HW4StatsLearn

Table of contents

1 HW 4 Stats Learn

```
library(ISLR)
library(e1071) #For Svm
library(MASS)
```

1.1 Question 1:

a.) A real life data example of when a false negative would be less tolerable than a false positive would be a medical screening for a lethal infectious disease.

b.) A real life data example of when a false positive would be less tolerable than a false negative would be in the case of a security system, should someone break in or not, to know would be more favorable than to not know and someone actually break in to your home.

c.) In the case where a false positive and a false negative are of equal importance would be in the case of a cancer screening, having the illness and thinking you have the illness would still result in the same amount of anxiety and harmful chemo treatment should you choose to seek that.

1.2 Question 2:

a.)

```
attach(USArrests)
```

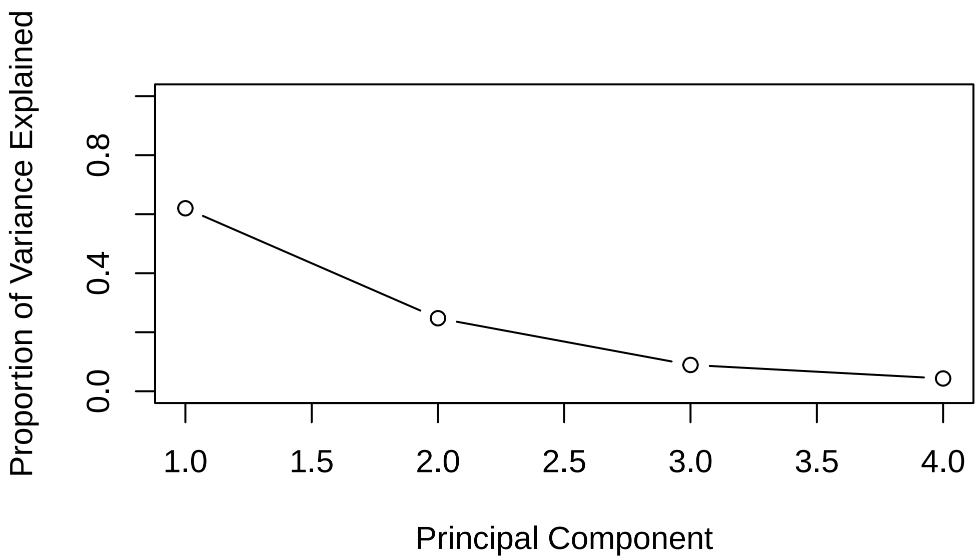
```
pr.out <- prcomp(USArrests,scale=TRUE)
(pr.out$sdev)
```

```
[1] 1.5748783 0.9948694 0.5971291 0.4164494
```

```
pr.var <- pr.out$sdev^2
pve <- pr.var / sum(pr.var)
(pve)
```

```
[1] 0.62006039 0.24744129 0.08914080 0.04335752
```

```
#present our results in plot
plot(pve, xlab="Principal Component", ylab=" Proportion of Variance Explained ",ylim=c(0,1)
```



An explanation: As you can see, the level of variance tends to decrease as the Principal Component iterates up (PCA are elements of the Natural number system only). This is because each Principle components impact decreases from that of the first, thus making the first the most important and the last the last significant.

b.)

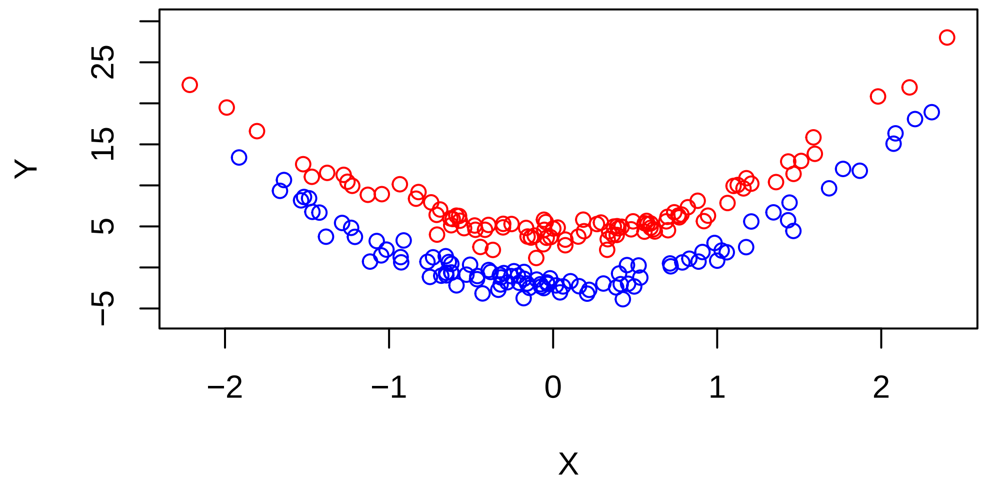
```
#obtain loadings from prcomp() function
loadings<-pr.out$rotation
#scale dataset just to make sure data we use is consistent
USArrests2 <- scale(USArrests)
#convert dataset into matrix, square each value in matrix, and sum them up
#to get the denominator of the equation
sumvalue<-sum(as.matrix(USArrests2)^2)
#multiple these two matrix and then sqaure
num<-(as.matrix(USArrests2)%*%loadings)^2
#calculate the column value for num matrix
colvalue<-c()
for (i in 1:length(num[1,])){
  colvalue[i]<-sum(num[,i])
}
#calculate new pve
pve1<-colvalue/sumvalue
(pve1)
```

```
[1] 0.62006039 0.24744129 0.08914080 0.04335752
```

1.3 Question 3:

a.)

```
set.seed(1)
x1 <- rnorm(200)
x2 <- 4 * x1^2 + 1 + rnorm(200)
y <- as.factor(c(rep(1,100), rep(-1,100)))
x2[y==1] <- x2[y==1] + 3
x2[y==-1] <- x2[y==-1]-3
plot (x1[y==1], x2[y==1], col = "red", xlab = "X", ylab = "Y", ylim = c(-6, 30))
points(x1[y==-1], x2[y==-1], col = "blue")
```




```
myDat <- data.frame(x1,x2,y)
```

```
set.seed(1)
train_index <- sample(1:nrow(myDat), size = 0.8 * nrow(myDat), replace = F)
train_data <- myDat[train_index, ]  # 80% training data
test_data  <- myDat[-train_index, ] # 20% testing data

# Check dimensions
#dim(train_data)
#dim(test_data)
```

b.)

i.)

```
set.seed(1)
svm_linear <- svm(y ~ . , kernel = "linear", data = train_data, cost = 0.01)
summary(svm_linear)
```

```
Call:
svm(formula = y ~ ., data = train_data, kernel = "linear", cost = 0.01)
```

```
Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  linear
       cost:  0.01

Number of Support Vectors:  144

 ( 72 72 )


Number of Classes:  2

Levels:
 -1 1
```

Error Rate Function:

```r
# calculate error rate
calc_error_rate <- function(svm_model, dataset, true_classes) {
  confusion_matrix <- table(predict(svm_model, dataset), true_classes)
  return(1 - sum(diag(confusion_matrix)) / sum(confusion_matrix))
}
```

So things are clear: What is happening here is a function declaration (calc_error_rate) we will be using it a few times so it is best to have it ready to go every time. So we pass a svm model, a data set (train or testing) and the true classes (response variable data). we then pass the svm model and the data set into the built in prediction function. and we take that out put as well as the true classes and put those two values into a table and store it into a variable appropriately called "confusion matrix". We then return from the function 1 - the sum of the correct entries of the confusion matrix divided by all the entries in the confusion matrix. It is a bit esoteric but once you understand confusion matrix and how to calculate error it is very straightforward.

Training error:

```r
cat("Training Error Rate:", 100 * calc_error_rate(svm_linear, train_data, train_data$y), "%\r
```

```
Training Error Rate: 39.375 %
```

Testing error:

```r
cat("Test Error Rate:", 100 * calc_error_rate(svm_linear, test_data, test_data$y), "%\n")
```

```
Test Error Rate: 57.5 %
```

Tune for linear Svm model with respect to cost.

```r
set.seed(1)
svm_tune_linear <- tune(svm, y ~ . , data = myDat, kernel = "linear",
                  ranges = list(cost = seq(0.01, 10, length = 25)))
summary(svm_tune_linear)
```

```

Parameter tuning of 'svm':

- sampling method: 10-fold cross validation

- best parameters:
   cost
 0.8425

- best performance: 0.165

- Detailed performance results:
       cost error dispersion
1   0.01000 0.385 0.12258784
2   0.42625 0.175 0.04859127
3   0.84250 0.165 0.04116363
4   1.25875 0.165 0.04116363
5   1.67500 0.165 0.04116363
6   2.09125 0.165 0.04116363
7   2.50750 0.165 0.04116363
8   2.92375 0.165 0.04116363
9   3.34000 0.165 0.04116363
10  3.75625 0.165 0.04116363
11  4.17250 0.165 0.04116363
12  4.58875 0.165 0.04116363
13  5.00500 0.165 0.04116363
14  5.42125 0.165 0.04116363
15  5.83750 0.165 0.04116363
16  6.25375 0.165 0.04116363
17  6.67000 0.165 0.04116363
```

```
18  7.08625 0.165 0.04116363
19  7.50250 0.165 0.04116363
20  7.91875 0.165 0.04116363
21  8.33500 0.165 0.04116363
22  8.75125 0.165 0.04116363
23  9.16750 0.165 0.04116363
24  9.58375 0.165 0.04116363
25 10.00000 0.165 0.04116363
```

Optimal Cost Value: .8425

Tuned training error:

```
set.seed(1)
svm_linear <- svm(y ~ . , kernel = "linear",
                  data = train_data, cost = svm_tune_linear$best.parameters$cost)

cat("Training Error Rate:", 100 * calc_error_rate(svm_linear, train_data, train_data$y), "%\n
```

```
Training Error Rate: 15.625 %
```

Tuned testing error:

```
cat("Test Error Rate:", 100 * calc_error_rate(svm_linear, test_data, test_data$y), "%\n")
```

```
Test Error Rate: 17.5 %
```

ii)

```
set.seed(1)
svm_poly <- svm(y ~ . , data = train_data, kernel = "poly", degree = 2, cost = .01)
summary(svm_poly)
```

```
Call:
svm(formula = y ~ ., data = train_data, kernel = "poly", degree = 2,
    cost = 0.01)


Parameters:
```

```
   SVM-Type:  C-classification
 SVM-Kernel:  polynomial
       cost:  0.01
     degree:  2
     coef.0:  0

Number of Support Vectors:  153

 ( 75 78 )


Number of Classes:  2

Levels:
 -1 1
```

Training error:

```
cat("Training Error Rate:", 100 * calc_error_rate(svm_poly, train_data, train_data$y), "%\n")
```

```
Training Error Rate: 46.875 %
```

Testing error:

```
cat("Training Error Rate:", 100 * calc_error_rate(svm_poly, test_data, test_data$y), "%\n")
```

```
Training Error Rate: 62.5 %
```

Tune

```
set.seed(1)
svm_tune_poly <- tune(svm, y ~ . , data = myDat, kernel = "poly",
                  ranges = list(cost = seq(0.01, 10, length = 25)))
summary(svm_tune_poly)
```

```
Parameter tuning of 'svm':

- sampling method: 10-fold cross validation
```

```
- best parameters:
  cost
 5.005

- best performance: 0.23

- Detailed performance results:
       cost error dispersion
1   0.01000 0.550 0.09428090
2   0.42625 0.290 0.08096639
3   0.84250 0.280 0.09775252
4   1.25875 0.265 0.08834906
5   1.67500 0.255 0.08959787
6   2.09125 0.250 0.09128709
7   2.50750 0.255 0.08959787
8   2.92375 0.250 0.09128709
9   3.34000 0.245 0.09846037
10  3.75625 0.240 0.10749677
11  4.17250 0.235 0.11067972
12  4.58875 0.235 0.11067972
13  5.00500 0.230 0.11352924
14  5.42125 0.230 0.11352924
15  5.83750 0.235 0.11067972
16  6.25375 0.235 0.11067972
17  6.67000 0.235 0.11067972
18  7.08625 0.235 0.11067972
19  7.50250 0.235 0.11067972
20  7.91875 0.235 0.11067972
21  8.33500 0.230 0.10852547
22  8.75125 0.230 0.10852547
23  9.16750 0.230 0.10852547
24  9.58375 0.230 0.10852547
25 10.00000 0.230 0.10852547
```

Optimal Cost Value: 5.005

Tuned training error:

```r
set.seed(1)
svm_poly <- svm(y ~ . , data = myDat, kernel = "poly",
                   cost = svm_tune_poly$best.parameters$cost)

cat("Training Error Rate:", 100 * calc_error_rate(svm_poly, train_data, train_data$y), "%\n")
```

```
Training Error Rate: 25 %
```

Tuned testing error:

```r
cat("Training Error Rate:", 100 * calc_error_rate(svm_poly, test_data, test_data$y), "%\n")
```

```
Training Error Rate: 12.5 %
```

iii.)

```r
set.seed(1)
svm_radial <- svm(y ~ . , data = train_data, kernel = "radial")
summary(svm_radial)
```

```
Call:
svm(formula = y ~ ., data = train_data, kernel = "radial")


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  radial
       cost:  1

Number of Support Vectors:  55

 ( 27 28 )


Number of Classes:  2

Levels:
 -1 1
```

Training error:

```r
cat("Training Error Rate:", 100 * calc_error_rate(svm_radial, train_data, train_data$y), "%\n
```

```
Training Error Rate: 0.625 %
```

Testing error:

```r
cat("Test Error Rate:", 100 * calc_error_rate(svm_radial, test_data, test_data$y), "%\n")
```

```
Test Error Rate: 2.5 %
```

Tune

```r
set.seed(1)
svm_tune_radial <- tune(svm, y ~ . , data = train_data, kernel = "radial",
                  ranges = list(cost = seq(0.01, 10, length = 25)))
summary(svm_tune_radial)
```

```

Parameter tuning of 'svm':

- sampling method: 10-fold cross validation

- best parameters:
 cost
   10

- best performance: 0.0125

- Detailed performance results:
       cost   error dispersion
1   0.01000 0.46875 0.14508738
2   0.42625 0.09375 0.13581672
3   0.84250 0.06250 0.10206207
4   1.25875 0.06250 0.10206207
5   1.67500 0.05625 0.09969572
6   2.09125 0.04375 0.08359334
7   2.50750 0.03750 0.07905694
8   2.92375 0.03750 0.07905694
9   3.34000 0.03125 0.07933097
10  3.75625 0.03125 0.07933097
11  4.17250 0.03125 0.07933097
12  4.58875 0.03125 0.07933097
13  5.00500 0.02500 0.06038074
14  5.42125 0.02500 0.06038074
15  5.83750 0.02500 0.06038074
```

```
16   6.25375 0.02500 0.06038074
17   6.67000 0.02500 0.06038074
18   7.08625 0.02500 0.06038074
19   7.50250 0.02500 0.06038074
20   7.91875 0.01875 0.04218428
21   8.33500 0.01875 0.04218428
22   8.75125 0.01875 0.04218428
23   9.16750 0.01875 0.04218428
24   9.58375 0.01875 0.04218428
25  10.00000 0.01250 0.02635231
```

Optimal Cost Value: 10

Tuned training error:

```r
svm_radial <- svm(y ~ . , data = train_data, kernel = "radial",
                  cost = svm_tune_radial$best.parameters$cost)

cat("Training Error Rate:", 100 * calc_error_rate(svm_radial, train_data, train_data$y), "%\n
```

```
Training Error Rate: 0 %
```

Tuned testing error:

```r
cat("Test Error Rate:", 100 * calc_error_rate(svm_radial, test_data, test_data$y), "%\n")
```

```
Test Error Rate: 0 %
```

It seems that the best tuned training error rate with respect to cost was, the radial kernel also at 0%.

It seems that the best tuned testing error rate with respect to cost was, the radial kernel at 0%.

1.4 Question 4

a.)

```r
median <- median(Auto$mpg)
Auto$high <- ifelse(Auto$mpg > median, 1, 0)
```

b.)

Tuned Linear Svm

```
set.seed(1)
auto_tune <- tune(svm,high~.,data=Auto,kernel="linear", ranges=list(cost=c(0.001, 0.01, 0.1, 
summary(auto_tune)
```

```
Parameter tuning of 'svm':

- sampling method: 10-fold cross validation

- best parameters:
 cost
    1

- best performance: 0.07424404

- Detailed performance results:
   cost      error dispersion
1 1e-03 0.09351655 0.02013225
2 1e-02 0.08379323 0.02362659
3 1e-01 0.07898470 0.02693908
4 1e+00 0.07424404 0.02693697
5 5e+00 0.08224114 0.03224958
6 1e+01 0.08874314 0.03324316
7 1e+02 0.11389623 0.03717388
```

Optimal Cost Value is 1.

Cross Validation Error: 0.0742

Tuned Radial Svm

```
set.seed(1)
auto_tune_radial <- tune(svm,high~.,data=Auto, kernel="radial",ranges=list(cost=c(0.1,1,10,10
summary(auto_tune_radial)
```

```
Parameter tuning of 'svm':
```

```
- sampling method: 10-fold cross validation

- best parameters:
 cost gamma
    1   0.5

- best performance: 0.05015197

- Detailed performance results:
    cost gamma      error  dispersion
1  1e-01   0.5 0.08034679 0.016888749
2  1e+00   0.5 0.05015197 0.021490457
3  1e+01   0.5 0.05192803 0.020693861
4  1e+02   0.5 0.05190241 0.020682680
5  1e+03   0.5 0.05190241 0.020682680
6  1e-01   1.0 0.29778763 0.037143511
7  1e+00   1.0 0.09787485 0.016102436
8  1e+01   1.0 0.09967585 0.015721409
9  1e+02   1.0 0.09967585 0.015721409
10 1e+03   1.0 0.09967585 0.015721409
11 1e-01   2.0 0.42493331 0.043642327
12 1e+00   2.0 0.20885987 0.011343016
13 1e+01   2.0 0.20923784 0.011048978
14 1e+02   2.0 0.20923784 0.011048978
15 1e+03   2.0 0.20923784 0.011048978
16 1e-01   3.0 0.43733581 0.042085800
17 1e+00   3.0 0.23287331 0.009238892
18 1e+01   3.0 0.23293218 0.009165893
19 1e+02   3.0 0.23293218 0.009165893
20 1e+03   3.0 0.23293218 0.009165893
21 1e-01   4.0 0.44150006 0.040824381
22 1e+00   4.0 0.23744335 0.007731394
23 1e+01   4.0 0.23745903 0.007714556
24 1e+02   4.0 0.23745903 0.007714556
25 1e+03   4.0 0.23745903 0.007714556
```

Optimal Cost Value is 1

Optimal Gamma Value is 0.5

Cross Validation Error: 0.051

Tuned Polynomial Svm

```r
set.seed(1)
auto_tune_poly <- tune(svm,high~.,data=Auto,kernel="polynomial",ranges=list(cost=c(0.1,1,10,1
summary(auto_tune_poly)
```

```
Parameter tuning of 'svm':

- sampling method: 10-fold cross validation

- best parameters:
 cost gamma degree
  0.1   0.5      3

- best performance: 0.09718451

- Detailed performance results:
    cost gamma degree      error dispersion
1  1e-01   0.5      2 0.12944945 0.02279433
2  1e+00   0.5      2 0.12445274 0.02477350
3  1e+01   0.5      2 0.15093365 0.01710134
4  1e+02   0.5      2 0.16260595 0.01488114
5  1e+03   0.5      2 0.16260595 0.01488114
6  1e-01   1.0      2 0.12009739 0.02361922
7  1e+00   1.0      2 0.13590085 0.02078812
8  1e+01   1.0      2 0.16260595 0.01488114
9  1e+02   1.0      2 0.16260595 0.01488114
10 1e+03   1.0      2 0.16260595 0.01488114
11 1e-01   2.0      2 0.12810976 0.02436339
12 1e+00   2.0      2 0.15384974 0.01574844
13 1e+01   2.0      2 0.16260595 0.01488114
14 1e+02   2.0      2 0.16260595 0.01488114
15 1e+03   2.0      2 0.16260595 0.01488114
16 1e-01   3.0      2 0.13510479 0.02112198
17 1e+00   3.0      2 0.16260943 0.01488408
18 1e+01   3.0      2 0.16260943 0.01488408
19 1e+02   3.0      2 0.16260943 0.01488408
20 1e+03   3.0      2 0.16260943 0.01488408
21 1e-01   4.0      2 0.14284539 0.01848847
22 1e+00   4.0      2 0.16260595 0.01488114
23 1e+01   4.0      2 0.16260595 0.01488114
24 1e+02   4.0      2 0.16260595 0.01488114
25 1e+03   4.0      2 0.16260595 0.01488114
```

26	1e-01	0.5	3	0.09718451	0.02835662
27	1e+00	0.5	3	0.12998892	0.04355520
28	1e+01	0.5	3	0.15903015	0.04735260
29	1e+02	0.5	3	0.15905585	0.04733557
30	1e+03	0.5	3	0.15905585	0.04733557
31	1e-01	1.0	3	0.12730425	0.04291076
32	1e+00	1.0	3	0.15550466	0.04586998
33	1e+01	1.0	3	0.15905585	0.04733557
34	1e+02	1.0	3	0.15905585	0.04733557
35	1e+03	1.0	3	0.15905585	0.04733557
36	1e-01	2.0	3	0.15276273	0.04492723
37	1e+00	2.0	3	0.15905585	0.04733557
38	1e+01	2.0	3	0.15905585	0.04733557
39	1e+02	2.0	3	0.15905585	0.04733557
40	1e+03	2.0	3	0.15905585	0.04733557
41	1e-01	3.0	3	0.15905134	0.04733299
42	1e+00	3.0	3	0.15905134	0.04733299
43	1e+01	3.0	3	0.15905134	0.04733299
44	1e+02	3.0	3	0.15905134	0.04733299
45	1e+03	3.0	3	0.15905134	0.04733299
46	1e-01	4.0	3	0.15905585	0.04733557
47	1e+00	4.0	3	0.15905585	0.04733557
48	1e+01	4.0	3	0.15905585	0.04733557
49	1e+02	4.0	3	0.15905585	0.04733557
50	1e+03	4.0	3	0.15905585	0.04733557
51	1e-01	0.5	4	0.27021361	0.16872913
52	1e+00	0.5	4	0.40019261	0.33379672
53	1e+01	0.5	4	0.44671434	0.39388902
54	1e+02	0.5	4	0.44671434	0.39388902
55	1e+03	0.5	4	0.44671434	0.39388902
56	1e-01	1.0	4	0.40976870	0.33894997
57	1e+00	1.0	4	0.44671434	0.39388902
58	1e+01	1.0	4	0.44671434	0.39388902
59	1e+02	1.0	4	0.44671434	0.39388902
60	1e+03	1.0	4	0.44671434	0.39388902
61	1e-01	2.0	4	0.44671434	0.39388902
62	1e+00	2.0	4	0.44671434	0.39388902
63	1e+01	2.0	4	0.44671434	0.39388902
64	1e+02	2.0	4	0.44671434	0.39388902
65	1e+03	2.0	4	0.44671434	0.39388902
66	1e-01	3.0	4	0.44668786	0.39382820
67	1e+00	3.0	4	0.44668786	0.39382820
68	1e+01	3.0	4	0.44668786	0.39382820

```
69 1e+02   3.0      4 0.44668786 0.39382820
70 1e+03   3.0      4 0.44668786 0.39382820
71 1e-01   4.0      4 0.44671434 0.39388902
72 1e+00   4.0      4 0.44671434 0.39388902
73 1e+01   4.0      4 0.44671434 0.39388902
74 1e+02   4.0      4 0.44671434 0.39388902
75 1e+03   4.0      4 0.44671434 0.39388902
```

Optimal Cost Value is 0.1

Optimal Gamma Value is 0.5

Optimal degree is 3

Cross Validation Error: 0.097

e.)

It seems that the best performing SVM kernel was the radial kernel with a

Cross Validation Error: 0.051

1.5 Question 5

a.)

```r
set.seed(1)

trainOJ <- sample(nrow(OJ), 800)
OJ_train <- OJ[trainOJ, ]
OJ_test <- OJ[-trainOJ, ]
```

b.)

```r
set.seed(1)

svm_linear_OJ <- svm(Purchase ~ . , kernel = "linear", data = OJ_train, cost = 0.01)
summary(svm_linear_OJ)
```

```
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
```

Training error: Linear

```r
cat("Training Error Rate:", 100 * calc_error_rate(svm_linear_OJ, OJ_train, OJ_train$Purchase)
```

```
Training Error Rate: 17.5 %
```

Testing error: Linear

```r
cat("Test Error Rate:", 100 * calc_error_rate(svm_linear_OJ, OJ_test, OJ_test$Purchase), "%\n
```

```
Test Error Rate: 17.77778 %
```

Tune: Linear

```r
set.seed(1)

svm_tune_OJ <- tune(svm, Purchase ~ . , data = OJ, kernel = "linear",
                  ranges = list(cost = seq(0.01, 10, length = 50)))
summary(svm_tune_OJ)
```

```
Parameter tuning of 'svm':

- sampling method: 10-fold cross validation
```

```
- best parameters:
     cost
 1.233265

- best performance: 0.1616822

- Detailed performance results:
        cost     error dispersion
1  0.0100000 0.1691589 0.04024604
2  0.2138776 0.1672897 0.03801389
3  0.4177551 0.1691589 0.03671522
4  0.6216327 0.1672897 0.03618270
5  0.8255102 0.1654206 0.03917066
6  1.0293878 0.1626168 0.03945456
7  1.2332653 0.1616822 0.03917066
8  1.4371429 0.1672897 0.03671522
9  1.6410204 0.1663551 0.03654300
10 1.8448980 0.1654206 0.03580522
11 2.0487755 0.1635514 0.03847068
12 2.2526531 0.1644860 0.03870959
13 2.4565306 0.1635514 0.03847068
14 2.6604082 0.1644860 0.03870959
15 2.8642857 0.1626168 0.03795001
16 3.0681633 0.1626168 0.03994348
17 3.2720408 0.1626168 0.03994348
18 3.4759184 0.1626168 0.03994348
19 3.6797959 0.1626168 0.03994348
20 3.8836735 0.1635514 0.04067777
21 4.0875510 0.1635514 0.04067777
22 4.2914286 0.1654206 0.03917066
23 4.4953061 0.1654206 0.03917066
24 4.6991837 0.1654206 0.03917066
25 4.9030612 0.1654206 0.03917066
26 5.1069388 0.1654206 0.03917066
27 5.3108163 0.1654206 0.03917066
28 5.5146939 0.1654206 0.03917066
29 5.7185714 0.1663551 0.03835699
30 5.9224490 0.1663551 0.03835699
31 6.1263265 0.1654206 0.03917066
32 6.3302041 0.1654206 0.03917066
33 6.5340816 0.1663551 0.03759029
34 6.7379592 0.1672897 0.03671522
```

```
35  6.9418367 0.1672897 0.03671522
36  7.1457143 0.1672897 0.03671522
37  7.3495918 0.1672897 0.03671522
38  7.5534694 0.1672897 0.03671522
39  7.7573469 0.1672897 0.03671522
40  7.9612245 0.1672897 0.03671522
41  8.1651020 0.1682243 0.03738318
42  8.3689796 0.1682243 0.03606180
43  8.5728571 0.1682243 0.03606180
44  8.7767347 0.1682243 0.03606180
45  8.9806122 0.1682243 0.03606180
46  9.1844898 0.1682243 0.03606180
47  9.3883673 0.1682243 0.03865942
48  9.5922449 0.1682243 0.03865942
49  9.7961224 0.1682243 0.03865942
50 10.0000000 0.1682243 0.03865942
```

Tuned Training Error: Linear

```
set.seed(1)
svm_linear <- svm(Purchase ~ . , kernel = "linear",
                  data = OJ_train, cost = svm_tune_OJ$best.parameters$cost)

cat("Training Error Rate:", 100 * calc_error_rate(svm_linear, OJ_train, OJ_train$Purchase), "
```

```
Training Error Rate: 16.375 %
```

Tuned Testing Error: Linear

```
cat("Test Error Rate:", 100 * calc_error_rate(svm_linear, OJ_test, OJ_test$Purchase), "%\n")
```

```
Test Error Rate: 15.55556 %
```

c.)

```
set.seed(1)

svm_poly_OJ <- svm(Purchase ~ . , data = OJ_train, kernel = "poly", degree = 2)
summary(svm_poly_OJ)
```

```
Call:
svm(formula = Purchase ~ ., data = OJ_train, kernel = "poly", 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
cat("Training Error Rate:", 100 * calc_error_rate(svm_poly_OJ, OJ_train, OJ_train$Purchase),
```

```
Training Error Rate: 18.25 %
```

```r
cat("Test Error Rate:", 100 * calc_error_rate(svm_poly_OJ, OJ_test, OJ_test$Purchase), "%\n")
```

```
Test Error Rate: 22.22222 %
```

```r
set.seed(1)

svm_tune_poly_OJ <- tune(svm, Purchase ~ . , data = OJ_train, kernel = "poly",
                  degree = 2, ranges = list(cost = seq(0.01, 10, length = 100)))
summary(svm_tune_poly_OJ)
```

```
Parameter tuning of 'svm':

- sampling method: 10-fold cross validation
```

```
- best parameters:
     cost
 2.431818

- best performance: 0.17125

- Detailed performance results:
          cost   error dispersion
1    0.0100000 0.39125 0.04210189
2    0.1109091 0.31750 0.04937104
3    0.2118182 0.22375 0.04016027
4    0.3127273 0.20000 0.04208127
5    0.4136364 0.20125 0.04185375
6    0.5145455 0.20500 0.04338138
7    0.6154545 0.20500 0.03961621
8    0.7163636 0.20250 0.04031129
9    0.8172727 0.20250 0.04322101
10   0.9181818 0.20250 0.04479893
11   1.0190909 0.20125 0.03928617
12   1.1200000 0.20125 0.04016027
13   1.2209091 0.19625 0.04411554
14   1.3218182 0.19250 0.04495368
15   1.4227273 0.19125 0.04411554
16   1.5236364 0.19000 0.04322101
17   1.6245455 0.18875 0.04308019
18   1.7254545 0.18500 0.04199868
19   1.8263636 0.18375 0.04411554
20   1.9272727 0.18125 0.04177070
21   2.0281818 0.18125 0.04177070
22   2.1290909 0.17875 0.04210189
23   2.2300000 0.17375 0.03793727
24   2.3309091 0.17375 0.03747684
25   2.4318182 0.17125 0.03729108
26   2.5327273 0.17375 0.03884174
27   2.6336364 0.17375 0.03884174
28   2.7345455 0.17500 0.03818813
29   2.8354545 0.17500 0.03818813
30   2.9363636 0.17625 0.03793727
31   3.0372727 0.17625 0.03793727
32   3.1381818 0.17750 0.03670453
33   3.2390909 0.18000 0.03545341
34   3.3400000 0.18000 0.03545341
```

35	3.4409091	0.17875	0.03586723
36	3.5418182	0.17875	0.03586723
37	3.6427273	0.17875	0.03537988
38	3.7436364	0.17875	0.03537988
39	3.8445455	0.18125	0.03498512
40	3.9454545	0.18250	0.03395258
41	4.0463636	0.18250	0.03395258
42	4.1472727	0.18375	0.03438447
43	4.2481818	0.18500	0.03425801
44	4.3490909	0.18500	0.03425801
45	4.4500000	0.18500	0.03425801
46	4.5509091	0.18375	0.03387579
47	4.6518182	0.18375	0.03387579
48	4.7527273	0.18250	0.03496029
49	4.8536364	0.18250	0.03496029
50	4.9545455	0.18250	0.03496029
51	5.0554545	0.18250	0.03496029
52	5.1563636	0.18250	0.03496029
53	5.2572727	0.18375	0.03537988
54	5.3581818	0.18625	0.03143004
55	5.4590909	0.18625	0.03143004
56	5.5600000	0.18375	0.03064696
57	5.6609091	0.18375	0.03064696
58	5.7618182	0.18500	0.03162278
59	5.8627273	0.18625	0.03304563
60	5.9636364	0.18625	0.03304563
61	6.0645455	0.18500	0.03162278
62	6.1654545	0.18500	0.03162278
63	6.2663636	0.18500	0.03162278
64	6.3672727	0.18500	0.03162278
65	6.4681818	0.18500	0.03162278
66	6.5690909	0.18500	0.03162278
67	6.6700000	0.18625	0.03356689
68	6.7709091	0.18500	0.03162278
69	6.8718182	0.18500	0.03162278
70	6.9727273	0.18500	0.03162278
71	7.0736364	0.18625	0.03251602
72	7.1745455	0.18500	0.03525699
73	7.2754545	0.18375	0.03387579
74	7.3763636	0.18375	0.03387579
75	7.4772727	0.18375	0.03387579
76	7.5781818	0.18250	0.03291403
77	7.6790909	0.18250	0.03291403

```
78   7.7800000 0.18250 0.03291403
79   7.8809091 0.18125 0.03448530
80   7.9818182 0.18125 0.03448530
81   8.0827273 0.18125 0.03240906
82   8.1836364 0.18000 0.03129164
83   8.2845455 0.18125 0.02841288
84   8.3854545 0.18250 0.02898755
85   8.4863636 0.18250 0.02898755
86   8.5872727 0.18000 0.02838231
87   8.6881818 0.17875 0.02766993
88   8.7890909 0.17875 0.02766993
89   8.8900000 0.17875 0.02766993
90   8.9909091 0.17875 0.02766993
91   9.0918182 0.17750 0.02751262
92   9.1927273 0.17750 0.02751262
93   9.2936364 0.17750 0.02751262
94   9.3945455 0.17750 0.02751262
95   9.4954545 0.17750 0.02751262
96   9.5963636 0.17875 0.02766993
97   9.6972727 0.17875 0.02766993
98   9.7981818 0.17875 0.02766993
99   9.8990909 0.17875 0.02766993
100 10.0000000 0.18125 0.02779513
```

```
set.seed(1)
svm_poly_OJ <- svm(Purchase ~ . , data = OJ_train, kernel = "poly",
                 degree = 2, cost = svm_tune_poly_OJ$best.parameters$cost)

cat("Training Error Rate:", 100 * calc_error_rate(svm_poly_OJ, OJ_train, OJ_train$Purchase),
```

```
Training Error Rate: 15.625 %
```

```
cat("Test Error Rate:", 100 * calc_error_rate(svm_poly_OJ, OJ_test, OJ_test$Purchase), "%\n")
```

```
Test Error Rate: 20.74074 %
```

d.)

```
set.seed(1)

svm_radial_OJ <- svm(Purchase ~ . , data = OJ_train, kernel = "radial")
summary(svm_radial_OJ)
```

```
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
```

Train

```r
cat("Training Error Rate:", 100 * calc_error_rate(svm_radial_OJ, OJ_train, OJ_train$Purchase
```

```
Training Error Rate: 15.125 %
```

Test

```r
cat("Test Error Rate:", 100 * calc_error_rate(svm_radial_OJ, OJ_test, OJ_test$Purchase), "%\
```

```
Test Error Rate: 18.51852 %
```

Tune

```r
set.seed(1)

svm_tune_OJ_radial <- tune(svm, Purchase ~ . , data = OJ_train, kernel = "radial",
                  ranges = list(cost = seq(0.01, 10, length = 100)))
summary(svm_tune_OJ_radial)
```

```
Parameter tuning of 'svm':

- sampling method: 10-fold cross validation

- best parameters:
      cost
 0.5145455

- best performance: 0.16625

- Detailed performance results:
          cost   error dispersion
1    0.0100000 0.39375 0.04007372
2    0.1109091 0.18625 0.02853482
3    0.2118182 0.18250 0.03238227
4    0.3127273 0.17875 0.03230175
5    0.4136364 0.17625 0.02531057
6    0.5145455 0.16625 0.02433134
7    0.6154545 0.16875 0.02301117
8    0.7163636 0.16750 0.02776389
9    0.8172727 0.17000 0.02513851
10   0.9181818 0.16750 0.02220485
11   1.0190909 0.17000 0.02058182
12   1.1200000 0.17250 0.02188988
13   1.2209091 0.17250 0.02108185
14   1.3218182 0.17250 0.02266912
15   1.4227273 0.17375 0.02389938
16   1.5236364 0.17625 0.02161050
17   1.6245455 0.17625 0.02161050
18   1.7254545 0.17750 0.02188988
19   1.8263636 0.17625 0.02079162
20   1.9272727 0.17625 0.02079162
21   2.0281818 0.17750 0.02188988
22   2.1290909 0.17875 0.02128673
23   2.2300000 0.17875 0.02128673
24   2.3309091 0.17750 0.02266912
25   2.4318182 0.17750 0.02266912
26   2.5327273 0.17625 0.02239947
27   2.6336364 0.17625 0.02239947
28   2.7345455 0.17625 0.02239947
29   2.8354545 0.17625 0.02239947
30   2.9363636 0.17625 0.02239947
```

31	3.0372727	0.17625	0.02239947
32	3.1381818	0.17750	0.02266912
33	3.2390909	0.17750	0.02266912
34	3.3400000	0.17750	0.02266912
35	3.4409091	0.17875	0.02360703
36	3.5418182	0.17875	0.02360703
37	3.6427273	0.17875	0.02360703
38	3.7436364	0.17875	0.02360703
39	3.8445455	0.18000	0.02371708
40	3.9454545	0.18000	0.02371708
41	4.0463636	0.18125	0.02301117
42	4.1472727	0.18125	0.02301117
43	4.2481818	0.18125	0.02301117
44	4.3490909	0.18125	0.02301117
45	4.4500000	0.18125	0.02144923
46	4.5509091	0.18125	0.02144923
47	4.6518182	0.18125	0.02144923
48	4.7527273	0.18125	0.02144923
49	4.8536364	0.18125	0.02144923
50	4.9545455	0.18000	0.02220485
51	5.0554545	0.18000	0.02220485
52	5.1563636	0.18000	0.02220485
53	5.2572727	0.18000	0.02220485
54	5.3581818	0.18000	0.02220485
55	5.4590909	0.18000	0.02220485
56	5.5600000	0.18000	0.02220485
57	5.6609091	0.18000	0.02220485
58	5.7618182	0.18000	0.02220485
59	5.8627273	0.18000	0.02220485
60	5.9636364	0.18000	0.02220485
61	6.0645455	0.18000	0.02220485
62	6.1654545	0.18000	0.02220485
63	6.2663636	0.18125	0.02301117
64	6.3672727	0.18125	0.02301117
65	6.4681818	0.18125	0.02301117
66	6.5690909	0.18125	0.02301117
67	6.6700000	0.18125	0.02301117
68	6.7709091	0.18125	0.02301117
69	6.8718182	0.18125	0.02301117
70	6.9727273	0.18250	0.02371708
71	7.0736364	0.18375	0.02503470
72	7.1745455	0.18250	0.02443813
73	7.2754545	0.18250	0.02443813

```
74    7.3763636 0.18375 0.02638523
75    7.4772727 0.18375 0.02638523
76    7.5781818 0.18375 0.02638523
77    7.6790909 0.18375 0.02638523
78    7.7800000 0.18375 0.02638523
79    7.8809091 0.18375 0.02638523
80    7.9818182 0.18375 0.02638523
81    8.0827273 0.18250 0.02648375
82    8.1836364 0.18125 0.02447363
83    8.2845455 0.18000 0.02443813
84    8.3854545 0.18000 0.02443813
85    8.4863636 0.18000 0.02443813
86    8.5872727 0.18000 0.02443813
87    8.6881818 0.18000 0.02443813
88    8.7890909 0.18375 0.02703521
89    8.8900000 0.18375 0.02703521
90    8.9909091 0.18375 0.02703521
91    9.0918182 0.18375 0.02703521
92    9.1927273 0.18375 0.02703521
93    9.2936364 0.18625 0.02853482
94    9.3945455 0.18625 0.02853482
95    9.4954545 0.18625 0.02853482
96    9.5963636 0.18625 0.02853482
97    9.6972727 0.18625 0.02853482
98    9.7981818 0.18625 0.02853482
99    9.8990909 0.18625 0.02853482
100  10.0000000 0.18625 0.02853482
```

Train Tune Error

```r
set.seed(1)
svm_radial_OJ <- svm(Purchase ~ . , data = OJ_train, kernel = "radial",
                   cost = svm_tune_OJ_radial$best.parameters$cost)

cat("Training Error Rate:", 100 * calc_error_rate(svm_radial_OJ, OJ_train, OJ_train$Purchase)
```

```
Training Error Rate: 14.875 %
```

Test Tune Error

```
cat("Test Error Rate:", 100 * calc_error_rate(svm_radial_OJ, OJ_test, OJ_test$Purchase), "%\n
```

```
Test Error Rate: 17.77778 %
```

e.)

The best performing SVM Kernel was: The tuned radial kernel

Training Error: 14.875 %

However the best performing SVM Kernel for testing data was the tuned linear kernel

Test Error: 15.55556 %

1.6 Question 6

a.)

```
set.seed(1)

trainIrisIndex <- sample(1:nrow(iris), 0.8 * nrow(iris))
train_iris <- iris[trainIrisIndex, ]
test_iris <- iris[-trainIrisIndex, ]

#y is Species
```

b.) & c.)

i.)

```
set.seed(1)

svm_tune_iris_linear <- tune(svm, Species ~ . , data = train_iris, kernel = "linear",
                  ranges = list(cost = seq(0.01, 10, length = 50)))
summary(svm_tune_iris_linear)
```

```
Parameter tuning of 'svm':

- sampling method: 10-fold cross validation

- best parameters:
```

```
      cost
 0.6216327

- best performance: 0.025

- Detailed performance results:
        cost      error dispersion
1   0.0100000 0.20000000 0.16292087
2   0.2138776 0.03333333 0.04303315
3   0.4177551 0.03333333 0.04303315
4   0.6216327 0.02500000 0.04025382
5   0.8255102 0.02500000 0.04025382
6   1.0293878 0.03333333 0.04303315
7   1.2332653 0.04166667 0.04392052
8   1.4371429 0.04166667 0.04392052
9   1.6410204 0.05000000 0.05826716
10  1.8448980 0.04166667 0.05892557
11  2.0487755 0.04166667 0.05892557
12  2.2526531 0.04166667 0.05892557
13  2.4565306 0.05000000 0.05826716
14  2.6604082 0.05000000 0.05826716
15  2.8642857 0.05000000 0.05826716
16  3.0681633 0.05000000 0.05826716
17  3.2720408 0.04166667 0.05892557
18  3.4759184 0.04166667 0.05892557
19  3.6797959 0.04166667 0.05892557
20  3.8836735 0.04166667 0.05892557
21  4.0875510 0.04166667 0.05892557
22  4.2914286 0.04166667 0.05892557
23  4.4953061 0.04166667 0.05892557
24  4.6991837 0.04166667 0.05892557
25  4.9030612 0.04166667 0.05892557
26  5.1069388 0.04166667 0.05892557
27  5.3108163 0.04166667 0.05892557
28  5.5146939 0.04166667 0.05892557
29  5.7185714 0.04166667 0.05892557
30  5.9224490 0.04166667 0.05892557
31  6.1263265 0.04166667 0.05892557
32  6.3302041 0.04166667 0.05892557
33  6.5340816 0.04166667 0.05892557
34  6.7379592 0.04166667 0.05892557
35  6.9418367 0.04166667 0.05892557
36  7.1457143 0.04166667 0.05892557
```

```
37  7.3495918 0.04166667 0.05892557
38  7.5534694 0.04166667 0.05892557
39  7.7573469 0.04166667 0.05892557
40  7.9612245 0.04166667 0.05892557
41  8.1651020 0.04166667 0.05892557
42  8.3689796 0.04166667 0.05892557
43  8.5728571 0.04166667 0.05892557
44  8.7767347 0.04166667 0.05892557
45  8.9806122 0.04166667 0.05892557
46  9.1844898 0.04166667 0.05892557
47  9.3883673 0.04166667 0.05892557
48  9.5922449 0.04166667 0.05892557
49  9.7961224 0.04166667 0.05892557
50 10.0000000 0.04166667 0.05892557
```

Train Error

```r
set.seed(1)
svm_linear_iris <- svm(Species ~ . , kernel = "linear",
                  data = train_iris, cost = svm_tune_iris_linear$best.parameters$cost)

cat("Training Error Rate:", 100 * calc_error_rate(svm_linear_iris, train_iris, train_iris$Spe
```

```
Training Error Rate: 1.666667 %
```

Test Error

```r
cat("Test Error Rate:", 100 * calc_error_rate(svm_linear_iris, test_iris, test_iris$Species)
```

```
Test Error Rate: 0 %
```

ii.)

```r
set.seed(1)

svm_tune_poly_iris <- tune(svm, Species ~ . , data = train_iris, kernel = "poly",
                  degree = 2, ranges = list(cost = seq(0.01, 10, length = 100)))
summary(svm_tune_poly_iris)
```

```
Parameter tuning of 'svm':

- sampling method: 10-fold cross validation

- best parameters:
     cost
 1.523636

- best performance: 0.08333333

- Detailed performance results:
          cost      error dispersion
1    0.0100000 0.69166667 0.14724843
2    0.1109091 0.24166667 0.12076147
3    0.2118182 0.17500000 0.09976825
4    0.3127273 0.15000000 0.10243938
5    0.4136364 0.13333333 0.10540926
6    0.5145455 0.12500000 0.10577463
7    0.6154545 0.11666667 0.11249143
8    0.7163636 0.10833333 0.11145779
9    0.8172727 0.10000000 0.10243938
10   0.9181818 0.10000000 0.10243938
11   1.0190909 0.10000000 0.10243938
12   1.1200000 0.10000000 0.10243938
13   1.2209091 0.10000000 0.10243938
14   1.3218182 0.10000000 0.10243938
15   1.4227273 0.09166667 0.08286908
16   1.5236364 0.08333333 0.07856742
17   1.6245455 0.08333333 0.07856742
18   1.7254545 0.09166667 0.07296625
19   1.8263636 0.09166667 0.07296625
20   1.9272727 0.10000000 0.09460770
21   2.0281818 0.10000000 0.07657805
22   2.1290909 0.09166667 0.06148873
23   2.2300000 0.09166667 0.06148873
24   2.3309091 0.09166667 0.06148873
25   2.4318182 0.09166667 0.06148873
26   2.5327273 0.09166667 0.06148873
27   2.6336364 0.09166667 0.06148873
28   2.7345455 0.10833333 0.09662515
29   2.8354545 0.11666667 0.11915339
30   2.9363636 0.11666667 0.11915339
```

31	3.0372727	0.11666667	0.11915339
32	3.1381818	0.11666667	0.11915339
33	3.2390909	0.11666667	0.11915339
34	3.3400000	0.11666667	0.11915339
35	3.4409091	0.11666667	0.11915339
36	3.5418182	0.11666667	0.11915339
37	3.6427273	0.11666667	0.11915339
38	3.7436364	0.11666667	0.11915339
39	3.8445455	0.12500000	0.11283387
40	3.9454545	0.12500000	0.11283387
41	4.0463636	0.12500000	0.11283387
42	4.1472727	0.12500000	0.11283387
43	4.2481818	0.12500000	0.11283387
44	4.3490909	0.11666667	0.08958064
45	4.4500000	0.11666667	0.08958064
46	4.5509091	0.12500000	0.11283387
47	4.6518182	0.12500000	0.11283387
48	4.7527273	0.12500000	0.11283387
49	4.8536364	0.13333333	0.11915339
50	4.9545455	0.13333333	0.11915339
51	5.0554545	0.13333333	0.11915339
52	5.1563636	0.13333333	0.11915339
53	5.2572727	0.13333333	0.11915339
54	5.3581818	0.13333333	0.11249143
55	5.4590909	0.13333333	0.11249143
56	5.5600000	0.13333333	0.11249143
57	5.6609091	0.13333333	0.11249143
58	5.7618182	0.13333333	0.11249143
59	5.8627273	0.13333333	0.11249143
60	5.9636364	0.13333333	0.11249143
61	6.0645455	0.13333333	0.11249143
62	6.1654545	0.13333333	0.11249143
63	6.2663636	0.13333333	0.11249143
64	6.3672727	0.13333333	0.11249143
65	6.4681818	0.13333333	0.11249143
66	6.5690909	0.13333333	0.11249143
67	6.6700000	0.13333333	0.11249143
68	6.7709091	0.13333333	0.11249143
69	6.8718182	0.13333333	0.11249143
70	6.9727273	0.13333333	0.11249143
71	7.0736364	0.13333333	0.11249143
72	7.1745455	0.14166667	0.11817804
73	7.2754545	0.14166667	0.11817804

```
74    7.3763636 0.14166667 0.11817804
75    7.4772727 0.14166667 0.11817804
76    7.5781818 0.14166667 0.11817804
77    7.6790909 0.14166667 0.11817804
78    7.7800000 0.14166667 0.11817804
79    7.8809091 0.14166667 0.11817804
80    7.9818182 0.14166667 0.11817804
81    8.0827273 0.14166667 0.11817804
82    8.1836364 0.14166667 0.11817804
83    8.2845455 0.14166667 0.11817804
84    8.3854545 0.14166667 0.11817804
85    8.4863636 0.14166667 0.11817804
86    8.5872727 0.13333333 0.11915339
87    8.6881818 0.13333333 0.11915339
88    8.7890909 0.13333333 0.11915339
89    8.8900000 0.13333333 0.11915339
90    8.9909091 0.13333333 0.11915339
91    9.0918182 0.13333333 0.11915339
92    9.1927273 0.13333333 0.11915339
93    9.2936364 0.13333333 0.11915339
94    9.3945455 0.13333333 0.11915339
95    9.4954545 0.13333333 0.11915339
96    9.5963636 0.13333333 0.11915339
97    9.6972727 0.13333333 0.11915339
98    9.7981818 0.13333333 0.11915339
99    9.8990909 0.13333333 0.11915339
100  10.0000000 0.13333333 0.11915339
```

Train Error

```r
set.seed(1)
svm_poly_iris <- svm(Species ~ . , data = train_iris, kernel = "poly",
                 degree = 2, cost = svm_tune_poly_iris$best.parameters$cost)

cat("Training Error Rate:", 100 * calc_error_rate(svm_poly_iris, train_iris, train_iris$Spec
```

```
Training Error Rate: 8.333333 %
```

Test Error

```r
cat("Test Error Rate:", 100 * calc_error_rate(svm_poly_iris, test_iris, test_iris$Species), '
```

```
Test Error Rate: 13.33333 %
```

iii.)

```r
set.seed(1)

svm_tune_iris_radial <- tune(svm, Species ~ . , data = train_iris, kernel = "radial",
                     ranges = list(cost = seq(0.01, 10, length = 100)))
summary(svm_tune_iris_radial)
```

```
Parameter tuning of 'svm':

- sampling method: 10-fold cross validation

- best parameters:
     cost
 1.019091

- best performance: 0.04166667

- Detailed performance results:
          cost      error dispersion
1    0.0100000 0.69166667 0.14724843
2    0.1109091 0.10000000 0.06573422
3    0.2118182 0.05000000 0.05826716
4    0.3127273 0.05000000 0.05826716
5    0.4136364 0.05000000 0.05826716
6    0.5145455 0.05833333 0.05624571
7    0.6154545 0.05000000 0.04303315
8    0.7163636 0.05000000 0.04303315
9    0.8172727 0.05000000 0.04303315
10   0.9181818 0.05000000 0.05826716
11   1.0190909 0.04166667 0.04392052
12   1.1200000 0.05000000 0.05826716
13   1.2209091 0.05000000 0.05826716
14   1.3218182 0.05000000 0.05826716
15   1.4227273 0.05000000 0.05826716
16   1.5236364 0.05000000 0.05826716
```

17	1.6245455	0.05000000	0.05826716
18	1.7254545	0.05000000	0.05826716
19	1.8263636	0.05833333	0.05624571
20	1.9272727	0.05833333	0.05624571
21	2.0281818	0.05833333	0.05624571
22	2.1290909	0.05833333	0.05624571
23	2.2300000	0.05833333	0.05624571
24	2.3309091	0.05833333	0.05624571
25	2.4318182	0.05833333	0.05624571
26	2.5327273	0.05833333	0.05624571
27	2.6336364	0.05833333	0.05624571
28	2.7345455	0.05833333	0.05624571
29	2.8354545	0.05833333	0.05624571
30	2.9363636	0.05833333	0.05624571
31	3.0372727	0.05833333	0.05624571
32	3.1381818	0.05833333	0.05624571
33	3.2390909	0.05833333	0.05624571
34	3.3400000	0.05833333	0.05624571
35	3.4409091	0.05833333	0.05624571
36	3.5418182	0.05833333	0.05624571
37	3.6427273	0.05833333	0.05624571
38	3.7436364	0.06666667	0.06573422
39	3.8445455	0.06666667	0.06573422
40	3.9454545	0.06666667	0.06573422
41	4.0463636	0.06666667	0.06573422
42	4.1472727	0.06666667	0.06573422
43	4.2481818	0.06666667	0.06573422
44	4.3490909	0.06666667	0.06573422
45	4.4500000	0.06666667	0.06573422
46	4.5509091	0.06666667	0.06573422
47	4.6518182	0.06666667	0.06573422
48	4.7527273	0.06666667	0.06573422
49	4.8536364	0.06666667	0.06573422
50	4.9545455	0.05833333	0.06860605
51	5.0554545	0.05833333	0.06860605
52	5.1563636	0.05833333	0.06860605
53	5.2572727	0.05833333	0.06860605
54	5.3581818	0.05833333	0.06860605
55	5.4590909	0.05833333	0.06860605
56	5.5600000	0.05833333	0.06860605
57	5.6609091	0.05833333	0.06860605
58	5.7618182	0.05833333	0.06860605
59	5.8627273	0.05833333	0.06860605

```
60    5.9636364 0.05833333 0.06860605
61    6.0645455 0.05833333 0.06860605
62    6.1654545 0.05833333 0.06860605
63    6.2663636 0.05833333 0.06860605
64    6.3672727 0.05833333 0.06860605
65    6.4681818 0.05833333 0.06860605
66    6.5690909 0.05833333 0.06860605
67    6.6700000 0.05833333 0.06860605
68    6.7709091 0.05833333 0.06860605
69    6.8718182 0.05833333 0.06860605
70    6.9727273 0.05833333 0.06860605
71    7.0736364 0.05833333 0.06860605
72    7.1745455 0.05833333 0.06860605
73    7.2754545 0.05833333 0.06860605
74    7.3763636 0.05833333 0.06860605
75    7.4772727 0.05833333 0.06860605
76    7.5781818 0.05833333 0.06860605
77    7.6790909 0.05833333 0.06860605
78    7.7800000 0.05833333 0.06860605
79    7.8809091 0.05833333 0.06860605
80    7.9818182 0.05833333 0.06860605
81    8.0827273 0.05833333 0.06860605
82    8.1836364 0.05833333 0.06860605
83    8.2845455 0.05833333 0.06860605
84    8.3854545 0.05833333 0.06860605
85    8.4863636 0.05833333 0.06860605
86    8.5872727 0.05833333 0.06860605
87    8.6881818 0.05833333 0.06860605
88    8.7890909 0.05833333 0.06860605
89    8.8900000 0.05833333 0.06860605
90    8.9909091 0.05833333 0.06860605
91    9.0918182 0.05833333 0.06860605
92    9.1927273 0.05833333 0.06860605
93    9.2936364 0.05833333 0.06860605
94    9.3945455 0.05833333 0.06860605
95    9.4954545 0.05833333 0.06860605
96    9.5963636 0.05833333 0.06860605
97    9.6972727 0.06666667 0.06573422
98    9.7981818 0.06666667 0.06573422
99    9.8990909 0.06666667 0.06573422
100  10.0000000 0.06666667 0.06573422
```

Train Error

```r
set.seed(1)
svm_radial_iris <- svm(Species ~ . , data = train_iris, kernel = "radial",
                    cost = svm_tune_iris_radial$best.parameters$cost)

cat("Training Error Rate:", 100 * calc_error_rate(svm_radial_iris, train_iris, train_iris$Spe
```

```
Training Error Rate: 3.333333 %
```

Test Error

```r
cat("Test Error Rate:", 100 * calc_error_rate(svm_radial_iris, test_iris, test_iris$Species)
```

```
Test Error Rate: 3.333333 %
```

It seems that the linear SVM model has the best Testing and Training error at 1.666667 % and 0% respectively.

d.)

N/A

1.7 Question 7

a.)

```r
library(dplyr) #this is simply so we can use the pipe command for feasability
```

```
Attaching package: 'dplyr'

The following object is masked from 'package:MASS':

    select

The following objects are masked from 'package:stats':

    filter, lag
```

```
The following objects are masked from 'package:base':

    intersect, setdiff, setequal, union
```

```r
data(Boston)
summary(Boston)
```

```
      crim                zn             indus            chas        
 Min.   : 0.00632   Min.   :  0.00   Min.   : 0.46   Min.   :0.00000  
 1st Qu.: 0.08205   1st Qu.:  0.00   1st Qu.: 5.19   1st Qu.:0.00000  
 Median : 0.25651   Median :  0.00   Median : 9.69   Median :0.00000  
 Mean   : 3.61352   Mean   : 11.36   Mean   :11.14   Mean   :0.06917  
 3rd Qu.: 3.67708   3rd Qu.: 12.50   3rd Qu.:18.10   3rd Qu.:0.00000  
 Max.   :88.97620   Max.   :100.00   Max.   :27.74   Max.   :1.00000  
      nox               rm             age              dis        
 Min.   :0.3850   Min.   :3.561   Min.   :  2.90   Min.   : 1.130  
 1st Qu.:0.4490   1st Qu.:5.886   1st Qu.: 45.02   1st Qu.: 2.100  
 Median :0.5380   Median :6.208   Median : 77.50   Median : 3.207  
 Mean   :0.5547   Mean   :6.285   Mean   : 68.57   Mean   : 3.795  
 3rd Qu.:0.6240   3rd Qu.:6.623   3rd Qu.: 94.08   3rd Qu.: 5.188  
 Max.   :0.8710   Max.   :8.780   Max.   :100.00   Max.   :12.127  
      rad              tax           ptratio          black       
 Min.   : 1.000   Min.   :187.0   Min.   :12.60   Min.   :  0.32  
 1st Qu.: 4.000   1st Qu.:279.0   1st Qu.:17.40   1st Qu.:375.38  
 Median : 5.000   Median :330.0   Median :19.05   Median :391.44  
 Mean   : 9.549   Mean   :408.2   Mean   :18.46   Mean   :356.67  
 3rd Qu.:24.000   3rd Qu.:666.0   3rd Qu.:20.20   3rd Qu.:396.23  
 Max.   :24.000   Max.   :711.0   Max.   :22.00   Max.   :396.90  
     lstat            medv      
 Min.   : 1.73   Min.   : 5.00  
 1st Qu.: 6.95   1st Qu.:17.02  
 Median :11.36   Median :21.20  
 Mean   :12.65   Mean   :22.53  
 3rd Qu.:16.95   3rd Qu.:25.00  
 Max.   :37.97   Max.   :50.00  
```

```r
#We used the pipe opperator here, to rename the data set and also remove the chas column
BOS = Boston %>% select(- c(chas))
head(BOS)
```

```
     crim zn indus   nox    rm  age    dis rad tax ptratio  black lstat medv
```

```
1 0.00632 18  2.31 0.538 6.575 65.2 4.0900   1 296    15.3 396.90  4.98 24.0
2 0.02731  0  7.07 0.469 6.421 78.9 4.9671   2 242    17.8 396.90  9.14 21.6
3 0.02729  0  7.07 0.469 7.185 61.1 4.9671   2 242    17.8 392.83  4.03 34.7
4 0.03237  0  2.18 0.458 6.998 45.8 6.0622   3 222    18.7 394.63  2.94 33.4
5 0.06905  0  2.18 0.458 7.147 54.2 6.0622   3 222    18.7 396.90  5.33 36.2
6 0.02985  0  2.18 0.458 6.430 58.7 6.0622   3 222    18.7 394.12  5.21 28.7
```

We removed *chas* as it does not aid in PCA, we could do one hot encoding but I do not think that is necessary for this assignment.

b.)

```r
set.seed(1)
train_index_BOS <- sample(1:nrow(BOS), 0.8 * nrow(BOS))
train_BOS <- BOS[train_index_BOS,]
test_BOS <- BOS[-train_index_BOS,]
```

c.)

```r
set.seed(1)
BOS_pca <- prcomp(train_BOS %>% select(-medv), scale= TRUE)
summary(BOS_pca)
```

```
Importance of components:
                          PC1    PC2     PC3     PC4    PC5     PC6     PC7
Standard deviation     2.4917 1.1646 1.07883 0.88720 0.7990 0.71136 0.64019
Proportion of Variance 0.5174 0.1130 0.09699 0.06559 0.0532 0.04217 0.03415
Cumulative Proportion  0.5174 0.6304 0.72741 0.79300 0.8462 0.88837 0.92252
                           PC8     PC9    PC10    PC11   PC12
Standard deviation     0.52282 0.48026 0.42882 0.41361 0.2660
Proportion of Variance 0.02278 0.01922 0.01532 0.01426 0.0059
Cumulative Proportion  0.94530 0.96452 0.97985 0.99410 1.0000
```

d.)

To explain AT LEAST 80% of the variation in this data set you would need 5 PC's.