1. 執行環境：Jupyter Notebook
2. 程式語言：Python 3
3. 執行方式：
    1. Open terminal
    2. $ pip install nltk (for python)
    3. $ python3 pa3.py
    4. Output (csv file): （藍色粗體）

    .
    ├── **output: hw3_final_r10725048.csv: 紀錄 testing document Id 及其分類結果 Value。**
    ├── pa3.py
    └── report

4. 作業處理邏輯說明：
    1. 載入套件

```
import re
import os
import nltk
import math
import numpy as np
import pandas as pd

from nltk.stem import PorterStemmer
from nltk.corpus import stopwords
nltk.download('stopwords')
```

    2. Preprocessing (copy from pa2.py)
    這次作業沒有特別修改 preprocessing，直接沿用 pa2

```
def preprocess(text):  # pa1: preprocess function

    # tokenization
    tokens = re.findall("[a-zA-Z]+", text)
    # lower-casing
    lowercase = [token.lower() for token in tokens]
    # stop-words removal
    stops = stopwords.words('english')
    filtered = [w for w in lowercase if w not in stops]
    # stemming
    ps = PorterStemmer()
    stemming = [ps.stem(i) for i in filtered]

    # remove single character like 'a', 'x', 'w'...
    word = [i for i in stemming if len(i) > 1]

    return word
```

3. Training

3.1.事前準備：這部分在讀取 training.txt 檔案，建立所有 training data 之 token 的字典（講義上的大V），以及個別 class 的字典。作為下一步驟要計算 Likelihood ratio 的事前準備。

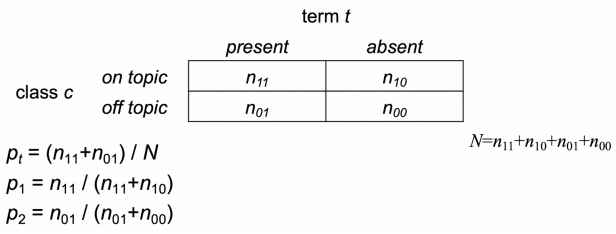
- getClassesList(C): 輸入 classes 資訊，也就是 training.txt，取得各 classes 的 training data 編號。輸出 data frame 與其 1–d list 的格式：

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0															
1	11	19	29	113	115	169	278	301	316	317	321	324	325	338	341
2	1	2	3	4	5	6	7	8	9	10	12	13	14	15	16
3	813	817	818	819	820	821	822	824	825	826	828	829	830	832	833
4	635	680	683	702	704	705	706	708	709	719	720	722	723	724	726
5	646	751	781	794	798	799	801	812	815	823	831	839	840	841	842
6	995	998	999	1003	1005	1006	1007	1009	1011	1012	1013	1014	1015	1016	1019
7	700	730	731	732	733	735	740	744	752	754	755	756	757	759	760
8	262	296	304	308	337	397	401	443	445	450	466	480	513	533	534
9	130	131	132	133	134	135	136	137	138	139	140	141	142	143	145
10	31	44	70	83	86	92	100	102	305	309	315	320	326	327	328
11	240	241	243	244	245	248	250	254	255	256	258	260	275	279	295
12	535	542	571	573	574	575	576	578	581	582	583	584	585	586	588
13	485	520	523	526	527	529	530	531	532	536	537	538	539	540	541

- 取得 training data 組成的字典 (trainV) 與每個字的 document frequency（此數字 ≤ 195）
- 取得每個 class 各自的字典 (list_classDict) 與其中每個字的 document frequency（此數字 ≤ 15）

3.2.Feature Selection: Likelihood Ratio

- 輸入 class id 以及 k (最後每個 class 欲篩選出**幾個**代表性單字)
- 這部分參考課堂講義計算 n11, n10, n01, n00 與 likelihood



- 單一 class 中所有 token 之 likelihood 計算如下圖所示

```
# Feature Selection with Likelihood Ratio
def lrFeatureSelection(c, k):
    '''
    Parameter
    - c: target class number
    - k: output number of feature selection
    "Global dict./word" in this block means dict./word of all training data
    '''

    likelihood = {}  # store likelihood ratio of each word in global dictionary
    sortedLikelihood = {}  # store sorted likedlihood
    topWordEachClass = []  # store top k word for each class

    for w in trainV:  # trainV is global dictionary

        # print(w)  # key
        # print(trainV[w])  # value: n_11+n_01

        if w in list_classDict[c]:  # count n_11 if global word exists in classX
            n_11 = list_classDict[c][w]
        else:
            n_11 = 0  # global word not found in classX

        n_01 = trainV[w] - n_11
        n_10 = 15 - n_11
        n_00 = 180 - n_01
        p_t = (n_11+n_01)/195
        p_1 = n_11/15
        p_2 = n_01/180

        result = -2*math.log( (p_t**n_11*(1-p_t)**n_10*p_t**n_01*(1-p_t)**n_00) / (p_1**n_11*(1-p_1)**n_10*p_2**n_01*(
        likelihood.update({w: result})

    sortedLikelihood = sorted(likelihood.items(), key=lambda x:x[1], reverse=True)[:k]  # sort likelihood in descendir
    topWordEachClass =[i[0] for i in sortedLikelihood]  # store top sorted word into list

    return topWordEachClass
```

- 上面為單一 class 裡面所有單字的 likelihood 計算，但我們要 apply 到所有 class，取得所有訓練 class 的重要單字 (topWordClass)，如下圖：

```
# apply feature selection to all classes
'''
store top word of each class like:
{1:['a','b','c',...],
 2:['d','e','f',...],...}
'''
topWordClass = {}

for i in range(1,14):
    topWordClass[i] = lrFeatureSelection(i, 30)
```

- 照理說按照上面執行完，總共會有 13(#class)*30(#k) =390 個重要的字，但是為了避免不同 class 中出現一樣的重要單字造成混淆，決定只保留不重複的字並存到 trainV_featured，最後其長度為 383，證明真的有 7 個重要單字重複出現在不同 class 中。

```
# store all classes' top words into flatten list with non-duplicated
trainV_featured = []

for c in topWordClass:
    for w in topWordClass[c]:
        # print(w)
        if w not in trainV_featured:
            trainV_featured.append(w)
trainV_featured
```

- 由於在寫 likelihood 的過程中觀察到其數值下降滿快的，如果取到前 500 個單字可能也會涵蓋到許多不重要的字，於是將 k 定為 100 以內的數字，並根據 Kaggle 的結果進行微調，細節將在第五點 (p.5) 提及。

3.3.Training with Naive Bayes：這部分在用 Multinomial Naive Bayes訓練分類器，會用到幾個 functions 如下：

- concatAllDocInClass: 將某個 class 的所有文件 tokenized 並儲存到一個 list
- countTokensOfTerm: 由 training data 建立出的字典計算每個字在每個 class 分別出現幾次
- 開始訓練時，其實就是在計算事前機率 (prior) 與事後的條件機率 (condProb)，並記錄下來在 testing 階段使用。

```
# training

countTrainData = 195  # total number of training documents
countDocsInClass = 15  # number of documents in each class

# store prior prob. for all classes
prior = {}  # they're all same(0.077)

# store every conditional probability of words in each class
condProb = {}  # 383x13


D = "data"

for c in range(1,14):
    # print(c)
    prior[c] = countDocsInClass/countTrainData
    text_c = concatAllDocInClass(D,c)
    lenText = sum(len(t) for t in text_c)  # total text length for each class(words may be repeated)
    term_count_each_class = countTokensOfTerm(text_c, trainV_featured)  # 算global dict.的字在這個class出現幾次
    # print(term_count_ineachClass)

    eachClassCondProb = {}

    for t in trainV_featured:
        # print(t)
        n_tct_smooth = term_count_each_class[t]+1  # 分子
        eachClassCondProb.update({t: (n_tct_smooth / (lenText+len(trainV_featured)))})

    condProb[c] = eachClassCondProb
```

4. Testing

4.1.applyNaiveBayes: (僅單一文件)

- 輸入欲分類之文件 id
- 計算該文件 tokenized 之後回傳該文件最高分時的 parameter index 值
- 以本次作業來說，各個 class 的事前機率都一樣，所以最重要的地方在於分類器看到 term 之後的條件機率（上個步驟中的 condProb），也就是下圖紅框處：

```
def applyNaiveBayes(testingDoc):
    '''
    testingDoc: index of testing doc
    '''
    with open('./data/' + str(testingDoc) + '.txt', 'rb') as txt:
        content = txt.read().decode('latin-1').replace('\n', '')  # decoding and ignore new line
        tokens = preprocess(content)

    W = []  # testingDoc ∩ training vocabulary
    for w in tokens:
        if w in trainV_featured:
            W.append(w)  # 取出認得的字 for 之後計算分數

    score = {}  # score of this testing document in each class
    for i in range(1, 14):
        score[i] = math.log(prior[i])  # basic
        for t in W:
            score[i] += math.log(condProb[i][t])  # add conditional probability

    return max(score, key=score.get)
```

4.2.套用在所有 testing data，取得分類結果並輸出成 csv 檔。

```
# apply Naive Bayes on testing file
apply = [applyNaiveBayes(i) for i in test_id]
apply
```

```
# store result as dataframe
output = pd.DataFrame(apply, index=test_id, columns=['Value'])
```

```
# output result dataframe into csv
output.to_csv('hw3_final_r10725048.csv', index_label = ['Id'])
```

5. Upload to Kaggle

第一次上傳時因為格式錯誤 score 是 0 分，
之後成功上傳的幾次分別有 k=50, k=100, k=30, k=25，
表現最好的一次為 k=30，score 99.333%，也就是分錯 6 篇文章。

Submission and Description	Public Score
hw3_max_25.csv a day ago by chh	0.99222
hw3_max_30.csv a day ago by chh	0.99333
hw3_max_100.csv a day ago by chh	0.97222
hw3_max.csv k=50 a day ago by chh	0.98111