第一章

簡介

「資料！資料！資料！」他很不耐煩地叫喊著。
「沒有磚土，哪燒得出磚塊呢？」

亞瑟・柯南・道爾（*Arthur Conan Doyle*）

資料的威力

我們生活在一個快被大量資料淹沒的世界。許多網站都會追蹤記錄每個使用者的每一個點擊動作。你的智慧型手機會把你每天的所在位置、每秒的移動速度，製成一筆又一筆的記錄。有些人甚至隨身佩戴各種裝置，隨時記錄心率、移動、飲食習慣和睡眠模式。智慧型汽車會收集駕駛習慣，智慧型家庭會收集個人居家習慣，智慧型商場則收集顧客的購買習慣。網際網路本身也呈現出一幅巨大的知識圖譜，包括維基百科這類彼此交叉相連的知識，還有各個特定領域（包括電影、音樂、運動結果、彈珠檯、迷因 [memes]、雞尾酒等等）的資料庫，以及政府機構所提供的各式統計資料（其中有些確實是真實內容！），種種這一切都能輕易把你搞得昏頭轉向。

這世上數不盡的問題（有些甚至還沒有人提問過），答案全都埋藏在資料之中。本書的目的，就是教你如何把它們挖掘出來。

「資料科學」究竟是什麼？

有個笑話說，資料科學家就是比電腦科學家更懂統計學、也比統計學家更懂電腦的人（我知道，這個笑話有點冷）。事實上，有些資料科學家確實是統計學家（這有實質上的意義），有些資料科學家看起來根本就跟軟體工程師沒什麼兩樣。他們有些人是機器學習的專家，有些人則對機器學習一竅不通。有些人擁有高學歷與著作，有些人則完全沒讀過任何學術

```
    { "id": 8, "name": "Kate" },
    { "id": 9, "name": "Klein" }
]
```

他也給了你一份「朋友關係」（friendship）的資料，這份列表（list）裡的資料，全都是兩兩成對的 ID 編號：

```
friendship_pairs = [(0, 1), (0, 2), (1, 2), (1, 3), (2, 3), (3, 4),
                    (4, 5), (5, 6), (5, 7), (6, 8), (7, 8), (8, 9)]
```

舉例來說，**(0,1)** 這樣的一個 tuple 元組，就表示**編號 0**（Hero）和**編號 1**（Dunn）這兩位資料科學家是朋友的關係。這些朋友關係所形成的網路，如圖 1-1 所示。

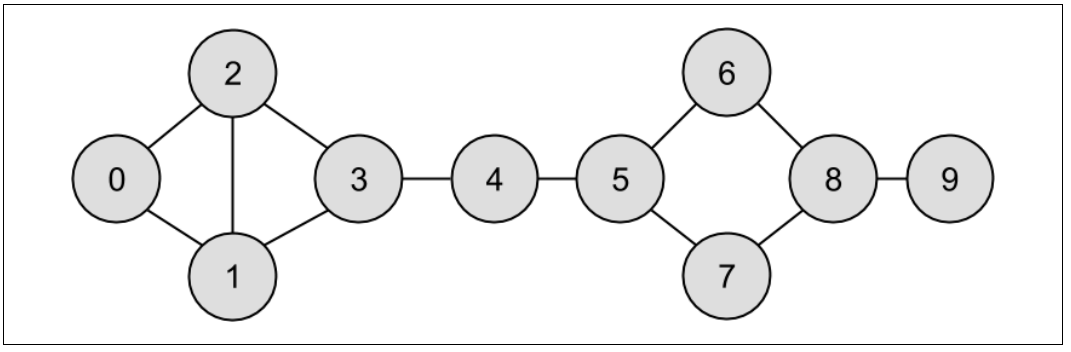

圖 1-1　DataSciencester 的朋友關係網路

把兩兩成對的朋友關係用一個 list 列表來表示，並不是最簡單的做法。如果想找出 1 號使用者所有的朋友關係，你就必須以迭代方式把所有包含 1 的成對資料全都找出來。如果有很多成對的資料，就會耗費很長的時間。

另一種做法則是建立一個 `dict` 字典（在 `dict` 中找東西速度快多了），其中鍵值是使用者 **`id` 編號**，相應值則是包含該使用者所有朋友 **`id` 編號**的一個 list 列表。

現在這個階段，各位還不大需要在意程式碼的細節。到了「第 2 章：Python 速成班」，我們就會給你來一堂 Python 速成課。目前你只需要大致瞭解我們在做什麼就可以了。

為了建立這個 `dict` 字典，我們還是必須逐一查看每組成對的資料，但這個動作只需要執行一次，隨後就可以進行快速的查詢了：

```
# 針對每個使用者編號，都用一個空的 list 列表來初始化這個 dict 字典：
friendships = {user["id"]: [] for user in users}
```

```
            if interested_user_id != user["id"]
        )
```

我們可以運用以上做法，根據共同朋友與共同興趣的資訊，針對「你可以去認識一下的資料科學家」建立更豐富的功能。到了「第 23 章：推薦系統」，我們還會再進一步探索這類型的應用。

薪水和年資的關係

就在你打算去吃午飯時，公關部門的副總跑來問你，能不能針對資料科學家賺多少錢，提供給他一些有趣的分析資料。薪水資料當然是很敏感的，不過，他可以提供給你一份匿名的資料，裡頭只包含每個人的**薪水**（salary，以美元為單位）以及該人員從事資料科學工作的**年資**（tenure，以年為單位）：

```
salaries_and_tenures = [(83000, 8.7), (88000, 8.1),
                        (48000, 0.7), (76000, 6),
                        (69000, 6.5), (76000, 7.5),
                        (60000, 2.5), (83000, 10),
                        (48000, 1.9), (63000, 4.2)]
```

第一步自然是把這些資料畫成圖形（稍後「第 3 章：資料視覺化」就會教你怎麼做）。你可以看到，結果就在圖 1-3 中。

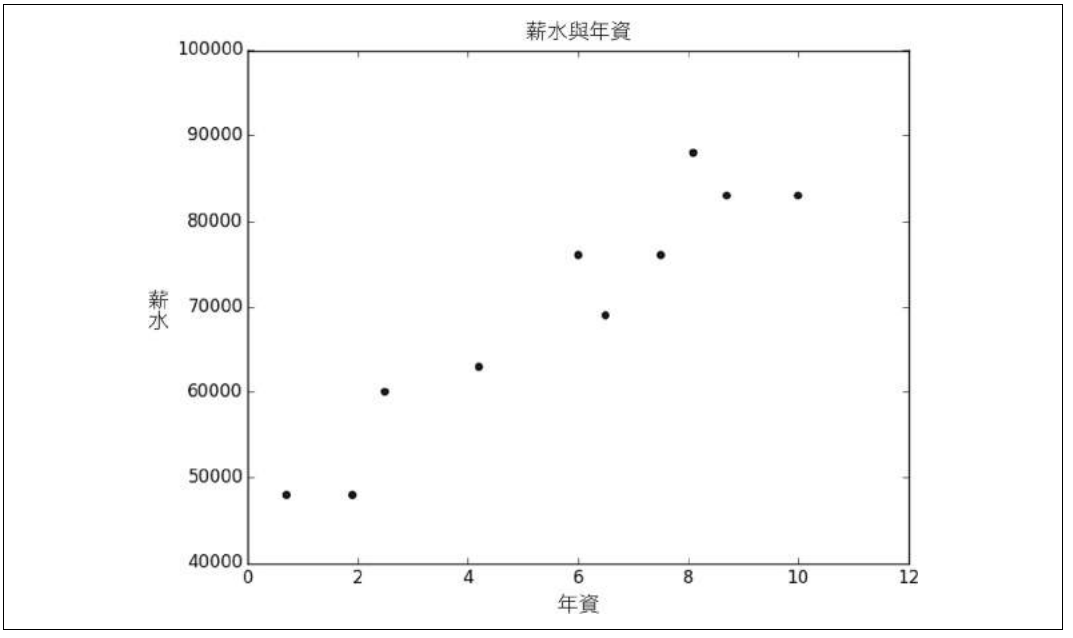

圖 1-3　薪水與年資的關係

第三章

資料視覺化

我相信視覺化就是達成個人目標最有力的做法之一。

哈維．馬偕（*Harvey Mackay*）

在資料科學家的工具箱裡，資料視覺化絕對是少不了的基礎工具。要以視覺化方式呈現資料並不難，但要呈現得好也沒那麼容易。

資料視覺化有兩個主要的用途：

- 瀏覽資料的內容
- 與資料進行溝通

本章的重點是建立一些技能，以便讓各位探索各種資料；本書後續的各種視覺化效果，也會用到本章所介紹的技巧。資料視覺化和本書其他章節所介紹的主題一樣，都是具有十分豐富內涵的研究領域，即使用整本書來介紹也不為過。不過，這裡只打算介紹一些觀念，讓你知道哪些視覺化的做法很棒，哪些做法不怎麼樣。

matplotlib

如果想用視覺化方式呈現資料，可使用的工具非常多。我們打算使用的是 matplotlib（*http://matplotlib.org/*），它是個廣泛被運用的函式庫（而且它也算是有點年紀了）。如果你想製作的是十分精巧、可在網路上進行互動的視覺化效果，它或許並不是最好的選擇，但如果只是製作簡單的長條圖、折線圖、散點圖，這些對它來說倒是游刃有餘。

如前所述，matplotlib 並不是 Python 核心函式庫的一部分。請在啟用的虛擬環境下（如果尚未設定的話，可參考之前「虛擬環境」一節的內容，並按照說明進行設定），使用以下指令把它安裝起來：

```
python -m pip install matplotlib
```

我們會使用的是 matplotlib.pyplot 模組。只要運用最簡單的方式，pyplot 就可以讓你一步一步把視覺化效果建立起來。完成之後，你可以用 savefig 把結果保存起來，或是用 show 把結果顯示出來。

舉例來說，如果要畫出像圖 3-1 那樣的簡單圖形，其實非常簡單：

```
from matplotlib import pyplot as plt

years = [1950, 1960, 1970, 1980, 1990, 2000, 2010]
gdp = [300.2, 543.3, 1075.9, 2862.5, 5979.6, 10289.7, 14958.3]

# 建立一個折線圖，x 軸是年份，y 軸是 GDP
plt.plot(years, gdp, color='green', marker='o', linestyle='solid')

# 加個標題
plt.title("Nominal GDP")

# 在 y 軸上加個標籤
plt.ylabel("Billions of $")
plt.show()
```

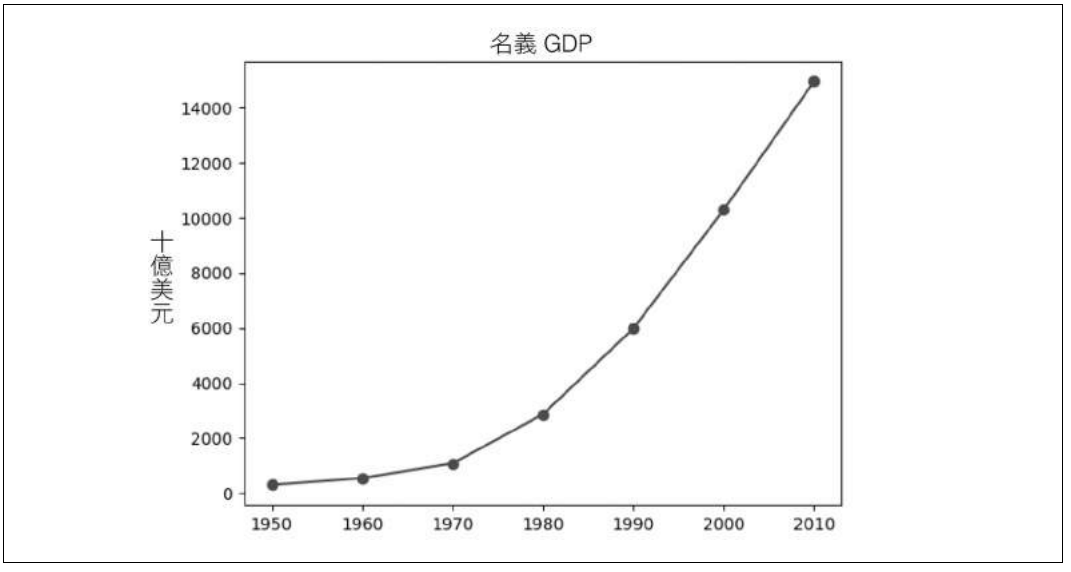

圖 3-1　一個簡單的折線圖

這麼一來，我們就可以計算出向量列表中每個向量（維度相同）各元素的平均值：

```
def vector_mean(vectors: List[Vector]) -> Vector:
    """計算每個元素的平均值"""
    n = len(vectors)
    return scalar_multiply(1/n, vector_sum(vectors))

assert vector_mean([[1, 2], [3, 4], [5, 6]]) == [3, 4]
```

還有一種比較常用的工具叫做**點積**（*dot product*）。兩個向量的點積，就是相應元素相乘之後加總的結果：

```
def dot(v: Vector, w: Vector) -> float:
    """計算 v_1 * w_1 + ... + v_n * w_n"""
    assert len(v) == len(w), "兩個向量必須具有相同的維度"

    return sum(v_i * w_i for v_i, w_i in zip(v, w))

assert dot([1, 2, 3], [4, 5, 6]) == 32  # 1 * 4 + 2 * 5 + 3 * 6
```

如果 `w` 的長度為 1，那麼點積所衡量的就是向量 `v` 在 `w` 方向上的分量。舉例來說，如果 `w=[1,0]`，那麼 `dot(v,w)` 就等於向量 `v` 的第一個元素值。還有另一種說法，就是向量 `v` **投影**在向量 `w` 上的向量長度（如圖 4-2 所示）。

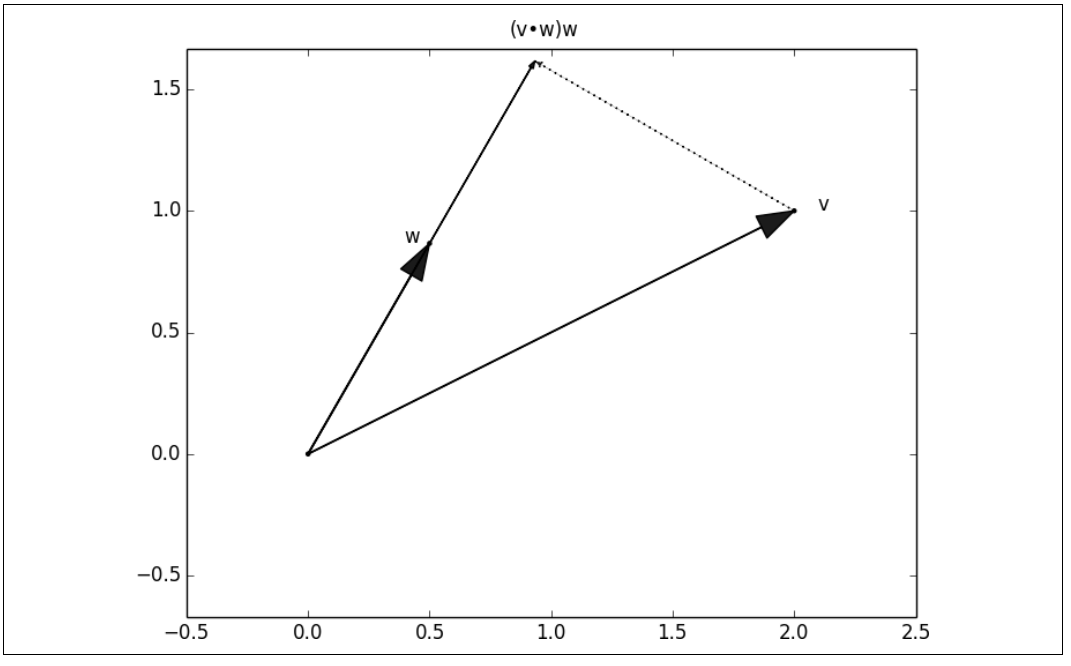

圖 4-2　點積就是向量的投影分量

第五章

統計學

事實往往很棘手，但統計結果看起來好多了。

馬克・吐溫（*Mark Twain*）

如果想進一步瞭解資料，往往會用到統計學方面的數學技巧。統計學是個內涵豐富且高度發展的領域，恐怕需要圖書館的整個書架（甚至整個閱覽室）才足以容納其內涵，所以本章肯定無法進行太深入的討論。既然如此，我就調整一下本章的目標，希望至少可以激發你對統計學的興趣，讓你有個好的開始，可以進一步自我學習。

描述單一組資料

由於使用者之間口耳相傳，再加上一點好運，如今 DataSciencester 的會員數量已獲得相當大的成長；此時融資部門的副總希望你能針對每個會員的朋友數量，提供給他一些更具有描述效果的資料，好讓他能安排一些行銷活動。

只要運用「第 1 章：簡介」所提到的技巧，你就可以輕易製作出這樣的資料。但現在你所面臨的問題是，該如何**描述**這些資料。

無論哪一種資料，最直白的描述方式就是把資料直接呈現出來：

```
num_friends = [100, 49, 41, 40, 25,
               # ... 還有很多
              ]
```

如果資料的數量不多，這種做法甚至可說是最好的一種描述方式。但如果資料數量很多，這樣可能就會顯得很笨拙，而且資料太多反而會讓人搞不清楚狀況（想像一下，如果讓你

看一百萬個數字，那是什麼感覺…）。因此，我們通常會用一些統計數字來描述我們的資料，以說明資料本身所具有的特性。

第一種做法，就是利用 Counter 和 plt.bar，把朋友數量繪製成直方圖（如圖 5-1 所示）：

```
from collections import Counter
import matplotlib.pyplot as plt

friend_counts = Counter(num_friends)
xs = range(101)                         # 最大數字為 100
ys = [friend_counts[x] for x in xs]     # 高度就是朋友的數量
plt.bar(xs, ys)
plt.axis([0, 101, 0, 25])
plt.title("Histogram of Friend Counts")
plt.xlabel("# of friends")
plt.ylabel("# of people")
plt.show()
```

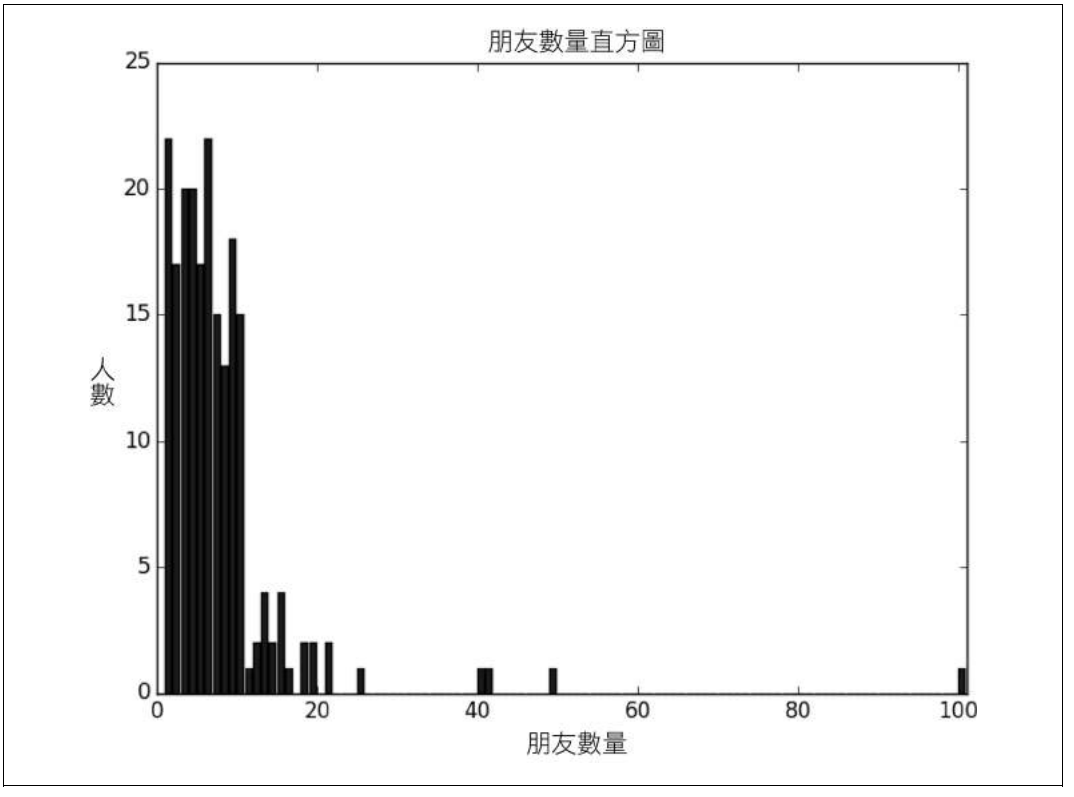

圖 5-1　朋友數量的直方圖

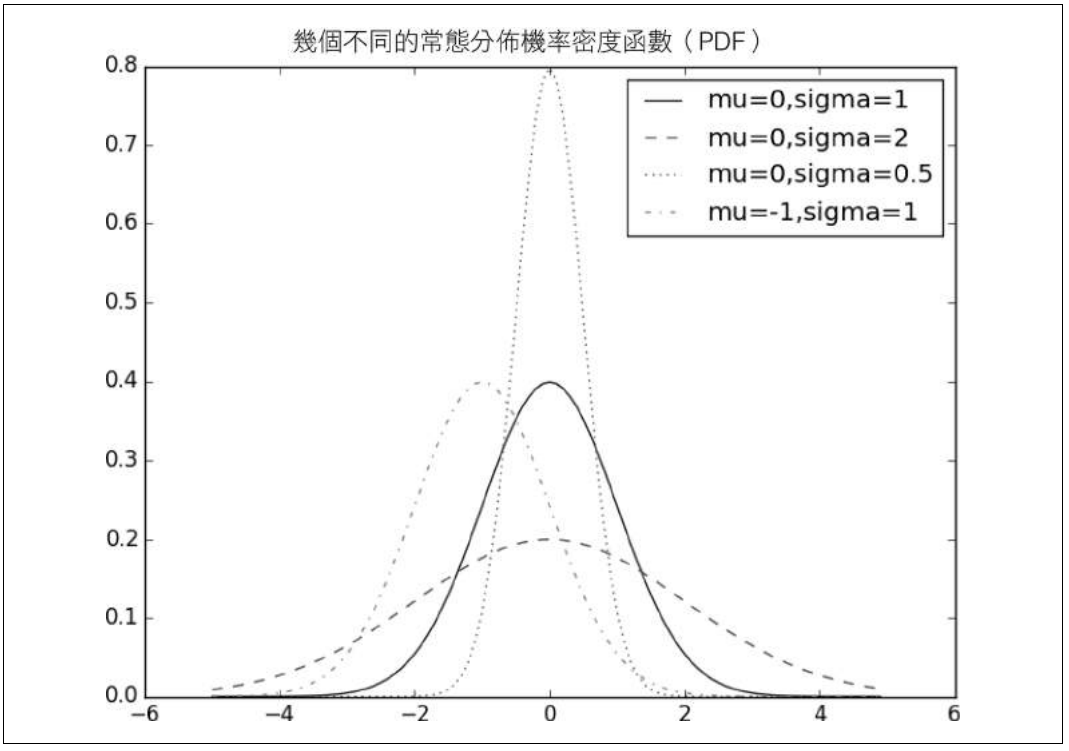

圖 6-2　幾個不同的常態分佈 PDF 機率密度函數

如果 $\mu = 0$ 且 $\sigma = 1$，我們就稱它為「**標準常態分佈**」。如果 Z 是一個標準常態隨機變數，其他的常態分佈就可以表示如下：

$$X = \sigma Z + \mu$$

其中 X 表示的是平均值為 μ、標準差為 σ 的常態分佈隨機變數。反過來說，如果 X 是一個平均值為 μ、標準差為 σ 的常態隨機變數：

$$Z = (X - \mu)/\sigma$$

這樣即可得出標準常態隨機變數 Z。

常態分佈的 CDF 累積分佈函數無法用「簡單」的解析形式來表示，不過，我們可以運用 Python 的 `math.erf` 誤差函式（*http://en.wikipedia.org/wiki/Error_function*）：

```
def normal_cdf(x: float, mu: float = 0, sigma: float = 1) -> float:
    return (1 + math.erf((x - mu) / math.sqrt(2) / sigma)) / 2
```

在圖 6-3 中，我們再次畫出幾個不同的常態分佈 CDF 圖形：

```
xs = [x / 10.0 for x in range(-50, 50)]
plt.plot(xs,[normal_cdf(x,sigma=1) for x in xs],'-',label='mu=0,sigma=1')
plt.plot(xs,[normal_cdf(x,sigma=2) for x in xs],'--',label='mu=0,sigma=2')
plt.plot(xs,[normal_cdf(x,sigma=0.5) for x in xs],':',label='mu=0,sigma=0.5')
plt.plot(xs,[normal_cdf(x,mu=-1) for x in xs],'-.',label='mu=-1,sigma=1')
plt.legend(loc=4) # 右下角
plt.title("Various Normal cdfs")
plt.show()
```

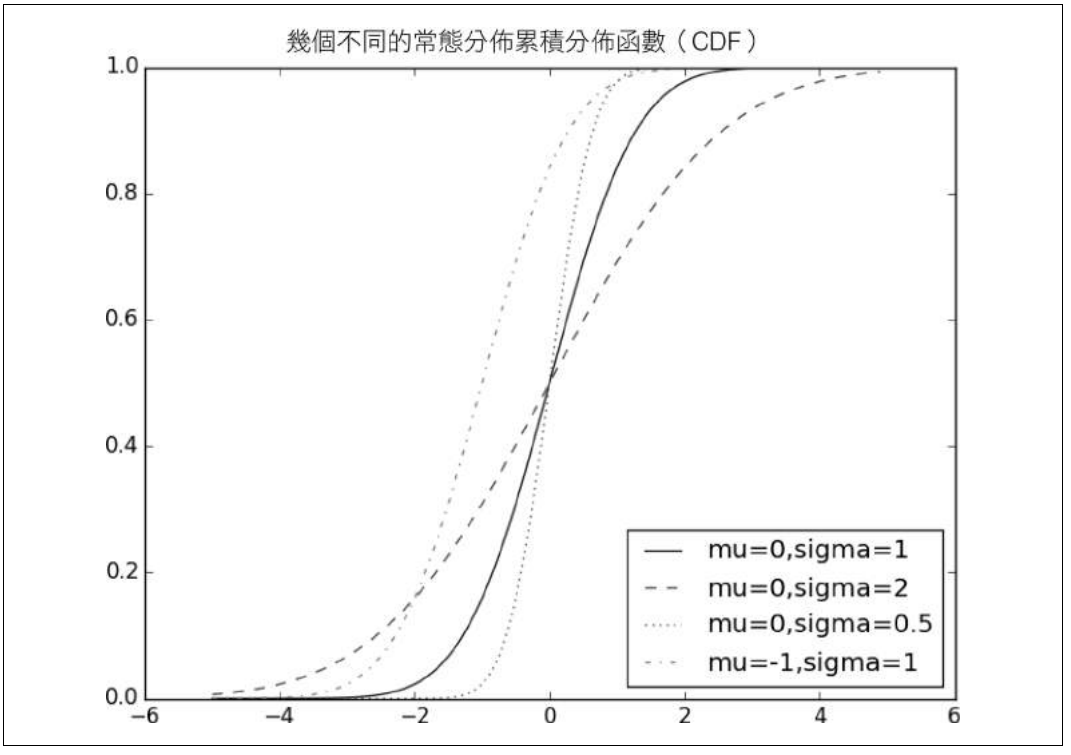

圖 6-3　幾個不同的常態分佈 CDF 累積分佈函數

有時候我們也需要 normal_cdf 的逆向操作，根據某特定機率求出所對應的值。這種逆向操作並沒有簡單的做法，不過，由於 normal_cdf 是一種數值會嚴格遞增的連續函數，所以我們可以採用**二元搜尋**（*binary search*）的做法（*http://en.wikipedia.org/wiki/Binary_search_algorithm*）：

```
def inverse_normal_cdf(p: float,
                       mu: float = 0,
                       sigma: float = 1,
                       tolerance: float = 0.00001) -> float:
```

第八章

梯度遞減

總愛拿祖先出來炫耀的人，其實只是在說明，自己的成就遞減與不足。

聖力嘉學院（*Seneca*）

從事資料科學相關工作時，經常需要針對某些特定狀況找出最佳的模型。「最佳」的意思有時候是指「模型誤差最小化」，有時則是「最大化資料的可能性」。

這也就表示，我們會有一大堆最佳化的問題需要解決。而且我們必須從最基本開始，逐一解決這些問題。我們會採用一種叫做「**梯度遞減**（*gradient descent*）」的做法，這種做法很適合我們從頭開始逐步解決問題。或許你並不覺得這個做法有什麼特別之處，但它確實可以讓我們在本書中，做出許多相當厲害的事情，所以，跟著我繼續前進就對了！

梯度遞減背後的概念

假設有一個函數 `f`，可以接受一個實數向量做為輸入，然後輸出一個單一的實數。像這樣的一個簡單函數範例如下：

```
from scratch.linear_algebra import Vector, dot

def sum_of_squares(v: Vector) -> float:
    """計算 v 之中所有元素的平方和"""
    return dot(v, v)
```

我們經常需要求取這類函數的最大值（或最小值）。也就是說，我們需要找出某個輸入值 `v`，能讓函數得出最大（或最小）的可能值。

像我們前面所給出的函數，只要根據其**梯度**（如果你還記得微積分的話，這裡所謂的梯度，就是指向量的偏導數），就可以知道輸入值沿著哪個方向移動，能讓函數值獲得最大的增長（如果你已經忘了微積分的相關知識，建議你上網查查）。

因此，如果想求出函數的最大值，其中一種方法就是先取個隨機的起始點，計算其梯度，然後沿著梯度方向（也就是函數增長最多的方向）移動一小步，再重複以上步驟即可。同樣根據類似的概念，只要往梯度**反方向**移動，就可以求出函數的最小值，如圖 8-1 所示。

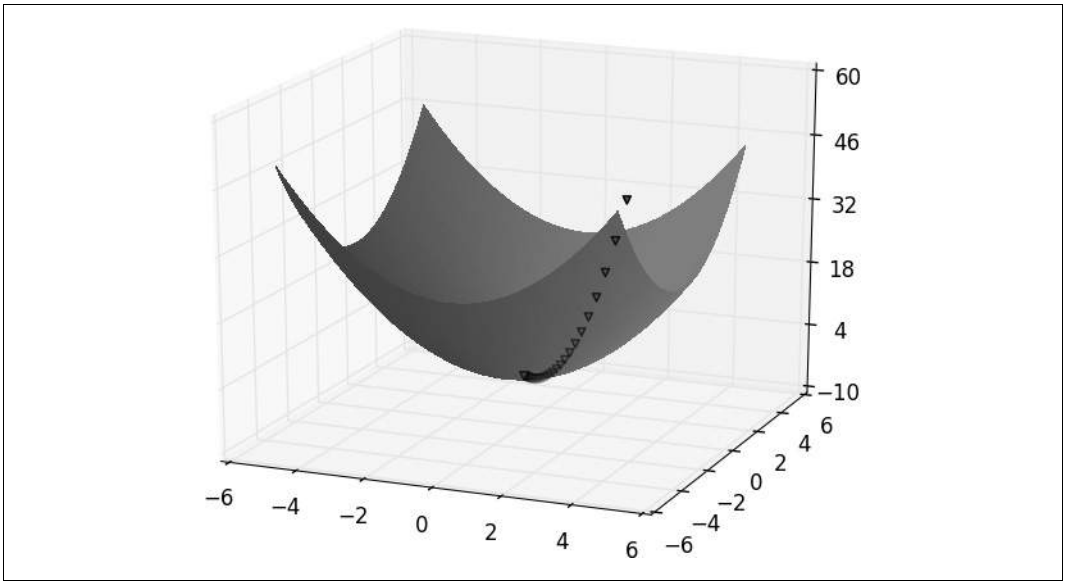

圖 8-1　運用梯度遞減的概念，找出最小值

如果函數只有一個全域最小值，以上程序就很有可能找出這個值。但如果函數有好幾個（局部的）極小值，這個程序就有可能找錯點；萬一要是這樣的話，或許就應該多嘗試幾個不同的起始點重新執行該程序。如果函數根本沒有最小值，這個程序就會不斷執行而永遠無法結束。

梯度的估算

如果 `f` 函數只有一個變數，那麼在點 `x` 所謂的導數（derivative），就是衡量 `x` 出現小小變化時 `f(x)` 跟著變化的程度。導數可定義為差商（difference quotient）的極限：

```
from typing import Callable

def difference_quotient(f: Callable[[float], float],
```