Statistical Learning HW2

Yaniv Bronshtein

10/31/2021

Import necessary libraries

```
library(plotrix)
library(ISLR)
library(e1071)
library(dslabs)
library(glmnet)
```

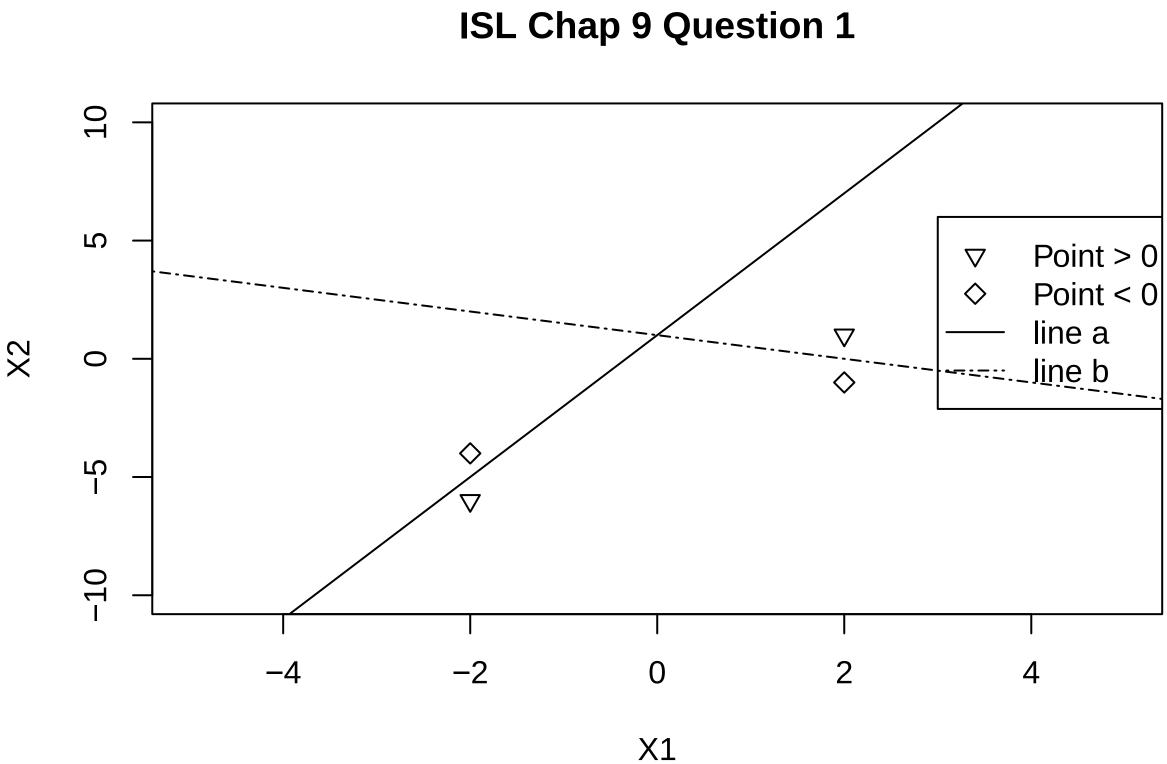
```
## Loading required package: Matrix
```

```
## Loaded glmnet 4.1-2
```

```
library(Matrix)
```

Question 1

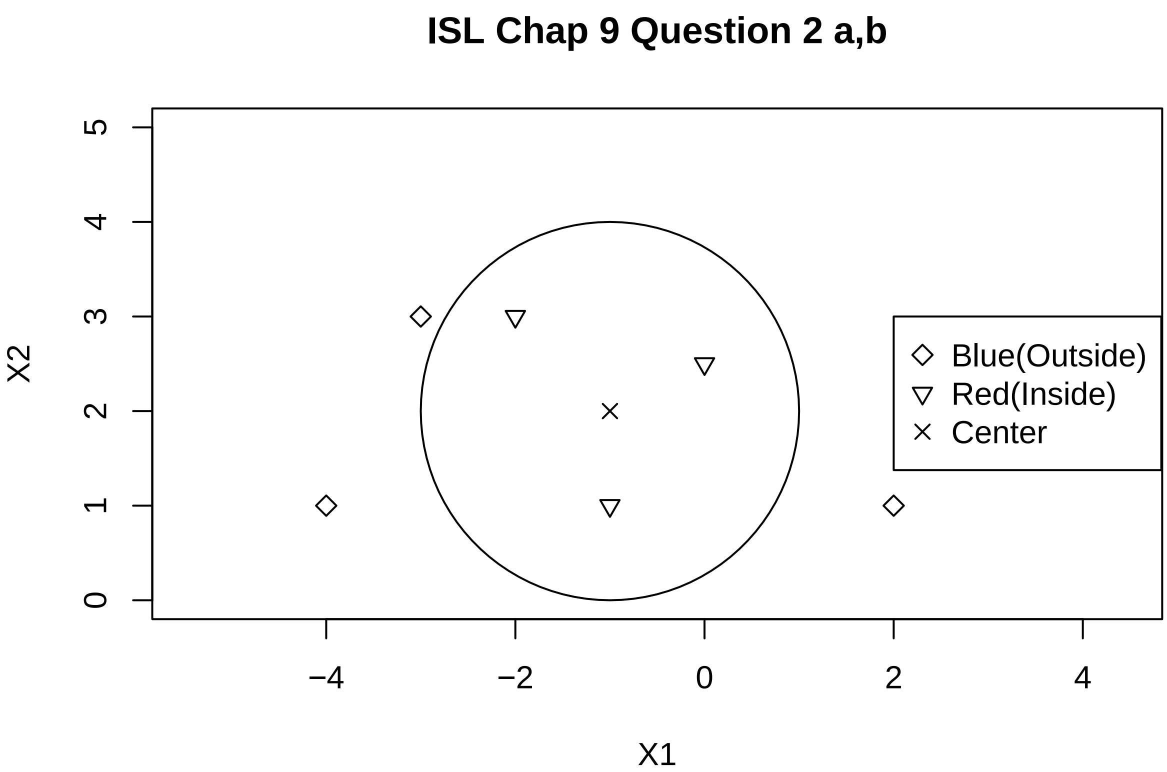
```
plot(0,type="n",xlab='X1', ylab='X2',
     ylim = c(-10,10),xlim = c(-5,5),main="ISL Chap 9 Question 1")
abline(1,3,lty=1) #Line (a): 1+3X1-X2=0 (solid)
abline(1,-0.5, lty=6) #Line (b): -2+X1+2X2=0 (dashed)
# Points where Line(a)>0 and Line(a)<0
points(-2,-4,pch=5) #Points where line a > 0. diamond
points(-2,-6,pch=6) #Points where line a < 0. triangle.
# Points where Line(b)>0 and Line(b)<0
points(2,1,pch=6) #Points where line b > 0. triangle
points(2,-1,pch=5) #Points where line b < 0. diamond
legend(3,6,legend=c("Point > 0", "Point < 0", "line a", "line b"),
       pch=c(6,5,NA,NA),lty=c(NA,NA,1,6))
```



Question 2

Here we have a circle that follows the equation:: (x − h)^2+ (y − k)2 = r^2 where the center is (h,k) In our case (h,k)=(-1,2) and r=2 Below is the solution for (a) and (b)

```
plot(x=-3:2,y=0:5,type="n",asp=1, xlab='X1', ylab='X2', main="ISL Chap 9 Question 2 a,b")
draw.circle(x=-1,y=2,radius=2)
points(-1,2, pch=4)
# Points outside decision boundary
points(c(-4,2,-3),c(1,1,3),pch=5)
#Points inside decision boundary
points(c(-1,-2,0),c(1,3,2.5),pch=6)
legend(x=2, y=3,   # Coordinates (x also accepts keywords)
       c('Blue(Outside)','Red(Inside)','Center'), # Vector with the name of each group
       pch=c(5,6,4)
)
```

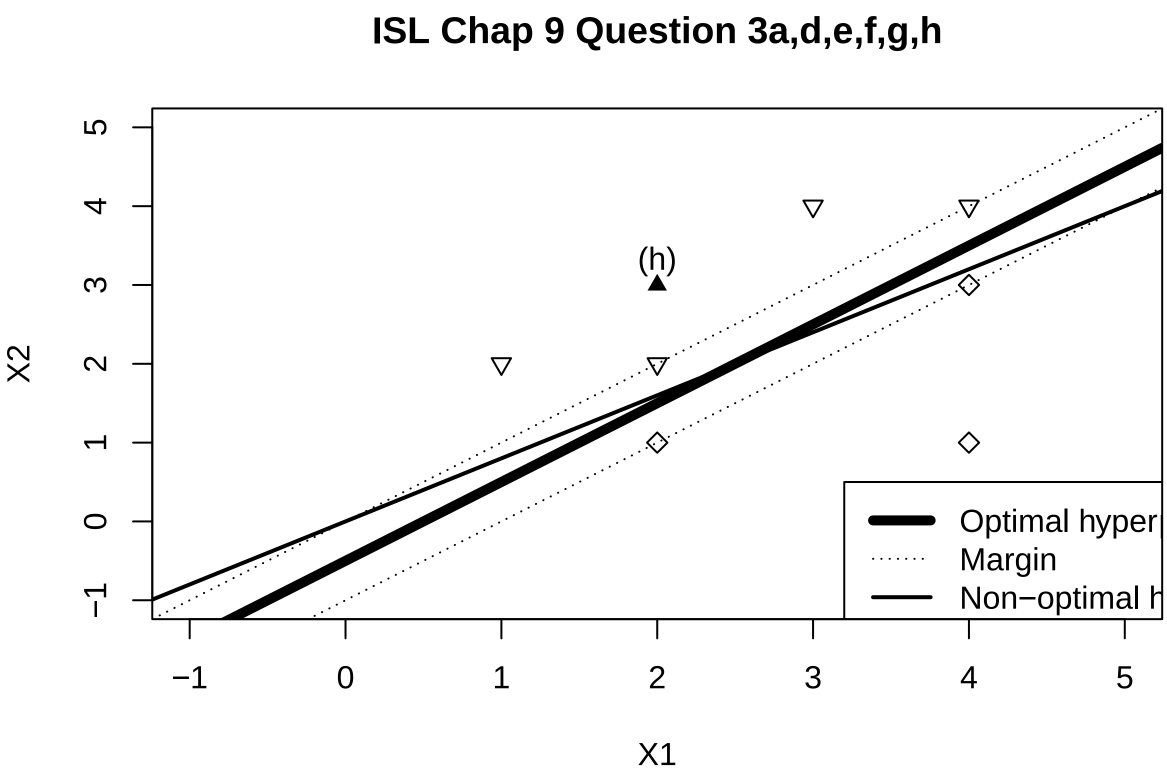


(c).

(0,0) is classified as belonging to the blue class. (-1,1) is classified as belonging to the red class. (2,2) is classified as belonging to the blue class. (3,8) is classified as belonging to the blue class.

Question 3

```
plot(-1:5,-1:5,type="n",xlab='X1', ylab='X2', main="ISL Chap 9 Question 3a,d,e,f,g,h")
points(c(3,2,4,1),c(4,2,4,2), pch=6)
points(c(2,4,4),c(1,3,1),pch=5)
points(2,3,pch=17)
abline(-0.5,1,lwd=5) #y intercept=-0.5 and gradient=1.
abline(-1, 1, lty='dotted')
abline(0, 1, lty='dotted')
abline(0,0.8, lwd=2)
text(2,3.3,'(h)')
legend(3.2,0.5,
       legend=c("Optimal hyperplane", "Margin", "Non-optimal hyperplane"),
       lty=c(1,3,1),
       lwd=c(5,1,2))
```



Question 4

(ISL Chap 9 Question 8) (a) Create a training set containing a random sample of 800 observations, and a test set containing the remaining observations.

```
data(OJ)
set.seed(1)
train <- sample(nrow(OJ),size = 800)
test <- -train
oj_train <- OJ[train, ]
oj_test <- OJ[test, ]
```

(b) Fit a support vector classifier to the training data using cost=0.01, with Purchase as the response and the other variables as predictors. Use the summary() function to produce summary statistics, and describe the results obtained.

```
svm_fit <- svm(Purchase~ .,kernel="linear", data =oj_train, cost=0.01)
```

Let's get the summary

```
summ_svm_fit <- summary(svm_fit)
summ_svm_fit
```

```
##
## Call:
## svm(formula = Purchase ~ ., data = oj_train, kernel = "linear", cost = 0.01)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
```

```
##        cost:  0.01
##
## Number of Support Vectors:  435
##
##  ( 219 216 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

The linear support vector classifier creates a classification out of 435 support vectors from 800 observations with 219 classified as CH and 216 classified as MM

(c) What are the training and test error rates?

```r
#Get the predictions
train_pred <- predict(svm_fit, oj_train)
test_pred <- predict(svm_fit, oj_test)

#Create the confusion matrices
table1 <- table(oj_train$Purchase, train_pred)
table2 <- table(oj_test$Purchase, test_pred)
table1
```

```
##     train_pred
##       CH  MM
##   CH 420  65
##   MM  75 240
```

```r
cat('*****************','\n')
```

```
## *****************
```

```r
table2
```

```
##     test_pred
##       CH  MM
##   CH 153  15
##   MM  33  69
```

```r
get_err_rate <- function(my_table){
  return((my_table[2,1] + my_table[1,2])/sum(my_table))
}

train_err = get_err_rate(table1)
test_err = get_err_rate(table2)
cat('*****************','\n')
```

```
## *****************
```

```r
cat("Train Error:", train_err,'\n')
```

```
## Train Error: 0.175
```

```r
cat("Test Error:", test_err,'\n')
```

```
## Test Error: 0.1777778
```

(d) Use the tune() function to select an optimal cost. Consider values in the range 0.01 to 10.

```
set.seed(2)
tune_out <- tune(svm,Purchase ~ ., data=oj_train, kernel="linear", ranges=list(cost=10^seq(-2,1,by=0.25)
summ_tune <- summary(tune_out)
```

Let us see the best parameter cost and best performance

```
cat("Best parameter cost:\n")
```

```
## Best parameter cost:
```

```
best_cost <- summ_tune$best.parameters$cost
best_cost
```

```
## [1] 1.778279
```

```
cat("Best performance:\n")
```

```
## Best performance:
```

```
best_performance <- summ_tune$best.performance
best_performance
```

```
## [1] 0.1675
```

(e) Compute the training and test error rates using this new value for cost.

```
svm_fit_best <- svm(Purchase~ .,kernel="linear", data =oj_train, cost=best_cost)
#Get the predictions
train_pred_best <- predict(svm_fit, oj_train)
test_pred_best <- predict(svm_fit, oj_test)

#Create the confusion matrices
table1_best <- table(oj_train$Purchase, train_pred_best)
table2_best <- table(oj_test$Purchase, test_pred_best)
table1_best
```

```
##     train_pred_best
##       CH  MM
##   CH 420  65
##   MM  75 240
```

```
cat('*****************','\n')
```

```
## *****************
```

```
table2_best
```

```
##     test_pred_best
##       CH  MM
##   CH 153  15
##   MM  33  69
```

```
train_err_best = get_err_rate(table1_best)
test_err_best = get_err_rate(table2_best)
cat('*****************','\n')
```

```
## *****************
```

```
cat("Train Error Best:", train_err_best,'\n')
```

```
## Train Error Best: 0.175
```

```r
cat("Test Error Best:", test_err_best,'\n')
```

```
## Test Error Best: 0.1777778
```

(f) Repeat parts (b) through (e) using a support vector machine with a radial kernel. Use the default value for gamma.

```r
svm_fit_radial <- svm(Purchase~ .,kernel="radial", data =oj_train)
svm_radial_summ <- summary(svm_fit_radial)
svm_radial_summ
```

```
##
## Call:
## svm(formula = Purchase ~ ., data = oj_train, kernel = "radial")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  1
##
## Number of Support Vectors:  373
##
##  ( 188 185 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

```r
#Get the predictions
train_pred_radial <- predict(svm_fit_radial, oj_train)
test_pred_radial <- predict(svm_fit_radial, oj_test)

#Create the confusion matrices
table1_radial <- table(oj_train$Purchase, train_pred_radial)
table2_radial <- table(oj_test$Purchase, test_pred_radial)
table1_radial
```

```
##     train_pred_radial
##       CH  MM
##   CH 441  44
##   MM  77 238
```

```r
cat('*****************','\n')
```

```
## *****************
```

```r
table2_radial
```

```
##     test_pred_radial
##       CH  MM
##   CH 151  17
##   MM  33  69
```

```r
train_err_radial = get_err_rate(table1_radial)
test_err_radial = get_err_rate(table2_radial)
```

```r
cat('*****************','\n')
```

```
## *****************
```

```r
cat("Train Error Radial SVM:", train_err_radial,'\n')
```

```
## Train Error Radial SVM: 0.15125
```

```r
cat("Test Error Radial SVM:", test_err_radial,'\n')
```

```
## Test Error Radial SVM: 0.1851852
```

(g) Repeat parts (b) through (e) using a support vector machine with a polynomial kernel. Set degree=2.

```r
svm_fit_poly <- svm(Purchase~ .,kernel="polynomial",degree=2, data=oj_train)
svm_poly_summ <- summary(svm_fit_poly)
svm_poly_summ
```

```
##
## Call:
## svm(formula = Purchase ~ ., data = oj_train, kernel = "polynomial",
##     degree = 2)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  polynomial
##        cost:  1
##      degree:  2
##      coef.0:  0
##
## Number of Support Vectors:  447
##
##  ( 225 222 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

```r
#Get the predictions
train_pred_poly <- predict(svm_fit_poly, oj_train)
test_pred_poly <- predict(svm_fit_poly, oj_test)

#Create the confusion matrices
table1_poly <- table(oj_train$Purchase, train_pred_poly)
table2_poly <- table(oj_test$Purchase, test_pred_poly)
table1_poly
```

```
##     train_pred_poly
##       CH  MM
##   CH 449  36
##   MM 110 205
```

```r
cat('*****************','\n')
```

```
## *****************
```

```
table2_poly
```

```
##     test_pred_poly
##        CH  MM
##   CH 153  15
##   MM  45  57
```

```
train_err_poly = get_err_rate(table1_poly)
test_err_poly = get_err_rate(table2_poly)
cat('*****************','\n')
```

```
## *****************
```

```
cat("Train Error Polynomial, Degree 2 SVM:", train_err_poly,'\n')
```

```
## Train Error Polynomial, Degree 2 SVM: 0.1825
```

```
cat("Test Error Polynomial, Degree 2 SVM:", test_err_poly,'\n')
```

```
## Test Error Polynomial, Degree 2 SVM: 0.2222222
```

(h) Overall, which approach seems to give the best results on this data? *It seems like radial kernel gives the best result*

Question 5

```
mnist <- read_mnist()
```

Now create the training and test set for this problem as follows, each of size 800

```
# Select the first 400 images of "3" in mnist$test$images, and the first 400 images
# of "5" in mnist$test$images, as the training set.
# Create the corresponding label vector, which has length 800.

all_labels <- mnist$test$labels
all_images <- mnist$train$images

#select images for testing and training
train_images_3 <- mnist$test$images[all_labels ==3,][1:400,]
train_images_5 <- mnist$test$images[all_labels ==5,][1:400,]
test_images_3 <- mnist$test$images[all_labels ==3,][401:800,]
test_images_5 <- mnist$test$images[all_labels ==5,][401:800,]

#The labels_vec is used for both train_df and test_df
labels_vec <- rep(c('3','5'),each=400)
train_images <- rbind(train_images_3,train_images_5)
test_images <- rbind(test_images_3,test_images_5)

#Create the dataframes for train and test data
df_train <- data.frame(
  labels=labels_vec,
  images=train_images
  )
df_test <- data.frame(
  labels=labels_vec,
  images=test_images
```

```
)
```

(a) Perform logistic regression on the training set, and use it to predict the labels of the test set. Report the training and testing mis-classification rates.

```
lr_fit <- glm(formula=as.factor(labels) ~ .,family=binomial(link=logit),data=df_train)
```

```
## Warning: glm.fit: algorithm did not converge

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

Repeat function to get error rate

```
get_err_rate <- function(my_table){
  return((my_table[2,1] + my_table[1,2])/sum(my_table))
}
```

Let's predict

```
#Get the logistic regression probabilities
lr_prob_train <- predict(lr_fit, df_train, type='response')
```

```
## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
## prediction from a rank-deficient fit may be misleading
```

```
lr_prob_test <- predict(lr_fit, df_test, type='response')
```

```
## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
## prediction from a rank-deficient fit may be misleading
```

```
#Get the classification by checking the value relative to the threshold
lr_pred_train<- rep('3', 800)
lr_pred_test<- rep('3', 800)
lr_pred_train[lr_prob_train > .5] <- '5'
lr_pred_test[lr_prob_test > .5] <- '5'

train_table_lr <- table(df_train$labels, lr_pred_train)
test_table_lr <- table(df_test$labels, lr_pred_test)
train_err_lr = get_err_rate(train_table_lr)
test_err_lr = get_err_rate(test_table_lr)
cat("train_err_lr",train_err_lr)
```

```
## train_err_lr 0
```

```
cat("\ntest_err_lr",test_err_lr)
```

```
##
## test_err_lr 0.2375
```

(b) For the logistic regression, the size of the training set is N = 800, and the number of features is p = 784, which is almost the same as N. Now try to run the logistic regression using the glmnet() function in the glmnet package. This function adds a Lasso type penalty to the logistic regression. Use the tuning parameter lambda=.1 and family="binomial" in the glmnet() function (you don't need to specify any other parameters). Report the training and testing mis-classification rates.

```
glmnet_lr <- glmnet(x=df_train[,-1], y=df_train[,1], family='binomial', lambda=.1)
```

Let's get predictions and misclassification rates for glmnet()

```
#Get the logistic regression probabilities
glmnet_lr_prob_train <- predict(glmnet_lr, as.matrix(df_train[,-1]), type='response')
```

```
glmnet_lr_prob_test <- predict(glmnet_lr, as.matrix(df_test[,-1]), type='response')
#Get the classification by checking the value relative to the threshold
glmnet_lr_pred_train<- rep('3', 800)
glmnet_lr_pred_test<- rep('3', 800)
glmnet_lr_pred_train[glmnet_lr_prob_train > .5] <- '5'
glmnet_lr_pred_test[glmnet_lr_prob_test > .5] <- '5'

train_table_glmnet_lr <- table(df_train$labels, glmnet_lr_pred_train)
test_table_glmnet_lr <- table(df_test$labels, glmnet_lr_pred_test)
train_err_glmnet_lr = get_err_rate(train_table_glmnet_lr)
test_err_glmnet_lr = get_err_rate(test_table_glmnet_lr)

cat("train_err_glmnet_lr",train_err_glmnet_lr)
```

```
## train_err_glmnet_lr 0.11125
```

```
cat("\ntest_err_glmnet_lr",test_err_glmnet_lr)
```

```
##
## test_err_glmnet_lr 0.145
```

(c) Try some other values of lambda, and report the smallest testing mis-classification rate you obtain, with the corresponding value of lambda.

```
test_ms_rates <- NULL
my_range <- 10^seq(-4,-1,by=0.2)
for(i in my_range){
  model <- glmnet(x=df_train[,-1], y=df_train[,1], family='binomial', lambda=i)
    model_prob_test <- predict(model, as.matrix(df_test[,-1]), type='response')
  model_pred_test<- rep('3', 800)

    model_pred_test[model_prob_test > .5] <- '5'
  test_table_model <- table(df_test$labels, model_pred_test)
    test_err_model = get_err_rate(test_table_model)
    test_ms_rates <- c(test_ms_rates,test_err_model)

    }
```

**Get the minimum test misclassification rate

```
idx <- which.min(test_ms_rates)

cat("Minimum test mis-classification rate:",min(test_ms_rates), "with lambda:", my_range[idx])
```

```
## Minimum test mis-classification rate: 0.065 with lambda: 0.003981072
```

(d) Build a support vector classifier using the training set, and use it to predict the labels of the test set. Report the training and testing mis-classification rates. [Hint. You can use cost=1, and add scale=FALSE in the svm() function.]

Build an svm model

```
set.seed(1)
svm_model <- svm(as.factor(labels)~ .,kernel="linear",
                data =df_train, cost=1, scale=FALSE)
```

Get the training and testing mis-classification rates

```r
train_pred_svm_model <- predict(svm_model, df_train)
test_pred_svm_model <- predict(svm_model, df_test)

svm_model_table1 <- table(df_train$labels, train_pred_svm_model)
svm_model_table2 <- table(df_test$labels, test_pred_svm_model)
svm_model_train_err <- get_err_rate(svm_model_table1)
svm_model_test_err <- get_err_rate(svm_model_table2)
cat("Train Misclassification rate:",svm_model_train_err,
    "Test Misclassification rate:",svm_model_test_err )
```

```
## Train Misclassification rate: 0 Test Misclassification rate: 0.085
```

(e) From now on only use the 400 images of "3" in the training set. Plot the average image of them.

```r
avg_image <- apply(df_train[1:400,-1], 2, mean)
avg_image
```

```
##   images.1   images.2   images.3   images.4   images.5   images.6   images.7
##     0.0000     0.0000     0.0000     0.0000     0.0000     0.0000     0.0000
##   images.8   images.9  images.10  images.11  images.12  images.13  images.14
##     0.0000     0.0000     0.0000     0.0000     0.0000     0.0000     0.0000
##  images.15  images.16  images.17  images.18  images.19  images.20  images.21
##     0.0000     0.0000     0.0000     0.0000     0.0000     0.0000     0.0000
##  images.22  images.23  images.24  images.25  images.26  images.27  images.28
##     0.0000     0.0000     0.0000     0.0000     0.0000     0.0000     0.0000
##  images.29  images.30  images.31  images.32  images.33  images.34  images.35
##     0.0000     0.0000     0.0000     0.0000     0.0000     0.0000     0.0000
##  images.36  images.37  images.38  images.39  images.40  images.41  images.42
##     0.0000     0.0000     0.0000     0.0000     0.0000     0.0000     0.0000
##  images.43  images.44  images.45  images.46  images.47  images.48  images.49
##     0.0000     0.0000     0.0000     0.0000     0.0000     0.0000     0.0000
##  images.50  images.51  images.52  images.53  images.54  images.55  images.56
##     0.0000     0.0000     0.0000     0.0000     0.0000     0.0000     0.0000
##  images.57  images.58  images.59  images.60  images.61  images.62  images.63
##     0.0000     0.0000     0.0000     0.0000     0.0000     0.0000     0.0000
##  images.64  images.65  images.66  images.67  images.68  images.69  images.70
##     0.0000     0.0000     0.0000     0.0000     0.0000     0.0000     0.0000
##  images.71  images.72  images.73  images.74  images.75  images.76  images.77
##     0.0000     0.0000     0.0000     0.0000     0.0000     0.0000     0.0000
##  images.78  images.79  images.80  images.81  images.82  images.83  images.84
##     0.0000     0.0000     0.0000     0.0000     0.0000     0.0000     0.0000
##  images.85  images.86  images.87  images.88  images.89  images.90  images.91
##     0.0000     0.0000     0.0000     0.0000     0.0000     0.5425     0.6375
##  images.92  images.93  images.94  images.95  images.96  images.97  images.98
##     0.5675     1.5325     2.7650     3.4175     4.5075     6.1350     4.8275
##  images.99 images.100 images.101 images.102 images.103 images.104 images.105
##     2.4600     1.1550     0.2725     0.0000     0.0000     0.0000     0.0000
## images.106 images.107 images.108 images.109 images.110 images.111 images.112
##     0.0000     0.0000     0.0000     0.0000     0.0000     0.0000     0.0000
## images.113 images.114 images.115 images.116 images.117 images.118 images.119
##     0.0000     0.0000     0.0000     0.0225     0.9675     1.5875     4.1425
## images.120 images.121 images.122 images.123 images.124 images.125 images.126
##     9.9700    20.6700    33.2075    47.9950    62.6600    72.2250    72.1250
## images.127 images.128 images.129 images.130 images.131 images.132 images.133
##    63.1000    49.8350    33.8800    17.6425     7.4800     2.7050     0.4650
```

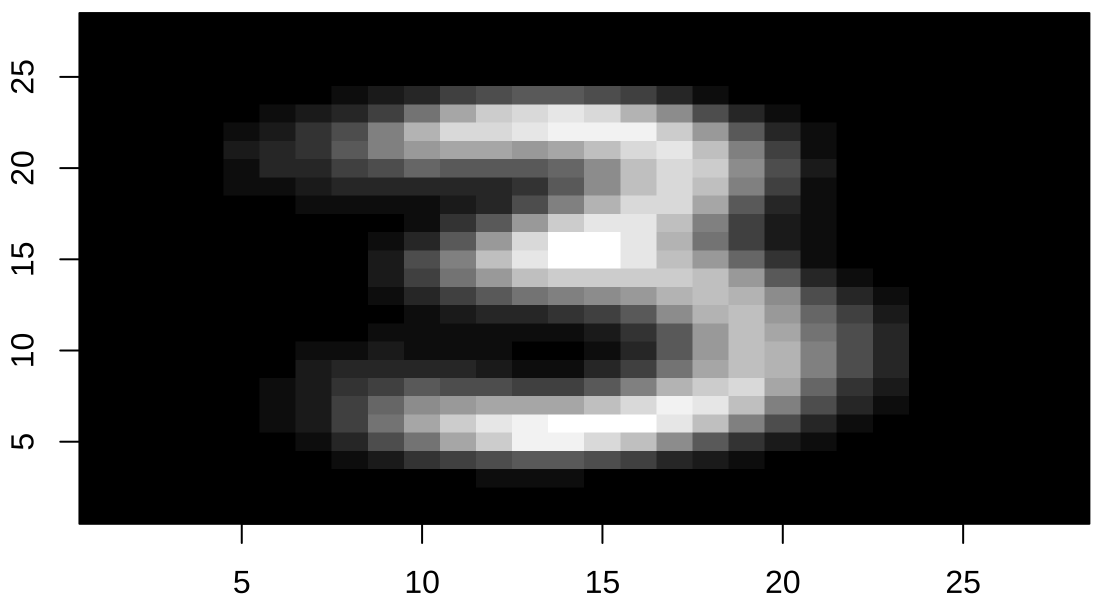
```
## images.134 images.135 images.136 images.137 images.138 images.139 images.140
##     0.1925     0.0000     0.0000     0.0000     0.0000     0.0000     0.0000
## images.141 images.142 images.143 images.144 images.145 images.146 images.147
##     0.0000     0.0000     0.0000     0.2125     4.7050    11.0075    22.1850
## images.148 images.149 images.150 images.151 images.152 images.153 images.154
##    36.1000    54.5000    90.1450   126.2550   148.2850   160.8775   164.8700
## images.155 images.156 images.157 images.158 images.159 images.160 images.161
##   156.0825   132.9550   101.0650    63.8175    32.9125    12.6100     4.6400
## images.162 images.163 images.164 images.165 images.166 images.167 images.168
##     1.2300     0.1400     0.0000     0.0000     0.0000     0.0000     0.0000
## images.169 images.170 images.171 images.172 images.173 images.174 images.175
##     0.0000     0.0000     0.0000     3.4650    11.9850    25.0325    39.2225
## images.176 images.177 images.178 images.179 images.180 images.181 images.182
##    61.4475    96.5375   131.7450   155.0200   163.9150   170.3000   174.6550
## images.183 images.184 images.185 images.186 images.187 images.188 images.189
##   179.3900   177.7900   154.3850   115.9750    70.9175    30.4000    11.3650
## images.190 images.191 images.192 images.193 images.194 images.195 images.196
##     4.0275     1.1650     0.1875     0.0000     0.0000     0.0000     0.0000
## images.197 images.198 images.199 images.200 images.201 images.202 images.203
##     0.0000     0.0000     0.0525     5.4025    18.2500    30.6175    43.4100
## images.204 images.205 images.206 images.207 images.208 images.209 images.210
##    67.0825    93.8200   113.3525   120.1050   118.5325   117.2800   123.5175
## images.211 images.212 images.213 images.214 images.215 images.216 images.217
##   141.3925   160.6400   165.0825   145.2100    97.5300    49.2600    16.5775
## images.218 images.219 images.220 images.221 images.222 images.223 images.224
##     6.1300     1.8325     0.4300     0.0000     0.0000     0.0000     0.0000
## images.225 images.226 images.227 images.228 images.229 images.230 images.231
##     0.0000     0.0000     0.1875     5.2400    16.7375    27.4450    35.5850
## images.232 images.233 images.234 images.235 images.236 images.237 images.238
##    49.0175    63.6800    73.5100    72.6250    67.2650    65.8850    77.4975
## images.239 images.240 images.241 images.242 images.243 images.244 images.245
##   106.6475   138.8875   159.1850   149.0200   104.1050    55.4375    20.6275
## images.246 images.247 images.248 images.249 images.250 images.251 images.252
##     6.9525     2.5050     0.2225     0.0000     0.0000     0.0000     0.0000
## images.253 images.254 images.255 images.256 images.257 images.258 images.259
##     0.0000     0.0000     0.4950     4.7300     9.7000    16.2125    23.6775
## images.260 images.261 images.262 images.263 images.264 images.265 images.266
##    28.1150    35.4750    36.1725    35.5250    32.4500    41.6250    68.2700
## images.267 images.268 images.269 images.270 images.271 images.272 images.273
##   103.9100   143.0800   161.9725   141.7125    93.5775    46.7675    17.7925
## images.274 images.275 images.276 images.277 images.278 images.279 images.280
##     7.1325     3.1300     0.5325     0.0000     0.0000     0.0000     0.0000
## images.281 images.282 images.283 images.284 images.285 images.286 images.287
##     0.0000     0.0000     0.1050     3.2150     5.5100     7.5550    12.8175
## images.288 images.289 images.290 images.291 images.292 images.293 images.294
##    13.0500    14.6875    16.9725    18.6675    31.8125    59.3225    99.5650
## images.295 images.296 images.297 images.298 images.299 images.300 images.301
##   136.3275   158.6225   156.5175   119.4050    72.1875    31.2100    12.8400
## images.302 images.303 images.304 images.305 images.306 images.307 images.308
##     6.6750     2.2675     0.7300     0.0000     0.0000     0.0000     0.0000
## images.309 images.310 images.311 images.312 images.313 images.314 images.315
##     0.0000     0.0000     0.0000     1.4750     2.6275     4.7700     5.3550
## images.316 images.317 images.318 images.319 images.320 images.321 images.322
##     5.2875     9.0125    17.1450    37.7550    68.2675   111.3450   148.2575
```

```
## images.323 images.324 images.325 images.326 images.327 images.328 images.329
##   172.2275   172.3975   142.9050    95.8825    51.6375    26.5425    13.2000
## images.330 images.331 images.332 images.333 images.334 images.335 images.336
##     6.3175     1.8400     0.2950     0.0000     0.0000     0.0000     0.0000
## images.337 images.338 images.339 images.340 images.341 images.342 images.343
##     0.0000     0.0000     0.0000     0.5525     1.6875     2.9425     3.9125
## images.344 images.345 images.346 images.347 images.348 images.349 images.350
##     5.2225    13.5725    33.8225    69.0125   117.0825   157.8725   186.5700
## images.351 images.352 images.353 images.354 images.355 images.356 images.357
##   189.7925   170.8550   134.7525    88.4925    49.6700    26.0875    12.7600
## images.358 images.359 images.360 images.361 images.362 images.363 images.364
##     5.9300     1.6375     0.1900     0.0000     0.0000     0.0000     0.0000
## images.365 images.366 images.367 images.368 images.369 images.370 images.371
##     0.0000     0.0000     0.0000     0.0000     0.6650     1.3875     2.2175
## images.372 images.373 images.374 images.375 images.376 images.377 images.378
##     7.2125    21.6575    54.7625    94.2075   139.9050   172.7175   187.2625
## images.379 images.380 images.381 images.382 images.383 images.384 images.385
##   185.9625   169.3900   145.3950   111.7800    75.0600    37.6900    17.1825
## images.386 images.387 images.388 images.389 images.390 images.391 images.392
##     8.1250     3.4400     1.2225     0.0275     0.0000     0.0000     0.0000
## images.393 images.394 images.395 images.396 images.397 images.398 images.399
##     0.0000     0.0000     0.0000     0.0000     0.1725     0.6825     2.1375
## images.400 images.401 images.402 images.403 images.404 images.405 images.406
##     8.2150    23.0975    54.5725    87.1325   115.8375   139.3275   153.0225
## images.407 images.408 images.409 images.410 images.411 images.412 images.413
##   153.3725   154.3575   152.9875   137.4575   110.1900    69.8975    34.9950
## images.414 images.415 images.416 images.417 images.418 images.419 images.420
##    17.1250     6.4650     2.1700     0.3250     0.0000     0.0000     0.0000
## images.421 images.422 images.423 images.424 images.425 images.426 images.427
##     0.0000     0.0000     0.0000     0.0000     0.0500     0.5550     2.3400
## images.428 images.429 images.430 images.431 images.432 images.433 images.434
##     6.5250    15.9325    33.9675    52.1150    69.1975    83.6650    93.4300
## images.435 images.436 images.437 images.438 images.439 images.440 images.441
##   103.4475   117.6425   133.6825   142.8000   131.9675   100.7500    60.0875
## images.442 images.443 images.444 images.445 images.446 images.447 images.448
##    32.3450    13.3025     4.3775     0.6350     0.0000     0.0000     0.0000
## images.449 images.450 images.451 images.452 images.453 images.454 images.455
##     0.0000     0.0000     0.0000     0.0325     0.2400     1.6000     2.0050
## images.456 images.457 images.458 images.459 images.460 images.461 images.462
##     4.1425     7.6000    15.6300    25.6025    30.3350    34.4400    40.3725
## images.463 images.464 images.465 images.466 images.467 images.468 images.469
##    50.6900    71.8150   103.5125   133.3625   139.0275   118.4400    80.4450
## images.470 images.471 images.472 images.473 images.474 images.475 images.476
##    46.8075    21.9975     6.4800     0.4075     0.0000     0.0000     0.0000
## images.477 images.478 images.479 images.480 images.481 images.482 images.483
##     0.0000     0.0000     0.0000     0.5225     1.9475     3.0325     2.8625
## images.484 images.485 images.486 images.487 images.488 images.489 images.490
##     5.9325    10.8050    12.7875    13.9125    12.0900    12.4250    13.2600
## images.491 images.492 images.493 images.494 images.495 images.496 images.497
##    18.5950    39.2425    72.2550   116.8725   140.9525   126.3625    88.4700
## images.498 images.499 images.500 images.501 images.502 images.503 images.504
##    57.1525    29.0550     8.3200     0.2825     0.0000     0.0000     0.0000
## images.505 images.506 images.507 images.508 images.509 images.510 images.511
##     0.0000     0.0000     0.0000     0.8800     2.8775     5.9600    11.8375
```

```
## images.512 images.513 images.514 images.515 images.516 images.517 images.518
##    17.5025    20.1875    17.4425    14.6550    11.5400     6.1900     5.6675
## images.519 images.520 images.521 images.522 images.523 images.524 images.525
##    13.8150    31.0475    67.4675   112.2175   140.6850   130.9925    94.6175
## images.526 images.527 images.528 images.529 images.530 images.531 images.532
##    59.7825    31.4375     8.5450     0.4775     0.0000     0.0000     0.0000
## images.533 images.534 images.535 images.536 images.537 images.538 images.539
##     0.0000     0.0000     0.0000     1.7125     2.8250     8.2700    19.8075
## images.540 images.541 images.542 images.543 images.544 images.545 images.546
##    29.6850    35.3550    35.7975    28.4975    20.0075    15.0125    17.6050
## images.547 images.548 images.549 images.550 images.551 images.552 images.553
##    28.8925    48.7300    83.7775   125.8700   145.6075   131.9250    95.2700
## images.554 images.555 images.556 images.557 images.558 images.559 images.560
##    55.0575    27.3875     6.8250     0.3625     0.0000     0.0000     0.0000
## images.561 images.562 images.563 images.564 images.565 images.566 images.567
##     0.0000     0.0000     0.3650     1.6200     3.3800     9.8350    23.5275
## images.568 images.569 images.570 images.571 images.572 images.573 images.574
##    41.5875    53.3125    64.1600    63.6725    57.7025    49.5100    53.9375
## images.575 images.576 images.577 images.578 images.579 images.580 images.581
##    67.9650    91.7875   128.1500   154.0450   155.9375   123.4175    81.0000
## images.582 images.583 images.584 images.585 images.586 images.587 images.588
##    42.7650    21.4375     3.7175     0.3625     0.0000     0.0000     0.0000
## images.589 images.590 images.591 images.592 images.593 images.594 images.595
##     0.0000     0.0000     0.5525     1.5200     4.1500    10.6925    27.0050
## images.596 images.597 images.598 images.599 images.600 images.601 images.602
##    53.0525    77.9025   101.3100   115.2400   119.9925   119.4025   123.8025
## images.603 images.604 images.605 images.606 images.607 images.608 images.609
##   137.4750   159.7650   175.2675   171.1025   142.3150    97.3375    58.7600
## images.610 images.611 images.612 images.613 images.614 images.615 images.616
##    29.1075    10.5075     0.9725     0.0475     0.0000     0.0000     0.0000
## images.617 images.618 images.619 images.620 images.621 images.622 images.623
##     0.0000     0.0000     0.0875     1.4725     4.5375     9.6250    21.8200
## images.624 images.625 images.626 images.627 images.628 images.629 images.630
##    50.2525    83.9200   122.2425   150.7725   168.4375   177.1475   184.1450
## images.631 images.632 images.633 images.634 images.635 images.636 images.637
##   191.4625   188.2875   171.3925   138.1700    96.3800    59.4825    30.9575
## images.638 images.639 images.640 images.641 images.642 images.643 images.644
##    13.6400     3.5275     0.0575     0.0000     0.0000     0.0000     0.0000
## images.645 images.646 images.647 images.648 images.649 images.650 images.651
##     0.0000     0.0000     0.0000     0.2475     3.2550     6.7625    12.3875
## images.652 images.653 images.654 images.655 images.656 images.657 images.658
##    32.8750    59.0375    87.0350   124.4875   154.8825   174.5900   176.5825
## images.659 images.660 images.661 images.662 images.663 images.664 images.665
##   161.9300   137.2550   105.6900    71.0550    42.0575    20.8600     9.9050
## images.666 images.667 images.668 images.669 images.670 images.671 images.672
##     3.8225     0.3825     0.0000     0.0000     0.0000     0.0000     0.0000
## images.673 images.674 images.675 images.676 images.677 images.678 images.679
##     0.0000     0.0000     0.0000     0.0125     1.2575     3.0175     5.3100
## images.680 images.681 images.682 images.683 images.684 images.685 images.686
##    13.9100    27.0400    37.2225    49.0025    61.9750    70.9000    69.1600
## images.687 images.688 images.689 images.690 images.691 images.692 images.693
##    62.5950    48.1700    31.1900    18.8000    10.6500     5.6300     2.2500
## images.694 images.695 images.696 images.697 images.698 images.699 images.700
##     0.1125     0.0000     0.0000     0.0000     0.0000     0.0000     0.0000
```

```
## images.701 images.702 images.703 images.704 images.705 images.706 images.707
##      0.0000     0.0000     0.0000     0.0000     0.2575     1.2250     1.7975
## images.708 images.709 images.710 images.711 images.712 images.713 images.714
##      3.7900     6.7650     8.2875     7.9325     9.9025    10.6900     9.3750
## images.715 images.716 images.717 images.718 images.719 images.720 images.721
##      6.9325     5.0400     3.2250     3.0550     2.1400     1.1325     0.1600
## images.722 images.723 images.724 images.725 images.726 images.727 images.728
##      0.0000     0.0000     0.0000     0.0000     0.0000     0.0000     0.0000
## images.729 images.730 images.731 images.732 images.733 images.734 images.735
##      0.0000     0.0000     0.0000     0.0000     0.0000     0.0000     0.0000
## images.736 images.737 images.738 images.739 images.740 images.741 images.742
##      0.0925     0.3075     0.6325     0.7125     0.6200     0.5275     0.4975
## images.743 images.744 images.745 images.746 images.747 images.748 images.749
##      1.0525     1.1950     1.2650     0.8700     0.1150     0.0000     0.0000
## images.750 images.751 images.752 images.753 images.754 images.755 images.756
##      0.0000     0.0000     0.0000     0.0000     0.0000     0.0000     0.0000
## images.757 images.758 images.759 images.760 images.761 images.762 images.763
##      0.0000     0.0000     0.0000     0.0000     0.0000     0.0000     0.0000
## images.764 images.765 images.766 images.767 images.768 images.769 images.770
##      0.0000     0.0000     0.0000     0.0000     0.0000     0.0000     0.0000
## images.771 images.772 images.773 images.774 images.775 images.776 images.777
##      0.0000     0.0000     0.0000     0.0000     0.0000     0.0000     0.0000
## images.778 images.779 images.780 images.781 images.782 images.783 images.784
##      0.0000     0.0000     0.0000     0.0000     0.0000     0.0000     0.0000
```

```
image(1:28,
      1:28,
      matrix(as.numeric(avg_image), nrow=28)[ , 28:1],
      col = gray(seq(0, 1, 0.05)), xlab = "", ylab="")
```



(f) Perform the PCA, and plot the images given by the first three principal directions. [Hint. You can use svd() as I did in the lecture, but you need to center the data first by yourself. Or you can use the function prcomp(), which does the centering automatically. See the book ISL for more details on the function prcomp().]

```
pr_out <- prcomp(matrix(as.numeric(avg_image), nrow=28)[,1:28])
transp <- t(pr_out$rotation[,1:3])
recon <- pr_out$x[,1:3] %*% transp
```

```r
c <- -(pr_out$center)
recon <- scale(recon, center = c, scale=FALSE)
image(1:28, 1:28, matrix(recon, nrow=28)[,28:1], col=gray(seq(0,1,0.05)), xlab="", ylab="")
```