Chapter 9: Exercise 8

a

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

b

```
library(e1071)
```

```
## Warning: package 'e1071' was built under R version 3.0.2
```

```
## Loading required package: class
```

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

```
##
## Call:
## svm(formula = Purchase ~ ., data = OJ.train, kernel = "linear",
##     cost = 0.01)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  0.01
##       gamma:  0.05556
##
## Number of Support Vectors:  432
##
##  ( 217 215 )
##
##
## 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`.

c

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

```
##     train.pred
##       CH  MM
##   CH 439  53
##   MM  82 226
```

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

```
## [1] 0.1688
```

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

```
##     test.pred
##       CH  MM
##   CH 142  19
##   MM  29  80
```

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

```
## [1] 0.1778
```

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

d

```
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)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##    cost
##  0.3162
##
## - best performance: 0.1687
##
## - Detailed performance results:
##        cost  error dispersion
## 1   0.01000 0.1688    0.03692
## 2   0.01778 0.1688    0.03398
## 3   0.03162 0.1713    0.03230
## 4   0.05623 0.1725    0.03162
## 5   0.10000 0.1700    0.03291
## 6   0.17783 0.1712    0.03336
## 7   0.31623 0.1687    0.03499
## 8   0.56234 0.1700    0.03129
## 9   1.00000 0.1687    0.03398
## 10  1.77828 0.1688    0.03241
## 11  3.16228 0.1688    0.03294
## 12  5.62341 0.1713    0.03121
## 13 10.00000 0.1713    0.03283
```

Tuning shows that optimal cost is 0.3162

e

```
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)
```

```
##     train.pred
##       CH  MM
##   CH 435  57
##   MM  71 237
```

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

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

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

```
##     test.pred
##       CH  MM
##   CH 141  20
##   MM  29  80
```

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

```
## [1] 0.1815
```

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

f

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

```
##
## Call:
## svm(formula = Purchase ~ ., data = OJ.train, kernel = "radial")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  1
##       gamma:  0.05556
##
## Number of Support Vectors:  367
##
##  ( 184 183 )
##
##
## 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 452  40
##   MM  78 230
```

```
(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 146  15
##   MM  27  82
```

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

```
## [1] 0.1556
```

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
##  0.5623
##
## - best performance: 0.165
##
## - Detailed performance results:
##        cost  error dispersion
## 1   0.01000 0.3850    0.06258
## 2   0.01778 0.3850    0.06258
## 3   0.03162 0.3762    0.06908
## 4   0.05623 0.2100    0.03855
## 5   0.10000 0.1862    0.03143
## 6   0.17783 0.1837    0.03230
## 7   0.31623 0.1712    0.03438
## 8   0.56234 0.1650    0.03764
## 9   1.00000 0.1750    0.03584
## 10  1.77828 0.1738    0.04059
## 11  3.16228 0.1763    0.03748
## 12  5.62341 0.1763    0.03839
## 13 10.00000 0.1738    0.03459
```

```
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 452  40
##   MM  77 231
```

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

```
## [1] 0.1462
```

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

```
##     test.pred
##       CH  MM
##   CH 146  15
##   MM  28  81
```

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

```
## [1] 0.1593
```

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

g

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

```
##
## Call:
## svm(formula = Purchase ~ ., data = OJ.train, kernel = "poly",
##     degree = 2)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  polynomial
##        cost:  1
##      degree:  2
##       gamma:  0.05556
##      coef.0:  0
##
## Number of Support Vectors:  452
##
##  ( 232 220 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

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

```
##     train.pred
##       CH  MM
##   CH 460  32
##   MM 105 203
```

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

```
## [1] 0.1713
```

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

```
##     test.pred
##       CH  MM
##   CH 149  12
##   MM  37  72
```

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

```
## [1] 0.1815
```

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.

```
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
##  5.623
##
## - best performance: 0.1837
##
## - Detailed performance results:
##        cost  error dispersion
## 1   0.01000 0.3850    0.05426
## 2   0.01778 0.3675    0.05076
## 3   0.03162 0.3575    0.05177
## 4   0.05623 0.3425    0.04937
## 5   0.10000 0.3150    0.05231
## 6   0.17783 0.2487    0.03929
## 7   0.31623 0.2088    0.05684
## 8   0.56234 0.2088    0.05653
## 9   1.00000 0.2000    0.06095
## 10  1.77828 0.1938    0.04497
## 11  3.16228 0.1862    0.04185
## 12  5.62341 0.1837    0.03336
## 13 10.00000 0.1837    0.04042
```

```
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 455  37
##   MM  84 224
```

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

```
## [1] 0.1512
```

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

```
##     test.pred
##       CH  MM
##   CH 148  13
##   MM  34  75
```

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

```
## [1] 0.1741
```

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, radial basis kernel seems to be producing minimum misclassification error on both train and test data.