Dcard Intern Quiz - 潛力熱門文章預測 作業報告

報告人：莊于霆

一、 實驗目的

本次作業之目的為透過文章發表後 10 小時的屬性資料，來預測該篇文章在 36 小時之後能否達到熱門文章的標準。而熱門文章的標準為愛心數大於等於 1000 個愛心。

二、 實驗觀察

在本次作業中 Dcard 提供總共 10 個有關於文章屬性的資料表，主要分別為文章發文後 36 小時候的總愛心數和文章發文後 10 小時內每小時的愛心數、分享數、收藏數與留言數，而又將其分為訓練與測試的資料表。各個資料表的資料筆數分別為：

1. posts_train：793750 筆
2. posts_test：225985 筆
3. post_shared_train：304259 筆
4. post_shared_test：83375 筆
5. post_comment_created_train：2372228 筆
6. post_comment_created_test：607250 筆
7. post_liked_train：3395903 筆
8. post_liked_test：908909 筆
9. post_collected_train：1235126 筆
10. post_collected_test：275072 筆

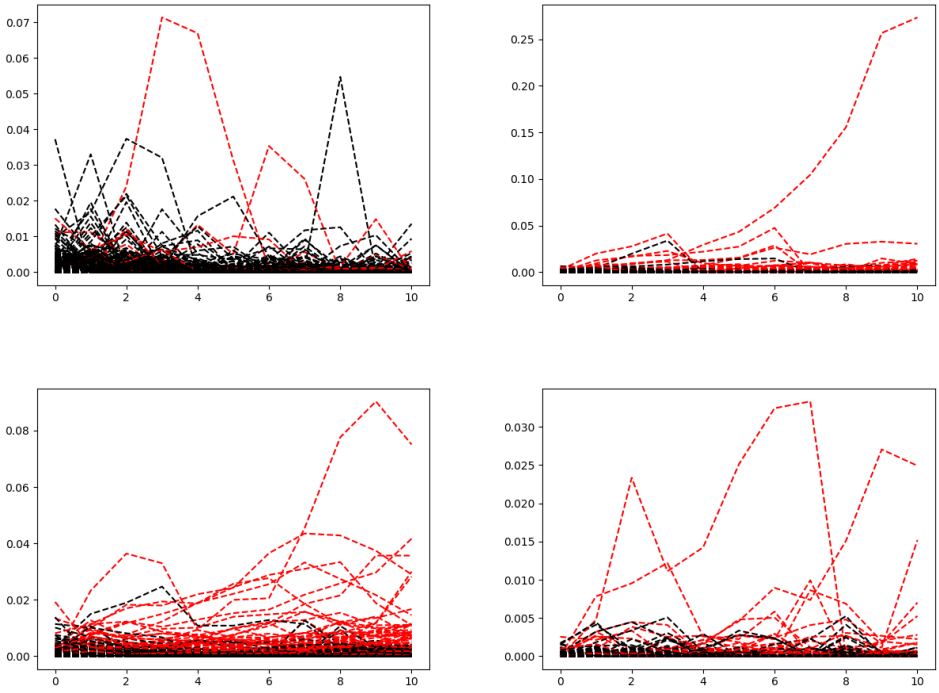
在訓練資料集中，熱門文章的資料筆數遠遠小於非熱門文章筆數，在 793750 篇訓練文章中，僅有 18376 篇為熱門文章，約略 2.3%的比重，可以說兩者資料數量是極度的不平衡，未來在訓練模型時，容易出現全部預測結果是非熱門文章的問題。

(一) 以每小時所獲的分享、愛心、收藏、留言數視覺化資料特徵

為了能更清楚的觀察訓練資料所展現的特性以提升資料前處理的效果，便整理四個訓練的資料表，查看熱門文章與非熱門文章在發文後 10 小時的四個屬性有何差異。為此我依據每篇文章的 post_key 去查找其每小時的屬性並組成屬於該文章的資料，並將每篇文章的資料綜合在一張圖中，以 matplotlib 畫出來。例如某文章於 2019 年 4 月 3 日 13:00 發表，該文章

在每小時的紀錄中，於發文後第一小時、第三小時、第八小時分別獲得10、6、33 個愛心，則我將其表示為[10, 0, 6, 0, 0, 0, 0, 33, 0, 0]，並以該屬性的全部資料做 MinMax 標準化。

而 matplotlib 會因為資料量過大而出現繪圖過久的狀況，所以剛開始我僅取訓練資料的前 5000 筆作為觀察目標。圖中，紅色線條與黑色線條分別為熱門文章與非熱門文章的特性。而觀察結果為以下四圖，左上為每小時留言數、右上為每小時收藏數、左下為每小時愛心數、右下為每小時分享數：



圖一、前 5000 筆訓練資料每小時獲得留言、收藏、愛心、分享數

在上圖中我們可以發現以下幾個特徵：

1. 每小時留言數(左上圖)：

   偶爾幾篇熱門文章的每小時留言數會很突出，但在普遍上非熱門文章的每小時留言數與熱門文章的每小時留言數並無明顯差異，甚至非熱門文章每小時留言數大於熱門文章每小時留言數。

2. 每小時收藏數(右上圖)：

   可以觀察出熱門文章每小時收藏數普遍大於非熱門文章每小時收藏數，但普遍差異並不明顯。

3. 每小時愛心數(左下圖)：

熱門文章的每小時愛心數明顯大於非熱門文章。

4. 每小時分享數(右下圖)：

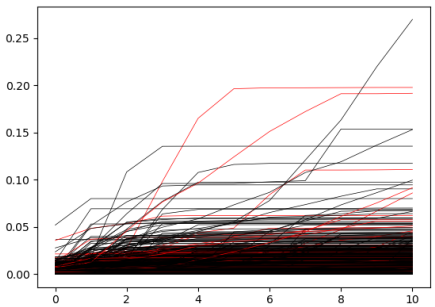
雖有某部分熱門文章的每小時分享數大於非熱門文章的每小時分享數，但大部分與非熱門文章的差異並不明顯。

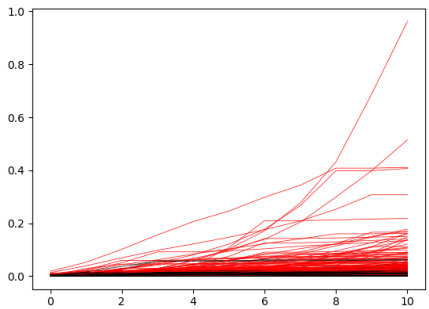
(二) 以分享、愛心、收藏、留言每小時累計數視覺化資料特徵

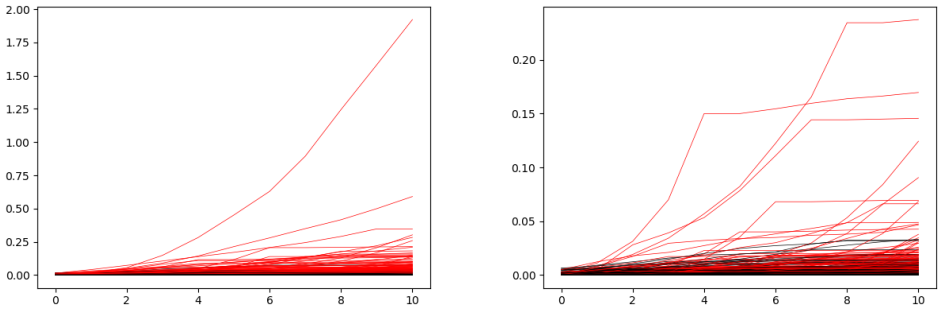
從圖一所觀察出的資料特性可得知，熱門文章與非熱門文章除了每小時獲得愛心數有明顯的差異外，其餘皆無明顯差異。在經過思考後，我認為的確以各篇文章每小時獲得的分享、愛心、收藏、留言數的表示方式不能明確顯示熱門文章與非熱門文章的差異，因為在 Dcard 世界中，使用者能看到的文章愛心留言數並不會因為這小時過後而重新歸零，使用者看到的愛心留言數是累計加總的結果，而重要的是，使用者看到這篇文章的愛心留言數可能會影響其是否要點進去觀看該文章的決策。因此，後續我的資料前處理與視覺化的方式採用每小時累計的分享、愛心、收藏、留言數。

同樣以上一節的舉例為例子，例如某文章於 2019 年 4 月 3 日 13:00 發表，該文章在每小時的紀錄中，於發文後第一小時、第三小時、第八小時分別獲得 10、6、33 個愛心，則我將其表示為[10, 10, 16, 16, 16, 16, 16, 49, 49, 49]，同樣的數量在累計前就以該屬性的全部資料做 MinMax 標準化。

本次觀察我取訓練資料的前 10000 筆作為觀察目標。圖中，紅色線條與黑色線條分別為熱門文章與非熱門文章的特性。而觀察結果為以下四圖，左上為每小時留言數、右上為每小時收藏數、左下為每小時愛心數、右下為每小時分享數：







圖二、前 10000 筆訓練資料每小時累計的留言、收藏、愛心、分享數

在上圖我們可以發現以下特徵：

1. 每小時累計留言數(左上)：

   熱門文章與非熱門文章數的每小時累計留言數仍然無明顯差異，可以說單純看每小時留言數無法預測該文章未來是否會是熱門文章。

2. 每小時累計收藏、愛心數(右上與左下)：

   熱門文章與非熱門文章每小時累計收藏與累計愛心數可觀察出有明顯差異，非熱門文章所代表的黑色線條普遍皆在熱門文章所代表的熱門文章之下。可以說每小時累積的收藏數與愛心數和未來是否是熱門文章有相當高的相關性。

3. 每小時累計分享數(右下)：

   熱門文章與非熱門文章每小時的累計分享數雖然仍然有重疊的部分，但是可發現相較於上一節中以每小時分享數相比，熱門文章與非熱門文章的差異有較明顯的區分。

根據以上兩節的觀察，本人決定在資料前處理上採用每小時累計分享、留言、愛心、收藏數來做為做資料前處理的處理方式，期望能在訓練模型上有較好的結果。

二、 實作程式方法及原因

(一)程式架構與流程

1. 程式架構：

在本次作業中我將整個程式分為四個部分，分別為資料取得、資料前處理、模型建構與訓練、預測，四支程式的檔名分別為 get_data.py、preprocess_data.py、train.py 與 predict.py，而各個程式所擁有的方法與功能以下表展示：

表一、程式的方法詳細資料

程式名稱	方法名稱	輸入/輸出	功能
get_data.py	postgres_connector()	無 / sqlalchemy.create_engine()	建立與 postgres sql 的連線
	createTrainCSV()	host / train, getPST, getPCCT, getPLT, getPCT	從伺服器取得訓練資料 Dataframe，並以 csv 格式將其儲存起來。
	createTestCSV()	host / test, getPSTS, getPCCTS, getPLTS, getPCTS	從伺服器取得包含 like_count_36_hour 的測試資料集，回傳 Dataframe 後，以 csv 儲存起來。
	getTestFromServer()	host / test, getPSTS, getPCCTS, getPLTS, getPCTS	從伺服器取得須預測的資料，並回傳 Dataframe。
preprocess_data.py	getTarget()	df, postKey / time, count	篩選出各個 table 中，屬於 postKey 的資料，回傳時間與記數的 Dataframe。
	preprocessingNPForTest()	host / testData, test['post_key']	將要預測的資料集做前處理後，回傳處理後的資料與其 post_key。
	preprocessingNP()	Host / trainData, trainLabel, testData, testLabel	將要訓練與測試的資料集做前處理後，回傳處理後的資料與其標籤。
train.py	getRecall()	y_true, y_pred / recall	計算 recall 值，並回傳。
	getPrecision()	y_true, y_pred / precision	計算 precision 值，並回傳。
	getF1()	y_true, y_pred / f1	計算 f1 值，並回傳。
	showTrainHistory()	train_history, train,	顯示訓練歷程的歷史走勢

		validation / 圖	圖，包含 f1 值與 loss 值。
	showROC()	y, prob / 圖	計算 ROC 分數與畫 ROC 曲線圖。
predict.py	getRecall()	y_true, y_pred / recall	計算 recall 值，並回傳。
	getPrecision()	y_true, y_pred / precision	計算 precision 值，並回傳。
	getF1()	y_true, y_pred / f1	計算 f1 值，並回傳。

2. 程式執行流程

(1) 訓練流程：

train.py ➔ preprocess_data.py ➔ get_data.py ➔ preprocess_data.py ➔ train.py

(2) 預測流程：

predict.py ➔ preprocess_data.py ➔ get_data.py ➔ preprocess_data.py ➔ predict.py

(二)資料取得

1. 透過 postgres_connector()取得資料後回傳 Pandas Dataframe

在資料處理方面我選擇目前普遍使用的且較熟悉的 pandas 套件，從伺服器資料庫取得資料後即回傳相對應的 dataframe。

2. 將各個資料表儲存成 xxx.csv

為了加速下次取得資料的速度與移除執行本作業需要網路的限制，將資料以 csv 格式儲存在本地端。

```
def createTrainCSV(host):

    engine = postgres_connector(
    host,
    5432,
    "intern_task",
    "candidate",
    "dcard-data-intern-2020"
    )

    train = pd.read_sql(getTrainQuery,engine)
    train.to_csv('train.csv')
    print("已完成train下載")

    os.getcwd()
    getPST = pd.read_sql(getPSTQuery,engine)
    getPST.to_csv('PST.csv')
    print("PST.csv成功")
    getPCCT = pd.read_sql(getPCCTQuery,engine)
    getPCCT.to_csv('PCCT.csv')
    print("PCCT.csv成功")
    getPLT = pd.read_sql(getPLTQuery,engine)
    getPLT.to_csv('PLT.csv')
    print("PLT.csv成功")
    getPCT = pd.read_sql(getPCTQuery,engine)
    getPCT.to_csv('PCT.csv')
    print("PCT.csv成功")
    return train,getPST,getPCCT,getPLT,getPCT
```

```
getTrainQuery = """
SELECT * FROM posts_train
"""

getTestQuery = """
SELECT * FROM posts_test
"""

getPSTQuery = """
SELECT *
FROM post_shared_train AS PST
"""

getPSTSQuery = """
SELECT *
FROM post_shared_test AS PSTS
"""

getPCCTQuery = """
SELECT *
FROM post_comment_created_train AS PCCT
"""

getPCCTSQuery = """
SELECT *
FROM post_comment_created_test AS PCCTS
"""
```

圖三、取得資料後回傳 dataframe 與儲存成 csv 檔

(三)資料前處理

1. 資料標準化與轉換資料格式：

(1) MinMaxScaler() 標準化：

為了優化未來訓練模型時梯度下降的速度與提高模型的精準度，分別將留言、分享、愛心、收藏四個屬性資料進行了 min max 標準化，讓特徵資料按比例縮放到 0 到 1 的區間。

```
def preprocessingNPForTest(host):
    print("進來資料預處理")
    test, getPSTS, getPCCTS, getPLTS, getPCTS = get_data.getTestFromServer(host)
    print(test)
    scaler = MinMaxScaler()
    getPSTS['count'] = scaler.fit_transform(getPSTS[['count']])
    getPCCTS['count'] = scaler.fit_transform(getPCCTS[['count']])
    getPLTS['count'] = scaler.fit_transform(getPLTS[['count']])
    getPCTS['count'] = scaler.fit_transform(getPCTS[['count']])
```

圖四、將取得的資料進行 min max 標準化

(2) 將 created_at_hour 欄位轉成 datetime 格式：

因將資料存成 CSV 時 created_at_hour 時間的欄位會轉為 String Object 格式，後續計算需轉換成 datetime 型態方便計算，也指定轉成的 datetime 格式，以加速其運算時間。

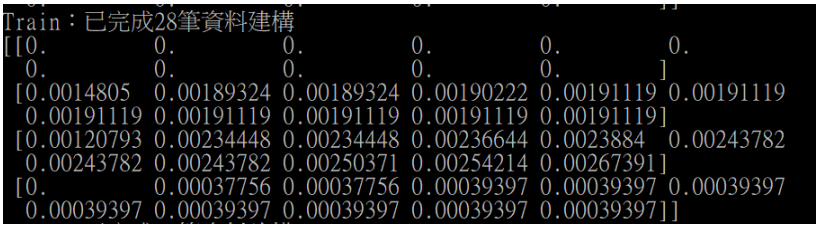
```
train['created_at_hour'] = pd.to_datetime(train['created_at_hour'], format = "%Y-%m-%d %H")
test['created_at_hour'] = pd.to_datetime(test['created_at_hour'], format = "%Y-%m-%d %H")

getPST['created_at_hour'] = pd.to_datetime(getPST['created_at_hour'], format = "%Y-%m-%d %H")
getPCCT['created_at_hour'] = pd.to_datetime(getPCCT['created_at_hour'], format = "%Y-%m-%d %H"
```

圖五、將資料表中的 created_at_hour 欄位

2. 組成所需的資料模式：

(1) 將四個資料表的資料組成 4 * 11 的資料模式：

根據上一章資料特徵的觀察，本人認為將資料每小時的特徵數量轉成每小時特徵數量的累計較能展現熱門文章與非熱門文章的差異，所以將每篇文章 4 個屬性各個小時的紀錄轉成 4 個(愛心、留言、分享、收藏)大小為 11(文章發表後每小時累積的紀錄)的陣列。



圖六、一筆資料的資料格式

而因為一篇文章的特徵屬性資料分別散落在不同的 dataframe 之中，且該文章在 4 個 dataframe 的資料數量不一致，可能在愛心資料表的特徵資料有 10 筆，但在留言資料表數量僅有 5 筆，不容易 Join 起來，所以在取得一個文章的 postKey 後，分別去四個資料表篩選出屬於該文章的資料，再根據資料中的 created_at_hour 計算該資料的紀錄時間與此文章發文時間的差，將資料放到該放的位置上，最後再轉換成累計的資料。

而再資料集中有一些錯誤的資料，例如某文章於下午 1 點刊登，但有早上 10 點的一些收藏留言或分享的資料。本人以 try…except 處理該問題，若遇到此錯誤資料，顯示出來後直接略過。

```
if PSTTime.shape[0] != 0:
    PSTTimestap = ((PSTTime - createTime[0]).astype('timedelta64[h]')).astype('int16')
    for rowNum in range(PSTTimestap.shape[0]):
        index = PSTTimestap[rowNum]
        npToDate = datetime.fromtimestamp((PSTTime[rowNum]-np.datetime64('1970-01-01T00:00:00Z')) / np.tir
        postTime = npToDate.hour
        if postTime == 1:
            postTime ==25
        elif postTime == 2:
            postTime ==26
        postTime = postTime/26
        try:
            testData[testRow][0][index] = PST[rowNum] * postTime
        except IndexError:
            indexErrorCount += 1
            errorKey.append(postKey)
            print(postKey)
            print("錯誤index："+str(index))
            print("出現錯誤次數："+str(indexErrorCount))
            continue
    for plus in range(1,11):
        testData[testRow][0][plus] += testData[testRow][0][plus - 1]
```

圖七、將本文章每小時分享記錄放在資料正確的位置上

但以該方式組成資料會有速度緩慢的問題，因每次要篩選資料時 pandas 需查看整個 dataframe，而每個 dataframe 因資料量龐大而篩選緩慢，處理大量資料時需要大量的時間。對此問題，本人已嘗試過以 pandas 的 apply() 函數與 iterrow() 函數來嘗試解決此問題，皆未果。也有試圖安裝新的 Modin 函式庫試圖加速 pandas 運算速度，但 window 系統不易安裝此函式庫，也未能成

功。最後是以 df.loc[np.in1d(df['post_key'], postKeyList)] 取代 df[df["post_key"] == postKey]，速度也僅提升原本的 2.4 倍(1000 筆訓練資料需處理 2.5 分鐘；1000 筆測試資料需處理 45 秒)，仍然遠遠不及本作業的速度要求。而此問題本人目前覺得能透過 T-SQL 處理，但因時間關係未能測試。

```
def getTarget(df,postKey):
    postKeyList = [postKey]
    getTarget = df.loc[np.in1d(df['post_key'], postKeyList)]
    time = getTarget.loc[:,"created_at_hour"].values
    count = getTarget.loc[:,"count"].values

    return time,count
```

圖八、篩選出屬於此文章的資料

(2) 依據 like_count_36_hour 數量形成對應的標籤值：

將訓練資料與測試資料根據熱門文章的定義轉換成標籤，熱門文章的標籤值為 1，非熱門文章的標籤值為 0。

```
trainLabel = (train['like_count_36_hour']>=1000)
testLabel = (test['like_count_36_hour']>=1000)
```

圖九、形成標籤值

(四)模型選擇、建構與訓練

1. 以 ADASYN() Oversampling 解決資料不平衡的問題

因非熱門文章的數量遠遠大於熱門文章的數量，為了避免未來模型訓練完後會全部預測非熱門文章的問題，我以 Oversampling 對熱門文章重複抽樣，而 ADASYN()會根據熱門文章的資料特性去生成相似的資料，避免單純重複抽樣 Overfitting 的問題，也平衡熱門文章與非熱門文章的數量差異。

```
(90000, 4, 11)
(32000, 4, 11)
(90000,)
(32000,)
Counter({False: 87958, True: 2042})
dict_items([(False, 87958), (True, 87876)])
```

圖十、執行 ADASYN()前後的熱門文章與非熱門文章的數量差異

2. 以一維 CNN 類神經網路組成本作業之模型：

CNN 類神經網路有尋找資料特徵的特性，可以透過 CNN 的幫助，找到除了單純查看愛心、收藏、留言、分享每小時累積數的特徵，例如或許愛心數與收藏數有時間上的一些相關性，透過 CNN 去幫助我們找到我們沒發現的特徵資料。

而一維的 CNN 網路也適合處理時間序列的問題，因此本作業採用一維 CNN 的模型進行訓練。本作業採兩層卷積層，一層池化層與 3 層 NN 作為設計，中間也穿插幾層 Dropout 層避免模型 Overfitting。

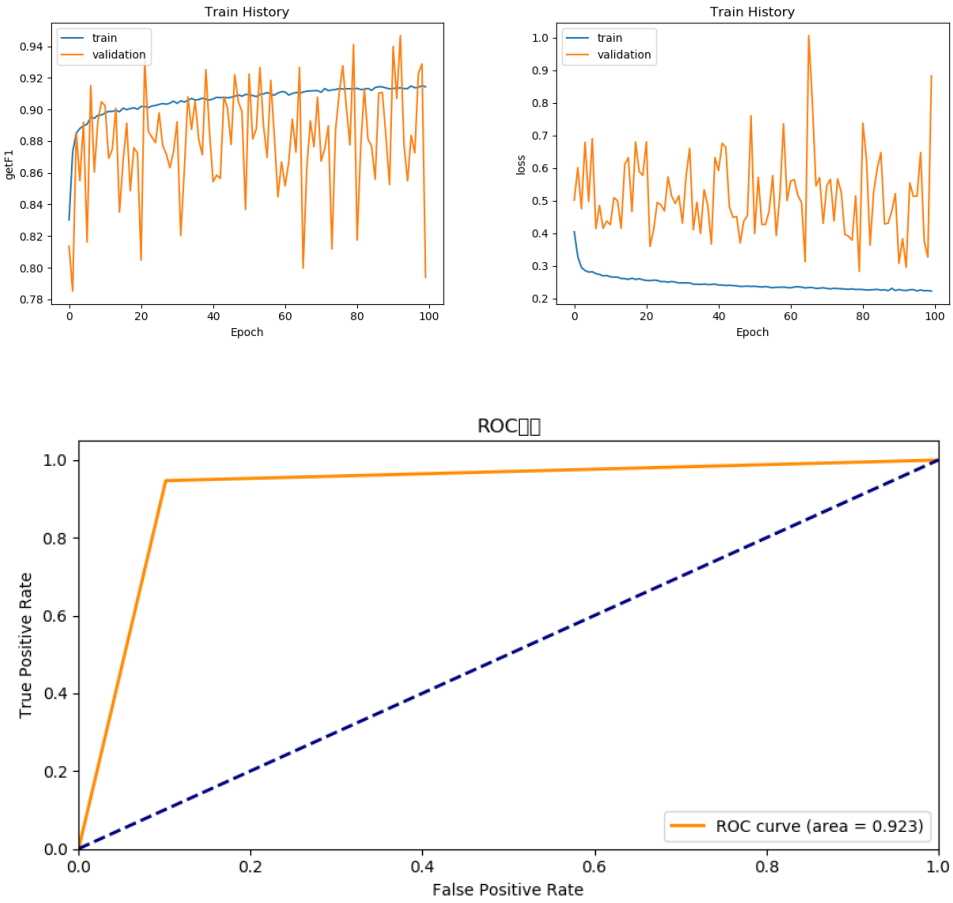
Epochs 與 batch size 我分別設定 100 與 30，評估以 f1 值與 ROC 曲線作為模型衡量優劣標準。

```
model = Sequential()
model.add(Reshape((11, 4), input_shape=(44,1)))
model.add(Conv1D(70, 5, activation='relu', input_shape=(11, 4),padding='same'))
model.add(Conv1D(70, 5, activation='relu',padding='same'))
model.add(MaxPooling1D(7))
model.add(Conv1D(100, 3, activation='relu',padding='same'))
model.add(GlobalAveragePooling1D())
model.add(Dropout(0.5))
model.add(Dense(units=33,kernel_initializer='normal', activation='relu'))
model.add(Dropout(rate=0.5))
model.add(Dense(units=22,kernel_initializer='normal', activation='relu'))
model.add(Dropout(rate=0.5))
model.add(Dense(units=10,kernel_initializer='normal', activation='relu'))
model.add(Dropout(rate=0.5))
model.add(Dense(units=2,kernel_initializer='normal',activation='sigmoid'))
model.compile(loss='binary_crossentropy',
              optimizer='adam',
                metrics=['accuracy',getF1])

# print(model.summary())

trainHistory=model.fit(trainDataArrayReshape,
                       trainLabelArrayOneHot,
                       validation_split=0.3,
                       epochs=100,
                       batch_size=100,
                       verbose=2)
```

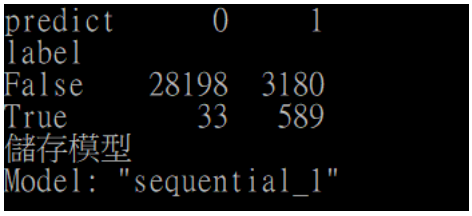
圖十一、模型架構

三、 程式於測試資料的評估結果

因資料處理時間的關係，我僅以 90000 筆訓練資料與 32000 筆測試資料進行評估我的模型。評估的結果 F1 值為 0.89，ROC 分數為 0.92，準確率為 0.89，相關訓練過程與結果以下圖示：



圖十二、模型表現



圖十三、混淆矩陣

四、 程式使用方法

本作業所訓練好的模型放在本資料夾根目錄中，名稱為 DcardQuiz.h5。

(一) 模型訓練：

1. 在 command line 輸入：

   python train.py {database_host} {model_filepath}

(二) 預測資料：

1. 在 command line 輸入 ：

   python predict.py {database_host} {model_filepath} {output_filepath}