COM S 573 - Lab Assignment 1

Submitted by Anjana Deva Prasad

Import Libraries

In [95]:
```
import numpy as np
import pandas as pd
from collections import defaultdict
import math
```

Import Train data

In [96]:
```
train_data=pd.read_csv('train_data.csv', names=['DocId','WordId', 'Count']) #read train_data
train_label=pd.read_csv('train_label.csv', names=['Category'])
train_data.index=train_data.index+1
train_label.index=train_label.index+1
```

Import Test data

In [97]:
```
test_data=pd.read_csv('test_data.csv', names=['DocId','WordId', 'Count'])
test_label=pd.read_csv('test_label.csv', names=['Category'])
test_data.index=test_data.index+1
test_label.index=test_label.index+1
```

Import Vocabulary and Maps

In [98]:
```
maps=pd.read_csv('map.csv', names=['Category','Category Name'])
maps.index=maps.index+1
vocabulary= pd.read_csv('vocabulary.txt', sep=" ", names=['Words'])
vocabulary.insert(0, 'WordId', range(1, 1 + len(vocabulary)))
vocabulary.index=vocabulary.index+1
V=len(vocabulary)
print(len(vocabulary))
```

```
61188
```

Train Data

In [99]:
```
print(train_data)
```

```
         DocId  WordId  Count
1            1       1      4
2            1       2      2
3            1       3     10
4            1       4      4
5            1       5      2
...        ...     ...    ...
1467341  11269   47387      1
1467342  11269   48339      1
1467343  11269   48919      1
1467344  11269   51544      1
1467345  11269   53958      1

[1467345 rows x 3 columns]
```

Test Data

In [100]:
```
print(test_data)
```

```
        DocId  WordId  Count
1           1       3      1
2           1      10      1
3           1      12      8
4           1      17      1
5           1      23      8
...       ...     ...    ...
967870   7505   44515      1
967871   7505   47720      1
967872   7505   50324      1
967873   7505   59935      1
967874   7505   61188      2

[967874 rows x 3 columns]
```

Train Labels

In [101]:
```
print(train_label)
```

```
       Category
1             1
2             1
3             1
4             1
5             1
...         ...
11265        20
11266        20
11267        20
11268        20
11269        20

[11269 rows x 1 columns]
```

Test Labels

In [102]:
```
print(test_label)
```

```
      Category
1            1
2            1
3            1
4            1
5            1
...        ...
7501        20
7502        20
7503        20
7504        20
7505        20

[7505 rows x 1 columns]
```

Calculate Prior Probabilities

In [103]:
```
def calculate_P_Omega_j(train_label):
    p_wj=dict()
    for i in range(1,21):
        p_wj[i]=0
    train_label=np.array(train_label)
    print(train_label.shape)
    for i in range(train_label.shape[0]):
        val=int(train_label[i])
        p_wj[val]+=1

    for i in range(1,21):
        p_wj[i]/=train_label.shape[0]

    return dict(p_wj)
```

2.1 a. Prior Probabilities

In [104]:
```
P_wj=calculate_P_Omega_j(train_label)
P_wj
```

```
(11269, 1)
```

Out[104]:
```
{1: 0.04259472890229834,
 2: 0.05155736977549028,
 3: 0.05075871860857219,
 4: 0.05208980388676901,
 5: 0.051024935664211554,
 6: 0.052533498979501284,
 7: 0.051646108794036735,
 8: 0.052533498979501284,
 9: 0.052888455053687104,
 10: 0.0527109770165942,
 11: 0.05306593309078002,
 12: 0.0527109770165942,
 13: 0.05244475996095483,
 14: 0.0527109770165942,
 15: 0.052622237998047744,
 16: 0.05315467210932647,
 17: 0.04836276510781791,
 18: 0.05004880646020055,
 19: 0.04117490460555506,
 20: 0.033365870973467035}
```

In [105]:
```
train_label.insert(0, 'DocId', range(1, 1 + len(train_label))) #Insert DocId to train labels
test_label.insert(0, 'DocId', range(1, 1 + len(test_label)))   #Insert DocId to test labels

print(train_label)
print(test_label)
```

```
       DocId  Category
1          1         1
2          2         1
3          3         1
4          4         1
5          5         1
...      ...       ...
11265  11265        20
11266  11266        20
11267  11267        20
11268  11268        20
11269  11269        20

[11269 rows x 2 columns]
      DocId  Category
1         1         1
2         2         1
3         3         1
4         4         1
5         5         1
...     ...       ...
7501   7501        20
7502   7502        20
7503   7503        20
7504   7504        20
7505   7505        20

[7505 rows x 2 columns]
```

In [106]:
```
result = pd.merge(train_label, train_data, on='DocId')   #Merge train_data and train_label
result.index=result.index+1
result
```

Out[106]:

	DocId	Category	WordId	Count
1	1	1	1	4
2	1	1	2	2
3	1	1	3	10
4	1	1	4	4
5	1	1	5	2
...	...	...	...	...
1467341	11269	20	47387	1
1467342	11269	20	48339	1
1467343	11269	20	48919	1
1467344	11269	20	51544	1
1467345	11269	20	53958	1

1467345 rows × 4 columns

In [107]:
```
vocabulary
```

Out[107]:

	WordId	Words
1	1	archive
2	2	name
3	3	atheism
4	4	resources
5	5	alt
...	...	...
61184	61184	aeroplane
61185	61185	gosple
61186	61186	ephas
61187	61187	kltensme
61188	61188	etrbom

61188 rows × 2 columns

In [108]:
```
result1=result

result1 = pd.merge(vocabulary, result, on='WordId', how='outer')
result1=result1.fillna(0)
result1
```

Out[108]:

	WordId	Words	DocId	Category	Count
0	1	archive	1.0	1.0	4.0
1	1	archive	47.0	1.0	2.0
2	1	archive	196.0	1.0	3.0
3	1	archive	432.0	1.0	2.0
4	1	archive	433.0	1.0	2.0
...	...	...	...	...	...
1474553	61184	aeroplane	0.0	0.0	0.0
1474554	61185	gosple	0.0	0.0	0.0
1474555	61186	ephas	0.0	0.0	0.0
1474556	61187	kltensme	0.0	0.0	0.0
1474557	61188	etrbom	0.0	0.0	0.0

1474558 rows × 5 columns

In [109]:
```
p_ij = result1.groupby(['WordId','Category'])
p_j = result1.groupby(['Category'])
```

2.1 b,c Calculate n and n_k

In [110]:
```
n_k=p_ij['Count'].sum()
n=p_j['Count'].sum()


n_k=n_k.unstack()
n_k=n_k.fillna(0)  #Fill all the NaN terms with 0

n_k=n_k.drop(n_k.columns[0], axis=1)
n=n.drop(n.index[0])
```

n values:

In [111]:
```
n
```

Out[111]:
```
Category
1.0     148812.0
2.0     110358.0
3.0      90767.0
4.0      99146.0
5.0      86190.0
6.0     152846.0
7.0      61094.0
8.0     114102.0
9.0     102631.0
10.0    107898.0
11.0    141267.0
12.0    200456.0
13.0    103173.0
14.0    155338.0
15.0    153714.0
16.0    201267.0
17.0    175914.0
18.0    254805.0
19.0    186426.0
20.0    119096.0
Name: Count, dtype: float64
```

n_k values

In [112]:
```
n_k
```

Out[112]:

Category	1.0	2.0	3.0	4.0	5.0	6.0	7.0	8.0	9.0	10.0	11.0	12.0	13.0	14.0	15.0	16.0	17.0	18.0	19.0	20.0
WordId																				
1	13.0	60.0	11.0	8.0	6.0	47.0	0.0	9.0	14.0	1.0	1.0	52.0	3.0	15.0	48.0	0.0	19.0	10.0	0.0	0.0
2	63.0	59.0	69.0	31.0	33.0	222.0	28.0	54.0	67.0	33.0	67.0	90.0	33.0	39.0	82.0	123.0	33.0	154.0	39.0	45.0
3	275.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	16.0	0.0	0.0	0.0	9.0
4	9.0	17.0	17.0	0.0	1.0	79.0	2.0	0.0	4.0	2.0	0.0	11.0	2.0	13.0	22.0	7.0	6.0	9.0	23.0	2.0
5	82.0	14.0	21.0	10.0	1.0	15.0	2.0	13.0	4.0	1.0	1.0	59.0	5.0	20.0	12.0	14.0	11.0	2.0	17.0	23.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
61184	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
61185	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
61186	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
61187	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
61188	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

61188 rows × 20 columns

Calculate Probabilities P_{BE} and P_{MLE}

In [113]:
```
P_le=n_k.divide(n)
P_be=  (n_k+1) / (n+V)
```

Observations made on P_{BE} and P_{MLE}

P_{BE} values never become zero but P_{MLE} values have lots of zeros. This is because for BE there is an n_k +1 term that takes care of never encountering any zero probabilities.

P_{be}

In [114]:
```
P_be
```

Out[114]:

Category	1.0	2.0	3.0	4.0	5.0	6.0	7.0	8.0	9.0	10.0	11.0	12.0	13.0	
WordId														
1	0.000067	0.000356	0.000079	0.000056	0.000047	0.000224	0.000008	0.000057	0.000092	0.000012	0.000010	0.000203	0.000024	0.
2	0.000305	0.000350	0.000461	0.000200	0.000231	0.001042	0.000237	0.000314	0.000415	0.000201	0.000336	0.000348	0.000207	0.
3	0.001314	0.000006	0.000007	0.000006	0.000007	0.000005	0.000008	0.000006	0.000006	0.000006	0.000005	0.000004	0.000006	0.
4	0.000048	0.000105	0.000118	0.000006	0.000014	0.000374	0.000025	0.000006	0.000031	0.000018	0.000005	0.000046	0.000018	0.
5	0.000395	0.000087	0.000145	0.000069	0.000014	0.000075	0.000025	0.000080	0.000031	0.000012	0.000010	0.000229	0.000037	0.
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
61184	0.000005	0.000006	0.000007	0.000006	0.000007	0.000005	0.000008	0.000006	0.000006	0.000006	0.000005	0.000004	0.000006	0.
61185	0.000005	0.000006	0.000007	0.000006	0.000007	0.000005	0.000008	0.000006	0.000006	0.000006	0.000005	0.000004	0.000006	0.
61186	0.000005	0.000006	0.000007	0.000006	0.000007	0.000005	0.000008	0.000006	0.000006	0.000006	0.000005	0.000004	0.000006	0.
61187	0.000005	0.000006	0.000007	0.000006	0.000007	0.000005	0.000008	0.000006	0.000006	0.000006	0.000005	0.000004	0.000006	0.
61188	0.000005	0.000006	0.000007	0.000006	0.000007	0.000005	0.000008	0.000006	0.000006	0.000006	0.000005	0.000004	0.000006	0.

61188 rows × 20 columns

P_{mle}

In [115]:
```
P_le
```

Out[115]:

Category	1.0	2.0	3.0	4.0	5.0	6.0	7.0	8.0	9.0	10.0	11.0	12.0	13.0	
WordId														
1	0.000087	0.000544	0.000121	0.000081	0.000070	0.000307	0.000000	0.000079	0.000136	0.000009	0.000007	0.000259	0.000029	0.
2	0.000423	0.000535	0.000760	0.000313	0.000383	0.001452	0.000458	0.000473	0.000653	0.000306	0.000474	0.000449	0.000320	0.
3	0.001848	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.
4	0.000060	0.000154	0.000187	0.000000	0.000012	0.000517	0.000033	0.000000	0.000039	0.000019	0.000000	0.000055	0.000019	0.
5	0.000551	0.000127	0.000231	0.000101	0.000012	0.000098	0.000033	0.000114	0.000039	0.000009	0.000007	0.000294	0.000048	0.
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
61184	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.
61185	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.
61186	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.
61187	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.
61188	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.

61188 rows × 20 columns

In [116]:
```
P_le=P_le.to_dict()  #Converting Probabilities to dictionaries to make them faster
P_be=P_be.to_dict()
```

2.2 Evaluate the Performance of your Classifier

In [117]:

```
def BE(data,P_be,P_wj,V):
    print(data)
    df_dict=data.to_dict()
    new_dict=defaultdict(dict)


    for index in range(1,len(df_dict['DocId'])+1):
        docId=df_dict['DocId'][index]
        wordId=df_dict['WordId'][index]
        count=df_dict['Count'][index]
        new_dict[docId][wordId]=count

    new_dict=dict(new_dict)


    predictions=[]
    for docId in range(1,len(new_dict)+1):
        score=[]
        for category in range(1,21):
            val=0
            for wordId in new_dict[docId]:

                Prob=P_be[category][wordId]
                try:
                    power=new_dict[docId][wordId]
                except:
                    power=0
                log_prob=(np.log(Prob))
                val=val+log_prob*power



            category_prob= np.log(P_wj[category])
            score.append(category_prob+val)

        prediction = np.argmax(score, axis=0)+1
        predictions.append(prediction)

    return predictions
```

```
def MLE(data,P_be,P_wj,V):
    print(data)
    df_dict=data.to_dict()
    new_dict=defaultdict(dict)



    for index in range(1,len(df_dict['DocId'])+1):
        docId=df_dict['DocId'][index]
        wordId=df_dict['WordId'][index]
        count=df_dict['Count'][index]
        new_dict[docId][wordId]=count

    new_dict=dict(new_dict)


    predictions=[]
    for docId in range(1,len(new_dict)+1):
        score=[]
        for category in range(1,21):
            val=0
            for wordId in new_dict[docId]:


                Prob=P_be[category][wordId]
                if(Prob==0):
                    val=-math.inf             #If the Probability is zero, then a log cannot be taken. So
ignore.
                    continue

                power=new_dict[docId][wordId]
                    #except:
                        #power=0
                log_prob=(np.log(Prob))
                val=val+log_prob*power
            category_prob= np.log(P_wj[category])
            score.append(category_prob+val)

        prediction = np.argmax(score, axis=0)+1
```

```
        predictions.append(prediction)

    return predictions


def CalcAccuracy(predict_list,file):

    df = pd.read_csv(file, names = ['Category'])
    df.index = df.index+1
    df['Predicted'] = pd.Series(predict_list, index = df.index)
    match = df[df['Category'] == df['Predicted']]
    correct = match.shape[0]
    return (correct/df.shape[0])

def CalcGroupAccuracy(predict_list,file,i):
#     Read training Label
    df = pd.read_csv(file, names = ['Category'])
    df.index = df.index+1
    df['Predicted'] = pd.Series(predict_list, index = df.index)
    df=df[df['Category']==i]
    match = df[df['Category'] == df['Predicted']]
    correct = match.shape[0]
    return (correct/df.shape[0])

def ConfusionMatrix(predict_list,file):
#     Read training Label
    confusion_matrix=[[0]*21]*21
    count=0
    df = pd.read_csv(file, names = ['Category'])
    df.index = df.index+1
    df['Predicted'] = pd.Series(predict_list, index = df.index)
    for i in range(1,21):
        row=df[ df['Category']==i]
        for j in range(1,21):
            match = row[row['Predicted']==j]
            count=match.shape[0]
            confusion_matrix[i][j]=count
        print(confusion_matrix[i][1:])
```

2.2.1 Performance on Training Data

Naive Bayes on train data for BE

In [118]:
```
predictions_train_be=BE(train_data,P_be,P_wj,V)
```

```
         DocId  WordId  Count
1            1       1      4
2            1       2      2
3            1       3     10
4            1       4      4
5            1       5      2
...        ...     ...    ...
1467341  11269   47387      1
1467342  11269   48339      1
1467343  11269   48919      1
1467344  11269   51544      1
1467345  11269   53958      1

[1467345 rows x 3 columns]
```

Naive Bayes on train data for MLE

In [119]:
```
predictions_train_mle=MLE(train_data,P_le,P_wj,V)
```

```
          DocId  WordId  Count
1             1       1      4
2             1       2      2
3             1       3     10
4             1       4      4
5             1       5      2
...         ...     ...    ...
1467341   11269   47387      1
1467342   11269   48339      1
1467343   11269   48919      1
1467344   11269   51544      1
1467345   11269   53958      1

[1467345 rows x 3 columns]
```

Accuracy for BE on training data

In [120]:
```
accuracy_train_BE = CalcAccuracy(predictions_train_be,'train_label.csv')
print(accuracy_train_BE*100)
```

```
94.10772916851539
```

Accuracy for MLE on training data

In [121]:
```
accuracy_train_MLE = CalcAccuracy(predictions_train_mle,'train_label.csv')
print(accuracy_train_MLE*100)
```

```
99.1214837163901
```

Class accuracy for BE

In [122]:
```
for i in range(1,21):
    group_accuracy=CalcGroupAccuracy(predictions_train_be,'train_label.csv',i)
    print(i," : ",group_accuracy)
```

```
1  :  0.9666666666666667
2  :  0.919104991394148
3  :  0.8793706293706294
4  :  0.9301533219761499
5  :  0.9408695652173913
6  :  0.9493243243243243
7  :  0.7749140893470791
8  :  0.9662162162162162
9  :  0.9630872483221476
10  :  0.9713804713804713
11  :  0.9782608695652174
12  :  0.9797979797979798
13  :  0.9238578680203046
14  :  0.9764309764309764
15  :  0.9780775716694773
16  :  0.9833055091819699
17  :  0.9853211009174312
18  :  0.9680851063829787
19  :  0.9698275862068966
20  :  0.7606382978723404
```

Class Accuracy for MLE

In [123]:
```
for i in range(1,21):
    group_accuracy=CalcGroupAccuracy(predictions_train_mle,'train_label.csv',i)
    print(i," : ",group_accuracy)
```

```
1  :  0.9979166666666667
2  :  0.9793459552495697
3  :  0.9912587412587412
4  :  0.9880749574105622
5  :  0.9895652173913043
6  :  0.9847972972972973
7  :  0.993127147766323
8  :  0.9915540540540541
9  :  0.9966442953020134
10  :  0.9932659932659933
11  :  0.9899665551839465
12  :  1.0
13  :  0.9898477157360406
14  :  0.9966329966329966
15  :  0.9966273187183811
16  :  0.988313856427379
17  :  0.9963302752293578
18  :  0.9911347517730497
19  :  0.9870689655172413
20  :  0.9787234042553191
```

Confusion Matrix for BE

In [124]:
```
ConfusionMatrix(predictions_train_be,'train_label.csv')
```

```
[464, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 11, 0, 1, 1, 2]
[1, 534, 6, 15, 1, 9, 2, 0, 1, 0, 0, 2, 1, 1, 2, 4, 0, 0, 2, 0]
[1, 10, 503, 23, 1, 20, 2, 0, 0, 0, 0, 7, 1, 1, 0, 2, 0, 0, 1, 0]
[0, 10, 4, 546, 4, 4, 6, 2, 0, 0, 0, 0, 3, 0, 1, 2, 0, 2, 2, 1]
[2, 5, 2, 7, 541, 3, 1, 0, 2, 0, 0, 2, 1, 2, 2, 3, 0, 1, 1, 0]
[0, 11, 8, 1, 2, 562, 0, 0, 1, 1, 0, 2, 0, 1, 1, 0, 1, 0, 1, 0]
[2, 3, 2, 34, 6, 2, 451, 17, 1, 3, 3, 16, 15, 5, 4, 5, 5, 1, 7, 0]
[1, 0, 0, 3, 1, 2, 3, 572, 1, 1, 0, 1, 0, 0, 0, 1, 1, 1, 3, 1]
[0, 1, 0, 1, 1, 0, 4, 1, 574, 0, 0, 0, 0, 2, 0, 2, 6, 1, 3, 0]
[0, 3, 0, 1, 0, 1, 1, 3, 0, 577, 4, 0, 0, 1, 0, 1, 2, 0, 0, 0]
[1, 0, 1, 2, 0, 1, 0, 2, 0, 0, 585, 1, 0, 0, 0, 1, 0, 2, 2, 0]
[0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 582, 0, 1, 0, 0, 3, 1, 5, 0]
[1, 4, 0, 15, 5, 0, 3, 2, 0, 0, 1, 5, 546, 2, 2, 1, 2, 0, 2, 0]
[0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 2, 580, 0, 5, 2, 0, 1, 0]
[2, 2, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 2, 580, 1, 0, 0, 2, 0]
[0, 0, 0, 2, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 589, 1, 3, 2, 0]
[0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 2, 537, 2, 3, 0]
[1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 2, 0, 6, 0, 546, 5, 0]
[2, 2, 0, 0, 0, 0, 0, 0, 0, 1, 0, 3, 0, 1, 0, 1, 2, 2, 450, 0]
[25, 0, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 1, 39, 15, 4, 4, 286]
```

Confusion Matrix for MLE

In [125]:
```
ConfusionMatrix(predictions_train_mle,'train_label.csv')
```

```
[479, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1]
[0, 569, 2, 4, 1, 1, 3, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 4, 567, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 2, 1, 580, 1, 1, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[1, 0, 0, 1, 569, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0]
[0, 5, 2, 0, 0, 583, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 1, 578, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 1, 0, 0, 1, 587, 2, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 1, 0, 1, 0, 594, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 1, 0, 0, 590, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 1, 0, 0, 0, 2, 0, 1, 592, 1, 0, 0, 0, 1, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 594, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 1, 2, 0, 2, 0, 0, 0, 0, 0, 585, 1, 0, 0, 0, 0, 0, 0]
[1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 592, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 591, 0, 0, 0, 0, 0]
[1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 592, 1, 0, 0, 3]
[0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 543, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 2, 0, 559, 1, 0]
[1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 458, 1]
[2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 4, 1, 0, 1, 368]
```

2.2.2 Performance on testing data

Naive Bayes on test data for BE

In [126]:
```
predictions_test_be=BE(test_data,P_be,P_wj,V)
```

```
        DocId  WordId  Count
1           1       3      1
2           1      10      1
3           1      12      8
4           1      17      1
5           1      23      8
...       ...     ...    ...
967870   7505   44515      1
967871   7505   47720      1
967872   7505   50324      1
967873   7505   59935      1
967874   7505   61188      2

[967874 rows x 3 columns]
```

Naive Bayes on test data for MLE

In [127]:
```
predictions_test_mle=MLE(test_data,P_le,P_wj,V)
```

```
        DocId  WordId  Count
1           1       3      1
2           1      10      1
3           1      12      8
4           1      17      1
5           1      23      8
...       ...     ...    ...
967870   7505   44515      1
967871   7505   47720      1
967872   7505   50324      1
967873   7505   59935      1
967874   7505   61188      2

[967874 rows x 3 columns]
```

Accuracy for BE on test data

In [128]:
```
accuracy_test_BE = CalcAccuracy(predictions_test_be,'test_label.csv')
print(accuracy_test_BE*100)
```

```
78.10792804796802
```

Accuracy for MLE on test data

In [129]:
```
accuracy_test_MLE = CalcAccuracy(predictions_test_mle,'test_label.csv')
print(accuracy_test_MLE*100)
```

```
9.460359760159893
```

Class Accuracy for BE

In [130]:
```
for i in range(1,21):
    group_accuracy=CalcGroupAccuracy(predictions_test_be,'test_label.csv',i)
    print(i," : ",group_accuracy*100)
```

```
1  :  73.89937106918238
2  :  76.09254498714652
3  :  52.94117647058824
4  :  77.8061224489796
5  :  71.27937336814621
6  :  78.46153846153847
7  :  59.16230366492147
8  :  90.12658227848101
9  :  88.9168765743073
10  :  86.90176322418137
11  :  95.48872180451127
12  :  91.39240506329114
13  :  65.9033078880407
14  :  82.44274809160305
15  :  85.45918367346938
16  :  94.72361809045226
17  :  89.28571428571429
18  :  86.43617021276596
19  :  59.354838709677416
20  :  35.45816733067729
```

Class Accuracy for MLE

In [131]:
```
for i in range(1,21):
    group_accuracy=CalcGroupAccuracy(predictions_test_mle,'test_label.csv',i)
    print(i," : ",group_accuracy)
```

```
1  :  0.9937106918238994
2  :  0.06683804627249357
3  :  0.04859335038363171
4  :  0.07142857142857142
5  :  0.057441253263707574
6  :  0.08461538461538462
7  :  0.12041884816753927
8  :  0.04810126582278481
9  :  0.05037783375314862
10  :  0.05289672544080604
11  :  0.09022556390977443
12  :  0.043037974683544304
13  :  0.020356234096692113
14  :  0.035623409669211195
15  :  0.04336734693877551
16  :  0.0678391959798995
17  :  0.03296703296703297
18  :  0.0398936170212766
19  :  0.025806451612903226
20  :  0.02390438247011952
```

Confusion Matrix for BE

In [132]:
```
ConfusionMatrix(predictions_test_be,'test_label.csv')
```

```
[235, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 2, 3, 45, 3, 10, 7, 9]
[3, 296, 6, 12, 7, 22, 1, 3, 2, 0, 0, 17, 4, 4, 7, 4, 0, 0, 1, 0]
[3, 33, 207, 58, 11, 31, 0, 2, 2, 2, 1, 17, 1, 4, 4, 5, 0, 0, 9, 1]
[0, 8, 15, 305, 21, 2, 4, 6, 0, 0, 1, 6, 23, 0, 1, 0, 0, 0, 0, 0]
[0, 8, 10, 37, 273, 3, 4, 4, 1, 1, 0, 6, 17, 8, 2, 0, 3, 0, 6, 0]
[0, 42, 7, 10, 2, 306, 1, 0, 2, 1, 0, 10, 0, 0, 3, 2, 1, 1, 2, 0]
[0, 8, 4, 50, 20, 1, 226, 33, 5, 0, 1, 3, 11, 2, 3, 4, 2, 3, 6, 0]
[1, 1, 0, 2, 0, 1, 5, 356, 4, 2, 0, 1, 4, 0, 2, 1, 4, 2, 9, 0]
[0, 1, 0, 0, 0, 0, 0, 26, 353, 2, 0, 1, 1, 1, 0, 1, 4, 2, 5, 0]
[4, 1, 0, 1, 1, 2, 3, 3, 1, 345, 17, 2, 2, 0, 0, 3, 1, 2, 9, 0]
[2, 0, 0, 0, 0, 0, 1, 1, 0, 4, 381, 1, 0, 2, 1, 2, 0, 1, 3, 0]
[0, 4, 1, 1, 2, 1, 1, 0, 0, 0, 0, 361, 3, 2, 0, 2, 8, 0, 8, 1]
[2, 18, 0, 27, 8, 3, 1, 10, 2, 0, 0, 46, 259, 6, 3, 6, 0, 2, 0, 0]
[10, 7, 1, 3, 0, 0, 0, 4, 0, 1, 0, 1, 3, 324, 3, 17, 3, 6, 10, 0]
[3, 7, 0, 0, 0, 2, 0, 0, 1, 0, 1, 4, 4, 4, 335, 5, 1, 2, 22, 1]
[7, 2, 1, 0, 1, 2, 0, 0, 0, 0, 0, 1, 0, 1, 0, 377, 2, 2, 1, 1]
[1, 0, 0, 0, 1, 0, 1, 2, 1, 1, 1, 3, 0, 1, 2, 3, 325, 2, 16, 4]
[12, 1, 0, 0, 0, 0, 0, 2, 1, 1, 1, 4, 0, 0, 0, 8, 3, 325, 18, 0]
[6, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 3, 0, 3, 7, 3, 95, 5, 184, 1]
[47, 3, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 3, 5, 70, 19, 5, 8, 89]
```

Confusion Matrix for MLE

In [133]:
```
ConfusionMatrix(predictions_test_mle,'test_label.csv')
```

```
[316, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1]
[352, 26, 2, 1, 1, 4, 0, 0, 0, 0, 0, 0, 1, 0, 2, 0, 0, 0, 0, 0]
[354, 3, 19, 6, 3, 1, 0, 0, 0, 0, 0, 1, 1, 2, 0, 0, 1, 0, 0, 0]
[350, 3, 5, 28, 2, 0, 4, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[351, 2, 0, 2, 22, 0, 1, 0, 0, 0, 0, 0, 4, 1, 0, 0, 0, 0, 0, 0]
[351, 2, 0, 1, 1, 33, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0]
[317, 4, 2, 4, 1, 0, 46, 2, 2, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1]
[369, 0, 0, 0, 0, 0, 1, 19, 2, 0, 0, 1, 2, 0, 0, 0, 0, 0, 0, 1]
[376, 0, 0, 0, 0, 0, 0, 1, 20, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[372, 0, 0, 0, 0, 0, 1, 0, 0, 21, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[363, 0, 0, 0, 0, 0, 0, 0, 0, 0, 36, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[375, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 17, 1, 0, 0, 0, 0, 0, 0, 0]
[373, 1, 0, 1, 0, 1, 3, 1, 0, 0, 0, 3, 8, 1, 1, 0, 0, 0, 0, 0]
[375, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 14, 0, 0, 1, 0, 0, 0]
[375, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 17, 0, 0, 0, 0, 0]
[367, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 27, 0, 1, 1, 1]
[350, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 12, 1, 0, 1]
[358, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 15, 0, 0]
[297, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 2, 0, 8, 0]
[237, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 4, 0, 0, 2, 6]
```

Observations made on training data

- Bayesian Estimate works well and predicts the training data with a 94.1% accuracy.
- Maximum Likelihood works very well on the training data with accuracy as high as 99.1%
- For the training data, Maximum Likelihood Estimate performs better than Bayesian Estimate
- Based on observations made on the Class accuracy values, BE does well in most classes with an accuracy of greater than 90% but does poorly on classes 7,20 and 3 with resp accuracies below 80% for the 7,20 and around 87% for class 3.
- Based on observations made on the Class accuracy values, MLE performs exceedingly well for Class 12 with a 100% class accuracy and has 97% accuracy and more for all the other classes.
- The Confusion Matrix of MLE is more populated than BE along the major diagonal. This means that MLE predicts higher classes correctly, which explains why the accuracy of MLE is more than that of BE.

Observations made on test data

- BE performs reasonably well on test data as well with an accuracy of 78%.
- MLE performs poorly on test data with an accuracy of around 9.5%
- BE performs better than MLE on test data.
- Based on the class accuracy values, it is found that BE performs better than 75%(approx) because of higher accuracy among class predictions with 90% for most of them and some around 60%. Only class 20 has poor performance with hust 35% accuracy.
- MLE performs well with a 99% accuracy for the first Group and poorly for every other class. This means that the only Class 1 is identified correctly by MLE and most other documents are also identified as class 1.
- The confusion matrix of BE is densely filled on the major diagonal but it still has inconsistencies all over the matrix. This might explain the 74% accuracy.

Conclusion

For this dataset, Bayesian Estimation has a better overall performance than Maximum Likelihood Estimation because MLE overfits for the training data and has a very poor performance on the test data.