Code

Final exam-problem 2-solution

Deepak Sharma

2021-12-14

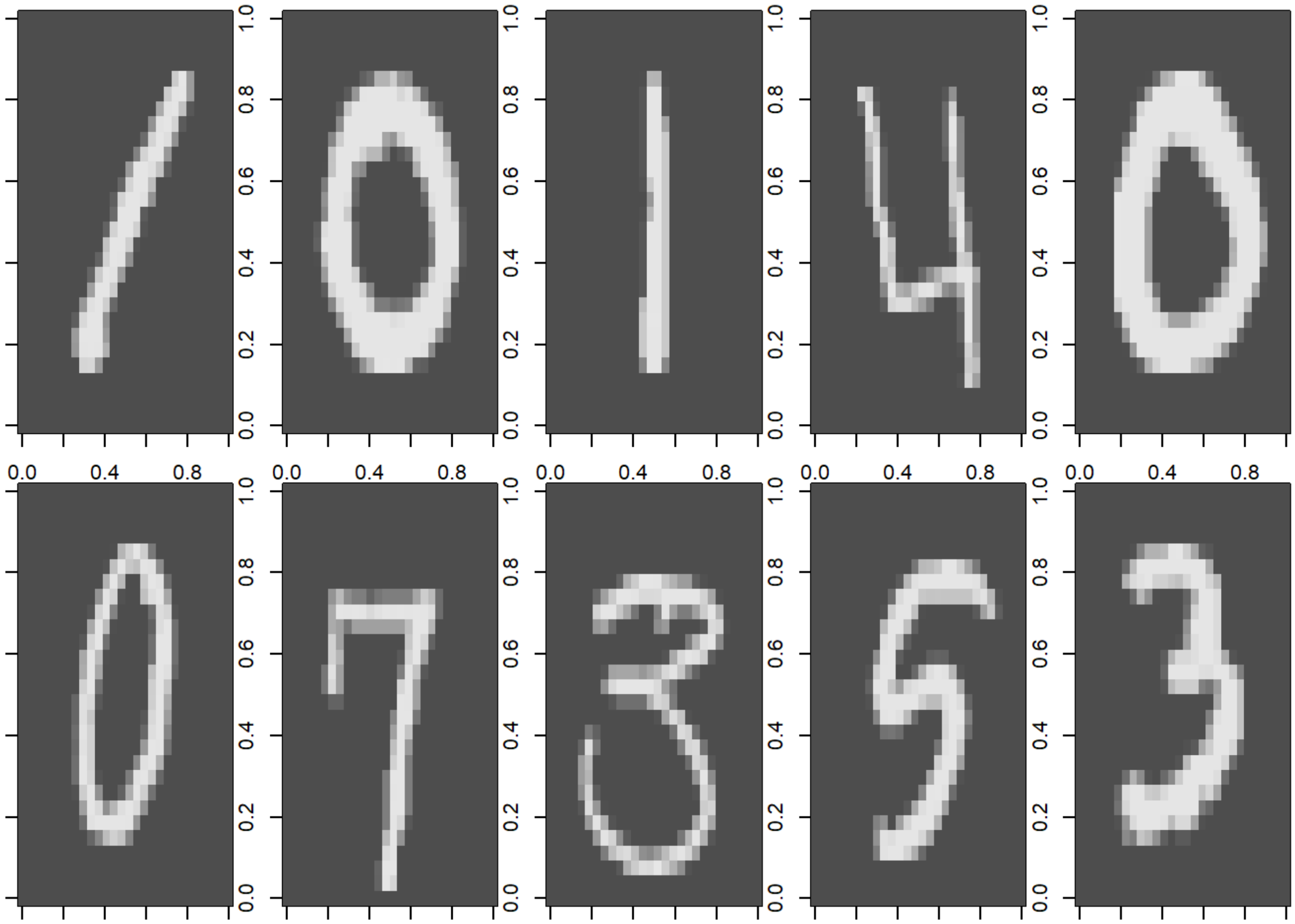
#Load the data from MNIST

Question 3

Using the training.csv file, plot representations of the first 10 images to understand the data format. Go ahead and divide all pixels by 255 to produce values between 0 and 1. (This is equivalent to min-max scaling.) (5 points)

Hide

```
par(mar=c(1,1,1,1))
par(mfrow=c(2,5))
for(i in 1:10)
{
  m<-matrix(unlist(train[i,-1]),nrow=28,byrow=TRUE)
  #image(m,col=grey.colors(255))
  rotate<- t(apply(m,2,rev))
  image(rotate,col=grey.colors(255))
}
```



Hide

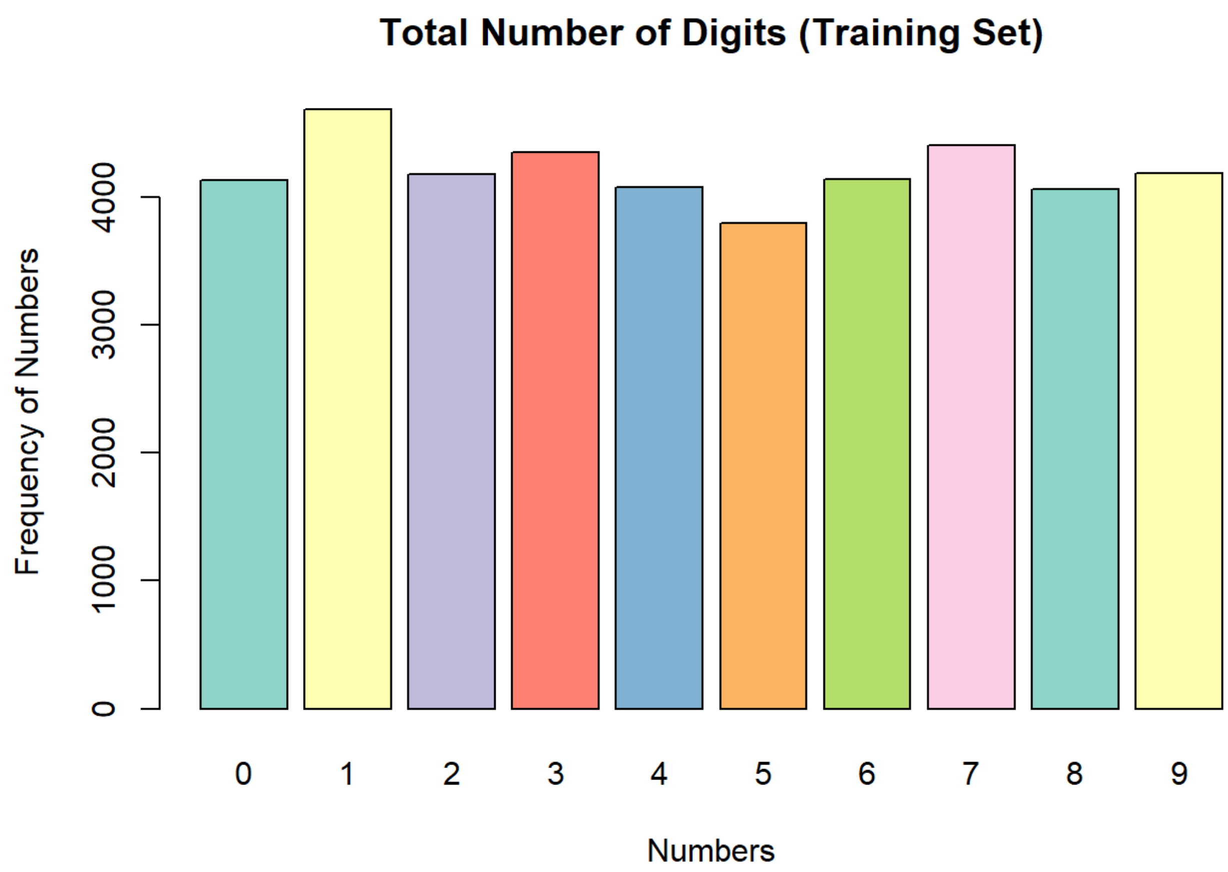
```
train_x <- train/255.0
test_x <- test/255.0
```

Question 4

What is the frequency distribution of the numbers in the dataset? (5 points)

Hide

```
#  the display.brewer.all function will plot all of them along with their name.
#display.brewer.all()
barplot(table(train$label), main="Total Number of Digits (Training Set)", col=brewer.pal(8
         ,"Set3"),
    xlab="Numbers", ylab = "Frequency of Numbers")
```



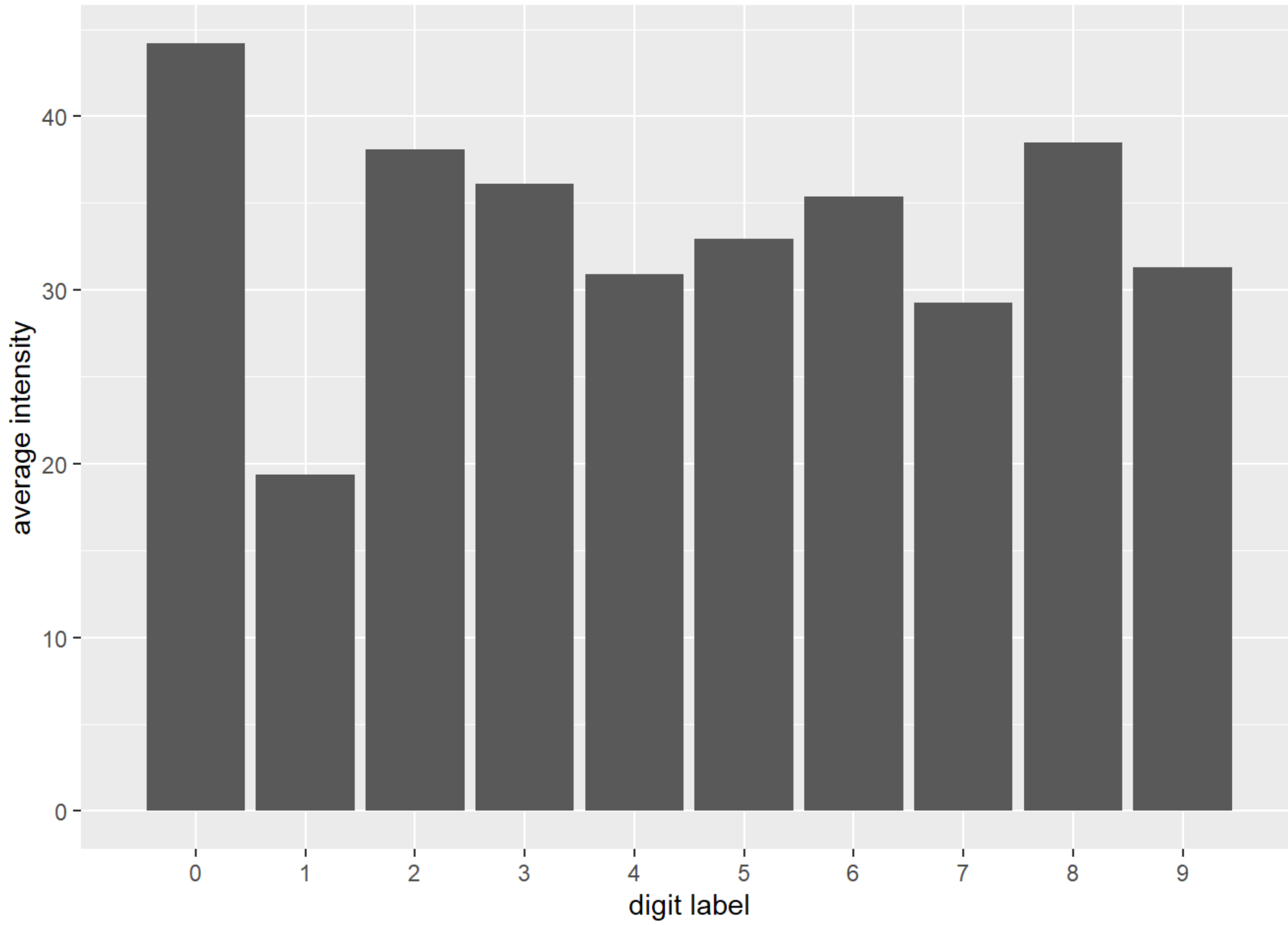
Question 5

For each number, provide the mean pixel intensity. What does this tell you? (5 points)

It does seem that the distributions for 4 and 7 are less “normal” than the distrubution for 1. The distribution for 4 looks almost bimodal - a telling sign thay perhaps there are two different ways people tend to write their fours. average intensity could have some predictive power and also that there is a lot of variability in the way people write digits

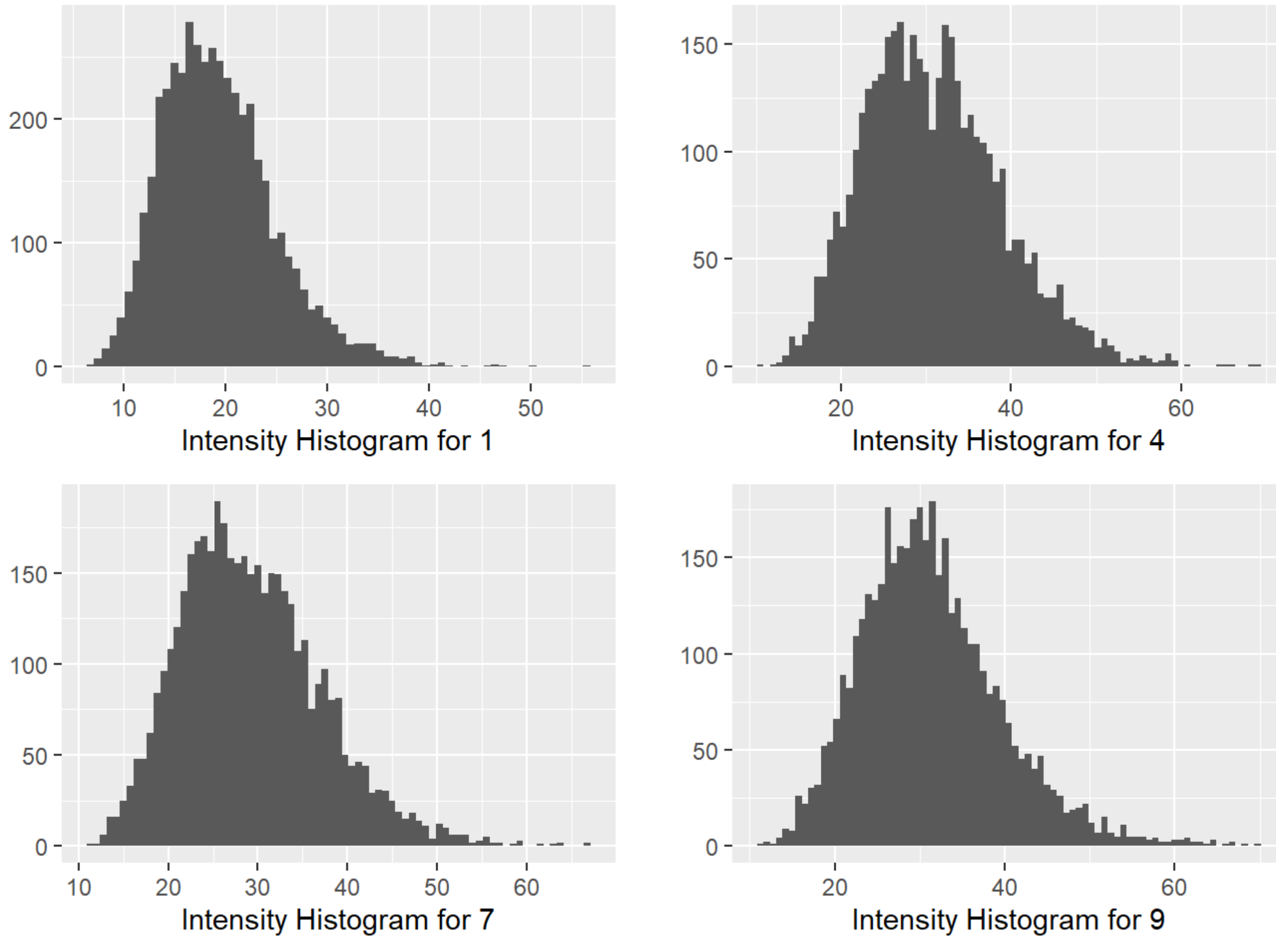
Hide

```
#average intensity
train$intensity <- apply(train[,-1], 1, mean) #takes the mean of each row in train

intbylabel <- aggregate (train$intensity, by = list(train$label), FUN = mean)

plot <- ggplot(data=intbylabel, aes(x=Group.1, y = x)) +
    geom_bar(stat="identity")
plot + scale_x_discrete(limits=0:9) + xlab("digit label") +
    ylab("average intensity")
```



Hide

```
#As we can see there are some differences in intensity. The digit “1” is the less intense
          while the digit “0” is the most intense. So this new feature seems to have some
#predictive value if you wanted to know if say your digit is a “1” or no



p1 <- qplot(subset(train, label ==1)$intensity, binwidth = .75,
            xlab = "Intensity Histogram for 1")

p2 <- qplot(subset(train, label ==4)$intensity, binwidth = .75,
            xlab = "Intensity Histogram for 4")

p3 <- qplot(subset(train, label ==7)$intensity, binwidth = .75,
            xlab = "Intensity Histogram for 7")

p4 <- qplot(subset(train, label ==9)$intensity, binwidth = .75,
            xlab = "Intensity Histogram for 9")

grid.arrange(p1, p2, p3,p4, ncol = 2)
```



Question 6

Reduce the data by using principal components that account for 95% of the variance. How many components did you generate? Use PCA to generate all possible components (100% of the variance). How many components are possible? Why? (5 points)

From these plots you can see that trainingdata has ~200 PC's that cumulatively explain ~95% of total variance.

From these plots you can see that trainingdata has ~400 PC's that cumulatively explain ~100% of total variance.

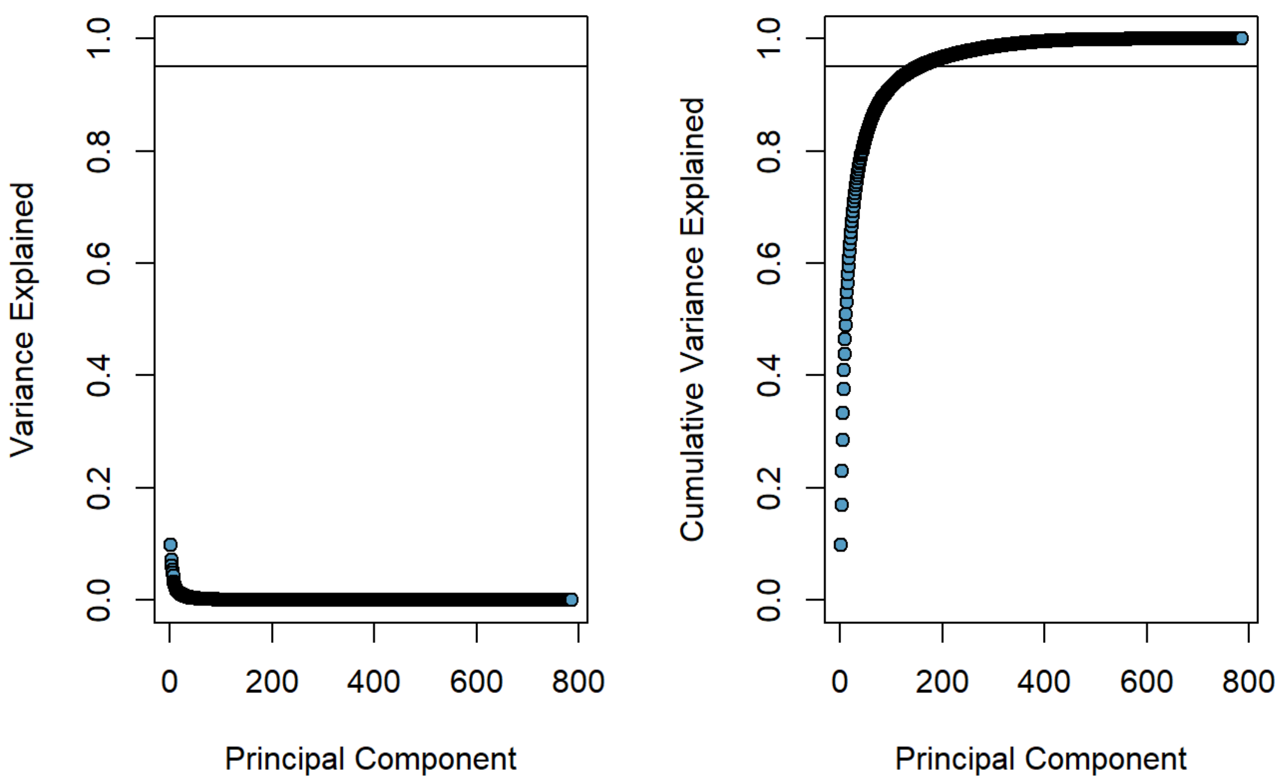
Hide

```
runPCA <- function(mat = 'Unadjusted matrix') eigen(cov(apply(mat, 2, function(i) i - mean
        (i))))
pca <- runPCA(train)
summary(pca)
```

```
##         Length Class  Mode
## values     786 -none- numeric
## vectors 617796 -none- numeric
```

Hide

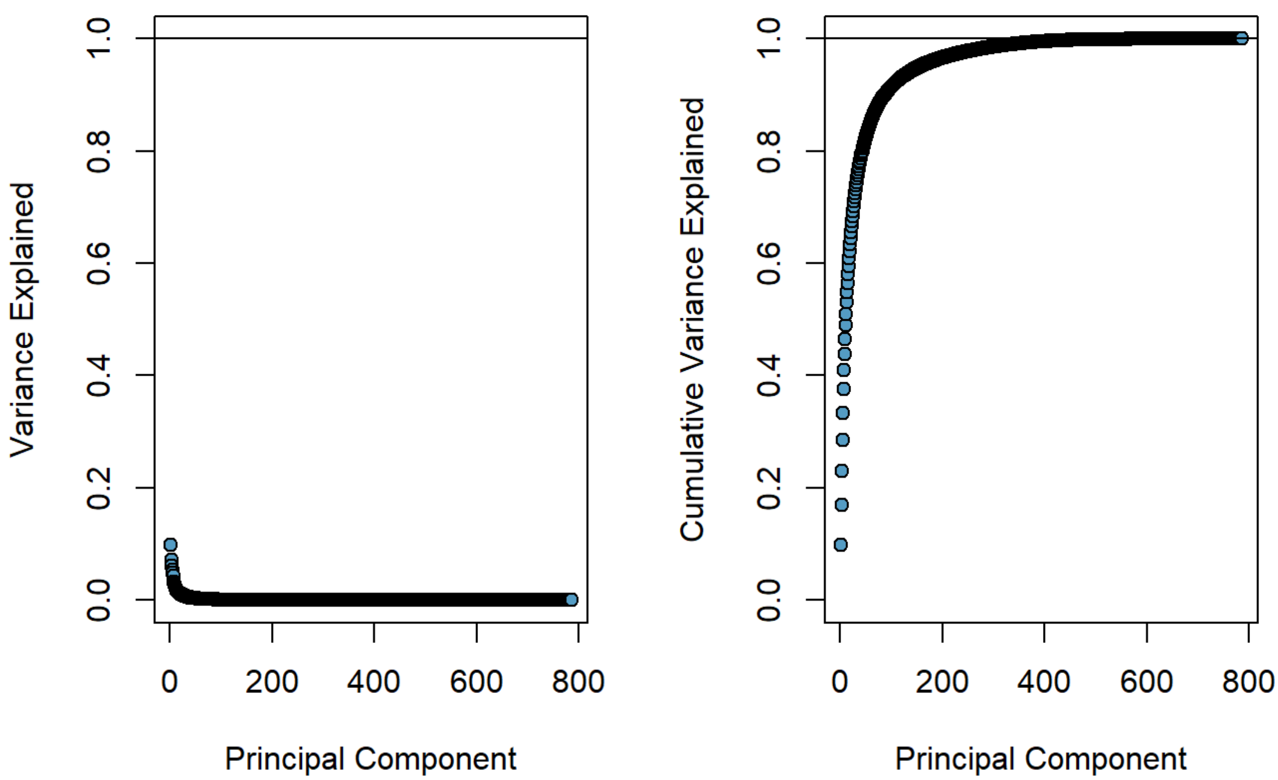
```r
varExplained <- function(eigenList) {
par(mfrow = c(1,2))
plot(
 eigenList$value / sum(eigenList$value), pch = 21, col = 'black',
 bg = '#549cc4', ylim = c(0, 1), xlab = 'Principal Component',
 ylab = 'Variance Explained'
 ) + abline(h = 0.95)
plot(
 cumsum(eigenList$value) / sum(eigenList$value), pch = 21,
 col = 'black', bg = '#549cc4', ylim = c(0, 1), xlab = 'Principal Component',
 ylab = 'Cumulative Variance Explained'
 ) + abline(h = 0.95)
}

varExplained(pca)
```




```
## integer(0)
```

Hide

```
varExplained_100 <- function(eigenList) {
par(mfrow = c(1,2))
plot(
 eigenList$value / sum(eigenList$value), pch = 21, col = 'black',
 bg = '#549cc4', ylim = c(0, 1), xlab = 'Principal Component',
 ylab = 'Variance Explained'
 ) + abline(h = 1)
plot(
 cumsum(eigenList$value) / sum(eigenList$value), pch = 21,
 col = 'black', bg = '#549cc4', ylim = c(0, 1), xlab = 'Principal Component',
 ylab = 'Cumulative Variance Explained'
 ) + abline(h = 1)
}

varExplained_100(pca)
```




```
## integer(0)
```

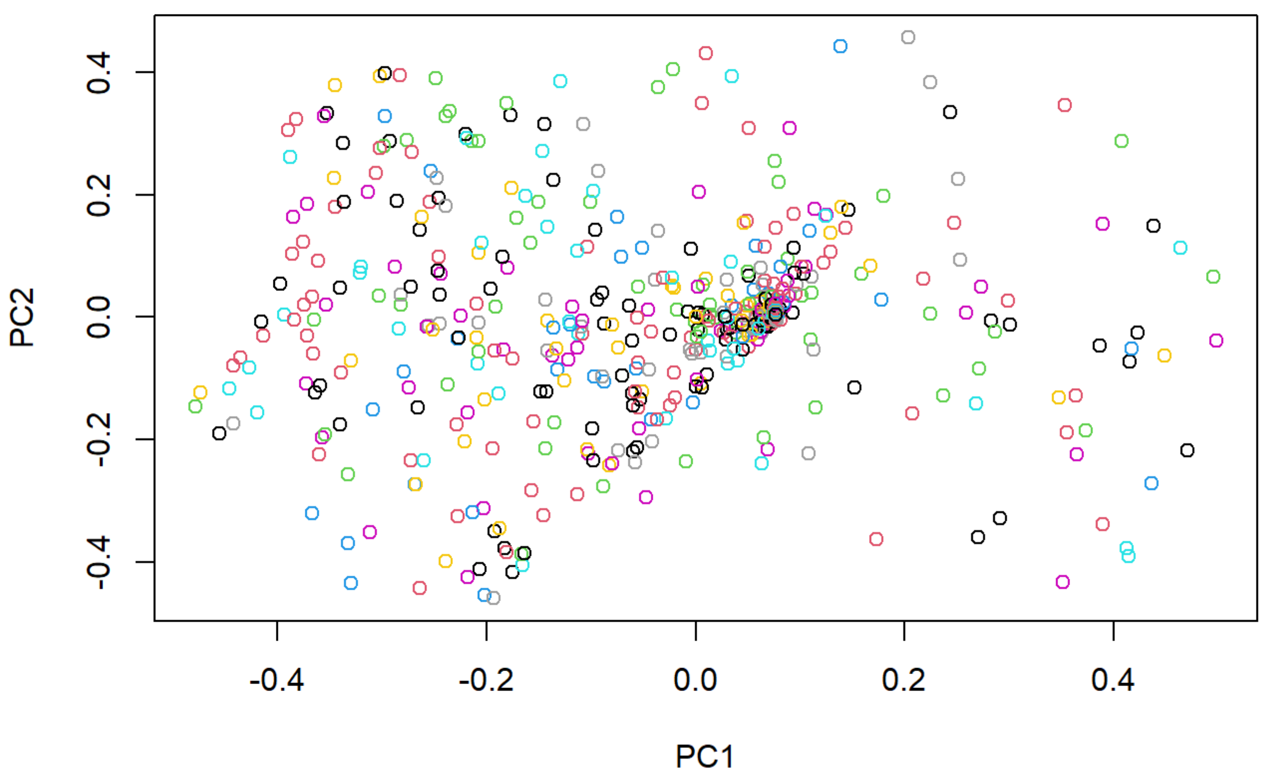
Question 7

Plot the first 10 images generated by PCA. They will appear to be noise. Why? (5 points)

##Answer Because there are some components of lower variance i.e, of lower eigenvalues(and the intent of PCA is to reduce that) .Because PCs of higher eigenvalues are capturing the more generalized features. As you are taking more and more PCs, the specialized features are also being added. If you take all of them the 100% of the data-variations will be restored like the original dimensions. So removing removing some PCs with lower eigenvalues actually acting as some sort of regularization and your model is only learning the more general features and not being confused by very fine detail which are likely not the general properties of that class. This is how overfitting is being prevented upto a certain level.

Hide

```
train_norm<-as.matrix(train[,-1])/255
train_norm_cov <- cov(train_norm)
pca <- prcomp(train_norm_cov)

labelClasses <- factor(train$label)
plot(main="",pca$x, col = labelClasses)
```



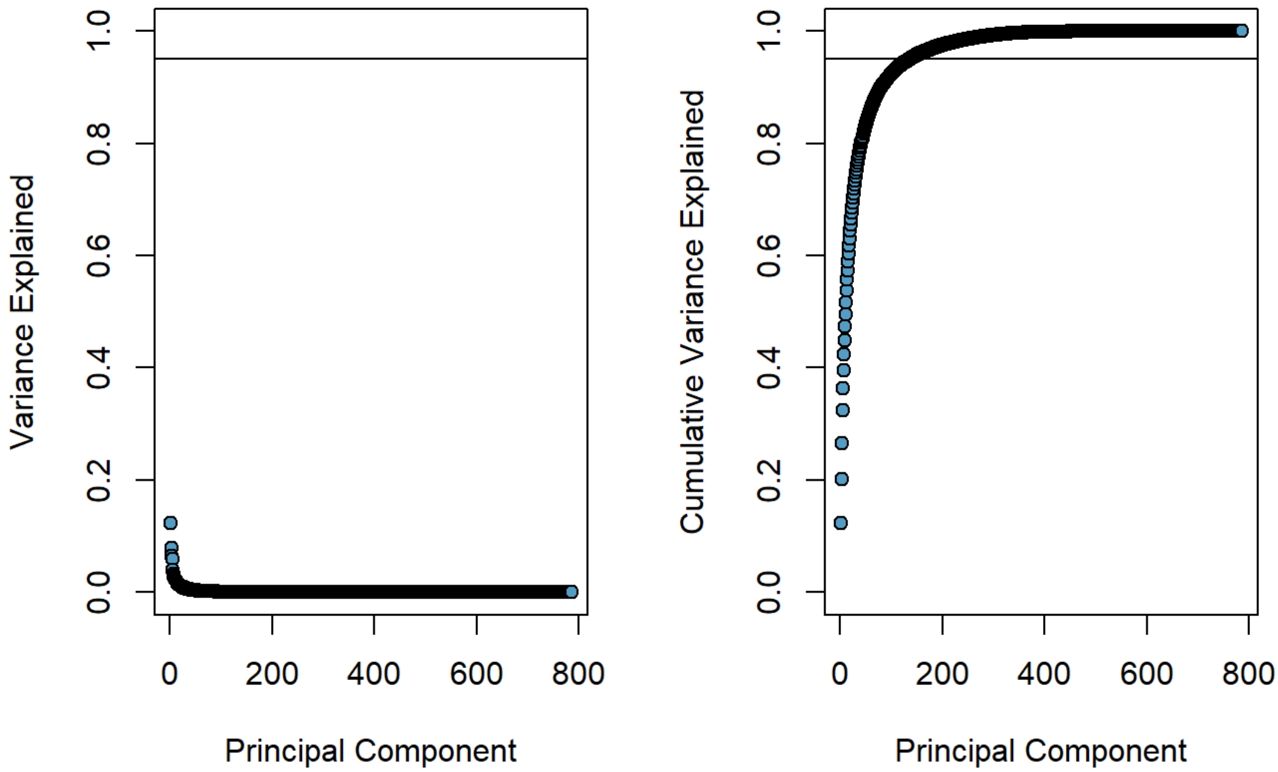
Question 8

Now, select only those images that have labels that are 8’s. Re-run PCA that accounts for all of the variance (100%). Plot the first 10 images. What do you see? (5 points)

#Answer: re-running the pca will give us the relative components but at the same time more distortion of the image will be there.We need to have a correct trade off.

Hide

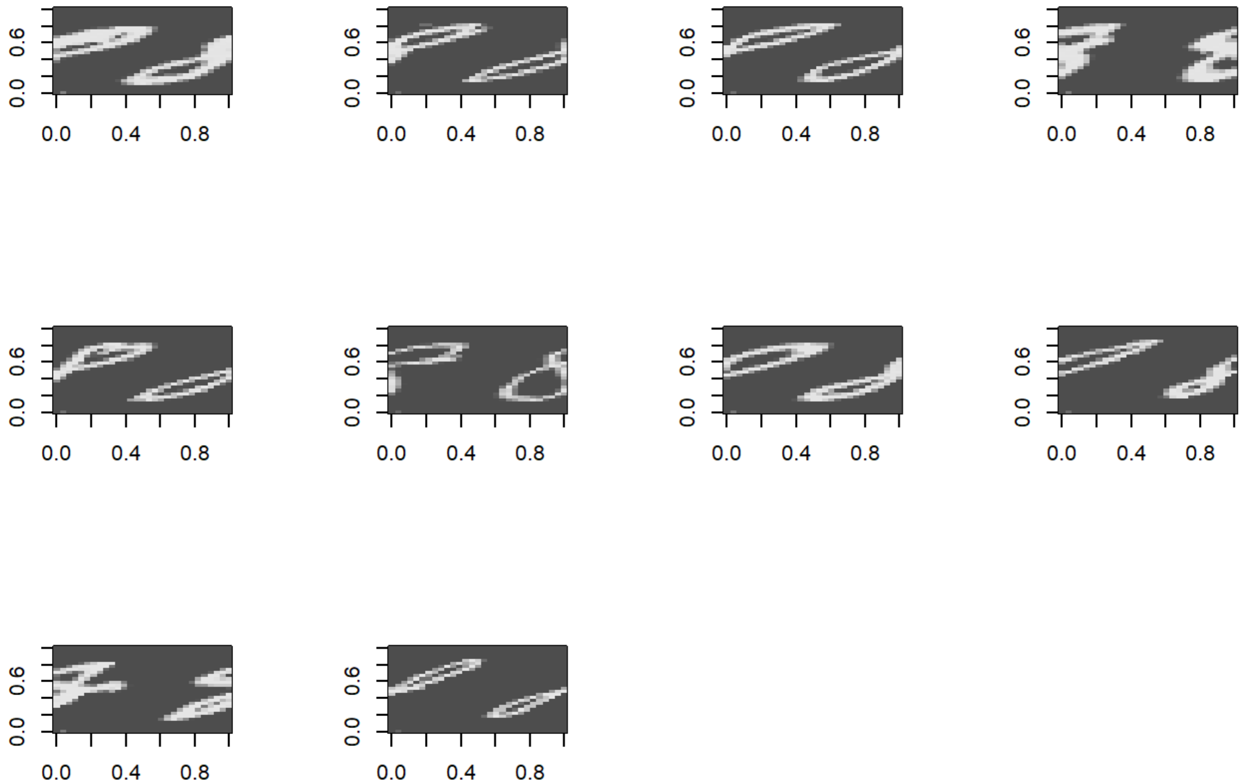
```
pcaDraw <- function(x) {
    x.var <- x$sdev ^ 2
    x.pvar <- x.var/sum(x.var)
    par(mfrow=c(1,1))
    plot(x.pvar,xlab="Principal component", ylab="Proportion of variance explained", ylim=
         c(0,1), type='b')
    plot(cumsum(x.pvar),xlab="Principal component", ylab="Cumulative Proportion of varianc
         e explained", ylim=c(0,1), type='b')
    screeplot(x,type="l")
    par(mfrow=c(1,1))
}

digit<-function(x){
  m<-matrix(unlist(x), nrow=28, byrow=T)
  m<-t(apply(m, 2, rev))
  image(m, col=grey.colors(255))
}


train_sub_8<-subset(train, label ==8)
#pca_8 <- prcomp(train_sub_8)
#pcaDraw(pca_8)

pca_8 <- runPCA(train_sub_8)

varExplained <- function(eigenList) {
par(mfrow = c(1,2))
plot(
 eigenList$value / sum(eigenList$value), pch = 21, col = 'black',
 bg = '#549cc4', ylim = c(0, 1), xlab = 'Principal Component',
 ylab = 'Variance Explained'
 ) + abline(h = 0.95)
plot(
 cumsum(eigenList$value) / sum(eigenList$value), pch = 21,
 col = 'black', bg = '#549cc4', ylim = c(0, 1), xlab = 'Principal Component',
 ylab = 'Cumulative Variance Explained'
 ) + abline(h = 0.95)
}

varExplained(pca_8)
```




```
## integer(0)
```

Hide

```
par(mfrow=c(3,4))
for(i in 1:10){
  digit(train_sub_8[i, -1])
}
```



Question 9

An incorrect approach to predicting the images would be to build a linear regression model with y as the digit values and X as the pixel matrix. Instead, we can build a multinomial model that classifies the digits. Build a multinomial model on the entirety of the training set. Then provide its classification accuracy (percent correctly identified) as well as a matrix of observed versus forecast values (confusion matrix). This matrix will be a 10 x 10, and correct classifications will be on the diagonal. (10 points)

Answer :

#Note:running the model takes lot of time with the whole data set so taken a small sample of data and ran the model on it.

I have used Gradient boosted trees model for multinomial which is a machine learning technique used in regression and classification tasks.Gradient boosted trees also run directly on the multiclass labels. It gives a prediction model in the form of an ensemble of weak prediction models. I could also play with the learning rate, but won't fiddle with that here for now. The model performs much better if I increase the interaction depth slightly. Increasing it past 2-3 is beneficial in large models.

Hide

```r
set.seed(222)
train <- read.csv("train1.csv")

sample_data = round(nrow(train)*.70) # setting what is 70%
index <- sample(seq_len(nrow(train)), size = sample_data)

train <- train[index, ]
test <- train[-index, ]

Xtrain <- as.matrix(train)
Xtest <- as.matrix(test)
ytrain <- train[,1]
ytest <- test[,1]

# Gradient boosted trees model for multinomial
outGbm <- gbm.fit(Xtrain,  factor(ytrain), distribution="multinomial", n.trees=500, intera
         ction.depth=2)
```

```
## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1        2.3026             nan     0.0010    0.0106
##      2        2.2959             nan     0.0010    0.0085
##      3        2.2904             nan     0.0010    0.0094
##      4        2.2845             nan     0.0010    0.0123
##      5        2.2772             nan     0.0010    0.0105
##      6        2.2707             nan     0.0010    0.0109
##      7        2.2641             nan     0.0010    0.0088
##      8        2.2583             nan     0.0010    0.0143
##      9        2.2505             nan     0.0010    0.0081
##     10        2.2451             nan     0.0010    0.0088
##     20        2.1836             nan     0.0010    0.0094
##     40        2.0787             nan     0.0010    0.0105
##     60        1.9798             nan     0.0010    0.0079
##     80        1.8915             nan     0.0010    0.0065
##    100        1.8083             nan     0.0010    0.0059
##    120        1.7348             nan     0.0010    0.0050
##    140        1.6663             nan     0.0010    0.0059
##    160        1.6018             nan     0.0010    0.0042
##    180        1.5429             nan     0.0010    0.0045
##    200        1.4861             nan     0.0010    0.0048
##    220        1.4328             nan     0.0010    0.0038
##    240        1.3833             nan     0.0010    0.0041
##    260        1.3349             nan     0.0010    0.0038
##    280        1.2898             nan     0.0010    0.0034
##    300        1.2468             nan     0.0010    0.0031
##    320        1.2058             nan     0.0010    0.0037
##    340        1.1667             nan     0.0010    0.0029
##    360        1.1284             nan     0.0010    0.0030
##    380        1.0917             nan     0.0010    0.0032
##    400        1.0564             nan     0.0010    0.0021
##    420        1.0225             nan     0.0010    0.0027
##    440        0.9893             nan     0.0010    0.0022
##    460        0.9577             nan     0.0010    0.0025
##    480        0.9269             nan     0.0010    0.0028
##    500        0.8977             nan     0.0010    0.0023
```

Hide

```
predGbm <- apply(predict(outGbm, Xtest, n.trees=outGbm$n.trees),1,which.max) - 1L
# Prediction
predGbm
```

```
##  [1] 3 0 3 6 5 1 6 1 0 1 8 8 2 6 5 3 6 4 1 8 6 2 6 3 9 7 2 3 3 3 4 3 0 3 9 6 1 2
## [39] 1 1 2 8 8 2 2 1 4 4 9 9 9 6 3 5 0 8 0 7 3 6 9 6 8 0 0 2 8 7 9 0 3 0 5 4 6 0
## [77] 3 2 3 4 6 0
```

I would like to try another machine learning model for prediction and confusion matrix to see if there are improved predictions, lets try Support vector machines for clear confusion matrix. We can give the multiclass problem directly to the support vector machine, and one-vs-one prediction is done on all combinations of the classes. I found the radial kernel performed the best for this type of data class

Hide

```
outSvm <- svm(Xtrain,  factor(ytrain), kernel="radial", cost=1)
predSvm <- predict(outSvm, Xtest)

# Prediction
predSvm
```

```
## 254  18 151  22 266  97  90 141 197  78 407 319 411 375 368 108 400  40  36 136
##   3   0   3   6   5   1   6   1   0   1   8   8   2   6   5   5   6   4   1   8
## 217  98 134 193 369 337 272 312 182  37 391 199 158 380  34  92 183  57 257 210
##   6   2   6   3   9   7   2   3   3   3   4   3   0   3   9   6   1   2   1   1
## 362  11 396 384  85  62 259 185 308 181 114  75   8 382 376 291 286  77 418 283
##   2   8   8   2   2   1   4   4   9   9   9   6   3   5   0   8   0   7   3   6
## 166 124  31 374 358  35 301 309  41 274 156 189 163 379 341  64 213 236 167 171
##   9   6   8   0   0   2   8   7   9   0   3   0   5   4   6   0   3   2   3   4
## 353 111
##   6   0
## Levels: 0 1 2 3 4 5 6 7 8 9
```

Based on the confusion matrix below, there seems to be a correlation between 3 and 6. It may be because both images has light intensity on either of the side in transformed image. We also see that 8 and 3 are particularly difficult, with 1 being quite easy to predict.

Hide

```
# Confusion Matrix
table(predSvm,ytest)
```

```
##        ytest
## predSvm  0  1  2  3  4  5  6  7  8  9
##       0 11  0  0  0  0  0  0  0  0  0
##       1  0  8  0  0  0  0  0  0  0  0
##       2  0  0  9  0  0  0  0  0  0  0
##       3  0  0  0 13  0  0  0  0  0  0
##       4  0  0  0  0  6  0  0  0  0  0
##       5  0  0  0  0  0  5  0  0  0  0
##       6  0  0  0  0  0  0 12  0  0  0
##       7  0  0  0  0  0  0  0  3  0  0
##       8  0  0  0  0  0  0  0  0  8  0
##       9  0  0  0  0  0  0  0  0  0  7
```