AC50001 ASSIGNMENT 2

CLASSIFICATION AND CLUSTERING

Solution Report

1. The code for this part is saved in part1.mat. It uses Kmeans clustering after applying PCA to reduces the images of 784 pixels each to 2pixel (dimensions). The scatter plots generated by program is shown below:

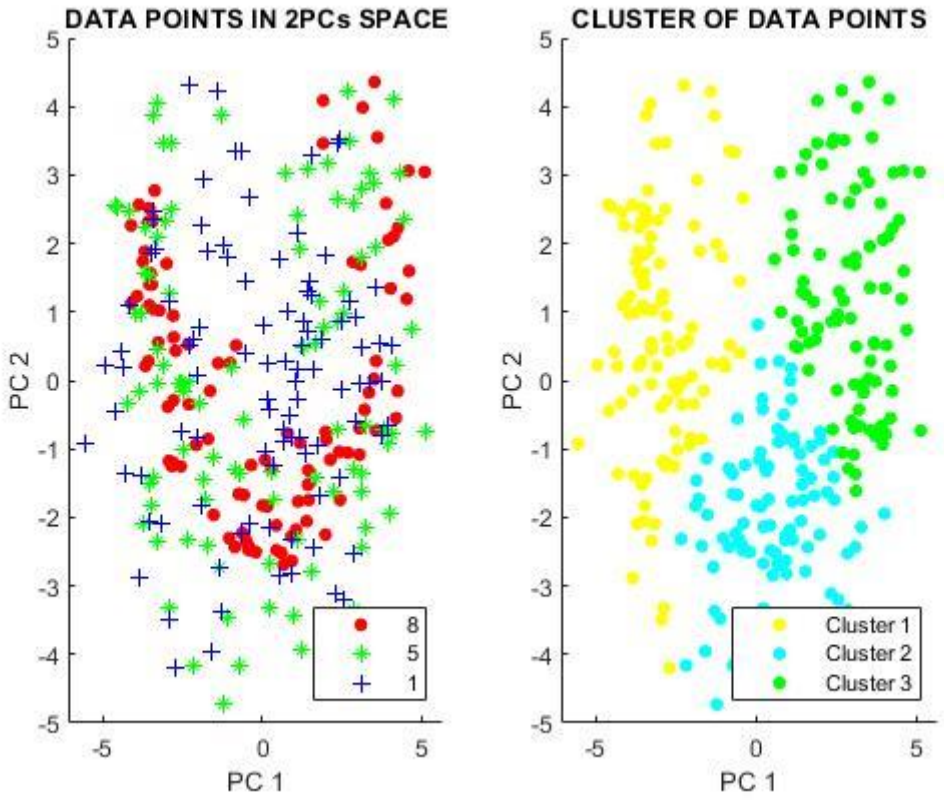


Figure 1a: *Scatter plots of points for each digit and the clustering.*

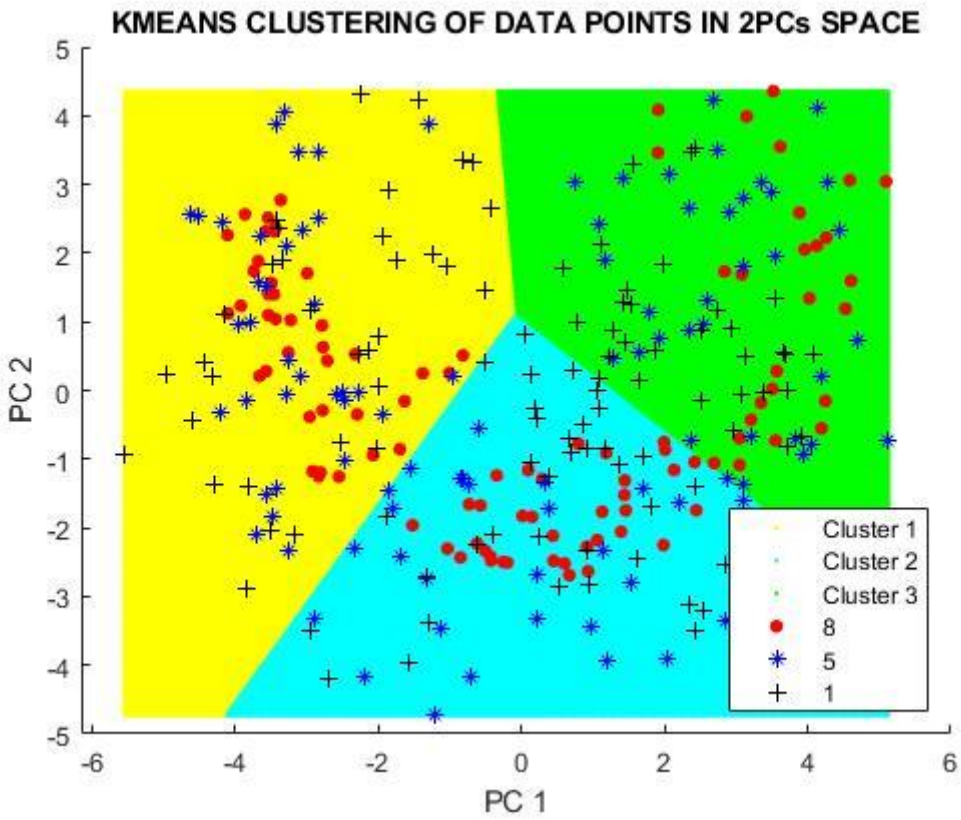


Figure 1b: *A scatter plot of points for each digit and the cluster in they belong.*

From the scatter plots, kmeans could not separate the data points into distinct clusters because points from the same digit class were far away or points from different digits overlapped. It simply found three clusters which does not group the digits into classes. The confusion matrix generated by the program also illustrate this.

DIGIT/CLUSTER	1	2	3
8	36	34	30
5	35	30	35
1	38	36	26

Table 1: Confusion Matrix

The first two (2) PCs showed the direction of maximum spread for each digit class but there is no seperation between the classes for meaningful custers to be formed by kmeans (it uses the minimum distance of a point from the centroids of clusters).

2. Here, LDA is used for dimensionality reduction before applying kmeans clustering. The scatter plots generated are given below:

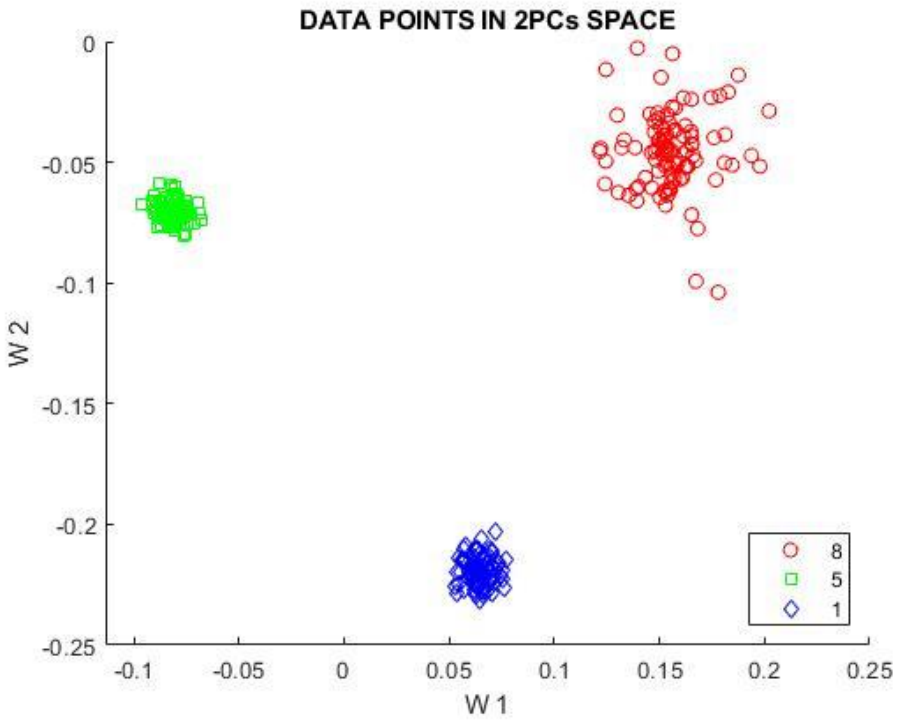


Figure 2a: *Data points after applying LDA for dimensionality reduction*

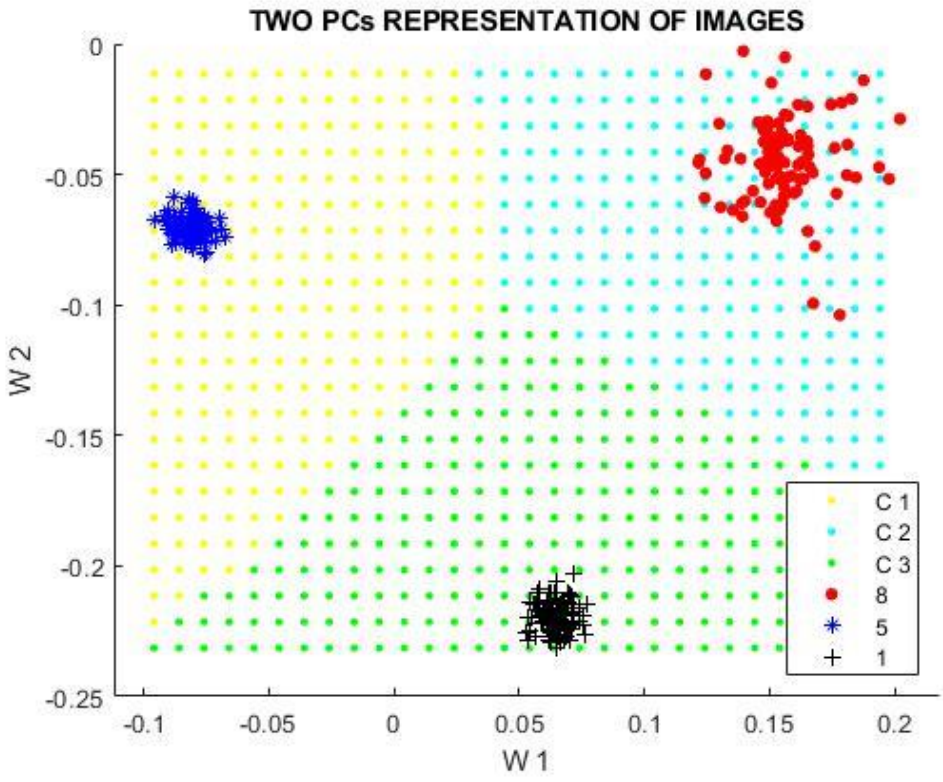


Figure 2b: *Data points and the cluster they are assigned to after clustering*

With LDA, the data belonging to a class was clearly separated as shown figure 2a. Hence, kmeans perfectly assigned every data point in each class to the same cluster. Digit 8 to cluster 2, digit 5 to cluster 1 and digit 1to cluster 3.

LDA made the data optimal for clustering (figure 2a) as compared to PCA (figure 1a). Most clustering algorithms (not only kmeans) rely on minimum distance metric between data points for clustering, figure 2b proves that the principle of minimizing within-class scatter and maximizing between-class scatter enhances clustering performance. This shows that LDA is a better dimensionality reduction approach for high dimensional data before clustering.

3. Method Description: Using linear kernel and RBF Support Vector Machine and a neural network with one hidden layer classifiers digit '5' is separated from '1' and '8'. Afterwards, ROC and AUC is used to compare the performance of these classifiers.

Background: Support vector machines (SVMs) are two class classifiers that use a decision boundary and support vectors (SVs) for classification. They seek to maximize the margin between decision boundary to SVs while minimizing wrong classification of data. To achieve this, the kernel trick is used. The kernel trick uses different kernel function for mapping the data set to a higher dimension. Particularly:

Linear kernel function=$x_i.x_j$
Radial basis function(rbf)=$\exp(-\| x_i.x_j \|^2 /2\alpha^2)$. RBF kernel has two parameters, C and gamma. It is good to have a balance between these parameters.

i. Gamma controls the decision boundary and invariably the width of the kernel. If gamma is low, the decision boundary is smooth curve and has a broad region for each class. If gamma is high, the decision boundary is wobbly, and the classifier tries to fit to data points, and we see regions around points (overfitting).

ii. C is the cost of misclassification. A low C means there is less penalty for misclassification while a high C means the classifier is heavily penalized for misclassified data.

A neural network uses a training data and the desired output for learning in other to make predictions when applied to any new data. Neural networks have layers and in each layer are many connected units called neurons. Using the weights of neurons and an activation function a neural network is trained and can be used to make prediction.

Implementation: The files **part3a.m, part3b.m, part3c.m** represent codes for SVM with linear kernel, SVM with RBF, and neural network classifier respectively.

- **SVM classifiers**: libsvm library for Matlab is used. The method **svmtrain()** is used to build the models and **svmpredict()** is used to classify test data into a class.

```
svmtrain(training_label_vector,training_instance_matrix, '-t 0 -b 1 -q');

svmtrain(training_label_vector,training_instance_matrix, '-t 2 -g 0.07 -c 1 -b 1 -q')
```

  Where

  -t: is parameter for type of kernel (0 for linear and 2 for RBF)

  -b: is parameter for decision probabilities to be generated by method

  -q: is parameter to execute method without output messages

  -g: RBF gamma parameter

  -c: Cost parameter. Default=1. *(Parameter tuning is only applied on RBF -c and -g are tested)*

- Neural Network: **patternnet(1)** where 1 is one hidden layer is used.

- Cross Validation: Each program uses a 5-fold cross validation for training and test sets; each saves a matrix of probabilities for roc curve calculation, displays an overall confusion matrix and classification accuracy. **cvopartition()** is used to split data into five-folds of training and test sets. To implement cross validation, a for-loop is used for training model and testing five times.
- ROC and AUC: The files **plotroccurve.m, roc.m** implements these. roc.m is a function which returns true positive rate (TPR), false positive rate (FPR) and AUC of each classifier. Plotroccurve.m uses the probability matrices created by the three classifiers and **roc.m** to plot curve.

Perform Classification: Place all source codes and "AC50001_assignment2_data.mat" in Matlab's current folder. To run **part3a.m, part3b.m, part3c.m** open source programs in Matlab and execute normally. Libsvm library need to be downloaded and installed, and installation folder added to Matlab's path.

	0	1
0	191	9
1	8	92

Linear Kernel SVM
Accuracy=0.9433

	0	1
0	194	6
1	2	98

RBF Kernel SVM
Accuracy=0.9733

	0	1
0	195	5
1	4	96

Neural Network
Accuracy=0.9700

ROC Curve and AUC: To run **plotroccurve**.m open source program in Matlab and execute normally. ROC curve for classifiers is shown in figure 3a below:

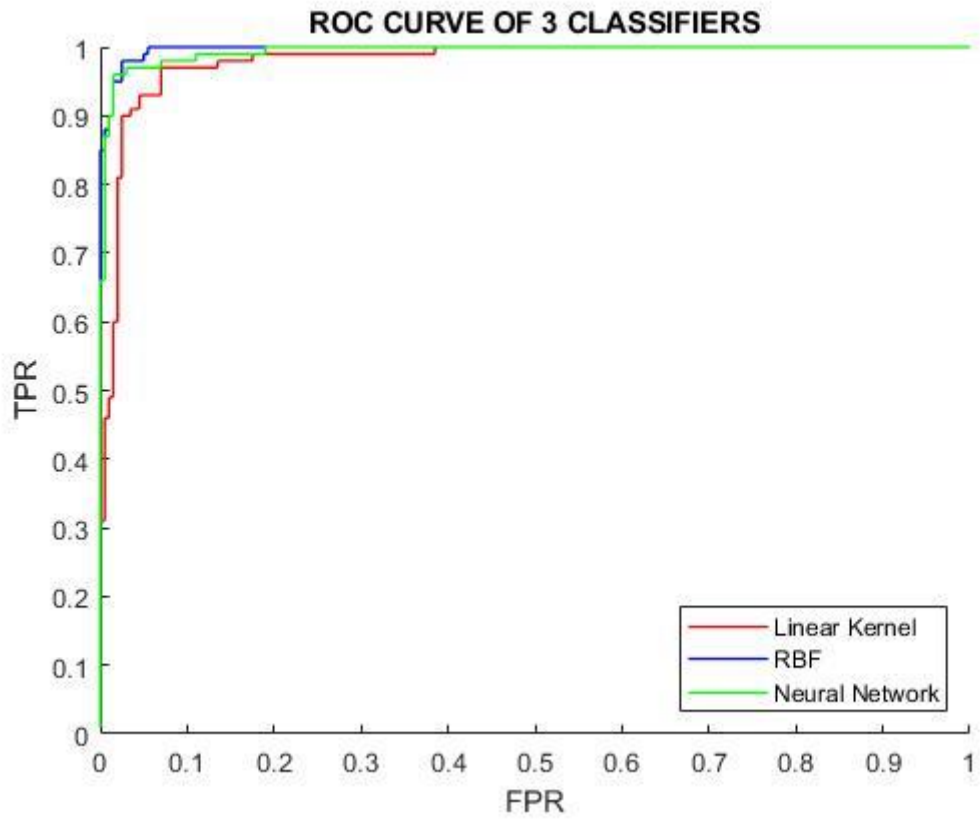


Figure 3a: *ROC curve for classifiers*

Where AUC (area under the curve) =sum(area(TPR,FPR))=trapz(FPR,TPR);

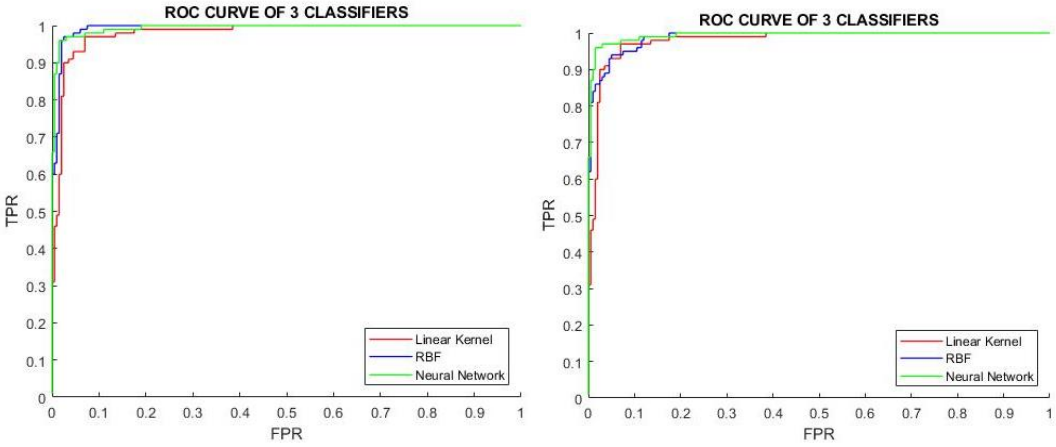
- Linear Kernel AUC=0.9799
- RBF AUC=0.9971
- Neural Network AUC=0.9938

From the plot, RBF classifier and Neural Network (NN) having 0.0 FPR until at TPR 0.7 where NN have an FPR ~0.01. NN had a slightly higher TPR at 0.96 but RBF maintained higher TPR afterwards.

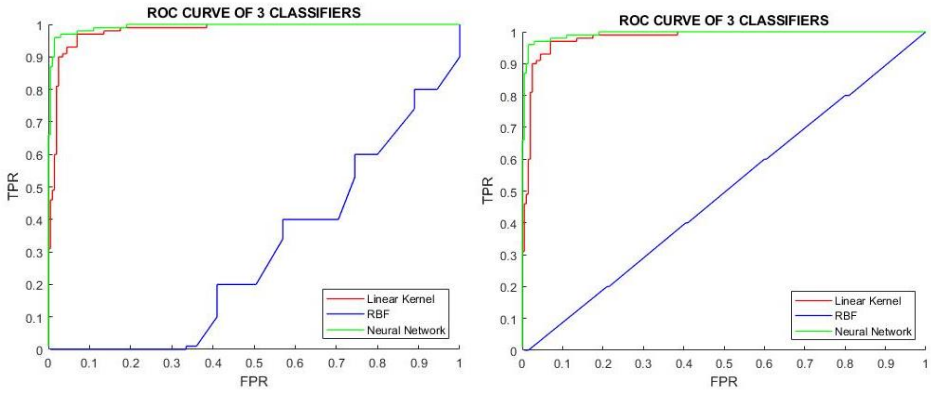
The ROC curve showed that both RBF and Neural Network classifiers performed better than Linear Kernel classifier and RBF was a better classifier to Neural Network. This is further proven by the AUC values. RBF had the highest AUC value hence the overall best performance.

Tuning RBF Classifier:

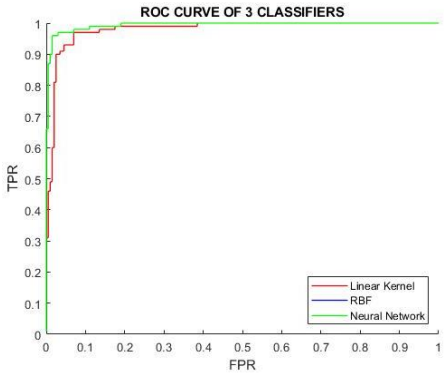
To test the effect of gamma, C is 1 by default, while gamma is varied:



3b: gamma=0.01 and gamma=0.1



3c: gamma=1 and gamma=10

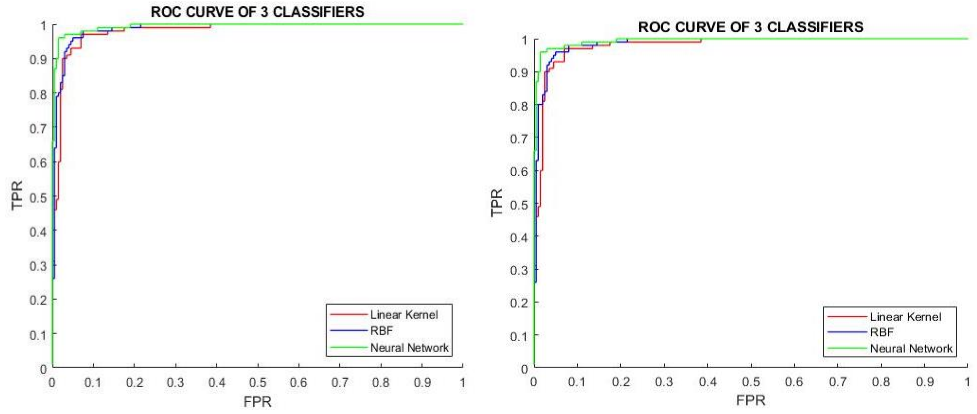


3d: gamma=100

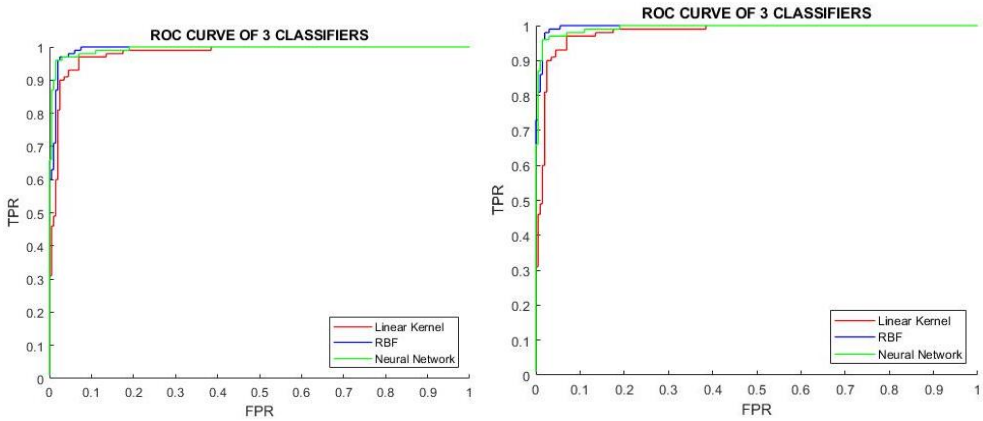
When gamma is low and there is a broad decision boundary, the classifier is performing well on test data set. As gamma get high, the classifier ties to fit to

the data as seen in gamma=0.1,1,10 and 100. At 1, gamma has no effect on the **W** as such, undecisive and very poor classifier, at 10, the model is overfitted and performs poorly. At 100, the data is overfitted and no data point test data is classified correctly.

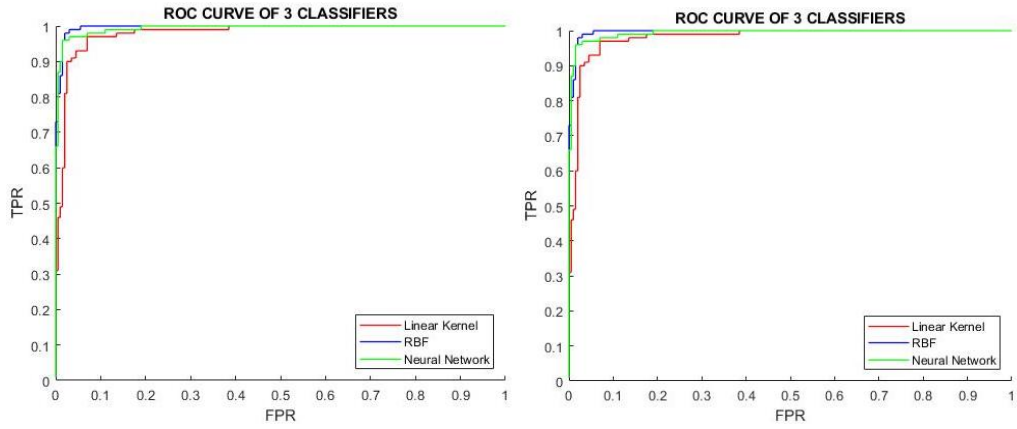
To test the effect of C, we keep gamma=0.01, and C is varied:



3e: C=0.01 and c=0.1



3f: c=1 and c=10



3g: c=100 and c=1000

When C is low, there is tolerance for misclassification as such we see RBF classifier with an AUC less the Neural network at c=0.01,0.1 As C increases, the classifier is strict on low misclassification and as such classification performance is better, however performance after C=10 does not change. RBF Curve at c=10,100 and 1000 is the same.

Conclusion: RBF classifier shows better performance than neural network while linear kernel classifier has the least performance. RBF classifier parameters can be tuned to improve performance. A good range for gamma is (0.01,0.1). A good range for C is (1,100).