

HW6

1. 小組成員：

統計113孫亞瑄H24091304、統計113陳亭瑄H24096150、統計113龍以欣H24094051

2. 題目敘述與目標：

這次的作業為仇恨言論偵測，利用機器學習將需要預測的資料判斷為Hateful (y=0)、Offensive (y=1)、Clean (y=2)，並期望得到較高的Final Score。

3. 資料前處理：

首先先嘗試用sklearn.feature_extraction.text中的CountVectorizer及TfidfVectorizer兩種方式對資料進行bag of words的處理(下圖)。

```
#CountVectorizer
count=CountVectorizer()
count.fit(train_X)
train_X=count.transform(train_X)
test_X=count.transform(test_X)
#training
clf=XGBClassifier(eval_metric='mlogloss',use_label_encoder=False)
clf.fit(train_X,train_Y)
pred_Y=clf.predict(test_X)
score(pred_Y,test_Y)

HateF: 0.6445422566712113
AllF: 0.7162835873461506
Final: 0.673238788941187

#TfidfVectorizer
tfidf=TfidfVectorizer()
tfidf.fit(train_X)
train_X=tfidf.transform(train_X)
test_X=tfidf.transform(test_X)
#training
clf=XGBClassifier(eval_metric='mlogloss',use_label_encoder=False)
clf.fit(train_X,train_Y)
pred_Y=clf.predict(test_X)
score(pred_Y,test_Y)

HateF: 0.6492579852579852
AllF: 0.7163783998274461
Final: 0.6761061510857695
```

嘗試後發現在不調任何參數的情況下，經TfidfVectorizer處理後的產生的feature比經CountVectorizer處理產生的feature能獲得更高的分數(模擬評分標準)，因此之後都採用TfidfVectorizer做前處理。

在觀察原始資料後，發現有許多訊息中包括RT及@開頭的使用者名稱，還有許多數字，因此我們也嘗試使用nltk，將簡訊中@開頭的使用者名稱拿掉，以及將stop_word中加入RT及I(因原本stop_word中無大寫的I)，和將數字拿掉。

```
#add stopwords
stop_words=set(stopwords.words("english"))
stop_words.update(["RT","I"])

#text preprocessing
for i in range(len(train_X)):
    token=word_tokenize(train_X[i])
    for j in range(len(token)):
        if token[j]=="@" and j<len(token)-1:
            token[j+1] ="."
    train_X[i]=' '.join(token)

for i in range(len(test_X)):
    token=word_tokenize(test_X[i])
    for j in range(len(token)):
        if token[j]=="@" and j<len(token)-1:
            token[j+1] ="."
    test_X[i]=' '.join(token)

for i in range(len(train_X)):
    token=RegexpTokenizer(r"\w+").tokenize(train_X[i])
    token=[w for w in token if w not in stop_words]
    token=[w for w in token if w.isdigit()==False]
    train_X[i]=' '.join(token)

for i in range(len(test_X)):
    token=RegexpTokenizer(r"\w+").tokenize(test_X[i])
    token=[w for w in token if w not in stop_words]
    token=[w for w in token if w.isdigit()==False]
    test_X[i]=' '.join(token)
```

4. 特徵處理與分析：

首先先嘗試使用CountVectorizer及TfidfVectorizer兩種處理方式，發現經TfidfVectorizer處理後的產生的feature比經CountVectorizer處理產生的feature能獲得更高的分數，這跟原先預期的相同，因為TF-IDF有考慮到出現頻率，因此可以較平衡的給出每個字的權重，所以能獲得較高的分數。

在觀察原始資料後，發現有許多訊息中包括RT及@開頭的使用者名稱，還有許多數字，因為我們推測RT和@開頭的使用者名稱還有數字應該不會是成為判斷是否為仇恨言論的重要feature，拿掉的話應該會降低資料中的雜訊，因此我們用nltk將其去除，然而去除後的feature放入模型學習的成果反而不如原先，這讓我們蠻意外的。

推測可能是我們不小心將某些可以判斷是否為仇恨言論的資訊去除了(例如某些特定數字)，使判斷的feature變少所以分數才降低了。

5. 預測訓練模型：

我們使用xgboost進行分類，因為xgboost可以直接進行多類別的分類。

```
#training
clf=XGBClassifier(eval_metric='mlogloss',use_label_encoder=False,max_depth=6,gamma=0.1,eta=0.3)
clf.fit(train_X,train_Y)
pred_Y=clf.predict(test_X)
```

調整參數的部分，我們針對max_depth(最深深度)、gamma(懲罰項係數)及eta(學習速率)這些參數進行修改。

此外，也使用LogisticRegression、RandomForest等模型，但皆未有更佳的準確率。

6. 預測結果分析：

在這個階段，我們有自行切分訓練資料，套用模型進行預測。

```
#load data
train=pd.read_csv("train.csv")
X=train["tweet"]
Y=train["class"]
train_X,test_X,train_Y,test_Y=train_test_split(X,Y,train_size=0.8,stratify=Y,random_state=2)
```

而評分標準則是模擬教授公布的評分標準寫成函數。

```
#score
def score(pred_Y,test_Y):
    #AllF
    list_score_all=f1_score(pred_Y,test_Y,average=None)
    score_all=np.mean(list_score_all)

    #HateF
    h_pred_Y=pred_Y.copy()
    h_test_Y=test_Y.copy()
    h_pred_Y[h_pred_Y==2]=1
    h_test_Y[h_test_Y==2]=1
    list_score_hate=f1_score(h_pred_Y,h_test_Y,average=None)
    score_hate=np.mean(list_score_hate)

    print("HateF:",score_hate)
    print("AllF:",score_all)
    print("Final:",0.6*score_hate+0.4*score_all)
```

最後獲得的分數如下。

```
#training
clf=XGBClassifier(eval_metric='mlogloss',use_label_encoder=False)
clf.fit(train_X,train_Y)
pred_Y=clf.predict(test_X)
score(pred_Y,test_Y)

HateF: 0.6445422566712113
AllF: 0.7162835873461506
Final: 0.673238788941187
```

另外我們也自己嘗試寫for迴圈和用我們寫出的評分標準函數進行cross validation，記錄不同的模型參數得到的分數。

```
#cross validation
k=5
n_samples=len(X)
fold_size=n_samples//k
scores=[]
masks=[]
for fold in range(k):

    #generate a boolean mask for the test set in this fold
    test_mask=np.zeros(n_samples,dtype=bool)
    test_mask[fold*fold_size:(fold+1)*fold_size]=True

    #create training and testing sets using this mask
    test_X,test_Y=X[test_mask],Y[test_mask]
    train_X,train_Y=X[~test_mask],Y[~test_mask]

    #TfidfVectorizer
    tfidf=TfidfVectorizer()
    tfidf.fit(train_X)
    train_X=tfidf.transform(train_X)
    test_X=tfidf.transform(test_X)

    #training
    clf=XGBClassifier(eval_metric='mlogloss',use_label_encoder=False)
    clf.fit(train_X,train_Y)
    pred_Y=clf.predict(test_X)
    scores.append(score(pred_Y,test_Y))

#original
print(scores)
print("mean:",np.mean(scores))
print("var:",np.var(scores))

[0.6796019929301387, 0.6761227431118297, 0.6361164994319732, 0.6638705473304485, 0.6461884264531902]
mean: 0.6603800418515161
var: 0.0002839242226195708
```

最後我們也嘗試自己寫gridsearch來找出最佳參數。

```
#grid search
best_hyperparameters=[0]
list_depth=[4,5,6,7,8]
list_gamma=[0.05,0.1,0.3,0.5]
list_eta=[0.05,0.1,0.15,0.2,0.25,0.3]
for depth in list_depth:
    for gamma in list_gamma:
        for eta in list_eta:
            k=5
            n_samples=len(X)
            fold_size=n_samples//k
            scores=[]
            masks=[]
            for fold in range(k):

                #generate a boolean mask for the test set in this fold
                test_mask=np.zeros(n_samples,dtype=bool)
                test_mask[fold*fold_size:(fold+1)*fold_size]=True

                #create training and testing sets using this mask
                test_X,test_Y=X[test_mask],Y[test_mask]
                train_X,train_Y=X[~test_mask],Y[~test_mask]

                #TfidfVectorizer
                tfidf=TfidfVectorizer()
                tfidf.fit(train_X)
                train_X=tfidf.transform(train_X)
                test_X=tfidf.transform(test_X)

                #training
                clf=XGBClassifier(eval_metric='mlogloss',use_label_encoder=False,max_depth=depth, gamma=gamma, eta=eta)
                clf.fit(train_X,train_Y)
                pred_Y=clf.predict(test_X)
                scores.append(score(pred_Y,test_Y))

if np.mean(scores)>best_hyperparameters[0]:
    best_hyperparameters=[np.mean(scores),depth,gamma,eta]
```

而以下是我们找到的最佳參數。

```
print(best_hyperparameters)
#score,depth,gamma,eta

[0.6649232053280147, 6, 0.1, 0.3]
```

7. 感想與心得：

陳亭瑄：

在這次的作業中，因為許多考試及期末專題的關係，導致我們有的時間相當有限，而且這次的作業是要進行文字處理再用機器學習的方式去做判斷，對我而言也是相當有難度，我們利用上課所教的TFIDF及NLTK進行前處理並使用XGBoostClassifier完成這次的比賽，在初步的模型撰寫完之後，我也嘗試利用GridSearchCV去找出最佳的參數值，但可能因為給定參數範圍過大導致程式跑的時間很久，所以最後並沒有成功得到GridSearchCV的結果，此外我也用嘗試其他的模型，如：LogisticRegression、RandomForest等等，但得出的結果都沒有XGBoost來的好。另外，網站上的三個分數似乎有設定錯誤，導致三者的分數都是一樣的，以至於我們並不能清楚的知道各項確切的成績。如果有將這次的比賽安排在期末專題前完成，就能讓我們有更充足的時間進行討論，應該會達到更好的成果。

龍以欣：

這次的競賽是我們這組第三個比賽，漸漸的比較熟悉比賽流程，會先做資料前處理，先了解整個資料的內容，再開始分析和預測，做起來的速度比之前的還快一些，也比較上手，而這次比賽是做字串的處理，感覺有一點的難度，但是使用上課時所教的NLTK來處理，將字串轉換成數字來進行預測，照著助教所教的步驟處理，使我對字串處理比較沒有那麼害怕。我們使用了XGBoost 的多類別的預測方法，在先前就有先自學多類別的預測，也比較了解其背後的原理，使用起來比較不會太困難，最後因為這次的作業是壓在期末考周後，時間比較緊湊，可惜的地方是沒有使用到資料不平衡的處理。

孫亞瑄：

這次的競賽是進行文字處理的分類預測，從開學的課程走到現在，學會了許多方式可以嘗試，從資料前處理初次使用sklearn的tfidfvectozizer，以及使用nltk進行文字的細部處理，雖然非常不熟，但是都讓我印象深刻，而這些處理比起上次競賽也更加有難度(也可能是上次競賽經驗不夠，沒有對資料進行太多前處理)，十分有趣。

而在模型訓練的部分，因為我們開始的時間較晚我們主要使用xgboost進行，原先有點苦惱該怎麼對多類別進行分類，沒想到稍微查一下就發現xgboost可以直接進行多類別分類，真是太強拉~而我們也模擬評分標準自己寫出了score函式及做了cross validation還有gridsearch，感覺自己將教授期末教的內容嘗試後更加的印象深刻！

這是最後一次的資導作業了，雖然每次作業都要花大量的時間(可能是我太笨)，不過我覺得這些都讓我十分感興趣，也可以體會到教授出這些作業及競賽的用心，只有在將時間和精力投入後才能獲得這些回報，也變成大二上的美好回憶，最後想說教授和助教辛苦了！