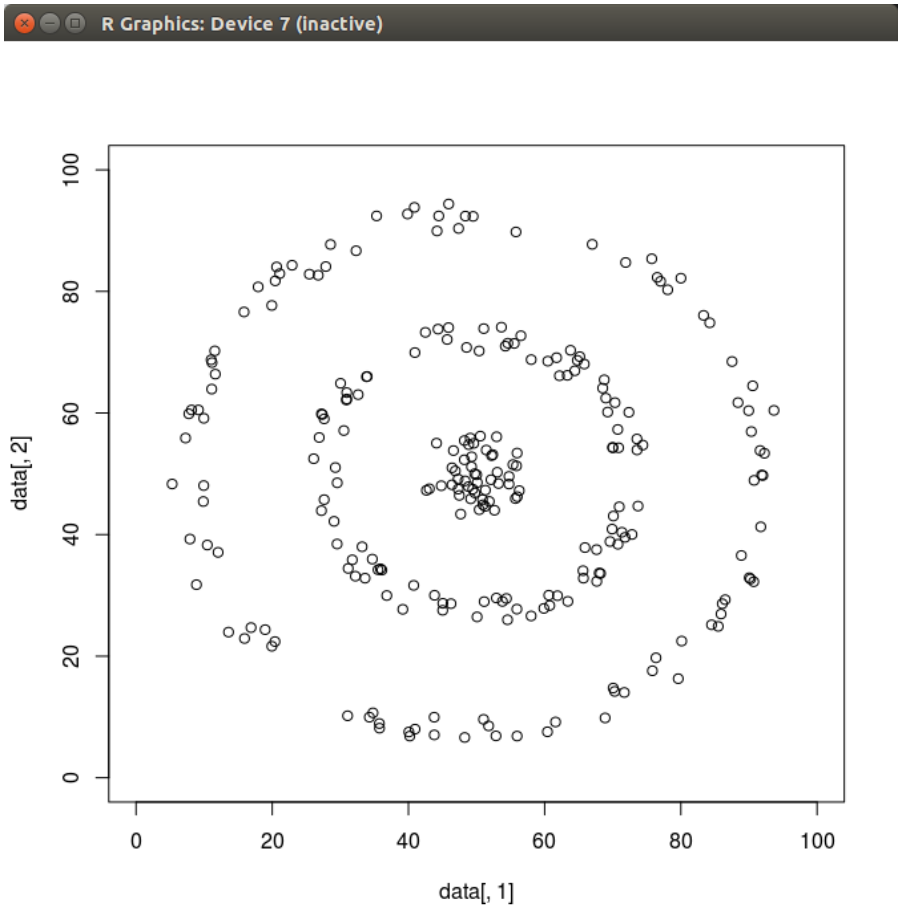
Name: Rakesh Gopal Kavodkar
UnityID: rgopalk
StudentID: 200066020

NOTE 1: For SVM, the library used is *e1071,* for the rest it is *kernlab*. The metrics were calculated separately by taking the original cluster IDs and the predicted cluster IDs from the algorithms, since some of the APIs used does not directly give the predicted cluster IDs in their object attributes but print it on the console. The metric calculation was done in Python. The algorithm to calculate the metrics was taken from the web. The *performance.py* file is attached with the others source files

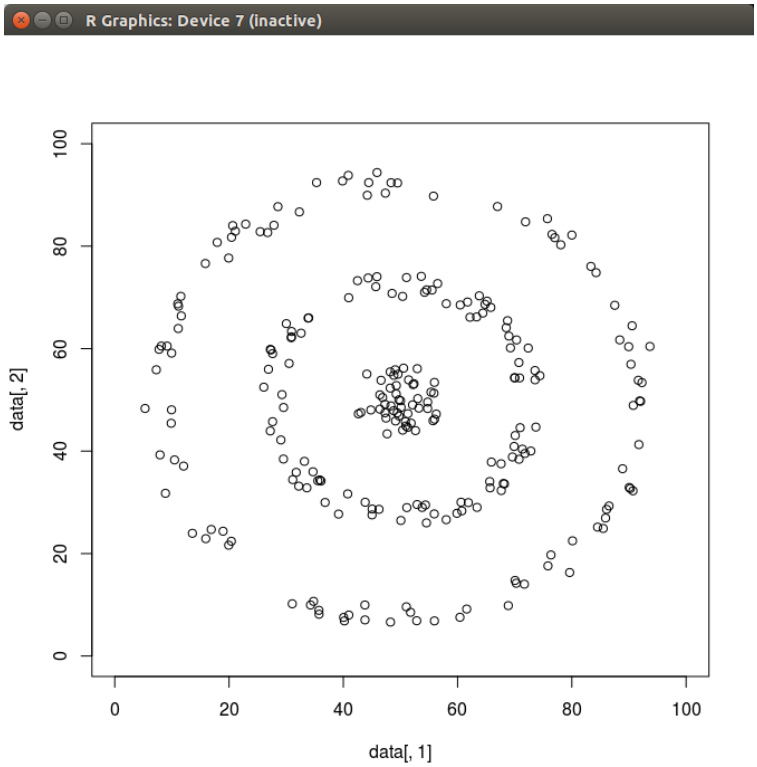
NOTE 2: Please find attached the source code files in the folder named *code*. For each algorithm, there is a different file. {KMeans: kmeans.r, PCA: pca.R, SVM: svm.R, problem 4: problem_4.R}

NOTE 3: The same data set works bad for all three of the data sets

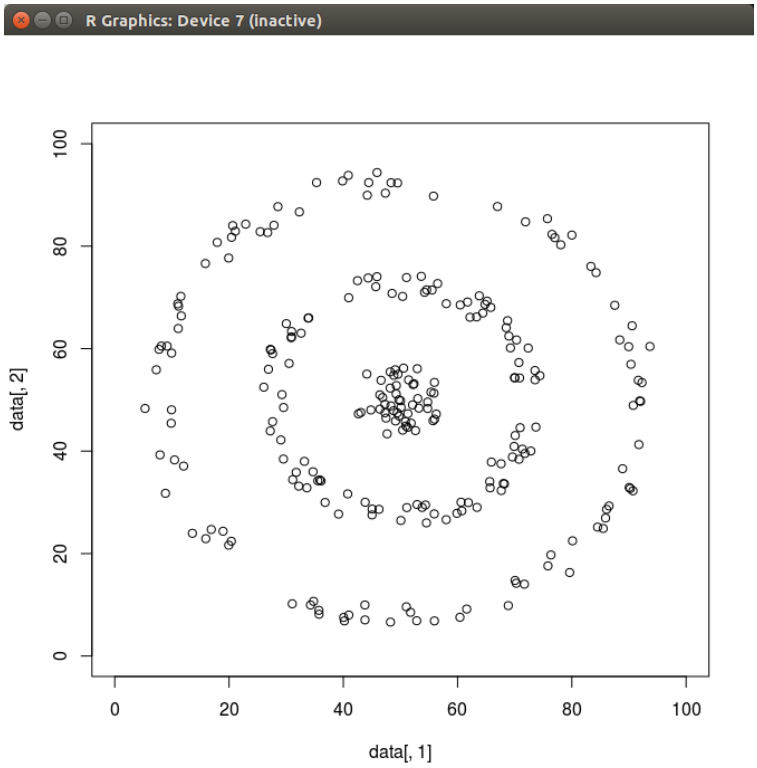
1. Bad data sets for the following methods
   a. Bad KMeans: Two concentric circular points and a gaussian set points in the middle



b. Bad PCA: Two concentric circular points and a gaussian set points in the middle



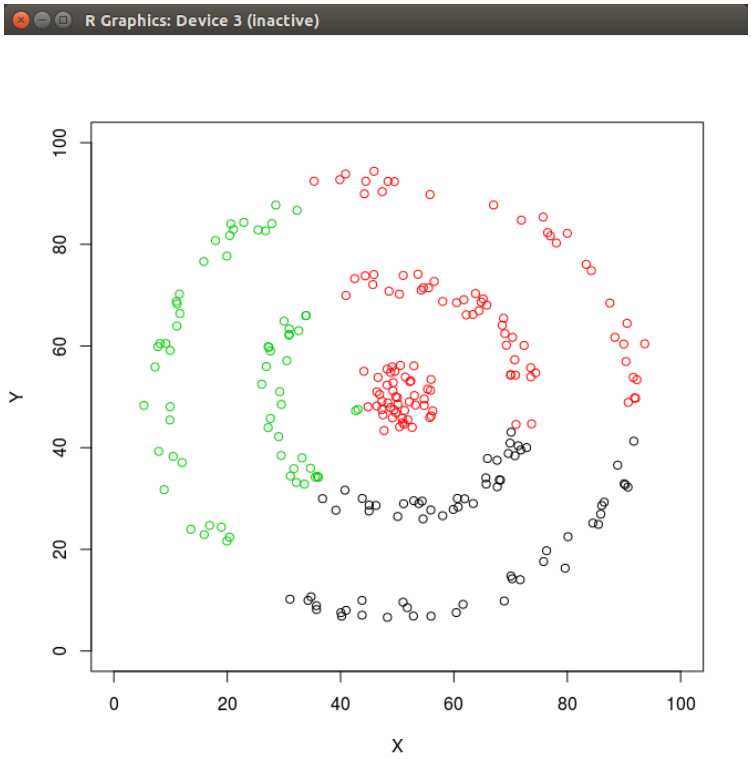
c. Bad SVM: Two concentric circular points and a gaussian set points in the middle



d. The plots are represented above.

2. The plots below represents the corresponding tasks without kernelization

a. Bad KMeans

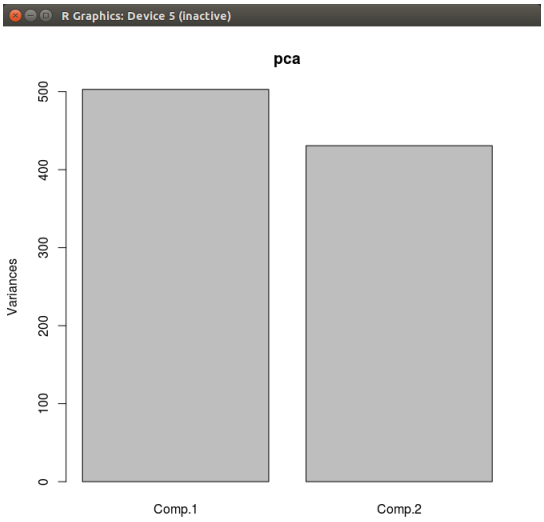


The metrics for the above run was:

Accuracy: 0.569
Precision: 0.396
Recall: 0.395
F-Score: 0.396

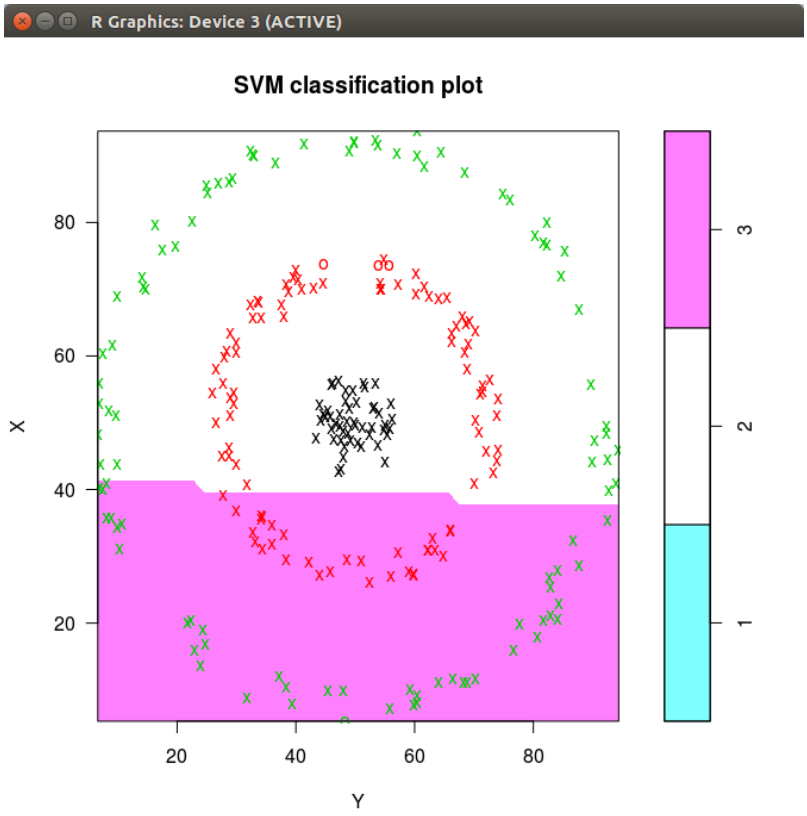
b. Bad PCA

	Comp 1	Comp 2
Variance	0.5386317	0.4613683



c. Bad SVM:

Used the “svm(..)” function from library “e1071” to get he below plot.



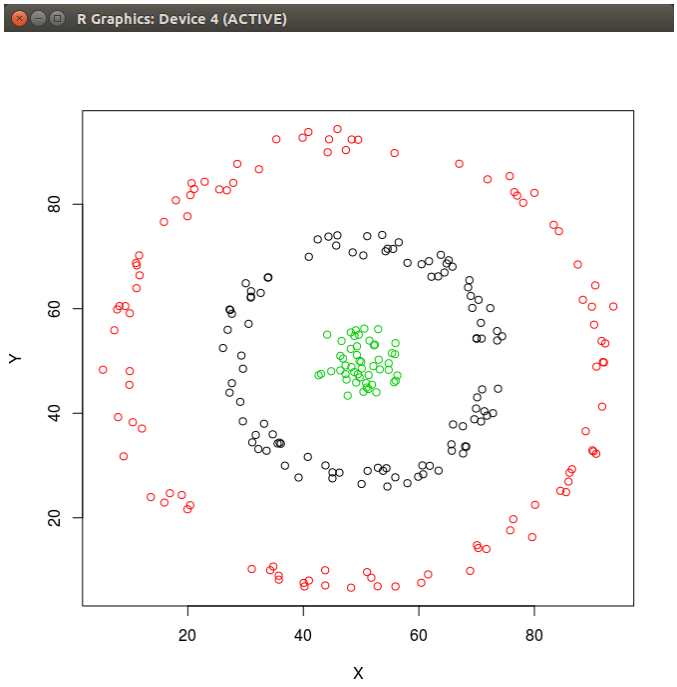
Accuracy: 0.483
Precision: 0.362
Recall: 0.586
F-Score: 0.448

3.

a. KMeans:

Upon applying the RBFDOT kernel for the Kernel Kmeans clustering, we get the below metrics and the corresponding graph

Accuracy: 1.0
Precision: 1.0
Recall: 1.0
F-Score: 1.0



Upon applying the POLYNOMIAL kernel for the Kernel Kmeans clustering, we get the below metrics and the corresponding graph

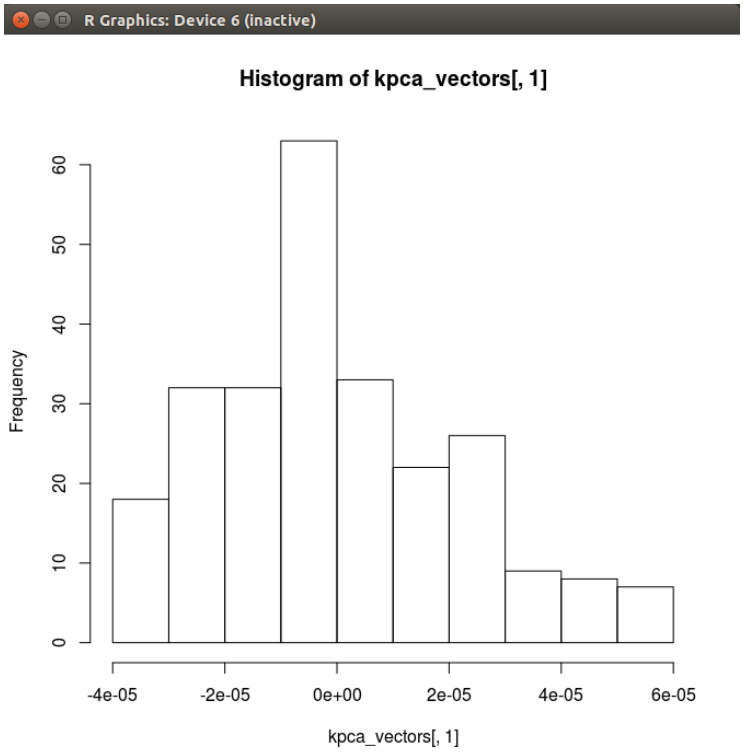
Accuracy: 1.0
Precision: 1.0
Recall: 1.0
F-Score: 1.0

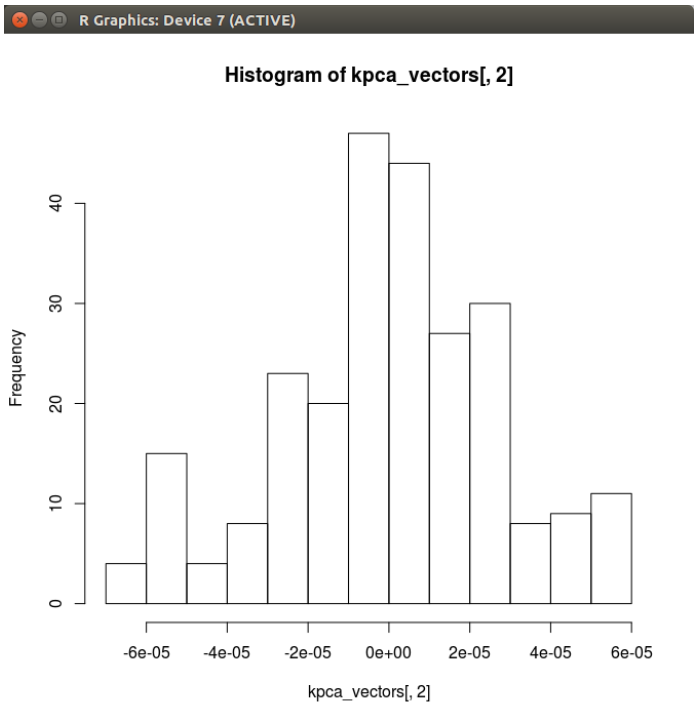
b. PCA

Upon applying the POLYDOT for the PCA for the given graph, we get the following metrics and the corresponding graphs and histograms

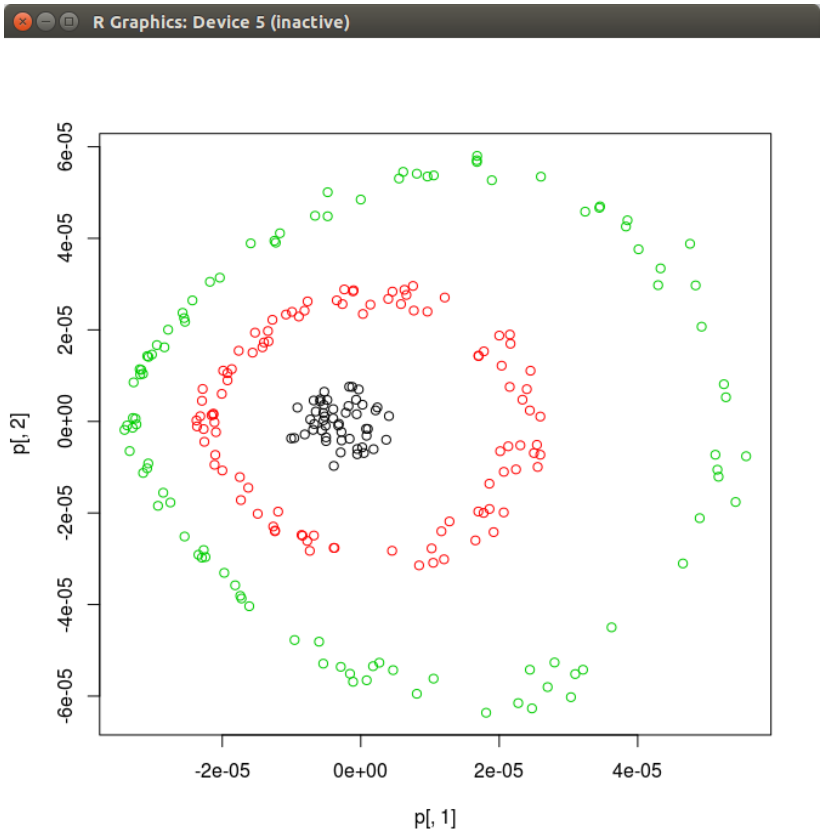
	Comp 1	Comp 2
Variance	0.6217029	0.3782971

The histograms of the first and second components using POLYDOT





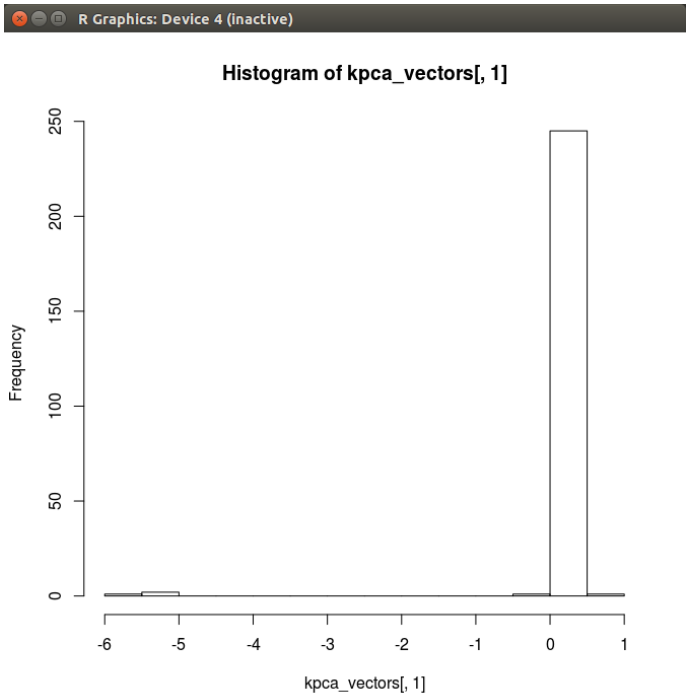
Reconstruction of the data using POLYDOT

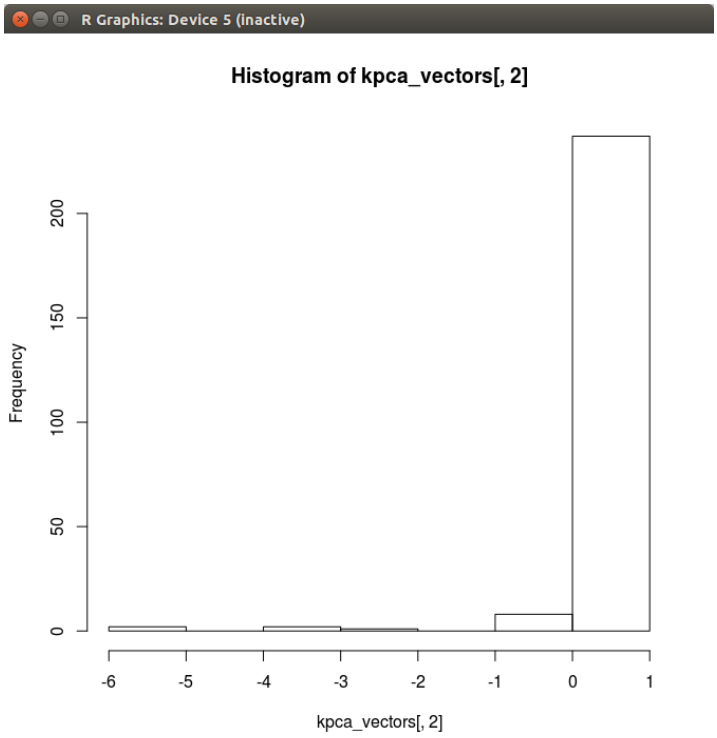


Upon applying the RBFDOT for the PCA for the given graph, we get the following metrics and the corresponding graphs and histograms

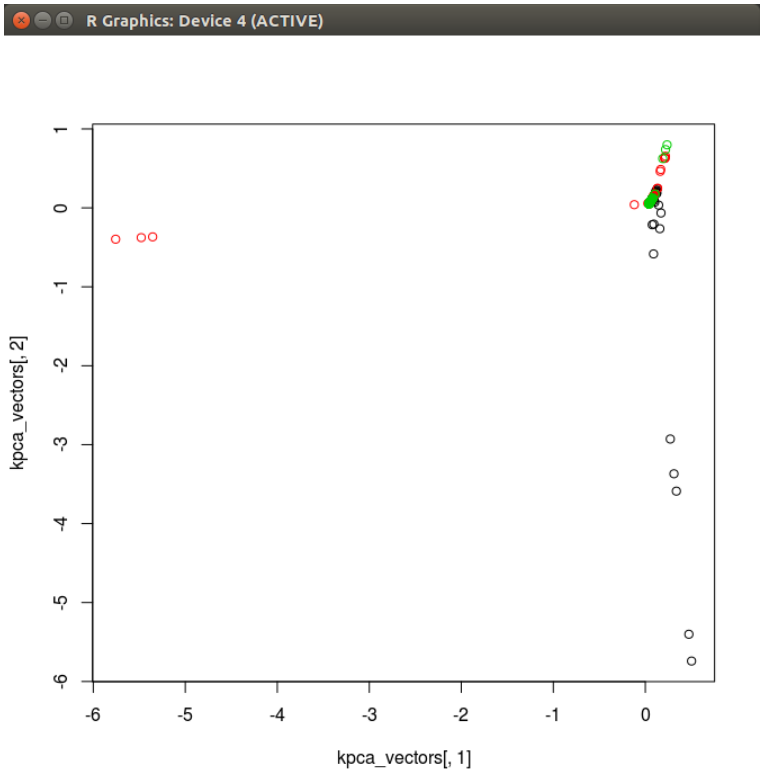
	Comp 1	Comp 2
Variance	0.48223	0.51777

Histograms of the first and second components using RBFDOT





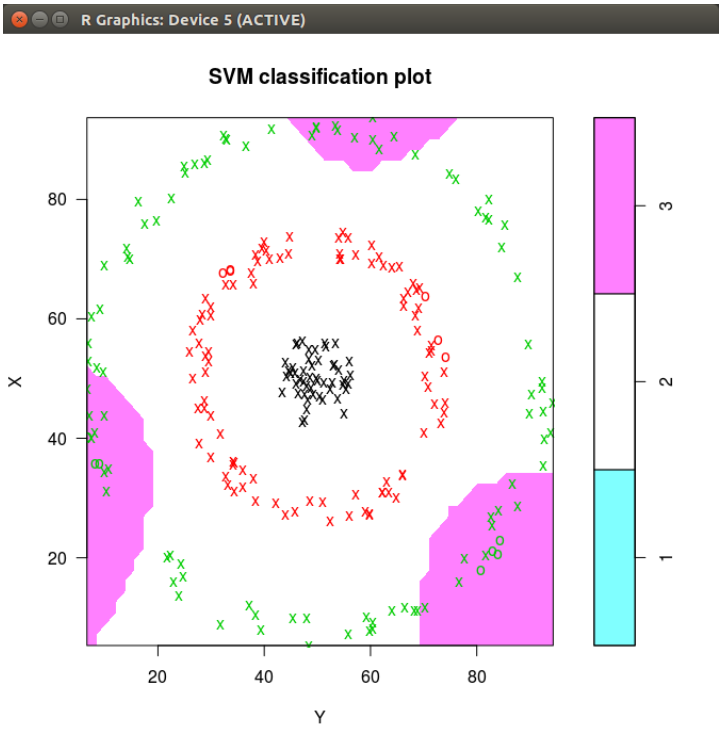
Reconstruction of the data using RBFDOT



c. SVM (using library e1071)

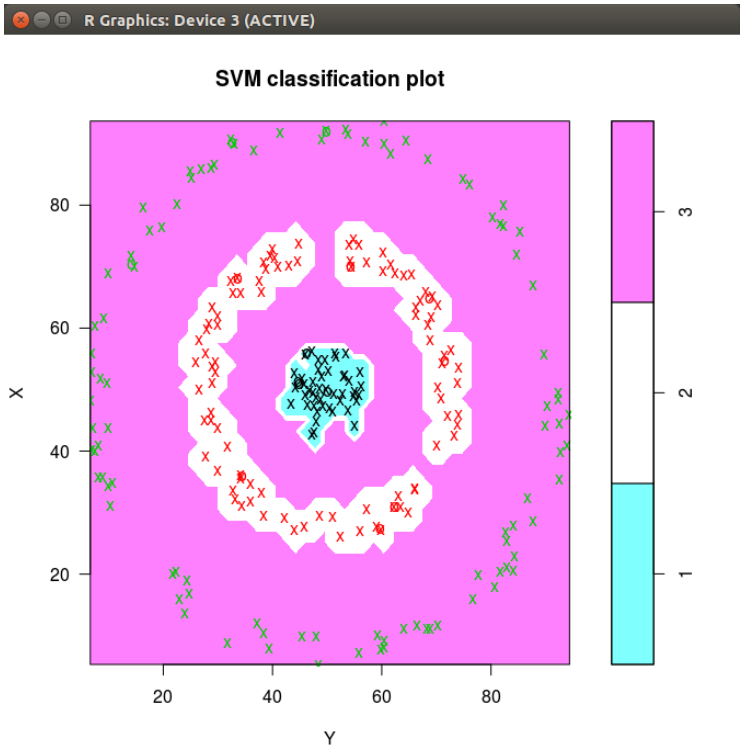
Upon applying the POLYNOMIAL kernel for the Kernel SVM, we get the below metrics and the corresponding graph

Polynomial Kernel SVM
Accuracy: 0.465
Precision: 0.38
Recall: 0.788
F-Score: 0.513



Upon applying the RADIAL(analogous to RBFDOT) kernel for the Kernel SVM, we get the below metrics and the corresponding graph

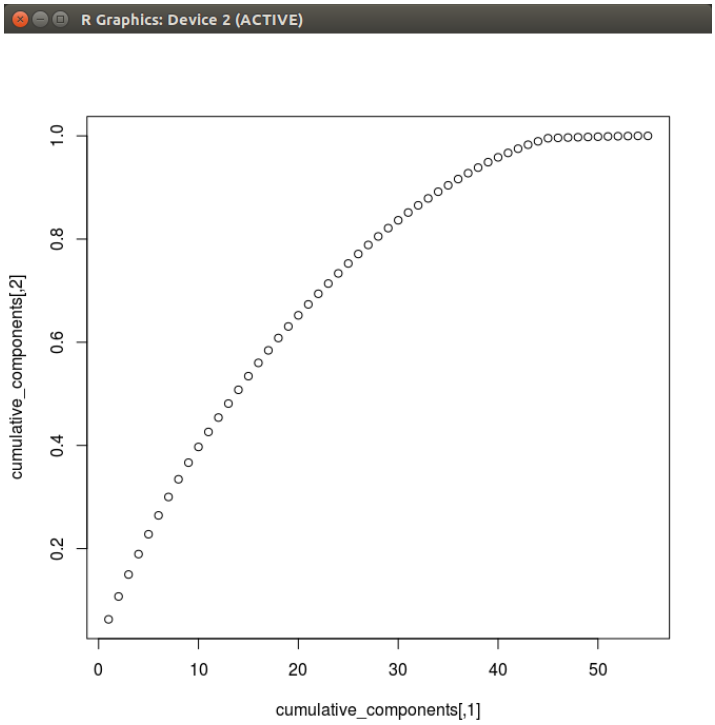
RBFDOT Kernel SVM
Accuracy: 1.0
Precision: 1.0
Recall: 1.0
F-Score: 1.0



4. High Dimensionality
   a. Script in the file name problem_4.R
   b. Upon running K-means on the high dimensional data, we get the following performance metrics (performance calculated using *performance.py*)

      Accuracy: 0.558
      Precision: 0.54
      Recall: 0.773
      F-Score: 0.636

   c. The plot of the eigen components vs the cumulative sum of the variance is shown below. There is a point at which the values almost saturate. Considering only those components, the variability will be preserved upto 0.9954169 (99.54%)



d. The projection snippet is in the source code file (problem_4.R)

e. Upon running KMeans on this new data, we get the following metrics which is much better than in the previous case (refer performance.py)

   Accuracy: 0.975
   Precision: 0.975
   Recall: 0.976
   F-Score: 0.975

f. The performance of KMeans after running PCA on the data has improved. The accuracy, precision, recall and f-score are all above 97%

g. Upon running KKMeans on the original dataset, we see that the performance is comparable with what was achieved through PCA. Below are the metrics for polynomial and rbf kernels. The code snippet can be found in problem_4.R

   Kernel KMeans Poly
   Accuracy: 0.966
   Precision: 0.965
   Recall: 0.966
   F-Score: 0.965

   Kernel KMeans RBF
   Accuracy: 0.97
   Precision: 0.97
   Recall: 0.971
   F-Score: 0.97

Kernel KMeans uses some Kernel Function internally, which may or may not be dimension reduction, and then runs normal KMeans on the Kernel Data. Using PCA to reduce the dimension first and then using KMeans will reduce the complexity of the algorithm since the dimensions are reduced.