HW3 Report

R07725021 資管碩 1 洪靖雯

1.執行環境

- Sublime
- Cmder

2.程式語言

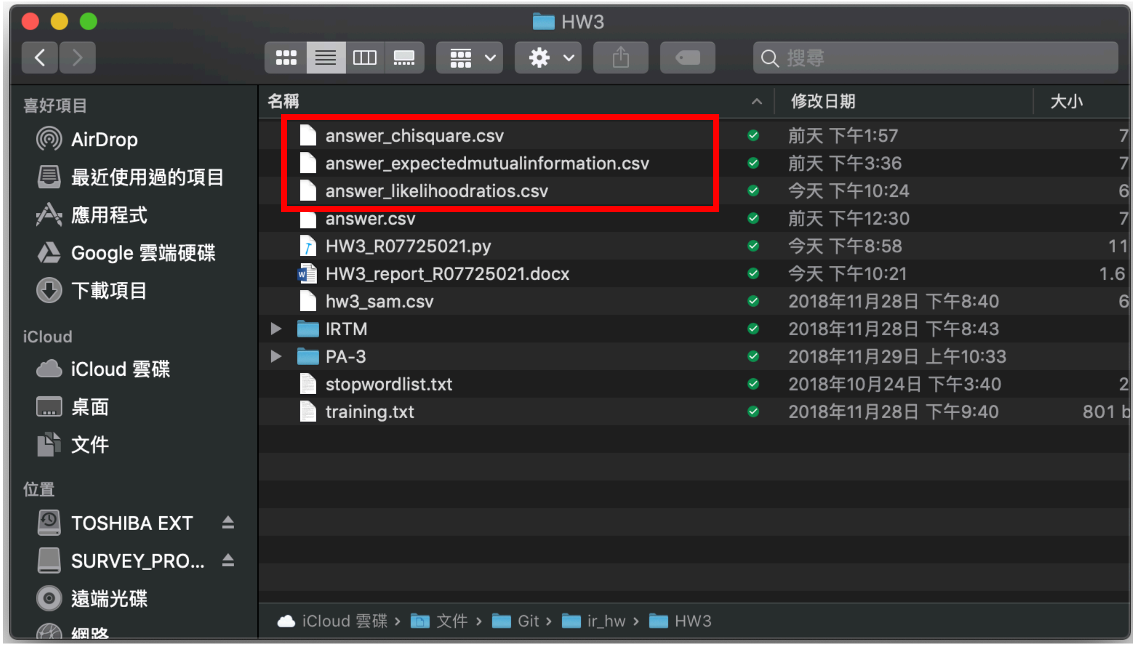
- Python 3.6.6

3.執行方式

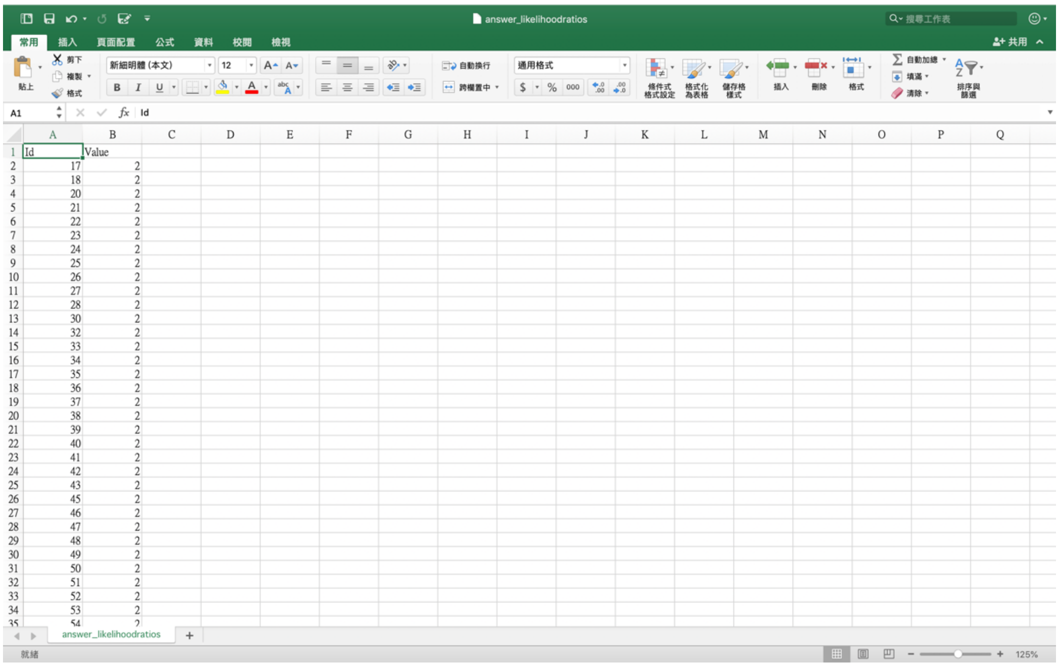
- 套件: nltk
  - 有另外用 conda 4.5.8 架虛擬環境安裝，直接用在 system 裡應該也可行
- 編譯方式: 使用 python HW3_R07725021.py 即可

➢ 輸出結果

i. answer_xxx.csv (總共有三個檔案，分別是用 chi-square, likelihood ratios 和 expected mutual information(EMI) feature selection 做出的結果，預測出來效果略有差異)



● answer_likelihood.csv（目前效果最好），共 900 個 testing data



4.作業處理邏輯說明

i. HW3_R07725021_1.py

- Import 必要的套件

```python
from nltk.tokenize import word_tokenize
from nltk.stem import PorterStemmer
from collections import defaultdict
import math
import csv
```

- 先將 1095 個 documents 做 tokenize(運用之前 HW1 的方法)，運用 dictionary 的結構來儲存，key 為 docid, value 為該文章的 terms。

```python
def tokenize(docid): #term
    file = open("IRTM/"+docid+".txt","r")
    data = file.read()
    file.close()

    #tokenize(用nltk的tokenizer)
    tokendata = word_tokenize(data)
    regex = [c for c in tokendata if c.isalpha()] #去除non-alphanumeric

    #lowercasing
    lowerString = [elements.lower() for elements in regex]

    #stemming
    stemmer = PorterStemmer()
    stemString = [stemmer.stem(lowerstring) for lowerstring in lowerString]

    #stopword removal
    file2 = open("stopwordlist.txt","r")
    stopwordList = file2.read()
    file2.close()
    removeStopword = [word for word in stemString if word not in stopwordList]

    return removeStopword #回傳tokenize後該文章的term (字串)
```

```python
dict_doc = {} #儲存tokenize後的1095文章的term
for i in range(1,1096): #將1095個doc做tokenize
    dict_doc[str(i)] = tokenize(str(i)) #key為docid,value為該doc的所有term (i和docid轉成strin
```

- 讀取 training.txt 得到各 class 及對應的 training doc，並將 training data 做 tokenize！一樣用 dictionary 結構儲存，dict_class 儲存 traing.txt，key 為 classid, value 為 15 個 docid; dict_train_doc 儲存 tokenize 後的 training data，key 為 docid, value 為 terms。

```python
#讀取training.txt，得到各class及對應的training docid
dict_class = {} # key為classid,value為docid_list
with open('training.txt','r') as f:
    for line in f:
        (classID,docID_list) = line.split(' ',1)
        docID_set = docID_list.split() #去除docid中最後面\n
        dict_class[classID] = docID_set

#tokenize training document
dict_train_doc = {} #key為docid,value為terms
for key,value in dict_class.items(): # key為classid,value為docid_list
    for docid in value:
        tokens = tokenize(docid)
        dict_train_doc[docid] = tokens
```

➢ 除去 training data,剩下的為 testing data,也做 tokenize 用 dict 儲存，key 為 docid, value 為 terms。

```python
#去除train doc，剩下1095-195=900個test doc
dict_test_doc = dict_doc.copy() #複製1095個doc
for classid,docid_list in dict_class.items():
    for docid in docid_list:
        del dict_test_doc[docid] #刪除在train data的doc，留下來都當test data
```

➢ 將 training data 做 feature selection，計算每個 term 的數值，用 dict 儲存，key 為 term, value 為 feature selection 值

```python
#feature selection
#計算每個train doc的term的feature selection(chi-square, likelihood ratios, EMI)
dict_feature_selection = {} #key為term,value為feature selection值
for terms_list in dict_train_doc.values(): #key為docid,value為terms，取得每一個term
    for term in terms_list:
        feature_selection = contingencyTable(term,dict_class,dict_train_doc)
        if(dict_feature_selection.get(term)): #有重複的term,跳過
            continue
        dict_feature_selection[term] = feature_selection
```

◆ 先算出 contingency table

有些值像是全部文章數為 1095 篇, 13 個 class 每個 class 全部文章為 15 篇，非該 class 的文章為 195 – 15 = 180 篇；其他皆以 list 來儲存個別 class 的每個 term 的 present 和 absent 儲存，以及 on class 和 not on class 也儲存。後半則以選擇 feature selection 方法來做呼叫。

```
def contingencyTable(term,dict_class,dict_train_doc): #計算每個term在13個class各別的chi-square
    #計算個chi squares需要的值 (c-p,c-a,notc-p,notc-a,c-all,p-all,doc_all)
    doc_all = 195
    c_all = 15
    notc_all = 180

    #計算在每一個class各出現幾個doc(df) [class-present]
    c_p = []
    for classid,docid_list in dict_class.items(): #key為classid,value為docid和terms
        count_df = 0
        for docid in docid_list: #取每一個docid
            if term in dict_train_doc[docid]:#看terms list有沒有含這個term，有就代表這個class中
                count_df += 1
        c_p.append(count_df)  #這個term在這個class的df值

    c_a = [] #[class-absent]
    notc_p = [] #[not class-present]
    notc_a = [] #[not class-absent]

    for i in range(0,13): #第幾個class
        c_a.append(15-c_p[i]) #c_all

        count_notc_p = 0 #計算各個class中not c-present的值
        for j in range(0,13): #加總not c-present總合
            if(j == i):
                continue;
            count_notc_p += c_p[j]
        notc_p.append(count_notc_p)
        notc_a.append(notc_all - notc_p[i])

```

```
    #計算各別class的p_all,a_all
    p_all = []
    a_all = []
    for i in range(0,13):
        p_all.append(c_p[i]+notc_p[i])
        a_all.append(c_a[i]+notc_a[i])

    #chi-square: 計算該term在全部class的4個chi square值，四個chi-square值加總，c_all,notc_all一
    #得到這個term在13個class中最大的chi-square值
    # chi_square_sum = countChiSquare(c_p,c_a,notc_p,notc_a,c_all,notc_all,p_all,a_all,doc

    #likelihood-ratios:計算該term在全部class的likelihood-ratios，得到最大的
    likelihood_ratios_sum = countLikelihoodRatios(c_p,c_a,notc_p,notc_a,doc_all)

    #expected-mutual-information(EMI):計算該term在全部class的4個EMI值，加總後回傳在13個class中最
    # expected_mutual_information_sum = countExpectedMutualInformation(c_p,c_a,notc_p,notc

    # return chi_square_sum
    return likelihood_ratios_sum
    # return expected_mutual_information_sum
```

- ◆ Chi-square feature selection

```
def countChiSquare(c_p,c_a,notc_p,notc_a,c_all,notc_all,p_all,a_all,doc_all): #計算該term在
    chi_square_list = []
    chi_square_sum = 0 #總和
    for i in range(0,13): #計算每一個class的chi-square
        list_p = [p_all[i],a_all[i],p_all[i],a_all[i]]
        list_c = [c_all,c_all,notc_all,notc_all]
        list_observed = [c_p[i],c_a[i],notc_p[i],notc_a[i]]
        for j in range(0,4): #四個chi-square加總
            observed_frequency = list_observed[j]
            expected_count = doc_all*(list_p[j]/doc_all)*(list_c[j]/doc_all)
            chi_square = ((observed_frequency- expected_count)**2)/expected_count
            chi_square_sum += chi_square
        chi_square_list.append(chi_square_sum)

    return max(chi_square_list) #回傳最高的chi-square值

```

- ◆ Likelihood ratios feature selection

```
def countLikelihoodRatios(c_p,c_a,notc_p,notc_a,doc_all): #計算likelihood ratios
    likelihood_ratios_list = []
    likelihood_ratios_sum = 0
    for i in range(0,13):
        upcount = (((c_p[i]+notc_p[i])/doc_all)**c_p[i]) * ((1-((c_p[i]+notc_p[i])/doc_all
        downcount = ((c_p[i]/(c_p[i]+c_a[i]))**c_p[i]) * ((1-(c_p[i]/(c_p[i]+c_a[i])))**c_
        likelihood_ratios_sum = (-2) * math.log(upcount/downcount)
        likelihood_ratios_list.append(likelihood_ratios_sum)
    return max(likelihood_ratios_list)
```

◆ Expected mutual information

```
def countExpectedMutualInformation(c_p,c_a,notc_p,notc_a,c_all,notc_all,p_all,a_all,doc_al
    expected_mutual_information_list = []
    expected_mutual_information_sum = 0
    for i in range(0,13):
        list_p = [p_all[i],a_all[i],p_all[i],a_all[i]]
        list_c = [c_all,c_all,notc_all,notc_all]
        list_observed = [c_p[i],c_a[i],notc_p[i],notc_a[i]]
        for j in range(0,4):
            # print("j:"+str(j)+"\n")
            upcount = list_observed[j]/doc_all
            # print("upcount:"+str(upcount)+"\n")
            downcount = (list_p[j]/doc_all) * (list_c[j]/doc_all)
            # print("downcount:"+ str(downcount)+"\n")
            if(upcount == 0) : #不知為何有list_observed(c_p)為0的情況，會導致log0情況，所以直接加
                expected_mutual_information_sum += 0
            else:
                expected_mutual_information_sum += (list_observed[j]/doc_all) * math.log(
        expected_mutual_information_list.append(expected_mutual_information_sum)

    return max(expected_mutual_information_list)

```

以上述任一種 feature selection 方法，計算出每個 terms 在各 13 個 class 的數值，選擇最大的數值回傳，以做後續的比較。

➢ 計算出 term 的數值後，排序 dict，取前 500 個數值高 terms 數來做 vocabulary。(目前發現 terms 取 150 個預測準確度最高)

```
#取前500個chi-square大的term
feature_selection_list = []
count = 1
for key,value in sorted(dict_feature_selection.items(), key = lambda x:x[1],reverse=True):
    # print("%s %s\n" % (key,value))
    if(count > 150): #500,450,430,400,350,300,200,150,125,100;目前150跑出來最好
        break
    feature_selection_list.append(key)
    count += 1

```

➢ 用這些 feature selection 後的 term，來將 training data 和 testing data 做過濾，只剩下有出現在 500 個 feature selection terms 裡的，排除沒看過的字。

```python
#過濾train data和test data，只剩這500個term
dict_train_doc_filter = {} #key為docid,value為terms
dict_test_doc_filter = {} #key為docid,value為terms
for docid,terms in dict_train_doc.items(): #將train data過濾
    filter_terms = [term for term in terms if term in feature_selection_list]  #只留下在fea
    dict_train_doc_filter[docid] = filter_terms

for docid,terms in dict_test_doc.items(): #將test data過濾
    filter_terms = [term for term in terms if term in feature_selection_list]  #只留下在fea
    dict_test_doc_filter[docid] = filter_terms
```

➢ 接著以 multinomial Naïve Bayes model 來做分類

◆ Training phase, 算出 condprob, prior_c

```python
def trainMultinomialNB(dict_class,dict_train_doc_filter,feature_selection_list): #Multinom
    Nc = 15 # 每個class有15個train doc
    N = 195 # 13個class，有13*15 = 195個train doc
    prior_c = [] #儲存每個class的P(c)
    prior_c.insert(0,0) #index從1開始
    condprob =  defaultdict(dict) #two dimensional dict

    for classid,docid_list in dict_class.items():
        prior_c.insert(int(classid),(Nc/N)) #P(c) = Nc / N
        text_c = 0 #該class所有terms數(textc)
        # text_term = 0 #每個term在該class的doc中總共出現幾次(Tct)
        dict_text_term = {}
        for term in feature_selection_list: #初始化
            dict_text_term[term] = 0

        for docid in docid_list: #取得每個class的docid list，得每一個doc
            text_c += len(dict_train_doc_filter[docid]) #計算這個class總terms數
            #計算每個term in V(feature selection的500個terms)在這個class中所有doc裡出現幾次
            for term in dict_train_doc_filter[docid]:
                dict_text_term[term] += 1

        for term,frequency in dict_text_term.items():
            condprob[term][classid] = (frequency+1)/(text_c+len(dict_text_term.keys())) #

    return condprob,prior_c
```

◆ Testing phase,丟入 testing data，為每一個 test doc 做分類，得出它在 13 個 class 中數值最高，便把它分配在這個 class。

```python
def ApplyMultinomialNB(dict_class,test_data,condprob,prior_c): # multinomial model in test
    #判斷該test doc屬於哪一個class
    score = [] #儲存term在每一個class的score,score = logP(c) + logP(X = t | c)
    score.insert(0,0) #index從1開始
    for classid,docid_list in dict_class.items():
        score.insert(int(classid),math.log(prior_c[int(classid)]))
        for term in test_data: #該test doc的term，在這個class的分數加總，就是這個doc屬於這個class
            score[int(classid)] += math.log(condprob[term][classid])
    del score[0] #刪除為0的 (index恢復到從0開始)
    return score.index(max(score))+1 #最大的值，代表這個class (index為0開始，所以要加一)
```

◆ 最後回傳 test docid 和對應的 classid，以 dict 儲存，key 為 <u>docid</u>, value 為 <u>classid</u>。

```python
#分類
condprob,prior_c = trainMultinomialNB(dict_class,dict_train_doc_filter,feature_selection_l
#test data，算出每一個doc屬於哪一個class
dict_answer = {}
for docid,terms in sorted(dict_test_doc_filter.items()):
    doc_class = ApplyMultinomialNB(dict_class,terms,condprob,prior_c) #得到該doc屬於哪個clas
    dict_answer[int(docid)] = int(doc_class)

```

➢ 最後將答案輸出成 csv 檔

```python
#將答案寫入成csv檔案
# with open('answer_chisquare.csv','w',newline='') as csvfile:
with open('answer_likelihoodratios.csv','w',newline='') as csvfile:
# with open('answer_expectedmutualinformation.csv','w',newline='') as csvfile:
    #建立csv檔寫入器
    writer = csv.writer(csvfile)

    #第一行：id,Value
    writer.writerow(['Id','Value'])
    #第二行開始輸入答案，格式為docid,classid
    for docid,classid in sorted(dict_answer.items()):
        writer.writerow([str(docid),str(classid)])

```