Project III: Kernel PCA and Association Rule

Appiah Prince* University of Texas at El Paso (UTEP)

October 11, 2022

Contents

*pappiah@miners.utep.edu

1 Bring in and examine the data

1.1 Bring in both the train and the test data

```
# BRING IN THE DATA
train <- read.table(file=
"http://archive.ics.uci.edu/ml/machine-learning-databases/optdigits/optdigits.tra",
sep=",", header = FALSE, na.strings = c("NA", "", " "),
col.names = c(paste("x", 1:64, sep=""), "digit"))
test <- read.table(file=
"http://archive.ics.uci.edu/ml/machine-learning-databases/optdigits/optdigits.tes",
sep=",", header = FALSE, na.strings = c("NA", "", " "),
col.names = c(paste("x", 1:64, sep=""), "digit"))

dim(train)
```

```
## [1] 3823   65
```

```
dim(test)
```

```
## [1] 1797   65
```

1.2 Checking for columns that are unary or close to unary

```
library(caret)
nearZeroVar(train[,-65], uniqueCut = 10, saveMetrics = TRUE)
```

```
##       freqRatio percentUnique zeroVar   nzv
## x1     0.000000    0.02615747    TRUE  TRUE
## x2    10.912162    0.23541721   FALSE FALSE
## x3     3.096296    0.44467696   FALSE FALSE
## x4     1.591376    0.44467696   FALSE FALSE
## x5     1.742919    0.44467696   FALSE FALSE
## x6     3.288235    0.44467696   FALSE FALSE
## x7    14.912821    0.44467696   FALSE FALSE
## x8   132.464286    0.41851949   FALSE  TRUE
## x9  3820.000000    0.10462987   FALSE  TRUE
## x10    7.327703    0.41851949   FALSE FALSE
## x11    1.950324    0.44467696   FALSE FALSE
## x12    2.312020    0.44467696   FALSE FALSE
```

```
## x13    2.734628    0.44467696   FALSE FALSE
## x14    1.256849    0.44467696   FALSE FALSE
## x15   11.789474    0.44467696   FALSE FALSE
## x16  108.264706    0.39236202   FALSE  TRUE
## x17  953.500000    0.13078734   FALSE  TRUE
## x18    6.856089    0.44467696   FALSE FALSE
## x19    1.407407    0.44467696   FALSE FALSE
## x20    1.744000    0.44467696   FALSE FALSE
## x21    1.458861    0.44467696   FALSE FALSE
## x22    1.679339    0.44467696   FALSE FALSE
## x23   10.776256    0.44467696   FALSE FALSE
## x24  187.800000    0.23541721   FALSE  TRUE
## x25  954.750000    0.05231494   FALSE  TRUE
## x26    6.376667    0.44467696   FALSE FALSE
## x27    1.296804    0.44467696   FALSE FALSE
## x28    1.592129    0.44467696   FALSE FALSE
## x29    1.497948    0.44467696   FALSE FALSE
## x30    1.368601    0.44467696   FALSE FALSE
## x31   10.923077    0.44467696   FALSE FALSE
## x32  476.625000    0.07847240   FALSE  TRUE
## x33  763.600000    0.05231494   FALSE  TRUE
## x34   10.218750    0.41851949   FALSE FALSE
## x35    1.430473    0.44467696   FALSE FALSE
## x36    1.457386    0.44467696   FALSE FALSE
## x37    2.160878    0.44467696   FALSE FALSE
## x38    1.200000    0.44467696   FALSE FALSE
## x39    5.162319    0.39236202   FALSE FALSE
## x40    0.000000    0.02615747    TRUE  TRUE
## x41  344.000000    0.20925974   FALSE  TRUE
## x42   11.033473    0.44467696   FALSE FALSE
## x43    2.636861    0.44467696   FALSE FALSE
## x44    1.878834    0.44467696   FALSE FALSE
## x45    1.222982    0.44467696   FALSE FALSE
## x46    1.223035    0.44467696   FALSE FALSE
## x47    7.383333    0.44467696   FALSE FALSE
## x48  117.968750    0.13078734   FALSE  TRUE
## x49  189.550000    0.20925974   FALSE  TRUE
## x50   10.383513    0.44467696   FALSE FALSE
## x51    1.899743    0.44467696   FALSE FALSE
## x52    3.366534    0.44467696   FALSE FALSE
## x53    3.143396    0.44467696   FALSE FALSE
## x54    1.010249    0.44467696   FALSE FALSE
## x55    9.602094    0.44467696   FALSE FALSE
## x56   58.370968    0.28773215   FALSE  TRUE
## x57 3822.000000    0.05231494   FALSE  TRUE
```

```
## x58   17.326425    0.28773215    FALSE FALSE
## x59    3.471774    0.44467696    FALSE FALSE
## x60    1.907950    0.44467696    FALSE FALSE
## x61    2.320930    0.44467696    FALSE FALSE
## x62    2.797222    0.44467696    FALSE FALSE
## x63    9.741313    0.44467696    FALSE FALSE
## x64   53.970149    0.41851949    FALSE  TRUE
```

```r
nearZeroVar(train[,-65], names = TRUE)
```

```
##  [1] "x1"  "x8"  "x9"  "x16" "x17" "x24" "x25" "x32" "x33" "x40" "x41" "x48"
## [13] "x49" "x56" "x57" "x64"
```

```r
library(caret)
nearZeroVar(test[,-65], uniqueCut = 10, saveMetrics = TRUE)
```

```
##        freqRatio percentUnique zeroVar   nzv
## x1      0.000000     0.0556483    TRUE  TRUE
## x2     11.960938     0.5008347   FALSE FALSE
## x3      3.006993     0.9460211   FALSE FALSE
## x4      1.735160     0.9460211   FALSE FALSE
## x5      1.800000     0.9460211   FALSE FALSE
## x6      2.987879     0.9460211   FALSE FALSE
## x7     15.211111     0.9460211   FALSE FALSE
## x8    145.750000     0.8903728   FALSE  TRUE
## x9    447.500000     0.1669449   FALSE  TRUE
## x10     8.540323     0.9460211   FALSE FALSE
## x11     2.192308     0.9460211   FALSE FALSE
## x12     2.193237     0.9460211   FALSE FALSE
## x13     2.575540     0.9460211   FALSE FALSE
## x14     1.361011     0.9460211   FALSE FALSE
## x15    10.067797     0.9460211   FALSE FALSE
## x16   109.250000     0.7234279   FALSE  TRUE
## x17   597.666667     0.1669449   FALSE  TRUE
## x18     6.770992     0.9460211   FALSE FALSE
## x19     1.675556     0.9460211   FALSE FALSE
## x20     1.483193     0.9460211   FALSE FALSE
## x21     1.513605     0.9460211   FALSE FALSE
## x22     2.016529     0.9460211   FALSE FALSE
## x23    10.924528     0.9460211   FALSE FALSE
## x24   147.000000     0.4451864   FALSE  TRUE
## x25   897.500000     0.1112966   FALSE  TRUE
## x26     6.287770     0.8903728   FALSE FALSE
```

```
## x27    1.244648     0.9460211   FALSE FALSE
## x28    1.388462     0.9460211   FALSE FALSE
## x29    1.549521     0.9460211   FALSE FALSE
## x30    1.504202     0.9460211   FALSE FALSE
## x31    9.705357     0.8903728   FALSE FALSE
## x32  448.250000     0.1112966   FALSE  TRUE
## x33    0.000000     0.0556483    TRUE  TRUE
## x34   10.915789     0.8347245   FALSE FALSE
## x35    1.354167     0.9460211   FALSE FALSE
## x36    1.352941     0.9460211   FALSE FALSE
## x37    1.894545     0.9460211   FALSE FALSE
## x38    1.070470     0.9460211   FALSE FALSE
## x39    4.807910     0.8347245   FALSE FALSE
## x40    0.000000     0.0556483    TRUE  TRUE
## x41  357.600000     0.2782415   FALSE  TRUE
## x42    8.862595     0.9460211   FALSE FALSE
## x43    2.284698     0.9460211   FALSE FALSE
## x44    1.823529     0.9460211   FALSE FALSE
## x45    1.300912     0.9460211   FALSE FALSE
## x46    1.698020     0.9460211   FALSE FALSE
## x47    8.714286     0.9460211   FALSE FALSE
## x48  147.916667     0.3895381   FALSE  TRUE
## x49  896.500000     0.2225932   FALSE  TRUE
## x50    9.622378     0.8347245   FALSE FALSE
## x51    2.148810     0.9460211   FALSE FALSE
## x52    2.562500     0.9460211   FALSE FALSE
## x53    2.645161     0.9460211   FALSE FALSE
## x54    1.283217     0.9460211   FALSE FALSE
## x55    9.559140     0.9460211   FALSE FALSE
## x56   52.531250     0.6677796   FALSE  TRUE
## x57 1796.000000     0.1112966   FALSE  TRUE
## x58   17.533333     0.5564830   FALSE FALSE
## x59    3.162963     0.9460211   FALSE FALSE
## x60    2.365854     0.9460211   FALSE FALSE
## x61    2.724138     0.9460211   FALSE FALSE
## x62    2.590426     0.9460211   FALSE FALSE
## x63    8.822222     0.9460211   FALSE FALSE
## x64   58.172414     0.9460211   FALSE  TRUE
```

```r
nearZeroVar(test[,-65], names = TRUE)
```

```
##  [1] "x1"  "x8"  "x9"  "x16" "x17" "x24" "x25" "x32" "x33" "x40" "x41" "x48"
## [13] "x49" "x56" "x57" "x64"
```

```r
train <- train[,-c(1,8,9,16,17,24,25,32,33,40,41,48,49,56,57,64)]

test <- test[,-c(1,8,9,16,17,24,25,32,33,40,41,48,49,56,57,64)]
```

Remarks

I removed the columns 1,8,9,16,17,24,25,32,33,40,41,48,49,56,57,64 from the columns of both train and test data since they have some values that are unary and some close to unary.

1.3 Checking for missing values

```r
library(questionr)
freq.na(train)
```

```
##       missing %
## x2          0 0
## x3          0 0
## x4          0 0
## x5          0 0
## x6          0 0
## x7          0 0
## x10         0 0
## x11         0 0
## x12         0 0
## x13         0 0
## x14         0 0
## x15         0 0
## x18         0 0
## x19         0 0
## x20         0 0
## x21         0 0
## x22         0 0
## x23         0 0
## x26         0 0
## x27         0 0
## x28         0 0
## x29         0 0
## x30         0 0
## x31         0 0
## x34         0 0
## x35         0 0
```

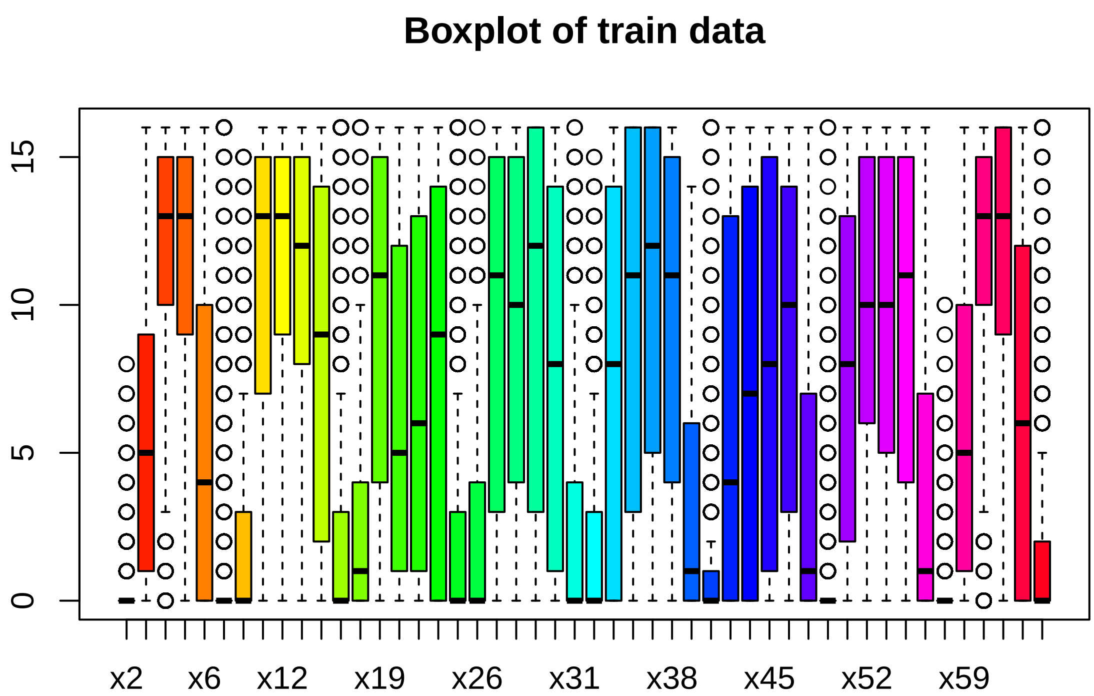
```
## x36         0 0
## x37         0 0
## x38         0 0
## x39         0 0
## x42         0 0
## x43         0 0
## x44         0 0
## x45         0 0
## x46         0 0
## x47         0 0
## x50         0 0
## x51         0 0
## x52         0 0
## x53         0 0
## x54         0 0
## x55         0 0
## x58         0 0
## x59         0 0
## x60         0 0
## x61         0 0
## x62         0 0
## x63         0 0
## digit       0 0
```

Remarks

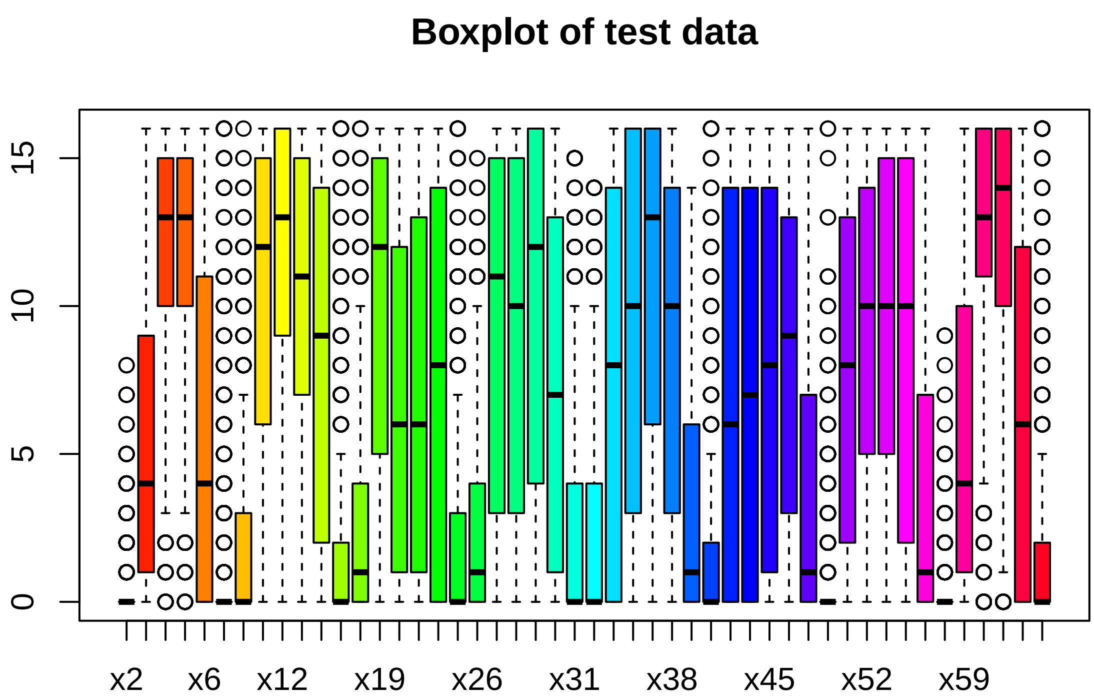
There are no missing values in the train data.

2 Ordinary Principal Components Analysis (PCA)

```
# Parallel Boxplot of the attributes of the train data
boxplot(train[,-49], col = rainbow(ncol(train[,-49])), main="Boxplot of train data")
```




```
# Parallel Boxplot of the attributes of the test data
boxplot(test[,-49], col = rainbow(ncol(test[,-49])), main="Boxplot of test data")
```

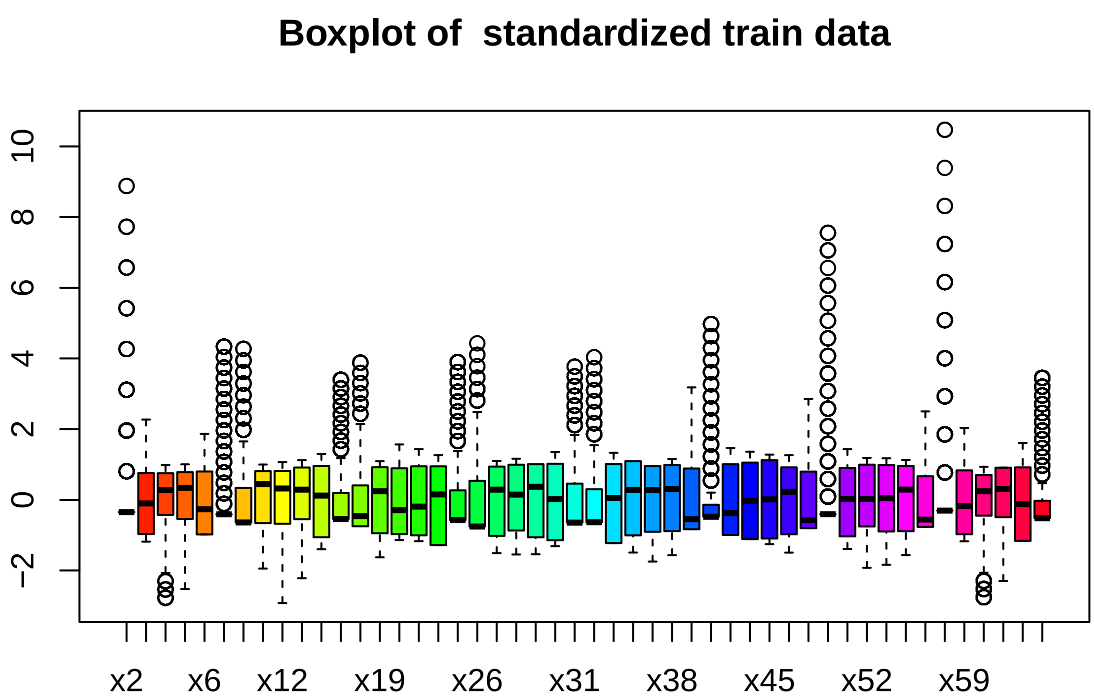


Remarks

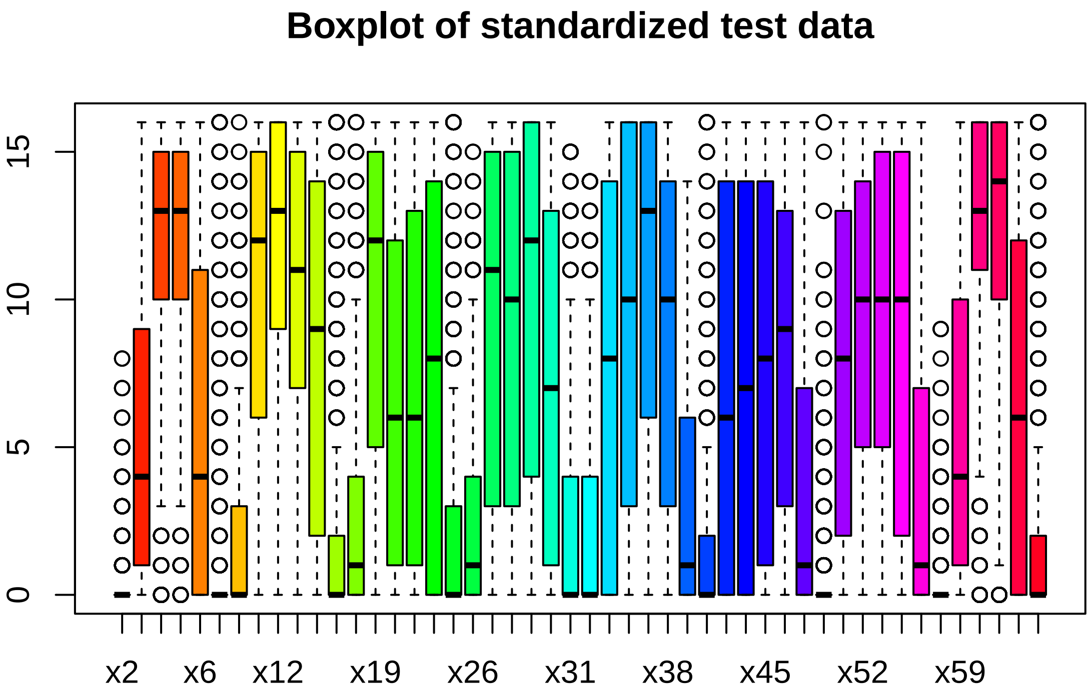
- Majority of the predictors in both the train and test data have unequal range and unequal variation.
- Hence,scaling is necessary for some modeling approaches.

```
# scaling the train and test data
train_scaled <- data.frame(apply(train[,-49], 2, scale,center=T, scale=T))

mean <- apply(train_scaled, 2, mean)
sd <-  apply(train_scaled, 2, sd)
test_scaled <- data.frame(scale(test[, -49], center = mean, scale = sd))
```

```
boxplot(train_scaled, col = rainbow(ncol(train_scaled)), main="Boxplot of  standardized
```




```
boxplot(test_scaled, col = rainbow(ncol(test_scaled)), main="Boxplot of standardized tes
```



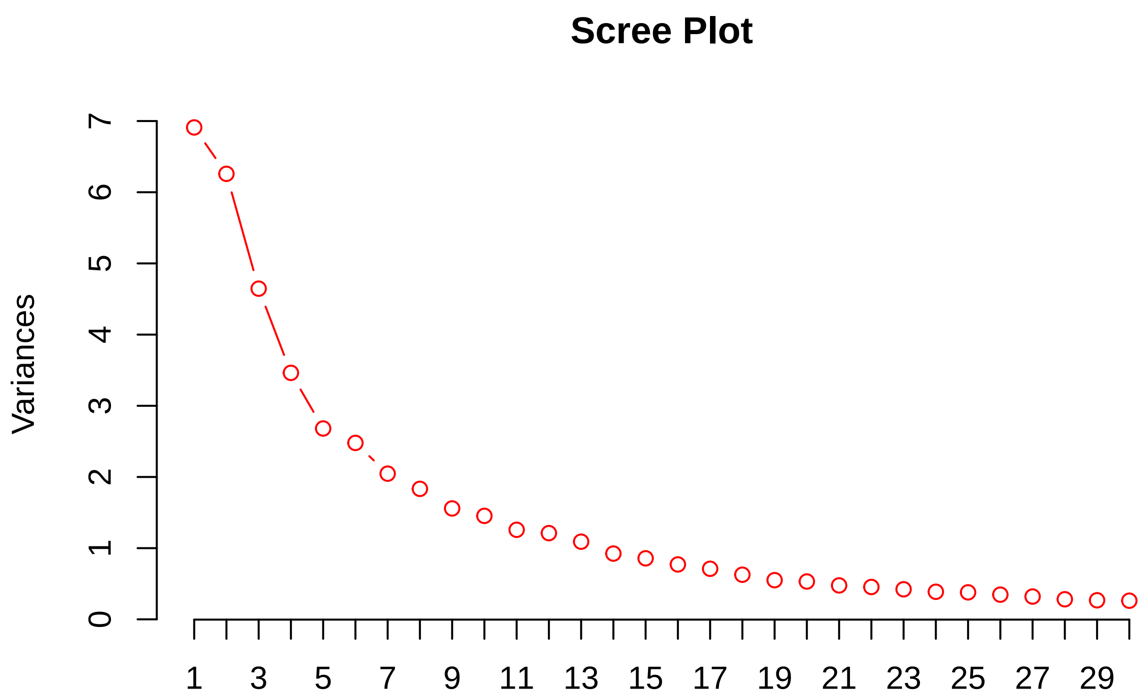
Remarks

After scaling both the test and train data, we see that very few of the attributes of test and train data have unequal range and variation. Hence,we can now run the ordinary principal components analysis (PCA).

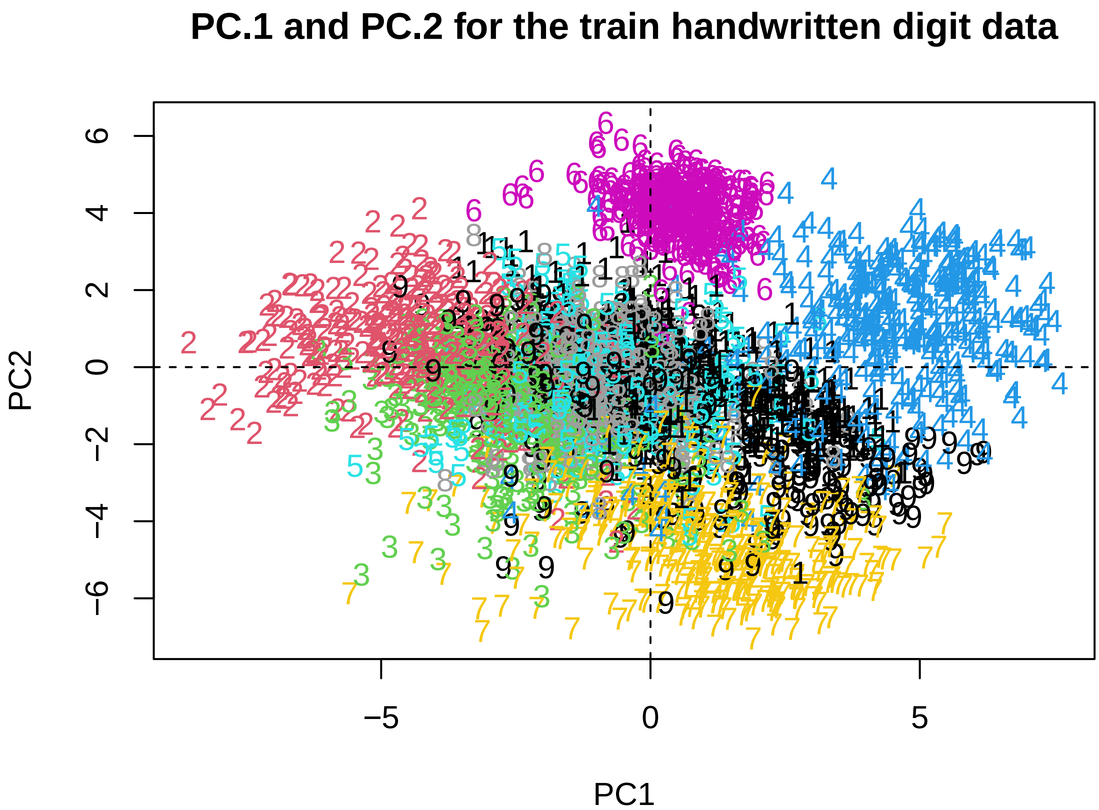
```
pca <- prcomp(train_scaled, retx=TRUE, center=F, scale=F)

# OBTAIN EIGENVALUES
lambda <- eigen(cov(train_scaled), only.values = T)$values
lambda
```

```
##  [1] 6.91007451 6.25858444 4.64555316 3.46269437 2.68099761 2.47851068
##  [7] 2.04654171 1.83311085 1.55795400 1.45344719 1.25778760 1.21108619
## [13] 1.09097620 0.92376560 0.85699940 0.77122785 0.71038203 0.62704218
## [19] 0.55046754 0.53121523 0.47656857 0.45452988 0.42244094 0.38724553
## [25] 0.37911851 0.34677924 0.32034160 0.28199851 0.26768836 0.26197379
## [31] 0.25311899 0.21488696 0.20678233 0.19538461 0.17944841 0.16851168
## [37] 0.16195405 0.15399054 0.13799300 0.13102482 0.12169469 0.11560572
## [43] 0.10320643 0.10273704 0.09181544 0.07903824 0.06715225 0.05855154
```

```
#screeplot of variance
screeplot(pca, npcs = 30,  type="lines", main="Scree Plot", col = "red")
```




```
# PLOT FIRST TWO PCs
par(mfrow=c(1,1), mar=rep(4,4))
plot(pca$x[,1:2], pch="", main="PC.1 and PC.2 for the train handwritten digit data")
text(pca$x[,1:2], labels=train$digit, col= train$digit)
abline(v=0, lty=2)
abline(h=0, lty=2)
```



Remarks

We can see from the plot graph that the first two PCs fairly successfully separate the digits. We see, for instance, that most 6s lie on the top of the plot, most 4s lie on the upper right, most 7s on the bottom right, and most 2s on the middle-top left. There are however regions of overlap.

3 Kernel PCA

```
# Using different kernel functions
library(kernlab)
```

```
##
## Attaching package: 'kernlab'

## The following object is masked from 'package:ggplot2':
##
##     alpha
```

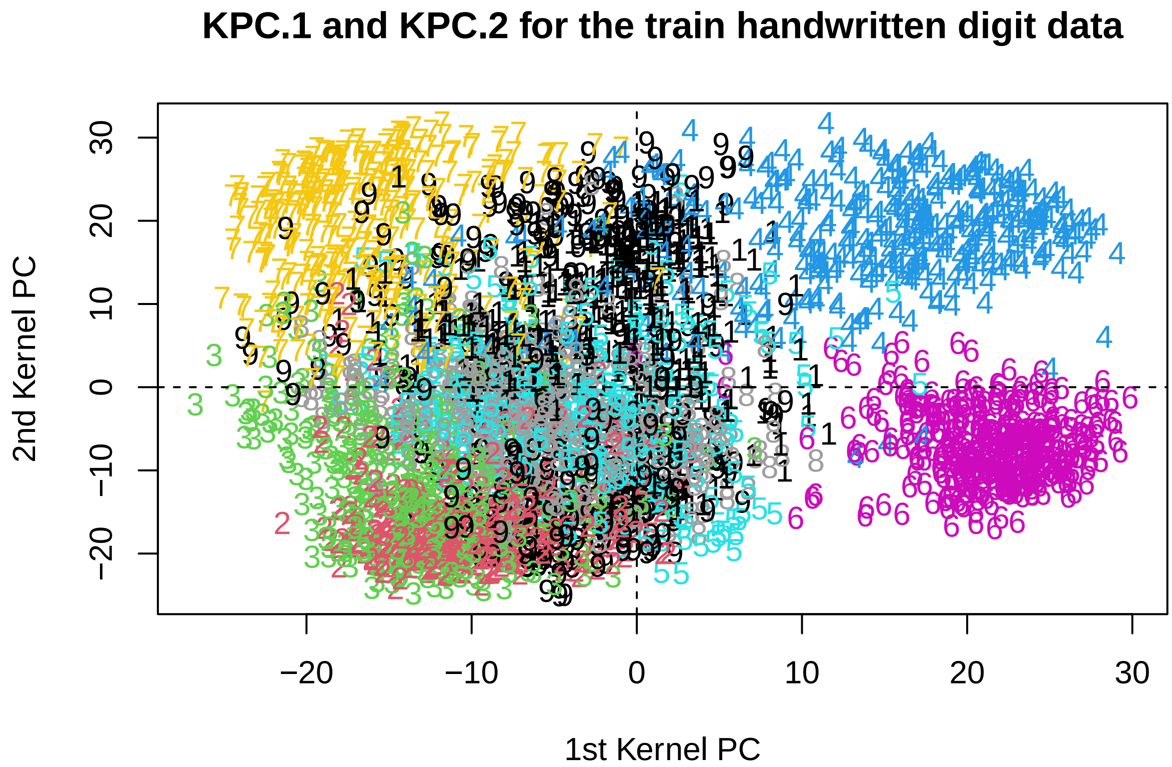
```
kernel_pca1 <- kpca(~., data=train_scaled, kernel="rbfdot", kpar=list(sigma=0.01),featur

kernel_pca2 <- kpca(~., data=train_scaled, kernel="vanilladot",kpar=list(),
                    features = 10)
kernel_pca3 <- kpca(~.,data=train_scaled, kernel="polydot", kpar=list(degree=2),
                    features=10)
kernel_pca4 <- kpca(~.,data=train_scaled,kernel="laplacedot",
                    kpar=list(sigma=0.01),features=10)
```

```
# Get the variance of each kernel pca.
var.pc1 <- eig(kernel_pca1)
var.pc2 <- eig(kernel_pca2)
var.pc3 <- eig(kernel_pca3)
var.pc4 <- eig(kernel_pca4)
variance <- data.frame(var.pc1, var.pc2, var.pc3, var.pc4)
variance
```

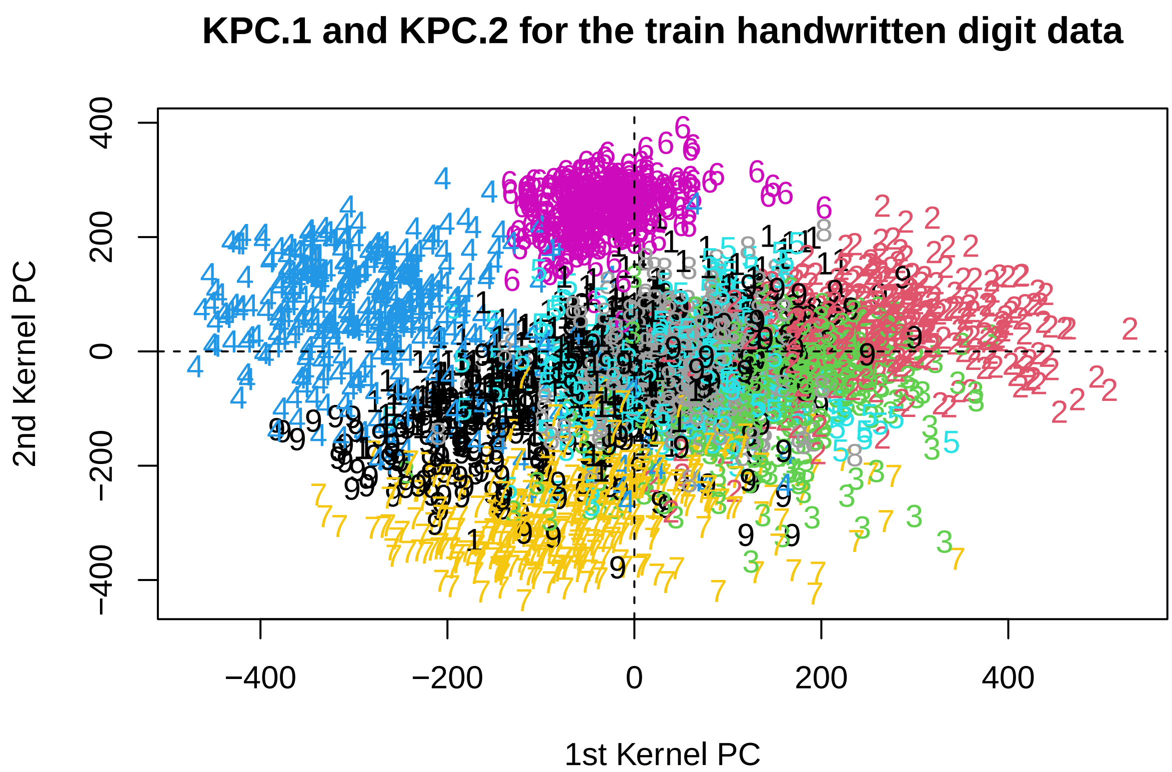
```
##             var.pc1  var.pc2   var.pc3     var.pc4
## Comp.1  0.05023886 6.908267 159.43549 0.006244509
## Comp.2  0.04913771 6.256947 139.35988 0.006058360
## Comp.3  0.04201080 4.644338 119.44195 0.004937771
## Comp.4  0.02887515 3.461789  93.99018 0.003473661
## Comp.5  0.02499205 2.680296  78.91073 0.002813834
## Comp.6  0.02282597 2.477862  75.12011 0.002615203
## Comp.7  0.02019387 2.046006  67.26845 0.002314876
## Comp.8  0.01788108 1.832631  63.87454 0.002129724
## Comp.9  0.01420419 1.557546  56.27348 0.001654528
## Comp.10 0.01301382 1.453067  43.95150 0.001544828
```

- Plotting the first two PCs for each of the kernel pca

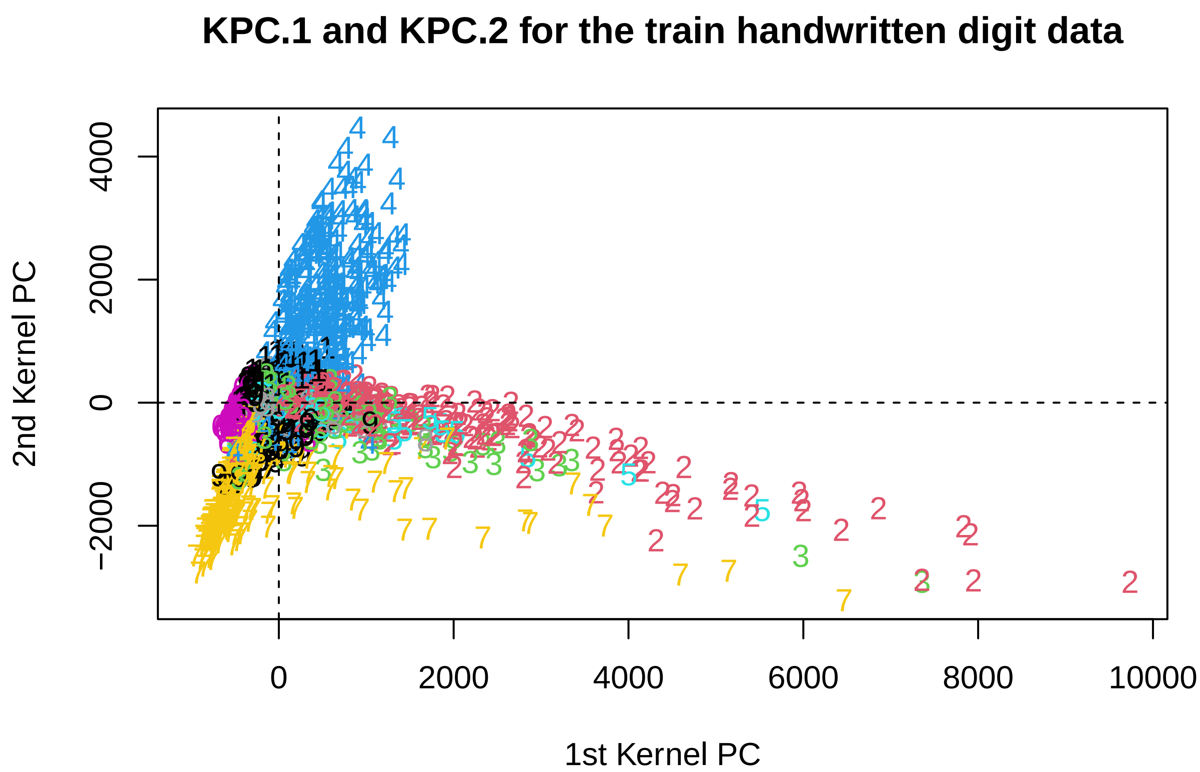
```
PC1 <- rotated(kernel_pca1)     # returns the data projected in the (kernel) pca space
plot(PC1[, 1:2],col=train$digit, pch="",
     main="KPC.1 and KPC.2 for the train handwritten digit data",
     xlab="1st Kernel PC", ylab="2nd Kernel PC")
text(PC1[, 1:2], labels=train$digit, col= train$digit)
abline(v=0, lty=2)
abline(h=0, lty=2)
```



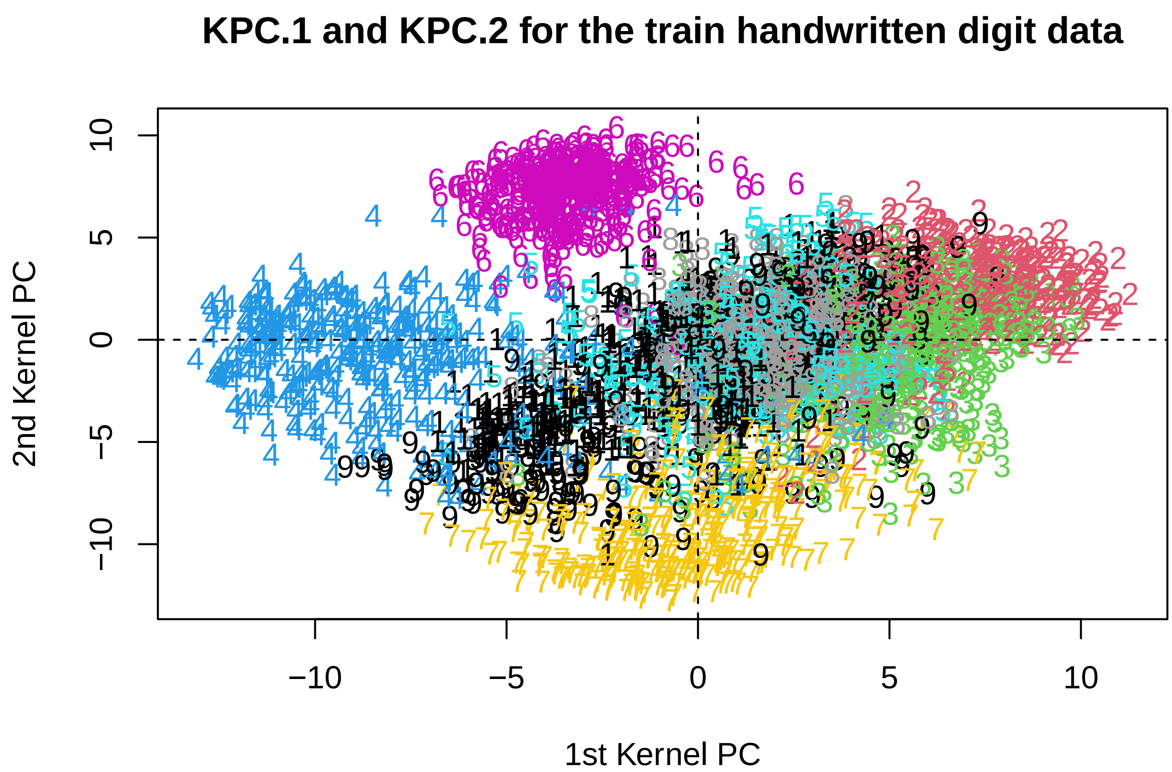

```r
PC2 <- rotated(kernel_pca2)     # returns the data projected in the (kernel) pca space
plot(PC2[, 1:2],col=train$digit, pch="",
     main="KPC.1 and KPC.2 for the train handwritten digit data",
     xlab="1st Kernel PC", ylab="2nd Kernel PC")
text(PC2[, 1:2], labels=train$digit, col= train$digit)
abline(v=0, lty=2)
abline(h=0, lty=2)
```




```r
PC3 <- rotated(kernel_pca3)     # returns the data projected in the (kernel) pca space
plot(PC3[, 1:2],col=train$digit, pch="",
     main="KPC.1 and KPC.2 for the train handwritten digit data",
     xlab="1st Kernel PC", ylab="2nd Kernel PC")
text(PC3[, 1:2], labels=train$digit, col= train$digit)
abline(v=0, lty=2)
abline(h=0, lty=2)
```



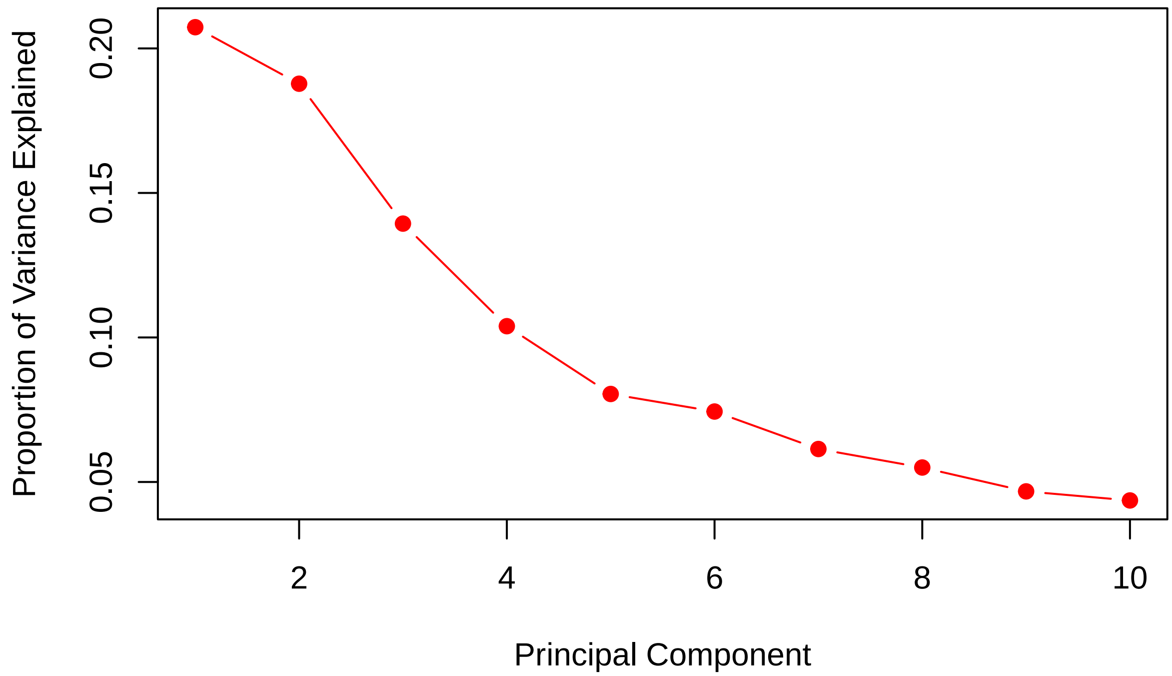

```r
PC4 <- rotated(kernel_pca4)     # returns the data projected in the (kernel) pca space
plot(PC4[, 1:2],col=train$digit, pch="",
     main="KPC.1 and KPC.2 for the train handwritten digit data",
     xlab="1st Kernel PC", ylab="2nd Kernel PC")
text(PC4[, 1:2], labels=train$digit, col= train$digit)
abline(v=0, lty=2)
abline(h=0, lty=2)
```



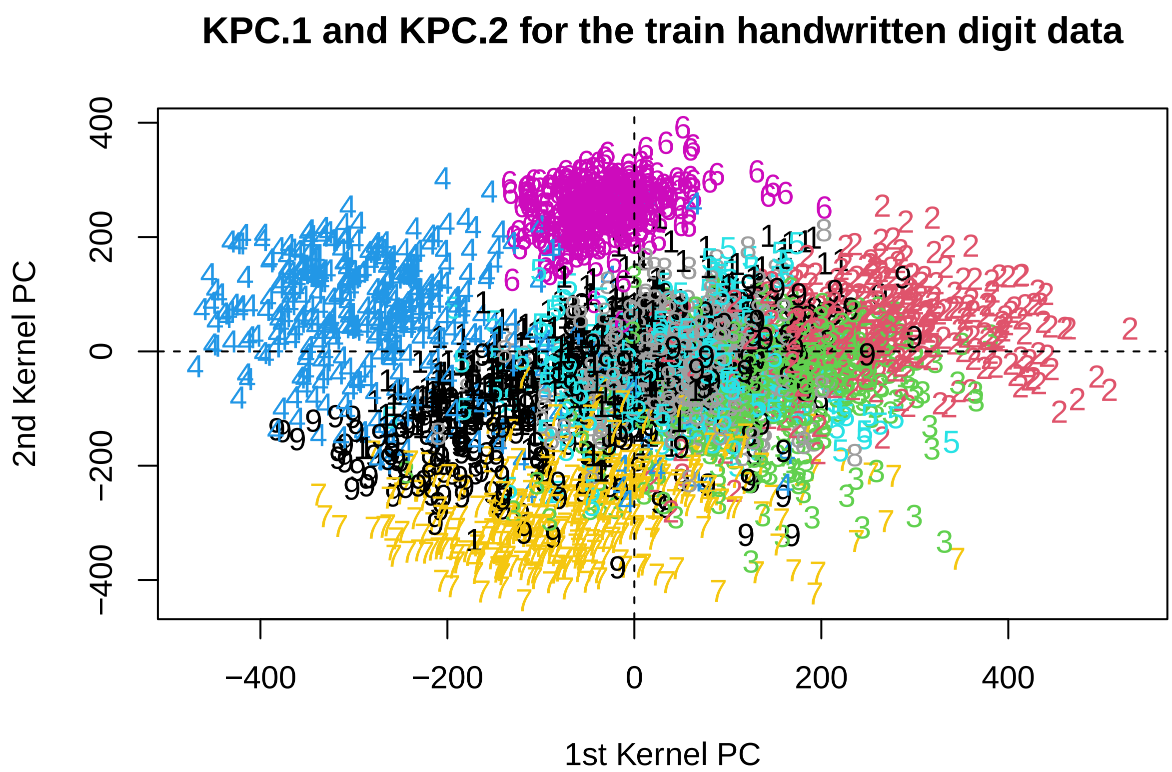
Remarks

We observed that the kernel pca using the vanilladot kernel function separated or clustered the digits well as compared to the other kernel pca's using different kernel functions. So, I choose the kernel pca using the vanilladot kernel function.

```
#screeplot for the variance of the kernel pca using vanilladot.
var.pc <- eig(kernel_pca2)
prop.pc <- var.pc/sum(var.pc)
plot(prop.pc, xlab = "Principal Component", col = "red",
    ylab = "Proportion of Variance Explained", type = "b", pch = 19)
```




```
# Plot THE DATA PROJECTION ON THE KERNEL PCS
PC <- rotated(kernel_pca2)     # returns the data projected in the (kernel) pca space
plot(PC[, 1:2],col=train$digit, pch="",
     main="KPC.1 and KPC.2 for the train handwritten digit data",
     xlab="1st Kernel PC", ylab="2nd Kernel PC")
text(PC[, 1:2], labels=train$digit, col= train$digit)
abline(v=0, lty=2)
abline(h=0, lty=2)
```

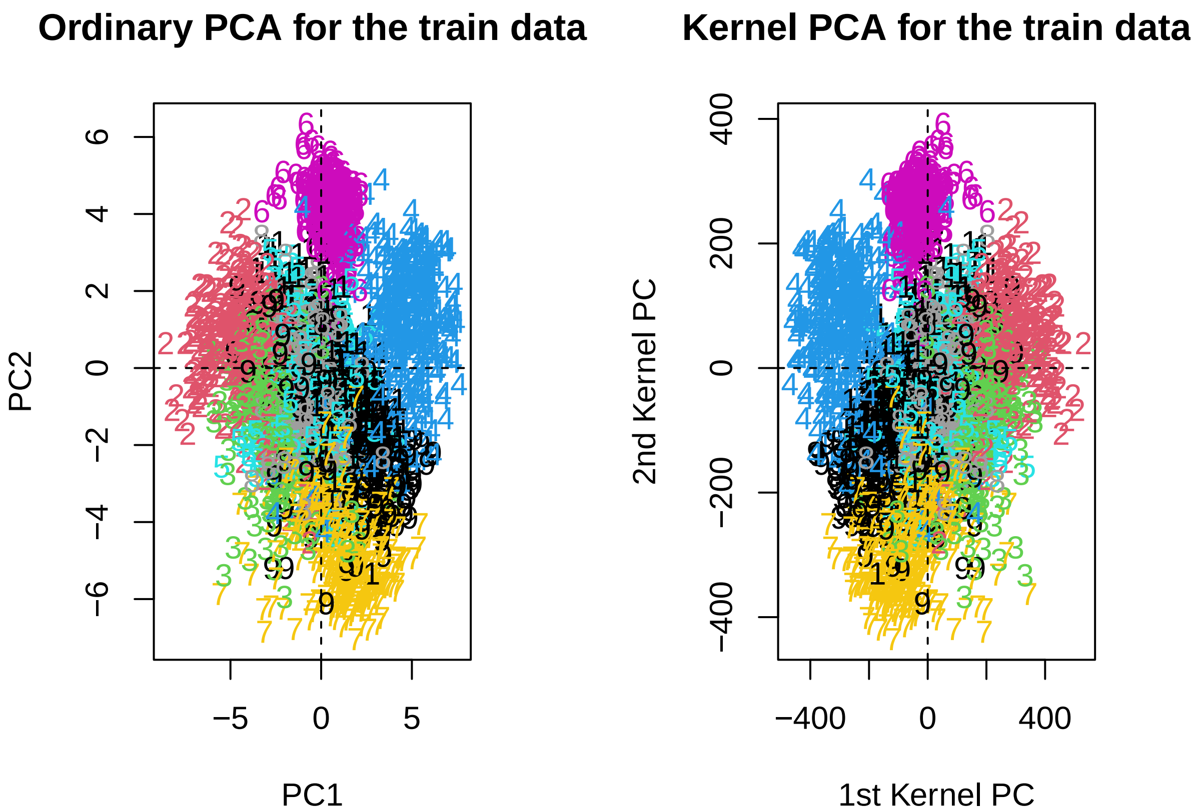


Remarks

- We can see from the above plots that the first two PCs fairly successfully separate the digits. We see, for instance, that most 6s lie on the top of the plot, most 4s on the top left, most 7s on the bottom and most 2s lie on the middle right.There are however regions of overlap.
- The choice of kernel function used is vanilladot(linear kernel function).
- The parameter is degree = 1.
- comparison of PCA and KPCA

```
par(mfrow=c(1,2), mar=rep(4,4))
plot(pca$x[,1:2], pch="", main="Ordinary PCA for the train data")
text(pca$x[,1:2], labels=train$digit, col= train$digit)
abline(v=0, lty=2)
abline(h=0, lty=2)



plot(PC[, 1:2],col=train$digit, pch="",
     main="Kernel PCA for the train data",
```

```
     xlab="1st Kernel PC", ylab="2nd Kernel PC")
text(PC[, 1:2], labels=train$digit, col= train$digit)
abline(v=0, lty=2)
abline(h=0, lty=2)
```



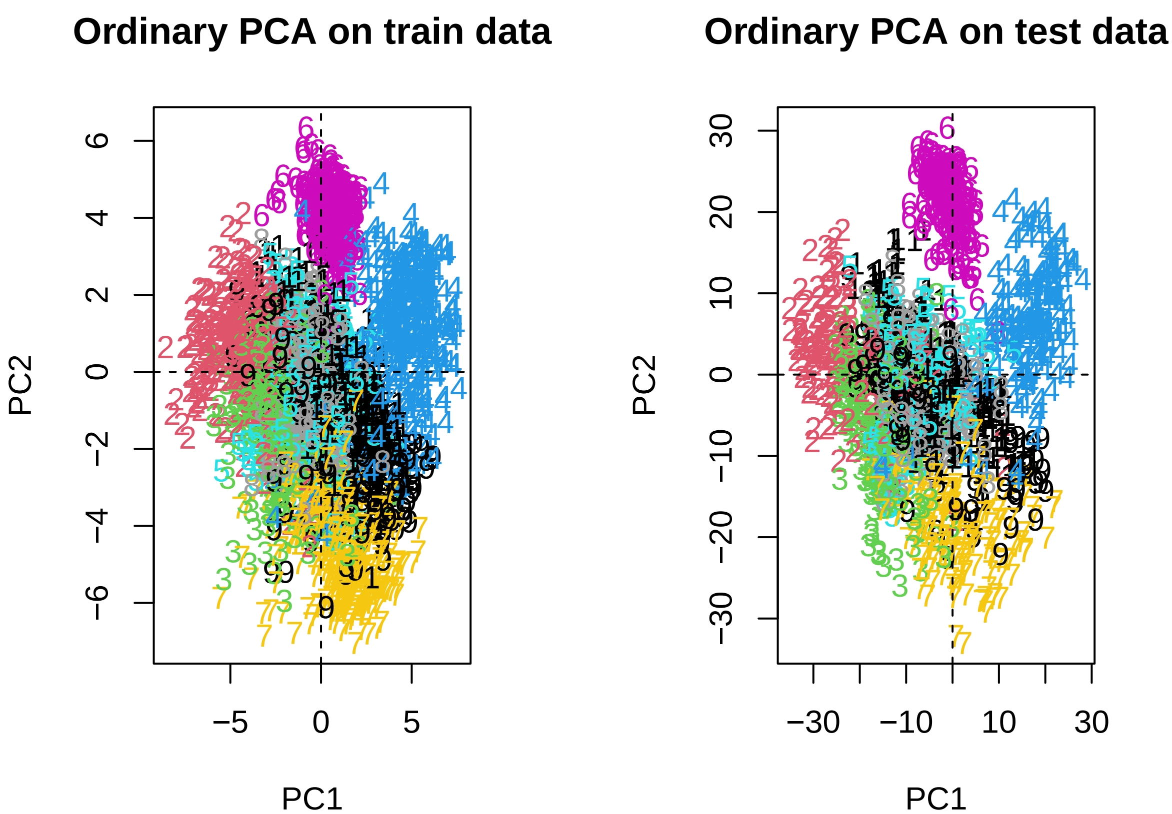
Remarks

We see from the above plots that there is no significant difference between clustering of the digits. Both methods show that the first two PCs explain a substantial portion of the variation in the data.

4 PCA and KPCA on the test data

```
#ordinary pca
pred_pca <- predict(pca, test_scaled)

# comparison of the PCA  results on the train and test data

par(mfrow=c(1,2), mar=rep(4,4))
```

```
plot(pca$x[,1:2], pch="", main="Ordinary PCA on train data")
text(pca$x[,1:2], labels=train$digit, col= train$digit)
abline(v=0, lty=2)
abline(h=0, lty=2)


plot(pred_pca[,1:2], pch="", main="Ordinary PCA on test data")
text(pred_pca[,1:2], labels=test$digit, col= test$digit)
abline(v=0, lty=2)
abline(h=0, lty=2)
```

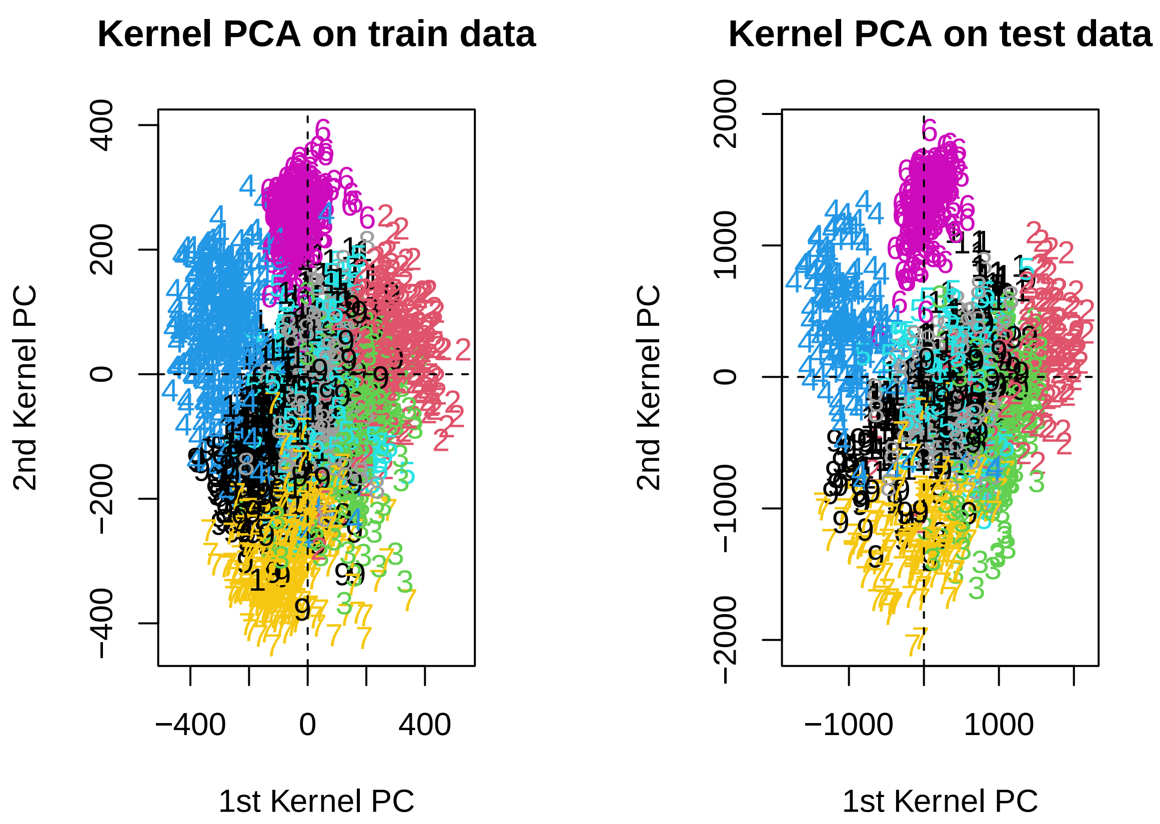


Remarks

There is no significant difference between ordinary pca on both the train and test data.

```
#kernel pca
pred_kernel_pca <- predict(kernel_pca2, test_scaled)

par(mfrow=c(1,2), mar=rep(4,4))
plot(PC[, 1:2],col=train$digit, pch="",
     main="Kernel PCA on train data",
```

```
     xlab="1st Kernel PC", ylab="2nd Kernel PC")
text(PC[, 1:2], labels=train$digit, col= train$digit)
abline(v=0, lty=2)
abline(h=0, lty=2)


plot(pred_kernel_pca[, 1:2],col=test$digit, pch="",
     main="Kernel PCA on test data",
     xlab="1st Kernel PC", ylab="2nd Kernel PC")
text(pred_kernel_pca[, 1:2], labels=test$digit, col= test$digit)
abline(v=0, lty=2)
abline(h=0, lty=2)
```

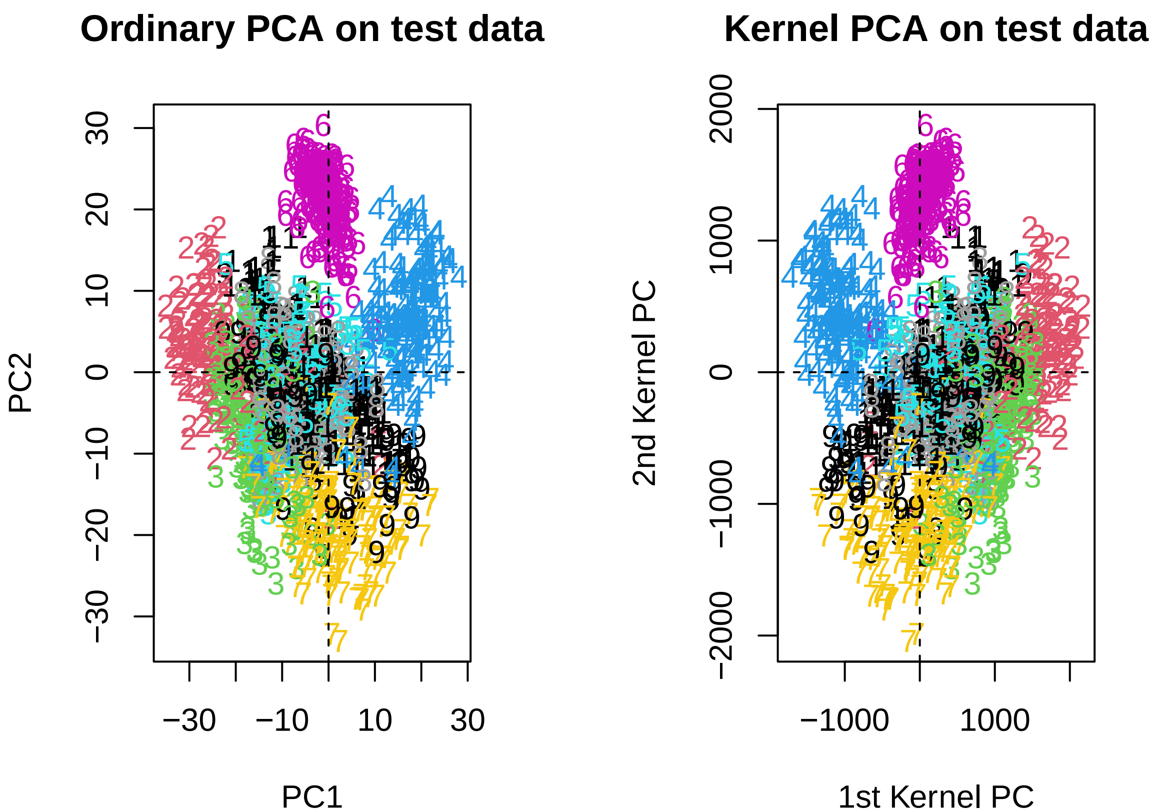


Remarks

There is no significant difference between the result of the Kernel pca on the train data and the kernel pca on test data.

- comparison of ordinary pca and kernel pca on the test data

```
par(mfrow=c(1,2), mar=rep(4,4))

plot(pred_pca[,1:2], pch="", main="Ordinary PCA on test data")
text(pred_pca[,1:2], labels=test$digit, col= test$digit)
abline(v=0, lty=2)
abline(h=0, lty=2)

plot(pred_kernel_pca[, 1:2],col=test$digit, pch="",
     main="Kernel PCA on test data",
     xlab="1st Kernel PC", ylab="2nd Kernel PC")
text(pred_kernel_pca[, 1:2], labels=test$digit, col= test$digit)
abline(v=0, lty=2)
abline(h=0, lty=2)
```



Remarks

- There is no signifcaant difference betweeen thee results of the ordinary pca and the kernel pca on the test data.
- We observe that most 2s lie on the middle left on the ordinary pca while they lie on the middle right on the kernel pca.

- In both cases they fairly separate the digits well.

5 ASSOCIATION RULES

5.1 Read in Data

```
library(arules)
```

```
## Loading required package: Matrix
```

```
##
## Attaching package: 'arules'
```

```
## The following object is masked from 'package:kernlab':
##
##     size
```

```
## The following objects are masked from 'package:base':
##
##     abbreviate, write
```

```
bible <- read.transactions(file="AV1611Bible.txt",
format = "basket", sep =" ", rm.duplicates =F,
quote="") # DOUBLE/SINGLE QUOTE ISSUE
dat <- bible; dim(dat)
```

```
## [1] 31101 12767
```

```
inspect(dat[1:5, ])
```

```
##     items
## [1] {beginning,
##      created,
##      earth,
##      god,
##      heaven}
## [2] {darkness,
##      deep,
##      earth,
```

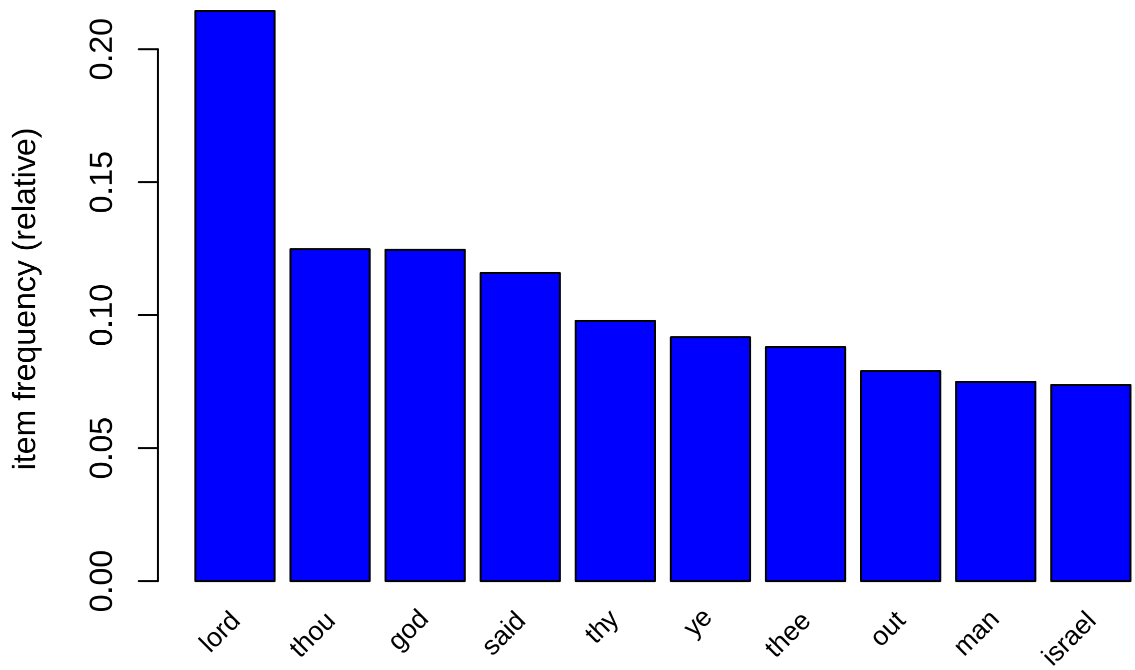
```
##      face,
##      form,
##      god,
##      moved,
##      spirit,
##      upon,
##      void,
##      waters,
##      without}
## [3] {god,
##      let,
##      light,
##      said,
##      there}
## [4] {darkness,
##      divided,
##      god,
##      good,
##      light,
##      saw}
## [5] {called,
##      darkness,
##      day,
##      evening,
##      first,
##      god,
##      light,
##      morning,
##      night}
```

5.2 Perform frequent itemsets and association rule analysis.

```
# The first 15 items (frequency/support)
itemFrequency(dat[, 1:15])
```

```
##        aaron      aaron's    aaronites      abaddon      abagtha        abana
## 9.742452e-03 9.967525e-04 6.430661e-05 3.215331e-05 3.215331e-05 3.215331e-05
##       abarim        abase       abased      abasing       abated         abba
## 1.286132e-04 1.286132e-04 1.286132e-04 3.215331e-05 1.929198e-04 9.645992e-05
##         abda       abdeel         abdi
## 6.430661e-05 3.215331e-05 9.645992e-05
```

```
# Plot items with high frequencies.
itemFrequencyPlot(dat, topN=10, support = 0.01, cex.names = 0.8, col="blue")
```



Remarks

We observe that lord has the highest frequency.

```
summary(dat)
```

```
## transactions as itemMatrix in sparse format with
##  31101 rows (elements/itemsets/transactions) and
##  12767 columns (items) and a density of 0.0009590938
##
## most frequent items:
##    lord    thou     god    said     thy (Other)
##    6667    3881    3875    3602    3044  359755
##
## element (itemset/transaction) length distribution:
## sizes
##    2    3    4    5    6    7    8    9   10   11   12   13   14   15   16   17
##   13  235  550  840 1554 2258 2536 2611 2428 2465 2283 2139 1925 1751 1490 1248
##   18   19   20   21   22   23   24   25   26   27   28   29   30   31   32   33
## 1084  876  666  574  412  321  258  182  129  107   57   47   22   14    2    8
##   34   35   36   37
```

```
##    5    4    5    2
##
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    2.00    8.00   12.00   12.24   15.00   37.00
##
## includes extended item information - examples:
##      labels
## 1     aaron
## 2   aaron's
## 3 aaronites
```

Remarks

- Itemset/transaction with size 9 has the highest frequency of 2611
- Itemset/transaction with the highest size 37 and size 32 have the lowest frequency of 2.

```r
#Association Rule Analysis
rules <- apriori(dat, parameter = list(support = 0.01, confidence = 0.5,
    target = "rules", maxlen=5))
```

```
## Apriori
##
## Parameter specification:
##  confidence minval smax arem  aval originalSupport maxtime support minlen
##         0.5    0.1    1 none FALSE            TRUE       5    0.01      1
##  maxlen target  ext
##       5  rules TRUE
##
## Algorithmic control:
##  filter tree heap memopt load sort verbose
##     0.1 TRUE TRUE  FALSE TRUE    2    TRUE
##
## Absolute minimum support count: 311
##
## set item appearances ...[0 item(s)] done [0.00s].
## set transactions ...[12767 item(s), 31101 transaction(s)] done [0.17s].
## sorting and recoding items ... [230 item(s)] done [0.01s].
## creating transaction tree ... done [0.02s].
## checking subsets of size 1 2 3 4 done [0.01s].
## writing ... [29 rule(s)] done [0.00s].
## creating S4 object  ... done [0.02s].
```

```r
inspect(rules[1:5])
```

```
##     lhs           rhs    support    confidence coverage   lift      count
## [1] {answered} => {said} 0.01067490 0.6775510  0.01575512  5.850226 332
## [2] {art}      => {thou} 0.01434037 0.9867257  0.01453329  7.907280 446
## [3] {word}     => {lord} 0.01189672 0.5497771  0.02163918  2.564664 370
## [4] {moses}    => {lord} 0.01485483 0.5976714  0.02485451  2.788087 462
## [5] {she}      => {her}  0.01408315 0.6016484  0.02340761 15.684715 438
```

```r
Rules <- as(rules, "data.frame")
head(Rules); tail(Rules)
```

```
##                   rules    support confidence   coverage      lift count
## 1 {answered} => {said} 0.01067490  0.6775510 0.01575512  5.850226   332
## 2      {art} => {thou} 0.01434037  0.9867257 0.01453329  7.907280   446
## 3     {word} => {lord} 0.01189672  0.5497771 0.02163918  2.564664   370
## 4    {moses} => {lord} 0.01485483  0.5976714 0.02485451  2.788087   462
## 5       {she} => {her} 0.01408315  0.6016484 0.02340761 15.684715   438
## 6    {thus} => {saith} 0.01462975  0.6435644 0.02273239 16.721383   455
```

```
##                    rules    support confidence   coverage     lift count
## 24 {god,thee} => {lord} 0.01054628  0.5996344 0.01758786 2.797244   328
## 25  {god,thy} => {thou} 0.01135012  0.5912898 0.01919552 4.738393   353
## 26  {god,thou} => {thy} 0.01135012  0.5064562 0.02241085 5.174539   353
## 27  {god,thy} => {lord} 0.01340793  0.6984925 0.01919552 3.258409   417
## 28 {lord,thy} => {thou} 0.01530497  0.5157096 0.02967750 4.132720   476
## 29 {god,thou} => {lord} 0.01305424  0.5824964 0.02241085 2.717297   406
```

```r
dim(Rules)
```

```
## [1] 29  6
```

```r
inspect(rules[1:10], ruleSep = "---->", itemSep = " + ", setStart = "",
        setEnd ="",linebreak = FALSE)
```

```
##      lhs            rhs    support    confidence coverage   lift      count
## [1]  answered ----> said   0.01067490 0.6775510  0.01575512  5.850226 332
## [2]  art      ----> thou   0.01434037 0.9867257  0.01453329  7.907280 446
## [3]  word     ----> lord   0.01189672 0.5497771  0.02163918  2.564664 370
## [4]  moses    ----> lord   0.01485483 0.5976714  0.02485451  2.788087 462
## [5]  she      ----> her    0.01408315 0.6016484  0.02340761 15.684715 438
```

```
## [6]  thus    ----> saith 0.01462975 0.6435644  0.02273239 16.721383 455
## [7]  thus    ----> lord  0.01626957 0.7157001  0.02273239  3.338682 506
## [8]  pass    ----> came  0.01488698 0.5758706  0.02585126  9.337932 463
## [9]  thine   ----> thy   0.01318286 0.5012225  0.02630141  5.121065 410
## [10] thine   ----> thou  0.01395454 0.5305623  0.02630141  4.251744 434
```

```
quality(rules[1:15])
```

```
##       support confidence   coverage      lift count
## 1  0.01067490  0.6775510 0.01575512  5.850226   332
## 2  0.01434037  0.9867257 0.01453329  7.907280   446
## 3  0.01189672  0.5497771 0.02163918  2.564664   370
## 4  0.01485483  0.5976714 0.02485451  2.788087   462
## 5  0.01408315  0.6016484 0.02340761 15.684715   438
## 6  0.01462975  0.6435644 0.02273239 16.721383   455
## 7  0.01626957  0.7157001 0.02273239  3.338682   506
## 8  0.01488698  0.5758706 0.02585126  9.337932   463
## 9  0.01318286  0.5012225 0.02630141  5.121065   410
## 10 0.01395454  0.5305623 0.02630141  4.251744   434
## 11 0.02684801  0.9835100 0.02729816  7.881511   835
## 12 0.01691264  0.5394872 0.03134947  2.516663   526
## 13 0.02819845  0.7326650 0.03848751  3.417821   877
## 14 0.03900196  0.9991763 0.03903411  8.007055  1213
## 15 0.02286100  0.5511628 0.04147777  6.012527   711
```

```
summary(rules)
```

```
## set of 29 rules
##
## rule length distribution (lhs + rhs):sizes
##  2  3
## 15 14
##
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   2.000   2.000   2.000   2.483   3.000   3.000
##
## summary of quality measures:
##     support          confidence        coverage            lift
##  Min.   :0.01055   Min.   :0.5012   Min.   :0.01132   Min.   : 2.517
##  1st Qu.:0.01190   1st Qu.:0.5759   1st Qu.:0.01550   1st Qu.: 3.339
##  Median :0.01389   Median :0.6776   Median :0.02241   Median : 5.121
##  Mean   :0.01547   Mean   :0.7261   Mean   :0.02215   Mean   : 6.556
##  3rd Qu.:0.01514   3rd Qu.:0.9495   3rd Qu.:0.02630   3rd Qu.: 7.993
```

```
##  Max.   :0.03900   Max.   :1.0000   Max.   :0.04148   Max.   :22.183
##      count
##  Min.   : 328
##  1st Qu.: 370
##  Median : 432
##  Mean   : 481
##  3rd Qu.: 471
##  Max.   :1213
##
## mining info:
##  data ntransactions support confidence
##   dat         31101    0.01        0.5
##
##  apriori(data = dat, parameter = list(support = 0.01, confidence = 0.5, target = "rul
```

Remarks

- The parameters used for the R function arules are : support = 0.01,confidence = 0.5,target = "rules",maxlen=5.
- The maximum support is 0.03900 and the minimum support is 0.01055.
- The maximum confidence is 1.0000 and the minimum confidence is 0.5012.
- The maximum lift is 22.183 and the minimum lift is 2.517.

5.3 Top 5 rules in decreasing order of confidence (conf) for item sets of size/length 2 or 3.

```r
rules0 <- data.frame(matrix(unlist(strsplit(as.character(Rules$rules), split="=>")),
    ncol=2, byrow=TRUE))
colnames(rules0) <- c("LHS", "RHS")
rule.size <- function(x){length(unlist(strsplit(as.character(x), split=",")))}
rules0$size <- apply(rules0, 1, rule.size)
```

```r
z <- data.frame(Rules, size=rules0$size)
top.support <- z[order(z$confidence, decreasing = T),]
head(top.support, 5)
```

```
##                          rules    support confidence   coverage     lift count size
## 20 {shalt,thee} => {thou} 0.01270056  1.0000000 0.01270056 8.013656   395    3
```

```
## 21  {shalt,thy} => {thou} 0.01514421  1.0000000 0.01514421 8.013656   471    3
## 14      {shalt} => {thou} 0.03900196  0.9991763 0.03903411 8.007055  1213    2
## 22 {lord,shalt} => {thou} 0.01225041  0.9973822 0.01228256 7.992678   381    3
## 2         {art} => {thou} 0.01434037  0.9867257 0.01453329 7.907280   446    2
```

Remarks

We observed that the rule (shalt) => (thou) is a creditable rule since it has a large level of support(0.03900196),large confidence(0.9991763) factor and a value of lift(8.007055) greater than 1. Thus, we expect to see **shalt** followed by **thou** in the King James Bible1. In other words, shalt and thou are words that commonly occur together in sentences.

5.4 Top 5 rules in decreasing order of the lift measure for item sets of size 2 or 3.

```r
z <- data.frame(Rules, size=rules0$size)
top5.lift <- z[order(z$lift, decreasing = T),]
head(top5.lift, 5)
```

```
##                      rules    support confidence   coverage      lift count size
## 17 {lord,thus} => {saith} 0.01389023  0.8537549 0.01626957 22.182650   432    3
## 6       {thus} => {saith} 0.01462975  0.6435644 0.02273239 16.721383   455    2
## 5          {she} => {her} 0.01408315  0.6016484 0.02340761 15.684715   438    2
## 8        {pass} => {came} 0.01488698  0.5758706 0.02585126  9.337932   463    2
## 20 {shalt,thee} => {thou} 0.01270056  1.0000000 0.01270056  8.013656   395    3
```

Remarks

We observed that the rule (thus) => (saith) is a fairly a creditable rule since it has a fairly large level of support(0.01462975),fairly large confidence(0.6435644) factor and a value of lift(16.721383) greater than 1. Hence, thus and saith are words that commonly occur together in sentences in the King James Bible.

5.5 Conviction measures for the top-lift 5 rules in Part (d)

```r
top5_liftrules <- sort(rules, decreasing = T, by='lift')[1:5,] # top lift 5 rules
interestMeasure(top5_liftrules, "conviction", transactions = dat)
```

```
## [1] 6.574666 2.697577 2.414051 2.212367      Inf
```

Remarks

- The problem associated with both the confidence and the lift measures is that they are not sensitive to rule direction. On the other hand, conviction is sensitive to rule direction. It attempts to measure the degree of implication of a rule. That is unlike lift, conviction($A \Rightarrow B$) $\neq$ conviction($B \Rightarrow A$).
- The conviction((shalt,thee) => (thou)) $= \infty$. This is because the confidence obtained for the rule (shalt,thee) => (thou) in part b is 1.