310706034 資管碩一 資料探勘研究與實務 HW2 - sentiment analysis

1.a 資料前處理

In [1]:
```
import pandas as pd
#1. 資料前處理
#a. 讀取csv檔僅保留"text"、"stars"兩個欄位，並將stars欄位內值大於等於4的轉成1，其餘轉成0，1: positive; 0: negative
df = pd.read_csv("./yelp.csv")
df = df[["stars","text"]]
df['stars'] = df['stars'].map(lambda x : 1 if x >=4 else 0)
```

1.b 去除停頓詞

In [2]:
```
#b.去除停頓詞stop words
import nltk
from sklearn.feature_extraction.text import CountVectorizer

corpus = df["text"].tolist()
nltk.download('stopwords')
nltk_stopwords = nltk.corpus.stopwords.words('english')
stop_words = nltk_stopwords

#c.文字探勘前處理，將文字轉換成向量，實作 tf-idf
vectorizer  = CountVectorizer(stop_words=stop_words, min_df=0.01)
X = vectorizer.fit_transform(corpus)
features = vectorizer.get_feature_names()

from sklearn.feature_extraction.text import TfidfTransformer
transformer = TfidfTransformer()
tfidf = transformer.fit_transform(X)
tfidf_vec = tfidf.toarray()
tfidf_df = pd.DataFrame(tfidf_vec)
tfidf_df['y'] = df['stars']
tfidf_df = tfidf_df.dropna()
tfidf_df
```

1.c 生成 tf-idf 文字向量

```
#c.文字探勘前處理，將文字轉換成向量，實作 tf-idf
vectorizer  = CountVectorizer(stop_words=stop_words, min_df=0.01)
X = vectorizer.fit_transform(corpus)
features = vectorizer.get_feature_names()

from sklearn.feature_extraction.text import TfidfTransformer
transformer = TfidfTransformer()
tfidf = transformer.fit_transform(X)
tfidf_vec = tfidf.toarray()
tfidf_df = pd.DataFrame(tfidf_vec)
tfidf_df['y'] = df['stars']
tfidf_df = tfidf_df.dropna()
tfidf_df
```

tf-idf 文字向量如下，min_df 設為 0.01，得到 1047 個字詞作為解釋變數

	0	1	2	3	4	5	6	7	8	9	...	1039	1040	1041	1042	1043	1044	1045	1046	1047	y
0	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	0.000000	0.0	...	0.0	0.0	0.000000	0.0	0.0	0.000000	0.0	0.0	0.0	1
1	0.0	0.0	0.0	0.106818	0.0	0.0	0.0	0.0	0.000000	0.0	...	0.0	0.0	0.000000	0.0	0.0	0.000000	0.0	0.0	0.0	1
2	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	0.000000	0.0	...	0.0	0.0	0.000000	0.0	0.0	0.000000	0.0	0.0	0.0	1
3	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	0.000000	0.0	...	0.0	0.0	0.000000	0.0	0.0	0.000000	0.0	0.0	0.0	1
4	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	0.000000	0.0	...	0.0	0.0	0.000000	0.0	0.0	0.000000	0.0	0.0	0.0	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
9995	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	0.000000	0.0	...	0.0	0.0	0.000000	0.0	0.0	0.000000	0.0	0.0	0.0	0
9996	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	0.000000	0.0	...	0.0	0.0	0.000000	0.0	0.0	0.133587	0.0	0.0	0.0	1
9997	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	0.000000	0.0	...	0.0	0.0	0.072867	0.0	0.0	0.000000	0.0	0.0	0.0	1
9998	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	0.000000	0.0	...	0.0	0.0	0.000000	0.0	0.0	0.000000	0.0	0.0	0.0	0
9999	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	0.176187	0.0	...	0.0	0.0	0.000000	0.0	0.0	0.000000	0.0	0.0	0.0	1

10000 rows × 1049 columns

1.c 生成 Word2vec 文字向量

In [6]:
```
#c.文字探勘前處理，將文字轉換成向量，實作 word2vec
#將corpus分割並儲存到text_seg_list(需要一些時間)
text_seg_binary_list = X.toarray()
text_seg_list = []
for i in range(len(text_seg_binary_list)):
    temp_list = []
    for j in range(len(text_seg_binary_list[i])):
        if text_seg_binary_list[i][j]==1:
            temp_list.append(features[j])
    text_seg_list.append(temp_list)

#訓練Word2Vec
from gensim.models import Word2Vec
vector_size=250
model = Word2Vec(sentences=text_seg_list, vector_size=vector_size, epochs=10)

#word embedding
import numpy as np
word2vec_vec = []
for i in range(len(text_seg_list)):
    vector_sum = np.zeros(vector_size)
    count = 0
    for j in range(len(text_seg_list[i])):
        try:
            vector_sum = vector_sum+model.wv[text_seg_list[i][j]]
            count += 1
        except: #該字沒有vector
            pass
    vector_average = vector_sum/count
    word2vec_vec.append(vector_average.tolist())

word2vec_df = pd.DataFrame(word2vec_vec)
word2vec_df['y'] = df['stars']
word2vec_df = word2vec_df.dropna()
word2vec_df
```

Word2vec 文字向量如下，vector size 設為 250，得到 250 維的解釋變數:

Out[6]:

	0	1	2	3	4	5	6	7	8	9	...	241	242	243	244	245	246	247	248	249	y
0	-0.050049	0.113066	-0.046956	0.026392	-0.058163	-0.017296	0.069441	0.028987	-0.040140	-0.070442	...	0.095286	0.006063	0.101681	0.037004	0.036976	-0.020608	0.088101	0.019789	0.036564	1
1	0.029677	0.016582	0.053899	-0.066003	-0.116661	0.071871	0.151930	0.001461	-0.105516	-0.030631	...	0.097542	0.042107	0.140081	-0.022947	-0.030034	0.161497	-0.036331	0.009361	0.062695	1
2	0.055479	-0.101007	0.275994	-0.116992	-0.272068	0.256404	0.306160	-0.295847	-0.171191	-0.464580	...	0.473160	0.043700	0.241626	-0.002753	-0.101020	0.048830	-0.005510	-0.103201	-0.006752	1
3	-0.097416	0.056591	0.106150	-0.066595	0.010239	-0.322291	0.310473	-0.226178	0.200717	-0.000072	...	0.113818	0.010490	0.111801	-0.108279	-0.211855	-0.244971	0.249645	-0.027878	0.009122	1
4	-0.110224	0.183650	-0.113521	-0.042847	-0.032658	-0.212107	0.164896	-0.173682	0.129333	0.129139	...	0.101592	0.068200	0.133065	-0.090661	-0.185392	-0.095740	0.155627	0.076352	0.079668	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
9995	0.011231	0.192692	-0.022830	-0.064385	-0.099639	0.021208	0.117457	-0.287946	-0.022162	-0.209555	...	0.232246	-0.001770	0.308364	0.075263	-0.002502	0.000113	-0.030961	0.073323	0.024151	0
9996	-0.024310	0.133972	0.066144	-0.001713	-0.009403	-0.077672	0.066406	-0.032907	-0.007777	-0.008316	...	-0.003631	0.034871	0.086678	0.043639	0.027575	0.027116	0.039378	-0.010343	0.030947	1
9997	-0.065638	0.112801	0.008149	-0.016467	0.033962	-0.177322	0.088195	0.042192	0.039698	0.133972	...	-0.059399	0.041757	0.032257	-0.020990	-0.039216	-0.046744	0.093902	0.007946	0.061591	1
9998	-0.003420	0.063326	0.046174	0.013401	-0.051807	-0.064723	0.142993	0.118737	-0.028699	0.040442	...	-0.002855	0.002578	0.091543	0.002999	0.004950	0.110170	0.042653	-0.002051	0.058397	0
9999	-0.041704	0.103570	-0.043953	-0.084847	-0.144134	0.005421	0.130938	-0.130757	-0.049222	-0.044757	...	0.222643	0.071850	0.205540	-0.021532	-0.053645	0.047065	-0.005712	0.051598	0.073516	1

9989 rows × 251 columns

2.K-fold CV Function & random forest

In [20]:
```
#2.
#K fold cv function and random forest
import random
from sklearn import ensemble

def K_fold_CV(k, data):
    partition = []
    partition_index = []
    data_num = len(data)
    for i in range(k):
        if i != k-1:
            partition.append(data_num//k)
        else:
            partition.append(data_num//k+data_num%k)
        partition_index.append([sum(partition)-partition[i],sum(partition)-1])

    random.seed(123)
    data_df_shuffled = data.sample(frac=1).reset_index(drop=True) #data shuffle
    Accuracy = 0
    for i in range(k):
        print("目前test fold=",str(i+1),end=' ')
        test_x = data_df_shuffled.iloc[partition_index[i][0]:partition_index[i][1]+1].drop("y", axis = 1)
        test_y = data_df_shuffled.iloc[partition_index[i][0]:partition_index[i][1]+1]["y"]
        train_x = data_df_shuffled.drop(list(range(partition_index[i][0], partition_index[i][1]+1))).drop("y", axis = 1)
        train_y = data_df_shuffled.drop(list(range(partition_index[i][0], partition_index[i][1]+1)))["y"]

        forest_fit = forest.fit(train_x, train_y)
        test_y_predicted = forest.predict(test_x)

        count = 0
        for j in range(len(test_y)):
            if(test_y.iloc[j]==test_y_predicted[j]):
                count = count + 1
        print(", accuracy=",str(round((count/len(test_y)),4)))
        Accuracy = Accuracy + count/len(test_y)

    print("----------------------------------------")
    print("Average accuracy:")
    return(round(Accuracy/k,4))
```

3.預測結果

皆使用兩百棵樹作為隨機森林參數

In [22]:
```
# 建立 random forest 模型
forest = ensemble.RandomForestClassifier(n_estimators = 200, criterion="entropy")
# tfidf_df 4-fold cv
K_fold_CV(4, tfidf_df)
```

```
目前test fold= 1 , accuracy= 0.8008
目前test fold= 2 , accuracy= 0.8004
目前test fold= 3 , accuracy= 0.8064
目前test fold= 4 , accuracy= 0.792
---------------------------------------
Average accuracy:
```

Out[22]: 0.7999

In [23]:
```
# 建立 random forest 模型
forest = ensemble.RandomForestClassifier(n_estimators = 200, criterion="entropy")
# word2vec 4-fold cv
K_fold_CV(4, word2vec_df)
```

```
目前test fold= 1 , accuracy= 0.7024
目前test fold= 2 , accuracy= 0.7165
目前test fold= 3 , accuracy= 0.6884
目前test fold= 4 , accuracy= 0.7002
---------------------------------------
Average accuracy:
```

Out[23]: 0.7019

tf-idf 的平均準確率為 0.7999、Word2vec 的平均準確率為 0.7019，可能原因是 Word2vec 有一些字詞並沒有 word vector，導致資訊量較 tf-idf 低，造成平均準確率較低。