Module 9: Recommended Exercises

Statistical Learning V2021

alexaoh

17 mars, 2021

Problem 1

Work through the lab in Section 9.6.1 of ISLR.

Problem 2 (Book Ex.2)

We have seen that in $p = 2$ dimensions, a linear decision boundary takes the form $\beta_0 + \beta_1 X_1 + \beta_2 X_2 = 0$. We now investigate a non-linear decision boundary.

a) Sketch the curve

$$(1 + X_1)^2 + (2 - X_2)^2 = 4.$$

Done by hand. This is a circle with center $(-1, 2)$ and radius 2.

b) On your sketch, indicate the set of points for which

$$(1 + X_1)^2 + (2 - X_2)^2 > 4,$$

as well as the set of points for which

$$(1 + X_1)^2 + (2 - X_2)^2 \leq 4.$$

$(1 + X_1)^2 + (2 - X_2)^2 > 4$ are the points outside this circle, while $(1 + X_1)^2 + (2 - X_2)^2 \leq 4$ are the points inside and on the boundary of the circle.

c) Suppose that a classifier assigns an observation to the blue class if

$$(1 + X_1)^2 + (2 - X_2)^2 > 4,$$

and to the red class otherwise. To what class is the observation $(0, 0)$ classified? $(-1, 1)$? $(2, 2)$? $(3, 8)$?

- $(0, 0)$ is blue
- $(-1, 1)$ is red
- $(2, 2)$ is blue
- $(3, 8)$ is blue

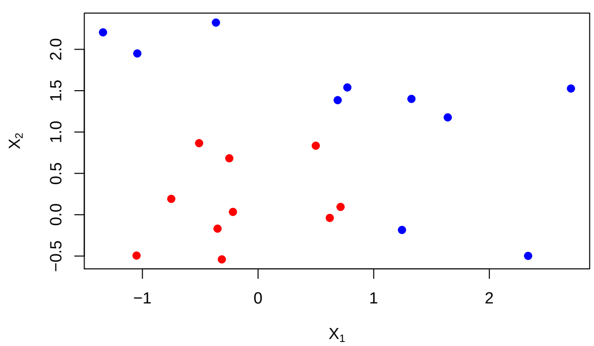
d) Argue that while the decision boundary in (c) is not linear in terms of X_1 and X_2, it is linear in terms of X_1, X_1^2, X_2, and X_2^2.

By expanding the terms in the formula for the circle, it is apparent that it contains terms like X_1, X_1^2, X_2, and X_2^2, in addition to a constant term. This means that the circle can be regarded as linear in terms of X_1, X_1^2, X_2, and X_2^2, by allowing these transformations of the variables X_i.

Problem 3

This problem involves plotting of decision boundaries for different kernels and it's taken from Lab video.

```r
# code taken from video by Trevor Hastie.
set.seed(10111)
x <- matrix(rnorm(40), 20, 2)
y <- rep(c(-1, 1), c(10, 10))
x[y == 1, ] <- x[y == 1, ] + 1
plot(x, col = y + 3, pch = 19, xlab = expression(X[1]), ylab = expression(X[2]))
```



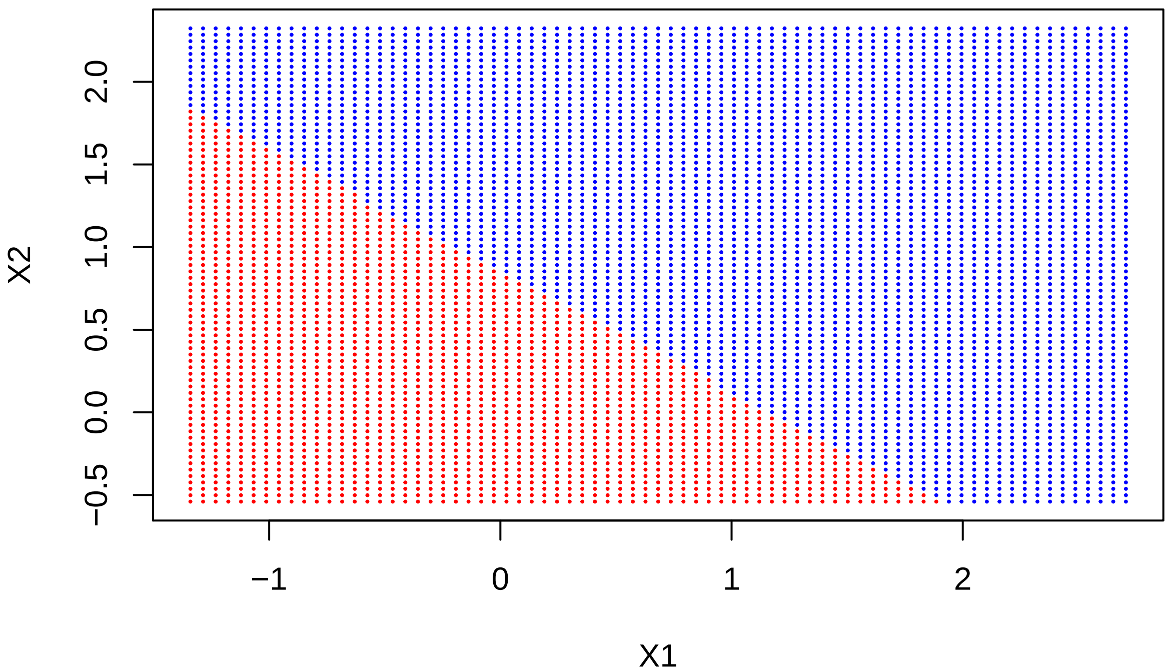

```r
dat = data.frame(x, y = as.factor(y))
```

(a) Plot the linear decision boundary of the `svmfit` model by using the function `make.grid`. Hint: Use the predict function for the grid points and then plot the predicted values {-1,1} with different colors.

```r
library(e1071)
svmfit = svm(y ~ ., data = dat, kernel = "linear", cost = 10, scale = F)  # E.g. cost 10
```

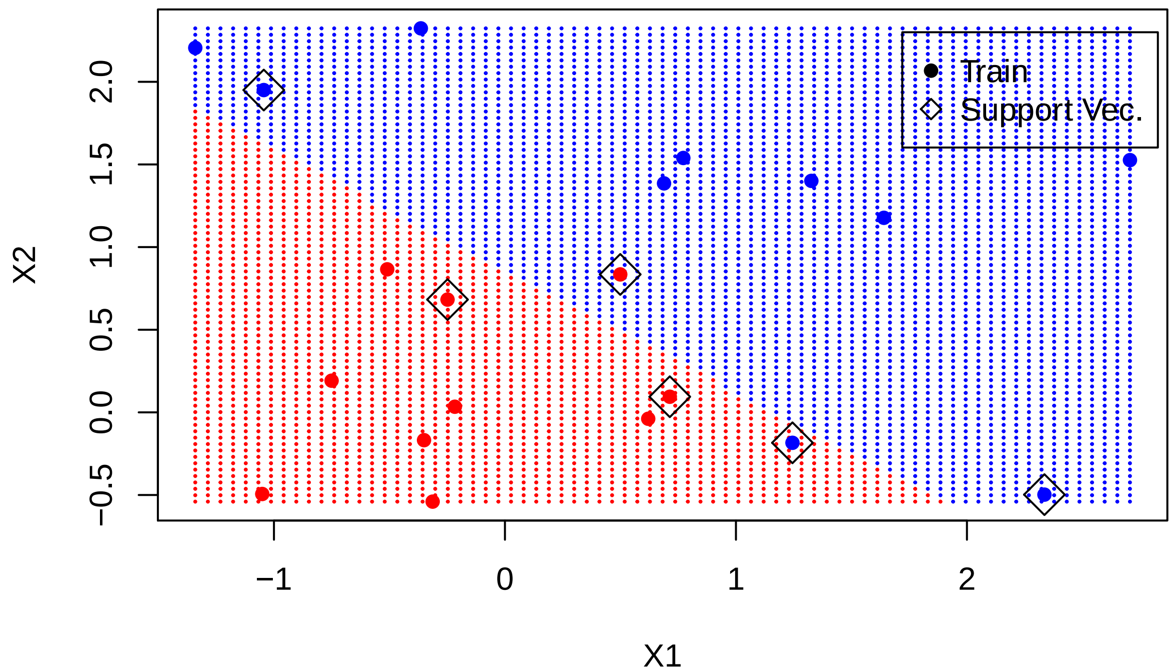
The following function may help you to generate a grid for plotting:

```r
make.grid = function(x, n = 75) {
    # takes as input the data matrix x and number of grid points n in
    # each direction the default value will generate a 75x75 grid
    grange = apply(x, 2, range)  # range for x1 and x2
    x1 = seq(from = grange[1, 1], to = grange[2, 1], length.out = n)  # sequence from the lowest to the
    x2 = seq(from = grange[1, 2], to = grange[2, 2], length.out = n)  # sequence from the lowest to the
    expand.grid(X1 = x1, X2 = x2)  #create a uniform grid according to x1 and x2 values
}
x <- as.matrix(dat[, c("X1", "X2")])  # Retrieve only the predictors X1 and X2.
xgrid <- make.grid(x)  # Make the grid with the supplied function.
preds <- predict(svmfit, xgrid)  # Make predictions on grid of testpoints.
plot(xgrid, col = c("red", "blue")[as.numeric(preds)], cex = 0.2, pch = 20)
```



(b) On the same plot add the training points and indicate the support vectors.

```
plot(xgrid, col = c("red", "blue")[as.numeric(preds)], cex = 0.2, pch = 20)
points(x, pch = 16, col = c("red", "blue")[as.numeric(dat[, 3])])
points(x[svmfit$index, ], pch = 5, cex = 2)
legend(1.72, 2.3, legend = c("Train", "Support Vec."), pch = c(16, 5))
```



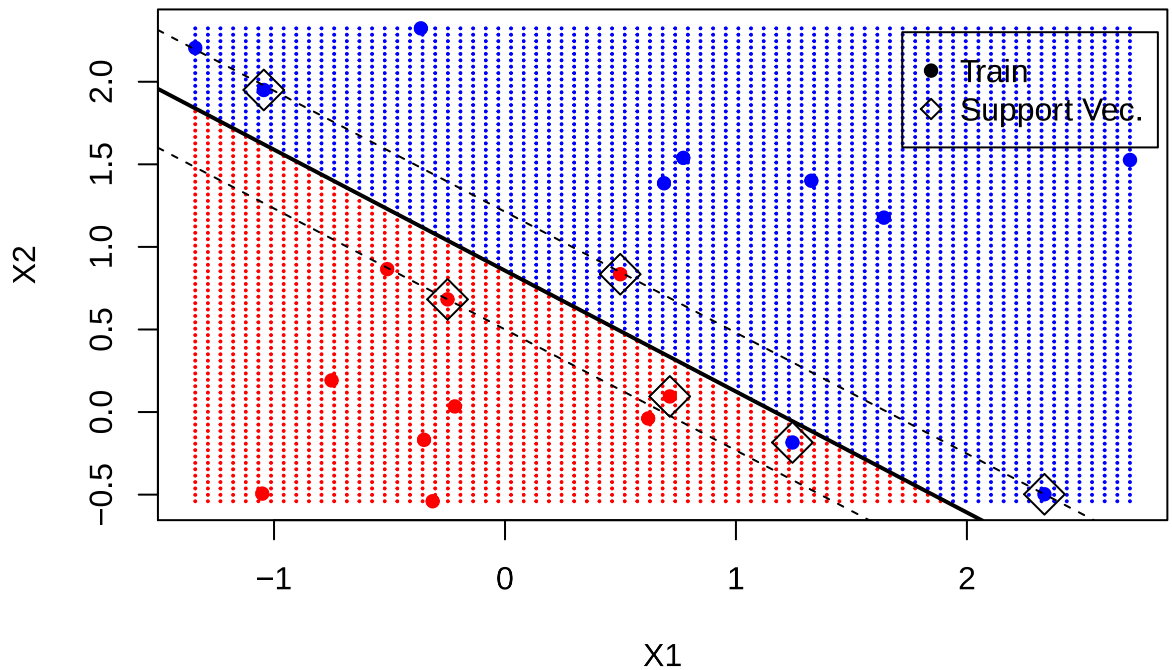
(c) The solutions to the SVM optimization problem is given by

$$\hat{\beta} = \sum_{i \in \mathcal{S}} \hat{\alpha}_i y_i x_i \ ,$$

where S is the set of the support vectors. From the `svm()` function we cannot extract $\hat{\beta}$, but instead we have access to $\text{coef}_i = \hat{\alpha}_i y_i$, and $\hat{\beta}_0$ is given as *rho*. For more details see here.

Calculate the coefficients $\hat{\beta}_0, \hat{\beta}_1$ and $\hat{\beta}_2$. Then add the decisision boundary and the margins using the function `abline()` on the plot from (b).

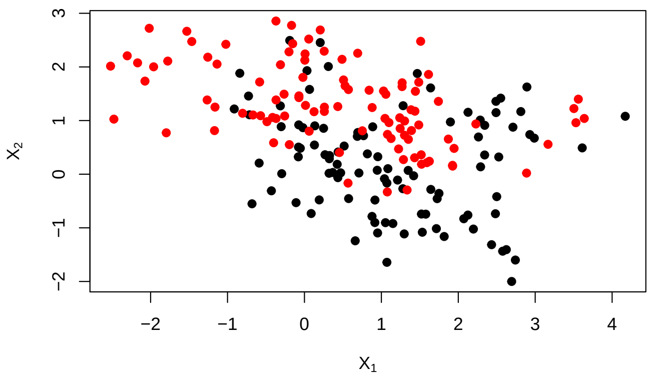
```
beta0 <- svmfit$rho
coefs <- svmfit$coefs
beta <- t(coefs) %*% x[svmfit$index, ]
plot(xgrid, col = c("red", "blue")[as.numeric(preds)], cex = 0.2, pch = 20)
points(x, pch = 16, col = c("red", "blue")[as.numeric(dat[, 3])])
points(x[svmfit$index, ], pch = 5, cex = 2)
legend(1.72, 2.3, legend = c("Train", "Support Vec."), pch = c(16, 5))

# Plot decision boundary.
abline(beta0/beta[2], -beta[1]/beta[2], lwd = 2)

# Plot upper and lower margin.
abline((beta0 - 1)/beta[2], -beta[1]/beta[2], lty = 2)
abline((beta0 + 1)/beta[2], -beta[1]/beta[2], lty = 2)
```



Problem 4

Now we fit an svm model with radial kernel to the following data taken from @ESL. Use cross-validation to find the best set of tuning parameters (cost C and γ). Using the same idea as in Problem 4a) plot the non-linear decision boundary, and add the training points. Furthermore if you want to create the decision boundary curve you can use the argument `decision.values=TRUE` in the function predict, and then you can plot it by using the `contour()` function.

```r
load(url("https://web.stanford.edu/~hastie/ElemStatLearn/datasets/ESL.mixture.rda"))
# names(ESL.mixture)
rm(x, y)
attach(ESL.mixture)
plot(x, col = y + 1, pch = 19, xlab = expression(X[1]), ylab = expression(X[2]))
```



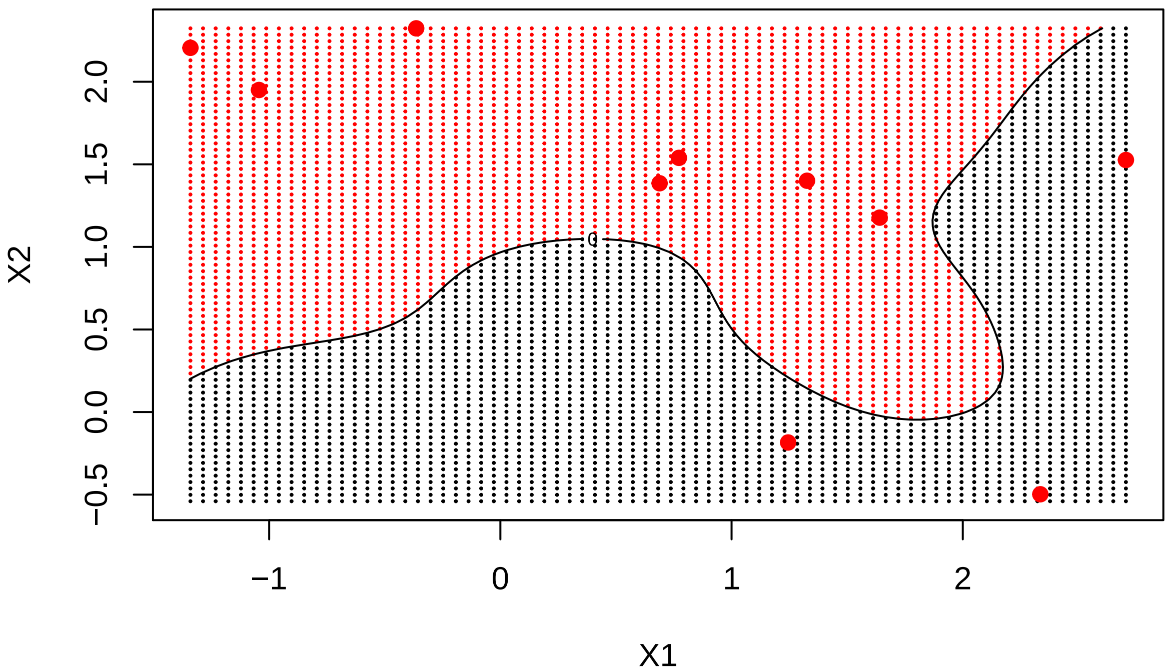

```r
dat = data.frame(y = factor(y), x)
```

To run cross-validation over a grid for (C, γ), you can use a two-dimensional list of values in the `ranges` argument:

```r
r.cv <- tune(svm, factor(y) ~ ., data = dat, kernel = "radial", ranges = list(cost = c(0.1,
    1, 10, 100, 1000), gamma = c(0.5, 1, 2, 3, 4, 6, 8)))
bestfit <- r.cv$best.model
```

For the plot:

```r
xgrid = make.grid(x)
ygrid = predict(bestfit, xgrid)
plot(xgrid, col = as.numeric(ygrid), pch = 20, cex = 0.2)
points(x, col = y + 1, pch = 19)

# decision boundary
func = predict(bestfit, xgrid, decision.values = TRUE)
func = attributes(func)$decision
contour(unique(xgrid[, 1]), unique(xgrid[, 2]), matrix(func, 75, 75),
    level = 0, add = TRUE)  #svm boundary
```



Problem 5 - optional (Book Ex. 7)

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.

```
library(ISLR)
data(OJ)
set.seed(4268)
train.indices <- sample(1:nrow(OJ), 800)
train.data <- OJ[train.indices, ]
test.data <- OJ[-train.indices, ]
```

(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.

```
linearOJ <- svm(Purchase ~ ., data = OJ, subset = train.indices, kernel = "linear",
    cost = 0.01)  # Scale = T is default.
summary(linearOJ)
```

```
#>
#> Call:
#> svm(formula = Purchase ~ ., data = OJ, kernel = "linear", cost = 0.01,
#>     subset = train.indices)
#>
#>
```

```
#> Parameters:
#>    SVM-Type:  C-classification
#>  SVM-Kernel:  linear
#>        cost:  0.01
#>
#> Number of Support Vectors:  431
#>
#>  ( 217 214 )
#>
#>
#> Number of Classes:  2
#>
#> Levels:
#>  CH MM
```

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

```r
# Training error rates.
train.pred <- predict(linearOJ, newdata = train.data)
conf.table.train <- table(predict = train.pred, true = train.data[, "Purchase"])
conf.table.train
```

```
#>        true
#> predict  CH  MM
#>      CH 431  78
#>      MM  56 235
```

```r
train.error.rate <- 1 - sum(diag(conf.table.train))/(sum(conf.table.train))
train.error.rate
```

```
#> [1] 0.1675
```

```r
# Testing error rates.
test.pred <- predict(linearOJ, newdata = test.data)
conf.table.test <- table(predict = test.pred, true = test.data[, "Purchase"])
conf.table.test
```

```
#>        true
#> predict  CH  MM
#>      CH 143  25
#>      MM  23  79
```

```r
test.error.rate <- 1 - sum(diag(conf.table.test))/(sum(conf.table.test))
test.error.rate
```

```
#> [1] 0.1777778
```

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

```r
linear.cv <- tune(svm, Purchase ~ ., data = train.data, kernel = "linear",
    ranges = list(cost = 10^seq(-2, 1, by = 0.25)))
bestfit.linear.OJ <- linear.cv$best.model
```

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

```r
# Training error rates.
train.pred.best.linear <- predict(bestfit.linear.OJ, newdata = train.data)
conf.table.train.best.linear <- table(predict = train.pred.best.linear,
    true = train.data[, "Purchase"])
conf.table.train.best.linear
```

```
#>        true
#> predict  CH  MM
#>      CH 432  74
#>      MM  55 239
```

```r
train.error.rate.best.linear <- 1 - sum(diag(conf.table.train.best.linear))/(sum(conf.table.train.best.l
train.error.rate.best.linear
```

```
#> [1] 0.16125
```

```r
# Testing error rates.
test.pred.best.linear <- predict(bestfit.linear.OJ, newdata = test.data)
conf.table.test.best.linear <- table(predict = test.pred.best.linear,
    true = test.data[, "Purchase"])
conf.table.test.best.linear
```

```
#>        true
#> predict  CH  MM
#>      CH 142  24
#>      MM  24  80
```

```r
test.error.rate.best.linear <- 1 - sum(diag(conf.table.test.best.linear))/(sum(conf.table.test.best.line
test.error.rate.best.linear
```

```
#> [1] 0.1777778
```

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

```r
radialOJ <- svm(Purchase ~ ., data = OJ, subset = train.indices, kernel = "radial")  # Scale = T is def
summary(radialOJ)
```

```
#>
#> Call:
#> svm(formula = Purchase ~ ., data = OJ, kernel = "radial", subset = train.indices)
#>
#>
#> Parameters:
#>    SVM-Type:  C-classification
#>  SVM-Kernel:  radial
#>        cost:  1
#>
#> Number of Support Vectors:  365
#>
#>  ( 185 180 )
#>
#>
#> Number of Classes:  2
#>
#> Levels:
#>  CH MM
```

```r
# Training error rates.
train.pred.radial <- predict(radialOJ, newdata = train.data)
conf.table.train.radial <- table(predict = train.pred.radial, true = train.data[,
    "Purchase"])
conf.table.train.radial
```

```
#>        true
```

```
#> predict  CH  MM
#>      CH 446  72
#>      MM  41 241
```

```
train.error.rate.radial <- 1 - sum(diag(conf.table.train.radial))/(sum(conf.table.train.radial))
train.error.rate.radial
```

```
#> [1] 0.14125
```

```
# Testing error rates.
test.pred.radial <- predict(radialOJ, newdata = test.data)
conf.table.test.radial <- table(predict = test.pred.radial, true = test.data[,
    "Purchase"])
conf.table.test.radial
```

```
#>        true
#> predict  CH  MM
#>      CH 145  25
#>      MM  21  79
```

```
test.error.rate.radial <- 1 - sum(diag(conf.table.test.radial))/(sum(conf.table.test.radial))
test.error.rate.radial
```

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

```
# Tune the model to find the best gamma and cost.
radial.cv <- tune(svm, Purchase ~ ., data = train.data, kernel = "radial",
    ranges = list(cost = 10^seq(-2, 1, by = 0.25)))  # Not tuning gamma.
bestfit.radial.OJ <- radial.cv$best.model

# Training error rates for tuned model.
train.pred.best.radial <- predict(bestfit.radial.OJ, newdata = train.data)
conf.table.train.best.radial <- table(predict = train.pred.best.radial,
    true = train.data[, "Purchase"])
conf.table.train.best.radial
```

```
#>        true
#> predict  CH  MM
#>      CH 450  73
#>      MM  37 240
```

```
train.error.rate.best.radial <- 1 - sum(diag(conf.table.train.best.radial))/(sum(conf.table.train.best.
train.error.rate.best.radial
```

```
#> [1] 0.1375
```

```
# Testing error rates for tuned model.
test.pred.best.radial <- predict(bestfit.radial.OJ, newdata = test.data)
conf.table.test.best.radial <- table(predict = test.pred.best.radial,
    true = test.data[, "Purchase"])
conf.table.test.best.radial
```

```
#>        true
#> predict  CH  MM
#>      CH 146  28
#>      MM  20  76
```

```
test.error.rate.best.radial <- 1 - sum(diag(conf.table.test.best.radial))/(sum(conf.table.test.best.radi
test.error.rate.best.radial
```

```
#> [1] 0.1777778
```

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

```r
polOJ <- svm(Purchase ~ ., data = OJ, subset = train.indices, kernel = "polynomial",
    degree = 2)  # Scale = T is default.
summary(polOJ)
```

```
#>
#> Call:
#> svm(formula = Purchase ~ ., data = OJ, kernel = "polynomial", degree = 2,
#>     subset = train.indices)
#>
#>
#> Parameters:
#>    SVM-Type:  C-classification
#>  SVM-Kernel:  polynomial
#>        cost:  1
#>      degree:  2
#>      coef.0:  0
#>
#> Number of Support Vectors:  454
#>
#>  ( 230 224 )
#>
#>
#> Number of Classes:  2
#>
#> Levels:
#>  CH MM
```

```r
# Training error rates.
train.pred.pol <- predict(polOJ, newdata = train.data)
conf.table.train.pol <- table(predict = train.pred.pol, true = train.data[,
    "Purchase"])
conf.table.train.pol
```

```
#>        true
#> predict  CH  MM
#>      CH 453 109
#>      MM  34 204
```

```r
train.error.rate.pol <- 1 - sum(diag(conf.table.train.pol))/(sum(conf.table.train.pol))
train.error.rate.pol
```

```
#> [1] 0.17875
```

```r
# Testing error rates.
test.pred.pol <- predict(polOJ, newdata = test.data)
conf.table.test.pol <- table(predict = test.pred.pol, true = test.data[,
    "Purchase"])
conf.table.test.pol
```

```
#>        true
#> predict  CH  MM
#>      CH 152  33
#>      MM  14  71
```

```r
test.error.rate.pol <- 1 - sum(diag(conf.table.test.pol))/(sum(conf.table.test.pol))
test.error.rate.pol
```

```
#> [1] 0.1740741
```

```r
# Tune the model to find the best gamma and cost.
pol.cv <- tune(svm, Purchase ~ ., data = train.data, kernel = "polynomial",
    ranges = list(cost = 10^seq(-2, 1, by = 0.25)))  # Not tuning degrees.
bestfit.pol.OJ <- pol.cv$best.model

# Training error rates for tuned model.
train.pred.best.pol <- predict(bestfit.pol.OJ, newdata = train.data)
conf.table.train.best.pol <- table(predict = train.pred.best.pol, true = train.data[,
    "Purchase"])
conf.table.train.best.pol
```

```
#>        true
#> predict  CH  MM
#>      CH 459  82
#>      MM  28 231
```

```r
train.error.rate.best.pol <- 1 - sum(diag(conf.table.train.best.pol))/(sum(conf.table.train.best.pol))
train.error.rate.best.pol
```

```
#> [1] 0.1375
```

```r
# Testing error rates for tuned model.
test.pred.best.pol <- predict(bestfit.pol.OJ, newdata = test.data)
conf.table.test.best.pol <- table(predict = test.pred.best.pol, true = test.data[,
    "Purchase"])
conf.table.test.best.pol
```

```
#>        true
#> predict  CH  MM
#>      CH 148  32
#>      MM  18  72
```

```r
test.error.rate.best.pol <- 1 - sum(diag(conf.table.test.best.pol))/(sum(conf.table.test.best.pol))
test.error.rate.best.pol
```

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

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

Summarized, we have

```r
msrate = cbind(c(train.error.rate.best.linear, train.error.rate.best.radial,
    train.error.rate.best.pol), c(test.error.rate.best.linear, test.error.rate.best.radial,
    test.error.rate.best.pol))
rownames(msrate) = c("linear", "radial", "polynomial")
colnames(msrate) = c("msrate.train", "msrate.test")
msrate
```

```
#>            msrate.train msrate.test
#> linear          0.16125   0.1777778
#> radial          0.13750   0.1962963
#> polynomial      0.14250   0.1592593
```

Based on the testing errors, the SVM with polynomial kernel seems to give the best results.