

X<sub>E</sub>T<sub>E</sub>X

X<sub>E</sub>T<sub>E</sub>X

# 統計應用數學與計算

郭翊萱

March 31, 2023



# 目錄

序	vi
<b>1 X<sub>E</sub>ΛT<sub>E</sub>X 基礎應用教學</b>	<b>1</b>
1.1 基本操作 . . . . .	1
1.1.1 頁面配置 . . . . .	1
1.1.2 文字設定 . . . . .	4
1.2 數學符號與方程式 . . . . .	7
1.2.1 常用數學符號 . . . . .	8
1.2.2 函數 . . . . .	10
1.2.3 聯立方程式 . . . . .	11
1.2.4 矩陣 . . . . .	13
1.2.5 範例 . . . . .	15
1.3 表格 . . . . .	18
1.3.1 基本表格運用 . . . . .	18
1.3.2 加入底色的表格 . . . . .	20
1.3.3 表格並列 . . . . .	20
1.3.4 文字表格並排 . . . . .	21
1.3.5 複雜表格建立 . . . . .	21
1.3.6 表格單元格合併 . . . . .	21
1.3.7 表格粗細 . . . . .	22
1.3.8 表格單元欄列合併 . . . . .	22
1.3.9 寬度過寬表格 . . . . .	23
1.3.10 長度過長表格 . . . . .	23
1.4 圖片 . . . . .	25
1.4.1 圖片並排 . . . . .	26

1.4.2 文字與圖片並排 . . . . .	26
1.4.3 圖片旋轉 . . . . .	27
1.5 結論 . . . . .	27
<b>2 Python 基本函數繪圖</b>	<b>29</b>
2.1 基本機率函數繪圖 . . . . .	29
2.2 專題練習 . . . . .	37
2.3 結論 . . . . .	40
<b>3 常見機率分配與抽樣分配</b>	<b>41</b>
3.1 離散型分配 . . . . .	41
3.1.1 二項分配 . . . . .	41
3.1.2 超幾何分配 . . . . .	43
3.1.3 幾何分配 . . . . .	44
3.1.4 卜瓦松分配 . . . . .	44
3.2 連續型分配 . . . . .	51
3.2.1 卡方分配 . . . . .	51
3.2.2 指數分配 . . . . .	55
3.2.3 雙指數分配 . . . . .	56
3.2.4 伽瑪分配 . . . . .	57
3.2.5 貝塔分配 . . . . .	61
3.2.6 柯西分配 . . . . .	64
3.2.7 常態分配 . . . . .	65
3.2.8 司徒頓 t 分配 . . . . .	65
3.2.9 F 分配 . . . . .	66
3.3 專題討論 . . . . .	67
3.4 結論 . . . . .	68
<b>4 迴歸分析與羅吉斯迴歸的分類問題</b>	<b>69</b>
4.1 機器學習基礎理論 . . . . .	69
4.1.1 監督式學習與非監督式學習 . . . . .	69
4.1.2 分類與迴歸問題 . . . . .	70
4.2 簡單線性迴歸模型 . . . . .	71

---

4.2.1 簡單線性迴歸理論 . . . . .	71
4.2.2 簡單線型迴歸分類方法 . . . . .	71
4.2.3 簡單線性迴歸分類範例 . . . . .	72
4.3 加廣迴歸模型 . . . . .	75
4.4 簡單線性迴歸與加廣迴歸模型比較 . . . . .	76
4.5 羅吉斯迴歸模型 . . . . .	81
4.5.1 羅吉斯迴歸理論 . . . . .	81
4.5.2 羅吉斯迴歸分類範例 . . . . .	82
4.6 結論 . . . . .	85
<b>5 判別分析與 K-近鄰演算法的分類問題</b>	<b>87</b>
5.1 線性判別分析、二次判別分析與 K-近鄰演算法 . . . . .	87
5.1.1 線性判別分析 . . . . .	87
5.1.2 線性判別分析分類效果 . . . . .	92
5.1.3 二次判別分析 . . . . .	93
5.1.4 線性判別分析與二次判別分析比較 . . . . .	93
5.1.5 K-近鄰演算法 . . . . .	97
5.1.6 K-近鄰演算法分類效果 . . . . .	98
5.2 線性判別分析、二次判別分析與 K-近鄰演算法的分類效果 . . . . .	105
5.2.1 線性判別分析、二次判別分析與 K-近鄰演算法的兩群組分類 . . .	106
5.2.2 線性判別分析、二次判別分析與 K-近鄰演算法的三群組分類 . .	111
5.3 結論 . . . . .	115
<b>6 類神經網路的分類問題</b>	<b>117</b>
6.1 類神經網路理論介紹 . . . . .	117
6.2 機械手臂方程式 . . . . .	120
6.2.1 最佳化模型 . . . . .	121
6.3 圖形識別 . . . . .	127
6.4 結論 . . . . .	129
<b>7 蒙地卡羅模擬實驗：多種分類方法的評比</b>	<b>131</b>
7.1 七種分類方法介紹 . . . . .	131
7.1.1 簡單線性迴歸簡述 . . . . .	131

7.1.2 加廣迴歸簡述 . . . . .	132
7.1.3 羅吉斯迴歸簡述 . . . . .	132
7.1.4 線性判別分析簡述 . . . . .	133
7.1.5 二次判別分析簡述 . . . . .	133
7.1.6 K-近鄰演算法簡述 . . . . .	134
7.1.7 類神經網路模型簡述 . . . . .	134
7.2 七種機器學習方法對兩群組資料的分類問題 . . . . .	136
7.2.1 兩群組更改樣本數大小的分類問題 . . . . .	136
7.2.2 兩群組更改群平均的分類問題 . . . . .	138
7.2.3 兩群組更改群共變異數的分類問題 . . . . .	139
7.3 七種機器學習方法對三群組資料的分類問題 . . . . .	141
7.3.1 三群組更改樣本數大小的分類問題 . . . . .	141
7.3.2 三群組更改群平均的分類問題 . . . . .	142
7.3.3 三群組更改群共變異數的分類問題 . . . . .	143
7.4 結論 . . . . .	145
<b>8 參考文獻的使用與引用</b>	<b>147</b>
8.1 初步觀念 . . . . .	147
8.2 參考文獻的引用：作者與年份 . . . . .	147
8.3 文獻引用方式 . . . . .	148
8.4 製作方式 . . . . .	148

# 圖目錄

圖 1.1 範例圖 (jpeg 圖) . . . . .	25
圖 1.2 圖形並排的作法 . . . . .	26
圖 1.3 旋轉圖片 . . . . .	27
圖 2.1 $f(x) = \sin(x) + \cos(x)$ . . . . .	30
圖 2.2 $f(x) = (1 - e^{-2x})/(1 + e^{-2x})$ . . . . .	30
圖 2.3 $f(x) = \sqrt[3]{(4 - x^3)/(1 + x^2)}$ . . . . .	31
圖 2.4 $f(x) = 1/x$ . . . . .	32
圖 2.5 $f(x) = x^{2/3}$ . . . . .	33
圖 2.6 $f(x) = 1/(2\sqrt{2\pi}) \exp(-(x - 1)^2/8)$ . . . . .	33
圖 2.7 $f(x) = \ln(x)/x^3$ . . . . .	34
圖 2.8 $f(x) = 2x^3 - x^4$ . . . . .	35
圖 2.9 $f(x) = 3$ . . . . .	36
圖 2.10 <i>Square</i> . . . . .	36
圖 2.11 $S_n$ diverge when $n \rightarrow \infty$ . . . . .	37
圖 2.12 $\gamma_n$ and $S_n + \ln(n + 1)$ . . . . .	38
圖 2.13 $\frac{1}{2}(1 - \frac{1}{n+1}) < \gamma_n < 1$ . . . . .	39
圖 3.1 二項分配的機率質量函數圖與累積機率函數圖 . . . . .	42
圖 3.2 二項分配與常態分配的機率函數圖 . . . . .	43
圖 3.3 超幾何分配的機率質量函數圖 . . . . .	43
圖 3.4 幾何分配的機率質量函數圖 . . . . .	44
圖 3.5 卜瓦松分配的機率質量函數圖 . . . . .	45
圖 3.6 二項分配與卜瓦松分配的機率質量函數圖 . . . . .	45
圖 3.7 樣本數為 80 且服從 $Poisson(\lambda_1 + \lambda_2)$ 之隨機變數 $X + Y$ 的直方圖、箱型圖、qqplot 圖與經驗分布函數圖 . . . . .	46

---

圖 3.8 樣本數為 1000 且服從 $Poisson(\lambda_1 + \lambda_2)$ 之隨機變數 $X + Y$ 的直方圖、箱型圖、qqplot 圖與經驗分布函數圖 . . . . .	47
圖 3.9 樣本數為 20 的卜瓦松抽樣分配直方圖 . . . . .	48
圖 3.10 樣本數為 100 的抽樣卜瓦松分配直方圖 . . . . .	49
圖 3.11 樣本數為 1000 的抽樣卜瓦松分配直方圖 . . . . .	49
圖 3.12 樣本數為 20 的抽樣卜瓦松分配 qqplot 圖 . . . . .	49
圖 3.13 樣本數為 100 的抽樣卜瓦松分配 qqplot 圖 . . . . .	50
圖 3.14 樣本數為 1000 的抽樣卜瓦松分配 qqplot 圖 . . . . .	50
圖 3.15 卡方分配的機率密度函數圖 . . . . .	51
圖 3.16 卡方分配、常態分配與伽瑪分配的機率密度函數圖 . . . . .	52
圖 3.17 樣本數為 100 的 $\chi^2(1)$ 直方圖、箱型圖、qqplot 圖與經驗分布函數圖 . . . . .	53
圖 3.18 樣本數為 10000 的 $\chi^2(1)$ 直方圖、箱型圖、qqplot 圖與經驗分布函數圖 . . . . .	54
圖 3.19 指數分配的機率密度函數圖 . . . . .	55
圖 3.20 錯誤的指數分配機率密度函數圖與累積機率函數圖 . . . . .	56
圖 3.21 雙指數分配的機率密度函數圖與累積機率函數圖 . . . . .	57
圖 3.22 伽瑪分配的機率密度函數圖 . . . . .	58
圖 3.23 樣本數為 100 的 $\Gamma(\alpha_1 + \alpha_2, \beta_1 + \beta_2)$ 機率密度函數圖、qqplot 圖與經驗分布函數圖 . . . . .	58
圖 3.24 樣本數為 1000 的 $\Gamma(\alpha_1 + \alpha_2, \beta_1 + \beta_2)$ 機率密度函數圖、qqplot 圖與經驗分布函數圖 . . . . .	59
圖 3.25 樣本數為 50 的伽瑪抽樣分配直方圖 . . . . .	60
圖 3.26 樣本數為 1000 的伽瑪抽樣分配直方圖 . . . . .	60
圖 3.27 樣本數為 50 的伽瑪抽樣分配 qqplot 圖 . . . . .	60
圖 3.28 樣本數為 1000 的伽瑪抽樣分配 qqplot 圖 . . . . .	61
圖 3.29 貝塔分配的機率密度函數圖 . . . . .	61
圖 3.30 貝塔抽樣分配直方圖 . . . . .	62
圖 3.31 貝塔分配 qqplot 圖 . . . . .	63
圖 3.32 貝塔分配經驗分布函數圖 . . . . .	63
圖 3.33 柯西分配的機率密度函數圖與累積機率函數圖 . . . . .	64
圖 3.34 常態分配機率分布圖 . . . . .	65
圖 3.35 司徒頓 t 分配機率分布圖 . . . . .	66
圖 3.36 F 分配機率分布圖 . . . . .	67

---

圖 3.37 使用迴圈繪製取後放回直方圖 . . . . .	68
圖 4.1 資料集 <i>la_2.txt</i> 的散布圖 . . . . .	72
圖 4.2 資料集 <i>la_2.txt</i> 之簡單線性迴歸模型 . . . . .	74
圖 4.3 資料集 <i>la_2.txt</i> 之簡單線性迴歸與加廣線性迴歸模型 . . . . .	75
圖 4.4 樣本數為 1000 進行簡單線性迴歸與加廣迴歸的訓練與測試資料準確度 . . . . .	77
圖 4.5 樣本數為 200 進行簡單線性迴歸與加廣迴歸的訓練與測試資料準確度 . . . . .	77
圖 4.6 群平均為 (2, 4) 與 (5, 3) 進行簡單線性迴歸與加廣迴歸的訓練與測試資 料準確度 . . . . .	79
圖 4.7 群平均為 (4, 4) 與 (6, 5) 進行簡單線性迴歸與加廣迴歸的訓練與測試資 料準確度 . . . . .	79
圖 4.8 群共變異數為 0.4 與 0.5 進行簡單線性迴歸與加廣迴歸的訓練與測試 資料準確度 . . . . .	80
圖 4.9 群共變異數為 0.9 與 0.2 進行簡單線性迴歸與加廣迴歸的訓練與測試 資料準確度 . . . . .	81
圖 4.10 第一個羅吉斯迴歸分類實驗分類準確度 . . . . .	82
圖 4.11 第二個羅吉斯迴歸分類實驗分類準確度 . . . . .	83
圖 4.12 第三個羅吉斯迴歸分類實驗之分類準確度 . . . . .	84
圖 4.13 第四個羅吉斯迴歸分類實驗之分類準確度 . . . . .	85
圖 5.1 多變量常態母體資料之 2D 圖像 . . . . .	89
圖 5.2 多變量常態母體資料之 3D 圖像 . . . . .	90
圖 5.3 線性判別分析之二元分類範例 . . . . .	93
圖 5.4 線性判別分析之二元分類圖 . . . . .	95
圖 5.5 第一種線性判別分析與二次判別分析之二元分類圖 . . . . .	96
圖 5.6 第二種線性判別分析與二次判別分析之二元分類圖 . . . . .	97
圖 5.7 利用理論進行 K-近鄰演算法之二元分類圖 . . . . .	99
圖 5.8 利用 sklearn 進行 K-近鄰演算法之二元分類圖 . . . . .	100
圖 5.9 K-近鄰演算法之兩群組雙變量常態資料之 2D 圖 . . . . .	101
圖 5.10 K-近鄰演算法之兩群組雙變量常態資料之 3D 圖 . . . . .	101
圖 5.11 K-近鄰演算法之三群組雙變量常態資料之 2D 圖 . . . . .	102
圖 5.12 K-近鄰演算法之三群組雙變量常態資料之 3D 圖 . . . . .	102

---

圖 5.13 設置不同 K 值之 K-近鄰演算法的兩組與三組資料的訓練與測試資料分類誤差	103
圖 5.14 K-近鄰演算法之兩群組分類圖	104
圖 5.15 K-近鄰演算法之三群組分類圖	105
圖 5.16 改變樣本數實驗的兩群組雙變量常態資料散布圖	106
圖 5.17 改變群平均實驗的兩群組雙變量常態資料散布圖	108
圖 5.18 改變群共變異數實驗的兩群組雙變量常態資料散布圖	110
圖 5.19 改變群樣本數實驗的三群組雙變量常態資料散布圖	111
圖 5.20 改變群平均實驗的三群組雙變量常態資料散布圖	113
圖 5.21 改變群共變異數實驗的三群組雙變量常態資料散布圖	114
 圖 6.1 類神經網路 ( ANN ) 示意圖	118
圖 6.2 兩截式機械手臂示意圖	120
圖 6.3 扇形圖	121
圖 6.4 改變隱藏層之類神經網路模型	122
圖 6.5 套件 <i>MLPRegressor</i> 之類神經網路模型	124
圖 6.6 套件 <i>Neurolab.newff</i> 之類神經網路模型	125
圖 6.7 手寫數字範例圖	127
圖 6.8 1000 筆資料集之類神經網路分類模型的測試資料混淆矩陣	128
圖 6.9 10000 筆類神經網路分類模型的測試資料混淆矩陣	129
 圖 7.1 比較七種方法改變樣本數實驗的兩群組雙變量常態資料散布圖	137
圖 7.2 比較七種方法改變群平均實驗的兩群組雙變量常態資料散布圖	138
圖 7.3 比較七種方法改變群共變異數實驗的兩群組雙變量常態資料散布圖	140
圖 7.4 比較七種方法改變群樣本數實驗的三群組雙變量常態資料散布圖	141
圖 7.5 比較七種方法改變群平均實驗的三群組雙變量常態資料散布圖	143
圖 7.6 改變群共變異數實驗的三群組雙變量常態資料散布圖	144

# 表目錄

表 1.1	最基本的表格	7
表 1.2	羅馬符號	8
表 1.3	數學運算符號	8
表 1.4	數學運算符號	9
表 1.5	數學運算符號	9
表 1.6	數學運算符號	10
表 1.7	範例表格	19
表 1.8	含橫線跟豎線的表格	19
表 1.9	設置每一列的數值居左/居中/居右的表格	19
表 1.10	三線表格	19
表 1.11	加入底色的表格	20
表 1.12	兩個表格並列	20
表 1.13	複雜表格	21
表 1.14	合併列的表格	21
表 1.15	表格粗細	22
表 1.16	合併欄列的表格	22
表 1.17	寬度過寬表格	23
表 1.18	長型表格	23
表 2.1	套件 matplotlib 幾種繪圖函數	34
表 5.1	三種機器學習方法改變樣本數的兩群組訓練與測試資料分類錯誤率	107
表 5.2	三種機器學習方法改變群平均的兩群組訓練與測試資料分類錯誤率	109
表 5.3	三種機器學習方法改變群共變異數的兩群組訓練與測試資料分類錯誤率	110
表 5.4	三種機器學習方法改變樣本數的三群組訓練與測試資料分類錯誤率	112
表 5.5	三種機器學習方法改變群平均的三群組訓練與測試資料分類錯誤率	113

表 5.6 三種機器學習方法改變群共變異數的三群組訓練與測試資料分類錯誤率	115
表 6.1 不同隱藏層之類神經網路模型比較 . . . . .	123
表 6.2 機械手臂兩種套件改變隱藏層的類神經網路模型好壞比較 . . . . .	125
表 6.3 機械手臂兩種套件改變樣本數的類神經網路模型好壞比較 . . . . .	126
表 6.4 機械手臂兩種套件同時改變樣本數與隱藏層的類神經網路模型好壞比較	126
表 6.5 兩筆資料集改變隱藏層之測試資料分類準確度 . . . . .	129
表 7.1 七種機器學習方法改變樣本數的兩群組訓練資料分類錯誤率 . . . . .	137
表 7.2 七種機器學習方法改變樣本數的兩群組測試資料分類錯誤率 . . . . .	137
表 7.3 七種機器學習方法改變群平均的兩群組訓練資料分類錯誤率 . . . . .	139
表 7.4 七種機器學習方法改變群平均的兩群組測試資料分類錯誤率 . . . . .	139
表 7.5 七種機器學習方法改變群共變異數的兩群組訓練資料分類錯誤率 . . .	140
表 7.6 七種機器學習方法改變群共變異數的兩群組測試資料分類錯誤率 . . .	140
表 7.7 七種機器學習方法改變樣本數的三群組訓練資料分類錯誤率 . . . . .	142
表 7.8 七種機器學習方法改變樣本數的三群組測試資料分類錯誤率 . . . . .	142
表 7.9 七種機器學習方法改變群平均的三群組訓練資料分類錯誤率 . . . . .	143
表 7.10 七種機器學習方法改變群平均的三群組測試資料分類錯誤率 . . . . .	143
表 7.11 七種機器學習方法改變群共變異數的三群組訓練資料分類錯誤率 . . .	144
表 7.12 七種機器學習方法改變群共變異數的三群組測試資料分類錯誤率 . . .	145

# 序

統計學在許多人心中或許是一門玄學，它基於機率用合理的猜去做合理的推論，但是人們往往對於如何去猜沒有很清晰的認知，許多人會認為統計學就是一門詐騙術，或在政治中操控民調結果，或在統計新冠肺炎數據時用似是而非的方式來引導人們的認知，而這都是基於對陌生事物的恐懼感。

人類往往習慣先看見再相信，而對未知事物的恐懼又會阻撓人類睜開雙眼，但是 21 世紀的人類最寶貴的資產或許就是想睜眼就能看見，而不似前人頂著狂風摸著黑暗前行，一昧的逃避與拒絕了解只會憑白浪費先人與世界給予我們的饋贈，人的一生短短數十年，了解這個世界的邏輯並熱愛這個世界數十年的人生或許才不會活著而像是沒有活著，當你不能善用這個世界給你的禮物而是閉著雙眼生活的時候。

這本書希望能藉由一個單元又一個單元去慢慢引導讀者對於統計學與數學的認知，希望能逐漸引導人們消彌對數學的恐懼以及對統計學的誤解，這本書將利用程式語言來幫助人們了解統計學這片廣袤的土地，利用各種各樣的模擬實驗來幫助讀者一窺其中的有趣之處。本書將首先介紹這本書的製作方式，即利用 L<sup>A</sup>T<sub>E</sub>X 來協助所有的排版、目錄建立以及書本修訂，希望能幫助讀者未來有機會使用到這個工具時能藉由這本書瞭解大部分常用的指令。

第二章與第三章開始則將利用正當風靡的程式語言 Python 來幫助讀者大致了解一些基本的函數圖形以及 Python 中的套件 Matplotlib 的基本使用方式，並介紹多種統計學的基礎機率分配以及這些分配的基本性質，希望能幫助讀者對分配有一些基本的概念。

第四章到第七章則將為讀者介紹七種常用的機器學習方法，機器學習主要有兩個常見的領域，一個是建立變數間的函數利用函數做選擇題，也就是分類問題，另一種則是預測問題，也就是建立函數來做計算題，本書將介紹基本的七種方法，包含簡單線性迴歸 ( Simple Linear Regression )、加廣迴歸 ( Augmented Regression )、羅吉斯迴歸

( Logistic Regression )、線性判別分析 ( Linear Discriminant Analysis )、二次判別分析 ( Quadratic Discriminant Analysis )、K-近鄰演算法 ( KNN ) 以及深度學習領域的類神經網路模型 ( ANN )，並利用統計模擬的方式一一建立模擬資料來檢視這些方法的訓練誤差與測試誤差，希望能幫助讀者對這些方法的理論以及實際應用有更深入的了解，並能在閱讀完本書後能降低一些對未知事物的恐懼甚至嘗試去喜歡它熱愛它，了解這個世界更多一些從來都沒有壞處只有好處，望共勉。

郭翊萱

2023 年 1 月於台北大學

# 第 1 章

## X<sub>E</sub>LATEX 基礎應用教學

在此章節中將淺度探討 LATEX 的一些基本應用，主要包含版面設置、數學式、表格、圖表與各種字形色彩之應用。在首節中，先闡述 LATEX 如何進行基本的版面設定，並在第二節說明各種數學式的展現，以及如何標記定義、定理與範例。接著，在第三節中會舉列幾項表格的各種排版方式，並進行各種色彩與粗細的變換。到了最後一節則將詳述如何插入各種不同類型的圖片檔，包含 jpeg、eps 與 png，及多張圖片的各種排版方式。

### 1.1 基本操作

在建立一個 LATEX 文件時，首要的一步便是載入各種 LATEX 套件並建立一個屬於自己的設定檔，以便將來之重複使用。為此，本章節將介紹一些基本的版面設定方式，以便持續更新與完善設定檔，並應對未來有可能產生的任何需求。本節將首先介紹一些基本頁面配置方法以及文字的相關設定，其中包含目錄設定、章節設定、各種文字字體與顏色等等。首先介紹一些基本頁面配置的方式。

#### 1.1.1 頁面配置

在此小節中將介紹幾個常用的版面設定方式，以下指令呈現了 LATEX 常用的其中兩種頁面配置方式。

```
\documentclass[options1]{class}
\geometry{options2}
```

其中

- (a) options1 指定了該文檔的版面設置，如若 options1 為 [11pt, twoside, a4paper]，即代表上述指令告訴 LATEX 本文檔長寬 A4 紙張相同，文字採雙邊編排（即奇偶頁左右

邊距正好相反)，基本字體大小是 11pt。

- (b) class 指定了將要被創建的文檔的類型，包含 article ( 短篇報告與期刊文章) , book ( 實體書籍) , report ( 長篇報告) , slides ( 課程大綱) , beamer ( ppt ) 等不同文檔類型。
- (c) options2 指定了該文檔上下左右的留白處大小。假設文檔設定 options2 為 [a4paper, left=2.5cm, right=2.5cm, top=2cm, bottom=2cm]，即代表 L<sup>A</sup>T<sub>E</sub>X 指定文檔左右空白處寬 2.5 公分，上下則留白 2 公分。

除了利用\geometry 進行設定之外也可直接在專屬的設定檔裡面直接利用此套件設定版面，設定方式如下，即直接載入 geometry 包並設定左右留白 3 公分且上下留白 2 公分。如有更多其他版面設定需求，可查閱 geometry 包的相關介紹說明。

```
\usepackage[left=3in,right=3in,top=2in,bottom=2in]{geometry}
```

## 頁眉頁腳設定

除了以上的紙張大小、留白等訊息，本章節也將在此介紹基本的頁眉頁腳設定方式。頁眉與頁腳通常位於文件的頂部與尾部，一般文字內容為作者、文章名稱等等。頁眉與頁腳的內容通常相似或互為補充。

```
\usepackage{fancyhdr}\pagestyle{fancy}
```

在利用以上指令載入 fancyhyd 包後，則接著加入對頁眉頁腳相關設定的指令，本章節列舉了七個 fancyhyd 包中常見的指令，其內容如下：

1. \fancyhf{} : 清除頁眉跟頁腳
2. \rhead{} : 在頁眉的右側顯示括號內文字
3. \lhead{} : 在頁眉的左側顯示括號內文字
4. \chead{} : 在頁眉的中間顯示括號內文字
5. \rfoot{} : 在頁腳的左側顯示括號內文字
6. \lfoot{} : 在頁腳的右側顯示括號內文字
7. \cfoot{} : 在頁腳的中間顯示括號內文字

除此之外，亦可改變頁眉頁腳裝飾線的寬度，其指令分別如下：

```
\renewcommand{\headrulewidth}{2pt}
\renewcommand{\footrulewidth}{2pt}
```

## 頁碼設置

在建立 L<sup>A</sup>T<sub>E</sub>X 時，可能也會希望頁碼的位置能夠進行改變，因此本章節亦提供一些基本的相關命令，以下即是幾個常用的頁碼設置方式。

```
\pagestyle{type}
\thispagestyle{type}
```

其中，`\pagestyle{type}` 是同時對所有該文件頁面進行設置，而`\thispagestyle{type}` 則是僅對當前所在頁面進行頁碼之相關設置。在 L<sup>A</sup>T<sub>E</sub>X 中，共默認提供了以下幾種頁面樣式：

- empty: 沒有頁眉跟頁腳。
- plain: 沒有頁眉，頁腳包含一個居中的頁碼。
- heading: 沒有頁腳，頁眉包含章節名字跟頁碼。
- myheadings: 沒有頁腳，頁眉包含頁碼。

另外，在 L<sup>A</sup>T<sub>E</sub>X 中亦可利用指令改變頁碼的顯示風格，其中之頁碼風格包含五類，指令與類別如下：

```
\pagenumbering{style}
```

- arabic: 阿拉伯數字
- roman: 小寫羅馬數字
- Roman: 大寫羅馬數字
- alph: 小寫字符形式
- Alph: 大寫字符形式

## 目錄設置

介紹完頁碼的設置指令後，接著介紹如何在 L<sub>A</sub>T<sub>E</sub>X 中為文件建立目錄，令 L<sub>A</sub>T<sub>E</sub>X 可以與任何章節或標題的變動進行連動，其基本指令如下：

```
\tableofcontents  
\newpage
```

其中\newpage 是當希望目錄占滿頁面時接在上面指令後面的指令。接著本章節將陸續介紹關於目錄設定的功能，包含目錄顏色、處理長標題，以及自動令文章中的圖片與表格也生成目錄。

## 目錄顏色

如果載入 hyperref 這個套件，就能令目錄產生顏色，可以更好的告訴顯示此紅色文字是可以進行點擊跳轉的。

```
\usepackage[colorlinks=true]{hyperref}
```

另外，如果在章節前加入星號，即\section\*{introduction}，即可讓此章節不會顯示在目錄中。

## 長標題處理

當遇到長標題可能超出目錄一行所能顯示的量時，L<sub>A</sub>T<sub>E</sub>X 可以讓文件在目錄顯示短標題而在內文中仍然顯示長標題，如以下指令，在目錄中僅會顯示 L<sub>A</sub>T<sub>E</sub>X 目錄製作，而不會顯示中括號中的長內文。

```
\section[\LaTeX 目錄製作]{目錄目錄目錄目錄目錄目錄目錄}
```

另外，如果希望能夠圖片與表格生成目錄，則加入指令如下：

```
\listoffigures  
\listoftables
```

### 1.1.2 文字設定

#### 文字字體與粗細

以下將直接展現幾種常用的英文與中文字體，以及加粗或斜體等功能。

## 英文字體

- 一般字體

Normal text in Times New Roman font

```
\textnormal{Normal text in Times New Roman font}
```

- 加粗字體

**Bold text in Times New Roman font**

```
\textbf{Bold text in Times New Roman font}
```

- 斜字體

*Italic text in Times New Roman font*

```
\textit{Italic text in Times New Roman font}
```

- 粗體加斜體字

***My name is Jason.***

```
\emph{\textbf{My name is Jason.}}
```

- 無襯線字體

Sans-serif text in Tahoma font

```
\textsf{Sans-serif text in Tahoma font}
```

- 等寬字體

Monospace text in Courier New font

```
\texttt{Monospace text in Courier New font}
```

## 中文字體

- 微軟正黑體

這是微軟正黑體。

```
{\MB 這是微軟正黑體。}
```

- 縮小版新細明體

這是縮小版新細明體

```
{\SM 這是縮小版新細明體}
```

- 標楷體

這是標楷體

```
{\K 這是標楷體 }
```

- 仿宋體

這是仿宋體

```
{\BB 這是仿宋體 }
```

## 文字顏色與大小

想要改變顏色的話有幾種不同的方法，本章節將介紹 L<sub>A</sub>T<sub>E</sub>X 兩種常用的改變文字顏色的方法，以及改變字體大小的指令。

### 字體顏色

```
\usepackage{color}
\textcolor{red/blue/green/black/white/cyan/yellow}{text}
\textcolor[rgb]{r,g,b}{Introduction to machine learning}
{\color[rgb]{r,g,b} Introduction to machine learning}
```

以上即為常見的變更字體顏色的方式，第一種是在 *textcolor* 中包含定義的顏色以供之後使用，第二種與第三種則是建立一個調色盤，其中 r, g, b 代表 red, green, blue 三原色的組合，取值範圍在 [0, 1] 之間，以下對字體顏色的變化進行展示。

Introduction to Machine Learning

Introduction to Machine Learning

Introduction to machine learning

Introduction to machine learning

### 字體大小

介紹完如何改變字體顏色後，亦接著展示如何改變字體的大小，如表 1.1 中介紹了幾種常用的改變字體大小指令<sup>1</sup>，另外，此章節亦在下方展示如何同時改變字體、字體顏色與字體大小。

---

<sup>1</sup>以下內容參考自 <https://blog.csdn.net/yhcwjh/article/details/116516011>

表 1.1: 最基本的表格

指令	對應字號
\tiny	5pt
\scriptsize	7pt
\footnotesize	8pt
\small	9pt
\normalsize	10pt
\large	12pt
\Large	14.4pt
\LARGE	17.28pt
\huge	20.74pt
\Huge	24.88pt

Introduction to machine learning

## 1.2 數學符號與方程式

為了將來能在 L<sup>A</sup>T<sub>E</sub>X 撰寫數學式上流暢無礙，本章節將在第一小節列舉一些常見的數學式作為參考，並在第二小節示範如何利用 L<sup>A</sup>T<sub>E</sub>X 妥善排版定義、定理以及範例題目。輸入數學式時，有兩個地方需要特別注意：

- 隨文數式前後請留一空格，才不會顯得擁擠。
- 展示數式上下不須多留一空行，L<sup>A</sup>T<sub>E</sub>X 會自行調整間距。

接著，列舉幾項常見的羅馬符號與數學運算記號。

### 1.2.1 常用數學符號

#### 常用羅馬符號

表 1.2: 羅馬符號

指令	對應字母	指令	對應字母
\alpha	$\alpha$	\upsilon	$\upsilon$
\beta	$\beta$	\Upsilon	$\Upsilon$
\gamma	$\gamma$	\Gamma	$\Gamma$
\delta	$\delta$	\Delta	$\Delta$
\epsilon	$\epsilon$	\xi	$\xi$
\varepsilon	$\varepsilon$	\Xi	$\Xi$
\zeta	$\zeta$	\pi	$\pi$
\eta	$\eta$	\Pi	$\Pi$
\theta	$\theta$	\rho	$\rho$
\vartheta	$\vartheta$	\sigma	$\sigma$
\iota	$\iota$	\Sigma	$\Sigma$
\kappa	$\kappa$	\tau	$\tau$
\lambda	$\lambda$	\Lambda	$\Lambda$
\mu	$\mu$	\phi	$\phi$
\nu	$\nu$	\Phi	$\Phi$

#### 常用數學運算符號

表 1.3: 數學運算符號

指令	對應字母	指令	對應字母
\pm	$\pm$	\mp	$\mp$
\ast	$*$	\star	$\star$
\cdot	$\cdot$	\cap	$\cap$
\oplus	$\oplus$	\ominus	$\ominus$
\sqcup	$\sqcup$	\wr	$\wr$
\vee	$\vee$	\diamond	$\diamond$
\triangleleft	$\triangleleft$	\triangleright	$\triangleright$

\dagger	†	\ddagger	‡
\times	×	\div	÷
\circ	◦	\bullet	•
\cup	∪	\amalg	II
\otimes	⊗	\sqcap	□
\wedge	∧	\setminus	\backslash
\bigtriangleup	△	\bigtriangledown	▽
\odot	⊙	\bigcirc	○

## 常用數學關係符號

表 1.4: 數學運算符號

指令	對應字母	指令	對應字母
\leq	≤	\geq	≥
\succeq	≥	\prec	≺
\gg	»	\ll	«
\subset	⊂	\subseteq	⊆
\supset	⊃	\supseteq	⊇
\ni	∋	\in	∈
\dashv	⊥	\vdash	⊤
\sim	~	\simeq	≈
\cong	≅	\neq	≠
\doteq	≈	\propto	∝
\perp	⊥	\models	⊧
\mid		\parallel	

## 其他常用數學符號

表 1.5: 數學運算符號

指令	對應字母	指令	對應字母
\leftarrow	←	\Longleftrightarrow	↔
\Leftarrow	⇐	\longmapsto	→

---

\rightarrow	→	\hookrightarrow	↪
\Rightarrow	⇒	\rightrightarpoonup	↗
\leftrightarrow	↔	\rightharpoondown	↘
\Leftrightarrow	↔	\mapsto	↪
\uparrow	↑	\hookleftarrow	↙
\Uparrow	↑↑	\simeq	≈
\leftharpoonup	←	\downarrow	↓
\leftharpoondown	←	\Downarrow	↓↓
\rightleftharpoons	⇒⇒	\updownarrow	↕
\longleftarrow	←←	\Updownarrow	↕↕
\Longleftarrow	←←←	\nearrow	↗↗
\longrightarrow	→→	\Updownarrow	↕↕
\searrow	↘	\Updownarrow	↕↕
\Longrightarrow	→→→	\swarrow	↙↙
\longleftrightarrow	↔↔	\nwarrow	↖↖

---

## 其他常用數學運算符號

表 1.6: 數學運算符號

指令	對應字母	指令	對應字母
\sum	Σ	\bigcap	∩
\bigodot	⊙	\prod	Π
\bigcup	∪	\bigotimes	⊗
\coprod	∏	\bigoplus	⊕
\int	∫	\bigvee	∨
\biguplus	⊕	\oint	∮
\bigwedge	∧		

以上介紹了幾種常用的改變字體大小指令，其中一些應用將在下面進行示例。

### 1.2.2 函數

- 機率分配函數

$$f(x) = \binom{n}{x} p^x (1-p)^{1-x}, \quad x = 0, 1, 2, \dots, n$$

$$f(x) = \frac{e^{-\lambda} \lambda^x}{x!}, \quad x = 0, 1, 2, \dots$$

$$f(x) = \frac{1}{\Gamma(\alpha)\beta^\alpha} x^{\alpha-1} e^{-\frac{x}{\beta}}, \quad x \geq 0$$

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}, \quad -\infty < x < \infty$$

- 積分函數

$$\int_0^\infty x^{\alpha-1} e^{-\lambda x} dx = \frac{\Gamma(\alpha)}{\lambda^\alpha} \quad (1.1)$$

方程式 (1.1) 是廣義  $\Gamma$  積分。

- 開立方根

$$f(x) = \sqrt[3]{\frac{4-x^3}{1+x^2}}$$

- 極限與微分

$$f'(x) = \frac{df(x)}{dx} = \lim_{h \rightarrow 0} \left( \frac{f(x+h) - f(x)}{h} \right)$$

### 1.2.3 聯立方程式

以下介紹幾種常用的聯立方程式。

1. 排列整齊的符號：

$$\begin{array}{lll} a + b + c & m + n & xy \\ a + b & p + n & yz \\ b + c & m - n & xz \end{array}$$

2. 等號對齊的函數組合（不編號）

$$b_1 = d_1 + c_1$$

$$a_2 = c_2 + e_2$$

3. 等號對齊的函數組合（編號在最後一行），如式 (1.2)

$$\begin{array}{ll} b_1 & = d_1 + c_1 \\ a_2 & = c_2 + e_2 \end{array} \quad (1.2)$$

4. 使用套件 amsmath 的指令 align (控制編號在第一行) , 如式 (1.3)

$$b_1 = d_1 + c_1 \quad (1.3)$$

$$a_2 = c_2 + e_2$$

5. 兩組數學式分別對齊且同時編號 , 如式 (1.4) 與式 (1.5)

$$\alpha_1 = \beta_1 + \gamma_1 + \delta_1, \quad a_1 = b_1 + c_1 \quad (1.4)$$

$$\alpha_2 = \beta_2 + \gamma_2 + \delta_2, \quad a_2 = b_2 + c_2 \quad (1.5)$$

6. 編號在中間 (split 指令環境) , 如式 (1.6)

$$\begin{aligned} \alpha_1 &= \beta_1 + \gamma_1 \\ \alpha_2 &= \beta_2 + \gamma_2 \end{aligned} \quad (1.6)$$

7. 只是居中對齊的數學式組 (環境指令 gather )

$$\alpha_1 + \beta_1$$

$$\alpha_2 + \beta_2 + \gamma_2$$

8. 長數學式的表達 (注意第二行加號的位置) , 如式 (1.7)

$$\begin{aligned} y &= x_1 + x_2 + x_3 \\ &\quad + x_4 + x_5 \end{aligned} \quad (1.7)$$

9. 聯立方程式 (不加編號)

$$\rho_k = \text{Corr}(Y_t, Y_{t-k}) = \frac{\gamma_k}{\gamma_0} = \begin{cases} 1 & \text{if } k=0 \\ \frac{\theta}{1+\theta^2} & \text{if } k=1 \\ 0 & \text{if } k \geq 2 \end{cases}$$

10. 聯立方程式 (加編號) , 如式 (1.8)

$$Y_t = \begin{cases} X_t & \text{for } t \text{ odd} \\ X_t + 3 & \text{for } t \text{ even} \end{cases} \quad (1.8)$$

11. 進階聯立方程式(加編號)，如式(1.9)

$$Cov(Y_t, Y_{t-k}) = \begin{cases} Cov(X_t, X_{t-k} + 3) & \text{if } t \text{ odd, } k \text{ odd} \\ Cov(X_t, X_{t-k}) & \text{if } t \text{ odd, } k \text{ even} \\ Cov(X_t + 3, X_{t-k}) & \text{if } t \text{ even, } k \text{ odd} \\ Cov(X_t + 3, X_{t-k} + 3) & \text{if } t \text{ even, } k \text{ even} \end{cases} = Cov(X_t, X_{t-k}) \quad (1.9)$$

12. 進階聯立方程式

$$E(Y_t) = \begin{cases} E(X_t) & \text{if } t \text{ odd} \\ E(X_t + 3) & \text{if } t \text{ even} \end{cases} = \begin{cases} \mu & \text{if } t \text{ odd} \\ \mu + 3 & \text{if } t \text{ even} \end{cases} \quad (1.10)$$

#### 1.2.4 矩陣

1. 左右方括號的使用及各直行的對齊方式：

$$A = \begin{bmatrix} a+b & mnop & xy \\ a+b & pn & yz \\ b+c & mp & xyz \end{bmatrix}$$

2. 左右圓括號的使用及各式點狀：

$$A = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{pmatrix}$$

3. 複雜矩陣：

$$\begin{aligned} M &= \begin{bmatrix} Cov(Y_1, Y_1) & Cov(Y_1, Y_2) & \dots & Cov(Y_1, Y_n) \\ Cov(Y_2, Y_1) & Cov(Y_2, Y_2) & \dots & Cov(Y_2, Y_n) \\ \vdots & \vdots & & \vdots \\ Cov(Y_n, Y_1) & Cov(Y_n, Y_2) & \dots & Cov(Y_n, Y_n) \end{bmatrix}_{n \times n} \\ &= \begin{bmatrix} \gamma_0 & \gamma_1 & \cdots & \gamma_n \\ \gamma_1 & \gamma_0 & \ddots & \vdots \\ \vdots & \ddots & \ddots & \gamma_1 \\ \gamma_{n-1} & \cdots & \gamma_1 & \gamma_0 \end{bmatrix}_{n \times n} \end{aligned}$$

## 4. 巨大矩陣

$$(I - N)^{-1} = \left( \begin{array}{ccccccccc} (0,0) & (1,0) & (2,0) & (3,0) & (4,0) & (5,0) & (6,0) & (7,0) & (8,0) & (9,0) & (10,0) & (11,0) \\ (0,0) & \left( \begin{array}{c} 1 \\ 0 \\ (1,0) \\ (2,0) \\ (3,0) \\ (4,0) \\ (5,0) \\ (6,0) \\ (7,0) \\ (8,0) \\ (9,0) \\ (10,0) \\ (11,0) \end{array} \right) & \left( \begin{array}{c} 11 \\ 11 \\ 11 \\ 11 \\ 11 \\ 11 \\ 11 \\ 11 \\ 11 \\ 11 \\ 11 \\ 11 \\ 11 \end{array} \right) & \left( \begin{array}{c} 55 \\ 55 \\ 165 \\ 165 \\ 330 \\ 330 \\ 462 \\ 462 \\ 330 \\ 330 \\ 165 \\ 165 \\ 330 \end{array} \right) & \left( \begin{array}{c} 165 \\ 165 \\ 363 \\ 363 \\ 363 \\ 363 \\ 2541 \\ 2541 \\ 363 \\ 363 \\ 165 \\ 165 \\ 330 \end{array} \right) & \left( \begin{array}{c} 462 \\ 462 \\ 5 \\ 5 \\ 5 \\ 5 \\ 2541 \\ 2541 \\ 5 \\ 5 \\ 165 \\ 165 \\ 330 \end{array} \right) & \left( \begin{array}{c} 462 \\ 462 \\ 7777 \\ 7777 \\ 7777 \\ 7777 \\ 15 \\ 15 \\ 3 \\ 3 \\ 1111 \\ 1111 \\ 330 \end{array} \right) & \left( \begin{array}{c} 462 \\ 462 \\ 7777 \\ 7777 \\ 7777 \\ 7777 \\ 1111 \\ 1111 \\ 1111 \\ 1111 \\ 1111 \\ 1111 \\ 330 \end{array} \right) & \left( \begin{array}{c} 55 \\ 55 \\ 1111 \\ 1111 \\ 1111 \\ 1111 \\ 1111 \\ 1111 \\ 1111 \\ 1111 \\ 1111 \\ 1111 \\ 330 \end{array} \right) & \left( \begin{array}{c} 11 \\ 11 \\ 11 \\ 11 \\ 11 \\ 11 \\ 11 \\ 11 \\ 11 \\ 11 \\ 11 \\ 11 \\ 11 \end{array} \right) \end{array} \right)$$

## 5. 分數裡的矩陣

$$\Pi = \begin{vmatrix} \frac{\partial R}{\partial X} & \frac{\partial \Phi}{\partial X} \\ \frac{\partial R}{\partial Y} & \frac{\partial \Phi}{\partial Y} \end{vmatrix}^{-1} = \frac{1}{\begin{vmatrix} \frac{\partial X}{\partial R} & \frac{\partial X}{\partial \Phi} \\ \frac{\partial Y}{\partial R} & \frac{\partial Y}{\partial \Phi} \end{vmatrix}} = \frac{1}{\begin{vmatrix} \sin(2\pi ft + 2\pi\Phi) & 2\pi R \cos(2\pi ft + 2\pi\Phi) \\ \cos(2\pi ft + 2\pi\Phi) & -2\pi R \sin(2\pi ft + 2\pi\Phi) \end{vmatrix}}$$

## 6. 加入實線的矩陣

$$\left( \begin{array}{c|c|c} a & b & c \\ \hline d & e & f \\ \hline g & h & i \end{array} \right)$$

### 1.2.5 範例

本章節將舉一些數學式於定義、定理與數學式的應用範例供以後使用時進行參考。

**Definition 1.2.1.** A nonnegative integer random variable  $X_n(\wedge)$  can be imbedded into a finite Markov chain if:

- (a) There exist a finite Markov chain  $\{Y_t : t \in \Gamma_n\}$  defined on a finite state space  $\Omega$  with initial probability distribution  $\delta$ .
- (b) There exists a finite partition  $\{C_x : x = 0, 1, \dots, l_n\}$  of the state  $\Omega$ .
- (c) For every  $x=0, 1, \dots, l_n$ , we have:

$$P(X_n(\wedge) = x) = P(Y_n \in C_x \mid \delta) \quad (1.11)$$

where  $\wedge$  is some pattern of interest in a problem.

**定理 1.2.1.** Given a transition probability matrix of a homogeneous Markov chain  $\{Y_t\}$  as the form in (1.11), the probability for the time index  $n$  when  $Y_t$  first enters the set of absorbing states can be obtained by:

$$P(W(\wedge) = n) = P(Y_n \in A, Y_{n-1} \notin A, \dots, Y_1 \notin A \mid \delta) = \delta N(\wedge)^{n-1} (I - N(\wedge)) \mathbf{1}' \quad (1.12)$$

where  $\delta = (\delta_1, \delta_2, \dots, \delta_{m-k} : 0_{1 \times k})$ .

**定理 1.2.2.** Suppose that  $M$  is the transition probability matrix, in the form of (1.12) of an imbedded Markov chain  $\{Y_t\}$  for the waiting-time random variable  $W(\wedge)$  of some pattern  $\wedge$ , then:

$$E[W(\wedge)] = \delta(I - N(\wedge))^{-1} \mathbf{1}' \quad (1.13)$$

We continued to use the Markov chain imbedding to solve the random walk problem.

**Example 1.** Suppose  $Y_t = \beta_0 + \beta_1 t + X_t$ , where  $\{X_t\}$  is a zero-mean stationary series with autocovariance function  $\gamma_k$  and  $\beta_0$  and  $\beta_1$  are constants.

- (a) Show that  $\{Y_t\}$  is not stationary but that  $W_t = \nabla Y_t = Y_t - Y_{t-1}$  is stationary.
- (b) In general, show that if  $Y_t = \mu_t + X_t$ , where  $\{X_t\}$  is a zero-mean stationary series and  $\mu_t$  is a polynomial in  $t$  of degree  $d$ , then  $\nabla^m Y_t = \nabla(\nabla^{m-1} Y_t)$  is stationary for  $m \geq d$  and nonstationary for  $0 \leq m < d$ .

**Answer:**  $Y_t = \beta_0 + \beta_1 t + X_t$ .  $\{X_t\}$  is stationary with zero mean and autocovariance function  $\gamma_k$ .

- (a) Since  $E(Y_t) = \beta_0 + \beta_1 t$  depends on  $t$ ,  $\{Y_t\}$  is nonstationary.

$$W_t = \nabla Y_t = Y_t - Y_{t-1} = \beta_0 + \beta_1 t + X_t - (\beta_0 + \beta_1(t-1) + X_{t-1}) = \beta_1 + (X_t - X_{t-1}).$$

$$E(W_t) = \beta_1 + E(X_t) - E(X_{t-1}) = \beta_1$$

$$\text{Cov}(W_t, W_{t-1}) = \text{Cov}(\beta_1 + X_t - X_{t-1}, \beta_1 + X_{t-1} - X_{t-2}) = 2\gamma_k - \gamma_{k+1} - \gamma_{k-1}$$

Since  $E(W_t)$  is a constant over time and  $Cov(W_t, W_{t-k})$  depends only on  $k$ ,  $\{W_t\}$  is stationary.

- (b) (i) When  $d=1$ ,  $Y_t = \beta_0 + \beta_1 t + X_t$ . By (a),  $\nabla^m Y_t$  is nonstationary for  $m=0$  and is stationary for  $m=1$ .  $\nabla^m Y_t$  is also stationary for  $m \geq 1$ .

$\therefore$  The statement holds when  $d=1$ .

- (ii) Assume that  $d=h$  holds, that is  $\nabla^m Y_t = \nabla^m(\mu_t + X_t) = \nabla^m \mu_t + \nabla^m X_t$  is nonstationary for  $0 \leq m < h$  and is stationary for  $m \geq h$  where  $\mu_t$  is polynomial in  $t$  of degree  $n$ .

It implies

$$\nabla^m \mu_t \text{ depends on } t \text{ if } 0 \leq m < h$$

and is a constant if  $m \geq h$

Now, we consider  $d = h + 1$ ,  $Y_t = \beta_0 + \dots + \beta_n t^h + \beta_{h+1} t^{h+1} + X_t = (\mu'_t X_t) + \beta_{n+1} t^{h+1}$ , where  $\mu'_t = \beta_0 + \dots + \beta_n t^n$  is polynomial in  $t$  of degree  $h$ .

Then,  $\nabla^m(\mu'_t + X_t)$  is nonstationary for  $m \geq h$

$$\nabla^n(\beta_{n+1} t^{h+1}) = \beta + h + 1[t^{h+1} - (t-1)^{h+1}] = \beta_{n+1} \nabla^{h-1} [t^h + t^{h-1}(t-1) + \dots + t(t-1)^{n-1} + (t-1)^n] \text{ depends on } t. \text{ (because of (1))}$$

For  $m \geq h$ ,

$$\nabla^{m+1}(\beta_{n+1} t^{n+1}) = \beta_{n+1} \nabla^m [t^h + t^{h+1}(t-1) + \dots + t(t-1)^{h-1} + (t-1)^h] \text{ is a constant.}$$

$\nabla^m Y_t$  is nonstationary for  $0 \leq m < h+1$ , and is stationary for  $m \geq h+1$

- (iii) By mathematical induction, the statement holds  $\forall d \in \mathbb{N}$ .

**Example 2.** Suppose that a stationary time series,  $\{Y_t\}$ , has an autocorrelation function of the form  $\rho_k = \phi^k$  for  $k > 0$ , where  $\phi$  is a constant in the range  $(-1, +1)$ .

- (a) Show that  $Var(\bar{Y}) = \frac{\gamma_0}{n} \left[ \frac{1+\phi}{1-\phi} - \frac{2\phi(1-\phi^n)}{n(1-\phi)^2} \right]$ .

$$\sum_{k=0}^n \phi^k = \frac{1-\phi^{n+1}}{1-\phi}, \text{ and the related sum } \sum_{k=0}^n k\phi^{k-1} = \frac{d}{d\phi} [\sum_{k=0}^n \phi^k].$$

- (b) If  $n$  is large, argue that  $Var(\bar{Y}) \approx \frac{\gamma_0}{n} \left[ \frac{1+\phi}{1-\phi} \right]$ .

- (c) Plot  $(1 + \phi)/(1 - \phi)$  for  $\phi$  over the range -1 to +1. Interpret the plot in terms of the precision in estimating the process mean.

**Answer:**

(a)

$$\begin{aligned} Var(\bar{Y}) &= \frac{\gamma_0}{n} [1 + 2 \sum_{k=1}^{n-1} \left(1 - \frac{k}{n}\right) \rho_k] = \frac{\gamma_0}{n} [1 + 2 \sum_{k=0}^{n-1} \left(1 - \frac{k}{n}\right) \phi^k - 2] \\ &= \frac{\gamma_0}{n} [-1 + 2 \sum_{k=0}^{n-1} \phi^k - \frac{2}{n} \sum_{k=0}^{n-1} k \phi^k] \end{aligned}$$

Consider  $\sum_{k=0}^{n-1} \phi^k = \frac{1-\phi^n}{1-\phi}$ . Then

$$\begin{aligned} \sum_{k=0}^{n-1} k \phi^k &= \phi \sum_{k=0}^{n-1} k \phi^{k-1} = \phi \sum_{k=0}^{n-1} \frac{d}{d\phi} [\phi^k] = \phi \frac{d}{d\phi} \left[ \sum_{k=0}^{n-1} \phi^k \right] = \phi \frac{d}{d\phi} \left[ \frac{1-\phi^n}{1-\phi} \right] \\ &= \phi \frac{(1-\phi)(-n\phi^{n-1}) - (1-\phi^n)(-1)}{(1-\phi)^2} \end{aligned}$$

$$\begin{aligned} Var(\bar{Y}) &= \frac{\gamma_0}{n} \left[ -1 + \frac{2(1-\phi^n)}{1-\phi} - \frac{2\phi}{n} \frac{(1-\phi)(-n\phi^{n-1}) + (1-\phi^n)}{(1-\phi)^2} \right] \\ &= \frac{\gamma_0}{n} \left[ \frac{-(1-\phi) + 2(1-\phi^n) + 2\phi^n}{1-\phi} - \frac{2\phi}{n} \frac{1-\phi^n}{(1-\phi)^2} \right] \\ &= \frac{\gamma_0}{n} \left[ \frac{1+\phi}{1-\phi} - \frac{2\phi}{n} \frac{1-\phi^n}{(1-\phi)^2} \right] \end{aligned}$$

(b) Assume  $A_n = \frac{\gamma_0}{n} \left[ \frac{1+\phi}{1-\phi} \right]$ ,  $B_n = -\frac{\gamma_0}{n^2} \frac{2\phi(1-\phi^n)}{(1-\phi)^2}$

$$\because \lim_{n \rightarrow \infty} \frac{B_n}{A_n} = \lim_{n \rightarrow \infty} \left[ -\frac{1}{n} \frac{2\phi(1-\phi^n)}{(1+\phi)(1-\phi)} \right] = 0 \therefore Var(\bar{Y}) = A_n + B_n \approx A_n = \frac{\gamma_0}{n} \left[ \frac{1+\phi}{1-\phi} \right]$$

## 1.3 表格

接著本章節將在此介紹一些 L<sub>A</sub>T<sub>E</sub>X 在建立表格方面的基本應用，包含表格置中、表格背景顏色、表格線條粗細、長表格處理等等內容，以供之後製作表格時之參照。

### 1.3.1 基本表格運用

```
\begin{table} [h]
\centering %將表格置中
\caption{} %表格標題
\extrarowheight=6pt %加高行高 6pt
\begin{tabular}{lcc} %取消欄間的直線
```

```
& & & \\
& & &
\end{tabular}
\end{table}
```

La<sub>T</sub>E<sub>X</sub> 可利用以上指令得到一個置中的表格，其結果示例如下表 1.7:

表 1.7: 範例表格

a	b	c
d	e	f
g	h	i

表 1.7 為 La<sub>T</sub>E<sub>X</sub> 建立表格的一般指令，本節將接著介紹幾種基本的表格樣式，包含表 1.8、表 1.9 與表 1.10。

### 1. 含橫線跟豎線的表格

表 1.8: 含橫線跟豎線的表格

(1,1)	(1,2)
(2,1)	(2,2)

### 2. 設置每一列的數值居左/居中/居右

表 1.9: 設置每一列的數值居左/居中/居右的表格

Leftleftleftleft	Stevesteve	Rightrightright
Matlab	Math	Maple

### 3. 三線表格

表 1.10: 三線表格

Leftleftleftleft	Stevesteve	Rightrightright
Matlab	Math	Maple

在介紹完幾種基本的表格，包含三線表格以及表格內文字的更改後，接著本章節將介紹其他較複雜的表格以因應往後撰寫文件或做模擬時可能出現大筆的資料需要顯示在文件中，其中包含改變表格的底色、還有夠複雜化的表格建立。

### 1.3.2 加入底色的表格

首先介紹改變表格顏色的方式，在建立表格時，可能會希望讓整張表的顏色都發生改變，或者是標題列或行具有顏色，並與資料列的顏色交錯顯示，以創造更好看的表格， $\text{\LaTeX}$  可透過載入以下套件達到此效果：

```
\usepackage{array}
\definecolor{slight}{gray}{0.9}
```

在載入 array 套件後，利用\definecolor 事先定義顏色以作為表格底色使用。如果是要將顏色應用在整張表的話  $\text{\LaTeX}$  會使用\colorbox 指令，如果僅應用在標題列的話則使用\rowcolor，其可產生的表如表 1.11。

表 1.11: 加入底色的表格

Leftleftleftleft	Stevesteve	Rightrightright
Matlab	Math	Maple
Cat	Dog	Tiger

表 1.11 即為改變全部底色的表格示例，其中亦設定第一欄偏左設置、第二欄置中設置且第三欄偏右設置。

### 1.3.3 表格並列

另外，除了單純的置中，表格同樣也有其他排版方式，如果有兩張表格的話，在排版時也可以選擇將兩張表格並排顯示，示例如下表 1.12。

表 1.12: 兩個表格並列

Leftleftleftleft	Stevesteve	Rightrightright	stephen
Matlab	Math	Maple	Leftleftleftleft
Cat	Dog	Tiger	jasonjason
			Matlab
			Math
			Cat
			Dog

### 1.3.4 文字表格並排

表格常需置入跨越多欄的一列，可以使用指令 `multicolumn`，配合指令 `cline` 畫出適當長度的橫線，如右表所示。這裡採用了 La<sub>T</sub>E<sub>X</sub> 常見的 `minipage` 語法，將頁面切成兩欄模式，左邊放文字，右邊放一張簡單的表格。強制將文字與表格綁在一起，不會讓 La<sub>T</sub>E<sub>X</sub> 將表格放在它認為適合的位置。

國家	經濟表現	
	央行	物價
	獨立性	上漲率
義大利	0.5	16.12
英國	2	12.3
加拿大	3	8.1
美國	1	19.21

### 1.3.5 複雜表格建立

表 1.13: 複雜表格

	A		B		C		D		E	
	$C = 0$		$C = 0$		$C = 0$		$C = 0$		$C = 0$	
	$C1$	$C2$								
A	0.01	0(0)	0.0091	0(0)	0	0(0)	0	1(0)	0	0(0)
A	0.01	0(0)	0.01	0(0)	0	0(0)	1	1(0)	0	0(0)
A	0.04	0(0)	0.02	0(0)	0	0(0)	0	1(0)	0	0(0)
A	0.0	0(0)	0.03	0(0)	0	0(0)	0	1(0)	0	0(0)
A	0.0	0(0)	0.04	0(0)	0	0(0)	0	1(0)	0	0(0)
A	0.0	0(0)	0.05	0(0)	0	0(0)	0	1(0)	0	0(0)
A	0.1	0(0)	0.6	0(0)	0	0(0)	0	1(0)	0	0(0)
A	0.0	0(0)	0.6	0(0)	0	0(0)	0	1(0)	0	0(0)
A	0.1	0(0)	0.6	0(0)	0	0(0)	0	1(0)	0	0(0)
A	0.2	0(0)	0.6	0(0)	0	0(0)	0	1(0)	0	0(0)

### 1.3.6 表格單元格合併

除了基本的表格創立方式，在建立表格時也常常會利用到合併欄、合併列，跨越多欄等功能，在此時可以使用指令 \multicolumn 來達成此一目的，如表 1.14。

表 1.14: 合併列的表格

學生資料		
姓名	學號	性別
Jack	001	Male
Jenny	005	Female

### 1.3.7 表格粗細

在 L<sub>A</sub>T<sub>E</sub>X 中同樣有套件\booktabs 可以使表格欄列的粗細發生變化，此套件同樣提供了欄寬的相關設定，令使用者在表格的製作上可以有更多靈活多樣的選擇。<sup>2</sup>

表 1.15: 表格粗細

姓名	學號	性別
Steve Jobs	001	Male
Bill Gates	002	Female
Jenny Kuo	007	Female

```
\toprule
姓名 & 學號 & 性別
\midrule
Steve Jobs & 001 & Male
Bill Gates & 002 & Female
Jenny Kuo & 007 & Female
\bottomrule
```

### 1.3.8 表格單元欄列合併

表 1.16: 合併欄列的表格

策略	事件					
	50	0	100	200	300	300
	100	100	0	100	200	200

```
\begin{table}
\centering
\extrarowheight=6pt
\caption{合併列的表格}\label{tb:combine_two}
\begin{tabular}{c|c|c|c|c|c|c}
\hline
& \multicolumn{6}{c}{事件} \\
\hline
策略 & 50 & 0 & 100 & 200 & 300 & 300 \\
\hline
& 100 & 100 & 0 & 100 & 200 & 200 \\
\hline
\end{tabular}
\end{table}
```

<sup>2</sup>資料參考自網路 <https://blog.csdn.net/JueChenYi/article/details/77116011>

### 1.3.9 寬度過寬表格

當遇到很長的表格而無法在正常版面設置下將其全部展示時會選擇將表格進行旋轉，以令表格可以被完整容納，示例如表 1.17<sup>3</sup>。

表 1.17: 寬度過寬表格

Source	Df	SS	MS	F value	Pr> F
model	2	543.6	271.8	16.08	0.0004
Error	12	202.8	16.9		
Total	14	746.4			

### 1.3.10 長度過長表格

在建立表格時，除了遇到過寬的表格外，也可能會遇到過長的表格而需要使表格跨越頁面，如表 1.18，此時會使用\longtable 套件來進行處理。

表 1.18: 長型表格

年度	勝率	平均得分	平均失分	平均籃板	平均助攻
1896	—	—	—	60.31	59.16
1896	—	—	—	60.31	59.16
1896	—	—	—	60.31	59.16
1896	—	—	—	60.31	59.16
1896	—	—	—	60.31	59.16
1896	—	—	—	60.31	59.16

續接下頁

<sup>3</sup>以上部分內容摘錄自汪群超老師 <https://ntpuccw.blog/supplements/xetex-tutorial/>

承接上頁

## 1.4 圖片

在數據分析領域撰寫文書時，常常會需要將程式跑好的圖片自動存入電腦，而人們同樣也期望這些存在電腦中的文件能同樣輕鬆地被插入 L<sub>A</sub>T<sub>E</sub>X 的文件當中，且不論圖片檔的類型是 JPEG、EPS 或者是 PNG，L<sub>A</sub>T<sub>E</sub>X 都能插入圖片而不會造成任何困擾。因此本節將介紹如何在 L<sub>A</sub>T<sub>E</sub>X 中插入各種類型的檔案，並將圖片適當地進行標號與排版。本節首先介紹檔案插入的方式，首先需為所有將插入文件的圖片設立一個統一的資料夾，並將插入目錄進行指定，這樣可以避免混亂並縮短指令所需長度，其程式碼如下。

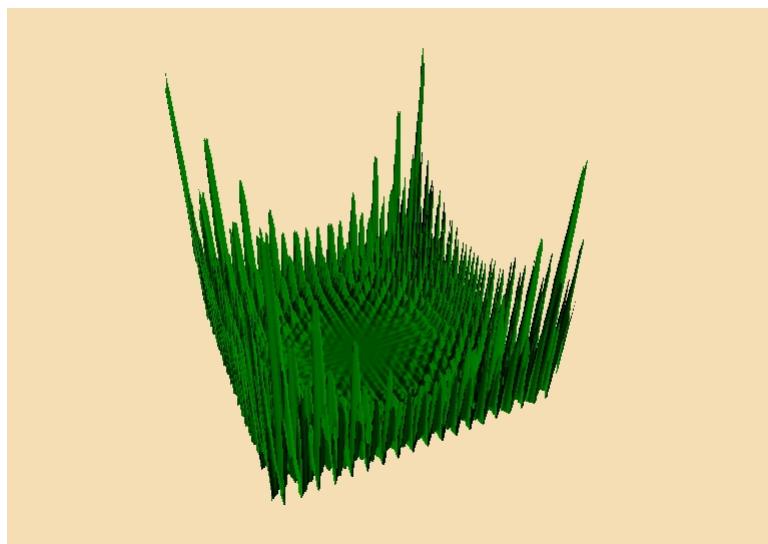


圖 1.1: 範例圖 (jpeg 圖)

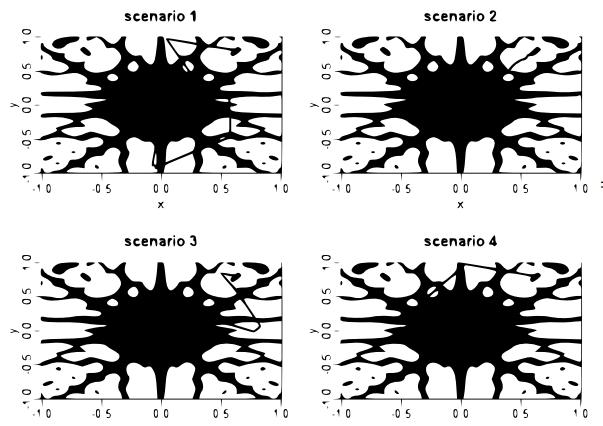
```
\begin{figure}[H]
\centering
\includegraphics[scale=0.6]{C:/Users/adfghkfr/OneDrive
    /文件/NTPU_STAT/images/Rplot64.jpeg}
\caption{範例圖}
\label{fig:scale}
\end{figure}
```

圖形的大小可以透過一些指令來進行變化，比如利用 scale 來使原圖縮小，或者利用 width 將原圖縮小成內文行寬的 a 倍，其調整依等比例進行縮放。在圖 1.1 中，本章節已經示範了在 L<sub>A</sub>T<sub>E</sub>X 中插入 JPEG 圖，接著在此示範如何在文件中插入 EPS 檔案與 PNG 檔案，並使兩個檔案並排顯示。其中此章節利用 \imgdir 定義一個與編譯文章路徑相同的子目錄，如圖 1.2。

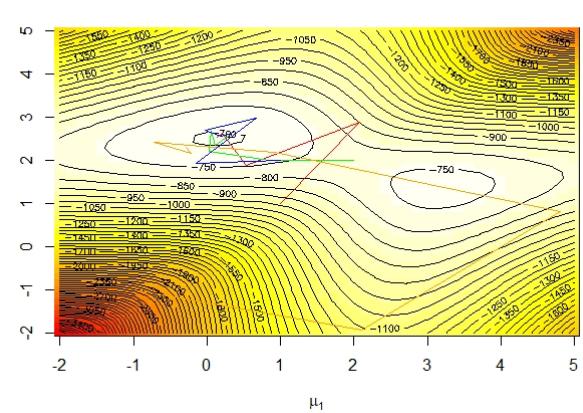
### 1.4.1 圖片並排

如圖 1.2，利用以下指令中的 `subfloat` 讓圖片在左右兩邊呈現，並可以分別為兩張圖片設置各自的子標題，同時也展現兩張圖片並排的母標題。

```
\subfloat[範例圖 (eps 圖)]{
\includegraphics[scale=0.35]{\imgdir Rplot65.eps}}
\subfloat[範例圖 (png 圖)]{
\includegraphics[scale=0.5]{\imgdir AnyConv.com_Rplot
68.png}}
```



(a) 範例圖 (eps 圖)

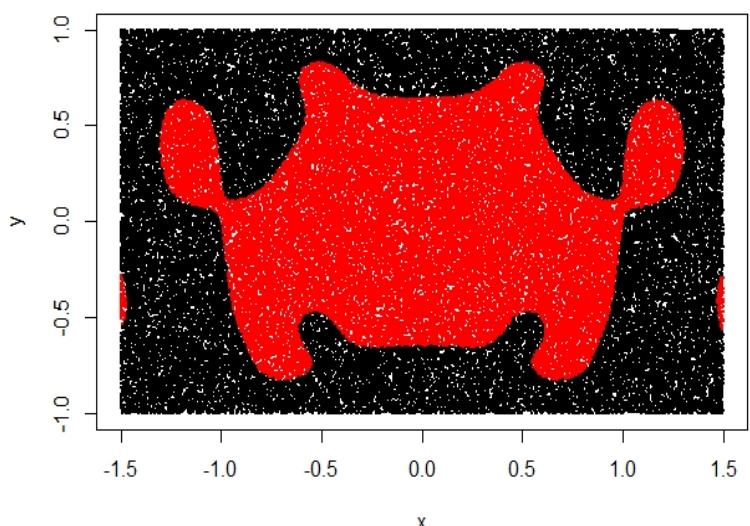


(b) 範例圖 (png 圖)

圖 1.2: 圖形並排的作法

### 1.4.2 文字與圖片並排

跟表格一樣，在 La<sub>T</sub>E<sub>X</sub> 中同樣可以使用 `\minipage` 使文字與圖片並排呈現。`minipage` 是另一種控制圖形位置的方式，適合必須與文字密切貼近的圖形。讓文字描述圖形時，可以用左圖或右圖，而不是依賴圖形編號。譬如，右圖示範的圖檔格為 PDF。此時，因為不能用 `\begin{figure}`，沒有 `label` 與 `caption`。



### 1.4.3 圖片旋轉

圖 1.3 展示了 L<sub>A</sub>T<sub>E</sub>X 如何將圖片插入文件中並進行旋轉，如下所示，利用指令 `includegraphics[]` 來對圖片進行插入並設定其中的 `angle` 來轉換圖片的角度。

```
\begin{figure}[H]
\centering
\includegraphics[angle=30, width=15cm, height=10cm]{imgdir Rplot56.jpeg}
\caption{旋轉圖片}
\label{fig:angle}
\end{figure}
```

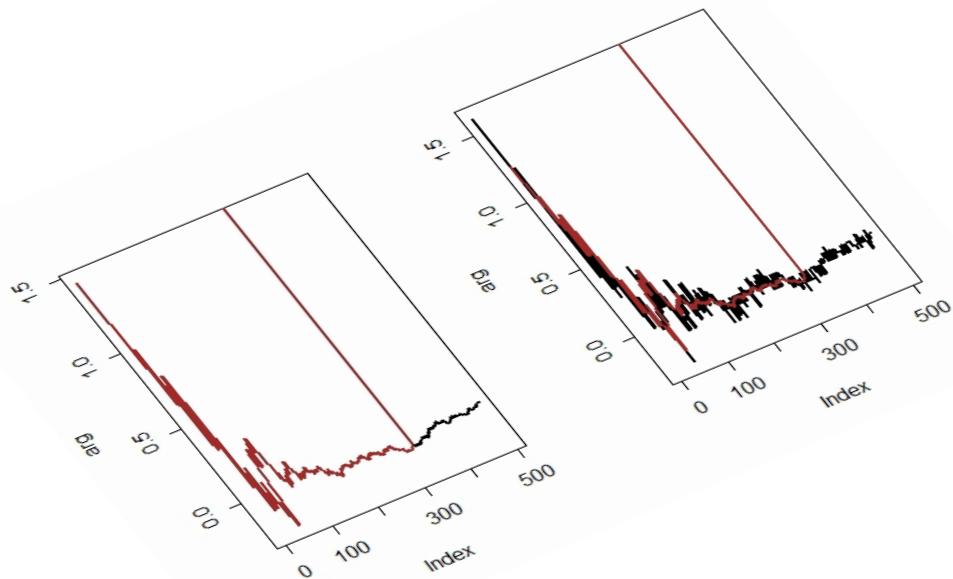


圖 1.3: 旋轉圖片

## 1.5 結論

在此章節中初步了解了 L<sub>A</sub>T<sub>E</sub>X 的基礎應用，包含設置頁面、字型字體、數學式、表格與插入圖片，其中頁面設置包含設置頁眉頁腳、目錄設置等基礎應用，以及如何對文字字體、字型與顏色進行變化，接著，此章節探討了利用 L<sub>A</sub>T<sub>E</sub>X 建立函數與矩陣的方式，且深入介紹了如何設定定義定理與範例並進行較佳的展示，並在最後介紹了各種表格建立方式以及圖片的插入與版面配置，期望將來遇到 L<sub>A</sub>T<sub>E</sub>X 相關問題時能在此章節中成功找到解答。



## 第 2 章

# Python 基本函數繪圖

Python 跟 R 是很常拿來作統計分析的程式語言，在 R 中常常使用 ggplot2 與 shiny 來繪製統計相關圖形，而在 Python 中則廣為使用 Matplotlib、Plotly 與 seaborn 來進行繪圖，故在此章節中，嘗試使用 Matplotlib 來進行一些基本函數圖形繪製，以展示套件 Matplotlib 的一些常見繪圖呈現方式。最後，本章節亦將進行一個小專題對繪圖方式進行練習。

## 2.1 基本機率函數繪圖

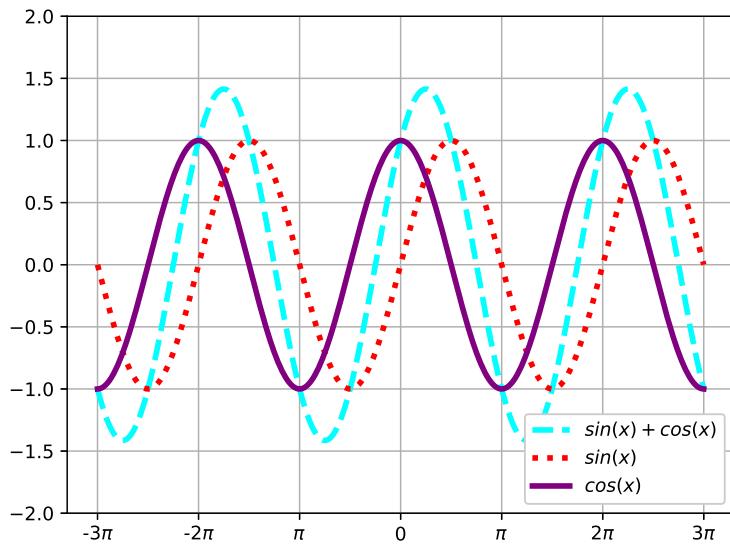
本章節在此介紹幾個基本函數圖形的繪製，以供之後利用套件 Matplotlib 進行繪圖之用，首先介紹函數  $f(x) = \sin(x) + \cos(x)$  的圖形繪製，如圖 2.1，該圖呈現了該函數與函數  $f(x) = \sin(x)$  與  $f(x) = \cos(x)$  的關係，其部分程式碼亦呈現如下。

- $f(x) = \sin(x) + \cos(x)$

在圖 2.1 中，展示了三種常用的圖形線條表現方式，其中  $\cos(x)$  用基本的實線表示、 $\sin(x)$  則透過設置 `linestyle = "--"` 來呈現虛線，設置 `linestyle = ":"` 則展現如  $\sin(x) + \cos(x)$  的點狀線條。

```
ax.plot(x, y, color='cyan', linestyle = "--", linewidth = 3, label = "$\sin(x)+\cos(x)$")
ax.plot(x, y_1, color='red', linestyle = ":", linewidth = 3, label = "$\sin(x)$")
ax.plot(x, y_2, color='purple', linestyle = "-", linewidth = 3, label = "$\cos(x)$")

ax.set_xticks(np.array([-3, -2, -1, 0, 1, 2, 3])*np.pi)
ax.set_xticklabels(['-3$\pi$', '-2$\pi$', '$\pi$', '0', '$\pi$', '2$\pi$', '3$\pi'], \
    fontsize=10, color = 'black')
```

圖 2.1:  $f(x) = \sin(x) + \cos(x)$ 

- $f(x) = \frac{1-e^{-2x}}{1+e^{-2x}}$

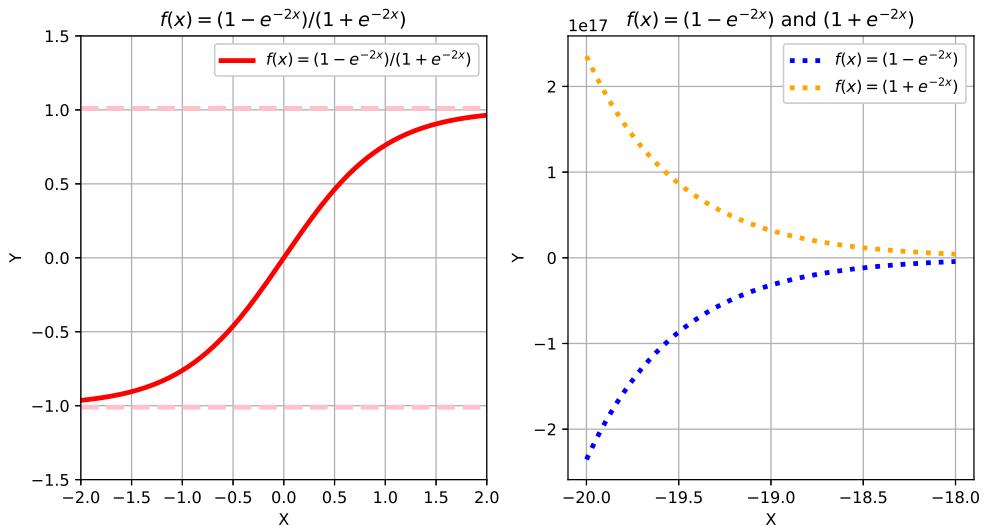
圖 2.2:  $f(x) = (1 - e^{-2x}) / (1 + e^{-2x})$ 

圖 2.2 中的左圖展現了函數  $f(x) = (1 - e^{-2x}) / (1 + e^{-2x})$  的函數圖形，右圖則繪製了函數  $f(x) = (1 - e^{-2x})$  與  $f(x) = (1 + e^{-2x})$  的函數圖形與其進行參照，從該圖可知，此函數在  $x = 0$  對稱且會漸進於  $y = 1$  與  $y = -1$ ，而  $f(x) = (1 - e^{-2x})$  與  $f(x) = (1 + e^{-2x})$  則會對稱且趨近於  $y = 0$ 。

```

x = np.linspace(-10, 11, 100)
f = lambda x : (1 - np.exp(-2*x)) / (1 + np.exp(-2*x))
plt.plot(x, f(x), color = 'r', linewidth = 3)
plt.axhline(1.01, color="b",linestyle="--")
plt.axhline(-1.01, color="b",linestyle="--")

```

- $f(x) = \sqrt[3]{(4 - x^3)/(1 + x^2)}$

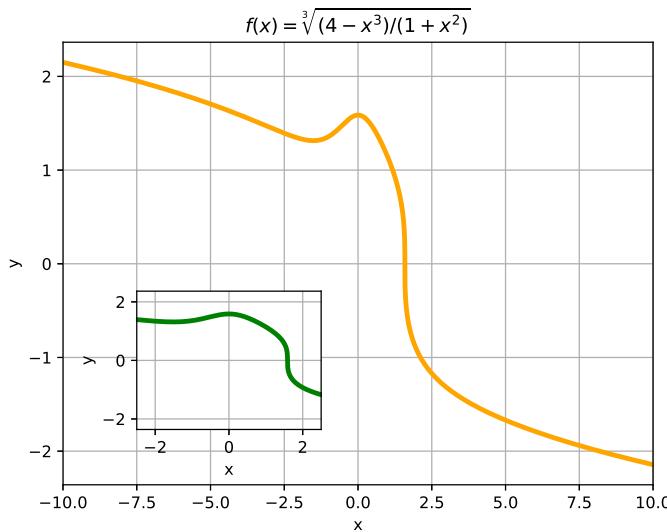


圖 2.3:  $f(x) = \sqrt[3]{(4 - x^3)/(1 + x^2)}$

圖 2.3 中展現了函數  $f(x) = \sqrt[3]{(4 - x^3)/(1 + x^2)}$  的函數圖形，其中利用了 Matplotlib 套件中的 `fig.add_axes()` 指令來建立兩個視窗進行繪圖並設置其 `left`, `bottom`, `width`, `height` 來指定其視窗位置，這四個值分別代表了該視窗佔整個坐標系左側、右側、上側與下側的百分比，其程式碼呈現如下。

```

fig = plt.figure()
left, bottom, width, height = 0.1, 0.1, 0.8, 0.8 #佔整個坐
標系的百分比
ax1 = fig.add_axes([left, bottom, width, height])
ax1.plot(x, f(x), "orange", linewidth = 3)

ax2 = fig.add_axes([0.2, 0.2, 0.25, 0.25])
ax2.plot(x, f(x), "green", linewidth= 3)

```

- $f(x) = 1/x$

圖 2.4 中展示了函數  $f(x) = 1/x$  的函數圖形，根據以下程式碼可知，由於此函數在  $x = 0$  時值不存在，故將函數分成大於 0 與小於 0 來繪製函數圖形，且由圖 2.4 可知，此函數會趨近於  $x = 0$  與  $y = 0$ ，且會對稱於函數  $y = -x$ 。

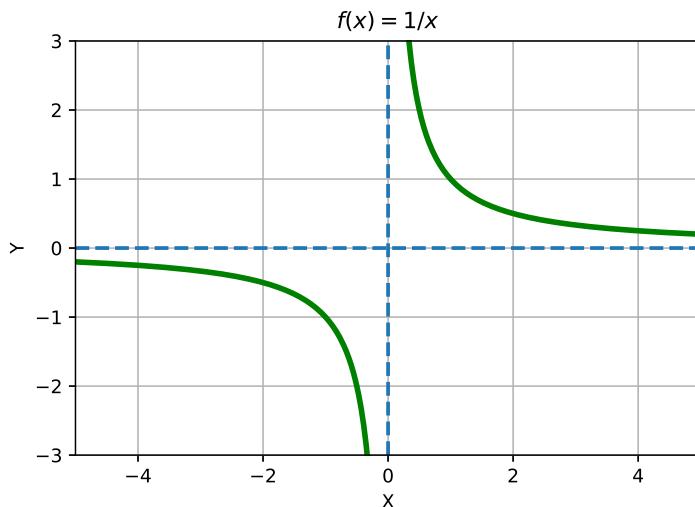


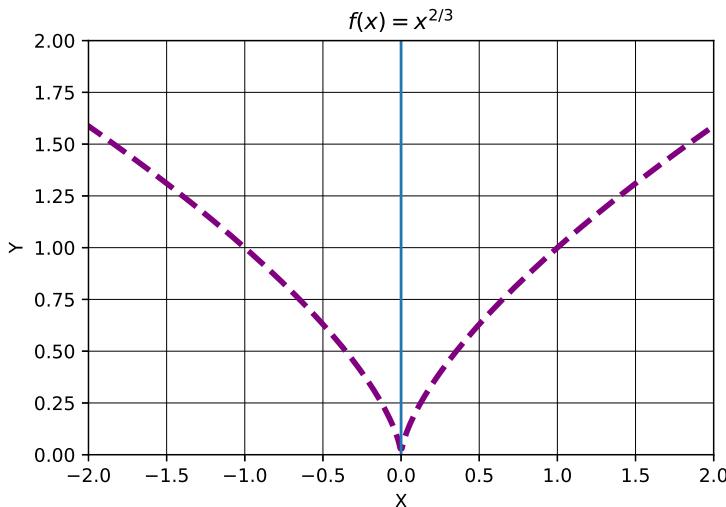
圖 2.4:  $f(x) = 1/x$

```
x = np.linspace(0.01, 10, 500) #type(x):array
# formulate a function f
f = lambda x : 1/x
fig = plt.figure(figsize=[6, 4])
plt.plot(x, f(x), color = 'green', linewidth = 3)
x = np.linspace(-10, -0.01, 500) #type(x):array
plt.plot(x, f(x), color = 'green', linewidth = 3)
plt.axhline(0, linestyle="--", linewidth = 2)
plt.axvline(0, linestyle="--", linewidth = 2)
```

- $f(x) = x^{2/3}$

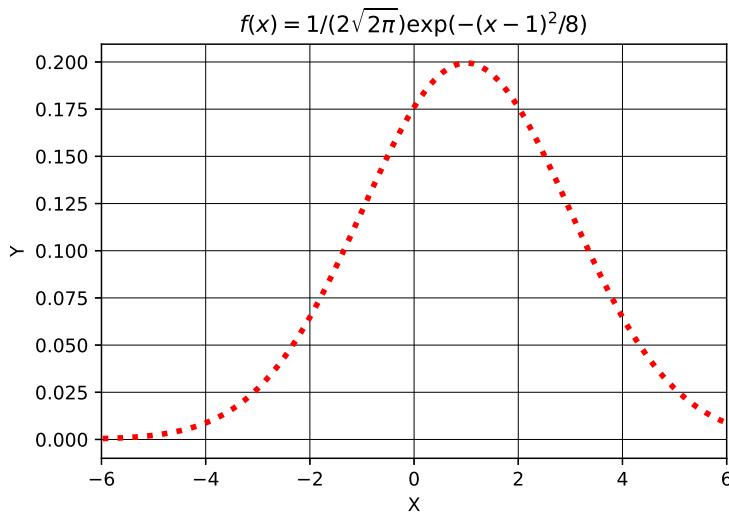
圖 2.5 為函數  $f(x) = x^{2/3}$  的函數圖形，由圖可知，此函數圖形會對稱於  $y = 0$  的函數，且其端點位在點  $(x, y) = (0, 0)$ ，另外，在進行繪圖時，會利用 Python 中的函數 `np.cbrt()` 計算該函數中的立方根，其程式碼呈現如下：

```
x = np.linspace(-5, 5, 1000)
f = lambda x : np.cbrt(x**2)
```

圖 2.5:  $f(x) = x^{2/3}$ 

- $f(x) = \frac{1}{2\sqrt{2\pi}}e^{-(x-1)^2/8}$

圖 2.6 即為函數  $f(x) = \frac{1}{2\sqrt{2\pi}}e^{-(x-1)^2/8}$ ，根據此函數型式可知，此函數是一個期望值為 1、變異數為 4 的常態分配。

圖 2.6:  $f(x) = 1/(2\sqrt{2\pi}) \exp(-(x-1)^2/8)$ 

- $f(x) = \ln(x)/x^3$  與  $f(x) = 3, 1 \leq x \leq 5$

圖 2.7 中之左圖是函數  $f(x) = \ln(x)/x^3$  的函數圖形，右圖則是函數  $f(x) = 3$  的函數圖形，此圖展現了各種繪圖表現技巧，包含更改座標軸刻度顏色、更改單點座標軸刻度顏色、更改畫布底色等等技巧，其主要程式碼呈現如下表 ??。

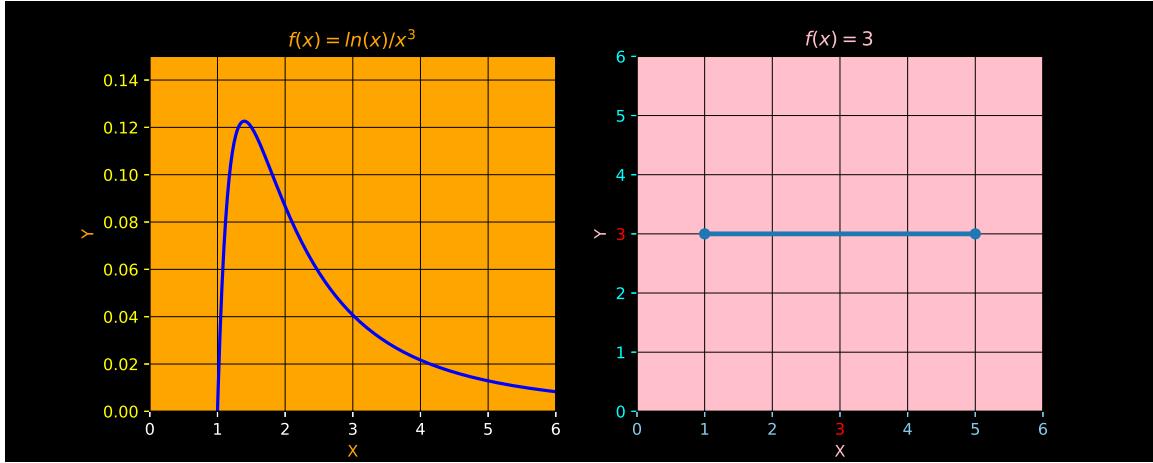
圖 2.7:  $f(x) = \ln(x)/x^3$ 

表 2.1: 套件 matplotlib 幾種繪圖函數

繪圖函數	繪圖作用
<code>fig.patch.set_facecolor("black")</code>	更改繪圖背景
<code>ax.set_facecolor("orange")</code>	更改各圖像顏色
<code>ax.tick_params(axis = "x", colors = "white")</code>	更改 x 軸座標顏色
<code>ax.tick_params(axis = "y", colors = "white")</code>	更改 y 軸座標顏色
<code>ax.get_xticklabels()[3].set_color("red")</code>	更改某 x 座標刻度顏色
<code>ax.get_yticklabels()[5].set_color("red")</code>	更改某 y 座標刻度顏色

以下程式碼則展現了如何繪製函數  $f(x) = 3, 1 \leq x \leq 5$ ，其中在函數 `plt.hlines()` 中設置了該函數的  $x$  值範圍，另外，為了特別標記欲標記的點座標，設置兩個列表 (list) 放置  $(x, y)$  的值，最後再利用函數 `plt.plot()` 標記出該點座標。

```
x=[1,5]
y=[3,3]
fig, ax = plt.subplots()
ax.hlines(y=3,xmin=1,xmax=5,linewidth=2)
plt.plot(x,y,marker="o")
```

- $f(x) = 2x^3 - x^4$

如圖 2.8，其中該圖利用小視窗與大視窗呈現該函數在  $x$  介於  $(-10, 10)$  之間的函數圖形以及在  $x$  範圍落在  $(-2, 2)$  的函數圖形。如果只看  $(-10, 10)$  之間的圖形的話，可能會誤以為此函數是一個 U 型函數且是一個趨近於 0 的函數，然而將圖形放大的話會發現此函數的最大值其實落在  $(1.49, 1.69)$  的點，計算最大值的程式碼

呈現如下，此程式先計算函數  $f(x)$  的最大值，故得到 1.69 這個值，並建立反函數求得該點的  $x$  值，即得到產生最大值的  $x$  為 1.49。

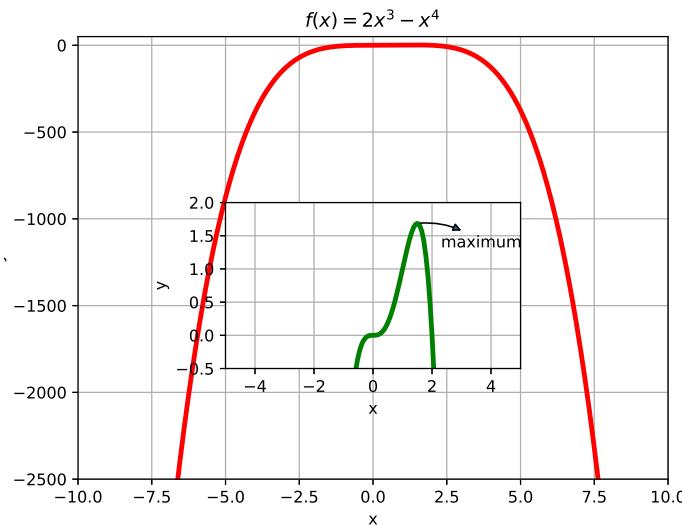


圖 2.8:  $f(x) = 2x^3 - x^4$

```
x = np.linspace(-20, 50, 1000)
f = lambda x : 2*x**3-x**4
Max = max(f(x))
inv = inversefunc(f, y_values = Max)
```

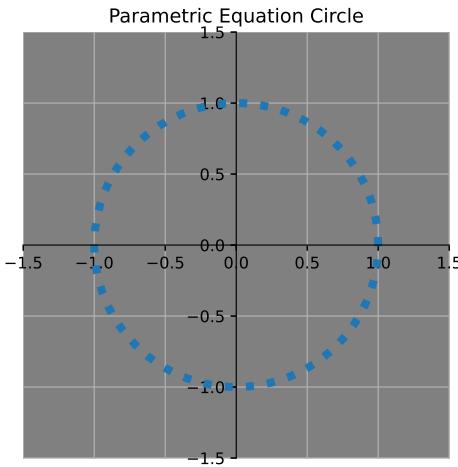
圖 2.8 亦展現如何在圖中指定一個點進行標記，其程式碼如下呈現，利用函數 `plt.annotate()` 指定標記的文字內容、標記的點以及標記箭頭樣式。

```
ax2.annotate('maximum', xy=(inv, Max), xytext=(15, -15),
             textcoords='offset pixels',
             arrowprops=dict(arrowstyle='|->', connectionstyle="arc3",
                             rad=.2))
```

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

圖 2.9 中展示了對半徑為 1 中心點座標為  $(x, y) = (0, 0)$  的圓形函數繪圖的結果，此圖透過移動座標軸的位置呈現出了與上面的繪圖不同的效果，其部分重要程式碼呈現如下，利用函數 `spines` 中的 `set_color("none")` 讓原本位於右邊與上面的邊線不要顯示，並利用 `set_position` 移動 x 軸與 y 軸的位置，將 x 軸綁定在  $y = 0$  的位置上且將 y 軸綁定在 x 軸 50 % 的位置上，即可得到圖 2.9 的圖形。

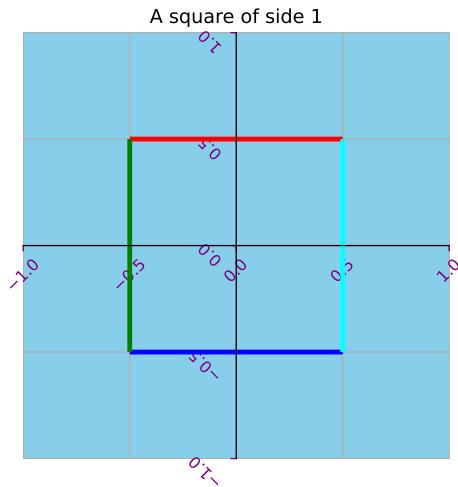
```
ax.spines["right"].set_color("none")
ax.spines["top"].set_color("none")
```

圖 2.9:  $f(x) = 3$ 

```
ax.spines["bottom"].set_position(("data", 0))
ax.spines["left"].set_position(("axes", 0.5))
ax.set_aspect("equal")
```

- Square

圖 2.10 則展示了繪製正方形的方法，其移動座標中的方式與圖 2.9 相同，其函數則透過分別繪製四條水平線或垂直線再給定繪圖範圍來進行繪製，另外，此圖中亦透過函數 `plt.xticks(rotation = 45)` 與 `plt.yticks(rotation = 135)` 來對座標軸刻度進行轉向。

圖 2.10: *Square*

```
ax.hlines(y=0.5, xmin=-0.5, xmax=0.5, linewidth=2)
ax.hlines(y=-0.5, xmin=-0.5, xmax=0.5, linewidth=2)
ax.vlines(x=0.5, ymin=-0.5, ymax=0.5, linewidth=2)
ax.vlines(x=-0.5, ymin=-0.5, ymax=0.5, linewidth=2)
```

## 2.2 專題練習

接著本章節列舉三個小專題來展示套件的應用。

Let  $S_n = \sum_{k=1}^n \frac{1}{k} = 1 + \frac{1}{2} + \frac{1}{3+...+\frac{1}{n}}$

- (a) Verify that  $\lim_{n \rightarrow \infty} S_n$  diverges.
- (b) Let  $\gamma_n$  denote the sum of the shade areas. Show that  $\gamma_n - \ln(n+1)$ .
- (c) Verify that  $\frac{1}{2}(1 - \frac{1}{n+1}) < \gamma_n < 1$ .

**Answer:**

- (a) 圖 2.11 證明了  $S_n$  發散的結果，其程式碼呈現如下，圖 2.11 建立了空集合  $S_n$  儲存不同的  $n$  所能得到的值，並嘗試將  $n$  值設為  $10^1$ 、 $10^2$  到  $10^9$  並建立迴圈來進行計算，最後針對  $S_n$  的值來進行繪圖，即可得到該圖之結果。

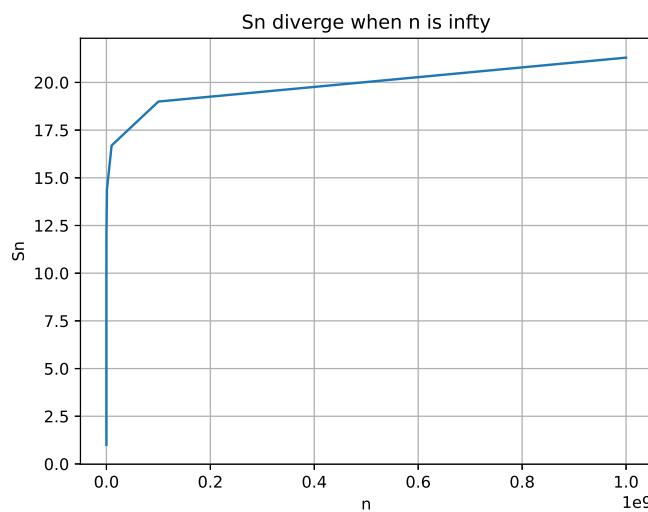


圖 2.11:  $S_n$  diverge when  $n \rightarrow \infty$

```
n = 10**np.arange(10)
Sn=np.zeros(len(n)) #挖空集合放不同的n得到的結果
for i in np.arange(len(n)):
```

```

a = np.arange(1,n[i]+1)
Sn[i] = (1/a).sum()
x = n
y = Sn
values = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
plt.plot(n,y)

```

- (b) 根據圖 2.12 以及以下程式碼可以求得從  $n = 1$  到  $n = 100$  時， $\gamma_n$  與  $S_n + \ln(n+1)$  個別的值，由此可以發現兩者的值基本趨近相同，故可得證。

另外，也可直接利用數學進行證明，其過程如下：

$$\begin{aligned}
\gamma_n &= \sum_{k=1}^n \int_{x=k}^{k+1} \left(\frac{1}{k} - \frac{1}{x}\right) dx = \sum_{k=1}^n \left( \int_k^{k+1} \frac{1}{k} dx - \int_k^{k+1} \frac{1}{x} dx \right) = \sum_{k=1}^n \left( \frac{1}{k} - \ln|k+1| + \ln|k| \right) \\
&= S_n - (\ln 2 + \ln 3 + \dots + \ln(n) + \ln(n+1)) + (\ln 1 + \ln 2 + \ln 3 + \dots + \ln(n)) \\
&= S_n - \ln(n+1)
\end{aligned}$$

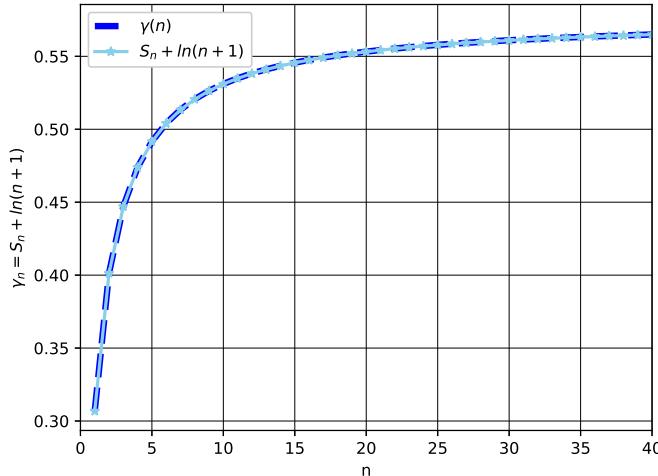


圖 2.12:  $\gamma_n$  and  $S_n + \ln(n+1)$

```

x = np.arange(0,1000,0.1)
f = lambda x : 1/x

def func(x):
    return 1/x
x = np.linspace(2,3,1000)
dx = (3-2)/1000
y = func(x)
area = np.sum(y*dx)

```

```

n = np.arange(1,101)
gamma2=np.zeros(len(n))
def func(x):
    return 1/x
for i in np.arange(len(n)):
    x = np.linspace(n[i],n[i]+1,1000)
    dx = (2-1)/1000
    y = func(x)
    area = np.sum(y*dx)
    gamma2[i] = func(i+1)-area+gamma2[i-1]
print(gamma2)

n = np.arange(1,101)
Sn = np.zeros(len(n))
gamma1 = np.zeros(len(n))
for i in np.arange(len(n)):
    a = np.arange(1,n[i]+1)
    Sn[i] = (1/a).sum()
    gamma1[i] = Sn[i]-np.log(n[i]+1)

```

- (c) 由上一小題的程式碼即可以繪製三條函數的函數圖形，並可得到圖 2.13 之結果，根據該圖即可得證  $\frac{1}{2}(1 - \frac{1}{n+1}) < \gamma_n < 1$ 。

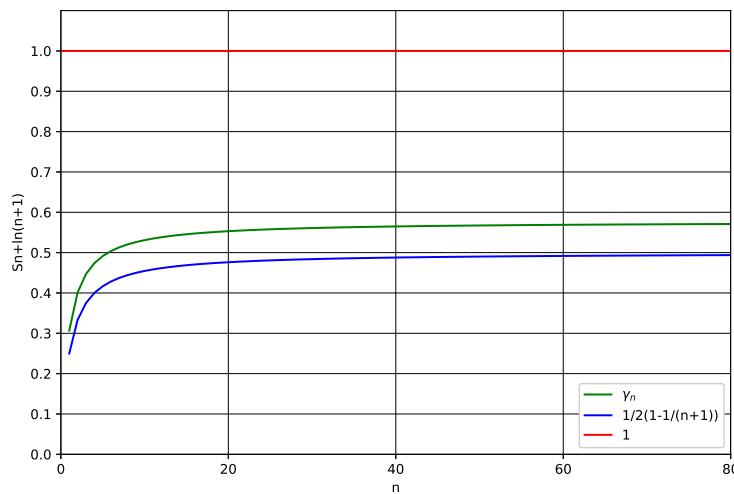


圖 2.13:  $\frac{1}{2}(1 - \frac{1}{n+1}) < \gamma_n < 1$

## 2.3 結論

在此章節中闡述了幾種函數圖形的展現，對幾種常見的函數的基本函數特性有了一些了解，另外，本章節亦透過繪製函數圖形的過程展示了許多常見的 Matplotlib 套件的繪圖技巧，期望以後在利用 Matplotlib 進行繪圖時能在此章節中迅速找到其基本繪圖技巧。

## 第 3 章

### 常見機率分配與抽樣分配

統計學的基礎是機率，而機率構成了分配，因此對於各種分配的掌握是學習統計的重要部分，了解每個常用分配的基本特性才能更好的理解統計學，並做出正確的分析。因此在本章中，首先將討論在改變參數時，一些常見的離散型分配與連續型分配，例如二項分配 ( Binomial Distribution )、超幾何分配 ( Hypergeometric Distribution )、幾何分配 ( Geometric Distribution ) 與卡方分配 ( Chi-squared Distribution ) 等等的機率密度函數圖 ( pdf ) 與累積機率函數圖 ( cdf )，並利用箱型圖、qqplot 圖與累績機率分配圖來了解其中的卜瓦松分配 ( Poisson Distribution ) 與伽瑪分配 ( Gamma Distribution ) 的加成性，另外，此節亦隨機抽取服從卜瓦松分配、卡方分配與伽瑪分配的隨機變數來進一步探討改變抽取樣本數大小與參數時分配的機率直方圖變化，最後以一個抽樣小專題來進行本章的收尾。

## 3.1 離散型分配

在此節中將介紹幾種常見的離散型分配，包含二項分配 ( Binomial Distribution )、超幾何分配 ( Hypergeometric Distribution )、幾何分配 ( Geometric Distribution ) 與卜瓦松分配 ( Poisson Distribution ) 的基本性質，並繪製其機率密度函數圖 ( pdf ) 與累積機率函數圖 ( cdf )。首先先介紹二項分配 ( Binomial Distribution )。

### 3.1.1 二項分配

二項分配 ( Binomial Distribution ) 是  $n$  個獨立的試驗中成功次數的離散機率分布，每次成功的機率都設為  $p$ ，而一次成功的分配則稱為伯努利分配 ( Bernouli Distribution )，其設置不同參數所得到的機率密度函數圖與累積機率函數圖呈現於下圖 3.1。

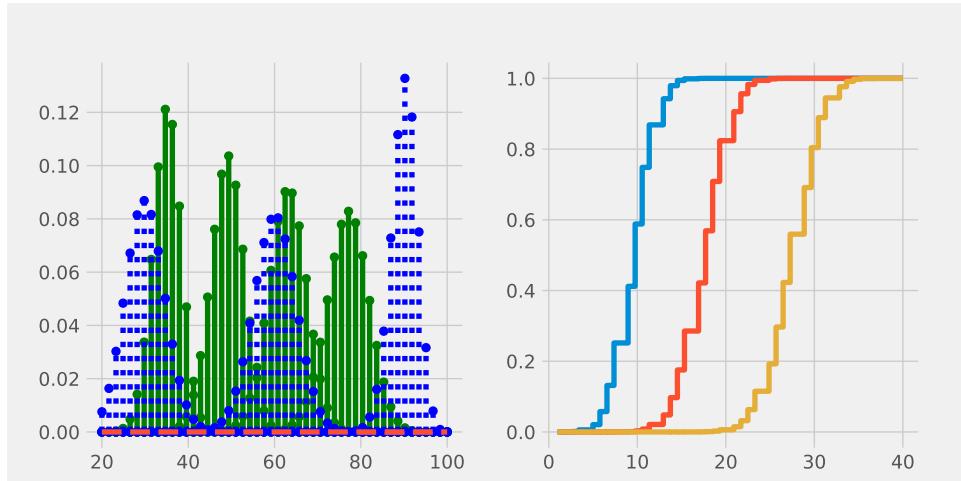


圖 3.1: 二項分配的機率質量函數圖與累積機率函數圖

圖 3.1 中左圖的綠色 stem 圖是固定參數  $p$  並設置  $n = 50, 70, 90, 110$  所得到的機率質量函數圖，藍色 stem 圖則是固定樣本數  $n$  並設置  $p = 0.3, 0.6, 0.9$  所繪製成的機率質量函數圖，其程式碼呈現如下，其中 `linefmt` 可以改變 stem 圖的線條型狀以及線條顏色，`markerfmt` 則可以改變端點的形狀。由綠色的機率質量函數圖可知，隨著  $n$  值變大，整個圖型會往右偏移，且會逐漸趨近於鐘型分配，圖形似乎有趨近常態分配的傾向。接著觀察藍色的機率質量函數圖同樣可以發現，隨著  $p$  增大，圖型也會逐漸向右移動，即期望值變大，且圖形的最高點也從 0.08 升至 0.12 左右。

```

n = np.arange(50, 120, 20)
p = 0.7
x = np.linspace(20, 100, 50)
for i in n:
    y = binom.pmf(x, i, p)
    axes[0].stem(x, y, linefmt='g-', markerfmt='o', basefmt =
                  'C1--')
n = 100
p = np.arange(0.3, 0.9, 0.3)
for i in p:
    y = binom.pmf(x, n, i)
    axes[0].stem(x, y, linefmt='b:', markerfmt='o', basefmt =
                  'C1--')

```

圖 3.1 中的右圖則是繪製三組不同  $(n, p)$  情況下的累積機率函數圖，包含  $(20, 0.5)$ ,  $(30, 0.6)$  以及  $(40, 0.7)$ ，從該圖可以看出，隨著  $n$  與  $p$  的值增加，cdf 圖會整體向右移動。

## 二項分配趨近常態分配

根據圖 3.1 可以觀察到二項分配似乎有趨近常態分配的性質，因此在此以二項分配  $B(100, 0.1)$  與常態分配  $N(10, 3)$  來嘗試進行驗證，其驗證結果如下，由圖 3.2 可知，二項分配在樣本數夠大且  $np > 5$  時會趨近常態分配。

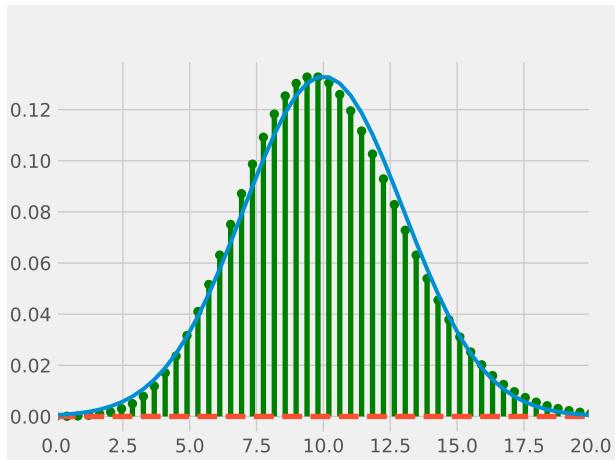


圖 3.2: 二項分配與常態分配的機率函數圖

### 3.1.2 超幾何分配

接著對超幾何分配 ( Hypergeometric Distribution ) 進行介紹。超幾何分配是統計學上的一種離散型分配，代表從有限的  $n$  個物件中抽取  $N$  個物件，並成功從其中指定的  $N$  個物件中抽出  $x$  個物件的機率 ( 取出不放回 )，其機率質量函數圖呈現如圖 3.3。

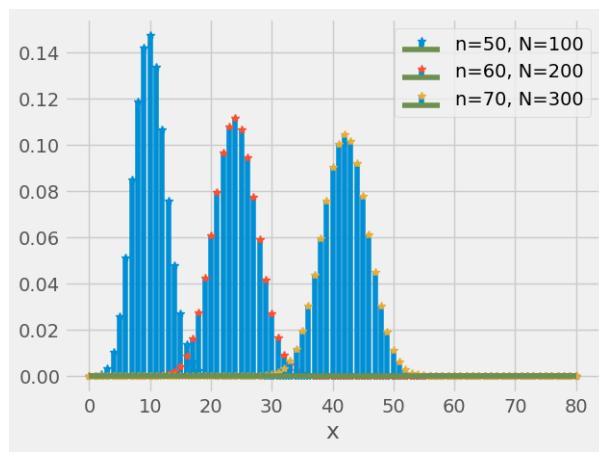


圖 3.3: 超幾何分配的機率質量函數圖

透過圖 3.3 可知，隨著參數  $n$  與  $N$  的值增加，超幾何分配的圖形也會逐漸向右偏移，即期望值變大，而且圖型的最高點會逐漸變小，從 0.15 左右下降至 0.1 左右。

### 3.1.3 幾何分配

幾何分配 (Geometric Distribution) 可以用兩種方法來進行解釋，第一種解釋是”在伯努利試驗中，得到一次成功所需的試驗次數 ( $X$ )”，另一種意思則為”在得到第一次成功之前所經歷的失敗次數 ( $Y = X - 1$ )”，第一種解釋的機率質量函數為：

$$P(X = k) = (1 - p)^{k-1}p \quad (3.1)$$

其中  $k = 1, 2, 3, \dots$

接著透過設置參數為  $p = 0.2, 0.4, 0.5, 0.7$  來繪製幾何分配的機率質量函數圖，其結果如圖 3.4，根據圖 3.4 可知，隨著  $p$  值增大，pmf 圖的端點會從 0.2 增至 0.7，其尾部也從遞減至  $x = 12$  變成遞減至  $x = 6$  左右。

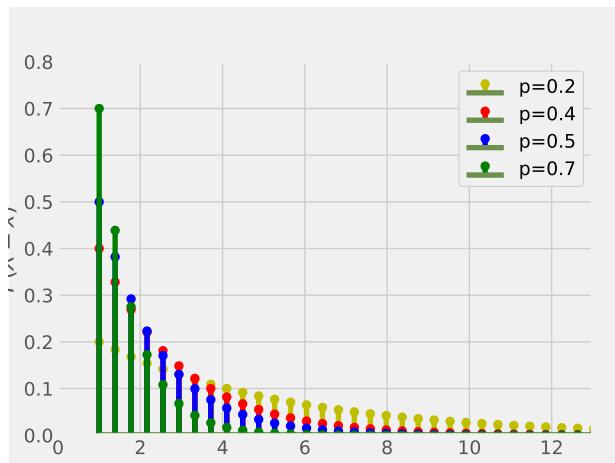


圖 3.4: 幾何分配的機率質量函數圖

### 3.1.4 卜瓦松分配

卜瓦松分配 (Poisson Distribution) 是用於描述單位時間內隨機事件發生的次數的機率分配，其參數  $\lambda$  為隨機事件發生次數的期望值，其機率質量函數 (pmf) 為：

$$P(X = k) = \frac{e^{-\lambda} \lambda^k}{k!} \quad (3.2)$$

圖 3.5 為設定參數為  $\lambda = 1, 4, 6, 8$  所繪製成的機率質量函數圖 (pmf)，由圖圖 3.5 可知，當  $\lambda$  較小時，卜瓦松分配為右偏分配，而當  $\lambda$  較大時，卜瓦松分配則為鐘型分配，其可能與二項分配相同，具有趨近常態分配的性質。

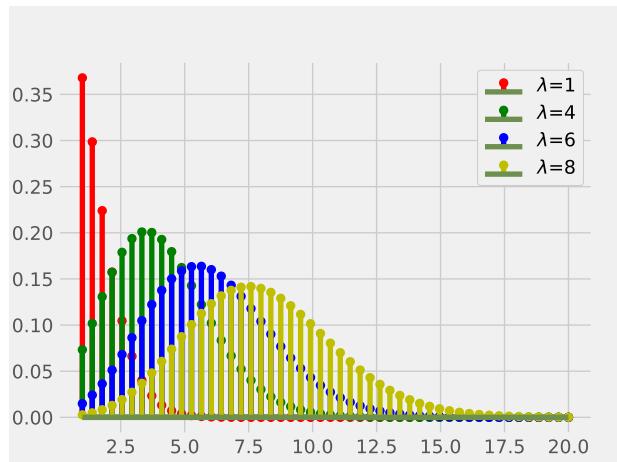


圖 3.5: 卜瓦松分配的機率質量函數圖

## 二項分配趨近卜瓦松分配

在數理統計中曾提到，當  $n$  趨近於無限大、 $p$  趨近於 0 時二項分配會趨近於卜瓦松分配，因此在此設置  $\lambda = np$  並改變二項分配的參數  $n, p$  以及卜瓦松分配的參數  $\lambda$  來觀察此趨近性質，其機率質量函數如圖 3.6。

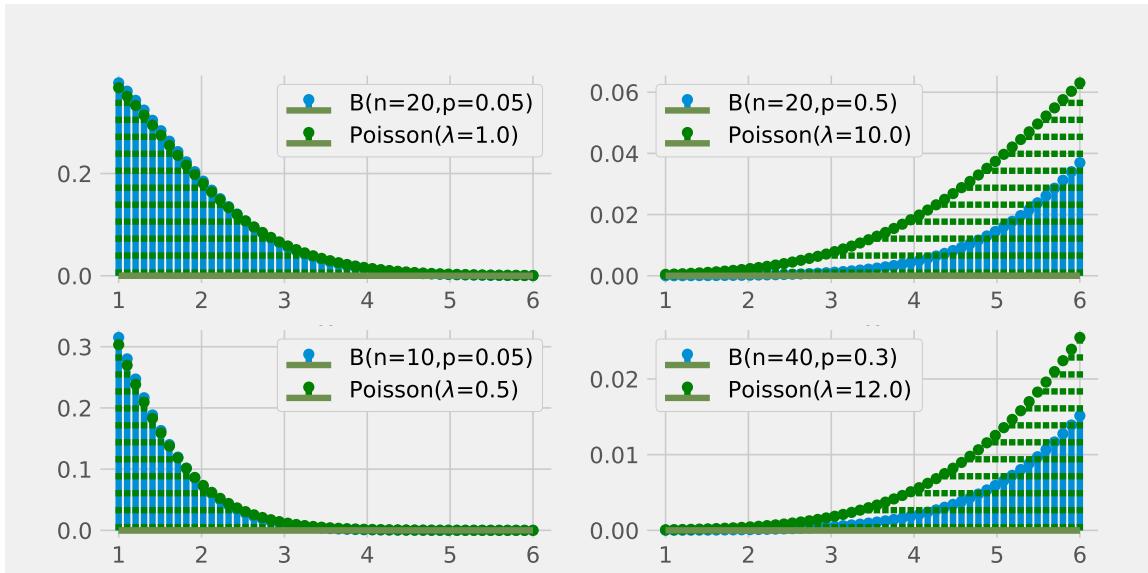


圖 3.6: 二項分配與卜瓦松分配的機率質量函數圖

由圖 3.6 的左上圖與右上圖可知，當  $n = 20$ 、 $p = 0.05$  時，二項分配趨近於卜瓦松分配，但當  $n = 20$ 、 $p = 0.5$  時，二項分配不會趨近於卜瓦松分配。接著，根據圖 3.6 的左下圖與右下圖可知，當  $n$  夠大且  $p$  夠小時二項分配會趨近於卜瓦松分配，而當  $n$  夠大但  $p$  不夠小時，二項分配不會趨近於卜瓦松分配，其部分程式碼呈現如下。

```

fig, axes = plt.subplots(2, 2, figsize=(10, 5))
x = np.linspace(1, 6, 50)

n, p = 10, 0.05
lamb = n*p
y = binom.pmf(x, n, p)
axes[1][0].stem(x, y, label="B(n={},p={})".format(n,p))
y1 = poisson.pmf(x, lamb)
axes[1][0].stem(x, y1, linefmt='g:', label="Poisson($\lambda$={})".format(lamb))
axes[1][1].set_xlabel("x")

plt.savefig(img_dir+"binom-poisson.eps", format="eps")
plt.show()

```

## 卜瓦松分配加成性

在數理統計中同樣提到卜瓦松分配的加成性質，即當  $X \sim Poisson(\lambda_1)$ 、 $X \sim Poisson(\lambda_2)$  時， $X + Y \sim Poisson(\lambda_1 + \lambda_2)$ 。因此在此利用亂數產生  $X$  與  $Y$  的卜瓦松分配隨機變數與  $X + Y \sim Poisson(\lambda_1 + \lambda_2)$  的機率質量函數圖進行比較，試圖證明此加成性確實成立，並繪製直方圖 (Histogram)、箱型圖 (Boxplot)、qqplot 圖與經驗分布函數圖 (Empirical CDF) 來觀察如果更改亂數產生的樣本數大小 ( $n$ ) 是否會使加成性質發生改變。

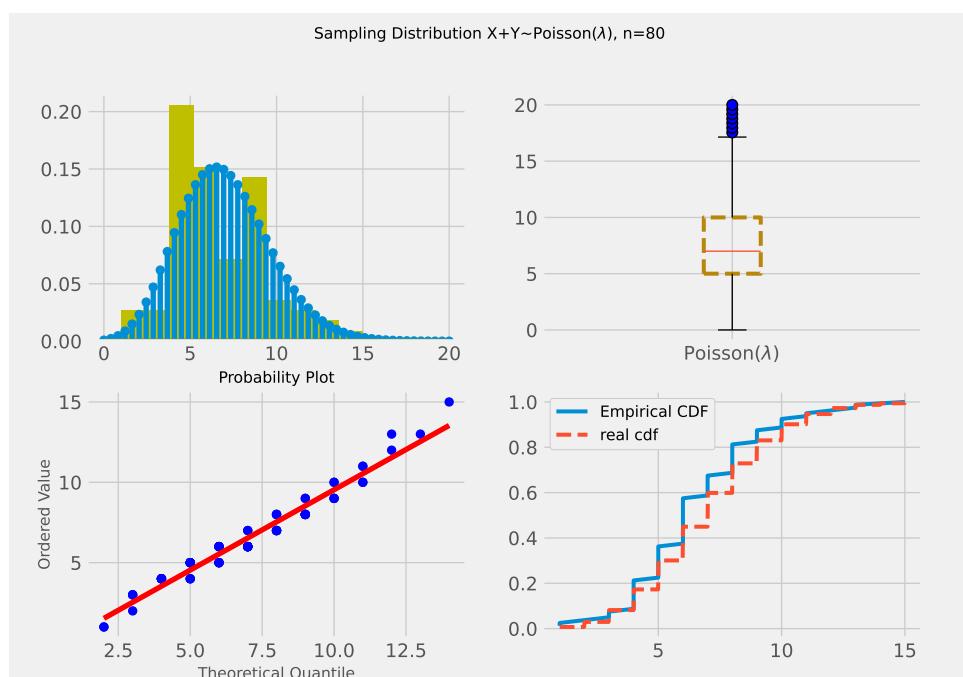


圖 3.7: 樣本數為 80 且服從  $Poisson(\lambda_1 + \lambda_2)$  之隨機變數  $X + Y$  的直方圖、箱型圖、qqplot 圖與經驗分布函數圖

首先設定樣本數為 80、兩個卜瓦松分配的參數為  $\lambda = 3$  以及  $\lambda = 4$  來繪製這四種圖形，圖 3.7 即為利用亂數產生兩個分配的結果。由圖 3.7 可知，雖然 qqplot 圖基本上貼合  $Poisson(\lambda_1 + \lambda_2)$  的 pmf 圖，但從其直方圖與經驗分布函數圖卻可以看出，似乎亂數產生的分配的經驗分布函數與  $Poisson(\lambda_1 + \lambda_2)$  的累積機率函數圖並未完全貼合，其部分程式碼如下。

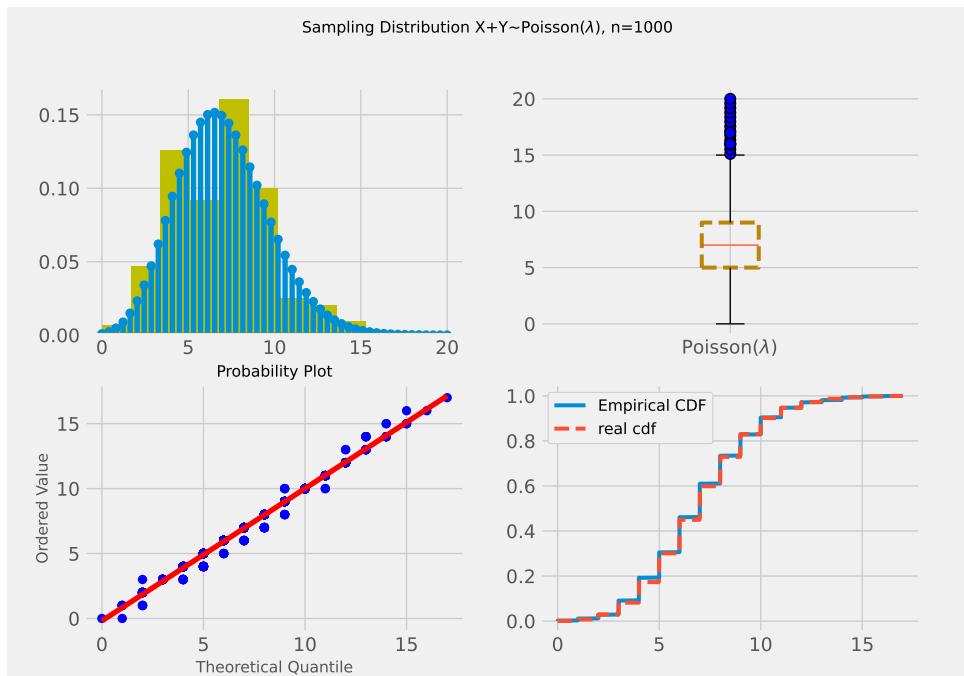


圖 3.8: 樣本數為 1000 且服從  $Poisson(\lambda_1 + \lambda_2)$  之隨機變數  $X + Y$  的直方圖、箱型圖、qqplot 圖與經驗分布函數圖

接著觀察將樣本數從 80 提升至 1000 時兩個卜瓦松分配的直方圖、箱型圖、qqplot 圖與經驗分布函數圖，如圖 3.8，從圖 3.8 中可以看出，與  $n = 80$  時不同的是，此時的 qqplot 圖點幾乎完全貼合在線上，且經驗分布函數圖與  $Poisson(\lambda_1 + \lambda_2)$  的累積機率函數圖也已完全貼合，由此可證得卜瓦松分配確實具有加成性。

```
np.random.seed(seed=1294) # 設定種子

n = 1000
lamb1 = 3
x1 = poisson.rvs(lamb1, size = n)
lamb2 = 4
x2 = poisson.rvs(lamb2, size = n)

## 兩個亂數抽樣分配相加
x4 = x1+x2
bins = 10
```

```

axes[0][0].hist(x4, density=True, bins = bins, alpha=0.5,
color="y")

##Poisson分配 (lamb1+lamb2)
x3 = np.linspace(0, 20, 50)
lamb3 = lamb1+lamb2
y = poisson.pmf(x3, lamb3)
axes[0][0].stem(x3, y)

##ECDF##
x_sort = np.sort(x4)
F = np.arange(1 ,n+1) / n
axes[1][1].plot(x_sort, F, lw =3, label = "Empirical CDF")

X = np.linspace(x_sort[0], x_sort[-1], 1000)
y = poisson.cdf(X, lamb3)
axes[1][1].plot(X, y, linestyle="--", lw = 3, label = "real
cdf")

```

## 卜瓦松抽樣分配

繪製卜瓦松分配的機率質量函數圖時曾經提到，卜瓦松分配 (Poisson Distribution) 似乎與二項分配 (Binomial Distribution) 都具有趨近常態分配的性質，因此在此節中嘗試利用亂數抽取卜瓦松分配並繪製其直方圖與 qqplot 圖，觀察卜瓦松分配與常態分配的差異以及樣本數改變可能導致的差異，此實驗以  $\text{Normal}(\mu, \sigma)$  作為比較，觀察 Poisson(2), Poisson(5) 與 Poisson(20) 三個分配，首先觀察在樣本數  $n = 20, n = 100, n = 1000$  時的直方圖。

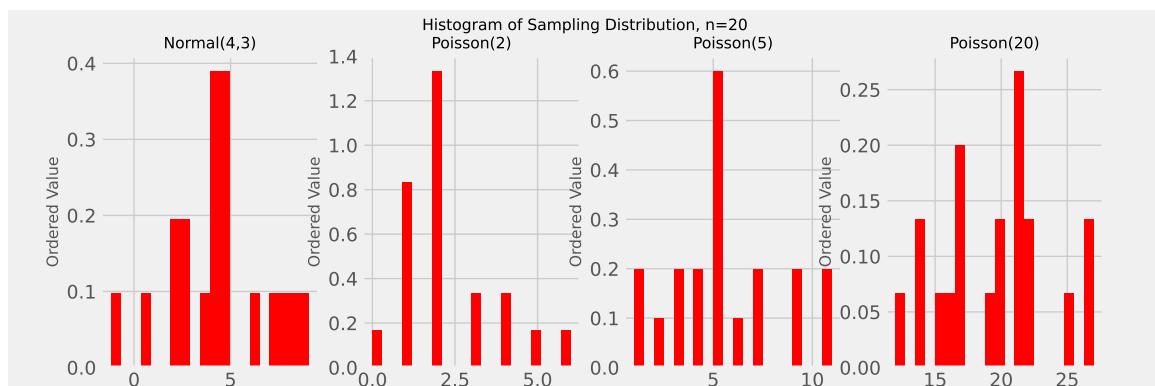


圖 3.9: 樣本數為 20 的卜瓦松抽樣分配直方圖

從圖 3.9 可知，在樣本數為 20 時，從直方圖中似乎無法觀察出甚麼特別的性質，因此持續將樣本數增加至  $n = 100$ ，甚至是  $n = 1000$ ，如圖 3.10 與 3.11。

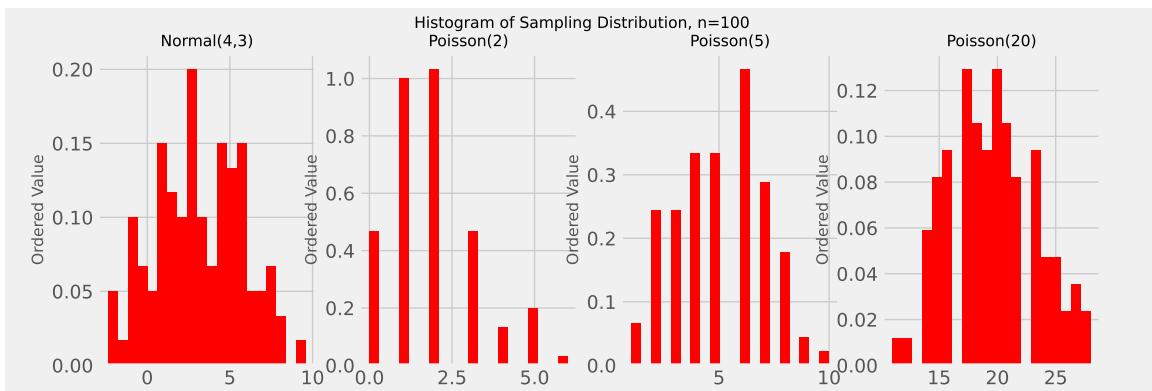


圖 3.10: 樣本數為 100 的抽樣卜瓦松分配直方圖

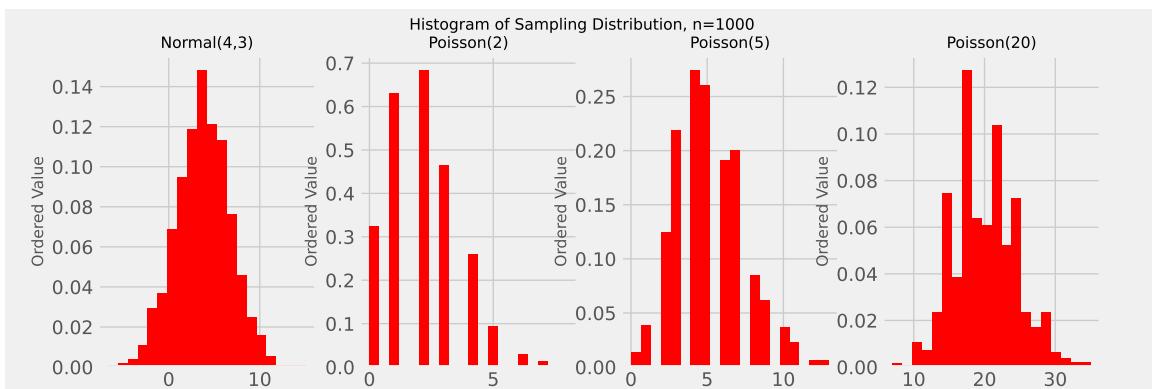


圖 3.11: 樣本數為 1000 的抽樣卜瓦松分配直方圖

根據上圖 3.10 與 3.11 可以發現，隨著樣本數變大以及卜瓦松分配的參數值變大，卜瓦松分配的直方圖會越來越趨近於鐘型分配的形狀，接著來觀察不同樣本數下卜瓦松分配的 qqplot 圖。

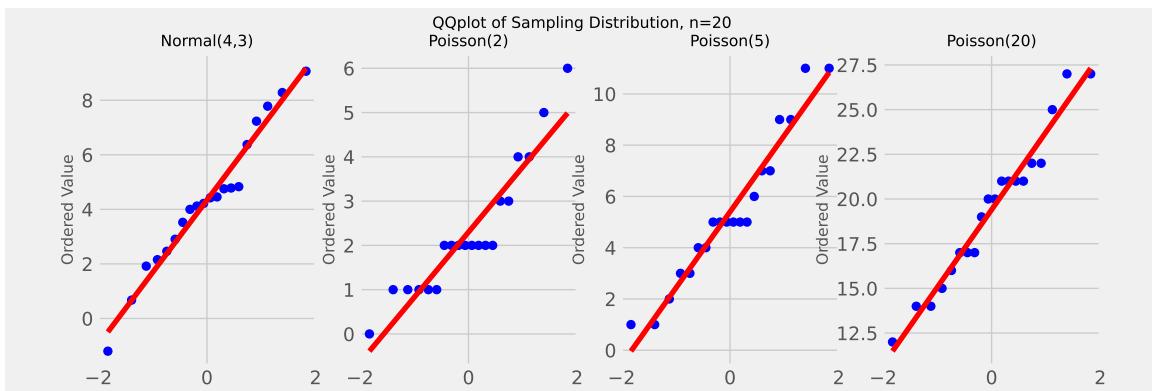


圖 3.12: 樣本數為 20 的抽樣卜瓦松分配 qqplot 圖

圖 3.12 展示了在樣本數為 20 的情況下，卜瓦松分配與常態分配的 qqplot 圖，在此圖中仍然可以觀察到卜瓦松分配的離散性質，並沒有觀察到趨近常態的特性，因此將樣本

數增加至  $n = 100$  甚至是  $n = 1000$  再重新繪製 qqplot 圖，如圖 3.13 與 3.14。

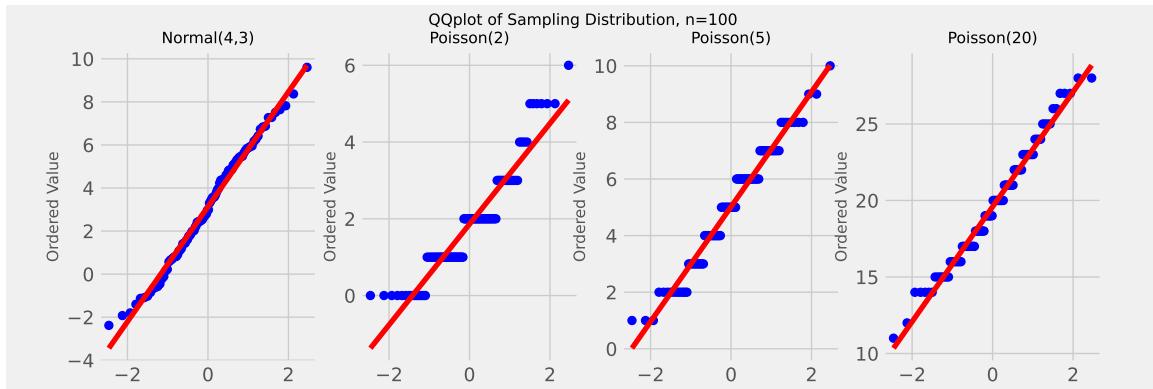


圖 3.13: 樣本數為 100 的抽樣卜瓦松分配 qqplot 圖

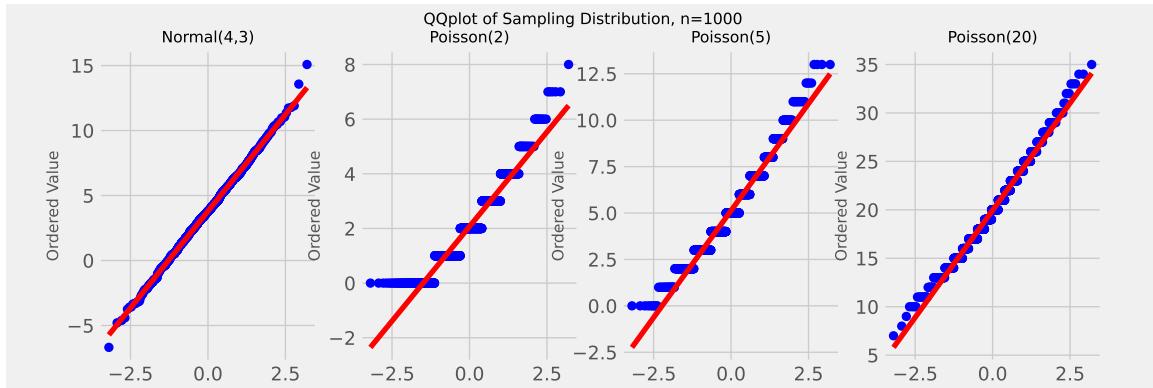


圖 3.14: 樣本數為 1000 的抽樣卜瓦松分配 qqplot 圖

由上圖 3.13 與 3.14 可觀察到，隨著樣本數增大，卜瓦松分配會與常態分配越來越貼近，此觀察到的結果與數理統計中所證明的性質一致，其部分程式碼呈現如下：

```

mu = 4
sigma = 3
n = 1000
rv_norm = norm.rvs(loc = mu, scale = sigma, size = n)
lamb1 = 2
lamb2 = 5
lamb3 = 20
rv_poi1 = poisson.rvs(lamb1, size = n)
rv_poi2 = poisson.rvs(lamb2, size = n)
rv_poi3 = poisson.rvs(lamb3, size = n)
ax1.hist(rv_norm, bins = bins, density=True, color="r", alpha
= 0.5)
stats.probplot(rv_poi1, dist = "norm", plot = ax2)

```

## 3.2 連續型分配

在前面的章節中介紹了幾種常見的離散型分配的機率質量函數 ( pmf )，包含二項分配 ( Binomial Distribution ) 以及卜瓦松分配 ( Poisson Distribution )，也利用抽樣方法介紹了一些分配與分配之間的關係，因此在此章節中將同樣列舉幾個常用的連續型分配並呈現其機率密度函數圖與累積機率函數圖，包含卡方分配 ( Chi-squared Distribution )、指數分配 ( Exponential Distribution )、雙指數分配 ( Double Exponential Distribution )、伽瑪分配 ( Gamma Distribution )、貝塔分配 ( Beta Distribution )、柯西分配 ( Cauchy Distribution )、常態分配 ( Normal Distribution )、司徒頓 t 分配 ( Student-t Distribution ) 以及 F 分配，此章節亦將探索一些分配之間的關係，例如卡方分配會趨近於常態分配與伽瑪分配的特性，並利用抽樣方法觀察卡方分配、伽瑪分配以及貝塔分配，首先介紹卡方分配。

### 3.2.1 卡方分配

$k$  個獨立的標準常態分配變數的平方和服從自由度為  $k$  的卡方分配 ( Chi-squared Distribution )，它也是一種特殊的伽瑪分配 ( Gamma Distribution )。其機率密度函數如式 (3.3)。

$$f_k(x) = \frac{1}{2^{\frac{k}{2}} \Gamma(\frac{k}{2})} 2^{\frac{k}{2}-1} e^{-\frac{x}{2}} \quad (3.3)$$

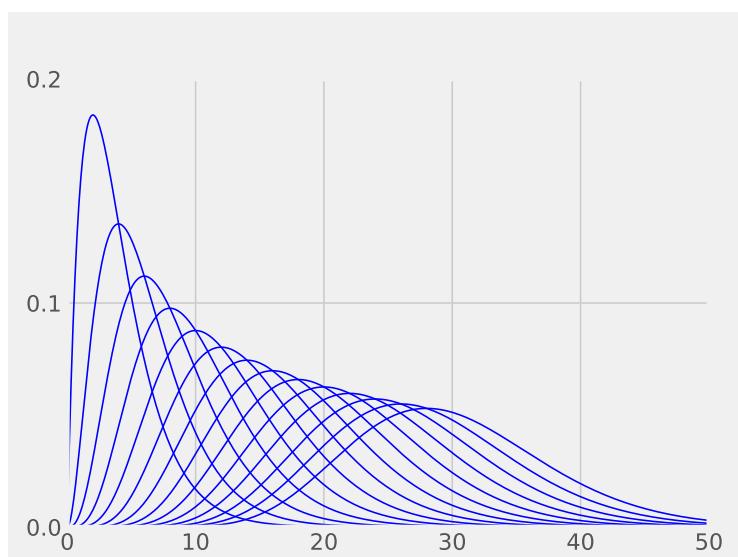


圖 3.15: 卡方分配的機率密度函數圖

在圖 3.15 可以觀察到設置卡方分配自由度分別為  $df = 4, 6, 8, \dots, 32$  時其機率密度函數圖型的變化，根據圖 3.15 可知，隨著參數的值變大，卡方分配會漸漸從右偏分配變成趨近於鐘型分配，似乎確實具有趨近於常態分配的性質，以下是繪圖的部分程式碼：

```

xlim = [0, 50]
x = np.linspace(xlim[0], xlim[1], 1000)
df = np.arange(4, 32, 2)
plt.figure()
plt.axis([xlim[0], xlim[1], 0, 0.2])
for i in df:
    y = chi2.pdf(x, i)
    plt.plot(x,y, lw=1, color='blue', alpha=0.4)

plt.yticks([0, 0.1, 0.2])
plt.savefig(img_dir+"chi-squared.eps", format="eps")
plt.show()

```

## 卡方分配趨近常態分配與伽瑪分配

繪製機率密度函數圖時可以發現，卡方分配似乎隨著自由度的增加會開始趨近於常態分配，因此在此透過繪製卡方分配  $\chi^2(1000)$  與常態分配  $Normal(\mu, \sigma)$  的 pdf 圖來嘗試驗證此想法是否是正確的。另外，在數理統計中曾經提到，若卡方分配的參數為  $v$ ，則此卡方分配與伽瑪分配  $\Gamma(\frac{v}{2}, \beta)$  相等，故在此章節中亦嘗試繪製兩者之機率密度函數圖來驗證此理論確實成立，此兩實驗的結果如圖 3.16。

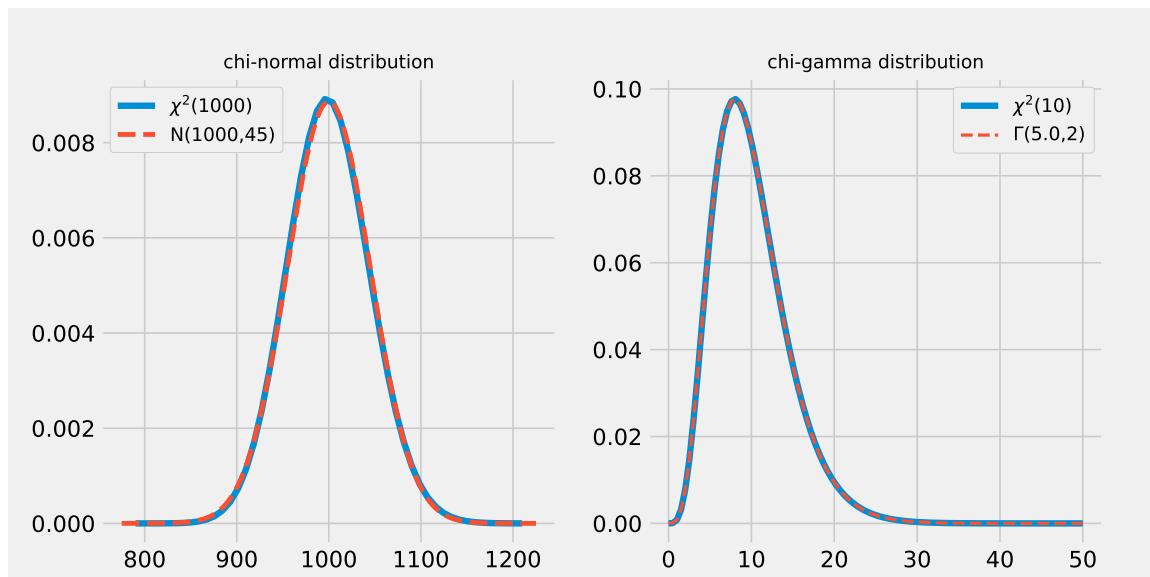


圖 3.16：卡方分配、常態分配與伽瑪分配的機率密度函數圖

由上圖 3.16 可知，隨著卡方分配的參數增大，卡方分配會由右偏分配變成趨近於鐘型分配，而由左圖可知，卡方分配與常態分配的 pdf 圖相同，其期望值為 1000。另外，由 3.16 之右圖可知，卡方分配  $\chi^2(v)$  與  $\Gamma(\frac{v}{2}, \beta)$  確實相等。其部分相關程式碼如下：

```
df = 1000
x = np.linspace(790, 1210), y = chi2.pdf(x.reshape(-1,1), df=df)
mu = 1000, sigma = 45
X = np.linspace(mu-5*sigma, mu+5*sigma, 500)
Y = norm.pdf(X, loc=mu, scale=sigma)
axes[0].plot(X, Y, lw=3, linestyle="--", label="Normal Distribution")

v = 10, df = v/2
x = np.linspace(0, 50, 100)
y1 = chi2.pdf(x.reshape(-1,1), df = v)
y2 = gamma.pdf(x, a = df, scale = 2)
```

## 卡方抽樣分配

在數理統計中曾經介紹過卡方分配與常態分配的關係，因此在此章節中將利用亂數產生服從常態分配的隨機變數，並繪製直方圖、箱型圖、qqplot 圖與經驗分布函數圖來觀察兩分配究竟是否相同，期望能證明標準常態分配的平方  $Z^2$  與  $\chi^2(1)$  相等，另外，此章節亦透過改變生成亂數的樣本大小  $n$  來觀察是否樣本數大小會造成兩分配的差異。

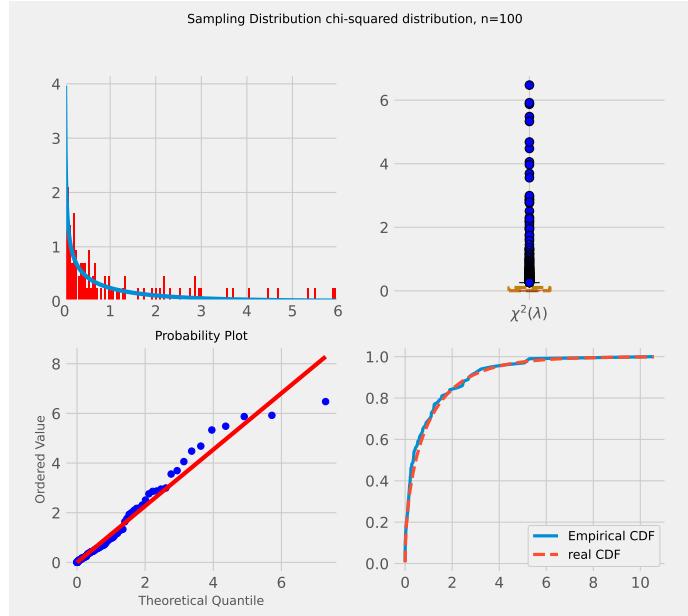


圖 3.17: 樣本數為 100 的  $\chi^2(1)$  直方圖、箱型圖、qqplot 圖與經驗分布函數圖

首先設置樣本數為 100 來繪製自由度為 1 的卡方分配直方圖、箱型圖、qqplot 圖與經驗分布函數圖，其結果如圖 3.15，根據圖 3.17 可以明顯發現，當樣本數達到 100 筆時，亂數產生的常態隨機變數的機率圖與卡方分配  $\chi^2(1)$  並未完全貼合，無法證明標準常態分配的平方  $Z^2$  與  $\chi^2(1)$  相等，因此接著將樣本數增大至 10000 筆並重新觀察直方圖、箱型圖、qqplot 圖與經驗分布函數圖，其結果如圖 3.18，其部分程式碼如下。

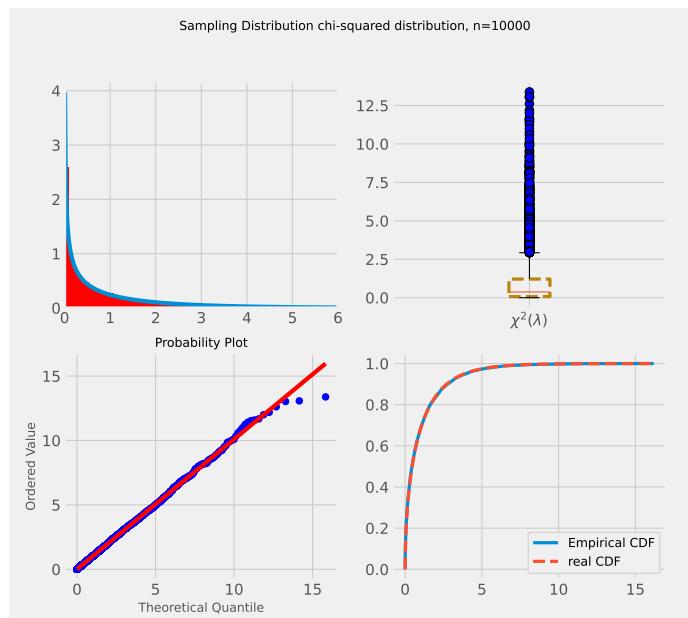


圖 3.18: 樣本數為 10000 的  $\chi^2(1)$  直方圖、箱型圖、qqplot 圖與經驗分布函數圖

由圖 3.18 可知，當樣本數由  $n = 100$  增加至  $n = 10000$  筆時，標準常態分配的平方  $Z^2$

與  $\chi^2(1)$  相等。

```
n = 100, bins = 150
x = norm.rvs(loc = 0, scale = 1, size = n), x = x**2

x_z = np.linspace(0, 10, 1000), z = chi2.pdf(x_z, df = 1)
boxprops = dict(linestyle = '--', linewidth = 3, color =
    'darkgoldenrod')
flierprops = dict(marker='o', markerfacecolor = 'blue',
    markersize = 8, linestyle = 'none')
labels = ["$\chi^2(\lambda)$"]
axes[0][1].boxplot(np.r_[x, z], boxprops = boxprops,
    flierprops = flierprops, labels = labels)
stats.probplot(x, dist = "chi2", sparams=(df), plot=axes
[1][0])
```

### 3.2.2 指數分配

指數分配 (Exponential Distribution) 是一種連續機率分配，用來表示獨立隨機事件發生所需的時間間隔，其機率密度函數如式 (3.4)。

$$f(x; \lambda) = \lambda e^{-\lambda x}, x \geq 0 \quad (3.4)$$

式 (3.4) 中的  $\lambda$  為分配的母數，代表每單位時間發生數件的次數，比例母數  $\beta$  代表該事件在每單位時間的發生率，可以利用  $\lambda = \frac{1}{\beta}$  來代替  $\lambda$ 。指數分配的期望值是  $\frac{1}{\lambda}$ ，變異數則為  $\frac{1}{\lambda^2}$ ，其機率密度函數與累積機率函數如圖 3.19。

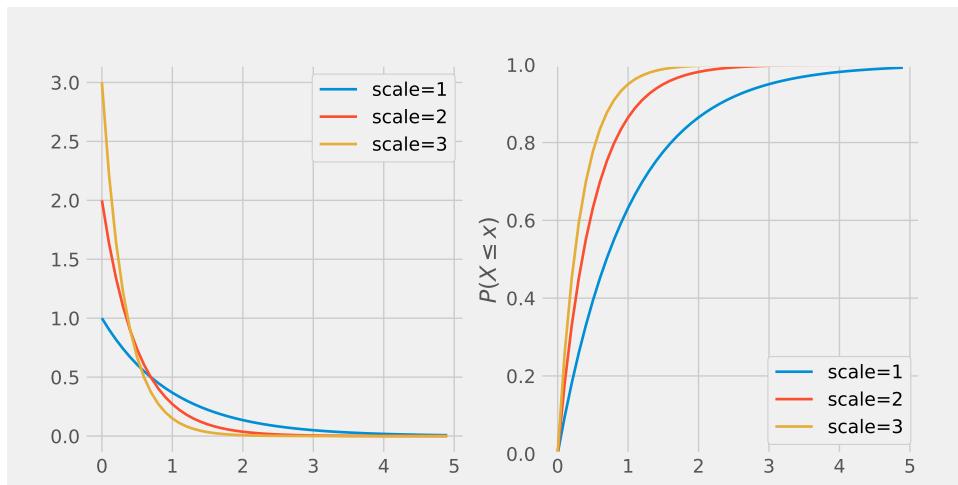


圖 3.19: 指數分配的機率密度函數圖

圖 3.19 中設定了參數  $\lambda = 1, 2, 3$  來觀察機率密度函數圖的變化，根據圖 3.19 可知，隨著參數  $\lambda$  的值增大，單位時間發生事件次數是 0 的次數變大，且機率密度函數圖遞減的速度加快。以下呈現部分的程式碼：

```
x1 = np.arange(0, 5, 0.1) #(15,)
scale = np.arange(1, 4) #(4,
for i in scale:
    y1 = i * np.exp(-i*x1)
    axes[0].plot(x1, y1, lw=2, label="scale={}".format(i))
```

在此題繪圖時遇到一些問題。若我們使用 `expon.pdf` 來生成 pdf 圖，則繪製出的圖形會有點怪異，如下圖 3.20，此結果來自於  $\lambda$  與 *scale* 的關係，其部分程式碼亦呈現於下。

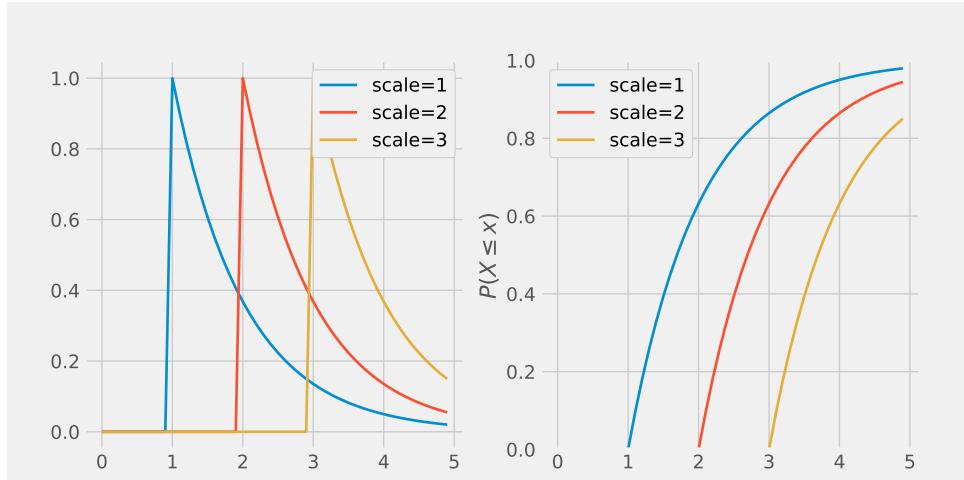


圖 3.20: 錯誤的指數分配機率密度函數圖與累積機率函數圖

```
x1 = np.arange(0, 5, 0.1) #(15,)
scale = np.arange(1, 4) #(4,)
for i in scale:
    y1 = expon.pdf(x1, i)
    axes[0].plot(x1, y1, lw=2, label="scale={}".format(i))
```

### 3.2.3 雙指數分配

雙指數分配 (Double Exponential Distribution)，亦稱為拉普拉斯分配 (Laplace Distribution)，此分配可以看作兩個平移指數分配背靠背拼接在一起，其機率密度函數如式 (3.5)。

$$f(x|\mu, b) = \frac{1}{2b} \exp\left(-\frac{|x - \mu|}{b}\right) \quad (3.5)$$

圖 3.21 是利用函數 `laplace.pdf` 繪圖所得到的結果，由圖 3.21 可知，當參數  $b$  從 1 增加到 4 時，整個機率密度函數圖會逐漸變寬，變異數變大且端點的值從 0.5 左右下降至 0.1 左右。 $\mu$  的改變則會讓圖形產生左移或右移，若  $\mu$  值變大會往右移，變小則往左移。

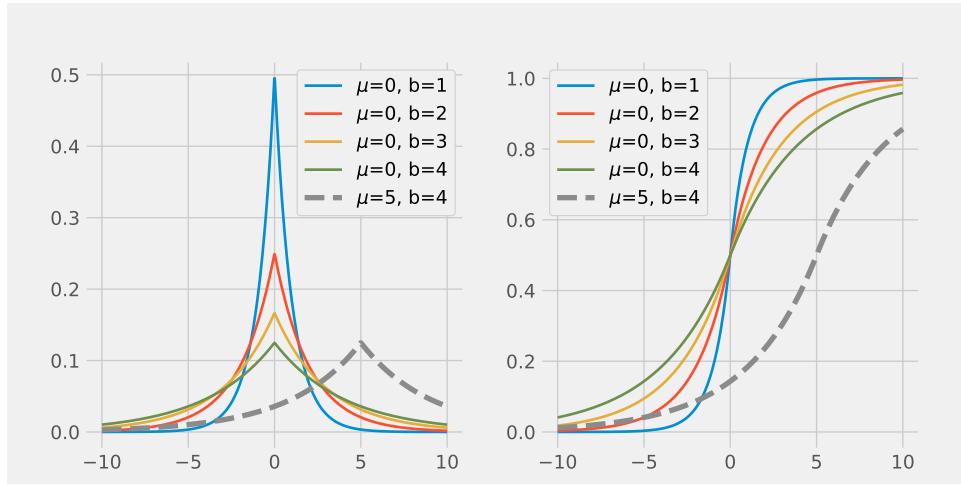


圖 3.21：雙指數分配的機率密度函數圖與累積機率函數圖

### 3.2.4 伽瑪分配

假設  $X_1, X_2, \dots, X_n$  為連續發生事件的等候時間，且這  $n$  次等候時間相互獨立，則  $Y = X_1 + X_2 + \dots + X_n$  服從伽瑪分配 (Gamma( $\alpha, \beta$ ))，其中  $\alpha = n$ ，而  $\beta$  與  $\lambda$  互為倒數， $\lambda$  代表單位時間內事件的發生率。其機率密度函數如式 (3.6)，圖形則呈現於圖 3.22。

$$f(x) = \frac{x^{\alpha-1} \lambda^{-\alpha} e^{(-\lambda x)}}{\Gamma(\alpha)}, x > 0 \quad (3.6)$$

圖 3.22 即為伽瑪分配的機率密度函數圖，其程式碼呈現於上方，圖中深藍色的機率密度函數圖為設置伽瑪分配的參數為  $\alpha = 1, 2.5, 4, 5.5, 7, 8.5, 10, 11.5$  且  $\beta = 1, 1.5, 2, 2.5, 3, 3.5, 4, 4.5$  共八組不同參數所得到的結果，另外亦設置參數  $(\alpha, \beta) = (1, 2), (1, 5), (1, 9)$  來觀察改變參數  $\beta$  造成的圖形變化，其結果為淺藍色的圖形。根據圖 3.22 可知，隨著  $\alpha$  值與  $\beta$  值變大，伽瑪分配會逐漸從右偏分配變成鐘型分配，此性質與卡方分配相近，而若僅僅增大參數  $\beta$  的值，分配則會逐漸向右偏移。

```
x = np.linspace(0, 20, 1000)
k = np.arange(1, 13, 1.5) # 0 1.5 3 4.5 6 7.5
theta = np.arange(1, 5, 0.5) # 0 0.5 1 1.5 2 2.5
param = np.vstack((k, theta)) #(2, 6) #矩陣
param = param.T #(6, 2)
for i in range(8):
    y = gamma.pdf(x, param[i][0], param[i][1])
    plt.plot(x, y, lw=2, c="blue", alpha=0.5)
```

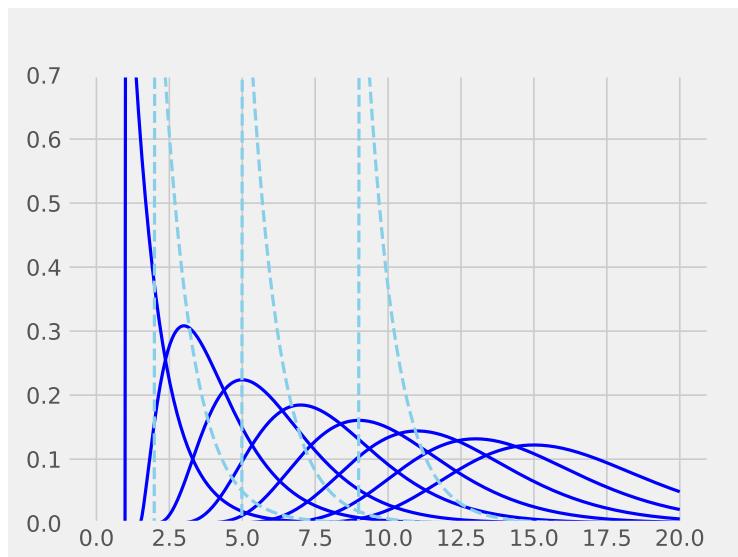
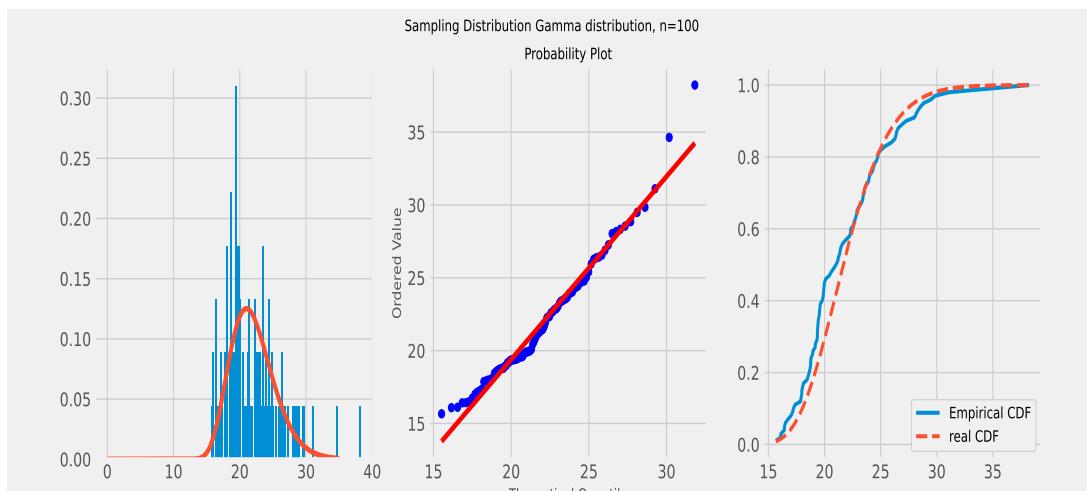


圖 3.22: 伽瑪分配的機率密度函數圖

### 伽瑪分配加成性

在了解伽瑪分配的基本性質後，接著利用亂數產生伽瑪分配來嘗試證明伽瑪分配的可加性，即當  $X \sim \Gamma(\alpha_1, \beta_1)$ 、 $Y \sim \Gamma(\alpha_2, \beta_2)$  時， $X + Y \sim \Gamma(\alpha_1 + \alpha_2, \beta_1 + \beta_2)$ 。在此實驗中將亂數產生  $X, Y$  兩個獨立分配再與  $X + Y$  分配共同繪製直方圖、qqplot 圖與 ecdf 圖，並更改樣本數來觀察樣本數可能造成的差異。

圖 3.23: 樣本數為 100 的  $\Gamma(\alpha_1 + \alpha_2, \beta_1 + \beta_2)$  機率密度函數圖、qqplot 圖與經驗分布函數圖

設置樣本數為 100 來觀察抽樣分配  $X \sim \Gamma(1, 10)$ ,  $Y \sim \Gamma(10, 1)$  與  $X+Y \sim \Gamma(1+10, 10+1)$  的擬合結果，如圖 3.23，其結果都是兩者並不是非常貼近無法證明伽瑪分配的可加性，故接著將樣本數提高到  $n = 1000$  再重新繪製三種圖形，其部分程式碼呈現如下。

```

alpha1 = 1, beta1 = 10
beta2 = 1, alpha2 = 10
n = 100

x1 = gamma.rvs(alpha1, beta1, size=n)
x2 = gamma.rvs(alpha2, beta2, size=n)
x1 = np.sort(x1), x2 = np.sort(x2), x3 = x1+x2
axes[0].hist(x3, bins=100, density=True)
x4 = np.linspace(0, 35, 1000)
y = gamma.pdf(x4, alpha1+alpha2, beta1+beta2)
axes[0].plot(x4, y)

x_sort = np.sort(x3), F = np.arange(1 ,n+1) / n
axes[2].plot(x_sort, F, lw =3, label="Empirical CDF")
X = np.linspace(x_sort[0], x_sort[-1], 1000)
y = gamma.cdf(X, alpha1+alpha2, beta1+beta2)
axes[2].plot(X, y, linestyle="--", lw = 3, label="real CDF")

```

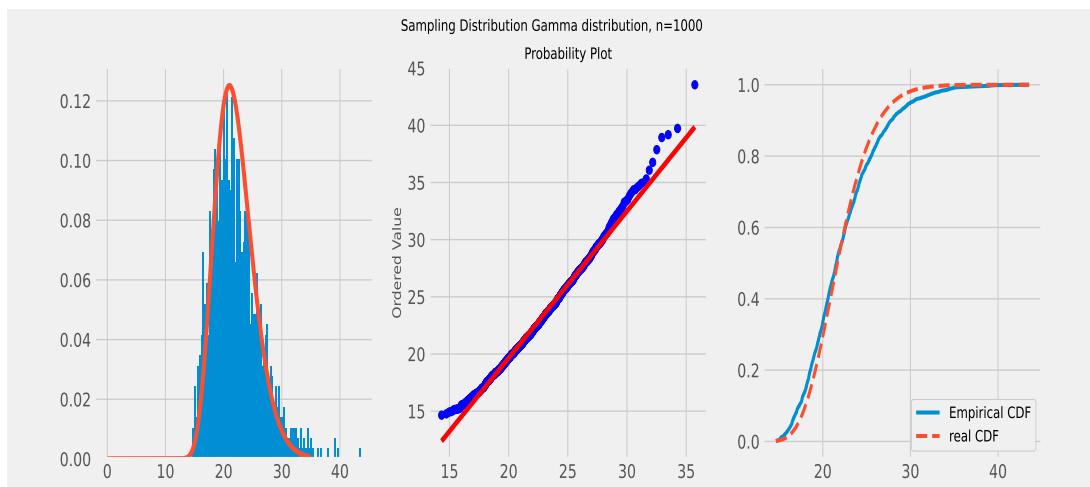


圖 3.24: 樣本數為 1000 的  $\Gamma(\alpha_1 + \alpha_2, \beta_1 + \beta_2)$  機率密度函數圖、qqplot 圖與經驗分布函數圖

由圖 3.24 可知，亂數產生的  $X \sim \Gamma(\alpha_1, \beta_1)$ 、 $Y \sim \Gamma(\alpha_2, \beta_2)$  與分配  $X + Y \sim \Gamma(\alpha_1 + \alpha_2, \beta_1 + \beta_2)$  已經幾乎擬合，由此可證伽瑪分配的可加性。

## 伽瑪抽樣分配

在此章節中，透過亂數抽取服從伽瑪分配的隨機變數並繪製直方圖與 qqplot 圖來觀察其與常態分配的差異以及改變樣本數可能導致的差異。此實驗將觀察 Norma(4, 3) 與  $\Gamma(1, 31)$ 、 $\Gamma(31, 1)$  與  $\Gamma(50, 50)$  三個分配在樣本數  $n = 50, n = 1000$  時的直方圖。

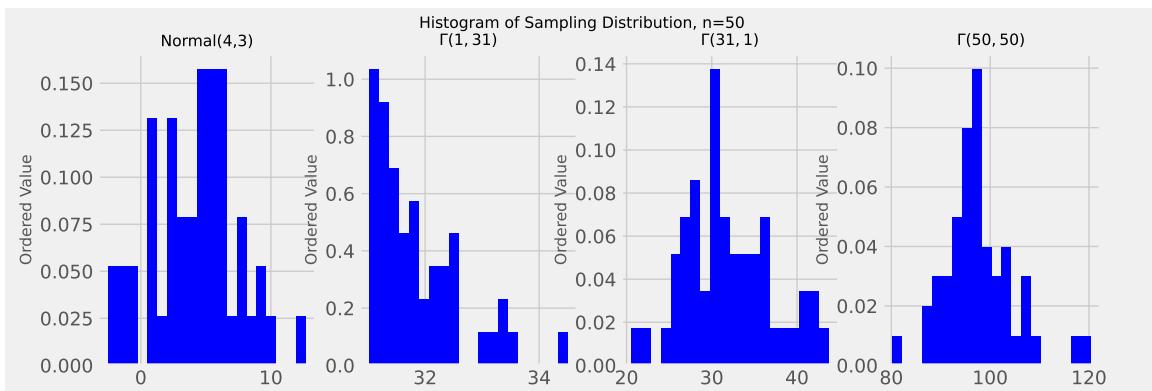


圖 3.25: 樣本數為 50 的伽瑪抽樣分配直方圖

首先圖 3.25 呈現了在樣本數  $n = 50$  時四個分配的機率直方圖，從此直方圖可以看出伽瑪分配的右偏與左偏等特性。接著將抽取的樣本數增加至  $n = 1000$ ，可得圖 3.26。

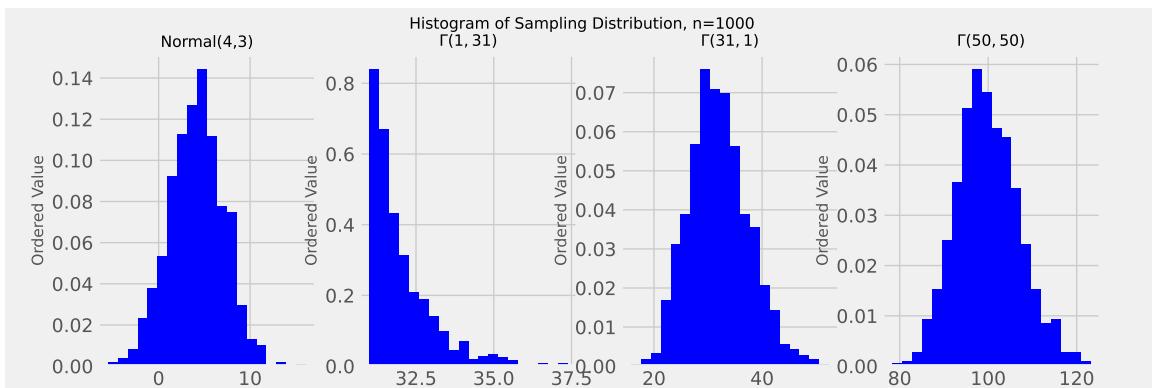


圖 3.26: 樣本數為 1000 的伽瑪抽樣分配直方圖

從圖 3.26 可以看出，當樣本數增加至  $n = 1000$  時， $\Gamma(50, 50)$ 、 $\Gamma(31, 1)$  與常態分配的形狀趨近。最後觀察  $\text{Normal}(4, 3)$  與  $\Gamma(1, 31)$ 、 $\Gamma(31, 1)$  與  $\Gamma(50, 50)$  三個分配的 qqplot 圖。

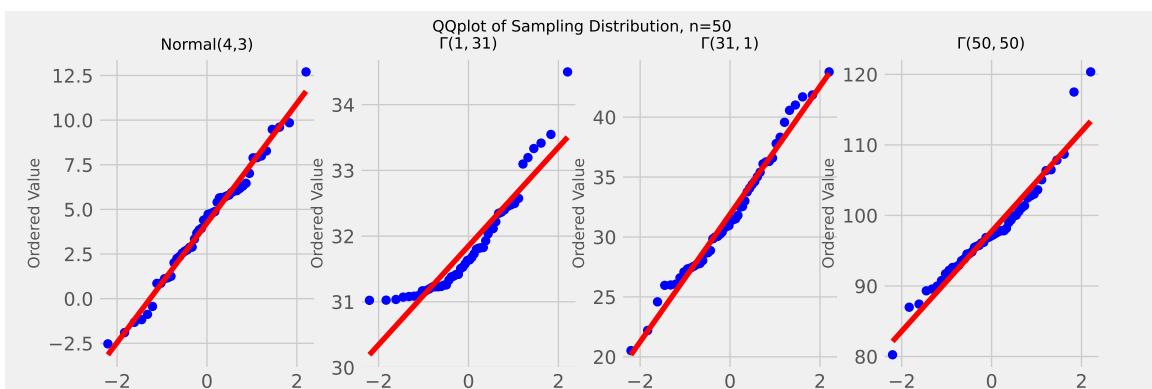


圖 3.27: 樣本數為 50 的伽瑪抽樣分配 qqplot 圖

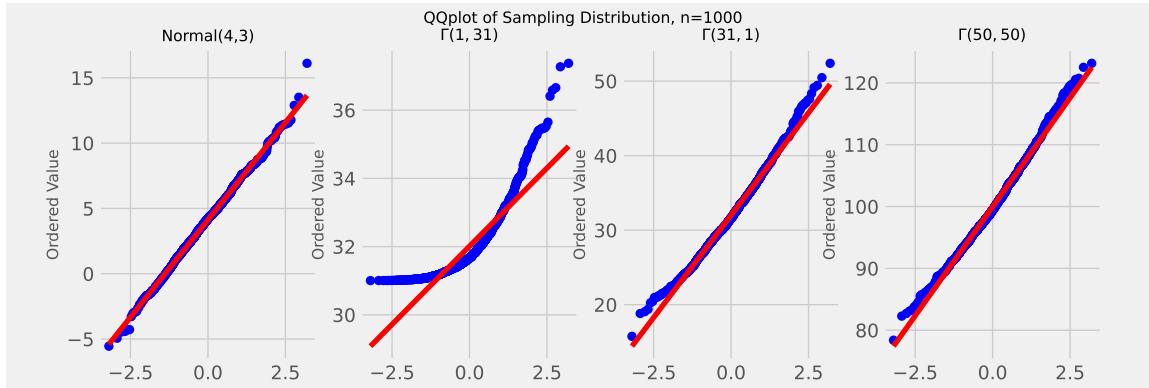


圖 3.28: 樣本數為 1000 的伽瑪抽樣分配 qqplot 圖

由圖 3.27 與 3.28 可以看出，無論樣本數有多大， $\Gamma(1, 31)$  都不會貼近常態分配，而  $\Gamma(31, 1)$  與  $\Gamma(50, 50)$  在樣本數僅有 50 時還不與常態分配貼近，但當樣本數達到 1000 時，兩個分配都會貼近常態分配。

### 3.2.5 貝塔分配

貝塔分配 (Beta Distribution) 是一組定義在區間  $(0,1)$  上的連續機率分配，其機率密度函數如式 (3.7)，圖形則呈現於圖 3.29。

$$f(x; \alpha, \beta) = \frac{x^{\alpha-1}(1-x)^{\beta-1}}{\int_0^1 u^{\alpha-1}(1-u)^{\beta-1} du} = \frac{\Gamma(\alpha + \beta)}{\Gamma(\alpha)\Gamma(\beta)} x^{\alpha-1}(1-x)^{\beta-1} \quad (3.7)$$

另外，貝塔分配的期望值為  $\frac{\alpha}{\alpha+\beta}$ 、變異數為  $\frac{\alpha\beta}{(\alpha+\beta)^2(\alpha+\beta+1)}$ 。

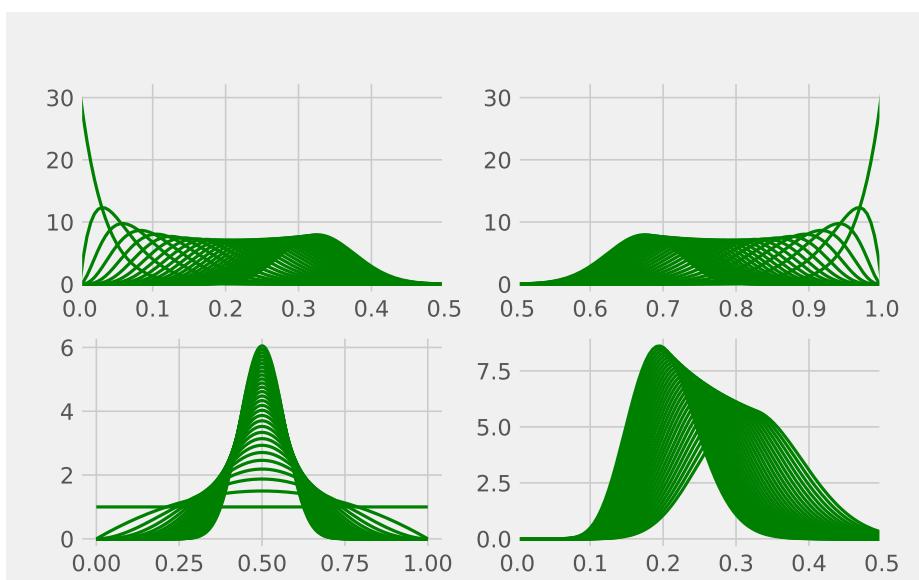


圖 3.29: 貝塔分配的機率密度函數圖

圖 3.29 設置了不同的  $\alpha$ 、 $\beta$  來理解貝塔分配的機率密度函數。在該圖中，左上圖設置參數  $a > b$ ， $a$  的值介在 1 到 30 之間、 $b$  的值介在 31 到 60 之間。右上圖則設置參數為  $a < b$ ，即  $a$  的值介在 31 到 60 之間、 $b$  的值介在 1 到 30 之間。在左下圖中，設置  $a$  與  $b$  的值相同，且介在 1 到 30 之間，最後的右下圖中則繪製給定  $a = 15$ ， $b$  介在 31 到 60 之間的貝塔分配機率密度函數圖。

由圖 3.29 可知，在  $a > b$  時，貝塔分配為右偏分配、在  $a < b$  時，貝塔分配為左偏分配，而在  $a = b$  時，貝塔分配則為鐘型分配。另外，在固定  $a$  的情況下，若  $b$  值增加時 ( $a < b$ )，貝塔分配會逐漸變成左偏分配，以下呈現其部分程式碼。

```
a = np.arange(1, 30)
b = np.arange(31, 60)
x = np.linspace(0, 1, 200) # 向量
Y = beta.pdf(x.reshape(-1,1), a, b) # 矩陣
axes[0][0].plot(x, Y, lw=2, c="g", alpha=0.5)
axes[0][0].set_xlim(0, 0.5)
```

## 貝塔抽樣分配

在此小節中，嘗試利用亂數產生抽樣分配  $Beta(15, 30)$ 、 $Beta(30, 30)$  以及  $Beta(30, 15)$ ，並繪製樣本數大小分別為  $n = 200$  與  $n = 1000$  的直方圖、qqplot 圖與經驗分布函數圖，觀察參數的變化對圖形產生的影響，以及貝塔分配與常態分配的關係。

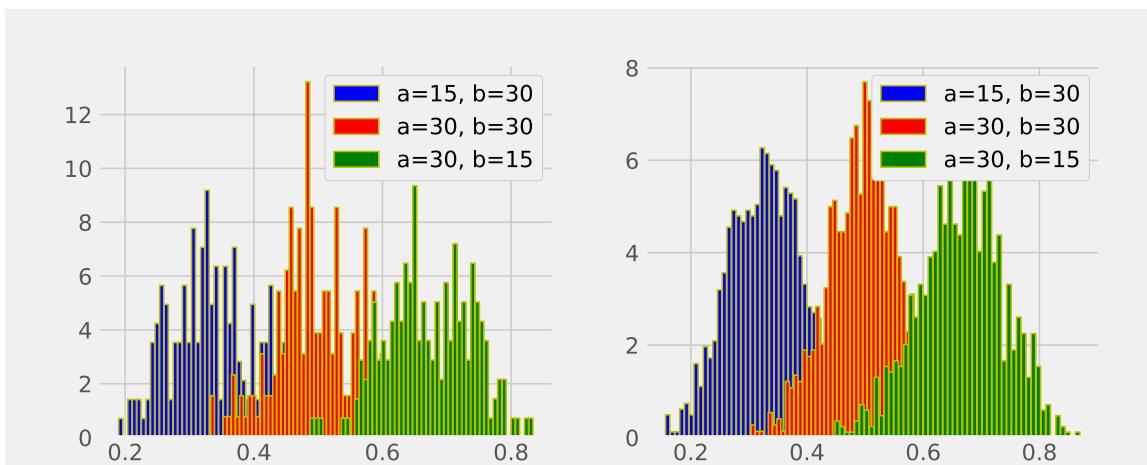


圖 3.30: 貝塔抽樣分配直方圖

在圖 3.30 的左圖為樣本數  $n = 200$  的直方圖，右圖則為樣本數  $n = 1000$  的直方圖。由圖 3.30 可知，在樣本數夠大時，無論  $a < b$ 、 $a = b$  還是  $a > b$ ，貝塔分配的機率密度函數圖都會趨近於鐘型分配，似乎與常態分配相同。接著，繪製在樣本數為 1000 時， $Beta(15, 30)$ 、 $Beta(30, 30)$  以及  $Beta(30, 15)$  的 qqplot 圖。

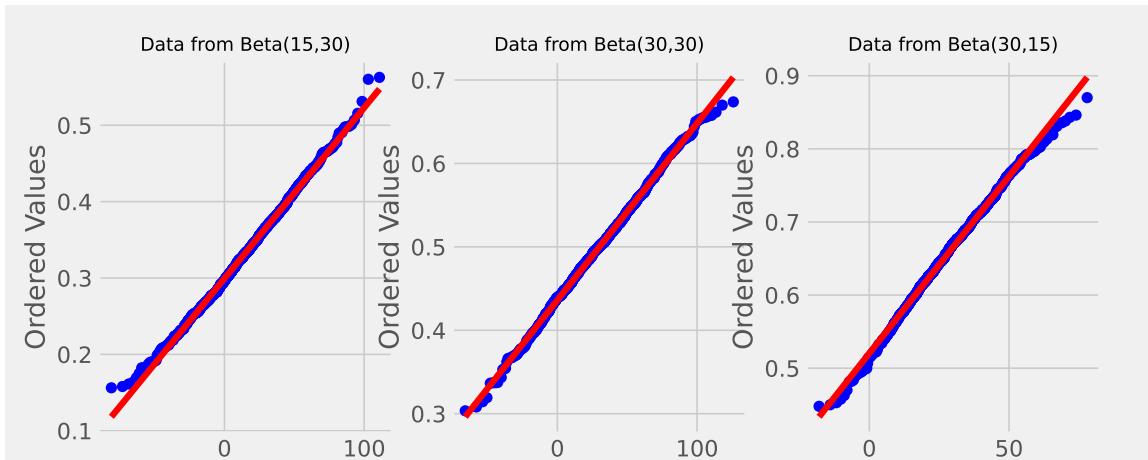


圖 3.31: 貝塔分配 qqplot 圖

由圖 3.31 可知，亂數產生服從貝塔分配的隨機變數會幾乎貼合紅色的線，即在樣本數夠大時，無論參數的值為何，貝塔分配都會趨近於常態分配。最後，繪製不同樣本數時的經驗分布函數圖，如圖 3.32。

由圖 3.32 可知，當  $n = 20$  時，亂數抽取的抽樣分配與貝塔分配的經驗分布函數不完全相同，但是在樣本數  $n = 500$  時，兩者則幾乎完全貼近。以下為其部分程式碼：

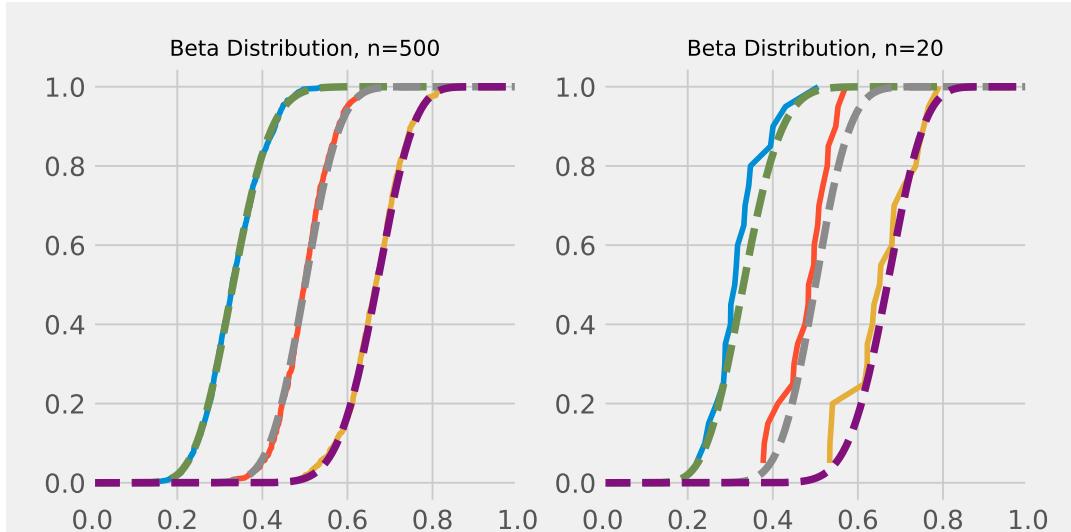


圖 3.32: 貝塔分配經驗分布函數圖

```

n = 500
x1 = beta.rvs(a1, b1, size=n)
x2 = beta.rvs(a2, b2, size=n)
x3 = beta.rvs(a3, b3, size=n)
x_sort1 = np.sort(x1)
F = np.arange(1 ,n+1) / n
axes[0].plot(x_sort1, F, lw =3)

```

```

x_sort2 = np.sort(x2)
x_sort3 = np.sort(x3)

x = np.linspace(0, 1, 1000)
y1 = beta.cdf(x.reshape(-1, 1), a1, b1)
y2 = beta.cdf(x, a2, b2)
y3 = beta.cdf(x, a3, b3)

```

### 3.2.6 柯西分配

柯西分配 (Cauchy Distribution) 是一種連續機率分配，其機率密度函數如式 (3.8)。

$$f(x; x_0, \gamma) = \frac{1}{\pi \gamma [1 + (\frac{x-x_0}{\gamma})^2]} = \frac{1}{\pi} \left[ \frac{\gamma}{(x - x_0)^2 + \gamma^2} \right] \quad (3.8)$$

在  $x_0 = 0$  且  $\gamma = 1$  的特例為標準柯西分配，其機率密度函數呈現於式 (3.9)。

$$f(x; 0, 1) = \frac{1}{\pi(1 + x^2)} \quad (3.9)$$

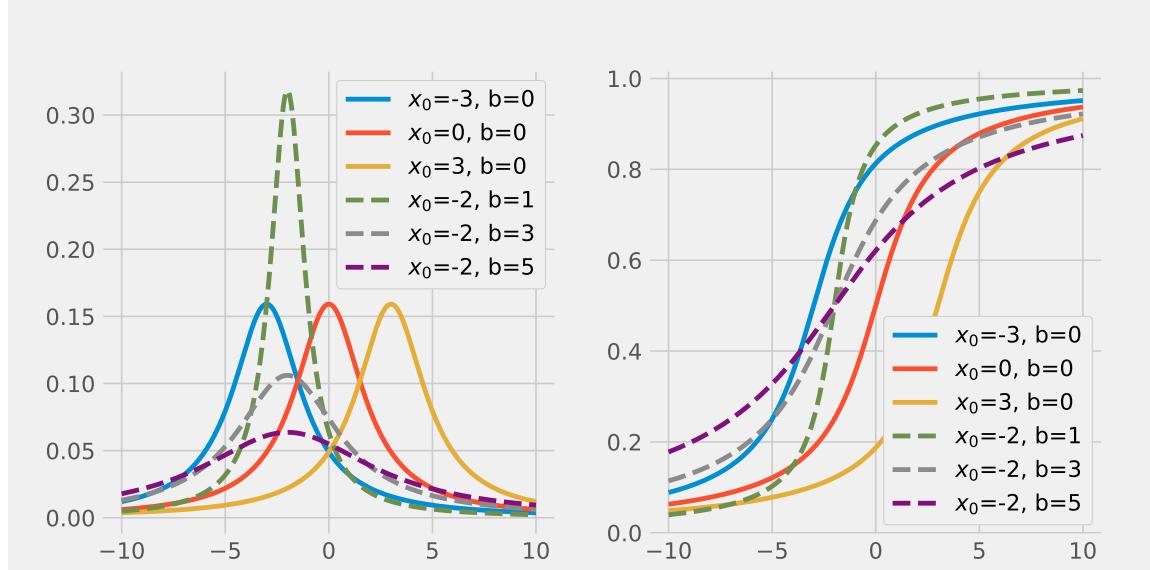


圖 3.33: 柯西分配的機率密度函數圖與累積機率函數圖

在圖 3.33 中，透過改變參數  $x_0$  與  $b$  的值來觀察柯西分配的機率密度函數圖與累積機率函數圖。由其左圖可知，改變  $x_0$  會改變其分配的位置，而改變  $b$  的值則會改變圖形的形狀。隨著  $b$  值變大，分配的端點會從 0.3 變成 0.05 左右，而其累積機率函數圖同樣會隨著  $x_0$  與  $b$  的改變而變化。

### 3.2.7 常態分配

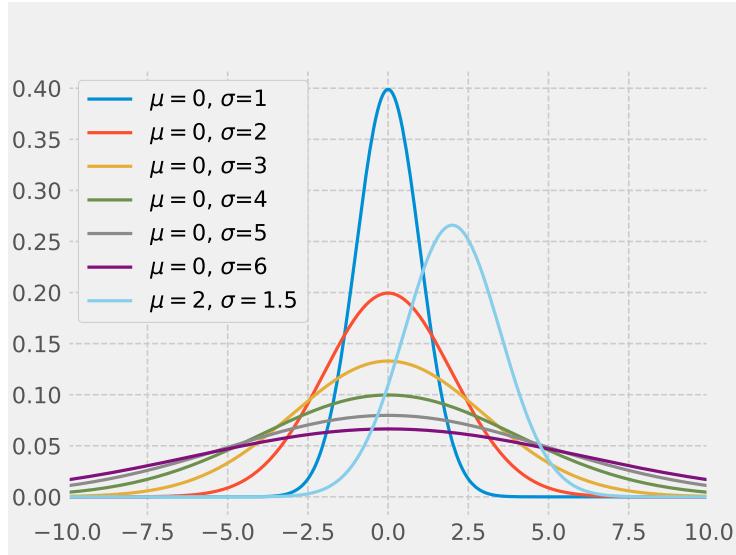


圖 3.34: 常態分配機率分布圖

常態分配式統計學中應用最多性質最佳之分配，圖 3.34 即為常態分配改變參數所得之機率密度函數圖，由圖可知， $\mu$  的改變會使分配平移， $\sigma$  的改變則會使分配的圖形改變，以下呈現其部分的程式碼：

```
mu = 0
sigma = np.arange(1, 7)
xlim = [mu - 5 * sigma.max(), mu + 5 * sigma.max()]
x = np.linspace(xlim[0], xlim[1], 1000)
Y = norm.pdf(x.reshape(-1, 1), loc=mu, scale=sigma)

plt.plot(x, Y, label = ["$\mu=${},".format(i) for i
    in sigma], lw=2)
```

### 3.2.8 司徒頓 t 分配

司徒頓 t 分配 (Student's t-distribution) 是用以根據小樣本來估計母體呈常態分配且標準差未知的期望值，若母體標準差已知或樣本數夠大時則用常態分配進行估計。t 分配的機率密度函數如式 (3.10)，改變其參數所得之機率密度函數圖亦呈現於圖 3.35。

$$f(t) = \frac{\Gamma \frac{v+1}{2}}{\sqrt{v\pi} \Gamma \left(\frac{v}{2}\right)} \left(1 + \frac{t^2}{v}\right)^{-\frac{(v+1)}{2}} \quad (3.10)$$

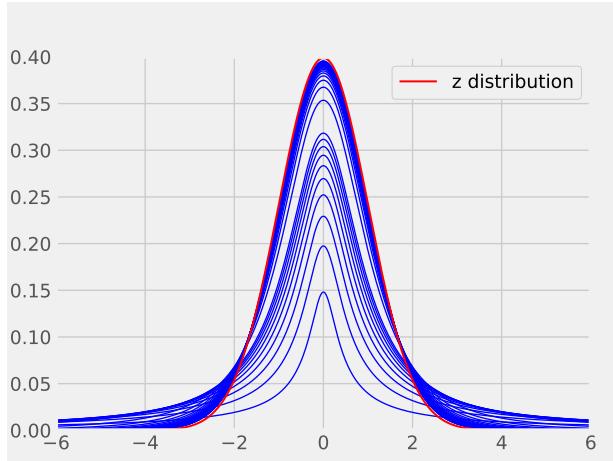


圖 3.35: 司徒頓 t 分配機率分布圖

圖 3.35 中比較了標準常態分配與改變各種參數得之 t 分配，由該圖可知，在 t 分配的自由度夠大時，t 分配會與標準常態分配相同，以下呈現其部分程式碼：

```

xlim = [-6, 6]
x = np.linspace(xlim[0], xlim[1], 1000)
df = np.r_[np.arange(0.1, 1, 0.1), np.arange(1, 30)]
plt.figure()
plt.axis([xlim[0], xlim[1], 0, 0.2])
for i in df:
    y=t.pdf(x, i)
    plt.plot(x,y, lw=1, color='blue', alpha=0.3)

```

### 3.2.9 F 分配

定義隨機變數  $X$  有母數為  $d_1$  與  $d_2$  的 F 分配，寫作  $X \sim F(d_1, d_2)$ ，對於實數  $x \geq 0$ ，其機率密度函數如式 (3.11)。

$$f(x; d_1, d_2) = \frac{\sqrt{\frac{(d_1 x)^{d_1} d_2^{d_2}}{(d_1 x + d_2)^{d_1 + d_2}}}}{x B\left(\frac{d_1}{2}, \frac{d_2}{2}\right)} \quad (3.11)$$

接著嘗試設置不同的參數  $d_1$  與  $d_2$  來繪製 F 分配的機率密度函數圖與累積機率函數圖，如圖 3.36，設置  $d_1 = 1, 2, 3, 4$  且  $d_2 = 1, 2, 3, 4$ ，可得到圖中的實線圖，另外，亦設置參數  $d_1 = 10, 30, 50, 70, 90$  與  $d_2 = 60$  來繪製機率密度函數圖與累積機率函數圖，可得到圖 3.36 中的虛線圖。

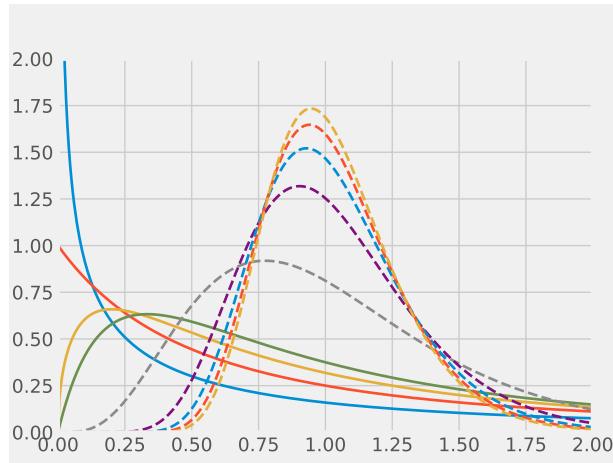


圖 3.36: F 分配機率分布圖

```

x = np.linspace(0, 5, 1000)
gamma1 = np.arange(1, 5)
gamma2 = np.arange(1, 5)
param = np.vstack((gamma1, gamma2))
param = param.T
for i in range(4):
    y = f.pdf(x, param[i][0], param[i][1])
    plt.plot(x, y, lw=2)

```

### 3.3 專題討論

在此章節中，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  $Y$  的機率質量函數圖。本題可以直接計算每個平均數的機率，但在此請使用隨機抽樣的方式，估計出這些機率值。其中抽樣的次數可以高至百萬以上。

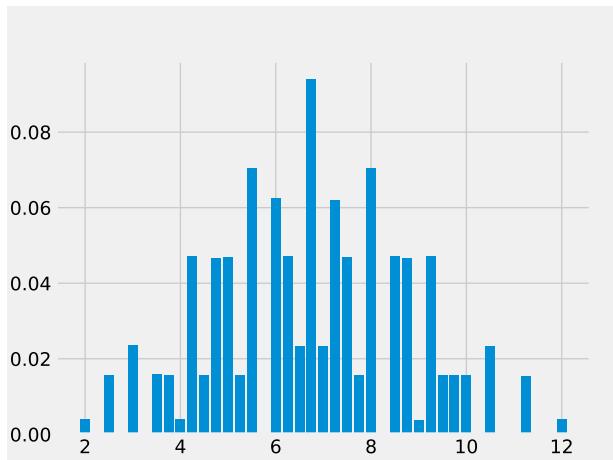


圖 3.37: 使用迴圈繪製取後放回直方圖

圖 3.37 為此抽樣的結果，有兩種方法可以繪製此直方圖，其中一種是使用迴圈，另一種則是使用函數 `np.random.choice(a, 4).choice()` 來進行抽樣，其部分程式碼亦呈現如下。

```
# for loop
np.random.seed(seed=1294)
x = np.array([2, 4, 9, 12])
number = 10000
y = np.zeros(number)
for i in np.arange(len(y)):
    rc = np.random.choice(x, 4, replace=True)
    y[i] = rc.mean()
weight = np.ones_like(y)/len(y)
plt.hist(y, bins=50, weights=weight)

# function
N = 1000000
a = [2, 4, 9, 12]
nr = np.random.choice(a, [N, 4], replace = True)
nr_mu = nr.mean(axis=1)
A = np.unique(nr_mu, return_counts=True)
allmeans = A[0]
meancount = A[1] / N
```

## 3.4 結論

在本章中，成功利用各種圖形來更深入了解各個常用的機率分配的特性，望以後在對分配的特性產生疑問時，能直接在此文章中得到解答，也能更快速的利用 Python 來理解其他本文未提及之分配的特性。



## 第 4 章

# 迴歸分析與羅吉斯迴歸的分類問題

在機器學習中，迴歸分析方法是最常見的方法之一，但大多應用於預測定量資料的情況，而在本文中，將利用一般資料集探討在反應變量是定性資料的情況下如何利用簡單線性迴歸模型 ( Simple Linear Regression ) 以及 ( Augmented Regression ) 來準確地進行二元分類，並利用各種模擬的資料來更好的比較兩種不同迴歸分析方法的效能。另外，本文也將探討如何利用羅吉斯迴歸 ( Logistic Regression ) 分類多類別的資料。

## 4.1 機器學習基礎理論

機器學習 ( Machine Learning ) 是一套用以理解數據的龐大工具，其工具主要分為兩大類，即監督式學習 ( Supervised Learning ) 與非監督式學習 ( Unsupervised Learning )。一般來說，監督式學習與非監督式學習有不同的用途：監督式學習一般用於建立預測統計模型，或者利用一個或多的輸入變量 ( input variable ) 來估計某個輸出變量 ( output variable )。而在非監督式學習中，則大多指定輸入變量卻不指定輸出變量。假設觀察到一個定量的輸出變量  $Y$  以及  $p$  個不同的輸入變量  $X_1, X_2, \dots, X_p$ ，若  $Y$  與  $X = (X_1, X_2, \dots, X_p)$  有關係，則可以將兩者寫作一個表達式：

$$Y = f(X) + \varepsilon \quad (4.1)$$

式 (4.1) 中的  $f$  是  $X_1, X_2, \dots, X_p$  的函數， $\varepsilon$  是均值為 0 且與  $X$  獨立的隨機誤差項。而機器學習就是用來估計  $f$  的一系列方法。

### 4.1.1 監督式學習與非監督式學習

接著在此節更詳細的介紹監督式學習與非監督式學習的差異。監督式學習 ( Supervised Learning ) 一般通過建立預測變量 ( Predictor variable ) 與反應變量 ( Response ) 之間的關

係，精準預測反應變量或更好的理解兩者之間的關係。許多傳統統計學習方法，比如線性迴歸 ( Linear Regression )、羅吉斯迴歸 ( Logistic Regression )，以及廣義可加模型 ( GAM )、提升法 ( Boosting ) 與支持向量機 ( SVM ) 都屬於監督式學習方法的一種。

相反，非監督式學習 ( Unsupervised Learning ) 更有挑戰性。在非監督式學習中，只有預測變量  $x_i, i = 1, 2 \dots n$  已知，且這些變量沒有相應的反應變量  $y_i$  對應。由於缺乏響應變量，因此在這類問題中建立線性模型是不可能的，因此往往利用非監督式學習來理解變量之間或者觀察值之間的關係，例如聚類分析 ( Cluster Analysis )。在本文中將聚焦在監督式學習中的分類方法 ( Classification )，因此不對非監督式學習進行過多闡述。

### 4.1.2 分類與迴歸問題

變量常分為定量 ( Quantitive ) 與定性 ( Qualitative ) 兩種，定量變量如年齡 ( Age )、銷售量 ( Sales ) 或者收入 ( Income )，定性變量則是類別變量，例如性別、產品的品牌、是否違約等等。一般在建立模型時會將響應變量為定性的問題歸類到分類問題 ( Classification )，而反應變量為定量的問題則歸類到迴歸問題。然而，迴歸問題與分類問題的區分並不是如此絕對，例如，羅吉斯迴歸 ( Logistic Regression ) 一般被認為是一種分類方法，但由於它估計了每個類別發生的機率，因此同樣被認為是一種迴歸問題；支持向量機 ( SVM ) 一般被用來進行二元分類或多元分類問題，但同樣可以利用支持向量機迴歸來預測定量資料得到很好的結果。大部分機器學習方法都可以應用在迴歸問題與分類問題中，在下節中，即將利用簡單線性迴歸來嘗試處理二元分類問題。

以下簡單的介紹分類問題中模型精度的計算方式。假設建模的目標是在  $(x_1, y_1), \dots (x_n, y_n)$  尋找對  $f$  的估計，其中  $y_1, \dots, y_n$  是定性變量。最常用的衡量估計  $\hat{f}$  準確度的方法是計算錯誤率 ( error rate )，也就是對訓練資料 ( Training Data ) 使用估計模型  $\hat{f}$  所造成的誤差比例，如式 (4.2) 所示：

$$\frac{1}{n} \sum_{i=1}^n I(y_i \neq \hat{y}_i) \quad (4.2)$$

其中  $\hat{y}_i$  是使用  $\hat{f}$  預測數據的第  $i$  個值，而如果  $I(y_i \neq \hat{y}_i) = 0$ ，那麼第  $i$  個觀測值用分類模型實現了正確分類，否則就是被誤分。

## 4.2 簡單線性迴歸模型

### 4.2.1 簡單線性迴歸理論

線性迴歸 (Linear Regression) 是一種常見的監督式學習方法，是一種常見的預測定量響應變量 (Response) 的工具，它假定了  $X$  與  $Y$  之間存在線性關係，在數學上將這種關係記為：

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \varepsilon_i \quad (4.3)$$

在上式 (4.3) 中假設存在兩個預測變量  $X_1$  與  $X_2$ ，假設有  $N$  筆已知的輸入與輸出資料 ( $[x_1(i) \ x_2(i)], y(i)$ )，利用最小平方法期望使觀察值與估計值的誤差越小越好。其計算方法如下矩陣：

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} = \begin{bmatrix} 1 & X_1(1) & X_2(1) \\ 1 & X_1(2) & X_2(2) \\ \vdots & & \\ 1 & X_1(n) & X_2(n) \end{bmatrix} \begin{bmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_n \end{bmatrix}$$

透過上述矩陣即可求得最佳解如式 (4.4)，式 (4.4) 假設反矩陣  $(X^T X)^{-1}$  存在，且每個輸出值  $y(i)$  根據其類別分類至 0 或 1。

$$\hat{\beta} = (X^T X)^{-1} X^T y \quad (4.4)$$

### 4.2.2 簡單線型迴歸分類方法

雖然迴歸分析一般應用於響應變量  $y$  是定量資料的情況，在定性資料中的表現普通，甚至只能用於二元資料分類，在本節中仍嘗試利用簡單線性迴歸方法來對二元資料進行分類模型的建立，以更了解迴歸分析方法的應用，二元資料包含響應變量為男生或女生、是否成功賣出產品等等。

利用迴歸分析方法來建立模型後，當得到新的預測變量  $(x_1, x_2)$  時，將首先得到  $y$  的估計值  $\hat{y} = x^T \hat{\beta}$ ，其中  $x^T = [1 \ x_1 \ x_2]$ 。在得到估計值後，若  $\hat{y} \leq 0.5$ ，則將該筆資料分類至第一類，若  $\hat{y} > 0.5$ ，則將該筆資料分類至第二類。

利用迴歸分析為二元資料建模並進行預測時會遵循以下步驟：

1. 繪製散布圖：透過繪圖了解兩個類別資料的關係。
  - 利用 *scatter* 函數進行畫圖。
  - 先將資料分成二群再分別進行畫圖。
2. 參數估計：
  - 寫程式做最小平方法估計參數並建立迴歸模型。
  - 利用 sklearn 套件直接建立迴歸模型。
3. 畫分界線：利用配適好的迴歸模型繪製分界線。
4. 算準確率：計算模型繪製出的分界線的分類準確率。

### 4.2.3 簡單線性迴歸分類範例

#### 繪製散布圖

在本節中，將利用資料集 *la\_2.txt* 來嘗試建立迴歸模型進行類別分類。該資料為  $200 * 4$  的資料集，包含兩組不同類別的觀察值以及他們的類別 0 ( Group A ) 與 1 ( Group B )。其散布圖如圖 4.1。

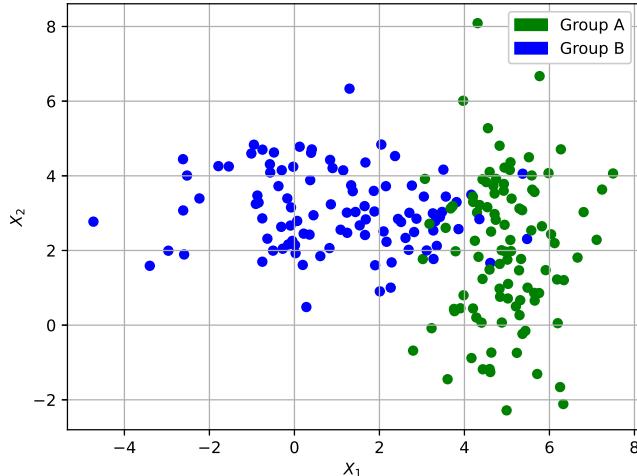


圖 4.1：資料集 *la\_2.txt* 的散布圖

由圖 4.1 可知，類別 0 ( Group A ) 的觀察值的變異相較於類別 1 ( Group B ) 較小，觀察點較為密集，而類別 1 ( Group B ) 的資料點變異較大，資料點較為分散。另外，兩組資料之間重疊的點似乎並不算多。

```

data_dir = "D:/vscodepython/Statistical Calculation/Homework4_
Regression/"
D = np.loadtxt(data_dir + "la_2.txt", comments="#")

```

在此問題中利用上面的程式碼將資料讀入 Python。如前面所述，一般有兩種常用的方法可以用來繪製散布圖，其一是利用 *scatter* 畫圖，其二則是先將資料分開再繪製散布圖。以下呈現兩種方法的部分程式碼：

```

##利用scatter函數畫圖##
colors = ["green" if i == 0 else "blue" for i in D[:,2]]
#colors = [[1,0,0] if i == 0 else [0,0,1] for i in D[:,2]]
#[R,G,B]

plt.scatter(D[:, 0], D[:, 1], c = colors, s = s, marker = "o",
            alpha = 0.5)

##個別分群後畫圖##
Idx = (D[:,2]==0)
plt.plot(D[Idx, 0], D[Idx, 1], "ro", alpha = 0.5, label =
         "Group A")
Idx = (D[:,2]==1)
plt.plot(D[Idx,0], D[Idx,1],"bo", alpha = 0.5, label =
         "Group B")

```

## 參數估計

在利用散布圖觀察完資料簡單的特性後，接著將進行參數估計並建立迴歸模型，一般同樣有兩種常用的方法，第一種是利用程式計算最小平方法所估計出的參數  $\hat{\beta} = (X^T X)^{-1} X^T y$ ，並利用此估計參數建立迴歸模型，其部分程式碼呈現如下：

```

n = len(D[:, 0])
X = np.c_[np.ones(n), D[:, 0:2]] #不會再橫豎不分
y = D[:, 2]
b = LA.inv(X.T @ X) @ X.T @ y
#@:矩陣相乘 #inv:inverse

```

另外也可以直接利用 Python 的 *sklearn* 套件來建立迴歸模型，其程式碼如下：

```

Mdl = LinearRegression()
X = D[:, 0:2]
y = D[:, 2]
n = len(y)

```

```

## 配適模型
Mdl.fit(X, y)
## R-squared
R2 = Mdl.score(X, y)

## 截距項
intrcpt = Mdl.intercept_
## 係數
coeffs = Mdl.coef_

```

畫分界線並算出準確率

建立迴歸模型後，接著畫令  $y = 0.5$  時的點  $(x_1, x_2)$  所形成的分界線，即令：

$$0.5 = \beta_0 + \beta_1 X_1 + \beta_2 X_2 \quad (4.5)$$

接著將該式 (4.5) 轉化成式 (4.6)，以  $X_1 = (-3, 5)$  代入求出分界線，並透過圖 4.2 觀察其分類準確度，若是迴歸模型預測的值大於 0.5，則分類至群組 1 (Group B)，若是小於 0.5，則分類至群組 0 (Group A)。透過計算可知，此迴歸模型的準確度為 91%。

$$X_2 = \frac{-(\beta_0 - 0.5 + \beta_1 X_1 + \beta_2 X_2)}{\beta_2} \quad (4.6)$$

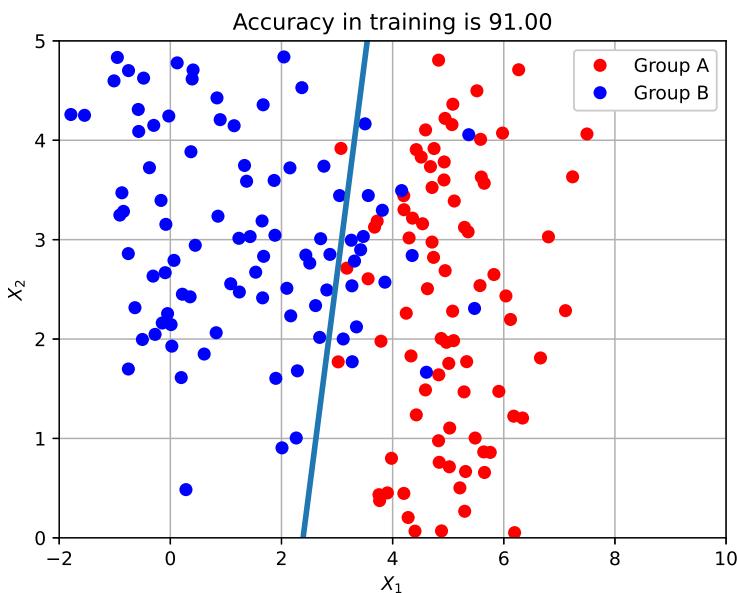


圖 4.2: 資料集 *la\_2.txt* 之簡單線性迴歸模型

### 4.3 加廣迴歸模型

在利用簡單線性迴歸模型求得圖 4.2 後可知，Group A 與 Group B 的點似乎重合度較高，用簡單線性迴歸來配適模型似乎並不是一個好方法，因此可嘗試重新配適一個非線性的迴歸模型以得到較好的分類準確度，故重新假設其迴歸模型如式 (4.8)：

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_1 X_2 + \beta_4 X_1^2 + \beta_5 X_2^2 \quad (4.7)$$

接著與配適簡單線性迴歸 ( Simple Linear Regression ) 相同，先利用 sklearn 套件進行參數估計建立模型，再畫出分界線算出模型分類準確率，其結果與簡單線性迴歸模型相比如下圖 4.3：

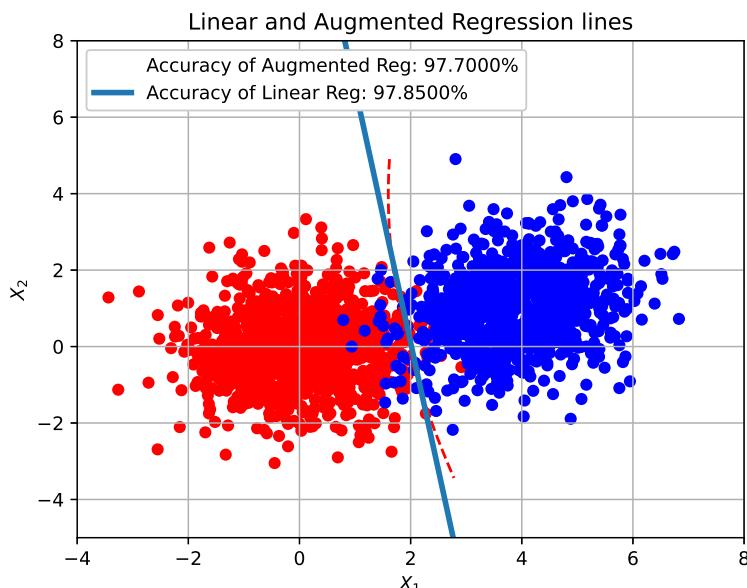


圖 4.3: 資料集 *la\_2.txt* 之簡單線性迴歸與加廣線性迴歸模型

由圖 4.3 可知，利用加廣迴歸模型來進行分類得到的準確度為 97.7%，反而比利用簡單線性迴歸來分類得到的準確度 97.85% 來的低，其部分繪製圖形程式碼呈現如下：

```
x1 = D[:, 0:1]
x2 = D[:, 1:2]
X = np.hstack((x1, x2, x1 * x2, x1 ** 2, x2 ** 2))
y = D[:, 2]
n = len(y)
Mdl = LinearRegression()
Mdl.fit(X, y) # 進行估計 ( 配適 )
intrcpt = Mdl.intercept_
```

```

coeffs = Mdl.coef_
y_hat = Mdl.predict(X)

y_pre = [1 if i > 0.5 else 0 for i in y_hat]
x = np.array([-3, 5])
f =
lambda x: intrcp
+ coeffs[0] * x[0]
+ coeffs[1] * x[1]
+ coeffs[2] * x[0] * x[1]
+ coeffs[3] * x[0] ** 2
+ coeffs[4] * x[1] ** 2)
xx = np.linspace(x1.min(), x1.max(), 100)
yy = np.linspace(x1.min(), x2.max(), 100)
X, Y = np.meshgrid(xx, yy) #網格:100*100的網格
Z = f([X, Y])
contours = plt.contour(
X, Y, Z, levels = [0.5], colors="red", linestyles="--", lw=3)

labels = ["Accuracy of Augmented Reg: {:.4f} %".format(100 * np.
mean(y_pre == y))]
for i in range(len(labels)):
    contours.collections[i].set_label(labels[i])

```

## 4.4 簡單線性迴歸與加廣迴歸模型比較

為了更好的了解簡單線性迴歸 (Simple Linear Regression) 以及加廣迴歸模型 (Augmented Regression) 的分類準確度，在此節中將模擬六組不同的資料，將其切割成 80% 的訓練資料集 (Training Data) 以及 20% 的測試資料集 (Testing Data) 並觀察其準確度。為了方便寫程式，將資料與資料的類別各自進行命名，分割資料集的程式碼呈現如下：

```

train_data, test_data, train_label, test_label = train_test_
_split(data, label, test_size=0.2)

```

接著模擬六組不同的相依雙變量常態母體來觀察兩種迴歸模型方法的分類效能。

- 兩組模擬資料樣本數大小改變

在此比較中，將分別設樣本數為  $n_1 = 200, n_2 = 200$  以及  $n_1 = n_2 = 1000$ ，並設置兩雙變量常態的變數各自如下矩陣，其結果則呈現如圖 4.4。

$$\mu_1 = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \mu_2 = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \Sigma_1 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \Sigma_2 = \begin{bmatrix} 1 & 0.2 \\ 0.2 & 1 \end{bmatrix},$$

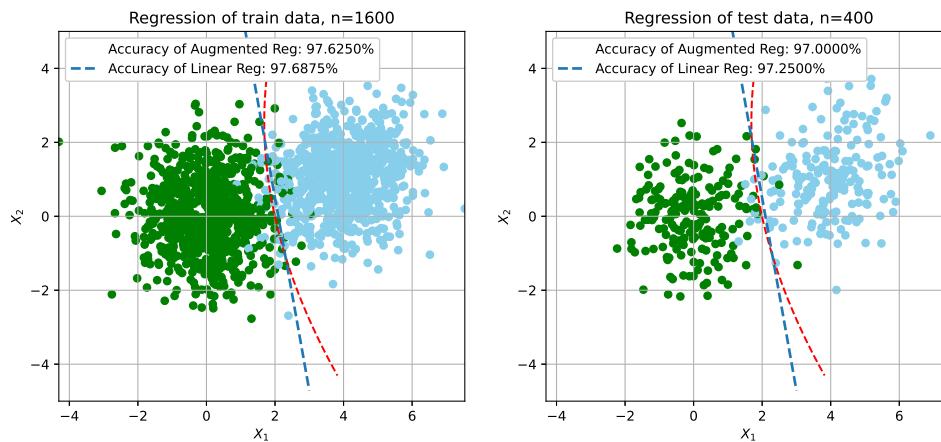


圖 4.4: 樣本數為 1000 進行簡單線性迴歸與加廣迴歸的訓練與測試資料準確度

由圖 4.4 可知，訓練資料集有 1600 筆資料，測試資料集有 400 筆資料，且簡單線性迴歸在訓練資料與測試資料集的準確度都高於加廣迴歸模型。接著將兩組資料的樣本數皆調整至  $n_1 = n_2 = 200$ 。由圖 4.5 可知，簡單線性迴歸的準確度仍高於加廣迴歸模型。以下亦呈現其部分程式碼。

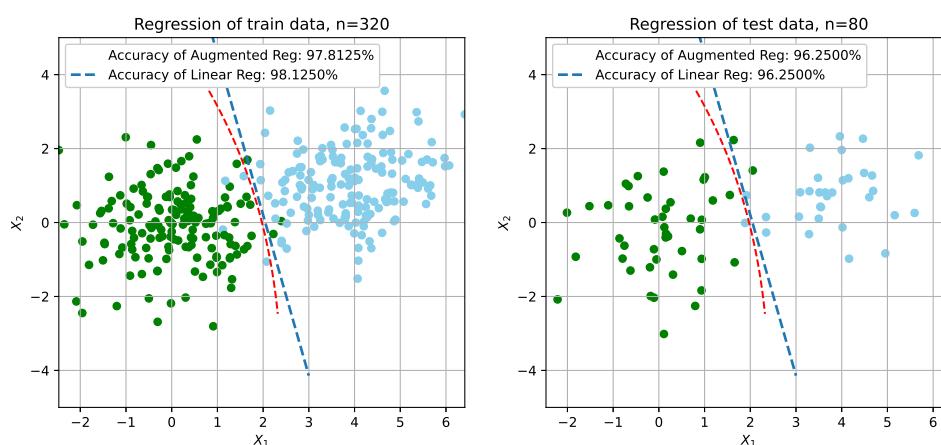


圖 4.5: 樣本數為 200 進行簡單線性迴歸與加廣迴歸的訓練與測試資料準確度

```
from scipy.stats import multivariate_normal
n1, n2 = 1000, 1000
m1, m2 = np.array([3, 3]), np.array([2, 4])
a, b = 0.4, 0.5
```

```

Cov1 = np.array([[1, a], [a, 1]])
Cov2 = np.array([[1, b], [b, 1]])

## 兩組多變量常態資料
mvn1 = multivariate_normal(mean = m1, cov = Cov1)
mvn2 = multivariate_normal(mean = m2, cov = Cov2)
## 多變量常態隨機變數
A, B = mvn1.rvs(n1), mvn2.rvs(n2)
## 資料矩陣
D = np.vstack((A, B))
## 群組值0 or 1
y = np.hstack((np.zeros(n1), np.ones(n2)))
np.savetxt("demo_data.txt", np.c_[D, y], fmt = "% .4f % .4f
           %d")
# 存下資料

data_dir = "D:/vscodepython/Statistical Calculation/"
D = np.loadtxt(data_dir + "demo_data.txt", comments="%")

data = D[:, 0:2]
label = D[:, 2]

# train:1600 test:400
train_data, test_data, train_label, test_label = train_
test_split(data, label, test_size=0.2)

##### Augmented Regression
x1 = train_data[:, 0:1]
x2 = train_data[:, 1:2]
X = np.hstack((x1, x2, x1 * x2, x1 ** 2, x2 ** 2))
y = train_label
n = len(y)

```

- 兩組模擬資料平均數改變

接著嘗試改變兩雙變量常態母體的平均數來觀察分類準確度，我們設樣本數皆為  $n_1 = n_2 = 1000$ ，且其各自的參數為：

$$\mu_1 = \begin{bmatrix} 2 \\ 4 \end{bmatrix}, \mu_2 = \begin{bmatrix} 5 \\ 3 \end{bmatrix}, \Sigma_1 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \Sigma_2 = \begin{bmatrix} 1 & 0.2 \\ 0.2 & 1 \end{bmatrix},$$

其結果如圖 4.6：

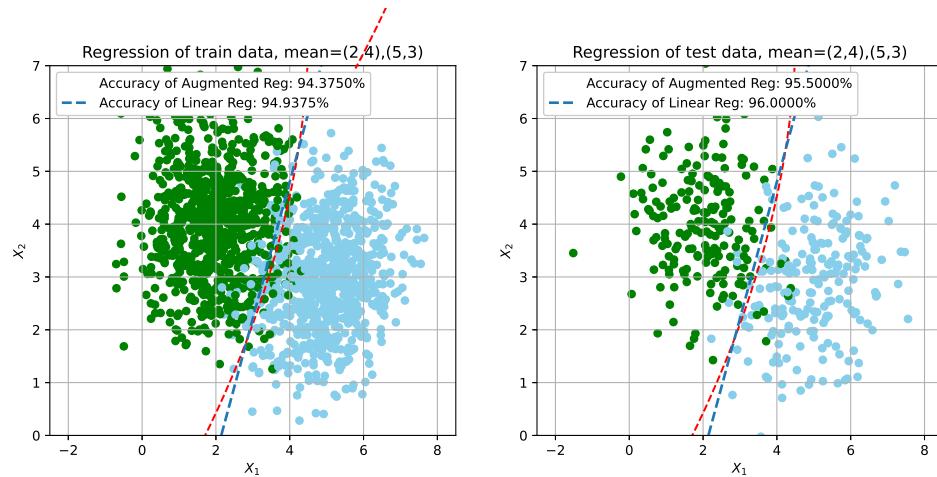


圖 4.6: 群平均為  $(2, 4)$  與  $(5, 3)$  進行簡單線性迴歸與加廣迴歸的訓練與測試資料準確度

由圖 4.6 可知，簡單線性迴歸在訓練資料集與測試資料集表現仍稍微優於加廣迴歸模型，接著將母體參數修正為：

$$\mu_1 = \begin{bmatrix} 4 \\ 4 \end{bmatrix}, \mu_2 = \begin{bmatrix} 6 \\ 5 \end{bmatrix}, \Sigma_1 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \Sigma_2 = \begin{bmatrix} 1 & 0.2 \\ 0.2 & 1 \end{bmatrix},$$

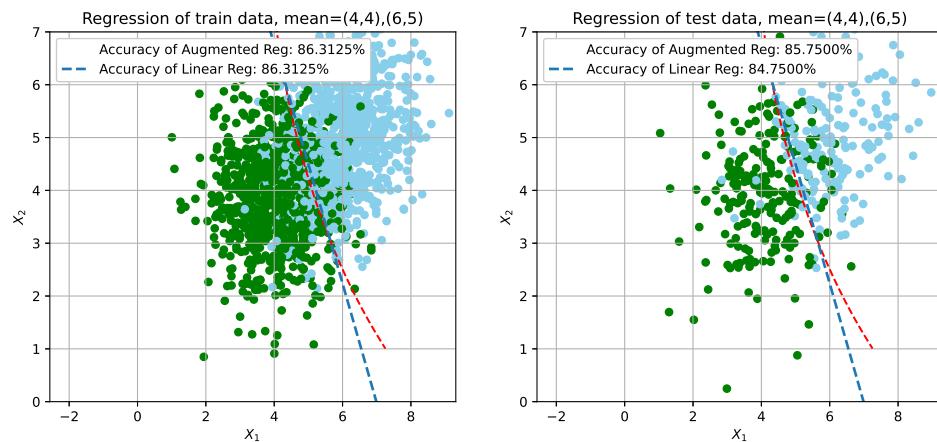


圖 4.7: 群平均為  $(4, 4)$  與  $(6, 5)$  進行簡單線性迴歸與加廣迴歸的訓練與測試資料準確度

由圖 4.7 可知，在此時資料比起圖 4.6 更加貼近，加廣迴歸模型在訓練資料集的準確度與簡單線性迴歸相同，但其在測試資料集的準確度略高於肩擔線性迴歸模型。

- 兩組模擬資料變異數改變

維持樣本數皆為  $n_1 = n_2 = 1000$ ，並更改共變異數矩陣中的參數，其參數如下：

$$\mu_1 = \begin{bmatrix} 3 \\ 3 \end{bmatrix}, \mu_2 = \begin{bmatrix} 2 \\ 4 \end{bmatrix}, \Sigma_1 = \begin{bmatrix} 1 & 0.4 \\ 0.4 & 1 \end{bmatrix}, \Sigma_2 = \begin{bmatrix} 1 & 0.5 \\ 0.5 & 1 \end{bmatrix},$$

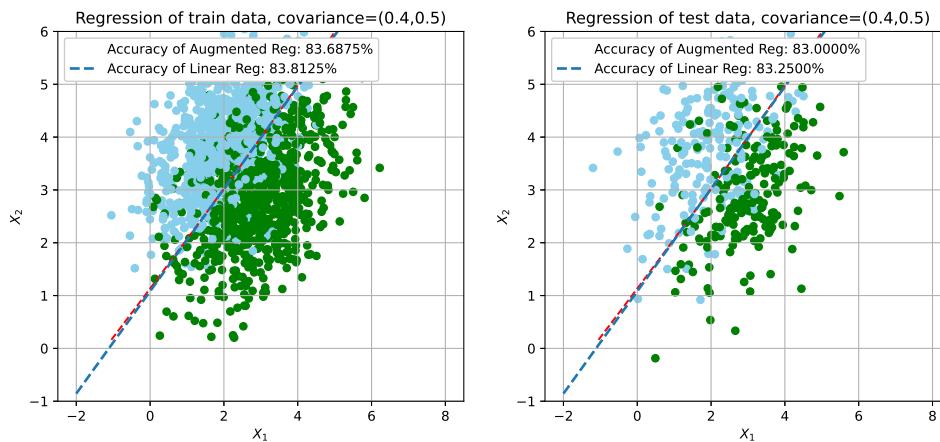


圖 4.8：群共變異數為 0.4 與 0.5 進行簡單線性迴歸與加廣迴歸的訓練與測試資料準確度

由圖 4.8 可知，兩筆資料的重合度偏高時，兩種方法的準確度都下降到 83% 左右，且基本擬合出的分界線一模一樣，只是簡單線性迴歸的準確度略高一點點，基本可以忽略不計。

接著再變更參數進行觀察一次，我們將母體參數更改如下：

$$\mu_1 = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \mu_2 = \begin{bmatrix} 3 \\ 3 \end{bmatrix}, \Sigma_1 = \begin{bmatrix} 1 & 0.9 \\ 0.9 & 1 \end{bmatrix}, \Sigma_2 = \begin{bmatrix} 1 & 0.2 \\ 0.2 & 1 \end{bmatrix},$$

根據圖 4.9 可知，在兩筆資料重合部分，且其中一筆資料散布較廣、點也較密集時（綠色點），加廣迴歸模型在訓練資料集的準確度略高於簡單線性迴歸，測試資料集的準確度則略低於簡單線性迴歸模型。

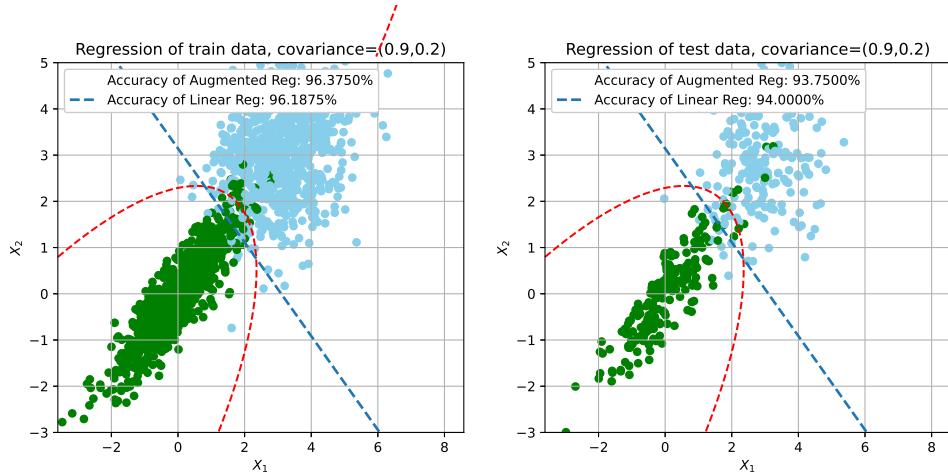


圖 4.9: 群共變異數為 0.9 與 0.2 進行簡單線性迴歸與加廣迴歸的訓練與測試資料準確度

## 4.5 羅吉斯迴歸模型

### 4.5.1 羅吉斯迴歸理論

在上文中探討了利用簡單線性迴歸與加廣迴歸模型來進行二元分類的各種例子，但其實在機器學習的分類問題中，較多的使用羅吉斯迴歸 ( Logistic Regression ) 來進行分類。假設存在兩個類別來解釋羅吉斯迴歸的簡單原理。在線性迴歸中利用  $p(x) = \beta_0 + \beta_1 X_1 + \beta_2 X_2$  來表示機率，而在羅吉斯迴歸中，則使用羅吉斯函數式 (4.8) 結合最大概似估計 ( MLE ) 方法來擬合模型。

$$p(X) = \frac{e^{\beta_0 + \beta_1 X_1 + \beta_2 X_2}}{1 + e^{\beta_0 + \beta_1 X_1 + \beta_2 X_2}} \quad (4.8)$$

接著將式 (4.8) 改寫成式 (4.9)，並將其稱之為勝算比 ( odd )。

$$\frac{p(X)}{1 - p(X)} = e^{\beta_0 + \beta_1 X_1 + \beta_2 X_2} \quad (4.9)$$

若將式 (4.9) 取  $\log$ ，則將其稱為對數勝算比 ( log-odd )，寫作式 (4.10)

$$\log\left(\frac{p(X)}{1 - p(X)}\right) = \beta_0 + \beta_1 X_1 + \beta_2 X_2 \quad (4.10)$$

### 4.5.2 羅吉斯迴歸分類範例

接著直接模擬各種不同參數的三筆相依常態母體，並觀察羅吉斯迴歸的分類準確度。

首先假定樣本數為  $n_1 = n_2 = n_3 = 1000$ ，且其參數為：

$$\mu_1 = \begin{bmatrix} 1 \\ 5 \end{bmatrix}, \mu_2 = \begin{bmatrix} 3 \\ 2 \end{bmatrix}, \mu_3 = \begin{bmatrix} 4 \\ 4 \end{bmatrix}$$

$$\Sigma_1 = \begin{bmatrix} 1 & 0.9 \\ 0.9 & 1 \end{bmatrix}, \Sigma_2 = \begin{bmatrix} 1 & 0.2 \\ 0.2 & 1 \end{bmatrix}, \Sigma_3 = \begin{bmatrix} 1 & 0.5 \\ 0.5 & 1 \end{bmatrix}$$

根據圖 4.10 可知，此三筆資料羅吉斯迴歸 ( Logistic Regression ) 的分類準確度為 89.73%，以下亦呈現部分程式碼。

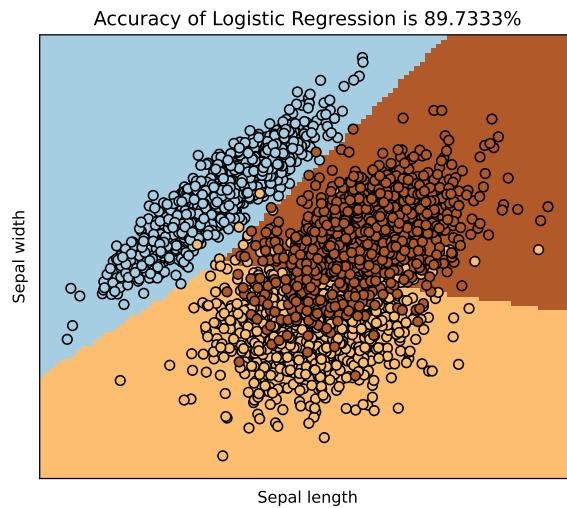


圖 4.10: 第一個羅吉斯迴歸分類實驗分類準確度

```
logreg = LogisticRegression(multi_class='multinomial', solver='lbfgs', class_weight='balanced', max_iter=1000)
logreg.fit(X, Y)
y_hat = logreg.predict(X), score = logreg.score(X, Y)
y_pro = logreg.predict_proba(X), y_predict = logreg.predict(X)
print("Accuracy in logistic regression: {:.4f}%".format(100 * np.mean(y_predict == Y)))

_, ax = plt.subplots(figsize=(6, 5))
DecisionBoundaryDisplay.from_estimator(
```

```

logreg,
X,
cmap=plt.cm.Paired,
ax=ax,
response_method="predict",
plot_method="pcolormesh",
shading="auto",
xlabel="Sepal length",
ylabel="Sepal width",
eps=0.5,
)

```

接著模擬各種不同參數的三筆相依常態母體，並觀察羅吉斯迴歸的分類準確度。假定樣本數為  $n_1 = n_2 = n_3 = 1000$ ，且其參數為：

$$\mu_1 = \begin{bmatrix} 5 \\ 5 \end{bmatrix}, \mu_2 = \begin{bmatrix} 7 \\ 1 \end{bmatrix}, \mu_3 = \begin{bmatrix} 3 \\ 6 \end{bmatrix}$$

$$\Sigma_1 = \begin{bmatrix} 1 & 0.4 \\ 0.4 & 1 \end{bmatrix}, \Sigma_2 = \begin{bmatrix} 1 & 0.2 \\ 0.2 & 1 \end{bmatrix}, \Sigma_3 = \begin{bmatrix} 1 & 0.5 \\ 0.5 & 1 \end{bmatrix}$$

根據圖 4.11 可知，此三筆資料利用羅吉斯迴歸 ( Logistic Regression ) 分類的分類準確度為 94.43%。

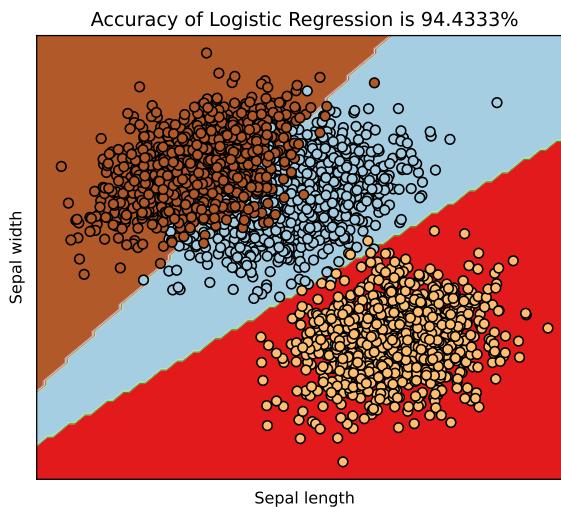


圖 4.11：第二個羅吉斯迴歸分類實驗分類準確度

接著繼續模擬各種不同參數的三筆相依常態母體，並觀察羅吉斯迴歸的分類準確度。

假定樣本數為  $n_1 = n_2 = n_3 = 1000$ ，且其參數為：

$$\mu_1 = \begin{bmatrix} 5 \\ 5 \end{bmatrix}, \mu_2 = \begin{bmatrix} 7 \\ 7 \end{bmatrix}, \mu_3 = \begin{bmatrix} 9 \\ 1 \end{bmatrix}$$

$$\Sigma_1 = \begin{bmatrix} 1 & 0.4 \\ 0.4 & 1 \end{bmatrix}, \Sigma_2 = \begin{bmatrix} 1 & 0.2 \\ 0.2 & 1 \end{bmatrix}, \Sigma_3 = \begin{bmatrix} 1 & 0.5 \\ 0.5 & 1 \end{bmatrix}$$

根據圖 4.12 可知，此三筆資料利用羅吉斯迴歸分類的準確度為 92.83%。

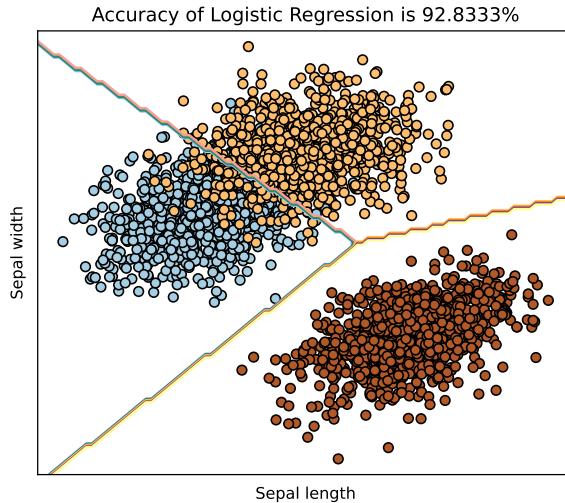


圖 4.12：第三個羅吉斯迴歸分類實驗之分類準確度

接著再模擬各種不同參數的三筆相依常態母體，並觀察羅吉斯迴歸的分類準確度。假定樣本數為  $n_1 = n_2 = n_3 = 1000$ ，且其參數為：

$$\mu_1 = \begin{bmatrix} 5 \\ 5 \end{bmatrix}, \mu_2 = \begin{bmatrix} 9 \\ 9 \end{bmatrix}, \mu_3 = \begin{bmatrix} 5 \\ 4 \end{bmatrix}$$

$$\Sigma_1 = \begin{bmatrix} 1 & 0.9 \\ 0.9 & 1 \end{bmatrix}, \Sigma_2 = \begin{bmatrix} 1 & 0.6 \\ 0.6 & 1 \end{bmatrix}, \Sigma_3 = \begin{bmatrix} 1 & 0.8 \\ 0.8 & 1 \end{bmatrix}$$

根據圖 4.13 可知，此三筆資料利用羅吉斯迴歸分類的準確度為 87%。

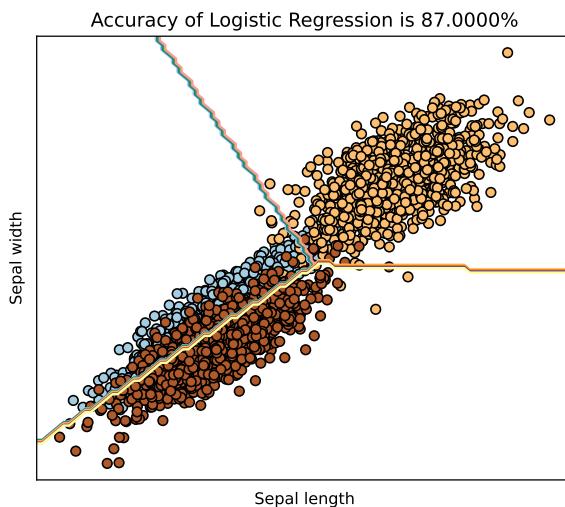


圖 4.13: 第四個羅吉斯迴歸分類實驗之分類準確度

## 4.6 結論

在本章節中介紹了利用簡單線性迴歸以及加廣迴歸模型來執行二元分類可以發現，迴歸分析在處理分類問題時也可以得到很不錯的準確度，但它並不能應用在多元分類以上的問題。另外，與原本預想加廣迴歸模型的分類準確度都會比簡單線性迴歸高不同，在一般的問題上簡單線性迴歸的表現甚至比加廣迴歸模型略好一點。除此之外，羅吉斯迴歸在各種模擬資料的表現都很不錯，依據羅吉斯迴歸的理論，原本預想羅吉斯迴歸只能處理二元分類問題，但透過本文可以了解其如何利用 sklearn 套件進行多元分類以及繪圖。



## 第 5 章

# 判別分析與 K-近鄰演算法的分類問題

在機器學習中，存在有很多種方法能對資料進行分類並建立模型，包含僅能利用於二元資料的迴歸分析方法 ( Regression Analysis )、常用的羅吉斯迴歸 ( Logistic Regression ) 與本節即將介紹的三種方法-線性判別分析 ( Linear Discriminant Analysis )、二次判別分析 ( Quadratic Discriminant Analysis ) 以及 K-近鄰演算法 ( K-Nearest Neighbors )。在本節中將依次闡述線性判別分析、二次判別分析與 K-近鄰演算法的理論並展示各自如何針對二群與三群的類別資料進行分類，以及各自的訓練誤差與測試誤差，其中，本節將採用 Bootstrapping 方法來計算各自的平均訓練誤差與測試誤差。最後，本節將對三種方法的分類效能進行比較，期望能對三種分類方法有更深入的了解。

## 5.1 線性判別分析、二次判別分析與 K-近鄰演算法

### 5.1.1 線性判別分析

在線性判別分析 ( Linear Discriminant Analysis ) 中，試圖找出預測變數  $X$  在不同的反應變數  $G$  ( 不同類別) 的分配，並利用貝氏定理去估計  $P(G = k|X = x)$ 。假設先驗分配 ( prior probability ) ( $P(G = k)$ ) 代表隨機從類別  $k$  抽出觀察值的機率，而後驗分配  $f_k(x) = P(X = x|G = k)$  代表觀察值來自類別  $k$  時  $X$  的機率密度函數，則透過貝氏定理可知：

$$P(G = k|X = x) = \frac{P(G = k)P(X = x|G = k)}{\sum_{l=1}^k P(G = l)P(x = X|g = l)} \quad (5.1)$$

其中，若使後驗分配  $P(X = x|G = k)$  最大，則此分類的分類誤差 ( error rate ) 最少。

接著首先描述  $p = 1$  的情況，即存在一個預測變數  $X$  時，線性判別分析 ( LDA ) 如何有

效的進行二元分類。假設  $f_k(x) = P(X = x|G = k)$  來自常態母體 (Normal) 或多變量常態母體 (Multivariate Normal)，其機率密度函數為：

$$f_k(x) = \frac{1}{\sqrt{2\pi}\sigma_k} \exp\left(-\frac{1}{2\sigma_k^2}(x - \mu_k)^2\right) \quad (5.2)$$

其中  $\mu_k$  與  $\sigma_k^2$  為第  $k$  類的平均數與變異數。在線性判別分析 (LDA) 中會假設  $\sigma_1^2 = \dots = \sigma_k^2$ ，故由此可得：

$$P(G = k|X = x) = \frac{P(G = k) \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{1}{2\sigma^2}(x - \mu_k)^2\right)}{\sum_{l=1}^k P(G = l) \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{1}{2\sigma^2}(x - \mu_l)^2\right)} \quad (5.3)$$

接著，將式 (5.3) 取  $\log$ ，即可得到式 (5.4)：

$$\delta_k(x) = x \frac{\mu_k}{\sigma^2} - \frac{\mu_k^2}{2\sigma^2} + \log(P(G = k)) \quad (5.4)$$

由式 (5.4) 可知，若  $2x(\mu_1 - \mu_2) > \mu_1^2 - \mu_2^2$ ，則將該筆觀察值分類至第一群 (其平均數為  $m_i$ )，且此分類的分界線如式 (5.5)：

$$x = \frac{\mu_1^2 - \mu_2^2}{2(\mu_1 - \mu_2)} = \frac{\mu_1 + \mu_2}{2} \quad (5.5)$$

在線性判別分析中假設了預測變數  $X$  的每個分類皆是來自於常態母體或者是多變量常態母體，但在實務中往往並不知道其母體服從何種分配。而在此練習中，假設預測變數  $X$  的每個類別都來自常態母體去估計母體參數  $\mu_1, \dots, \mu_k, \pi_1, \dots, \pi_k$  以及  $\sigma^2$ ，得到式 (5.6)、式 (5.7) 以及式 (5.8) 之估計量。

$$\hat{\mu}_k = \frac{1}{n_k} \sum_{i:y_i=k} x_i \quad (5.6)$$

$$\hat{\sigma}^2 = \frac{1}{n - K} \sum_{k=1}^K \sum_{i:y_i=k} (x_i - \hat{\mu}_k)^2 \quad (5.7)$$

$$\hat{\pi}_k = n_k/n \quad (5.8)$$

其中  $n$  為所有訓練資料集的觀察值的數量、 $n_k$  為第  $k$  個類別中訓練資料集的觀察值數量，而  $\hat{\sigma}^2$  可被視為  $K$  個類別的樣本變異數的加權平均。另外，若缺少了  $\pi_1, \dots, \pi_k$  任何一個值，則線性判別分析 (LDA) 會利用式 (5.8) 的  $\hat{\pi}_k$  來估計  $\pi_k$ 。

在介紹完  $p = 1$ ，即預測變數  $X$  只有一個時線性判別分析 (LDA) 的理論後，接著將線性判別分析擴增至  $p > 1$  的情況下進行討論，即預測變數服從多變量常態分配母體 (Multivariate Normal Distribution) 的情況，另外，在線性判別分析中皆假設所有類別的共變異數相等。為了進一步了解多變量常態母體資料，首先先模擬雙變量常態資料 ( $p = 2$ ) 並繪製多變量常態母體的散布圖來了解其基本性質，假設平均數為  $(0, 0)$ ，而共變異數矩陣各為：

$$\Sigma_1 = \begin{bmatrix} 1 & -0.8 \\ -0.8 & 1 \end{bmatrix}, \Sigma_2 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \Sigma_3 = \begin{bmatrix} 1 & 0.8 \\ 0.8 & 1 \end{bmatrix},$$

則多變量常態分配的圖形如圖 5.1，其部分程式碼亦呈現如下。

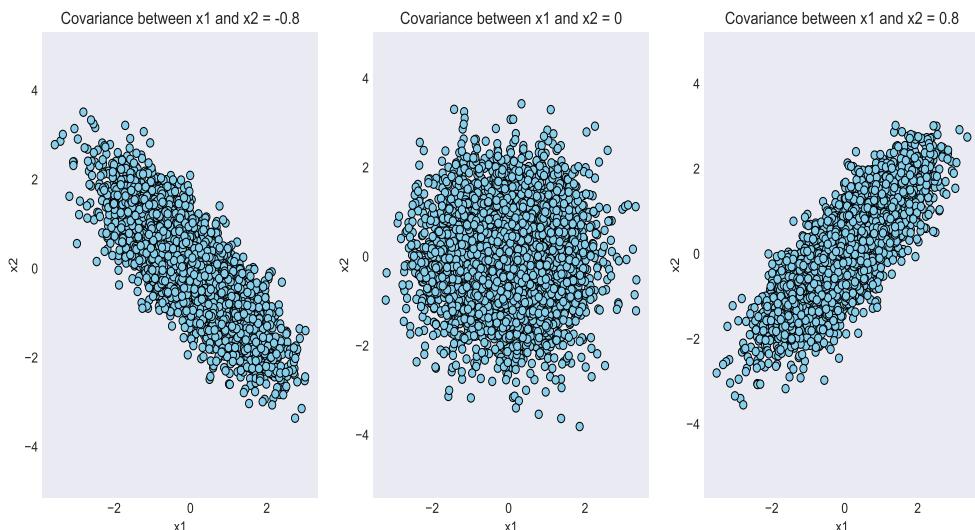


圖 5.1: 多變量常態母體資料之 2D 圖像

```
plt.style.use('seaborn-dark')
plt.rcParams['figure.figsize']=14, 6
fig = plt.figure()

random_seed=1000
cov_val = [-0.8, 0, 0.8]
mean = np.array([0, 0])
```

```

for idx, val in enumerate(cov_val):
    plt.subplot(1, 3, idx+1)
    cov = np.array([[1, val], [val, 1]])

    distr = multivariate_normal(cov = cov, mean = mean,
                                 seed = random_seed)
    data = distr.rvs(size = 5000)

    plt.plot(data[:,0],data[:,1], 'o', c='red',
              markeredgewidth = 0.5,
              markeredgecolor = 'black')

```

透過圖 5.1 難以觀察到兩多變量常態分配的關係究竟為何，因此嘗試將圖形繪製成 3D 圖形來重新觀察兩變量的關係，其結果如圖 5.2，可以明顯看出兩變數  $x_1$  與  $x_2$  的關係。其部分程式碼亦呈現如下。

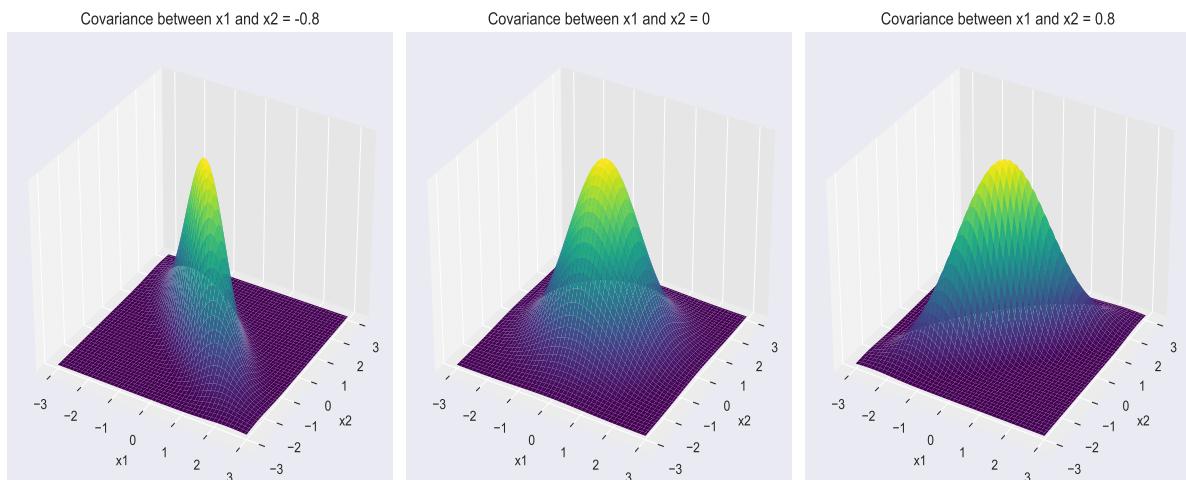


圖 5.2: 多變量常態母體資料之 3D 圖像

```

pdf_list = []
for idx, val in enumerate(cov_val):
    cov = np.array([[1, val], [val, 1]])
    distr = multivariate_normal(cov = cov, mean = mean,
                                 seed = random_seed)

    mean_1, mean_2 = mean[0], mean[1]
    sigma_1, sigma_2 = cov[0,0], cov[1,1]

    x = np.linspace(-3*sigma_1, 3*sigma_1, num=100)

```

```

y = np.linspace(-3*sigma_2, 3*sigma_2, num=100)
X, Y = np.meshgrid(x,y)

pdf = np.zeros(X.shape)
for i in range(X.shape[0]):
    for j in range(X.shape[1]):
        pdf[i,j] = distr.pdf([X[i,j], Y[i,j]])

key = 131+idx
ax = fig.add_subplot(key, projection = '3d')
ax.plot_surface(X, Y, pdf, cmap = 'viridis')
pdf_list.append(pdf)
ax.axes.zaxis.set_ticks([])

```

接著詳細討論線性判別分析 ( LDA ) 在母體為多變量常態分配 ( Multivariate Normal Distribution ) 時的理論方法。假設  $X \sim N(\mu, \Sigma)$ ，且其機率密度函數如式 (5.9)：

$$f(x) = \frac{1}{(2\pi)^{p/2} |\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(x - \mu)^T \Sigma^{-1} (x - \mu)\right) \quad (5.9)$$

而線性判別分析 ( LDA ) 會將觀察值  $X = x$  分類至使式 (5.10) 的值最大的第  $k$  類。

$$\delta_k(x) = x^T \Sigma^{-1} \mu_k - \frac{1}{2} \mu_k^T \Sigma^{-1} \mu_k + \log P(G = k) \quad (5.10)$$

透過以上式 (5.10) 即可得到線性判別分析的分群結果，而在繪製圖形時，若欲繪製線性判別分析的群組分界線，則將繪製屬於群組  $k$  和屬於群組  $l$  的機率相同的地方，即繪製式 (5.11) 成立的函數。

$$P(G = k | X = x) = P(G = l | X = x) \quad (5.11)$$

亦可將上式 (5.11) 取  $\log$  轉換成如式 (5.12)

$$\begin{aligned}
\ln \frac{P(G = k | X = x)}{P(G = l | X = x)} &= \ln \frac{f_k(x)}{f_l(x)} + \ln \frac{P(G = k)}{P(G = l)} \\
&= \ln \frac{P(G = k)}{P(G = l)} - \frac{1}{2} (\mu_k + \mu_l)^T \Sigma^{-1} (\mu_k - \mu_l) \\
&\quad + x^T \Sigma^{-1} (\mu_k - \mu_l) \\
&= 0
\end{aligned} \quad (5.12)$$

### 5.1.2 線性判別分析分類效果

為了理解線性判別分析 (LDA) 的分類效果，在此利用 Python 來更深入了解其分群的效能與方法，其部分程式碼呈現如下。

```
C1, C2 = X[y==0,:], X[y==1,:]

n = D[:,0].size
n1, n2 = C1[:,0].size, C2[:,0].size
pi1, pi2 = n1/n, n2/n
mu1, mu2 = np.mean(C1, axis = 0), np.mean(C2, axis = 0)
Sigma = (np.cov(C1.T) + np.cov(C2.T))/2

K = np.log(pi1/pi2) - 0.5 * (mu1 + mu2) \
@ LA.inv(Sigma) @ (mu1 - mu2).T
L = LA.inv(Sigma) @ (mu1 - mu2).T
f = lambda x : -L[0]/L[1] * x - K/L[1]
x = np.linspace(1, 5, 10)
plt.plot(x, f(x))
```

$$K = \ln \frac{P(G=1)}{P(G=2)} - \frac{1}{2}(\mu_k + \mu_l)^T \Sigma^{-1} (\mu_k - \mu_l) \quad (5.13)$$

設置  $C1, C2$  分別是資料集中屬於類別 0 以及屬於類別 1 的資料，並根據式 (5.6)、式 (5.7) 以及式 (5.8) 對兩變量的參數進行估計得到各自的  $\hat{\mu}_k, \hat{\sigma}^2$  以及  $\hat{\pi}_k$ 。程式碼中的  $K$  則來自於式 (5.12)，由於此範例是式 (5.12) 雙變量的情況，故將式 (5.12) 改寫成如式 (5.13)，並設其中之  $\Sigma^{-1}(\mu_k - \mu_l)$  為式 (5.14)：

$$L = \Sigma^{-1}(\mu_k - \mu_l) \quad (5.14)$$

$$K + L(1)x_1 + L(2)x_2 = 0 \quad (5.15)$$

透過以上之方式，可以將函式改寫成如式 (5.15)，接著即可以利用程式碼 `plt.plot(x, f(x))` 求出如圖 5.3 之分界線。

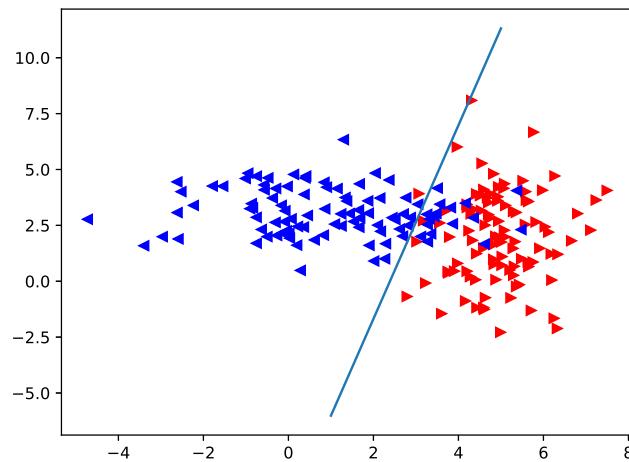


圖 5.3: 線性判別分析之二元分類範例

### 5.1.3 二次判別分析

在線性判別分析中假設母體服從常態分配且每個類別的共變異數相同（共變異矩陣一樣），但在實務上往往很難滿足此假設，而二次判別分析中完善了此一缺點。二次判別分析假設了每個類別都有各自的共變異數，因此式 (5.10) 可被改寫成如式 (5.16)：

$$\begin{aligned}\delta_k(x) &= -\frac{1}{2}(x - \mu_k)^T \Sigma_k^{-1} (x - \mu_k) - \frac{1}{2} \ln |\Sigma_k| + \ln P(G = k) \\ &= -\frac{1}{2}x^T \Sigma_k^{-1} x + x^T \Sigma_k^{-1} \mu_k - \frac{1}{2} \mu_k^T \Sigma_k^{-1} \mu_k - \frac{1}{2} \ln |\Sigma_k| + \ln P(G = k)\end{aligned}\quad (5.16)$$

其中，二次判別分析 (QDA) 需估計  $\Sigma_k$ ,  $\mu_k$  以及  $P(G = k)$  並將觀察值  $X = x$  分類至使式 (5.16) 的值最大的類別  $k$ 。

另外，二次判別分析的分界線為變數的二次方，被寫作如式 (5.17)

$$\{x | \delta_k(x) = \delta_l(x)\} \quad (5.17)$$

### 5.1.4 線性判別分析與二次判別分析比較

在了解線性判別分析與二次判別分析的理論後，嘗試利用 Python 實際進行操作並繪圖以更了解如何利用此兩種機器學習方法進行分群，以下列出基本的分群步驟：

- 繪製散布圖

首先利用以下程式碼繪製資料集 *la2.txt* 中兩群資料的散布圖，其中利用函數 *np.random.randint* 來產生小於 50、大小為該資料集大小的整數。

```
X = D[:, 0:2]
y = D[:, 2]
fig, ax = plt.subplots(figsize=(8, 6))
area = 2 * np.random.randint(50, size = D[:, 0].size)

grp_color = [[1,0,0] if i == 0 else [0,0,1] for i in y]

plt.scatter(D[:, 0], D[:, 1], c = grp_color, s = area,
            alpha = 0.5, marker = "o" )
```

- 訓練線性判別分析 (LDA) 與二次判別分析 (QDA) 模型

接著利用以下程式碼來分別訓練 LDA 模型以及 QDA 模型，並且利用函數 *Lda.score* 與 *Qda.score* 來計算兩種方法各自的訓練誤差。

```
Lda = LinearDiscriminantAnalysis(tol = 1e-6)
Lda.fit(X, y)
trainErrLDA = 1 - Lda.score(X, y)

Qda = QuadraticDiscriminantAnalysis(tol = 1e-6, store_
    covariance = True)
Qda.fit(X, y)
trainErrQDA = 1 - Qda.score(X, y)
```

- 將畫布切成網格以備進行繪圖

以下即是相關程式碼，利用 *x* 以及 *y* 來將畫布切割成網格，並利用 *np.meshgrid* 來產生 *xx* 與 *yy* 的矩陣，即每一個網格交錯點的 *x* 座標以及 *y* 座標。

```
nx, ny = 100, 100
x_min, x_max = plt.xlim()
y_min, y_max = plt.ylim()
x_ = np.linspace(x_min, x_max, nx)
y_ = np.linspace(y_min, y_max, ny)
xx, yy = np.meshgrid(x_, y_)
```

- 計算後驗機率

接著利用以下程式碼計算每一個網格交錯點座標的機率，即  $P(G = K|X = x)$ ，為了進行計算，會先利用函數 `ravel()` 將 `xx` 與 `yy` 從矩陣拉成一整條以進行 broadcasting ( 拉成一條線才能運算，矩陣則不能運行)。在計算好機率後，再利用函數 `reshape` 將 `Z` 中第一欄屬於群組 0 的機率取出並重新變回矩陣。

```
Z = Lda.predict_proba(np.c_[xx.ravel(), yy.ravel()])
Z = Qda.predict_proba(np.c_[xx.ravel(), yy.ravel()])
Z = Z[:, 1].reshape(xx.shape)
```

- 定義繪圖色系並繪製地板分界線

在此介紹兩種繪圖的方法，第一種是如果不想利用系統預設之顏色，也可以利用套件 `colors` 中的函數 `LinearSegmentColormap` 來定義自己喜歡的顏色，並利用函數 `pcolormesh` 來繪製地板分界線，結果如圖 5.4，其部分程式碼呈現如下。

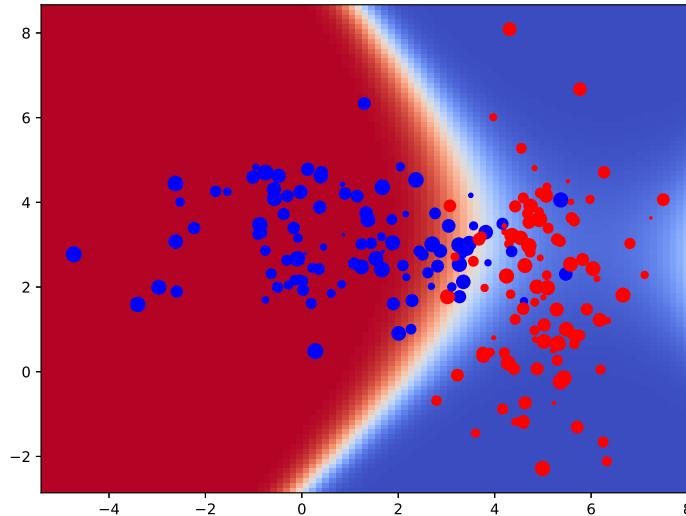


圖 5.4: 線性判別分析之二元分類圖

```
cdit = {"red": [(0, 1, 1), (1, 0.7, 0.7)],
        "green": [(0, 0.7, 0.7), (1, 0.7, 0.7)],
        "blue": [(0, 0.7, 0.7), (1, 1, 1)]}
cmap = colors.LinearSegmentedColormap("coolwarm", cdit)
plt.pcolormesh(xx, yy, Z, cmap = "coolwarm",
               norm = colors.Normalize(0., 1.),
               shading = "auto", zorder = 0)
```

第二種方式是將畫布切割成網格並對每個交叉點的座標進行預測，以下面的程式碼為例，將畫布切割出  $200 * 100$  筆資料點，預測每一個點並將結果放在變數  $Z$  中，最後將這些預測結果分開並繪製散布圖，即形成地板分界線，其結果如圖 5.5。

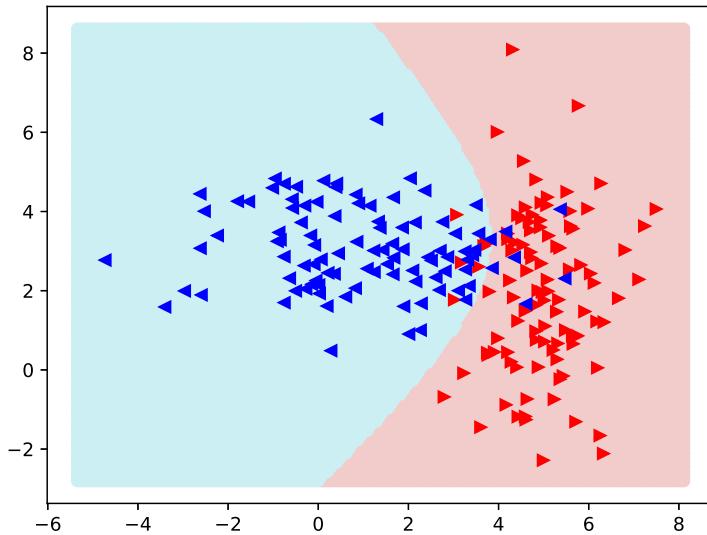


圖 5.5: 第一種線性判別分析與二次判別分析之二元分類圖

```

nx, ny = 200, 100
x_min, x_max = plt.xlim(), y_min, y_max = plt.ylim()
x_ = np.linspace(x_min, x_max, nx), y_ = np.linspace(y_min,
, y_max, ny)
xx, yy = np.meshgrid(x_, y_)
x1, x2 = xx.ravel(), yy.ravel()
zz = Qda.predict(np.c_[x1, x2])

colors = ["#F2CFCB", "#CBEFF2"]
for i in range(2) :
    plt.scatter(x1[zz==i], x2[zz==i], marker="o", color=
    colors[i])

```

- 繪製分界線並比較訓練誤差

在繪製地板分界線後，接著利用以下程式碼進行繪製分類在群組 1 的機率為 0.5 的分界線，如圖 5.6，另外由此亦可知，線性判別分析 (LDA) 的訓練誤差為 0.09，而二次判別分析 (QDA) 的訓練誤差則為 0.065，故此例中二次判別分析 (QDA) 的分類效果比線性判別分析 (LDA) 還要好。

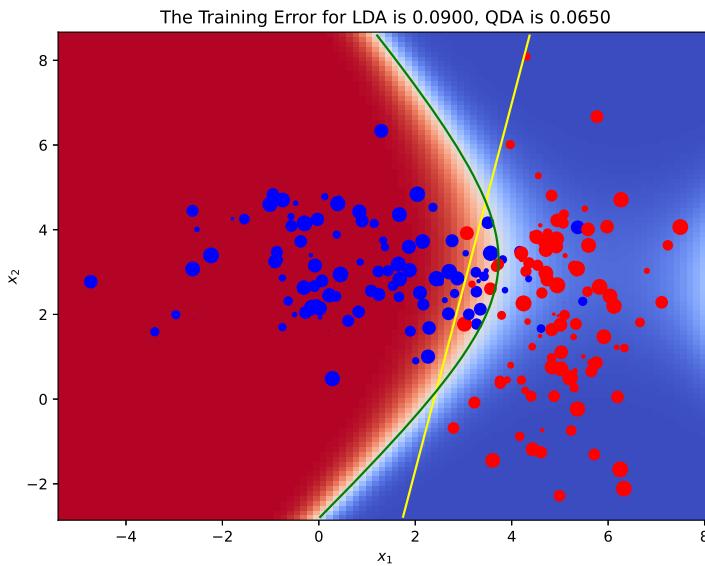


圖 5.6: 第二種線性判別分析與二次判別分析之二元分類圖

```

contoursLDA = plt.contour(xx, yy, z, [0.5], colors = "yellow")
contoursQDA = plt.contour(xx, yy, z, [0.5], colors = "green")

```

### 5.1.5 K-近鄰演算法

雖然在機器學習中有幾種常見的方法是利用貝氏定理來求得條件機率  $P(Y|X = x)$ ，其中  $x$  與  $y$  分別是反應變數與預測變數，但在真實世界中，很難知道此條件機率的值，因此利用貝氏定理來分類有時是不可行的，而有一些方法便嘗試去估計  $P(Y|X = x)$ ，其中包括 K-近鄰演算法 (KNN)。K-近鄰演算法透過給定  $K$  值以及任一測試點  $x_0$ ，觀察離該點  $x_0$  最近的  $K$  個點 (即  $N_k(x)$ )，並將這些鄰近的  $K$  筆資料所對應的  $y$  值取平均，得到式 (5.18)：

$$\hat{y} = \text{Ave}(y_i | x_i \in N_k(x)) = \frac{1}{K} \sum_{x_i \in N_k(x)} y_i \quad (5.18)$$

且其給定  $x$  時  $y$  的條件機率如式 (5.19)：

$$P(Y = j | X = x_0) = \frac{1}{K} \sum_{i \in N_0} I(y_i = j) \quad (5.19)$$

### 5.1.6 K-近鄰演算法分類效果

大致了解 K-近鄰演算法 (KNN) 的理論後，接著一樣透過繪圖來了解其分群的效果到底如何，與線性判別分析 (LDA) 相同，KNN 一樣可以有兩種方式來配適模型，第一種是從理論來著手，第二種則是直接使用 sklearn 套件中 *neighbors* 的函數 *KNeighborsClassifier* 來配適模型，而這兩種方法都將在之後分別進行展示。

- 繪製散布圖

在繪製圖形時，首先利用下面的程式碼就資料集 *la3.txt* 的兩個變數繪製散布圖，此處嘗試利用套件 seaborn 來進行繪製，以利後續之繪圖。

```
cmap_bold = ["darkblue", "darkorange"]
Group_name = np.array(["Group A", "Group B"])
plt.figure(figsize=(8, 6))

sns.scatterplot(x = X[:, 0], y = X[:, 1], hue = Group_name
[y], palette = cmap_bold, alpha = 0.9, edgecolor = "
black")
```

- 配適 KNN 模型

一般有兩種方法用以配適 KNN 模型，此處首先先展示如何利用理論來進行配適。假設  $K = 15$ ，與線性判別分析 (LDA) 相同，利用函數 *meshgrid* 將畫布切成間距為 0.2、大小比照兩變數極大值與極小值的網格，並建立一個 for 迴圈，設立變數 *tmp* 儲存將網格的值拉成向量後再沿 y 軸複製 n 倍的資料集，每一次迴圈都會得到一個  $200 * 2$  的矩陣。接著根據理論，計算觀察點  $x_0$  到每一個點的距離得到變數 *d*，並利用 *np.mean(y[idx[:K]])* 計算距離最近的  $K = 15$  個點的均值，若此均值小於 0.5 則歸類在類別 1，若大於 0.5 則歸類在類別 2，由此即可得到該點所屬的類別。

```
K = 15
intrvl = 0.2
x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, intrvl), np.
arange(y_min, y_max, 0.1))

z = np.zeros(xx.size)
for i in range(xx.size) :
    tmp = np.tile([xx.ravel()[i], yy.ravel()[i]], (n, 1))
```

```
#d = ((tmp-X)**2).sum(axis=1) #兩種都可以
d = np.linalg.norm(tmp - X, axis = 1)
idx = np.argsort(d)
z[i] = np.mean(y[idx[:K]])

z = [0 if i < 0.5 else 1 for i in z]
```

在配適完成 KNN 模型後，即可利用函數 `sns.scatterplot` 繪製每個網格點的類別散布圖當作地板分界線，並利用函數 `plt.contour` 繪製分界線，得到圖 5.7 之結果。

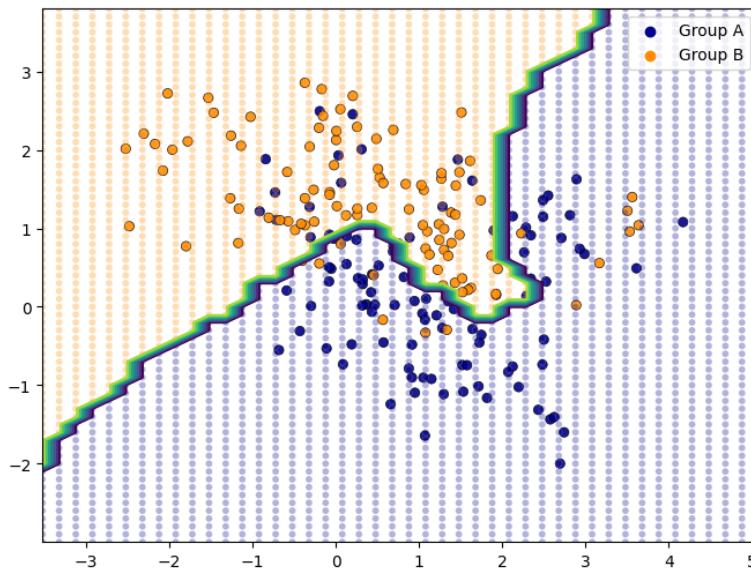


圖 5.7: 利用理論進行 K-近鄰演算法之二元分類圖

第二種方法是利用 `sklearn` 套件來配適模型，如下方之程式碼。KNN 與線性判別分析 (LDA) 與二次判別分析 (QDA) 相同，可以利用函數 `Knn.score` 來得到 KNN 模型的訓練誤差，並建立網格來預測每個網格點座標所屬之類別，最後利用函數 `plt.contourf` 得到如圖 5.8 之分類結果。此模型的訓練誤差為 0.155。

```
K = 15, weights = "uniform"
Knn = neighbors.KNeighborsClassifier(K, weights = weights)
Knn.fit(X, y)
trainingErr = 1 - Knn.score(X, y)
x_min, x_max = X[:,0].min() - 1, X[:,0].max() + 1
y_min, y_max = X[:,1].min() - 1, X[:,1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.1), \
np.arange(y_min, y_max, 0.1))
z = Knn.predict(np.c_[xx.ravel(), yy.ravel()])
Z = z.reshape(xx.shape)
```

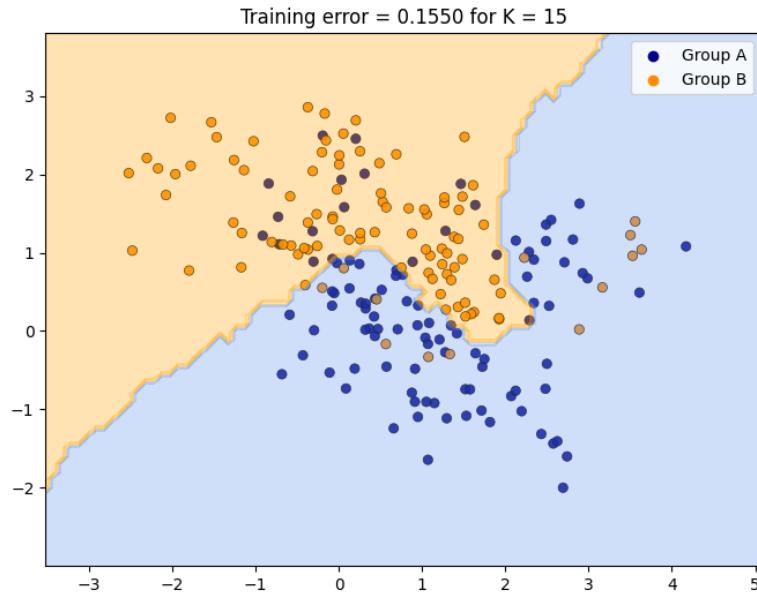


圖 5.8: 利用 sklearn 進行 K-近鄰演算法之二元分類圖

```
cmap_light = ListedColormap(["cornflowerblue", "orange"])
plt.contourf(xx, yy, Z, cmap = cmap_light, alpha = 0.3)
```

為了更好的理解如何設定  $K$  值以得到最佳的 KNN 分類模型，接著將分別模擬兩群、三群的雙變量常態資料，並觀察在  $k = 1$  到  $k = 30$  之間，利用 Bootstrapping 方法重複重樣 100 次所得到的平均訓練誤差以及平均測試誤差的變化。首先介紹分為兩群的雙變量常態資料，在此範例中，將原始資料集切割成 80% 的訓練資料集以及 20% 的測試資料集，且兩個變量的模擬樣本數設為  $n_1 = n_2 = 1000$ ，並設置各變量的平均數、樣本數與共變異數分別為：

$$\mu_1 = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \mu_2 = \begin{bmatrix} 4 \\ 2 \end{bmatrix}$$

$$\Sigma_1 = \begin{bmatrix} 1 & 0.2 \\ 0.2 & 1 \end{bmatrix}, \Sigma_2 = \begin{bmatrix} 1 & 0.5 \\ 0.5 & 1 \end{bmatrix}$$

為了協助更好的理解此雙變量常態資料，將分別對此雙變量常態資料繪製 2D 散布圖以及 3D 的圖形，其結果如圖 5.9 以及圖 5.10。

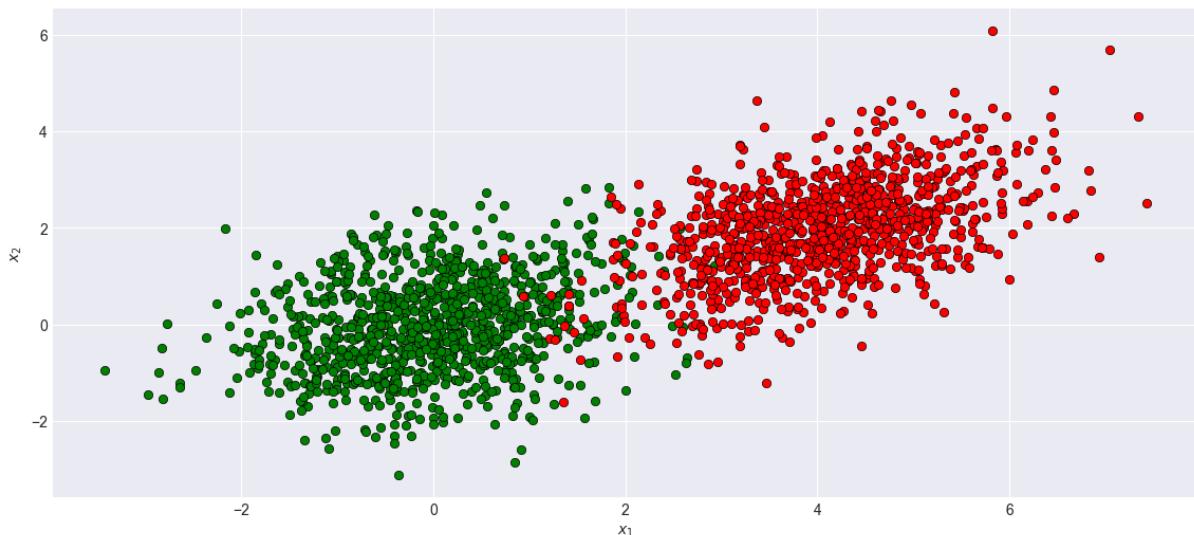


圖 5.9: K-近鄰演算法之兩群組雙變量常態資料之 2D 圖

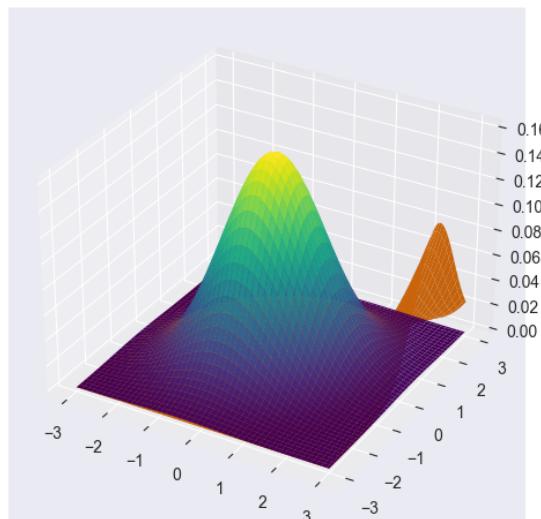


圖 5.10: K-近鄰演算法之兩群組雙變量常態資料之 3D 圖

透過圖 5.9 以及圖 5.10 可以明顯看出兩生成之模擬資料分配的不同之處。圖 5.9 中綠色的散布點是來自於平均數為  $\mu_1$  且共變異數為  $\Sigma_1$  的雙變量常態資料，紅色的散布點則來自於平均數為  $\mu_2$  且共變異數為  $\Sigma_2$  的雙變量常態資料。圖 5.10 中橘色的部分來自於平均數為  $\mu_2$  且共變異數為  $\Sigma_2$  的雙變量常態資料，另一個則是來自於平均數為  $\mu_1$  且共變異數為  $\Sigma_1$  的雙變量常態資料。

接著介紹生成的三群雙變量常態資料，其平均數與變異數分別為：

$$\mu'_1 = \begin{bmatrix} 2 \\ 2 \end{bmatrix}, \mu'_2 = \begin{bmatrix} 4 \\ 1 \end{bmatrix}, \mu'_3 = \begin{bmatrix} 6 \\ 3 \end{bmatrix}$$

$$\Sigma'_1 = \begin{bmatrix} 1 & 0.2 \\ 0.2 & 1 \end{bmatrix}, \Sigma'_2 = \begin{bmatrix} 1 & 0.5 \\ 0.5 & 1 \end{bmatrix}, \Sigma'_3 = \begin{bmatrix} 1 & 0.8 \\ 0.8 & 1 \end{bmatrix}$$

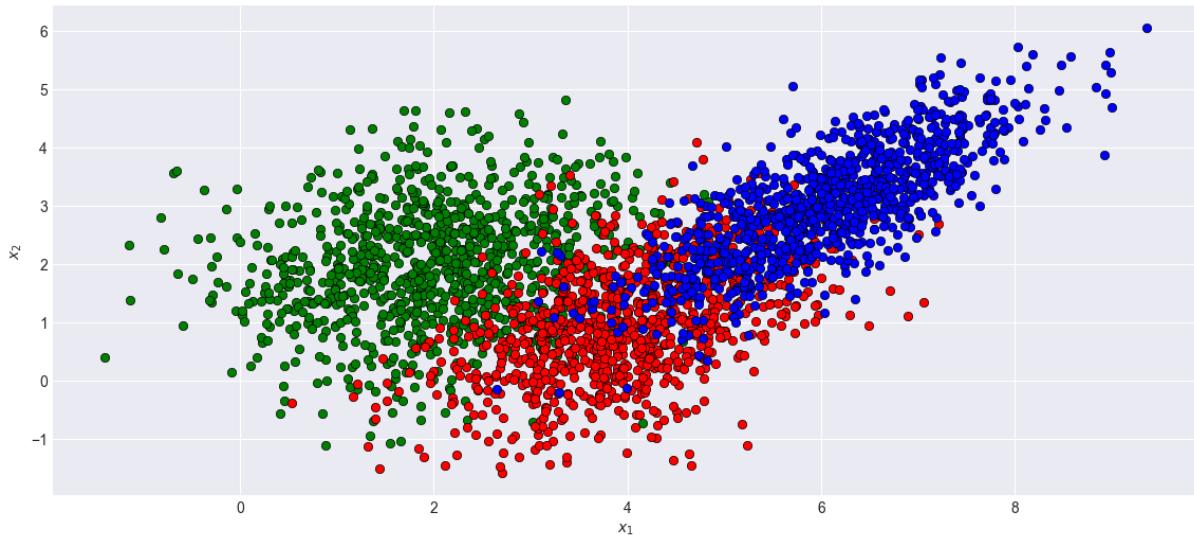


圖 5.11: K-近鄰演算法之三群組雙變量常態資料之 2D 圖

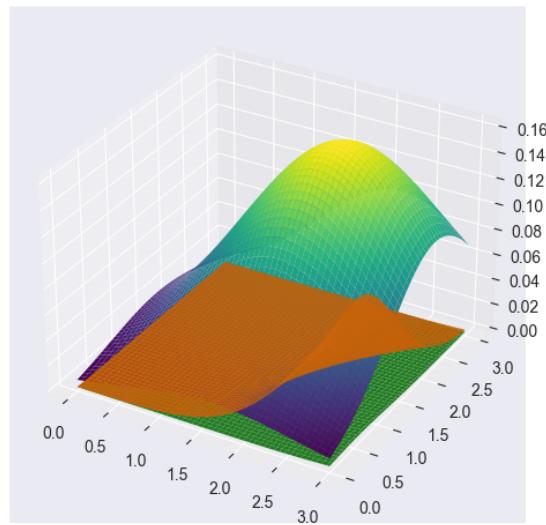


圖 5.12: K-近鄰演算法之三群組雙變量常態資料之 3D 圖

圖 5.11 是三群雙變量常態資料的散布圖，綠色點是平均數為  $\mu'_1$ 、共變異數為  $\Sigma'_1$  的雙變量常態資料，紅色點是平均數為  $\mu'_2$ 、共變異數為  $\Sigma'_2$  的雙變量常態資料，藍色點則為參數為  $\mu'_3$  與  $\Sigma'_3$  的雙變量常態資料。由圖 5.11 與圖 5.12 可以看出，在 2D 圖中很多點

會混在一起，似乎難以找到適合的分界線來分開三群不同類別的資料點，但如果畫成 3D 圖就能較為清楚的看出三組資料的差異。以下呈現部分繪製 3D 圖的程式碼。

```
plt.style.use('seaborn-dark')
plt.rcParams['figure.figsize']=14, 6
fig = plt.figure()
sigma_1, sigma_2 = Cov1[0,0], Cov1[1,1]
x = np.linspace(-3*sigma_1, 3*sigma_1, num=100)
y = np.linspace(-3*sigma_2, 3*sigma_2, num=100)
X, Y = np.meshgrid(x,y)

pdf = np.zeros(X.shape)
for i in range(X.shape[0]):
    for j in range(X.shape[1]):
        pdf[i,j] = mvn1.pdf([X[i,j], Y[i,j]])
key = 131
ax = fig.add_subplot(key, projection = '3d')
ax.plot_surface(X, Y, pdf, cmap = 'viridis')
plt.tight_layout()
```

接著利用  $\mu_1$ 、 $\mu_2$ 、 $\Sigma_1$  與  $\Sigma_2$  來對兩群雙變量常態資料配適 KNN 模型，並利用  $\mu'_1$ 、 $\mu'_2$ 、 $\mu'_3$ 、 $\Sigma'_1$ 、 $\Sigma'_2$  與  $\Sigma'_3$  對三群雙變量常態資料配適 KNN 模型。以下面的程式碼為例，建立兩個全為 0 的集合  $Errortrain$  與  $Errortest$  來裝  $K = 1$  到  $K = 30$  時 KNN 模型 100 次模擬的平均訓練誤差與平均測試誤差，另外，利用全為 0 的集合  $KNNtrainingError$  以及  $KNNtestingError$  來存取 100 次 Bootstrapping 抽樣所得到的 100 個訓練誤差與測試誤差，最後分別得到如下圖 5.13 之結果。



圖 5.13: 設置不同 K 值之 K-近鄰演算法的兩組與三組資料的訓練與測試資料分類誤差

根據圖 5.13 可知，在對兩群雙變量常態資料配適分類模型時， $K = 13$ 、 $K = 19$  與  $K = 30$  的測試誤差是 30 個 KNN 模型中最小的三個，而在三群雙變量常態資料中， $K = 2$ 、 $K = 7$  與  $K = 16$  則能得到較好的結果，其程式碼呈現如下。此處有個可改進之處是此迴圈在兩群雙變量資料花了 3 分鐘的時間才跑完，在此三群雙變量資料的分群中更是花了 10 分鐘，故有待研究更快速更有效的方法來求得此結果。

```
N = 100
K = 30
Errortrain = np.zeros(K)
Errortest = np.zeros(K)
weights = "uniform"
for j in range(K):
    Knn = neighbors.KNeighborsClassifier(K, weights = weights)
    KNNtrainingError = np.zeros(N)
    KNNtestingError = np.zeros(N)
    for i in range(N) :
        X_train, X_test, y_train, y_test = train_test_split(X,
                                                y, test_size = 0.2)
        Knn.fit(X_train, y_train)
        KNNtrainingError[i] = 1 - Knn.score(X_train, y_train)
        Knn.predict(X_test)
        KNNtestingError[i] = 1 - Knn.score(X_test, y_test)
    Errortrain[j] = KNNtrainingError.mean()
    Errortest[j] = KNNtestingError.mean()
```

最後嘗試繪製利用 KNN 模型 ( $K = 13$ ) 與 KNN 模型 ( $K = 30$ ) 分類兩群雙變量資料的分界線圖，以及利用 KNN 模型 ( $K = 2$ ) 與 KNN 模型 ( $K = 16$ ) 分類此三群雙變量資料的分界線圖，其結果如下圖 5.14 與 5.15。

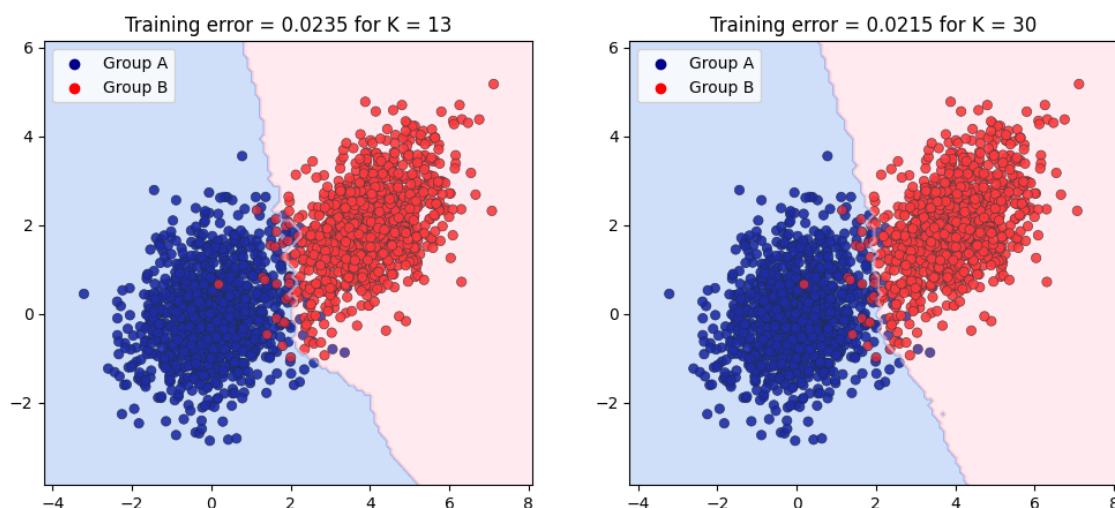


圖 5.14: K-近鄰演算法之兩群組分類圖

根據圖 5.14 可知 KNN 模型 ( $K = 13$ ) 在此模擬之兩群雙變量常態資料的分類訓練誤差為 0.0235，在  $K = 30$  時的訓練誤差為 0.0215。接著看在三群雙變量常態資料時 KNN 模型的分類效果。

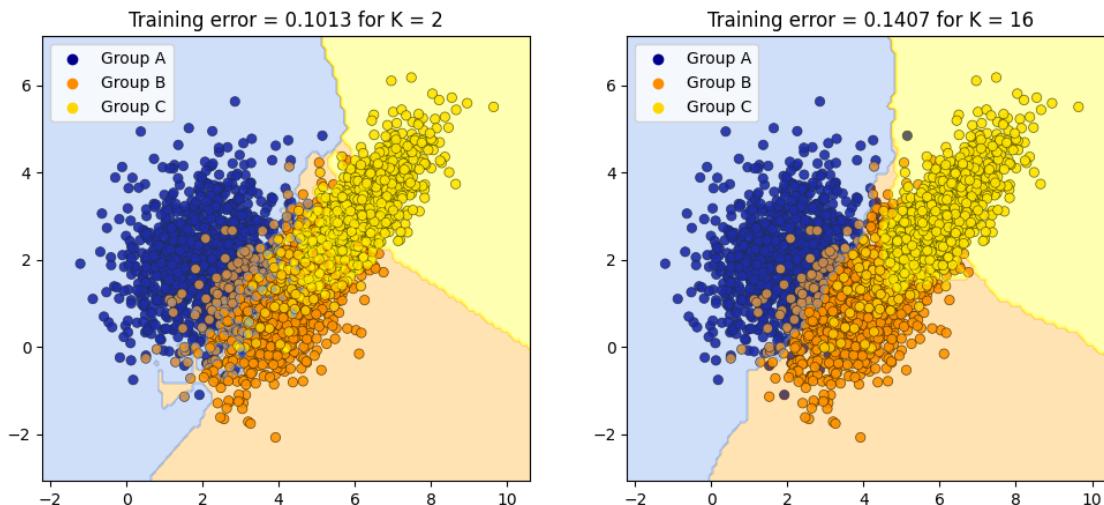


圖 5.15: K-近鄰演算法之三群組分類圖

根據圖 5.15 可知，KNN 模型在  $K = 2$  時對三群雙變量常態資料的訓練誤差為 0.1013，在  $K = 16$  時的訓練誤差則為 0.1407。

## 5.2 線性判別分析、二次判別分析與 K-近鄰演算法的分類效果

為了更了解線性判別分析 (LDA)、二次判別分析 (QDA) 與 K-近鄰演算法在分類二群與三群資料時的表現，在此節中將分別模擬兩群與三群的雙變量常態資料，並透過改變各自的樣本數大小、平均數與共變異數來觀察三種方法的訓練誤差與測試誤差，為了更方便進行比較，先建立一個函數來儲存所有欲更改的參數，該函數部分程式碼呈現如下，接著便先就分類兩群雙變量常態資料的情況下進行討論，再討論分類三群雙變量常態資料的情況。

```
def param(n1, n2, mean1, mean2, mean3, mean4, val1, val2):
    m1, m2 = np.array([mean1, mean2]), np.array([mean3, mean4])
    Cov1 = np.array([[1, val1], [val1, 1]])
    Cov2 = np.array([[1, val2], [val2, 1]])
    mvn1 = multivariate_normal(mean = m1, cov = Cov1)
    mvn2 = multivariate_normal(mean = m2, cov = Cov2)
    A = mvn1.rvs(n1)
```

```
B = mvn2.rvs(n2)
Xvar = np.vstack((A, B))#2000*1矩陣\
y = np.hstack((np.zeros(n1), np.ones(n2)))#2000
param = np.c_[Xvar, y]
return param
```

### 5.2.1 線性判別分析、二次判別分析與 K-近鄰演算法的兩群組分類

- 改變樣本數大小 (sample size)

首先先嘗試改變樣本數大小來對三種方法的訓練誤差與測試誤差進行觀察。假設二群雙變量常態資料的參數分別如下，且其散布圖如圖 5.16。

$$\mu_1 = \begin{bmatrix} 2 \\ 2 \end{bmatrix}, \mu_2 = \begin{bmatrix} 4 \\ 1 \end{bmatrix}$$

$$\Sigma_1 = \begin{bmatrix} 1 & 0.2 \\ 0.2 & 1 \end{bmatrix}, \Sigma_2 = \begin{bmatrix} 1 & 0.2 \\ 0.2 & 1 \end{bmatrix}$$

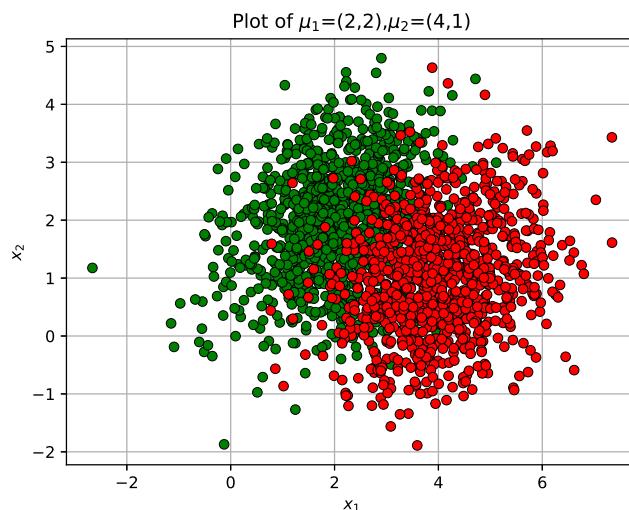


圖 5.16: 改變樣本數實驗的兩群組雙變量常態資料散布圖

根據圖 5.16 可以大致觀察到此兩群雙變量常態資料的關係，接著嘗試建立 LDA、QDA、以及  $K = 5$  與  $K = 10$  時的 KNN 在樣本數分別為  $n_1 = n_2 = 200$ 、 $n_1 = n_2 = 500$ 、 $n_1 = n_2 = 1000$  以及  $n_1 = 1000$  且  $n_2 = 500$  的模型，進而得到以下表 5.1 的結果，其部分程式碼亦於下面呈現。

```

n = 100
K1 = 5
weights = "uniform"
for i in range(n) :
    X_train, X_test, y_train, y_test = train_test_split(X,
        y, test_size = 0.2)
    Lda.fit(X_train, y_train)
    Lda.predict(X_test)
    LDAtestingError[i] = 1 - Lda.score(X_train, y_train)
    LDAtestingError[i] = 1 - Lda.score(X_test, y_test)
    Qda.fit(X_train, y_train)
    Qda.predict(X_test)
    QDAtestingError = 1 - Qda.score(X_train, y_train)
    QDAtestingError = 1 - Qda.score(X_test, y_test)
    Knn = neighbors.KNeighborsClassifier(K1, weights =
        weights)
    Knn.fit(X_train, y_train)
    Knn.predict(X_test)
    KNNtestingError1 = 1 - Knn.score(X_train, y_train)
    KNNtestingError1 = 1 - Knn.score(X_test, y_test)

```

表 5.1: 三種機器學習方法改變樣本數的兩群組訓練與測試資料分類錯誤率

	LDA		QDA		KNN( $K = 5$ )		KNN( $K = 10$ )	
	Train	Test	Train	Test	Train	Test	Train	Test
$n_1 = n_2 = 200$	0.1261	<b>0.1275</b>	0.1188	0.15	0.0969	<b>0.1875</b>	0.1125	0.16
$n_1 = n_2 = 500$	0.1049	<b>0.1056</b>	0.1075	0.105	0.1012	<b>0.095</b>	0.1112	0.1
$n_1 = n_2 = 1000$	0.1066	<b>0.1088</b>	0.1081	0.115	0.1	<b>0.14</b>	0.1	0.11
$n_1 = 1000, n_2 = 500$	0.0999	0.0992	0.1025	<b>0.0967</b>	0.0808	<b>0.13</b>	0.0958	0.12

為了計算三種機器學習方法在不同樣本數下的訓練誤差 ( Training Error ) 與測試誤差 ( Testing Error )，建立一個 for 迴圈同時對模擬的兩群雙變量常態資料配適三種模型，並各自利用 Bootstrapping 方法來將資料切割成訓練資料與測試資料 100 次，並計算這 100 次的平均訓練誤差與測試誤差，最後得到如表 5.1 之結果。在表 5.1 中將測試誤差最小的標為紅字，而誤差最大的模型則標為粗體，根據其結果可知，在兩群模擬之雙變量常態資料的樣本數都只有 200 時，KNN 模型 ( $K = 5$ ) 的訓練誤差最小，為 0.0969，但 LDA 模型的測試誤差才是最小的，只有 0.1275，其在樣本數為 1000 時的測試誤差也最小，只有 0.1088。

- 改變平均數向量 ( mean vector )

接著嘗試設置兩群雙變量常態資料的樣本數為  $n_1 = n_2 = 1000$ ，變異數皆為 0.2，並透過更改平均數向量來了解三種模型的分類表現。除了圖 5.16 的模擬資料，亦設置兩組新的模擬資料來進行比較，其參數如下且其散佈圖如圖 5.17。

$$\mu_1 = \begin{bmatrix} 2 \\ 2 \end{bmatrix}, \mu_2 = \begin{bmatrix} 3 \\ 3 \end{bmatrix}$$

$$\mu'_1 = \begin{bmatrix} 2 \\ 6 \end{bmatrix}, \mu'_2 = \begin{bmatrix} 5 \\ 9 \end{bmatrix}$$

$$\Sigma_1 = \Sigma'_1 = \begin{bmatrix} 1 & 0.2 \\ 0.2 & 1 \end{bmatrix}, \Sigma_2 = \Sigma'_2 = \begin{bmatrix} 1 & 0.2 \\ 0.2 & 1 \end{bmatrix}$$

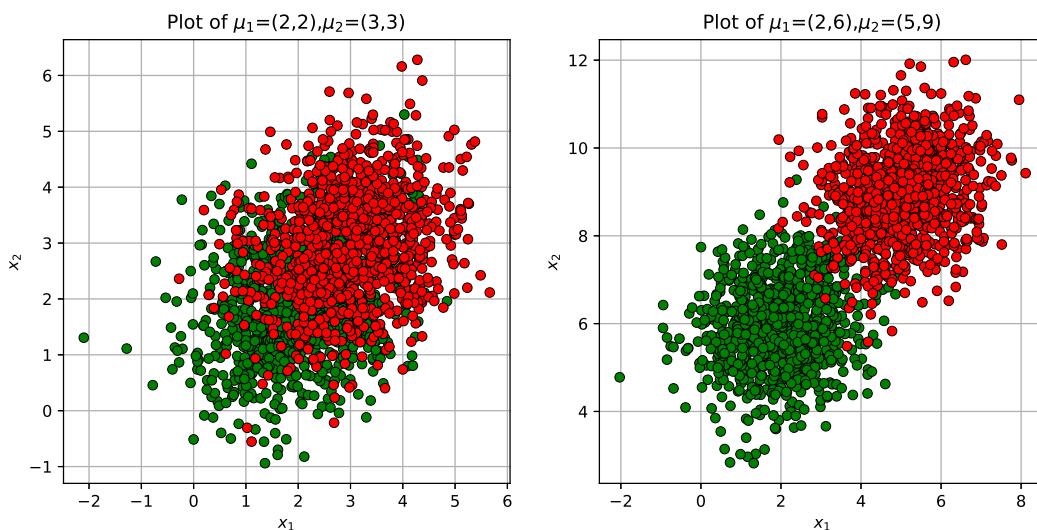


圖 5.17: 改變群平均實驗的兩群組雙變量常態資料散佈圖

在圖 5.17 中，左圖為平均數為  $\mu_1$  與  $\mu_2$ 、共變異矩陣為  $\Sigma_1$  與  $\Sigma_2$  的雙變量常態散佈圖，右圖則為平均數為  $\mu'_1$  與  $\mu'_2$ 、共變異矩陣為  $\Sigma'_1$  與  $\Sigma'_2$  的雙變量常態散佈圖。接著觀察利用三種方法配適此兩種雙變量常態資料以及圖 5.16 之模擬資料的平均

訓練誤差與測試誤差，其結果如表 5.2。根據表 5.2 可知四種模型各有優劣的表現，其所能得到的分類效果相差不大。

表 5.2: 三種機器學習方法改變群平均的兩群組訓練與測試資料分類錯誤率

	LDA		QDA		KNN( $K = 5$ )		KNN( $K = 10$ )	
	Train	Test	Train	Test	Train	Test	Train	Test
$\mu_1 = (2, 2), \mu_2 = (4, 1)$	0.1066	<b>0.1088</b>	0.1081	0.115	0.1	<b>0.14</b>	0.1	0.11
$\mu_1 = (2, 2), \mu_2 = (3, 3)$	0.2687	0.27	0.2681	<b>0.2625</b>	0.2125	<b>0.3175</b>	0.2525	0.2825
$\mu_1 = (2, 6), \mu_2 = (5, 9)$	0.018	<b>0.0175</b>	0.02	<b>0.015</b>	0.0156	<b>0.015</b>	0.0162	<b>0.015</b>

- 改變共變異矩陣 ( covariance matrix )

在 LDA 模型中有個前提假設是資料服從常態且每群的共變異數相同，但在實務資料中往往很難服從此前提假設，因此嘗試觀察改變共變異數時四種模型的訓練誤差與測試誤差發生的改變。我們設置幾種參數如下並觀察各自的模擬資料散布圖 5.18。

$$\mu_1 = \mu'_1 = \mu''_1 = \begin{bmatrix} 2 \\ 2 \end{bmatrix}, \mu_2 = \mu'_2 = \mu''_2 = \begin{bmatrix} 4 \\ 1 \end{bmatrix}$$

$$\Sigma_1 = \begin{bmatrix} 1 & 0.5 \\ 0.5 & 1 \end{bmatrix}, \Sigma_2 = \begin{bmatrix} 1 & 0.5 \\ 0.5 & 1 \end{bmatrix}$$

$$\Sigma'_1 = \begin{bmatrix} 1 & 0.2 \\ 0.2 & 1 \end{bmatrix}, \Sigma'_2 = \begin{bmatrix} 1 & 0.8 \\ 0.8 & 1 \end{bmatrix}$$

$$\Sigma''_1 = \begin{bmatrix} 1 & 0.9 \\ 0.9 & 1 \end{bmatrix}, \Sigma''_2 = \begin{bmatrix} 1 & 0.3 \\ 0.3 & 1 \end{bmatrix}$$

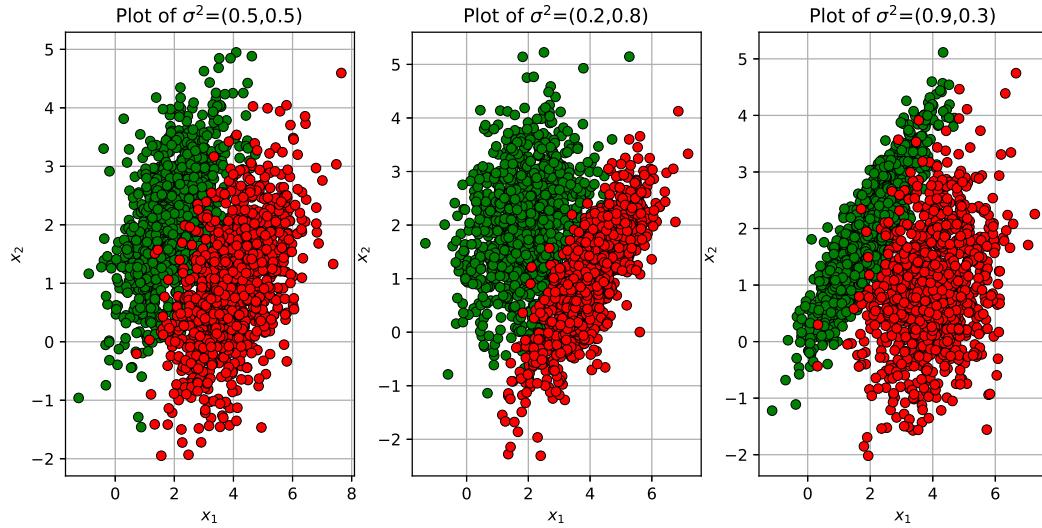


圖 5.18: 改變群共變異數實驗的兩群組雙變量常態資料散佈圖

根據圖 5.18 可以觀察平均數分別為  $(2, 2)$ 、 $(4, 1)$  而共變異矩陣為  $\Sigma_1$  與  $\Sigma_2$ 、 $\Sigma'_1$  與  $\Sigma'_2$  以及  $\Sigma''_1$  與  $\Sigma''_2$  的常態資料的散佈圖。另外亦將圖 5.16 中平均數為  $(2, 2)$ 、 $(4, 1)$  而變異數為  $(0.2, 0.2)$  的雙變量常態資料列入比較並計算其訓練誤差與測試誤差，其結果如表 5.3。

表 5.3: 三種機器學習方法改變群共變異數的兩群組訓練與測試資料分類錯誤率

	LDA		QDA		KNN( $K = 5$ )		KNN( $K = 10$ )	
	Train	Test	Train	Test	Train	Test	Train	Test
$\sigma^2 = (0.2, 0.2)$	0.1066	<b>0.1088</b>	0.1081	0.115	0.1	<b>0.14</b>	0.1	0.11
$\sigma^2 = (0.5, 0.5)$	0.0728	0.076	0.0725	<b>0.085</b>	0.0613	0.0725	0.0669	<b>0.07</b>
$\sigma^2 = (0.2, 0.8)$	0.0715	<b>0.0726</b>	0.0537	<b>0.06</b>	0.05	0.065	0.0506	<b>0.06</b>
$\sigma^2 = (0.9, 0.3)$	0.0547	<b>0.0545</b>	0.0262	<b>0.025</b>	0.0212	0.03	0.0288	0.03

接著透過表 5.3 可以觀察四種模型的訓練誤差與測試誤差，可以發現在變異數相同，即  $\sigma^2 = (0.2, 0.2)$  與  $\sigma^2 = (0.5, 0.5)$  時，QDA 模型與 KNN ( $K = 5$ ) 的測試誤差最大，而 LDA 模型的表現算是不錯，但是在變異數不同時，LDA 模型的測試誤差明顯比其他三種模型要高得多，尤其在  $\sigma^2 = (0.9, 0.3)$  時，其他三種模型的測試誤差最高只到 0.03，LDA 模型的測試誤差則達到 0.0545。

### 5.2.2 線性判別分析、二次判別分析與 K-近鄰演算法的三群組分類

在觀察過改變樣本數大小、平均數向量以及共變異矩陣的二群雙變量常態資料後，接著嘗試改變三群雙變量常態資料的樣本數、平均數與共變異並同樣觀察 LDA 模型、QDA 模型、KNN 模型 ( $K = 5$ ) 與 KNN 模型 ( $K = 10$ ) 四種模型的訓練誤差與測試誤差進行比較，首先先觀察改變樣本數大小的情況。

- 改變樣本數大小 (sample size)

此處將設置三群雙變量常態資料的參數如下，並比較  $n_1 = n_2 = n_3 = 200$ 、 $n_1 = n_2 = n_3 = 500$ 、 $n_1 = n_2 = n_3 = 1000$  以及  $n_1 = 1000$ 、 $n_2 = 500$  且  $n_3 = 200$  四種樣本數的雙變量常態資料， $n_1 = n_2 = n_3 = 1000$  的散布圖亦呈現於圖 5.19。

$$\mu_1 = \begin{bmatrix} 2 \\ 2 \end{bmatrix}, \mu_2 = \begin{bmatrix} 4 \\ 1 \end{bmatrix}, \mu_3 = \begin{bmatrix} 4 \\ 5 \end{bmatrix}$$

$$\Sigma_1 = \begin{bmatrix} 1 & 0.2 \\ 0.2 & 1 \end{bmatrix}, \Sigma_2 = \begin{bmatrix} 1 & 0.2 \\ 0.2 & 1 \end{bmatrix}, \Sigma_3 = \begin{bmatrix} 1 & 0.2 \\ 0.2 & 1 \end{bmatrix}$$

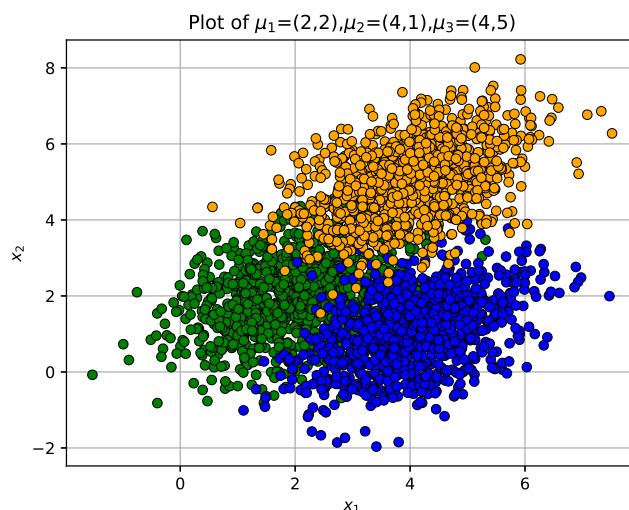


圖 5.19: 改變群樣本數實驗的三群組雙變量常態資料散布圖

圖 5.19 即為此三群雙變量常態資料的散布圖，其中綠色點是平均數為  $\mu_1$  共變異數為  $\Sigma_1$  的資料點、藍色點是平均數為  $\mu_2$  共變異數為  $\Sigma_2$  的資料點， $\mu_3$  與  $\Sigma_3$  則是黃色雙變量常態資料的參數，其配適四種模型的誤差如表 5.4。

表 5.4: 三種機器學習方法改變樣本數的三群組訓練與測試資料分類錯誤率

	LDA		QDA		KNN( $K = 5$ )		KNN( $K = 10$ )	
	Train	Test	Train	Test	Train	Test	Train	Test
$n_1 = n_2 = n_3 = 200$	0.3876	<b>0.4095</b>	0.3896	0.3833	0.2667	<b>0.3333</b>	0.3521	0.4
$n_1 = n_2 = n_3 = 500$	0.3889	0.4032	0.3717	0.4	0.2892	<b>0.3833</b>	0.315	<b>0.42</b>
$n_1 = n_2 = n_3 = 1000$	0.399	0.4105	0.3942	<b>0.415</b>	0.2825	<b>0.395</b>	0.3129	0.4133
$n_1 = 1000, n_2 = 500, n_3 = 200$	0.2084	<b>0.2081</b>	0.2022	0.2294	0.1801	0.2382	0.1958	<b>0.25</b>

根據表 5.4 可以發現，KNN 模型 ( $K = 5$ ) 在樣本數為 200、500 與 1000 時的測試誤差都最小，而在三群資料的樣本數不同時，則是 LDA 模型表現最好。

- 改變平均數向量 ( mean vector )

在觀察過改變樣本數時四種模型的誤差後，接著觀察在三群資料的樣本數都是 1000 時，平均數與共變異數如下之三群雙變量常態資料配是四種模型的誤差，此結果會與圖 5.19 的結果一起進行比較。

$$\mu_1 = \begin{bmatrix} 3 \\ 3 \end{bmatrix}, \mu_2 = \begin{bmatrix} 4 \\ 4 \end{bmatrix}, \mu_3 = \begin{bmatrix} 5 \\ 5 \end{bmatrix}$$

$$\mu'_1 = \begin{bmatrix} 1 \\ 6 \end{bmatrix}, \mu'_2 = \begin{bmatrix} 2 \\ 5 \end{bmatrix}, \mu'_3 = \begin{bmatrix} 5 \\ 5 \end{bmatrix}$$

$$\Sigma_1 = \Sigma'_1 = \begin{bmatrix} 1 & 0.4 \\ 0.4 & 1 \end{bmatrix}, \Sigma_2 = \Sigma'_2 = \begin{bmatrix} 1 & 0.4 \\ 0.4 & 1 \end{bmatrix}, \Sigma_3 = \Sigma'_3 = \begin{bmatrix} 1 & 0.4 \\ 0.4 & 1 \end{bmatrix}$$

圖 5.20 展示了上方參數的資料散布圖。其左圖展示了參數為  $\mu_1$ 、 $\mu_2$ 、 $\mu_3$  與  $\Sigma_1$ 、 $\Sigma_2$  以及  $\Sigma_3$  的三群雙變量常態資料，右圖則展示了參數為  $\mu'_1$ 、 $\mu'_2$ 、 $\mu'_3$  與  $\Sigma'_1$ 、 $\Sigma'_2$  以及  $\Sigma'_3$  的三群雙變量常態資料，其訓練誤差與測試誤差如表 5.5。

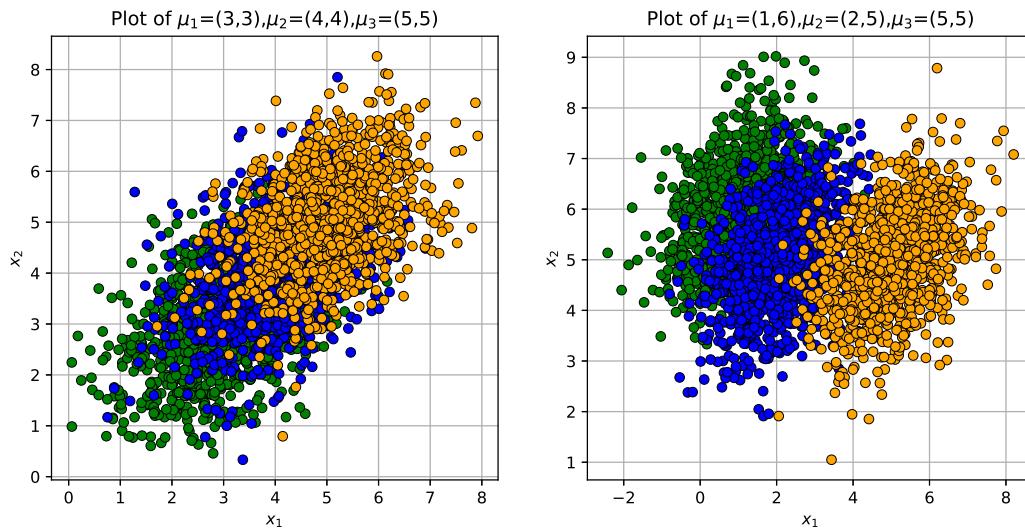


圖 5.20: 改變群平均實驗的三群組雙變量常態資料散布圖

表 5.5: 三種機器學習方法改變群平均的三群組訓練與測試資料分類錯誤率

	LDA		QDA		KNN( $K = 5$ )		KNN( $K = 10$ )	
	Train	Test	Train	Test	Train	Test	Train	Test
$\mu_1 = (2, 2), \mu_2 = (4, 1), \mu_3 = (4, 5)$	0.399	0.4105	0.3942	<b>0.415</b>	0.2825	<b>0.395</b>	0.3129	0.4133
$\mu_1 = (3, 3), \mu_2 = (4, 4), \mu_3 = (5, 5)$	0.4963	<b>0.5081</b>	0.5046	0.515	0.3571	<b>0.5317</b>	0.4087	0.525
$\mu_1 = (1, 6), \mu_2 = (2, 4), \mu_3 = (5, 5)$	0.497	<b>0.5099</b>	0.4829	0.5117	0.3579	<b>0.5383</b>	0.3992	0.5333

根據表 5.5 可知，LDA 模型在  $\mu_1 = (3, 3), \mu_2 = (4, 4), \mu_3 = (5, 5)$  以及  $\mu_1 = (1, 6), \mu_2 = (2, 4), \mu_3 = (5, 5)$  的測試誤差最低，但在  $\mu_1 = (2, 2), \mu_2 = (4, 1), \mu_3 = (4, 5)$  時則是 KNN 模型 ( $K = 5$ ) 的測試誤差最低，因此並沒有在所有改變平均數的模擬資料表現最佳之模型。

- 改變共變異矩陣 ( covariance matrix )

接著再嘗試改變共變異數矩陣來比較四種模型的誤差，LDA 模型設定了資料共變異數相同的假設，因此期望能看到在改變共變異數時，QDA 模型的誤差一定會比 LDA 模型還低之結果。故設置以下參數之三群雙變量常態資料來進行此觀察，其資料散布圖如圖 5.21，並與兩種 KNN 模型進行比較。

$$\mu_1 = \mu'_1 = \mu''_1 = \begin{bmatrix} 2 \\ 2 \end{bmatrix}, \mu_2 = \mu'_2 = \mu''_2 = \begin{bmatrix} 4 \\ 1 \end{bmatrix}, \mu_3 = \mu'_3 = \mu''_3 = \begin{bmatrix} 4 \\ 5 \end{bmatrix}$$

$$\Sigma_1 = \begin{bmatrix} 1 & 0.4 \\ 0.4 & 1 \end{bmatrix}, \Sigma_2 = \begin{bmatrix} 1 & 0.4 \\ 0.4 & 1 \end{bmatrix}, \Sigma_3 = \begin{bmatrix} 1 & 0.4 \\ 0.4 & 1 \end{bmatrix}$$

$$\Sigma'_1 = \begin{bmatrix} 1 & 0.1 \\ 0.1 & 1 \end{bmatrix}, \Sigma'_2 = \begin{bmatrix} 1 & 0.7 \\ 0.7 & 1 \end{bmatrix}, \Sigma'_3 = \begin{bmatrix} 1 & 0.9 \\ 0.9 & 1 \end{bmatrix}$$

$$\Sigma''_1 = \begin{bmatrix} 1 & 0.9 \\ 0.9 & 1 \end{bmatrix}, \Sigma''_2 = \begin{bmatrix} 1 & 0.3 \\ 0.3 & 1 \end{bmatrix}, \Sigma''_3 = \begin{bmatrix} 1 & 0.1 \\ 0.1 & 1 \end{bmatrix}$$

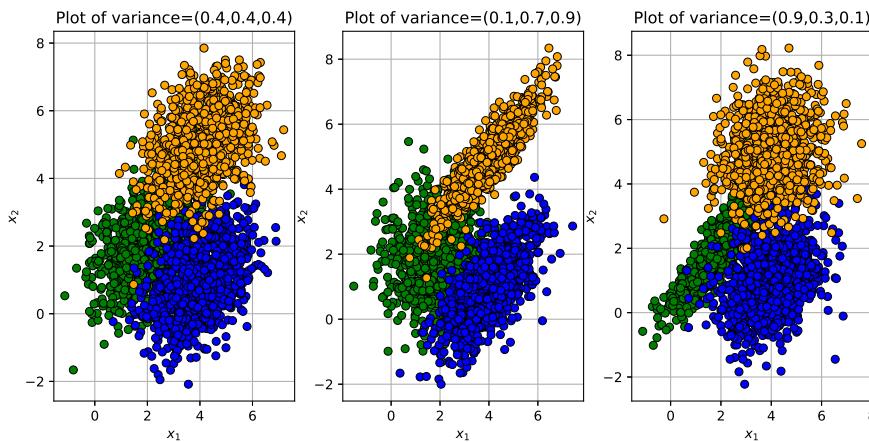


圖 5.21: 改變群共變異數實驗的三群組雙變量常態資料散布圖

圖 5.21 分別表示了三群樣本數皆為 1000、平均數為  $(2, 2)$ 、 $(4, 1)$  與  $(4, 5)$  之三群雙變量常態資料，其變異數分別為  $(0.4, 0.4, 0.4)$ 、 $(0.1, 0.7, 0.9)$  與  $(0.9, 0.3, 0.1)$ ，而其配適四種模型的訓練誤差與測試誤差呈現於表 5.6。

根據表 5.6 可知，在改變共變異數前，LDA 模型的測試誤差都比 QDA 模型還低，而在改變共變異數後，LDA 模型的測試誤差都變得比 QDA 模型還高，尤其在  $\sigma^2 = (0.9, 0.3, 0.1)$  時，LDA 模型的測試誤差甚至比兩種 KNN 模型都還要高。另外，KNN 模型 ( $K = 5$ ) 在改變共變異數前的測試誤差都比 KNN 模型 ( $K = 10$ ) 還低，而在改變共變異數後則是 KNN 模型 ( $K = 10$ ) 的表現都比 KNN 模型 ( $K = 5$ ) 還要好。

表 5.6: 三種機器學習方法改變群共變異數的三群組訓練與測試資料分類錯誤率

	LDA		QDA		KNN( $K = 5$ )		KNN( $K = 10$ )	
	Train	Test	Train	Test	Train	Test	Train	Test
$\sigma^2 = (0.2, 0.2, 0.2)$	0.399	0.4105	0.3942	<b>0.415</b>	0.2825	<b>0.395</b>	0.3129	0.4133
$\sigma^2 = (0.4, 0.4, 0.4)$	0.3804	<b>0.3952</b>	0.3646	0.4183	0.2608	0.4217	0.3087	<b>0.425</b>
$\sigma^2 = (0.1, 0.7, 0.9)$	0.3674	0.372	0.3579	<b>0.37</b>	0.2458	<b>0.3783</b>	0.2879	0.3717
$\sigma^2 = (0.9, 0.3, 0.1)$	0.3609	<b>0.3788</b>	0.3488	0.3533	0.2317	0.3467	0.2775	<b>0.345</b>

### 5.3 結論

在此章中介紹了三種機器學習方法，線性判別分析 ( LDA )、二次判別分析 ( QDA ) 與 K-近鄰演算法的理論與如何利用 Python 配適模型並繪圖進行觀察。為了了解三種模型的優劣，本章節亦利用 Bootstrapping 方法模擬了兩群與三群的雙變量常態資料對模型的訓練誤差與測試誤差進行比較，期望能協助更好的了解三種模型的分類效果。



# 第 6 章

## 類神經網路的分類問題

類神經網路，又稱 ANN (Artificial Neuralwork)，其核心概念是人類希望能利用計算機模擬人類神經元收發信號的學習系統，透過設置變數來模擬神經元分布，對函式進行估計和模擬，讓資訊（細胞）經過變數的時候搭配權重（Weight）與偏差（Bias），利用閥值模擬細胞決定是否將訊息傳遞給下一個神經元。類神經網路（ANN）透過大量的人工神經元連結進行計算，並能自行透過學習更多資訊進而對模型進行改進，因此是一種具備學習功能的方法，它通過統計學的學習方法（Learning Method）進行最佳化用來解決各種各樣的實際問題，包含影像辨識與語音處理，概念也使類神經網路帶動了全球流行 AI 人工智能的風潮，並成為深度學習（Deep Learning）領域中常見的圖形辨識方法之一。在本章節中將嘗試將類神經網路模型應用於經典的機器手臂方程式（Robot Arm Equation）探討連續型資料問題以及圖形識別（Pattern Recognition）探討類別型資料問題，並試驗如何調整參數與樣本數來使類神經網路模型（ANN）最佳化。

### 6.1 類神經網路理論介紹

首先先了解類神經網路（ANN）的基礎理論。如上所說，類神經網路是一種模仿大腦運作的機器學習方式，透過大量的人工神經元連結進行計算，並能自行透過學習更多資訊進而對模型進行改進，其結構一般包含輸入層（Input）、隱藏層（Hidden Layer）以及輸出層（Ouput），其中隱藏層的數量是可以進行改變的，其模型結構如圖 6.1<sup>1</sup>。

---

<sup>1</sup> 圖片來源: <https://case.ntu.edu.tw/blog/?p=26340>

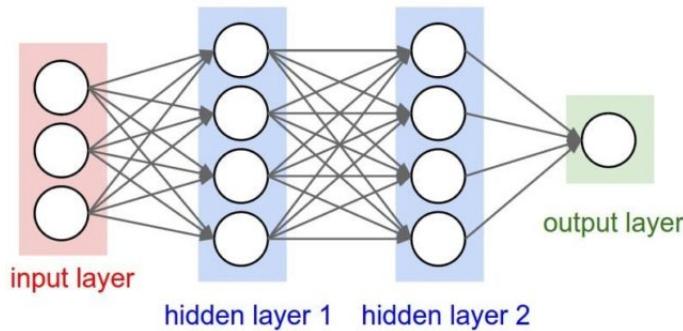


圖 6.1: 類神經網路 ( ANN ) 示意圖

欲了解類神經網路的細部原理，首先須先了解甚麼是機器學習？在機器學習中一般有三種常見的領域，包含迴歸 ( Regression )、分類 ( Classification ) 以及結構學習 ( Structured Learning )。迴歸是透過建立函數來求得數值，分類是利用函數來進行選擇題，結構學習則是利用函數來進行創造，產生具有結構的物件 ( 例如：文章 )。為了了解類神經網路理論，假設欲利用 youtube 後台所有數據來預測 2 月 26 日的觀看量<sup>2</sup>，則一般會遵循以下步驟：

- 建立包含未知參數的函數

假設函數為  $y = b + wx_1$ ，其中  $x_1$ : views on 2/25、 $y$  : views on 2/26， $w$  ( weight ) 與  $b$  ( bias ) 為模型之未知參數，其從已有資料中進行學習。

- 定義損失函數 ( Loss Function )

定義函數的輸入 ( Input ) 為未知參數  $b$ 、 $w$ ，而損失函數 ( Loss Function ) 則用來表示在將 input 丟入函數後，此函數的預測效果有多好，其函數可寫作如式 (6.1)。

$$Loss = \frac{1}{N} \sum_n e_n \quad (6.1)$$

其中  $e = (y - \hat{y})^2$ 。

- 利用不同的未知參數計算不同的損失函數並畫圖
- 最佳化模型 ( Optimization )

最佳化模型有很多方法，其中有一個常見的方法是梯度下降 ( Gradient Descent )，嘗試求得合適的未知參數  $w$ 、 $b$  使得損失函數的值最小，在此文不進行過多討論。

<sup>2</sup>資料參考來源：李宏毅機器學習 <https://speech.ee.ntu.edu.tw/hylee/ml/2022-spring.php>

在此範例中，所有的模型調整全部建立在線性模型 ( Linear Model ) 之上進行調整，但線性模型往往太過簡單，就算選出了具有最小損失函數的模型，其依舊被初始設置的線性模型所限制 ( model bias )。而如何設置更複雜的模型？Sigmoid Function 協助達成了此目的，其被表示如式 (6.2)。

$$\text{sigmoid} = \frac{1}{1 + e^{-x}} \quad (6.2)$$

故將新的預測模型寫作式 (6.3)。

$$y = c \frac{1}{1 + e^{-(b+wx_1)}} = c \times \text{sigmoid}(b + wx_1) \quad (6.3)$$

式 (6.3) 中的  $w$  控制函數坡度、 $b$  控制左右移動， $c$  則可以改變函數的高度，若是具有兩個隱藏層，則利用此原理可以將類神經網路模型寫作式 (6.4)。

$$\hat{y} = b_k^2 + \sum_{i=1}^q w_{ki}^2 g\left(\sum_{j=1}^p w_{ij}^1 x_j + b_i^1\right) + b_k^2, 1 \leq k \leq r \quad (6.4)$$

其中函式  $g(\bullet)$  為映射函數 ( Sigmoid Function )，其數學式如式 (6.5)。

$$g(z) = c_1 \frac{1 - e^{-c_2 z}}{1 + e^{-c_2 z}} \quad (6.5)$$

在式 (6.4) 中， $q$  為映射函數 ( Sigmoid Function ) 的數量、 $p$  為輸入的特徵 ( feature ) 的數量、 $w_{ij}^1$  與  $b_i^1$  代表第一個隱藏層的第  $i$  個神經元與第  $j$  個 input 的權重 ( Weight ) 跟偏誤 ( Biases )，而  $w_{ki}^2$  與  $b_k^2$  則是第二層隱藏層的第  $k$  個神經元與前一隱藏層的第  $i$  個輸出的權重與偏誤，為了使模型最佳化，期望能得到使此模型的損失函數 ( Loss Function ) 最小的未知參數。

以上即是類神經網路的基本原理，但據此亦可知雖然其原理簡單有效，如果資料為高維資料時，如何求得使損失函數最小的未知參數也變得困難，如何定義需要多少隱藏層也成為挑戰，故下節將展示機器手臂 ( Robot Arm Equation ) 與圖形識別 ( Pattern Equation ) 在不同訓練過程下產生的不同測試結果。

## 6.2 機械手臂方程式

為了更了解類神經網路 ( ANN ) 的應用，利用機械手臂來作為連續型資料的範例<sup>3</sup>，如圖 6.2<sup>4</sup> 中展示了簡易的兩截式機械手臂。圖 6.2 中的機械手臂有兩截手臂，長度分別是  $l_1$  跟  $l_2$ ，其位置會隨著角度  $\theta_1$  與  $\theta_2$  改變而改變，使手臂頂端可以移動到目的地  $(x, y)$  ( Desired Location )，而為了了解手臂頂端可以抵達的範圍，必須計算出  $\theta_1$  與  $\theta_2$  的角度，即嘗試解逆運動方程式，如式 (6.6) 的 FKE 與式 (6.7) 的 LKE。

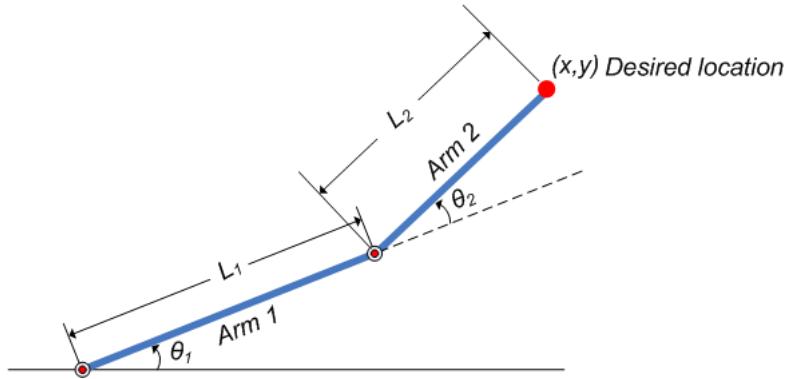


圖 6.2: 兩截式機械手臂示意圖

$$\begin{aligned} x &= l_1 \cos(\theta_1) + l_2 \cos(\theta_1 + \theta_2) \\ y &= l_1 \sin(\theta_1) + l_2 \sin(\theta_1 + \theta_2) \end{aligned} \quad (6.6)$$

$$\theta_2 = \cos^{-1}\left(\frac{x^2 + y^2 - l_1^2 - l_2^2}{2l_1 l_2}\right) \theta_1 = \tan^{-1}\left(\frac{y}{x}\right) - \tan^{-1}\left(\frac{l_2 \sin(\theta_2)}{l_1 + l_2 \cos(\theta_2)}\right) \quad (6.7)$$

直接解逆運動方程式看似是可行的，但是在機械手臂從兩截變成三截甚至以上時此方式是很困難的，因此使用類神經網路 ( ANN ) 來達成此效果。類神經網路會將手臂頂端能到達的位置  $(x, y)$  作為輸入值，而角度  $\theta_1$  與  $\theta_2$  作為輸出值來建立模型，最後透過訓練來找出輸入與輸出的關係，並計算當面對新的位置資料時，預測出的輸出值 ( 角度 ) 與真實值的均方根誤差 ( RMSE )。

在使用類神經網路 ( ANN ) 建立模型時，一般會使用兩種套件，一種是 sklearn 套件中的 *neuralnetwork.MLPRegressor*，另一種則是 *Neurolab.newff*，首先將展示套件

<sup>3</sup> 資料參考: <https://ntpuccw.blog/python-in-learning/sml-lesson-4-artificial-neural-network-ann-deterministic-approach/>

<sup>4</sup> 圖片來源: <https://www.mathworks.com/help/fuzzy/modeling-inverse-kinematics-in-a-robotic-arm.html>

*neuralnetwork.MLPRegressor* 建立類神經網路的過程，並接著分別利用兩種套件嘗試改變樣本數大小、神經元個數與隱藏層個數，找出表現最好，即 RMSE 與 SSE 最小之類神經網路模型。

### 6.2.1 最佳化模型

在此節中將一一詳述建立類神經網路模型 ( ANN ) 的過程，建立模型的過程一般分為四個步驟，首先先繪製機械手臂頂端經過路線的扇形圖，並從此扇形區域中抽取訓練資料，接著分別準備類神經網路的輸入 ( 位置  $(x, y)$  ) 與輸出值 ( 角度  $(\theta_1, \theta_2)$  )，並分別利用兩個套件 *neuralnetwork.MLPRegressor* 與 *NeuroLab.newff* 建立類神經網路模型，最後繪製真實值與估計值的散布圖並計算其均方根誤差 ( RMSE ) 、 R-squared 、損失函數 ( Loss function ) 以及迭代次數 ( Number of iteration )，此處利用套件 *neuralnetwork.MLPRegressor* 來進行過程之展示。

- 繪製扇形圖

首先利用以下程式碼繪製扇形圖，展示機械手臂頂端會經過的區域，其圖形如下圖 6.3 之藍色區域。

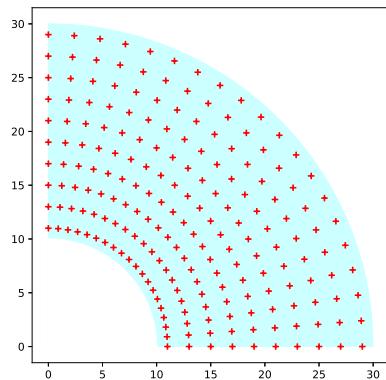


圖 6.3: 扇形圖

```
11, 12 = 20, 10
f1 = lambda x: np.sqrt((11 + 12)**2 - x **2)
f2 = lambda x: np.sqrt(12 **2 - x **2)

plt.figure(figsize = (6, 6))
x = np.linspace(0, 11+12, 200)
plt.fill_between(x, f1(x), 0, color = "#CCFFFF")
x = np.linspace(0, 12, 200)
plt.fill_between(x, f2(x), 0, color = "white")
```

- 準備輸入值與輸出值

接著根據式 (6.6) 的 FKE 與式 (6.7) 的 LKE，分別設置類神經網路的輸入值 (位置) 與輸出值 (角度)，輸入值繪製如圖 6.3 中之紅點，其程式碼亦呈現如下，其中，我們會利用函數 `ravel()` 將輸入值與輸出值的向量合併成矩陣。

```
theta2 = np.arccos((X.ravel()**2 + Y.ravel()**2 - 11**2 - 12**2)/(2*11*12))
theta1 = np.arctan(Y.ravel()/X.ravel()) - np.arctan(12*np.sin(theta2)/(11+12*np.cos(theta2)))

InputX = np.c_[X.ravel(), Y.ravel()]
OutputY = np.c_[theta1, theta2]
```

- 設置參數並建立類神經網路模型

為了更了解類神經網路模型，利用上述建立的輸入值與輸出值觀察改變隱藏層數量，觀察不同隱藏層數量產生的不同均方根誤差 (RMSE)、Rsquared、損失函數 (Loss function) 以及迭代次數 (Number of iteration) 的值，其圖展示如圖 6.4，程式碼亦呈現如下。

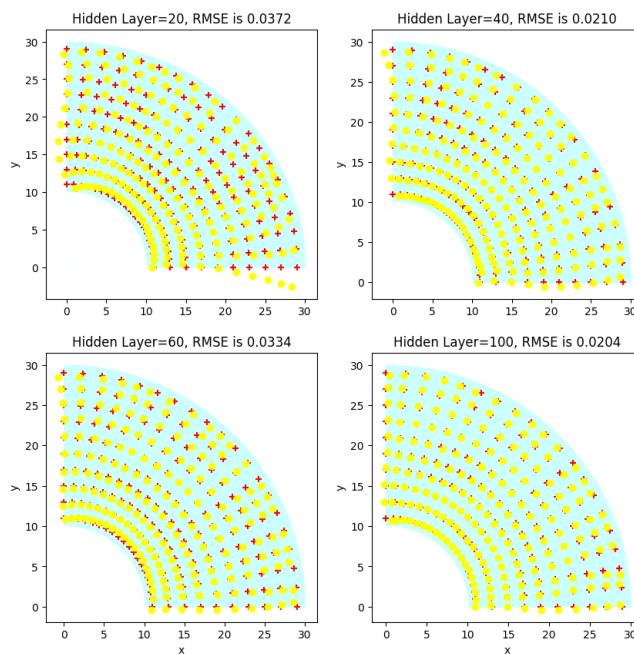


圖 6.4: 改變隱藏層之類神經網路模型

表 6.1: 不同隱藏層之類神經網路模型比較

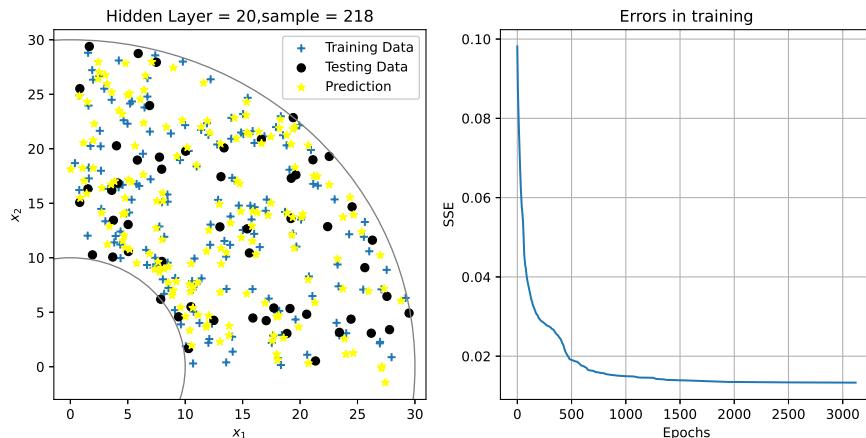
	number=20	number=40	number=60	number=100
RMSE	0.0372	0.021	0.0334	<b>0.0204</b>
Loss Function	0.0007	0.0003	0.0006	<b>0.0002</b>
Number of Iteration	<b>1502</b>	4712	1662	3187
R-squared	0.996	0.998	0.997	<b>0.999</b>

圖 6.4 中紅色的點是真實值，黃色的點是類神經網路預測出來的值，而表 6.1 則展示了不同隱藏層的模型好壞，根據此表可知，隱藏層數設為 100 時類神經網路模型的表現會最好，RMSE、R-squared 與 Loss Function 都是最小的。在建立類神經網路模型時，設置參數  $max\_iter = 8000$ ，即限制迴圈更新估計值的最大迴圈數為 8000，演算法則設置為  $solver = "lbfgs"$ ，即 LBFGS 演算法，另外也有 adam 與 sgd 兩種演算法可供選擇，最後，此模型之非線性函數  $activation$  則選用了 logistic 演算法。

```
v = 100
hidden_layers = (v, ) , solver = "lbfgs"
mlp_reg = MLPRegressor(max_iter = 8000, solver = solver,
hidden_layer_sizes = hidden_layers, verbose = False,
activation = "logistic", tol=1e-6, random_state = 0)
mlp_reg.fit(InputX, OutputY) # Training

OutputY_hat = mlp_reg.predict(InputX) # Calculate fitted
values
theta1_hat, theta2_hat = OutputY_hat[:,0], OutputY_hat
[:,1]
x_hat = 11 * np.cos(theta1_hat) + 12 * np.cos(theta1_hat+
theta2_hat)
y_hat = 11 * np.sin(theta1_hat) + 12 * np.sin(theta1_hat+
theta2_hat)
```

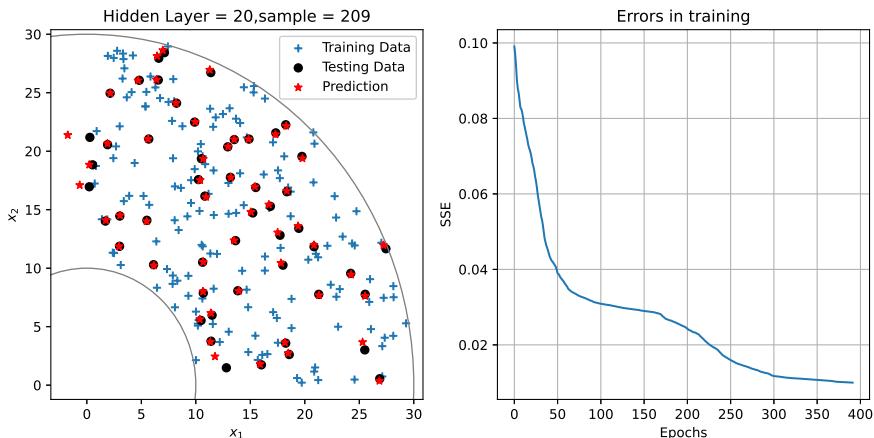
以上即是在類神經網路在機械手臂問題上如何建立模型的過程，但以上升成訓練資料的方式其實並非最好的選擇，因為這些訓練資料並非亂數生成的，而該如何更好的在特定區域內公平抽取訓練資料呢？此處將藉助均勻分配來協助達到此目的，並展示利用均勻分配抽出訓練資料與測試資料後，如何分別利用套件 *neuralnetwork.MLPRegressor* 與 *NeuroLab.newff* 建立類神經網路模型、觀察訓練誤差的變化、並比較兩種套件改變隱藏層與神經元的情況下各自的 RMSE 與 SSE 之變化。

圖 6.5: 套件 *MLPRegressor* 之類神經網路模型

```
def randsphere(center, radius, n_per_sphere):
    r = radius
    ndim = center.size
    x = np.random.normal(size=(n_per_sphere, ndim))
    ssq = np.sum(x ** 2, axis=1)
    fr = r * gammainc(ndim / 2, ssq / 2) ** (1 / ndim) / np.sqrt(ssq)
    frtiled = np.tile(fr.reshape(n_per_sphere, 1), (1, ndim))
    p = center + np.multiply(x, frtiled)
    return p
```

首先分別繪製利用兩套件所訓練的類神經網路模型的訓練結果圖，如圖 6.5 與圖 6.6，分別展示了在隱藏層數為 20 的情況下各自的預測位置的成果，以及隨著 epoch 增加訓練誤差的變化。首先利用均勻分配生成一個圓內的隨機變數，其樣本數設 1000，再擷取位在第一象限的隨機變數作為資料，如上程式碼所示，其中函數 *randsphere* 是用來產生高度空間球體內均勻散佈的點，其第一個參數 *center* 代表中心點位置。*radius* 代表半徑、*n\_per\_sphere* 則是樣本數；所以 *randsphere(np.array([0, 0], 30, 1000)* 代表在半徑 30 的圓內均勻產生 1000 個亂數。

而在圖 6.5 與圖 6.6 的範例中，最後得到樣本數分別為 209 個與 218 個的資料，並將其切成訓練資料與測試資料，其程式碼呈現如下，再計算 RMSE 以及 SSE 的值。此處將分別就不同的隱藏層數與樣本數所得到的 RMSE 與 SSE 進行比較，而圖 6.5 與圖 6.6 僅僅展示了隱藏層為 20 個的情況下繪製的圖形，其他參數所繪製出的圖則不予贅述，其結果分別展示於下表 6.2、表 6.3 與表 6.4。

圖 6.6: 套件 *Neurolab.newff* 之類神經網路模型

```

11, 12 = 20, 10
n = 1000
# data distributed randomly in a circle
center = np.array([0, 0])
radius_in, radius_out = 10, 30
p = randsphere(center, radius_out, n)
p = p[(p[:,0] > 0) & (p[:,1] > 0), :]
# sample in the first quadrant
d = np.sum(p**2, axis=1)

p = p[d >= radius_in**2, :]
# sample in the fan area
x1, x2 = p[:,0], p[:,1]
x_train, x_test, y_train, y_test =
    train_test_split(x1, x2, test_size = 0.25)

theta2 = np.arccos((x_train**2 + y_train**2 - 11**2 - 12**2) /
    (2*11*12))
theta1 = np.arctan(y_train/x_train) - np.arctan(12 * np.sin(
    theta2)/(11 + 12 * np.cos(theta2)))

```

表 6.2: 機械手臂兩種套件改變隱藏層的類神經網路模型好壞比較

	<i>MLPRegressor</i>		<i>Neurolab.newff</i>	
	RMSE	SSE	RMSE	SSE
Hidden Layer = 20	0.061	0.01	0.008	<b>0.031</b>
Hidden Layer = 40	0.0296	0.01	0.007	0.044
Hidden Layer = 60	<b>0.023</b>	<b>0.01</b>	<b>0.0077</b>	0.032
Hidden Layer = 100	0.031	0.01	0.0078	0.279

根據表 6.2 觀察兩個套件在樣本數相同的情況下改變隱藏層數的 RMSE 與 SSE，假設一開始從均勻分配抽出的樣本數為 1000，則套件 *MLPRegressor* 在不同隱藏層數下的 RMSE 都比套件 *Neurolab.newff* 高，但 SSE 却是 *neuralab.newff* 都較低，另外，設置隱藏層數為 60 時兩個套件的 RMSE 與 SSE 都最小，因此判定在樣本數為 1000 時隱藏層數設 60 的模型表現最佳。接著根據表 6.3 觀察在隱藏層數都是 60 時改變樣本數兩個套件的模型表現，此時同樣也是一個套件在 RMSE 的表現較好，另一個套件在 SSE 的表現最好，因此兩種套件互有優劣，另外，在隱藏層設為 40 時樣本數為 3000 的模型表現最好。

表 6.3: 機械手臂兩種套件改變樣本數的類神經網路模型好壞比較

	<i>MLPRegressor</i>		<i>Neurolab.newff</i>	
	RMSE	SSE	RMSE	SSE
n = 1000	0.0342	0.01	0.0075	0.031
n = 2000	<b>0.0259</b>	0.01	0.005	0.006
n = 3000	0.0374	<b>0.01</b>	0.004	<b>0.0038</b>
n = 5000	0.04	0.01	<b>0.003</b>	0.01

最後，根據表 6.4 觀察同時改變樣本數與隱藏層的模型表現，套件 *Neoralab.newff* 在 RMSE 與 SSE 的表現都比套件 *MLPRegressor* 還要好，另外，設置隱藏層為 60 樣本數 2000 的模型是表現最佳的類神經網路模型。

表 6.4: 機械手臂兩種套件同時改變樣本數與隱藏層的類神經網路模型好壞比較

	<i>MLPRegressor</i>		<i>Neurolab.newff</i>	
	RMSE	SSE	RMSE	SSE
Hidden Layer = 20, n = 2000	0.0513	0.03	0.006	0.11
Hidden Layer = 20, n = 5000	0.0463	0.03	0.007	0.03
Hidden Layer = 60, n = 2000	<b>0.0283</b>	<b>0.03</b>	0.005	<b>0.013</b>
Hidden Layer = 60, n = 3000	0.03	0.03	0.004	0.035
Hidden Layer = 60, n = 5000	0.034	0.03	<b>0.0035</b>	0.016
Hidden Layer = 40, n = 2000	0.05	0.03	0.0054	0.015

## 6.3 圖形識別

除了在連續型資料上的應用，類神經網路模型 ( ANN ) 也常常應用在圖形識別 ( Pattern Recognition ) 上，如圖 6.7，該筆資料集含有手寫的 0-9 的數字，由於手寫習慣都不同，也會有潦草跟工整的區別，因此如何成功判別該手寫數字是哪個數字也成為一大挑戰，而類神經網路即可解決此問題。



圖 6.7: 手寫數字範例圖

以下之程式碼呈現了如何利用 Python 讀入資料進行展示。此筆資料的原始資料為  $1000 \times 784$  的圖片影像，即共 1000 筆資料，每一張圖片的像素為  $28 \times 28$ ，首先將每張圖片的像素從矩陣展開成  $1 \times 784$  的向量，接著取出 600 張圖片，並設置  $n = 20$  且  $m = 30$  來展示解析度為  $20 \times 30$  的圖片矩陣。如下之迴圈設置矩陣 M，將前述展開之  $1 \times 784$  向量重新 reshape 回  $28 \times 28$  的矩陣並一一放入矩陣 M 中。

```

n = 20
M = 30
sz = np.sqrt(X.shape[1]).astype("int")
M = np.zeros((m*sz, n*sz)), A = X[:m*n,:]
for i in range(m) :
    for j in range(n) :
        M[i*sz: (i+1)*sz, j*sz:(j+1)*sz] = A[i*n+j,:,:].reshape(
            sz, sz)
    
```

觀察過手寫數字資料後，接著利用類神經網路 ( ANN ) 來建立分類模型，此處使用 sklearn 套件中的 *MLPClassifier* 來進行訓練，以下程式碼展現模型建立的方式，首

先建立一個 `MLPClassifier()` 的學習器，與 `MLPRegression` 相同，其中 `solver` 採用 `adam` 演算法，另外，有時候訓練模型時可能會因為在某個區域極小值卡住而導致結果不理想，因此會設置 `random_state = 0` 紿予類神經網路不同的初始值，並固定隨機亂數產生的位置，若希望產生的位置不同時，則不要給固定的數字。在完成訓練後，可以嘗試繪製混淆矩陣來觀察訓練模型的測試資料準確度，其範例如圖 6.8，其中採用了隱藏層數量為 20 來進行訓練。

```
v = 20, hidden_layers = (v,)
solver = "adam"
clf = MLPClassifier(max_iter = 10000, solver = solver,
hidden_layer_sizes = hidden_layers, verbose = True,
activation = "logistic", tol = 1e-6, random_state = 0)
```

根據圖 6.8 可知，此類神經網路模型 (ANN) 的測試資料準確度為 86 %，其中手寫數字 4 最常被誤判成手寫數字 6、手寫數字 2 也最常被誤判成 6、數字 1 跟 5 較常被誤判成 8，數字 7 則較常被誤判成 9。

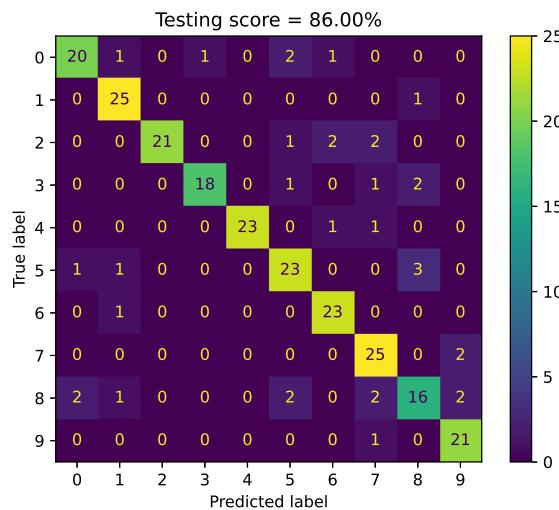


圖 6.8: 1000 筆資料集之類神經網路分類模型的測試資料混淆矩陣

接著與機械手臂相同，嘗試觀察不同隱藏層與不同樣本數對類神經網路模型的分類準確度的影響，在上述範例中使用的是資料集 Digits train 中的 1000 筆手寫資料集再進行分割，而在下面的試驗中將同時利用 Digits train 10000 資料集中的 10000 筆手寫資料來建立類神經網路模型，並比較改變隱藏層數時準確度的變化，其結果如表 6.5。另外，此處亦展示 10000 筆手寫資料、20 個隱藏層的分類模型混淆矩陣，如圖 6.9。

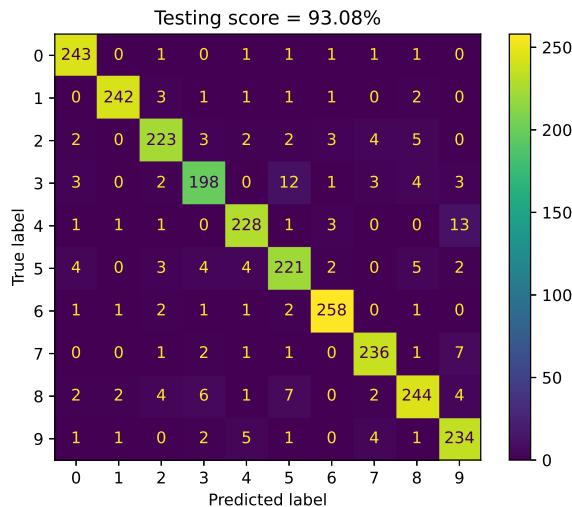


圖 6.9: 10000 筆類神經網路分類模型的測試資料混淆矩陣

根據圖 6.9 可以明顯發現，手寫數字 3 最常被分類到數字 5，而手寫數字 4 最常被分類到數字 9，數字 7 也很常被分類成數字 9，此結果與樣本數量為 1000 時的誤判不太相同，在 1000 筆資料時，數字 4 最常被分類到數字 6。

表 6.5: 兩筆資料集改變隱藏層之測試資料分類準確度

	n=1000	n=10000
Hidden Layer = 20	86%	93.08%
Hidden Layer = 40	91.2%	93.76%
Hidden Layer = 60	90.4%	95%
Hidden Layer = 100	88.8%	95.96%
Hidden Layer = 100	92.8%	96.2%

根據表 6.5 可以觀察到，在樣本數為 1000 筆跟 10000 筆時隱藏層為 100 的準確度都最高，另外，樣本數為 10000 筆時準確度似乎隨著隱藏層數增加而增加。

## 6.4 結論

在此節中探討了類神經網路 ( ANN ) 的理論以及其在連續型資料與類別型資料的應用，期望能幫助了解深度學習的基礎理論以及如何成功利用 Python 來對機械手臂以及圖像識別進行分析，並了解不同的參數設定對於模型效果的影響。



## 第 7 章

# 蒙地卡羅模擬實驗：多種分類方法的評比

在前面的章節中，分別對簡單線性迴歸 ( Simple Linear Regression )、加廣迴歸模型 ( Augmented Regression )、羅吉斯迴歸 ( Logistic Regression ) 三種決定性 ( Deterministic ) 機器學習方法與線性判別分析 ( LDA )、二次判別分析 ( QDA )、K-近鄰演算法三種機率性 ( Probabilistic ) 機器學習方法跟類神經網路模型的分類問題進行了探討，也大致了解了不同方法的理論以及分類上可能遇到的問題以及特性，但是卻未同時對決定性機器學習方法、機率性機器學習方法與類神經網路的分類準確度進行比較，因此在此章節中將模擬生成多個兩群與三群的雙變量常態資料來同時比較這七種方法的分類準確度。另外，本章節亦會嘗試生成非常態的資料來進行分類問題之比較。

## 7.1 七種分類方法介紹

### 7.1.1 簡單線性迴歸簡述

線性迴歸 ( Linear Regression ) 是一種常見的監督式學習方法，是一種常見的預測定量響應變量 ( Response ) 的工具，它假定了  $X$  與  $Y$  之間存在線性關係，在數學上將這種關係記為：

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \varepsilon_i \quad (7.1)$$

在上式 (7.1) 中假設存在兩個預測變量  $X_1$  與  $X_2$ ，假設有  $N$  筆已知的輸入與輸出資料 ( $[x_1(i) \ x_2(i)], y(i)$ )，利用最小平方法期望使觀察值與估計值的誤差越小越好。其計算方法如下矩陣：

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} = \begin{bmatrix} 1 & X_1(1) & X_2(1) \\ 1 & X_1(2) & X_2(2) \\ \vdots & \vdots & \vdots \\ 1 & X_1(n) & X_2(n) \end{bmatrix} \begin{bmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_n \end{bmatrix}$$

透過上述矩陣即可求得最佳解如式 (7.2)，式 (7.2) 假設反矩陣  $(X^T X)^{-1}$  存在，且每個輸出值  $y(i)$  根據其類別分類至 0 或 1。

$$\hat{\beta} = (X^T X)^{-1} X^T y \quad (7.2)$$

### 7.1.2 加廣迴歸簡述

加廣迴歸模型是一種非線性的迴歸模型，它透過擴充簡單線性迴歸模型來試圖得到較好的分類準確度，其重新假設迴歸模型如式 (7.3)：

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_1 X_2 + \beta_4 X_1^2 + \beta_5 X_2^2 \quad (7.3)$$

### 7.1.3 羅吉斯迴歸簡述

假設存在兩個類別來解釋羅吉斯迴歸的簡單原理。在線性迴歸中利用  $p(x) = \beta_0 + \beta_1 X_1 + \beta_2 X_2$  來表示機率，而在羅吉斯迴歸中，則使用羅吉斯函數式 (7.4) 結合最大概似估計 (MLE) 方法來擬合模型。

$$p(X) = \frac{e^{\beta_0 + \beta_1 X_1 + \beta_2 X_2}}{1 + e^{\beta_0 + \beta_1 X_1 + \beta_2 X_2}} \quad (7.4)$$

接著將式 (7.4) 改寫成式 (7.5)，並將其稱之為勝算比 (odd)。

$$\frac{p(X)}{1 - p(X)} = e^{\beta_0 + \beta_1 X_1 + \beta_2 X_2} \quad (7.5)$$

若將式 (7.5) 取 log，則將其稱為對數勝算比 (log-odd)，寫作式 (7.6)

$$\log\left(\frac{p(X)}{1-p(X)}\right) = \beta_0 + \beta_1 X_1 + \beta_2 X_2 \quad (7.6)$$

### 7.1.4 線性判別分析簡述

在線性判別分析 (Linear Discriminant Analysis) 中，試圖找出預測變數  $X$  在不同的反應變數  $G$  (不同類別) 的分配，並利用貝氏定理去估計  $P(G = k|X = x)$ 。假設先驗分配 (prior probability) ( $P(G = k)$ ) 代表隨機從類別  $k$  抽出觀察值的機率，而後驗分配  $f_k(x) = P(X = x|G = k)$  代表觀察值來自類別  $k$  時  $X$  的機率密度函數，則透過貝氏定理可知：

$$P(G = k|X = x) = \frac{P(G = k)P(X = x|G = k)}{\sum_{l=1}^k P(G = l)P(x = X|g = l)} \quad (7.7)$$

其中，若使後驗分配  $P(X = x|G = k)$  最大，則此分類的分類誤差 (error rate) 最少。

假設  $X \sim N(\mu, \Sigma)$ ，且其機率密度函數如式 (7.8)：

$$f(x) = \frac{1}{(2\pi)^{p/2} |\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(x - \mu)^T \Sigma^{-1} (x - \mu)\right) \quad (7.8)$$

而線性判別分析 (LDA) 會將觀察值  $X = x$  分類至使式 (7.9) 的值最大的第  $k$  類。

$$\delta_k(x) = x^T \Sigma^{-1} \mu_k - \frac{1}{2} \mu_k^T \Sigma^{-1} \mu_k + \ln P(G = k) \quad (7.9)$$

透過以上式 (7.9) 即可得到線性判別分析的分群結果。

### 7.1.5 二次判別分析簡述

在線性判別分析中假設母體服從常態分配且每個類別的共變異數相同 (共變異矩陣一樣)，但在實務上往往很難滿足此假設，而二次判別分析中完善了此一缺點。二次判別分析假設了每個類別都有各自的共變異數，因此式 (7.9) 可被改寫成如式 (7.10)：

$$\begin{aligned} \delta_k(x) &= -\frac{1}{2}(x - \mu_k)^T \Sigma_k^{-1} (x - \mu_k) - \frac{1}{2} \ln |\Sigma_k| + \ln P(G = k) \\ &= -\frac{1}{2}x^T \Sigma_k^{-1} x + x^T \Sigma_k^{-1} \mu_k - \frac{1}{2} \mu_k^T \Sigma_k^{-1} \mu_k - \frac{1}{2} \ln |\Sigma_k| + \ln P(G = k) \end{aligned} \quad (7.10)$$

其中，二次判別分析 ( QDA ) 需估計  $\Sigma_k$ ,  $\mu_k$  以及  $P(G = k)$  並將觀察值  $X = x$  分類至使式 (7.10) 的值最大的類別  $k$ 。

另外，二次判別分析的分界線為變數的二次方，被寫作如式 (7.11)

$$\{x | \delta_k(x) = \delta_l(x)\} \quad (7.11)$$

### 7.1.6 K-近鄰演算法簡述

雖然在機器學習中有幾種常見的方法是利用貝氏定理來求得條件機率  $P(Y|X = x)$ ，其中  $x$  與  $y$  分別是反應變數與預測變數，但在真實世界中，很難知道此條件機率的值，因此利用貝氏定理來分類有時是不可行的，而有一些方法便嘗試去估計  $P(Y|X = x)$ ，其中包含 K-近鄰演算法 ( KNN )。K-近鄰演算法透過給定  $K$  值以及任一測試點  $x_0$ ，觀察離該點  $x_0$  最近的  $K$  個點 ( 即  $N_k(x)$  )，並將這些鄰近的  $K$  筆資料所對應的  $y$  值取平均，得到式 (7.12)：

$$\hat{y} = Ave(y_i | x_i \in N_k(x)) = \frac{1}{K} \sum_{x_i \in N_k(x)} y_i \quad (7.12)$$

且其給定  $x$  時  $y$  的條件機率如式 (7.13)：

$$P(Y = j | X = x_0) = \frac{1}{K} \sum_{i \in N_0} I(y_i = j) \quad (7.13)$$

### 7.1.7 類神經網路模型簡述

類神經網路是一種模仿大腦運作的機器學習方式，透過大量的人工神經元連結進行計算，並能自行透過學習更多資訊進而對模型進行改進，其結構一般包含輸入層 ( Input )、隱藏層 ( Hidden Layer ) 以及輸出層 ( Output )，其中隱藏層的數量是可以進行改變的，一般會遵循以下步驟：

- 建立包含未知參數的函數

假設函數為  $y = b + wx_1$ ，其中  $x_1$ : views on 2/25、 $y$ : views on 2/26， $w$  ( weight ) 與  $b$  ( bias ) 為模型之未知參數，其從已有資料中進行學習。

- 定義損失函數 ( Loss Function )

定義函數的輸入 ( Input ) 為未知參數  $b$ 、 $w$ ，而損失函數 ( Loss Function ) 則用來表示在將 input 丟入函數後，此函數的預測效果有多好，其函數可寫作如式 (7.14)。

$$Loss = \frac{1}{N} \sum_n e_n \quad (7.14)$$

其中  $e = (y - \hat{y})^2$ 。

- 利用不同的未知參數計算不同的損失函數並畫圖
- 最佳化模型 ( Optimization )

最佳化模型有很多方法，其中有一個常見的方法是梯度下降 ( Gradient Descent )，嘗試求得合適的未知參數  $w$ 、 $b$  使得損失函數的值最小，在此文不進行過多討論。

在此步驟中，所有的模型調整全部建立在線性模型 ( Linear Model ) 之上進行調整，但線性模型往往太過簡單，就算選出了具有最小損失函數的模型，其依舊被初始設置的線性模型所限制 ( model bias )。而如何設置更複雜的模型？Sigmoid Function 協助達成了此目的，其被表示如式 (7.15)。

$$sigmoid = \frac{1}{1 + e^{-x}} \quad (7.15)$$

故將新的預測模型寫作式 (7.16)。

$$y = c \frac{1}{1 + e^{-(b+wx_1)}} = c \times sigmoid(b + wx_1) \quad (7.16)$$

式 (7.16) 中的  $w$  控制函數坡度、 $b$  控制左右移動， $c$  則可以改變函數的高度，若是具有兩個隱藏層，則利用此原理可以將類神經網路模型寫作式 (7.17)。

$$\hat{y} = b_k^2 + \sum_{i=1}^q w_{ki}^2 g\left(\sum_{j=1}^p w_{ij}^1 x_j + b_i^1\right) + b_k^2, 1 \leq k \leq r \quad (7.17)$$

其中函式  $g(\bullet)$  為映射函數 ( Sigmoid Function )，其數學式如式 (7.18)。

$$g(z) = c_1 \frac{1 - e^{-c_2 z}}{1 + e^{-c_2 z}} \quad (7.18)$$

在式 (7.17) 中， $q$  為映射函數 ( Sigmoid Function ) 的數量、 $p$  為輸入的特徵 ( feature ) 的數量、 $w_{ij}^1$  與  $b_i^1$  代表第一個隱藏層的第  $i$  個神經元與第  $j$  個 input 的權重 ( Weight ) 跟偏誤 ( Biases )，而  $w_{ki}^2$  與  $b_k^2$  則是第二層隱藏層的第  $k$  個神經元與前一隱藏層的第  $i$  個輸出的權重與偏誤，為了使模型最佳化，期望能得到使此模型的損失函數 ( Loss Function ) 最小的未知參數。

以上即是類神經網路的基本原理，但據此亦可知雖然其原理簡單有效，如果資料為高維資料時，如何求得使損失函數最小的未知參數也變得困難，如何定義需要多少隱藏層也成為挑戰。

## 7.2 七種機器學習方法對兩群組資料的分類問題

在此章節中將分別建立兩群組的雙變量常態資料跟三群組的雙變量常態資料，利用上面所述的七種機器學習方法來建立模型進行分類並比較其訓練資料準確度與測試資料準確度。與上面幾個章節相同，本節將模擬各種不同樣本數、不同平均向量與不同共變異數的資料集來進行比較，首先介紹模擬兩群組雙變量常態資料的情況，並嘗試改變其樣本數大小來觀察其訓練資料準確度與測試資料準確度。

### 7.2.1 兩群組更改樣本數大小的分類問題

此節設置雙變量常態資料的參數如下，其散布圖亦呈現於圖 7.1。

$$\mu_1 = \begin{bmatrix} 2 \\ 2 \end{bmatrix}, \mu_2 = \begin{bmatrix} 4 \\ 1 \end{bmatrix}$$

$$\Sigma_1 = \begin{bmatrix} 1 & 0.2 \\ 0.2 & 1 \end{bmatrix}, \Sigma_2 = \begin{bmatrix} 1 & 0.2 \\ 0.2 & 1 \end{bmatrix}$$

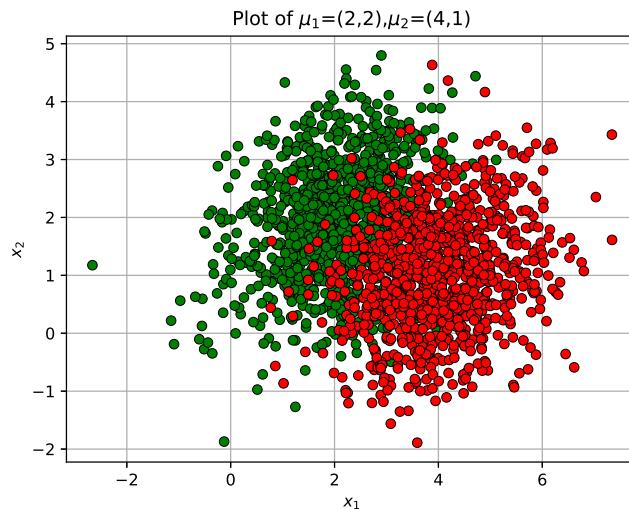


圖 7.1: 比較七種方法改變樣本數實驗的兩群組雙變量常態資料散布圖

根據圖 7.1 可以觀察到此兩群組雙變量常態資料的關係，接著嘗試利用七種機器學習方法建立分類模型，其中 KNN 模型的 K 值在此章節中都設定為 10，並利用 Bootstrapping 方法對資料抽樣 100 次並分別求其誤差，再將 100 次的誤差取平均後即可得到如表 7.1 與表 7.2 的結果。

表 7.1: 七種機器學習方法改變樣本數的兩群組訓練資料分類錯誤率

	Regression	Augmented	Logistic	LDA	QDA	KNN	ANN
$n_1 = n_2 = 200$	0.146	0.142	0.147	0.146	0.147	<b>0.116</b>	0.145
$n_1 = n_2 = 500$	0.120	0.116	0.120	0.120	0.115	<b>0.096</b>	0.118
$n_1 = n_2 = 1000$	0.112	0.111	0.112	0.112	0.116	<b>0.099</b>	0.114
$n_1 = 1000, n_2 = 500$	0.088	0.098	0.091	0.087	0.09	<b>0.075</b>	0.087

表 7.2: 七種機器學習方法改變樣本數的兩群組測試資料分類錯誤率

	Regression	Augmented	Logistic	LDA	QDA	KNN	ANN
$n_1 = n_2 = 200$	0.135	<b>0.134</b>	0.135	0.135	0.15	0.15	0.137
$n_1 = n_2 = 500$	0.1195	<b>0.1175</b>	0.1188	0.1195	0.115	0.14	0.118
$n_1 = n_2 = 1000$	0.112	0.112	0.112	0.112	<b>0.085</b>	0.099	0.115
$n_1 = 1000, n_2 = 500$	0.086	0.097	0.089	0.085	<b>0.073</b>	0.087	0.086

根據表 7.1 與表 7.2 可知，雖然 KNN 模型無論在樣本數為多少時訓練誤差都是最小的，似乎應該是表現最好的分類模型，但是測試誤差卻是加廣迴歸模型 ( Augmented Regression ) 與 QDA 模型在不同樣本數下的表現最佳。

## 7.2.2 兩群組更改群平均的分類問題

接著觀察改變群平均時七種機器學習方法的訓練誤差與測試誤差，其雙變量常態資料的參數亦呈現如下，散布圖則如圖 7.2，此兩組雙變量常態資料會與圖 7.1 的散布圖資料一起比較，最後會得到表 7.3 與 7.4 的結果。

$$\mu_1 = \begin{bmatrix} 2 \\ 2 \end{bmatrix}, \mu_2 = \begin{bmatrix} 3 \\ 3 \end{bmatrix}$$

$$\mu'_1 = \begin{bmatrix} 2 \\ 6 \end{bmatrix}, \mu'_2 = \begin{bmatrix} 5 \\ 9 \end{bmatrix}$$

$$\Sigma_1 = \Sigma'_1 = \begin{bmatrix} 1 & 0.2 \\ 0.2 & 1 \end{bmatrix}, \Sigma_2 = \Sigma'_2 = \begin{bmatrix} 1 & 0.2 \\ 0.2 & 1 \end{bmatrix}$$

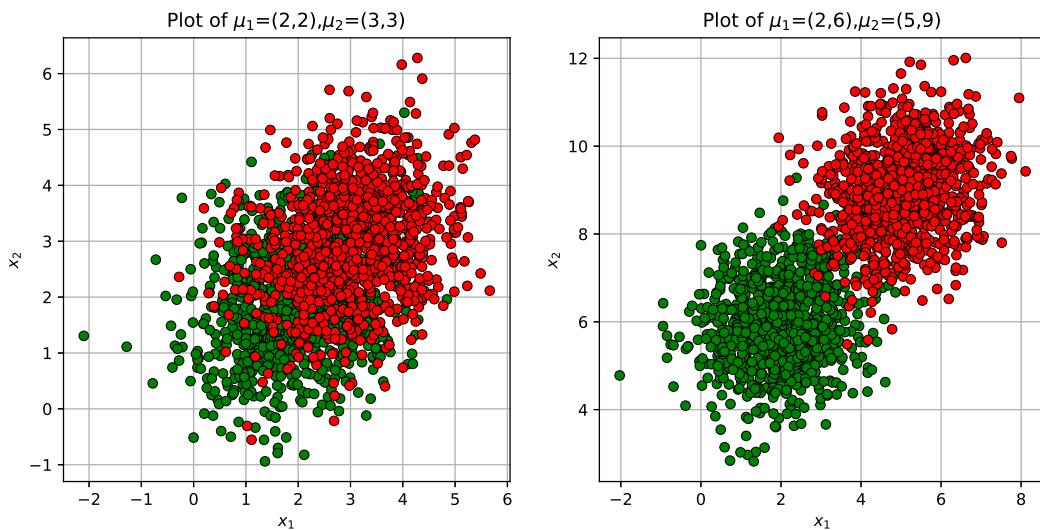


圖 7.2: 比較七種方法改變群平均實驗的兩群組雙變量常態資料散布圖

根據表 7.3 與表 7.4 可知，平均數向量的改變並不影響 KNN 模型是七種方法中訓練資料準確度最高的模型，但是與樣本數的實驗相同，KNN 模型並非測試資料中準確度最高的模型，反而 QDA 模型似乎與樣本數實驗相同，都是七種方法中表現最好的模型。

表 7.3: 七種機器學習方法改變群平均的兩群組訓練資料分類錯誤率

	Regression	Augmented	Logistic	LDA	QDA	KNN	ANN
$\mu_1 = (2, 2), \mu_2 = (4, 1)$	0.112	0.111	0.112	0.112	0.116	<b>0.099</b>	0.114
$\mu_1 = (2, 2), \mu_2 = (3, 3)$	0.274	0.273	0.274	0.274	0.276	<b>0.211</b>	0.274
$\mu_1 = (2, 6), \mu_2 = (5, 9)$	0.032	0.033	0.031	0.032	0.031	<b>0.027</b>	0.032

表 7.4: 七種機器學習方法改變群平均的兩群組測試資料分類錯誤率

	Regression	Augmented	Logistic	LDA	QDA	KNN	ANN
$\mu_1 = (2, 2), \mu_2 = (4, 1)$	0.112	0.112	0.112	0.112	<b>0.085</b>	0.099	0.115
$\mu_1 = (2, 2), \mu_2 = (3, 3)$	0.276	0.276	0.276	0.276	<b>0.265</b>	0.27	0.276
$\mu_1 = (2, 6), \mu_2 = (5, 9)$	<b>0.031</b>	0.032	<b>0.031</b>	<b>0.031</b>	0.037	0.04	<b>0.031</b>

### 7.2.3 兩群組更改群共變異數的分類問題

接著來看改變共變異數時七種機器學習方法的分類效果，其參數設定如下所示，此小節中除了比較圖 7.1 中的雙變量常態資料外，也比較另外四種雙變量常態資料，其散布圖如圖 7.3。

$$\mu_1 = \mu'_1 = \mu''_1 = \begin{bmatrix} 2 \\ 2 \end{bmatrix}, \mu_2 = \mu'_2 = \mu''_2 = \begin{bmatrix} 4 \\ 1 \end{bmatrix}$$

$$\Sigma_1 = \begin{bmatrix} 1 & 0.5 \\ 0.5 & 1 \end{bmatrix}, \Sigma_2 = \begin{bmatrix} 1 & 0.5 \\ 0.5 & 1 \end{bmatrix}$$

$$\Sigma'_1 = \begin{bmatrix} 1 & 0.2 \\ 0.2 & 1 \end{bmatrix}, \Sigma'_2 = \begin{bmatrix} 1 & 0.8 \\ 0.8 & 1 \end{bmatrix}$$

$$\Sigma''_1 = \begin{bmatrix} 1 & 0.9 \\ 0.9 & 1 \end{bmatrix}, \Sigma''_2 = \begin{bmatrix} 1 & 0.3 \\ 0.3 & 1 \end{bmatrix}$$

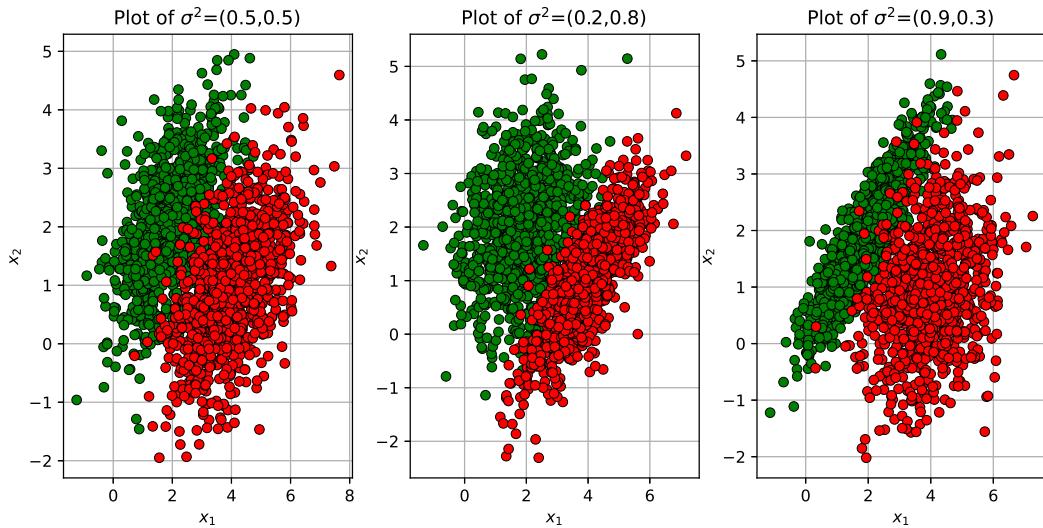


圖 7.3: 比較七種方法改變群共變異數實驗的兩群組雙變量常態資料散布圖

表 7.5: 七種機器學習方法改變群共變異數的兩群組訓練資料分類錯誤率

	Regression	Augmented	Logistic	LDA	QDA	KNN	ANN
$\sigma^2 = (0.2, 0.2)$	0.112	0.111	0.112	0.112	0.116	<b>0.099</b>	0.114
$\sigma^2 = (0.5, 0.5)$	0.032	0.033	0.031	0.032	0.031	<b>0.027</b>	0.032
$\sigma^2 = (0.2, 0.8)$	0.053	0.052	0.053	0.053	0.057	<b>0.046</b>	0.052
$\sigma^2 = (0.9, 0.3)$	0.052	0.032	0.032	0.052	<b>0.027</b>	<b>0.027</b>	0.031

表 7.6: 七種機器學習方法改變群共變異數的兩群組測試資料分類錯誤率

	Regression	Augmented	Logistic	LDA	QDA	KNN	ANN
$\sigma^2 = (0.2, 0.2)$	0.112	0.112	0.112	0.112	<b>0.085</b>	0.099	0.115
$\sigma^2 = (0.5, 0.5)$	<b>0.031</b>	<b>0.032</b>	<b>0.031</b>	<b>0.031</b>	0.037	0.04	<b>0.031</b>
$\sigma^2 = (0.2, 0.8)$	0.054	0.050	0.052	0.054	<b>0.027</b>	0.04	0.05
$\sigma^2 = (0.9, 0.3)$	0.052	0.032	0.052	0.027	<b>0.026</b>	0.0958	0.12

圖 7.3 是實驗中的雙變量常態資料散布圖，而表 7.5 與表 7.6 則是七種機器學習方法的分類準確度，根據兩個表格可知，KNN 模型依舊是訓練模型中表現最好的模型，但是 KNN 模型的測試資料準確度並非最高，QDA 模型則是整體表現最好的分類模型。至此本節完成了對兩群組雙變量常態的分類效果之觀察，下一節中則將繼續觀察七種機器學習方法在三群組雙變量常態資料的分類準確度。

### 7.3 七種機器學習方法對三群組資料的分類問題

在此章節中將分別針對七種機器學習方法建立分類模型，並比較這七種方法的訓練資料誤差與測試資料誤差，判斷出哪個模型在三群組分類中的表現是最好的，此節與前述章節相同，資料都是來自於雙變量常態母體資料，並透過三種實驗，改變樣本數大小、改變平均數與改變共變異數來比較這七種機器學習方法。

#### 7.3.1 三群組更改樣本數大小的分類問題

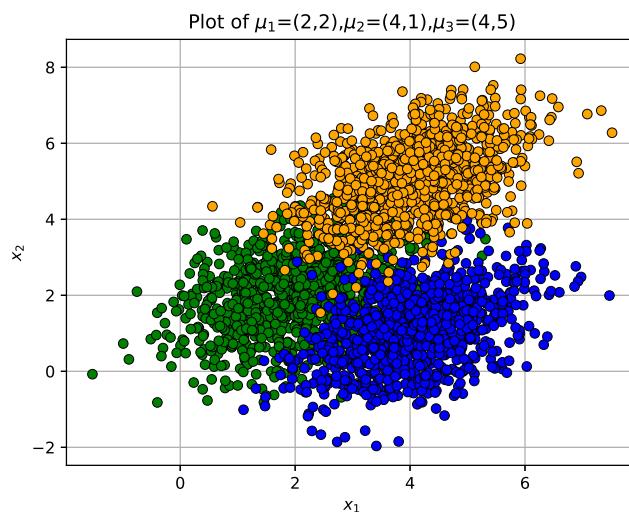


圖 7.4: 比較七種方法改變群樣本數實驗的三群組雙變量常態資料散佈圖

$$\mu_1 = \begin{bmatrix} 2 \\ 2 \end{bmatrix}, \mu_2 = \begin{bmatrix} 4 \\ 1 \end{bmatrix}, \mu_3 = \begin{bmatrix} 4 \\ 5 \end{bmatrix}$$

$$\Sigma_1 = \begin{bmatrix} 1 & 0.2 \\ 0.2 & 1 \end{bmatrix}, \Sigma_2 = \begin{bmatrix} 1 & 0.2 \\ 0.2 & 1 \end{bmatrix}, \Sigma_3 = \begin{bmatrix} 1 & 0.2 \\ 0.2 & 1 \end{bmatrix}$$

首先觀察改變樣本數大小造成的分類準確度變化，圖 7.4 是此實驗中用到的雙變量常態資料的散佈圖，其資料的參數亦呈現於上方。

表 7.7 與表 7.8 即為改變樣本數時七種機器學習的訓練誤差與測試誤差的比較，從表中可以知道，KNN 模型是訓練資料中誤差最小表現最好的模型，但是與兩群組實驗相同的是 QDA 模型才是七個模型中綜合表現最好的模型。

表 7.7: 七種機器學習方法改變樣本數的三群組訓練資料分類錯誤率

	Regression	Augmented	Logistic	LDA	QDA	KNN	ANN
$n_1 = n_2 = n_3 = 200$	0.467	0.480	0.366	0.365	0.356	<b>0.262</b>	0.368
$n_1 = n_2 = n_3 = 500$	0.477	0.489	0.390	0.385	0.372	<b>0.286</b>	0.384
$n_1 = n_2 = n_3 = 1000$	0.481	0.493	0.408	0.405	0.401	<b>0.288</b>	0.404
$n_1 = 1000, n_2 = 500, n_3 = 200$	0.247	0.226	0.294	0.202	0.199	<b>0.165</b>	0.203

表 7.8: 七種機器學習方法改變樣本數的三群組測試資料分類錯誤率

	Regression	Augmented	Logistic	LDA	QDA	KNN	ANN
$n_1 = n_2 = n_3 = 200$	0.463	0.475	0.383	0.401	0.433	<b>0.358</b>	0.397
$n_1 = n_2 = n_3 = 500$	0.474	0.486	0.396	0.399	<b>0.363</b>	0.445	0.397
$n_1 = n_2 = n_3 = 1000$	0.477	0.489	0.413	0.415	<b>0.388</b>	0.437	0.415
$n_1 = 1000, n_2 = 500, n_3 = 200$	0.248	0.226	0.304	<b>0.203</b>	0.218	0.247	0.204

### 7.3.2 三群組更改群平均的分類問題

$$\mu_1 = \begin{bmatrix} 3 \\ 3 \end{bmatrix}, \mu_2 = \begin{bmatrix} 4 \\ 4 \end{bmatrix}, \mu_3 = \begin{bmatrix} 5 \\ 5 \end{bmatrix}$$

$$\mu'_1 = \begin{bmatrix} 1 \\ 6 \end{bmatrix}, \mu'_2 = \begin{bmatrix} 2 \\ 5 \end{bmatrix}, \mu'_3 = \begin{bmatrix} 5 \\ 5 \end{bmatrix}$$

$$\Sigma_1 = \Sigma'_1 = \begin{bmatrix} 1 & 0.4 \\ 0.4 & 1 \end{bmatrix}, \Sigma_2 = \Sigma'_2 = \begin{bmatrix} 1 & 0.4 \\ 0.4 & 1 \end{bmatrix}, \Sigma_3 = \Sigma'_3 = \begin{bmatrix} 1 & 0.4 \\ 0.4 & 1 \end{bmatrix}$$

接著來觀察三群組雙變量常態資料改變群平均實驗的七種機器學習方法的分類效果，其使用的資料如圖 7.4 與 7.5，其資料參數亦呈現如上。

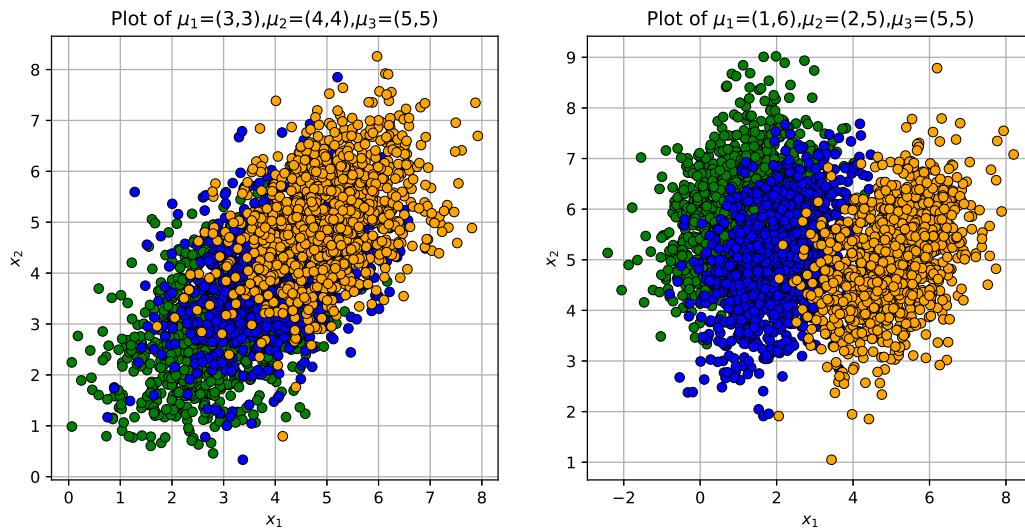


圖 7.5: 比較七種方法改變群平均實驗的三群組雙變量常態資料散布圖

表 7.9: 七種機器學習方法改變群平均的三群組訓練資料分類錯誤率

	Regression	Augmented	Logistic	LDA	QDA	KNN	ANN
$\mu_1 = (2, 2), \mu_2 = (4, 1), \mu_3 = (4, 5)$	0.481	0.493	0.408	0.405	0.401	<b>0.288</b>	0.404
$\mu_1 = (3, 3), \mu_2 = (4, 4), \mu_3 = (5, 5)$	0.608	0.598	0.512	0.515	0.505	<b>0.384</b>	0.516
$\mu_1 = (1, 6), \mu_2 = (2, 4), \mu_3 = (5, 5)$	0.477	0.486	0.397	0.398	0.395	<b>0.283</b>	0.397

根據表 7.9 與表 7.10 可知，KNN 模型是訓練資料中表現最好的模型，羅吉斯迴歸 (Logistic Regression) 則是測試誤差綜合最小的模型。

表 7.10: 七種機器學習方法改變群平均的三群組測試資料分類錯誤率

	Regression	Augmented	Logistic	LDA	QDA	KNN	ANN
$\mu_1 = (2, 2), \mu_2 = (4, 1), \mu_3 = (4, 5)$	0.477	0.489	0.413	0.415	<b>0.388</b>	0.437	0.415
$\mu_1 = (3, 3), \mu_2 = (4, 4), \mu_3 = (5, 5)$	0.609	0.599	<b>0.514</b>	0.521	0.543	0.565	0.522
$\mu_1 = (1, 6), \mu_2 = (2, 4), \mu_3 = (5, 5)$	0.478	0.488	<b>0.404</b>	0.412	0.422	0.438	0.410

### 7.3.3 三群組更改群共變異數的分類問題

最後，此章節透過改變共變異數來比較七種機器學習方法的訓練資料誤差與測試資料誤差，其使用的雙變量常態資料的參數如下，圖 7.6 則是這些資料的散布圖。

$$\mu_1 = \mu'_1 = \mu''_1 = \begin{bmatrix} 2 \\ 2 \end{bmatrix}, \mu_2 = \mu'_2 = \mu''_2 = \begin{bmatrix} 4 \\ 1 \end{bmatrix}, \mu_3 = \mu'_3 = \mu''_3 = \begin{bmatrix} 4 \\ 5 \end{bmatrix}$$

$$\Sigma_1 = \begin{bmatrix} 1 & 0.4 \\ 0.4 & 1 \end{bmatrix}, \Sigma_2 = \begin{bmatrix} 1 & 0.4 \\ 0.4 & 1 \end{bmatrix}, \Sigma_3 = \begin{bmatrix} 1 & 0.4 \\ 0.4 & 1 \end{bmatrix}$$

$$\Sigma'_1 = \begin{bmatrix} 1 & 0.1 \\ 0.1 & 1 \end{bmatrix}, \Sigma'_2 = \begin{bmatrix} 1 & 0.7 \\ 0.7 & 1 \end{bmatrix}, \Sigma'_3 = \begin{bmatrix} 1 & 0.9 \\ 0.9 & 1 \end{bmatrix}$$

$$\Sigma''_1 = \begin{bmatrix} 1 & 0.9 \\ 0.9 & 1 \end{bmatrix}, \Sigma''_2 = \begin{bmatrix} 1 & 0.3 \\ 0.3 & 1 \end{bmatrix}, \Sigma''_3 = \begin{bmatrix} 1 & 0.1 \\ 0.1 & 1 \end{bmatrix}$$

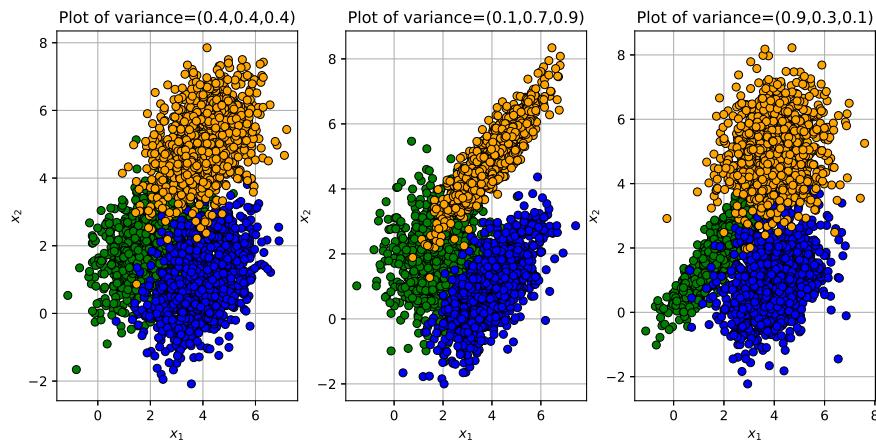


圖 7.6: 改變群共變異數實驗的三群組雙變量常態資料散布圖

表 7.11: 七種機器學習方法改變群共變異數的三群組訓練資料分類錯誤率

	Regression	Augmented	Logistic	LDA	QDA	KNN	ANN
$\sigma^2 = (0.2, 0.2, 0.2)$	0.481	0.493	0.408	0.405	0.401	<b>0.288</b>	0.404
$\sigma^2 = (0.4, 0.4, 0.4)$	0.032	0.033	0.031	0.032	0.031	<b>0.027</b>	0.032
$\sigma^2 = (0.1, 0.7, 0.9)$	0.447	0.442	0.369	0.371	0.368	<b>0.242</b>	0.367
$\sigma^2 = (0.9, 0.3, 0.1)$	0.450	0.469	0.371	0.372	0.370	<b>0.255</b>	0.371

根據表 7.11 與表 7.12 可觀察到七種方法的訓練誤差與測試誤差，其中無論共變異數是多少，KNN 模型都是訓練誤差最小的機器學習方法，而羅吉斯迴歸與 QDA 模型則是比較測試誤差時綜合表現較好的模型。

表 7.12: 七種機器學習方法改變群共變異數的三群組測試資料分類錯誤率

	Regression	Augmented	Logistic	LDA	QDA	KNN	ANN
$\sigma^2 = (0.2, 0.2, 0.2)$	0.477	0.489	<b>0.413</b>	0.415	0.388	0.437	0.415
$\sigma^2 = (0.4, 0.4, 0.4)$	<b>0.031</b>	0.032	<b>0.031</b>	<b>0.031</b>	0.037	0.04	<b>0.031</b>
$\sigma^2 = (0.1, 0.7, 0.9)$	0.448	0.443	0.376	0.383	<b>0.371</b>	0.378	0.38
$\sigma^2 = (0.9, 0.3, 0.1)$	0.453	0.473	0.377	0.380	<b>0.363</b>	0.393	0.380

## 7.4 結論

在此章節中依次重新簡述了前面章節所介紹到的七種機器學習方法，包含簡單線性迴歸 ( Simple Linear Regression )、加廣迴歸 ( Augmented Regression )、羅吉斯迴歸 ( Logistic Regression )、線性判別分析 ( LDA )、二次判別分析 ( QDA )、K-近鄰演算法 ( KNN ) 與類神經網路 ( ANN )，並就每種方法模擬生成兩群組與三群組雙變量常態資料建立分類模型並透過改變樣本數、平均數與共變異數來比較各自的訓練誤差與測試誤差，根據此章節的實驗可知，KNN 模型是所有方法中訓練誤差最小表現最好的模型，但是 QDA 模型才是其中測試誤差最低的模型。



# 第 8 章

## 參考文獻的使用與引用

### 8.1 初步觀念

參考文獻的引用分兩部分：一、內文的引用方式與呈現，二、參考文獻的排序呈現。不管是哪一部分都沒有統一的標準，隨期刊書籍自訂規範。在 L<sup>A</sup>T<sub>E</sub>X 裡，這些規範表現在 bibliography style 所引用的 bst 檔。這些檔案有些是公開的，可以直接引用，譬如，美國數學學會的 amsplain.bst、abbrvnat.bst 或 unsrtnat.bst。有些需要下載，如統計計算與模擬期刊 (Journal of Statistical Computation and Simulation) 的 gSCS.bst 檔 (如附檔)、統計軟體期刊 (Journal of Statistical Software) 的 jss.bst 檔 (如附檔)。文獻規範檔 (bibliography style) 一方面用來呈現不同刊物的需求與特色，一方面也能減輕寫作者的負擔，無需為符合不同刊物的規定，撰寫不同格式的參考文獻。

本文以 bibtex 的文獻資料庫方式呈現文獻的引用。除了運用 bibliography style 檔外，也增加一個知名的 package: natbib，可以選用文獻引用的呈現方式。讀者可以從本文原始檔下方的 \bibliographystyle 試用不同的 bst 檔，看看結果有何不同。

### 8.2 參考文獻的引用：作者與年份

The second class of MVN tests in this package examine the skewness and kurtosis of the data. Two approaches are adopted. One uses the combination of the univariate skewness and kurtosis for all marginals, as proposed by ?, and ?. The other approach considers multivariate skewness and kurtosis proposed by ?. ? and ? consider the MVN test statistics by Small as "among the most powerful" and "of practical importance," while ? consider Mardia's procedures, based on multivariate kurtosis, as among the commonly used tests of MVN. Mardia's procedures are considered as a competitor in many related studies. In particular, the omnibus test by ? is widely cited in economics and business journals. Section 3 introduces these procedures, and explain

how they are implemented in the TWVN software package ?. Comprehensive comparisons between these two types of tests were conducted by ?.

## 8.3 文獻引用方式

所謂 bibtex 的文獻資料庫是一個檔案（副檔名為 bib），將所有文獻依固定格式輸入（請參考附檔 WANG\_ref.bib 檔），譬如典型的幾個欄位「author」、「title」、「journal」、「year」、「volume」及「pages」。作者可以將所有曾經引用過文獻都放在這個檔案一起維護。需要引用時，只要 \cite 標號（label）即可。最大的優點是不需要為每篇文章重複的文獻資料再輸入或複製一份。也因為格式固定的關係，維護與管理都很方便，當需要以不同方式呈現時，只要呼叫適當的 bibliographystyle 即可。

上節的陳述方式所配合的 bibliographystyle 是 plainnat。這是個常見的格式，有很多選項（options），預設為「作者 [年份]」，也就是上文看到的樣子。與前兩篇文章不同之處在引用時不需要再輸入作者姓氏，直接用 \cite 引用就會出現在作者資料庫檔的作者名字與年份。讀者可以試著採用不同的 bibliographystyle，<sup>1</sup>看看有什麼不同。

## 8.4 製作方式

由於使用了另一個檔案（bib 檔），編輯的過程要經過幾道程序。編譯前先準備好 bib 檔（格式如附件的 WANG\_ref.bib），及本檔（檔名：template\_ref.tex，引用參考文獻的方式見前段的示範）。對本檔案編譯四次，程序如下：

1. XeLatex
2. Bib Tex
3. XeLatex
4. XeLatex

第一次引用 bib 檔或是更新 bib 檔時才需要四道程序，如果只是修改文章內容或更動引用，只需進行一般的編譯。但如果出現異常狀況，可以試著先清除所有編譯過程的附屬檔，再重新執行上述程序。

---

<sup>1</sup>請查看本文原始檔引用 bibliographystyle 的地方，旁邊有註解好幾個不同的格式。