Exercise 7

Support Vector Machines

Wenjie Tu

March 2021

(a)

```
# load relevant libraries
library(ISLR) # OJ dataset
library(e1071) # SVM
library(scales) # print percentage

# check the structure of the dataset
head(OJ)
```

```
##   Purchase WeekofPurchase StoreID PriceCH PriceMM DiscCH DiscMM SpecialCH
## 1       CH            237       1    1.75    1.99   0.00    0.0         0
## 2       CH            239       1    1.75    1.99   0.00    0.3         0
## 3       CH            245       1    1.86    2.09   0.17    0.0         0
## 4       MM            227       1    1.69    1.69   0.00    0.0         0
## 5       CH            228       7    1.69    1.69   0.00    0.0         0
## 6       CH            230       7    1.69    1.99   0.00    0.0         0
##   SpecialMM  LoyalCH SalePriceMM SalePriceCH PriceDiff Store7 PctDiscMM
## 1         0 0.500000        1.99        1.75      0.24     No  0.000000
## 2         1 0.600000        1.69        1.75     -0.06     No  0.150754
## 3         0 0.680000        2.09        1.69      0.40     No  0.000000
## 4         0 0.400000        1.69        1.69      0.00     No  0.000000
## 5         0 0.956535        1.69        1.69      0.00    Yes  0.000000
## 6         1 0.965228        1.99        1.69      0.30    Yes  0.000000
##   PctDiscCH ListPriceDiff STORE
## 1  0.000000          0.24     1
## 2  0.000000          0.24     1
## 3  0.091398          0.23     1
## 4  0.000000          0.00     1
## 5  0.000000          0.00     0
## 6  0.000000          0.30     0
```

```
# set the seed for reproducibility
set.seed(2021)

# get a random sample of 800 indices from the dataset
train.idx <- sample(1:dim(OJ)[1], size=800, replace=F)
```

```r
# get training data and test data
train <- OJ[train.idx,]
test <- OJ[-train.idx,]

# initialize dataframe to store error rates
results <- data.frame(matrix(ncol=2,nrow=6))
colnames(results) <- c('Train ER', 'Test ER')
rownames(results) <- c('Linear with cost = 0.01','Linear with optimal cost',
              'Radial with default cost','Radial with optimal cost',
              'Polynomial with default cost','Polynomial with optimal cost')
```

(b)

Linear Kernel

$$K(x_i, x_i') = \sum_{j=1}^{p} x_{ij} x_{i'j}$$

```r
# use a support vector classifier
svm.linear <- svm(Purchase~., kernel='linear', data=train, cost=0.01)
summary(svm.linear)
```

```
##
## Call:
## svm(formula = Purchase ~ ., data = train, kernel = "linear", cost = 0.01)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  0.01
##
## Number of Support Vectors:  434
##
##  ( 217 217 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

The summary statistics tells us that a linear kernel was used with $cost = 0.01$, and there were 434 support vectors, 217 in CH class and 217 in MM class.

(c)

```r
# calculate the training error rate
train.error.rate <- mean(svm.linear$fitted!=train$Purchase)

# calculate the test error rate
```

```r
test.predict <- predict(svm.linear, newdata=test)
test.error.rate <- mean(test.predict!=test$Purchase)

# store the error rates
results[1,] <- c(train.error.rate, test.error.rate)

# print error rates
cat(sprintf('Training error rate is %s
            \nTest error rate is %s',
            label_percent(accuracy = 0.01)(train.error.rate),
            label_percent(accuracy = 0.01)(test.error.rate)))
```

```
## Training error rate is 16.50%
##
## Test error rate is 17.41%
```

(d)

```r
# set the seed for reproducibility
set.seed(0308)

# search for the optimal cost from 0.01 to 10
tune.out <- tune(svm, Purchase~., data=train, kernel='linear',
                 ranges=list(cost=10^seq(-2, 1, by=0.2)))
summary(tune.out)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##      cost
##  1.584893
##
## - best performance: 0.15875
##
## - Detailed performance results:
##           cost   error dispersion
## 1   0.01000000 0.17250 0.04594683
## 2   0.01584893 0.16750 0.04794383
## 3   0.02511886 0.16750 0.04866267
## 4   0.03981072 0.16250 0.04487637
## 5   0.06309573 0.16750 0.04721405
## 6   0.10000000 0.16750 0.04830459
## 7   0.15848932 0.16625 0.04860913
## 8   0.25118864 0.16500 0.05329426
## 9   0.39810717 0.16500 0.05394184
## 10  0.63095734 0.16500 0.05583955
## 11  1.00000000 0.16250 0.05137012
## 12  1.58489319 0.15875 0.05337563
```

```
## 13  2.51188643 0.16000 0.05394184
## 14  3.98107171 0.16375 0.05382908
## 15  6.30957344 0.16375 0.05050096
## 16 10.00000000 0.16375 0.05015601
```

```
# print the best cost
cost.best <- round(tune.out$best.parameters$cost, digits=3)
print(paste('The optimal cost is', cost.best))
```

```
## [1] "The optimal cost is 1.585"
```

(e)

```
# calculate the training error rate
train.error.rate <- mean(tune.out$best.model$fitted!=train$Purchase)

# calculate the test error rate
test.predict <- predict(tune.out$best.model, newdata=test)
test.error.rate <- mean(test.predict!=test$Purchase)

# store the error rates
results[2,] <- c(train.error.rate, test.error.rate)

# print error rates
cat(sprintf('Training error rate is %s
            \nTest error rate is %s',
            label_percent(accuracy = 0.01)(train.error.rate),
            label_percent(accuracy = 0.01)(test.error.rate)))
```

```
## Training error rate is 15.38%
##
## Test error rate is 17.78%
```

(f)

Radial Kernel

$$K(x_i, x_i') = \exp(-\gamma \sum_{j=1}^{p} (x_{ij} - x_{i'j})^2)$$

```
# set the seed for reproducibility
set.seed(2020)

# use support vector machine with a radial kernel
svm.radial <- svm(Purchase~., data=train, kernel='radial')
summary(svm.radial)
```

```
##
## Call:
## svm(formula = Purchase ~ ., data = train, kernel = "radial")
```

```
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  1
##
## Number of Support Vectors:  366
##
##  ( 186 180 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

```
# calculate the training error rate
train.error.rate <- mean(svm.radial$fitted!=train$Purchase)

# calculate the test error rate
test.predict <- predict(svm.radial, newdata=test)
test.error.rate <- mean(test.predict!=test$Purchase)

# store the error rates
results[3,] <- c(train.error.rate, test.error.rate)

# print error rates
cat(sprintf('Training error rate is %s
            \nTest error rate is %s',
            label_percent(accuracy = 0.01)(train.error.rate),
            label_percent(accuracy = 0.01)(test.error.rate)))
```

```
## Training error rate is 14.00%
##
## Test error rate is 19.26%
```

The summary statistics tells us that a radial kernel was used with a default $\gamma = \frac{1}{Dimension}$, and there were 639 support vectors, among which 321 observations were classified as CH class and 318 were classified as MM class.

```
# set the seed for reproducibility
set.seed(0901)

# search for the optimal cost from 0.01 to 10
tune.out <- tune(svm, Purchase~., data=train, kernel='radial',
                 ranges=list(cost=10^seq(-2, 1, by=0.2)))
summary(tune.out)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
```

```
##
## - best parameters:
##      cost
##  2.511886
##
## - best performance: 0.16
##
## - Detailed performance results:
##           cost   error dispersion
## 1   0.01000000 0.39750 0.03809710
## 2   0.01584893 0.39750 0.03809710
## 3   0.02511886 0.39750 0.03855011
## 4   0.03981072 0.22125 0.05744865
## 5   0.06309573 0.18875 0.04505013
## 6   0.10000000 0.17875 0.05070681
## 7   0.15848932 0.17375 0.04427267
## 8   0.25118864 0.16625 0.04715886
## 9   0.39810717 0.16750 0.04866267
## 10  0.63095734 0.16250 0.04564355
## 11  1.00000000 0.16250 0.04526159
## 12  1.58489319 0.16000 0.05027701
## 13  2.51188643 0.16000 0.05130248
## 14  3.98107171 0.16250 0.04487637
## 15  6.30957344 0.16375 0.04505013
## 16 10.00000000 0.16750 0.04338138
```

```r
# calculate the training error rate
train.error.rate <- mean(tune.out$best.model$fitted!=train$Purchase)

# calculate the test error rate
test.predict <- predict(tune.out$best.model, newdata=test)
test.error.rate <- mean(test.predict!=test$Purchase)

# store the error rates
results[4,] <- c(train.error.rate, test.error.rate)

# print error rates
cat(sprintf('Training error rate is %s
            \nTest error rate is %s',
            label_percent(accuracy = 0.01)(train.error.rate),
            label_percent(accuracy = 0.01)(test.error.rate)))
```

```
## Training error rate is 13.38%
##
## Test error rate is 19.26%
```

(g)

Polynomial Kernel

$$K(x_i, x_i') = (1 + \sum_{j=1}^{p} x_{ij}x_{i'j})^d$$

```r
# set the seed for reproducibility
set.seed(0316)

# use support vector machine with a polynomial kernel
svm.polynomial <- svm(Purchase~., data=train, kernel='polynomial',
                      degree=2)
summary(svm.polynomial)
```

```
##
## Call:
## svm(formula = Purchase ~ ., data = train, kernel = "polynomial",
##     degree = 2)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  polynomial
##        cost:  1
##      degree:  2
##      coef.0:  0
##
## Number of Support Vectors:  446
##
##  ( 227 219 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

```r
# calculate the training error rate
train.error.rate <- mean(svm.polynomial$fitted!=train$Purchase)

# calculate the test error rate
test.predict <- predict(svm.polynomial, newdata=test)
test.error.rate <- mean(test.predict!=test$Purchase)

# store the error rates
results[5,] <- c(train.error.rate, test.error.rate)

# print error rates
cat(sprintf('Training error rate is %s
            \nTest error rate is %s',
            label_percent(accuracy = 0.01)(train.error.rate),
            label_percent(accuracy = 0.01)(test.error.rate)))
```

```
## Training error rate is 18.12%
##
## Test error rate is 20.00%
```

The summary statistics tells us that a polynomial kernel was used with $degree = 2$ and $cost = 1$, and there were 642 support vectors, among which 324 observations were classified as CH class and 318 observations were classified as MM class.

```r
# set the seed for reproducibility
set.seed(0323)

# search for the optimal cost from 0.01 to 10
tune.out <- tune(svm, Purchase~., data=train, kernel='polynomial',
                 ranges=list(cost=10^seq(-2, 1, by=0.2)))
summary(tune.out)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##      cost
##  3.981072
##
## - best performance: 0.175
##
## - Detailed performance results:
##           cost   error dispersion
## 1   0.01000000 0.37125 0.04528076
## 2   0.01584893 0.37250 0.04518481
## 3   0.02511886 0.34750 0.04440971
## 4   0.03981072 0.33000 0.04338138
## 5   0.06309573 0.32000 0.04090979
## 6   0.10000000 0.29125 0.04210189
## 7   0.15848932 0.24375 0.05376453
## 8   0.25118864 0.21375 0.05447030
## 9   0.39810717 0.20000 0.04787136
## 10  0.63095734 0.18625 0.05185785
## 11  1.00000000 0.17875 0.04788949
## 12  1.58489319 0.18125 0.03738408
## 13  2.51188643 0.17500 0.03632416
## 14  3.98107171 0.17500 0.03333333
## 15  6.30957344 0.18000 0.02898755
## 16 10.00000000 0.18375 0.02829041
```

```r
# calculate the training error rate
train.error.rate <- mean(tune.out$best.model$fitted!=train$Purchase)

# calculate the test error rate
test.predict <- predict(tune.out$best.model, newdata=test)
test.error.rate <- mean(test.predict!=test$Purchase)

# store the error rates
results[6,] <- c(train.error.rate, test.error.rate)

# print error rates
cat(sprintf('Training error rate is %s
            \nTest error rate is %s',
            label_percent(accuracy = 0.01)(train.error.rate),
            label_percent(accuracy = 0.01)(test.error.rate)))
```

```
## Training error rate is 14.37%
##
## Test error rate is 18.52%
```

(h)

Table 1: Summary results

	Train ER	Test ER
Linear with cost = 0.01	0.165	0.174
Linear with optimal cost	0.154	0.178
Radial with default cost	0.140	0.193
Radial with optimal cost	0.134	0.193
Polynomial with default cost	0.181	0.200
Polynomial with optimal cost	0.144	0.185

Relatively speaking, support vector classifier (SVM with a linear kernel) gives us the most decent results on both training data and test data. As shown on the table above, we can tell that SVM with a radial kernel and SVM with a polynomial kernel overfit the data. They perform well on the training data but bad on the test data. When evaluating a machine learning model, we also have to consider the generalization of the trained model.