Financial Data Mining
Homework 2
Yen-Hsiu Chang

6. PCA

```
rm(list=ls(all=TRUE))
# Data
load("hw1.RData")
dim(train2); dim(train3)
```

```
## [1] 731 256
```

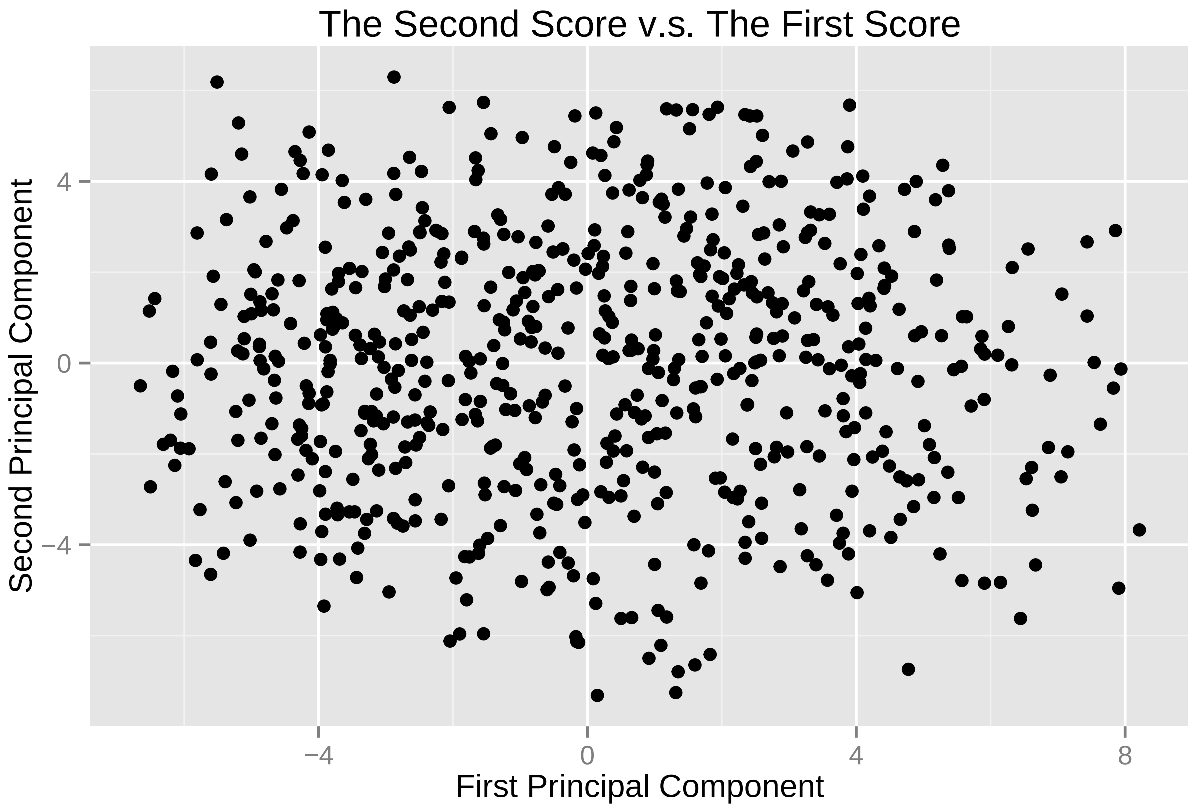
```
## [1] 658 256
```

(a)

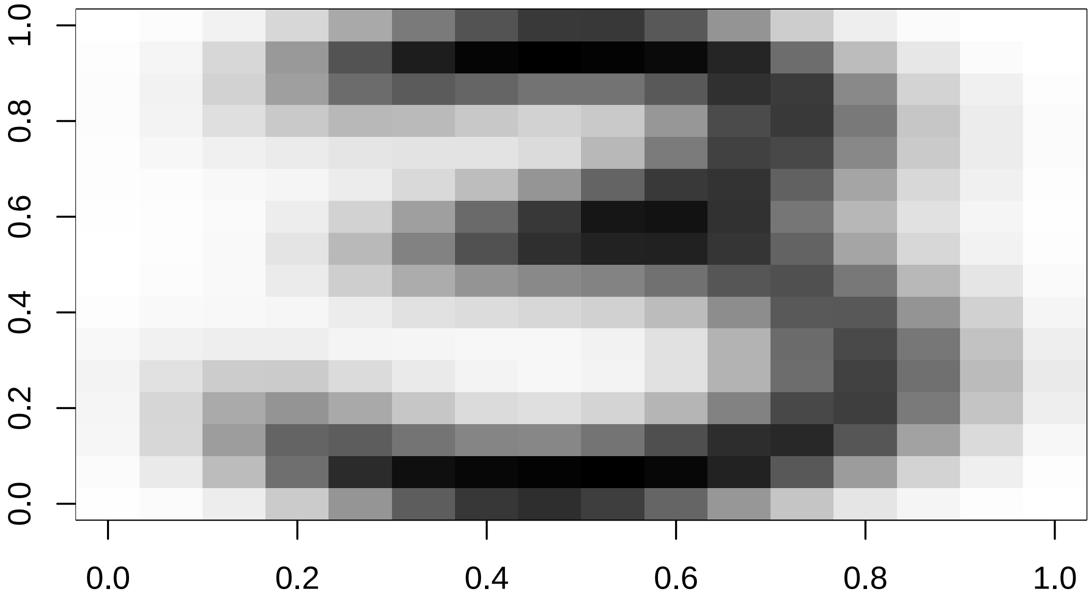
```
prc.3=prcomp(train3)
names(prc.3)
```

```
## [1] "sdev"     "rotation" "center"   "scale"    "x"
```

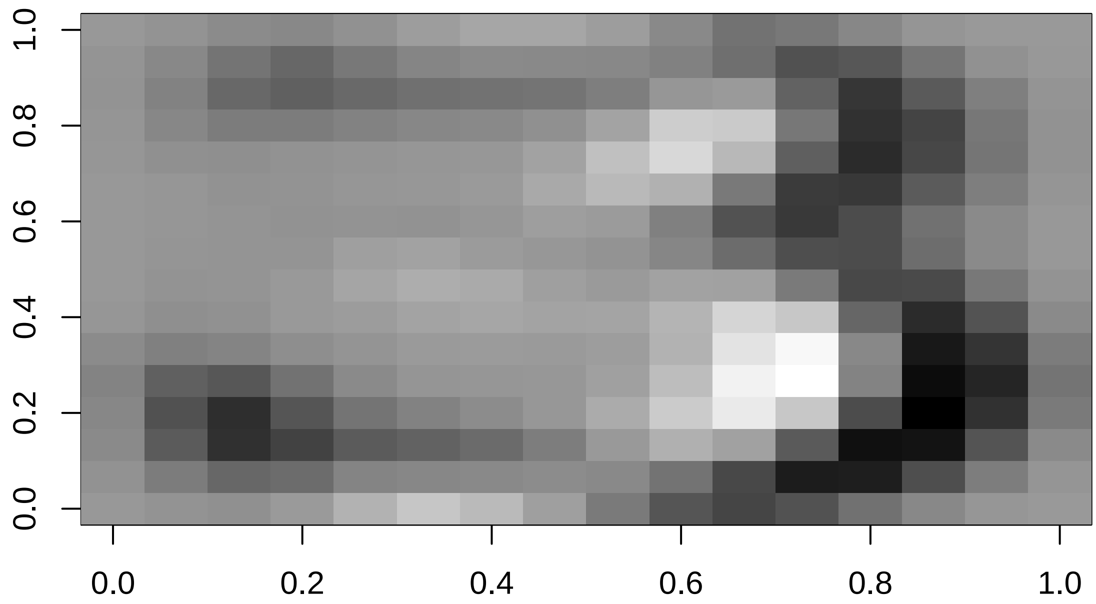
```
library(ggplot2)
data.plot <-data.frame(first=prc.3$x[,1],second=prc.3$x[,2])
p <- ggplot(data.plot,aes(x=first,y=second))+
  geom_point(pch=19)+
  labs(x="First Principal Component",y="Second Principal Component",title=
         "The Second Score v.s. The First Score")
p
```



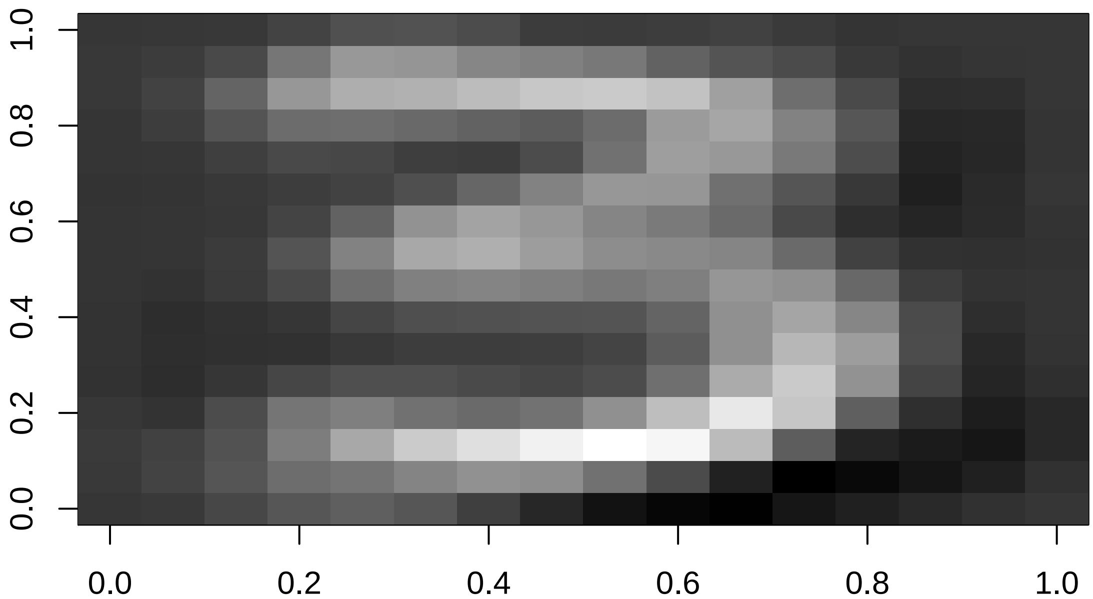

```r
library(graphics)
image3.center <- matrix(prc.3$center,nrow=16)
image3.center <- image3.center[,dim(image3.center)[2]:1]
image(image3.center,col=grey(1000:0/1000))
```



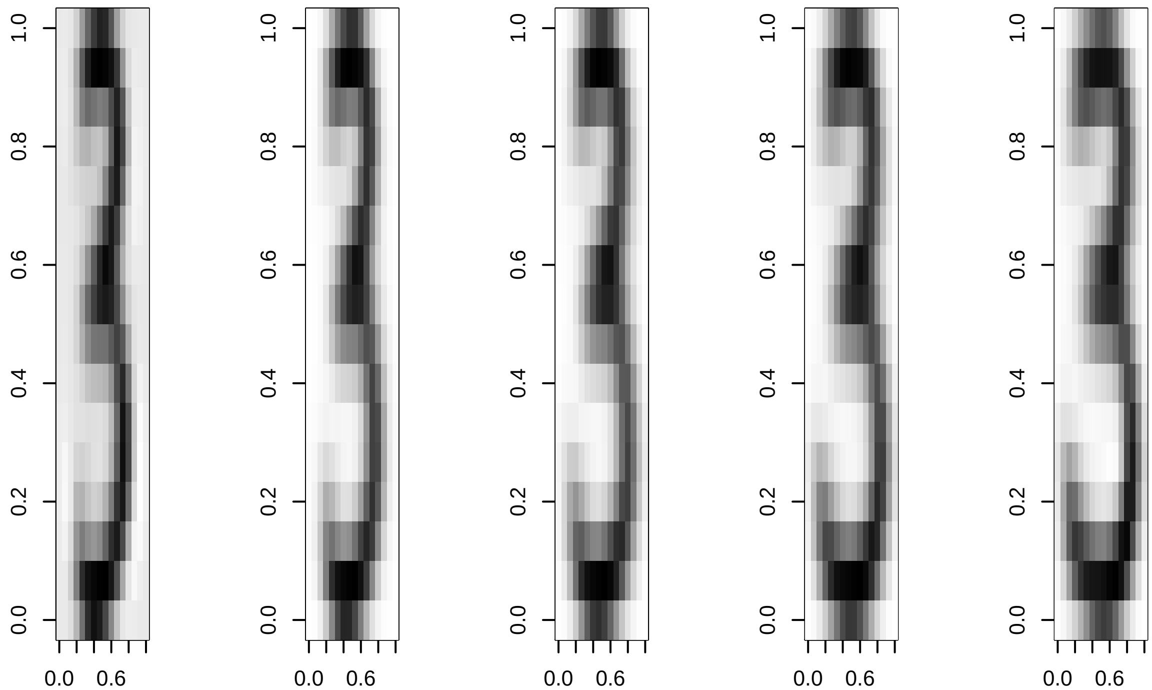

```
image3.1st.loading <- matrix(prc.3$rotation[,1],nrow=16)
image3.1st.loading <- image3.1st.loading[,dim(image3.1st.loading)[2]:1]
image(image3.1st.loading,col=grey(1000:0/1000))
```



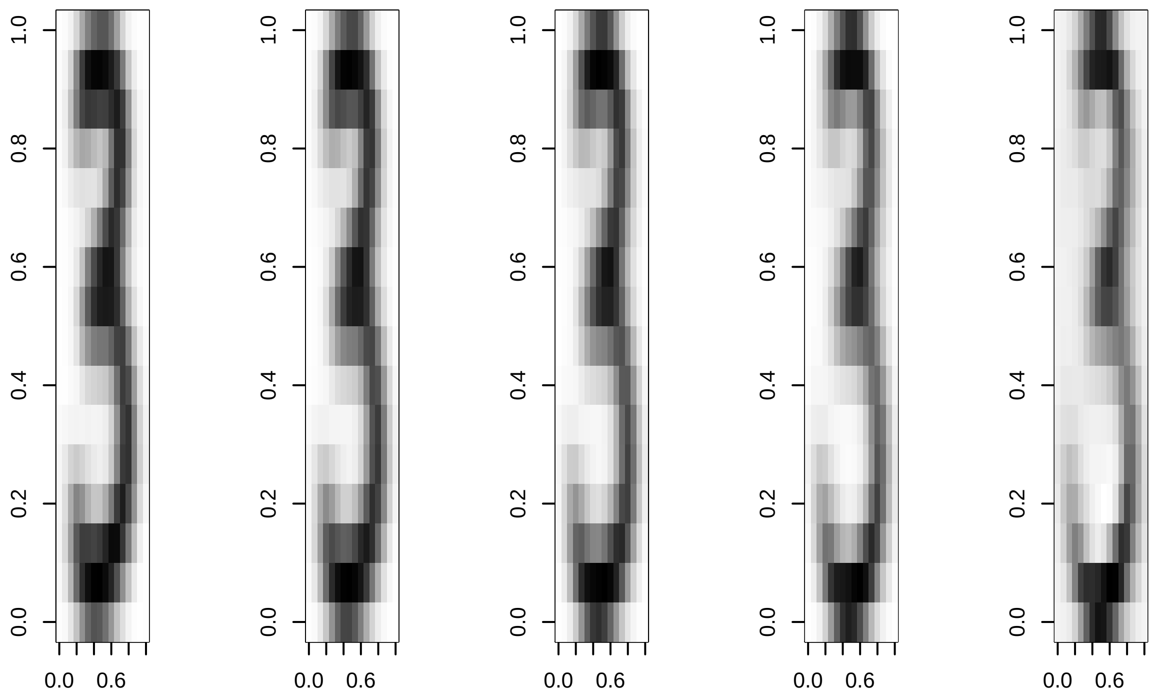

```r
image3.2nd.loading <- matrix(prc.3$rotation[,2],nrow=16)
image3.2nd.loading <- image3.2nd.loading[,dim(image3.2nd.loading)[2]:1]
image(image3.2nd.loading,col=grey(1000:0/1000))
```




```r
par(mfrow=c(1,5))
# First Principal Component
for(i in seq(-4,4,length=5)){
test.image <- matrix(prc.3$center+i*prc.3$rotation[,1],nrow=16)
test.image <- test.image[,dim(test.image)[2]:1]
image(test.image,col=grey(1000:0/1000))
}
```



```
# Second Principal Component
for(i in seq(-4,4,length=5)){
test.image <- matrix(prc.3$center+i*prc.3$rotation[,2],nrow=16)
test.image <- test.image[,dim(test.image)[2]:1]
image(test.image,col=grey(1000:0/1000))
}
```

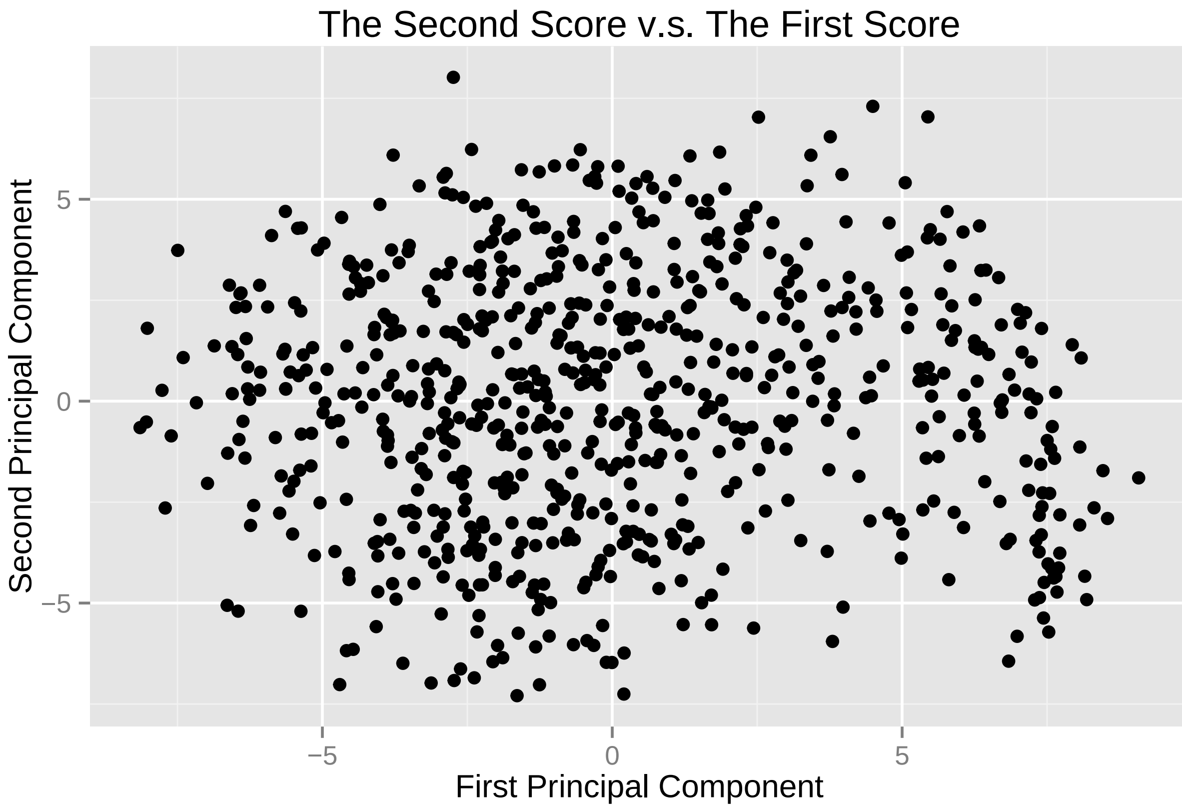



```
par(mfrow=c(1,1))
```

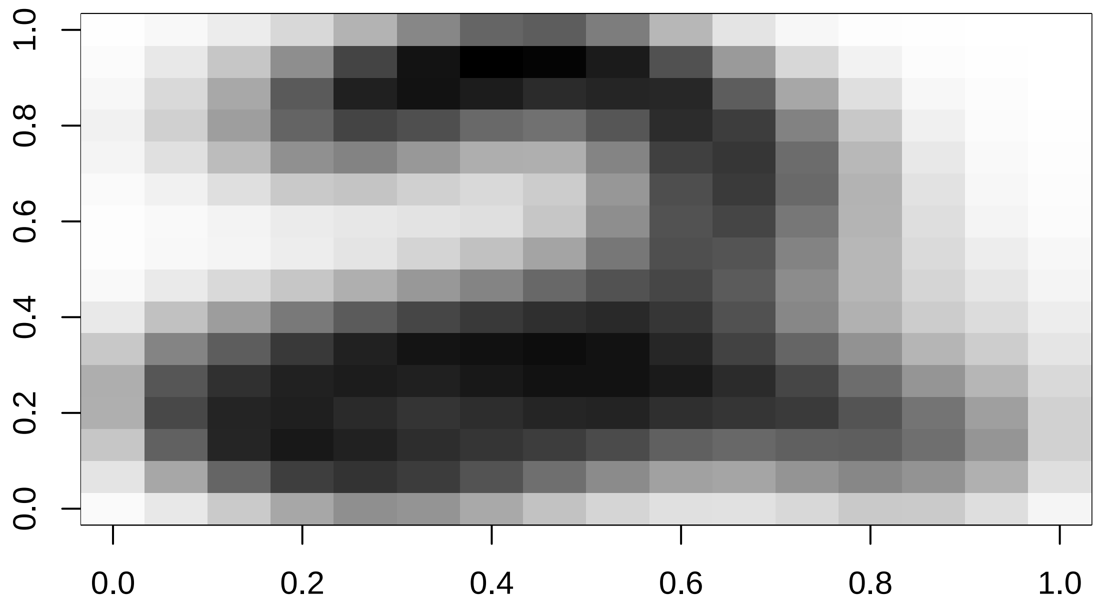
The fisrt loading component accounts for the lengthening of the lower tail of the three, while the second loading component accounts for character thickness.

(b)

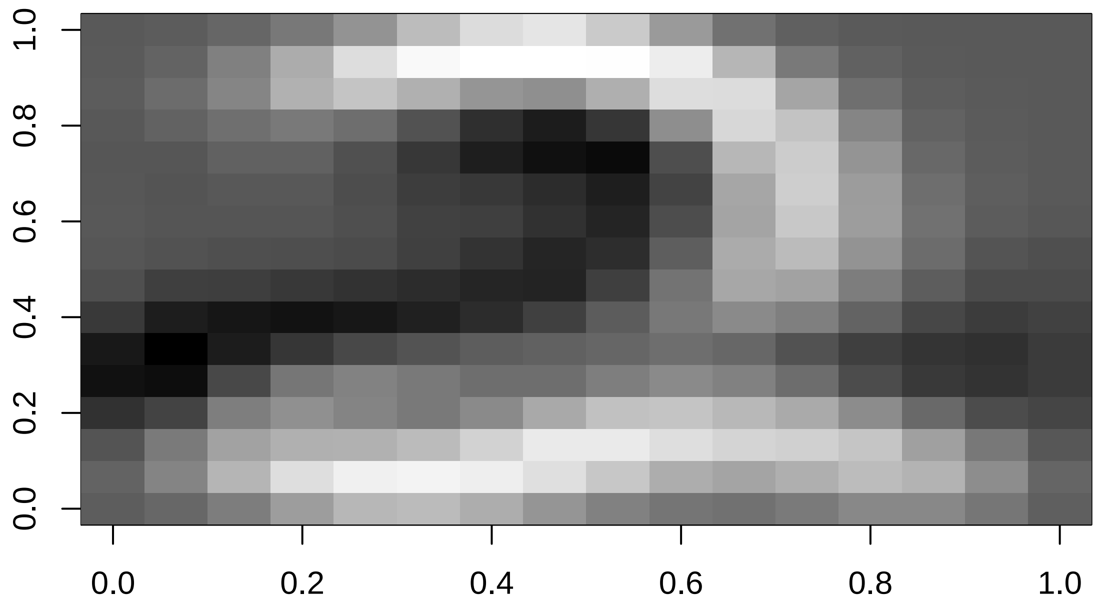
```
prc.2=prcomp(train2)

data.plot <-data.frame(first=prc.2$x[,1],second=prc.2$x[,2])
p <- ggplot(data.plot,aes(x=first,y=second))+
  geom_point(pch=19)+
  labs(x="First Principal Component",y="Second Principal Component",title=
         "The Second Score v.s. The First Score")
p
```



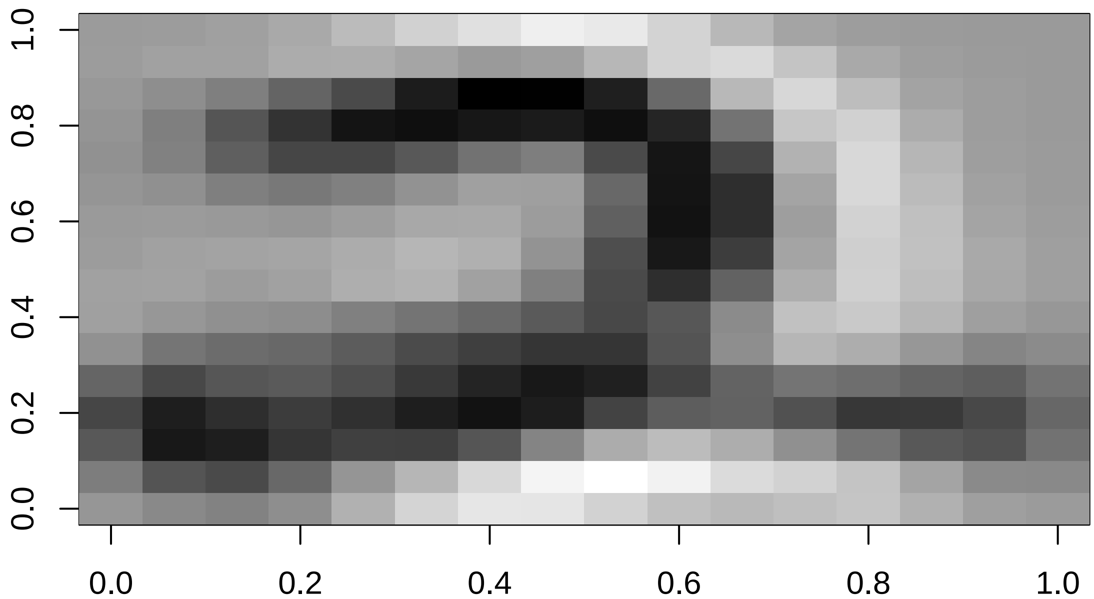

```
image2.center <- matrix(prc.2$center,nrow=16)
image2.center <- image2.center[,dim(image2.center)[2]:1]
image(image2.center,col=grey(1000:0/1000))
```



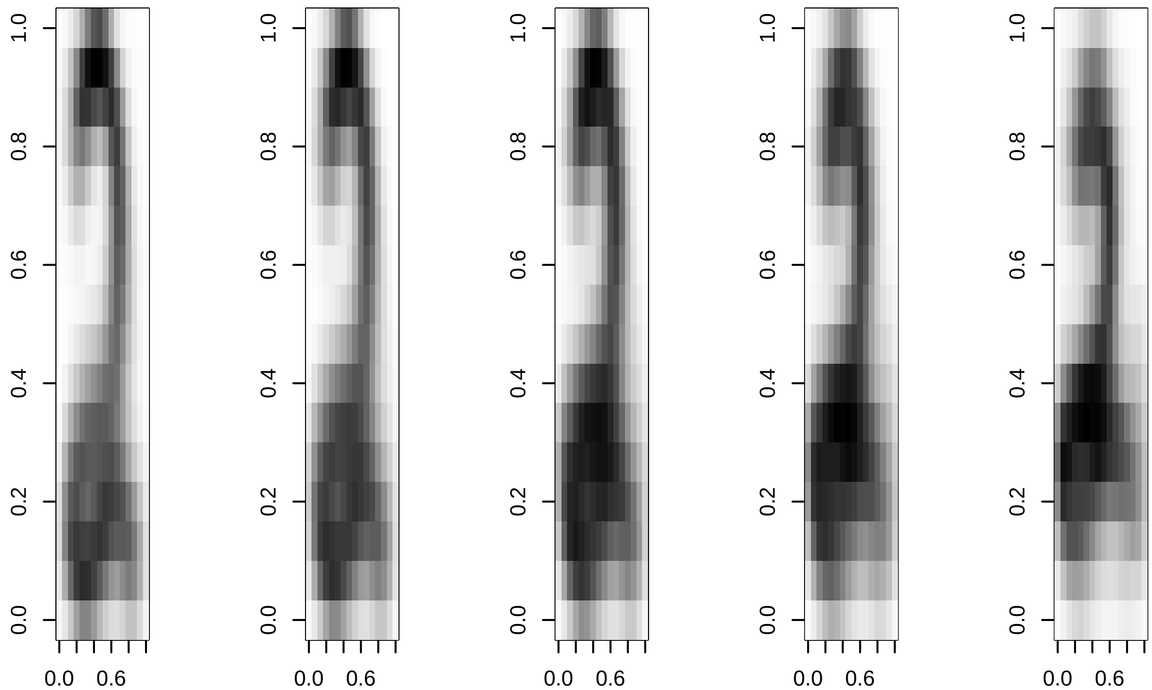

```
image2.1st.loading <- matrix(prc.2$rotation[,1],nrow=16)
image2.1st.loading <- image2.1st.loading[,dim(image2.1st.loading)[2]:1]
image(image2.1st.loading,col=grey(1000:0/1000))
```



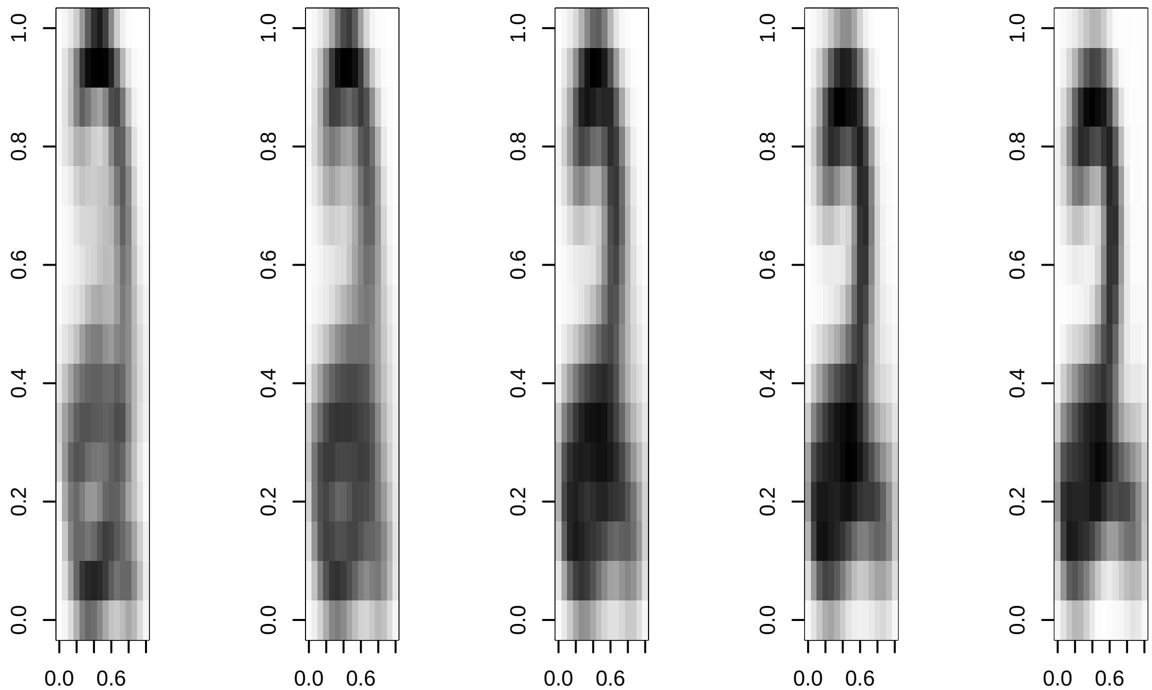

```
image2.2nd.loading <- matrix(prc.2$rotation[,2],nrow=16)
image2.2nd.loading <- image2.2nd.loading[,dim(image2.2nd.loading)[2]:1]
image(image2.2nd.loading,col=grey(1000:0/1000))
```



```
par(mfrow=c(1,5))
# First Principal Component
for(i in seq(-4,4,length=5)){
test.image <- matrix(prc.2$center+i*prc.2$rotation[,1],nrow=16)
test.image <- test.image[,dim(test.image)[2]:1]
image(test.image,col=grey(1000:0/1000))
}
```




```
# Second Principal Component
for(i in seq(-4,4,length=5)){
test.image <- matrix(prc.2$center+i*prc.2$rotation[,2],nrow=16)
test.image <- test.image[,dim(test.image)[2]:1]
image(test.image,col=grey(1000:0/1000))
}
```

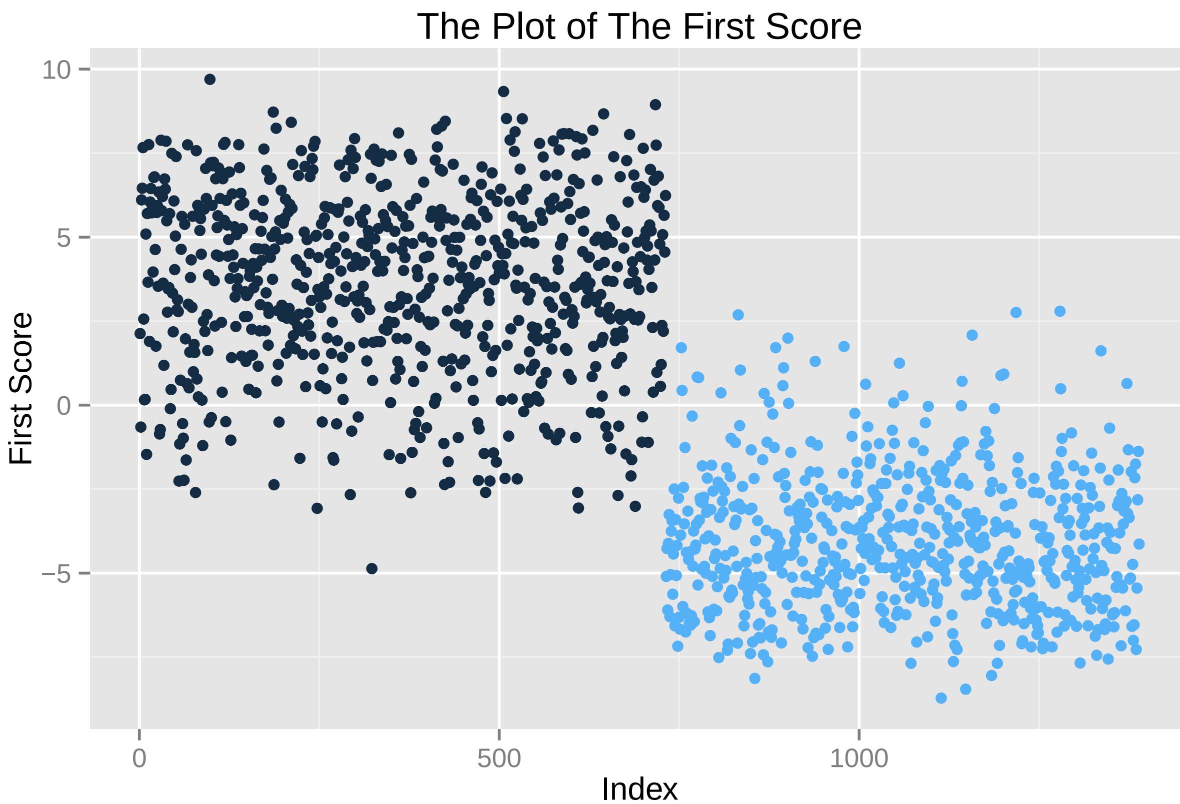



```
par(mfrow=c(1,1))
```

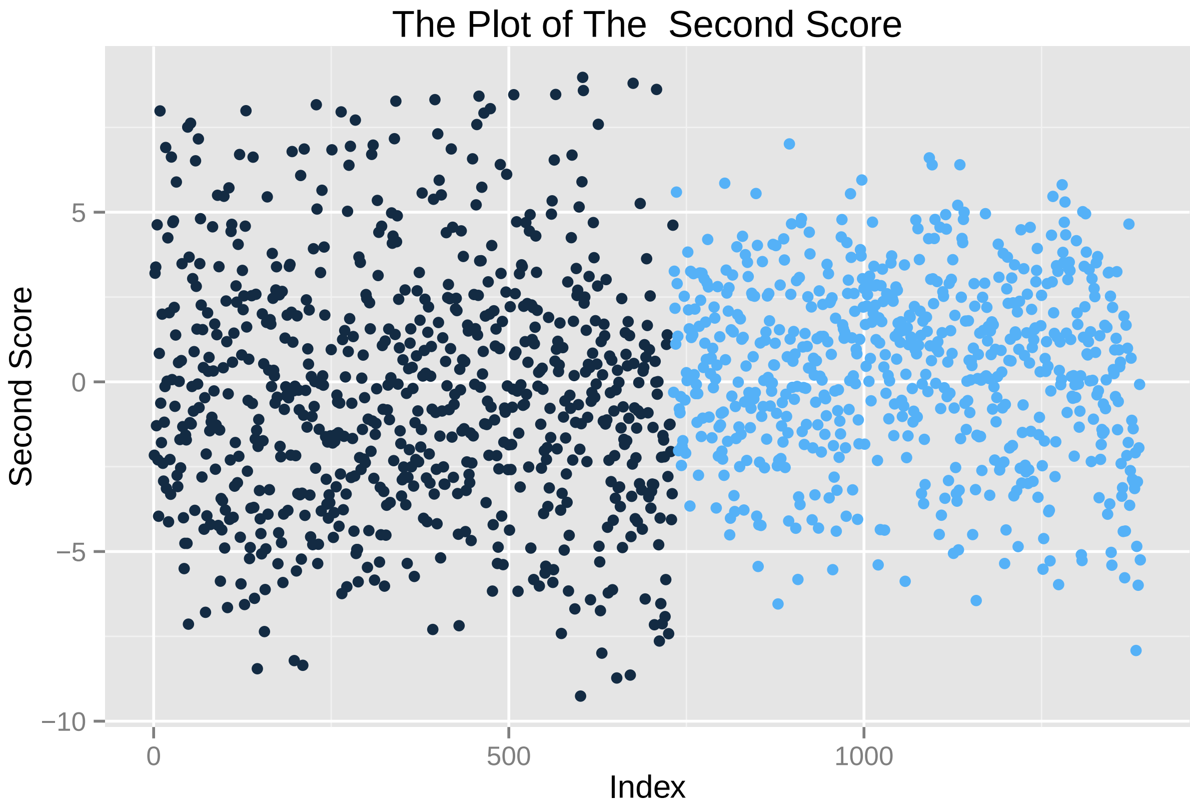
The fisrt loading component accounts for the size of the two, while the second loading component accounts for character thickness.

(c)

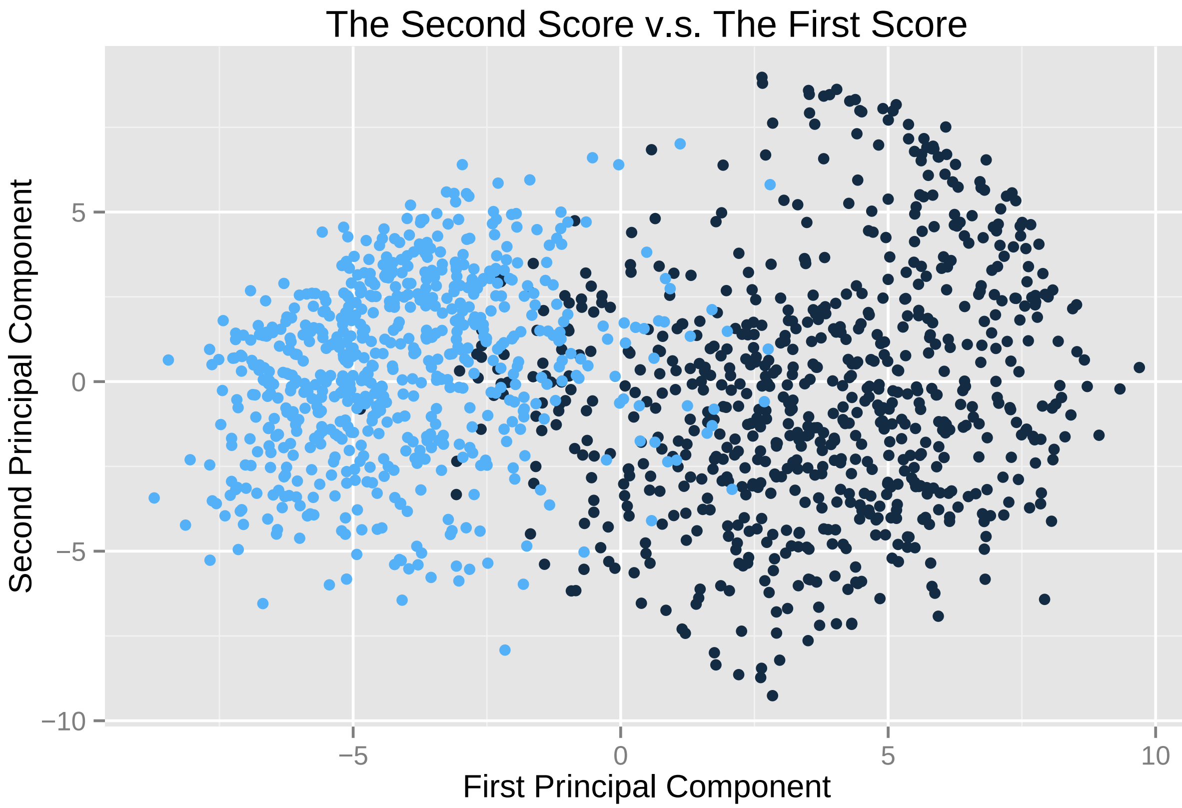
```
y.train <- c(rep(0,dim(train2)[1]),rep(1,dim(train3)[1]))
x.train <- rbind(train2,train3)
y.test <- c(rep(0,dim(test2)[1]),rep(1,dim(test3)[1]))
x.test <- rbind(test2,test3)

prc.23 <- prcomp(x.train)
# Plot
len <- length(prc.23$x[,1])
plot.data <- data.frame(cbind(1:len,prc.23$x[,1],y.train))
p1 <- ggplot(plot.data,aes(x=V1,y=V2,colour=y.train))+
  geom_point(colours=y.train)+
  scale_color_gradient(guide=F)+
  labs(x="Index",y="First Score",title="The Plot of The First Score")
p1
```



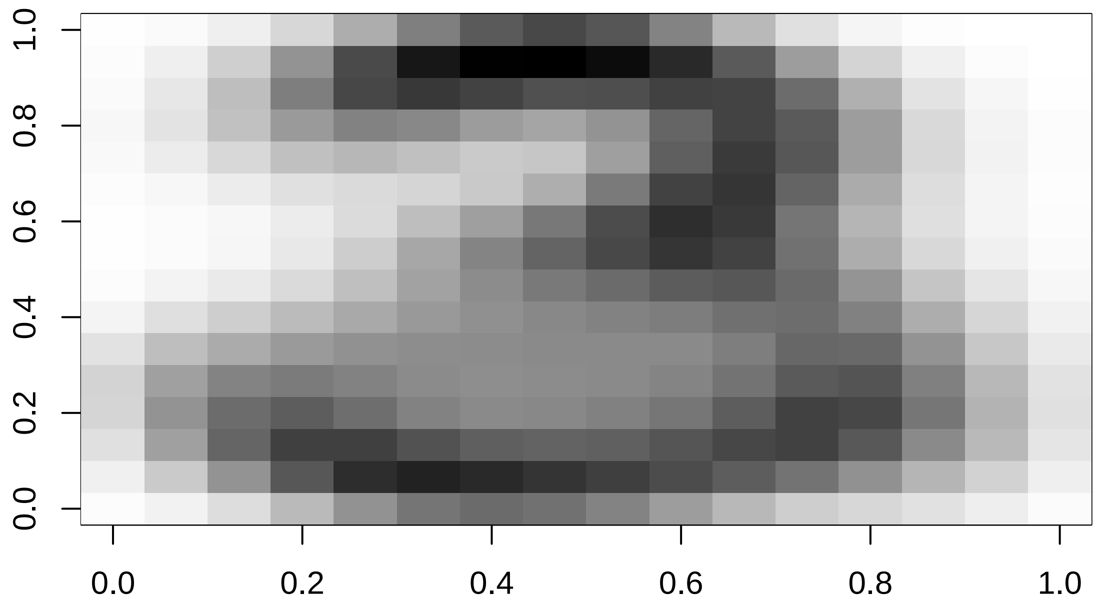

```
plot.data <- data.frame(cbind(1:len,prc.23$x[,2],y.train))
p2 <- ggplot(plot.data,aes(x=V1,y=V2,colour=y.train))+
  geom_point(colours=y.train)+
  scale_color_gradient(guide=F)+
  labs(x="Index",y="Second Score",title="The Plot of The  Second Score")
p2
```



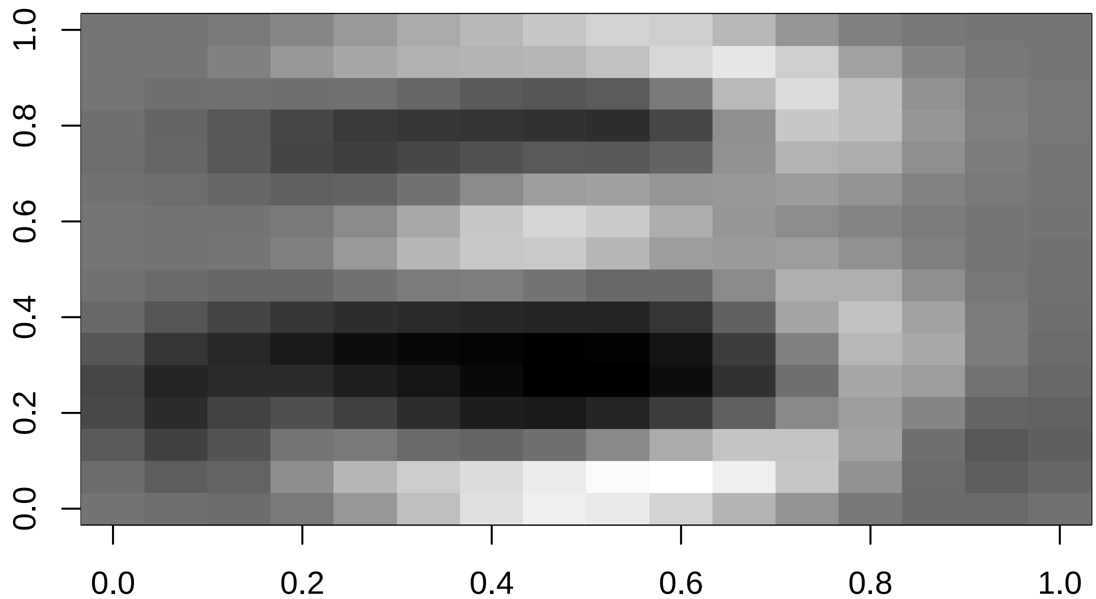

```
plot.data <- data.frame(x=prc.23$x[,1],y=prc.23$x[,2],y.train)
p3 <- ggplot(plot.data,aes(x,y,colour=y.train))+
  geom_point()+
  scale_color_gradient(guide=F)+
  labs(x="First Principal Component",y="Second Principal Component",title=
         "The Second Score v.s. The First Score")
p3
```



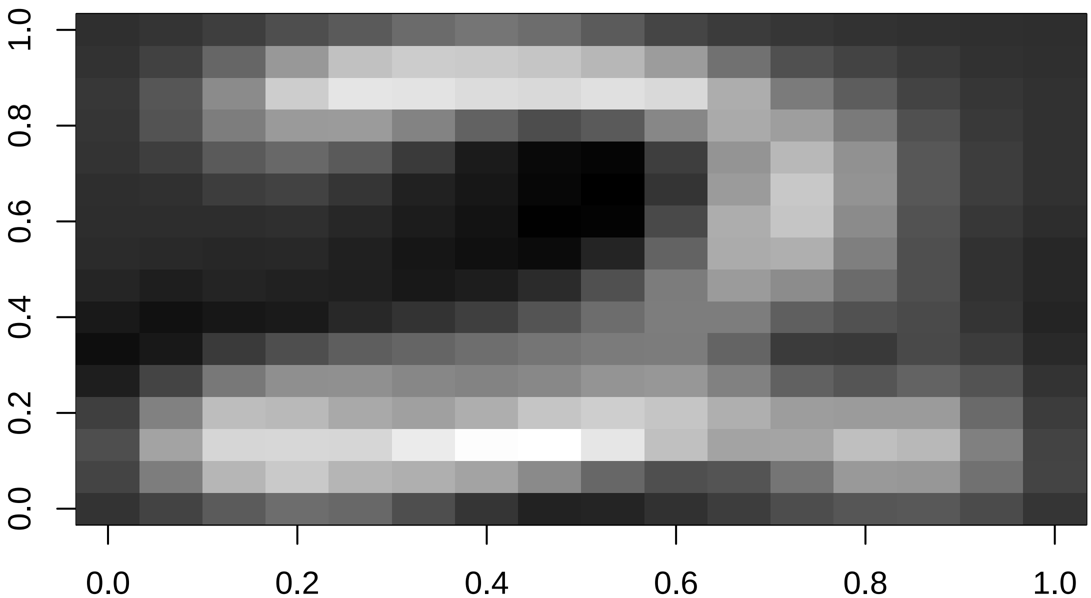

```r
# Show images
image23.center <- matrix(prc.23$center,nrow=16)
image23.center <- image23.center[,dim(image23.center)[2]:1]
image(image23.center,col=grey(1000:0/1000))
```




```
image23.1st.loading <- matrix(prc.23$rotation[,1],nrow=16)
image23.1st.loading <- image23.1st.loading[,dim(image23.1st.loading)[2]:1]
image(image23.1st.loading,col=grey(1000:0/1000))
```




```r
image23.2nd.loading <- matrix(prc.23$rotation[,2],nrow=16)
image23.2nd.loading <- image23.2nd.loading[,dim(image23.2nd.loading)[2]:1]
image(image23.2nd.loading,col=grey(1000:0/1000))
```



We can easily observe that the first score can distinguish these two numbers. Also, the image of mean looks like the combination of 2 and 3. And, from the image of the first loading vector, we can know that if the first score goes up, the image will become more like two. This is consistent with the plot of the second score vs the first score.

7. CV for OLS

(a)

```
apply(x.train,2,var)[16]
```

```
##           V16
## 6.479482e-09
```

```
x.train <- x.train[,-16]
library(boot)
train.data <- as.data.frame(cbind(x.train,y.train))
glm.fit <- glm(y.train~.-1,data=train.data)
set.seed(1)
cv.err <- cv.glm(train.data,glm.fit,K=5)
cv.err$delta[1]
```

```
## [1] 0.04329293
```

CV error without adjustment is 0.04329293.

(b)

```r
set.seed(1)
cv.err.class <- cv.glm(train.data,glm.fit,cost=function(y, y.hat) mean((y.hat>.5)!=y),K=5)
cv.err.class$delta[1]
```

```
## [1] 0.03023758
```

CV error now is 0.03023758.

(c)

```r
x.test <- x.test[,-16]
y.hat.test <- x.test%*%coef(glm.fit)
true.test.error <- mean((y.hat.test>.5)!=y.test);true.test.error
```

```
## [1] 0.03846154
```

The true test error is 0.03846154. No, CV from part(b) doesn't produce a reasonable estimate.

8. CV for KNN

(a) and (b)

```r
knn <- function(klist,x.train,y.train,x.test) {
    # k-nearest neighbors classification

    # klist is a list of values of k to be tested
    # x.train, y.train: the training set
    # x.test: the test set
    # Output: a matrix of predictions for the test set (one column for each k in klist)
    # Number of training and test examples
    n.train <- nrow(x.train)
    n.test <- nrow(x.test)
    # Matrix to store predictions
    p.test <- matrix(NA, n.test, length(klist))
    # Vector to store the distances of a point to the training points
    dsq <- numeric(n.train)
    # Loop on the test instances
    for (tst in 1:n.test)
    {
        # Compute distances to training instances
        for (trn in 1:n.train)
        {
            dsq[trn] <- sum((x.train[trn,] - x.test[tst,])^2)
        }
        # Sort distances from smallest to largest
        ord <- order(dsq)
        # Make prediction by averaging the k nearest neighbors
        for (ik in 1:length(klist)) {
                p.test[tst,ik] <- mean(y.train[ord[1:klist[ik]]])
```

```r
        }
    }
    # Return the matrix of predictions
    invisible(p.test)
}
knn.cv <- function(klist,x.train,y.train,nfolds) {
    # Cross-validation for kNN
    #
    # Perform nfolds-cross validation of kNN, for the values of k in klist

    # Number of instances
    n.train <- nrow(x.train)
    # Matrix to store predictions
    p.cv <- matrix(NA, n.train, length(klist))
    # Prepare the folds
    s <- split(sample(n.train),rep(1:nfolds,length=n.train))
    # Cross-validation
    for (i in seq(nfolds)) {
        p.cv[s[[i]],] <- knn(klist,x.train[-s[[i]],], y.train[-s[[i]]], x.train[s[[i]],])
    }
    # Return matrix of CV predictions
    invisible(p.cv)
}
klist <- seq(1,21,by=2)
# test error
y.pred.test <- knn(klist,x.train,y.train,x.test)
test.error.knn <- apply((y.pred.test>0.5)!=y.test,2,mean)
# train error
y.pred.train <- knn(klist,x.train,y.train,x.train)
train.error.knn <- apply((y.pred.train>0.5)!=y.train,2,mean)
# CV error
nfolds <- 5
set.seed(1)
y.pred.cv <- knn.cv(klist,x.train,y.train,nfolds)
cv.error <- apply((y.pred.cv>0.5)!=y.train,2,mean)

ind <- which.min(cv.error);ind
```

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

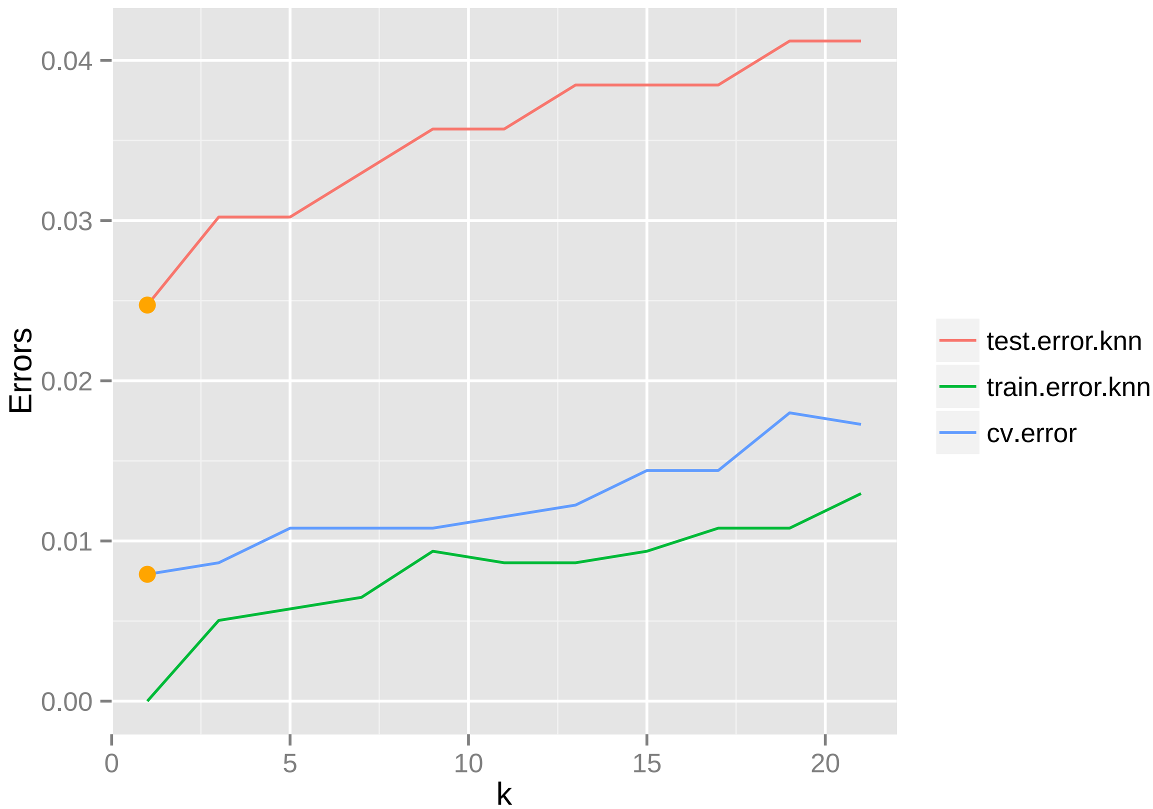
```r
cv.error[ind]; test.error.knn[ind]
```

```
## [1] 0.007919366
```

```
## [1] 0.02472527
```

```r
# Plot errors against k
data.plot <- as.data.frame(cbind(klist,test.error.knn,train.error.knn,cv.error))
library(reshape)
data.plot <- melt(data.plot,id="klist")
data.point <- data.frame(x=c(klist[ind],klist[ind]),y=c(cv.error[ind],test.error.knn[ind]))
library(ggplot2)
```

```r
p <- ggplot(data.plot,aes(x=klist,y=value,colour=variable))+
  geom_line()+
  geom_point(data=data.point,aes(x,y),size=3,colour="orange")+
  scale_color_hue(name="")+
  labs(x="k",y="Errors")
p
```



According to the CV error, the best k is 1. The CV error and test error corresponding to that K are 0.007919366 and 0.02472527,respectively.

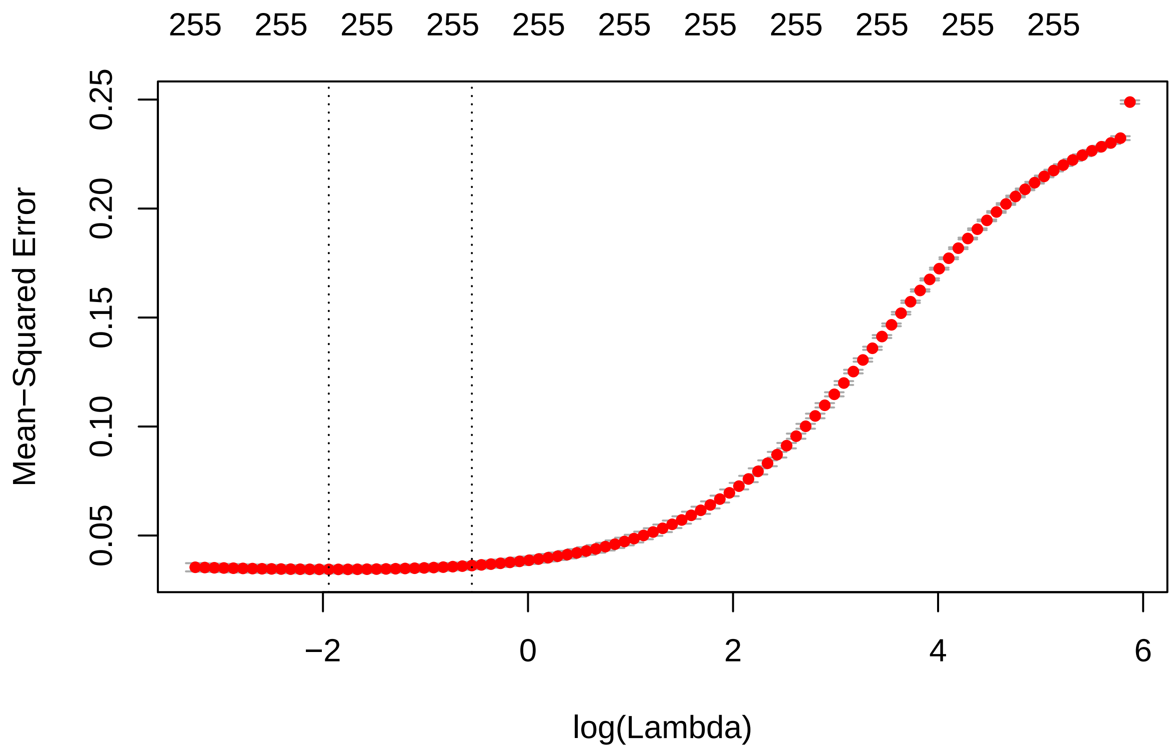
9. CV for Ridge

(a)

```r
library(glmnet)
```

```
## Warning: package 'glmnet' was built under R version 3.2.2

## Warning: package 'foreach' was built under R version 3.2.2
```

```r
# lambda.grid <- 10^seq(4,-3,length=100)
set.seed(1)
cv.out = cv.glmnet(x.train,y.train,alpha=0,nfolds=5)
plot(cv.out)
```




```r
lamda.min = cv.out$lambda.min; lamda.min
```

```
## [1] 0.1432573
```

```r
ridge.mod <- glmnet(x.train,y.train,alpha=0,standardize=F)
y.pred.test.ridge1 <- predict(ridge.mod,s=lamda.min,newx=x.test,exact=TRUE)
ridge.test.err1 <- mean((y.pred.test.ridge1>.5)!=y.test);ridge.test.err1
```

```
## [1] 0.02472527
```

```r
lambda.se = cv.out$lambda.1se; lambda.se
```

```
## [1] 0.5783322
```

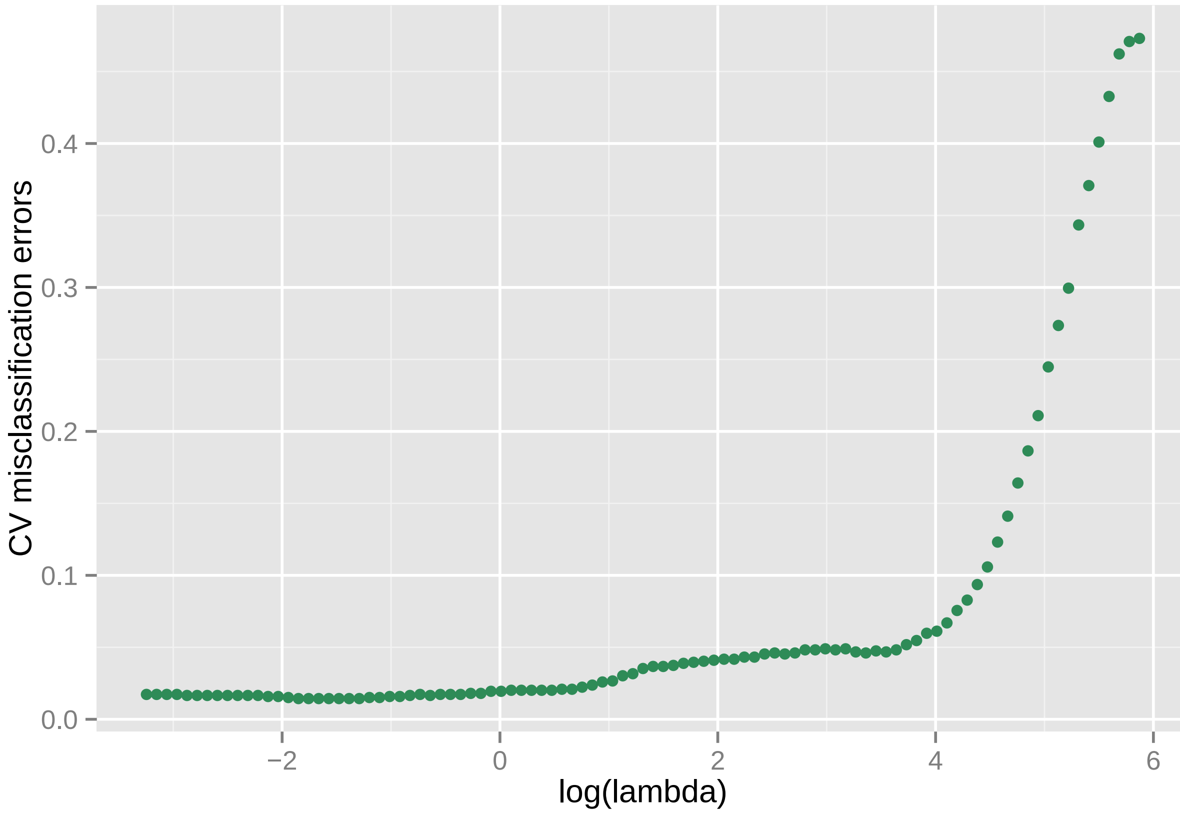
```r
y.pred.test.ridge2 <- predict(ridge.mod,s=lambda.se,newx=x.test,exact=TRUE)
ridge.test.err2 <- mean((y.pred.test.ridge2>.5)!=y.test);ridge.test.err2
```

```
## [1] 0.03021978
```

The λ that has the smallest cross-validated MSE is 0.1432573 and the true test misclassification error for this λ is 0.02472527. The best λ according to "One-Standard Error" Rule is 0.5783322 and the true test misclassification error for this λ is 0.03021978.

(b)

```r
ridge.cv <- function(lambda,x.train,y.train,nfolds) {
    # Cross-validation for Ridge regression
    # Perform nfolds-cross validation of ridge, for the values of in lambda

  # Number of instances
    n.train <- nrow(x.train)

    # Matrix to store predictions
    p.cv <- matrix(NA, n.train, length(lambda))

    # Prepare the folds
    s <- split(sample(n.train),rep(1:nfolds,length=n.train))

    # Cross-validation
    for (i in seq(nfolds)) {
        ridge.temp <- glmnet(x.train[-s[[i]],],y.train[-s[[i]]]
                             ,lambda=lambda,alpha=0,standardize=F)
      p.cv[s[[i]],] <- predict(ridge.temp,s=lambda,newx=x.train[s[[i]],],exact=T)
    }
    cv.err <- apply(((p.cv>.5)!=y.train),2,mean)
    newlist <- list("cv.error" = cv.err, "lambda.min" = lambda[which.min(cv.err)]
                   ,"lambda"=lambda,"err.min" = min(cv.err),"pred.mtx"=p.cv)
}
# Run a 5-fold CV and obtain CV errors
lambda.grid <- cv.out$lambda
set.seed(1)
new.cv <- ridge.cv(lambda.grid,x.train,y.train,5)
#  Plot the CV error against lambda
dataplot <- data.frame("log.lambda"=log(new.cv$lambda),"cv.err"=new.cv$cv.error)
p <- ggplot(dataplot,aes(x=log.lambda,y=cv.err))+
  geom_point(colour="seagreen")+
  labs(x="log(lambda)",y="CV misclassification errors")
p
```




```
new.cv$lambda.min ; new.cv$err.min
```

```
## [1] 0.2747547
```

```
## [1] 0.01439885
```

```
y.pred.test.ridge3 <- predict(ridge.mod,s=new.cv$lambda.min,newx=x.test,exact=TRUE)
ridge.test.err3 <- mean((y.pred.test.ridge3>.5)!=y.test);ridge.test.err3
```

```
## [1] 0.03021978
```

The λ that has the smallest cross-validated misclassification error is 0.2747547 and the CV error and the true test misclassification error for this λ are 0.01439885 and 0.03021978.

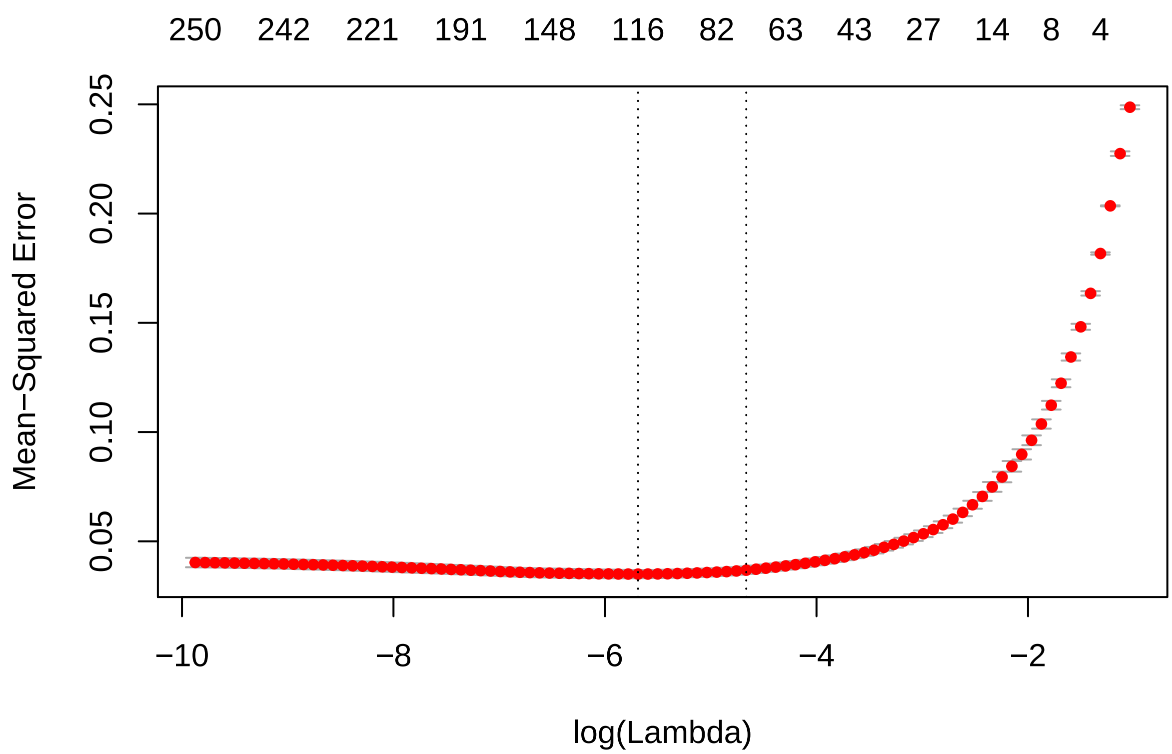
(c)

The reason is that, in part (a), we use `cv.glmnet`, and there is no `predict` funciton in the for loop of `cv.glmnet`. This can save some time. And `cv.glmnet` will give a fitted glmnet object for the full data rather than predicted y's. However, we put `predict` function in the for loop of our function. This way will take longer time.

10. CV for LASSO

(a)

```r
# lambda.grid <- 10^seq(4,-3,length=100)
set.seed(1)
cv.out = cv.glmnet(x.train,y.train,alpha=1,nfolds=5)
plot(cv.out)
```




```r
lamda.min = cv.out$lambda.min; lamda.min
```

```
## [1] 0.003387304
```

```r
lasso.mod <- glmnet(x.train,y.train,alpha=1,standardize=F)
y.pred.test.lasso1 <- predict(lasso.mod,s=lamda.min,newx=x.test,exact=TRUE)
lasso.test.err1 <- mean((y.pred.test.lasso1>.5)!=y.test);lasso.test.err1
```

```
## [1] 0.03846154
```

```r
lambda.se = cv.out$lambda.1se; lambda.se
```

```
## [1] 0.009425374
```

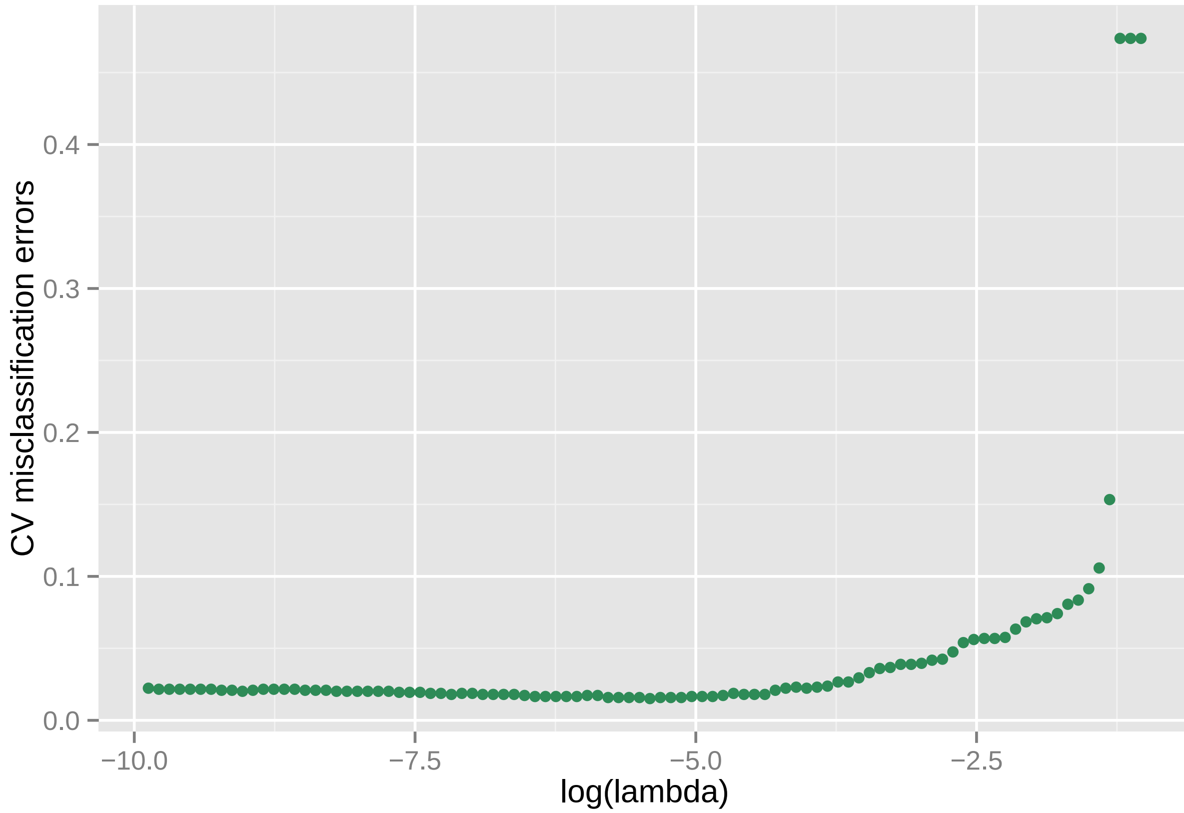
```r
y.pred.test.lasso2 <- predict(lasso.mod,s=lambda.se,newx=x.test,exact=TRUE)
lasso.test.err2 <- mean((y.pred.test.lasso2>.5)!=y.test);lasso.test.err2
```

```
## [1] 0.04395604
```

The λ that has the smallest cross-validated MSE is 0.003387304 and the true test misclassification error for this λ is 0.03846154 The best λ according to "One-Standard Error" Rule is 0.009425374 and the true test misclassification error for this λ is 0.04395604

(b)

```
lasso.cv <- function(lambda,x.train,y.train,nfolds) {
    # Cross-validation for lasso regression
    # Perform nfolds-cross validation of lasso, for the values of in lambda

  # Number of instances
    n.train <- nrow(x.train)

    # Matrix to store predictions
    p.cv <- matrix(NA, n.train, length(lambda))

    # Prepare the folds
    s <- split(sample(n.train),rep(1:nfolds,length=n.train))

    # Cross-validation
    for (i in seq(nfolds)) {
        lasso.temp <- glmnet(x.train[-s[[i]],],y.train[-s[[i]]]
                         ,lambda=lambda,alpha=1,standardize=F)
      p.cv[s[[i]],] <- predict(lasso.temp,s=lambda,newx=x.train[s[[i]],],exact=T)
    }
    cv.err <- apply(((p.cv>.5)!=y.train),2,mean)
    newlist <- list("cv.error" = cv.err, "lambda.min" = lambda[which.min(cv.err)]
                    ,"lambda"=lambda,"err.min" = min(cv.err),"pred.mtx"=p.cv)
}
# Run a 5-fold CV and obtain CV errors
lambda.grid <- cv.out$lambda
set.seed(1)
new.cv <- lasso.cv(lambda.grid,x.train,y.train,5)
#  Plot the CV error against lambda
dataplot <- data.frame("log.lambda"=log(new.cv$lambda),"cv.err"=new.cv$cv.error)
p <- ggplot(dataplot,aes(x=log.lambda,y=cv.err))+
  geom_point(colour="seagreen")+
  labs(x="log(lambda)",y="CV misclassification errors")
p
```




```
new.cv$lambda.min ; new.cv$err.min
```

```
## [1] 0.004477816
```

```
## [1] 0.01511879
```

```
y.pred.test.lasso3 <- predict(lasso.mod,s=new.cv$lambda.min,newx=x.test,exact=TRUE)
lasso.test.err3 <- mean((y.pred.test.lasso3>.5)!=y.test);lasso.test.err3
```

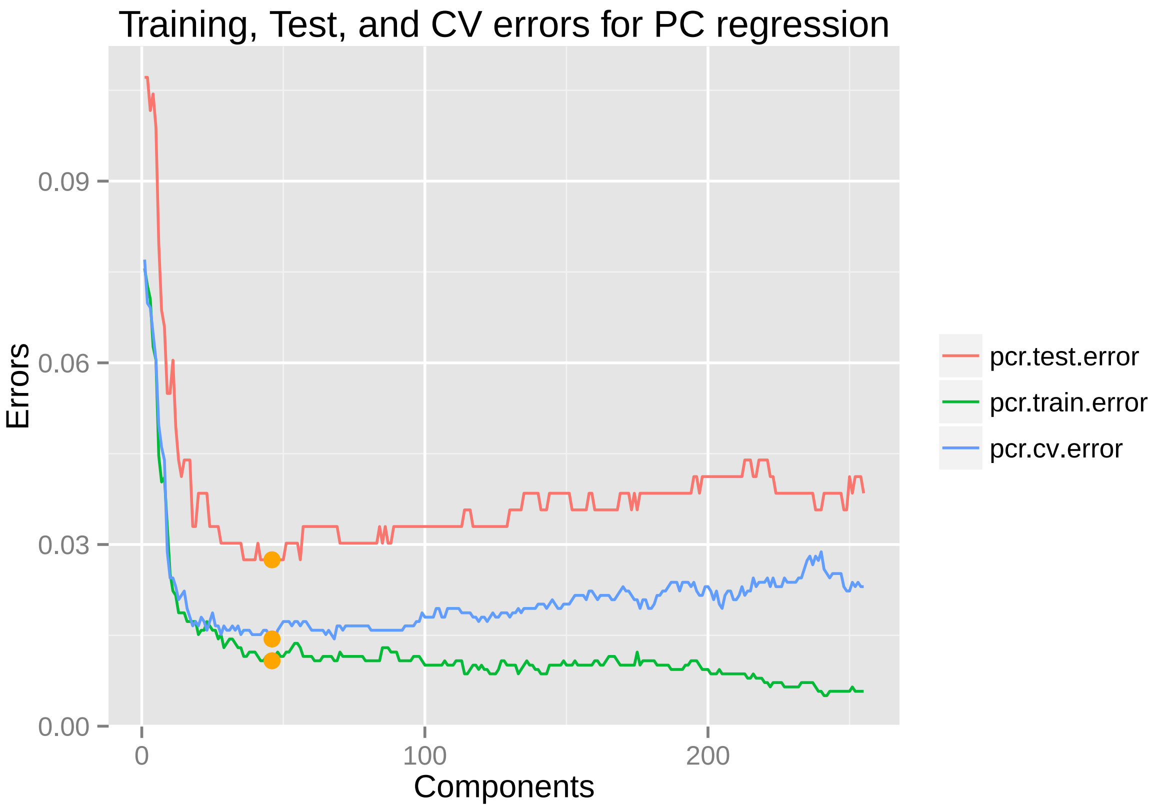
```
## [1] 0.04120879
```

The λ that has the smallest cross-validated misclassification error is 0.004477816 and the CV error and the true test misclassification error for this λ are 0.01511879 and 0.04120879,respectively.

11. CV for PC regression

```
library(pls)

pcr.fit <- pcr(y.train~., data=train.data,validation="none")
# PC regression train errors
pcr.pred.train <- matrix(predict(pcr.fit,x.train,ncomp=1:255)
                        ,nrow=length(y.train))
pcr.train.error <- apply(((pcr.pred.train>.5) != y.train), 2, mean)
```

```r
# PC regression test errors
pcr.pred.test <- matrix(predict(pcr.fit,x.test,ncomp=1:255)
                        ,nrow=length(y.test))
pcr.test.error <- apply(((pcr.pred.test>.5) != y.test), 2, mean)

# CV errors for PC regression
pcr.cv <- function(components,x.train,y.train,nfolds) {
    # Cross-validation for PC regression
    # Perform nfolds-cross validation for different components

  # Number of instances
    n.train <- nrow(x.train)

    # Matrix to store predictions
    p.cv <- matrix(NA, n.train, length(components))

    # Prepare the folds
    s <- split(sample(n.train),rep(1:nfolds,length=n.train))

    # Cross-validation
    for (i in seq(nfolds)) {
        pcr.temp <- pcr(y.train[-s[[i]]]~x.train[-s[[i]],],validation="none")
      p.cv[s[[i]],] <-matrix(predict(pcr.temp,x.train[s[[i]],],ncomp=components),nrow=length(s[[i]]))
    }
    cv.err <- apply(((p.cv>.5)!=y.train),2,mean)
    newlist <- list("cv.error" = cv.err, "component.min" = components[which.min(cv.err)], "component"=c
}
components <- 1:255
set.seed(1)
cv.pcr <- pcr.cv(components,x.train,y.train,5)
pcr.cv.error <- cv.pcr$cv.error
dataplot <- as.data.frame(cbind(components,pcr.test.error,pcr.train.error,pcr.cv.error))
comp.min <- cv.pcr$component.min
datapoint <- data.frame(x=c(comp.min,comp.min,comp.min),y=c(cv.pcr$err.min,pcr.test.error[comp.min],pcr

dataplot <- melt(dataplot,id="components")
#colnames(dataplot) <- c("c","v","b")
g <- ggplot(dataplot,aes(x=components,y=value,colour=variable))+
  geom_line()+
  geom_point(data=datapoint,aes(x,y),size=3,colour="orange")+
  scale_color_hue(name="")+
  labs(x="Components",y="Errors",title="Training, Test, and CV errors for PC regression")
g
```



According to the figure, we should choose 46 components. The training and test errors with that many components selected are 0.01079914 and 0.02747253.

12. CV for Subset Selection

```
library(leaps)
subset.cv <- function(subsets,x.train,y.train,nfolds) {
    # Cross-validation for subset selection
    # Perform nfolds-cross validation for different subsets

  # Number of instances
    n.train <- nrow(x.train)

    # Matrix to store predictions
    p.cv <- matrix(NA, n.train, length(subsets))

    # Prepare the folds
    s <- split(sample(n.train),rep(1:nfolds,length=n.train))

    # Cross-validation
    for (i in seq(nfolds)) {
        regfit.temp <- regsubsets(y.train[-s[[i]]]~x.train[-s[[i]],],data=NULL,nvmax=dim(x.train)[2],rea
        m.temp <- t(summary(regfit.temp)$which)
    m.temp[m.temp==TRUE]<-unlist(coef(regfit.temp,subsets))
        p.cv[s[[i]],] <- x.train[s[[i]],]%*%m.temp
    }
```

```
    cv.err <- apply(((p.cv>.5)!=y.train),2,mean)
    newlist <- list("cv.error" = cv.err, "subset.min" = subsets[which.min(cv.err)], "subsets"=subsets,"
}

regfit.fwd <- regsubsets(y.train~x.train,data=NULL,nvmax=dim(x.train)[2],really.big=T,method="forward",
subsets <- 1:dim(x.train)[2]
m <- t(summary(regfit.fwd)$which)
m[m==TRUE]<-unlist(coef(regfit.fwd,subsets))

# Train error for subset selection
y.pred.train.subset <- x.train %*% m
train.error.subset <- apply((y.pred.train.subset>.5)!=y.train,2,mean)
m0.train.pred <- rep(mean(y.train),length=length(y.train))
m0.train.error <- mean((m0.train.pred>.5)!=y.train)
train.error.subset <- c(m0.train.error,train.error.subset)
# Test error for subset selection
y.pred.test.subset <- x.test %*% m
test.error.subset <- apply((y.pred.test.subset>.5)!=y.test,2,mean)
m0.test.pred <- rep(mean(y.train),length=length(y.test))
m0.test.error <- mean((m0.test.pred>.5)!=y.test)
test.error.subset <- c(m0.test.error,test.error.subset)
# CV error for subset selection
set.seed(1)
cv.subset <- subset.cv(subsets,x.train,y.train,5)
cv.error.subset <- cv.subset$cv.error
cv.error.subset <- c(m0.train.error,cv.error.subset)
plotdata <- data.frame(subsets=c(0,subsets),train.error.subset,test.error.subset,cv.error.subset)

subset.best <- cv.subset$subset.min; subset.best
```

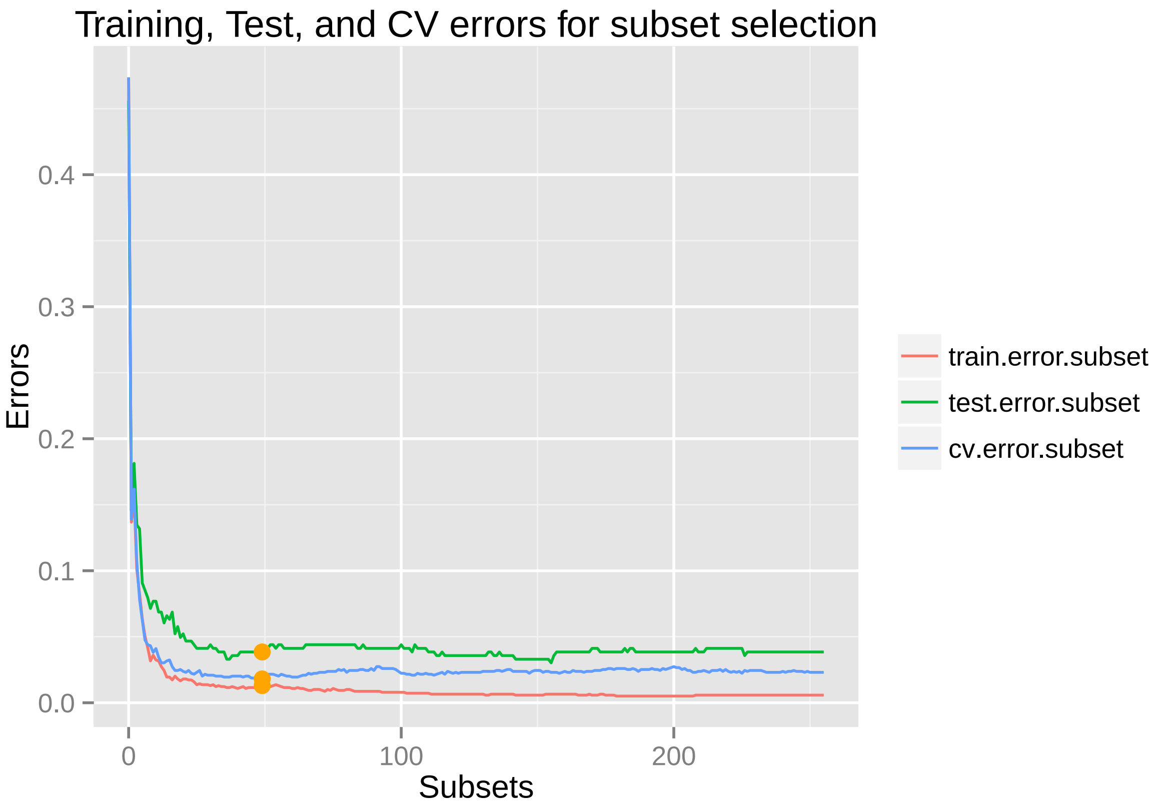
```
## [1] 49
```

```
train.error.subset[subset.best+1];test.error.subset[subset.best+1]
```

```
##         49
## 0.01151908
```

```
##         49
## 0.04120879
```

```
pointdata <- data.frame(x=rep(subset.best,3),y=c(cv.subset$err.min,train.error.subset[subset.best],test

plotdata <- melt(plotdata,id="subsets")
p <- ggplot(plotdata,aes(x=subsets,y=value,colour=variable))+
  geom_line()+
  geom_point(data=pointdata,aes(x,y),size=3,colour="orange")+
  scale_color_hue(name="")+
  labs(x="Subsets",y="Errors",title="Training, Test, and CV errors for subset selection")
p
```



We should choose 49 predictors. The training and test errors with that many predictors selected are 0.01151908 and 0.04120879, respectively.