Project III: kPCA and Association Rules

Quaye, George Ekow

Contents

1 Bringing in the Data for Train and Test

(a) Bring in the training set optdigits.tra, which has sixty-four (p = 64) inputs plus the target variable that indicates the digit 0-9. Examine the data briefly. Remove any column that is unary (i.e., containing only one values) and check on possible missing values.

```
# Read both the training data set optdigits.tra and the test data set optdigits.tes into R.
# 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); dim(test)
```

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

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

```
data_miss <- rbind(train, test); dim(data_miss)
```

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

```
# INSPECT THE DISTINCT VALUES OF EACH X
for (j in 1:NCOL(data_miss)){
  x <- data_miss[,j]
  print(table(x, useNA="ifany"))
}
```

```
# Listing the missing rate for each variable.
miss.info <- function(dat, filename=NULL){
  vnames <- colnames(dat); vnames
  n <- nrow(dat)
  out <- NULL
  for (j in 1: ncol(dat)){
    vname <- colnames(dat)[j]
    x <- as.vector(dat[,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)
}
miss.info(data_miss)
```

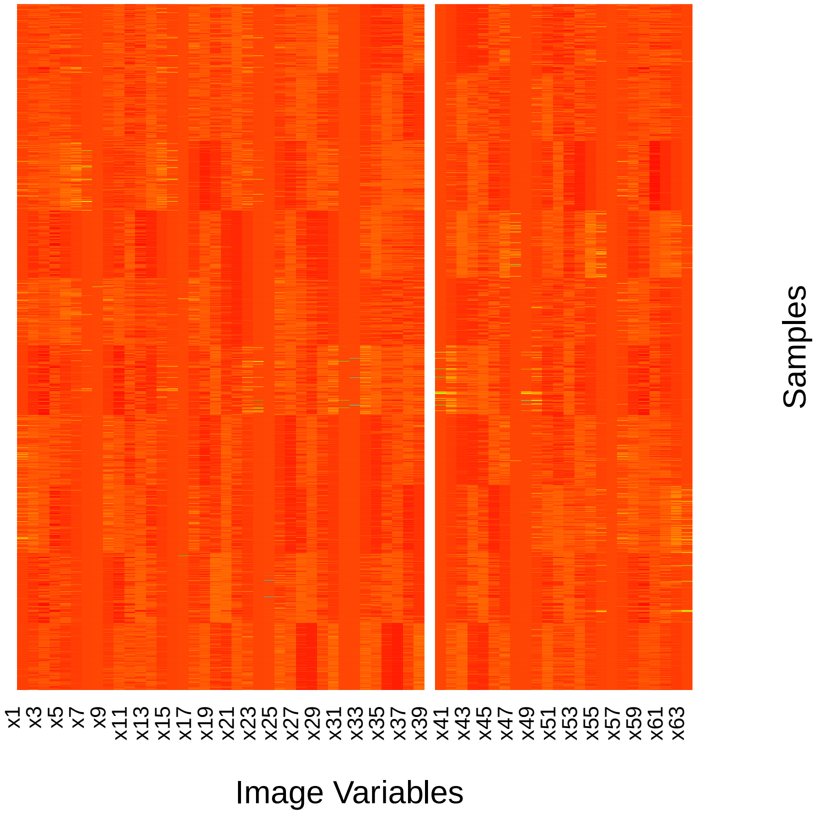
```
##    col.number vname    mode n.levels ncomplete miss.perc
## 1           1    x1 numeric        1      5620         0
## 2           2    x2 numeric        9      5620         0
## 3           3    x3 numeric       17      5620         0
## 4           4    x4 numeric       17      5620         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 output, there are no missing values in both the test and train data sets.

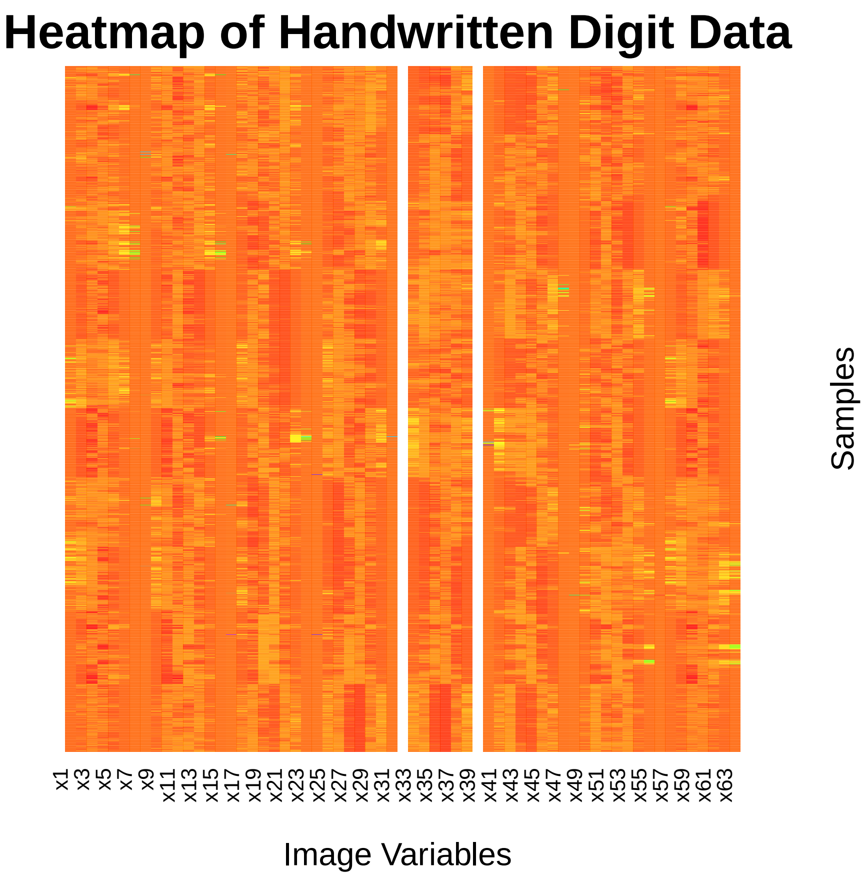
```r
# Heat Map on the Train Data
dat1 <- data.matrix(train[order(train$digit), -65])
n <- NROW(dat1)
color <- rainbow(n, alpha = 0.8)
heatmap(dat1, col=color, scale="column", Rowv=NA, Colv=NA,
labRow=FALSE, margins=c(4,4), xlab="Image Variables", ylab="Samples",
main="Heatmap of Handwritten Digit Data")
```

Heatmap of Handwritten Digit Data



The heatmap indicates how the handwritten digit data is clustered. From the heat map, we can observe different patterns where each pattern corresponds to the digits 0-9 and also there are no observations recorded for the 1st and 40th variable.

```r
# Heat Map on the Test Data
dat0 <- data.matrix(test[order(test$digit), -65])
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="Heatmap of Handwritten Digit Data")
```



The heatmap indicates how the handwritten digit data is clustered.We observe different patterns where each pattern corresponds to the digits 0-9 from the heatmap and also there are no observations recorded for the 1st, 33rd and 40th variable.

(b) 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.

```
# removing target variable
dat00 <- data.matrix(train[,-c(33,65)]) # Train
dat01 <- data.matrix(test[,-c(33,65)]) # Test


# Remove the Unary variables
newTrain <- dat00[,apply(dat00, 2, var, na.rm=TRUE) != 0] # Train
newTest <- dat01[,apply(dat01, 2, var, na.rm=TRUE) != 0]  # Test
dim(newTrain);dim(newTest)
```

```
## [1] 3823   61

## [1] 1797   61
```

The test data had 3 variables with the target variable removed. The train data had 3 variable with the target variable removed instead just one with the target variable, just to make the two data sets conformable for further analysis.

```r
# STANDARDIZE THE Train DATA
newTrain.scaled <- data.frame(apply(newTrain, 2, scale, center=T, scale=T))
```

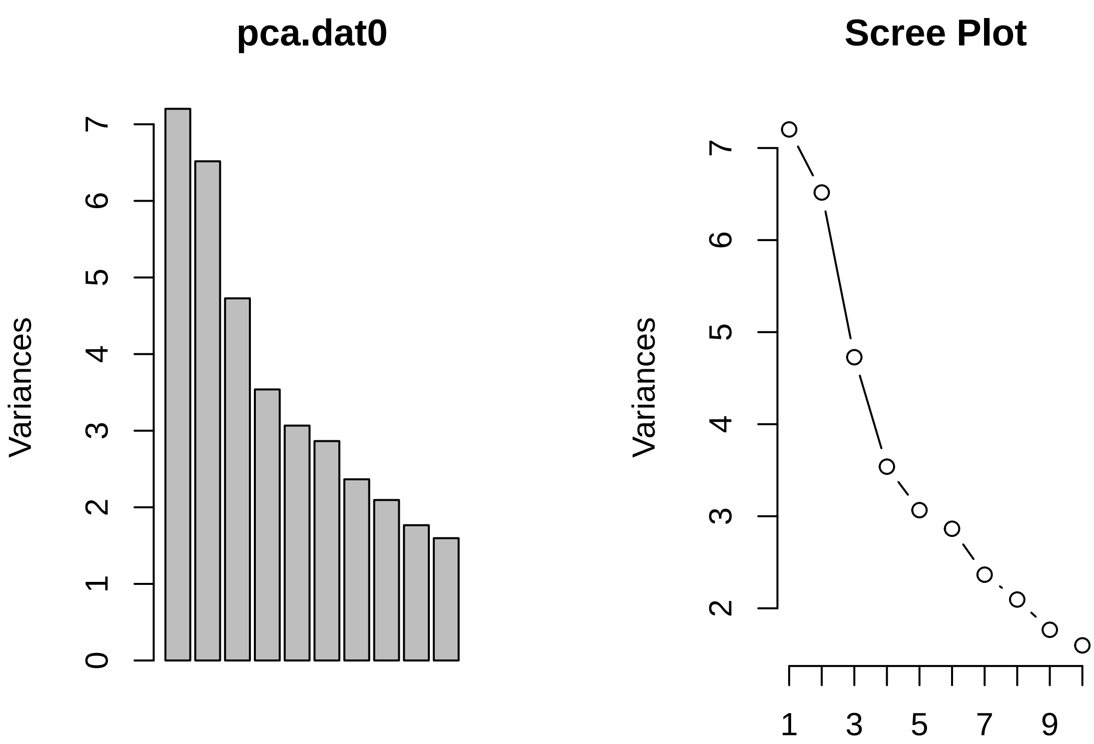
```r
# STANDARDIZE THE Test DATA
newTest.scaled <- data.frame(apply(newTest, 2, scale, center=T, scale=T))
```

```r
# ORDINARY PCA
pca.dat0 <- prcomp(newTrain.scaled, scale=FALSE, retx=TRUE);
```

```r
# OBTAIN EIGENVALUES AND COMPARE
lambda <- eigen(cov(newTrain.scaled), only.values = T)$values
lambda
```

```
##  [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
```

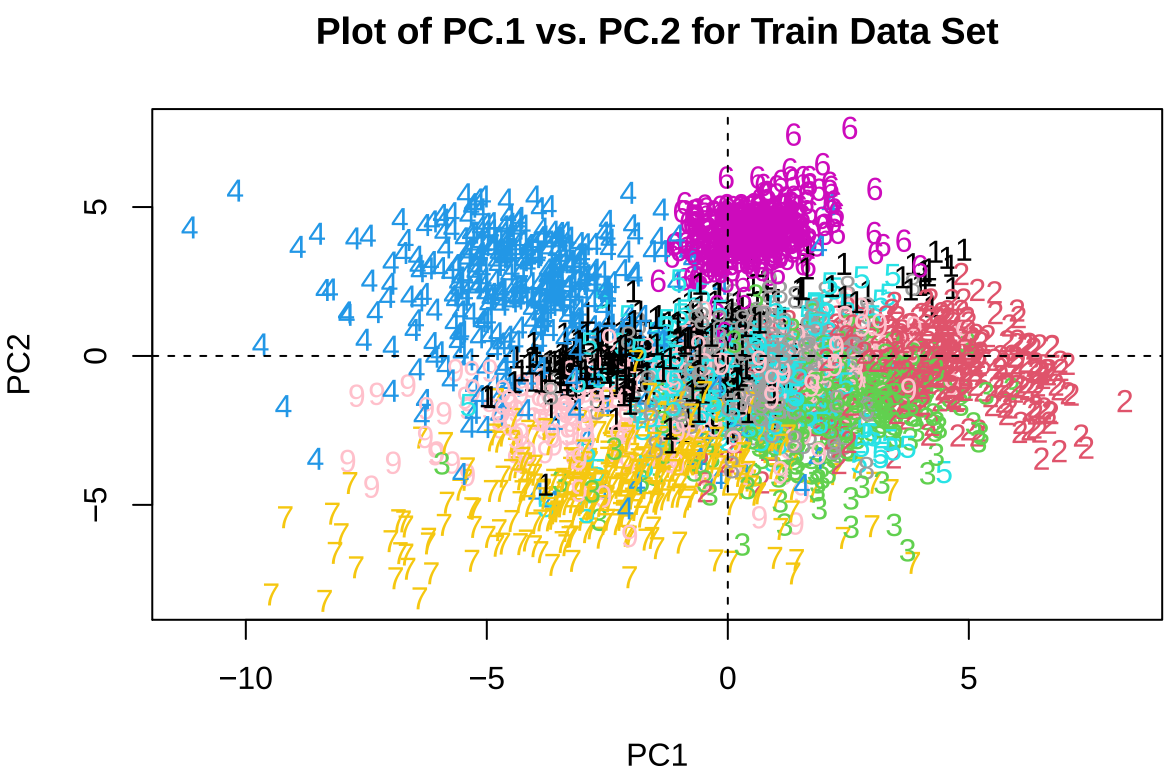
```r
# PLOT THE VARIANCES
par(mfrow=c(1,2), mar=rep(4,4))
plot(pca.dat0)
screeplot(pca.dat0, type="lines", main="Scree Plot")
```



Given the Scree plot, we choose $\lambda = 2$.

```
##  [1] "black"   "#DF536B" "#61D04F" "#2297E6" "#28E2E5" "#CD0BBC" "#F5C710"
##  [8] "gray62"  "pink"    "orange"  "brown"
```

```
#Scatter plot of the first two PCs
plot(pca.dat0$x[,1:2], pch="", main="Plot of PC.1 vs. PC.2 for Train Data Set")
text(pca.dat0$x[,1:2], labels=train$digit, col=train$digit)
abline(v=0, lty=2)
abline(h=0, lty=2)
```



The plot of PC2 vs. PC1 with a scatterplot displays the structure of remoteness-like handwritten data digit as a geometrical picture. Given the plot, digits regularized closer to one another are more similar than those regularized further away. Again from the plot we observe that the two components hold some information, especially for specific digits, but clearly not enough to set all of them apart.

(c) Run kernel PCA on the input variables only. 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.

```
# KERNEL PCA
library(kernlab)
kpc <- kpca(~., data=newTrain.scaled, kernel="rbfdot",
    kpar=list(sigma=0.01), features=30);kpc
```

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

```
##      Comp.1      Comp.2      Comp.3      Comp.4      Comp.5      Comp.6
```

```
## 0.047519592 0.045664048 0.039840558 0.030705225 0.026628409 0.021413114
##      Comp.7      Comp.8      Comp.9     Comp.10     Comp.11     Comp.12
## 0.019701070 0.016472839 0.013885911 0.012663215 0.011952366 0.010649484
##     Comp.13     Comp.14     Comp.15     Comp.16     Comp.17     Comp.18
## 0.009883164 0.008694631 0.008245204 0.007618617 0.007425526 0.006615084
##     Comp.19     Comp.20     Comp.21     Comp.22     Comp.23     Comp.24
## 0.005794819 0.005245154 0.005194623 0.004942674 0.004663917 0.004531311
##     Comp.25     Comp.26     Comp.27     Comp.28     Comp.29     Comp.30
## 0.004386965 0.004076748 0.003712558 0.003697003 0.003567404 0.003344525
```

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

```
## Gaussian Radial Basis kernel function.
##  Hyperparameter : sigma =  0.01
```

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

```
## [1] 3823   30
```

```
head(PCV)
```

```
##             [,1]        [,2]        [,3]         [,4]        [,5]       [,6]
## [1,] -0.10195063 -0.05860036  0.10655910  0.063704508 -0.15656419 -0.1079505
## [2,] -0.12076060 -0.06290083  0.09896195  0.146398447 -0.07637423 -0.1519156
## [3,]  0.08968359  0.07906699  0.05674893  0.036808090 -0.12922881  0.2290957
## [4,] -0.01005968  0.12538306  0.08280258 -0.038311254  0.07791338 -0.1604375
## [5,] -0.11006213 -0.07259154 -0.02951478  0.007796194 -0.02984472  0.1696078
## [6,]  0.02929571 -0.04214359 -0.06273069  0.113924498  0.07585350 -0.1097459
##              [,7]        [,8]        [,9]       [,10]       [,11]        [,12]
## [1,] -0.004607112 -0.09783233 -0.01011826 -0.14962265 -0.01716951  0.109869527
## [2,]  0.109466552 -0.03320728 -0.01261205 -0.09856210  0.15253131  0.152885466
## [3,]  0.191680618 -0.03668010  0.14840631 -0.18583229  0.06298150  0.227902867
## [4,] -0.155939895  0.16011851 -0.10150844 -0.06262252  0.11287617  0.002287816
## [5,] -0.174282304 -0.04086835  0.06932212 -0.04642227  0.11257955  0.080497045
## [6,]  0.008420583  0.32920636 -0.03668993 -0.02262363 -0.02229629 -0.126847960
##            [,13]        [,14]       [,15]      [,16]       [,17]       [,18]
## [1,]  0.06802592 -0.008133507 -0.08944444  0.0883160  0.03050148  0.27773294
## [2,]  0.19064647  0.036422483 -0.16048459 -0.3781034  0.17611290  0.05081149
## [3,]  0.15012432  0.016612504 -0.13542876  0.1322222 -0.09273964 -0.15085773
## [4,] -0.17927493 -0.088225861  0.24633339  0.0188809 -0.20222785  0.02454553
## [5,] -0.25432585 -0.255438747 -0.13814529 -0.1596825  0.02924247  0.13338633
## [6,]  0.21579924  0.086504138 -0.54825173  0.1863846 -0.68509077 -0.34454893
##            [,19]       [,20]       [,21]      [,22]      [,23]      [,24]
## [1,]  0.16027120  0.24415049  0.07002552 -0.3159765 -0.1686883 -0.1780075
## [2,]  0.04013325  0.30157177 -0.26667695 -0.3899953 -0.0353040  0.1106751
## [3,] -0.02853426 -0.30954552  0.16743361 -0.1439710  0.0362897 -0.1944131
## [4,] -0.10373599 -0.31941136 -0.27576780 -0.1648694  0.1628660  0.5242731
## [5,] -0.13699258  0.02858652  0.03728481 -0.2250845 -0.2399408 -0.1000898
## [6,]  0.37351387  0.08244603  0.06522432  0.1707159 -0.1540556 -0.5966795
##            [,25]       [,26]       [,27]      [,28]       [,29]       [,30]
## [1,]  0.21432353  0.15627531 -0.21554684  0.07160721  0.01582294 -0.40031991
```

```
## [2,]  0.11293778 -0.17950900  0.36159617  0.07630550 -0.08979419 -0.51742371
## [3,] -0.16748760 -0.33850186 -0.51126554 -0.17912318 -0.26740416  0.07576810
## [4,]  0.11191848  0.19842149 -0.03731941  0.16107506  0.16768467 -0.14699849
## [5,]  0.01274817  0.29400951 -0.12399878  0.30683293  0.10059179 -0.08452944
## [6,]  0.29715875 -0.06267483  0.09262491  0.58412432 -0.01817243 -0.63318102
```

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

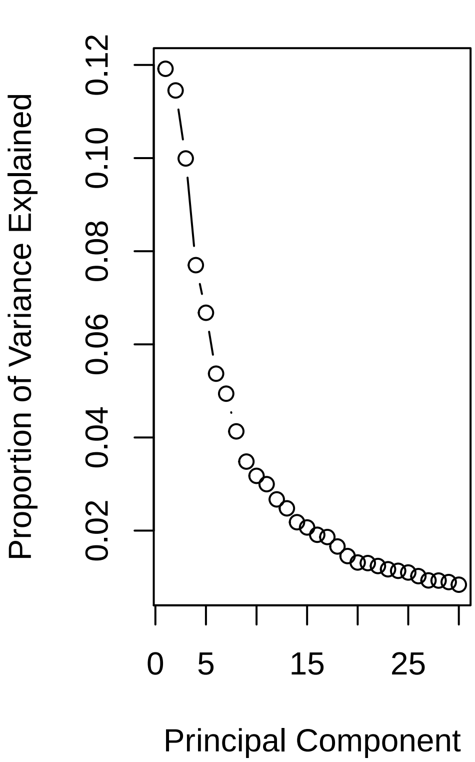
```
## [1] 3823   30
```

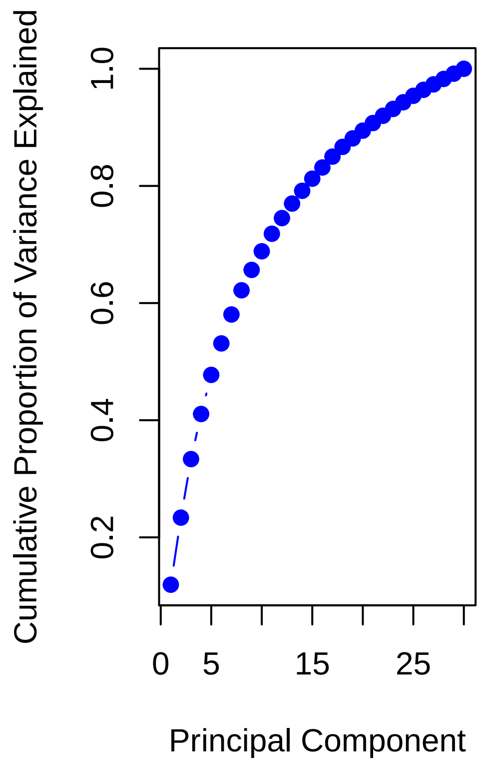
```
head(PC);
```

```
##          [,1]      [,2]      [,3]       [,4]       [,5]       [,6]        [,7]
## 1 -18.521106 -10.23008 16.230065  7.4780221 -15.938299  -8.837080  -0.3469948
## 2 -21.938264 -10.98083 15.072939 17.1851389  -7.774927 -12.436164   8.2447131
## 3  16.292584  13.80301  8.643455  4.3207572 -13.155546  18.754312  14.4368455
## 4  -1.827515  21.88858 12.611698 -4.4972077   7.931614 -13.133785 -11.7449547
## 5 -19.994700 -12.67257 -4.495409  0.9151646  -3.038204  13.884487 -13.1264534
## 6   5.322076  -7.35716 -9.554540 13.3731494   7.721918  -8.984057   0.6342147
##         [,8]       [,9]     [,10]     [,11]       [,12]     [,13]      [,14]
## 1 -6.161056 -0.5371362 -7.243453 -0.784542  4.47311548  2.570246 -0.2703543
## 2 -2.091251 -0.6695212 -4.771536  6.969750  6.22442239  7.203260  1.2106678
## 3 -2.309954  7.8782735 -8.996415  2.877870  9.27860407  5.672198  0.5521925
## 4 10.083570 -5.3886610 -3.031648  5.157752  0.09314382 -6.773605 -2.9325900
## 5 -2.573712  3.6800229 -2.247370  5.144198  3.27727430 -9.609279 -8.4906751
## 6 20.731989 -1.9477159 -1.095243 -1.018804 -5.16435802  8.153615  2.8753607
##         [,15]       [,16]       [,17]      [,18]      [,19]      [,20]
## 1  -2.819415   2.5722895   0.8658695  7.0237180  3.5505826  4.8957600
## 2  -5.058702 -11.0126314   4.9994553  1.2849954  0.8890956  6.0471843
## 3  -4.268906   3.8510999  -2.6326730 -3.8151115 -0.6321363 -6.2070756
## 4   7.764777   0.5499246  -5.7408009  0.6207435 -2.2981248 -6.4049077
## 5  -4.354535  -4.6509098   0.8301291  3.3732692 -3.0348776  0.5732232
## 6 -17.281671   5.4286331 -19.4482100 -8.7134588  8.2746736  1.6532262
##         [,21]     [,22]      [,23]      [,24]      [,25]      [,26]      [,27]
## 1  1.3906399 -5.970642 -3.0077375  -3.083659  3.5944984  2.4356146 -3.0592796
## 2 -5.2959497 -7.369291 -0.6294757   1.917247  1.8941209 -2.7977211  5.1321736
## 3  3.3250716 -2.720453  0.6470509  -3.367858 -2.8089959 -5.2756899 -7.2564470
## 4 -5.4764852 -3.115346  2.9039253   9.082088  1.8770257  3.0924801 -0.5296784
## 5  0.7404408 -4.253161 -4.2781809  -1.733876  0.2138042  4.5822585 -1.7599281
## 6  1.2952927  3.225820 -2.7468354 -10.336400  4.9837582 -0.9768129  1.3146353
##        [,28]      [,29]      [,30]
## 1  1.012071  0.2157962 -5.1185387
## 2  1.078475 -1.2246296 -6.6158420
## 3 -2.531663 -3.6469072  0.9687801
## 4  2.276577  2.2869144 -1.8795405
## 5  4.336667  1.3718894 -1.0808036
## 6  8.255804 -0.2478389 -8.0959290
```

```
# COMPUTE NONCUMULATIVE/CUMULATIVE PROPORTIONS OF VARIATION EXPLAINED
var.pc <- eig(kpc)
names(var.pc) <- 1:length(var.pc)
```

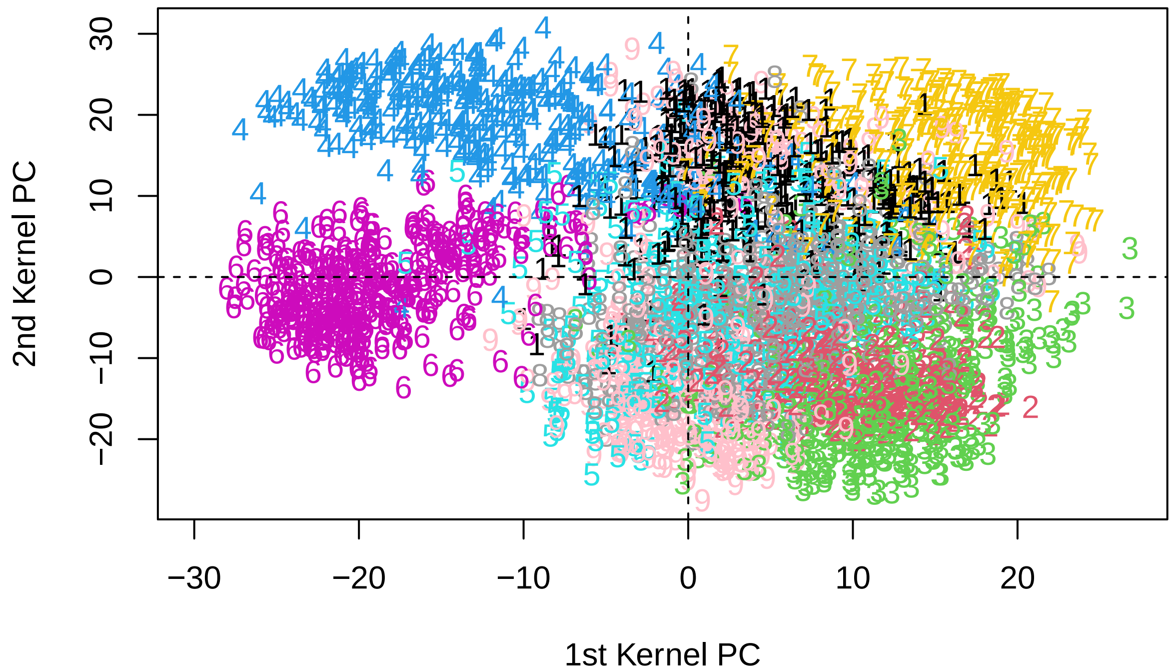
```r
prop.pc <- var.pc/sum(var.pc)

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 Scree plot, we choose $\lambda = 2$.

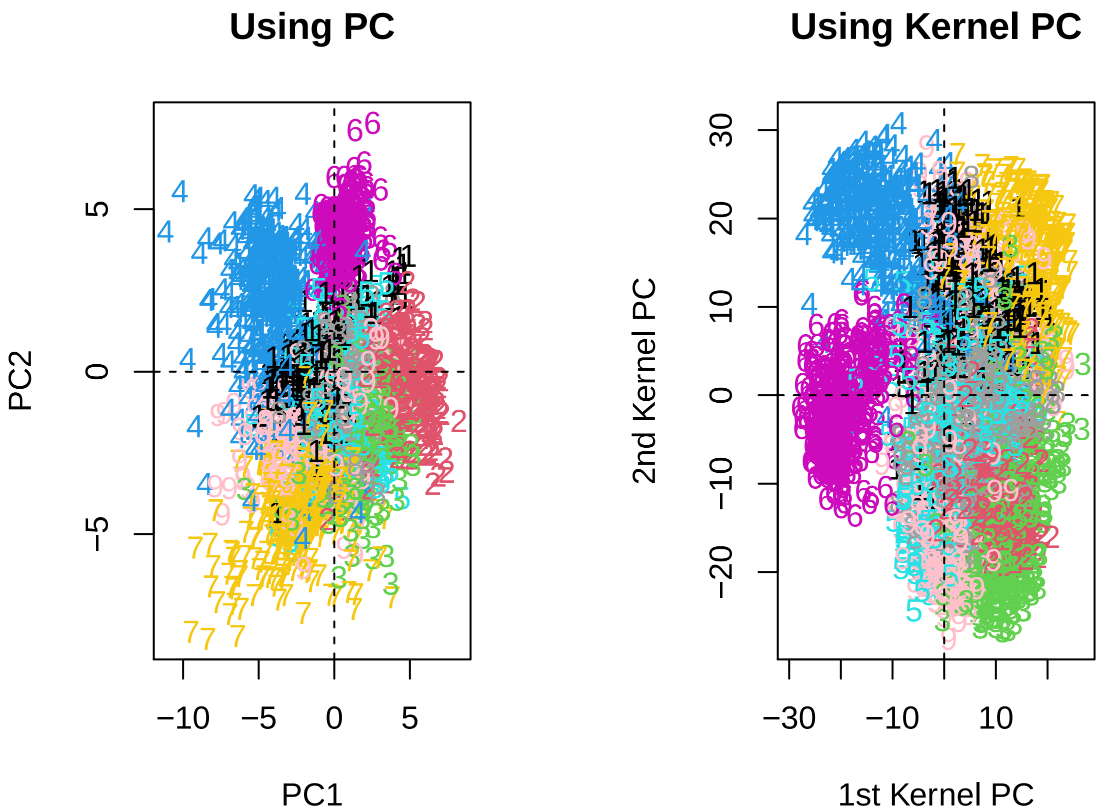
```r
# Plot THE DATA PROJECTION ON THE KERNEL PCS
plot(PC[, 1:2],col=train$digit, pch="",
    xlab="1st Kernel PC", ylab="2nd Kernel PC")
text(PC[,1:2], labels=train$digit, col=train$digit)
abline(v=0, lty=2)
abline(h=0, lty=2)
```



From the Kernel PCA plots, digits ordinated closer to one another are more similar than those ordinated further away.Again from the plot we observe that the two components hold some information, especially for specific digits,but clearly not spaced apart.

Now we perform comparison of the ordinary PCA and Kernel PCA as required by the question

```
par(mfrow=c(1,2), mar=rep(4,4))
#Scatter plot of the first two PCs
plot(pca.dat0$x[,1:2], pch="", main="Using PC")
text(pca.dat0$x[,1:2], labels=train$digit, col=train$digit)
abline(v=0, lty=2)
abline(h=0, lty=2)

# Plot THE DATA PROJECTION ON THE KERNEL PCS
plot(PC[, 1:2],col=train$digit, pch="",main="Using Kernel PC" ,
    xlab="1st Kernel PC", ylab="2nd Kernel PC")
text(PC[,1:2], labels=train$digit, col=train$digit)
abline(v=0, lty=2)
abline(h=0, lty=2)
```

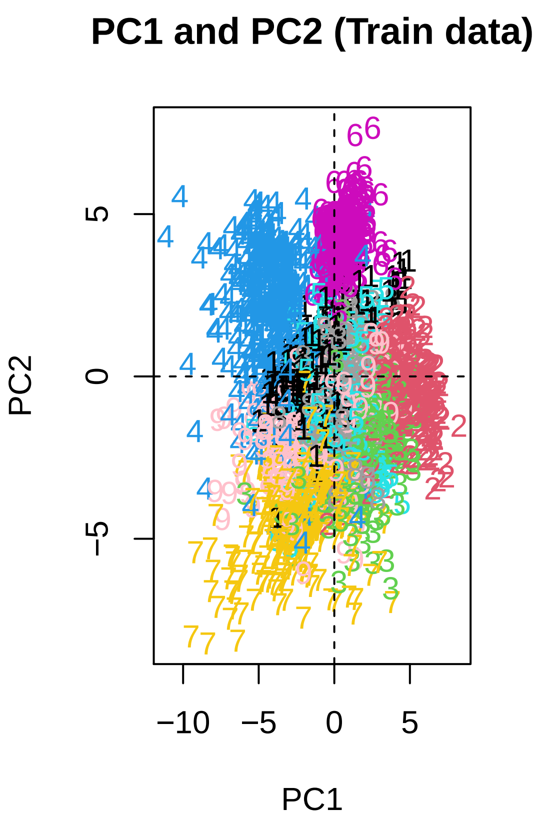


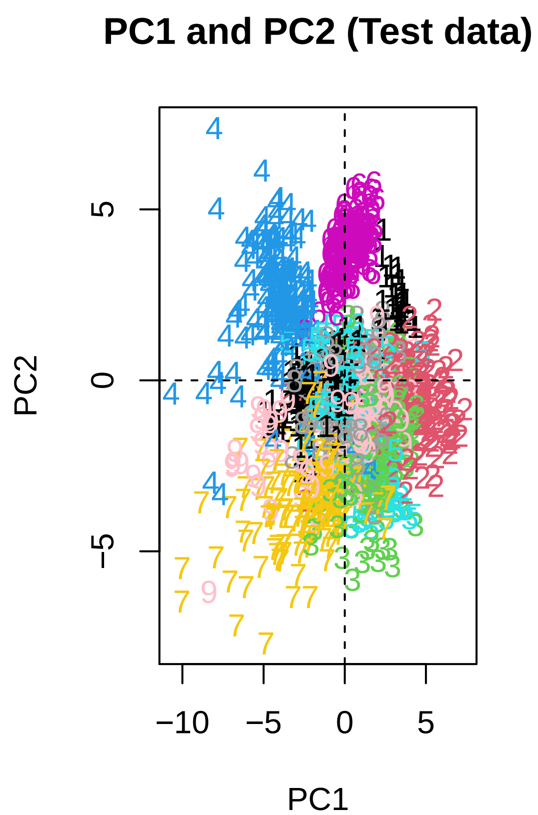
From the two plots, we observe that the Kernel PCA is able to find good representative directional outcome.

(d) Apply both PCA and kPCA to the test set optdigits.tes. Obtain the first two principal components and make similar plots as Part (b) & (c) and compare.

```
# PCA
pred_pca <- predict(pca.dat0, newTest.scaled);
```

```
par(mfrow=c(1,2), mar=rep(4,4))
#Scatter plot of the first two PCs
plot(pca.dat0$x[,1:2], pch="", main="PC1 and PC2 (Train data)")
text(pca.dat0$x[,1:2], labels=train$digit, col=train$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=test$digit, col=test$digit)
abline(v=0, lty=2)
abline(h=0, lty=2)
```

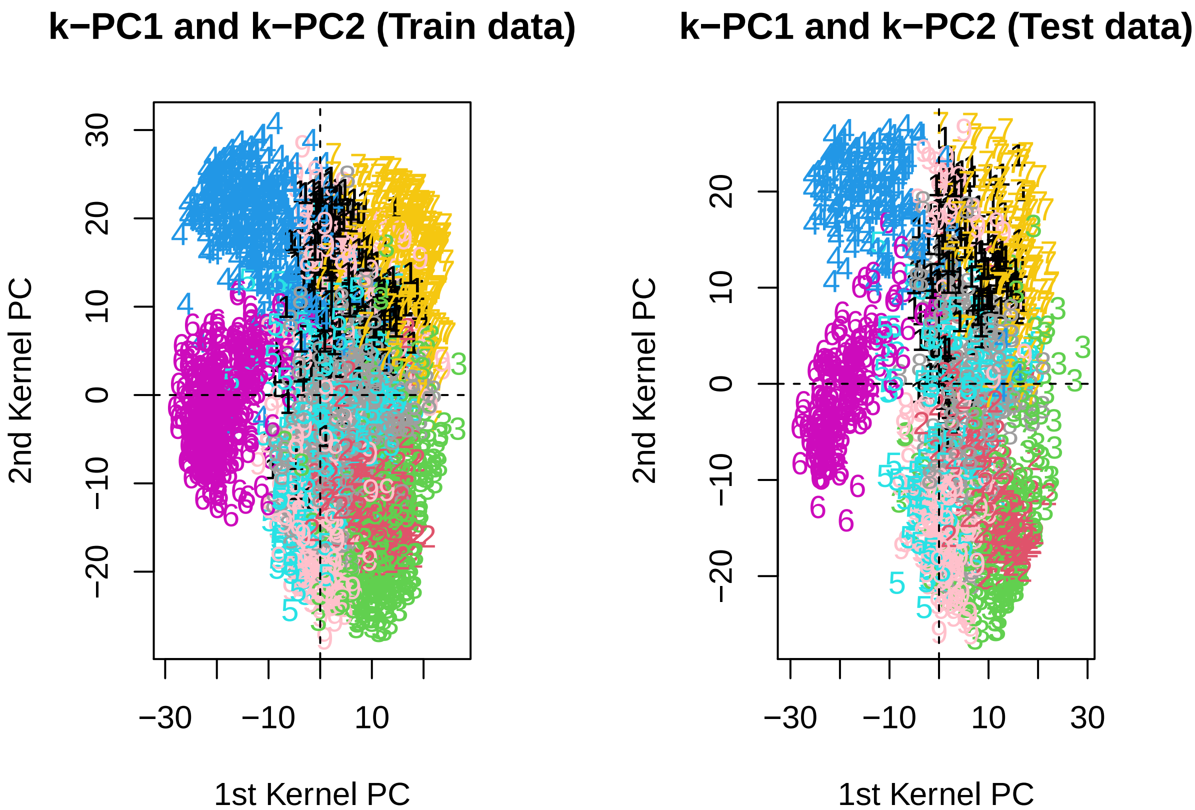




From the plots above we observe that the two PCA plots are kind of similar with second plot that is the test slightly dispersed. This shows that the PCA somehow sufficiently predicts the test data.

```
# KPCA
pred_kpca <- predict(kpc, newTest.scaled);
```

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

#Scatter plot of the first two KERNEL PCS
plot(PC[, 1:2],col=train$digit, main="k-PC1 and k-PC2 (Train data)", pch="",
    xlab="1st Kernel PC", ylab="2nd Kernel PC")
text(PC[,1:2], labels=train$digit, col=train$digit)
abline(v=0, lty=2)
abline(h=0, lty=2)

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



From the plots above we observe that the two Kernel PCA plots are also similar. This shows that the Kernel PC effectively predicts the test data.

2 (Association Rules)

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

```
library(arules)
bible <- read.transactions(file="http://snap.stanford.edu/class/cs246-data/AV1611Bible.txt",
format = "basket", sep =" ", rm.duplicates =F)
dat <- bible; dim(dat)
```

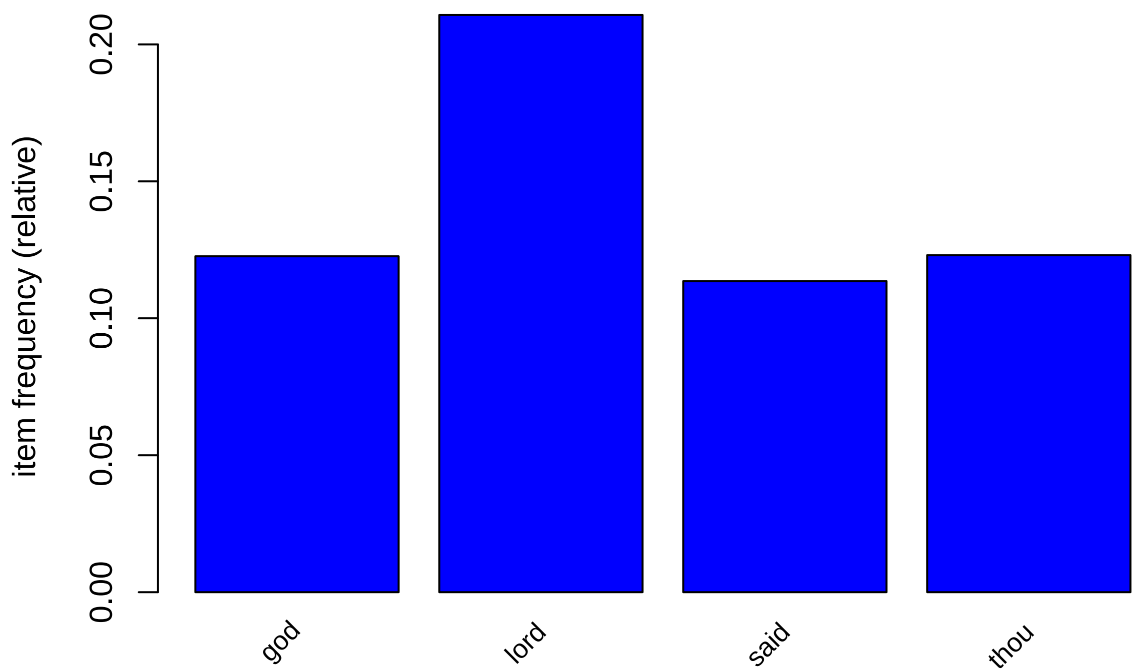
```
## [1] 31101 13978
```

```
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}
```

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

```
# PLOT ITEMS WITH HIGH FREQUENCIES
itemFrequencyPlot(dat, support = 0.1, cex.names = 0.8, col="blue")
```



From the above output, "lord" is the item with the highest frequency.

```
# THE TOP 20 ITEMS
item.freq <- itemFrequency(dat, type = "relative")
item.freq <- sort(item.freq, decreasing = TRUE)
item.freq[1:20]
```

```
##       lord       thou        god       said        thy         ye       thee
## 0.21076493 0.12305071 0.12263271 0.11356548 0.09633131 0.09035079 0.08578502
##        out        man     israel       upon         by       then      there
## 0.07739301 0.07318093 0.07237709 0.07128388 0.07015852 0.06681457 0.06549629
##        you       came     people       hath       come        had
## 0.06330986 0.06035176 0.06009453 0.05768303 0.05710427 0.05498215
```

The output given above list the top 20 frequent itemsets.

```
# Association analysis
rules <- apriori(dat, parameter = list(support = 0.01, confidence = 0.6,
    target = "rules", maxlen=5))
```

```
## 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
##       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 ...[13978 item(s), 31101 transaction(s)] done [0.10s].
## sorting and recoding items ... [222 item(s)] done [0.00s].
## creating transaction tree ... done [0.01s].
## checking subsets of size 1 2 3 4 done [0.01s].
## writing ... [18 rule(s)] done [0.00s].
## creating S4 object  ... done [0.01s].
```

```
#rules
inspect(rules[1:18])
```

```
##      lhs              rhs     support    confidence coverage   lift      count
## [1]  {answered}    => {said}  0.01054628 0.6735113  0.01565866  5.930599  328
## [2]  {art}         => {thou}  0.01408315 0.9887133  0.01424391  8.035007  438
## [3]  {she}         => {her}   0.01327932 0.6020408  0.02205717 16.395859  413
## [4]  {thus}        => {saith} 0.01459760 0.6513630  0.02241085 17.066588  454
## [5]  {thus}        => {lord}  0.01617311 0.7216643  0.02241085  3.424025  503
## [6]  {hast}        => {thou}  0.02614064 0.9830713  0.02659078  7.989156  813
## [7]  {shalt}       => {thou}  0.03829459 0.9991611  0.03832674  8.119913 1191
## [8]  {saith}       => {lord}  0.02797338 0.7329402  0.03816598  3.477524  870
## [9]  {saith,thus} => {lord}  0.01389023 0.9515419  0.01459760  4.514707  432
## [10] {lord,thus}   => {saith} 0.01389023 0.8588469  0.01617311 22.502947  432
## [11] {hast,thy}    => {thou}  0.01054628 0.9732938  0.01083566  7.909697  328
## [12] {shalt,thee} => {thou}  0.01241118 1.0000000  0.01241118  8.126731  386
## [13] {shalt,thy}   => {thou}  0.01475837 1.0000000  0.01475837  8.126731  459
## [14] {lord,shalt} => {thou}  0.01183242 0.9972900  0.01186457  8.104707  368
## [15] {god,saith}   => {lord}  0.01106074 0.9005236  0.01228256  4.272644  344
## [16] {god,israel} => {lord}  0.01080351 0.7073684  0.01527282  3.356196  336
## [17] {god,thee}    => {lord}  0.01044982 0.6040892  0.01729848  2.866175  325
## [18] {god,thy}     => {lord}  0.01311855 0.6962457  0.01884184  3.303423  408
```

```
summary(rules)
```

```
## set of 18 rules
##
## rule length distribution (lhs + rhs):sizes
##  2  3
##  8 10
##
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   2.000   2.000   3.000   2.556   3.000   3.000
##
## summary of quality measures:
##     support           confidence        coverage             lift
```

```
##  Min.   :0.01045   Min.   :0.6020   Min.   :0.01084   Min.   : 2.866
##  1st Qu.:0.01125   1st Qu.:0.6990   1st Qu.:0.01433   1st Qu.: 3.676
##  Median :0.01358   Median :0.8797   Median :0.01592   Median : 7.949
##  Mean   :0.01577   Mean   :0.8356   Mean   :0.01912   Mean   : 7.973
##  3rd Qu.:0.01472   3rd Qu.:0.9873   3rd Qu.:0.02232   3rd Qu.: 8.125
##  Max.   :0.03829   Max.   :1.0000   Max.   :0.03833   Max.   :22.503
##      count
##  Min.   : 325.0
##  1st Qu.: 350.0
##  Median : 422.5
##  Mean   : 490.4
##  3rd Qu.: 457.8
##  Max.   :1191.0
##
## mining info:
##  data ntransactions support confidence
##   dat         31101    0.01        0.6
```

(c) List the top 5 rules in decreasing order of confidence (conf) for item sets of size 2 or 3 which satisfy the support threshold that you have specified.

```
RULES <- as(rules, "data.frame")
rules0 <- data.frame(matrix(unlist(strsplit(as.character(RULES$rules), split="=>")), ncol=2, byrow=TRUE
colnames(rules0) <- c("LHS", "RHS")# LHS=Left hand side, RHS= Right had side.
rule.size <- function(x){length(unlist(strsplit(as.character(x), split=",")))}
rules0$size <- apply(rules0, 1, rule.size)
rules0$size[as.character(rules0$LHS)=="{} "] <- rules0$size[as.character(RULES$LHS)=="{} "]-1   # HANDL
RULES <- cbind(RULES, rules0)
head(RULES)
```

```
##                  rules    support confidence   coverage      lift count
## 1 {answered} => {said} 0.01054628  0.6735113 0.01565866  5.930599   328
## 2      {art} => {thou} 0.01408315  0.9887133 0.01424391  8.035007   438
## 3       {she} => {her} 0.01327932  0.6020408 0.02205717 16.395859   413
## 4    {thus} => {saith} 0.01459760  0.6513630 0.02241085 17.066588   454
## 5     {thus} => {lord} 0.01617311  0.7216643 0.02241085  3.424025   503
## 6     {hast} => {thou} 0.02614064  0.9830713 0.02659078  7.989156   813
##          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
```

```
# Top 5 rules in decreasing order of confidence
RULES2 <- RULES[RULES$size==2, ]
RULES2 <- RULES[ order(RULES$confidence,decreasing = TRUE), ]
head(RULES2,n=5)
```

```
##                        rules    support confidence   coverage     lift count
## 12 {shalt,thee} => {thou} 0.01241118  1.0000000 0.01241118 8.126731   386
## 13  {shalt,thy} => {thou} 0.01475837  1.0000000 0.01475837 8.126731   459
```

```
## 7       {shalt} => {thou} 0.03829459  0.9991611 0.03832674 8.119913  1191
## 14 {lord,shalt} => {thou} 0.01183242  0.9972900 0.01186457 8.104707   368
## 2         {art} => {thou} 0.01408315  0.9887133 0.01424391 8.035007   438
##              LHS     RHS size
## 12 {shalt,thee}   {thou}    3
## 13  {shalt,thy}   {thou}    3
## 7       {shalt}   {thou}    2
## 14 {lord,shalt}   {thou}    3
## 2         {art}   {thou}    2
```

The output above gives top 5 rules in decreasing order of confidence

d)

```
# Top 5 rules in decreasing order of lift
RULES2 <- RULES[RULES$size==2, ]
RULES2 <- RULES[ order(RULES$lift, decreasing = TRUE), ]
head(RULES2,n=5)
```

```
##                       rules    support confidence   coverage      lift count
## 10 {lord,thus} => {saith} 0.01389023  0.8588469 0.01617311 22.502947   432
## 4       {thus} => {saith} 0.01459760  0.6513630 0.02241085 17.066588   454
## 3          {she} => {her} 0.01327932  0.6020408 0.02205717 16.395859   413
## 12 {shalt,thee} => {thou} 0.01241118  1.0000000 0.01241118  8.126731   386
## 13  {shalt,thy} => {thou} 0.01475837  1.0000000 0.01475837  8.126731   459
##              LHS      RHS size
## 10  {lord,thus}   {saith}    3
## 4        {thus}   {saith}    2
## 3         {she}     {her}    2
## 12 {shalt,thee}    {thou}    3
## 13  {shalt,thy}    {thou}    3
```

The output above gives top 5 rules in decreasing order of lift

(e) Explain how this measure avoids the problems associated with both the confidence and the lift measures.

```
M <- interestMeasure(rules[1:5], c( "conviction"), transactions=dat)
M
```

```
## [1]  2.715054 77.697707  2.420552  2.758841  2.835551
```

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

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

```
intM
```

```
##      support chiSquared confidence conviction    cosine   coverage    leverage
## 1 0.01054628   1540.928  0.6735113   2.715054 0.2500915 0.01565866 0.008768001
## 2 0.01408315   3120.851  0.9887133  77.697707 0.3363900 0.01424391 0.012330425
## 3 0.01327932   6338.074  0.6020408   2.420552 0.4666110 0.02205717 0.012469397
## 4 0.01459760   7302.977  0.6513630   2.758841 0.4991305 0.02241085 0.013742269
## 5 0.01617311   1118.774  0.7216643   2.835551 0.2353235 0.02241085 0.011449691
##        lift oddsRatio
## 1  5.930599  17.64791
## 2  8.035007 704.85819
## 3 16.395859  61.60438
## 4 17.066588  75.62716
## 5  3.424025  10.43283
```

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 (i.e $conv(A \rightarrow B) \neq conv(B \rightarrow A)$). Unlike Confidence, the support of both antecedent and consequent are considered in conviction.