1. 執行環境: VS code
2. 程式語言: Python 3.11.5
3. 執行需求:
   - 文章資料放置在./data/ 路徑之下
   - Final_result 最終結果會放置與 pa3.py 同一層資料夾
   - 需要 pip install numpy, nltk, pandas
4. 作業處理邏輯

Step1. 將資料讀進，並切割成 training 與 testing，並各自坐前處理 (preprocessing)

```python
def preprocessing(text):
    text = text.lower()
    text = Tokenize(text)
    text = RemoveStopwords(text,StopWords)
    text = Porter_Stemmmer(text)
    return text

all_text = pd.DataFrame(columns=['index', 'Text','class']) #將所有文檔轉換成 dataframe
for text_number in range(1, 1096):
    text = OpenFile(f'.\data\{text_number}.txt')
    text= preprocessing(text)
    new_row = pd.DataFrame({'index': [f"{str(text_number)}"], 'Text': [text]})
    all_text = pd.concat([all_text, new_row], ignore_index=True)
train_index = OpenFile('.\\training.txt') #get training data index
train_index=train_index.split('\n')
for i in range(len(train_index)):
    train_index[i] = train_index[i].split(" ")
    if '' in train_index[i]:
        train_index[i].remove('')
    train_index[i]= [int(x) for x in train_index[i]]
train_dic={lis[0]:lis[1:] for lis in train_index} #turn training class's doc_id into dict

#split training set & testing set
training = pd.DataFrame(columns=['index', 'Text','class'])
testing = pd.DataFrame(columns=['index','Text','class'])
for index_list in train_index:
    for index in index_list[1:]:
        row = all_text[all_text['index']== str(index)]
        all_text.drop(all_text[all_text['index']==
str(index)].index,inplace= True)
        row['class']=index_list[0]
        training=pd.concat([training,row],ignore_index=True)
testing = all_text.reset_index()
```

Step2.　計算 training 與 testing 各自的 term frequency，放置在'tf'的欄位中

```
#for each documents in training & testing dataset get it's term
frequency
training['tf']= training['Text'].apply(TermFrequency)
training = training[['index','Text','tf','class']]
testing['tf']= testing['Text'].apply(TermFrequency)
testing = testing[['index','Text','tf','class']]
```

Step3.　利用 chi_square 在 training 中得到前 500 個有影響力的 term，將結果儲存在 new_vocabulary

```
#get dictionary
def Vocabulary(token_lists:list):
    vocabulary=list()
    for token_list in token_lists:
        keys = token_list.keys()
        for key in keys:
            if key not in vocabulary:
                vocabulary.append(key)
    return vocabulary

#get chi_square
def ChiSquare(df:pd.DataFrame, Class:dict):
    vocabulary = Vocabulary(df.tf)
    chi = dict()
    N = len(df)
    for term in vocabulary:
        chi_value = 0
        martix = dict()
        martix['tpresent'] = df[df['tf'].apply(lambda x: term in x)]
        martix['tabsent'] = df[df['tf'].apply(lambda x: term not in x)]
        for c in train_dic:
            martix['cpresent'] = df[df['class']==c]
            martix['cabsent'] = df[df['class']!=c]
            martix['tpresent_cpresent'] =
martix['tpresent'][martix['tpresent']['class'] == c]
            martix['tpresent_cabsent'] =
martix['tpresent'][martix['tpresent']['class'] != c]
            martix['tabsent_cpresent'] =
martix['tabsent'][martix['tabsent']['class'] == c]
            martix['tabsent_cabsent'] =
martix['tabsent'][martix['tabsent']['class'] != c]
            temp_chi = 0
            for i in ['tpresent','tabsent']:
                for j in ['cpresent','cabsent']:
                    E = len(martix[i]) * len(martix[j]) / N
                    temp_chi += ((len(martix[f'{i}_{j}']) - E) ** 2) / E
            chi_value += temp_chi
        chi[term] = chi_value

    vocabulary = sorted(chi, key=chi.get, reverse=True)[:500]
    return vocabulary

#get top 500 term as new_vocabulary
new_vocabulary = ChiSquare(training,train_dic)
```

Step4.　利用 train_NB 涵式訓練模型

```
def train_NB(C:dict,df:pd.DataFrame,vocabulary:list):
    N = len(df)
    prior = dict()
    condprob = {term: {} for term in vocabulary}
    for c in C:
        n = len(C[c])
        class_text = df[df['class'] == c]
        print(n)
        prior[c] = n / N
        tct = dict()
        for term in vocabulary:
            Tct = 0
            for row in class_text['tf']:
                if term in row:
                    Tct += int(row[term])
            tct[term] = Tct
        for term in vocabulary:
            condprob[term][c] = (tct[term]+1) / 
(sum(tct.values())+len(vocabulary))
    return vocabulary, prior, condprob

v,p,con = train_NB(train_dic,training, new_vocabulary)
```

Step5.　將模型套用在 testing 得到最終預測的分類結果

```
def apply_NB(df,C,vocabulary,p,con):
    tf = df['tf']
    score = dict()
    for c in C:
        score[c] = np.log(p[c])
        for term in tf:
            if term in vocabulary:
                score[c] += (np.log(con[term][c])*(int(tf[term])))
    return max(score,key=score.get)
testing['class'] = None
testing['class'] = testing.apply(apply_NB, C=train_dic, 
vocabulary=new_vocabulary, p=p, con=con, axis=1)
Step6.
```

Kaggle score : 0.98111