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

1. 作業處理邏輯
2. 將資料讀進，並切割成training與testing，並各自坐前處理(preprocessing)

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()

1. 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']]

1. 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)

1. 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)

1. 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)

Kaggle score : 0.98111