Assign 5 IR
Text Classification

- Shalini Bhardwaj(MT19045)

1.Preprocessing

Dataset Used: 5 folders of the 20news_group:

comp.graphics, sci.med, talk.politics.misc, rec.sport.hockey,sci.space

These 5 folders are assumed to be the 5 classes and documents inside them are classified after splitting them into test and train.

1.As per the user selection test and train are split into desired ratio 50 :50 or 70:30 or 80:20 after shuffling the all 5000 documents.

2.It is assumed each class is numbered as c1,c2,c3 ,c4 and c5 respectively.

3.Every document is read from all folders based on whether it is present on training or testing list of documents .

4.Various information in form of dictionaries is saved while preprocessing/reading each document likefor testing documnets list of tokens present in all documents of a particular class are saved and various information needed for tfidf calculation is done at preprocessing time.

Preprocessing for all terms in documents is done by following steps:

1. To Lowercase: query is transformed to lower case
2. Punctuation is removed.
3. Stop words are removed
4. Tokenization : converted to tokens
5. Lemmatization : converting to root dictionary word

Methodology:
Naïve Bay's

1. For tfidf as feature selection:
   In each class (c1,c2,c3,c4,c5) for each term tf idf is calculated.
   Tf: number of times that term is present in class
   Idf: in how many classes term is present.

```
def cal_tf_idf():
    for  class1 in class_list:
        print(class1,"in cal tf idf")
        for word2 in list(set(text_class_train[class1])):
            tf_idf[class1][word2]=(1+math.log((text_class_train[class1].count(word2))+1))*(5/df_dic[word2])
    print("tf_idf done")
```

   This dictionary is sorted and K highest words are selected.

K is chosen to be top k% terms of that class.

Now, we have reduced the features to this K% of that class.Training of model is done assuming this is the new vocabulary.

2.If the feature selection is MI then

MI is calculated that is number of documents considering N11,N01,N00,N10.

MI is calculated between each term of class and the class in which it is present.

For each class based on the top k% mi score term. Term list of that class is updated to be that of new terms extracted.

MI is calculated as: for every term numberof documents present in it is stored as N11

Rest N10,N10,N00 are done below.

Like here for class c1 the MI is calculated for its every terms as shown below.

```
for i in mi_info['c1']:   # i: word
    n11=mi_info['c1'][i]
    n01=doc_count['c1']-n11
    for p in class_list:
        if p != 'c1':
            if i not in mi_info[p]:
                mi_info[p][i]=0
    n10=mi_info['c2'][i]+mi_info['c3'][i]+mi_info['c4'][i]+mi_info['c5'][i]
    n00=(doc_count['c2']+doc_count['c3']+doc_count['c4']+doc_count['c5'])-n10
    mi=cal_mi(n11,n01,n10,n00)
    store_mi['c1'][i]=mi
```

```
def cal_mi(n11,n01,n10,n00):
    N=len(train)
    n1_= n11+n01
    n_1= n01+n11
    n0_= n01+n00
    n_0= n10+n00
    m=0

    if((N*n11)!=0 and (n1_*n_1)!=0):
        m+=((n11/N)*math.log2((N*n11)/(n1_*n_1)))
    if((N*n10)!=0 and (n1_*n_0)!=0):
        m+=((n10/N)*math.log2((N*n10)/(n1_*n_0)))
    if((N*n01)!=0 and (n0_*n_1)!=0):
        m+=((n01/N)*math.log2((N*n01)/(n0_*n_1)))
    if((N*n00)!=0 and (n0_*n_0)!=0):
        m+=((n00/N)*math.log2((N*n00)/(n0_*n_0)))
```

After the feature selection is done. Training is done:

Training

In this training,we need 1)conditional probability 2)Prior probability

Prior Probability is calculated : Number of docs in that class/total number of docs.

Conditional Prob : for vocab of each class its conditional prob was calculated

While Testing :

Each testing document is to be classified.Therefore for each of the class, score of document to that class is calculated.Class which receives maximum score is assigned. During calvulating score conditional probability of terms saved during testing is used.

2.KNN

For KNN feature selection methods are same as done in Naïve bays.

IN Knn as per k chosen(1,3,5).K nearest neighbours are chosen and majority score is assigned.

Score is calculated by cosine similarity between every test doc and train doc which contains training terms.

Docs are sorted in descending as per score.

For each class top k docs are summed with respective classes.Class which ha shigh score is finally assigned.

To run the code: Follow steps below

1.Run this cell located near bottom to initialize before running any of classifier

```
text_class_train={}                  #all class and its token
selected_count_class={}             #total number of count of all  selected by tf idfword in class
selected_count={}
selected_count_mi={}# stores selected by tf idf count/word in class
text_class_test={}
cond_prob={}                         #conditional prob of class
prior_prob={}
doc_count={}                          # no.of docs in class
test_doc={}
train_doc={}
store_mi={}#docs in tetsing with terms
selected_terms={}
selected_terms_mi={}
df_dic={}
tf_idf={}
score={}
mi_info={}
all_doc=[]
train =[]
test=[]
ground_truth=[]
predicted=[]
pred=[]
selected_terms_un={}

test_doc_knn={}
```

Select the size of train set ,here 0.8 is done

```
for j in range(1,5001):
    all_doc.append(j)
print(len(train))
print(len(test))
random.shuffle(all_doc)
train_size=(0.8*5000)

split_train_test(all_doc,train_size)
print(len(train))
```

```
0
0
4000
1000
4000
```

Then chose from various variations

```
Choose the method for text classification:
1.Tf-idf Naive Bays   2.MI Naive Bays 3.Tfidf KNN 4.MI KNN
3
20_newsgroups_assi4\comp.graphics
in test.............. 7
in test              12
```

Assumption:

1.classes are assumed to be c1 c2 c3 c4 c5 respectively.

2.Everytime randomly docs are shuffled to get train /test docs and then 80 /70/ 50 for training is fetched.

3.Docs are numbered 1,2,3…5000