Chapter 9 Problem 7

Andira Putri

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 a gas mileage below the median.

```
#taken from Ch. 4 Problem 11
library(ISLR)
data(Auto)
#set a boundary using median
med=median(Auto$mpg)
#create binary variables
#1 if mpg is above median mpg, 0 otherwise
mpg01=ifelse(Auto$mpg>med,1,0)
#add binary variables to our Auto data set
auto2=data.frame(Auto,mpg01)
```

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.

Background on support vector classifiers (SVCs):

Rather than seeking the largest margin (hint: maximal margin classifier), we allow some observations to be on the incorrect side of the margin in support vector classifiers. This gives us greater robustness (not a lot of change in predictions when training set is modified) to individual observations and better classification of most training observations. In other words, misclassifying a few observations is WORTH IT!

SVCs make use of slack variables that allow observations to be on the wrong side of the hyperplane. A tuning parameter C bounds the sum of the slack variables, so it acts as a budget for how many violations we will allow.

- C=0, absolutely no budget for error –> maximal margin classifier
- small C –> low bias, high variance
- large C –> high bias, low variance from large margins and room for violations

CODING

We use the `e1071` library to create our SVC. To create a SVC, use the `svm()` function and use the argument `kernel="linear"` to differentiate from a support vector machine. We account for the tuning parameter `C` using the `cost` argument. Please note that the `cost` argument is opposite our definition of C previously; a high `cost` results in narrow margins and therefore smaller room for violations.

```
set.seed(1)
library(e1071)
svc=svm(mpg01~.,data=auto2,kernel="linear",cost=10,scale=FALSE)
```

Now, let's use CV and find the `cost` value where error is minimized. `tune()` performs ten-fold cross-validation on our model of interest.

```
tune.out=tune(svm,mpg01~.,data=auto2,kernel="linear",ranges=list(cost=c(0.01,1,5,10)))
summary(tune.out)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##     1
##
## - best performance: 0.07114663
##
## - Detailed performance results:
##    cost      error dispersion
## 1  0.01 0.08348075 0.03015789
## 2  1.00 0.07114663 0.02028651
## 3  5.00 0.07806023 0.02413691
## 4 10.00 0.08426130 0.02555851
```

We see that when `cost`= 1, we have the minimum error.

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

Support vector machines (SVMs) are almost the same as SVCs, except they have an enlarged feature space to accommodate a non-linear boundary. SVMs make use of basis kernels, which are general representations of the inner product between two observations. In this problem, we extend from the linear kernel of SVCs and make use of the radial and polynomial basis kernels.

CODING

SVMs are implemented generally the same way as SVCs. However, now we have additional arguments for `kernel` and `gamma` to think about.

```
#radial kernel
svm.rad=svm(mpg01~.,data=auto2,kernel="radial",gamma=0.5,cost=1,scale=FALSE)
#polynomial kernel
svm.poly=svm(mpg01~.,data=auto2,kernel="polynomial",cost=10,degree=2,scale=FALSE)
```

Again, we use the `tune()` function to perform ten-fold cross-validation on our SVMs, over a range of values for `cost`. For radial kernel, we have an additional range of values for `gamma`. For the polynomial kernel only, we have a range of values for `degree`.

```
tune.rad=tune(svm,mpg01~.,data=auto2,kernel="radial",ranges=list(cost=c(0.01,1,5,10),
                                                                 gamma=c(0.5,1,2,3)))

summary(tune.rad)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost gamma
##     1   0.5
##
## - best performance: 0.0494452
```

```
##
## - Detailed performance results:
##     cost gamma      error dispersion
## 1   0.01   0.5 0.40894884 0.04580611
## 2   1.00   0.5 0.04944520 0.01577709
## 3   5.00   0.5 0.05049183 0.01612975
## 4  10.00   0.5 0.05105067 0.01576364
## 5   0.01   1.0 0.48094134 0.04663955
## 6   1.00   1.0 0.09747304 0.01936442
## 7   5.00   1.0 0.09937784 0.02034672
## 8  10.00   1.0 0.09937784 0.02034672
## 9   0.01   2.0 0.49740491 0.04644036
## 10  1.00   2.0 0.20816385 0.01493330
## 11  5.00   2.0 0.20859325 0.01513708
## 12 10.00   2.0 0.20859325 0.01513708
## 13  0.01   3.0 0.49884416 0.04624527
## 14  1.00   3.0 0.23225338 0.01226836
## 15  5.00   3.0 0.23233516 0.01220982
## 16 10.00   3.0 0.23233516 0.01220982
```

```
tune.poly=tune(svm,mpg01~.,data=auto2,kernel="polynomial",ranges=list(cost=c(0.01,1,5,10),
                                                                      degree=c(2,3,4,5)))

summary(tune.poly)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost degree
##    10      2
##
## - best performance: 0.3578685
##
## - Detailed performance results:
##     cost degree     error dispersion
## 1   0.01      2 0.5302841 0.03204003
## 2   1.00      2 0.5088526 0.03218225
## 3   5.00      2 0.4321202 0.04213325
## 4  10.00      2 0.3578685 0.05463216
## 5   0.01      3 0.5304061 0.03204928
## 6   1.00      3 0.5216354 0.03248782
## 7   5.00      3 0.4873679 0.03522315
## 8  10.00      3 0.4472252 0.03975769
## 9   0.01      4 0.5304921 0.03204616
## 10  1.00      4 0.5303639 0.03203947
## 11  5.00      4 0.5298466 0.03201414
## 12 10.00      4 0.5291710 0.03200847
## 13  0.01      5 0.5304932 0.03204625
## 14  1.00      5 0.5304727 0.03204801
## 15  5.00      5 0.5303901 0.03205511
## 16 10.00      5 0.5302868 0.03206403
```

For the radial kernel, the best values for `cost` and `gamma` are 1 and 0.5, respectively.

For the polynomial kernel, the best values for `cost` and `degree` are 10 and 2, respectively.

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