COL774 ASSIGNMENT 2

Raval Vedant Sanjay

2017CS10366

1 Text Classification

1.1 Implementing Naive Bayes

In this section, I just split the input data as it is and then run the Naive Bayes Algorithm on it to get the conditional probabilities. And then, used them to get the final accuracies

The Accuracy reported for the Training set is: 0.862466875
The Accuracy reported for the Test set is: 0.8217270194986073

1.2 Random prediction and Majority Prediction

The Accuracy reported on the Test set for the Random Prediction is: 0.5041782729805014
The Accuracy reported for the Test set for the Majority Prediction is: 0.5069637883008357

Thus, from these values, we can clearly see that the Naive Bayes has a humongous improvement over the two given methods

1.3 Confusion Matrix

The confusion matrix obtained was:

	Actual 0	Actual 1
Predicted 0	147	34
Predicted 1	30	148

The Highest value of the Diagonal Entry is corresponding to the case where the data is actually having a polarity 1 and is predicted 1 as well. So, this means that among all the predictions, the one for the case where the test of polarity 1 is also reported as having a polarity 1 is the most accurate. Or we can also say that in the overall accuracy of the test data, the maximum contribution comes from the **True One's**. Though, we can't say that the accuracy of predicting a true 1 is higher than the accuracy of predicting a true 0 (Since it also depends on the false 1's and 0's respectively)

From the confusion matrix, we can obtain the accuracy as *diagonal/total*, where *diagonal* is the sum of the diagonal entries and *sum* is the sum of all the entries in the matrix. From the confusion matrix, we can also get the total number of data points which are actually having the polarity as zero and one, by taking the sum of the elements of the corresponding columns

1.4 Stop work removal and Stemming

First of all, I would split all the messages by having the separation criteria as " ". Then, I will go through the words and remove the ones having "@" as their first character i.e. Removing the twitter handles. Then, I will remove the special characters in the words and will remove that word if the resulting length of it is less than or equal to one. Thus, I also have removed the special characters from the message. Now, I will check for the stop words and if not, then I will perform stemming of the word and store it in the new message and in this way, I will have a modified message list with appropriate data processing.

The Accuracy reported for the Training set is: 0.80668375
The Accuracy reported for the Test set is: 0.8272980501392758

Thus, here I find that the accuracy for the training set is reduced but the accuracy for the test set seems to be slightly improved! So, such data processing will have a positive influence on the accuracies of the test set since they are removing all the stop words and all which bring around substantial amount of noise in the measurements!

1.5 Feature Engineering

1.5.1 Bi-grams

The bi-gram formation can be explained by the example below:

Input String: ['a','b','c','d']
Bi-gram string: ['a-b','b-c','c-d']

Thus I performed this operation on the string lists coming from the data processing done in the Part (D) and used the Same Naive Bayes Algorithm. With that, the results obtained are as follows:

The Accuracy reported for the Training set is: 0.9232075
The Accuracy reported for the Test set is: 0.7632311977715878

So here I observe that the accuracy for the training set has improved substantially while for the test set it seems to have decreased slightly from the Previous algorithms.

1.5.2 Skip-grams

The skip gram formation can be explained by the example below:

Input String: ['a','b','c','d']
Skip-gram string: ['a-b','a-c','a-d','b-c','b-d','c-d']

Thus I performed this operation on the string lists coming from the data processing done in the Part (D) and used the Same Naive Bayes Algorithm. With that, the results obtained are as follows:

The Accuracy reported for the Training set is: 0.91679
The Accuracy reported for the Test set is: 0.841225626740947

So here I observe that the accuracy for the training set has improved substantially and so has the accuracy for the test set as compared to the results of Part(A) and Part(D). So from this, it can be seen that for the given data, skip-gram features perform the best among among the ones considered, improving the overall accuracy

1.6 TF-IDF

In this section, initally there were issues of the memory size because of the large files given to us. That's why I divided the data into batches of size 1000 and did partial fit on them! I also fit the model on the train data and transformed the train as well as the train data to that model in order to have a consistent dictionary. I also experimented by taking the terms having some minimum document frequency in order to reduce the token size and thus get the results faster. I also removed the features by using the SelectPercentile as asked in the problem statement. The results obtained are as follows:

By taking all the features:

- Test Accuracy: 0.5013927576601671
- Total Time taken: 12668.694222688675 seconds

By taking an argument of $Min\_df = 0.00001$ to get only those tokens having the probability for occurring in the dictionary to be more than 0.00001:

- Test Accuracy: 0.6323119777158774
- Total Time taken: 542.2315518856049 seconds

By taking an argument of $Min\_df = 0.001$ to get only those tokens having the probability for occurring in the dictionary to be more than 0.001:

- Test Accuracy: 0.766016713091922
- Total Time taken: 71.49561762809753 seconds

By taking an argument of $Min\_df = 0.01$ to get only those tokens having the probability for occurring in the dictionary to be more than 0.01:

- Test Accuracy: 0.6601671309192201
- Total Time taken: 51.691855669021606 seconds

By taking a total of 2 percentile in the SelectPercentile function:

- Test Accuracy: 0.5348189415041783
- Total Time taken: 291.038950920105 seconds

By taking a total of 1 percentile in the SelectPercentile function:

- Test Accuracy: 0.6601671309192201
- Total Time taken: 203.39629316329956 seconds

By taking a total of 0.5 percentile in the SelectPercentile function:

- Test Accuracy: 0.766016713091922
- Total Time taken: 112.98161363601685 seconds

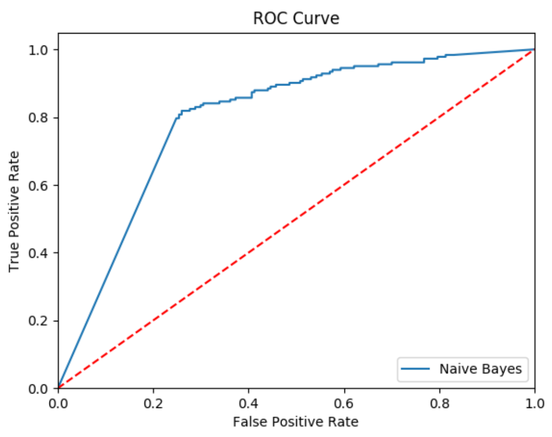
By taking a total of 0.1 percentile in the SelectPercentile function:

- Test Accuracy: 0.7493036211699164
- Total Time taken: 64.53151297569275 seconds

The Maximum Accuracy for the SelectPercentile term comes corresponding to the case of taking the first 0.5 percentiles of scores. While, for the case of the Min_df thing, the maximum comes corresponding to the minimum frequencies of 0.01. Though an important thing to notice here is that even though these notions of SelectPercentile and Min_df terms might seem to be the same, but they are indeed not so! We can even see their differences corresponding to the time taken as well as the accuracies corresponding to the seemingly same paremeter values! The Min_df has this notion of taking the features of having a minimum document frequency, while the SelectPercentile term has an advanced notion of the scores of the different terms involved (For obtaining this, we also need to pass the labels for the different terms)

1.7 ROC Curves

Since here we were supposed to be doing binary classification, so only one ROC curve needs to be shown in this case. I drew the curve corresponding to the above TF-IDF implementation, along with the minimum document frequency set as 0.001. The plot obtained is given below:



So, here we can see that the curve lies well above the line of TPR=FPR. Thus the current classifier can be termed as a "good" one. Here, we can see that even at a FPR of 0.2, the model reaches a TPR of 0.8. After this point, the TPR doesn't increase much. Thus, here we can see that the model classifies most of the examples with high confidence values, but a small number of examples are tougher to classify because of the slope of the ROC curve getting lesser after reaching the TPR of 0.8

2 Fashion MNIST Article Classification

2.1 Binary Classification

Here, I made the use of the CVXOPT package by representing the dual objective in the form $\frac{1}{2}\alpha^T P\alpha + q^T\alpha$ and the constrains as $Ax = b$ and $Gx <= h$, with these values being:

$$\mathbf{P_{m\times m}}[i][j] = -y^{(i)}y^{(j)}K(x^{(i)}, x^{(j)})$$
$$\mathbf{Q_{m\times 1}}[i] = 1$$
$$\mathbf{A_{1\times m}}[i] = y^{(i)}$$
$$\mathbf{b} = 0$$
$$\mathbf{G_{2m\times m}} = [-Identity(m); Identity(m)]$$
$$\mathbf{h_{2m\times 1}} = [Zeros_{m\times 1}; C \times Ones_{m\times 1}]$$

Note that since the CVXOPT package actually minimizes the objective but for the SVM algorithm, we actually need to maximize the function $W(\alpha)$. So, while solving we use $min_\alpha(-W(\alpha))$ instead

Since my entry number ends at 6, so for me it was the value of **d** considered for all the further experimentation. In the case of a Linear Kernel, there is no issue of having any infinite size feature vector, so we can save and store the weight **w** and the intercept **b** as well! But for the Gaussian kernel, we can not use the weights **w** directly because of the feature vector being infinite dimensional. So, for that part, rather than using **w** to calculate $w^T\phi(x)$, I used the result of $w = \Sigma\alpha_i y^{(i)}\phi(x^{(i)})$ for the same, thus reducing the term of $w^T\phi(x)$ into the inner products of the form $< \phi(x), \phi(x^{(i)}) >$ which can be expressed as $K(x, x^{(i)})$

Another important point to see here is that that after solving the convex optimization problem by the CVXOPT package, we obtain all the **m** (the size of the training set) values of α but it does not report the value of α_i corresponding to a non-support vector as 0, but instead these values are some really small values. Now, there is an issue of filter out the smaller values by taking them as zero, and so a choice of a threshold parameter becomes very important here. In my experiments in this section, I have taken this threshold as 5×10^{-4}

2.1.1 Implementing SVM with Linear Kernel, using CVXOPT package

The Test set accuracy obtained is: 1.0
The Validation set accuracy obtained is: 1.0

2.1.2 Implementing SVM with Gaussian Kernel, using CVXOPT package

The Test set accuracy obtained is: 0.999
The Validation set accuracy obtained is: 0.992

2.1.3 Implementing SVM using Scikit package

2.1.3.1 Linear Kernel

The Test set accuracy obtained is: 1.0
The Validation set accuracy obtained is: 1.0

The Number of support Vectors ([Y=6,Y=7]): [23 29]
The Number of support Vectors for CVXOPT case([Y=6,Y=7]): [21 27]

The intercept term (b): 0.01020624
The intercept term for the CVXOPT implementation (b): 0.00689937
The norm of the weights of the two implementations: 0.002693224891995568

The running time for the Scikit case: 11.938684463500977 seconds
The running time for the CVXOPT case: 444.93955874443054 seconds

We can see that both the methods have the same accuracies. Though their support vectors differ slightly and this slight difference can be seen in their intercepts which differ from themselves by a small amount and are not too close to be considered approximately the same. Also, I tried to compare the weights of the two algorithms by taking their norm and this value comes out to be very small and so these methods have an agreement here as well. The only point of large difference is their running times, where the Scikit method performs exponentially better than the Convex Optimizer case and from this we can feel the result of the improvement of the traditional SVM algorithms over the manual optimizing method in the case of inputs having very high dimensions, along with having the same efficiency

2.1.3.2 Gaussian Kernel

The Test set accuracy obtained is: 0.999
The Validation set accuracy obtained is: 0.992

The Number of support Vectors ([Y=6,Y=7]): [459 215]
The Number of support Vectors for CVXOPT case([Y=6,Y=7]): [459 215]

The intercept term (b): -0.57690413
The intercept term for the CVXOPT implementation (b): -0.6041728714077751
The norm of the weights of the two implementations: N/A

The running time for the Scikit case: 14.87147307395935 seconds
The running time for the CVXOPT case: 561.6134850978851 seconds

We can see both the methods have the same accuracies, same support vectors, and same intercepts. But the big difference here can be seen in the running times of these algorithms just as in the case of the Linear Kernel case. So it seems that the Sklearn algorithm is very much optimized as compared to solving such large problem by some convex optimizer, and could possibly be using highly efficient algorithms like SMO.

2.2 Multi-class Classification

2.2.1 One Vs One Classifier Using CVXOPT

I implemented the One Vs One Multi-class classifier by obtaining a Model $M_{(i,j)}$ for every pair (i, j) corresponding to the different labels (from 0 to 9), and testing the data on the model along with!

The Test Set accuracy obtained is: 0.8508
The Validation set accuracy obtained is: 0.8492
The total time of execution for the method: 24585.234556674957 seconds

The confusion matrix obtained for the test data is:

	True 0	True 1	True 2	True 3	True 4	True 5	True 6	True 7	True 8	True 9
Predicted0	404	0	0	17	0	1	52	0	1	0
Predicted1	0	484	0	10	1	0	1	0	0	0
Predicted2	7	6	414	2	55	0	52	0	1	0
Predicted3	7	2	3	410	12	0	5	0	0	0
Predicted4	0	0	26	6	366	0	20	0	0	0
Predicted5	0	0	0	0	0	432	0	48	0	5
Predicted6	65	6	44	39	51	0	351	0	3	0
Predicted7	0	0	0	0	0	7	0	411	0	6
Predicted8	17	2	13	16	15	48	19	6	495	2
Predicted9	0	0	0	0	0	12	0	35	0	487

The confusion matrix obtained for the Validation data is:

	True 0	True 1	True 2	True 3	True 4	True 5	True 6	True 7	True 8	True 9
Predicted0	201	0	2	11	0	0	19	0	0	0
Predicted1	2	240	0	7	2	0	0	0	0	0
Predicted2	1	2	208	0	30	0	27	0	0	0
Predicted3	4	2	1	197	5	1	1	0	0	0
Predicted4	0	0	13	6	185	0	11	0	0	0
Predicted5	0	0	0	0	0	225	0	24	0	2
Predicted6	38	3	14	20	19	0	184	0	2	0
Predicted7	0	0	0	0	0	1	0	199	2	4
Predicted8	4	3	12	9	9	17	8	6	245	5
Predicted9	0	0	0	0	0	6	0	21	0	239

2.2.2 One Vs One Classifier Using Scipy

I implemented the One Vs One Multi-class classifier by obtaining a Model $M_{(i,j)}$ for every pair (i, j) corresponding to the different labels (from 0 to 9), and testing the data on the model along with!

The Test Set accuracy obtained is: 0.8808
The Validation set accuracy obtained is: 0.8788
The total time of execution for the method: 5042.879487276077 seconds

The confusion matrix obtained for the test data is:

	True 0	True 1	True 2	True 3	True 4	True 5	True 6	True 7	True 8	True 9
Predicted0	432	1	5	12	2	0	80	0	1	0
Predicted1	0	482	0	0	1	0	0	0	0	0
Predicted2	5	4	410	3	39	0	53	0	1	0
Predicted3	12	9	7	457	14	0	9	0	1	0
Predicted4	3	0	37	9	399	0	34	0	2	0
Predicted5	0	0	0	0	0	473	0	14	2	11
Predicted6	38	4	33	14	39	0	317	0	2	0
Predicted7	0	0	0	0	0	16	0	471	2	14
Predicted8	10	0	8	5	6	5	7	1	489	1
Predicted9	0	0	0	0	0	6	0	14	0	474

The confusion matrix obtained for the Validation data is:

	True 0	True 1	True 2	True 3	True 4	True 5	True 6	True 7	True 8	True 9
Predicted0	212	0	5	6	1	0	33	0	0	0
Predicted1	0	237	0	0	1	0	0	0	0	0
Predicted2	1	3	205	0	24	0	28	0	1	0
Predicted3	8	7	3	228	8	1	4	0	1	0
Predicted4	0	0	19	6	200	0	19	0	1	0
Predicted5	0	0	0	1	0	241	0	8	0	5
Predicted6	26	2	13	9	15	0	165	0	1	0
Predicted7	0	0	0	0	0	2	0	230	2	8
Predicted8	3	1	5	1	1	1	1	1	244	2
Predicted9	0	0	0	0	0	5	0	11	0	235

2.2.3 Comparision among the above two methods

In the above methods, we can see that the method corresponding to the OvO Scipy worked much faster as compared to the implementation of the OvO Matrix Solver, because of the highly optimized methods and the extremely fast SMO algorithm.

While comparing the outputs of the two methods, we can see that both of them report a very similar accuracy but the accuracy obtained for the SMO method is somewhat better than the one obtained corresponding to the standard matrix solver, giving 3% better accuracy than the CVXOPT method

If we look at the individual confusion matrices for both these methods (corresponding to the test and the validation data), we can see that the articles corresponding to the label of 6 are the ones that are the most misclassified, for both the test and validation data and corresponding to both the methods! While the articles with the label of 8 are the most correctly classified in all the cases.

If we look at the confusion matrices of both the cases, then they seem to be sharing nearly the same values especially at the diagonals (that's why they have a very similar performance in terms of accuracy). Also if we look at the link `http://fashion-mnist.s3-website.eu-central-1.amazonaws.com/`, we can see that the result obtained corresponding to the given dataset and the paremeters set by us, then we get that even the standard experimentation on the given dataset has the accuracy of about 88%! Thus, from these observations we can see that the results obtained by us do make sense!

A big reason for the difference between these two algorithms is corresponding to the number of support vectors! Since in the CVXOPT implementation, we need to set a threshold to get the different support vectors, so there would be a possibility that some more support vectors would be added or some support vectors would be missing in the outputs of this method, when compared with the ones obtained for the SMO method.

If we look into the given dataset closely, then the articles classified in the most correct way are corresponding to the "Bag" and there are no other objects which are closer to this object in the entire dataset. While the articles most misclassified are the ones corresponding to "Shirt" and they are mostly misclassified into "Tshirt" and "Pullover" and this makes sense because these two are the most similar to Shirts. We also find Tshirts and Pullovers getting misclassified a lot into Shirts as well. We also find "Coat" getting misclassified to a decent extent into "Shirt" and vice-versa which is quite expected as well ("Shirt" is label 6, "Tshirt" is label 0, "Coat" is label 4, and "Pullover" is label 2)

We also find the footwears getting somewhat misclassified among each other as well! While the trousers (label=1) don't get much misclassified into others because of their uniqueness!

Thus all these results actually make sense as such errors are also very easy to be made by the human eyes by observing images of just 28×28 pixel density and that too in a grey-scale!

2.2.4 K-fold implementation

Here, I divided the Training data set into 5 random equal parts and then run the 5-fold cross validation on them as discussed in the problem statement. The 5-fold accuracies obtained for the different values of C are as follows:

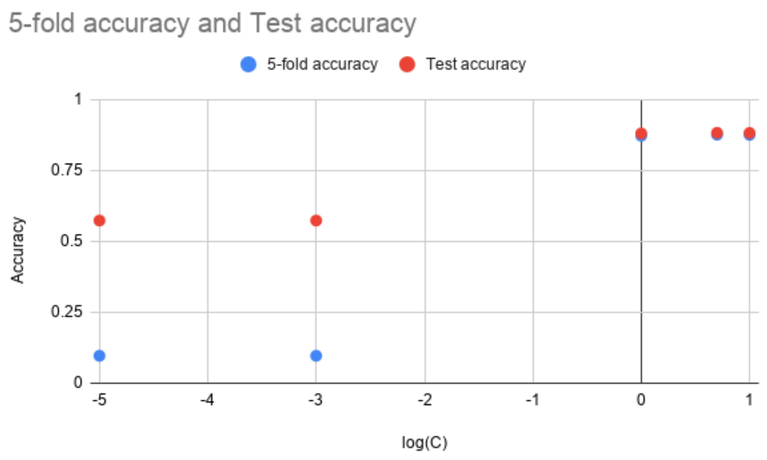
For the value of C = 10^{-5}: 0.0968006
For the value of C = 10^{-3}: 0.0968006
For the value of C = 1: 0.8725630
For the value of C = 5: 0.87623584
For the value of C = 10: 0.8758164

The values of the accuracies obtained for running the test sets for the different values of C are as follows:

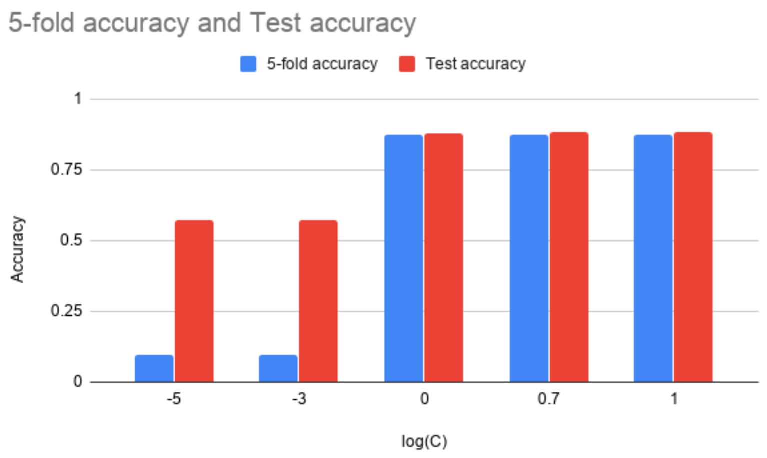
For the value of C = 10^{-5}: 0.5736
For the value of C = 10^{-3}: 0.5736
For the value of C = 1: 0.8808
For the value of C = 5: 0.8828
For the value of C = 10: 0.8824

Here, we find that for the lower values of C, the algorithm returns the same 5-fold accuracy as well as the same test accuracy, which means that there is no difference between bounding the parameters of the dual optimization problem by either 0.0001 or 0.000001. Also, we see that for the higher values of C, the 5-fold as well as the test data accuracies are nearly the same across all the three of them! But among them, the maximum value for the 5-fold cross-validation accuracy comes the maximum for the value of C=5 and the same trend is seen for the test data accuracies! Thus, we can see that the 5-fold cross-validation method works well here to obtain the best model parameters. The system took approximately 2 hours to run

The Scatter plot obtained for these values is given below:



The Column Chart obtained for these values is as follows:


5-fold accuracy and Test accuracy
5-fold accuracy
Test accuracy
Accuracy
log(C)