HW8

Norman Hong

April 17, 2019

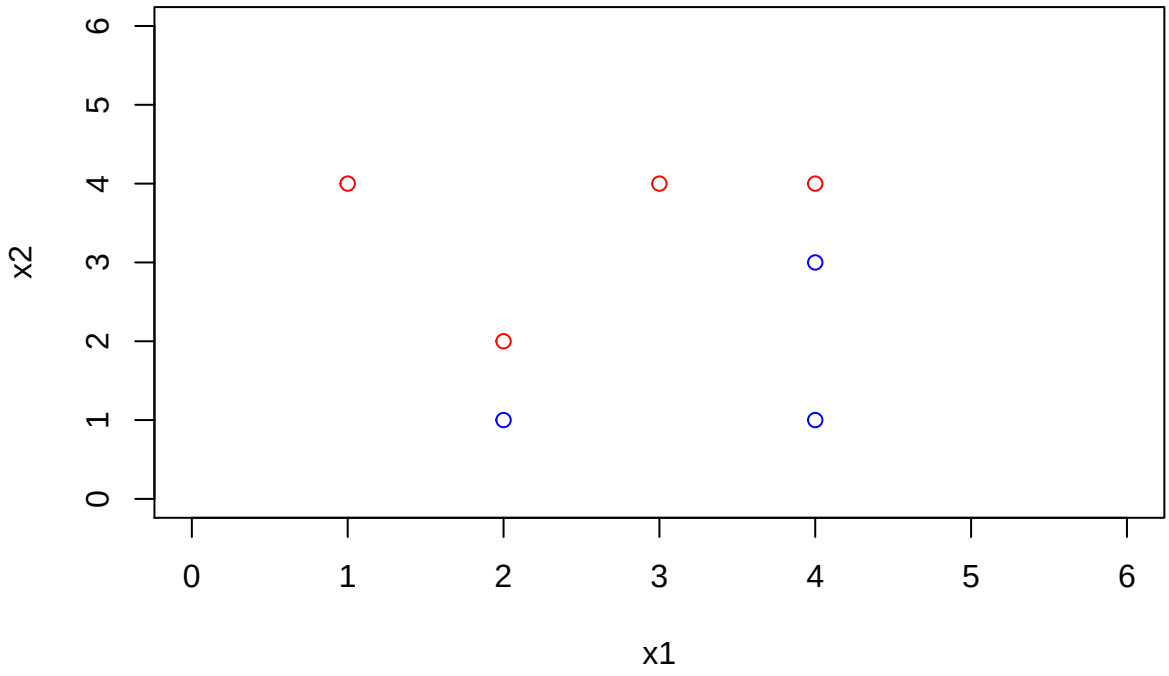
Book 3

Here we explore the maximal margin classifier on a toy data set.

(a)

We are given n=7 observations in p = 2 dimensions. For each observation, there is an associated class label. Sketch the observations.

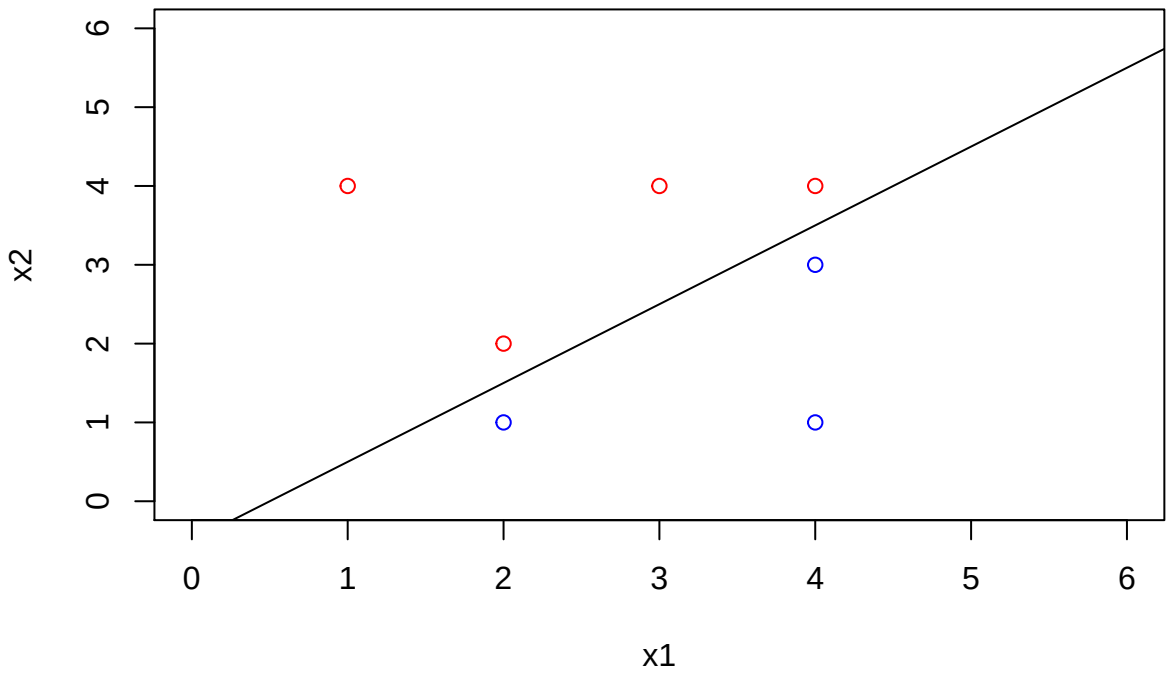
```
x1 = c(3, 2, 4, 1, 2, 4, 4)
x2 = c(4, 2, 4, 4, 1, 3, 1)
colors = c("red", "red", "red", "red", "blue", "blue", "blue")
plot(x1, x2, col = colors, xlim = c(0, 6), ylim = c(0, 6))
```



(b)

Sketch the optimal separating hyperplane and provide the equation for this hyperplane (of the form 9.1).

```
plot(x1, x2, col = colors, xlim = c(0, 6), ylim = c(0, 6))
abline(-0.5, 1)
```



(c)

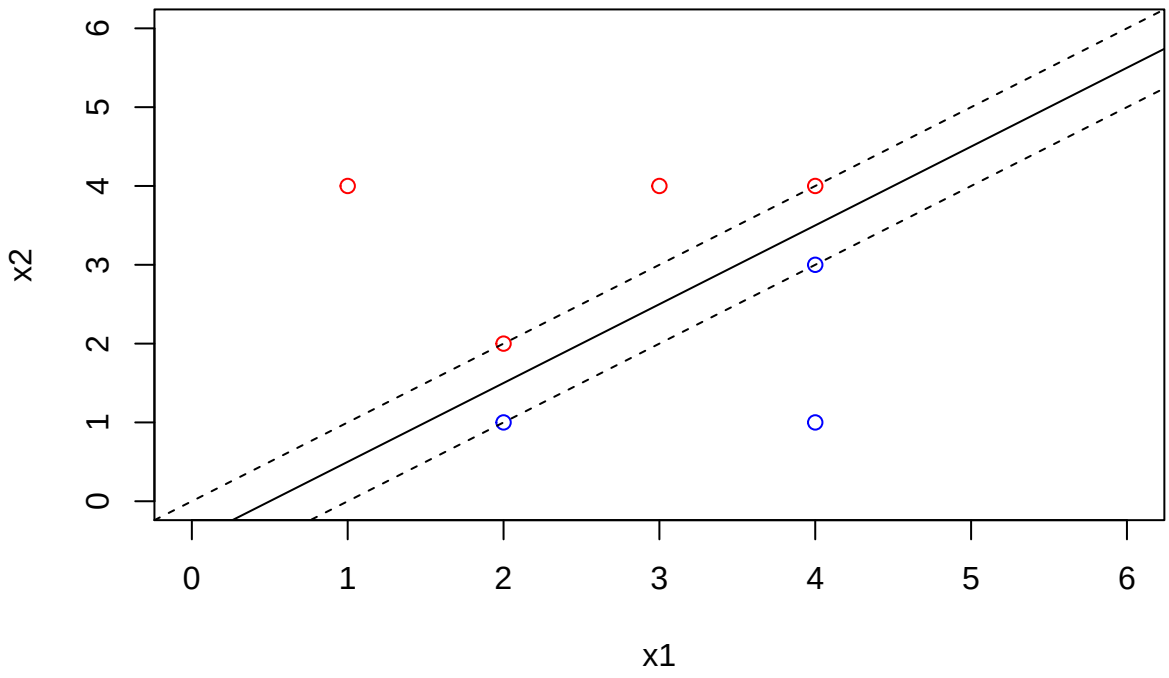
Describe the classification rule for the maximal margin classifier. It should be something along the lines of "Classify to Red if greater if $\beta_0 + \beta_1 * X_1 + \beta_2 * X_2 > 0$ and classify to Blue otherwise". Provide the values for the beta's.

The classification rule is classify to red if $-.5 + X_1 - X_2 > 0$ and classify to blue otherwise.

(d)

On your sketch, indicate the margin for the maximal margin hyperplane.

```
plot(x1, x2, col = colors, xlim = c(0, 6), ylim = c(0, 6))
abline(-1, 1, lty = 2)
abline(-0.5, 1)
abline(0, 1, lty = 2)
```



(e)

Indicate the support vectors for the maximal margin classifier.

The support vectors are the points (2,2), (2,1), (4,4), (4,3).

(f)

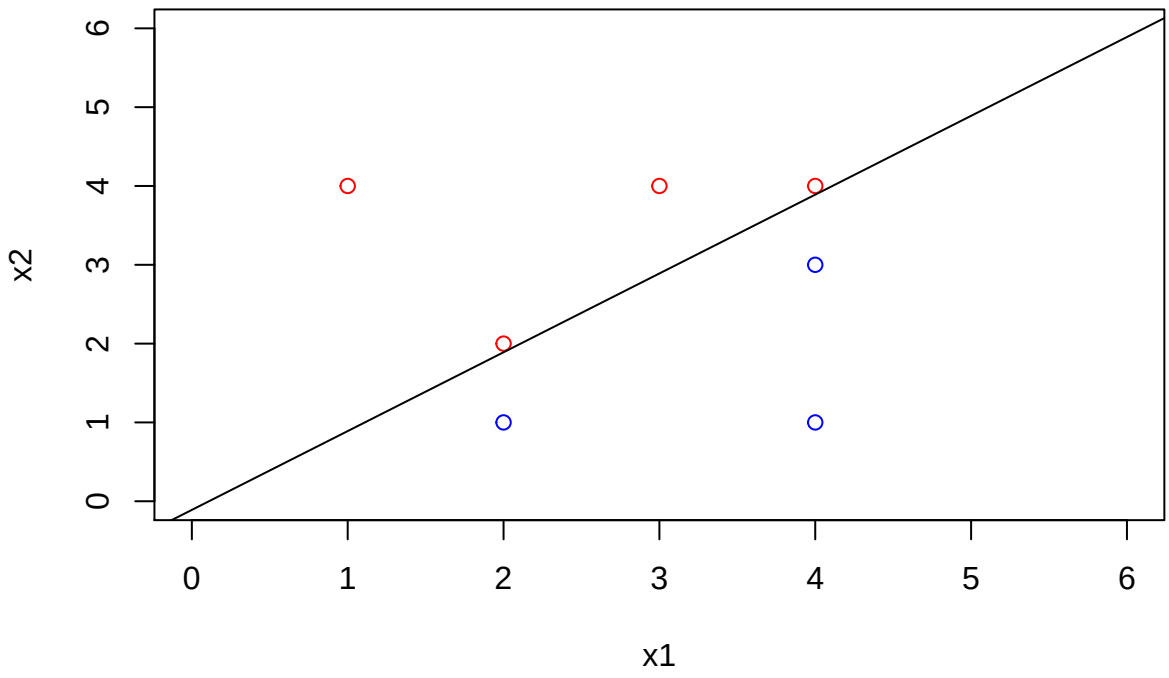
Argue that a slight movement of the seventh observation would not affect the maximal margin hyperplane.

The reason a slight movement of the 7th observation does not affect the hyperplane is because this observation is not a support vector. As long as we keep it away from the hyperplane, it will not affect the margin.

(g)

Sketch a hyperplane that is not the optimal separating hyperplane, and provide the equation for this hyperplane.

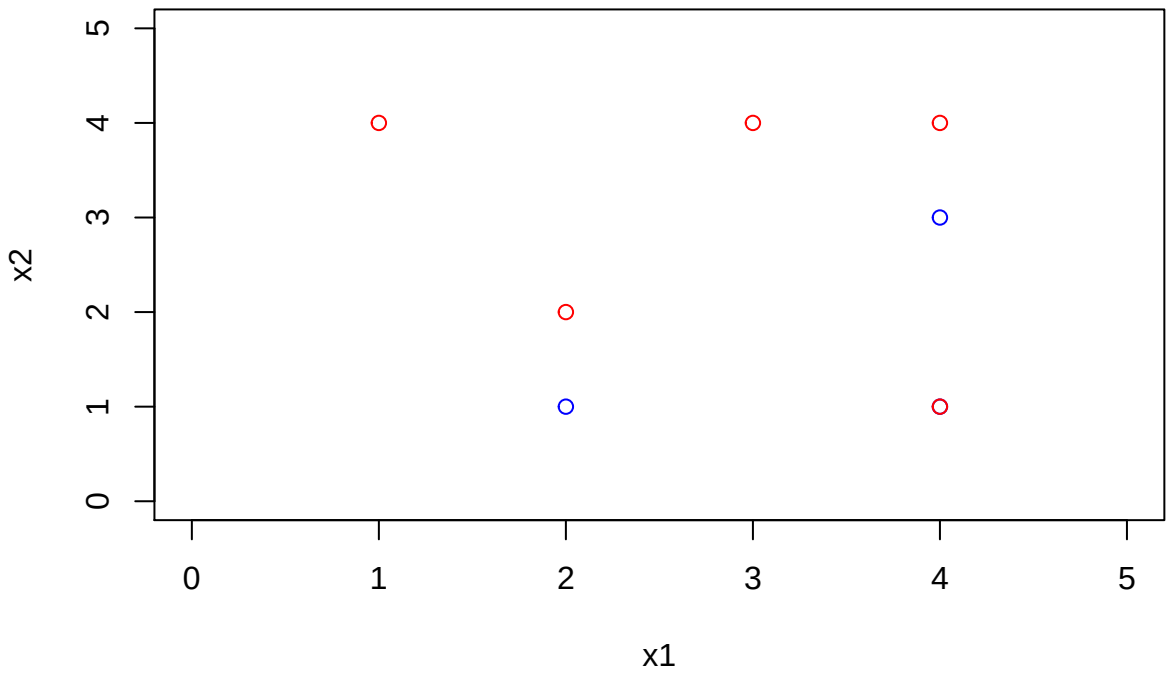
```
plot(x1, x2, col = colors, xlim = c(0, 6), ylim = c(0, 6))
abline(-0.11, 1)
```



(h)

Draw an additional observation on the plot so that the two classes are no longer separable by a hyperplane.

```
plot(x1, x2, col = colors, xlim = c(0, 5), ylim = c(0, 5))
points(c(4), c(1), col = c("red"))
```



Book 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$mpg <- as.factor(ifelse(Auto$mpg > median(Auto$mpg), 1, 0))
```

(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.

When cost is .01, the error is 8.94%, which is the lowest in the range of values tested.

```
set.seed(100)
tune.svm <- tune(svm, mpg~., data=Auto, kernel='linear', ranges=list(cost=c(.001, .01, .1, 1, 10, 100)))
summary(tune.svm)
```

```
##
## Parameter tuning of 'svm':
##
```

```
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##  0.01
##
## - best performance: 0.08942308
##
## - Detailed performance results:
##    cost      error dispersion
## 1 1e-03 0.12762821 0.07353560
## 2 1e-02 0.08942308 0.04419023
## 3 1e-01 0.09961538 0.05349840
## 4 1e+00 0.10448718 0.05563850
## 5 1e+01 0.12230769 0.05599171
## 6 1e+02 0.12230769 0.04903630
```

(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.

The lowest cross-validation error was found to be 7.92%. This corresponded to a cost of 1 and gamma 1

```
set.seed(100)
tune.svm <- tune(svm, mpg~., data=Auto, kernel='radial', ranges=list(cost=c(.001, .01, .1, 1, 10, 100),
summary(tune.svm)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost gamma
##     1     1
##
## - best performance: 0.07929487
##
## - Detailed performance results:
##     cost gamma      error dispersion
## 1  1e-03 1e-04 0.55115385 0.04842542
## 2  1e-02 1e-04 0.55115385 0.04842542
## 3  1e-01 1e-04 0.55115385 0.04842542
## 4  1e+00 1e-04 0.55115385 0.04842542
## 5  1e+01 1e-04 0.11230769 0.06739738
## 6  1e+02 1e-04 0.08685897 0.04239744
## 7  1e-03 1e-03 0.55115385 0.04842542
## 8  1e-02 1e-03 0.55115385 0.04842542
## 9  1e-01 1e-03 0.55115385 0.04842542
## 10 1e+00 1e-03 0.11487179 0.07368344
## 11 1e+01 1e-03 0.08685897 0.04239744
## 12 1e+02 1e-03 0.09448718 0.06062896
## 13 1e-03 1e-02 0.55115385 0.04842542
## 14 1e-02 1e-02 0.55115385 0.04842542
```

```
## 15 1e-01 1e-02 0.11230769 0.06739738
## 16 1e+00 1e-02 0.08685897 0.04239744
## 17 1e+01 1e-02 0.09711538 0.06390657
## 18 1e+02 1e-02 0.08923077 0.04705980
## 19 1e-03 1e+00 0.55115385 0.04842542
## 20 1e-02 1e+00 0.55115385 0.04842542
## 21 1e-01 1e+00 0.55115385 0.04842542
## 22 1e+00 1e+00 0.07929487 0.05064964
## 23 1e+01 1e+00 0.08179487 0.04345042
## 24 1e+02 1e+00 0.08179487 0.04345042
```

The best parameters were found to be .1 cost, 1 gamma, degree 3.

```
set.seed(100)
tune.svm <- tune(svm, mpg~., data=Auto, kernel='polynomial', ranges=list(cost=c(.001, .01, .1, 1, 10, 1
summary(tune.svm)
```

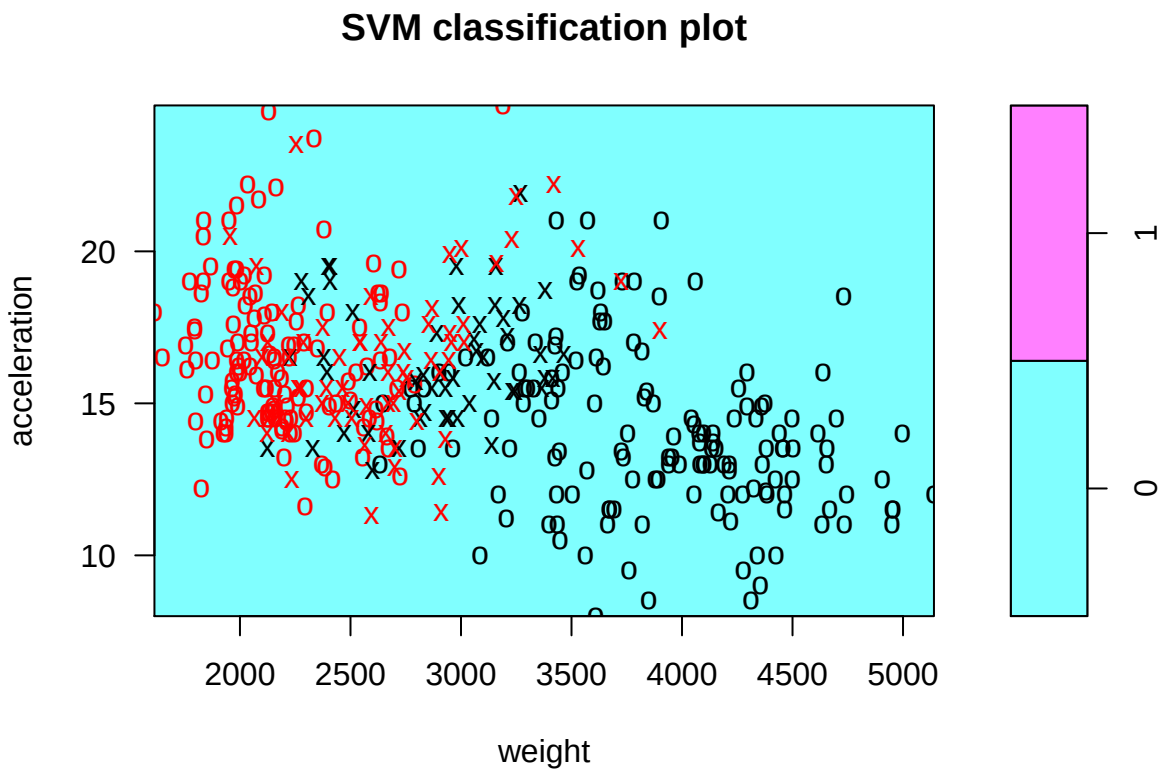
```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost gamma degree
##   0.1     1      3
##
## - best performance: 0.08416667
##
## - Detailed performance results:
##     cost gamma degree      error dispersion
## 1  1e-03 1e-04      1 0.55115385 0.04842542
## 2  1e-02 1e-04      1 0.55115385 0.04842542
## 3  1e-01 1e-04      1 0.55115385 0.04842542
## 4  1e+00 1e-04      1 0.55115385 0.04842542
## 5  1e+01 1e-04      1 0.12762821 0.07353560
## 6  1e+02 1e-04      1 0.08942308 0.04419023
## 7  1e-03 1e-03      1 0.55115385 0.04842542
## 8  1e-02 1e-03      1 0.55115385 0.04842542
## 9  1e-01 1e-03      1 0.55115385 0.04842542
## 10 1e+00 1e-03      1 0.12762821 0.07353560
## 11 1e+01 1e-03      1 0.08942308 0.04419023
## 12 1e+02 1e-03      1 0.09961538 0.05349840
## 13 1e-03 1e-02      1 0.55115385 0.04842542
## 14 1e-02 1e-02      1 0.55115385 0.04842542
## 15 1e-01 1e-02      1 0.12762821 0.07353560
## 16 1e+00 1e-02      1 0.08942308 0.04419023
## 17 1e+01 1e-02      1 0.09961538 0.05349840
## 18 1e+02 1e-02      1 0.10192308 0.05228803
## 19 1e-03 1e+00      1 0.12762821 0.07353560
## 20 1e-02 1e+00      1 0.08942308 0.04419023
## 21 1e-01 1e+00      1 0.09961538 0.05349840
## 22 1e+00 1e+00      1 0.10448718 0.05563850
## 23 1e+01 1e+00      1 0.12230769 0.05599171
## 24 1e+02 1e+00      1 0.12230769 0.04903630
## 25 1e-03 1e-04      2 0.55115385 0.04842542
```

```
## 26 1e-02 1e-04      2 0.55115385 0.04842542
## 27 1e-01 1e-04      2 0.55115385 0.04842542
## 28 1e+00 1e-04      2 0.55115385 0.04842542
## 29 1e+01 1e-04      2 0.55115385 0.04842542
## 30 1e+02 1e-04      2 0.55115385 0.04842542
## 31 1e-03 1e-03      2 0.55115385 0.04842542
## 32 1e-02 1e-03      2 0.55115385 0.04842542
## 33 1e-01 1e-03      2 0.55115385 0.04842542
## 34 1e+00 1e-03      2 0.55115385 0.04842542
## 35 1e+01 1e-03      2 0.55115385 0.04842542
## 36 1e+02 1e-03      2 0.49987179 0.10740057
## 37 1e-03 1e-02      2 0.55115385 0.04842542
## 38 1e-02 1e-02      2 0.55115385 0.04842542
## 39 1e-01 1e-02      2 0.55115385 0.04842542
## 40 1e+00 1e-02      2 0.49987179 0.10740057
## 41 1e+01 1e-02      2 0.31384615 0.06292886
## 42 1e+02 1e-02      2 0.28320513 0.07937325
## 43 1e-03 1e+00      2 0.31384615 0.06292886
## 44 1e-02 1e+00      2 0.28320513 0.07937325
## 45 1e-01 1e+00      2 0.17089744 0.04353021
## 46 1e+00 1e+00      2 0.18115385 0.04596372
## 47 1e+01 1e+00      2 0.21435897 0.07042613
## 48 1e+02 1e+00      2 0.25538462 0.09470427
## 49 1e-03 1e-04      3 0.55115385 0.04842542
## 50 1e-02 1e-04      3 0.55115385 0.04842542
## 51 1e-01 1e-04      3 0.55115385 0.04842542
## 52 1e+00 1e-04      3 0.55115385 0.04842542
## 53 1e+01 1e-04      3 0.55115385 0.04842542
## 54 1e+02 1e-04      3 0.55115385 0.04842542
## 55 1e-03 1e-03      3 0.55115385 0.04842542
## 56 1e-02 1e-03      3 0.55115385 0.04842542
## 57 1e-01 1e-03      3 0.55115385 0.04842542
## 58 1e+00 1e-03      3 0.55115385 0.04842542
## 59 1e+01 1e-03      3 0.55115385 0.04842542
## 60 1e+02 1e-03      3 0.55115385 0.04842542
## 61 1e-03 1e-02      3 0.55115385 0.04842542
## 62 1e-02 1e-02      3 0.55115385 0.04842542
## 63 1e-01 1e-02      3 0.55115385 0.04842542
## 64 1e+00 1e-02      3 0.55115385 0.04842542
## 65 1e+01 1e-02      3 0.30352564 0.09327468
## 66 1e+02 1e-02      3 0.25525641 0.08224070
## 67 1e-03 1e+00      3 0.10455128 0.05573152
## 68 1e-02 1e+00      3 0.09166667 0.04326699
## 69 1e-01 1e+00      3 0.08416667 0.03627957
## 70 1e+00 1e+00      3 0.09192308 0.03481162
## 71 1e+01 1e+00      3 0.09192308 0.04407224
## 72 1e+02 1e+00      3 0.09192308 0.04407224
```

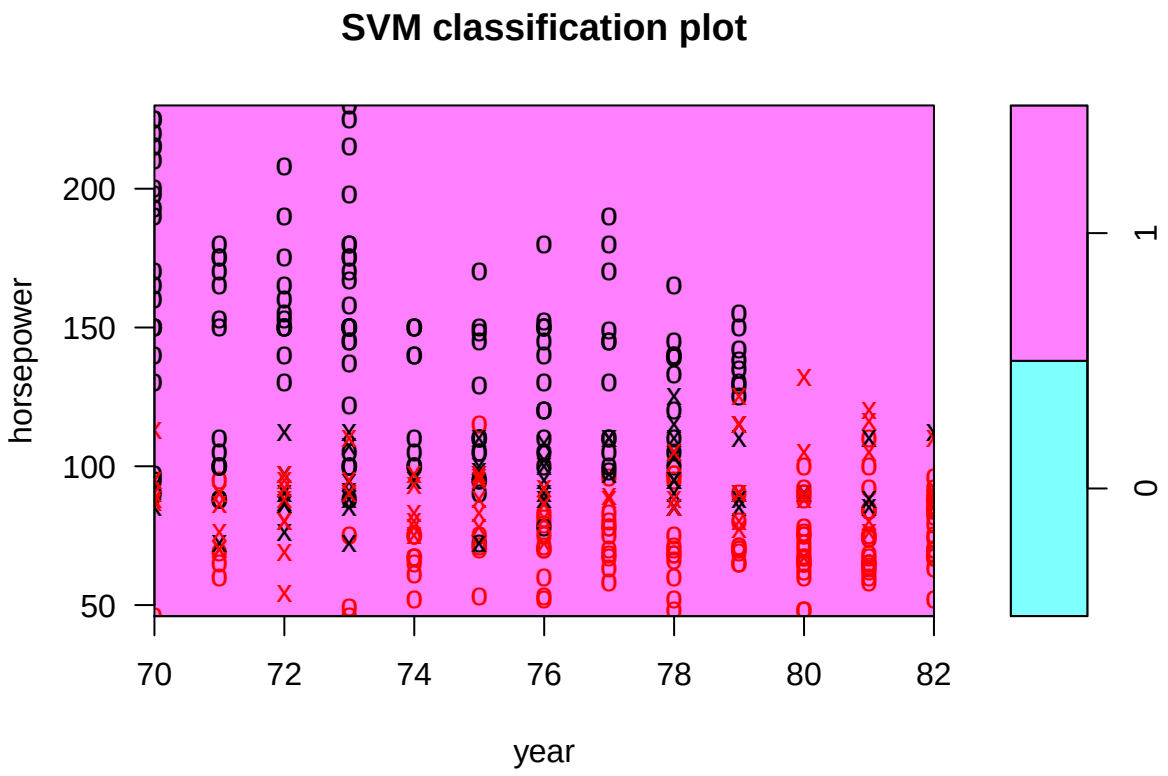
(d)

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

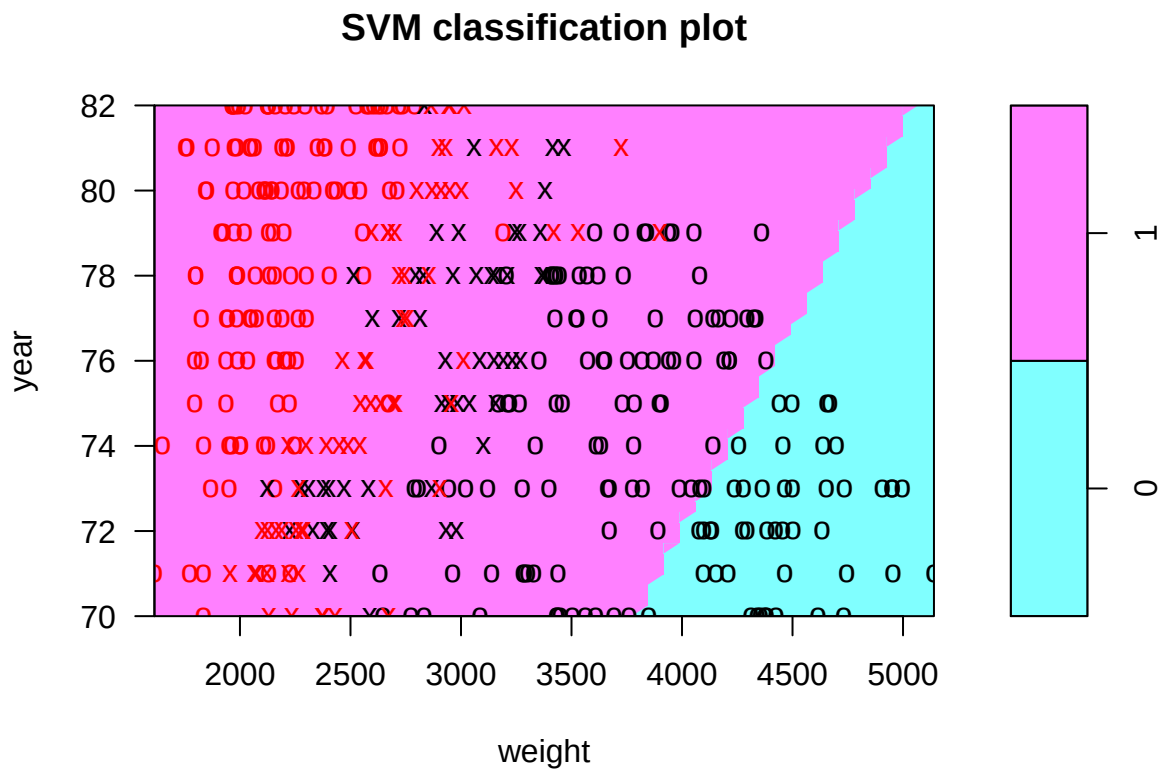
```
svm.Auto <- svm(mpg~., data=Auto, kernel='linear', cost=1, scale=TRUE)
plot(svm.Auto, Auto, acceleration~weight)
```



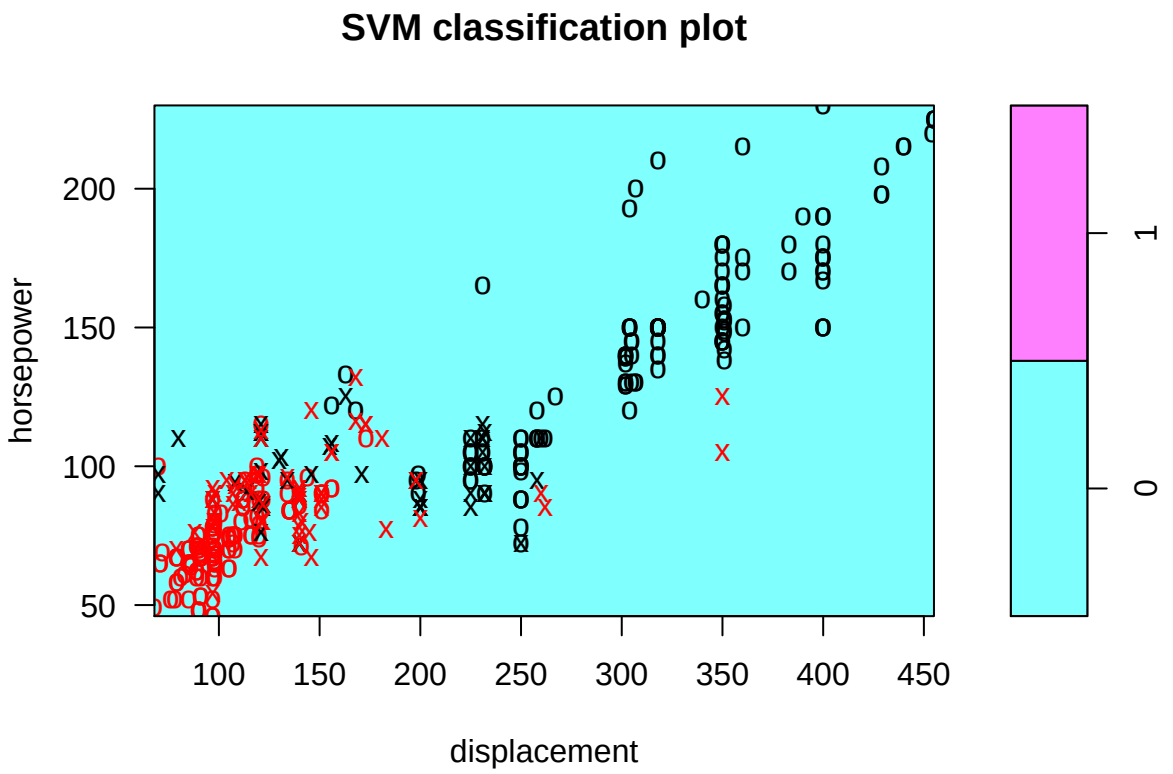

```
plot(svm.Auto, Auto, horsepower~year)
```



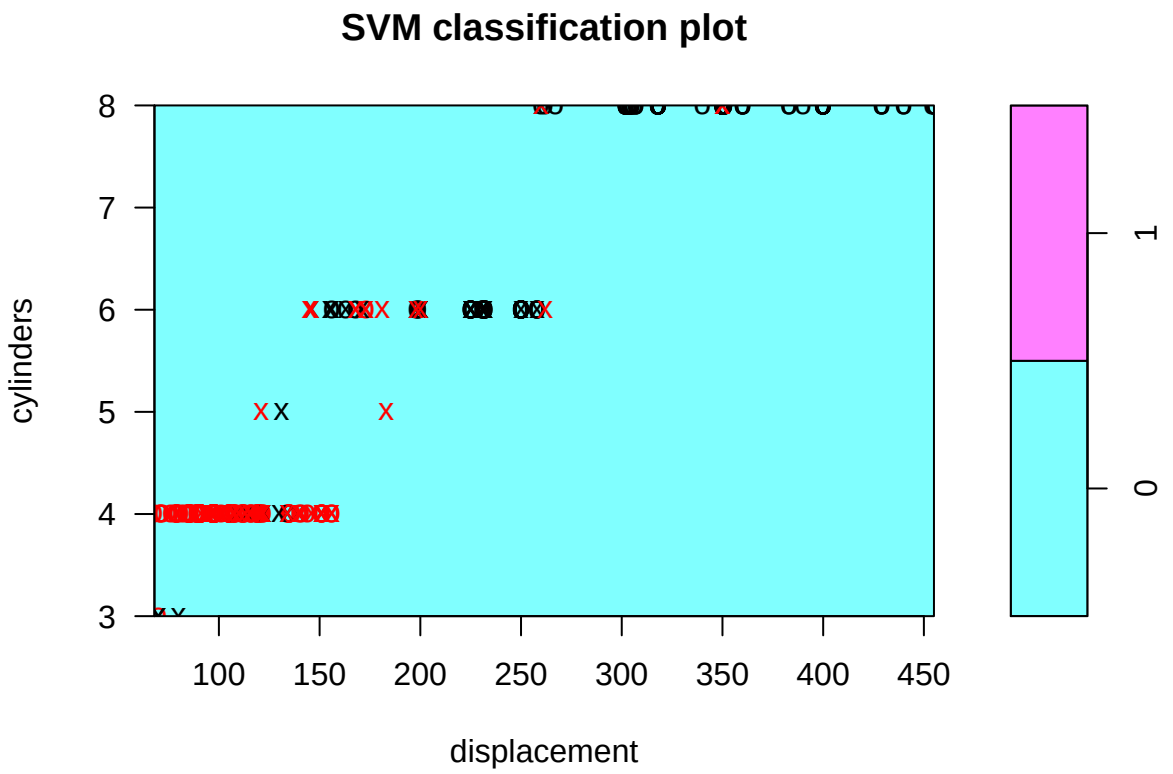

```
plot(svm.Auto, Auto, year~weight)
```



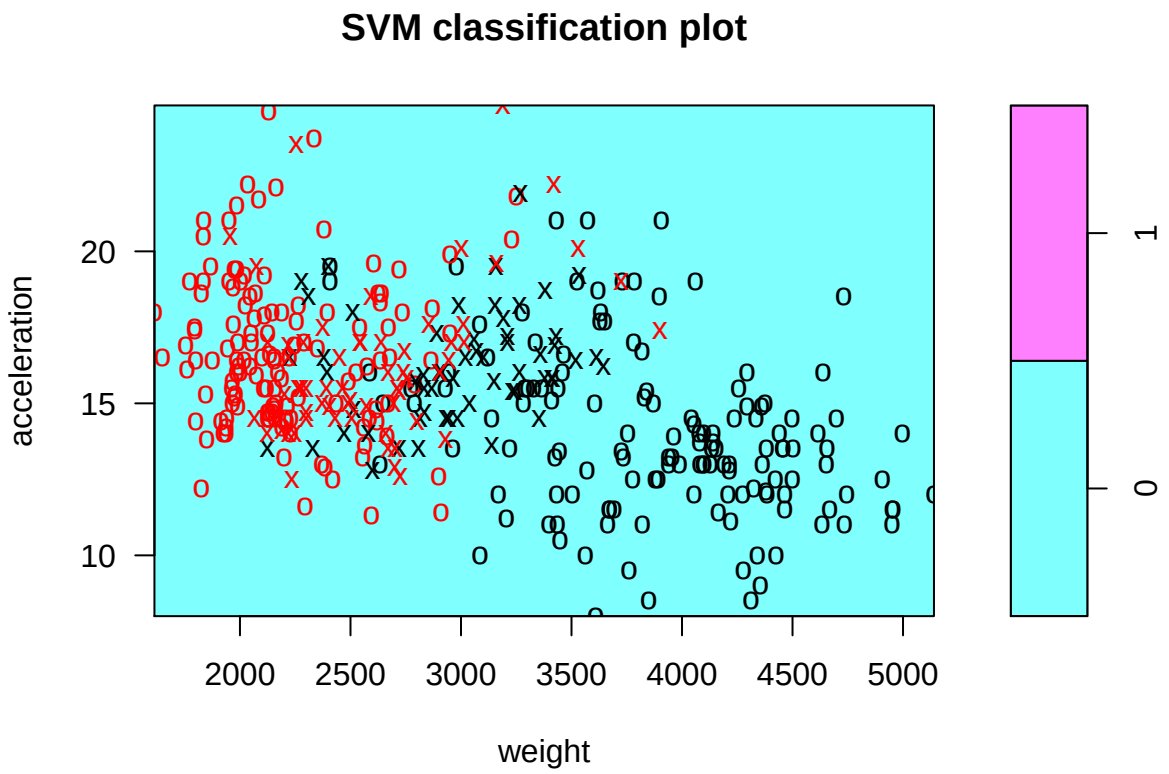

```
plot(svm.Auto, Auto, horsepower~displacement)
```



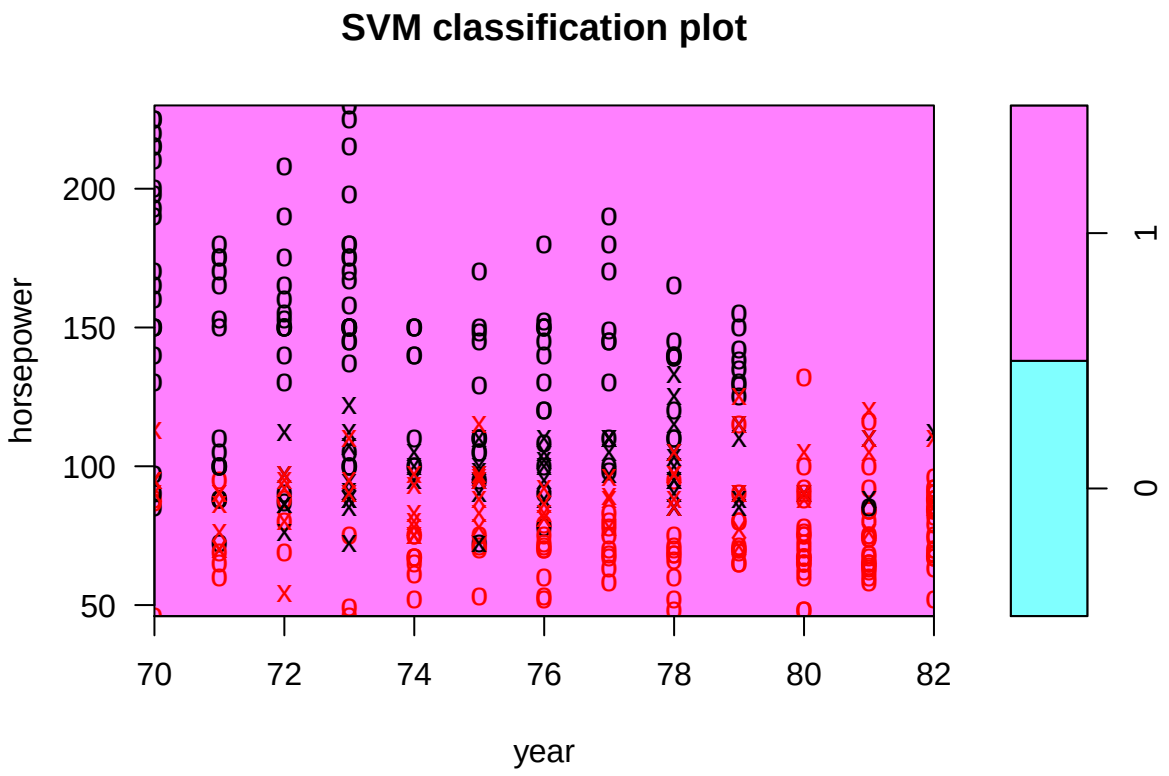

```
plot(svm.Auto, Auto, cylinders~displacement)
```



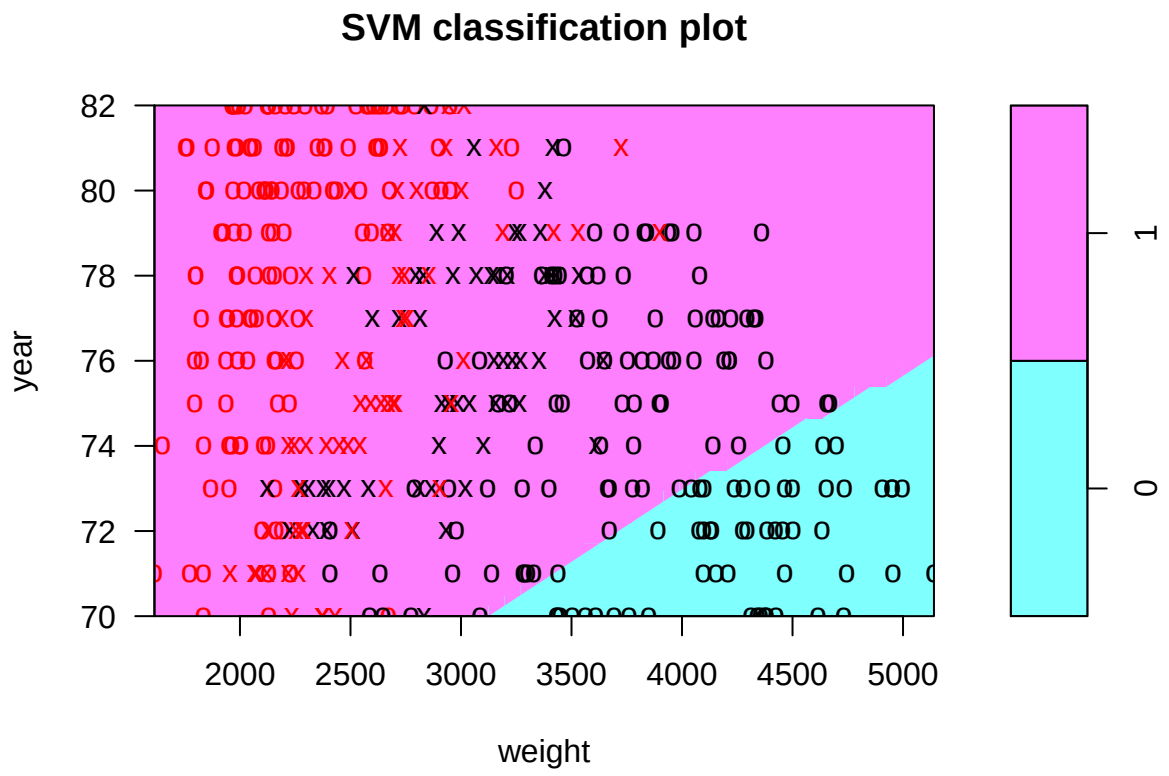

```
svm.Auto <- svm(mpg~., data=Auto, kernel='polynomial', cost=.1, gamma=1, degree=3, scale=TRUE)
plot(svm.Auto, Auto, acceleration~weight)
```



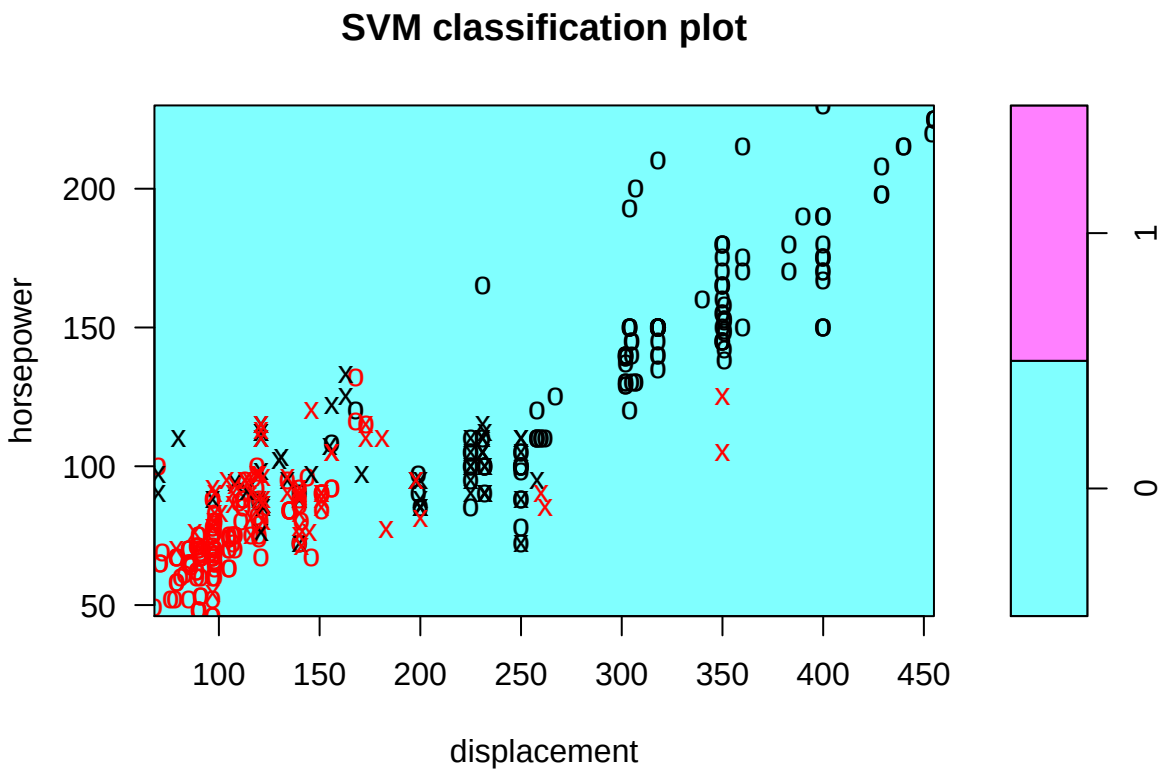

```
plot(svm.Auto, Auto, horsepower~year)
```



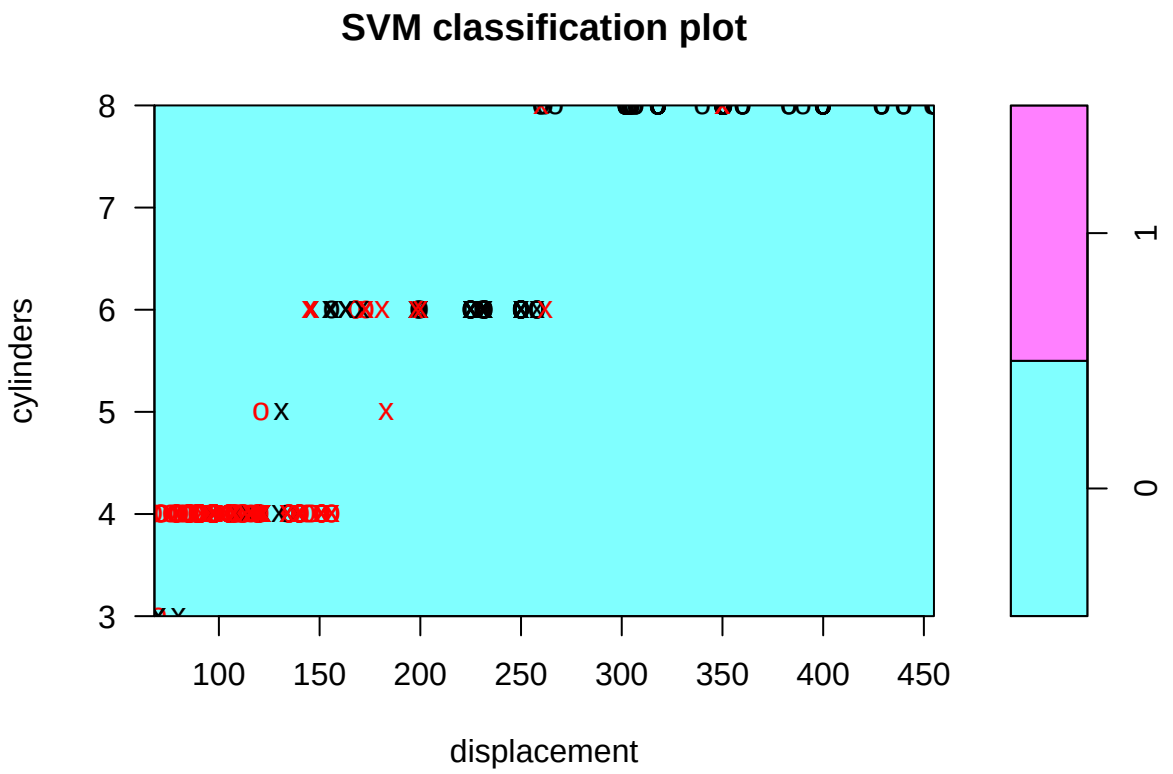

```
plot(svm.Auto, Auto, year~weight)
```



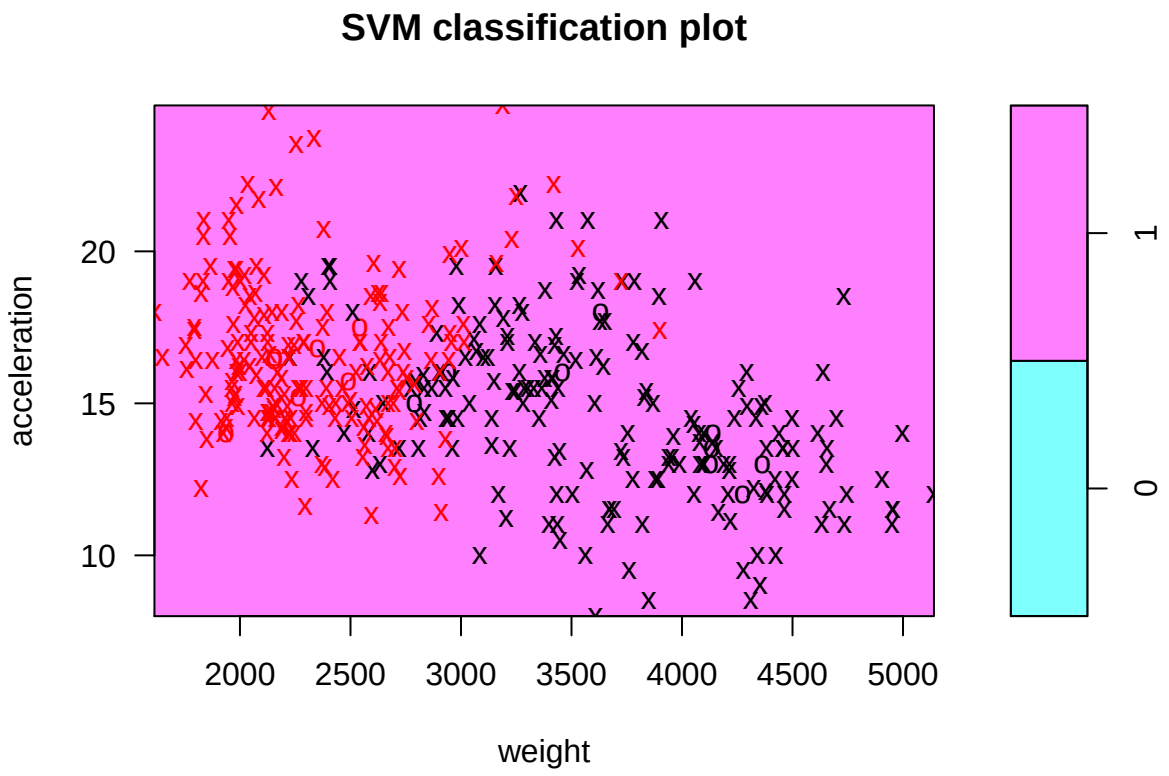

```
plot(svm.Auto, Auto, horsepower~displacement)
```



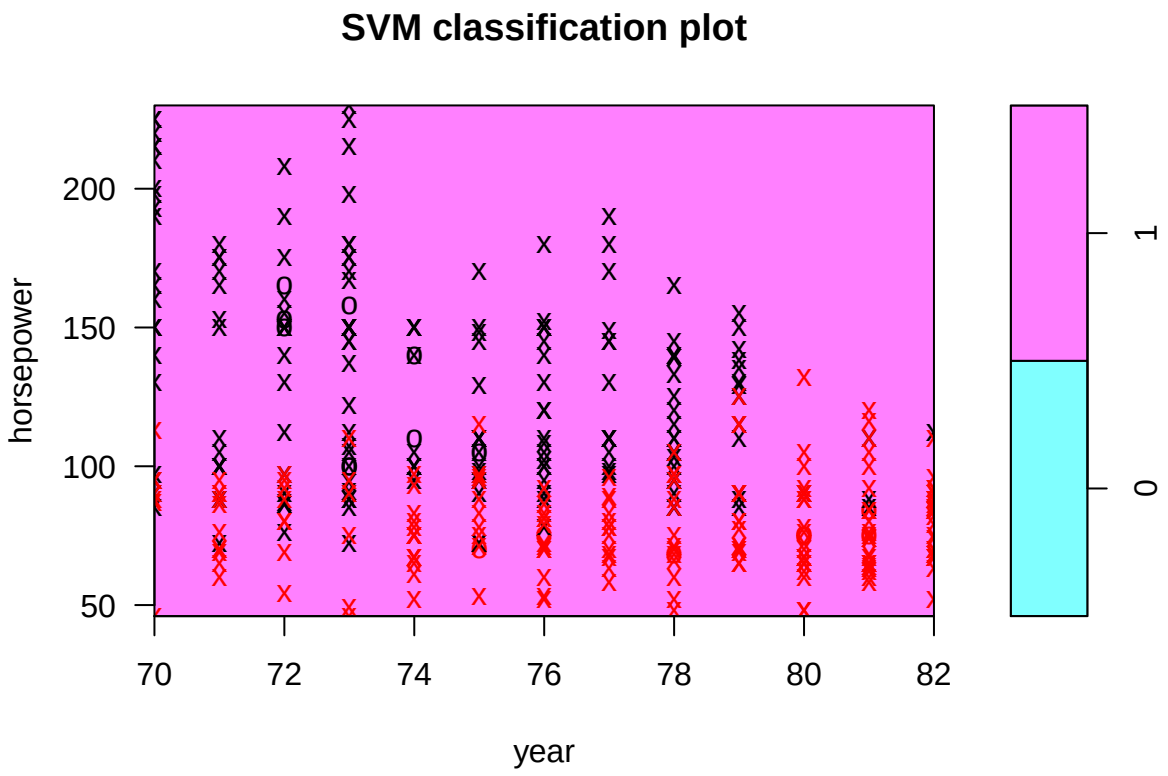

```
plot(svm.Auto, Auto, cylinders~displacement)
```



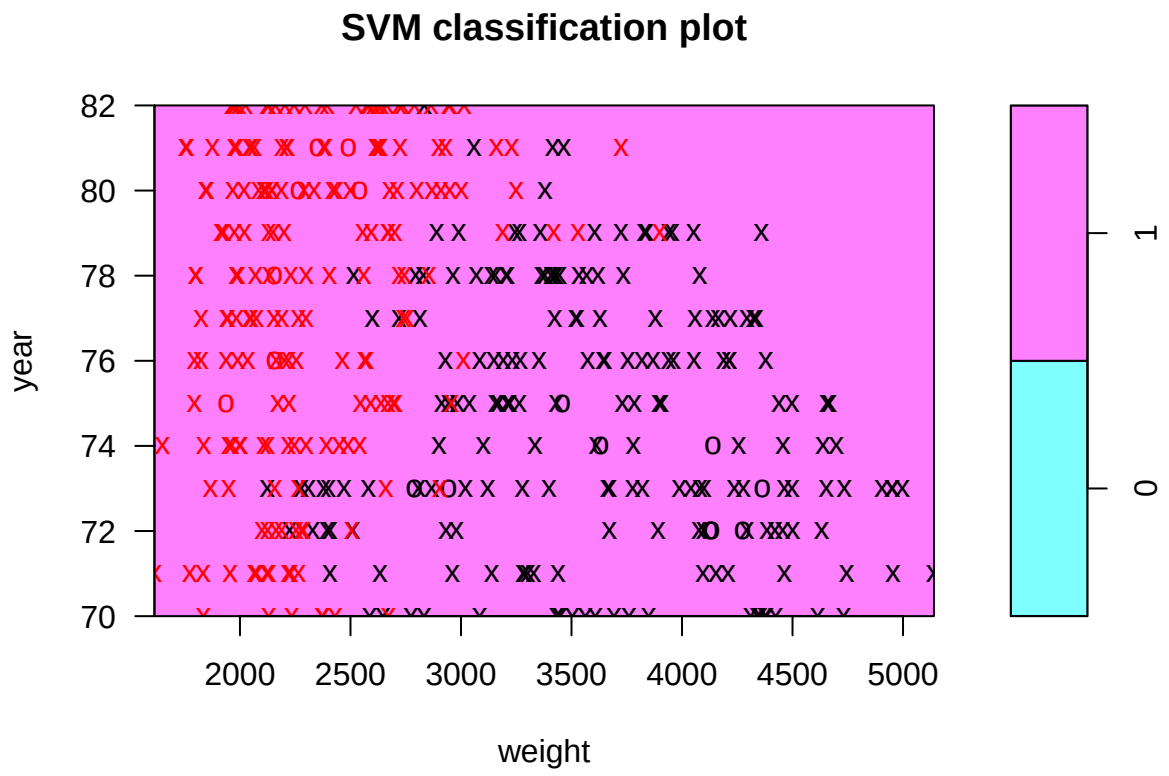

```
svm.Auto <- svm(mpg~., data=Auto, kernel='radial', cost=1, gamma=1, scale=TRUE)
plot(svm.Auto, Auto, acceleration~weight)
```



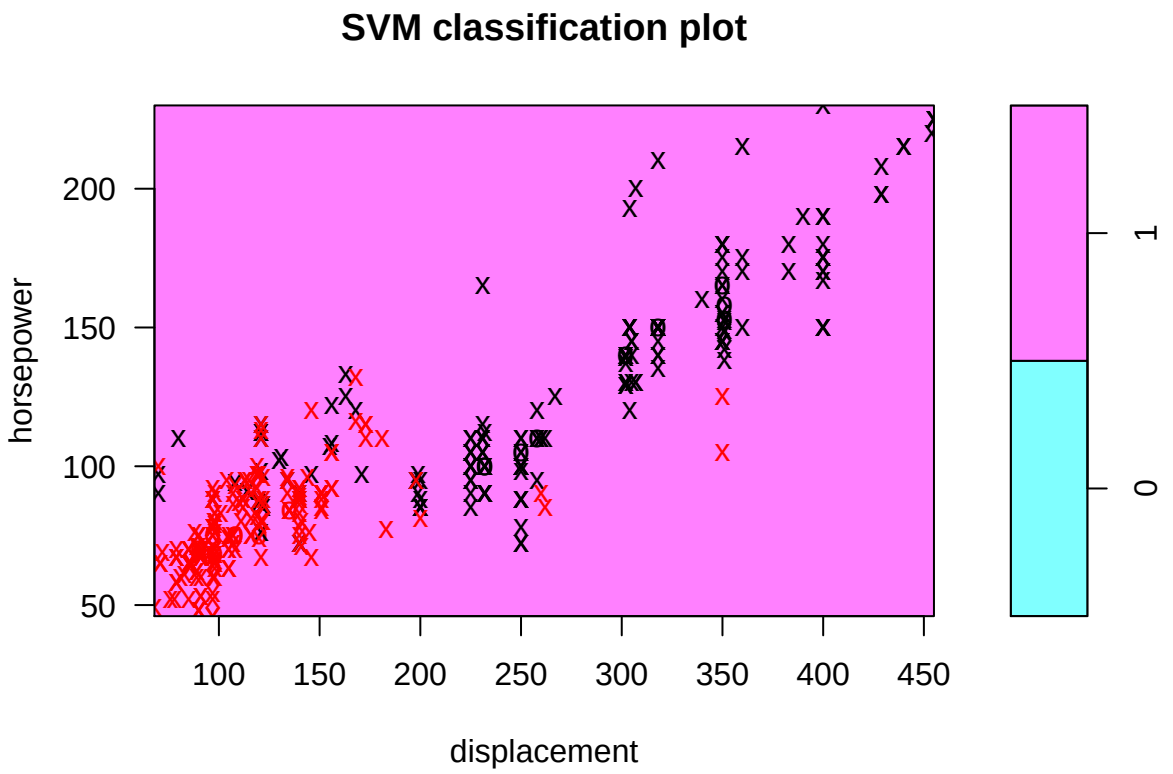

```
plot(svm.Auto, Auto, horsepower~year)
```



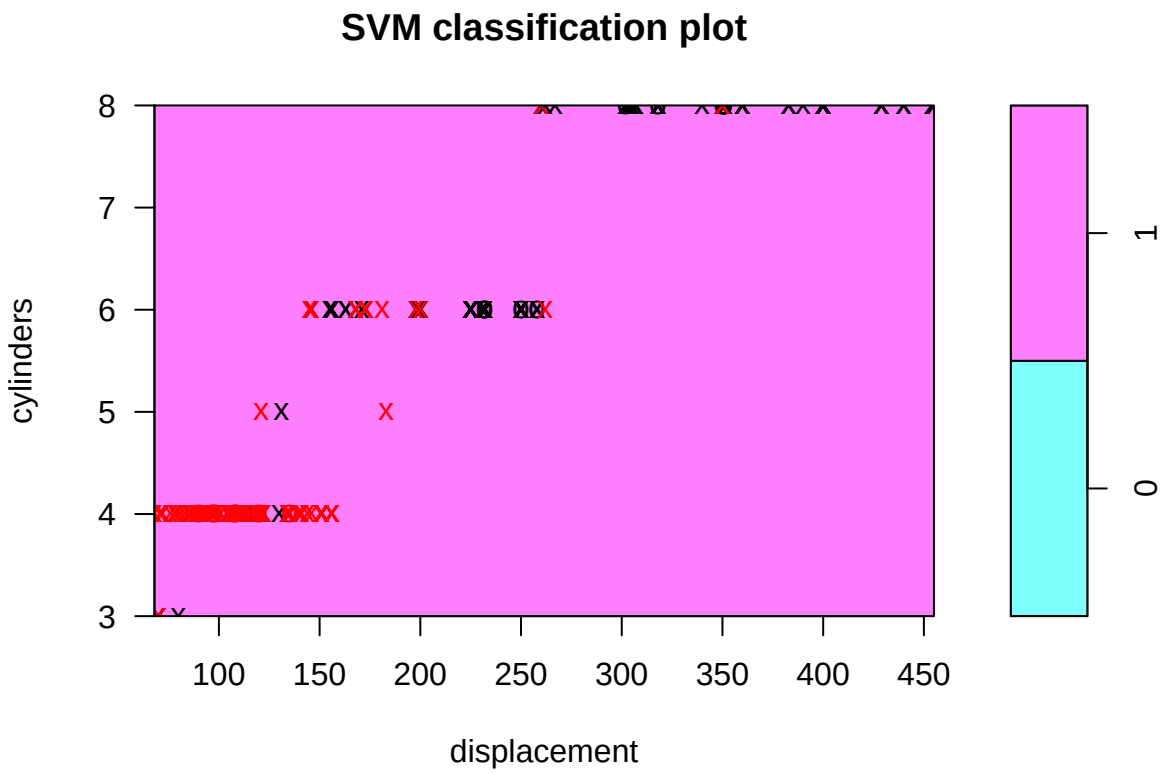

```
plot(svm.Auto, Auto, year~weight)
```




```
plot(svm.Auto, Auto, horsepower~displacement)
```




```
plot(svm.Auto, Auto, cylinders~displacement)
```



Book 8

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.

```
trainIdx <- sample(nrow(OJ), 800, replace=FALSE)
trainOJ <- OJ[trainIdx,]
testOJ <- OJ[-trainIdx,]
```

(b)

Fit a support vector classifier to the training data using cost=.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.

The summary statistics show a classification svm is used with a linear kernel, so it is a svc model. The cost is .01 and gamma is .055. The number of support vectors is 455.

```
svm.oj <- svm(Purchase~., data=trainOJ, cost=.01, kernel='linear')
summary(svm.oj)
```

```
##
## Call:
```

```
## svm(formula = Purchase ~ ., data = trainOJ, cost = 0.01, kernel = "linear")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  0.01
##       gamma:  0.05555556
##
## Number of Support Vectors:  455
##
##  ( 228 227 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

(c)

What are the training and test error rates?

Training error is 18.1%. Test error is 14.07%

```
# Training error rate
pred <- predict(svm.oj, newdata=trainOJ)
cm <- table(pred, trainOJ$Purchase)
cat('training error', (cm[2] + cm[3])/sum(cm))
```

```
## training error 0.18125
```

```
# Test error rate
pred <- predict(svm.oj, newdata=testOJ)
cm <- table(pred, testOJ$Purchase)
cat('\ntest error', (cm[2] + cm[3])/sum(cm))
```

```
##
## test error 0.1407407
```

(d)

Use the tune() function to select an optimal cost. Consider values in the range .01 to 10.

The optimal cost value is found to be 3.

```
set.seed(100)
tune.oj <- tune(svm, Purchase~., data=trainOJ, kernel='linear', ranges=list(cost=c(.01, .1, 1, 2, 3, 4,
summary(tune.oj)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##     3
```

```
##
## - best performance: 0.18
##
## - Detailed performance results:
##     cost   error dispersion
## 1   0.01 0.18750 0.03173239
## 2   0.10 0.18375 0.02889757
## 3   1.00 0.18375 0.03335936
## 4   2.00 0.18125 0.03397814
## 5   3.00 0.18000 0.03238227
## 6   4.00 0.18250 0.03184162
## 7   5.00 0.18250 0.03073181
## 8   6.00 0.18375 0.02949223
## 9   7.00 0.18375 0.02949223
## 10  8.00 0.18250 0.03073181
## 11  9.00 0.18375 0.02889757
## 12 10.00 0.18375 0.02889757
```

(e)

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

The training error is 17.6% and the test error is 14.07%.

```
svm.oj <- svm(Purchase~., data=trainOJ, cost=3, kernel='linear')
```

```
# Training error rate
pred <- predict(svm.oj, newdata=trainOJ)
cm <- table(pred, trainOJ$Purchase)
cat('training error', (cm[2] + cm[3])/sum(cm))
```

```
## training error 0.17625
```

```
# Test error rate
pred <- predict(svm.oj, newdata=testOJ)
cm <- table(pred, testOJ$Purchase)
cat('\ntest error', (cm[2] + cm[3])/sum(cm))
```

```
##
## test error 0.1407407
```

(f)

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

The summary statistics show a classification svm is used with a radial kernel, so it is a svm model. The cost is .01 and gamma is .055. The number of support vectors is 641.

```
set.seed(100)
svm.oj <- svm(Purchase~., data=trainOJ, cost=.01, kernel='radial')
summary(svm.oj)
```

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

```
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  0.01
##       gamma:  0.05555556
##
## Number of Support Vectors:  641
##
##  ( 322 319 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

Training error is 39.87% and test error is 36.29%.

```
# Training error rate
pred <- predict(svm.oj, newdata=trainOJ)
cm <- table(pred, trainOJ$Purchase)
cat('training error', (cm[2] + cm[3])/sum(cm))
```

```
## training error 0.39875
```

```
# Test error rate
pred <- predict(svm.oj, newdata=testOJ)
cm <- table(pred, testOJ$Purchase)
cat('\ntest error', (cm[2] + cm[3])/sum(cm))
```

```
##
## test error 0.362963
```

The optimal cost value is found to be 1.

```
set.seed(100)
tune.oj <- tune(svm, Purchase~., data=trainOJ, kernel='radial', ranges=list(cost=c(.01, .1, 1, 2, 3, 4,
summary(tune.oj)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##     1
##
## - best performance: 0.19125
##
## - Detailed performance results:
##    cost   error dispersion
## 1  0.01 0.39875 0.06908379
## 2  0.10 0.19375 0.03346329
## 3  1.00 0.19125 0.04168749
## 4  2.00 0.20000 0.03280837
## 5  3.00 0.19750 0.04199868
```

```
## 6   4.00 0.19750 0.04779877
## 7   5.00 0.19625 0.04825065
## 8   6.00 0.19625 0.04825065
## 9   7.00 0.19750 0.04993051
## 10  8.00 0.19625 0.04931827
## 11  9.00 0.19750 0.04851976
## 12 10.00 0.19750 0.04851976
```

```
svm.oj <- svm(Purchase~., data=trainOJ, cost=1, kernel='radial')
```

The training error is 16.25% and the test error is 12.59%.

```
# Training error rate
pred <- predict(svm.oj, newdata=trainOJ)
cm <- table(pred, trainOJ$Purchase)
cat('training error', (cm[2] + cm[3])/sum(cm))
```

```
## training error 0.1625
```

```
# Test error rate
pred <- predict(svm.oj, newdata=testOJ)
cm <- table(pred, testOJ$Purchase)
cat('\ntest error', (cm[2] + cm[3])/sum(cm))
```

```
##
## test error 0.1259259
```

(g)

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

The summary statistics show a classification svm is used with a polynomial kernel, so it is a svm model. The cost is .01, degree is 2, and gamma is .055. The number of support vectors is 644.

```
set.seed(100)
svm.oj <- svm(Purchase~., data=trainOJ, cost=.01, kernel='polynomial', degree=2)
summary(svm.oj)
```

```
##
## Call:
## svm(formula = Purchase ~ ., data = trainOJ, cost = 0.01, kernel = "polynomial",
##     degree = 2)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  polynomial
##        cost:  0.01
##      degree:  2
##       gamma:  0.05555556
##      coef.0:  0
##
## Number of Support Vectors:  644
##
##  ( 325 319 )
##
##
## Number of Classes:  2
```

```
##
## Levels:
##  CH MM
```

Training error is 37.62% and test error is 35.18%.

```
# Training error rate
pred <- predict(svm.oj, newdata=trainOJ)
cm <- table(pred, trainOJ$Purchase)
cat('training error', (cm[2] + cm[3])/sum(cm))
```

```
## training error 0.37625
```

```
# Test error rate
pred <- predict(svm.oj, newdata=testOJ)
cm <- table(pred, testOJ$Purchase)
cat('\ntest error', (cm[2] + cm[3])/sum(cm))
```

```
##
## test error 0.3518519
```

The optimal cost value is found to be 8.

```
set.seed(100)
tune.oj <- tune(svm, Purchase~., data=trainOJ, kernel='polynomial', ranges=list(cost=c(.01, .1, 1, 2, 3
summary(tune.oj)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##     8
##
## - best performance: 0.20125
##
## - Detailed performance results:
##     cost   error dispersion
## 1   0.01 0.39500 0.07100469
## 2   0.10 0.34000 0.05197489
## 3   1.00 0.20750 0.03496029
## 4   2.00 0.20625 0.03644345
## 5   3.00 0.20625 0.04050463
## 6   4.00 0.20250 0.03476109
## 7   5.00 0.20375 0.03283481
## 8   6.00 0.20125 0.03793727
## 9   7.00 0.20250 0.03622844
## 10  8.00 0.20125 0.03557562
## 11  9.00 0.20250 0.03425801
## 12 10.00 0.20250 0.03670453
```

```
svm.oj <- svm(Purchase~., data=trainOJ, cost=8, kernel='polynomial', degree=2)
```

The training error is 17.00% and the test error is 15.55%.

```r
# Training error rate
pred <- predict(svm.oj, newdata=trainOJ)
cm <- table(pred, trainOJ$Purchase)
cat('training error', (cm[2] + cm[3])/sum(cm))
```

```
## training error 0.17
```

```r
# Test error rate
pred <- predict(svm.oj, newdata=testOJ)
cm <- table(pred, testOJ$Purchase)
cat('\ntest error', (cm[2] + cm[3])/sum(cm))
```

```
##
## test error 0.1555556
```

(h)

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

The radial kernel gave the lowest test error when optimized. Therefore, this is the best svm model.

Extra 63

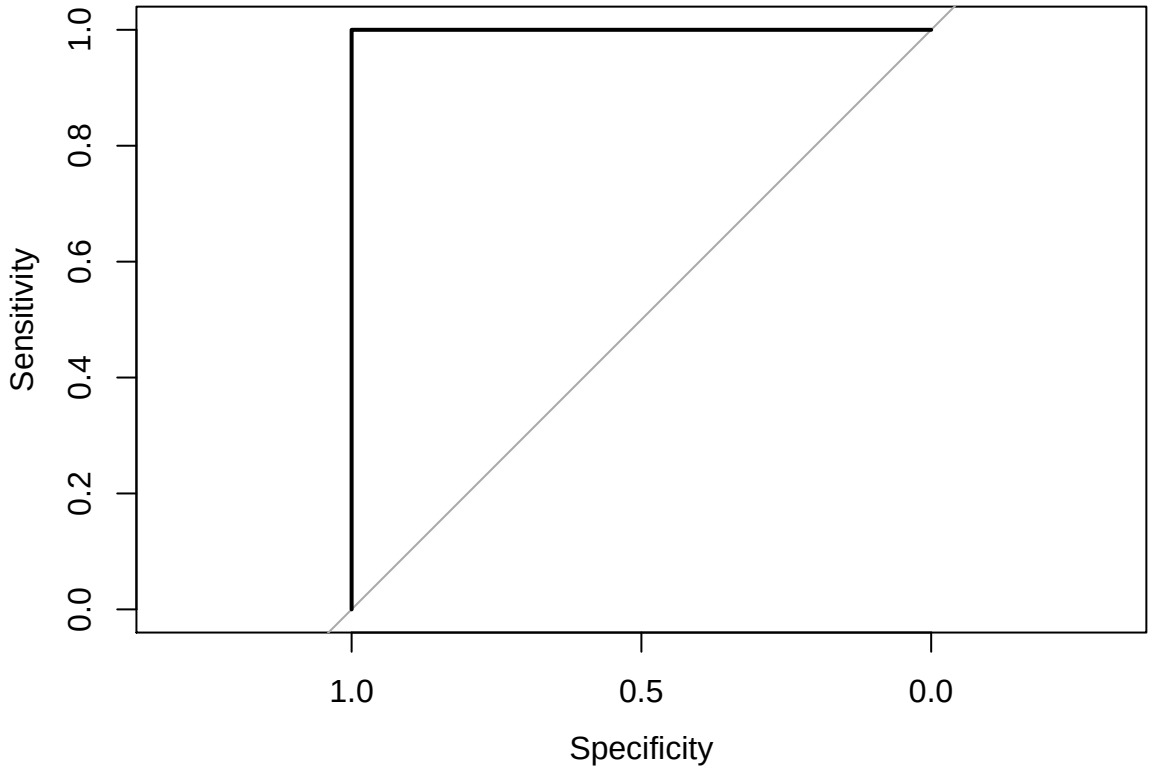
In this problem, we use the BreastCancer data, which comes as part of the package mlbench. Install the package and read the description of the data.

We want to predict the class of an observation (benign or malignant).

(a)

Fit a logistic model to the data. Plot the ROC curve.

```r
log.bc <- suppressWarnings(glm(y~., data=BreastCancer[-1], family='binomial', maxit=1000))
pred <- predict(log.bc, BreastCancer[-1], type='response')
plot(roc(BreastCancer$y, pred))
```




```
auc(roc(BreastCancer$y, pred))
```

```
## Area under the curve: 1
```

(b)

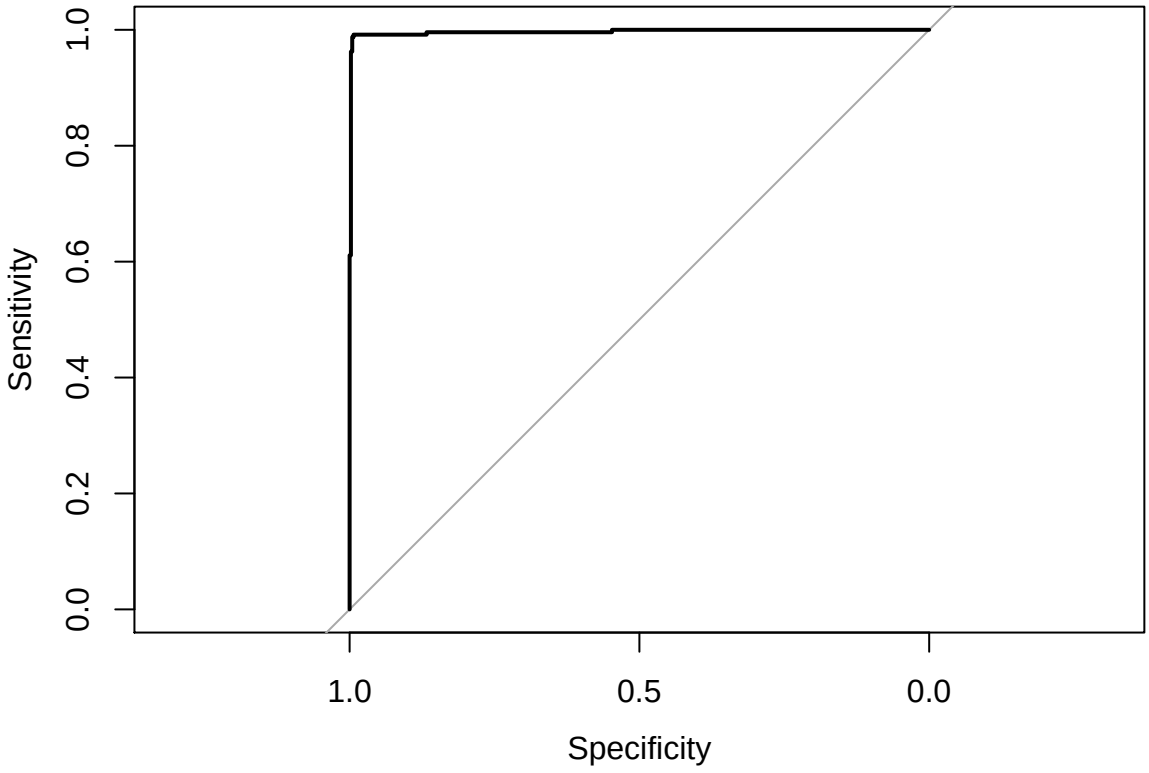
Use a support vector classifier (linear kernels) to classify. Plot the ROC curve in the plot oyu made in part a

```
tune.bc <- tune(svm, y~., data=BreastCancer[-1], kernel='linear', ranges=list(cost=c(.001, .01, 1, 10,
tune.bc$best.parameters
```

```
##   cost
## 3    1
```

```
BreastCancer <- na.omit(BreastCancer)
# Class 1 = malignant
BreastCancer$y <- as.factor(ifelse(BreastCancer$y == 'malignant', 1, 0))
```

```
svm.bc <- svm(y~., data=BreastCancer[-1], kernel='linear', cost=1, decision.values=T)
pred <- as.vector(attributes(predict(svm.bc, BreastCancer[-1], decision.values=TRUE))$decision.values)
plot(roc(BreastCancer$y, pred))
```



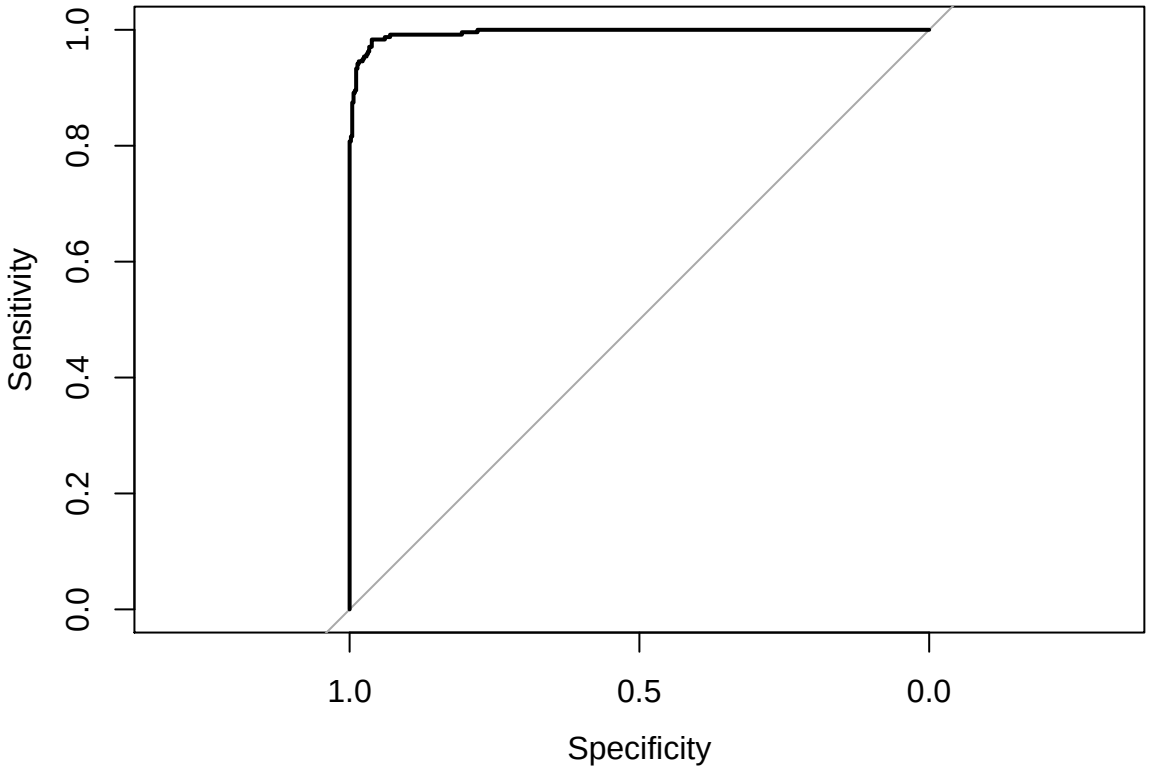

```
auc(roc(BreastCancer$y, pred))
```

```
## Area under the curve: 0.9966
```

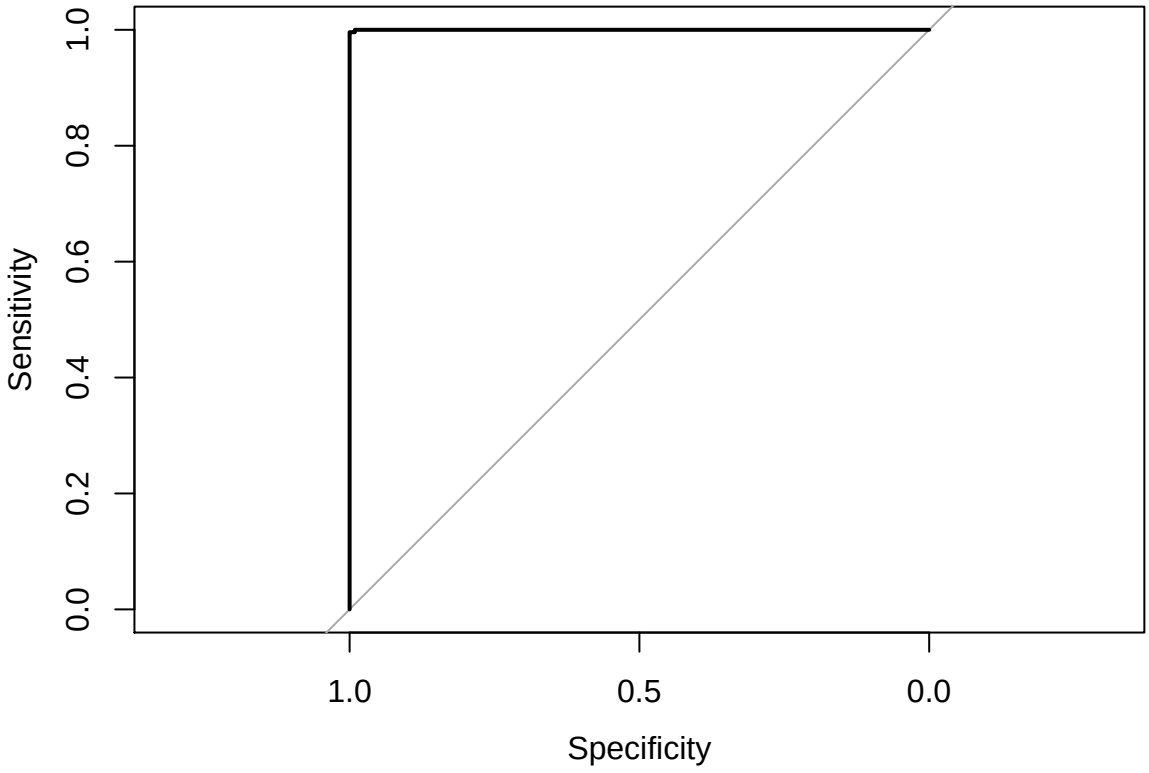
(c)

Use SVMs with polynomial kernels for the same classification task, with several different degrees and otherwise using the same hyper-parameters. Plot the ROC curves.

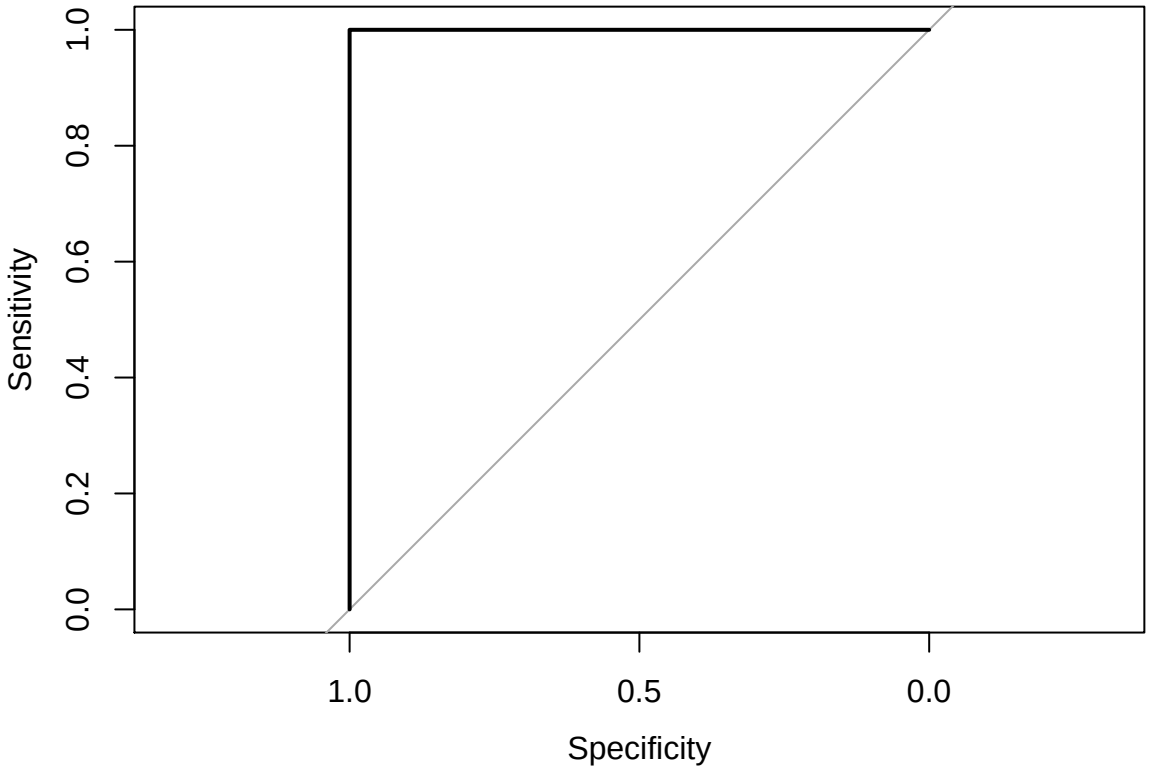
```
svm.bc <- svm(y~., data=BreastCancer[-1], kernel='polynomial', cost=1, degree=2, decision.values=T)
pred <- as.vector(attributes(predict(svm.bc, BreastCancer[-1], decision.values=TRUE))$decision.values)
plot(roc(BreastCancer$y, pred))
```




```
svm.bc <- svm(y~., data=BreastCancer[-1], kernel='polynomial', cost=1, degree=6, decision.values=T)
pred <- as.vector(attributes(predict(svm.bc, BreastCancer[-1], decision.values=TRUE))$decision.values)
plot(roc(BreastCancer$y, pred))
```




```
svm.bc <- svm(y~., data=BreastCancer[-1], kernel='polynomial', cost=1, degree=10, decision.values=T)
pred <- as.vector(attributes(predict(svm.bc, BreastCancer[-1], decision.values=TRUE))$decision.values)
plot(roc(BreastCancer$y, pred))
```



(d)

Compare the results of a), b), and c). Are the support vector classifier and logistic regression comparable? Is SVM better? Does this appear to depend on the degree?

The support vector classifier and logistic regression are basically identical in performance, but the logistic regression has an auc of 1 and svc has auc of .9966. SVM performs better with higher degrees. SVM with degree of 2 performs worst than logistic regression and svc.

Extra 66

This problem uses the MNIST image classification data, available as mnist_all.RData that were used earlier. We want to distinguish between 3 and 8. Extract the relevant training and test data and place them in suitable data frames. Remove all variables (pixels) with 0 variance from the training data and remove these also from the test data.

```
train.x <- train$x[(train$y == 3) | (train$y == 8),]
train.y <- train$y[train$y == 3 | train$y == 8]
train.y <- as.factor(as.numeric(train.y == 3)) # Class 1 = 3; class 0 = 8

test.x <- test$x[(test$y == 3) | (test$y == 8),]
test.y <- test$y[test$y == 3 | test$y == 8]
test.y <- as.factor(as.numeric(test.y == 3))

vars <- apply(train.x, MARGIN=2, var)
var.0 <- vars != 0
```

```
train.x <- train.x[,var.0]
test.x <- test.x[,var.0]

train.df <- data.frame(train.x, y=train.y)
test.df <- data.frame(test.x, y=test.y)
```

(a)

Fit a random forest model to the training data. Experiment with hyperparameters until you get a very good classification on the training data. Then evaluate the model on the test data and make a confusion matrix.

The training misclassification error rate is 0%. The test misclassification rate is 1.91%.

```
set.seed(100)
forest.mnist <- randomForest(y~., data=train.df, mtry=5, ntree=200)
```

```
# Training misclassification error rate
pred <- predict(forest.mnist, newdata=train.df, type='class')
cm <- table(pred, train.df$y)
cm
```

```
##
## pred    0    1
##    0 5851    0
##    1    0 6131
```

```
(cm[2] + cm[3])/sum(cm)
```

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

```
# Test misclassification error rate
pred <- predict(forest.mnist, newdata=test.df, type='class')
cm <- table(pred, test.df$y)
cm
```

```
##
## pred   0   1
##    0 958  22
##    1  16 988
```

```
(cm[2] + cm[3])/sum(cm)
```

```
## [1] 0.01915323
```

(b)

Can you get a similar or better performance on the training data if you use an SVM classifier with kernel=‘radial’? Experiment with hyperparameters. Do not report all trials, only the best result. Then evaluate the model on the test data and make a confusion matrix.

I was able to get a svm model to perform better than the random forest model. The training error is roughly 0% and the test error is 1.3%, which is lower than the random forest model.

```
set.seed(100)
svm.mnist <- svm(y~., data=train.df, kernel='radial', cost=10)
```

```
# Training misclassification error rate
pred <- predict(svm.mnist, newdata=train.df, type='class')
```

```
cm <- table(pred, train.df$y)
cm
```

```
##
## pred    0    1
##    0 5851    1
##    1    0 6130
```

```
(cm[2] + cm[3])/sum(cm)
```

```
## [1] 8.345852e-05
```

```
# Test misclassification error rate
pred <- predict(svm.mnist, newdata=test.df, type='class')
cm <- table(pred, test.df$y)
cm
```

```
##
## pred    0    1
##    0  957    9
##    1   17 1001
```

```
(cm[2] + cm[3])/sum(cm)
```

```
## [1] 0.01310484
```

(c)

Compare the results of a) and b). Is there a clear difference in performance? Do any of these methods tend to overfit? DO there runtimes differ substantially?

There is no clear difference in performance, however, the svm model performs slightly better. Both models have roughly perfect classification test error. These methods overfit slightly because the test error is not lower than the training error and the training error is 0. However, the test error is still very low. The SVM model took a substantially longer time than the random forest model. For this reason, the random forest model is preferred.