對本書的讚譽

從長遠來看，絕大多數人工智慧將不會只在資料中心使用，而是在資料真正的所在之處使用，即數十億個真實世界的設備上。由於 Swift 結合了速度和表現力，已經使它成為實現這一轉變的首選語言。

這本書介紹了一個美好的工具，這個工具將成為下一個十年的人工智慧的基礎。

—*Dr. Jameson Toole, Cofounder and CTO of Fritz AI*

香蕉和蘋果對機器學習開發者的重要性怎麼強調都不過分。幸運的是，這本書既有香蕉也有蘋果，或至少能讓人繼續深入研究下去。所以，如果您像我一樣，猜測機器學習的未來是在設備上，那麼這本書是一個很好的起點。

—*Dr. Alasdair Allan, Babilim Light Industries*

在過去的幾年裡，人工智慧已經從學術界和科幻小說變成了解決現實問題的實用技術。這本書展示了如何把人工智慧帶到您的手上，在您手上實現 ML 功能現在已成為可能。我非常興奮地看到讀者們如何利用他們從這本書中學到的東西。

—*Chris Devers, Technical Lead of Sustaining Engineering, EditShare*

前言

歡迎閱讀本書。

這本書採取了一個以任務為導向的說明方法，使用 Swift 實際執行人工智慧（AI）。我們這樣做是因為我們認為，您不應該為了在您的 iOS 應用程式中擁有聰明的人工智慧和機器學習的功能，而深入研究複雜的數學和演算法。人工智慧不應該是只屬於人工智慧專家的專業領域。對非專家來說，也應該可以掌握人工智慧。

我們生活在一個這些技術變得越來越普遍的世界，它們正在成為我們與電腦互動結構的一部分。由於機器學習的不可思議的力量，以及它衝擊、受益、影響和控制人類的能力，人們要能夠掌握如何建立和理解人工智慧的知識，就像知道如何操作電腦一樣重要。

為此，這本書的目的是讓您對常見的機器學習任務有一個實際的理解。有了這些，您將能夠建立更好的工具，並瞭解更廣泛的世界中其他人所用工具的行為。我們很高興您閱讀了本書，讓我們一起建造人工智慧。

本書資源

我們建議在閱讀本書時，在閱讀每一章的過程中自己撰寫程式碼。如果您卡住了，或者只是想要一份程式碼的副本，您可以到我們的網站（*http://www.aiwithswift.com*）找到您需要的東西。

適用讀者和閱讀方法

這本書是為那些已經知道如何用 Swift 做程式設計，並希望瞭解有關機器學習威力的具體功能和技術的人寫的。這不是一本教您程式設計基礎知識的書，也就是說，使用這些範例需要對程式設計語言有一定的熟悉程度。

除了使用 Swift 之外，我們偶爾還會講到 Python。Python 在機器學習和人工智慧領域非常普遍。這本書的主要內容是 Swift，所以我們將會解釋為什麼我們需要使用 Python。關於這一點，我們會提醒您們幾次。

最後，我們還假設，作為使用者，您對 macOS 和 iOS 的操作相當熟悉，您可以使用 Mac 進行開發，也可以使用 iOS 設備進行測試，完成需要特定感測器的任務，比如運動追蹤或影像分析。

本書組織

這本書分為三部分。

第一部分是**基礎與工具**，我們將介紹機器學習和人工智慧背後的基本概念。我們為您設定了將要使用的語言和工具，並讓您準備好可開始建立有用的東西。

第二部分是**任務**，我們廣泛地從電腦視覺、音訊、動作和語言領域中，取出一些有趣的主題。對於每個主題，我們將展示並建立一個 App，用它來突顯各種技術和 API；在每一章的最後，您將有一個完整可用的示範 App，用來展示您的程式碼可以完成的實際任務。

第三部分是**進階**，我們看一看幕後到底做了什麼，並詳細查看那些提升 App 的技術。我們將仔細研究機器學習的理論，並讓您對設備實際在做什麼有更深層的瞭解。

使用這本書

我們希望這本書能深深扎根於「**實用**」。正因為如此，我們的內容圍繞著建立實際**任務**，這些任務可能是您在處理人工智慧和 Swift 時，可能會想要執行的任務。我們把這本書分成了三部分。

第一部分討論 Swift 和人工智慧，以及本書如何採用任務導向介紹人工智慧的方法，以及用於 Swift 和人工智慧的工具（第 2 章），還有在您的實際人工智慧應用中，如何找到資料集合，以及為什麼要使用該資料集合（這是本部分的第 3 章）。

第二部分的每一章，都探討了可以集成到您 Swift App 的不同領域的人工智慧，以分類主題的角度來看：視覺相關任務（第 4 章），音訊任務（第 5 章），語言和文字的任務（第 6 章），運動和手勢的任務（第 7 章），生成和推薦這類增益任務（第 8 章），進階（第 9 章）中將探索我們在第一部分結束所接觸到的更進階的工具和 framework。在下一節中，我們將概述每個任務的內容。

第三部分探討了人工智慧**方法**實際上是如何工作的（第 10 章），以及我們在第二部分中所做的實際任務是如何工作的（第 11 章），以及最後艱難的道路（第 12 章），是用一個簡單版本說明您如何自己實作這些事情。

我們的任務

在這本書中，我們涵蓋了以下領域的任務：

- **視覺**
  - 人臉偵測
  - 條碼偵測
  - 重點偵測
  - 圖像相似度
  - 圖像分類
  - 繪圖識別
  - 風格分類
- **音訊**
  - 語音辨識
  - 聲音分類
- **文字和語言**
  - 語言識別
  - 命名實體識別
  - 詞元還原、標籤、拆分

— 情感分析
— 自訂文字分類器

- **運動和手勢**
  — 活動識別
  — 繪圖手勢分類
  — 活動分類
  — 將人工智慧用於擴增實境
- **增益**
  — 圖像風格轉換
  — 句子生成
  — 圖像生成
  — 電影推薦
  — 回歸
- **進階**
  — 安裝 Swift for TensorFlow
  — 使用 Python with Swift
  — 使用 Swift for TensorFlow 訓練分類器
  — 使用 CoreML Community Tools
  — 在設備上的模型更新
  — 在設備上下載模型

這本書的大部分內容都致力於探索，如何在您的應用程式中實作由人工智慧驅動的東西，而且（主要是）使用 Swift。因為我們採用自上向下的、以任務導向的人工智慧視角，所以我們決定在第 2 章先討論您可能會用到的工具。

在第 2 章中，我們將會探討一些工具，您可以用 Swift 建立機器學習和人工智慧任務的模型，以及一些用於操縱模型、處理資料和通常用 Swift 實踐人工智慧的工具。

第一章

人工智慧！?

許多關於人工智慧（AI）、機器學習和深度學習的文獻充斥著看起來複雜可怕的數學和晦澀的學術術語。如果您是一個相對精實的軟體工程師，想要涉足人工智慧的世界，並在應用程式中實作一些由人工智慧所驅動的功能，在一頭栽進去的時候會覺得有點可怕。

別擔心，現在的 AI 很簡單（好吧，是**比較簡單**），我們在這裡向您展示如何在您的 Swift 應用程式中**實作**它。

這本書所介紹關於實作人工智慧的概念和流程，均會用 Swift 以各式各樣的**實用的**方式呈現。

在開始之前，我們需要做一些基礎工作：首先，這本書講的是 AI 和 *Swift*（希望您已經在封面上看到了這些既大又友好的文字）。Swift 是一種令人驚歎的程式設計語言，它是構建 iOS、macOS、tvOS、watchOS、甚至 web 應用程式所需要的全部工具。

儘管很長一段時間以來，人工智慧世界一直與另一種語言——Python——保持著一種共生關係，但情節出現了轉折。因此，儘管這本書**著重**使用 Swift，但它並不是我們在書中使用的唯一程式設計語言。我們想在一開始就點明這件事：有時候我們確實會使用 Python。

在進行機器學習（machine learning）和人工智慧時，不可避免會使用到 Python。但這本書的主體使用的是 Swift，當我們改為使用 Python 時會解釋為什麼。關於這件事，之後我們還會提醒您們幾次。

我們需要預先確定的是（正如我們在前言中所說的），我們希望您已經知道如何寫程式。

這本書不是寫給才剛進入 Swift 程式設計與剛學習做人工智慧的程式設計師看的，我們設計這本書是為已懂得程式設計的人看的。您不需要成為一個專業的程式設計師，但是您需要知道發生了什麼。

儘管這本書的名字是《人工智慧開發實務｜使用 *Swift*》，但如果您不理解您正在研究的東西底下的一些理論基礎，就不可能創造出有效的人工智慧功能。這就代表著我們有時會研究一下理論的部分。這麼做是有用的，我們保證，而且這些內容大部分只會出現在書的第三部分。不過，建議您先讀完第二部分，才能獲得實際實作的具體細節。

實用的 AI 和 Swift…和 Python？

正如標題所述，我們確實在本書中使用 Swift 執行**大部分**的工作。但 Swift 並不是做機器學習的唯一方式。Python 在機器學習社群中非常流行，用於訓練模型。Lua 也得到了大量的使用。總的來說，您選擇的機器學習工具將會決定您要選擇哪種語言。CoreML 和 CreateML 是本書執行機器學習的主要工具，它們都假定使用 Swift。

我們在這裡假設讀者已熟悉 Swift 程式設計和建立基本的 iOS 應用程式。由於人工智慧和機器學習是一個如此大的話題，所以若我們也把 Swift 和 iOS 納入內容，這本書將會太大本，以致於在您的書架上看起來會很可笑。當我們讀這本書的時候，我們仍然會解釋每一步都發生了什麼，所以不要擔心我們會把您推入深淵。

如果您還在探索著怎麼使用 Swift 和 iOS，覺得需要複習一下，或者只是想再讀一本 Swift 的書，那就看看 *Learning Swift*（*http://oreil.ly/DLVGh*）吧。在這本書裡，我們從零開始，一路到建立出一個完整的應用程式。

雖然我們很偏心，但我們認為這是一本好書。但請不要我講了您就信，請讀一下就知道了。

本書從實作的角度出發，代表著我們試圖在所有事情上都是實際的（和實用的），包括語言的選擇。一項工作若適合由 Python 完成，我們也將向您展示如何用 Python 完成。

務實這件事，在本書的內容中的意思是：我們會動手做一些由 AI 驅動的功能，我們不會去管該功能如何工作，為什麼要做，或者是什麼使它有用。我們只關心怎麼使它工作。關於人工智慧、機器學習、深度學習理論以及事物如何運作的書籍已經足夠多了；我們不需要為您再寫一本。

程式碼範例

您可能知道，Swift 常會使用一些非常長的方法呼叫。在您超級高解析度的螢幕上有大量的像素，提供不可思議的水平捲動能力，所以您很少需要將方法呼叫和變數宣告拆成多行。

遺憾的是，儘管紙（和它的數位親戚）是一項令人印象深刻的技術，但它在水平空間方面有一些相當嚴格的限制。因此，我們經常將範例程式碼分成多行。

特別聲明，這不代表您的程式碼也需要分成多行；這只是我們為了要讓範例程式碼能配合您手上的這本書的印刷，所以才這麼做的。所以，如果您覺得一行程式碼看起來有點奇怪，您想知道為什麼我們把它分成很多行，這就是為什麼。

在這本書中，我們不打算討論程式碼風格。它是一個值得用另一本書（比如 Pragmatic 出版社的 *Swift Style* 第二版（*http://bit.ly/33ub91o*））介紹的大主題，而且它是主觀的，是一個很容易引發爭論的主題，所以我們就跳過這個主題了。每個人都有適合自己的方法，所以堅持下去，您會比一味地模仿別人更好。

做自己！

如果您不想輸入書本程式碼，我們已經在 GitHub（*http://bit.ly/2IZjW3o*）上建立了一個包含所有程式碼範例的儲存庫供您隨時使用。此外，在我們的網站上還放了額外的資源（*https://aiwithswift.com*）。

範例是用 Swift 5.x 撰寫的，螢幕截圖是擷取 macOS Catalina 上的 Xcode 11 家族的畫面。在更舊或更新的版本中，範例的行為可能不太相同。不過，Swift 現在已經穩定了，所以即使 Xcode 的使用者介面看起來有點不同，也應該能正常執行。

每個範例應該在相當長的一段時間內都能正常執行，即使是在這本書的出版日期一陣子之後也一樣。

我們在程式碼範例中也廣泛使用了 Swift 擴展（extension）。Swift 擴展允許您添加新功能到現有類別和枚舉中，看個例子比說明容易理解。

如果您是一個頭髮花白的 Objective-C 程式設計師，您可能記得 category 這種東西。擴展有點像 category，只差在擴展不會有名稱。

假設我們有一個叫做 `Spaceship` 類別：

```
class Spaceship
{
    var fuelReserves: Double = 0

    func setFuel(gallons: Double) {
        fuelReserves = gallons
    }
}
```

希望這段程式碼本身能表達它想做什麼，但為了確保我們有相同的理解，我們還是要解釋它：它定義了一個名為 `Spaceship` 的類別，這個類別有一個變數名為 `fuelReserves`（它的類型是 `Double`，但這現在不重要）。

`Spaceship` 類還有一個叫做 `setFuel()` 的函式，它有一個 `Double` 型態名為 `gallons`（加侖）的參數（我們不確定加侖是什麼，因為我們是澳大利亞人，但是我們的編輯告訴我們要這樣定）。該函式將 `fuelReserves`（燃料儲量）設定為 `gallons`，講完了。

假設我們想在程式碼的其他地方使用 `Spaceship` 類別，但是又想添加一個能讓我們以友善方式印出燃料儲量的函式。我們可以使用一個擴展：

```
extension Spaceship
{
    func printFuel() {
        print("There are \(fuelReserves) gallons of fuel left.")
    }
}
```

現在您可以呼叫 `Spaceship` 類別的一個實例的 `printFuel()` 函式，就好像它原本就寫在類別定義中一樣：

```
let starbug = Spaceship()
starbug.setFuel(gallons: 100.00)
starbug.printFuel()
```

我們也可以使用擴展來滿足協定（protocol）：

```
extension Spaceship: StarshipProtocol {
    // 在這裡寫 StarshipProtocol 要求的實作
}
```

在這個程式碼範例中，我們要求 `Spaceship` 類別要符合 `StarshipProtocol` 協定。由於我們會經常用到擴展，所以我們想確保這個範例能讓您理解擴展。

如果您需要更多關於擴展的資訊，請查看 Swift 文件（*http://bit.ly/328ovjy*）。同樣地，如果需要更多關於協定的資訊，也請查看文件（*http://bit.ly/32ajAi5*）。

為什麼要用 Swift ？

當我們談論人工智慧和 Swift 時，我們經常遇到會這樣說的人：「您是在講 Python 嗎？」（有時他們並沒有真的說出來，但您可以從他們的眼睛裡看出他們在想什麼）。

我們也常被人這樣問：「人工智慧？您是說機器學習嗎？」我們經常交替使用這兩個術語，因為時至今日用哪個詞已經不重要了，我們也不想加入這個爭論。您可以隨便叫它什麼；我們都不反對，我們稍後也會再談到這個討論。

以下是我們為什麼喜歡用 Swift 做人工智慧的原因。

Swift 是一個由蘋果公司發起的開源系統和資料科學程式設計語言專案，旨在建立一種強大、簡單、易於學習的語言，將學術界和產業界的最佳實踐整合到一種多範式、安全、可互動操作的語言中，這種語言適用撰寫從低階的程式設計到高階腳本間的任何東西，它在不同面向都取得了成功。

Swift 是一種受歡迎的、蓬勃發展的熱門語言，目前估計有 100 多萬位程式設計師使用它。有著多樣化和充滿活力的社群、成熟的工具。Swift 在蘋果的 iOS、macOS、tvOS 和 watchOS 的開發中佔據主導地位，但在伺服器和雲計算方面正在努力中。

Swift 有很多很棒的特質，若是想學習如何開發軟體，或想做現實世界的各種開發，Swift 是一個很好的選擇，這些特質包括：

設計

在早期，Swift 更新的快速又鬆散，也進行了許多語言上的修改。因此，Swift 現在是穩定的、優雅的、語法相當穩定的語言。Swift 也屬於 C 語言衍生家族的一員。

教學性

Swift 大部分的學習曲線很平緩，它遵循著逐步揭露複雜性和高局部性推理的設計目標。這代表著它很容易教，也很容易學。

高生產力

Swift 程式碼清晰、易於撰寫，並且樣板很小。它被設計成易於撰寫，更重要的是易於閱讀和除錯。

安全

Swift 天生就是安全的，該語言的設計使其很難造成記憶體安全問題，並且很容易找到和糾正邏輯錯誤。

性能

Swift 速度快，記憶體用量合理。這一點可能是因為它一開始是一種移動程式設計語言（mobile programming language）。

發佈 / 執行

Swift 會被編譯成原生機器碼，不會有惱人的垃圾收集器或肥胖的執行時期函式庫。一個機器學習模型可以被編譯一個目的檔案和一個 header 檔。

除了所有這些很棒的特質，從 2018 年年中開始，語言團隊宣佈提升語言穩定性：穩定性在語言誕生的早期，這是一個相當重要的因素。穩定代表著重構較少，每個參與者也比較不用重新熟悉語言，這是一件好事。

我們的學習途徑

除了蘋果平台語言 Swift 的歷史之外，這本書介紹 Swift 時還是以概念為先，其次才是平台。平台也會被好好介紹一番，但我們想要讓本書中列出的方法與平台無關。

Swift 是一種使用者友好的語言，若與其他人工智慧應用程式常會使用的語言相比，它很容易學習。我們希望提供的教學可幫助讀者學習發展 AI 的一般技能，甚至超越語言的使用。

我們的重點在完成任務和實作功能，而不是去討論人工智慧的本質。

為什麼要用人工智慧 ？

儘管許多非 Swift 開發人員認為 Swift 是一種只活在於蘋果生態系統中，適用於應用程式開發的語言，但現在情況已經不同了。Swift 已經發展成為一種強大而堅實的現代語言，具有廣泛的功能集合，我們（作者）認為這些功能集合適用於機器學習實作和教育。

近年來，蘋果公司發佈了兩款重要工具來鼓勵人們在其作業系統上追求上述的目標：

CoreML

一個 framework 和相應的模型格式，使已經訓練完成的模型有高度的可移植性和高效能。對於現行的第二個發行版本中，它的功能重點集中在電腦視覺和自然語言應用程式上，但是您仍可以將它應用於其他應用程式。

CreateML

一個用於建立和執行 CoreML 模型的 framework 和一個應用程式。它使訓練機器學習模型成為一個簡單和視覺化的流程。該版本於 2018 年發佈，並於 2019 年進行了改進，為那些一直在改編其他格式的模型、想把模型變成為可以與 CoreML 一起使用的人，提供了一個新的選擇。

我們將在第 2 章詳細介紹 CoreML 和 CreateML。

對智慧應用的推動並沒有止步於此。隨著 Siri shortcuts（一個可讓使用者使用個人助理自訂能力的功能集合）等新產品的問世，蘋果公司參與了一場盛大的文化運動，將個人化與智慧設備的發展延伸到了消費者層面。

所以使用者想要的是具有人工智慧的東西，而我們現在有了一個很酷的語言，合適的平台，以及一些很棒的工具。但許多人還是無法擺脫過去已熟悉的語言。那麼，如果他們可以使用一些他們已經熟悉的工具呢？您也可以用您喜歡的工具噢。

對於 python 有偏好的使用者，蘋果提供了 Turi Create（*http://bit.ly/2opKB2p*）。在 CreateML 出現之前，Turi Create 是首選：這個 framework 的用途是客製化機器學習模型的快速和視覺化訓練，可與 CoreML 一起使用。特別針對表格式或圖形資料，您可以切換到 Python 中，訓練一個模型，將其視覺化以驗證其適用性，然後直接回到一般開發流程裡。

但我們要用的是 Swift，記得嗎？

在出現 Swift for TensorFlow 專案時（*http://bit.ly/2MNJUbj*），它的專案頁上面自豪地說「……機器學習工具就是這樣重要，以致於它們配得上一個一流的語言……」，用這句話來迷惑死忠的 Python 使用者。TensorFlow 和相關的 Python 函式庫的公開可以直接在 Swift 中存取，即利用了兩種語言的優點。Swift 提供了更好的可用性和型態安全性，在不損害 Python 機器學習的靈活性的情況下，加上了 Swift 編譯器的強大功能。

在 2018 年的 TensorFlow 開發峰會上，Chris Lattner（Swift 語言的原作者，以及其他許多東西的創作者）公開了 Swift for TensorFlow，可在 YouTube（*https://youtu.be/Yze693W4MaU*）看到錄影，錄影更詳細地介紹了該專案存在背後的原因。

如果您想知道更多背後的故事，Swift for TensorFlow 的 GitHub 頁面有一個非常深入的文章（*http://bit.ly/2qaz2wB*）解釋他們為什麼選擇 Swift。

現在，在使用 Swift 做 AI 時選擇用什麼工具變得非常容易決定：如果是圖像或自然語言應用，或者如果目的是學習，就使用 CreateML；如果是要處理原始資料（大型值表等），輸入資料很多，或者需要進一步客製化，那麼請使用 TensorFlow。

我們將在第 2 章中討論更多關於工具，並且會在第 9 章中簡要介紹一下 Swift for TensorFlow。

內建機器學習的設備

歷史上，蘋果平台上的人工智慧功能略有不同，因為它們非常鼓勵在設備上做機器學習。許多其他製造商或開發人員，則是選擇將使用者資料發送到雲端以進行訓練，再透過網路回傳結果的產品。蘋果則提出，這種行為對使用者資料安全造成不必要的風險，因為資料可能包含個人資訊或習慣。

不意外地，這必然要付出代價。整個蘋果機器學習套件必須將性能放在首位，以允許單個設備（甚至是 iPhone）處理本地和離線智慧功能所需的所有操作。即便如此，一些資源密集型的系統功能也只有在設備處於非活動狀態、並且供電無虞時才能執行。

在我們正式開始討論用 Swift 實作人工智慧之前，我們必須為那些沒有人工智慧背景的人岔題一下。

人工智慧是什麼，它能做什麼？

人工智慧是一個研究領域和方法，試圖賦予技術一種智慧的**外表**。什麼屬於人工智慧或者什麼不屬於人工智慧存在著激烈的爭論，因為在一個**被告知**答案的系統，和被告知**如何找到答案的**系統之間有一條模糊的界線。

人工智慧基本上通常包括如**專家系統**（*expert system*）般的架構：專家系統是一種應用程式，以高度特定的資訊檢索或決策制定，來支援或替代一個領域的專家。它們可以由專門的語言或 framework 建造，但它們的核心歸納成許多巢式的 *if* 述句。事實上，我們敢打賭肯定是這樣。舉例來說：

```
func queryExpertSystem(_query: String) -> String {
    var response: String

    if query == "Does this patient have a cold or the flu?" {
        response = ask("Do they have a fever? (Y/N)")

        if response == "Y" {
```

```
            return "Most likely the flu."
        }

        reponse = ask("Did symptoms come on rapidly? (Y/N)")

        if response == "N" {
            return "Most likely a cold."
        }

        return "Results are inconclusive."
    }

    // ...
}
```

專家系統有一些實用的應用，但是構建這些系統非常耗時，並且始終需要對問題給出客觀的答案。儘管這樣一個大規模的系統可能看起來擁有大量的領域專業知識，但它顯然沒有靠自己發現任何新知識。雖然編纂了人類知識，甚至超越個人回想和回應時間，但其本身並不能形成智力，它只是依被告知的說而已。

那麼，如果是一個被告知如何找出或猜測答案，或者如何自行發現知識的系統呢？這就是如今大多數人口中所說的人工智慧。像類神經網路這樣熱門的方法，其核心是可以獲取大量資料的演算法，這些資料包括清楚的屬性和結果，並在一定程度上識別出人類無法識別的連結和複雜性。嗯，若人類有很長的時間和很多紙的話，也許一樣可以識別。

關鍵在於，人工智慧所涉及的一切都不是魔法：它並不能做到人無法做到的事；相反地，它是把簡單的事情做得比人類快得多。

您可能會問「等等，為什麼識別過去資料中的關聯是有用的？這如何使系統得到智慧？」。嗯，如果我們非常清楚什麼條件或屬性會導致什麼結果，當這些條件或屬性再次出現時，我們就可以有信心地預測會出現什麼結果。基本上，它使我們能夠做出更明智的**猜測**。

知道了這一點後，人工智慧不是什麼和不能做什麼就變得更清晰了：

- 人工智慧不是魔法。
- 對於敏感問題，不能信任人工智慧的產出。

- 人工智慧不能用於必須完全精確的應用中。
- 如果不存在現有的豐富知識，那人工智慧無法識別新知識。

這代表著人工智慧可以查看大量資料，並使用統計資料分析來顯示相關性。比方說「**大多數買了 *X* 書的人也買了 *Y* 書**」。

但是，如果沒有外部輸入或設計，它無法將這些資訊轉化為行動。例如「**推薦 *Y* 書給那些已買了 *X* 書（而且還沒有買 *Y* 書）的人**」。

而且，如果不提供更多資訊，它就不能推斷出包含新變數所包含的資訊。例如「**誰最有可能購買一本還沒有人買過的新書 *Z*？**」。

人工智慧與機器學習

無論如何，就機器而言擁有智力並不等於有進一步學習的能力。人工智慧和機器學習的區別取決於您問的是誰，但在這本書中，我們堅持以下幾點：

> **人工智慧有一些回授方法，讓它的訓練可隨著時間增長和完善，它具有機器學習的能力，並屬於機器學習的範疇。**

在技術層面上，機器學習可以簡單地實作為具定期重新訓練架構的人工智慧。像蘋果這樣的公司經常把整個領域稱為機器學習，更是造成更多混淆。這兩個術語之間的界限並不重要；重要的是一個特定的技術解決方案的需求描述。

一個能夠根據溫度和濕度預測天氣的系統，可以在訓練一次以後，就使用很長一段時間，而一個推薦系統應該在任何時候納入到新產品和新的使用者行為。

深度學習和人工智慧？

您可能已經注意到，除了術語「機器學習」和「人工智慧」，有時人們還會提到**深度學習**（*deep learning*）。深度學習是機器學習的一個子集。這是一個有點時髦的名詞，但它也是一種必須倚靠多層次的重複工作來執行任務。

深度學習是使用越來越複雜的類神經網路層來進一步從資料集中提取出實際相關資訊。其目標是將輸入轉換為抽象的表示形式，以便稍後用於各種目的，例如分類（classification）或推薦（recommendation）。本質上，深度學習之所以被稱作深度，是因為它透過分層的類神經網路進行了大量重複的學習。

類神經網路是哪來的？

根據您的背景不同，您可能會認為人工智慧和機器學習**不過**就是類神經網路。這不是真的，從來都不是真的，未來也不會是真的。類神經網路只是一個流行詞，也是圍繞這些主題的**炒作**主題之一。

在本書的後面，在第二部分的所有**實踐**任務之後，我們將在第 10 章中以更理論的角度來研究類神經網路，但是這本書著重的還是實作。事實上，現在您是否關心類神經網路並不重要；由於我們有很好的工具，所以您可以在不需要知道或關心工具是如何工作的情況下，構建**特徵**（*feature*）。

合乎道德、有效，並適當地使用人工智慧

人工智慧可以被用來做壞事。這麼說不令人意外；幾乎所有一切的東西都可以用來做壞事。但是我們人類很久以前就知道如何做出智慧技術，而且在投入大量精力製造能**自我解釋**的智慧技術之前，智慧技術就已經相當流行了。在這一領域，人工智慧研究還處於起步階段。

現在，如果我們有一個其他類型的系統，但它偶爾會出錯，我們可以對它進行除錯，找出出錯的地方和原因。

但我們不能對人工智慧做一樣的事。

如果我們有一個其他類型的系統，其輸出會因為輸入改變，而我們不能修改系統本身的話，我們可以去檢查並嘗試手動分析那些會導致錯誤的輸入。

但我們不能對人工智慧做一樣的事。

無法自我解釋的智慧系統如果出錯，通常找不到要從何處著手。

無法修改已部署的系統這件事，也引發人們規避責任，人們宣稱自己建立的系統不代表他們的觀點，並拒絕為自己的錯誤負責。

最近的一個例子是：一款為移動裝置發行的照片編輯應用程式，號稱可以將您的照片調整到幾個事先定義的模式。若您輸入一張自拍照，它會將這張自拍照中的您改為不同性別、變更年輕或變更老。其中一個可輸出的模式聲稱會對照片進行微調，使個人看起來「更性感」，但膚色較深的使用者很快就注意到，這個功能會把他們的膚色變亮。

毫無疑問地，這使得使用者受到了傷害，因為他們覺得膚色越淺越有吸引力。這種被號稱為「種族歧視設計」對開發人員造成困惑，但也承認他們的應用程式的輸入資料集是在內部建立的。建立時會給無數人的照片標上標籤，供他們的人工智慧在訓練時使用，以致於他們傳遞了自己固有的偏見：他們個人認為歐洲血統的人更有吸引力。

但是演算法不知道種族主義是什麼，它不會有人類的偏見，不知道美，也不知道自尊。這個系統只是收到大量的照片，被告知哪些照片是有吸引力的，並被要求要複製所有它所識別出來的共同屬性，全都是很明確的行為。

當他們在進行內部測試時，執行者碰巧也是白人，所以沒有察覺出問題，於是這個應用程式就發行出去了。

還有更多影響深遠的例子也遭受了類似的問題：例如，汽車保險系統認定所有女人都很危險，求職系統認為名字是不是叫 Jared 事關重大，假釋系統認為所有深色皮膚的人會犯罪，醫療保險系統以超市的信用卡交易資料評估一個人的飲食和健康狀況，認為生鮮市場購買食物的人情況較差。

所以，重點是要明白雖然人工智慧是一個非常酷和有趣的新技術領域，我們可以用它來取得卓越的進步和變得更好，但它真的只能複製我們社會既有的條件，它無法有**道德感**。

問問自己：第一部分

1. 人工智慧系統套用於現實世界中，會產生什麼樣的問題？
2. 如何在系統中改善這些問題？例如：
   - 將輸入資料整理成更能表達概念的形式
   - 用多個不同系統，以互相補正或挑戰其他系統
   - 重新建構詢問系統的問題，以求得更客觀的答案
3. 如果這個系統給出的所有答案都不對，會導致的最大損害是什麼？
4. 如何將風險或潛在傷害減低？例如：
   - 在發佈以前確認達到一定的精準度
   - 內建故障安全裝置以忽略或警示潛在有害的回應
   - 人為調整回應
5. 就算仍有風險或潛在的危害，在最壞情況發生時，您仍然可以處理嗎？

這並不是說設計人員 / 開發人員沒有這種權力。修改輸入資料塑造成我們想要的世界，而不是實際的世界，但是目標化也會導致其他的不準確性。現在，我們所能期望的最好的結果就是意識到並接受責任。

至少要這樣做。

然而，即便是合乎道德地使用人工智慧，也並不總是有效或恰當的。許多人都太沉迷於開發一個充滿未來科技的智慧系統，而這種智慧系統現在正風靡一時；他們不再關注他們試圖解決的問題或試圖創造的使用者體驗。

假設您可以訓練一個類神經網路來告訴某人他們可能會活多久。它利用了所有現有的歷史和當前的醫學研究，數百年來對數十億人的治療和觀察，這使得它相當準確——排除了意外或外力造成的死亡。它甚至可以為不斷增長的平均壽命背書。

當一個人打開這個應用程式 /web 頁面 / 無論什麼，然後輸入他們所有的資訊，送出資訊，然後得到一個答案，這個答案只有一個數字，人們只能相信該數字，沒有其他更多解釋，所以它無法提供對人類來說有意義的東西。

即使是那些得到良好結果的人，也可能會造成嚴重的困擾。

因為人類需要的不是一個神奇的答案。當然，他們喜歡得到人類無法解決的問題的答案，但只給一個單獨的回應並不能解決問題。大多數人都想知道他們的預期壽命，這樣他們就可以延長壽命，或者知道如何在以後提高他們的生活品質。真正的問題是，他們能做些什麼來應對可能出現的問題，而這個答案並不存在於系統的回應中。

一個專家系統可能表現得更好：它知道的事情雖然更少，但它可以解釋自己產出的結果，給出**可操作的**答案。在大多數應用程式中，這正是智慧系統的設計所追求的目標，而不是其他的。代價是犧牲智慧甚至一點點準確性，類神經網路解決方案就無法達到這種**效果**。

問問自己：第二部分

1. 如果有足夠的時間或工作，一個「笨」的解決方案能解決這個問題嗎？
2. 如果是這樣，那聰明的解決方案又有什麼價值呢？
3. 如果一個人類被問到一個系統可以回答的問題，您希望他們除了回答以外，還能有什麼解釋呢？

4. 一個聰明的解決方案能提供一個可比較的解釋嗎？

5. 您試圖為最終產品的使用者提供什麼樣的體驗？

最後一個例子：假設有一個公司正困擾著如何在第一次更新後留住會員，管理層級為如何留住會員而苦惱，但用於通信的通知訂閱率、讀者群或回復率都很低。大多數新成員甚至在他們的第一個會員效期過半後都無法與他們取得聯繫，管理層級廣泛尋找解決方案。管理層級希望知道如何獲得更多關於其成員的資訊，以便能知道什麼構成問題，或更換其他方式聯絡方式。

收到的解決方案包括資料收集、聘用資料機構或雇傭管理顧問這種公司會清除所有與資料相關的障礙。公司需要的是告訴電腦更多的客戶資料，然後電腦會告訴它該做什麼。公司將建立一個系統來識別出很可能不會進行更新的成員，並確定哪些保留會籍的方案是最有效的。管理層級告訴自己，這個系統會很棒。

但這裡的問題不在於該公司擁有成員資料太少。問題在於公司無法**聯繫**它的成員，不能充分地**向成員展示保留會籍的價值**，而且資料揭示的資訊太少，以致於公司不**認識**它的會員或產業，也許管理層級根本不想知道。它偏離了最初提出的問題，現在未能設計出**適合**其目的的解決方案。

問問自己：第三部分

1. 如果沒有人工智慧，人類要如何解決這個問題？請考慮以下的領域：

社會研究

現今的使用者分析方法所能得知的事情，已經可以取代傳統市場調查、文獻分析與客戶調查等。

數學統計

對許多人來說，手動分析大量的資料不是件有趣的事，但電腦可以用來識別屬性和結果之間的關係，對於該領域的人員來說，只要用常識就可以識別出其中重要且有意義的關連。

土耳其機器人（*Mechanical Turk, http://bit.ly/2ptZCjJ*）

這個名詞出自 18 世紀的一款革命性的國際象棋機器人，但這個機器人其實只是有一個人躲在盒子裡而已。千萬不要低估人類解決問題的能力，我們有許多技能是電腦無法模仿的。

2. 問題真的要問的是什麼？
   - 這個解決方案能回答問題嗎？
   - 如果不能回答，任何智慧解決方案可以回答同一個問題嗎？
   - 如果不能，更換的資料或可用方法是否可以解決原有的問題？

有了這些思想上的原則（道德的、有效的、適當的），現在就讓我們前進，學習為自己建立一些具人工智慧功能吧（這就是這本書後面要講的東西，所以我們希望您能繼續讀下去）。

實際人工智慧任務

我們想要寫這本書，是因為我們厭倦了那些聲稱從探索如何實作令人驚艷的一堆類神經網路開始，聲稱要教一些有用的、實用的人工智慧技術的書（但最後因完全沒使用環境而變成毫無用處收場）。我們讀過的大多數關於人工智慧的書實際上都非常好，但它們都是從自下而上的方法開始的，從演算法和類神經網路開始，到最後的實作和實際應用。

典型任務導向方法

我們將在本書中採用的典型方法是自上向下的說明，我們將把所關注的 AI 的每個領域分解成需要解決的個別任務。

會這麼做是因為很少您一開始就問「我想要風格分類器系統」（雖然您有可能問出這樣的問題）。您通常在開始時會問「我想做一個應用程式，它可以告訴我如果這幅畫屬於**浪漫主義**或**前拉菲爾派**的風格」（這是一個在我們人生中可能會出現的問題）。

在本書第二部分中的每個章節都以我們想要解決的問題開始，並以一個能解決此問題的**實用**的系統作為結束。流程都是一樣的，每章節只有一些小的變化：

- 對我們正要處理的問題的描述
- 決定用一種通用的方法來解決它
- 收集或建立用於解決此問題的資料集
- 做出我們的工具
- 建立機器學習模型來解決問題
- 訓練模型
- 建立使用該模型的應用程式或 Playground
- 將模型連接到 App 或 Playground
- 拿出來秀一下

我們採用這種安排的原因很簡單——這本書被稱為「**實用**」**人工智慧與** *Swift*，而不是「**有趣但脫離現實世界不切實際的一個系統**」**人工智慧與** *Swift*。將我們想說的包裝成一個個的任務，我們要求本書中的元素能夠最好地解決手頭的問題，同時要求它們能夠以實際的方式使用。

幾乎所有的章節都會使用 CoreML 作為我們建立的模型 framework，所以我們所做的任何事情都需要支援它。

在某些方面，CoreML 是我們畫出的一條分際線，不過在另一方面來說，它給了我們明確的限制條件，我們都喜歡明確的限制條件。

人工智慧發展上大量的工作是由學者和大公司進行，儘管他們也有自己想做的事，但他們想做的事在目標和訓練、推理和資料資源的限制上，都與那些不得不自己將工作變得有用的人非常地不同，而（通常）那些人需求和資源都較少。

模型和方法可能對構建和研究很有吸引力，但卻需要強大的機器來訓練和執行，或者那些看起來很有趣但不能解決任何特定問題的模型和方法，其實根本不實用。您可以在普通的桌上型電腦或筆記型電腦上執行這本書中的所有內容，然後在 iPhone 上編譯和執行它。在很多方面，我們覺得這才是務實的本質。

因為我們要把這本書分解成不同的任務，所以我們不能做到一半；我們需要建立一些可用的東西來完成每個任務。大多的人工智慧方法不是用來處理人工智慧模型的建立，就是用來將模型連接到使用者所面對的系統。這兩者我們都會做。