從簡單的數學瞭解機器學習

作者：哈利 Harry

版權聲明

目錄

前言

怎麼會想寫這本書？

會購買此書的你想必也有感覺到，AI時代即將來臨。面對幾乎所有事情都能夠有AI輔助（或甚至從頭到尾幫你處理）的時代，你有兩個選擇：掌握AI科技，運用AI創造極大的價值。或者，你也可以被動地接受改變，變成被AI科技控制的人。你如果想要成爲前者，你必須要瞭解AI。而瞭解AI的第一步，就是機器學習。

目前幾乎所有的AI系統背後的技術，都是機器學習。我認爲這個學科在未來會像是其他的基礎自然科目一樣重要。

因爲看到AI系統的强大，我們很容易把機器學習想得過於困難。當然，你想要變成研究機器學習的最前沿，你是會需要蠻困難的數學觀念。但是并不是每個人都想成爲AI博士啊！

若是單純地想要打開機器學習的黑盒子，瞭解它是如何運作的，你不需要很困難的數學觀念。這本書就是活生生的例子。

機器學習是每個人都可以瞭解/應該理解的，無論背景或是經驗。**我希望這本書可以幫助每一個想要搞懂AI的人。幫助他們騎在浪頭上，不被浪淹沒。**

關於本書

《從簡單的數學瞭解機器學習》旨在用簡單的數學，帶你徹底地瞭解機器學習運作的原理。數學跟機器學習是分不開的，我認爲從數學理論切入機器學習是最扎實的學習方法。同時，儘管數學是可以很難的，機器學習的數學真的不用很複雜。

這本書非常適合機器學習初學者。任何有高中數學知識的人都能夠理解所有內容，不需要先懂機器學習或是大學程度的數學。但這並不代表這本書不具有挑戰性。有些概念即使具備不錯的高中數學基礎，也會有一定難度。我會盡可能的用最容易懂的方式解釋，并且假設讀者只有高中的數學基礎。

本書分為7個章節。 第1至3章是機器學習、線性代數和微積分的概述。第4章是簡短的數學概念懶人包，第5章是機器學習案例研究。第4章是專門為第5章設計的，只包含了理解第5章會用到的數學。 第6章是下一步，第7章是線性代數概念的補充。

整本書最重要的是第五章。其他1~4章其實都是爲了讓你完全瞭解第五章而設計的。你如果看完這本書后發現能把第五章的內容幾乎全部理解，那我覺得就成功了！

跟其他所有學科一樣，機器學習也有分「理論」跟「實務」。這本書是偏「理論」，但是我仍然有附上我的程式碼，讓讀者一窺實務上會怎麼建機器學習模型。

同時，這本書介紹的許多概念都是僅在高層次上介紹，並且可能沒有完全詳盡的解釋跟例子。這是爲了易讀性必須犧牲的細節。

第1章：機器學習

什麼是機器學習

機器學習是「透過數據和演算法讓電腦能夠做出決策或預測」的技術。我們會使用演算法構建模型，使用數據訓練模型，並使用模型進行預測。目前絕大多數的AI系統都是由機器學習的技術驅動的，從Instagram的推薦演算法到ChatGPT都是。

你聽別人講到「機器學習模型」時，你要想到的是一個有「輸入」跟「輸出」的物件。你只要「輸入」這個物件一個符合物件需求格式的值，物件就會根據這個輸入值進行一系列的運算，并且吐出一個相應的輸出值。就像Instagram reels將用戶的行為/興趣作為輸入，並輸出要推薦的視頻；ChatGPT將使用者的提示作為輸入，並輸出答案。任何的機器學習模型都是這樣。

具體來說，模型長什麼樣子？

我知道，現在你可能還是覺得「機器學習模型」是很抽象的東西。讓我們使用一個非常簡單的例子來理解這個概念。

假設你想要建一個會根據人的身高猜測（預測）人的體重的機器學習模型。換句話說，該模型以一個人的身高作為輸入，並輸出那個人的估計體重。

這個模型可能長這樣：

$$w_1 X^1 + w_0 = y^{pred}$$

X^1 是輸入（身高），y^{pred} 是模型的輸出（體重）（$pred$ 代表"預測"的意思）。

上述公式中的 w_1 和 w_0 被稱為該模型的「**參數**」。 這些參數也稱為「**權重**」（因此使用w符號。Weights 的意思）。 在本書中，我會交替使用這兩個術語，因爲在本書提到的內容裏面這兩個詞都是在講一樣的東西（雖然它們之間確實有微小的差別）。

你可以看到，就這麼簡單的一行數學函數，也可以稱做一個「機器學習模型」。本質上，機器學習的模型就是一個數學函數而已，就連ChatGPT都一樣（雖然它的式子複雜得多，不可能直接寫出來，但一樣就是數學函數而已）。

權重（w_1 和 w_0）描述了 X^1 和 y^{pred} 之間的關係。在建這個模型的時候，最關鍵的目標就是找出最佳權重值。何謂「最佳」？就是當我們將任何人的身高作為 X^1 輸入時，模型輸出的 y^{pred} 會最接近那個人的實際體重。

我們通過「訓練模型」的過程來取得最佳權重。簡單地說，訓練過程就是電腦利用過去的數據來找出權重的最佳值的過程。訓練過程中，電腦會不斷比較模型輸出（y^{pred}）和實際輸出（y^{actual}）之間的差距，并且不斷調整權重來縮小這個差距。

在電腦找到了權重（w_1和 w_0）的最佳值之後，人們會評估模型的能力，幫它打個分數。若是分數夠高，這個模型可能就會被用來解決實際的問題。若是不夠高，則會持續的調整模型。

雖然機器學習的模型都長得不太一樣，而且大多數比 $w_1 X^1 + w_0 = y^{pred}$ 複雜得多，但是我上面講的這些內容在大部分機器學習模型都是通用的—模型的最終目標都是根據輸入產生「最佳」的輸出，且是透過權重的優化（不斷比較模型輸出跟實際輸出）來做到這一點。我們將用這種方式學習的模型歸類爲「監督式學習」。

「監督式學習」是最常見、應用最廣汎的機器學習方法，也是本書著重的部分（第五章的個案就是這種）。其他的學習方法還有「非監督式學習」跟「增強式學習」這兩種類別。在「非監督式學習」中，模型會自動

發現數據中的結構、模式或關聯性，而無需比較模型輸出跟實際輸出。在「增强式學習」中，則是讓機器學習從錯誤中學習，通過試錯的過程來找出最好的行動策略，以獲得最大的獎勵。

機器學習會用到什麼數學

數學是機器學習的核心。 正如我之前提到的，所有機器學習模型本質上都只是數學函數而已。

但是，請放心！你不需要完全掌握所有的數學才能開始學機器學習。事實上，你只需要瞭解三個核心數學領域，而且不必完全理解這些領域的所有概念。

這些領域有：

- 線性代數
- 微積分
- 幾率論和統計學

你可以把這三個數學領域視為三個獨立的工具包。 每個工具包都有特定目的：**線性代數**主要用於表示數據和操作數據，**微積分**主要用於優化模型，而**概率論和統計學**主要用於方法論和評估結果。

你在使用機器學習解決問題的時候（建機器學習模型的時候），你并不會用到每個工具包中的每個工具。這意味著你不必掌握這三個領域中所有的概念，就可以開始解機器學習的問題了。先學用得到的就行了，而且用得到的數學一點都不難！

本書的設計，就是在2~4章只教你剛好足夠你理解第5章的個案的數學。我不會教的太多太難，讓你想放棄；也不會教的太少，少到讓你不知道第5章在幹嘛。

第2章：線性代數

什麼是線性代數？

線性代數是研究**線性方程組**的學科。什麼是**線性方程組**？ 以下是一個例子：

$$\begin{cases} x + y = 3 \\ 3x + 2y = 8 \end{cases}$$

你一定見過長這樣的方程組了吧？ 這是一個由兩個**線性方程式**組合成的一個**線性方程組**。一個線性方程組就是一個或多個線性方程式的組合，描述了變數（變數就是 x 和 y）之間的關係。

那麼爲什麼這些方程式都被稱作「線性」？線性究竟是什麼意思？

數學上，「線性」的定義是滿足**可加性**跟**一次齊次性**。詳細定義如下：

> 💬 若$f(x)$是一個線性函數，則滿足：
> 可加性：$f(x+t) = f(x) + f(t)$
> 一次齊次性：$f(mx) = mf(x)$

通常，你如果看到方程式所有的變數都只有到1次冪（就是只有一次方），並且變數之間只有用到加法（或減法），那這個方程式就是個線性方程式。

如果你看到長這樣的方程式：$x^2 + 2yz = 4$，它不是線性方程，因為 x 有2次方並且變數之間存在乘法（$2yz$），因此不包括在線性代數的研究範圍內。

線性代數引進了**向量**和**矩陣**這兩個數學物件。線性方程組通常都會由**向量**和**矩陣**來表示。上面的例子用矩陣和向量表示的話就會長這樣：

$$\begin{bmatrix} 1 & 1 \\ 3 & 2 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 3 \\ 8 \end{bmatrix}$$

所有線性代數的觀念幾乎都是圍繞著這些叫做**向量**和**矩陣**的奇怪的數字塊。所以讓我們先來瞭解一下它們是什麼。

什麼是向量和矩陣？

這是一個蠻難回答的問題。首先讓我們看看教科書通常如何定義它們（可能會有點抽象，但是不用擔心！實際例子很快就來了）：

向量和矩陣是**可以使用明確定義的規則進行運算的數學物件**。它們具有許多重要的代數和幾何特性。

它們通常表示如下：

向量：

- 代數表示：一列數字，通常表示為一個方括弧中的一列。可以是直的（列向量）或是橫的（行向量）。
    - 列向量

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ ... \end{bmatrix}$$

- 行向量

$$\begin{bmatrix} x_1 & x_2 & x_3 & ... \end{bmatrix}$$

- 幾何表示：空間中的一條線，具有長度和方向。

矩陣：

- 代數表示：一個二維的矩形數字組，位於方括弧內。一個 m X n 矩陣由 m 行和 n 列組成。

$$\begin{bmatrix} a_{11} & a_{12} & ... & a_{1n} \\ a_{21} & a_{22} & ... & a_{2n} \\ ... & ... & ... & ... \\ a_{m1} & a_{m2} & ... & a_{mn} \end{bmatrix}$$

- 幾何表示：一個可以將向量空間旋轉，拉伸和翻轉的**線性變換**。它可以應用於向量，將其旋轉，拉伸和翻轉。

> 💬 你還可以將矩陣的每一列視為一個向量。 因此，在這個意義上，矩陣由多個向量組成。

你可能會覺得這些概念非常抽象，很難理解向量和矩陣到底是什麼？爲什麼要把一些數字框起來？尤其**線性變換**又是啥鬼？ 這是很正常的，因為教科書通常會故意保持概念的抽象性，以便你之後將這些概念應用到不同的領域。要擴大觀念的廣汎適用性，就得犧牲具體性。

因此，與其思考向量和矩陣到底是什麼，不如思考**哪些東西可以用向量和矩陣表示**，因為向量和矩陣從來都不是具體的物件，它們只是一個數學概念，**只有在代表實際東西的時候，它們才有意義。**

而廣義來説，任何滿足線性代數公理的東西都可以用向量跟矩陣來表示。接下來我會詳細説明。

我們爲什麼要學線性代數？

數學家們發現向量和矩陣有一些非常好的特性，讓人們能夠對它們進行各種計算、描述、分析和操作。所以只要能把問題用向量跟矩陣來表示，我們就能運用向量跟矩陣的特性，進行各種厲害的操作。

幸運的是，許多領域的問題都可以用向量和矩陣來描述，包括工程、物理、經濟學，當然還有計算機科學和機器學習。利用向量和矩陣的好特性，這些問題可以非常容易且高效地被解決。這就是為什麼線性代數是許多理工專業必修課程的原因。

舉個例子：

在工程中，向量和矩陣可以用於表示物理系統，如電路和機械結構，並模擬它們在不同條件下的行為。

在金融領域，向量和矩陣可以用於表示市場數據，如股票價格，並進行金融建模和分析。

在機器學習中，向量和矩陣用於表示數據，如常見的數據表格，甚至是圖像或文本。你很快就會看到具體的例子了。

線性代數在機器學習有什麼用？

線性代數可以讓整個機器學習的過程變得更加高效！

實務上，在訓練機器學習模型時，幾乎所有的計算都是通過線性代數完成的。輸入數據是用矩陣表示，而權重則用向量（或矩陣）表示。之所以這樣表示，是因為對於有GPU的計算機來說，執行矩陣操作比逐行計算要高效得多，因為它可以讓計算機進行**平行運算**。

> 💬 GPU是圖形處理器（Graphics Processing Unit）的簡稱，它是一種專門在個人電腦、工作站、遊戲機和一些行動裝置上執行繪圖運算工作的微處理器。它類似於中央處理器（CPU），但扮演不同功能。CPU架構比較複雜，功能比較泛用，而GPU採用的平行運算架構比較單純、核心數量較多，適合處理專精的工作，像是AI模型的訓練。

讓我詳細解釋一下**平行運算**是啥。回到第1章中的例子（當我們有一個像 $w_1X^1 + w_0 = y^{pred}$ 這樣的機器學習模型時），如果你收集到了10000個人的體重和身高數據，那麼問題本質上就是一個有10000個線性方程式的線性方程組，它看起來像這樣：

$$\begin{cases} w_1X_1^1 + w_0 = y_1^{pred} \\ w_1X_2^1 + w_0 = y_2^{pred} \\ w_1X_3^1 + w_0 = y_3^{pred} \\ ... \\ w_1X_{10000}^1 + w_0 = y_{10000}^{pred} \end{cases}$$

沒有矩陣的話，電腦會一行一行計算這些方程式。

然而，當我們用矩陣和向量來表示這個線性方程組的時候，你可以只用一個方程式表示它，如下所示：

$$\begin{bmatrix} X_1^1 & 1 \\ X_2^1 & 1 \\ X_3^1 & 1 \\ ... & \\ X_{10000}^1 & 1 \end{bmatrix} \begin{bmatrix} w_1 \\ w_0 \end{bmatrix} = \begin{bmatrix} y_1^{pred} \\ y_2^{pred} \\ y_3^{pred} \\ ... \\ y_{10000}^{pred} \end{bmatrix}$$

對於電腦來說，進行上面這一行程式的計算等於是在執行一次「矩陣乘法」。這是一種有明確定義且被優化過的線性代數運算。在這個「矩陣乘法」中，計算機等於是將所有方程式同時平行處理，因為它們現在都包含在兩個矩陣物件中。在資料量非常大，計算非常複雜時，與逐行計算相比，用向量跟矩陣計算會更加高效。在訓練大型神經網路（有極大量參數）的時候，這個差別就會非常有感了。

第3章：微積分

什麼是微積分？

微積分是**研究事物如何變化**的一種數學。 它是拿來分析/描述正在變化的事物的一種工具。

顧名思義，微積分分成兩個部分：微分跟積分。微分關注的是事物變化的速率，而積分則處理隨時間累積的變化。

讓我們用一個現實生活的例子來理解微積分是什麼。

想象你正在路上開著你的特斯拉。你的時速表壞了，但是你的里程表是好的，而且車子會幫你記錄每時每刻里程表的資訊。如果你想知道你的平均行駛速度，你可以使用非常簡單的國中數學：

$$速度 = 距離/時間$$

假設你行駛了120公里，行駛了2小時，那麼你的平均速度是60公里/小時。

但是這種方法只能找到「平均速度」。 如果你需要更具體的訊息怎麼辦？

比如說，你回想起在第57分鐘12秒時，你看到了一個測速相機（別問我你是怎麼記得這個的）。你想知道你在那個時間點的速度是否超過了限速，要怎麼計算？

這正是微積分中的**微分**可以幫助你解決的問題。使用微分的方法，你可以從現有的資訊，像是行駛距離跟時間的函數（從車子記錄的里程表資訊得到），取得**速度函數**。拿到速度函數之後，你只需將確切時間放入函數，它就會告訴你該時間點的速度。

換言之，你可以從一個現有的函數（里程跟時間的函數）得到一個新的函數（速度函數），然後用新的函數得到你想要的資訊。

在第4章中，你會學到如何做到這件事。

積分則是完全相反。假設現在你的特斯拉時速表已經修好了，但反而里程表壞了。當你駕駛時，特斯拉會記錄下每個時間點的速度。

只有這些資訊，也就是只知道你在每個時間點的行駛速度，你要如何知道你開了多遠？

如果你一直以恆定的速度行駛，那麼這又是初中數學問題。只需要用以下公式就行了：

$$距離 = 速度 * 時間$$

但你的車速不太可能一直維持不變。在車速每分每秒都在改變的情況之下，你要怎麼算出你開的距離？積分可以告訴你如何解決這個問題。透過積分的方法，你可以從時間的角度獲得**距離函數**。然後一樣，你只需要把你想知道的時間插入新的距離函數，它就會告訴你在這段期間你走了多遠。

爲什麼要學微積分？

微積分是現代科學發展中最重要的因素之一。說是它造就了工業革命也不爲過。

微積分非常有價值，因為它可以把一個非常困難的問題分解為許多簡單的小部分。具體來說，它把**一段時間內的變化**分解為許多**瞬時變化**（非常短的時間間隔內的變化）。這些瞬時變化是比較好拿來分析的單位。

變化無處不在，而微積分提供的工具讓我們能更好的分析這些變化。這就是為什麼微積分是所有理工學科的必修課程的原因。每個科目都有想要瞭解的**變化**。

例如，微積分可以幫助我們理解物體的運動，系統的行為以及變數之間的關係。它可以用於找到問題的最佳解，例如最小化成本或最大化利潤。它還可以用於分析和預測未來的趨勢，例如人口增長，經濟發展或氣候變化。

微積分在機器學習中有什麼用處？

微積分，尤其是微分，對機器學習非常有用，尤其是在模型的訓練過程中。更具體地說，它提供了一種方便的方法來將**損失函數**最小化。意思就是說，微分會跟我們說應該要往哪個方向持續調整調整模型的權重。

還不知道什麼是損失函數的，不用擔心，第5章會有詳細解說！

許多知名且強大的機器學習演算法都需要用微積分來達成訓練的目的，包括神經網路，線性回歸，支援向量機等等。沒有微積分，這些演算法根本無法優化。

第4章：基本數學概念

在這一章，我會簡單介紹一些線性代數跟微積分的基本概念。這些概念學完剛好足以讓你理解第5章的個案研究。相信我，機器學習的數學比你想像的要簡單得多！

請注意，因爲篇幅限制，同時也因爲我不想把你嚇跑，下面講的概念定義和範例並不是最完整和詳盡的。有一些更深入的解釋，或是一些極端的範例，都被我省略了。但是我能保證的是，我提到的所有內容都是跟機器學習領域最重要的基礎概念。

切記，你不需要背起下面的內容，也不需要每一行都完全瞭解。看懂大觀念（這是什麼？爲什麼要學？）是最重要的。有些內容如果你覺得太細太繁雜，也可以先跳過不看。**定義**看不太懂，也可以先看**直覺理解**的部分。

線性代數概念

1：矩陣/向量運算

- **這是什麼：**

  矩陣和向量的運算規則，包括：加法、減法、乘法、標量運算、轉置和逆矩陣。

- **爲什麼要學這個：**

  這些是我們可以在矩陣上執行的基本運算。

- **速查表**

  ▼ *加法和減法*

  - 規則
    - 兩個矩陣相加時，只需將相同位置的所有元素相加即可。結果為一個新矩陣。
    - 只有具有相同行數和列數的矩陣才能相加或相減。
  - 範例

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \quad B = \begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{bmatrix}$$

$$A + B = \begin{bmatrix} a_{11} + b_{11} & a_{12} + b_{12} & a_{13} + b_{13} \\ a_{21} + b_{21} & a_{22} + b_{22} & a_{23} + b_{23} \\ a_{31} + b_{31} & a_{32} + b_{32} & a_{33} + b_{33} \end{bmatrix}$$

  - **特性**（A, B, C都是相同大小的矩陣）
    - 交換律：$A + B = B + A$，$A - B \neq B - A$
    - 結合律：$(A + B) + C = A + (B + C)$，$(A - B) - C \neq A - (B - C)$
    - 加法恆等元存在：存在一個矩陣 O（通常稱為零矩陣），使得任何矩陣 A 滿足 $A + O = A$
    - 加法逆元存在：對於任何矩陣 A，存在一個矩陣 $-A$，使得 $A + (-A) = O$，其中 O 是零矩陣。

  ▼ *乘法*

- 規則
  - A, B 是兩個矩陣。要計算乘積 AB 時，我們將 A 的第 i 行的元素乘以 B 的第 j 列的元素，然後將結果相加。
  - 例如，AB 的(i, j)元素為：$(AB)_{ij} = a_{i1}b_{1j} + a_{i2}b_{2j} + ... + a_{in}b_{nj}$
- 範例

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \quad B = \begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{bmatrix}$$

$$AB = \begin{bmatrix} a_{11}b_{11} + a_{12}b_{21} + a_{13}b_{31} & a_{11}b_{12} + a_{12}b_{22} + a_{13}b_{32} & a_{11}b_{13} + a_{12}b_{23} + a_{13}b_{33} \\ a_{21}b_{11} + a_{22}b_{21} + a_{23}b_{31} & a_{21}b_{12} + a_{22}b_{22} + a_{23}b_{32} & a_{21}b_{13} + a_{22}b_{23} + a_{23}b_{33} \\ a_{31}b_{11} + a_{32}b_{21} + a_{33}b_{31} & a_{31}b_{12} + a_{32}b_{22} + a_{33}b_{32} & a_{31}b_{13} + a_{32}b_{23} + a_{33}b_{33} \end{bmatrix}$$

- **特性**
  - 並非所有矩陣都可以相乘。要相乘兩個矩陣，第一個矩陣的列數必須等於第二個矩陣的行數
  - 矩陣乘法不滿足交換律。也就是說，通常情況下，$AB \neq BA$
  - 結合律對矩陣乘法成立。也就是說，對於任何三個矩陣 A, B, C，我們有：$(AB)C = A(BC)$
  - 分配律也對矩陣乘法成立。也就是說，對於任何三個矩陣 A, B, C，我們有：$A(B + C) = AB + AC$

▼ *標量加法*

- 規則
  - 標量就是一個數值的意思，像是5或是21之類的。
  - 要將標量 s 加到矩陣 A 中，只需將 s 加到 A 的每個元素中即可。
- 範例

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \quad s = 2$$

$$A + s = \begin{bmatrix} a_{11} + 2 & a_{12} + 2 & a_{13} + 2 \\ a_{21} + 2 & a_{22} + 2 & a_{23} + 2 \\ a_{31} + 2 & a_{32} + 2 & a_{33} + 2 \end{bmatrix}$$

▼ *標量乘法*

- 規則
  - 要將矩陣 A 乘以標量 s ，只需將 A 的每個元素乘以 s 即可。
- 範例

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \quad s = 2$$

$$sA = \begin{bmatrix} 2a_{11} & 2a_{12} & 2a_{13} \\ 2a_{21} & 2a_{22} & 2a_{23} \\ 2a_{31} & 2a_{32} & 2a_{33} \end{bmatrix}$$

- 特性
  - 交換律：$As = sA$
  - 結合律：$s(tA) = (st)A$
  - 分配律：$(s + t)A = sA + tA$

▼ *轉置*

- 規則
  - 把一個矩陣轉置，就是交換其行和列的意思。
  - 比如說，如果 A 是一個 m X n 的矩陣，其元素為 a_{ij}，則 A 的轉置（表示為 A^T或A'），是一個 n X m 的矩陣，其元素為 b_{ij}，使得 $b_{ij} = a_{ji}$。
- 範例

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{bmatrix}$$

$$A^T = \begin{bmatrix} a_{11} & a_{21} \\ a_{12} & a_{22} \\ a_{13} & a_{23} \end{bmatrix}$$

- 特性
  - $(A^T)^T = A$，即矩陣的轉置的轉置是原始矩陣。
  - $(A + B)^T = A^T + B^T$，即兩個矩陣的和的轉置等於它們的轉置的和。
  - $(cA)^T = cA^T$，即標量和矩陣的乘積的轉置等於矩陣的轉置和標量的乘積。
  - $(AB)^T = B^T A^T$，即兩個矩陣的乘積的轉置等於它們的轉置的乘積的逆序。

▼ *逆矩陣*

- 規則
  - **方陣** A（方陣是正方形的矩陣，意即行數跟列數一樣）的逆矩陣，表示為 A^{-1}，有這樣的特性：當 A乘以它的逆矩陣時，得到的結果是一個單位矩陣 I。
  - 單位矩陣 I 是一種特殊的方陣，它在主對角線上（從左上到右下）都是1，其他地方都是0。

$$I_3 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

  - 也就是說，$AA^{-1} = A^{-1}A = I$。
- 範例

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$$

$$AA^{-1} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$A^{-1} = \frac{1}{\det(A)} \begin{bmatrix} \begin{vmatrix} a_{22} & a_{23} \\ a_{32} & a_{33} \end{vmatrix} & -\begin{vmatrix} a_{12} & a_{13} \\ a_{32} & a_{33} \end{vmatrix} & \begin{vmatrix} a_{12} & a_{13} \\ a_{22} & a_{23} \end{vmatrix} \\ -\begin{vmatrix} a_{21} & a_{23} \\ a_{31} & a_{33} \end{vmatrix} & \begin{vmatrix} a_{11} & a_{13} \\ a_{31} & a_{33} \end{vmatrix} & -\begin{vmatrix} a_{11} & a_{13} \\ a_{21} & a_{23} \end{vmatrix} \\ \begin{vmatrix} a_{21} & a_{22} \\ a_{31} & a_{32} \end{vmatrix} & -\begin{vmatrix} a_{11} & a_{12} \\ a_{31} & a_{32} \end{vmatrix} & \begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix} \end{bmatrix}$$

- 屬性
  - **並非所有矩陣都有逆矩陣。**只有方陣（行數和列數相等的矩陣）才有逆矩陣，且方陣的行列式非零的時候才有。
  - 如果 A 是一個有逆矩陣 A^{-1} 的方陣，則該逆矩陣是唯一的。
  - 如果 A 和 B是可逆矩陣，則 AB也是可逆的，且$(AB)^{-1} = B^{-1}A^{-1}$。
  - 如果 A 是一個可逆矩陣，則 $(A^T)^{-1} = (A^{-1})^T$，即轉置的逆矩陣等於逆矩陣的轉置。
  - 如果 A 是一個可逆矩陣，則 $(kA)^{-1} = \frac{1}{k}A^{-1}$，其中 k是非零標量。
  - 如果 A 是一個可逆矩陣，則 $(A^{-1})^{-1} = A$，即逆矩陣的逆矩陣等於原始矩陣。

2：點積

- **這是什麼：**

  點積是一個函數。它將兩個向量作為輸入，並輸出一個數字（或者我們在線性代數中稱之為標量）。這個數字代表了這兩個向量指向彼此的程度。數字越大，代表這兩個向量的指向越接近。

  兩個向量 a 和 b 的點積（表示爲 $a \cdot b$）定義為它們個別對應的元素的乘積之和。換句話說，如果你有向量 a 和 b：

$$a = \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ ... \\ a_n \end{bmatrix}, b = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ ... \\ b_n \end{bmatrix}$$

  那麼它們的點積是：$a \cdot b = a1b1 + a2b2 + ... + anbn$

- **直覺理解：**

  除了數值定義外，點積還具有許多很好的幾何特性。但現在，你可以把點積視為「當我們要把兩個向量相乘時候所使用的一個工具」

- **爲什麼要學這個：**

在機器學習中，權重跟輸入資料總是由向量表示。當我們要計算輸入值跟權重相乘的時候，我們就會用點積來做這件事。

- **速查表：**
  - 點積用「$\cdot$」符號表示，例如 a 跟 b 的點積就是：$a \cdot b$
  - 點積同時也可以被理解成「第一個向量的轉置乘上第二個向量」，也就是 $a \cdot b = a^T b$

微積分概念

1：偏導數

- **這是什麼：**

  假設 f 是一個多變數函數，裏面有一個變數叫做 x。那麼「f 對 x 的偏導數」是另一個函數。該函數描述了在保持所有其他變數不變的情況下，f 相對於 x 的變化速率。
  就數學上的定義來看，對於函數 $f(x,y)$，其對 x 的偏導數表示為 $\frac{\partial f}{\partial x}$，定義為：

$$\frac{\partial f}{\partial x} = \lim_{\Delta x \to 0} \frac{f(x+\Delta x, y) - f(x,y)}{\Delta x}$$

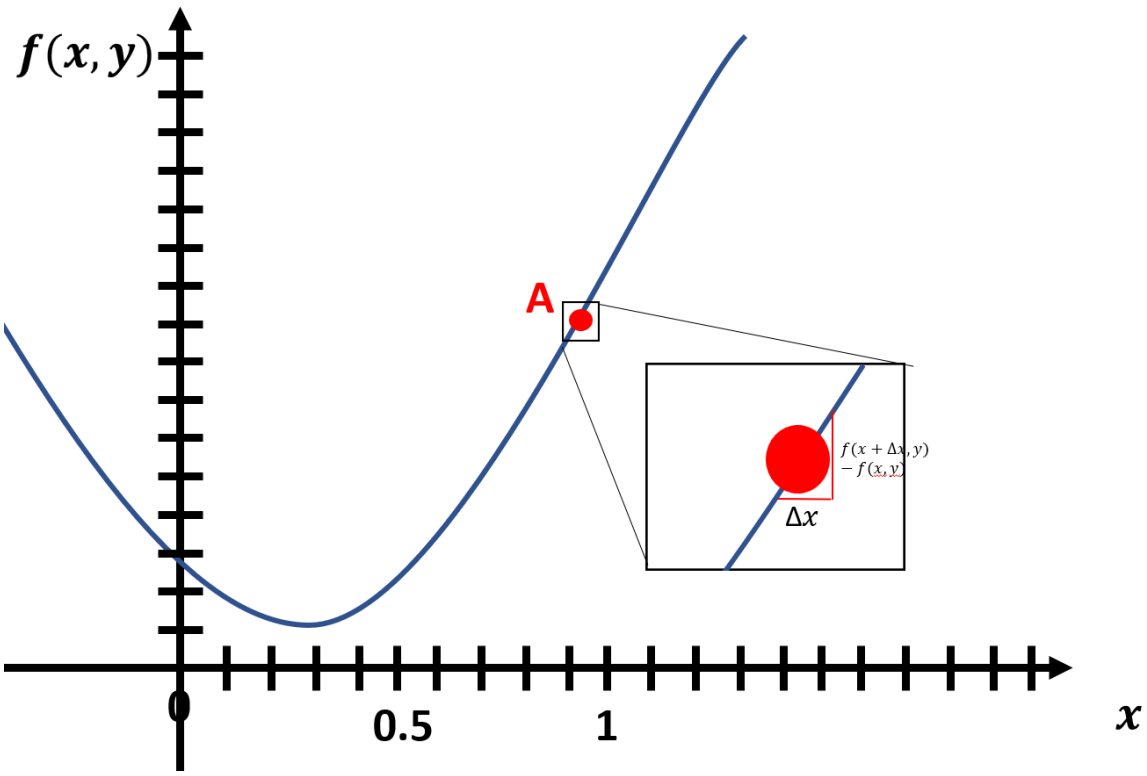
  導偏數也有很有趣的幾何意義。$\frac{\partial f}{\partial x}$ 取值在某一點的結果，會等於該點的切線斜率。這邊你看待會的圖像範例可能會比較明白一點。

- **直覺理解：**

  當你看到「f 對 x 的偏導數」，它的意思是「f 會在 x 改變一點點時發生多大的變化」。 f 是一個具有 x 作為其中一個變數的多變數函數。這個得出來的偏導數是一個以 x 作爲變數的函數。
  回到第三章駕駛的例子。「距離對時間的偏導數」是「汽車在極短時間內行駛的距離」。你稍微思考一下，你就會意識到「汽車在極短時間內行駛的距離」這就是在說「汽車在某一時刻的速度」，因爲速度的定義就是「在某段時間內行駛的距離」。
  從數學公式看這個概念會更加清楚。首先，回到偏導數的定義，$\lim_{\Delta x \to 0}$ 告訴你 Δx 非常接近0，這意味著 Δx 是 x 的極小變化。

  至於分子 $f(x+\Delta x, y) - f(x,y)$，則是代表了「（$x$增加一點點后 $f(x,y)$的值）-（x 保持不變，$f(x,y)$的值）」，這同時也代表了「$f(x,y)$在 x 變化一點點時變化了多少」。在我們的例子中，這是「汽車在極短時間內行駛的距離」。

  因爲分母 Δx 代表的是極短時間。所以，距離/時間 $=$ 速度！
  至於偏導數跟斜率的關聯，如果你把它畫出來，就可以很清楚看到：



斜率的定義就是Y軸的變動除以X軸的變動，這是高中就學過的。

在點A的偏導數，是X軸的方向移動 Δx，Y軸的方向移動 $f(x+\Delta x, y) - f(x, y)$。兩者相除的結果：$\frac{f\ (x+\Delta x, y)\ -f\ (x,\ y)}{\Delta x}$，就是偏導數的公式。如圖所示，在A點的斜率（嚴格來說應該是在A點跟 f 相切的直綫的斜率）是等於偏導數的。

請注意，儘管在圖片中 Δx 看起來已經非常小了，但實際上 Δx 要小得多，應該要小到看不見，因為它非常接近於0。

- **為什麼要學這個：**

  偏導數的概念在模型訓練的過程中扮演著非常重要的角色。在訓練過程中，偏導數可以告訴我們應該朝著哪個方向繼續調整模型以獲得更好的結果。在第5章的個案中，這個概念會變得更加清楚。

- **速查表：**

  - 規則：
    對於一個函數 $f(x, y, z)$，它對 x 的偏導數的規則是：
    - ax^n 變成 $n * ax^{n-1}$
    - 所有其他變量被視爲常數，且常數消失
  - 範例：
    對於函數 $f(x, y, z) = x^3 + 4y^2 + 5z + 18$，它對 x 的偏導數為：

$$\frac{\partial f(x, y, z)}{\partial x} = 3x^2$$

2：鏈鎖率

- **這是什麼**：

  微積分中的一條規則，可以幫助你輕鬆找到複合函數的導數。複合函數是由其他函數組成的函數，例如 $f(g(x))$。鏈鎖率告訴我們可以透過乘以 f 和 g 的導數來找到 $f(g(x))$ 的導數。

- **直覺理解**：

  其實，你仔細想想就會發現，所有函數都可以寫成複合函數的形式。因此鏈鎖率實際上只是一個工具，可以幫助你把很難解的微分問題拆成好解的小部分。當你碰到無法展開的函數時，鏈鎖率尤其強大。

- **為什麼要學這個**：

  在機器學習中，我們需要微分（求導數）的函數通常是一種稱為「損失函數」的函數。許多損失函數很難（或不可能）展開，因此鏈鎖率非常關鍵。

- **速查表**

  - 規則:

    假設 $y = f(u)$ 且 $u = g(x)$，則：

    $$\frac{dy}{dx} = \frac{dy}{du} \cdot \frac{du}{dx}$$

  - 範例（注意：嚴格來說，此函數不需要使用鏈鎖率，因為它可以輕鬆地展開和微分。但因爲這個函數比較好解釋，所以我以它爲例。用牛刀殺鷄的概念。）:

    假設有一函數： $f(x, y) = (2x + 3y)^2$

    我們想找到 f 對 x 的偏導數。使用鏈式法則，我們可以假設 $u = 2x + 3y$ ，然後將 f 對 x 的偏導數表示為：

    $$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial u} \cdot \frac{\partial u}{\partial x}$$

    要找到 f 對 u 的偏導數，我們可以簡單地對 f 進行 u 的微分：

    $$\frac{\partial f}{\partial u} = 2u = 2(2x + 3y)$$

    接下來，我們需要找到 u 對 x 的偏導數：

    $$\frac{\partial u}{\partial x} = 2$$

    現在我們將這些表達式代入鏈鎖率的公式中，就可以找到 f 對 x 的偏導數：

    $$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial u} \cdot \frac{\partial u}{\partial x} = 2(2x + 3y) \cdot 2$$

第5章：機器學習個案

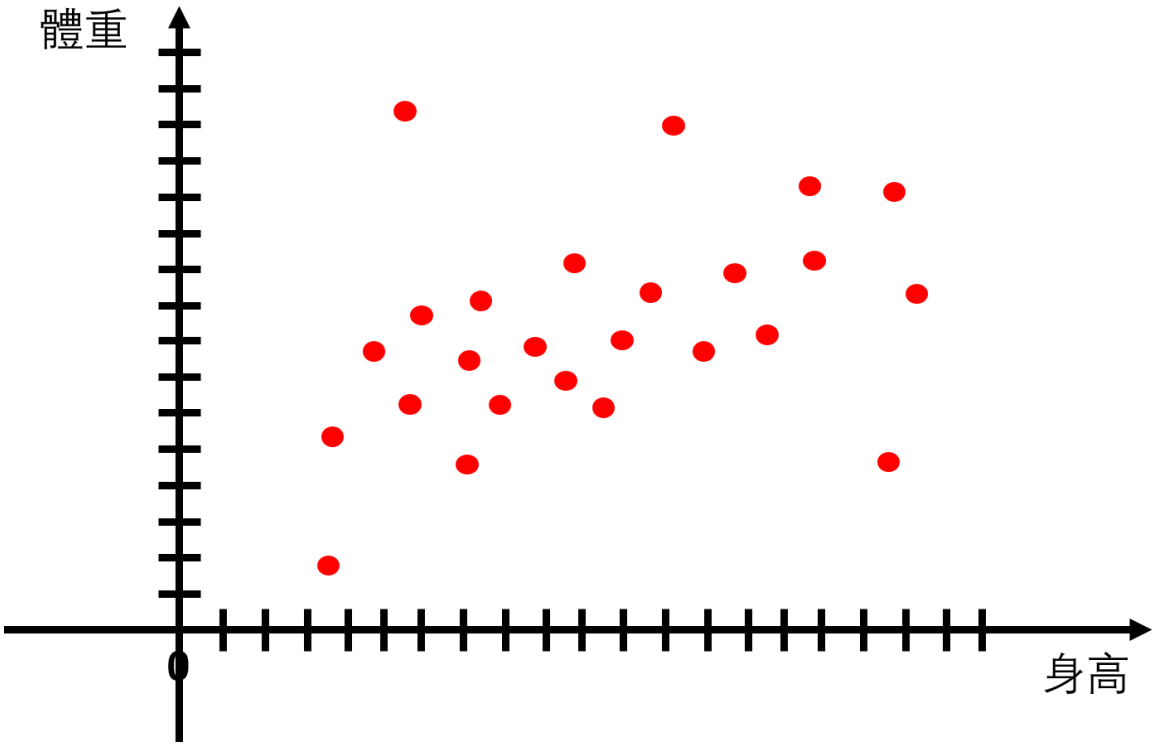
復習一下前面所說的，機器學習就是在尋找一個**能夠在給定正確輸入時生成所需輸出的函數**。你可能有聽說過一些演算法，包括神經網絡、線性回歸、隨機森林等等。這些都是構建這個函數的不同方法。

在這一章，我會帶你一起從零開始建構一個機器學習模型。我們會使用**線性回歸**作爲我們的演算法。

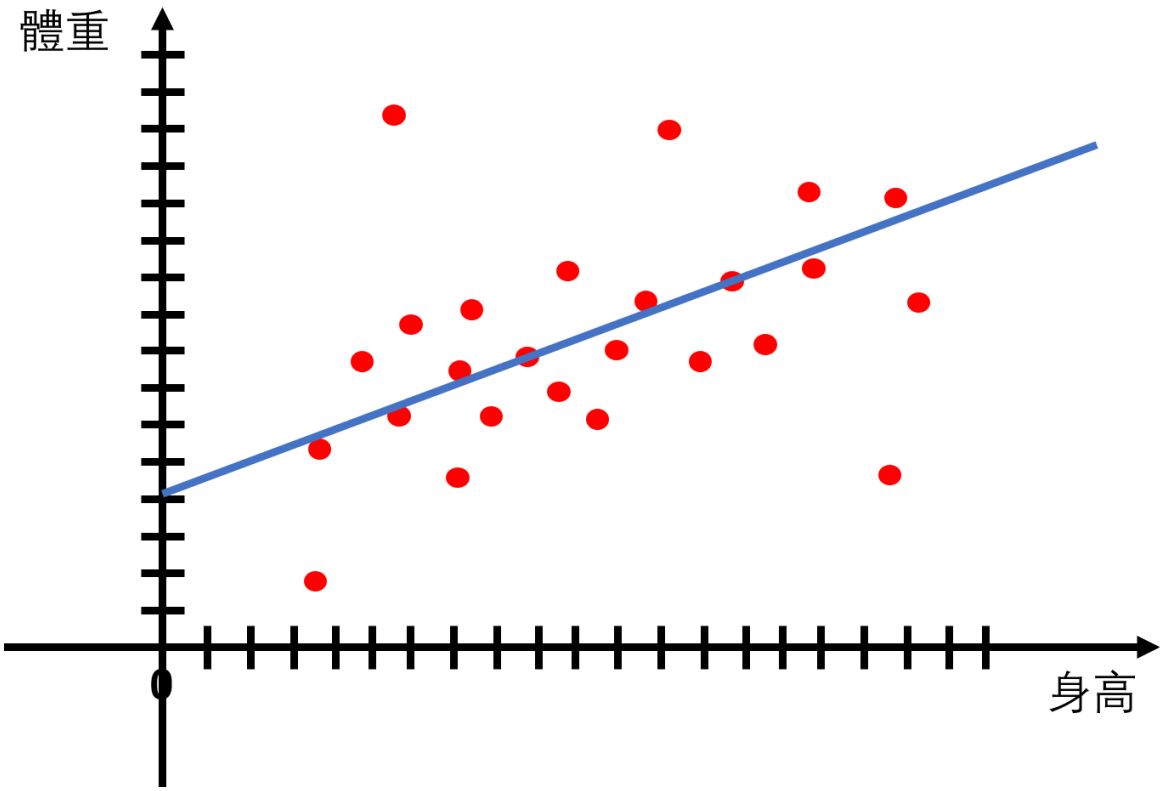
線性回歸是個基本又好用的機器學習模型。會選擇它是因爲它很單純好理解，而且同時也很强大（如果用在對的地方）。實務上也常常會用到這個模型，通常是拿來建一個基線（比較基準）。

什麼是線性回歸

生活中有許多事情是有線性關係的，像是我們第1章提到的「身高」和「體重」。身高越高的人，體重通常越重。當然也有例外，但是當你搜集很多人的資料的時候，你一定會看到一個線性關係。假設你把這些人的資料畫在一個2D的坐標中，你應該會看到類似下圖（每個紅點代表一個人）：



線性回歸這個演算法，就是「用一條線來解釋不同變數之間的關係」。我們會在所有資料中間畫出一條線，這條線必須是「跟每一筆訓練資料的距離總和是所有線裏面最小的」。這條線被稱作「最佳擬合線」，像是下圖：



這一條藍線就是我們的線性回歸模型。我們可以拿它來解釋變數之間的關係，以及預測新的資料。預測的方法很簡單：根據這條線，每個身高都有對應到一個體重。因此當我們知道一個人的身高，我們可以根據這條線大概猜說他的體重應該是多少。

來講一些專有名詞的部分。在統計的語言中，變數有「應變數」和「自變數」之分。「應變數」是我們最關心的變數，「自變數」則是會影響「應變數」的變數。

在機器學習中，我們稱應變數為「標籤」，而自變數則稱為「特徵」。如上圖所見，特徵跟標籤的關係是由一條線表示，而這一條線同時也是一個數學函數。該函數就是我們的模型。模型的訓練就是用資料推導出這個函數的過程。

如果上面的內容有些抽象，讓你看不太懂的話，別緊張！很快你就會看到實際的案例了。

個案背景

你可能已經註意到，我在第1章中提到的機器學習模型範例其實就是一個線性回歸模型。這個範例用看起來像這樣的函數來模擬身高和體重之間的關係（這個函數代表了前面圖中的那條線）：

$$w_1 X^1 + w_0 = y^{pred}$$

當找到最佳的 w_1 和 w_0 時，這個函數就代表了 X^1（身高，自變數）跟 y^{pred}（體重，應變數）之間的關係，也代表了「最佳擬合線」，并且可以拿來進行預測。但是這個範例有點太過簡單。實務上，你會有更多的特徵。也就是說，你會有更多身高以外的其他因素來幫助你更準確地預測一個人的體重。針對一些比較複雜的問題，可能會有10幾個，或甚至更多。但我也不想要讓這個案例太複雜，所以我們再添加一個就好——生理性別。

因此，我們想要找到的函數會長這樣：

$$w_1 X^1 + w_2 X^2 + w_0 = y^{pred}$$

來復習一下。w_1, w_2, w_0 是權重， 是我們要透過訓練找到的數字。

X^1, X^2 是特徵。他們分別代表了身高以及性別。

y^{pred} 是模型對於標簽，也就是 y^{actual}（體重），的預測值。我們希望這個預測值盡可能接近實際值（標簽），也就是說我們希望 y^{pred}盡可能接近 y^{actual}。它越接近，模型的預測性就越好。

X^1, X^2 是輸入，y^{pred} 是輸出，唯一的未知數是權重 w_1, w_2, w_0。

因此，讓我們重新定義一下我們的目標——找到最佳權重，來讓 y^{pred} 盡可能的接近 y^{actual}。

我們會用到的工具

整個第五章我會詳細地介紹機器學習的流程以及線性回歸的數學原理。這本書講的很多步驟在實務上你其實看不到，因爲Python（一個程式語言）有提供非常方便的工具（模組）來幫你解決數學的部分。一般來說，你要做的就是使用這些工具而已。

但是，瞭解模型背後的運作原理也是非常重要的。如果都在用別人設計好的黑盒子，想必你也不安心！

雖然程式語言不是這本書的重點，我還是有附上我所有的程式碼。接下來的每一個步驟都有相對應的程式碼可以看。當你閱讀時，你可以在旁邊打開我的程式碼進行比較。你也可以自己跑跑看，或甚至是修改我的程式碼來玩玩看。

機器學習的流程

我們整個專案的過程有四個步驟：資料準備和處理 → 模型訓練 → 模型評估 → 模型部署。實務上的流程也是這樣。讓我們直接開始吧！

步驟1: 資料準備與處理

我們現在要建一個機器學習模型。這個模型要可以從一個人的身高和性別正確預測出這個人的體重。我們決定使用線性回歸作爲演算法來完成這個模型。

我們去了一家醫院，要到了如下經過個資處理的數據集：（假設總共有1000行，只顯示前4行）

身高（公尺）	性別	体重（公斤）
1.52	女性	42
1.82	男性	80
1.78	男性	73
1.67	女性	56
…	…	…

這個數據集可能是個excel檔案或是csv檔案 。我們會先將這個檔案讀入我們的程式語言（通常是Python），以便進行以下所有步驟。假設現在已經完成了。 （在我的程式中，數據是隨機生成的不是讀入的。我並沒有真的去醫院要數據。）

由於我們之後是要找到一個數學函數，而資料中的「男性」跟「女性」很明顯的不是數字，不可能被放到數學函數中，所以我們需要將所有這種所謂的「分類數據」轉換為數字表示。因此，我們首先將「男性」轉換為「0」，將「女性」轉換為「1」。

身高（公尺）	性別	体重（公斤）
1.52	1	42
1.82	0	80

1.78	0	73
1.67	1	56
…	…	…

接下來，我們將所有數據分成**訓練數據**和**測試數據**兩份。訓練數據是拿來訓練模型使用的，測試數據則是拿來檢視模型成效。訓練數據就像是很多份的學測歷屆考題，測試數據則像是正式學測。所以這邊的切分必須要很公正才行，我們不希望歷屆跟正式學測題目差太多，也不希望有重複的題目出現。

我們就隨機抽80%的數據（800行）作為訓練數據，剩下的20%的數據（200行）作為測試數據。

用矩陣跟向量表示資料

在我們的程式碼裏面，資料是以矩陣的形式存在我們的電腦裏。 讓我們來看看這些矩陣長什麼樣。

首先，記得在第 2 章中，我說過線性方程組可以用矩陣和向量來表示嗎？

另外，回想一下，我們的目標是在下面這個線性方程組中找到最佳權重（w_1, w_2, w_0）：

$$\begin{cases} w_1 X_1^1 + w_2 X_1^2 + w_0 = y_1^{pred} \\ w_1 X_2^1 + w_2 X_2^2 + w_0 = y_2^{pred} \\ w_1 X_3^1 + w_2 X_3^2 + w_0 = y_3^{pred} \\ ... \\ w_1 X_{800}^1 + w_2 X_{800}^2 + w_0 = y_{800}^{pred} \end{cases}$$

（只有800行是訓練數據）

X_1^1是第一筆數據的第一個特徵（第一個人的身高），X_2^1是第二筆數據的第一個特徵（第二個人的身高），以此類推。

X_1^2是第一筆數據的第二個特徵（第一個人的性別），X_2^2是第二筆數據的第二個特徵（第二個人的性別），以此類推。

如果使用矩陣表示資料，我們可以只用一個方程式來表示這整個線性方程組：

$$Xw = y^{pred}$$

X是我們所有的訓練數據堆成的矩陣。注意，這邊只有訓練數據的特徵（身高，性別）沒有標籤（體重，y^{actual}）。w 是我們所有要找到的權重堆成的一個向量。y^{pred} 是所有模型的預測結果堆成的一個向量。

現在看可能還有點抽象，我們來一步一步把線性方程組換成在這個矩陣跟向量組成的方程式。

首先，我們需要稍微修改線性方程組的方程式。 我們將恆等於 1 的常數 X^0 分配給 w_0，讓每個 w 都對應到一個 X：

$$w_1 X_i^1 + w_2 X_i^2 + w_0 X^0 = y_i^{pred}$$

現在我們可以將所有 X_i^1, X_i^2, X^0表示為矩陣 X。

$$X = \begin{bmatrix} X_1^1 & X_1^2 & X^0 \\ X_2^1 & X_2^2 & X^0 \\ X_3^1 & X_3^2 & X^0 \\ X_4^1 & X_4^2 & X^0 \\ ... & ... & ... \end{bmatrix}$$

權重的向量則是長這樣：

$$w = \begin{bmatrix} w_1 \\ w_2 \\ w_3 \end{bmatrix}$$

並且，根據矩陣向量乘法，我們現在可以把：

$w_1 X_i^1 + w_2 X_i^2 + w_0 X^0 = y_i^{pred}$

寫成：

$Xw = y^{pred}$

y^{pred} 是 Xw 的乘積，看起來像這樣：

$$y^{pred} = \begin{bmatrix} y_1^{pred} \\ y_2^{pred} \\ y_3^{pred} \\ y_4^{pred} \\ \cdots \end{bmatrix}$$

舉例來說：

$y_1^{pred} = X_1^1 w_1 + X_1^2 w_2 + X^0 w_0$

$y_2^{pred} = X_2^1 w_1 + X_2^2 w_2 + X^0 w_0,$

$y_3^{pred} = X_3^1 w_1 + X_3^2 w_2 + X^0 w_0$

以此類推。

看到這裏你想必已經看代數看的很煩了。讓我們來看看我們的實際數據以矩陣形式展現是什麼樣子。

首先，我們的特徵矩陣 X（人的身高、性別和一個常數），是一個 800x3 的矩陣：

$$X = \begin{bmatrix} 1.53 & 1 & 1 \\ 1.92 & 0 & 1 \\ 1.76 & 0 & 1 \\ 1.61 & 1 & 1 \\ \cdots & \cdots & \cdots \end{bmatrix}$$

而我們的標籤（人們的實際體重。我們稱之為 y）：

$$y^{actual} = y = \begin{bmatrix} 43 \\ 89 \\ 58 \\ 62 \\ \cdots \end{bmatrix}$$

注意，標籤是叫做 y^{actual}，而不是 y^{pred}。y^{pred} 是模型的預測結果（預測出的體重），不是實際的體重。

我們現在的目標是找出這個列向量 w 的最佳值：

$$w = \begin{bmatrix} w_1 \\ w_2 \\ w_0 \end{bmatrix}$$

之後每次我提到 w 時，我在指的就是這個列向量。

好的，現在我們已經準備好要訓練模型了！

步驟2: 訓練模型

損失函數

要找到最佳的 w，我們首先需要找到一種方法來衡量 y^{pred} 與 y^{actual} 的接近程度。 在機器學習中，衡量這一點的函數被稱為「損失函數」。 當損失函數最小化時，即當 y^{pred} 最接近 y^{actual} 時，這時候的w就是我們要找的最佳 w。「損失函數」也稱作「成本函數」。

有很多種不同的損失函數可以選擇，每個都有自己的屬性和偏差。 線性回歸最常見的損失函數是均方誤差 (Mean Squared Error，簡稱MSE)。

MSE 測量 y^{pred} 和 y^{actual} 之間的平均距離。計算方法如下：

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i^{pred} - y_i^{actual})^2$$

其中 n 是樣本數，在我們的例子中是 800（所有訓練數據）。

$\sum_{i=1}^{n}(y_i^{pred} - y_i^{actual})^2$ 是每對 y^{pred} 和 y^{actual} 之間所有平方差的總和。$\frac{1}{n}$ 是分母，取平均值的意思。所以這個損失函數才會叫做「均方誤差」。

會取平方一部分是因爲有取絕對值的效果，另一部分也是因爲這麼做可以會給差異大的項更大的權重。這可以是個優點也可以是個缺點，我們這邊先不深入討論。

按照機器學習慣例，我們通常會將 MSE 除以 2 來讓計算更簡單一點。待會你就會明白為什麼了。 因此，我們的損失函數現在是：

$$MSE = \frac{1}{2n}\sum_{i=1}^{n}(y_i^{pred} - y_i^{actual})^2$$

定義損失函數後，我們的目標變得非常量化：找到使上述公式最小化的 w。（復習：y_i^{pred} 是由 w 跟 X 算出來的）

要做到這點，有幾種不同的方法。機器學習中最常用的方法是「梯度下降」。這也是我們接下來會使用的方法。

梯度下降

梯度下降在做的事情，簡單來說，就是一次一次地不斷調整 w 的值，每一次調整都讓損失函數（MSE）變小，直到MSE無法再變小爲止。

- 步驟1：初始化 w
- 步驟2：決定模型應該學習多快
- 步驟3：看看我們應該朝哪個方向調整 w 才能使損失函數變小
- 步驟4：調整 w
- 步驟5：重複步驟3~4，直到損失函數最小化

讓我們一步一步來！

梯度下降步驟1：初始化 w

一開始我們必須給 w 一個起始值，然後從該起始值開始慢慢調整。步驟1就是初始化這個起始值。

起始值會影響模型訓練的速度，也會影響模型的成效，但是在一開始我們通常沒什麼綫索可以告訴我們哪個起始值才是最好的，所以在實務上，大家通常只使用平均數為零且標準差小的高斯分佈（常態分佈）來初始化所有權重。

在我們的個案中，我們只有三個參數。 所以為了方便起見，我們就把所有的權重初始化為零：

$$w = \begin{bmatrix} w_1 \\ w_2 \\ w_0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

梯度下降步驟2：決定模型應該學多快

在這邊我們要決定一個很關鍵的數字：「學習率」。這個數字控制了模型的權重在每次迭代中的調整幅度（一次迭代指的就是一次步驟3~4）。

「學習率」通常介於 0 和 1 之間，例如：0.01。如果這個數字太大，模型會過度調整權重，有可能讓我們離目標越來越遠。如果數字太小，模型達到目標的時間就會加長。因此，設定一個適合的學習率是至關重要的。

按照機器學習的慣例，我們將學習率表示為 α。 在我們的個案中，我們將設 $\alpha = 0.01$。

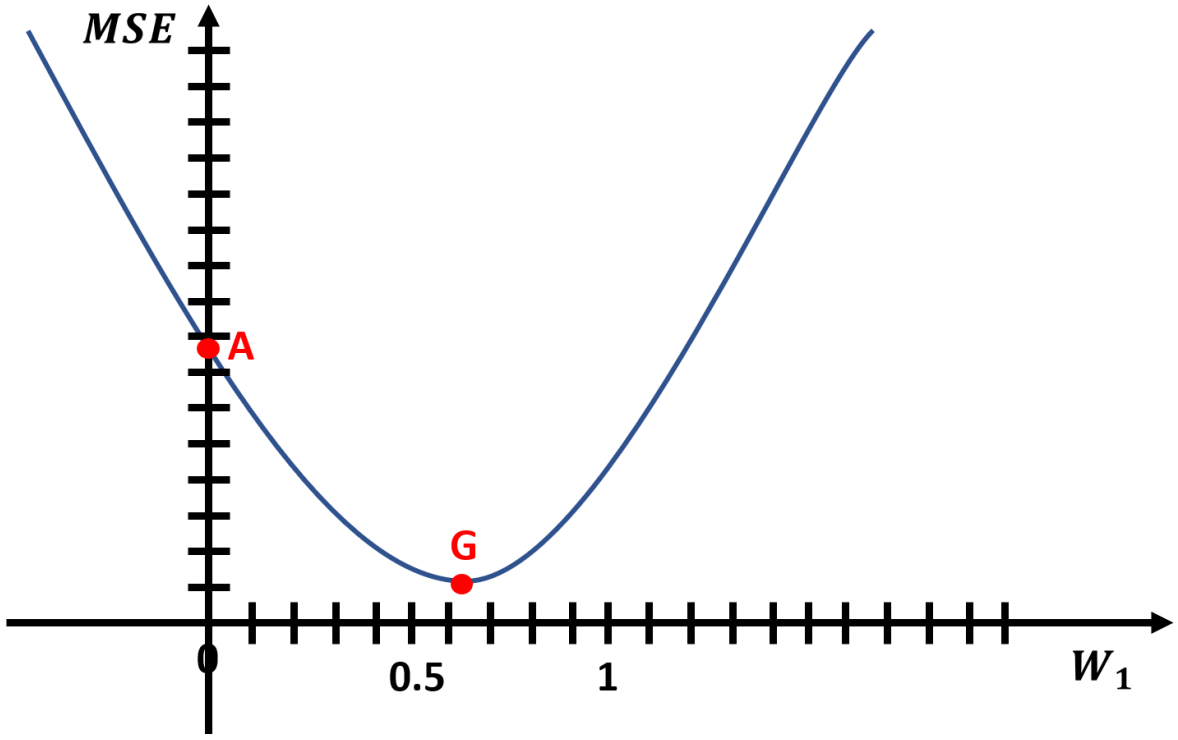
> 💬 學習率是一個「超參數」。你可以把「超參數」想成一個機器學習模型的基本設定。

梯度下降步驟3：決定調整權重的方向

在這邊我們要計算「梯度」。梯度是一個數字，代表了我們應該向哪個方向、調整多少 w。 我們會為每個權重（w_1, w_2, w_0）計算出它們自己的梯度。每個權重的梯度，就是「損失函數對它的偏導數」。

舉例來說，w_1 的梯度就是MSE對 w_1 的偏導數。如同第四章提到的，這個偏導數的意義代表了「當w_1 變動一點點的時候，MSE會變動多少」。你如果忘了就回去復習一下吧！

為了更清楚地說明這一點，讓我們先選擇一個權重 w_1。保持其他權重不變的情況下，假設我們得出 w_1 和MSE之間的關係如下（下圖為一個函數：$MSE(w_1)$）：



Graph 1

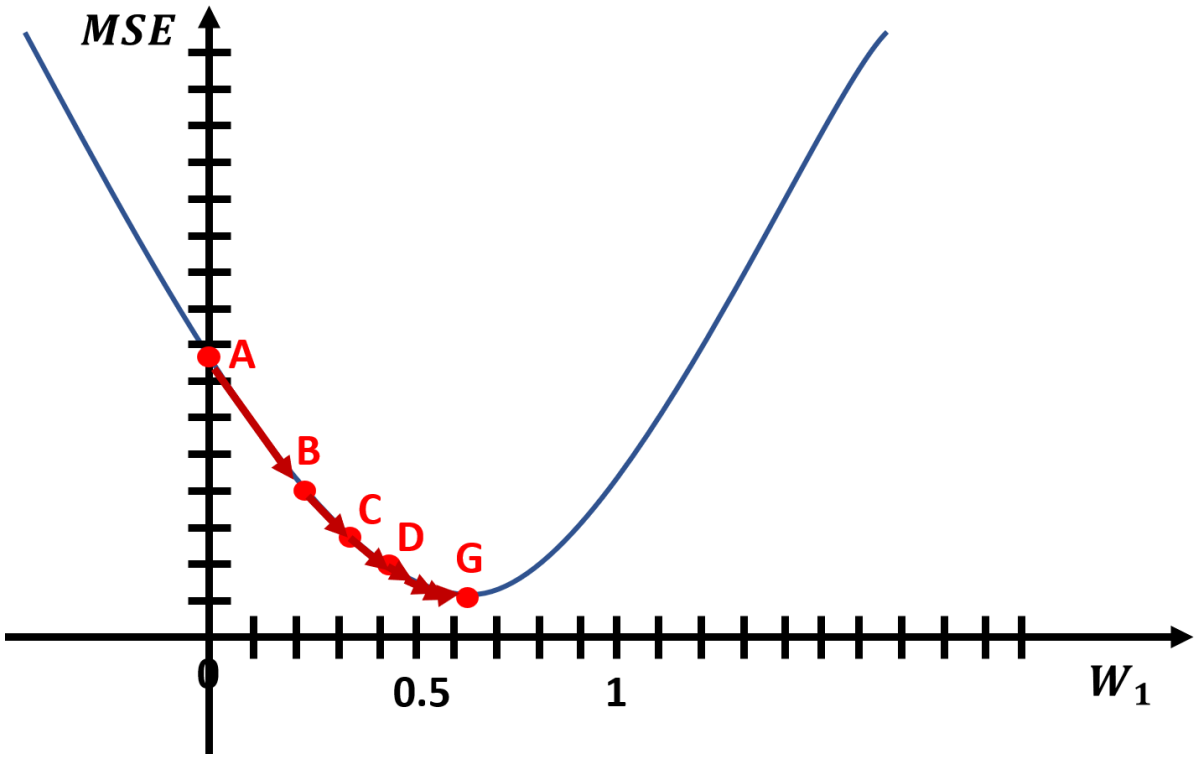
在梯度下降步驟1中，我們將 w_1 初始化為 0，因此目前我們處於 A 點。但是，你可以看到 MSE 在 G 點才是最低的，那時候的 w_1 約為 0.63。這也是我們訓練的最終目標：讓$w_1 = 0.63$。

但是在現實世界，我們沒法直接得知這個函數圖，所以我們當然無法立刻得知G點時 w_1 的值。

那我們怎麼透過梯度下降的方法走到那裏？

> 💬 其實有一種方法可以不透過梯度下降直接計算G點，稍後我會解釋。但是，這個方法僅限於線性回歸。大多數演算法都比較複雜，只能使用梯度下降的方式，不能這樣作弊。

梯度下降的方法是先將點 A 移到右側到點 B（見下圖），然後是點 C，然後是點 D，依此類推。每次移動的距離都會縮小，當我們到達 G 點時，移動距離會變成零，也就是停住不動了的意思。那時我們就知道權重已被最優化，可以停止了。



Graph 2

那要怎麼從 A 走到 B 點呢？

我們將首先計算點 A 的梯度。我們取 MSE 對 w_1 的偏導數：

$$\frac{\partial MSE}{\partial w_1} = \frac{\partial}{\partial w_1}\left[\frac{1}{2n}\sum_{i=1}^{n}(y_i^{pred} - y_i^{actual})^2\right]$$

因為 $w_1 X_i^1 + w_2 X_i^2 + w_0 X^0 = y_i^{pred}$，我們可以將 y_i^{pred} 替換，得到：

$$= \frac{\partial}{\partial w_1}\left[\frac{1}{2n}\sum_{i=1}^{n}(w_1 X_i^1 + w_2 X_i^2 + w_0 X^0 - y_i^{actual})^2\right]$$

接著要使用鏈鎖率了（可以復習一下第四章），我們先令 $u = w_1 X_i^1 + w_2 X_i^2 + w_0 X^0 - y_i^{actual}$，替換進去之後我們得到：

$$= \frac{\partial}{\partial w_1}\left[\frac{1}{2n}\sum_{i=1}^{n} u^2\right]$$

現在的MSE可以表示成 u 的函數：$MSE(u) = \left[\frac{1}{2n}\sum_{i=1}^{n} u^2\right]$，并且替換進去。根據鏈鎖率：

$$\frac{\partial MSE(u)}{\partial w_1} = \frac{\partial MSE(u)}{\partial u}\frac{\partial u}{\partial w_1}$$

這樣就可以微分了！微分後展開，我們得到：

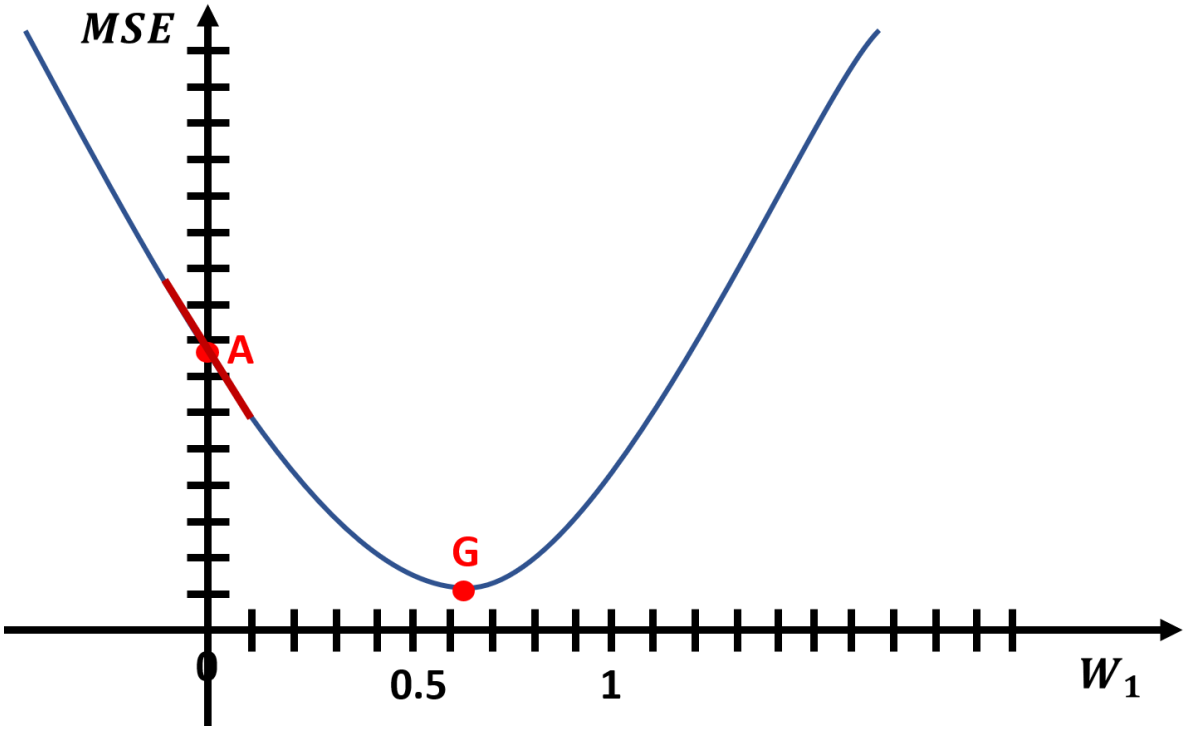
$$= \frac{1}{2n}\sum_{i=1}^{n} 2u * X_i^1$$

展開 u 并且消掉 2 後（我們那時候多放的2！），我們得到：

$$= \frac{1}{n}\sum_{i=1}^{n}(w_1 X_i^1 + w_2 X_i^2 + w_0 X^0 - y_i^{actual})X_i^1$$

上面這行就是我們計算 w_1 梯度的公式。如果把 A 點的數字代入，即 w_1, w_2, w_0 都等於 0，並代入 $X_i^1, X_i^2, y_i^{actual}$ 的所有800行數據，你就會得到 w_1 在A點的梯度。

我知道這些數學看起來很醜陋，但從圖形上看， w_1 在A點的梯度有一個美麗的含義：它等於 A 點的斜率！（記得嗎？我們在偏導數部分討論過這個）



Graph 3

這個梯度的數字同時也表示當我們稍微改變 w_1 時，MSE 會往什麼方向（負或正）增加（或減少）多少。這就是我們步驟3想要求得的！

梯度下降步驟4：調整權重

現在我們將根據梯度調整 w。我們將 w 調整為 $w - \alpha * gradient$，其中 α 是學習率。現在你知道學習率爲什麼可以控制模型學習的快慢了吧！

回到我們的例子，w_1 的調整後的值是 $w_1 - \alpha * gradient$，即：

$$= w_1 - \alpha\frac{1}{n}\sum_{i=1}^{n}(w_1 X_i^1 + w_2 X_i^2 + w_0 X^0 - y_i^{actual})X_i^1$$

如果要實際把這個數字算出來，這本書大概會有幾百頁。但我們知道，這些醜陋的數學 $\frac{1}{n}\sum_{i=1}^{n}(w_1 X_i^1 + w_2 X_i^2 + w_0 X^0 - y_i^{actual})X_i^1$ 等於 A 點的斜率，所以我們大概知道它的值在-2左右。（回想一下在高中時，你學過到從左上角切到右下角時斜率是負的，斜率的值則是等於y軸的變動除以x軸的變動。）

此外，我們設定的學習率為 0.01。 因此，我們的新 w_1 現在是 $w_1 - \alpha * gradient$ 即 $0 - 0.01(-2) = 0.02$。

換句話說，在第一次迭代中，我們將 w_1 從 0 調整為 0.02。太好了，我們現在離 G 點（0.63）更近了一步！ 我們會為 w_2 和 w_0 做同樣的事情。

但是請記住，我們的數據現在是用矩陣表示的。 所以：

$$\begin{cases} w_1^{new} = w_1 - \alpha \frac{1}{n} \sum_{i=1}^{n} (w_1 X_i^1 + w_2 X_i^2 + w_0 X^0 - y_i^{actual}) X_i^1 \\ w_2^{new} = w_2 - \alpha \frac{1}{n} \sum_{i=1}^{n} (w_1 X_i^1 + w_2 X_i^2 + w_0 X^0 - y_i^{actual}) X_i^2 \\ w_0^{new} = w_0 - \alpha \frac{1}{n} \sum_{i=1}^{n} (w_1 X_i^1 + w_2 X_i^2 + w_0 X^0 - y_i^{actual}) X^0 \end{cases}$$

可以寫成：

$$w^{new} = \begin{bmatrix} w_1^{new} \\ w_2^{new} \\ w_0^{new} \end{bmatrix} = w - \alpha \frac{1}{n} X^T \cdot (Xw - y)$$

如果您忘記了 w, X, y 代表什麼，可以回到前幾頁復習。

此外，X^T 是 X 的轉置矩陣。

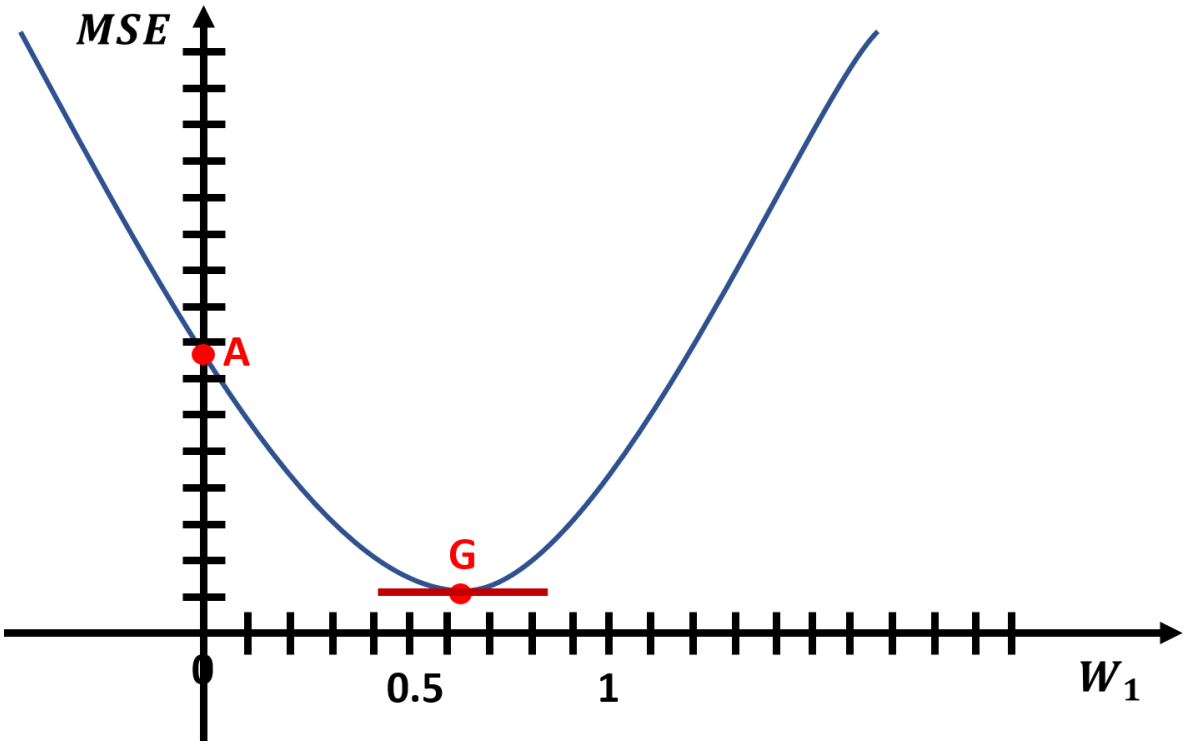
在我的程式裏面，梯度就是用上面這個公式計算的。

一次迭代的過程，就是計算 $w - \alpha \frac{1}{n} X^T \cdot (Xw - y)$一次。

梯度下降步驟5：重複步驟3~4，直到損失函數最小化

在對所有權重進行一輪調整後，我們完成了一次迭代。然後我們將使用 w 的新值繼續第二次迭代：計算它們目前的梯度（使用 $w - \alpha \frac{1}{n} X^T \cdot (Xw - y)$ ），調整它們的值，然後重複。

經過每次迭代，權重會越來越接近最佳值。當我們調整到G點的時候，就如前面所說，G 點的梯度，即G點的斜率，為零。（與 x 軸水平的斜率為零）



因此，當 w_1 到達點 G 時，基本上不再需要調整。 $-\alpha * gradient = -0.010 = 0$。 權重最佳化就完成了！

梯度下降小結

經過一定次數的迭代後，所有的權重都會被最佳化，我們模型的訓練也就結束了。我們也就有一個可以根據身高和性別預測一個人的體重的模型了！

回顧一下，我們的 w 一開始為 $[0, 0, 0]$，經過每次迭代的調整，會變成最佳值。我的程式有實際把這個過程寫出來，最後的結果是：

經過了200,000次迭代，w 從 $[0, 0, 0]$ 變成$[64.31418, -5.37382, -42.46981]$。（我的程式）

我們的模型寫出來會像這樣：

$$64.31418X_i^1 - 5.37382X_i^2 - 42.46981 = y_i^{pred}$$

如果你在上面這個數學公式的 X_i^1（以公尺為單位）中插入一個人的身高，在 X_i^2 中插入性別（女性為 1，男性為 0），減去 43.00115，你將得到大約這個人的體重（y^{pred}，單位為公斤）。

比如說，你想用我們的模型預測一個身高185的男生體重多少，你就可以做以下計算：

$64.31418 * 1.85 - 5.37382 * 0 - 42.46981 = 76.51$

所以這個男生大概是76.5公斤。

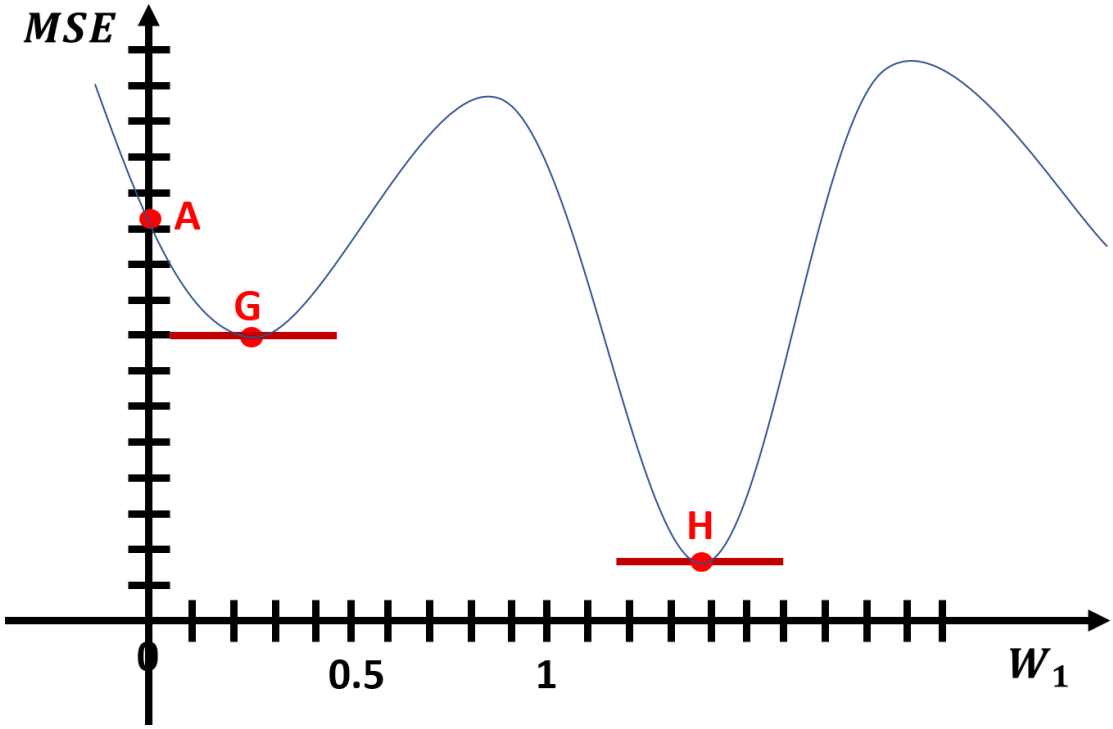
你可以看到，權重解釋了特徵跟標籤之間的關係。用來乘上性別的權重，-5.35758，代表了對於相同身高的男性和女性，女性往往會輕 5.35758 公斤。

前面說過，梯度下降是大多數機器學習演算法的學習方式，包括最熱門的「神經網絡」。但是，梯度下降這並不是訓練機器學習模型的唯一方式，儘管在大多數情況下梯度下降是最好且唯一可行的方式。

其實，要訓練線性回歸的模型，有一種方法可以作弊——Moore-Penrose偽逆矩陣。本章的下一部分會詳細介紹這個方法。這一部分是補充教材，因爲它是一個只適用於線性回歸的方法，而且用到的數學有點超出範圍，所以你想要的話可以直接跳過。

有一點要注意的是，梯度下降并不會總是給我們最好的結果。這是因為每個演算法的損失函數都長得不一樣。

假設你今天要優化的模型的損失函數長這樣：



當你從 A 點開始執行梯度下降時，你會到達G點。但是 H 點其實有更低的MSE！在機器學習中，我們稱 G 點為局部最小值，H 點為全域最小值。全域最小值才是最佳答案。

講到這裏，你應該也知道爲什麼一開始的權重起始值會影響模型成效了！

不過對於線性回歸，我們不太需要擔心我們只找到局部最小值的問題，因爲線性回歸的損失函數總是 U 形（或碗形），并不是向上圖一樣凹凹凸凸的。這代表損失函數只會有一個最小值（全域最小值），這意味著經過夠多的步驟後，權重幾乎總是會收斂到最佳點。

Moore Penrose 偽逆矩陣（補充）

儘管對線性回歸來說，梯度下降幾乎總是可以收斂到全域最小值，但這並不是訓練線性回歸模型的最理想方法。我之所以先提梯度下降，主要是因爲梯度下降是機器學習裏面使用最廣泛且最重要的學習方法。

在實務上，你會使用Python（一種程式語言）的套件來幫你訓練線性回歸的機器學習模型。而這些套件背後的算法，并不是梯度下降，而是Moore-Penrose 偽逆矩陣。

與梯度下降法相比，Moore-Penrose 偽逆矩陣通常更快、更穩定，並且可以準確地到達全域最小值。再次提醒，此方法僅適用於某些類型的回歸類型算法哦！

讓我們來看看 Moore-Penrose 偽逆矩陣是如何計算的吧！

如果你還記得，y^{pred} 和 y^{actual} 分別可以寫成：

$$y^{pred} = Xw$$
$$y^{actual} = y$$

我們把這兩項帶到MSE的公式裏面，我們會得到（復習一下第4章點積的部分）：

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i^{pred} - y_i^{actual})^2 = \frac{1}{n}(Xw-y)\cdot(Xw-y)$$
$$= \frac{1}{n}(Xw-y)^T(Xw-y)$$

因爲我們的目的是要找到可以讓 $\frac{1}{n}(Xw-y)^T(Xw-y)$ 最小的 w 的值，所以我們直接令$\frac{1}{n}(Xw-y)^T(Xw-y)$ 對 w 的偏導數為零。

會這麼做是因為，根據我們從梯度下降那部分得到的結論，MSE 對 w 的偏導數為零的時候，MSE會是最小的，而且 w 會是最佳值。你可能會問，但是現在我們的MSE是一個矩陣的函數， w 則是一個矩陣，我們要怎麼得到矩陣函數對一個矩陣的導數？

這涉及到一些「矩陣微積分」。矩陣微積分基本上只是普通微積分與矩陣運算的結合。這一部分不是太重要，想瞭解更多的可以閱讀這篇文章。

回到我們的算式。MSE 對 w 的導數如下所示。 我們令它為零：

$$\nabla MSE(w) = \frac{2}{n}X^T(Xw-y) = 0$$

現在讓我們嘗試求解 w。先把算式乘開來：

$$\frac{2}{n}X^TXw - \frac{2}{n}X^Ty = 0$$

然後把它簡化：

$$X^TXw = X^Ty$$

如果 X^TX 是可逆的，我們可以將兩邊都乘以 (X^TX) 的逆矩陣，即 $(X^TX)^{-1}$。我們得到：

$$(X^TX)^{-1}(X^TX)w = (X^TX)^{-1}X^Ty$$

$(X^TX)^{-1}(X^TX)$ 等於單位矩陣，可以忽略（單位矩陣乘上仍和矩陣都會還是原本的矩陣。可以復習第4章）。所以也就等於：

$$w = (X^TX)^{-1}X^Ty$$

現在剩下的就是做一些簡單的矩陣乘法，然後我們就得到最佳的 w 了！

但是，在第4章有講到，并不是所有矩陣都是可逆的。**如果 X^TX 不可逆，**我們就無法使用一般的逆矩陣解決問題。這時候，就要計算偽逆矩陣。

對於所有的 X^TX，我們都可以找到一個偽逆矩陣。

X^TX 的偽逆矩陣，是用 $(X^TX)^+$ 表示。偽逆矩陣與一般的逆矩陣具有非常相似的性質：

1. $(X^TX)(X^TX)^+(X^TX) = X^TX$
2. $(X^TX)^+(X^TX)(X^TX)^+ = (X^TX)^+$
3. $(X^TX)(X^TX)^+$ 和 $(X^TX)^+(X^TX)$ 是對稱的

偽逆矩陣跟一般的逆矩陣的差別，就在於逆矩陣乘上原本的矩陣是得到一個乾净的單位矩陣，而偽逆矩陣乘上原本的矩陣則是得到一個最接近單位矩陣的矩陣。計算偽逆矩陣的方法比較複雜，涉及到奇異值分解（SVD）的概念，這邊就不深究。

回到算式。我們得到：

$$w = (X^T X)^+ X^T y$$

接下來就是做一些簡單的矩陣乘法，然後我們就得到最佳的 w 了！

Python中很受歡迎的機器學習套件 Scikit-learn 就是使用這種方法來優化（訓練）線性回歸模型。在我的程式中，我有用三種方法：梯度下降，Moore-Penrose 偽逆矩陣和 Scikit-learn 來訓練線性回歸模型，用的都是同樣的資料。你如果比較我程式裏面的結果，你會發現Moore-Penrose 偽逆矩陣和 Scikit-learn這兩種方法得到一摸一樣的結果，而梯度下降則得到很相似但是有些微不同的結果。

步驟3: 評估模型成效

建出了第一版的模型之後，我們的下一步就是要看看這個模型表現有多好。

還記得一開始我們把所有1000筆的數據拆成訓練數據（800筆）跟測試數據（200筆）嗎？現在就是測試數據派上用場的時候了。

我們將測試數據帶進我們的模型裏面，然後計算 MSE。

要做到這件事，一樣也只需要用到非常簡單的矩陣乘法：

$$X^{test} w = y^{pred}$$

然後我們計算測試數據的MSE：

$$MSE = \frac{1}{200} (y^{pred} - y^{actual})^2$$

這時候你會得到一個數字。根據我的程式算出來的結果，MSE是 92.66。

這個數字代表了模型預測結果跟真實結果的平均距離，但是這不是很直覺的理解方法。這個數字對你來說可能根本毫無意義，因為你沒有任何東西可以拿來跟這個數字進行比較。

通常，人們會將這個數字與他們為同一個數據集建的其他模型進行比較。這時候，就能很明顯的看出哪一個模型的MSE比較低，也就是比較好的模型。但是對於我們來說，真的沒有什麼可以比的，因爲我們只有建一個模型。

但沒關係！還有一個評估模型的方法，就是直接拿我們的模型來做預測然後一筆一筆看它預測的結果好不好。

你可以做出像是下面這樣的一個表格（下圖是我的程式跑出來的結果），並查看模型預測的接近程度。看起來大多數預測都離得不會太遠。

Height	Gender	Predicted_Weight	Actual_weight
1.7251	1	63.11395852	76.80412353
1.855476	0	76.89618844	79.06928953
1.776005	1	66.4033411	66.61281951
1.70835	0	67.38920339	81.93468537
1.722318	0	68.29176391	68.04010644
1.792391	1	67.46215775	72.41702759
1.702654	1	61.66352736	52.05790726
1.698997	0	66.78482723	55.64994066
1.889252	0	79.07873319	75.60535426
1.803759	1	68.19673879	84.80996912
1.721879	1	62.90584733	59.74983531
1.610622	1	55.71663493	54.68776023
1.548812	0	57.08015101	63.77342329
1.80897	0	73.89109522	69.88286249

當然，如果你對這個結果不滿意，你可以收集更多可以解釋體重的特徵（例如：人的國籍、年齡、飲食習慣等），或者嘗試其他演算法。前者通常更有幫助。

步驟4: 部署模型

當你對模型的結果感到滿意，你就可以開始使用它來解決你的問題。如果你正在建一個個人健康APP，並且需要用這個模型來幫助填充缺失的體重數據，你可以將這個模型部署到你的後端程式碼，讓它對資料庫進行更新。

第6章：下一步

耶！結束了！恭喜你讀完了這本又臭又長的機器學習電子書。現在你學會了如何從頭開始構建線性回歸的機器學習模型，並了解其背後的所有數學原理。

那，接下來你該學什麼呢？

如果你沒有寫程式的經驗，我建議你去學一下python。切記不要學得太深！如果你學到可以能理解我 80% 的程式，你就可以停下來了。你不需要完整的上完一整堂 Python 課程。要知道，python的應用很廣泛，很容易就會不小心學的太偏。

學會了程式的基礎知識後，我建議你開始自己建一些機器學習模型！找到一個你有熱情的問題，並嘗試用線性回歸模型來解決它。在整個過程中，你會發現在實際的機器學習專案中有很多細節我還沒有提到（比如數據清洗，探索性數據分析）。

你還會發現，雖然線性回歸可以作為一個很好的基準點，但還有更強大的演算法可以更好地解決你的問題。甚至，有些你想要解決的問題可能根本無法通過線性回歸來解決，需要截然不同的東西。如果你發現自己在思考這些問題，那麼恭喜你，這代表你走在正確的道路上。

熟悉了基本的過程之後，你的下一個里程碑該訂什麼？這就跟你個人的目標有關了。歡迎來找我討論！

在整個過程中要記住的一件事是，在每一步都要不斷地問自己問題，不要放過自己的好奇心。為什麼線性回歸對這個問題表現如此糟糕？ 為什麼結果出來很奇怪？ 有沒有更好的辦法？ 誠實面對你不知道的事情，並致力於找出答案。

你的背景是什麼並不重要。有了正確的心態並實際投入時間，任何人都可以成為機器學習專家。

第7章：線性代數觀念補充

有很多線性代數的概念是我們的個案有間接用到，或是本書有間接提到的，但我認為沒有必要在前面就提出。這些也是很重要的基本概念，所以我列在這裡供大家參考。

線性代數公理

- **這是什麼：**一組無需證明，自然成立的規則。他是線性代數的基本定義，拿來作爲進一步推理和演繹的基礎。
- **直覺：**如果你所定義的數學物件要使用線性代數的概念，那你的數學物件必須要滿足這些公理。 換句話說，當一個系統遵循所有這 8 條規則時，所有線性代數的概念都可以應用於該系統。
- **速查表：**

  有8個公理：

  （$\vec{u}, \vec{v}, \vec{w}$ 是向量，a, b 是標量）

  1. $\vec{u} + (\vec{v} + \vec{w}) = (\vec{u} + \vec{v}) + \vec{w}$
  2. $\vec{v} + \vec{w} = \vec{w} + \vec{v}$
  3. 存在一個向量 0 使得 $0 + \vec{v}$ 對於所有 $\vec{v}$
  4. 對於每個向量 $\vec{v}$ 都有一個向量 $-\vec{v}$ 使得 $\vec{v} + (-\vec{v}) = 0$
  5. $a(b\vec{v}) = (ab)\vec{v}$
  6. $1\vec{v} = \vec{v}$
  7. $a(\vec{v} + \vec{w}) = a\vec{v} + a\vec{w}$
  8. $(a + b)\vec{v} = a\vec{v} + b\vec{v}$

線性變換

- **這是什麼：**線性變換以線性方式將一個向量空間轉換為另一個。它會拉伸或旋轉向量空間（該空間中的所有向量）。
- **直覺：**這是一種將向量變形的方法。一種線性變換可以用一個矩陣表示。一個向量乘上一個矩陣會得到另一個向量。若我們把那個矩陣視爲一種線性變換，則最後產出的向量就是變換之後的向量。
- **為什麼要學習這個：**由於數據在機器學習中可以用向量來表示，線性變換成為用來將數據變形的一種方式。各種機器學習操作中都有這個概念，包括降維、特徵工程、數據規範化等。

> 💬 如果你是第一次看到這個概念，可能會覺得有點困惑。如果你想更直覺的理解，我建議你看這個影片：線性變換和矩陣 | 第三章，線性代數的本質

行列式

- **這是什麼：**如果你把矩陣想成一個線性變換的話，行列式（寫作：$det()$）是一個描述變換的行為的數字。矩陣的行列式會告訴你這個變換對空間的縮放程度以及方向。另外，行列式還可以告訴你這個矩陣是否可逆。
- **為什麼要學習這個：**有時我們在求解模型參數的時候會需要算逆矩陣。行列式是檢查矩陣是否可逆的一種快速方法。
- **速查表：**

$$A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$$
$$det(A) = ad - bc$$

$$A = \begin{bmatrix} a & b & c \\ d & e & e \\ g & h & i \end{bmatrix}$$
$$det(A) = aei + bfg + cdh - ceg - bdi - afh$$

線性依賴

- **這是什麼：**一個術語，用來描述一組向量是否具有冗餘（有的話就是「線性相關」，沒有的話就是「線性獨立」）。根據定義，如果一個向量集中至少有一個列（或行）可以用其他列（或行）的線性組合表示，則稱該向量集中的列（或行）是「線性相關」的。若沒有可一個列（或行）可以用其他列（或行）的線性組合表示，則該向量集是「線性獨立」的。
- **直覺：**向量集中的每個向量其實都帶著一條資訊。這條資訊，是關於「變量之間關係」的資訊。例如，下面系統中的每個方程式（可以將其視為行向量）都包含一條關於 x 和 y 之間關係的信息。下面的系統是「線性獨立」的，因為兩個向量攜帶的信息不同，沒有重複。

$$\begin{cases} x + y = 3 \\ 3x + 2y = 8 \end{cases}$$

  現在，如果我們再添加兩行：

$$\begin{cases} x + y = 3 \\ 3x + 2y = 8 \\ 2x + 2y = 6 \\ 2x + y = 5 \end{cases}$$

  雖然這兩行看起來可能不同，但它們只是前兩行的線性組合。第三個只是第一個乘以 2，而最後一個只是第二個減去第一個。這兩個新行不帶有任何新信息，是多餘的。 當存在這樣的冗餘時，我們就說這個向量集是線性相關的。
- **為什麼要學這個：** 正如您在上面看到的，檢查系統是否線性相關是檢查一組向量（或矩陣）中是否存在冗餘的好方法。 在機器學習中，由於矩陣是數據集，冗餘是數據集中的噪聲。 識別這些冗餘並消除它們可以提高模型性能並降低維度。

矩陣的秩

- **這是什麼：**延續上面「向量帶有訊息」的說法，矩陣的秩會告訴你這個矩陣包含多少信息。它是由矩陣中線性獨立的列或行的數量定義的。
- **直覺：**為了建立直覺，您可以將矩陣中每個線性獨立的行或列視為代表一條資訊。跟其他行有「線性相關」的行不帶有資訊，因為它只是以不同方式表示的相同的行。矩陣的秩是矩陣具有的資訊的總數。