1. 執行環境

- Jupyter Notebook

2. 程式語言&版本

- 語言：Python
- 版本：3.6.6

3. 執行方式

打開 cmd，cd 到檔案位置

再輸入以下指令，即可執行並顯示 output：

藍色部分填入 python.exe 及其路徑，或是如上圖輸入在 python 開啟的指令，python3 是我在環境變數下設定 python 3.6.6 版本的指令

（在環境變數先加入 python.exe 的 path，再更改檔名為 python3.exe, 即可直接啟動）

紅色填入檔名 HW3_r07725044.py
得到 output：

在當前目錄下會產出一個名爲 result.csv 的檔案，是分類後的結果，

也是上傳到 kaggle 的檔案。

4. 作業處理邏輯說明

1. 前處理

- **Import 套件**

將所有要用的套件 import ，用途如註解

```
#用以定義dictionary的資料結構
from collections import defaultdict

#同作業一，斷出term的
import nltk
import string
# from collections import Counter
from nltk.stem.porter import PorterStemmer #import porter algorithm的套件
from nltk.corpus import stopwords

#打開文檔並讀取
import glob
import re
import os
import operator
import sys

#random select
import random

#計算用的sqrt () log ()
import math

#csv寫入
import csv

#矩陣用
import numpy as np

#抓取 training data編號
from bs4 import BeautifulSoup
import requests
from lxml import html
from bs4 import BeautifulSoup
headers = {'User-Agent' :'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/69.0.3497.100 S
```

- **爬蟲得到每個 class 的文件編號**

```
#抓每個類別的下的文件編號
url = 'https://ceiba.ntu.edu.tw/course/88ca22/content/training.txt'
txt = requests.get(url = url, headers = headers)
bs = BeautifulSoup(txt.text, 'lxml')
c_result = bs.select("p")
for s in c_result:
    string1 = s.text
    string1 = nltk.word_tokenize(string1)


#每個類別的文件編號
class_doc_id = {}
total_doc_id = []
for i in range(0, len(string1)):
    if i % 16 == 0:
        doc_id = []
    else:
        doc_id.append(string1[i])
        total_doc_id.append(string1[i])
        continue
    class_doc_id[string1[i]] = doc_id

# print(class_doc_id)
```

- **加入前幾次作業的 function**

將 document 轉換成 term 的 function

```
def term(word):
    res = nltk.word_tokenize(word)
    porter = PorterStemmer()  #定義方法
    stemmer = [ porter.stem(element) for element in res]  #stemming
    stop = set(stopwords.words('english'))
    final = []
    for s in stemmer:
        if s not in stop:
            if s not in final:
                final.append(s)
    return final
```

排序 function：

```
import operator
def sortdict(x):
    new = {}
    for word in x:
        new[word] = index[word]
    sort = sorted(new.items(), key=operator.itemgetter(1))  #根據dictionary的value來排序
    sort = dict(sort)
    return sort
```

計算 term frequency 的 function ：

```
def TF(string):
    tf = defaultdict(int) #建立Dictionary的資料結構，以term作爲key，頻率做value，e.g. 'word' : 3
    #hw1的方式產出term
    res = nltk.word_tokenize(string)
    porter = PorterStemmer()
    stemmer = [ porter.stem(element) for element in res]  #stemming
    stop = set(stopwords.words('english'))
    final = []
    for s in stemmer:
        if s not in stop:
            if s.isalpha():       #判斷是否為英文字母
                final.append(s)
    for t in final:               #將斷出來的字統計為term的次數
        tf[t] += 1

    return tf
```

- **計算各文件的 term frequency 和 term 的 collection frequency**

```
path = 'IRTM/IRTM'  #文件集的 path
N = 15

Class_CF = {}
doc_tf = {}
dictionary = []

#掃過每個class
for classid in class_doc_id:
    CF = defaultdict(int) #儲存每個term出現在 collection 的次數的 dictionary

    #每個文件的 TF 和每個 class 的 CF dictionary，每個 Term 的所有 TF 加起來就是他的 CF
    for fileid in class_doc_id[classid]: #分別讀取每個文件
        filename = glob.glob(os.path.join(path, fileid + '.txt')).pop()
        words = open(filename, 'r').read().lower()  #Lowercase
        word = words.translate(str.maketrans(string.punctuation, ' '*len(string.punctuation))) #將標點符號換成 whitespace，方便處理
        tf = TF(word)
        doc_tf[fileid] = tf
        for w in tf:
            CF[w] += tf[w]
            if w not in dictionary:
                dictionary.append(w)
    Class_CF[classid] = CF
```

建立 training data 的 term list，收錄所有 document 斷出來的 term，以 term 在 list 中的 position 作爲 term 的編號

計算每個 document 中 term 的 tf，存入字典

加總 term 在每個 document 中的 tf，存成 term 的 collection frequency

2. Feature Selection

共嘗試了 3 種方法，如以下

- Chi Square Selection

Chi-Square

```
matrix = np.zeros((13, len(dictionary)))  #把每個term在每個class裏的特徵值算出來
termid = 1

for term in dictionary:
    for classid in Class_CF:
        n11 = 0
        n10 = 0
        n01 = 0
        n00 = 0
        for fileid in class_doc_id[classid]: #判斷 on topic 的 document 是否含有 這個 term，計算 n11 和 n10
            if term in doc_tf[fileid]:
                n11 += 1
            else:
                n10 += 1
        for fileid in total_doc_id:
            if fileid not in class_doc_id[classid]:
                if term in doc_tf[fileid]:
                    n01 += 1
                else:
                    n00 += 1

        #根據公式計算
        N = n11 + n10 + n01 + n00

        E11 = N*((n11+n01)/N)*(n11+n10)/N  #true present
        E10 = N*((n11+n10)/N)*(n10+n00)/N  #true absent
        E01 = N*((n01+n00)/N)*(n11+n01)/N  #false present
        E00 = N*((n01+n00)/N)*(n10+n00)/N  #false absent

        X11 = ((n11-E11) ** 2)/E11
        X10 = ((n10-E10) ** 2)/E10
        X01 = ((n01-E01) ** 2)/E01
        X00 = ((n00-E00) ** 2)/E00

        value = X11 + X10 + X01 + X00
        #存入 matrix
        matrix[int(classid) - 1][termid - 1] = value
    termid += 1  #換下一個 term
```

根據公式，看兩個 random variable 是否無關，計算 4 種狀況下，training data 的文件各有幾篇，分別是 present Λ on topic、absent Λ on topic、present Λ off topic、absent Λ off topic 計算 4 種狀況的 Expected frequency，再結合以上資訊代入公式，得到該 term 對於 class 的特徵值，存入對應的矩陣位置

- **Likelihood Ratio**

Likelihood ratio

```
matrix = np.zeros((13, len(dictionary)))  #把每個term在每個class裏的特徵值算出來
termid = 1


for term in dictionary:
    for classid in Class_CF:
        n11 = 0
        n10 = 0
        n01 = 0
        n00 = 0
        for fileid in class_doc_id[classid]:
            if term in doc_tf[fileid]:
                n11 += 1                    #true present
            else:
                n10 += 1                    #true absent
        for fileid in total_doc_id:
            if fileid not in class_doc_id[classid]:
                if term in doc_tf[fileid]:
                    n01 += 1                #false present
                else:
                    n00 += 1                #false absent

        N = n11 + n10 + n01 + n00
        numerator = ((((n11+n01)/N)**n11) * ((1-((n11+n01)/N))**n10)) * ((((n11+n01)/N)**n01) * ((1-((n11+n01)/N))**n00))
        denominator = (((n11/(n11+n10))**n11) * ((1-n11/(n11+n10))**n10)) * (((n01/(n01+n00))**n01) * ((1-n01/(n01+n00))**n00))

        value = (-2) * math.log(numerator/denominator)   #算兩個假設的 likelihood ratio

        matrix[int(classid) - 1][termid - 1] = value
    termid += 1#換下一個 term
```

類似 Chi Square，產出 4 種狀況下，training data 的文件各有幾篇，代入公式，得到的值存入對應的矩陣位置，大概的概念是，建立 2 個假設，分別是‘term 的出現與 class 有關’和‘term 的出現與 class 有關’，再計算 2 個 Hypothesis 的 likelihood，若 likelihood ratio 是正的，代表 term 和 class 是 dependent。

- **Expected Mutual Information**

類似 Chi Square，產出 4 種狀況下，training data 的文件各有幾篇，代入公式，得到的值存入對應的矩陣位置，大概的概念是這個 term 和 class 有多相關

EMI

```
matrix = np.zeros((13, len(dictionary)))  #把每個term在每個class裏的特徵值算出來
termid = 1

for term in dictionary:
    for classid in Class_CF:
        n11 = 0
        n10 = 0
        n01 = 0
        n00 = 0
        for fileid in class_doc_id[classid]:
            if term in doc_tf[fileid]:
                n11 += 1
            else:
                n10 += 1
        for fileid in total_doc_id:
            if fileid not in class_doc_id[classid]:
                if term in doc_tf[fileid]:
                    n01 += 1
                else:
                    n00 += 1

        #防止 0 的問題出現
        if n11 == 0:
            n11 = 1
        if n10 == 0:
            n10 = 1
        if n01 == 0:
            n01 = 1
        if n00 == 0:
            n00 = 1

        N = n11 + n10 + n01 + n00

        E11 = n11/N  #true present
        E10 = n10/N  #true absent
        E01 = n01/N  #false present
        E00 = n00/N  #false absent

        X11 = E11 * math.log(E11/(((n11+n01)/N)*((n11+n10)/N)))
        X10 = E10 * math.log(E10/(((n11+n10)/N)*((n10+n00)/N)))
        X01 = E01 * math.log(E01/(((n01+n00)/N)*((n11+n01)/N)))
        X00 = E00 * math.log(E00/(((n01+n00)/N)*((n10+n00)/N)))

        value = X11 + X10 + X01 + X00

        matrix[int(classid) - 1][termid - 1] = value
    termid += 1
```

- **特徵選取**

隨機選取每個 class 中特徵值最大的幾個 term，去掉重複的 term，剩下的作爲特徵值

以結果看，performance 最好的是**使用 Likelihood Ratio，每個 class 選取最大的 36 個 term**，因此本次作業采用這組方法和參數，其餘註解

```
def getmax500(matrix):
    feature_list = []
    incase = []
    for i in range(0,13):
        classid_list = matrix[i][0:]
        position = sorted(range(len(classid_list)), key=lambda i: classid_list[i], reverse = True)
#         print(classid_list)

        pos = []
        for p in range(0,36):
            pos.append(position[p])
        for ft in pos:
            if dictionary[ft] not in feature_list:
                feature_list.append(dictionary[ft])

    return feature_list
```

```
feature_term_list = getmax500(matrix)
```

3. Naïve Bayes 分類

- **Training**

依據 Pseudocode:

計算 prior probability，各為 15/195 = 1/13

使用前幾步所提到的 dictionary 和 term list，根據 add one smoothing 後的公式，計算每個 term 在每個 class 出現的機率，存入矩陣對應的位置

```
path = 'IRTM/IRTM'  #文件集的 path
N = 15

condprob = np.zeros((len(feature_term_list), 13))  #建立 condition probability 的 matrix

#掃過每個class
for classid in class_doc_id:
    CF = defaultdict(int) #儲存每個term出現在 collection 的次數的 dictionary

    denominator = 0
    for fileid in class_doc_id[classid]: #分別讀取每個文件
        filename = glob.glob(os.path.join(path, fileid + '.txt')).pop()

        words = open(filename, 'r').read().lower()  #Lowercase
        word = words.translate(str.maketrans(string.punctuation, ' '*len(string.punctuation))) #將標點符號換成 whitespace，方便處理
        tf = TF(word)
        for w in tf:
            CF[w] += tf[w]
            denominator += tf[w]

    denominator = denominator + len(feature_term_list)

    for i in range(0, len(feature_term_list)):
        term = feature_term_list[i]
        prob = (CF[term] + 1) / denominator
        condprob[i][int(classid)-1] = prob
```

- **Testing**

根據 Pseudocode：

建立 Testing data 的 document 的 term list

對於每個 class，先將 prior probability 取 log

再將 term list 中每個 term 對照到矩陣中的位置，如果出現 Out Of Vocabulary 的狀況，則忽略

從矩陣中得到的值做 log，累加

最後得到每個 class 對這個 document 的分數的 list，取 argmax，即分數最大的 class 就是這個 document 屬於的 class

將結果寫入 result.csv 檔案

```
path = 'IRTM/IRTM'  #文件集的 path
  # 建立 CSV 檔寫入器
writer = open('result.csv', 'w', newline='')
writerows = csv.writer(writer)
writerows.writerow(['Id', 'Value'])

for fileid in testing_doc_id: #分別讀取每個文件
    score = []
    row = []
    row.append(fileid)

    #prior probability
    for i in range(0,13):
        score.append(math.log(1/13))

    #取 testing data 中每個 document 的 term
    filename = glob.glob(os.path.join(path, fileid + '.txt')).pop()
    words = open(filename, 'r').read().lower()  #Lowercase
    word = words.translate(str.maketrans(string.punctuation, ' '*len(string.punctuation))) #將標點符號換成 whitespace，方便處理
    token = tokenize(word)

    #計算對於這個 document，每個class的score
    for classid in range(0, 13):
        for t in token:
            if t in feature_term_list:
                tid = feature_term_list.index(t)
                score[classid] += math.log(condprob[tid][classid])

    #取 argmax
    row.append(str(score.index(max(score)) + 1))
    writerows.writerow(row)
writer.close()
```

將 result.csv 上傳 Kaggle 的結果：