TEXT CLASSIFICATION USING NAIVE BAYES CLASSIFIER

Multinomial Naive Bayes

GETTING THE DATA

For this problem, we'll be using the 20 Newsgroups data set. There are several versions of it on the web. We download "20news-bydate.tar.gz" from

http://qwone.com/~jason/20Newsgroups/

Unpack it and look through the directories at some of the files. Overall, there are roughly 19,000 documents, each from one of 20 newsgroups. The label of a document is the identity of its newsgroup. The documents are divided into a training set and a test set.

The same website has a processed version of the data, "20news-bydate-matlab.tgz", that is particularly convenient to use. Download this and also the file "vocabulary.txt". Look at the first training document in the processed set and the corresponding original text document to understand the relation between the two.

I suggest to find the relation on your own first. The relation is explained in the next slide

B. RELATION BETWEEN THE DATA

Train.data

	docid	wordid	count
0	1	1	4
1	1	2	2
2	1	3	10
3	1	4	4
4	1	5	2

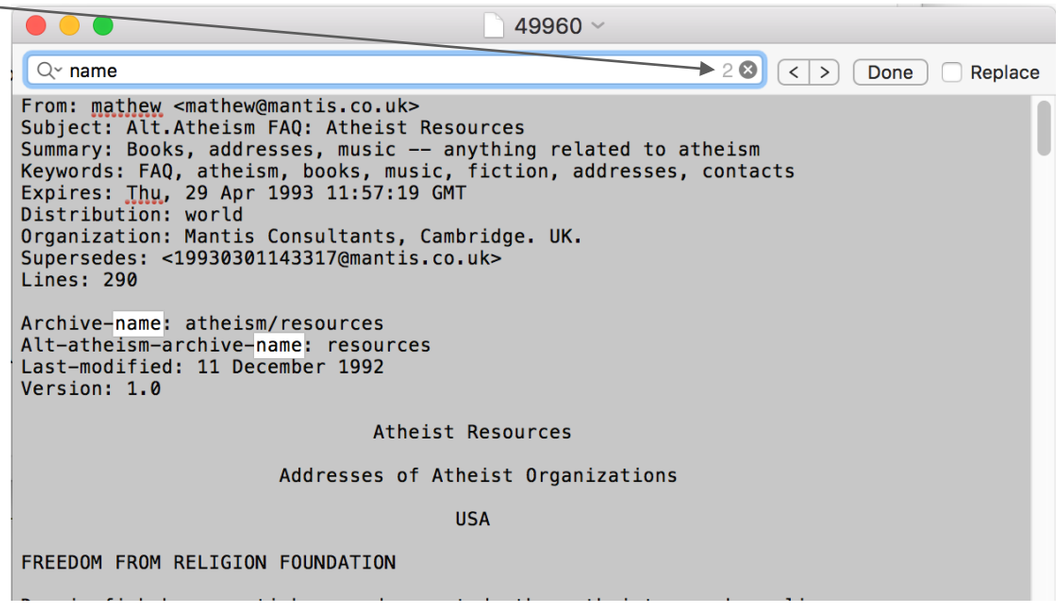
Wordid=vocabid

Vocabulary

	vocabulary
1	archive
2	name
3	atheism
4	resources
5	alt

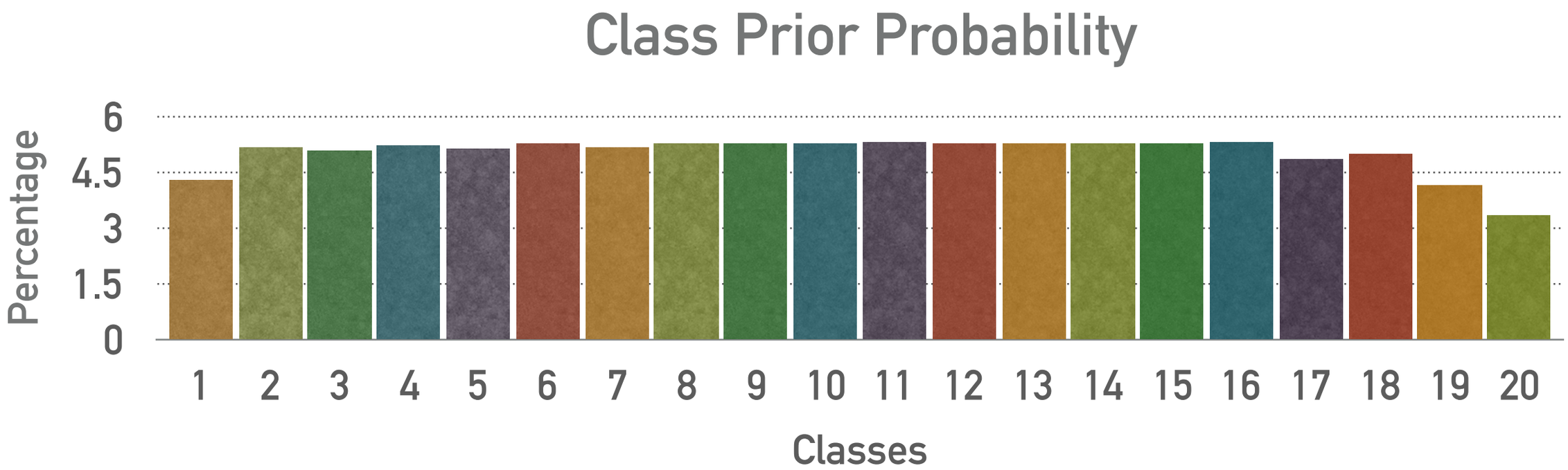
count is the # of times word appears in doc

example word name appears twice in doc1



C. CALCULATE PI_J

PI_J - Fraction of documents in each class



You can find the code to calculate prior probabilities in Appendix

C. CALCULATE P_J

P_J - Probability distribution over V for each class

P_Jw- Probability of each word in each class

*We end up with a 20 * 53975 matrix*

Have shown a part of is below:-

wordid	1	2	3	4	5	6	7	8	9	10	...	53966	53967
classid													
1	6.192857e-05	0.000300	1.309548e-03	4.288095e-05	0.000391	0.000195	2.859524e-05	4.785714e-06	0.000162	6.666905e-04	...	2.380952e-08	2.380952e-08
2	3.497896e-04	0.000344	2.380952e-08	9.912793e-05	0.000082	0.000338	5.832255e-05	3.500519e-05	0.001003	1.751717e-05	...	2.380952e-08	2.380952e-08
3	7.242276e-05	0.000454	2.380952e-08	1.119081e-04	0.000138	0.000224	1.319470e-05	1.319470e-05	0.000948	2.380952e-08	...	2.380952e-08	2.380952e-08
4	4.992703e-05	0.000193	2.380952e-08	2.380952e-08	0.000062	0.000299	1.250515e-05	6.268165e-06	0.000299	2.380952e-08	...	2.380952e-08	2.380952e-08
5	4.074557e-05	0.000224	2.380952e-08	6.819200e-06	0.000007	0.000319	6.819200e-06	2.380952e-08	0.000319	2.380952e-08	...	2.380952e-08	2.380952e-08
6	2.196146e-04	0.001037	2.380952e-08	3.691236e-04	0.000070	0.000243	9.813861e-05	1.403983e-05	0.001107	2.380952e-08	...	2.380952e-08	2.380952e-08
7	2.380952e-08	0.000229	2.380952e-08	1.639653e-05	0.000016	0.000262	2.380952e-08	2.380952e-08	0.000229	2.457434e-05	...	2.380952e-08	2.380952e-08
8	5.137201e-05	0.000308	2.380952e-08	2.380952e-08	0.000074	0.000491	3.996235e-05	1.714302e-05	0.000103	2.380952e-08	...	2.380952e-08	2.380952e-08
9	8.549069e-05	0.000409	2.380952e-08	2.444771e-05	0.000024	0.000507	1.834342e-05	6.134820e-06	0.000024	2.380952e-08	...	2.380952e-08	2.380952e-08
10	5.943721e-06	0.000195	2.380952e-08	1.185787e-05	0.000006	0.001786	5.943721e-06	2.380952e-08	0.000012	2.380952e-08	...	2.380952e-08	2.380952e-08
11	4.964066e-06	0.000331	2.380952e-08	2.380952e-08	0.000005	0.001032	9.903435e-06	3.953965e-05	0.000020	2.380952e-08	...	2.380952e-08	2.380952e-08
12	1.987624e-04	0.000344	2.380952e-08	4.206097e-05	0.000226	0.000310	9.556879e-05	2.677302e-05	0.000424	2.380952e-08	...	2.380952e-08	2.380952e-08
13	1.828293e-05	0.000201	2.380952e-08	1.219876e-05	0.000030	0.000183	3.045126e-05	1.219876e-05	0.000176	2.380952e-08	...	2.380952e-08	2.380952e-08
14	6.929884e-05	0.000180	2.380952e-08	6.006207e-05	0.000092	0.000319	4.641475e-06	4.158854e-05	0.000115	2.380952e-08	...	2.380952e-08	2.380952e-08
15	2.233809e-04	0.000382	2.380952e-08	1.023955e-04	0.000056	0.000475	1.070488e-04	4.190282e-05	0.000112	2.380952e-08	...	2.380952e-08	2.380952e-08
16	2.380952e-08	0.000469	6.098188e-05	2.669029e-05	0.000053	0.000385	2.380952e-08	2.380952e-08	0.000065	2.133890e-04	...	2.380952e-08	2.380952e-08
17	8.015538e-05	0.000139	2.380952e-08	2.532665e-05	0.000046	0.000447	8.456276e-06	3.376184e-05	0.000080	2.380952e-08	...	2.380952e-08	2.380952e-08
18	3.166209e-05	0.000487	2.380952e-08	2.849747e-05	0.000006	0.000494	2.533284e-05	9.495464e-05	0.000032	3.180450e-06	...	2.380952e-08	2.380952e-08
19	2.380952e-08	0.000158	2.380952e-08	9.290670e-05	0.000069	0.000610	4.058737e-06	6.867544e-05	0.000040	2.380952e-08	...	2.380952e-08	2.380952e-08
20	2.380952e-08	0.000250	4.994897e-05	1.112134e-05	0.000128	0.000244	2.776175e-05	5.574538e-06	0.000105	2.776175e-05	...	5.574538e-06	5.574538e-06

Note: The image here has Laplace smoothing

C. LOGIC TO CREATE P_JW AND P_J

1. *Let CountWij be the number of times word Wi occurs in class j*
2. *Let CountWj be the total number of words in Class j*
3. *Then P_JW = CountWij/CountWj*
4. *We calculate P_JW for all the words for each class*
5. *This gives us a 20*53975 matrix which is P_J*

Please find the code to derive the distribution in Appendix

D. WRITING A ROUTINE TO PREDICT CLASSES

We have calculated the following:-

1. *Prior probability of classes*
2. *Probability of words in each class*

ALGORITHM:-

Input - List of words in document with their freq.

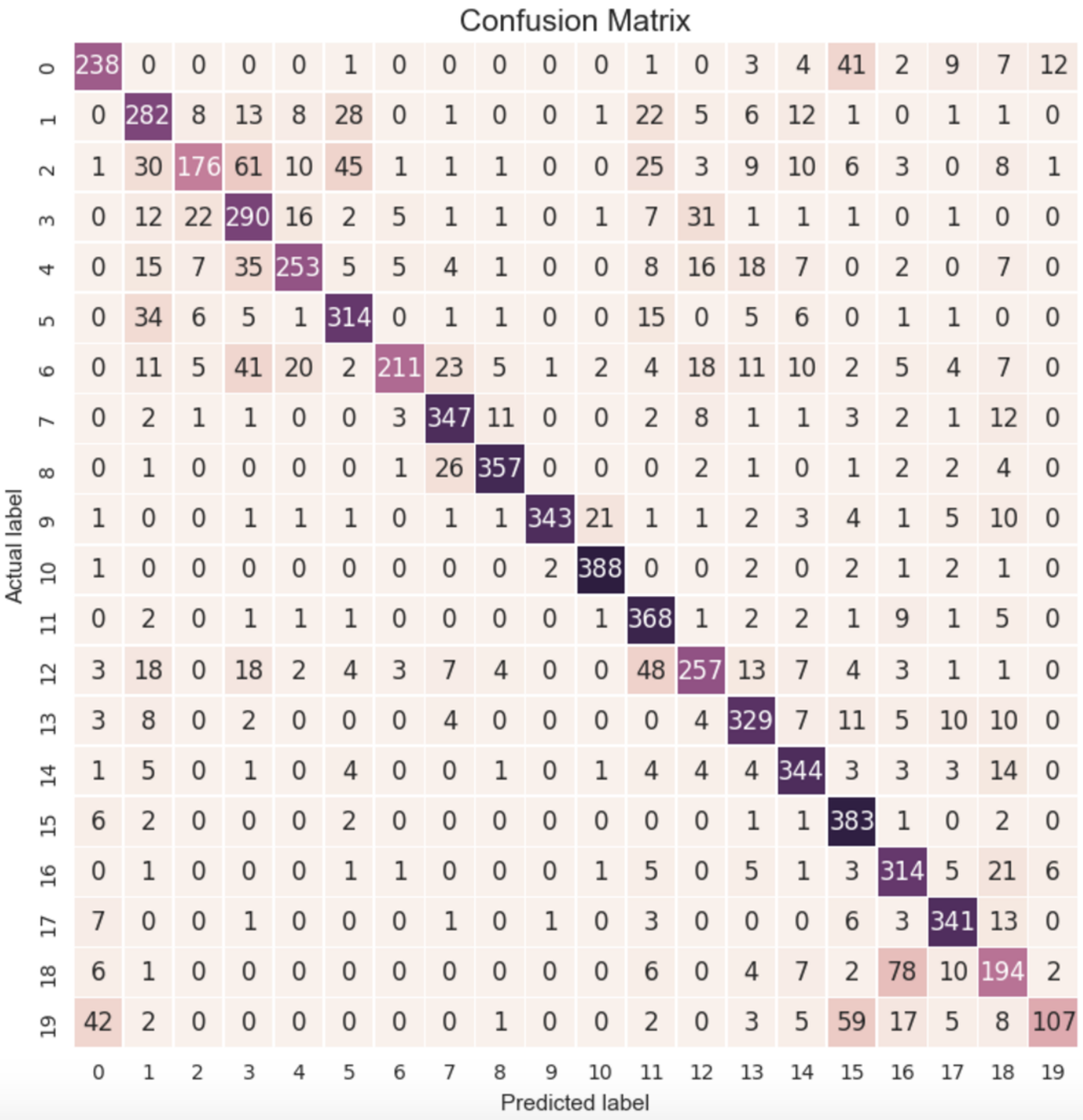
Output - Predicted class of the given document

1. *For class Ci (where i - 1 to 20), Calculate prob. P(w|Ci) - Likelihood of each word in class Ci*
2. *Calculate power(P(w|j),f) * P(Ci)*
3. *Take the log of the above value*
4. *Add up all values for class Ci*
5. *This gives us the likelihood of the document in class Ci*
6. *Repeat steps 1 to 5 for all classes*
7. *Return the class with max likelihood among the 20 classes*

Please find the code for the same in Appendix

E. HOW DID MY MODEL PERFORM? (METHOD1)

Accuracy score - 77.76149233844104 1%



Classes

1 alt.atheism
2 comp.graphics
3 comp.os.ms-windows.misc
4 comp.sys.ibm.pc.hardware
5 comp.sys.mac.hardware
6 comp.windows.x
7 misc.forsale
8 rec.autos
9 rec.motorcycles
10 rec.sport.baseball
11 rec.sport.hockey
12 sci.crypt
13 sci.electronics
14 sci.med
15 sci.space
16 soc.religion.christian
17 talk.politics.guns
18 talk.politics.mideast
19 talk.politics.misc
20 talk.religion.misc

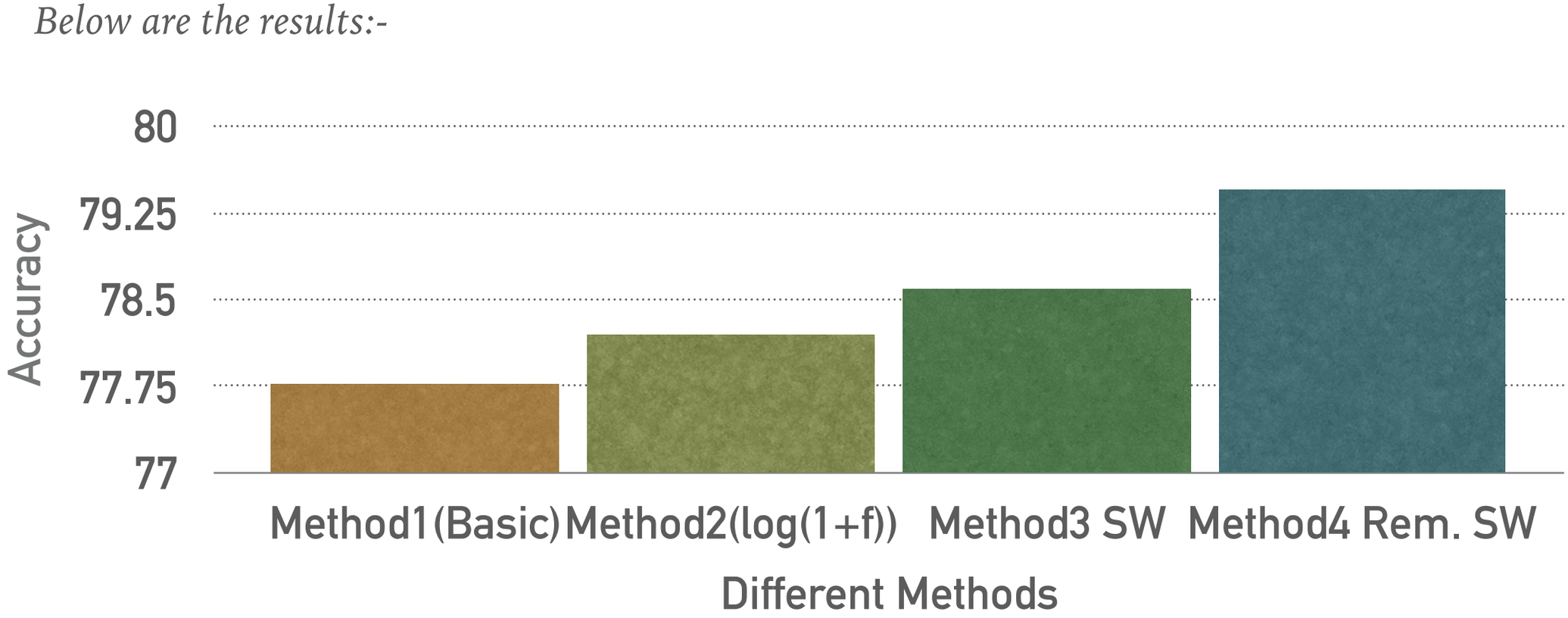
F. CAN WE IMPROVE FURTHER?

That was pretty good for the first model but what can we do to improve further?

I tried below method to improve the model:-

- *If f is the freq. of words, then use log(1+f) instead of f (Method2)*
- *Set the prob. of stop words to default in training data (Method3)*
- *Remove stop words from training and test data (Method4)*

Below are the results:-



APPENDIX

Code to get class prior probabilities

```
pi={}

#0-19 => 1-20
train_map=open("train.map").readlines()
train_map_names=[] #list of class names
for cmap in train_map:
    i=cmap.split()[0]
    train_map_names.append(i)
    j=int(cmap.rstrip().split()[1])
    pi[j]=0

train_label=open("train.label").readlines()

tot=len(train_label)

label = []
for line in train_label:
    label.append(int(line.split()[0]))
    l=int(line.split()[0])
    pi[l]+=1

for i in pi:
    pi[i]=float(pi[i])/tot

print ("Percentage of documents in each class are:")
for k,v in pi.items():
    print k,((pi[k])*100)
```

Code to get train data and test data

```
train = open("train.data")
train_data=pd.read_csv(train, delimiter=' ',names=['docid','wordid','count'])

train_data.head()
```

Add class label to train

```
class_label = []
j=0
for i in range(len(train_data)-1):
    class_label.append(label[j])
    if train_data['docid'][i] != train_data['docid'][i+1]:
        j+=1
class_label.append(label[j])
```

Calculate the total words by class and word

```
p_j = train_data.groupby(['classid'])
p_j[‘count'].sum()
p_ij = train_data.groupby(['classid','wordid'])
p_ij['count'].sum()
```

Probability of each word in each class without Laplace smoothing

```
P_jw = p_ij['count'].sum()/p_j['count'].sum()
P_unstack = P_jw.unstack()
```

Probability of each word in each class with Laplace smoothing

```
a=.005 #alpha
P_jw_ws = ((p_ij['count'].sum()+a)/(p_j['count'].sum()+61188))
P_unstack_ws = P_jw_ws.unstack()
```

Function to segregate each doc and send to the predictor¶

```
def call_predict(test_dict):
    #get the start time for prediction
    ts = time.time()
    st1 = datetime.datetime.fromtimestamp(ts).strftime('%Y-%m-%d %H:%M:%S')
    print st1
    doc_num = len(test_data.docid.unique())
    prediction={}
    for i in range(1,doc_num+1):
        cls={}
        l={}
        l=test_dict[i]
        cls=predict_bayes(i,l)
        prediction[i]=cls[1]
    ts = time.time()
st = datetime.datetime.fromtimestamp(ts).strftime('%Y-%m-%d %H:%M:%S')
print st
return prediction
```

Function to predict the class for each document

```
def predict_bayes(docid,test_doc):
    j=1
    p_jw_dict = {}
    for j in range(1,21):
        p_d_c =0
        p_f_log=0
        p_f=0
        p=0
        for i in test_doc.keys(): #i is the word id
            cnt=test_doc[i] #freq of word in doc
            try:
                p = P_unstack_ws[i][j]   #Prob of word in class j
            except:
                p = a/(float(p_j['count'].sum()[j]+61188)) #Set to default if word doesn't exist
            #p_f = cnt * np.log(p)
            p_f = np.log(1+cnt) * np.log(p) #change word freq to log(1+f)
            p_d_c +=p_f #add up log values for class j
        p_d_c += np.log(pi[j])
        p_jw_dict[docid,j]=p_d_c
    return max(p_jw_dict, key=p_jw_dict.get)
```

Function to remove stop words

```
def remove_stop_words(data,stop_words,vocab,del_flag):

    #create a vocab df with vocab index as in train and test data
    vocab_df = pd.DataFrame(vocab,columns=['words'])
    vocab_df=vocab_df.set_index(vocab_df.index + 1)

    #get the list of bad words in vocab list
    bad_words=[]
    list3 = set(vocab)&set(stop_words) # we don't need to list3 to actually be a list
    bad_words = sorted(list3, key = lambda k : vocab.index(k))

    #get the indices of all the bad words
    bad_index = [vocab_df.index[vocab_df['words']==i][0] for i in bad_words]

    #data = [(data.loc[data.wordid == bad_index[i], 'count'] = 1) for i in range(len(bad_index))]

#change count of all words to 1
if del_flag==1:
   for i in range(len(bad_index)):
      data=data[data.wordid != bad_index[i]]
else:
   for i in range(len(bad_index)):
      data.loc[data.wordid == bad_index[i], 'count'] = 1

return data
```

Getting the accuracy

```
import numpy as np
from sklearn.metrics import accuracy_score

accuracy_score(test_label, pred.values())
```

Plot the confusion matrix

```
import pandas as pd
import seaborn as sns

plt.figure(figsize=(11,11))
confusion_matrix = pd.DataFrame(data = cm_normalized)
sns.heatmap(confusion_matrix,annot=True,fmt="d",linewidths=.5, square = True); #cmap = 'Greens_r'
plt.ylabel('Actual label');
plt.xlabel('Predicted label');
plt.title('Confusion Matrix', size = 15);
#tick_marks = np.arange(len(train_map))
#plt.xticks(tick_marks, train_map_names, rotation=90)
#plt.yticks(tick_marks, train_map_names[::-1],rotation=360)
plt.show()
```