PTT推噓分析

機器學習導論期末專題

105061225 周宣呈 **105061254** 林士平

(註：此code需要在code資料夾下才能執行)

一、專題目標

針對不同的版，分析文章推噓數和特定關鍵字詞之間的關係。最終目標是針對不同版各訓練出一個模型，將文章丟入便可以推測它的推噓數區間。

二、專題流程

In [1]:
```python
from IPython.display import Image
Image(filename='image/專題流程.jpg', width=600)
```

Out[1]:

三、專題內容 -- 網路爬蟲介紹

1. 利用**request**得到網頁資料，再利用**beautiful soup**從**html**格式擷取我們需要的資訊。

2. 我們找了**ptt**較熱門的**11**個版，而且沒有封鎖推噓功能。

3. 由於**ptt**上有些版是**18+**的，所以爬蟲程式中需要額外做自動通過**18+**的確認動作。

4. 我們每個版的**data**做**200**頁，非**18+**的版需要**40**分鐘、**18+**的版需要**90**分鐘。

5. 得到了**11**個**database**，文章的推文數-噓文數為**label**，文章內文去除引用及推噓文內容為**feature**。

下圖為**database**基本資料與對應的**code**：

In [11]:
```python
from IPython.display import Image
Image(filename='image/專題資料庫簡介.jpg', width=1000)
```

Out[11]:

版	文章量(篇)	對應的.csv檔	對應的程式檔
媽寶版(BabyMother board)	3958	babymother_max.csv	train_babymother_new.ipynb
八卦版(Gossiping board)	3541	gossiping_max.csv	train_gossiping_new.ipynb
政黑板(HatePolitics board)	3774	hatepolitics_max.csv	train_hatepolitics_new.ipynb
韓星版(KoreaStar board)	3983	koreastar_max.csv	train_koreastar_new.ipynb
省錢版(lifeismoney board)	3969	lifeismoney_max.csv	train_lifeismoney_new.ipynb
婚姻版(marriage board)	3905	marriage_max.csv	train_marriage_new.ipynb
行動通訊/手機(mobilecomm board)	3949	mobilecomm_max.csv	train_mobilecomm_new.ipynb
電影版(movie board)	3970	movie_max.csv	train_movie_new.ipynb
西斯版(sex board)	3945	sex_max.csv	train_sex_new.ipynb
股版(stock board)	3962	stock_max.csv	train_stock_new.ipynb
女版(womantalk board)	3967	womantalk_max.csv	train_womantalk_new.ipynb

(註：網路爬蟲程式檔見code/Web Crawler.ipynb；database檔案均在code資料夾中)

四、專題內容 -- 預處理、萃取feature vector與label encoding

(註：此報告將以train_gossiping_new.ipynb為例子，其餘的程式檔案見code資料夾)

In [33]:
```python
# read database
import pandas as pd

f = open('gossiping_max.csv')
df= pd.read_csv(f, header = None)
df = df.fillna('0')

X = df.iloc[:, 2].values
y = df.iloc[:, 1].values
```

文章處理的步驟如下圖：

In [12]:
```python
from IPython.display import Image
Image(filename='image/專題預處理步驟.jpg', width=600)
```

Out[12]:

label encoding的方式如下圖：

In [17]:
```python
from IPython.display import Image
Image(filename='image/label_encoding.jpg', width=400)
```

Out[17]:

(一) (推文數 - 噓文數) < 0 ： 0。
(二) (推文數 - 噓文數)介於0-10 ： 1。
(三) (推文數 - 噓文數)介於11-50 ： 2。
(四) (推文數 - 噓文數) >= 51 ： 3。

接下來為預處理、萃取**feature vector**與**label encoding**的程式說明：

1. 拿掉中英文標點以及其他無用字元

In [34]:
```python
# preprocessing : extracting feature vector

import re
from zhon.hanzi import punctuation
import string

eng_punc = string.punctuation
def preprocessor(text):
    text = re.sub('\n', '', text)

    re_punctuation = "[{}]+".format(punctuation)  #拿掉中文標點
    text = re.sub(re_punctuation, "", text)

    re_punctuation_eng = "[{}]+".format(eng_punc) #拿掉英文標點
    text = re.sub(re_punctuation_eng, "", text)

    ptt_punc = '\n\t ※'
    re_punctuation_ptt = "[{}]+".format(ptt_punc) #拿掉其他無用字元
    text = re.sub(re_punctuation_ptt, "", text)

    text = re.sub('[0-9A-Za-z]+','',text)

    return text

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

2. 中文分詞並拿掉中文**stop word**

In [35]:
```python
# preprocessing : 移除stop word
import jieba.analyse
import jieba

jieba.set_dictionary("jieba_dict/dict.txt.big.txt")
tags = []

for i in range(0, len(X)):
    tags.append(jieba.analyse.extract_tags(X[i], topK=50, withWeight=True))
stopWords = []
with open('stop_word/stop_word.txt', 'r', encoding='UTF-8') as file:
    for data in file.readlines():
        data = data.strip()
        stopWords.append(data)

X_cut = X
X_cut_stop = X

for i in range(0, len(X)):
    X_cut[i] = jieba.cut(X[i], cut_all=False)
    X_cut_stop[i] = list(filter(lambda a: a not in stopWords and a != '\n', X_cut[i]))
```

```
Building prefix dict from C:\Users\HP\Desktop\python_code\105061225_105061254\code\jieba_dict\dict.txt.big.txt ...
Loading model from cache C:\Users\HP\AppData\Local\Temp\jieba.u235963241ba676e9e2b010a6a19de547.cache
Loading model cost 7.273 seconds.
Prefix dict has been built succesfully.
```

中文自然語言處理和英文自然語言處理最大的不同在於英文詞和詞之間會有空格但中文卻沒有，所以就有許多相對應演算法的誕生。

在此我們使用**jieba**(結巴)分詞庫，**jieba**中文斷詞所使用的演算法是基於 **Trie Tree** 結構去生成句子中中文字所有可能成詞的情況，然後使用動態規劃算法來找出最大機率的路徑，這個路徑就是基於詞頻的最大斷詞結果。

對於辨識新詞（字典詞庫中不存在的詞）則使用了 **HMM** 模型（**Hidden Markov Model**）及 **Viterbi** 算法來辨識出來。

3. label encoding

In [36]:
```python
# preprocessing : labeling y_train
# encoding of 推噓 :
import math

for i in range(0, len(y)):
    if y[i] == '爆':
        y[i] = 3
    elif y[i][0] == 'X':
        y[i] = 0
    else:
        y[i] = int(y[i])
        if y[i] == 0:
            y[i] = 1
        elif y[i] >= 1 and y[i] <= 10:
            y[i] = 1
        elif y[i] >= 11 and y[i] <= 50:
            y[i] = 2
        elif y[i] >= 51 and y[i] <= 99:
            y[i] = 3
'''
for i in range(0, len(y)):
    if y[i] == '爆':
        y[i] = 1
    elif y[i][0] == 'X':
        y[i] = 0
    else:
        y[i] = int(y[i])
        if y[i] == 0:
            y[i] = 0
        elif y[i] >= 1 and y[i] <= 10:
            y[i] = 0
        elif y[i] >= 11 and y[i] <= 50:
            y[i] = 1
        elif y[i] >= 51 and y[i] <= 99:
            y[i] = 1
'''
```

Out[36]:
```
"    \nfor i in range(0, len(y)):\n    if y[i] == '爆':\n        y[i] = 1\n    elif y[i][0] == 'X':\n        y[i] = 0\n    else:\n        y[i] = int(y[i])\n        if y[i] == 0:\n            y[i] = 0\n        elif y[i] >= 1 and y[i] <= 10:\n            y[i] = 0\n        elif y[i] >= 11 and y[i] <= 50:\n            y[i] = 1\n        elif y[i] >= 51 and y[i] <= 99:\n            y[i] = 1\n"
```

4. 將文章轉成**CountVectorizer**可使用的型式

In [37]:
```python
# Preprocessing : 轉換成CountVectorizer可使用
import nltk
import numpy as np

X_cut_NTLK = []

for i in range(0, len(X)):
    article = ''
    for word in X_cut_stop[i]:
        article = article + word
        article = article + ' '
    X_cut_NTLK = np.hstack((X_cut_NTLK, article))
```

經由**jieba**分詞後的文章必須經由以上的轉換才能使用**CountVectorizer**和**TfidfTransformer**做後續的處理

5. train-test split(70%-30%)

In [38]:
```python
from sklearn.model_selection import train_test_split

y = y.astype('int')

X_train, X_test, y_train, y_test = train_test_split(
    X_cut_NTLK, y, test_size=0.3, random_state=1, stratify=y)

print('Labels counts in y:', np.bincount(y))
print('Labels counts in y_train:', np.bincount(y_train))
print('Labels counts in y_test:', np.bincount(y_test))
```

```
Labels counts in y: [ 131 2568  560  282]
Labels counts in y_train: [  92 1797  392  197]
Labels counts in y_test: [ 39 771 168  85]
```

每一個**label**的**sample**數量不平衡，所以要對**training set**做**upsample**。

6. 使用**bag of word**和**tf-idf**得到**feature vector**

In [39]:
```python
# Preprocessing : Bag of Word
# Preprocessing : Term frequency - inverse document frequency (tf - idf) using NTLK
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import CountVectorizer

count = CountVectorizer(max_features=1000)
bag_train = count.fit_transform(X_train)
bag_test = count.transform(X_test)

tfidf = TfidfTransformer(use_idf = True,
                         norm = 'l2',
                         smooth_idf = True)
X_train = tfidf.fit_transform(bag_train)
X_test = tfidf.transform(bag_test)
print(X_train.shape[1])
```

```
1000
```

In [40]:
```python
print(y.shape[0])
print(X_train.shape[0])
print(X_test.shape[0])
```

```
3541
2478
1063
```

7. resample使**training set**中每個**label**的**sample**數量相同

In [41]:
```python
print(len(X_train.toarray()[:,0]))
print(len(X_test.toarray()[:,0]))

X_all = np.concatenate((X_train.toarray(),X_test.toarray()),axis=0)
y_all = np.concatenate((y_train,y_test))

print(len(X_all[:,0]))
print(len(y_all))

group = np.zeros(len(y_all), dtype=np.int)
group[0:len(X_train.toarray()[:,0])] = group[0:len(X_train.toarray()[:,0])]+1
df_re = pd.DataFrame(X_all)
df_re['label'] = y_all
df_re['group'] = group
```

```
2478
1063
3541
3541
```

In [44]:
```python
from sklearn.utils import resample

df_0 = df_re[df_re.group==1][df_re.label==0]
df_1 = df_re[df_re.group==1][df_re.label==1]
df_2 = df_re[df_re.group==1][df_re.label==2]
df_3 = df_re[df_re.group==1][df_re.label==3]

df_0 = resample(df_0, replace=True, n_samples=np.bincount(y_train).max(), random_state=123)
df_1 = resample(df_1, replace=True, n_samples=np.bincount(y_train).max(), random_state=123)
df_2 = resample(df_2, replace=True, n_samples=np.bincount(y_train).max(), random_state=123)
df_3 = resample(df_3, replace=True, n_samples=np.bincount(y_train).max(), random_state=123)

df_upsampled = pd.concat([df_0, df_1,df_2, df_3])

X_train_up = np.asarray(df_upsampled.iloc[:,0:1000])
y_train_up = np.asarray(df_upsampled.iloc[:,1000])
X_test_up = np.asarray(df_re[df_re.group==0].iloc[:,0:1000])
y_test_up = np.asarray(df_re[df_re.group==0].iloc[:,1000])

print('train label 分布:',np.bincount(y_train_up))
print('test label 分布:',np.bincount(y_test_up))
print('評分標準(只猜一群):',len(y_test_up[y_test_up == 1])/len(y_test_up))
```

```
c:\users\hp\anaconda3\envs\tensorflow\lib\site-packages\ipykernel_launcher.py:3: UserWarning: Boolean Series key will be reindexed to match DataFrame index.
  This is separate from the ipykernel package so we can avoid doing imports until
c:\users\hp\anaconda3\envs\tensorflow\lib\site-packages\ipykernel_launcher.py:4: UserWarning: Boolean Series key will be reindexed to match DataFrame index.
  after removing the cwd from sys.path.
c:\users\hp\anaconda3\envs\tensorflow\lib\site-packages\ipykernel_launcher.py:5: UserWarning: Boolean Series key will be reindexed to match DataFrame index.
  """
c:\users\hp\anaconda3\envs\tensorflow\lib\site-packages\ipykernel_launcher.py:6: UserWarning: Boolean Series key will be reindexed to match DataFrame index.
```

```
train label 分布: [1797 1797 1797 1797]
test label 分布: [ 39 771 168  85]
評分標準(只猜一群): 0.7253057384760113
```

五、專題內容 -- 使用一系列機器學習演算法

1. 我們以**logistic regression**作為**baseline**

In [13]:
```python
# logistic regression using Scikit-learn
from sklearn.linear_model import LogisticRegression

lr = LogisticRegression(C = 10, random_state = 1)
lr.fit(X_train_up, y_train_up)

# Prediction and Performance Measurement
y_pred = lr.predict(X_test_up)
print('Misclassified samples: %d' % (y_test_up != y_pred).sum())
from sklearn.metrics import accuracy_score
print('Accuracy: %.2f' %accuracy_score(y_test_up, y_pred))
print('Accuracy: %.2f' %lr.score(X_test_up, y_test_up))

print('\ntest --- train\n')

# Prediction and Performance Measurement
y_pred2 = lr.predict(X_train_up)
print('Misclassified samples: %d' % (y_train_up != y_pred2).sum())
from sklearn.metrics import accuracy_score
print('Accuracy: %.2f' %accuracy_score(y_train_up, y_pred2))
print('Accuracy: %.2f' %lr.score(X_train_up, y_train_up))
```

```
c:\users\hp\anaconda3\envs\tensorflow\lib\site-packages\sklearn\linear_model\logistic.py:432: FutureWarning: Default solver will be changed to 'lbfgs' in 0.22. Specify a solver to silence this warning.
  FutureWarning)
c:\users\hp\anaconda3\envs\tensorflow\lib\site-packages\sklearn\linear_model\logistic.py:469: FutureWarning: Default multi_class will be changed to 'auto' in 0.22. Specify the multi_class option to silence this warning.
  "this warning.", FutureWarning)
```

```
Misclassified samples: 512
Accuracy: 0.52
Accuracy: 0.52

test --- train

Misclassified samples: 312
Accuracy: 0.96
Accuracy: 0.96
```

In [14]:
```python
print(np.bincount(y_pred))
```

```
[ 69 584 272 138]
```

In [15]:
```python
print(np.bincount(y_test))
```

```
[ 39 771 168  85]
```

In [16]:
```python
j = y_pred[y_pred != 1]
k = y_test_up[y_pred != 1]

print(sum(j==k)/(len(j)+1))   ## 小群的正確率
```

```
0.1875
```

2. 接著試試**SVM**

In [17]:
```python
# training with SVM(RBF)
from sklearn.svm import SVC

svm = SVC(kernel = 'rbf', C = 100, gamma = 1000, random_state = 1)
svm.fit(X_train_up, y_train_up)

# Prediction and Performance Measurement
y_pred = svm.predict(X_test_up)
print('Misclassified samples: %d' % (y_test_up != y_pred).sum())
from sklearn.metrics import accuracy_score
print('Accuracy: %.2f' %accuracy_score(y_test_up, y_pred))
print('Accuracy: %.2f' %svm.score(X_test_up, y_test_up))

print('\ntest --- train\n')

# Prediction and Performance Measurement
y_pred2 = svm.predict(X_train_up)
print('Misclassified samples: %d' % (y_train_up != y_pred2).sum())
from sklearn.metrics import accuracy_score
print('Accuracy: %.2f' %accuracy_score(y_train_up, y_pred2))
print('Accuracy: %.2f' %svm.score(X_train_up, y_train_up))
```

```
Misclassified samples: 293
Accuracy: 0.72
Accuracy: 0.72

test --- train

Misclassified samples: 1
Accuracy: 1.00
Accuracy: 1.00
```

In [18]:
```python
print(np.bincount(y_pred))
```

```
[   0 1062    0    1]
```

In [19]:
```python
print(np.bincount(y_test))
```

```
[ 39 771 168  85]
```

In [20]:
```python
j = y_pred[y_pred != 1]
k = y_test_up[y_pred != 1]

print(sum(j==k)/(len(j)+1))   ## 小群的正確率
```

```
0.0
```

SVM的問題在明顯**overfitting**，而且幾乎只猜最大群，固然**Accuracy**比**baseline**高，但這是因為它都猜最大群。

於是我們把目標放在提升小群的預測比例和整體準確度，所以我們想到**bagging**和**boosting**

3. bagging

In [21]:
```python
# training with forest
from sklearn.ensemble import RandomForestClassifier

forest = RandomForestClassifier(criterion='entropy', n_estimators=500, random_state=1, n_jobs=-1)
forest.fit(X_train_up, y_train_up)

# Prediction and Performance Measurement
y_pred = forest.predict(X_test_up)
print('Misclassified samples: %d' % (y_test_up != y_pred).sum())
from sklearn.metrics import accuracy_score
print('Accuracy: %.2f' %accuracy_score(y_test_up, y_pred))
print('Accuracy: %.2f' %forest.score(X_test_up, y_test_up))

print('\ntest --- train\n')

# Prediction and Performance Measurement
y_pred2 = forest.predict(X_train_up)
print('Misclassified samples: %d' % (y_train_up != y_pred2).sum())
from sklearn.metrics import accuracy_score
print('Accuracy: %.2f' %accuracy_score(y_train_up, y_pred2))
print('Accuracy: %.2f' %forest.score(X_train_up, y_train_up))
```

```
Misclassified samples: 325
Accuracy: 0.69
Accuracy: 0.69

test --- train

Misclassified samples: 1
Accuracy: 1.00
Accuracy: 1.00
```

In [22]:
```python
print(np.bincount(y_pred))
```

```
[ 34 905  63  61]
```

In [23]:
```python
print(np.bincount(y_test))
```

```
[ 39 771 168  85]
```

In [24]:
```python
j = y_pred[y_pred != 1]
k = y_test_up[y_pred != 1]

print(sum(j==k)/(len(j)+1))   ## 小群的正確率
```

```
0.2641509433962264
```

bagging使用**randomforest**，可以觀察到相對**SVM**大幅解決只猜最大群的問題，且**accuracy**比**logistic regression**高。

4. boosting

In [25]:
```python
from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier

tree = DecisionTreeClassifier(max_depth=3, criterion='entropy', random_state=1)
ada = AdaBoostClassifier(base_estimator=tree, n_estimators=30, learning_rate=0.1, random_state=1)
ada = ada.fit(X_train_up,y_train_up)

# Prediction and Performance Measurement
y_pred = ada.predict(X_test_up)
print('Misclassified samples: %d' % (y_test_up != y_pred).sum())
from sklearn.metrics import accuracy_score
print('Accuracy: %.2f' %accuracy_score(y_test_up, y_pred))
print('Accuracy: %.2f' %ada.score(X_test_up, y_test_up))

print('\ntest --- train\n')

# Prediction and Performance Measurement
y_pred2 = ada.predict(X_train_up)
print('Misclassified samples: %d' % (y_train_up != y_pred2).sum())
from sklearn.metrics import accuracy_score
print('Accuracy: %.2f' %accuracy_score(y_train_up, y_pred2))
print('Accuracy: %.2f' %ada.score(X_train_up, y_train_up))
```

```
Misclassified samples: 412
Accuracy: 0.61
Accuracy: 0.61

test --- train

Misclassified samples: 2871
Accuracy: 0.60
Accuracy: 0.60
```

In [26]:
```python
print(np.bincount(y_pred))
```

```
[ 25 788  85 165]
```

In [27]:
```python
print(np.bincount(y_test))
```

```
[ 39 771 168  85]
```

In [28]:
```python
j = y_pred[y_pred != 1]
k = y_test_up[y_pred != 1]

print(sum(j==k)/(len(j)+1))   ## 小群的正確率
```

```
0.15942028985507245
```

boosting使用**adaboost**，與前者比較稍微犧牲準確度，但是針對小群的預測也更多，這是由於**boosting**的設計本來就是針對特例情況。但是也有缺點，訓練某些板時收斂速度極慢，至少要**10**個小時可能才符合需求。

5. Multilayer perceptron(MLP)

In [29]:
```python
from sklearn.neural_network import MLPClassifier

mlp = MLPClassifier(random_state=1,hidden_layer_sizes=(100, ))
mlp.fit(X_train_up, y_train_up)

# Prediction and Performance Measurement
y_pred = mlp.predict(X_test_up)
print('Misclassified samples: %d' % (y_test_up != y_pred).sum())
from sklearn.metrics import accuracy_score
print('Accuracy: %.2f' %accuracy_score(y_test_up, y_pred))
print('Accuracy: %.2f' %mlp.score(X_test_up, y_test_up))

print('\ntest --- train\n')

# Prediction and Performance Measurement
y_pred2 = mlp.predict(X_train_up)
print('Misclassified samples: %d' % (y_train_up != y_pred2).sum())
from sklearn.metrics import accuracy_score
print('Accuracy: %.2f' %accuracy_score(y_train_up, y_pred2))
print('Accuracy: %.2f' %mlp.score(X_train_up, y_train_up))
```

```
Misclassified samples: 481
Accuracy: 0.55
Accuracy: 0.55

test --- train

Misclassified samples: 1
Accuracy: 1.00
Accuracy: 1.00
```

In [30]:
```python
print(np.bincount(y_pred))
```

```
[ 42 661 243 117]
```

In [31]:
```python
print(np.bincount(y_test))
```

```
[ 39 771 168  85]
```

In [32]:
```python
j = y_pred[y_pred != 1]
k = y_test_up[y_pred != 1]

print(sum(j==k)/(len(j)+1))   ## 小群的正確率
```

```
0.18610421836228289
```

最後我們嘗試深度學習中的**MLP**。用最少的時間完成接近**adaboost**的**accuracy**且兼顧小群比例。

六、結果整理

1. 首先是各個版的文字雲分析結果，可以看到不同的版之間關鍵字確實有明顯不同，不過也可以觀察到某些版之間有重疊的關鍵字

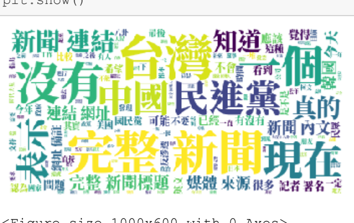
In [32]:
```python
# (Not necessary) : word cloud for training set
import numpy as np
import matplotlib.pyplot as plt
from wordcloud import WordCloud

cloud = ''

for i in range(0, len(X)):
    for word in X_cut_stop[i]:
        cloud = cloud + word
        cloud = cloud + ' '
```

In [33]:
```python
# 產生文字雲
wc = WordCloud(font_path="word/NotoSerifCJKtc-Black.otf", #設置字體
               background_color="white", #背景顏色
               max_words = 2000)         #文字雲顯示最大詞數
wc.generate(cloud)

# 視覺化呈現
plt.imshow(wc)
plt.axis("off")
fig = plt.figure(figsize=(10,6), dpi = 100)
plt.show()
```



```
<Figure size 1000x600 with 0 Axes>
```

(1) 媽寶版(BabyMother board)

In [7]:
```python
Image(filename='image/baby_mother_wordcloud.png', width=600)
```

Out[7]:

(2) 八卦版(Gossiping board)

In [9]:
```python
Image(filename='image/gossiping_wordcloud.png', width=600)
```

Out[9]:

(3) 政黑板(HatePolitics board)

In [10]:
```python
Image(filename='image/hatepolitics_wordcloud.png', width=600)
```

Out[10]:

(4) 韓星版(KoreaStar board)

In [11]:
```python
Image(filename='image/koreastar_wordcloud.png', width=600)
```

Out[11]:

(5) 省錢版(lifeismoney board)

In [12]:
```
Image(filename='image/lifeismoney_wordcloud.png', width=600)
```
Out[12]:

(6) 婚姻版(marriage board)

In [13]:
```
Image(filename='image/marriage_wordcloud.png', width=600)
```
Out[13]:

(7) 行動通訊/手機(mobilecomm board)

In [14]:
```
Image(filename='image/mobilecomm_wordcloud.png', width=600)
```
Out[14]:

(8) 電影版(movie board)

In [15]:
```
Image(filename='image/movie_wordcloud.png', width=600)
```
Out[15]:

(9) 西斯版(sex board)

In [16]:
```
Image(filename='image/sex_wordcloud.png', width=600)
```
Out[16]:

(10) 股版(stock board)

In [17]:
```
Image(filename='image/stock_wordcloud.png', width=600)
```
Out[17]:

(11) 女版(womantalk board)

In [18]:
```
Image(filename='image/womantalk_wordcloud.png', width=600)
```
Out[18]:

2. 各個版training model結果整理

In [3]:
```
Image(filename='image/專題結果.jpg', width=1000)
```
Out[3]:

Database	Testing set label分布 (y_test)	Testing set 預測Accuracy與預測label分布(y_pred)				
		Logistic regression	SVM	Bagging	Boosting	MLP
babymother_max.csv	[3 535 545 105]	[3 492 515 178] 0.51	[0 3 1185 0] 0.46	[0 524 621 43] 0.56	[0 708 431 49] 0.53	[0 528 527 133] 0.49
gossiping_max.csv	[39 771 168 85]	[69 584 272 138] 0.52	[0 1062 0 1] 0.72	[34 905 63 61] 0.69	[25 788 85 165] 0.61	[42 661 243 117] 0.55
hatepolitics_max.csv	[13 889 203 28]	[19 666 372 76] 0.56	[0 1132 0 1] 0.78	[3 1094 32 4] 0.77	[9 891 221 12] 0.67	[10 761 319 43] 0.61
koreastar_max.csv	[102 141 415 537]	[175 210 377 433] 0.42	[2 1 1 1191] 0.45	[112 69 403 611] 0.46	[145 81 284 685] 0.45	[138 169 408 480] 0.41
lifeismoney_max.csv	[19 398 596 178]	[28 434 429 300] 0.41	[0 5 1170 16] 0.49	[0 397 630 164] 0.46	[13 53 354 771] 0.27	[18 428 486 259] 0.40
marriage_max.csv	[36 496 396 244]	[64 434 366 308] 0.40	[0 1165 0 7] 0.42	[0 589 378 205] 0.50	[22 567 381 202] 0.47	[25 442 392 313] 0.40
mobilecomm_max.csv	[5 632 487 61]	[3 529 522 131] 0.49	[0 1185 0 0] 0.53	[0 652 530 3] 0.58	[2 545 606 32] 0.51	[0 550 570 65] 0.51
movie_max.csv	[11 664 420 96]	[12 557 464 158] 0.50	[0 1191 0 0] 0.56	[0 772 396 23] 0.56	[2 339 710 140] 0.41	[5 588 460 138] 0.50
sex_max.csv	[11 570 472 131]	[12 478 404 290] 0.42	[0 1184 0 0] 0.48	[0 580 585 19] 0.47	[1 275 606 302] 0.38	[3 507 449 225] 0.43
stock_max.csv	[9 473 580 127]	[31 451 490 217] 0.47	[3 6 1158 22] 0.50	[14 514 603 58] 0.54	[3 675 460 51] 0.50	[13 473 521 182] 0.50
womantalk_max.csv	[11 844 254 82]	[11 645 360 175] 0.48	[0 1191 0 0] 0.71	[1 1151 38 1] 0.69	[7 641 216 327] 0.45	[6 701 368 116] 0.51

3. 結論

(1) 可以看得出來**SVM**有明顯**overfitting**的問題。

(2) bagging和**boosting**可以解決**svm**只預測小群的問題，而且**accuracy**普遍也比**baseline**好。

(3) 其中**bagging**使用**random forest**，效率高，但有時會有**overfitting**的問題。

(4) 其中**boosting**使用**adaboost**，由於是**boosting**所以較容易找到少數族群。

(5) NN的參數在使用預設值的狀況下就有不錯的**accuracy**，而且預測結果也兼顧小群比例，如果未來再調整參數或許會有更好的結果。

(6) 我們認為困難點在**data**來源是論壇文章，過於廣泛且具時效性，文筆也很難在字詞間判斷，而且很難定義出分布均勻且有意義的**label**適用各版。

七、未來展望

(1) 使用更多不同的深度學習演算法，例如：**CNN**、**RNN**。

(2) 使用其他特徵像是發文日期、時間、作者、分類......等等。

(3) 用不同版的**model**預測其他版的文章來看不同的版之間是否有共同特徵。例如從文字雲可以看的出來八卦版和政黑版有類似的關鍵字，或許拿八卦版的**model**來預測政黑版的文章會有不錯的結果。

(4) 使用除了**bag of word**和**tfidf**以外的方式來得到**feature vector**，使用上述兩種方式得到的**feature vector**最大的缺點是特徵矩陣為稀疏矩陣，會影響**model**的訓練時間，也可能會影響準確度。

八、參考資料

1. 斷詞

[NLTK 初學指南(一)：簡單易上手的自然語言工具箱－探索篇]

[如何使用 jieba 結巴中文分詞程式]

[以 jieba 與 gensim 探索文本主題：五月天人生無限公司歌詞分析 (I)]

2. jieba github

[original jieba github]

[Taiwan jieba github]

3. csv_read for Chinese

[pandas read_csv(中文名稱 中文路徑)]

4. 拿掉標點符號

[jieba分詞，並去除所有標點]

[zhon包]

[拿掉標點方法]

5. 文字雲

[python（wordcloud）實現中文詞雲]

6. NLTK

[NLTK 初學指南(一)：簡單易上手的自然語言工具箱－探索篇]

7. 移除中文停用自

[10-2 中文斷詞-移除停用詞]

8. 流程

[NLP入門-- 文本預處理Pre-processing]

9. 關於NLP

[進入NLP 世界的最佳橋樑：寫給所有人的自然語言處理與深度學習入門指南]

[Practical Text Classification With Python and Keras]

10. 深度學習

[sklearn.neural_network.MLPClassifier]

[Convolutional Neural Network (CNN)]

[Deep Learning:An MIT Press book / Ian Goodfellow and Yoshua Bengio and Aaron Courville]