Project03

Isaiah Thompson Ocansey

2022-10-11

(1a) Bring in the training set optdigits.tra, which has sixty-four inputs plus the target variable that indicates the digit . Examine the data briefly. Remove columns that are unary (i.e., containing only one values) or close to being unary (i.e., nearly all values are the same except a few). And check on possible missing values.

```
traindat<- read.table(file=
"https://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"))
testdat <- read.table(file=
"https://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(traindat); dim(testdat)
```

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

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

The training data set has 3823 observations and 65 variables whiles the testing data set has 1797 observations and 65 variables.

```
data<-rbind(traindat,testdat)
head(data);dim(data)
```

```
##   x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15 x16 x17 x18 x19 x20 x21
## 1  0  1  6 15 12  1  0  0  0   7  16   6   6  10   0   0   0   8  16   2   0
## 2  0  0 10 16  6  0  0  0  0   7  16   8  16   5   0   0   0  11  16   0   6
## 3  0  0  8 15 16 13  0  0  0   1  11   9  11  16   1   0   0   0   0   0   7
## 4  0  0  0  3 11 16  0  0  0   0   5  16  11  13   7   0   0   3  15   8   1
## 5  0  0  5 14  4  0  0  0  0   0  13   8   0   0   0   0   0   3  14   4   0
## 6  0  0 11 16 10  1  0  0  0   4  16  10  15   8   0   0   0   4  16   3  11
##   x22 x23 x24 x25 x26 x27 x28 x29 x30 x31 x32 x33 x34 x35 x36 x37 x38 x39 x40
## 1  11   2   0   0   5  16   3   0   5   7   0   0   7  13   3   0   8   7   0
## 2  14   3   0   0  12  12   0   0  11  11   0   0  12  12   0   0   8  12   0
## 3  14   0   0   0   0   3   4  14  12   2   0   0   1  16  16  16  16  10   0
## 4  15   6   0   0  11  16  16  16  16  10   0   0   1   4   4  13  10   2   0
## 5   0   0   0   0   6  16  14   9   2   0   0   0   4  16   3   4  11   2   0
## 6  13   0   0   0   1  14   6   9  14   0   0   0   0   0   0  12  10   0   0
##   x41 x42 x43 x44 x45 x46 x47 x48 x49 x50 x51 x52 x53 x54 x55 x56 x57 x58 x59
## 1   0   4  12   0   1  13   5   0   0   0  14   9  15   9   0   0   0   0   6
```

```
## 2   0   7  15   1   0  13  11   0   0   0  16   8  10  15   3   0   0   0  10
## 3   0   2  12  16  10   0   0   0   0   0   2  16   4   0   0   0   0   0   9
## 4   0   0   0   0  15   4   0   0   0   0   0   3  16   0   0   0   0   0   0
## 5   0   0  14   3   0   4  11   0   0   0  10   8   4  11  12   0   0   0   4
## 6   0   0   0   6  16   6   0   0   0   0   5  15  15   8   8   3   0   0  10
##   x60 x61 x62 x63 x64 digit
## 1  14   7   1   0   0     0
## 2  16  15   3   0   0     0
## 3  14   0   0   0   0     7
## 4   1  15   2   0   0     4
## 5  12  14   7   0   0     6
## 6  16  16  16  16   6     2
```

```
## [1] 5620   65
```

In order to delete unitary variables and check for missing values, we combine both the testing and the training data sets. The combined data set has 5620 observations and 65 variables.

```
#Check Distinct values of the data

for (j in 1:NCOL(data)){
x <- data[,j]
print(table(x, useNA="ifany"))
}
```

```
## x
##    0
## 5620
## x
##    0    1    2    3    4    5    6    7    8
## 4761  424  221  112   55   23   11    9    4
## x
##    0    1    2    3    4    5    6    7    8    9   10   11   12   13   14   15
## 1266  358  381  363  355  363  368  333  317  312  263  206  195  174  110  151
##   16
##  105
## x
##    0    1    2    3    4    5    6    7    8    9   10   11   12   13   14   15
##  162  108   81   72   91   82  110  117  211  229  302  343  637  588  626  706
##   16
## 1155
## x
##    0    1    2    3    4    5    6    7    8    9   10   11   12   13   14   15
##  183   97   98   94  110  108  151  165  236  211  289  322  555  556  542  689
##   16
## 1214
## x
##    0    1    2    3    4    5    6    7    8    9   10   11   12   13   14   15
## 1611  505  367  282  251  250  220  177  217  154  189  145  244  200  156  212
##   16
##  440
## x
##    0    1    2    3    4    5    6    7    8    9   10   11   12   13   14   15
```

```
## 4277  285  178  132  104   90   68   71   59   44   44   48   52   28   44   32
##   16
##   64
## x
##    0    1    2    3    4    5    6    7    8    9   10   11   12   13   14   15
## 5458   40   28   14   12   13   10    4   12    4    5    4    8    1    3    3
##   16
##    1
## x
##    0    1    2    5
## 5610    5    4    1
## x
##    0    1    2    3    4    5    6    7    8    9   10   11   12   13   14   15
## 3228  420  356  307  292  200  177  179  154   79   79   66   36   18   19    9
##   16
##    1
## x
##    0    1    2    3    4    5    6    7    8    9   10   11   12   13   14   15
##  502  131  133  160  150  161  175  177  227  210  228  276  320  368  455  645
##   16
## 1302
## x
##    0    1    2    3    4    5    6    7    8    9   10   11   12   13   14   15
##   53   29   63   44  145  172  186  189  370  330  324  323  440  496  500  598
##   16
## 1358
## x
##    0    1    2    3    4    5    6    7    8    9   10   11   12   13   14   15
##  217  114  113  111  247  226  206  217  448  277  338  352  396  368  373  414
##   16
## 1203
## x
##    0    1    2    3    4    5    6    7    8    9   10   11   12   13   14   15
## 1111  191  200  185  241  176  169  190  288  185  213  244  336  332  305  393
##   16
##  861
## x
##    0    1    2    3    4    5    6    7    8    9   10   11   12   13   14   15
## 3652  290  295  210  204  134   98  109  124   72   50   62   56   57   56   53
##   16
##   98
## x
##    0    1    2    3    4    5    6    7    8    9   10   11   12   13   15
## 5429   50   34   26   11   15   12   13    8    8    3    6    2    2    1
## x
##    0    1    2    3    5
## 5607    7    3    2    1
## x
##    0    1    2    3    4    5    6    7    8    9   10   11   12   13   14   15
## 2745  393  402  345  357  247  227  220  237  113  107   77   74   23   26   11
##   16
##   16
## x
##    0    1    2    3    4    5    6    7    8    9   10   11   12   13   14   15
```

```
##  792  163  162  161  137  157  152  171  200  199  236  264  354  351  445  501
##   16
## 1175
## x
##    0    1    2    3    4    5    6    7    8    9   10   11   12   13   14   15
## 1225  422  304  268  282  257  229  204  244  214  191  189  196  199  219  239
##   16
##  738
## x
##    0    1    2    3    4    5    6    7    8    9   10   11   12   13   14   15
## 1367  343  259  235  239  224  194  165  218  150  210  176  235  220  243  216
##   16
##  926
## x
##    0    1    2    3    4    5    6    7    8    9   10   11   12   13   14   15
## 1504  136  177  114  169  123  158  148  223  191  200  222  313  296  396  403
##   16
##  847
## x
##    0    1    2    3    4    5    6    7    8    9   10   11   12   13   14   15
## 3518  305  305  241  325  140  176  107  125   54   63   60   57   20   36   39
##   16
##   49
## x
##    0    1    2    3    4    5    6    7    8
## 5520   32   21   15   15    8    6    1    2
## x
##    0    1
## 5614    6
## x
##    0    1    2    3    4    5    6    7    8    9   10   11   12   13   14   15
## 2787  419  333  328  399  329  274  229  251   99   70   49   35    7    5    5
##   16
##    1
## x
##    0    1    2    3    4    5    6    7    8    9   10   11   12   13   14   15
##  984  195  182  171  165  161  139  169  200  171  149  215  332  280  378  470
##   16
## 1259
## x
##    0    1    2    3    4    5    6    7    8    9   10   11   12   13   14   15
##  819  214  169  199  237  188  193  204  267  210  236  236  330  263  290  314
##   16
## 1251
## x
##    0    1    2    3    4    5    6    7    8    9   10   11   12   13   14   15
## 1044  155  102  105  154  122  101  137  196  153  174  187  288  283  387  452
##   16
## 1580
## x
##    0    1    2    3    4    5    6    7    8    9   10   11   12   13   14   15
## 1160  248  221  240  214  206  234  202  255  215  220  214  285  273  283  326
##   16
##  824
```

```
## x
##    0    1    2    3    4    5    6    7    8    9   10   11   12   13   14   15
## 3359  296  228  188  209  194  226  173  320   92   87   59   59   52   41   34
##   16
##    3
## x
##    0    1    2
## 5606   12    2
## x
##    0    1
## 5615    5
## x
##    0    1    2    3    4    5    6    7    8    9   10   11   12   13   14   15
## 3326  311  252  263  249  260  161  194  265   84   76   74   52   28   23    2
## x
##    0    1    2    3    4    5    6    7    8    9   10   11   12   13   14   15
## 1422  265  216  170  163  160  140  199  275  172  158  177  328  227  226  310
##   16
## 1012
## x
##    0    1    2    3    4    5    6    7    8    9   10   11   12   13   14   15
## 1061  129  163  139  212  158  147  148  278  163  189  208  275  217  303  321
##   16
## 1509
## x
##    0    1    2    3    4    5    6    7    8    9   10   11   12   13   14   15
##  822  120  101   91  146  120  122  141  229  182  210  196  323  314  359  441
##   16
## 1703
## x
##    0    1    2    3    4    5    6    7    8    9   10   11   12   13   14   15
##  924  167  154  148  208  159  181  198  271  215  248  216  363  307  378  459
##   16
## 1024
## x
##    0    1    2    3    4    5    6    7    8    9   10   11   12   13   14
## 2632  341  312  269  307  287  256  254  522  174  147   53   46   13    7
## x
##    0
## 5620
## x
##    0    1    2    3    4    5    6    7
## 5572   16   13    7    6    3    2    1
## x
##    0    1    2    3    4    5    6    7    8    9   10   11   12   13   14   15
## 3798  370  293  234  255  169   99   87   88   34   30   31   32   24   19   31
##   16
##   26
## x
##    0    1    2    3    4    5    6    7    8    9   10   11   12   13   14   15
## 2087  232  164  163  167  115  118  105  146  119  146  151  244  247  253  334
##   16
##  829
## x
```

```
##    0    1    2    3    4    5    6    7    8    9   10   11   12   13   14   15
## 1783  256  138  138  179  136  126  140  196  185  167  173  220  240  281  304
##   16
##  958
## x
##    0    1    2    3    4    5    6    7    8    9   10   11   12   13   14   15
## 1322  298  204  185  213  161  156  150  214  197  190  183  251  248  247  341
##   16
## 1060
## x
##    0    1    2    3    4    5    6    7    8    9   10   11   12   13   14   15
## 1012  177  168  135  222  180  166  209  280  268  257  298  354  368  371  406
##   16
##  749
## x
##    0    1    2    3    4    5    6    7    8    9   10   11   12   13   14   15
## 2626  336  237  256  242  245  204  251  332  192  116  168  175   88   71   47
##   16
##   34
## x
##    0    1    2    3    4    5    6
## 5550   44   14    2    6    1    3
## x
##    0    1    2    3    4    5    7    8   10
## 5584   22    4    4    1    2    1    1    1
## x
##    0    1    2    3    4    5    6    7    8    9   10   11   12   13   14   15
## 4273  422  276  204  142   88   48   53   41   23   13    9    8    8    1    6
##   16
##    5
## x
##    0    1    2    3    4    5    6    7    8    9   10   11   12   13   14   15
## 1100  212  197  193  242  218  221  233  313  253  251  338  336  294  332  330
##   16
##  557
## x
##    0    1    2    3    4    5    6    7    8    9   10   11   12   13   14   15
##  339  154  176  139  330  262  242  237  354  301  310  305  332  314  339  313
##   16
## 1173
## x
##    0    1    2    3    4    5    6    7    8    9   10   11   12   13   14   15
##  366  203  174  141  333  286  239  262  355  257  260  228  315  316  337  387
##   16
## 1161
## x
##    0    1    2    3    4    5    6    7    8    9   10   11   12   13   14   15
## 1057  115  134  135  183  132  150  167  238  198  221  265  358  344  413  541
##   16
##  969
## x
##    0    1    2    3    4    5    6    7    8    9   10   11   12   13   14   15
## 2723  274  257  251  252  199  188  219  216  126  114  156  160   87   98  139
##   16
```

```
##  161
## x
##    0    1    2    3    4    5    6    7    8    9   10   12   13
## 5300   94   78   54   29   32   12    7    7    2    3    1    1
## x
##    0    1
## 5618    2
## x
##    0    1    2    3    4    5    6    7    8    9   10
## 4922  283  193  101   58   33   12    9    3    4    2
## x
##    0    1    2    3    4    5    6    7    8    9   10   11   12   13   14   15
## 1288  339  373  325  324  339  314  308  294  254  264  245  204  216  138  198
##   16
##  197
## x
##    0    1    2    3    4    5    6    7    8    9   10   11   12   13   14   15
##  225   55   68   83   79   79  116  122  175  217  245  361  561  578  576  683
##   16
## 1397
## x
##    0    1    2    3    4    5    6    7    8    9   10   11   12   13   14   15
##  394   94  107   79   90  104   85  114  160  181  213  266  575  494  494  619
##   16
## 1551
## x
##    0    1    2    3    4    5    6    7    8    9   10   11   12   13   14   15
## 1494  273  215  253  202  215  218  207  273  245  238  207  316  230  217  269
##   16
##  548
## x
##    0    1    2    3    4    5    6    7    8    9   10   11   12   13   14   15
## 3714  394  204  166  144  120  112  118   88   78   63   71   75   63   30   44
##   16
##  136
## x
##    0    1    2    3    4    5    6    7    8    9   10   11   12   13   14   15
## 5303   96   44   40   15   27   21   12   11    8    3   10    7    6    4    7
##   16
##    6
## x
##   0   1   2   3   4   5   6   7   8   9
## 554 571 557 572 568 558 558 566 554 562
```

```
missing_percent <- function(df, filename=NULL){
vnames <- colnames(df); vnames
n <- nrow(df)
out <- NULL
for (j in 1: ncol(df)){
vname <- colnames(df)[j]
x <- as.vector(df[,j])
n1 <- sum(is.na(x), na.rm=T)
n2 <- sum(x=="NA", na.rm=T)
n3 <- sum(x=="", na.rm=T)
```

```
nmiss <- n1 + n2 + n3
ncomplete <- n-nmiss
out <- rbind(out, c(col.number=j, vname=vname,
mode=mode(x), n.levels=length(unique(x)),
ncomplete=ncomplete, miss.perc=nmiss/n))
}
out <- as.data.frame(out)
row.names(out) <- NULL
if (!is.null(filename)) write.csv(out, file = filename, row.names=F)

return(out)
}
missing_percent(data)
```

```
##    col.number vname    mode n.levels ncomplete miss.perc
## 1           1    x1 numeric        1      5620         0
## 2           2    x2 numeric        9      5620         0
## 3           3    x3 numeric       17      5620         0
## 4           4    x4 numeric       17      5620         0
## 5           5    x5 numeric       17      5620         0
## 6           6    x6 numeric       17      5620         0
## 7           7    x7 numeric       17      5620         0
## 8           8    x8 numeric       17      5620         0
## 9           9    x9 numeric        4      5620         0
## 10         10   x10 numeric       17      5620         0
## 11         11   x11 numeric       17      5620         0
## 12         12   x12 numeric       17      5620         0
## 13         13   x13 numeric       17      5620         0
## 14         14   x14 numeric       17      5620         0
## 15         15   x15 numeric       17      5620         0
## 16         16   x16 numeric       15      5620         0
## 17         17   x17 numeric        5      5620         0
## 18         18   x18 numeric       17      5620         0
## 19         19   x19 numeric       17      5620         0
## 20         20   x20 numeric       17      5620         0
## 21         21   x21 numeric       17      5620         0
## 22         22   x22 numeric       17      5620         0
## 23         23   x23 numeric       17      5620         0
## 24         24   x24 numeric        9      5620         0
## 25         25   x25 numeric        2      5620         0
## 26         26   x26 numeric       17      5620         0
## 27         27   x27 numeric       17      5620         0
## 28         28   x28 numeric       17      5620         0
## 29         29   x29 numeric       17      5620         0
## 30         30   x30 numeric       17      5620         0
## 31         31   x31 numeric       17      5620         0
## 32         32   x32 numeric        3      5620         0
## 33         33   x33 numeric        2      5620         0
## 34         34   x34 numeric       16      5620         0
## 35         35   x35 numeric       17      5620         0
## 36         36   x36 numeric       17      5620         0
## 37         37   x37 numeric       17      5620         0
## 38         38   x38 numeric       17      5620         0
```

```
## 39         39   x39 numeric       15      5620         0
## 40         40   x40 numeric        1      5620         0
## 41         41   x41 numeric        8      5620         0
## 42         42   x42 numeric       17      5620         0
## 43         43   x43 numeric       17      5620         0
## 44         44   x44 numeric       17      5620         0
## 45         45   x45 numeric       17      5620         0
## 46         46   x46 numeric       17      5620         0
## 47         47   x47 numeric       17      5620         0
## 48         48   x48 numeric        7      5620         0
## 49         49   x49 numeric        9      5620         0
## 50         50   x50 numeric       17      5620         0
## 51         51   x51 numeric       17      5620         0
## 52         52   x52 numeric       17      5620         0
## 53         53   x53 numeric       17      5620         0
## 54         54   x54 numeric       17      5620         0
## 55         55   x55 numeric       17      5620         0
## 56         56   x56 numeric       13      5620         0
## 57         57   x57 numeric        2      5620         0
## 58         58   x58 numeric       11      5620         0
## 59         59   x59 numeric       17      5620         0
## 60         60   x60 numeric       17      5620         0
## 61         61   x61 numeric       17      5620         0
## 62         62   x62 numeric       17      5620         0
## 63         63   x63 numeric       17      5620         0
## 64         64   x64 numeric       17      5620         0
## 65         65 digit numeric       10      5620         0
```

From the above output, it can be observed that we have no missing values in the data set

```
# removing target variable
df1 <- data.matrix(traindat[,-c(33,65)]) # Train
df2 <- data.matrix(testdat[,-c(33,65)]) # Test
```

```
# Remove the Unary variables
new_traindat <- df1[,apply(df1, 2, var, na.rm=TRUE) != 0] # Train
new_testdat <- df2[,apply(df2, 2, var, na.rm=TRUE) != 0] # Test
dim(new_traindat);dim(new_testdat)
```

```
## [1] 3823   61
```

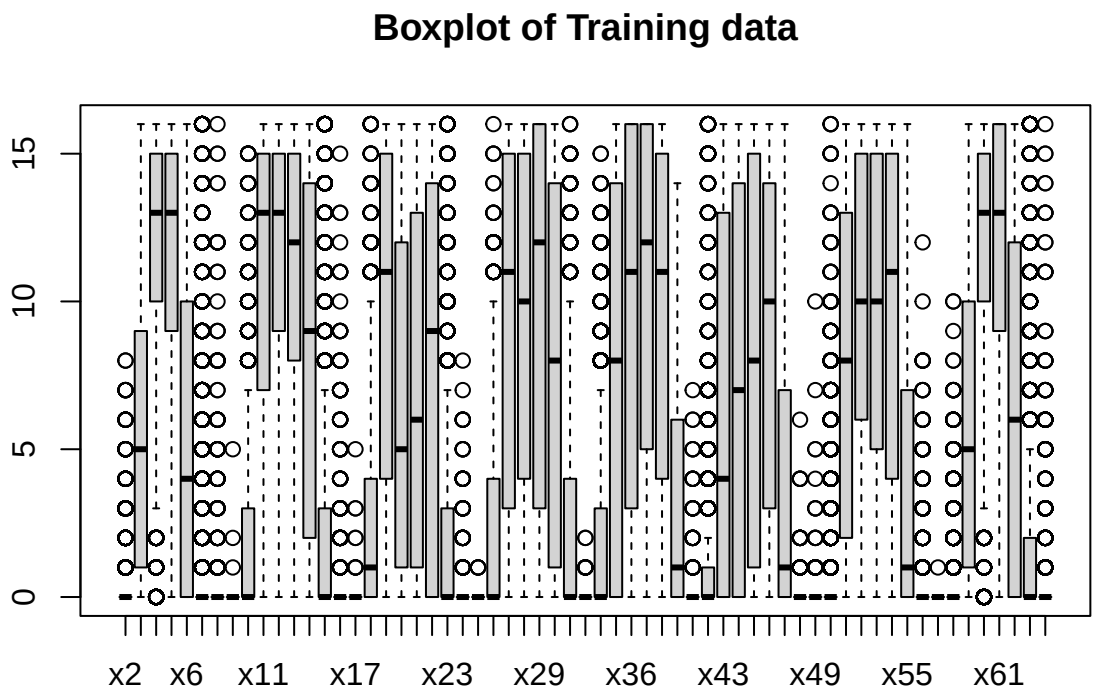
```
## [1] 1797   61
```

I have removed all the columns with unitary values from both the testing and training data sets. After removing the columns with unitary values, we have 3823 observations 61 variables for the training set. Likewise, We have 1797 observations and 61 variables for the testing set.

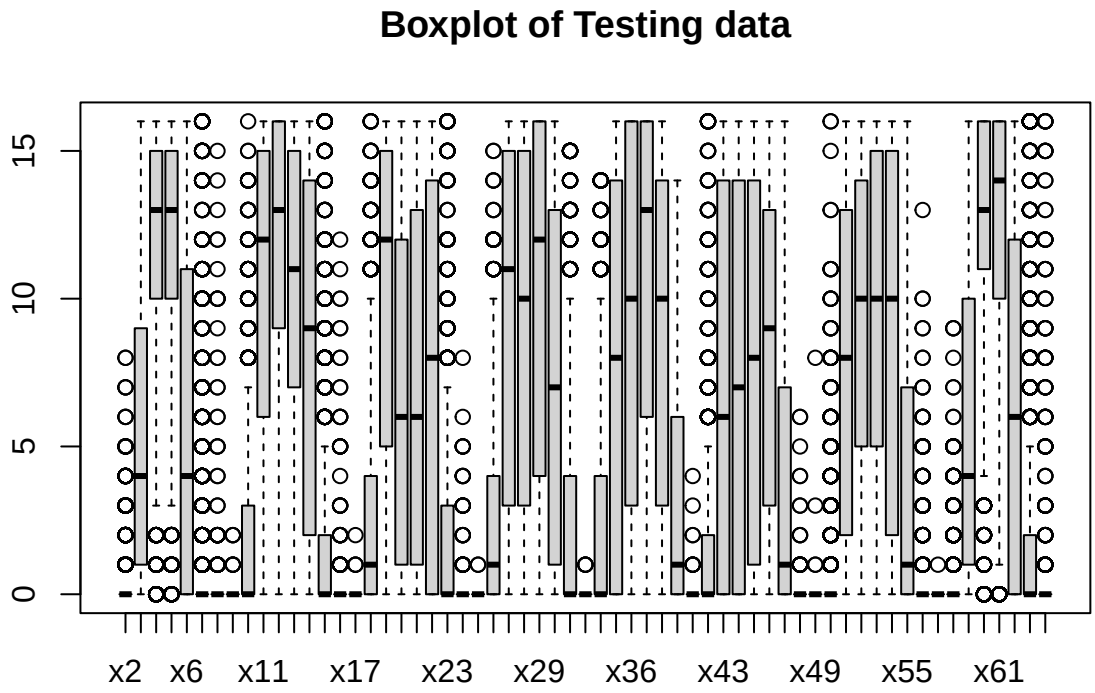
(1b) Excluding the target variables, run the ordinary principal components analysis (PCA) with the training set. Output the scree plot of the variances (i.e., eigenvalues) of the principal components. Make a scatter plot of the first two PCs and show the target class variable (i.e., digit number) with different symbols and colors. Recall that this also corresponds to a multidimensional scaling (MDS) analysis of data. Interpret results. • Be careful with the normalization or standardization of the data before performing PCA. Make parallel boxplots of the attributes and inspect for necessity of data normalization.

BOXPLOTS OF TRAINING AND TESTING DATA SETS

```
boxplot(new_traindat, main="Boxplot of Training data")
```



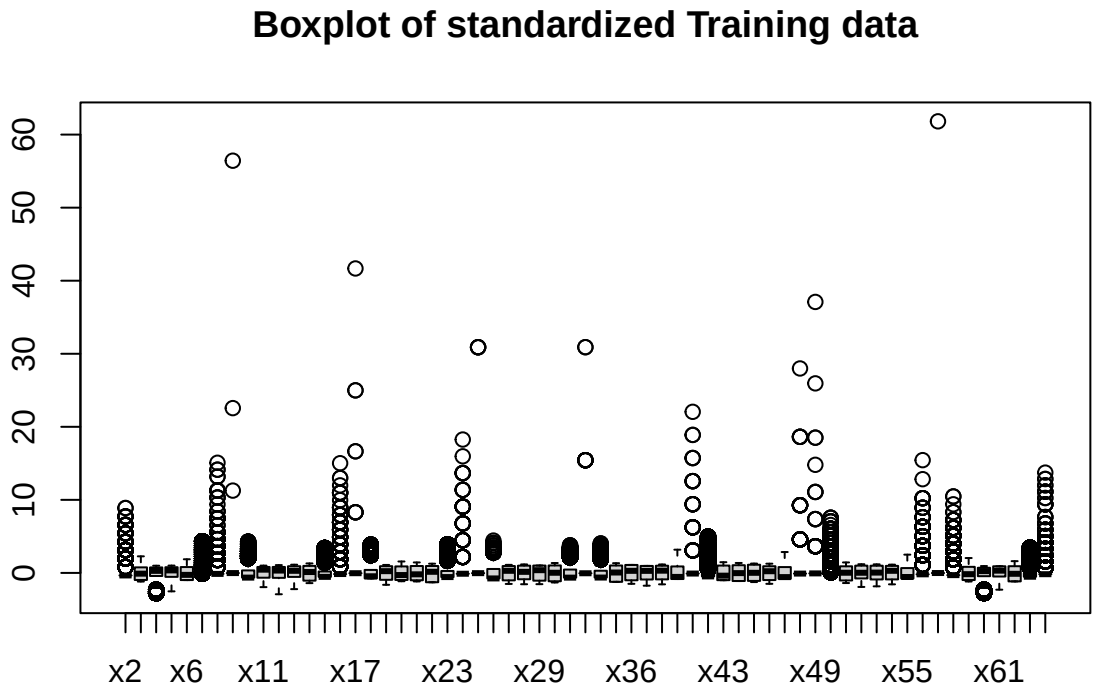

```
boxplot(new_testdat, main="Boxplot of Testing data")
```



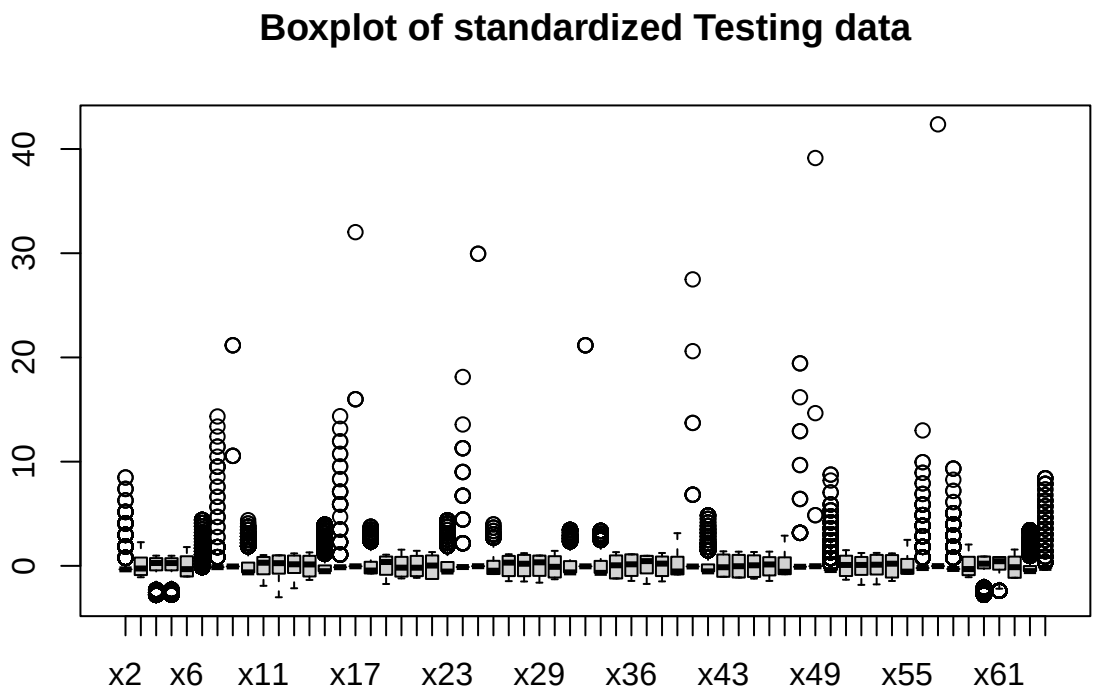
Clearly, from the above box plots, it is necessary that we standardize the data before the principal component analysis

STANDARDIZE THE TRAINING AND TESTING DATA

```
scaled_new_traindat <- data.frame(apply(new_traindat, 2, scale, center=T, scale=T))

boxplot(scaled_new_traindat, main="Boxplot of standardized Training data")
```




```
scaled_new_testdat <- data.frame(apply(new_testdat, 2, scale, center=T, scale=T))

boxplot(scaled_new_testdat, main="Boxplot of standardized Testing data")
```



From the boxplots above, we can observe that the standardized data sets seem normal for the principal component analysis

ORDINARY PRINCIPAL COMPONENT ANALYSUS ON THE TRAINING DATA SET

```
pca.train <- prcomp(scaled_new_traindat, scale=FALSE, retx=TRUE,center=FALSE);
```

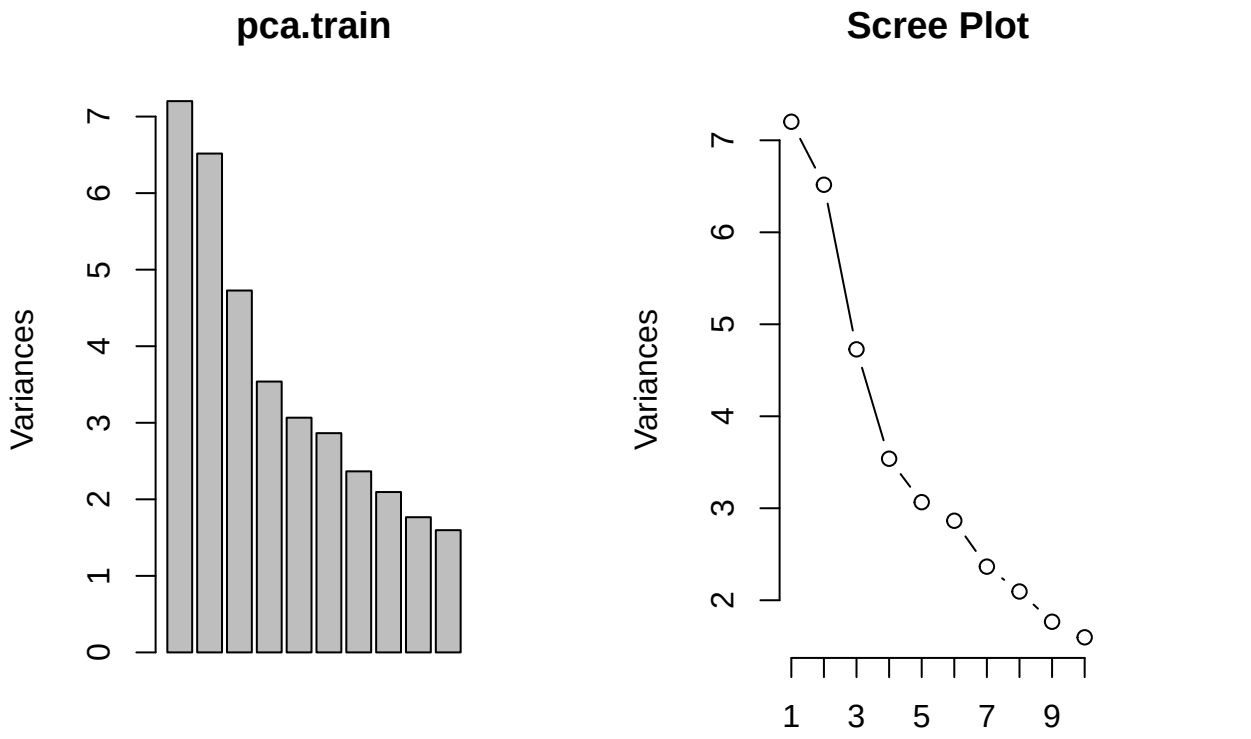
OBTAINING EIGENVALUES

```
eigen(cov(scaled_new_traindat), only.values = T)$values
```

```
##  [1] 7.20179133 6.51696036 4.72764717 3.53828965 3.06630997 2.86412950
##  [7] 2.36531710 2.09493689 1.76627760 1.59705595 1.49407201 1.48721245
## [13] 1.37760221 1.29374146 1.17282747 1.14859807 1.12561870 1.02565153
## [19] 0.99925276 0.91357286 0.87165403 0.85442179 0.70748871 0.68849032
## [25] 0.64403360 0.61343679 0.57631445 0.56141407 0.53047561 0.49360755
## [31] 0.45218590 0.43610867 0.40015798 0.38867786 0.37555770 0.36762791
## [37] 0.32845474 0.29940488 0.27970539 0.27017947 0.26224683 0.24059564
## [43] 0.21433266 0.20658560 0.19220898 0.19048782 0.17229587 0.16675202
## [49] 0.15946094 0.15080540 0.14628710 0.13536792 0.12890361 0.11961022
## [55] 0.11202967 0.10121850 0.09530269 0.08829583 0.07680299 0.06610966
## [61] 0.05803759
```

PLOTING THE VARIANCES USING SCREE AND BAR CHART

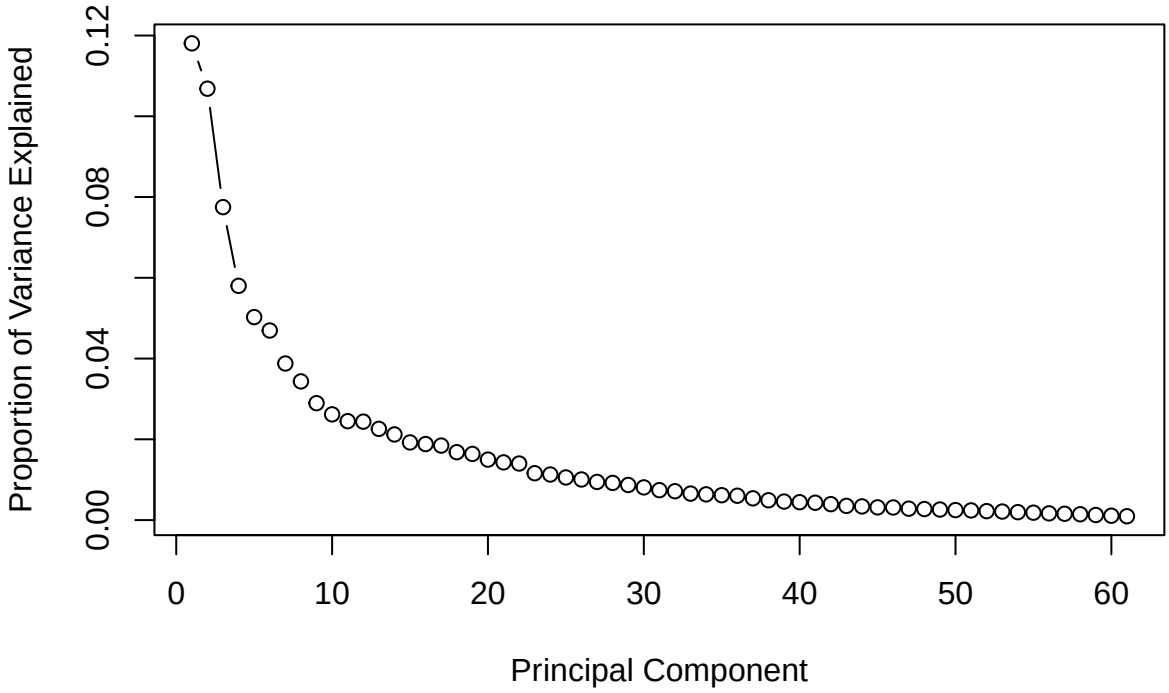
```
par(mfrow=c(1,2), mar=rep(4,4))
plot(pca.train)
screeplot(pca.train, type="lines", main="Scree Plot")
```



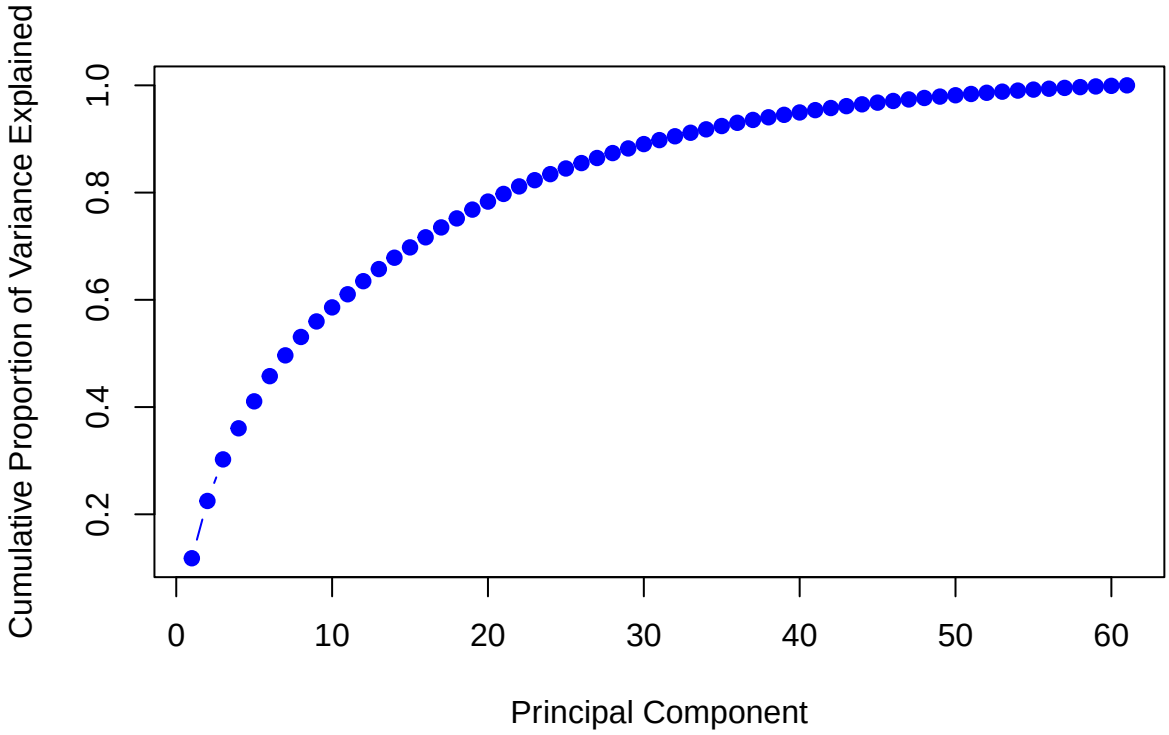
From the above bar charts and scree plot, we can observe that the first two principal components explains the most variations in the training data set

CUMULATIVE PROPORTIONS OF VARIATION EXPLAINED

```
sd.pc <- pca.train$sdev
var.pc <- sd.pc^2
prop.pc <- var.pc/sum(var.pc)
# NONCUMULATIVE
plot(prop.pc, xlab = "Principal Component",
    ylab = "Proportion of Variance Explained", type = "b")
```



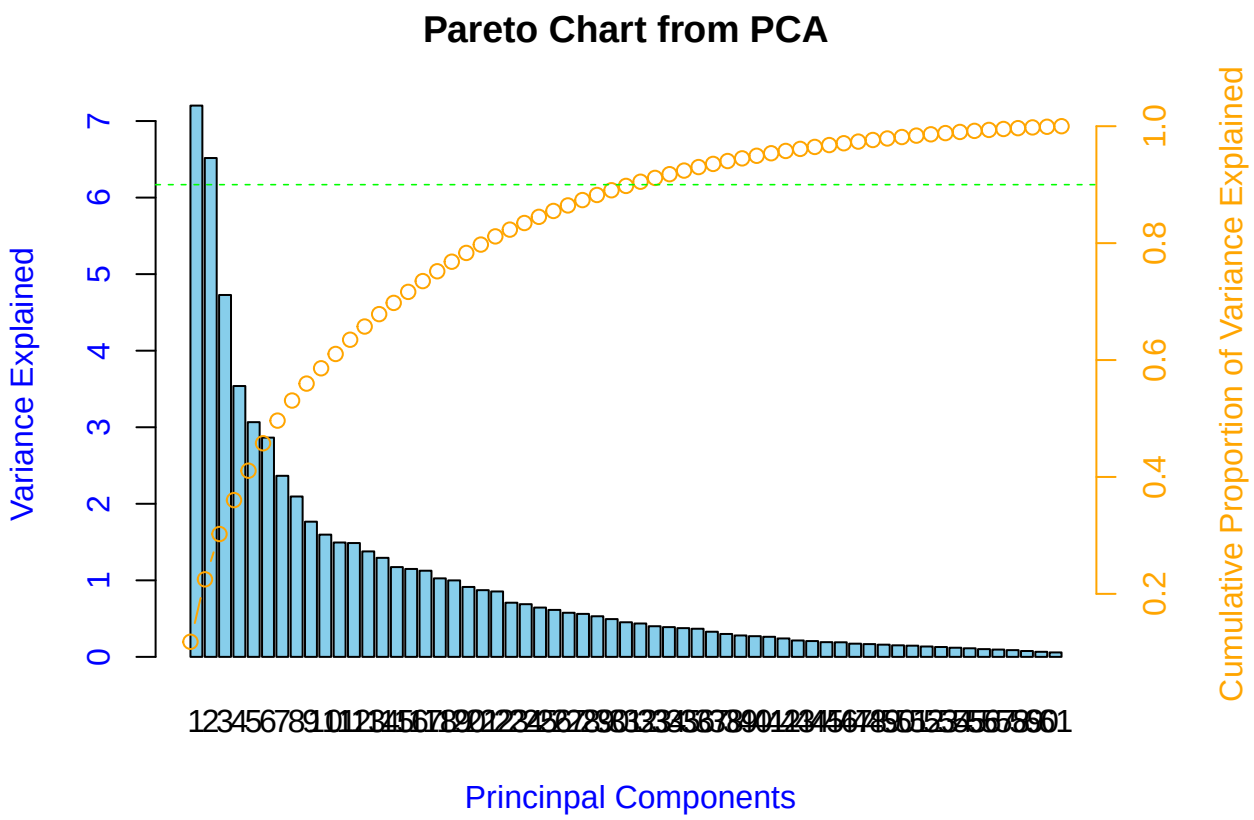

```
plot(cumsum(prop.pc), xlab = "Principal Component", col="blue",
              ylab = "Cumulative Proportion of Variance Explained",
              type = "b", pch=19)
```



We further illustrate the principal components using the cumulative proportion of the variances explained. We can also observe here that the first two principal components explained the most variations.

THE PARETO PLOT

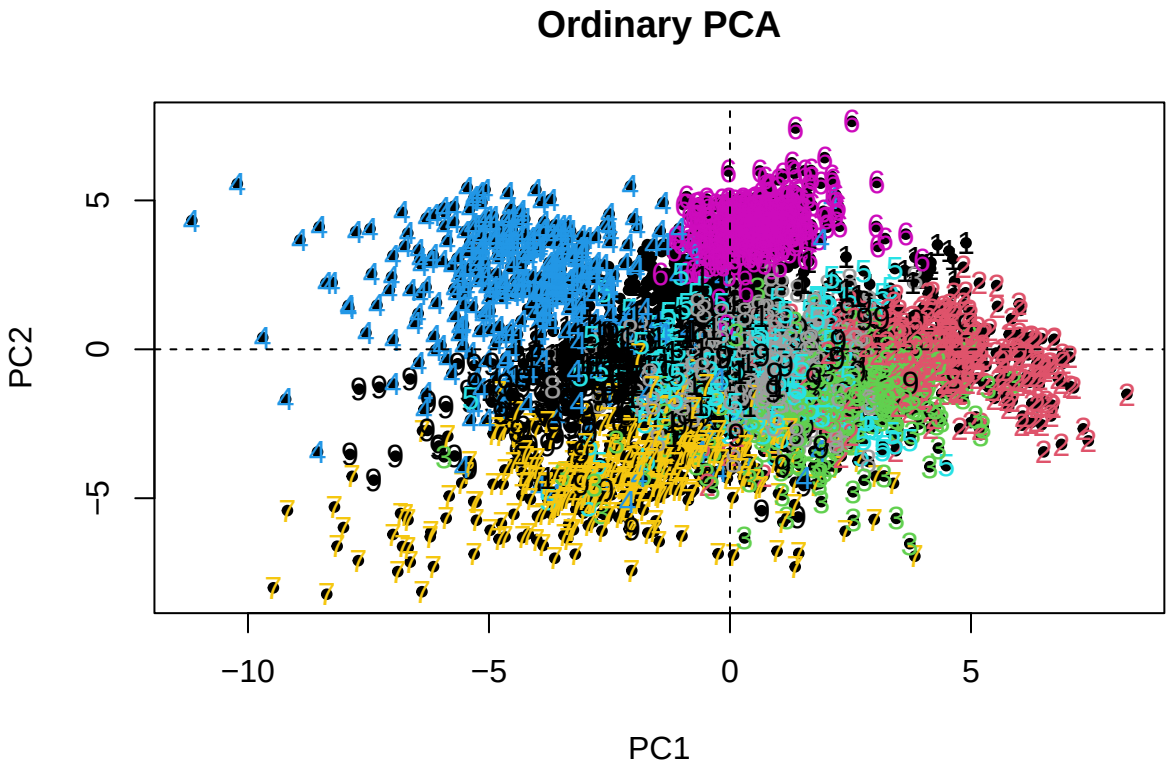
```
 #PUTS VARIANCE AND CUMULATIVE PROPORTIONS TOGETHER -- THE PARETO PLOT

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')
```



From the pareto chart, it can be observed that the first two principal components explains over 90% of the total variations in the dataset

PLOT OF THE FIRST TWO PCs

```
plot(pca.train$x[,1:2], pch=20, main="Ordinary PCA")
text(pca.train$x[,1:2],labels=traindat$digit, col=traindat$digit)
abline(v=0, lty=2)
abline(h=0, lty=2)
```



We plot PC1 against PC2 with a scatter plot. From the plot above, digits closer to one another are more similar than those further away. Also, from the plot we can observe that though the two components hold some information it is unable to set the digits apart very well

(1c) Run kernel PCA on the input variables only. Explain your choice of kernel function and the choice of parameters involved. Output the scree plot of the variances (i.e., eigenvalues) of the resultant principal components. Plot the first two PCs with scatted points and show the target class variable with different symbols and colors. Compare the kPCA results with the PCA results and comment with interpretations.

KERNEL PRINCIPAL COMPONENT ANALYSIS USING GAUSSIAN RADIAL BASIS KERNEL FUNCTION

```
library(kernlab)

kpc<- kpca(~., data=scaled_new_traindat, kernel="rbfdot",
    kpar=list(sigma=0.02), features=35);
```

From the above, we perform the kernel principal component analysis using the Gaussian radial basis kernel family (rbfdot) with a chosen sigma of 0.2 and 35 principal components.

OBTAINING THE EIGENVALUES OF THE KERNEL PRINCIPAL COMPONENTS

```
eig(kpc)        # returns the eigenvalues
```

```
##      Comp.1      Comp.2      Comp.3      Comp.4      Comp.5      Comp.6
## 0.045765453 0.041596708 0.035691196 0.029532836 0.025804486 0.022385444
##      Comp.7      Comp.8      Comp.9     Comp.10     Comp.11     Comp.12
```

```
## 0.020061733 0.014998013 0.014650549 0.012660528 0.012487758 0.011212081
##     Comp.13     Comp.14     Comp.15     Comp.16     Comp.17     Comp.18
## 0.010560911 0.009314247 0.008510327 0.008035757 0.007693795 0.007335730
##     Comp.19     Comp.20     Comp.21     Comp.22     Comp.23     Comp.24
## 0.006301117 0.006250427 0.005773536 0.005396386 0.005290807 0.005062650
##     Comp.25     Comp.26     Comp.27     Comp.28     Comp.29     Comp.30
## 0.004917508 0.004713774 0.004617841 0.004528420 0.004277059 0.004047205
##     Comp.31     Comp.32     Comp.33     Comp.34     Comp.35
## 0.004002562 0.003851532 0.003702313 0.003540216 0.003430265
```

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

```
## Gaussian Radial Basis kernel function.
##  Hyperparameter : sigma =  0.02
```

OBTATINING THE PRINCIPAL COMPNENT VECTORS

```
PCV <- pcv(kpc)
dim(PCV);head(PCV)
```

```
## [1] 3823   35
```

```
##             [,1]          [,2]        [,3]         [,4]        [,5]        [,6]
## [1,] -0.15377681 -0.0210302110  0.07401273 -0.002588424 -0.15766563 -0.06129563
## [2,] -0.13316465  0.0037432672  0.06845996  0.087264717 -0.11218842 -0.13114761
## [3,]  0.09570385  0.0153768249  0.13563393  0.010412746 -0.14249052  0.22167158
## [4,]  0.02260643  0.0752750714  0.12147832 -0.032030868  0.11793969 -0.10390653
## [5,] -0.11206548  0.0115997368 -0.09727151  0.028578037  0.02663608  0.20578261
## [6,]  0.02278909 -0.0001427619 -0.03030259  0.144154511 -0.00250532 -0.11269760
##             [,7]        [,8]          [,9]       [,10]        [,11]       [,12]
## [1,] -0.07582866  0.10589364  0.0002964134 -0.07128271  0.001667696 -0.1708688
## [2,]  0.03478120  0.05823238 -0.0263797391  0.09782841  0.033592349 -0.2142274
## [3,]  0.23155259  0.03140502  0.1328871845 -0.12546343  0.066143078 -0.3460351
## [4,] -0.14807000 -0.18057407 -0.1187910864 -0.02040819  0.062778032  0.0461844
## [5,] -0.09496439 -0.01077657  0.0230569225  0.03263365  0.115701197 -0.1918324
## [6,] -0.01238641 -0.21437392 -0.0454048480 -0.01066358 -0.007374907  0.1059449
##            [,13]       [,14]       [,15]       [,16]        [,17]       [,18]
## [1,]  0.09082621  0.02207833  0.01234199  0.14667457  0.128935031 -0.19379824
## [2,]  0.17007562  0.06449209  0.30520245  0.19311296 -0.183456267 -0.17530776
## [3,]  0.08367048  0.06119612 -0.26104564 -0.04627755  0.006140849  0.27486238
## [4,] -0.06084380 -0.12573223  0.17972315 -0.13885907  0.403044453  0.05070018
## [5,] -0.07260800 -0.43175143  0.11189319  0.08399271 -0.009879835 -0.06207247
## [6,]  0.11808361  0.14382334 -0.15352559  0.44308355 -0.221770091  0.64966223
##            [,19]       [,20]       [,21]       [,22]       [,23]       [,24]
## [1,]  0.16370786 -0.13281356  0.24009823 -0.37443240  0.11522068  0.28082652
## [2,] -0.02368080 -0.02139025  0.11852891 -0.45571934  0.02396476  0.03683011
## [3,]  0.04094780  0.30592344  0.10573454  0.06947842  0.01431223 -0.06425728
## [4,] -0.15730431  0.12295945 -0.04799753 -0.16303425 -0.43340600 -0.11503891
## [5,]  0.03093163  0.20294461 -0.06114783 -0.16355212  0.09237306  0.20416548
## [6,] -0.17000552 -0.26042520  0.34180757 -0.05323068  0.12007913  0.13355167
##            [,25]       [,26]       [,27]       [,28]       [,29]       [,30]
## [1,] -0.0572758 -0.0692656  0.11813644  0.03638899 -0.16273140  0.11561867
```

```
## [2,]  0.1831370  0.3332759  0.33173143  0.12858770 -0.25328998  0.30744913
## [3,]  0.3795642 -0.4165232  0.40063120 -0.23358437  0.03781283 -0.01387292
## [4,]  0.2970328  0.3275094 -0.17815082 -0.25907118  0.27001007  0.08246498
## [5,] -0.4043934  0.1241914  0.07161319  0.10302336  0.33313514 -0.24960766
## [6,] -0.5174760 -0.2252214  0.25539888 -0.51350174  0.38686158 -0.12769768
##            [,31]       [,32]       [,33]       [,34]       [,35]
## [1,] -0.00859139 -0.48203301 -0.03962543  0.14485765  0.09159488
## [2,]  0.27877066  0.13679216 -0.13740537 -0.08143077 -0.69873768
## [3,]  0.28950673 -0.20637243 -0.04433273  0.06595394  0.13778203
## [4,]  0.27303414  0.10875458 -0.16981821 -0.64911523  0.53481498
## [5,]  0.01518385 -0.08581967  0.24618294 -0.07237327  0.39618133
## [6,]  0.05140023 -0.43602192 -0.63593244 -0.50801579  0.25609868
```

THE DATA PROJECTED IN THE KERNEL PCA SPACE

```
PC <- rotated(kpc)
dim(PC); head(PC);
```

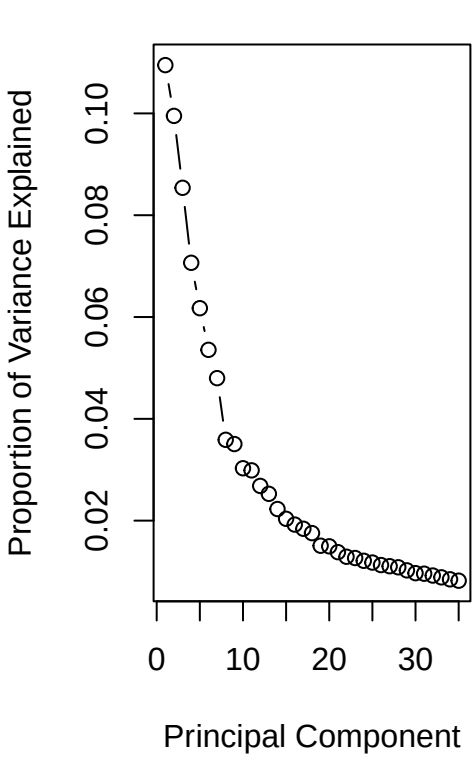
```
## [1] 3823   35
```

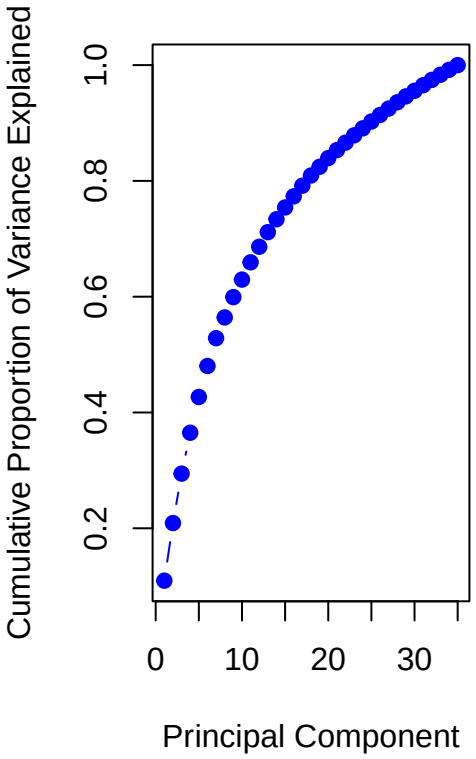
```
##        [,1]       [,2]       [,3]       [,4]        [,5]       [,6]        [,7]
## 1 -26.90499 -3.3443128  10.098847 -0.2922435 -15.5538019  -5.245653  -5.8157552
## 2 -23.29866  0.5952701   9.341186  9.8525386 -11.0674494 -11.223554   2.6675788
## 3  16.74447  2.4452875  18.506903  1.1756410 -14.0567688  18.970554  17.7591582
## 4   3.95525 11.9705593  16.575406 -3.6164142  11.6348153  -8.892275 -11.3563770
## 5 -19.60712  1.8446391 -13.272449  3.2265756   2.6276641  17.610783  -7.2833889
## 6   3.98721 -0.0227026  -4.134712 16.2756258  -0.2471512  -9.644610  -0.9499885
##          [,8]        [,9]      [,10]       [,11]      [,12]     [,13]
## 1   6.0716663  0.01660183 -3.4501684  0.07961698  -7.324084  3.667050
## 2   3.3388933 -1.47750412  4.7350125  1.60372216  -9.182597  6.866695
## 3   1.8006825  7.44288492 -6.0725805  3.15771670 -14.832373  3.378142
## 4 -10.3536480 -6.65337586 -0.9877809  2.99706702   1.979638 -2.456530
## 5  -0.6179005  1.29139632  1.5795079  5.52365583  -8.222661 -2.931502
## 6 -12.2916437 -2.54308239 -0.5161301 -0.35208323   4.541198  4.767550
##         [,14]      [,15]     [,16]      [,17]     [,18]      [,19]      [,20]
## 1   0.7861731  0.4015462  4.505945  3.7924148 -5.434974  3.9435864 -3.1736307
## 2   2.2964580  9.9297556  5.932565 -5.3960685 -4.916418 -0.5704509 -0.5111282
## 3   2.1790937 -8.4931146 -1.421679  0.1806231  7.708377  0.9863984  7.3101574
## 4  -4.4771191  5.8472891 -4.265848 11.8548987  1.421861 -3.7893303  2.9381630
## 5 -15.3739630  3.6404428  2.580315 -0.2905993 -1.740791  0.7451173  4.8494390
## 6   5.1213140 -4.9949519 13.611837 -6.5230075 18.219450 -4.0952920 -6.2229596
##       [,21]     [,22]      [,23]      [,24]     [,25]     [,26]     [,27]
## 1  5.299502 -7.724683  2.3305407  5.4352595 -1.076764 -1.248219  2.085582
## 2  2.616197 -9.401664  0.4847293  0.7128286  3.442908  6.005885  5.856389
## 3  2.333797  1.433366  0.2894901 -1.2436681  7.135668 -7.506064  7.072746
## 4 -1.059412 -3.363459 -8.7663978 -2.2265218  5.584107  5.901969 -3.145076
## 5 -1.349669 -3.374143  1.8684073  3.9515226 -7.602448  2.238024  1.264260
## 6  7.544454 -1.098169  2.4288113  2.5848271 -9.728360 -4.058662  4.508814
##        [,28]     [,29]      [,30]      [,31]     [,32]      [,33]      [,34]
## 1  0.6299717 -2.660853  1.7889057 -0.1314637 -7.097651 -0.5608560  1.9605391
## 2  2.2261297 -4.141594  4.7569957  4.2656917  2.014184 -1.9448277 -1.1021041
## 3 -4.0438479  0.618285 -0.2146483  4.4299728 -3.038712 -0.6274829  0.8926369
## 4 -4.4850794  4.414988  1.2759364  4.1779126  1.601347 -2.4035973 -8.7852856
```

```
## 5  1.7835560  5.447158 -3.8620456  0.2323402 -1.263644  3.4844592 -0.9795176
## 6 -8.8898199  6.325650 -1.9757978  0.7865158 -6.420165 -9.0009514 -6.8756110
##       [,35]
## 1  1.201166
## 2 -9.163178
## 3  1.806860
## 4  7.013511
## 5  5.195483
## 6  3.358453
```

COMPUTATION OF THE NONCUMULATIVE CUMULATIVE PROPORTIONS OF VARIATION EXPLAINED

```
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)
```

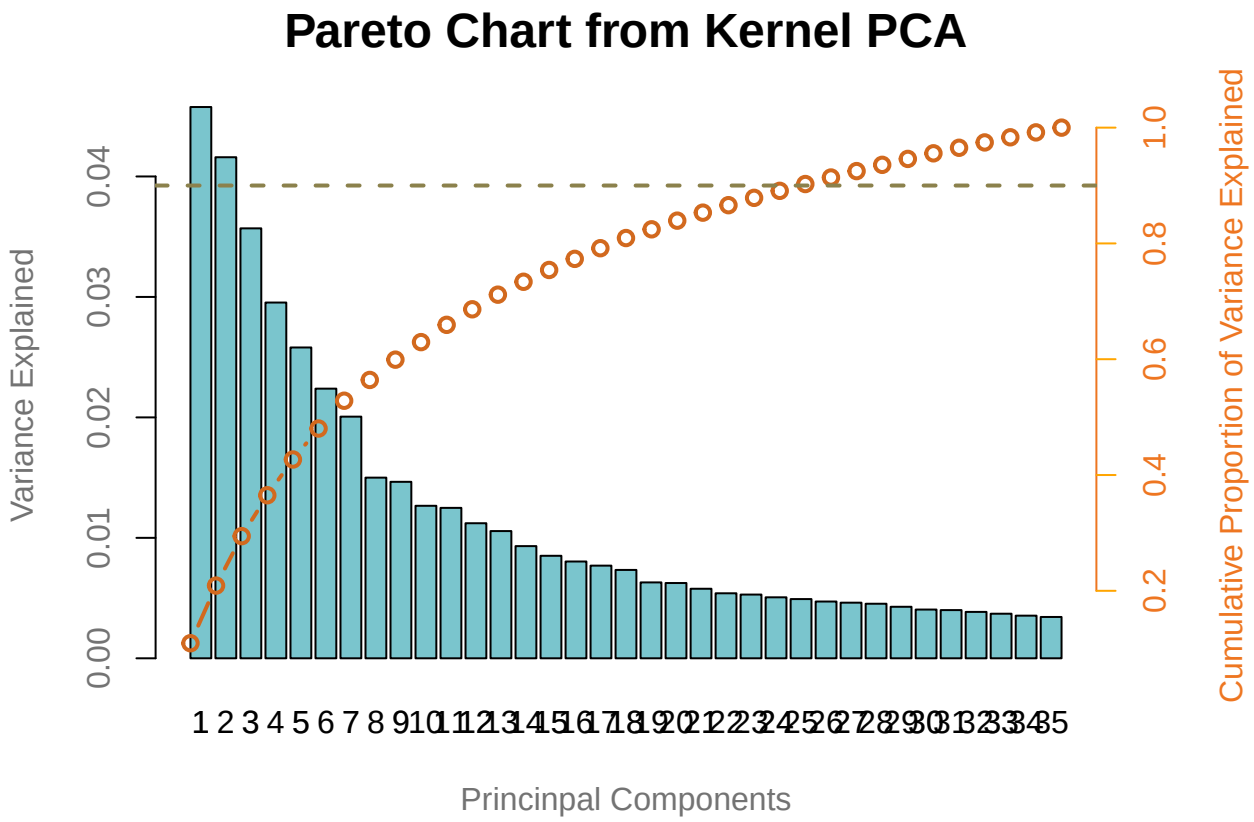




From the above plots, we can also observe that the first two kernel principal component explains a significant variations in the data set

PARETO CHART FROM THE KERNEL PRINCIPAL COMPONENT ANALYSIS

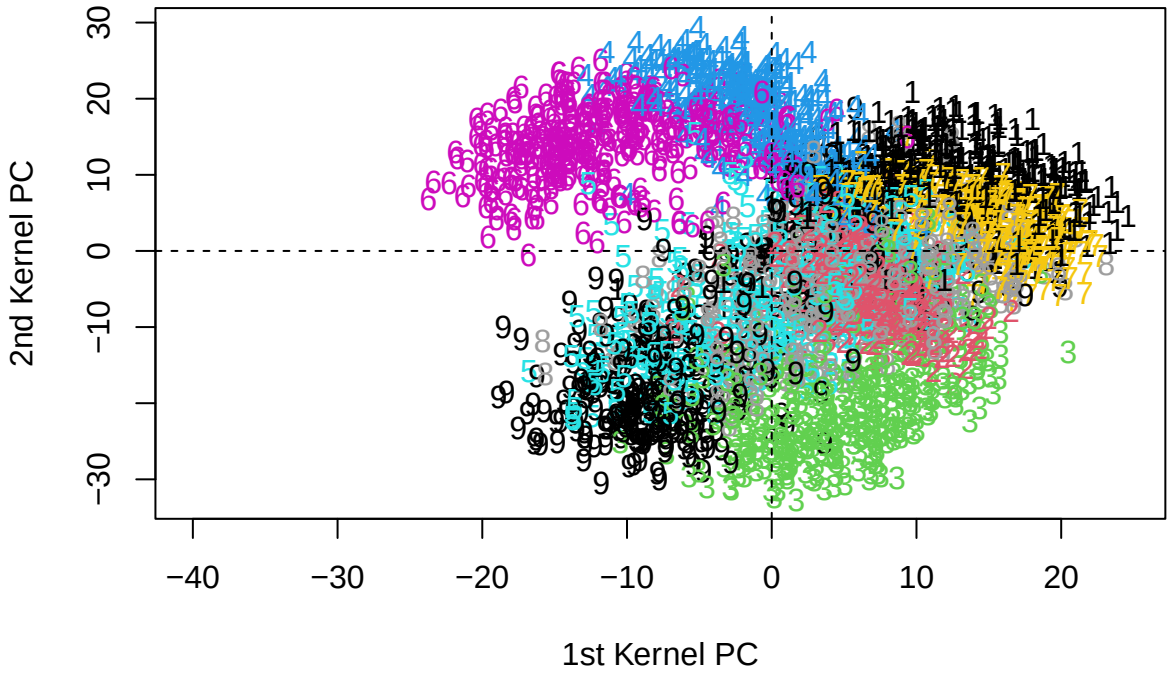
```
#  PUTS VARIANCE AND CUMULATIVE PROPORTIONS TOGETHER -- THE PARETO PLOT
par(mar=c(4, 4, 4, 4), mfrow=c(1,1))
bar <- barplot(var.pc, ylab="Variance Explained", col="cadetblue3",
    xlab="Princinpal Components", col.axis="gray45", col.lab="gray45")
mtext(1:length(var.pc),side=1, line=1,at=bar,col="black")
# mtext(" ",side=1,line=3,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)
```



We can observe from the above pareto chart that the first two principal components explains over 90% of the total variations in the data set.

```
# Plot THE DATA PROJECTION ON THE KERNEL PCS
plot(PC[, 1:2],col=traindat$digit, pch="",
xlab="1st Kernel PC", ylab="2nd Kernel PC")
text(PC[,1:2], labels=traindat$digit, col=traindat$digit)
```

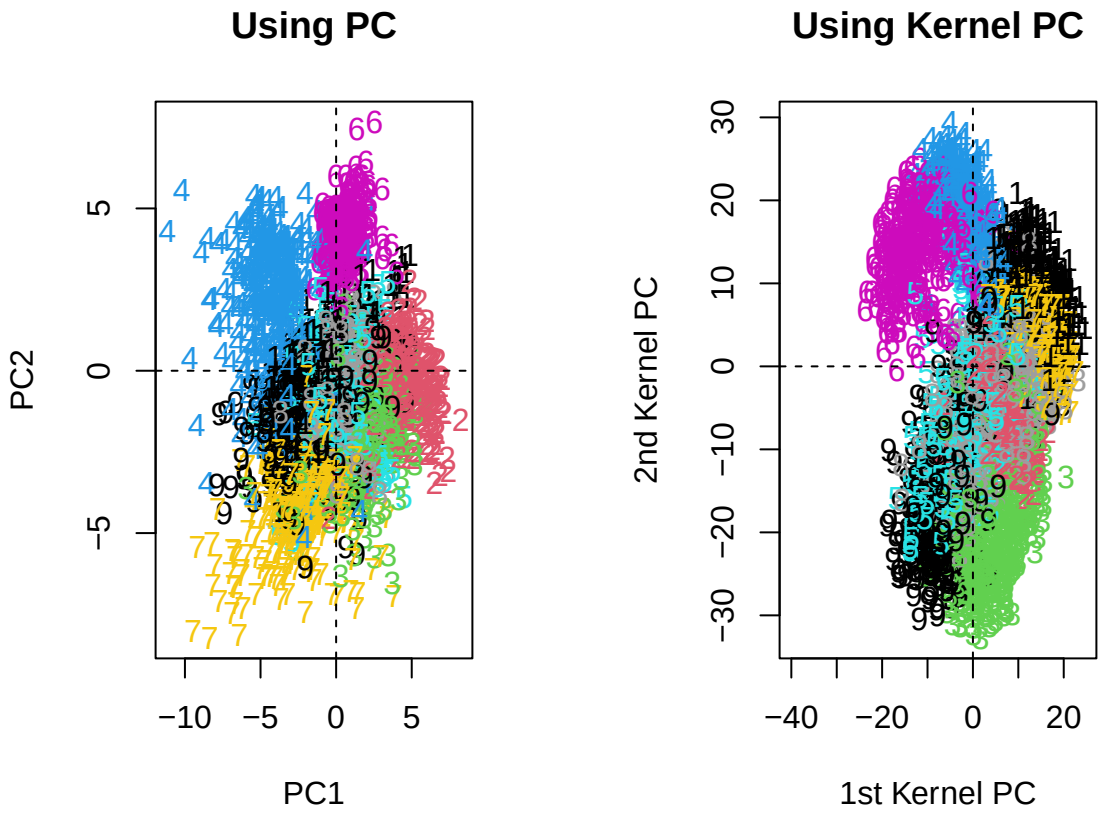
```
abline(v=0, lty=2)
abline(h=0, lty=2)
```



From the above plot, we plot the first kernel principal component against the second kernel principal component and we can also observe from here that though the kernel principal components contain some information it is unable to properly set the hand digits apart

COMPARING ORDINARY PCA AND KPCA ON THE TRAIN DATA

```
par(mfrow=c(1,2), mar=rep(4,4))
#Scatter plot of the first two PCs
plot(pca.train$x[,1:2], pch="", main="Using PC")
text(pca.train$x[,1:2], labels=traindat$digit, col=traindat$digit)
abline(v=0, lty=2)
abline(h=0, lty=2)
# Plot THE DATA PROJECTION ON THE KERNEL PCS
plot(PC[, 1:2],col=traindat$digit, pch="",main="Using Kernel PC" ,
xlab="1st Kernel PC", ylab="2nd Kernel PC")
text(PC[,1:2], labels=traindat$digit, col=traindat$digit)
abline(v=0, lty=2)
abline(h=0, lty=2)
```

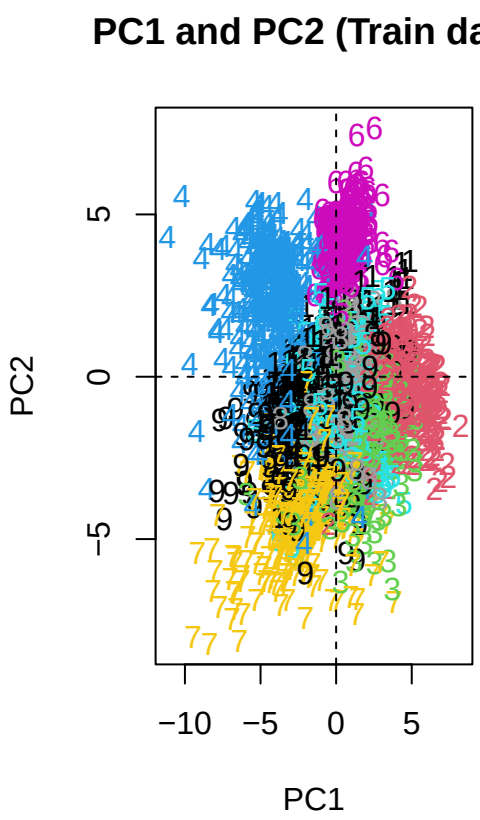


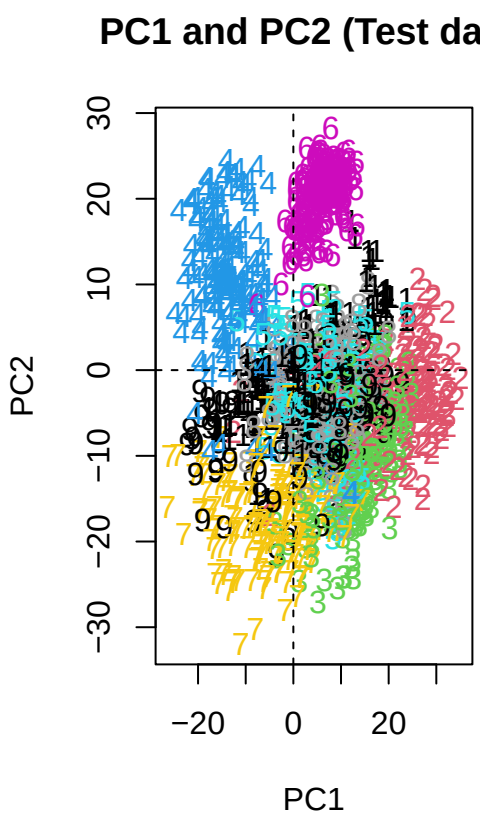
From the above plots, it can be observe that the kernel principal component(kpca) is able to cluster hand-digits better than the ordinary principal component(pc)

(1d) Apply both the PCA and kPCA results learned from the training data to the test set optdigits.tes, which can be simply done by using the predict() function. Obtain the first two principal components in each case and make similar plots as Part (b) & (c) and compare.

APPLY PCA AND KPCA TO TEST DATA

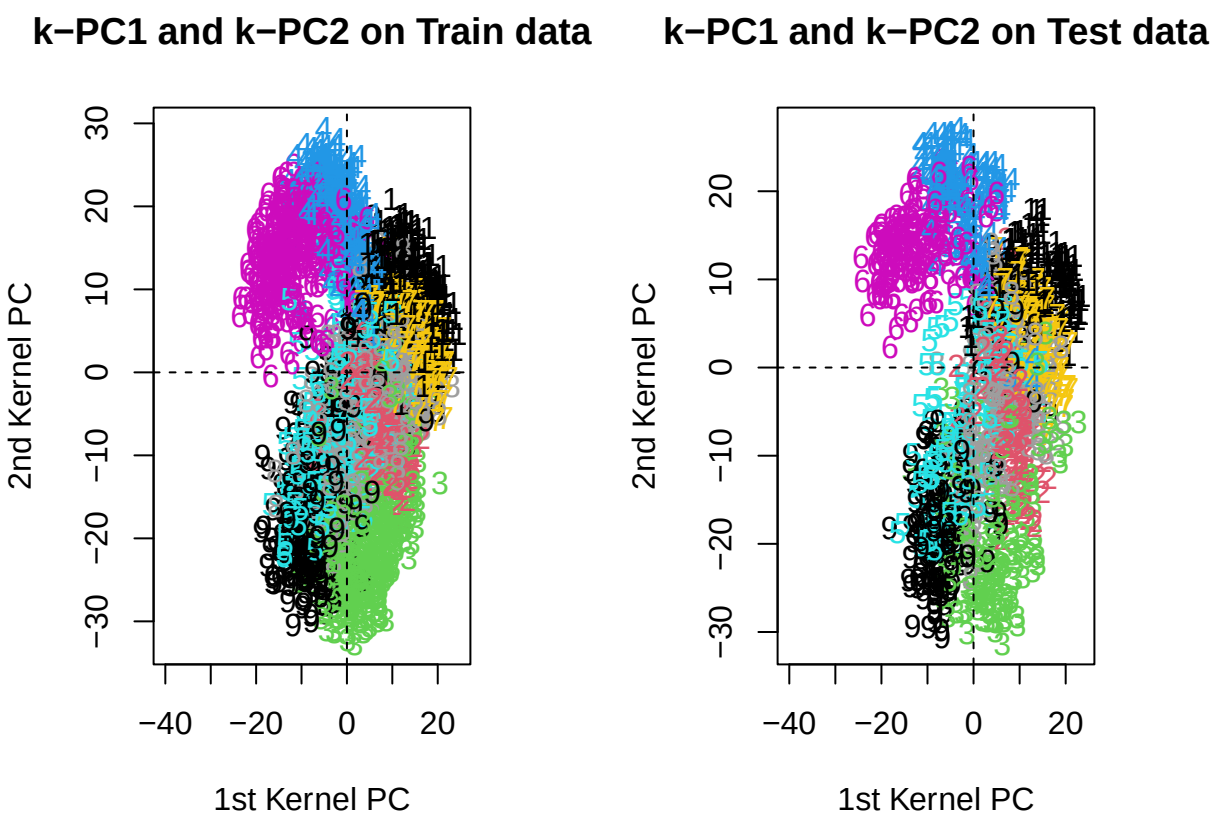
```
# PCA
pred_pca <- predict(pca.train, new_testdat);
par(mfrow=c(1,2), mar=rep(4,4))
#Scatter plot of the first two PCs
plot(pca.train$x[,1:2], pch="", main="PC1 and PC2 (Train data)")
text(pca.train$x[,1:2], labels=traindat$digit, col=traindat$digit)
abline(v=0, lty=2)
abline(h=0, lty=2)
#Scatter plot of the first two PCs (Predicted)
plot(pred_pca[,1:2], pch="", main="PC1 and PC2 (Test data)")
text(pred_pca[,1:2], labels=testdat$digit, col=testdat$digit)
abline(v=0, lty=2)
abline(h=0, lty=2)
```





From the two PC plots above, we can observe that the two PCA looks almost similar. Which means PCA predicted the test data somewhat well.

```
# KPCA
pred_kpca <- predict(kpc, scaled_new_testdat );


par(mfrow=c(1,2), mar=rep(4,4))
#Scatter plot of the first two KERNEL PCS
plot(PC[, 1:2],col=traindat$digit, main="k-PC1 and k-PC2 on Train data", pch="",
xlab="1st Kernel PC", ylab="2nd Kernel PC")
text(PC[,1:2], labels=traindat$digit, col=traindat$digit)
abline(v=0, lty=2)
abline(h=0, lty=2)
#Scatter plot of the first two KERNEL PCS Predicted
plot(pred_kpca[, 1:2],col=testdat$digit,main="k-PC1 and k-PC2 on Test data", pch="",
xlab="1st Kernel PC", ylab="2nd Kernel PC")
text(pred_kpca[,1:2], labels=testdat$digit, col=testdat$digit)
abline(v=0, lty=2)
abline(h=0, lty=2)
```



Similarly, the plots above seem similar which means the kernel pc has been able to predict the test data fairly well though we have some dispersion with the prediction on the test data.

ASSOCIATION RULES

(2a) First read the data into R as transaction data type. This can be done using R function read.transactions() in the arules package:

```
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 QUOTES
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}
```

(2b) Set up the parameters in R function arules appropriately with your own choices and then perform frequent itemsets and association rule analysis.

THE TOP 10 ITEMS

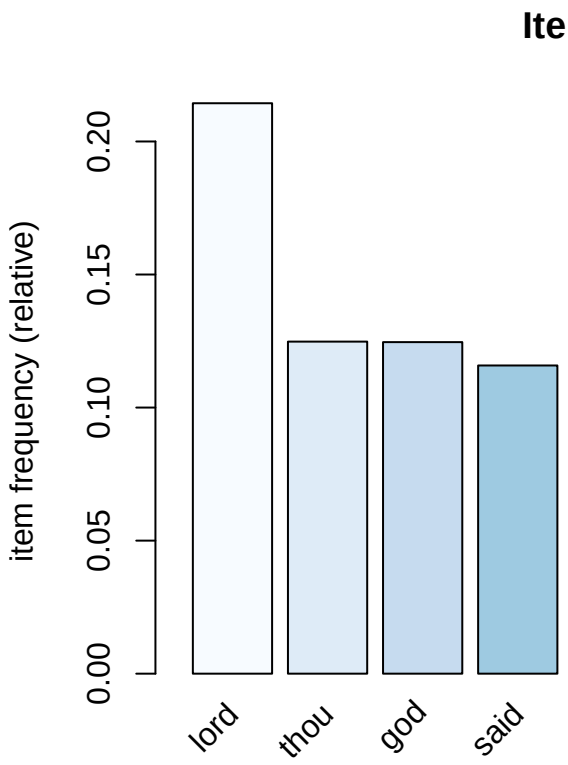
```
item.freq <- itemFrequency(dat, type = "relative")
item.freq <- sort(item.freq, decreasing = TRUE)
item.freq[1:10]
```

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

The above output shows the top 10 items

```
#tem.freq0 <- sort(item.freq[1:10], decreasing = TRUE)
#tem.freq0
```

```
itemFrequencyPlot(dat,support=0.1, type = c("relative"), cex=1,col=blues9,topN=10, main="Item Frequency
```



The above chart shows the top item frequency using a chosen support of 0.1. It can be observed that lord has the highest frequency followed by thou,god and said.

RULES

```
rules <- apriori(dat, parameter = list(support = 0.005, 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   0.005      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: 155
##
## set item appearances ...[0 item(s)] done [0.00s].
## set transactions ...[12767 item(s), 31101 transaction(s)] done [0.12s].
## sorting and recoding items ... [451 item(s)] done [0.00s].
## creating transaction tree ... done [0.01s].
## checking subsets of size 1 2 3 4 5

## Warning in apriori(dat, parameter = list(support = 0.005, confidence = 0.5, :
## Mining stopped (maxlen reached). Only patterns up to a length of 5 returned!

##  done [0.01s].
## writing ... [107 rule(s)] done [0.00s].
## creating S4 object  ... done [0.00s].
```

```
summary(rules)
```

```
## set of 107 rules
##
## rule length distribution (lhs + rhs):sizes
##  2  3  4
## 26 55 26
##
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##       2       3       3       3       3       4
##
## summary of quality measures:
##     support           confidence        coverage             lift
##  Min.   :0.005016   Min.   :0.5011   Min.   :0.005080   Min.   : 2.343
##  1st Qu.:0.006077   1st Qu.:0.5475   1st Qu.:0.009389   1st Qu.: 3.524
##  Median :0.007331   Median :0.6176   Median :0.011350   Median : 5.462
##  Mean   :0.009194   Mean   :0.6926   Mean   :0.013777   Mean   : 7.915
##  3rd Qu.:0.010787   3rd Qu.:0.8370   3rd Qu.:0.015627   3rd Qu.: 7.923
##  Max.   :0.039002   Max.   :1.0000   Max.   :0.041478   Max.   :50.943
##      count
##  Min.   : 156.0
##  1st Qu.: 189.0
##  Median : 228.0
##  Mean   : 285.9
##  3rd Qu.: 335.5
##  Max.   :1213.0
```

```
##
## mining info:
##  data ntransactions support confidence
##   dat         31101   0.005        0.5
##                                                                                                                ca
##  apriori(data = dat, parameter = list(support = 0.005, confidence = 0.5, target = "rules", maxlen = 5
```

```
inspect(rules[1:10])
```

```
##      lhs          rhs        support     confidence coverage    lift      count
## [1]  {wilt}    => {thou}     0.006687888 1.0000000  0.006687888  8.013656 208
## [2]  {silver}  => {gold}     0.005273142 0.5815603  0.009067233 50.102788 164
## [3]  {round}   => {about}    0.009260152 0.9729730  0.009517379 50.943489 288
## [4]  {hosts}   => {lord}     0.008777853 0.9545455  0.009195846  4.452875 273
## [5]  {right}   => {hand}     0.005433909 0.5090361  0.010674898 12.225122 169
## [6]  {cut}     => {off}      0.006912961 0.6980519  0.009903219 45.418648 215
## [7]  {burnt}   => {offering} 0.006109128 0.5792683  0.010546285 35.745681 190
## [8]  {pray}    => {thee}     0.006076975 0.6176471  0.009838912  7.020995 189
## [9]  {side}    => {on}       0.007041574 0.6403509  0.010996431 11.626125 219
## [10] {another} => {one}      0.007523874 0.5735294  0.013118549 10.523503 234
```

```
summary(rules)
```

```
## set of 107 rules
##
## rule length distribution (lhs + rhs):sizes
##  2  3  4
## 26 55 26
##
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##       2       3       3       3       3       4
##
## summary of quality measures:
##     support           confidence        coverage             lift
##  Min.   :0.005016   Min.   :0.5011   Min.   :0.005080   Min.   : 2.343
##  1st Qu.:0.006077   1st Qu.:0.5475   1st Qu.:0.009389   1st Qu.: 3.524
##  Median :0.007331   Median :0.6176   Median :0.011350   Median : 5.462
##  Mean   :0.009194   Mean   :0.6926   Mean   :0.013777   Mean   : 7.915
##  3rd Qu.:0.010787   3rd Qu.:0.8370   3rd Qu.:0.015627   3rd Qu.: 7.923
##  Max.   :0.039002   Max.   :1.0000   Max.   :0.041478   Max.   :50.943
##      count
##  Min.   : 156.0
##  1st Qu.: 189.0
##  Median : 228.0
##  Mean   : 285.9
##  3rd Qu.: 335.5
##  Max.   :1213.0
##
## mining info:
##  data ntransactions support confidence
##   dat         31101   0.005        0.5
##                                                                                                                ca
##  apriori(data = dat, parameter = list(support = 0.005, confidence = 0.5, target = "rules", maxlen = 5
```

The above shows 10 rules used

```
# LOOK AT ITEMSETS
itemsets <- as(unique(generatingItemsets(rules)), "data.frame")
frequentItemsets <- itemsets[with(itemsets, order(-support,items)),]
names(frequentItemsets)[1] <- "itemset"
write.table(frequentItemsets, file = "", sep = ",", row.names = FALSE)[1:10]
```

```
## "itemset","support"
## "{shalt,thou}",0.039001961351725
## "{lord,saith}",0.0281984502106042
## "{hast,thou}",0.0268480113179641
## "{ye,your}",0.022861001253979
## "{lord,o}",0.016912639464969
## "{lord,thus}",0.0162695733256165
## "{lord,thou,thy}",0.0153049741165879
## "{shalt,thou,thy}",0.0151442075817498
## "{came,pass}",0.0148869811260088
## "{lord,moses}",0.0148548278190412
## "{saith,thus}",0.0146297546702678
## "{art,thou}",0.0143403749075592
## "{her,she}",0.0140831484518183
## "{thine,thou}",0.0139545352239478
## "{lord,saith,thus}",0.0138902286100125
## "{god,lord,thy}",0.0134079290054982
## "{thine,thy}",0.0131828558567249
## "{god,lord,thou}",0.0130542426288544
## "{shalt,thee,thou}",0.0127005562522105
## "{lord,shalt,thou}",0.0122504099546638
## "{lord,word}",0.01189672357802
## "{god,thou,thy}",0.0113501173595704
## "{god,lord,saith}",0.0110928909038295
## "{hast,thou,thy}",0.0110285842898942
## "{god,israel,lord}",0.0108999710620237
## "{answered,said}",0.0106748979132504
## "{god,lord,thee}",0.0105462846853799
## "{lord,ye,you}",0.00967814539725411
## "{god,thee,thy}",0.00964599209028649
## "{god,thee,thou}",0.00938876563454551
## "{about,round}",0.00926015240667503
## "{lord,ye,your}",0.0092279990997074
## "{ye,you,your}",0.0092279990997074
## "{god,lord,ye}",0.00890646603003119
## "{hosts,lord}",0.0087778528021607
## "{children,israel,lord}",0.00874569949519308
## "{god,lord,thou,thy}",0.00852062634641973
## "{hast,thee,thou}",0.00807048004887303
## "{god,lord,thus}",0.00784540690009967
## "{hast,lord,thou}",0.00765248705829395
## "{another,one}",0.00752387383042346
## "{thine,thou,thy}",0.00745956721648822
## "{god,lord,saith,thus}",0.00733095398861773
## "{god,lord,thee,thy}",0.00733095398861773
## "{god,saith,thus}",0.00733095398861773
```

```
## "{god,lord,your}",0.00729880068165011
## "{lord,thee,thou,thy}",0.00726664737468249
## "{on,side}",0.00704157422590913
## "{god,shalt,thou}",0.00697726761197389
## "{commanded,lord}",0.00694511430500627
## "{cut,off}",0.00691296099803865
## "{god,lord,you}",0.00684865438410341
## "{hast,said,thou}",0.00678434777016816
## "{thou,wilt}",0.0066878878492653
## "{god,ye,your}",0.00662358123533005
## "{lord,shalt,thou,thy}",0.00659142792836243
## "{lord,shalt,thy}",0.00659142792836243
## "{god,lord,our}",0.00649496800745957
## "{shalt,thee,thou,thy}",0.00617343493778335
## "{israel,lord,saith}",0.00614128163081573
## "{burnt,offering}",0.00610912832384811
## "{lord,o,thou}",0.00607697501688049
## "{pray,thee}",0.00607697501688049
## "{lord,o,thy}",0.00604482170991286
## "{god,lord,thee,thou}",0.00598051509597762
## "{god,thee,thou,thy}",0.00591620848204238
## "{god,land,lord}",0.00578759525417189
## "{israel,lord,said}",0.00575544194720427
## "{god,lord,shalt,thou}",0.00572328864023665
## "{god,lord,shalt}",0.00572328864023665
## "{hand,right}",0.00543390887752805
## "{lord,saith,ye}",0.00530529564965757
## "{gold,silver}",0.00527314234268995
## "{thee,thine,thy}",0.00527314234268995
## "{lord,o,thee}",0.00524098903572232
## "{god,hast,thou}",0.0052088357287547
## "{god,lord,therefore}",0.00514452911481946
## "{god,lord,upon}",0.00514452911481946
## "{god,lord,people}",0.00511237580785184
## "{god,lord,out}",0.00508022250088422
## "{god,shalt,thou,thy}",0.00508022250088422
## "{god,shalt,thy}",0.00508022250088422
## "{before,god,lord}",0.00504806919391659
## "{came,lord,saying}",0.00501591588694897

## NULL
```

(2c) List the top 5 rules in decreasing order of confidence (conf) for item sets of size or length 2 or 3 which satisfy the support threshold that you have specified. Are they interesting rules within the problem context?

```
RULES <- as(rules, "data.frame")
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)
rules0$size[as.character(rules0$LHS)=="{} "] <- rules0$size[as.character(RULES$LHS)=="{} "]-1
RULES <- cbind(RULES, rules0)
head(RULES)
```

```
##                rules     support confidence    coverage      lift count
## 1   {wilt} => {thou} 0.006687888  1.0000000 0.006687888  8.013656   208
## 2 {silver} => {gold} 0.005273142  0.5815603 0.009067233 50.102788   164
## 3 {round} => {about} 0.009260152  0.9729730 0.009517379 50.943489   288
## 4  {hosts} => {lord} 0.008777853  0.9545455 0.009195846  4.452875   273
## 5  {right} => {hand} 0.005433909  0.5090361 0.010674898 12.225122   169
## 6     {cut} => {off} 0.006912961  0.6980519 0.009903219 45.418648   215
##         LHS      RHS size
## 1   {wilt}    {thou}    2
## 2 {silver}    {gold}    2
## 3  {round}   {about}    2
## 4  {hosts}    {lord}    2
## 5  {right}    {hand}    2
## 6    {cut}     {off}    2
```

```
# Top 5 rules in decreasing order of confidence
RULES1 <- RULES[RULES$size==c(2,3), ]
```

```
## Warning in RULES$size == c(2, 3): longer object length is not a multiple of
## shorter object length
```

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

```
##                          rules     support confidence    coverage     lift count
## 1            {wilt} => {thou} 0.006687888          1 0.006687888 8.013656   208
## 50     {shalt,thee} => {thou} 0.012700556          1 0.012700556 8.013656   395
## 52      {shalt,thy} => {thou} 0.015144208          1 0.015144208 8.013656   471
## 54      {god,shalt} => {thou} 0.006977268          1 0.006977268 8.013656   217
## 82 {god,saith,thus} => {lord} 0.007330954          1 0.007330954 4.664917   228
##                  LHS     RHS size
## 1            {wilt}   {thou}    2
## 50     {shalt,thee}   {thou}    3
## 52      {shalt,thy}   {thou}    3
## 54      {god,shalt}   {thou}    3
## 82 {god,saith,thus}   {lord}    4
```

The output above shows the top 5 rules in decreasing order of confidence for item sets of size/length 2 or 3

(d) List the top 5 rules in decreasing order of the lift measure for item sets of size 2 or 3. Always interpret the results.

```
# Top 5 rules in decreasing order of lift
RULESa <- RULES[RULES$size==c(2,3), ]
```

```
## Warning in RULES$size == c(2, 3): longer object length is not a multiple of
## shorter object length
```

```
RULESa <- RULES[ order(RULES$lift,decreasing = TRUE), ]
head(RULESa,n=5)
```

```
##                          rules     support confidence    coverage     lift count
## 3          {round} => {about} 0.009260152  0.9729730 0.009517379 50.94349   288
## 2          {silver} => {gold} 0.005273142  0.5815603 0.009067233 50.10279   164
## 6              {cut} => {off} 0.006912961  0.6980519 0.009903219 45.41865   215
## 7       {burnt} => {offering} 0.006109128  0.5792683 0.010546285 35.74568   190
## 85 {god,lord,saith} => {thus} 0.007330954  0.6608696 0.011092891 29.07172   228
##                   LHS         RHS size
## 3           {round}      {about}    2
## 2          {silver}       {gold}    2
## 6             {cut}        {off}    2
## 7           {burnt}   {offering}    2
## 85 {god,lord,saith}       {thus}    4
```

The above output shows the top 5 rules in decreasing order of the lift measure for item sets of size 2 or 3. We can observe that the top 5 rules have positive lifts which shows a positive relationship

(2e) Find the conviction measures for the top-lift 5 rules in Part (d) and explain how this measure avoids the problems associated with both the confidence and the lift measures.

```
rules.sorted <- sort(rules, by="lift")
#inspect(rules.sorted)
subset.matrix <- is.subset(rules.sorted, rules.sorted)
subset.matrix[lower.tri(subset.matrix, diag=T)] <- F
redundant <- apply(subset.matrix, 2, any)
rules.pruned <- rules.sorted[!redundant]
#inspect(rules.pruned)

M0 <- interestMeasure(rules.pruned[1:5], c("support", "chiSquare", "confidence", "conviction", "cosine"
    "coverage", "leverage", "lift", "oddsRatio"), transactions=dat)
dim(M0);
```

```
## [1] 5 9
```

```
inspect(head(rules.pruned)); head(M0)
```

```
##     lhs                    rhs        support     confidence coverage
## [1] {round}             => {about}    0.009260152 0.9729730  0.009517379
## [2] {silver}            => {gold}     0.005273142 0.5815603  0.009067233
## [3] {cut}               => {off}      0.006912961 0.6980519  0.009903219
## [4] {burnt}             => {offering} 0.006109128 0.5792683  0.010546285
## [5] {god, lord, saith} => {thus}     0.007330954 0.6608696  0.011092891
## [6] {god, saith}        => {thus}     0.007330954 0.5937500  0.012346870
##     lift     count
## [1] 50.94349 288
## [2] 50.10279 164
## [3] 45.41865 215
## [4] 35.74568 190
## [5] 29.07172 228
## [6] 26.11912 228
```

```
##       support chiSquared confidence conviction    cosine    coverage
## 1 0.009260152  14514.081  0.9729730  36.293335 0.6868366 0.009517379
## 2 0.005273142   8057.876  0.5815603   2.362091 0.5140030 0.009067233
```

```
## 3 0.006912961   9580.406  0.6980519   3.260928 0.5603368 0.009903219
## 4 0.006109128   6592.220  0.5792683   2.338295 0.4673061 0.010546285
## 5 0.007330954   6394.891  0.6608696   2.881687 0.4616529 0.011092891
##      leverage     lift oddsRatio
## 1 0.009078379 50.94349 3588.1176
## 2 0.005167896 50.10279  216.0375
## 3 0.006760756 45.41865  268.3654
## 4 0.005938223 35.74568  133.5551
## 5 0.007078786 29.07172  123.1761
```

```
#M0 <- interestMeasure(rules, c( "conviction"), transactions=dat)
#dim(M0)
```

The Conviction measures the implication strength of the rule from statistical independence. Conviction produces an association rule with better predictive ability, unlike lift, Conviction takes into account the strength of the directed association, unlike Confidence, the support of both antecedent and consequent are considered in conviction.