# HW7

Melchor Ronquillo

2022-12-08

8. This problem involves the OJ data set which is part of the ISLR2 package.
   (a) Create a training set containing a random sample of 800 observations, and a test set containing the remaining observations.

```
install.packages('ISLR2', repos = "http://cran.us.r-project.org")
```

```
##
## The downloaded binary packages are in
##  /var/folders/b4/vbzhgztj3tj299xgxnklp28c0000gn/T//RtmpFbKzMC/downloaded_packages
```

```
library(ISLR2)

Juice <- OJ
#Juice

set.seed(777)
randomJuice= Juice[sample(1:nrow(Juice)), ]
#randomJuice

nrow(randomJuice)
```

```
## [1] 1070
```

```
train_OJ <- randomJuice[1:800,]
test_OJ <- randomJuice[801:1070,]

#train_OJ
#test_OJ
```

```
(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.
```

```
install.packages('e1071', repos = "http://cran.us.r-project.org")
```

```
##
## The downloaded binary packages are in
##  /var/folders/b4/vbzhgztj3tj299xgxnklp28c0000gn/T//RtmpFbKzMC/downloaded_packages
```

```
library(e1071)

set.seed(778)
JuiceSVC <- svm(Purchase ~ ., data = train_OJ, kernel = "linear", cost = 0.01, scale = TRUE)
#plot(JuiceSVM, train_OJ)
summary(JuiceSVC)
```

```
##
## Call:
## svm(formula = Purchase ~ ., data = train_OJ, kernel = "linear", cost = 0.01,
##     scale = TRUE)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  0.01
##
## Number of Support Vectors:  432
##
##  ( 216 216 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

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

```r
train_OJ_pred = predict(JuiceSVC, train_OJ)
table(train_OJ$Purchase, train_OJ_pred)
```

```
##     train_OJ_pred
##       CH  MM
##   CH 433  57
##   MM  75 235
```

```r
train_error = (57+75) / (453+57+75+235)
train_error
```

```
## [1] 0.1609756
```

Train error rate:
16% error

```r
test_OJ_pred = predict(JuiceSVC, test_OJ)
table(test_OJ$Purchase, test_OJ_pred)
```

```
##     test_OJ_pred
##       CH  MM
##   CH 137  26
##   MM  24  83
```

```r
test_error = (26+24) / (137+26+24+83)
test_error
```

```
## [1] 0.1851852
```

Test error rate:
18.5%

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

```r
set.seed(779)
tuneJuice <- tune(svm, Purchase ~ ., data = Juice, kernel = "linear",
                  ranges = list(cost = c(0.01, 0.05, 0.1, 0.5, 1, 5, 10))
                  )
summary(tuneJuice)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##     1
##
## - best performance: 0.1654206
##
## - Detailed performance results:
##    cost     error dispersion
## 1  0.01 0.1691589 0.03098083
## 2  0.05 0.1700935 0.03261371
## 3  0.10 0.1710280 0.03470411
## 4  0.50 0.1682243 0.03296886
## 5  1.00 0.1654206 0.03442332
## 6  5.00 0.1672897 0.03002637
## 7 10.00 0.1700935 0.03170844
```

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

```r
bestmod <- tuneJuice$best.model
summary(bestmod)
```

```
##
## Call:
## best.tune(METHOD = svm, train.x = Purchase ~ ., data = Juice, ranges = list(cost = c(0.01,
##     0.05, 0.1, 0.5, 1, 5, 10)), kernel = "linear")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  1
##
## Number of Support Vectors:  442
##
##  ( 221 221 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

```r
bestpred_OJ_SVC <- predict(bestmod, test_OJ)
table(test_OJ$Purchase, bestpred_OJ_SVC)
```

```
##     bestpred_OJ_SVC
##       CH  MM
##   CH 140  23
##   MM  23  84
```

```r
test_error = (23+23) / (140+23+23+84)
test_error
```

```
## [1] 0.1703704
```

    Test Error: 17%

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

```r
#Create SVM
set.seed(780)
JuiceSVM <- svm(Purchase ~ ., data = train_OJ, kernel = "radial",
                gamma = 1, cost = 0.01, scale = TRUE)
#plot(JuiceSVM, train_OJ)
summary(JuiceSVM)
```

```
##
## Call:
## svm(formula = Purchase ~ ., data = train_OJ, kernel = "radial", gamma = 1,
##     cost = 0.01, scale = TRUE)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  0.01
##
## Number of Support Vectors:  653
##
##  ( 343 310 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

```r
#train and test errors
train_OJ_predSVM = predict(JuiceSVM, train_OJ)
table(train_OJ$Purchase, train_OJ_predSVM)
```

```
##     train_OJ_predSVM
##       CH  MM
##   CH 490   0
##   MM 310   0
```

```r
test_OJ_predSVM = predict(JuiceSVM, test_OJ)
table(test_OJ$Purchase, test_OJ_predSVM)
```

```
##     test_OJ_predSVM
##       CH  MM
```

```
##   CH 163   0
##   MM 107   0
```

```r
#Tune
set.seed(781)
tuneJuiceSVM <- tune(svm, Purchase ~ ., data = Juice, kernel = "radial",
                  ranges = list(cost = c(0.01, 0.05, 0.1, 0.5, 1, 5, 10))
                  )

summary(tuneJuiceSVM)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##   0.5
##
## - best performance: 0.1728972
##
## - Detailed performance results:
##    cost     error dispersion
## 1  0.01 0.3897196 0.04746052
## 2  0.05 0.1971963 0.02222562
## 3  0.10 0.1869159 0.02530853
## 4  0.50 0.1728972 0.03475999
## 5  1.00 0.1728972 0.02963598
## 6  5.00 0.1738318 0.03152426
## 7 10.00 0.1766355 0.03002637
```

```r
bestmodSVM <- tuneJuiceSVM$best.model
summary(bestmodSVM)
```

```
##
## Call:
## best.tune(METHOD = svm, train.x = Purchase ~ ., data = Juice, ranges = list(cost = c(0.01,
##     0.05, 0.1, 0.5, 1, 5, 10)), kernel = "radial")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  0.5
##
## Number of Support Vectors:  523
##
##  ( 265 258 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

```r
bestpred_OJ_SVM <- predict(bestmodSVM, test_OJ)
table(test_OJ$Purchase, bestpred_OJ_SVM)
```

```
##     bestpred_OJ_SVM
##       CH  MM
##   CH 149  14
##   MM  26  81
```

```r
test_error = (14+26) / (149+14+26+81)
test_error
```

```
## [1] 0.1481481
```

Test Error: 14.8%

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

```r
set.seed(782)
JuiceSVCPoly <- svm(Purchase ~ ., data = train_OJ, kernel = "polynomial",
                    cost = 0.01,
                    degree = 2,
                    scale = TRUE)
summary(JuiceSVCPoly)
```

```
##
## Call:
## svm(formula = Purchase ~ ., data = train_OJ, kernel = "polynomial",
##     cost = 0.01, degree = 2, scale = TRUE)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  polynomial
##        cost:  0.01
##      degree:  2
##      coef.0:  0
##
## Number of Support Vectors:  627
##
##  ( 317 310 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

```r
#train and test errors
train_OJ_predPoly = predict(JuiceSVCPoly, train_OJ)
table(train_OJ$Purchase, train_OJ_predPoly)
```

```
##     train_OJ_predPoly
##       CH  MM
```

```
##   CH 489   1
##   MM 306   4
```

```
test_OJ_predPoly = predict(JuiceSVCPoly, test_OJ)
table(test_OJ$Purchase, test_OJ_predPoly)
```

```
##     test_OJ_predPoly
##       CH  MM
##   CH 163   0
##   MM 106   1
```

```
#tune
set.seed(779)
tuneJuicePoly <- tune(svm, Purchase ~ ., data = Juice, kernel = "polynomial",
                      degree = 2, scale = TRUE,
                      ranges = list(cost = c(0.01, 0.05, 0.1, 0.5, 1, 5, 10))
                  )
summary(tuneJuicePoly)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##    10
##
## - best performance: 0.1757009
##
## - Detailed performance results:
##    cost     error dispersion
## 1  0.01 0.3691589 0.04322254
## 2  0.05 0.3299065 0.04340180
## 3  0.10 0.2990654 0.04427623
## 4  0.50 0.2056075 0.02643390
## 5  1.00 0.1943925 0.02636037
## 6  5.00 0.1785047 0.02970140
## 7 10.00 0.1757009 0.03935604
```

```
#best model
bestmodPoly <- tuneJuicePoly$best.model
summary(bestmodPoly)
```

```
##
## Call:
## best.tune(METHOD = svm, train.x = Purchase ~ ., data = Juice, ranges = list(cost = c(0.01,
##     0.05, 0.1, 0.5, 1, 5, 10)), kernel = "polynomial", degree = 2,
##     scale = TRUE)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  polynomial
##        cost:  10
##      degree:  2
```

```
##       coef.0:  0
##
## Number of Support Vectors:  450
##
##  ( 230 220 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

```r
bestpred_OJ_Poly <- predict(bestmodPoly, test_OJ)
table(test_OJ$Purchase, bestpred_OJ_Poly)
```

```
##     bestpred_OJ_Poly
##       CH  MM
##   CH 148  15
##   MM  25  82
```

```r
test_error = (15+25) / (148+15+25+82)
test_error
```

```
## [1] 0.1481481
```

Test Error: 14.8%

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

Overall, the approach witht he best results was both radial and polynomial, as they both had the lowest test error rate of 14.8%