STAT413_HW4_Hongyu_Mao

Chapter 8 Exercise 9

(a)

```
library(ISLR)
library('caret')
```

```
## Loading required package: lattice
```

```
## Loading required package: ggplot2
```

```
library('tidyverse')
```

```
## ── Attaching packages ─────────────────────────────────────── tidyverse 1.2.1 ──
```

```
## ✔ tibble  2.0.1     ✔ purrr   0.2.5
## ✔ tidyr   0.8.2     ✔ dplyr   0.7.8
## ✔ readr   1.3.1     ✔ stringr 1.3.1
## ✔ tibble  2.0.1     ✔ forcats 0.3.0
```

```
## ── Conflicts ────────────────────────────────────────── tidyverse_conflicts() ──
## ✖ dplyr::filter() masks stats::filter()
## ✖ dplyr::lag()    masks stats::lag()
## ✖ purrr::lift()   masks caret::lift()
```

```
library('ggthemes')
library('rpart')
library('rpart.plot')
library('knitr')
library('kableExtra')
library(tree)
library("e1071")

set.seed(2)

dat <- OJ
train_idx <- sample(c(1:1070), size = 800)
train <- dat[train_idx, ]
test <- dat[-train_idx, ]
```

(b)

```
# tree <- rpart(Purchase ~ ., data = train, method = 'class', control = rpart.control
(cp = 0))

tree <- tree(Purchase ~ ., data = train)
summary(tree)
```

```
##
## Classification tree:
## tree(formula = Purchase ~ ., data = train)
## Variables actually used in tree construction:
## [1] "LoyalCH"       "PriceDiff"     "ListPriceDiff" "PctDiscMM"
## Number of terminal nodes:  8
## Residual mean deviance:  0.7659 = 606.6 / 792
## Misclassification error rate: 0.1675 = 134 / 800
```

The variables actually used in tree construction are LoyalCH, PriceDiff, ListPriceDiff, PctDiscMM. The training error rate is 0.1675. The tree has 8 terminal nodes.

(c)

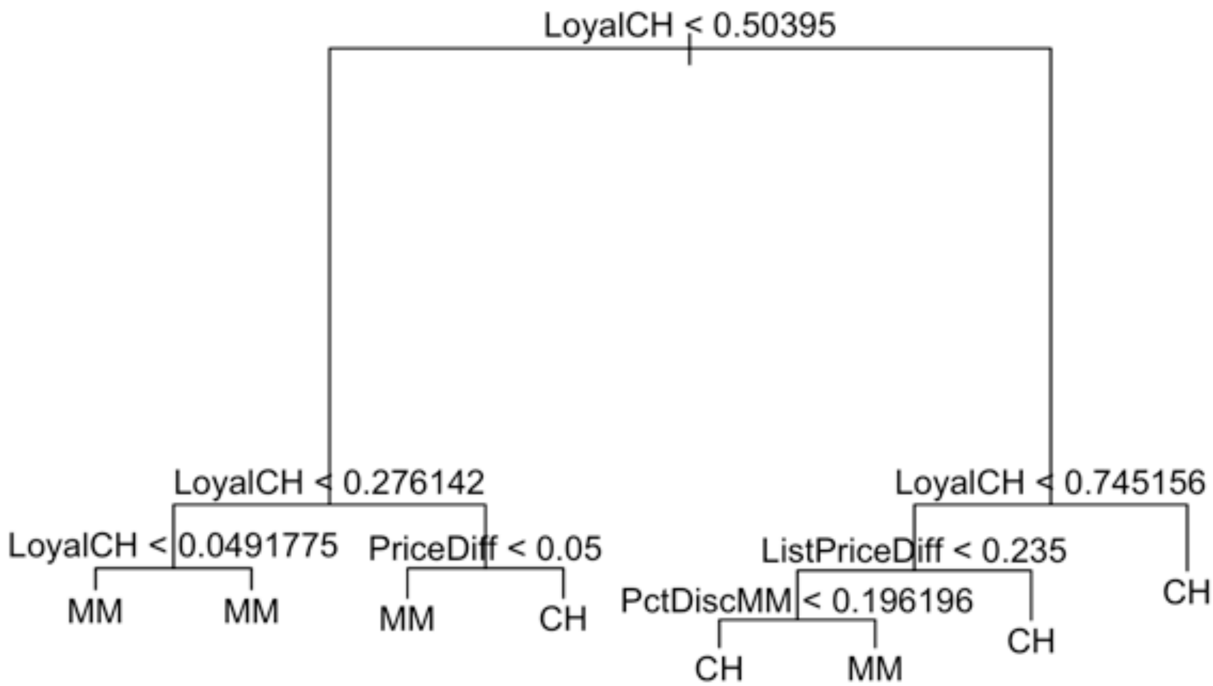
```
tree
```

```
## node), split, n, deviance, yval, (yprob)
##       * denotes terminal node
##
##  1) root 800 1077.00 CH ( 0.60000 0.40000 )
##    2) LoyalCH < 0.50395 360  425.40 MM ( 0.27778 0.72222 )
##      4) LoyalCH < 0.276142 176  132.60 MM ( 0.12500 0.87500 )
##        8) LoyalCH < 0.0491775 63   10.27 MM ( 0.01587 0.98413 ) *
##        9) LoyalCH > 0.0491775 113  108.50 MM ( 0.18584 0.81416 ) *
##      5) LoyalCH > 0.276142 184  250.80 MM ( 0.42391 0.57609 )
##       10) PriceDiff < 0.05 71   75.77 MM ( 0.22535 0.77465 ) *
##       11) PriceDiff > 0.05 113  155.60 CH ( 0.54867 0.45133 ) *
##    3) LoyalCH > 0.50395 440  350.50 CH ( 0.86364 0.13636 )
##      6) LoyalCH < 0.745156 182  210.00 CH ( 0.73626 0.26374 )
##       12) ListPriceDiff < 0.235 70   97.04 CH ( 0.50000 0.50000 )
##         24) PctDiscMM < 0.196196 51   66.22 CH ( 0.64706 0.35294 ) *
##         25) PctDiscMM > 0.196196 19   12.79 MM ( 0.10526 0.89474 ) *
##       13) ListPriceDiff > 0.235 112   80.42 CH ( 0.88393 0.11607 ) *
##      7) LoyalCH > 0.745156 258   97.07 CH ( 0.95349 0.04651 ) *
```

If we look at node 8, which is a terminal node. It means that there are 63 observations that fall into the branch where LoyalCH < 0.049 (this is the split standard) with a deviance of 10.27; the prediction is MM; and 1.6% of the observations are CH and 98.4% are MM. This node means that if a customer scores a LoyalCH < 0.049, then he/she is expected to purhcase Minute Maid.

(d)

```
plot(tree)
text(tree,pretty=0)
```



We can use this plot to predict what type of juice a customer will buy given the information of this customer on LoyalCH, PriceDiff, ListPriceDiff, and PctDiscMM. We first look at if LoyalCH < 0.504. If yes, then we go to the left branch for next step; if no, then we go to the right branch for next step. We repeat such steps with different split standards, then eventually arrive at a terminal node which tells us if this customer will buy CH or MM juice. From the graph, we can conclude that the most important variable is LoyalCH because both the first layer and the second layer split on LoyalCH.

(e)

```
pred <- predict(tree,test,type="class")
table_temp <- table(test$Purchase, pred)
table_temp
```

```
##     pred
##       CH  MM
##   CH 161  12
##   MM  28  69
```

```
test_err_tree <- (table_temp[1,2] + table_temp[2,1]) / sum(table_temp)
cat("The test error rate is:", test_err_tree, "\n")
```

```
## The test error rate is: 0.1481481
```

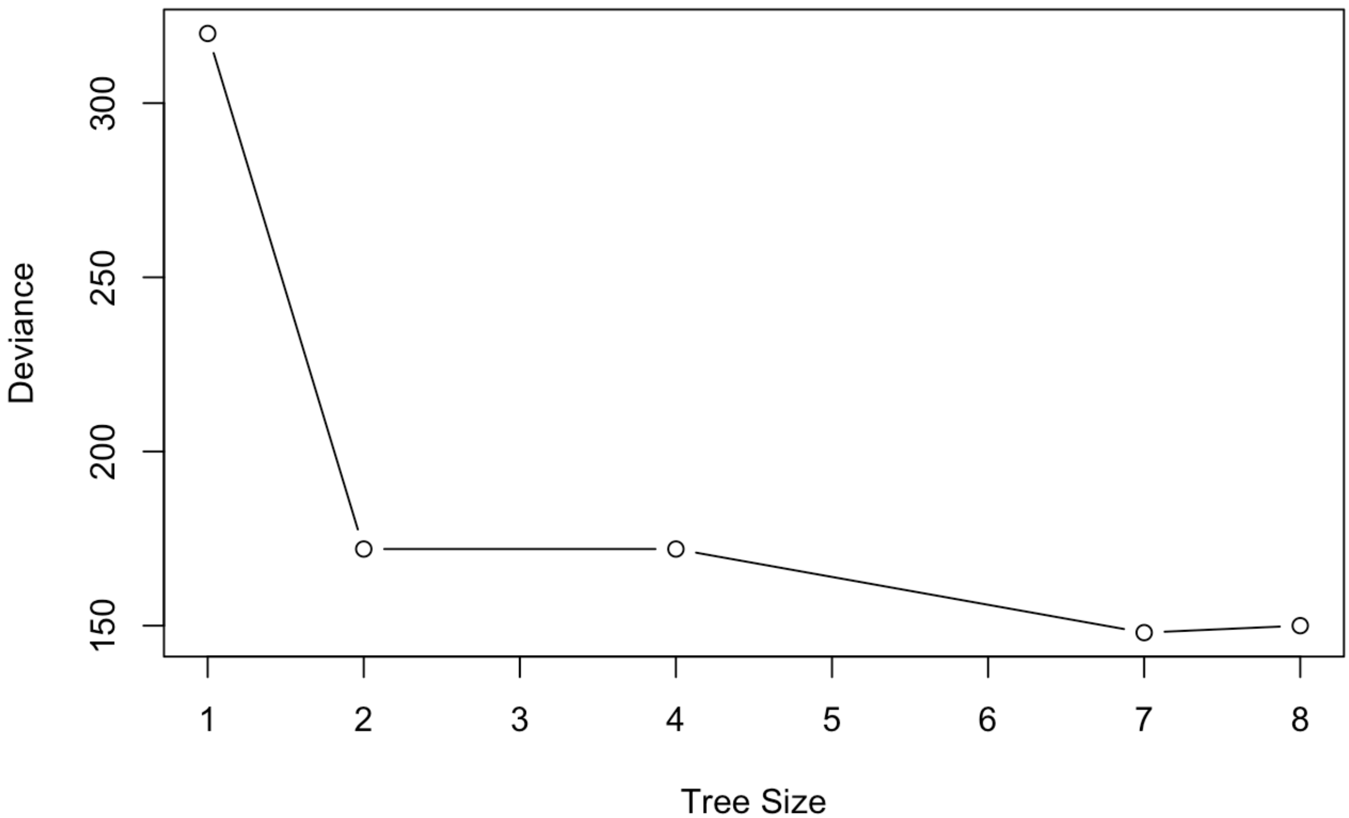
(f)

```
cv_oj=cv.tree(tree,FUN=prune.misclass)
cv_oj
```

```
## $size
## [1] 8 7 4 2 1
##
## $dev
## [1] 150 148 172 172 320
##
## $k
## [1]  -Inf   0.0   5.0   5.5 160.0
##
## $method
## [1] "misclass"
##
## attr(,"class")
## [1] "prune"         "tree.sequence"
```

(g)

```
plot(cv_oj$size ,cv_oj$dev ,type="b", xlab = "Tree Size", ylab = "Deviance")
```




```
cat("\n")
```

(h)

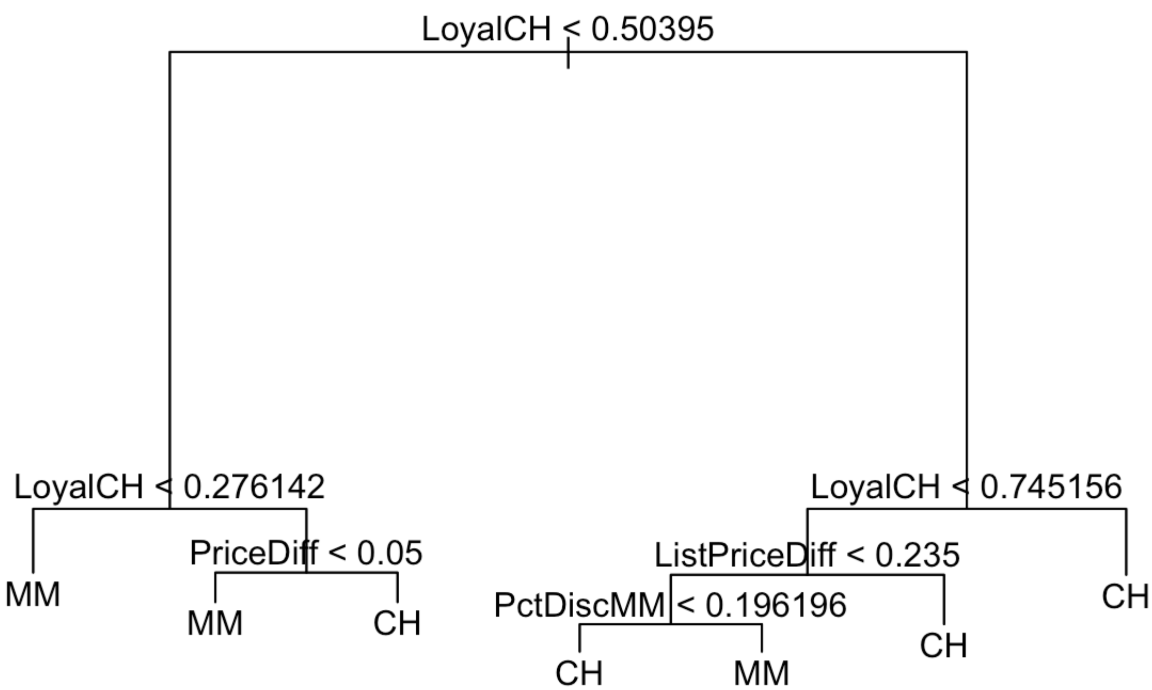
```
opt_size <- cv_oj$size[which.min(cv_oj$dev)]
opt_size
```

```
## [1] 7
```

Size 7 corresponds to the lowest cross-validated classification error rate.

(i)

```
pruned_tree <- prune.misclass(tree, best = opt_size)
plot(pruned_tree)
text(pruned_tree, pretty = 0)
```




```
cat("\n")
```

(j)

```
summary(pruned_tree)
```

```
##
## Classification tree:
## snip.tree(tree = tree, nodes = 4L)
## Variables actually used in tree construction:
## [1] "LoyalCH"       "PriceDiff"     "ListPriceDiff" "PctDiscMM"
## Number of terminal nodes:  7
## Residual mean deviance:  0.7824 = 620.5 / 793
## Misclassification error rate: 0.1675 = 134 / 800
```

The pruned tree’s training error rate is 0.1675, which is the same as the un-pruned tree’s training error rate.

(k)

```
pruned_pred <- predict(pruned_tree, test, type = "class")
table_temp <- table(test$Purchase, pruned_pred)
table_temp
```

```
##     pruned_pred
##       CH  MM
##   CH 161  12
##   MM  28  69
```

```
test_err_pruned <- (table_temp[1,2] + table_temp[2,1]) / sum(table_temp)
cat("The test error rate is:", test_err_pruned, "\n")
```

```
## The test error rate is: 0.1481481
```

The pruned tree’s testing error rate is 0.1481, which is the same as the un-pruned tree’s testing error rate.

Chapter 9 Exercise 8

(a) Same training and testing set as the first exercise

(b)

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

```
##
## Call:
## svm(formula = Purchase ~ ., data = train, kernel = "linear",
##     cost = 0.01)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  0.01
##       gamma:  0.05555556
##
## Number of Support Vectors:  440
##
##  ( 220 220 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

There are 440 support vectors obtained from 800 obersavations from the training set, in which 220 are CH and 220 are MM.

(c)

```
train_pred_svm_lin <- predict(svm_lin, train)
table_temp <- table(train$Purchase, train_pred_svm_lin)
table_temp
```

```
##     train_pred_svm_lin
##       CH  MM
##   CH 419  61
##   MM  83 237
```

```
train_err_svm_lin <- (table_temp[1,2] + table_temp[2,1]) / sum(table_temp)
cat("Linear kernel's training error rate is:", train_err_svm_lin, "\n")
```

```
## Linear kernel's training error rate is: 0.18
```

```
test_pred_svm_lin <- predict(svm_lin, test)
table_temp <- table(test$Purchase, test_pred_svm_lin)
table_temp
```

```
##     test_pred_svm_lin
##       CH  MM
##   CH 156  17
##   MM  21  76
```

```
test_err_svm_lin <- (table_temp[1,2] + table_temp[2,1]) / sum(table_temp)
cat("Linear kernel's test error rate is:", test_err_svm_lin, "\n")
```

```
## Linear kernel's test error rate is: 0.1407407
```

(d)

```
tune_svm_lin <- tune(svm, Purchase ~ ., data = train, kernel = "linear", ranges = lis
t(cost = 10^seq(-2, 1, by = 0.25)))
summary(tune_svm_lin)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##     1
##
## - best performance: 0.17625
##
## - Detailed performance results:
##           cost   error dispersion
## 1   0.01000000 0.18625 0.03251602
## 2   0.01778279 0.18000 0.02648375
## 3   0.03162278 0.18250 0.03073181
## 4   0.05623413 0.18125 0.03294039
## 5   0.10000000 0.18125 0.03448530
## 6   0.17782794 0.17750 0.03809710
## 7   0.31622777 0.17875 0.03910900
## 8   0.56234133 0.17875 0.03910900
## 9   1.00000000 0.17625 0.03928617
## 10  1.77827941 0.17750 0.03809710
## 11  3.16227766 0.17625 0.03701070
## 12  5.62341325 0.18250 0.03238227
## 13 10.00000000 0.18250 0.03446012
```

```
opt_cost_svm_lin <- tune_svm_lin$best.parameters$cost
cat("The optimal cost is:", opt_cost_svm_lin)
```

```
## The optimal cost is: 1
```

(e)

```
svm_lin_opt <- svm(Purchase ~ ., data = train, kernel = "linear", cost = opt_cost_svm
_lin)

train_pred_svm_lin_opt <- predict(svm_lin_opt, train)
table_temp <- table(train$Purchase, train_pred_svm_lin_opt)
table_temp
```

```
##     train_pred_svm_lin_opt
##       CH  MM
##   CH 422  58
##   MM  78 242
```

```
train_err_svm_lin_opt <- (table_temp[1,2] + table_temp[2,1]) / sum(table_temp)
cat("Linear kernel's new training error rate is:", train_err_svm_lin_opt, "\n")
```

```
## Linear kernel's new training error rate is: 0.17
```

```
test_pred_svm_lin_opt <- predict(svm_lin_opt, test)
table_temp <- table(test$Purchase, test_pred_svm_lin_opt)
table_temp
```

```
##     test_pred_svm_lin_opt
##       CH  MM
##   CH 154  19
##   MM  18  79
```

```
test_err_svm_lin_opt <- (table_temp[1,2] + table_temp[2,1]) / sum(table_temp)
cat("Linear kernel's new test error rate is:", test_err_svm_lin_opt, "\n")
```

```
## Linear kernel's new test error rate is: 0.137037
```

For linear kernel, tuning indeed reduced both training error rate and testing error rate.

(f) - (b) part

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

```
##
## Call:
## svm(formula = Purchase ~ ., data = train, kernel = "radial")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  1
##       gamma:  0.05555556
##
## Number of Support Vectors:  372
##
##  ( 189 183 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

For radial kernel, there are 372 support vectors obtained from 800 obersavations from the training set, in which 189 are CH and 183 are MM.

(f) - (c) part

```
train_pred_svm_rad <- predict(svm_rad, train)
table_temp <- table(train$Purchase, train_pred_svm_rad)
table_temp
```

```
##     train_pred_svm_rad
##       CH  MM
##   CH 440  40
##   MM  84 236
```

```
train_err_svm_rad <- (table_temp[1,2] + table_temp[2,1]) / sum(table_temp)
cat("Radial kernel's training error rate is:", train_err_svm_rad, "\n")
```

```
## Radial kernel's training error rate is: 0.155
```

```
test_pred_svm_rad <- predict(svm_rad, test)
table_temp <- table(test$Purchase, test_pred_svm_rad)
table_temp
```

```
##     test_pred_svm_rad
##       CH  MM
##   CH 153  20
##   MM  26  71
```

```
test_err_svm_rad <- (table_temp[1,2] + table_temp[2,1]) / sum(table_temp)
cat("Radial kernel's test error rate is:", test_err_svm_rad, "\n")
```

```
## Radial kernel's test error rate is: 0.1703704
```

(f) - (d) part

```
tune_svm_rad <- tune(svm, Purchase ~ ., data = train, kernel = "radial", ranges = lis
t(cost = 10^seq(-2, 1, by = 0.25)))
summary(tune_svm_rad)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##     1
##
## - best performance: 0.17375
##
## - Detailed performance results:
##           cost   error dispersion
## 1   0.01000000 0.40000 0.03818813
## 2   0.01778279 0.40000 0.03818813
## 3   0.03162278 0.31125 0.06022239
## 4   0.05623413 0.20000 0.04487637
## 5   0.10000000 0.18750 0.04330127
## 6   0.17782794 0.18875 0.04226652
## 7   0.31622777 0.18375 0.04752558
## 8   0.56234133 0.17875 0.05104804
## 9   1.00000000 0.17375 0.04466309
## 10  1.77827941 0.17625 0.04803428
## 11  3.16227766 0.17750 0.04322101
## 12  5.62341325 0.17750 0.03809710
## 13 10.00000000 0.18625 0.03972562
```

```
opt_cost_svm_rad <- tune_svm_rad$best.parameters$cost
cat("The optimal cost is:", opt_cost_svm_rad)
```

```
## The optimal cost is: 1
```

(f) - (e) part

```
svm_rad_opt <- svm(Purchase ~ ., data = train, kernel = "radial", cost = opt_cost_svm
_rad)

train_pred_svm_rad_opt <- predict(svm_rad_opt, train)
table_temp <- table(train$Purchase, train_pred_svm_rad_opt)
table_temp
```

```
##     train_pred_svm_rad_opt
##       CH  MM
##   CH 440  40
##   MM  84 236
```

```
train_err_svm_rad_opt <- (table_temp[1,2] + table_temp[2,1]) / sum(table_temp)
cat("Radial kernel's new training error rate is:", train_err_svm_rad_opt, "\n")
```

```
## Radial kernel's new training error rate is: 0.155
```

```
test_pred_svm_rad_opt <- predict(svm_rad_opt, test)
table_temp <- table(test$Purchase, test_pred_svm_rad_opt)
table_temp
```

```
##     test_pred_svm_rad_opt
##       CH  MM
##   CH 153  20
##   MM  26  71
```

```
test_err_svm_rad_opt <- (table_temp[1,2] + table_temp[2,1]) / sum(table_temp)
cat("Radial kernel's new test error rate is:", test_err_svm_rad_opt, "\n")
```

```
## Radial kernel's new test error rate is: 0.1703704
```

For radial kernel, tuning didn’t really change training error rate or testing error rate.

(g) - (b) part

```
svm_poly <- svm(Purchase ~ ., data = train, kernel = "polynomial", degree = 2)
summary(svm_poly)
```

```
##
## Call:
## svm(formula = Purchase ~ ., data = train, kernel = "polynomial",
##     degree = 2)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  polynomial
##        cost:  1
##      degree:  2
##       gamma:  0.05555556
##      coef.0:  0
##
## Number of Support Vectors:  448
##
##  ( 226 222 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

For polynomial kernel, there are 448 support vectors obtained from 800 obersavations from the training set, in which 226 are CH and 222 are MM.

(g) - (c) part

```
train_pred_svm_poly <- predict(svm_poly, train)
table_temp <- table(train$Purchase, train_pred_svm_poly)
table_temp
```

```
##     train_pred_svm_poly
##       CH  MM
##   CH 449  31
##   MM 113 207
```

```
train_err_svm_poly <- (table_temp[1,2] + table_temp[2,1]) / sum(table_temp)
cat("Polynomial kernel's training error rate is:", train_err_svm_poly, "\n")
```

```
## Polynomial kernel's training error rate is: 0.18
```

```
test_pred_svm_poly <- predict(svm_poly, test)
table_temp <- table(test$Purchase, test_pred_svm_poly)
table_temp
```

```
##     test_pred_svm_poly
##       CH  MM
##   CH 159  14
##   MM  46  51
```

```
test_err_svm_poly <- (table_temp[1,2] + table_temp[2,1]) / sum(table_temp)
cat("Polynomial kernel's test error rate is:", test_err_svm_poly, "\n")
```

```
## Polynomial kernel's test error rate is: 0.2222222
```

(g) - (d) part

```
tune_svm_poly <- tune(svm, Purchase ~ ., data = train, kernel = "polynomial", degree
= 2, ranges = list(cost = 10^seq(-2, 1, by = 0.25)))
summary(tune_svm_poly)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##    10
##
## - best performance: 0.18125
##
## - Detailed performance results:
##           cost   error dispersion
## 1   0.01000000 0.39000 0.04743416
## 2   0.01778279 0.37500 0.04930066
## 3   0.03162278 0.34875 0.06493854
## 4   0.05623413 0.32500 0.05400617
## 5   0.10000000 0.31375 0.04387878
## 6   0.17782794 0.24000 0.03944053
## 7   0.31622777 0.20375 0.03910900
## 8   0.56234133 0.20125 0.03606033
## 9   1.00000000 0.20000 0.03726780
## 10  1.77827941 0.19500 0.03073181
## 11  3.16227766 0.19125 0.03175973
## 12  5.62341325 0.18375 0.03335936
## 13 10.00000000 0.18125 0.03498512
```

```
opt_cost_svm_poly <- tune_svm_poly$best.parameters$cost
cat("The optimal cost is:", opt_cost_svm_poly)
```

```
## The optimal cost is: 10
```

(g) - (e) part

```
svm_poly_opt <- svm(Purchase ~ ., data = train, kernel = "polynomial", degree = 2, co
st = opt_cost_svm_poly)

train_pred_svm_poly_opt <- predict(svm_poly_opt, train)
table_temp <- table(train$Purchase, train_pred_svm_poly_opt)
table_temp
```

```
##     train_pred_svm_poly_opt
##       CH  MM
##   CH 444  36
##   MM  85 235
```

```
train_err_svm_poly_opt <- (table_temp[1,2] + table_temp[2,1]) / sum(table_temp)
cat("Polynomial kernel's new training error rate is:", train_err_svm_poly_opt, "\n")
```

```
## Polynomial kernel's new training error rate is: 0.15125
```

```
test_pred_svm_poly_opt <- predict(svm_poly_opt, test)
table_temp <- table(test$Purchase, test_pred_svm_poly_opt)
table_temp
```

```
##     test_pred_svm_poly_opt
##       CH  MM
##   CH 158  15
##   MM  28  69
```

```
test_err_svm_poly_opt <- (table_temp[1,2] + table_temp[2,1]) / sum(table_temp)
cat("Polynomial kernel's new test error rate is:", test_err_svm_poly_opt, "\n")
```

```
## Polynomial kernel's new test error rate is: 0.1592593
```

For polynomial kernel, tuning reduced both training error rate and testing error rate.

RF - (b) part

```
require(randomForest)
```

```
## Loading required package: randomForest
```

```
## randomForest 4.6-14
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

```
##
## Attaching package: 'randomForest'
```

```
## The following object is masked from 'package:dplyr':
##
##     combine
```

```
## The following object is masked from 'package:ggplot2':
##
##     margin
```

```
forest <- randomForest( Purchase ~ ., data = train, ntree = 100, nodesize = 20)
forest
```

```
##
## Call:
##  randomForest(formula = Purchase ~ ., data = train, ntree = 100,      nodesize = 2
0)
##                Type of random forest: classification
##                      Number of trees: 100
## No. of variables tried at each split: 4
##
##         OOB estimate of  error rate: 17.88%
## Confusion matrix:
##     CH  MM class.error
## CH 417  63     0.13125
## MM  80 240     0.25000
```

RF - (c) part

```
forest$confusion
```

```
##     CH  MM class.error
## CH 417  63     0.13125
## MM  80 240     0.25000
```

```
train_err_rf <- tail(forest$err.rate[, 1], n=1)
cat("RF's training error rate is:", train_err_rf, "\n")
```

```
## RF's training error rate is: 0.17875
```

```
pred_rf <- predict(forest, test)
table_temp <- table(test$Purchase, pred_rf)
table_temp
```

```
##     pred_rf
##       CH  MM
##   CH 148  25
##   MM  25  72
```

```
test_err_rf <- (table_temp[1,2] + table_temp[2,1]) / sum(table_temp)
cat("RF's test error rate is:", test_err_rf, "\n")
```

```
## RF's test error rate is: 0.1851852
```

RF - (d) part

```
tune_forest <- tune(randomForest, train.x = Purchase ~ ., data = train, validation.x
= test)
tune_forest
```

```
##
## Error estimation of 'randomForest' using 10-fold cross validation: 0.2075
```

RF - (e) part

```
forest_tuned <- tune_forest$best.model

forest_tuned$confusion
```

```
##     CH  MM class.error
## CH 402  78    0.162500
## MM  83 237    0.259375
```

```
train_err_forest_tuned <- tail(forest_tuned$err.rate[, 1], n=1)
cat("RF's new training error rate is:", train_err_forest_tuned, "\n")
```

```
## RF's new training error rate is: 0.20125
```

```
pred_forest_tuned <- predict(forest_tuned, test)
table_temp <- table(test$Purchase, pred_forest_tuned)
table_temp
```

```
##     pred_forest_tuned
##       CH  MM
##   CH 148  25
##   MM  28  69
```

```
test_err_forest_tuned <- (table_temp[1,2] + table_temp[2,1]) / sum(table_temp)
cat("RF's new test error rate is:", test_err_forest_tuned, "\n")
```

```
## RF's new test error rate is: 0.1962963
```

For random forest, tuning actually increased both training error rate and testing error rate.

(h)

Overall, all the models’ error rate on both training and testing data are similar, but the SVM with radial kernel is slightly better.