國立台灣科技大學資訊工程系

一零八學年度第一學期專題研究

總報告

假新聞分類與真實度評分系統

研　究　組　員

黃信恩	B10532008
王繹崴	B10532024
王韋翔	B10532037

指導老師：陳　錫　明

中　華　民　國　108　年　12　月　15　日

國立台灣科技大學資訊工程系

一零八學年度第一學期專題研究

總報告

假新聞分類與真實度評分系統

研　究　組　員

黃信恩	B10532008
王繹崴	B10532024
王韋翔	B10532037

指導老師：_____

中　華　民　國　108　年　12　月　15　日

摘 要

資訊量越來越多的時代，不時會有虛假的新聞藏於其中讓大眾接受並吸收之；為了搏得大家的目光，新聞的文字也逐漸的聳動，使得人們難以相信其真實性。為了能夠協助人們能夠分析眼前消息的可信度，於是我們決定透過機器學習的方式，設計一個分析新聞可信度的模型。

我們將分析假新聞的模型分成兩部分：一種是分類為何種假新聞(煽動仇恨、扭曲事實、諷刺評論、政府操縱等)的 Classification Model，會藉由 4 種分析字詞關鍵程度的評分方式(TF-IDF , Chi-Square 等方式)構築該類別對應的關鍵字的詞典，並利用 Classification 的方式，讓模型在得到新的文章，看到該關鍵字時可以判斷該文章可能屬於何種新聞。另一種是分析可信度的 Regression Model，用來對該新聞的可信程度進行評分，我們會利用原本在 dataset 中的可信度評分，透過 RNN 或著一般的回歸分析計算出新聞文本中的可信度。我們希望能夠有足夠的正確率，使其可以成為文章可信度的參考依據，再追求更高的正確率。

目錄

圖 目 錄

表 目 錄

第一章　緒論

自然語言處理 (Natural Language Processing) 是包括文本摘要(Text Summarization)、情緒分析(Sentiment Analysis)、翻譯(Translation)等的人工智慧領域，可以應用在聊天機器人、翻譯機、文章的語意分析等方面。

1.1 研究動機

新聞媒體是讓大家知道消息的窗口，不論過去透過報章雜誌、到電視普及於每個家庭、直到現在網路普及，消息的傳播速度越來越快，訊息量也越來越多。其中不乏有為了吸引眼球而過度膨脹的新聞、甚至為了達成某些目標而有惡意中傷他人的謠言。身為發布消息者，應要為新聞的真實性負責，但卻為了利益而捨棄之，並將惡意的假新聞混雜於海量的訊息之中，供人們吸收。

那麼就要靠著閱聽人來分辨是否是假新聞了，但接收的資訊一天比一天多，一定會有難以消化而真假難分的時候。為此我們希望能夠藉由自然語言處理的力量，找出文章中使其立場偏頗的關鍵字，利用機器學習的模型，協助我們能夠對於眼前的資訊有一個可以參考的標準。

1.2 專題目標

由於假新聞的氾濫，我們期待利用機器學習中的深度學習方法，將大量的新聞資料作為訓練資料，來提出一個能辨識新聞真假的系統，期待這個系統能夠對每篇新聞提出正確的判斷，而針對不同的資料集我們有不同的目標，對於資料訓練時的標籤不同，部分資料我們希望辨識出這篇新聞是可信、謊言、仇恨言論等等其中之一，另一部分我們也希望我們的系統能夠給出一個代表這篇新聞真假的數字，數字愈大代表真實程度愈高，相反的數字愈小這篇新聞愈可能是假新聞。

而新聞當中除了文章也有其他不同的項目，例如作者、來源、時間等等，我

們認為這些也會是影響新聞可信度的因素，因此我們也將對各項資料進行分析，利用一些視覺化的方式讓研究更加直觀，同時因為文章的數量相當龐大，需要透過自然語言處理的方式進行分析，所以我們會使用較為複雜的機器學習模型做出學習資料後的判斷，試著找出對於新聞可信度有關的項目，甚至他們的重要程度，最後提出一個結合分析及判斷的系統，讓使用者可以得到他們閱讀的是否為假新聞的建議。

第二章　技術原理與相關研究

2.1 Classification

TF-IDF（Term Frequency–Inverse Document Frequency）[1]是一種統計單字出現頻率以及該單字對文章分類判斷的重要性的方式，將 TF 值和 IDF 值計算出來，再將兩者相乘即可得到 TF-IDF 值。

TF (Term Frequency) [1] 用來計算「單字在一篇文章中出現的次數」，透過除以文章字數的方式避免判斷被單純因文章較長而出現較多次的單字影響。TF 的公式如下:

TF = 文字在文章出現次數/該文章長度

IDF（Inverse Document Frequency）[1]用來計算「有出現該單字的文章數」，這方法用來避免分類判斷被過度常見的文字影響，IDF 值越大，代表著這個單字在所有文章中較為少見，具有關鍵性的影響力。IDF 算法如下[1]：

$IDF = \log_{10}$(文章數/有出現該文字的文章數)

將兩者相乘便可得到 TF-IDF 值，此值越大就代表著這個詞在單篇文章中常見而不一定在許多文章中都有出現，對於幫助文章分類有很大的效益。

Chi-Square 用來探討單字與該類別是否有相依性，藉由假設單字跟類別是獨立的，來計算觀察值與期望值的差距，若差距過大並非互相獨立，而是有相依性，以表 1 來說明[1][2]

表 1　Chi-Square

↓單字\文章→	是這個類別的文章數	不是這個類別文章數
出現此單字的文章數	A(N_{11})	B(N_{01})
沒有此單字的文章數	C(N_{10})	D(N_{00})

- 在此類別有出現此單字的文章數的期望值 M_{11} [2]:

$$M_{11} = N^* (A+ C)/ N^* (A+ B)/ N$$

- “不在"此類別有出現此單字的文章數的期望值 M_{10} [2] :

$$M_{10} = N^* (A+ C)/ N^* (C+ D)/ N:$$

- 在此類別”沒有”出現此單字的文章數的期望值 M_{01}[2] :

$$M_{01} = N^* (A+ B)/N^* (B+ D)/ N$$

- “不在"此類別且”沒有”出現此單字的文章數的期望值 M_{00} [2] :

$$M_{00} = N*(C+ D)/N* (B+D)/N$$

則卡方檢驗值 Chi(X) [2]：

$$\text{Chi(X)} = \sum_{a\epsilon\{0,1\}}\sum_{b\epsilon\{0,1\}}\left(\left(N_{ab} - M_{ab}\right)^2 \Big/ M_{ab}\right)$$

Likelihood Ratio [3]可看作是 Chi-Square 的改良版，更適合用於較分散的資料，且也更容易解釋它也可以表現出單字跟類別的相關程度，若 ratio = 0 則互相獨立，ratio 越高相關性越高，意味著此單字在分類中越重要，Log Likelihood Ratio(LLR)公式如下(A,B,C,D 皆參考表 1) [3]:

$$\text{LLR}=-2\log\left[\frac{\left(A+B/N\right)^{A} * \left(1 - A+B/N\right)^{B} * \left(C+D/N\right)^{C} * \left(1 - C+D/N\right)^{D}}{\left(A/A+B\right)^{A} * \left(1 - A/A+B\right)^{B} * \left(C/C+D\right)^{C} * \left(1 - D/C+D\right)^{D}}\right]$$

Expected Mutual Information (EMI) [4]也可以用來表示單字跟類別之間的關係。EMI 是從 PMI(Pointwise Mutual Information)改良而來，改善了 PMI 可能有負值跟對於罕見字眼會給較高分數的問題，EMI 的公式如下[4]：

$$\text{EMI} = \sum_{e_t\epsilon\{0,1\}}\sum_{e_c\epsilon\{0,1\}} P(e_t, e_c) log_2 \frac{P(e_t \cap e_c)}{P(e_t)P(e_c)}$$

其中e_c表文章是否為此類別，e_t表文章是否有此單字，P 代表其比例

2.2 Regression

Word Embedding 的概念[5]是建立字詞向量（Word Vector），例如我定義一個向量的每個維度對應到什麼字，並且將句子中每個字轉換為向量，最後結合起來變成矩陣。

卷積神經網路[6]包含了卷積層、池化層以及全連接層。其中卷積層是透過卷積運算來提取特徵值，而全連結層則是將抽取出來的特徵放到下一層。

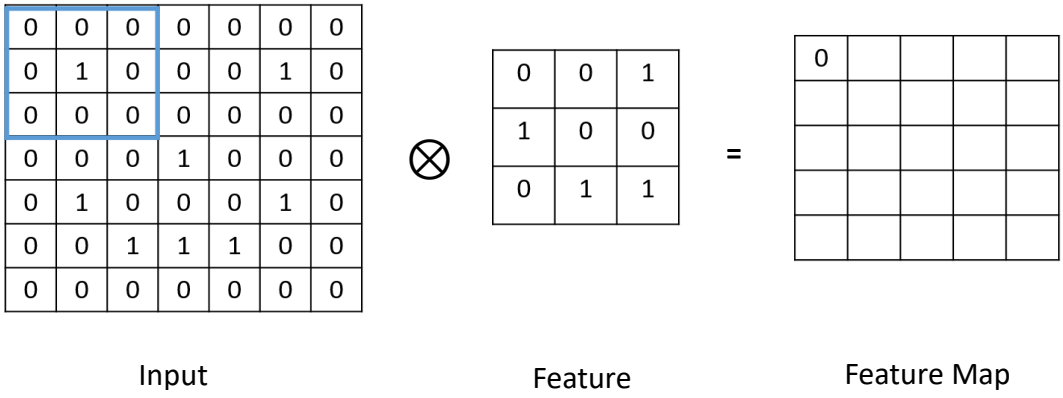


圖 1 卷積運算說明圖[6]。

LSTM [7]是用來改善 RNN 的一種模型，利用 4 個閘門來決定資料是否要加入記憶或是輸出。

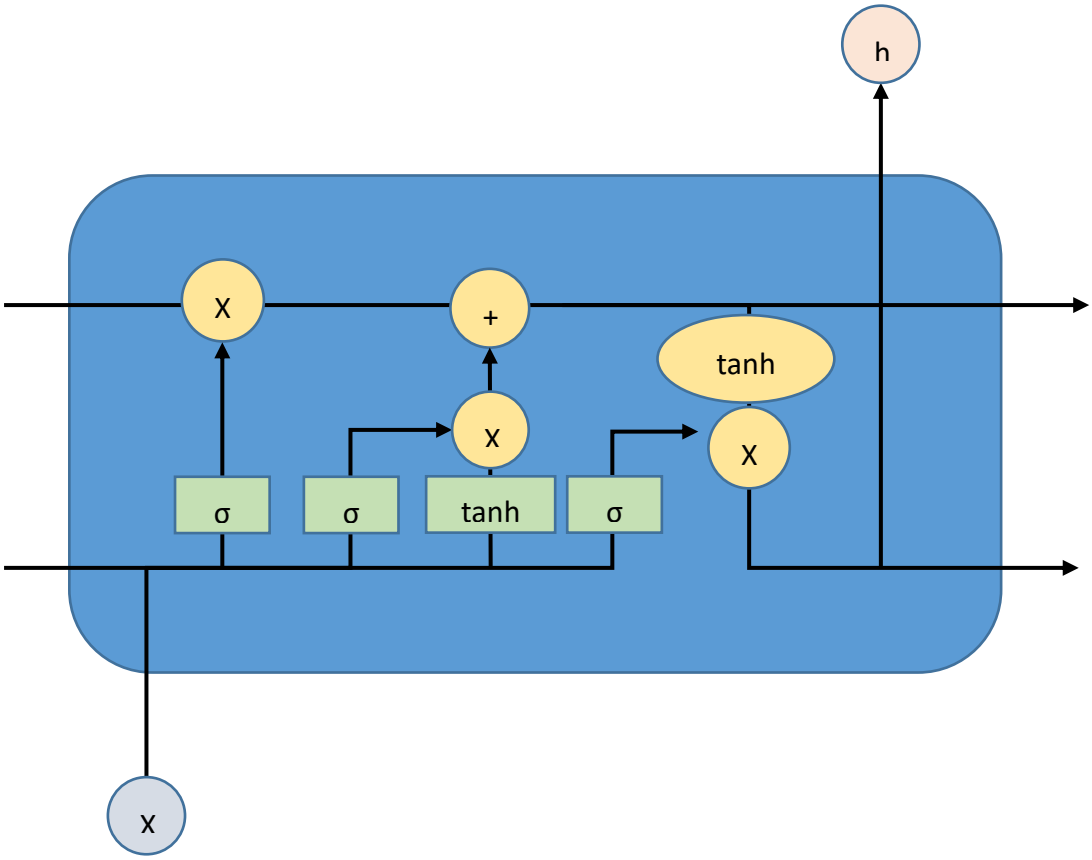


圖 2 LSTM Model 說明圖[7]。

第三章　系統設計

我們先對 3 個資料集各自做需要的 Preprocessing、特徵處理後再將整體假新聞偵察機分成兩部分：Classification 與 Regression 分析。

3.1 資料集

在資料集的選擇部分，我們找到了三個不同的公開的假新聞資料集：

1. UCSB 所發表的 fake news benchmark dataset [8]。
2. Kaggle 競賽所使用的 Fake News dataset [9]。
3. webhose.io API 所爬的 244 個網站再利用 BS detector 進行 labeling 的公開 Kaggle 資料集[10]

3.2 Preprocessing

為了得到更好的成效，前處理是不可或缺的一部分，對各個資料集我們都會先做以下幾種處理：

(1) 資料的空值處理：

在資料的各個欄位若有空值，將會影響之後的模型演算法運算，因此我們將所有為 NAN 的空值填入空字串，而若是在新聞正文的部分為空值，那我們就會刪除整條該新聞的資料。

(2) 文字的小寫轉換：

在英文語言中，句子的首字母會使用大寫，但是機器學習演算法無法辨別大小寫的不同，因此我們將文章中所有字皆轉為小寫。

(3) 去除標點符號：

文章中的標點符號並無特別意義，因此在前處理中去除。

(4) 去除 stop words：

在英文中的停用詞就是像 "a"，"the"，"to"，"their" 等等經常出現，但是又沒有對文章語意有特別影響的代詞或連接詞等，因此我們使用 NLTK 套件中所提供的 stop words 來進行比對去除。

(5) 詞幹提取 (Stemming)：

英文中的單字由於句型或上下文關係，會有不同的變化，但是原來都是相同的單字，例如"called" 和 "call" 二字只有時態不同，我們可以將它變回"call"。

3.3 Classification

對假新聞進行分類，判斷他是屬屬於哪一種假新聞(偏向胡謅、陰謀論、假科學等等)。我們在 Training 時，先對 Training Data 用 4 種 feature selection 的評分去評分，把對分類有較大幫助的文字抽取出來，減少 overfitting 的可能發生：

我們將 Dataset 中 7 種 label 挑出來，並加入 True Label，如表 2 所示，表二為將 Dataset [10]中的 label 翻譯後的類別。

表 2 Label 類別

Bias	政治宣傳與嚴重扭曲事實
Conspiracy	陰謀論
Fake	虛假故事
Hate	仇恨
State	政府監督下產出的新聞
Junk-sci	假科學
Satire	諷刺性評論
True	可信度高的新聞

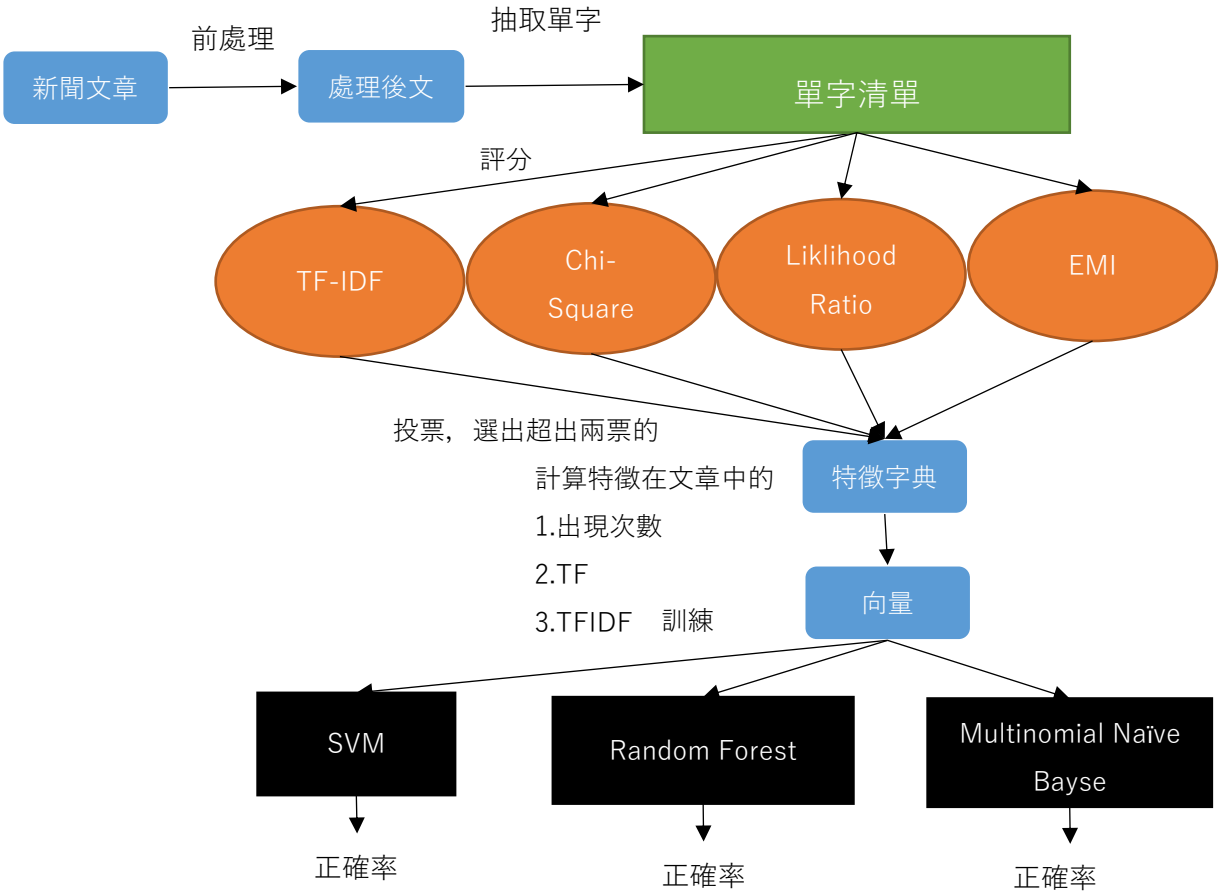
接著先計算每個文字的 TF-IDF 值。除 TF-IDF 外，其他三種投票方式(Chi-Square , Likelihood Ratio , Expected Mutual Information)都是探討「單字是否對領域分類有幫助」所以會考量「單字出現在每個領域的文章數」。

四種計分方式計算完後，接著開始投票，若該單字的四種計算方法中每有一種分數大於所有單字的平均值，票數變加一。投完票後，將票數超過兩票的單字挑選出來，做為 classification 模型的輸入向量。

在 Classification 模型的訓練中，輸入的向量值分成三種型態，分別是：

- 該單字在該篇文章的出現次數。
- 該單字在該篇文章的 TF 值。
- 該單字在該篇文章的 TF-IDF 值。

將這三種型態的 vector 都分別輸入 scikit-learn [11]的 Multinomial Naïve Bayes Classifier , SVM 和 Random forest 進行訓練，比較其訓練與測試的正確率為何，這裡我隨機抽取其中 2 成的資料做為測試集，其他的皆為訓練集，圖 3 所示為 Classification 訓練流程。



。

圖 3 Classification 訓練流程。

3.4 Regression

對新聞文本進行分析，判斷其可信度。首先利用 Keras 中 text.preprocessing [12]。將資料集做 tokenize。接下來因為新聞文本長度不一，因此我們計算出平均長度，並把所有的文本切割成固定平均長度，再將原本少於平均長度的文本補齊到平均長度，以利後續的訓練。

圖 4 所示為我們設計的 Model 架構圖，比較特別的地方是我們採用了雙輸入，實作方式利用利用 Keras 中 text.preprocessing [12]，將我們認為較為重要的

演講者也一併作為訓練用的資訊加入。然後先經過 Embedding layer 建立 word vector，接下來利用 Convolution layer 做卷積提取特徵值，將兩種輸入結合起來之後放到 LSTM layer 以及 GRU layer 做訓練，最後到 Fully Connected layer 準備做最後的輸出。

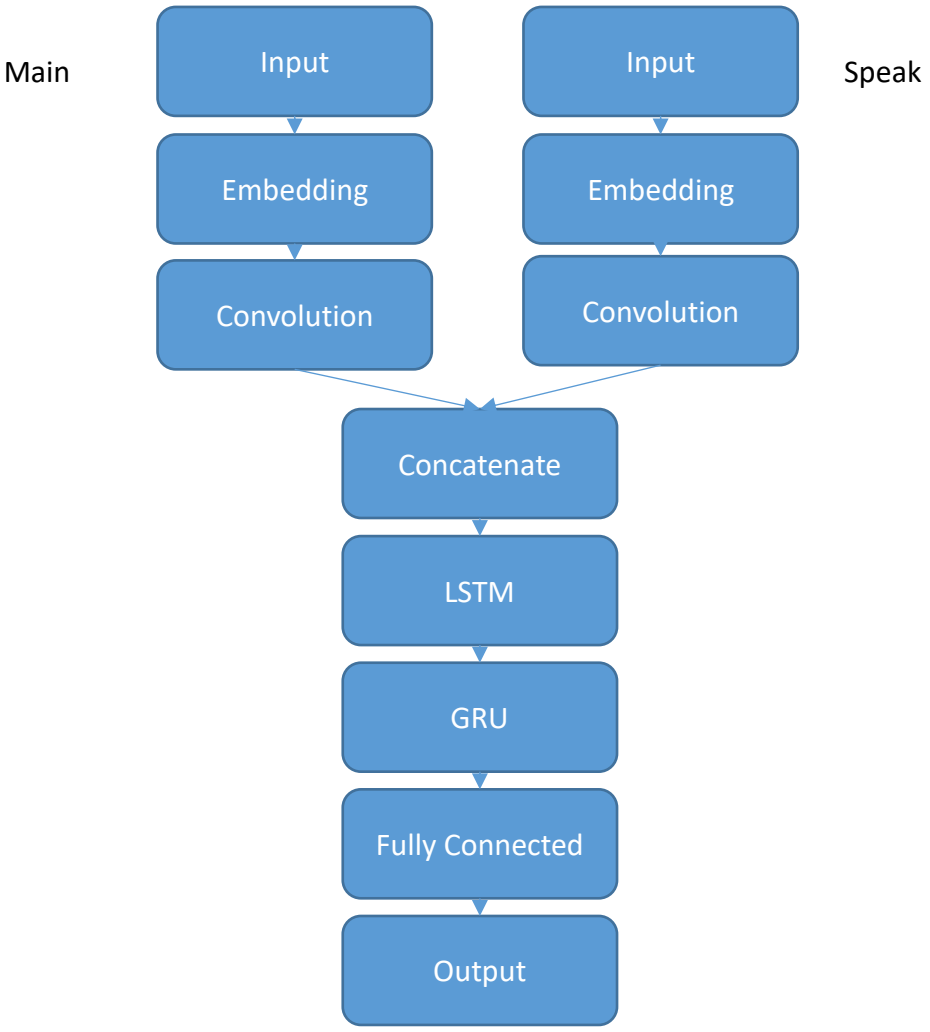


圖 4　Regression Model 架構圖。

第四章　程式碼與專題成果

4.1 Classification

Classification 取用了資料集，其中包含了用於 Regression 評分的資料，我們將 0.8 分以上的資料做為 true label，0.5 分以下的作為 fake label，接著跟其他資料(一組有上述除 true label 外 7 種分類的以及另一組只有分成 true 跟 false label 的資料)合併轉換並且前處理完後開始做 TF-IDF 評分。

```
In [26]: for news in tqdm(combined_df['text'].astype("str")): # tf 與 df 計算
             news_list = news.split(' ')
             term_count = dict(Counter(news_list))
             keys = list(term_count)
             values = list(term_count.values())
             key_term = df_tfidf["term"].isin(keys)
             key_tf = df_tfidf.loc[key_term , "tf"]
             key_tf = key_tf.reset_index().index.tolist()
             # add
             df_tfidf.loc[key_term , "tf"] = df_tfidf.loc[key_term , "tf"] + np.array(values)[key_tf]
             # add df
             df_tfidf.loc[key_term , "df"] = df_tfidf.loc[key_term , "df"] + 1

         100%|██████████| 31435/31435 [22:35<00:00, 23.20it/s]

In [27]: df_tfidf["tf_idf"] = 0
         df_tfidf
```

圖 5 計算出現次數與出現的文章數。

```
In [23]: #evaluate tf-idf
         for news in tqdm(combined_df["text"].astype("str")):
             news_list = news[:-1].split(' ')
             term_count = dict(Counter(news_list))
             #print(term_count)
             keys = list(term_count)
             values = list(term_count.values())
             key_in_term = df_tfidf["term"].isin(keys)
             key_in_df = df_tfidf.loc[key_in_term].reset_index().index.tolist()
             #print(key_in_df)

             #tf = 該篇文章出現次數 / 總字數
             df_tfidf.loc[key_in_term , "tf_idf"] += (np.array(values)[key_in_df] / sum(np.array(values)[key_in_df]) * \
                                                      np.log10(bs_fake.shape[0] / df_tfidf.loc[key_in_term , "df"]))

         df_tfidf["tf_idf"] /= df_tfidf["df"] #最後再除一次出現的檔案數是為了平均詞在每個檔案的TF-IDF
         df_tfidf
```

圖 6 計算 TF-IDF。

圖 11 最後在除以該文字出現的檔案數量表示每個檔案平均的 TF-IDF 數量為何。

接著做 Likelihood Ratio , Chi-Square , Mutual Information 的評分。先做新聞領域的分類：

```
In [38]: class_token_dict = dict() #生成 類別 -> 新聞列表的對應字典
         for i in combined_df["type"].unique().tolist():
             cat = combined_df[combined_df["type"] == i]
             class_token_dict[i] = cat["text"].tolist()
         len(class_token_dict["bias"]) , len(class_token_dict)

Out[38]: (443, 8)
```

圖 7 將類別跟單字做對應。

```
In [39]: count = []
         for k,v in class_token_dict.items():
             count.append(len(v))
         count

Out[39]: [443, 430, 15473, 146, 246, 102, 121, 14474]

In [40]: class_token_dict_terms = {} # 生成 類別 -> 該類別中有出現的單字，並計算有出現的文章個數
         for k , v in class_token_dict.items():
             cat_list = []
             for r in v:
                 r = str(r)
                 li = list(set(r.split(' ')))
                 cat_list.extend(li)
                 class_token_dict_terms[k] = cat_list
         class_token_dict_terms['bias']

Out[40]: ['muslim',
          '',
          'numer',
          'bust',
          'interest',
          'control',
          'fraud',
          'govern',
          'pay',
          'famili',
          'way',
          'steal',
          'money',
          'four',
          'go',
          'year',
          'benefit',
          'systemit',
          'refugeesimmigr',

In [41]: n = Counter(class_token_dict_terms["bias"])
         n

Out[41]: Counter({'muslim': 20,
                  '': 443,
                  'numer': 11,
                  'bust': 4,
                  'interest': 31,
                  'control': 17,
```

圖 8 計算該分類單字出現次數。

```
for i in range(len(class_token_dict)):
    catergory = combined_df["type"].unique().tolist() # 所有catergory list
    num_c = count[i]  #該領域總文章數
    num_not_c = sum(count) - num_c
    term_df['n11'] = 0 # n11 = 該領域文章中有出現該term的文章數
    term_df['n10'] = 0 # n10 = 該領域文章中 "沒有" 該term的文章數
    term_df['n01'] = 0 # n01 = "非 "該領域文章中有該term的文章數
    term_df['n00'] = 0 # n00 = "非 "該領域文章中 "沒有" 該term的文章數
    n11 = dict(Counter(class_token_dict_terms[catergory[i]])) #算出tokenize後文章每個term的出現次數
    keys = list(n11)
    values = list(n11.values())
    key_term = term_df["term"].isin(keys)
    key_index = term_df.loc[key_term].reset_index().index.tolist()
    term_df.loc[key_term , "n11"] = np.array(values)[key_index]
    term_df["n10"] = num_c - term_df["n11"]
   # print(term_df["n10"])

    catergory.remove(catergory[i])
    n01 = itemgetter(*catergory)(class_token_dict_terms)

    n01 = sum(list(n01) , [])
    n01 = dict(Counter(n01))
    keys = list(n01)
    values = list(n01.values())
    key_term = term_df["term"].isin(keys)
    key_index = term_df.loc[key_term].reset_index().index.tolist()
    term_df.loc[key_term , "n01"] = np.array(values)[key_index]
    term_df["n00"] = num_not_c - term_df["n01"]
```

圖 9 計算觀察值。

觀察值代表著「在這分類及這分類以外，分別幾篇文章存在/不存在此單字」

```
    n11 = term_df["n11"] + 1 # log liklihood ratio
    n01 = term_df['n01'] + 1
    n10 = term_df['n10'] + 1
    n00 = term_df['n00'] + 1
    N = term_df['N'] = n11 + n01 + n10 + n00

    score = ((((n11+n01)/N) ** n11) *               \
             ((1 - ((n11+n01)/N)) ** n10) *         \
             (((n11+n01)/N) ** n01) *               \
             ((1 - ((n11+n01)/N)) ** n00))

    score /= (((n11/(n11+n10)) ** n11) *            \
              ((1 - (n11/(n11+n10))) ** n10) *      \
              ((n01/(n01+n00)) ** n01) *            \
              ((1 - (n01/(n01+n00))) ** n00))

    score = -2 * np.log10(score)
    term_df['score_llr'] += score
```

圖 10 Log Liklihood Ratio。

```
   n11 = term_df['n11'] + 1e-6 #chi-Square
   n01 = term_df['n01'] + 1e-6
   n10 = term_df['n10'] + 1e-6
   n00 = term_df['n00'] + 1e-6
   N = term_df['N'] = n11 + n01 + n10 + n00
   e11 = N * (n11+n01)/N * (n11+n10)/N #計算期望值
   e10 = N * (n11+n10)/N * (n10+n00)/N
   e01 = N * (n11+n01)/N * (n01+n00)/N
   e00 = N * (n01+n00)/N * (n10+n00)/N
   term_df['score_chi'] += ((n11-e11)**2)/e11 + ((n10-e10)**2)/e10 + ((n01-e01)**2)/e01 + ((n00-e00)**2)/e00
```

圖 11 Chi-Square。

```
    n11 = term_df['n11'] + 1e-8
    n01 = term_df['n01'] + 1e-8
    n10 = term_df['n10'] + 1e-8
    n00 = term_df['n00'] + 1e-8
    N = term_df['N'] = n11 + n01 + n10 + n00

    m11 = (n11/N) * np.log2(((n11/N)/((n11+n01)/N * (n11+n10)/N))) #計算mutual Information
    m10 = n10/N * np.log2((n10/N)/((n11+n10)/N * (n10+n00)/N))
    m01 = n01/N * np.log2((n01/N)/((n11+n01)/N * (n01+n00)/N))
    m00 = n00/N * np.log2((n00/N)/((n01+n00)/N * (n10+n00)/N))
    term_df['score_emi'] += m11 + m10 + m01 + m00
```

圖 12 Expected Mutual Information。

開始投票 找平均值以上的

In [49]:
```
df_no_min["vote"] = 0
tfidf_avg = np.mean(df_no_min["tf_idf"])
chi_avg = np.mean(df_no_min["score_chi"])
llr_avg = np.mean(df_no_min["score_llr"])
emi_avg = np.mean(df_no_min["score_emi"])
```

In [50]:
```
vote_tfidf = df_no_min[df_no_min["tf_idf"] >= tfidf_avg]
vote_chi = df_no_min[df_no_min["score_chi"] >= chi_avg]
vote_llr = df_no_min[df_no_min["score_llr"] >= llr_avg]
vote_emi = df_no_min[df_no_min["score_emi"] >= emi_avg]
```

In [51]:
```
df_no_min.loc[vote_tfidf.index,'vote'] += 1
df_no_min.loc[vote_chi.index, 'vote'] += 1
df_no_min.loc[vote_llr.index , 'vote'] += 1
df_no_min.loc[vote_emi.index , 'vote'] += 1
df_no_min
```

Out[51]:

	term	tf_idf	score_chi	score_llr	score_emi	vote
0	aljas	0.008255	45.947080	31.512815	0.001483	1
1	alliancesbut	0.012789	4675.671425	217.858910	0.011599	2
2	americamurderess	0.004873	100.264129	63.568856	0.003170	1
3	amerindian	0.003805	72.112049	45.824264	0.002408	1
4	amorit	0.012463	8.171042	8.911371	0.000276	0
5	amphibientunnel	0.040745	155.334707	95.884330	0.005004	3
6	ampliaci	0.004342	51.518284	33.386198	0.001652	1
7	analysiert	0.026400	9.028636	8.077238	0.000323	1
8	anantnag	0.133815	11.892974	10.108557	0.000388	1
9	anfangen	0.026121	40.103569	25.553455	0.001309	2
10	angelastro	0.061314	95.538844	55.663915	0.003002	2
11	angstrom	0.031466	103.578333	63.154566	0.003293	2
12	anthropomorph	0.005402	68.176320	39.960841	0.002108	1
13	anticathol	0.009972	132.045009	inf	0.004355	3
14	anticlimact	0.030016	12.611017	12.349306	0.000401	1

圖 13 進行投票若該單字每有一個高於平均值的項目，投票數便+1。

In [ ]:
```
#把fake news中的text取出形成list
token_list = temp.tolist()
token_list
```

In [25]:
```
whole_string = "".join(token_list)
token_set = set(whole_string.split(' '))
token_list = list(token_set)

for token in token_list:
    if (len(token) < 3):
            token_list.remove(token)

token_list = sorted(token_list)
token_list
```

圖 14 濾除過短單字。

In [52]:
```
vote_over_2 = df_no_min[df_no_min['vote'] > 2]
vote_over_2
vote_over_2.reset_index(inplace = True , drop = True)
vote_over_2
```

1666	wilcox	0.021111	189.446034	28.794264	0.000599	3
1667	wisconsin	0.014552	189.446034	28.794264	0.000599	3
1668	wont	0.018037	577.110110	41.166674	0.001185	3
1669	wor	0.019663	576.609102	41.007621	0.001173	3
1670	workforc	0.021831	295.604672	32.707799	0.000782	3
1671	wouldnt	0.024496	448.206042	36.931563	0.000981	3
1672	wouldv	0.053147	322.054078	27.890504	0.000572	3
1673	wwf	0.017024	258.243286	32.105778	0.000832	3
1674	youd	0.034269	426.748453	36.423774	0.000958	3
1675	youv	0.031967	703.889962	41.398835	0.001221	3

1676 rows × 6 columns

圖 15 抽取出超過 2 票的單字，接近 1700 個。

```
In [5]: dict_over_2 = pd.read_csv("./Dataset/term_vote_over_2.csv")
        dict_over_1 = pd.read_csv("./Dataset/term_vote_over_1.csv")
        bs_train = pd.read_csv("./Dataset/combined_news.csv")

In [6]: dict_over_2
```

Out[6]:

	term	tf_idf	score_chi	score_llr	score_emi	vote
0	amphibientunnel	0.040745	155.334707	95.884330	0.005004	3
1	anticathol	0.009972	132.045009	inf	0.004355	3
2	antiislam	0.022367	3065.063033	inf	0.010990	4
3	apparatu	0.005325	11177.076659	inf	0.031226	3
4	aprovech	0.080187	152.180561	inf	0.004856	4
5	arrangementth	0.018695	1816.693496	141.892459	0.007506	3
6	artania	3.167371	126.958453	78.996724	0.004043	3
7	articleand	0.006446	1118.856804	inf	0.005250	3
8	asiabriefingnytim	0.004944	112.771234	inf	0.003622	3
9	atheisticalantimalelesbianfemin	0.001068	125.121120	inf	0.003968	3

```
In [7]: vector = dict_over_2.filter(["term"])
```

圖 16 單字轉化成特徵向量。

```
In [15]: bs_df = bs_df.reindex(columns = df_col , fill_value = 0)
         bs_df
```

Out[15]:

	text	amphibientunnel	anticathol	antiislam	apparatu	aprovech
0	print pay back money plu interest entir famili...	0	0	0	0	0
1	attorney gener loretta lynch plead fifth barra...	0	0	0	0	0
2	red state fox news sunday report morn anthoni ...	0	0	0	0	0
3	email kayla mueller prison tortur isi chanc	0	0	0	0	0

圖 17 轉化成 dataframe 的 column。

```
In [19]: tf_df = bs_df.copy() #TF 作為向量
         tfidf_df = bs_df.copy()# TF-IDF 作為向量
```

```
In [20]: for news in tqdm(bs_df['text'].astype("str")):
             news_list = news.split(' ')
             term_count = dict(Counter(news_list))
             keys = list(term_count)
             values = list(term_count.values())
         #    bs_df.loc[bs_df['text'] == news , 'length'] += sum(np.array(values))
             total = sum(np.array(values))

             for term in keys:
                 if (term in term_set):
                     bs_df.loc[bs_df['text'] == news , term] = (bs_df.loc[bs_df['text'] == news , term] + term_count[term])
                     tf_df.loc[bs_df['text'] == news , term] = (tf_df.loc[bs_df['text'] == news , term] + term_count[term])
                     vector.loc[vector["term"] == term,"df"] += 1

         100%|██████████| 31435/31435 [33:02<00:00, 19.39it/s]
```

```
In [22]: bs_df[bs_df['wont'] != 0]
```

tiislam	apparatu	aprovech	arrangementth	artania	articleand	asiabriefingnytim	...	wisconsin	wont	wor	workforc	wouldnt	wouldv	wwf	youd	youv	type
0	0	0	0	0	0	0	...	0	1	0	0	0	0	0	0	0	bias
0	0	0	0	0	0	0	...	0	1	0	0	0	0	0	0	0	conspiracy
0	0	0	0	0	0	0	...	0	1	0	0	0	0	0	0	0	conspiracy
0	0	0	0	0	0	0	...	0	1	1	0	0	0	0	0	0	bias
0	0	0	0	0	0	0	...	0	1	0	0	0	0	0	0	0	bias
0	0	0	0	0	0	0	...	0	2	0	0	0	0	0	0	0	conspiracy

圖 18 計算特徵出現次數與 TF 值。

```
In [4]: tfidf_df = tf_df.copy()
        tfidf_df
```

7	7	7	break nypd readi make arrest weiner casehillar...	0.0	0.0	0.0
8	8	8	limbaugh said revel wikileak materi start hurt...	0.0	0.0	0.0
9	9	9	email peopl sick evil stop noth get way law me...	0.0	0.0	0.0
10	10	10	nan	0.0	0.0	0.0
11	11	11	comedian would move spain buy hou anoth countr...	0.0	0.0	0.0

```
In [5]: for term in vector['term']:
            idf = vector.loc[vector["term"] == term , 'idf']
            if(float(idf) < 0):
                print(float(idf))
                tfidf_df = tfidf_df.drop([term] , axis = 1)
            else:
                idf = np.full(tfidf_df.shape[0] , idf)
                tfidf_df.loc[: , term] = tfidf_df.loc[: , term] * idf
```

圖 19 計算特徵 TF-IDF 值。

接著開始訓練，我們將所有資料的 80%用作訓練，剩下的 20%用作測試：

只看詞出現次數

```
In [11]: testY.unique()
Out[11]: array(['true', 'fake', 'bias', 'conspiracy', 'hate', 'junksci', 'satire',
                'state'], dtype=object)

In [12]: NBModel = MultinomialNB()
         NBModel.fit(trainX , trainY)
Out[12]: MultinomialNB(alpha=1.0, class_prior=None, fit_prior=True)

In [13]: resultY = NBModel.predict(testX)

In [14]: accuracy_score(testY , resultY)
         result_graph.loc['MultinomialNB','counting'] = accuracy_score(testY , resultY)

In [15]: RandomForest = RandomForestClassifier(max_depth = None  , min_samples_split = 2 , min_samples_leaf = 1)

In [16]: RandomForest.fit(trainX , trainY)
         /home/alfred/anaconda3/envs/my_env/lib/python3.6/site-packages/sklearn/ensemble/forest.py:246: FutureWarning: The
         default value of n_estimators will change from 10 in version 0.20 to 100 in 0.22.
           "10 in version 0.20 to 100 in 0.22.", FutureWarning)
Out[16]: RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
                    max_depth=None, max_features='auto', max_leaf_nodes=None,
                    min_impurity_decrease=0.0, min_impurity_split=None,
                    min_samples_leaf=1, min_samples_split=2,
                    min_weight_fraction_leaf=0.0, n_estimators=10, n_jobs=None,
                    oob_score=False, random_state=None, verbose=0,
                    warm_start=False)

In [28]: svc = SVC(C = 1.0, kernel = 'rbf' , degree = 3 , max_iter = -1)
         svc.fit(trainX , trainY)
```

```
In [23]: tfidf_MultinomialNB = MultinomialNB()
         tfidf_MultinomialNB.fit(train_tfidf_X , train_tfidf_Y)
Out[23]: MultinomialNB(alpha=1.0, class_prior=None, fit_prior=True)

In [24]: tfidf_RandomForest = RandomForestClassifier(max_depth = No
         tfidf_RandomForest.fit(train_tfidf_X , train_tfidf_Y)
         /home/alfred/anaconda3/envs/my_env/lib/python3.6/site-pack
         default value of n_estimators will change from 10 in versi
           "10 in version 0.20 to 100 in 0.22.", FutureWarning)
Out[24]: RandomForestClassifier(bootstrap=True, class_weight=None,
                     max_depth=None, max_features='auto', max_leaf_
                     min_impurity_decrease=0.0, min_impurity_split=
                     min_samples_leaf=1, min_samples_split=2,
                     min_weight_fraction_leaf=0.0, n_estimators=10,
                     oob_score=False, random_state=None, verbose=0,
                     warm_start=False)

In [32]: tfidf_SVC = SVC()
         tfidf_SVC.fit(train_tfidf_X , train_tfidf_Y)
```

圖 20 將所有資料的 80%用作訓練，剩下的 20%的用作測試。

再將這三種值各自用 SVM、Random Forest、Multinomial Naïve Bayes 做訓練：

```
result = RandomForest.predict(testX)
accuracy_score(testY , result)
result_graph.loc['RandomForest','counting'] = accuracy_score(testY , result)
```

```
result = svc.predict(testX)
accuracy_score(testY , result)
result_graph.loc['SVC','counting'] = accuracy_score(testY , result)
```

```
result_graph
```

圖 21 得到正確率。

4.2 Regression

Regression 訓練的部份我們分成訓練以及預測兩個檔案來做：

Regression_training.py

```
voc_list = training.Stemmed_Statement.tolist()
voc_list_str = [str(x) for x in voc_list]
tokenizer = Tokenizer(num_words=None)
tokenizer.fit_on_texts(voc_list_str)
tokens_enc = tokenizer.texts_to_sequences(voc_list_str)

pickle.dump(obj=tokenizer, file=open('output/tokenizer.pkl', 'wb'))

speaker = training.Speaker.tolist()
speaker_str = [str(x) for x in speaker]
tokenspeaker = Tokenizer(num_words=None)
tokenspeaker.fit_on_texts(speaker_str)
tokens_sp = tokenspeaker.texts_to_sequences(speaker_str)

pickle.dump(obj=tokenspeaker, file=open('output/token_sp.pkl', 'wb'))
```

圖 22 tokenize。

針對 main text 以及 speaker 做 tokenize：

```
max_len = 20
train_X = []
train_Y = []
train_sp = []
for tokens, label, sp in zip(tokens_enc, trainY, tokens_sp):
    length = len(tokens)
    iter_ = int(length / max_len)
    if iter_ > 0:
        temp = tokens
        for it in range(iter_):

            pre = temp[:max_len]
            temp = temp[max_len:]

            train_X.append(pre)
            train_Y.append(label)
            train_sp.append(sp)
            if (len(temp) <= max_len) and (len(temp) >= 9):

                train_X.append(temp)
                train_Y.append(label)
                train_sp.append(sp)
    else:
        if len(tokens) >= 9:
            train_X.append(tokens)
            train_Y.append(label)
            train_sp.append(sp)
```

圖 23 切割文本。

計算出平均長度後，將文本進行切割：

```
inputs_sp = Input(shape=(max_sp_len,))
emb_vec_sp = Embedding(input_dim=np.max(
    train_sp)+1, output_dim=128, input_length=max_sp_len)(inputs_sp)
conv_sp = Conv1D(64, kernel_size=(9,), strides=1, padding='causal',
                 data_format='channels_last')(emb_vec_sp)
bn2 = BatchNormalization()(conv_sp)


inputs = Input(shape=(max_len,))
emb_vec = Embedding(input_dim=np.max(
    train_X)+1, output_dim=128, input_length=max_len)(inputs)
conv = Conv1D(64, kernel_size=(9,), strides=1, padding='causal',
              data_format='channels_last')(emb_vec)
bn1 = BatchNormalization()(conv)

bn = Concatenate(axis=1)([bn1, bn2])
```

圖 24 Model 雙輸入。

實作出 model 的雙輸入部分：

```
BiLSTM = Bidirectional(LSTM(64, return_sequences=True, dropout=0.1,
                            recurrent_dropout=0.1, kernel_initializer='lecun_normal'))(bn)
bn = BatchNormalization()(BiLSTM)
rnn = Bidirectional(GRU(64, return_sequences=False, dropout=0.2,
                        recurrent_dropout=0.2, kernel_initializer='lecun_normal'))(bn)
bn1 = BatchNormalization()(rnn)
dense = Dense(128, activation='selu', kernel_initializer='lecun_normal')(bn1)
bn2 = BatchNormalization()(dense)
do = Dropout(0.3)(bn2)
bn = Concatenate()([do, bn1])
dense = Dense(64, activation='selu', kernel_initializer='lecun_normal')(bn)
do = Dropout(0.4)(dense)
bn = BatchNormalization()(do)
bn = Concatenate()([bn, bn2])
dense = Dense(64, activation='selu', kernel_initializer='lecun_normal')(bn)
do = Dropout(0.5)(dense)
bn = BatchNormalization()(do)
output = Dense(1, activation='relu', kernel_initializer='lecun_normal')(bn)
```

圖 25 Model 其餘 layer。

接著丟到 LSTM layer 以及 GRU layer，再利用 Dropout 以及 BatchNormalization 避免 overfitting：

```
# FIT
# 1st

opt = Adam()
batchSize = 2048
patien = 30
epoch = 1000

saveP = 'model/Reg_keras.h5'
logD = './model/logs/'
history = History()
print("input:", train_X.shape[1])

model.compile(optimizer=opt, loss='mse', metrics=['mae'])
callback = [
    ReduceLROnPlateau(monitor='loss', factor=0.5, patience=int(
        patien/3), min_lr=1e-6, mode='min'),
    EarlyStopping(patience=patien, monitor='val_loss', verbose=1),
    ModelCheckpoint(saveP, monitor='val_mean_absolute_error',
                    verbose=1, save_best_only=True, save_weights_only=True),
    TensorBoard(log_dir=logD),
    history,
]
model.fit([train_X, train_sp], train_Y,
          epochs=epoch,
          batch_size=batchSize,
          shuffle=True,
          validation_data=([valid_X, valid_sp], valid_Y),
          callbacks=callback,
          )
model.save(saveP+"_all.h5")
```

圖 26 1st fit。

```
# 2nd time
# * lower lr
# * lower patience

opt = Nadam(lr=0.0009)
batchSize = 512
patien = 15
epoch = 100

saveP = 'model/Reg_keras2.h5'
logD = './model/logs/'
history = History()
print("input:", train_X.shape[1])

model.load_weights('model/Reg_keras.h5')
model.compile(optimizer=opt, loss='mse', metrics=['mae'])

callback = [
    ReduceLROnPlateau(monitor='val_loss', factor=0.5,
                      patience=int(patien/1.5), min_lr=1e-6, mode='min'),
    EarlyStopping(patience=patien, monitor='val_loss', verbose=1),
    ModelCheckpoint(saveP, monitor='val_mean_absolute_error',
                    verbose=1, save_best_only=True, save_weights_only=True),
    TensorBoard(log_dir=logD),
    history,
]
model.fit([train_X, train_sp], train_Y,
          epochs=epoch,
          batch_size=batchSize,
          shuffle=True,
          validation_data=([valid_X, valid_sp], valid_Y),
          callbacks=callback,
          class_weight='auto'
          )
model.save(saveP+"_all.h5")
```

圖 27　2nd fit。

最後在 fit 的部分我們用了第一次訓練完的 weight 做一些調整進行第二次訓練：

predict.py

```
tokenizer = pickle.load(open('output/tokenizer.pkl', 'rb'))
speaker = pickle.load(open('output/token_sp.pkl', 'rb'))
test_list = testing['Stemmed_Statement'].astype(str).tolist()
test_speaker = testing['Speaker'].astype(str).tolist()
test_id = testing.ID.tolist()
test_data = tokenizer.texts_to_sequences(test_list)
test_sp = speaker.texts_to_sequences(test_speaker)
test_score = testing.Trust_Score.tolist()
```

圖 28　tokenize。

利用 training set 的 tokenizer 一樣對 testing set 做 tokenize，並且一樣做長度切割，這部分的程式碼就不再重複：

```
# predict 1

model = load_model('model/Reg_keras.h5_all.h5')
model.load_weights('model/Reg_keras.h5')
loss, acc = model.evaluate([test_X, test_SP], test_SC)
print(loss, acc)
ans1 = model.predict([test_X, test_SP])

# predict 2

model2 = load_model('model/Reg_keras2.h5_all.h5')
model2.load_weights('model/Reg_keras2.h5')
loss, acc = model2.evaluate([test_X, test_SP], test_SC)
print(loss, acc)
ans2 = model2.predict([test_X, test_SP])
```

圖 29 predict。

接著分別用兩個訓練完的 model 進行預測：

```
ans = (ans1 + ans2)/2
ans = np.squeeze(ans)
ans1 = np.squeeze(ans1)
ans2 = np.squeeze(ans2)

ans_df = pd.DataFrame(
    data={'id': list(test_ID), 'score_avg': list(ans),
    'score1': list(ans1), 'score2': list(ans2), 'Trust_Score': list(test_SC)})
ans_df

ans_df = ans_df.groupby('id').mean().reset_index()
ans_df['label_avg'] = ans_df['score_avg'].apply(transfer_bin)
ans_df['label1'] = ans_df['score1'].apply(transfer_bin)
ans_df['label2'] = ans_df['score2'].apply(transfer_bin)


def accuracy(x, y):
    count = 0
    for i in range(len(x)):
        if x[i] == y[i]:
            count += 1
    length = len(x)
    return count/length

print(accuracy(ans_df['Trust_Score'], ans_df['label_avg']))
print(accuracy(ans_df['Trust_Score'], ans_df['label1']))
print(accuracy(ans_df['Trust_Score'], ans_df['label2']))

ans_df.to_csv('testing.csv', index=False)
```

圖 30 計算正確率。

將預測出的兩種答案進行平均，轉換成 binary，並分別與原本的 Trust_Score 比較，來計算正確率。

4.3 成果

Classification：

Out[44]:

	counting(TrainingData)	Tf(TrainingData)	TF-IDF(TrainingData)
RandomForest	0.732901	0.80746	0.759981
MultinomialNB	0.62168	0.493757	0.494672
SVC	0.529625	0.49157	0.49157

圖 31 Training Data 的正確率。

Out[46]:

	counting	TF	TF-IDF
RandomForest	0.595037	0.599173	0.581836
MultinomialNB	0.590743	0.494672	0.495626
SVC	0.536027	0.494831	0.494831

圖 32 Testing Data 的正確率。

	counting(Training)	counting
RandomForest	0.732901	0.595037
MultinomialNB	0.62168	0.590743
SVC	0.529625	0.536027

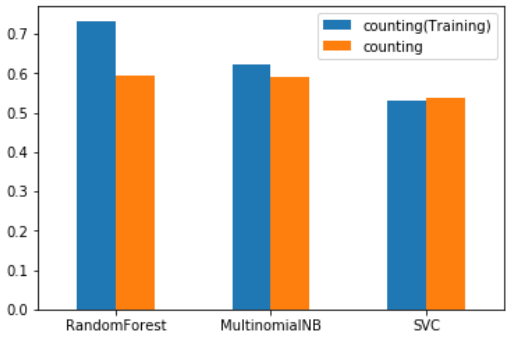


圖 33 向量值為"文字在文章中的出現次數"的正確率。

長條圖為視覺化後的結果：

Out[44]:

	counting(TrainingData)	Tf(TrainingData)	TF-IDF(TrainingData)
RandomForest	0.732901	0.80746	0.759981
MultinomialNB	0.62168	0.493757	0.494672
SVC	0.529625	0.49157	0.49157

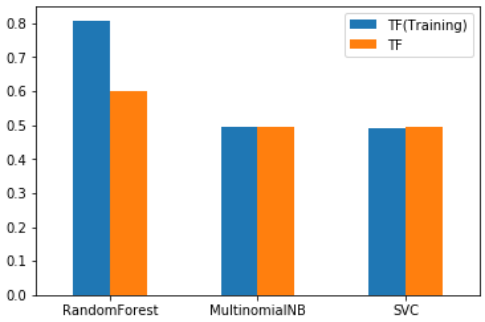


圖 34 向量值為"TF 值"的正確率。

	TF-IDF(Training)	TF-IDF
RandomForest	0.759981	0.581836
MultinomialNB	0.494672	0.495626
SVC	0.49157	0.494831

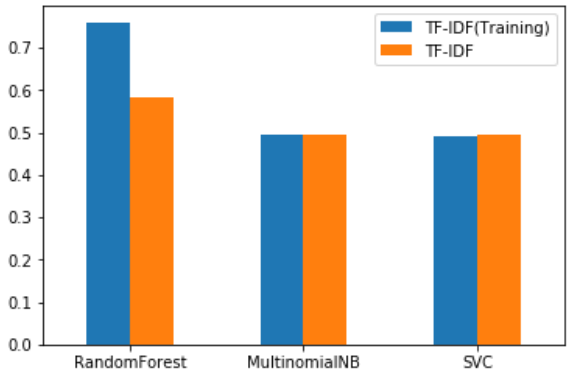


圖 35 向量值為”TF-IDF”的結果。

Regression：

id	score_avg	score1	score2	Trust_Score	label_avg	label1	label2
10040.json	0	0	0	0	0	0	0
10041.json	0	0	0	0	0	0	0
1005.json	0.109572	0	0.219143	0	0	0	0
10055.json	0	0	0	1	0	0	0
10061.json	0.141079	0	0.282158	0	0	0	0
10064.json	0.299881	0.065228	0.534533	0	0	0	1
10069.json	0.215876	0	0.431753	0	0	0	0
10085.json	0	0	0	0	0	0	0

圖 36 輸出結果。

label 為 score 轉換成 binary 後的結果

```
1382/1382 [==============================] - 4s 3ms/step
0.3367742345137346 0.35814208378301
1382/1382 [==============================] - 2s 2ms/step
0.3298952093647116 0.38071579601502453
```

圖 37 單輸入模型的 loss 值。

```
1382/1382 [==============================] - 3s 2ms/step
0.30836228487067 0.35949097030867544
1382/1382 [==============================] - 2s 2ms/step
0.28712249773402204 0.36448048152212814
```

圖 38 雙輸入模型的 loss 值。

可以看到比起單輸入有降低：

```
score_avg: 0.5651144435674822
score1:    0.5682715074980268
score2:    0.55327545382794
```

圖 39 單輸入模型的正確率。

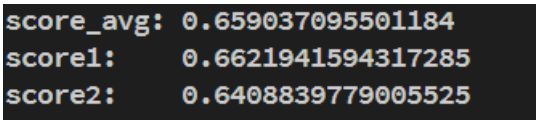


圖 40 雙輸入模型的正確率。

第五章　結論

經過研究後發現,在分類模型上的分類準確率非常依賴資料的來源以及平均性，在加入額外資料、以及平衡每種資料的數量後，正確率均有顯著上升，而利用四種方法抽取特徵值的作法也確實能夠有效的對文章作分類。分類模型和輸入向量的關係中，我們發現不管哪一種向量，Random Forest 都有 Overfitting 的傾向，而 SVM 和 Naïve Bayes 雖正確率較低，但沒有 Overfitting。可以看出非線性的分類器比較容易產生 overfitting，至於輸入向量跟正確率關係中，除 Random Forest 外，只計算出現次數的正確率竟然比 TF 跟 TF-IDF 都還來得高，我們認為可能的原因如下:

(1) 訓練和測試的資料分配不均:因為是隨機抽取的，所以可能並不是每個 type 都按照比例抽取出來。

(2) TF 和 TF-IDF 數值較為集中：雖然 TF 和 TF-IDF 比較能夠反映文章跟單字的真實情況，但因為範圍較為集中，所以較難分辨。

迴歸模型的建構與訓練則較為複雜，我們測試了多種不同的模型，其中有相當多的參數需要調整與重新訓練，我們找到的最好成果是利用多個輸入後經過卷積層，最後配合 LSTM 模型的訓練法。迴歸模型相較於分類模型的輸出結果則更為簡單易懂，輸出為 0 到 1 的數字，越大的數字代表越高的可信度，這可以非常直觀的給新聞讀者一個可信度的參考。從這次的研究我們發現資料的重要性，不管是使用何種分析的方法，資料的完整性以及種類的平均性都會很大部分的決定我們能做到的訓練成果，而各種不同的前處理方法也可以有效的幫助我們獲得更好的結果。而正因為資料的得來不易，如果能夠善加利用，就有辦法發展出良好的預測及分析系統，這也是我們這次專題的主要目標，透過分析大量的新聞及相關資料，能夠建立一個能夠幫助判斷新聞真假的參考。

參考文獻

[1] 文字分類——特徵選擇概述 https://www.itread01.com/content/1541842872.html.

[2]「卡方檢定-獨立性檢定(The Chi-Squared Test of Independence)-統計說明與SPSS操作」網站中的「三、假說檢定」

https://www.yongxi-stat.com/chi-squared-test-of-independence/

[3] Liklihood-Ratio 和 Log Liklihood Ratio 的解釋參考自

https://wangcc.me/LSHTMlearningnote/likelihood-definition.html

https://wangcc.me/LSHTMlearningnote/llr.html

[4] Mutual information - Stanford NLP Group

https://nlp.stanford.edu/IR-book/html/htmledition/mutual-information-1.html

[5] 能被電腦理解的文字 NLP（一） — Word Embedding

https://medium.com/life-of-small-data-engineer/%E8%83%BD%E8%A2%AB%E9%9B%BB%E8%85%A6%E7%90%86%E8%A7%A3%E7%9A%84%E6%96%87%E5%AD%97-nlp-%E4%B8%80-word-embedding-4146267019cb

[6] 卷積神經網絡介紹(Convolutional Neural Network)

https://medium.com/jameslearningnote/%E8%B3%87%E6%96%99%E5%88%86%E6%9E%90-%E6%A9%9F%E5%99%A8%E5%AD%B8%E7%BF%92-%E7%AC%AC5-1%E8%AC%9B-%E5%8D%B7%E7%A9%8D%E7%A5%9E%E7%B6%93%E7%B6%B2%E7%B5%A1%E4%BB%8B%E7%B4%B9-convolutional-neural-network-4f8249d65d4f

[7] Understanding LSTM Networks:

https://colah.github.io/posts/2015-08-Understanding-LSTMs/?fbclid=IwAR0RGeSseIAIfYC9WAc9ffd6Qjumo1Bi3O1roud-1dEOFkhI4WMVJahaoH4

[8] William Yang Wang, ""Liar, liar pants on fire": A new benchmark dataset for fake news detection," in Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics, vol. 2, pp. 422-426, Vancouver, Canada, 2017.

[9] Fake News, Build a system to identify unreliable news articles

https://www.kaggle.com/c/fake-news/data

[10] Getting Real about Fake News

https://www.kaggle.com/mrisdal/fake-news

[11] scikit-learn的Multinomial Naïve Bayse , SVM 和 RandomForest:

https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html

https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

[12] Keras Documentation 中的Multi-input and multi-output models, preprocessing:

https://keras.io/getting-started/functional-api-guide/

https://keras.io/preprocessing/text/