Introduction to Statistical Learning 8/13/2017

Chapter 9: Support Vector Machines

Name: roninlaw

Email: roninlaw

1

(a)

```
# Sketch Hyperplane
x1 = -20:20
x2 = 1 + 3 * x1
plot(x1, x2, type = "l", col = "green")
```

(b)

```
# Sketch Hyperplane on Same Plot
lines(x1, 1 - x1/2)
```

5

(a)

```
# Generate data
set.seed(1)
x1 = runif(500) - .5
x2 = runif(500) - .5
y = 1*(x1^2 - x2^2 > 0)
```

(b)

```
# Plot observations
plot(x1[y == 0], x2[y == 0], col = "green")
points(x1[y == 1], x2[y == 1], col = "red")
```

(c)

```
# Fit Logistic Regression Model
glm.fit = glm(y ~ x1 + x2, family = binomial)
```

(d)

```
# Training Model Prediction
set.seed(1)
dat = data.frame(x1 = x1, x2 = x2, y = y)
glm.probs = predict(glm.fit, dat, type = "response")
glm.pred = rep(0, 500)
glm.pred[glm.probs > .5] = 1
lrRed = dat[glm.pred == 0, ]
lrGreen = dat[glm.pred == 1, ]
plot(lrRed$x1, lrRed$x2, col = "red", type = "p")
points(lrGreen$x1, lrGreen$x2, col = "green")
```

(e)

```
# Fit Logistic Regression w/ Non-Linear Functions
glm.nlfit = glm(y ~ log(x1) + log(x2^2), family = binomial)
```

(f)

```r
# Non-Linear Training Model Prediction
nlfit.probs = predict(glm.nlfit, dat, type = "response")
nlfit.pred = rep(0, 500)
nlfit.pred[nlfit.probs > .5] = 1
nlfit.lrRed = dat[nlfit.pred == 0, ]
nlfit.lrGreen = dat[nlfit.pred == 1, ]
plot(nlfit.lrRed$x1, nlfit.lrRed$x2, col = "red", type = "p")
points(nlfit.lrGreen$x1, nlfit.lrGreen$x2, col = "green")
```

(g)

```r
# Fit SVC
set.seed(1)
svmfit = svm(as.factor(y) ~ ., data = dat, kernel = "linear", cost = 1, scale = FALSE)
svm.pred = predict(svmfit, dat)
svm.red = dat[svm.pred == 0, ]
svm.green = dat[svm.pred == 1, ]
plot(svm.red$x1, svm.red$x2, col = "red", type = "p")
points(svm.green$x1, svm.green$x2, col = "green")
```

(h)

```r
# Fit SVM
set.seed(1)
svmfit = svm(as.factor(y) ~ ., data = dat, kernel = "radial", gamma = 5, scale = FALSE)
rad.pred = predict(svmfit, dat)
rad.red = dat[rad.pred == 0, ]
rad.green = dat[rad.pred == 1, ]
plot(rad.red$x1, rad.red$x2, col = "red", type = "p")
points(rad.green$x1, rad.green$x2, col = "green")
```

(i) SVM allows us to get a wide range of decision boundaries by simply specifiying which kernel function we want to use. It also provides a lot more flexibility than just using logistic regression.

7

(a)

```r
# Create Binary Variable
library(e1071)
mpg01 = rep(0, 392)
mpg01[Auto$mpg > median(Auto$mpg)] = 1
Auto = data.frame(Auto, mpg01)
```

(b)

```r
# Fit SVC w/ Various Values of Cost
set.seed(1)
tune.out = tune(svm, mpg01 ~ ., data = Auto, kernel = "linear",
              ranges = list(cost = c(.001, .01, 1, 5, 10, 100))) # cost = 1
```

(c)

```r
# Polynomial Kernel
set.seed(1)
tune.poly.out = tune(svm, mpg01 ~ ., data = Auto, kernel = "polynomial",
              ranges = list(cost = c(.001, 1, 5, 10, 100),
              degree = c(2, 3, 4))) # cost = 100, degree = 3
# Radial Kernel
```

```r
set.seed(1)
tune.rad.out = tune(svm, mpg01 ~ ., data = Auto, kernel = "polynomial",
                   ranges = list(cost = c(.001, 1, 5, 10, 100),
                   gamma = c(.01, .1, 1))) # cost 5, gamma = .1
```

(d)

```r
# Linear Plots
svm.lin = svm(mpg01 ~ ., data = Auto, kernel = "linear",
             cost = 1)
plot(svm.lin, Auto, acceleration ~ mpg)
plot(svm.lin, Auto, cylinders ~ mpg)
plot(svm.lin, Auto, displacement ~ mpg)
plot(svm.lin, Auto, horsepower ~ mpg)
plot(svm.lin, Auto, weight ~ mpg)
plot(svm.lin, Auto, year ~ mpg)
# Polynomial Plots
svm.poly = svm(mpg01 ~ ., data = Auto, kernel = "polynomial",
              cost = 100, degree = 3, scale = FALSE)
plot(svm.poly, Auto, i ~ mpg) # for i in [acceleration, cylinders, displacement,
    horsepower, weight, year]
# Radial Plots
svm.rad = svm(mpg01 ~ ., data = Auto, kernel = "radial",
             cost = 5, gamma = .1, scale = FALSE)
plot(svm.rad, Auto, i ~ mpg) # for i in [acceleration, cylinders, displacement,
    horsepower, weight, year]
```

8

(a)

```r
# Train/Test Split
train = sample(dim(OJ)[1], 800)
```

(b)

```r
# Fit SVC
svmfit = svm(Purchase ~ ., data = OJ[train, ], kernel = "linear",
             cost = .01)
summary(svmfit)
```

(c)

```r
# Test Error
table(true = OJ[-train, "Purchase"], pred = predict(svmfit,
                                              newdata = OJ[-train, ]))

# Train Error
table(true = OJ[train, "Purchase"], pred = predict(svmfit,
                                              newdata = OJ[train, ]))
```

(d)

```r
# Tune for Optimal Cost
tune.out = tune(svm, Purchase ~ ., data = OJ[train, ], kernel = "linear",
               ranges = list(cost = c(.01, .01, 1, 5, 10))) # cost = 10
```

(e)

```r
# Test Error w/ Optimal Cost
```

```r
table(true = OJ[-train, "Purchase"], pred = predict(tune.out$best.model,
                                                    newdata = OJ[-train, ]))
# Train Error w/ Optimal Cost
table(true = OJ[train, "Purchase"], pred = predict(tune.out$best.model,
                                                    newdata = OJ[train, ]))
```

(f)

```r
# Fit SVM w/ Radial Kernel
svmrad = svm(Purchase ~ ., data = OJ[train, ], kernel = "radial",
                    cost = .01)
summary(svmrad)

# Test Error
table(true = OJ[-train, "Purchase"], pred = predict(svmrad,
                                                    newdata = OJ[-train, ]))
# Train Error
table(true = OJ[train, "Purchase"], pred = predict(svmrad,
                                                    newdata = OJ[train, ]))
# Tune for Optimal Cost
tune.rad = tune(svm, Purchase ~ ., data = OJ[train, ], kernel = "radial",
               ranges = list(cost = c(.01, .01, 1, 5, 10)))# cost = 1
# Test Error w/ Optimal Cost
table(true = OJ[-train, "Purchase"], pred = predict(tune.rad$best.model,
                                                    newdata = OJ[-train, ]))
# Train Error w/ Optimal Cost
table(true = OJ[train, "Purchase"], pred = predict(tune.rad$best.model,
                                                   newdata = OJ[train, ]))
```

(g)

```r
# Fit SVM w/ Polynomial Kernel
svmpoly = svm(Purchase ~ ., data = OJ[train, ], kernel = "polynomial",
            cost = .01, degree = 2)
summary(svmpoly)
# Test Error
table(true = OJ[-train, "Purchase"], pred = predict(svmpoly,
                                                    newdata = OJ[-train, ]))
# Train Error
table(true = OJ[train, "Purchase"], pred = predict(svmpoly,
                                                    newdata = OJ[train, ]))
# Tune for Optimal Cost
tune.poly = tune(svm, Purchase ~ ., data = OJ[train, ], kernel = "polynomial", degree =
    2,
               ranges = list(cost = c(.01, .01, 1, 5, 10))) # cost = 10
# Test Error w/ Optimal Cost
table(true = OJ[-train, "Purchase"], pred = predict(tune.poly$best.model,
                                                    newdata = OJ[-train, ]))
# Train Error w/ Optimal Cost
table(true = OJ[train, "Purchase"], pred = predict(tune.poly$best.model,
                                                    newdata = OJ[train, ]))
```

(h) The polynomial kernel performed the best. In all of the models it seemed like starting with a random cost value did not seem like a good strategy for selecting a model as performing CV resulted in a decrease in train/test error rate for all of the kernel choices. One way the models could have been optimized even further would be to incorporate different degree and gamma values for the polynomial and radial kernels, respectively.