HW10 solutions

Homework 10:

Chapter 9, Exercise 8

```
library(ISLR)
set.seed(9004)
train = sample(dim(OJ)[1], 800)
OJ.train = OJ[train, ]
OJ.test = OJ[-train, ]
```

a Create a training set containing a random sample of 800 observations, and a test set containing the remaining observations.

```
library(e1071)
svm.linear = svm(Purchase~., kernel="linear", data=OJ.train, cost=0.01)
summary(svm.linear)
```

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.

```
##
## 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:  442
##
##  ( 222 220 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

Support vector classifier creates 432 support vectors out of 800 training points. Out of these, 217 belong to level `CH` and remaining 215 belong to level `MM`.

```r
train.pred = predict(svm.linear, OJ.train)
table(OJ.train$Purchase, train.pred)
```

c What are the training and test error rates?

```
##     train.pred
##       CH  MM
##   CH 432  51
##   MM  80 237
```

```r
(82 + 53) / (439 + 53 + 82 + 226)
```

```
## [1] 0.16875
```

```r
test.pred = predict(svm.linear, OJ.test)
table(OJ.test$Purchase, test.pred)
```

```
##     test.pred
##       CH  MM
##   CH 146  24
##   MM  22  78
```

```r
(19 + 29) / (142 + 19 + 29 + 80)
```

```
## [1] 0.1777778
```

The training error rate is 16.9% and test error rate is about 17.8%.

```r
set.seed(1554)
tune.out = tune(svm, Purchase~., data=OJ.train, kernel="linear",
                ranges=list(cost=10^seq(-2, 1, by=0.25)))
summary(tune.out)
```

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

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##      cost
##  3.162278
##
## - best performance: 0.1625
##
## - Detailed performance results:
##           cost   error dispersion
## 1   0.01000000 0.16750 0.03395258
## 2   0.01778279 0.16875 0.02960973
## 3   0.03162278 0.16625 0.02638523
## 4   0.05623413 0.16875 0.03076005
## 5   0.10000000 0.16875 0.02901748
## 6   0.17782794 0.16750 0.02838231
## 7   0.31622777 0.17000 0.02898755
## 8   0.56234133 0.16875 0.02841288
## 9   1.00000000 0.16500 0.03106892
```

```
## 10  1.77827941 0.16500 0.03106892
## 11  3.16227766 0.16250 0.03118048
## 12  5.62341325 0.16375 0.02664713
## 13 10.00000000 0.16750 0.02581989
```

Tuning shows that optimal cost is 0.3162

```r
svm.linear = svm(Purchase~., kernel="linear", data=OJ.train,
                 cost=tune.out$best.parameters$cost)
train.pred = predict(svm.linear, OJ.train)
table(OJ.train$Purchase, train.pred)
```

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

```
##     train.pred
##       CH  MM
##   CH 428  55
##   MM  74 243
```

```r
(57 + 71) / (435 + 57 + 71 + 237)
```

```
## [1] 0.16
```

```r
test.pred = predict(svm.linear, OJ.test)
table(OJ.test$Purchase, test.pred)
```

```
##     test.pred
##       CH  MM
##   CH 146  24
##   MM  20  80
```

```r
(29 + 20) / (141 + 20 + 29 + 80)
```

```
## [1] 0.1814815
```

The training error decreases to 16% but test error slightly increases to 18.1% by using best cost.

```r
set.seed(410)
svm.radial = svm(Purchase~., data=OJ.train, kernel="radial")
summary(svm.radial)
```

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

```
##
## Call:
## svm(formula = Purchase ~ ., data = OJ.train, kernel = "radial")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  1
##
## Number of Support Vectors:  371
##
##  ( 188 183 )
```

```
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

```
train.pred = predict(svm.radial, OJ.train)
table(OJ.train$Purchase, train.pred)
```

```
##     train.pred
##       CH  MM
##   CH 441  42
##   MM  74 243
```

```
(40 + 78) / (452 + 40 + 78 + 230)
```

```
## [1] 0.1475
```

```
test.pred = predict(svm.radial, OJ.test)
table(OJ.test$Purchase, test.pred)
```

```
##     test.pred
##       CH  MM
##   CH 148  22
##   MM  27  73
```

```
(27 + 15) / (146 + 15 + 27 + 82)
```

```
## [1] 0.1555556
```

The radial basis kernel with default gamma creates 367 support vectors, out of which, 184 belong to level `CH` and remaining 183 belong to level `MM`. The classifier has a training error of 14.7% and a test error of 15.6% which is a slight improvement over linear kernel. We now use cross validation to find optimal gamma.

```
set.seed(755)
tune.out = tune(svm, Purchase~., data=OJ.train, kernel="radial",
                ranges=list(cost=10^seq(-2, 1, by=0.25)))
summary(tune.out)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##       cost
##  0.3162278
##
## - best performance: 0.1675
##
## - Detailed performance results:
##           cost   error dispersion
## 1   0.01000000 0.39625 0.06615691
## 2   0.01778279 0.39625 0.06615691
## 3   0.03162278 0.35375 0.09754807
## 4   0.05623413 0.20000 0.04249183
## 5   0.10000000 0.17750 0.04073969
```

```
## 6   0.17782794 0.17125 0.03120831
## 7   0.31622777 0.16750 0.04216370
## 8   0.56234133 0.16750 0.03782269
## 9   1.00000000 0.17250 0.03670453
## 10  1.77827941 0.17750 0.03374743
## 11  3.16227766 0.18000 0.04005205
## 12  5.62341325 0.18000 0.03446012
## 13 10.00000000 0.18625 0.04427267
```

```
svm.radial = svm(Purchase~., data=OJ.train, kernel="radial",
                 cost=tune.out$best.parameters$cost)
train.pred = predict(svm.radial, OJ.train)
table(OJ.train$Purchase, train.pred)
```

```
##     train.pred
##       CH  MM
##   CH 440  43
##   MM  81 236
```

```
(77 + 40) / (452 + 40 + 77 + 231)
```

```
## [1] 0.14625
```

```
test.pred = predict(svm.radial, OJ.test)
table(OJ.test$Purchase, test.pred)
```

```
##     test.pred
##       CH  MM
##   CH 145  25
##   MM  28  72
```

```
(28 + 15) / (146 + 15 + 28 + 81)
```

```
## [1] 0.1592593
```

Tuning slightly decreases training error to 14.6% and slightly increases test error to 16% which is still better than linear kernel.

```
set.seed(8112)
svm.poly = svm(Purchase~., data=OJ.train, kernel="poly", degree=2)
summary(svm.poly)
```

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

```
##
## 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:  456
##
##  ( 232 224 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

```r
train.pred = predict(svm.poly, OJ.train)
table(OJ.train$Purchase, train.pred)
```

```
##     train.pred
##       CH  MM
##   CH 450  33
##   MM 111 206
```

```r
(32 + 105) / (460 + 32 + 105 + 203)
```

```
## [1] 0.17125
```

```r
test.pred = predict(svm.poly, OJ.test)
table(OJ.test$Purchase, test.pred)
```

```
##     test.pred
##       CH  MM
##   CH 149  21
##   MM  34  66
```

```r
(12 + 37) / (149 + 12 + 37 + 72)
```

```
## [1] 0.1814815
```

Summary shows that polynomial kernel produces 452 support vectors, out of which, 232 belong to level `CH` and remaining 220 belong to level `MM`. This kernel produces a train error of 17.1% and a test error of 18.1% which are slightly higher than the errors produces by radial kernel but lower than the errors produced by linear kernel.

```r
set.seed(322)
tune.out = tune(svm, Purchase~., data=OJ.train, kernel="poly",
                degree=2, ranges=list(cost=10^seq(-2, 1, by=0.25)))
summary(tune.out)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##    10
##
## - best performance: 0.18
##
## - Detailed performance results:
##           cost   error dispersion
## 1   0.01000000 0.39250 0.05749396
```

```
## 2   0.01778279 0.37500 0.05863020
## 3   0.03162278 0.36375 0.05756940
## 4   0.05623413 0.33875 0.06626179
## 5   0.10000000 0.30375 0.05172376
## 6   0.17782794 0.24000 0.04440971
## 7   0.31622777 0.21000 0.04362084
## 8   0.56234133 0.20250 0.03987829
## 9   1.00000000 0.20375 0.03634805
## 10  1.77827941 0.19500 0.04866267
## 11  3.16227766 0.18750 0.04409586
## 12  5.62341325 0.18875 0.04185375
## 13 10.00000000 0.18000 0.03593976
```

```r
svm.poly = svm(Purchase~., data=OJ.train, kernel="poly",
               degree=2, cost=tune.out$best.parameters$cost)
train.pred = predict(svm.poly, OJ.train)
table(OJ.train$Purchase, train.pred)
```

```
##     train.pred
##       CH  MM
##   CH 447  36
##   MM  85 232
```

```r
(37 + 84) / (455 + 37 + 84 + 224)
```

```
## [1] 0.15125
```

```r
test.pred = predict(svm.poly, OJ.test)
table(OJ.test$Purchase, test.pred)
```

```
##     test.pred
##       CH  MM
##   CH 148  22
##   MM  28  72
```

```r
(13 + 34) / (148 + 13 + 34 + 75)
```

```
## [1] 0.1740741
```

Tuning reduces the training error to 15.12% and test error to 17.4% which is worse than radial kernel but slightly better than linear kernel.

h Overall, which approach seems to give the best results on this data? Overall, radial basis kernel seems to be producing minimum misclassification error on both train and test data.