Support Vector Machines Ch. 9 Exercises

Lucy L.

2024-02-03

```
library(ISLR2)
library(e1071)
```

Exercise 5

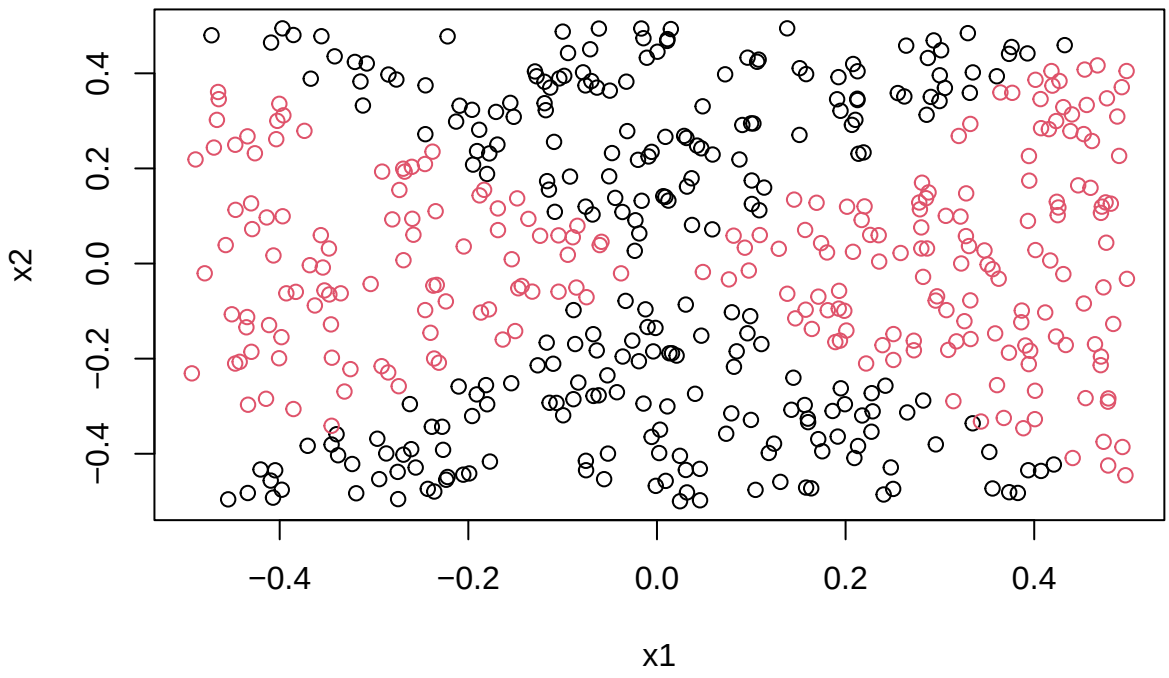
We have seen that we can ft an SVM with a non-linear kernel in order to perform classification using a non-linear decision boundary. We will now see that we can also obtain a non-linear decision boundary by performing logistic regression using non-linear transformations of the features.

(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:

```
x1 <- runif(500) - 0.5
x2 <- runif(500) - 0.5
y <- 1 * (x1^2 - x2^2 > 0)
x_mat = cbind(x1, x2)
df1 = data.frame(x1, x2, y = as.factor(y))
```

(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.

```
set.seed(1)
plot(x_mat, col = df1$y)
```

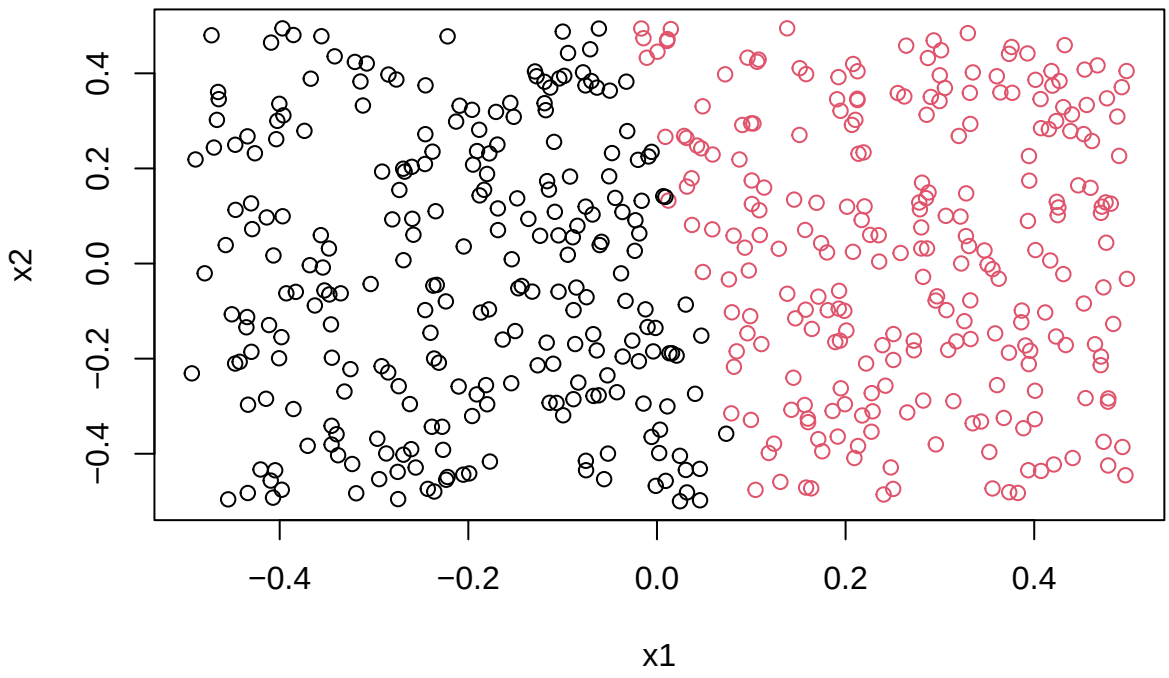


(c) Fit a logistic regression model to the data, using X_1 and X_2 as predictors.

```
log_mod = glm(y ~ x1 + x2, family = "binomial")
```

(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.

```
preds1 = predict(log_mod, df1)
preds2 <- ifelse(preds1 < mean(preds1), 0, 1)

plot(x1, x2, col = as.factor(preds2))
```

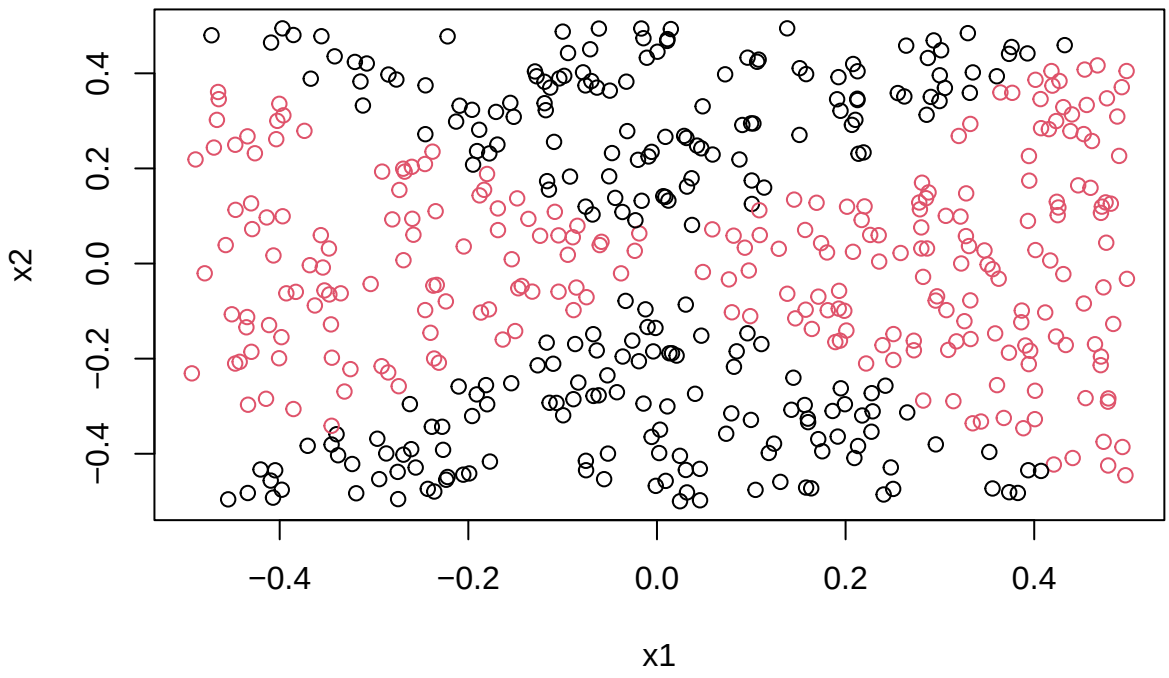


(e) Now fit a logistic regression model to the data using non-linear functions of X_1 and X_2 as predictors (e.g. X_1^2, $X_1 X_2$, $log(X_2)$, and so forth).

```
x1_sq = x1 ** 2
x2_sq = x2 ** 2
df2 = data.frame(x1_sq, x2_sq, y)

mod2 = glm(y ~ ., data = df2, family = "binomial")
```

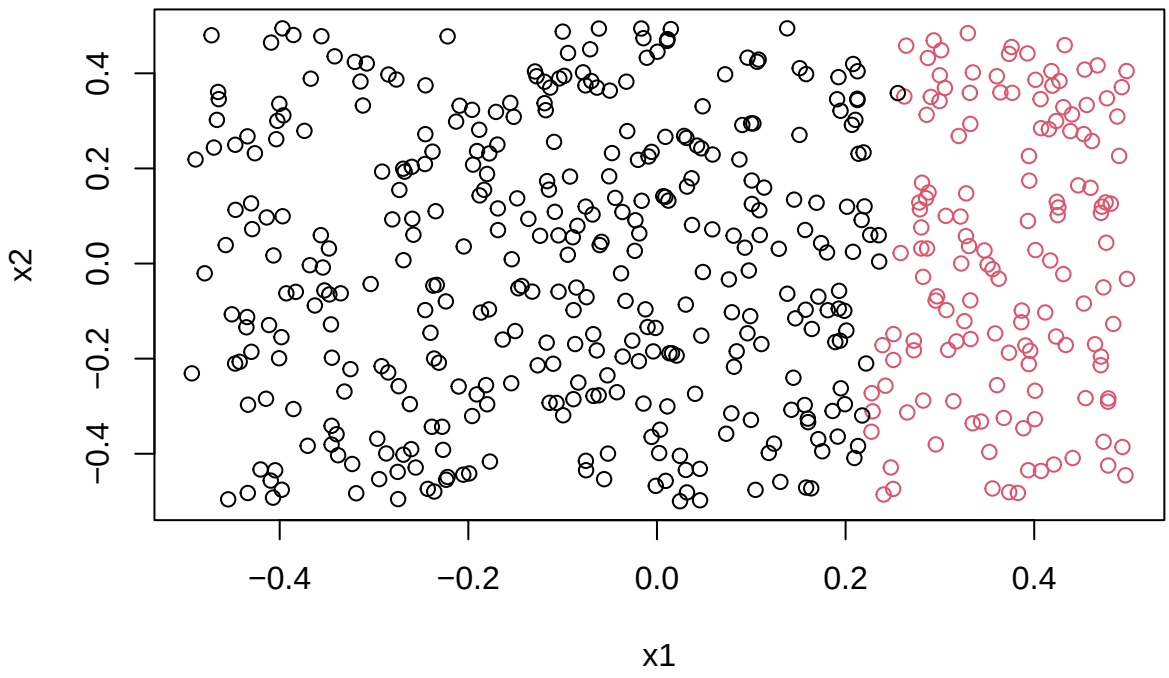
(f) Apply this model to the training data in order to obtain a predicted class label for each training observation. Plot the observations, colored according to the predicted class labels. The decision boundary should be obviously non-linear. If it is not, then repeat (a)-(e) until you come up with an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

```
preds3 = predict(mod2, df2)
preds4 <- ifelse(preds3 < mean(preds3), 0, 1)

plot(x_mat, col = as.factor(preds4))
```



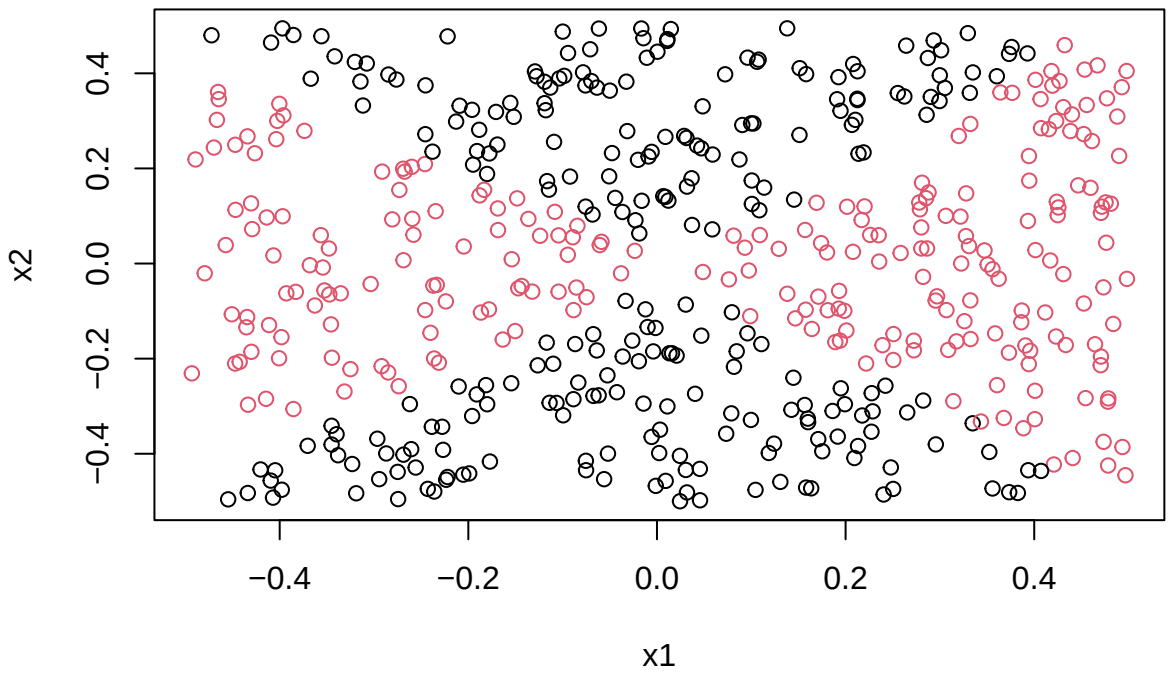
(g) Fit a support vector classifier to the data with X_1 and X_2 as predictors. Obtain a class prediction for each training observation. Plot the observations, colored according to the predicted class labels.

```
dat = data.frame(x = x_mat, y = as.factor(y))
svmfit1 = svm(y ~ ., data = dat, kernel = "linear", cost = 1e5)

pred5 = predict(svmfit1, dat)
plot(x_mat, col = pred5)
```



(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.

```
svmfit2 = svm(y ~ ., data = dat, kernel = "radial", gamma = 1, cost = 1)
preds7 = predict(svmfit2, dat)
plot(x_mat, col = preds7)
```



(i) Comment on your results.

The SVM with a radial kernel outperforms logistic regression and the SVM with a linear kernel for predicting the simulated data by a long shot evidently. The predictions for the SVM with a radial kernel produce nearly identical results to the original data.

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.

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

```
set.seed(1)
x = matrix(rnorm(200 * 2), ncol = 2)
x[1:100, ] = x[1:100, ] + 2
x[101:150, ] = x[101:150, ] - 2
y = c(rep(1, 150), rep(2, 50))
dat = data.frame(x = x, y = as.factor(y))
```

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

```r
set.seed(1)
train = sample(200, 100)
tune.out.6 = tune(svm, y ~ ., data = dat[train, ], kernel = "radial",
                  ranges = list(cost = c(0.1, 1, 10, 100, 1000),
                                gamma = c(0.5, 1, 2, 3, 4)))
summary(tune.out.6)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost gamma
##     1   0.5
##
## - best performance: 0.11
##
## - Detailed performance results:
##     cost gamma error dispersion
## 1  1e-01   0.5  0.29 0.18529256
## 2  1e+00   0.5  0.11 0.07378648
## 3  1e+01   0.5  0.12 0.10327956
## 4  1e+02   0.5  0.15 0.12692955
## 5  1e+03   0.5  0.20 0.12472191
## 6  1e-01   1.0  0.28 0.20439613
## 7  1e+00   1.0  0.12 0.10327956
## 8  1e+01   1.0  0.15 0.12692955
## 9  1e+02   1.0  0.19 0.12866839
## 10 1e+03   1.0  0.20 0.11547005
## 11 1e-01   2.0  0.30 0.20000000
## 12 1e+00   2.0  0.12 0.10327956
## 13 1e+01   2.0  0.18 0.12292726
## 14 1e+02   2.0  0.20 0.14142136
## 15 1e+03   2.0  0.22 0.13984118
## 16 1e-01   3.0  0.30 0.17638342
## 17 1e+00   3.0  0.13 0.12516656
## 18 1e+01   3.0  0.18 0.12292726
## 19 1e+02   3.0  0.21 0.15951315
## 20 1e+03   3.0  0.24 0.16465452
## 21 1e-01   4.0  0.29 0.18529256
## 22 1e+00   4.0  0.15 0.11785113
## 23 1e+01   4.0  0.19 0.13703203
## 24 1e+02   4.0  0.25 0.17159384
## 25 1e+03   4.0  0.28 0.16865481
```

```r
table(true = dat[train, "y"], pred = predict(tune.out.6$best.model,
                                             newdata = dat[train, ]))
```

```
##     pred
## true  1  2
##    1 65  6
##    2  4 25
```

```r
8 / (8 + 65 + 25)
```

```
## [1] 0.08163265
```

Given the cost (1) and gamma (0.5) of the best model, the computed CV error is about 0.11. This is higher than the misclassification rate for the corresponding model on the training data, which is about 0.08.

(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
tune.out.7 = tune(svm, y ~ ., data = dat[-train, ], kernel = "radial",
                  ranges = list(cost = c(0.1, 1, 10, 100, 1000),
                                gamma = c(0.5, 1, 2, 3, 4)))
summary(tune.out.7)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost gamma
##    10     4
##
## - best performance: 0.08
##
## - Detailed performance results:
##     cost gamma error dispersion
## 1  1e-01   0.5  0.21 0.11005049
## 2  1e+00   0.5  0.11 0.08755950
## 3  1e+01   0.5  0.12 0.11352924
## 4  1e+02   0.5  0.14 0.11737878
## 5  1e+03   0.5  0.13 0.10593499
## 6  1e-01   1.0  0.21 0.11005049
## 7  1e+00   1.0  0.11 0.07378648
## 8  1e+01   1.0  0.11 0.11005049
## 9  1e+02   1.0  0.13 0.10593499
## 10 1e+03   1.0  0.11 0.11972190
## 11 1e-01   2.0  0.21 0.11005049
## 12 1e+00   2.0  0.11 0.08755950
## 13 1e+01   2.0  0.11 0.11005049
## 14 1e+02   2.0  0.12 0.13984118
## 15 1e+03   2.0  0.14 0.12649111
## 16 1e-01   3.0  0.21 0.11005049
## 17 1e+00   3.0  0.10 0.09428090
## 18 1e+01   3.0  0.09 0.11972190
## 19 1e+02   3.0  0.13 0.12516656
## 20 1e+03   3.0  0.14 0.12649111
## 21 1e-01   4.0  0.21 0.11005049
## 22 1e+00   4.0  0.13 0.15670212
## 23 1e+01   4.0  0.08 0.12292726
```

```
## 24 1e+02   4.0  0.13 0.11595018
## 25 1e+03   4.0  0.13 0.11595018
```

```
table(true = dat[-train, "y"], pred = predict(tune.out.7$best.model,
                                              newdata = dat[-train, ]))
```

```
##     pred
## true  1  2
##    1 77  2
##    2  1 20
```

```
3 / (3 + 77 + 20)
```

```
## [1] 0.03
```

The same values of `cost` and `gamma` as from the previous training model result in the lowest 10-fold CV error among the test data. The lowest CV error is about 0.1, while the misclassification rate is only about 0.03.

For the range of `cost` we considered, a cost of 1 (the second lowest, after 0.01) consistently resulted in the lowest CV errors among both the training and test data.

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.

(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.

```
auto = Auto

auto$med = ifelse(Auto$mpg < median(Auto$mpg), 0, 1)
```

(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. Note you will need to fit the classifier without the gas mileage variable to produce sensible results.

```
tune.cv = tune(svm, med ~ . - mpg - name, data = auto, kernel = "linear",
               ranges = list(cost = c(0.1, 1, 10, 100)))
summary(tune.cv)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
```

```
##   0.1
##
## - best performance: 0.1072975
##
## - Detailed performance results:
##    cost     error dispersion
## 1   0.1 0.1072975 0.02951964
## 2   1.0 0.1085526 0.02966352
## 3  10.0 0.1089380 0.02990246
## 4 100.0 0.1089285 0.02994876
```

The lowest CV errors occur at a cost of 0.1.

(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.

```
tune.cv.r = tune(svm, med ~ . - mpg - name, data = auto, kernel = "radial",
                 ranges = list(cost = c(0.1, 1, 10, 100),
                               gamma = c(0.5, 1, 2)))
summary(tune.cv.r)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost gamma
##     1     1
##
## - best performance: 0.06169571
##
## - Detailed performance results:
##     cost gamma      error dispersion
## 1    0.1   0.5 0.07133861 0.02742232
## 2    1.0   0.5 0.06223564 0.02741506
## 3   10.0   0.5 0.06317814 0.03172540
## 4  100.0   0.5 0.09732556 0.04682033
## 5    0.1   1.0 0.08277036 0.02424059
## 6    1.0   1.0 0.06169571 0.02705795
## 7   10.0   1.0 0.07019667 0.03608094
## 8  100.0   1.0 0.11491776 0.04892062
## 9    0.1   2.0 0.11935483 0.02662626
## 10   1.0   2.0 0.07133329 0.02576441
## 11  10.0   2.0 0.08660397 0.03605380
## 12 100.0   2.0 0.09384102 0.03497640
```

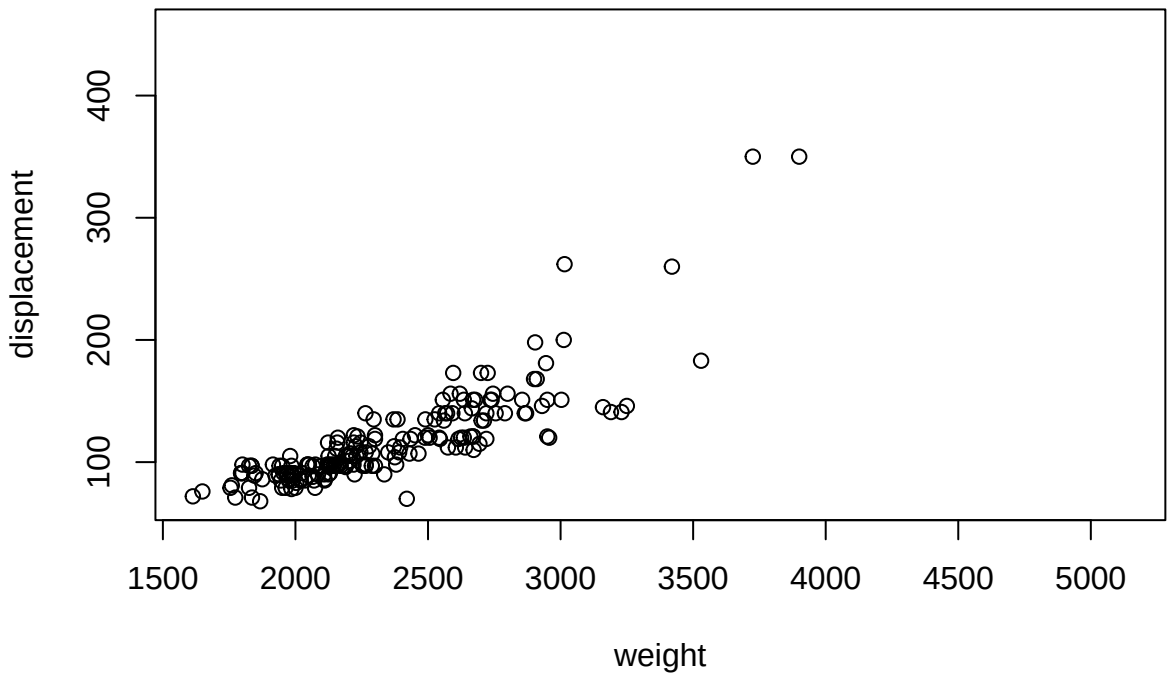
```
tune.cv.p = tune(svm, med ~ . - mpg - name, data = auto, kernel = "polynomial",
                 ranges = list(cost = c(0.1, 1, 10),
                               gamma = c(0.5, 1)))
summary(tune.cv.p)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost gamma
##     1   0.5
##
## - best performance: 0.09909844
##
## - Detailed performance results:
##   cost gamma      error dispersion
## 1  0.1   0.5 0.10887532 0.02907115
## 2  1.0   0.5 0.09909844 0.03216641
## 3 10.0   0.5 0.10000808 0.03349042
## 4  0.1   1.0 0.10012179 0.03224450
## 5  1.0   1.0 0.09932735 0.03286887
## 6 10.0   1.0 0.10907899 0.04411403
```

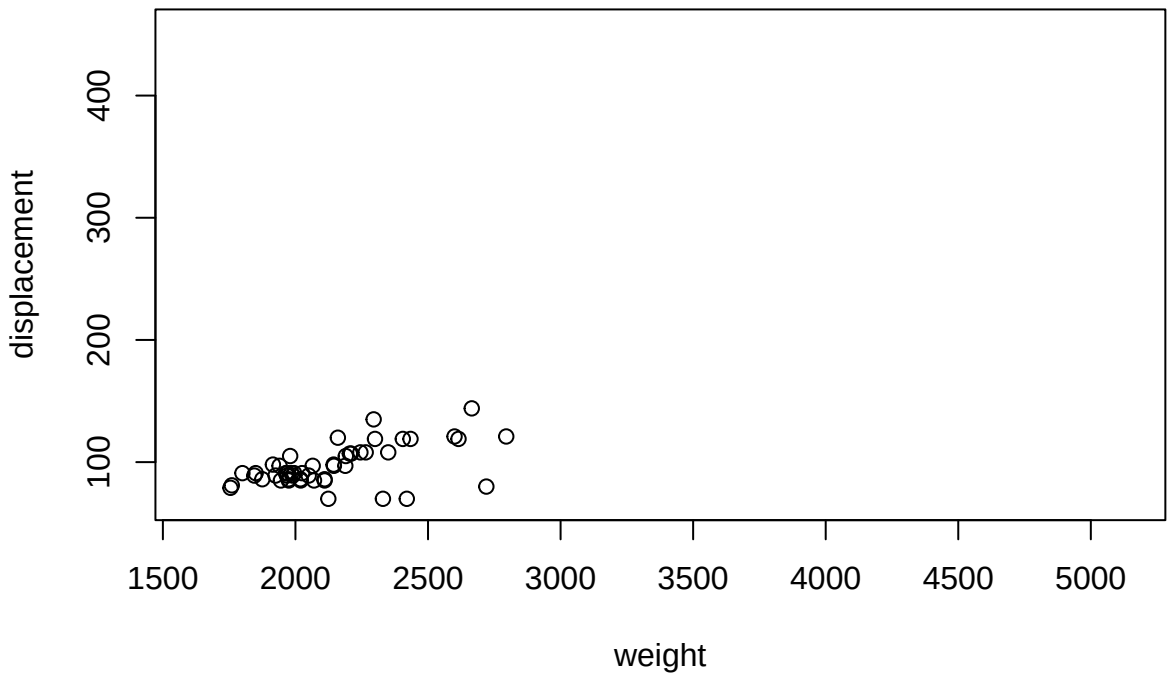
For the radial kernel, the lowest CV error occurs at a cost of 1, while for the polynomial kernel it occurs at a cost of 10. A radial kernel may be the best choice for fitting an SVM on this data and the following plots support this claim since the SVM plots closely match the predictor plots.

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

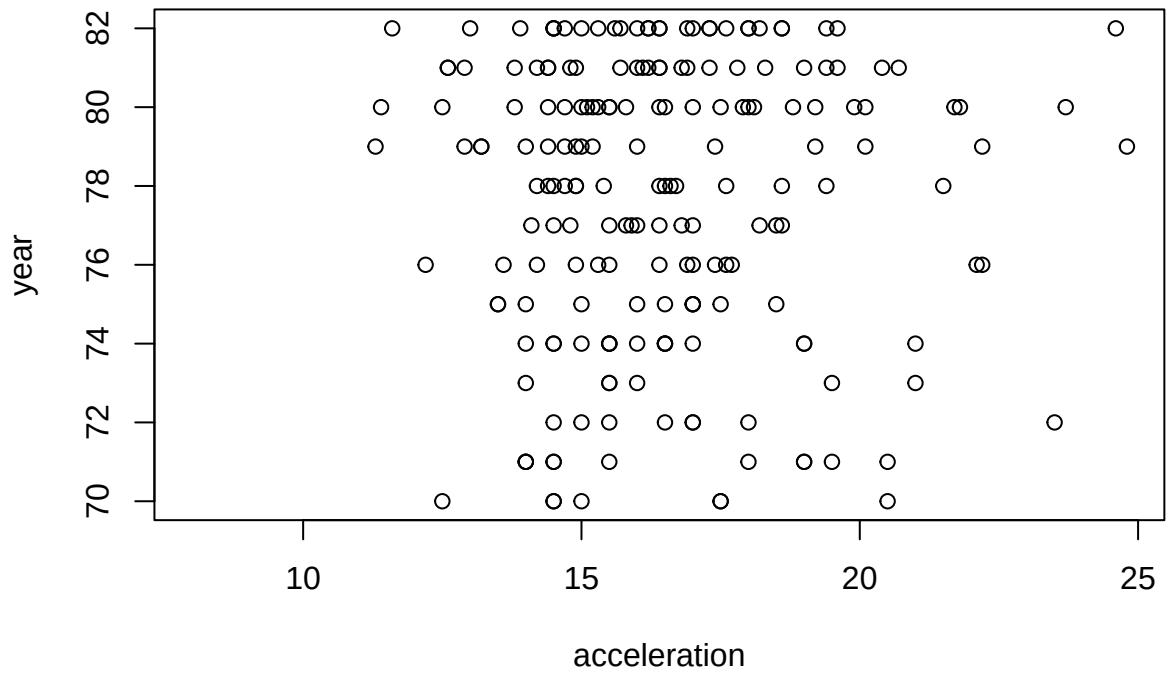
```r
attach(auto)
x_mat2 = cbind(cylinders, displacement, horsepower, weight, acceleration,
               year, origin)
df3 = data.frame(x_mat2, med)

svmfit.lin = svm(med ~ ., data = df3, kernel = "linear", cost = 0.1)

plot(weight, displacement, col = df3$med)
```



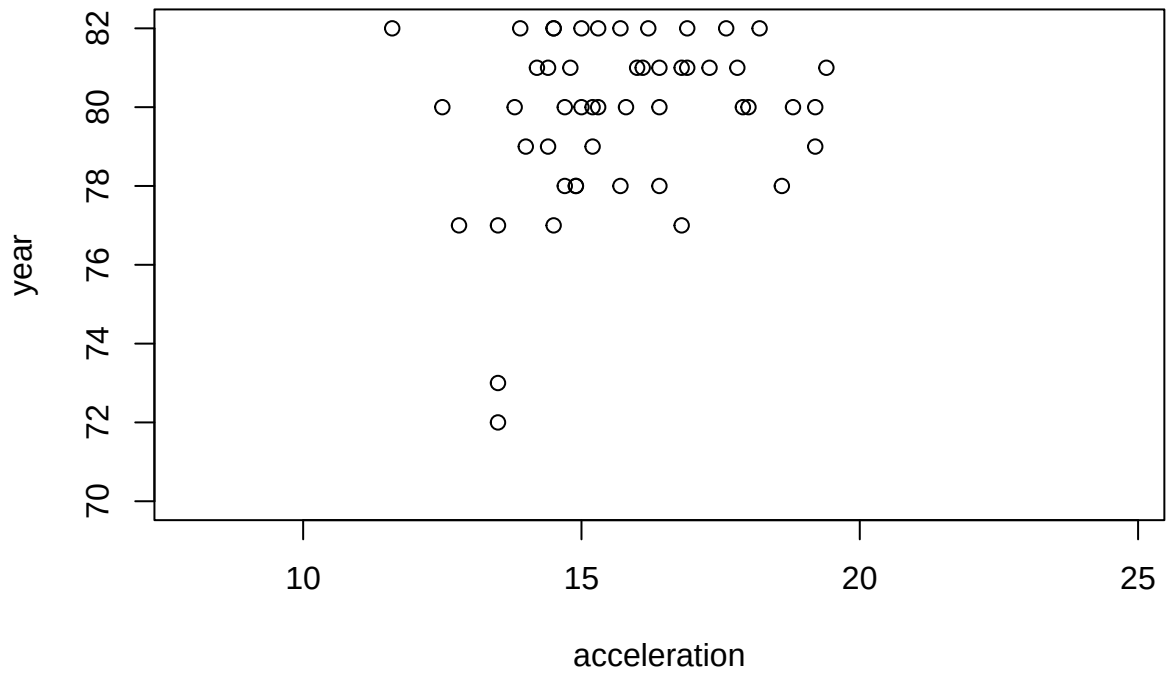

```
plot(weight, displacement, col = svmfit.lin$fitted)
```



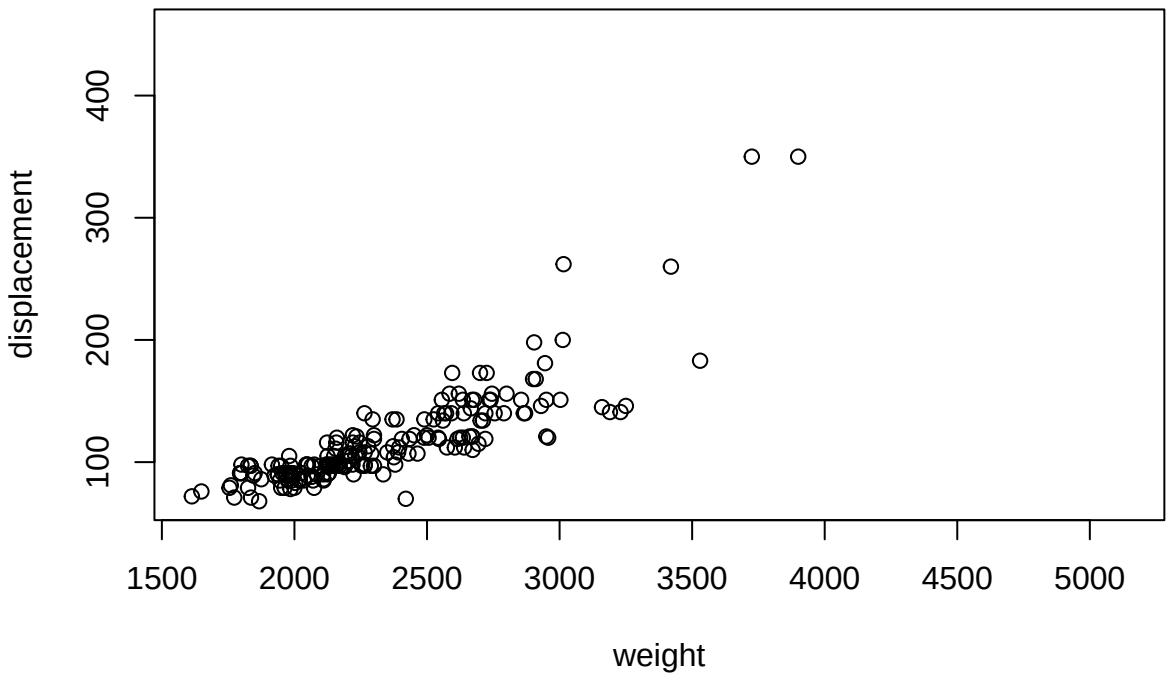

```
plot(acceleration, year, col = df3$med)
```



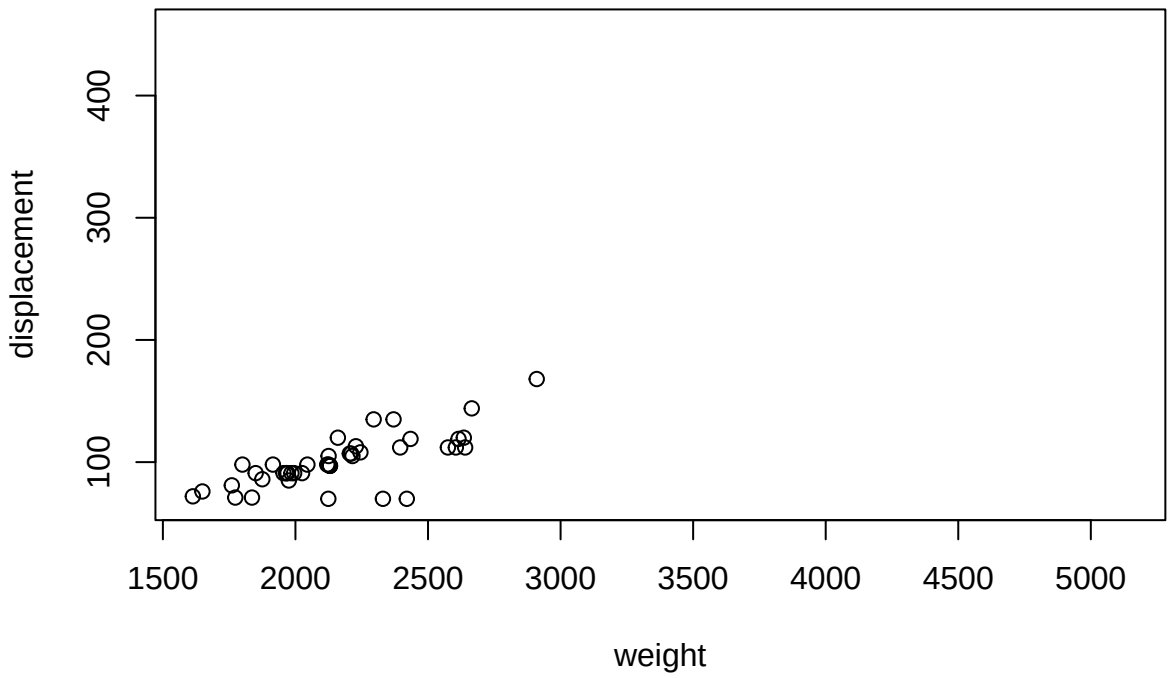

```
plot(acceleration, year, col = svmfit.lin$fitted)
```



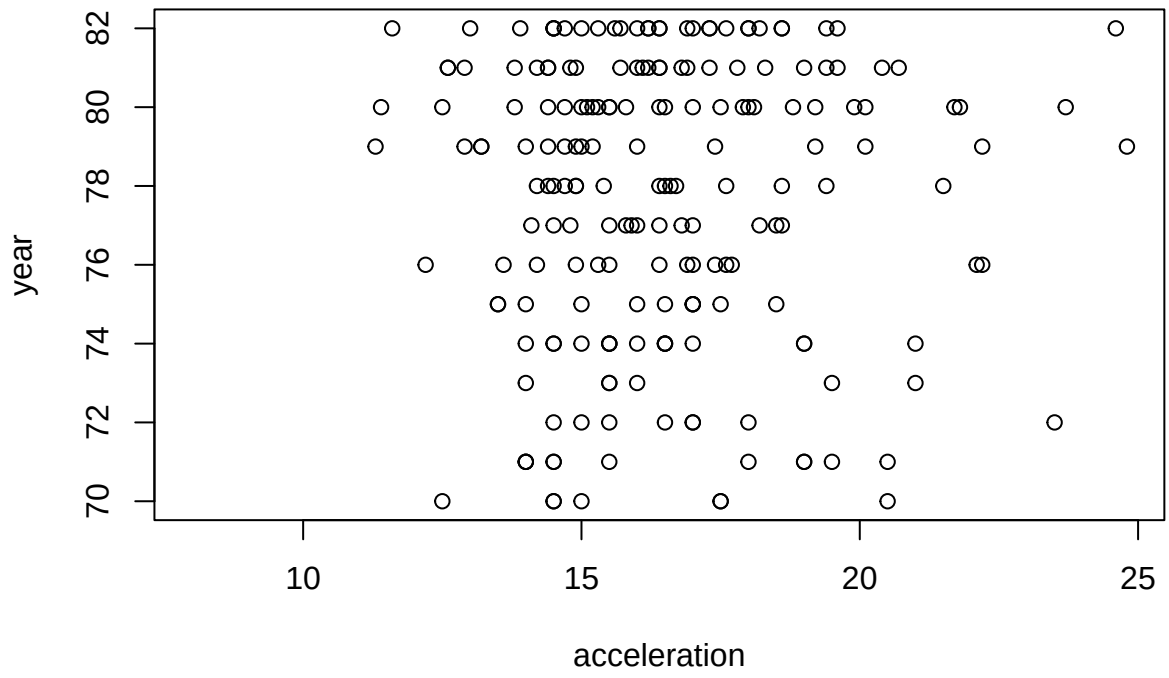

```
svmfit.poly = svm(med ~ ., data = df3, kernel = "polynomial", cost = 1)
plot(weight, displacement, col = df3$med)
```



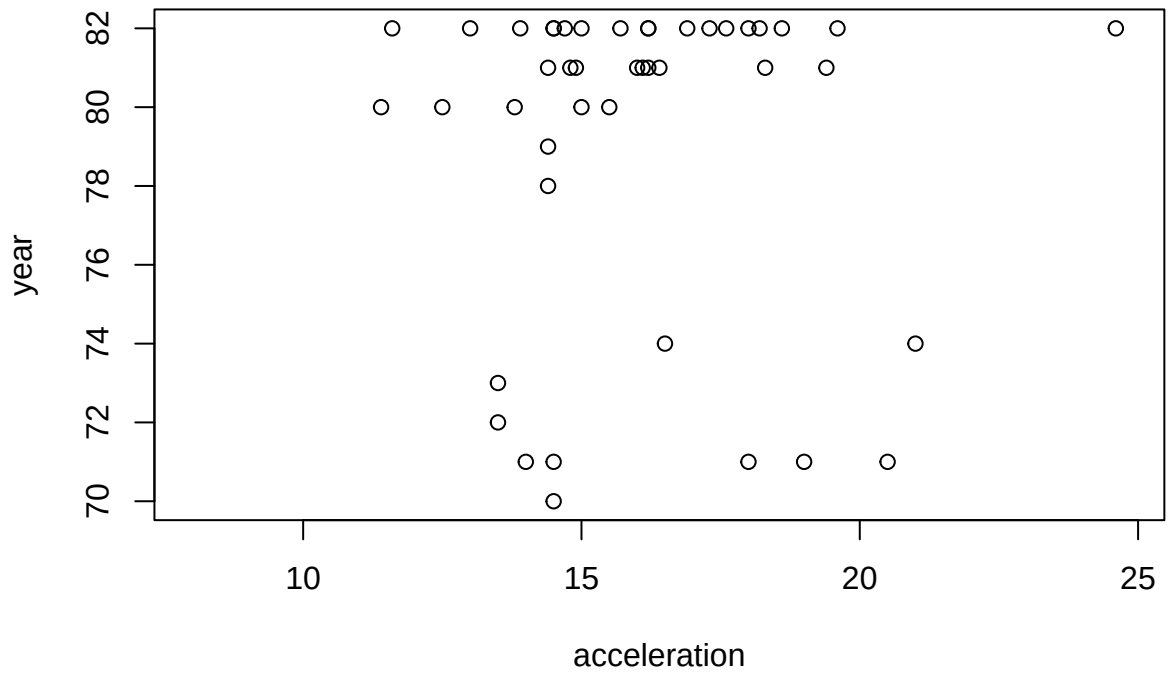

```
plot(weight, displacement, col = svmfit.poly$fitted)
```



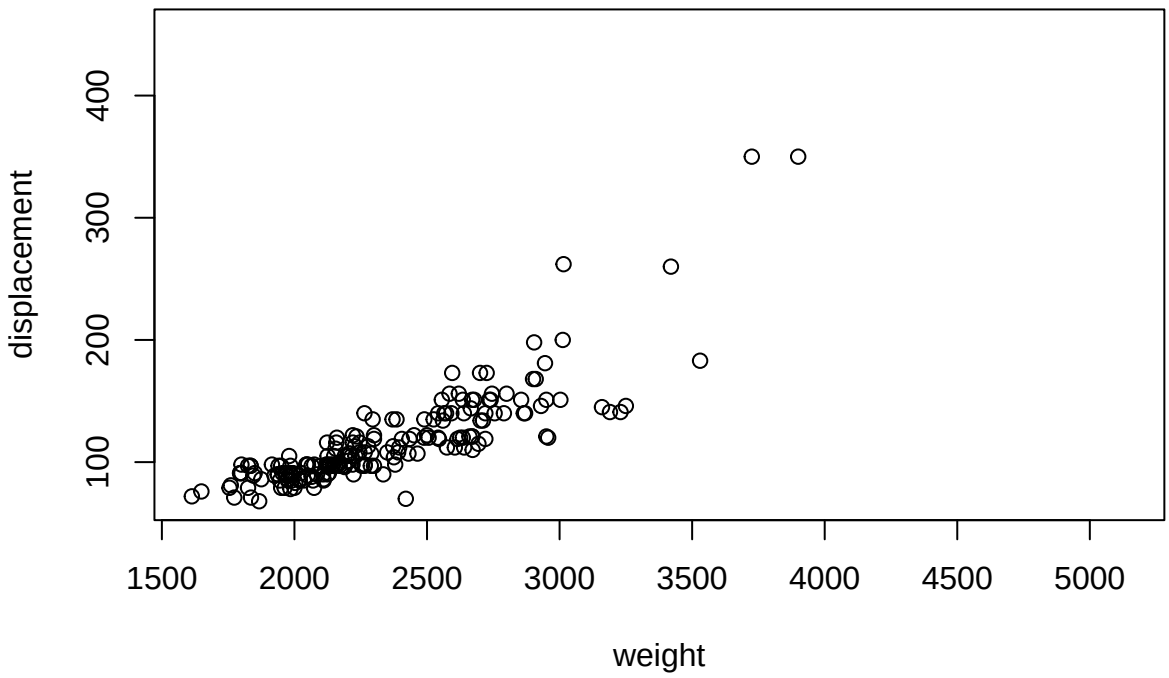

```
plot(acceleration, year, col = df3$med)
```



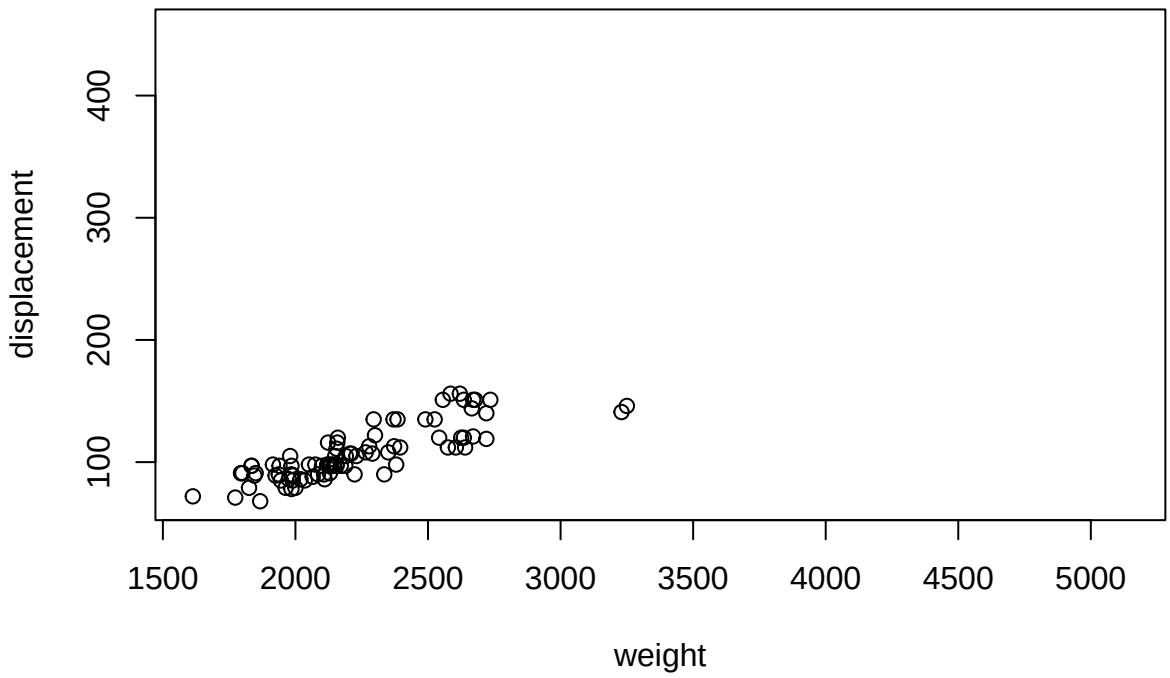

```
plot(acceleration, year, col = svmfit.poly$fitted)
```



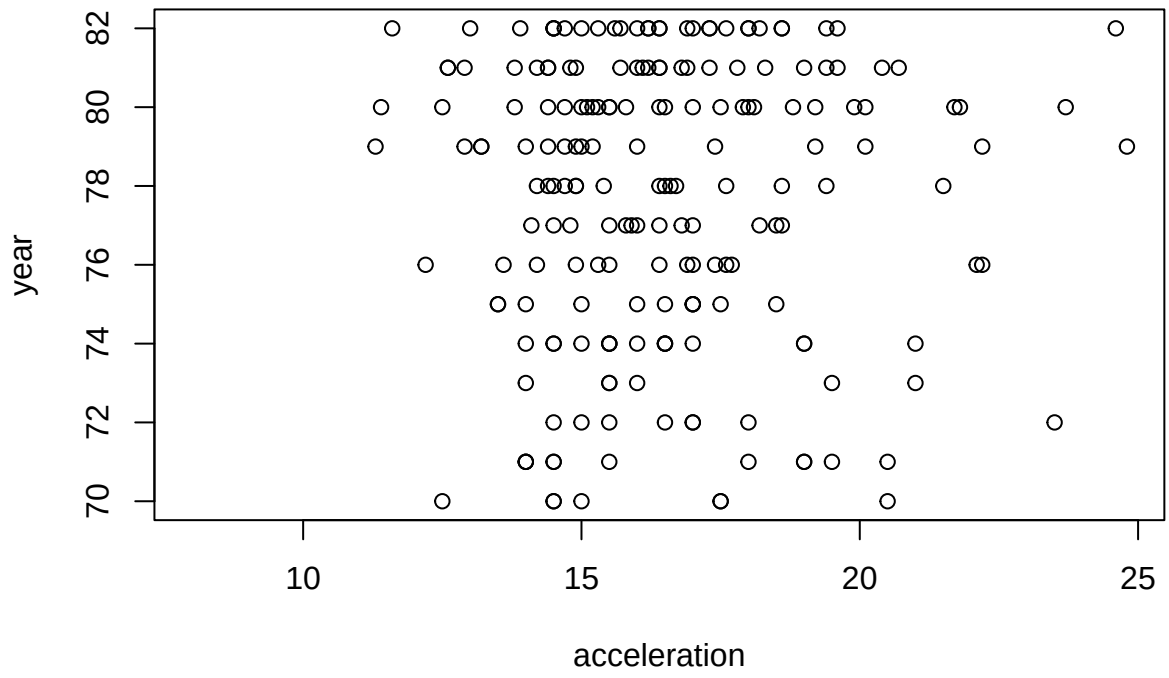

```
svmfit.rad = svm(med ~ ., data = df3, kernel = "radial",  cost = 10)
plot(weight, displacement, col = df3$med)
```



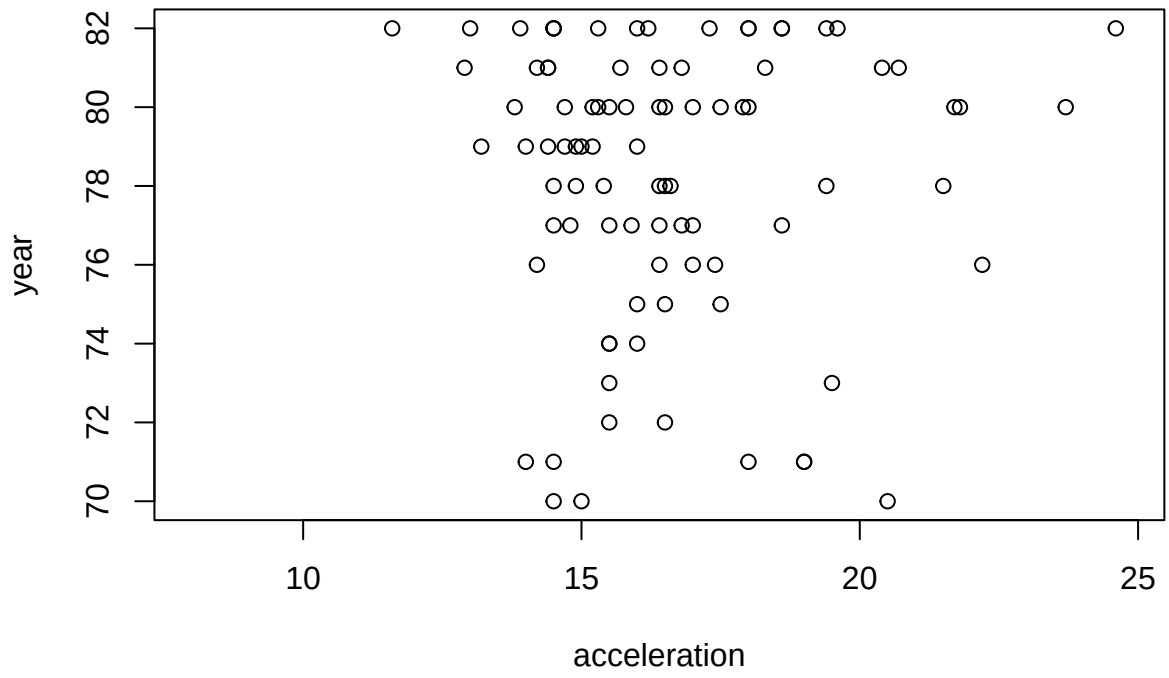

```
plot(weight, displacement, col = svmfit.rad$fitted)
```




```
plot(acceleration, year, col = df3$med)
```




```
plot(acceleration, year, col = svmfit.rad$fitted)
```



Exercise 8

This problem involves the `OJ` data set which is part of the `ISLR2` package.

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

```
oj = OJ
train_subset = sample(800, 200)
train_oj = oj[train_subset, ]
test_oj = oj[-train_subset, ]
```

(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.

```
svmfit.oj.lin = svm(Purchase ~ ., data = train_oj, kernel = "linear",
                    cost = 0.01)
summary(svmfit.oj.lin)
```

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

```
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  0.01
##
## Number of Support Vectors:  137
##
##  ( 67 70 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

There are a significant amount of support vectors (134) for a linear fit.

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

```
table(svmfit.oj.lin$fitted, train_oj$Purchase)
```

```
##
##       CH  MM
##   CH 111  19
##   MM  12  58
```

```
pred.oj = predict(svmfit.oj.lin, test_oj)
table(pred.oj, test_oj$Purchase)
```

```
##
## pred.oj  CH  MM
##      CH 426  79
##      MM 104 261
```

```
(16 + 22) / (114 + 22 + 16 + 48)
```

```
## [1] 0.19
```

```
(37 + 125) / (486+125+37+222)
```

```
## [1] 0.1862069
```

The training error rate is about 0.19, while the test error rate is about 0.186.

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

```
tune.cv.train = tune(svm, Purchase ~ ., data = train_oj, kernel = "linear",
                 ranges = list(cost = c(0.1, 1, 10, 100),
                               gamma = c(0.5, 1, 2)))
summary(tune.cv.train)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost gamma
##   0.1   0.5
##
## - best performance: 0.17
##
## - Detailed performance results:
##     cost gamma error dispersion
## 1    0.1   0.5  0.17 0.08232726
## 2    1.0   0.5  0.18 0.06324555
## 3   10.0   0.5  0.19 0.06582806
## 4  100.0   0.5  0.20 0.07817360
## 5    0.1   1.0  0.17 0.08232726
## 6    1.0   1.0  0.18 0.06324555
## 7   10.0   1.0  0.19 0.06582806
## 8  100.0   1.0  0.20 0.07817360
## 9    0.1   2.0  0.17 0.08232726
## 10   1.0   2.0  0.18 0.06324555
## 11  10.0   2.0  0.19 0.06582806
## 12 100.0   2.0  0.20 0.07817360
```

As seen above, the cost that minimizes the CV error (0.24 at the lowest) can be either 0.1 or 1.

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

```
svmfit.oj.new = svm(Purchase ~ ., data = train_oj, kernel = "linear", cost = 1)

table(svmfit.oj.new$fitted, train_oj$Purchase)
```

```
##
##       CH  MM
##   CH 107  13
##   MM  16  64
```

```
pred.oj2 = predict(svmfit.oj.new, test_oj)
table(pred.oj2, test_oj$Purchase)
```

```
##
## pred.oj2  CH  MM
##       CH 423  54
##       MM 107 286
```

```
(15 + 18) / (115 + 18 + 15 + 52)
```

```
## [1] 0.165
```

```
(44 + 109) / (479 + 109 + 44 + 238)
```

```
## [1] 0.1758621
```

The new training error rate is about 0.165 while the test error rate is about 0.176.

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

```
(14 + 18) / (116 + 18 + 14 + 52)
```

```
## [1] 0.16
```

```
(45 + 108) / (478 + 108 + 45 + 239)
```

```
## [1] 0.1758621
```

Under a radial kernel, with a cost of 1, the new training error rate is about 0.16 while the test error rate is about 0.176 as well. The CV error is about 0.24. With a cost of 0.1, the training error rate was about 0.35 while the test error rate was about 0.399.

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

```
set.seed(1)

# b
svmfit.oj.lin = svm(Purchase ~ ., data = train_oj, kernel = "polynomial",
                    cost = 0.01, degree = 2)
summary(svmfit.oj.lin)
```

```
##
## Call:
## svm(formula = Purchase ~ ., data = train_oj, kernel = "polynomial",
##     cost = 0.01, degree = 2)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  polynomial
##        cost:  0.01
##      degree:  2
##      coef.0:  0
##
## Number of Support Vectors:  160
```

```
##
##  ( 77 83 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

```
# c
table(svmfit.oj.lin$fitted, train_oj$Purchase)
```

```
##
##       CH  MM
##   CH 123  77
##   MM   0   0
```

```
pred.oj = predict(svmfit.oj.lin, test_oj)
table(pred.oj, test_oj$Purchase)
```

```
##
## pred.oj  CH  MM
##      CH 530 340
##      MM   0   0
```

```
70 / 200
```

```
## [1] 0.35
```

```
347 + (523 + 347)
```

```
## [1] 1217
```

```
# d
tune.cv.train = tune(svm, Purchase ~ ., data = train_oj, kernel = "polynomial",
                ranges = list(cost = c(0.1, 1, 10, 100),
                              gamma = c(0.5, 1, 2)), degree = 2 )
summary(tune.cv.train)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost gamma
##    10   0.5
##
## - best performance: 0.21
##
```

```
## - Detailed performance results:
##     cost gamma error dispersion
## 1    0.1   0.5 0.235 0.09442810
## 2    1.0   0.5 0.215 0.10013879
## 3   10.0   0.5 0.210 0.09660918
## 4  100.0   0.5 0.265 0.11067972
## 5    0.1   1.0 0.225 0.09501462
## 6    1.0   1.0 0.230 0.11832160
## 7   10.0   1.0 0.225 0.10341395
## 8  100.0   1.0 0.300 0.12018504
## 9    0.1   2.0 0.240 0.12866839
## 10   1.0   2.0 0.220 0.10327956
## 11  10.0   2.0 0.285 0.10554093
## 12 100.0   2.0 0.285 0.11067972
```

There is no need to change the level of cost while using the polynomial kernel because a cost of 0.1 already results in the lowest CV error (0.26). Again, with a cost of 0.1, the training error rate is about 0.35 while the test error rate is about 0.399.

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

The radial kernel results in the lowest training, test, and CV errors across the board.