

# Intern Report



曾宏鈞

## 怎麼使用

### Requirements

打開命令提示字元並輸入以下指令可以載專案所需的相關套件：

```
pip install -r requirements.txt
```

### 如何執行

- 訓練階段

此階段會在save\_models資料夾裡面新增4個訓練的模型以供未來使用  
並且在output/img底下會產生origin\_data.png的資料視覺化圖形

```
python train.py {model_path}.joblib {host_ip} {port_number}  
# ex.  
python train.py best.joblib
```

- 測試階段(產生答案)

此階段會產生結果在output/answer.csv

```
python test.py {model_path}.joblib {host_ip} {port_number}  
{save__answer_path}.csv  
# ex.  
python test.py best.joblib answer.csv
```

- 評估階段

此階段會評估所選取的模型並儲存結果到output/img、output/record，檢查此兩資料夾可以選擇要在測試階段使用的模型

```
python evaluate.py {model_path}.joblib {host_ip} {port_number}  
{answer_path}.csv  
# ex. python evaluate.py best.joblib answer.csv
```

## 專案架構樹

```
|— evaluate.py
|— file_tree.txt
|— output
|   |— answer.csv
|   |— evaluate_clean_data.csv
|   |— img
|   |   |— after_sampling_data.png
|   |   |— origin_data.png
|   |   |— result_matrix.png
|   |— record
|— requirements.txt
|— save_models
|   |— DecisionTreeClassifier.joblib
|   |— GaussianNB.joblib
|   |— KNeighborsClassifier.joblib
|   |— SVC.joblib
|— test.py
|— train.py
|— utils
|   |— SQL.py
|   |— method.py
|   |— pylab.py
```

## 專案架構說明

### 通用程式碼

- utils/
  - SQL.py > 處理資料庫的鏈結及查詢
  - method.py
    - MLsolution()
      - get\_data()
      - preprocessing\_data() > 處理通用的資料前處理不管在哪個階段
      - run\_predict()
      - run\_train()
      - run\_evaluate()
    - DLsolution() > 將於未來新增
  - pylib.py > 處理剛開始執行的參數設置  
class:
    - Utils()
      - host
      - port
      - model\_path
      - save\_path
      - get\_args() > 處理讀取參數
- save\_models/ > 模型儲存的位置
- output/
  - img/ > 執行時輸出的表格或圖片會輸出成 \*.png在此
  - record/ > 執行evaluate的結果報表、混淆矩陣會儲存在此

說明example，以test.py為例

```
import utils.method as m
def main():
    solution = m.MLsolution() # 新增一個solution物件
    # handle data
    solution.get_data() # 查詢SQL資料
    solution.preprocessing_data() # 資料前處理
    # make a X_test(feature) for model's input
    X_test = solution.features
    solution.run_predict(X_test) # 預測

if __name__ == "__main__":
    main()
```

操作影片

就讓我們以一首歌的時間來看怎麼操做吧👉  
點一下圖片即可

## Data Engineering Intern

<https://youtu.be/CoGo9ygEv8g>

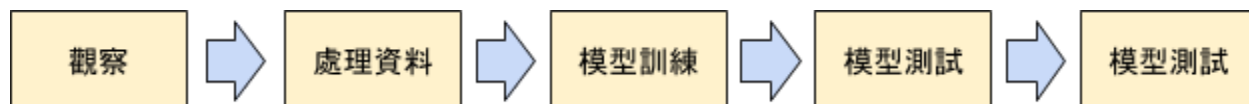
# 方法及為什麼要這麼做

資料整理完後，將特徵(feature)、答案(target)依照兩個不同的階段(train、test)，切分資料為X\_train、y\_train、X\_test、y\_test，經過資料前處理後、使用ExtraTreesClassifier觀察每一個feature的重要性去做特徵選擇。

之後我們測試4種常見分類問題的演算法，觀察f1-score及混淆矩陣來選擇最佳模型演算法，雖然經實驗觀察後尚有需要改進的空間，但在下一段落最後一節也會說明之後可以怎麼改善此分類模型，請繼續參考以下的實驗步驟。

## 實驗步驟

以下5個章節說明實驗步驟，及藉由觀察的結果來調整：



## 1. 實驗觀察

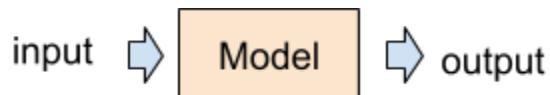
### 觀察原始資料

根據題目敘述，共有一個資料庫及5個類別，分別有10個資料表。

他們詳細的資料形狀如下：

Training	Testing
posts_train : (793751, 3) post_shared_train : (1059305, 3) post_comment_created_train : (2372228, 3) post_liked_train : (3395903, 3) post_collected_train : (3481165, 3)	posts_test : (225986, 3) post_shared_test : (83376, 3) post_comment_created_test : (607251, 3) post_liked_test : (908910, 3) post_collected_test : (803069, 3)

## 1-1 定義問題



Input (feature) :

1. 文章每小時內的分享次數 (post\_shared)
2. 文章每小時內的留言次數 (post\_comment\_created)
3. 文章每小時內的愛心次數 (post\_liked)
4. 文章每小時內被收藏的次數(post\_collected)

Output (Target) :

1. (bool) 每篇文章36小時的愛心數(posts)>=1000

值域

- 每篇文章36小時的愛心數 : (0, +INF)
- 文章每小時內的分享次數 : (0, +INF)
- 文章每小時內的留言次數 : (0, +INF)
- 文章每小時內的愛心次數 : (0, +INF)
- 文章每小時內被收藏的次數 : (0, +INF)

有紀錄的話才會寫進資料庫，因此有可能的範圍為0到正無限(理論上)。

## 2. 資料處理

### 2-1 特徵工程 Feature Engineering

	name	coef
0	count_post_shared_train	0.30024
1	count_post_comment_created_train	0.0471394
2	count_post_liked_train	0.231252
3	count_post_collected_train	0.421369

使用sklearn的 ExtraTreesClassifier，我們可以知道每一個feature的重要性。如上表可知，**文章每小時被收藏的次數的最高** (count\_post\_collected\_train)、**文章每小時被留言的次數最低** (count\_post\_comment\_created)。回想一下平常的使用情形，在具有爭議性話題的文章時，通常討論度很高(留言很多)，但愛心很少。因此我們可以在特徵選擇(Feature Selection)時，試著把count\_post\_comment\_created的權重調低或捨去，因其與最好的特徵重要性差了10倍。

## 2-2 資料整合及清理

### 2-2-1 資料整合

首先我們將資料從資料庫下載下來，並整理成以下的資料總表：

Training				Shape: (11102352, 7)			
	post_key	created_at_hour	count_post_shared_train	count_post_comment_created_train	count_post_liked_train	count_post_collected_train	answer
0	0002f1f8-c96b-4332-8d19-9cdfa9900f75	2019-06-01 05:00:00	0.0	0.0	0.0	0.0	False
1	000c74b1-533d-4445-94ab-038ed4b9a28d	2019-09-13 15:00:00	0.0	0.0	0.0	0.0	False
2	000d9763-e88c-408e-907c-02db7656bb1f	2019-08-26 19:00:00	0.0	0.0	0.0	0.0	False

Testing				Shape:(2628592, 7)			
	post_key	created_at_hour	count_post_shared_test	count_post_comment_created_test	count_post_liked_test	count_post_collected_test	answer
0	00017cc1-df93-4ce1-be7b-0b3c76cb3dc6	2019-11-19 09:00:00	0.0	0.0	0.0	0.0	False
1	00021bc3-7699-4c97-9ec5-20edaac60cc1	2019-11-15 14:00:00	0.0	0.0	0.0	0.0	False
2	000295a1-63ed-4081-b55a-a9b7648eaa7c	2019-12-16 06:00:00	0.0	0.0	0.0	0.0	False

由於有些文章可能在整合時有可能會有缺項的情形，例如：有一篇文章剛發表，有人按愛心但沒有人留言，整合不同資料表時就有可能會有缺值的情形，因此我們採用以下做法：

缺值處理：直接補0，代表資料庫還沒有這筆資料 (EX. 尚未有人留言)。

再來就是觀察訓練集(Training)及測試集(Testing)每一類別答案(Target)的數量，避免其中一個類別過度訓練的問題，然而由下表可知，在訓練集的資料中，False的類別佔了最大宗，比True多了將近42倍之多，因此在資料預處理時，必須先將訓練集的資料抽樣到兩者相同效果會較佳。



觀察完每一個類別的分佈情形時，就要開始去切分資料了，由於我們是要用10小時內的資料去做預測，因此我們需要先將資料做分組，在下一章資料預處理會有更詳細的處理過程。

## 2-3 資料預處理

### 2-3-1 正規化

每一筆資料的單位且值域皆相同(每小時內的資料筆數)，因此不需要正規化或標準化。

### 2-3-2 特徵選擇及切分資料

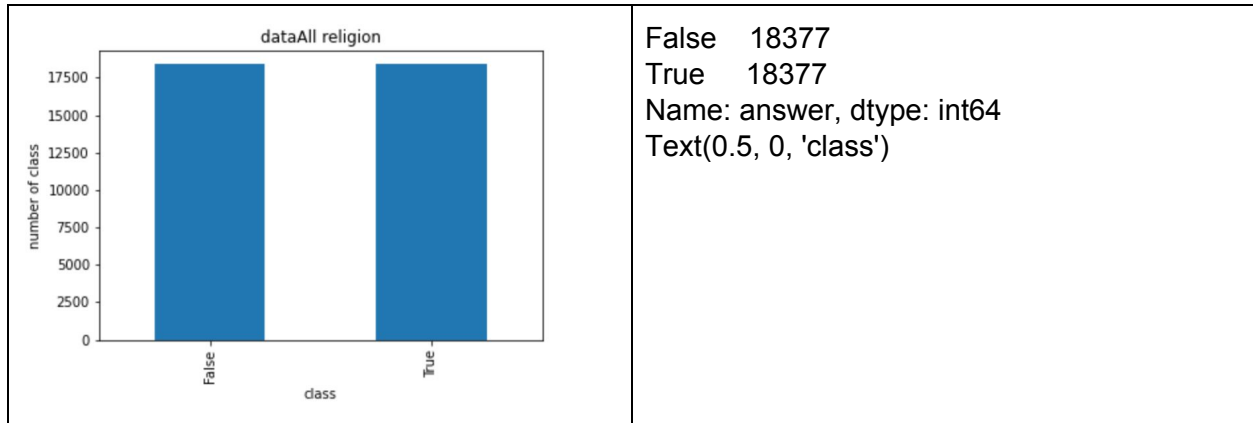
在2-1節有提到，因文章每小時被留言的次數與結果關聯最低 (count\_post\_comment\_created)，再加上以往使用Dcard的經驗，常常有一些時事版的文章在**討論爭議話題時通常討論度很高**(很多留言)，但**不見得會被按愛心**，因此我們在此選擇將對應的欄位捨棄，來避免影響到模型的預測準確度。

然而，因為題目要求是需要用10小時內的資料去做預測，這裡用的是簡單的加總，將10小時內的資料先依照文章id群組，再將每一欄位加總來當作為每一筆資料，完成後就可以將資料拆成特徵(feature)與答案(target)。



### 2-3-3 訓練集資料處理

資料集中的兩類別存在極大的差異，因此我們需要先將訓練集資料做隨機抽樣，讓兩者的訓練資料不會差太多而導致分類不佳的問題，但這樣會少了97%的False資料。



## 3. 模型訓練

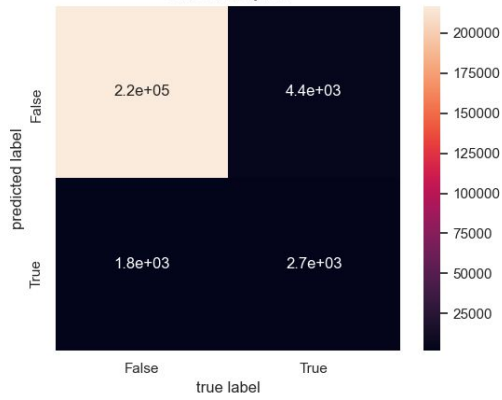

我們使用了四種常見的分類器，分別是GaussianNB、DecisionTreeClassifier、SVM及KNeighborsClassifier，以下是詳細的訓練資訊。

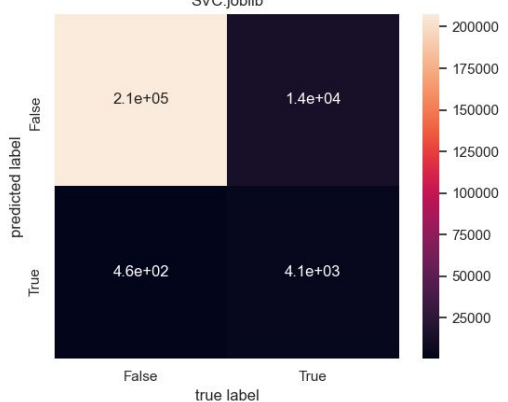
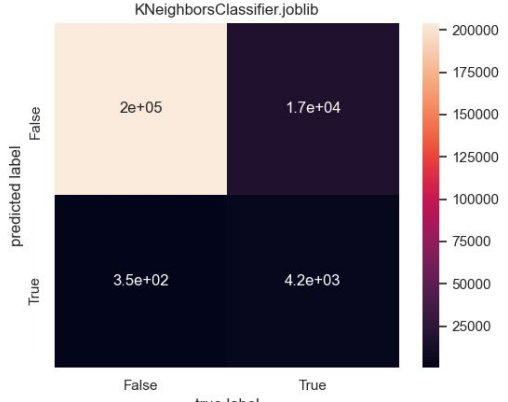
GaussianNB(貝式分類)	DecisionTreeClassifier(決策樹)
<b>優點</b> <ul style="list-style-type: none"><li>● 訓練快速</li></ul> <b>缺點</b> <ul style="list-style-type: none"><li>● 資料需要接近常態分佈(很多資料)</li></ul>	<b>優點</b> <ul style="list-style-type: none"><li>● 簡單好理解(直觀)</li></ul> <b>缺點</b> <ul style="list-style-type: none"><li>● 容易過度擬合=&gt;需要剪枝</li></ul>
SVM(支援向量機)	KNeighborsClassifier(K鄰居法)
<b>優點</b> <ul style="list-style-type: none"><li>● 適用小樣本學習</li><li>● 速度快</li><li>● 可適用分類及回歸問題</li></ul> <b>缺點</b> <ul style="list-style-type: none"><li>● 樣本數很大會很消耗記憶體</li></ul>	<b>優點</b> <ul style="list-style-type: none"><li>● 簡單好理解</li><li>● 準確率高</li></ul> <b>缺點</b> <ul style="list-style-type: none"><li>● 訓練速度極慢</li></ul>

## 4. 測試 & 評估結果

### 3.1 實驗結果

觀察以下圖左側的報告可知，以f1-score上來說，四者模型在False類別的最高表現皆為0.99。但在True類別的分類上GaussianNB比其他模型多3%，為0.47。因此在此分類問題中，選擇上GaussianNB為最佳演算法模型，且不使用DecisionTreeClassifier的原因他的f1-score最低，視覺化此模型後發現樹的寬度太寬，可能需要剪枝來優化演算法，但仍可藉此模型得知背後的意義(ex. 10小時內被收藏的資料>100時，該文章可能會成為熱門文章)。

GaussianNB	acc: 0.9727																																							
<table><tr><td></td><td>precision</td><td>recall</td><td>f1-score</td><td>support</td></tr><tr><td>False</td><td>0.98</td><td>0.99</td><td>0.99</td><td>218834</td></tr><tr><td>True</td><td>0.61</td><td>0.38</td><td>0.47</td><td>7152</td></tr><tr><td>accuracy</td><td></td><td></td><td>0.97</td><td>225986</td></tr><tr><td>macro avg</td><td>0.79</td><td>0.69</td><td>0.73</td><td>225986</td></tr><tr><td>weighted avg</td><td>0.97</td><td>0.97</td><td>0.97</td><td>225986</td></tr></table>		precision	recall	f1-score	support	False	0.98	0.99	0.99	218834	True	0.61	0.38	0.47	7152	accuracy			0.97	225986	macro avg	0.79	0.69	0.73	225986	weighted avg	0.97	0.97	0.97	225986	<div><p>GaussianNB.joblib</p><table><tr><td></td><td>False</td><td>True</td></tr><tr><td>False</td><td>2.2e+05</td><td>4.4e+03</td></tr><tr><td>True</td><td>1.8e+03</td><td>2.7e+03</td></tr></table></div>		False	True	False	2.2e+05	4.4e+03	True	1.8e+03	2.7e+03
	precision	recall	f1-score	support																																				
False	0.98	0.99	0.99	218834																																				
True	0.61	0.38	0.47	7152																																				
accuracy			0.97	225986																																				
macro avg	0.79	0.69	0.73	225986																																				
weighted avg	0.97	0.97	0.97	225986																																				
	False	True																																						
False	2.2e+05	4.4e+03																																						
True	1.8e+03	2.7e+03																																						
DecisionTreeClassifier	acc: 0.92677																																							
<table><tr><td></td><td>precision</td><td>recall</td><td>f1-score</td><td>support</td></tr><tr><td>False</td><td>0.93</td><td>1.00</td><td>0.96</td><td>206147</td></tr><tr><td>True</td><td>0.87</td><td>0.20</td><td>0.32</td><td>19839</td></tr><tr><td>accuracy</td><td></td><td></td><td>0.93</td><td>225986</td></tr><tr><td>macro avg</td><td>0.90</td><td>0.60</td><td>0.64</td><td>225986</td></tr><tr><td>weighted avg</td><td>0.92</td><td>0.93</td><td>0.91</td><td>225986</td></tr></table>		precision	recall	f1-score	support	False	0.93	1.00	0.96	206147	True	0.87	0.20	0.32	19839	accuracy			0.93	225986	macro avg	0.90	0.60	0.64	225986	weighted avg	0.92	0.93	0.91	225986	<div><p>DecisionTreeClassifier.joblib</p><table><tr><td></td><td>False</td><td>True</td></tr><tr><td>False</td><td>2.1e+05</td><td>1.6e+04</td></tr><tr><td>True</td><td>6.1e+02</td><td>3.9e+03</td></tr></table></div>		False	True	False	2.1e+05	1.6e+04	True	6.1e+02	3.9e+03
	precision	recall	f1-score	support																																				
False	0.93	1.00	0.96	206147																																				
True	0.87	0.20	0.32	19839																																				
accuracy			0.93	225986																																				
macro avg	0.90	0.60	0.64	225986																																				
weighted avg	0.92	0.93	0.91	225986																																				
	False	True																																						
False	2.1e+05	1.6e+04																																						
True	6.1e+02	3.9e+03																																						
SVM																																								

					acc: 0.937363
	precision	recall	f1-score	support	<div><div>SVC.joblib</div></div>
False	0.94	1.00	0.97	208234	
True	0.90	0.23	0.36	17752	
accuracy			0.94	225986	
macro avg	0.92	0.61	0.67	225986	
weighted avg	0.94	0.94	0.92	225986	
KNeighborsClassifier					acc: 0.92238
	precision	recall	f1-score	support	<div><div>KNeighborsClassifier.joblib</div></div>
False	0.92	1.00	0.96	204639	
True	0.92	0.19	0.32	21347	
accuracy			0.92	225986	
macro avg	0.92	0.60	0.64	225986	
weighted avg	0.92	0.92	0.90	225986	

以上的ACC皆為 混淆矩陣(TF、FT、FF、TT) / (TF、FT、FF、TT)