# Project-II

*Md Al Masum Bhuiyan*

*October 08, 2018*

## Contents

**1 Data collection** **2**
1.1 Examine the data briefly . . . . . 2
1.1.1 Detecting missing values . . . . . 2
1.1.2 Detecting NA values . . . . . 4
1.1.3 Qualitative analysis . . . . . 4
1.1.4 inspecting the distinct values . . . . . 6
1.2 Detect unary column . . . . . 6

**2 Problme 1(b)** **8**
2.1 Principal Component Analysis (PCA) . . . . . 8

**3 Problme 1(c)** **13**
3.1 Kernel PCA . . . . . 13
3.2 Comparison . . . . . 16
3.3 Problem 1(d) . . . . . 17
3.3.1 Test data collection . . . . . 17
3.3.2 Detecting missing values for test data . . . . . 17

**4 Problem 2** **22**
4.1 Problem 2(a) . . . . . 22
4.2 Problem 2(b) . . . . . 23
4.3 Problem 2(c) . . . . . 26
4.4 Problem 2(d) . . . . . 27
4.5 Problem 2(e) . . . . . 27

# 1 Data collection

We bring the data from UCI Machine Learning Repository.

```
train <- read.table(file="http://archive.ics.uci.edu/ml/machine-learning-databases/optdigits/op
                    col.names = c(paste("x", 1:64, sep=""), "digit"))
dim(train)
```

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

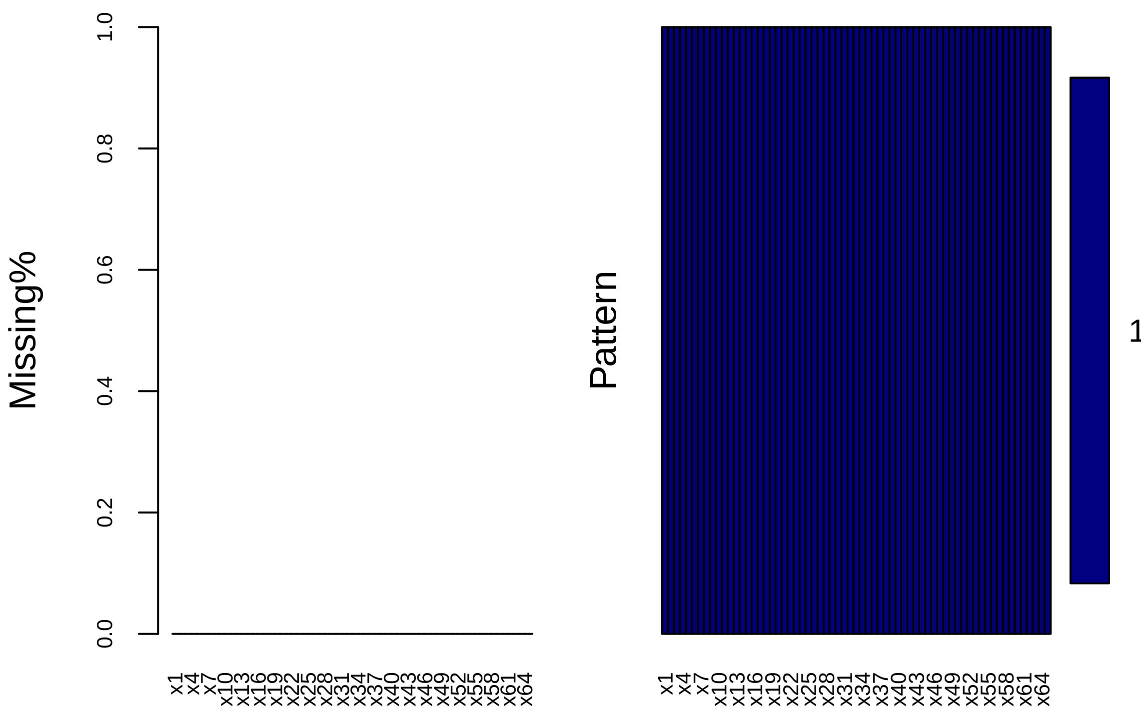
We see that we have 64 measurements for a total of 3823 handwritten digits. The 65th "measurement" is not actually a measurement, but rather the actual (true) digit given the values from the 64 other measurements.

## 1.1 Examine the data briefly

We next explore the data by detecting the missing values as follows:

### 1.1.1 Detecting missing values

```
#install.packages("VIM")
library(VIM)
aggr(train, col=c('navyblue','yellow'), numbers=TRUE, sortVars=TRUE,
    labels=names(train), cex.axis=.7, gap=3, ylab=c("Missing%","Pattern"))
```




```
##
##  Variables sorted by number of missings:
```

```
##   Variable Count
##         x1     0
##         x2     0
##         x3     0
##         x4     0
##         x5     0
##         x6     0
##         x7     0
##         x8     0
##         x9     0
##        x10     0
##        x11     0
##        x12     0
##        x13     0
##        x14     0
##        x15     0
##        x16     0
##        x17     0
##        x18     0
##        x19     0
##        x20     0
##        x21     0
##        x22     0
##        x23     0
##        x24     0
##        x25     0
##        x26     0
##        x27     0
##        x28     0
##        x29     0
##        x30     0
##        x31     0
##        x32     0
##        x33     0
##        x34     0
##        x35     0
##        x36     0
##        x37     0
##        x38     0
##        x39     0
##        x40     0
##        x41     0
##        x42     0
##        x43     0
##        x44     0
##        x45     0
##        x46     0
##        x47     0
```

```
##        x48      0
##        x49      0
##        x50      0
##        x51      0
##        x52      0
##        x53      0
##        x54      0
##        x55      0
##        x56      0
##        x57      0
##        x58      0
##        x59      0
##        x60      0
##        x61      0
##        x62      0
##        x63      0
##        x64      0
##      digit      0
```

There is no missing values in the trained data.

### 1.1.2 Detecting NA values

```r
any(is.na(train))   #Checking for NAs
```

```
## [1] FALSE
```

There is no NA value in the trained data.

### 1.1.3 Qualitative analysis

Now we perform the qualitative anaysis to examine the data briefly.

```r
# ======================  quntitative analysis=============
library("funModeling")
df_status(train)
```

```
##    variable q_zeros p_zeros q_na p_na q_inf p_inf    type unique
## 1        x1    3823  100.00    0    0     0     0 integer      1
## 2        x2    3230   84.49    0    0     0     0 integer      9
## 3        x3     836   21.87    0    0     0     0 integer     17
## 4        x4     112    2.93    0    0     0     0 integer     17
## 5        x5     146    3.82    0    0     0     0 integer     17
## 6        x6    1118   29.24    0    0     0     0 integer     17
## 7        x7    2908   76.07    0    0     0     0 integer     17
## 8        x8    3709   97.02    0    0     0     0 integer     16
## 9        x9    3820   99.92    0    0     0     0 integer      4
## 10      x10    2169   56.74    0    0     0     0 integer     16
```

```
## 11      x11   347    9.08   0   0    0    0 integer     17
## 12      x12    41    1.07   0   0    0    0 integer     17
## 13      x13   155    4.05   0   0    0    0 integer     17
## 14      x14   734   19.20   0   0    0    0 integer     17
## 15      x15  2464   64.45   0   0    0    0 integer     17
## 16      x16  3681   96.29   0   0    0    0 integer     15
## 17      x17  3814   99.76   0   0    0    0 integer      5
## 18      x18  1858   48.60   0   0    0    0 integer     17
## 19      x19   567   14.83   0   0    0    0 integer     17
## 20      x20   872   22.81   0   0    0    0 integer     17
## 21      x21   922   24.12   0   0    0    0 integer     17
## 22      x22  1016   26.58   0   0    0    0 integer     17
## 23      x23  2360   61.73   0   0    0    0 integer     17
## 24      x24  3756   98.25   0   0    0    0 integer      9
## 25      x25  3819   99.90   0   0    0    0 integer      2
## 26      x26  1913   50.04   0   0    0    0 integer     17
## 27      x27   657   17.19   0   0    0    0 integer     17
## 28      x28   559   14.62   0   0    0    0 integer     17
## 29      x29   731   19.12   0   0    0    0 integer     17
## 30      x30   802   20.98   0   0    0    0 integer     17
## 31      x31  2272   59.43   0   0    0    0 integer     17
## 32      x32  3813   99.74   0   0    0    0 integer      3
## 33      x33  3818   99.87   0   0    0    0 integer      2
## 34      x34  2289   59.87   0   0    0    0 integer     16
## 35      x35   967   25.29   0   0    0    0 integer     17
## 36      x36   704   18.41   0   0    0    0 integer     17
## 37      x37   547   14.31   0   0    0    0 integer     17
## 38      x38   605   15.83   0   0    0    0 integer     17
## 39      x39  1781   46.59   0   0    0    0 integer     15
## 40      x40  3823  100.00   0   0    0    0 integer      1
## 41      x41  3784   98.98   0   0    0    0 integer      8
## 42      x42  2637   68.98   0   0    0    0 integer     17
## 43      x43  1445   37.80   0   0    0    0 integer     17
## 44      x44  1225   32.04   0   0    0    0 integer     17
## 45      x45   894   23.38   0   0    0    0 integer     17
## 46      x46   669   17.50   0   0    0    0 integer     17
## 47      x47  1772   46.35   0   0    0    0 integer     17
## 48      x48  3775   98.74   0   0    0    0 integer      5
## 49      x49  3791   99.16   0   0    0    0 integer      8
## 50      x50  2897   75.78   0   0    0    0 integer     17
## 51      x51   739   19.33   0   0    0    0 integer     17
## 52      x52   211    5.52   0   0    0    0 integer     17
## 53      x53   242    6.33   0   0    0    0 integer     17
## 54      x54   690   18.05   0   0    0    0 integer     17
## 55      x55  1834   47.97   0   0    0    0 integer     17
## 56      x56  3619   94.66   0   0    0    0 integer     11
## 57      x57  3822   99.97   0   0    0    0 integer      2
## 58      x58  3344   87.47   0   0    0    0 integer     11
```

```
## 59      x59    861   22.52   0    0    0    0 integer    17
## 60      x60    156    4.08   0    0    0    0 integer    17
## 61      x61    280    7.32   0    0    0    0 integer    17
## 62      x62   1007   26.34   0    0    0    0 integer    17
## 63      x63   2523   66.00   0    0    0    0 integer    17
## 64      x64   3616   94.59   0    0    0    0 integer    16
## 65    digit    376    9.84   0    0    0    0 integer    10
```

### 1.1.4 inspecting the distinct values

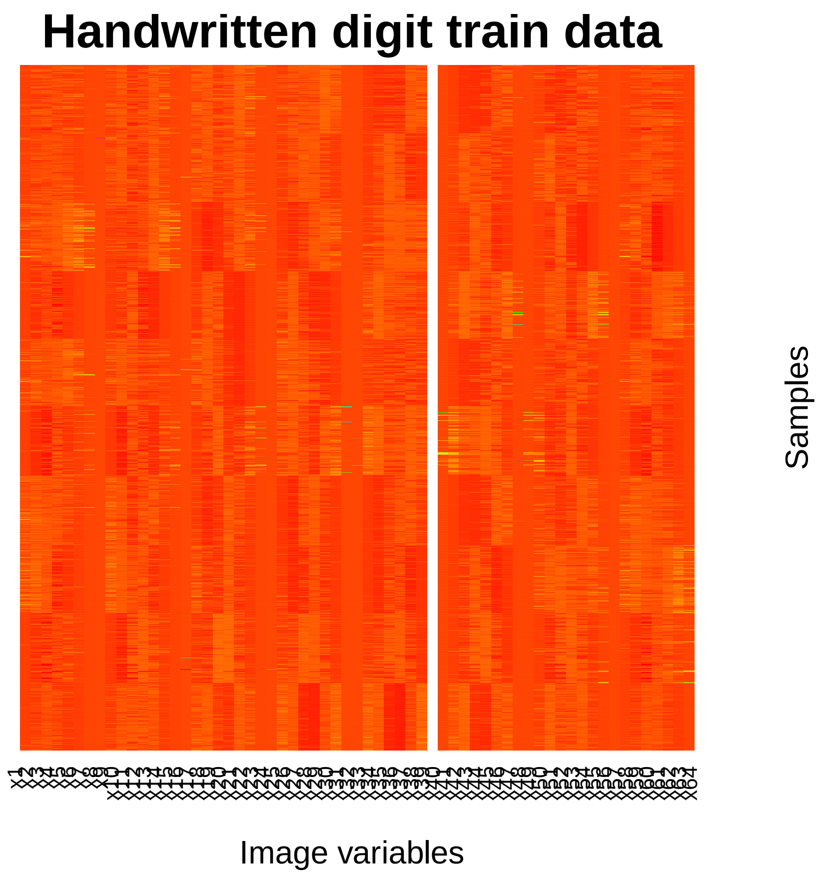
We ommitted the results due to large size of outputs.

```
#===================== inspecting the distinct values=======
for (j in 1:NCOL(train)){
  print(colnames(train)[j])
  print(table(train[,j], useNA="ifany"))
}
```

## 1.2 Detect unary column

I next order train data based on train$digit (the true digit). I place the true digits into a vector to be used later for labeling, remove the known digit in the 65th column. We remove the measurements that have uniary column, as they contribute no additional information and thus their inclusion is not necessary for analysis.

```
dat0 <- data.matrix(train[order(train$digit),])
labs <- dat0[, c(65)] # Labels for plotting needed later
dat0 <- dat0[,-c(65)] # Remove the known digits
n <- NROW(dat0)
color <- rainbow(n, alpha=0.8)
heatmap(dat0, col=color, scale="column", Rowv=NA, Colv=NA,
        labRow=FALSE, margins=c(4,4), xlab="Image variables", ylab="Samples", main="Handwritten
```



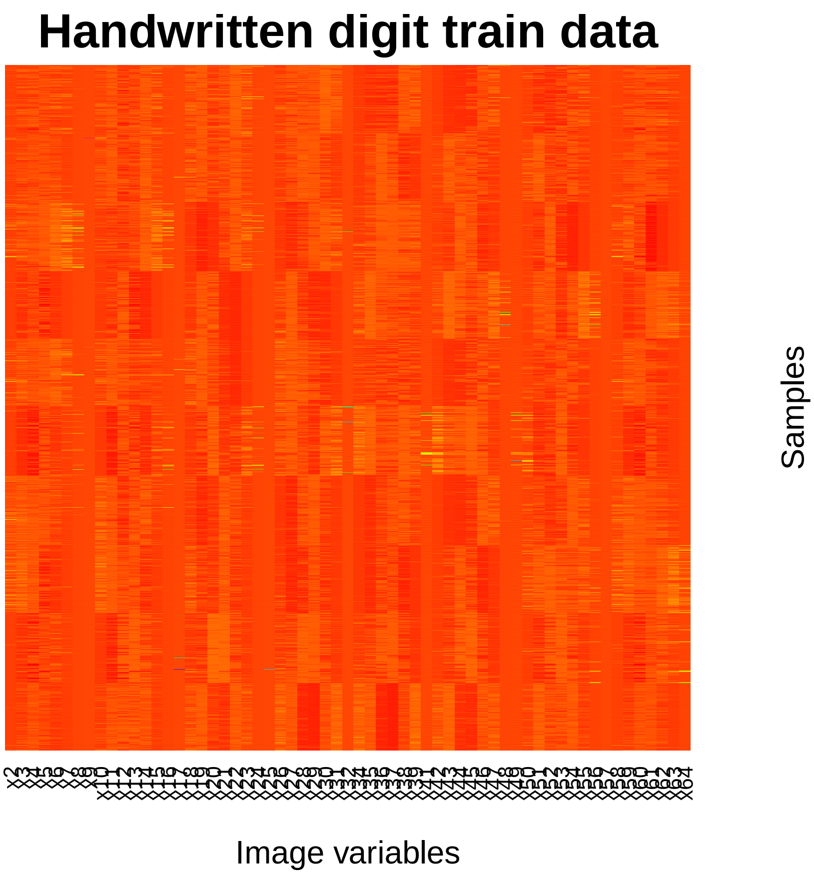

Before computing the number of unary column, we first see the heat map of train data. From the heatmap, it is clear that there are no observations recorded for the 1st and 40th variable of train data.

```
uniq <- apply(dat0, 2, unique)
  for (k in 1:length(uniq)){
    if (length(unique(dat0[,k])) == 1)
      print(paste("x", k))
  }
```

```
## [1] "x 1"
## [1] "x 40"
```

So the 1st and 40 th columns are unary in train data. I remove the unary colums and I have measurements for each of the variables for which the value recorded in the region observed is not constant. I also remove the 33th column, which has been discussed in the testing data section. I then plot these values with heatmap() again to identify any potential patterns.

```
dat0 <- dat0[, -c(1, 33, 40)]
n <- NROW(dat0)
color <- rainbow(n, alpha=0.8)
heatmap(dat0, col=color, scale="column", Rowv=NA, Colv=NA,
        labRow=FALSE, margins=c(4,4), xlab="Image variables", ylab="Samples", main="Handwritte
```




While not initially visible, there seems to be approximately ten distinct regions when looking at the plot in heatmap figure. This makes sense as the values were sorted prior to plotting, and we are looking at the handwritten digits of $0, 1, \cdots 9$.

## 2 Problme 1(b)

I excluded the target variable in the previous subsection. Now I begin to perform PCA with the data. I first scale the data, and then compute the Principal Components (PCs) for the train data.

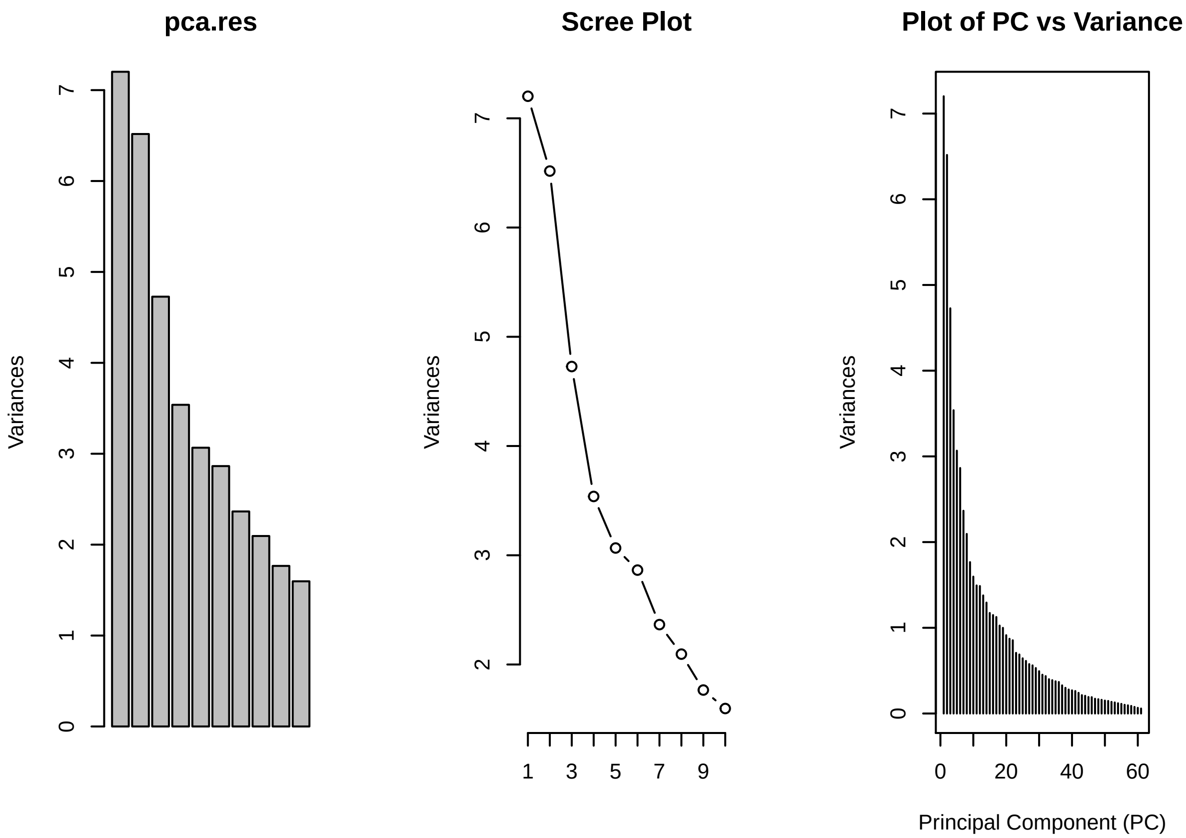
```
library(RColorBrewer)
palette(brewer.pal(n=length(unique(labs)), name="Set3"))
dat0.scaled <- data.frame(apply(dat0, 2, scale))
```

### 2.1 Principal Component Analysis (PCA)

```
pca.res <- prcomp(dat0.scaled, retx=TRUE)
sd.pc <- pca.res$sdev
var.pc <- sd.pc**2
```

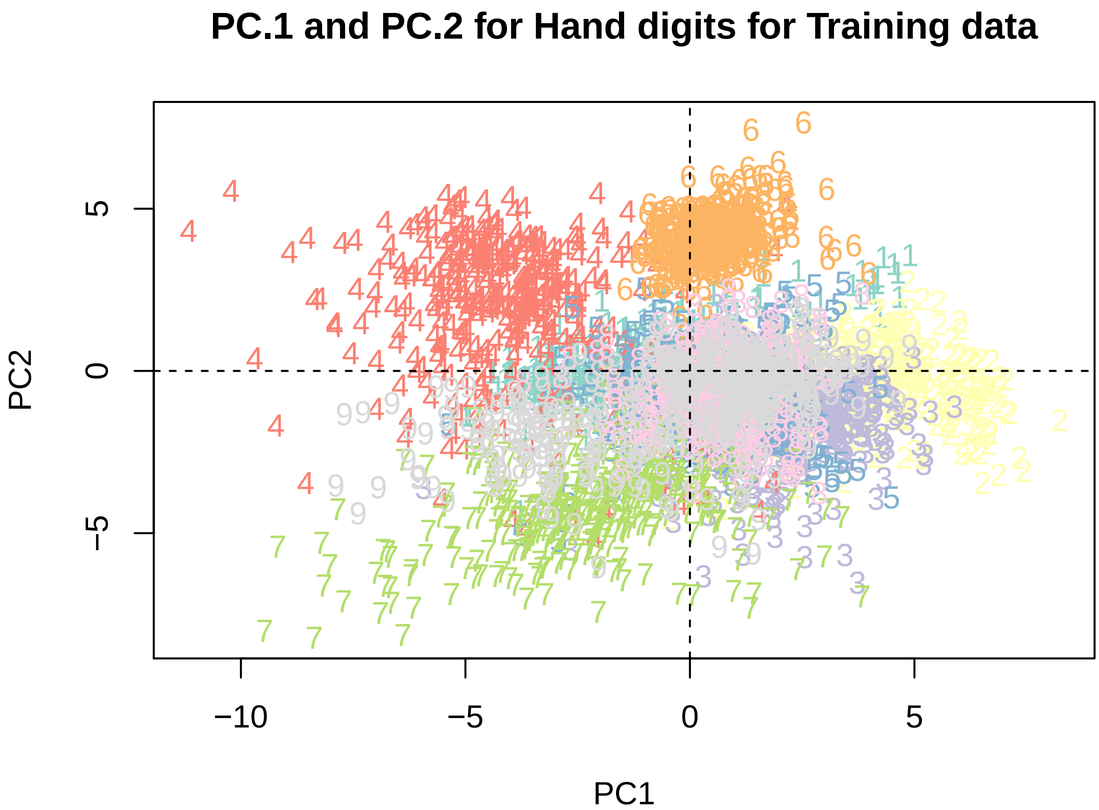
Next, I construct a scree plot showing the cumulative proportions of variation for all of the PCs and I choose two eigenvalues.

```r
par(mfrow=c(1,3), mar=rep(4,4))
plot(pca.res)
screeplot(pca.res, type="lines", main="Scree Plot")
plot(var.pc, type="h", xlab="Principal Component (PC)",
     ylab="Variances", main="Plot of PC vs Variance")
```




Now I plot PC2 versus PC1, where the ‘dots’ for each digit are represented with different colors and digit symbols of training data.

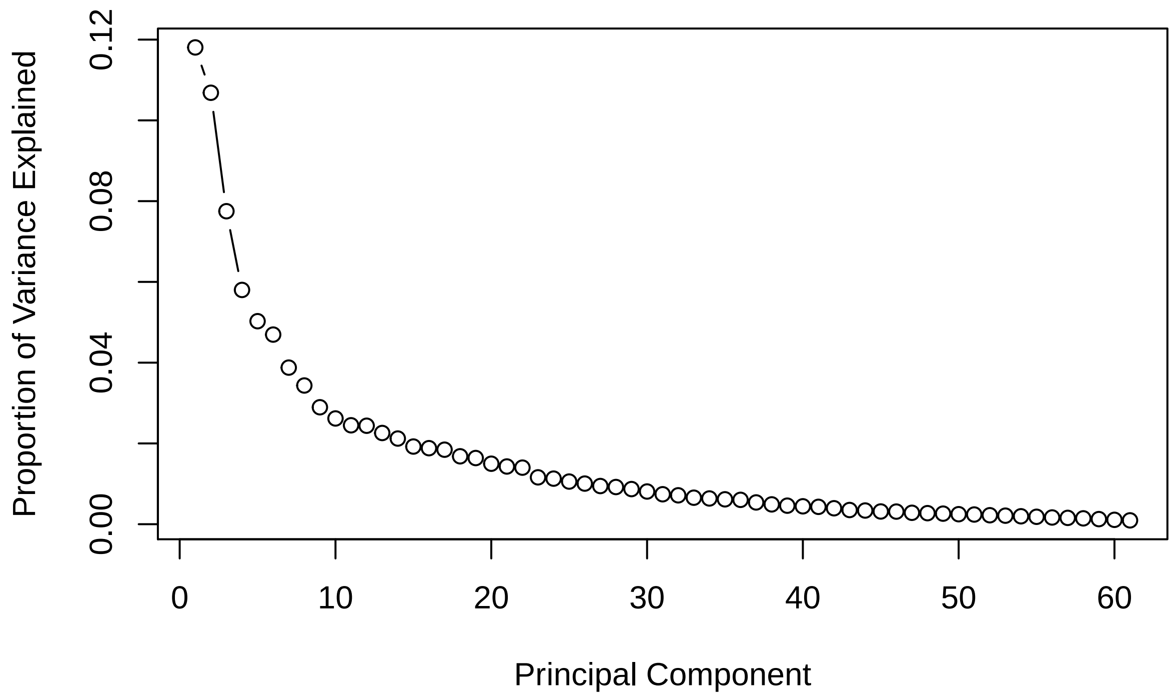
```r
par(mfrow=c(1,1), mar=rep(4,4))
plot(pca.res$x[,1:2], pch="", main="PC.1 and PC.2 for Hand digits for Training data")
text(pca.res$x[,1:2], labels=labs, col=labs)
abline(v=0, lty=2)
abline(h=0, lty=2)
```



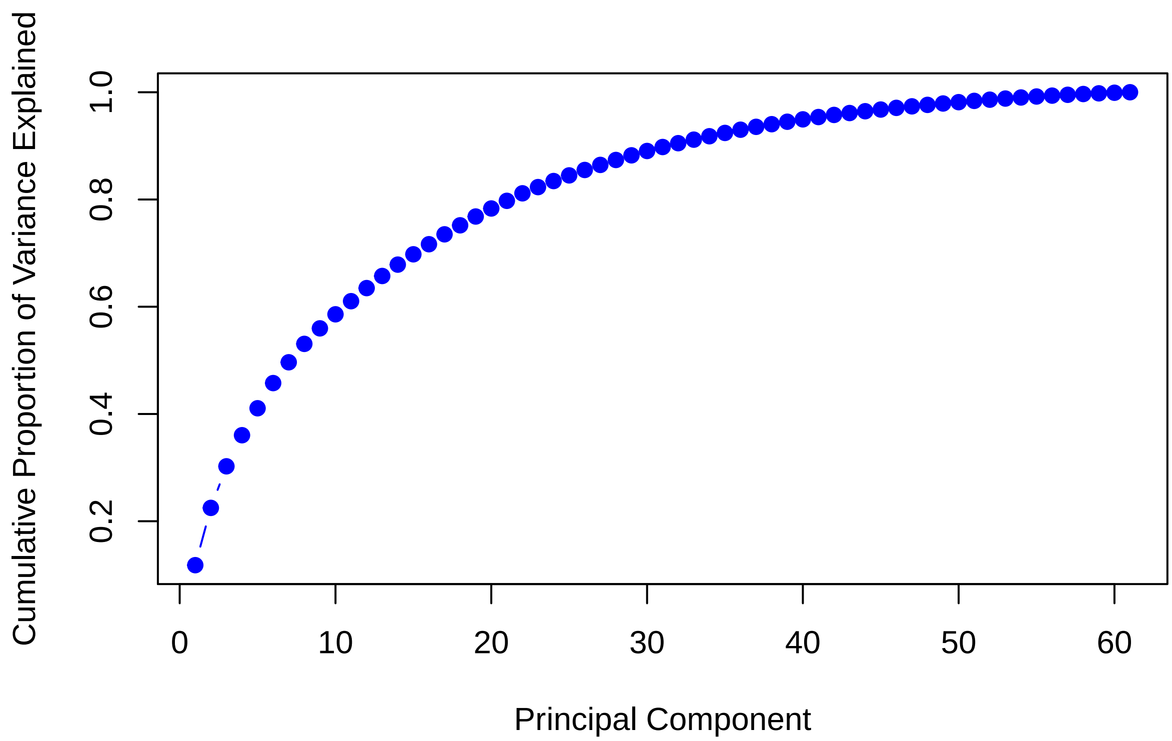

In the plot we see that the similar digits are closer to one another and make a cluster. Sometimes they overlap with the same or different digits.

Here we plot the proprotion of variance and cumulative proportion of variance with their prinicipal components. After that we show the PARETO plot to put variances and cumulative variances together.

```
prop.pc <- var.pc/sum(var.pc)
plot(prop.pc, xlab = "Principal Component",
     ylab = "Proportion of Variance Explained", type = "b")
```

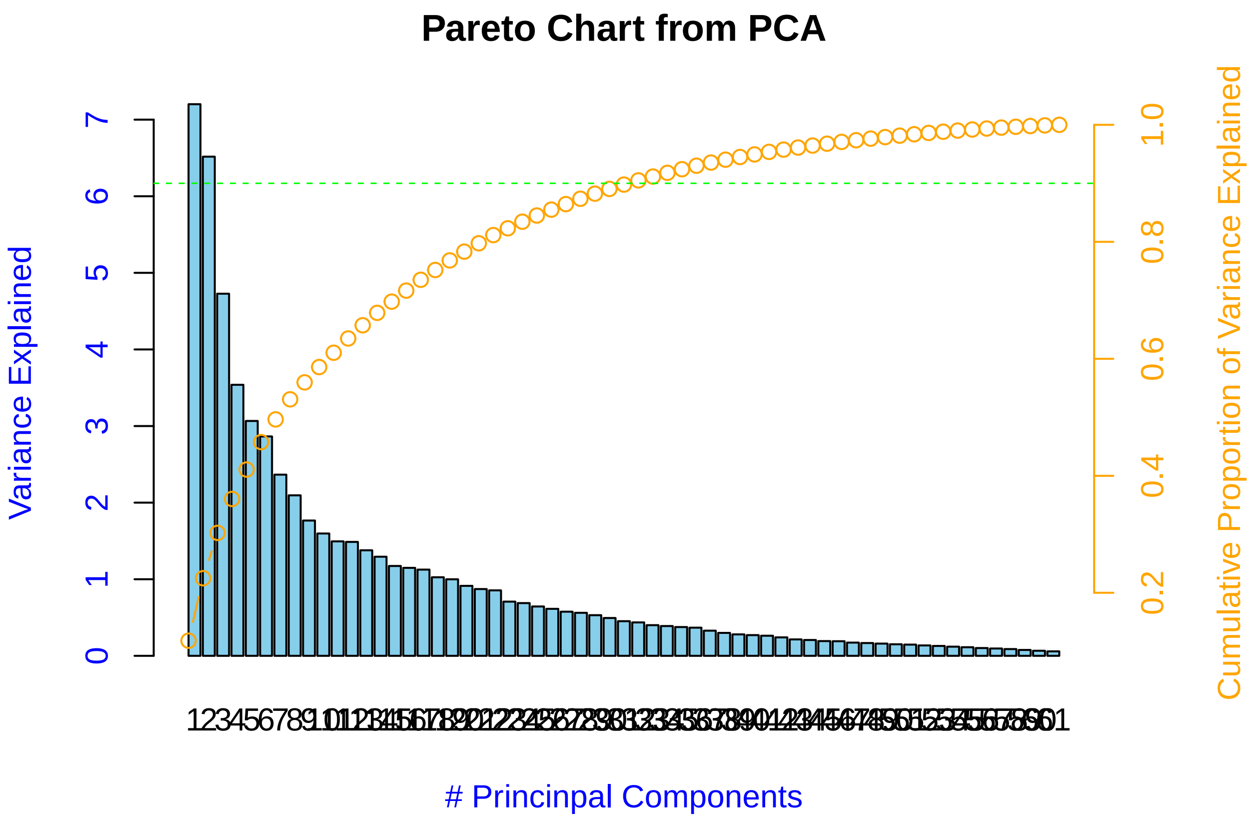



```r
plot(cumsum(prop.pc), xlab = "Principal Component", col="blue",
     ylab = "Cumulative Proportion of Variance Explained",
     type = "b", pch=19) # for cumulative
```




```r
#The Pareto plot
par(mar=c(4, 4, 4, 4))
bar <- barplot(var.pc, ylab="Variance Explained", col="skyblue",
```

```
               xlab="# Princinpal Components", col.axis="blue", col.lab="blue")
mtext(1:length(var.pc),side=1, line=1,at=bar,col="black")
par(new=T)
plot(bar, cumsum(prop.pc),axes=F,xlab="", ylab="", col="orange", type="b",
     col.lab="orange", col.lab="orange")
axis(4,col="orange", col.ticks="orange", col.axis="orange")
abline(h=.90, lty=2, col="green", lwd=.8)
mtext("Cumulative Proportion of Variance Explained",side=4,line=3,col="orange")
title(main = 'Pareto Chart from PCA')
```



# 3 Problme 1(c)

Now we analyze the Kernel PC for hand digits training data.

## 3.1 Kernel PCA

```
library(kernlab)
```

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

```
kpc <- kpca(~., data=dat0.scaled, kernel="rbfdot", kpar=list(sigma=0.0001), features=2)
eig(kpc)        # returns the eigenvalues
```

```
##      Comp.1      Comp.2
## 0.001414212 0.001282923
```

```
kernelf(kpc)    # returns the kernel used when kpca was performed
```

```
## Gaussian Radial Basis kernel function.
##  Hyperparameter : sigma =  1e-04
```

```
PCV <- pcv(kpc) # returns the principal component vectors
dim(PCV)
```

```
## [1] 3823    2
```

```
head(PCV)
```

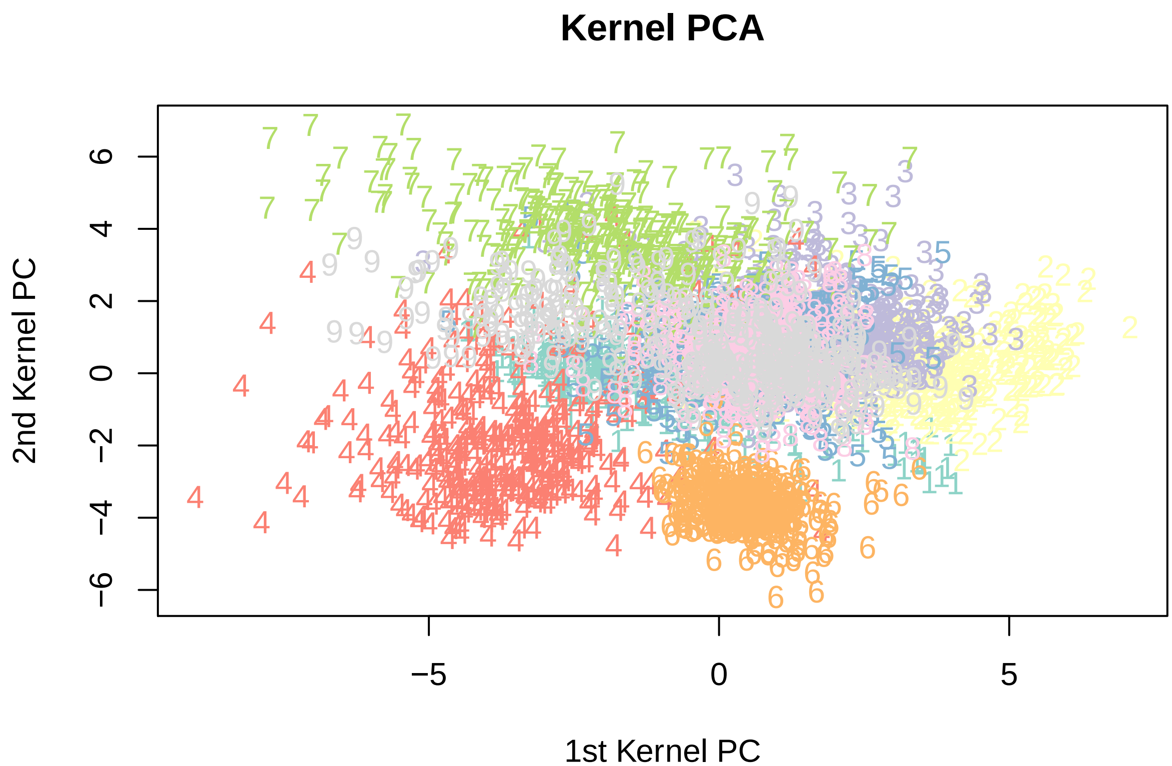
```
##               [,1]       [,2]
## [1,]  0.0004294026 -0.2721069
## [2,]  0.0753832032 -0.5374814
## [3,] -0.3856908536 -0.1507654
## [4,]  0.0063539937 -0.2296744
## [5,] -0.0522150114 -0.4988959
## [6,]  0.1379991890 -0.4527125
```

```
PC <- rotated(kpc)     # returns the data projected in the (kernel) pca space
head(PC);
```

```
##           [,1]       [,2]
## 1  0.002321578 -1.3345790
## 2  0.407561621 -2.6361388
## 3 -2.085249537 -0.7394462
## 4  0.034353064 -1.1264640
## 5 -0.282302075 -2.4468916
## 6  0.746096886 -2.2203801
```

Now we plot the data projections for two KPC of digits train data.

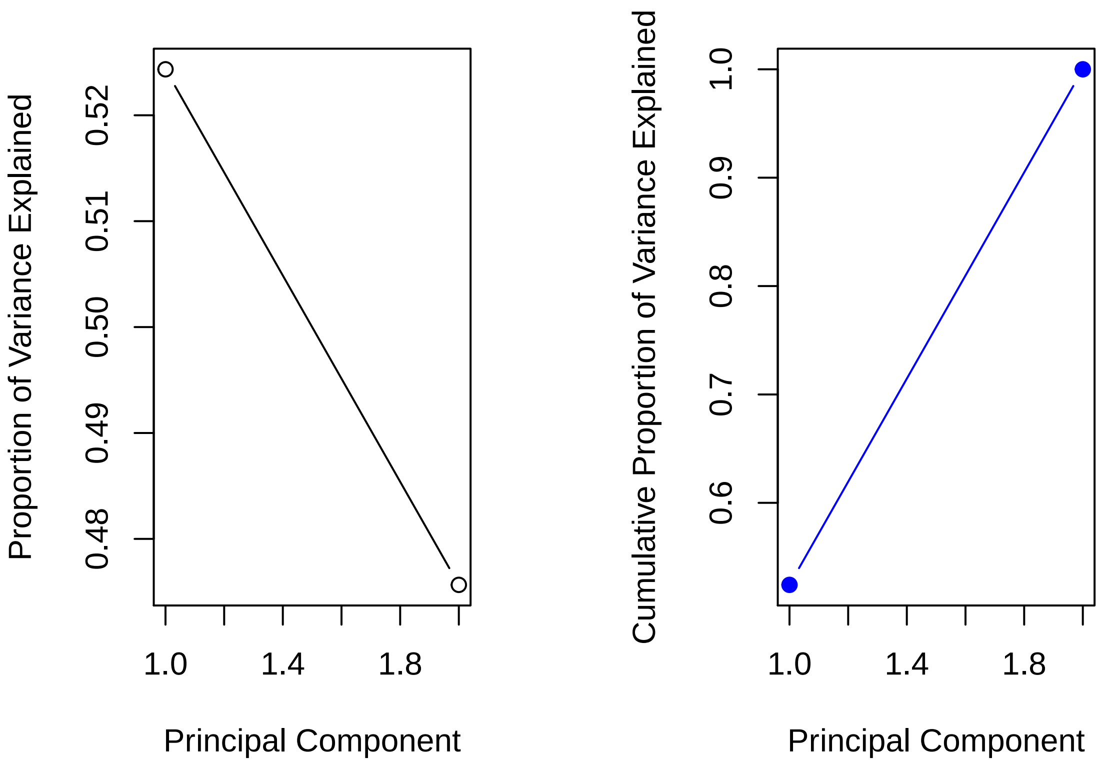
```
plot(PC, pch="", xlab="1st Kernel PC", ylab="2nd Kernel PC", main="Kernel PCA")
text(PC, labels=labs, col=labs)
```



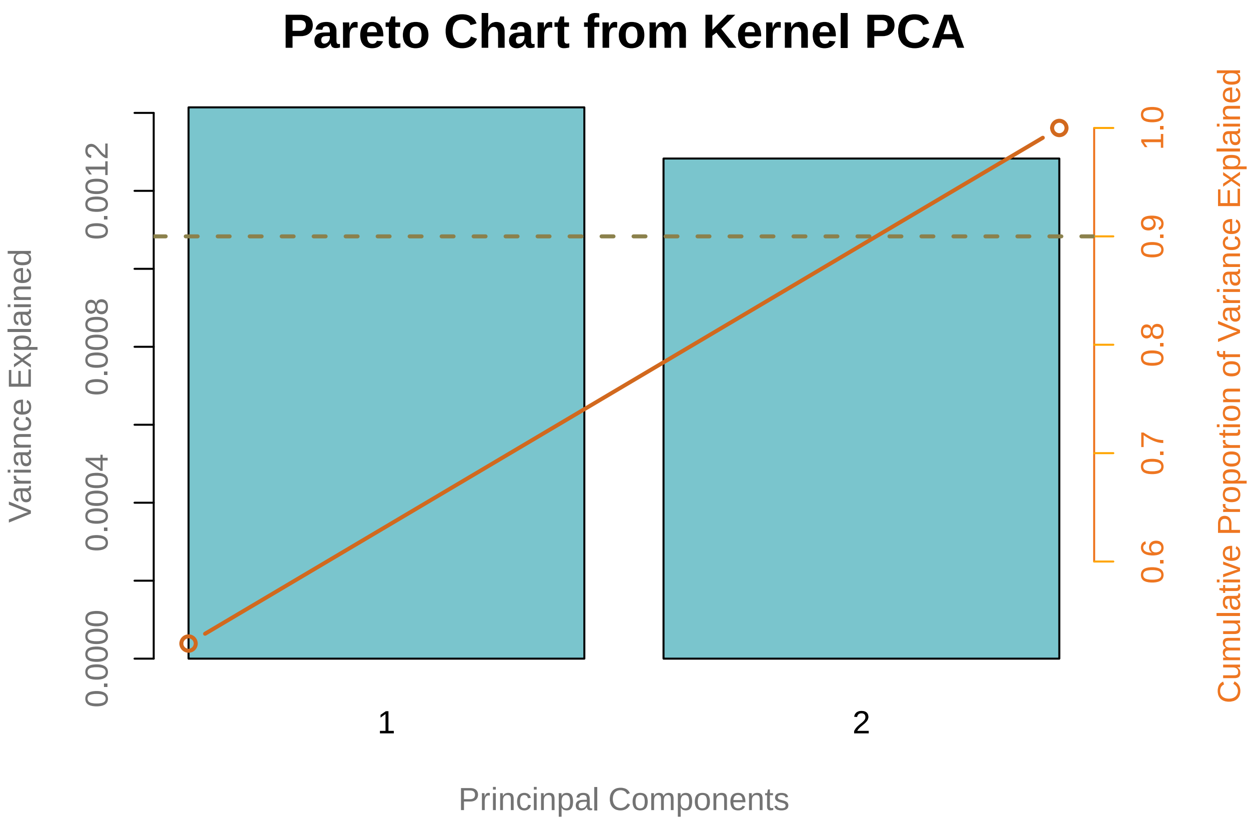

Here we compute noncumulative and cumulative proportions of variation explained. And, we show the pareto plot to put variances and cumulative variances together.

```
var.pc <- eig(kpc)
names(var.pc) <- 1:length(var.pc)
prop.pc <- var.pc/sum(var.pc)

par(mfrow=c(1,2), mar=rep(4,4))
# NONCUMULATIVE
plot(prop.pc, xlab = "Principal Component",
     ylab = "Proportion of Variance Explained", type = "b")
# CUMULATIVE
plot(cumsum(prop.pc), xlab = "Principal Component", col="blue",
     ylab = "Cumulative Proportion of Variance Explained",
     type = "b", pch=19)
```




```r
#THE PARETO PLOT
par(mar=c(4, 4, 4, 4), mfrow=c(1,1))
bar <- barplot(var.pc, ylab="Variance Explained", col="cadetblue3",
               xlab="Princinpal Components", col.axis="gray45", col.lab="gray45")
mtext(1:length(var.pc),side=1, line=1,at=bar,col="black")
par(new=T)
plot(bar, cumsum(prop.pc),axes=F,xlab="", ylab="", col="chocolate", type="b",
     col.lab="orange", col.lab="orange", lwd=2)
axis(4, col="chocolate2", col.ticks="orange", col.axis="chocolate2")
abline(h=.90, lty=2, col="lightgoldenrod4", lwd=2)
mtext("Cumulative Proportion of Variance Explained", side=4, line=3,col="chocolate2")
title(main = 'Pareto Chart from Kernel PCA', cex.main=1.5)
```

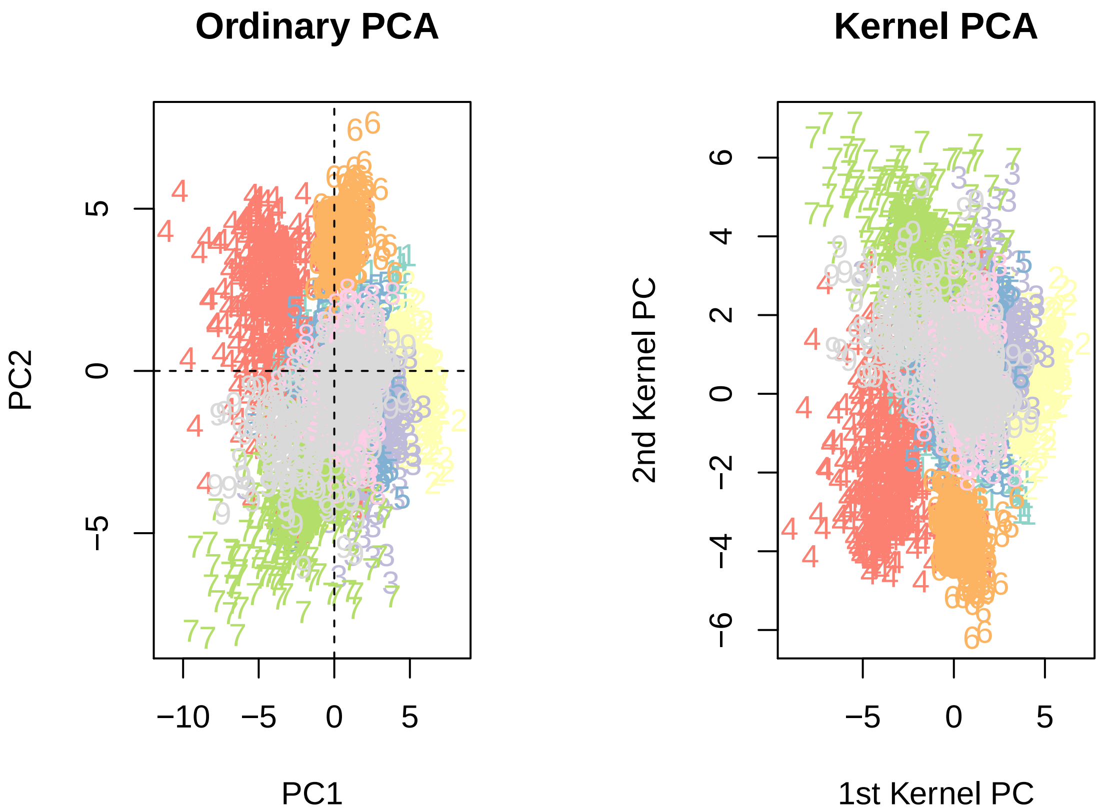



### 3.2 Comparison

I then compare the oridnary PC and kernel PC as follows:

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

plot(PC, pch="", xlab="1st Kernel PC", ylab="2nd Kernel PC", main="Kernel PCA")
text(PC, labels=labs, col=labs)
```




From these plots, we notice that the Kernel PCA provides good representative direction. Clusters are more visible in the case of Kernal PCA than the oridinary PCA. Digits are more overlapping in Kernel PCA plot than the ordinary plot.

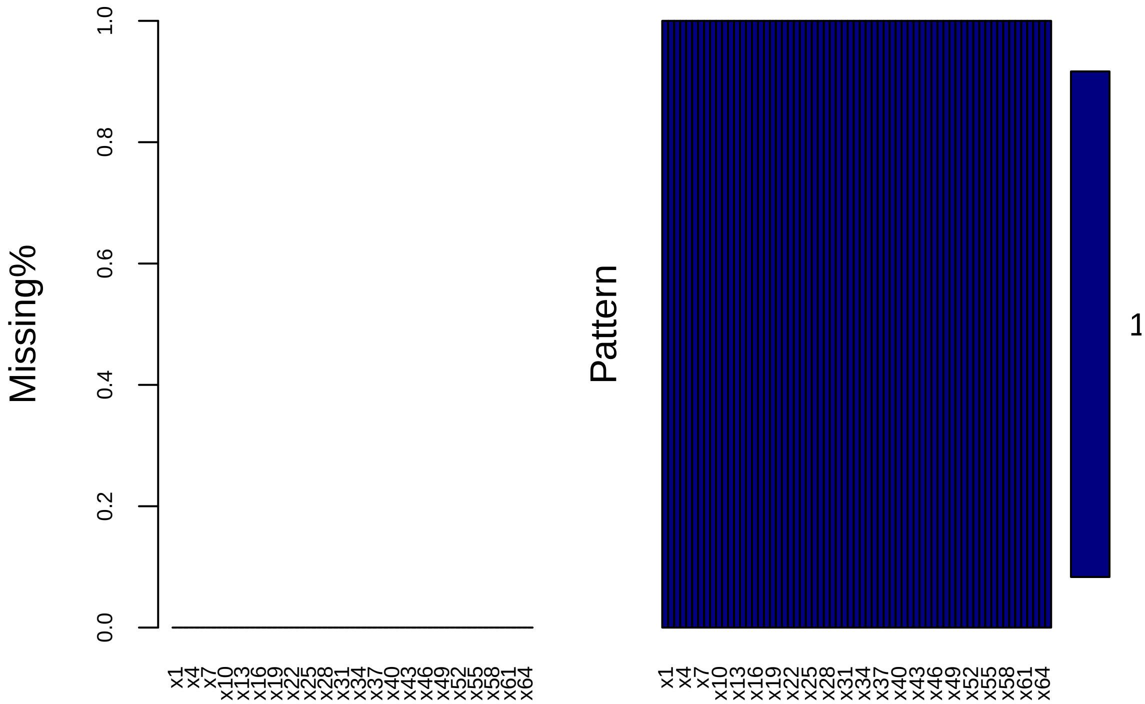
## 3.3 Problem 1(d)

### 3.3.1 Test data collection

```
# ====================================================================
test <- read.table(file="http://archive.ics.uci.edu/ml/machine-learning-databases/optdigits/op
```

### 3.3.2 Detecting missing values for test data

```
#install.packages("VIM")
library(VIM)
aggr(test, col=c('navyblue','yellow'), numbers=TRUE, sortVars=TRUE,
    labels=names(train), cex.axis=.7, gap=3, ylab=c("Missing%","Pattern"))
```




```
##
##  Variables sorted by number of missings:
##  Variable Count
##        x1     0
##        x2     0
##        x3     0
##        x4     0
##        x5     0
##        x6     0
##        x7     0
##        x8     0
##        x9     0
##       x10     0
##       x11     0
##       x12     0
##       x13     0
##       x14     0
##       x15     0
##       x16     0
##       x17     0
##       x18     0
##       x19     0
##       x20     0
##       x21     0
##       x22     0
##       x23     0
##       x24     0
##       x25     0
```

```
##        x26     0
##        x27     0
##        x28     0
##        x29     0
##        x30     0
##        x31     0
##        x32     0
##        x33     0
##        x34     0
##        x35     0
##        x36     0
##        x37     0
##        x38     0
##        x39     0
##        x40     0
##        x41     0
##        x42     0
##        x43     0
##        x44     0
##        x45     0
##        x46     0
##        x47     0
##        x48     0
##        x49     0
##        x50     0
##        x51     0
##        x52     0
##        x53     0
##        x54     0
##        x55     0
##        x56     0
##        x57     0
##        x58     0
##        x59     0
##        x60     0
##        x61     0
##        x62     0
##        x63     0
##        x64     0
##      digit     0
```

```
any(is.na(test))  #Checking for NAs
```

```
## [1] FALSE
```

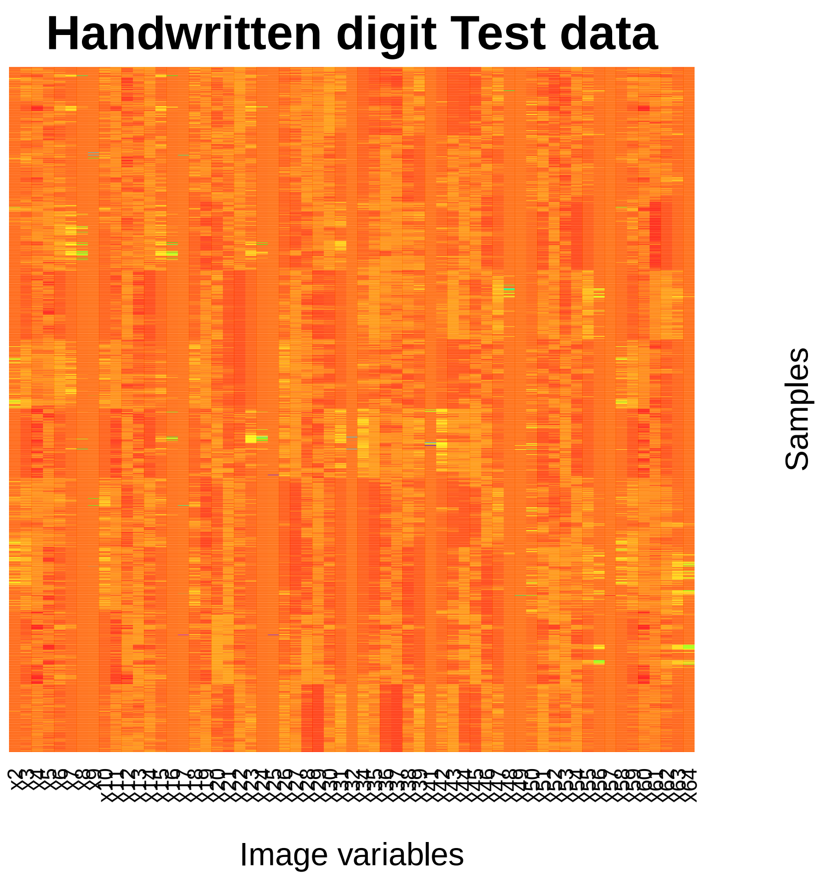
There are no missing values in the test data.

Since we would use the predict function for test data, so we remove the knonw digits column (65) and column (1, 33 and 40) to make same dimension of train data. Here, 33th colmun has zero values, so it is better to remove for scaling the testing data.

```
dat1 <- data.matrix(test[order(test$digit),])
labs <- dat1[, c(65)] # Labels for plotting needed later
table(test$x33)
```

```
##
##    0
## 1797
```

```
dat1 <- dat1[, -c(1, 40, 33, 65)]
n <- NROW(dat1)
color <- rainbow(n, alpha=0.8)
heatmap(dat1, col=color, scale="column", Rowv=NA, Colv=NA,
        labRow=FALSE, margins=c(4,4), xlab="Image variables", ylab="Samples", main="Handwritten
```



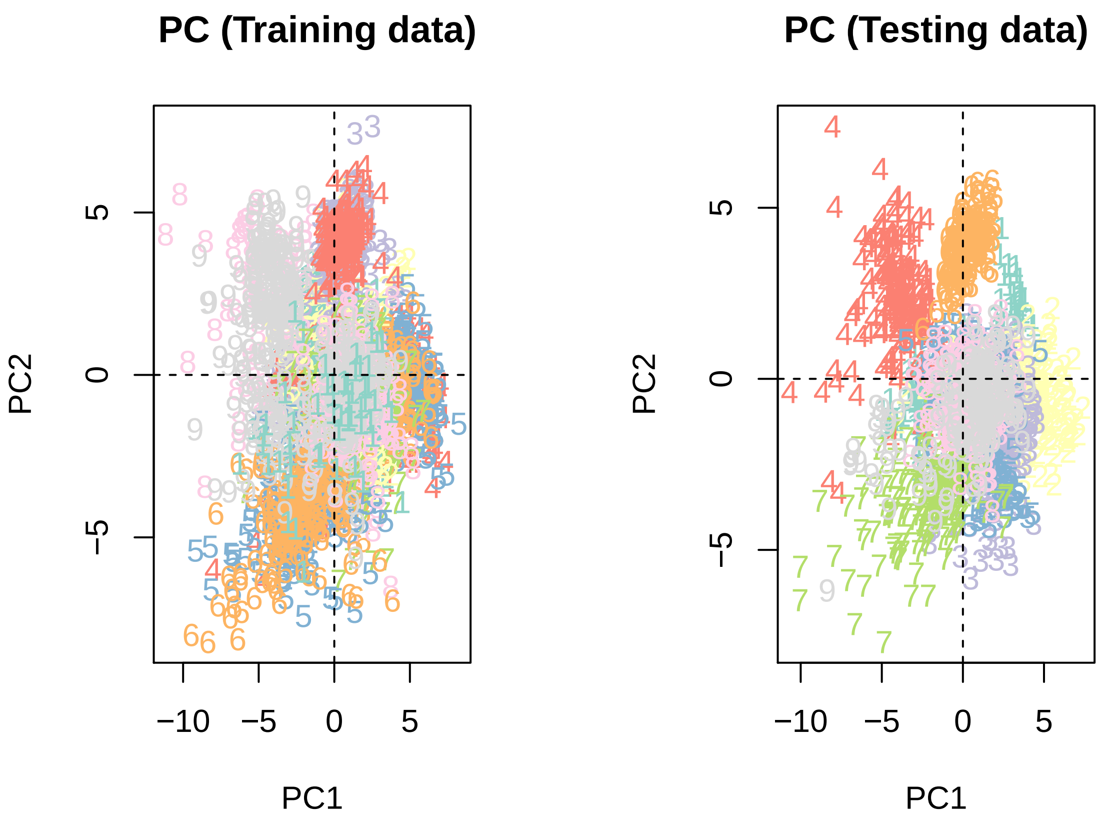

Now we scale the test data before prediction.

```
test.scaled <- data.frame(apply(dat1, 2, scale))
pred_pca <- predict(pca.res, test.scaled)
```

Now we plot PC2 versus PC1, where the 'dots' for each digit are represented with different colors and digit symbols of testing data.

```
par(mfrow=c(1,2), mar=rep(4,4))
plot(pca.res$x[,1:2], pch="", main=" PC (Training data)")
```

```r
text(pca.res$x[,1:2], labels=labs, col=labs)
abline(v=0, lty=2)
abline(h=0, lty=2)
#=======================================================
plot(pred_pca[,1:2], pch="", main="PC (Testing data)")
text(pred_pca[,1:2], labels=labs, col=labs)
abline(v=0, lty=2)
abline(h=0, lty=2)
```

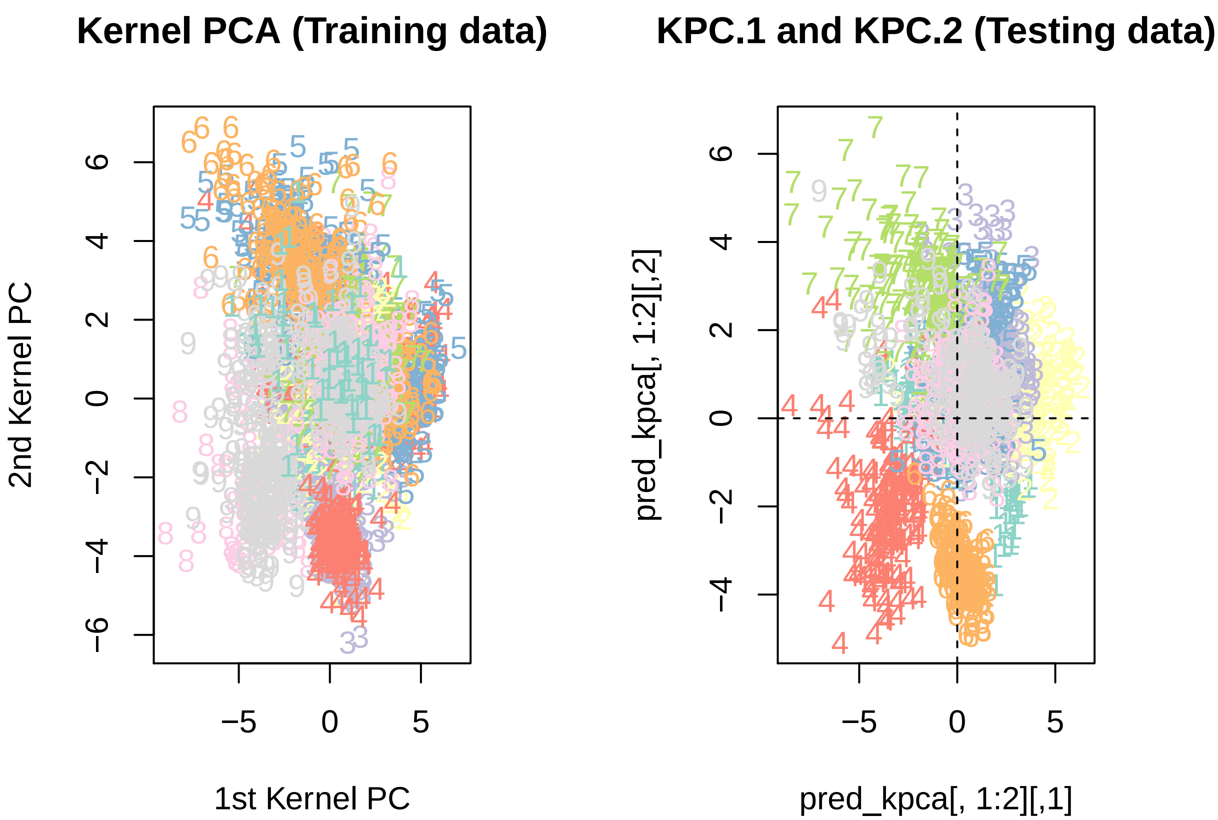



The two plots show the PCA between train data and test data. We conlclude that PCs of testing data are able to effecively predict the test data, since they show the more visible cluster than train data.

Now we plot KPC2 versus KPC1, where the 'dots' for each digit are represented with different colors and digit symbols of testing data.

```r
pred_kpca <- predict(kpc, test.scaled)
```

```r
par(mfrow=c(1,2), mar=rep(4,4))
plot(PC, pch="", xlab="1st Kernel PC", ylab="2nd Kernel PC", main="Kernel PCA (Training data)")
text(PC, labels=labs, col=labs)
#==================================
plot(pred_kpca[,1:2], pch="", main="KPC.1 and KPC.2 (Testing data)")
text(pred_kpca[,1:2], labels=labs, col=labs)
abline(v=0, lty=2)
abline(h=0, lty=2)
```




We conlclude that KPCs of testing data are able to effecively predict the test data, since they show the more visible cluster than train data.

# 4 Problem 2

In this section, we analyze the association rules.

## 4.1 Problem 2(a)

```
library(arules)
dat2 <- read.transactions(file="/Users/masum/Desktop/Fall_2018/Data_Mining/LAB/Lecture-5/AV1611
dim(dat2)
```

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

```
inspect(dat2[1:5, ])
```

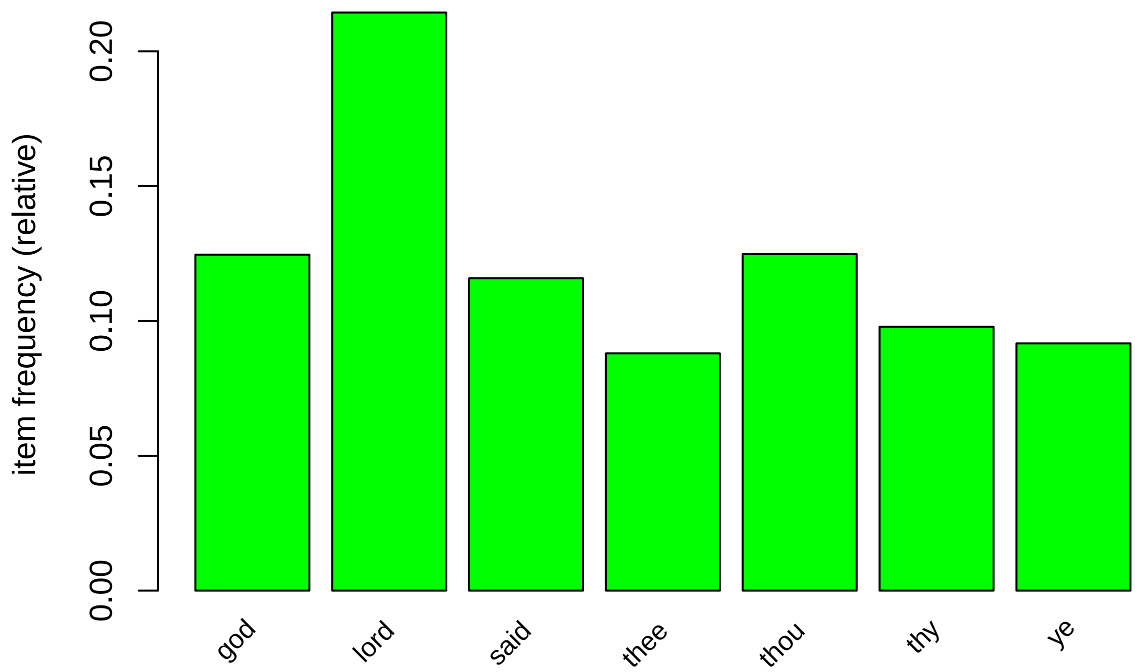
```
##     items
## [1] {beginning,
##      created,
##      earth,
##      god,
##      heaven}
```

```
## [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}
```

### 4.2 Problem 2(b)

I plot the items with high frequencies and perform the frequent itemsets with support 0.08.

```
itemFrequencyPlot(dat2, support = 0.08, cex.names = 0.8, col="green")
```




Therefore, the item with the higest frequency is “lord”.

```
# The top ten items
item.freq <- itemFrequency(dat2, type = "relative")
item.freq <- sort(item.freq, decreasing = TRUE)
item.freq[1:10]
```

```
##       lord       thou        god       said        thy         ye
## 0.21436610 0.12478698 0.12459406 0.11581621 0.09787467 0.09166908
##       thee        out        man     israel
## 0.08797145 0.07893637 0.07491721 0.07372753
```

I then Set up the parameters and function “apriori” with choices support 0.001 explore the resultant rules using summary function.

```
(rules <- apriori(dat2, parameter = list(support = 0.01, confidence = 0.6,
                                         target ="rules", maxlen=6)))
```

```
## Apriori
##
## Parameter specification:
##  confidence minval smax arem  aval originalSupport maxtime support minlen
##         0.6    0.1    1 none FALSE            TRUE       5    0.01      1
##  maxlen target   ext
##       6  rules FALSE
##
## 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 ... [17 rule(s)] done [0.00s].
## creating S4 object  ... done [0.01s].

## set of 17 rules
```

```r
inspect(rules[1:12])
```

```
##      lhs              rhs     support    confidence lift      count
## [1]  {answered}    => {said}  0.01067490 0.6775510   5.850226  332
## [2]  {art}         => {thou}  0.01434037 0.9867257   7.907280  446
## [3]  {she}         => {her}   0.01408315 0.6016484  15.684715  438
## [4]  {thus}        => {saith} 0.01462975 0.6435644  16.721383  455
## [5]  {thus}        => {lord}  0.01626957 0.7157001   3.338682  506
## [6]  {hast}        => {thou}  0.02684801 0.9835100   7.881511  835
## [7]  {saith}       => {lord}  0.02819845 0.7326650   3.417821  877
## [8]  {shalt}       => {thou}  0.03900196 0.9991763   8.007055 1213
## [9]  {saith,thus} => {lord}  0.01389023 0.9494505   4.429108  432
## [10] {lord,thus}  => {saith} 0.01389023 0.8537549  22.182650  432
## [11] {hast,thy}   => {thou}  0.01102858 0.9744318   7.808762  343
## [12] {god,saith}  => {lord}  0.01109289 0.8984375   4.191136  345
```

```r
summary(rules)
```

```
## set of 17 rules
##
## rule length distribution (lhs + rhs):sizes
## 2 3
## 8 9
##
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   2.000   2.000   3.000   2.529   3.000   3.000
##
## summary of quality measures:
##     support           confidence          lift            count
##  Min.   :0.01067   Min.   :0.6016   Min.   : 3.258   Min.   : 332.0
##  1st Qu.:0.01225   1st Qu.:0.7033   1st Qu.: 4.191   1st Qu.: 381.0
##  Median :0.01389   Median :0.8984   Median : 7.882   Median : 432.0
##  Mean   :0.01637   Mean   :0.8480   Mean   : 8.116   Mean   : 509.2
##  3rd Qu.:0.01514   3rd Qu.:0.9867   3rd Qu.: 8.014   3rd Qu.: 471.0
##  Max.   :0.03900   Max.   :1.0000   Max.   :22.183   Max.   :1213.0
##
## mining info:
##  data ntransactions support confidence
##  dat2         31101    0.01        0.6
```

### 4.3 Problem 2(c)

```
RULES <- as(rules, "data.frame")
rules0 <- data.frame(matrix(unlist(strsplit(as.character(RULES$rules), split="=>")), ncol=2, by
colnames(rules0) <- c("LHS", "RHS")
rule.size <- function(x){length(unlist(strsplit(as.character(x), split=",")))}
rules0$size <- apply(rules0, 1, rule.size)
rules0$size[as.character(rules0$LHS)=="{} "] <- rules0$size[as.character(RULES$LHS)=="{} "]-1
RULES <- cbind(RULES, rules0)
head(RULES,12)
```

```
##                       rules    support confidence      lift count
## 1    {answered} => {said} 0.01067490  0.6775510  5.850226   332
## 2         {art} => {thou} 0.01434037  0.9867257  7.907280   446
## 3          {she} => {her} 0.01408315  0.6016484 15.684715   438
## 4       {thus} => {saith} 0.01462975  0.6435644 16.721383   455
## 5        {thus} => {lord} 0.01626957  0.7157001  3.338682   506
## 6        {hast} => {thou} 0.02684801  0.9835100  7.881511   835
## 7       {saith} => {lord} 0.02819845  0.7326650  3.417821   877
## 8       {shalt} => {thou} 0.03900196  0.9991763  8.007055  1213
## 9  {saith,thus} => {lord} 0.01389023  0.9494505  4.429108   432
## 10 {lord,thus} => {saith} 0.01389023  0.8537549 22.182650   432
## 11   {hast,thy} => {thou} 0.01102858  0.9744318  7.808762   343
## 12  {god,saith} => {lord} 0.01109289  0.8984375  4.191136   345
##              LHS       RHS size
## 1    {answered}    {said}    2
## 2         {art}    {thou}    2
## 3         {she}     {her}    2
## 4        {thus}   {saith}    2
## 5        {thus}    {lord}    2
## 6        {hast}    {thou}    2
## 7       {saith}    {lord}    2
## 8       {shalt}    {thou}    2
## 9  {saith,thus}    {lord}    3
## 10  {lord,thus}   {saith}    3
## 11    {hast,thy}    {thou}    3
## 12  {god,saith}    {lord}    3
```

I list the top 5 rules in decreasing order of confidence (conf) for item sets of size 2 and 3 which satisfy the support threshold 0.01.

```
RULES1 <- RULES[RULES$size==2, ]
RULES1 <- RULES[ order(RULES$confidence,decreasing = TRUE), ]
head(RULES1,n=5)
```

```
##                       rules    support confidence     lift count
## 13 {shalt,thee} => {thou} 0.01270056  1.0000000 8.013656   395
## 14  {shalt,thy} => {thou} 0.01514421  1.0000000 8.013656   471
## 8       {shalt} => {thou} 0.03900196  0.9991763 8.007055  1213
```

```
## 15 {lord,shalt} => {thou} 0.01225041  0.9973822 7.992678   381
## 2         {art} => {thou} 0.01434037  0.9867257 7.907280   446
##               LHS     RHS size
## 13 {shalt,thee}   {thou}    3
## 14  {shalt,thy}   {thou}    3
## 8       {shalt}   {thou}    2
## 15 {lord,shalt}   {thou}    3
## 2         {art}   {thou}    2
```

We then perform frequent itemsets by listing the top 5 rules in decreasing order of the lift measure for item sets of size 2 and 3.

### 4.4 Problem 2(d)

```r
RULES1 <- RULES[RULES$size==2, ]
RULES1 <- RULES[ order(RULES$lift, decreasing = TRUE), ]
head(RULES1,n=5)
```

```
##                         rules    support confidence      lift count
## 10 {lord,thus} => {saith} 0.01389023  0.8537549 22.182650   432
## 4       {thus} => {saith} 0.01462975  0.6435644 16.721383   455
## 3           {she} => {her} 0.01408315  0.6016484 15.684715   438
## 13 {shalt,thee} => {thou} 0.01270056  1.0000000  8.013656   395
## 14  {shalt,thy} => {thou} 0.01514421  1.0000000  8.013656   471
##               LHS      RHS size
## 10  {lord,thus}   {saith}    3
## 4        {thus}   {saith}    2
## 3         {she}     {her}    2
## 13 {shalt,thee}    {thou}    3
## 14  {shalt,thy}    {thou}    3
```

### 4.5 Problem 2(e)

Conviction actually captures the notion of an implication rules. Unlike lift, Conviction is sensitive to rule direction (i.e. $conv(A \to B) \neq conv(B \to A)$). It attempts to measure the degree of implication of a rule. Unlike Confidence, the support of the both antecedent and consequent are considered in conviction.