COL 774 : Assignment 2

Prakhar Ganesh - 2015CS10245

1. Text Classification

   a. Simple Naive Bayes Implementation ->

   Training Set Accuracy = 68.856%
   Test Set Accuracy = 37.388%

   b. Random and Majority Prediction ->

   Random Prediction Accuracy = 11.172%
   Majority Prediction Accuracy = 20.088%

   Clearly there is a visible improvement in the accuracy when we use naive bayes in place of random prediction or majority prediction.

   From the implementation of random prediction, we see an increase in accuracy of 26.216% in naive bayes.

   From the implementation of majority prediction, we see an increase in accuracy of 17.3% in naive bayes.

   c. Confusion Matrix ->

   The confusion matrix formed is (Rows are predicted labels, columns are true labels):

4726	2028	1958	1674	656	615	435	975
0	2	1	2	0	0	0	0
4	4	28	4	0	2	0	2
34	56	140	241	55	33	9	8
9	13	24	62	91	34	16	13
33	44	120	218	436	469	210	210
0	0	0	1	0	2	0	1
216	155	270	433	1069	1695	1674	3790

   Highest value of the diagonal entry is of the class 0, i.e. rating 1. This means that the model predicts class 0 with maximum confidence and that this class has the maximum contribution in the total number of true positives in my results.

Studying the matrix further we can clearly say that the prediction of class 0 happens far more times than any other class and thus number of false negatives for class 0 are also a lot. This means that there is a lot of confusion in the model in distinguishing any class from class 0. Also we can say that most of the predictions turn out to be either class 0 or class 7, i.e. rating 10 in our model.

Another thing we can interpret is that the model is only capable of correctly identifying a certain number of classes and not all the classes and thus a different distribution of labels in the test dataset could drastically change the accuracy of the model.

d. Stopword Removal and Stemming ->

New Accuracy = 38.712%

Confusion Matrix (Rows are predicted labels, columns are true labels):

4243	1560	1336	977	360	380	298	632
82	55	69	53	18	20	9	18
159	167	237	259	92	87	36	50
240	282	496	654	283	201	108	128
41	62	101	202	375	294	153	186
69	46	110	219	514	701	434	508
24	9	17	30	86	142	111	175
164	121	175	241	579	1025	1195	3302

The confusion matrix formed this time is far more distributed than previously and there are a significant number of true positives (diagonal values) for every class.

This would mean that even though the increase in accuracy on this test dataset is small, this accuracy will not drastically change on changing the distribution of labels in the test dataset.

e. Feature Engineering ->

There are a number of features that can be added on top of our multinomial Naive Bayes. These can include ->

1. Adding Bi-grams in our vocabulary.
2. Removing uni-grams/bi-grams with less than a certain threshold count.
3. Giving more importance to all Capital Words by adding one more occurence of them in the sentences.
4. Including negation by changing the words near ‘not’. Eg -> ‘not good’ get changed to ‘not NOT_good’.

After adding all these features on top of my Naive bayes model, I got the following results ->

Test Accuracy -> 36.96%

Confusion matrix (Rows are predicted labels, columns are true labels):

4791	2101	2128	1863	672	589	403	820
0	0	0	0	0	0	0	0
1	0	7	2	0	2	0	0
9	17	44	111	25	17	7	6
4	2	8	21	45	17	5	2
15	17	61	120	224	207	91	92
0	0	0	0	0	0	0	0
202	165	293	518	1341	2018	1838	4079

Adding these features caused a clear sparsity in the confusion matrix. This can be explained by the fact that the documents of a certain class which were present in the training data a lot had a higher chance of having all the new features that I added on top of my naive bayes which should intuitively represent the data more accurately.

Thus we can see that even strong intuitive features added does not cause too much of a change in the accuracy of our model, because of the class imbalance present in our training data, our disregard for the number of words present in every class and finally because of our bag of words model.

2. Support Vector Machines (SVMs)

a. Pegasos Algorithm ->

Pegasos: Primal Estimated sub-GrAdient SOlver for SVM was implemented for a batch of size 100 and values of ‘w’ and ‘b’ were calculated for kC2 classifiers (45 classifier in our case).

The following results were found after 10,000 iterations and with C=1:

Training Dataset Accuracy -> 88.715%
Test Dataset Accuracy -> 90.12%

b. LIBSVM for Linear and Gaussian kernel->

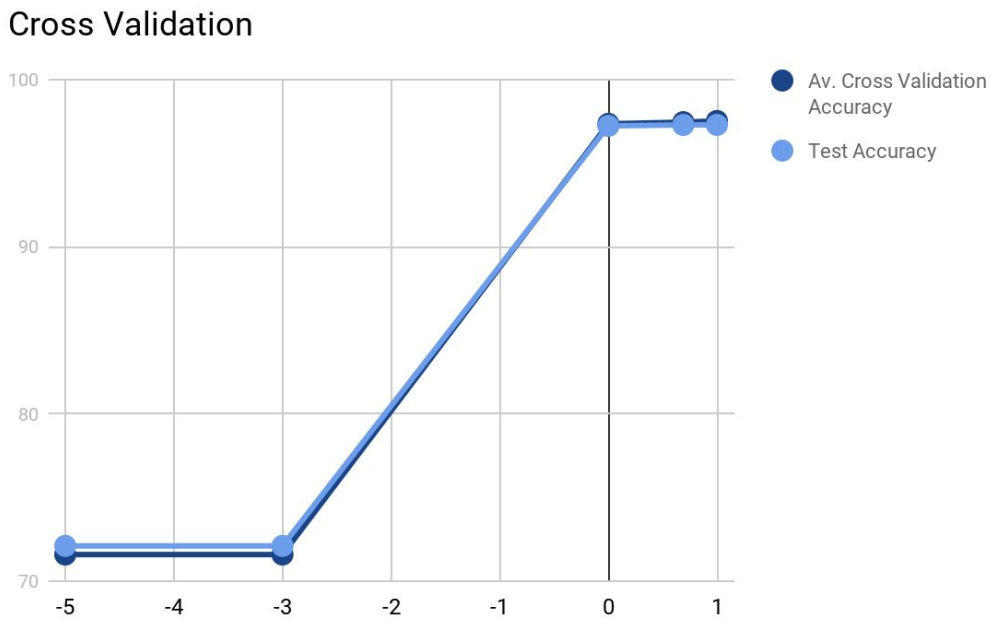
The data given to us was first converted into the required format after which LIBSVM library was trained on scaled training dataset and run with both linear and gaussian kernels. The test dataset accuracies obtained were ->

Linear kernel -> 92.78%
Gaussian kernel -> 97.23%

The test dataset accuracy obtained by LIBSVM with a linear kernel is slightly better than our implementation of SVM using Pegasos Algorithm, due to different convergence criteria and different algorithms used

c. LIBSVM with cross validation ->

LIBSVM was run on the training dataset with a gaussian kernel, 10-fold cross validation and for values of C = 0.00001, 0.001, 1, 5 and 10



The first thing we notice from the graph is that increase in C causes an increase in both the training and the test accuracies.
Also we can clearly see from the graph that at value of C=1, the accuracy reaches a sort of saturation point beyond which increase in value of C only causes an insignificant increase in accuracy.
However if we look at the exact values, we get maximum training and test accuracies both at C=10.

C	Av. Cross Validation Accuracy(%)	Test Accuracy(%)
0.00001	71.59	72.11
0.001	71.59	72.11
1	97.355	97.23
5	97.455	97.29
10	97.525	97.29

Now, for the best value of C (C=10), the confusion matrix formed is (Rows are predicted labels, columns are true labels)->

```
[ 969.    0.    4.    0.    1.    2.    5.    1.    4.    4.]
[   0. 1122.    0.    0.    0.    0.    4.    4.    0.    4.]
[   1.    3. 1000.    8.    4.    3.    0.   20.    3.    3.]
[   0.    2.    4.  985.    0.    6.    0.    2.   10.    8.]
[   0.    1.    2.    0.  962.    1.    3.    3.    1.    9.]
[   3.    2.    0.    4.    0.  866.    4.    0.    5.    4.]
[   4.    2.    1.    0.    5.    7.  940.    0.    3.    0.]
[   1.    0.    6.    7.    0.    1.    0.  986.    3.    9.]
[   2.    2.   15.    5.    2.    5.    2.    2.  942.   11.]
[   0.    1.    0.    1.    8.    1.    0.   10.    3.  957.]
```

Clearly, the class which was predicted wrong the most time is class ‘9’

Also the big off-diagonal values we see are between class ‘8’ and ‘2’ (15) and class ‘7’ and ‘2’ (20), which shows that distinct between these two is comparatively the most difficult for our model.

Wrong Predictions (Examples)->

Actual -> 3 Predicted -> 5

Actual -> 3 Predicted -> 7

Actual -> 9 Predicted -> 8