Chapter 9 Applied Exercises

Contents

Exercise 4

Generate a simulated two-class data set with 100 observations and two features in which there is a visible but non-linear separation between the two classes. Show that in this setting, a support vector machine with a polynomial kernel (with degree greater than 1) or a radial kernel will outperform a support vector classifier on the training data. Which technique performs best on the test data? Make plots and report training and test error rates in order to back up your assertions.

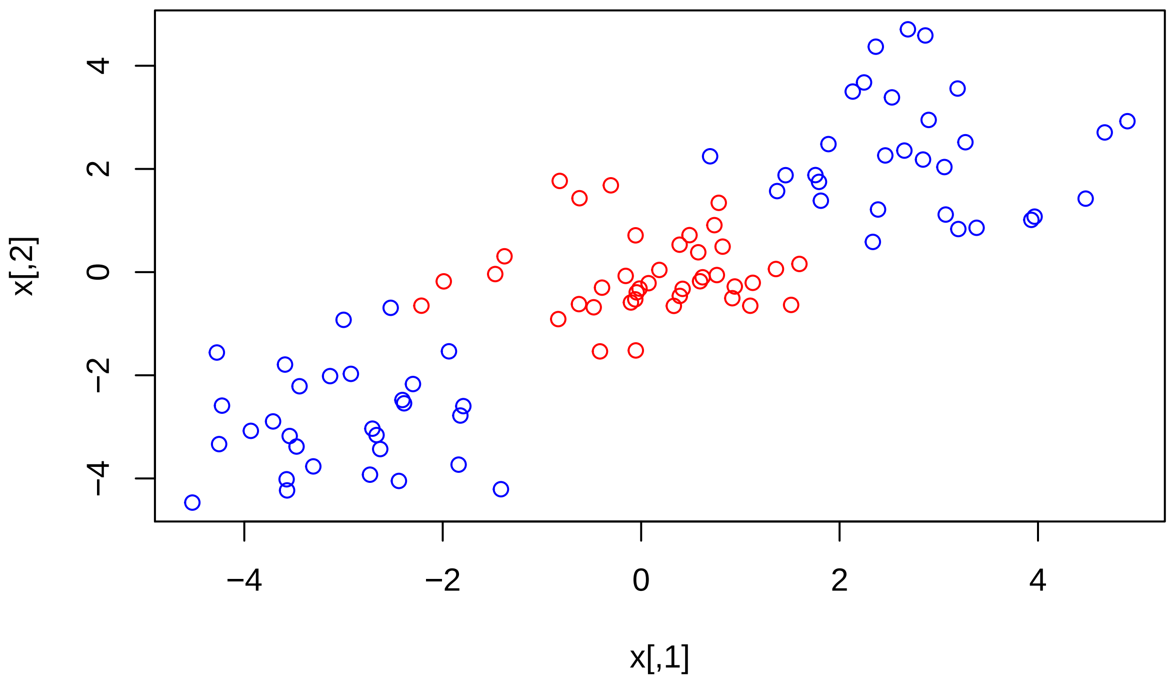
In order to generate non-linearly separable data, we follow the same strategy as the one in the labs. We first create the data, then shift one class to the extent that makes the classess separable. To make the data non-linearly separable, we divide one class to two groups. An alternative strategy is to produce random data and divide them to two groups according to a non-linear function. This strategy is pursued in Exercise 5.

```r
set.seed(1)
x <- matrix(rnorm(100*2), ncol = 2)
# shifting x corresponding to -1 to upright and down-left
x[41:70, ] <- x[41:70, ] + 2.5
x[71:100, ] <- x[71:100, ] -3

y <- rep(-1, length = 100)
y[1:40] <- 1

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

plot(x, col = 3 - y)
```



Let `cost = 1` in all models.
Linear kernel:

```r
library(e1071)
set.seed(1)
train <- sample(100, size = 60)
test <- -train
gam <- 1
deg <- 5
svmfit_lin <- svm(y ~ ., data = dat[train, ], cost = 1,
                  kernel = "linear")
```

```r
svmfit_poly <- svm(y ~ ., data = dat[train, ], cost = 1,
                   kernel = "polynomial", degree = deg)
svmfit_rad <- svm(y ~ ., data = dat[train, ], cost = 1,
                  kernel = "radial", gamma = gam)
```

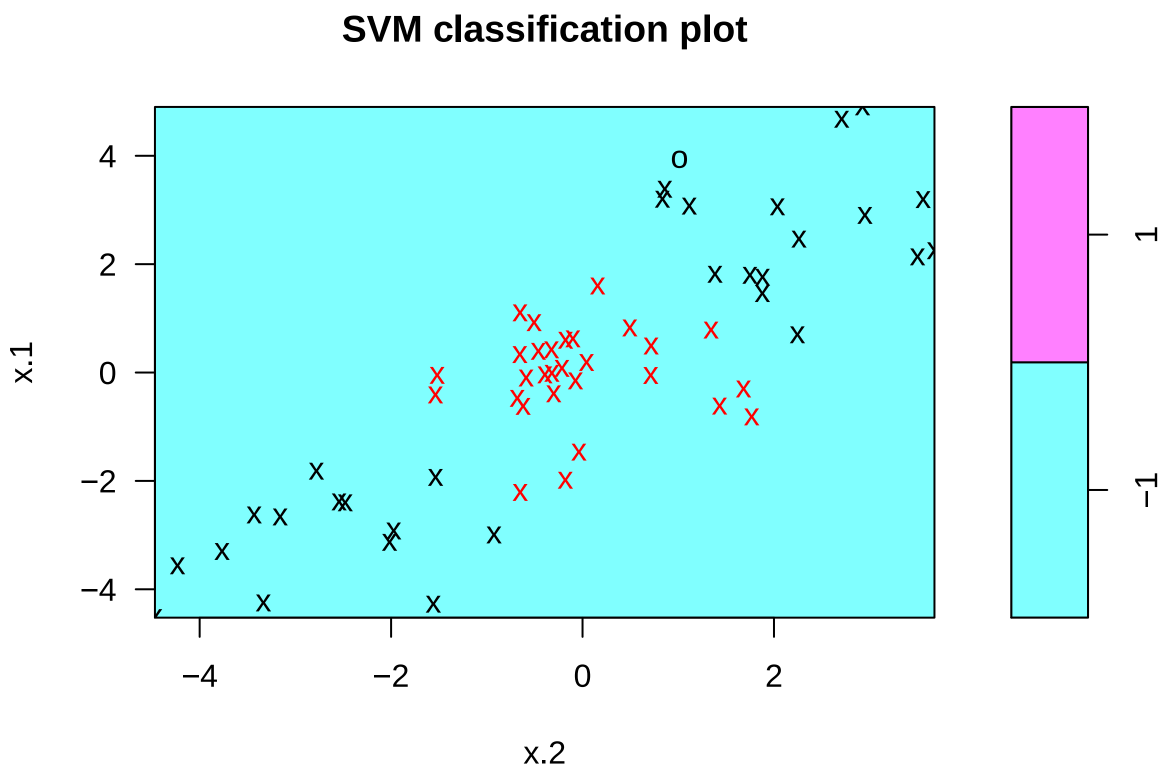
```r
# matrix of errors for different models:
error_mat <- matrix(NA, nrow = 2, ncol = 3, dimnames =
                      list(c("train", "test"), c("lin", "poly", "rad")))
## Linear kernel
pred_lin_train <- predict(svmfit_lin, newdata = dat[train, ])
error_mat["train", "lin"] <- mean(pred_lin_train != y[train])
pred_lin_test <- predict(svmfit_lin, newdata = dat[test, ])
error_mat["test", "lin"] <- mean(pred_lin_test != y[test])
## polynomial kernel
pred_poly_train <- predict(svmfit_poly, newdata = dat[train, ])
error_mat["train", "poly"] <- mean(pred_poly_train != y[train])
pred_poly_test <- predict(svmfit_poly, newdat = dat[test, ])
error_mat["test", "poly"] <- mean(pred_poly_test != y[test])
## radial kernel
pred_rad_train <- predict(svmfit_rad, newdata = dat[train, ])
error_mat["train", "rad"] <- mean(pred_rad_train != y[train])
pred_rad_test <- predict(svmfit_rad, newdata = dat[test, ])
error_mat["test", "rad"] <- mean(pred_rad_test != y[test])

error_mat
```

```
##             lin poly   rad
## train 0.4833333 0.35 0.000
## test  0.2750000 0.50 0.025
```

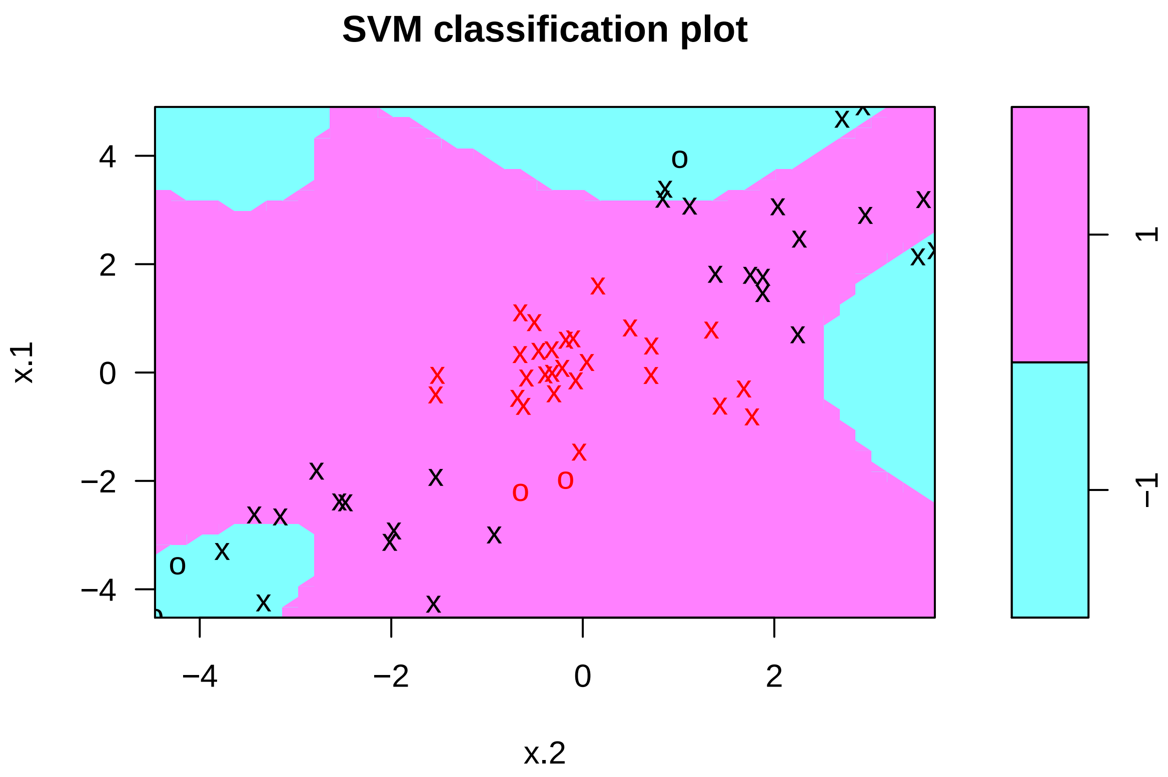
In the example above radial basis perform very well. For a complete assessment, however, we need to do cross-validation.

```r
plot(svmfit_lin, data = dat[train, ])
```

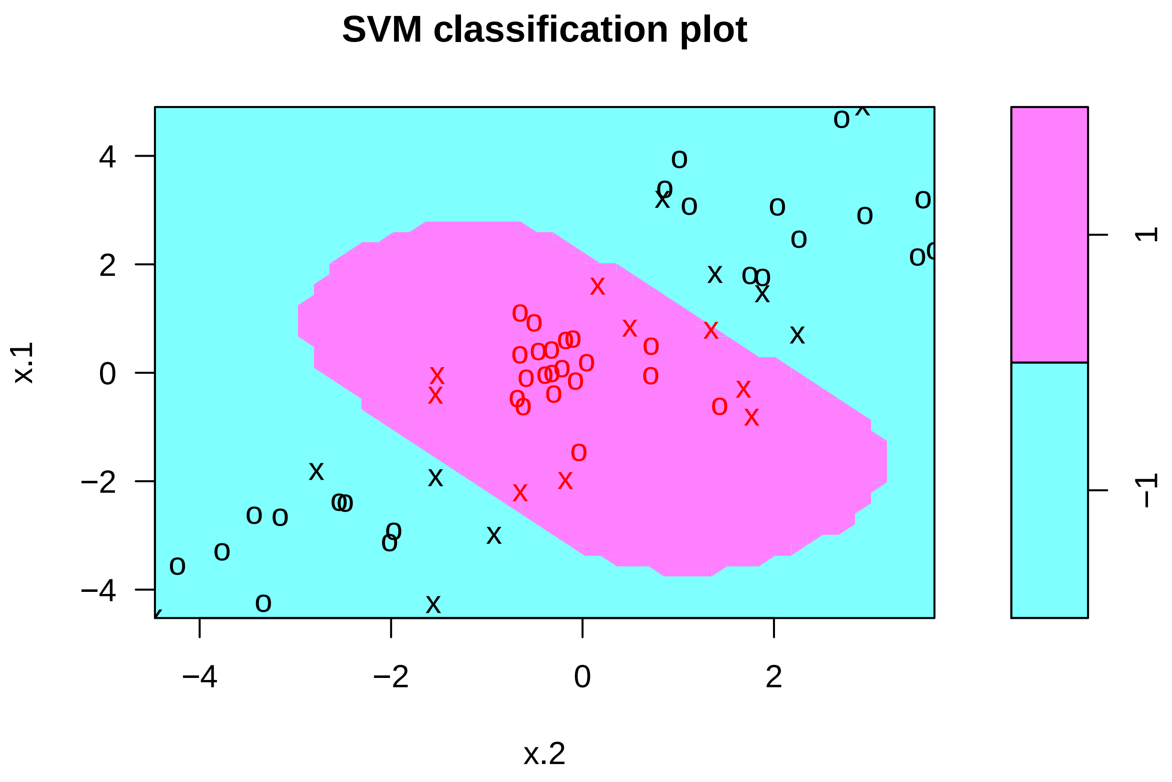


All the points are assigned to the class -1 in support vector classification.

```
plot(svmfit_poly, data = dat[train, ])
```



```r
plot(svmfit_rad, data = dat[train, ])
```



Exercise 5

We have seen that we can fit an SVM with a non-linear kernel in order to perform classification using a non-linear decision boundary. We will now see that we can also obtain a non-linear decision boundary by performing logistic regression using non-linear transformations of the features.

Part 5.a)

Generate a data set with n = 500 and p = 2, such that the observations belong to two classes with a quadratic decision boundary between them. For instance, you can do this as follows . . .

How many instances of the word "test" are there in this exercise? It appears that there are none! No use of the test data for evaluation in this question is directly related to the strategy suggested for generating the data: generating random data all at once, then assigning to classes according to their position with respect to a curve. But we should be very cautious not to generate the test data according to this strategy.
As to why we should take caution, consider producing both training and test data according to this strategy. Then both test and training data will be perfectly separable, and we get the non-generalizable result that as the margin approaches 0 (i.e. `cost` increases), the performance of the fit according to the test data does *not* diminish. The reason is that the variability in the test data generated using this method would not be correct, i.e. it would not reflect the variability of an independent test data set.

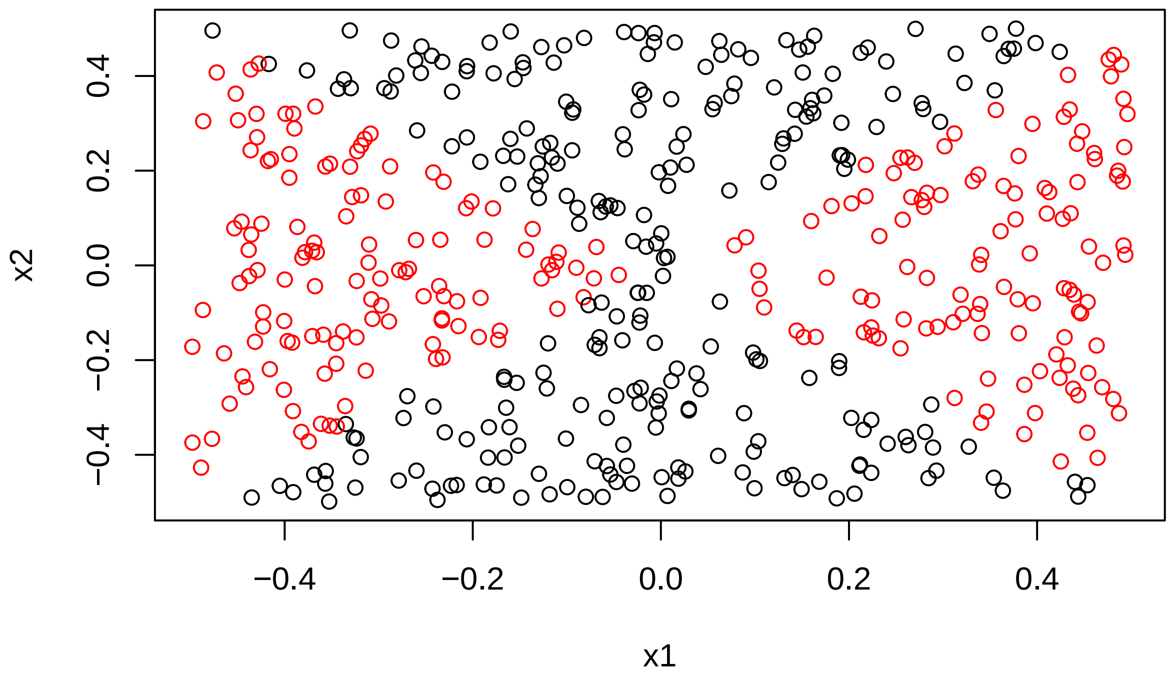
```
set.seed(1)
x1 <- runif(500) - 0.5
```

```r
x2 <- runif(500) - 0.5
y <- 1*(x1^2 - x2^2 > 0)
```

Part 5.b)

Plot the observations, colored according to their class labels. Your plot should display X_1 on the x-axis, and X_2 on the y-axis.

```r
plot(x1, x2, col = 1+y)
```



Part 5.c)

Fit a logistic regression model to the data, using X_1 and X_2 as predictors.

Whole data (no training and test division):

```r
dat <- data.frame(x1, x2, y)
glmfit_lin_all <- glm(y ~ ., data = dat, family = "binomial")
summary(glmfit_lin_all)
```

```
##
## Call:
## glm(formula = y ~ ., family = "binomial", data = dat)
##
## Deviance Residuals:
```

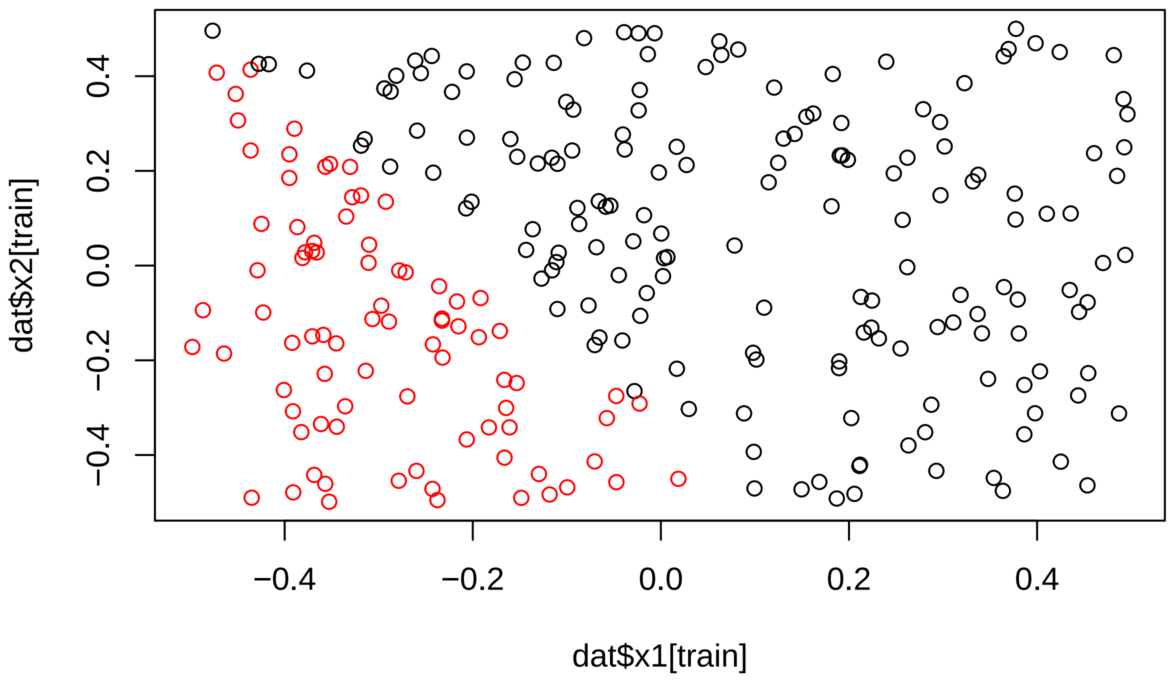
```
##    Min      1Q  Median      3Q     Max
## -1.179  -1.139  -1.112   1.206   1.257
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept) -0.087260   0.089579  -0.974    0.330
## x1           0.196199   0.316864   0.619    0.536
## x2          -0.002854   0.305712  -0.009    0.993
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 692.18  on 499  degrees of freedom
## Residual deviance: 691.79  on 497  degrees of freedom
## AIC: 697.79
##
## Number of Fisher Scoring iterations: 3
```

Part 5.d)

Apply this model to the training data in order to obtain a predicted class label for each training observation. Plot the observations, colored according to the predicted class labels. The decision boundary should be linear.

```r
set.seed(1)
train <- sample(500, size = 250)
glmfit_lin <- glm(y ~ ., data = dat[train, ], family = "binomial")
prob_lin <- predict(glmfit_lin, newdata = dat[train, ], type = "response")
pred_lin <- 1*(prob_lin > 0.5)
```

```r
plot(dat$x1[train], dat$x2[train], col = 1 + pred_lin)
```



The decision boundary is linear.

Part 5.e)

Now fit a logistic regression model to the data using non-linear functions of X_1 and X_2 as predictors.

First, note that direct log-transformation would result in deleting negative observations, so we avoid using it (although we could have used variations of it, such as $log(2 + x)$).

```
glmfit_nlin_all <- glm(y ~ poly(x1, 2) + poly(x2, 2) + x1:x2,
                       data = dat, family = "binomial")
```

```
## Warning: glm.fit: algorithm did not converge
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
summary(glmfit_nlin_all)
```

```
##
## Call:
## glm(formula = y ~ poly(x1, 2) + poly(x2, 2) + x1:x2, family = "binomial",
##     data = dat)
##
## Deviance Residuals:
##        Min          1Q      Median          3Q         Max
## -8.240e-04  -2.000e-08  -2.000e-08   2.000e-08   1.163e-03
##
## Coefficients:
```

```
##                Estimate Std. Error z value Pr(>|z|)
## (Intercept)      -102.2     4302.0  -0.024    0.981
## poly(x1, 2)1     2715.3   141109.5   0.019    0.985
## poly(x1, 2)2    27218.5   842987.2   0.032    0.974
## poly(x2, 2)1     -279.7    97160.4  -0.003    0.998
## poly(x2, 2)2  -28693.0   875451.3  -0.033    0.974
## x1:x2            -206.4    41802.8  -0.005    0.996
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 6.9218e+02  on 499  degrees of freedom
## Residual deviance: 3.5810e-06  on 494  degrees of freedom
## AIC: 12
##
## Number of Fisher Scoring iterations: 25
```

We get the warning "fitted probabilities numerically 0 or 1 occurred" because the classes can be perfectly separated using only one of the variables. Although this is a serious problem for inference, it may not be as much for prediction.

Part 5.f)

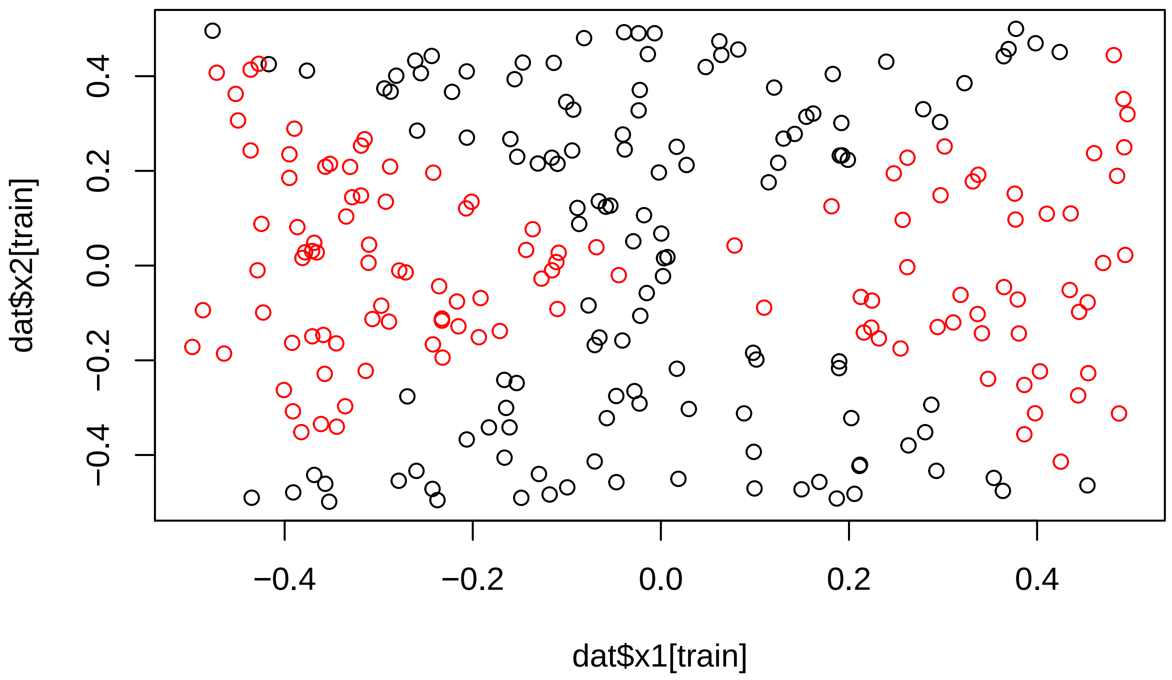
Apply this model to the training data in order to obtain a predicted class label for each training observation. Plot the observations, colored according to the predicted class labels. The decision boundary should be obviously non-linear. If it is not, then repeat (a)-(e) until you come up with an example in which the predicted class labels are obviously non-linear.

```
glmfit_nlin <- glm(y ~ poly(x1, 2) + poly(x2, 2) + x1:x2,
                   data = dat[train, ], family = "binomial")
```

```
## Warning: glm.fit: algorithm did not converge
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
prob_nlin_train <- predict(glmfit_nlin, newdata = dat[train, ], type = "response")
pred_nlin_train <- ifelse(prob_nlin_train > 0.5, 1, 0)
plot(dat$x1[train], dat$x2[train], col = pred_nlin_train + 1)
```



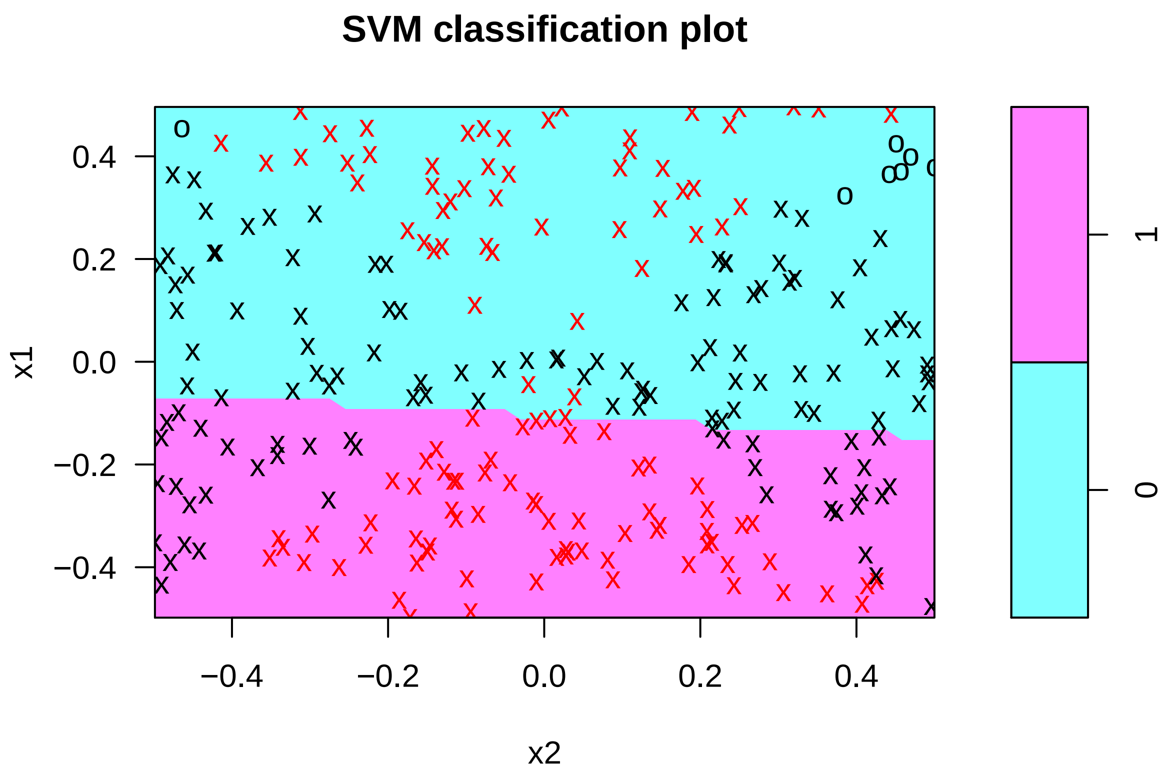
A look at how the training data are fit shows that the logistic regression with non-linear transformations performs well.

Part 5.g)

Fit a support vector classifier to the data with X1 and X2 as predictors. Obtain a class prediction for each training observation. Plot the observations, colored according to the predicted class labels.

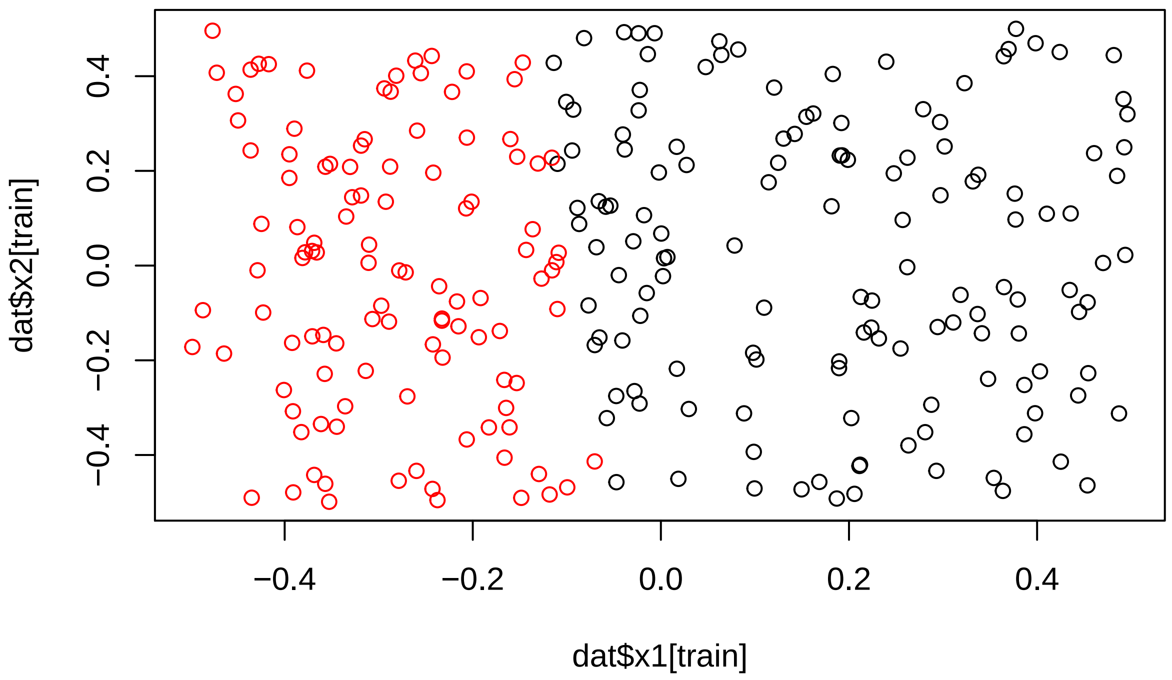
For intstructional purposes, we first draw a plot using `plot.svm()`.

```
dat$y <- as.factor(dat$y)
svmfit_lin <- svm(y ~ ., data = dat[train, ], cost = 1, kernel = "linear")
plot(svmfit_lin, data = dat[train, ])
```



It is important to make sure the response is a factor variable in `svm()` to be able to work with `plot()`. Otherwise no plot will be returned without any error message. On the other hand, for logistic regression it is more convenient to work with a numeric response to avoid any potential confusion in terms of how R codes the factor variables into dummies.

```
pred_lin <- predict(svmfit_lin, newdata = dat[train, ])
plot(dat$x1[train], dat$x2[train], col = pred_lin)
```

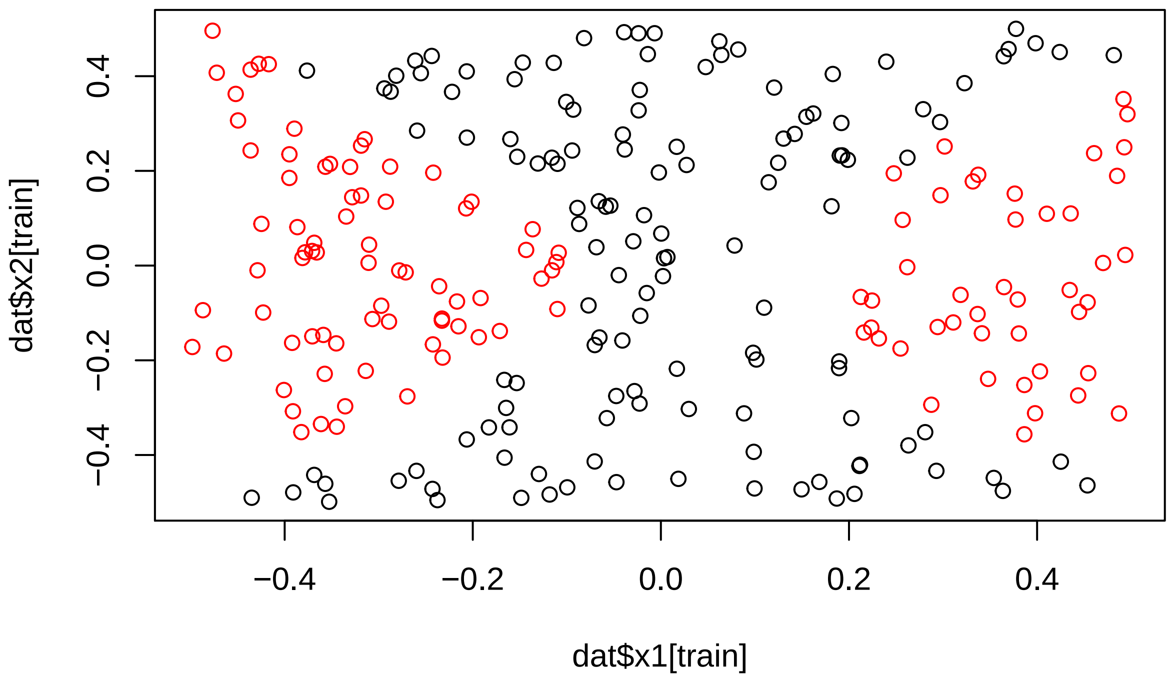


Since `pred_lin` is a string, it is coerced into numeric. On the other hand, if we put `col = pred_lin + 1`, we would get an error message. The correct syntax would be `col = as.numeric(pred_lin + 1)`.

Part 5.h)

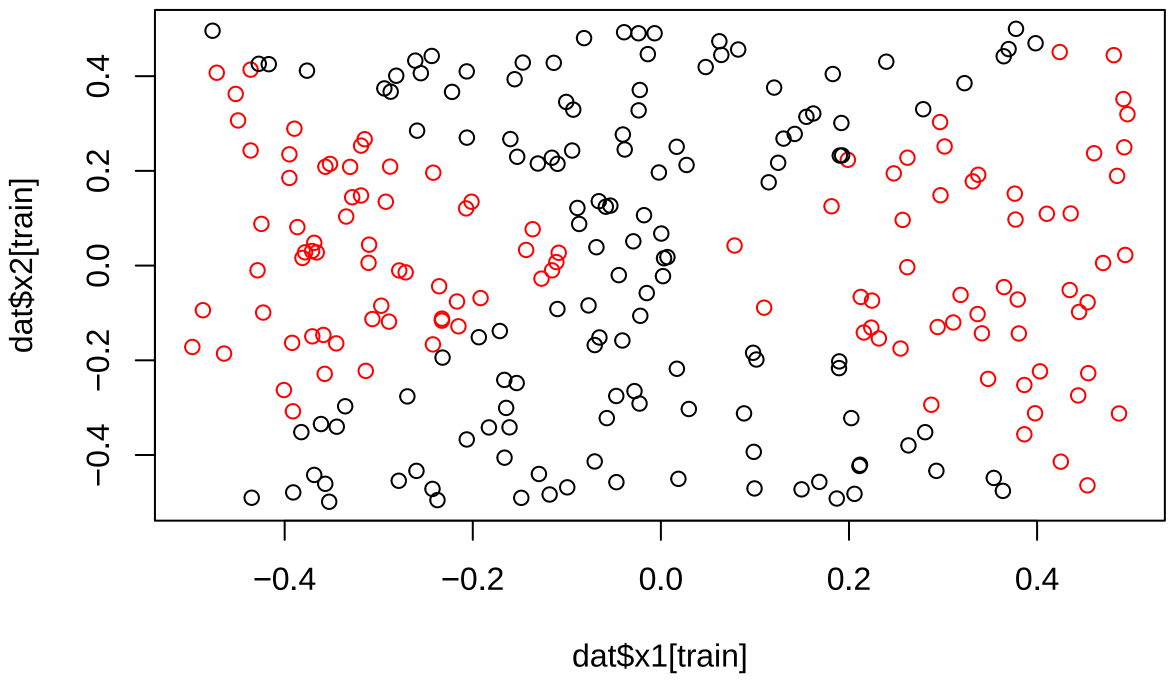
Fit a SVM using a non-linear kernel to the data. Obtain a class prediction for each training observation. Plot the observations, colored according to the predicted class labels.

```
svmfit_nlin <- svm(y ~ ., data = dat[train, ], cost = 1, kernel = "radial",
                   gamma = 1)
pred_nlin <- predict(svmfit_nlin, newdata = dat[train, ])
plot(dat$x1[train], dat$x2[train], col = as.numeric(pred_nlin))
```



The radial kernel perform reasonably well for values of `gamma` between 0.1 and 100. What can we say about polynomial kernel?

```r
svmfit_nlin <- svm(y ~ ., data = dat[train, ], cost = 1, kernel = "polynomial",
                   degree = 2)
pred_nlin <- predict(svmfit_nlin, newdata = dat[train, ])
plot(dat$x1[train], dat$x2[train], col = as.numeric(pred_nlin))
```



Polynomial kernel of degree 2 works very well, but if you change it to any higher degree it performs much worse. This highlights the importance of cross-validation in finding the optimal parameter when we use kernels, especially for polynomial kernels in this example.

Part 5.i)

Comment on your results.

Logistic regression can perform a good job fitting non-linear classification problems if we use non-linear transfromations of the variables. However, at the form implemented in this question, SVM has the advantage of being done automatically (without the need to explicitly define the transformations) and control over flexibility (through the penalty term, i.e. the cost parameter). Nevertheless, a logistic kernel regression that uses logistic loss instead of hinge loss would have both of these advantages. In addition, it is more natural to use kernel logistic regression in cases where costs of false positive and false negative are different, since it is a probability model. The drawback of logistic kernel regression is that it is more computationally costly than SVM (for instance, in SVM non-support-vector observations play zero role in the solution, while that is not the case for kernel logistic regression).

Exercise 6

At the end of Section 9.6.1, it is claimed that in the case of data that is just barely linearly separable, a support vector classifier with a small value of cost that misclassifies a couple of training observations may perform better on test data than one with a huge value of cost that does not misclassify any training observations. You will now investigate this claim.

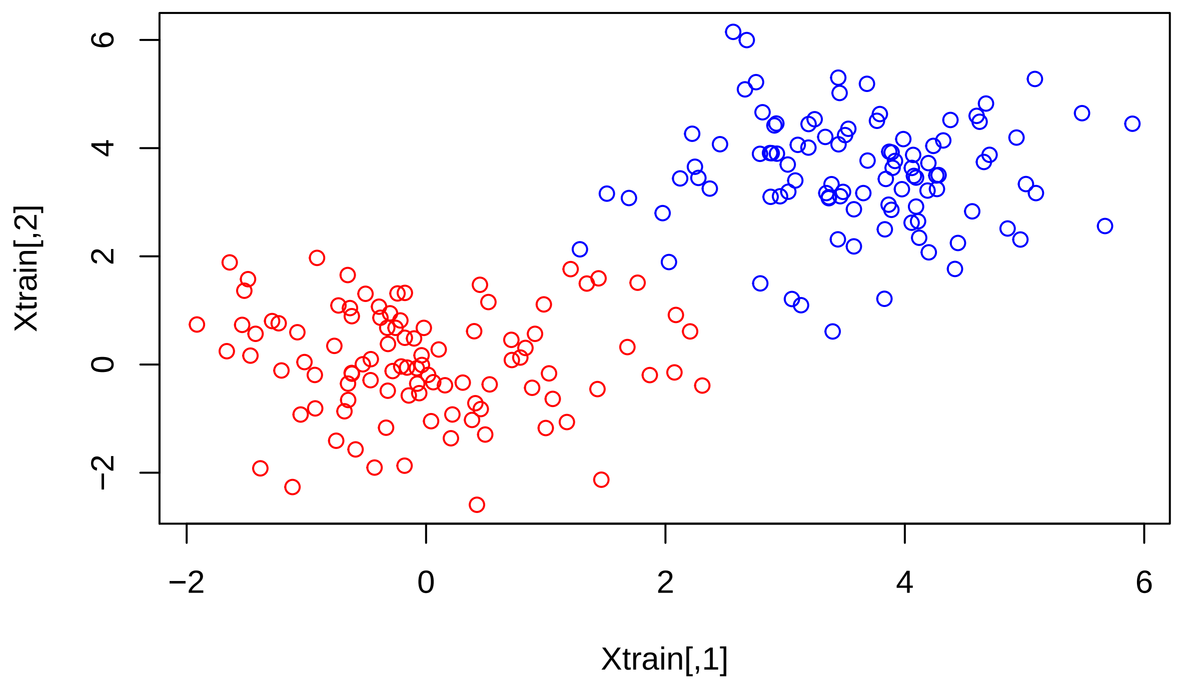
Part a)

Generate two-class data with $p = 2$ in such a way that the classes are just barely linearly separable.

What strategy should we follow to generate the data? See Part (a) of Exercise 5 to see why we could not first generate the data and then split it according to a line. We follow the strategy for generating random data that was suggeted in the chapter's lab.
Note that for generating random data, we want a distribution that does not have many points close the boundary; so we use normal distribution. If we used uniform distribution, cross-validation would most likely imply that using very small margins would perform well (To see why, you can try barely separated uniformly distributed classes and draw separating lines using random subsets of the observations: they lines would look very similar; hence, increasing the penalty would not lead to little, if not zero, change in the error rates).
We first generate the training data:

```
rm(list = ls())
set.seed(1)
Xtrain <- matrix(rnorm(200*2), ncol = 2)
ytrain <- c(rep(1, length = 100), rep(-1, length = 100))
Xtrain[ytrain == 1, ] = Xtrain[ytrain == 1, ] + 3.5

plot(Xtrain, col = ytrain + 3)
```



Part b)

Compute the cross-validation error rates for support vector classifiers with a range of cost values. How many training ~~errors~~ observations are misclassified for each value of cost considered, and how does this relate to the *cross-validation errors obtained?*

```r
# number of misclassified training [note we don't need test data]
set.seed(1)
dat_train <- data.frame(x = Xtrain, y = as.factor(ytrain))
cost_grid <- 10^(seq(-2, 10))
Num_mis <- rep(NA, length = length(cost_grid))
Svmfit <- vector(mode = "list", length = length(cost_grid))
for (i in 1:length(cost_grid)) {
  Svmfit[[i]] <- svm(y ~ ., data = dat_train, cost = cost_grid[i],
                     kernel = "linear")  # we store fits to use in validation set
  pred_train <- predict(Svmfit[[i]], newdata = dat_train)
  Num_mis[i] <- sum(pred_train != dat_train$y)
}
Num_mis
```

```
##  [1] 1 1 1 1 0 0 0 0 0 0 0 0 0
```

```r
set.seed(1)
tune.out <- tune(svm, y ~ ., data = dat_train, kernel = "linear",
                  ranges = list(cost = cost_grid))
summary(tune.out)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##  0.01
##
## - best performance: 0.005
##
## - Detailed performance results:
##     cost error dispersion
## 1  1e-02 0.005 0.01581139
## 2  1e-01 0.010 0.02108185
## 3  1e+00 0.005 0.01581139
## 4  1e+01 0.005 0.01581139
## 5  1e+02 0.010 0.02108185
## 6  1e+03 0.010 0.02108185
## 7  1e+04 0.010 0.02108185
## 8  1e+05 0.010 0.02108185
## 9  1e+06 0.010 0.02108185
## 10 1e+07 0.010 0.02108185
## 11 1e+08 0.010 0.02108185
## 12 1e+09 0.010 0.02108185
## 13 1e+10 0.010 0.02108185
```

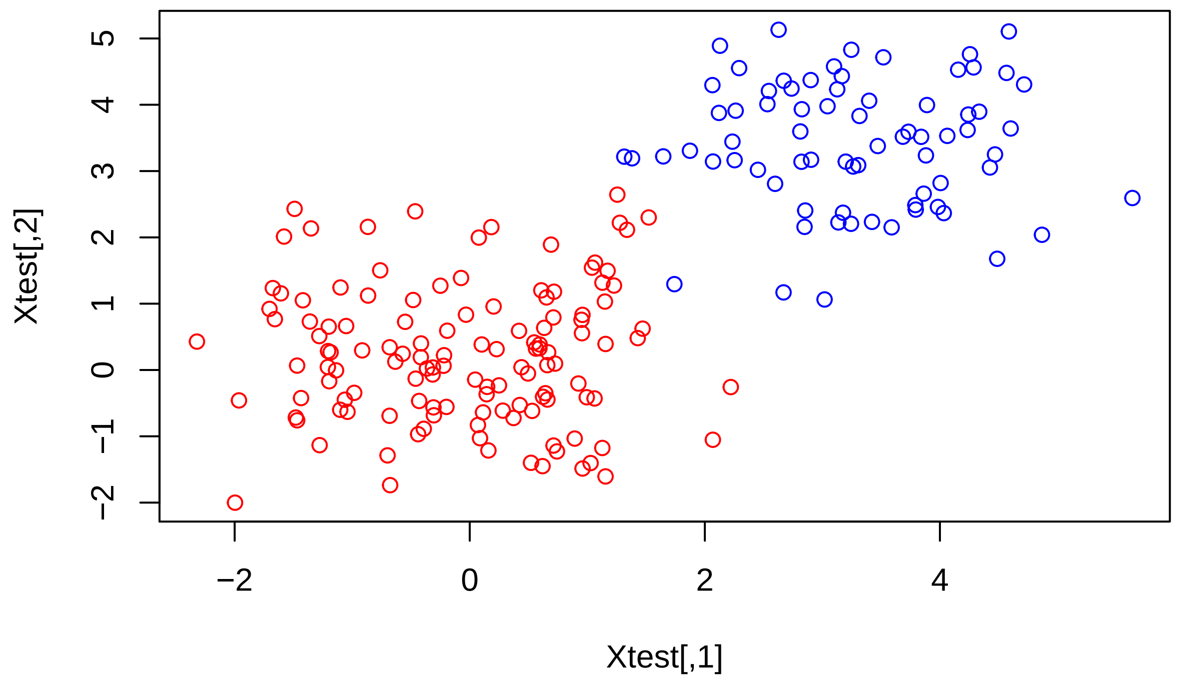
```r
# training miscalssified
```

While the number of misclassified observations falls when the cost of violating the margin rises, that is not the case for cross-validation error.

Part c)

Generate an appropriate test data set, and compute the test errors corresponding to each of the values of cost considered. Which value of cost leads to the fewest test errors, and how does this compare to the values of cost that yield the fewest training errors and the fewest cross-validation errors?

```r
set.seed(10)  # should use a different seed from the one used for training data
Xtest <- matrix(rnorm(200*2), ncol = 2)
# we can have any number of observations in either classes in the test set
ytest <- c(rep(1, length = 73), rep(-1, length = 200 - 73))
# All matters is to have the Xtest correspond correctly to classes
Xtest[ytest == 1, ] = Xtest[ytest == 1, ] + 3.5
plot(Xtest, col = ytest + 3)
```



```r
dat_test <- data.frame(x = Xtest, y = as.factor(ytrain))  # same naming as for train data
test_err <- rep(NA, length = length(cost_grid))
for (i in 1:length(cost_grid)) {
  pred_test <- predict(Svmfit[[i]], newdata = dat_test)
  test_err[i] <- mean(pred_test != dat_test$y)
}
test_err
```

```
##  [1] 0.140 0.145 0.140 0.140 0.150 0.150 0.150 0.150 0.150 0.150 0.150
## [12] 0.150 0.150
```

The test error rates correspond closely to cross-validation errors, but not to the training error rates.

Part d)

Discuss your results.

The claim stated above is correct in our case, and would be correct in general. Note, however, that in this cases where we simulate the data and choose the training data according to a criterion, the cross-validation error might not reflect the variation in the test error. For example, if we choose the training data in a way that many training observations align close to the decision boundary on the correct side, the cross-validation error might decrease with the cost, unlike the test error rate (see the discussion about using uniform distribution in Part (a) above). The reason is that the training data in this case is not random (since we have chosen it in a way that classes are barely separable), hence cannot be used to proxy for the test data in cross-validation. One way to alleviate this issue is to find an amount of shifting (in the variables belonging to one class) that makes the data barely separable most of the times, then gernerate randomly both training and test data according to that amount of shifting. But since this does not solve the problem completely, the upshot is that we should take cross-validation errors in such simulation cases with a grain of salt, and rely on the test error rates.

Exercise 7

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.

Part 7.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.

```r
rm(list = ls())
library(ISLR)
dim(Auto)
```

```
## [1] 392   9
```

```r
summary(Auto)
```

```
##       mpg          cylinders      displacement     horsepower
##  Min.   : 9.00   Min.   :3.000   Min.   : 68.0   Min.   : 46.0
##  1st Qu.:17.00   1st Qu.:4.000   1st Qu.:105.0   1st Qu.: 75.0
##  Median :22.75   Median :4.000   Median :151.0   Median : 93.5
##  Mean   :23.45   Mean   :5.472   Mean   :194.4   Mean   :104.5
##  3rd Qu.:29.00   3rd Qu.:8.000   3rd Qu.:275.8   3rd Qu.:126.0
##  Max.   :46.60   Max.   :8.000   Max.   :455.0   Max.   :230.0
##
##      weight      acceleration        year           origin
##  Min.   :1613   Min.   : 8.00   Min.   :70.00   Min.   :1.000
##  1st Qu.:2225   1st Qu.:13.78   1st Qu.:73.00   1st Qu.:1.000
##  Median :2804   Median :15.50   Median :76.00   Median :1.000
##  Mean   :2978   Mean   :15.54   Mean   :75.98   Mean   :1.577
##  3rd Qu.:3615   3rd Qu.:17.02   3rd Qu.:79.00   3rd Qu.:2.000
##  Max.   :5140   Max.   :24.80   Max.   :82.00   Max.   :3.000
##
##                  name
##  amc matador       :  5
##  ford pinto        :  5
##  toyota corolla    :  5
##  amc gremlin       :  4
##  amc hornet        :  4
##  chevrolet chevette:  4
##  (Other)           :365
```

```
# response
summary(Auto$mpg)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    9.00   17.00   22.75   23.45   29.00   46.60
```

```
# variables
sum(is.na(Auto))
```

```
## [1] 0
```

```
# define dummy
Auto$mpg_fac <- as.factor(1*(Auto$mpg > median(Auto$mpg)))
table(Auto$mpg_fac)
```

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

Part 7.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.

It appears that this code takes a rather long time to run, so we consider a less refined grid. However, in unreported runs, we arrived at the same optimal error rate using a more refined grid (`cost_grid <- 10^(seq(8, -2))`)

```
library(e1071)
set.seed(1)
cost_grid <- 10^(seq(6, -1, by = -2))
tune_lin <- tune(svm, mpg_fac ~ . - mpg - name, data = Auto,
                   ranges = list(cost = cost_grid), kernel = "linear")
summary(tune_lin)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##   cost
##  10000
##
## - best performance: 0.08679487
##
## - Detailed performance results:
##    cost      error dispersion
## 1 1e+06 0.16551282 0.16233715
## 2 1e+04 0.08679487 0.05567394
## 3 1e+02 0.08679487 0.05567394
## 4 1e+00 0.09192308 0.05827672
```

It takes a long time to run the chunk above, since the underlying optimization does not converge in some cases. That is why we get the warning “reaching max number of iterations” which means that the quadratic programming problem that is used to solve for support vector classification has reached its maximum number of iterations. As a result, the chunk takes a long time to run and the solution to some of the problems solved in class-validation may be sub-optimal. `e1071::tune()` does not allow for changing the maximum number of itertions; hence, we should use an alternative library if we want to make sure we reach the optimum. For the purpose of this question, we will ignore the warnings and take the errors at their face value.
The cross-validation errors can be seen above and are minimized for values of cost equal to 100 or 10000.

Part 7.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.

Radial

```r
library(e1071)
set.seed(1)
tune_rad <- tune(svm, mpg_fac ~ . - mpg - name, data = Auto,
                 ranges = list(cost = 10^seq(4, -2), gamma = c(0.1, 0.5, 1, 2, 3)))
summary(tune_rad)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost gamma
##     1     1
##
## - best performance: 0.07391026
##
## - Detailed performance results:
##     cost gamma      error dispersion
## 1  1e+04   0.1 0.11217949 0.07645931
## 2  1e+03   0.1 0.10448718 0.05699964
## 3  1e+02   0.1 0.08666667 0.05680772
## 4  1e+01   0.1 0.08673077 0.05562295
## 5  1e+00   0.1 0.08929487 0.05827503
## 6  1e-01   0.1 0.09179487 0.05812971
## 7  1e-02   0.1 0.15576923 0.06126259
## 8  1e+04   0.5 0.10448718 0.06072287
## 9  1e+03   0.5 0.10448718 0.06072287
## 10 1e+02   0.5 0.10448718 0.06862934
## 11 1e+01   0.5 0.07897436 0.04567335
## 12 1e+00   0.5 0.07391026 0.05315495
## 13 1e-01   0.5 0.08923077 0.05559893
## 14 1e-02   0.5 0.56115385 0.04344202
## 15 1e+04   1.0 0.10698718 0.07386765
## 16 1e+03   1.0 0.10698718 0.07386765
## 17 1e+02   1.0 0.10698718 0.07386765
## 18 1e+01   1.0 0.08916667 0.05799733
## 19 1e+00   1.0 0.07391026 0.05176240
## 20 1e-01   1.0 0.08923077 0.06408390
## 21 1e-02   1.0 0.56115385 0.04344202
## 22 1e+04   2.0 0.09923077 0.07799865
## 23 1e+03   2.0 0.09923077 0.07799865
## 24 1e+02   2.0 0.09923077 0.07799865
## 25 1e+01   2.0 0.09423077 0.07220895
## 26 1e+00   2.0 0.07634615 0.05879482
## 27 1e-01   2.0 0.13512821 0.05778941
## 28 1e-02   2.0 0.56115385 0.04344202
## 29 1e+04   3.0 0.09923077 0.07316609
## 30 1e+03   3.0 0.09923077 0.07316609
## 31 1e+02   3.0 0.09923077 0.07316609
## 32 1e+01   3.0 0.09673077 0.06968463
## 33 1e+00   3.0 0.07634615 0.05867200
## 34 1e-01   3.0 0.30128205 0.11716574
## 35 1e-02   3.0 0.56115385 0.04344202
```

Using radial kernel yields a lower error rate, which suggests that the decision boundary is non-linear. Interestingly, the radial-kernel SVM takes much less time than the linear-kernel SVM and avoids warnings. Also note that the optimal cost paramter (in the radial kernel space) is smaller that the one we found for linear kernel.

Polynomial

```
set.seed(1)
tune_pol <- tune(svm, mpg_fac ~ . - name - mpg, data = Auto,
                 ranges = list(cost = 10^seq(4, -2), degree = c(2, 3, 4, 5)))
summary(tune_pol)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost degree
##    10      2
##
## - best performance: 0.07897436
##
## - Detailed performance results:
##     cost degree      error dispersion
## 1  1e+04      2 0.13006410 0.07957699
## 2  1e+03      2 0.10448718 0.06423064
## 3  1e+02      2 0.09935897 0.05557281
## 4  1e+01      2 0.07897436 0.04724571
## 5  1e+00      2 0.08673077 0.05562295
## 6  1e-01      2 0.09179487 0.05812971
## 7  1e-02      2 0.16602564 0.05349652
## 8  1e+04      3 0.13006410 0.07957699
## 9  1e+03      3 0.10448718 0.06423064
## 10 1e+02      3 0.09935897 0.05557281
## 11 1e+01      3 0.07897436 0.04724571
## 12 1e+00      3 0.08673077 0.05562295
## 13 1e-01      3 0.09179487 0.05812971
## 14 1e-02      3 0.16602564 0.05349652
## 15 1e+04      4 0.13006410 0.07957699
## 16 1e+03      4 0.10448718 0.06423064
## 17 1e+02      4 0.09935897 0.05557281
## 18 1e+01      4 0.07897436 0.04724571
## 19 1e+00      4 0.08673077 0.05562295
## 20 1e-01      4 0.09179487 0.05812971
## 21 1e-02      4 0.16602564 0.05349652
## 22 1e+04      5 0.13006410 0.07957699
## 23 1e+03      5 0.10448718 0.06423064
## 24 1e+02      5 0.09935897 0.05557281
## 25 1e+01      5 0.07897436 0.04724571
## 26 1e+00      5 0.08673077 0.05562295
## 27 1e-01      5 0.09179487 0.05812971
## 28 1e-02      5 0.16602564 0.05349652
```

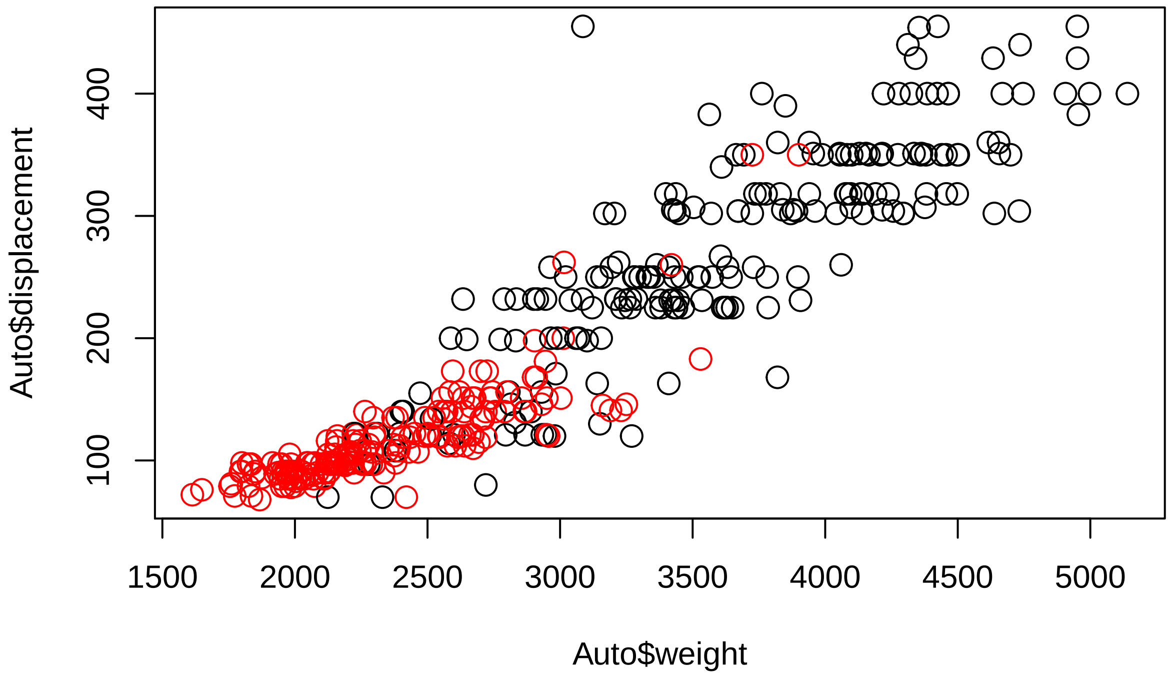
Polynomial and radial kernels both outperform the linear kernel, but not by a very large margin. This suggests that the decision boundary is non-linear, but the non-linearity is not extreme.

Part 7.d)

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

Let's first look at the actual response values. Since we have many variables, we cannot see the how linear the decision boundary is by focusin on only two variables at a time. But it is useful to look at the two-dimensional plots to get an idea of how the data looks like and how the classifier works.
We introduce noise to the discrete variables using `jitter()` to make the plots more readable.

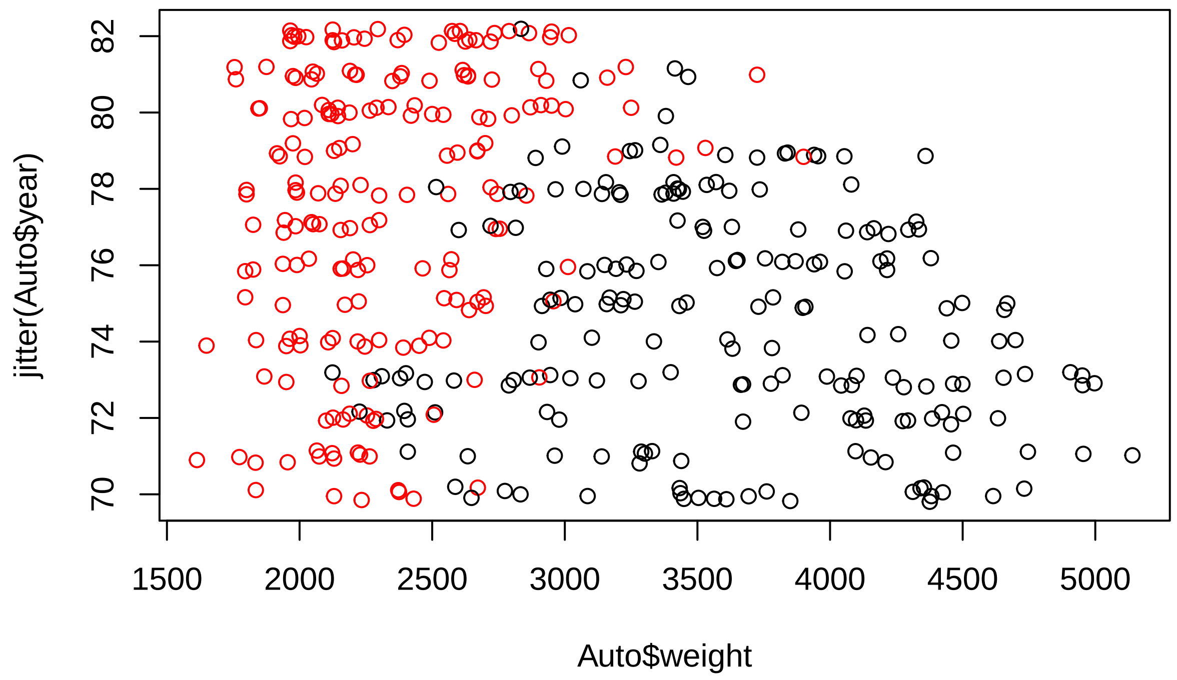
```r
set.seed(1)
plot(Auto$weight, Auto$displacement, col = Auto$mpg_fac, cex = 1.5)
```



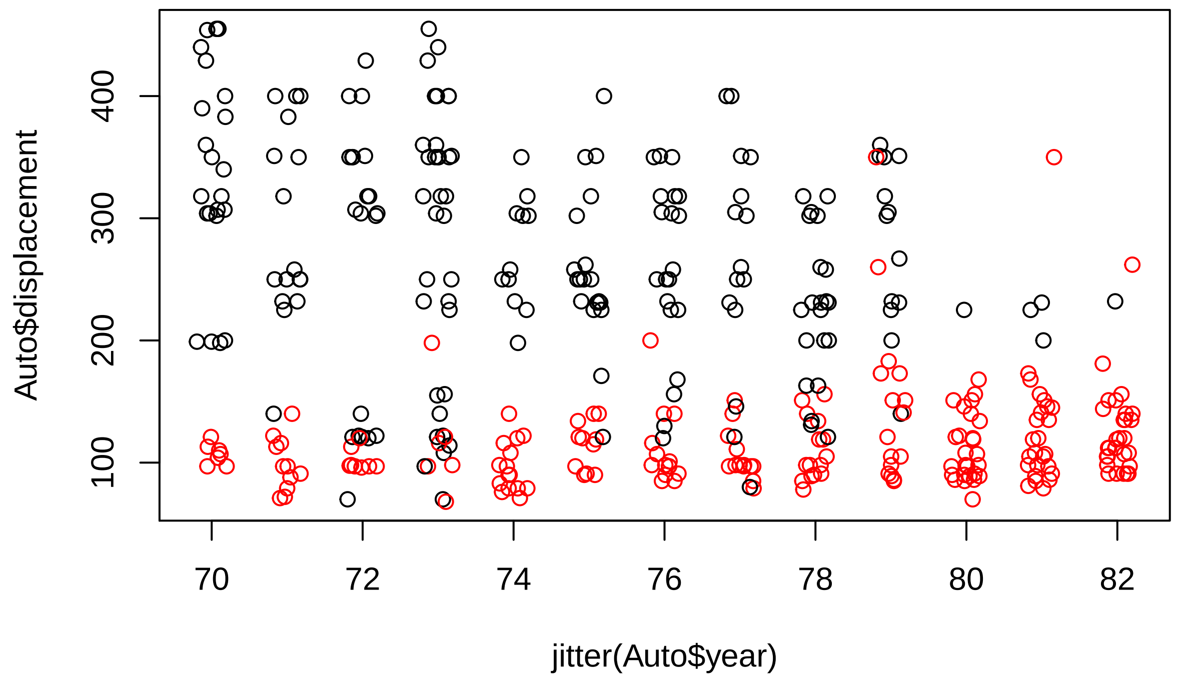
A linear classifier might have a difficult time prdicting the response value for some of the cars with weight lower than 3500 that have very small displacement (unless other variables are quite informative about these cars' gas mileage; this can be further explored graphically using different aesthetics from the library `ggplot2`). They are depicted in black in the plot and denote some low-weight fuel-inefficeint cars.

Also consider a number of other plots:

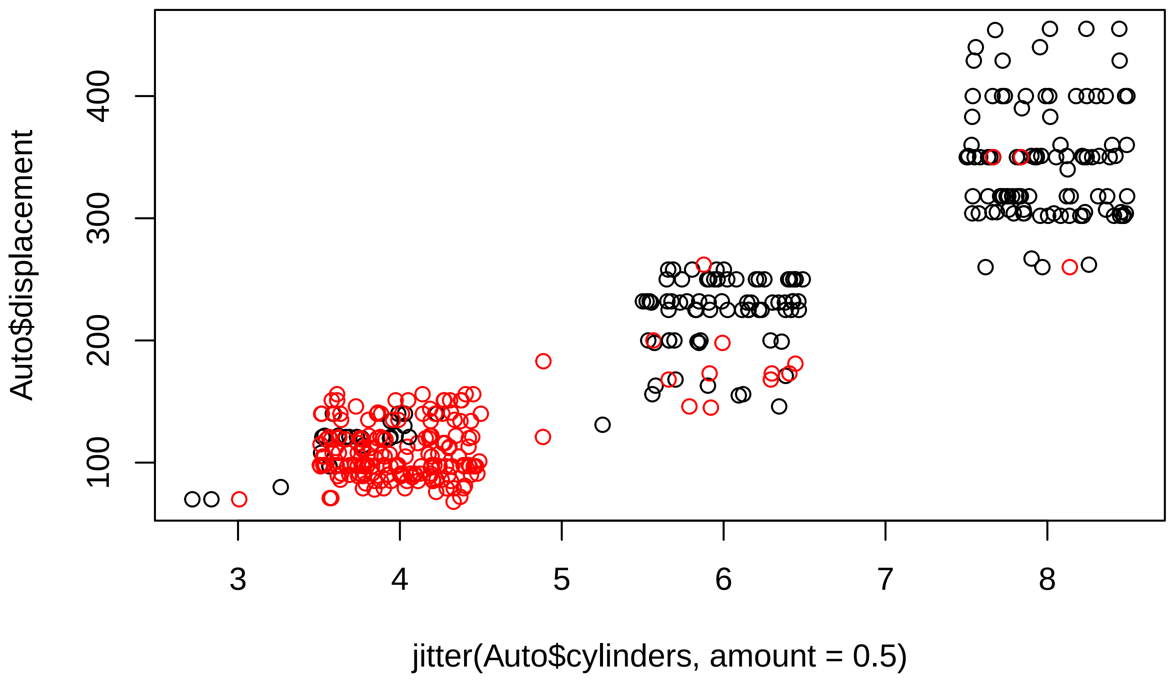
```r
plot(Auto$weight, jitter(Auto$year), col = Auto$mpg_fac)
```



```r
plot(jitter(Auto$year), Auto$displacement, col = Auto$mpg_fac)
```

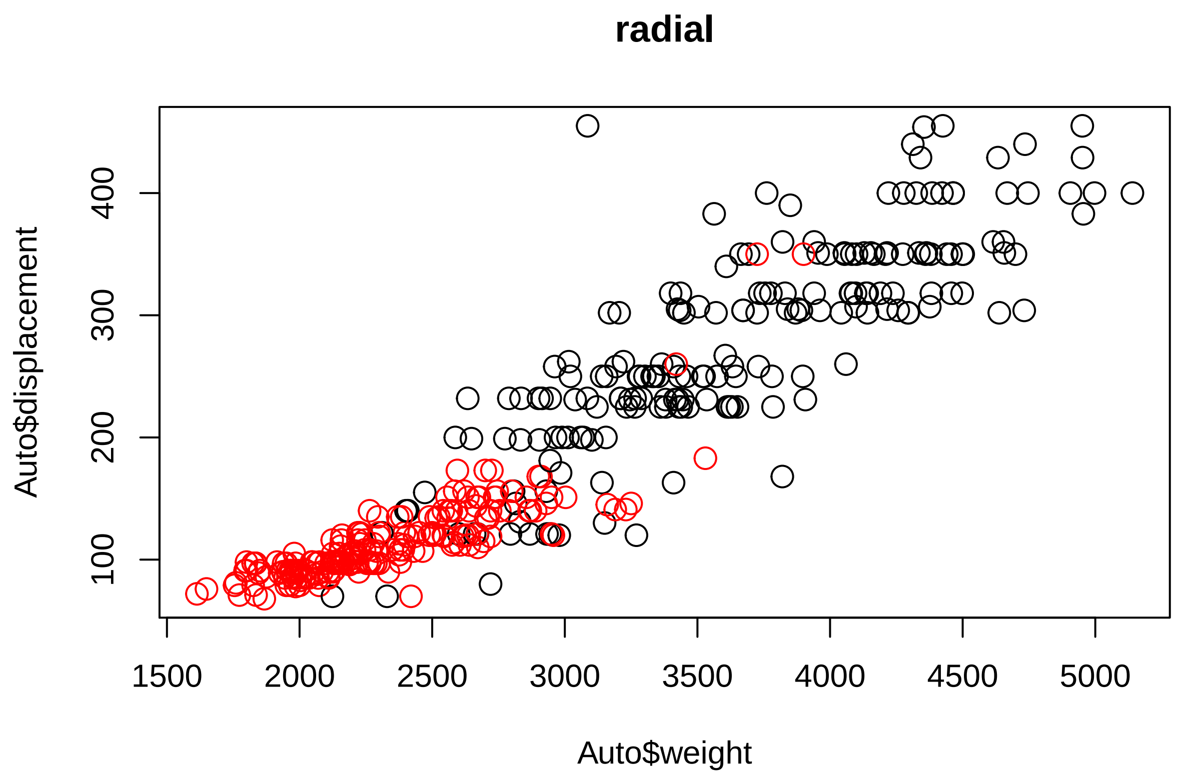


```
plot(jitter(Auto$cylinders, amount = 0.5), Auto$displacement, col = Auto$mpg_fac)
```

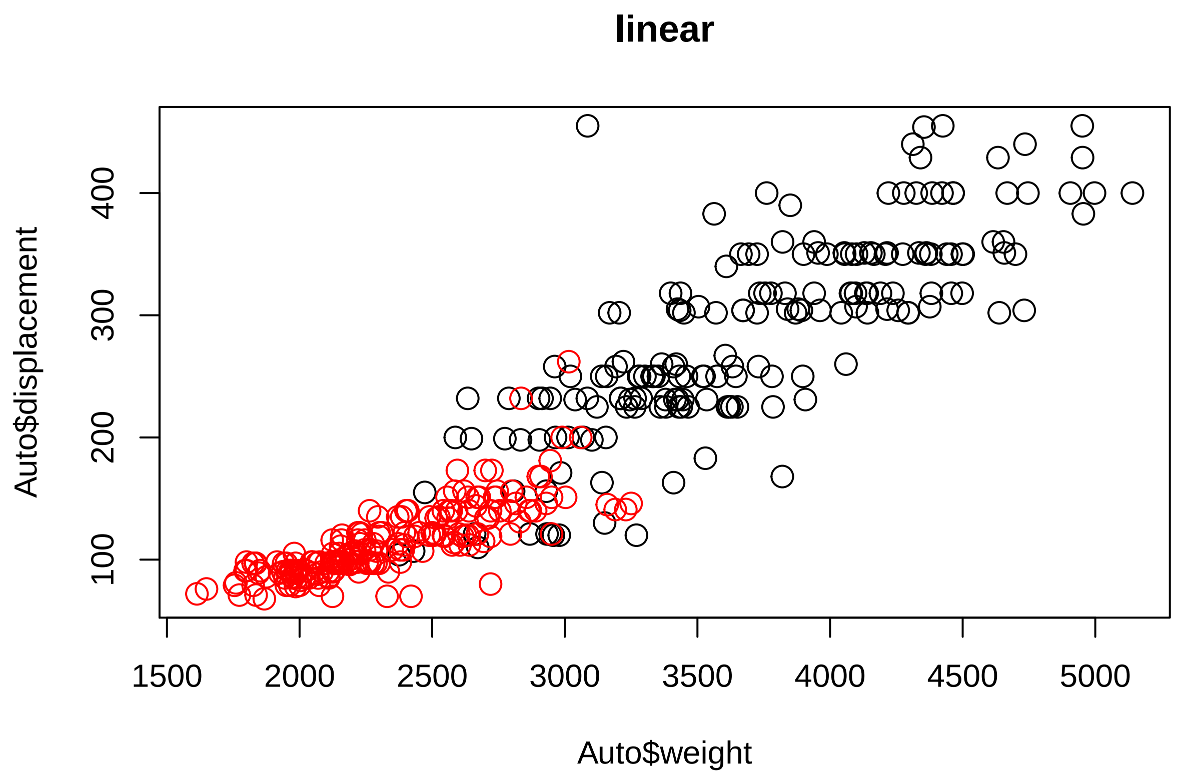


Now cosider the predicted response in the best model with radial basis. We focus on `weight` and `displacement`, which we know are important predictors from previous chapters (can be verified using logistic regression or tree-based methods, e.g. looking at partial dependence plots).

```
pred_rad <- predict(tune_rad$best.model, newdata = Auto)
plot(Auto$weight, Auto$displacement, col = pred_rad, cex = 1.5, main = "radial")
```



```r
pred_lin <- predict(tune_lin$best.model, newdata = Auto)
plot(Auto$weight, Auto$displacement, col = pred_lin, cex = 1.5, main = "linear")
```



A comparison of the predicted responsed in the plots above with the actual responses shows how radial kernel is better able to capture the black marks in the heavily-red-marked areas and red marks in the heavily-black-marked areas of the figure.

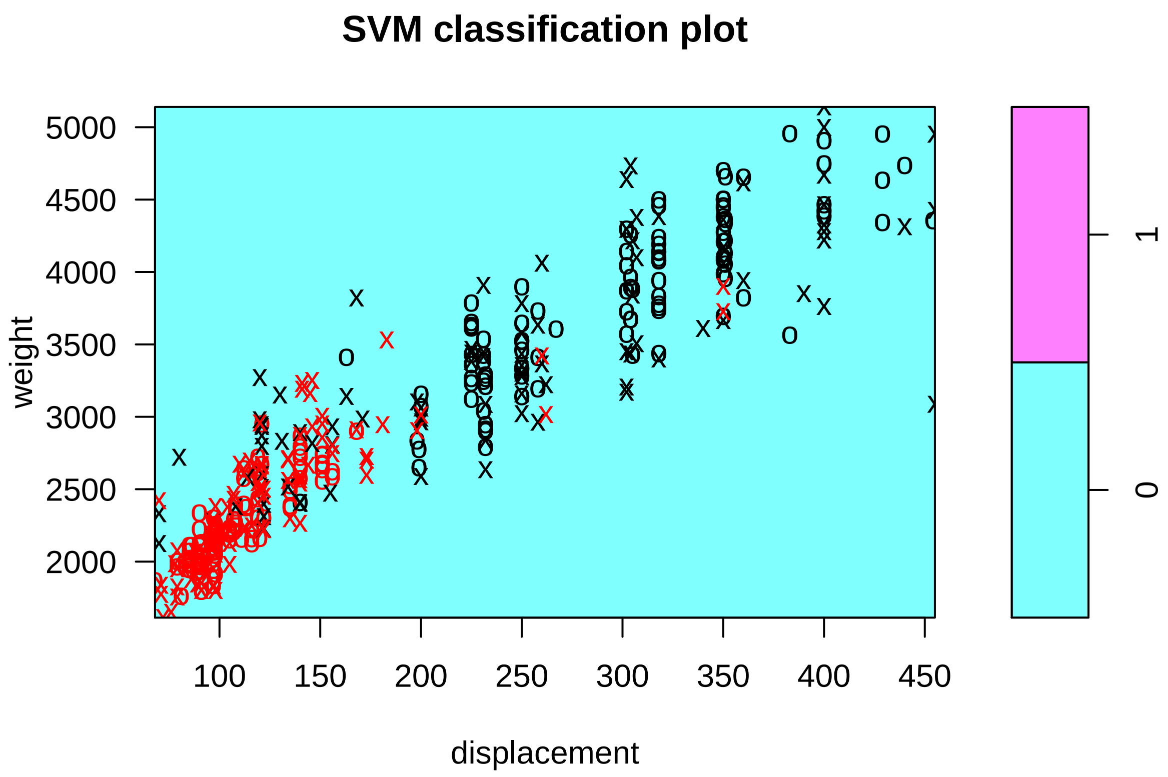
Using `plot.svm()`

Now let's try using `plot.svm`, which can show the support vectors on the figures:

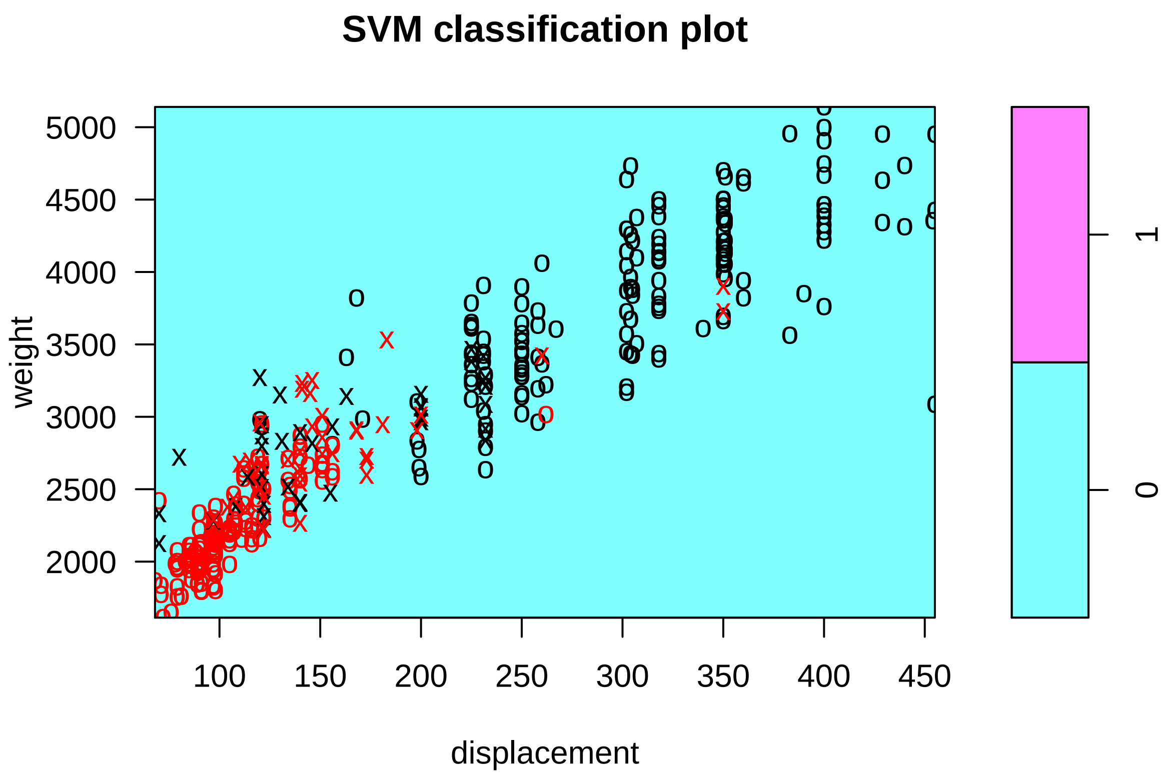
```r
try(plot(tune_lin$best.model, data = Auto, weight ~ displacement, main = "linear"))
```

The reason for the error above is that the formula `mpg_fac ~ . - mpg - name` which is used by `tune()` turns out not to be recognized by `plot.svm`, because of the negative sign. So we rewrite the formula to be able to draw the figures:

```r
library(e1071)
set.seed(1)
form <- (mpg_fac ~ cylinders + displacement + horsepower +
                   weight + acceleration + year + origin)
tune_rad <- tune(svm, form, data = Auto,
                 ranges = list(cost = 10^seq(4, -2), gamma = c(0.1, 0.5, 1, 2, 3)))
plot(tune_rad$best.model, data = Auto, weight ~ displacement, main = "radial kernel")
```



```r
set.seed(1)
form <- (mpg_fac ~ cylinders + displacement + horsepower +
                   weight + acceleration + year + origin)
svmfit <- svm(form, data = Auto, cost = 10000, kernel = "linear")
plot(svmfit, data = Auto, weight ~ displacement, main = "linear")
```



In the two plots above, the mark colors are not informative since they represent the actual response values. The only information we can use to compare these two plots is the position of support vectors, marked by crosses. It turns out that using radial kernel leads to more support vectors at the areas where we suspect a linear kernel might underperform.

Exercise 8

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

```r
library(ISLR)
dim(OJ)
```

```
## [1] 1070   18
```

```r
summary(OJ)
```

```
##  Purchase WeekofPurchase     StoreID        PriceCH         PriceMM     
##  CH:653   Min.   :227.0   Min.   :1.00   Min.   :1.690   Min.   :1.690  
##  MM:417   1st Qu.:240.0   1st Qu.:2.00   1st Qu.:1.790   1st Qu.:1.990  
##           Median :257.0   Median :3.00   Median :1.860   Median :2.090  
##           Mean   :254.4   Mean   :3.96   Mean   :1.867   Mean   :2.085  
##           3rd Qu.:268.0   3rd Qu.:7.00   3rd Qu.:1.990   3rd Qu.:2.180  
##           Max.   :278.0   Max.   :7.00   Max.   :2.090   Max.   :2.290  
##      DiscCH             DiscMM         SpecialCH        SpecialMM     
##  Min.   :0.00000   Min.   :0.0000   Min.   :0.0000   Min.   :0.0000  
##  1st Qu.:0.00000   1st Qu.:0.0000   1st Qu.:0.0000   1st Qu.:0.0000  
##  Median :0.00000   Median :0.0000   Median :0.0000   Median :0.0000
##  Mean   :0.05186   Mean   :0.1234   Mean   :0.1477   Mean   :0.1617
##  3rd Qu.:0.00000   3rd Qu.:0.2300   3rd Qu.:0.0000   3rd Qu.:0.0000
##  Max.   :0.50000   Max.   :0.8000   Max.   :1.0000   Max.   :1.0000
##     LoyalCH          SalePriceMM     SalePriceCH      PriceDiff
##  Min.   :0.000011   Min.   :1.190   Min.   :1.390   Min.   :-0.6700
##  1st Qu.:0.325257   1st Qu.:1.690   1st Qu.:1.750   1st Qu.: 0.0000
##  Median :0.600000   Median :2.090   Median :1.860   Median : 0.2300
##  Mean   :0.565782   Mean   :1.962   Mean   :1.816   Mean   : 0.1465
##  3rd Qu.:0.850873   3rd Qu.:2.130   3rd Qu.:1.890   3rd Qu.: 0.3200
##  Max.   :0.999947   Max.   :2.290   Max.   :2.090   Max.   : 0.6400
##  Store7      PctDiscMM        PctDiscCH       ListPriceDiff
##  No :714   Min.   :0.0000   Min.   :0.00000   Min.   :0.000
##  Yes:356   1st Qu.:0.0000   1st Qu.:0.00000   1st Qu.:0.140
##            Median :0.0000   Median :0.00000   Median :0.240
##            Mean   :0.0593   Mean   :0.02731   Mean   :0.218
##            3rd Qu.:0.1127   3rd Qu.:0.00000   3rd Qu.:0.300
##            Max.   :0.4020   Max.   :0.25269   Max.   :0.440
##      STORE
##  Min.   :0.000
##  1st Qu.:0.000
##  Median :2.000
##  Mean   :1.631
##  3rd Qu.:3.000
##  Max.   :4.000
```

```r
# response
summary(OJ$Purchase)
```

```
##  CH  MM
## 653 417
```

```r
contrasts(OJ$Purchase)
```

```
##    MM
## CH  0
## MM  1
```

```r
# variables
sum(is.na(OJ))
```

```
## [1] 0
```

Part 8.a)

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

```r
set.seed(1)
train <- sample(nrow(OJ), size = 800)
test <- -train
```

Part 8.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.

```r
library(e1071)
svm_cost01 <- svm(Purchase ~ ., data = OJ[train, ], cost = 0.01,
                  kernel = "linear")
summary(svm_cost01)
```

```
##
## Call:
## svm(formula = Purchase ~ ., data = OJ[train, ], cost = 0.01,
##     kernel = "linear")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  0.01
##       gamma:  0.05555556
##
## Number of Support Vectors:  432
##
##  ( 215 217 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

We have a low cost of violating the margin and as a result many support vectors.

Part 8.c)

What are the training and test error rates?

```r
err_vec <- c(NA, NA)
names(err_vec) <- c("train", "test")
# training error
pred_cost01_tr <- predict(svm_cost01, newdata = OJ[train, ])
err_vec["train"] <- mean(pred_cost01_tr != OJ$Purchase[train])
# test error
pred_cost01_te <- predict(svm_cost01, newdat = OJ[test, ])
err_vec["test"] <- mean(pred_cost01_te != OJ$Purchase[test])

err_vec
```

```
##     train      test
## 0.1662500 0.1814815
```

Part 8.d)

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

```
set.seed(1)
tune_lin <- tune(svm, Purchase ~ ., data = OJ[train, ], kernel = "linear",
                 ranges = list(cost = 10^seq(-2, 1, by = 0.25)))
summary(tune_lin)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##        cost
##  0.01778279
##
## - best performance: 0.1725
##
## - Detailed performance results:
##           cost   error dispersion
## 1   0.01000000 0.17500 0.03996526
## 2   0.01778279 0.17250 0.04116363
## 3   0.03162278 0.17375 0.03884174
## 4   0.05623413 0.17500 0.04082483
## 5   0.10000000 0.17875 0.03821086
## 6   0.17782794 0.17625 0.03557562
## 7   0.31622777 0.17625 0.03701070
## 8   0.56234133 0.17750 0.03717451
## 9   1.00000000 0.17750 0.03717451
## 10  1.77827941 0.17875 0.03488573
## 11  3.16227766 0.18000 0.03496029
## 12  5.62341325 0.17875 0.03634805
## 13 10.00000000 0.18000 0.04005205
```

The cost paramter equal to 0.0177 results in minimum cross-validation error, which is 17.25%.

Part 8.e)

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

```
err_lin <- c(NA, NA)
names(err_lin) <- c("train", "test")
# training error
pred_lin_tr <- predict(tune_lin$best.model, newdata = OJ[train, ])
err_lin["train"] <- mean(pred_lin_tr != OJ$Purchase[train])
# test error
pred_lin_te <- predict(tune_lin$best.model, newdata = OJ[test, ])
err_lin["test"] <- mean(pred_lin_te != OJ$Purchase[test])

err_lin
```

```
##     train      test
## 0.1600000 0.1851852
```

Part 8.f)

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

```r
err_rad <- matrix(NA, nrow = 2, ncol = 2,
                  dimnames = list(model = c("cost01", "tune"),
                                  subset = c("train", "test")))
## cost = 0.01 with default gamma
svm_rad_cost01 <- svm(Purchase ~ ., data = OJ[train, ], cost = 0.01,
                      kernel = "radial")
# train
pred_rad01_tr <- predict(svm_rad_cost01, newdata = OJ[train, ])
err_rad["cost01", "train"] <- mean(pred_rad01_tr != OJ$Purchase[train])
# test
pred_rad01_te <- predict(svm_rad_cost01, newdata = OJ[test, ])
err_rad["cost01", "test"] <- mean(pred_rad01_te != OJ$Purchase[test])

## tune with default gamma
tune_rad <- tune(svm, Purchase ~ ., data = OJ[train, ],
                 ranges = list(cost = 10^seq(-2, 1, by = 0.25)),
                 kernel = "radial")
# train
pred_radtune_tr <- predict(tune_rad$best.model, newdata = OJ[train, ])
err_rad["tune", "train"] <- mean(pred_radtune_tr != OJ$Purchase[train])
# test
pred_radtune_te <- predict(tune_rad$best.model, newdata = OJ[test, ])
err_rad["tune", "test"] <- mean(pred_radtune_te != OJ$Purchase[test])

summary(tune_rad)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##       cost
##  0.3162278
##
## - best performance: 0.1625
##
## - Detailed performance results:
##           cost   error dispersion
## 1   0.01000000 0.38250 0.04297932
## 2   0.01778279 0.38250 0.04297932
## 3   0.03162278 0.37250 0.04923018
## 4   0.05623413 0.19875 0.05382908
## 5   0.10000000 0.17875 0.05834821
## 6   0.17782794 0.16750 0.05779514
## 7   0.31622777 0.16250 0.05170697
## 8   0.56234133 0.16375 0.05084358
## 9   1.00000000 0.16375 0.05318012
## 10  1.77827941 0.16375 0.04693746
## 11  3.16227766 0.17125 0.04860913
## 12  5.62341325 0.17125 0.05304937
## 13 10.00000000 0.17125 0.05466120
```

```
err_rad
```

```
##         subset
## model     train      test
##   cost01 0.3825 0.4111111
##   tune   0.1550 0.1629630
```

Both training and test error rates decrease compared to linear kernel, even though we do not cross-validate over gamma.

Part 8.g)

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

```
err_pol <- matrix(NA, nrow = 2, ncol = 2,
                  dimnames = list(model = c("cost01", "tune"),
                                  subset = c("train", "test")))
## cost = 0.01 with default gamma
svm_pol_cost01 <- svm(Purchase ~ ., data = OJ[train, ], cost = 0.01,
                      kernel = "polynomial", degree = 2)
# train
pred_pol01_tr <- predict(svm_pol_cost01, newdata = OJ[train, ])
err_pol["cost01", "train"] <- mean(pred_pol01_tr != OJ$Purchase[train])
# test
pred_pol01_te <- predict(svm_pol_cost01, newdata = OJ[test, ])
err_pol["cost01", "test"] <- mean(pred_pol01_te != OJ$Purchase[test])

## tune with default gamma
tune_pol <- tune(svm, Purchase ~ ., data = OJ[train, ],
                 ranges = list(cost = 10^seq(-2, 1, by = 0.25)),
                 kernel = "polynomial", degree = 2)
# train
pred_poltune_tr <- predict(tune_pol$best.model, newdata = OJ[train, ])
err_pol["tune", "train"] <- mean(pred_poltune_tr != OJ$Purchase[train])
# test
pred_poltune_te <- predict(tune_pol$best.model, newdata = OJ[test, ])
err_pol["tune", "test"] <- mean(pred_poltune_te != OJ$Purchase[test])

summary(tune_pol)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##      cost
##  5.623413
##
## - best performance: 0.175
##
## - Detailed performance results:
```

```
##           cost   error dispersion
## 1   0.01000000 0.38250 0.06800735
## 2   0.01778279 0.36750 0.06645801
## 3   0.03162278 0.36000 0.06917169
## 4   0.05623413 0.33750 0.06066758
## 5   0.10000000 0.32625 0.05415064
## 6   0.17782794 0.26875 0.04135299
## 7   0.31622777 0.21750 0.04456581
## 8   0.56234133 0.21000 0.05296750
## 9   1.00000000 0.19750 0.04923018
## 10  1.77827941 0.18875 0.04839436
## 11  3.16227766 0.18750 0.04208127
## 12  5.62341325 0.17500 0.03632416
## 13 10.00000000 0.18125 0.03784563
```

```
err_pol
```

```
##         subset
## model     train      test
##   cost01 0.3825 0.4111111
##   tune   0.1475 0.1851852
```

Although we see some improvement on *training* error rate of the tuned polynomial kernel SVM, the test training rate remains the same as the linear kernel.

Part 8.h)

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

Radial-kernel SVM. However, we should also highlight the fact that we didn't cross-validate over `gamma` and `degree`.