# HW5 Report

## 1. Members

- 統計 112 劉恩兆 B14081027
- 統計 112 宋穎恩 B14081302

## 2. Introduction

此次的競賽主題為「仇恨言論辨識」，我們需要訓練一個模型，以辨識 Twitter 留言資料是屬於仇恨言論、冒犯言論還是一般言論。在競賽中，我們須將文字資料轉換成可供模型使用的數字資料，因此，文字資料處理方式與模型挑選將很大地影響競賽分數。

此次競賽的資料集含有訓練集與測試集，分別有 14869 筆集 9914 筆，而我們需要在測試集預測的即為訓練集中的「class」欄位，共有 0、1、2 三種分類，分別為仇恨言論、冒犯言論及一般言論，以下為訓練集圖示：

圖 1 、訓練集示意圖

| class | tweet |
|---|---|
| 1 | [9-1-13] 2:50 pm "son of a bitch ate my mac n cheese" http://t.co/My5oJYZ8w9 |
| 1 | RT @BryceSerna: Don't be a pussy grab the booty. Love the booty. Appreciate the booty. |
| 2 | RT @ClicquotSuave: bunch of rappers boutta flood the internets w/ trash remixes |
| 2 | @michigannews13 wow. Thats great language coming from a HS coach, you sure are a role model, you're trash, just like your account |
| 1 | and this is why I'm single, I don't fuck with bitches or there attitudes foh. |
| 1 | RT @__thaRealist: @Dono_44 yea that hoe was rocking Friday and last night |
| 1 | They should free all da real niccas n give the snitches da time |
| 0 | @bellaangeletti ur a fag |

⬆ To be predicted

## 3. Data Preprocessing

### 3-1. 將資料分出 input 及 label

訓練資料的 input 為訓練集中的「tweet」欄位，label 為訓練集中的「class」欄位；而測試資料的 input 為訓練集中的「tweet」欄位，output 為模型預測的「class」結果。

### 3-2. 檢查遺失值

使用「is.na()」檢查資料集是否含有遺失值需要填補或是刪除，結果於訓練集與測試集皆並未發現含有遺失值。

### 3-3. 文字清洗

使用「正則表達式 (regular expression)」依贅字規律，將不必要的資訊清除，以免模型學習到錯誤特徵。清除的不必要資訊包含：回文 tag 用戶名稱、回文時間、有規律但不具意義之文字、網址、emoji 等等。此外，因為後續有使用 Bert 模型，因此也有嘗試將標點符號去除，原因是 Bert 為 transformer 模型，而 transformer 模型通常會使用

WordPiece 分割法將文本分割成若干個子詞，並將這些子詞轉換成數值向量。但 WordPiece 分割法只會考慮文本的字彙和語法，不會考慮標點符號，因此，Bert 模型在處理資料時通常會去掉標點符號，而後續也有實驗是否去除標點符號之比較，來實證模型是否能夠捕捉到標點符號對文本情緒的影響。

表 1、文本資料清洗前後比較示意表

| | |
|---|---|
| **清理前** | @WendyDavisTexas @GregAbbott_TX a judge that was appointed by obama who here doesnt see the agenda fucking crooked bitch |
| **清理後** | a judge that was appointed by obama who here doesnt see the agenda fucking crooked bitch |

【註：灰色字表示會被清理的不必要資訊】

# 4. Methodology

## 4-1. TF-IDF

**TF-IDF 介紹：**TF-IDF (term frequency–inverse document frequency) 是一種統計方法，用以評估單詞對於一份文件重要程度的衡量手法。

- Term Frequency (TF) 詞頻：計算並找到出現次數最多的詞
- Inverse Document Frequency (IDF) 逆文檔頻率：在詞頻的基礎上，對每個詞分配一個重要性的權重，最常見的詞給予最小的權重，較少見的詞給予較大的權重，也就是大小與一個詞的常見程度成反比。
- TF-IDF 的公式如下：$TF\text{-}IDF = TF \times IDF$。其中，TF-IDF 值越高表示該詞越重要。
- 停用詞 (stop words)：停用詞是指在 TF-IDF 文本分析中不會對分析產生任何貢獻的詞，通常是一些非常常見(詞頻高)的詞，如："the"、"and"、"but"等。
  在計算 TF-IDF 時，我們通常會將停用詞排除在外，因為停用詞出現的頻率很高，但對文本意義的貢獻很小，因此不值得考慮。

**實作流程：**

1. 資料清洗：如 **3.3** 所述，使用正則表達式清除不必要的資訊，包含：回文 tag 用戶名稱、回文時間、有規律但不具意義之文字、網址、emoji 等等，以及使用 Python 內建的 nltk (Natural Language Toolkit) 英文停用詞列表作為索引將停用詞清除
2. 將文字資料轉為詞頻矩陣：從 sklearn 套件 import CountVectorizer 創建詞袋數據庫，並透過 CountVectorizer 中的 transform 函數將文本中的詞語轉換成詞頻矩陣
3. 將詞頻矩陣轉為 TF-IDF 矩陣：從 sklearn 套件 import TfidfVectorizer，並利用

TfidfVectorizer 建構一個計算 TF-IDF 的函式，依詞頻出現多寡給予各單詞不同的權重，最終將詞頻矩陣轉換成 TF-IDF 權重矩陣。藉由上述的步驟，即可將文字資料轉為數字組成矩陣，方可以放進模型計算

4. 將轉換成數字矩陣的資料放入模型，使用的模型有：Random Forest、LightGBM、MLP

### 4-2. Bert

**Bert 介紹：**Bert (Bidirectional Encoder Representations from Transformers) 是一種 Google 開發的自然語言處理的神經網路模型，使用了 transformer 的神經網路結構，與傳統的 RNN、LSTM 不同，Bert 可以雙向的處理序列(上下文)，更加靈活地從輸入的序列中獲取更多資訊，並且可以從網路上下載他人訓練過的預訓練包，大幅提升效率及模型表現。

**實作流程：**

1. 資料清洗：如 **3.3** 所述，使用正則表達式清除不必要的資訊，包含：回文 tag 用戶名稱、回文時間、有規律但不具意義之文字、網址、emoji 等等
2. 將輸入的句子使用預訓練的 BertTokenizer 進行斷詞、分詞，並將每個詞轉換為一個向量，並且 padding 到最長長度 512
3. 將輸入的句子作為序列輸入到 Bert 模型中，通過多層的注意力機制，來自動學習輸入序列中所有位置之間的關係
4. 將輸出的向量通過分類器進行分類預測

## 5. Analysis of Prediction Results

### 5-1. TF-IDF

**模型表現結果：**使用 TF-IDF 方法將文字資料轉換成數字矩陣的資料放入模型，其中使用的模型有：Random Forest、LightGBM、MLP

表 2、TF-IDF 模型表現結果

| Model | HateFscore | AllFscore | Final |
|---|---|---|---|
| **RandomForestClassifier()** | 0.5492 | 0.5616 | 0.5541 |
| **LGBMClassifier()** | **0.6640** | **0.7186** | **0.6858** |
| **MLPClassifier()** | 0.5217 | 0.5977 | 0.5521 |

【註：Random Forest、LightGBM 皆使用預設參數，MLP 則為(50,50)，iteration=20】

由表 2 可見，經 TF-IDF 轉換後表現最佳的模型為 LightGBM，而 MLP 因需要較適合的參數調整，因此於後續調整兩模型之參數進行比較。

表 3、MLP 模型不同參數表現結果

| parameter | HateFscore | AllFscore | Final |
|---|---|---|---|
| **hidden_layer_sizes=(20,10) max_iter=15** | 0.4857 | 0.2923 | 0.4084 |
| **hidden_layer_sizes=(50,50) max_iter=20** | **0.5217** | **0.5977** | **0.5521** |
| **hidden_layer_sizes=(100,50) max_iter=50** | 0.4857 | 0.5819 | 0.5242 |

【註：其他使用參數固定如下：solver='sgd', activation='relu',alpha=1e-4】

由表 3 可見，經參數調整後，可以發現不論是提高 layer 間的 neurons，或是提高 max_iteration，MLP 模型皆未有良好的表現，因此改往調整 LightGBM 之參數。

表 4、LightGBM 模型不同 learning rate 表現結果

| parameter | HateFscore | AllFscore | Final |
|---|---|---|---|
| **learning_rate = 0.0001** | 0.4857 | 0.2923 | 0.4084 |
| **learning_rate = 0.001** | 0.4857 | 0.2923 | 0.4084 |
| **learning_rate = 0.1** | **0.6640** | **0.7186** | **0.6858** |
| **learning_rate = 0.25** | 0.6629 | 0.7168 | 0.6845 |

【註：其他使用參數皆為預設值】

由表 4 可見，LightGBM 於 learning_rate = 0.1 表現最佳，因此繼續調整其他參數以追求更佳表現。

表 5、LightGBM 模型不同 learning rate 表現結果

| parameter | HateFscore | AllFscore | Final |
|---|---|---|---|
| **learning_rate = 0.0001** | 0.4857 | 0.2923 | 0.4084 |
| **learning_rate = 0.001** | 0.4857 | 0.2923 | 0.4084 |
| **learning_rate = 0.1** | **0.6640** | **0.7186** | **0.6858** |
| **learning_rate = 0.25** | 0.6629 | 0.7168 | 0.6845 |

【註：其他使用參數皆為預設值】

Learning_rate 的意義為每次迭代時，模型更新的參數量的倍數。由表 5 可見，LightGBM 於 learning_rate = 0.1 表現最佳，因此繼續調整其他參數以追求更佳表現。特別注意的是，原先認為較小的學習率雖然會導致訓練過程較慢，但因其較仔細(梯度提升較慢)，可能會得出較好的結果，但實驗結果將 learning_rate 調過小會導致文本分類全部被分至 1，也就是冒犯言論；而 learning_rate = 0.1 與 learning_rate = 0.25 的表現則相差不大。

表 6、LightGBM 模型不同 n_estimators 表現結果

| parameter | HateFscore | AllFscore | Final |
|---|---|---|---|
| **n_estimators = 100** | 0.6640 | 0.7186 | 0.6858 |
| **n_estimators = 200** | **0.6675** | **0.7228** | **0.6896** |
| **n_estimators = 500** | 0.6591 | 0.7108 | 0.6798 |
| **n_estimators = 1000** | 0.6584 | 0.7062 | 0.6775 |

【註：learning_rate=0.1，其他使用參數皆為預設值】

N_estimators 表示 LightGBM 會在訓練過程中使用的弱學習器個數。由表 6 可見，LightGBM 於 n_estimators = 200 表現最佳，通常來說，提高 n_estimators 可以讓模型表現更好，但實驗結果中，若是 n_estimators 超過 200 後，就會有過擬合產生，導致模型表現不佳。

表 7、LightGBM 模型不同 max_depth 表現結果

| parameter | HateFscore | AllFscore | Final |
|---|---|---|---|
| **max_depth = -1** | 0.6675 | 0.7228 | 0.6896 |
| **max_depth = 1** | 0.5683 | 0.6481 | 0.6002 |
| **max_depth = 3** | 0.6659 | 0.7233 | 0.6889 |
| **max_depth = 5** | **0.6794** | **0.7349** | **0.7016** |
| **max_depth = 7** | 0.6743 | 0.7310 | 0.6970 |

【註：learning_rate=0.1、n_estimators = 200，其他使用參數皆為預設值】

Max_depth 表示 LightGBM 中樹的最大深度，默認為-1，表示無限制，其中，較大

的樹深度較有可能有好的模型表現，但也可能有過擬合之風險。由表 7 可見，LightGBM 於 max_depth=5 時表現最佳，而在實驗結果中，若是 max_depth 超過 5 後，就會有過擬合產生，導致模型表現不佳。

**TF-IDF 模型表現小結：**在 TF-IDF 中，我們嘗試了 3 種模型，分別為 Random Forest、LightGBM 與 MLP，其中，LightGBM 在調整參數後有非常突出的表現，final score 達到 7 成多，但 MLP 與 Random Forest 表現就相對沒那麼亮眼，推測可能是因為 LightGBM 模型複雜度藉由 n_estimators 調整後較高，且 LightGBM 可以自動從資料中學習有用的特徵，而 TF-IDF 較為簡單，無法挑出太有用的特徵，因此若不是人工設計特徵工程，LightGBM 應可以有較高的表現。

## 5-2. Bert

**不同設定下，模型表現結果比較：**

下方比較若無特別說明，則基本設定如下：

- Batch size = 6　(為節省訓練時間，將 batch size 調至硬體最大容忍值)
- Epoch = 10 (據實驗觀察，epoch 若超過 10 輪，並不會有太大變動
- Learning rate = 0.00001
- Loss = CrossEntropy
- Optimizer = Adam
- Neuron = (768,3)
- Dropout = 0.5

表 8、Bert 模型清理無用文字表現結果差異

| 是否有清除無用文字 | HateFscore | AllFscore | Final |
|---|---|---|---|
| **是** | 0.7006 | 0.7511 | 0.7208 |
| **否** | 0.6924 | 0.7390 | 0.7111 |

由表 8 可見，清理無用文字後的確可以避免模型學習到錯誤特徵，進而提高模型表現。

表 9、Bert 模型 padding 長度表現結果差異

| Padding length | HateFscore | AllFscore | Final |
|---|---|---|---|
| **Padding = 512** | 0.6924 | 0.7390 | 0.7111 |
| **Padding = 256** | 0.5878 | 0.6731 | 0.6219 |

在 Bert 模型中，因為模型需要輸入的序列皆為同樣長度，因此會使用 padding 將文本序列填充到相同的長度，其中 padding 最長長度為 512。而由表 9 可見，當

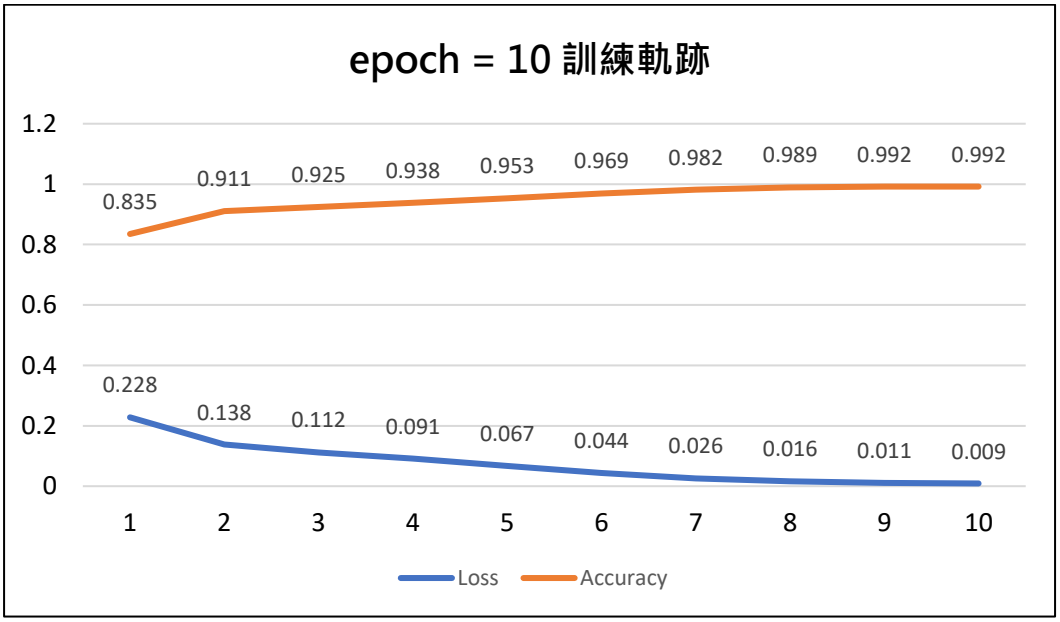
padding=256 時，明顯模型表現有所下降，這是因為 Bert 模型在 padding 設定過小時，較長的文本會被截斷，可能導致有用的訊息被刪除。反之，若是 padding 設定的足夠大，Bert 會對不夠長的文本序列填充 “padding” 之標記，但這些對 Bert 模型是毫無意義的，因此，不會像是 padding 設定過小時可能將有用訊息刪除。
不過，padding 較小也是有其優勢的，優勢為運算速度較快，並且可以調高 batch size 進一步將運算速度提高，也較不受硬體資源限制。

表 10、Bert 模型不同 epoch 表現結果差異

| # of epoch | HateFscore | AllFscore | Final |
|---|---|---|---|
| **Epoch = 2** | 0.5160 | 0.6268 | 0.5603 |
| **Epoch = 5** | 0.6812 | 0.7396 | 0.7045 |
| **Epoch = 10** | **0.6924** | **0.7390** | **0.7111** |
| **Epoch = 12** | 0.6741 | 0.7312 | 0.6969 |

在深度學習中，epoch 表示訓練過程中使用全部訓練資料迭代一次的過程，通常來說 epoch 越多，模型會越貼合數據，但也可能導致過擬合的情形發生。而由表 10 可見，當 epoch=2 時，模型明顯還沒訓練起來，尚未足夠貼合數據，而訓練到 epoch=5 時，已經大致能夠貼合數據了，直到 epoch=10 時，模型很好地貼合數據，並且當 epoch=12 時就有過擬合之狀況發生，以下為 Bert 模型在 epcoh=10 的訓練軌跡。

圖 2、Bert 模型 epoch=10 時的訓練軌跡



由圖 2 可見，training loss 在 epoch=5 時就下降到足夠低了，表示模型已經大致貼合數據，直到 epoch=10 時 training loss 已經非常低，再訓練更多 epoch 可能導致過擬合的狀況發生。

表 11、Bert 模型不同 learning rate 表現結果差異

| Learning rate | HateFscore | AllFscore | Final |
|---|---|---|---|
| **Leaning rate = 0.0001** | 0.4857 | 0.2923 | 0.4084 |
| **Leaning rate = 0.000001** | 0.6924 | 0.7390 | 0.7111 |

在 Bert 模型中，learning rate 是非常重要的超參數，代表著訓練過程對模型更新的步長，高 learning rate 可以導致收斂速度提升，但可能讓模型跳過最佳解，而收斂至次佳解。由表 11 可見，將 learning rate 提升 100 倍後，final score 就從 0.7111 降至 0.4084，是非常懸殊的表現。因此，雖然 learning rate=0.000001 會讓模型訓練很久，不過帶來的表現卻可以輾壓 learning rate=0.0001 的表現。

## 5-3. TF-IDF vs. Bert 最佳模型比較

**各自模型設定：**

- TF-IDF + LightGBM
  - 有清除不必要資訊及去除停用詞
  - LightGBM：learning_rate=0.1、n_estimators = 200、max_depth=5
- Bert
  - 有清除不必要資訊
  - Leaning rate = 0.000001、epoch=10、padding=512

表 12、TF-IDF 與 Bert 最佳模型比較表現結果差異

| method | HateFscore | AllFscore | Final |
|---|---|---|---|
| **TF-IDF + LightGBM** | 0.6794 | 0.7349 | 0.7016 |
| **Bert** | 0.7006 | 0.7511 | 0.7208 |

由表 12 可見，Bert 模型的表現相較 TF-IDF 較高，原因推測為以下幾點：

1. Bert 為 transformer based 模型，可以理解上下文及單詞間的依賴性，因此，對於語言理解與情感分析，是 Bert 大勝 TF-IDF 這種只依詞頻作為特徵的方法之原因

2. Bert 模型有使用 masked language modeling 這種方法，白話來說，就是在訓練過程中將依些單字用 mask 遮住，讓 Bert 預測句子的缺失標記(mask)，以此方法可以更加良好的學習單詞之間的上下文關係，來捕捉句字含意及文意理解
3. Bert 有更多的參數可以調整、使用，簡單來說就是可以更加客製化地依應用場景調整
4. Bert 已利用網路的大數據進行預訓練，奠基良好的解決自然語言任務基礎

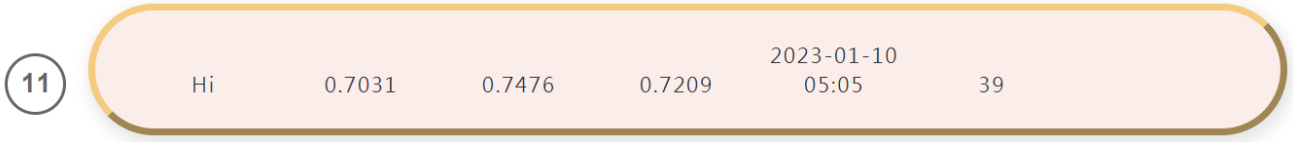
# 6. Conclusion and Thoughts

## 6-1. 最佳模型設定與 public leader board 排名

在這次競賽中，原先使用 TF-IDF + LightGBM 模型就獲得了不錯的排名及表現，但為了追求更高的預測表現，就使用了 Bert 此強項為語意理解的模型來進一步地追求更高排名，以下為 Bert 表現最佳之模型設定：

- Batch size = 6
- Epoch = 10
- Learning rate = 0.00001
- Loss = CrossEntropy
- Optimizer = Adam
- Neuron = (768,3)
- Dropout = 0.5

最終 public leader board 排名如下：




## 6-2. Conclusion

總結來說，這次競賽的應用場景非常適合使用 Bert 模型這種強項為可以理解上下文、語意及情感分析的模型，應能比許多文字處理搭配上機器學習模型的方法來的更加出色，但可惜因 Bert 訓練參數過於龐大，在競賽時程中較難依應用場景細膩的調整參數，不過我認為此模型是還有提升表現的空間的。

## 6-3. Thoughts

恩兆心得：這次的競賽開啟了我第一次文字資料分析的大門，不論是學著使用 regular expression 來清理資料，又或是學著使用簡單的 TF-IDF 文字處理方法來做仇恨言論分類，再到學習使用 Pytorch 框架來寫 Google 開發的 Bert 模型，無不是新的挑戰，而我也從中學習到了許多文字資料分析的基本觀念，以及學習如何更加有效率地查資料並應用。在這次的競賽中，我深深體認到原來資料分析不是挑選一個好的模型即可，更重要的是根據資料特性、應用場景來選擇對應的資料處理及特徵處理，在一開始時就直接一

股腦地使用 Bert，原先以為可以直接霸榜，沒想到只能勉強擠入前十名，到後來試著開始做資料清理後，馬上排名就有提升，雖然在最後一天被後來居上的同學擠出了 public leader board 前三名，但我認為這次的競賽相較上次是更有收穫許多的，也非常好奇其他同學使用的方法，希望老師可以出個方法特輯來介紹一下有甚麼推薦的方法可以使用 XD

最後也想檢討一下這次的競賽表現，一是很可惜這次競賽卡到期末，期間剛好有一堆考試及報告，無法很專心地刷榜，並且使用 Bert 訓練模型也是要花極大的時間以及運算資源，原先嘗試使用 colab 的免費 GPU 來幫忙運算，但只要 padding 開到最大長度，那 colab 都只能訓練個四輪即停止運算，後來只能靠自己的筆電，在睡前訓練模型，並於隔天起床查看結果，由此可見是比較低效率的，也無法即時做出更正，導致運算資源的浪費。此外，自己對文字處理的方法知道的也不夠多，因此無法很精確地依場景挑選出適合的模型、資料處理方式，這也是之後可以持續改進的地方。