

國立成功大學

109 學年度第二學期

競賽報告

課程名稱: 資料科學導論

授課老師: 李政德

報告主題: HW6(In-class competition)

組別: 教父

組員: 統計系 112 H24081333 林家同
 資訊系 112 F74086250 莊上緣
 資訊系 112 F74082272 李培綸

Brief Instruction of the problem

這次的作業是以競賽的形式呈現的，主要是要讓我們藉由老師提供的 data 來做 churn_prediction。Data 內有一份 train.csv 與 test.csv 與 sample_upload.csv，我們必須藉由機器學習演算法來將 train.csv 內的資料作為訓練依據 train 出一個預測模型，然後再針對 test.csv 去做預測，得出一個 prediction result，並按照 sample_upload.csv 的格式上傳至競賽平台。平台會提供每次我們上傳的 csv 檔與真實結果之 confusion_matrix 值來讓我們參考，並且平台會以 $\text{accuracy} \times 0.3 + \text{prediction} \times 0.4 + \text{fscore} \times 0.3$ 來得出該筆資料的總評，並會對每組進行排名。

True/False 預測正確？		Positive/Negative 預測方向	
		實際 YES	實際 NO
預測 YES		TP (True Positive)	FP (False Positive) Type I Error
		FN (False Negative) Type II Error	TN (True Negative)

$$\begin{aligned}\text{Accuracy} &= (TP+TN) / \text{Tot. N} \\ \text{Precision} &= TP / (TP+FP) \\ \text{Recall} &= TP / (TP+FN) \\ \text{F1 Score} &= \frac{2}{\frac{1}{\text{Precision}} + \frac{1}{\text{Recall}}}\end{aligned}$$

Data Analysis&preprocess

這次的資料含有 13 個特徵值與 1 個預測目標(Exited)，我們使用了跟 final_project 一樣的相關係數測試方式，發現把以下這三個特徵: RowNumber, CustomerId, Surname 給 drop 掉後，用剩下的 10 個特徵值來預測會較為妥當。但是 Geography 與 Gender 的資料為離散型，因此我們將他們改以數字代替。

```
#資料預處理，刪掉不影響預測結果的特徵
df_train=df_train.drop(columns=['RowNumber','CustomerId','Surname'])
#將類別型資料編碼
df_train['Geography'] = df_train['Geography'].replace(['France', 'Germany', 'Spain'], [0, 1, 2])
df_train['Gender'] = df_train['Gender'].replace(['Male', 'Female'], [0, 1])

#資料預處理，刪掉不影響預測結果的特徵
df_test=df_test.drop(columns=['RowNumber','CustomerId','Surname'])
#將類別型資料編碼
df_test['Geography'] = df_test['Geography'].replace(['France', 'Germany', 'Spain'], [0, 1, 2])
df_test['Gender'] = df_test['Gender'].replace(['Male', 'Female'], [0, 1])
```

由於要得知預測結果的評比就必須上傳，而上傳就會花掉一次次數，因此我們決定先在本地端模擬。原本的作業方式是要以 8000 筆的 train.csv 資料切成 X_train(那 10 個特徵值)與 y_train(train.csv 內的" Exited")，丟入 ML 演算法後得出預測模型，接著再把 X_test(也就是 test.csv)套入模型並得出 y_predict，並且將 y_predict 上傳至平台去獲得我們的模型評比，但若是每次都得上傳才知道分數，勢必會花掉大量上傳次數，因此我們的做法是同樣以 4:1 的比例，只針對 train.csv 去切成 X_train、y_train、X_test、y_test，也就是 train 的資料變成 6400 筆，而 test 則為 1600 筆，這樣我們自己這邊就有 y_test 來

測試我們的 y_predict 是否準確，可以將參數微調至最佳情形，再以這些最佳參數進行正式的 8000 筆 train 資料來預測 2000 筆 test 資料，得出結果再上傳就好了。

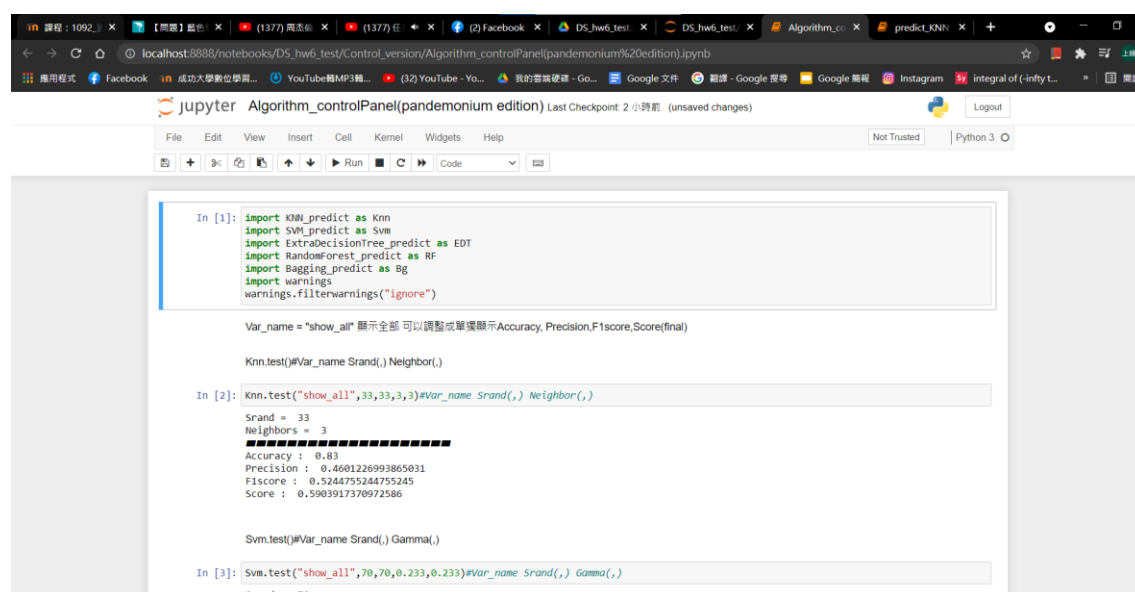
```
#將訓練集切分為4:1
X_train=df_train.iloc[:,df_train.shape[1]-1]
y_train=df_train.loc[:,['Exited']]
ytrain_series=y_train['Exited']
print("number of 1:",ytrain_series.tolist().count(0))
X_train, X_test, y_train, y_test = train_test_split(X_train, y_train, test_size = 0.2, random_state = 33, stratify=y_train)

#train資料算出accu
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score,precision_score,f1_score
classifier = KNeighborsClassifier(n_neighbors=3)
classifier.fit(X_train, y_train)
```

Method

我們的第一個想法是先試試看我們所學過的演算法，看看哪個演算法的表現最好，因此我們測試了 KNN、RandomForest、ExtraDecisionTree、SVM 與 Bagging，我們將每個演算法都寫成一個 ipynb 檔，再使用上方提過的切分 train 來做本地端表現測試的方式(沒有上傳去獲得評分，而是 4:1 切分 train.csv)，但是這五個演算法在參數都使用 default 的狀況下，似乎只有 SVM 的表現比較好一點(acc 大約是 0.86 precision 是 0.82、fscore 為 0.44 左右)，為了知道每個演算法在這次作業的情況下使用哪種參數最適合，我們將每個演算法模組化並寫了一個 argument_test.ipynb，這個方式是將原本個別的五個演算法 ipynb 檔都使用 sublime 這個

文字編輯器寫成.py 檔，並使用我們在資工系待了兩年所學到的分檔傳參數的方式，這樣只需要在 argument_test.ipynb 內 import 我們寫好的五個演算法的 py 檔，並輸入欲測試的參數範圍，他就可以用迴圈的方式個別呼叫那五個演算法 py 檔，並傳入參數，印出該演算法在哪個參數組合下的 acc, precision, fscore 的表現是多少，再搭配這次的 final_score 權重計算，這樣我們就可以知道該參數是否是較佳的參數了。



```
In [1]: import Knn_predict as Knn
import SVM_predict as Svm
import ExtraDecisionTree_predict as EDT
import RandomForest_predict as RF
import Bagging_predict as Bg
import warnings
warnings.filterwarnings("ignore")

Var_name = "show_all" 顯示全部 可以調整或單獨顯示Accuracy, Precision, F1score, Score(final)

Knn.test(Var_name Srand(), Neighbor())

In [2]: Knn.test("show_all", 33, 33, 3, 3) #Var_name Srand(), Neighbor()

Srand = 33
Neighbors = 3
=====
Accuracy : 0.83
Precision : 0.4601226993865031
F1score : 0.5244755244755245
Score : 0.5903917370972586

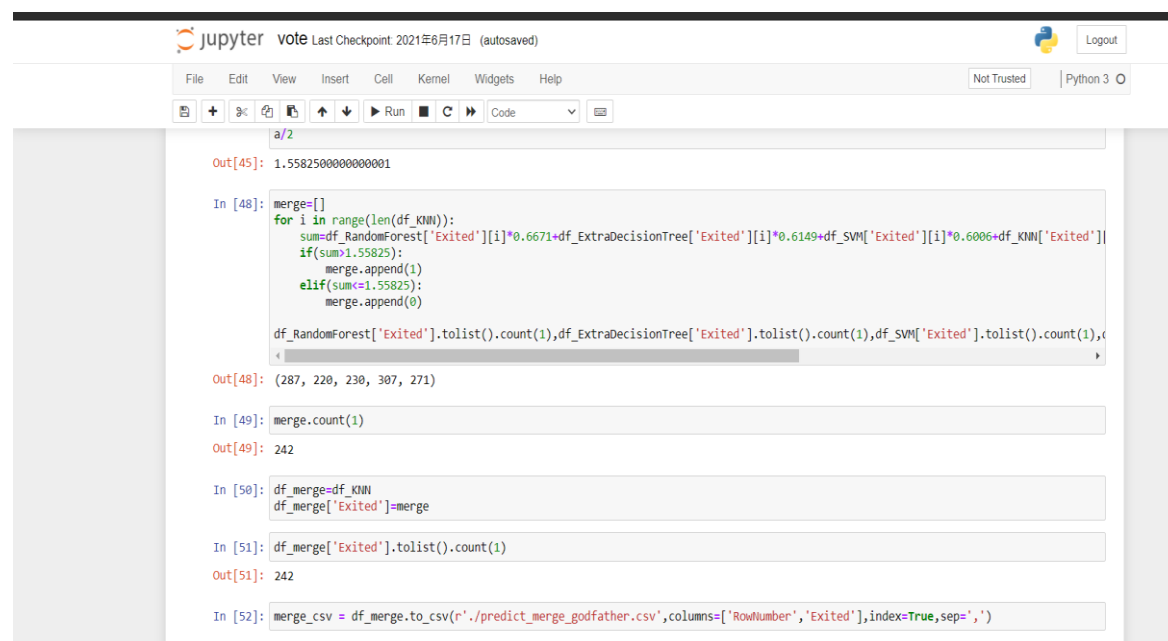
Svm.test(Var_name Srand(), Gamma())

In [3]: Svm.test("show_all", 70, 70, 0.233, 0.233) #Var_name Srand(), Gamma()
```

在經過了參數測試模組化後，我們的排名提升到了 15 名左右，但是似乎是遇到了瓶頸，不論怎麼試，三個分數都分別卡在 0.865, 0.83, 0.47 左右，而且我們發現了一個問題：在本地端以 6400:1600 測試出的表現和實際上用 8000:2000 上傳得到的表現評比會有落差，因此我們就想出了第二種辦法來試圖提升表現。

第二種方式是這樣的，我們寫了一個投票程式(vote.ipynb)，

具體運作流程是這樣的，五個演算法各自得出一個 `y_predict` 的 csv，接著以這個 `vote.ipynb` 去開啟那五個 csv 檔，以迴圈去看那 2000 筆 `Exited` 中，每個演算法的預測個別是 0 還是 1，若是有一半的演算法在該筆資料結果是 0(或 1)，則我們的上傳.csv 中的該處就會 append 0(或 1)，我們為了避免不同演算法的表現參差不齊，還加入了權重考量，以每個演算法提交上去評測系統得到的總評作為權重，每個演算法有不同的表現，就會有不同的權重，他的概念就很像全班有五個學生先個別做同一份考題，然後再一起討論考題的答案，最終交出一份他們五人認為最正確的答案上去，而成績越好的學生在討論答案時，越具有話語權。



```
Out[45]: 1.5582500000000001

In [48]: merge=[]
for i in range(len(df_KNN)):
    sum=df_RandomForest['Exited'][i]*0.6671+df_ExtraDecisionTree['Exited'][i]*0.6149+df_SVM['Exited'][i]*0.6006+df_KNN['Exited'][i]*0.6006
    if(sum>1.55825):
        merge.append(1)
    elif(sum<=1.55825):
        merge.append(0)

df_RandomForest['Exited'].tolist().count(1),df_ExtraDecisionTree['Exited'].tolist().count(1),df_SVM['Exited'].tolist().count(1),df_KNN['Exited'].tolist().count(1)

Out[48]: (287, 220, 230, 307, 271)

In [49]: merge.count(1)

Out[49]: 242

In [50]: df_merge=df_KNN
df_merge['Exited']=merge

In [51]: df_merge['Exited'].tolist().count(1)

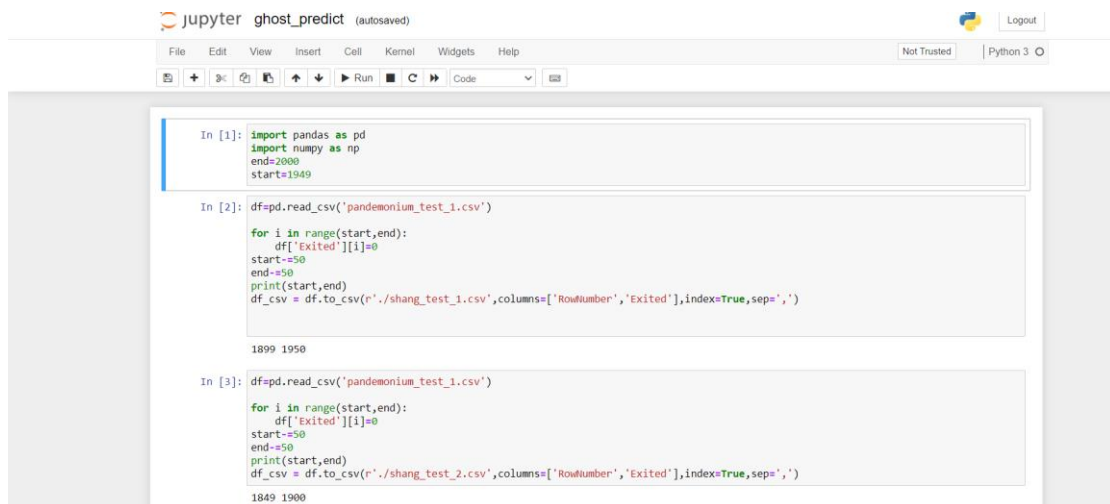
Out[51]: 242

In [52]: merge_csv = df_merge.to_csv(r'./predict_merge_godfather.csv',columns=['RowNumber','Exited'],index=True,sep=',')
```

上面這個投票的方式幫助我們的三個評比的表現上升到了 0.87, 0.84, 0.48，但是又卡在了一個瓶頸，眼看繳交期限就快到了

(期末考完僅剩兩三天)，在逼不得已的狀況下，我們想出了一個我們稱之為”masking_guess_number”的遮罩猜數字大法。

我們以剛剛提到的投票法所得出的 csv 為主，從 1~2000 以每 50 筆資料為一個區段，一次將一個區段改為全部都是 0 並上傳至平台來得出結果(比如說第一次將 1~50 的 Exited 都改成 0，而其餘 1950 筆資料都和原本的投票法 csv 一樣，其餘以此類推)，透過觀察這 40 個評比結果與原本的投票 csv 的評比結果的 precision 差異，將 precision 有變得比原本高的區段記錄下來，最後以原本的投票 csv 為主，將裡面的那些區段的 Exited 都改成 0(比如這 40 個評比結果中有 5 個區段的 precision 是比原本的投票 csv 之 precision 還高，代表將該區段都改成 0 會增加整體 precision)，這個方式被我們稱為 0 遮罩，靈感來源是資工系課程中的 booting mask，我們在 public leaderboard 中，precision 為 1.0000 就是這麼測試出來的。雖然這不太算是 ML method，不過在針對預測結果為二元化時，這的確不失為一種方式，不過這就造成了我們在 public leaderboard 的表現相當不佳。理論上，若有更多的次數，我們可以將區段切分從 50 個往下縮小成 10 個、5 個甚至更小，並以此來得出 2000 筆正確的 Exited，就不會有 public leaderboard 的精確度為 1 但是 private leaderboard 非常慘烈的狀況了。



```
In [1]: import pandas as pd
import numpy as np
end=2000
start=1949

In [2]: df=pd.read_csv('pandemonium_test_1.csv')

for i in range(start,end):
    df['Exited'][i]=0
start+=50
end+=50
print(start,end)
df_csv = df.to_csv(r'./shang_test_1.csv',columns=['RowNumber','Exited'],index=True,sep=',')

1899 1950

In [3]: df=pd.read_csv('pandemonium_test_1.csv')

for i in range(start,end):
    df['Exited'][i]=0
start+=50
end+=50
print(start,end)
df_csv = df.to_csv(r'./shang_test_2.csv',columns=['RowNumber','Exited'],index=True,sep=',')

1849 1900
```

我們靠著這個方式，在僅有的上傳次數中，將總排名提升到了第 6，但是我們希望能再更進一步，因此我們採用了類似 Q_learning 的概念：用上方的遮罩猜數字大法得出的 precision 為 1 的較準確的 csv 先與 test.csv 拼接(因為 test.csv 那兩千筆資料只是缺少了一個 Exited，因此補上後就成為了完整的資料)，再將拼接好的完整的 2000 筆資料直接接在我們的 train.csv 裡面，讓 train.csv 變成 10000 筆資料，再用這 10000 筆資料去跑我們測出的基礎演算法中表現最好的 SVM 來預測 test.csv，但是結果並沒有比我們的遮罩猜數字大法還好，我們猜想應該是原本那 8000 筆資料佔的資料量相對於 2000 比來說還是太多了，才導致結果沒有改善。


```
df_test['Gender'] = df_test['Gender'].replace(['Male', 'Female'], [0, 1])
df_ghost_predict=pd.read_csv('shang_test_8_remix_22.csv')
exit_list=df_ghost_predict['Exited'].tolist()
df_exit=pd.DataFrame(exit_list,columns=['Exited'])
#將訓練集切分為4:1
X_train=df_train.iloc[:,df_train.shape[1]-1]
y_train=df_train.loc[:,['Exited']]
X_test=df_test

#X_train, X_test, y_train, y_test = train_test_split(X_train, y_train, test_size = 0.2, random_state = 10, stratify=y_train)
df_test1=pd.read_csv('test.csv')
df_test1=df_test1.drop(columns=['RowNumber', 'CustomerId', 'Surname'])
#將類別型資料編碼
df_test1['Geography'] = df_test1['Geography'].replace(['France', 'Germany', 'Spain'], [0, 1, 2])
df_test1['Gender'] = df_test1['Gender'].replace(['Male', 'Female'], [0, 1])
X_train=pd.concat([X_train,df_test1])
y_train=pd.concat([y_train,df_exit])
X_train,y_train
```

最終我們的表現如下：

Public:

6	教父	0.8725	1.0000	0.4848	0.7858	2021-06-28 23:59	152
---	----	--------	--------	--------	--------	---------------------	-----

Private:

25	教父	0.8175	0.7021	0.4848	0.6103	2021-06-28 23:59	152
----	----	--------	--------	--------	--------	---------------------	-----

賽後檢討與總評

很多演算法因為是第一次接觸，所以在使用時還不是非常了解背後的數學式，以至於會不太有根據地調整參數。第二個我覺得是除了 row data 外，我們的 domain knowledge 還不夠，如果對題目指定的特定領域有更深入了解，我相信能更容易找出資料裡的特徵。

總結來說，雖然我們這組的競賽結果不是特別突出，但在過程中還是嘗試了不少提升模型準確度的方式，也詢問過很多同學的意見，相信這次經驗可以成為我往後在資料科學精進的養分。

附錄一. 最佳參數集

KNN: Srand=33, Neighbors=3

SVM: Srand=70, Gamma=0.233

ExtraDecisionTree(EDT): Srand=33, Trees=95, Max_depth=None, Min_samples_split=7, Min_leaf=2

RandomForest(RF): Srand=30, Trees=115, Max_features=10

Bagging(BG): Srand=70, Bag_nums=82, Max_samples=0.3, Max_features=1.0

	KNN	SVM	EDT	RF	BG
Accuracy	0.830	0.865	0.872	0.868	0.869
Precision	0.460	0.429	0.484	0.506	0.493
F1score	0.524	0.564	0.607	0.609	0.604
Final_score	0.590	0.601	0.637	0.645	0.640

附錄二. 上傳檔案說明

這次上傳的檔案包含五份基礎演算法 ipynb, 五份為了模組化測試參數而改寫的基礎演算法 py, 一份資料預處理 py, 一份參數測試模組化 ipynb, 一份投票法 ipynb, 一份遮罩猜數字法 ipynb, 一份模仿 Q_learning 強化式學習的 Q_learning.ipynb, 五份基礎演算法得出的 csv 檔(這樣才能跑得動投票 ipynb), 一份我們使用遮罩猜數字法的最終上傳版本(final_version.csv)以及最一開始檔案內就有的 train.csv, test.csv 與 sample_upload.csv。

另外, 心得的部分, 由於字數會超過 10 頁上限, 因此特別獨立成一個檔案。

