



---

# PYTHON 在統計的應用 與 LATEX 基礎語法

---

一本讓你從 PYTHON 的初心者，到利用 SKLEARN 套件進行機器學習的  
工具書，是你學習程式的最好夥伴！

作者  
林哲兆

國立臺北大學統計系研究所



2024.01.10  
NTPU





# 目錄

序	x
<b>1 X<sub>E</sub>L<sub>A</sub>T<sub>E</sub>X 的模板</b>	<b>1</b>
1.1 字型變化 . . . . .	1
1.1.1 基本字型更改 . . . . .	1
1.1.2 字體顏色的變更 . . . . .	1
1.2 數學式的語法與數學模式 . . . . .	2
1.2.1 數學模式與統計數學式 . . . . .	2
1.2.2 分配數學式 . . . . .	2
1.2.3 其他數學式 . . . . .	3
1.2.4 聯立方程式 . . . . .	3
1.2.5 矩陣 . . . . .	3
1.3 插入表格的方法 . . . . .	4
1.3.1 基本表格 . . . . .	4
1.3.2 合併文字表格 . . . . .	4
1.3.3 兩個表格並列 . . . . .	5
1.3.4 跨頁表格 . . . . .	5
1.4 插入圖片各種方式 . . . . .	7
1.4.1 基本圖片與寬度調整 . . . . .	7
1.4.2 文繞圖 . . . . .	7
1.4.3 多張圖片的放置 . . . . .	8
1.4.4 使用 minipage 放置圖片 . . . . .	8
1.4.5 把註解放置兩旁的圖片 . . . . .	9
1.5 結論 . . . . .	9
<b>2 函數的繪製：初入 python</b>	<b>11</b>
2.1 函數繪圖 . . . . .	11
2.1.1 三角函數 . . . . .	11
2.1.2 sigmoid 函數 . . . . .	13
2.1.3 震盪函數 . . . . .	15
2.1.4 定義域非實數的函數 . . . . .	16
2.1.5 多項式函數與反函數 . . . . .	17
2.1.6 常態分配的機率密度函數 . . . . .	18
2.1.7 多項式函數的極值與微分 . . . . .	19
2.1.8 常見極限函數 . . . . .	20
2.1.9 水平分段函數 . . . . .	21
2.1.10 圓形 . . . . .	21



2.1.11 正方形 . . . . .	23
2.2 專題 . . . . .	24
2.2.1 第一小題 . . . . .	24
2.2.2 第二小題 . . . . .	25
2.2.3 第三小題 . . . . .	26
2.3 結論 . . . . .	27
<b>3 常用分配與亂數產生</b>	<b>29</b>
3.1 離散型機率分配的圖形 . . . . .	29
3.1.1 伯努力分配 . . . . .	29
3.1.2 二項分配 . . . . .	30
3.1.3 卜瓦松分配 . . . . .	31
3.1.4 幾何分配 . . . . .	32
3.1.5 負二項分配 . . . . .	32
3.1.6 離散均勻分配 . . . . .	33
3.1.7 超幾何分配 . . . . .	34
3.2 連續型機率分配的圖形 . . . . .	35
3.2.1 指數分配 . . . . .	35
3.2.2 迦馬分配 . . . . .	35
3.2.3 貝塔分配 . . . . .	37
3.2.4 t 分配 . . . . .	38
3.2.5 卡方分配 . . . . .	39
3.2.6 F 分配 . . . . .	40
3.2.7 柯西分配 . . . . .	41
3.3 亂數產生 . . . . .	41
3.3.1 卡方分配 . . . . .	42
3.3.2 迦馬分配 . . . . .	42
3.3.3 貝塔分配 . . . . .	42
3.4 抽樣分配 . . . . .	43
3.4.1 卜瓦松分配加成性 . . . . .	43
3.4.2 指數分配加成性 . . . . .	45
3.4.3 中央極限定理 . . . . .	47
3.5 數理統計題目驗證 . . . . .	49
3.6 結論 . . . . .	50
<b>4 監督式學習之迴歸學習器</b>	<b>51</b>
4.1 簡單線性迴歸模型 . . . . .	51
4.1.1 模型理論 . . . . .	51
4.1.2 資料實作 . . . . .	52
4.2 加廣型線性回歸模型 . . . . .	54
4.2.1 模型理論 . . . . .	54
4.2.2 資料實作 . . . . .	54
4.3 邏輯斯回歸模型 . . . . .	56
4.4 結論 . . . . .	58
<b>5 監督式學習之 LDA、QDA 與 KNN 學習器</b>	<b>59</b>
5.1 線性判別分析 (LDA) . . . . .	59
5.1.1 理論背景 . . . . .	59
5.1.2 資料實作 . . . . .	60



---

5.2 二次判別分析 (QDA) . . . . .	64
5.2.1 理論背景 . . . . .	64
5.2.2 資料實作 . . . . .	64
5.3 K-近鄰演算法 (KNN) . . . . .	67
5.3.1 理論 . . . . .	67
5.3.2 資料實作 . . . . .	67
5.4 結論 . . . . .	70
<b>6 監督式學習之類神經網路的原理及應用</b>	<b>73</b>
6.1 類神經網路之原理 . . . . .	73
6.2 類神經網路之應用 . . . . .	74
6.2.1 機械手臂運動 . . . . .	74
6.2.2 手寫圖形辨識 . . . . .	79
6.3 結論 . . . . .	81
<b>7 多種學習器的比較</b>	<b>83</b>
7.1 蒙地卡羅模擬 . . . . .	83
7.2 模型比較 . . . . .	83
7.2.1 亂數產生資料 . . . . .	83
7.2.2 消費者資料 . . . . .	84
7.3 結論 . . . . .	85





## 圖目錄

圖 1.1 插入圖片的基本語法 . . . . .	7
圖 1.2 文繞圖示例 . . . . .	7
圖 1.3 將圖片設置成 2*2 的作法 . . . . .	8
圖 1.4 SCfigure 的使用 . . . . .	9
圖 2.1 使用 subplot 展示四個函數圖形 . . . . .	12
圖 2.2 $\alpha = 1$ 時的 Sigmoid 函數 . . . . .	14
圖 2.3 不同 $\alpha$ 值的 Sigmoid 函數 . . . . .	14
圖 2.4 更多不同 $\alpha$ 值的 Sigmoid 函數 . . . . .	15
圖 2.5 函數 $y = e^{-\frac{x}{10}} \sin(x)$ 的走勢 . . . . .	15
圖 2.6 $y = \frac{1}{x-2}$ 與其漸進線 . . . . .	16
圖 2.7 函數 $y = x^3 + 2$ 與它的反函數 . . . . .	17
圖 2.8 函數 $y = \frac{1}{\sqrt{2\pi}} e^{-\frac{(x-3)^2}{2}}$ . . . . .	18
圖 2.9 函數 $y = 3x^3 - x^4$ ，正確與錯誤示範 . . . . .	19
圖 2.10 函數 $y = \frac{\ln x}{x^2}$ 與其漸進線 . . . . .	20
圖 2.11 函數 $y = f(x)$ . . . . .	21
圖 2.12 函數 $x^2 + y^2 = 1$ . . . . .	22
圖 2.13 邊長為 1 正方形 . . . . .	23
圖 2.14 $S_n$ 隨著 $n$ 的變化 . . . . .	25
圖 2.15 $\gamma_n = \text{sum of the red areas}$ . . . . .	25
圖 2.16 $\frac{1}{2}(1 - \frac{1}{n+1}) < \gamma_n < 1$ . . . . .	26
圖 3.1 伯努力分配 ( $p = 0.3$ ) . . . . .	29
圖 3.2 不同參數下二項分配的圖形 . . . . .	30
圖 3.3 二項分配的圖形與近似的常態分配 . . . . .	31
圖 3.4 卜瓦松分配分配的圖形 . . . . .	31
圖 3.5 不同參數的幾何分配圖形 . . . . .	32
圖 3.6 不同參數的負二項分配圖形 . . . . .	33
圖 3.7 離散均勻分配 (1,5) . . . . .	34
圖 3.8 超幾何分配不同參數下的圖形 . . . . .	34
圖 3.9 指數分配不同參數的圖形 . . . . .	35
圖 3.10 迦馬分配不同參數的圖形 . . . . .	36
圖 3.11 貝塔分配不同參數的圖形 . . . . .	37
圖 3.12 t 分配不同參數的圖形 . . . . .	38
圖 3.13 卡方分配不同參數的圖形 . . . . .	39
圖 3.14 F 分配不同參數的圖形 . . . . .	40
圖 3.15 柯西分配不同尺度參數的圖形 . . . . .	41
圖 3.16 卡方 1 的亂數 . . . . .	42



圖 3.17 $\Gamma(2,1)$ 的亂數 . . . . .	43
圖 3.18 $Beta(2,5)$ 的亂數 . . . . .	44
圖 3.20 20 個樣本數 1000 的 $poisson(1)$ 分配相加 . . . . .	44
圖 3.19 20 個樣本數 100 的 $poisson(1)$ 分配相加 . . . . .	45
圖 3.21 20 個樣本數 10000 的 $poisson(1)$ 分配相加 . . . . .	45
圖 3.22 30 個 $exp(1)$ 樣本進行轉換 . . . . .	46
圖 3.23 100 個 $exp(1)$ 樣本進行轉換 . . . . .	46
圖 3.24 1000 個 $exp(1)$ 樣本進行轉換 . . . . .	47
圖 3.25 $n = 20$ . . . . .	48
圖 3.26 $n = 100$ . . . . .	48
圖 3.27 $n = 1000$ . . . . .	49
圖 3.28 $n = 100000$ . . . . .	49
圖 3.29 四個數字隨機抽樣的平均數 . . . . .	50
圖 4.1 透過 <code>make_blob</code> 生成的樣本分類 . . . . .	52
圖 4.2 消費者購買行為的線性迴歸分類 . . . . .	53
圖 4.3 二維常態亂數樣本的線性迴歸分類 . . . . .	53
圖 4.4 透過 <code>make_blob</code> 生成樣本的線性迴歸分類 . . . . .	54
圖 4.5 消費者購買行為的線性迴歸分類 . . . . .	55
圖 4.6 二維常態亂數樣本的線性迴歸分類 . . . . .	55
圖 4.7 透過 <code>make_blob</code> 生成樣本的邏輯斯迴歸分類 . . . . .	56
圖 4.8 消費者購買行為的邏輯斯迴歸分類 . . . . .	57
圖 4.9 二維常態亂數樣本的邏輯斯迴歸分類 . . . . .	57
圖 4.10 三個群組樣本的邏輯斯迴歸分類 . . . . .	58
圖 5.1 資料離散程度大的分類結果 . . . . .	60
圖 5.2 共變異數矩陣相同的資料分類結果 . . . . .	61
圖 5.3 共變異數矩陣不相同的資料分類結果 . . . . .	61
圖 5.4 消費者購買行為資料分類結果 . . . . .	62
圖 5.5 三種且共變異數矩陣相同資料分類結果 . . . . .	63
圖 5.6 三種且共變異數矩陣不同資料分類結果 . . . . .	63
圖 5.7 LDA 與 QDA 比較 . . . . .	64
圖 5.8 LDA 與 QDA 比較 (共變異數矩陣相同) . . . . .	65
圖 5.9 LDA 與 QDA 比較 (共變異數矩陣不相同) . . . . .	65
圖 5.10 LDA 與 QDA 比較 (消費者資料) . . . . .	66
圖 5.11 三分類問題 . . . . .	66
圖 5.12 簡單資料分類 . . . . .	68
圖 5.13 共變異數矩陣相同分類 . . . . .	68
圖 5.14 共變異數矩陣不同分類 . . . . .	69
圖 5.15 消費者資料分類 . . . . .	69
圖 5.16 共變異數矩陣相同的三分類資料 . . . . .	70
圖 5.17 共變異數矩陣不同的三分類資料 . . . . .	70
圖 6.1 類神經網路的原理圖示 . . . . .	73
圖 6.2 機械手臂示意圖 . . . . .	74
圖 6.3 機械手臂活動範圍 . . . . .	75
圖 6.4 機械手臂分佈點的訓練資料 . . . . .	75
圖 6.5 機械手臂在不同神經元數的訓練結果 . . . . .	76
圖 6.6 機械手臂分佈點的訓練資料 . . . . .	77



---

圖 6.7 機械手臂分佈點的訓練資料與測試資料 . . . . .	77
圖 6.8 樣本數多寡對模型的影響 . . . . .	78
圖 6.9 不同套件的展示 . . . . .	79
圖 6.10 手寫數字資料 . . . . .	79
圖 6.11 手寫數字資料辨識結果 (神經元數 = 30) . . . . .	80
圖 6.12 手寫數字資料辨識結果 (神經元數 = 40) . . . . .	80
圖 6.13 手寫字母資料 . . . . .	81
圖 6.14 不同神經元的混淆矩陣比較 . . . . .	81
圖 7.1 常態亂數資料 . . . . .	84
圖 7.2 消費者購買行為資料 . . . . .	85





## 表目錄

表 1.1 矩陣語法整理 . . . . .	4
表 1.4 <b>multicolumn</b> 橫跨表格 . . . . .	4
表 1.2 最基本的表格 . . . . .	5
表 1.3 有空格的表格 . . . . .	5
表 1.5 兩表並列 . . . . .	5
表 1.6 長型表格 . . . . .	6
表 5.1 三種模型在三筆二分類資料的誤判率 . . . . .	71
表 5.2 三種模型在兩筆二分類資料的誤判率 . . . . .	71
表 7.1 七種模型在亂數資料的誤判率 . . . . .	84
表 7.2 七種模型在消費者資料的誤判率 . . . . .	84





# 序

作為現代數據處理的重要工具，Python 已經成為統計學領域的一把鑰匙。它以其靈活性、強大的套件庫和廣泛的應用範圍，帶來了統計學的新思維和可能性。同時，**LaTeX** 作為一個優雅而強大的排版系統，為呈現數據和概念提供了優美的展示方式。

《Python 在統計的應用與 LaTeX 基礎語法》旨在探索這兩個領域的結合，將它們帶給讀者們以全新的方式。本書通過探討 Python 在統計分析中的實際應用，與 LaTeX 作為一種排版工具，向讀者展示如何以精準和美觀的方式呈現數據和概念。

在本書中，我們將探討 Python 如何處理統計數據，以及如何運用 LaTeX 的語法和排版技巧，來展示這些統計分析的結果。無論您是統計學的新手、Python 程式設計的愛好者，或是對 LaTeX 排版感興趣的人，這本書都將為您帶來新的知識和技能。

作者期待著與讀者一起探索 Python 在統計學中的應用，以及 LaTeX 排版的魅力。這是一趟充滿發現和啟發的旅程，何不馬上啟程呢？

林哲兆  
2024 年 1 月於台北大學





# 第 1 章

## X<sub>E</sub>LATE<sub>X</sub> 的模板

LaTeX 是一個專業的排版系統，被廣泛應用於學術、科技和出版領域。它基於 TeX，提供強大的排版功能和靈活的文件佈局。LaTeX 通過標記語言的方式書寫，使用宏包和命令來控制文件樣式和結構，使得使用者可以專注於內容創作而不必過多關注格式。其優點包括優秀的數學公式排版、高質量的文件輸出、跨平台支持以及開放的可擴展性。使用者可以創建複雜的文件，如論文、書籍、報告和演示文稿。LaTeX 的學習曲線較為陡峭，但一旦掌握，將會為使用者帶來強大的排版能力和精美的文件效果。

### 1.1 字型變化

#### 1.1.1 基本字型更改

以下是一些不同的字型變化的示例：

- LaTeX 提供出色的排版品質，特別擅長處理複雜的數學公式、科技文件或學術論文。其自動化的排版系統能夠生成高品質、專業風格的文件，包括合理的斷詞斷行、美觀的頁面配置以及對圖表和數學公式的優秀支持。
- LaTeX 使用標記語言書寫，允許使用者將內容與格式分開。這表示您可以專注於文檔內容的創作，而不必頻繁關注格式細節。透過使用宏包和命令，能夠在不改變內容的情況下輕鬆調整文檔的外觀和結構。
- LaTeX 是跨平台的，可在不同作業系統上運行。它是開源的，具有廣泛的宏包和支持社群。這表示可以在不同系統上免費使用，並且有著強大的擴展性，使用者可以根據需要添加各種功能和定制化工具。

#### 1.1.2 字體顏色的變更

LaTeX 是一個優秀的排版系統，但也有一些缺點：

- 對於新手來說，LaTeX 的學習曲線可能較為陡峭。相較於所見即所得的編輯器，它採用標記語言書寫，需要一定時間來熟悉其語



法和工作流程。初學者可能需要投入更多時間來掌握基本命令和宏包的使用。

- 對於簡單的文件或佈局較為簡單的需求，LaTeX 可能顯得過於繁瑣。在一些基本的編輯、格式設置或者排版需求方面，使用所見即所得的編輯器可能更加直觀和高效。

綜合來看，LaTeX 是一款強大且廣泛應用的排版系統，尤其適合處理複雜的數學公式、科技文件或學術論文。但對於某些場景下的初學者或簡單文件的需求，可能需要權衡其學習曲線和使用上的複雜度。

## 1.2 數學式的語法與數學模式

### 1.2.1 數學模式與統計數學式

latex 具有特別的數學模式功能，可以利用指定語法，將鍵盤不存在的符號打印出來，以下利用統計學常用的幾個數學公式展示 latex 的數學模式：

$$\bar{x} = \frac{1}{n} \sum_{i=1}^n x_i \quad (1.1)$$

方程式 (1.1) 是最常見的平均數公式

$$\sigma^2 = \frac{1}{n} \sum_{i=1}^n (x_i - \bar{x})^2 \quad (1.2)$$

方程式 (1.2) 是變異數的公式

$$\mathbb{E}(X) = \sum_{i=1}^n x_i p_i \quad (1.3)$$

方程式 (1.3) 是離散型隨機變數的期望值公式

$$\mathbb{E}(X) = \int_{-\infty}^{\infty} x f(x) dx \quad (1.4)$$

方程式 (1.4) 是連續型隨機變數的期望值公式

### 1.2.2 分配數學式

下面是一些特殊的機率分配：

雙指數分配：

$$f(x; \mu, b) = \frac{1}{2b} \exp\left(-\frac{|x - \mu|}{b}\right), \quad -\infty < x < \infty$$



科西分配:

$$f(x; \alpha, \beta) = \frac{1}{\pi \beta \left[ 1 + \left( \frac{x-\alpha}{\beta} \right)^2 \right]}, \quad -\infty < x < \infty$$

貝他分配:

$$f(x; \alpha, \beta) = \frac{1}{B(\alpha, \beta)} \cdot x^{\alpha-1} \cdot (1-x)^{\beta-1}, \quad 0 \leq x \leq 1$$

### 1.2.3 其他數學式

以下是部分基礎數學式子：

$$(a+b)^2 = a^2 + 2ab + b^2$$

$$\cos 2x = \cos^2 x - \sin^2 x$$

$$(y^m)^n = y^{mn}$$

$${}_b^a Y_d^c$$

$$e^{t \cos \theta}$$

$$\lim_{n \rightarrow \infty} \sum_{i=1}^n \frac{1}{n}$$

$$y_1 = \frac{1}{3}(x_1 + \omega x_2 + \omega^2 x_3)$$

### 1.2.4 聯立方程式

左邊需要括號右邊不用，右邊可以用 right.  
mbox 可以強制不換行

$$g(x, y) = \begin{cases} f(x, y), & \text{if } x < y \\ f(y, x), & \text{if } x > y \\ 0, & \text{otherwise.} \end{cases}$$

### 1.2.5 矩陣

#### 1. 基本矩陣表示

$$A = \begin{bmatrix} a & m & x \\ b & n & y \\ c & o & z \end{bmatrix}$$

#### 2. 兩矩陣相乘

$$\begin{bmatrix} a & m & x \\ b & n & y \\ c & o & z \end{bmatrix} \times \begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} a & a+m & a+m+x \\ b & b+n & b+n+y \\ c & c+o & c+o+z \end{bmatrix}$$



### 3. 矩陣行列式計算

$$\begin{vmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{vmatrix} = 1 \times \begin{vmatrix} 5 & 6 \\ 8 & 9 \end{vmatrix} - 2 \times \begin{vmatrix} 4 & 6 \\ 7 & 9 \end{vmatrix} + 3 \times \begin{vmatrix} 4 & 5 \\ 7 & 8 \end{vmatrix}$$

### 4. 無窮矩陣

$$\begin{pmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,m} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m,1} & a_{m,2} & \cdots & a_{m,m} \end{pmatrix}$$

表 1.1: 矩陣語法整理

語法	分界符號	語法	分界符號
<code>matrix</code>	無括號	<code>bmatrix</code>	方括號 []
<code>pmatrix</code>	小括號 ()	<code>Bmatrix</code>	大括號 {}
<code>vmatrix</code>	單垂線	<code>Vmatrix</code>	雙垂線

## 1.3 插入表格的方法

### 1.3.1 基本表格

以下是一些簡單的表格示例：

### 1.3.2 合併文字表格

交易所交易基金（Exchange-Traded Fund，ETF）是一種投資工具，通常被描述為股票和基金的混合體。它們結合了股票的流動性和基金的多元化投資策略，使投資者能夠輕鬆地投資於多個資產類別，如股票、債券、商品、貨幣和其他金融工具。ETF 是一種受歡迎的投資工具，表 1.3.2 是目前的熱門的幾檔 etf。

股票代碼	排行	股票資訊	
		殖利率%	目前股價 \$
00878	1	13.61	22.56
0056	2	16.41	34.89
00713	3	10.87	46.59
006208	4	9.30	70.9
0050	5	9.24	122.56

表 1.4: multicolumn 橫跨表格



表 1.2: 最基本的表格

機型	售價	處理器	材質
iphone 15	30000	A14	鋁金屬
iphone 15 pro	35000	A15	鈦金屬
iphone 15 pro max	45000	A15	鈦金屬

表 1.3: 有空格的表格

季度	物品	價格 (\$)	成本 (\$)	銷售數量
第一季	桌子	1000	650	5
	椅子	800	550	10
第二季	桌子	1000	650	5
	椅子	800	550	10

### 1.3.3 兩個表格並列

表 1.5: 兩表並列

車種	去年售出
Toyota Corolla Cross	2,927 輛
Honda CR-V 1.5	1,378 輛
Toyota Town Ace	1,369 輛
Lexus NX	1,351 輛
TESLA MODEL Y	1,284 輹

學生 \ 科目	微積分	統計學	會計學
小王	50	67	88
小美	89	22	91
小明	68	66	87

### 1.3.4 跨頁表格



表 1.6: 長型表格



## 1.4 插入圖片各種方式

圖片的插入可以有四種檔案 jpg,png,eps 與 pdf，格式可以建立純圖片、文繞圖、以及 mxn 的多張圖片擺放。

### 1.4.1 基本圖片與寬度調整

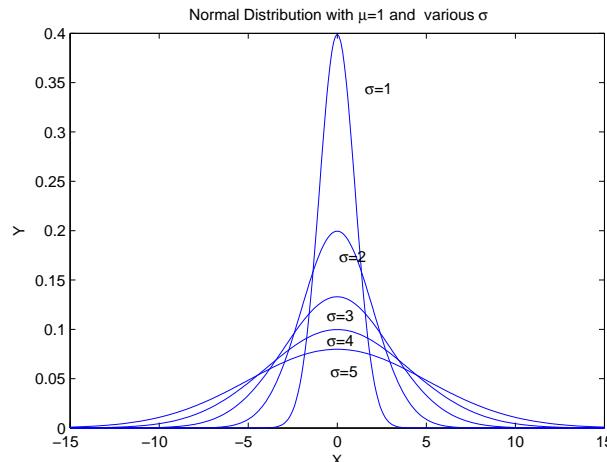


圖 1.1: 插入圖片的基本語法

### 1.4.2 文繞圖

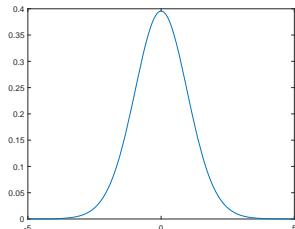
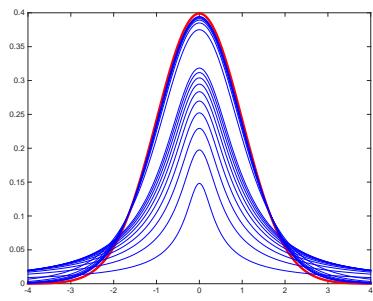


圖 1.2: 文繞圖示例

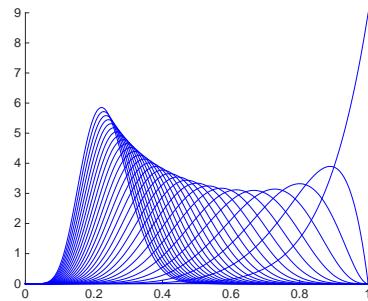
使用文繞圖需要使用 `wrapfigure` 套件，參數可以控制圖片排在左邊或是右邊，以及控制圖片的寬度。必要的時候也可以透過 `vspace` 去控制圖片的上下位置。圖 1.2 是一張常態分配的圖形，常態分配是一個非常常見的連續機率分布。常態分布在統計學上十分重要，經常用在自然和社會科學來代表一個不明的隨機變數。



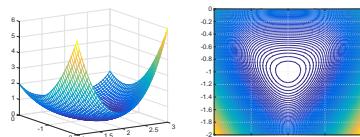
### 1.4.3 多張圖片的放置



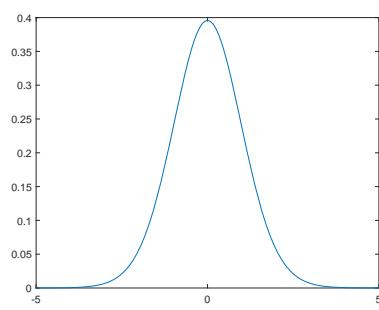
(a) 分配圖



(b) 分配圖



(c) 綜合圖

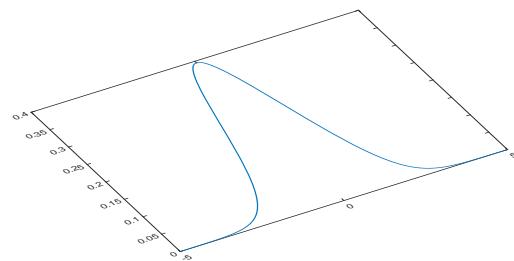


(d) 單一參數分配圖

圖 1.3: 將圖片設置成  $2 \times 2$  的作法

### 1.4.4 使用 `minipage` 放置圖片

使用 `minipage` 可以自行調整兩邊的寬度，適合必須與文字密切貼近的圖形。除了調整寬度外，也可以利用 `angle`、`width`、`heights` 去調整圖形的角度與長寬，但要注意 PDF 沒辦法進行 `FIGURE` 的程序。





#### 1.4.5 把註解放置兩旁的圖片



圖 1.4: SCfigure 的使用

## 1.5 結論

本文將常用的 `latex` 語法、數學式舉例出來，利用本文作為往後撰寫文章的模板，可以省去很多查找語法的過程，希望讀者也能善用 `latex` 這個自動排版軟體，撰寫美觀的文章。





## 第 2 章

# 函數的繪製：初入 python

本文使用 Python 的 Matplotlib 來繪製函數，除了基本的函數外，還包括三角函數、Sigmoid 函數以及常態分配 PDF 等，並會在最後一小節討論一個小型專題。另外，本文除了將函數繪製完成之外，也會特別將需要注意的程式碼列出來，用以紀錄避免未來發生同樣的錯誤。

### 2.1 函數繪圖

#### 2.1.1 三角函數

第一題包括四個函數，其中第四個是自行添加的，目的都是希望利用圖片能找出函數在  $x$  趨近於 0 時的極限，使用四個函數如下，都是經典的三角函數極限題目，使用羅必達可以迅速解出，但我們想利用函數的繪製來觀察  $x$  趨近於 0 時的極限。以下為四個函數與對應圖形：

$$y = \frac{\sin x}{x}$$

$$y = \frac{\sin x^2}{x}$$

$$y = \frac{(\sin 2x)^2}{x^2}$$

$$y = \frac{1 - \cos x}{x^2}$$

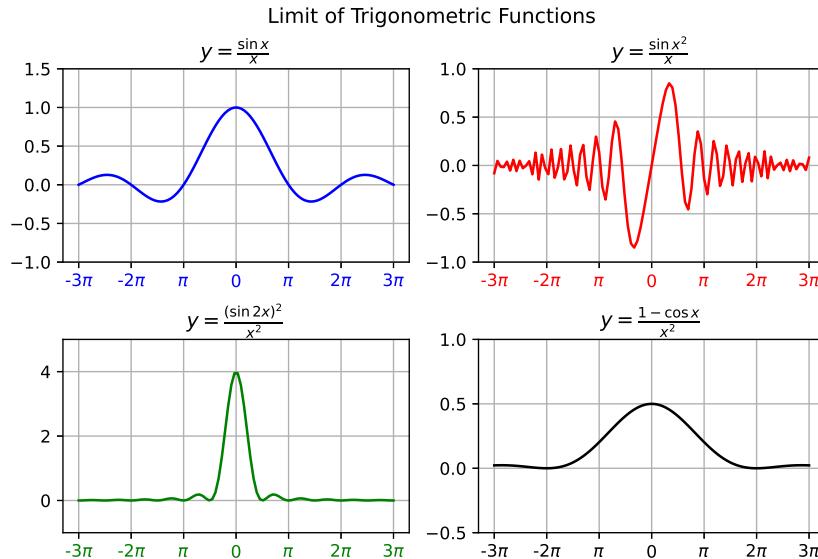


圖 2.1: 使用 subplot 展示四個函數圖形

藉由圖 2.1 我們可以看出四個函數的極限分別為:

$$\lim_{x \rightarrow 0} \frac{\sin x}{x} = 1$$

$$\lim_{x \rightarrow 0} \frac{\sin x^2}{x} = 0$$

$$\lim_{x \rightarrow 0} \frac{(\sin 2x)^2}{x^2} = 4$$

$$\lim_{x \rightarrow 0} \frac{1-\cos x}{x^2} = \frac{1}{2}$$



在繪製這題的時候，我們需要特別注意的有幾項：

- 透過 subplot 產生  $2/times2$  的子圖之後，若要在某一格繪圖，必須用括號去指定要在哪張圖作畫，參數是 m、n，也就是第 m 列、第 n 行 (m,n 從 0 開始)。
- 在設定 ax 變數後，要改變 x、y 軸的上下限、下標，需要加上 set。
- 在標題的部分子標題的語法與主標題有所不同。
- 如果要調整子圖之間的距離，可以用 vspace 或 hspace 調整間距。

範例程式碼如下：

```

ax[0, 0].plot(x, y, color='b')                      #ax.[m,
                                                       n]
ax[0, 0].set_ylim([-0.5, 1])                         #ax需要
                                                       加set_
ax[0, 0].set_xstick([...])
ax[0, 0].set_title('.....')                          #子標題
plt.suptitle('.....')                               #主標題
plt.subplots_adjust(hspace=0.4)                        #調整距
                                                       離

```

### 2.1.2 sigmoid 函數

第二題是在機器學習會看到的 sigmoid 函數，也叫做羅吉斯函數，在神經網絡中它經常被當作激活函數，尤其是在早期的神經網絡中。這題我們將畫出它的圖形，並透過更改不同的 a 值，來觀察圖形的走向是如何，以及這個函數的漸進線。

$$y = \frac{e^{\alpha x}}{1 + e^{\alpha x}}$$

此即 sigmoid 函數，圖形呈現 s 形，由圖可以看出這個函數 y 值會介於 0 ~ 1 的，也正因為如此，最常看到的應用情境，就是當我們在訓練模型做二分類的時候。或是我們將模型的最後一層神經網路設定為只有一個神經元，再將最後神經元所輸出的值輸入 sigmoid 函數，這樣一來，就會得到一個介於 [0,1] 之間的數值，即可進行分類。由於函數會介於 0 到 1 之間，因此兩條漸進線分別是  $y = 0$  與  $y = 1$ ，當  $x$  趨近無限大時  $y$  會靠近 1；當  $x$  趨近無限大時  $y$  會靠近 0，圖形與漸進線的繪圖如下：

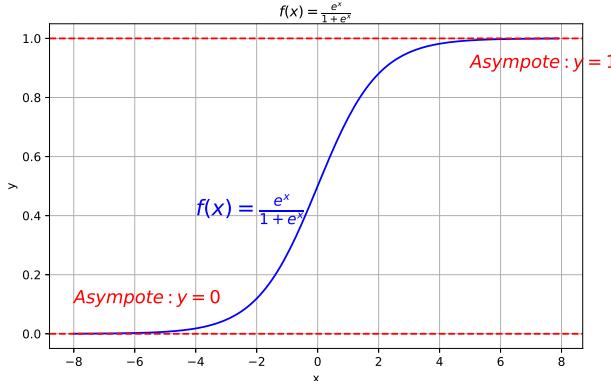
圖 2.2:  $\alpha = 1$  時的 Sigmoid 函數

圖 2.2 是當  $\alpha = 1$  時的結果，如果將  $\alpha = 2, 3, 4, 5$  的圖形一起畫出來，如圖 2.3，可以發現圖形隨著  $\alpha$  值增加，函數會更加陡峭，也就是在  $x = 0$  之間的變動會非常大。

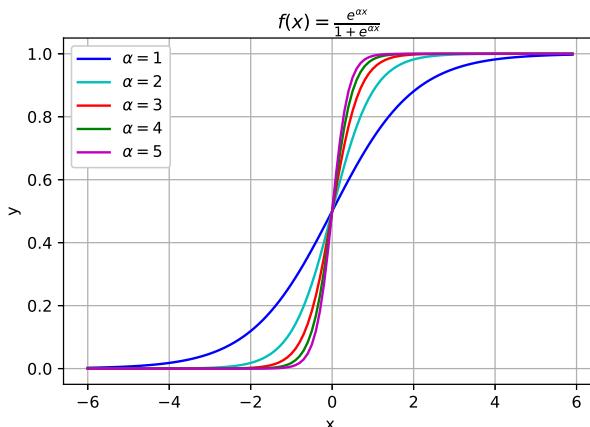
圖 2.3: 不同  $\alpha$  值的 Sigmoid 函數

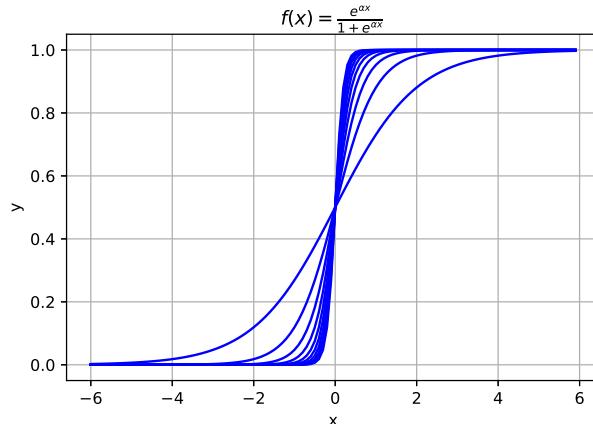
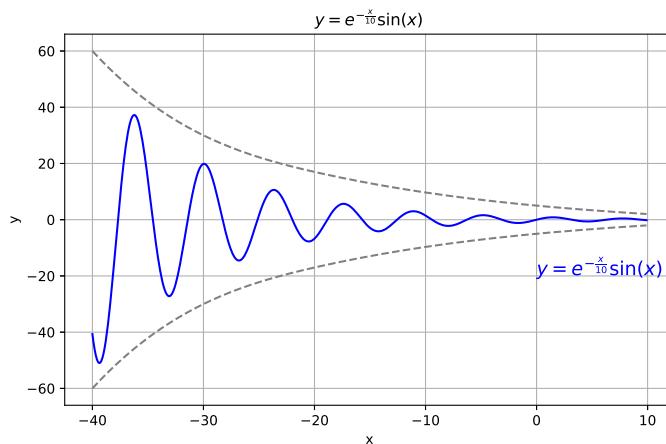
圖 2.3 是使用迴圈畫的，其中有幾個地方需要注意：

- 顏色要自己選取會比較麻煩，除非再寫一個隨機生成顏色的函數。
- 可以把標籤也用迴圈的參數表示，但前面須加上一個小  $f$ ，這樣使用 `plt.legend()` 就會產生五條線分別的標籤。

以下是附上迴圈的寫法：

```
for alpha in range(1,n+1):
    y = np.exp(alpha * x) / ( 1 + np.exp(alpha * x)
        )
    plt.plot(x, y, label = f'$\alpha={alpha}$',
              color = colors[alpha-1])
```

圖 2.4 是使用 `broadcast` 畫的，相較迴圈可以畫更多不同  $\alpha$  值，運算速度也較快，但顏色要改變相對就比較麻煩，如果只是需要看出趨勢，使用 `broadcast` 來畫圖會更快且程式碼更精簡。片段程式碼如下：

圖 2.4: 更多不同  $\alpha$  值的 Sigmoid 函數圖 2.5: 函數  $y = e^{-\frac{x}{10}} \sin(x)$  的走勢

```
alpha = np.arange(1, n+1, 1)
y = np.exp(x.reshape(-1, 1) * alpha) / (1 +
                                         np.exp(x.reshape(-1, 1) * alpha))
plt.plot(x, y, color = 'b', alpha = 0.5)
```

### 2.1.3 震盪函數

第三題我們要繪製的函數如下：

$$y = e^{-\frac{x}{10}} \sin(x)$$

這是一個指數函數與三角函數的結合函數，由於  $e^{-\frac{x}{10}}$  隨著  $x$  變大值會變小，因此使得後面的  $\sin(x)$  震盪幅度也隨著  $x$  值變大而變小，從而讓整個函數產生一個震盪衰減的趨勢，適合呈現這個特徵的  $x$  軸範圍大約介於-40 到 10 之間，函數在  $x$  超過 10 之後震盪幅度接近 0，圖形如下所示：



繪製此函數的時候步驟幾乎與前面相同，決定適合的  $x$  值域就好，但在呈現畫趨勢的兩條灰線時，有使用到別的語法，我使用的是 `scipy` 裡的 `make_interp_spline`，透過這個函數可以自選幾個座標點，讓它們連起來會是一條曲線，而不是直線，座標點需要多次測試，才能找到完全能符合震盪趨勢的曲線。曲線語法如下：

```
from scipy.interpolate import make_interp_spline
x = np.arange(-40, 10, 0.1)
a = np.array([-40, -30, -20, 0, 10])
b = np.array([-60, -30, -17, -5, -2])
curve1 = make_interp_spline(a, b)
ys=curve1(x)
plt.plot(x, ys, linestyle='--', color='gray')
```

## 2.1.4 定義域非實數的函數

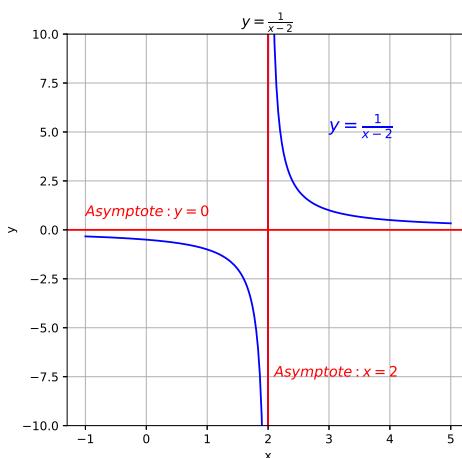
第四題要繪製的函數如下：

$$y = \frac{1}{x-2}$$

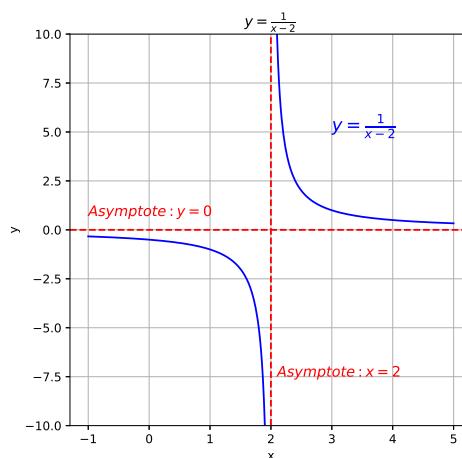
這個函數在 0 的時候沒有定義，所以我們在設定  $x$  的值域時，須把 2 排除在外，除了可以分成 -2 左邊一段與 -2 右邊一段外，也可以使用集合的概念，設定一段  $x$  的區間，並計算這個區間與 2 的差集，就能產生不包含 2 的一段區間。語法如下：

```
x = np.setdiff1d([np.linspace(-1, 5, 200)], [2])
```

`np.setdiff1d` 是差集的函數，透過這個函數我們可以產生正確的  $x$  的 domain，然後畫出下圖左邊的 (a) 圖：



(a) 未更改值域的圖形



(b) 更改值域後的圖形

圖 2.6:  $y = \frac{1}{x-2}$  與其漸進線

值得注意的是這樣做 `plot` 函數會把  $y$  值趨近於  $\infty$  與  $-\infty$  的兩點連起來，但實際上的圖形是不會連起來的，如果要解決的話似乎只能做兩



次圖，一次是  $x > 2$  的圖，一次是  $x < 2$  的圖，最後畫出如圖 2.6 右邊的圖。

### 2.1.5 多項式函數與反函數

第五題要繪製的函數為：

$$y = x^3 + 2$$

除此之外要加上這個函數的反函數，而我們可以知道一個函數與它的反函數會對稱於  $y = x$  這條斜直線，因此我們在繪製完兩個函數之後會再加上他們的對稱線與交點，交點透過解方程式可以算出來，兩函數交於  $(-1.54, -1.54)$ ，最後呈現的圖形如下，加上對稱線後像一隻三叉戟：

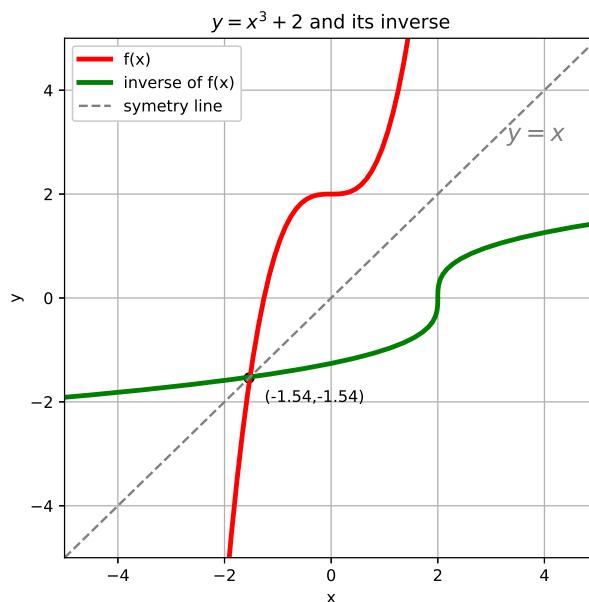


圖 2.7: 函數  $y = x^3 + 2$  與它的反函數

這題為了清楚的呈現兩者對稱的圖形，需要把  $x$  的值域設置的窄一點，因為三次方程式的函數值膨脹的非常快，設置太寬可能看不出變化。另外，繪製反函數非常容易，只需要把  $x$  與  $y$  順序對調就能輕鬆畫出反函數，不需要特地去算反函數。程式碼如下：

```
plt.plot(x , y , label = 'f(x)' , color = 'r' ,
         linewidth=3)
plt.plot(y , x , color = 'g' , label = 'inverse
          of f(x)' , linewidth=3)
plt.scatter(-1.54,-1.54,color='black')
```

在進行描點的時候要用到的語法是 `plt.scatter`，裡面有 `marker`、`color`、`size` 等參數可以控制點的形狀、大小與顏色，`marker` 默認的點會是實心點，所以這題不更改 `marker` 的參數，至於 `size` 也覺得差不多，所以就不做更改。



## 2.1.6 常態分配的機率密度函數

第六題要呈現的函數是常態分配的機率密度函數：

$$y = \frac{1}{\sqrt{2\pi}} e^{\frac{-(x-3)^2}{2}}$$

由函數可以明顯看出是一個  $\mu = 3, \sigma^2 = 2$  的常態分配，因此在決定 x 軸範圍的時候要把  $x = 3$  放在圖的正中央，以呈現常態分配對稱的情形。另外，我更改 x 軸的座標，讓 x 的座標是以距離平均數幾個標準差的形式呈現，因為使用常態分配不外乎就是檢定某筆資料是否距離平均數很遠，透過把 x 的座標更改，可以更清楚呈現某筆資料是否距離平均數很遠，以及距離幾個標準差。最後圖形呈現如下：

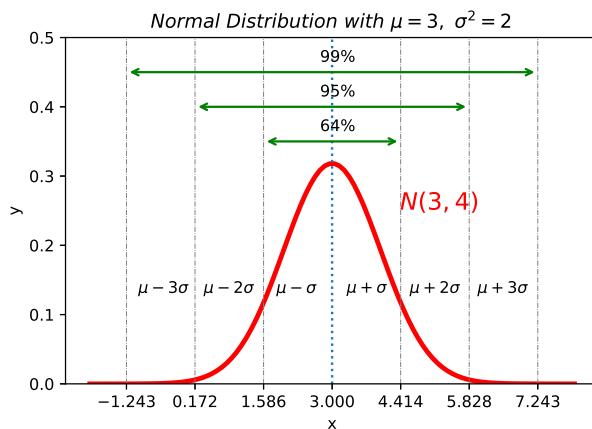


圖 2.8: 函數  $y = \frac{1}{\sqrt{2\pi}} e^{\frac{-(x-3)^2}{2}}$

這題在繪出主圖形時與前面過程大致相同，但在繪製區隔線的時候，我希望一次全部畫出來，於是使用迴圈畫這六條鉛直線，過程中我原本想要更改直線的顏色，需要讓迴圈跑兩個變數，所以用了 zip 的語法，讓 x 值與顏色值可以一起同時變動，但最後礙於顏色太多很雜亂還是統一灰色，下面是迴圈程式碼：

```

x_values = [avg-3*sigma, avg-2*sigma, avg-1*
            sigma, avg+sigma, avg+2*sigma, avg+3*sigma]
colors = ['gray', 'gray', 'gray', 'gray', 'gray',
          'gray']
for x, color in zip(x_values, colors):
    plt.axvline(x=x, color=color,      linestyle
                ='-.', linewidth=0.5)

```

繪製完成圖形後為了讓讀者了解這幾條線的意義，於是使用 `plt.text` 在圖上標註是距離平均數幾個標準差，最後再把  $N(3,4)$  加上就完成繪製了。



## 2.1.7 多項式函數的極值與微分

這題的目標是繪製一個簡單的多項式函數，並找出這個函數的最大值與它的根，函數如下所示：

$$y = 3x^3 - x^4$$

這題可以透過一階微分計算找出最大值的位置，而  $y = f(x)$  的根可以也透過簡單的數學解出來，同時兩者要用 **python** 的程式碼去解出來也是行得通的。下面是解一階微分程式碼：

```
import sympy as sp
X = sp.symbols('X')
F = 3*X**3-X**4
df = sp.diff(F, X)
solutions = sp.solve(df, X)
for sol in solutions:
    print(f'X = {sol}, F'(X) = 0')
```

透過 **Sympy** 套件，設定好原函數以及微分的對象之後，塞到 **sp.diff** 裡就會產生一階導函數，再將導函數拿去 **solve**，就能計算出解。而由於解不只一個，因此用迴圈把結果 **print** 出來。

得到結果後一樣使用 **plt.scatter** 來把點描上去，但特別的是因為要區別最大值與根的點，所以把兩個不同的點使用 **label** 做標籤，最後加上 **legend** 後就可以得到下圖左方的結果：

```
fig, ax = plt.subplots(1, 2, figsize=(12, 4))
ax[0].plot(x, f(x), color='b')
ax[0].scatter(x_roots, y_roots, color='red',
               marker='o', label='roots')
ax[0].scatter(x_max, y_max, color='black',
               marker='o', label='maximum')
```

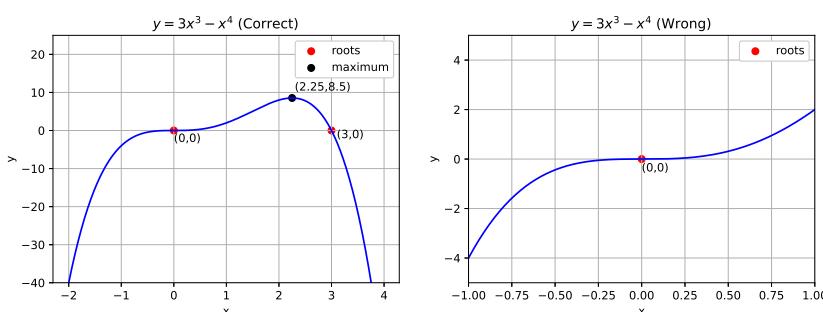


圖 2.9: 函數  $y = 3x^3 - x^4$ ，正確與錯誤示範

圖 2.9 的右圖是一個錯誤示範，因為沒有設定好  $x$  的值域，所以圖最後只看得出一個解，為避免此種情況發生最好多嘗試幾種  $x$  的值域，等大致上確定圖形趨勢後，再選擇一個相對適合的值域，否則無法完整呈現函數圖形。



## 2.1.8 常見極限函數

這題的函數常被用來作為極限的題目，用羅必達法則可以很快的解出來，函數如下所示：

$$y = \frac{\ln x}{x^2}$$

透過羅必達法則我們可解出：

$$\lim_{x \rightarrow 0} \frac{\ln x}{x^2} = -\infty$$

$$\lim_{x \rightarrow \infty} \frac{\ln x}{x^2} = 0$$

所以我們可以知道這個函數會存在著兩條漸進線，分別是  $x = 0$  與  $y = 0$ ，將漸進線與函數圖形一同繪出後，我們還需計算出最大值的位置，透過上一節 Scipy 的套件可以計算出來，最後標上最大值的點就能產生下圖：

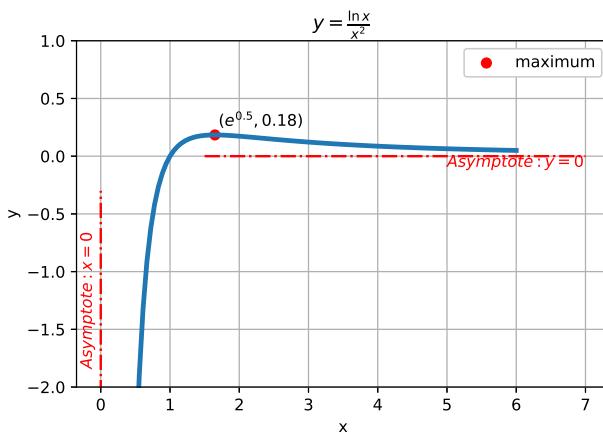


圖 2.10: 函數  $y = \frac{\ln x}{x^2}$  與其漸進線

這個圖形在設定  $x$  的值域的時候要當心，起始點不能離 0 太近，一開始我設定的範圍在 0.01 到 7 之間，但畫出來後發現，0.01 會使得  $y$  值過大，導致在標 text 的時候無法標到正確的位置上，於是最後更改到 0.7 到 7，讓函數圖形可以在保有其趨勢的情況下，還能正確顯示。另外，在 plt.text 的參數中，有一個 rotation 可以調整，指定數字就可以讓文字旋轉指定的度數，例如 rotation=90，就是讓文字由水平變成垂直，其預設是 0，所以文字都會以水平顯示，語法如下與部分繪製的程式碼如下：

```
fig, ax = plt.subplots()
ax.plot(x, f(x), linestyle='-', linewidth=3)
ax.hlines(y=0, xmin=1.5, xmax=7, linestyles='--',
           colors='r')
ax.vlines(x=0, ymin=-3, ymax=-0.3, linestyles
           ='--', colors='r')
ax.set_ylim([-2, 1])
ax.scatter(x_max, y_max, color='r', label='maximum')
```



### 2.1.9 水平分段函數

第九題是一個分段函數，它的值域與對應函數值如下：

$$f(x) = \begin{cases} 1, & 1 \leq x < 3 \\ 2, & 3 \leq x < 5 \\ 3, & 5 \leq x < 7 \end{cases}$$

這題的繪圖相對來說簡單，需要的注意是有等號的一邊需要加上實心點，沒有等號的部份加上空心點，其中控制空心與實心的參數是 `facecolors`，當 `facecolors=none`，就會畫出一個空心的點。以下是部分繪圖的程式碼與繪製完成的圖形：

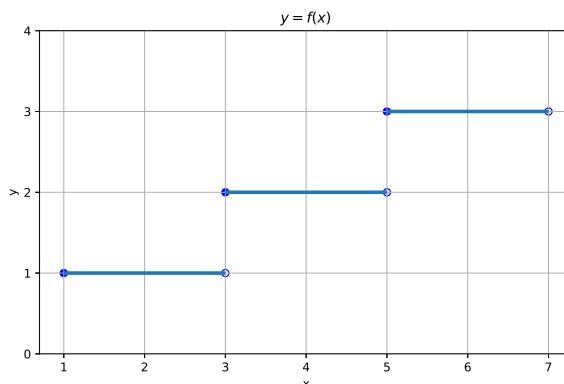


圖 2.11: 函數  $y = f(x)$

```
ax.hlines(y=1 ,xmin=1 ,xmax=3,linewidth=3)
ax.hlines(y=2 ,xmin=3 ,xmax=5,linewidth=3)
ax.hlines(y=3 ,xmin=5 ,xmax=7,linewidth=3)
ax.scatter(x1_point ,y1_point ,marker='o',color ='blue')
ax.scatter(x2_point ,y2_point ,marker='o',color ='blue', facecolors='none')
```

### 2.1.10 圓形

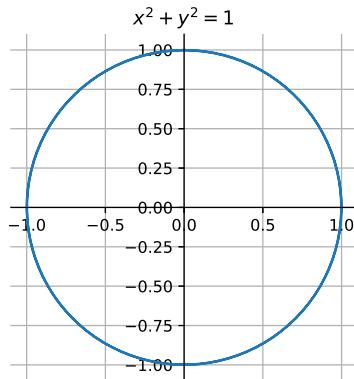
第十題要畫的函數是一個半徑為 1，圓心在  $(0,0)$  的圓：

$$x^2 + y^2 = 1$$

畫圓有很多方法，首先介紹第一個，使用極座標表示：

先把  $\theta$  範圍定義出來，再將  $x$  設為  $\sin(\theta)$ ， $y$  設為  $\cos(\theta)$ ，這時候將  $x,y$  放入 `plot` 函數就能畫出一個圓。以下為部分程式碼與繪製的圖：

```
theda = np.linspace(-2*np.pi , 2*np.pi,100)
x = np.sin(theta)
y = np.cos(theta)
fig, ax = plt.subplots()
plt.plot(x,y)
```

圖 2.12: 函數  $x^2 + y^2 = 1$ 

第二個方法可以使用 `matplotlib` 內建的函數，也就是 `matplotlib.patches`，透過這個套件可以直接指定半徑與圓心，不需要設定值域就能將圓畫出來，也可以指定要不要填滿圓以及圓的顏色，而最後的繪圖結果會與圖 2.12 相同，部分程式碼如下：

```
import matplotlib.patches as ptc
circle = ptc.Circle((0, 0), radius=1, fill=False
                     , color='blue')
fig, ax = plt.subplots()
ax.add_patch(circle)
```

第三個方法使用的是隱函數的方法，也就是將方程式表示成：

$$x^2 + y^2 - 1 = 0$$

先將  $x$ 、 $y$  分別設置為兩個一維值域，再透過 `numpy` 裡的 `meshgrid` 讓  $x, y$  由分別的一維數據轉換成結合的二維數據，最後透過等高線的函數 `plt.contour`，把符合這個隱函數的點繪製上去，連成等高線，就能繪製出圓。

其中在 `contour` 中有一個 `level` 參數可以控制隱函數的值，也就是讓隱函數等於多少，這題我們希望隱函數  $x^2 + y^2 - 1 = 0$ ，所以將 `level` 設置為 0，於是最後也可以畫出與圖 2.12 相同的圖，部分程式碼如下：

```
x = np.linspace(-1, 1, 200)
y = np.linspace(-1, 1, 200)
x, y = np.meshgrid(x, y)
circle = x**2 + y**2 - 1**2
plt.contour(x, y, circle, levels=[0], colors='blue')
```

繪製出圓後，為了要符合老師講義的座標軸，以及把座標軸設置為正方形，我們還需要加上幾行程式碼：

```
ax.spines['top'].set_visible(False)
```



```
ax.spines['right'].set_visible(False)
ax.spines['left'].set_position(('data', 0))
ax.spines['bottom'].set_position(('data', 0))
plt.axis('equal')
```

`ax.spines` 可以把上、下、左、右的邊界進行位移或隱藏，位移使用 `set_position`，隱藏則使用 `set_visible`，最後一行則是將座標軸設置為等寬，透過這幾行程式碼，就能將圖片調整成與老師講義相同的格式。

### 2.1.11 正方形

第十一題要繪製的是一個正方形，正方形一樣有不少方法可以畫出來，首先介紹第一個，也是用 `matplotlib.patches` 直接畫出來，只要指定左下的頂點與長、寬，就能產生一個正方形，再把座標軸調整一下就能畫出與講義相同的圖形，程式碼與圖形如下：

```
fig, ax = plt.subplots()
square = ptc.Rectangle((-0.5, -0.5), 1, 1,
                       linewidth=2, edgecolor='blue', facecolor='none')
ax.add_patch(square)
```

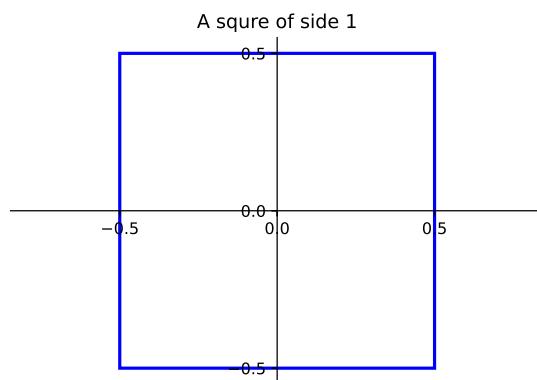


圖 2.13: 邊長為 1 正方形

第二種則是把頂點座標挑出來，形成一個陣列，再讓 `plot` 函數把他們連起來，值得注意的是起始點與終止點必須一樣，這樣就能畫出與圖 2.13 一樣的圖，部分程式碼如下：

```
x = [-0.5, 0.5, 0.5, -0.5, -0.5]
y = [-0.5, -0.5, 0.5, 0.5, -0.5]
plt.plot(x, y, color='blue')
```

最後一種是用 `hline` 與 `vline` 畫出兩條垂直線與兩條水平線，讓四條線為成一個正方形，部分程式碼如下：



```
plt.vlines(x=[-0.5, 0.5], ymin=-0.5, ymax=0.5,
            color='blue')
plt.hlines(y=[-0.5, 0.5], xmin=-0.5, xmax=0.5,
            color='blue')
```

## 2.2 專題

$$\text{令 } S_n = \sum_{k=1}^n \frac{1}{k} = 1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \cdots + \frac{1}{k}$$

1. 驗證  $\lim_{n \rightarrow \infty} S_n$  發散
2. 令  $\gamma_n$  為圖 2.14 的陰影處面積。驗證  $\gamma_n = S_n - \ln(n+1)$  (可以直接證明)
3. 驗證  $\frac{1}{2}(1 - \frac{1}{n+1}) < \gamma_n < 1$

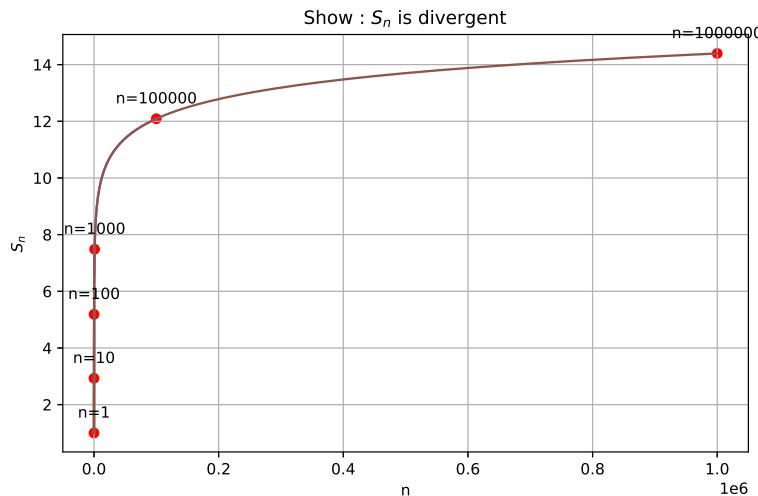
### 2.2.1 第一小題

雖然微積分教過我們  $S_n$  這個級數和會發散，但我們在紙上始終無法算出當  $n$  很大的時候， $S_n$  到底是多少，但借用電腦我們可以找出來，利用單迴圈我們可以計算  $S_n$  的值，外面再加一層迴圈，可以觀察隨著  $n$  越來越大， $S_n$  的情況，圖形與程式碼如下：

```
ns = [1, 10, 100, 1000, 10**5, 10**6] ## 加到多少
result = [] ## 預先留空間
for n in ns: ## 把指定的n都做
    Sn = 0
    Sn_values = []
    for k in range(1, n + 1): ## 級數計算
        Sn = Sn + 1 / k
        Sn_values.append(Sn) ## 把舊值覆蓋
    result.append(Sn_values) ## 把結果放進 []

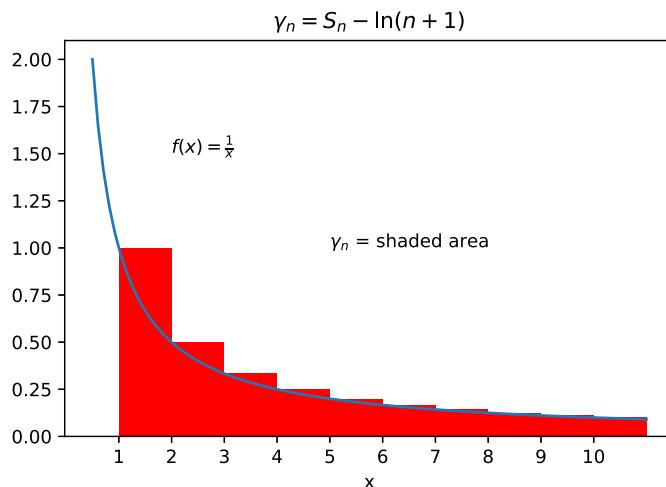
plt.figure(figsize=(8, 5))
for i in range(len(ns)): ## 總共畫6次
    ## 把每次x,y的點都用i來表示寫出迴圈
    plt.plot(range(1, ns[i] + 1), result[i])

## 取指定的n與對應的Sn出來，點在圖上
for i, n in enumerate(ns):
    ## result取每次畫圖出來的最後一項，當作縱軸值
    plt.scatter(n, result[i][-1], c='red')
    ## 標籤n={n}要放在比指定點高10單位的地方，並置
    ## 中
    plt.annotate(f'n={n}', (n, result[i][-1]),
                textcoords="offset points", xytext=(0, 10),
                ha='center')
```

圖 2.14:  $S_n$  隨著  $n$  的變化

從圖中我們可以看出雖然  $S_n$  值增加的速度有趨緩，但始終找不到一條漸進線可以趨近，即便  $n$  到了 1000000，還是有在慢慢增加的趨勢，因此可以知道這個級數是發散的。

## 2.2.2 第二小題

圖 2.15:  $\gamma_n = \text{sum of the red areas}$ 

第二題由圖 2.15 可知， $\gamma_n$  的值是所有長條面積加起來減掉  $y = \frac{1}{x}$  從 1 積到  $n + 1$  的積分值，其中長條圖面積為：

$$1 \times 1 + 1 \times \frac{1}{2} + 1 \times \frac{1}{3} + 1 \times \frac{1}{4} + \cdots + 1 \times \frac{1}{n} = S_n$$

$y = 1/x$  從 1 積到  $n + 1$  的積分值積分值為：

$$\int_1^{n+1} \frac{1}{x} dx = \ln(n+1) - \ln(1) = \ln(n+1)$$



因此可得證：

$$\gamma_n = S_n - \ln(n + 1)$$

在畫圖 2.15 時，要先把長條圖與函數  $y = \frac{1}{x}$  先畫好，再來要蓋掉曲線下的長條圖，也就是把  $y = \frac{1}{x}$  與  $x$  軸圍出的面積填上白色，使用 `fillbetween` 語法就能做到，繪圖程式碼如下所示：

```
a, b, n = 1, 10, 10
x1 = np.linspace(a, b, n)
x2 = np.linspace(0.5, 11, 100)
y1 = 1 / x1
y2 = 1 / x2

plt.bar(x1, y1, 1, align='edge', color='red',
        alpha=0.5)
plt.plot(x2, y2)
plt.fill_between(x2, y2, 0, where=y2 > 0,
                 color='white')
plt.xlabel('x')
plt.xticks(range(1, 11, 1))
plt.title('$\gamma_n = S_n - \ln(n+1)$')
plt.text(2, 1.5, '$f(x) = \frac{1}{x}$')
plt.text(5, 1, '$\gamma_n$ = shaded area')
plt.savefig("pj2.eps", format="eps", dpi=300)
plt.show()
```

### 2.2.3 第三小題

第三小題使用圖形比較，分別繪製三條函數的圖形：

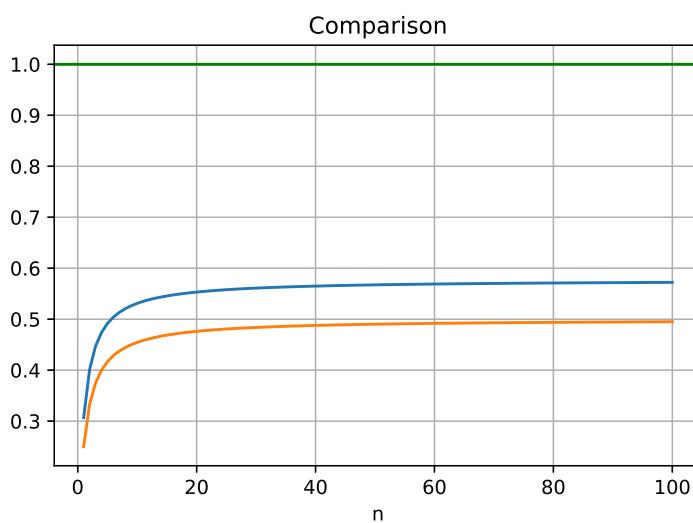


圖 2.16:  $\frac{1}{2}(1 - \frac{1}{n+1}) < \gamma_n < 1$

由圖片可以清楚看出  $\frac{1}{2}(1 - \frac{1}{n+1}) < \gamma_n < 1$ 。這題的  $S_n$  改用第二種方法，



使用 `cumsum` 來計算  $S_n$ ，也可以算出級數和，另一方面這樣呈現  $\gamma_n$  也比較直觀，附上繪圖程式碼：

```
n= np.arange(1, 101)
S_n = np.cumsum(1 / n)
gamma_n = S_n - np.log(n + 1)
y_n = 1/2*(1-1/(n+1))

plt.plot(n, gamma_n, label='$\gamma_n = S_n - \ln(n+1)$')
plt.plot(n, y_n, label='$1/2*(1-1/(n+1))$')
plt.axhline(1, xmin=0, xmax=100, label='1', color='green')

plt.xlabel('n')
plt.title('Comparison')
plt.grid(True)
plt.savefig("pj3.eps", format="eps", dpi=300)
plt.legend()
plt.show()
```

## 2.3 結論

本文透過繪製函數圖形，介紹了許多 python 的基礎語法，包括 `numpy`、`matplotlib` 套件等，但在繪製圖形時，不只要確保程式碼順利執行，也要讓讀者清楚看見圖形所要表達的主旨，所以註解與圖形範圍選擇也是很重要的一部分，希望藉由本文，讀者對使用 `python` 繪圖能有更深的了解。





## 第 3 章

# 常用分配與亂數產生

本文旨在使用 python 繪製不同的分配函數圖形，並驗證抽樣分配中，理想與實際狀況的差異。相較於過去在數理統計課本的探討的純數學算式，本文將著重討論不同分配實際的形狀、參數對函數的影響與驗證抽樣分配理論，希望透過視覺化的圖形，讓讀者更了解分配的特性。

### 3.1 離散型機率分配的圖形

以下將會繪製數個離散型分配的機率質量函數圖形。

#### 3.1.1 伯努力分配

伯努力分配是離散型分配中最簡單的分配，只有一個參數  $p$ ，樣本觀察值也只有 0,1，以下是的機率質量函數與圖形：

$$P(X = x) = p^x(1 - p)^{1-x}$$

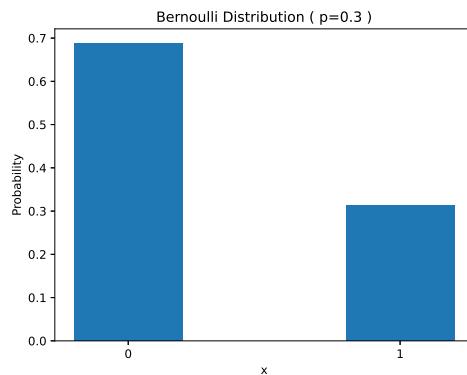


圖 3.1: 伯努力分配 ( $p = 0.3$ )



### 3.1.2 二項分配

二項分配為伯努力分配的相加，因此參數除了  $p$  之外，還多了樣本數  $n$ ，而  $x$  代表的則是成功次數，以下是它的機率質量函數與不同參數下的圖形：

$$P(X = k) = \binom{n}{k} p^k (1-p)^{n-k}$$

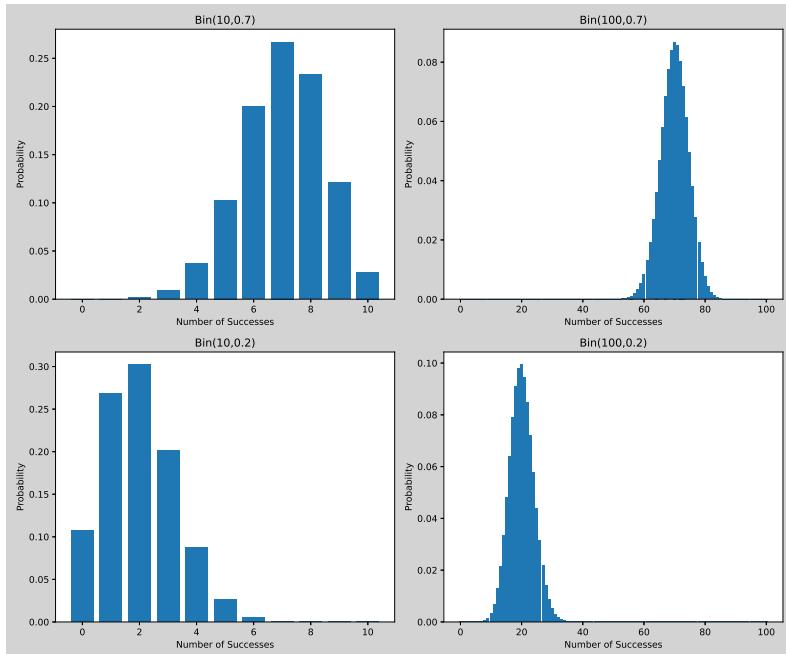


圖 3.2: 不同參數下二項分配的圖形

從圖 3.2 的上左圖與上右圖可以發現，改變  $n$  的話分配形狀不變，但整體圖形會右移，而且將樣本增加 10 倍，可以看出圖形原本中心點的中心點會一致原本中心點 10 倍的地方，驗證了數理統計上  $\mu = np$  的性質。

再來，從圖 3.2 的上左圖與下左圖可以發現，改變  $p$  的話分配形狀會改變，大致呈現以  $x=5$  為中線，左右對稱的情況。

最後我們看到圖 3.2 的上右圖與下右圖，兩者趨近於一個鐘型分配。為了更精確的驗證二項分配當樣本數夠大時，會趨近常態分配，所以在畫了一張圖來呈現此性質，圖如下：

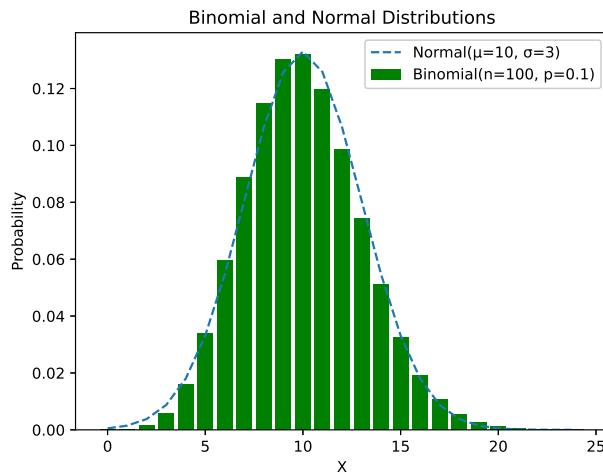


圖 3.3: 二項分配的圖形與近似的常態分配

從圖 3.3 可以看出二項分配的圖形與常態分配的函數圖形幾乎一致，成功驗證了二項分配當樣本數夠大時，會趨近常態分配的性質。

### 3.1.3 卜瓦松分配

卜瓦松分配的參數只有  $\lambda$  表示一段時間內事件發生的頻率， $x$  則表示一段時間內事件發生的次數，其機率質量函數與  $\lambda = 10$  的圖形如下：

$$P(X = k) = \frac{e^{-\lambda} \lambda^k}{k!}$$

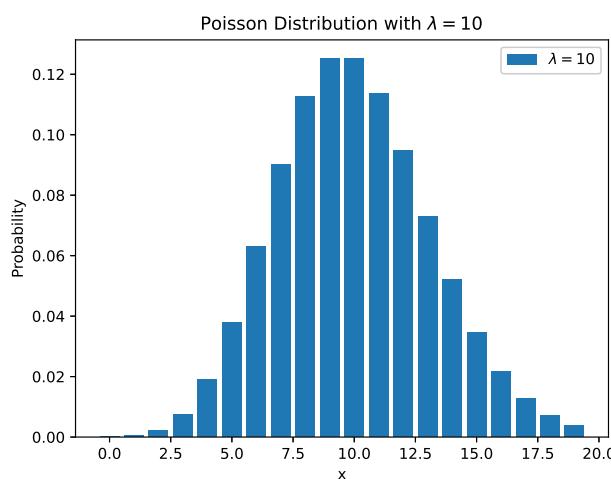


圖 3.4: 卜瓦松分配分配的圖形



### 3.1.4 幾何分配

幾何分配的參數只有  $p$ ，表示成功的機率， $x$  則表示成功一次所需次數，其機率質量函數與  $p = 0.3, p = 0.6$  的圖形如下：

$$P(X = k) = (1 - p)^{k-1} p$$

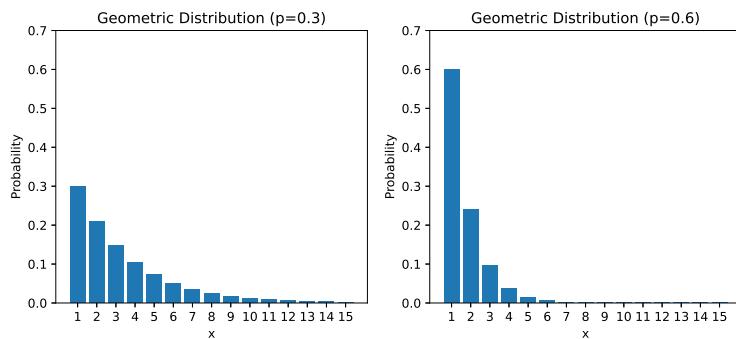


圖 3.5: 不同參數的幾何分配圖形

從圖 3.5 可以看出當  $p$  越大時，分配的圖形越陡， $p$  越小時，分配的圖形則較平緩。

### 3.1.5 負二項分配

負二項分配為幾何分配的相加，參數有  $p$  (成功機率)、 $r$  (成功次數)， $x$  表示的則是成功  $r$  次所需花費的次數，其機率質量函數與不同參數的圖形如下：

$$P(X = k) = \binom{k+r-1}{k} p^k (1-p)^r$$

在畫圖之前，我們知道負二項分配為幾何分配的相加，因此推論其走向會與幾何分配類似，也就是當  $p$  越大時，分配的圖形越陡。而  $r$  改變的會是圖形的位置，因為幾何分配的平均數是  $\frac{1}{p}$ ，負二項分配則是  $\frac{r}{p}$ ，因此不難猜測增加  $r$  會使的圖形向右移。

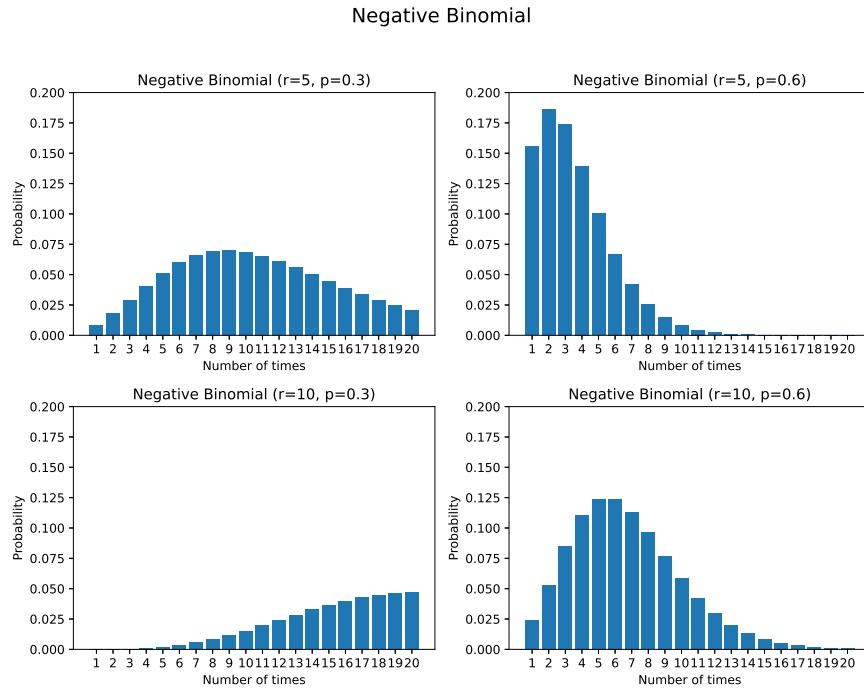


圖 3.6: 不同參數的負二項分配圖形

從圖 3.6 的左上圖與右上圖可以看出，與我們推論的一樣，當  $p$  增加  $r$  保持不變時，圖形會變得更陡。

從圖 3.6 的左上圖與左下圖可以看出，其中心位置往右偏移，這也與我們猜測的一樣，當  $r$  增加時  $p$  保持不變時，圖形會向右移動。

### 3.1.6 離散均勻分配

離散均勻分配也是在離散分配中相對簡單的分配，參數有  $a$ 、 $b$ ，分別為整數並決定上下界，其圖形與機率質量函數如下：

$$P(X = k) = \frac{1}{b - a}$$

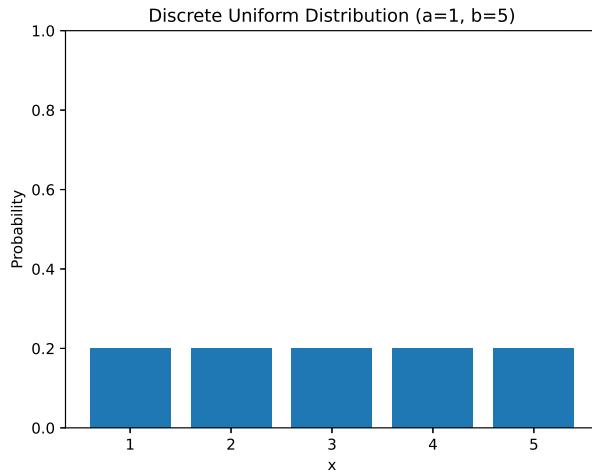


圖 3.7: 離散均勻分配 (1,5)

### 3.1.7 超幾何分配

超幾何分配的參數有三個， $N$  表總樣本數， $K$  樣本中符合特定描述的樣本數， $n$  表抽取數目，其不同參數下的圖形與機率質量函數如下：

$$P(X = k) = \frac{\binom{K}{k} \binom{N-K}{n-k}}{\binom{N}{n}}$$

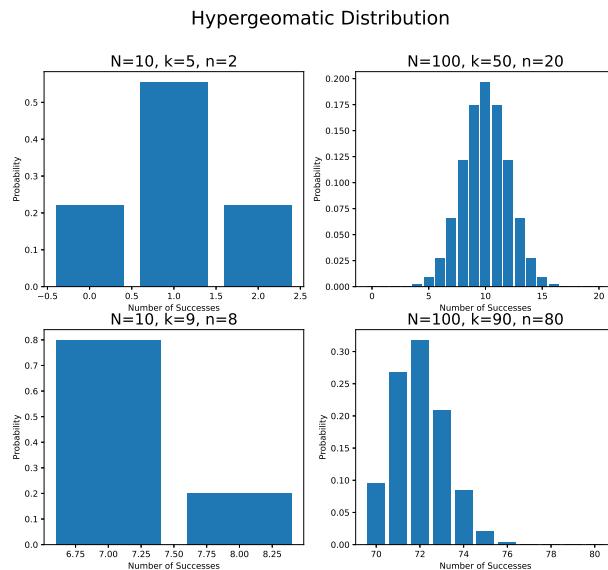


圖 3.8: 超幾何分配不同參數下的圖形

圖 3.8 右上圖的參數是左上圖的 10 倍，我們可以發現當把參數都乘以



10，其分配形狀是不會改變的，兩者都是對稱分配。但當我們把  $K$ 、 $n$  改變，可以看到分配會從對稱變為右偏圖形(上右圖與下右圖)。

## 3.2 連續型機率分配的圖形

以下將會繪製數個連續型分配的機率質量函數圖形。

### 3.2.1 指數分配

指數分配的參數有  $\lambda$ ，分配多用來描述事件間的時間間隔，其機率密度函數與不同參數的圖形如下：

$$f(x; \lambda) = \lambda e^{-\lambda x}$$

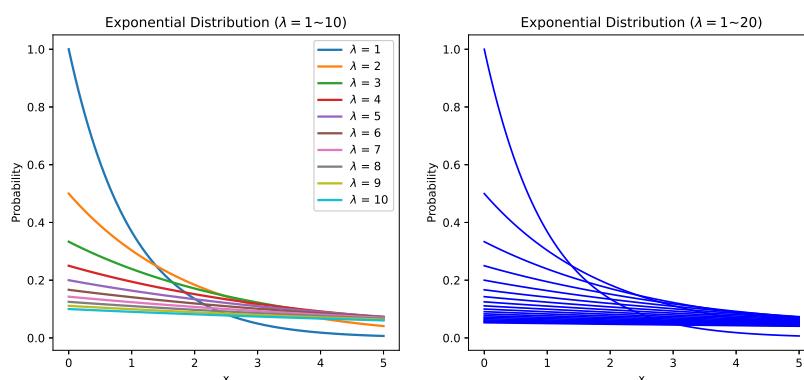


圖 3.9: 指數分配不同參數的圖形

從圖 3.9 可以看出來，當  $\lambda$  增加時，圖形會逐漸變得平緩，但不管  $\lambda$  值如何，其機率密度函數都會隨著  $x$  增加而逐漸趨近於 0。

### 3.2.2 迦馬分配

迦馬分配有兩個參數， $\alpha$  為形狀參數， $\theta$  為尺度參數，其中  $\theta$  與指數分配中的  $\lambda$  相同，兩者都是屬於 Poisson family，其機率密度函數與不同參數的圖形如下：

$$f(x; \alpha, \beta) = \frac{\theta^\alpha}{\Gamma(\alpha)} x^{\alpha-1} e^{-\theta x}$$



Gamma Distribution

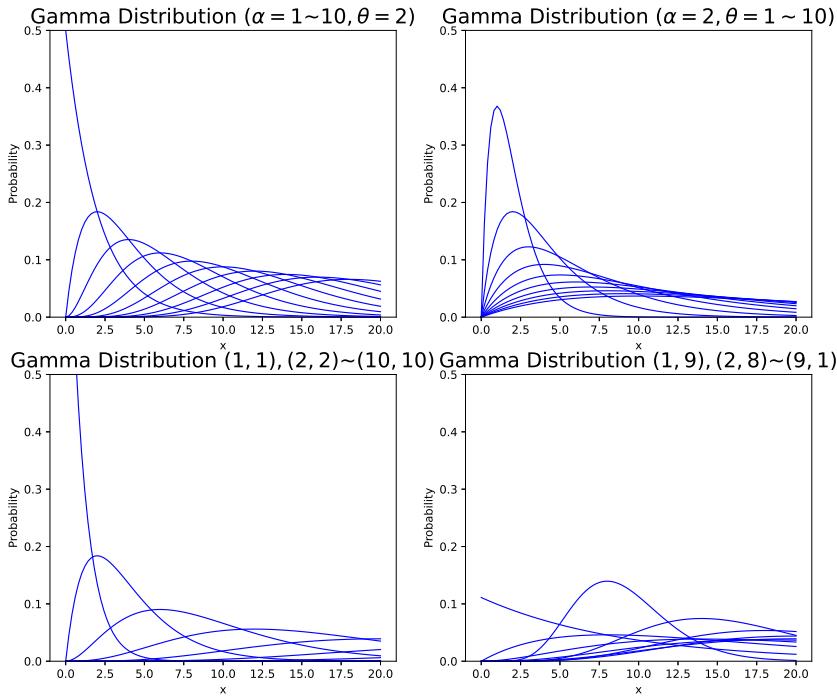


圖 3.10: 迦馬分配不同參數的圖形

圖 3.10 左上圖可以看出固定  $\theta$  時，改變  $\alpha$ ，會讓分配形狀從右偏到對稱再到左偏，符合其形狀參數的名字。右上圖則是固定  $\alpha$  改變  $\theta$ ，其變化的趨勢是隨著  $\theta$  增加，圖形會逐漸平緩，與指數分配的變化趨勢雷同。

圖 3.10 的左下圖與右下圖呈現的是兩個參數同時改變的狀況，兩個參數同時增加時，位置與平緩度都會同時改變；一個增加一個減少時則看不太出變化趨勢。



### 3.2.3 貝塔分配

貝塔分配有兩個參數， $\alpha$ 、 $\beta$ ，兩者都是形狀參數，而  $x$  的值域界於  $0 \sim 1$ ，其機率密度函數與不同參數的圖形如下：

$$f(x; \alpha, \beta) = \frac{1}{B(\alpha, \beta)} x^{\alpha-1} (1-x)^{\beta-1}$$

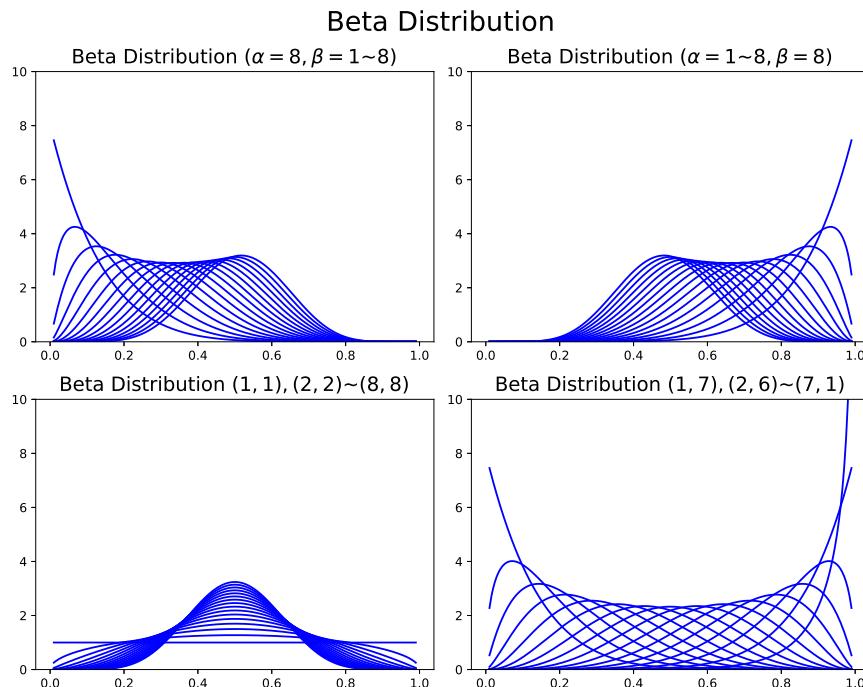


圖 3.11: 貝塔分配不同參數的圖形

由圖 3.11 上左圖與上右圖可以看出當  $\alpha > \beta$  時，分配會右偏； $\alpha < \beta$  時，分配會左偏；當  $\alpha = \beta$  時會是對稱分配。

當  $\alpha = \beta$  時，隨著兩個參數一起增加，圖形的峰態會增加，也就是中間的區域佔的機率會較大。值得注意的是，從圖 3.11 左下圖可以看到有一條水平線，是  $\beta(1,1)$  的圖形，我們過去學過  $\beta(1,1) = U(0,1)$ ，這個圖形也確實驗證了這樣的結果。



### 3.2.4 t 分配

t 分配只有一個參數  $df = v$ ，其分配形狀與常態分配類似，我們知道當  $df$  足夠大的時候，t 分配會漸進  $N(0,1)$ ，除此之外，當  $v = 1$  時，也剛好會是  $\text{Cauchy}(0,1)$ ，下面是驗證這兩個性質的結果與 t 分配的機率密度函數：

$$f(t; v) = \frac{\Gamma(\frac{v+1}{2})}{\sqrt{v\pi} \Gamma(\frac{v}{2})} \left(1 + \frac{t^2}{v}\right)^{-\frac{v+1}{2}}$$

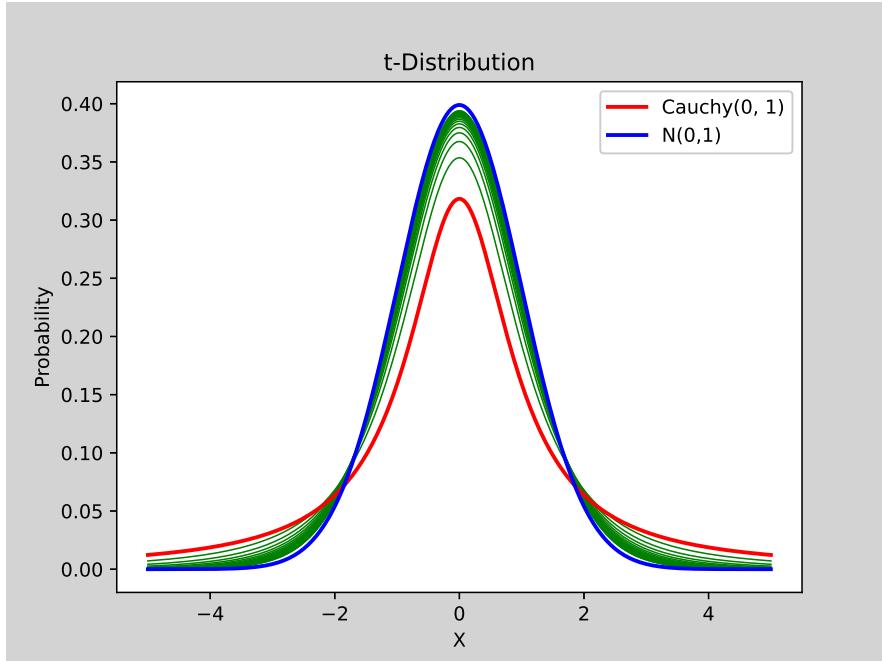


圖 3.12: t 分配不同參數的圖形

由圖 3.12 可以看出，t 分配的變動範圍會落在  $\text{Cauchy}(0,1)$  與  $N(0,1)$  之間，隨著  $v$  變大，t 分配的雙尾會逐漸縮小，機率分配集中在 0 周圍，而且看得出  $v = 0 \sim 4$  的時候變化幅度最大，後面逐漸縮小變化幅度。



### 3.2.5 卡方分配

卡方分配只有一個參數  $df = k$ ，我們也知道它即是  $Gamma(\frac{k}{2}, 2)$ ，因此不難猜測會與迦馬分配，當  $\alpha$  固定不動改變  $\theta$  的變動一樣，由右偏往左偏變化。下面是機率分配函數與不同參數的卡方分配圖形：

$$f(x; k) = \frac{1}{2^{\frac{k}{2}} \Gamma(\frac{k}{2})} x^{\frac{k}{2}-1} e^{-\frac{x}{2}}$$

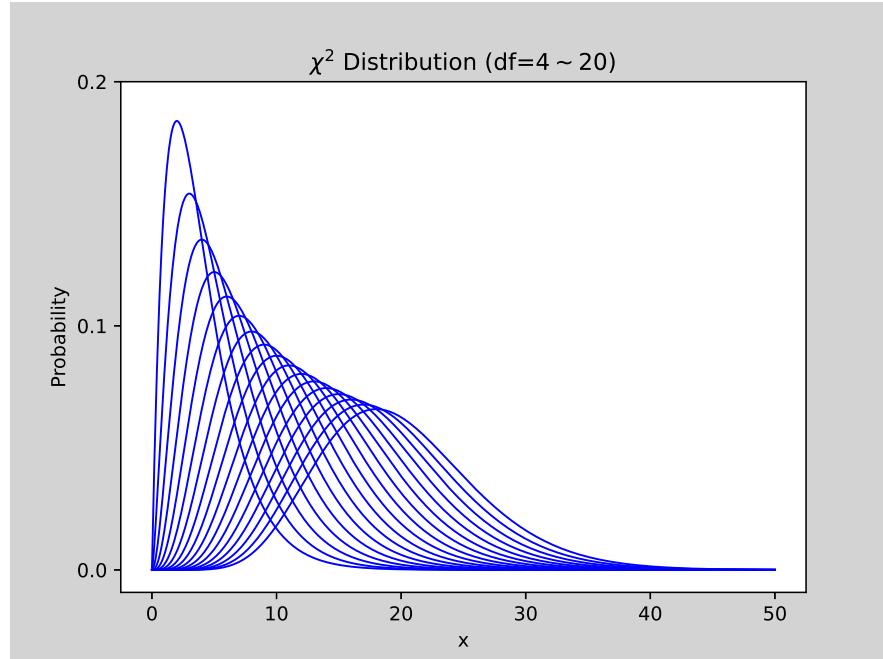


圖 3.13: 卡方分配不同參數的圖形

從圖 3.13 我們可以知道，為何卡方自由度增加時，臨界值也會跟著增加，因為分配漸漸往右移動，使得臨界值也隨著往右移動。另外，卡方分配不像 t 分配一樣變動會遞減，可以看出隨著  $df$  增加，分配圖形幾乎是等距往右移，不會漸漸減少變動幅度。



### 3.2.6 F 分配

F 分配有兩個參數， $d1, d2$ ，兩者都是自由度，其不同參數下的圖形與機率密度函數如下：

$$f(x; d1, d2) = \frac{\Gamma(\frac{d1+d2}{2})}{\Gamma(\frac{d1}{2})\Gamma(\frac{d2}{2})} \left( \frac{d1/d1x}{d1/d1x + d2} \right)^{\frac{d1}{2}} \left( \frac{d2}{d1x + d2} \right)^{\frac{d2}{2}}$$

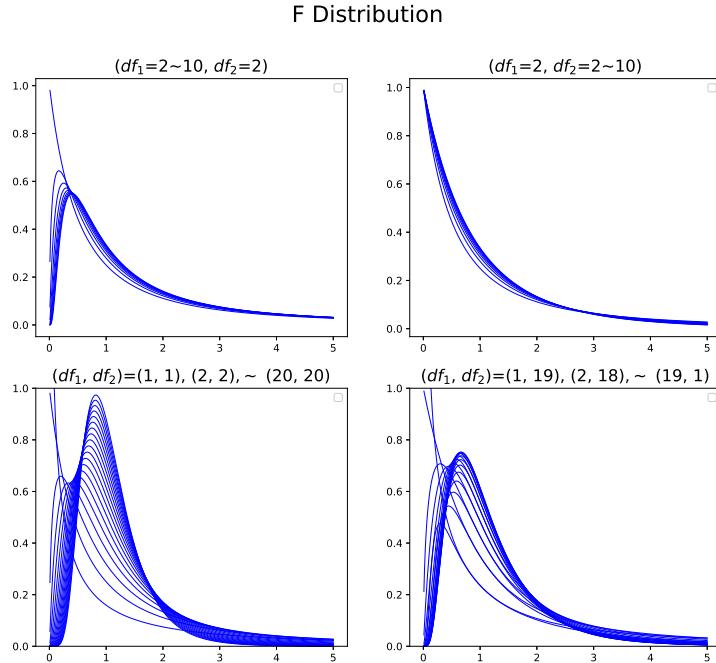


圖 3.14: F 分配不同參數的圖形

從 3.14 左上、右上圖可知，固定  $d2$  改變  $d1$  會使得圖形些微的往右移，變動幅度不大；若是固定  $d1$  改變  $d2$  圖形會由平緩漸漸變陡，值得注意的是，當  $d1 < d2$  時，分配形狀會與  $d1 > d2$  不一樣，是屬於單調遞減函數。再來，從圖 3.14 左下、右下圖可知，同時改變兩個參數

也會改變分配的位置，當兩個參數相等時，一起上升會使的圖形右移、變陡。



### 3.2.7 柯西分配

科西分配有兩個參數，與常態分配類似，具有位置參數  $m$  與尺度參數  $\gamma$ ，但不同的是柯西分配屬於厚尾分配，雖然類似鐘形分配，但與常態分配還是有一定差距，以下是其機率密度函數，與更改不同尺度參數的柯西分配圖形：

$$f(x; m, \gamma) = \frac{1}{\pi \gamma \left(1 + \left(\frac{x-m}{\gamma}\right)^2\right)}$$

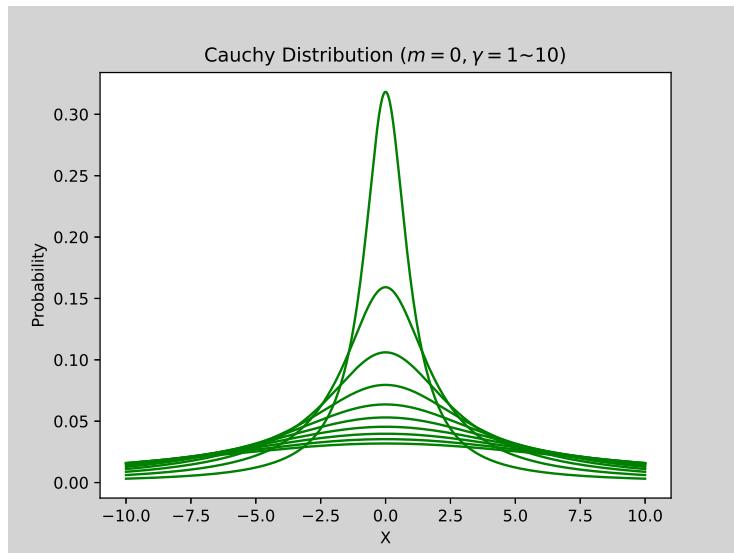


圖 3.15: 柯西分配不同尺度參數的圖形

從圖 3.15 可以看出，增加尺度參數會使得圖形趨向均勻分配，兩尾的面積會逐漸增加，中間的面積會逐漸減少，若尺度參數增加至更大，其分配函數會往一條水平線趨近。

## 3.3 亂數產生

此小節將會利用 python 的亂數產生器，產生三個分配的樣本，分別是：

- 卡方分配 ( $\chi^2$  distribution)
- 貝塔分配 (Beta distribution)
- 迦馬分配 (Gamma distribution)

並將產生的亂數與理論的 pdf、cdf 比較，檢視產生的亂數是否真的服從該分配。



### 3.3.1 卡方分配

選擇  $n = 1000$  產生卡方亂數，分別繪製直方圖、箱型圖、常態分位數圖，最後再將樣本與 Empirical CDF 比較：

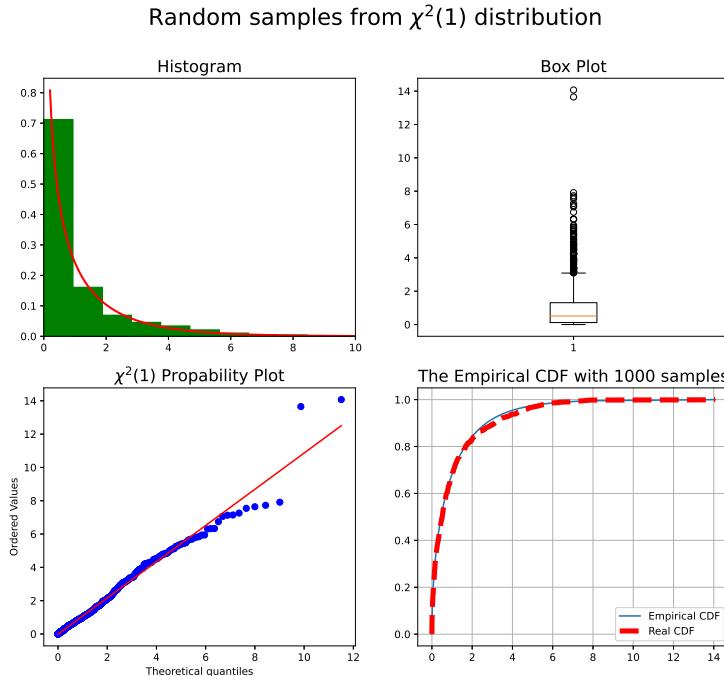


圖 3.16: 卡方 1 的亂數

從圖 3.16 的直方圖、分位數圖 ECDF 可以看出，亂數生成的樣本確實與實際理論的  $\chi^2(1)$  分配一樣，資料點幾乎貼著理論值，幾乎沒有誤差。

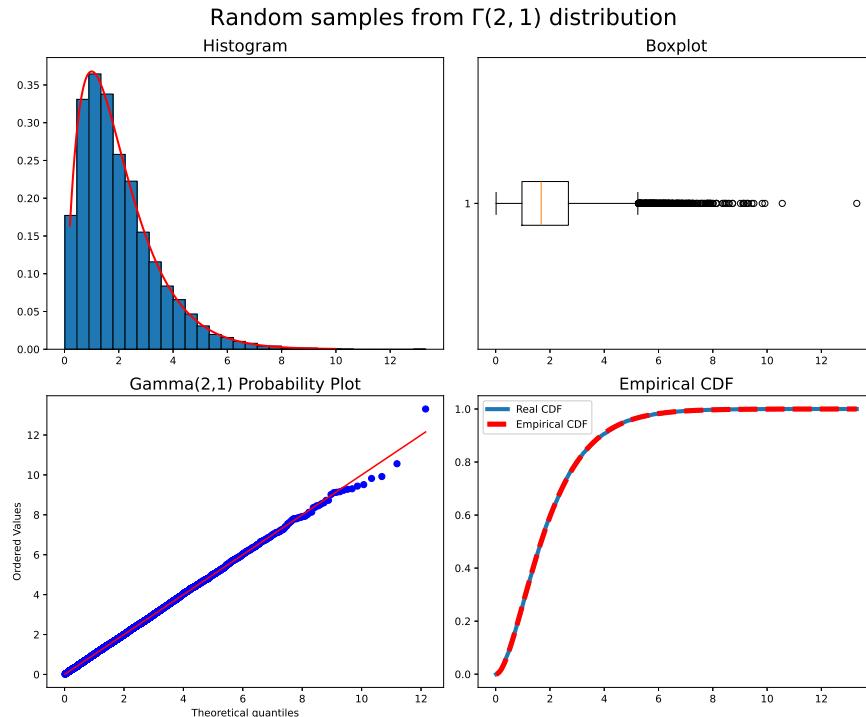
### 3.3.2 迦馬分配

這次選擇  $n = 10000$  產生迦馬分配亂數，分別繪製直方圖、箱型圖、常態分位數圖，最後再將樣本與 Empirical CDF 比較：

從圖 3.16 的直方圖與分位數圖可以看出，亂數生成的樣本確實與實際理論的  $\Gamma(2, 1)$  分配一樣，除了幾個極端值，資料點幾乎貼著理論值，幾乎沒有誤差。

### 3.3.3 貝塔分配

這次選擇  $n = 100$  產生貝塔分配亂數，分別繪製直方圖、箱型圖、常態分位數圖，最後再將樣本與 Empirical CDF 比較：當  $n = 100$  時，我們可以發現不管在直方圖、分位數圖還是 ECDF 圖，都比起選擇  $n = 1000, 10000$  存在著較大的誤差，雖然存在誤差，但亂數還是大致照著理論分配走。

圖 3.17:  $\Gamma(2, 1)$  的亂數

## 3.4 抽樣分配

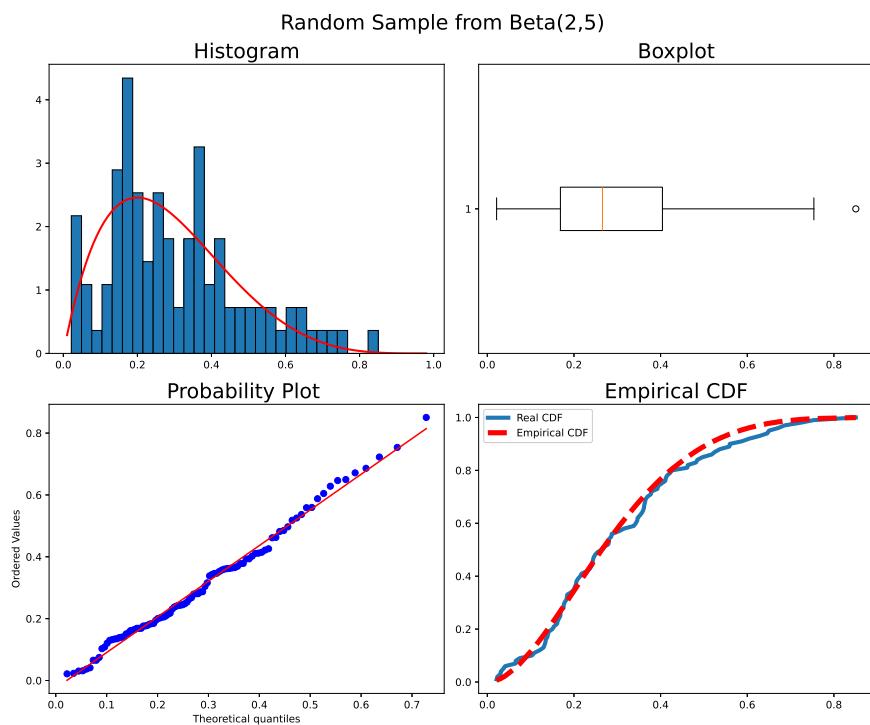
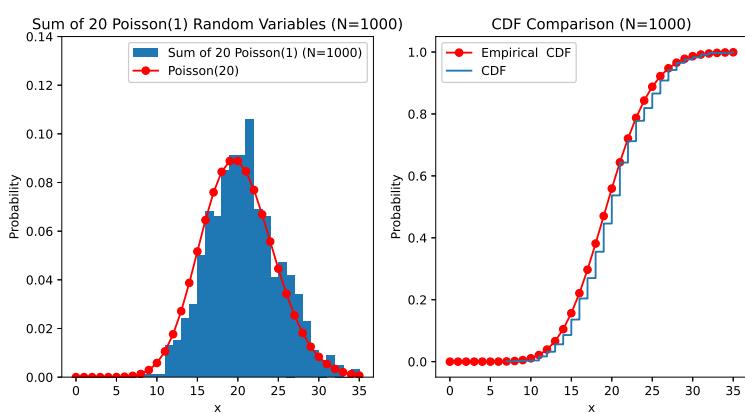
在這小節將會介紹三種抽樣分配，以程式繪圖去驗證我們當初在數理統計上的結果，檢驗理論是否是真的正確，並且每個分配都會分別試驗抽取不同的樣本數，來檢視樣本大小對抽樣分配理論的重要性，以下是三個抽樣分配與結果：

- $x_i \stackrel{\text{i.i.d.}}{\sim} Poisson(1)$  then  $\sum_{i=1}^n x_i \sim Poisson(n)$
- $x_i \stackrel{\text{i.i.d.}}{\sim} exp(\lambda)$  then  $2\lambda \sum_{i=1}^n x_i \sim \chi^2(2n)$
- 中央極限定理

分別為卜瓦松加成性、指數分配加成性與中央極限定理的驗證。

### 3.4.1 卜瓦松分配加成性

在機率概論曾學過， $n$  個相同  $\lambda$  的獨立卜瓦松分配，相加還會是一個卜瓦松分配，而且  $\lambda$  值會變成每個卜瓦松分配的  $\lambda$  相加，也就是  $n\lambda$ 。為了驗證這項結果，我將產生 20 個獨立且相同的  $Poisson(1)$  分配，三次分別抽取 100, 1000, 10000 個樣本，並驗證這 20 個獨立分配相加後會服從  $Poisson(20)$ 。從圖 3.19 可以看出在  $n = 100$  時直方圖與實際上  $poisson(20)$  的分配還是存在著明顯差距，但轉換成 cdf 後，與理論分配的差距並不算大，我們接著繼續看  $n = 1000, 10000$  的情況：

圖 3.18:  $\text{Beta}(2,5)$  的亂數圖 3.20: 20 個樣本數 1000 的  $\text{poisson}(1)$  分佈相加

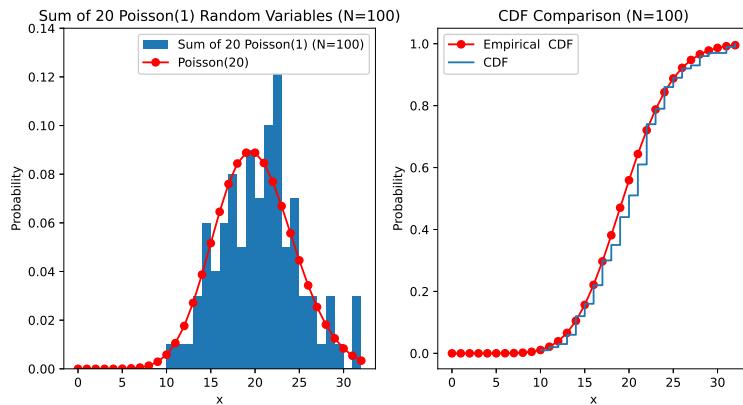


圖 3.19: 20 個樣本數 100 的  $poisson(1)$  分配相加

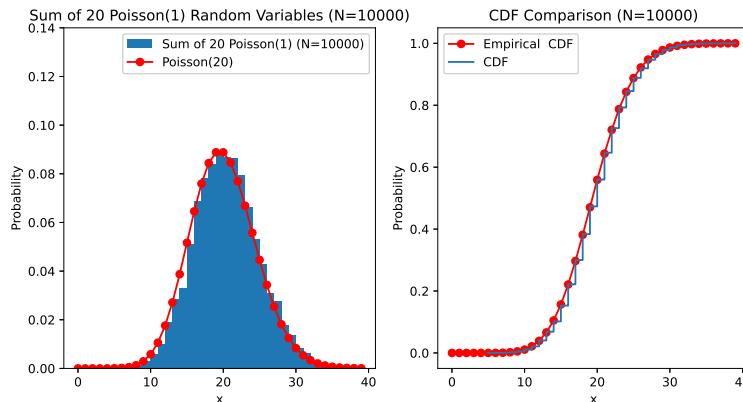


圖 3.21: 20 個樣本數 10000 的  $poisson(1)$  分配相加

從圖 3.20、圖 3.21 可以看到當  $n$  足夠大的時候，抽樣分配不管是在直方圖還是 CDF 都已經和理論值呈現同樣的分布趨勢。

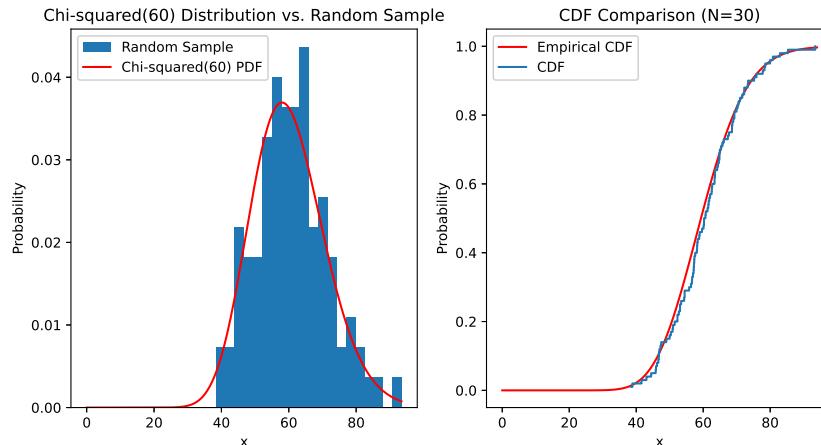
但若將 3 種情況的平均數計算出來，分別為：

$$\mu_{100} = 19.95, \mu_{1000} = 20.12, \mu_{10000} = 19.9$$

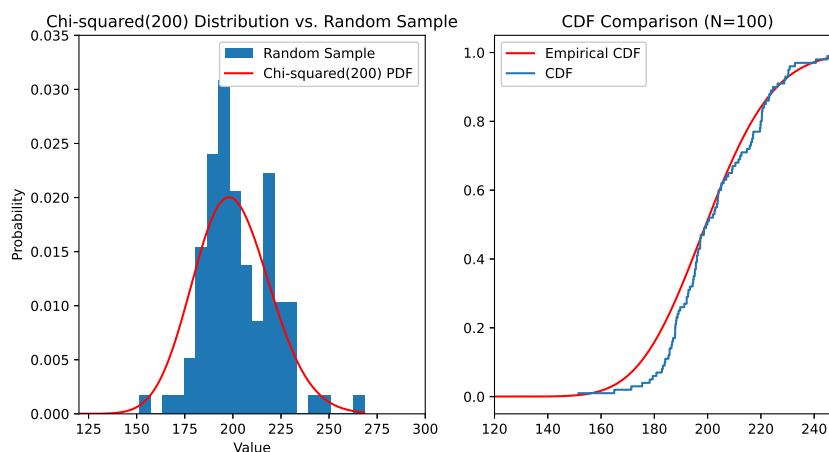
可以發現三者與理論平均值的差距不會隨著  $n$  增加而縮小，可見平均數不需要樣本夠大即可趨近於理論值。

### 3.4.2 指數分配加成性

在數理統計的假設檢定中，我們常把指數分配轉換為卡方分配進行檢定統計量的計算，因為透過卡方分配我們才能做查表，以做出拒絕或不拒絕的結論。在這個例子中，我將分別產生 30,100,1000 個獨立且相同的  $exp(1)$  樣本，每個樣本包含 100 個觀察值，驗證把樣本相加再乘以  $2\lambda$  後，會服從  $\chi^2(2n)$ 。

圖 3.22: 30 個  $\exp(1)$  樣本進行轉換

$N = 30$  時，與理論誤差不算太大，直方圖與 CDF 圖幾乎都依靠著理論分配，故猜測隨著  $N$  再繼續增加的話，效果不會太明顯。

圖 3.23: 100 個  $\exp(1)$  樣本進行轉換

$N = 100$  時，誤差竟然是比  $N = 30$  時更大，本文反覆抽樣了數次，發現誤差仍然都與  $N = 30$  時大一些，為了更了解樣本數會不會影響此抽樣分配的精準度，故將  $N$  增加至 1000：

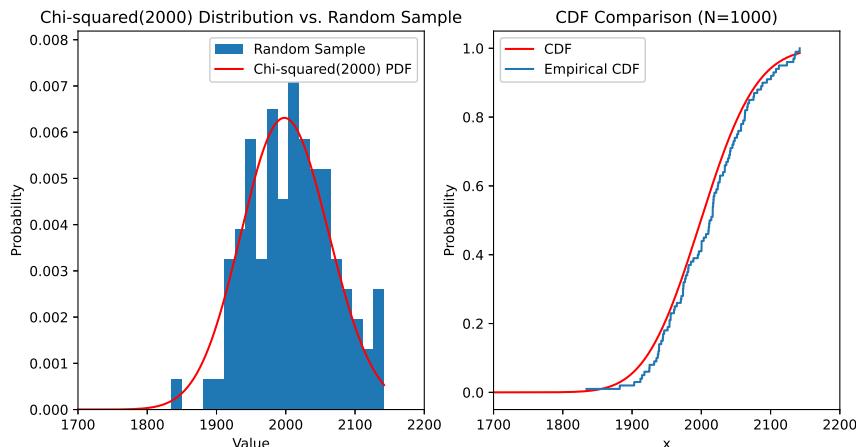


圖 3.24: 1000 個  $\exp(1)$  樣本進行轉換

當  $N = 1000$  時，不像前一小節在  $N = 1000$  時已經相當貼近理論分配，可以從 3.24 的直方圖與 CDF 圖看出雖然趨勢與理論分配相同，但都還是存在不小的誤差，故推論此抽樣分配在樣本數的要求上並沒有那麼嚴格，其不會隨著樣本數增加，而增加精準度。

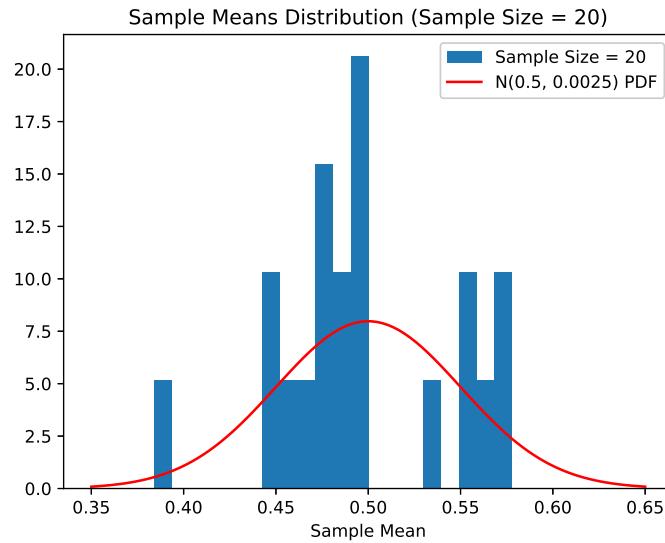
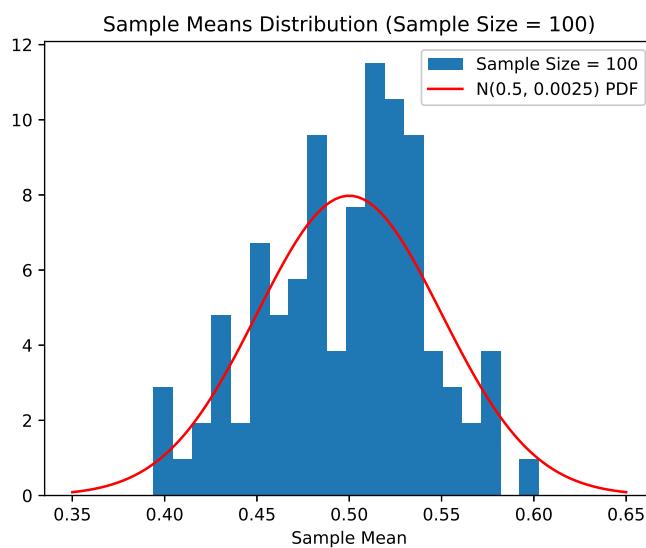
若將 3 種情況的平均數計算出來，分別為：

$$\mu_{30} = 60.84, \mu_{100} = 203.01, \mu_{1000} = 2009.34$$

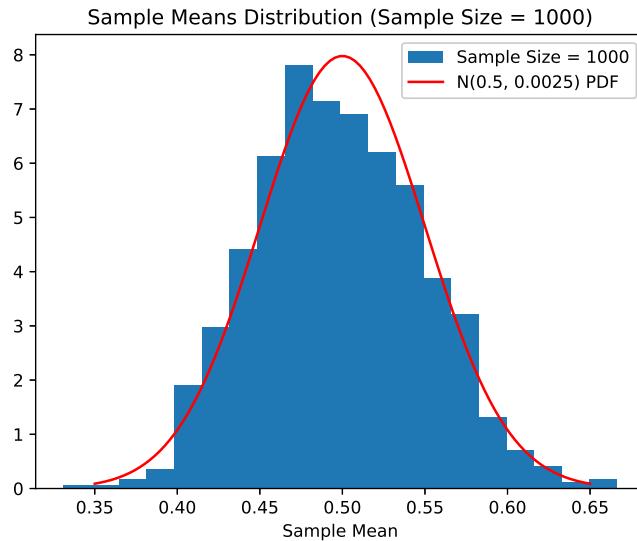
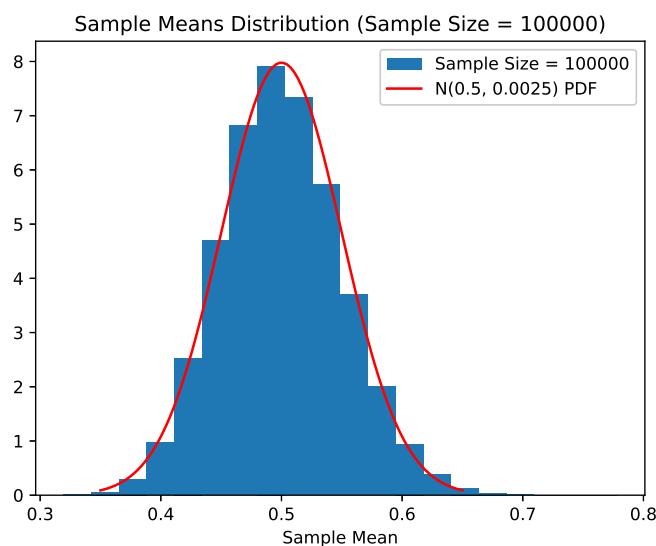
可以發現三者與理論平均值的差距不會隨著  $N$  增加而縮小，可見平均數不需要樣本夠大即可趨近於理論值。

### 3.4.3 中央極限定理

中央極限定理為相當廣為人知的理論，其說明不論從何種分配抽取樣本，只要抽取樣本數足夠大，並抽自同一分配，則這些樣本的樣本平均數會服從常態分配，其中平均數恰為母體平均數，變異數則是母體變異數除以  $n$  (抽取的樣本數)。以下我將產生 20,100,1000,100000 個  $\exp(\lambda = 2)$  樣本，以驗證中央極限定理的結論，確認樣本平均數是否會服從  $N(\frac{1}{2}, \frac{1}{4n})$

圖 3.25:  $n = 20$ 圖 3.26:  $n = 100$ 

由圖 3.25、圖 3.26 可以看出當  $n$  較小時，雖然分配圖形與實際誤差較大，但還是能看出其為鐘形分布，且中心點位在 0.5 的位置。

圖 3.27:  $n = 1000$ 圖 3.28:  $n = 100000$ 

由圖 3.27、圖 3.28 可以看出當  $n$  非常大時，幾乎與常態分配的 pdf 重疊，可知樣本數對中央極限定理影響相當之大，樣本數越大則理論越可靠，透過這幾張圖我們也驗證了中央分配理論是不可爭的事實。

### 3.5 數理統計題目驗證

該小節我們將探討數理統計課本上的一題習題，使用 python 驗證其結果，原題目如下與結果如下：



給定四個數字 (2, 4, 9, 12)。從這四個數字中隨機抽取四個數字（取後放回）並計算其平均數。假設隨機變數  $Y$  代表這四個數字的平均數。請繪製隨機變數  $Y$  的 PMF。

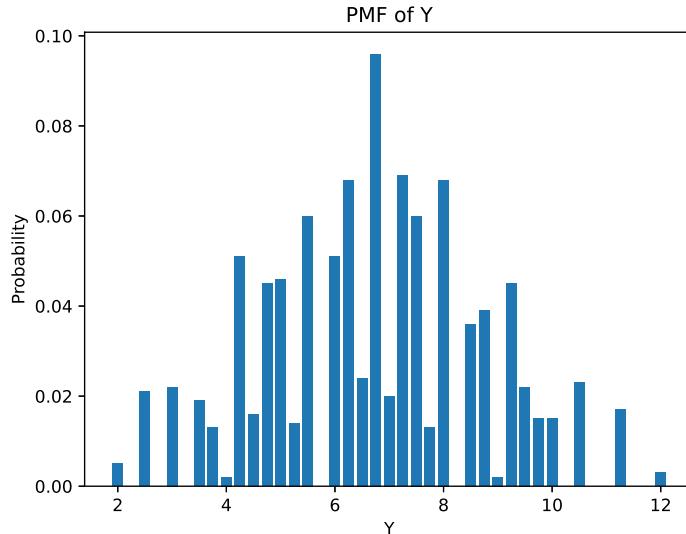


圖 3.29: 四個數字隨機抽樣的平均數

該題用迴圈與 `unique` 的語法就能產生平均數，再使用長條圖將這些平均數繪製上去，以下是該題程式碼：

```
Ys = []
n = 1000
for _ in range(n):
    sample = np.random.choice([2, 4, 9, 12], size=4, replace=True)
    Y = np.mean(sample)
    Ys.append(Y)

unique_Y, counts = np.unique(Ys, return_counts=True)
density = counts / n
plt.bar(unique_Y, density, width=0.2)
```

## 3.6 結論

本文使用 `python` 繪製了許多常見分配，可以了解改變參數對於分配的實際影響，不管是在連續型分配還是離散型分配。也透過亂數產生，繪製了理論值與亂數值之間的差異圖，從中了解亂數產生的語法以及繪製各種圖形的方式。最後我們透過實際作圖驗證了數理統計中的三個定理，經過實作確實能更清楚瞭解定理為何成立，與成立的條件，把這些結果驗證完畢之後，甚至更改了本文作者過去對數理統計的模糊的概念，收穫良多！



## 第 4 章

# 監督式學習之迴歸學習器

機器學習的學習器有非常多種，其中又分為監督式學習與非監督式學習，差別在於是否需要資料訓練。而本文將介紹監督式學習中的三種學習器，三者皆與迴歸有關，分別是**簡單線性迴歸模型**、**增廣性線性迴歸模型**和**邏輯斯迴歸模型**，並用不同資料展示三種模型的分類情況，檢視不同資料在不同模型配適下的成果。

### 4.1 簡單線性迴歸模型

簡單線性迴歸對我們來說並不陌生，在很多領域都會使用到這項統計方法，因為簡單線性迴歸多適用在連續型資料，顯然把它放在分類器並不合適，但我們可以試著了解若將它作為分類器，是否真的會產生較大的誤差。

#### 4.1.1 模型理論

- 學習原理：

簡單線性回歸用於預測因變量 ( $y$ ) 與單個自變量 ( $x$ ) 之間的關係。它通過配適一條最適合資料的直線來建立模型，建立迴歸直線的方法則是透過最小化實際資料點與預測點的差距，來學習特徵和目標變量之間的關係。

- 預測原理：

當模型訓練完成後，利用這條直線對新的自變量數值進行預測。將新的自變量輸入模型，通過直線方程來預測相應的因變量值。



以兩個連續型變數以及一個類別變數為例，迴歸模型將透過兩個特徵進行分類，首先將特徵 ( $x$ ) 輸入之後會產生預測值  $\hat{y}$ ，並根據函數 G 的規則，以輸出值 0.5 為界進行分類：

$$Y = \beta_0 + \beta_1 x_1 + \beta_2 x_2$$

$$G = \begin{cases} Group\ 0, & \hat{y} \leq 0.5 \\ Group\ 1, & \hat{y} > 0.5 \end{cases}$$

#### 4.1.2 資料實作

第一份資料是利用 `sklearn` 內建套件 “`make_blob`” 生成的一個樣本，總樣本數 100，其中包含兩個連續型變數，與一個類別變數 (0、1 兩類)，這份資料兩種類別的離散程度較大，個別比較集中，是非常好進行分類的資料，簡單線性迴歸的分類結果如下：

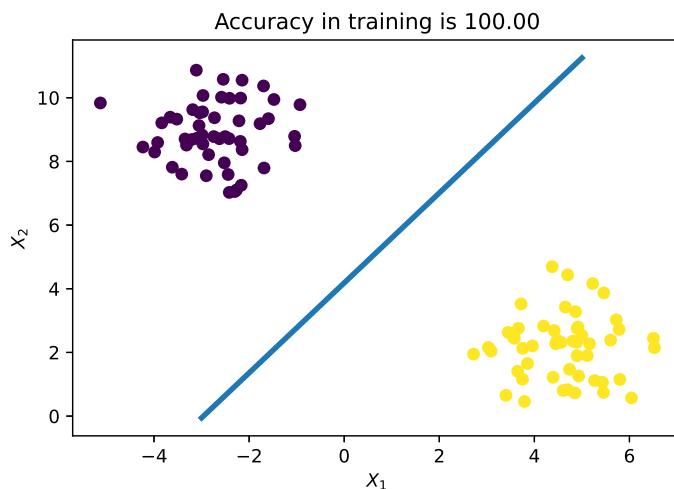


圖 4.1: 透過 `make_blob` 生成的樣本分類

在容易分類的情況下，使用線性迴歸進行分類無傷大雅，顯然線性回歸仍具備非常基礎的分類能力，在這份樣本中達到了 100 的準確度。為此，本文將再測試資料混合程度更高的資料，以檢視其分類情況。

下面資料是來自 `kaggle` 的一份消費者購買行為的分類資料，其特徵包括兩個：觀測者的年齡與薪資，類別則是其購買與否 (0、1)，樣本數為 400，以下為分類狀況：

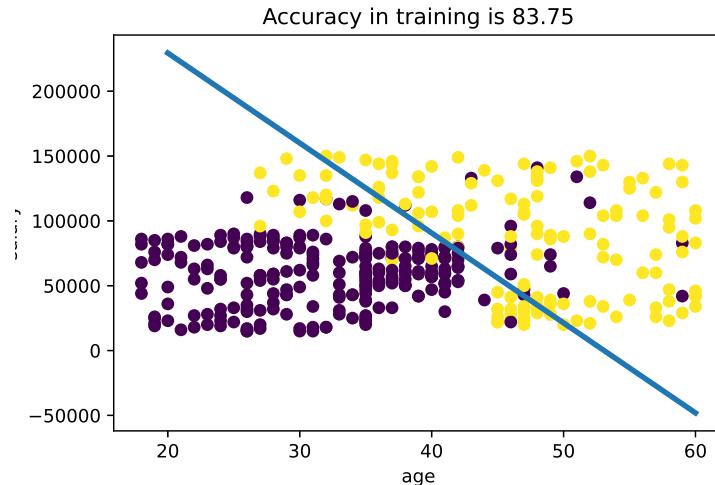


圖 4.2: 消費者購買行為的線性迴歸分類

由圖 4.2 可以看的出來當資料混合程度較高、邊界較複雜，有些資料點就會被分類錯誤，雖然如此，但其正確分類率還是有 83.75，這仍是一個滿高的數字，因此本文將再生成一個複雜程度高的資料，繼續觀察此模型分類的正確率。

下面是透過二維常態隨機亂數生成的一組樣本，其平均數與共變異數矩陣如下，並附上分類後的決策邊界與準確度：

$$\mu_0 = \begin{bmatrix} -2 \\ 3 \end{bmatrix}, \mu_1 = \begin{bmatrix} -1 \\ -2 \end{bmatrix}, \Sigma_0 = \begin{bmatrix} 15 & 1.2 \\ 1.2 & 20 \end{bmatrix}, \Sigma_1 = \begin{bmatrix} 5 & 2 \\ 2 & 18 \end{bmatrix}$$

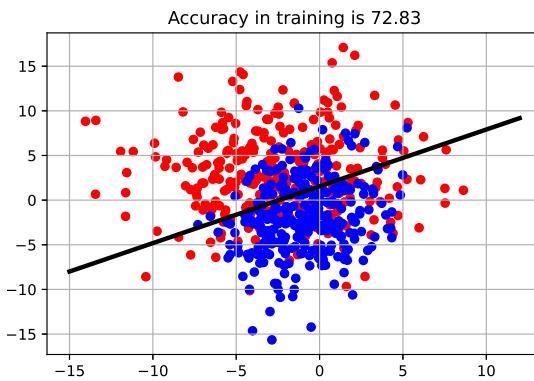


圖 4.3: 二維常態亂數樣本的線性迴歸分類

在這份較為複雜的資料中，雖然準確度還是有 72.83，但從圖 4.3 可以看見，以一條直線作為分群的界線已經不足夠，圖中可以也發現，如果要將這兩個組別完全盡量的分開，那麼兩組的決策邊界應該會是一條彎曲的線，使用直線作為邊界是一個比較粗糙的分類方法，我們也終於在這份資料看出了簡單線性迴歸的弊端。



## 4.2 加廣型線性回歸模型

加廣型線性回歸，顧名思義就是簡單線性迴歸的加深版本，在上一小節中，透過驗證發現一條直線並不能很好的解決分類的任務，因此為了改善這個問題，我們需要一條更貼合邊界的曲線，而加廣型線性回歸的引進，就稍微改善了這樣的問題。

### 4.2.1 模型理論

- 學習原理：

增強回歸模型是對簡單線性回歸模型的擴展，它更考慮了自變量之間的關係，以及自變數的次方關係。與簡單線性迴歸相同，也用於預測因變量與多個自變量之間的關係。另外，這種模型可以處理多維度的資料，並通過擬合最適合資料的多維平面或超平面來建立模型。

- 預測原理：

當模型訓練完成後，利用這條迴歸線對新的自變量數值進行預測。將新的自變量輸入模型，通過方程式來預測相應的因變量值。

### 4.2.2 資料實作

同樣以兩個特徵、兩種組別的資料為例，加廣型線性迴歸的模型如下，與簡單線性迴歸相比，增加了二次方項與交互作用項，分類方法同樣是將特徵 ( $x$ ) 輸入之後產生預測值  $\hat{y}$ ，並以輸出值 0.5 為界進行分類：

$$Y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_1^2 + \beta_4 x_2^2 + \beta_5 x_1 x_2$$

$$G = \begin{cases} \text{Group 0, } \hat{y} \leq 0.5 \\ \text{Group 1, } \hat{y} > 0.5 \end{cases}$$

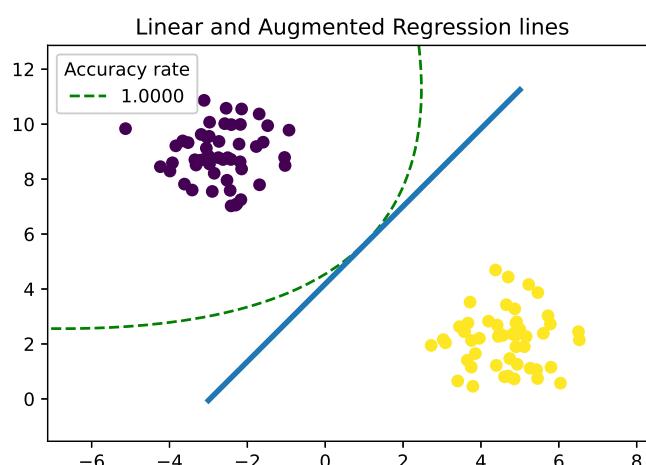


圖 4.4: 透過 make\_blob 生成樣本的線性迴歸分類



從圖 4.4 中可以看得出來，增廣型線性迴歸的決策邊界是一條曲線，雖然與簡單線性迴歸的決策邊界不同，但分類結果是相同的，正確率也是 100，這樣的結果應該是理所當然，因為更複雜的模型將會對結果有更高的適配性，所以如果較基礎的簡單線型迴歸都能做到完全分類，那更高階的增廣型線性迴歸也能做到。再來同樣使用第二份資料，也就是消費者行為分類的資料，其分類結果如下：

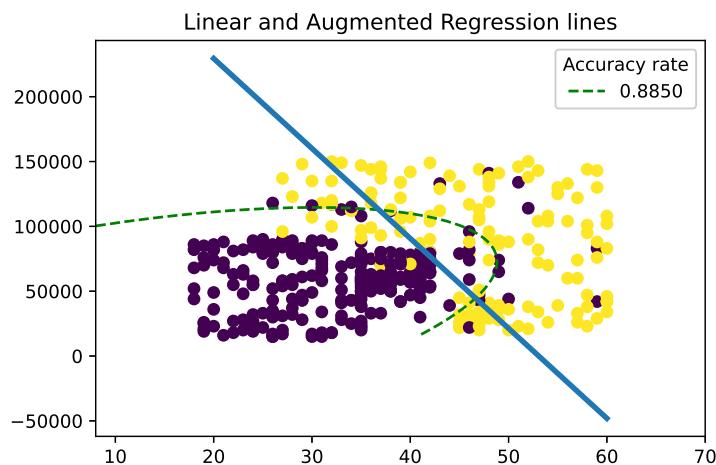


圖 4.5: 消費者購買行為的線性迴歸分類

在圖 4.5 中，簡單線性迴歸與增廣型線性迴歸的優劣一眼可見。這份資料的邊界類似一個直角，要正確的分類這份資料勢必需要一個非線性模型，因此增廣型線性迴歸是一個很好的選擇，它的邊界呈現如一個鉤子狀，相當近似這份資料的直角邊界，把組別 1、0 清楚地分別開來，正確率也達到了 88.5，相比先前的 83.75 有很大的提升。

最後看到第三份資料，也就是兩類別混合程度較高的資料，其分類結果如下：

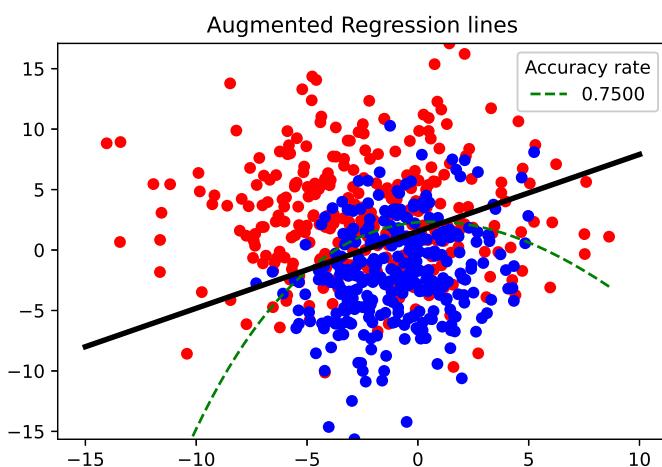


圖 4.6: 二維常態亂數樣本的線性迴歸分類



在這份資料中，增廣型線性迴歸的表現並不向上個資料那麼突出，其正確率僅僅增加 1.5，雖然決策邊界有更貼近，但由於資料混合程度較高，所以儘管為某部分資料調整邊界，仍會造成另一部分錯誤分類，但不可否認的，這樣的曲線邊界仍是比單一條直線來得好。

### 4.3 邏輯斯回歸模型

以上兩種迴歸事實上多用於預測“連續資料”，在預測類別資料方面，有另一個比較適當的模型，也就是邏輯斯迴歸，其特點在於它的輸出本來就會界於 0 ~ 1 之間，因此將其作為分類器相當合理。

- 學習原理：

邏輯斯迴歸是一種分類模型，用於預測多元結果（兩個類別以上）的機率。它藉由將線性方程的結果通過一個邏輯函數 (sigmoid 函數，式 4.1) 轉換成概率值。模型學習適應訓練資料的權重，使得預測結果能夠最好地符合實際觀察到的分類。

$$y = \frac{e^{\alpha x}}{1 + e^{\alpha x}} \quad (4.1)$$

- 預測原理：

在新的資料上，模型會輸出一個介於 0 和 1 之間的概率值，通過設定閾值，可以將這個概率轉換成二元的分類結果（例如 0 或 1）。

同樣使用前面小節的三份資料來分類，首先第一份資料的分類情況如下：

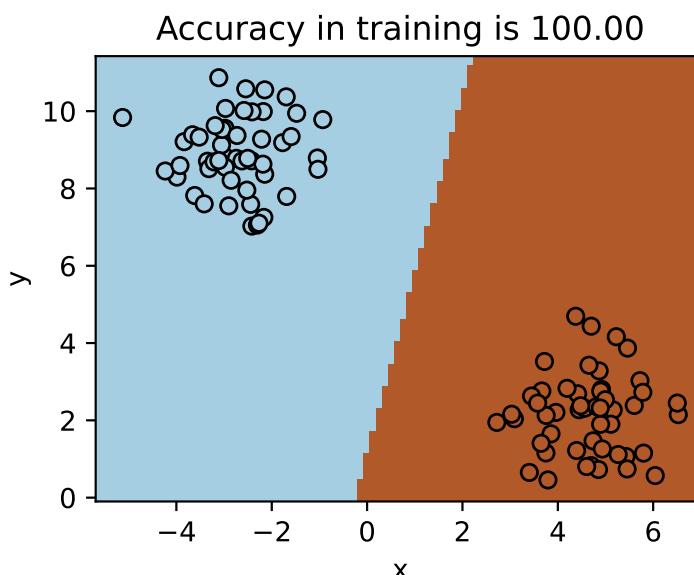


圖 4.7: 透過 make\_blob 生成樣本的邏輯斯迴歸分類

分類結果依然是完全正確，但值得注意的是，邏輯斯迴歸產生的決策邊界比簡單線性迴歸產生的更加接近垂直，雖然都是直線，但邏輯斯



迴歸生成的邊界，離兩邊資料點的距離是較近一點的，也就是其邊界相對來說比較沒有那麼貼合資料。接下來我們使用第二份資料：

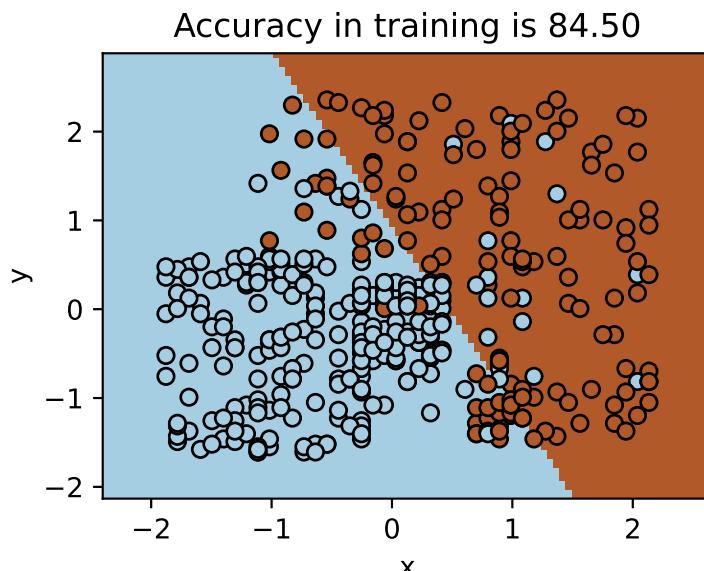


圖 4.8: 消費者購買行為的邏輯斯迴歸分類

從圖 4.8 可以看出邏輯斯分類的正確率高於線性迴歸 (83.5) 一些，從邊界圖可知其邊界也會是一條直線，與線性回歸不同的是其邊界較接近垂直，因此在此資料中使用邏輯斯迴歸與簡單線性迴歸的差異是不大的。最後是第三份資料的分類情況：

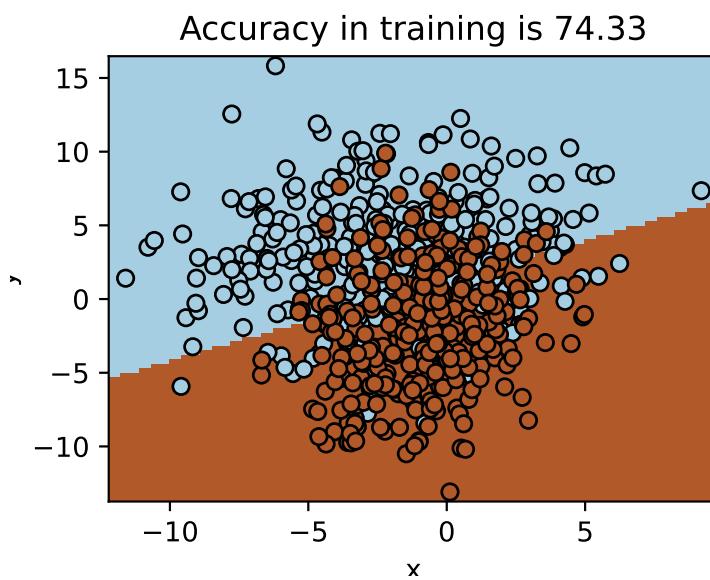


圖 4.9: 二維常態亂數樣本的邏輯斯迴歸分類

以上所討論的都是進行兩類分群的情況，當要分類三群或以上的時候，繼續使用線性迴歸將會變得複雜，例如分成三類的話，就得再額外去決定兩個閥值，隨著需要分類的類別增加，要設定的閥值也會越來越



多，然而，羅吉斯回歸在操作這種多群分類就會較簡易，只需要將資料放進模型訓練，產生出的預測值就會是“群”，下面是使用邏輯斯迴歸進行三種類資料的分類結果，一樣是使用兩個特徵，在不需要設定閥值下，其能輕易做出決策邊界，並且準確度來到 82。

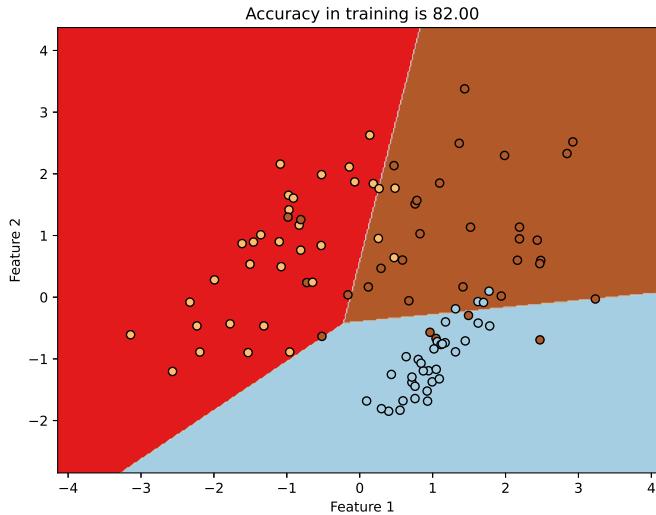


圖 4.10: 三個群組樣本的邏輯斯迴歸分類

## 4.4 結論

本文將三份不同分散程度的資料，分別放進三個迴歸學習器進行分類，可以知道在資料較為簡單的情況下，即使用簡單線性迴歸模型做分群，仍然可以正確分類大部份資料，但僅限在資料不複雜的情況；而遇到複雜情況時，增廣型線性模型提出了相對好的決策邊界，雖然如此，但在只有兩個特徵的情況下，增廣型線性模型就需要高達 6 個參數，很難保證其不會產生過度配適的問題，因為在過度配適的情況下，準確度很高是必然的。因此在分類問題方面，邏輯斯回歸還是最佳解，畢竟這就是一個為分類存在的模型，雖然在本文的例子中，邏輯斯迴歸的準確度都不算特別優秀(因為資料較簡單)，但隨著資料複雜度增加，使用邏輯斯迴歸在解釋或是應用上都會更方便。



## 第 5 章

# 監督式學習之 LDA、QDA 與 KNN 學習器

本文將介紹三種監督式學習器，三者的功用一樣都是將樣本進行分群，並檢視分類正確率，與迴歸不同的地方是，LDA、QDA、KNN 的決策邊界是利用後驗分配計算出來的，也就是在知道結果的情況下再對樣本進行分群預測，但 LDA、QDA 兩者與 KNN 計算後驗分配的方式有很大的差異，為此，本文將會利用五組資料檢驗三個學習器的優劣比較。

## 5.1 線性判別分析 ( LDA )

本小節將會介紹線性判別分析 ( LDA ) 的理論依據，並透過資料實作，展現在何種資料下，LDA 會有較好的表現；又在什麼情況下，LDA 給出的結果比較差，以此讓讀者了解該將何種資料使用 LDA 進行分類。

### 5.1.1 理論背景

線性判別分析，顧名思義會利用線性的決策邊界進行分群，而讓決策邊界形成線性的原因，主要來自此模型對資料的兩個假設：

- 每個類別的數據都來自於同一個常態分配，但具有不同的均值。
- 每個常態分配的共變異數矩陣相同。

而決策邊界計算的方法是使用後驗分配計算，透過給定  $x$  計算出  $x$  屬於某組的機率，將  $x$  分類進後驗分配較高的組別。

$$Pr(G = k | X = x)$$

因為在計算後驗分配上比較困難，所以常會用貝氏方法，利用先驗分配計算出後驗分配：

$$P(G = k | X) = \frac{P(X | G = k)P(G = k)}{\sum_l P(X | G = l)P(G = l)}$$



在計算出後驗分配後，兩組後驗分配機率相同的地方就會是決策邊界，再透過對數轉換可以得到分界線函數，其數學式如下：

$$\begin{aligned}\ln \frac{Pr(G = k | X = x)}{Pr(G = l | X = x)} &= \ln \frac{f_k(x)}{f_l(x)} + \ln \frac{Pr(G = k)}{Pr(G = l)} \\ &= \ln \frac{Pr(G = k)}{Pr(G = l)} - \frac{1}{2}(\mu_k + \mu_l)^T \Sigma^{-1}(\mu_k - \mu_l) \\ &\quad + x^T \Sigma^{-1}(\mu_k - \mu_l) = 0\end{aligned}$$

另外，由於 LDA 假設共變異數矩陣相同，所以在參數估計方面，僅需要估計一個共變異數矩陣，相對下一小節的 QDA，其模型簡化許多。本文將會利用四筆二元資料與一筆三元資料，來展示 LDA 的決策邊界與誤判率。

### 5.1.2 資料實作

第一筆資料是由套件 `make_blob` 所生成的二元分類資料，資料包含兩個特徵與該筆觀察值類別，其中兩群資料離散的程度較大，屬於很好分類的資料，以下是 LDA 模型產生的決策邊界，其錯誤分類率不出意外的是 0%：

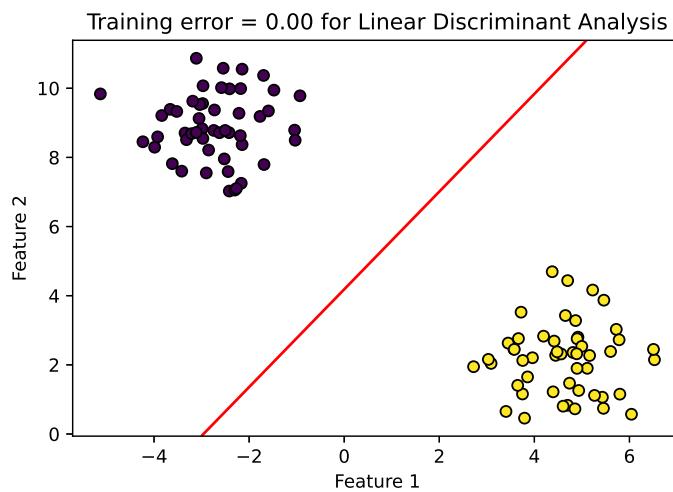


圖 5.1: 資料離散程度大的分類結果

第二筆資料由兩組多變量常態分配亂數產生，其中為了展示在假設成立下，LDA 的分類狀況如何，因此把兩組多變量常態分配的共變異數矩陣設成相等，均值向量設置不同，而樣本數共 400，以下是多變量常態參數的設置與分類結果：

$$\mu_0 = \begin{bmatrix} 7 \\ 3 \end{bmatrix}, \mu_1 = \begin{bmatrix} -1 \\ -2 \end{bmatrix}, \Sigma_0 = \Sigma_1 = \begin{bmatrix} 15 & 1.2 \\ 1.2 & 20 \end{bmatrix}$$

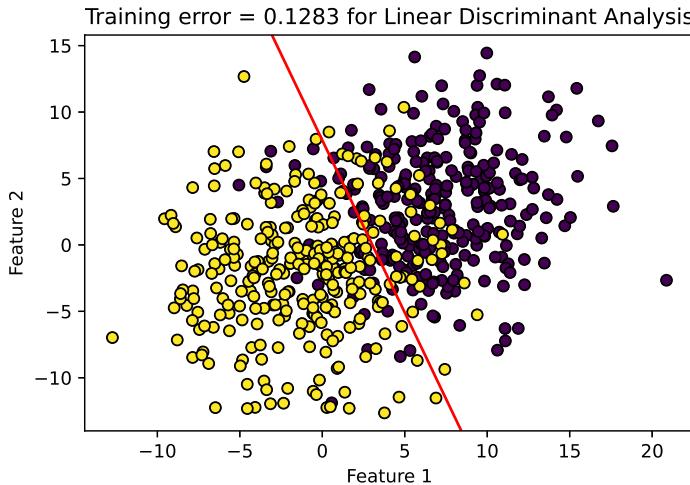


圖 5.2: 共變異數矩陣相同的資料分類結果

從圖 5.2 可以看出在共變異數矩陣相同下，兩個組別離散程度相同，因此用一條直線便可以將資料很好的分類，錯誤分類率是 12.68%，因此可以知道在假設成立的情況下，LDA 的分類表現是相當出色的。

第三筆資料同樣由兩組多變量常態分配亂數產生，與第一組資料不同的在共變異數矩陣，為了檢視在假設不成立下 LDA 的分類情況，因此特意把共變異數矩陣設置不同，但均值保持與前一筆資料相同，用以檢視假設成立與不成立時的差距，而樣本數共 400 以下是參數設置與分類結果：

$$\mu_0 = \begin{bmatrix} 7 \\ 3 \end{bmatrix}, \mu_1 = \begin{bmatrix} -1 \\ -2 \end{bmatrix}, \Sigma_0 = \begin{bmatrix} 15 & 1.2 \\ 1.2 & 20 \end{bmatrix}, \Sigma_1 = \begin{bmatrix} 5 & 2 \\ 2 & 60 \end{bmatrix}$$

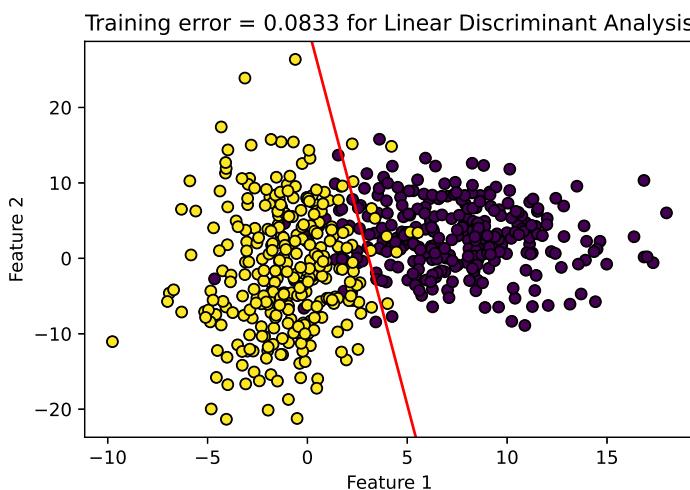


圖 5.3: 共變異數矩陣不相同的資料分類結果



從圖 5.3 可知在這筆資料的誤判率是 8.33%，特意生成一個共變異數矩陣不同的資料，為的是與下一小節的 QDA 做比較，因此關於這筆資料的結論會在下一小節說明。

最後一筆資料是來自 kaggle 的一份消費者購買行為資料，其特徵包括兩個：觀測者的年齡與薪資，類別則是其購買與否 (0、1)，樣本數為 400，因為前面都是自行生成的樣本，具教育性但不具實用性，因此特意找來一個比較實用的資料，以下是 LDA 的分類結果：

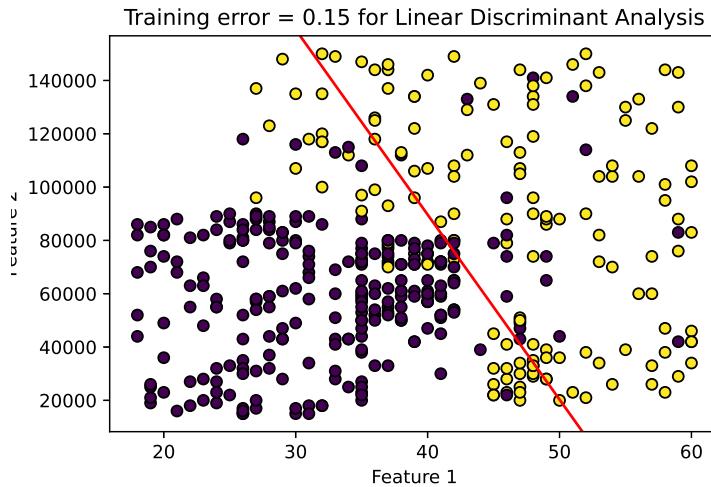


圖 5.4: 消費者購買行為資料分類結果

從圖 5.6 可以看出雖然錯誤分類率並不高，但這筆資料的兩個分類的界線屬於不規則型，因此用一條直線將兩者分類可能有點粗糙，LDA 或許不會是這筆資料最佳的分類器。

第五筆資料是一筆三群分類的資料，同樣來自多變量常態亂數，樣本數為 900，具有相同的共變異數矩陣，但均值向量不同，以下是參數設置與分類情況：

$$\mu_0 = \begin{bmatrix} -5 \\ 3 \end{bmatrix}, \mu_1 = \begin{bmatrix} -1 \\ -2 \end{bmatrix}, \mu_2 = \begin{bmatrix} 7 \\ 8 \end{bmatrix}, \Sigma_i = \begin{bmatrix} 5 & 2 \\ 2 & 18 \end{bmatrix}, i = 0 \sim 2$$

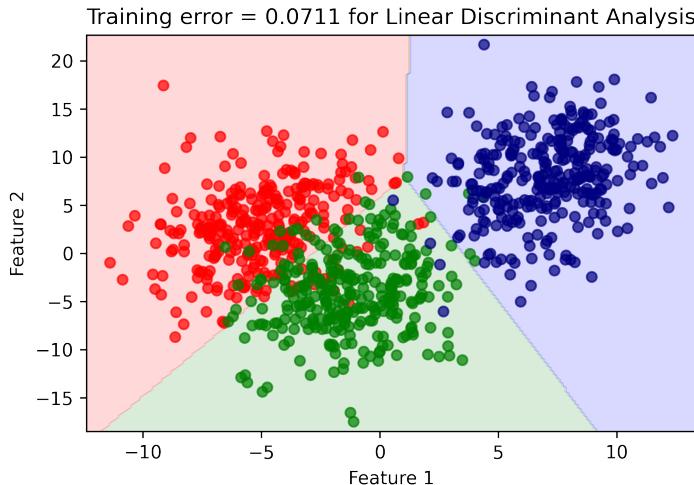


圖 5.5: 三種且共變異數矩陣相同資料分類結果

從圖 5.6 可以看出分類錯誤率為 0.711，並不算太高，但為了與假設不成立時的情況對比，所以下一筆資料會把共變異數矩陣設置成不相同，透過兩筆資料來檢視假設成立與否對三分類問題的影響，以下是第六筆資料的參數設置與分類結果：

$$\mu_0 = \begin{bmatrix} -5 \\ 3 \end{bmatrix}, \mu_1 = \begin{bmatrix} -1 \\ -2 \end{bmatrix}, \mu_2 = \begin{bmatrix} 7 \\ 8 \end{bmatrix}$$

$$\Sigma_0 = \begin{bmatrix} 15 & 1.2 \\ 1.2 & 20 \end{bmatrix}, \Sigma_1 = \begin{bmatrix} 5 & 2 \\ 2 & 18 \end{bmatrix}, \Sigma_2 = \begin{bmatrix} 10 & 0 \\ 0 & 5 \end{bmatrix}$$

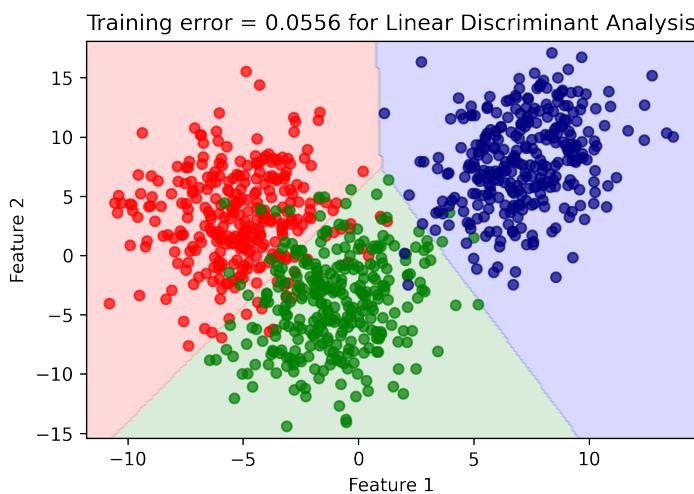


圖 5.6: 三種且共變異數矩陣不同資料分類結果

在共變異數不同的情況下，LDA 並沒有表現的更差，或許是亂數的資料邊界類似線性，使得即時假設不成立仍然表現的比假設成立時好，因此與第三筆資料相同，下一小節將會與接下來介紹的 QDA 比較，並做出結論。



## 5.2 二次判別分析 (QDA)

在介紹完 LDA 後，可以發現 LDA 存在著很強力的假設，並且決策分界較簡易，但當資料變得複雜時，LDA 必定無法很好的分類資料，因此本小節將介紹 LDA 的增廣版：二次判別分析 (QDA)。

### 5.2.1 理論背景

前一小節提到 LDA 有一個很強的假設——共變異數矩陣相同，但在二次判別分析中拿掉了這項假設，條件放寬了許多，但需要估計的參數將會增加很多，以下是 QDA 與 LDA 假設不同的地方：

- 每個類別的數據可來自不同多變量常態分配，具有不同的均值。
- 允許各個多變量常態分配的共變異數矩陣不同。

### 5.2.2 資料實作

在前一小節有說道，造成線性決策邊界的原因是共變異數矩陣相同的假設，因此當拿掉這項假設時，決策邊界會由線性變成非線性，在二維時會類似一個二次曲線，二次判別分析的名字也由此而來。本小節同樣會使用同樣的六筆資料，來展現 LDA 與 QDA 的比較。

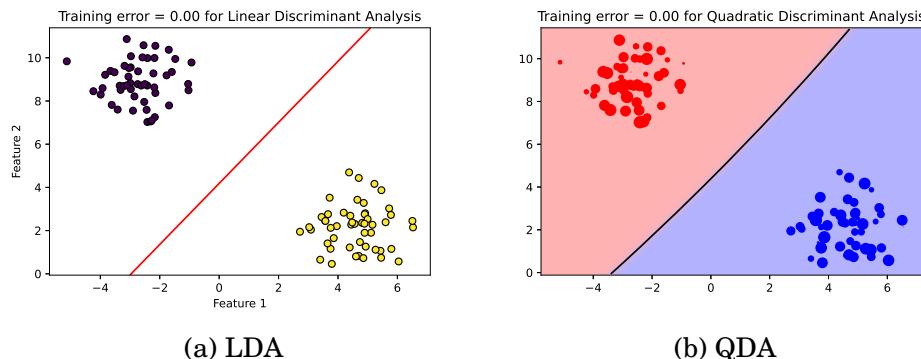


圖 5.7: LDA 與 QDA 比較

從圖 5.7 中可知，在資料比較簡單時兩者的決策邊界幾乎看不出差異，因此如果在需要進行分類的資料樣本較小、或是類別離散程度很大的時候，可以使用較簡單的 LDA 模型即可，可以在估計較少參數下達成目的。

圖 5.8 使用的是共變異數矩陣相同的亂數常態資料，可以看出兩者的決策邊界幾乎是一樣的，因為當共變異數假設成立時，二次判別函數會幾乎退化成線性判別函數，導致兩者的界線大致相同，因此可知在共變異數相同時，使用 LDA 即可，兩者的誤判率並不會有太大的差異。

在圖 5.9 中顯然 QDA 誤判率勝過 LDA，可以看出 QDA 與 LDA 的決策邊界相差甚遠，QDA 的邊界更貼合資料，LDA 僅藉由一條直線分界，

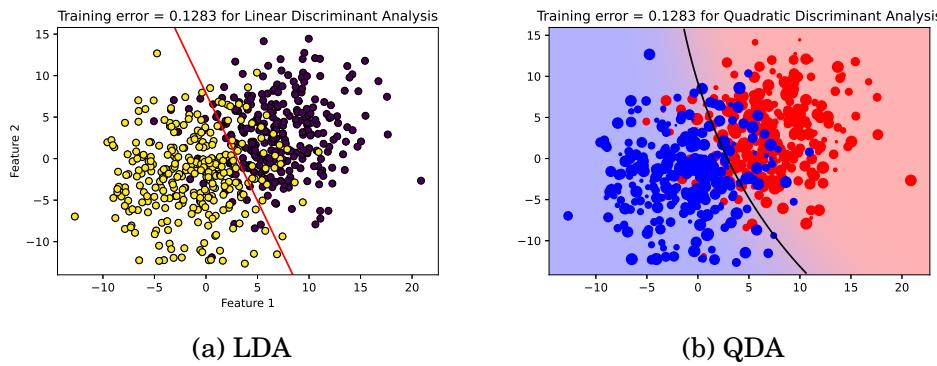


圖 5.8: LDA 與 QDA 比較 (共變異數矩陣相同)

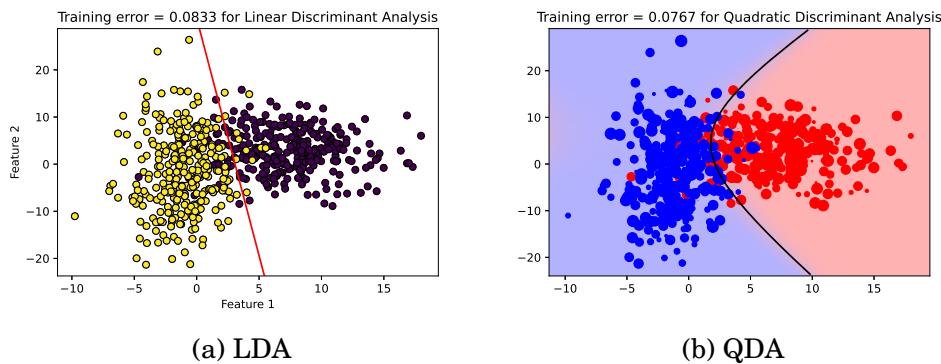


圖 5.9: LDA 與 QDA 比較 (共變異數矩陣不相同)

當資料量增加、複雜度增加時，將無法很好的進行預測，也間接證明了當共變異數矩陣不相同時，QDA 的表現會比 LDA 來的更好。

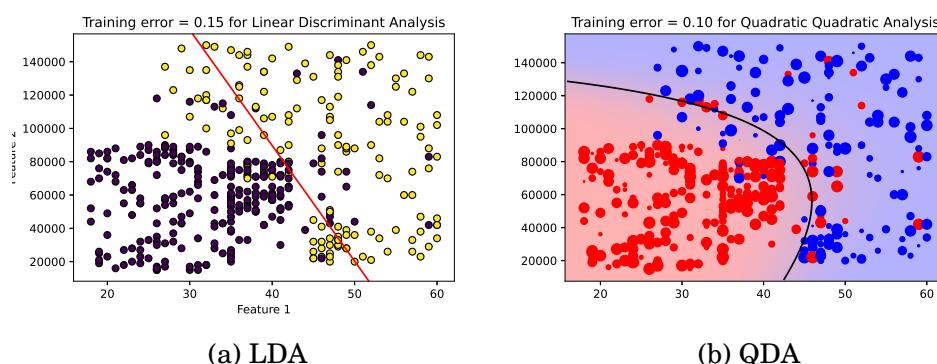


圖 5.10: LDA 與 QDA 比較 (消費者資料)

圖 5.10 是使用消費者資料的分類結果，在前一小節可以發現 LDA 並不能很好的分類這筆資料，因為資料分界並非線性，但透過 QDA 分類很好的貼合了資料，誤判率比 LDA 整整少了 0.05，是在所有資料中差異最顯著的，也證實了在複雜資料、分界線非線性時，QDA 相對 LDA



有較好的表現。

二分類的資料結束後，接著將同時展示兩筆三分類的資料，其中一筆是共變異數矩陣相同的常態亂數，另一筆則是不同的共變異數矩陣，分類結果如下：

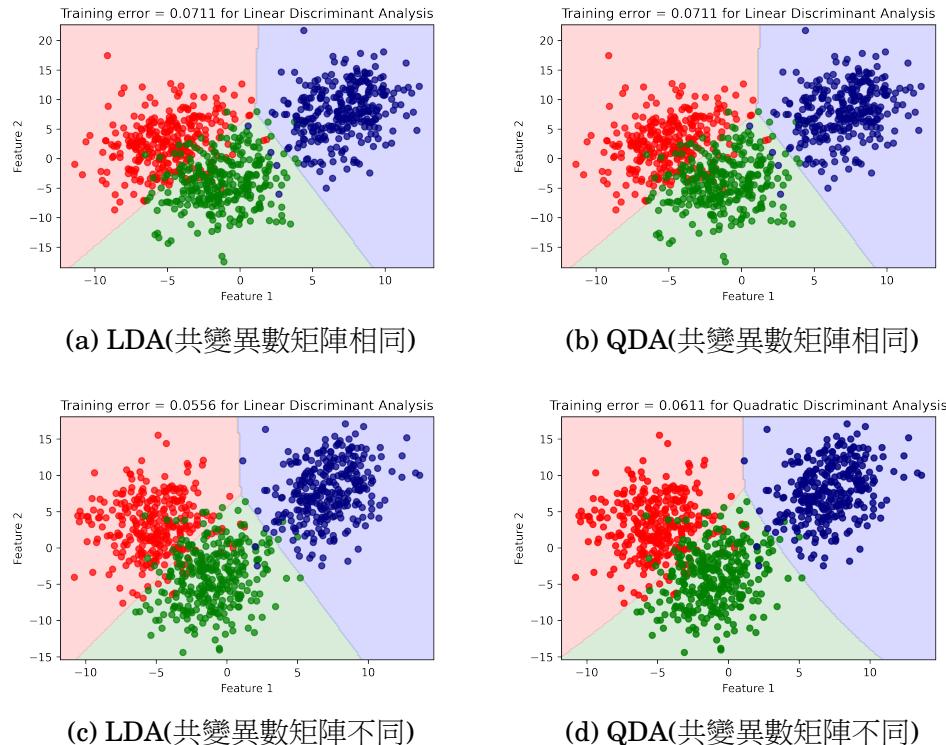


圖 5.11: 三分類問題

在三分類問題下，四者的差異並不大，推測是資料的邊界較趨近於線性，不太能展現 QDA 的優勢，常態亂數生成的樣本較難產生非線性的邊界，因此改進效果有限，但可以確定的是在共變異數矩陣不同時，QDA 仍是比 LDA 降低了 0.01 的誤判率。另外，可以看到 QDA 並不會出現比 LDA 表現得差的情況，可見 QDA 不管在資料複雜或簡單時都能很好的進行分類，但缺點是需要損失較多的自由度進行參數估計。

### 5.3 K-近鄰演算法 (KNN)

KNN 是一個淺顯易懂的分類器，透過設定適當的 K 值即可完成分類且容易解釋，與 LDA、QDA 相同的是，三者都是使用後驗分配計算出決策邊界，但 KNN 計算後驗分配的方式則與前兩者截然不同，因此本小節將會介紹 KNN 的原理，並使用同樣的六筆資料進行實作，展示其與前兩者的差異性。



### 5.3.1 理論

KNN 並沒有對資料進行任何假設，其運作原理是透過捕捉觀察值周圍  $K$  個資料點的組別數據來計算後驗分配，舉例而言，若某觀察值周圍屬於組別 A 的資料點佔比最高，則會將其歸類為組別 A，而參數  $K$  就是用來決定要捕捉周圍幾個資料點，此種方法不存在繁雜的觀念，透過直觀的方法將樣本進行分類，以下是 KNN 的優點與缺點：

KNN 的優點：

- 無需訓練：KNN 是一種基於實例的學習方法，它不需要對數據進行太多的訓練過程，只需儲存訓練數據，因此在訓練階段可以更快。
- 適用於多類別問題：KNN 可以輕鬆處理多類別分類問題。
- 適用於非線性數據：KNN 不對數據做出任何假設，可以處理非線性關係。

KNN 的缺點：

- 在增加新資料時，需要計算新數據點與所有訓練數據點之間的距離，這可能導致計算複雜度較高。
- 比較高維度的數據集或者具有大量特徵的數據會導致"維度災難"，影響演算法的性能。
- 數據不平衡問題：當類別不平衡時，KNN 往往會偏向於具有更多樣本的類別。

### 5.3.2 資料實作

在優點與缺點相當的情況下，該在什麼樣的資料下使用 KNN 更是一件重要的事，因此接下來將透過資料展示 KNN 的分群情況，並使用兩個不同的  $K$  值呈現參數差異 ( $K = 5, K = 15$ )。

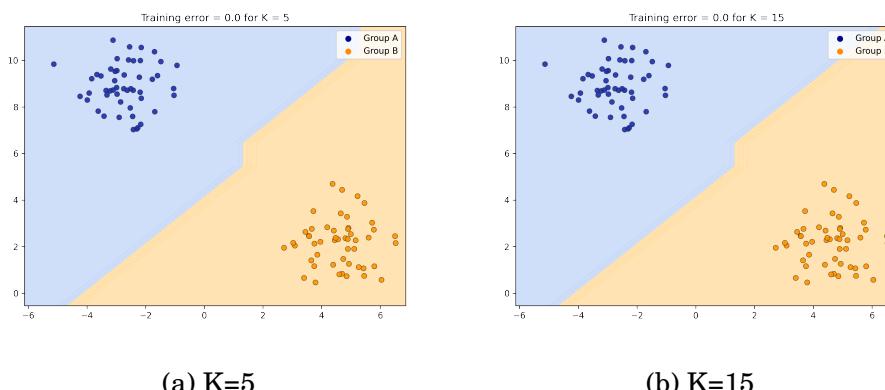


圖 5.12: 簡單資料分類

第一筆資料與前面的結果相同，都是完全準確的分類，唯一 KNN 與前兩者不同的是，其決策邊界並非線性或是二次曲線，而是不規則的形



狀，可以從中間的鋸齒狀的看出來，這即是 KNN 與 LDA、QDA 在邊界上最大的不同。

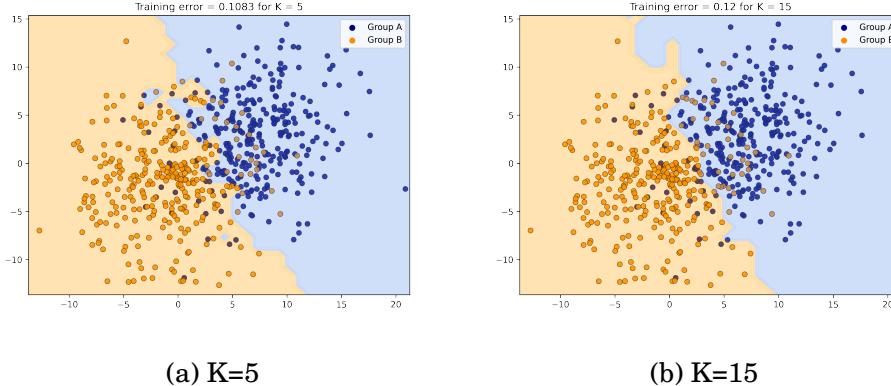


圖 5.13: 共變異數矩陣相同分類

第二筆資料是共變異數矩陣相同的常態亂數，從圖 5.13 可以看出雖然邊界類似，但 KNN 在 K=5 時的誤判率是比 K=15 還低的，這表示不代表 K 值取的越大就會越準確，因為取較大的 K 會受到比較多的干擾，如果資料不是相當複雜的話，取適當的 K 值即可。

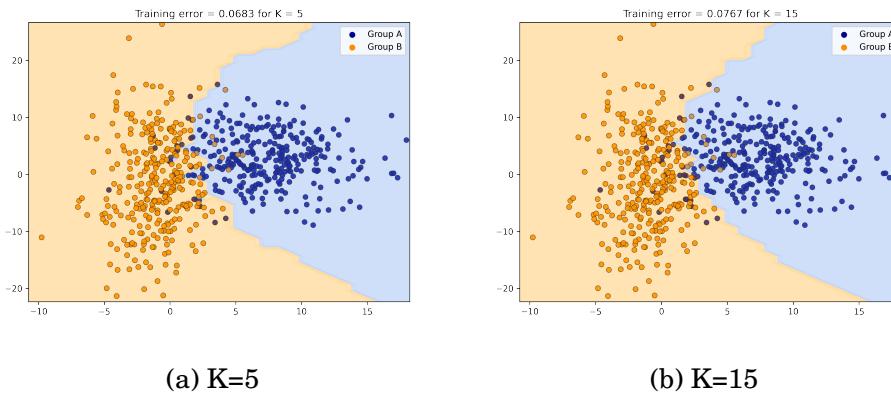


圖 5.14: 共變異數矩陣不同分類

第三筆資料是共變異數矩陣不同的常態亂數，從圖 5.14 可以看出 KNN 在這個資料的誤判率是稍微勝過 LDA(0.0833)、QDA(0.0767)，而 K=5 依然比 K=15 表現來的好，特別的是這四種情況的分類界線都不一樣，各有各的決策方法。

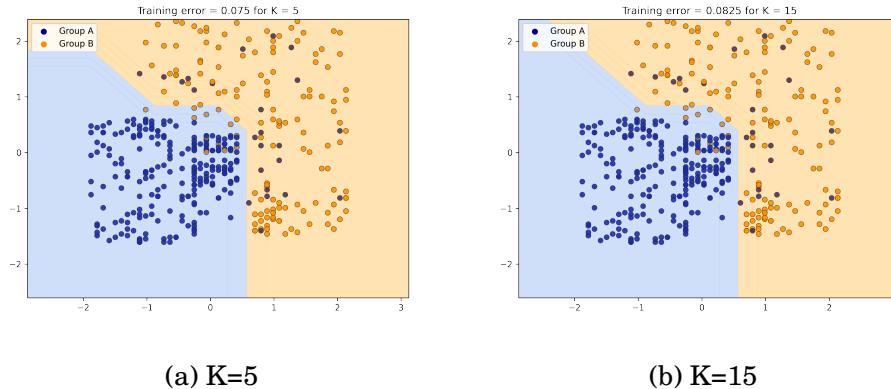


圖 5.15: 消費者資料分類

第四筆資料是消費者資料，可以看出 KNN 在這份資料中的誤判率遠勝過 LDA(0.15) 與 QDA(0.1)，不管是在  $K=5$  或是  $K=15$ ，前一小節有提到這份資料是一個不規則形狀類似一個，其邊界類似一個鏡像  $\Gamma$  形狀，雖然利用 QDA 已經很好的改善 LDA 的分類結果，但判別分析 (DA) 終究是需要一條直線或平滑曲線將樣本分群，無法完全貼合這筆資料的鏡像  $\Gamma$  形狀邊界，因為即使是平滑曲線，其仍舊無法做到在某個點直接 90 度轉向，而 KNN 很好的做到了貼合資料分群邊界，驗證了在不規則邊界下，表現甚至會比能適應不規則邊界的 QDA 好。

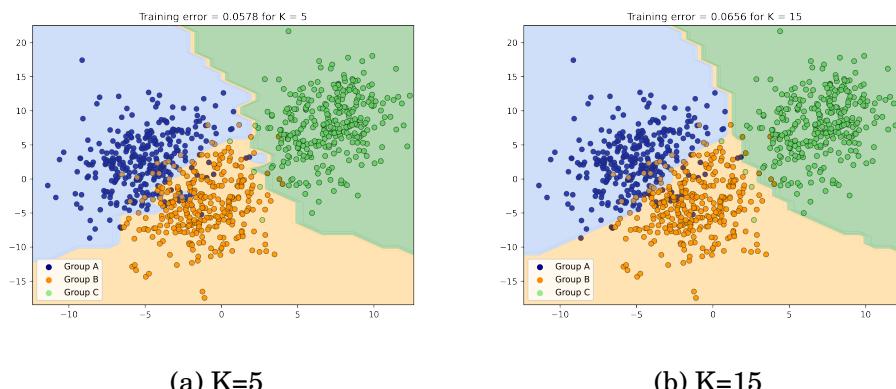


圖 5.16: 共變異數矩陣相同的三分類資料

圖 5.17 開始進入三分類的資料，這筆資料是相同共變異數矩陣的三分類樣本，無論是在  $K=5$  或是  $K=15$ ，誤判率都比 LDA(0.0711)、QDA(0.0711) 還低，三者大致分界都類似一個倒 Y，但 KNN 在資料交會的地方做出較多正確的判斷，因此取得比較低的誤判率。

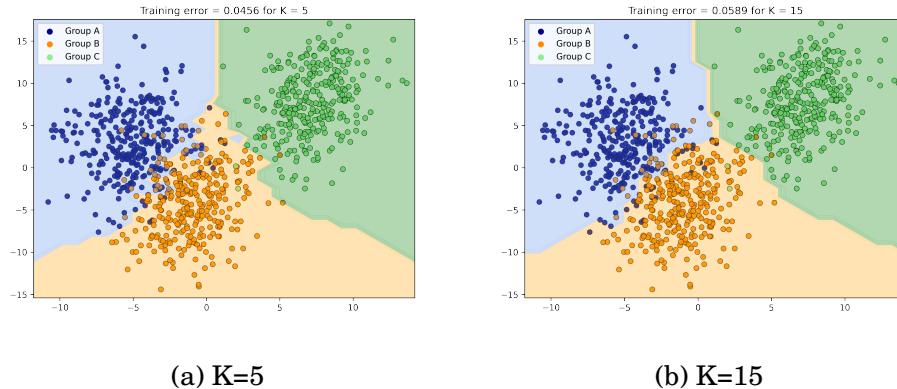


圖 5.17: 共變異數矩陣不同的三分類資料

依舊與第五筆資料的結果相同，誤判率都明顯比 LDA、QDA 還要低，證明了 KNN 確實不需要對資料進行假設，仍舊能有比較低的誤判率，也證明在低維度、資料比較簡單的情況，KNN 是一個很好的分類學習器。

上述六筆資料分類圖展示了 KNN 的特色，也就是非線性的邊界，在大部分的資料中，表現都比兩個判別分析來的出色，其中 K=5 又比 K=15 出色，但有時非線性邊界表現不一定比較好，仍需要視樣本各類別的資料特徵而定。

## 5.4 結論

本文介紹三種透過後驗分配計算決策邊界的學習器，三者特色皆有不同，以下利用表格展示三種模型在二分類與三分類資料的誤判率：

表 5.1: 三種模型在三筆二分類資料的誤判率

模型	共變異數矩陣相同	共變異數矩陣不同	消費者資料
LDA	0.1283	0.0833	0.15
QDA	0.1283	0.0767	0.10
KNN(K=5)	0.1083*	0.0683*	0.075*
KNN(K=15)	0.12	0.0767	0.0825

在二分類資料中，KNN 很顯然的贏過 LDA 與 QDA，而 QDA 又略勝於 LDA，但前三筆資料中差異並不大，可以說三者分類能力在分界明顯、資料離散程度大的時候，正確率是差不多的，因此在正確率差不多的情況下，使用比較簡易、易解釋的 LDA 或許會更方便。但當資料為非線性分類邊界時，KNN 的正確率肯定是會比 LDA、QDA 還要優



秀的，如同表 5.1 消費者資料該欄呈現，KNN 的誤判率甚至只有 LDA 的一半，顯然 KNN 對資料的限制並不大。在 K 值選擇上，由於資料並不複雜，因此 K 選擇 5 即可，選擇過大的 K 在這四份資料中反而造成誤判率更高，所以在實際分析資料時，需反覆嘗試出適合的 K 值，K 值的大小並不保證正確率。

表 5.2: 三種模型在兩筆二分類資料的誤判率

模型	共變異數矩陣相同	共變異數矩陣不同
LDA	0.0711	0.0566
QDA	0.0711	0.0611
KNN(K=5)	0.0578*	0.0456*
KNN(K=15)	0.0656	0.0589

在三分類資料中，結果大致上與二分類結果相同，都是 KNN 表現最出色，但在共變異數矩陣不同時，QDA 的表現竟然是比 LDA 差的，推論應該是資料產生時邊界就已呈現線性，因此使用 QDA 不會比 LDA 更有效率。KNN 則是與前面的結果相同，選擇 K=5 即可。

藉由三者的比較希望能讓讀者更了解 LDA、QDA 與 KNN 的不同與相同之處，或是在什麼情況該使用何者，畢竟分類問題沒有準確的答案，需要的是一次又一次的嘗試，才能找出屬於該筆資料最適合的分類學習器。





## 第 6 章

# 監督式學習之類神經網路的原理及應用

近年神經網路的學習方法崛起，產生如 CNN、RNN、ANN 等多種神經網路的分支，而本文將介紹類神經網路 (ANN)，一種在 90 年代相當熱門的學習器，過去礙於硬體設備限制了 ANN 的發展，直至最近才因為設備的進步得以發揮其真正的效能。本文將藉由簡易的原理敘述，再透過資料來實際操作 ANN，以實現 ANN 的應用。

## 6.1 類神經網路之原理

作為最初始發展的神經網路，ANN 的原理並不太複雜，透過模擬人類腦神經的運作，將  $p$  維的資料輸入後產生  $r$  個輸出值，圖 6.1 展示了其運作原理，其中中間的神經網路又分為輸入層、隱藏層與輸出層，各層中的節點代表的是神經元，而隱藏層如何運作是不得而知的，分析人員僅知道其運作原理，至於怎麼產生最後的輸出值無法得知，但可以透過設定隱藏層的層數、神經元數與節點的激發函數，改變神經網路的輸出過程。

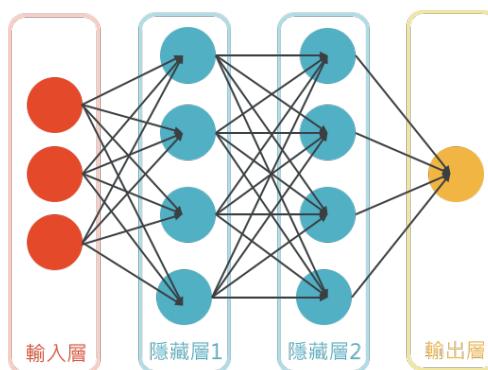


圖 6.1: 類神經網路的原理圖示

當我們對神經元輸入  $p$  個變數 ( $x_1 \sim x_p$ ) 後，經過激發函數 ( $\sigma$ ) 與內部迴歸模型對輸入的權重 ( $w_i$ ) 加乘，再加入偏誤 ( $b$ )，便完成了該節點的輸出。接著輸出會再傳給下一個神經元，作為該神經元的輸入



值，如此一層層傳遞下去，直到最後一層的輸出層，產生預測結果，式(6.1)即每個節點的輸出公式：

$$f_{w,b}(x) = \sigma \left( \sum_i w_i x_i + b \right) \quad (6.1)$$

因此可以知道類神經網路是一種非線性的模型，且隨著資料維度越大、隱藏層越多，所需要估計的參數就更多，這正是為何 ANN 為何會遇到前述硬體設備不夠先進的問題，然而一旦設備進步，那麼其在  $x, y$  的適配程度非常很高的。

## 6.2 類神經網路之應用

類神經網路之應用非常廣泛，從基礎分類到複雜的影像辨識都能使用，本文將利用兩個範例：機械手臂運動、字母數字圖形辨識，來演示使用類神經網路分析的過程與結果。

### 6.2.1 機械手臂運動

圖 6.2 是一張平面的機械手臂運動的示意圖，其手臂有固定長度的兩截 ( $l_1, l_2$ )，透過改變  $\theta_1$  與  $\theta_2$  能控制手臂的移動方位，使得機械手臂的活動範圍會如圖 6.6 散佈在兩個弧形之間，其中範圍內任何一點為機械手臂鉗子的位置，而鉗子在各位置的機率皆相同。

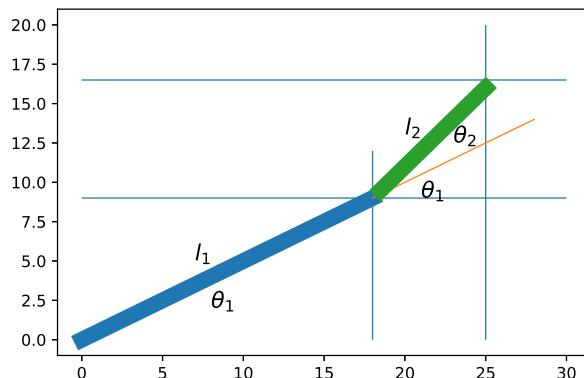


圖 6.2: 機械手臂示意圖

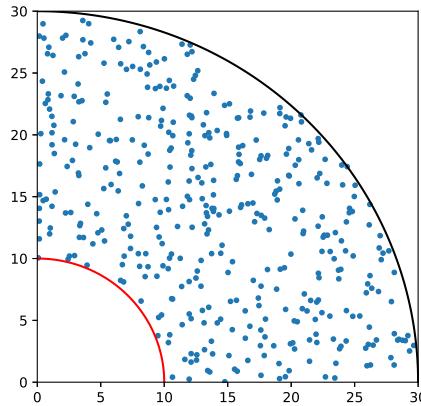


圖 6.3: 機械手臂活動範圍

為了訓練機械手臂能到正確位置夾取物品，因此需要讓機械手臂有座標的概念，但機械手臂能控制的只有角度，所以需要將角度透過極座標轉換成一般座標，言下之意即輸入給定座標，訓練機械手臂能產出正確的角度，最後將手臂移動到該處。為此，輸入的  $x$ 、 $y$  座標將透過以下算式表達：

$$x = l_1 \cos(\theta_1) + l_2 \cos(\theta_1 + \theta_2)$$

$$y = l_1 \sin(\theta_1) + l_2 \sin(\theta_1 + \theta_2)$$

並計算反函數，得出  $\theta_1$ 、 $\theta_2$  的方程式：

$$\begin{aligned}\theta_2 &= \cos^{-1} \left( \frac{x^2 + y^2 - l_1^2 - l_2^2}{2l_1l_2} \right) \\ \theta_1 &= \tan^{-1} \left( \frac{y}{x} \right) - \tan^{-1} \left( \frac{l_2 \sin(\theta_2)}{l_1 + l_2 \cos(\theta_2)} \right)\end{aligned}$$

如此便能將  $x$ 、 $y$  座標作為輸入， $\theta_1$ 、 $\theta_2$  作為輸出，讓機械手臂使用類神經網路進行學習。以下第一個範例使用較單純的 `linspace` 語法，生成了一個簡單的訓練資料，希望利用類神經網路，使得給定座標能夠產出正確的角度：

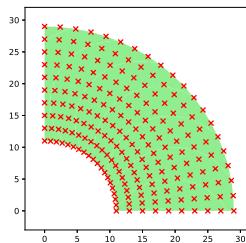


圖 6.4: 機械手臂分佈點的訓練資料



在將訓練資料放入類神經網路前，需要考慮到不同的參數設定，尤其在隱藏層中神經元的數量。前一小節有提到層數越高、神經元越多，需要估計的參數就越多，也就代表著運算時間會更長，因此該如何選擇隱藏層數、神經元個數是一件重要的事，最理想的情況當然是希望能在最少的層數下有最少的誤差，為此，本文在這筆資料中設定了每層 10、20、40、80 個神經元四種情況，來觀察神經元個數對模型的影響，結果如下圖所示：

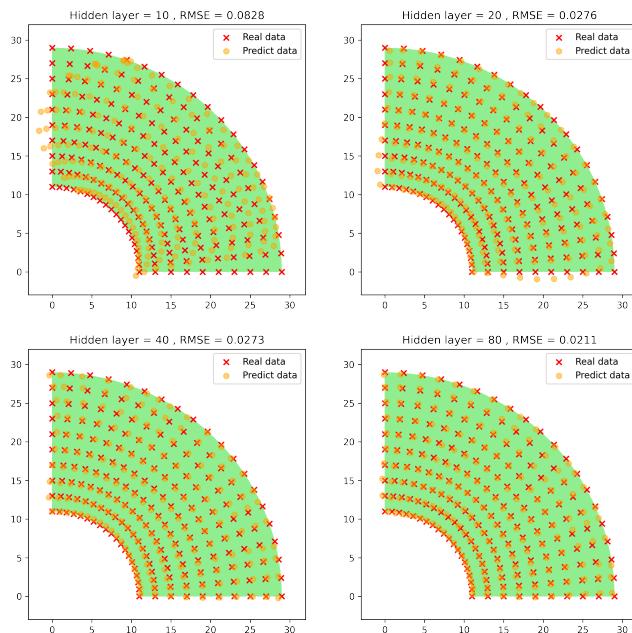


圖 6.5: 機械手臂在不同神經元數的訓練結果

RMSE (Root Mean Squared Error) 是用來衡量誤差的一種標準，若預測點與實際點相差大，則 RMSE 也會較大，反之則較小。由圖 6.5 可以看出，隨著神經元數越來越多，RMSE 有越來越小的趨勢，但變小的速度是逐漸遞減的，從 10 個到 20 個的 RMSE 跳動最大，20 個到 80 個則相差不大，因此若考慮效率的話，20 個神經元的模型會是一個好選擇，在不需要估計太多參數下，有一個相對低的 RMSE；但若講求精確性的話，RMSE 最小的 80 神經元模型依然是比較好的選擇。

這筆訓練資料的好處是較簡單，但可以看出並不符合勾爪位置隨機分佈的設定，尤其在越靠近左下方的資料點會較密集，右上方則較鬆散。為了解決該問題，於是這次使用了均勻分配的亂數，先產生一個給定圓內的均勻亂數，再透過適當裁減，將不屬於機械手臂活動範圍的樣本點刪除，最後形成如下圖的樣本：

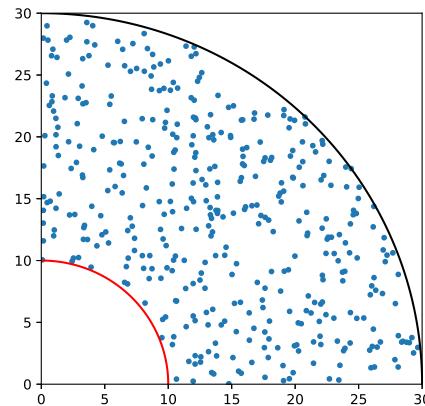


圖 6.6: 機械手臂分佈點的訓練資料

接著，同樣使用 10、20、40、80 四種神經元數的模型來觀察哪一種模型的 RMSE 較低，與前者不同的在於，此資料會分為訓練資料 (70%) 與測試資料 (30%)，因此 RMSE 所衡量的會是測試資料的部分，以下為訓練結果：

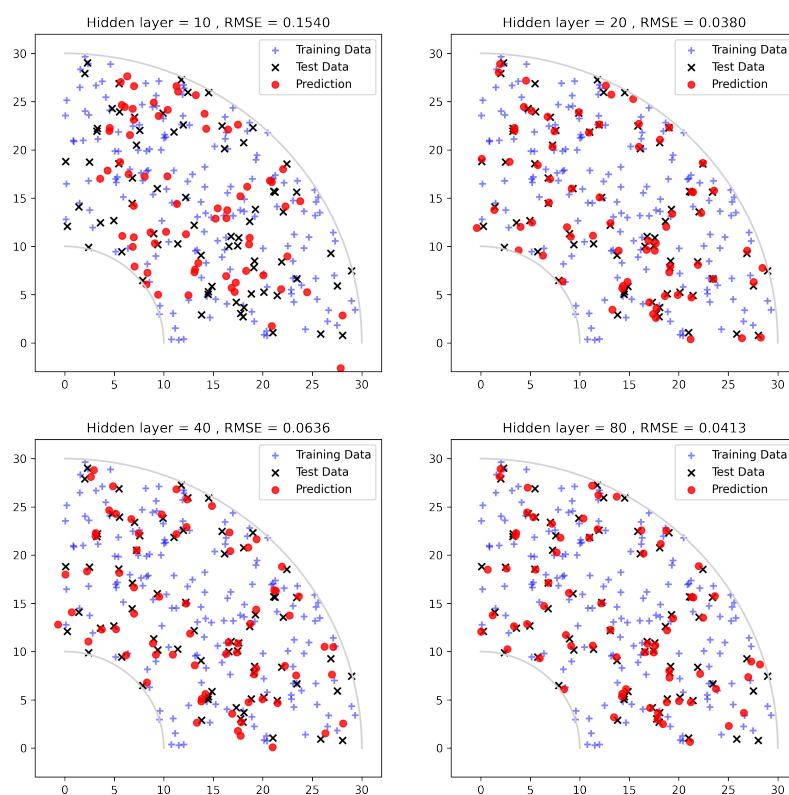


圖 6.7: 機械手臂分佈點的訓練資料與測試資料



在這個例子中，在 20 個神經元數模型的 RMSE 達到了最低，在 40 個與 80 個反而增加，因此可知並不是神經元個數越多，對預測結果會更適配，訓練資料的品質、分布也是一個重要的因素，所以當實務上應用時，應該多嘗試幾次，因為 ANN 的隱藏層如何運作無法得知，但訓練資料的品質是可以控制的。

因此為了增加訓練資料的參考性，本文進一步探討不同樣本數下的模型差異，也就是檢視樣本數的多寡是否會影響模型的表現，以下展示的是總樣本數為 239 與 662 下的差異：

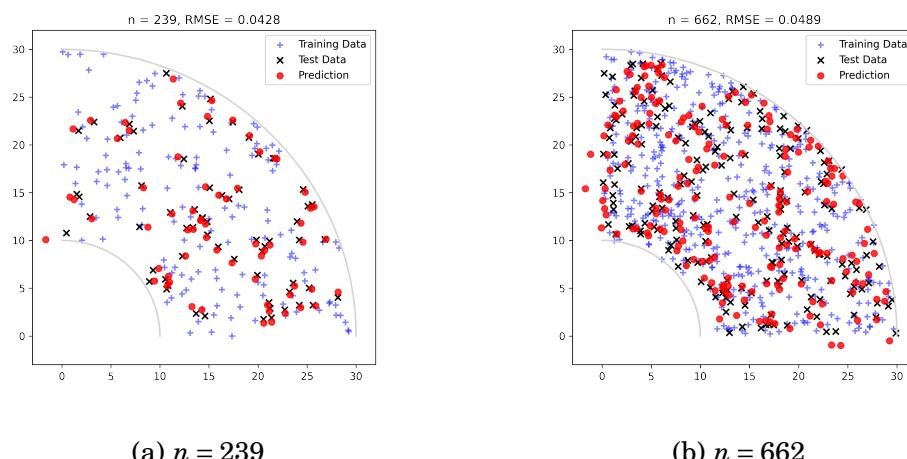


圖 6.8: 樣本數多寡對模型的影響

從圖 6.8 可以看出，樣本數增加會使得 RMSE 下降，證明了訓練資料的品質確實會影響模型的預測能力，因此神經元個數固然重要，但樣本品質也是不可忽略的一環。

上述訓練資料的程式，所使用的是 `sklearn` 中的 `MLPRegressor` 套件，除了這個套件外，`NeuroLab` 也是進行類神經網路學習的套件，其參數設定與 `MLPRegressor` 大同小異，但算法有一些不同，至於不同之處套件的開發者並沒有多著墨。`NeuroLab` 除了能進行學習外，還能把計算所需的迭代次數、誤差輸出，讓分析人員能了解一部分運算的過程。

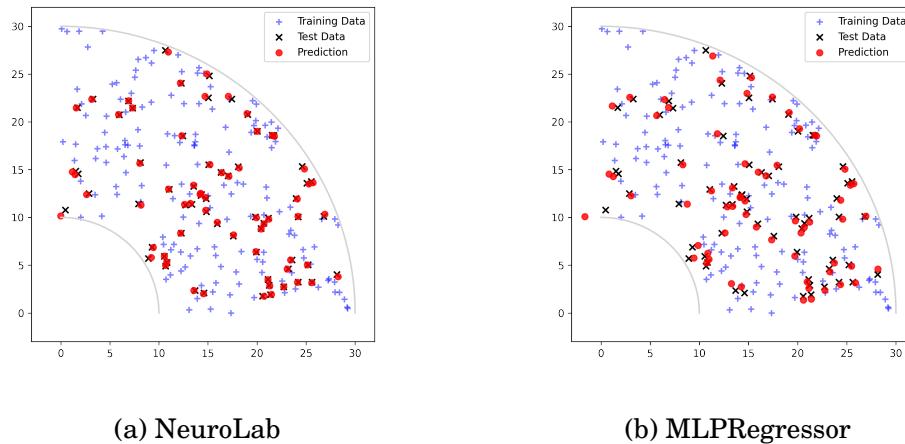


圖 6.9: 不同套件的展示

圖 6.9 (a) 是使用 NeuroLab 套件對資料做訓練，資料採用的是樣本數為 239 的樣本，神經元個數設置為 20，圖 6.9 (b) 則是同筆資料使用 MLP 的結果，可以看出在這筆資料中，使用 NeuroLab 的預測結果似乎比較好，大部分的預測點都比 MLPRegressor 來得更靠近資料點，所以可知不同套件也有可能影響模型的優劣。

## 6.2.2 手寫圖形辨識

第二個例子是手寫圖形辨識，圖形辨識是神經網路最常見的用途。圖形資料的維度較大，複雜度高，但透過神經網路層次的運作結構，讓每一層都能從前一層中學習更加抽象和高級的特徵，這種階層式學習使得神經網路能夠捕捉複雜的模式和關係，並將結果輸出。

The Montage of handwriting digits
8 5 6 8 7 5 8 0 9 7 1 3 4 7 4 8 8 8 / 8 5 4 4 0 1 0 3 8 0 8 1
9 1 4 2 4 0 7 2 3 0 0 2 6 1 + 1 / 6 2 2 6 1 1 8 1 9 0 4 8 9
6 1 / 4 7 4 5 4 - 1 0 8 3 6 0 8 / 3 1 2 7 1 / 8 1 4 4 0 0 3
3 8 2 8 0 2 0 6 1 6 1 9 9 9 1 1 3 9 5 4 7 0 5 7 5 9 6 0 2 3
6 1 8 9 8 8 0 7 9 9 9 5 8 4 5 1 7 0 2 8 0 / 4 0 6 7 3 6 2 4
6 9 8 4 2 1 4 1 3 7 9 5 7 1 9 4 9 7 5 1 2 3 7 2 8 6 0 6 7 7
7 9 8 8 5 2 3 7 6 9 4 4 7 8 9 8 8 / 5 4 5 0 2 3 2 7 \ 8 1
1 / 1 8 5 3 9 2 5 6 0 1 5 0 9 3 8 5 6 3 2 7 5 6 9 2 8 3 4 9 3
9 4 0 8 4 8 4 7 7 1 3 1 8 0 5 3 4 9 0 4 7 8 5 2 0 / 7 1 1 2
0 7 2 5 8 4 1 0 0 2 7 8 4 4 6 0 3 0 9 3 5 4 3 2 9 1 3 9 0 7
9 0 9 8 7 4 3 1 9 4 8 6 0 5 0 0 5 8 3 3 2 / 5 5 0 1 8 4 5 6
6 1 1 5 3 8 3 2 1 2 / 2 1 3 8 9 2 2 6 7 5 0 3 1 0 8 5 3 1 5
8 4 9 4 9 2 0 0 7 3 7 4 8 0 5 6 6 1 6 3 5 6 6 0 2 0 0 2 4 8
4 4 0 8 2 8 4 5 6 7 7 3 3 7 1 2 5 2 9 3 0 4 4 3 6 1 8 5 4 3
8 4 9 5 4 4 6 3 3 4 0 5 9 9 0 5 0 1 8 1 0 6 2 8 8 8 9 9 8 4
3 5 5 2 5 0 8 0 5 3 1 9 6 1 7 4 0 3 6 9 8 3 3 3 0 6 3 4 4 1 7
3 9 1 0 6 1 5 0 3 4 9 5 5 3 1 5 8 1 0 2 5 5 1 2 1 3 0 5 1 9
7 2 0 2 1 1 8 9 4 9 9 1 9 4 8 5 7 2 5 0 9 9 0 3 9 1 6 3 1 8
0 8 6 2 0 4 9 1 8 5 0 6 6 4 1 / 7 3 5 6 3 5 2 4 8 4 5 8 6
9 2 9 8 8 2 2 7 9 5 2 9 6 0 0 9 8 4 4 9 6 8 8 8 5 7 5 3 0 2

圖 6.10: 手寫數字資料



本例使用的資料是一筆手寫數字資料，樣本數共 1000，資料來自從美國郵局用戶蒐集的手寫數字，如圖 6.10 所呈現。由於每個人寫字的方式、面貌不一，所以需要一個系統來辨識肉眼無法辨認的手寫數字，因此本小節將透過手寫資料來訓練，利用類神經網路盡可能讓模型能夠正確辨認數字。

在神經元數設定為 30 下，模型的結果如下圖的矩陣所呈現，此矩陣是一個混淆矩陣 (confusion matrix)，對角線上的數字代表正確辨識的資料數，非對角線則為辨識錯誤的資料數，可以看到標題的 Accuracy 為 89.6%，表示正確辨識之資料 (對角線上的資料) 共佔了 86.0%，最容易誤判的則是 4 和 9，共四次被錯誤判斷。

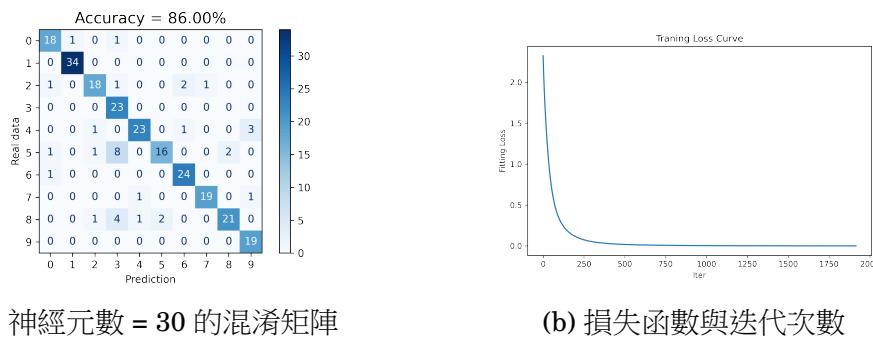


圖 6.11: 手寫數字資料辨識結果 (神經元數 = 30)

接著嘗試 40 個神經元的模型：

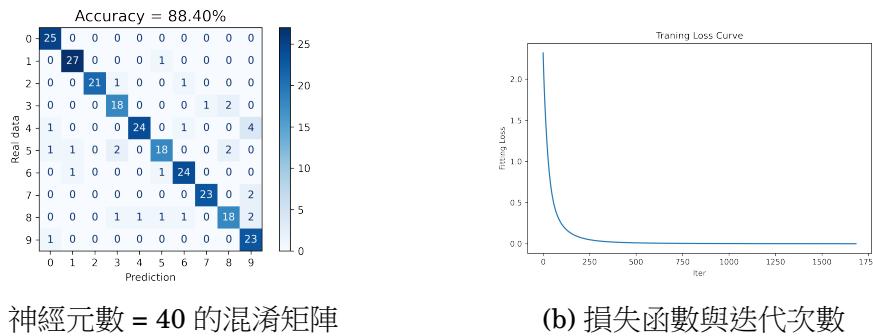


圖 6.12: 手寫數字資料辨識結果 (神經元數 = 40)

可以看到準確率從 86% 上升至 88.4%，且迭代次數甚至比神經元為 30 的模型還低，雖然不見得神經元數越多模型越好，但可知可以嘗試不同的神經元，找到一個最好的結果。

下一個例子使用的是手寫英文字母資料，依舊是利用類神經網路訓練，讓無法肉眼辨識的字母透過電腦正確辨識，但這筆資料的樣本數增加到了 10000，類別種類增加至 26，相對數字資料而言複雜許多。以下為分別採用 30 個神經元與 40 個神經元進行訓練產生的混淆矩陣：



圖 6.13: 手寫字母資料

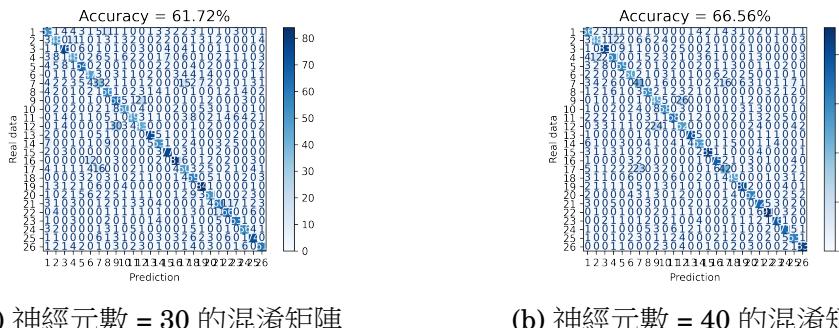


圖 6.14: 不同神經元的混淆矩陣比較

從圖 6.14 可以看出其正確率相對數字資料而言降低很多，因此可以發現在資料變得更加複雜時，正確判斷的難度會上升，並且程式執行時間由原本的 20 秒變成將近 2 分鐘，增加了將近 6 倍的時間，雖然迭代次數都是 1000 多，但每次迭代的計算時間會因為資料變得複雜而增加。另外，在本例中增加較多神經元數的模型也依舊比較低神經元的表現好，這也驗證了一開始所說，如果硬體設備足夠，那類神經網路的表現會越來越好。

### 6.3 結論

本文介紹了類神經網路的運作原理，也展示了實際使用類神經網路的分析過程，更驗證影響類神經網路表現的三個因素：神經元個數、訓練資料品質與不同套件的差異，當然，影響模型表現的絕對不只這三個因素，在程式碼中還有許多參數可以進行微調，可能一點微調都會對模型有莫大的改善。類神經網路與過去比較簡單的模型不同，在使用較簡單模型時，分析人往往會花更多時間在解釋模型與思考模型合理性，但類神經網路則不然，由於過程不得而知，便轉而要求結果表現好、硬體設備可以承受即可，所以反而會在模型微調上下更多的功



夫，這是與過去簡單模型最大的不同。希望透過本文能讓讀者了解類神經網路的背景與應用，並進一步往深度學習領域探究。



## 第 7 章

### 多種學習器的比較

前面的章節介紹了多種學習器，雖然它們的原理各不相同，但其共同目標是將樣本進行正確分類。因此，在這一章節中，我們將運用相同的資料集來訓練不同的學習器，並比較它們在相同資料下的訓練成果，找出優劣之處。為了確保公平的抽樣，本文將採用蒙地卡羅模擬進行重複抽樣，以最小化抽樣所產生的誤差。

#### 7.1 蒙地卡羅模擬

蒙地卡羅模擬是一種數學的計算方法，主要用於模擬和分析不確定性事件，這種方法依靠隨機抽樣和統計推斷來解決問題，特別適用於沒辦法準確預測結果的複雜系統。蒙地卡羅方法通常分為三個步驟：

- 第一步 通過隨機抽樣生成大量可能的輸入值
- 第二步 將這些輸入值放入模型進行運算與模擬
- 第三步 通過統計分析處理這些結果，以做出統計推論

蒙地卡羅模擬在金融、工程、物理學、生物學等領域都有其應用，因為它能夠處理高度複雜和不確定的問題，所以受到很大青睞。

#### 7.2 模型比較

本小節將使用兩筆資料進行模型訓練，而每次訓練都會重新分割訓練資料與測試資料，達到重複抽樣的目的。最後，將資料放進六個選定的模型進行比較。

##### 7.2.1 亂數產生資料

在樣本設定為 200 下，分別生成兩個連續型變數的特徵與一個類別變數，以下是常態亂數的參數設置與資料散佈圖：

$$\mu_0 = \begin{bmatrix} -2 \\ 3 \end{bmatrix}, \mu_1 = \begin{bmatrix} -1 \\ -2 \end{bmatrix}, \Sigma_0 = \begin{bmatrix} 15 & 1.2 \\ 1.2 & 20 \end{bmatrix}, \Sigma_1 = \begin{bmatrix} 5 & 2 \\ 2 & 18 \end{bmatrix}$$

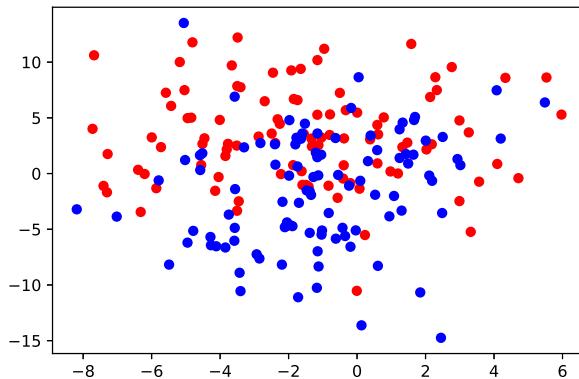


圖 7.1: 常態亂數資料

在進行重複抽取訓練資料 (70%) 100 次後，將資料放入分別的七個模型：LDA、QDA、Logistic Regression、KNN(K=5)、KNN(K=15)、ANN(神經元數 10)、ANN(神經元數 20)，進行模型配適，最後得到的結果如下：

表 7.1: 七種模型在亂數資料的誤判率

模型	<b>LDA</b>	<b>QDA</b>	<b>Logistic</b>	<b>KNN(K=5)</b>	<b>KNN(K=15)</b>
誤判率	0.2173	0.218	0.2052	0.2175	0.213
模型	<b>ANN(神經元數 10)</b>		<b>ANN(神經元數 20)</b>		
誤判率		0.2082		0.2502	

在這筆資料中，七種模型的差異不大，除了 ANN(神經元數 20) 模型的誤判率比較大之外，其他誤判率都介於 0.2 ~ 0.22 之間。

## 7.2.2 消費者資料

第二筆資料是來自 kaggle 的一份消費者購買行為的分類資料，其特徵包括兩個：觀測者的年齡與薪資，類別則是其購買與否 (0、1)，樣本數為 400，以下為資料散佈圖：

同樣重複抽取訓練資料 (70%) 100 次後，將資料放入分別的七個模型，最後得到的結果如下：

表 7.2: 七種模型在消費者資料的誤判率

模型	<b>LDA</b>	<b>QDA</b>	<b>Logistic</b>	<b>KNN(K=5)</b>	<b>KNN(K=15)</b>
誤判率	0.1528	0.1517	0.0988	0.0789	0.0884
模型	<b>ANN(神經元數 10)</b>		<b>ANN(神經元數 20)</b>		
誤判率		0.1232		0.1116	

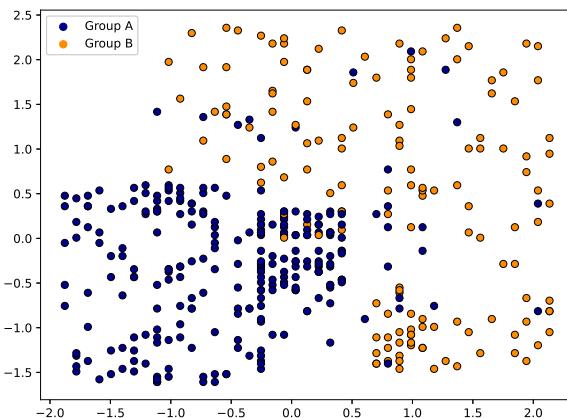


圖 7.2: 消費者購買行為資料

從表 7.2 可以看出在這筆資料中誤判率有些差異，KNN(K=5) 的誤判率是最低的，LDA 則是最高等級，特別的是這筆資料的執行時間從上一筆的 5 分鐘變成 20 分鐘，可知樣本數對運算時間有很巨大的影響，在樣本增加 2 倍時，時間增加了 4 倍。

### 7.3 結論

從兩份資料來看，六個模型的差異並不是到特別大，但礙於 LDA、QDA 較適合放入低維資料，因此無法透過高維資料來同時比較這六個模型，這也導致了擅長處理高維資料的 ANN 無法展現其長處。機器學習的學習器眾多，每一種學習器都其有適合的資料，也因為機器學習較難進行統計推論，所以選擇模型會是一個最重要的課題。

