Module 9: Solutions to Recommended Exercises

TMA4268 Statistical Learning V2020

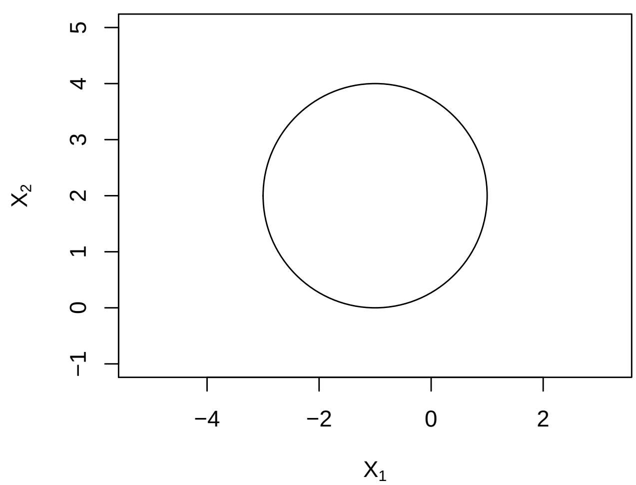
Martina Hall, Michail Spitieris, Stefanie Muff, Department of Mathematical Sciences, NTNU

March 12, 2020

Problem 2

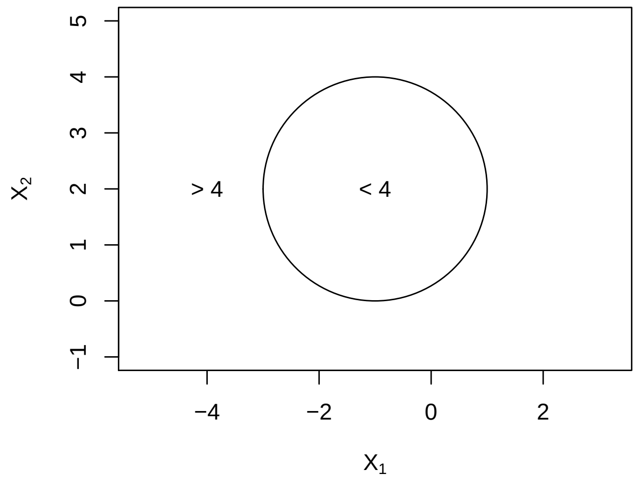
a) The curve is a circle with center (-1,2) and radius 2. You can sketch the curve by hand. If you want to do it in R, you can use the function `symbols()` (this is a bit advanced, though):

```
plot(NA,NA, # initialize a plot
     type = "n", # does not produce any points or lines
     xlim = c(-4,2),ylim = c(-1,5), xlab = expression(X[1]), ylab = expression(X[2]),
     asp = 1)
symbols(c(-1), c(2), circles = c(2), add = TRUE,  inches = FALSE)
```



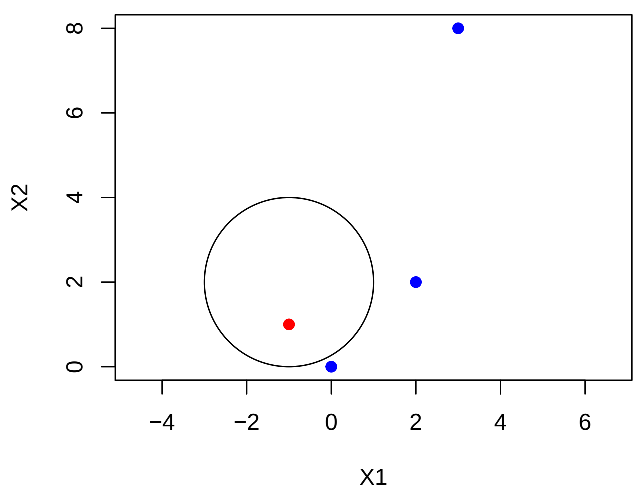
b) Again, feel free to do this by hand. A simple R solution could look like this:

```
plot(NA,NA, # initialize a plot
     type = "n", # does not produce any points or lines
     xlim = c(-4,2),ylim = c(-1,5), xlab = expression(X[1]), ylab = expression(X[2]),
     asp = 1)
symbols(c(-1), c(2), circles = c(2), add = TRUE, inches = FALSE)
text(c(-1), c(2), "< 4")
text(c(-4), c(2), "> 4")
```



c) You can do this by hand. Here we again use R and color the points according to the class they belong to:

```
plot(c(0, -1, 2, 3), c(0, 1, 2, 8), col = c("blue", "red", "blue", "blue"),
    type = "p", pch = 19, asp = 1, xlab = "X1", ylab = "X2")
symbols(c(-1), c(2), circles = c(2), add = TRUE, inches = FALSE)
```



d) Since equation

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

or

$$X_1^2 + X_2^2 + 2X_1 - 4X_2 + 1 = 0$$

includes quadratic terms, the decision boundary is not linear, though it's linear in terms of X_1^2, X_2^2, X_1, and X_2.

Problem 3

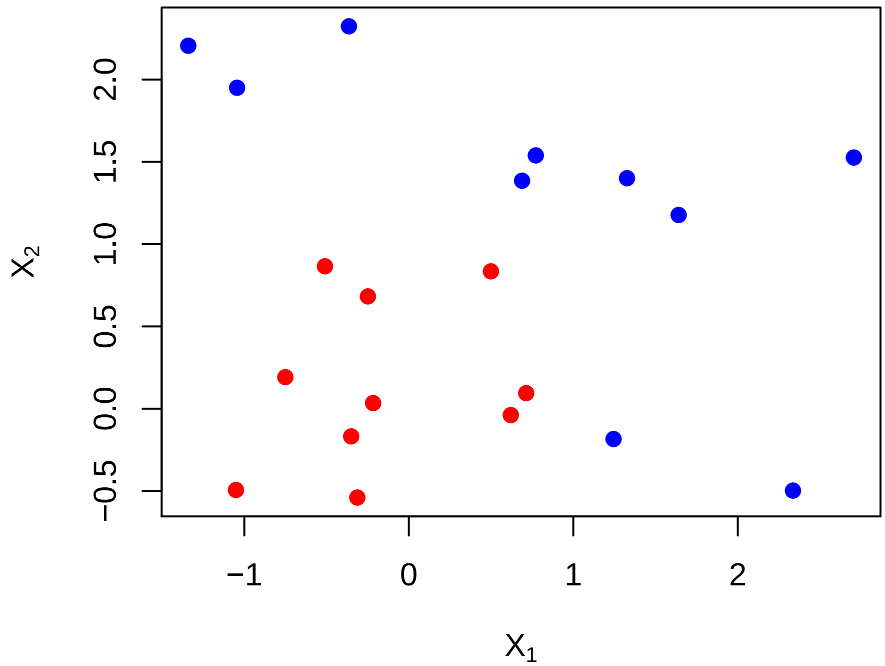
The input vector (X_1, X_2) is denoted by X. We can then expand the kernel as follows:

$$\begin{aligned}
K(X, X') &= (1 + \langle X, X' \rangle) \\
&= (1 + X_1X_1' + X_2X_2')^2 \\
&= 1 + 2X_1X_1' + 2X_2X_2' + (X_1X_1')^2 + (X_2X_2')^2 + 2X_1X_1'X_2X_2'.
\end{aligned}$$

We have that $h_1(X) = 1$, $h_2(X) = \sqrt{2}X_1$, $h_3(X) = \sqrt{2}X_2$, $h_4(X) = X_1^2$, $h_5(X) = X_2^2$, $h_6(X) = \sqrt{2}X_1X_2$, that is the the enlarged space created by the Kernel $K(X, X') = \langle h(X), h(X')\rangle$ has dimension $M = 6$.

Problem 4

```
# 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]))
```




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

(a)

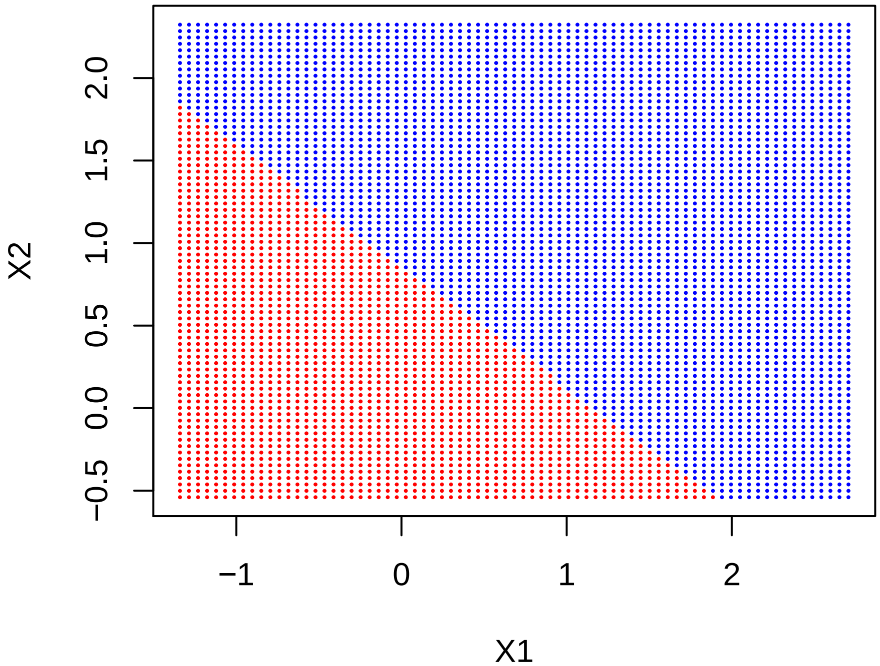
```
library(e1071)
svmfit = svm(y ~ ., data = dat, kernel = "linear", cost = 10, scale = FALSE)

# grid for plotting
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
}
```

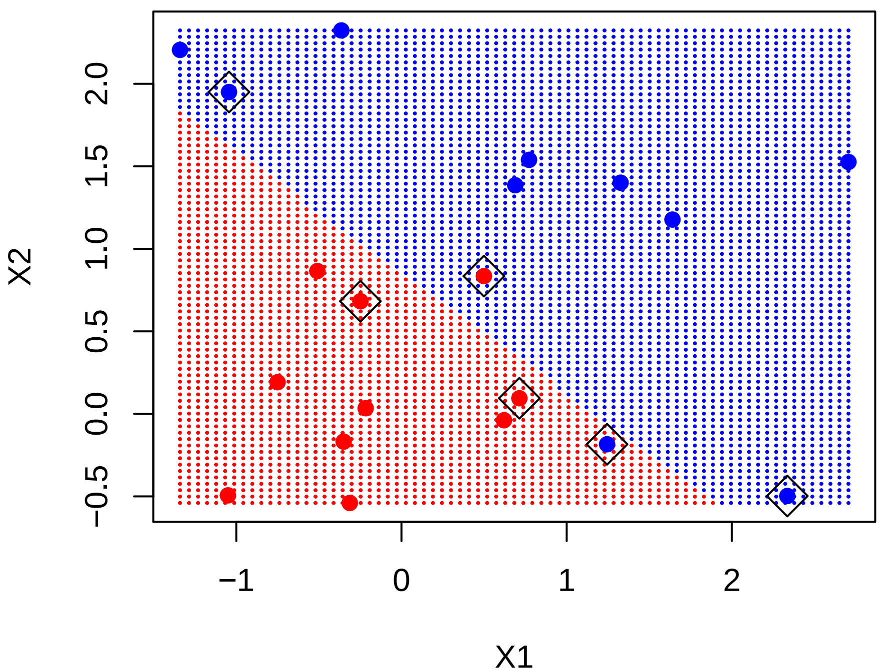
```
x = as.matrix(dat[, c("X1", "X2")])
xgrid = make.grid(x)
ygrid = predict(svmfit, xgrid)
```

```r
plot(xgrid, col = c("red", "blue")[as.numeric(ygrid)], pch = 20, cex = 0.2)
```



(b)

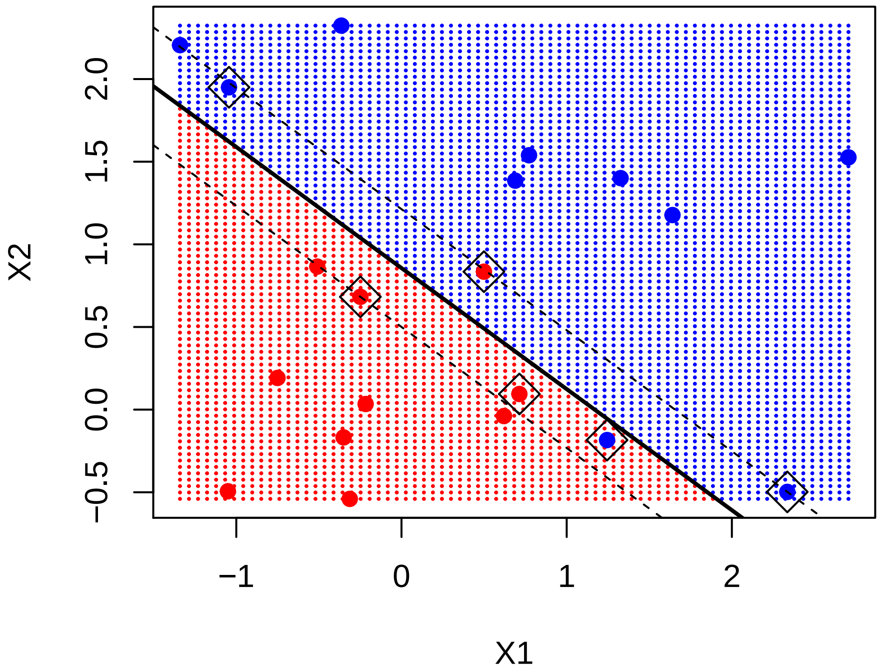
```r
plot(xgrid, col = c("red", "blue")[as.numeric(ygrid)], pch = 20, cex = 0.2)
points(x, col = y + 3, pch = 19)
points(x[svmfit$index, ], pch = 5, cex = 2)
```



(c)

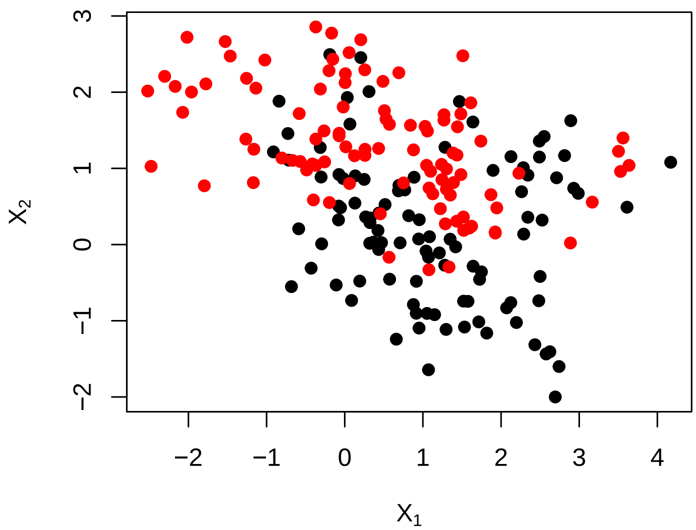
```r
beta = drop(t(svmfit$coefs) %*% x[svmfit$index, ])
beta0 = svmfit$rho
```

```r
plot(xgrid, col = c("red", "blue")[as.numeric(ygrid)], pch = 20, cex = 0.2)
points(x, col = y + 3, pch = 19)
points(x[svmfit$index, ], pch = 5, cex = 2)
abline(beta0/beta[2], -beta[1]/beta[2], lwd = 2)  #class boundary
abline((beta0 - 1)/beta[2], -beta[1]/beta[2], lty = 2)  #class boundary-margin
abline((beta0 + 1)/beta[2], -beta[1]/beta[2], lty = 2)  #class boundary+margin
```



Problem 5

```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)
```

```r
r.cv <- tune(svm, factor(y) ~ ., data = dat, kernel = "radial", ranges = list(cost = c(0.01,
    0.1, 1, 5, 10, 100, 1000), gamma = c(0.01, 0.1, 1, 10, 100)))
summary(r.cv)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost gamma
##     1    10
##
## - best performance: 0.165
##
## - Detailed performance results:
##     cost gamma error dispersion
## 1  1e-02 1e-02 0.525 0.14191155
## 2  1e-01 1e-02 0.525 0.14191155
## 3  1e+00 1e-02 0.285 0.08834906
## 4  5e+00 1e-02 0.320 0.06749486
## 5  1e+01 1e-02 0.315 0.07472171
## 6  1e+02 1e-02 0.310 0.07745967
## 7  1e+03 1e-02 0.295 0.07245688
## 8  1e-02 1e-01 0.525 0.14191155
## 9  1e-01 1e-01 0.305 0.07975657
## 10 1e+00 1e-01 0.325 0.07546154
## 11 5e+00 1e-01 0.295 0.07245688
## 12 1e+01 1e-01 0.290 0.06992059
## 13 1e+02 1e-01 0.290 0.06146363
## 14 1e+03 1e-01 0.215 0.04743416
## 15 1e-02 1e+00 0.535 0.12258784
## 16 1e-01 1e+00 0.285 0.06258328
## 17 1e+00 1e+00 0.205 0.04377975
## 18 5e+00 1e+00 0.175 0.03535534
```

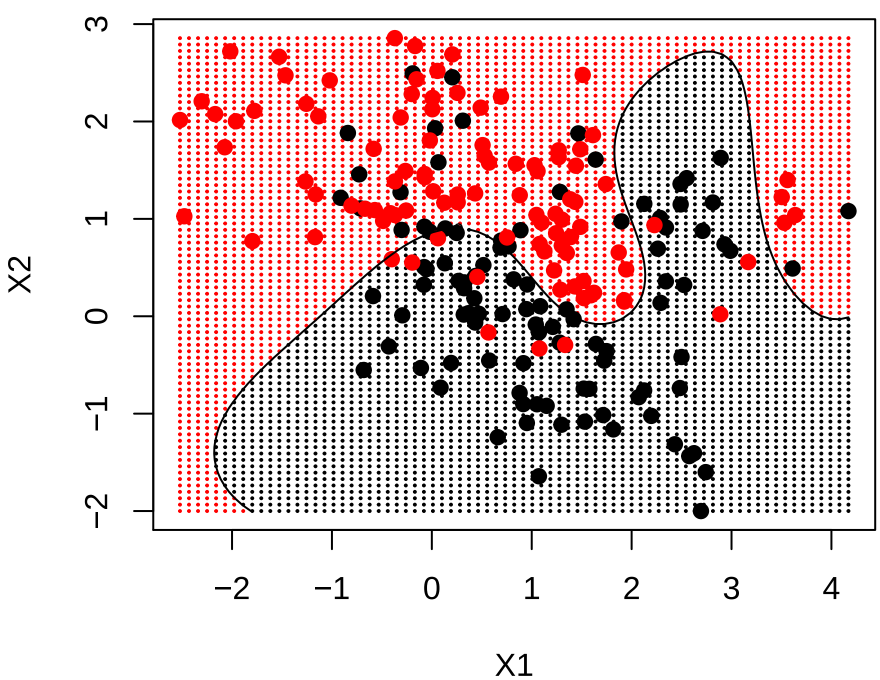
```
## 19 1e+01 1e+00 0.180 0.04830459
## 20 1e+02 1e+00 0.185 0.05797509
## 21 1e+03 1e+00 0.190 0.07378648
## 22 1e-02 1e+01 0.495 0.19923465
## 23 1e-01 1e+01 0.465 0.18715709
## 24 1e+00 1e+01 0.165 0.07472171
## 25 5e+00 1e+01 0.195 0.06851602
## 26 1e+01 1e+01 0.215 0.06258328
## 27 1e+02 1e+01 0.270 0.10327956
## 28 1e+03 1e+01 0.300 0.10540926
## 29 1e-02 1e+02 0.505 0.18173546
## 30 1e-01 1e+02 0.505 0.18173546
## 31 1e+00 1e+02 0.315 0.15284342
## 32 5e+00 1e+02 0.305 0.12122064
## 33 1e+01 1e+02 0.315 0.12258784
## 34 1e+02 1e+02 0.310 0.12202003
## 35 1e+03 1e+02 0.310 0.12202003
```

```
fit <- r.cv$best.model
```

Now we plot the non-linear decision boundary, and add the training points.

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

# decision boundary
func = predict(fit, 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 6

(a)

```r
library(ISLR)
data(OJ)
# head(OJ)
n = nrow(OJ)
set.seed(4268)
train = sample(1:n, 800)
OJ.train = OJ[train, ]
OJ.test = OJ[-train, ]
```

(b)

```r
library(e1071)
linear = svm(Purchase ~ ., data = OJ, subset = train, kernel = "linear",
    cost = 0.01)
summary(linear)
```

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

We have 431 Support vectors, where 217 belong to the class CH (Citrus Hill) and 214 belong to the class MM (Minute Maid Orange Juice).

(c)

```r
pred.train = predict(linear, OJ.train)
(ta = table(OJ.train$Purchase, pred.train))
```

```
##     pred.train
##       CH  MM
##   CH 431  56
##   MM  78 235
```

```r
msrate = 1 - sum(diag(ta))/sum(ta)
msrate
```

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

```r
pred.test = predict(linear, OJ.test)
(ta = table(OJ.test$Purchase, pred.test))
```

```
##     pred.test
##       CH  MM
##   CH 143  23
##   MM  25  79
```

```r
msrate = 1 - sum(diag(ta))/sum(ta)
msrate
```

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

(d)

```r
set.seed(4268)
cost.val = 10^seq(-2, 1, by = 0.25)
tune.cost = tune(svm, Purchase ~ ., data = OJ.train, kernel = "linear",
    ranges = list(cost = cost.val))
# summary(tune.cost)
```

(e)

```r
svm.linear = svm(Purchase ~ ., kernel = "linear", data = OJ.train, cost = tune.cost$best.parameter$cost)
train.pred = predict(svm.linear, OJ.train)
```

```r
(ta = table(OJ.train$Purchase, train.pred))
```

```
##     train.pred
##       CH  MM
##   CH 434  53
##   MM  73 240
```

```r
msrate.train.linear = 1 - sum(diag(ta))/sum(ta)
msrate.train.linear
```

```
## [1] 0.1575
```

```r
test.pred = predict(svm.linear, OJ.test)
(ta = table(OJ.test$Purchase, test.pred))
```

```
##     test.pred
##       CH  MM
##   CH 143  23
##   MM  23  81
```

```r
msrate.test.linear = 1 - sum(diag(ta))/sum(ta)
msrate.test.linear
```

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

(f) Radial Kernel Model

```r
svm.radial = svm(Purchase ~ ., kernel = "radial", data = OJ.train)
# summary(svm.radial)
```

Train and test error rate

```r
pred.train = predict(svm.radial, OJ.train)
(ta = table(OJ.train$Purchase, pred.train))
```

```
##     pred.train
##       CH  MM
##   CH 446  41
##   MM  72 241
```

```r
msrate.train.radial = 1 - sum(diag(ta))/sum(ta)
msrate.train.radial
```

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

```r
pred.test = predict(svm.radial, OJ.test)
(ta = table(OJ.test$Purchase, pred.test))
```

```
##     pred.test
##       CH  MM
##   CH 145  21
##   MM  25  79
```

```r
msrate.test.radial = 1 - sum(diag(ta))/sum(ta)
msrate.test.radial
```

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

Optimal cost

```r
set.seed(4268)
cost.val = 10^seq(-2, 1, by = 0.25)
tune.cost = tune(svm, Purchase ~ ., data = OJ.train, kernel = "radial",
    ranges = list(cost = cost.val))
# summary(tune.cost)
```

```r
svm.radial = svm(Purchase ~ ., kernel = "radial", data = OJ.train, cost = tune.cost$best.parameter$cost)
train.pred = predict(svm.radial, OJ.train)
```

Train and test error for optimal cost

```r
(ta = table(OJ.train$Purchase, train.pred))
```

```
##     train.pred
##       CH  MM
##   CH 450  37
##   MM  73 240
```

```r
msrate.train.linear = 1 - sum(diag(ta))/sum(ta)
msrate.train.linear
```

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

```r
test.pred = predict(svm.radial, OJ.test)
(ta = table(OJ.test$Purchase, test.pred))
```

```
##     test.pred
##       CH  MM
##   CH 146  20
##   MM  28  76
```

```r
msrate.test.linear = 1 - sum(diag(ta))/sum(ta)
msrate.test.linear
```

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

(g) Polynomial Kernel Model of degree 2

```r
svm.poly = svm(Purchase ~ ., kernel = "polynomial", degree = 2, data = OJ.train)
# summary(svm.poly)
```

Train and test error rate

```r
pred.train = predict(svm.poly, OJ.train)
(ta = table(OJ.train$Purchase, pred.train))
```

```
##     pred.train
##       CH  MM
##   CH 453  34
##   MM 109 204
```

```r
msrate.train.poly = 1 - sum(diag(ta))/sum(ta)
msrate.train.poly
```

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

```r
pred.test = predict(svm.poly, OJ.test)
(ta = table(OJ.test$Purchase, pred.test))
```

```
##     pred.test
##       CH  MM
##   CH 152  14
##   MM  33  71
```

```r
msrate.test.poly = 1 - sum(diag(ta))/sum(ta)
msrate.test.poly
```

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

Optimal cost

```r
set.seed(4268)
cost.val = 10^seq(-2, 1, by = 0.25)
tune.cost = tune(svm, Purchase ~ ., data = OJ.train, kernel = "poly",
    degree = 2, ranges = list(cost = cost.val))
# summary(tune.cost)
```

```r
svm.poly = svm(Purchase ~ ., kernel = "poly", degree = 2, data = OJ.train,
    cost = tune.cost$best.parameter$cost)
train.pred = predict(svm.poly, OJ.train)
```

Train and test error for optimal cost

```r
(ta = table(OJ.train$Purchase, train.pred))
```

```
##     train.pred
##       CH  MM
##   CH 452  35
##   MM  84 229
```

```r
msrate.train.poly = 1 - sum(diag(ta))/sum(ta)
msrate.train.poly
```

```
## [1] 0.14875
```

```r
test.pred = predict(svm.poly, OJ.test)
(ta = table(OJ.test$Purchase, test.pred))
```

```
##     test.pred
##       CH  MM
##   CH 148  18
##   MM  27  77
```

```r
msrate.test.poly = 1 - sum(diag(ta))/sum(ta)
msrate.test.poly
```

```
## [1] 0.1666667
```

(h) For the three choices of kernels and for the optimal cost we have

```r
msrate = cbind(c(msrate.train.linear, msrate.train.radial, msrate.train.poly),
    c(msrate.test.linear, msrate.test.radial, msrate.test.poly))
rownames(msrate) = c("linear", "radial", "polynomial")
colnames(msrate) = c("msrate.train", "msrate.test")
msrate
```

```
##            msrate.train msrate.test
## linear          0.13750   0.1777778
## radial          0.14125   0.1703704
## polynomial      0.14875   0.1666667
```