Assignment 2 (COL 774)

By Ritvik Vij (2017CS50417)

1. Naive Bayes for text classification

(a)_I have used the multinomial event model for classifying the given tweets as positive or negative.

Preprocessing of tweets :

1. All user tags after (@) were removed from the tweets.
2. All text was put to lower case, numbers were also removed from the text.
3. The text strings were also split by a set of punctuation marks (“,.!?/.&-:;% ").

A dictionary was maintained of words,with the string as key and the value as their word count In positive and negative tweets.

Laplace Smoothing was also done with the parameter(C=1)

Results:

1. A dictionary of size 4,38,313 is formed.
2. The accuracy on the testset is 80.44%

(b) **Random Classification and Majority Classification**

Accuracy with Random Classification is 51.95%
Accuracy with Majority Classification is 50.55%

Our Multinomial Event model gives almost 30% baseline improvement over Random/Majority Classification. Both random and majority classification have similar accuracy because the number of both classes is almost same in both training and test set.

(c) **Confusion Matrix:**

Actual Class(row) vs Predicted Class(Column)

	Positive	Negative
Positive	142	35
Negative	35	146

True Negative Values were the best predicted, amongst all. This might be because the number of negative classes test examples is the most, although the percentage is approximately the same.The training is also done on both these classes equally so there is not really much difference.

(d) **Stemming and Stop Word Removal**

Using the Natural Language Toolkit(nltk) library, stopwords are removed from the strings. Stemming is done to replace similar words with the same word in the dictionary such as [studying, studied, study] are replaced with the same word study. This reduces the size of dictionary to about 3,88,250, and increases the accuracy to about 81%. Stopwords such as couldn’t, didn’t etc. are removed by applying manual stemming on the list of stopwords (for eg could and not are both already present in the list of stopwords.This increases the accuracy to 82.3%.

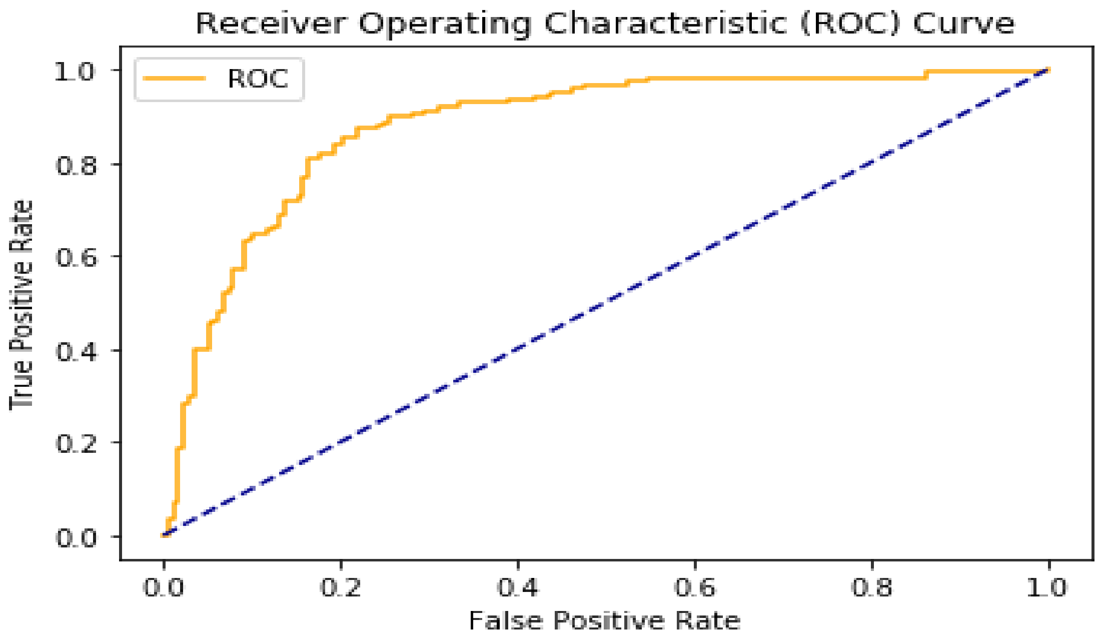
(e) **Feature Engineering on Text**

I have used two features on top of my existing model, bigrams with unigrams and trigrams with unigrams. I have created two extra independent dictionaries, to store pairs/ triplets of consecutive words with value as the number of times they occur in positive and negative tweets.

Bi grams along with unigram increases the accuracy to 83.6% and Trigrams keeps the accuracy increases a little bit but remains almost the same at 81.84%

(f) -------------------NOT ATTEMPTED----------------------

(g)



This is the ROC curve(True Positive Rate) vs (False Positive Rate).

An ROC curve plots TPR vs. FPR at different classification thresholds. Lowering the classification threshold classifies more items as positive, thus increasing both False Positives and True Positives. The following figure shows a typical ROC curve.

2. **<u>MNIST FASHION DATASET CLASSIFICATION USING SVM</u>**

1. Binary Classification

Using a Linear Kernel for SVM classification

My Entry number ends with 7 so i’ve chosen class 7 and 8 for the binary classification.

By formulating our objective function in the format used by CVXOPT format ($\alpha TP\alpha+qT\alpha+c$), we get alpha from the equation. We can get our weights and bias from alpha, hence giving us the output prediction for each input on the validation and test set in linear SVM prediction.

Results :

(a) There are a total of 120 support vectors from the optimisation. They are calculating by finding the number of dimensions of alpha with a value sizeable(>10^-4).

After carefully calculating the Weight Vector and the Bias Vectors:-

(b) Validation Accuracy comes out to be 99.2 % (496/500) test cases.

(c) Test Accuracy comes out to be 99.4 % (994/1000) test cases.

<u>Using a Gaussian Kernel for SVM classification</u>

All equations now, for input X will be replaced with Phi(X). Now, we don't know what Phi is because it is an infinite dimension matrix but we know what <Phi(X1),Phi(X2)> is, which is the Gaussian Kernel ($K(x1, x2) = \exp{-\gamma * // x1 - x2 // 2}$).

After we calculate alpha, we cannot directly take out the weight matrix because we don't know Phi(X). Instead, we can take out the output value as that involves Phi(X_train)Phi(X_Test) which is simply the kernel value. The Bias is calculated in the same way as before(min+max/2).

Results :

(d) There are a total of 1051 support vectors from the optimisation. They are calculating by finding the number of dimensions of alpha with a value sizeable(>10^-4).This number is much larger here than with the linear kernel.

After carefully calculating the Weight Vector and the Bias Vectors:-

(e) Validation Accuracy comes out to be 98 % (490/500) test cases.

(f) Test Accuracy comes out to be 99.2 % (992/1000) test cases.

(c) Now we used scikit-learn to train our binary classification SVM Model using both linear and gaussian kernel.

The code used to fit the data for each model is

clf = SVC(C = 1,kernel = 'linear') #Linear Fit

clf = SVC(C = 1,gamma=0.05, kernel = 'rbf') #Gaussian Fit

Y_pred = clf.predict(X_test) predicts the test dataset and returns a list of predicted classes.

Results :

Linear Kernel

(g) There are a total of 120 support vectors from the optimisation. They are found using the SVC attribute <SVC object> .n_support_. It’s identical from our own optimization mechanism for linear
(h) Validation Accuracy comes out to be 99.2 % (496/500) test cases.
(i) Test Accuracy comes out to be 99.4 % (994/1000) test cases.

Creation of matrices for cvxopt and time taken for the model to fit using our technique takes 24 seconds, by the CVXOPT library. Calculation of bias takes 0.9 seconds. Validation and test accuracy calculation is irrelevantly small.

The Sklearn library is rather very fast as compared to manual computation. It takes only 3 seconds for the model to fit. Prediction also takes only 1 second making it almost 6 times faster than the manual approach.

<u>Gaussian Kernel</u>

(j) There are a total of 1041 support vectors from the optimisation. They are found using the SVC attribute <SVC object> .n_support_. It's lesser by 10 than our own optimization mechanism for linear.

(k) Validation Accuracy comes out to be 98.6 % (492/500) test cases.

(l) Test Accuracy comes out to be 99.5 % (995/1000) test cases.

Creation of matrix P for cvxopt takes 12 seconds, time taken to find the optimal solution by the CVXOPT library took 31 seconds for 15 iterations. Calculation of bias takes almost 11 minutes . Validation and test accuracy calculation also takes about 1 minute each to predict

The Sklearn library is rather very fast as compared to manual computation. It takes only 5 seconds for the model to fit. Prediction also takes only 2 second making it almost 100 (gosh!) times faster than the manual approach.

2. Multi Class Classification

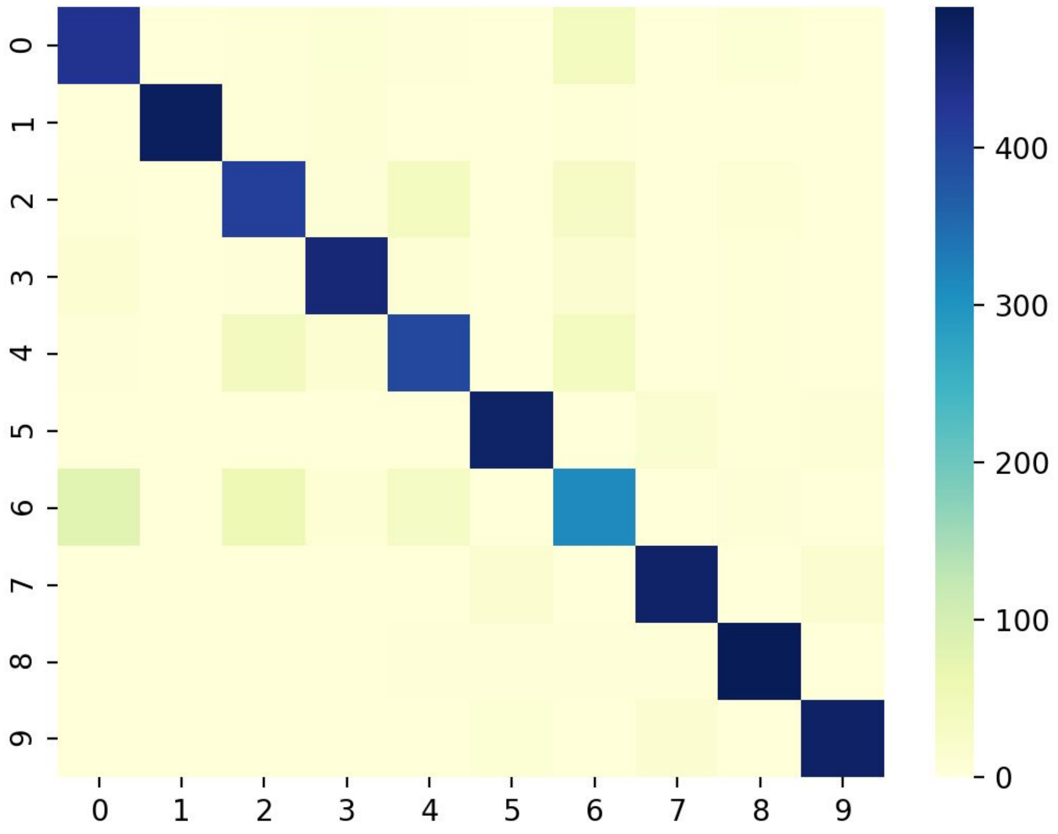
Using a Gaussian Kernel for SVM multi classification

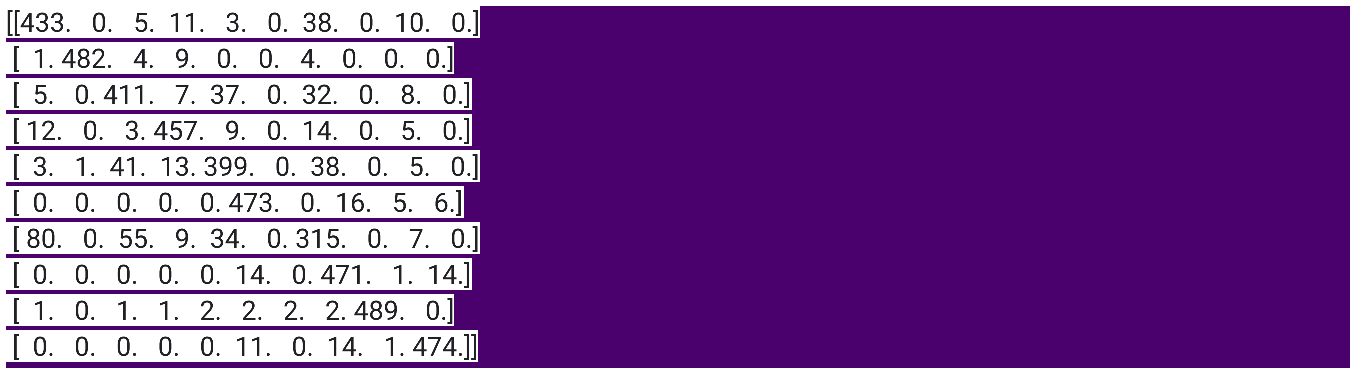
(a) In order to extend this to the multi-class setting, we train a model on each pair of classes to get C(k,2) classifiers,
k being the number of classes. During prediction time, we output the class which has the maximum number of votes from all the C(k,2) classifiers.

A one vs one classifier was implemented. The training time for this dataset was huge, it took around 2.5 hours for it to train majorly being calculation of P matrix and bias (which is now 45 times than the binary classification), validating and predicting val and test datasets was almost now sizeable, it took around 30 mins. Overall this was very computationally heavy, the accuracy comes out to 86.55% for validation dataset and 85.10% for test dataset.

(b) Using SVC was much faster but it took 13 minutes, 11 minutes for training and 2 minutes for prediction. The code to fit the data is no different than binary classification as sklearn allows you to input multi class data as well. The default mode is ovo(one vs one) so the code remains the same. The accuracy is around 88.8% for test dataset.

(c) Confusion Matrix is as shown below, using a seaborn heatmap. We can see that almost all our diagonal entries vary between 300-500, which means that our classifier is robust enough to correctly classify most of the examples.





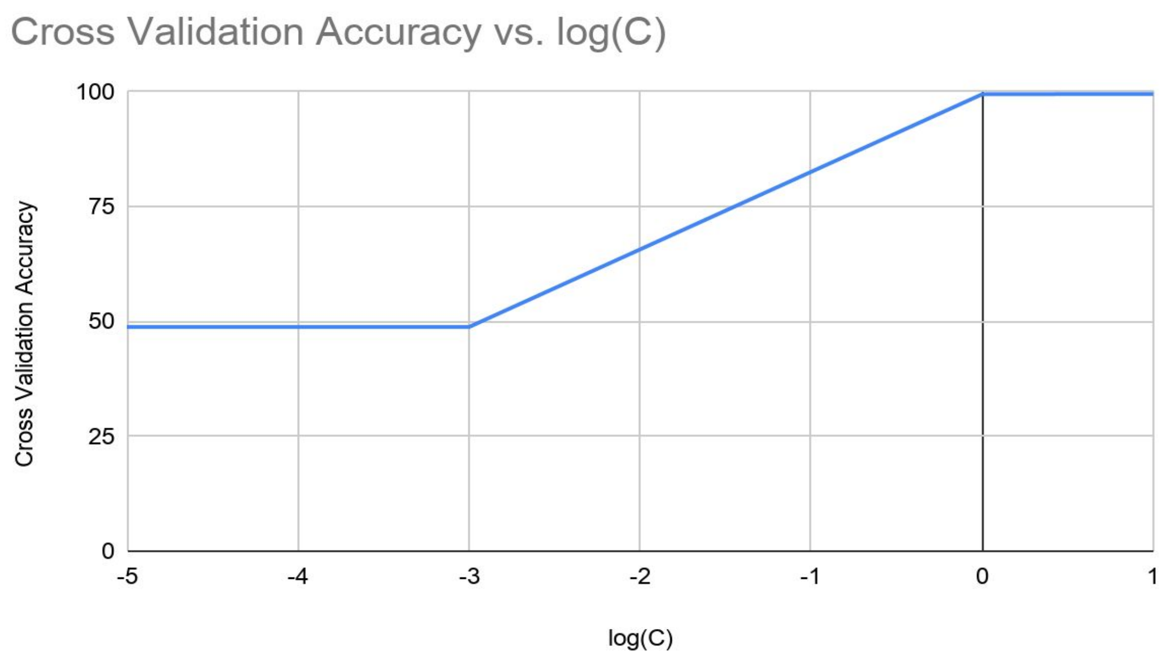
```
[[433.   0.   5.  11.   3.   0.  38.   0.  10.   0.]
 [  1. 482.   4.   9.   0.   0.   4.   0.   0.   0.]
 [  5.   0. 411.   7.  37.   0.  32.   0.   8.   0.]
 [ 12.   0.   3. 457.   9.   0.  14.   0.   5.   0.]
 [  3.   1.  41.  13. 399.   0.  38.   0.   5.   0.]
 [  0.   0.   0.   0.   0. 473.   0.  16.   5.   6.]
 [ 80.   0.  55.   9.  34.   0. 315.   0.   7.   0.]
 [  0.   0.   0.   0.   0.  14.   0. 471.   1.  14.]
 [  1.   0.   1.   1.   2.   2.   2.   2. 489.   0.]
 [  0.   0.   0.   0.   0.  11.   0.  14.   1. 474.]]
```

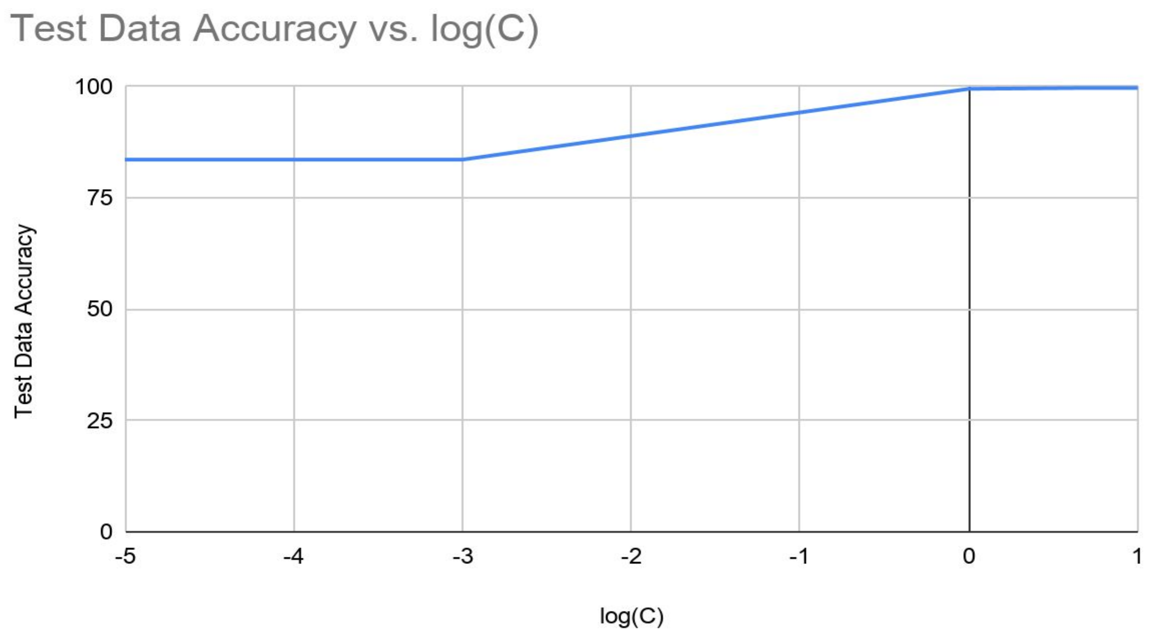
This is a short snippet of the values in the confusion matrix direct from the terminal. From this we observe that Class Number 5 and Class number 7, which are coat and shirt respectively are the most mis-classified classes, with shirt being most mis-classified as a coat and vice-versa which might also be a human mistake if the image was not very clear or blur.

(d) **5 Fold Cross Validation**

C	log(C)	Cross Validation Accuracy	Test Data Accuracy
0.00001	-5	48.8	83.6
0.001	-3	48.8	83.6
1	0	99.466	99.5
5	0.69897	99.511	99.7
10	1	99.511	99.7

According to the observations, we get the highest accuracy for both test and validation at C=5 and 10, and hence K fold cross validation is a wonderful technique to increase accuracy by finding the right hyperparameters.


Cross Validation Accuracy vs. log(C)
Cross Validation Accuracy
100
75
50
25
0
-5
-4
-3
-2
-1
0
1
log(C)


Test Data Accuracy vs. log(C)
Test Data Accuracy
100
75
50
25
0
-5
-4
-3
-2
-1
0
1
log(C)