COL774: Machine Learning- Assignment 2
Naive Bayes and Support Vector Machines (SVMs)

Rajat Jaiswal (2017CS50415)

8th March 2020

1 Naive Bayes

(a) The accuracy obtained over the **training set** is equal to **83.456%** and the accuracy obtained over the **test set** is equal to **81.337%**.

(b) By **randomly** guessing the accuracy varies for each run but for one of the runs the accuracy obtained was equal to **53.482%**. The algorithm gives an **improvement** of around **28%** over random prediction. And by **majority prediction** on the basis of training data, the accuracy obtained over test data is equal to **51.254%**. The algorithm gives an **improvement** of **around 30%** over random prediction.

(c) The confusion matrix is as shown below. The highest diagonal entry is for class = 0 i.e. negative emotions, though both the categories have almost same number of diagonal entry. This means that the classifier is slightly good at predicting for the class 0 articles. The non-diagonal entry for the class 0 is also slightly more than that of class 4. This means that the classifier is slightly biased towards predicting class 0. This may owe to the fact that while predicting, when the article has equal probability for both the classes, it is classified in class 0.

		Actual Classes	
		Class 0	Class 4
Predicted Classes	Class 0	147	37
	Class 4	30	145

Table 1: Confusion Matrix

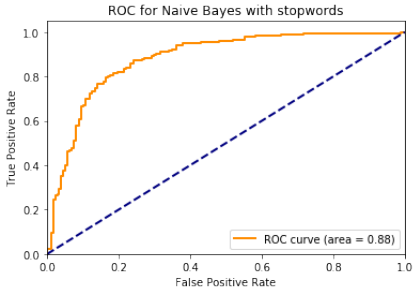

Fig 1: ROC Curve with stopwords

(d) In this part, I implemented stemming(using PorterStemmer of nltk library) and removed stop words. The accuracy for the **test set** increased to **81.894%**. This is a slight improvement of 0.5%. This may be because sometimes stemming can degrade accuracy.

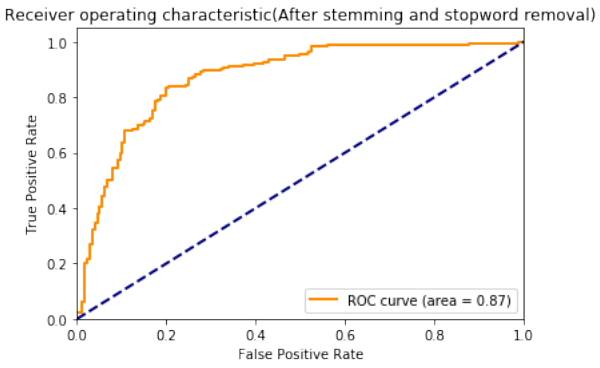

Fig 2: ROC Curve after stemming and stopword removal

(e) In this part I used two different features; Bigrams and Trigrams.

- **Bigrams:** After applying bigrams on top of the model, I got an accuracy of **84.123%** on the **test data**, and an accuracy of **91.422%** on the **training data**. It is better than the accuracies obtained in parts (a) and (d). While, forming bigrams, I also removed stop words from the text and applied stemming. This shows that bigrams gives us a better idea about the context of a text.
- **Trigrams:** After applying bigrams on top of the model, I got an accuracy of **82.73%** on the **test data**, and an accuracy of **95.505%** on the **training data**. It is better than the accuracies obtained in parts (a) and (d). While, forming trigrams, I also removed stop words from the text and applied stemming. This shows that though trigrams give better accuracy on the training data, they couldn't perform that well on new unseen data as much as the model with the bigrams did. This shows that trigrams may not be that good a feature for classification as bigrams are.

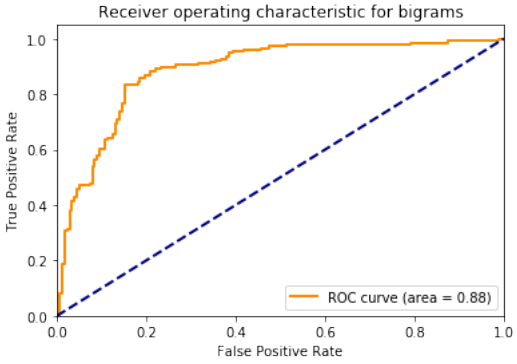

Fig 3: ROC Curve for bigrams

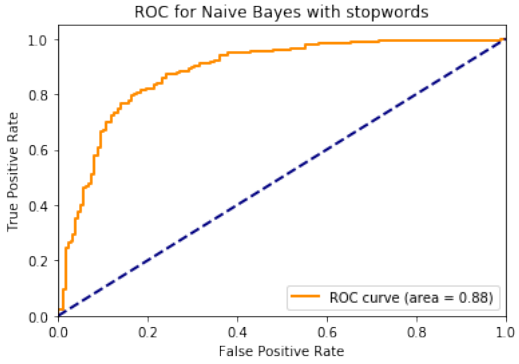

Fig 4: ROC Curve for trigrams.

So, bigrams help us to improve the overall accuracy of our model, though trigrams do too, but not as much as bigrams.

(f) The feature vector for training was very large and couldn't fit into the memory all at once in dense form. So I used the partial fit function of model, to train batch wise. I took the batch size = 500 to train the model. I also used stemmed words after stop words removal which also helped in reducing the number of features.

Percentile	Accuracy on test data	Training time(in s)
0.01	60.167%	30.414
0.1	73.816%	43.287
1	72.702%	94.803
100	49.303%	3382.924

Table 2: Comparison of accuracy and training time.

- The Gaussian NB here gives a very low accuracy (less than our implementation) because for all (100 percentile) features the gaussian model overfits and thus gives poor test accuracy.
- It is evident that taking a smaller number of features (1 and 0.01) leads to faster training times. Moreover, it also increases the accuracy, this is because now we are only considering those features which give us the maximum information about the document (thus reducing noise) and preventing over-fitting.
- On 0.01 percentile of the data, the accuracy reduces as we are now even ignoring some important features and thus under-fitting the data.

(g) **Receiver Operating Characteristic(ROC)** curve for the respective model is already given in the part where its accuracy is reported. The area under the ROC curve is 0.88 or 0.87 in all the cases, which means that the classifiers are a good one. The area under the ROC curve is a good measure of classifier's strength as AUC is scale invariant, i.e., it measure how well the prediction is ranked and not their absolute value. It is also classification threshold invariant, i.e., it measures the quality of the model's predictions irrespective of what classification threshold is chosen. So having a high area under ROC, suggests that a classifer is a good one. A perfect classifier's ROC has an area of 1

2 Support Vector Machines (SVMs)

In this section, we trained a model to classify fashion articles in their respective categories i.e. whether an image contains a sneaker, shirt, bag, etc. The articles were categorized into following classes:

- Class 0 - Tshirt/Top
- Class 1 - Trouser
- Class 2 - Pullover
- Class 3 - Dress
- Class 4 - Coat
- Class 5 - Sandal
- Class 6 - Shirt
- Class 7 - Sneaker
- Class 8 - Bag
- Class 9 - Ankle boot

2.1 Binary Classification

In this part a binary classifier was modeled to classify articles between **Sandal and Shirt (5 and 6)**.

(a) For **linear kernel**, the training time was 47.447s and the accuracies observed were **99.8% on validation set** and **99.6% on test set**. The number of support vectors were **86**. α_i was counted as a support vector if $\alpha_i > 10^{-7}$.

(b) For **gaussian kernel**, the training time was 36.824s and the accuracies observed were **99.6% on validation set** and **99.8% on test set**. The number of support vectors were **1161**. α_i was counted as a support vector if $\alpha_i > 10^{-7}$. Only b (intercept) could be calculated at train time but; w_i had to be calculated at test time. Though, a gaussian kernel is a more general kernel as it has more parameters, the accuracies obtained in linear and gaussian are almost same, this may be because sandals and shirts are very easy to differentiate and thus both the classifiers work well.

(c) It is evident from the data below that the accuracies are almost 100% for all implementations. However, the computational cost (training cost) of Scikit SVM package implementations are very less as compared to my implementation. The value of w and b are nearly the same for linear kernel between my implementation and Scikit SVM package's implementation.

In gaussian kernel, the $\alpha_{i's}$ obtained are nearly the same for Scikit SVM packgae and my implemenation, however, the value of b is a little different. Possible reason for this can be that we have used hard-margin formula for calculating b. This leads to approximate results as we have not included a term to handle the error ϵ_i which leads to a reduction in accuracy. Also, we have used a general convex optimiser and Scikit-learn uses SVM specific algorithms (and/or convergence criteria), which may lead to slightly different results.

	Linear Kernel		Gaussian Kernel	
	Scikit-learn SVM	My Implementation	Scikit-learn SVM	My Implementation
Training Time(in sec)	0.514	47.447	5.681	36.824
Test Accuracy	99.6%	99.6%	100%	99.8%
Validation Accuracy	99.8%	99.8%	99.6%	99.6%
No. of Support Vectors	85	86	1123	1161

Table 3: Comparison between different kernels and different implementations.

2.2 Multi-Class Classification

In this section, we will work with the entire subset of the data focusing on a multi-class classification problem for all the 10 given classes.

(a) We trained $\binom{10}{2}$ binary classifiers using the gaussian kernel. This was our own implementation using CVXOPT package. The one-vs-one multi-class SVM took **1476.589** seconds to train. The accuracies obtained were **84.96% on validation set** and **85.08% on test set**.

(b) Now we trained a multi-class SVM on this dataset using the Scikit-learn SVM library, this was done by training $\binom{10}{2}$ binary classifiers using the gaussian kernel. It took **287.775** seconds to train. The accuracies obtained were **87.88% on validation set** and **87.88% on test set**. We observe that Scikit-learn SVM library gives higher accuracy, and the training time is very less compared to our implementation. This is because of parallelization and several other code optimisation in Scikit-learn's SVM library.

(c) The confusion matrices for the **Validation Data** were:
(Rows are predicted category, Columns are actual classes)

My Implementation:

$$\begin{bmatrix} 201 & 0 & 2 & 11 & 0 & 0 & 19 & 0 & 0 & 0 \\ 2 & 240 & 0 & 7 & 2 & 0 & 0 & 0 & 0 & 0 \\ 1 & 2 & 208 & 0 & 30 & 0 & 27 & 0 & 1 & 0 \\ 4 & 2 & 1 & 197 & 5 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 13 & 6 & 185 & 0 & 11 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 226 & 0 & 24 & 0 & 2 \\ 38 & 3 & 14 & 20 & 19 & 0 & 184 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 199 & 2 & 4 \\ 4 & 3 & 12 & 9 & 9 & 16 & 8 & 6 & 245 & 5 \\ 0 & 0 & 0 & 0 & 0 & 6 & 0 & 21 & 0 & 239 \end{bmatrix}$$

Scikit-learn SVM Library:

$$\begin{bmatrix} 212 & 0 & 5 & 6 & 1 & 0 & 33 & 0 & 0 & 0 \\ 0 & 237 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 3 & 205 & 0 & 24 & 0 & 28 & 0 & 1 & 0 \\ 8 & 7 & 3 & 228 & 8 & 1 & 4 & 0 & 1 & 0 \\ 0 & 0 & 19 & 6 & 200 & 0 & 19 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 241 & 0 & 8 & 0 & 5 \\ 26 & 2 & 13 & 9 & 15 & 0 & 165 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 2 & 0 & 230 & 2 & 8 \\ 3 & 1 & 5 & 1 & 1 & 1 & 1 & 1 & 244 & 2 \\ 0 & 0 & 0 & 0 & 0 & 5 & 0 & 11 & 0 & 235 \end{bmatrix}$$

The confusion matrices for the **Test Data** were:
(Rows are predicted category, Columns are actual classes)

My Implementation:

$$\begin{bmatrix} 404 & 0 & 0 & 17 & 0 & 1 & 52 & 0 & 1 & 0 \\ 0 & 484 & 0 & 10 & 1 & 0 & 1 & 0 & 0 & 0 \\ 7 & 6 & 414 & 2 & 55 & 0 & 52 & 0 & 1 & 0 \\ 7 & 2 & 3 & 410 & 12 & 0 & 5 & 0 & 0 & 0 \\ 0 & 0 & 26 & 6 & 366 & 0 & 20 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 432 & 0 & 48 & 0 & 5 \\ 65 & 6 & 44 & 39 & 51 & 0 & 351 & 0 & 23 & 0 \\ 0 & 0 & 0 & 0 & 0 & 7 & 0 & 411 & 0 & 6 \\ 17 & 2 & 13 & 16 & 15 & 48 & 19 & 6 & 495 & 2 \\ 0 & 0 & 0 & 0 & 0 & 12 & 0 & 35 & 0 & 487 \end{bmatrix}$$

Scikit-learn SVM Library:

$$\begin{bmatrix} 432 & 1 & 5 & 12 & 2 & 0 & 80 & 0 & 1 & 0 \\ 0 & 482 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 5 & 4 & 410 & 3 & 39 & 0 & 53 & 0 & 1 & 0 \\ 12 & 9 & 7 & 457 & 14 & 0 & 9 & 0 & 1 & 0 \\ 3 & 0 & 37 & 9 & 399 & 0 & 34 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 473 & 0 & 14 & 2 & 11 \\ 38 & 4 & 33 & 14 & 39 & 0 & 317 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 16 & 0 & 471 & 2 & 14 \\ 10 & 0 & 8 & 5 & 6 & 5 & 7 & 1 & 489 & 1 \\ 0 & 0 & 0 & 0 & 0 & 6 & 0 & 14 & 0 & 474 \end{bmatrix}$$

- The item that has very good classification rate is Bag(Class 8). Then comes the Trouser(Class 1) and Ankle Boot(Class 9).
- Ankle Boots are sometimes misclassified as Sneakers(Class 7) or Sandals(Class 5), and vice-versa, due to their similar appearances.
- The item that is very often misclassified is Shirt(Class 6). It is oftenly misclassified into Tshirt(Class 0), Pullover(Class 2), and Coat(Class 4), owing very similar appearance to all these classes.
- Other items that have very bad classification accuracy are Tshirt(Class 0), Pullover(Class 2), and Coat(Class 4). Tshirts and shirts are oftenly misclassified into each other. And Pullovers, Shirts and Coats are also oftenly misclassified as each other. This is also the result of similar appearnace.
- A Dress(Class 3) is oftenly misclassified into Tshirt/Top and Shirt, but not the other way around. This is because Tshirt, Shirt and top are all categories of dress and the reverse is not true.
- Most of the items are mostly misclassified into a Shirt. This is because maximum number of classes in the model are upper wear and hence can be misinterpreted to be a Shirt.

(d) In this part, each value of C had to train 5 different models for 5-fold cross validation accuracy and hence the **multiprocessing** of library of python was used to train 5 different models in parallel using **hyper-threading**.

Value of C	Accuracy(Test)	Cross Validation Accuracy	Training time(in s)
10^{-5}	10.0%	9.484%	1488.728
10^{-3}	10.0%	9.484%	1502.265
1	87.512%	88.084%	491.997
5	88.032%	88.62%	529.057
10	88.024%	88.60%	535.461

Table 4: Comparison of accuracy for different Value of C.

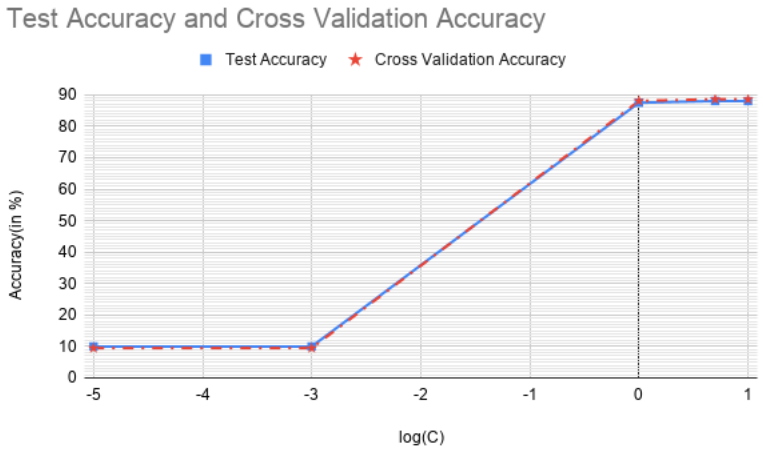

Fig 5: Test Accuracy and Cross Validation Accuracy for different Value of C.

Test Accuracy and Cross Validation Accuracy is almost same for all the values of C taken into consideration. The value of C = 5 gives the best cross-validation accuracy. At this value, the test accuracy is also the highest. So value of C = 5, is the best among the given 5 values. A smaller value of C means that the classifier increases the margin without caring about mis-classification of data points, thus giving a very low validation accuracy. A larger value of C means a smaller margin is taken in order to prevent mis-classification, so it gives higher accuracy. So, the optimal value of C depends on the scale of the data, how far or how close are the data points from the separator.

The model is trained in subject to constraints that $0 \leq \alpha_i \leq C$, hence when the value of C is very low the range for α_i is very less and a very precise value of α_i has to be obtained i.e. the $\alpha_{i's}$ have to calculated till more significant digits and hence the training time is higher for models with very small value of C.