資訊檢索與文字探勘導論

作業三

會計四
B06702064
林聖硯

1. 執行環境

Anaconda Spyder

2. 程式語言

Python 3.6.8

3. 執行方式

(1) 需要 import nltk(preprocessor.py 檔需要)、numpy、pandas 及 math 套件

使用 pip install nltk、pip install numpy、pip install pandas

(2) preprocessor.py 放入 hw1 裡面文字前處理的 function，stopwords.txt 裡面放入要被讀進來的 stopwords，training_index 裡面放入每一個 class 的訓練集文章的編號，IRTM 資料夾裡放入所有文章的 txt 檔

(3) 執行 204 到 211 行即可得到結果

```
204  training_index_txt, training_index_all = get_training_index()
205  df_list, tf_list = calculate_df_tf(training_index_txt)
206  df_all, tf_all = calculuate_df_tf_all(df_list, tf_list)
207  total_training_doc_num = calculate_num_doc(training_index_txt)
208  features = chi_square_feature_selection(df_list, df_all, total_training_doc_num)
209  conditional_prob, prior = train_NB_clf(tf_list, tf_all, features)
210  result_class = test_NB_clf(training_index_all, prior, features, conditional_prob)
211  save_result(result_class)
```

4. 程式處理邏輯

(0) import 需要套件

```
1  #import necessary modules from hw1
2  from Preprocessor import text_preprocessing, get_text
3
4  #import necessary module
5  import numpy as np
6  import pandas as pd
7  import math
```

----------前處理----------

(1) 建立兩個 list

training_index_txt (list of list)： 外層的 list 代表每一個 class，內層的 list 放入所有 training document 的編號並且在結尾加上.txt

training_index_all (list)： 將所有的 training document 的編號放入一個 list 內

```
def get_training_index():
    '''
    training_index_txt : list of list, each element consists of document number in each class with .txt at the end
    training_index_all : list, each element consists of document number in each class
    '''
    training_index_txt = []
    training_index_all = []
    f = open('training_index.txt')
    lines = f.readlines()
    for line in lines:
        line = line.split(' ')
        line = line[1:16]
        temp = []
        for docNum in line:
            training_index_all.append(int(docNum))
            temp.append(docNum + '.txt')
        training_index_txt.append(temp)
    return training_index_txt, training_index_all
```

(2) 計算每個 class 內每個 term 的 term frequency 和 document frequency 並且存入 df_list 及 tf_list 內。兩個 list 中的每一個元素都是一個 dictionary(總共 13 個)，放入每個 term 的 df 及 tf

```
#for each class, calculate document frequency and term frequency for each term and record them by using dictionary
def calculate_df_tf(training_index_txt):
    '''
    df_list : 13 dictionary in a list, each ditionary contains word and its document frequency
    tf_list :13 dictionary in a list, each ditionary contains word and its term frequency
    '''
    df_list = []
    tf_list = []
    for index_by_class in training_index_txt:
        temp_list_df = [] #contains dictionary with keys = words, values = 1
        temp_dict_tf = dict()
        temp_dict_df = dict()
        #iterate each document in class
        for index in index_by_class:
            raw_text = get_text(index)
            text = text_preprocessing(raw_text)

            #count df by combining dictionaries
            temp_list_df.append(dict((word, 1) for word in set(text)))

            #iterate each word in document
            for word in text:
                if word not in temp_dict_tf.keys():
                    temp_dict_tf[word] = 1
                else:
                    temp_dict_tf[word] += 1

        for my_dict in temp_list_df:
            for key, value in my_dict.items():
                temp_dict_df.setdefault(key, 0)
                temp_dict_df[key] += value

        tf_list.append(temp_dict_tf)
        df_list.append(temp_dict_df)

    #do not record word:''
    for my_dict in tf_list:
        my_dict.pop('')

    for my_dict in df_list:
        my_dict.pop('')
    return df_list, tf_list
```

(3) 對所有 term 計算所有 class 合併的 df 及 tf 留著備用

```python
#calculate df and tf for each term for all training documents
def calculuate_df_tf_all(df_list, tf_list):
    df_all = {}
    for my_dict in df_list:
        for key, value in my_dict.items():
            df_all.setdefault(key, 0)
            df_all[key] += value

    tf_all = {}
    for my_dict in tf_list:
        for key, value in my_dict.items():
            tf_all.setdefault(key, 0)
            tf_all[key] += value

    return df_all, tf_all
```

(4) 計算每一個 class document 的數量及所有 class 加起來 document 的數量留著備用

```python
#calculate number of training document
def calculate_num_doc(training_index_txt):
    doc_num = []
    for index in training_index_txt:
        doc_num.append(len(index))
    total_training_doc_num = sum(doc_num)

    return total_training_doc_num
```

----------feature selection----------

(5) chi-square

對所有 class，去計算裡面每一個字的 chi-square，並且挑出每一個 class 內 chi-square 值最高的 500 個存到 set 內，再將這 13 個 set 放入名為 features 的 list 中。

```
############2. for each class, choose 500 top features (words) by chi-square methods###########
def chi_square_feature_selection(df_list, df_all, total_training_doc_num):
    k = 500
    features = []
    for my_dict in df_list:
        word_list = []
        score_list = []
        for word in my_dict.keys():
            word_list.append(word)
            '''
            presentNon : number of document that this word appears in class c
            absentNon : number of document that this word doesn't appear in class c
            presentNoff : number of document that this word appears in class other than c
            absentNoff : number of document that this word doesn't appear in class other than c
            '''
            on_topic = 15
            off_topic = total_training_doc_num - on_topic
            present = df_all[word]
            absent = total_training_doc_num - present

            presentNon = my_dict[word]
            absentNon = on_topic - presentNon
            presentNoff = present - presentNon
            absentNoff = absent - absentNon

            E_00 = absent * off_topic / total_training_doc_num
            E_01 = absent * on_topic / total_training_doc_num
            E_10 = present * off_topic / total_training_doc_num
            E_11 = present * on_topic / total_training_doc_num

            chi_00 = ((absentNoff - E_00) ** 2) / E_00
            chi_01 = ((absentNon - E_01) ** 2) / E_01
            chi_10 = ((presentNoff - E_10) ** 2) / E_10
            chi_11 = ((presentNon - E_11) ** 2) / E_11

            score = chi_00 + chi_01 + chi_10 + chi_11
            score_list.append(score)
        score_arr = np.asarray(score_list)
        index = (-score_arr).argsort()[:k]
        feature_set = set([word_list[i] for i in index])
        features.append(feature_set)
    return features
```

----------NB model----------

(6) training Multinomial Naïve Bass Classifier

對所有的字，計算在每一個 class 內的條件機率，若有出現在挑出來的 500 個 feature 內才存入最後的字典中。最後，對每一個 class 都蒐集這 500 字的條件機率，並且將 13 個字典存入一個名為 conditional_prob 的 list。

```
def train_NB_clf(tf_list, tf_all, features):
    conditional_prob = []
    count = 0
    prior = []
    #iterate each class
    for class_dict in tf_list:
        conditional_prob_dict = dict()
        prior.append(15 / 195)
        word_num = len(tf_all.keys()) ##500 * 13 or 全部不重複的字
        sum_tf = sum(class_dict.values())
        for word in tf_all.keys():
            if word in features[count]:
                #add-one smoothing
                cond_prob = (class_dict[word] + 1) / (sum_tf + word_num)
                conditional_prob_dict[word] = cond_prob
        conditional_prob.append(conditional_prob_dict)
        count += 1

    return conditional_prob, prior
```

(7) predicting class by Multinomial Naïve Bass Classifier

對每個 document 裡面的 term，去對每一個 class 裡面的字典搜尋條件機率，若有找到則取自然對數後加到最後的 score 中。最後每一個 term 都會有 13 個 score，將 class 最高的 score 取出並且存入 result_class 這個 dictionary 內。(key : document index, value : predicting class)

```
def test_NB_clf(training_index_all, prior, features, conditional_prob):
    result_class = dict()
    for index in range(1,1096,1):
        if index not in training_index_all:
            raw_text = get_text(str(index) + '.txt')
            text = text_preprocessing(raw_text)
            score_list = []
            for i in range(13):
                score = 0
                score += math.log(prior[i])
                for word in text:
                    if word in features[i]:
                        score += math.log(conditional_prob[i][word])
                score_list.append(score)
            score_arr = np.asarray(score_list)
            best_class = (-score_arr).argsort()[:1] + 1
            result_class[index] = best_class[0]
    return result_class
```

(8) save result

將結果轉成 dataframe 後存入'hw3_result'.csv 並上傳 kaggle

```
def save_result(result_class):
    result = pd.DataFrame(result_class.items(), columns = ['Id', 'Value'])
    result = result.set_index('Id')
    result.to_csv('hw3_result.csv')
```