寫下一筆樣本是什麼，以及資料是怎麼蒐集的。辨識系統許多出人意料的失敗都來自這一步，而不是分類器本身。

一張影像、一位病人、一段影片、影片中的一個裁切區域，是不同的**抽樣單位**（sampling unit）。如果二十個影格來自同一段錄影，把它們當成二十個獨立個案，會讓測試看起來比實際部署容易得多，因為測試影格幾乎就是訓練影格的複本。

精確版。辨識流程可以寫成

$$o \longrightarrow x = \phi(o) \longrightarrow p(c \mid x) \longrightarrow a(x).$$

其中 o 是原始觀測（影像、波形）；ϕ 是前處理與特徵抽取，把觀測映射成特徵向量 $x \in \mathbb{R}^d$；$c \in \{c_1, \dots, c_K\}$ 是 K 個類別之一；$p(c \mid x)$ 是模型對類別的信念；$a(x)$ 是最後採取的行動。學習階段用**訓練**樣本調整 ϕ 與機率模型的參數；**驗證**樣本用來選擇門檻、正則化等設定；保留的**測試**集只用一次，評估整個已選定的程序（PR 09 會仔細討論：驗證與集成方法）。

手工特徵（邊緣直方圖、形狀矩）與學習得到的嵌入（預訓練網路的輸出）佔據同一個位置 ϕ。換掉特徵抽取器，就改變了分類器能拿到的資訊。被裁切、量化或池化移除的資訊，下游任何分類器都救不回來。用決策理論的語言說：問題的貝氏錯誤率是相對於特徵而定義的；差的 ϕ 會抬高任何分類器都無法低於的下限。

貝氏定理：證據如何改變信念

白話版。貝氏定理告訴我們，如何利用「這個量測值在各類別中有多典型」，把「各類別有多常見」更新成「對這個物體而言各類別的可能性」。

精確版。令類別為 $c_1, \dots, c_K$。貝氏定理給出

$$P(c_k \mid x) = \frac{p(x \mid c_k)\, P(c_k)}{\sum_{j=1}^{K} p(x \mid c_j)\, P(c_j)}.$$

- **先驗**（prior）$P(c_k)$ 是看這筆觀測之前類別 c_k 的機率，例如產線上缺陷零件的比例。
- **似然**（likelihood，或類別條件密度）$p(x \mid c_k)$ 描述類別 c_k 內量測值的分布。
- **證據**（evidence）$p(x) = \sum_j p(x \mid c_j) P(c_j)$ 是 x 的整體密度；它對每個類別都相同，只負責正規化。
- **後驗**（posterior）$P(c_k \mid x)$ 是觀測到 x 之後類別 c_k 的機率。

連續量測的似然是密度，數值不一定介於零與一之間，也不是「恰好觀測到這個實數」的機率。類別後驗機率則落在 $[0, 1]$，並且對 k 加總為一。把兩個條件方向 $p(x \mid c)$ 與 $P(c \mid x)$ 搞混，就是經典的**基本率謬誤**（base-rate error），下面的程式範例會具體呈現。

因為證據是共用的，做決策只需要乘積 $p(x \mid c_k) P(c_k)$。圖 1 以兩個一維高斯類別說明其幾何意義：邊界落在兩條乘上先驗後的似然曲線交會之處。當類別 c_1 的先驗從 0.5 降到 0.1，它的加權曲線縮小，邊界往 c_1 的平均值移動：系統現在需要更強的證據，才會判定為較稀少的類別。

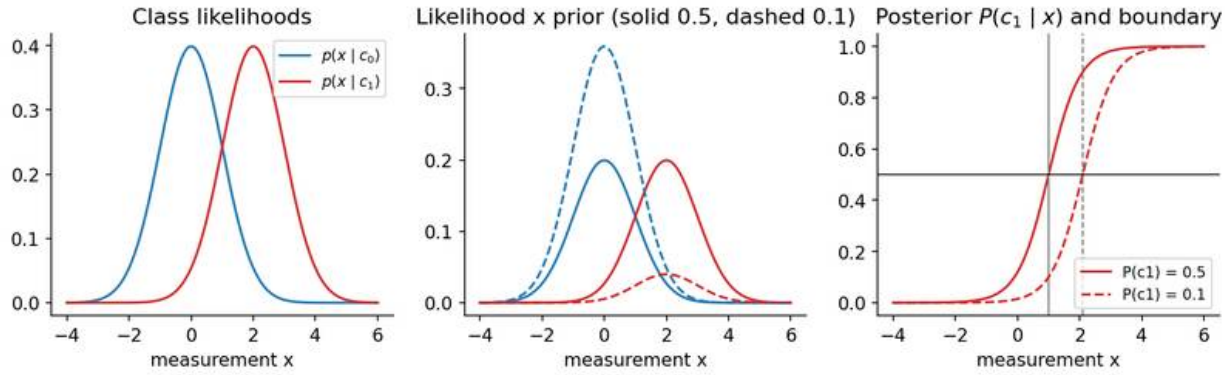


圖 *1* — 先驗改變時，似然（左）不變；乘上先驗的似然（中）與後驗（右）則會改變。$P(c1) = 0.5$ 時邊界在 $x = 1$；$P(c1) = 0.1$ 時移到 $1 + ln(9)/2 \approx 2.10$。

圖 1 的單位變異數情形有封閉解。平均值 $\mu_0 = 0$、$\mu_1 = 2$、變異數為 1 時，加權似然的對數比為 $2x - 2 + \ln \frac{P(c_1)}{P(c_0)}$，在 $x = 1 + \frac{1}{2} \ln \frac{P(c_0)}{P(c_1)}$ 處為零。先驗相等時是 $x = 1$，$P(c_1) = 0.1$ 時則是 $x \approx 2.10$。

程式碼：稀有事件改變後驗

這個 NumPy 範例使用假設的檢測統計：偵測器能標出 90% 的缺陷零件（靈敏度），也會誤標 8% 的良品（偽陽性率），而缺陷零件佔 2%。程式計算偵測器輸出陽性與陰性後的後驗，再比較最小錯誤率與最小風險的行動（後面幾節會推導）。

```
import numpy as np

prior = 0.02
sensitivity = 0.90
false_positive_rate = 0.08
p_positive = (sensitivity * prior /
              (sensitivity * prior + false_positive_rate * (1 - prior)))
p_negative = ((1 - sensitivity) * prior /
              ((1 - sensitivity) * prior + (1 - false_positive_rate) * (1 - prior)))
p = np.array([p_negative, p_positive])
c_fp, c_fn = 1.0, 9.0
threshold = c_fp / (c_fp + c_fn)
risks = np.column_stack([c_fn * p, c_fp * (1 - p)])
decisions = risks.argmin(axis=1)
assert np.array_equal(decisions, (p > threshold).astype(int))
print("posteriors [negative observation, positive observation]:", p)
print("minimum-risk actions:", decisions, "threshold:", threshold)
print("minimum-error actions:", (p > 0.5).astype(int))
```

輸出顯示：陰性後的後驗約為 0.0022，陽性後也只有 0.187。陽性觀測並不代表 90% 的缺陷機率：靈敏度以真實類別為條件，後驗則以觀測為條件；在 2% 的先驗下，大部分陽性標記其實來自數量龐大的良品。成本相同時，兩種觀測都不會判為「缺陷」（$0.187 < 0.5$）。若漏掉缺陷的代價是多做一次檢查的九倍，門檻降到 0.1，陽性標記就會送去檢查。機率模型沒變，變的只有決策規則。

最小錯誤率決策

白話版。如果每種錯誤的代價都一樣，就永遠選擇對這筆觀測而言機率最高的類別。平均而言，沒有其他規則能犯更少的錯。

精確版。決策規則 $a(x)$ 把每個 x 指派到一個類別，因而把特徵空間切成**決策區域** $\mathcal{R}_1, \dots, \mathcal{R}_K$，其中 $\mathcal{R}_k = \{x : a(x) = c_k\}$。錯誤機率為

$$P(\text{error}) = \int p(x)\,\bigl[1 - P(a(x) \mid x)\bigr]\,dx.$$

其中 $P(a(x) \mid x)$ 是所選類別的後驗。被積函數非負，而 $p(x)$ 與規則無關，所以只要在每個 x 上分別讓 $1 - P(a(x) \mid x)$ 最小，整個積分就最小，也就是選

$$a^*(x) = \arg\max_k P(c_k \mid x) = \arg\max_k p(x \mid c_k) P(c_k).$$

這就是**最大後驗（MAP）規則**，也稱**貝氏分類器**。它的錯誤率

$$P^*_{\text{err}} = \int p(x)\Bigl[1 - \max_k P(c_k \mid x)\Bigr]dx$$

稱為**貝氏錯誤率**（Bayes error）：在這組特徵下，任何分類器能達到的最低錯誤率。只有當各類別在特徵空間中完全不重疊時，它才是零。在貝氏錯誤率為 7% 的問題上，測試錯誤率 8% 已接近最佳；同樣的 8%，若貝氏錯誤率是 1%，就還有很大的改進空間。真實任務的貝氏錯誤率通常未知，但這個概念能把兩個問題分開：「特徵的資訊夠不夠？」（貝氏錯誤率），以及「分類器有沒有把特徵中的資訊榨乾？」（與下限之間的差距）。

兩類別時，MAP 規則成為**似然比檢定**（likelihood-ratio test）：

$$\Lambda(x) = \frac{p(x \mid c_1)}{p(x \mid c_0)} \;\gtrless\; \frac{P(c_0)}{P(c_1)}.$$

比值超過右邊的門檻就選 c_1，否則選 c_0。資料只透過 $\Lambda(x)$ 進入決策；先驗以及接下來會看到的成本，只會移動門檻。這就是同一個分數可以服務許多不同決策政策的原因。

損失矩陣與最小風險

白話版。有些錯誤比其他錯誤嚴重。為每一種錯誤訂一個成本，在目前信念下計算每個可能行動的平均成本，然後採取最便宜的行動。

精確版。令可用行動為 $\alpha_1, \dots, \alpha_A$（通常是「判為類別 i」，也可能包含「拒絕」）。**損失矩陣** $\lambda_{ij} = L(\alpha_i, c_j)$ 給出真實類別為 c_j 時採取行動 α_i 的成本。一個行動的**條件風險**（conditional risk）是它在後驗下的期望損失：

$$R(\alpha_i \mid x) = \sum_{j=1}^{K} \lambda_{ij}\, P(c_j \mid x), \qquad a^*(x) = \arg\min_i R(\alpha_i \mid x).$$

整體風險 $R = \int R(a(x) \mid x)\, p(x)\, dx$ 可用與前面相同的逐點論證最小化，所以這條**貝氏決策規則**達到可能的最小期望損失，即**貝氏風險** [1, 2]。在**零一損失**（$i = j$ 時 $\lambda_{ij} = 0$，否則為 1）下，$R(\alpha_i \mid x) = 1 - P(c_i \mid x)$，規則化約為 MAP。

兩類別時，記 λ_{10} 為真實是 c_0 卻判為 c_1 的成本（偽陽性），λ_{01} 為真實是 c_1 卻判為 c_0 的成本（偽陰性）。當 $R(\alpha_1 \mid x) < R(\alpha_0 \mid x)$ 時選 c_1 較便宜，整理後得到

$$\frac{p(x \mid c_1)}{p(x \mid c_0)} > \frac{\lambda_{10} - \lambda_{00}}{\lambda_{01} - \lambda_{11}} \cdot \frac{P(c_0)}{P(c_1)}.$$

這個比值形式假設兩類先驗皆為正，且 $\lambda_{01} - \lambda_{11} > 0$，如一般分類成本。若此差為零，直接比較條件風險；若為負，除法會反轉不等號。在正確決策不花成本時，把偽陰性成本加倍，等同把**先驗勝算** $P(c_1)/P(c_0)$ 加倍，而不是只把陽性類別的機率加倍；兩類先驗仍須相加為一。若正確決策不花成本（$\lambda_{00} = \lambda_{11} = 0$），記 $C_{\mathrm{FP}} = \lambda_{10}$、$C_{\mathrm{FN}} = \lambda_{01}$、$p = P(c_1 \mid x)$，規則就成為後驗上的門檻：

$$C_{\mathrm{FP}}(1 - p) < C_{\mathrm{FN}}\, p \quad \Longleftrightarrow \quad p > \frac{C_{\mathrm{FP}}}{C_{\mathrm{FP}} + C_{\mathrm{FN}}}.$$

漏掉缺陷的代價是多做一次檢查的九倍時，門檻是 0.1，而不是 0.5（圖 2 右）。

實務上有兩點很重要。第一，損失矩陣是一個建模決定，常常由機器學習團隊以外的人（臨床醫師、安全工程師、產品負責人）決定；請明確寫下來，讓人可以審查。第二，後驗門檻只有在後驗**經過校準**（calibrated）時才有意義，也就是說，被給予機率 0.1 的個案中，真的大約有 10% 是陽性。許多現代分類器並非如此，這是下方「現代觀點」的主題之一。

拒絕選項

白話版。系統不確定時，可以把案例交給人處理。拒絕判斷有固定成本；做出判斷的期望成本，則在後驗分散時很高。當判斷的期望成本高於拒絕成本時，就拒絕。

精確版。加入一個行動 α_{reject}，不論真實類別為何，損失都是常數 C_R；K 個分類行動仍用零一損失。最佳分類行動的風險為 $1 - \max_k P(c_k \mid x)$，因此貝氏規則成為 **Chow 規則** [5]：

$$a^*(x) = \begin{cases} \arg\max_k P(c_k \mid x) & \text{if } \max_k P(c_k \mid x) \geq 1 - C_R, \\ \text{reject} & \text{otherwise.} \end{cases}$$

若拒絕成本為 0.2，系統只在最大後驗至少為 0.8 時才接受（圖 2 左）。兩個極端有助於建立直覺：若 $C_R = 0$（請人看完全免費），系統會拒絕所有不確定的案例；若 $C_R \geq 1 - 1/K$，系統永遠不拒絕，因為最大後驗至少是 $1/K$。

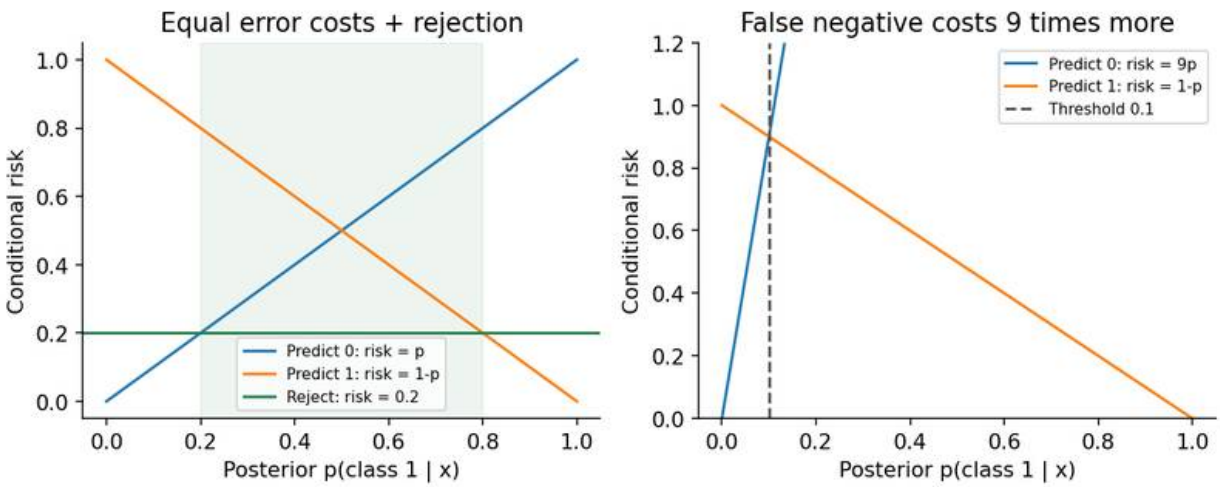


圖 *2* — 條件風險隨後驗變化。左：成本相同，再加上成本 *0.2* 的拒絕行動；後驗介於 *0.2* 與 *0.8* 時拒絕最便宜。右：偽陰性成本是偽陽性的九倍，決策門檻移到 *0.1*。

Chow 指出，掃過門檻會畫出一條**錯誤–拒絕權衡曲線**：每多拒絕一個百分點的樣本，就能降低剩餘樣本的錯誤，但效益遞減 [5]。圖 3 以三個重疊的高斯類別示範，它們的真實後驗是已知的。不拒絕時約有四分之一樣本被分錯；拒絕最不確定的 40% 後，全部樣本中的錯誤降到約 8%。

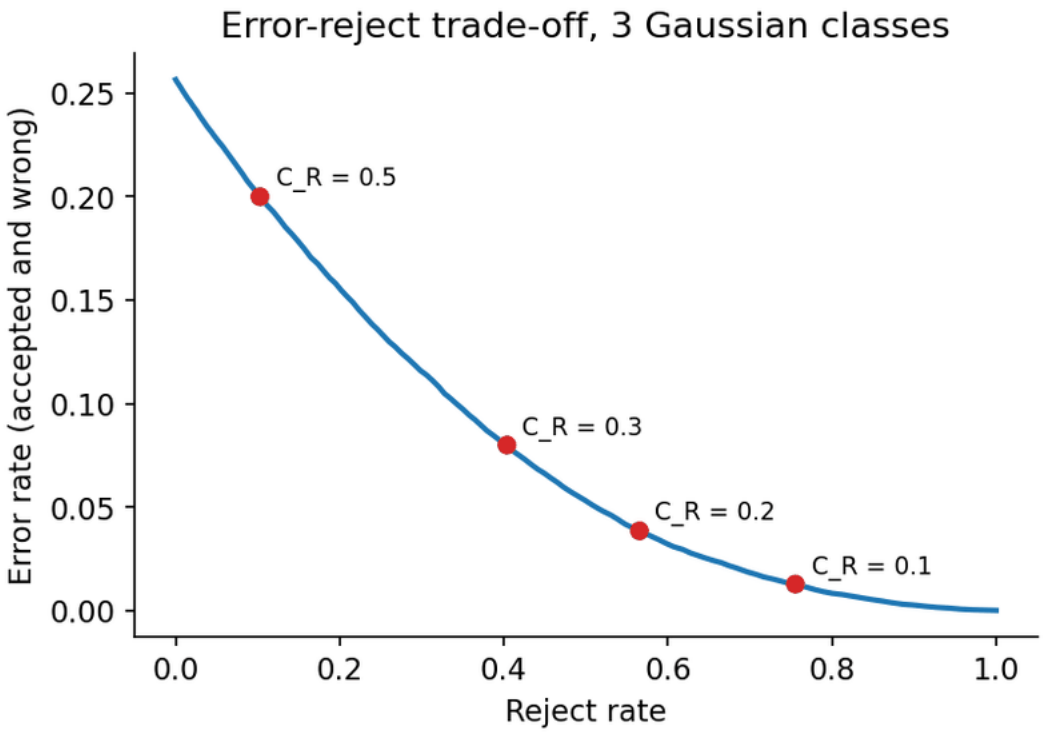


圖 *3* — 三個二維高斯類別（後驗已知，*20,000* 筆合成樣本）的錯誤–拒絕權衡。每個點對應一個拒絕成本 *C_R* 下的 *Chow* 規則；縱軸計算被接受且分錯的樣本比例。

程式碼：Chow 規則

```
import numpy as np

rng = np.random.default_rng(2)
means = np.array([[0, 0], [2, 0], [1, 1.7]])
y = rng.integers(0, 3, 20000)
X = means[y] + rng.normal(0, 1, (len(y), 2))

# exact posteriors: equal priors, identity covariance
logits = -0.5 * ((X[:, None, :] - means[None]) ** 2).sum(-1)
post = np.exp(logits - logits.max(1, keepdims=True))
post /= post.sum(1, keepdims=True)
pred, pmax = post.argmax(1), post.max(1)

for c_r in [0.5, 0.3, 0.2, 0.1]:
    accept = pmax >= 1 - c_r                    # Chow's rule
    reject_rate = 1 - accept.mean()
    err_accepted = np.mean(pred[accept] != y[accept])
    risk = np.mean(np.where(accept, pred != y, c_r))
    print(f"C_R={c_r}: reject {reject_rate:.3f}, "
          f"error on accepted {err_accepted:.3f}, average loss {risk:.3f}")
```

在這個亂數種子下，$C_R = 0.3$ 約拒絕 40% 的樣本，被接受的樣本中錯誤約為 13.5%；$C_R = 0.1$ 約拒絕 76%，剩下約 4.4% 的錯誤。平均損失隨 C_R 下降而下降，因為拒絕本身變便宜了。取指數之前先減去最大 logit（`logits.max`），是讓 softmax 數值穩定的標準技巧。

只有當不確定性估計有意義時，拒絕才有幫助。Chow 規則處理的是**模糊性**（ambiguity）：落在已知類別之間的觀測。它處理不了**新穎性**（novelty）：訓練時從未見過的物體，可能離每個類別都很遠，卻仍得到很有自信的後驗，因為後驗只在已知類別之間正規化。偵測這類輸入是開放集辨識與分布外偵測的課題（見「現代觀點」）。

判別函數與決策區域

白話版。分類器可以寫成一組評分函數，每個類別一個：算出所有分數，選最大的。許多不同的分數會給出相同的決策。

精確版。分類器由**判別函數**（discriminant function）$g_1(x), \dots, g_K(x)$ 以及規則「若對所有 $j \neq k$ 都有 $g_k(x) > g_j(x)$，就把 x 判給 c_k」定義 [1]。取 $g_k(x) = P(c_k \mid x)$ 就得到貝氏最小錯誤率分類器，但對所有判別函數套用任何**單調遞增**函數 f，每個決策都不變。因此以下三種選擇等價：

$$g_k(x) = P(c_k \mid x), \qquad g_k(x) = p(x \mid c_k)P(c_k), \qquad g_k(x) = \ln p(x \mid c_k) + \ln P(c_k).$$

實務上用的是對數形式：它把許多小密度的乘積變成加總，避免數值下溢。最小風險時則用 $g_k(x) = -R(\alpha_k \mid x)$。

c_i 與 c_j 之間的**決策邊界**是 $g_i(x) = g_j(x)$、且兩者都大於其他所有分數的點集合。兩類別時只需要一個函數，稱為**二分器**（dichotomizer）：

$$g(x) = g_1(x) - g_0(x) = \ln \frac{p(x \mid c_1)}{p(x \mid c_0)} + \ln \frac{P(c_1)}{P(c_0)}, \qquad \text{decide } c_1 \text{ if } g(x) > 0.$$

這是對數似然比加上對數先驗勝算。它也等於後驗的 **logit**，因為 $P(c_1 \mid x) = \sigma(g(x))$，其中 $\sigma(z) = 1/(1+e^{-z})$ 是邏輯斯 sigmoid 函數。請記住這個恆等式：在 PR 03 中，邏輯斯迴歸直接把 $g(x)$ 建模成線性函數，而不是從密度推導。

判別函數的觀點很有用，因為它把分類器計算什麼（邊界的形狀）和它如何得到（密度、直接擬合、間隔）分開。線性判別函數 $g_k(x) = w_k^\top x + w_{k0}$ 產生超平面邊界與凸的決策區域；二次判別函數在二維中產生圓錐曲線，一般情形則是超二次曲面。高斯情形正好說明各自何時出現。

高斯情形：LDA 與 QDA 邊界

白話版。如果每個類別都像一團多變量高斯形狀的「雲」，貝氏規則就是在量測 x 與每團雲中心的距離，而且單位會依照雲的形狀調整。所有雲形狀相同時，邊界是直的；形狀不同時，邊界會彎曲。

精確版。假設 $p(x \mid c_k) = \mathcal{N}(x; \mu_k, \Sigma_k)$，平均值 $\mu_k \in \mathbb{R}^d$，共變異數矩陣 $\Sigma_k \in \mathbb{R}^{d \times d}$（對稱、正定）：

$$\mathcal{N}(x; \mu, \Sigma) = \frac{1}{(2\pi)^{d/2} |\Sigma|^{1/2}} \exp\!\Big(-\tfrac{1}{2}(x-\mu)^\top \Sigma^{-1}(x-\mu)\Big),$$

其中 $|\Sigma|$ 是行列式。去掉所有類別共用的常數 $-\frac{d}{2}\ln 2\pi$ 後，對數判別函數為

$$g_k(x) = -\tfrac{1}{2}\,(x-\mu_k)^\top \Sigma_k^{-1}(x-\mu_k) - \tfrac{1}{2}\ln|\Sigma_k| + \ln P(c_k).$$

二次式 $\Delta_k^2(x) = (x-\mu_k)^\top \Sigma_k^{-1}(x-\mu_k)$ 是 x 到 μ_k 的**馬氏距離**（Mahalanobis distance）平方：先把類別「白化」成單位球，再量歐氏距離。圖 4 的三種情形涵蓋了所有可能 [1, 4]。

情形 1：所有 k 都有 $\Sigma_k = \sigma^2 I$。特徵互不相關，變異數皆為 σ^2。展開 $\|x-\mu_k\|^2$ 並丟掉與類別無關的 $x^\top x$，得到線性判別函數

$$g_k(x) = w_k^\top x + w_{k0}, \qquad w_k = \frac{\mu_k}{\sigma^2}, \qquad w_{k0} = -\frac{\mu_k^\top \mu_k}{2\sigma^2} + \ln P(c_k).$$

類別 i 與 j 的邊界是超平面 $w^\top(x - x_0) = 0$，其中 $w = \mu_i - \mu_j$，且

$$x_0 = \tfrac{1}{2}(\mu_i + \mu_j) - \frac{\sigma^2}{\|\mu_i - \mu_j\|^2} \ln \frac{P(c_i)}{P(c_j)}\,(\mu_i - \mu_j).$$

邊界垂直於兩平均值的連線。先驗相等時它通過中點，分類器正是 IP 13 的**最小距離（最近平均值）分類器**。先驗不等時，平面會往遠離較可能類別平均值的方向滑動。

情形 2：所有類別共用 $\Sigma_k = \Sigma$。二次項 $x^\top \Sigma^{-1} x$ 同樣是共用的而被消去，剩下

$$w_k = \Sigma^{-1}\mu_k, \qquad w_{k0} = -\tfrac{1}{2}\mu_k^\top \Sigma^{-1}\mu_k + \ln P(c_k).$$

邊界仍是超平面 $w^\top(x - x_0) = 0$，其中 $w = \Sigma^{-1}(\mu_i - \mu_j)$，但不再垂直於平均值差：它會傾斜，順著共用的相關結構（圖 4 中）。這就是機率形式的**線性判別分析**（linear discriminant analysis,

LDA）。先驗相等時，它是最小馬氏距離分類器。方向 $\Sigma^{-1}(\mu_1 - \mu_0)$ 與 Fisher 在 1936 年由投影準則推得的方向相同，PR 03 會重新推導。

情形 3：任意 Σ_k。沒有任何項會被消去：

$$g_k(x) = x^\top W_k x + w_k^\top x + w_{k0}, \quad W_k = -\tfrac{1}{2}\Sigma_k^{-1}, \quad w_k = \Sigma_k^{-1}\mu_k, \quad w_{k0} = -\tfrac{1}{2}\mu_k^\top\Sigma_k^{-1}\mu_k - \tfrac{1}{2}\ln|\Sigma_k| + \ln P(c_k).$$

邊界是**超二次曲面**：在二維中可能是直線、拋物線、橢圓或雙曲線。這就是**二次判別分析**（quadratic discriminant analysis, QDA）。在圖 4（右）中，緊密的藍色類別被一個橢圓包住，因為在任何方向上離藍色平均值很遠的點，都更能由分布寬廣的紅色類別解釋。決策區域甚至可以不連通，這在線性判別函數中不會發生。

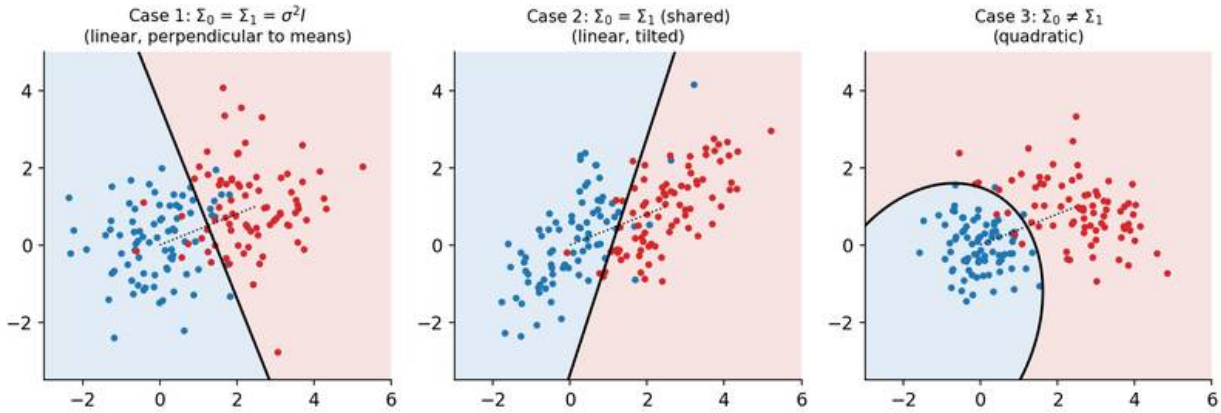


圖 *4* — 先驗相等的兩個高斯類別的貝氏邊界（黑線）。情形 *1*：共用球形共變異數，得到垂直平分線。情形 *2*：共用且相關的共變異數讓直線傾斜（*LDA*）。情形 *3*：共變異數不同，得到二次邊界（*QDA*）。點線連接兩個平均值。

兩類別共用共變異數且先驗相等時，貝氏錯誤率有封閉解：

$$P^*_{\text{err}} = \Phi\!\left(-\frac{\Delta}{2}\right), \qquad \Delta = \sqrt{(\mu_1 - \mu_0)^\top\Sigma^{-1}(\mu_1 - \mu_0)},$$

其中 Φ 是標準常態累積分布函數，Δ 是兩平均值之間的馬氏距離。即使兩類別相距三個標準差（$\Delta = 3$），仍有 $\Phi(-1.5) \approx 6.7\%$ 無法避免的錯誤。在高斯玩具資料上訓練分類器時，這條公式是很好的合理性檢查。

該用哪一種情形？這是偏差與變異數的取捨。每個類別的完整共變異數有 $d(d+1)/2$ 個自由元素；$d = 100$ 時，每類別就是 5,050 個數字，所需的樣本遠多於許多資料集能提供的量。共用一個共變異數（LDA）、限制為對角矩陣（即高斯樸素貝氏），或往簡單目標收縮，都是以彈性換取穩定。如何估計這些參數，是下一章 PR 02：密度估計的主題。

程式碼：從零實作高斯判別函數

這個範例擬合高斯類別模型，用 Cholesky 分解計算對數判別函數（絕不顯式求反矩陣），並在圖 4 情形 3 的資料上，確認決策與 scikit-learn 的 `LinearDiscriminantAnalysis`、`QuadraticDiscriminantAnalysis` 一致。

```
import numpy as np
from sklearn.discriminant_analysis import (LinearDiscriminantAnalysis,
                                           QuadraticDiscriminantAnalysis)

rng = np.random.default_rng(0)
means = [np.array([0.0, 0.0]), np.array([2.5, 1.0])]
covs = [np.array([[0.4, 0.0], [0.0, 0.4]]), np.array([[2.0, -0.6], [-0.6, 1.0]])]
X = np.vstack([rng.multivariate_normal(m, S, 300) for m, S in zip(means, covs)])
y = np.repeat([0, 1], 300)

def fit_gaussians(X, y, shared):
    params = []
    pooled = sum(np.cov(X[y == k].T, bias=True) * (y == k).sum()
                 for k in (0, 1)) / len(y)
    for k in (0, 1):
        Xk = X[y == k]
        S = pooled if shared else np.cov(Xk.T, bias=True)
        params.append((Xk.mean(0), S, len(Xk) / len(y)))
    return params

def discriminants(X, params):
    g = []
    for mu, S, prior in params:
        L = np.linalg.cholesky(S)               # S = L L^T
        z = np.linalg.solve(L, (X - mu).T)      # whitened residuals
        maha = (z ** 2).sum(0)                  # squared Mahalanobis distance
        logdet = 2 * np.log(np.diag(L)).sum()
        g.append(-0.5 * maha - 0.5 * logdet + np.log(prior))
    return np.stack(g, axis=1)

for shared, ref in [(True, LinearDiscriminantAnalysis(store_covariance=True)),
                    (False, QuadraticDiscriminantAnalysis())]:
    mine = discriminants(X, fit_gaussians(X, y, shared)).argmax(1)
    theirs = ref.fit(X, y).predict(X)
    name = "LDA (shared)" if shared else "QDA (separate)"
    print(f"{name}: training error {np.mean(mine != y):.3f}, "
          f"agrees with scikit-learn on {np.mean(mine == theirs):.1%} of points")
```

兩個手寫分類器在每一個點上都與 scikit-learn 一致。這組資料的真實共變異數不同，QDA 的訓練錯誤率（4.3%）低於 LDA（5.5%）。不過這是訓練錯誤率：在高維且樣本少時，保留資料上的排序經常反過來。

ROC 分析與 Neyman–Pearson 觀點

白話版。一個兩類別分數可以在許多門檻上切開。每個門檻都在「抓到更多陽性」與「發出更多假警報」之間取捨。ROC 曲線一次畫出所有取捨，讓你等到成本或先驗確定之後再挑門檻。

精確版。令 $s(x)$ 為任何「越大越偏向 c_1」的分數（對數似然比、後驗、網路的 logit），當 $s(x) > t$ 時判為 c_1。對每個門檻 t 定義

$$\mathrm{TPR}(t) = P(s(x) > t \mid c_1), \qquad \mathrm{FPR}(t) = P(s(x) > t \mid c_0).$$

真陽性率（靈敏度、召回率）是被抓到的陽性比例；偽陽性率（1− 特異度）是被誤標的陰性比例。**接收者操作特徵**（receiver operating characteristic, ROC）曲線在 t 從 $+\infty$ 掃到 $-\infty$ 時，畫出 TPR 對 FPR 的關係 [7]。無用的分數落在對角線上；完美的分數會通過左上角。因為兩個比率都以真實類別為條件，類別先驗改變時，ROC 曲線不會改變。

Neyman–Pearson 準則。有時成本很難說清楚，但限制條件很容易講：「假警報最多 1%」。Neyman–Pearson 引理 [6] 指出：在所有滿足 $\mathrm{FPR} \le \alpha$ 的決策規則中，TPR 最高的是似然比檢定 $\Lambda(x) > \eta$，其中 η 選成讓偽陽性率恰好等於 α（必要時在邊界上隨機化）。所以似然比的 ROC 曲線是可達到的最佳曲線；其他任何分數的 ROC 都在它之下或與之重合。貝氏規則與 Neyman–Pearson 規則都是似然比檢定，差別只在門檻怎麼選。

依成本選擇操作點。在門檻 t 下，每筆樣本的期望成本為

$$\mathbb{E}[\text{cost}] = C_{\mathrm{FP}}\, P(c_0)\, \mathrm{FPR}(t) + C_{\mathrm{FN}}\, P(c_1)\, (1 - \mathrm{TPR}(t)).$$

在 ROC 空間中，成本相同的點落在斜率為 $C_{\mathrm{FP}}P(c_0)/(C_{\mathrm{FN}}P(c_1))$ 的**等成本直線**上；這些直線的斜率固定相同。把直線往較高 TPR（較大的縱軸截距）平移，直到最後接觸可達的點；該點使成本最小 [7]。對似然比分數而言，切點正是先前推導的貝氏門檻。

曲線下面積。 AUC 用一個數字概括整條曲線。它等於隨機抽一個陽性樣本與一個陰性樣本，陽性得分較高的機率（平手算一半）[7]。對兩個相距 d' 個標準差、單位變異數的高斯分數分布，$\mathrm{AUC} = \Phi(d'/\sqrt{2})$：圖 5 中 $d' = 2$ 得到 0.921。AUC 忽略成本、先驗與校準，所以兩個 AUC 相同的分類器，在實際部署的門檻上成本可能差很多。

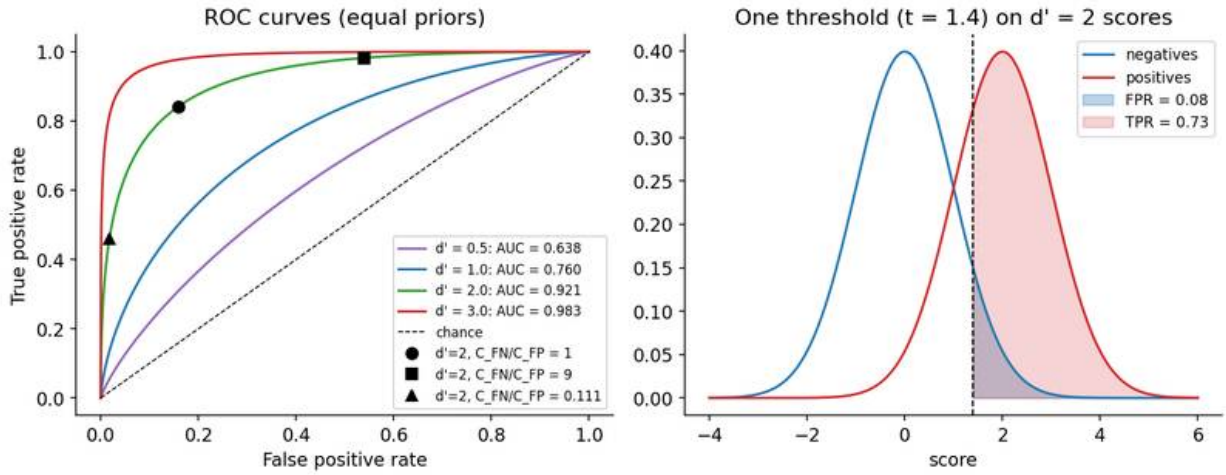


圖 5 — 左：等變異數高斯分數的 *ROC* 曲線；標記為先驗相等時，$d' = 2$ 曲線上三種成本比的貝氏操作點。右：在 $d' = 2$ 的分數上取一個門檻，以及它產生的 *FPR* 與 *TPR*。

精確率–召回率曲線。當陽性很稀少（詐欺、缺陷、偵測中的小物體）時，即使 FPR 很小，假警報的絕對數量仍可能很多，因為陰性實在太多。精確率 $= \mathrm{TP}/(\mathrm{TP} + \mathrm{FP})$ 取決於先驗，能把這件事暴露出來。Davis 與 Goadrich [8] 證明：一條曲線在 ROC 空間中支配另一條，若且唯若它在精確率–召回率空間中也支配；他們也指出 PR 點之間不能用直線內插。Saito 與 Rehmsmeier [9] 以模擬和已發表的研究說明，在不平衡資料上 PR 圖比 ROC 更有資訊量。物件偵測基準因此報告

平均精確率（average precision）（見 DLCV 03）。

程式碼：ROC、AUC 與成本最佳門檻

```
import numpy as np
from sklearn.metrics import roc_auc_score, roc_curve

rng = np.random.default_rng(1)
n_neg, n_pos = 9000, 1000                       # prevalence 10%
scores = np.r_[rng.normal(0, 1, n_neg), rng.normal(2, 1, n_pos)]
labels = np.r_[np.zeros(n_neg), np.ones(n_pos)]

fpr, tpr, thr = roc_curve(labels, scores)
auc = roc_auc_score(labels, scores)
# Mann-Whitney view of AUC: P(score_pos > score_neg)
pos, neg = scores[labels == 1], scores[labels == 0]
auc_pairs = (pos[:, None] > neg[None, :]).mean()
print(f"AUC = {auc:.4f}, pairwise estimate = {auc_pairs:.4f}")

# choose the operating point with minimum expected cost per sample
c_fp, c_fn, prior = 1.0, 9.0, n_pos / (n_pos + n_neg)
cost = c_fp * fpr * (1 - prior) + c_fn * (1 - tpr) * prior
i = np.argmin(cost)
print(f"best threshold {thr[i]:.2f}: FPR {fpr[i]:.3f}, TPR {tpr[i]:.3f}, "
      f"expected cost {cost[i]:.3f}")
# Bayes threshold from the true model: predict 1 when the likelihood ratio
# exceeds c_fp (1 - prior) / (c_fn prior); for unit Gaussians, log LR = 2s - 2
t_bayes = (np.log(c_fp * (1 - prior) / (c_fn * prior)) + 2) / 2
print(f"Bayes threshold from the true densities: {t_bayes:.2f}")
```

經驗 AUC（0.9166）與成對排序估計完全一致，也接近理論值 $\Phi(2/\sqrt{2}) \approx 0.921$。先驗為 10%、成本比為 9:1 時，先驗勝算與成本比互相抵消，貝氏門檻是 $t = 1$；經驗搜尋找到 0.99。實際專案中，門檻搜尋必須在驗證資料上進行，絕不能用測試集。

假設在哪裡失效

貝氏規則的最佳性，是相對於假設的分布與指定的損失而言。部署時，每一項都可能出錯。

先驗偏移。訓練集與部署環境的類別比例可能不同，例如兩座工廠之間，或平衡的研究資料集與真實診所之間。若類別條件分布 $p(x \mid c)$ 不變（**標籤偏移**，label shift），不必重新訓練就能修正後驗：把每個後驗乘上新舊先驗的比值，再重新正規化，

$$P_{\text{new}}(c_k \mid x) \propto P_{\text{old}}(c_k \mid x)\, \frac{P_{\text{new}}(c_k)}{P_{\text{old}}(c_k)}.$$

新先驗未知時，Saerens、Latinne 與 Decaestecker [10] 在未標註的部署資料上用 EM 程序估計；Lipton、Wang 與 Smola 的黑箱偏移估計（black-box shift estimation）[11] 則從分類器的混淆矩陣估計。若新相機改變的是似然本身（**共變量偏移**或**概念偏移**），只修正先驗並不夠。

後驗未校準。本章每個門檻都假設 $P(c \mid x)$ 名副其實。SVM、提升樹或深度網路的分數常有系統性的過度自信或自信不足；在保留資料上重新校準（Platt scaling、保序迴歸、溫度縮放）是標準修正方式 [12, 14]。

標籤雜訊與代理標籤。標籤記錄的可能是一次不完美的檢查，而非物理真相。這時模型學到的是檢查員的後驗，而不是世界的後驗。

單一數字的摘要。兩個準確率相同的分類器，偽陰性率可能不同，期望損失也可能差很多。請一併報告混淆矩陣、操作點與成本。

現代觀點

貝氏決策理論已有半世紀以上的歷史，而它的詞彙（後驗、風險、拒絕、操作點）如今正是可信賴深度學習的詞彙。2017 年以來的研究，大多在處理一件事：當分類器是大型網路時，如何讓餵給貝氏規則的輸入值得信賴。

校準。 Guo 等人 [12] 發現現代深度網路比十年前較淺的網路校準差得多。他們把這與深度、寬度、權重衰減和批次正規化連結起來，並證明**溫度縮放**（temperature scaling，把所有 logit 除以一個在驗證資料上擬合的純量，即單參數版的 Platt scaling）出奇有效。Minderer 等人 [13] 在 2021 年前後的影像分類器上重新檢視這個問題，發現最新的模型，尤其是不使用卷積的模型，屬於校準最好的一群；早期觀察到的「模型越大、或分布偏移時校準越差」的趨勢，在新架構上不那麼明顯。Silva Filho 等人的綜述 [14] 是最好的入門：涵蓋適當評分規則（proper scoring rules）與校準指標、可靠度圖，以及二元與多類別的事後校準方法的完整整理。Gawlikowski 等人 [23] 回顧神經網路不確定性這個更大的領域，區分可降低的模型不確定性與不可降低的資料不確定性，並涵蓋貝氏神經網路、集成與測試時資料擴增。

拒絕與選擇性分類。 Chow 的拒絕規則 [5] 以**選擇性分類**（selective classification）之名被重新發現。Geifman 與 El-Yaniv [16] 示範如何在訓練好的網路外加上拒絕門檻，使被接受樣本上的風險以高機率低於使用者指定的水準，用覆蓋率換取有保證的準確率。Hendrickx 等人的綜述 [15] 形式化兩種拒絕：**模糊拒絕**（Chow 的情形）與**新穎拒絕**（與訓練資料不同的輸入），回顧如何同時評估模型的預測品質與拒絕品質，並分類學習含拒絕選項模型的架構。

不依賴分布的保證。共形預測（conformal prediction）把單一標籤換成一個標籤集合，在「校準資料與測試資料可交換」這個唯一假設下，保證以指定機率（例如 90%）包含真實類別。Angelopoulos 與 Bates [17] 提供附程式碼的實作導論。預測集合很大時，扮演的就是拒絕的角色：模型在告訴你它無法縮小範圍。

開放集辨識與分布外偵測。在 K 個已知類別上正規化的後驗，沒辦法說「以上皆非」。Geng、Huang 與 Chen [18] 綜述開放集辨識（open-set recognition）：定義、模型家族、資料集與評估方式，以及它和含拒絕選項的分類、零樣本與少樣本學習的關係。Yang 等人 [19] 把異常偵測、新穎偵測、開放集辨識、分布外（OOD）偵測與離群值偵測統一為**廣義 OOD 偵測**框架下的子任務。有兩個結果與本章直接相關。Hendrycks 與 Gimpel [20] 指出，**最大 softmax 機率**（也就是 Chow 的統計量）是偵測誤分類與分布外輸入的強大基準。Vaze 等人 [21] 發現分類器說「以上皆非」的能力，與它的封閉集準確率高度相關，因此改善一般分類器就能改善開放集表現。Lee 等人 [22] 在預訓練網路的特徵上擬合**共用共變異數的類別條件高斯**（正是上面的情形 2 ／ LDA），並以馬氏距離作為信心分數。一個 1970 年代風格的生成式分類器，放在學習到的特徵上，至今仍是有競爭力的 OOD 偵測器。

開放問題。成本很少能精確得知，而且會隨時間改變；在一個分布上量到的校準，不一定能移轉到另一個分布；共形覆蓋率這類保證是邊際的（對輸入平均），而非針對特定輸入或子群體的條件保證。損失矩陣應該是什麼、由誰決定，仍是沒有任何估計器能回答的人類問題。

重點整理

- 辨識系統把觀測映射到特徵、特徵映射到信念、信念映射到行動；抽樣單位與資料切分是問題定義的一部分。
- 先驗、似然與後驗回答不同的問題。稀有類別需要強證據：盛行率 2% 時，靈敏度 90% 的檢測得到的後驗低於 20%。
- 最大後驗使錯誤率最小；最小條件風險可處理任何損失矩陣。兩類別時兩者都是似然比檢定，門檻取決於先驗乘上成本。
- 當最大後驗低於 $1 - C_R$ 時，Chow 規則拒絕；它處理模糊性，不處理新穎性。
- 判別函數在單調轉換下等價。高斯類別共用共變異數時邊界為線性（LDA），共變異數不同時為二次（QDA）。
- ROC 曲線把分數與門檻分開；Neyman–Pearson 引理使似然比成為最佳分數；陽性稀少時，PR 曲線更有資訊量。
- 這一切都仰賴校準良好的機率與穩定的先驗，而校準、選擇性分類、共形預測與 OOD 偵測等現代研究，正是在設法確保這兩件事。

練習

1. **不對稱成本**。偽陽性成本為四、偽陰性成本為一時，推導後驗 $p = P(c_1 \mid x)$ 上的二元決策門檻。這條規則現在偏向哪個類別？為什麼？

   提示

   當 $4(1-p) < p$ 時選 c_1，所以門檻是 $p > 0.8$。現在假警報才是昂貴的錯誤，規則需要強證據才會說「陽性」。

2. **拒絕與否**。後驗為 $(0.55, 0.35, 0.10)$、拒絕成本 $C_R = 0.3$，在零一分類損失下選出最小風險行動。C_R 為哪些值時答案會改變？

   提示

   最佳分類風險為 $1 - 0.55 = 0.45 > 0.3$，所以拒絕。當 $C_R \geq 0.45$ 時，系統會分類（判為 c_1）。

3. **移動的邊界**。兩個一維類別 $p(x \mid c_0) = \mathcal{N}(0, 1)$、$p(x \mid c_1) = \mathcal{N}(3, 1)$。求先驗相等與 $P(c_1) = 0.2$ 時的最小錯誤率邊界；再求先驗相等、偽陰性成本為偽陽性四倍時的最小風險邊界。

   提示

   對數似然比為 $3x - 4.5$。先驗相等：$x = 1.5$。$P(c_1) = 0.2$：$3x - 4.5 = \ln 4$，所以 $x \approx 1.96$。$C_{\text{FN}} = 4C_{\text{FP}}$：$3x - 4.5 = \ln(1/4)$，所以 $x \approx 1.04$。先驗與成本透過同一個門檻作用，這裡方向相反。

4. **馬氏距離與歐氏距離**。兩類別共用 $\Sigma = \begin{bmatrix} 4 & 0 \\ 0 & 1 \end{bmatrix}$，平均值 $\mu_0 = (0,0)$、$\mu_1 = (2,1)$，先驗相等。分別用歐氏最近平均值規則與貝氏（LDA）規則分類 $x = (2,0)$，並解釋為何不一致。

   提示

   歐氏距離平方：到 μ_0 為 4，到 μ_1 為 1，所以最近平均值規則判為 c_1。馬氏距離平方：到 μ_0 為 $(2^2/4) + 0 = 1$，到 μ_1 為 $0 + 1 = 1$，平手：x 恰好在 LDA 邊界上。第一個特徵變異很大，所以沿它偏移 2 個單位只是弱證據；歐氏規則高估了它。

5. **從零計算 ROC**。只用 NumPy，把 ROC 程式範例中的分數排序後計算 ROC 曲線與 AUC，並與 `roc_auc_score` 比對。接著把盛行率改成 1%，說明 ROC、AUC 與固定門檻下的精確率中，哪些會改變。

   提示

   把分數由大到小排序；標籤的累積和給出 TP 數，1− 標籤的累積和給出 FP 數；用梯形法把 TPR 對 FPR 積分。ROC 與 AUC 只取決於兩個類別條件分數分布，因此（除抽樣雜訊外）不變；固定門檻下的精確率會大幅下降，因為偽陽性現在來自大得多的陰性群體。

6. **抽樣單位**。某測試集包含與訓練所用影片相同影片中的相鄰影格。說明它的準確率估計的是什麼，並提出能估計新影片、新相機或新受試者表現的切分方式。

   提示

   它估計的是在訓練影格近似複本上的準確率（內插任務）。請以錄影為單位切分（分組切分）；如果部署時會遇到新相機或新的人，也要保留整台相機或整位受試者。

參考文獻

1. R. O. Duda, P. E. Hart and D. G. Stork, *Pattern Classification*, 2nd ed., Wiley, 2001 (Ch. 2, Bayesian decision theory). publisher page
2. C. M. Bishop, *Pattern Recognition and Machine Learning*, Springer, 2006 (Sections 1.2 and 1.5). official page and free PDF
3. S. Theodoridis and K. Koutroumbas, *Pattern Recognition*, 4th ed., Academic Press, 2008 (Ch. 1–2). publisher page
4. T. Hastie, R. Tibshirani and J. Friedman, *The Elements of Statistical Learning*, 2nd ed., Springer, 2009 (Section 4.3, linear discriminant analysis). official page
5. C. K. Chow, “On optimum recognition error and reject tradeoff,” *IEEE Transactions on Information Theory*, vol. 16, no. 1, pp. 41–46, 1970. doi
6. J. Neyman and E. S. Pearson, “On the problem of the most efficient tests of statistical hypotheses,” *Philosophical Transactions of the Royal Society of London, Series A*, vol. 231, pp. 289–337, 1933. doi
7. T. Fawcett, “An introduction to ROC analysis,” *Pattern Recognition Letters*, vol. 27, no. 8, pp. 861–874, 2006. doi

8. J. Davis and M. Goadrich, “The relationship between precision-recall and ROC curves,” in *Proc. International Conference on Machine Learning* (*ICML*), 2006. doi
9. T. Saito and M. Rehmsmeier, “The precision-recall plot is more informative than the ROC plot when evaluating binary classifiers on imbalanced datasets,” *PLOS ONE*, vol. 10, no. 3, e0118432, 2015. doi
10. M. Saerens, P. Latinne and C. Decaestecker, “Adjusting the outputs of a classifier to new a priori probabilities: a simple procedure,” *Neural Computation*, vol. 14, no. 1, pp. 21–41, 2002. doi
11. Z. C. Lipton, Y.-X. Wang and A. Smola, “Detecting and correcting for label shift with black box predictors,” in *Proc. ICML*, PMLR 80, 2018. paper
12. C. Guo, G. Pleiss, Y. Sun and K. Q. Weinberger, “On calibration of modern neural networks,” in *Proc. ICML*, PMLR 70, 2017. paper
13. M. Minderer, J. Djolonga, R. Romijnders, F. Hubis, X. Zhai, N. Houlsby, D. Tran and M. Lucic, “Revisiting the calibration of modern neural networks,” in *Advances in Neural Information Processing Systems* (*NeurIPS*), 2021. paper
14. T. Silva Filho, H. Song, M. Perello-Nieto, R. Santos-Rodriguez, M. Kull and P. Flach, “Classifier calibration: a survey on how to assess and improve predicted class probabilities,” *Machine Learning*, vol. 112, pp. 3211–3260, 2023. doi
15. K. Hendrickx, L. Perini, D. Van der Plas, W. Meert and J. Davis, “Machine learning with a reject option: a survey,” *Machine Learning*, vol. 113, pp. 3073–3110, 2024. doi
16. Y. Geifman and R. El-Yaniv, “Selective classification for deep neural networks,” in *Advances in Neural Information Processing Systems* (*NeurIPS*), 2017. paper
17. A. N. Angelopoulos and S. Bates, “Conformal prediction: a gentle introduction,” *Foundations and Trends in Machine Learning*, vol. 16, no. 4, pp. 494–591, 2023. doi
18. C. Geng, S.-J. Huang and S. Chen, “Recent advances in open set recognition: a survey,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 43, no. 10, pp. 3614–3631, 2021. doi
19. J. Yang, K. Zhou, Y. Li and Z. Liu, “Generalized out-of-distribution detection: a survey,” *International Journal of Computer Vision*, vol. 132, pp. 5635–5662, 2024. doi
20. D. Hendrycks and K. Gimpel, “A baseline for detecting misclassified and out-of-distribution examples in neural networks,” in *Proc. International Conference on Learning Representations* (*ICLR*), 2017. arXiv
21. S. Vaze, K. Han, A. Vedaldi and A. Zisserman, “Open-set recognition: a good closed-set classifier is all you need,” in *Proc. ICLR*, 2022. arXiv
22. K. Lee, K. Lee, H. Lee and J. Shin, “A simple unified framework for detecting out-of-distribution samples and adversarial attacks,” in *Advances in Neural Information Processing Systems* (*NeurIPS*), 2018. paper
23. J. Gawlikowski et al., “A survey of uncertainty in deep neural networks,” *Artificial Intelligence Review*, vol. 56, suppl. 1, pp. 1513–1589, 2023. doi

PR 02・密度估計與貝氏學習

估計貝氏分類器所需的密度：最大似然、MAP 與完整貝氏估計；直方圖、Parzen 窗與核頻寬選擇；k-NN 密度與 k-NN 分類器；以及維度詛咒。

網站版本: https://blog.shyandram.dev/zh-tw/learn/pattern-recognition/density-estimation/

你將學到

- 為什麼貝氏分類器需要密度估計，以及參數式與非參數式估計的差別。
- 如何推導高斯分布的最大似然（ML）估計、為何樣本少時共變異數會奇異，以及收縮（shrinkage）如何解決。
- MAP 與完整貝氏估計如何使用參數的先驗，並以高斯平均值與 Bernoulli 比率實際推導。
- 直方圖、Parzen 窗（核密度估計）與 k 近鄰估計如何由同一條公式推出，以及如何選擇頻寬或 k。
- k-NN 分類器如何從 k-NN 密度估計推得、Cover–Hart 上界說了什麼，以及為何所有局部方法都受維度詛咒所苦。

先看全貌

PR 01 說明了最佳分類器需要類別先驗 $P(c_k)$ 與類別條件密度 $p(x \mid c_k)$。實務上沒有人會把這些分布交給我們；我們只有一組有限的標註樣本。本章談的是如何把樣本變成密度。有了 $\hat{p}(x \mid c_k)$ 與 $\hat{P}(c_k)$，就能代入貝氏規則。這是分類的**生成式**（generative）路線。下一章則走**判別式**（discriminative）路線，直接擬合決策邊界。

> **用最簡單的話說**
>
> 每次在公園看到一隻鳥，就在地圖上貼一張貼紙。現在猜猜下一隻鳥會出現在哪裡。你可以在大部分貼紙外面畫一**個大橢圓**（參數式模型：只要決定橢圓的中心和形狀）。也可以在地圖上鋪**格子**，數每一格有幾張貼紙（直方圖）。或者在每張貼紙上堆一**小座沙丘**，看沙子在哪裡堆得最高（核估計）。每一種都是不同的猜法，也各有失敗的可能：如果鳥喜歡兩個不同的池塘，一個橢圓就太簡單；如果只有五張貼紙，沙丘又會太凹凸不平。

估計方法分成兩大類 [1, 2, 3]：

- **參數式**（parametric）方法假設一種函數形式（例如高斯），並估計固定數量的參數。形式正確時需要的資料很少，形式錯誤時則會系統性地出錯。
- **非參數式**（non-parametric）方法讓資料決定形狀，複雜度隨樣本數增加。它們很有彈性，但需要多得多的資料，在高維時尤其如此。

兩類方法都面對同一個核心問題：手上的資料能支撐多少結構？一個有彈性的估計器可以把訓練資料描述得很漂亮，卻依然對新資料預測得很差。下面每個方法都有一個「平滑旋鈕」（參數數量、先驗、組距、頻寬、k），把它調好比選對方法更重要。

最大似然估計

白話版。選擇讓觀測到的資料出現機率最高的參數值。

精確版。令 $D = \{x_1, \dots, x_N\}$ 為從密度 $p(x \mid \theta)$ 獨立抽出的樣本，參數向量 θ 未知。θ 的**似然**是 $p(D \mid \theta) = \prod_n p(x_n \mid \theta)$，ML 估計最大化它的對數：

$$\hat{\theta}_{\mathrm{ML}} = \arg\max_{\theta} \ell(\theta), \qquad \ell(\theta) = \sum_{n=1}^{N} \log p(x_n \mid \theta).$$

獨立性假設談的是抽樣（不同觀測互不影響），而不是同一個特徵向量內各座標彼此獨立。做分類器時，我們為每個類別各擬合一個模型，只用該類別的樣本，並以類別頻率 $\hat{P}(c_k) = N_k/N$ 估計先驗。

高斯情形

d 維高斯 $\mathcal{N}(\mu, \Sigma)$ 的對數似然為

$$\ell(\mu, \Sigma) = -\frac{N}{2}\log|\Sigma| - \frac{1}{2}\sum_{n}(x_n - \mu)^{\top}\Sigma^{-1}(x_n - \mu) + \text{const}.$$

令對 μ 的梯度為零，得到 $\Sigma^{-1}\sum_n (x_n - \mu) = 0$。對 Σ^{-1} 微分（利用 $\partial \log|A|/\partial A = A^{-\top}$ 與 $\partial\, a^{\top}Aa/\partial A = aa^{\top}$），得到估計值

$$\hat{\mu} = \frac{1}{N}\sum_{n} x_n, \qquad \hat{\Sigma} = \frac{1}{N}\sum_{n}(x_n - \hat{\mu})(x_n - \hat{\mu})^{\top}.$$

樣本平均是不偏的。ML 共變異數則略有**偏差**：$\mathbb{E}[\hat{\Sigma}] = \frac{N-1}{N}\Sigma$，因為 $\hat{\mu}$ 是用同一批資料擬合的，比真實平均更貼近這些資料。常見的樣本共變異數除以 $N-1$ 來消除這個偏差。N 很大時，兩者差異可以忽略。

ML 失效之時：參數太多

完整共變異數有 $d(d+1)/2$ 個獨立元素。$d = 64$ 個特徵時，每個類別就是 2,080 個數字，平均值還要再加 64 個。更糟的是，$\hat{\Sigma}$ 是以 $\hat{\mu}$ 為中心的 N 個秩一矩陣之和，所以秩最多為 $N-1$。**當一個類別的樣本數不超過維度時，它的 ML 共變異數是奇異的**：某些方向上變異數為零，$\log|\hat{\Sigma}| = -\infty$，PR 01 的高斯判別函數無法定義。即使 N 只比 d 稍大，最小的特徵值也會被系統性低估，而反矩陣 $\hat{\Sigma}^{-1}$ 會大幅放大這些方向上的雜訊。

正則化把共變異數往較簡單的目標**收縮**：

$$\tilde{\Sigma} = (1 - \alpha)\hat{\Sigma} + \alpha\tau I, \qquad \tau = \mathrm{tr}(\hat{\Sigma})/d, \qquad 0 \le \alpha \le 1.$$

其中 τI 是平均變異數相同的球形矩陣，α 是收縮強度。收縮把大的特徵值往下拉、小的往上推，用一點偏差換來條件數好得多的估計。Ledoit 與 Wolf [14] 推導出使期望平方誤差最小的 α 的封閉式估計，因此不需要交叉驗證。其他常見的限制有**對角**共變異數（忽略相關；即下文的高斯樸素貝氏），以及各類別**共用**共變異數（匯集證據；即 PR 01 的 LDA）。這些是不同的建模假設，不是可以互換的數值技巧。

程式碼：ML 共變異數何時失效

```
import numpy as np
from scipy.stats import multivariate_normal
from sklearn.covariance import EmpiricalCovariance, LedoitWolf

rng = np.random.default_rng(0)
d = 40
A = rng.normal(size=(d, d))
true_cov = A @ A.T / d + 0.5 * np.eye(d)          # a well-conditioned truth
test = rng.multivariate_normal(np.zeros(d), true_cov, 2000)

for n in [30, 60, 200, 1000]:
    train = rng.multivariate_normal(np.zeros(d), true_cov, n)
    mu = train.mean(0)
    S_ml = EmpiricalCovariance().fit(train).covariance_   # divisor n
    S_lw = LedoitWolf().fit(train)
    rank = np.linalg.matrix_rank(S_ml)
    ll_ml = (multivariate_normal(mu, S_ml, allow_singular=True).logpdf(test).mean()
             if rank == d else -np.inf)
    ll_lw = multivariate_normal(mu, S_lw.covariance_).logpdf(test).mean()
    print(f"n={n:4d}: rank(S_ML)={rank:2d}, test log-lik ML={ll_ml:8.2f}, "
          f"Ledoit-Wolf={ll_lw:8.2f} (shrinkage {S_lw.shrinkage_:.2f})")
```

$d = 40$、$n = 30$ 時，ML 共變異數的秩是 29，幾乎對每個測試點都給出零密度（對數似然 $-\infty$）。$n = 60$ 時它可逆但很差（測試對數似然 -94.6，Ledoit–Wolf 為 -63.0）。自動選出的收縮強度隨 n 增加從 0.65 降到 0.08，$n = 1000$ 時兩種估計幾乎重合。經驗法則：每個參數分到的樣本越少，就該收縮得越多。

MAP 與貝氏估計

白話版。最大似然完全相信資料。貝氏估計則從對參數的先驗信念出發，讓資料逐步推翻它。資料少時先驗很重要；資料多時幾乎無關緊要。

精確版。把 θ 視為隨機變數，給它**參數先驗** $p(\theta)$。貝氏定理給出參數後驗

$$p(\theta \mid D) = \frac{p(D \mid \theta)\, p(\theta)}{\int p(D \mid \theta')\, p(\theta')\, d\theta'} \;\propto\; p(D \mid \theta)\, p(\theta).$$

參數先驗與 PR 01 的類別先驗不同。類別先驗說的是某個類別有多常見；參數先驗說的是在看到這份訓練集之前，哪些模型設定是合理的。後驗常見的用法有三種：

- **MAP 估計**只保留眾數：$\hat{\theta}_{\text{MAP}} = \arg\max_\theta \left[\log p(D \mid \theta) + \log p(\theta)\right]$。對數先驗扮演正則項；權重上的高斯先驗正好是 L_2（ridge）懲罰，PR 03 會用到。
- **完整貝氏預測**對所有參數值做平均，而不是只挑一個：

$$p(x_{\text{new}} \mid D) = \int p(x_{\text{new}} \mid \theta)\, p(\theta \mid D)\, d\theta.$$

這個**後驗預測**分布把參數的不確定性帶進預測。

- **代入式預測**（plug-in）使用 $p(x_{\text{new}} \mid \hat{\theta})$，其中 $\hat{\theta}$ 來自 ML 或 MAP。它忽略參數不確定性，因此資料稀少時會過度自信。

有一個細節：MAP 對重新參數化不具不變性。σ 上密度的眾數，不等於對應的 σ^2 上密度眾數的平方根，因為變數變換時密度會多出 Jacobian 因子。ML 估計與完整後驗沒有這個問題。

實例：高斯的平均值

令 $x_n \sim \mathcal{N}(\mu, \sigma^2)$，$\sigma^2$ 已知，並給未知平均值高斯先驗 $\mu \sim \mathcal{N}(m_0, s_0^2)$。把似然與先驗相乘，對 μ 配方後可知後驗仍是高斯，$\mu \mid D \sim \mathcal{N}(m_N, s_N^2)$，其中 [1, 2]

$$\frac{1}{s_N^2} = \frac{1}{s_0^2} + \frac{N}{\sigma^2}, \qquad m_N = s_N^2 \left(\frac{m_0}{s_0^2} + \frac{N\bar{x}}{\sigma^2} \right),$$

$\bar{x}$ 是樣本平均。有兩種讀法能讓這條公式好記。**精度相加**：後驗精度 $1/s_N^2$ 等於先驗精度加上 N 倍單一觀測的精度。**平均值是精度加權平均**：先驗平均與樣本平均依精度加權。當 $N \to \infty$，$m_N \to \bar{x}$（ML 估計）且 $s_N \to 0$。因為後驗是高斯，眾數等於平均，所以這裡 $\hat{\mu}_{\text{MAP}} = m_N$。後驗預測分布為 $\mathcal{N}(m_N,\, \sigma^2 + s_N^2)$：比代入式的 $\mathcal{N}(m_N, \sigma^2)$ 寬，多出的部分就是對 μ 剩餘的不確定性。

後驗與先驗屬於同一分布族的先驗稱為**共軛**（conjugate）先驗。共軛配對（高斯–高斯、Beta–Bernoulli、Dirichlet–類別分布、平均與共變異數的 Normal–inverse-Wishart）都有封閉式更新；其他先驗則需要 PR 08 的近似推論方法。

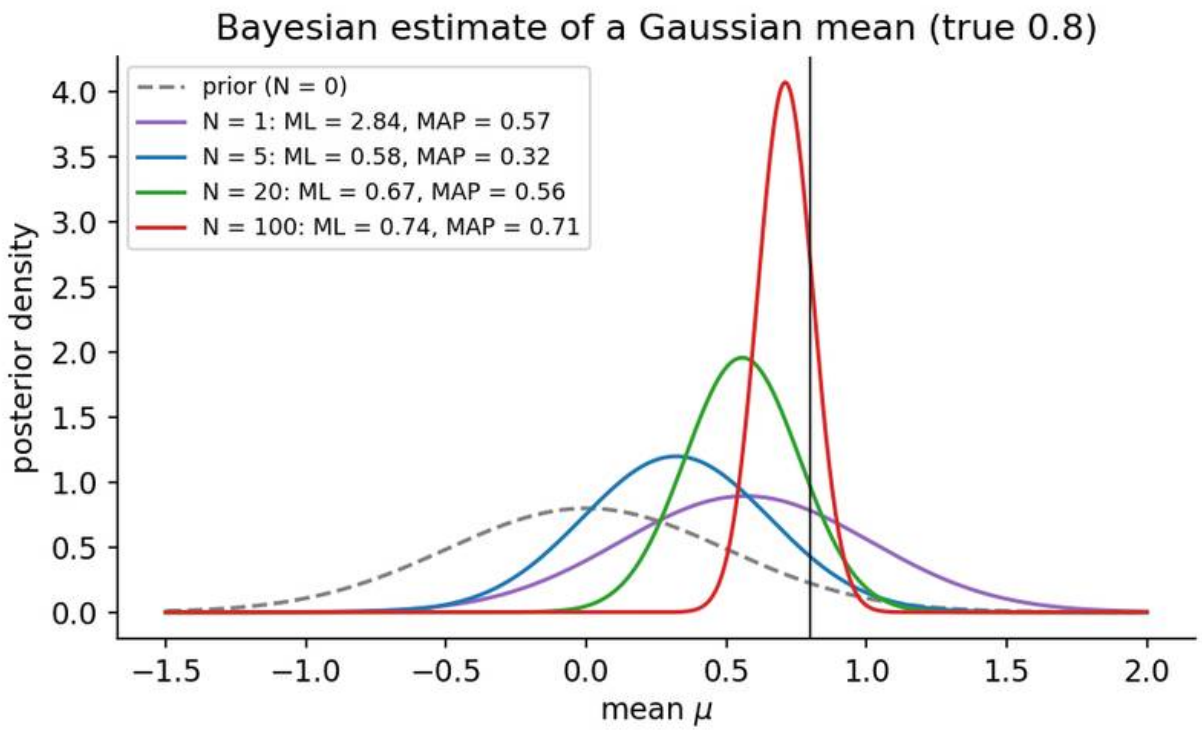


圖 *1* — 高斯平均值的貝氏估計（真值 *0.8*，先驗 $N(0,\,0.5^2)$）。只有一筆樣本時，*ML* 估計（*2.84*）偏得很遠，後驗平均被拉回到 *0.57*；有 *100* 筆樣本時，*ML*（*0.74*）與 *MAP*（*0.71*）幾乎一致，後驗也很窄。

實例：Bernoulli 比率

對 Bernoulli 比率 q（例如產線的缺陷率）給定先驗 $\mathrm{Beta}(a, b)$，在 N 次試驗中觀測到 s 次成功後，

$$q \mid D \sim \mathrm{Beta}(a+s,\; b+N-s), \qquad p(x_{\text{new}}=1 \mid D) = \frac{a+s}{a+b+N}.$$

先驗就像 a 次假想的成功與 b 次假想的失敗。一次成功、零次失敗時，ML 預測機率為一，這顯然荒謬。均勻先驗 $\mathrm{Beta}(1,1)$ 預測 $2/3$，這就是 Laplace 的接續法則（rule of succession）。同樣的概念稱為**加法平滑**或 **Laplace 平滑**，它能避免樸素貝氏文字分類器只因某個字在訓練時從未與某類別一起出現，就給整份文件零機率。

程式碼：ML、MAP 與後驗預測

```
import numpy as np

rng = np.random.default_rng(3)
sigma = 1.0                 # known noise standard deviation
m0, s0 = 0.0, 0.5           # prior on the mean: N(m0, s0^2)
```

```
x = rng.normal(0.8, sigma, 5)
n = len(x)

mu_ml = x.mean()
post_var = 1 / (1 / s0**2 + n / sigma**2)
post_mean = post_var * (m0 / s0**2 + x.sum() / sigma**2)   # = MAP for a Gaussian
pred_var = sigma**2 + post_var                              # predictive variance

# brute-force check on a grid: posterior is proportional to likelihood x prior
grid = np.linspace(-3, 3, 200001)
log_post = (-0.5 * ((x[:, None] - grid) ** 2).sum(0) / sigma**2
            - 0.5 * (grid - m0) ** 2 / s0**2)
w = np.exp(log_post - log_post.max())
w /= w.sum()
print(f"ML {mu_ml:.3f} | MAP/posterior mean {post_mean:.3f} "
      f"(grid {np.sum(w * grid):.3f}) | posterior sd {np.sqrt(post_var):.3f}")
print(f"predictive sd {np.sqrt(pred_var):.3f} vs plug-in sd {sigma:.3f}")
```

五筆樣本時，ML 平均為 0.577，後驗平均為 0.320（網格積分結果一致），後驗標準差為 0.333。預測標準差 1.054 比代入值寬 5%：這裡不大，但資料越少或維度越高時會迅速變大。

由密度建立分類器

白話版。為每個類別擬合一個密度，乘上類別頻率，選乘積最大的類別。每個密度怎麼建模，決定了分類器能學到什麼。

高斯代入式分類器。估計 $\hat{\mu}_k$、$\hat{\Sigma}_k$ 與 $\hat{P}(c_k)$，代入 PR 01 的高斯判別函數。各類別共變異數分開就是 QDA，共用（匯集）共變異數就是 LDA，邊界的幾何形狀與 PR 01 推導的一樣。計算時請用 Cholesky 分解或解線性方程組，不要顯式求反矩陣。

樸素貝氏。假設特徵**在給定類別下條件獨立**：

$$p(x \mid c_k) = \prod_{j=1}^{d} p(x_j \mid c_k).$$

每個類別只需要 d 個一維密度，而不是一個 d 維密度，因此很少的資料就能估計。它並不假設特徵在整體母體中獨立；它假設的是類別解釋了特徵之間所有的相依性。高斯樸素貝氏為每個座標在每個類別中各給一個一維高斯；當各類別的變異數不同時，邊界是二次的。當特徵高度相關（或重複）時，樸素貝氏會把同一份證據計算好幾次，所以即使決策仍算合理，後驗也會過度自信。

生成式與判別式。Ng 與 Jordan [15] 比較了樸素貝氏與它的判別式對應方法邏輯斯迴歸。他們證明：在資料無限多的極限下，邏輯斯迴歸的錯誤率較低（或相等）；但樸素貝氏以快得多的速度逼近它自己較高的漸近錯誤率，所需樣本數與維度呈對數關係，而非線性關係。所以樣本少時生成式模型常勝出，樣本多時判別式模型勝出。這是理解兩條路線取捨的好心智模型。

非參數估計：一條公式，三種估計器

白話版。要估計某一點的密度，就看它周圍的一個小區域，數有多少樣本落在裡面。密度就是「個數除以（樣本總數乘上區域大小）」。你可以固定區域去數個數，也可以固定個數去擴大區域。

精確版。樣本落在 x 周圍小區域 $\mathcal{R}$ 內的機率為 $P = \int_{\mathcal{R}} p(x')\,dx' \approx p(x)\,V$，其中 V 是 $\mathcal{R}$ 的體積。若 N 個樣本中有 k 個落在裡面，$k \approx NP$，所以 [1, 2]

$$\hat{p}(x) = \frac{k}{N\,V}.$$

這裡藏了兩個近似，而且方向相反。$P \approx p(x)V$ 需要小區域（否則 $\hat{p}$ 是 p 的模糊平均：**偏差**）。$k \approx NP$ 需要區域裡有很多樣本（否則計數充滿雜訊：**變異數**）。Duda、Hart 與 Stork [1] 證明：當 $N \to \infty$ 時，若區域縮小（$V_N \to 0$）而計數增加（$k_N \to \infty$），但增加速度比 N 慢（$k_N/N \to 0$），估計就會收斂到 $p(x)$。有兩種滿足方式：

- **固定** V、**計數** k：直方圖與 Parzen 窗（核密度估計）；
- **固定** k、**量測** V：k 近鄰密度估計。

直方圖

把軸切成寬 w 的組，對落在第 i 組（計數為 n_i）的 x，令 $\hat{p}(x) = n_i/(Nw)$。直方圖簡單又快，但有三個弱點，見圖 2。估計在組界處不連續。它除了組距之外還取決於網格的**起點**：把所有組平移 0.3，圖形就變了。而在 d 維中，組數以 $(1/w)^d$ 成長，遠在 d 到達十之前，大部分的組就是空的。Scott 的分析 [8] 顯示，使平均積分平方誤差最小的組距只以 $N^{-1/3}$ 的速度縮小，所以更多資料只能慢慢換來更細的組。

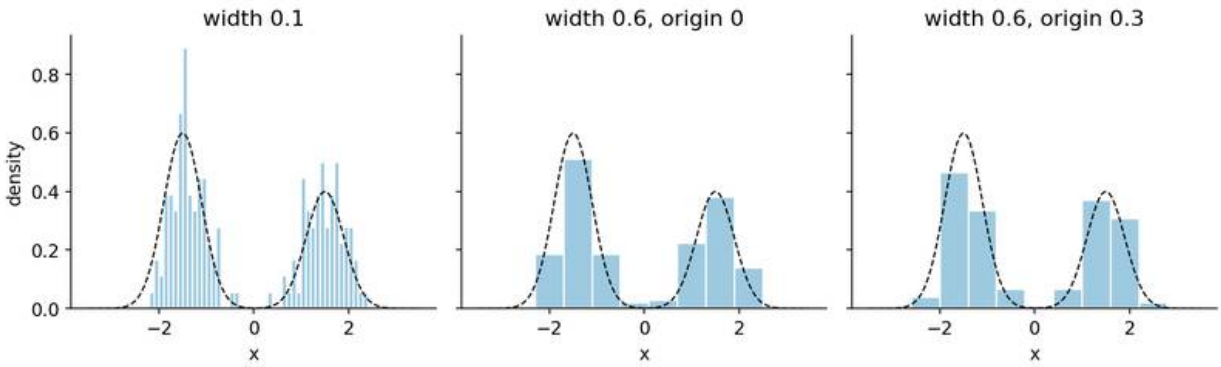


圖 2 — 同樣的 *180* 筆樣本（虛線為真實密度）。組距窄則雜訊多；組距寬則取決於網格從哪裡開始。右邊兩張只差在組的起點。

Parzen 窗與核密度估計

Parzen 窗或**核密度估計**（kernel density estimate, KDE）由 Rosenblatt [5] 提出、Parzen [6] 加以分析，它不固定網格，而是在每個樣本上放一個窗：

$$\hat{p}_h(x) = \frac{1}{N h^d} \sum_{n=1}^{N} K\!\left(\frac{x - x_n}{h}\right).$$

其中 K 是**核**，滿足 $K(u) \ge 0$ 且 $\int K(u)\,du = 1$（所以 $\hat{p}_h$ 是合法的密度），$h > 0$ 是**頻寬**（bandwidth），扮演組距的角色。用盒狀核（在單位立方體內 $K = 1$）時，這條公式字面上就是「數出 x 周圍邊長 h 的立方體內有幾個樣本」，也就是上面固定 V 的估計器。高斯核 $K(u) = (2\pi)^{-d/2} e^{-\|u\|^2/2}$ 把硬性計數換成平滑權重，得到平滑的估計（圖 3 左）。實務上，核的形狀遠不如頻寬重要。

偏差與變異數。對一維的平滑密度，KDE 的偏差約隨 h^2（乘上曲率 $p''(x)$）成長，變異數約隨 $1/(Nh)$ 成長。兩者之和最小時，最佳頻寬與 $N^{-1/5}$ 成正比，平均積分平方誤差以 $N^{-4/5}$ 下降，比形式正確的參數模型的 N^{-1} 慢 [7, 8]。在 d 維中，最佳頻寬按 $N^{-1/(d+4)}$ 縮放，誤差按 $N^{-4/(d+4)}$ 縮放。隨 d 變慢，正是維度詛咒的一個面向。

選擇頻寬。這是實務上最重要的決定。常見選項：

- **經驗法則**。 Silverman 法則 $h = 0.9\,\min(\hat{\sigma},\ \mathrm{IQR}/1.34)\,N^{-1/5}$ [7]，其中 $\hat{\sigma}$ 是樣本標準差、IQR 是四分位距；資料為高斯時它是最佳的，對多峰資料則會過度平滑。
- **交叉驗證**。選擇讓保留樣本對數似然最大的 h（似然交叉驗證），或讓積分平方誤差的不偏估計最小的 h（最小平方交叉驗證，Scott 與 Terrell [9] 有分析）。
- **代入法**。 Sheather 與 Jones [10] 從資料估計最佳頻寬公式中未知的曲率項並代入；他們的選擇器是一維情形下廣泛使用、可靠的預設值。

無論用哪一種規則，都不要看最終測試資料來選 h。圖 4 說明選擇為何重要：在雙峰樣本上，$h = 0.08$ 產生虛假的小峰，$h = 1.2$ 把兩個峰合而為一，驗證對數似然則正確地偏好 $h = 0.2$。

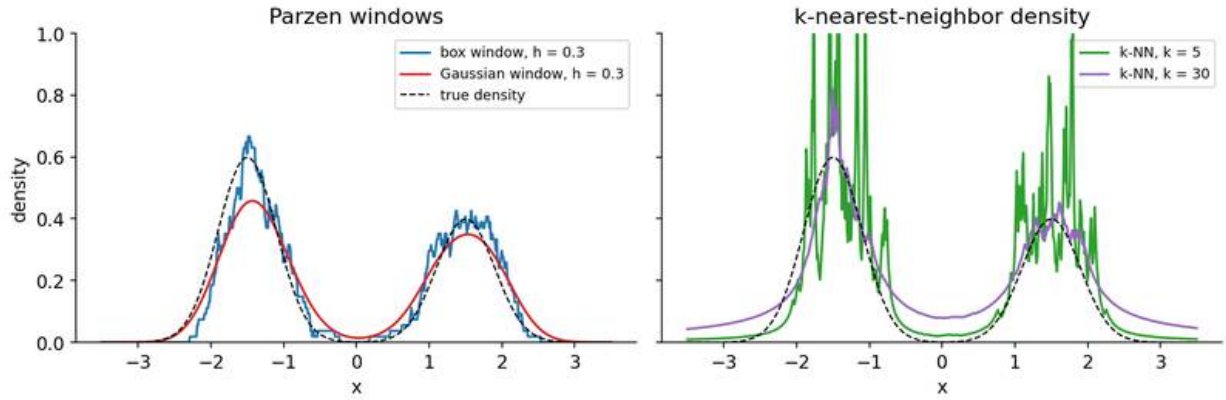


圖 3 — 左：盒狀窗得到鋸齒狀估計，同寬的高斯窗則得到平滑估計。右：*k-NN* 密度估計會依資料調整寬度，但 k 小時很尖刺，而且尾巴衰減得太慢。

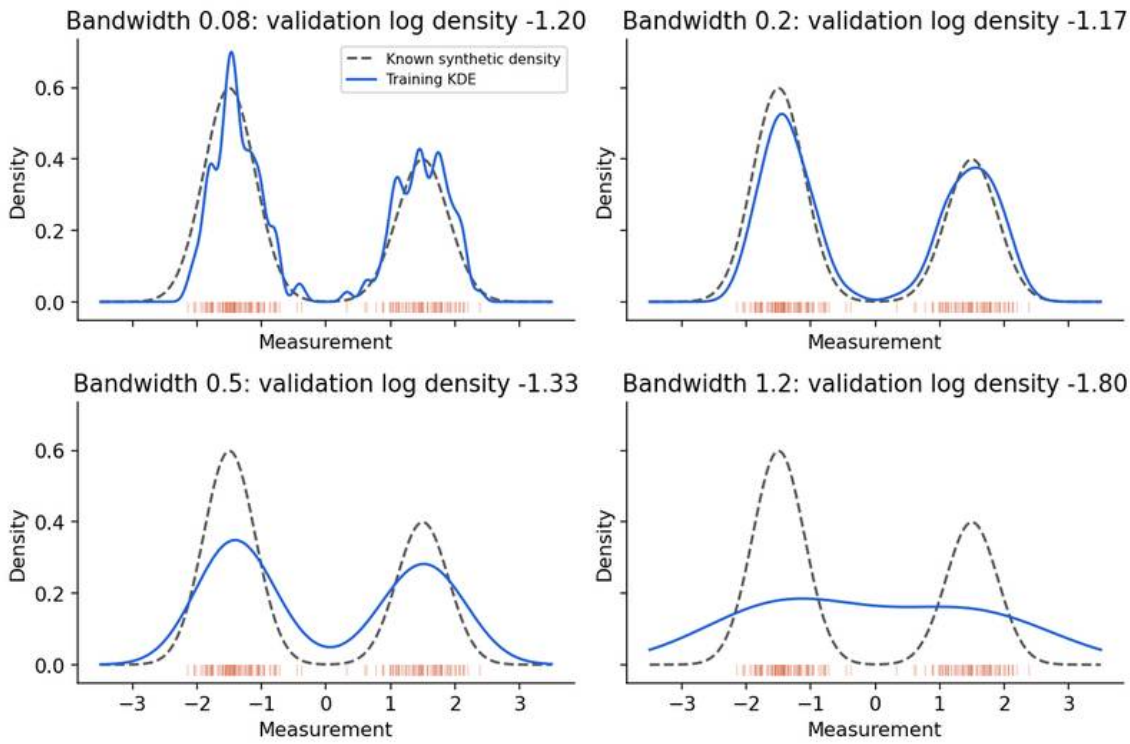


圖 *4* — 在下方程式碼所用的訓練樣本上，以四種頻寬做 *KDE*。窄頻寬顯露抽樣波動；寬頻寬把峰模糊掉。每張標題中的平均驗證對數密度在 $h = 0.2$ 時最高。

程式碼：頻寬選擇與 Beta 後驗

這個範例在合成的雙峰分布上使用 scikit-learn 的 `KernelDensity` [26]，搭配獨立的驗證樣本、用於比較的 Silverman 法則，以及上一節的封閉式 Beta 後驗。

```
import numpy as np
from scipy.stats import beta
from sklearn.neighbors import KernelDensity

rng = np.random.default_rng(4)
def sample(n):
    means = np.where(rng.random(n) < 0.6, -1.5, 1.5)
    return (means + rng.normal(0, 0.4, n)).reshape(-1, 1)
train, validation = sample(180), sample(100)
bandwidths = [0.08, 0.2, 0.5, 1.2]
scores = []
for h in bandwidths:
    model = KernelDensity(bandwidth=h).fit(train)
    scores.append(model.score(validation) / len(validation))
best = bandwidths[int(np.argmax(scores))]

# Silverman's rule of thumb for comparison
```

```
x = train.ravel()
iqr = np.subtract(*np.percentile(x, [75, 25]))
h_silverman = 0.9 * min(x.std(ddof=1), iqr / 1.34) * len(x) ** (-1 / 5)

posterior = beta(2, 1)  # uniform prior + one success
print("validation mean log densities:",
      {h: round(float(s), 3) for h, s in zip(bandwidths, scores)})
print("selected bandwidth:", best, "| Silverman's rule:", round(h_silverman, 3))
print("posterior mean:", posterior.mean())
print("95% credible interval:", posterior.ppf([0.025, 0.975]))
assert np.isclose(posterior.mean(), 2 / 3)
```

驗證選出 $h = 0.2$，Silverman 法則則給出 $h \approx 0.475$。這條法則假設資料是單一高斯；而這份樣本的分散主要來自兩峰之間的間隔，所以法則過度平滑。均勻先驗下觀測一次成功後，Beta 後驗的平均為 $2/3$，95% 可信區間約為 $[0.16, 0.99]$：一筆觀測能告訴我們的非常有限。

可信區間描述的是在假設的先驗與似然下的不確定性，並不自動等於校準過的頻率學派覆蓋率保證。同樣地，密度分數好，也不保證分類邊界好：模型描述的某些變異可能與標籤無關。

k 近鄰密度估計

改為固定 k 而非 V：以 x 為中心擴大一個球，直到剛好包含 k 個樣本，令 $V_k(x)$ 為球的體積。則 [11]

$$\hat{p}_k(x) = \frac{k}{N\, V_k(x)}.$$

Loftsgaarden 與 Quesenberry [11] 證明，當 $k \to \infty$ 且 $k/N \to 0$ 時這個估計是一致的。它的頻寬會自動調整：資料密集處窄、稀疏處寬。但圖 3（右）顯示了它的缺點。k 小時非常尖刺，因為樣本聚集處的 V_k 很小。在尾部，一維中 V_k 只隨距離線性成長，所以 $\hat{p}_k$ 以 $1/|x|$ 衰減，估計**積分不為一**（積分發散）。因此它是很差的密度估計，但正如下一節所示，它是很好的分類基礎。

k 近鄰分類器

白話版。要分類一個新點，找出離它最近的 k 個訓練點，然後多數決。不需要訓練模型；訓練集本身就是模型。

精確版。在 x 周圍畫一個總共包含 k 個樣本的球，其中 k_c 個屬於類別 c。對每個類別分別套用 $k/(NV)$ 估計，類別 c 有 N_c 個樣本，得到

$$\hat{p}(x \mid c) = \frac{k_c}{N_c V}, \qquad \hat{P}(c) = \frac{N_c}{N}, \qquad \hat{p}(x) = \frac{k}{NV} \quad \Longrightarrow \quad \hat{P}(c \mid x) = \frac{\hat{p}(x \mid c)\hat{P}(c)}{\hat{p}(x)} = \frac{k_c}{k}.$$

體積消去了，後驗估計就是各類別鄰居所佔的比例。於是貝氏規則成為**多數決**。$k = 1$ 時就是**最近鄰規則**。

它有多好？ Cover 與 Hart [12] 證明，當 $N \to \infty$ 時，1-NN 規則的錯誤率 $P_{1\mathrm{NN}}$ 滿足

$$P^* \le P_{1\text{NN}} \le P^*\Big(2 - \frac{K}{K-1}P^*\Big) \le 2P^*,$$

其中 P^* 是貝氏錯誤率，K 是類別數。這是在定理取樣假設下的錯誤率上界，不是說最近鄰含有資訊理論意義下「一半的資訊」。隨著 k 增加（且 $k/N \to 0$），k-NN 的錯誤率會逼近 P^* 本身。這是漸近的敘述；資料有限且維度很多時，情況就沒那麼樂觀。

選擇 k。 k 小則偏差低、變異數高：圖 5 中 1-NN 的邊界會繞著個別的雜訊點打轉。k 大則邊界平滑，但最後會忽略真正的結構。請用交叉驗證選 k。兩個實務細節：兩類別時用奇數 k 以避免平手，並且先**標準化特徵**，因為歐氏距離會被數值範圍最大的特徵主導。PR 04 會把核方法與最近鄰與其他非線性分類器比較，學習式距離度量則在該章與後續章節討論。

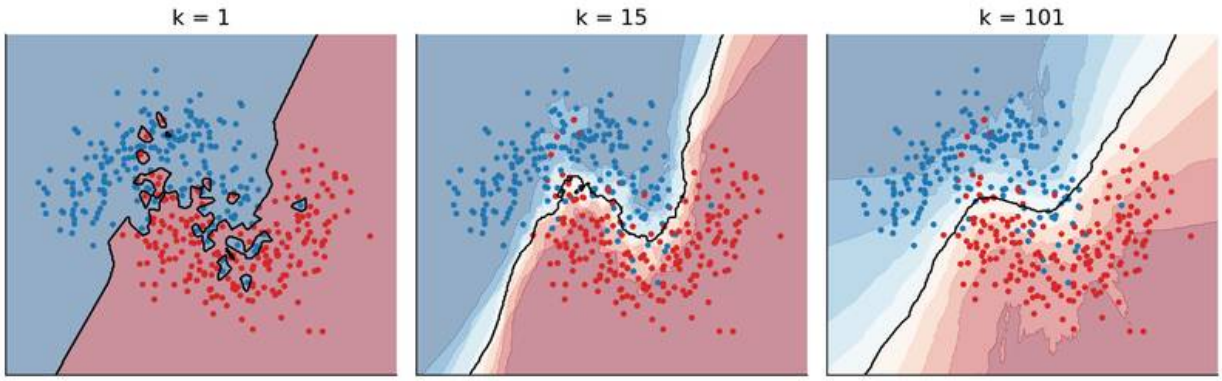


圖 *5* — 在 *400* 個帶雜訊的雙月形點上做 *k-NN*。色階表示投票比例 *k_c/k*；黑色曲線是 *0.5* 邊界。*k = 1* 過度擬合個別點，*k = 101* 開始把月形拉直。

程式碼：k-NN 與高斯分類器

```
import numpy as np
from sklearn.datasets import make_moons
from sklearn.model_selection import GridSearchCV, train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis

X, y = make_moons(1000, noise=0.3, random_state=0)
Xtr, Xte, ytr, yte = train_test_split(X, y, test_size=0.5, random_state=0)

search = GridSearchCV(KNeighborsClassifier(),
                      {"n_neighbors": [1, 3, 5, 9, 15, 31, 61, 101]}, cv=5)
search.fit(Xtr, ytr)                    # k chosen on training folds only
print("cross-validated k:", search.best_params_["n_neighbors"])
models = {"1-NN": KNeighborsClassifier(1),
          f"{search.best_params_['n_neighbors']}-NN": search.best_estimator_,
          "Gaussian naive Bayes": GaussianNB(),
          "QDA": QuadraticDiscriminantAnalysis()}
for name, m in models.items():
    m.fit(Xtr, ytr)
```

```
    print(f"{name:22s} test error {1 - m.score(Xte, yte):.3f}")
```

交叉驗證選出 $k = 31$，測試錯誤率 10.6%；1-NN 為 12.4%。高斯模型的類別形狀只是一個橢圓，跟不上月形，表現較差（樸素貝氏 17.4%，QDA 17.6%）。在這個二維問題上，非參數方法勝出，因為參數形式是錯的。下一節說明為何維度升高時這個優勢會縮小。

維度詛咒

白話版。局部方法需要「近」的鄰居。當許多座標帶有分散或無關的變動時，距離可能不再是好線索：即使只想蒐集一小部分資料，也得橫跨大半個空間，而最近點與最遠點的距離幾乎一樣。

精確版。「維度詛咒」（curse of dimensionality）這個說法涵蓋幾個相關的效應；Bishop [2, Section 1.4] 有幾何上的說明。其中三個很容易驗證（圖 6）：

1. **鄰域必須很大**。在單位立方體 $[0,1]^d$ 內均勻分布的資料中，包含比例 r 資料的子立方體，邊長為 $r^{1/d}$。要涵蓋 10% 的資料，邊長在二維是 0.32、十維是 0.79、一百維是 0.98。「局部」鄰域幾乎跨越每個特徵的全部範圍，估計就不再是局部的了。
2. **體積都在角落與外殼**。單位球體積除以外接立方體 $[-1,1]^d$ 的體積，會迅速趨近於零；而球的體積幾乎全集中在靠近表面的薄殼中。高維高斯的樣本與平均值的距離幾乎相同，而不是聚在平均值附近。
3. **距離集中**。 Beyer 等人 [13] 證明，在資料分布滿足廣泛條件下，最遠鄰與最近鄰距離的比值隨 d 增加趨近於一，「最近鄰」因而失去意義。在下面的模擬中，最近與最遠距離之比從二維的 0.02 升到一百維的約 0.7。

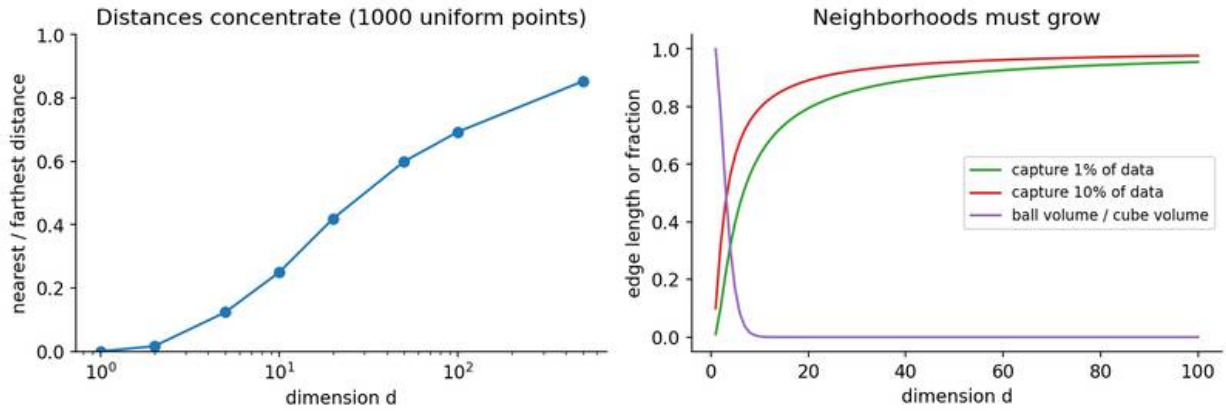


圖 *6* — 左：*1,000* 個均勻點與 *50* 個查詢點的距離集中現象。右：涵蓋均勻資料 *1%* 或 *10%* 所需的子立方體邊長，以及單位球佔立方體 *[-1, 1]^d* 的體積比例。

程式碼：距離集中

```
import numpy as np
```

```
rng = np.random.default_rng(0)
for d in [1, 2, 10, 100, 1000]:
    pts = rng.random((1000, d))          # 1000 uniform points in [0, 1]^d
    q = rng.random((50, d))              # 50 query points
    dist = np.sqrt(((q[:, None, :] - pts[None]) ** 2).sum(-1))
    ratio = (dist.min(1) / dist.max(1)).mean()
    edge = 0.1 ** (1 / d)                # sub-cube edge holding 10% of the volume
    print(f"d={d:5d}: nearest/farthest = {ratio:.3f}, "
          f"edge for 10% of data = {edge:.3f}")
```

最近／最遠距離比在二維是 0.017，十維 0.27，一百維 0.70，一千維 0.89。

那為什麼高維時還有方法行得通？真實資料並非均勻分布。影像、聲音與文字都貼近巨大環境空間中的低維結構（流形、群集），有資訊的方向遠少於原始座標數。因此實務上的防禦是：先降維或學習表徵（PR 05：特徵與維度）、限制模型（共用或對角共變異數、樸素貝氏），以及正則化。最近鄰方法今天之所以表現很好，正因為它們用在學習到的嵌入上，而不是原始像素上，如「現代觀點」所述。

現代觀點

自教科書寫成以來，密度估計分成兩股潮流：傳統統計學精進了 KDE 與共變異數估計；深度學習則建立了能擴展到影像的密度模型，而最近鄰搜尋成了基礎設施。

重新檢視核密度估計。 Chen 的教學文 [16] 是溫和的現代入門：收斂速率、頻寬選擇、信賴帶及其偏差處理，以及用 KDE 估計密度的幾何與拓撲特徵（峰值、等高集、脊線）的近期研究。Wang 與 Scott [17] 回顧為高維資料設計的非參數密度估計器，包括峰值分群（modal clustering）等應用，並列出開放的研究方向。Scott 的專書 [8] 與 Silverman 的經典 [7] 仍是理論的標準參考。

高維共變異數。本章的奇異共變異數問題本身就是一整個研究領域。Fan、Liao 與 Liu [18] 回顧大型共變異數矩陣與精度（反共變異數）矩陣的估計方法，著重於基於秩的方法與基於因子模型的方法；Ledoit–Wolf 收縮 [14] 至今仍是函式庫中的預設正則化方式。

深度密度模型。正規化流（normalizing flow）把簡單的基底分布送過一連串可逆、可微的映射，建構出有彈性的密度；變數變換公式則給出精確的對數似然。Papamakarios 等人 [19] 與 Kobyzev、Prince、Brubaker [20] 是兩篇標準綜述：前者強調設計原則、表達能力與計算取捨，後者著重流的建構方式與開放問題。自迴歸模型、VAE 與擴散模型是學習高維密度的其他方法；見 ML 07：生成模型與 ML 08：擴散與流匹配。Murphy 的兩冊教科書 [4] 以同一套記號涵蓋本章的傳統估計器與這些現代模型。

大規模最近鄰。 k-NN 如今無所不在，只是用在學習到的嵌入上，而非原始特徵。Johnson、Douze 與 Jégou [21] 示範如何在 GPU 上進行十億規模的相似度搜尋；Faiss 函式庫論文 [22] 描述向量搜尋的取捨空間（準確度、速度、記憶體）以及現代向量資料庫背後的索引方法。Papernot 與 McDaniel 的**深度 k-NN**（deep k-NN）[23] 在每一層的表徵中把測試輸入與它的訓練鄰居比較，以鄰居間標籤的不一致作為信心與穩健性的訊號。在分布外偵測上，Sun 等人 [24] 直接使用特徵空間中第 k 個最近鄰的距離，不做任何分布假設，並報告相對於參數式馬氏距離基準的大幅改進。那個參數式基準來自 Lee 等人 [25]，它在網路特徵上擬合共用共變異數的類別條件高斯，也就是 PR 01 的 LDA。本章參數式與非參數式之間的抉擇，正在深度特徵之上重演。

開放問題。高維中可靠的密度估計依然困難。Nalisnick 等人 [27] 發現，在某個影像資料集上訓練的流模型、VAE 與 PixelCNN，可能給另一個資料集更高的似然（例如在日常物體資料集上訓練的模型，給門牌號碼照片資料集更高的分數），所以單靠似然並不是安全的新穎性分數。嵌入空間中頻寬或 k 的選擇，以及非參數估計的校準不確定性，都仍是活躍的研究問題。

重點整理

- 生成式分類器需要 $\hat{p}(x \mid c_k)$ 與 $\hat{P}(c_k)$；估計器的假設決定了邊界的形狀。
- 高斯的 ML 估計是樣本平均與除以 N 的共變異數；樣本數 $N \le d$ 時共變異數奇異，需要收縮（例如 Ledoit–Wolf）或共用／對角結構。
- MAP 保留後驗眾數；完整貝氏預測對參數積分，資料稀少時比代入式預測更寬。共軛先驗有封閉解：高斯平均的精度相加，Beta–Bernoulli 比率的計數相加。
- 非參數估計都源自 $\hat{p} = k/(NV)$：直方圖與 Parzen ／ KDE 固定 V，k-NN 固定 k。頻寬或 k 比核的形狀重要；用交叉驗證或有根據的規則選擇，絕不用測試資料。
- k-NN 分類器估計 $P(c \mid x) \approx k_c/k$；在漸近意義下，1-NN 的錯誤率不超過貝氏錯誤率的兩倍。
- 高維時鄰域不再局部、距離會集中；局部方法是否有效取決於內在結構、特徵尺度與樣本數，不能只看環境維度。

練習

1. **計算參數數量**。計算一個 64 維高斯的平均值與共變異數參數數量。解釋為何用 30 筆樣本擬合會失敗，並提出兩種修正方式。

   提示

   64 個平均值加上 $64 \cdot 65/2 = 2{,}080$ 個共變異數元素。30 筆樣本時，$\hat{\Sigma}$ 的秩最多為 29，所以是奇異的。修正方式：往 τI 收縮（Ledoit–Wolf）、對角共變異數（樸素貝氏）、各類別共用共變異數，或先降維。

2. **Beta–Bernoulli 更新**。從 $\mathrm{Beta}(2,3)$ 開始，觀測到三次成功、兩次失敗，計算後驗、MAP 估計與成功的後驗預測機率。

   提示

   後驗 $\mathrm{Beta}(5,5)$。MAP（眾數）$= (5-1)/(5+5-2) = 0.5$。預測機率 $= 5/10 = 0.5$。與 ML 的 $3/5 = 0.6$ 比較。

3. **精度加權平均**。高斯平均的先驗為 $\mathcal{N}(0,1)$、雜訊變異數 $\sigma^2 = 4$，要幾筆樣本後資料權重才等於先驗權重？此時後驗標準差是多少？

   提示

   先驗精度為 1；每筆樣本增加 $1/4$。$N = 4$ 時權重相等。後驗精度為 2，所以 $s_N = 1/\sqrt{2} \approx 0.71$。

4. **重複的特徵**。在高斯樸素貝氏中複製一個特徵。預測準確率與後驗信心會怎麼變，再用 scikit-learn 在 `load_breast_cancer` 上驗證。

   提示

   多出的座標沒有新資訊，但分解後的似然把它算兩次，相當於把它的似然比平方。決策可能稍有改變；後驗會更靠近 0 或 1（更過度自信）。比較前後 `predict_proba` 的直方圖與對數損失。

5. **k-NN 密度不是密度**。證明在一維中，k-NN 估計 $\hat{p}_k(x)$ 在遠離資料處以 $1/|x|$ 衰減，並由此推論 $\int \hat{p}_k(x)\,dx$ 發散。為什麼這不影響 k-NN 分類器？

   提示

   遠離所有樣本時，到第 k 個鄰居的距離約為 $|x|$，所以 $V_k \approx 2|x|$、$\hat{p}_k \approx k/(2N|x|)$，其積分以對數方式發散。分類器只在同一點使用比例 k_c/k，共同的體積會消去。

6. **頻寬規則**。從 $\mathcal{N}(-2, 0.5^2)$ 與 $\mathcal{N}(2, 0.5^2)$ 的混合分布產生 500 筆樣本。比較 Silverman 法則、似然交叉驗證（對 `KernelDensity` 做 `GridSearchCV`），以及（若有安裝 `statsmodels`）代入式選擇器。哪一個過度平滑？為什麼？

   提示

   Silverman 法則使用整體分散程度（這裡約 2.1），它主要來自兩峰之間的間隔，因此給出的頻寬比每個峰所需的大好幾倍。交叉驗證似然選出的值接近單峰內的最佳值。

參考文獻

1. R. O. Duda, P. E. Hart and D. G. Stork, *Pattern Classification*, 2nd ed., Wiley, 2001 (Ch. 3, maximum-likelihood and Bayesian estimation; Ch. 4, non-parametric techniques). publisher page
2. C. M. Bishop, *Pattern Recognition and Machine Learning*, Springer, 2006 (Sections 1.4, 2.1–2.3, 2.5 and 4.2). official page and free PDF
3. S. Theodoridis and K. Koutroumbas, *Pattern Recognition*, 4th ed., Academic Press, 2008 (Ch. 2, density estimation and Bayesian classification). publisher page
4. K. P. Murphy, *Probabilistic Machine Learning: An Introduction*, MIT Press, 2022. official page
5. M. Rosenblatt, "Remarks on some nonparametric estimates of a density function," *The Annals of Mathematical Statistics*, vol. 27, no. 3, pp. 832–837, 1956. doi
6. E. Parzen, "On estimation of a probability density function and mode," *The Annals of Mathematical Statistics*, vol. 33, no. 3, pp. 1065–1076, 1962. doi
7. B. W. Silverman, *Density Estimation for Statistics and Data Analysis*, Chapman and Hall, 1986 (reissued by Routledge, 2018). doi
8. D. W. Scott, *Multivariate Density Estimation: Theory, Practice, and Visualization*, 2nd ed., Wiley, 2015. doi

9. D. W. Scott and G. R. Terrell, “Biased and unbiased cross-validation in density estimation,” *Journal of the American Statistical Association*, vol. 82, no. 400, pp. 1131–1146, 1987. doi
10. S. J. Sheather and M. C. Jones, “A reliable data-based bandwidth selection method for kernel density estimation,” *Journal of the Royal Statistical Society, Series B*, vol. 53, no. 3, pp. 683–690, 1991. doi
11. D. O. Loftsgaarden and C. P. Quesenberry, “A nonparametric estimate of a multivariate density function,” *The Annals of Mathematical Statistics*, vol. 36, no. 3, pp. 1049–1051, 1965. doi
12. T. M. Cover and P. E. Hart, “Nearest neighbor pattern classification,” *IEEE Transactions on Information Theory*, vol. 13, no. 1, pp. 21–27, 1967. doi
13. K. Beyer, J. Goldstein, R. Ramakrishnan and U. Shaft, “When is ‘nearest neighbor’ meaningful?,” in *Database Theory — ICDT’99*, Lecture Notes in Computer Science, Springer, 1999. doi
14. O. Ledoit and M. Wolf, “A well-conditioned estimator for large-dimensional covariance matrices,” *Journal of Multivariate Analysis*, vol. 88, no. 2, pp. 365–411, 2004. doi
15. A. Y. Ng and M. I. Jordan, “On discriminative vs. generative classifiers: a comparison of logistic regression and naive Bayes,” in *Advances in Neural Information Processing Systems* (*NIPS*), 2001. paper
16. Y.-C. Chen, “A tutorial on kernel density estimation and recent advances,” *Biostatistics & Epidemiology*, vol. 1, no. 1, pp. 161–187, 2017. doi
17. Z. Wang and D. W. Scott, “Nonparametric density estimation for high-dimensional data — algorithms and applications,” *WIREs Computational Statistics*, vol. 11, no. 4, 2019. doi
18. J. Fan, Y. Liao and H. Liu, “An overview of the estimation of large covariance and precision matrices,” *The Econometrics Journal*, vol. 19, no. 1, pp. C1–C32, 2016. doi
19. G. Papamakarios, E. Nalisnick, D. J. Rezende, S. Mohamed and B. Lakshminarayanan, “Normalizing flows for probabilistic modeling and inference,” *Journal of Machine Learning Research*, vol. 22, no. 57, pp. 1–64, 2021. arXiv
20. I. Kobyzev, S. J. D. Prince and M. A. Brubaker, “Normalizing flows: an introduction and review of current methods,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 43, no. 11, pp. 3964–3979, 2021. doi
21. J. Johnson, M. Douze and H. Jégou, “Billion-scale similarity search with GPUs,” *IEEE Transactions on Big Data*, vol. 7, no. 3, pp. 535–547, 2021. doi
22. M. Douze, A. Guzhva, C. Deng, J. Johnson, G. Szilvasy, P.-E. Mazaré, M. Lomeli, L. Hosseini and H. Jégou, “The Faiss library,” arXiv:2401.08281, 2024. arXiv
23. N. Papernot and P. McDaniel, “Deep k-nearest neighbors: towards confident, interpretable and robust deep learning,” arXiv:1803.04765, 2018. arXiv
24. Y. Sun, Y. Ming, X. Zhu and Y. Li, “Out-of-distribution detection with deep nearest neighbors,” in *Proc. International Conference on Machine Learning* (*ICML*), PMLR 162, 2022. paper
25. K. Lee, K. Lee, H. Lee and J. Shin, “A simple unified framework for detecting out-of-distribution samples and adversarial attacks,” in *Advances in Neural Information Processing Systems* (*NeurIPS*), 2018. paper
26. scikit-learn developers, “Density estimation,” scikit-learn user guide. documentation
27. E. Nalisnick, A. Matsukawa, Y. W. Teh, D. Gorur and B. Lakshminarayanan, “Do deep generative models know what they don’t know?,” in *Proc. International Conference on Learning Representations* (*ICLR*), 2019. arXiv

PR 03・迴歸與分類的線性模型

從基本原理推導線性判別：最小平方法、Fisher 判別、感知器、邏輯斯迴歸，以及最大間隔 SVM 與其對偶問題，並以損失函數統一它們。

網站版本: https://blog.shyandram.dev/zh-tw/learn/pattern-recognition/linear-models/

你將學到

- 看懂線性判別函數的幾何（權重向量、偏差、帶號距離），以及用兩類別分類器組成多類別分類器的陷阱。
- 推導最小平方法、ridge 迴歸及其貝氏詮釋，並理解為何最小平方法是脆弱的分類器。
- 推導 Fisher 線性判別，並連結到 PR 01 的高斯 LDA。
- 訓練感知器、證明它的收斂定理，並說明它為何可能停在任何一條分隔線上。
- 用牛頓法擬合邏輯斯迴歸，並解釋可分資料為何會把權重推向無限大。
- 推導最大間隔 SVM、它的 Lagrange 對偶問題與支援向量，並把這些方法都看成同一個線性分數上的不同損失。

先看全貌

線性模型是每門圖形識別課程中第一個真正的分類器，也一直是面對新表徵時第一個該試的方法。如果一條直線就能在特徵空間中分開兩個類別，大型非線性分類器可能只增加成本，而不增加有用的準確率。如果線性基準失敗了，它的錯誤能幫助我們找出缺少的特徵或非線性結構。在現代深度學習中，幾乎每個分類器的最後一層都是學習特徵上的線性模型，而「線性分類器在這些特徵上表現如何？」正是評估表徵的標準測試。

> **用最簡單的話說**
>
> 每個量測值都有一票。把每個量測值乘上它的投票力道，全部加起來，再和一個門檻比較：高於門檻就是「是」，低於就是「否」。本章所有方法用的都是這台投票機，差別只在**怎麼決定投票力道**：讓總分盡量接近正確答案（最小平方法）、把兩群拉開（Fisher）、一次修正一個錯誤（感知器）、預測誠實的機率（邏輯斯迴歸），或在兩群之間留下最寬的空隙（SVM）。

線性分數是 $f(x) = w^\top \phi(x) + b$。**特徵映射** ϕ 可以是原始強度、平方值、梯度，或凍結的神經網路嵌入。「線性」指的是參數 w 與 b，不一定是原始量測 x：取 $\phi(x) = (x, x^2)$ 時，線性模型在 x 上畫出的是彎曲的邊界。本章依循 Duda、Hart 與 Stork [1, Ch. 5]、Bishop [2, Ch. 3–4, 7] 以及 Theodoridis 與 Koutroumbas [3, Ch. 3]，統計觀點參考 Hastie 等人 [4]。讓 ϕ 變成隱式且非常豐富的核方法，是 PR 04 的主題。

線性判別函數

白話版。兩類別線性分類器是穿過特徵空間的一刀平切：二維中是直線，三維中是平面，一般情形是超平面。權重向量決定這一刀面向哪裡，偏差決定它切在哪裡。

精確版。兩類別時，若 $g(x) = w^\top x + b > 0$ 判為 c_1，否則判為 c_0。決策邊界 $g(x) = 0$ 是超平面 H。由線性分類器的幾何可得三個事實 [1, 2, 3]：

- w 是 H 的**法向量**：對 H 上任兩點 x_a, x_b，$w^\top(x_a - x_b) = 0$。
- 任一點 x 到 H 的**帶號距離**為 $r = g(x)/\|w\|$。所以 $g(x)$ 正比於 x 離邊界多遠、在哪一側。
- 當 $w \neq 0$，原點到 H 的一般距離是 $|b|/\|w\|$；依上面的符號慣例，有號距離為 $b/\|w\|$，而 H 上離原點最近的點是 $-bw/\|w\|^2$。偏差只平移平面，不旋轉它。

把輸入**擴增**一個常數來吸收偏差很方便：$\tilde{x} = (x, 1)$、$\tilde{w} = (w, b)$，於是 $g(x) = \tilde{w}^\top \tilde{x}$。程式中我們默默這麼做。標籤取 $y \in \{-1, +1\}$ 時，樣本被正確分類的充要條件是它的**函數間隔**（functional margin）$y\,g(x)$ 為正。下面每個方法都能用這個量來表示。

超過兩個類別。兩類別分類器有兩種直覺的組合方式：一**對其餘**（one-versus-rest，K 個分類器，各自把一類和其他所有類分開），以及一**對一**（one-versus-one，每對類別一個，共 $K(K-1)/2$ 個，再投票）。兩者都可能在輸入空間留下投票結果模糊的區域 [2, Section 4.1.2]。乾淨的解法是**線性機**（linear machine）：每個類別一個判別函數 $g_k(x) = w_k^\top x + b_k$，把 x 判給分數最大的類別。它的決策區域一定是凸且單連通的，因為若 x_a 與 x_b 都是 g_k 最大，兩點之間每一點也是如此（每個 g_j 沿線段都是線性的）。PR 01 的高斯 LDA、多類別邏輯斯迴歸，以及神經網路的最後一層，都是線性機。

最小平方法與 ridge 迴歸

白話版。要預測一個數值，就選讓預測誤差平方最小的權重。如果權重比資料能支撐的多，再加上對大權重的懲罰。

精確版。令**設計矩陣** $\Phi \in \mathbb{R}^{N \times M}$ 的第 n 列是特徵向量 $\phi(x_n)^\top$（含截距用的常數座標），$t \in \mathbb{R}^N$ 是目標值。最小平方法最小化

$$\mathcal{L}(w) = \frac{1}{2}\|\Phi w - t\|_2^2,$$

其梯度 $\Phi^\top(\Phi w - t)$ 在**正規方程** $\Phi^\top \Phi w = \Phi^\top t$ 處為零。對滿欄秩設計矩陣，形成 $\Phi^\top\Phi$ 就會使 2-範數條件數平方，而顯式求逆更沒有必要；基於 QR 或 SVD 的解法（`np.linalg.lstsq`）處理近乎相依的欄位更可靠。若假設目標值帶有獨立、同變異數的高斯雜訊，此目標函數與負對數似然只差一個正比例因子及加法常數，因此最小平方法就是最大似然估計（PR 02）。雜訊假設說明了統計詮釋，但不代表平方誤差適合所有目標。

Ridge 迴歸 [8] 加上懲罰 $\frac{\lambda}{2}\|w\|^2$，得到

$$(\Phi^\top\Phi + \lambda I)\, w = \Phi^\top t.$$

只要 $\lambda > 0$，左邊的矩陣一定可逆，即使 $M > N$ 也是。截距通常不納入懲罰。特徵縮放很重要，因為同樣的懲罰對不同單位的座標效果不同；縮放統計量只能在訓練資料上擬合。

貝氏連結

給權重高斯先驗 $w \sim \mathcal{N}(0, \alpha^{-1} I)$（精度 α）與精度為 β 的高斯雜訊，權重的後驗是高斯分布，其共變異數與平均值為 [2, Section 3.3]

$$S = (\alpha I + \beta \Phi^\top \Phi)^{-1}, \qquad m = \beta S \Phi^\top t.$$

對新的特徵向量 ϕ_*，預測分布為

$$p(t_* \mid x_*, D) = \mathcal{N}(t_* \mid m^\top \phi_*,\ \beta^{-1} + \phi_*^\top S \phi_*).$$

第一個變異數項是觀測雜訊；第二項表達對權重的不確定性，在訓練資料很少涵蓋的特徵方向上會比較大。後驗平均 m 等於 $\lambda = \alpha/\beta$ 的 ridge 解：ridge 是 MAP 估計，和 PR 02 的高斯平均是同一個故事。

用最小平方法做分類

白話版。可以把類別標籤當成數字（+1 與 −1，或 one-hot 向量），再用最小平方法擬合。它很快、有封閉解，但工具不對：它會懲罰「太正確」的點。

精確版。對 K 個類別，把目標編碼成 **one-hot** 向量 $t_n \in \{0,1\}^K$，用最小平方法擬合線性機 $g(x) = W^\top \tilde{x}$，$W = (\tilde{X}^\top \tilde{X})^{-1} \tilde{X}^\top T$，再依最大輸出分類。兩類別時若選用適當的目標值（類別 1 為 N/N_1，類別 0 為 $-N/N_0$），最小平方的方向正好是下一節推導的 Fisher 方向 [2, Section 4.1.5]，所以這個方法並非毫無道理。

它有兩個眾所周知的失敗模式：

1. **對遠在正確一側的點很敏感**。對目標 +1 而言，平方誤差懲罰 $g(x) = 5$ 和懲罰 $g(x) = -3$ 一樣重。圖 1 加入二十個已被正確分類、但位置很遠的紅點；最小平方法為了減少它們的「誤差」而旋轉邊界，開始把藍點分錯。邏輯斯迴歸幾乎不動。
2. **三類以上時的遮蔽（masking）**。當類別大致排成一直線時，中間類別的輸出可能處處被鄰居壓過，於是中間類別永遠不會被預測出來 [4, Section 4.2]。

從機率的角度看，最小平方法背後的高斯雜訊模型對二元目標根本是錯的。解法是使用為分類設計的損失，本章其餘部分就在做這件事。

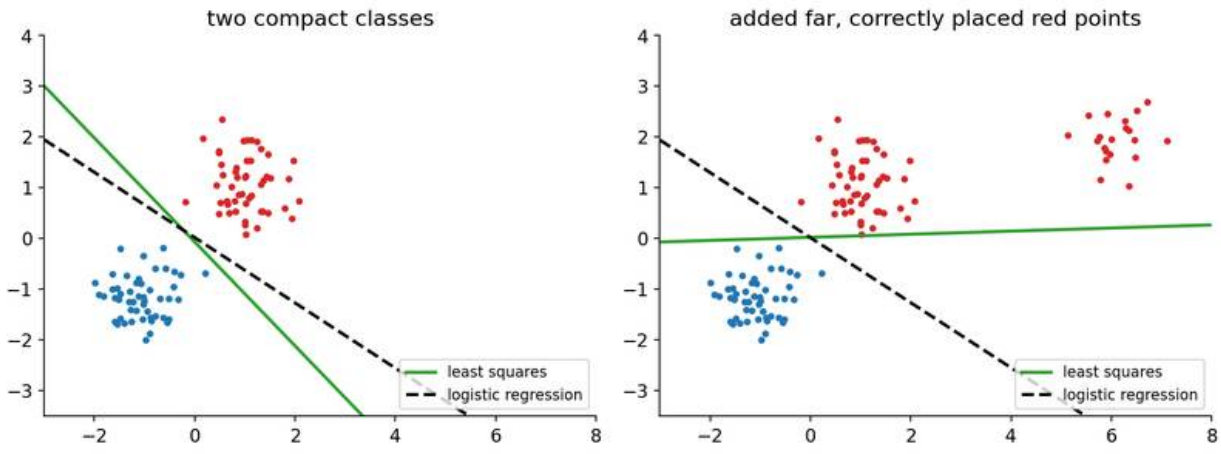


圖 1 — 最小平方法與邏輯斯迴歸。加入早已被正確分類的紅點（右）會把最小平方邊界（綠線）拉進藍色類別；邏輯斯迴歸（虛線）不受影響。

Fisher 線性判別

白話版。把所有點投影到一條線上，看兩個類別的一維直方圖。好的線會讓兩類平均值離得遠，而且每類都很窄。Fisher 方向就是讓這個比值最大的那條線。

精確版。 Fisher [5] 提議選擇投影 $z = w^\top x$，使投影後類別平均值的分離程度相對於各類內部的分散最大。令 $m_k = \frac{1}{N_k}\sum_{n\in c_k} x_n$ 為類別平均。投影後的平均是 $w^\top m_k$，類別 k 投影後的類內散布為 $\sum_{n\in c_k}(w^\top x_n - w^\top m_k)^2 = w^\top S_k w$，其中 $S_k = \sum_{n\in c_k}(x_n - m_k)(x_n - m_k)^\top$。**Fisher 準則**為

$$J(w) = \frac{(w^\top m_1 - w^\top m_0)^2}{w^\top S_0 w + w^\top S_1 w} = \frac{w^\top S_B w}{w^\top S_W w},$$

其中

$$S_B = (m_1 - m_0)(m_1 - m_0)^\top, \qquad S_W = S_0 + S_1$$

分別是**類間**與**類內散布矩陣**。J 與 w 的長度無關，只與方向有關。

推導。令比值的梯度為零，得到

$$(w^\top S_B w)\, S_W w = (w^\top S_W w)\, S_B w.$$

括號中的兩項都是純量。此外 $S_B w = (m_1 - m_0)\,[(m_1 - m_0)^\top w]$ 永遠指向 $m_1 - m_0$。因此 $S_W w \propto m_1 - m_0$，又因為只有方向重要，

$$w^\star \propto S_W^{-1}(m_1 - m_0).$$

若 S_W 是單位矩陣的倍數（等向、不相關的類別），Fisher 方向就是平均值差。否則 S_W^{-1} 會把它從類別分散較大的方向轉開。圖 2 顯示兩個細長、相關的類別的效果：投影到平均值差上，直方圖嚴重重疊（$J = 0.70$）；投影到 Fisher 方向，兩者分得很開（$J = 3.18$）。

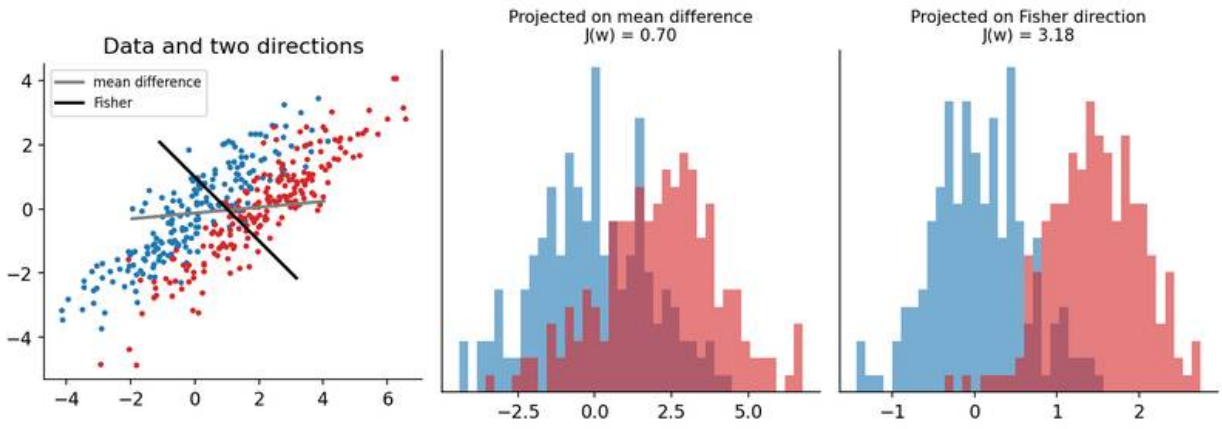


圖 2 — 兩個相關的類別分別投影到平均值差（中，$J = 0.70$）與 *Fisher* 方向（右，$J = 3.18$）。*S_W* 使用各類別的共變異數；類別大小相同時這只會縮放 J。

與高斯 LDA 的關係。 Fisher 準則不做任何分布假設；它是表徵的準則。但當 S_W 正比於匯集的類內共變異數時，它的方向與 PR 01 的高斯 LDA 方向 $\Sigma^{-1}(\mu_1 - \mu_0)$ 相同。兩者只差在門檻：Fisher 不指定投影軸上的切點，高斯 LDA 則由類別平均與先驗決定切點。所以兩者都稱為「LDA」。（在主題模型中，LDA 也指 latent Dirichlet allocation，與此無關。）Fisher 投影本身不提供校準過的機率。

超過兩個類別。 K 個類別、整體平均為 m 時，定義 $S_W = \sum_k S_k$ 與 $S_B = \sum_k N_k (m_k - m)(m_k - m)^\top$。使多類別準則最大的投影 $W \in \mathbb{R}^{d \times (K-1)}$，由 $S_W^{-1} S_B$ 的前幾個特徵向量張成 $[1, 2]$。因為 S_B 是受整體平均約束的 K 個秩一項之和，秩最多為 $K - 1$，所以 Fisher 最多只能產生 $K - 1$ 個有用的特徵。這就是 PR 05 中的監督式降維。

程式碼：Fisher 方向

```
import numpy as np
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis

rng = np.random.default_rng(2)
cov = np.array([[3.0, 2.4], [2.4, 2.4]])
X0 = rng.multivariate_normal([0, 0], cov, 200)
X1 = rng.multivariate_normal([2, 0], cov, 200)
m0, m1 = X0.mean(0), X1.mean(0)
S_W = np.cov(X0.T) + np.cov(X1.T)       # within-class scatter (per-class covariances)
w = np.linalg.solve(S_W, m1 - m0)                           # Fisher direction

def J(v):                          # Fisher criterion for two classes
    return (v @ (m1 - m0)) ** 2 / (v @ S_W @ v)

lda = LinearDiscriminantAnalysis().fit(np.vstack([X0, X1]), np.repeat([0, 1], 200))
cos = w @ lda.coef_[0] / np.linalg.norm(w) / np.linalg.norm(lda.coef_[0])
angles = np.linspace(0, np.pi, 3601)
best = max(J(np.array([np.cos(a), np.sin(a)])) for a in angles)
print(f"J(mean difference) = {J(m1 - m0):.4f}, J(Fisher) = {J(w):.4f}, "
      f"grid maximum = {best:.4f}")
```

```
print(f"cosine between Fisher direction and LDA coef_: {cos:.6f}")
```

封閉解達到網格搜尋的最大值（$J = 3.1841$），超過平均值差（0.6953）的四倍；它的方向與 scikit-learn 高斯 LDA 的係數在小數點後六位一致（餘弦 1.000000）。

感知器

白話版。從任意一條線開始。逐一檢查訓練點；每當有點落在錯誤的一側，就把線往能正確分類它的方向推一點。如果類別可以被一條線分開，這個過程保證會停下來。

精確版。 Rosenblatt 的感知器（perceptron）[6] 使用標籤 $y_n \in \{-1, +1\}$ 與擴增特徵 $\tilde{x}_n$。它最小化**感知器準則**

$$J_P(w) = -\sum_{n \in \mathcal{M}} y_n \, w^\top \tilde{x}_n,$$

其中 $\mathcal{M}$ 是被分錯的樣本集合（$y_n w^\top \tilde{x}_n \le 0$ 者）。每一項都非負，且 $J_P = 0$ 的充要條件是所有樣本都被正確分類。每次對一個分錯的樣本做隨機梯度下降，得到更新規則

$$w \leftarrow w + \eta \, y_n \tilde{x}_n.$$

以零初始化 w 時，學習率 $\eta > 0$ 只會縮放 w，不會改變決策序列，所以常取 $\eta = 1$。IP 13 的感知器一節以影像圖樣的語言介紹同一條規則，並連結到多層網路。

收斂定理。假設資料**以間隔** $\gamma > 0$ **線性可分**：存在單位向量 u，使所有 n 都滿足 $y_n u^\top \tilde{x}_n \ge \gamma$。令 $R = \max_n \|\tilde{x}_n\|$。那麼感知器（從 $w_0 = 0$ 開始）最多更新 $(R/\gamma)^2$ 次。證明只要三行。經過 t 次更新後：

- 每次更新讓 $w^\top u$ 增加 $y_n \tilde{x}_n^\top u \ge \gamma$，所以 $w_t^\top u \ge t\gamma$；
- 每次更新都發生在錯誤上，此時 $y_n w^\top \tilde{x}_n \le 0$，所以 $\|w_t\|^2 = \|w_{t-1}\|^2 + 2 y_n w_{t-1}^\top \tilde{x}_n + \|\tilde{x}_n\|^2 \le \|w_{t-1}\|^2 + R^2$，因此 $\|w_t\|^2 \le t R^2$；
- 由 Cauchy–Schwarz 不等式，$t\gamma \le w_t^\top u \le \|w_t\| \le \sqrt{t}\, R$，所以 $t \le R^2/\gamma^2$。

這個上界與樣本數和維度都無關，只取決於類別之間的空隙相對於資料大小有多「寬」。

限制。如果類別重疊，感知器永遠不會停止，權重會一直循環（實務上常見的修正是保留目前看過最好的權重）。即使收斂了，它也停在第一條碰到的分隔超平面上，而這取決於初始化與資料順序（圖 3 左）。它的分數也不是機率。從損失的角度看，感知器使用 $\max(0, -y\,g(x))$，任何正確分類（無論多勉強）的損失都是零。邏輯斯迴歸與 SVM 都要求比「剛好正確」更多，藉此修正這一點。

程式碼：感知器更新次數與上界

```
import numpy as np
from sklearn.svm import SVC
```

```
rng = np.random.default_rng(1)
X = np.vstack([rng.normal([-1.0, -0.7], 0.6, (30, 2)),
               rng.normal([1.0, 0.7], 0.6, (30, 2))])
y = np.r_[-np.ones(30), np.ones(30)]
Phi = np.c_[X, np.ones(len(X))]          # augmented features (bias as last weight)

def perceptron(Phi, y, rng, max_epochs=1000):
    w, updates = np.zeros(Phi.shape[1]), 0
    for epoch in range(max_epochs):
        errors = 0
        for i in rng.permutation(len(y)):
            if y[i] * (w @ Phi[i]) <= 0:  # misclassified or on the boundary
                w += y[i] * Phi[i]        # learning rate 1 (scale is irrelevant)
                errors += 1
        updates += errors
        if errors == 0:
            return w, updates
    return w, updates

runs = [perceptron(Phi, y, np.random.default_rng(s)) for s in range(200)]
updates = np.array([u for _, u in runs])
print(f"updates over 200 random orders: mean {updates.mean():.1f}, "
      f"max {updates.max()}")
print("all runs separate the data:",
      all(np.all(np.sign(Phi @ w) == y) for w, _ in runs))

# convergence bound (R / gamma)^2, with gamma from the max-margin separator
svm = SVC(kernel="linear", C=1e6).fit(X, y)
u = np.r_[svm.coef_[0], svm.intercept_[0]]
gamma = np.min(y * (Phi @ u)) / np.linalg.norm(u)
R = np.max(np.linalg.norm(Phi, axis=1))
print(f"bound (R/gamma)^2 = {(R / gamma) ** 2:.1f}")
```

在 200 種隨機呈現順序下，感知器平均需要 2.7 次更新（最多 10 次），且每次執行都能分開資料。最壞情況上界是 691：它是保證，不是預測。上界需要一個有效的間隔 γ；我們從硬間隔 SVM（下方 SVM 一節的主題）取得它。由於 SVM 不懲罰偏差，擴增空間中真正的最大間隔可能稍大一些，會給出稍緊的上界。

邏輯斯迴歸

白話版。不要硬性的是／否，而是把線性分數通過一條 S 形曲線壓成機率。擬合權重，讓真實標籤被賦予的機率盡可能高。

精確版。標籤取 $y \in \{0, 1\}$，二元邏輯斯迴歸 [7] 的模型為

$$P(y = 1 \mid x) = \sigma(w^\top \tilde{x}), \qquad \sigma(z) = \frac{1}{1 + e^{-z}}.$$

分數 $w^\top \tilde{x}$ 是**對數勝算** $\ln \frac{P(y=1|x)}{P(y=0|x)}$。這個形式並非隨意選的。PR 01 證明了共用共變異數的高斯類別，其後驗正好是 $\sigma(w^\top x + w_0)$，其中 $w = \Sigma^{-1}(\mu_1 - \mu_0)$；許多其他類別條件分布族也是如

此。邏輯斯迴歸保留這個後驗的函數形式，但直接擬合 w，只有 $d+1$ 個參數，而非高斯模型的 $2d + d(d+1)/2 + 1$ 個。Ng 與 Jordan [18] 分析了這種生成式與判別式之間的取捨（PR 02）。

訓練。 Bernoulli 模型的負對數似然是**交叉熵**

$$E(w) = -\sum_{n=1}^{N} \big[y_n \ln p_n + (1 - y_n) \ln(1 - p_n) \big], \qquad p_n = \sigma(w^\top \tilde{x}_n).$$

因為 $\sigma'(z) = \sigma(z)(1 - \sigma(z))$，它的梯度與 Hessian 矩陣出奇簡單 [2, Section 4.3]：

$$\nabla E = \sum_n (p_n - y_n)\, \tilde{x}_n = \tilde{X}^\top (p - y), \qquad H = \tilde{X}^\top R\, \tilde{X}, \quad R = \operatorname{diag}(p_n(1 - p_n)).$$

梯度和最小平方法一樣是「預測減目標，乘上輸入」的形式。H 是半正定的，所以 E 是凸函數，沒有虛假的局部最小值。牛頓法 $w \leftarrow w - H^{-1}\nabla E$ 可以改寫成每次迭代權重 R 都會改變的加權最小平方問題：**迭代重新加權最小平方法**（iteratively reweighted least squares, IRLS）。程式中請從 logit 以穩定的公式計算損失，例如 $\ln(1 + e^z) =$ `np.logaddexp(0, z)`，而不是對捨入後的機率取對數。

可分資料。如果有超平面能分開訓練類別，把 w 放大會讓每個 p_n 更接近它的標籤，所以 E 一直下降趨近於零，未正則化的解並不存在：$\|w\| \to \infty$。實務上請一定要正則化，例如加上 $\frac{\lambda}{2}\|w\|^2$（scikit-learn 的 `C` 就是 $1/\lambda$）。值得注意的是，權重如何發散是明確定義的：在可分資料上，單純的梯度下降會在方向上收斂到最大間隔 SVM 的解 [19]，「現代觀點」會再回到這一點。

多類別。把 sigmoid 換成 **softmax** $P(y = k \mid x) = \exp(w_k^\top \tilde{x}) / \sum_j \exp(w_j^\top \tilde{x})$，交叉熵換成 K 類別版本。這是有機率輸出的線性機，也正是標準神經網路分類器的最後一層加損失函數。在標籤雜訊或分布偏移下，機率輸出不會自動保持校準（PR 01）。

程式碼：牛頓法（IRLS）與可分資料

```
import numpy as np
from sklearn.linear_model import LogisticRegression

rng = np.random.default_rng(0)
X = np.vstack([rng.normal(-1, 1, (100, 2)), rng.normal(1, 1, (100, 2))])
y = np.r_[np.zeros(100), np.ones(100)]
Phi = np.c_[np.ones(len(X)), X]

def nll(w, lam):
    z = Phi @ w
    # log(1 + e^z) - y z, written stably with logaddexp
    return np.sum(np.logaddexp(0, z) - y * z) + 0.5 * lam * np.sum(w[1:] ** 2)

# Newton's method (iteratively reweighted least squares)
lam, w = 1.0, np.zeros(3)
for it in range(20):
    p = 1 / (1 + np.exp(-(Phi @ w)))
    grad = Phi.T @ (p - y) + lam * np.r_[0, w[1:]]
    H = Phi.T @ (Phi * (p * (1 - p))[:, None]) + lam * np.diag([0, 1, 1])
    step = np.linalg.solve(H, grad)
    w -= step
```

```
    if np.linalg.norm(step) < 1e-10:
        break
sk = LogisticRegression(C=1 / lam, tol=1e-10, max_iter=1000).fit(X, y)
print(f"Newton converged in {it + 1} iterations, loss {nll(w, lam):.4f}")
print("Newton:      ", np.round(w, 4))
print("scikit-learn:", np.round(np.r_[sk.intercept_, sk.coef_[0]], 4))
```

牛頓法在 8 次迭代內收斂，得到與 scikit-learn 相同的權重（截距、w_1、w_2）$=(0.0777, 1.7205, 1.5300)$。懲罰不包含截距，與 scikit-learn 的慣例一致。

下一段程式在可被一條通過原點的直線分開的資料上，對未正則化的邏輯斯損失執行單純的梯度下降，並追蹤權重範數，以及 w 與最大間隔方向之間的餘弦。

```
import numpy as np
from sklearn.svm import LinearSVC

rng = np.random.default_rng(1)
X = np.vstack([rng.normal([2.0, 1.0], 0.6, (40, 2)),
               rng.normal([-2.0, -1.0], 0.6, (40, 2))])
y = np.r_[np.ones(40), -np.ones(40)]              # separable through the origin

# max-margin direction without bias (hard margin approximated by large C)
svm = LinearSVC(fit_intercept=False, loss="hinge", C=1e4, max_iter=200000).fit(X, y)
u = svm.coef_[0] / np.linalg.norm(svm.coef_[0])

w, lr = np.zeros(2), 0.1
for t in range(1, 100001):
    m = y * (X @ w)
    grad = -(X * (y / (1 + np.exp(m)))[:, None]).mean(0)   # logistic loss gradient
    w -= lr * grad
    if t in (10, 100, 1000, 10000, 100000):
        print(f"step {t:6d}: ||w|| = {np.linalg.norm(w):6.2f}, "
              f"cos(w, SVM) = {w @ u / np.linalg.norm(w):.5f}")
```

範數無上限地增長，但很慢（10 步後 0.68，10^5 步後 8.64），而與 SVM 方向的餘弦從 0.87 緩慢爬升到 0.98。這正是 Soudry 等人 [19] 預測的、以對數速度收斂到最大間隔方向的現象。

支援向量機：最大間隔

白話版。在所有能分開類別的直線中，選周圍那條空「馬路」最寬的一條。只有站在馬路邊緣的點才重要；其他點只要不越過馬路，怎麼移動答案都不變。

精確版。使用標籤 $y_n \in \{-1, +1\}$ 與分數 $f(x) = w^\top x + b$（這裡偏差分開寫）。對分隔超平面而言，x_n 到邊界的距離是 $y_n f(x_n)/\|w\|$。因為 (w, b) 與 (cw, cb) 定義同一個平面，我們固定尺度，使最近的點滿足 $y_n f(x_n) = 1$（**標準形式**）。於是每一側的間隔是 $1/\|w\|$，馬路寬度是 $2/\|w\|$。最大化它等價於凸二次規劃 [9, 10]

$$\min_{w,b} \ \frac{1}{2}\|w\|^2 \quad \text{subject to} \quad y_n(w^\top x_n + b) \geq 1, \quad n = 1, \dots, N.$$

Lagrange 函數與對偶。為每個限制式引入乘數 $a_n \ge 0$：

$$\mathcal{L}(w, b, a) = \frac{1}{2}\|w\|^2 - \sum_n a_n [y_n (w^\top x_n + b) - 1].$$

令 $\partial \mathcal{L}/\partial w = 0$、$\partial \mathcal{L}/\partial b = 0$，得到

$$w = \sum_n a_n y_n x_n, \qquad \sum_n a_n y_n = 0.$$

代回後消去 w 與 b，剩下**對偶問題**

$$\max_a \sum_n a_n - \frac{1}{2}\sum_n \sum_m a_n a_m y_n y_m x_n^\top x_m \quad \text{subject to} \quad a_n \ge 0,\ \sum_n a_n y_n = 0.$$

有三個重要的推論 [2, 11]：

- **支援向量**。 Karush–Kuhn–Tucker（KKT）條件包含**互補鬆弛性** $a_n[y_n f(x_n) - 1] = 0$。所以只有恰好落在間隔上的點才有 $a_n > 0$。這些點就是**支援向量**（support vector）；其他點的 $a_n = 0$，拿掉也不會改變解。偏差可由任一支援向量 s 求得：$b = y_s - w^\top x_s$（為了數值穩定，對所有支援向量取平均）。
- **預測的稀疏性**。$f(x) = \sum_{n \in \mathcal{S}} a_n y_n \, x_n^\top x + b$ 只取決於支援向量集合 $\mathcal{S}$。
- **只有內積**。資料在對偶問題與預測中只以內積 $x_n^\top x_m$ 出現。把內積換成核函數 $k(x_n, x_m)$，就得到非線性 SVM，而完全不必計算 ϕ。這就是**核技巧**，在 PR 04 中發展。

軟間隔與鉸鏈損失。真實類別會重疊，所以限制式不可能全部成立。Cortes 與 Vapnik [10] 加入**鬆弛變數** $\xi_n \ge 0$、限制式 $y_n f(x_n) \ge 1 - \xi_n$ 與懲罰 $C \sum_n \xi_n$。在最佳解處 $\xi_n = \max(0, 1 - y_n f(x_n))$，所以軟間隔 SVM 等價於無限制問題

$$\min_{w,b} \ \frac{1}{2}\|w\|^2 + C \sum_n \max(0,\ 1 - y_n f(x_n)).$$

第二項是**鉸鏈損失**（hinge loss）：超出間隔的點為零，在間隔內與錯誤一側則線性增加。對偶問題中唯一的改變是盒狀限制 $0 \le a_n \le C$。所以線性 SVM 就是「ridge 正則化的鉸鏈損失最小化」，可以直接和正則化的邏輯斯迴歸比較。C、核頻寬與特徵縮放的作用，以及 SVM 分數不是機率這件事，都在 PR 04 討論。

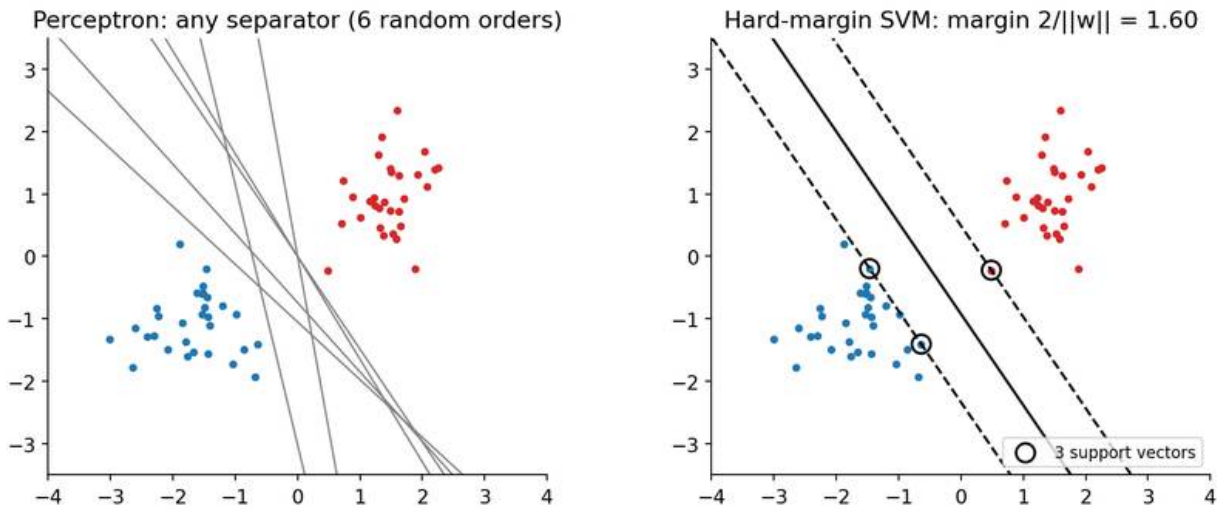


圖 3 — 左：感知器停在它最先碰到的分隔線上（六種隨機順序）。右：硬間隔 *SVM* 選出間隔最寬的唯一直線；只有三個被圈起的支援向量決定它。

程式碼：實際驗證對偶關係

這段檢查擬合一個（近乎）硬間隔的線性 SVM，並以數值驗證對偶關係：$w = \sum_n a_n y_n x_n$、$\sum_n a_n y_n = 0$，以及支援向量上 $y_n f(x_n) = 1$。

```
import numpy as np
from sklearn.svm import SVC

rng = np.random.default_rng(7)
X = np.vstack([rng.normal([-1.5, -1.0], 0.6, (30, 2)),
               rng.normal([1.5, 1.0], 0.6, (30, 2))])
y = np.r_[-np.ones(30), np.ones(30)]

svm = SVC(kernel="linear", C=1e6).fit(X, y)          # huge C ~ hard margin
alpha_y = svm.dual_coef_[0]                          # alpha_n * y_n for SVs
w_dual = alpha_y @ svm.support_vectors_              # w = sum_n alpha_n y_n x_n
f = X @ svm.coef_[0] + svm.intercept_[0]

print("support vectors:", len(svm.support_), "of", len(y))
print("w from dual:", np.round(w_dual, 4), " coef_:", np.round(svm.coef_[0], 4))
print("sum_n alpha_n y_n =", round(float(alpha_y.sum()), 8))
print("y f(x) on support vectors:", np.round(y[svm.support_] * f[svm.support_], 4))
print("smallest y f(x) over all points:", round(float(np.min(y * f)), 4))
print("margin width 2/||w|| =", round(2 / np.linalg.norm(svm.coef_[0]), 4))
```

60 個點中只有 3 個是支援向量（圖 3 中被圈起的點）。由對偶係數重建的權重向量與 `coef_` 完全一致，對偶係數總和為零，支援向量落在 $yf(x) \approx 1$（在求解器容許誤差內），沒有點落在間隔之內，間隔寬度為 1.60。scikit-learn 的 `SVC` 包裝了 LIBSVM [16]；對大型稀疏問題，`LinearS`

VC 與 `LogisticRegression(solver="liblinear")` 使用 LIBLINEAR [14]，而 Pegasos [15] 這類隨機次梯度求解器可擴展到非常大的資料集。

一張圖看懂：線性分類器就是損失函數

白話版。本章所有方法都用同一個線性函數為點打分數。差別在於對每個間隔值 $y\,g(x)$ 收多少費用：只要正確就不收（感知器）、安全地正確才不收（SVM）、平滑遞減的費用（邏輯斯），或以恰好 1 為中心的拋物線（最小平方）。

精確版。標籤取 $y \in \{-1, +1\}$、間隔 $m = y\,g(x)$，每個方法都在最小化某個**代理損失**（surrogate loss）$\varphi(m)$ 的（可能正則化的）平均值，用來代替非凸的零一損失 $\mathbb{1}[m \le 0]$（圖 4）：

方法	損失 $\varphi(m)$	行為
感知器	$\max(0, -m)$	任何正確的點都為零；與尺度無關，所以 $w = 0$ 是平凡的最小解
鉸鏈（SVM）	$\max(0, 1 - m)$	超出間隔即為零；解是稀疏的（支援向量）
邏輯斯	$\ln(1 + e^{-m})$	平滑、永不為零；能給出機率
平方	$(1 - m)^2 = (y - g(x))^2$	懲罰 $m > 1$：「太正確」的點會拉動邊界

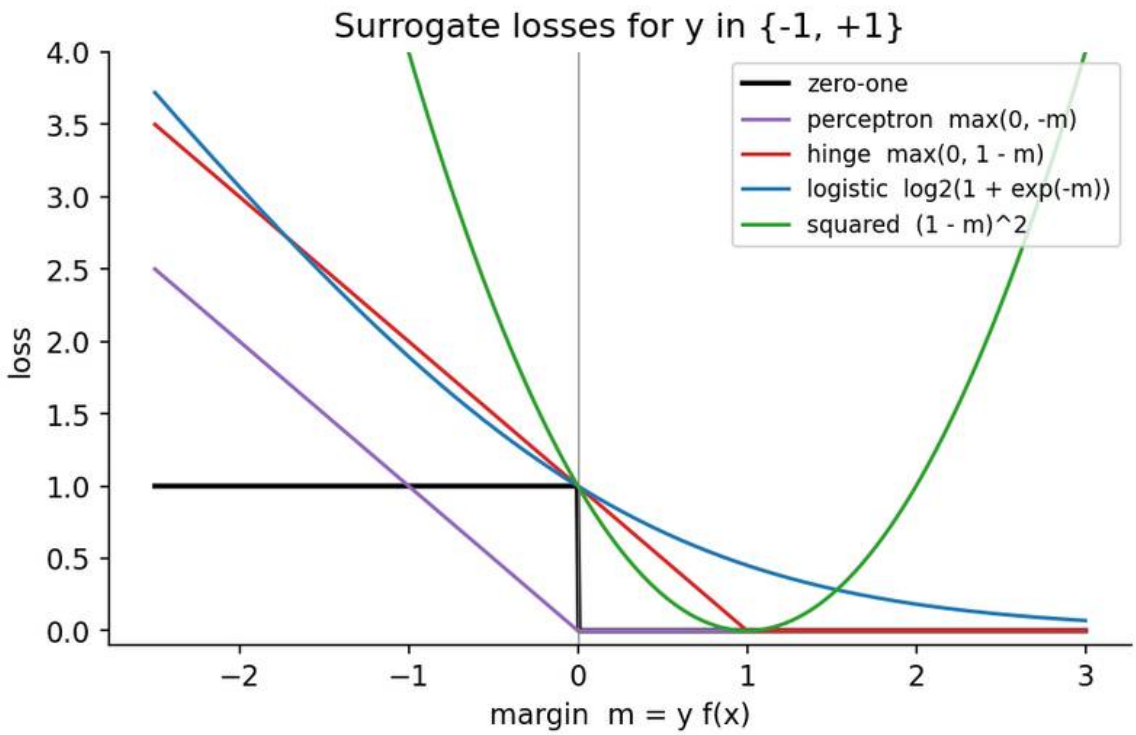


圖 *4* — 以間隔 $m = y\,f(x)$ 表示的代理損失。邏輯斯損失乘上 $1/\ln 2$，使它通過 $(0, 1)$。只有平方損失在 $m > 1$ 時增加，這解釋了圖 *1*。

在資料足夠時，哪些代理損失會得到貝氏最佳分類器？Bartlett、Jordan 與 McAuliffe [12] 給出精確的答案：凸代理損失是**分類校準的**（classification-calibrated，最小化其期望值會得到貝氏決策 $\operatorname{sign}(2\eta(x)-1)$，其中 $\eta(x)=P(y=1\mid x)$），若且唯若它在 0 可微且 $\varphi'(0)<0$。鉸鏈、邏輯斯、平方與指數損失都符合；感知器損失不符合。他們也把代理風險的超額量與分類風險的超額量連結起來。Rosasco 等人 [13] 比較不同凸損失所隱含的收斂速率，結論是對分類而言，鉸鏈損失收斂得和邏輯斯損失差不多快，比平方損失快得多。不過這些損失估計的東西仍不同：逐點最小化期望邏輯斯損失得到 $g(x)=\ln\frac{\eta(x)}{1-\eta(x)}$，所以它的分數是對數勝算（經 sigmoid 即為機率）；而期望鉸鏈損失的逐點最小解是 $\operatorname{sign}(2\eta(x)-1)$，不帶任何機率資訊。這就是 SVM 分數需要另外校準的原因。

程式碼：在 digits 上比較線性分類器

內建的 digits 資料集（8×8 影像、10 類）不需下載。這只是一次可重現的保留集示範，不是基準排名。每個縮放器都在管線內、只用訓練資料擬合。

```
from sklearn.datasets import load_digits
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.linear_model import LogisticRegression, Perceptron, RidgeClassifier
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.svm import LinearSVC

X, y = load_digits(return_X_y=True)
Xtr, Xte, ytr, yte = train_test_split(
    X, y, test_size=0.25, stratify=y, random_state=7)
models = {
    "least squares (ridge)": make_pipeline(StandardScaler(), RidgeClassifier(alpha=1.0)),
    "perceptron": make_pipeline(StandardScaler(),
                                Perceptron(max_iter=1500, random_state=7)),
    "logistic": make_pipeline(StandardScaler(),
                              LogisticRegression(C=1, max_iter=1500)),
    "linear SVM": make_pipeline(StandardScaler(),
                                LinearSVC(C=0.1, max_iter=20000, random_state=7)),
    "Gaussian LDA": LinearDiscriminantAnalysis(solver="lsqr", shrinkage="auto"),
}
for name, model in models.items():
    model.fit(Xtr, ytr)
    print(f"{name:22s} test accuracy: {accuracy_score(yte, model.predict(Xte)):.4f}")
```

在這次切分上，邏輯斯迴歸達到 97.1%，線性 SVM 96.4%，感知器 95.1%，ridge 正則化最小平方法與收縮 LDA 各為 94.7%。五者都是同樣 64 個像素上的線性機；差異完全來自損失與正則化。看過這些測試結果後再調整正則化，會讓這個測試集變成驗證集。真正的模型選擇需要獨立的驗證切分，或在訓練資料內做交叉驗證（PR 09）。

現代觀點

線性分類器沒有被深度學習取代；它們移到了深度學習的最上層。三條近期的研究脈絡說明它們為何仍然重要，而經典的綜述文獻對方法本身仍很有用。

綜述與求解器。 Burges 的教學文 [11] 仍是 SVM 最易讀的推導，從 VC 維度的動機一路講到對偶與核。Cervantes 等人 [17] 綜述 2020 年時的 SVM 分類：各種變形、應用，以及大型資料集上的訓練成本、類別不平衡與模型選擇等挑戰。計算方面，LIBLINEAR [14] 讓擁有數百萬稀疏特徵的線性 SVM 與邏輯斯迴歸成為例行工作；LIBSVM [16] 是 scikit-learn `SVC` 背後的核 SVM 參考實作；Pegasos [15] 則證明在 SVM 原始問題上做隨機次梯度下降，對線性核而言達到指定準確度所需的執行時間不直接取決於訓練集大小，這與用 SGD 訓練神經網路是同一個概念。

線性探針：衡量表徵的尺。 Alain 與 Bengio [21] 提議在網路的每一層獨立訓練線性分類器（「探針」，probe），並觀察到線性可分性隨深度單調增加。Kornblith、Shlens 與 Le [22] 在 12 個資料集上比較 16 個網路，發現把網路當固定特徵抽取器、上接邏輯斯迴歸分類器時，ImageNet 準確率與遷移準確率高度相關；但有些讓 ImageNet 準確率略為提升的正則化方式，會讓倒數第二層的特徵差很多。CLIP [23] 把它的零樣本分類器與在凍結特徵上訓練的邏輯斯迴歸探針做比較；DINOv2 [24] 也主要以凍結骨幹上的線性分類器評估其自監督特徵。Kumar 等人 [25] 證明完整微調可能扭曲好的預訓練特徵，在分布外資料上表現不如線性探針，並提出先擬合線性探針再微調（LP-FT）。換句話說：在凍結的基礎模型上，訓練一個良好正則化的邏輯斯迴歸，是很強的基準，往往也是正確的第一個實驗。

隱式偏差：邏輯斯迴歸偷偷找到了 SVM。 Soudry 等人 [19] 證明，在線性可分資料上對未正則化的邏輯斯（或交叉熵）損失做梯度下降，會在方向上收斂到硬間隔 SVM 的解；他們也把結果推廣到其他指數尾部的損失、多類別問題，以及受限設定下的深度網路。收斂速度只有對數級，這有助於解釋為何在訓練錯誤率為零後繼續訓練，仍能持續改善間隔。上面的梯度下降程式在二維中呈現了這個效果。

神經崩塌：訓練末期的 Fisher 圖像。 Papyan、Han 與 Donoho [20] 在三種架構與七個資料集上觀察到：在訓練的終端階段（訓練錯誤率為零之後），每個類別最後一層的特徵會崩塌到類別平均；類別平均排列成最大分離的對稱構型（單純形等角緊框架，simplex equiangular tight frame）；分類器權重與類別平均對齊；決策規則變成最近類別平均。用本章的語言說，類內散布 S_W 縮向零，Fisher 準則無限增大，網路最後一層變成 PR 01 的最小距離分類器。

開放問題。線性探針的結果取決於正則化與資料量，我們何時能相信它是表徵品質的衡量？隱式偏差的結果如何推廣到真實的、不可分的、有資料擴增的訓練？線性輸出層的機率在分布偏移下何時能保持校準？這些問題把本系列最古老的分類器和當前研究連在一起。

重點整理

- 線性判別函數 $g(x) = w^\top x + b$ 是以 w 為法向量的超平面；$g(x)/\|w\|$ 是帶號距離。多類別問題請每類用一個判別函數（線性機）。
- 最小平方法與 ridge 有封閉解與貝氏 MAP 詮釋，但最小平方法是差勁的分類器：它懲罰「太正確」的點，也可能遮蔽類別。
- Fisher 方向 $w \propto S_W^{-1}(m_1 - m_0)$ 最大化類間對類內的分散比；它等於高斯 LDA 的方向，後者再由平均與先驗加上門檻。

- 在可分資料上，感知器最多更新 $(R/\gamma)^2$ 次後收斂，但會停在任意一條分隔線上，在重疊資料上則會循環。
- 邏輯斯迴歸以凸的交叉熵最小化（牛頓法／IRLS）擬合對數勝算；沒有正則化時，在可分資料上權重會發散，方向趨向最大間隔解。
- 硬間隔 SVM 最大化間隔 $2/\|w\|$；它的對偶顯示只有支援向量重要，而且資料只以內積出現，為核方法（PR 04）開啟大門。軟間隔就是 ridge 正則化的鉸鏈損失。
- 這些方法都是同一個線性分數配上不同的代理損失；鉸鏈、邏輯斯與平方損失都是分類校準的，但只有邏輯斯損失能估計機率。

練習

1. **對參數線性**。證明以最小平方法擬合 $g(x) = w_0 + w_1 x + w_2 x^2$ 是線性迴歸問題，並描述若把同樣的特徵用在兩類別線性分類器上，x 上的決策區域長什麼樣子。

   提示

   建立設計矩陣的列 $(1, x, x^2)$；模型對 (w_0, w_1, w_2) 是線性的。x 的二次式最多變號兩次，所以一個類別的區域可以是一段區間，另一類則是它的補集（兩條半直線）。

2. **單位與 ridge**。兩個特徵量測同一段距離，一個以公尺、一個以毫米表示。說明沒有特徵縮放時 ridge 迴歸會怎麼做，以及為何標準化能解決問題。

   提示

   描述同樣效果時，毫米特徵的係數可以小 1,000 倍，因此受到的懲罰遠小得多；ridge 會把幾乎所有權重放到毫米特徵上。標準化之後，兩個座標在同樣的尺度上受懲罰。

3. **手算 Fisher**。兩類別的平均為 $m_0 = (0, 0)$、$m_1 = (2, 0)$，類內散布 $S_W = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}$。計算 Fisher 方向並與平均值差比較。兩個方向何時會重合？

   提示

   $S_W^{-1} = \frac{1}{3}\begin{bmatrix} 2 & -1 \\ -1 & 2 \end{bmatrix}$，所以 $w \propto S_W^{-1}(2, 0)^\top = \frac{1}{3}(4, -2) \propto (2, -1)$，因為類別沿 $(1, 1)$ 方向拉長，所以方向從平均值差 $(1, 0)$ 轉開。當 $m_1 - m_0$ 是 S_W 的特徵向量時兩方向恰好重合，例如 $m_1 = (2, 2)$。

4. **永不停止的感知器**。建構一個讓感知器永遠更新下去的一維資料集，並說明收斂定理的哪個假設不成立。

   提示

   在同一座標放兩個標籤相反的點（或任何重疊的類別）。不存在 $\gamma > 0$ 的 u，所以上界不適用；每次修正一個點的更新，都會弄壞另一個點。

5. **手算 SVM**。求兩點 $x_1 = (1, 1)$、$y_1 = +1$ 與 $x_2 = (-1, -1)$、$y_2 = -1$ 的硬間隔 SVM。給出 w、b、對偶變數與間隔寬度，並檢查 KKT 條件。

   提示

   由對稱性 $b = 0$、$w = c(1, 1)$。限制式 $y_1 w^\top x_1 = 1$ 給出 $2c = 1$，所以 $w = (\tfrac{1}{2}, \tfrac{1}{2})$，寬度 $2/\|w\| = 2\sqrt{2}$，正是兩點間的距離。由 $w = a_1 x_1 - a_2 x_2$ 與 $a_1 = a_2$ 得 $a_1 = a_2 = \tfrac{1}{4}$。兩點都是支援向量。

6. **邏輯斯與鉸鏈**。用 digits 管線，在訓練集上以 5 折交叉驗證，為多個 `C` 值擬合 `LogisticRegression` 與 `LinearSVC`。比較兩者最佳的驗證準確率，再比較它們的分數：哪一個能給出有意義的機率？另一個要如何得到機率？

   提示

   準確率通常很接近，正如損失分析所預測。只有邏輯斯迴歸的 `predict_proba` 估計後驗；對 SVM，要在保留資料上擬合校準映射（Platt scaling 或保序迴歸，例如 `CalibratedClassifierCV`）。

參考文獻

1. R. O. Duda, P. E. Hart and D. G. Stork, *Pattern Classification*, 2nd ed., Wiley, 2001 (Ch. 5, linear discriminant functions). publisher page
2. C. M. Bishop, *Pattern Recognition and Machine Learning*, Springer, 2006 (Sections 3.1–3.3, 4.1–4.3 and 7.1). official page and free PDF
3. S. Theodoridis and K. Koutroumbas, *Pattern Recognition*, 4th ed., Academic Press, 2008 (Ch. 3, linear classifiers). publisher page
4. T. Hastie, R. Tibshirani and J. Friedman, *The Elements of Statistical Learning*, 2nd ed., Springer, 2009 (Ch. 3–4 and 12). official page
5. R. A. Fisher, "The use of multiple measurements in taxonomic problems," *Annals of Eugenics*, vol. 7, no. 2, pp. 179–188, 1936. doi
6. F. Rosenblatt, "The perceptron: a probabilistic model for information storage and organization in the brain," *Psychological Review*, vol. 65, no. 6, pp. 386–408, 1958. doi
7. D. R. Cox, "The regression analysis of binary sequences," *Journal of the Royal Statistical Society, Series B*, vol. 20, no. 2, pp. 215–232, 1958. doi
8. A. E. Hoerl and R. W. Kennard, "Ridge regression: biased estimation for nonorthogonal problems," *Technometrics*, vol. 12, no. 1, pp. 55–67, 1970. doi
9. B. E. Boser, I. M. Guyon and V. N. Vapnik, "A training algorithm for optimal margin classifiers," in *Proc. 5th Annual Workshop on Computational Learning Theory (COLT)*, pp. 144–152, 1992. doi
10. C. Cortes and V. Vapnik, "Support-vector networks," *Machine Learning*, vol. 20, no. 3, pp. 273–297, 1995. doi
11. C. J. C. Burges, "A tutorial on support vector machines for pattern recognition," *Data Mining and Knowledge Discovery*, vol. 2, no. 2, pp. 121–167, 1998. doi
12. P. L. Bartlett, M. I. Jordan and J. D. McAuliffe, "Convexity, classification, and risk bounds," *Journal of the American Statistical Association*, vol. 101, no. 473, pp. 138–156, 2006. doi

13. L. Rosasco, E. De Vito, A. Caponnetto, M. Piana and A. Verri, “Are loss functions all the same?,” *Neural Computation*, vol. 16, no. 5, pp. 1063–1076, 2004. doi
14. R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang and C.-J. Lin, “LIBLINEAR: a library for large linear classification,” *Journal of Machine Learning Research*, vol. 9, pp. 1871–1874, 2008. paper
15. S. Shalev-Shwartz, Y. Singer, N. Srebro and A. Cotter, “Pegasos: primal estimated sub-gradient solver for SVM,” *Mathematical Programming*, vol. 127, no. 1, pp. 3–30, 2011. doi
16. C.-C. Chang and C.-J. Lin, “LIBSVM: a library for support vector machines,” *ACM Transactions on Intelligent Systems and Technology*, vol. 2, no. 3, article 27, 2011. doi
17. J. Cervantes, F. Garcia-Lamont, L. Rodríguez-Mazahua and A. Lopez, “A comprehensive survey on support vector machine classification: applications, challenges and trends,” *Neurocomputing*, vol. 408, pp. 189–215, 2020. doi
18. A. Y. Ng and M. I. Jordan, “On discriminative vs. generative classifiers: a comparison of logistic regression and naive Bayes,” in *Advances in Neural Information Processing Systems* (*NIPS*), 2001. paper
19. D. Soudry, E. Hoffer, M. S. Nacson, S. Gunasekar and N. Srebro, “The implicit bias of gradient descent on separable data,” *Journal of Machine Learning Research*, vol. 19, no. 70, pp. 1–57, 2018. paper
20. V. Papyan, X. Y. Han and D. L. Donoho, “Prevalence of neural collapse during the terminal phase of deep learning training,” *Proceedings of the National Academy of Sciences*, vol. 117, no. 40, pp. 24652–24663, 2020. doi
21. G. Alain and Y. Bengio, “Understanding intermediate layers using linear classifier probes,” arXiv:1610.01644, 2016. arXiv
22. S. Kornblith, J. Shlens and Q. V. Le, “Do better ImageNet models transfer better?,” in *Proc. IEEE/CVF Conference on Computer Vision and Pattern Recognition* (*CVPR*), 2019. paper
23. A. Radford et al., “Learning transferable visual models from natural language supervision,” in *Proc. International Conference on Machine Learning* (*ICML*), PMLR 139, 2021. paper
24. M. Oquab et al., “DINOv2: learning robust visual features without supervision,” *Transactions on Machine Learning Research*, 2024. arXiv
25. A. Kumar, A. Raghunathan, R. Jones, T. Ma and P. Liang, “Fine-tuning can distort pretrained features and underperform out-of-distribution,” in *Proc. International Conference on Learning Representations* (*ICLR*), 2022. arXiv

PR 04・近鄰、核與非線性分類器

從最近鄰與核技巧，到軟間隔 SVM、RBF 網路、決策樹與多層感知器：分類器如何讓決策邊界彎曲，以及每一種彎法的代價。

網站版本: https://blog.shyandram.dev/zh-tw/learn/pattern-recognition/kernels-nonlinear-classifiers/

你將學到

- 最近鄰規則為什麼不必擬合任何參數就能產生非線性決策區域，以及距離度量為何和 k 一樣重要。
- 核技巧（kernel trick）是什麼、什麼樣的函數才是有效的（Mercer）核，以及如何在有限的 Gram 矩陣上檢查。
- 軟間隔支持向量機（soft-margin SVM）：原始問題與對偶問題、支持向量，以及超參數 C 與 γ 如何塑造邊界。
- 高斯過程與相關向量機如何以機率的角度重複利用核。
- 另外三條通往非線性的路：徑向基函數（RBF）網路、以不純度（impurity）準則生長的決策樹（CART），以及多層感知器。
- 如何公平比較這些方法家族，以及它們在 2026 年的位置。

先看全貌

PR 03 建立的分類器，決策邊界都是直線、平面或超平面。許多真實問題並非如此：環內的點無法用任何一條直線和環外的點分開；手寫的「1」與「7」在像素空間中沿著彎曲、扭轉的區域互相重疊。我們需要會彎的邊界。

讓邊界彎曲的基本做法只有幾種：

1. **和儲存的樣本比較**。找出與新點最相似的訓練點，參考它們的標籤（最近鄰）。
2. **先換表徵，再維持線性**。把每個輸入 x 映射成新特徵 $\phi(x)$，讓超平面在新空間中可行。核方法隱式地做這件事；RBF 網路與多層感知器則以固定或學習得到的基底函數明確地做。
3. **把空間切成小塊**。決策樹提出一連串是非題，每條回答路徑最後落在一個帶有自己標籤的小方塊。

用最簡單的話說

想像你在公園裡替撿到的葉子分類。你可以把新葉子放在已貼好標籤的收藏旁，照抄最像的那幾片的標籤（近鄰）。你可以戴上一副特殊眼鏡，讓差異一下子跳出來，於是簡單的規則又能用了（核與特徵）。你也可以玩「二十個問題」：「比我的拇指寬嗎？邊緣有鋸齒嗎？」問到只剩一種葉子為止（決策樹）。

本章每一種方法都有同一個前提：非線性分類器只能利用量測中已有的資訊。如果兩個類別在你記錄的特徵裡看起來一模一樣，再怎麼彎曲邊界也分不開。特徵設計是 PR 05 的主題。本章的教科書背景是 Theodoridis 與 Koutroumbas 第 2、4 章 [1]，以及 Bishop 第 2.5.2、5.1–5.3、6.1–6.4、7.1–7.2 與 14.4 節 [2]。

最近鄰分類

白話版。要分類一個新點，就找出離它最近的 k 個訓練點，讓它們投票。除了把資料存起來，什麼都沒有「學」，但得到的邊界可以順著資料呈現任何形狀。

規則與它的保證

給定訓練資料對 (x_n, y_n)，$n = 1, \dots, N$，以及查詢點 x，令 $N_k(x)$ 為在某個距離 d 之下最接近 x 的 k 個訓練點索引。k 近鄰（k-NN）規則預測

$$\hat{y}(x) = \arg\max_{c} \sum_{n \in N_k(x)} w_n \, \mathbf{1}[y_n = c],$$

其中 c 走遍所有類別；$\mathbf{1}[\cdot]$ 在括號內條件成立時為 1，否則為 0；w_n 是投票權重：單純多數決時 $w_n = 1$，距離加權投票時則取 $d(x, x_n)$ 的遞減函數。

各類票數比例是類別後驗機率 $P(c \mid x)$ 的粗略估計：k-NN 其實是一種換了外衣的無母數密度方法（PR 02），鄰域大小由 k 決定，而非固定的頻寬 [2]。Cover 與 Hart 證明了 1-NN 規則的經典結果：當訓練樣本數無限增加時，它的錯誤率至多是貝氏錯誤率的兩倍 [3]。這是漸近的敘述；在有限資料下，尤其是高維度時，k-NN 可能離這個界限很遠。

度量方式是一種建模決策

歐氏距離把每個座標上的一單位差異視為同樣重要。如果一個特徵以公釐量測、另一個以公里量測，前者會主宰每個鄰域。把特徵標準化、學習一個 Mahalanobis 度量 $d(x, z)^2 = (x - z)^\top M (x - z)$（$M$ 為半正定矩陣），或以 cosine 相似度比較正規化後的嵌入（embedding），都會改變誰算是鄰居。

對於已正規化為單位長度的向量 u 與 v，

$$\|u - v\|^2 = \|u\|^2 + \|v\|^2 - 2u^\top v = 2 - 2u^\top v.$$

因此歐氏距離最近的鄰居，恰好就是 cosine 相似度最大的鄰居：兩個準則對候選點的排序剛好相反，選出的鄰居卻相同。這就是為什麼在正規化的深度特徵上做最近鄰搜尋（ML 06）時，兩者可以互換。

互動示範（網站版）：請開啟本篇教學的網站版本，操作此示範。開啟示範

選擇 k，以及預測的成本

小的 k（例如 1）會給出鋸齒狀的邊界，緊跟每個帶雜訊的訓練點：偏差低、變異高。大的 k 在較大區域內平均，使邊界平滑，但可能把小類別整個抹掉。k、度量方式與尺度都應該只用訓練資料，以交叉驗證來選擇。

k-NN 把所有成本都推到預測階段。單純的查詢要把 x 和全部 N 個訓練點比較，對 D 維輸入的成本是 $O(ND)$。樹狀索引與近似最近鄰搜尋在實務上可以降低這個成本。在高維度時還會出現另一個問題：當許多座標帶有無關的變異，隨機點之間的距離會集中在某個共同值附近，「最近」便不再代表「最相似」。本章稍後圖 3 的右圖顯示二維玩具問題上的 7-NN 邊界，那裡不會出現這些問題。

核：核技巧與 Mercer 條件

白話版。許多線性演算法只透過內積 $x^\top z$ 使用輸入。如果把每個內積換成某個「相似度函數」$k(x, z)$，而它恰好等於某個更豐富特徵空間中的內積，我們就能用線性演算法的價格得到非線性演算法，而且從頭到尾不必算出那些豐富的特徵。

把資料抬升

看圖 1 左邊的環狀資料。沒有任何直線能把內圈的圓盤和外環分開。加入第三個特徵：半徑平方 $r^2 = x_1^2 + x_2^2$。在新的三維空間中，內圈點位置低、外環點位置高，一個平面就能完美分開它們。抬升後空間中的線性分類器，在原空間中就是一個圓。

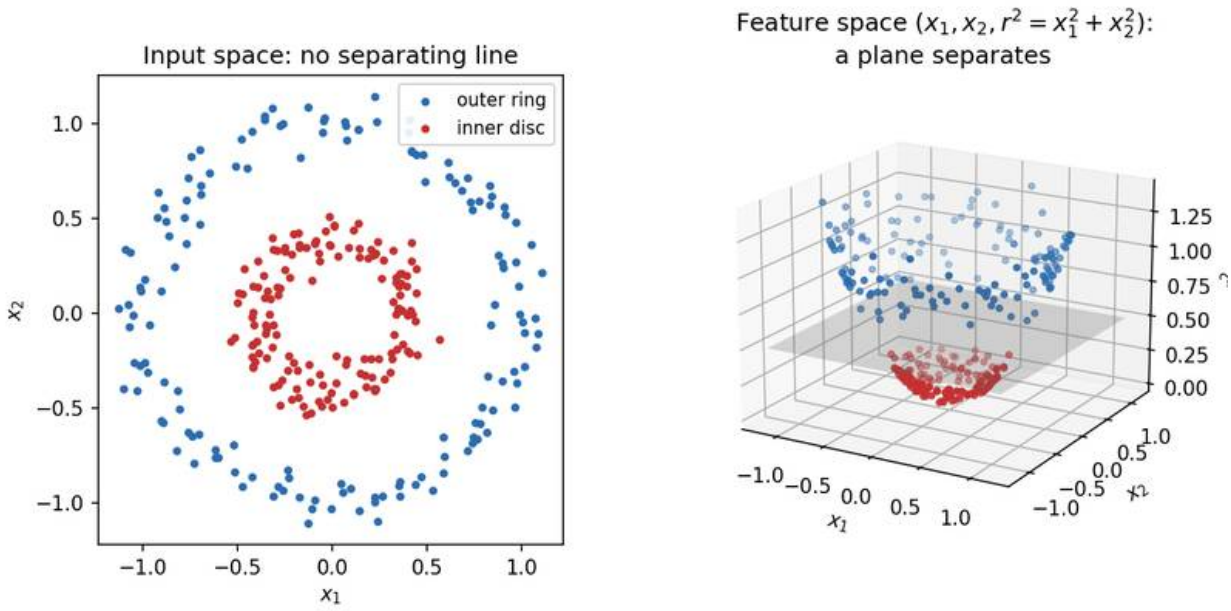


圖 *1* — 抬升到特徵空間。加入特徵 $r^2 = {x_1}^2 + {x_2}^2$ 後，圓形邊界變成一個平面。

明確抬升很快就變得昂貴。D 個變數、次數不超過 m 的所有單項式共有 $\binom{D+m}{m}$ 個；對 16×16 的影像（$D=256$）且 $m=3$，大約是 290 萬個特徵。核方法可以避免把它們建出來。

核技巧

核是一個函數 $k(x,z)$，存在某個映射到內積空間的特徵映射 ϕ，使得

$$k(x,z)=\phi(x)^{\top}\phi(z).$$

其中 x 與 z 是輸入，$\phi(x)$ 是 x 的（可能非常長、甚至無限維的）特徵向量，右式是它們的內積。舉例來說，在二維中令 $x=(a,b)$，

$$(x^{\top}z+1)^2=\phi(x)^{\top}\phi(z),\qquad \phi(x)=(1,\ \sqrt{2}a,\ \sqrt{2}b,\ a^2,\ b^2,\ \sqrt{2}ab).$$

計算左式只需一次內積與一次平方；右式每個點需要六個特徵。在 D 維、次數 m 時，左式的成本仍是 $O(D)$。

核技巧指的是：任何完全以資料點之間內積寫成的演算法，都可以把 $x_i^{\top}x_j$ 換成 $k(x_i,x_j)$ 而「核化」[7]。SVM 對偶問題、核嶺迴歸、核 PCA（PR 05）與高斯過程都屬於這一類。所有核值收集在 **Gram 矩陣** $K\in\mathbb{R}^{N\times N}$，$K_{ij}=k(x_i,x_j)$。

下面的程式以數值驗證這個等式，接著示範一個常見、卻不是有效核的「相似度」。

```
import numpy as np

rng = np.random.default_rng(0)
x, z = rng.normal(size=2), rng.normal(size=2)

def phi(v):  # explicit feature map of (v.w + 1)^2 in 2-D
    a, b = v
    s = np.sqrt(2)
    return np.array([1, s * a, s * b, a * a, b * b, s * a * b])

lhs = (x @ z + 1) ** 2
rhs = phi(x) @ phi(z)
print("kernel:", round(lhs, 6), " explicit:", round(rhs, 6))
assert np.isclose(lhs, rhs)

X = rng.normal(size=(40, 2))
K_sig = np.tanh(2.0 * X @ X.T - 1.0)        # "sigmoid kernel"
print("sigmoid Gram min eigenvalue:", np.linalg.eigvalsh(K_sig).min().round(3))
```

兩個數字相同。tanh「sigmoid 核」的 Gram 矩陣有明顯為負的特徵值（此例約為 -15），所以在這組參數下，不存在任何特徵映射 ϕ 能重現它。

哪些函數是核？Mercer 條件

函數 k 是有效的核，若且唯若它**對稱**，即 $k(x,z)=k(z,x)$，而且**半正定**：對任何有限點集 $x_1,\dots,x_N$ 與任何實數向量 $a\in\mathbb{R}^N$，

$$a^{\top} K a = \sum_{i=1}^{N}\sum_{j=1}^{N} a_i a_j\, k(x_i, x_j) \;\geq\; 0.$$

換句話說，每個 Gram 矩陣的特徵值都非負。只有對稱是不夠的，sigmoid 的例子就說明了這點。Mercer 定理 [4] 給出連續版本。對緊緻定義域 $\mathcal{X}$ 上連續且對稱的 k，若它在積分意義下半正定，

$$\int_{\mathcal{X}}\int_{\mathcal{X}} g(x)\,k(x,z)\,g(z)\,dx\,dz \geq 0 \quad \text{對所有平方可積的} g,$$

則可以展開成 $k(x,z)=\sum_{i=1}^{\infty}\lambda_i\, e_i(x)\, e_i(z)$，其中特徵值 $\lambda_i \geq 0$，e_i 為正交歸一的特徵函數。此時特徵映射就是 $\phi(x) = (\sqrt{\lambda_1}e_1(x), \sqrt{\lambda_2}e_2(x), \dots)$。用現代的說法，每個半正定核都定義一個函數構成的**再生核希爾伯特空間**（reproducing kernel Hilbert space, RKHS），而表示定理（representer theorem）指出：在這個空間中做正則化擬合，解必定是有限組合 $f(x)=\sum_{n}\alpha_n k(x_n, x)$ [7]。

有效的核可以由有效的核組合出來。若 k_1 與 k_2 是核，那麼 k_1+k_2、$c\,k_1$（$c>0$）、$k_1 k_2$、$\exp(k_1)$，以及對任意函數 f 的 $f(x)k_1(x,z)f(z)$ 也都是核 [2]。常見的核如下：

核	公式	參數	說明
線性	$x^{\top}z$	無	回到線性模型
多項式	$(\gamma x^{\top}z + r)^m$	$\gamma>0$，$r>0$，整數 $m\geq 1$	包含次數至多 m 的所有單項式（$r=0$ 時只有 m 次項）
高斯 RBF	$\exp(-\gamma\lVert x-z\rVert^2)$	$\gamma>0$	無限維特徵空間；局部相似度
Laplacian	$\exp(-\gamma\lVert x-z\rVert_1)$	$\gamma>0$	函數比高斯核粗糙
Sigmoid	$\tanh(\gamma x^{\top}z + r)$	γ, r	一般而言不是半正定

高斯 RBF 核是預設的非線性選擇。參數 γ（有時寫成 $1/(2\sigma^2)$，σ 為頻寬）決定相似度能延伸多遠。γ 大時，每個訓練點只影響極小的鄰域；γ 小時，核在資料範圍內幾乎是常數，模型趨近非常平滑、近乎線性的函數。由於 γ 乘在距離平方上，它和特徵的尺度綁在一起：尺度處理本身就是模型的一部分。不同單位混用時，標準化是實用的起點；若特徵本來就具有一致且有意義的單位，不標準化也可能較好，PR 10 綜合專題就是例子。應在驗證流程中比較選項，每個縮放器只用當折訓練資料擬合 [27]。

核嶺迴歸：一行看懂核技巧

使用特徵 ϕ 的嶺迴歸最小化 $\sum_n (t_n - w^{\top}\phi(x_n))^2 + \lambda\lVert w\rVert^2$。它的解可以寫成 $w=\sum_n \alpha_n \phi(x_n)$，於是

$$\alpha = (K + \lambda I)^{-1} t, \qquad f(x) = k_x^{\top}\alpha,$$

其中 $t=(t_1,\dots,t_N)^{\top}$ 是目標值，$\lambda>0$ 是正則化強度，I 是 $N\times N$ 單位矩陣，$k_x=(k(x_1,x),\dots,k(x_N,x))^{\top}$ 是查詢點 x 與各訓練點的相似度。原本 D 維特徵空間中的求解，變成 $N\times N$ 樣本空間中的求解：特徵多、樣本少時很划算；N 很大時則很昂貴（時間 $O(N^3)$，記憶體 $O(N^2)$）。

```
import numpy as np
from sklearn.kernel_ridge import KernelRidge
```

```
from sklearn.metrics.pairwise import rbf_kernel

rng = np.random.default_rng(1)
X = rng.uniform(-3, 3, size=(60, 1))
t = np.sin(X[:, 0]) + 0.1 * rng.normal(size=60)
lam, gamma = 0.1, 0.5

K = rbf_kernel(X, X, gamma=gamma)
alpha = np.linalg.solve(K + lam * np.eye(len(X)), t)
Xq = np.linspace(-3, 3, 5)[:, None]
f_ours = rbf_kernel(Xq, X, gamma=gamma) @ alpha
f_skl = KernelRidge(alpha=lam, kernel="rbf", gamma=gamma).fit(X, t).predict(Xq)
print(np.round(f_ours, 4))
assert np.allclose(f_ours, f_skl)
```

手寫的求解與 scikit-learn 的 `KernelRidge` 結果一致，五個查詢點上的預測也貼合 $\sin x$。

支持向量機

白話版。在所有能分開兩類的邊界中，挑出中間那條空白「街道」最寬的一條。只有位在街道邊緣或街道內的點才重要，它們就是支持向量。允許少數點以代價 C 進入街道，方法就能處理互相重疊的資料；把內積換成核，街道就能彎曲。

硬間隔與軟間隔

使用標籤 $y_n \in \{-1, +1\}$ 與分數 $f(x) = w^\top \phi(x) + b$，$w$ 是權重向量，b 是偏差；預測取 f 的正負號。若資料可分，可以調整 w 與 b 的尺度，使最近的點滿足 $y_n f(x_n) = 1$。兩條線 $f = \pm 1$ 之間的距離，也就是**間隔**（margin），等於 $2/\|w\|$。最大化間隔，等於在所有 n 都滿足 $y_n f(x_n) \geq 1$ 的條件下最小化 $\tfrac{1}{2}\|w\|^2$：這是 Boser、Guyon 與 Vapnik 的硬間隔 SVM [5]。

真實資料會重疊，因此 Cortes 與 Vapnik 引入鬆弛變數 $\xi_n \geq 0$，衡量每個點違反間隔的程度 [6]。**軟間隔 SVM** 求解

$$\min_{w,b,\xi} \; \frac{1}{2}\|w\|^2 + C\sum_{n=1}^{N} \xi_n \quad \text{subject to} \quad y_n f(x_n) \geq 1 - \xi_n,\;\; \xi_n \geq 0.$$

$\xi_n = 0$ 的點位在正確的一側，且間隔至少為 1；在最佳解處，$0 < \xi_n < 1$ 表示點落在街道內但仍分類正確；$\xi_n = 1$ 位於決策邊界，$\xi_n > 1$ 才是分類錯誤。常數 $C > 0$ 是違反間隔的價格。消去 ξ 可得等價的無約束形式

$$\min_{w,b} \; \frac{1}{2}\|w\|^2 + C\sum_{n} \max\left(0,\; 1 - y_n f(x_n)\right),$$

也就是加上正則化的**鉸鏈損失**（hinge loss）。滿足 $y_n f(x_n) \geq 1$ 的點完全不貢獻損失，這正是 SVM 稀疏性的來源。（有些教科書把損失取平均、把常數放在正則項，寫成 $\frac{\lambda}{2}\|w\|^2 + \frac{1}{N}\sum_n \text{hinge}$；兩者在 $C = 1/(\lambda N)$ 時相同。）

對偶問題與支持向量

對每個間隔約束引入拉格朗日乘數 $\alpha_n \geq 0$，再消去 w、b 與 ξ，就得到**對偶問題** [2], [6]：

$$\max_{\alpha} \sum_{n=1}^{N} \alpha_n - \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} \alpha_n \alpha_m y_n y_m \, k(x_n, x_m) \quad \text{subject to} \quad 0 \leq \alpha_n \leq C, \ \sum_{n=1}^{N} \alpha_n y_n = 0.$$

這是 N 個乘數 α_n 的凸二次規劃問題，而資料只透過核值 $k(x_n, x_m)$ 出現，核技巧就是從這裡進入。最佳權重向量是 $w = \sum_n \alpha_n y_n \phi(x_n)$，所以預測也只需要核：

$$f(x) = \sum_{n=1}^{N} \alpha_n y_n \, k(x_n, x) + b.$$

Karush–Kuhn–Tucker（KKT）最佳性條件把訓練點分成三組：

- $\alpha_n = 0$：點在街道外（$y_n f(x_n) \geq 1$），對 f 沒有影響；
- $0 < \alpha_n < C$：點恰好落在間隔上（$y_n f(x_n) = 1$），由這些點決定 b；
- $\alpha_n = C$：點在街道內或被分錯（$y_n f(x_n) \leq 1$）。

$\alpha_n > 0$ 的點就是**支持向量**。刪掉任何其他點再重新訓練，得到的分類器完全相同。支持向量有多少取決於資料：在嚴重重疊的問題上，訓練集中很大一部分都可能成為支持向量，預測就會和最近鄰搜尋一樣昂貴。

C 與 γ 如何塑造 RBF-SVM

使用高斯核時，一切由兩個數字控制。C 決定訓練錯誤相對於寬間隔的代價；γ 決定核有多局部。圖 2 在同一份帶雜訊資料上掃過兩者。

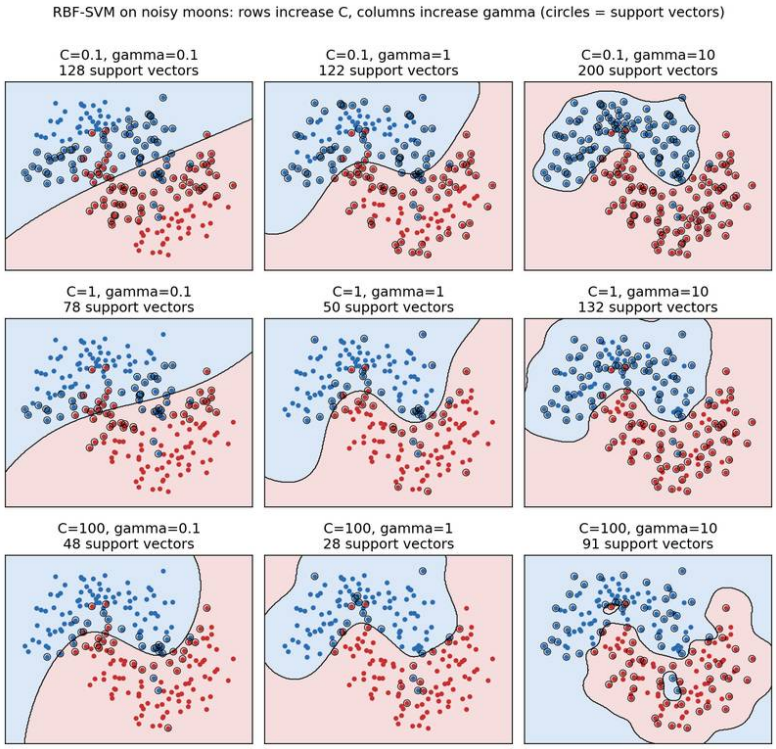


圖 2 — 帶雜訊雙月形資料（已標準化）上的 *RBF-SVM*。γ 小時邊界近乎線性；γ 大時會在個別點周圍形成小島。C 小時容許許多間隔違反（支持向量多）；C 大時更用力擬合訓練點。由 *scripts/figures/pr_04_kernels_nonlinear_classifiers_extra.py* 產生。

這張格狀圖可以這樣讀。往下走（C 變大），邊界會追著個別訓練點跑，而支持向量數變少，因為允許進入間隔的點變少了。往右走（γ 變大），每個點的影響範圍縮小；到 $\gamma = 10$ 時分類器開始畫出小島，這就是過擬合（overfitting）。好的區域是一條對角帶：γ 變大時，通常需要較小的 C。實務上，會在對數尺度的網格上用交叉驗證搜尋 C 與 γ。

```
import numpy as np
from sklearn.datasets import make_moons
from sklearn.model_selection import GridSearchCV, train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC

X, y = make_moons(n_samples=500, noise=0.18, random_state=2)
Xtr, Xte, ytr, yte = train_test_split(
    X, y, test_size=0.3, stratify=y, random_state=2)
models = {
    "linear SVM": SVC(kernel="linear", C=1),
    "RBF SVM": SVC(kernel="rbf", C=3, gamma=0.7),
    "7-neighbor": KNeighborsClassifier(n_neighbors=7),
}
for name, estimator in models.items():
    model = make_pipeline(StandardScaler(), estimator).fit(Xtr, ytr)
```

```
    print(name, "test accuracy:", round(model.score(Xte, yte), 4))

search = GridSearchCV(
    make_pipeline(StandardScaler(), SVC(kernel="rbf")),
    {"svc__C": [0.1, 1, 10, 100], "svc__gamma": [0.01, 0.1, 1, 10]},
    cv=5)
search.fit(Xtr, ytr)
best = search.best_estimator_
print("best:", search.best_params_,
      "CV acc:", round(search.best_score_, 4),
      "test acc:", round(best.score(Xte, yte), 4),
      "support vectors:", best[-1].n_support_.sum(), "of", len(Xtr))
```

在這個切分上，線性 SVM 的測試準確率是 0.827，手選參數的 RBF-SVM 與 7-NN 都是 0.96。網格搜尋選出 $C = 1$、$\gamma = 1$（交叉驗證準確率 0.971），350 個訓練點中有 59 個是支持向量，測試準確率 0.953。調參後的模型並沒有任何實質意義上的「比較差」：150 個測試點中，一個點就等於 0.0067 的準確率，兩者只差一個點。單一切分上這種大小的差異應視為雜訊（PR 09）。標準化器放在 pipeline 裡，才會在每個訓練摺上重新擬合。

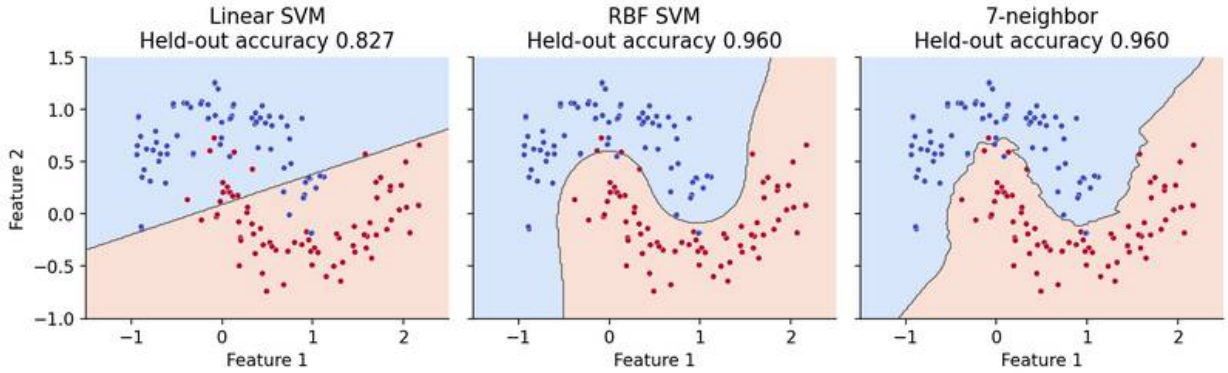


圖 3 — 與上面程式相同切分下的線性 *SVM*、*RBF-SVM* 與 *7-NN*。標準化器只以訓練資料擬合；圖中的點與分數都來自保留的測試資料。這是合成範例，不是一般性的基準測試。

分數不是機率

SVM 的輸出 $f(x)$ 是一個帶正負號、類似距離的分數，不是 $P(y = +1 \mid x)$；它的訓練目標從未要求它經過校準。若需要機率（例如拒絕選項，PR 01，或是要結合多個模型），應在保留資料上另外擬合一個由分數到機率的單調映射，並用可靠度圖檢查（PR 09）。多類別 SVM 通常由二元 SVM 組成，採用一對其餘（one-versus-rest）或一對一（one-versus-one）；scikit-learn 的 `SVC` 使用一對一。

高斯過程與相關向量

白話版。高斯過程使用同樣的核，但它不只回傳一個最佳函數，而是回傳一整個「合理函數」的分布，所以每個預測都附帶誤差棒。

高斯過程（Gaussian process, GP）是函數 f 上的先驗，使得任何有限組函數值 $(f(x_1),\dots,f(x_N))$ 都服從聯合高斯分布，平均（通常）為零、共變異數為 $K_{ij}=k(x_i,x_j)$ [8]。因此核必須是有效的共變異數，也就是半正定。對於觀測雜訊為高斯、變異數 σ^2 的迴歸，查詢點 x 的後驗是高斯分布，

$$\mu(x)=k_x^\top(K+\sigma^2 I)^{-1}t,\qquad s^2(x)=k(x,x)-k_x^\top(K+\sigma^2 I)^{-1}k_x,$$

其中 $\mu(x)$ 是預測平均，$s^2(x)$ 是潛在函數的變異數。平均值正好就是 $\lambda=\sigma^2$ 的核嶺迴歸。變異數是新的資訊：它在訓練輸入附近很小，遠離訓練輸入時回升到先驗變異數 $k(x,x)$。核的超參數（如 γ 與 σ^2）可以用最大化邊際概似（marginal likelihood）來選擇，而不必用交叉驗證 [2], [8]。

有兩點要注意。不確定度是以所選的核為條件；核選錯時，GP 可能很有自信地犯錯。另外，GP 分類比較困難：在高斯潛在函數上接伯努利概似，會讓後驗不再是高斯，因此需要 Laplace、期望傳播（expectation propagation）或變分近似（PR 08）。

相關向量機（relevance vector machine, RVM）由 Tipping 提出 [9]，並收錄於 Bishop 第 7.2 節 [2]。它使用和 SVM 相同的函數形式 $f(x)=\sum_n w_n k(x_n,x)+b$，但為每個權重 w_n 設定一個平均為零的高斯先驗，各自帶有自己的精度超參數。最大化邊際概似會把許多精度推向無窮大，等於把對應的基底函數關掉。存活下來的訓練點稱為「相關向量」，數量通常遠少於 SVM 的支持向量；輸出具有機率意義；基底函數也不必滿足 Mercer 條件。代價是訓練問題不再是凸的。因此「稀疏」有好幾種意思：儲存的基底函數少（SVM、RVM）、非零的輸入權重少（lasso），或輸入資料本身稀疏。請說清楚你指的是哪一種。

徑向基函數網路

白話版。在輸入空間中幾個有代表性的位置放上高斯「小山丘」，量測新點讓每座山丘被激發的程度，再把這些激發值交給線性分類器。這就是圖 1 的抬升想法，只是改用一小組明確的特徵。

RBF 網路計算

$$f(x)=\sum_{j=1}^{M} w_j\,\exp\!\left(-\gamma\|x-\mu_j\|^2\right)+w_0,$$

其中 $\mu_1,\dots,\mu_M$ 是 M 個**中心**，γ 決定每座山丘的寬度，w_j 是輸出權重，w_0 是偏差。Moody 與 Darken 提出經典的兩階段訓練 [10]：先不使用標籤放置中心，例如用 k-means 分群（PR 07）；中心固定後，輸出權重就是一般的線性最小平方或邏輯斯迴歸問題。Park 與 Sandberg 證明 RBF 網路是萬能近似器：只要中心夠多，就能在緊緻集合上任意逼近任何連續函數 [11]。

和前面的核機器比較：核 SVM 等於在每個支持向量上放一座山丘，由間隔準則決定；RBF 網路則使用 $M\ll N$ 座山丘，位置來自分群（若所有參數一起訓練，則來自梯度下降），所以預測很便宜。它的弱點是非監督地放置中心時忽略了標籤，中心可能浪費在兩類根本不相接的地方。

```
import numpy as np
from sklearn.cluster import KMeans
from sklearn.datasets import make_moons
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

X, y = make_moons(n_samples=500, noise=0.18, random_state=2)
```

```
Xtr, Xte, ytr, yte = train_test_split(
    X, y, test_size=0.3, stratify=y, random_state=2)
scaler = StandardScaler().fit(Xtr)
Xtr, Xte = scaler.transform(Xtr), scaler.transform(Xte)

M, gamma, lam = 12, 1.0, 1e-3
centers = KMeans(M, n_init=10, random_state=0).fit(Xtr).cluster_centers_

def design(X):  # N x (M+1): Gaussian bumps plus a bias column
    d2 = ((X[:, None, :] - centers[None, :, :]) ** 2).sum(-1)
    return np.c_[np.exp(-gamma * d2), np.ones(len(X))]

Phi = design(Xtr)
t = 2.0 * ytr - 1.0                       # targets in {-1, +1}
w = np.linalg.solve(Phi.T @ Phi + lam * np.eye(M + 1), Phi.T @ t)
pred = (design(Xte) @ w > 0).astype(int)
print("RBF network test accuracy:", round((pred == yte).mean(), 4))
```

十二座山丘加上最小平方輸出層，在雙月形切分上得到 0.953 的測試準確率，和保留數十個支持向量的 RBF-SVM 相當。圖 5 左邊是同樣的網路改用邏輯斯輸出層的結果。

決策樹

白話版。決策樹和資料玩「二十個問題」。每個問題只看一個特徵，問「它是否不超過這個門檻？」每個答案把點送往左邊或右邊，直到抵達一片帶有標籤的葉節點。樹是貪婪地長出來的：每一步都挑出能讓分出來的兩群盡可能「純」的那個問題。

不純度準則

設某個節點中的訓練點類別比例為 $p_1, \dots, p_C$（節點中屬於各類別 c 的點所占比例）。**不純度**函數在某一類占據全部點時為零，在各類均勻混合時最大。三種標準選擇為

$$\text{Gini:}\quad G = 1 - \sum_{c=1}^{C} p_c^2 = \sum_{c} p_c (1 - p_c),$$

$$\text{Entropy:}\quad H = -\sum_{c=1}^{C} p_c \log_2 p_c,$$

$$\text{Misclassification:}\quad E = 1 - \max_c p_c.$$

Gini 指標是從節點中隨機（取後放回）抽兩個點、它們標籤不同的機率。熵（entropy）是標籤在資訊理論意義下的不確定度，單位是位元。誤分類率是節點以多數類別作為預測時的錯誤率。圖 4 左邊畫出兩類情況下的三者。

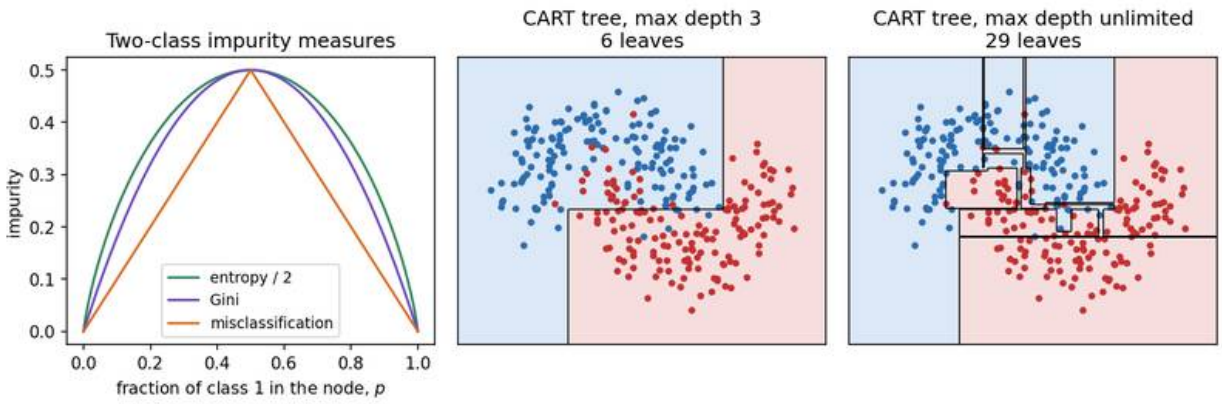


圖 4 — 左：兩類的不純度準則（熵乘以 ½ 以共用座標軸）。中：深度 3 的 *CART* 樹給出粗糙的軸對齊階梯。右：深度不限的樹在雜訊點周圍切出細小方塊，對訓練資料過擬合。

Gini 與熵都是嚴格凹函數，因此切分節點永遠不會讓它們的加權平均變大。誤分類率是分段線性的，當一次切分讓子節點更純、卻沒有改變任何子節點的多數類別時，它常常無法給予獎勵。這就是為什麼生長樹時用 Gini 或熵，評估時才用錯誤率。

長一棵樹：CART 演算法

Breiman、Friedman、Olshen 與 Stone 的 CART 程序 [12] 由上而下生長二元樹。在點集為 S 的節點，它考慮每個特徵 j 與相鄰觀測值之間的每個門檻 τ，把 S 分成 $S_L = \{x \in S : x_j \le \tau\}$ 與 $S_R = S \setminus S_L$，並選擇不純度下降量最大的切分：

$$\Delta i(j,\tau) = i(S) - \frac{|S_L|}{|S|}\, i(S_L) - \frac{|S_R|}{|S|}\, i(S_R),$$

其中 $i(\cdot)$ 是所選的不純度，$|S|$ 是集合中的點數。接著對兩個子節點遞迴，直到停止規則成立（節點已純、節點太小、達到最大深度）。Quinlan 的 ID3 [13] 是近親：它使用熵，把 Δi 稱為**資訊增益**（information gain），並對類別型特徵做多路切分。

下面的程式以暴力法找出最佳根節點切分，並和 scikit-learn 比對；scikit-learn 的 `DecisionTreeClassifier` 實作的是最佳化過的 CART。

```
import numpy as np
from sklearn.datasets import make_moons
from sklearn.tree import DecisionTreeClassifier

def gini(labels):
    p = np.bincount(labels, minlength=2) / len(labels)
    return 1.0 - (p ** 2).sum()

def best_split(X, y):
    best = (np.inf, None, None)
    for j in range(X.shape[1]):
        values = np.unique(X[:, j])
        for thr in (values[:-1] + values[1:]) / 2:
```

```
            left = X[:, j] <= thr
            n_l, n_r = left.sum(), (~left).sum()
            score = (n_l * gini(y[left]) + n_r * gini(y[~left])) / len(y)
            if score < best[0]:
                best = (score, j, thr)
    return best

X, y = make_moons(n_samples=300, noise=0.25, random_state=1)
score, j, thr = best_split(X, y)
root = DecisionTreeClassifier(max_depth=1).fit(X, y).tree_
print(f"ours: feature {j}, threshold {thr:.4f}, weighted Gini {score:.4f}")
print(f"sklearn: feature {root.feature[0]}, threshold {root.threshold[0]:.4f}")
assert j == root.feature[0] and np.isclose(thr, root.threshold[0])
```

兩者都選擇特徵 1、門檻 0.0547，把不純度從根節點的 0.5 降到加權後的 0.287。

剪枝與樹的性格

一棵長到每片葉子都純的樹會把訓練集背下來（圖 4 右）。CART 的解法是**成本複雜度剪枝**（cost-complexity pruning）[12]：先長一棵大樹 T_0，再對每個由合併內部節點得到的子樹 T 計算

$$R_\alpha(T) = R(T) + \alpha\,|\widetilde{T}|,$$

其中 $R(T)$ 是訓練誤分類率，$|\widetilde{T}|$ 是葉節點數，$\alpha \ge 0$ 是複雜度的價格。增加 α 會得到一串巢狀、越來越小的樹；再以交叉驗證選擇 α（scikit-learn 中的 `ccp_alpha`）。

樹有鮮明的性格。它不需要特徵標準化，能處理數值與類別混合的輸入，若某個無關特徵從未幫上切分就會被忽略，還能產生人看得懂的規則。它的邊界是軸對齊的階梯，所以斜的邊界要花很多次切分。它也不穩定：資料的小變動可能改變根節點的切分，進而改變整棵樹。把許多隨機化的樹平均起來（bagging、隨機森林），或依序加入（boosting），可以解決不穩定的問題，這是 PR 09 的主題。

多層感知器作為非線性分類器

白話版。多層感知器（multilayer perceptron, MLP）就像把 RBF 網路的山丘換成學習得到的「斜坡」，而且斜坡本身是從標籤學來的。每個隱藏單元畫一條線；下一層把這些線組合成彎曲的區域。

對 C 個類別，單一隱藏層的 MLP 計算

$$h = \sigma(W_1 x + b_1), \qquad P(y = c \mid x) = \mathrm{softmax}(W_2 h + b_2)_c,$$

其中 W_1, b_1 把 D 個輸入映射到 H 個隱藏單元，σ 是逐元素的非線性函數（ReLU $\max(0,\cdot)$、logistic 或 tanh），W_2, b_2 把隱藏單元映射到各類分數。所有權重以梯度下降最小化交叉熵、共同訓練，梯度由反向傳播計算 [14]。和前面所有方法不同，特徵（h）與分類器都從標籤學來。Cybenko [15] 與 Hornik [16] 的萬能近似定理保證：只要單元夠多，單一隱藏層就能在緊緻集合上逼近任何連續函數；但定理沒有說需要多少單元，也沒有說梯度下降一定找得到它們。從神經元、激活函數到反向傳播與訓練技巧的完整內容在 ML 02；影像分類的版本在 IP 13。

從幾何上看，使用 ReLU 時每個隱藏單元以一個超平面切開輸入空間，網路輸出在切出的各個區塊上是分段線性的，所以圖 5 中 MLP 的邊界由直線段組成。

```
from sklearn.datasets import make_moons
from sklearn.model_selection import train_test_split
from sklearn.neural_network import MLPClassifier
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler

X, y = make_moons(n_samples=500, noise=0.18, random_state=2)
Xtr, Xte, ytr, yte = train_test_split(
    X, y, test_size=0.3, stratify=y, random_state=2)
for hidden in [(2,), (16,), (16, 16)]:
    mlp = make_pipeline(
        StandardScaler(),
        MLPClassifier(hidden_layer_sizes=hidden, alpha=1e-3,
                      max_iter=3000, random_state=0))
    mlp.fit(Xtr, ytr)
    print(hidden, "test accuracy:", round(mlp.score(Xte, yte), 4))
```

只有兩個隱藏單元時，網路只能把邊界彎一次，準確率 0.833。十六個單元達到 0.947，再加第二層 16 個單元在這個簡單的二維問題上沒有幫助。深度在影像這類高維輸入上才顯出價值，因為它讓網路能層層建構特徵（ML 03）。

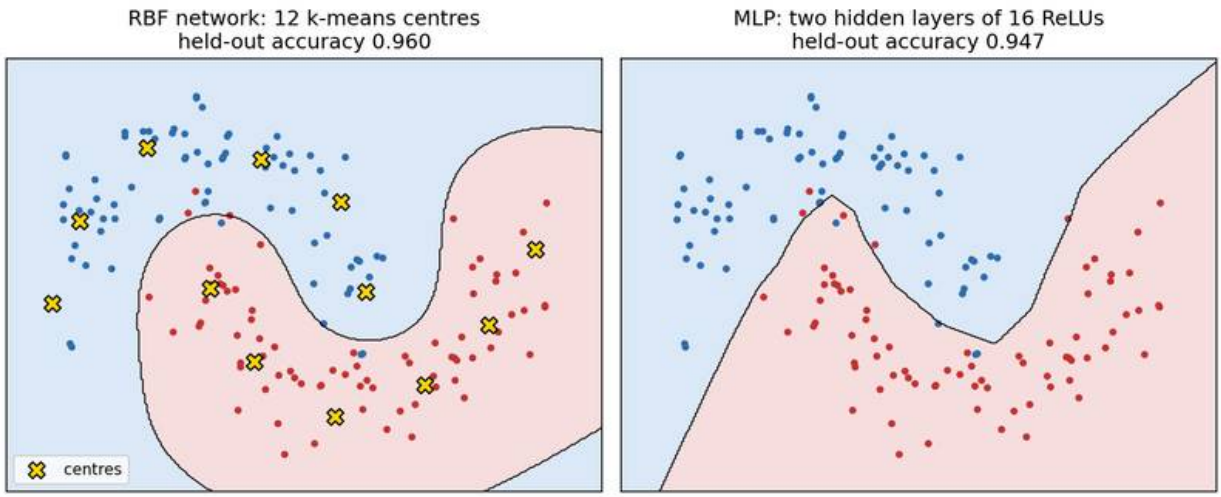


圖 *5* — 兩種明確的特徵映射。左：*12* 個 *k-means* 中心加上邏輯斯輸出層的 *RBF* 網路，給出平滑、局部的曲線。右：使用 *ReLU* 單元的 *MLP*，給出分段線性的邊界。圖中為與程式相同切分的保留資料。

如何在非線性分類器之間選擇

方法	邊界形狀	訓練成本	預測成本	主要超參數	典型優點
k-NN	任意；順著資料	無（儲存資料）	每次查詢 $O(ND)$	k、度量、尺度	簡單的基準；搭配好的嵌入時效果好
核 SVM	平滑（RBF）	約 $O(N^2)$–$O(N^3)$	$O(N_{\text{sv}}D)$	C、核、γ	中小型資料、理論完整
高斯過程	平滑，附不確定度	精確解 $O(N^3)$	平均 $O(N)$、變異數 $O(N^2)$	核；以邊際概似設定	校準良好的不確定度（迴歸）
RBF 網路	平滑、局部	分群 + 線性求解	$O(MD)$	M、γ	預測便宜
決策樹	軸對齊方塊	每層 $O(DN \log N)$	$O(\text{depth})$	深度、葉大小、α	可解釋；可處理混合型特徵
MLP	分段線性（ReLU）	迭代式 SGD	$O(\text{weights})$	寬度、深度、權重衰減、學習率	可擴展到大型資料；自行學習特徵

其中 N_{sv} 是支持向量數，M 是 RBF 中心數。複雜度欄位是數量級，不是精確計數。沒有任何一列在所有情況下都勝出。請在同一套驗證流程下比較候選方法，並記住網格搜尋範例的教訓：單一切分上不到一個百分點的差異，很少是真的。

現代觀點

從 1990 年代末到 2012 年左右深度網路接手大規模感知任務之前，核方法主導了圖形識別。它們並沒有消失，而是換了角色。以下六條線索勾勒出目前的樣貌。

經典領域的綜述。 Hofmann、Schölkopf 與 Smola 發表於 *Annals of Statistics* 的綜述 [7] 是核方法的標準地圖：正定核與其 RKHS、表示定理、為字串與圖等結構化資料建構核，以及大間隔背後的統計學習理論。Cervantes 等人 [19] 從應用面綜述 SVM 分類：在基本 SVM 表現不佳之處發展出的變體（非常大的資料集、多類別、類別不平衡）、與演化式參數搜尋的結合、大量應用案例，以及方法本身的限制。關於決策樹，Costa 與 Pedreira [20] 圍繞任何預測學習器的三個目標來回顧近年的研究：擬合訓練資料、推廣到訓練資料之外，以及保持可解釋性。

讓核方法擴展。 Gram 矩陣 $O(N^2)$ 的記憶體需求曾是主要障礙。Rahimi 與 Recht 的隨機特徵 [17] 以一個明確的隨機映射 $z(x) \in \mathbb{R}^R$ 近似 RBF 這類平移不變核，使 $z(x)^\top z(x') \approx k(x, x')$，於是在 R 個隨機特徵上的線性模型就能代替核機器。Liu 等人的綜述 [18] 整理了十年來的後續研究：隨機特徵的取樣方式、達到給定精度需要多少特徵的理論，以及與深度網路的關聯。把這類近似與 GPU 線性代數結合，FALKON 系列研究能在數十億個點上訓練核嶺迴歸 [23]；GPyTorch [24] 則以迭代式的矩陣–矩陣求解，讓精確與近似的高斯過程推論在 GPU 上變得實用。

以核的角度看深度學習。 Jacot、Gabriel 與 Hongler 證明：在適當尺度下的無限寬度極限中，以梯度下降訓練神經網路等價於使用固定**神經正切核**（neural tangent kernel）的核迴歸 [21]。Belkin、Ma 與 Mandal 指出，深度學習中令人困惑的現象，例如完全擬合訓練資料卻仍能推廣，在核機器中同樣會出現，因此理解核是理解深度網路的前提 [22]。真實的有限寬度網路確實會學習特徵，這是固定的核做不到的，所以這個類比是理論工具，而不是替代品。

樹仍然稱霸表格資料。對於表格資料（由異質量測構成的資料列，而非像素或文字），Grinsztajn、Oyallon 與 Varoquaux 在 45 個中型資料集上比較樹集成與深度架構，發現樹模型仍然領先，並把差距歸因於神經網路對無資訊特徵較敏感、具有旋轉不變性，以及偏好平滑函數 [25]。最新的

挑戰者是**表格基礎模型**：TabPFN 是一個在數百萬個合成資料集上訓練的 transformer，只要一次前向傳播就能對新的小型資料集做預測。作者報告它在樣本數至多 10,000 的資料集上勝過先前的方法，包括經過調參的梯度提升樹集成，而且訓練時間少得多 [26]。

在學習到的特徵上找鄰居。最近鄰規則作為大型預訓練嵌入之上的分類器，重新受到重視。一旦特徵夠好，k-NN 或 cosine 相似度分類器往往是最簡單又強的基準，而且不必重新訓練就能加入新類別。IP 13 與 ML 06 討論了這種做法。

未解問題。精確的核方法與 GP 在 N 上的擴展性仍然不佳，近似品質與速度之間的取捨仍是活躍的研究方向 [18]。為結構化資料選擇核大多仍靠人工。單棵樹依然不穩定，而全域最佳的樹計算代價很高 [20]。至於表格資料，基礎模型式的方法能否在小型資料集以外取代調好參數的樹集成，到 2026 年仍是開放問題。

重點整理

- 非線性邊界來自三個想法：和儲存樣本比較（k-NN）、抬升到線性規則可行的特徵空間（核、RBF 網路、MLP），或切割空間（樹）。
- k-NN 的行為由度量與尺度決定的程度不亞於 k；對單位向量而言，歐氏與 cosine 鄰居相同。
- 一個核有效，若且唯若每個 Gram 矩陣都對稱半正定（Mercer 條件）。核技巧在任何以內積寫成的演算法中，把內積換成核值。
- 軟間隔 SVM 最小化 $\frac{1}{2}\|w\|^2$ 加上 C 倍的鉸鏈損失。它的對偶問題只透過核依賴資料，只有支持向量（$\alpha_n > 0$）影響預測。在標準化的特徵上，以交叉驗證同時調整 C 與 γ。
- GP 給出附帶不確定度的核預測；RVM（Tipping）給出稀疏的機率式核模型。SVM 分數不是機率。
- RBF 網路使用少數明確的高斯山丘；樹依不純度下降（Gini 或熵）貪婪生長，需要剪枝；MLP 自行學習特徵，詳見 ML 02。
- 沒有哪個家族處處勝出：中小型平滑問題用核，表格資料用樹及其集成，原始感知資料用深度網路，好的嵌入之上用近鄰。

練習

1. **距離與 cosine**。推導單位向量的 $\|u - v\|^2 = 2 - 2u^\top v$，並證明歐氏距離下的 1-NN 等於 cosine 相似度下的 1-NN。若向量沒有正規化，結論還成立嗎？

   提示

   展開 $\|u - v\|^2 = u^\top u - 2u^\top v + v^\top v$。單位長度時，距離是 $u^\top v$ 的遞減函數，所以一者的最小值就是另一者的最大值。沒有正規化時長度會介入：一個方向稍有不同的長向量，在歐氏距離上可能比方向完全一致的短向量更近。

2. **建構核**。寫出 $\mathbb{R}^2$ 中 $k(x, z) = (x^\top z)^2$ 的特徵映射，藉此證明它是有效的核。再用組合規則論證高斯核 $\exp(-\gamma\|x - z\|^2)$ 是有效的。

   提示

$\phi(x) = (x_1^2, x_2^2, \sqrt{2}x_1x_2)$。對高斯核，寫成 $\exp(-\gamma\|x-z\|^2) = \exp(-\gamma\|x\|^2)\,\exp(2\gamma x^\top z)\,\exp(-\gamma\|z\|^2)$。中間的因子是有效（縮放過的線性）核取 exp，外側兩個因子構成 $f(x)\,k(x,z)\,f(z)$ 的形式。

3. **γ 的極限**。對固定的有限資料集，當 $\gamma \to 0$ 與 $\gamma \to \infty$ 時，RBF Gram 矩陣分別趨近什麼？把兩個極限分別對應到圖 2 最左與最右的欄。

   提示

   $\gamma \to 0$ 時每個元素都趨近 1（全為 1 的矩陣）：所有點看起來都一樣，邊界變得近乎線性或常數。$\gamma \to \infty$ 時非對角元素趨近 0，$K \to I$：每個點只和自己相似，SVM 可以背下每個訓練標籤，也就是「小島」的情形。

4. **支持向量與成本**。一個在 100,000 張影像上訓練的 RBF-SVM，保留了 90% 的影像作為支持向量。這代表資料有什麼特性？預測時間會如何？請提出兩個可以嘗試的做法。

   提示

   許多點落在間隔內或被分錯：類別在這個特徵空間中嚴重重疊，或是 C 太小、γ 選得不好。每次預測需要約 90,000 次核計算。可以重新調整 C 與 γ、改善特徵、改用隨機特徵加線性 SVM，或換一個模型家族。

5. **手算不純度**。某節點有 40 個 A 類點與 40 個 B 類點。切分 1 得到子節點 (30 A, 10 B) 與 (10 A, 30 B)；切分 2 得到 (40 A, 20 B) 與 (0 A, 20 B)。計算兩種切分的誤分類率下降量與 Gini 下降量。兩種準則各偏好哪一個切分？

   提示

   根節點 Gini $= 0.5$，錯誤率 $= 0.5$。切分 1：兩個子節點的錯誤率都是 0.25，Gini 都是 $2 \cdot 0.75 \cdot 0.25 = 0.375$；加權錯誤率 0.25，加權 Gini 0.375。切分 2：左子節點錯誤率 1/3，Gini $2 \cdot \frac{2}{3} \cdot \frac{1}{3} = 4/9$；右子節點是純的。加權錯誤率 $\frac{60}{80} \cdot \frac{1}{3} = 0.25$，加權 Gini $\frac{60}{80} \cdot \frac{4}{9} = 1/3$。錯誤率無法區分兩者（都是 0.25）；Gini 偏好切分 2，因為它產生了純節點。

6. **設計 RBF 網路**。在 RBF 網路的程式中，改變 $M \in \{2, 4, 12, 50\}$ 與 $\gamma \in \{0.1, 1, 10\}$。哪些組合欠擬合、哪些過擬合？為什麼 M 小而 γ 大的組合表現很差？

   提示

   中心少而 γ 大時，山丘很窄，大多數點幾乎不激發任何一座，設計矩陣在遠離中心處近乎為零，分類器只能退回偏差項。中心多且 γ 大時，行為開始像正則化不足的核機器。

參考文獻

1. S. Theodoridis and K. Koutroumbas, *Pattern Recognition*, 4th ed., Academic Press, 2009, Chapters 2 and 4. publisher page

2. C. M. Bishop, *Pattern Recognition and Machine Learning*, Springer, 2006, Sections 2.5.2, 5.1–5.3, 6.1–6.4, 7.1–7.2 and 14.4. official book page and PDF
3. T. Cover and P. Hart, "Nearest Neighbor Pattern Classification," *IEEE Transactions on Information Theory*, vol. 13, no. 1, pp. 21–27, 1967. doi:10.1109/TIT.1967.1053964
4. J. Mercer, "Functions of Positive and Negative Type, and Their Connection with the Theory of Integral Equations," *Philosophical Transactions of the Royal Society of London, Series A*, vol. 209, pp. 415–446, 1909. doi:10.1098/rsta.1909.0016
5. B. E. Boser, I. M. Guyon, and V. N. Vapnik, "A Training Algorithm for Optimal Margin Classifiers," *Proceedings of the Fifth Annual Workshop on Computational Learning Theory* (*COLT*), pp. 144–152, 1992. doi:10.1145/130385.130401
6. C. Cortes and V. Vapnik, "Support-Vector Networks," *Machine Learning*, vol. 20, pp. 273–297, 1995. doi:10.1007/BF00994018
7. T. Hofmann, B. Schölkopf, and A. J. Smola, "Kernel Methods in Machine Learning," *The Annals of Statistics*, vol. 36, no. 3, 2008. doi:10.1214/009053607000000677
8. C. E. Rasmussen and C. K. I. Williams, *Gaussian Processes for Machine Learning*, MIT Press, 2006. book page
9. M. E. Tipping, "Sparse Bayesian Learning and the Relevance Vector Machine," *Journal of Machine Learning Research*, vol. 1, pp. 211–244, 2001. JMLR
10. J. Moody and C. J. Darken, "Fast Learning in Networks of Locally-Tuned Processing Units," *Neural Computation*, vol. 1, no. 2, pp. 281–294, 1989. doi:10.1162/neco.1989.1.2.281
11. J. Park and I. W. Sandberg, "Universal Approximation Using Radial-Basis-Function Networks," *Neural Computation*, vol. 3, no. 2, pp. 246–257, 1991. doi:10.1162/neco.1991.3.2.246
12. L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone, *Classification and Regression Trees*, Chapman and Hall/CRC, 2017 edition. doi:10.1201/9781315139470
13. J. R. Quinlan, "Induction of Decision Trees," *Machine Learning*, vol. 1, pp. 81–106, 1986. doi:10.1007/BF00116251
14. D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning Representations by Back-Propagating Errors," *Nature*, vol. 323, pp. 533–536, 1986. doi:10.1038/323533a0
15. G. Cybenko, "Approximation by Superpositions of a Sigmoidal Function," *Mathematics of Control, Signals, and Systems*, vol. 2, pp. 303–314, 1989. doi:10.1007/BF02551274
16. K. Hornik, "Approximation Capabilities of Multilayer Feedforward Networks," *Neural Networks*, vol. 4, no. 2, pp. 251–257, 1991. doi:10.1016/0893-6080(91)90009-T
17. A. Rahimi and B. Recht, "Random Features for Large-Scale Kernel Machines," *Advances in Neural Information Processing Systems 20* (*NIPS*), 2007. paper
18. F. Liu, X. Huang, Y. Chen, and J. A. K. Suykens, "Random Features for Kernel Approximation: A Survey on Algorithms, Theory, and Beyond," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 44, no. 10, pp. 7128–7148, 2022. doi:10.1109/TPAMI.2021.3097011
19. J. Cervantes, F. Garcia-Lamont, L. Rodríguez-Mazahua, and A. Lopez, "A Comprehensive Survey on Support Vector Machine Classification: Applications, Challenges and Trends," *Neurocomputing*, vol. 408, pp. 189–215, 2020. doi:10.1016/j.neucom.2019.10.118

20. V. G. Costa and C. E. Pedreira, “Recent Advances in Decision Trees: An Updated Survey,” *Artificial Intelligence Review*, vol. 56, pp. 4765–4800, 2023. doi:10.1007/s10462-022-10275-5
21. A. Jacot, F. Gabriel, and C. Hongler, “Neural Tangent Kernel: Convergence and Generalization in Neural Networks,” *NeurIPS*, 2018. arXiv:1806.07572
22. M. Belkin, S. Ma, and S. Mandal, “To Understand Deep Learning We Need to Understand Kernel Learning,” arXiv:1802.01396, 2018. arXiv
23. G. Meanti, L. Carratino, L. Rosasco, and A. Rudi, “Kernel Methods Through the Roof: Handling Billions of Points Efficiently,” *Advances in Neural Information Processing Systems 33* (*NeurIPS*), 2020. paper
24. J. R. Gardner, G. Pleiss, D. Bindel, K. Q. Weinberger, and A. G. Wilson, “GPyTorch: Blackbox Matrix-Matrix Gaussian Process Inference with GPU Acceleration,” *NeurIPS*, 2018. arXiv:1809.11165
25. L. Grinsztajn, E. Oyallon, and G. Varoquaux, “Why Do Tree-Based Models Still Outperform Deep Learning on Typical Tabular Data?,” *Advances in Neural Information Processing Systems 35* (*NeurIPS*), *Datasets and Benchmarks Track*, 2022. paper
26. N. Hollmann, S. Müller, L. Purucker, A. Krishnakumar, et al., “Accurate Predictions on Small Data with a Tabular Foundation Model,” *Nature*, vol. 637, pp. 319–326, 2025. doi:10.1038/s41586-024-08328-6
27. scikit-learn developers, “Support Vector Machines,” user guide, accessed 2026-11-05. documentation

PR 05・特徵選擇、轉換與降維

選擇與建構特徵：過濾式、包裝式與嵌入式選擇，類別可分性準則，PCA、Fisher LDA、核 PCA、ICA 與流形學習（Isomap、LLE、t-SNE、UMAP），以及如何在不洩漏資訊的前提下驗證特徵流程。

網站版本: https://blog.shyandram.dev/zh-tw/learn/pattern-recognition/features-dimensionality/

你將學到

- 為什麼較少但較好的特徵常常勝過更多特徵，以及特徵**選擇**、特徵**萃取**與特徵**學習**的差別。
- 過濾式、包裝式與嵌入式選擇，以及最常見的錯誤：在交叉驗證之外選特徵。
- 如何為特徵與特徵子集評分：Fisher 比值、散佈矩陣準則、Bhattacharyya 距離與互資訊。
- 線性萃取：PCA（含白化）與 Fisher 線性判別分析的推導與比較。
- 非線性與非高斯的萃取：核 PCA、ICA（與 NMF），以及以 Isomap、LLE、t-SNE、UMAP 進行的流形學習，包括它們的圖能證明什麼、不能證明什麼。
- 經典的人工設計影像與訊號描述子，以及這一切在 2026 年的位置。

先看全貌

兩個辨識系統可以使用完全相同的分類器，行為卻大不相同，因為它們描述輸入的方式不同。紋理直方圖丟掉了「東西在哪裡」；形狀描述子可能忽略顏色；從網路照片學來的嵌入（embedding）可能悄悄編碼了背景。每一種表徵都保留部分變異、捨棄其餘部分，而分類器只能使用被保留的部分。

> **用最簡單的話說**
>
> 你要向朋友描述一個玩具，讓他能從玩具箱裡挑出來。「紅色、圓的、跟柳丁一樣大」很好用；「它是原子組成的」雖然沒錯卻沒用；把你看到的 500 件事全講一遍又記不住。好的描述很簡短，而且剛好保留了讓這個玩具和其他玩具區分開來的線索。

本章談的就是如何建立這種簡短而有用的描述。有三種做法：

- **選擇**保留原始量測的一個子集，例如從 5,000 個基因表現量中保留 20 個。保留的特徵仍然可以解釋。
- **萃取**以所有量測的函數計算出新座標，例如投影到前幾個主成分。
- **學習**從資料估計萃取規則，常用深度網路（ML 06）。PCA 本身就已是（非常簡單的）學習得到的萃取規則；這幾類是互相重疊的。

如果彈性很大的分類器可以使用全部特徵，為什麼還要費事？有三個理由。在訓練樣本數固定時，加入資訊量很弱的特徵可能讓訓練出的分類器變差，因為每多一維就多了要估計的參數與過

擬合的空間：這就是 Theodoridis 與 Koutroumbas 所討論的「維度詛咒」與峰值現象（peaking phenomenon）[1]。特徵越少，量測、儲存與計算越便宜。而且一組小而有意義的特徵，能幫助人理解分類器依賴的是什麼。Guyon 與 Elisseeff 在特徵選擇的經典導論中列出的也是這三個動機 [3]。本章的教科書背景是 Theodoridis 與 Koutroumbas 第 5–7 章 [1]，以及 Bishop 第 4.1.4–4.1.6 與 12.1–12.4 節 [2]。人工設計的影像特徵另請參考 IP 12。

特徵選擇

白話版。特徵選擇問的是：在已有的量測中，我該保留哪些？你需要一種為特徵（或特徵組合）評分的方法，以及一種搜尋方法，因為不可能把每個子集都試一遍。

D 個特徵共有 $2^D - 1$ 個非空子集：$D = 100$ 時約為 10^{30} 個。除了極小的 D，窮舉搜尋都不可行。各種方法的差別在於如何評分與如何搜尋 [3], [22], [23]。

過濾式方法

過濾式（filter）方法直接由資料計算某個準則，為每個特徵（或子集）評分，與最終的分類器無關，再保留最好的。典型的準則包括變異數（剔除近乎常數的特徵）、類別分離統計量（例如 ANOVA F 統計量或下一節的 Fisher 比值）、與標籤的相關係數，以及互資訊（mutual information）

$$I(x_j; y) = \sum_{v} \sum_{c} p(x_j = v, y = c) \, \log \frac{p(x_j = v, y = c)}{p(x_j = v)\, P(y = c)},$$

其中 x_j 是第 j 個特徵（離散化為數值 v），y 是取值為 c 的類別標籤，p、P 是由訓練集估計的聯合與邊際機率。$I(x_j; y) = 0$ 若且唯若該特徵與標籤在統計上獨立。

過濾式方法快速，且與分類器無關，但單變量過濾有兩個盲點。**冗餘**：同一個好特徵的兩份複本都會得高分，但第二份毫無貢獻。Peng、Long 與 Ding 的 mRMR 準則處理這個問題：獎勵與標籤的相關性，同時懲罰與已選特徵之間的互資訊 [6]。**交互作用**：兩個特徵可能各自毫無用處、合在一起卻具決定性。在互斥或（XOR）型態 $y = \mathbf{1}[x_1 > 0] \oplus \mathbf{1}[x_2 > 0]$ 中，每個特徵單獨看都和 y 獨立，所以任何單變量過濾都會把兩者都丟掉。

包裝式方法

包裝式（wrapper）方法以實際的分類器在候選子集上訓練，並以驗證表現作為該子集的分數 [4]。由於窮舉不可行，包裝式方法使用啟發式搜尋，例如：

- **循序前向選擇**：從空集合開始，每次加入最能提升驗證分數的特徵；
- **循序後向刪除**：從全部特徵開始，每次移除最沒用的一個；
- 允許撤回先前步驟的浮動（floating）變體 [1]。

包裝式方法透過分類器的眼睛看見交互作用與冗餘，但成本高（每個候選都要完整訓練一次），而且因為它在同一份驗證資料上嘗試許多子集，可能會對驗證集本身過擬合。提出這個名詞的 Kohavi 與 John 論文，至今仍是討論這些取捨的標準參考 [4]。

嵌入式方法

嵌入式（embedded）方法在訓練過程中順帶完成特徵選擇。最有名的例子是 lasso，它在線性模型上加入 ℓ_1 懲罰 [5]：

$$\hat{w} = \arg\min_{w} \frac{1}{2N} \sum_{n=1}^{N} \left(t_n - w^\top x_n\right)^2 + \lambda \lVert w \rVert_1, \qquad \lVert w \rVert_1 = \sum_{j=1}^{D} |w_j|,$$

其中 t_n 為目標值，x_n 為輸入，$\lambda \geq 0$ 為懲罰強度。ℓ_1 球的尖角會讓 $\hat{w}$ 的許多座標恰好為零，存活下來的座標就是被選中的特徵。同樣的想法也適用於分類用的邏輯斯迴歸。決策樹及其集成（PR 04、PR 09）也內建選擇：從未幫上任何切分的特徵就不會被使用。

選擇偏差的陷阱

監督式選擇會用到標籤。如果你先在完整資料集上選特徵，再只對分類器做交叉驗證，那麼保留的測試摺早已影響了選出哪些特徵，誤差估計就會過於樂觀。Ambroise 與 McLachlan 指出，這個錯誤讓已發表的基因表現分類器看起來近乎完美；把選擇放進每個交叉驗證摺內執行後，誤差就不再是零 [7]。下面的程式用純雜訊特徵把這個效應具體呈現出來。

```
import numpy as np
from sklearn.feature_selection import SelectKBest, f_classif
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import cross_val_score
from sklearn.pipeline import make_pipeline

rng = np.random.default_rng(0)
X = rng.normal(size=(60, 5000))          # 60 samples, 5000 pure-noise features
y = rng.integers(0, 2, size=60)          # labels unrelated to X

# Wrong: select on all data, then cross-validate the classifier only.
X_sel = SelectKBest(f_classif, k=20).fit_transform(X, y)
wrong = cross_val_score(LogisticRegression(max_iter=1000), X_sel, y, cv=5)

# Right: selection is part of the model and is refitted in every fold.
model = make_pipeline(SelectKBest(f_classif, k=20),
                      LogisticRegression(max_iter=1000))
right = cross_val_score(model, X, y, cv=5)
print(f"selection outside CV: {wrong.mean():.2f}")
print(f"selection inside CV:  {right.mean():.2f}")
```

標籤是隨機的，所以沒有任何方法真正能超過 50% 的準確率。在交叉驗證之外做選擇，回報的是 0.98；把選擇器放進 pipeline，回報 0.58，扣掉 60 個樣本估計的雜訊後就是隨機水準。這個規則可以推廣：每一個會看標籤的步驟（特徵選擇、LDA 這類監督式投影、決定保留幾個特徵）都屬於訓練程序的一部分。即使是 PCA 與標準化這類非監督步驟，也應該只在訓練摺上擬合，讓你評估的程序就是你要部署的程序。

評估特徵與特徵子集

白話版。好的特徵會讓各類別彼此離得遠，相對於每個類別本身的分散程度而言。有好幾個公式把這個想法變成一個數字，從單一特徵的簡單比值，到整個子集的準則。

單一特徵的 **Fisher 判別比值**比較類別平均之間的分散與類別之內的分散：

$$\mathrm{FDR}_j = \frac{\sum_c P_c\,(\mu_{cj} - \mu_j)^2}{\sum_c P_c\,\sigma_{cj}^2},$$

其中 P_c 是類別 c 的比例，μ_{cj} 與 σ_{cj}^2 是特徵 j 在類別 c 中的平均與變異數，μ_j 是整體平均 [1]。（兩類時，Theodoridis 與 Koutroumbas 寫成 $(\mu_1-\mu_2)^2/(\sigma_1^2+\sigma_2^2)$，概念相同。）

對特徵的**子集**，則使用散佈矩陣。令 x 為限制在該子集上的特徵向量，μ_c、μ 分別為類別平均與整體平均向量，

$$S_W = \sum_c P_c\,\Sigma_c, \qquad S_B = \sum_c P_c\,(\mu_c-\mu)(\mu_c-\mu)^{\top}, \qquad J = \operatorname{tr}(S_W^{-1} S_B),$$

其中 Σ_c 是類別 c 的共變異數矩陣，S_W 是類內散佈，S_B 是類間散佈，tr 是跡。當類別平均相對於類內分散相距很遠（在真正重要的方向上量測）時，J 就大 [1]。和把各特徵比值相加不同，J 考慮了特徵之間的相關。

另外兩個準則來自機率。兩個高斯類別 $\mathcal{N}(\mu_1,\Sigma_1)$ 與 $\mathcal{N}(\mu_2,\Sigma_2)$ 之間的 **Bhattacharyya 距離**為

$$B = \frac{1}{8}(\mu_1-\mu_2)^{\top}\Big(\frac{\Sigma_1+\Sigma_2}{2}\Big)^{-1}(\mu_1-\mu_2) + \frac{1}{2}\ln\frac{\left|\tfrac{1}{2}(\Sigma_1+\Sigma_2)\right|}{\sqrt{|\Sigma_1|\,|\Sigma_2|}},$$

$|\cdot|$ 為行列式。第一項衡量平均的分離，第二項衡量共變異數的差異。在先驗相等時，它給出貝氏錯誤率的上界 $P_e \le \tfrac{1}{2}e^{-B}$ [1]。前面介紹的、與標籤之間的**互資訊**，則是不依賴分布假設的對應版本。

下面的程式在 13 個特徵的葡萄酒（wine）資料上，比較單一特徵分數與子集準則。

```
import numpy as np
from sklearn.datasets import load_wine
from sklearn.preprocessing import StandardScaler

X, y = load_wine(return_X_y=True)
X = StandardScaler().fit_transform(X)
classes = np.unique(y)
mu = X.mean(axis=0)

# Per-feature Fisher ratio: between-class over within-class variance.
between = sum((y == c).mean() * (X[y == c].mean(0) - mu) ** 2 for c in classes)
within = sum((y == c).mean() * X[y == c].var(0) for c in classes)
fdr = between / within
print("top-3 features by Fisher ratio:", np.argsort(fdr)[::-1][:3],
      np.round(np.sort(fdr)[::-1][:3], 2))

def J(cols):  # scatter-matrix criterion trace(Sw^-1 Sb) on a feature subset
    Z = X[:, cols]
```

```
    m = Z.mean(0)
    Sw = sum((y == c).mean() * np.cov(Z[y == c].T, bias=True).reshape(len(cols), -1)
             for c in classes)
    Sb = sum((y == c).mean() * np.outer(Z[y == c].mean(0) - m, Z[y == c].mean(0) - m)
             for c in classes)
    return np.trace(np.linalg.solve(Sw, Sb))

top2 = list(np.argsort(fdr)[::-1][:2])
print("J(top-2 by ratio) =", round(J(top2), 2))
best_pair = max(((i, j) for i in range(13) for j in range(i + 1, 13)),
                key=lambda p: J(list(p)))
print("best pair by J:", best_pair, "J =", round(J(list(best_pair)), 2))
```

特徵 6、12、11 的個別比值最高（2.67、2.38、2.17）。然而依 J 選出的最佳配對是特徵 6 與 9（$J = 5.39$），而不是個別最好的前兩名（$J = 4.93$）：特徵 12 的資訊和特徵 6 重疊，特徵 9 則與它互補。這就是單變量過濾的冗餘問題，以數字呈現。

這些準則都只是代理指標。特徵集合的最終裁判，是整條流程在不洩漏資訊的情況下估計出的交叉驗證表現（PR 09）。

主成分分析

白話版。 PCA 旋轉座標系，讓第一個軸指向資料變化最大的方向，第二個軸指向剩餘變化最大的方向，依此類推。只保留前幾個軸，就得到最小平方意義下最好的低維摘要。它從不看標籤。

推導

設 $x_1, \dots, x_N \in \mathbb{R}^D$ 的平均為 $\bar{x}$，共變異數為 $\Sigma = \frac{1}{N-1}\sum_n (x_n - \bar{x})(x_n - \bar{x})^\top$。資料投影到單位向量 u 上的變異數為 $u^\top \Sigma u$。在 $u^\top u = 1$ 的約束下以拉格朗日乘數 λ 最大化它，得到

$$\Sigma u = \lambda u,$$

所以最佳方向是 Σ 的特徵向量，它捕捉到的變異數就是對應的特徵值 λ。第一主方向是特徵值最大的特徵向量；之後的主方向是其餘特徵向量，依特徵值遞減排列，且彼此正交 [2], [8], [9]。保留前 r 個方向也使重建誤差最小 [2]。本節共變異數除以 $N-1$，因此**每個樣本的平均平方歐氏誤差**為 $\frac{1}{N}\sum_n \|x_n - \hat{x}_n\|^2 = \frac{N-1}{N}\sum_{i>r}\lambda_i$。若改除以 $N-1$，才恰好等於被捨棄的特徵值總和；若要每個座標的平均，還需再除以 D。比較誤差圖與特徵值前，先說清楚正規化方式。

實務上，PCA 以中心化資料矩陣 $X_c \in \mathbb{R}^{N\times D}$（第 n 列為 $x_n - \bar{x}$）的奇異值分解（SVD）來計算：

$$X_c = U S V^\top, \qquad \lambda_i = \frac{s_i^2}{N-1}, \qquad Z = X_c V_r,$$

其中 U 與 V 的行向量正交歸一，$S = \mathrm{diag}(s_1, s_2, \dots)$ 依遞減順序存放奇異值，V_r 是 V 的前 r 行（主方向），$Z \in \mathbb{R}^{N\times r}$ 是**分數**（scores）。比值 $\lambda_i / \sum_k \lambda_k$ 是第 i 個成分的**解釋變異比例**。重建時計算 $\hat{x} = \bar{x} + V_r z$。

白化

白化（whitening）把每個分數縮放為單位變異數：

$$z_i^{\text{white}} = \frac{z_i}{\sqrt{\lambda_i + \epsilon}},$$

其中 z_i 是第 i 個成分上的分數，λ_i 是由訓練資料估計的該成分變異數，$\epsilon \geq 0$ 是避免極小特徵值造成爆量的小常數。保留的成分若皆有 $\lambda_i > 0$，取 $\epsilon = 0$ 才得到單位共變異數；若 $\epsilon > 0$，對角元素實際是 $\lambda_i/(\lambda_i + \epsilon)$，因此極小變異方向會被刻意壓低。零變異成分應捨棄，或使用正的 ϵ，不能直接除以零。這讓所有方向一樣「大聲」，有助於某些演算法（下文的 ICA、k-means、某些神經網路），但也會放大那些原本因為主要是雜訊而很小的方向。白化是一種建模選擇，不是免費的改進。

```
import numpy as np
from sklearn.datasets import load_digits
from sklearn.decomposition import PCA

X, _ = load_digits(return_X_y=True)          # 1797 images of 8x8 = 64 pixels
Xc = X - X.mean(axis=0)
U, S, Vt = np.linalg.svd(Xc, full_matrices=False)
eigvals = S ** 2 / (len(X) - 1)               # variances along each direction
ratio = eigvals / eigvals.sum()
r = int(np.searchsorted(np.cumsum(ratio), 0.90) + 1)
print("components for 90% variance:", r)

Z = Xc @ Vt[:r].T                             # scores, N x r
X_hat = Z @ Vt[:r] + X.mean(axis=0)           # reconstruction
print("relative reconstruction error:",
      round(np.linalg.norm(X - X_hat) / np.linalg.norm(Xc), 3))

mse_per_sample = np.mean(np.sum((X - X_hat) ** 2, axis=1))
assert np.isclose(mse_per_sample, (len(X) - 1) / len(X) * eigvals[r:].sum())

pca = PCA(n_components=r).fit(X)
assert np.allclose(pca.explained_variance_, eigvals[:r])
assert np.allclose(np.abs(pca.components_), np.abs(Vt[:r]))   # equal up to sign
Zw = Z / np.sqrt(eigvals[:r])                 # whitened scores
print("whitened covariance is identity:",
      np.allclose(np.cov(Zw.T), np.eye(r)))
```

64 個像素維度中，有 21 維就包含了數字影像 90% 的變異，以它們重建的相對誤差為 0.31。手寫的 SVD 與 scikit-learn 的結果除了每個成分的正負號外完全一致（特徵向量的正負號是任意的）。圖 1 顯示累積曲線與重建結果。

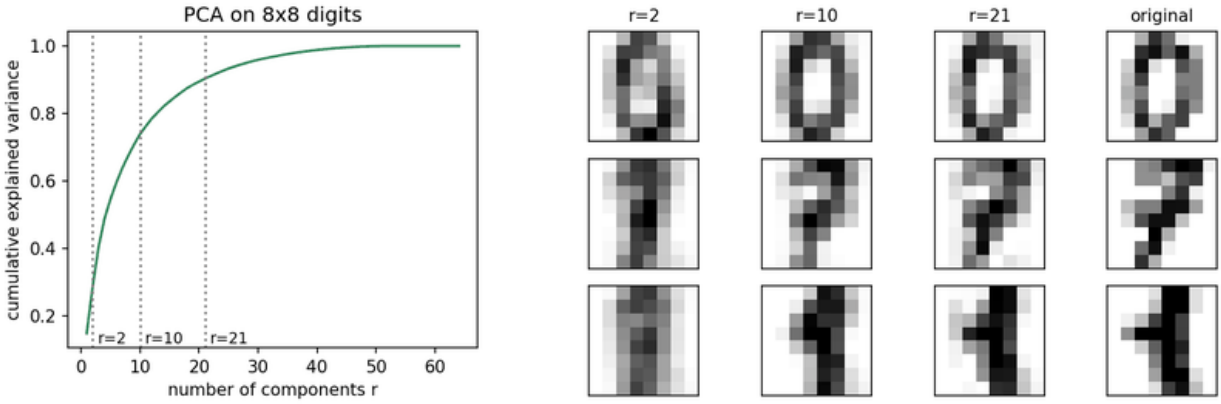


圖 *1* — *scikit-learn digits* 資料上的 *PCA*。兩個成分只得到模糊的團塊；十個成分已看得出數字；*21* 個成分（*90%* 的變異）已接近原圖。

變異不等於類別資訊

PCA 依變異數為方向排序，而變異數和對分類的用處並不相同。在圖 2 中，一個很大的干擾變異（想像成光照）主宰了第一主成分，而類別差異位在一個變異很小的方向。投影到 PC 1 保留了 96% 的變異，卻幾乎丟掉了全部的類別資訊。監督式投影（下一節的 Fisher LDA）則立刻找到有用的方向。

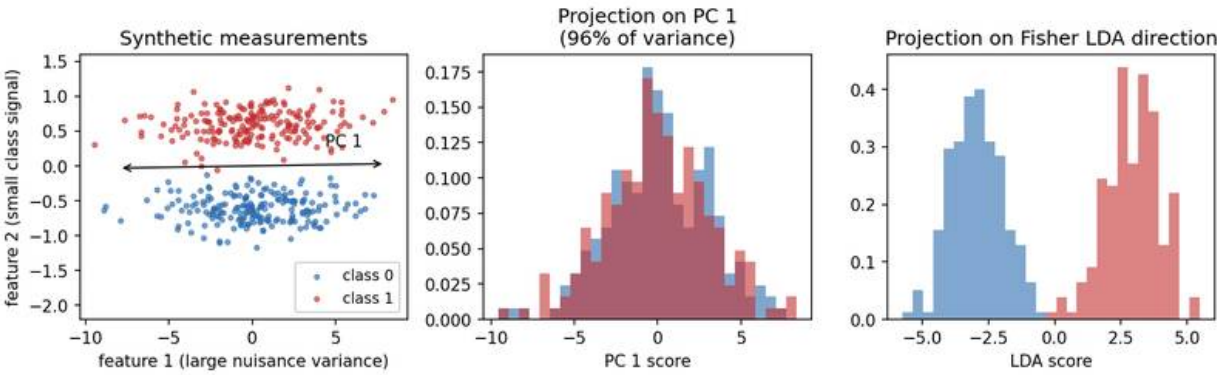


圖 *2* — 變異不等於類別資訊。*PC 1*（*96%* 的變異）沿著干擾軸，把兩類混在一起；*Fisher LDA* 方向則把它們分開。合成資料；由 *scripts/figures/pr_05_features_dimensionality_extra.py* 產生。

Jolliffe 與 Cadima 的綜述 [10] 說明了 PCA 能做與不能做的事，以及它的許多變體（例如稀疏 PCA 與穩健 PCA）。把 PCA 變成潛在變數模型的機率 PCA 與因素分析，在 Bishop 第 12 章 [2] 與 ML 06A 中有完整推導。

Fisher 線性判別分析

白話版。 LDA 尋找這樣的方向：在這些方向上，類別平均相距很遠，相對於各類別本身的分散程度。它使用標籤，所以找到的方向對分類有意義；但它最多只能產生比類別數少一個的方向。

投影方向 v 的 Fisher 準則是沿 v 的類間散佈與類內散佈之比 [11]：

$$J(v) = \frac{v^\top S_B v}{v^\top S_W v},$$

S_W 與 S_B 的定義同上一節。令梯度為零得到**廣義特徵值問題**

$$S_B v = \lambda S_W v,$$

最佳方向是特徵值 λ 最大的特徵向量（當 S_W 可逆時，等同於 $S_W^{-1} S_B$ 的特徵向量）[1], [2]。兩類時有封閉解 $v \propto S_W^{-1}(\mu_1 - \mu_2)$，它在 PR 03 中以高斯 LDA 分類器的形式再次出現。

為什麼最多 $K-1$ 個方向。 S_B 是 K 個秩為一的矩陣 $(\mu_c - \mu)(\mu_c - \mu)^\top$ 的加權和。向量 $\mu_c - \mu$ 線性相依，因為 $\sum_c P_c(\mu_c - \mu) = 0$。所以 S_B 的秩至多為 $K-1$，非零特徵值至多 $K-1$ 個。不論輸入維度多高，十個數字類別最多給出九個判別方向。

當 S_W 奇異時。樣本數少於特徵數時（影像中很常見），S_W 無法求逆。標準做法是先用 PCA、再在降維後的空間做 LDA，或把 S_W 往單位矩陣的倍數收縮（正則化 LDA）。

```
from sklearn.datasets import load_digits
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA
from sklearn.model_selection import cross_val_score
from sklearn.neighbors import KNeighborsClassifier
from sklearn.pipeline import make_pipeline
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler

X, y = load_digits(return_X_y=True)
lda = LDA().fit(X, y)
print("discriminant directions for 10 classes:", lda.transform(X).shape[1])
for name, reducer in [("PCA", PCA(n_components=2)),
                      ("LDA", LDA(n_components=2))]:
    pipe = make_pipeline(StandardScaler(), reducer,
                         KNeighborsClassifier(n_neighbors=5))
    acc = cross_val_score(pipe, X, y, cv=5).mean()
    print(f"{name} to 2-D, then 5-NN: CV accuracy {acc:.3f}")
```

一如預測，十個類別的 LDA 回傳九個方向。壓縮到二維後，在 LDA 座標上的 5-NN 分類器交叉驗證準確率為 0.624，在前兩個主成分上則為 0.512。因為 LDA 使用標籤，它被放在 pipeline 內，在每個摺中重新擬合。圖 3 說明原因：LDA 把整個數字類別拉開，PCA 則主要呈現像素變化最大的方向。

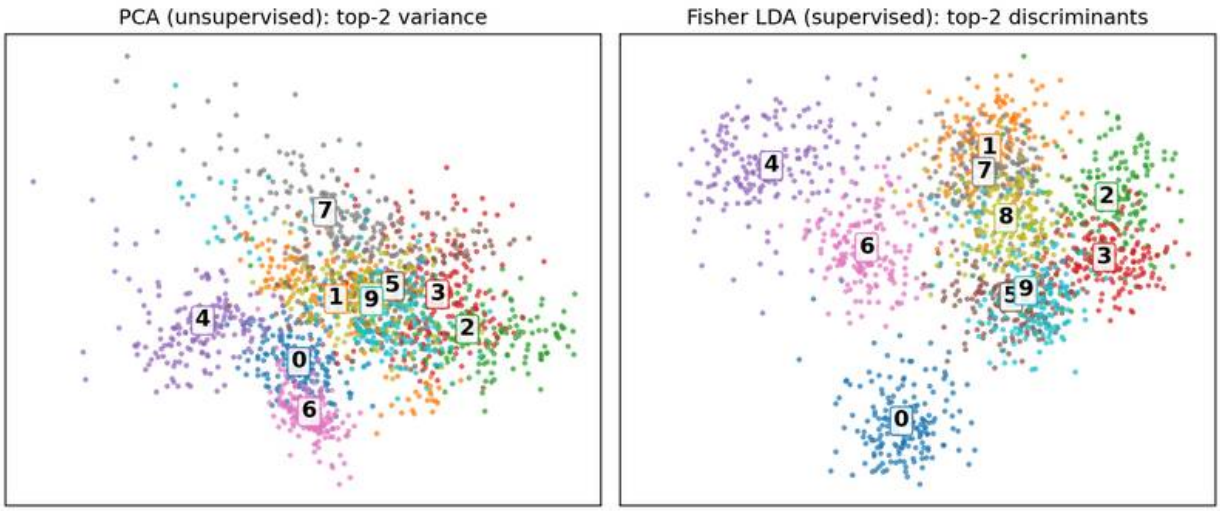


圖 3 — 數字資料投影到二維。左：前兩個主成分（未使用標籤）。右：前兩個 *Fisher* 判別方向（使用標籤）。數字標記各類別的平均位置。

核 PCA

白話版。核 PCA 就是在核的豐富特徵空間中做 PCA（PR 04），而且從不需要算出那些特徵。它能把任何線性投影都攤不開的結構攤開。

Schölkopf、Smola 與 Müller 證明，在特徵空間 $\phi(x)$ 中做 PCA 只需要核矩陣 $K_{ij} = k(x_i, x_j)$ [12]。先在特徵空間中將它中心化：

$$\tilde{K} = K - \mathbf{1}_N K - K\mathbf{1}_N + \mathbf{1}_N K \mathbf{1}_N,$$

其中 $\mathbf{1}_N$ 是每個元素皆為 $1/N$ 的 $N \times N$ 矩陣。對係數向量 $a_i \in \mathbb{R}^N$ 解特徵值問題 $\tilde{K} a_i = N\lambda_i a_i$，並正規化使 $N\lambda_i a_i^\top a_i = 1$。點 x 的第 i 個核主成分為

$$z_i(x) = \sum_{n=1}^{N} a_{in}\, \tilde{k}(x_n, x),$$

$\tilde{k}$ 是以同樣方式中心化的核。成本是一個 $N \times N$ 的特徵值問題，新點則需要 N 次核計算。

核 PCA 仍然是非監督的：它依特徵空間中的變異數為成分排序，而不是依類別分離程度。在雙環資料上，前兩個 RBF 核成分描述的是繞著環的角度，把兩類混在一起；第三個成分描述的是半徑，能把兩類完全分開（圖 4）。

```
import numpy as np
from sklearn.datasets import make_circles
from sklearn.decomposition import PCA, KernelPCA
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import cross_val_score
from sklearn.pipeline import make_pipeline
```

```
X, y = make_circles(n_samples=400, factor=0.4, noise=0.07, random_state=0)
Z = KernelPCA(n_components=4, kernel="rbf", gamma=2).fit_transform(X)
print("|corr(component, label)|:",
      np.round([abs(np.corrcoef(Z[:, i], y)[0, 1]) for i in range(4)], 3))

for name, reducer in [("linear PCA, 2 comps", PCA(n_components=2)),
                      ("kernel PCA, 3 comps", KernelPCA(n_components=3,
                                                        kernel="rbf", gamma=2))]:
    acc = cross_val_score(make_pipeline(reducer, LogisticRegression()),
                          X, y, cv=5).mean()
    print(f"{name} -> logistic regression: {acc:.3f}")
```

只有第三個成分和標籤相關（0.982；其餘都低於 0.05）。在兩個線性主成分上，邏輯斯迴歸無法勝過隨機猜測（0.432）；在三個核成分上則完全正確（1.000）。

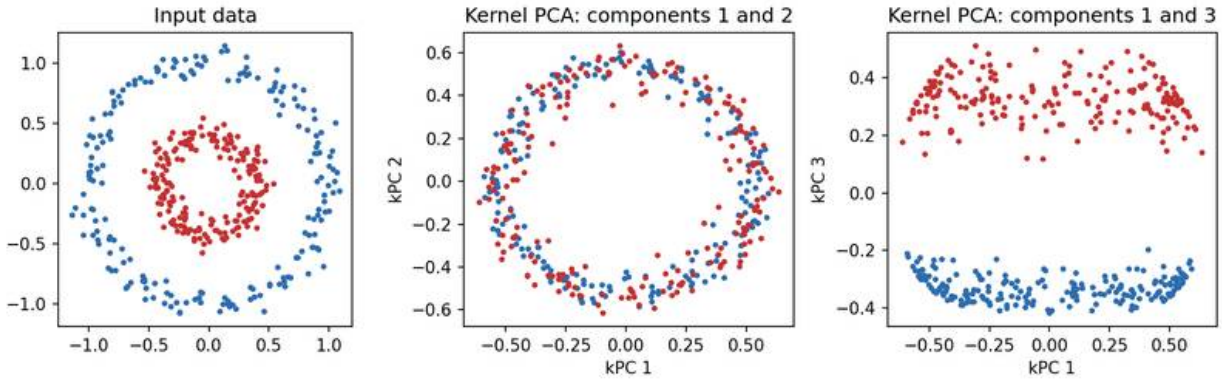


圖 *4* — 雙環資料上的核 *PCA*（*RBF*，$\gamma = 2$）。前面的成分捕捉角度並混合兩類；成分 *3* 捕捉半徑並把它們分開。特徵空間中的高變異仍然不代表高類別資訊。

獨立成分分析

白話版。在派對上，兩支麥克風各錄到兩個人聲的混合。ICA 試圖只從混合訊號中還原出各別的人聲，所依靠的是這些人聲彼此統計獨立而且不是高斯分布。

ICA 假設觀測向量是獨立訊號源的線性混合 [13]：

$$x = As,$$

其中 $s = (s_1, \dots, s_M)$ 是統計獨立、非高斯的訊號源，A 是未知的混合矩陣，x 是觀測到的訊號。目標是找到解混矩陣 $W \approx A^{-1}$，使 $\hat{s} = Wx$ 的各成分彼此獨立。為什麼要非高斯？根據中央極限定理，混合訊號往往比訊號源「更像高斯」，所以最大化 $w^{\top}x$ 的非高斯性就能找到一個訊號源。FastICA 以負熵（negentropy）的近似衡量非高斯性，並在資料白化後執行固定點迭代 [13]。有兩種模糊性永遠無法消除：訊號源的順序與尺度（包括正負號）。

和 PCA 的對比正是重點。PCA 只讓成分不相關（二階統計量）；ICA 使用高階統計量，讓成分獨立。如果訊號源是高斯的，不相關就已經等於獨立，白化後資料的任何旋轉都一樣好，ICA 就會

失敗。

```
import numpy as np
from sklearn.decomposition import FastICA

t = np.linspace(0, 8, 2000)
S = np.c_[np.sin(3 * t),                      # source 1: sine
          np.sign(np.sin(5 * t)),             # source 2: square wave
          ((2 * t) % 2) - 1]                  # source 3: sawtooth
A = np.array([[1.0, 0.5, 0.3],
              [0.6, 1.0, 0.4],
              [0.4, 0.3, 1.0]])               # unknown mixing matrix
X = S @ A.T                                   # what the sensors record

S_hat = FastICA(n_components=3, whiten="unit-variance",
                random_state=0).fit_transform(X)
corr = np.abs(np.corrcoef(S.T, S_hat.T)[:3, 3:])
print(np.round(corr, 3))                      # one entry near 1 per row
```

每個真實訊號源都恰好對應一個還原出的成分，相關係數絕對值為 0.999 或 1.000，其餘配對都低於 0.05。圖 5 顯示，對同樣的混合訊號做 PCA，得到的是互不相關、但仍然是混合的訊號。

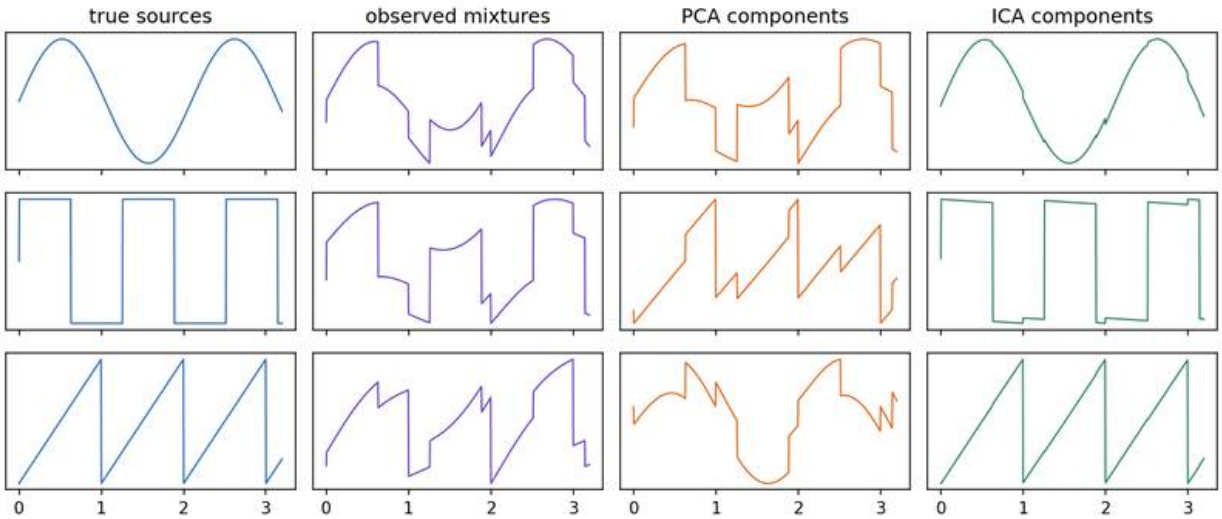


圖 5 — 盲訊號源分離。*PCA* 讓混合訊號去相關，卻沒有把它們分離；*FastICA* 在順序與尺度之外還原出正弦、方波與鋸齒波。

非負矩陣分解（non-negative matrix factorization, NMF）是適用於非負資料（例如像素強度或字詞計數）的相關分解：它把資料寫成 $X \approx WH$，其中 W, H 的每個元素都 ≥ 0。由於不能做減法，分解出的因子傾向於是可相加的「部件」，Lee 與 Seung 在人臉影像上展示了這一點 [14]。

流形學習

白話版。有時高維資料落在一張彎曲的低維薄片上，就像一張捲起來的海報。流形學習試著把薄片攤開：沿著薄片相鄰的點要保持靠近，而且不能被那些只是因為海報捲起來、才在三維中靠近的點騙過。

圖 6 的瑞士捲（Swiss roll）是標準例子：一個二維長方形在三維中捲起來。PCA 以投影把它壓平，讓捲的不同層疊在一起。以鄰域為基礎的方法表現較好：

- **Isomap** [15] 建立 k 近鄰圖，以圖上的最短路徑長度估計沿流形的測地距離，再找出歐氏距離與這些測地距離相符的低維座標（經典多維尺度法）。
- **局部線性嵌入**（locally linear embedding, LLE）[16] 把每個點寫成鄰居的加權組合 $x_n \approx \sum_{m \in \mathcal{N}(n)} W_{nm} x_m$，且 $\sum_m W_{nm} = 1$，再找出能以同一組權重重建的低維點 y_n。
- **t-SNE** [17] 把距離轉換成輸入空間中的鄰居機率 p_{ij}（高斯，頻寬由「perplexity」參數決定）與嵌入空間中的 q_{ij}（重尾的 Student-t），並最小化 Kullback–Leibler 散度 $\sum_{i \neq j} p_{ij} \log(p_{ij}/q_{ij})$。重尾讓分得很開的群可以散開。
- **UMAP** [18] 建立加權的 k 近鄰圖（一種「模糊」的拓撲表示），再以交叉熵目標最佳化低維配置。它通常比 t-SNE 快，也常用於相同目的。

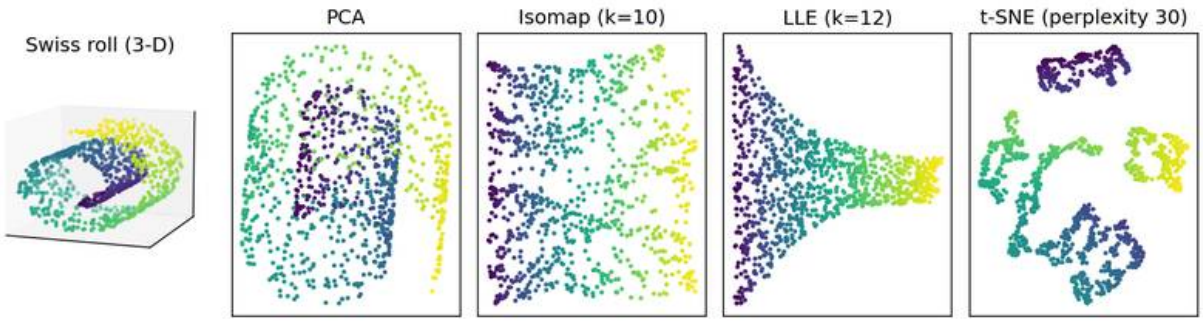


圖 6 — 瑞士捲與四種二維嵌入，依沿捲位置著色。*Isomap* 與 *LLE* 把它攤開；*PCA* 做不到；*t-SNE* 保持鄰居相鄰，卻把連續的薄片撕成分開的島嶼。

一種定量檢查是**可信度**（trustworthiness）：嵌入中每個點的 k 個最近鄰，在輸入空間中也是近鄰的程度（1.0 為完美）。

```
from sklearn.datasets import make_swiss_roll
from sklearn.decomposition import PCA
from sklearn.manifold import Isomap, LocallyLinearEmbedding, TSNE, trustworthiness

X, t = make_swiss_roll(n_samples=1000, noise=0.05, random_state=0)
methods = {
    "PCA": PCA(n_components=2),
    "Isomap": Isomap(n_neighbors=10, n_components=2),
    "LLE": LocallyLinearEmbedding(n_neighbors=12, n_components=2,
                                  random_state=0),
    "t-SNE": TSNE(n_components=2, perplexity=30, init="pca",
                  random_state=0),
}
```

```
for name, method in methods.items():
    Y = method.fit_transform(X)
    print(f"{name:7s} trustworthiness (k=10): "
          f"{trustworthiness(X, Y, n_neighbors=10):.3f}")
```

PCA 得 0.905；Isomap 與 t-SNE 得 0.999；LLE 得 0.996。注意這個數字看不到的東西：即使圖 6 顯示 t-SNE 把捲撕成了島嶼，它的分數仍和 Isomap 一樣高。局部鄰域保留了，全域結構卻沒有。

嵌入圖能證明什麼、不能證明什麼

這些方法主要是用來觀察資料的工具。從它們的運作方式可以推出三點提醒 [19]：

- **群的大小與間距預設沒有意義**。 t-SNE 會把局部密度正規化，所以緊密的群和鬆散的群可能看起來一樣大，群與群之間的距離也可能說明不了什麼。
- **結果取決於設定**。 perplexity、鄰居數、初始化與隨機種子都會改變圖形。Wattenberg、Viégas 與 Johnson 的互動式文章展示了同一份資料產生非常不同的 t-SNE 圖 [19]。
- **二維可能不夠**。投影到二維必然扭曲高維距離；圖中分得很開的島嶼並不能證明存在分開的族群。

在分類上還有一個實務缺口：t-SNE 與標準 LLE 只為訓練點學出座標，沒有自然的規則能嵌入新點。Isomap、核 PCA 與 UMAP 提供樣本外轉換，但如果非線性嵌入是分類器的一部分，就必須像其他步驟一樣在訓練摺內擬合。

影像與訊號描述子

白話版。在特徵可以學習之前，人們手工設計特徵，精確量測他們認為重要的性質：紋理、邊緣、形狀、頻率內容。它們至今仍是有用的基準與可解釋的檢查工具。

Theodoridis 與 Koutroumbas 以兩章篇幅介紹以轉換為基礎與針對應用的特徵 [1]；IP 12 詳細討論影像描述子。想一想每一種保留了什麼、對什麼敏感：

描述子	保留	典型的敏感因素
強度直方圖	數值的分布	完全失去空間排列
共生矩陣（GLCM）統計量 [20]	某一位移下成對強度的關係	量化階數與位移方向
方向梯度直方圖（HOG）[21]	格狀小區塊內的局部邊緣方向	裁切、尺度與對齊
面積、周長、緊緻度	區域幾何	分割錯誤與解析度
Fourier/DCT 係數	頻率內容	對齊方式與保留哪些頻率
小波係數	變化的位置與尺度	基底選擇與邊界處理

對量化影像 I 與位移 δ，灰階共生矩陣計算所有像素位置 p 上的配對 $(I(p), I(p+\delta))$；正規化為機率 $P(i,j)$ 後，Haralick 的紋理特徵包括對比 $\sum_{i,j}(i-j)^2 P(i,j)$ 與能量 $\sum_{i,j} P(i,j)^2$ [20]。務必註明位移與灰階數，因為數值取決於兩者。對面積為 A、周長為 P 的分割物件，緊緻度 $4\pi A/P^2$ 在理想連續圓時等於 1，形狀越不規則越小；以像素估計周長會讓小物件的數值失真。Dalal 與 Triggs

的 HOG 描述子由小區塊內正規化的梯度方向直方圖構成，最初用於行人偵測，後來成為最廣泛使用的人工設計影像特徵之一 [21]。

對音訊與其他一維訊號，特徵是在訊號近似穩定的短時窗內計算：各頻帶的頻譜能量、過零率等。時窗長度在時間解析度與頻率解析度之間取捨。

程式碼：同一個分類器，三種特徵集合

這個例子在內建的 digits 資料上，以同一個分類器比較固定的特徵策略。

```
from sklearn.datasets import load_digits
from sklearn.model_selection import train_test_split
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.feature_selection import SelectKBest, f_classif
from sklearn.linear_model import LogisticRegression

X, y = load_digits(return_X_y=True)
Xtr, Xte, ytr, yte = train_test_split(
    X, y, test_size=0.25, stratify=y, random_state=9)
transforms = {
    "all features": "passthrough",
    "24 PCA coordinates": PCA(n_components=24, random_state=9),
    "24 selected coordinates": SelectKBest(f_classif, k=24),
}
for name, transform in transforms.items():
    model = make_pipeline(
        StandardScaler(), transform, LogisticRegression(max_iter=1500))
    model.fit(Xtr, ytr)
    print(name, "test accuracy:", round(model.score(Xte, yte), 4))
```

全部 64 個像素的測試準確率為 0.969，24 個主成分為 0.956，ANOVA F 分數最高的 24 個像素為 0.953。在這裡降到約三分之一的維度，準確率損失很小。數字影像有些邊緣像素是常數，所以 F 檢定會對常數特徵發出警告；這個警告反映的是變異數為零，不是資訊洩漏，那些像素只是得不到有意義的分數。

現代觀點

人工的特徵工程大多已被學習得到的表徵取代，但本章的概念遍布現代流程：深度特徵同樣會被選擇、投影、白化、視覺化，並用相同的工具評估。以下是綜述文獻對這個領域的描繪。

特徵選擇。 Guyon 與 Elisseeff 的導論 [3] 確立了詞彙（過濾式、包裝式、嵌入式），討論了特徵建構、排序與驗證，並強調一個特徵可能單獨無用、組合起來卻有用。Chandrashekar 與 Sahin [23] 提供一份依同樣的過濾式／包裝式／嵌入式架構整理的精簡綜述。Li 等人 [22]「從資料的角度」重新檢視特徵選擇：他們把傳統資料的演算法分為基於相似度、基於資訊理論、基於稀疏學

習與基於統計四類，加入結構化、異質與串流資料的方法，並釋出開源的評測資料庫。Ambroise 與 McLachlan 關於資訊洩漏的警告 [7]，仍是最重要的單一實務教訓。

PCA 與其後繼者。 Jolliffe 與 Cadima 的綜述 [10] 把 PCA 說明為一種由資料定義的自適應技術，並整理了針對不同資料型態的變體。在現代深度學習中，PCA 以診斷工具（某一層的激活中，有多少方向承載了大部分變異）以及前處理與初始化步驟的形式延續，例如 t-SNE 建議採用的 PCA 初始化 [26]。

用於視覺化的降維。 Espadoto 等人 [24] 對投影技術做了大規模的定量綜述：他們以多種定量指標（例如鄰域與距離的保留程度）刻畫資料集、技術與投影品質，量測結果如何隨技術與資料變化，並把觀察整理成在特定情境下選擇投影方法的指引，所有程式與結果都以開放基準的形式釋出。Wang、Huang、Rudin 與 Shaposhnik [25] 分析了 t-SNE、UMAP 與 TriMAP 為何會有各自的行為，聚焦於保留局部結構與全域結構之間的取捨，並提出兼顧兩者的 PaCMAP。

二維嵌入的陷阱。 在 t-SNE 與 UMAP 已成例行工具的單細胞生物學中，出現了最尖銳的批評。Kobak 與 Berens [26] 提出讓 t-SNE 圖更忠實的流程：PCA 初始化、較高的學習率與多尺度相似度。Kobak 與 Linderman [27] 指出，常被歸功於 UMAP 的較佳全域結構，大部分來自它的初始化；使用同樣有資訊的初始化時，t-SNE 也能同樣好地保留全域結構。Chari 與 Pachter [28] 更進一步主張：極端地降到二維必然扭曲高維資料，這類圖不應用來做定量結論。

再談變異與有用性。 圖 2 的教訓在自監督學習中以更大的規模重現。Balestriero 與 LeCun [29] 認為，以重建輸入為目標訓練表徵，會強調資料中高變異的成分，而這些成分對感知任務的資訊量往往不如低變異的細節；因此以重建訓練的特徵需要額外步驟（例如微調）才能用於分類。這正是 PCA 的限制，換成了深度學習的場景。

未解問題。 在特徵高度相關、樣本又少的情況下，選擇結果依然不穩定：不同子集可能表現一樣好，讓解釋變得困難。評估嵌入仍缺乏能同時涵蓋局部與全域忠實度的公認指標 [24], [25]。至於學習得到的特徵，如何判斷哪些方向承載的是與任務相關、而不只是高變異的資訊，是活躍的研究問題 [29]。

重點整理

- 特徵選擇保留原始量測；萃取建立新座標；學習則從資料擬合萃取規則。三者都決定了分類器看得到什麼。
- 過濾式方法快速，但會漏掉交互作用與冗餘；包裝式方法看得見，但成本較高且可能對驗證資料過擬合；嵌入式方法（lasso、樹）在訓練中完成選擇。
- 任何使用標籤的步驟（特徵選擇、LDA、決定特徵數量）都必須在每個交叉驗證摺內重新擬合。在完整資料上選擇，可能在純雜訊上回報 98% 的準確率。
- Fisher 比值、$J = \mathrm{tr}(S_W^{-1} S_B)$、Bhattacharyya 距離與互資訊可為特徵評分；由於冗餘，子集必須整體評分。
- PCA 保留最大變異（共變異數的特徵向量，以 SVD 計算）；白化讓各方向變異相等。變異不等於類別資訊。
- Fisher LDA 解 $S_B v = \lambda S_W v$，最多給出 $K - 1$ 個方向。核 PCA、ICA 與 NMF 把線性萃取延伸到非線性、獨立或非負的結構。
- Isomap、LLE、t-SNE 與 UMAP 主要是視覺化工具：它們的圖取決於設定，群與群之間的間隙不能證明族群分離。

練習

1. **相同直方圖、不同影像**。構造兩張 4×4 的二值影像，強度直方圖相同但紋理不同。表中哪一種描述子能分辨它們？

   提示

   用八個值為 1 的像素，一張排成棋盤格，另一張填滿左半邊。兩者直方圖相同。水平位移 $(0, 1)$ 的共生矩陣可以分辨：棋盤格只有不相等的鄰居配對，左半填滿的影像則大多是相等的配對。

2. **$K-1$ 的上限**。證明 K 個類別的類間散佈 S_B 秩至多為 $K-1$。這對兩類問題上的 LDA 代表什麼？

   提示

   $S_B = \sum_c P_c d_c d_c^\top$，其中 $d_c = \mu_c - \mu$，且 $\sum_c P_c d_c = 0$，所以 K 個向量 d_c 至多張成 $K-1$ 維。兩類時 LDA 只給出一個方向 $v \propto S_W^{-1}(\mu_1 - \mu_2)$。

3. **99% 的變異也不夠時**。描述一個實際的成像情境，其中保留 99% 的 PCA 變異仍會丟掉類別訊號。你要如何偵測這個問題？又該怎麼做？

   提示

   大幅且獨立的光照或姿態變化，加上一個小而一致的瑕疵標記（換了外衣的圖 2）。比較有無 PCA 時的交叉驗證準確率，或逐一檢查各成分的類別分離程度（Fisher 比值）即可偵測。改用監督式投影、先把干擾因素（例如光照）正規化，或乾脆不用 PCA。

4. **XOR 陷阱**。產生在 $[-1, 1]$ 上均勻分布的 x_1, x_2 與八個無關的雜訊特徵，令 $y = \mathbf{1}[x_1 > 0] \oplus \mathbf{1}[x_2 > 0]$。分別以 `f_classif` 與 `mutual_info_classif` 為十個特徵排序，再以決策樹做循序前向選擇。哪些方法找得到 x_1 與 x_2？

   提示

   兩種單變量過濾給 x_1、x_2 的分數都接近零，和雜訊特徵一樣，因為每一個單獨都與 y 獨立。一次加一個特徵的包裝式方法，第一步也可能因為同樣的理由而卡住；能評估特徵配對的樹狀包裝式方法，或從全部特徵開始的後向刪除，則找得到它們。

5. **量化資訊洩漏**。在選擇偏差的程式中，改變雜訊特徵數（100、1,000、5,000）與樣本數（30、60、200）。「錯誤做法」的準確率如何變化？解釋這個趨勢。

   提示

   樂觀程度隨候選特徵數增加、隨樣本數減少：雜訊特徵越多，在小樣本中碰巧與標籤相關的就越多，而挑出最好的 20 個，挑中的正是這些。

6. **解讀 t-SNE 圖**。在 digits 資料上以 perplexity 5、30、100 執行 t-SNE，每種各用兩個隨機種子。圖中哪些特徵是穩定的（哪些數字彼此相鄰？），哪些會改變（群的大小、間隙、位置）？從單張這種圖，你願意宣稱什麼？

提示

局部鄰域的成員相當穩定；絕對位置、群的面積與群間距離則不穩定。站得住腳的說法是「這些點在輸入空間中局部相似」，而不是「這兩群相距很遠」或「這一群的多樣性是另一群的兩倍」。

參考文獻

1. S. Theodoridis and K. Koutroumbas, *Pattern Recognition*, 4th ed., Academic Press, 2009, Chapters 5–7. publisher page
2. C. M. Bishop, *Pattern Recognition and Machine Learning*, Springer, 2006, Sections 4.1.4–4.1.6 and 12.1–12.4. official book page and PDF
3. I. Guyon and A. Elisseeff, “An Introduction to Variable and Feature Selection,” *Journal of Machine Learning Research*, vol. 3, pp. 1157–1182, 2003. JMLR
4. R. Kohavi and G. H. John, “Wrappers for Feature Subset Selection,” *Artificial Intelligence*, vol. 97, pp. 273–324, 1997. doi:10.1016/S0004-3702(97)00043-X
5. R. Tibshirani, “Regression Shrinkage and Selection via the Lasso,” *Journal of the Royal Statistical Society, Series B*, vol. 58, no. 1, pp. 267–288, 1996. doi:10.1111/j.2517-6161.1996.tb02080.x
6. H. Peng, F. Long, and C. Ding, “Feature Selection Based on Mutual Information: Criteria of Max-Dependency, Max-Relevance, and Min-Redundancy,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 27, no. 8, pp. 1226–1238, 2005. doi:10.1109/TPAMI.2005.159
7. C. Ambroise and G. J. McLachlan, “Selection Bias in Gene Extraction on the Basis of Microarray Gene-Expression Data,” *Proceedings of the National Academy of Sciences*, vol. 99, no. 10, pp. 6562–6566, 2002. doi:10.1073/pnas.102102699
8. K. Pearson, “On Lines and Planes of Closest Fit to Systems of Points in Space,” *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, vol. 2, no. 11, pp. 559–572, 1901. doi:10.1080/14786440109462720
9. H. Hotelling, “Analysis of a Complex of Statistical Variables into Principal Components,” *Journal of Educational Psychology*, vol. 24, no. 6, pp. 417–441, 1933. doi:10.1037/h0071325
10. I. T. Jolliffe and J. Cadima, “Principal Component Analysis: A Review and Recent Developments,” *Philosophical Transactions of the Royal Society A*, vol. 374, 20150202, 2016. doi:10.1098/rsta.2015.0202
11. R. A. Fisher, “The Use of Multiple Measurements in Taxonomic Problems,” *Annals of Eugenics*, vol. 7, no. 2, pp. 179–188, 1936. doi:10.1111/j.1469-1809.1936.tb02137.x
12. B. Schölkopf, A. Smola, and K.-R. Müller, “Nonlinear Component Analysis as a Kernel Eigenvalue Problem,” *Neural Computation*, vol. 10, no. 5, pp. 1299–1319, 1998. doi:10.1162/089976698300017467
13. A. Hyvärinen and E. Oja, “Independent Component Analysis: Algorithms and Applications,” *Neural Networks*, vol. 13, no. 4–5, pp. 411–430, 2000. doi:10.1016/S0893-6080(00)00026-5
14. D. D. Lee and H. S. Seung, “Learning the Parts of Objects by Non-Negative Matrix Factorization,” *Nature*, vol. 401, pp. 788–791, 1999. doi:10.1038/44565

15. J. B. Tenenbaum, V. de Silva, and J. C. Langford, “A Global Geometric Framework for Nonlinear Dimensionality Reduction,” *Science*, vol. 290, no. 5500, pp. 2319–2323, 2000. doi:10.1126/science.290.5500.2319
16. S. T. Roweis and L. K. Saul, “Nonlinear Dimensionality Reduction by Locally Linear Embedding,” *Science*, vol. 290, no. 5500, pp. 2323–2326, 2000. doi:10.1126/science.290.5500.2323
17. L. van der Maaten and G. Hinton, “Visualizing Data using t-SNE,” *Journal of Machine Learning Research*, vol. 9, pp. 2579–2605, 2008. JMLR
18. L. McInnes, J. Healy, and J. Melville, “UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction,” arXiv:1802.03426, 2018. arXiv
19. M. Wattenberg, F. Viégas, and I. Johnson, “How to Use t-SNE Effectively,” *Distill*, 2016. doi:10.23915/distill.00002
20. R. M. Haralick, K. Shanmugam, and I. Dinstein, “Textural Features for Image Classification,” *IEEE Transactions on Systems, Man, and Cybernetics*, vol. SMC-3, no. 6, pp. 610–621, 1973. doi:10.1109/TSMC.1973.4309314
21. N. Dalal and B. Triggs, “Histograms of Oriented Gradients for Human Detection,” *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, vol. 1, pp. 886–893, 2005. doi:10.1109/CVPR.2005.177
22. J. Li, K. Cheng, S. Wang, F. Morstatter, R. P. Trevino, J. Tang, and H. Liu, “Feature Selection: A Data Perspective,” *ACM Computing Surveys*, vol. 50, no. 6, 2018. doi:10.1145/3136625
23. G. Chandrashekar and F. Sahin, “A Survey on Feature Selection Methods,” *Computers & Electrical Engineering*, vol. 40, no. 1, pp. 16–28, 2014. doi:10.1016/j.compeleceng.2013.11.024
24. M. Espadoto, R. M. Martins, A. Kerren, N. S. T. Hirata, and A. C. Telea, “Toward a Quantitative Survey of Dimension Reduction Techniques,” *IEEE Transactions on Visualization and Computer Graphics*, vol. 27, no. 3, pp. 2153–2173, 2021. doi:10.1109/TVCG.2019.2944182
25. Y. Wang, H. Huang, C. Rudin, and Y. Shaposhnik, “Understanding How Dimension Reduction Tools Work: An Empirical Approach to Deciphering t-SNE, UMAP, TriMAP, and PaCMAP for Data Visualization,” *Journal of Machine Learning Research*, vol. 22, 2021. arXiv:2012.04456
26. D. Kobak and P. Berens, “The Art of Using t-SNE for Single-Cell Transcriptomics,” *Nature Communications*, vol. 10, 2019. doi:10.1038/s41467-019-13056-x
27. D. Kobak and G. C. Linderman, “Initialization Is Critical for Preserving Global Data Structure in Both t-SNE and UMAP,” *Nature Biotechnology*, vol. 39, pp. 156–157, 2021. doi:10.1038/s41587-020-00809-z
28. T. Chari and L. Pachter, “The Specious Art of Single-Cell Genomics,” *PLOS Computational Biology*, vol. 19, e1011288, 2023. doi:10.1371/journal.pcbi.1011288
29. R. Balestriero and Y. LeCun, “Learning by Reconstruction Produces Uninformative Features for Perception,” arXiv:2402.11337, 2024. arXiv

PR 06・模板比對與結構化辨識

以相關、正規化互相關與動態時間扭曲比對圖形，以字串、文法與圖描述結構，並用圖模型、HMM（前向演算法、Viterbi）與 CRF 建模脈絡。

網站版本: https://blog.shyandram.dev/zh-tw/learn/pattern-recognition/templates-structured-models/

你將學到

- 模板比對如何以差平方和、互相關與正規化互相關（NCC）運作，以及各自能容忍哪些干擾變化。
- 動態時間扭曲（dynamic time warping, DTW）如何對齊以不同速度展開的序列，以及如何加以約束。
- 句法與結構式圖形識別：把圖形描述為字串、文法與圖，並以編輯距離與圖匹配加以比較。
- 有向與無向圖模型如何分解聯合分布，以及為什麼這讓推論變得有效率。
- 隱藏式馬可夫模型（HMM）：前向演算法、前向–後向平滑、Viterbi 解碼與 Baum–Welch 學習，附經過測試的程式。
- 條件隨機場（CRF）作為 HMM 的判別式對應版本，以及這些概念在 2026 年如何延續。

先看全貌

PR 03 與 PR 04 的分類器一次只看一個固定長度的特徵向量，並單獨決定它的類別。許多辨識問題並非如此：

- 要在影像中找到某個零件，你把參考圖片在影像上滑動，尋找最吻合的位置（**模板**）。
- 一個口說的詞或一個簽名隨時間展開，而不同的人產生它的速度不同（**對齊**）。
- 一個字元、一個分子或一張道路網，最好用它的部件以及部件如何連接來描述（**結構**）。
- 單獨一小塊影像可能是道路也可能是屋頂；潦草的手寫字母可能是「c」也可能是「e」。相鄰的區域或字母能消除這種歧義（**脈絡**）。

> **用最簡單的話說**
>
> 讀潦草的手寫字時，你不會盯著每個字母單獨看。你會把它的形狀和你認識的字母比較（模板），允許某些字母被拉長或壓扁（對齊），知道筆畫如何相連構成一個字母（結構），並用相鄰的字母猜出整個字（脈絡）。本章把這些習慣一一變成演算法。

整章有一條共同的主線：**動態規劃**。DTW、編輯距離、HMM 前向演算法、Viterbi 解碼與 CRF 推論，用的都是同一個技巧：把對所有對齊方式或標籤序列的龐大搜尋，拆成可以重複利用的小子問題，成本就從指數降到多項式。

開頭先提醒一點：只有當脈絡的假設符合現實時，脈絡才有幫助。過強的「鄰居要一致」規則可能抹掉小物件或短暫的事件，所以脈絡並非越多越好。本章的教科書背景是 Theodoridis 與 Koutroumbas 第 8、9 章 [1]，以及 Bishop 第 8.1–8.4 與 13.1–13.3 節 [2]。

模板比對

白話版。保留一張小的參考影像，也就是模板。把它在場景上滑動，在每個位置量測底下那塊影像有多相似，分數最好的位置就是匹配處。有趣的問題在於如何量測相似度，才不會被無關的變化（例如曝光變亮或變暗）騙過。

三種相似度

把模板寫成由 M 個像素值組成的向量 t，模板下方的影像區塊寫成同樣長度 M 的 x。

差平方和（SSD）。$\mathrm{SSD}(x,t) = \|x-t\|^2 = \sum_i (x_i - t_i)^2$，越小越好。這是最自然的距離，但場景中任何亮度偏移或對比變化都會讓它變大。

互相關。把平方展開，$\|x-t\|^2 = \|x\|^2 - 2x^\top t + \|t\|^2$。模板能量 $\|t\|^2$ 是常數，所以如果區塊能量 $\|x\|^2$ 也大致固定，最小化 SSD 就等同於最大化相關 $x^\top t$。實務上 $\|x\|^2$ 變化很大，單純的相關會被明亮區域吸引。

正規化互相關（NCC）。先減去平均，再除以範數：

$$\mathrm{NCC}(x,t) = \frac{(x-\bar{x}\mathbf{1})^\top (t-\bar{t}\mathbf{1})}{\|x-\bar{x}\mathbf{1}\|\,\|t-\bar{t}\mathbf{1}\|},$$

其中 $\bar{x}$ 與 $\bar{t}$ 是區塊與模板的平均像素值，$\mathbf{1}$ 是全為 1 的向量。NCC 是兩個減去平均後的向量夾角的餘弦，所以落在 $[-1,1]$；它等於 1 若且唯若對某個 $a>0$ 與任意 b 有 $x = a\,t + b\mathbf{1}$。因此它對正的對比變化（a）與亮度偏移（b）不變。模板或區塊為常數時 NCC 沒有定義（分母為零），各實作會回傳 0 或 NaN 之類的約定值。NCC 就是 IP 13 中的「以相關進行比對」。

有效率地計算

在 $N \times N$ 影像的每個位置以 $m \times m$ 模板計算 NCC，成本約為 $N^2 m^2$ 次運算。分子是一個相關運算，可以用 FFT 一次算出所有位置（相關定理，IP 04）。分母中的局部平均與能量隨位置變化；Lewis 說明如何由影像及其平方的累加表（積分影像）便宜地取得它們，讓 NCC 幾乎和單純的相關一樣快 [3]。

```
import cv2
import numpy as np
from skimage import data

img = data.camera().astype(np.float32)          # 512 x 512 grayscale
r0, c0, h, w = 300, 200, 40, 40
template = img[r0:r0 + h, c0:c0 + w].copy()
```

```
def ncc_at(window, t):
    a, b = window - window.mean(), t - t.mean()
    return float((a * b).sum() / (np.linalg.norm(a) * np.linalg.norm(b)))

# The scene is captured again under different exposure: darker and lower contrast.
scene = 0.5 * img + 20.0
ncc = cv2.matchTemplate(scene, template, cv2.TM_CCOEFF_NORMED)
ssd = cv2.matchTemplate(scene, template, cv2.TM_SQDIFF)
r, c = np.unravel_index(ncc.argmax(), ncc.shape)
print("NCC best (row, col):", (int(r), int(c)), "score", round(float(ncc.max()), 4))
r, c = np.unravel_index(ssd.argmin(), ssd.shape)
print("SSD best (row, col):", (int(r), int(c)))
print("hand-computed NCC at true location:",
      round(ncc_at(scene[r0:r0 + h, c0:c0 + w], template), 4))
```

NCC 在真實位置 (300, 200) 找到模板，分數為 1.0，手算值也和 OpenCV 的 `TM_CCOEFF_NORMED` 一致。SSD 被曝光變化騙過，選了 (130, 413)：這是一塊天空，變暗後的強度（平均約 129）比變暗後的真實區塊（平均約 92）更接近模板原本的數值（平均約 143）。

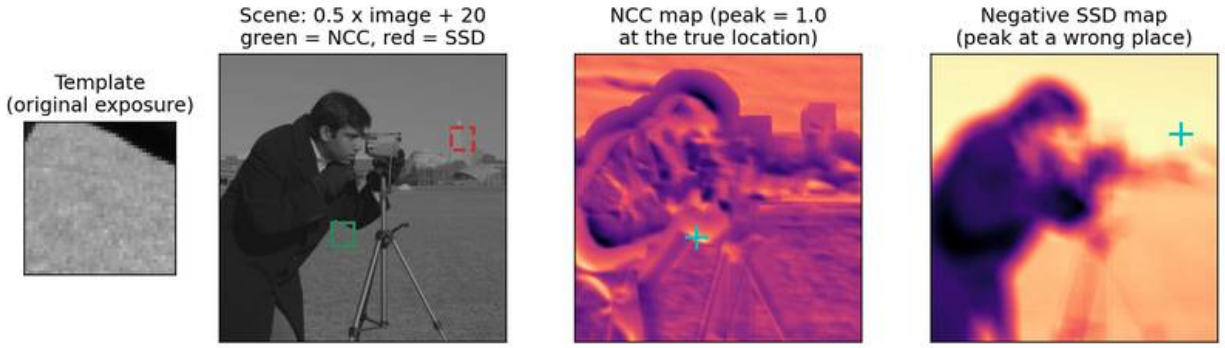


圖 *1* — 曝光變化下的模板比對。*NCC* 對增益 *0.5* 與偏移 *20* 不變，找到了真實位置；*SSD* 則否。由 *scripts/figures/pr_06_templates_structured_models_extra.py* 產生。

模板處理不了的事

NCC 只消除亮度與對比，其他一概不管。旋轉、尺度變化、透視、遮蔽與非剛性形變都會降低分數。經典的補救方法是儲存一整組模板（旋轉與縮放後的副本）、在影像金字塔上搜尋，或先比對小部件再檢查它們的幾何排列，後者就通往下文的結構式方法。當外觀受到嚴格控制時（例如工業檢測），模板比對仍是正確的工具，它也仍是追蹤與影像對位的核心步驟 [4]。

動態時間扭曲

白話版。兩個人說同一個詞，一個慢、一個快。逐點比較兩段錄音會失敗，因為聲音在時間上對不齊。DTW 拉長或壓縮其中一條序列的時間軸，讓它與另一條盡可能對齊，再量測剩下的差異。

遞迴式

令 $x = (x_1, \dots, x_n)$ 與 $t = (t_1, \dots, t_m)$ 為兩條序列，$d(x_i, t_j)$ 為局部成本，例如 $|x_i - t_j|$，或向量樣本的 $\|x_i - t_j\|^2$。**扭曲路徑**是一串索引對 $(i_1, j_1), \dots, (i_L, j_L)$，從 $(1,1)$ 開始、在 (n,m) 結束，每次只走 $(1,0)$、$(0,1)$ 或 $(1,1)$ 其中一步，因此在兩條序列中都不會倒退。DTW 是所有扭曲路徑中總成本最小者，以填表方式計算：

$$D(i,j) = d(x_i, t_j) + \min\{D(i-1,j),\ D(i,j-1),\ D(i-1,j-1)\},$$

其中 $D(0,0) = 0$，且對 $i, j \geq 1$ 有 $D(i,0) = D(0,j) = \infty$。於是 $\mathrm{DTW}(x,t) = D(n,m)$。每一格只需要它的三個鄰格，所以時間成本是 $O(nm)$。回溯每一格的最小值來自哪個鄰格，就能還原最佳對齊。這就是 Sakoe 與 Chiba 為口說詞辨識所最佳化的演算法 [5]。

約束與注意事項

不加約束的路徑可能把一條序列的單一樣本對應到另一條序列的一長段（「病態扭曲」），讓不相關的序列看起來相似。兩種標準的解法是：

- **全域約束**，例如 Sakoe–Chiba 帶 $|i - j| \leq w$，禁止遠離對角線的格子，也把成本降為 $O(nw)$ [5]；
- **斜率約束**或步伐權重，懲罰過長的水平或垂直移動 [5]。

帶寬 w 必須容得下真實的時間差。在程式與圖 2 中，3 個樣本的帶寬比真實的位移（約 6 個樣本）窄，所以加了帶的 DTW 無法完全對齊兩條正弦波。

實務上還有兩件事很重要。DTW 不是度量：它可能違反三角不等式，所以假設度量性質的資料結構要小心使用。此外，序列在比較前通常會先做 **z 正規化**（平均為零、變異數為一），讓 DTW 比較的是形狀，而不是偏移與尺度。

```
import numpy as np

def dtw(x, t, band=None):
    """DTW distance between 1-D sequences x (length n) and t (length m)."""
    n, m = len(x), len(t)
    D = np.full((n + 1, m + 1), np.inf)
    D[0, 0] = 0.0
    for i in range(1, n + 1):
        lo, hi = 1, m
        if band is not None:                    # Sakoe-Chiba band
            lo, hi = max(1, i - band), min(m, i + band)
        for j in range(lo, hi + 1):
            cost = abs(x[i - 1] - t[j - 1])
            D[i, j] = cost + min(D[i - 1, j], D[i, j - 1], D[i - 1, j - 1])
    return D[n, m]

s = np.linspace(0, 2 * np.pi, 60)
a = np.sin(s)
b = np.sin(s - 0.6)                             # same shape, shifted in time
c = 0.25 * np.sign(np.sin(3 * s))               # a different shape
```

```
print("lock-step a-b:", round(np.abs(a - b).sum(), 2),
      " a-c:", round(np.abs(a - c).sum(), 2))
print("DTW       a-b:", round(dtw(a, b), 2),
      " a-c:", round(dtw(a, c), 2))
print("DTW a-b with band 3:", round(dtw(a, b, band=3), 2))
```

兩種比較都使用同樣的局部成本 $|x_i - t_j|$。逐點（lock-step）比較時，平移後的副本 b 成本為 22.76，不同形狀的 c 為 35.18。DTW 把平移副本的成本降到 5.05，而不同形狀仍維持 34.57：扭曲吸收的是時間差，而不是形狀差。加上 3 個樣本的帶之後，平移副本的成本升到 12.76，因為對約 6 個樣本的位移而言，這個帶太窄了。

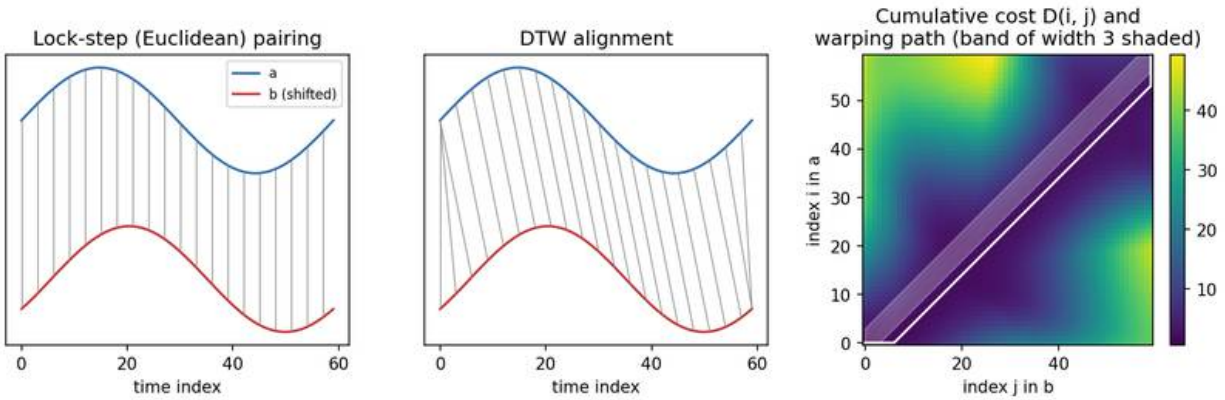


圖 2 — 動態時間扭曲。逐點比較把相同的時間索引配成一對；*DTW* 則把兩條曲線中相對應的部分配對。在成本矩陣中，最佳路徑偏離對角線約 *6* 個樣本，落在寬度 *3* 的帶外。

以 DTW 距離做 1-最近鄰分類，多年來一直是時間序列分類中強而難以超越的基準 [16]。結合便宜的下界與及早放棄無望的候選之後，在極長的序列中以 DTW 搜尋最近鄰才變得可行 [6]。

句法與結構式圖形識別

白話版。有些圖形最好不要用一串數字描述，而是用它的部件以及部件如何組合來描述，就像句子是依文法排列的字詞。結構式圖形識別把圖形描述成由基本部件組成的字串、樹或圖，再比較或剖析這些結構。

字串與文法

先選一小組**基本元素**（primitives）構成字母表，例如沿物件邊界走一步的四個方向。於是一個形狀就變成一個字串。4 方向**鏈碼**（chain code）在追蹤邊界時，向右一步寫 0、向上寫 1、向左寫 2、向下寫 3（參見 IP 12）。圖 3 顯示三個輪廓與它們的鏈碼。

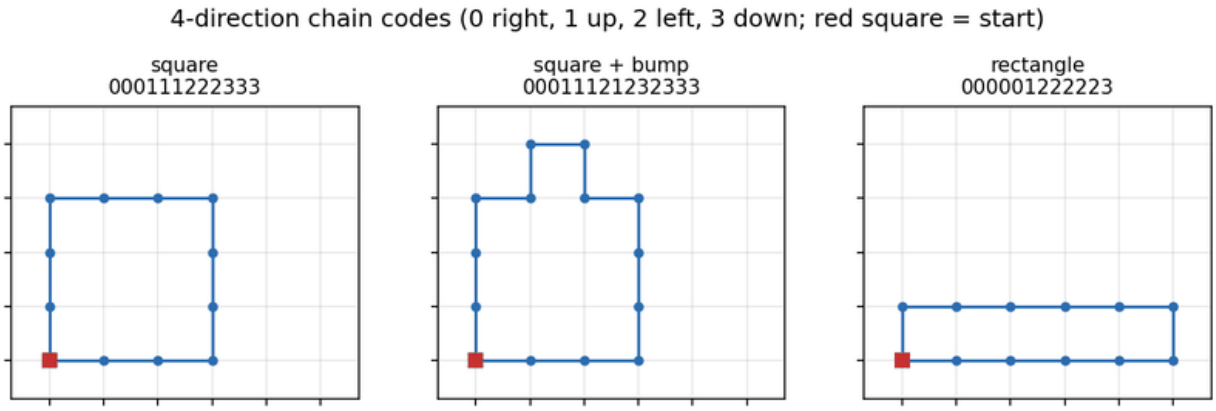


圖 3 — 把形狀寫成字串。每個輪廓都從紅色起點逆時針追蹤，邊界每走一步寫下一個符號。

在**句法**（syntactic）圖形識別中，每個類別由一套形式文法描述：一組改寫規則，恰好產生該類別所有合法的字串。辨識就是**剖析**：若某類別的文法能產生這個字串，字串就屬於該類別。例如規則 $S \to 0S2 \mid 1$ 產生字串 $0^k 1\, 2^k$，這個「類別」中每段向右的步數都有等長的向左步數與之對應。文法可以表達遞迴結構（巢狀部件、重複的子形狀），這是固定長度特徵向量做不到的。它的弱點是：真實資料有雜訊，單一個錯誤的基本元素就會讓字串不合文法；而且文法很難從範例中學出來。為規則附上機率的隨機文法可以處理第一個問題；下文的隱藏式馬可夫模型就是最簡單的隨機文法 [1]。

編輯距離

一種更寬容的做法，是用「把一個字串變成另一個的最低成本」來比較字串。**編輯距離**（Levenshtein 距離）計算最少需要幾次插入、刪除與替換。Wagner 與 Fischer 給出了動態規劃解法 [7]：

$$E(i,j) = \min\{E(i-1,j)+1,\; E(i,j-1)+1,\; E(i-1,j-1)+\mathbf{1}[s_i \neq u_j]\},$$

其中 $E(i,0)=i$、$E(0,j)=j$，s（長度 n）與 u（長度 m）為兩個字串，答案是 $E(n,m)$。它的結構和 DTW 完全相同，只是把「拉長」換成「插入」、「刪除」與「替換」三種操作。成本可以加權，例如讓相鄰方向之間的替換比相反方向便宜。

```
import numpy as np

def edit_distance(s, t):
    """Levenshtein distance: unit-cost insertions, deletions, substitutions."""
    n, m = len(s), len(t)
    D = np.zeros((n + 1, m + 1), dtype=int)
    D[:, 0] = np.arange(n + 1)
    D[0, :] = np.arange(m + 1)
    for i in range(1, n + 1):
        for j in range(1, m + 1):
            sub = 0 if s[i - 1] == t[j - 1] else 1
            D[i, j] = min(D[i - 1, j] + 1,          # delete s[i-1]
                          D[i, j - 1] + 1,          # insert t[j-1]
```

```
                        D[i - 1, j - 1] + sub)    # match or substitute
    return D[n, m]

# 4-direction chain codes (0=right, 1=up, 2=left, 3=down) of three outlines
square = "000111222333"           # 3 x 3 square, traced counter-clockwise
square_bump = "00011121232333"   # the same square with a one-pixel bump on top
rectangle = "000001222223"       # a flat 5 x 1 rectangle
for name, code in [("square + bump", square_bump), ("rectangle", rectangle)]:
    print(f"edit distance(square, {name}) = {edit_distance(square, code)}")
```

有凸起的正方形和正方形相差 2 次編輯（插入一個「上」與一個「下」）；扁平的長方形相差 4 次。以這個距離在字串上做最近鄰分類，就是 k-NN 的結構式版本。鏈碼取決於起點與旋轉；把它們正規化（例如選擇使字串最小的起點，並改用相鄰方向之間的差）是表徵設計的一部分。

圖與圖匹配

當部件之間的連接不只是一條鏈時，圖形就成為一張**帶屬性的圖**：節點是部件（筆畫、區域、原子），帶有大小或顏色等屬性；邊是關係（相接、在上方、鍵結）。辨識於是成為**圖匹配**：找出兩張圖的節點之間的對應關係。精確匹配（圖同構或子圖同構）不容許任何差異，一般而言計算困難；容錯匹配，例如**圖編輯距離**，以節點與邊的插入、刪除、替換推廣字串編輯距離，計算代價更高。Conte、Foggia、Sansone 與 Vento 綜述了圖形識別中三十年的圖匹配研究，從精確演算法到不精確演算法及其應用 [8]；Foggia、Percannella 與 Vento 則以十年間的圖匹配與圖學習方法更新了這幅圖像 [9]。

圖模型描述相依關係

白話版。圖模型是一張「哪些變數直接影響哪些變數」的圖。這張圖告訴你龐大的聯合機率如何拆成小的局部片段，而小片段正是讓計算變得可行的關鍵。

有向無環圖（貝氏網路）把變數 $z_1, \dots, z_M$ 的聯合分布分解為

$$p(z_1, \dots, z_M) = \prod_{i=1}^{M} p(z_i \mid \mathrm{pa}(z_i)),$$

其中 $\mathrm{pa}(z_i)$ 是 z_i 在圖中的父節點 [2]。**無向**圖（馬可夫隨機場）在它的團（clique，即兩兩相連的節點子集）C 上使用非負的位能函數 ψ_C：

$$p(z) = \frac{1}{Z} \prod_{C} \psi_C(z_C), \qquad Z = \sum_{z} \prod_{C} \psi_C(z_C).$$

位能是相容程度的分數，不一定是機率。**配分函數** Z 把乘積正規化；它對所有聯合組態求和，往往是最難計算的量。

條件獨立要依圖的規則判讀（有向圖用 d-分離，無向圖用圖分離），不能只看哪裡沒有箭頭。一個經典的意外是：兩個獨立的原因，在以它們共同的結果為條件後就會變得相依（「解釋消除」，explaining away）[2]。**因子圖**把變數與因子畫成兩種節點，讓局部的訊息傳遞變得明確；在樹

上，訊息傳遞（sum-product 演算法）能精確算出所有邊際分布，而下文的 HMM 演算法就是它在鏈上的特例 [2]。

在影像標記中，一種常見的無向模型是由一元資料項與成對平滑項組成的**能量**：

$$E(z; x) = \sum_{i} U_i(z_i; x) + \lambda \sum_{(i,j)\in\mathcal{E}} \mathbf{1}[z_i \neq z_j],$$

其中 z_i 是像素 i 的標籤，U_i 是在影像 x 下該標籤的成本（例如像素分類器輸出的負對數機率），$\mathcal{E}$ 是相鄰像素對的集合，$\lambda \geq 0$ 是相鄰像素標籤改變的代價（Potts 模型）。對應的分布是 $p(z \mid x) \propto \exp(-E(z;x))$。它鼓勵相鄰標籤一致，能清除雜訊，但 λ 太大時也會抹掉細長的結構。在一般的格點上精確最小化很困難，近似推論見 PR 08。

隱藏式馬可夫模型

白話版。一個系統在幾個隱藏狀態之間移動（靜音、母音、子音；走路、跑步），每個時間步發出一個只取決於當下狀態的觀測。你看得到觀測，卻永遠看不到狀態。HMM 讓你能計算一串觀測有多可能、它大概是由哪些隱藏狀態產生的，以及如何從資料中學出模型的機率。

模型

有 K 個隱藏狀態的 HMM，包含初始機率 $\pi_j = p(z_1 = j)$、**轉移矩陣** $A_{ij} = p(z_t = j \mid z_{t-1} = i)$，以及**發射**機率（或密度）$B_j(x_t) = p(x_t \mid z_t = j)$。狀態序列 $z_{1:T} = (z_1, \dots, z_T)$ 與觀測 $x_{1:T}$ 的聯合機率為

$$p(z_{1:T}, x_{1:T}) = \pi_{z_1} B_{z_1}(x_1) \prod_{t=2}^{T} A_{z_{t-1}, z_t} B_{z_t}(x_t).$$

其中內建了兩個假設：隱藏狀態是一階馬可夫的（未來只透過當下狀態依賴過去），且每個觀測只取決於當下狀態。觀測本身不是馬可夫的：把隱藏狀態積分掉後，觀測之間會出現長程相依。Rabiner 的教學文章至今仍是標準入門，它把 HMM 的計算整理成三個問題 [10]：**評估**（計算 $p(x_{1:T})$）、**解碼**（找出隱藏狀態）與**學習**（估計 π, A, B）。圖 4 對照 HMM 的有向圖與稍後討論的 CRF。

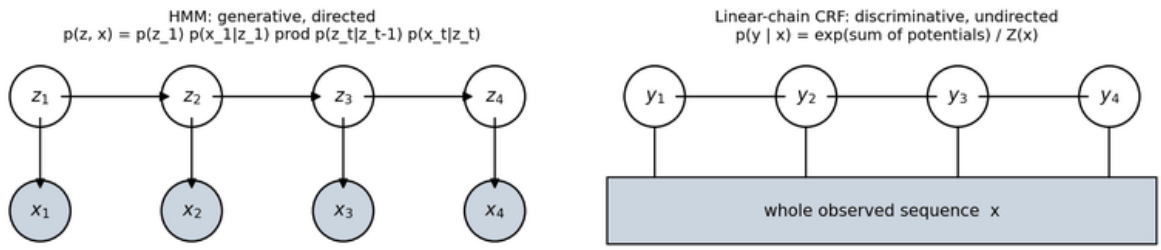


圖 4 — *HMM* 與線性鏈 *CRF*。*HMM* 是狀態與觀測的有向生成式模型；*CRF* 是以整條觀測序列為條件的標籤無向模型。

評估：前向演算法

對全部 K^T 條狀態序列加總聯合機率，在實際的 T 下毫無可能。定義**前向變數** $\alpha_t(j) = p(x_{1:t}, z_t = j)$。由於 z_t 總結了過去能告訴未來的一切，它滿足

$$\alpha_1(j) = \pi_j B_j(x_1), \qquad \alpha_t(j) = B_j(x_t) \sum_{i=1}^{K} \alpha_{t-1}(i) A_{ij},$$

且 $p(x_{1:T}) = \sum_j \alpha_T(j)$。成本是 $O(TK^2)$，而不是 $O(TK^T)$。由於 α_t 是許多機率的乘積，長序列會造成數值下溢。標準解法是每一步都正規化：記錄 $c_t = \sum_j \tilde{\alpha}_t(j)$ 與 $\hat{\alpha}_t = \tilde{\alpha}_t / c_t$，其中 $\tilde{\alpha}_t$ 是未正規化的更新值。這時 $\hat{\alpha}_t(j) = p(z_t = j \mid x_{1:t})$ 恰好就是**濾波**狀態機率，$c_t = p(x_t \mid x_{1:t-1})$，而且

$$\log p(x_{1:T}) = \sum_{t=1}^{T} \log c_t.$$

評估直接給出一個分類器：每個類別訓練一個 HMM（例如每個口說詞一個），新序列就歸給「其 HMM 概似乘以先驗」最高的類別。

平滑：前向–後向演算法

濾波只用過去與現在計算 $p(z_t \mid x_{1:t})$，這是線上系統在時間 t 所能知道的一切。**平滑**計算 $p(z_t \mid x_{1:T})$，連之後的觀測也一起使用。後向遞迴 $\beta_t(i) = p(x_{t+1:T} \mid z_t = i)$，

$$\beta_T(i) = 1, \qquad \beta_t(i) = \sum_{j=1}^{K} A_{ij}\, B_j(x_{t+1})\, \beta_{t+1}(j),$$

和前向變數結合，得到 $\gamma_t(j) = p(z_t = j \mid x_{1:T}) \propto \alpha_t(j)\beta_t(j)$ [2], [10]。加入尺度化時，每個後向步驟除以 c_{t+1}，於是 $\gamma_t = \hat{\alpha}_t \odot \hat{\beta}_t$（逐元素乘積）恰好成立。

解碼：Viterbi 演算法

我們常常想要的是機率最高的單一狀態序列 $\arg\max_{z_{1:T}} p(z_{1:T} \mid x_{1:T})$。Viterbi 演算法 [11] 把前向遞迴中的加總換成取最大值。在對數空間中，

$$\delta_1(j) = \log \pi_j + \log B_j(x_1), \qquad \delta_t(j) = \log B_j(x_t) + \max_i \{\delta_{t-1}(i) + \log A_{ij}\},$$

其中 $\delta_t(j)$ 是在時間 t 結束於狀態 j 的最佳路徑的對數機率。記錄使其最大的前一個狀態 $\psi_t(j) = \arg\max_i \{\cdots\}$，選出最佳的最終狀態，再沿指標往回追溯。成本同樣是 $O(TK^2)$。

機率最高的路徑，不等於在每個時間點各自挑選平滑機率 $\gamma_t(j)$ 最高的狀態。逐步挑選最大化的是「預期正確的狀態數」，但可能把模型認定不能相連的狀態串在一起。下面的程式在一個狀態必須依 $0 \to 1 \to 2 \to 0$ 循環的模型上展示這一點，其中轉移 $0 \to 2$ 的機率為零。

```
import itertools
import numpy as np
```

```
pi = np.array([1 / 3, 1 / 3, 1 / 3])            # initial state probabilities
A = np.array([[0.5, 0.5, 0.0],                  # cyclic model: 0 -> 1 -> 2 -> 0
              [0.0, 0.5, 0.5],
              [0.5, 0.0, 0.5]])
B = np.array([[0.7, 0.3],                       # P(symbol | state)
              [0.5, 0.5],
              [0.3, 0.7]])
obs = np.array([1, 0, 1, 0, 0])
T, K = len(obs), len(pi)

# Forward pass with scaling: alpha[t] = p(z_t | x_1..t), c[t] = p(x_t | x_1..t-1).
alpha, c = np.zeros((T, K)), np.zeros(T)
alpha[0] = pi * B[:, obs[0]]
c[0] = alpha[0].sum(); alpha[0] /= c[0]
for t in range(1, T):
    alpha[t] = B[:, obs[t]] * (alpha[t - 1] @ A)
    c[t] = alpha[t].sum(); alpha[t] /= c[t]
log_lik = np.log(c).sum()

# Backward pass (scaled) and smoothed marginals gamma[t] = p(z_t | x_1..T).
beta = np.ones((T, K))
for t in range(T - 2, -1, -1):
    beta[t] = A @ (B[:, obs[t + 1]] * beta[t + 1]) / c[t + 1]
gamma = alpha * beta

# Viterbi in log space (log 0 = -inf marks forbidden transitions).
with np.errstate(divide="ignore"):
    lA, lB, lpi = np.log(A), np.log(B), np.log(pi)
delta = lpi + lB[:, obs[0]]
back = []
for t in range(1, T):
    cand = delta[:, None] + lA
    back.append(cand.argmax(axis=0))
    delta = cand.max(axis=0) + lB[:, obs[t]]
path = [int(delta.argmax())]
for b in reversed(back):
    path.append(int(b[path[-1]]))
path = path[::-1]

# Brute force over all K**T paths as an independent check.
def joint(z):
    p = pi[z[0]] * B[z[0], obs[0]]
    for t in range(1, T):
        p *= A[z[t - 1], z[t]] * B[z[t], obs[t]]
    return p
paths = list(itertools.product(range(K), repeat=T))
assert np.isclose(log_lik, np.log(sum(joint(z) for z in paths)))
assert tuple(path) == max(paths, key=joint)

marginal = gamma.argmax(axis=1)
print("log p(x):", round(log_lik, 4))
print("Viterbi path:         ", path, " p =", round(joint(path), 5))
print("per-step argmax gamma:", marginal.tolist(),
      " p =", round(joint(marginal), 5))
```

尺度化的前向計算給出 $\log p(x) = -3.5449$，而概似與 Viterbi 路徑都和全部 $3^5 = 243$ 條路徑的暴力列舉一致。Viterbi 回傳 $1 \to 1 \to 2 \to 0 \to 0$，聯合機率為 0.00179。逐步挑選機率最高的狀態得到 $2, 0, 2, 0, 0$，其中包含被禁止的跳躍 $0 \to 2$，在模型下的機率恰好為零。哪一種解碼才對取決於損失：逐步準確率偏好邊際分布；要得到合法、一致的序列則需要 Viterbi。

圖 5 在一條較長的模擬序列上展示同樣的計算：濾波機率只對過去做出反應；平滑機率使用整條序列，因而更有把握；Viterbi 路徑除了觀測有歧義的地方之外，都跟得上真實狀態。

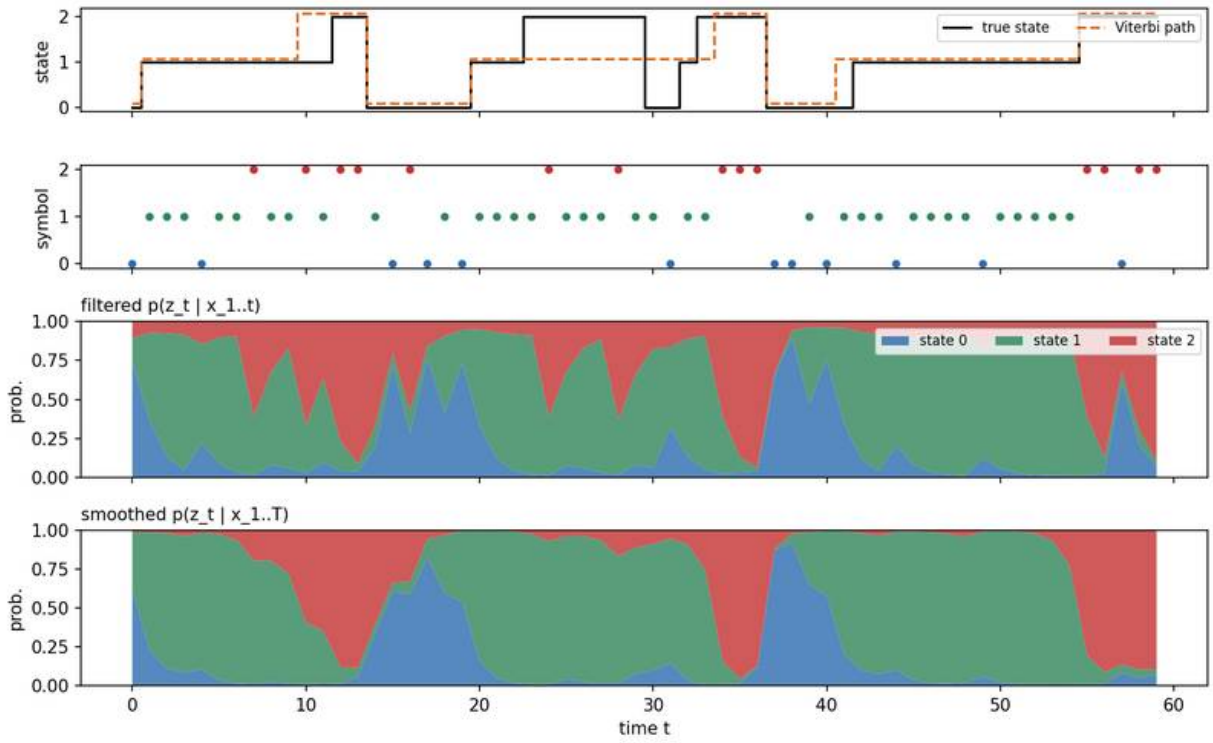


圖 *5* — 模擬的 *3* 狀態循環 *HMM*。濾波使用到時間 t 為止的觀測；平滑使用整條序列。在這次執行中，*Viterbi* 路徑在 *78%* 的時間步與真實狀態相符，逐步挑選平滑機率為 *77%*，逐步挑選濾波機率為 *73%*。

關於評估的提醒。如果系統必須線上做決策，請回報濾波（或固定延遲）下的表現。平滑結果用到了決策當下尚不存在的觀測，會高估部署後系統的能力。

學習：Baum–Welch

當隱藏狀態沒有標註時，參數以期望值最大化（EM）估計，在 HMM 中稱為 **Baum–Welch** 演算法 [12]。E 步驟執行前向–後向，得到預期的狀態佔用 $\gamma_t(j)$ 與預期的轉移 $\xi_t(i,j) = p(z_{t-1} = i, z_t = j \mid x_{1:T})$。M 步驟重新估計

$$\hat{A}_{ij} = \frac{\sum_{t=2}^{T} \xi_t(i,j)}{\sum_{t=2}^{T} \sum_{k} \xi_t(i,k)}, \qquad \hat{\pi}_j = \gamma_1(j),$$

並以 $\gamma_t(j)$ 加權的觀測估計發射參數（例如高斯發射的加權平均與共變異數）。每次迭代都不會降低概似，但結果取決於初始化，可能停在局部極大值；PR 07 介紹一般形式的 EM。如果隱藏狀態是連續的，而且一切都是線性高斯，同樣的前向遞迴就成為 Kalman 濾波器 [13]。以學習得到的向量取代離散狀態的神經遞迴模型，見 ML 04。

條件隨機場

白話版。 HMM 建模的是狀態如何產生觀測，這迫使它對觀測做簡單的假設。如果你只需要「給定觀測下的標籤」，CRF 就直接建模這件事，而且可以使用整個輸入的任何特徵，不論彼此是否重疊。

HMM 是**生成式**的：它建模 $p(z, x)$，包括觀測如何產生。這要求發射模型 $B_j(x_t)$ 完整描述 x_t；當特徵很多、彼此重疊又相依時（字詞本身、它的字尾、大小寫、相鄰的字詞……），這就很彆扭。Lafferty、McCallum 與 Pereira 提出條件隨機場，直接建模給定觀測 x 時標籤序列 $y = (y_1, \dots, y_T)$ 的**條件**分布 [14]。**線性鏈 CRF** 為

$$p(y \mid x) = \frac{1}{Z(x)} \exp\Big(\sum_{t=1}^{T} \sum_{k} \theta_k f_k(y_{t-1}, y_t, x, t) \Big), \qquad Z(x) = \sum_{y'} \exp\Big(\sum_{t,k} \theta_k f_k(y'_{t-1}, y'_t, x, t) \Big),$$

其中 f_k 是特徵函數，可以看當前與前一個標籤，以及輸入序列的任何部分；θ_k 是學習得到的權重；$Z(x)$ 針對這一個輸入，對所有標籤序列 y' 做正規化 [14], [15]。把 HMM 寫成這種形式，它就是特例：特徵只有「前一個標籤、當前標籤」與「當前標籤、當前觀測」兩類指示函數。CRF 可以加入任意的輸入特徵，而不必建模它們的分布。

推論使用和 HMM 相同的動態規劃：前向演算法（以 log-sum-exp 取代加總）計算 $Z(x)$ 與邊際分布，Viterbi 找出最佳標記。訓練時最大化條件對數概似 $\sum_n \log p(y^{(n)} \mid x^{(n)})$，通常加上 ℓ_2 懲罰。對線性鏈 CRF 而言，這個目標是凹函數，因此沒有不好的局部最佳解；它的梯度是「觀測到的特徵計數減去預期的特徵計數」，預期值由前向–後向算出 [15]。下面的程式以暴力法檢查 $\log Z$ 的前向計算。

```
import itertools
import numpy as np

rng = np.random.default_rng(0)
T, K = 5, 3
U = rng.normal(size=(T, K))       # unary scores  theta . f(y_t, x, t)
P = rng.normal(size=(K, K))       # transition scores  theta . f(y_{t-1}, y_t)

def score(y):
    return U[np.arange(T), y].sum() + sum(P[y[t - 1], y[t]] for t in range(1, T))

# log Z by the forward recursion in log space (log-sum-exp instead of sum).
log_alpha = U[0].copy()
for t in range(1, T):
    m = log_alpha[:, None] + P
    log_alpha = U[t] + np.logaddexp.reduce(m, axis=0)
log_Z = np.logaddexp.reduce(log_alpha)
```

```
brute = np.logaddexp.reduce([score(np.array(y))
                             for y in itertools.product(range(K), repeat=T)])
print("log Z forward:", round(log_Z, 6), " brute force:", round(brute, 6))
assert np.isclose(log_Z, brute)
```

兩者都得到 $\log Z = 6.173403$。這裡的分數是代替 $\sum_k \theta_k f_k$ 的隨機數；在真正的 CRF 中，它們來自 x 的特徵，在現代系統中則來自神經網路。

在影像上，同樣的想法得到像素上的格狀 CRF：一元項來自像素分類器，成對項類似上面的 Potts 能量。在有迴圈的格點上已無法做精確推論，因此使用近似方法（PR 08）。

現代觀點

模板、對齊、文法與馬可夫鏈，在深度網路旁邊看起來有點過時。事實上它們都延續了下來，有時就藏在網路內部。以下是綜述文獻與近期研究中的主要脈絡。

模板比對。 Hashemi 等人 [4] 回顧了比對的基本概念、不變特徵與可變形模型等進展，以及包括醫學影像分析在內的應用。他們的結論是：模板比對與其他辨識步驟結合時效果最好，例如在 NCC 之後接一個穩健的決策階段。當外觀受控時，以相關為基礎的比對快速、簡單又精確，至今仍是工業檢測、影像對位的標準做法，也是追蹤器的基本組件。當外觀會變化時，比對就轉移到學習得到的特徵空間：「模板」現在是一張特徵圖，相關運算在深度特徵之間進行，但運算本身是一樣的。

時間序列分類與 DTW。 Bagnall 等人的「bake off」[16] 在同一個框架中重新實作了 18 個時間序列分類演算法，並在 85 個 UCR 資料集上各做 100 次重抽樣比較。只有九個演算法顯著優於兩個基準（其中之一是 DTW 的 1-NN），而一個由轉換式集成組成的方法（COTE）勝過其他所有方法。Middlehurst、Schäfer 與 Bagnall 的「bake off redux」[17] 在擴充為 112 個資料集的資料庫與 30 個新問題上重做這項比較，在分類體系中加入以卷積、以特徵與深度學習為基礎的類別，並發現兩個較新的演算法 MultiROCKET+Hydra 與 HIVE-COTE 2 顯著優於其餘方法。Ismail Fawaz 等人 [18] 以統一的分類體系回顧了時間序列分類中的深度學習，並在同一個開放框架中於 97 個資料集上訓練了 8,730 個網路。與此同時，DTW 本身成為一個基本組件：soft-DTW [19] 把 DTW 遞迴中的最小值換成平滑的 soft-minimum，使它可微分，因此能作為訓練網路的損失，也能用來平均時間序列。

神經網路內部的對齊。連結時序分類（connectionist temporal classification, CTC）[20] 讓遞迴網路在沒有逐幀對齊的情況下輸出標籤序列，方法是以一個和 HMM 密切相關的前向–後向遞迴，對所有對齊方式加總。動態規劃留了下來，發射模型則變成了網路。

僅線上版：連結時序分類（*CTC*）— *https://blog.shyandram.dev/zh-tw/learn/pattern-recognition/templates-structured-models/notes/連結時序分類 ctc/*

結合深度特徵的 CRF。 Sutton 與 McCallum 的教學 [15] 至今仍是 CRF 建模、推論與訓練的標準參考。在視覺領域，具高斯邊位能的全連接 CRF 讓所有像素對上的平均場推論變得有效率 [21]，並成為語意分割常用的精修步驟；Zheng 等人接著把這個平均場推論寫成一個遞迴網路層，和 CNN 一起端到端訓練 [22]。在語言領域，Lample 等人的 BiLSTM-CRF [23] 在雙向 LSTM 上加了一層 CRF 輸出，用於命名實體辨識，讓標籤轉移被明確建模，而特徵則由學習得到。在這兩個

領域，後來具有大感受野或注意力機制的架構往往已建模足夠的脈絡，使明確的 CRF 幫助有限；但在標籤約束很重要的場合，結構化輸出層仍在使用。

結構與圖。 Conte 等人 [8] 與 Foggia 等人 [9] 的圖匹配綜述，描繪了從精確與容錯匹配，走向圖核、圖嵌入與以學習為基礎之方法的轉變。它們的現代後代，圖神經網路，以訊息傳遞學習節點與整張圖的表徵，正是因子圖局部計算的直接推廣。

狀態空間模型。「一個精簡的隱藏狀態總結了過去」這個概念，是 HMM 與 Kalman 濾波器 [13] 的核心，如今在序列建模中再度成為焦點。Mamba [24] 這類選擇性狀態空間模型使用與輸入相關的線性遞迴，執行時間與序列長度成線性關係，被提出作為長序列上注意力機制的替代方案。

未解問題。 DTW 與編輯距離每次比較都是平方成本，所以在大型資料集中搜尋必須依靠界限與剪枝 [6]。圖編輯距離的精確計算仍然昂貴。HMM 與 CRF 的結果取決於圖中內建的獨立性假設；當假設錯誤時，平滑可能很有自信地抹掉短暫事件。而對任何線上系統來說，濾波與平滑評估之間的區別都很容易弄錯。

重點整理

- NCC 減去平均並除以範數，因此對亮度偏移與正的對比變化不變；SSD 與單純的相關則否。兩者都無法處理旋轉、尺度或形變。
- DTW 以動態規劃在 $O(nm)$ 內找出成本最低的單調對齊；Sakoe–Chiba 帶限制病態扭曲並降低成本，但必須比真實的時間差更寬。
- 結構式辨識把圖形描述為字串、文法或圖；編輯距離以和 DTW 相同的動態規劃模式比較字串。
- 圖模型把聯合分布分解成局部片段；在鏈與樹上的訊息傳遞既精確又有效率。
- HMM 演算法：前向（$p(x)$ 與濾波）、前向–後向（平滑）、Viterbi（最佳路徑）、Baum–Welch（學習），成本都是 $O(TK^2)$。以尺度化或對數運算避免下溢。
- Viterbi 路徑與逐步最可能狀態回答的是不同的問題；後者可能是不可能出現的序列。
- CRF 直接建模 $p(y \mid x)$，可使用任意重疊的輸入特徵，與 HMM 共用推論演算法，而在線性鏈情況下訓練目標是凹函數。

練習

1. **NCC 與負片。**證明把區塊的每個像素換成 $ax_i + b$（$a > 0$）時 NCC 不變。$a < 0$ 時會怎樣？為什麼 NCC 無法告訴你 b 的值？

   提示

   減去平均會完全消除 b，所以關於它的資訊一點都不剩。因子 a 出現在分子，並以 $|a|$ 出現在分母；$a > 0$ 時互相抵消，$a < 0$ 時正負號翻轉（完美匹配的負片得分為 -1）。

2. **手算 DTW。**以成本 $|x_i - t_j|$ 計算 $x = (0, 1, 2)$ 與 $t = (0, 0, 1, 2)$ 的 DTW 距離。寫出 3×4 的表格與最佳路徑。逐點距離是多少？為什麼在這裡它根本沒有定義？

提示

最佳路徑把 x_1 同時對應到兩個 0，接著 $x_2 \leftrightarrow t_3$、$x_3 \leftrightarrow t_4$，總成本為 0。逐點比較需要等長，而這兩條序列不等長；DTW 自然地處理不等長的情況。

3. **被禁止的轉移**。自己構造一個兩狀態或三狀態的 HMM，把其中一個轉移機率設為零，並找出一串觀測，使 γ_t 的逐步 argmax 成為不可能的狀態序列。用上面的程式驗證。

提示

你需要兩個相鄰時間步，它們各自最可能的狀態之間恰好是被禁止的轉移。發射機率接近 0.5（有歧義）的對稱或循環模型很容易做到；接著檢查 `joint(marginal)` 是否為 0。

4. **線上與離線**。某活動辨識器以錄好的序列、根據平滑狀態機率回報 95% 的準確率。它將在手機上執行，每秒都必須做決定。這個數字包含了什麼資訊上的優勢？你應該改為回報什麼？

提示

平滑用到了每次決策之後的觀測。應回報濾波準確率，或應用可以容忍之延遲下的固定延遲平滑結果（例如延遲 2 秒的決策可以使用到 $t+2$ 為止的觀測）。

5. **從 HMM 到 CRF**。把 HMM 的 $\log p(z, x)$ 寫成特徵函數乘以權重的總和，即線性鏈 CRF 的形式。你得到哪些特徵與權重？在詞性標記中，CRF 可以加入哪些 HMM 難以使用的額外特徵？

提示

使用指示特徵 $\mathbf{1}[z_{t-1} = i, z_t = j]$，權重為 $\log A_{ij}$；$\mathbf{1}[z_t = j, x_t = v]$，權重為 $\log B_j(v)$；以及 $\mathbf{1}[t = 1, z_1 = j]$，權重為 $\log \pi_j$。CRF 可以加入「下一個字以 -ly 結尾」或「當前的字首字母大寫」這類特徵，它們和字詞本身重疊且相依。

6. **加權編輯距離**。修改 `edit_distance`，使相差 90 度的兩個方向替換成本為 1、相差 180 度為 2。重新計算鏈碼範例中的距離。排名會改變嗎？

提示

替換成本是方向碼之間的循環差 $\min(|a-b|, 4-|a-b|)$。有凸起的正方形只需要插入，所以距離仍是 2；請檢查長方形最便宜的編輯序列用到了哪些替換，以判斷它的距離是否改變。

參考文獻

1. S. Theodoridis and K. Koutroumbas, *Pattern Recognition*, 4th ed., Academic Press, 2009, Chapters 8 and 9. publisher page
2. C. M. Bishop, *Pattern Recognition and Machine Learning*, Springer, 2006, Sections 8.1–8.4 and 13.1–13.3. official book page and PDF

3. J. P. Lewis, “Fast Normalized Cross-Correlation,” *Vision Interface*, 1995. paper
4. N. S. Hashemi, R. B. Aghdam, A. S. B. Ghiasi, and P. Fatemi, “Template Matching Advances and Applications in Image Analysis,” arXiv:1610.07231, 2016. arXiv
5. H. Sakoe and S. Chiba, “Dynamic Programming Algorithm Optimization for Spoken Word Recognition,” *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 26, no. 1, pp. 43–49, 1978. doi:10.1109/TASSP.1978.1163055
6. T. Rakthanmanon, B. Campana, A. Mueen, G. Batista, B. Westover, Q. Zhu, J. Zakaria, and E. Keogh, “Searching and Mining Trillions of Time Series Subsequences under Dynamic Time Warping,” *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 262–270, 2012. doi:10.1145/2339530.2339576
7. R. A. Wagner and M. J. Fischer, “The String-to-String Correction Problem,” *Journal of the ACM*, vol. 21, no. 1, pp. 168–173, 1974. doi:10.1145/321796.321811
8. D. Conte, P. Foggia, C. Sansone, and M. Vento, “Thirty Years of Graph Matching in Pattern Recognition,” *International Journal of Pattern Recognition and Artificial Intelligence*, vol. 18, no. 3, pp. 265–298, 2004. doi:10.1142/S0218001404003228
9. P. Foggia, G. Percannella, and M. Vento, “Graph Matching and Learning in Pattern Recognition in the Last 10 Years,” *International Journal of Pattern Recognition and Artificial Intelligence*, vol. 28, no. 1, 1450001, 2014. doi:10.1142/S0218001414500013
10. L. R. Rabiner, “A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition,” *Proceedings of the IEEE*, vol. 77, no. 2, pp. 257–286, 1989. doi:10.1109/5.18626
11. A. J. Viterbi, “Error Bounds for Convolutional Codes and an Asymptotically Optimum Decoding Algorithm,” *IEEE Transactions on Information Theory*, vol. 13, no. 2, pp. 260–269, 1967. doi:10.1109/TIT.1967.1054010
12. L. E. Baum, T. Petrie, G. Soules, and N. Weiss, “A Maximization Technique Occurring in the Statistical Analysis of Probabilistic Functions of Markov Chains,” *The Annals of Mathematical Statistics*, vol. 41, no. 1, pp. 164–171, 1970. doi:10.1214/aoms/1177697196
13. R. E. Kalman, “A New Approach to Linear Filtering and Prediction Problems,” *Journal of Basic Engineering*, vol. 82, no. 1, pp. 35–45, 1960. doi:10.1115/1.3662552
14. J. D. Lafferty, A. McCallum, and F. C. N. Pereira, “Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data,” *Proceedings of the 18th International Conference on Machine Learning (ICML)*, pp. 282–289, 2001. paper page
15. C. Sutton and A. McCallum, “An Introduction to Conditional Random Fields,” *Foundations and Trends in Machine Learning*, vol. 4, no. 4, pp. 267–373, 2012. doi:10.1561/2200000013
16. A. Bagnall, J. Lines, A. Bostrom, J. Large, and E. Keogh, “The Great Time Series Classification Bake Off: A Review and Experimental Evaluation of Recent Algorithmic Advances,” *Data Mining and Knowledge Discovery*, vol. 31, pp. 606–660, 2017. doi:10.1007/s10618-016-0483-9
17. M. Middlehurst, P. Schäfer, and A. Bagnall, “Bake Off Redux: A Review and Experimental Evaluation of Recent Time Series Classification Algorithms,” *Data Mining and Knowledge Discovery*, vol. 38, pp. 1958–2031, 2024. doi:10.1007/s10618-024-01022-1
18. H. Ismail Fawaz, G. Forestier, J. Weber, L. Idoumghar, and P.-A. Muller, “Deep Learning for Time Series Classification: A Review,” *Data Mining and Knowledge Discovery*, vol. 33, pp. 917–963, 2019. doi:10.1007/s10618-019-00619-1
19. M. Cuturi and M. Blondel, “Soft-DTW: A Differentiable Loss Function for Time-Series,” *ICML*, 2017. arXiv:1703.01541

20. A. Graves, S. Fernández, F. Gomez, and J. Schmidhuber, “Connectionist Temporal Classification: Labelling Unsegmented Sequence Data with Recurrent Neural Networks,” *Proceedings of the 23rd International Conference on Machine Learning* (*ICML*), pp. 369–376, 2006. doi:10.1145/1143844.1143891
21. P. Krähenbühl and V. Koltun, “Efficient Inference in Fully Connected CRFs with Gaussian Edge Potentials,” *NIPS*, 2011. arXiv:1210.5644
22. S. Zheng, S. Jayasumana, B. Romera-Paredes, V. Vineet, Z. Su, D. Du, C. Huang, and P. H. S. Torr, “Conditional Random Fields as Recurrent Neural Networks,” *ICCV*, 2015. arXiv:1502.03240
23. G. Lample, M. Ballesteros, S. Subramanian, K. Kawakami, and C. Dyer, “Neural Architectures for Named Entity Recognition,” *NAACL*, 2016. arXiv:1603.01360
24. A. Gu and T. Dao, “Mamba: Linear-Time Sequence Modeling with Selective State Spaces,” arXiv:2312.00752, 2023. arXiv

PR 07・分群、混合模型與 EM

以 k-means 與 k-means++、階層式連結、DBSCAN 與譜分群為無標籤資料分組；推導高斯混合模型的 EM；用 BIC 與有效性指標選擇群數。

網站版本: https://blog.shyandram.dev/zh-tw/learn/pattern-recognition/clustering-mixtures/

你將學到

- 分群演算法實際上在最佳化什麼，以及為什麼特徵、尺度與距離的選擇，在任何演算法執行之前就已決定了答案。
- Lloyd 的 k-means 如何運作、為何一定收斂卻只到局部最佳，以及 k-means++ 播種如何給出可證明的 $O(\log K)$ 保證。
- 階層式分群（single、complete、average 與 Ward 連結）、DBSCAN 與譜分群各自如何定義「一群」。
- 如何從頭推導高斯混合模型期望值最大化（expectation–maximization, EM）的 E 步與 M 步、為何 EM 不會讓似然下降，以及它會在哪裡失敗。
- 如何用 BIC 與 AIC 選擇群數，以及如何解讀內部指標（silhouette、Davies–Bouldin、Calinski–Harabasz）、外部指標（ARI、NMI）與穩定度。

先看全貌

到目前為止，這個系列中的每一筆訓練樣本都附有標籤。分群（clustering）把標籤拿掉，問的是：資料裡有沒有自然的群組？這個問題的答案從來不只由資料本身決定。同一批影像可以依物體、依顏色、依背景，或依拍攝的相機來分組；每一種分法在它自己的相似性定義下都是「對的」。因此，一個分群演算法其實是兩件事：精確說明我們指的是哪一種相似性，再加上一個在該定義下尋找最佳分組的搜尋程序。

> **用最簡單的話說**
>
> 把一大盒鈕扣倒在地上，你想把它們整理成幾堆。你可以按顏色、按大小，或按扣眼數量來分。每一種分法都很整齊，但哪一種有用，要看你接下來要做什麼。分群方法就是一套「堆鈕扣的規則」。**k-means** 說：「選幾個堆的中心，每顆鈕扣放到最近的那一堆。」**DBSCAN** 說：「擠在一起的鈕扣屬於同一堆，孤零零的就當作剩下的。」**高斯混合模型**則說：「每一堆都是一團模糊的雲，一顆鈕扣可以有 70% 把握屬於這團、30% 屬於那團。」

為什麼這對辨識很重要：分群可以建立視覺詞彙（把局部影像描述子用 k-means 量化成「視覺詞」）、初始化與壓縮模型（向量量化、調色盤）、在已標註的類別中找出子類別、找出離群值、分割影像（像素分群，見 IP 10），而在現代自監督學習中，還能產生用來訓練深度網路的偽標籤。高斯混合模型也是**潛在變數模型**（latent-variable model）的標準範例，而 EM 是我們在 PR 08 會遇到的許多推論演算法的原型。

本章依循 Theodoridis 與 Koutroumbas 書中的分群部分（第 11–16 章）[1]，以及 Bishop 第 9 章的混合模型與 EM [2]。我們會整理主要的演算法家族，完整推導其中最重要的兩個，最後討論如何判斷一個分群結果好不好。

相似性、尺度與「群」的意義

白話版。在選演算法之前，先決定什麼叫做「近」。換了特徵或單位，群就會跟著改變。

精確版。大多數方法都從特徵向量 $x, x' \in \mathbb{R}^D$ 之間的相異度 $d(x, x')$ 出發。常見選擇有

$$d_2(x,x') = \|x - x'\|_2, \qquad d_M(x,x') = \sqrt{(x-x')^\top S^{-1}(x-x')}, \qquad d_{\cos}(x,x') = 1 - \frac{x^\top x'}{\|x\|\,\|x'\|},$$

其中 d_2 是歐氏距離；d_M 是以正定矩陣 S（通常是共變異數估計）定義的馬氏距離（Mahalanobis distance）；$d_{\cos}$ 是餘弦相異度，它忽略向量長度，常用於深度網路的嵌入向量（embedding）。

由此有三個實務上的後果。

1. **尺度是一個建模決策**。若一個座標以公釐計、另一個以公里計，歐氏距離就會被前者主導。把每個特徵標準化為單位變異數可以消除這個問題，但也會抹掉真正有意義的離散程度差異。做分類時可以交給驗證集決定；分群沒有標籤，這個選擇只能來自領域知識。
2. **硬指派與軟指派**。硬分割把每個點 x_n 指派到恰好一個群 $z_n \in \{1, \dots, K\}$。軟指派則保存隸屬程度 $r_{nk} \geq 0$，且 $\sum_k r_{nk} = 1$。在高斯混合模型中，這些是後驗機率；在模糊分群 [1] 中，它們是由目標函數定義的隸屬度，不應被解讀為機率。
3. **群不是類別**。群的編號是事後任意附上的。這一次的「第 3 群」下一次可能變成「第 1 群」，因此任何與類別標籤的比較都必須對排列不變，而這正是後面介紹的 ARI 與 NMI 的性質。

Theodoridis 與 Koutroumbas 把演算法分成序列式、階層式、以目標函數為基礎（k-means、混合模型、模糊分群），以及以圖或密度為基礎等家族 [1]。我們大致依照這個順序。

K-means 與 Lloyd 演算法

白話版。選 K 個中心。重複兩個步驟：把每個點交給最近的中心，再把每個中心移到它所屬點的平均位置。直到沒有任何變化為止。

目標函數

令 one-hot 指標 $z_{nk} \in \{0, 1\}$（若第 n 點屬於第 k 群則 $z_{nk} = 1$，且 $\sum_k z_{nk} = 1$），中心為 $\mu_1, \dots, \mu_K \in \mathbb{R}^D$。k-means 最小化群內平方和

$$J(z, \mu) = \sum_{n=1}^{N} \sum_{k=1}^{K} z_{nk} \, \|x_n - \mu_k\|^2.$$

其中 N 是點的數量，D 是維度。J 也稱為 *inertia* 或失真（distortion）。

兩個交替步驟

Lloyd 演算法 [3]（原本是為脈衝碼調變中的訊號量化而發展）以**區塊座標下降**（block coordinate descent）最小化 J：

- **指派步驟**。固定 μ。每一項只牽涉一個 z_n，因此對 $k = \arg\min_j \|x_n - \mu_j\|^2$ 設 $z_{nk} = 1$。
- **更新步驟**。固定 z。J 對每個 μ_k 都是凸二次函數；令梯度為零，

$$\frac{\partial J}{\partial \mu_k} = -2 \sum_n z_{nk}(x_n - \mu_k) = 0 \quad \Longrightarrow \quad \mu_k = \frac{\sum_n z_{nk} x_n}{\sum_n z_{nk}},$$

也就是目前屬於第 k 群各點的平均。「k-means」這個名稱由 MacQueen [4] 提出；他的版本在每個點到來後立即更新中心：$\mu_k \leftarrow \mu_k + (x_n - \mu_k)/N_k$，其中 N_k 是第 k 群到目前為止的點數。

為什麼會收斂，又為什麼只是局部

每一步只會讓 J 下降或維持不變：指派步驟為目前的 μ 挑出最佳的 z，更新步驟為目前的 z 挑出最佳的 μ。把 N 個點分成 K 組的分割方式是有限的，而且（只要平手時採用固定規則）演算法不會回到 J 更高的分割，因此在有限次迭代後一定停止。停下來的固定點是**局部**最小值：任何單一步驟都無法再改善的分割。不同的初始中心可能導致差異很大的固定點。

實作上有兩個細節。**空群**（沒有任何點被指派）沒有平均值；常見作法是保留舊中心，或在目前誤差最大的點重新播種。另外每次迭代需要 $O(NKD)$ 次距離計算，這促成了後面的小批次版本。

k-means 的假設

由於目標是各點到中心的歐氏距離平方和，兩群之間的邊界就是兩個中心的垂直平分線：k-means 把空間切成 **Voronoi 區域**。它適合緊密、大致呈球形且大小相近的群；它會切開細長的群、把靠近大群的小群併進去，也無法沿著彎曲形狀分群（圖 1，中間）。

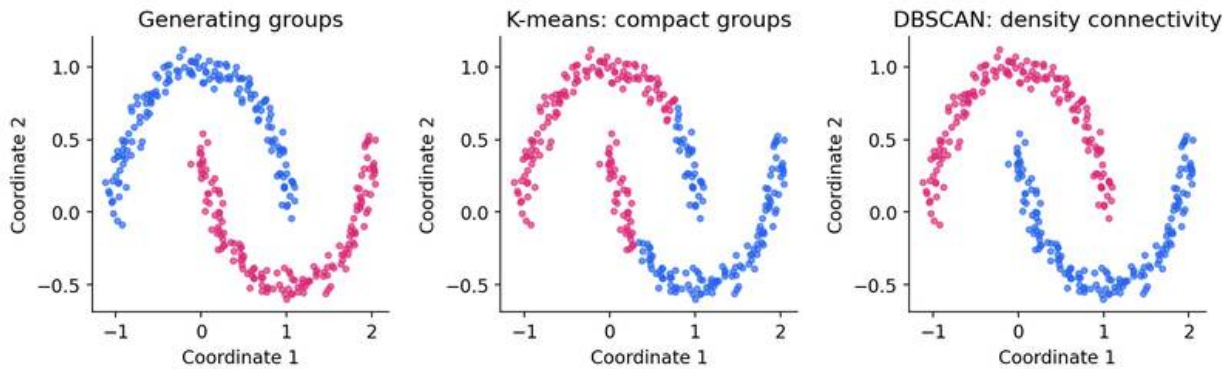


圖 *1* — 同一份雙月形資料，依生成標籤（左）、*k-means*（中）與 *DBSCAN*（右）分組。*k-means* 與 *DBSCAN* 都在沒有標籤的情況下擬合；顏色只是任意的群編號。

k-means++ 播種

白話版。先隨機選一個中心，之後每個新中心都盡量選離現有中心遠的點，但保留隨機性，免得離群值每次都被選中。

Arthur 與 Vassilvitskii [5] 提出 **D² 播種**：

1. 從資料中均勻隨機選出第一個中心。
2. 對每個點計算 $D(x_n)$，即它到目前已選中心中最近者的距離。
3. 以機率 $D(x_n)^2 / \sum_m D(x_m)^2$ 選出 x_n 作為下一個中心。
4. 重複步驟 2–3 直到選滿 K 個中心，再執行 Lloyd 演算法。

他們的主要定理指出，光是播種本身，在期望值意義下就已達到

$$\mathbb{E}[J] \le 8(\ln K + 2)\, J_{\mathrm{OPT}},$$

其中 J_{OPT} 是最佳的 k-means 目標值；之後的 Lloyd 迭代只會讓 J 更低。均勻隨機播種沒有這樣的保證。這個上界是最壞情況的敘述；實務上的好處主要表現在災難性的結果變少了。函式庫會把 k-means++ 與多次重新啟動結合（scikit-learn 的 `n_init`），保留 J 最低的一次。

程式：從頭實作 k-means++ 與 Lloyd

```
import numpy as np
from sklearn.datasets import make_blobs
from sklearn.cluster import KMeans

def kmeans_pp_init(X, k, rng):
    """D^2 seeding: each new center is drawn with probability proportional
    to the squared distance to the nearest center chosen so far."""
    centers = [X[rng.integers(len(X))]]
    for _ in range(k - 1):
        d2 = np.min(((X[:, None, :] - np.array(centers)[None]) ** 2).sum(-1), axis=1)
        centers.append(X[rng.choice(len(X), p=d2 / d2.sum())])
    return np.array(centers)

def lloyd(X, centers, max_iter=100):
    history = []
    for _ in range(max_iter):
        d2 = ((X[:, None, :] - centers[None]) ** 2).sum(-1)   # N x K
        z = d2.argmin(axis=1)                                 # assignment step
        history.append(d2[np.arange(len(X)), z].sum())        # objective J
        new = np.array([X[z == k].mean(axis=0) if np.any(z == k) else centers[k]
                        for k in range(len(centers))])        # update step
        if np.allclose(new, centers):
            break
        centers = new
    return centers, z, history

X, _ = make_blobs(n_samples=600, centers=4, cluster_std=1.8, random_state=7)
rng = np.random.default_rng(0)
best = None
```

```
for restart in range(10):                       # several seeds, keep the best J
    c, z, hist = lloyd(X, kmeans_pp_init(X, 4, rng))
    if best is None or hist[-1] < best[2][-1]:
        best = (c, z, hist)
c, z, hist = best
print("J per iteration:", np.round(hist, 1))
assert all(a >= b - 1e-9 for a, b in zip(hist, hist[1:]))   # monotone
ref = KMeans(n_clusters=4, n_init=10, random_state=0).fit(X)
print("from scratch J:", round(hist[-1], 2), " sklearn inertia:", round(ref.inertia_, 2))

# single runs (no restarts): uniform random seeding vs D^2 seeding, 15 clusters
X2, _ = make_blobs(n_samples=1500, centers=15, cluster_std=0.6, random_state=2)
for name, init in [("random", lambda r: X2[r.choice(len(X2), 15, replace=False)]),
                   ("k-means++", lambda r: kmeans_pp_init(X2, 15, r))]:
    finals = [lloyd(X2, init(np.random.default_rng(s)))[2][-1] for s in range(20)]
    print(f"{name:9s} final J over 20 runs: median {np.median(finals):7.1f}, "
          f"worst {np.max(finals):7.1f}")
```

輸出（NumPy 2.0、scikit-learn 1.6）：

```
J per iteration: [5865.6 3633.4 3625. ]
from scratch J: 3625.0  sklearn inertia: 3625.0
random    final J over 20 runs: median  1305.1, worst  4750.8
k-means++ final J over 20 runs: median  1147.5, worst  1525.2
```

從頭寫的版本與 scikit-learn 達到相同的目標值，而且 J 單調下降。在 15 群的問題上，均勻播種有時會把兩個種子放進同一團、另一團卻一個也沒有；Lloyd 演算法無法修正這種情況，最差的一次大約差了三倍。圖 2 顯示兩種播種各自最差的一次。

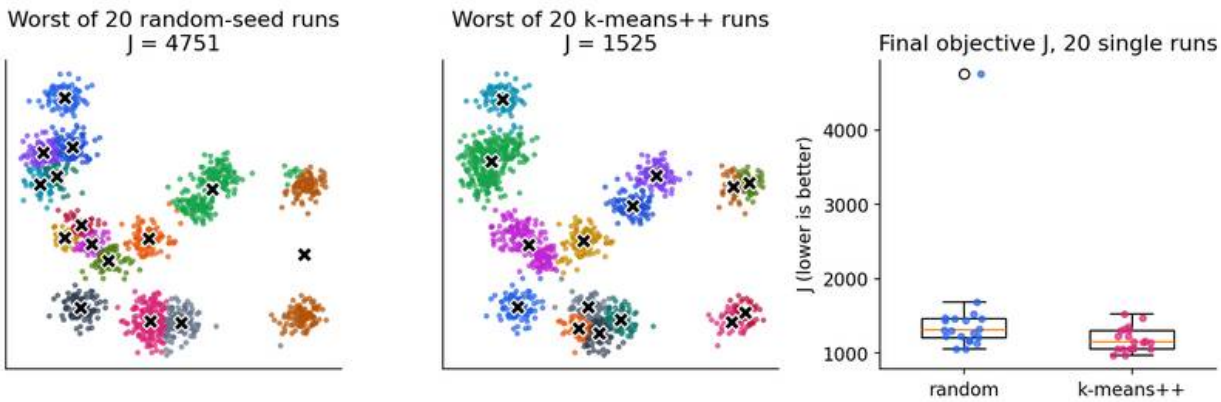


圖 2 — 播種很重要。*15* 團合成資料，每種播種規則各執行 *20* 次單一 *Lloyd*（不重新啟動）。隨機播種會讓某些執行卡在合併與切開的團；D^2（*k-means++*）播種的目標值較低、變異也較小。黑色叉號為最終中心。

小批次 k-means

面對數百萬個點時，Sculley 的小批次 k-means [6] 用小的隨機批次來更新中心：每個中心朝被指派給它的批次點移動，每個中心的學習率依 $1/(\text{已見過的點數})$ 遞減。它以略高的最終目標值換取大幅加速，在 scikit-learn 中為 `MiniBatchKMeans`，是建立大型視覺詞彙時的常見選擇。

階層式分群

白話版。一開始每個點自成一群，接著反覆把最近的兩群合併。合併紀錄形成一棵樹；在某個高度切開它，就得到各群。

聚合式演算法與連結方式

聚合式（agglomerative）分群需要定義兩個群 A 與 B 之間的距離。四種經典的**連結**（linkage）為

$$
\begin{aligned}
d_{\text{single}}(A,B) &= \min_{a\in A,\,b\in B} d(a,b),\\
d_{\text{complete}}(A,B) &= \max_{a\in A,\,b\in B} d(a,b),\\
d_{\text{average}}(A,B) &= \frac{1}{|A|\,|B|}\sum_{a\in A}\sum_{b\in B} d(a,b),\\
\Delta_{\text{Ward}}(A,B) &= \frac{|A|\,|B|}{|A|+|B|}\,\|\bar{a}-\bar{b}\|^2,
\end{aligned}
$$

其中 $|A|$ 是 A 中的點數，$\bar{a},\bar{b}$ 是兩群的平均。Ward 準則 [7] 是合併 A 與 B 所造成的群內平方和 J 的增量，所以 Ward 聚合是一種由下而上、貪婪地讓 J 保持很小的方法，目標與 k-means 相同。

每種連結都有自己的個性（圖 3）：

- **Single linkage** 只要任何兩點夠近就合併。它能沿著任意形狀走，但一條細細的點「橋」就可能把兩群串在一起。它等價於切斷最小生成樹（minimum spanning tree）中最長的邊，這也是 [1] 中的圖論方法之一。
- **Complete linkage** 要求所有點對都夠近，會產生緊密且直徑相近的群，對離群值敏感。
- **Average linkage** 介於兩者之間。
- **Ward** 產生緊密、大小相近、大致呈球形的群，很像 k-means。

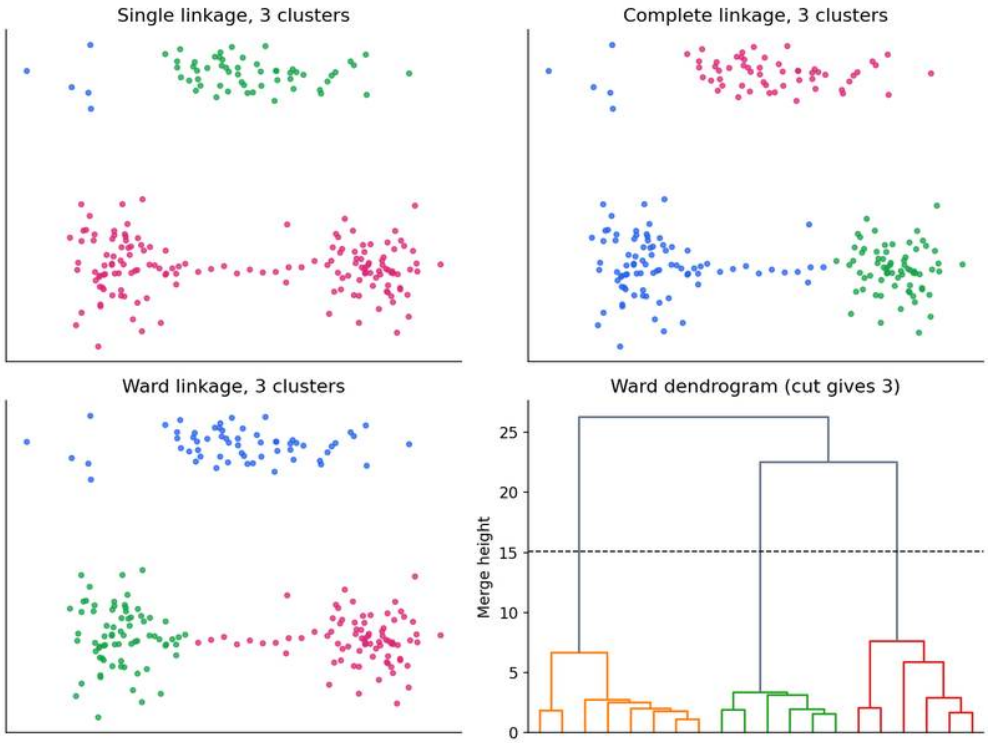


圖 3 — 同一份資料、三種連結。*Single linkage* 沿著細橋串接，把兩團併成一群，並把一群浪費在四個零散點上；*complete* 與 *Ward* 連結則讓兩團保持分開，把細橋分給兩側。右下：*Ward* 樹狀圖，在虛線處切開得到三群。

樹狀圖

合併歷程是一棵二元樹，稱為**樹狀圖**（dendrogram）；每個接點的高度就是合併當下的連結值。在高度 h 切開樹，就得到該層的分割；相鄰合併高度之間若有很大的落差，往往暗示一個自然的群數。不過切割高度仍然是一個選擇。

實務上有兩個限制：合併一旦發生就不會撤銷，早期的錯誤會一路傳下去；標準演算法需要儲存所有點對距離，記憶體為 $O(N^2)$，使它只適用於數萬個點以內。分裂式（由上而下）階層方法也存在，但較少使用。

以密度為基礎的分群：DBSCAN

白話版。一群就是一塊擁擠的區域。擁擠處的點讓群長大；空曠處的點被標為雜訊。

DBSCAN（Density-Based Spatial Clustering of Applications with Noise）[8] 有兩個參數：半徑 ε 與點數 MinPts。令 $N_\varepsilon(x) = \{x' : d(x, x') \le \varepsilon\}$ 為 x 的 ε 鄰域（包含 x 本身）。則

- 若 $|N_\varepsilon(x)| \ge$ MinPts，x 是**核心點**（core point）；
- 若 x 不是核心點，但落在某個核心點的 ε 鄰域內，它是**邊界點**（border point）；

- 其餘的點都是**雜訊**（noise）。

若存在一串核心點，相鄰兩者距離都不超過 ε，最後抵達某點的鄰域，則該點從起始核心點密度可達。一群就是彼此密度相連的點所構成的極大集合。DBSCAN 能找出任意形狀的群、不需要事先指定 K，而且明確拒絕指派孤立的點。

如何選 ε。原論文建議使用排序後的 **k 距離圖**：對每個點計算它到第 k 個最近鄰的距離（$k = \text{MinPts}$），排序後在曲線急遽上升的「膝點」附近選 ε（圖 4 左）。圖中膝點左側那些距離很大的點，就是雜訊的候選。scikit-learn 的 `min_samples` 會把點本身算進去，與上面的定義一致。

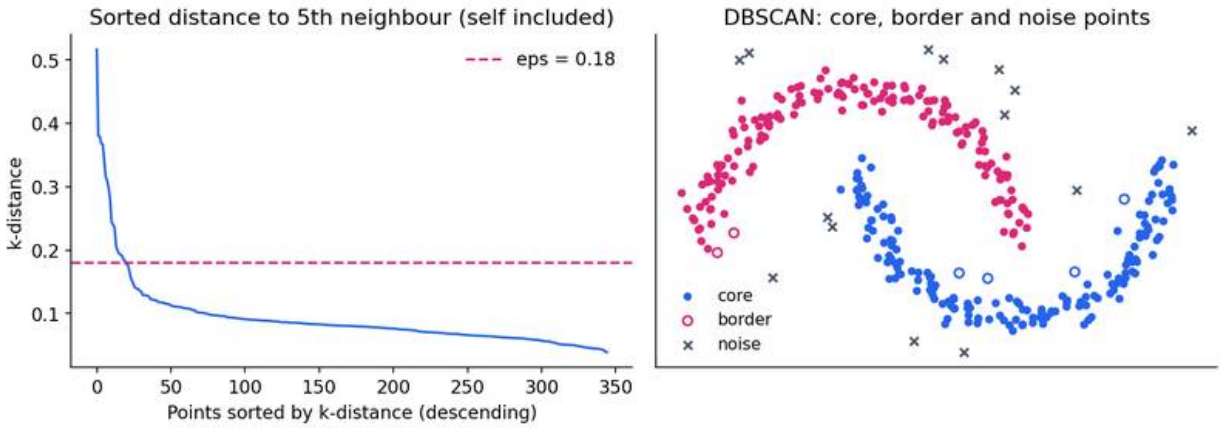


圖 *4* — 在雙月形資料加上 *25* 個均勻離群點上執行 *DBSCAN*。左：排序後的第 *5* 近鄰距離曲線；所選的 $\varepsilon = 0.18$ 位於曲線開始上升之處。右：核心點（實心）、邊界點（空心）以及被判為雜訊的點（叉號）。

限制。單一 ε 意味著單一密度水準。若一群很密、另一群很稀，就沒有任何 ε 能同時把兩者分開。在高維空間中距離會趨於集中，鄰域點數失去對比。**HDBSCAN** [9] 針對第一個問題，建立涵蓋所有密度水準的階層，並取出最穩定的群，以「最小群大小」取代 ε；scikit-learn 中提供 `HDBSCAN`。

一頁看懂譜分群

白話版。建一張把相近點連起來的圖，再找出切斷最少強連結的切法。

給定親和矩陣 W（$W_{ij} \geq 0$，相似的點數值大，例如 k 近鄰圖或 $W_{ij} = \exp(-\|x_i - x_j\|^2 / 2\sigma^2)$）與對角度數矩陣 $D_{ii} = \sum_j W_{ij}$，一種正規化圖拉普拉斯矩陣為

$$L_{\text{sym}} = I - D^{-1/2} W D^{-1/2}.$$

若圖有 K 個連通分量，L_{sym} 的特徵值 0 重數為 K，其特徵向量指出各分量。若圖接近不連通，最小 K 個特徵值對應的特徵向量仍近似指出各群。譜分群把這些特徵向量排成矩陣的行、把每一列正規化，再對各列執行 k-means [10]。結果可以沿著在原空間中會打敗 k-means 的彎曲形狀走，因為特徵向量衡量的是連通性，而不是到某個中心的距離。

圖本身就是模型的一部分。鄰居數或頻寬 σ 都必須選擇；一張意外不連通的圖（例如有一個孤立節點）會產生一個主導結果的平凡特徵向量。von Luxburg 的教學文章 [10] 解釋了圖切割的詮釋（ratio cut 與 normalized cut），也提供如何選圖的實務建議。

程式：同一種幾何，不同的假設

下面的雙月形資料是帶標籤生成的，但標籤只在事後用來以調整蘭德指數（adjusted Rand index, ARI；1 表示在重新編號下完全一致，約 0 表示與隨機相當）評分。

```
import numpy as np
from sklearn.datasets import make_moons
from sklearn.neighbors import NearestNeighbors
from sklearn.cluster import (KMeans, AgglomerativeClustering, DBSCAN,
                             HDBSCAN, SpectralClustering)
from sklearn.mixture import GaussianMixture
from sklearn.metrics import adjusted_rand_score

X, truth = make_moons(n_samples=320, noise=0.06, random_state=5)

# k-distance curve for choosing DBSCAN's eps (k = min_samples)
dist, _ = NearestNeighbors(n_neighbors=5).fit(X).kneighbors(X)
kdist = np.sort(dist[:, -1])                    # 5th neighbour, counting the point itself
print("k-distance quantiles 50/90/99%:", np.round(np.quantile(kdist, [0.5, 0.9, 0.99]), 3))

models = {
    "k-means": KMeans(n_clusters=2, n_init=10, random_state=5),
    "single linkage": AgglomerativeClustering(n_clusters=2, linkage="single"),
    "Ward linkage": AgglomerativeClustering(n_clusters=2, linkage="ward"),
    "Gaussian mixture": GaussianMixture(n_components=2, n_init=3, random_state=5),
    "DBSCAN": DBSCAN(eps=0.18, min_samples=5),
    "HDBSCAN": HDBSCAN(min_cluster_size=15),
    "spectral": SpectralClustering(n_clusters=2, affinity="nearest_neighbors",
                                   n_neighbors=20, assign_labels="kmeans",
                                   random_state=5),
}
for name, model in models.items():
    labels = model.fit_predict(X)
    print(f"{name:17s} ARI = {adjusted_rand_score(truth, labels):.3f}"
          f"   noise points = {int((labels == -1).sum())}")
```

```
k-distance quantiles 50/90/99%: [0.079 0.11  0.144]
k-means           ARI = 0.260   noise points = 0
single linkage    ARI = 1.000   noise points = 0
Ward linkage      ARI = 0.242   noise points = 0
Gaussian mixture  ARI = 0.488   noise points = 0
DBSCAN            ARI = 1.000   noise points = 0
HDBSCAN           ARI = 0.988   noise points = 2
spectral          ARI = 1.000   noise points = 0
```

圖 1 顯示其中兩個結果。以中心或橢圓為基礎的方法（k-means、Ward、兩個成分的高斯混合）把每個月形切成兩半；以連通性為基礎的方法（single linkage、DBSCAN、HDBSCAN、譜分群）則找回兩個月形。所選的 $\varepsilon = 0.18$ 高於 k 距離的第 99 百分位數，因此沒有任何月形上的點被當成雜訊。鄰居太少的 k 近鄰圖可能不連通，此時 scikit-learn 會警告譜嵌入可能不可靠；這裡用 20 個鄰居就避開了這個問題。但這並不代表以連通性為基礎的方法「比較好」：在相互重疊的高斯團上，single linkage 會把一切串在一起，而 k-means 勝出。比較方法時必須改變幾何、密度、雜訊與維度。

高斯混合模型與 EM 演算法

白話版。假裝每個點的產生方式是：先擲一顆 K 面骰子選一團雲，再從那團高斯雲中取樣。我們永遠看不到骰子。EM 會先為每個點猜測各團雲產生它的可能性（E 步），再用這些猜測當作權重重新擬合每團雲（M 步），然後重複。

模型

具有 K 個成分的高斯混合模型（Gaussian mixture model, GMM）的密度為

$$p(x \mid \theta) = \sum_{k=1}^{K} \pi_k \, \mathcal{N}(x \mid \mu_k, \Sigma_k), \qquad \pi_k \geq 0, \quad \sum_{k=1}^{K} \pi_k = 1,$$

其中 π_k 是**混合權重**（成分 k 的先驗機率），$\mu_k \in \mathbb{R}^D$ 是其平均，Σ_k 是其 $D \times D$ 共變異數矩陣，$\theta = \{\pi_k, \mu_k, \Sigma_k\}_{k=1}^{K}$ 收集所有參數，且

$$\mathcal{N}(x \mid \mu, \Sigma) = \frac{1}{(2\pi)^{D/2} |\Sigma|^{1/2}} \exp\Big(-\tfrac{1}{2} (x-\mu)^{\top} \Sigma^{-1} (x-\mu) \Big).$$

引入一個**潛在**的 one-hot 變數 z_n，令 $p(z_{nk} = 1) = \pi_k$、$p(x_n \mid z_{nk} = 1) = \mathcal{N}(x_n \mid \mu_k, \Sigma_k)$。把 z_n 加總消去，就回到混合密度。給定資料 $X = \{x_1, \dots, x_N\}$，對數似然為

$$\ln p(X \mid \theta) = \sum_{n=1}^{N} \ln \sum_{k=1}^{K} \pi_k \, \mathcal{N}(x_n \mid \mu_k, \Sigma_k).$$

「和的對數」把所有參數綁在一起，令梯度為零沒有封閉解。EM 是繞過這個困難的標準方法 [11]、[2]。

步驟 1：完整資料對數似然

如果我們知道指派 $Z = \{z_n\}$，(X, Z) 的對數似然就是

$$\ln p(X, Z \mid \theta) = \sum_{n=1}^{N} \sum_{k=1}^{K} z_{nk} \Big[\ln \pi_k + \ln \mathcal{N}(x_n \mid \mu_k, \Sigma_k) \Big].$$

現在對數只作用在單一高斯上，問題拆成 K 個各自獨立的加權高斯擬合：很容易。

步驟 2（E 步）：完整資料對數似然的期望值

我們不知道 Z，但在目前參數 θ^{old} 下，我們知道它的後驗。由貝氏定理，

$$r_{nk} \equiv p(z_{nk}=1 \mid x_n, \theta^{\text{old}}) = \frac{\pi_k \, \mathcal{N}(x_n \mid \mu_k, \Sigma_k)}{\sum_{j=1}^{K} \pi_j \, \mathcal{N}(x_n \mid \mu_j, \Sigma_j)},$$

在 θ^{old} 下計算。r_{nk} 是成分 k 對點 n 的**責任值**（responsibility），每一列加總為 1。由於完整資料對數似然對 z_{nk} 是線性的，在這個後驗下取期望，只要把 z_{nk} 換成 r_{nk}：

$$Q(\theta, \theta^{\text{old}}) = \sum_{n}\sum_{k} r_{nk}\Big[\ln \pi_k - \tfrac{D}{2}\ln 2\pi - \tfrac{1}{2}\ln|\Sigma_k| - \tfrac{1}{2}(x_n-\mu_k)^{\top}\Sigma_k^{-1}(x_n-\mu_k)\Big].$$

步驟 3（M 步）：以封閉形式最大化 Q

定義**有效點數** $N_k = \sum_n r_{nk}$。

混合權重。只有 $\sum_n \sum_k r_{nk}\ln\pi_k$ 與 π 有關，且受限於 $\sum_k \pi_k = 1$。引入拉格朗日乘數 λ，

$$\frac{\partial}{\partial \pi_k}\Big[\sum_{n,k} r_{nk}\ln\pi_k + \lambda\Big(\sum_k \pi_k - 1\Big)\Big] = \frac{N_k}{\pi_k} + \lambda = 0 \;\Rightarrow\; \pi_k = -\frac{N_k}{\lambda}.$$

對 k 加總得 $\lambda = -N$，所以 $\pi_k = N_k/N$。（限制 $\pi_k \geq 0$ 自動成立。）

平均。對 μ_k 的梯度為 $\sum_n r_{nk}\Sigma_k^{-1}(x_n-\mu_k)$。令其為零並乘上 Σ_k，

$$\mu_k = \frac{1}{N_k}\sum_n r_{nk}\, x_n.$$

共變異數。令 $\Lambda_k = \Sigma_k^{-1}$，則 $-\tfrac{1}{2}\ln|\Sigma_k| = \tfrac{1}{2}\ln|\Lambda_k|$，二次項為 $-\tfrac{1}{2}\operatorname{tr}\big(\Lambda_k(x_n-\mu_k)(x_n-\mu_k)^{\top}\big)$。利用 $\partial \ln|\Lambda|/\partial\Lambda = \Lambda^{-1}$，以及對對稱矩陣 A 有 $\partial \operatorname{tr}(\Lambda A)/\partial\Lambda = A^{\top}$，

$$\frac{\partial Q}{\partial \Lambda_k} = \frac{1}{2}\sum_n r_{nk}\Big[\Sigma_k - (x_n-\mu_k)(x_n-\mu_k)^{\top}\Big] = 0 \;\Rightarrow\; \Sigma_k = \frac{1}{N_k}\sum_n r_{nk}(x_n-\mu_k)(x_n-\mu_k)^{\top}.$$

這些正是單一高斯熟悉的最大似然公式（見 PR 02），只是每個點都依其責任值加權。EM 交替執行 E 步與 M 步，直到對數似然不再改善（圖 5）。

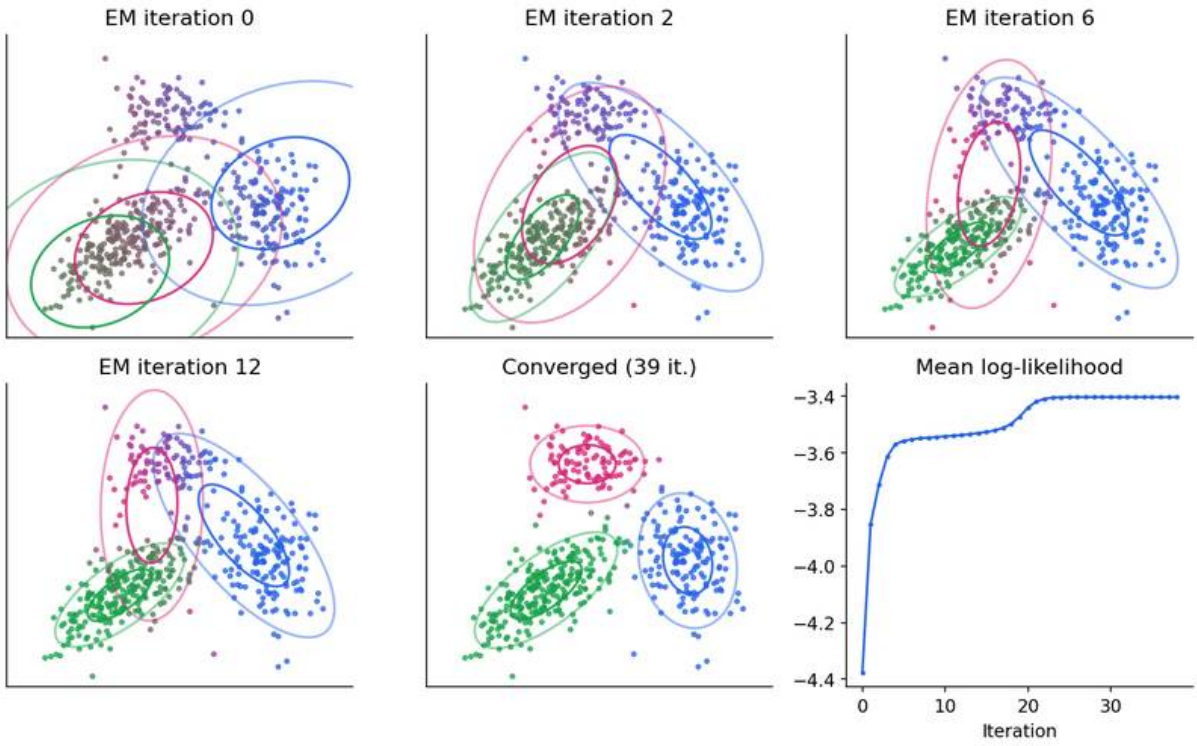


圖 5 — 三成分高斯混合的 *EM*。各成分從隨機資料點出發、使用整體共變異數；點的顏色依責任值混合各成分的顏色。右下：平均對數似然從不下降。

為什麼 EM 不會讓似然下降

對潛在指派上的任何分布 $q(Z)$，

$$\ln p(X \mid \theta) = \underbrace{\sum_{Z} q(Z) \ln \frac{p(X, Z \mid \theta)}{q(Z)}}_{\mathcal{L}(q,\theta)} + \underbrace{\sum_{Z} q(Z) \ln \frac{q(Z)}{p(Z \mid X, \theta)}}_{\mathrm{KL}(q \,\|\, p(Z \mid X,\theta)) \,\geq\, 0}.$$

因此 $\mathcal{L}(q,\theta)$ 是對數似然的下界，稱為**證據下界**（evidence lower bound, ELBO）。E 步令 $q(Z) = p(Z \mid X, \theta^{\text{old}})$，使 KL 項為零、下界在 θ^{old} 處貼合。M 步對 θ 最大化 $\mathcal{L}$；由於 $\mathcal{L}(q,\theta) = Q(\theta, \theta^{\text{old}}) +$ 常數，這等同於最大化 Q。新的似然至少等於新的下界，新的下界至少等於舊的下界，而舊的下界等於舊的似然。因此 $\ln p(X \mid \theta^{\text{new}}) \geq \ln p(X \mid \theta^{\text{old}})$ [11]、[2]。當精確後驗難以計算時，我們把 q 限制在一個可處理的分布族中，就得到**變分推論**（variational inference），這是 PR 08 的主題。

單調不代表全域最佳，也不保證安全收斂：在適當正則條件下，EM 可趨近駐點；不受限的高斯混合模型也可能反而走向下述奇異解。實務上的實作會先以 k-means 初始化，並多次重新啟動。

奇異解與如何避免

不受限 GMM 的似然是**無界**的。把某個成分的平均正好放在一個資料點上，$\mu_j = x_n$，並令 $\Sigma_j = \sigma^2 I$。這一項對 $p(x_n)$ 的貢獻為 $\pi_j (2\pi\sigma^2)^{-D/2}$，當 $\sigma \to 0$ 時趨於無限大，而其他成分仍讓其餘點的似然保持有限。因此不受限的最大似然估計不一定存在，EM 本身也不會阻止崩塌。應使用正的共變異數下限（scikit-learn 的 `reg_covar` [26]）、明確的變異數限制，或適當先驗。**只改成對角或球形共變異數並不能解決問題**：正在崩塌的 $\sigma^2 I$ 原本就同時符合這兩種形式。較簡單的共變異數能減少參數，但仍須防止零變異；重新啟動成分只是操作上的處理，不是原目標函數有界的證明。

可識別性與標籤交換

把成分的編號互換，密度不變，因此有 $K!$ 個等價的最大值。「成分 2」沒有固有意義，要把成分編號與類別標籤比較，必須先有一個配對規則。

k-means 是 EM 的極限

對所有 k 固定 $\pi_k = 1/K$ 與 $\Sigma_k = \epsilon I$。責任值變成

$$r_{nk} = \frac{\exp(-\|x_n - \mu_k\|^2 / 2\epsilon)}{\sum_j \exp(-\|x_n - \mu_j\|^2 / 2\epsilon)}.$$

當 $\epsilon \to 0$，距離最小的那一項主導，因此最近的中心 $r_{nk} \to 1$，其餘為 0：E 步變成 k-means 的指派步驟，μ_k 的 M 步變成 k-means 的更新 [2]。所以一般的 GMM 是在 k-means 上加入軟指派、不等大小（π_k）與橢圓形狀（Σ_k）的延伸。

程式：從頭實作 EM，並與 scikit-learn 對照

```
import numpy as np
from scipy.special import logsumexp
from scipy.stats import multivariate_normal
from sklearn.mixture import GaussianMixture

rng = np.random.default_rng(3)
true_means = np.array([[0.0, 0.0], [4.0, 1.0], [1.0, 4.0]])
true_covs = [np.array([[1.0, 0.6], [0.6, 0.8]]),
             np.array([[0.5, -0.2], [-0.2, 1.2]]),
             np.array([[0.7, 0.0], [0.0, 0.3]])]
counts = [250, 150, 100]
X = np.vstack([rng.multivariate_normal(m, S, n)
               for m, S, n in zip(true_means, true_covs, counts)])
N, D = X.shape
K, reg = 3, 1e-6                              # reg: small covariance floor

# initialise with K random points, shared data covariance, equal weights
mu = X[rng.choice(N, K, replace=False)]
```

```
Sigma = np.array([np.cov(X.T) for _ in range(K)])
pi = np.full(K, 1.0 / K)
loglik = []
for it in range(200):
    # E step: log of pi_k N(x_n | mu_k, Sigma_k), normalised with log-sum-exp
    log_joint = np.column_stack([np.log(pi[k]) +
                                 multivariate_normal(mu[k], Sigma[k]).logpdf(X)
                                 for k in range(K)])
    log_norm = logsumexp(log_joint, axis=1)
    R = np.exp(log_joint - log_norm[:, None])     # responsibilities r_nk
    loglik.append(log_norm.sum())
    # M step: closed-form weighted updates
    Nk = R.sum(axis=0)
    pi = Nk / N
    mu = (R.T @ X) / Nk[:, None]
    for k in range(K):
        diff = X - mu[k]
        Sigma[k] = (R[:, k, None] * diff).T @ diff / Nk[k] + reg * np.eye(D)
    if it > 0 and loglik[-1] - loglik[-2] < 1e-8:
        break

assert np.all(np.diff(loglik) > -1e-6)           # EM never decreases log-likelihood
print("iterations:", len(loglik), " final mean log-lik:", round(loglik[-1] / N, 4))
print("weights:", np.round(np.sort(pi), 3))
gm = GaussianMixture(3, covariance_type="full", n_init=5, random_state=0).fit(X)
print("sklearn mean log-lik:", round(gm.score(X), 4))
print("sklearn weights:", np.round(np.sort(gm.weights_), 3))
```

```
iterations: 40  final mean log-lik: -3.4032
weights: [0.202 0.299 0.499]
sklearn mean log-lik: -3.4033
sklearn weights: [0.203 0.3   0.497]
```

E 步在對數空間中計算：高維的密度值會向下溢位，而 `logsumexp` 能穩定地計算 $\ln \sum_k e^{a_k}$。估得的權重與生成比例 $0.5, 0.3, 0.2$ 及 scikit-learn 的解相符；微小差異來自 scikit-learn 自己的停止容忍度。

選擇群數

白話版。群越多，對訓練資料的擬合總是越好。模型選擇準則對每個多出來的參數收取代價，最佳的 K 在擬合與代價之間取得平衡。

混合模型的 BIC 與 AIC

對一個最大化似然為 $\hat{L}$、有 p 個自由參數、N 個資料點的模型，

$$\mathrm{BIC} = -2\ln \hat{L} + p \ln N, \qquad \mathrm{AIC} = -2\ln \hat{L} + 2p,$$

越小越好。貝氏資訊準則（Bayesian information criterion, BIC）[12] 來自對數邊際似然（模型證據，見 PR 08）的大樣本近似；赤池資訊準則（Akaike information criterion, AIC）[13] 估計的是預期的預測對數損失。由於 $N \geq 8$ 時 $\ln N > 2$，BIC 對複雜度的懲罰較重，傾向選擇較少的成分。

對 D 維、完整共變異數的 GMM，

$$p = \underbrace{(K-1)}_{\text{權重}} + \underbrace{KD}_{\text{平均}} + \underbrace{K\,\frac{D(D+1)}{2}}_{\text{共變異數}}.$$

在高維時共變異數占了大部分：當 $D = 128$（類似 SIFT 的描述子），每個完整共變異數有 8,256 個自由數值，這就是為什麼大型描述子通常使用對角共變異數。因此共變異數的形式也是模型選擇的一部分。

```
import numpy as np
from sklearn.datasets import make_blobs
from sklearn.mixture import GaussianMixture

X, _ = make_blobs(n_samples=600, centers=4, cluster_std=1.0, random_state=7)
X = X @ np.array([[1.0, 0.0], [0.6, 0.5]])        # shear: make clusters elliptical
results = []
for cov in ["spherical", "diag", "full"]:
    for K in range(1, 9):
        gm = GaussianMixture(K, covariance_type=cov, n_init=3, random_state=0).fit(X)
        results.append((gm.bic(X), gm.aic(X), cov, K))
for bic, aic, cov, K in sorted(results)[:5]:
    print(f"BIC={bic:8.1f}  AIC={aic:8.1f}  {cov:9s} K={K}")

# check the parameter count used by BIC for a full-covariance model
gm = GaussianMixture(4, covariance_type="full", random_state=0).fit(X)
D, K = X.shape[1], 4
p = (K - 1) + K * D + K * D * (D + 1) // 2
manual = -2 * gm.score(X) * len(X) + p * np.log(len(X))
print("free parameters:", p, " manual BIC:", round(manual, 1), " sklearn:", round(gm.bic(X),
 ↪  1))
```

```
BIC=  4301.1  AIC=  4200.0  full      K=4
BIC=  4320.7  AIC=  4193.2  full      K=5
BIC=  4357.6  AIC=  4203.7  full      K=6
BIC=  4388.4  AIC=  4208.2  full      K=7
BIC=  4414.9  AIC=  4243.4  diag      K=8
free parameters: 23  manual BIC: 4301.1  sklearn: 4301.1
```

BIC 在完整共變異數下選出真正的 $K = 4$（剪切變換讓各群變成橢圓，所以球形與對角模型需要額外的成分）。AIC 稍微偏好 $K = 5$：它較輕的懲罰讓第五個成分可以用小小的似然增益「買」到位置。一般而言兩個準則都不是「正確」的；它們都假設模型族足夠合適，而對非高斯的群，BIC 會很樂意用好幾個高斯來描述一個彎曲的群。若目標是密度估計，這沒有問題；若每個成分應該對應一個語意群，就有問題了。

其他經驗法則

手肘法（elbow method）畫出 k-means 的 J 對 K 的曲線，尋找彎折處；但 J 總是遞減，彎折處的判斷很主觀。下一節的 silhouette 與穩定度方法提供了其他選擇。貝氏作法也可以先放比需要更多的成分，再讓混合權重上的先驗把不需要的成分關掉；Bishop 以變分高斯混合模型完整示範了這一點 [2]。

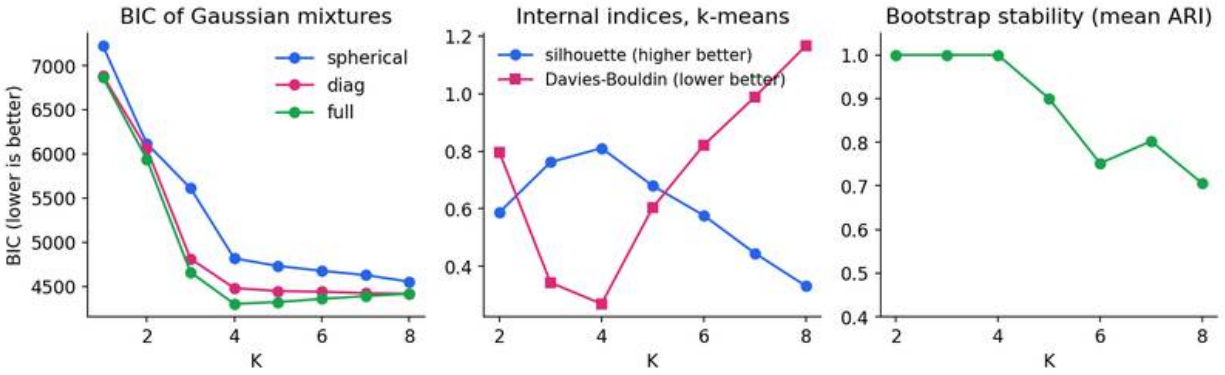


圖 6 — 在合成的四團資料上選擇 K 的三種方式。左：*BIC*（剪切後的資料）。中：*k-means* 的 *silhouette*（越高越好）與 *Davies–Bouldin*（越低越好）。右：自助法穩定度在 $K=3$ 與 $K=4$ 時都完美，因此光靠穩定度無法決定。

分群有效性：分群結果好不好？

白話版。沒有標籤時，「好」必須先被定義：緊密且分得開（內部指標）、與某個可信的參考一致（外部指標），或在資料受擾動時能重現（穩定度）。Theodoridis 與 Koutroumbas 用第 16 章專門討論這些問題 [1]；PR 09 會在監督式情境下回到驗證的議題。

內部指標

Silhouette [14]。對群 C 中的點 i，令 $a(i)$ 為它到 C 中其他點的平均距離，$b(i)$ 為它到任何其他群各點平均距離中的最小值。則

$$s(i) = \frac{b(i) - a(i)}{\max\{a(i), b(i)\}} \in [-1, 1].$$

$s(i)$ 接近 1 表示 i 穩穩位於所屬群內；接近 0 表示在邊界上；負值表示它可能被分錯群。所有點的平均 silhouette 概括了整個分割，而逐點的 silhouette 圖能指出哪些群比較弱。

Davies–Bouldin [15]。令 s_i 為群 i 各點到其質心 c_i 的平均距離，$d_{ij} = \|c_i - c_j\|$。則

$$\mathrm{DB} = \frac{1}{K} \sum_{i=1}^{K} \max_{j \neq i} \frac{s_i + s_j}{d_{ij}},$$

也就是每一群與另一群之間「最壞情況相似度」的平均。越低越好。

Calinski–Harabasz [16]。以群間散佈 $B = \sum_k N_k (c_k - \bar{x})(c_k - \bar{x})^\top$ 與群內散佈 $W = \sum_k \sum_{n \in C_k} (x_n - c_k)(x_n - c_k)^\top$（$\bar{x}$ 為整體平均），

$$\mathrm{CH} = \frac{\operatorname{tr}(B)/(K-1)}{\operatorname{tr}(W)/(N-K)},$$

這是一種變異數比統計量。越高越好。

這三個指標都偏好在所選幾何下緊密、凸、分得開的群。它們會給 k-means 在雙月形資料上的錯誤答案比正確的彎曲答案更高的分數。它們衡量的是與某種假設的群形狀是否一致，而不是語意上是否正確。

外部指標

當有參考標籤時（用於基準測試，或來自另外保留的標註），以對排列不變的指標比較分割。

調整蘭德指數（ARI） [17]。蘭德指數計算兩個分割在多少比例的點對上一致（同為「同一群」或同為「不同群」）。ARI 減去在群大小相同的隨機標記下的期望值，再重新縮放：

$$\mathrm{ARI} = \frac{\mathrm{RI} - \mathbb{E}[\mathrm{RI}]}{\max \mathrm{RI} - \mathbb{E}[\mathrm{RI}]},$$

因此隨機分割的分數約為 0，（在重新編號下）完全相同的分割得 1。

正規化互資訊（NMI）。把兩組標記 U 與 V 視為定義在點上的隨機變數，則

$$\mathrm{NMI}(U, V) = \frac{I(U;V)}{\frac{1}{2}\left(H(U) + H(V)\right)},$$

其中 I 為互資訊，H 為熵（這裡示範的算術平均正規化是 scikit-learn 的預設，也有其他正規化方式）。Vinh、Epps 與 Bailey [18] 指出，當群數相對於 N 很大時，未調整的 NMI 會向上偏差，並建議使用經過機率校正的版本，例如調整互資訊（adjusted mutual information, AMI）。

如果參考標籤曾被用來調整演算法或 K，它們就不再是獨立的檢驗。

穩定度

在自助法樣本或擾動後的資料上重新分群，並在相同的點上以 ARI 比較新分割與原分割。產生不穩定分割的 K 值得懷疑。但穩定是必要條件而非充分條件：把兩團相近的資料合併是完全可重現的（圖 6 右，$K = 3$），而一個穩定的分割仍可能反映的是相機或光線等無關因素。

程式：有效性指標與穩定度

```
import numpy as np
from sklearn.datasets import make_blobs
from sklearn.cluster import KMeans
from sklearn.metrics import (silhouette_score, davies_bouldin_score,
                             calinski_harabasz_score, adjusted_rand_score,
```

```
                            normalized_mutual_info_score)

X, truth = make_blobs(n_samples=600, centers=4, cluster_std=1.0, random_state=7)
print(" K  silhouette(+)  Davies-Bouldin(-)  Calinski-Harabasz(+)")
for K in range(2, 8):
    z = KMeans(K, n_init=10, random_state=0).fit_predict(X)
    print(f"{K:2d}  {silhouette_score(X, z):12.3f}  {davies_bouldin_score(X, z):17.3f}"
          f"  {calinski_harabasz_score(X, z):20.1f}")

z = KMeans(4, n_init=10, random_state=0).fit_predict(X)
print("external ARI:", round(adjusted_rand_score(truth, z), 3),
      " NMI:", round(normalized_mutual_info_score(truth, z), 3))
perm = np.array([2, 0, 3, 1])[z]                       # relabel clusters
print("ARI after relabelling:", round(adjusted_rand_score(truth, perm), 3))

# stability: refit on bootstrap samples, compare on the original points
rng = np.random.default_rng(1)
for K in [3, 4, 5, 6]:
    base = KMeans(K, n_init=10, random_state=0).fit(X)
    scores = []
    for b in range(10):
        idx = rng.integers(0, len(X), len(X))
        boot = KMeans(K, n_init=10, random_state=b).fit(X[idx])
        scores.append(adjusted_rand_score(base.predict(X), boot.predict(X)))
    print(f"K={K}: bootstrap stability (mean ARI) = {np.mean(scores):.3f}")
```

```
 K  silhouette(+)  Davies-Bouldin(-)  Calinski-Harabasz(+)
 2         0.588              0.796                850.9
 3         0.762              0.343               2235.5
 4         0.811              0.270               7641.8
 5         0.680              0.603               6341.0
 6         0.577              0.821               5620.2
 7         0.446              0.988               5304.4
external ARI: 1.0  NMI: 1.0
ARI after relabelling: 1.0
K=3: bootstrap stability (mean ARI) = 1.000
K=4: bootstrap stability (mean ARI) = 1.000
K=5: bootstrap stability (mean ARI) = 0.858
K=6: bootstrap stability (mean ARI) = 0.699
```

在這些分得很開的團上，所有內部指標都同意 $K = 4$，外部指標也確認不論群如何編號都完全吻合。真實資料很少這麼友善：各指標常常意見不一，而這種不一致本身就是資訊，它告訴你在所選尺度下，資料並沒有唯一明確的群結構。

現代觀點

白話版。本章的演算法已有數十年歷史，至今仍無所不在。改變的是表示法：今天我們通常對學習得到的嵌入向量分群，而不是對原始像素或手工描述子；深度網路有時甚至是藉由分群來訓練的。

值得一讀的綜述

- **Xu 與 Wunsch，“Survey of clustering algorithms”（2005）** [19] 是從圖型識別與類神經網路觀點出發的經典廣泛綜述。它先回顧鄰近度量，再把演算法整理為階層式、平方誤差（k-means 家族）、混合密度、圖論、模糊、類神經（自組織映射、自適應共振理論）、核方法、序列資料、大規模與高維方法，最後討論分群有效性。它是古典分群領域的一張好地圖。
- **Jain，“Data clustering: 50 years beyond K-means”（2010）** [20] 是一篇較短、觀點鮮明的概論。它的主要訊息至今仍適用：沒有放諸四海皆準的最佳演算法；表示法與「群」的定義比演算法更重要；群數與有效性仍是開放問題；附加資訊（成對限制、半監督分群）與分群集成是有價值的方向。
- **Ikotun 等人，“K-means clustering algorithms: A comprehensive review, variants analysis, and advances in the era of big data”（2023）** [21] 回顧了眾多 k-means 變體：初始化（包括 k-means++）、自動選擇 K、處理混合型或高維資料，以及可擴展與分散式的實作。
- **Zhou 等人，“A comprehensive survey on deep clustering: Taxonomy, challenges, and future directions”（ACM Computing Surveys，2024）** [22] 依「表示學習模組」與「分群模組」如何互動來整理深度分群：多階段方法（先學特徵再分群）、迭代方法（偽標籤與特徵學習交替進行）、生成式方法（如具混合先驗的自編碼器等潛在變數模型），以及同時最佳化方法（兩者聯合最佳化）。它也討論了基準、評估指標與應用。
- **Ren 等人，“Deep clustering: A comprehensive survey”（IEEE TNNLS，2025）** [23] 依資料來源與先驗知識整理此領域：單視角、半監督、多視角與遷移式深度分群。與 [22] 一起閱讀，可以看出深度分群的設計空間已變得多麼寬廣。

從視覺詞到自我標註

兩篇具影響力的視覺論文展示了古典分群如何變成訓練訊號。**DeepCluster** [24] 交替進行兩件事：對卷積網路的特徵執行 k-means，以及訓練該網路以群指派作為偽標籤來預測；在整個資料集上反覆進行，就能從無標籤影像學到有用的特徵。**SCAN** [25] 則把表示學習與分群分開：先用自監督的前置任務（pretext task）學特徵，再在該特徵空間中為每張影像找出最近鄰，訓練一個分群頭讓鄰居被分到同一群，最後以自我標註加以精煉。兩者都是把本章的概念（k-means 作為量化、鄰域作為相似性的證據）應用到學習得到的表示上；ML 06 說明了這些表示從何而來。

實務現況（2023–2026）與開放問題

在目前的實務中，為影像分組的常見作法是：從預訓練（通常是自監督）的編碼器取得嵌入向量，視需要降維，再執行 k-means、高斯混合或 HDBSCAN，並在一小部分已標註資料上驗證。本章的提醒依然完全適用：

- **表示法定義了相似性**。為某個目的訓練的嵌入向量，可能依風格、背景或相機分群，而不是依你在意的類別。
- **K 與有效性仍是開放問題**。資訊準則假設了模型族；內部指標假設了形狀；外部指標需要標籤。上述綜述沒有任何一篇宣稱有通用解法。
- **深度分群的評估通常仰賴有標籤的基準**，以及對排列不變的分數，例如把群與類別做最佳配對後的分群準確率、NMI 與 ARI [22]。這些分數衡量的是與某一種選定標記的一致程度，而不

是「真正的」結構。
- **規模與不平衡**。長尾資料、大量小群與串流資料，同時考驗古典與深度方法；[21] 與 [22] 都把它們列為活躍的研究問題。

重點整理

- 分群由它的相似性概念所定義。特徵、尺度與距離是在任何演算法執行前就做出的建模決策。
- K-means（Lloyd）交替進行最近中心指派與平均更新；J 單調下降至局部最小值。k-means++ 播種給出 $\mathbb{E}[J] \le 8(\ln K + 2) J_{\mathrm{OPT}}$，並大幅減少糟糕的結果；但仍應多次重新啟動。
- 連結方式定義了階層式分群：single linkage 會沿著細橋串接，complete 與 Ward 偏好緊密的群。樹狀圖仍然需要被切開。
- DBSCAN 把群定義為密度相連的區域，能找出任意形狀並標出雜訊；ε 由 k 距離曲線決定，HDBSCAN 能處理不同的密度。
- 高斯混合模型的 EM 由完整資料對數似然推出：E 步計算責任值，M 步計算加權平均與共變異數，並以拉格朗日乘數得到 $\pi_k = N_k/N$。ELBO 分解證明了單調改善；奇異解與標籤交換需要明確處理；k-means 是硬指派、等向的極限。
- $\mathrm{BIC} = -2\ln\hat{L} + p\ln N$ 與 $\mathrm{AIC} = -2\ln\hat{L} + 2p$ 在擬合與參數數量之間取捨；共變異數形式也是選擇的一部分。
- 內部指標（silhouette、Davies–Bouldin、Calinski–Harabasz）獎勵某種假設的幾何；外部指標（ARI、NMI）需要獨立的標籤；穩定度是必要但不充分的條件。

練習

1. **Lloyd 演算法不會增加 J**。證明指派步驟與更新步驟都不會增加 J，並據此論證演算法一定會停止。如果平手時隨機決定，終止論證哪裡會出問題？

   提示

   指派步驟分別最小化每一項 $\|x_n - \mu_{z_n}\|^2$；更新步驟利用 $\sum_{n\in C_k}\|x_n-\mu\|^2 = \sum_{n\in C_k}\|x_n-\bar{x}_k\|^2 + N_k\|\bar{x}_k-\mu\|^2$。分割最多有 K^N 種。若平手時隨機決定，兩個 J 相等的分割可能永遠來回切換；固定規則（例如平手時留在目前的群）可以避免。

2. **手算 k-means++**。一維資料為 $\{0, 1, 2, 10, 11\}$，$K = 2$，第一個種子為 0。計算每個點成為第二個種子的機率，並與均勻抽樣第二個種子比較。

   提示

   D^2 值為 $0, 1, 4, 100, 121$，總和 226。因此 $P(10) = 100/226$、$P(11) = 121/226$，合計約 0.98。若均勻抽樣，第二個種子落在右側那一群的機率只有 2/5（若排除第一個種子則為 2/4）。

3. **一維混合模型的 M 步**。對變異數為 σ_1^2, σ_2^2 的一維兩成分混合，不用矩陣微積分，直接從 Q 推導 σ_k^2 的 M 步更新。接著證明若某成分的責任值集中在單一點 x_n 上（該點 $r_{nk} \approx 1$、其餘點 ≈ 0），更新會把 σ_k^2 推向 0。scikit-learn 的 `reg_covar` 怎麼處理這件事？

提示

對 $\sum_n r_{nk}[-\frac{1}{2}\ln\sigma_k^2 - (x_n-\mu_k)^2/(2\sigma_k^2)]$ 關於 σ_k^2 微分，得到 $\sigma_k^2 = \sum_n r_{nk}(x_n-\mu_k)^2/N_k$。若所有權重都在同一點上，$\mu_k = x_n$，總和為零。`reg_covar` 在每次 M 步後於對角線加上一個小常數，讓變異數不會趨近於零。

4. **計算參數數量**。對 $D = 64$ 的局部描述子與 $K = 32$ 個成分，分別計算完整、對角與球形共變異數 GMM 的自由參數數量。若有 $N = 10^5$ 個描述子，各情況的 BIC 懲罰 $p\ln N$ 有多大？

提示

完整：$31 + 32\cdot 64 + 32\cdot 64\cdot 65/2 = 31 + 2048 + 66560 = 68{,}639$。對角：$31 + 2048 + 2048 = 4{,}127$。球形：$31 + 2048 + 32 = 2{,}111$。再乘上 $\ln 10^5 \approx 11.51$。完整模型必須讓 $-2\ln\hat{L}$ 比對角模型改善大約 7.4×10^5 才能打平。

5. **當指標誤導你**。在比較程式的雙月形資料上，以 $K = 2$ 執行 k-means 並執行 DBSCAN。計算兩個分割的 silhouette 分數，以及它們相對於生成標籤的 ARI。用各指標衡量的內容解釋兩者的分歧。

提示

Silhouette 以歐氏距離計算，偏好凸且分得開的群，因此 k-means 的切法通常比兩個月形得到更高分，儘管它的 ARI 約為 0.26，而 DBSCAN 為 1.0。內部指標隱含了形狀假設；這裡的「正確」答案是由連通性定義的。

6. **穩定不等於真相**。產生四團高斯資料，其中兩團非常接近（例如中心為 $(0,0)$ 與 $(1.2,0)$，標準差 0.5）。計算 $K = 3$ 與 $K = 4$ 的自助法穩定度。哪一個比較穩定？哪一個符合生成過程？還需要什麼額外資訊才能下定論？

提示

把相近的兩團合併是非常可重現的，因此 $K = 3$ 的穩定度可能與 $K = 4$ 一樣高甚至更高。生成過程有四群，但它們是否應該被分開取決於任務。領域知識、一小部分已標註資料，或以 GMM 的 BIC 這類以密度為基礎的準則，都能提供額外證據。

參考文獻

1. S. Theodoridis and K. Koutroumbas, *Pattern Recognition*, 4th ed., Academic Press, 2009, Ch. 11–16 (clustering basic concepts, sequential, hierarchical, function-optimization and graph-based algorithms, cluster validity). doi
2. C. M. Bishop, *Pattern Recognition and Machine Learning*, Springer, 2006, Ch. 9 (k-means, mixtures of Gaussians, EM, the general EM algorithm). book page
3. S. P. Lloyd, “Least squares quantization in PCM,” *IEEE Transactions on Information Theory*, vol. 28, no. 2, pp. 129–137, 1982. doi

4. J. MacQueen, “Some methods for classification and analysis of multivariate observations,” in *Proc. Fifth Berkeley Symposium on Mathematical Statistics and Probability*, vol. 1, pp. 281–297, 1967. Project Euclid
5. D. Arthur and S. Vassilvitskii, “k-means++: The advantages of careful seeding,” in *Proc. 18th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2007, pp. 1027–1035. pdf
6. D. Sculley, “Web-scale k-means clustering,” in *Proc. 19th International Conference on World Wide Web (WWW)*, 2010, pp. 1177–1178. doi
7. J. H. Ward, “Hierarchical grouping to optimize an objective function,” *Journal of the American Statistical Association*, vol. 58, pp. 236–244, 1963. doi
8. M. Ester, H.-P. Kriegel, J. Sander and X. Xu, “A density-based algorithm for discovering clusters in large spatial databases with noise,” in *Proc. 2nd International Conference on Knowledge Discovery and Data Mining (KDD)*, 1996. pdf
9. R. J. G. B. Campello, D. Moulavi and J. Sander, “Density-based clustering based on hierarchical density estimates,” in *Advances in Knowledge Discovery and Data Mining (PAKDD)*, LNCS, 2013, pp. 160–172. doi
10. U. von Luxburg, “A tutorial on spectral clustering,” *Statistics and Computing*, vol. 17, pp. 395–416, 2007. doi
11. A. P. Dempster, N. M. Laird and D. B. Rubin, “Maximum likelihood from incomplete data via the EM algorithm,” *Journal of the Royal Statistical Society: Series B*, vol. 39, no. 1, pp. 1–22, 1977. doi
12. G. Schwarz, “Estimating the dimension of a model,” *The Annals of Statistics*, vol. 6, no. 2, 1978. doi
13. H. Akaike, “A new look at the statistical model identification,” *IEEE Transactions on Automatic Control*, vol. 19, no. 6, pp. 716–723, 1974. doi
14. P. J. Rousseeuw, “Silhouettes: A graphical aid to the interpretation and validation of cluster analysis,” *Journal of Computational and Applied Mathematics*, vol. 20, pp. 53–65, 1987. doi
15. D. L. Davies and D. W. Bouldin, “A cluster separation measure,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-1, no. 2, pp. 224–227, 1979. doi
16. T. Caliński and J. Harabasz, “A dendrite method for cluster analysis,” *Communications in Statistics*, vol. 3, no. 1, pp. 1–27, 1974. doi
17. L. Hubert and P. Arabie, “Comparing partitions,” *Journal of Classification*, vol. 2, pp. 193–218, 1985. doi
18. N. X. Vinh, J. Epps and J. Bailey, “Information theoretic measures for clusterings comparison: Variants, properties, normalization and correction for chance,” *Journal of Machine Learning Research*, vol. 11, pp. 2837–2854, 2010. JMLR
19. R. Xu and D. Wunsch II, “Survey of clustering algorithms,” *IEEE Transactions on Neural Networks*, vol. 16, no. 3, pp. 645–678, 2005. doi
20. A. K. Jain, “Data clustering: 50 years beyond K-means,” *Pattern Recognition Letters*, vol. 31, no. 8, pp. 651–666, 2010. doi
21. A. M. Ikotun, A. E. Ezugwu, L. Abualigah, B. Abuhaija and J. Heming, “K-means clustering

algorithms: A comprehensive review, variants analysis, and advances in the era of big data," *Information Sciences*, vol. 622, pp. 178–210, 2023. doi

22. S. Zhou, H. Xu, Z. Zheng, J. Chen, Z. Li, J. Bu, J. Wu, X. Wang, W. Zhu and M. Ester, "A comprehensive survey on deep clustering: Taxonomy, challenges, and future directions," *ACM Computing Surveys*, vol. 57, no. 3, pp. 1–38, 2024 (arXiv:2206.07579). doi
23. Y. Ren, J. Pu, Z. Yang, J. Xu, G. Li, X. Pu, P. S. Yu and L. He, "Deep clustering: A comprehensive survey," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 36, no. 4, pp. 5858–5878, 2025 (arXiv:2210.04142). doi
24. M. Caron, P. Bojanowski, A. Joulin and M. Douze, "Deep clustering for unsupervised learning of visual features," in *Proc. European Conference on Computer Vision (ECCV)*, LNCS, 2018, pp. 139–156. doi
25. W. Van Gansbeke, S. Vandenhende, S. Georgoulis, M. Proesmans and L. Van Gool, "SCAN: Learning to classify images without labels," in *Proc. European Conference on Computer Vision (ECCV)*, LNCS, 2020, pp. 268–285. doi
26. scikit-learn developers, "GaussianMixture," API documentation, accessed 2026-11-05. documentation

PR 08・近似推論與貝氏模型複雜度

後驗積分為何困難，以及兩大類解法：取樣（Monte Carlo、重要性取樣、Metropolis–Hastings、Gibbs、HMC）與最佳化（變分推論、Laplace、EP），並連結到 VAE 與擴散模型。

網站版本: https://blog.shyandram.dev/zh-tw/learn/pattern-recognition/approximate-inference/

你將學到

- 為什麼貝氏後驗在差一個常數的意義下很容易「計算」，卻很難「積分」；以及為什麼網格法在幾個維度之後就失效。
- 普通 Monte Carlo、拒絕取樣（rejection sampling）與重要性取樣（importance sampling）如何把積分變成平均，以及如何判斷權重已經崩潰。
- 馬可夫鏈 Monte Carlo（MCMC）如何運作：細緻平衡（detailed balance）、Metropolis–Hastings（含非對稱提議的修正項）、Gibbs 取樣、Hamiltonian Monte Carlo，以及讓你保持誠實的診斷工具（$\hat{R}$、有效樣本數）。
- 變分推論（variational inference）如何把積分變成最佳化：用兩種方法推導證據下界（ELBO）、mean-field 座標上升、反向與正向 KL 的差別，以及搭配重參數化技巧（reparameterization trick）的隨機梯度。
- Laplace 近似與期望傳播（expectation propagation）的定位、模型證據如何懲罰複雜度，以及同樣的想法如何驅動 VAE 與擴散模型。

先看全貌

前面的章節都在找一組「最好」的參數：最大概似、MAP，或混合模型的 EM。貝氏辨識系統問的是更大的問題：看過資料之後，每一組參數各有多可信？把它們全部考慮進來，平均會預測什麼？回答這個問題需要對所有未知量積分，而幾乎所有實用模型的積分都沒有閉式解。本章介紹兩大類替代方案：**抽樣**，產生行為像後驗的樣本；或**擬合**一個較簡單、但盡量貼近後驗的分布。

> **用最簡單的話說**
>
> 想像一大片漆黑的田地，到處埋著多寡不一的寶藏，而你想知道寶藏的總量。找到最富的那一點很容易：一直往上坡走就好，這就是最佳化。要知道總量就難多了。一種做法是隨機但聰明地四處走動，在寶藏多的地方多停留，邊走邊記錄，這就是**取樣**。另一種做法是在田上蓋一條簡單平滑的毯子，盡量讓毯子的形狀像藏寶圖，然後改量毯子，這就是**變分推論**。走路慢但老實；毯子快，卻可能把它沒被設計來表現的起伏藏起來。

為什麼對辨識很重要：不確定性告訴系統何時該拒絕判斷（見 PR 01）；積分後的證據可以在沒有保留驗證集的情況下比較模型；而本章的工具，尤其是 ELBO 與 Langevin 式取樣，正是現代生成模型所用的方法。本章依循 Bishop 關於近似推論與取樣的章節 [1]；Theodoridis 與 Koutroumbas 則提供估計與最佳化的背景（第 2 章與附錄 C），並非與本章方法一一對應 [2]。

為什麼精確推論很難

白話版。貝氏定理說：後驗等於「先驗乘以概似，再除以一個正規化的數」。分子對任何單一猜測通常都好算；分母卻是對所有可能猜測的總和，麻煩就從這裡開始。

精確版。把所有未知量（潛在變數與參數）合記為 z，觀測資料記為 D，則

$$p(z \mid D) = \frac{p(D, z)}{p(D)}, \qquad p(D) = \int p(D, z)\, dz,$$

其中 $p(D, z) = p(D \mid z)\, p(z)$ 是聯合密度（概似乘以先驗），$p(D)$ 是**證據**（evidence），又稱邊際概似。我們想要的幾乎都是後驗下的期望值：

$$\mathbb{E}_{p(z \mid D)}[f(z)] = \int f(z)\, p(z \mid D)\, dz,$$

例如後驗平均（$f(z) = z$）、預測機率（$f(z) = p(y^* \mid x^*, z)$），或某參數為正的機率（$f(z) = \mathbf{1}[z > 0]$）。以下記 $\tilde{p}(z) = p(D, z)$ 為**未正規化**後驗：逐點計算很便宜，但正規化常數 $Z = p(D)$ 未知。

積分困難有三個原因：

1. **維度**。每軸 G 個點的網格在 d 維需要 G^d 次計算。$G = 201$、$d = 50$ 時約為 10^{115}，遠超任何電腦的能力。
2. **集中**。資料多時，後驗只占廣大空間中極小的區域，隨機猜測幾乎永遠落不進去。
3. **離散結構**。有 K 個成分、N 個點的混合模型，潛在指派有 K^N 種組合；即使每一項都很簡單，也無法精確加總。

MAP 估計藉由最大化 $\tilde{p}$ 繞過這些困難，但最大值不是積分。後驗的最高點可能位於一個很窄、質量很少的尖峰上，而單一點完全不帶有分散程度的資訊。PR 07 的 EM 介於兩者之間：它在 E step 對潛在指派積分（高斯混合可以精確做到），但最後仍回傳參數的點估計。

實例：貝氏邏輯斯迴歸

考慮二類問題，輸入 $x_n \in \mathbb{R}^2$（一個偏差特徵與一個量測值），標籤 $y_n \in \{0, 1\}$，權重 w，模型為

$$p(y_n = 1 \mid x_n, w) = \sigma(w^\top x_n), \qquad \sigma(s) = \frac{1}{1 + e^{-s}}, \qquad p(w) = \mathcal{N}(0, 4I).$$

對數聯合密度為 $\sum_n \left[y_n w^\top x_n - \log(1 + e^{w^\top x_n}) \right] - \frac{1}{8}\|w\|^2$ 加上常數。高斯先驗與邏輯斯概似不共軛，因此後驗不是高斯，正規化常數也沒有公式。只有兩個權重時，我們仍可以用網格暴力計算，得到一個可以拿來檢驗所有近似方法的標準答案。

```
import numpy as np

rng = np.random.default_rng(0)
n = 40
X = np.c_[np.ones(n), rng.normal(size=n)]
w_true = np.array([-0.5, 1.5])
y = rng.random(n) < 1 / (1 + np.exp(-X @ w_true))
```

```
def log_joint(w):                       # log p(y | X, w) + log p(w), w ~ N(0, 2^2 I)
    s = X @ w
    return np.sum(y * s - np.logaddexp(0, s)) - 0.5 * np.sum(w**2) / 4.0

# Brute-force normalization on a grid: fine in 2-D, hopeless in 50-D.
g = np.linspace(-4, 4, 201)
W0, W1 = np.meshgrid(g, g, indexing="ij")
logp = np.array([[log_joint(np.array([a, b])) for b in g] for a in g])
cell = (g[1] - g[0])**2
log_evidence = np.log(np.sum(np.exp(logp - logp.max())) * cell) + logp.max()
post = np.exp(logp - log_evidence)
mean = np.array([np.sum(W0 * post), np.sum(W1 * post)]) * cell
sd = np.sqrt(np.array([np.sum((W0 - mean[0])**2 * post),
                       np.sum((W1 - mean[1])**2 * post)]) * cell)
print("log evidence:", round(log_evidence, 3))
print("posterior mean:", mean.round(3), " sd:", sd.round(3))
print("grid points for 201 per axis in 50-D:", f"{201.0**50:.2e}")
```

輸出的對數證據為 -22.117，後驗平均為 $(-0.303, 1.631)$，標準差為 $(0.366, 0.578)$。注意取指數前先減去 `logp.max()`：真實資料的對數密度是很大的負數，在對數空間運算（log-sum-exp 技巧）是每位實作者的第一個習慣。稍後我們會用同一個模型來評估 Laplace 與變分近似。

Monte Carlo：把積分變成平均

白話版。如果能從某個分布抽出隨機樣本，那麼某個量在這些樣本上的平均，就能估計它的期望值。樣本越多估計越準，而誤差縮小的速度可以預測，而且與維度無關。

精確版。給定獨立樣本 $z^{(1)}, \dots, z^{(S)} \sim p$，$I = \mathbb{E}_p[f(z)]$ 的 Monte Carlo 估計量為

$$\hat{I}_S = \frac{1}{S}\sum_{s=1}^{S} f(z^{(s)}), \qquad \mathbb{E}[\hat{I}_S] = I, \qquad \operatorname{Var}[\hat{I}_S] = \frac{\operatorname{Var}_p[f(z)]}{S}.$$

其中 S 是樣本數，$\operatorname{Var}_p[f]$ 是 f 在 p 下的變異數。假設 $\mathbb{E}_p[|f|]$ 有限，估計量就是**不偏**的；上面的變異數與 $1/\sqrt{S}$ 標準誤敘述，還需要 $\operatorname{Var}_p[f]$ 有限：誤差要減半，樣本要變四倍。最驚人的是式中沒有 d。收斂速率的指數不直接依賴維度，但 f 的變異數與取得每個樣本的成本仍可能隨維度增加。若重尾量具有無限變異數，就沒有這項標準誤保證。

問題在於「從 p 抽樣」這個假設。後驗通常無法直接抽樣。本章取樣部分的其餘內容，就是在製造這樣的樣本，或修正從錯誤分布抽出的樣本。

拒絕取樣

假設我們能從簡單的**提議分布**（proposal）$q(z)$ 抽樣，並知道常數 M 使得對所有 z 都有 $M q(z) \geq \tilde{p}(z)$。抽 $z \sim q$ 與 $u \sim \text{Uniform}(0,1)$，若 $u < \tilde{p}(z)/(Mq(z))$ 就保留 z。保留下來的點正是來自 p 的精確樣本，因為包絡 Mq 底下被接受的區域形狀恰好就是 $\tilde{p}$。接受機率為 Z/M，其中 $Z = \int \tilde{p}$。在高維空間，可用的包絡必須比目標大上許多，幾乎所有樣本都會被拒絕。因此拒絕取樣是處理一維條件分布（例如在 Gibbs 取樣器內部）的積木，而不是通用的後驗引擎 [1]。

重要性取樣

白話版。從一個方便的分布抽樣，再給每個樣本一個權重，說明它在你真正關心的分布下比較可能（或比較不可能）多少。當方便的分布涵蓋重要區域時效果很好；涵蓋不到時，它會默默地失敗。

精確版。對任何在 $p(z)f(z) \neq 0$ 之處都滿足 $q(z) > 0$ 的 q，

$$\mathbb{E}_p[f(z)] = \int f(z)\,\frac{p(z)}{q(z)}\,q(z)\,dz \approx \frac{1}{S}\sum_{s=1}^{S} w_s f(z^{(s)}), \qquad w_s = \frac{p(z^{(s)})}{q(z^{(s)})}, \quad z^{(s)} \sim q.$$

比值 w_s 稱為**重要性權重**。若只有 $\tilde{p}$ 可用，把權重正規化就能消去未知的 Z：

$$\hat{I}_{\mathrm{SN}} = \sum_{s=1}^{S} \bar{w}_s f(z^{(s)}), \qquad \bar{w}_s = \frac{\tilde{w}_s}{\sum_t \tilde{w}_t}, \qquad \tilde{w}_s = \frac{\tilde{p}(z^{(s)})}{q(z^{(s)})}.$$

自我正規化與證據估計需要更強的支撐條件：凡 $\tilde{p}(z) > 0$ 之處都必須有 $q(z) > 0$，包含 $f(z) = 0$ 的區域，因為分母必須涵蓋整個目標分布的質量。否則正規化後估計的是截斷分布，而不是原本的後驗。

這個**自我正規化**（self-normalized）估計量在 S 有限時通常有偏差，但當 $S \to \infty$ 時一致。附帶的好處是 $\frac{1}{S}\sum_s \tilde{w}_s$ 是證據 Z 的不偏估計，這就是重要性取樣用於模型比較的方式。

估計的品質取決於權重有多不均勻。常用的摘要是權重的**有效樣本數**（effective sample size, ESS）：

$$\mathrm{ESS} = \frac{1}{\sum_s \bar{w}_s^2},$$

所有權重相等時為 S，一個權重獨占全部時為 1。有兩種失敗模式值得注意：

- **提議分布太窄或放錯位置**。p 有質量而 q 很小的區域，會產生罕見但巨大的權重。估計被少數樣本主導，變異數甚至可能是無限大。
- **尾部太輕**。比值 p/q 在尾部必須有界。經驗法則是讓 q 的尾部比 p 更重（用 Student-t 取代高斯，或用更寬的高斯）。

```
import numpy as np
from scipy.stats import norm

rng = np.random.default_rng(1)
N = 10_000
exact = norm.sf(3.0)                      # P(X > 3) for X ~ N(0, 1)

# Plain Monte Carlo: most samples never reach the tail.
x = rng.normal(0, 1, N)
plain = np.mean(x > 3)

# Importance sampling with a proposal centered on the tail, q = N(3, 1).
z = rng.normal(3, 1, N)
w = norm.pdf(z) / norm.pdf(z, loc=3)      # p(z) / q(z)
```

```
is_est = np.mean(w * (z > 3))
print(f"exact {exact:.6f}  plain MC {plain:.6f}  importance {is_est:.6f}")

# Self-normalized IS for an unnormalized target, with effective sample size.
def log_p_tilde(t):                       # N(1.5, 0.7^2) without its constant
    return -0.5 * ((t - 1.5) / 0.7) ** 2

for scale in [1.0, 3.0, 0.3]:
    t = rng.normal(0, scale, N)
    log_w = log_p_tilde(t) - norm.logpdf(t, scale=scale)
    wn = np.exp(log_w - log_w.max())
    wn /= wn.sum()
    ess = 1.0 / np.sum(wn**2)
    print(f"proposal N(0,{scale}^2): mean {np.sum(wn * t):.3f}  ESS {ess:7.1f} of {N}")
```

稀有事件的估計說明了重要性取樣存在的理由：精確的尾端機率是 0.001350；10,000 個樣本的普通 Monte Carlo 給出 0.001900（只有 19 個樣本落在尾端），平移後的提議分布則給出 0.001372。第二部分的目標平均為 1.5。兩個寬的提議分布給出 1.509 與 1.492，ESS 約為 1,870 與 2,830；窄的提議分布 $\mathcal{N}(0, 0.3^2)$ 給出 0.921，ESS 只有 13，看起來很有把握，答案卻是錯的。ESS 極小就是警鈴。

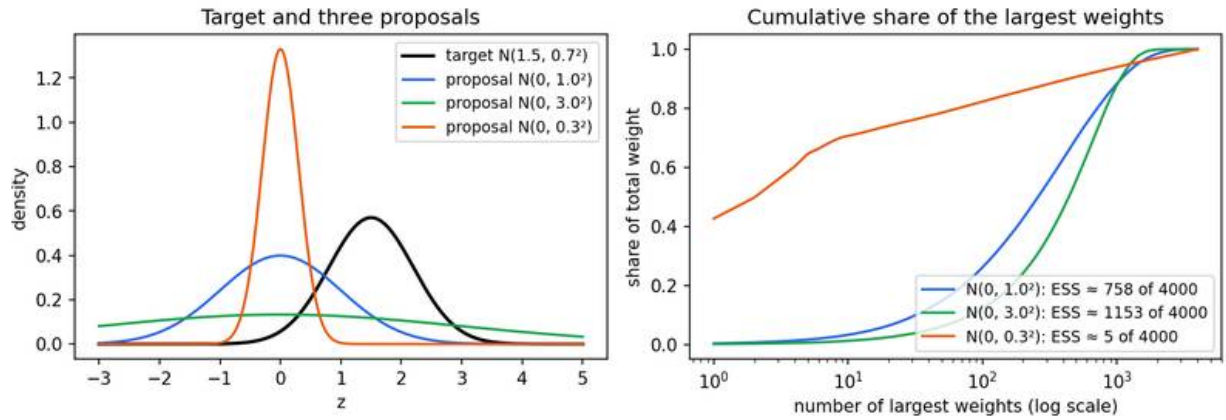


圖 *1* — 每個提議分布抽 *4,000* 個樣本的重要性取樣。窄的提議分布（橘色）幾乎碰不到目標，因此最大的單一權重就占了總權重的 *40%* 以上，*ESS* 約為 *5*。寬的提議分布則把權重分散到數百個樣本。由本篇的圖形腳本計算。

即使不是主要引擎，重要性取樣仍然很重要：它能把舊模型的樣本重新加權到新模型、能估計證據，也是**序列 Monte Carlo**（粒子濾波器）的核心；在序列 Monte Carlo 中，權重隨資料到來逐步更新並重新取樣。

馬可夫鏈 Monte Carlo

白話版。不再抽獨立樣本，而是進行一場隨機漫步，並設計規則讓它長期下來在每個區域停留的時間比例，恰好等於該區域的後驗機率。這樣沿路的時間平均就能估計後驗期望值。相鄰步驟彼此相關，所以一段長漫步的價值少於同樣數量的獨立樣本。

精確版。若轉移核 $T(z' \mid z)$ 滿足

$$\int T(z' \mid z)\, p(z)\, dz = p(z'),$$

則 p 是此馬可夫鏈的**穩態**（不變）分布。一個簡單的充分條件是**細緻平衡**（可逆性）：

$$p(z)\, T(z' \mid z) = p(z')\, T(z \mid z') \quad \text{for all } z, z'.$$

兩邊對 z 積分即得穩態條件。若鏈還能到達每個區域（不可約）且不會週期循環（非週期），則無論從哪裡出發，狀態的分布都會收斂到 p，遍歷平均 $\frac{1}{S}\sum_s f(z^{(s)})$ 也會收斂到 $\mathbb{E}_p[f]$。重點是細緻平衡只涉及比值 $p(z')/p(z)$，因此未知的正規化常數會消去。

Metropolis–Hastings

Metropolis 演算法 [3] 經 Hastings 推廣 [4]，可以從任何提議分布 $q(z' \mid z)$ 建構出這樣的轉移核：

1. 從目前狀態 z 提議 $z' \sim q(z' \mid z)$。
2. 以下列機率接受：

$$A(z', z) = \min\left(1,\ \frac{\tilde{p}(z')\, q(z \mid z')}{\tilde{p}(z)\, q(z' \mid z)}\right).$$

3. 若接受就移到 z'；否則**停在 z，並再記錄一次 z**。

其中 $q(z \mid z')$ 是提議反向移動的密度。因子 $q(z \mid z')/q(z' \mid z)$ 稱為 **Hastings 修正**：它抵銷提議分布對某些方向的偏好。對稱提議（例如高斯隨機漫步）滿足 $q(z' \mid z) = q(z \mid z')$，規則就化簡為 Metropolis 原本的 $\min(1, \tilde{p}(z')/\tilde{p}(z))$。

為什麼行得通？取 $z' \neq z$，並假設 $\tilde{p}(z')q(z \mid z') \le \tilde{p}(z)q(z' \mid z)$。則 $p(z)T(z' \mid z) = p(z)\, q(z' \mid z)\, \frac{p(z')q(z|z')}{p(z)q(z'|z)} = p(z')q(z \mid z')$；反方向的接受率為 1，所以 $p(z')T(z \mid z') = p(z')q(z \mid z')$。兩邊相等，細緻平衡成立。

忘了第 3 步（只記錄被接受的狀態）是經典錯誤：它會高估鏈容易穿過的區域，讓所有估計都產生偏差。

已知動差的高斯目標可以用來檢查隨機漫步鏈。斷言刻意設得寬鬆：這是數值上的基本檢查，不是收斂證明。

```
import numpy as np

rng = np.random.default_rng(12)
def log_target(x):
    return -0.5 * ((x - 1.5) / 0.7)**2
x, accepted = -3.0, 0
samples = []
for step in range(18000):
    proposal = x + rng.normal(0, 0.8)
    if np.log(rng.random()) < min(0.0, log_target(proposal) - log_target(x)):
        x = proposal
```

```
        accepted += 1
    if step >= 2000:                        # discard warm-up, keep repeats
        samples.append(x)
samples = np.asarray(samples)
rho1 = np.corrcoef(samples[:-1], samples[1:])[0, 1]
print("mean, std:", samples.mean(), samples.std())
print("acceptance:", accepted / 18000, "lag-one correlation:", rho1)
assert abs(samples.mean() - 1.5) < 0.12
assert abs(samples.std() - 0.7) < 0.12
```

這條鏈得到平均 1.50、標準差 0.69，接受率 0.67，lag-1 自相關 0.74。步長是主要的調整旋鈕，而光看接受率無法判斷步長好不好。極小的步長幾乎全被接受，卻像在爬行；極大的步長大多被拒絕，鏈停在原地。兩者都會產生高度相關的樣本。

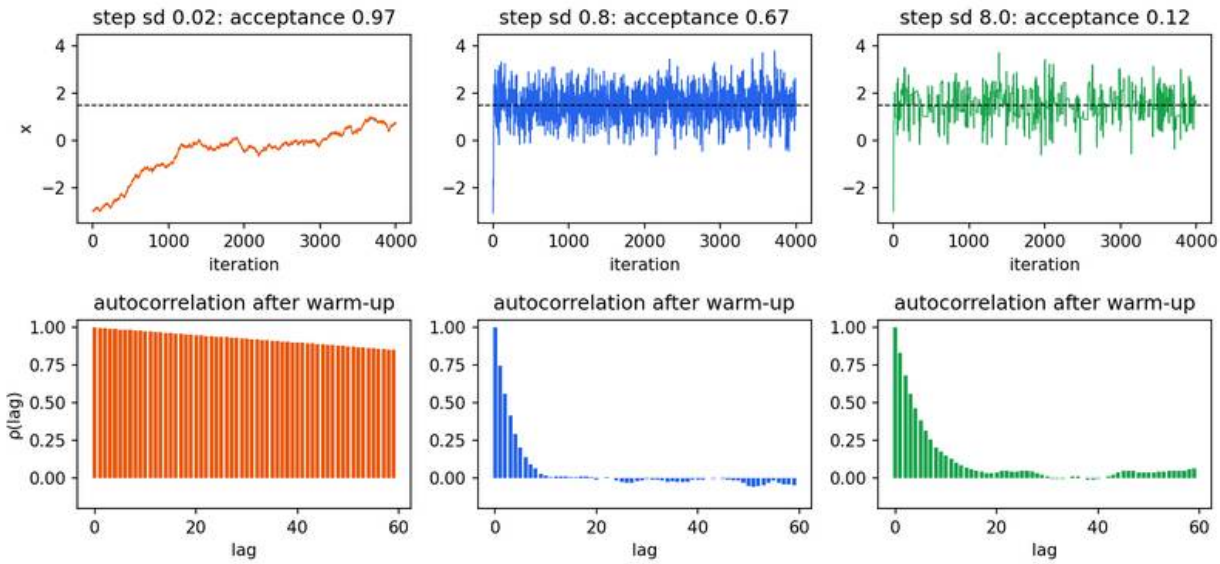


圖 2 — 對 $N(1.5, 0.7^2)$ 執行隨機漫步 *Metropolis*，起點為 -3。步長 *0.02*（接受率 *0.97*）在 *4,000* 步後仍未抵達目標，自相關幾乎不衰減。步長 *0.8* 混合良好。步長 *8*（接受率 *0.12*）長時間停在同一處。由本篇的圖形腳本計算。

Gibbs 取樣

白話版。一次只更新一個變數（或一個區塊），從「給定其他所有變數」的分布中抽它。如果這些條件分布很容易抽（共軛模型常是如此），每一步都會被接受。

精確版。 Gibbs 取樣由 Geman 與 Geman 為了以馬可夫隨機場做影像復原而提出 [5]。它依序走過區塊 $j = 1, \dots, J$，並抽

$$z_j \sim p(z_j \mid z_{-j}, D),$$

其中 z_{-j} 表示除了 j 以外的所有區塊。它是 Metropolis–Hastings 的特例：提議為 $q(z' \mid z) = p(z'_j \mid z_{-j})$ 且 $z'_{-j} = z_{-j}$，代入接受比恰好等於 1 [1]。以高斯混合為例，可以交替地依 responsibility 為

每個點抽成分標籤，再從共軛後驗抽成分參數，這是 EM 的隨機版親戚。

對平均為零、變異數為一、相關係數為 ρ 的二維高斯，條件分布為 $z_1 \mid z_2 \sim \mathcal{N}(\rho z_2,\, 1-\rho^2)$，$z_2$ 亦同。下面的程式從四個分散的角落各跑一條鏈，並計算下一小節介紹的 **split-$\hat{R}$** 收斂診斷。

```
import numpy as np

rng = np.random.default_rng(3)
rho = 0.85                      # target: zero-mean bivariate Gaussian, unit variances
sd = np.sqrt(1 - rho**2)        # conditional standard deviation

def gibbs(n_steps, start):
    z = np.array(start, dtype=float)
    out = np.empty((n_steps, 2))
    for t in range(n_steps):
        z[0] = rng.normal(rho * z[1], sd)   # z1 | z2 ~ N(rho z2, 1 - rho^2)
        z[1] = rng.normal(rho * z[0], sd)   # z2 | z1 ~ N(rho z1, 1 - rho^2)
        out[t] = z
    return out

def split_rhat(chains):         # chains: (m, n) draws of one scalar quantity
    m, n = chains.shape
    halves = chains[:, : n // 2 * 2].reshape(2 * m, n // 2)
    W = halves.var(axis=1, ddof=1).mean()          # within-chain variance
    B = (n // 2) * halves.mean(axis=1).var(ddof=1) # between-chain variance
    var_plus = (n // 2 - 1) / (n // 2) * W + B / (n // 2)
    return np.sqrt(var_plus / W)

starts = [(-4, -4), (4, 4), (-4, 4), (4, -4)]
chains = np.stack([gibbs(3000, s)[500:] for s in starts])   # drop warm-up
draws = chains.reshape(-1, 2)
print("sample correlation:", np.corrcoef(draws.T)[0, 1].round(3))
print("sample variances:", draws.var(axis=0).round(3))
lag1 = np.mean([np.corrcoef(c[:-1, 0], c[1:, 0])[0, 1] for c in chains])
print("lag-1 autocorrelation of z1:", lag1.round(3), " (theory rho^2 =", rho**2, ")")
print("split R-hat for z1:", split_rhat(chains[:, :, 0]).round(4))
```

輸出：樣本相關 0.849，變異數約 0.98，lag-1 自相關 0.717（理論值 $\rho^2 = 0.7225$），split-$\hat{R}$ 為 1.0009。自相關是沿座標軸移動的代價：目標是一條細長的斜脊時，每個座標只能走一小段就撞到「牆」。當 $\rho \to 1$，鏈會慢到幾乎不動。重新參數化（旋轉或白化變數），或把相關的變數當成一個區塊一起更新，就能解決這個問題。

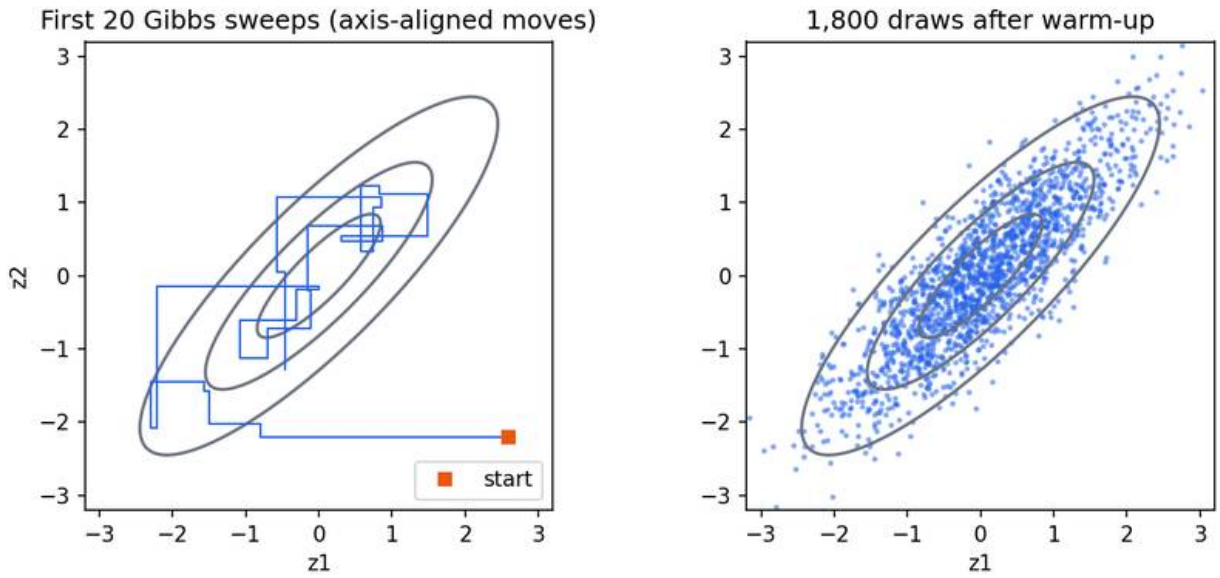


圖 3 — 對相關係數 *0.85* 的高斯進行 *Gibbs* 取樣。每半步都平行於座標軸移動（左），所以沿斜脊前進很慢；但暖機之後，樣本仍正確地填滿目標（右）。由本篇的圖形腳本計算。

Hamiltonian Monte Carlo

白話版。隨機漫步把力氣浪費在來回踱步上。Hamiltonian Monte Carlo（HMC）利用對數後驗的梯度，像一顆在曲面上無摩擦滑行的冰球一樣沿著分布滑行，做出又長又有方向的移動，同時仍以很高的機率被接受。

精確版。定義位能 $U(z) = -\log \tilde{p}(z)$，加入輔助動量 $r \sim \mathcal{N}(0, M)$（M 為質量矩陣），組成 Hamiltonian

$$H(z, r) = U(z) + \tfrac{1}{2} r^{\top} M^{-1} r.$$

Hamiltonian 動力學保持 H 守恆並保持體積。實作上以步長 ϵ 的 L 個 **leapfrog** 步驟模擬：

$$r \leftarrow r - \tfrac{\epsilon}{2} \nabla U(z), \qquad z \leftarrow z + \epsilon M^{-1} r, \qquad r \leftarrow r - \tfrac{\epsilon}{2} \nabla U(z).$$

leapfrog 積分器可逆且保持體積，但不完全守恆能量，因此終點 (z', r') 以 $\min\left(1, \exp(H(z,r) - H(z',r'))\right)$ 的機率被接受，也就是在聯合空間上的一步 Metropolis [6]。軌跡順著目標的幾何走，因此 HMC 能穿越會困住 Gibbs 或隨機漫步的相關斜脊。它的調整參數 ϵ 與 L 很敏感；**No-U-Turn Sampler**（NUTS）在路徑開始折返時停止，自動決定軌跡長度，並在暖機期間調整 ϵ [7]。Betancourt 的概念性導論以**典型集合**（typical set，後驗質量實際所在的那層薄殼）解釋 HMC 為何能擴展到高維 [8]。HMC 需要可微、連續的目標；離散潛在變數必須先加總掉，或用其他移動方式處理。

一個近親是 **Langevin** 更新

$$z \leftarrow z + \tfrac{\epsilon}{2} \nabla \log \tilde{p}(z) + \sqrt{\epsilon}\, \xi, \qquad \xi \sim \mathcal{N}(0, I),$$

它相當於重新抽動量後只走一步 leapfrog。沒有接受/拒絕步驟時，有限的 ϵ 只是近似正確；加上 Metropolis 修正就成為 Metropolis-adjusted Langevin algorithm。請記住這個更新，它會在擴散模型中再次出現。

診斷：這條鏈說的是實話嗎？

有限次的執行不可能證明收斂，診斷只能揭露問題。標準做法：

- **從分散的起點跑多條鏈**，並捨棄每條鏈開頭的暖機部分。
- **用 $\hat{R}$ 比較各鏈**。有 m 條（切成兩半的）長度為 n 的鏈，令 W 為鏈內變異數的平均、B/n 為各鏈平均的變異數，則

$$\hat{R} = \sqrt{\frac{\frac{n-1}{n}W + \frac{1}{n}B}{W}}.$$

若所有鏈探索的是同一個分布，$\hat{R} \approx 1$；明顯大於 1 表示各鏈意見不一。Vehtari 等人指出傳統版本會漏掉重尾與尺度差異，並提出秩正規化（rank-normalized）、折疊的 split-$\hat{R}$，以及 bulk 與 tail 有效樣本數；這就是目前軟體所回報的版本 [9]。

- **回報有效樣本數**。對於穩態鏈，若目標量的 lag-ℓ 自相關為 ρ_ℓ，

$$N_{\text{eff}} \approx \frac{N}{1 + 2\sum_{\ell \geq 1} \rho_\ell},$$

當估計的自相關變得雜亂時就截斷總和。Monte Carlo 標準誤大約是 $\text{sd}/\sqrt{N_{\text{eff}}}$，而不是 $\text{sd}/\sqrt{N}$。對上面的 Gibbs 鏈，$\rho_\ell = \rho^{2\ell}$，所以 $N_{\text{eff}}/N = (1-\rho^2)/(1+\rho^2) \approx 0.16$：儲存的 10,000 個樣本約只值 1,600 個獨立樣本。

- **檢查軌跡圖**；對梯度式取樣器，也要注意 divergence 警告，它標示出積分器跟不上的高曲率區域。

不同的量混合速度不同：平均看起來沒問題，尾端分位數卻可能還沒收斂。而且即使鏈混合得完美，它描述的也只是你寫下的模型的後驗，完全無法說明模型本身是否正確。

變分推論

白話版。選一族簡單的分布（例如彼此獨立的高斯），在其中尋找最接近後驗的那一個。積分變成最佳化，速度快、能處理巨量資料，代價是一種無法靠跑更久來消除的偏差。

精確版。選擇一族可處理的密度 $\mathcal{Q}$，以**變分參數** ϕ 編號。變分推論（VI）[10] 求解

$$q^* = \arg\min_{q \in \mathcal{Q}} D_{\text{KL}}(q(z) \,\|\, p(z \mid D)), \qquad D_{\text{KL}}(q\|p) = \mathbb{E}_q\left[\log \frac{q(z)}{p(z)}\right].$$

這個 KL 含有未知的 $p(D)$，無法直接計算。訣竅就是 ELBO。

用兩種方法推導 ELBO

推導一：分解。把 $\log p(z \mid D) = \log p(D,z) - \log p(D)$ 代入 KL：

$$D_{\mathrm{KL}}(q \| p(\cdot \mid D)) = \mathbb{E}_q[\log q(z)] - \mathbb{E}_q[\log p(D,z)] + \log p(D).$$

移項得到

$$\log p(D) = \mathcal{L}(q) + D_{\mathrm{KL}}(q(z) \,\|\, p(z \mid D)), \qquad \mathcal{L}(q) = \mathbb{E}_q[\log p(D,z)] - \mathbb{E}_q[\log q(z)].$$

$\log p(D)$ 與 q 無關，所以最大化 $\mathcal{L}$ 恰好等於最小化 KL。KL 非負，所以 $\mathcal{L}(q) \le \log p(D)$：它是**證據下界**（evidence lower bound），簡稱 ELBO。

推導二：Jensen 不等式。在證據積分中同乘同除 q，再利用 log 的凹性：

$$\log p(D) = \log \mathbb{E}_q\left[\frac{p(D,z)}{q(z)}\right] \ge \mathbb{E}_q\left[\log \frac{p(D,z)}{q(z)}\right] = \mathcal{L}(q).$$

第一個等號假設 q 涵蓋 $p(D,z)$ 的支撐集。若漏掉正的聯合機率質量，期望只積分 q 的支撐範圍，不再等於完整證據。只要各項定義良好，推導一仍保證 ELBO 是下界；在支撐涵蓋假設下，Jensen 不等式的差距才正好是該 KL。ELBO 有一個好用的讀法：$\mathcal{L} = \mathbb{E}_q[\log p(D \mid z)] - D_{\mathrm{KL}}(q(z) \| p(z))$，也就是要擬合資料，但不要離先驗太遠。EM 是一個特例：在 E step 把潛在變數的 q 設為精確後驗（讓下界變緊），再於 M step 對參數最大化下界 [1]。

Mean field 與座標上升（CAVI）

最常見的分布族是 **mean field**：$q(z) = \prod_j q_j(z_j)$，即 z 各區塊上彼此獨立的因子。固定 q_j 以外的所有因子，對 q_j 最大化 ELBO，得到

$$\log q_j^*(z_j) = \mathbb{E}_{q_{-j}}[\log p(D,z)] + \text{const},$$

其中 $\mathbb{E}_{q_{-j}}$ 對其他所有因子取平均，常數讓 q_j^* 正規化。（推導方式：把 $\mathcal{L}$ 寫成只含 q_j 的函數，它等於 $-D_{\mathrm{KL}}(q_j \| \tilde{q}_j)$ 加上常數，其中 $\log \tilde{q}_j = \mathbb{E}_{q_{-j}}[\log p(D,z)]$。）依序輪流更新各個 j，就是**座標上升變分推論**（CAVI）。每次更新都不會降低 ELBO，所以 CAVI 會收斂，但只收斂到局部最佳 [11]。對條件共軛的模型，每個 q_j^* 會與先驗同族，更新有閉式解。

實例：平均與精確度都未知的高斯。觀測 $x_n \sim \mathcal{N}(\mu, \tau^{-1})$，先驗 $\mu \mid \tau \sim \mathcal{N}(\mu_0, (\lambda_0\tau)^{-1})$、$\tau \sim \mathrm{Gamma}(a_0, b_0)$。精確後驗已知（Normal–Gamma），因此可以為近似打分數。取 $q(\mu,\tau) = q(\mu)q(\tau)$，mean-field 更新規則給出 $q(\mu) = \mathcal{N}(\mu_N, \lambda_N^{-1})$ 與 $q(\tau) = \mathrm{Gamma}(a_N, b_N)$，其中 [1]

$$\mu_N = \frac{\lambda_0\mu_0 + N\bar{x}}{\lambda_0 + N}, \qquad \lambda_N = (\lambda_0 + N)\,\mathbb{E}[\tau],$$
$$a_N = a_0 + \frac{N+1}{2}, \qquad b_N = b_0 + \tfrac{1}{2}\,\mathbb{E}_\mu\Big[\textstyle\sum_n (x_n - \mu)^2 + \lambda_0(\mu - \mu_0)^2\Big],$$

$\bar{x}$ 為樣本平均，$\mathbb{E}[\tau] = a_N/b_N$。兩個因子透過 $\mathbb{E}[\tau]$ 以及 $\mathbb{E}[\mu]$、$\mathbb{E}[\mu^2]$ 互相依賴，所以要反覆迭代。

```
import numpy as np

# Data x_n ~ N(mu, 1/tau); prior mu | tau ~ N(mu0, 1/(lam0 tau)), tau ~ Gamma(a0, b0).
rng = np.random.default_rng(4)
x = rng.normal(2.0, 0.5, size=30)
N, xbar = len(x), x.mean()
mu0, lam0, a0, b0 = 0.0, 1.0, 1.0, 1.0

# Mean field q(mu, tau) = q(mu) q(tau) with q(mu) = N(mu_N, 1/lam_N), q(tau) = Gamma(a_N,
 ↪  b_N).
E_tau = a0 / b0
for it in range(50):
    mu_N = (lam0 * mu0 + N * xbar) / (lam0 + N)
    lam_N = (lam0 + N) * E_tau
    a_N = a0 + (N + 1) / 2
    E_mu, E_mu2 = mu_N, mu_N**2 + 1 / lam_N
    b_N = b0 + 0.5 * (np.sum(x**2) - 2 * E_mu * np.sum(x) + N * E_mu2
                      + lam0 * (E_mu2 - 2 * mu0 * E_mu + mu0**2))
    E_tau = a_N / b_N

# Exact posterior is Normal-Gamma: compare the marginal moments of mu.
lam_post = lam0 + N
mu_post = (lam0 * mu0 + N * xbar) / lam_post
a_post = a0 + N / 2
b_post = b0 + 0.5 * (np.sum((x - xbar) ** 2) + lam0 * N * (xbar - mu0) ** 2 / lam_post)
var_mu_exact = b_post / ((a_post - 1) * lam_post)       # Student-t marginal variance
print("E[mu]   VI %.4f  exact %.4f" % (mu_N, mu_post))
print("Var[mu] VI %.5f  exact %.5f" % (1 / lam_N, var_mu_exact))
print("E[tau]  VI %.3f   exact %.3f" % (a_N / b_N, a_post / b_post))
```

平均一致（皆為 1.8668），$\mathbb{E}[\tau]$ 也一致（2.146），但 μ 的變異數在 VI 下是 0.01503，精確值是 0.01603：小了約 6%。在精確後驗中，μ 與 τ 相依（μ 的分散程度隨 $1/\tau$ 縮放），其邊際分布是尾部較重的 Student-t；因子化的高斯無法表達這兩種性質。

這種低估是系統性的。對精確度矩陣為 Λ 的高斯目標，最佳的 mean-field 因子 q_j 的變異數是 $1/\Lambda_{jj}$，也就是條件變異數，永遠不大於邊際變異數。圖 4 顯示極端情況。

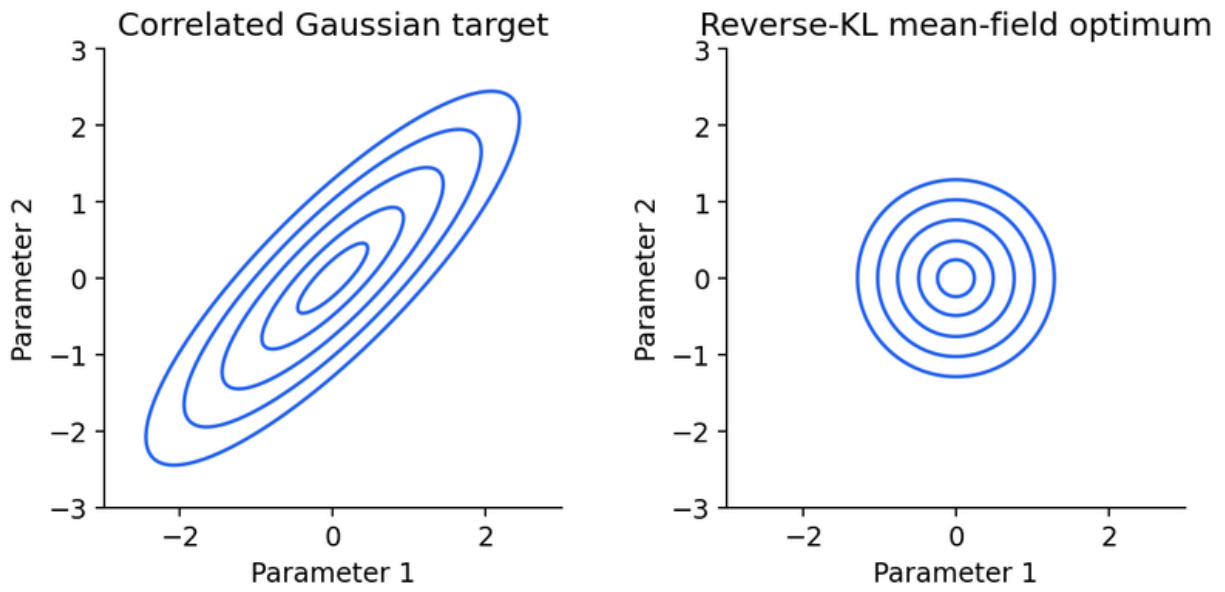


圖 *4* — 原創解析示意：目標的邊際變異數為一，相關係數為 *0.85*。在 *reverse KL* 下，最佳高斯 *mean-field* 的各因子變異數為 $1 - 0.85^2 \approx 0.28$，展示相關性的遺失與邊際不確定性的低估。等高線表示相對密度，不是模型效能。

用哪個 KL？找峰與覆蓋質量

KL 不對稱，方向不同答案就不同。標準 VI 使用的**反向 KL** $D_{\mathrm{KL}}(q\|p)$ 在 q 下取期望：它重罰把 q 的質量放在 p 極小的地方，卻不在意 q 忽略 p 的某些部分。結果是**找峰**（mode seeking）且**分散不足**。**正向 KL** $D_{\mathrm{KL}}(p\|q)$ 在 p 下取期望：它重罰漏掉 p 有質量的任何區域，因此 q 會伸展開來涵蓋一切，即**覆蓋質量**（mass covering），也可能把質量放在峰與峰之間的低谷。在高斯這類指數族之內，最小化正向 KL 恰好就是**動差匹配**：q 取 p 的平均與共變異數。

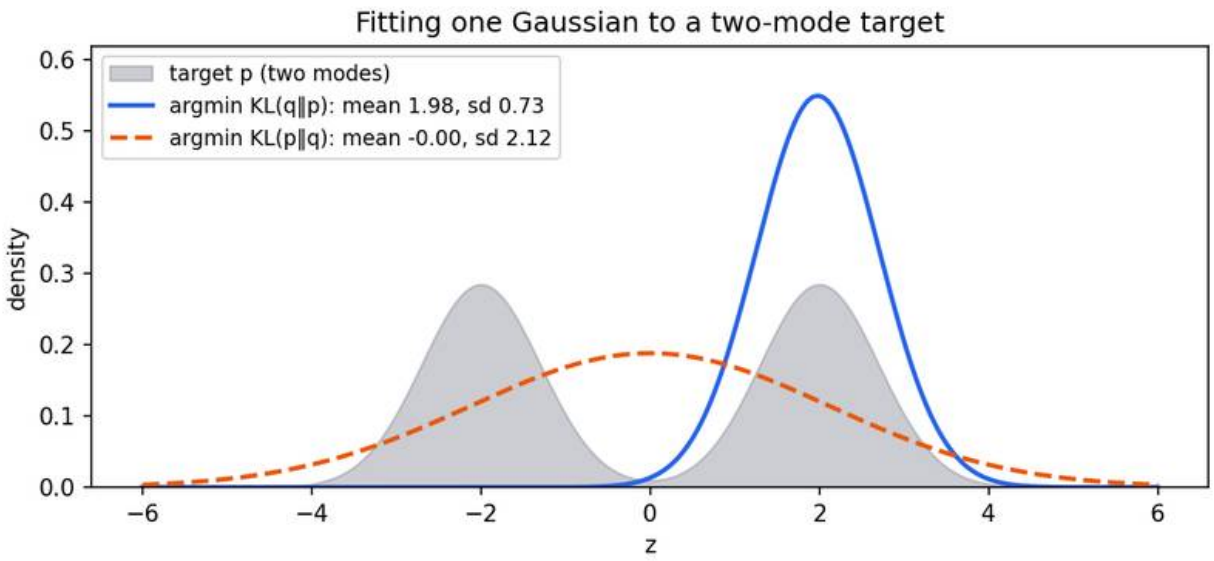


圖 5 — 以單一高斯擬合 $N(-2, 0.7^2)$ 與 $N(2, 0.7^2)$ 的等權混合。最小化反向 $KL(q\|p)$ 會鎖定其中一個峰（藍色；鎖定哪個峰取決於初始值）。最小化正向 $KL(p\|q)$ 則匹配混合分布的平均與變異數、涵蓋兩個峰，卻把峰值放在目標幾乎沒有質量之處（橘色虛線）。由本篇的圖形腳本以數值計算。

兩個方向都不是「唯一正確」的。反向 KL 好計算，答案銳利好用，但低估不確定性。正向 KL 是期望傳播背後的目標，也是以最大概似把模型擬合到資料時的目標；當漏掉一個峰的代價很高時，它比較安全。

隨機與黑箱 VI

CAVI 需要共軛性，而且每次更新都要掃過整個資料集。現代 VI 改用 ELBO 的**隨機梯度**，幾乎適用於任何模型，也能透過小批次（minibatch）抽樣處理大量資料 [11, 12]。$\nabla_\phi \mathcal{L}(\phi)$ 主要有兩種梯度估計量：

- **Score function（REINFORCE）估計量**，黑箱 VI 採用此法 [13]：

$$\nabla_\phi \mathcal{L} = \mathbb{E}_{q_\phi}\Big[\nabla_\phi \log q_\phi(z)\,\big(\log p(D,z) - \log q_\phi(z)\big)\Big].$$

它只需要計算 $\log p(D,z)$，也適用於離散的 z，但變異數高，需要控制變量（control variates）。

- **重參數化（pathwise）估計量**。把 z 寫成 $z = g_\phi(\varepsilon)$，其中雜訊 ε 來自固定分布，例如 $z = m + \sigma \odot \varepsilon$、$\varepsilon \sim \mathcal{N}(0, I)$。則

$$\nabla_\phi \mathcal{L} = \mathbb{E}_{\varepsilon}\Big[\nabla_\phi\big(\log p(D, g_\phi(\varepsilon)) - \log q_\phi(g_\phi(\varepsilon))\big)\Big].$$

它需要 $\log p$ 可微、z 連續，但變異數通常低得多。VAE 讓這個估計量廣為人知 [15]。

對參數為 $\phi = (m, s)$、$\sigma = e^s$ 的對角高斯，熵項等於 $\sum_j s_j$ 加上常數，所以它對 s 的梯度就是 1。套用到邏輯斯迴歸後驗：

```
import numpy as np

rng = np.random.default_rng(0)
n = 40
X = np.c_[np.ones(n), rng.normal(size=n)]
y = (rng.random(n) < 1 / (1 + np.exp(-X @ np.array([-0.5, 1.5])))).astype(float)
prior_var = 4.0

def grad_log_joint(W):                  # W: (S, 2) -> gradients (S, 2)
    p = 1 / (1 + np.exp(-W @ X.T))      # (S, n)
    return (y - p) @ X - W / prior_var

# Laplace: Newton's method to the MAP, then invert the Hessian.
w = np.zeros(2)
for _ in range(25):
    p = 1 / (1 + np.exp(-X @ w))
    H = X.T @ (X * (p * (1 - p))[:, None]) + np.eye(2) / prior_var
    w = w + np.linalg.solve(H, grad_log_joint(w[None])[0])
print("Laplace mean:", w.round(3), " sd:", np.sqrt(np.diag(np.linalg.inv(H))).round(3))

# Mean-field Gaussian VI with reparameterization: w = m + exp(s) * eps.
m, s = np.zeros(2), np.full(2, -1.0)
lr, S = 0.02, 64
for step in range(3000):
    eps = rng.normal(size=(S, 2))
    W = m + np.exp(s) * eps
    g = grad_log_joint(W)
    grad_m = g.mean(axis=0)                                  # d ELBO / d m
    grad_s = (g * eps).mean(axis=0) * np.exp(s) + 1.0        # + entropy term
    m += lr * grad_m
    s += lr * grad_s
print("VI mean:     ", m.round(3), " sd:", np.exp(s).round(3))
```

與網格的標準答案比較：平均 $(-0.303, 1.631)$、標準差 $(0.366, 0.578)$。程式輸出 Laplace 平均 $(-0.291, 1.500)$、標準差 $(0.355, 0.556)$；VI 平均 $(-0.303, 1.630)$、標準差 $(0.358, 0.562)$。VI 幾乎完全找回後驗平均，因為它對偏斜的後驗取了平均；Laplace 回報的是眾數，後驗偏斜時與平均不同。兩者都略為低估分散程度。40 個資料點時後驗接近高斯，差異不大；資料更少或類別可分時，差異會迅速擴大。

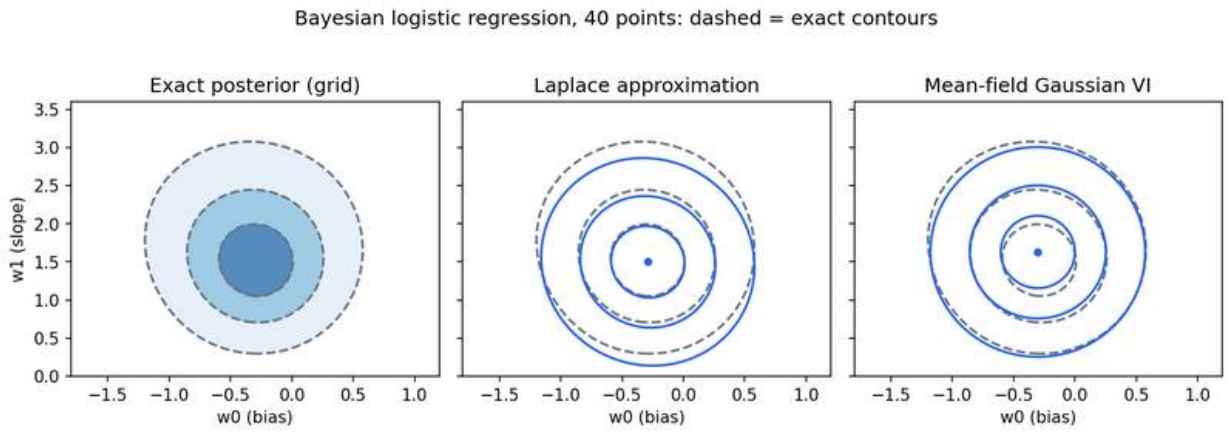


圖 *6* — 以 *40* 個合成資料點做兩權重的貝氏邏輯斯迴歸。灰色虛線：網格算出的精確後驗等高線。藍色：*Laplace*（以 *MAP* 為中心，往較小的斜率偏移）與 *mean-field* 高斯 *VI*（中心接近真正的後驗平均，略微太窄）。由本篇的圖形腳本計算。

Laplace 近似與期望傳播

Laplace：在峰頂放一個高斯

白話版。找到後驗的最高點，然後擬合一個在該處曲率相同的高斯。

精確版。在梯度為零的眾數 $\hat{z}$ 附近，把 $\log \tilde{p}(z)$ 展開到二階。令 $H = -\nabla^2 \log \tilde{p}(\hat{z})$ 為負對數後驗的 Hessian，

$$q(z) = \mathcal{N}(z \mid \hat{z},\, H^{-1}), \qquad \log p(D) \approx \log \tilde{p}(\hat{z}) + \frac{d}{2}\log 2\pi - \frac{1}{2}\log |H|,$$

其中 d 是 z 的維度。第二個式子是 **Laplace 證據近似**；只保留隨 N 成長的項，就得到 PR 07 中用來選擇混合成分數的 BIC 準則 [1]。Laplace 很便宜（一次最佳化加一個 Hessian），而且資料越多越準確，因為正則模型的後驗會越來越接近高斯。它在下列情況會失敗：後驗偏斜、眾數位於邊界上（例如變異數趨近零）、Hessian 奇異，以及多峰後驗（它只描述其中一個峰）。現代的「深度網路 Laplace」方法只對最後一層使用 Laplace，或採用結構化的 Hessian 近似，為訓練好的神經網路提供便宜的不確定性估計。

期望傳播

期望傳播（EP）[14] 處理可分解為 $p(z \mid D) \propto \prod_i f_i(z)$ 的後驗，例如每個資料點一個因子。它用指數族中的簡單**站點**（site）$\tilde{f}_i(z)$ 近似每個因子，使 $q(z) \propto \prod_i \tilde{f}_i(z)$，並逐一修正站點：

1. 移除站點 i，形成**空腔**（cavity）分布 $q^{\setminus i}(z) \propto q(z)/\tilde{f}_i(z)$。
2. 形成**傾斜**（tilted）分布 $\hat{p}_i(z) \propto f_i(z)\, q^{\setminus i}(z)$，即空腔乘上精確的因子。
3. 把新的 q 設為分布族中與 $\hat{p}_i$ 動差相同的成員，也就是最小化正向 KL $D_{\mathrm{KL}}(\hat{p}_i \| q)$。
4. 更新站點：$\tilde{f}_i \propto q^{\text{new}}/q^{\setminus i}$。

由於每一步在局部使用正向 KL，EP 傾向於覆蓋質量，變異數往往比 mean-field VI 校準得更好，例如在高斯過程分類中。代價是它沒有保證會改善的全域目標函數，可能不收斂，也可能產生無效（負變異數）的站點，需要阻尼（damping）[1, 14]。

證據與複雜度：貝氏版奧坎剃刀

白話版。一個什麼都能解釋的模型，把先驗的賭注分散得很薄，所以解釋你實際看到的資料時拿不到多少功勞。一個押注在這類資料上的較簡單模型，反而能拿到較多功勞。

精確版。模型 $\mathcal{M}$ 的證據是把概似對先驗積分：

$$p(D \mid \mathcal{M}) = \int p(D \mid \theta, \mathcal{M})\, p(\theta \mid \mathcal{M})\, d\theta.$$

彈性大的模型可以達到很高的峰值概似，但如果它大部分的先驗質量都落在擬合很差的參數上，積分仍然很小。Laplace 公式讓這一點一目了然：證據等於峰值乘上一個比較後驗體積與先驗體積的「奧坎因子」（Occam factor）。由此有兩點提醒。第一，證據取決於先驗，甚至取決於幾乎不影響後驗的面向，例如模糊先驗有多寬。第二，它回答的是「哪個模型最能預料到這份資料？」，這與「哪個模型最能預測新資料？」相關但不相同，後者由保留驗證回答（PR 09）。跨模型比較 ELBO 也要小心：每個 ELBO 與其證據的差距是不同且未知的 KL。

從 ELBO 到 VAE 與擴散模型

本章的工具是 ML 系列中兩類深度生成模型的骨幹。

VAE 是攤銷式（amortized）變分推論。變分自編碼器以潛在碼 z、先驗 $p(z)$ 與神經網路解碼器 $p_\theta(x \mid z)$ 為影像 x 建模。它不為每張影像各自最佳化一組變分參數，而是讓編碼器網路直接輸出這些參數：$q_\phi(z \mid x) = \mathcal{N}(\mu_\phi(x), \operatorname{diag} \sigma^2_\phi(x))$。這就是**攤銷式推論**：一個網路學會對任何輸入做推論。每張影像的目標函數正是 ELBO：

$$\mathcal{L}(\theta, \phi; x) = \mathbb{E}_{q_\phi(z \mid x)}[\log p_\theta(x \mid z)] - D_{\mathrm{KL}}(q_\phi(z \mid x) \,\|\, p(z)),$$

並以重參數化梯度對 θ 與 ϕ 聯合訓練 [15, 16]。它已知的弱點正是上面談過的：**攤銷差距**（編碼器的答案比逐張最佳化差）、反向 KL 的分散不足，以及會造成模糊的高斯解碼器。完整說明與程式碼見 ML 07。

擴散模型以 ELBO 訓練，並以 Langevin 式動力學取樣。去噪擴散機率模型固定一個逐步加入高斯雜訊的前向過程 $x_0 \to x_1 \to \dots \to x_T$，並學習反向過程 $p_\theta(x_{t-1} \mid x_t)$。前向的含雜訊狀態扮演潛在變數的角色，前向過程則是固定的變分後驗；訓練目標是 $-\log p_\theta(x_0)$ 的變分上界，常用的雜訊預測損失函數則是此上界的重新加權版本 [17]。Score-based 觀點 [18] 改為學習 $s_\theta(x, t) \approx \nabla_x \log p_t(x)$，也就是含雜訊資料對數密度的梯度，並透過積分反向時間的隨機微分方程來產生樣本；其 predictor–corrector 取樣器明確地穿插 Langevin MCMC 步驟，以 s_θ 取代 $\nabla \log \tilde{p}$。與本章前面的 Langevin 更新相比，唯一的差別是 score 是從資料學來的，而不是由公式算出來的。見 ML 08。

如何選擇方法

方法	回傳什麼	優點	主要失敗模式
網格／數值積分	數值近似	檢查網格加密後的低維參考值	截斷與離散化誤差；成本隨維度快速成長
重要性取樣	加權樣本、證據	簡單、可平行、能給出 $p(D)$	q 漏掉 p 時權重崩潰
隨機漫步 MH	相關樣本	只需要 $\tilde{p}$	高維時很慢；需調參
Gibbs	相關樣本	條件分布容易時不需調參	變數相關時很慢
HMC ／ NUTS	相關樣本	能擴展到大量連續參數	需要梯度；離散變數；divergence
Mean-field VI	分布 q	快速、可擴展、適合小批次	分散不足；局部最佳
Laplace	眾數上的高斯	便宜；可估計證據	偏斜、邊界、多峰
EP	高斯／指數族	變異數常較準	可能不收斂

合理的工作流程是：先用最佳化方法（MAP 或 VI）求速度，在較小版本的問題上用謹慎的取樣器對照檢查，再用已知參數模擬資料，確認整條流程能還原它應該還原的東西。

現代觀點

值得一讀的綜述。 Blei、Kucukelbir 與 McAuliffe 寫給統計學者的綜述 [11] 是 VI 的標準入門：它推導 ELBO 與 CAVI、完整走過高斯混合的例子、介紹隨機 VI，並明確指出 VI 以犧牲 MCMC 的漸近精確性換取速度，且傾向低估後驗變異數。Zhang、Bütepage、Kjellström 與 Mandt [12] 把後續文獻整理成四個方向：可擴展的 VI（隨機與分散式）、適用於非共軛模型的通用 VI（黑箱與重參數化梯度）、更準確的 VI（反向 KL 以外的散度、更豐富的分布族），以及攤銷式 VI（推論網路、VAE）。Jordan、Ghahramani、Jaakkola 與 Saul [10] 是圖模型變分方法的經典導論，至今仍是凸對偶觀點最清楚的來源。取樣方面，Neal 的專章 [6] 是 HMC 的標準推導，Betancourt [8] 提供 HMC 在高維為何有效的幾何圖像，Hoffman 與 Gelman [7] 描述 NUTS，也就是現代機率程式系統的預設取樣器。Vehtari 等人 [9] 訂定了目前收斂診斷的標準。Kingma 與 Welling 的專書 [16] 把 VAE 當作攤銷式 VI 加以綜述，Yang 等人 [19] 則綜述擴散模型，包括其變分與 score-based 表述。

2023–2026 的現況。有三個明顯趨勢。

1. 機器學習進入貝氏計算。 Winter、Campbell、Lin、Srivastava 與 Dunson [20] 回顧了正規化流（normalizing flows）、貝氏 coreset、分散式推論與變分方法如何被用來加速後驗計算，以及這些方法的理論保證在哪些地方仍弱於傳統 MCMC。
2. 擴散的想法被用於取樣，而不只是生成。即使只有未正規化的密度（沒有資料），仍然可以訓練擴散式的取樣器。Denoising Diffusion Samplers [21] 學習反轉一個加雜訊的過程，使其終點落在只知道差一個常數的目標上，得到近似樣本與正規化常數的估計；它與最佳控制及 Schrödinger bridge 有所連結，是快速成長的學習式取樣器家族中具代表性的一員。
3. 重視工作流程而非單一演算法。實務上已轉向預設執行 NUTS、回報秩正規化 $\hat{R}$ 與 bulk ／ tail ESS [9] 的機率程式系統，並把 VI 或 Laplace 當作快速的第一步，而不是最終答案。

開放問題。為非常大的模型提供可靠的不確定性仍然困難：全批次 MCMC 太昂貴，而便宜的近似（mean-field VI、最後一層 Laplace、集成）彼此結論不一。多峰後驗會同時擊敗局部 VI 與單一鏈。在沒有標準答案時評估近似的品質，大體上仍靠經驗法則；透過 ELBO 比較模型，也會受到未知近似差距的干擾。

重點整理

- 後驗在差一個常數的意義下容易計算、卻難以積分；網格成本是 G^d，超過幾個維度就非近似不可。
- Monte Carlo 誤差以 $1/\sqrt{S}$ 下降且與維度無關，但前提是能從正確的分布抽樣。重要性取樣修正錯誤的分布，權重的 ESS 告訴你修正何時已經崩潰。
- Metropolis–Hastings 只需要 $\tilde{p}$ 的比值，非對稱提議則需加上 Hastings 修正。Gibbs 是接受率為 1 的 MH。HMC 利用梯度走得更遠。一定要記錄被拒絕時的狀態、跑多條鏈，並檢查 $\hat{R}$ 與 ESS。
- VI 最大化 ELBO，等同最小化對後驗的反向 KL。它快速、可擴展、會找峰，並傾向低估變異數，在 mean field 下尤其如此。
- Laplace 在眾數上擬合高斯，提供便宜的證據估計（以及 BIC）；EP 在局部匹配動差，變異數常較準，但可能不收斂。
- 證據實現了奧坎剃刀，但取決於先驗，回答的問題也與保留資料上的預測不同。
- VAE 是搭配重參數化梯度的攤銷式 VI；擴散模型以變分上界訓練，並以學習到的 score 驅動類似 Langevin 的動力學來取樣。

練習

1. **手推 ELBO**。從 $D_{\mathrm{KL}}(q(z) \| p(z \mid D))$ 出發，推導 $\log p(D) = \mathcal{L}(q) + D_{\mathrm{KL}}$。再證明 $q(z) = p(z \mid D)$ 時下界是緊的，並說明這如何讓 EM 成為最大化 ELBO 的特例。

   提示

   代入 $\log p(z \mid D) = \log p(D, z) - \log p(D)$，並因為 $\log p(D)$ 與 z 無關而把它移出期望值。q 等於後驗時 KL 為零。EM 的 E step 把 q 設為目前參數下潛在變數的後驗（下界變緊）；M step 固定 q，對參數最大化 $\mathcal{L}$。

2. **Hastings 修正**。你對一個正值參數使用對數常態隨機漫步提議：$z' = z\,e^{\eta}$，$\eta \sim \mathcal{N}(0, s^2)$。計算 $q(z' \mid z)$ 與比值 $q(z \mid z')/q(z' \mid z)$。若直接使用 Metropolis 原本的規則，會出什麼問題？

   提示

   $q(z' \mid z) = \frac{1}{z' s\sqrt{2\pi}} \exp(-(\log z' - \log z)^2/(2s^2))$。指數部分對稱，所以比值為 z'/z。省略它會讓鏈從正比於 $\tilde{p}(z)/z$ 的密度取樣，偏向較小的值。

3. **步長與混合**。在 Metropolis 程式中，把提議標準差改為 0.02，再改為 8。對每種情況回報接受率、lag-1 自相關，並用診斷小節的公式估計 N_{eff}（在第一個負的自相關處截斷總和）。哪個步長每次迭代得到的有效樣本最多？

提示

對照圖 2。極小的步長接受率接近 1，但自相關也接近 1，所以 N_{eff} 非常小，鏈甚至可能還沒到達目標。極大的步長接受率低，會長時間重複同一個值。中間的步長（約為目標的標準差）能得到多得多的有效樣本。

4. **斜脊上的 Gibbs**。以 $\rho = 0.99$ 重新執行 Gibbs 取樣器。先預測 z_1 的 lag-1 自相關與 N_{eff}/N，再用模擬驗證。提出並實作一個修正方法。

提示

z_1 鏈是係數為 $\rho^2 \approx 0.98$ 的自迴歸過程，所以 $N_{\text{eff}}/N = (1-\rho^2)/(1+\rho^2) \approx 0.01$。修正方法：把 (z_1, z_2) 當成一個區塊一起抽，或旋轉到 $u = (z_1+z_2)/\sqrt{2}$、$v = (z_1-z_2)/\sqrt{2}$，兩者在目標下彼此獨立。

5. **Mean field 低估變異數**。對高斯目標 $\mathcal{N}(0, \Sigma)$，精確度 $\Lambda = \Sigma^{-1}$，用 mean-field 更新規則證明最佳的 q_j 是變異數為 $1/\Lambda_{jj}$ 的高斯。證明 $1/\Lambda_{jj} \le \Sigma_{jj}$，並在變異數為一、相關係數 0.85 時計算兩者。

提示

$\log p = -\tfrac{1}{2} z^\top \Lambda z + \text{常數}$；對 q_{-j} 取期望，保留含 z_j 的項：$-\tfrac{1}{2}\Lambda_{jj} z_j^2 - z_j \sum_{k \neq j} \Lambda_{jk} \mathbb{E}[z_k]$，這是精確度為 Λ_{jj} 的高斯。$1/\Lambda_{jj}$ 是給定其餘變數時 z_j 的條件變異數，由 Schur 補數可知它等於 Σ_{jj} 減去一個非負項。$\rho = 0.85$ 時：約 0.28 對 1。

6. **用重要性取樣估計證據**。對兩權重邏輯斯迴歸，分別以 (a) 先驗作為提議分布、(b) 共變異數放大 1.5 倍的 Laplace 高斯作為提議分布，以重要性取樣估計 $\log p(D)$。把兩者與網格值 -22.117 及權重 ESS 比較。在 50 維時你會信任哪一個？為什麼？

提示

估計值為 $\log \frac{1}{S}\sum_s \tilde{p}(z_s)/q(z_s)$，用 log-sum-exp 計算。先驗比後驗寬得多，大部分樣本權重微不足道，ESS 只占 S 的一小部分；以 Laplace 為基礎的提議分布落在後驗上，ESS 大得多。在高維時，先驗與後驗的落差呈指數擴大，以先驗為提議分布會變得毫無用處，而擬合良好、尾部略重的提議分布仍然可行。

參考文獻

1. C. M. Bishop, *Pattern Recognition and Machine Learning*, Springer, 2006, Sections 4.4 (Laplace), 10.1–10.2 and 10.7 (variational inference, EP) and 11.1–11.5 (sampling). 官方書籍頁面與 PDF
2. S. Theodoridis and K. Koutroumbas, *Pattern Recognition*, 4th ed., Academic Press, 2009, Chapter 2 (Bayes decision theory and parameter estimation) and Appendix C (cost function optimization)。這些是先備知識，並非與本章內容完全相同。doi
3. N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller and E. Teller, “Equation of state calculations by fast computing machines,” *The Journal of Chemical Physics*, vol. 21, no. 6, pp. 1087–1092, 1953. doi

4. W. K. Hastings, “Monte Carlo sampling methods using Markov chains and their applications,” *Biometrika*, vol. 57, no. 1, pp. 97–109, 1970. doi
5. S. Geman and D. Geman, “Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-6, no. 6, pp. 721–741, 1984. doi
6. R. M. Neal, “MCMC using Hamiltonian dynamics,” in S. Brooks, A. Gelman, G. Jones and X.-L. Meng (eds.), *Handbook of Markov Chain Monte Carlo*, Chapman and Hall/CRC, 2011 (arXiv:1206.1901). arXiv
7. M. D. Hoffman and A. Gelman, “The No-U-Turn Sampler: Adaptively setting path lengths in Hamiltonian Monte Carlo,” *Journal of Machine Learning Research*, vol. 15, pp. 1593–1623, 2014. JMLR
8. M. Betancourt, “A conceptual introduction to Hamiltonian Monte Carlo,” arXiv:1701.02434, 2017. arXiv
9. A. Vehtari, A. Gelman, D. Simpson, B. Carpenter and P.-C. Bürkner, “Rank-normalization, folding, and localization: An improved $\hat{R}$ for assessing convergence of MCMC (with discussion),” *Bayesian Analysis*, vol. 16, no. 2, 2021. doi
10. M. I. Jordan, Z. Ghahramani, T. S. Jaakkola and L. K. Saul, “An introduction to variational methods for graphical models,” *Machine Learning*, vol. 37, no. 2, pp. 183–233, 1999. doi
11. D. M. Blei, A. Kucukelbir and J. D. McAuliffe, “Variational inference: A review for statisticians,” *Journal of the American Statistical Association*, vol. 112, no. 518, pp. 859–877, 2017. doi
12. C. Zhang, J. Bütepage, H. Kjellström and S. Mandt, “Advances in variational inference,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 41, no. 8, pp. 2008–2026, 2019. doi
13. R. Ranganath, S. Gerrish and D. M. Blei, “Black box variational inference,” in *Proc. International Conference on Artificial Intelligence and Statistics* (*AISTATS*), PMLR vol. 33, 2014. PMLR
14. T. P. Minka, “Expectation propagation for approximate Bayesian inference,” in *Proc. Conference on Uncertainty in Artificial Intelligence* (*UAI*), 2001 (arXiv:1301.2294). arXiv
15. D. P. Kingma and M. Welling, “Auto-encoding variational Bayes,” *International Conference on Learning Representations* (*ICLR*), 2014; arXiv:1312.6114. arXiv
16. D. P. Kingma and M. Welling, “An introduction to variational autoencoders,” *Foundations and Trends in Machine Learning*, vol. 12, no. 4, pp. 307–392, 2019. arXiv
17. J. Ho, A. Jain and P. Abbeel, “Denoising diffusion probabilistic models,” in *Advances in Neural Information Processing Systems* (*NeurIPS*), 2020 (arXiv:2006.11239). arXiv
18. Y. Song, J. Sohl-Dickstein, D. P. Kingma, A. Kumar, S. Ermon and B. Poole, “Score-based generative modeling through stochastic differential equations,” in *Proc. International Conference on Learning Representations* (*ICLR*), 2021. arXiv
19. L. Yang, Z. Zhang, Y. Song, S. Hong, R. Xu, Y. Zhao, W. Zhang, B. Cui and M.-H. Yang, “Diffusion models: A comprehensive survey of methods and applications,” *ACM Computing Surveys*, vol. 56, no. 4, 2023. doi
20. S. Winter, T. Campbell, L. Lin, S. Srivastava and D. B. Dunson, “Machine learning and the future of Bayesian computation,” arXiv:2304.11251, 2023. arXiv
21. F. Vargas, W. Grathwohl and A. Doucet, “Denoising diffusion samplers,” in *Proc. International Conference on Learning Representations* (*ICLR*), 2023 (arXiv:2302.13834). arXiv

PR 09・集成、驗證與分群有效性

推導偏差–變異數分解、bagging、隨機森林、AdaBoost 與梯度提升，再以交叉驗證、bootstrap、巢狀與群組隔離驗證及成對統計檢定，誠實估計並比較辨識器。

網站版本: https://blog.shyandram.dev/zh-tw/learn/pattern-recognition/ensembles-validation/

你將學到

- 推導平方誤差的偏差–變異數分解（bias–variance decomposition），並用它說明為什麼平均能幫助不穩定的學習器。
- 說明決策樹如何切分特徵空間，以及 bagging 與隨機森林（random forest）如何降低其變異數，包括 out-of-bag 估計。
- 從指數損失推導 AdaBoost，連結到梯度提升（gradient boosting），並區分 boosting、投票、貝氏模型平均與 stacking。
- 以保留法（hold-out）、k 折、群組隔離與巢狀交叉驗證，以及 bootstrap（.632 與.632+），估計整個辨識程序的表現。
- 以 McNemar 檢定、5×2cv 與修正重抽樣 t 檢定，以及 Friedman/Nemenyi 程序比較兩個以上的分類器，並知道各自何時成立。
- 同時評估決策與機率，並把分群有效性視為一種模型選擇問題。

先看全貌

單一辨識器犯錯的原因有兩種，性質截然不同。有時它太簡單，表達不了真正的決策規則；有時它太有彈性，把這份訓練集的偶然細節也背了下來。集成方法組合許多模型，分別對付第二種問題（bagging、隨機森林）或第一種問題（boosting）。驗證則是故事的另一半：一旦我們能建立大量模型、調整大量旋鈕，就需要可信的方法估計「被選中的程序」在新資料上的表現，並判斷某個程序是否真的比另一個好。

> **用最簡單的話說**
>
> 想像大家一起猜罐子裡有幾顆糖。一位朋友的猜測可能差很多；但如果十位朋友**各自獨立**猜，再取平均，離譜的猜測會互相抵消，平均通常更接近。可是如果十位朋友都只是照抄最大聲那位的答案，平均就完全沒有幫助。現在假設你想知道誰最會猜：你必須拿一個他們**沒看過**的新罐子來考；用他們已經研究過的罐子來考，每個人看起來都會很厲害。

為什麼重要：樹集成至今仍是表格型特徵（包括手工設計的影像描述子）上最強的方法之一 [23]；深度網路集成是取得不確定性估計的標準作法 [24]；而幾乎每一篇比較辨識器的論文，都建立在交叉驗證與統計檢定之上。Theodoridis 與 Koutroumbas 在第 4 章討論決策樹、分類器組合與 boosting，第 10 章（「Supervised Learning: The Epilogue」）討論誤差估計與比較，第 16 章討論分群有效性 [1]。Bishop 第 14 章區分了數種數學上不同的模型組合方式，第 3.2 節則推導偏差–變異數分解 [2]。分類器本身已在 PR 03 與 PR 04 介紹過。

偏差、變異數與雜訊

白話版。想像用同一個學習演算法，在許多份來自同一來源的不同訓練集上重新訓練。**平均**預測仍可能偏離真值，這個系統性的偏離就是**偏差**（bias）。個別預測會在平均值周圍散開，這個散布就是**變異數**（variance）。即使是完美的預測器，也無法消除標籤本身的隨機性，那就是**雜訊**（noise）。

精確版。令目標為 $y = f(x) + \varepsilon$，其中 f 是未知的迴歸函數，ε 是雜訊，滿足 $\mathbb{E}[\varepsilon] = 0$、$\mathrm{Var}(\varepsilon) = \sigma^2$，且與其他一切獨立。學習演算法把隨機的訓練集 D 映射成預測器 $\hat{f}_D$。記 $\bar{f}(x) = \mathbb{E}_D[\hat{f}_D(x)]$ 為對訓練集平均後的預測。在固定輸入 x 上，對訓練集與測試雜訊取期望的平方誤差為

$$\begin{aligned}\mathbb{E}_{D,\varepsilon}\big[(y - \hat{f}_D(x))^2\big] &= \mathbb{E}\big[(\varepsilon + f(x) - \bar{f}(x) + \bar{f}(x) - \hat{f}_D(x))^2\big] \\ &= \underbrace{\sigma^2}_{\text{noise}} + \underbrace{\big(f(x) - \bar{f}(x)\big)^2}_{\text{bias}^2} + \underbrace{\mathbb{E}_D\big[(\hat{f}_D(x) - \bar{f}(x))^2\big]}_{\text{variance}}.\end{aligned}$$

三個交叉項都會消失：ε 期望為零且與 D 獨立；$f(x) - \bar{f}(x)$ 是常數；而依 $\bar{f}$ 的定義，$\mathbb{E}_D[\bar{f}(x) - \hat{f}_D(x)] = 0$。再對輸入分布 $p(x)$ 取平均，就得到期望測試誤差。Geman、Bienenstock 與 Doursat 把這個張力稱為**偏差／變異數兩難**（bias/variance dilemma）[3]：讓模型更有彈性，通常會降低偏差、提高變異數，最佳複雜度在兩者之間取得平衡。

下面的程式以暴力法，對雜訊正弦曲線上不同深度的迴歸樹估計每一項。程式抽出 200 份各含 60 點的訓練集；200 棵擬合樹的經驗平均與散布，給出偏差平方與變異數。

```
import numpy as np
from sklearn.tree import DecisionTreeRegressor

rng = np.random.default_rng(0)
f = lambda x: np.sin(2 * np.pi * x)          # true regression function
sigma = 0.3                                   # noise standard deviation
x_test = np.linspace(0, 1, 200)[:, None]

def bias_variance(depth, n_sets=200, n=60):
    preds = np.empty((n_sets, len(x_test)))
    for s in range(n_sets):                   # many independent training sets
        x = rng.random((n, 1))
        y = f(x[:, 0]) + rng.normal(0, sigma, n)
        tree = DecisionTreeRegressor(max_depth=depth, random_state=0)
        preds[s] = tree.fit(x, y).predict(x_test)
    mean_pred = preds.mean(axis=0)
    bias2 = np.mean((mean_pred - f(x_test[:, 0])) ** 2)
    var = np.mean(preds.var(axis=0))
    return bias2, var

for depth in [1, 2, 3, 5, 8, None]:
    b2, v = bias_variance(depth)
    print(f"depth={str(depth):>4}  bias^2={b2:.4f}  variance={v:.4f}  "
          f"expected test MSE~{b2 + v + sigma**2:.4f}")
```

在我們的執行中，深度 1 的偏差平方約 0.074、變異數約 0.035；深度 3 的總和最低（偏差平方

約 0.003、變異數約 0.045）；完全長成的樹幾乎沒有偏差，但變異數約 0.092，和雜訊變異數 $\sigma^2 = 0.09$ 本身一樣大。圖 1 以視覺方式呈現同一個實驗。

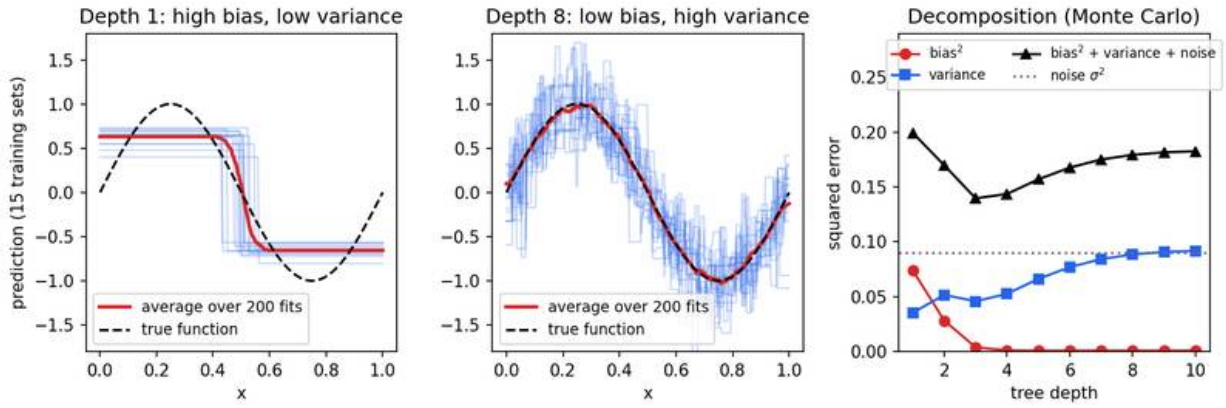


圖 *1* — 迴歸樹在 $y = sin(2\pi x)$ + 雜訊（$\sigma = 0.3$，每份訓練集 *60* 點，共 *200* 份訓練集）上的偏差–變異數分解。淺樹總是錯在同一個方向（偏差）；深樹平均起來是對的，但每一棵都很吵（變異數）。由本章的腳本產生。

有三點提醒讓這張圖保持誠實。第一，乾淨的分解只適用於平方誤差；對 0–1 分類損失，偏差與變異數有好幾種互相競爭的定義，而且兩者可能以非加法方式交互作用，因此對分類器而言，請把分解當作直覺，而不是精確的帳目。第二，「複雜度」不只是參數數量：樹深、正則化強度、boosting 回合數與提早停止（early stopping）都會讓模型在偏差–變異數軸上移動。第三，極大的過參數化模型，其測試誤差曲線可能在容量超過內插點後先升再降，也就是 Belkin 等人記錄的「雙重下降」（double descent）形狀 [25]。分解在每一點上仍然成立；改變的是變異數隨容量增長的行為。

僅線上版：雙重下降：*Belkin* 等人（*PNAS 2019*）閱讀指南 — *https://blog.shyandram.dev/zh-tw/learn/pattern-recognition/ensembles-validation/papers/雙重下降 belkin-等人 pnas-2019 閱讀指南/*

決策樹與空間切分

白話版。決策樹對特徵問一連串是非題，例如「特徵 3 是否小於 0.7？」，每個最終答案格（葉節點）存放一個預測。它是低偏差、高變異數學習器的經典例子，這也正是它成為集成方法最愛用的積木的原因。

精確版。決策樹以 $x_j \le b$ 等條件遞迴分割觀測。每個葉節點估計一個類別分布（分類）或一個常數（迴歸）。對類別比例為 p_k 的分類節點，**Gini 不純度**為

$$G = 1 - \sum_{k} p_k^2,$$

熵則為 $H = -\sum_k p_k \log p_k$。兩者在純節點為零，在均勻混合時最大。把父節點 P 分成子節點 L 與 R，分別含有父節點 n_P 個樣本中的 n_L 與 n_R 個，此分割以不純度下降量評分：

$$\Delta = G(P) - \frac{n_L}{n_P} G(L) - \frac{n_R}{n_P} G(R),$$

學習器選擇 Δ 最大的特徵與閾值。這是局部貪婪選擇：最好的單一分割不一定通往最好的樹。深樹能捕捉交互作用與非線性、與座標軸平行的邊界，但少數訓練點改變就可能讓整棵樹大幅改變；這種不穩定，正是圖 1 中看到的高變異數。

深度限制、最小葉節點樣本數與成本–複雜度剪枝可以控制複雜度。由於分割只取決於每個特徵的排序，樹對單調尺度轉換不變，因此比距離模型較不受特徵尺度影響；但缺失值、特徵定義，以及只由訓練資料擬合的轉換，仍需要仔細處理。Theodoridis 與 Koutroumbas（第 4 章）與 Bishop（第 14.4 節）都把樹視為非線性分類器的一個家族 [1], [2]。

Bagging 與隨機森林

白話版。 Bagging（bootstrap aggregating）以「放回抽樣」從原始資料建立許多份訓練集，在每份上訓練一個不穩定的模型，再平均或投票。隨機森林再加入一個隨機來源：每次分割時只看隨機挑出的一部分特徵，讓樹之間更不一致，平均效果也就更好。

精確版。 Breiman 提出 bagging，作為改善**不穩定**程序的方法，也就是訓練集稍有擾動、輸出就大幅改變的程序 [4]。給定 B 份大小皆為 n 的 bootstrap 樣本 $D^{(1)}, \dots, D^{(B)}$，bagging 迴歸器為 $\hat{f}_{\text{bag}}(x) = \frac{1}{B}\sum_{b} \hat{f}_{D^{(b)}}(x)$，bagging 分類器則採多數決（或平均類別機率）。

為什麼平均有幫助？假設在固定的 x 上，M 個純量預測器的變異數都是 σ^2，任兩者的相關係數都是 ρ。展開總和的變異數：

$$\operatorname{Var}\Big(\frac{1}{M}\sum_{m=1}^{M} f_m\Big) = \frac{1}{M^2}\Big[M\sigma^2 + M(M-1)\rho\sigma^2\Big] = \sigma^2\left[\rho + \frac{1-\rho}{M}\right].$$

第二項隨 M 增大而消失，第一項 $\rho\sigma^2$ 卻不會：彼此相關的錯誤形成一個下限（圖 2 左）。平均基本上不改變偏差，所以 bagging 對深樹這類低偏差、高變異數的學習器幫助最大，對線性判別式或大 k 的 k 近鄰這類穩定學習器幫助很小。這個示意公式說明了相關錯誤為何限制改善幅度；它並不保證每個分類集成都會提高準確率。

隨機森林直接降低 ρ [5]。在每份 bootstrap 樣本上長樹時，每次分割只考慮隨機選出的 m_{try} 個特徵（分類問題常見的預設是 d 個特徵中取 $\sqrt{d}$ 個）。Breiman 證明森林的泛化誤差可以用個別樹的**強度**與樹之間的**相關性**來界定 [5]，這和變異數公式是同一個取捨：較弱但相關性較低的樹，可能組成更強的森林。

Out-of-bag（OOB）估計。在 n 次放回抽樣中，某一筆觀測**沒有**被抽到的機率是 $(1-1/n)^n \to e^{-1} \approx 0.368$。因此每棵樹大約漏掉 37% 的資料，而每筆觀測都可以由大約三分之一沒見過它的樹來預測。彙整這些預測，幾乎不必額外成本就能得到 OOB 誤差估計 [5]。在圖 2 右，樹夠多之後，OOB 估計就會貼近測試誤差。但一般的 bootstrap 是對「列」重抽樣：如果好幾列屬於同一位受試者、書寫者或同一次錄製，一棵樹可能在某人的一張影像上訓練，又在同一人的另一張影像上做 OOB「測試」。這時對新群組部署而言，OOB 分數就會過度樂觀。

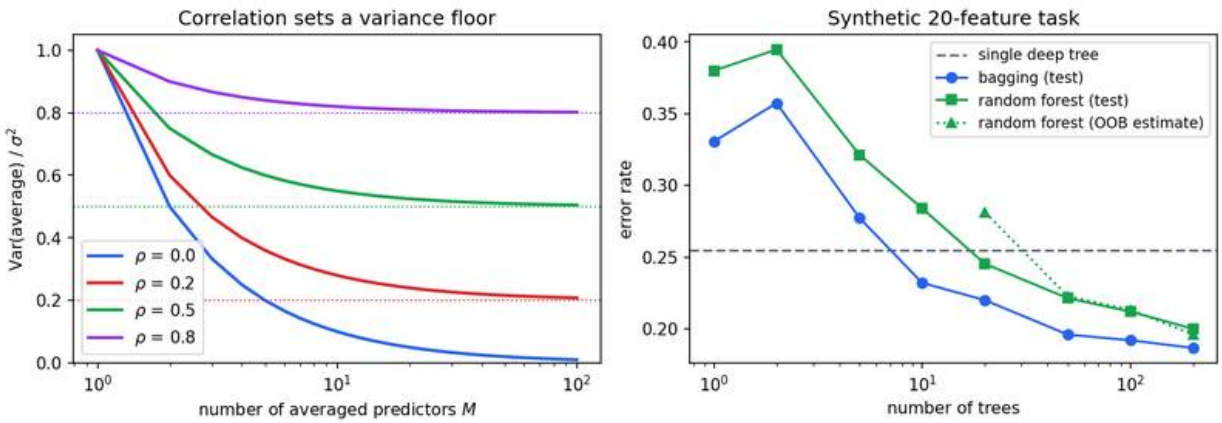


圖 2 — 左：M 個等相關預測器平均後的解析變異數。右：一個合成的 *20* 特徵任務（訓練 *750*、測試 *750* 筆）。兩種集成在有數十棵樹之後都勝過單一深樹；在這個特定任務上，單純 *bagging* 略優於隨機森林，提醒我們隨機特徵子集不保證一定有幫助。樹非常少（*1–2* 棵）時投票不穩定，也可能平手。

互動示範（網站版）：請開啟本篇教學的網站版本，操作此示範。開啟示範

Boosting 改變下一個學習器的工作

白話版。 Bagging 的成員彼此獨立、可以平行訓練。Boosting 則**依序**訓練：每個新的弱學習器專注在目前集成仍然答錯的樣本上，最後的預測是加權總和。Bagging 主要降低變異數；boosting 主要降低偏差，把許多「只比亂猜好一點」的規則變成強分類器。

精確版。 Boosting 建立加法模型

$$F_M(x) = \sum_{m=1}^{M} \alpha_m h_m(x),$$

其中每個 h_m 是弱學習器（常用深度 1 的樹，即**樹樁** stump），α_m 是它的權重。對標籤 $y \in \{-1, +1\}$，類別預測為 $\operatorname{sign} F_M(x)$。

AdaBoost 演算法

Freund 與 Schapire 的 AdaBoost [6] 對訓練資料 (x_i, y_i)，$i = 1, \dots, n$，運作如下：

1. 從均勻權重 $w_i^{(1)} = 1/n$ 開始。
2. 對 $m = 1, \dots, M$：
   - 以權重 $w^{(m)}$ 擬合 h_m；
   - 計算加權誤差 $\epsilon_m = \sum_i w_i^{(m)} \mathbf{1}[h_m(x_i) \neq y_i] / \sum_i w_i^{(m)}$；
   - 令 $\alpha_m = \frac{1}{2} \log \frac{1-\epsilon_m}{\epsilon_m}$；

- 更新 $w_i^{(m+1)} \propto w_i^{(m)} \exp\big(-\alpha_m y_i h_m(x_i)\big)$ 並重新正規化。

3. 輸出 $\operatorname{sign}\sum_m \alpha_m h_m(x)$。

被分錯的點滿足 $y_i h_m(x_i) = -1$，權重乘上 e^{α_m}；分對的點則乘上 $e^{-\alpha_m}$。$\epsilon_m < 0.5$ 的學習器得到正的投票權，而且越準，票越重。

從指數損失推導 AdaBoost

Friedman、Hastie 與 Tibshirani 證明 AdaBoost 就是以**指數損失** $L(y, F) = \exp(-yF)$ 進行的**前向逐步加法建模**（forward stagewise additive modeling）[7]。假設 F_{m-1} 已固定，我們加入一項 αh。訓練損失為

$$\sum_{i=1}^{n} \exp\big(-y_i[F_{m-1}(x_i) + \alpha h(x_i)]\big) = \sum_{i=1}^{n} w_i^{(m)} \exp\big(-\alpha y_i h(x_i)\big), \qquad w_i^{(m)} = \exp\big(-y_i F_{m-1}(x_i)\big).$$

權重 $w_i^{(m)}$ 與 α、h 都無關。把總和拆成分對與分錯的點：

$$\sum_i w_i^{(m)} e^{-\alpha y_i h(x_i)} = e^{-\alpha} \sum_{y_i = h(x_i)} w_i^{(m)} + e^{\alpha} \sum_{y_i \neq h(x_i)} w_i^{(m)} = (e^{\alpha} - e^{-\alpha}) \sum_i w_i^{(m)} \mathbf{1}[y_i \neq h(x_i)] + e^{-\alpha} \sum_i w_i^{(m)}.$$

第 *1* 步，選 h。對任何 $\alpha > 0$，第一個係數為正，所以最佳的 h 就是最小化**加權訓練誤差**的那一個，正好是「以權重 $w^{(m)}$ 擬合 h_m」。

第 *2* 步，選 α。除以 $\sum_i w_i^{(m)}$，並以 ϵ 表示正規化後的加權誤差。我們要最小化 $g(\alpha) = e^{-\alpha}(1-\epsilon) + e^{\alpha}\epsilon$。令 $g'(\alpha) = -e^{-\alpha}(1-\epsilon) + e^{\alpha}\epsilon = 0$，得到 $e^{2\alpha} = (1-\epsilon)/\epsilon$，也就是

$$\alpha_m = \frac{1}{2} \log \frac{1 - \epsilon_m}{\epsilon_m}.$$

第 *3* 步，更新權重。下一輪權重為 $w_i^{(m+1)} = \exp(-y_i F_m(x_i)) = w_i^{(m)} \exp(-\alpha_m y_i h_m(x_i))$，正是 AdaBoost 的更新式。演算法的每一步都能從指數損失的貪婪最小化推回來。

由此得到兩個結果。因為 $\mathbf{1}[yF < 0] \le e^{-yF}$，訓練誤差以平均指數損失為上界；採用最佳 α_m 時，只要 $\epsilon_m < 0.5$，每一回合都會把這個上界乘上 $2\sqrt{\epsilon_m(1-\epsilon_m)} < 1$，所以只要每個弱學習器都以某個差距勝過亂猜，訓練誤差就會指數下降 [6]。另外，期望指數損失的最小化者是 $F^*(x) = \frac{1}{2}\log\frac{P(y=1\mid x)}{P(y=-1\mid x)}$，即對數勝算（log-odds）的一半 [7]，因此 $\sigma(2F_M(x))$ 是一個機率估計（不一定校準良好）。指數權重也解釋了一個已知弱點：標錯的點會得到指數成長的權重，所以 AdaBoost 可能對標籤雜訊與離群值很敏感。

下面的程式用加權樹樁從零實作 AdaBoost，並與 scikit-learn 的 `AdaBoostClassifier` 比對。對兩類問題，scikit-learn 的 SAMME 變體使用 $\alpha = \log\frac{1-\epsilon}{\epsilon}$，是我們的兩倍；這會把 F 放大，但每個符號、以及重新正規化後的每個權重都不變。

```
import numpy as np
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import AdaBoostClassifier
```

```
X, y01 = make_classification(n_samples=600, n_features=10, n_informative=4,
                             flip_y=0.05, random_state=1)
y = 2 * y01 - 1                                   # labels in {-1, +1}
Xtr, Xte, ytr, yte = train_test_split(X, y, test_size=0.3, random_state=1)

def adaboost_fit(X, y, M=100):
    n = len(y)
    w = np.full(n, 1 / n)                         # step 1: uniform weights
    stumps, alphas = [], []
    for m in range(M):
        h = DecisionTreeClassifier(max_depth=1, random_state=0)
        h.fit(X, y, sample_weight=w)              # step 2a: weighted fit
        pred = h.predict(X)
        err = np.sum(w * (pred != y)) / np.sum(w) # step 2b: weighted error
        err = np.clip(err, 1e-12, 1 - 1e-12)
        alpha = 0.5 * np.log((1 - err) / err)     # step 2c: learner weight
        w = w * np.exp(-alpha * y * pred)         # step 2d: reweight
        w /= w.sum()
        stumps.append(h); alphas.append(alpha)
    return stumps, np.array(alphas)

def adaboost_score(stumps, alphas, X):
    return sum(a * h.predict(X) for h, a in zip(stumps, alphas))

stumps, alphas = adaboost_fit(Xtr, ytr)
F_tr = adaboost_score(stumps, alphas, Xtr)
F_te = adaboost_score(stumps, alphas, Xte)
print("from scratch: train err", np.mean(np.sign(F_tr) != ytr),
      " test err", np.mean(np.sign(F_te) != yte))
print("exp-loss bound on train err:", np.mean(np.exp(-ytr * F_tr)))

ref = AdaBoostClassifier(DecisionTreeClassifier(max_depth=1, random_state=0),
                         n_estimators=100, random_state=0).fit(Xtr, ytr)
agree = np.mean(ref.predict(Xte) == np.sign(F_te))
print("sklearn test err", np.mean(ref.predict(Xte) != yte),
      " agreement with ours:", agree)
```

在 scikit-learn 1.6 下，我們的執行印出訓練誤差 0.076、指數損失上界 0.384，兩種實作的測試誤差都是 0.156，測試預測 100% 一致。圖 3 顯示 300 回合的曲線，以及樹樁在二維中拼出的階梯狀邊界。

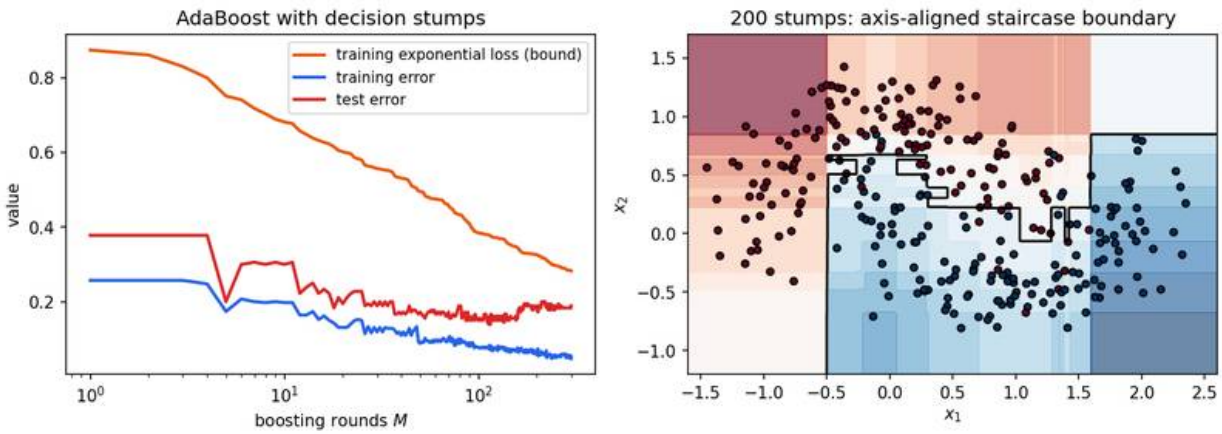


圖 3 — 以決策樹樁實作的 *AdaBoost*（我們從零寫的版本）。左：指數損失是訓練誤差的上界，而且持續縮小；測試誤差趨於平穩，在有標籤雜訊時可能稍微回升。右：把與座標軸平行的樹樁相加，在雙月形資料上得到階梯狀的決策邊界。

梯度提升

白話版。梯度提升把 AdaBoost 的想法推廣到任何可微分的損失：每一回合先算出每個訓練預測應該往哪個方向移動才能降低損失，再用一棵小樹去擬合這些方向。

精確版。 Friedman 的梯度提升機（gradient boosting machine）[8] 是在函數空間中做梯度下降。在第 m 回合，計算**偽殘差**（pseudo-residuals）

$$r_{im} = -\left.\frac{\partial L(y_i, F)}{\partial F}\right|_{F=F_{m-1}(x_i)},$$

以最小平方法把迴歸樹 h_m 擬合到 (x_i, r_{im})，用線搜尋選步長 ρ_m（或每個葉節點各取一個最佳常數，即 Friedman 的 TreeBoost），再以**學習率**（收縮，shrinkage）$0 < \nu \le 1$ 更新 $F_m = F_{m-1} + \nu \rho_m h_m$。對平方誤差 $L = \tfrac{1}{2}(y-F)^2$，偽殘差就是一般殘差 $y_i - F_{m-1}(x_i)$。對 $y \in \{0,1\}$、F 為對數勝算的二元對數損失，偽殘差是 $y_i - \sigma(F_{m-1}(x_i))$：標籤減去**預測機率**，而不是減去預測類別。若使用指數損失並搭配線搜尋，這個程序會化約為一種 AdaBoost。

學習率、回合數與樹的大小會互相影響：較小的 ν 需要更多回合，但通常泛化得更好；回合數以驗證資料上的提早停止來決定，絕不用最終測試集。XGBoost [9] 以正則化目標函數、二階（牛頓法）葉節點值、能處理稀疏資料的分割搜尋與加權分位數草圖（weighted quantile sketch），讓梯度提升樹變得快速且可擴展；scikit-learn 的 `HistGradientBoostingClassifier` 等以直方圖為基礎的實作也遵循相同原則 [26]。

平均不一定是同一種運算

許多方法都在「組合模型」，但它們回答的是不同問題。

方法	組合由什麼決定？	主要注意事項
投票或機率平均（bagging、森林）	對獨立訓練的預測器做固定、等權重的組合	錯誤彼此相關、校準不一致
Boosting	依序擬合；權重來自損失函數	標籤雜訊；需要提早停止
貝氏模型平均	模型假設的後驗機率	證據與先驗必須有意義
Stacking	學習一個以基底模型輸出為輸入的預測器	必須用 out-of-fold 預測來訓練
條件式混合（混合專家）	隨輸入改變的閘門或責任值	專家與閘門可能一起過擬合

貝氏模型平均（Bayesian model averaging，Bishop 第 14.1 節 [2]）使用

$$p(y \mid x, D) = \sum_m p(y \mid x, D, M_m)\, p(M_m \mid D),$$

其中 $p(M_m \mid D) \propto p(D \mid M_m) p(M_m)$ 是模型 M_m 的後驗機率。這些權重是後驗模型機率，不是任意的驗證準確率；資料充足時，它們往往集中在單一模型上。BMA 表達的是「資料是**哪一個**模型產生的」這種不確定性；委員會或集成則是用幾個部件組成一個更豐富的模型 [2]。模型證據 $p(D \mid M_m)$ 與 PR 08 的近似推論工具相連。

Stacking（堆疊泛化，stacked generalization，Wolpert [10]）則是**學習**一個從基底模型預測到目標的映射。它的訓練特徵必須來自沒有用這些樣本訓練過的基底模型；否則 stacker 學到的是不切實際、過擬合的簡單預測。標準做法是：把訓練集分成 K 折，每個基底模型在 $K-1$ 折上擬合並預測保留的那一折，收集這些 out-of-fold 預測當作特徵，在其上擬合一個簡單的後設學習器（常用正則化邏輯斯迴歸），最後再用全部訓練資料重新擬合各基底模型。

下面在 scikit-learn 內建的乳癌資料集（569 筆、30 個特徵）上比較各方法，所有方法都在同一個分層 5 折交叉驗證中執行，因此數字可以互相比較。`StackingClassifier` 會在每個外層訓練折內部，自行建立內層的 out-of-fold 預測（`cv=5`）。

```
import numpy as np
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import StratifiedKFold, cross_val_score
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import (BaggingClassifier, RandomForestClassifier,
                              HistGradientBoostingClassifier, StackingClassifier)

X, y = load_breast_cancer(return_X_y=True)
cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=0)
logreg = make_pipeline(StandardScaler(), LogisticRegression(max_iter=2000))
models = {
    "single tree": DecisionTreeClassifier(random_state=0),
    "bagged trees": BaggingClassifier(DecisionTreeClassifier(), n_estimators=100,
                                      random_state=0),
    "random forest": RandomForestClassifier(n_estimators=100, random_state=0),
    "hist. gradient boosting": HistGradientBoostingClassifier(random_state=0),
    "logistic regression": logreg,
}
models["stacking"] = StackingClassifier(
```

```
    estimators=[("rf", models["random forest"]),
                ("gb", models["hist. gradient boosting"]),
                ("lr", logreg)],
    final_estimator=LogisticRegression(max_iter=2000),
    cv=5, stack_method="predict_proba")
for name, model in models.items():
    s = cross_val_score(model, X, y, cv=cv, scoring="accuracy")
    print(f"{name:>24}: {s.mean():.3f} +/- {s.std():.3f}")
```

我們的執行得到平均準確率：單一決策樹 0.926、bagging 0.960、隨機森林 0.965、梯度提升 0.970、邏輯斯迴歸 0.979、stacking 0.979，各折標準差介於 0.009 到 0.022。由此得到兩個教訓。集成明顯修好了不穩定的單一決策樹。但在這個小而近乎線性可分的資料集上，一個正則化良好的**線性**模型已經和 stack 一樣好，而前四名之間的差距都在一個折標準差之內。它們之中是否有誰**顯著**較好，要交給本章稍後的統計檢定，而不是用眼睛看平均值。

驗證評估的是整個程序

白話版。只有當測試資料完全沒有參與模型的建立，測試分數才值得信任。我們看著資料時做的每一個選擇，包括特徵、超參數、閾值，甚至用哪個演算法，都是「模型」的一部分，都必須在不使用評分資料的情況下決定。

訓練擬合參數。驗證選擇超參數、特徵、閾值與校準方式。測試評估被選中的程序。每一個適應性選擇，都會消耗它所檢視資料中的資訊。嚴格來說，交叉驗證估計的是一個**程序**（演算法加上調參規則）在給定大小的資料集上訓練後的表現，而不是你最後要部署的那一個擬合模型的誤差；Bates、Hastie 與 Tibshirani 把這一點說得很精確，並指出天真的交叉驗證信賴區間可能涵蓋率不足 [19]。

保留法、k 折與留一法

- **保留法（hold-out）**。只切一次訓練與測試。簡單又便宜，但資料少時估計的變異數高、依賴切分方式，而且模型使用的訓練資料比手上擁有的少。
- **k 折交叉驗證**。把資料分成 k 折；在 $k-1$ 折上訓練、在剩下一折上評分，輪流讓每一折都被保留一次（圖 4 上）。CV 估計為 $\widehat{\mathrm{Err}}_{\mathrm{CV}} = \frac{1}{n}\sum_{i} L(y_i, \hat{f}^{(-\kappa(i))}(x_i))$，其中 $\kappa(i)$ 是樣本 i 所在的折，$\hat{f}^{(-k)}$ 是不含第 k 折訓練出的模型。$k=5$ 或 10 是常見的折衷：每個模型看到 80–90% 的資料，悲觀偏差小，成本是 k 次擬合。
- **留一法（leave-one-out, LOO）**。即 $k=n$ 的情形。對大小為 n 的程序幾乎不偏，但需要 n 次擬合（除非有封閉解，例如線性平滑器或 k-NN），而且因為 n 份訓練集幾乎相同，變異數可能很高。
- **分層（stratification）**讓各折的類別比例相近，對稀有類別很重要。它無法解決樣本之間的相依性。

Arlot 與 Celisse 回顧了這些選擇在估計風險與選擇模型兩方面的理論 [18]。

群組與時間

當觀測共享同一位病患、書寫者、相機或錄製時段時，要把**整個群組**放在一起（圖 4 中）。否則同一個人的近似重複影像會落在切分的兩側，分數衡量的是對那個人的記憶，而不是對新的人的辨識。依時間排序的資料需要依時間切分（用過去訓練、用未來測試）。原則是：切分方式要對應部署時的問題。「已知書寫者的新影像」與「新書寫者的影像」是不同問題，應該用不同的切分。

Bootstrap、.632 與.632+

白話版。不用折，改抽許多份 bootstrap 樣本，在每份上訓練，再用該份漏掉的點測試。

精確版。以放回抽樣抽出 B 份大小為 n 的 bootstrap 樣本。對每一份擬合模型，並在沒被抽到的點（約 36.8% 的資料）上評估。其平均即留一 bootstrap 誤差 $\widehat{\mathrm{Err}}^{(1)}$。由於每個 bootstrap 模型只用到約 63.2% 的**相異**點訓練，$\widehat{\mathrm{Err}}^{(1)}$ 偏悲觀；而再代入（訓練）誤差 $\overline{\mathrm{err}}$ 偏樂觀。Efron 的 **.632 估計量**把兩者混合：

$$\widehat{\mathrm{Err}}^{(.632)} = 0.368\,\overline{\mathrm{err}} + 0.632\,\widehat{\mathrm{Err}}^{(1)}.$$

它對 1-最近鄰這類嚴重過擬合、訓練誤差為零的學習器會失效。Efron 與 Tibshirani 的 **.632+ 估計量** [11] 在過擬合越嚴重時，把權重越往 $\widehat{\mathrm{Err}}^{(1)}$ 移動，藉此修正。令 $\hat{\gamma}$ 為**無資訊錯誤率**（no-information error rate，假如輸入與標籤彼此獨立時的錯誤率：$\hat{\gamma} = \sum_{k} \hat{p}_k (1 - \hat{q}_k)$，其中 $\hat{p}_k$ 是類別 k 的觀測比例，$\hat{q}_k$ 是預測為 k 的比例），並定義相對過擬合率 $\hat{R} = (\widehat{\mathrm{Err}}^{(1)} - \overline{\mathrm{err}})/(\hat{\gamma} - \overline{\mathrm{err}})$，截斷到 $[0, 1]$。則

$$\hat{w} = \frac{0.632}{1 - 0.368\hat{R}}, \qquad \widehat{\mathrm{Err}}^{(.632+)} = (1 - \hat{w})\,\overline{\mathrm{err}} + \hat{w}\,\widehat{\mathrm{Err}}^{(1)}.$$

$\hat{R} = 0$（沒有過擬合）時它化約為.632；$\hat{R} = 1$ 時它變成 $\widehat{\mathrm{Err}}^{(1)}$。

```
import numpy as np
from sklearn.datasets import make_classification
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import cross_val_score, KFold

X, y = make_classification(n_samples=200, n_features=6, n_informative=3,
                           flip_y=0.1, random_state=3)
model = KNeighborsClassifier(n_neighbors=1)          # resubstitution error is 0
rng = np.random.default_rng(3)
n, B = len(y), 200
resub = np.mean(model.fit(X, y).predict(X) != y)
oob_errs = []
for b in range(B):
    idx = rng.integers(0, n, n)                       # draw n with replacement
    out = np.setdiff1d(np.arange(n), idx)             # ~36.8% left out
    model.fit(X[idx], y[idx])
    oob_errs.append(np.mean(model.predict(X[out]) != y[out]))
err_boot1 = np.mean(oob_errs)                          # leave-one-out bootstrap
err_632 = 0.368 * resub + 0.632 * err_boot1
```

```
cv10 = 1 - cross_val_score(model, X, y,
                           cv=KFold(10, shuffle=True, random_state=3)).mean()
print(f"resubstitution {resub:.3f}  OOB bootstrap {err_boot1:.3f}  "
      f".632 {err_632:.3f}  10-fold CV {cv10:.3f}")

# .632+ : shrink toward the out-of-bootstrap error when overfitting is large.
pred = model.fit(X, y).predict(X)
gamma = sum(np.mean(y == k) * (1 - np.mean(pred == k)) for k in np.unique(y))
R = np.clip((err_boot1 - resub) / (gamma - resub), 0, 1)
w = 0.632 / (1 - 0.368 * R)
print(f"no-information rate {gamma:.3f}  R {R:.3f}  "
      f".632+ {(1 - w) * resub + w * err_boot1:.3f}")
```

對 1-NN，再代入誤差為 0.000，bootstrap 外誤差 0.332，.632 估計只有 0.210（被沒有意義的零拉低），.632+ 估計為 0.278，10 折 CV 為 0.325。這裡.632 估計量嚴重過度樂觀；.632+ 修補了大部分差距。實務上，10 折 CV（可重複多次）與.632+ 都合理；bootstrap 在樣本極少時以及建立信賴區間時特別有吸引力。

巢狀交叉驗證

白話版。如果你用交叉驗證調超參數，然後回報同一個交叉驗證分數，你回報的是**最幸運**那組設定的分數，會過度樂觀。巢狀 CV 把整個調參過程放進外層迴圈裡。

巢狀交叉驗證有負責選擇的**內層迴圈**，以及評估這個選擇程序的**外層迴圈**（圖 4 下）。在每個外層訓練集內，以內層 CV 選出超參數；選中的設定在整個外層訓練集上重新擬合，再在內層從未見過的外層保留折上評分。Varma 與 Simon 證明，回報內層（調參用）CV 分數可能有相當大的偏差，而巢狀 CV 給出幾乎不偏的估計 [12]。Cawley 與 Talbot 更進一步指出：對模型選擇準則本身過擬合，造成的損失可能與學習演算法之間的差距一樣大，因此低變異數的選擇準則與不偏的準則同樣重要 [13]。前處理（標準化、特徵選擇、PCA）要在每個內層折中重新擬合，通常把它放進 `Pipeline` 即可。回報外層各折之間的變異，但不要把彼此相關的折分數當成獨立測量，去算天真的信賴區間。

估計完程序之後，在開發資料上做選擇，最後在從未動過的測試集上評估一次。反覆查看測試誤差並重新設計模型，會把測試集變成另一個驗證集。

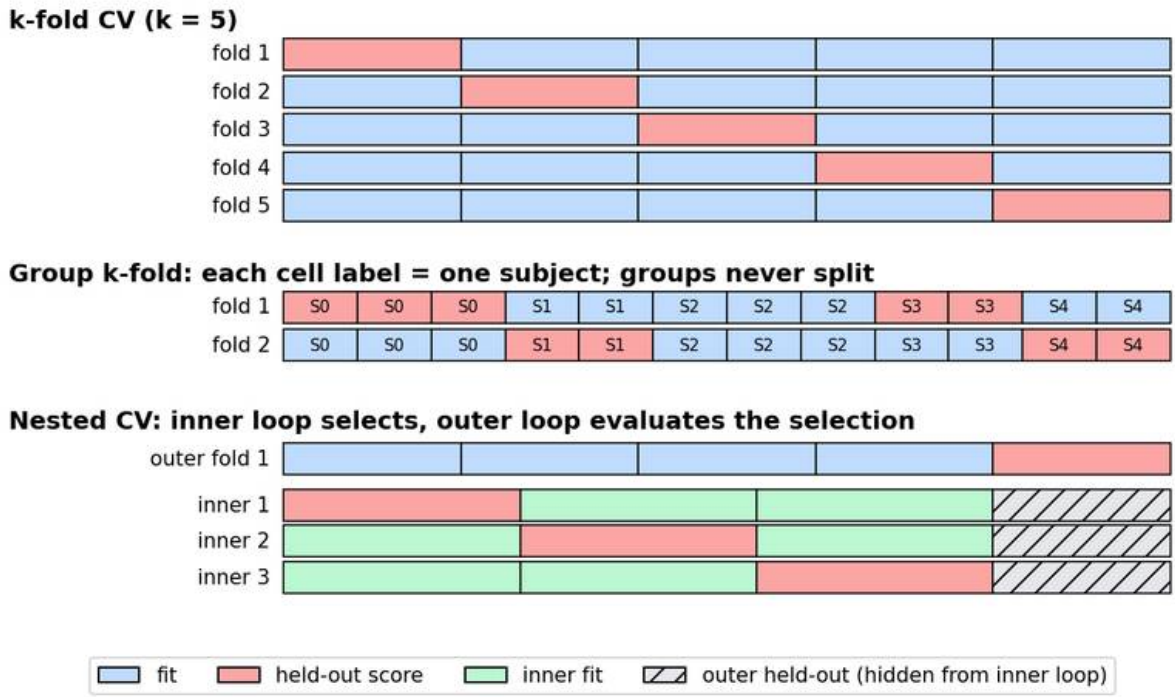


圖 *4* — 三種驗證配置。上：*5* 折 *CV*。中：群組 *k* 折讓每位受試者（*S0–S4*）的所有觀測在每次切分中都只落在同一側。下：巢狀 *CV*；內層迴圈只用外層訓練資料調參，外層保留折（斜線）在調好的模型評分前都保持隱藏。

程式碼：沒有群組洩漏的巢狀選模

這個 CPU 範例為每位合成受試者產生三筆帶雜訊的重複測量。群組標籤描述的是受試者，不是類別。外層迴圈估計所選森林程序的表現；另一份群組不重疊的測試集在最終回報前都不會被動到。

```
import numpy as np
from sklearn.datasets import make_classification
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split, GroupKFold, GridSearchCV
from sklearn.metrics import balanced_accuracy_score, log_loss

rng = np.random.default_rng(9)
subject_x, subject_y = make_classification(
    n_samples=180, n_features=8, n_informative=5, n_redundant=1,
    class_sep=1.0, random_state=9)
groups = np.repeat(np.arange(180), 3)
X = subject_x[groups] + rng.normal(0, 0.03, (len(groups), 8))
y = subject_y[groups]
dev_groups, test_groups = train_test_split(
    np.arange(180), test_size=0.25, stratify=subject_y, random_state=9)
dev = np.flatnonzero(np.isin(groups, dev_groups))
```

```
test = np.flatnonzero(np.isin(groups, test_groups))
assert not set(groups[dev]) & set(groups[test])

def search():
    return GridSearchCV(
        RandomForestClassifier(n_estimators=60, random_state=9, n_jobs=1),
        {"max_depth": [3, None], "min_samples_leaf": [1, 3]},
        cv=GroupKFold(3), scoring="balanced_accuracy")

outer_scores = []
for fit, valid in GroupKFold(3).split(X[dev], y[dev], groups[dev]):
    chosen = search().fit(X[dev[fit]], y[dev[fit]], groups=groups[dev[fit]])
    pred = chosen.predict(X[dev[valid]])
    outer_scores.append(balanced_accuracy_score(y[dev[valid]], pred))
final = search().fit(X[dev], y[dev], groups=groups[dev])
p = final.predict_proba(X[test])
pred = final.predict(X[test])
positive = p[:, np.flatnonzero(final.classes_ == 1)[0]]
print("Nested outer balanced accuracy:", np.round(outer_scores, 3))
print("Selected parameters:", final.best_params_)
print("Independent group test:", balanced_accuracy_score(y[test], pred))
print("Test log loss:", log_loss(y[test], p, labels=final.classes_))
print("Test Brier:", np.mean((positive - y[test]) ** 2))
```

我們的執行印出外層平衡準確率 0.869、0.784 與 0.861，選中 `max_depth=None, min_samples_leaf=3`，並在獨立群組測試集上得到平衡準確率 0.853、對數損失 0.378 與 Brier 分數 0.118。外層分數的分散（0.78 到 0.87）提醒我們，單一切分會有多嘈雜。這些結果描述的是合成受試者，不代表在影像上的表現。若把 `GroupKFold` 換成逐列打亂的折，同一受試者的近似複本就會出現在兩側，分數因而虛高（練習 3）。

以統計方法比較分類器

白話版。如果分類器 A 得 97.9%、B 得 96.5%，A 真的比較好，還是只是剛好在這份資料上運氣好？統計檢定問的是：如果兩者其實一樣好，觀察到這樣的差異有多令人意外。麻煩的是，教科書上常見的 t 檢定假設各次測量彼此獨立，而交叉驗證分數並不獨立。

Dietterich 對近似檢定的研究 [14] 是經典的起點；Raschka 的教學文章整理了主要選項並附有程式碼 [20]。

單一測試集：McNemar 檢定

兩個分類器各訓練一次，並在**同一個**測試集上評估。計算 n_{01}：A 對而 B 錯的測試點數；n_{10}：反過來的點數。兩者都對或都錯的點，對「誰比較好」沒有提供資訊。在錯誤率相等的虛無假設下，每一個不一致的點偏向哪一方的機會相等，因此 $n_{01} \sim \text{Binomial}(n_{01}+n_{10}, \tfrac{1}{2})$。連續性修正後的統計量

$$\chi^2 = \frac{(|n_{01} - n_{10}| - 1)^2}{n_{01} + n_{10}}$$

近似服從自由度 1 的 χ^2 分布；計數小時改用精確二項檢定。Dietterich 發現 McNemar 檢定的型一錯誤率可接受，並建議在演算法只能執行一次時使用它 [14]。它沒有納入訓練集選擇所造成的變異。

重複切分：為什麼天真的 t 檢定會失敗

一個誘人的替代做法是重複 J 次隨機訓練／測試切分，記錄誤差差值 d_j，再做成對 t 檢定。Dietterich 證明這種**重抽樣 t 檢定**的型一錯誤率嚴重膨脹 [14]：各訓練集高度重疊，所以 d_j 之間呈正相關，一般的標準誤 $\hat{\sigma}/\sqrt{J}$ 遠遠太小。常見的修正有兩種。

- **5×2cv 成對 t 檢定** [14]。做五次 2 折 CV。在第 i 次中，令 $p_i^{(1)}, p_i^{(2)}$ 為兩折上的誤差差值，$\bar{p}_i$ 為其平均，$s_i^2 = (p_i^{(1)} - \bar{p}_i)^2 + (p_i^{(2)} - \bar{p}_i)^2$。統計量 $t = p_1^{(1)} / \sqrt{\frac{1}{5}\sum_{i=1}^{5} s_i^2}$ 與自由度 5 的 t 分布比較。
- **修正重抽樣 t 檢定**（Nadeau 與 Bengio [15]）。保留 J 次隨機切分，每次有 n_{test} 個測試點與 n_{train} 個訓練點，但把變異數放大以反映重疊：

$$t = \frac{\bar{d}}{\sqrt{\left(\frac{1}{J} + \frac{n_{\text{test}}}{n_{\text{train}}}\right)\hat{\sigma}_d^2}},$$

其中 $\bar{d}$ 與 $\hat{\sigma}_d^2$ 是 d_j 的平均與樣本變異數，並與自由度 $J-1$ 的 t 分布比較。多出來的 $n_{\text{test}}/n_{\text{train}}$ 項不會隨 J 增大而縮小，這正是重點：對同一份資料重抽再多次，也無法創造更多資訊。

```
import numpy as np
from scipy import stats
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split, ShuffleSplit
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier

X, y = load_breast_cancer(return_X_y=True)
A = make_pipeline(StandardScaler(), LogisticRegression(max_iter=2000))
B = RandomForestClassifier(n_estimators=200, random_state=0)

# 1) McNemar's test on one held-out test set (paired predictions).
Xtr, Xte, ytr, yte = train_test_split(X, y, test_size=0.4, stratify=y,
                                      random_state=0)
ca = A.fit(Xtr, ytr).predict(Xte) == yte
cb = B.fit(Xtr, ytr).predict(Xte) == yte
n01 = int(np.sum(ca & ~cb))      # A right, B wrong
n10 = int(np.sum(~ca & cb))      # A wrong, B right
p_exact = stats.binomtest(n01, n01 + n10, 0.5).pvalue if n01 + n10 else 1.0
chi2 = (abs(n01 - n10) - 1) ** 2 / (n01 + n10) if n01 + n10 else 0.0
print(f"McNemar: n01={n01} n10={n10} chi2_cc={chi2:.3f} "
      f"p_chi2={stats.chi2.sf(chi2, 1):.3f} p_exact={p_exact:.3f}")

# 2) Corrected resampled t-test (Nadeau & Bengio) over J random splits.
```

```
J, test_frac = 30, 0.2
d = []
for tr, te in ShuffleSplit(J, test_size=test_frac, random_state=0).split(X):
    ea = np.mean(A.fit(X[tr], y[tr]).predict(X[te]) != y[te])
    eb = np.mean(B.fit(X[tr], y[tr]).predict(X[te]) != y[te])
    d.append(ea - eb)
d = np.array(d)
naive_se = np.sqrt(d.var(ddof=1) / J)
corr_se = np.sqrt(d.var(ddof=1) * (1 / J + test_frac / (1 - test_frac)))
for name, se in [("naive", naive_se), ("corrected", corr_se)]:
    t = d.mean() / se
    print(f"{name:>9} t={t:.3f}  p={2 * stats.t.sf(abs(t), J - 1):.3f}")
```

我們的執行得到 $n_{01} = 9$、$n_{10} = 5$，McNemar $p \approx 0.42$：在一個含 228 筆案例的測試集上，看不出差異的證據。在 30 次隨機 80/20 切分中，邏輯斯迴歸的誤差平均約低 1.8 個百分點。天真的 t 檢定給出 $t \approx -5.4$、$p < 0.001$；修正後的檢定給出 $t \approx -1.85$、$p \approx 0.075$（圖 5）。同一份資料，在 5% 水準下卻得到相反的結論：天真的檢定把 30 次高度重疊的重抽樣誤當成 30 次獨立實驗。

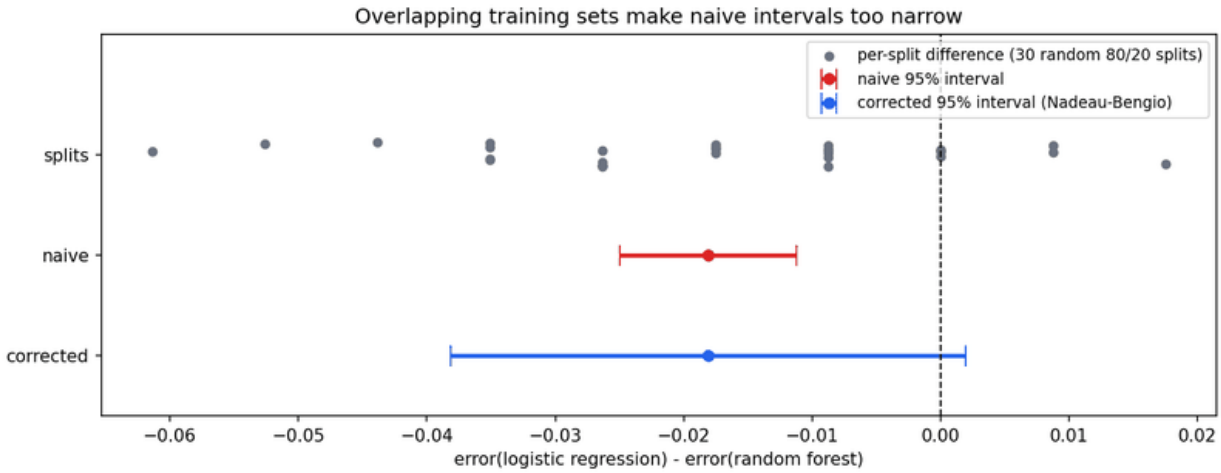


圖 5 — 乳癌資料上 *30* 次隨機 *80/20* 切分的誤差差值（邏輯斯迴歸減隨機森林）。天真的成對 t 區間不含零；考慮訓練集重疊的 *Nadeau–Bengio* 修正區間則含零。

多個分類器、多個資料集

在 N 個資料集上比較 k 個演算法時，Demšar 建議使用基於排名的無母數檢定 [16]：兩個分類器用 Wilcoxon 符號等級檢定；更多個則用 **Friedman 檢定**。在每個資料集上為演算法排名（1 為最佳），令 R_j 為演算法 j 的平均排名。在「全部等價」的虛無假設下，

$$\chi_F^2 = \frac{12N}{k(k+1)}\left[\sum_{j=1}^{k} R_j^2 - \frac{k(k+1)^2}{4}\right]$$

近似服從自由度 $k-1$ 的 χ^2 分布（Demšar 也給出一個較不保守、服從 F 分布的變體）。若虛無假設被拒絕，**Nemenyi 事後檢定**在兩個演算法的平均排名差距超過臨界差 $\mathrm{CD} = q_\alpha \sqrt{k(k+1)/(6N)}$

時判定兩者不同，其中 q_α 來自 Studentized range 統計量除以 $\sqrt{2}$；Demšar 的**臨界差圖**（critical difference diagram）能精簡地呈現結果 [16]。這裡彼此獨立的單位是資料集，而不是折。

Benavoli、Corani、Demšar 與 Zaffalon 主張更進一步，以**貝氏分析**取代虛無假設顯著性檢定，直接回報一個分類器在實務上較好、在實務等價區間（region of practical equivalence）內等價、或較差的機率 [17]。無論用哪種檢定，都要記得 p 值完全不說明差異**有多大**；也要回報效果量與區間。

同時量測決策與機率

準確率衡量答對的比例；**平衡準確率**（balanced accuracy）平均各類別的召回率，因此不會被多數類別主導。**Macro-F1** 在平均各類 F1 時給每個類別相同權重。混淆矩陣顯示發生了哪些錯誤。對稀有的正類，要在實際使用的閾值上檢查精確率與召回率；在閾值尚未決定時，檢查完整的精確率–召回率曲線或 ROC 曲線。

對數損失與 Brier 分數評估機率預測：

$$\text{LogLoss} = -\frac{1}{N}\sum_{n} \log p_n(y_n), \qquad \text{Brier} = \frac{1}{N}\sum_{n} (p_n - y_n)^2,$$

其中 $p_n(y_n)$ 是模型給第 n 筆樣本真實類別的機率；在 Brier 分數（二元情形）中，p_n 是預測為正類的機率，$y_n \in \{0, 1\}$。兩者都是**適當評分規則**（proper scoring rule）：回報真實機率時，其期望值最佳。**可靠度圖**（reliability diagram）在各個區間中比較預測機率與觀察到的頻率。好的排序、好的準確率與好的校準是不同的性質：隨機森林平均投票，傾向把機率推向中間；回合數很多的 boosting 可能把機率推向兩端；模型也可能 ROC AUC 很好，分數卻校準得很差。

擬合校準器時（scikit-learn 提供 sigmoid 與 isotonic 兩種映射 [26]），要使用沒有參與對應基底模型擬合的資料所產生的預測。若觀測有群組，校準用的折也必須遵守群組。校準不會自動修復領域偏移（domain shift）。

把分群有效性當作模型選擇

分群沒有標籤，所以「驗證」的意義不同。Theodoridis 與 Koutroumbas 以整個第 16 章討論分群有效性，因為整齊的分割不一定能服務任務 [1]。完整的指標（silhouette、Davies–Bouldin、Calinski–Harabasz、gap statistic、混合模型的 BIC）在 PR 07 中隨演算法一起介紹；這裡把它們放進本章的驗證框架。

內部準則只使用特徵與指派結果。點 i 的 silhouette 比較它到自己群的平均距離 $a(i)$，以及到其他群的最小平均距離 $b(i)$：

$$s(i) = \frac{b(i) - a(i)}{\max(a(i), b(i))} \in [-1, 1].$$

它偏好在所選幾何中分離良好的群。以最大化內部指標來選群數，是一個模型選擇步驟，和調超參數完全一樣；分數高並不代表各群對應到想要的語意類別。

外部準則把指派結果與獨立的參考標籤比較。調整蘭德指數（adjusted Rand index, ARI）對成對一致性做機率校正，而且對標籤排列不變，所以哪一群叫「1」並不重要。如果用標籤來選演算法或群數，那些標籤就不再是未被碰過的外部評估；選擇與評估分離的原則在這裡同樣適用。

穩定性比較重抽樣或擾動後的分割，很像 bootstrap：對多個子樣本分群，並在它們共有的觀測上衡量一致性（例如 ARI），同時處理標籤排列。穩定的群仍可能只反映與任務無關的相機或擷取假影。

現代觀點

集成學習綜述。 Dong、Yu、Cao、Shi 與 Ma 對集成學習做了廣泛的綜述 [21]：他們回顧主流的集成方法、依其特性分類、列出各方法的挑戰與研究方向，並討論集成與深度學習、強化學習的結合。Ganaie、Hu、Malik、Tanveer 與 Suganthan 回顧了**集成深度學習** [22]，把方法整理成以 bagging、boosting、stacking 為基礎的深度集成、負相關學習（negative correlation learning）、顯式與隱式集成（例如共享權重的 snapshot 或 dropout 式集成）、同質與異質成員，以及決策融合策略。在所有這些類別中，實務問題都和前面變異數公式提出的一樣：如何便宜地取得**多樣**的成員。

表格資料上的樹集成。儘管深度網路興起，梯度提升樹（XGBoost [9] 及其以直方圖為基礎的後繼者）與隨機森林 [5] 仍是中等規模表格問題的預設選擇；在辨識中，這類問題包括手工描述子、詮釋資料，以及其他模型的輸出。Grinsztajn、Oyallon 與 Varoquaux 在 45 個資料集上評比樹模型與深度架構，發現在約一萬筆樣本的規模下，樹模型仍是最先進的方法，並把差距歸因於神經網路對無資訊特徵敏感、具有旋轉不變性，以及難以學習不規則函數 [23]。

用集成估計不確定性。 Lakshminarayanan、Pritzel 與 Blundell 證明，**深度集成**（deep ensembles，以不同隨機初始化並搭配適當評分規則訓練的多個網路）在他們的實驗中，給出的預測不確定性與近似貝氏方法相當或更好，而且在分布外輸入上退化得較平緩 [24]。這把 bagging 的直覺（意見分歧代表變異數）連到 PR 08 的貝氏模型平均，也是新的不確定性方法都要比較的基準。

重新檢視偏差–變異數。 Belkin、Hsu、Ma 與 Mandal 記錄了**雙重下降** [25]：當容量超過模型能內插訓練資料的那一點後，測試誤差可能再次下降。這並沒有推翻分解，但提醒我們「參數越多、變異數越大」不是定律，而經典的 U 形曲線只描述了容量範圍的一部分。

交叉驗證到底在估計什麼。 Arlot 與 Celisse 的綜述 [18] 仍是 CV 在風險估計與模型選擇方面理論的參考文獻。Bates、Hastie 與 Tibshirani 證明，CV 估計的是在同樣大小的其他訓練集上擬合的模型的平均誤差，而不是手上這個特定模型的誤差，且標準的 CV 信賴區間涵蓋率可能很差；他們提出以巢狀 CV 估計變異數，得到涵蓋率較好的區間 [19]。隨著自動化超參數搜尋在同一組折上比較成千上萬種設定，Cawley 與 Talbot 關於「對選擇準則過擬合」的警告 [13] 變得更加重要。

比較演算法。實務已從天真的重抽樣 t 檢定，轉向單一資料集上的 McNemar 與修正檢定 [14], [15]、跨資料集時以排名為基礎並搭配臨界差圖的 Friedman/Nemenyi 檢定 [16]，以及具實務等價區間的貝氏比較 [17]。Raschka 的教學文章 [20] 是涵蓋以上所有方法的實用指南。

開放問題。在分布偏移（新相機、新場域或新族群）下的誠實評估，因為沒有任何隨機切分能對應部署情境；大型預訓練模型的預訓練資料可能包含測試影像所造成的洩漏；在推論時便宜的集成如何提供可靠的不確定性；以及只有單一大型基準時的統計比較。ML 11 會在深度視覺模型上延續這些主題。

重點整理

- 對平方誤差而言，期望測試誤差 = 雜訊 + 偏差平方 + 變異數；深樹這類有彈性的學習器以偏差換取變異數。
- 平均 M 個成對相關為 ρ 的預測器，剩下的變異數是 $\sigma^2[\rho + (1-\rho)/M]$：bagging 與隨機森林對不穩定的學習器有幫助，而降低成員間相關性（隨機特徵子集）是關鍵槓桿。
- AdaBoost 是對指數損失的貪婪前向逐步最小化；梯度提升把樹擬合到任意可微分損失的負梯度，收縮與提早停止是主要的正則化手段。
- 投票、boosting、貝氏模型平均與 stacking 以不同方式組合模型；stacking 必須用 out-of-fold 預測訓練。
- 驗證估計的是**程序**：每個適應性選擇都屬於迴圈內部，切分必須遵守群組與時間，而要回報調過參的模型表現需要巢狀 CV。
- Bootstrap 的.632+ 估計量修正了.632 對過擬合學習器的過度樂觀；out-of-bag 分數忽略了群組結構。
- 不要對重疊的 CV 折做天真的 t 檢定；改用 McNemar、5×2cv、修正重抽樣 t 檢定，或跨資料集的 Friedman/Nemenyi，並回報效果量。
- 內部分群指標、外部一致性（ARI）與穩定性回答不同的問題；用它們選 K 就是模型選擇。

練習

1. **相關性下限**。展開總和的變異數，推導平均變異數公式。當 $\rho = 0.8$ 時，需要多少個預測器才能讓變異數落在極限值的 1% 以內？這對 bagging 非常相似的模型有什麼啟示？

   提示

   總和有 M 個變異數項，以及 $M(M-1)$ 個有序共變異數項 $\rho\sigma^2$。極限是 $\rho\sigma^2 = 0.8\sigma^2$；超出的部分是 $(1-\rho)\sigma^2/M = 0.2\sigma^2/M$。要求 $0.2/M \le 0.008$ 得到 $M \ge 25$。超過幾十個高度相關的模型後幾乎沒有收穫；要再降低，唯一的方法是降低 ρ。

2. **手算 AdaBoost**。五個點的權重都是 0.2。第一個樹樁恰好分錯其中一點。計算 ϵ_1、α_1 與新的正規化權重。在新權重下，同一個樹樁的加權誤差是多少？為什麼這不是巧合？

   提示

   $\epsilon_1 = 0.2$，$\alpha_1 = \frac{1}{2}\log 4 = \log 2 \approx 0.693$。未正規化權重：分錯的點為 $0.2 \cdot 2 = 0.4$，其餘為 $0.2/2 = 0.1$；總和 0.8，所以新權重是 0.5 與 0.125（四個）。該樹樁的新加權誤差恰好是 0.5：採用最佳 α 時，AdaBoost 總是重新加權到讓上一個學習器不比亂猜好，迫使下一個學習器去找新的東西。

3. **會洩漏的折**。在巢狀 CV 程式中，把每個 `GroupKFold` 換成 `KFold(3, shuffle=True, random_state=0)`（並移除 `groups` 參數）。比較外層分數與群組不重疊的測試分數，並說明每種切分回答的是哪一個部署問題。

   提示

使用逐列的折時，同一受試者三筆幾乎相同的測量中，常有兩筆在訓練集、第三筆被拿來驗證，所以森林可以認出受試者而不是類別；內外層分數應該會高於獨立群組測試。逐列的折回答「已知受試者的新測量」；群組折回答「新的受試者」。

4. **穩定學習器的 bootstrap 與 CV**。用 `KNeighborsClassifier(n_neighbors=15)` 重跑 bootstrap 程式。再代入誤差、bootstrap 外誤差、.632、.632+ 與 10 折 CV 現在如何比較？為什麼.632 與.632+ 的差距縮小了？

   提示

   15-NN 分類器的訓練誤差不再為零，所以相對過擬合率 $\hat{R}$ 小得多，$\hat{w}$ 接近 0.632，兩個估計量幾乎一致。所有估計值應該都落在彼此幾個百分點之內。

5. **選擇檢定**。你有 (a) 一個固定的基準切分，以及兩個偵測器的預測；(b) 一個小資料集，以及足以把每個模型重新訓練十次的計算資源；(c) 五個演算法在十二個公開資料集上的評估結果。每種情況適合哪個檢定？各自彼此獨立的單位是什麼？

   提示

   (a) 對成對不一致使用 McNemar 檢定（或精確二項檢定）；單位是測試影像，訓練集造成的變異沒有被納入。(b) 5×2cv t 檢定或修正重抽樣 t 檢定；單位是重抽樣，並針對重疊做修正。(c) Friedman 檢定加 Nemenyi 事後檢定（或貝氏階層比較）；單位是資料集。

6. **意義錯誤的群**。構造（在紙上或用 `make_blobs`）一個物體影像資料集，它依相機強烈分群，而類別標籤描述的是物體身分。哪些有效性指標會揭露這種不一致？哪些不會？

   提示

   silhouette 等內部指標會給相機群很高的分數，因為它們分離得很好。對物體標籤的外部 ARI 會很低，而對相機詮釋資料的 ARI 會很高。穩定性也會很高，所以單靠穩定性無法揭露這個分群在語意上是錯的。

參考文獻

1. S. Theodoridis and K. Koutroumbas, *Pattern Recognition*, 4th ed., Academic Press, 2009. 第 4 章（決策樹、分類器組合、boosting）、第 10 章（「Supervised Learning: The Epilogue」）與第 16 章（「Cluster Validity」）。doi
2. C. M. Bishop, *Pattern Recognition and Machine Learning*, Springer, 2006. 第 1.3 與 3.2 節（模型選擇、偏差–變異數）及第 14.1–14.5 節（貝氏模型平均、委員會、boosting、樹、條件式混合）。官方書籍與 PDF
3. S. Geman, E. Bienenstock and R. Doursat, “Neural networks and the bias/variance dilemma,” *Neural Computation*, vol. 4, no. 1, pp. 1–58, 1992. doi
4. L. Breiman, “Bagging predictors,” *Machine Learning*, vol. 24, no. 2, pp. 123–140, 1996. doi

5. L. Breiman, “Random forests,” *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001. doi
6. Y. Freund and R. E. Schapire, “A decision-theoretic generalization of on-line learning and an application to boosting,” *Journal of Computer and System Sciences*, vol. 55, no. 1, pp. 119–139, 1997. doi
7. J. Friedman, T. Hastie and R. Tibshirani, “Additive logistic regression: A statistical view of boosting (with discussion),” *The Annals of Statistics*, vol. 28, no. 2, 2000. doi
8. J. H. Friedman, “Greedy function approximation: A gradient boosting machine,” *The Annals of Statistics*, vol. 29, no. 5, 2001. doi
9. T. Chen and C. Guestrin, “XGBoost: A scalable tree boosting system,” in *Proc. 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 785–794, 2016. doi
10. D. H. Wolpert, “Stacked generalization,” *Neural Networks*, vol. 5, no. 2, pp. 241–259, 1992. doi
11. B. Efron and R. Tibshirani, “Improvements on cross-validation: The .632+ bootstrap method,” *Journal of the American Statistical Association*, vol. 92, no. 438, pp. 548–560, 1997. doi
12. S. Varma and R. Simon, “Bias in error estimation when using cross-validation for model selection,” *BMC Bioinformatics*, vol. 7, 91, 2006. doi
13. G. C. Cawley and N. L. C. Talbot, “On over-fitting in model selection and subsequent selection bias in performance evaluation,” *Journal of Machine Learning Research*, vol. 11, pp. 2079–2107, 2010. JMLR
14. T. G. Dietterich, “Approximate statistical tests for comparing supervised classification learning algorithms,” *Neural Computation*, vol. 10, no. 7, pp. 1895–1923, 1998. doi
15. C. Nadeau and Y. Bengio, “Inference for the generalization error,” *Machine Learning*, vol. 52, no. 3, pp. 239–281, 2003. doi
16. J. Demšar, “Statistical comparisons of classifiers over multiple data sets,” *Journal of Machine Learning Research*, vol. 7, pp. 1–30, 2006. JMLR
17. A. Benavoli, G. Corani, J. Demšar and M. Zaffalon, “Time for a change: A tutorial for comparing multiple classifiers through Bayesian analysis,” *Journal of Machine Learning Research*, vol. 18, no. 77, pp. 1–36, 2017. JMLR
18. S. Arlot and A. Celisse, “A survey of cross-validation procedures for model selection,” *Statistics Surveys*, vol. 4, pp. 40–79, 2010. doi
19. S. Bates, T. Hastie and R. Tibshirani, “Cross-validation: What does it estimate and how well does it do it?,” *Journal of the American Statistical Association*, vol. 119, no. 546, pp. 1434–1445, 2024. doi
20. S. Raschka, “Model evaluation, model selection, and algorithm selection in machine learning,” arXiv:1811.12808, 2018. arXiv
21. X. Dong, Z. Yu, W. Cao, Y. Shi and Q. Ma, “A survey on ensemble learning,” *Frontiers of Computer Science*, vol. 14, no. 2, pp. 241–258, 2020. doi
22. M. A. Ganaie, M. Hu, A. K. Malik, M. Tanveer and P. N. Suganthan, “Ensemble deep learning: A review,” *Engineering Applications of Artificial Intelligence*, vol. 115, 105151, 2022. doi
23. L. Grinsztajn, E. Oyallon and G. Varoquaux, “Why do tree-based models still outperform

deep learning on typical tabular data?,” in *Advances in Neural Information Processing Systems* (*NeurIPS*) *Datasets and Benchmarks Track*, 2022 (arXiv:2207.08815). arXiv

24. B. Lakshminarayanan, A. Pritzel and C. Blundell, “Simple and scalable predictive uncertainty estimation using deep ensembles,” in *Advances in Neural Information Processing Systems* (*NIPS*), 2017 (arXiv:1612.01474). arXiv
25. M. Belkin, D. Hsu, S. Ma and S. Mandal, “Reconciling modern machine-learning practice and the classical bias–variance trade-off,” *Proceedings of the National Academy of Sciences*, vol. 116, no. 32, pp. 15849–15854, 2019. doi
26. scikit-learn developers, *scikit-learn User Guide*：集成方法、交叉驗證、機率校準與分群效能評估。
27. NumPy developers, “numpy.linalg.lstsq,” API documentation, accessed 2026-11-05. documentation

PR 10・辨識專題：拒絕、漂移與學習式特徵

以 scikit-learn 手寫數字資料建立端到端、可重現的辨識專題：定義任務、鎖住測試集、以巢狀驗證比較基準與分類器、做錯誤分析，再檢驗拒絕、未知類別與分布漂移。

網站版本: https://blog.shyandram.dev/zh-tw/learn/pattern-recognition/recognition-capstone/

你將學到

- 如何把模糊的目標（「辨識手寫數字」）寫成一份實驗協定：觀測單位、抽樣單位、損失、拒絕選項，以及每個資料切分各自能用來做什麼。
- 如何用同一條管線（pipeline）與巢狀交叉驗證，公平比較本系列介紹過的分類器（樣板、單純貝氏、k-NN、LDA、邏輯斯迴歸、核 SVM、隨機森林）。
- 如何用校正後的重抽樣 t 檢定（corrected resampled t-test）判斷兩個模型的差距是否真實，而不是只看各折平均。
- 如何在開發資料上做錯誤分析、解讀學習曲線，並且只打開鎖住的測試集一次。
- 如何區分封閉集合準確率、選擇性分類、未知類別拒絕與分布漂移下的穩健性，以及這些原則如何延伸到預訓練神經網路特徵。

先看全貌

前九章各自研究辨識器的一個部分：決策與風險（PR 01）、密度估計（PR 02）、線性與核分類器（PR 03、PR 04）、特徵（PR 05）、樣板（PR 06）、分群（PR 07）、近似推論（PR 08），以及集成與驗證（PR 09）。真實的專題需要同時用上它們，順序要對，而且每個決策能看到哪些資料都必須有紀律。

> **用最簡單的話說**
>
> 想像你在教朋友把一大罐鈕扣分到十個盒子裡。你們先講好規則：什麼算鈕扣、有哪些盒子、遇到奇怪的東西（例如一枚硬幣）該怎麼辦。接著你偷偷把一小把鈕扣放進口袋，練習時完全不拿出來。朋友用其餘的鈕扣練習、嘗試各種分類技巧，並仔細檢查每一次失誤。直到最後，你才把口袋裡的鈕扣拿出來，數一數有幾顆被放進正確的盒子。如果練習時偷看了口袋裡的鈕扣，最後的分數就會讓朋友看起來比實際更厲害。

為什麼重要：多數令人失望的辨識系統，並不是因為選錯了分類器，而是因為評估回答的問題和部署時的問題不同：測試集洩漏進模型選擇、切分時把同一個人的樣本分到兩邊、系統從沒被問過遇到沒見過的東西該怎麼辦，或者沒有人檢查影像平移一個像素時會發生什麼事。Theodoridis 與 Koutroumbas 把系統設計與評估視為圖形識別本身的一部分 [1]，Bishop 則把最後一步表述為在損失函數下、帶有明確拒絕選項的決策 [2]。本章圍繞這些想法，完成一個完整的專題。

專題總覽

白話版。我們依序走九個步驟。前七步只使用開發資料；第 8 步打開鎖住的測試集一次；第 9 步提出比「準確率多少」更難的問題。

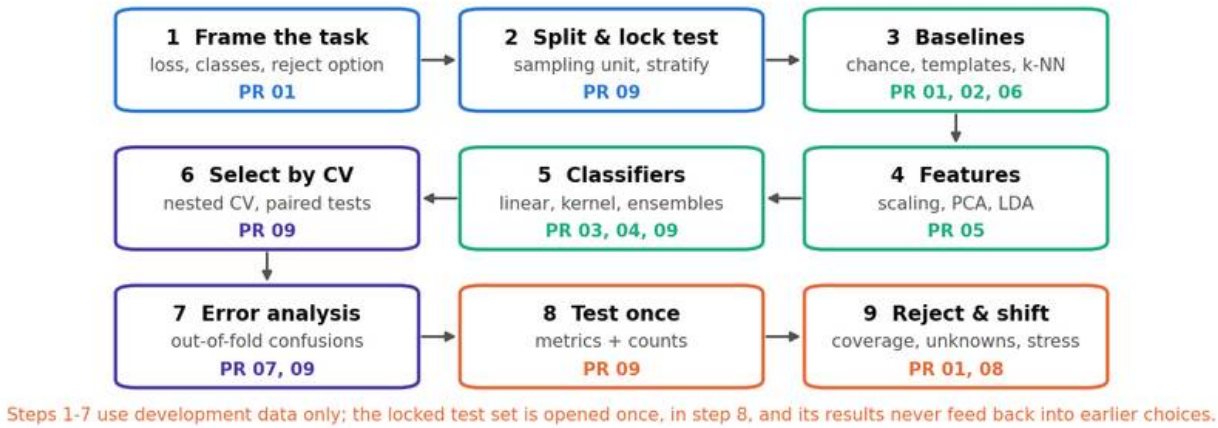


圖 *1* — 專題流程與各步驟所依據的章節。第 *1–7* 步只看開發資料；測試集只在第 *8* 步打開一次。

步驟	回答的問題	主要工具	對應章節
1 定義	什麼是一筆觀測、一個類別、一次錯誤？	損失矩陣、拒絕選項	PR 01
2 切分	哪些資料可以影響哪些決策？	分層、分組切分	PR 09
3 基準	在這裡「好」是什麼意思？	機率水準、類別平均、k-NN、單純貝氏	PR 01、02、06
4 特徵	分類器看到什麼表徵？	標準化、PCA、LDA	PR 05
5 分類器	哪些模型家族值得一試？	線性、核方法、集成	PR 03、04、09
6 選模	選哪個設定？差距是真的嗎？	巢狀交叉驗證、校正 t 檢定	PR 09
7 分析	為何失敗？更多資料有用嗎？	折外混淆、學習曲線	PR 07、09
8 測試	凍結後的流程表現如何？	含區間的準確率、各類別報告	PR 09
9 壓力測試	不確定、新類別與漂移怎麼辦？	涵蓋率–風險、未知類別、擾動	PR 01、08

以下所有程式都能在筆電 CPU 上於一分鐘內跑完，只使用 scikit-learn 內建資料 [4]。第 2 到第 8 步的程式片段構成同一支腳本：請在同一個 Python 工作階段中依序執行。

步驟 1 — 定義任務

白話版。動到模型之前，先寫下一筆樣本是什麼、可能的答案有哪些、犯錯的代價是多少，以及系統何時可以說「我不知道」。

一份書面協定

協定是在實驗之前寫下的簡短文件，能防止你在看到答案後才改問題。任何辨識專題至少要回答：

1. **觀測**。一筆輸入是什麼？一張影像、一個裁切區塊、一段影片，還是一整份文件？
2. **抽樣單位**。哪些觀測來自同一個來源（同一個人、同一台相機、同一位病人、同一天）？同源觀測彼此不獨立，在每一次切分中都必須留在同一側。
3. **標籤空間**。訓練時已知哪些類別？其他類別的輸入應如何處理？
4. **損失**。所有錯誤都一樣嚴重嗎？可以棄權嗎？棄權的代價是多少？
5. **資料角色**。訓練、驗證、測試資料各自可以拿來做什麼？
6. **部署條件**。評估必須涵蓋哪些裝置、書寫者、光照或時間區段？

我們想逼近的決策規則

PR 01 說明在 0–1 損失下，貝氏分類器會選擇後驗機率最大的類別。若加入代價為 λ_r 的拒絕選項（正確決策代價為 0，錯誤代價為 1），就得到 Bishop 在決策理論一節所介紹的拒絕規則 [2]：

$$\hat{y}(x) = \begin{cases} \arg\max_k P(\omega_k \mid x) & \text{if } \max_k P(\omega_k \mid x) \geq 1 - \lambda_r, \\ \text{reject} & \text{otherwise.} \end{cases}$$

其中 x 是特徵向量，ω_k 是第 k 類，$P(\omega_k \mid x)$ 是它的後驗機率，$0 \leq \lambda_r \leq 1$ 是拒絕相對於錯誤的代價。若 $\lambda_r = 0$，拒絕不必付出代價，只要不是百分之百確定就拒絕；若 K 個類別時 $\lambda_r \geq 1 - 1/K$，則永遠不拒絕。之後的一切（選分類器、驗證、設定門檻）都是在用有限資料逼近這條規則。請記得它的限制：這條規則只處理已知類別之間的不確定。一個全新的類別仍可能得到很高的後驗機率，所以第 9 步會另外處理未知類別。

本專題與它的資料卡

我們使用 `sklearn.datasets.load_digits` 辨識手寫數字 [4]。為資料寫一張簡短的「資料卡」也是定義任務的一部分（這正是資料集資料表 datasheets for datasets 的精神 [24]）：

- **來源**。這份資料是 Alpaydin 與 Kaynak 在 UCI 發布的「Optical Recognition of Handwritten Digits」資料集中測試部分的複本 [3]。
- **觀測**。共 1,797 張影像。每張原本是 32×32 的二值點陣圖，切成不重疊的 4×4 區塊；得到的 8×8 每一格計算區塊內「亮」的像素數，所以數值是 0 到 16 的整數 [3]。
- **類別**。十個數字，大致平衡（每類 174 到 183 張）。
- **來源者**。依資料集說明，這部分由 13 個人書寫，與書寫 UCI 訓練部分的 30 人不同 [3]。**資料沒有提供書寫者編號**。

最後一點限制了我們能做的所有宣稱。因為無法依書寫者分組，隨機切分會讓同樣這 13 個人的字同時出現在兩側。因此我們報告的測試準確率，估計的是這些書寫者的新影像上的表現，而不是新書寫者上的表現。UCI 的訓練部分由不同的人書寫，可以做到書寫者不重疊的測試，但它沒有內建在 scikit-learn 中，本章也不使用。坦白寫出這個限制，是誠實結果的一部分。

步驟 2 — 切分資料並鎖住測試集

白話版。一開始就把四分之一的資料放進「上鎖的抽屜」。在第 8 步之前，所有決策都不能用到它。

```
import numpy as np
import warnings
warnings.filterwarnings("ignore", message=".*within_class_std_dev_.*")
from sklearn.datasets import load_digits
from sklearn.model_selection import train_test_split

digits = load_digits()
X, y = digits.data, digits.target          # 1797 x 64, integer intensities 0..16
print("shape:", X.shape, "range:", X.min(), X.max())
print("class counts:", np.bincount(y))

idx = np.arange(len(y))
dev, test = train_test_split(idx, test_size=0.25, stratify=y, random_state=0)
assert not set(dev) & set(test)
print("development:", len(dev), "test (locked):", len(test))
```

輸出：`shape: (1797, 64) range: 0.0 16.0`，各類別數量介於 174 到 183，`development: 1347 test (locked): 450`。（warnings 過濾器只是關掉 `NearestCentroid` 的一則無害提示：某些邊緣像素在同一類別內恆為常數。）

為什麼要分層？測試集只有 450 張，隨機切分很容易讓某個數字偏少。分層（stratification）讓兩邊的類別比例一致。它不能解決觀測之間的相依性；只有依抽樣單位分組才能做到（PR 09）。

資料洩漏：最常見的自欺方式

Kapoor 與 Narayanan 整理了以機器學習為基礎的科學研究中的再現性失敗，發現資料洩漏（data leakage）遍布許多領域；他們提出的分類包括：訓練與測試沒有乾淨分離、在全部資料上擬合前處理、跨切分的重複樣本、不正當的特徵，以及測試集與目標母體不符 [5]。在辨識專題中，最常見的形式是：

- 在切分前，就用全部資料擬合標準化、PCA 或特徵選擇；
- 看著測試分數選擇超參數、特徵或門檻；
- 近乎重複的影像（連續影格、擴增後的副本、同一個人）落在切分兩側；
- 測試集來自與部署不同的母體（例如部署時面對新書寫者，測試卻只有同一群書寫者）。

解法是從結構上著手：把每一個需要擬合的轉換都放進 scikit-learn 的 `Pipeline`，讓交叉驗證在每一折的訓練部分重新擬合；並且在第 8 步之前，任何函式呼叫都不碰測試索引。

步驟 3 — 觀察資料並建立基準

白話版。訓練任何巧妙的模型之前，先看看樣本，並計算簡單方法能做到多好。一個花俏模型若只比極簡模型好一點點，就不算成功。

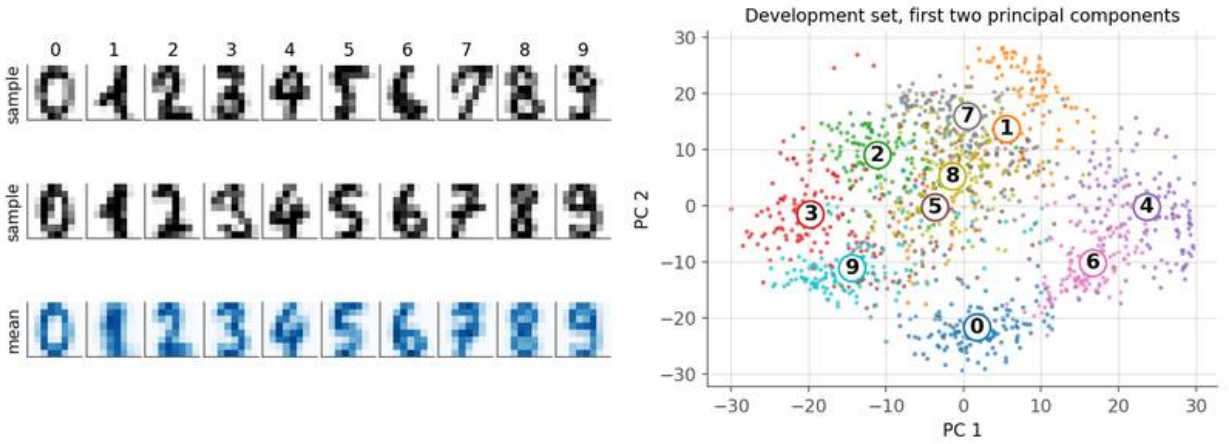


圖 2 — 左：每個類別兩張開發集影像與該類別的平均影像（也就是最近類別平均分類器的樣板）。右：開發集投影到前兩個主成分。數字 *0*、*4*、*6* 已大致分開；*1*、*7*、*8*、*5*、*9* 在這個二維視圖中嚴重重疊。

圖 2 已透露出難度的結構。類別平均看起來像乾淨的數字，所以樣板比對（PR 06）應該有不錯的效果。在兩個主成分中，有些類別分得很開，有些重疊；PR 05 解釋過，變異大的方向不一定最具鑑別力，因此兩個成分中的重疊，並不代表 64 維中也重疊。

我們使用四個基準，各自對應一章：

- **多數類別**。永遠預測最常見的數字。十個平衡類別下，準確率約 10%。這是下限。
- **最近類別平均**。把影像和每個類別平均比較，選最接近的：最簡單的樣板比對器。
- **高斯單純貝氏**。一種貝氏分類器（PR 01），類別條件密度為對角共變異數的高斯分布。像素之間高度相關，獨立假設在這裡嚴重不成立，我們預期它表現較差。
- **k 最近鄰**。一種以密度為基礎的無母數規則（PR 02）：由最接近的 k 張訓練影像投票。

步驟 4 — 決定特徵

白話版。分類器只看得到數字。要決定用哪些數字、如何縮放、要不要壓縮，而且這些轉換只能在訓練折上擬合。

我們的原始特徵是 64 個區塊計數。有三種常見轉換值得比較：

標準化。減去每個特徵在訓練資料上的平均，再除以標準差：

$$z_j = \frac{x_j - \bar{x}_j}{s_j},$$

其中 x_j 是第 j 個像素，$\bar{x}_j$ 與 s_j 是它在訓練折上的平均與標準差，z_j 是標準化後的值。當特徵單位差異很大時，這對以距離或梯度為基礎的方法很重要。但這裡所有特徵本來就是同一個單位（0 到 16 的計數）。更糟的是，邊緣像素幾乎總是 0，s_j 很小：標準化會把偶爾出現的雜散筆畫放大成很大的值。我們會看到這個選擇明顯拖累核 SVM。

主成分分析。 PCA 把資料投影到變異最大的方向（PR 05）。保留 d' 個成分時，表徵為 $u = W^\top(x - \bar{x})$，其中 W 存放訓練共變異數前 d' 個特徵向量。它能移除雜訊方向並加速核方法，但變異不等於類別資訊。

線性鑑別分析。 LDA（PR 03、PR 05）找出最多 $K - 1 = 9$ 個方向，使類間散布相對於類內散布最大。我們把它當作帶收縮（shrinkage）的分類器使用，收縮可正則化類內共變異數的估計。

> **特徵也是超參數**
>
> 要不要標準化、PCA 保留幾個成分、要加入哪些手工特徵，都是用資料做出的選擇。請像對待正則化常數一樣對待它們：放進管線，讓驗證來決定。在交叉驗證之前就用全部資料擬合一次，就是洩漏。

對真實影像而言，這一步也是加入工程特徵（HOG、LBP、矩等，見特徵擷取一章）或預訓練嵌入的地方。在 8×8 的解析度下，原始計數是合理的起點。

步驟 5 — 比較候選分類器

白話版。讓每個候選模型都跑過開發集上同一組交叉驗證切分，比較它們的分數分布，而不是只看最好的那一折。

```
from sklearn.model_selection import RepeatedStratifiedKFold, cross_val_score
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.dummy import DummyClassifier
from sklearn.neighbors import NearestCentroid, KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier

candidates = {
    "majority class (PR 01)": DummyClassifier(strategy="most_frequent"),
    "nearest class mean (PR 06)": NearestCentroid(),
    "Gaussian naive Bayes (PR 01)": GaussianNB(),
    "k-NN, k=3 (PR 02)": KNeighborsClassifier(n_neighbors=3),
    "LDA, shrinkage (PR 03/05)": LinearDiscriminantAnalysis(
        solver="lsqr", shrinkage="auto"),
    "logistic regression (PR 03)": make_pipeline(
        StandardScaler(), LogisticRegression(C=1.0, max_iter=3000)),
    "PCA-30 + RBF SVM (PR 04/05)": make_pipeline(
        StandardScaler(), PCA(n_components=30, random_state=0),
        SVC(C=10, gamma="scale")),
    "random forest (PR 09)": RandomForestClassifier(
```

```
        n_estimators=300, random_state=0, n_jobs=1),
}
cv = RepeatedStratifiedKFold(n_splits=5, n_repeats=2, random_state=0)
cv_scores = {}
for name, model in candidates.items():
    s = cross_val_score(model, X[dev], y[dev], cv=cv, scoring="accuracy")
    cv_scores[name] = s
    print(f"{name:30s} {s.mean():.3f} +/- {s.std():.3f}")
```

輸出（10 折的平均 ± 標準差）：

候選模型	交叉驗證平均準確率	標準差
多數類別	0.100	0.000
高斯單純貝氏	0.827	0.017
最近類別平均	0.897	0.011
帶收縮的 LDA	0.951	0.007
邏輯斯迴歸（標準化）	0.966	0.012
隨機森林，300 棵樹	0.975	0.010
標準化 + PCA-30 + RBF SVM	0.976	0.010
3 最近鄰	0.985	0.007

已經可以看出幾個教訓：

- **簡單模型的排名反映了它們的假設**。單純貝氏的獨立假設被相鄰像素間的相關性打破，是最弱的真正模型。每類只有一個樣板（最近平均）無法涵蓋同一數字的不同寫法。LDA 建模共享的完整共變異數，表現好得多。
- **未調參模型中最好的，是最簡單的無母數模型**。 3-NN 勝過所有未調參的有母數模型。1,347 張小影像、只有 13 位書寫者，許多影像在訓練折中都有近乎雙胞胎的樣本，這正是最近鄰最擅長的情況。這也提醒我們，這種切分衡量的是「已知書寫者的新影像」。
- **預設值不是公平的比較**。 SVM 與邏輯斯迴歸帶著標準化步驟與接近預設的設定。下結論前，強的候選模型值得調參，而且要在不會讓比較產生偏誤的協定下進行。

步驟 6 — 調參與選模，不自欺

白話版。調參是方法的一部分。要估計「含調參的 SVM」有多好，必須在每一個外層折內重做整個調參。然後再問：與次佳模型的剩餘差距，是否大於雜訊？

巢狀交叉驗證

內層迴圈選超參數，外層迴圈評估整個流程。如果直接回報內層最佳分數，等於回報許多雜訊數字中的最大值，必然偏樂觀。Varma 與 Simon 以實驗展示了交叉驗證選模的這種偏誤 [6]；Cawley 與 Talbot 則指出，過度擬合選模準則所造成的效能差異，可能與不同學習演算法之間的差異一樣大 [7]。Raschka 的教學式回顧整理了實務做法 [9]。

```
from sklearn.model_selection import GridSearchCV, StratifiedKFold
from sklearn.pipeline import Pipeline

svm = Pipeline([("scale", StandardScaler()), ("pca", PCA(random_state=0)),
                ("svc", SVC())])
grid = {"scale": [StandardScaler(), "passthrough"],
        "pca": [PCA(30, random_state=0), "passthrough"],
        "svc__C": [1, 10],
        "svc__gamma": ["scale", 0.001]}
inner = StratifiedKFold(5, shuffle=True, random_state=1)
outer = StratifiedKFold(5, shuffle=True, random_state=2)
search = GridSearchCV(svm, grid, cv=inner, scoring="accuracy")
nested = cross_val_score(search, X[dev], y[dev], cv=outer)
print("nested-CV accuracy of the tuning procedure:",
      nested.round(3), "mean", nested.mean().round(3))
search.fit(X[dev], y[dev])
print("chosen on all development data:", search.best_params_)
```

輸出：外層各折準確率 `[0.993 0.993 0.985 0.985 0.993]`，平均 `0.99`。在全部開發資料上選出的設定是**不標準化、PCA 保留 30 個成分**、$C = 10$、$\gamma = 0.001$。

兩點觀察。第一，網格本身包含了前處理的決定（`"passthrough"` 表示「略過這一步」），而驗證結果拒絕了標準化，印證了步驟 4 的論點。第二，巢狀估計值（0.990）描述的是「執行這個網格搜尋，再重新擬合」這個流程，也就是我們要部署的東西；它不是某一組超參數的性質。

贏家真的比較好嗎？

調參後的 SVM（巢狀 0.990）與 3-NN（0.985）相差半個百分點。這是真的嗎？對各折分數做一般的成對 t 檢定，會把各折當成獨立樣本；但不同折的訓練集高度重疊，各折分數正相關，一般檢定因此過度自信。Nadeau 與 Bengio 為重抽樣設計提出了校正後的變異數 [8]：

$$t = \frac{\bar{d}}{\sqrt{\left(\frac{1}{k} + \frac{n_{\text{te}}}{n_{\text{tr}}}\right)\hat{\sigma}_d^2}},$$

其中 d_i 是兩個模型在第 i 次重抽樣上的分數差，$\bar{d}$ 與 $\hat{\sigma}_d^2$ 是這 k 個差值的平均與樣本變異數，n_{tr} 與 n_{te} 是每次重抽樣的訓練與測試筆數。額外的 $n_{\text{te}}/n_{\text{tr}}$ 項放大變異數，以反映重疊。統計量與自由度 $k-1$ 的 t 分布比較。PR 09 推導了這個檢定，並討論 McNemar 檢定等替代方案。

```
from scipy import stats

def corrected_t(a, b, n_train, n_test):
    """Nadeau-Bengio corrected resampled t-test on paired fold scores."""
    d = np.asarray(a) - np.asarray(b)
    k = len(d)
    var = d.var(ddof=1) * (1 / k + n_test / n_train)
    t = d.mean() / np.sqrt(var)
```

```
    return t, 2 * stats.t.sf(abs(t), df=k - 1)

n_test_fold = len(dev) // 5
tuned = cross_val_score(search.best_estimator_, X[dev], y[dev], cv=cv)
t, p = corrected_t(tuned,
                   cv_scores["k-NN, k=3 (PR 02)"],
                   n_train=len(dev) - n_test_fold, n_test=n_test_fold)
print(f"tuned SVM vs k-NN: corrected t = {t:.2f}, p = {p:.3f}")
```

輸出：`tuned SVM vs k-NN: corrected t = 1.03, p = 0.331`。

這個輸出應視為**變異數修正的探索性示範**，不是確認性的顯著性檢定。SVM 設定在重複切折前已使用全部開發標籤選定；修正處理的是折間重疊，不是這項選擇偏差 [7, 8]。不顯著也不能證明兩者等效。若要重新比較，應使用 `cross_val_score(search, ...)`，讓每個評估折內都重新調參，並在相同外層切分上評估預先指定的比較方法，再交代統計檢定仍有的近似。此處保留 3-NN 作為強力替代方案，依先前巢狀評估及儲存／預測需求繼續使用調參後的 SVM，不宣稱已在統計上證明平手。

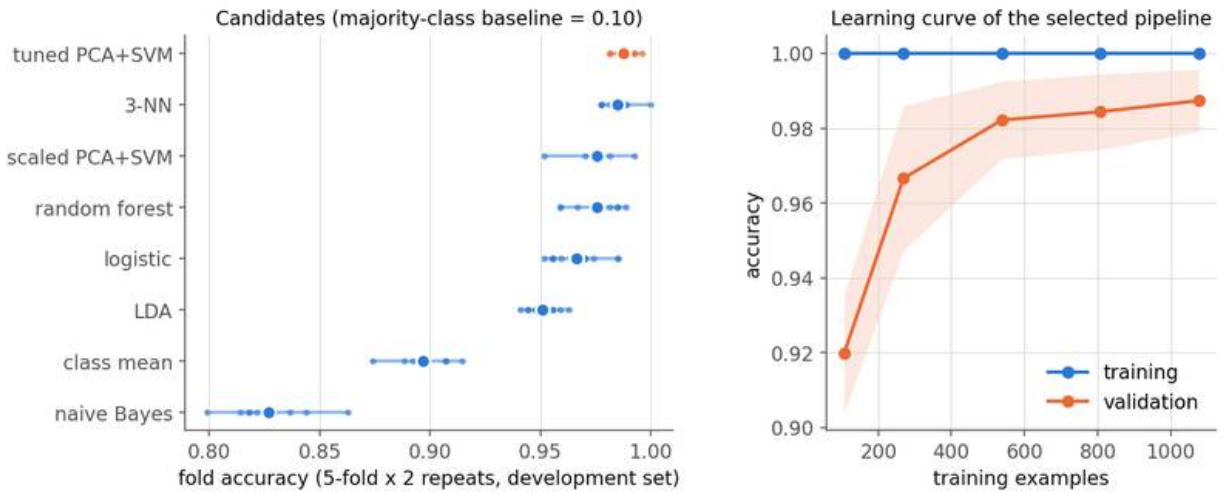


圖 *3* — 左：*10* 個開發折上的準確率（小點）、平均（大點）與範圍（橫條）。調參後的管線（橘色）使用搜尋選出的設定。右：所選管線的學習曲線；陰影為各折間 ± 1 個標準差。

步驟 7 — 在開發資料上做錯誤分析

白話版。去看錯誤，但要看開發資料上的錯誤。一旦因為測試錯誤而重新設計系統，測試集就不再是測試集了。

折外預測

`cross_val_predict` 會為每張開發影像回傳一個「沒有用它訓練過」的模型所做的預測。這些預測適合用於開發階段的錯誤分析。但 `chosen` 已使用全部開發標籤選定：每張影像只被排除於係

數擬合，並未被排除於超參數選擇。因此不能把此錯誤數報成調參程序的不偏評估；若目標是評估完整程序，就要把完整搜尋放進外層預測迴圈 [7]。

```
from sklearn.model_selection import cross_val_predict
from sklearn.metrics import confusion_matrix

chosen = search.best_estimator_
oof = cross_val_predict(chosen, X[dev], y[dev],
                        cv=StratifiedKFold(5, shuffle=True, random_state=3))
cm_dev = confusion_matrix(y[dev], oof)
off = cm_dev.copy()
np.fill_diagonal(off, 0)
order = np.argsort(off.ravel())[::-1][:5]
print("out-of-fold errors:", int(off.sum()), "of", len(dev))
for t_, p_ in zip(*np.unravel_index(order, off.shape)):
    print(f"  true {t_} -> predicted {p_}: {off[t_, p_]}")
dev_errors = dev[oof != y[dev]]
print("first misclassified dataset indices:", dev_errors[:8])
```

輸出：1,347 張中有 17 個錯誤。最常見的混淆是 9 → 8（3 次）、5 → 9（3 次）、8 → 1（2 次），其次是 9 → 5、6 → 8 等單次錯誤。

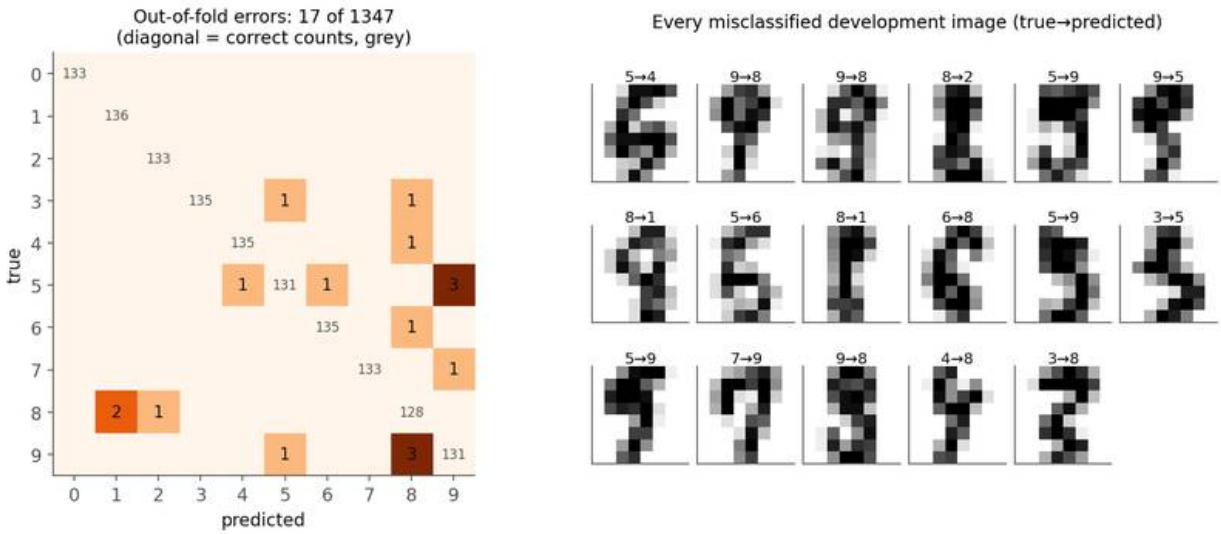


圖 4 — 所選管線的折外錯誤分析。左：非對角線計數（對角線以小字顯示正確數量）。右：每一張分錯的開發集影像，標示真實 → 預測。

解讀圖 4 時，就需要領域知識：

- **系統性混淆**。錯誤集中在上方有封閉圈的數字（5、8、9）與細長直筆畫的數字（1、8）。在 8×8 解析度下，區分 9 與 8 的缺口可能只有一格。
- **標籤或品質問題**。有些分錯的影像連人都難以判斷。在較大的專題中，你會逐一檢視，判斷它們是標註錯誤（要修正）、本質上模稜兩可的輸入（第 9 步拒絕的候選），還是一般失誤（以特徵或資料改善）。

- **把錯誤分群**。錯誤有上百個時，無法逐一檢查。常見技巧是在特徵空間中對分錯的樣本分群（PR 07），再檢查每群的幾個成員；一個群集通常對應一種失敗模式，例如某種書寫風格或擷取問題。

好的錯誤分析最後會得到假設，每個假設都搭配一個能在開發資料上執行的實驗，例如：「讓模型對一格平移不敏感，可能改善 5/9 混淆；試著在訓練折中加入平移後的副本，再重跑巢狀比較。」

更多資料有用嗎？學習曲線

```
from sklearn.model_selection import learning_curve

sizes, tr, va = learning_curve(
    chosen, X[dev], y[dev], cv=StratifiedKFold(5, shuffle=True, random_state=3),
    train_sizes=[0.1, 0.25, 0.5, 0.75, 1.0])
for n, a, b in zip(sizes, tr.mean(1), va.mean(1)):
    print(f"n={n:4d}  train={a:.3f}  validation={b:.3f}")
```

訓練影像數	訓練準確率	驗證準確率
107	1.000	0.920
269	1.000	0.967
538	1.000	0.982
807	1.000	0.984
1,077	1.000	0.987

各種訓練量下訓練準確率都是滿分，驗證準確率在右端仍緩慢上升（圖 3 右）。以偏差–變異的語言（PR 09）來說，剩下的誤差主要是變異：模型已經夠有彈性，更多（或更多樣）的訓練資料可能還有一點幫助。若兩條曲線一起停在低處，建議就相反：需要表達力更強的模型或更好的特徵。

步驟 8 — 打開測試集，只打開一次

白話版。凍結所有選擇，把最終管線用在鎖住的測試影像上，並連同不確定性與適用範圍一起報告結果。

```
from sklearn.metrics import (accuracy_score, balanced_accuracy_score,
                             classification_report)

final = search.best_estimator_            # already refit on all of dev
pred = final.predict(X[test])
print("TEST accuracy:", round(accuracy_score(y[test], pred), 4),
      "balanced:", round(balanced_accuracy_score(y[test], pred), 4),
      "errors:", int((pred != y[test]).sum()), "of", len(test))
cm = confusion_matrix(y[test], pred)
print(cm)
```

```
print(classification_report(y[test], pred, digits=3))

k, n, z = int((pred == y[test]).sum()), len(test), 1.96   # Wilson interval
phat = k / n
centre = (phat + z**2 / (2 * n)) / (1 + z**2 / n)
half = z * np.sqrt(phat * (1 - phat) / n + z**2 / (4 * n**2)) / (1 + z**2 / n)
print(f"95% Wilson interval for accuracy: [{centre - half:.3f}, {centre + half:.3f}]")
```

輸出：測試準確率 **0.9933**（450 張中 3 個錯誤：3 → 7、8 → 1、9 → 5 各一次），平衡準確率 0.9933，各類別召回率介於 0.977 到 1.000，95% Wilson 區間為 **[0.981, 0.998]**。

如何解讀與報告：

- **報告數量，而不只是百分比**。「450 張中錯 3 張」能讓讀者立刻知道小數點第三位有多可信。
- **報告區間**。準確率接近 1 時，常用的常態近似 $\hat{p} \pm 1.96\sqrt{\hat{p}(1-\hat{p})/n}$ 可能超過 1，並不可靠；上面使用的 Wilson 分數區間會留在 $[0,1]$ 之內，在邊界附近表現較好。若有 n 筆測試樣本、k 筆正確、$\hat{p} = k/n$，其中心為 $(\hat{p} + z^2/2n)/(1 + z^2/n)$，95% 涵蓋時 $z = 1.96$。
- **與開發估計比較**。測試結果（0.993）與巢狀估計（0.990）一致。若低很多，可能原因是開發過程有洩漏，或測試母體與開發不同，兩者都值得追查；但若因此修改系統，就需要一份新的未碰過的測試集。
- **說明範圍**。這是同樣 13 位書寫者的新影像上的準確率（步驟 1），不能證明對新書寫者、新掃描器或新前處理的表現。

程式中的協定註解很重要：`search.best_estimator_` 在測試前已在全部開發資料上重新擬合一次。若再用開發加測試資料重新擬合，然後報告測試分數，就毫無意義。部署時可以用全部有標籤的資料重新擬合凍結的流程，但報告的數字必須來自上面這次執行。

步驟 9 — 超越封閉集合準確率

一個在已知數字的平衡測試上達到 99% 的辨識器，若會自信地把一個英文字母標成數字，或在掃描器把影像平移時崩潰，依然可能毫無用處。這一步檢驗三個封閉集合數字無法回答的問題。

四種容易混淆的任務

任務	系統必須做什麼	評估必須包含什麼
封閉集合分類	在訓練過的類別中選一個	已知類別的獨立樣本
選擇性分類	對困難的已知樣本棄權	已知樣本的涵蓋率與被接受樣本的錯誤率
開放集合辨識	分類已知類別，並拒絕未知類別	保留的未知類別，以及已知樣本
異常偵測	標出偏離參考母體的樣本	定義清楚的正常母體與相關的偏離

信心低的已知樣本不一定是未知類別；不熟悉的類別可能得到很高的信心；一個有效的已知物件若拍得很差，也可能被拒絕。這些任務需要不同的資料與指標。

選擇性分類：涵蓋率與風險

白話版。讓分類器略過最沒把握的輸入，並量測兩件事：它還回答了多少輸入，以及這些回答有多常出錯。

設信心分數為 $q(x)$、接受門檻為 τ，在 N 筆已知類別測試樣本上，預測為 $\hat{y}_n$、標籤為 y_n：

$$\text{coverage}(\tau) = \frac{1}{N}\sum_{n=1}^{N} \mathbf{1}[q(x_n) \geq \tau],$$

$$\text{selective risk}(\tau) = \frac{\sum_n \mathbf{1}[q(x_n) \geq \tau]\,\mathbf{1}[\hat{y}_n \neq y_n]}{\sum_n \mathbf{1}[q(x_n) \geq \tau]}.$$

這裡 $\mathbf{1}[\cdot]$ 在條件成立時為 1，否則為 0。沒有任何樣本被接受時，選擇性風險沒有定義。掃描 τ、畫出風險對涵蓋率的關係，就得到風險–涵蓋率曲線；Geifman 與 El-Yaniv 說明了如何為深度網路選擇門檻，使其以高機率保證達到指定的風險 [11]。有用的門檻必須在驗證資料上選定，再只在測試資料上報告。

開放集合辨識：保留的類別

白話版。訓練時假裝某些數字不存在，再檢查它們出現時系統是否會拒絕。

Scheirer 等人將開放集合辨識形式化：分類器必須限制它標為已知的「開放空間」，否則一個以 K 類訓練的模型，會把世上任何輸入都歸到其中一類 [13]。最簡單的基準分數是最大 softmax（或類別）機率，由 Hendrycks 與 Gimpel 提出，用來偵測分錯的樣本與分布外樣本 [12]。

我們以縮小的標籤空間重做實驗。數字 0–5 為**已知**；數字 6–7 為開發階段可以看的**代理未知**；數字 8–9 為保留到最後評估的**測試未知**。驗證在兩條預先定義的管線之間做選擇，拒絕門檻設為驗證信心的第 5 百分位數，目標是接受約 95% 的已知數字。所有選擇在最後區塊之前就凍結。這支腳本是獨立的。

這是獨立的六類別實驗。變數放在 `open_set_demo()` 內，讓原本十類別模型、開發索引與鎖定的測試索引保持不變，供後面的壓力測試使用。

```
def open_set_demo():
    import numpy as np
    from sklearn.datasets import load_digits
    from sklearn.model_selection import train_test_split
    from sklearn.pipeline import make_pipeline
    from sklearn.preprocessing import StandardScaler
    from sklearn.decomposition import PCA
    from sklearn.linear_model import LogisticRegression
    from sklearn.svm import SVC
    from sklearn.metrics import balanced_accuracy_score, confusion_matrix, roc_auc_score

    digits = load_digits()
    X, y = digits.data, digits.target
    known = np.flatnonzero(y < 6)
    train, rest = train_test_split(known, test_size=0.4,
                                   stratify=y[known], random_state=10)
```

```
    valid, test = train_test_split(rest, test_size=0.5,
                                   stratify=y[rest], random_state=10)
    proxy_unknown = np.flatnonzero((y == 6) | (y == 7))
    test_unknown = np.flatnonzero(y >= 8)
    assert not set(train) & (set(valid) | set(test))
    assert not set(proxy_unknown) & set(test_unknown)

    candidates = {
        "linear": make_pipeline(
            StandardScaler(), LogisticRegression(C=1.0, max_iter=1500)),
        "PCA + RBF": make_pipeline(
            StandardScaler(), PCA(n_components=24, random_state=10),
            SVC(C=4.0, gamma=0.02, probability=True, random_state=10)),
    }
    scores = {}
    for name, model in candidates.items():
        model.fit(X[train], y[train])
        validation_p = model.predict_proba(X[valid])
        validation_pred = model.classes_[validation_p.argmax(axis=1)]
        scores[name] = balanced_accuracy_score(y[valid], validation_pred)
    name = max(scores, key=scores.get)
    model = candidates[name]
    threshold = np.quantile(model.predict_proba(X[valid]).max(axis=1), 0.05)
    proxy_conf = model.predict_proba(X[proxy_unknown]).max(axis=1)
    print("Validation candidates:", scores, "selected:", name)
    print("Fixed threshold:", threshold)
    print("Proxy-unknown acceptance:", np.mean(proxy_conf >= threshold))

    # Final evaluation begins here; do not revise choices from these results.
    known_p = model.predict_proba(X[test])
    known_conf = known_p.max(axis=1)
    known_pred = model.classes_[known_p.argmax(axis=1)]
    accept = known_conf >= threshold
    unknown_conf = model.predict_proba(X[test_unknown]).max(axis=1)
    print("Known test balanced accuracy:",
          balanced_accuracy_score(y[test], known_pred))
    print("Known coverage:", accept.mean())
    print("Accepted known accuracy:",
          np.mean(known_pred[accept] == y[test][accept]) if accept.any() else np.nan)
    print("Unknown-test acceptance:", np.mean(unknown_conf >= threshold))
    is_unknown = np.r_[np.zeros(len(test)), np.ones(len(test_unknown))]
    unknown_score = 1 - np.r_[known_conf, unknown_conf]
    print("Unknown detection AUROC:", roc_auc_score(is_unknown, unknown_score))
    print("Known-class confusion matrix:\n", confusion_matrix(y[test], known_pred))

open_set_demo()
```

依輸出順序：驗證平衡準確率 0.982（線性）對 0.995（PCA + RBF），因此選 RBF 管線；門檻 0.741；**代理未知接受率 0.083**；已知測試平衡準確率 0.981；已知涵蓋率 0.949；被接受的已知樣本準確率 0.995；**測試未知接受率 0.350**；未知偵測 AUROC 0.944。

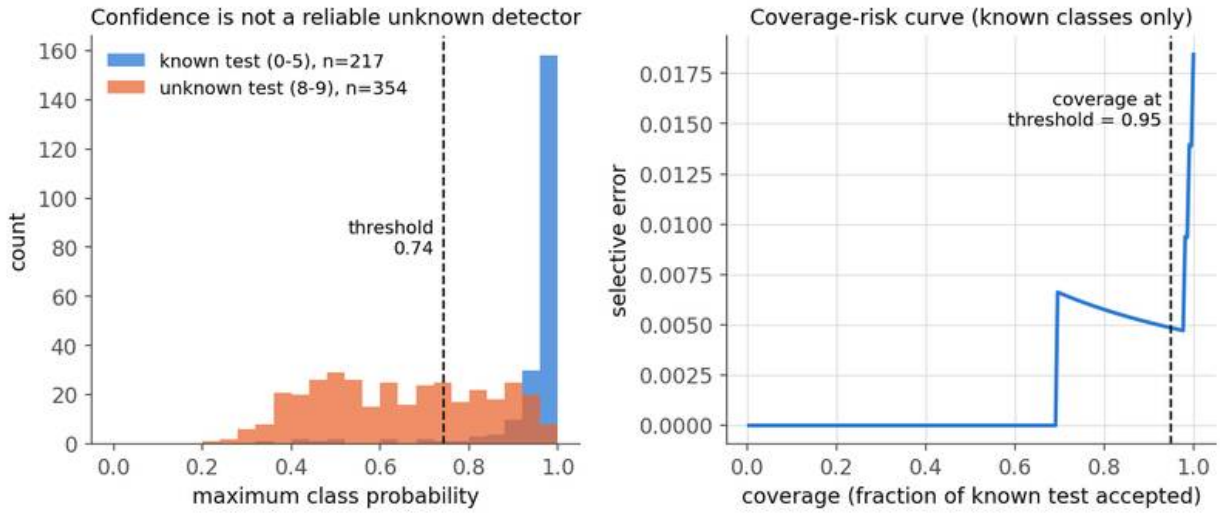


圖 *5* — 左：已知測試數字（*0–5*）與未知測試數字（*8–9*）的最大類別機率，門檻在驗證資料上固定。右：已知測試數字的風險–涵蓋率曲線；虛線為固定門檻下的涵蓋率。

這些數字道出開放集合評估的主要教訓：

- 在已知數字上，拒絕機制如預期運作：涵蓋率 95%，被接受的樣本準確率 99.5%。
- 代理未知（6 與 7）有 92% 被拒絕，這可能讓人以為信心門檻「解決了」未知類別。
- 真正的測試未知（8 與 9）**有 35% 被接受**。8 看起來像多了筆畫的 3 或 5，9 看起來像 5 或 3；分類器很有信心地把它們對應到這些已知數字。
- AUROC 0.944 概括的是對這幾種特定未知類別的排序能力，它不會決定操作門檻，也不代表其他種類的新奇輸入。

在一組代理未知上的表現，不會轉移到其他未知類別。請報告保留了哪些類別，也絕不要宣稱能拒絕「所有不熟悉的輸入」。Vaze 等人發現，較強的封閉集合分類器往往也是較好的開放集合辨識器，兩者在各方法間高度相關 [16]；因此改進基礎模型與使用更好的分數（特徵距離、能量或 logit 分數）是互補的。

另一類分數直接建模參考母體：單類 SVM（one-class SVM）估計包含大部分訓練資料的區域 [17]；孤立森林（Isolation Forest）則標出能被隨機軸向切分迅速孤立的點 [18]。「離群值」可能是擷取錯誤、罕見但有效的案例，或是新類別；統計上的不尋常並不會告訴你是哪一種。

分布漂移與壓力測試

白話版。部署之後，世界會改變。用受控的擾動提出「如果……」的問題，更好的做法是用來自新條件的真實資料。

令 $p_{\text{train}}(x, y)$ 與 $p_{\text{deploy}}(x, y)$ 分別為訓練與部署時特徵與標籤的聯合分布。Moreno-Torres 等人整理了分類問題中資料集漂移的術語 [19]。最重要的三種情況是：

- **先驗（標籤）漂移：**類別頻率 $p(y)$ 改變，$p(x \mid y)$ 不變。例如新表單版面讓某個數字變得常見許多。

- **共變量漂移：** $p(x)$ 改變，$p(y \mid x)$ 不變。例如新掃描器讓筆畫變粗。
- **概念漂移：** $p(y \mid x)$ 本身改變。例如寫 7 的新慣例多了一橫。

這些是建模假設，不是能從資料名稱讀出的診斷。換相機可能改變類別條件特徵；加入新類別則會徹底打破標籤空間。

我們無法在這裡蒐集新的手寫資料，但可以對測試影像施加受控擾動來探測敏感度。模型不重新訓練；擾動方式在看結果之前就決定好。

```
rng = np.random.default_rng(0)
def shift_right(X, k=1):
    imgs = X.reshape(-1, 8, 8)
    out = np.zeros_like(imgs)
    out[:, :, k:] = imgs[:, :, :-k]
    return out.reshape(len(X), 64)

conditions = {
    "clean": X[test],
    "noise sd=2": np.clip(X[test] + rng.normal(0, 2, X[test].shape), 0, 16),
    "noise sd=4": np.clip(X[test] + rng.normal(0, 4, X[test].shape), 0, 16),
    "shift 1 px right": shift_right(X[test], 1),
    "contrast x0.5": X[test] * 0.5,
}
for name, Xc in conditions.items():
    print(f"{name:18s} accuracy = {accuracy_score(y[test], final.predict(Xc)):.3f}")
```

（請接在步驟 8 的程式之後執行，而不是開放集合腳本之後，因為它使用十類版本的 `X`、`test` 與 `final`。）

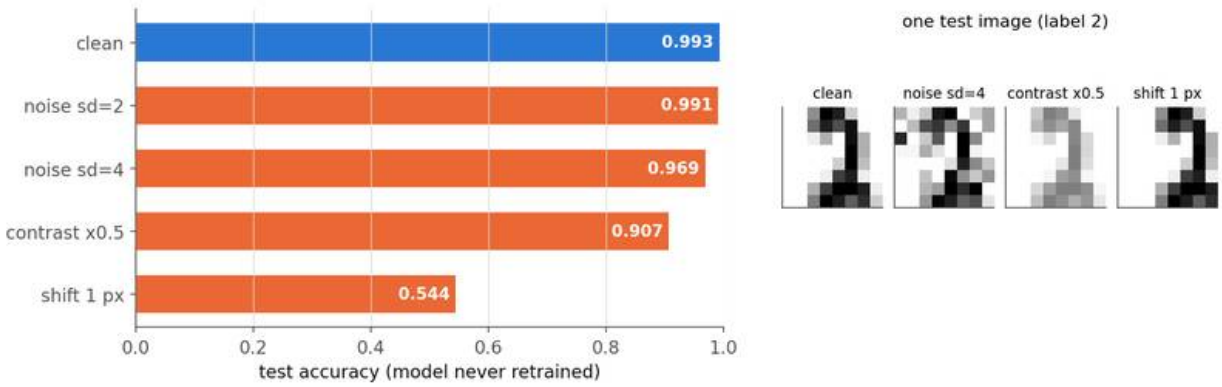


圖 *6* — 凍結管線在訓練時從未見過的擾動下的測試準確率，以及一張測試影像在各條件下的樣子。強度維持在資料集的 *0–16* 範圍內。

條件	測試準確率
乾淨	0.993
加性雜訊，標準差 2	0.991
加性雜訊，標準差 4	0.969

條件	測試準確率
對比減半	0.907
向右平移一格	**0.544**

加性雜訊幾乎沒有影響。對比減半損失約 9 個百分點，因為 RBF 核直接比較原始強度，而 γ 是針對全對比筆畫調的。**平移一格**幾乎讓準確率減半：以像素向量為輸入的分類器，並不知道平移後的 3 仍然是 3。一格相當於原始 32×32 點陣圖中 4 個像素的區塊，也就是影像寬度的八分之一，對裁切不精確的情況而言是很實際的錯位。這是把平移容忍度內建進模型的經典理由：透過資料擴增，或透過卷積與池化，LeCun 等人在文件辨識中展示了這一點 [10]（見 ML 03）。

合成擾動是便宜的初步檢查，無法取代新條件下的真實資料。WILDS 等基準蒐集了跨醫院、相機與地區的真實分布漂移，顯示從分布內到分布外有明顯的效能下降 [21]；Recht 等人依原始流程重建 CIFAR-10 與 ImageNet 測試集，即使沒有刻意的漂移，許多模型的準確率仍然下降 [20]。請依有意義的條件（書寫者、裝置、場域、時間）分別報告結果，而不只是彙總。

名副其實的機率

如果下游使用者會依據信心值行動，這些值就必須校準：信心 0.8 的預測中，約 80% 應該是對的。Guo 等人指出現代深度網路常常過度自信，而在驗證資料上擬合單一純量溫度（temperature scaling）是強而有效的修正 [22]。對 SVM 而言，`SVC(probability=True)` 會在內部以交叉驗證擬合 Platt 縮放；`CalibratedClassifierCV` 則提供明確的控制。校準必須在沒有用來訓練基礎模型的資料上擬合，而且它無法修復分布漂移。

銜接學習式與基礎模型特徵

白話版。把手工特徵換成預訓練網路的特徵，本章其餘的一切都不變。

用預訓練編碼器取代特徵映射：

$$x = \phi_{\text{pretrained}}(I), \qquad \hat{y} = g_\theta(x),$$

其中 I 是原始影像，$\phi_{\text{pretrained}}$ 是凍結的網路（例如 DINOv2 這類自監督視覺 Transformer [25]），x 是它的嵌入，g_θ 是以你的標籤訓練、參數為 θ 的分類器。**線性探測**（linear probe，在凍結嵌入上做邏輯斯迴歸）是標準基準；請在固定切分與指標的前提下，將它與嵌入上的 k-NN、正則化的非線性分類頭以及微調比較。ML 06 說明這類特徵如何學得，ML 10 說明如何調適它們。

協定可以自然延伸：

- 記錄檢查點、輸入縮放、正規化、池化方式與嵌入維度；
- 下游前處理（標準化、PCA）只在訓練嵌入上擬合；
- 檢查點、提示詞或層的選擇都算模型選擇，必須在驗證資料上進行；
- 若評估影像可能出現在預訓練資料中，請說明你掌握了哪些關於重疊的證據；
- 以距離為基礎的未知分數（到訓練嵌入的 k-NN 距離）在好的嵌入上往往比在原始像素上有效得多，但仍需要保留的未知類別來評估。

表徵學習改變的是量測的取得方式，它不會改變最小風險決策、誠實的驗證，以及把已知類別準確率與拒絕、漂移分開看待的原則；這正是本系列的框架 [1], [2]。

要繳交什麼

一個完成的辨識專題，至少應包含：

1. **任務定義：**觀測、標籤空間、損失、拒絕政策。
2. **資料卡：**來源、授權、抽樣單位、類別數量、已知限制（本例：13 位書寫者、沒有書寫者編號）[24]。
3. **切分說明：**每個切分的用途、切分單位、亂數種子。
4. **基準與候選模型：**試過的所有模型，而不只是贏家，附上開發分數與其變異。
5. **選模證據：**所選流程的巢狀估計，以及與第二名的統計比較。
6. **錯誤分析：**混淆結構、範例、假設、學習曲線。
7. **最終測試結果：**數量、區間、各類別結果、範圍說明。
8. **壓力測試結果：**拒絕（涵蓋率與風險）、保留的未知類別、漂移或擾動測試；若使用機率，還要有校準。
9. **可重現性：**環境與函式庫版本、完整的管線物件、類別順序；若重要，還有執行時間與記憶體。

模型卡（model cards）將這份清單中面向部署的部分形式化（預期用途、評估條件、在相關族群上的表現、注意事項）[23]。如果原本的測試集影響了任何重新設計，修改後的系統就需要一份新的受保護測試集。

現代觀點

本章流程的原則由來已久，實務仍在演進。以下回顧與研究是很好的入門。

Raschka，〈Model evaluation, model selection, and algorithm selection in machine learning〉（2018）[9]。一篇長篇教學式回顧，把評估分成三個不同目標：估計泛化效能、選擇超參數與模型、比較演算法。內容涵蓋保留法、自助法與交叉驗證、巢狀交叉驗證，以及統計檢定（McNemar、5×2cv 等），並依資料量給出實務建議。它是最接近本章步驟 2 與步驟 6 的單一參考文獻。

Kapoor 與 Narayanan，〈Leakage and the reproducibility crisis in machine-learning-based science〉（*Patterns*，2023）[5]。蒐集了許多科學領域中已發表的洩漏案例，提出洩漏類型的分類，並設計「模型資訊表」（model info sheets），要求作者逐項說明為何不存在各類洩漏。主要結論是：洩漏很常見，往往在報告的數字中看不出來，最好透過協定預防，而不是事後偵測。

Geng、Huang 與 Chen，〈Recent advances in open set recognition: a survey〉（TPAMI，2021）[14]。區分已知的已知、已知的未知、未知的已知與未知的未知等類別，並把開放集合方法分為判別式（傳統機器學習與深度網路）與生成式（實例生成與非實例生成）兩大家族。它也說明開放集合辨識與零樣本、單樣本以及帶拒絕的分類之間的關係，並回顧常用的資料集與評估準則。

Yang、Zhou、Li 與 Liu，〈Generalized out-of-distribution detection: a survey〉（IJCV，2024）

[15]。把異常偵測、新奇偵測、開放集合辨識、分布外偵測與離群值偵測統一在同一個框架中，依漂移屬於語意或共變量、以及分布內任務是什麼來區分。決定你的專題屬於「四種任務」表的哪一列時，應該使用這套詞彙。

Vaze、Han、Vedaldi 與 Zisserman，〈Open-set recognition: a good closed-set classifier is all you need?〉（ICLR，2022）[16]。顯示封閉集合準確率與開放集合表現高度相關，訓練良好的基準搭配最大 logit 分數即可與專門方法競爭；並提出「語意漂移基準」（Semantic Shift Benchmark），把語意上的新奇與低階漂移分開。

Koh 等人的 WILDS（ICML，2021）[21] 與 Recht 等人（ICML，2019）[20]。這是評估漂移的兩種互補視角：WILDS 整理跨領域的真實部署漂移；Recht 等人則顯示，即使小心地「從同一分布」重新蒐集測試集，準確率仍可能下降。兩者共同主張依條件分別報告結果，並把單一的獨立同分布測試分數視為上限。

Guo 等人，〈On calibration of modern neural networks〉（ICML，2017）[22] 與 **Geifman 與 El-Yaniv（NeurIPS，2017）[11]** 是深度模型信心品質與選擇性預測的起點。

最新進展（2023–2026）。實務上，辨識專題越來越常從凍結或輕度微調的基礎模型特徵（例如 DINOv2 [25]）加上線性或 k-NN 分類頭開始，把省下的力氣投入資料品質、評估設計與文件記錄。分布外評估則朝向統一的基準，以及 [15] 所提出的語意與共變量區分發展。資料表 [24] 與模型卡 [23] 提供了現成的範本，可與這些結果一起記錄資料與模型。

開放問題。

- **未知的未知**。任何保留的「未知」類別都只是一個樣本；沒有通用的方法能保證拒絕所有新奇輸入。
- **漂移下的評估**。為每一種部署條件蒐集有標籤的資料代價高昂，而合成擾動並不能可靠地預測真實漂移。
- **預訓練重疊**。在網路規模的預訓練下，越來越難保證測試影像從未被看過，這讓「鎖住測試集」的原則更加複雜。
- **小樣本比較**。許多應用專題只有數百筆而非數百萬筆有標籤樣本；此時本章每個估計的變異都很大，統計比較不可或缺。

重點整理

- 先寫協定：觀測、抽樣單位、標籤空間、損失、拒絕政策與每個切分的角色。多數評估失敗其實是協定失敗。
- 在任何建模之前鎖住測試集，把每個需要擬合的轉換放進管線，並把特徵與前處理當成超參數。
- 在相同的折上比較所有候選模型；基準告訴你什麼叫「好」，而在這份數字資料上，3-NN 基準很難被打敗。
- 用巢狀交叉驗證估計含調參的流程，並在宣稱某模型較好之前使用校正後的重抽樣檢定。
- 在開發資料的折外預測上做錯誤分析；用學習曲線判斷該加資料還是換更強的模型。
- 只打開測試集一次，並報告數量、區間與範圍（本例：同樣 13 位書寫者的新影像）。
- 封閉集合準確率、選擇性風險、未知類別拒絕與漂移下的穩健性是不同的性質：一個 99% 準確的數字分類器，仍接受了 35% 的未見數字，且在平移一格時失去將近一半的準確率。

練習

1. **以距離做拒絕**。在開放集合實驗中，把最大類別機率換成測試影像在「以訓練資料擬合的 PCA 空間」中到最近訓練影像的距離。只用驗證集的已知數字選門檻，再在測試集上報告已知涵蓋率與未知接受率。它是否勝過信心門檻？

   提示

   只在 `X[train]` 上擬合 `StandardScaler` 與 `PCA`，再對轉換後的訓練集使用 `NearestNeighbors(n_neighbors=1)`。分數就是距離（越大越像「未知」）；把門檻設在驗證距離的第 95 百分位數，讓約 95% 的已知驗證數字被接受。兩個數字都要報告；看到測試結果後不得更改門檻。

2. **不作弊的風險–涵蓋率曲線**。以十類調參 SVM 的決策值作為信心，先用開發資料的折外預測畫出風險–涵蓋率曲線，再（僅供報告）畫出測試集上的曲線。說明為什麼選擇性準確率可能提高，而有用的涵蓋率卻崩潰；涵蓋率為零時又會怎樣？

   提示

   依信心由高到低排序樣本；每個截斷點的涵蓋率是保留比例，風險是保留樣本中的錯誤率。涵蓋率很低時只剩少數樣本，風險很不穩定；涵蓋率為零時風險沒有定義。在開發曲線上選定任一操作點，再讀出它在測試上的值。

3. **修補平移弱點**。在每一個訓練折中，加入影像向左、右、上、下各平移一格的副本。重跑步驟 6 的巢狀比較與步驟 9 的壓力測試。哪些準確率改變了？為什麼擴增必須在折內進行？

   提示

   寫一個小型轉換器或自訂迴圈，只擴增每次切分中的訓練部分；在切分前擴增，會把驗證影像的平移副本放進訓練，這就是洩漏。預期平移測試的準確率會大幅上升，而乾淨資料的準確率維持相近或略有變化。

4. **以設計達成書寫者不重疊的評估**。 UCI 資料集的訓練部分由 30 人書寫，測試部分由另外 13 人書寫 [3]。（在紙上）設計一個衡量新書寫者泛化能力的實驗。你預期本章哪些數字會改變？只靠內建資料，有哪些結論就無法再下？

   提示

   在一群書寫者上訓練與選模，在另一群上測試。預期 k-NN 的準確率特別會下降，因為它的優勢來自同一書寫者的近似重複樣本。由於同一部分內沒有書寫者編號，開發階段仍無法做分組交叉驗證，所以驗證依然混合了書寫者。

5. **顯著嗎？**使用步驟 5 的 10 個折分數，對邏輯斯迴歸與隨機森林、以及 LDA 與邏輯斯迴歸套用校正後的重抽樣 t 檢定。再計算不含 $n_{\text{te}}/n_{\text{tr}}$ 項的一般成對 t 檢定。在哪種情況下，兩種檢定會得到不同結論？

提示

5 折交叉驗證下 $n_{\text{te}}/n_{\text{tr}} \approx 0.25$，遠大於 $1/k = 0.1$，因此校正後的變異數約為一般變異數的 3.5 倍，t 統計量約縮小 $\sqrt{3.5} \approx 1.9$ 倍。差異小但一致的情況，正是兩者結論分歧之處。

6. **凍結特徵對微調**。針對你選擇的真實影像資料集，設計一個比較「凍結預訓練特徵上的線性探測」與「完整微調」的實驗。列出每一個可能洩漏測試資訊的決策。

提示

包括檢查點與層的選擇、影像縮放與正規化、資料擴增、學習率與訓練輪數的選擇（提前停止）、門檻與校準的擬合、若使用視覺–語言模型則還有提示詞設計，以及預訓練資料與測試影像之間可能的重疊。

參考文獻

1. S. Theodoridis and K. Koutroumbas, *Pattern Recognition*, 4th ed., Academic Press (Elsevier), 2009; Ch. 1 “Introduction”, Ch. 5–7 on feature selection and generation, Ch. 10 “Supervised Learning: The Epilogue”. doi
2. C. M. Bishop, *Pattern Recognition and Machine Learning*, Springer, 2006; Sec. 1.5 (decision theory and the reject option), Sec. 1.3 (model selection), Ch. 14 (combining models). official page and PDF
3. E. Alpaydin and C. Kaynak, “Optical Recognition of Handwritten Digits,” UCI Machine Learning Repository, 1998. doi
4. F. Pedregosa et al., “Scikit-learn: Machine learning in Python,” *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011. JMLR; dataset loader: load_digits
5. S. Kapoor and A. Narayanan, “Leakage and the reproducibility crisis in machine-learning-based science,” *Patterns*, vol. 4, no. 9, 2023. doi
6. S. Varma and R. Simon, “Bias in error estimation when using cross-validation for model selection,” *BMC Bioinformatics*, vol. 7, 91, 2006. doi
7. G. C. Cawley and N. L. C. Talbot, “On over-fitting in model selection and subsequent selection bias in performance evaluation,” *Journal of Machine Learning Research*, vol. 11, pp. 2079–2107, 2010. JMLR
8. C. Nadeau and Y. Bengio, “Inference for the generalization error,” *Machine Learning*, vol. 52, pp. 239–281, 2003. doi
9. S. Raschka, “Model evaluation, model selection, and algorithm selection in machine learning,” arXiv:1811.12808, 2018. arXiv
10. Y. LeCun, L. Bottou, Y. Bengio and P. Haffner, “Gradient-based learning applied to document recognition,” *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998. doi
11. Y. Geifman and R. El-Yaniv, “Selective classification for deep neural networks,” in *Advances in Neural Information Processing Systems 30 (NIPS)*, 2017. proceedings
12. D. Hendrycks and K. Gimpel, “A baseline for detecting misclassified and out-of-distribution examples in neural networks,” in *Proc. ICLR*, 2017. arXiv

13. W. J. Scheirer, A. de Rezende Rocha, A. Sapkota and T. E. Boult, “Toward open set recognition,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 7, pp. 1757–1772, 2013. doi
14. C. Geng, S.-J. Huang and S. Chen, “Recent advances in open set recognition: A survey,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 43, no. 10, 2021. doi
15. J. Yang, K. Zhou, Y. Li and Z. Liu, “Generalized out-of-distribution detection: A survey,” *International Journal of Computer Vision*, 2024. doi
16. S. Vaze, K. Han, A. Vedaldi and A. Zisserman, “Open-set recognition: A good closed-set classifier is all you need?,” in *Proc. ICLR*, 2022. arXiv
17. B. Schölkopf, J. C. Platt, J. Shawe-Taylor, A. J. Smola and R. C. Williamson, “Estimating the support of a high-dimensional distribution,” *Neural Computation*, vol. 13, no. 7, pp. 1443–1471, 2001. doi
18. F. T. Liu, K. M. Ting and Z.-H. Zhou, “Isolation forest,” in *Proc. IEEE International Conference on Data Mining (ICDM)*, 2008. doi
19. J. G. Moreno-Torres, T. Raeder, R. Alaiz-Rodríguez, N. V. Chawla and F. Herrera, “A unifying view on dataset shift in classification,” *Pattern Recognition*, vol. 45, no. 1, pp. 521–530, 2012. doi
20. B. Recht, R. Roelofs, L. Schmidt and V. Shankar, “Do ImageNet classifiers generalize to ImageNet?,” in *Proc. ICML*, PMLR 97, pp. 5389–5400, 2019. PMLR
21. P. W. Koh, S. Sagawa, H. Marklund et al., “WILDS: A benchmark of in-the-wild distribution shifts,” in *Proc. ICML*, PMLR 139, pp. 5637–5664, 2021. PMLR
22. C. Guo, G. Pleiss, Y. Sun and K. Q. Weinberger, “On calibration of modern neural networks,” in *Proc. ICML*, 2017. arXiv
23. M. Mitchell, S. Wu, A. Zaldivar, P. Barnes, L. Vasserman, B. Hutchinson, E. Spitzer, I. D. Raji and T. Gebru, “Model cards for model reporting,” in *Proc. Conference on Fairness, Accountability, and Transparency (FAT*)*, 2019. doi
24. T. Gebru, J. Morgenstern, B. Vecchione, J. W. Vaughan, H. Wallach, H. Daumé III and K. Crawford, “Datasheets for datasets,” *Communications of the ACM*, vol. 64, no. 12, 2021. doi
25. M. Oquab, T. Darcet, T. Moutakanni et al., “DINOv2: Learning robust visual features without supervision,” arXiv:2304.07193, 2023. arXiv