# IR Project

May 26, 2018

### 0.0.1 Library imports for the code

```
In [1]: import numpy as np
        import scipy.io as sio
        import matplotlib.pyplot as plt
        import warnings
        import os, re
        from sklearn.feature_extraction.text import CountVectorizer, TfidfTransformer
        from sklearn.model_selection import train_test_split
        from sklearn.naive_bayes import MultinomialNB
        from sklearn.metrics import confusion_matrix, f1_score, accuracy_score
        from sklearn.metrics import precision_score, recall_score, classification_report
        from sklearn.neighbors import NearestCentroid, KNeighborsClassifier
        from datetime import datetime as dt
        %matplotlib inline
        warnings.filterwarnings('ignore')
```

### 0.0.2 Classifier Tester

Will test and show important metrics for any classifier passed to it

```
In [2]: def testClassifier(x_train, y_train, x_test, y_test, clf):
            metrics = []
            start = dt.now()
            clf.fit(x_train, y_train)
            end = dt.now()
            print ('training time: ', (end - start))
            metrics.append(end-start)
            start = dt.now()
            yhat = clf.predict(x_test)
            end = dt.now()
            print ('testing time: ', (end - start))
            metrics.append(end-start)
            print ('classification report: ')
            print(classification_report(y_test, yhat))
            print ('f1 score')
            print (f1_score(y_test, yhat, average='macro'))
            print ('accuracy score')
```

```
    print (accuracy_score(y_test, yhat))
    precision = precision_score(y_test, yhat, average=None)
    recall = recall_score(y_test, yhat, average=None)
    for p, r in zip(precision, recall):
        metrics.append(p)
        metrics.append(r)
    metrics.append(f1_score(y_test, yhat, average='macro'))
    print ('confusion matrix:')
    print (confusion_matrix(y_test, yhat))
    plt.imshow(confusion_matrix(y_test, yhat), interpolation='nearest')
    plt.show()
    return metrics
metrics_dict = []
```

### 0.1 DBWorld Emails Data

The data for this database is already pre-processed and in bag of words format. We just read and split the data into train test sets

```
In [3]: dbworld = 'Datasets/dbworld/MATLAB/dbworld_bodies_stemmed.mat'
        db_world = sio.loadmat(dbworld)
        db_world_inputs = db_world['inputs']
        db_world_labels = db_world['labels'].reshape(len(db_world['labels']),)
        X_train, X_test, y_train, y_test = train_test_split(db_world_inputs,
                                db_world_labels, test_size=0.33, random_state=42)
```

**Naive Bayes** We have chosen MultinomialNB as it gives the best results for Naive Bayes in case of text classification.

```
In [4]: mnb = MultinomialNB()
        mnb_me = testClassifier(X_train, y_train, X_test, y_test, mnb)
        metrics_dict.append({'name':'NaiveBayes', 'metrics':mnb_me})
```

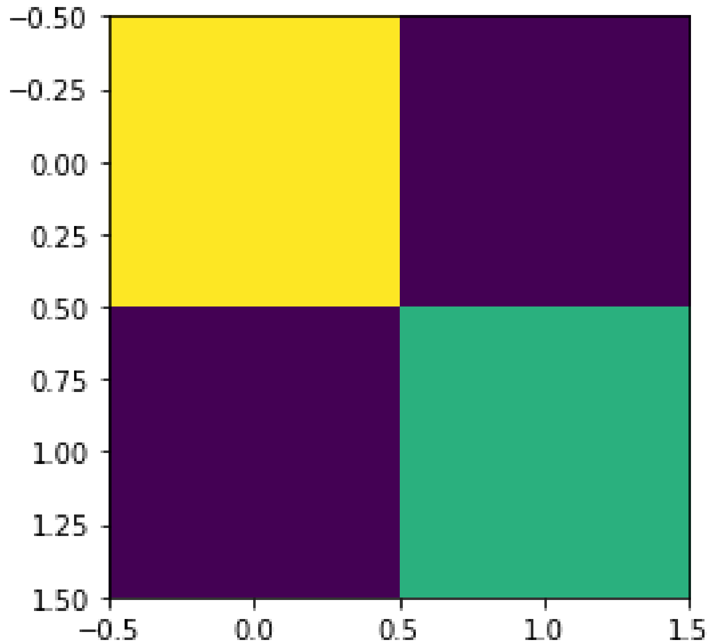
```
training time:  0:00:00.012191
testing time:  0:00:00
classification report:
             precision    recall  f1-score   support

          0       0.92      0.92      0.92        13
          1       0.89      0.89      0.89         9

avg / total       0.91      0.91      0.91        22

f1 score
0.905982905982906
accuracy score
0.9090909090909091
confusion matrix:
```

```
[[12  1]
 [ 1  8]]
```




**Rocchio Classification** For this we will be using the NearestCentroid classifier as when it is used for text classification with tf-idf vectors, this classifier is also known as the Rocchio classifier.

```
In [5]: tfidf = TfidfTransformer()
        tfidf.fit(X_train)
        train_tf = tfidf.transform(X_train)
        test_tf = tfidf.transform(X_test)
        ncr = NearestCentroid()
        ncr_me = testClassifier(train_tf, y_train, test_tf, y_test, ncr)
        metrics_dict.append({'name':'Rocchio', 'metrics':ncr_me})
```

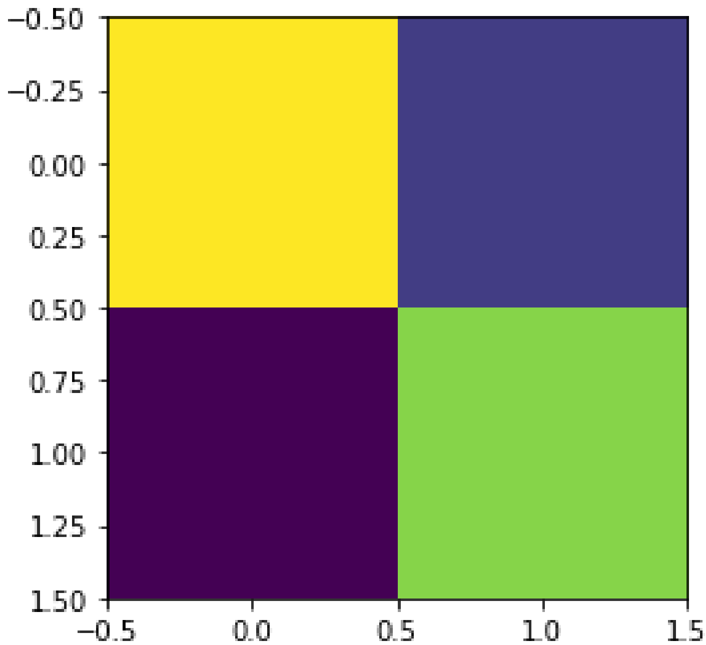
```
training time:  0:00:00.002045
testing time:  0:00:00.000997
classification report:
             precision    recall  f1-score   support

          0       1.00      0.85      0.92        13
          1       0.82      1.00      0.90         9

avg / total       0.93      0.91      0.91        22

f1 score
```

```
0.9083333333333333
accuracy score
0.9090909090909091
confusion matrix:
[[11  2]
 [ 0  9]]
```




**kNN Classification** We'll use kNearestNeighbor for classification now. We tried different values for k and 4 came out to be the best for this.

```
In [6]: knn = KNeighborsClassifier(n_neighbors = 4)
        knn_me = testClassifier(train_tf, y_train, test_tf, y_test, knn)
        metrics_dict.append({'name':'kNN', 'metrics':knn_me})
```

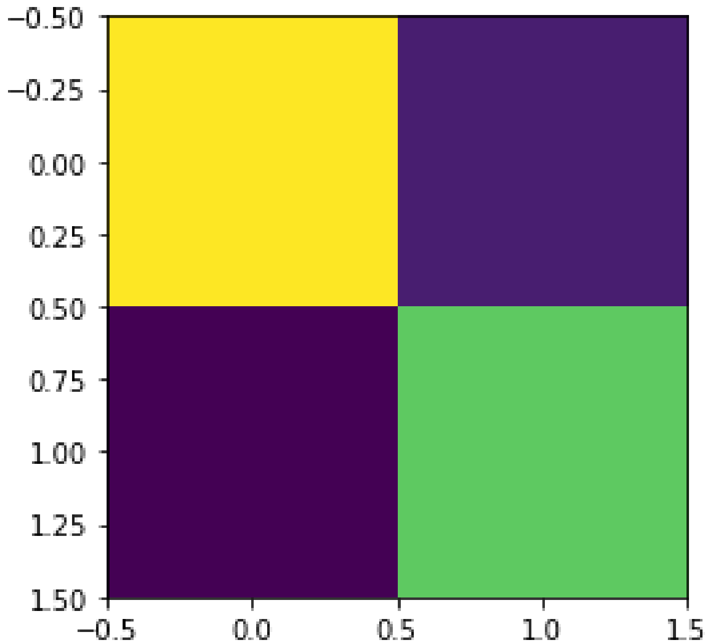
```
training time:  0:00:00
testing time:  0:00:00.003567
classification report:
             precision    recall  f1-score   support

          0       1.00      0.92      0.96        13
          1       0.90      1.00      0.95         9

avg / total       0.96      0.95      0.95        22

f1 score
```

```
0.9536842105263159
accuracy score
0.9545454545454546
confusion matrix:
[[12  1]
 [ 0  9]]
```




**Conclusion** As we can see the kNN classifier with k=4 gives the highest accuracy and f1 score for this document. The training time is negligible but the testing time is the highest among all as it is a known trait of kNN.

## 0.2 Health Tweets

First of all we need to do pre-processing on the data as it is in raw text format. We split the tweets according to the delimeter '|' and clean-up the text.

**Pre-processing** We divide the documents into different classes according to the news agency accounts. The documents are then converted to tf-idf vectors. Further they are split into train test sets.

```
In [7]: health_tweet = os.listdir('Datasets/Health-News-Tweets/Health-Tweets/')
        X_data = []
        y_data = []
        for files in health_tweet:
```

```
            file = open('Datasets/Health-News-Tweets/Health-Tweets/'+files, encoding="utf8")
            data = file.readlines()
            for line in data:
                try:
                    line = re.sub(r"http\S+", "", line.split('|')[2]).lower()
                    X_data.append(line.strip())
                    y_data.append(files.rstrip('.txt'))
                except: pass
            file.close()
        vectorizer = CountVectorizer()
        vectorizer.fit(X_data)
        train_mat = vectorizer.transform(X_data)
        tfidf = TfidfTransformer()
        tfidf.fit(train_mat)
        train_tfmat = tfidf.transform(train_mat)
        X_train, X_test, y_train, y_test = train_test_split(train_tfmat,
                            y_data, test_size=0.33, random_state=42)
```

**Naive Bayes** We have chosen MultinomialNB as it gives the best results for Naive Bayes in case of text classification.

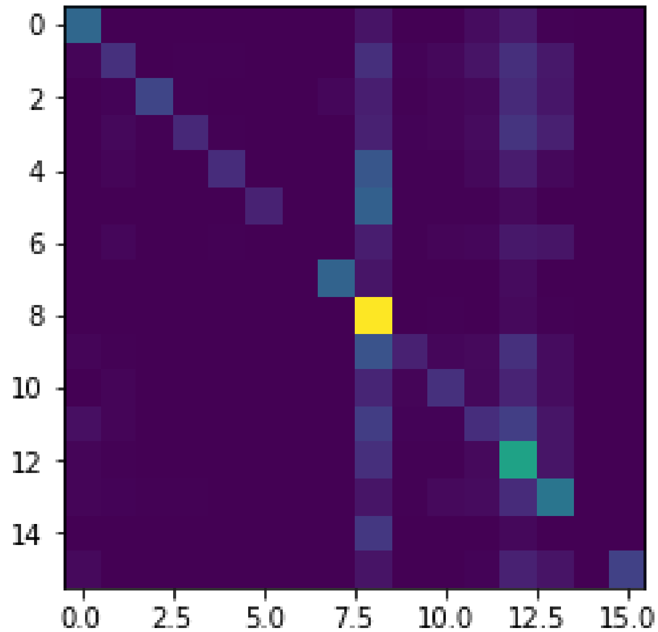
```
In [8]: mnb = MultinomialNB()
        mnb_me = testClassifier(X_train, y_train, X_test, y_test, mnb)
        metrics_dict.append({'name':'NaiveBayes', 'metrics':mnb_me})
```

```
training time:  0:00:00.274875
testing time:  0:00:00.024196
classification report:
                  precision    recall  f1-score   support

KaiserHealthNews       0.71      0.67      0.69      1202
       NBChealth       0.54      0.24      0.33      1419
       bbchealth       0.89      0.38      0.53      1323
       cbchealth       0.78      0.22      0.34      1274
       cnnhealth       0.75      0.23      0.36      1315
  everydayhealth       0.97      0.21      0.35      1083
   foxnewshealth       1.00      0.01      0.01       638
   gdnhealthcare       0.94      0.78      0.85       982
      goodhealth       0.33      0.95      0.49      2527
   latimeshealth       0.56      0.16      0.24      1425
   msnhealthnews       0.50      0.32      0.39      1041
       nprhealth       0.31      0.21      0.25      1534
   nytimeshealth       0.30      0.70      0.42      2005
  reuters_health       0.42      0.59      0.49      1597
    usnewshealth       1.00      0.01      0.02       472
       wsjhealth       0.95      0.44      0.60      1061

     avg / total       0.61      0.45      0.42     20898
```

```
f1 score
0.39768202422709514
accuracy score
0.44573643410852715
confusion matrix:
[[ 805    9    0    1    3    0    0    1  127    2    0   77  162   10
     0    5]
 [  36  335    8   13   16    1    0    0  327   23   53  126  324  156
     0    1]
 [   2   23  505   15    5    0    0   45  203   10   32   53  284  146
     0    0]
 [   4   52   13  280   11    0    0    1  220   22   30   68  362  209
     0    2]
 [   9   33    2    6  306    1    0    0  645   10   13   53  187   49
     0    1]
 [   1    0    0    0    3  231    0    0  744   12   17   10   57    8
     0    0]
 [   0   42    4    7   11    0    4    0  191   15   35   38  151  140
     0    0]
 [   1    0    2    1    3    0    0  767  134    0    0    3   69    1
     0    1]
 [   3    2    2    0    6    0    0    1 2412    6   16    7   58   14
     0    0]
 [  30   18    4    8    9    0    0    0  613  223   39   58  337   84
     0    2]
 [   8   31    4    3    2    1    0    0  254   29  330   56  243   80
     0    0]
 [ 101   31    8    9    8    0    0    1  434   19   21  316  449  136
     0    1]
 [  32   13    3    4    6    2    0    1  329   10   13   62 1394  134
     0    2]
 [  32   27   12   11    5    0    0    2  132   14   58   68  293  936
     0    7]
 [   6    1    0    0    7    3    0    1  389    1    0    3   53    2
     5    1]
 [  63    7    3    2    5    0    0    0  125    5    7   26  222  130
     0  466]]
```




**Rocchio Classification** For this we will be using the NearestCentroid classifier as when it is used for text classification with tf-idf vectors, this classifier is also known as the Rocchio classifier.

```
In [9]: tfidf = TfidfTransformer()
        tfidf.fit(X_train)
        train_tf = tfidf.transform(X_train)
        test_tf = tfidf.transform(X_test)
        ncr = NearestCentroid()
        ncr_me = testClassifier(train_tf, y_train, test_tf, y_test, ncr)
        metrics_dict.append({'name':'Rocchio', 'metrics':ncr_me})
```

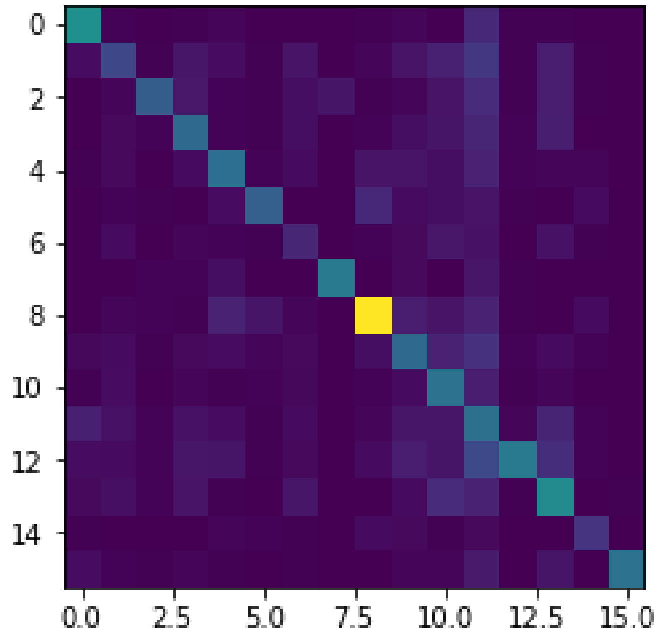
```
training time:  0:00:00.107705
testing time:  0:00:00.028529
classification report:
                  precision    recall  f1-score   support

KaiserHealthNews       0.65      0.72      0.68      1202
       NBChealth       0.40      0.26      0.32      1419
       bbchealth       0.78      0.38      0.51      1323
       cbchealth       0.45      0.47      0.46      1274
       cnnhealth       0.47      0.48      0.47      1315
  everydayhealth       0.70      0.48      0.57      1083
    foxnewshealth       0.24      0.29      0.26       638
   gdnhealthcare       0.85      0.72      0.78       982
      goodhealth       0.75      0.68      0.71      2527
```

```
   latimeshealth       0.38      0.42      0.40      1425
   msnhealthnews       0.33      0.62      0.44      1041
       nprhealth       0.20      0.41      0.27      1534
   nytimeshealth       0.82      0.35      0.49      2005
  reuters_health       0.43      0.52      0.47      1597
    usnewshealth       0.54      0.56      0.55       472
       wsjhealth       0.97      0.61      0.75      1061

     avg / total       0.57      0.50      0.51     20898

f1 score
0.5077887318330322
accuracy score
0.49842090152167673
confusion matrix:
[[ 860   16    4   12   29    2    2    6   12   31    5  197    7    9
     5    5]
 [  59  372   13  106   56   11   92    0   22   88  159  282   13  133
    11    2]
 [   2   30  507  116   20    7   65   99    8   30   90  204    8  128
     7    2]
 [   3   41   18  593   19    7   67    1   15   67   96  183   17  143
     4    0]
 [  13   41    6   52  626   19   52    3   93   90   65  173   16   33
    33    0]
 [   1   20    8    6   60  522    6    1  191   48   62   93    8    5
    52    0]
 [   1   51    1   32   15    8  187    1   16   41  109   82    6   81
     7    0]
 [   4    5   14   15   65    1    1  706    5   44    2  107    9    2
     2    0]
 [   5   26   15   11  173  100   22    2 1719  130   89  169   11    5
    50    0]
 [  38   50    9   47   55   26   42    5   64  595  170  242   16   52
    14    0]
 [  12   56    3   27   10   14   34    0   17   52  650  132    9   21
     4    0]
 [ 151   77   14   86   55    9   52    2   33  101  106  632   25  175
    15    1]
 [  60   50   15  101   98    7   46    2   52  140  100  382  704  224
    20    4]
 [  46   68   16   89    9    0  101    1    1   49  206  164    7  830
     2    8]
 [   7    1    1    4   24   16    7    0   51   42   12   42    0    0
   264    1]
 [  57   19    7   24   12    2    9    0    2   24   25  125    6   98
     2  649]]
```




**kNN Classification** We'll use kNearestNeighbor for classification now. We tried different values for k and 4 came out to be the best for this.

```
In [10]: knn = KNeighborsClassifier(n_neighbors = 5)
         knn_me = testClassifier(train_tf, y_train, test_tf, y_test, knn)
         metrics_dict.append({'name':'kNN', 'metrics':knn_me})
```

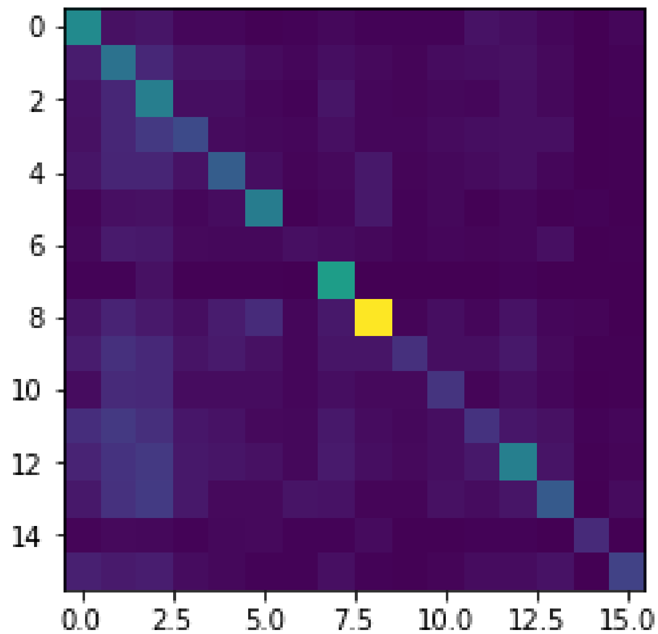
```
training time:  0:00:00.061494
testing time:  0:01:48.927677
classification report:
                  precision    recall  f1-score   support

KaiserHealthNews       0.37      0.60      0.45      1202
       NBChealth       0.21      0.40      0.28      1419
       bbchealth       0.23      0.49      0.31      1323
       cbchealth       0.29      0.26      0.28      1274
       cnnhealth       0.35      0.34      0.34      1315
  everydayhealth       0.47      0.58      0.52      1083
   foxnewshealth       0.14      0.09      0.11       638
   gdnhealthcare       0.47      0.85      0.61       982
      goodhealth       0.70      0.59      0.64      2527
   latimeshealth       0.42      0.15      0.22      1425
   msnhealthnews       0.29      0.22      0.25      1041
       nprhealth       0.27      0.14      0.19      1534
    nytimeshealth       0.44      0.32      0.37      2005
```

```
  reuters_health       0.43      0.26      0.33      1597
    usnewshealth       0.67      0.38      0.48       472
       wsjhealth       0.61      0.28      0.38      1061

     avg / total       0.41      0.38      0.37     20898

f1 score
0.3597917504797554
accuracy score
0.37960570389510956
confusion matrix:
[[ 717   67   84   24   28   11   13   34   13   12   15   73   56   24
     5   26]
 [ 114  565  169   77   77   45   26   53   39   23   49   58   66   40
     5   13]
 [  75  161  644   54   54   25   13   83   29   19   30   28   59   33
     2   14]
 [  70  164  251  335   46   30   24   64   25   25   45   57   64   60
     5    9]
 [  87  158  158   71  444   54   21   41   95   18   31   43   61   24
     2    7]
 [  23   62   66   24   48  628    9   24   97   14   32    9   26    8
    13    0]
 [  30  108   94   37   32   32   59   52   35   12   26   23   27   61
     4    6]
 [  12   14   65    9   10    6    0  838    4    3    1    2   12    3
     0    3]
 [  77  151  105   57  113  183   28   95 1501   18   54   26   73   24
    19    3]
 [ 112  211  172   80  111   66   28   83   86  211   57   58   94   40
     6   10]
 [  49  179  175   47   50   47   24   56   38   38  225   22   54   26
     3    8]
 [ 195  247  204   88   75   39   30   97   52   34   57  220   92   65
    14   25]
 [ 148  219  251   95   84   68   32  109   56   37   57   94  648   77
     8   22]
 [  98  213  253   96   38   38   82   72   22   20   68   47   83  423
     1   43]
 [  18   37   32   15   33   36   13   16   46    8   10   12   15    2
   177    2]
 [ 134  106  123   51   30   16   16   61   15   10   23   51   51   74
     2  298]]
```




**Conclusion** For this dataset the Rocchio outperformed the rest of the two classification algorithms and it also was the one that took the least amount of time for training as well testing of the data.

### 0.3 Sentence Corpus

First of all we need to do pre-processing on the data as it is in raw text format. We split the dataset according to the Argumentative Zones annotation scheme and clean-up the text.

**Pre-processing** We divide the documents into different classes according to the Argumentative Zones annotation scheme. The documents are then converted to tf-idf vectors. Further they are split into train test sets.

```
In [11]: sentence_corpus = os.listdir('Datasets/SentenceCorpus/SentenceCorpus/labeled_articles
         X_data = []
         y_data = []
         for files in sentence_corpus:
             file = open ('Datasets/SentenceCorpus/SentenceCorpus/labeled_articles/'+files)
             data = file.readlines()
             for lines in data:
                 if '###' not in lines:
                     lines = lines.split("\t")
                     try:
                         X_data.append(lines[1].lower().replace(' citation',''))
                         y_data.append(lines[0].lower().strip())
```

```
                except: pass
            file.close()
        vectorizer = CountVectorizer()
        vectorizer.fit(X_data)
        train_mat = vectorizer.transform(X_data)
        tfidf = TfidfTransformer()
        tfidf.fit(train_mat)
        train_tfmat = tfidf.transform(train_mat)
        X_train, X_test, y_train, y_test = train_test_split(train_tfmat,
                                    y_data, test_size=0.33, random_state=42)
```

**Naive Bayes** We have chosen MultinomialNB as it gives the best results for Naive Bayes in case of text classification.

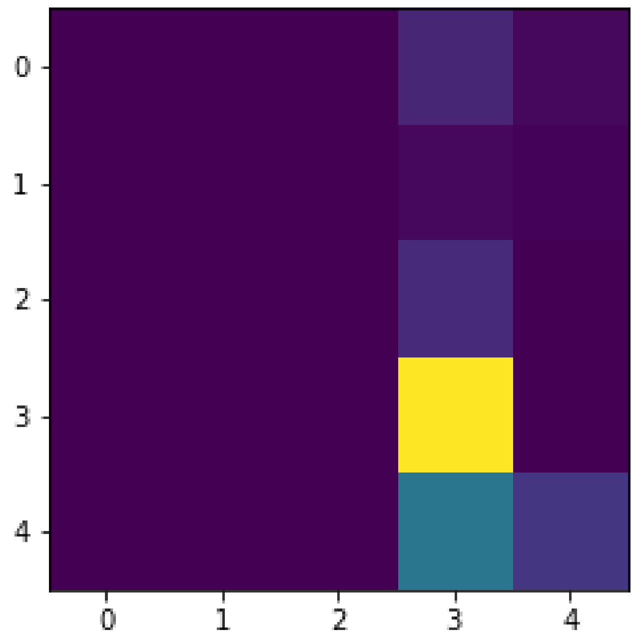
```
In [12]: mnb = MultinomialNB()
         mnb_me = testClassifier(X_train, y_train, X_test, y_test, mnb)
         metrics_dict.append({'name':'NaiveBayes', 'metrics':mnb_me})
```

```
training time:  0:00:00.008956
testing time:  0:00:00.001373
classification report:
             precision    recall  f1-score   support

       aimx       0.00      0.00      0.00        35
       base       0.00      0.00      0.00         9
       cont       0.00      0.00      0.00        33
       misc       0.61      1.00      0.76       272
       ownx       0.83      0.29      0.43       150

avg / total       0.58      0.63      0.54       499

f1 score
0.23646979440642255
accuracy score
0.6312625250501002
confusion matrix:
[[  0   0   0  29   6]
 [  0   0   0   6   3]
 [  0   0   0  33   0]
 [  0   0   0 272   0]
 [  0   0   0 107  43]]
```




**Rocchio Classification** For this we will be using the NearestCentroid classifier as when it is used for text classification with tf-idf vectors, this classifier is also known as the Rocchio classifier.

```
In [13]: tfidf = TfidfTransformer()
         tfidf.fit(X_train)
         train_tf = tfidf.transform(X_train)
         test_tf = tfidf.transform(X_test)
         ncr = NearestCentroid()
         ncr_me = testClassifier(train_tf, y_train, test_tf, y_test, ncr)
         metrics_dict.append({'name':'Rocchio', 'metrics':ncr_me})
```

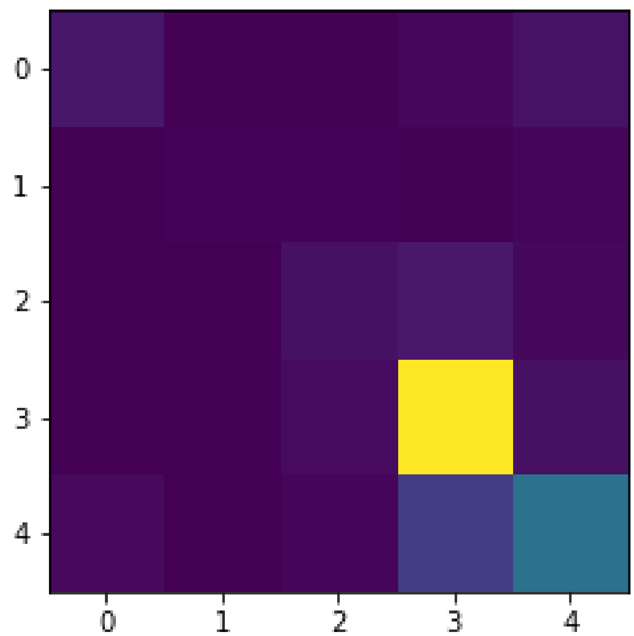
```
training time:  0:00:00.013593
testing time:  0:00:00.003481
classification report:
             precision    recall  f1-score   support

       aimx       0.73      0.46      0.56        35
       base       0.50      0.22      0.31         9
       cont       0.44      0.33      0.38        33
       misc       0.79      0.93      0.85       272
       ownx       0.75      0.63      0.68       150

avg / total       0.74      0.75      0.74       499

f1 score
```

```
0.5562834866867061
accuracy score
0.7535070140280561
confusion matrix:
[[ 16   1   1   5  12]
 [  0   2   2   1   4]
 [  0   0  11  17   5]
 [  0   1   7 253  11]
 [  6   0   4  46  94]]
```




**kNN Classification** We'll use kNearestNeighbor for classification now. We tried different values for k and 4 came out to be the best for this.

```
In [14]: knn = KNeighborsClassifier(n_neighbors = 5)
         knn_me = testClassifier(train_tf, y_train, test_tf, y_test, knn)
         metrics_dict.append({'name':'kNN', 'metrics':knn_me})
```

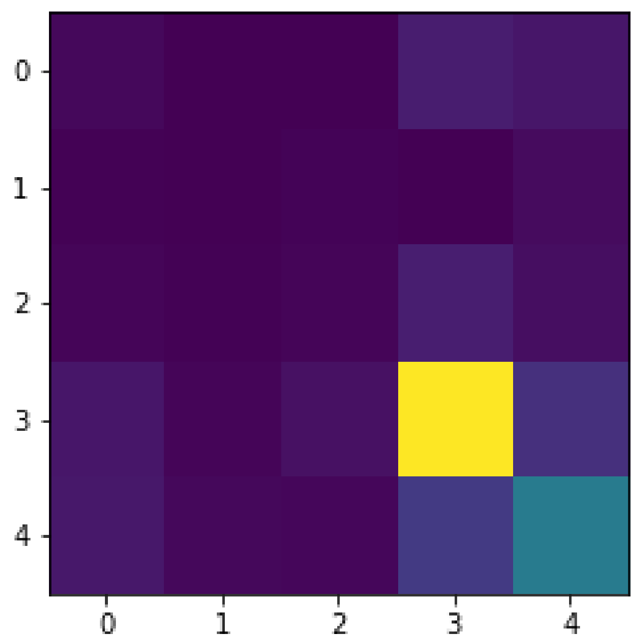
```
training time:  0:00:00.004380
testing time:  0:00:00.055142
classification report:
             precision    recall  f1-score   support

       aimx       0.14      0.14      0.14        35
       base       0.00      0.00      0.00         9
       cont       0.16      0.09      0.12        33
```

```
       misc       0.75      0.79      0.77       272
       ownx       0.61      0.60      0.61       150

avg / total       0.61      0.63      0.62       499

f1 score
0.32674377265926563
accuracy score
0.6292585170340681
confusion matrix:
[[  5   0   0  17  13]
 [  1   0   2   0   6]
 [  3   1   3  18   8]
 [ 13   3  10 216  30]
 [ 14   5   4  37  90]]
```




**Conclusion** For this dataset again the Rocchio outperformed the rest of the two classification algorithms and it also was the one that took the least amount of time for training as well testing of the data. The Naive Bayes performed the worst in this case