IRTM HW2 Report

R06725035 資管碩二 陳廷易

執行環境

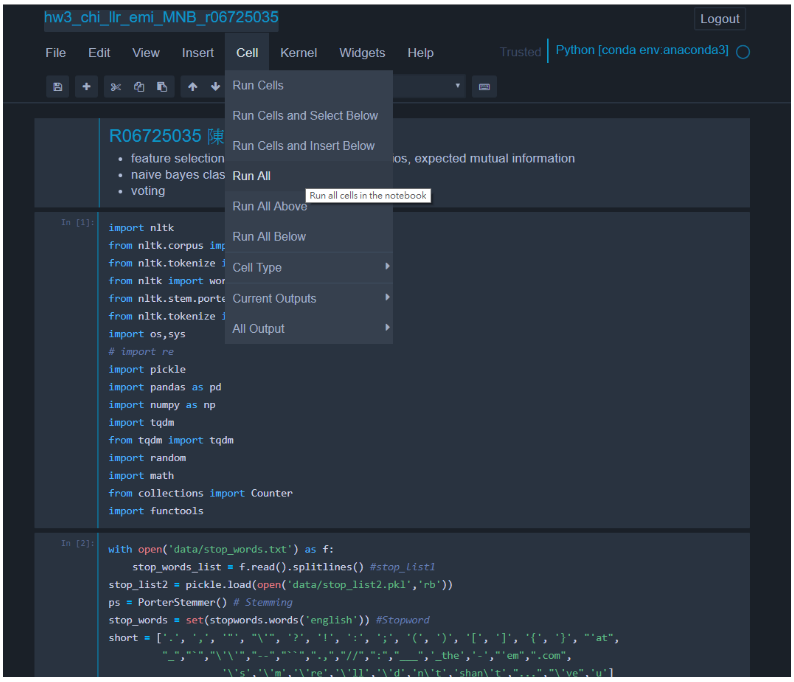
- Ubuntu 16.04
- Jupyter Notebook

程式語言

- Linux Anaconda 5 Python 3.6

執行方式

- 提供 jupyter notebook: *hw3_chi_llr_emi_MNB_r06725035.ipynb*
  - 可利用 jupyter 開啟 ipynb 以後依照下方邏輯說明來執行所需的 cells
  - 若僅需要對照 kaggle 所上傳的結果，則僅需要 Cell=>Run All 即可，同時也提供.py 供助教測試



- 需 pip install:

- ■ Python 內建既有套件: random、math、os、sys
- ■ nltk 並 download data
- ■ pickle、pandas、numpy
- ■ collections(協助計算數量)
- ■ tqdm(顯示當前進度)
- ■ functools(合併多個 csv 檔案成為 dataframe)

- 確保提供的 1095 篇 txt 預設放於 data/IRTM/目錄下
- data/目錄下需要有 stop_words.txt 及 stop_list.pkl 來製作 stop words list
- data/目錄下存放所給定的 training.txt
- data/目錄下還需要有 dictionary.txt，為作業二的 output 做為初始的 dictionary
- 九種 500 terms dictionary 的結果預設放於 output/目錄下: 500terms_df
- 各文章的 tf-idf 須放置於 data/tf-idf/目錄下: X.txt
- 各 model 所使用的 term dictionary 及各 model 的輸出結果皆放置於 output/資料夾下供參考，最後上傳 kaggle 的 csv 為綜合以上輸出:voting_rev.csv

作業邏輯說明

1. 讀入所有不需要的 stop words 以及標點符號等等，並利用 nltk 套件初始化 porterstemmer

stop words and stemming prepare:

```
with open('data/stop_words.txt') as f:
    stop_words_list = f.read().splitlines() #stop_list1
stop_list2 = pickle.load(open('data/stop_list2.pkl','rb'))
ps = PorterStemmer() # Stemming
stop_words = set(stopwords.words('english')) #Stopword
short = ['.', ',', '"', "\'", '?', '!', ':', ';', '(', ')', '[', ']', '{', '}', "'at",
         "_","`","\'\'","--","``",".,","//",":","___",'_the','-',"'em",".com",
                  '\'s','\'m','\'re','\'ll','\'d','n\'t','shan\'t',"...","\'ve",'u']
stop_words_list.extend(short)
stop_words_list.extend(stop_list2)
stop_words.update(stop_words_list) # remove it if you need punctuation
```

2. 定義前處理函式：將文件轉換為小寫、濾除 stop words、stemming、進行 tokenize、濾除數字

```python
def preprocess(texts):
    tokens = [i for i in word_tokenize(texts.lower()) if i not in stop_words]  # Tokenization.# Lowercasi
    token_result = ''
    token_result_ = ''
    for i,token in enumerate(tokens): #list2str
        token_result += ps.stem(token) + ' '
    token_result = ''.join([i for i in token_result if not i.isdigit()])
    token_result = [i for i in word_tokenize(token_result) if i not in stop_words]
    for i,token in enumerate(token_result):
        token_result_ += token + ' '
    return token_result_
```

3. 讀入作業二所輸出的 dictionary，將 training data 依照所屬類別讀入變成 dict 的資料結構，並將 training documents 進行上述二步驟的前處理

```python
dict_df = pd.read_csv('data/dictionary.txt',header=None,index_col=None,sep=' ')
terms = dict_df[1].tolist() #all terms
```

```python
with open('data/training.txt','r') as f:
    train_id = f.read().splitlines()
train_dict = {}
for trainid in train_id:
    trainid = trainid.split(' ')
    trainid = list(filter(None, trainid))
    train_dict[trainid[0]] = trainid[1:]
train_dict #class:doc_id
```

```
{'1': ['11',
  '19',
  '29',
  '113',
  '115'
```

```python
in_dir = 'data/IRTM/'
train_dict_ = {}
class_token = []
class_token_dict = {}
for c,d in train_dict.items():
    for doc in d:
        f = open('data/IRTM/'+doc+'.txt')
        texts = f.read()
        f.close()
        tokens_all = preprocess(texts)
        tokens_all = tokens_all.split(' ')
        tokens_all = list(set(filter(None,tokens_all)))
        class_token.append(tokens_all)
    class_token_dict[c]=class_token
    class_token=[]
```

4. 實作三種 Feature Selection(Likelihood Ratios、Chi-Square、Expected Mutual Information)

甲、參數初始化、計算 on topic present/absent 與 off topic present/absent 的篇數及總篇數

```python
dict_df['score'] = 0
dict_df['score_chi'] = 0
dict_df['score_emi'] = 0
# c=1
for term in tqdm(terms): #each term
    scores = []
    scores_chi = []
    scores_emi = []
    c=1
    for _ in range(len(class_token_dict)): # each class
        n11=e11=m11=0
        n10=e10=m10=0
        n01=e01=m01=0
        n00=e00=m00=0
```

```python
for k,v in class_token_dict.items():
    # print(k,c)
    if k == str(c): #ontopic
        for r in v:
            if term in r:
                n11+=1
            else:
                n10+=1
        # c+=1
    else: #off topic
        for r in v:
            if term in r:
                n01+=1
            else:
                n00+=1
N = n11+n10+n01+n00 #          c+=1
```

乙、計算各 term 於各類別的 Likelihood Ratios

```python
score = (((n11+n01)/N) ** n11) * ((1 - ((n11+n01)/N)) ** n10) * (((n11+n01)/N) ** n01) * ((1 - ((n11+n01)/
score /= ((n11/(n11+n10)) ** n11) * ((1 - (n11/(n11+n10))) ** n10) * ((n01/(n01+n00)) ** n01) * ((1 - (n01
score = -2 * math.log(score, 10) #LLR
scores.append(score)
```

```python
N)) ** n00)
/(n01+n00))) ** n00
```

丙、計算各 term 於各類別的 chi-square

```python
e11 = N * (n11+n01)/N * (n11+n10)/N #chi-squre
e10 = N * (n11+n10)/N * (n10+n00)/N
e01 = N * (n11+n01)/N * (n01+n00)/N
e00 = N * (n01+n00)/N * (n10+n00)/N
score_chi = ((n11-e11)**2)/e11 + ((n10-e10)**2)/e10 + ((n01-e01)**2)/e01 + ((n00-e00)**2)/e00
scores_chi.append(score_chi)
```

丁、計算各 term 於各類別的 EMI

```
m11 = (n11/N) * math.log(((n11/N)/((n11+n01)/N * (n11+n10)/N)),2) #EMI
m10 = n10/N * math.log((n10/N)/((n11+n10)/N * (n10+n00)/N),2)
m01 = n01/N * math.log((n01/N)/((n11+n01)/N * (n01+n00)/N),2)
m00 = n00/N * math.log((n00/N)/((n01+n00)/N * (n10+n00)/N),2)
score_emi = m11 + m10 + m01 + m00
scores_emi.append(score_emi)
```

戊、將三種分數於各類別的表現進行平均，可得各 term 的三種 feature selection 分數

```
    dict_df.loc[term,'score'] = np.mean(scores)
    dict_df.loc[term,'score_chi'] = np.mean(scores_chi)
    dict_df.loc[term,'score_emi'] = np.mean(scores_emi)
dict_df
```

```
100%|██████████| 8913/8913 [01:08<00:00, 129.96it/s]
```

	score	score_chi	score_emi
term			
abandon	0.468414	1.318717e+00	3.989761e-03
abc	0.337624	6.018518e-01	2.875750e-03

5. 將作業二所計算的各篇 tf-idf 進行加總再利用 document frequency 計算各 term 的平均 tf-idf，希望所選的五百字是可以具一定鑑別力的，並與上述的分數合併可得 dict_df3 如下

```
dict_df2 = pd.read_csv('data/dictionary.txt',header=None,index_col=None,sep=' ')
dict_df2.columns = ['id','term','freq']
dict_df2['sum'] = 0.0
```

```
tf_list = next(os.walk('../HW2/output/tf-idf/'))[2]
# df_list = [dict_df2]
for tf in tf_list:
#     print(tf)
    df2 = pd.read_csv('../HW2/output/tf-idf/'+tf,header=None,index_col=None,sep=' ',skiprows=[0])
    df2.columns = ['id','tfidf']
    df3 = pd.merge(dict_df2,df2,on='id',how='outer')
    df3.fillna(0,inplace=True)
    dict_df2['sum']+=df3['tfidf']
dict_df2['avg_tfidf'] = dict_df2['sum']/dict_df2['freq']
dict_df2 = dict_df2.drop(['freq','sum'],axis=1)
```

	id	term	avg_tfidf	score_chi	score	score_emi
0	1	abandon	0.005374	1.318717e+00	0.468414	3.989761e-03
1	2	abc	0.015857	6.018518e-01	0.337624	2.875750e-03

6. 選擇五百個以內的 term: 首先計算四種分數的平均值與標準差，取平均值加上一定倍數的標準差做為 threshold(下圖以 1.6 倍為範例)，超過該 threshold 者則對該 term 貢獻一票，取票數超過一定值以上者為最後要使用的 terms(下圖以超過兩票為例)，確保只使用 500 個以內的 term

```
threshold_tfidf = np.mean(dict_df3['avg_tfidf'])+1.6*np.std(dict_df3['avg_tfidf'])
threshold_chi = np.mean(dict_df3['score_chi'])+1.6*np.std(dict_df3['score_chi']) #1
threshold_llr = np.mean(dict_df3['score_llr'])+1.6*np.std(dict_df3['score_llr']) #1
threshold_emi = np.mean(dict_df3['score_emi'])+1.6*np.std(dict_df3['score_emi']) #1
```

```
df1 = dict_df3[dict_df3['avg_tfidf']>threshold_tfidf]
df2 = dict_df3[dict_df3['score_chi']>threshold_chi]
df3 = dict_df3[dict_df3['score_llr']>threshold_llr]
df4 = dict_df3[dict_df3['score_emi']>threshold_emi]
df_vote = dict_df3
df_vote['vote']=0
```

```
df_vote.loc[df1.id-1,'vote'] += 1
df_vote.loc[df2.id-1,'vote'] += 1
df_vote.loc[df3.id-1,'vote'] += 1
df_vote.loc[df4.id-1,'vote'] += 1
```

```
df_vote = df_vote[df_vote.vote>2] #(1,2)=>375 #
df_vote = df_vote.filter(['id','term','vote'])
df_vote
```

	id	term	vote
12	13	abidjan	3
16	17	aboard	3
31	32	absolut	3

7. Multinomial Naïve Bayes Classifier training：

甲、先計算各個 class 的 prior，求出各 class 在整個 training set 所佔的比例

乙、接下來要求出 condprob，做法為算出各 class 在 term dictionary 中出現次數的比例，而為了避免 0 probability 的情況，也會進行 add 1 smoothing

```python
def train_MNB(train_set=train_dict,term_list=terms_li,term_tf_mat=term_tf_mat):
    prior = np.zeros(len(train_set))
    cond_prob = np.zeros((len(train_set), len(term_list)))

    for i,docs in train_set.items(): #13 classes 1~13
        prior[int(i)-1] = len(docs)/len(train_ids) #那個類別的文章有幾個/總共的文章數目 0~12
        token_count=0
        class_tf = np.zeros(len(term_list))
        for idx,term in enumerate(term_list):
            try:
                class_tf[idx] = term_tf_mat[int(i)-1][term]  #term在class的出現次數
            except:
                token_count+=1

        class_tf = class_tf + np.ones(len(term_list)) #smoothing (可改)
        class_tf = class_tf/(sum(class_tf) +token_count) #該class總共的token數(可改)
        cond_prob[int(i)-1] = class_tf #0~12
    return prior, cond_prob
```

8. Multinomial Naïve Bayes Classifier testing：

甲、在 testing 部分會將文章進行前處理變成 token list

乙、將所有類別加入所對應的 prior 基本分

丙、若 token 是有出現在 term dictionary 中者，將所對應的 condprob 加入其類別分數中

丁、最後看該篇文章哪個類別的分數最高便為預測的結果

```python
def predict_MNB(test_id,prob=False,prior=prior,cond_prob=cond_prob,term_list=terms_li):
    f = open('data/IRTM/'+str(test_id)+'.txt')
    texts = f.read()
    f.close()
    tokens_all = preprocess(texts)
    tokens_all = tokens_all.split(' ')
    tokens_all = list(filter(None,tokens_all))

    class_scores = []
#     score = 0
    for i in range(13):
        score=0
#         print(prior[i])
        score += math.log(prior[i],10)
        for token in tokens_all:
            if token in term_list:
                score += math.log(cond_prob[i][term_list.index(token)])
        class_scores.append(score)
    if prob:
        return np.array(class_scores)
    else:
        return(np.argmax(class_scores)+1)
```

9. 反覆執行第 6 步驟，挑選不同的 threshold 與票數可以得到不同的 term dictionary 再重新訓練 Multinomial Naïve Bayes Classifier 並對 testing documents 重新預測其所屬類別，將一些比較有信心的文章重新執行 feature selection 與 MNB 訓練及預測，最後再將所有模型預測結果進行投票，票數最高者當成其最終類別上傳至 kaggle

```
do only in first time only

df1 = merged_df[(merged_df[0] == merged_df[1])&(merged_df[2]==merged_df[3])&(merged_df[1]==merged_df[2])]
df1.reset_index(inplace=True,drop=True)
df1['class'] = df1[0]
df1 = df1.filter(['id','class'])
```

```python
for i in range(13):
    df1 = df1.astype(str)
    li = df1[df1['class'] == str(i+1)]['id'].tolist()
    train_dict[str(i+1)].extend(li)
train_dict
```

```python
in_dir = './output/'
prefixed = [filename for filename in os.listdir('./output/') if filename.endswith("_sk.csv")]
df_from_each_file = [pd.read_csv(in_dir+f) for f in prefixed]
```

```python
merged_df = functools.reduce(lambda left,right: pd.merge(left,right,on='id'), df_from_each_file)
merged_df.columns = ['id',0,1,2,3,4,5,6,7,8]
merged_df
```

	id	0	1	2	3	4	5	6	7	8
0	17	2	2	2	2	2	2	2	2	2
1	18	2	2	2	2	2	2	2	2	2
2	20	2	2	2	2	2	2	2	2	2
3	21	2	2	2	2	2	2	2	2	2
4	22	2	2	2	2	2	2	2	2	9

```python
df01 = pd.DataFrame(merged_df.mode(axis=1)[0])
df02 = pd.DataFrame(merged_df['id'])
df_ans = pd.concat([df02,df01],axis=1)
df_ans = df_ans.astype('int')
df_ans.columns = ['id','Value']
df_ans.to_csv('output/voting_rev.csv',index=False)
df_ans
```

	id	Value
0	17	2
1	18	2
2	20	2