Abhishek 6

```
#In this problem, you will use support vector approaches to predict whether a
given car gets high or low gas mileage based on the Auto data set in the ISLR
package.

#(a) Create a binary variable that takes on a 1 for cars with gas mileage abo
ve the median, and a 0 for cars with gas mileage below the median. Use this v
ariable as response in the following analysis.

library(ISLR)
var <- ifelse(Auto$mpg > median(Auto$mpg), 1, 0)
Auto$mpglevel <- as.factor(var)



# (b) Fit a support vector classifier to the data with various values of cost
, to predict whether a car gets high or low gas mileage. Report the cross-val
idation errors associated with different values of this parameter. Comment on
your results.

set.seed(60)
library(e1071)
svm_linear <- tune(svm, mpglevel ~ ., data = Auto, kernel = "linear", ranges
= list(cost = c(0.01, 0.1, 1, 5, 10, 100, 1000)))
summary(svm_linear)

##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##     1
##
## - best performance: 0.01282051
##
## - Detailed performance results:
##    cost      error dispersion
## 1 1e-02 0.07423077 0.04281275
## 2 1e-01 0.04865385 0.04268849
## 3 1e+00 0.01282051 0.01813094
## 4 5e+00 0.01532051 0.01318724
## 5 1e+01 0.02044872 0.01619554
## 6 1e+02 0.03839744 0.02495604
## 7 1e+03 0.03839744 0.02495604
```

```r
# A cost of 1 seems to perform best since it has the least cross validation e
rror.

# (c) Now repeat (b), this time using SVMs with radial and polynomial kernels
, with different values of gamma, degree and cost. Comment on your results.

set.seed(1)
svm_radial <- tune(svm, mpglevel ~ ., data = Auto, kernel = "radial", ranges
= list(cost = c(0.01, 0.1, 1, 5, 10, 100), gamma = c(0.01, 0.1, 1, 5, 10, 100
)))
summary(svm_radial)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost gamma
##   100  0.01
##
## - best performance: 0.01532051
##
## - Detailed performance results:
##     cost gamma      error dispersion
## 1  1e-02 1e-02 0.56115385 0.04344202
## 2  1e-01 1e-02 0.09185897 0.03862507
## 3  1e+00 1e-02 0.07147436 0.05103685
## 4  5e+00 1e-02 0.04326923 0.04975032
## 5  1e+01 1e-02 0.02551282 0.03812986
## 6  1e+02 1e-02 0.01532051 0.01788871
## 7  1e-02 1e-01 0.19153846 0.07612945
## 8  1e-01 1e-01 0.07916667 0.05201159
## 9  1e+00 1e-01 0.05608974 0.05092939
## 10 5e+00 1e-01 0.03064103 0.02637448
## 11 1e+01 1e-01 0.02551282 0.02076457
## 12 1e+02 1e-01 0.02807692 0.01458261
## 13 1e-02 1e+00 0.56115385 0.04344202
## 14 1e-01 1e+00 0.56115385 0.04344202
## 15 1e+00 1e+00 0.06634615 0.06187383
## 16 5e+00 1e+00 0.06128205 0.06186124
## 17 1e+01 1e+00 0.06128205 0.06186124
## 18 1e+02 1e+00 0.06128205 0.06186124
## 19 1e-02 5e+00 0.56115385 0.04344202
## 20 1e-01 5e+00 0.56115385 0.04344202
## 21 1e+00 5e+00 0.49224359 0.03806832
## 22 5e+00 5e+00 0.48967949 0.03738577
## 23 1e+01 5e+00 0.48967949 0.03738577
## 24 1e+02 5e+00 0.48967949 0.03738577
## 25 1e-02 1e+01 0.56115385 0.04344202
```

```
## 26 1e-01 1e+01 0.56115385 0.04344202
## 27 1e+00 1e+01 0.51775641 0.04471079
## 28 5e+00 1e+01 0.51012821 0.03817175
## 29 1e+01 1e+01 0.51012821 0.03817175
## 30 1e+02 1e+01 0.51012821 0.03817175
## 31 1e-02 1e+02 0.56115385 0.04344202
## 32 1e-01 1e+02 0.56115385 0.04344202
## 33 1e+00 1e+02 0.56115385 0.04344202
## 34 5e+00 1e+02 0.56115385 0.04344202
## 35 1e+01 1e+02 0.56115385 0.04344202
## 36 1e+02 1e+02 0.56115385 0.04344202
```

```
# For a radial kernel, the lowest cross-validation error is obtained for a ga
mma of 0.01 and a cost of 100.

set.seed(1)
svm_poly <- tune(svm, mpglevel ~ ., data = Auto, kernel = "polynomial", range
s = list(cost = c(0.01, 0.1, 1, 5, 10, 100), degree = c(2, 3, 4)))
summary(svm_poly)
```

```
## 
## Parameter tuning of 'svm':
## 
## - sampling method: 10-fold cross validation
## 
## - best parameters:
##  cost degree
##   100      2
## 
## - best performance: 0.3013462
## 
## - Detailed performance results:
##     cost degree     error dispersion
## 1  1e-02      2 0.5611538 0.04344202
## 2  1e-01      2 0.5611538 0.04344202
## 3  1e+00      2 0.5611538 0.04344202
## 4  5e+00      2 0.5611538 0.04344202
## 5  1e+01      2 0.5382051 0.05829238
## 6  1e+02      2 0.3013462 0.09040277
## 7  1e-02      3 0.5611538 0.04344202
## 8  1e-01      3 0.5611538 0.04344202
## 9  1e+00      3 0.5611538 0.04344202
## 10 5e+00      3 0.5611538 0.04344202
## 11 1e+01      3 0.5611538 0.04344202
## 12 1e+02      3 0.3322436 0.11140578
## 13 1e-02      4 0.5611538 0.04344202
## 14 1e-01      4 0.5611538 0.04344202
## 15 1e+00      4 0.5611538 0.04344202
## 16 5e+00      4 0.5611538 0.04344202
```

```
## 17 1e+01      4 0.5611538 0.04344202
## 18 1e+02      4 0.5611538 0.04344202

# For a polynomial kernel, the lowest cross-validation error is obtained for
a degree of 2 and a cost of 100.

# This problem uses the OJ data set in the ISLR package.
# (a) Create a training set containing a random sample of 800 observations, a
nd a test set containing the remaining observations.

set.seed(1)
train <- sample(nrow(OJ), 800)
OJ.train <- OJ[train, ]
OJ.test <- OJ[-train, ]

# (b) Fit a support vector classifier to the training data using cost=0.01, w
ith Purchase as the response and the other variables as predictors. Use the s
ummary() function to produce summary statistics, and describe the results obt
ained.

svm_linear <- svm(Purchase ~. , data = OJ.train, cost = 0.01, kernel = 'linea
r')
summary(svm_linear)

##
## Call:
## svm(formula = Purchase ~ ., data = OJ.train, cost = 0.01, kernel = "linear
")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  0.01
##       gamma:  0.05555556
##
## Number of Support Vectors:  432
##
##  ( 215 217 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM

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

train.pred <- predict(svm_linear, OJ.train)
table(OJ.train$Purchase, train.pred)
```

```
##     train.pred
##       CH  MM
##   CH 439  55
##   MM  78 228
```

```
test.pred <- predict(svm_linear, OJ.test)
table(OJ.test$Purchase, test.pred)
```

```
##     test.pred
##       CH  MM
##   CH 141  18
##   MM  31  80
```

```
# The training error rate is 16.6% and test error rate is about 18.1%.

# (d) Use the tune() function to select an optimal cost. Consider value in th
e range 0.01 to 10.

set.seed(2)
tune.out <- tune(svm, Purchase ~ ., data = OJ.train, kernel = "linear", range
s = 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
##   0.1
##
## - best performance: 0.1625
##
## - Detailed performance results:
##           cost   error dispersion
## 1   0.01000000 0.17125 0.05172376
## 2   0.01778279 0.16500 0.05197489
## 3   0.03162278 0.16625 0.04604120
## 4   0.05623413 0.16500 0.04594683
## 5   0.10000000 0.16250 0.04787136
## 6   0.17782794 0.16250 0.04249183
## 7   0.31622777 0.16875 0.04379958
## 8   0.56234133 0.16625 0.03998698
## 9   1.00000000 0.16500 0.03670453
## 10  1.77827941 0.16625 0.03682259
## 11  3.16227766 0.16500 0.03717451
## 12  5.62341325 0.16500 0.03525699
## 13 10.00000000 0.16750 0.03917553
```

```
# We may see that the optimal cost is 0.1.
```

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

svm_linear <- svm(Purchase ~ ., kernel = "linear", data = OJ.train, cost = tu
ne.out$best.parameter$cost)

train.pred <- predict(svm_linear, OJ.train)
table(OJ.train$Purchase, train.pred)

##     train.pred
##       CH  MM
##   CH 438  56
##   MM  71 235

test.pred <- predict(svm_linear, OJ.test)
table(OJ.test$Purchase, test.pred)

##     test.pred
##       CH  MM
##   CH 140  19
##   MM  32  79

# We may see that, with the best cost, the training error rate is now 15.8% a
nd the test error rate is 18.8%.

# (f) Repeat parts (b) through (e) using a support vector machine with a radi
al kernel. Use the tune() function to select an optimal cost and gamma.

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

##
## Call:
## svm(formula = Purchase ~ ., data = OJ.train, kernel = "radial")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  1
##       gamma:  0.05555556
##
## Number of Support Vectors:  379
##
##  ( 188 191 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

```r
train.pred <- predict(svm_radial, OJ.train)
table(OJ.train$Purchase, train.pred)
```

```
##     train.pred
##       CH  MM
##   CH 455  39
##   MM  77 229
```

```r
test.pred <- predict(svm_radial, OJ.test)
table(OJ.test$Purchase, test.pred)
```

```
##     test.pred
##       CH  MM
##   CH 141  18
##   MM  28  83
```

```r
# Radial kernel with default gamma creates 379 support vectors, out of which,
188 belong to level CH and remaining 191 belong to level MM. The classifier h
as a training error of 14.5% and a test error of 17% which is a slight improv
ement over linear kernel. We now use cross validation to find optimal cost.

set.seed(2)
tune.out <- tune(svm, Purchase ~ ., data = OJ.train, kernel = "radial", range
s = 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
##     1
##
## - best performance: 0.16625
##
## - Detailed performance results:
##          cost   error dispersion
## 1   0.01000000 0.38250 0.04533824
## 2   0.01778279 0.38250 0.04533824
## 3   0.03162278 0.37500 0.04894725
## 4   0.05623413 0.21500 0.05886661
## 5   0.10000000 0.17875 0.04860913
## 6   0.17782794 0.17875 0.05497790
## 7   0.31622777 0.17875 0.05981743
## 8   0.56234133 0.17250 0.05458174
## 9   1.00000000 0.16625 0.05001736
## 10  1.77827941 0.16875 0.05008673
## 11  3.16227766 0.17500 0.04787136
```

```
## 12  5.62341325 0.18000 0.05244044
## 13 10.00000000 0.18250 0.05596378
```

```
svm_radial <- svm(Purchase ~ ., kernel = "radial", data = OJ.train, cost = tu
ne.out$best.parameter$cost)
summary(svm_radial)
```

```
##
## Call:
## svm(formula = Purchase ~ ., data = OJ.train, kernel = "radial",
##     cost = tune.out$best.parameter$cost)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  1
##       gamma:  0.05555556
##
## Number of Support Vectors:  379
##
##  ( 188 191 )
##
##
## 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 455  39
##   MM  77 229
```

```
test.pred <- predict(svm_radial, OJ.test)
table(OJ.test$Purchase, test.pred)
```

```
##     test.pred
##       CH  MM
##   CH 141  18
##   MM  28  83
```

```
# Tuning does not reduce train and test error rates as we already used the op
timal cost of 1.

# (g) Repeat parts (b) through (e) using a support vector machine with a poly
nomial kernel. Set degree=2. Use the tune() function to select an optimal cos
t.
```

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

```
##
## Call:
## svm(formula = Purchase ~ ., data = OJ.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:  454
##
##  ( 224 230 )
##
##
## 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 461  33
##   MM 105 201
```

```r
test.pred <- predict(svm_poly, OJ.test)
table(OJ.test$Purchase, test.pred)
```

```
##     test.pred
##       CH  MM
##   CH 149  10
##   MM  41  70
```

```r
# Polynomial kernel with default gamma creates 454 support vectors, out of wh
ich, 224 belong to level CH and remaining 230 belong to level MM. The classif
ier has a training error of 17.2% and a test error of 18.8% which is no impro
vement over linear kernel. We now use cross validation to find optimal cost.

set.seed(2)
tune.out <- tune(svm, Purchase ~ ., data = OJ.train, kernel = "polynomial", d
```

```r
egree = 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: 0.18125
##
## - Detailed performance results:
##           cost   error dispersion
## 1   0.01000000 0.38250 0.04533824
## 2   0.01778279 0.36750 0.04972145
## 3   0.03162278 0.36500 0.05458174
## 4   0.05623413 0.33375 0.05070681
## 5   0.10000000 0.32500 0.04677072
## 6   0.17782794 0.25875 0.05952649
## 7   0.31622777 0.21250 0.06123724
## 8   0.56234133 0.21250 0.05743354
## 9   1.00000000 0.19750 0.06687468
## 10  1.77827941 0.19375 0.05376453
## 11  3.16227766 0.19625 0.05653477
## 12  5.62341325 0.18375 0.05434266
## 13 10.00000000 0.18125 0.05245699
```

```r
svm_poly <- svm(Purchase ~ ., kernel = "polynomial", degree = 2, data = OJ.tr
ain, cost = tune.out$best.parameter$cost)
summary(svm_poly)
```

```
##
## Call:
## svm(formula = Purchase ~ ., data = OJ.train, kernel = "polynomial",
##     degree = 2, cost = tune.out$best.parameter$cost)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  polynomial
##        cost:  10
##      degree:  2
##       gamma:  0.05555556
##      coef.0:  0
##
## Number of Support Vectors:  342
##
```

```
##  ( 170 172 )
##
##
## 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  44
##   MM  72 234
```

```r
test.pred <- predict(svm_poly, OJ.test)
table(OJ.test$Purchase, test.pred)
```

```
##     test.pred
##       CH  MM
##   CH 140  19
##   MM  31  80
```

```r
# Tuning reduce train and test error rates.

# (h) Overall, which approach seems to give the best results on this data?

# Overall, radial basis kernel seems to be producing minimum misclassificatio
n error on both train and test data.
```