<u>Exercise 1</u>: ***Generating the data sets***. Write a script (in R, Matlab, or SAS) that generates three data sets in a 2-dimensional space, defined as follows (see examples in http://cran.rproject.org/web/packages/kernlab/vignettes/kernlab.pdf):

(a) **BAD_kmeans**: The data set for which the kmeans clustering algorithm will not perform well.

```
GetKMeansData = function(radius,numberOfPoints,x,y){
coordinates <- matrix(nrow=numberOfPoints,ncol = 2)
for (i in 1:numberOfPoints) {
angle <- runif(1)*2*pi
coordinates[i,] <- c(x-runif(1)+sin(angle)*radius, y-runif(1)+cos(angle)*radius)
}
return(coordinates)
}
```

(b) **BAD_pca**: The data set for which the Principal Component Analysis (PCA) dimension reduction method upon projection of the original points into 1-dimensional space (i.e., the first eigenvector) will not perform well.

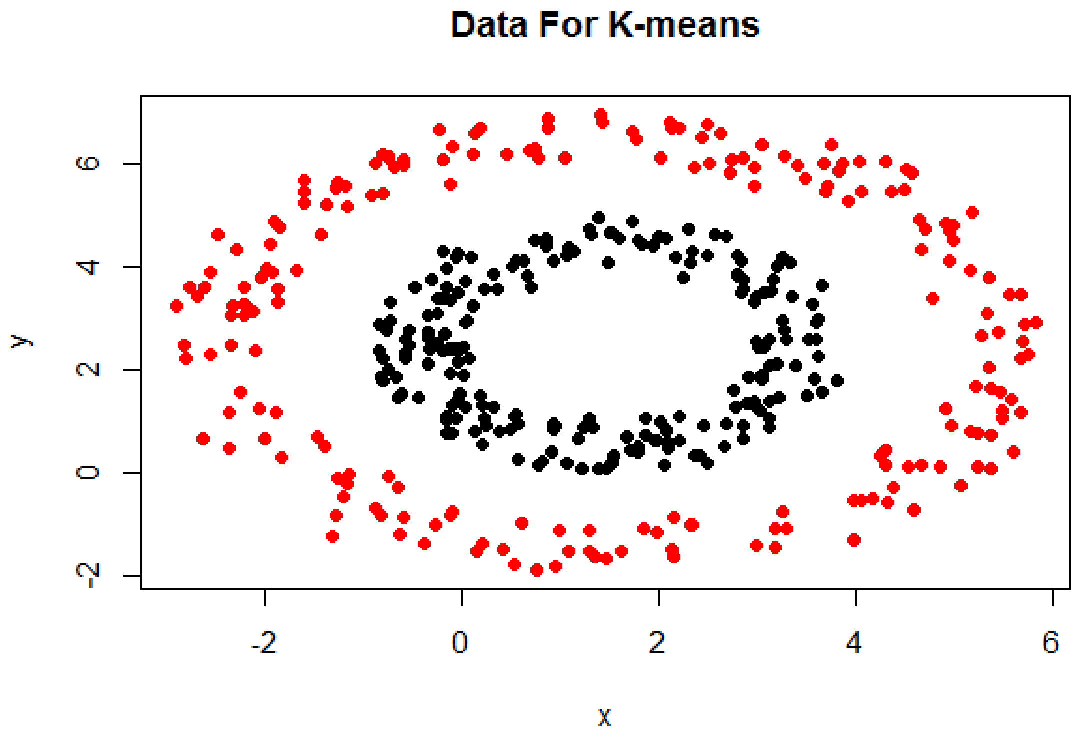
```
getPCAPoints = function(radius,numberOfPoints,x,y){
coordinates <- matrix(nrow=numberOfPoints,ncol = 2)
for (i in 1:numberOfPoints) {
angle <- runif(1)*2*pi
coordinates[i,] <- c(x+sin(angle)*radius, y+cos(angle)*radius)
}
return(coordinates)
}
```

(b) **BAD_svm**: The data set for which the *linear* Support Vector Machine (SVM) supervised classification method using two classes of points (positive and negative)will not perform well.

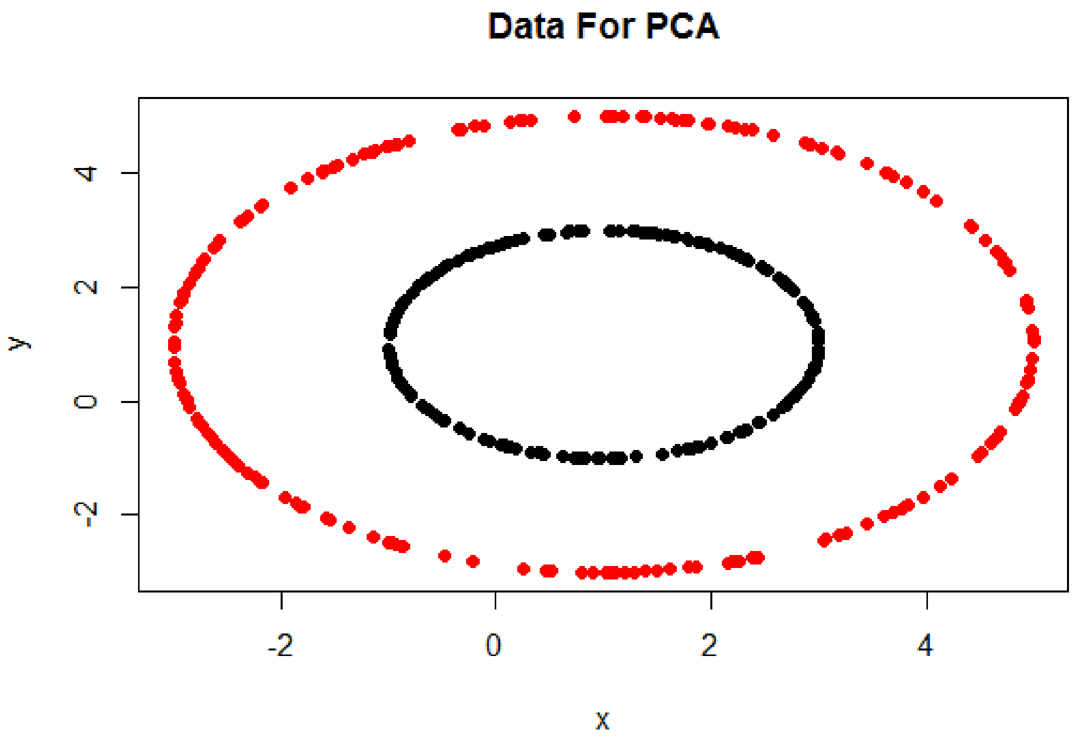
```
getSVMPoints = function(radius,numberOfPoints,x,y){
coordinates <- matrix(nrow=numberOfPoints,ncol = 2)
for (i in 1:numberOfPoints) {
angle <- runif(1)*pi
coordinates[i,] <- c(x+sin(angle)*radius, y+cos(angle)*radius)
}
return(coordinates)
}
```

(c) Plot each data set in a 2-dimensional space.

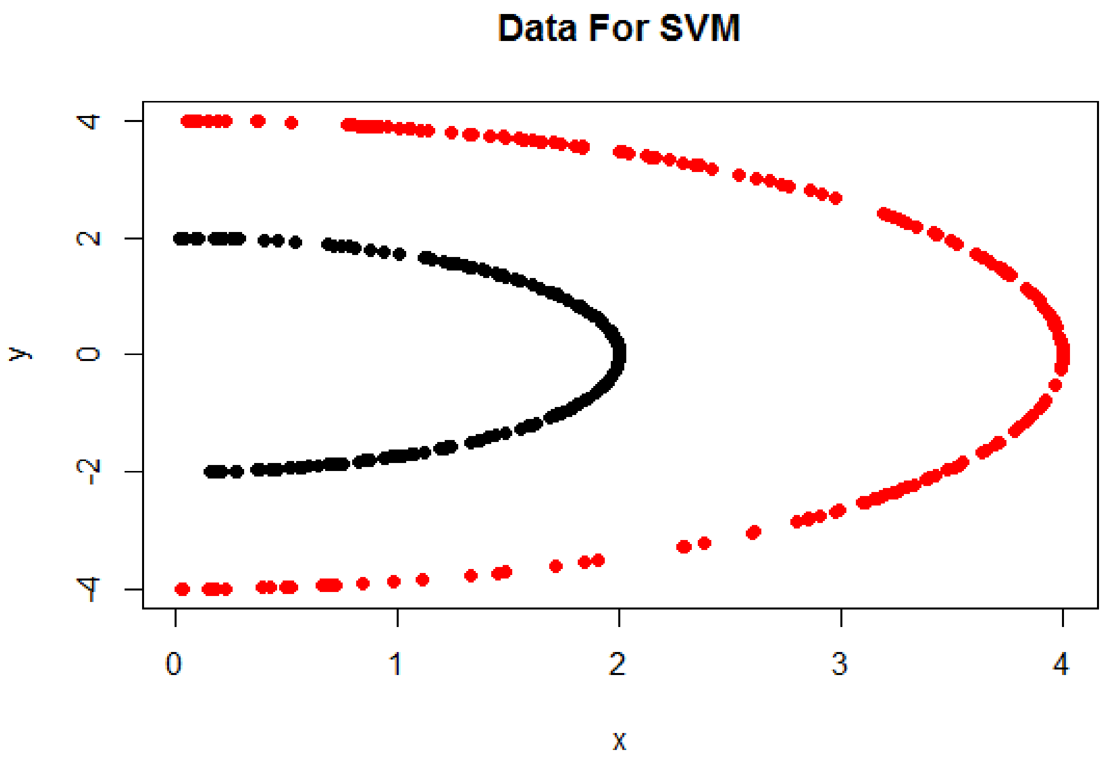
```
KmeansCluster1  = GetKMeansData(2,200,2,3)
KmeansClusterlabel1  =matrix(1,nrow=200,ncol=1)
KmeansCluster1  = cbind(KmeansCluster1,KmeansClusterlabel1)
KmeansCluster2  = GetKMeansData(4,200,2,3)
KmeansClusterlabel2  =matrix(2,nrow=200,ncol=1)
KmeansCluster2  = cbind(KmeansCluster2,KmeansClusterlabel2)
kmeansPoints  = rbind(KmeansCluster1,KmeansCluster2)
plot(kmeansPoints, xlab="x",ylab="y", main="Data For K-
means",col=kmeansPoints[,3],pch=16)
```



```
PCACluster1 = getPCAPoints(2,200,1,1)
PCAClusterlabel1 =matrix(1,nrow=200,ncol=1)
PCACluster1 = cbind(PCACluster1,PCAClusterlabel1)
PCACluster2 = getPCAPoints(4,200,1,1)
PCAClusterlabel2 =matrix(2,nrow=200,ncol=1)
PCACluster2 = cbind(PCACluster2,PCAClusterlabel2)
PCAPoints = rbind(PCACluster1,PCACluster2)
plot(PCAPoints, xlab="x",ylab="y", main="Data For
PCA",col=PCAPoints[,3],pch=19)
```



```
SVMCluster1 = getSVMPoints(2,200,0,0)
SVMClusterlabel1 =matrix(1,nrow=200,ncol=1)
SVMCluster1 = cbind(SVMCluster1,SVMClusterlabel1)
SVMCluster2 = getSVMPoints(4,200,0,0)
SVMClusterlabel2 =matrix(2,nrow=200,ncol=1)
SVMCluster2 = cbind(SVMCluster2,SVMClusterlabel2)
SVMPoints = rbind(SVMCluster1,SVMCluster2)
plot(SVMPoints, xlab="x",ylab="y", main="Data For
SVM",col=SVMPoints[,3],pch=19)
```



<u>Exercise 2</u>: ***Evaluating the "badness" of the data mining methods***. Write a script that uses the BAD data set in Exercise 2, runs the corresponding data mining method, produces the output from the method, and evaluates how bad the performance of this method is. You may use various performance metrics to assess each method (e.g., the variance, precision, recall, F1 measure). Not all the metrics could equally apply to each of the technique. Reading the Performance Metrics chapter by Kanchana and John from the Practical Graph Mining with R book is strongly encouraged for performing this exercise. Also, the book web-site provides the R scripts to play with these metrics, if interested. Report the summary of the performance metrics used and the performance results obtained.

```
getPerformance <- function(groundLabel, classLablel) {
TP=0.0
FN=0.0
FP=0.0
TN=0.0
groundLabelMat = as.matrix(groundLabel)
classLablelMat = as.matrix(classLablel)
for (i in 1:nrow(groundLabelMat)) {
        for (j in 1:nrow(classLablelMat)) {
        if(i == j) {
##do nothin
} else if((groundLabelMat[i]==groundLabelMat[j]) &&
          (classLablelMat[i]==classLablelMat[j])){
                    TP = TP +1
} else if((groundLabelMat[i]!=groundLabelMat[j]) &&
(classLablelMat[i]!=classLablelMat[j])) {
                     TN = TN +1
} else if((groundLabelMat[i]==groundLabelMat[j]) &&
(classLablelMat[i]!=classLablelMat[j])) {
                           FN =FN +1
} else if((groundLabelMat[i]!=groundLabelMat[j]) &&
(classLablelMat[i]==classLablelMat[j])) {
                           FP =FP +1
                     }
                  }}
  TP = TP/2
  TN = TN/2
  FN = FN/2
  FP = FP/2
  accuracy = (TP + TN) / (TP + TN + FP + FN)
  precision = TP/(TP + FP)
  recall = TP/(TP + FN)
  fscore = 2*precision*recall/(precision + recall)
  return(list(accuracy=accuracy,precision=precision,recall=recall,fscore=fscore))
}
```

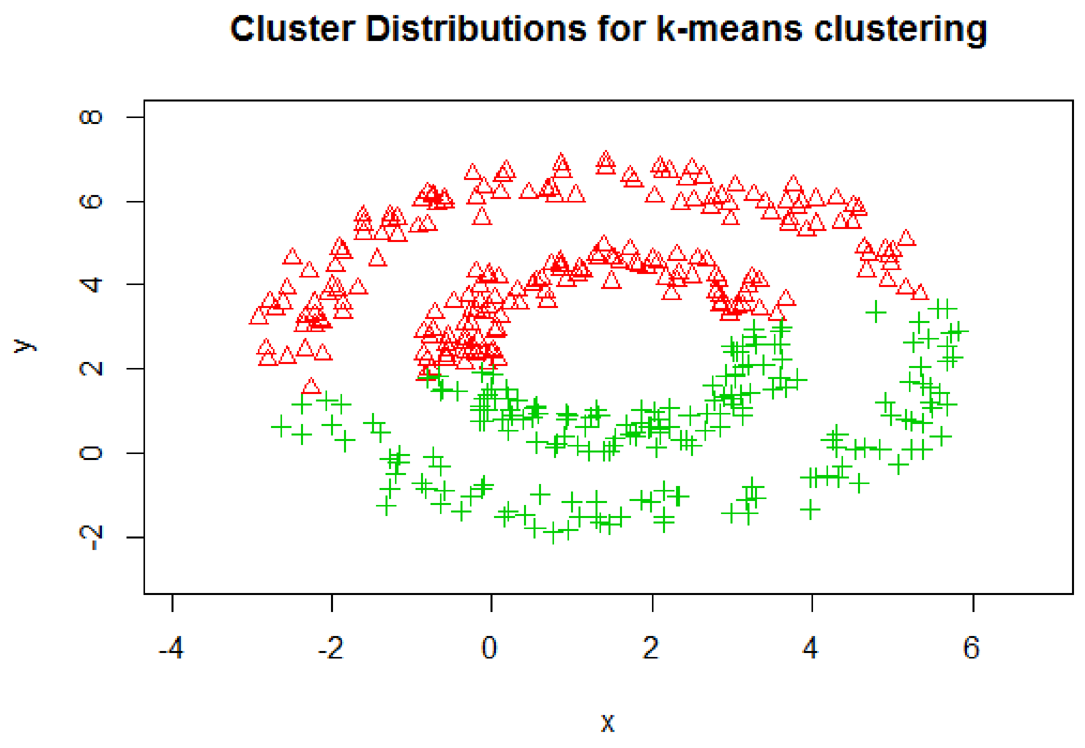
K-MEANS

```
kmeansOut <- kmeans(kmeansPoints,2)
kemansCluster1 = kmeansPoints[which(kmeansOut$cluster==1),]
kemansCluster2 = kmeansPoints[which(kmeansOut$cluster==2),]

x1 = min(kmeansPoints[,1]) - 1
x2 = max(kmeansPoints[,1]) + 1
y1 = min(kmeansPoints[,2]) - 1
y2 = max(kmeansPoints[,2]) + 1

plot(-100,-100, col=c(1), main = "Cluster Distributions for k-means clustering",
xlim=c(x1,x2), ylim=c(y1,y2), pch=c(19))
points(kemansCluster1, col=2, pch=2)
points(kemansCluster2, col=3, pch=3)
kemansperf = getPerformance(kmeansPoints[,3],kmeansOut$cluster)

$accuracy
[1] 0.4997619
$precision
[1] 0.4985069
$recall
[1] 0.4991206
$fscore
[1] 0.4988135
```

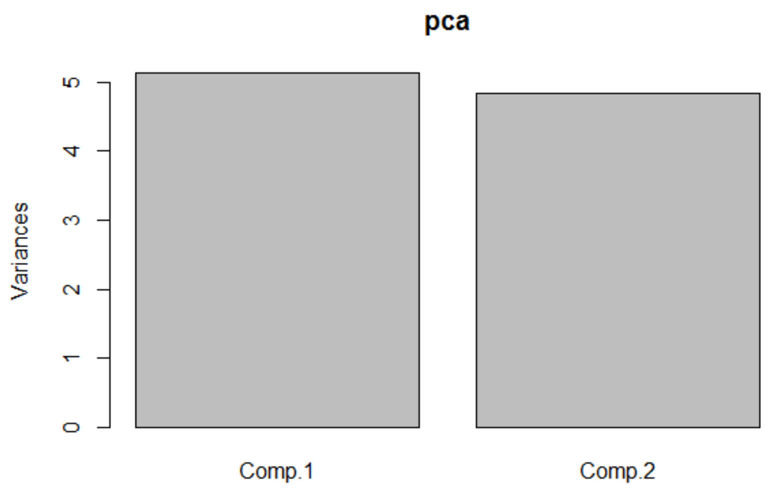


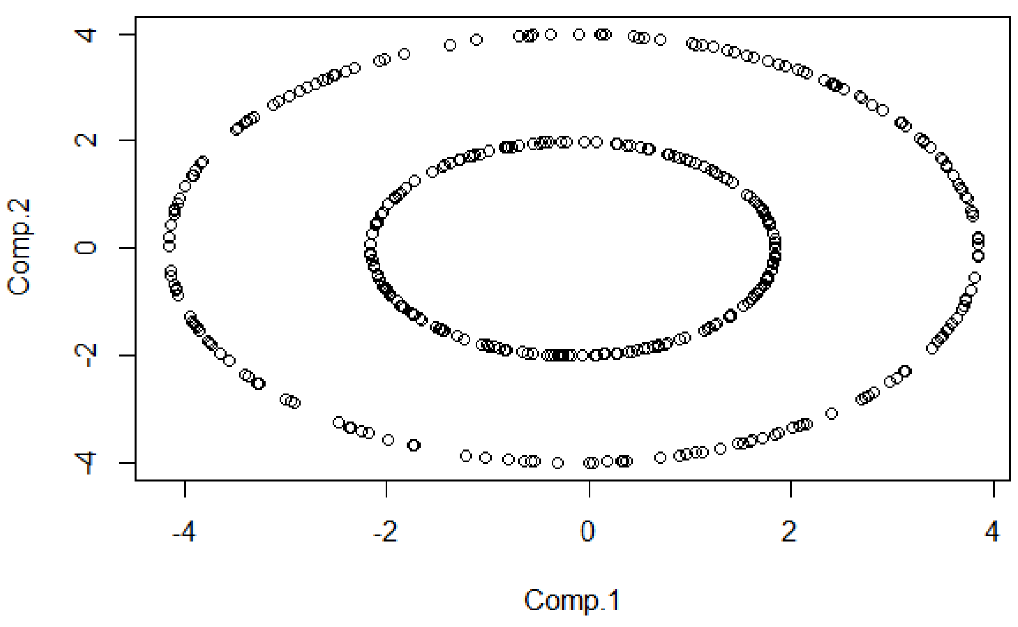
PCA

```
pca = princomp (PCAPoints[,-3], center=TRUE);
pca
plot (pca); # screeplot
pca_loadings=loadings(pca);      # matrix of eigenvectors
pca_summary=summary  (pca); # check proportion of variance
P=pca$scores;     # projection of X onto eigenvectors
proj_data_c1 <- P[, 1];
proj_data_c2 <- P[, 2];
plot(P)
plot (proj_data_c1, xlab="Point Index",ylab="Projection  on Component 1");
plot (proj_data_c2, xlab="Point Index",ylab="Projection  on Component 2");
barplot(proj_data_c1,main="BarPlot  of projection on Component1")
barplot(proj_data_c2,main="BarPlot  of projection on Component2")
```

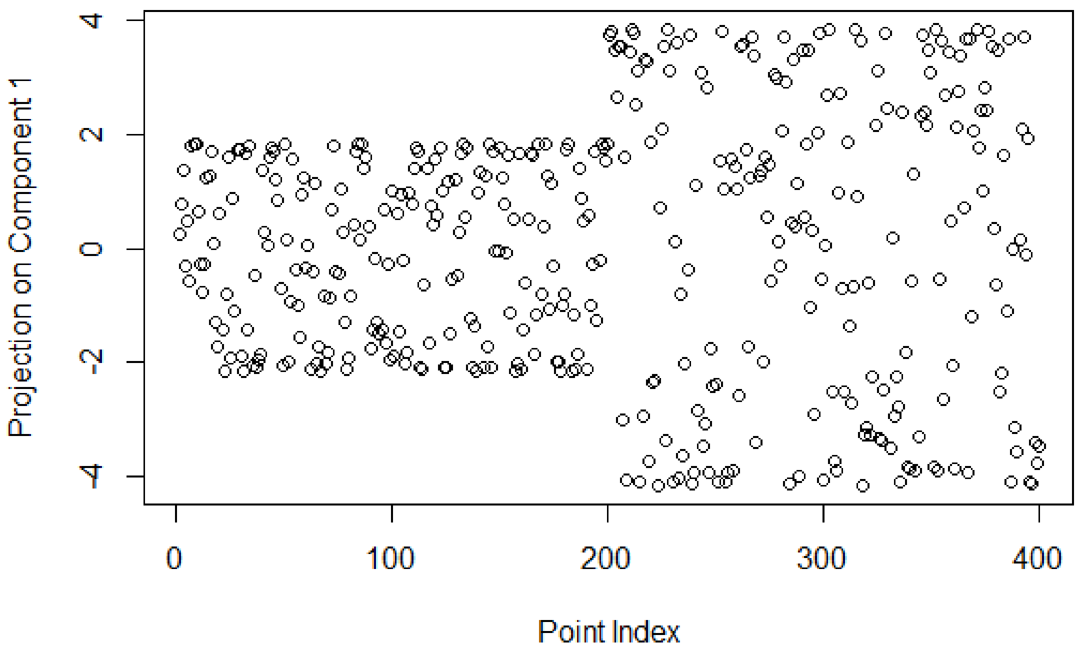
PCA summary	Comp.1	Comp.2
Standard deviation	2.2651	2.201
Proportion of Variance	0.5143	0.4857
Cumulative Proportion	0.5143	1

Since both the components/Eigen vectors have variance which cannot be ignored and equally distributed across them, data is evenly spread across new two components and dimension cannot be reduced. Hence PCA is performing badly here. Also projection of points on two components is not easy to cluster further as seen from the plot.

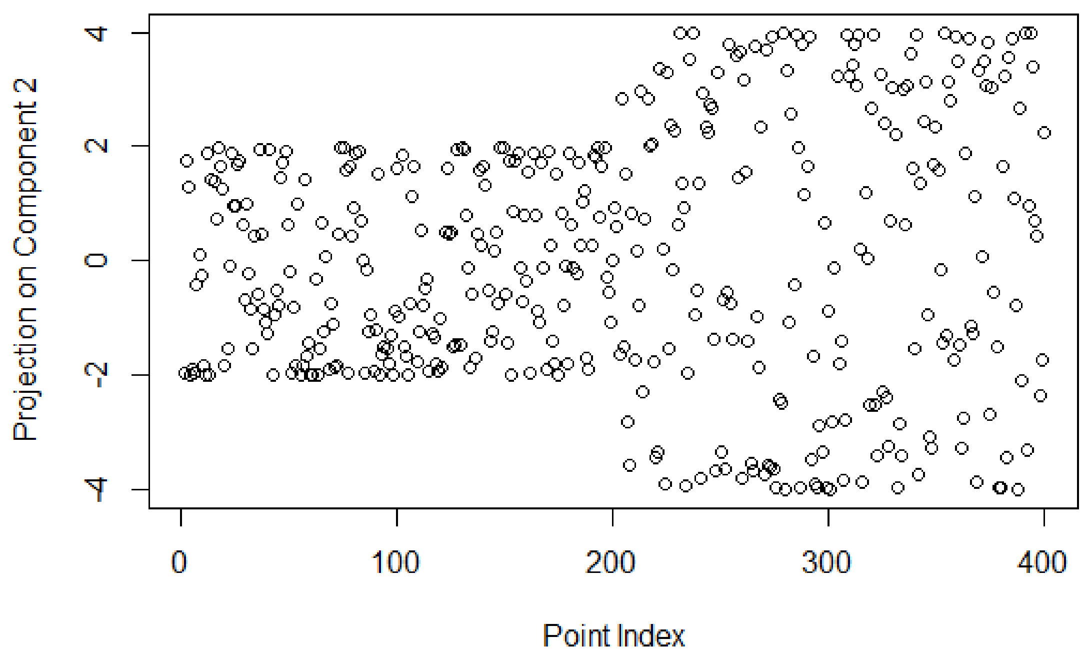




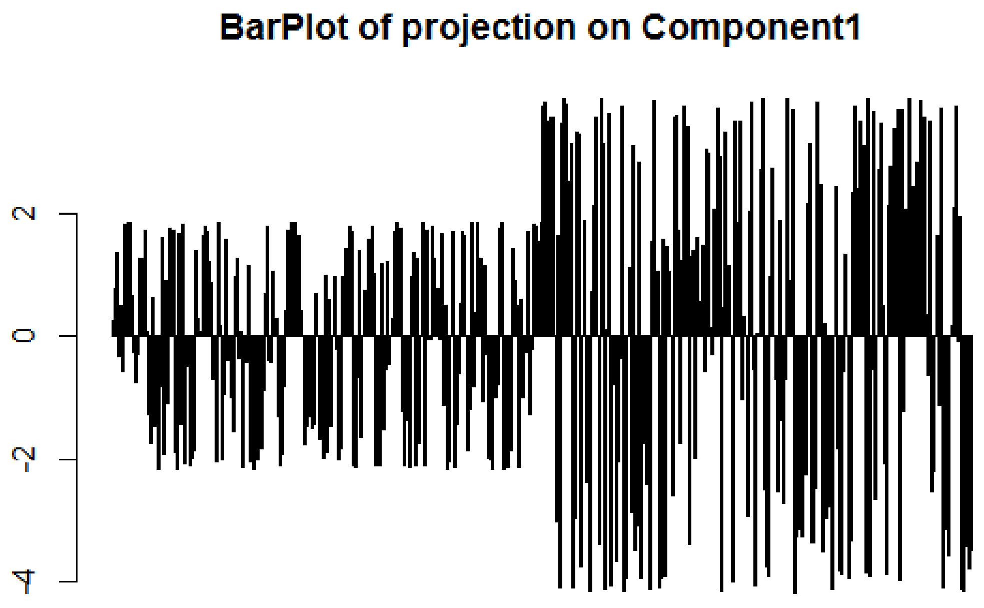
Comp.2
Comp.1
-4
-2
0
2
4



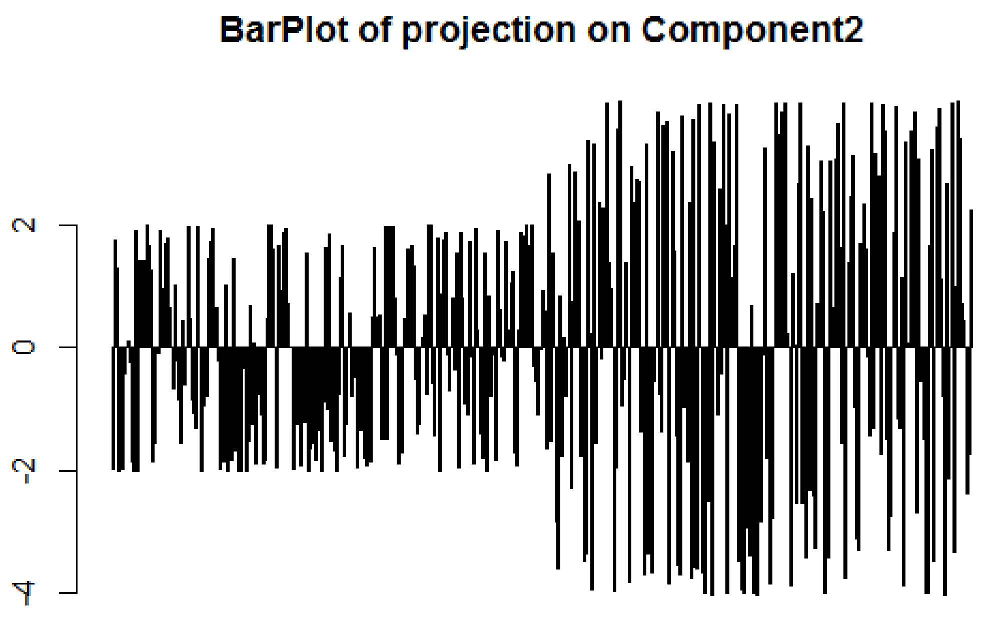
Projection on Component 1
Point Index
0
100
200
300
400
-4
-2
0
2
4

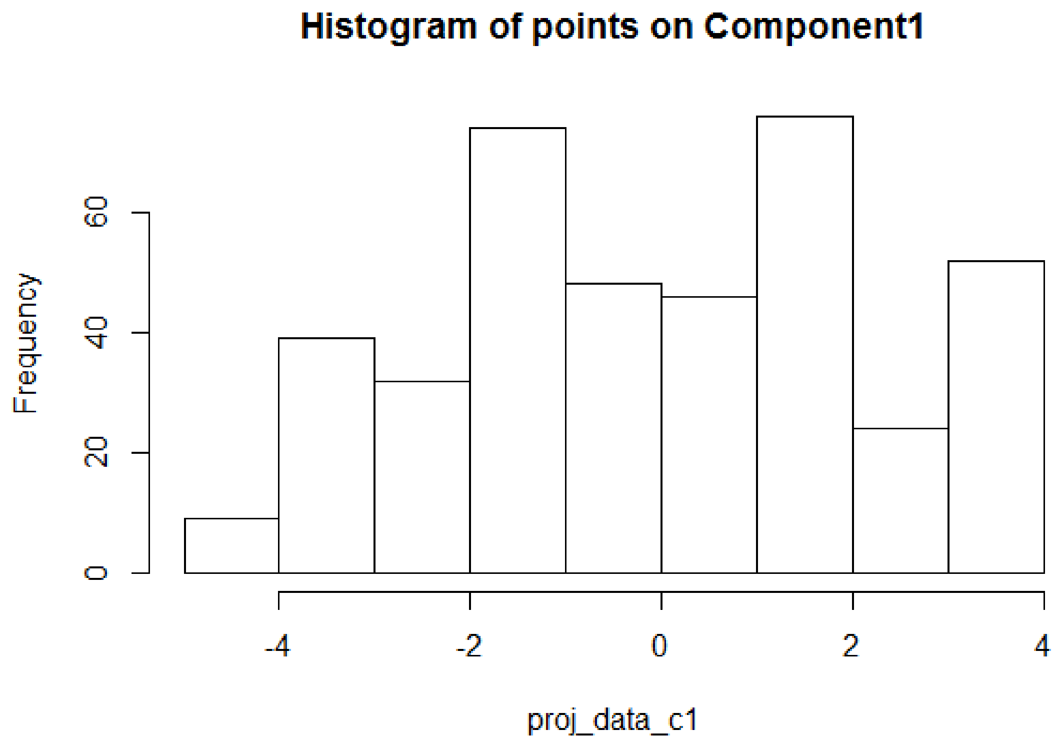


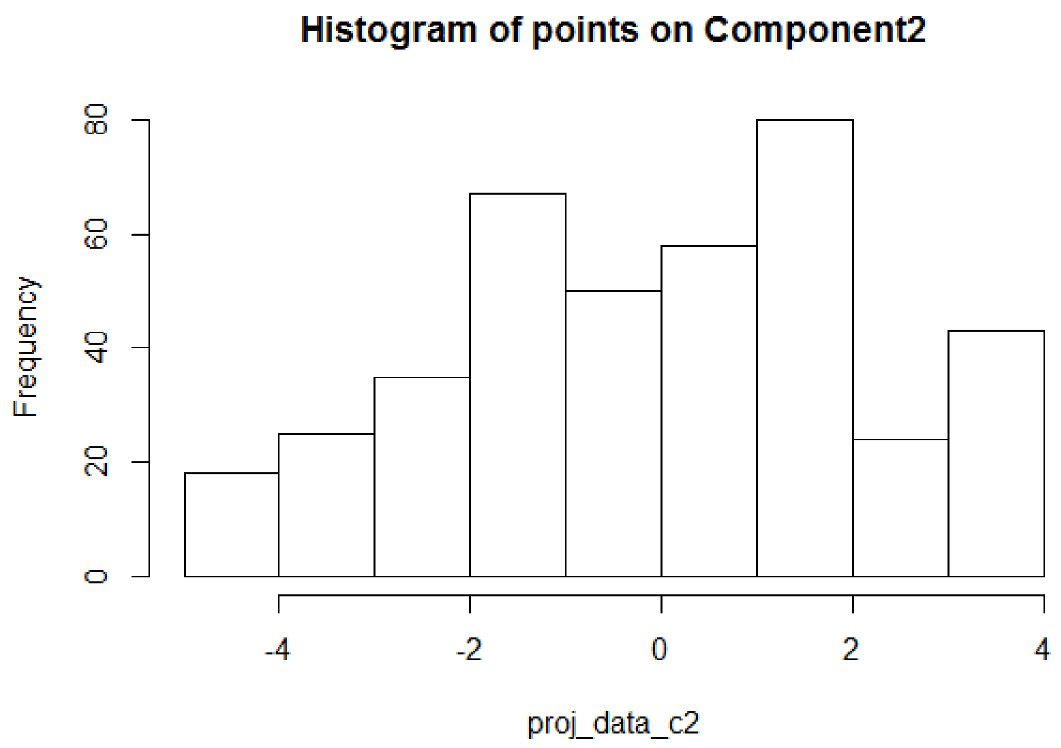
Projection on Component 2
Point Index



BarPlot of projection on Component1


BarPlot of projection on Component2
2
0
-2
-4

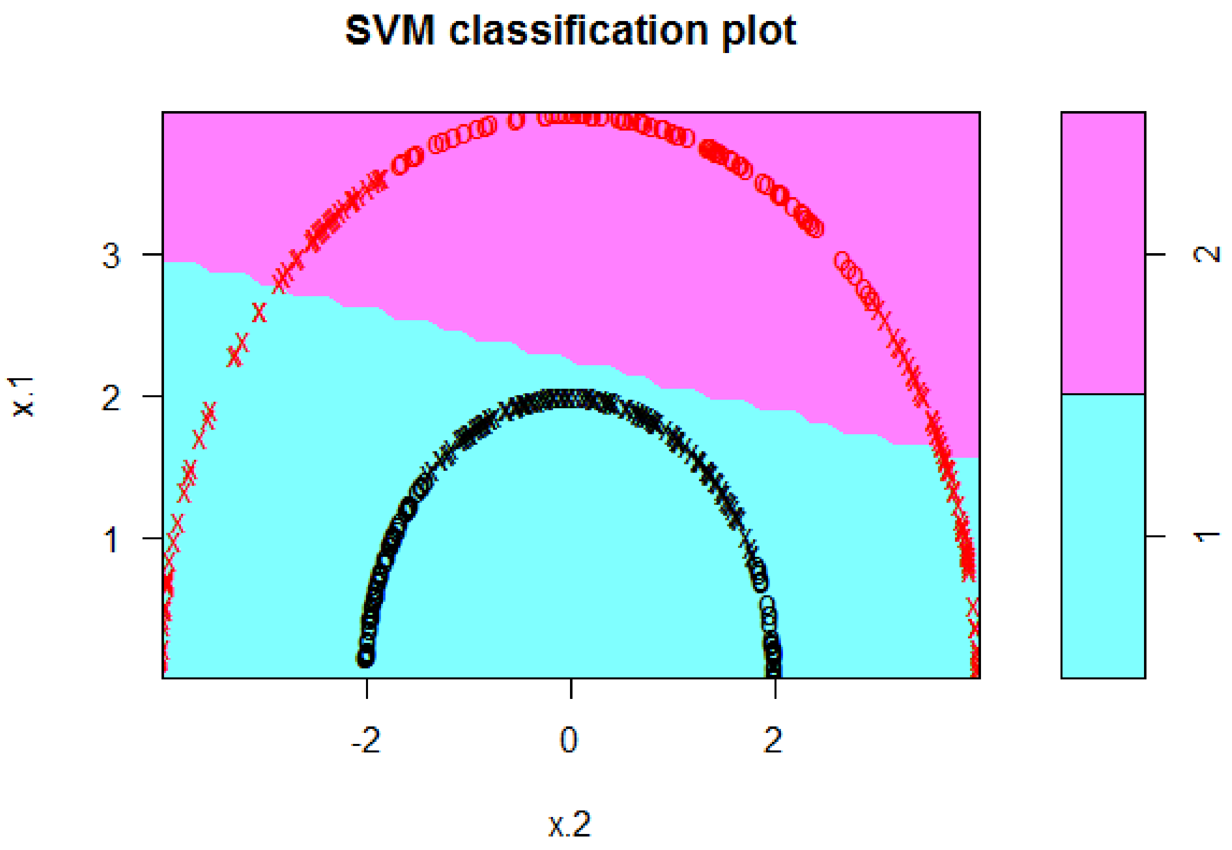

Histogram of points on Component1
Frequency
0
20
40
60
-4
-2
0
2
4
proj_data_c1

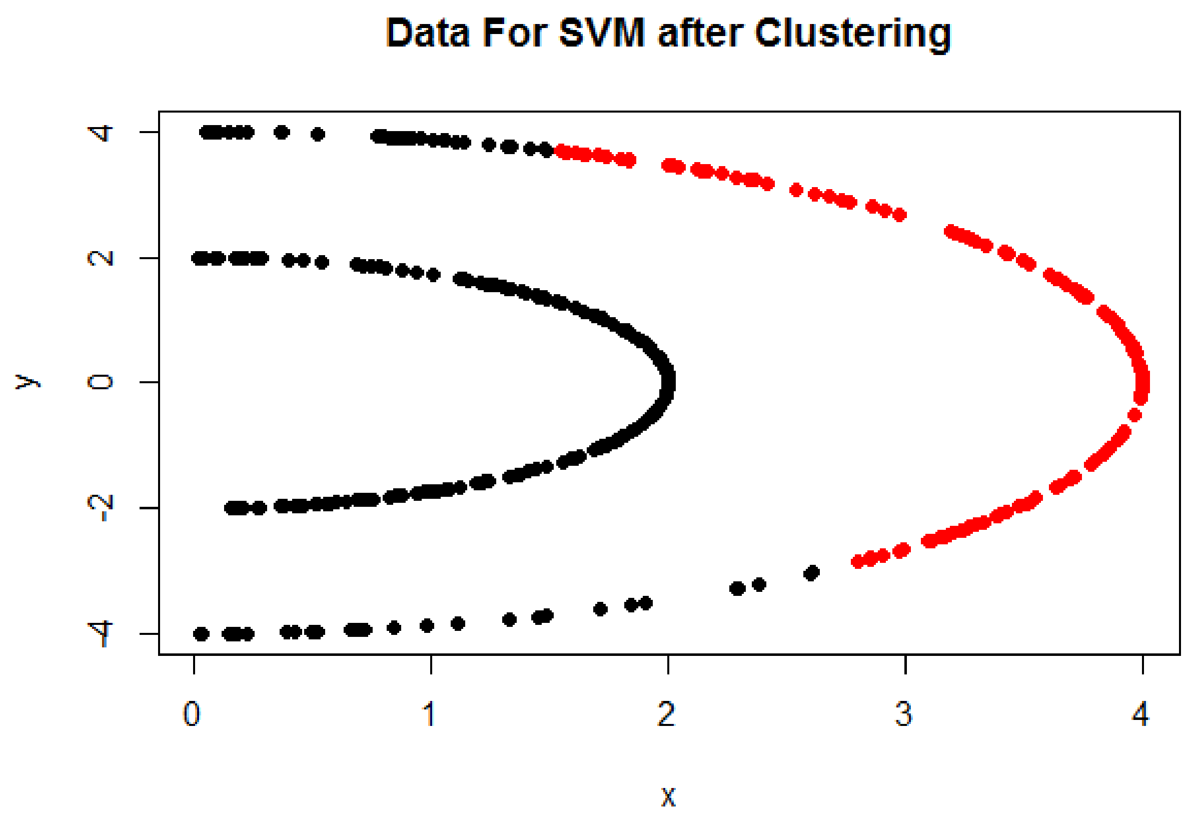


SVM

```
library("e1071")
dat=data.frame(x=SVMPoints[,-3], y=as.factor(SVMPoints[,3]))
svmfit=svm(y~., data=dat , kernel ="linear",
cost=10,scale=FALSE)
plot(svmfit,dat)
svmclusterLabel = as.matrix(svmfit$fitted)
storage.mode(svmclusterLabel) <- "integer"
svmperf = getPerformance(SVMPoints[,3],svmclusterLabel)
plot(SVMPoints[,-3], xlab="x",ylab="y", main="Data For SVM
after Clustering",col=svmclusterLabel,pch=19)
```

Performance of SVM	
accuracy	0.7305
precision	0.7084
recall	0.7813
fscore	0.7431





<u>Exercise 3</u>: ***Kernelizing the methods.*** Write a script that uses the *kernalized* version of each of the data mining method in Exercise 3 (e.g., you may consider using ***kernlab*** and ***kkmeans*** packages in R for kernel SVM+PCA and kmeans, resp.).

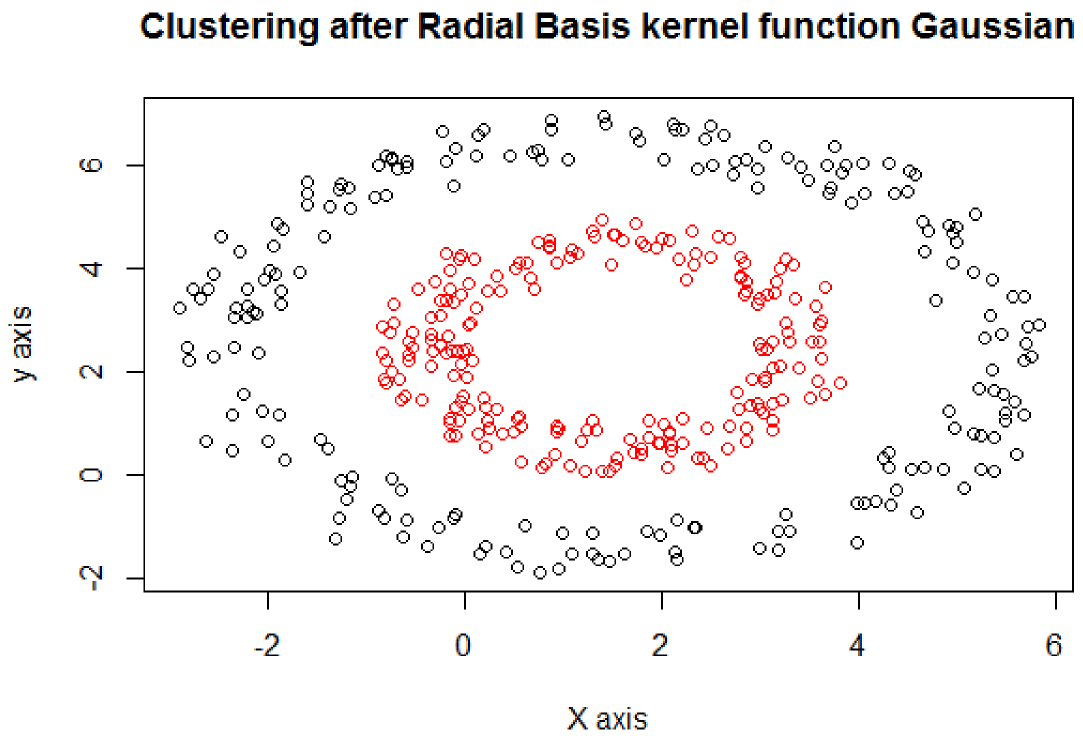
(a) Choose at least two kernels for each of the methods.

(b) Use the same performance metrics as in Ex. 3, and compare the performance obtained by the methods after applying the kernel trick versus the original un-kernelized versions of the techniques.

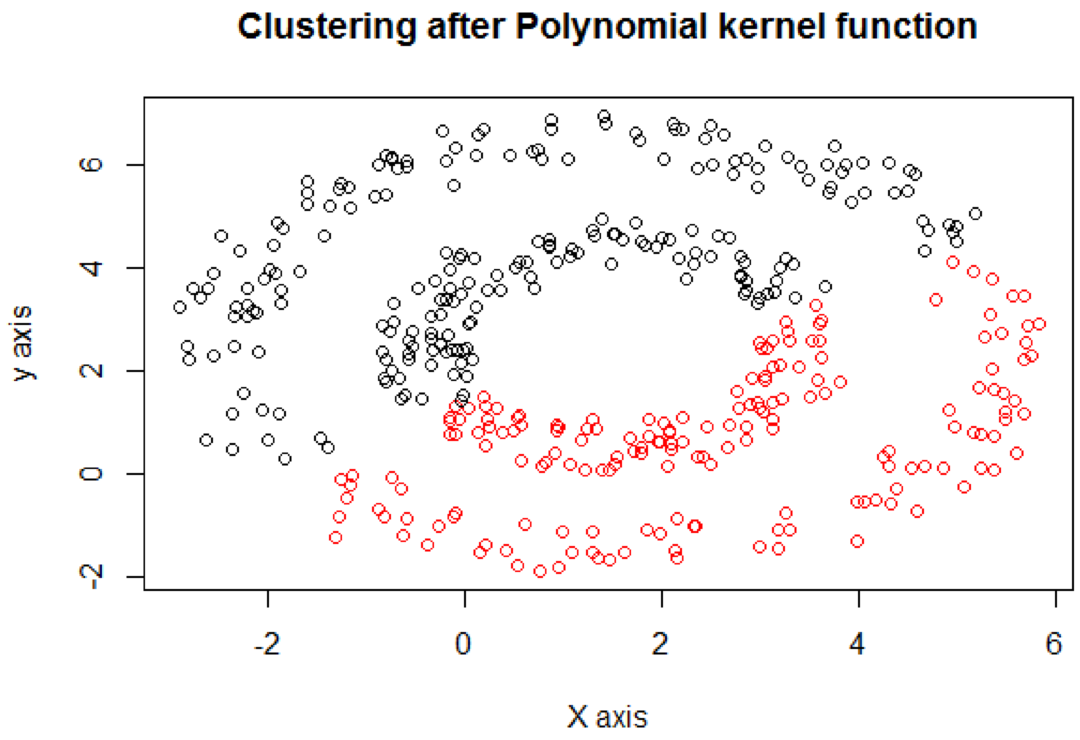
K-MEANS

```
dat=kmeansPoints[,-3]
#kkmeansRbf = specc(dat,2,kernel="rbfdot")
#kemansperf = getPerformance(kmeansPoints[,3],kkmeansLabel)
kkmeansRbf = kkmeans(dat,2,kernel="rbfdot")
kemansperf = getPerformance(kmeansPoints[,3],kkmeansLabel)
plot(dat, col=kkmeansRbf)
plot(dat, col=kkmeansRbf,xlab="X axis",ylab="y axis",main="Clustering after
Radial Basis kernel function Gaussian")
```

Performance of KMEANS	
accuracy	1
precision	1
recall	1
fscore	1



```
dat=kmeansPoints[,-3]
kkmeansPoly = kkmeans(dat,2,kernel="polydot")
kemansperf = getPerformance(kmeansPoints[,3],kkmeansLabel)
plot(dat, col=kkmeansPoly,xlab="X axis",ylab="y axis",main="Clustering after
Polynomial kernel function")
```

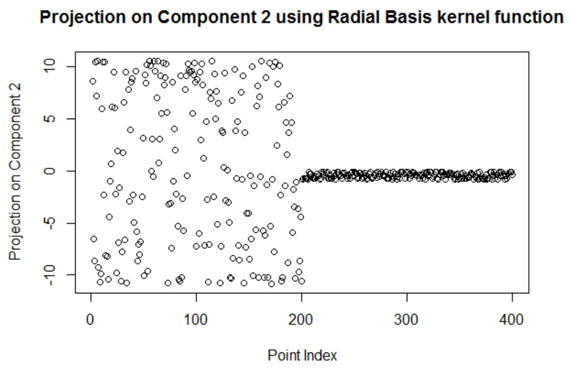


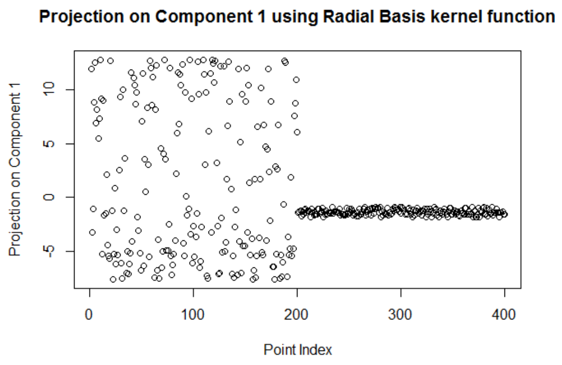
Performance of KMEANS	
accuracy	0.4993609
precision	0.4981238
recall	0.5036432
fscore	0.5008683

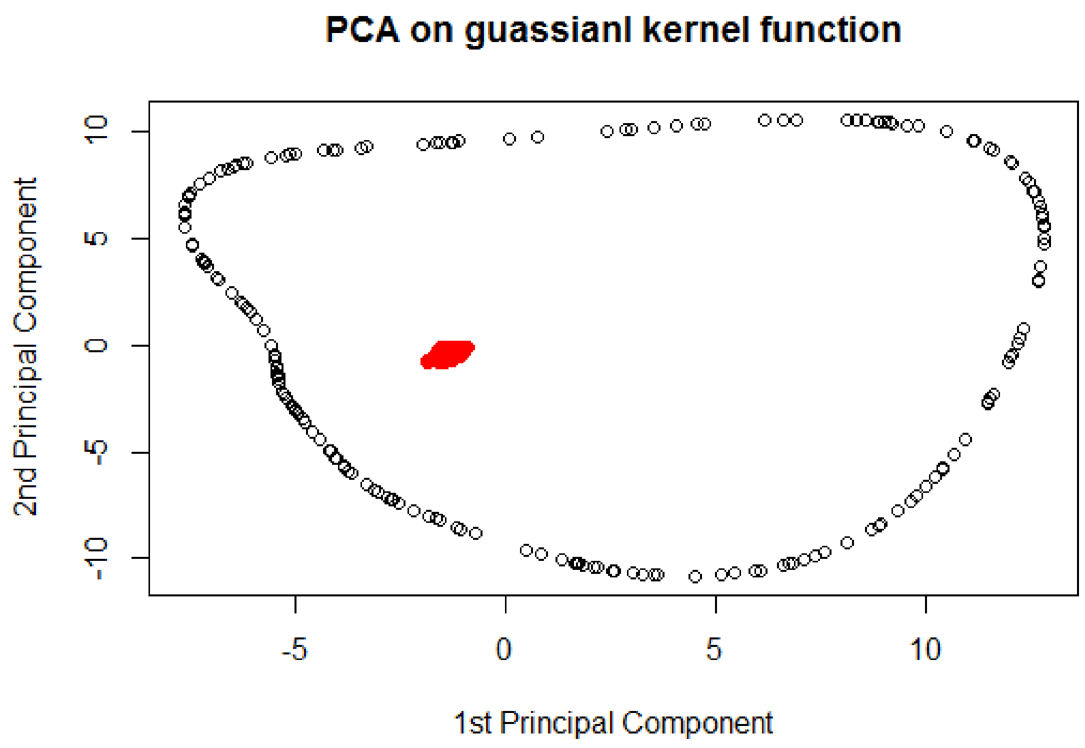
PCA

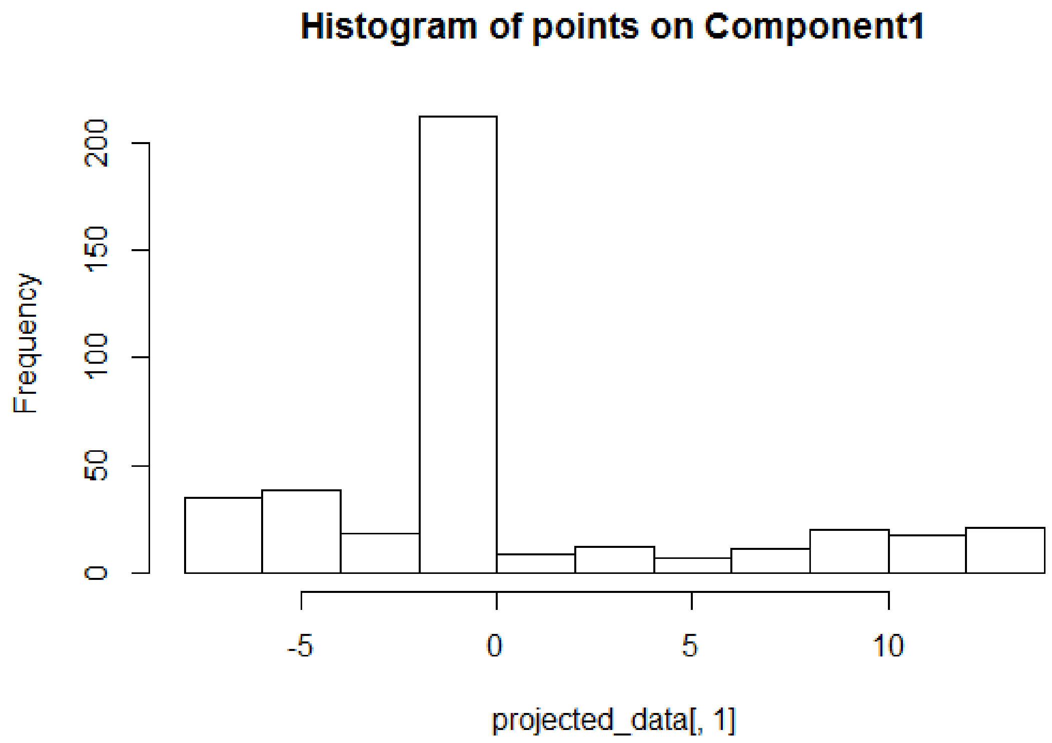
Kernel Trick:RBFDOT(Radial Basis kernel function)

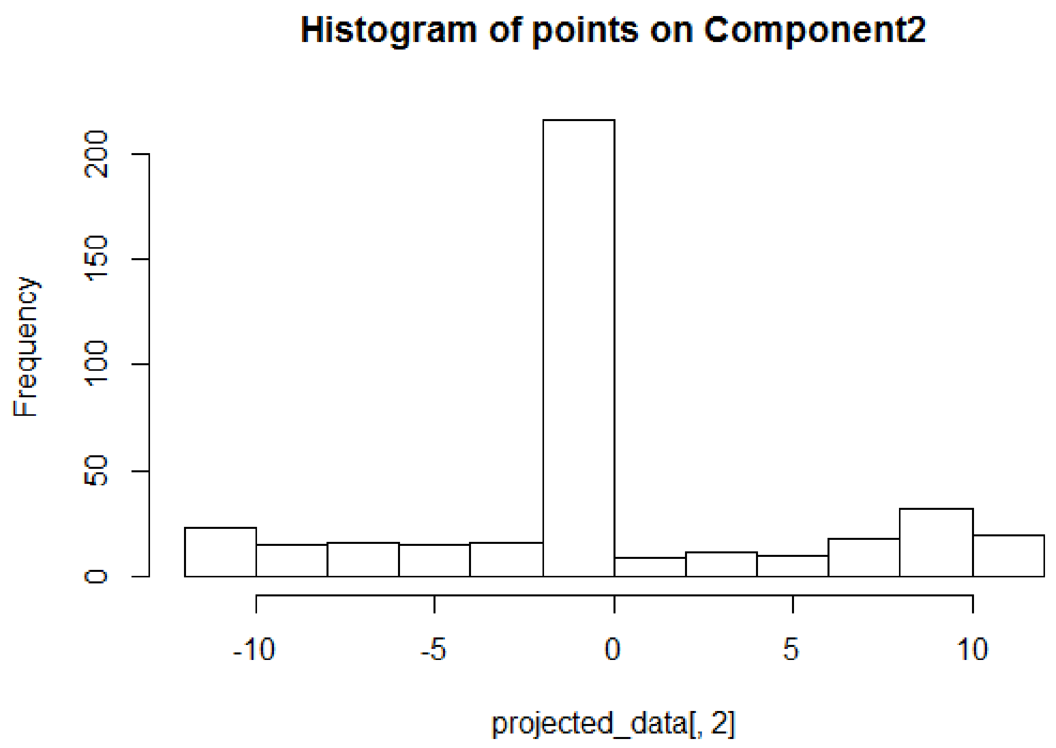
```
pca_kernel <- kpca(PCAPoints[,-3],
kernel="rbfdot",kpar=list(sigma=1),features=2)
pca_loadings_kernel <- pcv(pca_kernel);
pca_eigne_vales <- eig(pca_kernel)
projected_data <- rotated(pca_kernel);
plot (projected_data[,1], xlab="Point Index",ylab="Projection on
Component 1",main="Clustering after Radial Basis kernel function");
plot (projected_data[,2], xlab="Point Index",ylab="Projection on
Component 2",main="Clustering after Radial Basis kernel function");
pca_variance = pca_eigne_vales/sum(pca_eigne_vales)
plot(rotated(pca_kernel),col=as.integer(PCAPoints[,3]),xlab="1st Principal
Component",ylab="2nd Principal Component")
hist(projected_data[,1],main="Histogram of points on Component1")
hist(projected_data[,2],main="Histogram of points on Component2")
```






PCA on guassianl kernel function
2nd Principal Component
1st Principal Component
-10
-5
0
5
10


Histogram of points on Component1
Frequency
projected_data[, 1]
0
50
100
150
200
-5
0
5
10



We see that since points on each of the components are dominated by one set and clearly separated by rest of the results.

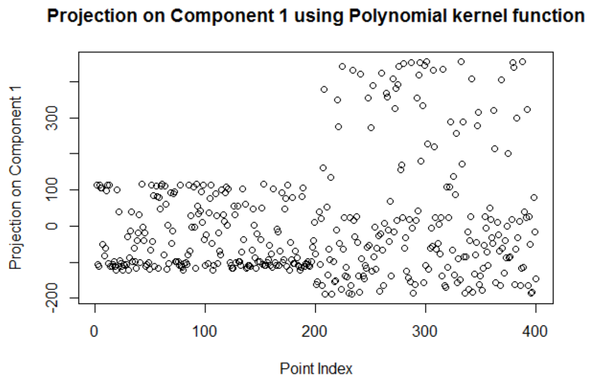
Proportion of Variance	0.636464	0.363536

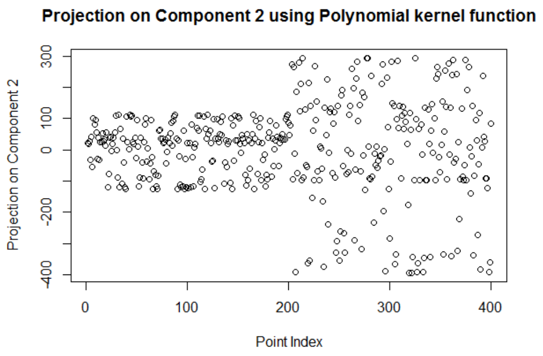
We see that variance is better as compared to the PCA in question2.

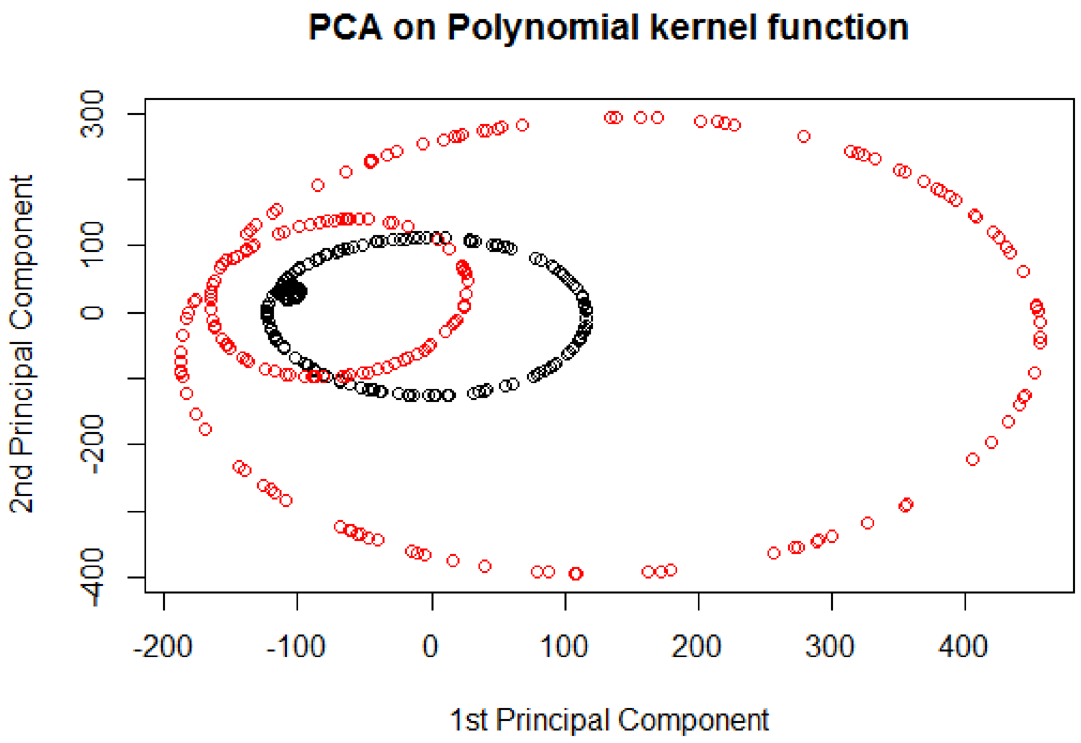
Kernel Trick: polydot (Polynomial kernel function)

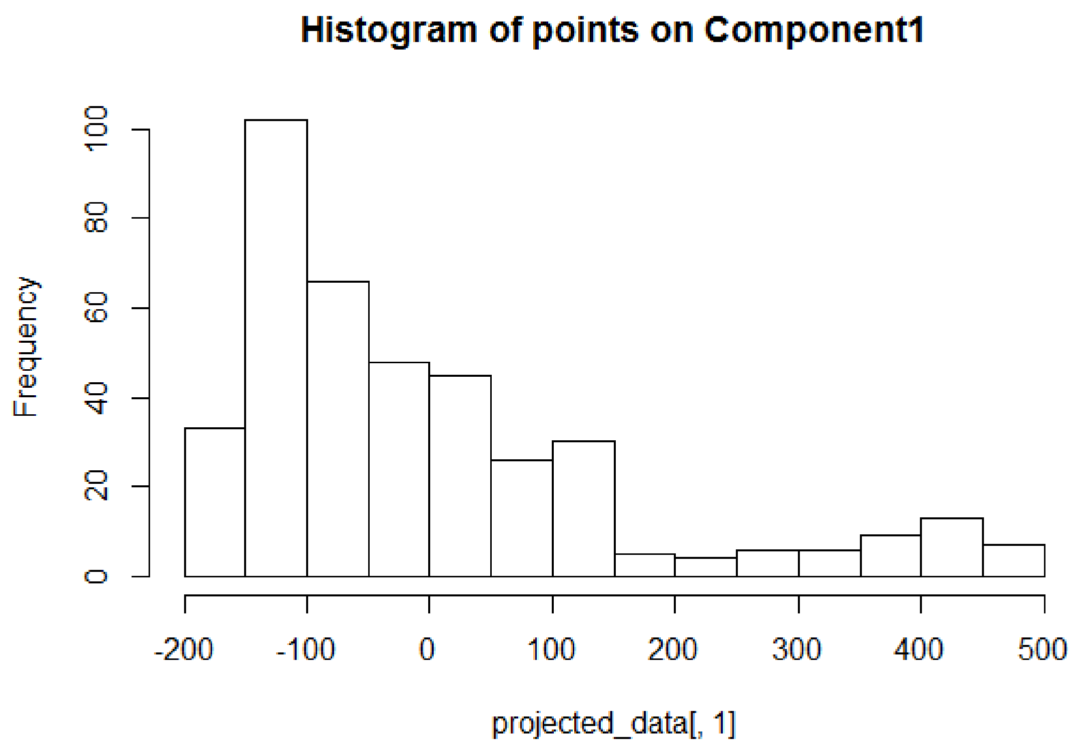
```
pca_kernel <- kpca(PCAPoints[,-3],
kernel="polydot",kpar=list(degree=2,scale=1,offset=0),features=2)
pca_loadings_kernel <- pcv(pca_kernel);
pca_eigne_vales <- eig(pca_kernel)
projected_data <- rotated(pca_kernel);
plot (projected_data[,1], xlab="Point Index",ylab="Projection on
Component 1",main="Clustering after Polynomial kernel function");
plot (projected_data[,2], xlab="Point Index",ylab="Projection on
Component 2",main="Clustering after Polynomial kernel function");
pca_variance = pca_eigne_vales/sum(pca_eigne_vales)
plot(rotated(pca_kernel),col=as.integer(PCAPoints[,3]),xlab="1st Principal
Component",ylab="2nd Principal Component",main="PCA on Polynomial
kernel function")
```

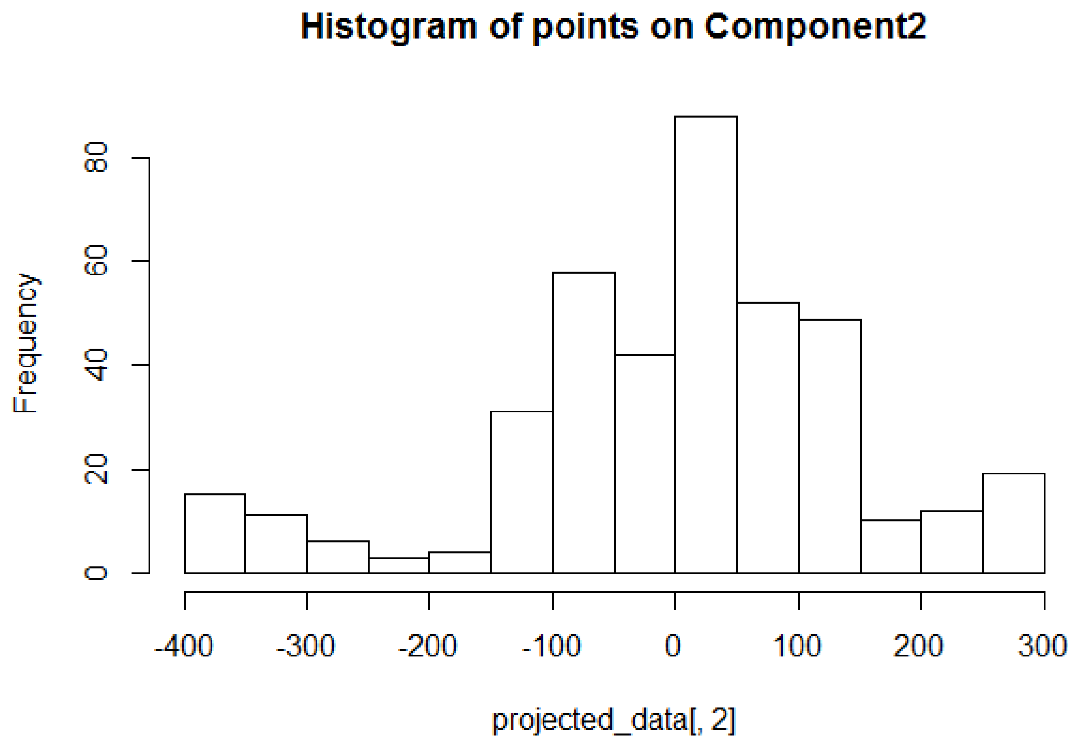
```
hist(projected_data[,1],main="Histogram of points on Component1")
hist(projected_data[,2],main="Histogram of points on Component2")
```








Histogram of points on Component1
Frequency
0
20
40
60
80
100
-200
-100
0
100
200
300
400
500
projected_data[, 1]


Histogram of points on Component2
Frequency
0
20
40
60
80
-400
-300
-200
-100
0
100
200
300
projected_data[, 2]

Proportion of Variance	0.536464	0.463536

Since varitions are not better and also points on component 1 and component 2 are not clearly separated polynomial kernel function does not perform well.

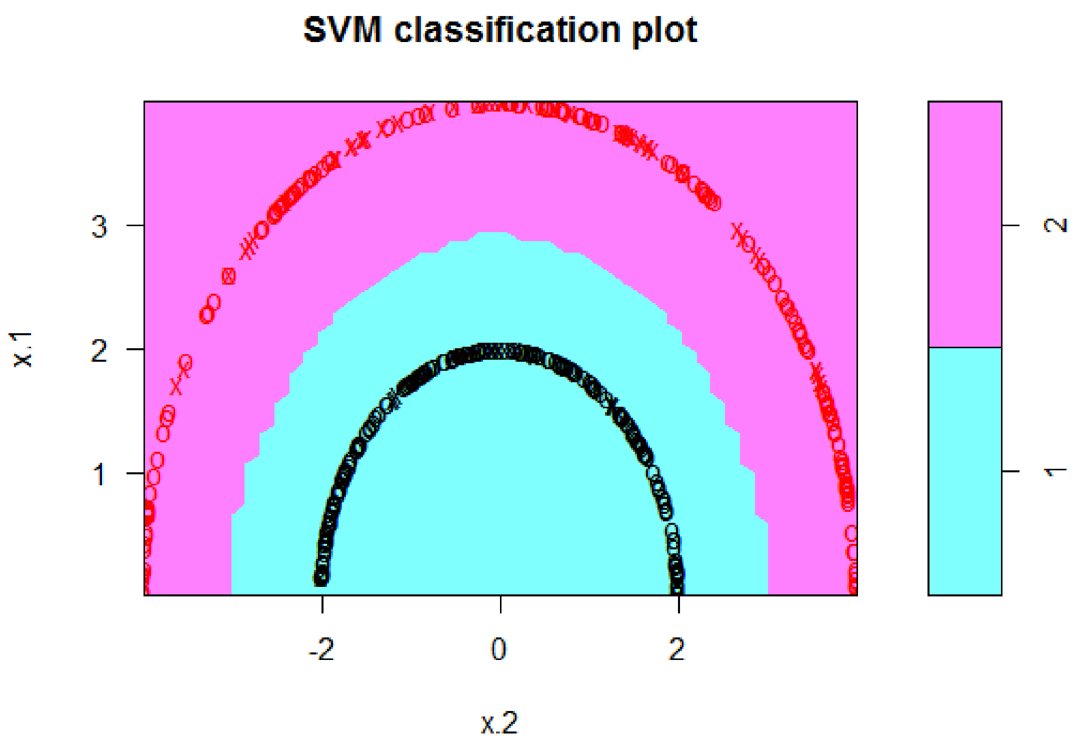
SVM

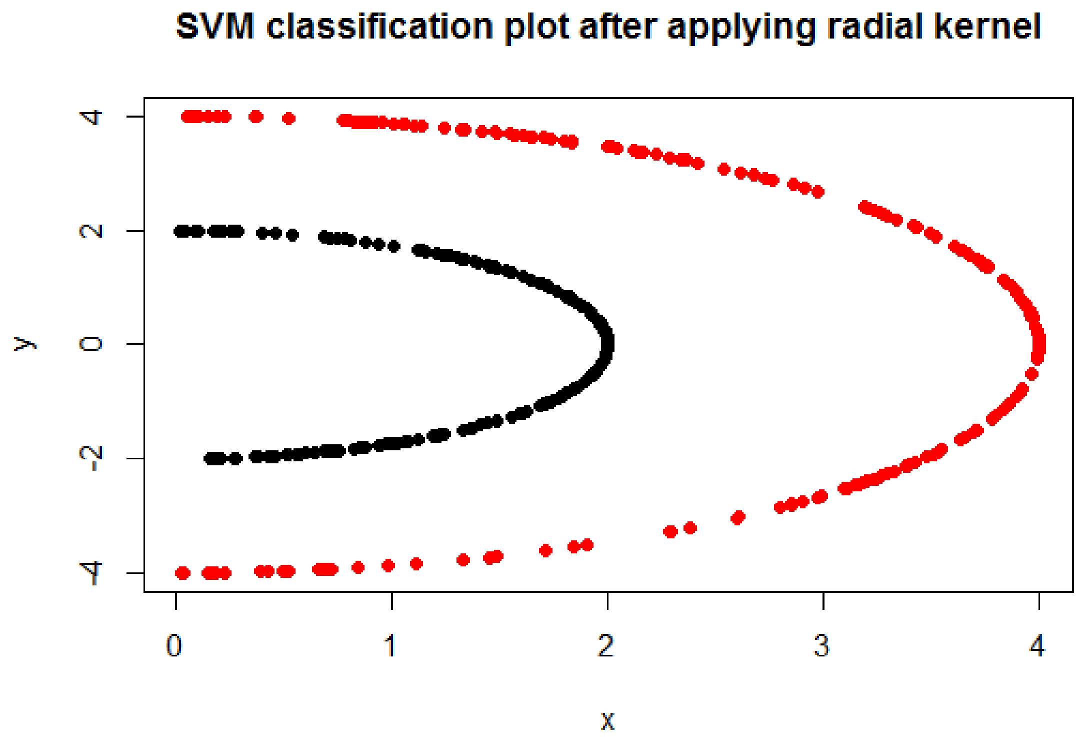
Kernel Trick:RBFDOT(Radial Basis kernel function)

```
library("e1071")
dat=data.frame(x=SVMPoints[,-3], y=as.factor(SVMPoints[,3]))
svmfit=svm(y~., data=dat , kernel ="radial", cost=10,scale=FALSE)
plot(svmfit,dat,main="SVM classification plot after applying radial kernel")
svmclusterLabel = as.matrix(svmfit$fitted)
storage.mode(svmclusterLabel) <- "integer"
svmperf = getPerformance(SVMPoints[,3],svmclusterLabel)
plot(SVMPoints[,-3], xlab="x",ylab="y", main="SVM classification plot after applying radial kernel",col=svmclusterLabel,pch=19)
```

As we see from results SVM is able to cluster points correctly after applying radial Gaussian kernel.

Performance of radial SVM	
accuracy	1
precision	1
recall	1
fscore	1


SVM classification plot
x.1
x.2
3
2
1
-2
0
2
2
1


SVM classification plot after applying radial kernel
y
x
4
2
0
-2
-4
0
1
2
3
4

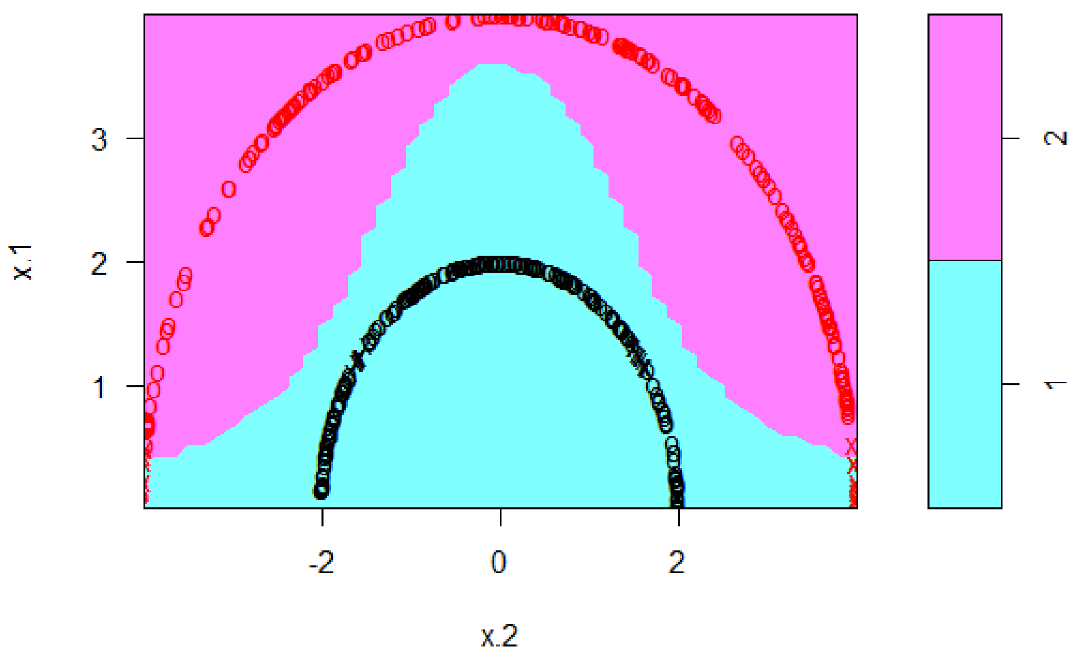
<u>Kernel Trick: polydot (Polynomial kernel function)</u>

```
library("e1071")
dat=data.frame(x=SVMPoints[,-3], y=as.factor(SVMPoints[,3]))
svmfit=svm(y~., data=dat , kernel ="polynomial", cost=10,scale=FALSE)
plot(svmfit,dat,main="SVM classification plot after applying radial kernel")
svmclusterLabel = as.matrix(svmfit$fitted)
storage.mode(svmclusterLabel) <- "integer"
svmperf = getPerformance(SVMPoints[,3],svmclusterLabel)
plot(SVMPoints[,-3], xlab="x",ylab="y", main="SVM classification plot after
applying polynomial kernel",col=svmclusterLabel,pch=19)
```

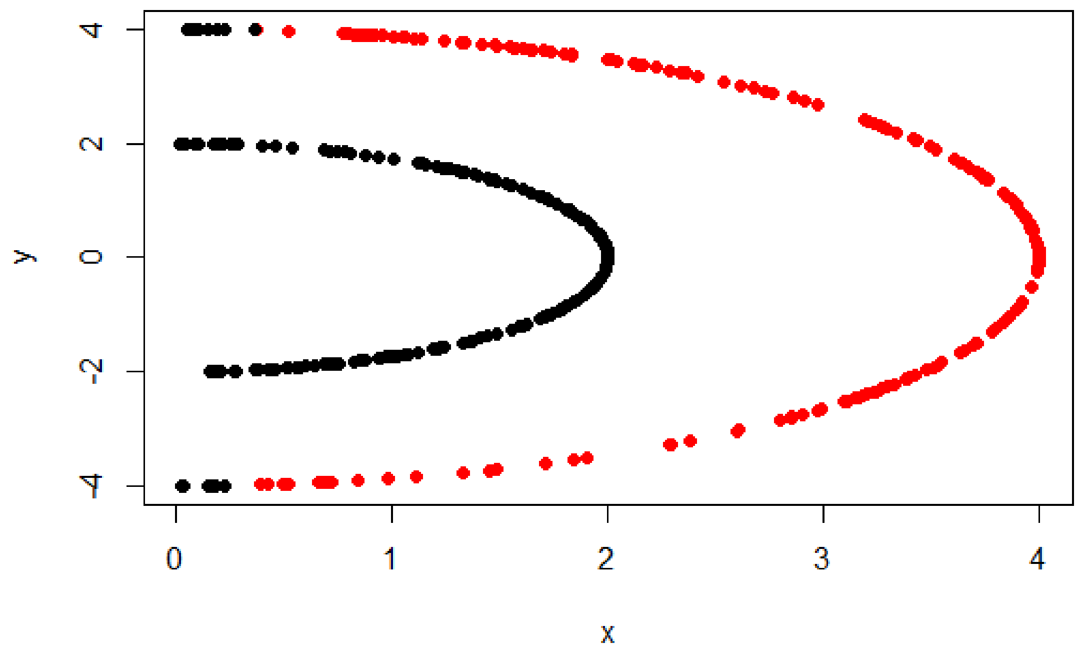
Performance of polynomial kernel SVM	
accuracy	0.9322807
precision	0.929993
recall	0.9345729
fscore	0.9322773

The results are better as compared to linear SVM.

SVM classification plot



SVM classification plot after applying polynomial kernel



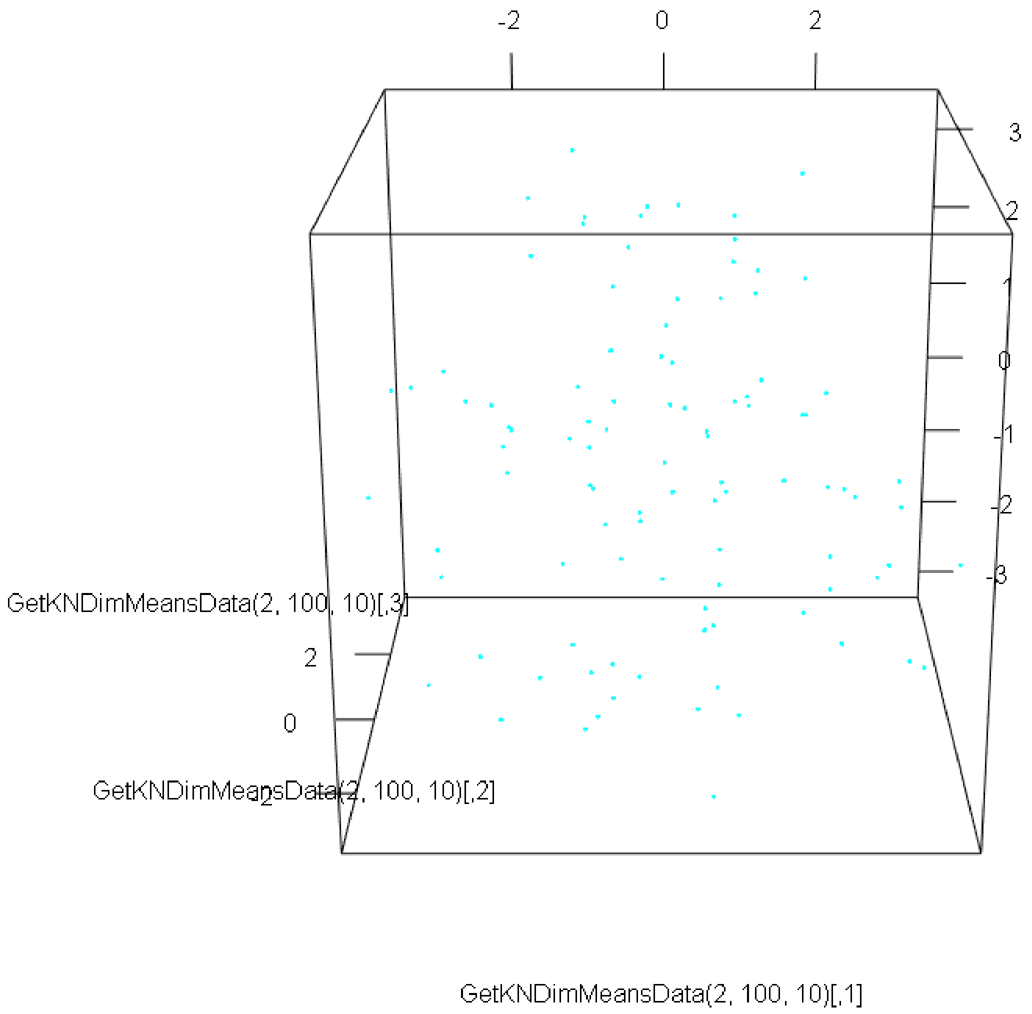
<u>Exercise 4</u>: ***Pipelining***. Dimension reduction is often used as the key data preprocessing step to other data mining techniques downstream of end-to-end data analysis. In this exercise we will use unsupervised kernel PCA as a preprocessing step to clustering. Later in the course, we will use*supervised dimension reduction methods* as a preprocessor to the supervised classification methods.

(a) Generalize your BAD_kmeans data set to very high-dimensional space (d>>2).

```
##The logic of n spehere is used to derive n dimensional data.For a n
spehere with centre as origin points will be such that
##      x1^2+x2^2+x3^2+x4^2+x5^2+......+xn^2 = r^2 where r is the radius.
##The idea used here is to first generate set of random points of n
dimension and then convert them to points in a unit radius
## n spehere and then scale it by multiplying with the radius.
GetKNDimMeansData = function(radius,numberOfPoints,numberOfDims){
library("mnormt")
points = matrix(0,nrow=numberOfPoints,ncol=numberOfDims)
mean=rep(0,numberOfDims)
Sigma = diag(length(mean))
#Generate n dimenisonal points
randomPoints = rmnorm(n = numberOfPoints, mean = rep(0,
nrow(Sigma)), Sigma);
randomPointsTemp = randomPoints^2
#Get the square root of sume of sqaures
randomPointsRowSum = sqrt(rowSums(randomPointsTemp))
randomPointsRowSum = as.matrix(randomPointsRowSum);
#Convert to nspehere of unit radius and multiply by radius to scale it up
for (i in 1:numberOfPoints) {
        points[i,] = radius*(randomPoints[i,]/randomPointsRowSum[i])
}
return(points)
}
```

#Example

```
library(rgl)
open3d()
plot3d(GetKNDimMeansData(2,100,10),col=5,type="p", radius=5)
```



(b) Show that the kmeans clustering method does not perform well on that data.

Lets apply k-means on 10 dimension data

```
#generate 200 points of  n-spehere of of 50 dimension and radius 2
NdimKmeansCluster1 = GetKNDimMeansData(2,200,50)
NdimKmeansClusterlabel1 =matrix(1,nrow=200,ncol=1)
NdimKmeansCluster1 =
cbind(NdimKmeansCluster1,NdimKmeansClusterlabel1)
#generate 200 points of n-spehere of 50 dimension and radius 10
NdimKmeansCluster2 = GetKNDimMeansData(10,200,50)
NdimKmeansClusterlabel2 =matrix(2,nrow=200,ncol=1)
NdimKmeansCluster2 =
cbind(NdimKmeansCluster2,NdimKmeansClusterlabel2)
NdimkmeansPoints = rbind(NdimKmeansCluster1,NdimKmeansCluster2)
NdimkmeansOut <- kmeans(NdimkmeansPoints[,-51],2)
```

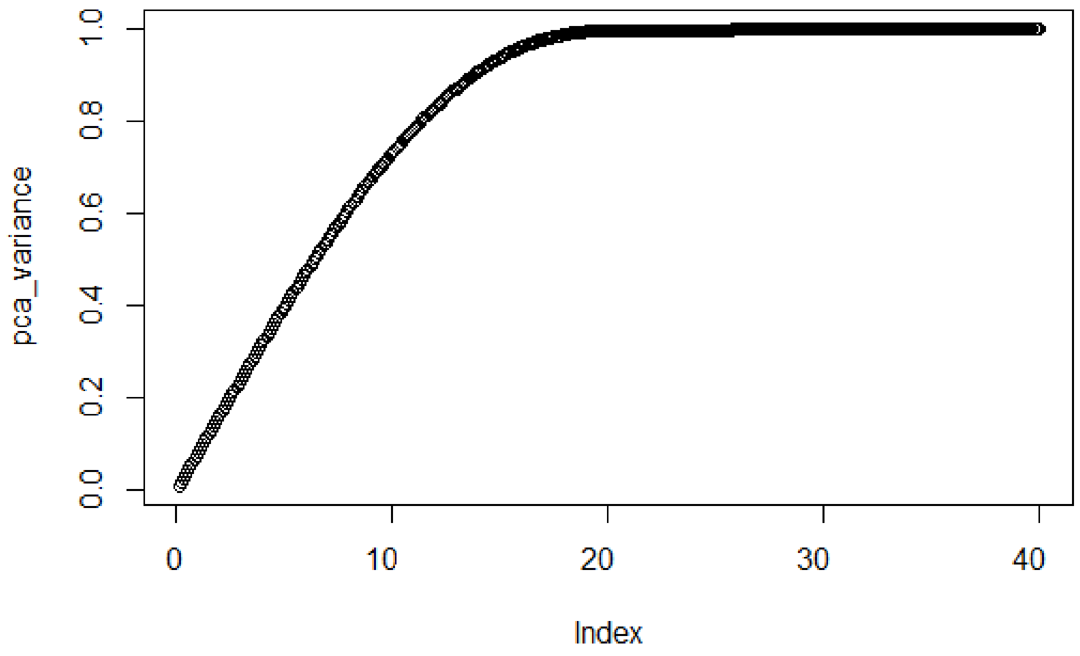
```
kemansperf =
getPerformance(NdimkmeansPoints[,51],NdimkmeansOut$cluster)
```

Performance of K-Means on 10 dimensional data	
accuracy	0.4990602
precision	0.4978355
recall	0.5056533
fscore	0.5017139

As we see from the data clustering results are bad since precision is just 50% as seen from the table.

(c) Apply the kernel PCA method to this high dimensional data and identify the number (*m*<<*d*) of principal components (i.e., eigenvectors) that provide a reasonably good low-dimensional approximation to your data (i.e., based on eigenvalue distribution). How much total variability of the data will be preserved upon using this low-dimensional representation?

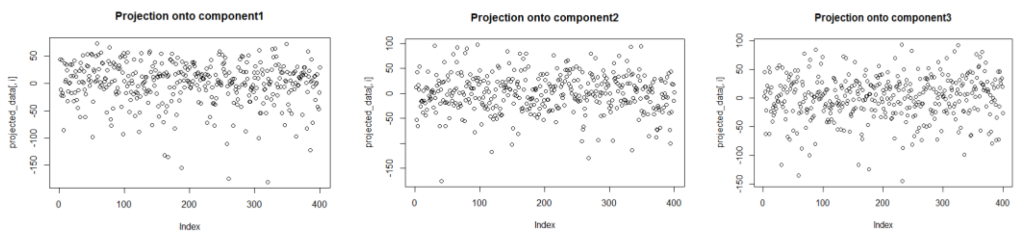
```
pca_kernel <- kpca(NdimkmeansPoints[,-51],
kernel="polydot",kpar=list(degree=2,scale=1,offset=0))
pca_loadings_kernel <- pcv(pca_kernel);
pca_eigne_vales <- eig(pca_kernel)
projected_data <- rotated(pca_kernel);
pca_variance = pca_eigne_vales/sum(pca_eigne_vales)
variablity = sum(pca_eigne_vales[1:25])/sum(pca_eigne_vales)
pca_variance_cum<-cumsum(pca_variance)
plot(pca_variance)
```

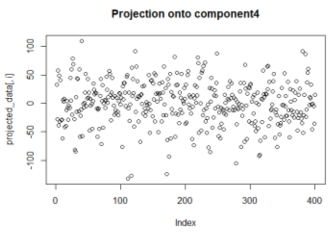
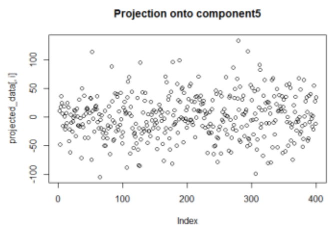
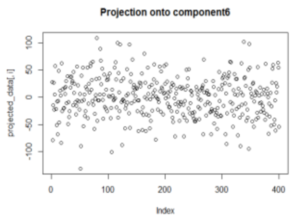
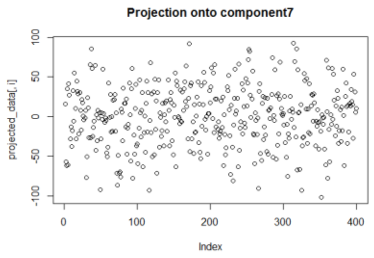
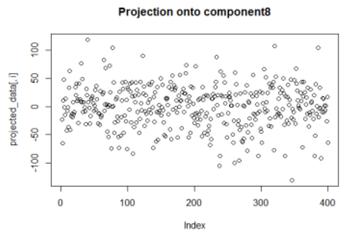
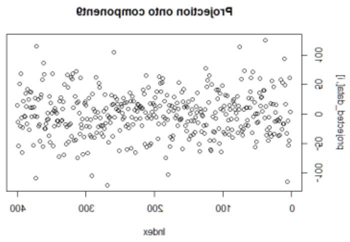
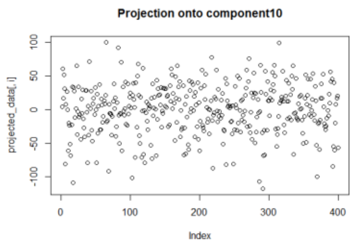
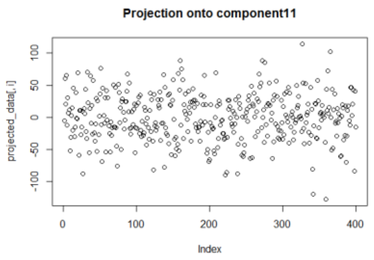
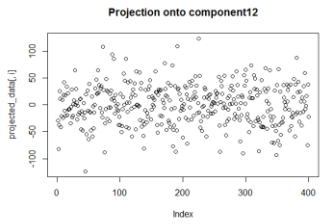
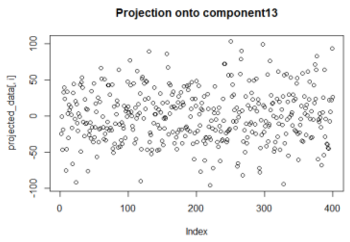
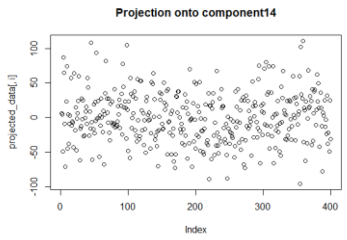
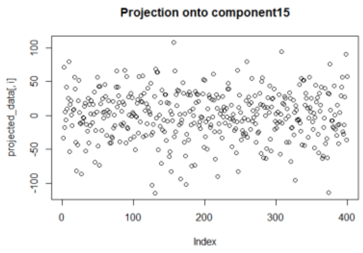
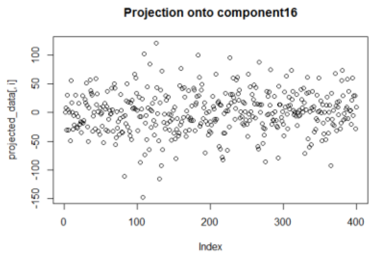
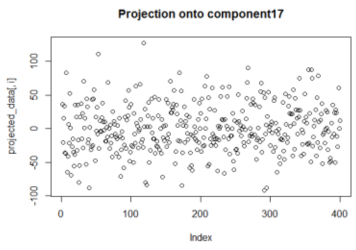
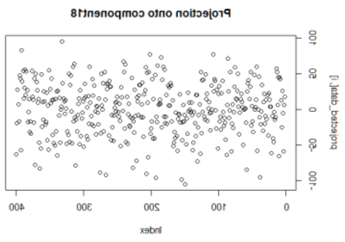


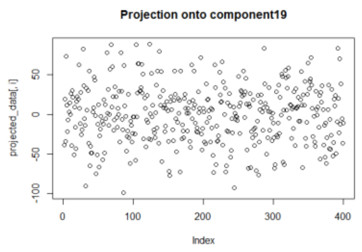
We select m as 25 since at that dimension the graph is close to cumulative pca variance of 1.
Total variability of the data will preserved = 0.9091273

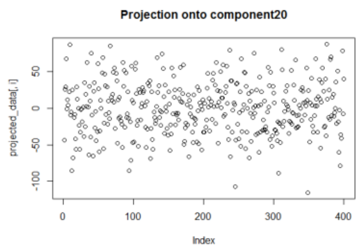
(d) Project your original data onto the top *m* eigenvectors corresponding the largest eigenvalues.

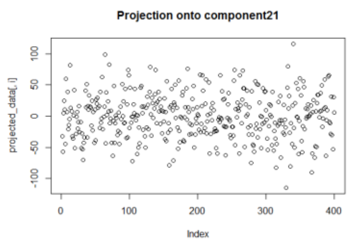
```
for (i in 1:25) {
title = paste("Projection onto component", i, sep = "")
plot(projected_data[,i],main=title)
}
```

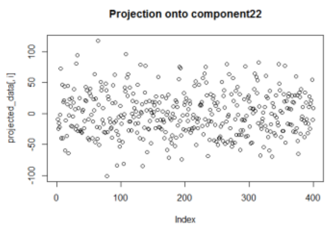


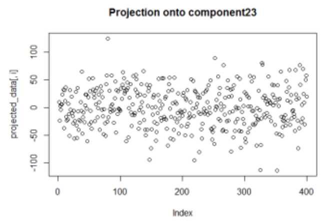

Projection onto component4
projected_data[, i]
Index

Projection onto component5
projected_data[, i]
Index

Projection onto component6
projected_data[, i]
Index

Projection onto component7
projected_data[, i]
Index

Projection onto component8
projected_data[, i]
Index

Projection onto component9
projected_data[, i]
Index

Projection onto component10
projected_data[, i]
Index

Projection onto component11
projected_data[, i]
Index

Projection onto component12
projected_data[, i]
Index

Projection onto component13
projected_data[, i]
Index

Projection onto component14
projected_data[, i]
Index

Projection onto component15
projected_data[, i]
Index

Projection onto component16
projected_data[, i]
Index

Projection onto component17
projected_data[, i]
Index

Projection onto component18
projected_data[, i]
Index

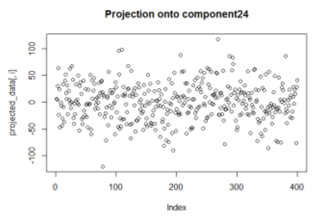


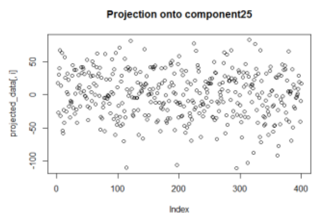












(e) Run the kmeans clustering algorithm on the projected low dimensional data.

```
NdimkmeansOut <- kmeans(projected_data[,1:25],2)
kemansperf =
getPerformance(NdimkmeansPoints[,51],NdimkmeansOut$cluster)
```

(f) Compare the performance of the kmeans on d-dimensional original data vs. the m-dimensional projected data. Has the performance improved? There is increase in the performance as seen from the table below.

Performance has increased by a great margin as seen from the table.

Performance of K-Means on 10 dimensional data		Without dimension reduction
accuracy	0.7553602	0.4990602
precision	0.755055	0.4978355
recall	0.7676533	0.5056533
fscore	0.7557139	0.5017139

(g) If you run the kernel kmeans clustering method on the original data, will get better/worse performance? Can you discuss the pros and cons of using kernel kmeans on the original data directly versus applying the kernel pca as the pre-processing step and then running the kmeans on the low-dimensional data

```
dat=NdimkmeansPoints[,-51]
kkmeansPoly = specc(dat,2,kernel="rbfdot")
kemansperf = getPerformance(NdimkmeansPoints[,51],kkmeansLabel)
```

Performance of K-Means on 50 dimensional data after kernel trick	
accuracy	0.635732
precision	0.629083
recall	0.657374
fscore	0.6496381

By applying kernel trick there is a slight improvement in the performance. But this performance is not better than performance achieved by dimension reduction of the data.

Dimension reduction reduces the computational complexity since we are reducing the dimensions. However kernel tricks may take more time and resources since we are dealing with higher dimension data.