Hate Speech Detection on Tweets

蕭云雅口無言不由衷於寫出來了

- 陳冠穎 / 統計109 / H24051312
- 李艾霓 / 統計111 / H24076095

2019/5/17

1. 競賽敘述與目標

此競賽為2019年國立成功大學統計學系李政德老師所開設的選修課程「資料科學導論」中之小組競賽(第二次)，在此競賽中我們會獲得兩個資料集，分別為訓練資料(14869筆)及測試資料(9914筆)，而我們的任務則是透過tweet的內文將其正確分成三類：「0=hateful, 1=offensive, 2=clean」(即訓練資料中的"class"欄位)，最終將我們預測測試資料的結果上傳到課程競賽網站，即獲得預測分數。

- 分數的計算方式為: 0.6*HateF + 0.4*AllF

1.1 問題分析

在這個競賽中要做的是一個多類別的文字分類問題，而分析三個類別：Hateful, Offensive, Clean，不難發現三者之間其實有著程度上的差異，一個有趣的問題是：在這三者之間的界線我們該如何界定？

對於同樣的一個句子，每個人的感受必定會有所不同，舉例來說："yall niggas b cuffing hoes cause yall aint never have bitches" 和 "@nosfiend215 @faggot696969 niggers"這兩則tweet，你會分別將他們歸為哪一類？這大概不是那麼容易可以分得出來的。

對於這樣的問題，真的有標準答案嗎？

在這個資料集中，我們並不知道這些答案是如何被決定出來的，更不能夠確定這些文字在被人工加上標籤時，他們的標準是否維持一致(尤其我想對於這種帶有主觀意識的問題，這想必是很難做到的)，若這之間並不存在一個標準，那機器當然也會很難學到一個規則來做出正確的預測，使結果難以達到預期。(對了，那兩個例子的標籤分別是Offensive和Hateful)

不過撇除上面的議題，我們今天在競賽中所關注的部分僅止於我們的模型是否能夠對於一個句子做出和資料集相同的預測，所以與其說我們要讓機器學習如何分類出文字的語意及情緒，或許說我們要讓機器學習到這個資料集中分類的一些規則會來得更加精確。

1.2 作法簡述

這份Note book只包含了我們在競賽中最後上傳結果所出自的那一個模型，各個分類器中的參數都還欠缺調整，必須說這並不是最好的一個模型。另外，我們在競賽結束之後還有嘗試使用了bert，不過這個部分就放到最後再做補充。

首先我們會先簡單處理資料，將**@tag**以及**http://網址**去除，亦會將多個連續的空白符縮減為一個。接著將文字做詞幹提取以及詞形還原，同時將其轉為詞頻矩陣，轉換完成的資料即準備好可以餵給模型了。這裡總共使用了三個不同的模型：SVC、XGBC以及LR。我們將三個分類器預測的結果以較特殊的規則合併，產生最終預測。

礙於時間不足，這三個模型並沒有被最佳化，其中XGBC的參數是隨意設置的，而LR則是完全使用預設參數，若想提升預測結果，可以考慮使用GridSearchCV重新調整參數，亦可以令各分類器預測出機率，再人工調整分類的界線，使分類結果更優(例如在我們的競賽中，HateF佔了較高的分數，我們可以降低分類為0的標準來使recall提高，同時會提高F1，就能得到比較好的分數)。

```
# Load in our libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import nltk
from wordcloud import WordCloud,STOPWORDS
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer,
ENGLISH_STOP_WORDS
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import MultinomialNB
from sklearn import metrics
```

```
from sklearn.metrics import f1_score, roc_auc_score, classification_report
from sklearn.pipeline import make_pipeline

import re
#nltk.download('stopwords')
from nltk.corpus import stopwords
import pickle
from nltk.stem import WordNetLemmatizer
from nltk.stem.snowball import SnowballStemmer
from nltk import word_tokenize

from sklearn.svm import SVC
from xgboost import XGBClassifier

import warnings
warnings.filterwarnings("ignore")
```

2. 資料前處理

2.1 載入資料

```
train = pd.read_csv("train.csv")
test = pd.read_csv("test.csv")

train.sample(2)
```

	class	tweet
4394	1	RT @HelloCupkake: Too many good single girls, ...
799	1	@New_Dinero Alright pussy.

```
train.shape, test.shape
```

```
((14869, 2), (9914, 2))
```

2.2 清理tweets

```
stemmer = WordNetLemmatizer()
for X in [train,test]:
```

```
    documents = []
    for i in range(0, len(X)):

        # Remove @user & http://...
        document = re.sub(r"(@[A-Za-z0-9]+)|(https?://[A-Za-z0-9./]+)", " ",
str(X.at[i,'tweet']))

        # Substituting multiple spaces with single space
        document = re.sub(r'\s+', ' ', document, flags=re.I)

        documents.append(document)
    X['cleaned_tweet'] = documents
train.head()
```

	class	tweet	cleaned_tweet
0	1	[9-1-13] 2:50 pm "son of a bitch ate my mac n ...	[9-1-13] 2:50 pm "son of a bitch ate my mac n ...
1	1	RT @BryceSerna: Don't be a pussy grab the boot...	RT : Don't be a pussy grab the booty. Love the...
2	2	RT @ClicquotSuave: bunch of rappers boutta flo...	RT : bunch of rappers boutta flood the interne...
3	2	@michigannews13 wow. Thats great language comi...	wow. Thats great language coming from a HS co...
4	1	and this is why I'm single, I don't fuck with ...	and this is why I'm single, I don't fuck with ...

3. 文字轉換為矩陣

3.1 CountVectorizer

```
class LemmaTokenizer(object):
    def __init__(self):
        self.wnl = WordNetLemmatizer()
    def __call__(self, doc):
        return [self.wnl.lemmatize(t) for t in word_tokenize(doc)]

stemmer = SnowballStemmer("english", ignore_stopwords=True)

#In order to trasform text data into vectors
#We combine the Stemmer with sklearn-Countvectorizer
class StemmedCountVectorizer(CountVectorizer):
```

```
    def build_analyzer(self):
        analyzer = super(StemmedCountVectorizer, self).build_analyzer()
        return lambda doc: ([stemmer.stem(w) for w in analyzer(doc)])

vect = StemmedCountVectorizer(min_df= 1,ngram_range=(1,3),tokenizer = LemmaTokenizer())
```

4. 建構模型

4.1 Train Test Split

在建立**Model**的步驟使用第一個cell做train_test_split; 要做**test.csv**的預測用第二個cell來取得測試集。

```
# train_test_split

X_train, X_test, y_train, y_test =
train_test_split(train['cleaned_tweet'],train['class'],test_size=0.3,random_state=333)
```

```
# test.csv Predict

X_train = train['cleaned_tweet']
y_train = train['class']
X_test = test['cleaned_tweet']
```

```
vect.fit(X_train)
X_train_vectorized = vect.transform(X_train)
X_test_vectorized = vect.transform(X_test)
```

4.2 訓練模型

若是建構模型的階段，將最後一行取消註解可以看到預測結果的F1_score。

1. SVC

```
model1 = SVC(kernel='linear',class_weight=
{0:7,1:1.5,2:1.5},probability=True).fit(X_train_vectorized, y_train)
pred1 = model1.predict(X_test_vectorized)
#print('F1 :', f1_score(y_test, pred1,average='macro'))
```

```
F1 : 0.7156383968342981
```

為了使結果更佳，我們改為預測機率再對結果做調整。

```
test_pred = model1.predict_proba(X_test_vectorized)
pred1 = []
for i in range(len(test_pred)):
    if test_pred[i,0]>0.1565 or (test_pred[i,0]>test_pred[i,1] and
test_pred[i,0]>test_pred[i,2]):
        pred1.append(0)
    elif test_pred[i,2]>0.353 or test_pred[i,2]>test_pred[i,1]:
        pred1.append(2)
    else:
        pred1.append(1)
#print('F1 :', f1_score(y_test, pred1,average='macro'))
```

```
F1 : 0.7400942306689897
```

2. XGBClassifier

```
model2 = XGBClassifier(subsample=0.6,max_depth=13).fit(X_train_vectorized, y_train)
pred2 = model2.predict(X_test_vectorized)
#print('F1 :', f1_score(y_test, pred2,average='macro'))
```

```
F1 : 0.7063962360560193
```

3. LogisticRrgression

```
model3 = LogisticRegression().fit(X_train_vectorized, y_train)
pred3 = model3.predict(X_test_vectorized)
#print('F1 :', f1_score(y_test, pred3,average='macro'))
```

```
F1 : 0.6650381990211757
```

4.3 將結果合併

```
pred = []
for i in range(len(pred1)):
    p = [pred1[i],pred2[i],pred3[i]]

    if p.count(0)>0:
        pred.append(0)
    elif p.count(2)>1:
        pred.append(2)
    elif p.count(1)>1:
```

```
        pred.append(1)
    else:
        pred.append(p[1])
y_test=np.array(y_test)
#print('F1 :', f1_score(y_test, pred,average='macro'))
```

```
F1 : 0.7468469830338492
```

4.4 最終預測及輸出

```
results = pd.DataFrame(data = {'id' : test.id, 'class' : pred})
results.to_csv('results.csv', index = False)
```

5. 預測結果及分析

5.1 驗證資料得分

```
#print('Test Confusion = \n{}'.format(metrics.confusion_matrix(pred, y_test)))
#print(classification_report(y_test, pred))
```

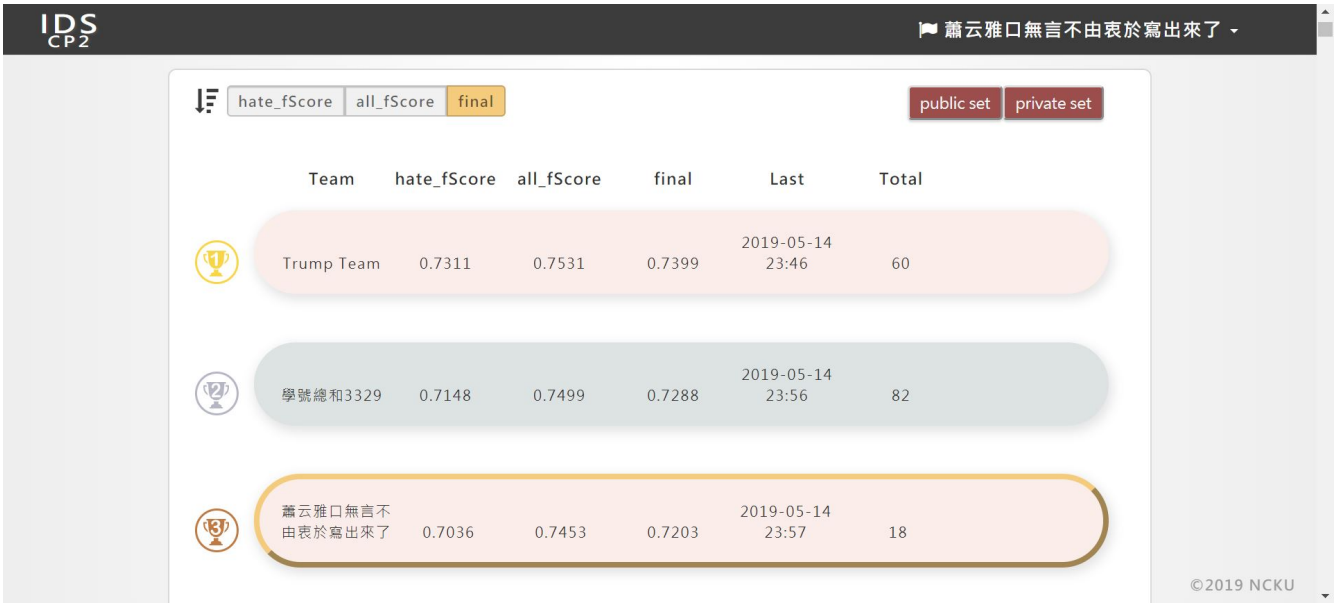
```
Test Confusion =
[[ 130  133   30]
 [ 127 3164   80]
 [  19  108  670]]
              precision    recall  f1-score   support

           0       0.44      0.47      0.46       276
           1       0.94      0.93      0.93      3405
           2       0.84      0.86      0.85       780

   micro avg       0.89      0.89      0.89      4461
   macro avg       0.74      0.75      0.75      4461
weighted avg       0.89      0.89      0.89      4461
```

5.2 測試資料上傳得分

-Public Set

5	蕭云雅口無言不由衷於寫出來了	0.7206	0.7648	0.7382	2019-05-14 23:57	18

-Private Set



5.3 結果分析

上傳結果HateF的項目不太確定詳細的算法，但顯然是和我們用的算法不一樣，AllF倒是沒有太大的差距，我們的驗證集的結果應該是和上傳得分沒有太大差距的。而在競賽中為了不讓Public和Private的分數有差距，我們堅持不針對上傳結果回來調整模型，而是完全以驗證資料為基準來校正，我想這是使我們最終在Private Set上排名沒有倒退很重要的一點。

6. 補充

其實在這中間我們還花了不少的時間在研究**BERT**，不過很可惜的是最終我們還是沒有找到合適的機器來跑，也因此浪費了很多時間，最後才趕著在截止之前回來做這些比較簡單的模型，因為時間的不足而在競賽上沒有滿意的結果。

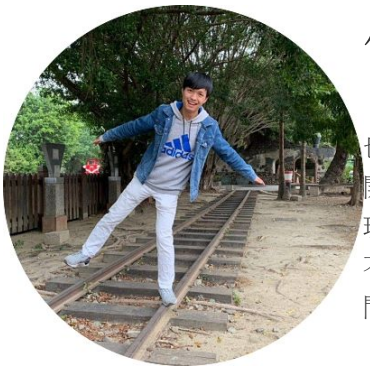
雖然來不及在競賽結束之前做完，但是在結束後我們還是想辦法跑出結果了:)

畢竟是做了很久的東西，還是會很好奇到底能得到怎樣的分數，在這裡也分享一下最後的結果～

```
Test Confusion =
[[  66   28   27]
 [ 449 7487  133]
 [  52  184 1488]]
              precision    recall  f1-score   support

           0       0.55      0.12      0.19       567
           1       0.93      0.97      0.95      7699
           2       0.86      0.90      0.88      1648

   micro avg       0.91      0.91      0.91      9914
   macro avg       0.78      0.66      0.67      9914
weighted avg       0.90      0.91      0.90      9914

==================================================
Test Confusion =
[[ 286  270   32]
 [ 225 7215   79]
 [  56  214 1537]]
              precision    recall  f1-score   support

           0       0.49      0.50      0.50       567
           1       0.96      0.94      0.95      7699
           2       0.85      0.93      0.89      1648

   micro avg       0.91      0.91      0.91      9914
   macro avg       0.77      0.79      0.78      9914
weighted avg       0.91      0.91      0.91      9914
```

模型預測出來的是Probability，上半部是用np.argmax()直接轉換成分類得到的結果，而下半部則是和我們在這裡的模型一樣，調整過分類界線所得到的結果。其實並沒有我們預先所想像的那麼神奇xD，不過F1能有0.3的進步確實也是不小了，和競賽的第一名相比分數也是高出了一些，尤其這個結果是來自於完全沒有經過處理的資料，如果在把資料餵進去之前先做初步的處理說不定還能夠有更好的結果～(不過train一次實在要太久了，我們就來不及再跑一次了哈哈)

7. 心得與感想



小穎

這次的主題是關於文字處理方面，一開始覺得很新鮮，因為自己本身對於這方面也陌生很多，也在想，如果是文字的話就很難像數字一樣量化，或是有倍率關係，一開始想了很多方法，把一些重複字刪掉啊，刪除奇怪的符號、暱稱等等，找了很多處理文字的模型什麼的，再套用入傳統的一些模型，利用logistic regression之類的，不過其實結果都大同小異，也因為最近很多期中考和作業的關係，所以其實付出的時間比上次少很多，不過我發現其他組也是XDD希望下次可以好好認真做哈哈！

艾霓

超可惜的啊啊啊啊啊啊啊啊啊啊啊啊啊啊啊啊啊啊啊啊啊啊

當初真的把希望都放在Bert上了哈哈哈，結果電腦完全跑不起來XD，禮拜一還拿者兩個記憶體跑去聽驛站，真的很好笑！

這次的競賽真的是在一片混亂之中度過的，覺得真的還有很多可以調整得更好的地方，可能自己更需要學習的是如何在有限的時間裡把事情都完成吧，像是這次的競賽到了最後一天才又要回來做原本的model，每個模型train一次就要花十幾分鐘甚至幾十分鐘，真的會來不及調整，結果就是明明知道還能做得更好卻也只能眼睜睜看著時間一分一秒過去，有點小失落。

在這個模型裡面，最有趣的部分其實是在CountVectorize的地方，那裡面不管是詞幹提取或是詞形還原，通通都一次一起做了，我覺得這樣的方法真的很酷！這個部分其實來自github，我在上面看到了roytsai用過這樣的做法(https://github.com/Roytsai27/Rating-Prediction-from-Movie-Review-) ，覺得很厲害就拿到了這裡使用xD，至於tf-idf因為在這個資料上表現得不好所以沒有採用。

再接再厲！！