Machine Learning - Assignment 2

Meraj Ahmed

March 2020

1 Naive Bayes

1.1 Text Classification

We have to implement Naive Bayes algorithm for classifications of tweets and predict its sentiment (Positive or Negative)

$$\phi_{k|y=1} = \frac{\sum_{i=1}^{m} \sum_{j=1}^{n_i} 1\{x_j^i = k \bigwedge y^i = 1\}}{\sum_{i=1}^{m} 1\{y^i = 1\} n_i + |V|}$$

$$\phi_{k|y=0} = \frac{\sum_{i=1}^{m} \sum_{j=1}^{n_i} 1\{x_j^i = k \bigwedge y^i = 0\}}{\sum_{i=1}^{m} 1\{y^i = 0\} n_i + |V|}$$

$$\phi_y = \frac{\sum_{i=1}^{m} 1\{y^i = 1\}}{m}$$

1.1.1 Parameters

Train Accuracy = 85.0578

Test Accuracy = 80.7799

Train Time = 30.00 sec

Majority prediction Accuracy = 49.3036

Random prediction Accuracy = 45.1253

1.1.2 Train

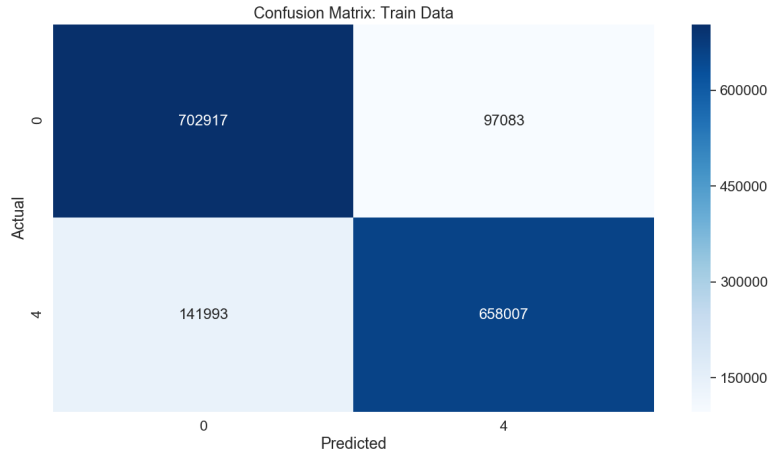

Figure 1: Confusion Matrix of Train

Figure 2: ROC of Train

1.1.3 Test

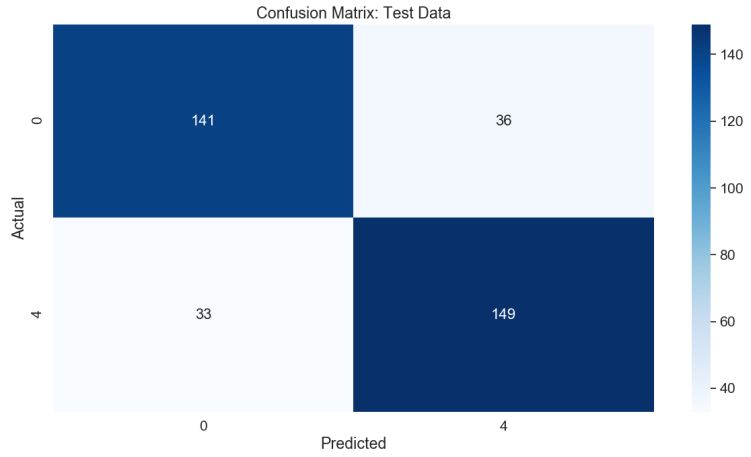

Figure 3: Confusion Matrix of Test

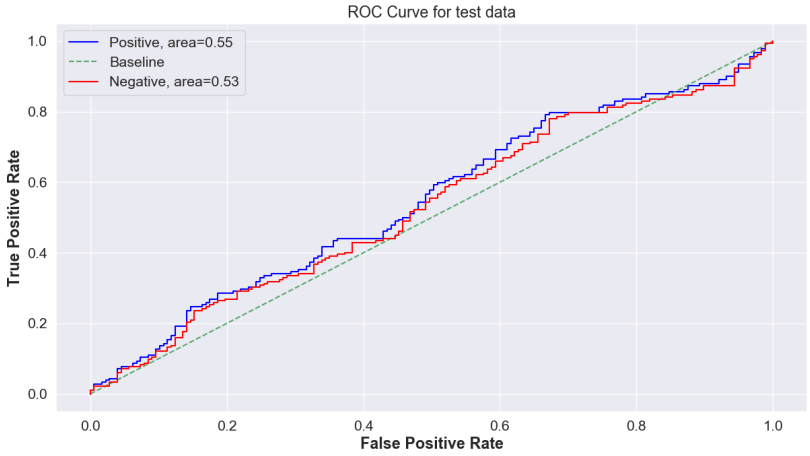

Figure 4: ROC of Test

1.1.4 Majority Prediction

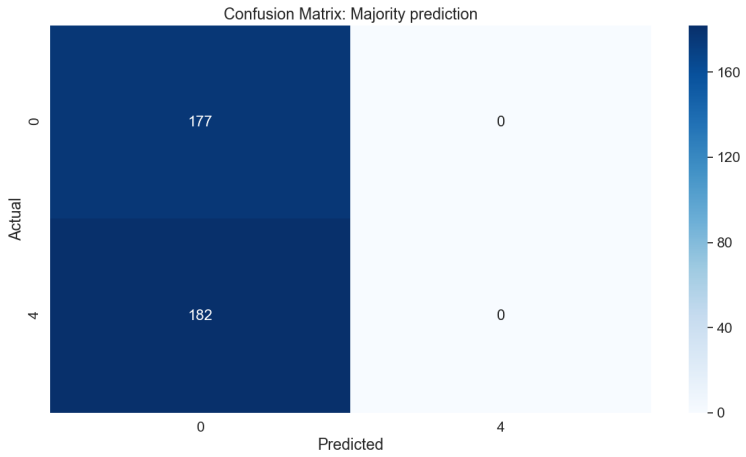

Figure 5: Confusion Matrix of Majority Prediction

1.1.5 Random Prediction

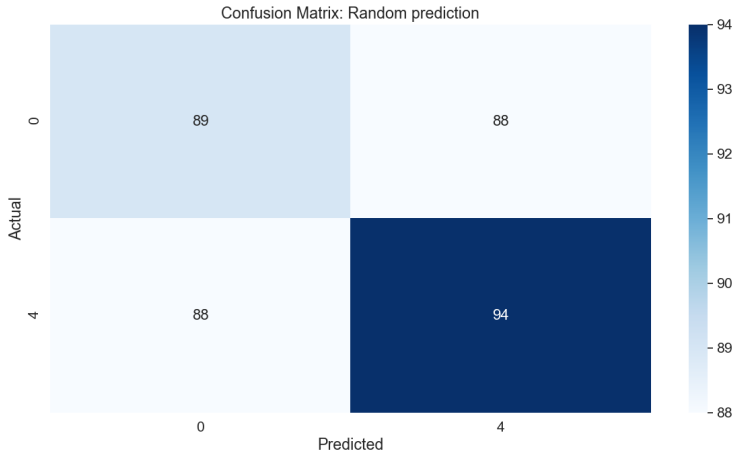

Figure 6: Confusion Matrix of Random Prediction

1.1.6 Comments

Majority prediction and Random prediction gives around 50 percent accuracy while Train and Test accuracy is around 80 percent. So there is difference of around 30-35 percent in accuracy's.

Train and test has the highest diagonal entry in confusion matrix.

Random Prediction has almost equal entry in all four boxes, because it has equally likely predicted both class and in actual both class have have almost same number of examples

Majority prediction has only entry in 2 boxes, because it predicted only single class for all test example.

Diagonal of confusion matrix captures the correct prediction and anti diagonal captures the wrong prediction

1.2 Stemming/Stop-Word Removal

We first remove the twitter handle and punctuation's from tweets by using TweetTokenizer and custom regular expression then we stem the training data using Snowball stemmer, then we remove the stopwords from the tweets using nltk stopwords.

1.2.1 Parameters

Stemmer= Snowball

Tokenizer = TweetTokenizer and custom regex

Accuracy = 81.6155

Train time = 34.00 sec

1.2.2 Test

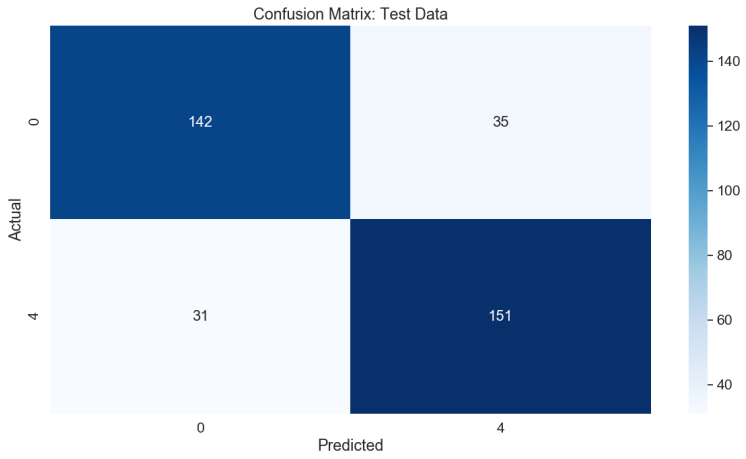

Figure 7: Confusion Matrix of Test

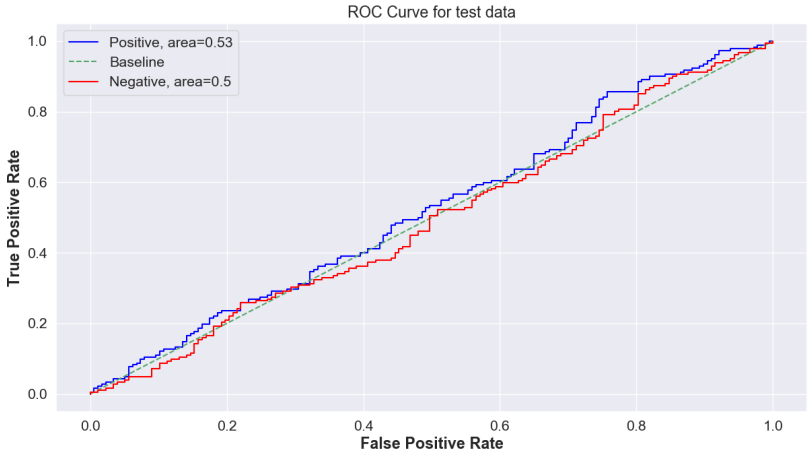

Figure 8: ROC of Test

1.2.3 Comments

Test accuracy increases by around 1 percent after stemming and stop word removals

1.3 Feature Engineering

1.3.1 Bigrams

We also included the bigrams of the tweets on the top of unigrams. For bigrams we use the CountVectorizer ngrams.

Accuracy on Train Data = 89.4160

Accuracy on Test Data = 81.6156

Train Time = 105.00 sec

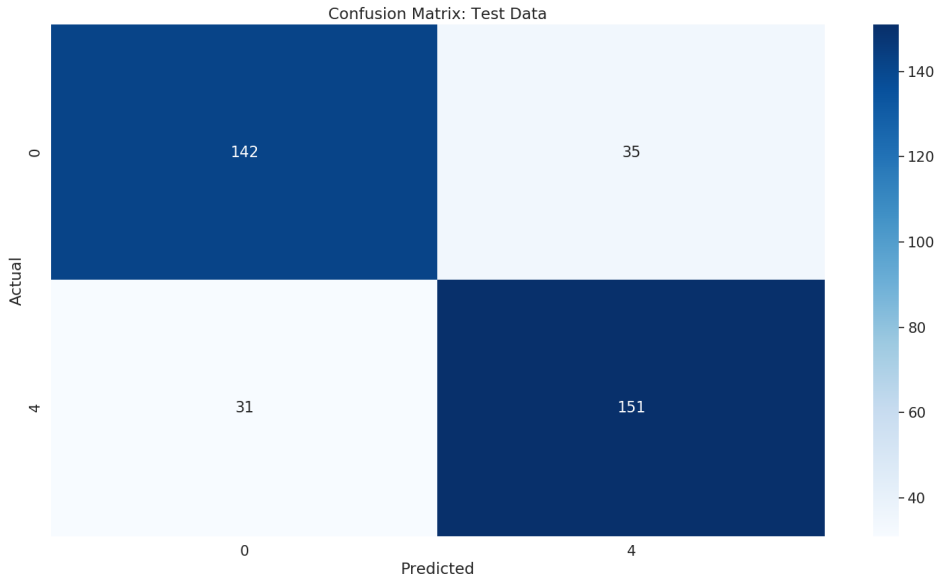

Figure 9: Bigram Confusion Matrix on Test

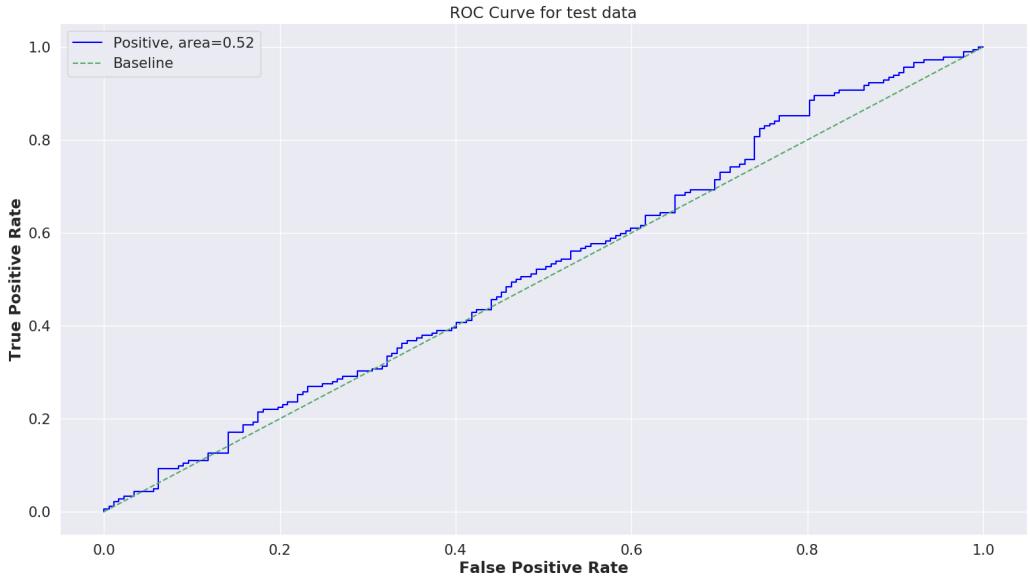

Figure 10: Bigram ROC on Test

1.3.2 Comments

By including Bigrams over unigrams, test accuracy improve slightly but train accuracy is increasing very much. So there may be case of overfit because training is more biased toward train data due to bigrams addition

1.3.3 POS Tag

In this, we uses the nltk POS tag library to tag the POS, then we remove the verbs from the tweets. After this we clean and do steaming and stop words removal from the tweet.

At the time testing, we do the same preprocessing to test example before testing

Accuracy on Train Data = 78.9149

Accuracy on Test Data = 70.1949

Train Time = 900.00 sec

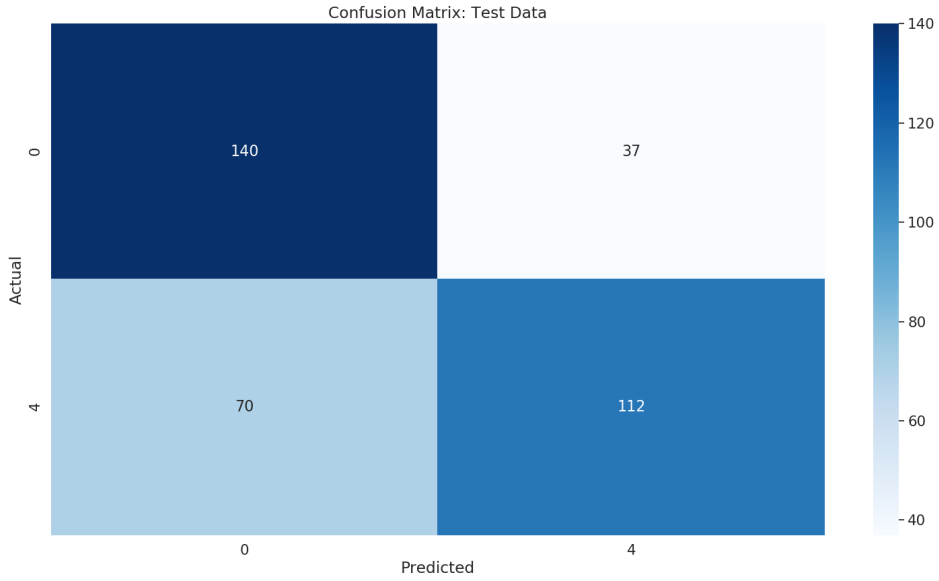

Figure 11: POS Tag Confusion Matrix on Test

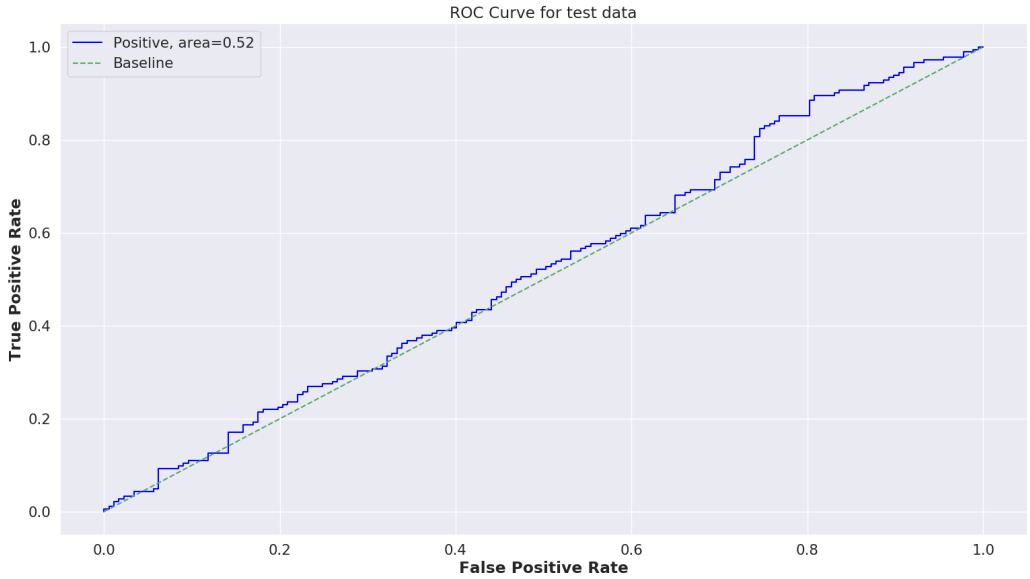

Figure 12: Bigram ROC on Test

1.3.4 Comments

By part of speech tagging, I am removing all kind of verbs and untagged word from documents. Accuracy is decreases to 70 percent, because we should also have to give some weight to all types of part of speech.

1.4 TF-IDF

In this part, we calculate TF-IDF for the cleaned tweet using TfIdfvectorizor. And at the time of testing we calculate TfIdf for cleaned test example and then test. Training time takes so much time so we used the min_dif parameter to ignore the less occurring words.
Also we uses the partialfit in the loop to fit all the examples.

Accuracy on Test = 78.5515
Training Time = 20 to 30 sec

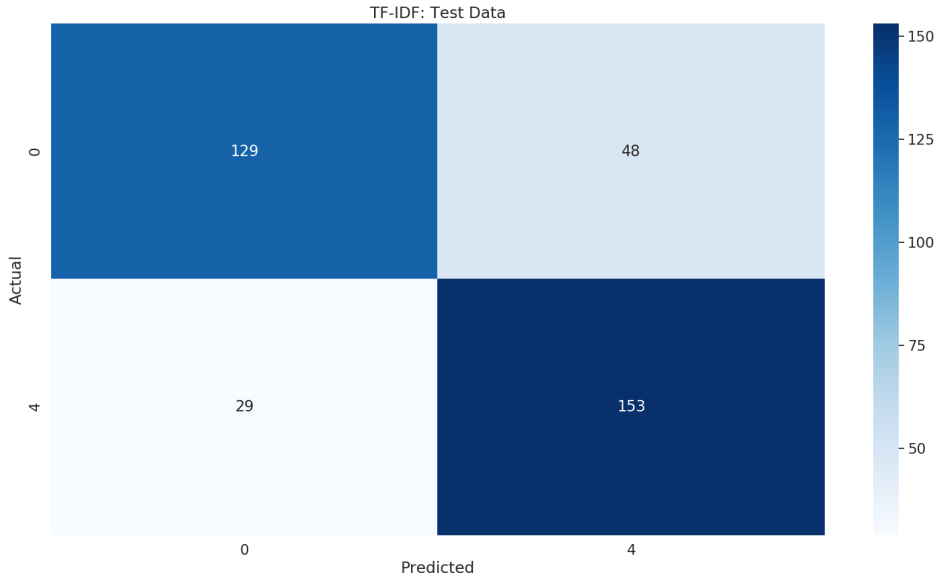

Figure 13: Confusion matrix on Test

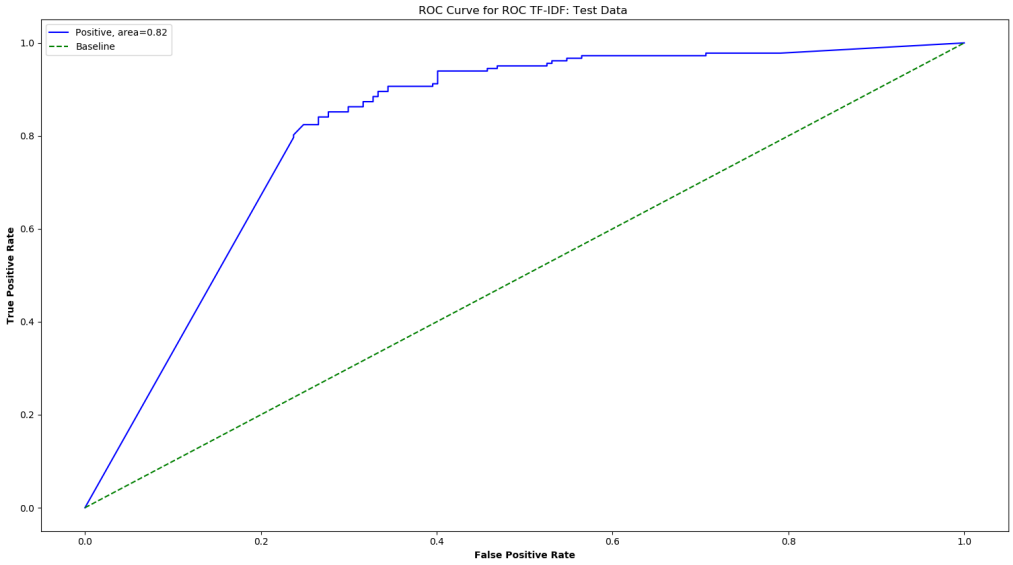

Figure 14: TFIDF ROC on Test

1.4.1 Comments

Using TF-IDF is computationally heavy because it generates the very large spare matrix. And when we convert it into dense it doesn't get fit into the RAM. Because the size of dense matrix it try to generate is order of Terabyte. So we have to use the partialfit to fit the train example.

1.5 TF-IDF SelectPercentile

1.5.1 P=2

Accuracy on Test = 68.8022

No. of features = 5832

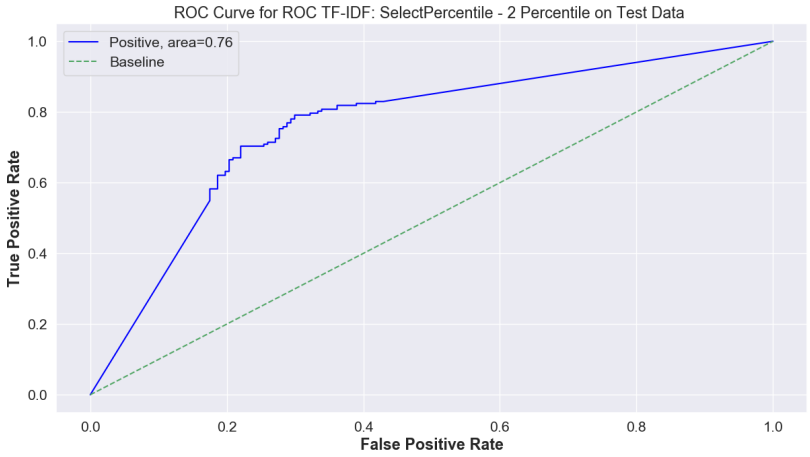

Figure 15: TFIDF Select Percentile ROC on Test

1.6 P=1

Accuracy on Test = 73.8166

No. of features = 2916

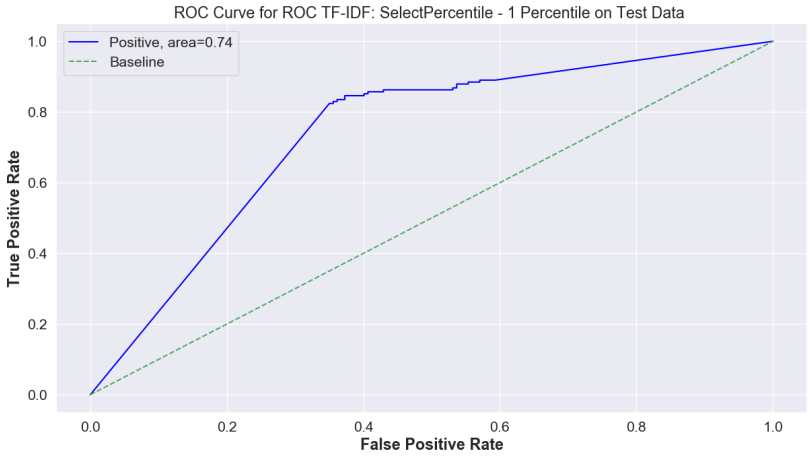

Figure 16: TFIDF Select Percentile ROC on Test

1.7 P=0.5

Accuracy on Test = 76.0445

No. of features = 1458

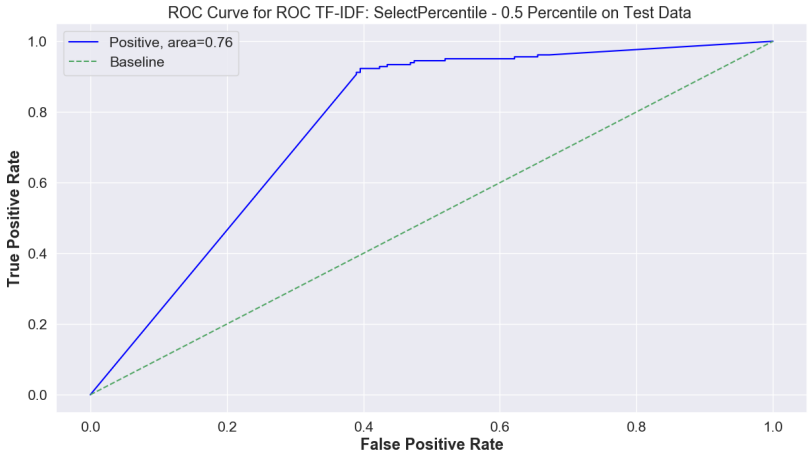

Figure 17: TFIDF Select Percentile ROC on Test

1.7.1 Comments

TFIDF training data is unable to fit in single fit function. So I used partial fit function in iterations to fit. Its Accuracy is 78 percent on test which is around 80. It doesn't improve the accuracy.

Using SelectPercentile with percentile=0.5 gives 76 percent accuracy. Increasing percentile from 0.5 to 2 decreases the accuracy from 76 to 68 percent. So we can conclude that selecting small number of features can also increase the accuracy. Because selecting large number of features may over fit to train data and does not generalizes much. In this case also we use partial fit because transformed dense test features cant fit into existing RAM

2 Support vector Machine

2.1 Binary Classification

Entry no. last digit: 5

Binary Classifier: 5 vs 6

2.1.1 Optimization Problem

$$max_{\alpha} W(\alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} y^{(i)} y^{(j)} \alpha_i \alpha_j < x^{(i)}, x^{(j)} > \tag{1}$$

$$\alpha_i \geq 0 \tag{2}$$

$$\sum_{i=1}^{m} \alpha_i y^{(i)} = 0 \tag{3}$$

2.1.2 CVXOPT Optimization Problem

$$\min_{x} \frac{1}{2} x^T P x + q^T x \tag{4}$$

$$Gx \leq h \tag{5}$$

$$Ax = b \tag{6}$$

2.1.3 Parameters

$$x = \alpha \tag{7}$$

$$P = (y @ y^T) * (X @ X^T) \tag{8}$$

$$q = [-1, -1... - 1]^T_{1Xm} \tag{9}$$

$$G = [-1, -1..., 1, 1.., 1]^T_{1X2m} \tag{10}$$

$$h = [0, ..0, 1...1]^T_{1X2m} \tag{11}$$

$$A = y^T \tag{12}$$

$$b = 0 \tag{13}$$

2.1.4 Linear SVM - Implemented

Classes = 5 vs 6

Training Time = 29.8138 sec

No. of support vectors = 86

b = -0.49050306

Train Accuracy = 1.0

Val Accuracy = 99.8

Test Accuracy = 99.8

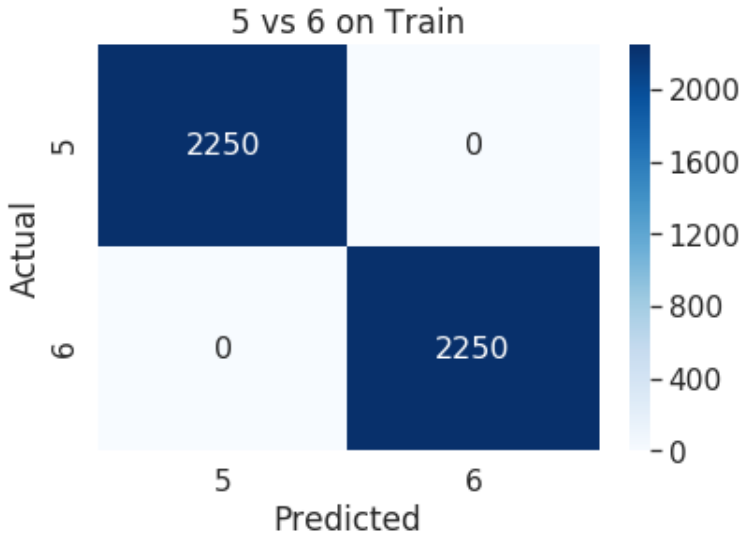

Figure 18: 5 vs 6 Train

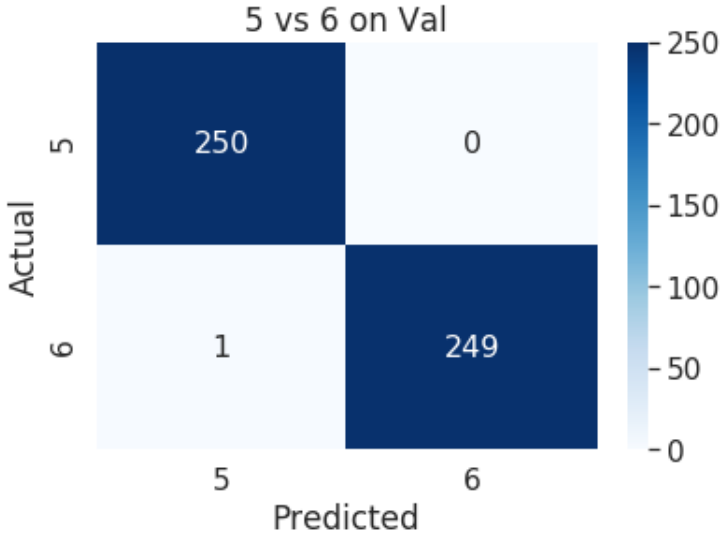

Figure 19: 5 vs 6 Validation

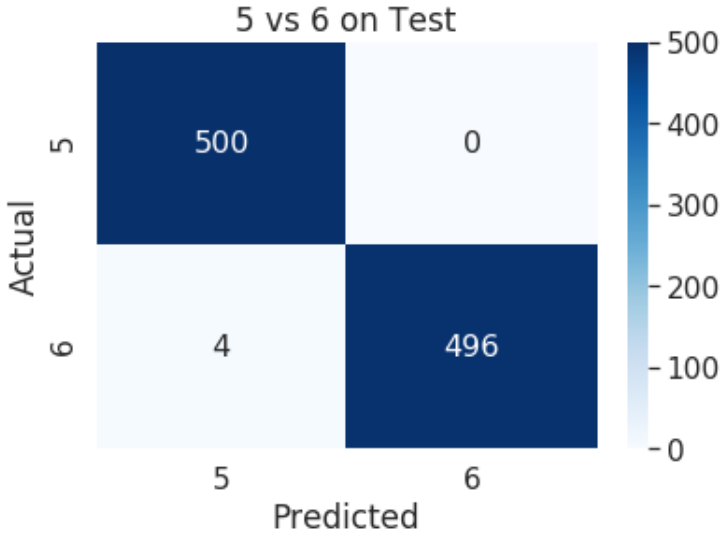

Figure 20: 5 vs 6 Test

2.1.5 Linear SVM - Sci-kit Learn

Classes = 5 vs 6

Training Time = 0.404498 sec

No. of support vectors = 85

b = -0.49029701

Train Accuracy = 1.0

Val Accuracy = 99.8

Test Accuracy = 99.6

2.1.6 Comments

Norm distance between w of sklearn and our implementation is 0.00085485, that means our w matches with that of sklearn.
b difference of our and sklearn's is 0.00020659, means b is also almost same.
Accuracy of SkLearn's and our implementation is also almost same. Test accuracy of our implementation is slightly better.
w is a vector of size n(no. of features), b is scaler 0.

2.1.7 Gaussian SVM - Implemented

Classes = 5 vs 6

Training Time = 34.11 sec

No. of support vectors = 85

b = -0.08654

Train Accuracy = 1.0

Val Accuracy = 99.8

Test Accuracy = 99.6

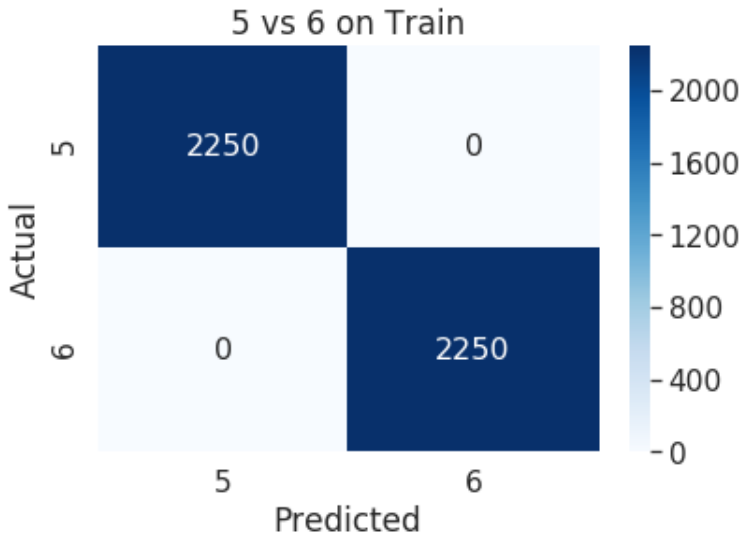

Figure 21: 5 vs 6 Train

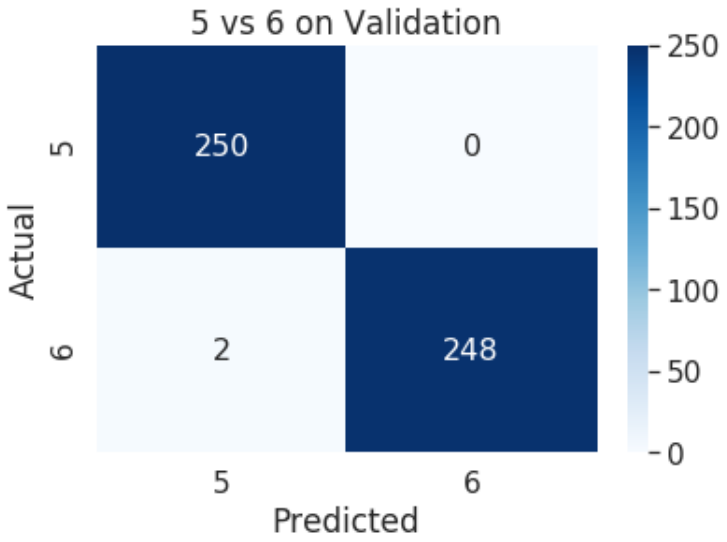

Figure 22: 5 vs 6 Validation

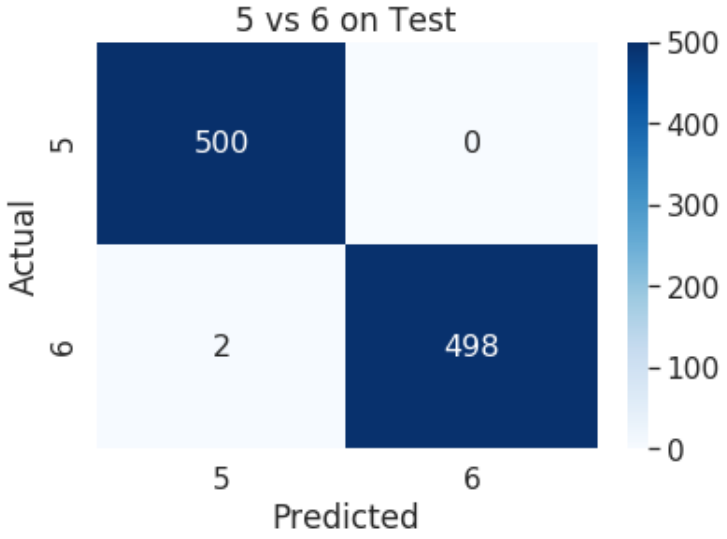

Figure 23: 5 vs 6 Test

2.1.8 Gaussian SVM -Sci-kit Learn

Classes = 5 vs 6

Training Time = 3..634 sec

No. of support vectors = 723

b = -0.122641

Train Accuracy = 1.0

Val Accuracy = 99.6

Test Accuracy = 1.0

2.2 Comments

Accuracy of Gaussian and Linear is almost same for our class 5 and 6, and it is close to 1. Class 5 and 6 are linearly separable.

No. of support vectors increases in Gaussian Kernel.

b value between our implementation and sklearn's differs, but slightly.

2.3 Multi-class Classification

In multi-class classification, we use the one vs one classifier. So we have to train total ${}^{10}C_2 = 45$ classifiers first.

Then at the time of testing, for each example, we test on the each 45 classifiers, then we output the class which has highest number of votes out of 45 votes. In case of ties we break ties with highest absolute sum $w^T x + b$. Means if in case of two classes have same number of highest votes then we sum the absolute value of $w^T x + b$ of 1^{st} highest class and 2^{nd} highest class separately and output classs which has the highest sum.

2.3.1 Implemented

Test Accuracy = 85.06

Test F1 Score = 85.135

Validation Accuracy = 85.04

Validation F1 Score = 85.0636

Train Time = 900 sec

Prediction Time on Test = 1331 sec

Prediction Time on Validation = 684 sec

Break Ties: with largest absolute sum of $w^T x + b$

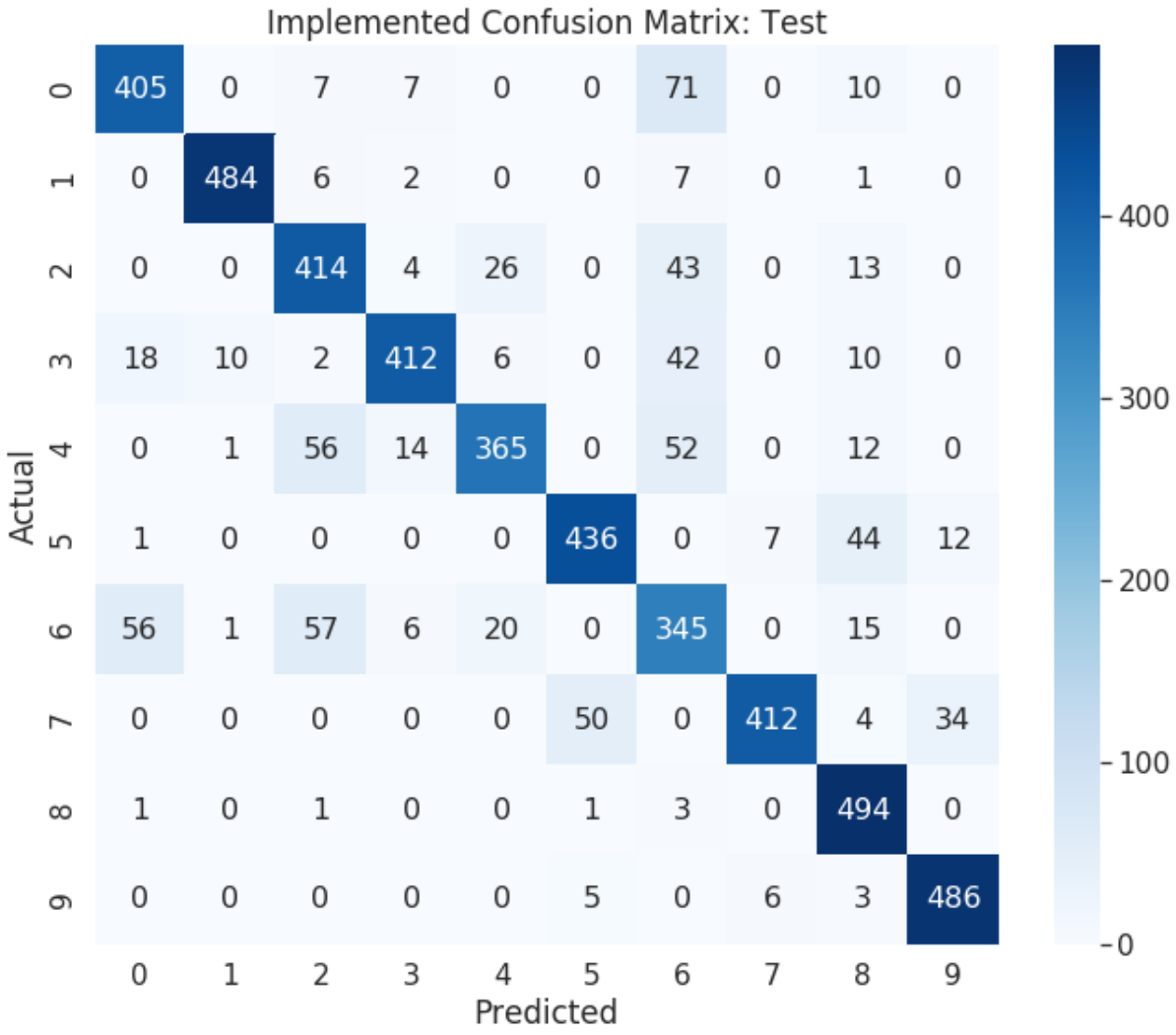

Figure 24: Confusion Matrix Test

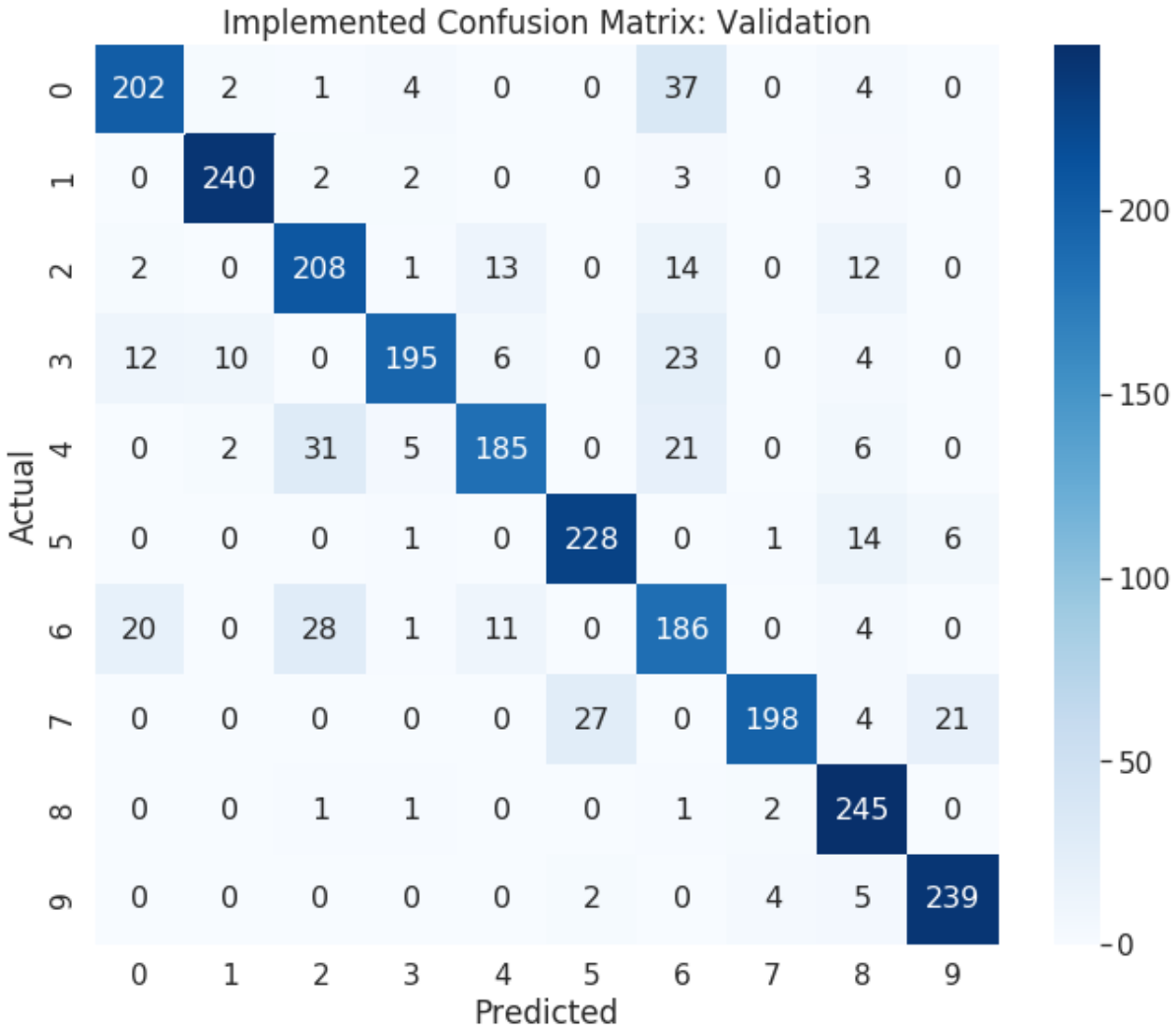

Figure 25: Confusion Matrix Validation

2.3.2 Sci-Kit Learn

Test Accuracy = 88.08

Test F1 Score = 87.980

Validation Accuracy = 87.88

Validation F1 Score = 87.8294

Train Time = 673.22587 sec

Prediction Time = 324.9282 sec

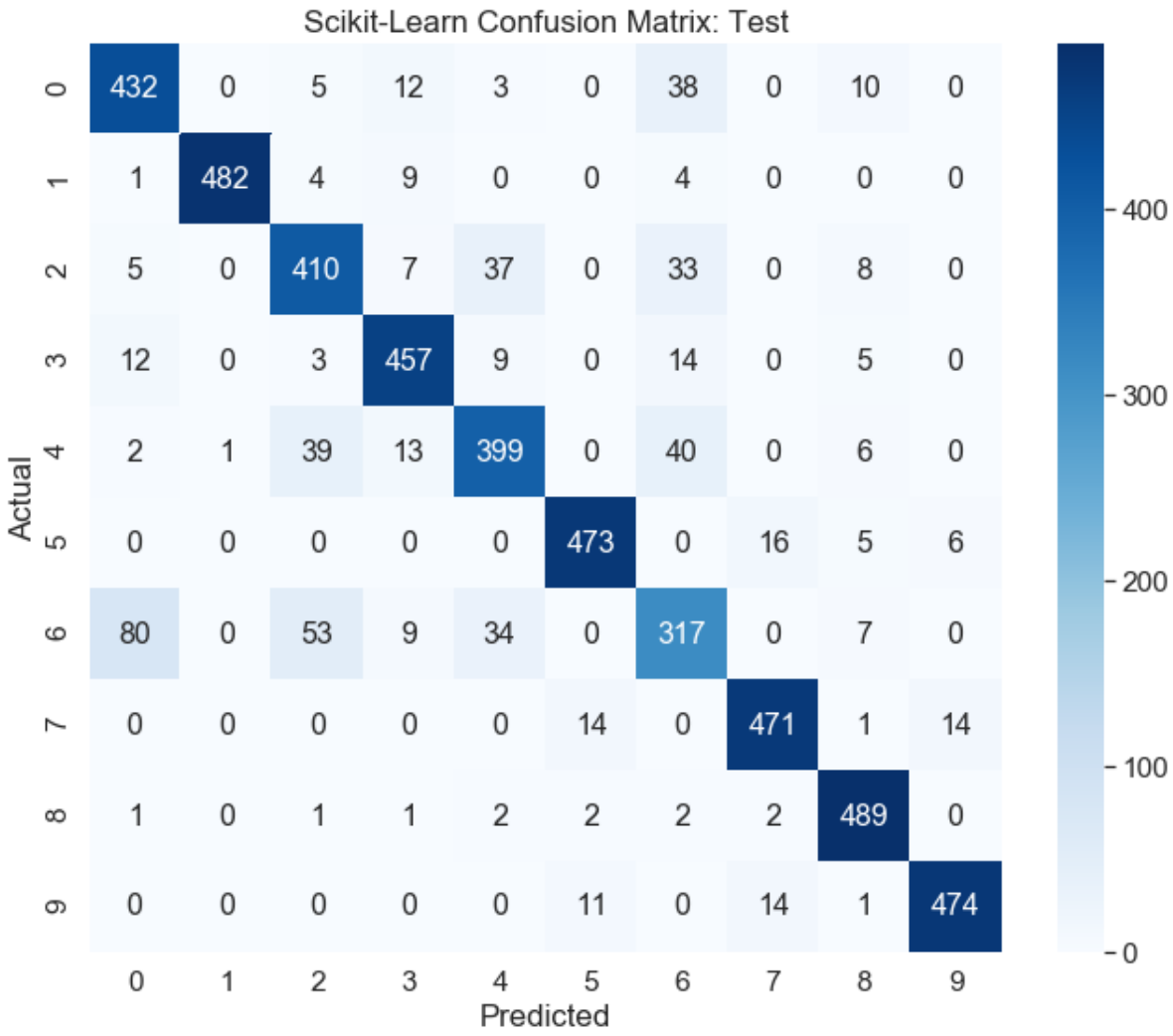

Figure 26: Confusion Matrix Test

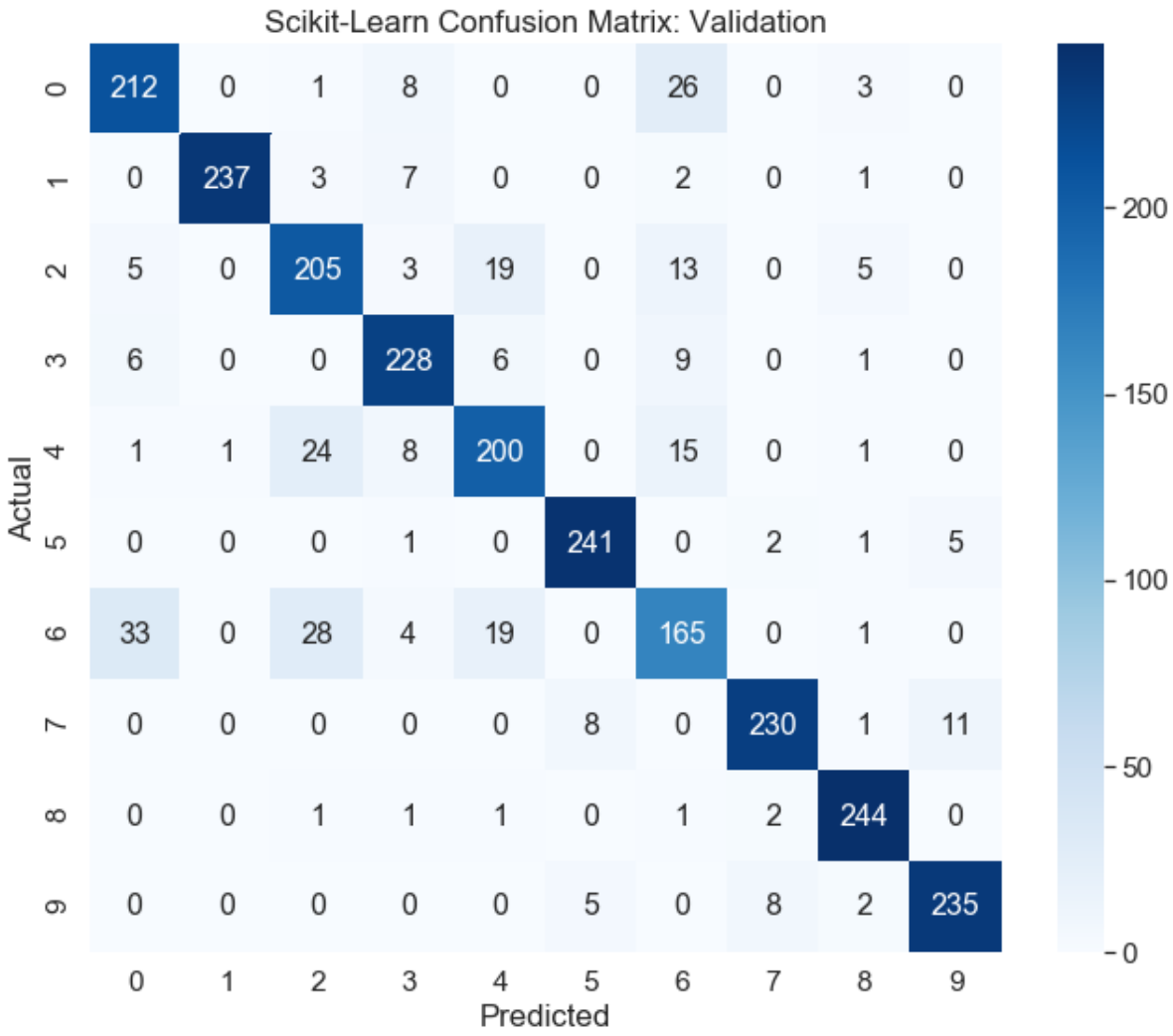

Figure 27: Confusion Matrix validation

2.3.3 Comments

Accuracies with SkLearns is 2 to 3 percent better than our implementation. Maximum 6 is classified into 0 class. It is misclassified 33 times to that class. Because 0 is T-shirt/top and 6 is shirt. Both looks similar, so they have high chance of being get misclassified.

Next maximum misclassified is 6 misclassified as class 2, and 2 is pullover so it also looks like the shirt.

Overall, class 0,2,6,4 are misclassifed maximum times because, they all looks somewhat same type intuitively.

Trousers and Bags are minimum misclassified to other classes

2.4 K Fold cross validation

We have use the sci-kit learn GridSearchCV which takes cv parameter which determines the cross-validation splitting strategy. I pass $cv = 5$. It finds the best parameter for the data. I have passed parameters as $C = [10^{-5}, 10^{-3}, 1, 5, 10]$

2.4.1 Parameters

Best parameter: C = 5
Time for Grid Search:3340.544 sec
Accuracy on Test with C=5:88.28
F1 score on Test with C=5:88.28
Accuracy on Val with C=5: 88.28
F1 score on Val with C=5:88.83

After finding Best C we trained 5 model for all 5 Cs and plot accuracy vs C for test and validation data

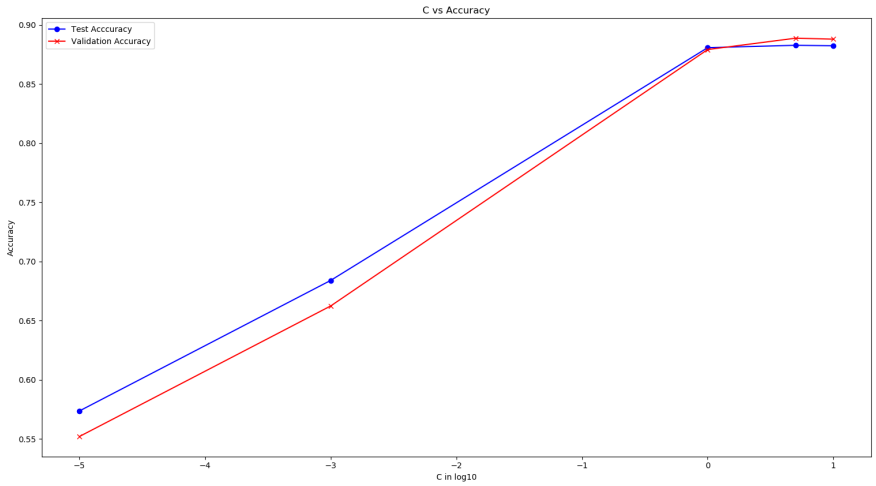

Figure 28: C vs Accuracy

2.4.2 Comments

We get the best accuracy on test set at C=5. At C=10, test accuracy decrease slightly. But Validation set accuracy increases.

We can observe that changing C from 5 to 10 decreases test accuracy slightly but increases validation accuracy slightly.