Predictive Modeling

Series 7

Exercise 7.1

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

a) Create a binary variable that takes on a 1 for cars with gas mileage above the median, and a 0 for cars with gas mileage below the median.

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

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

d) Make some plots to back up your assertions in (b) and (c).

   Hint: We used the `plot()` function for `svm` objects only in cases with $p = 2$. When $p > 2$, you can use the `plot()` function to create plots displaying pairs of variables at a time. Essentially, instead of typing

```
plot(svmfit, dat)
```

   where `svmfit` contains your fitted model and `dat` is a data frame containing your data, you can type

```
plot(svmfit, dat, x1 ~ x4)
```

   in order to plot just the first and fourth variables. However, you must replace `x1` and `x4` with the correct variable names. To find out more, type `?plot.svm`.

Exercise 7.2

This problem involves the `OJ` data set, which is part of the ISLR package.

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

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.

c) What are the training and test error rates? Also give the corresponding confusion matrices.

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

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

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

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

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

Result Checker

Predictive Modeling

Solutions to Series 7

Solution 7.1

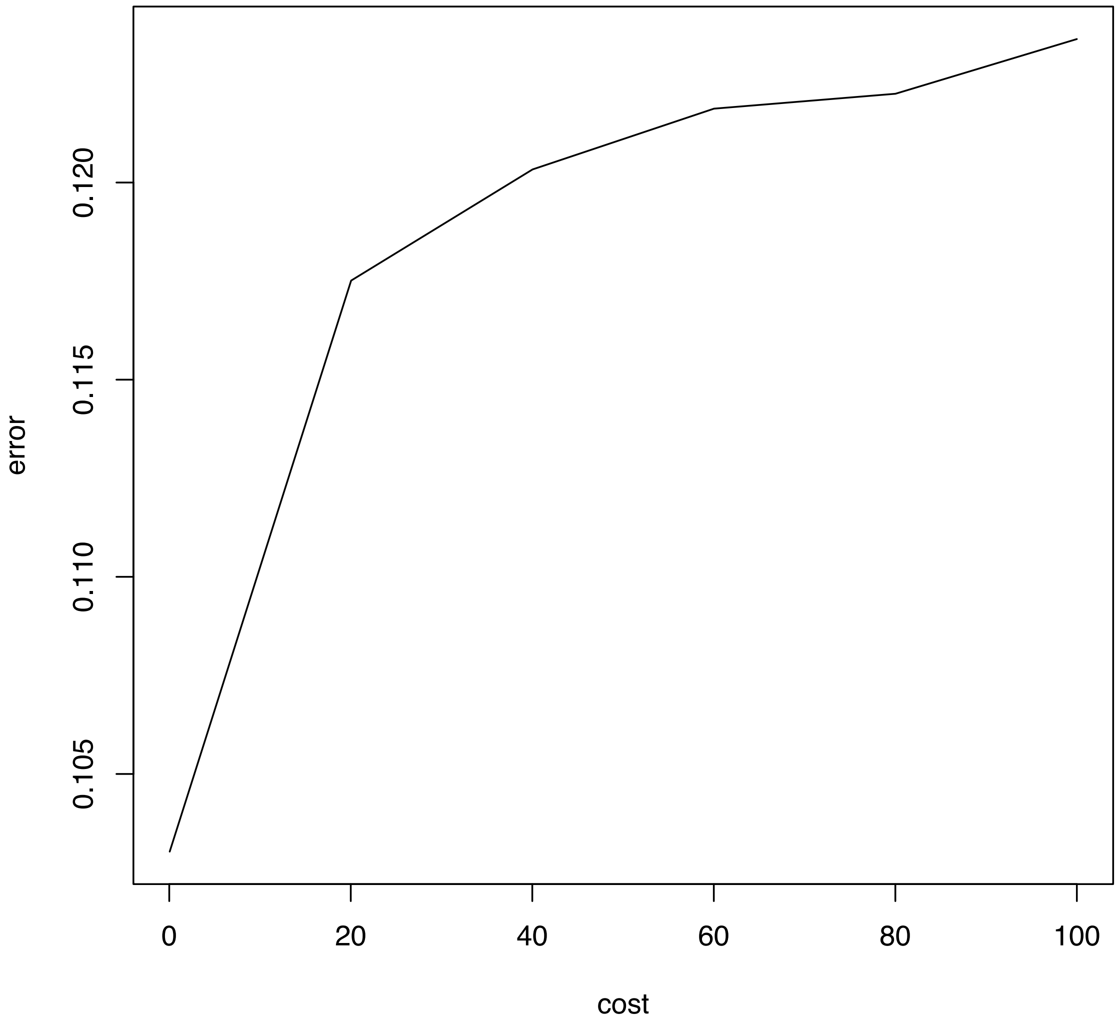
a)
```
require(ISLR)

## Loading required package: ISLR

Auto$mpg <- ifelse(Auto$mpg > median(Auto$mpg), 1, 0)
table(Auto$mpg)

##
##   0   1
## 196 196
```

b) The cross-validation errors can be produced with ease by the `tune` function from the `e1071` library. This is done by the code below.

```
require(e1071)

## Loading required package: e1071

# tuning grid for the cost parameter
costs <- data.frame(cost = seq(0.05, 100, length.out = 6))

# 10-fold cross validation
svm.tune.linear = tune(svm, mpg ~ ., data = Auto, ranges = costs,
    kernel = "linear")
svm.tune.linear

##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##  0.05
##
## - best performance: 0.103031
```

If we plot the values of the missclassification error, the minimum missclassification error occurs at cost 0.05.

```r
plot(svm.tune.linear$performance[, c(1, 2)], type = "l")
```

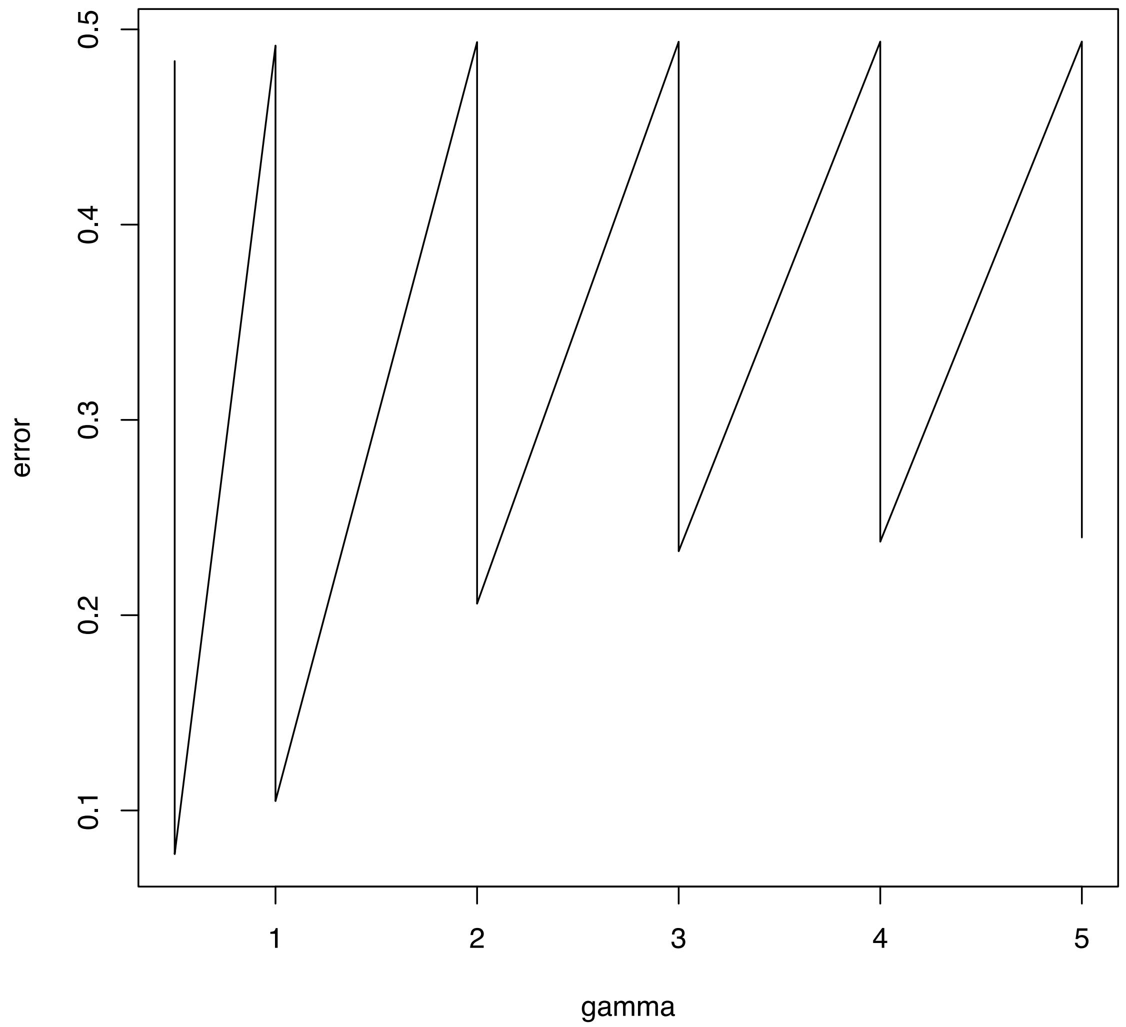


c) Radial Kernel:

```r
params <- data.frame(cost = seq(0.001, 240, length.out = 6),
    gamma = c(0.5, 1, 2, 3, 4, 5))
svm.tune.radial <- tune(svm, mpg ~ ., data = Auto, ranges = params,
    kernel = "radial")
svm.tune.radial

##
## Parameter tuning of 'svm':
##
```

```
## - sampling method: 10-fold cross validation
##
## - best parameters:
##      cost gamma
##   48.0008   0.5
##
## - best performance: 0.07764927
```

```
plot(svm.tune.radial$performance[, c(2, 3)], type = "l")
```

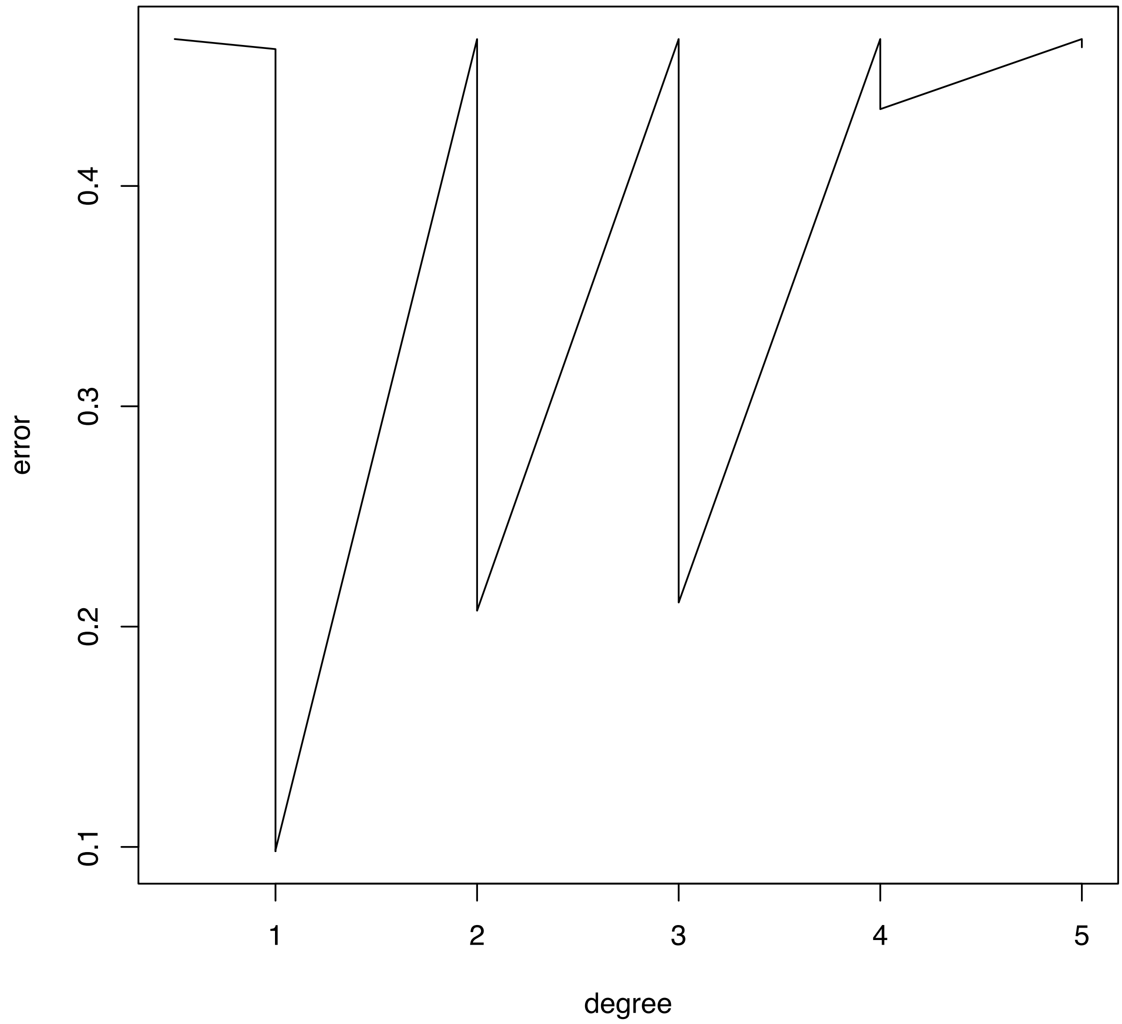


Polynomial Kernel:

```
params <- data.frame(cost = seq(0.001, 240, length.out = 6),
    degree = c(0.5, 1, 2, 3, 4, 5))
svm.tune.poly <- tune(svm, mpg ~ ., data = Auto, ranges = params,
    kernel = "polynomial")
svm.tune.poly
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##      cost degree
##  144.0004      1
##
## - best performance: 0.09806219
```

```r
plot(svm.tune.poly$performance[, c(2, 3)], type = "l")
```



d)
```r
plot(x = svm.tune.radial$best.model, data = Auto, formula = weight ~
    horsepower)
```

```r
plot(x = svm.tune.radial$best.model, data = Auto, formula = cylinders ~
    year)
```

Solution 7.2

a)
```
set.seed(42)
train = sample(1:1070, 800)
test = (1:1070)[-train]

tb = c()
res = c()
```

b)
```
require(ISLR)
require(e1071)

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

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

From the output of **R**'s summary function we can see that 432 observations are used as support vector. Moreover, the support vectors are almost equally split among the classes.

c)
```
# train
svm.pred = predict(svm.fit, OJ[train, ])
kable(table(OJ[train, "Purchase"], svm.pred))
```

	CH	MM
CH	432	60
MM	77	231

```
mean(OJ$Purchase[train] != svm.pred)

## [1] 0.17125
```

```
res = cbind(res, train = mean(OJ$Purchase[train] != svm.pred))
```

```
# test
svm.pred = predict(svm.fit, OJ[test, ])
kable(table(OJ[test, "Purchase"], svm.pred))
```

	CH	MM
CH	142	19
MM	25	84

```
mean(OJ$Purchase[test] != svm.pred)

## [1] 0.162963
```

```
res = cbind(res, test = mean(OJ$Purchase[test] != svm.pred))
```

d)
```
svm.tune = tune(svm, Purchase ~ ., data = OJ[train, ], ranges = data.
    10, 25)), kernel = "linear")
summary(svm.tune)

##
## Error estimation of 'svm' using 10-fold cross validation: 0.1825
```

```
res = cbind(res, CV = svm.tune$best.performance)
```

e)
```
svm.pred = predict(svm.tune$best.model, OJ[train, ])
kable(table(OJ[train, "Purchase"], svm.pred))
```

	CH	MM
CH	432	60
MM	77	231

```
mean(OJ$Purchase[train] != svm.pred)

## [1] 0.17125
```

```
res = cbind(res, train.tuned = mean(OJ$Purchase[train] !=
    svm.pred))
```

```
svm.pred = predict(svm.tune$best.model, OJ[test, ])
kable(table(OJ[test, "Purchase"], svm.pred))
```

	CH	MM
CH	142	19
MM	25	84

```
mean(OJ$Purchase[test] != svm.pred)

## [1] 0.162963
```

```
res = cbind(res, test.tuned = mean(OJ$Purchase[test] !=
    svm.pred))

tb = rbind(tb, res)
res = c()
```

f)

```
# b
svm.fit = svm(Purchase ~ ., data = OJ, subset = train, cost = 0.01,
    kernel = "radial")
summary(svm.fit)

##
## Call:
## svm(formula = Purchase ~ ., data = OJ, cost = 0.01,
##     kernel = "radial", subset = train)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  0.01
##
## Number of Support Vectors:  621
##
##  ( 308 313 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

```
# train
svm.pred = predict(svm.fit, OJ[train, ])
kable(table(OJ[train, "Purchase"], svm.pred))
```

	CH	MM
CH	492	0
MM	308	0

```
mean(OJ$Purchase[train] != svm.pred)

## [1] 0.385
```

```
res = cbind(res, train = mean(OJ$Purchase[train] != svm.pred))


# test
svm.pred = predict(svm.fit, OJ[test, ])
kable(table(OJ[test, "Purchase"], svm.pred))
```

	CH	MM
CH	161	0
MM	109	0

```
mean(OJ$Purchase[test] != svm.pred)

## [1] 0.4037037
```

```
res = cbind(res, train = mean(OJ$Purchase[test] != svm.pred))
```

```
svm.tune = tune(svm, Purchase ~ ., data = OJ[train, ], ranges = data.
    10, 25)))
summary(svm.tune)

##
## Error estimation of 'svm' using 10-fold cross validation: 0.385
```

```
res = cbind(res, CV = svm.tune$best.performance)
```

```
# train
svm.pred = predict(svm.tune$best.model, OJ[train, ])
kable(table(OJ[train, "Purchase"], svm.pred))
```

	CH	MM
CH	492	0
MM	308	0

```
mean(OJ$Purchase[train] != svm.pred)

## [1] 0.385
```

```r
res = cbind(res, train.tuned = mean(OJ$Purchase[train] !=
    svm.pred))


# test
svm.pred = predict(svm.tune$best.model, OJ[test, ])
kable(table(OJ[test, "Purchase"], svm.pred))
```

	CH	MM
CH	161	0
MM	109	0

```r
mean(OJ$Purchase[test] != svm.pred)

## [1] 0.4037037
```

```r
res = cbind(res, test.tuned = mean(OJ$Purchase[test] !=
    svm.pred))

tb = rbind(tb, res)
res = c()
```

g)
```r
# b
svm.fit = svm(Purchase ~ ., data = OJ, subset = train, cost = 0.01,
    degree = 2, kernel = "polynomial")
summary(svm.fit)

##
## Call:
## svm(formula = Purchase ~ ., data = OJ, cost = 0.01,
##     degree = 2, kernel = "polynomial", subset = train)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  polynomial
##        cost:  0.01
##      degree:  2
##      coef.0:  0
##
## Number of Support Vectors:  621
##
##  ( 308 313 )
##
```

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

```r
# train
svm.pred = predict(svm.fit, OJ[train, ])
kable(table(OJ[train, "Purchase"], svm.pred))
```

	CH	MM
CH	492	0
MM	308	0

```r
mean(OJ$Purchase[train] != svm.pred)

## [1] 0.385
```

```r
res = cbind(res, train = mean(OJ$Purchase[train] != svm.pred))

# test
svm.pred = predict(svm.fit, OJ[test, ])
kable(table(OJ[test, "Purchase"], svm.pred))
```

	CH	MM
CH	161	0
MM	109	0

```r
mean(OJ$Purchase[test] != svm.pred)

## [1] 0.4037037
```

```r
res = cbind(res, test = mean(OJ$Purchase[test] != svm.pred))
```

```r
svm.tune = tune(svm, Purchase ~ ., data = OJ[train, ], ranges = data.
    10, 25)), kernel = "polynomial", degree = 2)
summary(svm.tune)

##
## Error estimation of 'svm' using 10-fold cross validation: 0.385
```

```r
res = cbind(res, CV = svm.tune$best.performance)
```

```r
# train
svm.pred = predict(svm.tune$best.model, OJ[train, ])
kable(table(OJ[train, "Purchase"], svm.pred))
```

	CH	MM
CH	492	0
MM	308	0

```
mean(OJ$Purchase[train] != svm.pred)

## [1] 0.385
```

```
res = cbind(res, train.tuned = mean(OJ$Purchase[train] !=
    svm.pred))


# test
svm.pred = predict(svm.tune$best.model, OJ[test, ])
kable(table(OJ[test, "Purchase"], svm.pred))
```

	CH	MM
CH	161	0
MM	109	0

```
mean(OJ$Purchase[test] != svm.pred)

## [1] 0.4037037
```

```
res = cbind(res, test.tuned = mean(OJ$Purchase[test] !=
    svm.pred))

tb = rbind(tb, res)
```

h) The linear SVM performs best on this data. However, further investigating the optimal parameters for `gamma`, `degree` and `cost` could improve the behaviour of different classifiers.

```
rownames(tb) = c("LINEAR", "POLYNOMIAL", "RADIAL")
kable(tb)
```

	train	test	CV	train.tuned	test.tuned
LINEAR	0.17125	0.1629630	0.1825	0.17125	0.1629630
POLYNOMIAL	0.38500	0.4037037	0.3850	0.38500	0.4037037
RADIAL	0.38500	0.4037037	0.3850	0.38500	0.4037037