智慧化企業整合 期末報告

-顧客信用預測

指導教授：邱銘傳 博士

學生：李佩怡 108034543

一、動機與目的

隨著人類文明的高速發展，獲得生活上所需物品的方式由粗略的以物易物轉為可量化的實體貨幣、進一步的發展出塑膠貨幣、甚至是在肉眼看不見的空間中流通的虛擬貨幣。而貨幣形式不斷改變的主要原因可能來自於便利性，不管是行動支付、或者是信用卡支付，都可以為我們省去帶著一大堆硬幣或紙鈔出門的麻煩，也使得購物變得更簡單。

然而信用卡是一個先享受後付款的支付方式，由於購買當下並不需要真的付錢就能得到商品，因此逐漸衍生出「卡奴」的問題，有一部分消費者可能無法衡量自己的經濟能力，而是一股腦的衝動購物，導致負債的累積,信用卡公司也因為無法追討回欠款而產生財務漏洞。為了解決此問題，發卡公司會根據消費者的經濟能力、過往消費以及還款狀況進行評估，制定出合理的消費額度，盡量避免消費者刷卡後無法如期還款的情況。而本次研究是希望藉由消費者過往消費行為的各項資料，對他們的信用做分類。

二、資料集介紹

本次使用的資料來自 UCI machine learning，它是紀錄客戶違約與否的資料集，總共有三萬筆資料，且有二十四個屬性，分別是二十三個預測用的資料以及最後一個紀錄預測結果的反應變數欄位，沒有任何遺失值，因此所有資料皆採用，而這三萬筆資料中，有 23364 筆會如期還款、6636 筆有延遲還款的情況。

用於預測的二十三個屬性包含信用卡公司給予的刷卡額度、性別、教育程度、婚姻狀態、年齡、變數六到十一為客戶在該月的還款情形，其值介於[-1,+9],分別代表準時付款、及時付款、延遲一個月、延遲兩個月、…、到延遲九個月或以上、變數十二到十七表示該月的帳單金額、變數十八到二十三分別表是上一個次還款總額，以及最後的第二十四個變數，為預測結果，0 表示如期還款、而 1 表示有延期情況。詳細介紹如下表 2.1 所示：

表 2.1 資料集屬性介紹

編號	屬性名稱	變數範圍
1	信用卡額度	非負整數
2	性別（1：男 2：女）	類別變數
3	教育程度（1：研究所 2：大學 3：高中 4：其他）	類別變數
4	婚姻狀況（1：已婚 2：未婚 3：其他）	類別變數
5	年齡	非負整數
6	2005 年 9 月付款期況	整數[-1,+9]
7	2005 年 8 月付款期況	整數[-1,+9]
8	2005 年 7 月付款期況	整數[-1,+9]

9	2005 年 6 月付款期況	整數[-1,+9]
10	2005 年 5 月付款期況	整數[-1,+9]
11	2005 年 4 月付款期況	非負整數
12	2005 年 9 月帳單金額	非負整數
13	2005 年 8 月帳單金額	非負整數
14	2005 年 7 月帳單金額	非負整數
15	2005 年 6 月帳單金額	非負整數
16	2005 年 5 月帳單金額	非負整數
17	2005 年 4 月帳單金額	非負整數
18	2005 年 9 月前一期還款金額	非負整數
19	2005 年 8 月前一期還款金額	非負整數
20	2005 年 7 月前一期還款金額	非負整數
21	2005 年 6 月前一期還款金額	非負整數
22	2005 年 5 月前一期還款金額	非負整數
23	2005 年 4 月前一期還款金額	非負整數
24	下一期是否如期償還（0：是 1：否）	二元變數

三、資料前處理

本研究使用的資料集共有三萬筆資料，而每一個屬性的數值皆有不同的分布範圍，因此須要透過正規化進行處理，由於第二十四個變數，即預測結果的資料量有明顯的不平均情況，因此須透過使用 smote 演算法的方式消除；除此之外，由於此資料集中屬性數量較多，因此藉由隨機森林進行特徵值選取，再觀察特徵值數量對於結果有無影響。

(一)正規化與非平衡資料集處理

如第二章所提到的，此次研究所使用的資料集三萬筆數據是由 23 個預測用的屬性以及 1 個預測結果屬性共同組成，由於各屬性的數值範圍有所不同，因此在資料前處理階段先進行正規化處理，讓所有資料的分布範圍接限制在-1 到+1 間，避免某些數值較大的資料變成關鍵變數。程式碼如下圖 3.1 所示：

```
from sklearn.preprocessing import MinMaxScaler
import pandas as pd
data = pd.read_csv('original.csv')  #讀取原始資料檔案
scaler = MinMaxScaler(feature_range=(-1, 1)) #將資料範圍縮到-1到1之間
data_norm= scaler.fit_transform(data)
print(data_norm)

import csv #將正規化後的資料會出成csv檔
data_norm = pd.DataFrame(data_norm)
data_norm.to_csv('data_norm.csv',index = False)
```

圖 3.1 正規化 python 程式碼

而此資料集中，有 23364 筆結果為 1、僅有 6636 筆結果為 0，因此會產生資料不平衡的問題，為了解決此問題，接著使用 smote 演算法，將前二十三個變數與結果變數分開後，決定測試集與訓練及比例，而此次為訓練集 0.7、測試集 0.3，透過此演算法的精神，對少數類別樣本進行分析並根據少數類樣本人工合成新樣本新增到資料集中，部分程式碼如下圖 3.2 所示：

```
X = np.array(data.ix[:, data.columns != 'Y'])
y = np.array(data.ix[:, data.columns == 'Y'])
print('Shape of X: {}'.format(X.shape))
print('Shape of y: {}'.format(y.shape))

from imblearn.over_sampling import SMOTE

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=0)

print("Number transactions X_train dataset: ", X_train.shape)
print("Number transactions y_train dataset: ", y_train.shape)
print("Number transactions X_test dataset: ", X_test.shape)
print("Number transactions y_test dataset: ", y_test.shape)
```

圖 3.2 smote python 程式碼

經過此演算法處理資料集後，結果可由圖 3.3、圖 3.4 看出，訓練集的資料總筆數由 21000 筆擴增至 32608 筆，而其中結果為 0 的資料筆數維持 16304 筆，但結果為 1 的少數資料由 4696 筆增加至 16304 筆，解決資料不平衡問題。

圖 3.3 資料不平衡時資料筆數

圖 3.4 資料平衡後資料筆數

(二)特徵值選擇

特徵值選擇是指選擇獲得相應模型和算法最好能力的特徵集。在建構模型的過程中，我們可能蒐集到許多數據與屬性資料，當這些屬性不夠多時，通常所有屬性都會放到模型當中，然而當特徵屬性到達一定數量時，就可能會因為計算能力有限而很難達到收斂，造成維度災難。

在此次研究中，我使用的特徵值選擇方法為隨機森林，它的核心想法是藉由評量每個特徵值在每棵決策樹中做了多少貢獻，取出平均值，再比較所有特徵值的貢獻度。其主要步驟如下，而程式碼為圖 3.5：

1. 從原始訓練集中使用 Bootstrap 方法取出 m 個樣本，共進行 n tree 次採樣。生成 n tree 個訓練集。
2. N tree 個訓練集，分別訓練 n tree 個決策樹模型。
3. 對於單決策樹模型，假設訓練樣本特徵個數為 n，每次分裂時選擇最好的特徵進行分裂。
4. 每棵樹都繼續分下去，若知道該節點的所有訓練樣本都屬於同一類。在決策樹的分裂過程中不需要剪枝。
5. 將生成的多顆決策樹組成隨機森林。對於分類問題，按照多棵樹分類器投票決定最終分類結果。

```
tr = RandomForestClassifier() #使用隨機森林分類器
fit = tr.fit(X, y)

importances = forest.feature_importances_
std = np.std([tree.feature_importances_ for tree in forest.estimators_],
             axis=0)
indices = np.argsort(importances)[::-1]

# Print the feature ranking
print("Feature ranking:")

for f in range(X.shape[1]):
    print("%d. feature %d (%f)" % (f + 1, indices[f], importances[indices[f]]))

# Plot the feature importances of the forest
plt.figure() #畫出長條圖
plt.title("Feature importances")
plt.bar(range(X.shape[1]), importances[indices],
       color="b", yerr=std[indices], align="center")
plt.xticks(range(X.shape[1]), indices)
plt.xlim([-1, X.shape[1]])
plt.show()
#選擇出特徵值輸出存入data
data=df.iloc[:,indices[0:10]]
print('Selected Features:','\n',data.columns,'\n',data)
```

圖 3.5 特徵值選擇 python 程式碼

特徵值選擇後，可排序出各特徵值的重要程度，如圖 3.6 所示，根據此順序，選擇了前十個特徵值作為較重要的特徵，進行模型訓練。

```
Feature ranking:
1. feature 5 (0.096054)
2. feature 4 (0.066378)
3. feature 0 (0.065619)
4. feature 6 (0.050899)
5. feature 11 (0.050399)
6. feature 12 (0.046844)
7. feature 22 (0.046461)
8. feature 13 (0.045059)
9. feature 14 (0.044771)
10. feature 16 (0.044064)
11. feature 17 (0.043888)
12. feature 15 (0.043550)
13. feature 21 (0.042619)
14. feature 18 (0.042169)
15. feature 19 (0.042101)
16. feature 20 (0.040959)
17. feature 7 (0.033914)
18. feature 2 (0.032599)
19. feature 8 (0.030316)
20. feature 10 (0.029489)
21. feature 9 (0.028565)
22. feature 3 (0.021776)
23. feature 1 (0.011505)
```

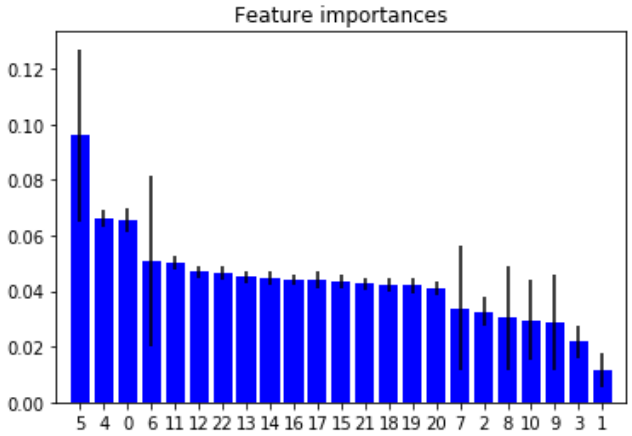

圖 3.6 特徵值重要程度排序

四、研究方法與結果比較

在本章中，會先介紹此次研究中使用到的各種機器學習、或者是深度學習的方法，接著將第三章已經透過資料前處理之數據與原始數據輸入模型，對其結果進行比較，與得到結論。而此次研究中會使用

到的模型包含 SVM、MLP、KNN、BPN，前三者以 python 為主、而最後一個則是使用 Matlab 進行模型建構。

(一)方法介紹與實際應用

1. SVM

SVM 是資料科學中最受歡迎的分類演算法之一。無論是小數據的運用、非線性可分的問題、高維模式識別問題上，SVM 都有不錯的表現。其概念是建構一個超平面，讓資料在空間中能夠被區分成兩類，所以又被稱為二元分類器(binary classifier)。在二維的空間中，超平面指的就是一條線，如圖 4.1、三維空間中，則是一個平面，如圖 4.2。之所以有一個「超」字，是因為資料往往不會只有二維、三維。在更高維度的空間中，我們無法觀察這個平面的形狀為何，於是就用「超平面(hyperplane)」一詞來概括。

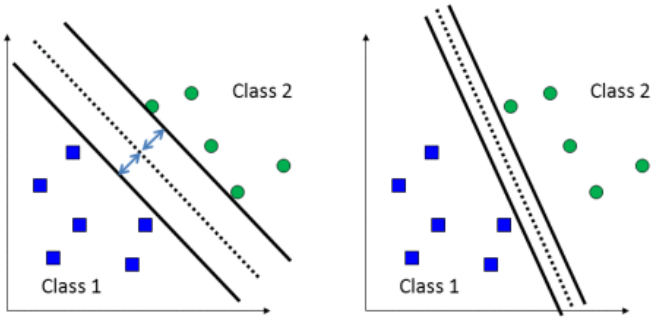

圖 4.1 二維空間中的 SVM 超平面

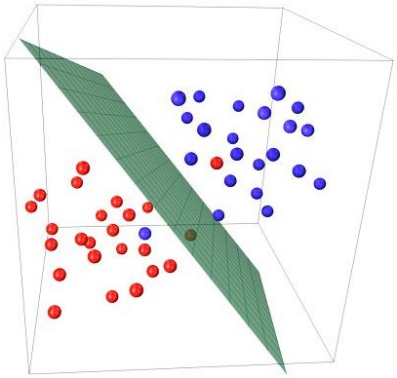

圖 4.2 三維空間中的 SVM 超平面

在本次研究中，將訓練集與測試集按照 7:3 的比例進行劃分，使用的 kernel 函數為 rbf，並將結果由混淆矩陣表示，部分程式碼如圖 4.3 所示。

```
import pandas as pd

df1 = pd.read_csv("data_selected.csv")
df1.head()
df1.shape
X = df1.drop('Y', axis=1)
y = df1['Y']

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.30)

from sklearn.svm import SVC
svclassifier = SVC(kernel='rbf')
svclassifier.fit(X_train, y_train)

y_pred = svclassifier.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix
print(confusion_matrix(y_test,y_pred))
print(classification_report(y_test,y_pred))
```

圖 4.3 SVM 程式碼

將四種資料前處理方式(不做前處理、正規化但不特徵選擇、不正規化但特徵選擇、正規化且特徵選擇)且測試集與訓練集比例分別是 0.3 及 0.7 結合成一張表，結果如下表 4.1 所示：

表 4.1 SVM 結果

SVM	正規化	無正規化
所有特徵	0.81	0.78
特徵選擇	0.81	0.79

2. MLP

多層感知器(Multilayer Perceptron,縮寫 MLP)是一種前向結構的人工神經網路，映射一組輸入向量到一組輸出向量。MLP 可以被看作是一個有向圖，由多個的節點層所組成，每一層都全連接到下一層，如圖 4.4 所示。除了輸入節點，每個節點都是一個帶有非線性激活函數的神經元（或稱處理單元）。

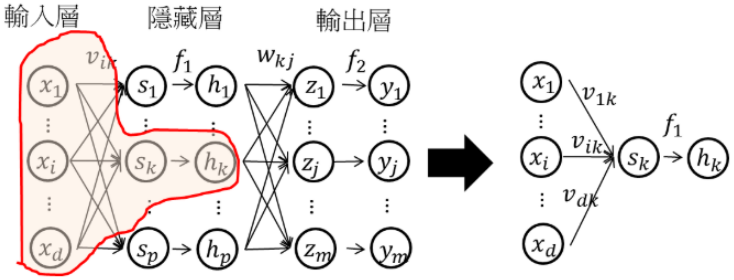

圖 4.4 MLP 結構

在本次研究所使用的 MLP 模型中，首先進行基本參數設定。隱藏層為三層且每一層各有十個神經元，最大迭代次數(max_iter)為三百次，看模型是先到最大次數還是先到達容忍值(tol)，alpha 為

預設值，solver 為優化方式，此選擇的方式為隨機梯度下降 sgd，verbose 為 2 表示輸出所有過程，tol 值為連續兩次的迭代無法降低成本函數，或是得分無法增加，就當成已經收斂完成而結束。部分程式碼如下圖 4.5 所示：

```
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix

y = data['Y']
x = data.drop(['Y'], axis=1)

x_train, x_test, y_train, y_test = train_test_split(x,y, test_size= 0.3, random_state=0)

clf = MLPClassifier(hidden_layer_sizes=(10,10,10), max_iter=300, alpha=0.0001,
                    solver='sgd', verbose=2,  random_state=1,tol=0.000000001)

clf.fit(x_train, y_train)
y_pred = clf.predict(x_test)
```

圖 4.5 MLP 部分程式碼

而研究亦將資料依據有無正規化、有無特徵選擇分為四類分別放入 MLP 模型，將其整合成一張表以後結果如下表 4.2：

表 4.2 MLP 結果

MLP	正規化	無正規化
所有特徵	0.817	0.778
特徵選擇	0.817	0.777

3. KNN

在 K 近鄰分類演算法中，對於預測的新樣本數據（未有分類標籤），將其與訓練樣本一一進行比較，找到最為相似的 K 個訓練樣本，並以這 K 個訓練樣本中出現最多的分類標籤作為最終新樣本數據的預測標籤。

在 KNN 中，先將資料匯入之後，由於 KNN 是藉由確認最近的觀察值來做預測分類，變數比例對結果影響很大，因此須載入標準化比例尺套件，此外亦將訓練集與測試集依照 7:3 分堆，接著進行 KNN 演算法，測試 k 由 1 到 30 的結果，藉由錯誤率找到最佳 k 值，並藉由混淆矩陣得到預測準確度，部分程式碼如下圖 4.6 所示：

```
scaler = StandardScaler() #載入標準化比例尺套件
scaler.fit(df.drop('Y',axis=1))
scaled_features = scaler.transform(df.drop('Y',axis=1))

df_feat = pd.DataFrame(scaled_features,columns=df.columns[:-1])
df_feat.head()

from sklearn.model_selection import train_test_split

X = df_feat
y = df['Y']
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.30,random_state=101) #將訓練資料分為兩類
```

```
from sklearn.neighbors import KNeighborsClassifier #使用KNN演算法

#從k值=1開始測試
knn = KNeighborsClassifier(n_neighbors=1)
knn.fit(X_train,y_train)
pred = knn.predict(X_test)

from sklearn.metrics import classification_report,confusion_matrix #測是演算法好壞
print(confusion_matrix(y_test,pred))
print(classification_report(y_test,pred))

error_rate = []

for i in range(1,30): #原本只建立K=1的模型 現在建立K為1到30的迴圈

    knn = KNeighborsClassifier(n_neighbors=i)
    knn.fit(X_train,y_train)
    pred_i = knn.predict(X_test)
    error_rate.append(np.mean(pred_i != y_test))


#將k=1~30的錯誤率製圖畫出，k=26之後，錯誤率就在10%左右震盪，
plt.figure(figsize=(10,6))
plt.plot(range(1,30),error_rate,color='blue', linestyle='dashed', marker='o',
         markerfacecolor='red', markersize=10)
plt.title('Error Rate vs. K Value')
plt.xlabel('K')
plt.ylabel('Error Rate')
```

```
knn = KNeighborsClassifier(n_neighbors=26) #選擇錯誤率最低的k=26來看confusionmatrix中各個值的表現

knn.fit(X_train,y_train)
pred = knn.predict(X_test)

print('WITH K=26')
print('\n')
print(confusion_matrix(y_test,pred))
print('\n')
print(classification_report(y_test,pred))
```

圖 4.6 KNN 部分程式碼

由於 KNN 有載入標準化比例尺套件，因此不管資料有無正規化皆會在進行資料處理一次，對結果影響並不大，但仍然將之顯示在下表 4.3 中，而從圖 4.7 到 4.10 可以看出錯誤率有逐漸下降的趨勢，在 k 為 26 時降至最低，並在附近震盪。

表 4.3 KNN 結果

KNN	正規化	無正規化
所有特徵	0.813	0.813
特徵選擇	0.823	0.823

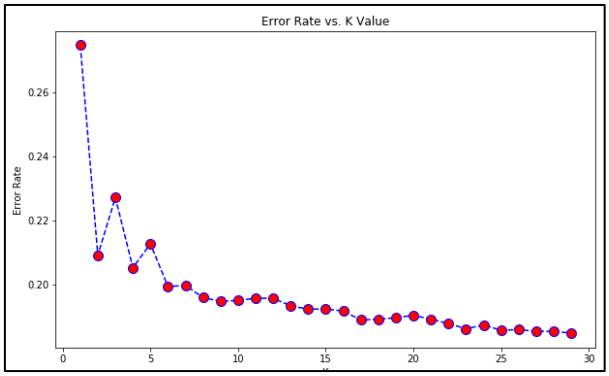

圖 4.7 有正規化無特徵選擇錯誤率

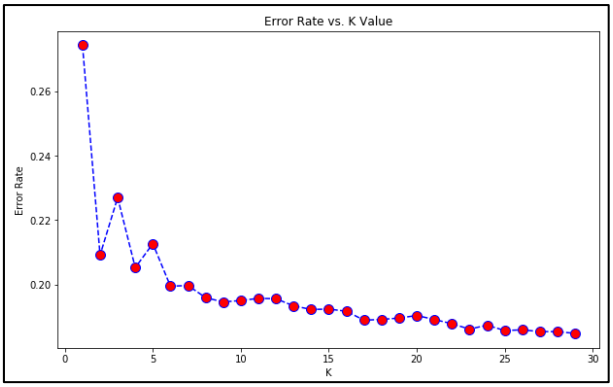

圖 4.8 無正規化無特徵選擇錯誤率

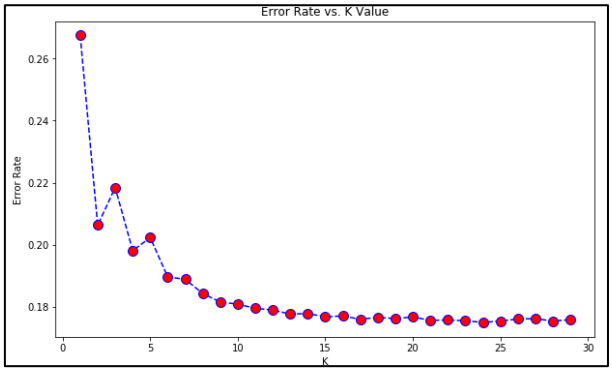

圖 4.9 有正規化且有特徵選擇錯誤率

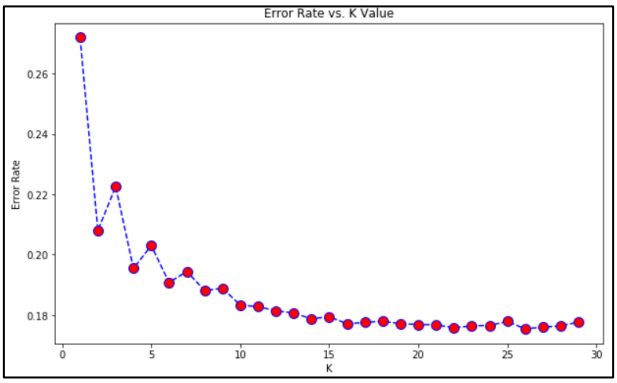

圖 4.10 無正規化但有特徵選擇錯誤率

4. BPN

BPN 的網路架構為 MLP，就是輸入層與輸出層之間多一層隱藏層，使用誤差倒傳遞演算法(Error Back Propagation, EBP)為學習演算法，屬於多層前饋式網路，以監督式學習方式，處理輸入輸出之間非線性映射關係。簡單來說，BPN 是由向前傳遞及向後傳遞兩個部份組成，如圖 4.11 所示，向前傳遞(由左至右的黑線)是先將訓練資料丟進網路去跑，在計算出輸出結果與對應目標之間誤差，而向後傳遞由右至左紅線)是依誤差值調整網路權重，經過這樣多次訓練後，就會將網路修正到誤差極小範圍內的輸出結果。

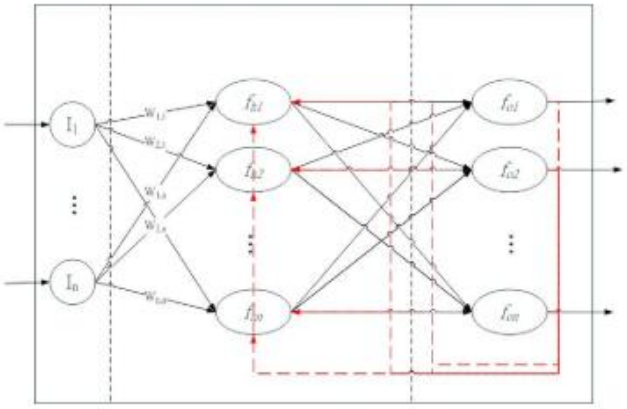

圖 4.11 BPN 結構

在 BPN 的部分，使用的是 Matlab 內建的神經網路工具，透過傳入輸入資料與目標資料(如圖 4.12)以後，可將資料分為 70%訓練、30%測試，再將測試集分出 15%作為驗證集，如圖 4.13 所示，而隱藏層與神經元數都是預設值，不進行更動，就會產生出一個神經網路模型如圖 4.14 所示，接著進行模型訓練。

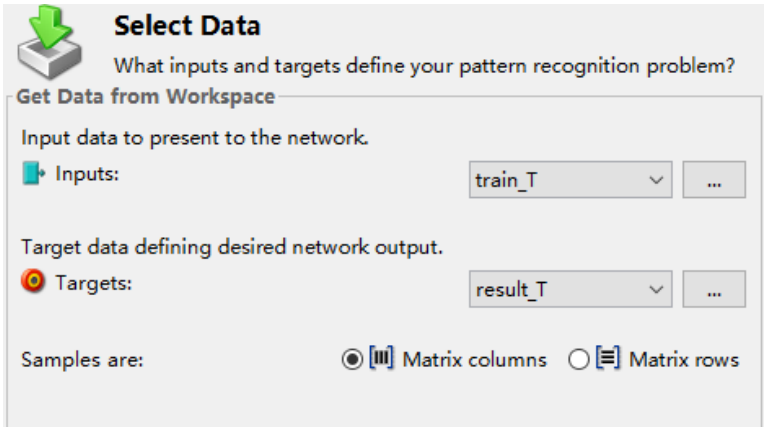

圖 4.12 選擇資料

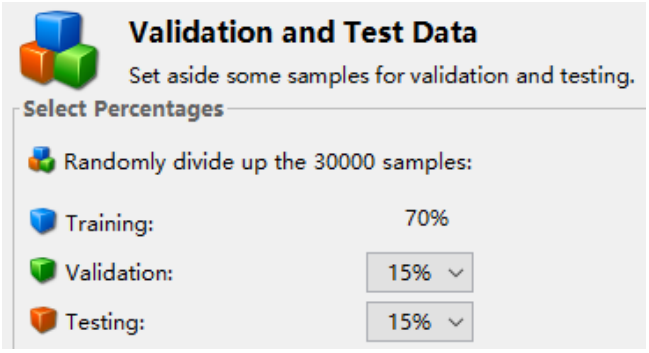

圖 4.13 將資料分為三個子集

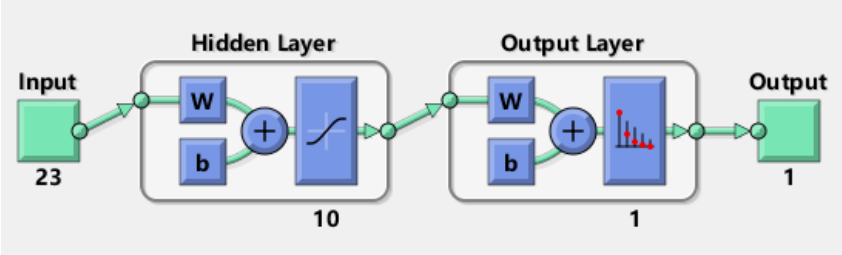

圖 4.14 神經網路模型

Matlab 會自動輸出混淆矩陣以及 ROC 曲線，此文以資料不進行特徵選取、且有正規化的情況下輸出的結果作為範例，如圖 4.15、圖 4.16 所示，此外，將四組資料分別輸入以後可得下表 4.4 結果：

表 4.4 BPN 結果

BPN	正規化	無正規化
所有特徵	82.2	81.3
特徵選擇	81.9	81.5

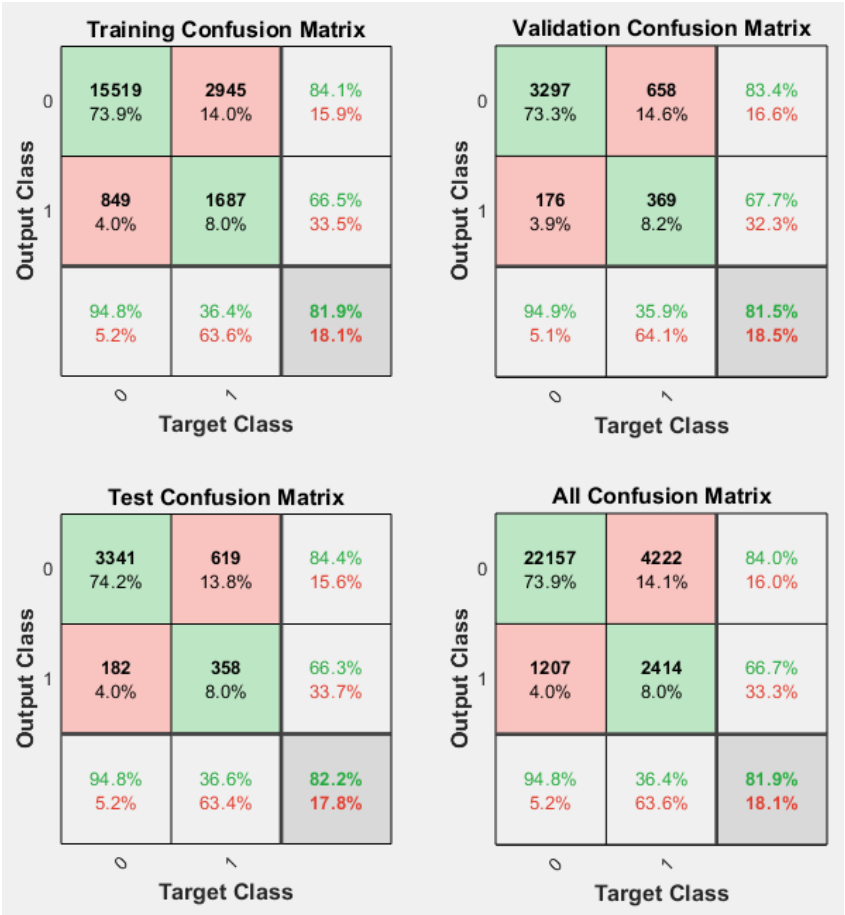

圖 4.15 混淆矩陣

混淆矩陣的基本概念如下，TP 為實際是正例，識別為正例。FN 為實際是正例，卻識別成了負例，可以理解為漏報。FP 為實際是負例，卻識別成了正例，可以理解為誤報。TN 為實際是負例，識別為負例。

有了混淆矩陣，可以構造出很多指標，來從不同角度反映分類器的分類準確程度，主要看以下幾個，準確率為(TP+TN)/(TP+TN+FN+FP)，即不管是哪種類別，預測結果與實際結果一致的比例。召回率為 TP/(TP+FN)，即有多少正樣本被準確識別。錯檢率為 FP/(FP+TN)，即有多少負樣本被錯誤識別。

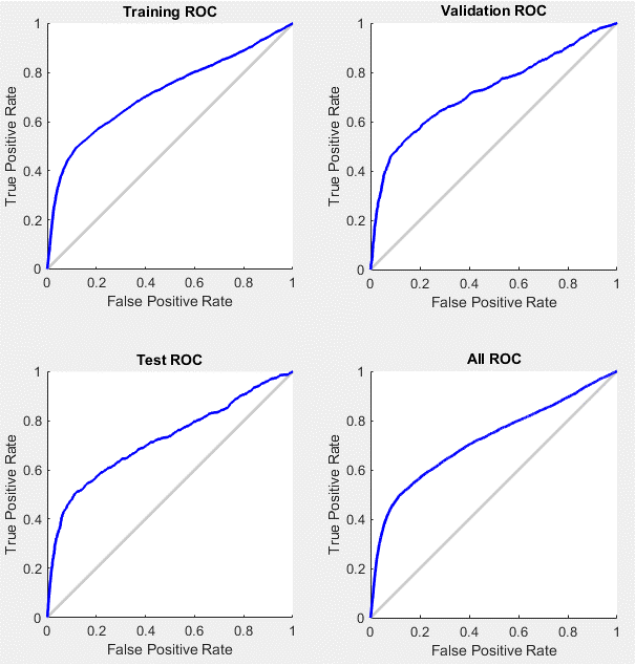

圖 4.16 ROC 曲線

受試者工作特徵曲線（ROC curve）表示一個特定的診斷方法對區別特定的患者組與非患者組樣本的檢測性能，不同診斷水準的真陽性率對假陽性率的函數關係。當診斷試驗完全無價值時，靈敏度等於假陽性率，是一條從原點到右上角的對角線，這條線稱為機會線，完全無價值的診斷試驗其 ROC 曲線與機會線重合；ROC 曲線越偏離機會線，曲線下面積就越大，則診斷試驗的真實性也就越好，說明診斷準確度越高。

(二)結果比較

根據上述四個方法的實驗結果，我們可以將它們整合在一起得到以下的總表，如表 4.5 所示。可以發現在準確度方面，所有特徵值都使用時，BPN 有稍微較佳的結果、而有進行特徵選擇時，則是 KNN 有較佳的結果，整體看來，數據有進行正規化會優於未進行正規化的結果。

表 4.5 各方法結果

	SVM	MLP	KNN	BPN
所有特徵	0.81	0.817	0.813	0.822
特徵選取	0.81	0.817	0.823	0.819

五、結論

在本次研究中,使用的資料前處理有正規化以及消除資料不平衡的smote 演算法，而學習方法有四種，分別是 SVM、MLP、KNN、以及 BPN，根據實驗結果顯示，在此次使用的資料集中，進行正規化是必要的、而是否進行特徵選擇則對結果影響不大，這四種方法並無顯著差別，皆達到 80%以上的準確度，表示它們皆可行。

未來展望與遇到的困難部分,由於對於程式語言並沒有太深入的了解，所以此次研究使用了兩種語言，分別是 python 以及 Matlab，當結果來自兩種不同環境，就無法精確的判斷出方法的優劣，因此若是能更精熟程式語言，將模型建立在同樣的基準點下，可得到可信度更好的結果。除了程式能力不足外,對於深度學習的理解也還有很大的進步空間,透過搜尋網路上的開源資料雖然可以更快速的在短時間內藉由修改參數完成模型訓練，但其中還有許多小細節是值得更進一步鑽研的。

參考資料

http://archive.ics.uci.edu/ml/datasets/default+of+credit+card+clients?fbclid=IwAR1rRoORDwnzIRpFHDIdcJOAAmNhLSvvEw_Gqpn1nHC8x2UqdVDjKCrtmqs

https://medium.com/@chih.sheng.huang821/%E6%A9%9F%E5%99%A8%E5%AD%B8%E7%BF%92-%E7%A5%9E%E7%B6%93%E7%B6%B2%E8%B7%AF-%E5%A4%9A%E5%B1%A4%E6%84%9F%E7%9F%A5%E6%A9%9F-multilayer-perceptron-mlp-%E5%90%AB%E8%A9%B3%E7%B4%B0%E6%8E%A8%E5%B0%8E-ee4f3d5d1b41

https://kknews.cc/zh-tw/tech/5v4943l.html