# Hw5-Team11 Arteon

陳泓吟 工資 112 H34086254

陳宜姍 統計 114 C14101137

陳凱騫 統計 114 H24101222

## 一.競賽目標與敘述

此次的競賽題目為仇恨言論預測。競賽的訓練資料集為 14870 筆推特留言，原始資料中紀錄了每筆留言的仇恨值分類，分類有三種，0 為 hateful，1 為 offensive，2 為 clean。競賽目標為使用這些原始資料進行模型訓練，使用模型預測測試資料中留言的仇恨值分類。

## 二. 資料前處理

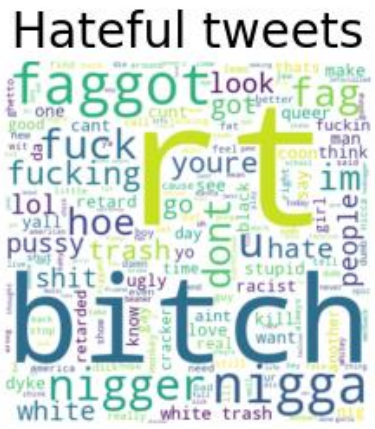
資料前處理的第一步我們先使用正則表達式清理文字資料，清除不相關符號及網址等等，並且也去除停用詞，上述步驟皆為避免不相關的符號及詞語影響預測結果。

接著在 nlp 的應用中會使用 stemming 或 lemmatize 的方式把所有不同時態或是不同變化的字詞變成同一個字。stemming 作用主要是去掉字詞後面的小字母，例如 called 變 call。Lemmatize 則主要是將字詞字根化。我們在前處理的部分測試了 SnowballStemmer 和 WordNetLemmatizer 兩種方式，以下也會說明不同的前處理方法對應的模型結果。

## 三.特徵處理與分析

特徵觀察的部分在文字清理完畢後，我們使用文字雲的方式呈現出所有留言字詞出現的頻率，已及區分三個 class 留言的字詞出現頻率。分別如下圖所示。從圖可看出每個分類留言最常出現的字詞。


圖(一)所有留言文字雲

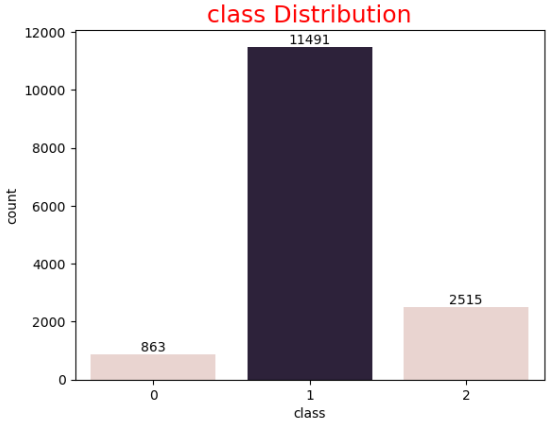



圖(二)class=0 文字雲


圖(三)class=1 文字雲


圖(四)class=2 文字雲

我們也繪出目標欄位 class 的分布情形，如圖(五)所示。可以發現三個類別的比例並不平衡，因此我們也有嘗試使用 NearMiss 及 SMOTE 處理目標欄位比例失衡的問題，這部分不同的處理所對應的模型結果也會在以下進行詳述。




圖(五)目標欄位分布

最後我們要將輸入的文字資料經過 Vectorizer 萃取轉換，這裡我們嘗試了 CountVectorizer 和 TfidfVectorizer 兩種不同的處理方式。CountVectorizer 是較簡單的方式，純粹將文字資料轉換成 token counts，TfidfVectorizer 則是會賦予不同字詞不同的重要度。

綜合以上我們所有資料前處理及特徵處理的方法如下表，以下預測訓練模型會說明我們有測試到的前處理方法組合及其對應的模型。

| 處理字詞變化 | 處理目標欄位不平衡 | 文字萃取 |
|---|---|---|
| SnowballStemmer | NearMiss | CountVectorizer |
| WordNetLemmatizer | SMOTE | TfidfVectorizer |

## 四.預測訓練模型

選用模型的過程中我們使用訓練資料自行進行切割測試，切割成 0.67 比例的訓練資料及 0.33 比例的測試資料。

資料前處理方法

| 處理字詞變化 | 處理目標欄位不平衡 | 文字萃取 |
|---|---|---|
| SnowballStemmer | 無 | CountVectorizer |
| SnowballStemmer | SMOTE | CountVectorizer |

測試 DecisionTreeClassifier、XGBClassifier 及 BaggingClassfier+ DecisionTreeClassifier。皆使用預設參數沒有另外進行調整。

資料前處理方法

| 處理字詞變化 | 處理目標欄位不平衡 | 文字萃取 |
|---|---|---|
| WordNetLemmatizer | 無 | TfidfVectorizer |
| WordNetLemmatizer | NearMiss | TfidfVectorizer |

測試 DecisionTreeClassifier 、 RandomForestClassifier 、 ExtraTreesClassifier BaggingClassfier + DecisionTreeClassifier 及 XGBClassifier。皆使用預設參數沒有另外進行調整。

LogisticRegression

使用 GridSearchCV 測試參數，測試範圍如下：

| | |
|---|---|
| C | [0.001,0.01,0.1,1,10] |

最後在 LogisticRegression(C=10)獲得最佳結果。

BaggingClassfier+ LogisticRegression

使用上述的 LogisticRegression 模型，再使用 GridSearchCV 測試參數，測試範圍如下：

| | |
|---|---|
| n_estimators | [50,100,150] |

最後在 n_estimators= 100 獲得最佳結果

StackingClassifier

StackingClassifier 結合 DecisionTreeClassifier 及上述的 BaggingClassifier 和 DecisionTreeClassifier 結合 LogisticRegression

資料前處理方法

| 處理字詞變化 | 處理目標欄位不平衡 | 文字萃取 |
|---|---|---|
| WordNetLemmatizer | 無 | CountVectorizer |

LogisticRegression

使用 GridSearchCV 測試參數，測試範圍如下：

| | |
|---|---|
| C | [0.001,0.01,0.1,1,10] |

最後在 LogisticRegression(C=10)獲得最佳結果。

BaggingClassfier+ LogisticRegression

使用上述的 LogisticRegression 模型，再使用 GridSearchCV 測試參數，測試範圍如下：

| | |
|---|---|
| n_estimators | [50,100,150,200] |

最後在 n_estimators= 200 獲得最佳結果

## 五.預測結果分析

在我們進行特徵處與分析時因未發現目標欄位有比例失衡的狀況，所以原先以為使用 NearMiss 或 SMOTE 的方法平衡目標欄位分類可以提升預測效果。但實測的結果使用 NearMiss 或 SMOTE 後預測分數反而大幅降低。

以以下模型為例，這裡使用的資料前處理方式是 WordNetLemmatizer 和 TfidfVectorizer，模型是 ExtraTreesClassifier 並使用預設參數。在未經過 NearMiss 處理前獲得的自行測試預測結果及網站分數為下圖及下表：

```
ExtraTreesClassifier Accuracy: 0.8868962706337885
              precision    recall  f1-score   support

           0       0.51      0.24      0.33       271
           1       0.91      0.95      0.93      3787
           2       0.83      0.80      0.81       849

    accuracy                           0.89      4907
   macro avg       0.75      0.66      0.69      4907
weighted avg       0.87      0.89      0.88      4907
```

| HateFscore | AllFscore | Final |
|---|---|---|
| 0.664 | 0.704 | 0.680 |

將訓練資料經過 NearMiss 處理後的預測結果為：

```
ExtraTreesClassifier Accuracy: 0.7458732423069085
              precision    recall  f1-score   support

           0       0.16      0.57      0.25       271
           1       0.95      0.74      0.83      3787
           2       0.72      0.83      0.77       849

    accuracy                           0.75      4907
   macro avg       0.61      0.71      0.62      4907
weighted avg       0.87      0.75      0.79      4907
```

| HateFscore | AllFscore | Final |
|---|---|---|
| 0.576 | 0.617 | 0.592 |

再以 SnowballStemmer 及 CountVectorizer 為資料前處理方式，SMOTE 為處理目標欄位不平衡的方式做前後對比。模型是 BaggingClassifier + DecisionTreeClassifier 並使用預設參數。在未經過 SMOTE 處理前獲得的自行測試預測結果及網站分數為下圖及下表：

```
bagging Accuracy : 0.8901569186875892
              precision    recall  f1-score   support

           0       0.42      0.31      0.35       271
           1       0.94      0.93      0.94      3787
           2       0.80      0.88      0.84       849

    accuracy                           0.89      4907
   macro avg       0.72      0.71      0.71      4907
weighted avg       0.88      0.89      0.89      4907
```

| HateFscore | AllFscore | Final |
|---|---|---|
| 0.680 | 0.719 | 0.695 |

將訓練資料經過 SMOTE 處理後的預測結果為：

| HateFscore | AllFscore | Final |
|---|---|---|
| 0.529 | 0.600 | 0.558 |

```
bagging Accuracy : 0.7263093539841043
              precision    recall  f1-score   support

           0       0.14      0.58      0.22       271
           1       0.95      0.71      0.82      3787
           2       0.78      0.83      0.80       849

    accuracy                           0.73      4907
   macro avg       0.62      0.71      0.61      4907
weighted avg       0.88      0.73      0.78      4907
```

除了以上兩組測試外我們也有在其他的模型組合嘗試使用 NearMiss 或 SMOTE 處理目標欄位比例失衡的問題，但結果皆如上述兩組模型相同，通過處理後的資料進行模型訓練預測分數就會大幅降低。推測原因可能是因為 f1-score 是 precision 和 recall 的共同計算出來的綜合指標，而在處理目標欄位不平衡問題時我們發現最明顯的變化是分類為 0 的 recall 值可以達到提升，但是相對其 precision 就會降低。另外分類為 1 的 recall 值則似乎也會受影響而降低。最終加總計算起來就會導致最後的總指標是下降的。這點是我們比較意外的，因為原先是認為將比例失衡的目標欄位處理後應該會有較好的結果。至於是否有方法可以達到上述指標間升降的折衷，我們在時間截止前尚未找到處理的辦法，但這或許是往後我們在遇到相同題目或情況時可以再加強探究的點。

另外我們在模型訓練的地方有提到我們測試了不同前處理的方法對模型預測結果的影響，以下是我們其中幾組的測試結果。我們以(SnowballStemmer 和 CountVectorizer)及(WordNetLemmatizer 和 TfidfVectorizer)兩種前處理方式搭配 DecisionTreeClassifier 及 XGBClassifier 皆使用預設參數。得到的預測結果為下表：(上兩行為 SnowballStemmer 和 CountVectorizer 搭配 DecisionTreeClassifier 及 XGBClassifier;下兩行為另外一組資料處理方法)

| HateFscore | AllFscore | Final |
|---|---|---|
| 0.655 | 0.691 | 0.669 |
| 0.651 | 0.694 | 0.668 |
| 0.647 | 0.693 | 0.665 |
| 0.623 | 0.668 | 0.641 |

透過比較我們認為在本題目中資料前處理的方式對模型預測效果的影響似乎也沒有很顯著。

最後以下是我們獲得最佳分數的模型:

**資料前處理方法**

| | | |
|---|---|---|
| SnowballStemmer | 無 | CountVectorizer |

**使用模型**

```
clf=DecisionTreeClassifier()
bag = BaggingClassifier(base_estimator= clf)
```

**預測結果**

| HateFscore | AllFscore | Final |
|---|---|---|
| 0.680 | 0.719 | 0.695 |

## 六. 感想與心得

陳泓吟:

這是我第一次處理文字資料的預測模型，因此整體上比較生疏，對於該使用的方法也必須重頭摸索。目前所使用到的一些文字處理方法感覺都還有待改善的空間。另外綜合兩次競賽作業我認為自己比較可惜的地方有兩項。一是對個模型套件的演算法其實沒有很熟悉，雖然會使用但對於這些模型確切是如何進行分類及參數的意義都還沒有十分了解。第二是對於實驗該如何設計缺乏概念，資料前處理、模型、模型參數在預測的過程中都有非常多選擇，而在這些選擇中該如何挑選來進行實驗設計及分析比較是我所不熟悉的，因此常常會測試一堆模型但不知道要從何比較起獲是該如何呈現在報告中是比較清楚的。因此若要給予這兩次作業一個小小的建議，我的建議是老師之後的課程或許可以稍微提及要如何進行眾多模型的實驗比較或是提供歷年作業表現較佳的組別報告讓大家做為格式或方法參考，這樣在書面報告的書寫上方向也會更加明確！

陳宜姍:

我是第一次處理文字資料，覺得文字處理真的比想像中難很多。一開始花了不少時間想了解並運用 nltk 套件，但是對於 dataframe 和 text 兩種類型的轉換花了很多時間，我現在認為有點不值得，因為有許多事情不見得要依靠套件才能處理。我上網查方法，試著將資料中常出現的字一一列出來，想運用在前處理上，沒想到最後在機器學習的時候又遇到瓶頸，未能成功運用，有一點挫折。顯然我對於文字處理以及機器學習還有太多不熟悉、要學習的地方，希望以後有機會能再補強這一部分。

陳凱騫:

我覺得這次的作業對我來說十分的有挑戰性，因為從以前到現在，幾乎沒處裡過文字上的處理，因此這次的作業可說是一個高牆，難以突破，在寫這次作業時，事先查了許多資料，像是文字雲的處理、刪除不必要運算的文字、文字的分級判斷等，並且這些文字在實際上在處理上又比看起來還要難的多，因此光是做出一次的預測就已耗費多時，而我也因這次的作業對程式有了更深的見解，雖然最後的準確地仍舊十分不堪，然而在查資料、資料處理、嘗試不同的模組等過程中，我仍舊獲益良多，也希望之後我也可以保持寫程式的熱度，將其技術運用在各個不同的領域。