前言

本書適合誰？

介紹性的機器學習書籍通常把重點放在探討機器學習（ML）是什麼，以及如何使用它，然後解釋由 AI 研究室提出的新方法的數學公式，並教導如何使用 AI 框架來實作那些方法。然而，這本書匯整了關於「為什麼」的經驗談，它們是老練的 ML 實踐者用機器學習來解決實際問題時，經常採取的招式和技巧的基礎。

我們假設你已經知道機器學習和資料處理了，本書不是機器學習的基本教科書，如果你是資料科學家、資料工程師或 ML 工程師，而且想要找到實際介紹機器學習的第二本書，本書是為你而寫的。如果你已經知道這些基礎知識，本書將介紹一系列的概念，你（ML 實踐者）可能已經認識其中的一些概念，並且為它們取名字，以便有信心地使用它們了。如果你是電腦科學學生，打算進入業界工作，這本書將充實你的知識，為你做好投入職場的準備，協助你了解如何建構高品質的 ML 系統。

本書不介紹什麼

這本書是專門寫給公司的 ML 工程師的，不是學術界或業界研究室裡面的 ML 科學家。

我們故意不討論活躍的研究領域，例如，本書很少談到機器學習模型架構（例如雙向編碼網路、專注機制，或短路層），因為我們假設你將使用現成的模型架構（像是 ResNet-50 或 GRUCell），而不會自行編寫圖像分類或遞迴神經網路。

以下是我們刻意保持距離的幾個具體領域，因為我們認為這些主題比較適合學校的課程與 ML 研究員：

ML 演算法

例如，我們不討論隨機森林與神經網路之間的差異，這是介紹性的機器學習教科書的主題。

基本組件

我們不討論各種類型的梯度下降優化函式或觸發函式。我們建議使用 Adam 與 ReLU，因為根據我們的經驗，其他的選項改善的性能非常有限。

ML 模型架構

如果你要做圖像分類，我們建議你使用現成的模型，例如 ResNet，或是你在閱讀這本書時最流行的模型。請把設計新的圖像或文本分類模型的工作留給專門處理這些問題的研究人員。

模型層

你不會在這本書看到摺積神經網路或遞迴神經網路，它們都沒有進入本書的資格——首先，它們是基本組件，其次，它們有現成作品的可用。

自訂訓練迴圈

在 Keras 裡面呼叫 `model.fit()` 就可以滿足實踐者的需求了。

本書只討論機器學習工程師在公司的日常工作中使用的模式。

以資料結構為例，雖然大學的資料結構課程都會深入研究各種資料結構的實作，且資料結構研究員必須學習如何以正式的方式表達它們的數學屬性，但實踐者比較務實，企業的軟體開發者只要知道如何有效地使用陣列（array）、鏈結串列（linked list）、對映（map）、集合（set）和樹狀結構（tree）即可。本書是為務實的機器學習實踐者而作。

範例程式

我們會用機器學習（有時用 Keras/TensorFlow，有時用 scikit-learn 或 BigQuery ML）和資料處理（用 SQL）來展示如何實作我們所討論的技術。書中的程式碼都被放在 GitHub repository（*https://github.com/Google CloudPlatform/ml-design-patterns*）上，你可以在那裡找到完整、可運作的 ML 模型。我們強烈鼓勵你試著執行那些範例程式。

第一章

為何需要機器學習設計模式

在工程學科中，設計模式描述的是常見問題的最佳實踐法和解決方案。它們將專家的知識和經驗整理成所有實踐者都可以依循的建議。本書匯整的機器學習設計模式是我們與數百個機器學習團隊合作的過程中觀察到的模式。

什麼是設計模式？

模式的概念，以及「將經過驗證的模式分門別類」的做法起源於建築領域的 Christopher Alexander 和另五位作者合著的書籍 A Pattern Language（Oxford University Press, 1977），他們在這本極具影響力的書裡整理了 253 種模式，並且這樣子介紹它們：

> 每一種模式都提出一個在我們的工作環境中反覆出現的問題，並且說明該問題的核心解決方案，讓你可以無數次地使用同樣的解決方案，而不需要重新摸索。
>
> …
>
> 在敘述每一個解決方案時，我們會提供解決問題所需的關係基本領域（essential field of relationships），但是會用一種極為廣義且抽象的方式，如此一來，你就可以自行解決問題，並且用你自己的方法、根據你的喜好以及當地條件調整它。

例如，在建構住宅時，Light on Two Sides of Every Room 與 Six-Foot Balcony 是考慮人類起居瑣事的兩種模式。想一下在你家裡，你最喜歡的房間，與最不喜歡的房間長怎樣。你最喜歡的房間是不是有兩面牆有窗戶？你最不喜歡的房間呢？ Alexander 這樣說：

> 雙面採光的房間可減少了人和物體周圍的眩光，可讓我們看到較複雜的東西，最重要的是，它能讓我們看到人臉一閃而過的表情細節…

為這個模式命名可以防止設計師不斷重新發現這個原則。然而，為特定的地點設計兩處採光則和建築師的技術有關。同樣地，陽台應該設計成多大？ Alexander 建議的面積是放得下 2 張（不搭配的！）椅子和一張邊桌的 6 英尺 ×6 英尺，或是如果你想要同時擁有遮陽和曬太陽的空間，則是 12 英尺 ×12 英尺。

Erich Gamma、Richard Helm、Ralph Johnson 與 John Vlissides 將這種想法引入到軟體，他們在 1994 年出版的 Design Patterns: Elements of Reusable Object-Oriented Software（Addison-Wesley，1995）裡面列出 23 種物件導向的設計模式：包括 Proxy、Singleton 和 Decorator 等模式，為物件導向領域帶來持久的影響。Association of Computing Machinery (ACM) 在 2005 年將年度 Programming Languages Achievement Award 頒給作者群，以表彰他們的成果「對程式實踐法和程式語言設計」造成的影響。

建構生產環境的機器學習模型已經逐漸成為一門工程學科，它們利用已經被研究環境證實的 ML 方法，並將它應用於商務問題上。隨著機器學習日益成為主流技術，利用經過試驗和驗證的方法來解決反覆出現的問題對實踐者來說非常重要。

我們在 Google Cloud 裡面的工作直接和顧客接觸，這個工作有一個好處是，它讓我們可以接觸來自世界各地的各種機器學習和資料科學團隊和開發者。與此同時，我們每個人都與 Google 內部團隊緊密合作，解決最新的機器學習問題。最後，我們有幸與 TensorFlow、Keras、BigQuery ML、TPU 和 Cloud AI Platform 團隊合作，這些團隊正在推動機器學習研究和基礎設施的民主化，這些經驗都讓我們獲得相當獨特的視角，並整理這些團隊的最佳實踐法。

這本書匯整了 ML 工程常見問題的設計模式和可重複使用的解決方案，例如，Transform 模式（第 6 章）強制拆開輸入、特徵和轉換方法，並且持久保存轉換方法，以簡化將 ML 模型遷移至生產環境的過程。類似地，第 5 章的 Keyed Predictions 模式可以大規模地發送批次預測，（例如）供推薦模型使用。

在每一種模式中，我們會描述它可以處理的問題，然後介紹各種解決方案、那些解決方案的優缺點，以及在這些方案之間進行選擇的建議。這些解決方案的程式是以 SQL（如果你正在使用 Spark SQL、BigQuery 等進行預先處理和其他 ETL，它很好用）、scikit-learn 和 / 或採用 TensorFlow 後端的 Keras 寫成的。

如何使用本書

本書將我們從許多團隊觀察到的模式分門別類，有些案例的基本概念已經為人所知很多年了。我們不會宣稱自己發明或發現這些模式，與之相反，我們希望提供一個通用的參考框架和一組工具給 ML 實踐者使用。如果這本書可以為你和你的團隊提供一組詞彙，讓你們可以討論已經在 ML 專案中直覺地使用的概念，我們就認為我們已經成功了。

我們不期望你按順序閱讀這本書（不過你可以！），而且預計你會先瀏覽一下這本書，再比較深入地閱讀其中一些章節，並且引用這些概念來和同事溝通，當你遇到好像看過的問題時，再回頭參考這本書。如果你打算跳著看，我們建議你先看第 1 章和第 8 章，再研究各個模式。

每一個模式都會簡短地敘述問題、提供一個典型的解決方案、解釋為何該解決方案有效，以及關於權衡取捨和替代方案的多部分（many-part）探討。我們建議你在閱讀探討的部分時牢牢記住典型解決方案，以便進行比較和對比。我們在介紹模式時，會展示典型解決方案實作的部分程式。你可以在 GitHub repository 找到完整的程式碼（*https://github.com/GoogleCloudPlatform/ml-design-patterns*）。我們強烈建議你在閱讀模式敘述時仔細地閱讀程式碼。

機器學習的術語

因為今日的機器學習實踐者可能有不同的專業領域，包括軟體工程、資料分析、DevOps 或統計等，不同的實踐者使用術語的方式可能有細微的差異。在這一節，我們來定義貫穿全書的術語。

模型與框架

機器學習的核心，就是建立「可從資料中學習的模型」的程序，這一點與傳統的程式設計不一樣，在傳統的程式設計裡，我們會編寫明確的規則來告訴程式如何行動。機器學習**模型**是從資料中學習模式的演算法。舉例來說，假設我們是一家搬家公司，需要為潛在客戶估算搬家成本。在傳統的程式設計中，我們可能會用 if 陳述式來解決這個問題：

```
if num_bedrooms == 2 and num_bathrooms == 2:
  estimate = 1500
elif num_bedrooms == 3 and sq_ft > 2000:
  estimate = 2500
```

可想而之，當我們加入更多變數（大型家具的數量、衣物的數量、易碎物品等等），並嘗試處理邊緣案例時，這個問題會變得多麼複雜。更重要的是，提前詢問客戶所有這些資訊可能會導致他們放棄評估。與之相反，我們可以訓練一個機器學習模型，根據公司服務過的家庭的資料來估算搬家成本。

這本書在範例中主要使用前饋神經網路模型，但我們也會使用線性回歸模型、決策樹、分群模型等。**前饋神經網路**（通常簡稱為**神經網路**）是一種機器學習演算法，它有很多層，每一層都有很多神經元，負責分析和處理資訊，然後將資訊發送到下一層，最後一層會產生一個預測作為輸出。儘管神經網路與我們腦中的神經元不一樣，但是由於節點之間的連接情況，以及它們有從資料進行類推並且產生新預測的能力，兩者經常被拿來相比。有超過一個**隱藏層**的神經網路（除了輸入與輸出層之外的神經層）屬於**深度學習**（見圖 1-1）。

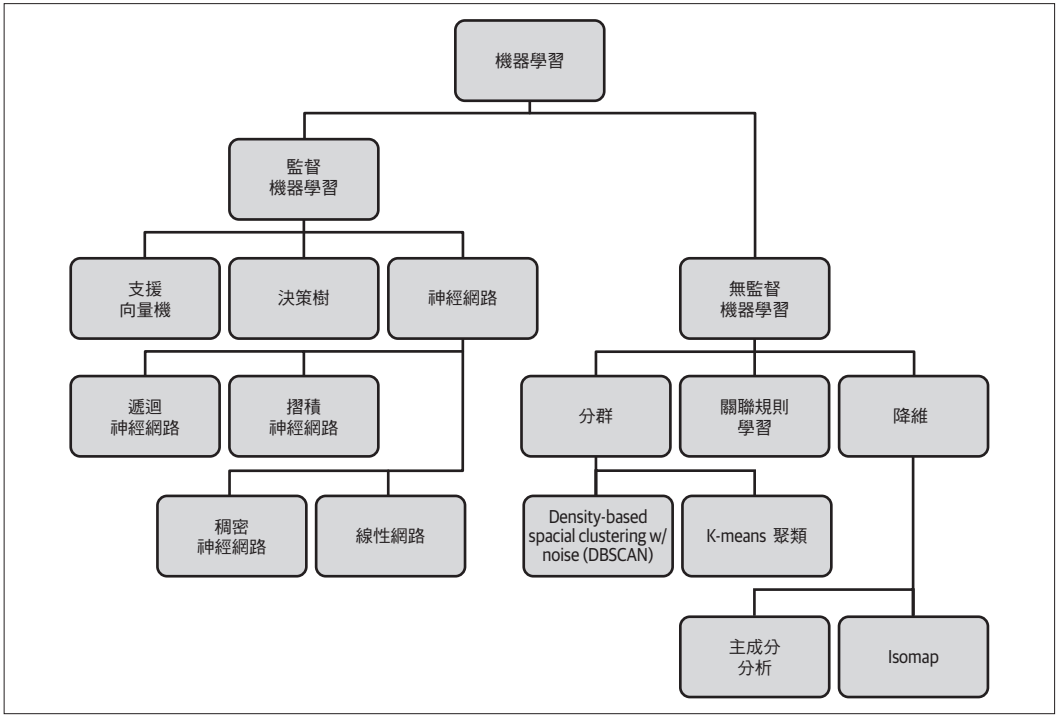

圖 1-1　各種機器學習類型，以及各種類型的例子。注意，雖然這張圖中沒有提及，但 autoencoder 這類的神經網路也可以用在無監督學習上。

不管機器學習模型在視覺上是如何呈現的，它們都是數學函數，因此可以用數值軟體包（numerical software package）從頭開始實現。然而，業界的 ML 工程師比較喜歡使用開源框架，這些框架旨在提供直觀的 API，方便大家建構模型。我們的大多數範例將使用 *TensorFlow*，它是 Google 製作的開源機器學習框架，並且把重心放在深度學習模型上。我們將在範例中使用 TensorFlow 程式庫的 *Keras* API，你可以用 `tensorflow.keras` 匯入它。Keras 是更高階的神經網路建構 API。Keras 支援很多後端，我們將使用它的 TensorFlow 後端。在其他範例中，我們將使用 scikit-learn、XGBoost 和 PyTorch，它們是其他的流行開源框架，提供了協助你準備資料的工具，以及用來建構線性模型和深度模型的 API。機器學習正持續變得更易用，有一種令人期待的發展是我們可以用 SQL 來表示機器學習模型，我們將以 BigQuery ML 為例，尤其是在我們想要結合資料的預先處理和模型的建立時。

只有一個輸入和輸出層的神經網路是機器學習的另一個子集合，稱為**線性模型**。線性模型以線性函數來表示它們從資料中學到的模式。決策樹是一種機器學習模型，它使用你的資料來製作帶有不同分支的路徑子集合。這些分支可以近似你的資料產生的各種結果。最後，聚類模型可尋找資料的不同子集合之間的相似性，並使用這些找出來的模式來將資料分組到聚類（cluster）裡。

機器學習問題（見圖 1-1）可以分成兩種類型：監督學習和無監督學習。監督學習處理的是你預先知道資料的基準真相標籤的問題。例如，這可能包括幫一張圖像標注「貓」標籤，或者幫一位嬰兒標注出生時 2.3 kg 的標籤。你會將這些標注好的資料傳給模型，希望它能夠學會足夠的知識來標注新的案例。在**無監督學習**中，你事前不知道資料的標籤，你的目標是建構一個可以找出資料的自然群組（稱為 *clustering*）、壓縮資訊內容（**降維**）或發現關聯規則的模型。本書的大部分內容將集中在監督學習上，因為在生產環境中的機器學習模型大都是監督模型。

採取監督學習的問題通常可以定義成分類或回歸問題。**分類**模型會幫你的輸入資料指定一個（或多個）標籤，這些標籤來自一組分立的、預先定義的類別。分類問題的例子包括確認照片中的寵物品種、標注文件，或預測一項交易是不是詐騙。**回歸**模型則是為你的輸入指定連續的數值。回歸模型的例子包括預測一趟自行車旅程的持續時間、公司未來的收入，或產品的價格。

資料與特徵工程

資料是任何機器學習問題的核心，當我們談到**資料組**時，我們指的是用來訓練、驗證與測試機器學習模型的資料。資料大部分都是**訓練資料**：在訓練程序中傳給模型的資料。**驗證資料**是從訓練組保留下來的資料，在每一個訓練 *epoch*（或跑過一遍訓練資料）之後，用來評估模型的表現。我們用模型處理驗證資料的表現來決定何時停止訓練回合，或選擇**超參數**，例如隨機森林模型裡的樹數量。**測試資料**完全不會在訓練程序中使用，其用途是評估訓練好的模型的表現如何。機器學習模型的性能必須用獨立的測試資料來評估，而不是訓練或驗證組。在分開資料時，讓全部的三個資料組（訓練、測試、驗證）都有相似的統計屬性也非常重要。

根據模型的類型，用來訓練模型的資料可能有許多種形式，我們將數值與分類資料稱為**結構性資料**。數值資料包括整數與浮點值，分類資料包括可分成有限組別的資料，例如汽車的型號或教育程度。你也可以將結構性資料想成通常可在試算表中找到的資料，在這本書裡，我們會交互使用**表格資料**和結構性資料。另一方面，**無結構資料**包括無法整潔地表示的資料，通常包含自由形式文本、圖像、視訊和音訊。

數值資料通常可以直接傳給機器學習模型，其他資料則需要先做各種**資料預先處理**，才能送給模型。這個預先處理步驟通常包含調整數值尺度，或將非數值資料轉換成模型可以理解的數值格式。預先處理的另一種說法是**特徵工程**。我們會在本書中交換使用兩種說法。

隨著資料經歷特徵工程程序的每一個階段，我們會用各種名稱來稱呼它。**輸入**是資料組被處理之前，在它裡面的一欄，**特徵**是它被處理**之後**，在它裡面的一欄。例如，時戳可能是輸入，特徵可能是星期幾。為了將資料從時戳轉換成星期幾，你必須做一些資料預先處理。這個預先處理步驟也可以稱為**資料轉換**。

實例就是你想要傳給模型來進行預測的項目。一個實例可能是測試資料組裡的一列（不含標籤欄）、你想要分類的圖像、或是你想要送給情緒分析模型的文本文件。模型收到實例的一組特徵之後會計算預測值，為了讓模型有計算能力，我們要用**訓練樣本**來訓練模型，訓練樣本的每個實例都有一個**標籤**。**訓練樣本**是傳給模型的資料組裡面的一個（列）資料實例。在時戳用例中，完整的訓練樣本可能包含「星期幾」、「縣市」和「汽車型號」。**標籤**是資料組裡面的輸出欄，也就是你的模型預測的項目。**標籤**可能指的是資料組裡內的目標欄（也稱為**基準真相標籤**），也可能是模型的輸出（也稱為**預測**）。上述的訓練案例中的樣本標籤可能是「行車期間」——在這個例子中，它是代表分鐘的浮點值。

當你整理好資料組並決定模型的特徵之後，**資料驗證**就是計算資料的統計數據、理解綱要（schema）、評估資料組以找出漂移（drift）和訓練 / 伺服傾斜（training-serving skew）等問題的程序。評估資料的各種統計數據可以幫助你確保資料組裡面的每個特徵都被平衡地表示。在無法收集更多資料的情況下，理解資料的平衡可以協助你設計出處理這種情況的模型。了解綱要的工作包括為每個特徵定義資料類型，以及找出值不正確或缺漏的訓練樣本。最後，資料驗證可以找出可能影響訓練和測試組品質的不一致狀況。例如，或許訓練資料組大多數的樣本都是**工作日**樣本，但測試組主要是**週末**的樣本。

機器學習程序

典型的機器學習程序的第一步就是**訓練**，也就是將訓練資料傳遞給模型，讓它能夠學習識別模式的過程。在訓練之後，下一步是檢驗模型處理訓練組之外的資料時的表現如何，這個步驟被稱為模型**評估**。你可能會進行多次訓練和評估，執行額外的特徵工程，以及調整模型的架構。當你滿意模型的表現之後，你可能希望讓模型提供服務，讓別人可以用它來進行預測。我們用**伺服**（*serving*）來代表將模型部署成微服務，來接受外來的請求，並且回傳預測。伺服基礎設施可能位於雲端、在內部部署或在設備上。

向模型傳遞新資料，並使用它的輸出稱為**預測**。預測可以代表用尚未部署的本地模型產生預測，也可以代表從已部署的模型取得預測。對已部署的模型而言，我們稱之為線上（online）和批次（batch）預測。以接近即時的方式取得樣本的預測結果稱為**線上預測**。線上預測強調低等待時間。另一方面，**批次預測**是以離線的方式為大量的資料產生預測。批次預測花費的時間比線上預測更久，適合用來預先計算預測（例如推薦系統），以及分析模型對大量新資料樣本所做的預測。

預測這個詞比較適合用來代表預測未來的值，例如預測騎自行車的時間，或預測會不會放棄購物車。在圖像和文本分類模型的案例中，預測比較不直覺。ML 模型觀察一則文字評論之後輸出正面情緒並不是真正的「預測」（因為它不是未來的結果）。因此，你也會看到有人用**推理**（*inference*）來代表預測。雖然這裡使用統計術語推理（inference），但它其實與「reasoning」無關。

通常，收集訓練資料、進行特徵工程、訓練和評估模型的流程是與生產管道分開處理的。若是如此，當你認為你有足夠的額外資料，可用來訓練新版本的模型時，你就要重新評估你的解決方案。若非如此，你可能要不斷接收新資料，並且要立刻處理那些資料，才能將它們發送給模型進行訓練或預測，這個情況稱為**串流**（*streaming*）。為了處理串流資料，你必須採取一個多步驟的解決方案，來執行特徵工程、訓練、評估與預測。這種多步驟的解決方案稱為 *ML* **處理管道**（*pipeline*）。

資料與模型工具

我們將使用各種 Google Cloud 產品，它們提供許多解決資料和機器學習問題的工具。這些產品只是實作本書介紹的設計模式的選擇之一，此外還有其他工具可用。本書使用的所有產品都是無伺服器的（serverless），這是為了讓我們更關注機器學習設計模式的實作，而不是它們背後的基礎設施。

BigQuery（*https://oreil.ly/7PnVj*）是企業資料倉庫，它的設計是為了使用 SQL 來快速分析大型的資料組。我們將在範例中使用 BigQuery 進行資料收集和特徵工程。在 BigQuery 裡面的資料會被做成 Dataset，一個 Dataset 可能有多個 Table。我們的許多範例都使用 Google Cloud Public Datasets（*https://oreil.ly/AbTaJ*）的資料，它是 BigQuery 代管的免費、公開的資料組。Google Cloud Public Datasets 包含上百個不同的資料組，包括自 1929 年以來的 NOAA 天氣資料，Stack Overflow 的問題和解答，GitHub 的開源程式碼，出生率資料，等等。我們將使用 BigQuery Machine Learning（*https://oreil.ly/_VjVz*）（或 BigQuery ML）來建立一些範例的模型。BigQuery ML 是使用 BigQuery 所儲存的資料來建構模型的工具。我們可以使用 BigQuery ML，以 SQL 對模型進行訓練、評估和產生預測。它支援分類和回歸模型，以及無監督聚類模型。我們也可以將訓練好的 TensorFlow 模型匯入 BigQuery ML 來進行預測。

Cloud AI Platform（*https://oreil.ly/90KLs*）包含各種可在 Google Cloud 上訓練和伺服自訂機器學習模型的產品。我們將在例子中使用 AI Platform Training 和 AI Platform Prediction。AI Platform Training 是讓你在 Google Cloud 上訓練機器學習模型的基礎設施。藉由 AI Platform Prediction，你可以使用 API 來部署訓練好的模型，並且用它們產生預測。這兩種服務都支援 TensorFlow、scikit-Learn 和 XGBoost 模型，以及用其他框架建構的模型的自訂容器。我們也會使用 Explainable AI（*https://oreil.ly/lDocn*），它是用來解釋模型預測結果的工具，可處理部署在 AI Platform 上面的模型。

角色

組織裡有許多與資料和機器學習有關的工作角色。接下來我們將定義一些本書經常提到的角色。本書主要針對資料科學家、資料工程師和 ML 工程師，所以我們從這些人開始談起。

資料科學家（*data scientist*）專注於收集、解釋和處理資料組。他們對資料進行統計和探索性分析。由於這些工作與機器學習有關，資料科學家可能也會從事資料收集、特徵工程、模型建構等工作。資料科學家經常在 notebook 環境中使用 Python 或 R 來工作，通常是在組織裡第一個做出機器學習模型的人。

資料工程師（*data engineer*）專門處理組織資料的基礎設施和工作流程。他們可能協助管理公司如何接收資料、資料處理管道，以及如何儲存和傳輸資料。資料工程師負責製作資料相關的基礎設施和處理管道。

機器學習工程師的工作與資料工程師類似，但是他們處理的是 ML 模型。他們接收資料科學家開發的模型，管理圍繞著模型的訓練和部署的基礎設施和操作。ML 工程師協助建構生產系統來處理模型的更新、模型的版本控制，和提供給終端用戶的預測。

公司的資料科學團隊越小，團隊就越敏捷，就越有可能讓同一個人扮演多個角色。如果你屬於這種情況，當你看完上面的三個角色之後，很有可能會認為自己屬於這三個角色。你可能通常以資料工程師的身分開始進行一項機器學習專案，並建構資料處理管道來將資料的接收作業化。然後，你會轉換成資料科學家角色，開始建構 ML 模型。最後，你會戴上 ML 工程師的帽子，將模型投入生產。在較大組織裡的機器學習專案可能有相同的階段，但是每一個階段可能會由不同的團隊負責。

雖然研究科學家、資料分析師和開發者也可能建構和使用 AI 模型，但本書的重點讀者不是這些工作角色。

研究科學家主要負責發現和開發新的演算法，以推進 ML 學科。ML 學科可能包含機器學習的各種子領域，例如模型架構、自然語言處理、計算機視覺、超參數調整、模型可解釋性等等。與這裡討論的其他角色不同的是，研究科學家的時間大部分都用來建構雛型和評估 ML 的新方法，不是建構 ML 生產系統。

資料分析師會評估資料並取得見解，然後整理這些見解，讓組織的其他團隊使用。他們往往使用 SQL 和試算表來工作，並使用商務智慧工具來將資料視覺化，來分享他們的發現。資料分析師與產品團隊密切合作，以了解他們的見解如何協助解決商務問題和創造價值。資料分析師的重點是找出既有資料的趨勢，並從中獲得見解，但資料科學家的重點是使用這些資料來產生未來的預測，以及將產生見解的過程自動化或擴大其範圍。隨著機器學習日益民主化，資料分析師可以提升自己的技術，成為資料科學家。

開發者負責建構可讓終端用戶使用 ML 模型的生產系統。他們經常參與特定 API 的設計，那些 API 可讓用戶透過 web 或行動 app 以友善的格式查詢模型並回傳預測。這項工作涉及在雲端托管的模型，或在設備上提供服務的模型。開發者會利用 ML 工程師製作的模型伺服基礎設施來建構 app 與用戶介紹，以顯示預測給模型用戶看。

圖 1-2 是這些角色在一個組織的機器學習模型開發過程中合作的情況。

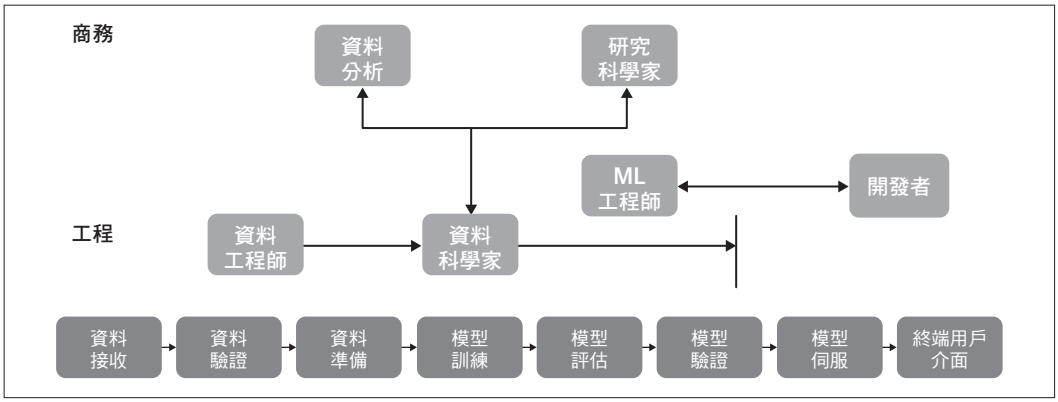

圖 1-2　與機器學習有關的工作角色與資料有很多，這些角色會在 ML 工作流程中合作，從資料接收，到模型伺服與終端用戶介面。例如，資料工程師負責資料接收和資料驗證，並與資料科學家密切合作。

機器學習常見的挑戰

為什麼我們需要一本關於機器學習設計模式的書？在建構 ML 系統的過程中，你會遇到各種影響 ML 設計的獨特挑戰，身為一位 ML 實踐者，了解這些挑戰可協助你為本書介紹的解決方案建立一個參考框架。

資料品質

機器學習模型的可靠性取決於訓練它們的資料。如果你用不完整的資料組、用選擇不當特徵的資料，或不能準確代表模型使用者的資料來訓練機器學習模型，那些資料將會直接反映在模型的預測上。因此，機器學習模型通常被稱為「garbage in, garbage out」。在此，我們提出資料品質的四個重要成分：準確性、完整性、一致性和及時性。

資料準確性包含訓練資料的特徵以及這些特徵的基準真相標籤。了解資料的來源和資料收集過程的潛在錯誤可以幫助你確保特徵的準確性。當你收集資料之後，你一定要進行徹底的分析，以找出打字錯誤、重複的項目、表格資料中的數值不一致、缺漏的特徵，以及任何其他可能影響資料品質的錯誤。例如，訓練資料組裡面的重複會導致模型為那些資料點錯誤地分配更多的權重。

準確的資料標籤和準確的特徵一樣重要，模型只會用訓練資料裡的基準真相標籤來更新它的權重以及將損失最小化，因此，訓練案例有不正確的標籤會產生具誤導性的模型準確度。例如，假設你要建立一個情緒分析模型，而且有 25% 的「正面」訓練案例被錯誤地標注為「負面」，你的模型會錯誤地認為哪些情況應視為負面情緒，並且直接反映在它的預測中。

說到資料**完整性**，假設你正在訓練一個模型來識別貓的品種。你用一個龐大的貓照片資料組來訓練這個模型，做出來的模型能夠以 99% 的準確率將照片分成 10 種分類（「孟加拉」、「暹羅」等等）之中的一個。然而，當你將模型部署到生產環境中時，你會發現用戶除了上傳貓的照片來分類之外，許多用戶也會上傳狗的照片，並且對模型的結果感到失望。因為這個模型只學會辨識 10 種不同的貓品種，這就是它所知道的一切。這 10 個品種基本上就是該模型的整個「世界觀」。無論你送給模型什麼，你都可以想像它會將它歸入這 10 個分類中的一個，它甚至可以對一個看起來一點都不像貓的照片充滿信心地這樣做。此外，如果訓練資料沒有「不是貓」的資料和標籤，你的模型就不可能回傳「不是貓」。

資料完整性的另一個層面是確保訓練資料含有每一種標籤的各種呈現方式。在貓品種檢測例子中，如果所有的照片都是貓臉的特寫（close-up），模型就無法正確地辨識貓的側面照片或全身照片。以表格資料為例，如果你要建構模型來預測特定城市的房地產價格，但訓練樣本只有面積超過 2,000 平方英尺的房屋，那麼你的模型就無法正確地處理較小的房屋。

資料品質的第三個層面是資料**一致性**。在處理大型的資料組時，大家通常將資料收集和標注工作交給同一組人員處理。為這個程序設計一套標準可確保整個資料組的一致性，因為參與其中的人員都難免在這個過程中引入他們自己的偏見。如同資料完整性，資料不一致性可以從資料特徵與標籤中發現。舉個不一致特徵的例子，假設你用溫度感測器來收集大氣資料。如果每一個感測器都是用不同的標準來校正的，它們會導致不準確和不可靠的模型預測。不一致也可能是指資料格式。如果你要描述位置資料，有些人可能會將完整的街道地址寫成「Main street」，而另一些人可能將它縮寫為「Main St.」。測量單位，如英里和公里，在世界各地也可能不同。

關於標籤不一致，我們回到文本情緒案例。在這個案例中，當人們標注訓練資料時，不一定會一致同意什麼是正面的，什麼是負面的。為了解決這個問題，你可以讓多位人員標注資料組的每一個案例，然後讓每一個項目都使用最多人選擇的標籤。注意標注人的偏見，並設計一個系統來應對它，可確保整個資料組的標籤維持一致。我們將在第 7 章，第 333 頁的「設計模式 30：Fairness Lens」探討偏見的概念。

資料的**即時性**指的是從事件發生到它被加入你的資料庫之間的時間。舉例來說，如果你在收集應用程式紀錄（log）裡的資料，一個錯誤紀錄可能需要幾個小時才會出現在紀錄資料庫裡。對記錄信用卡交易的資料組而言，從交易發生到它被回報至你的系統可能要花一天的時間。為了處理即時性，你可以盡量記錄特定資料點的資訊，並且在將資料轉換成機器學習模型的特徵時，確保這些資訊能夠反映出來。更具體地說，你可以追蹤事件何時發生以及它何時被加到你的資料組的時戳。然後，在進行特徵工程時，你可以相應地考慮這些差異。

再現性

在傳統的程式設計中，程式的輸出是可再現且有保證的。例如，如果你寫了一個將字串倒寫的 Python 程式，你可以確定輸入單字「banana」一定會輸出「ananab」。類似地，如果程式有 bug，導致包含數字的字串被錯誤地倒寫，你可以將這段程式送給同事，期望他們能夠用相同輸入重現錯誤（除非這個 bug 與持有不正確的內部狀態的程式、架構上的差異（例如浮點精度）或執行上的差異（例如執行緒）有關）。

另一方面，機器學習模型有內在的隨機元素。在訓練時，ML 模型的權重的初始值會被設成隨機的值。在訓練過程中，當模型反覆從資料中學習時，這些權重值會收斂。因此，用相同的資料來訓練相同的模型程式會在不同的訓練回合中產生稍微不同的結果，這就帶來了再現性的挑戰。雖然你將一個模型訓練到 98.1% 的準確率，但是重複地進行訓練不保證能達到相同的結果，所以我們很難在不同回合的實驗之間進行比較。

為了解決再現性的問題，我們通常設定一個隨機種子值讓模型使用，以確保每次訓練時，都採用相同的隨機值。在 TensorFlow 裡，你可以在程式的開始處執行 `tf.random.set_seed(value)` 來做這件事。

此外，在 scikit-learn 裡，許多可以洗亂資料的函式都可以設定隨機的種子值：

```
from sklearn.utils import shuffle
data = shuffle(data, random_state=value)
```

請記住，在訓練模型時，你必須使用相同的資料和相同的隨機種子，以確保在不同的實驗中得到可重複的、可再現的結果。

在訓練 ML 模型的過程中，你必須固定幾個元素才能確保再現性：所使用的資料、用來產生訓練和驗證資料組的拆分機制、資料準備和模型超參數，以及批次大小、學習率排程等變數。

再現性也適用於機器學習框架的依賴項目。除了手動設置隨機種子之外，框架也在內部實作了一些隨機元素，會在你呼叫函式來訓練模型時執行。如果這個底層的實作在不同的框架版本之間發生變化，我們就無法保證可重複性。舉個具體的例子，如果某個框架版本的 `train()` 方法呼叫 13 次 `rand()`，同一個框架的新版本呼叫 14 次，在不同的實驗使用不同的版本會造成稍微不同的結果，即使使用相同的資料與模型程式碼。在容器裡運行 ML 工作負載，並且使用標準化的程式庫版本，有助於確保可重複性。第 6 章會介紹一系列讓 ML 程序可以再現的模式。

最後，再現性可能是指模型的訓練環境。通常由於大型的資料組和複雜性，許多模型都需要用大量的時間來訓練。我們可以藉著使用資料或模型平行化之類的分散式策略來加速（見第 5 章）。然而，這種加速可能會在你重新執行這些利用分散式策略來訓練的程式碼時額外遇到可重複性問題。

資料漂移

雖然機器學習模型通常代表輸入與輸出之間的靜態關係，但資料可能會隨著時間而大幅改變。資料漂移是當你想要確保機器學習模型可以與時俱進，並且讓模型的預測可以準確地反映它們所處的環境時必須克服的挑戰。

例如，假設你要訓練一個模型來將新聞標題分類為「政治」、「商業」和「技術」等類別。如果你用 20 世紀的歷史新聞文章來訓練和評估模型，它可能無法很好地處理現在的資料。在現代，我們知道標題有「智慧型手機」的文章很可能與科技有關，然而，用歷史資料訓練的模型不認識這個詞。為了解決漂移問題，我們必須不斷更新訓練資料組、重新訓練模型，以及修改模型分配給特定輸入資料群組的權重。

我們來看一個沒那麼明顯的例子，它是在 BigQuery 裡的 NOAA 強烈風暴資料組（*https://oreil.ly/obzvn*）。如果我們訓練一個模型來預測特定地區發生風暴的可能性，我們就要考慮天氣報告隨著時間的過去的變化（*https://github.com/GoogleCloudPlatform/ml-designpatterns/ blob/master/01_need_for_design_patterns/ml_challenges.ipynb*）。從圖 1-3 可以看出，自 1950 年以來，有紀錄的強烈風暴的總數一直在穩步增加。

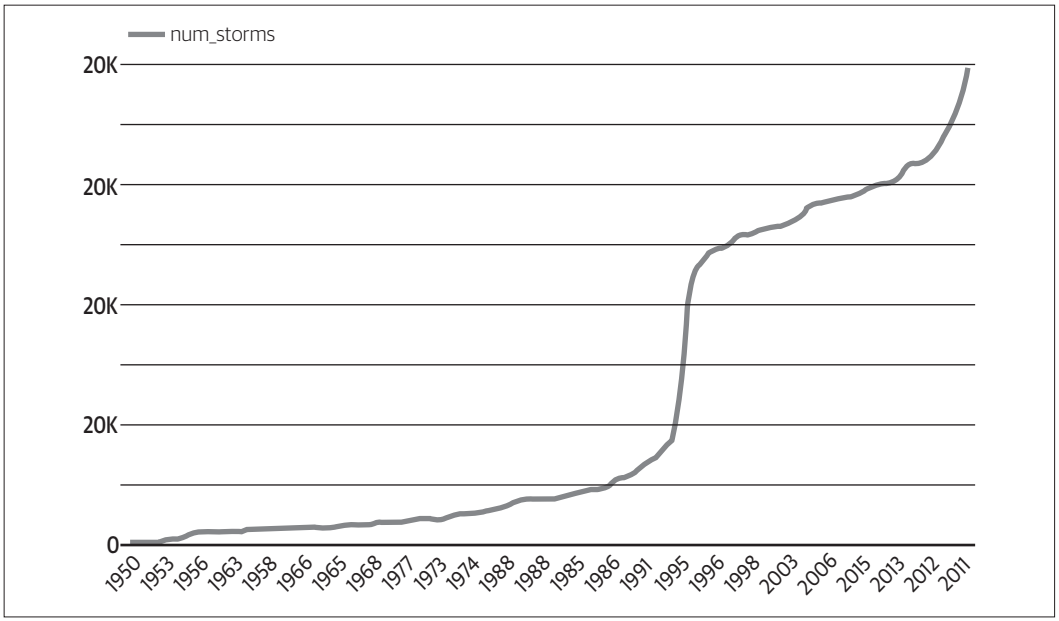

圖 1-3　逐年的強烈風暴數量，來自 NOAA 從 1950 至 2011 記錄的資料。

從這個趨勢可以看出，用 2000 年之前的資料訓練模型，並用它來預測今日的風暴將產生不準確的結果。除了被回報的風暴總數增加之外，我們也必須考慮可能影響圖 1-3 的其他因素。例如，觀測風暴的技術會隨著時間而改善，最明顯的是自 1990 年起開始使用的氣象雷達。在特徵方面，這可能意味著較新的資料包含了每個風暴的更多資訊，而今日資料中可用的特徵在 1950 年可能還沒有被觀測到。探索性資料分析可以協助辨識這類的漂移，而且可以讓你知道該用哪段時間的資料來訓練才對。如果資料組的特徵會隨著時間的推移而改善，第 258 頁的「設計模式 23：Bridged Schema」提供一種處理這種資料組的方法。

擴展

典型的機器學習工作流程的許多階段都有擴展方面的挑戰。你可能會在資料收集與預先處理、訓練和伺服方面遇到關於擴展的挑戰。當你為機器學習模型接收和準備資料時，資料組的大小將決定解決方案所需的工具。資料工程師的工作通常是建構能夠擴展為可處理數百萬行資料組的資料處理管道。

在訓練模型時，ML 工程師要負責為特定的訓練工作決定必要的基礎設施。取決於資料組的類型和大小，模型的訓練可能會非常耗時、計算成本很高，需要使用專門為 ML 工作而設計的基礎設施（例如 GPU）。例如，訓練圖像模型所使用的基礎設施通常比完全用表格資料來訓練模型更多。

在模型伺服的背景下，支援一組資料科學家從模型雛型獲得預測所需的基礎設施和支援每小時要處理數百萬個預測請求的生產模型所需的基礎設施完全不一樣。開發者和 ML 工程師通常負責處理與模型部署相關的擴展挑戰和伺服預測的請求。

如果不考慮組織的成熟度，本書中的大多數 ML 模式都是有用的。然而，第 6 章和第 7 章的一些模式會用不同的方法來處理復原力（resilience）和再現性方面的挑戰，在它們之間的選擇通常取決於具體用例，和你的組織消化複雜性的能力。

多個目標

雖然負責建構機器學習模型的團隊通常只有一個，但是在同一個組織裡面的不同團隊可能會分別以某種方式來使用模型，這些團隊難免對於成功的模型該如何定義有不同的想法。

為了理解這種情況在實務中如何發生的，假設你正在建構一個從照片中認出瑕疵品的模型。身為資料科學家，你的目標可能是盡量降低模型的交叉熵損失。另一方面，產品經理可能希望減少被錯誤分類並發送給客戶的瑕疵品的數量。最後，管理團隊的目標可能是增加 30% 的收入。優化每一個目標的原因都不一樣，在組織內平衡這些需求可能是一個挑戰。

身為資料科學家，你可以將產品團隊的需求轉換到模型中，例如設定偽陰性的代價是偽陽性的 5 倍。因此，在設計模型時，你應該優化 recall 而不是 precision 來滿足這一個需求，如此一來，你就可以在產品團隊的優化目標和你的目標（將模型的損失最小化）之間找到平衡點。

當你為模型定義目標時，你一定要考慮組織裡的每個團隊的需求，以及每個團隊的需求與模型有什麼關係。在制定解決方案之前，分析每個團隊正在優化什麼可以協助你找到妥協的領域，讓你可以讓這些目標有良好的平衡。

小結

設計模式是匯整專家的知識和經驗來讓所有實踐者都可以遵循的方法。本書的設計模式描述在設計、建構和部署機器學習系統時常見的問題的最佳實踐法和解決方案。機器學習常見的挑戰往往與資料品質、再現性、資料漂移、擴展和滿足多個目標的需求有關。

我們往往在 ML 生命週期的不同階段使用不同的 ML 設計模式。有一些模式在建構問題框架和評估可行性時很好用。大多數的模式處理的是開發或部署層面，此外也有相當多的模式處理這些階段之間的相互作用。