# HW6_SVM_Shuting_Li

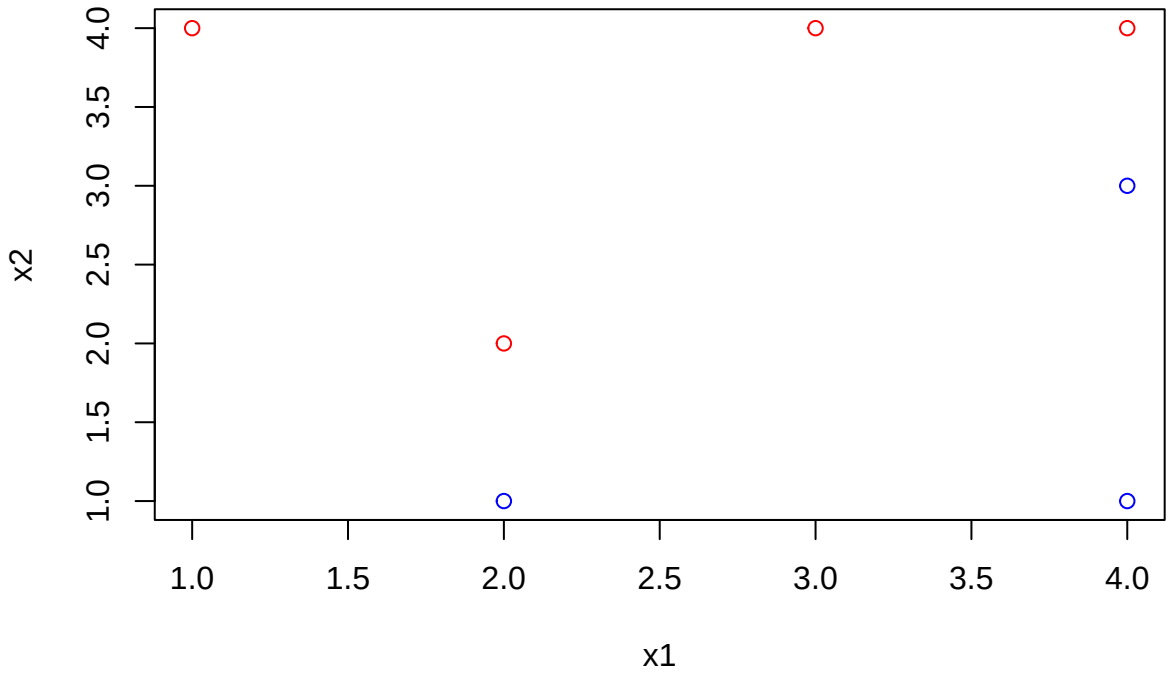
Shuting

3/09/2022

## 9.3

### (a)

```
x1 = c(3, 2, 4, 1, 2, 4, 4)
x2 = c(4, 2, 4, 4, 1, 3, 1)
colors = rep(c("red", "blue"),c(4,3))
plot(x1, x2, col = colors)
```

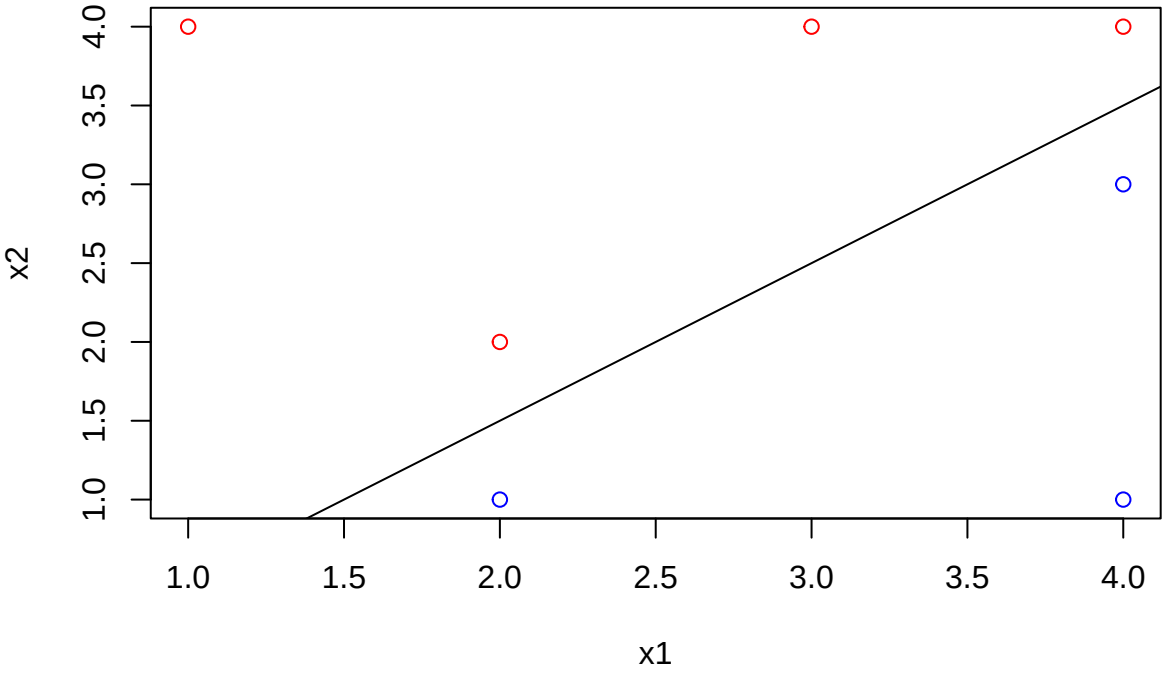



**Answer:**

### (b)

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

```
p1 <- c(2,(1+2)/2)
p2 <- c(4,(3+4)/2)
a <- ((3+4)/2 - (1+2)/2)/(4-2)
b <- (1+2)/2 - 2*a
plot(x1, x2, col = colors)
abline(b,a)
```




**Answer:**

**(c)**

Describe the classification rule for the maximal margin classifier. It should be something along the lines of "Classify to Red if $_0 + _1X1 + _2X2 > 0$, and classify to Blue otherwise." Provide the values for $_0$, $_1$, and $_2$.

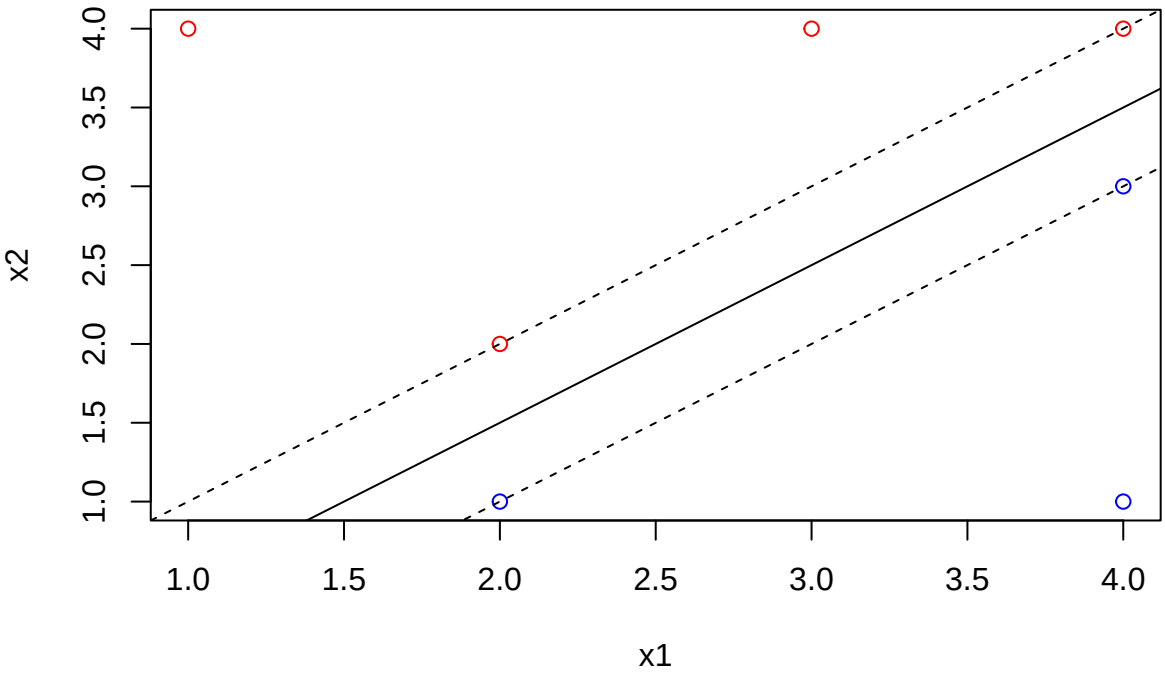
```
beta0 <- 0.5
beta1 <- -1
beta2 <- 1
```

**Answer:**

**(d)**

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

```
plot(x1, x2, col = colors)
abline(b, a)
abline(-1, 1, lty = 2)
abline(0, 1, lty = 2)
```

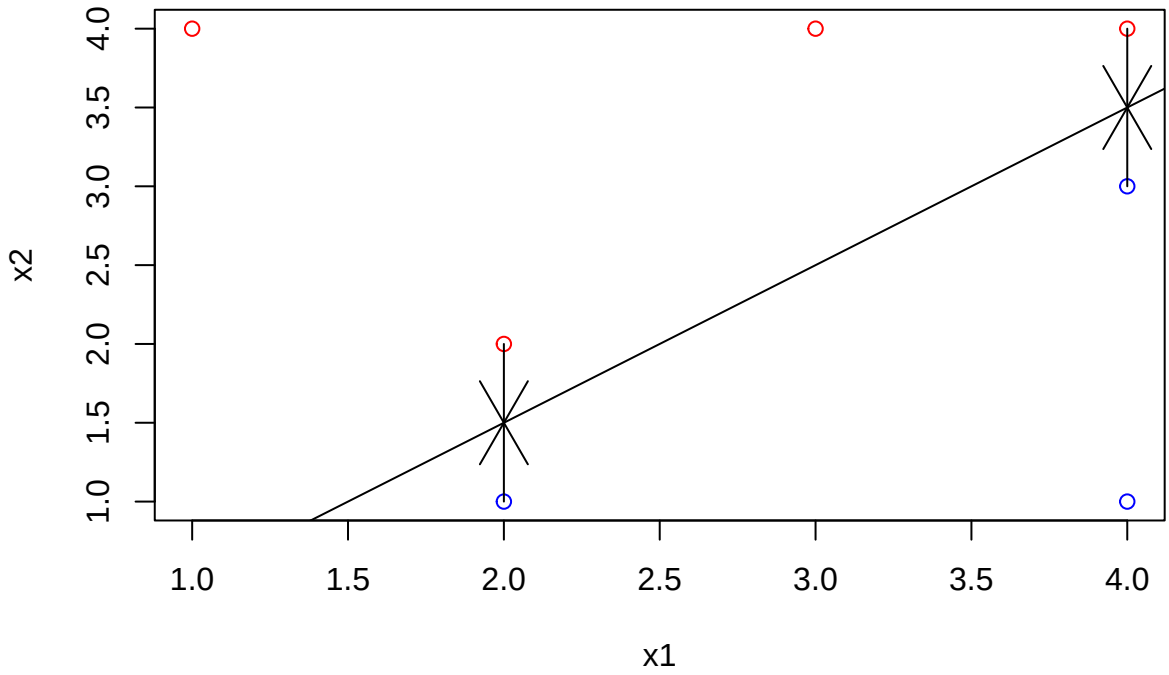



**Answer:**

**(e)**

Indicate the support vectors for the maximal margin classifier.

```
plot(x1, x2, col = colors)
abline(b, a)
arrows(2, 1, 2, 1.5)
arrows(2, 2, 2, 1.5)
arrows(4, 4, 4, 3.5)
arrows(4, 3, 4, 3.5)
```



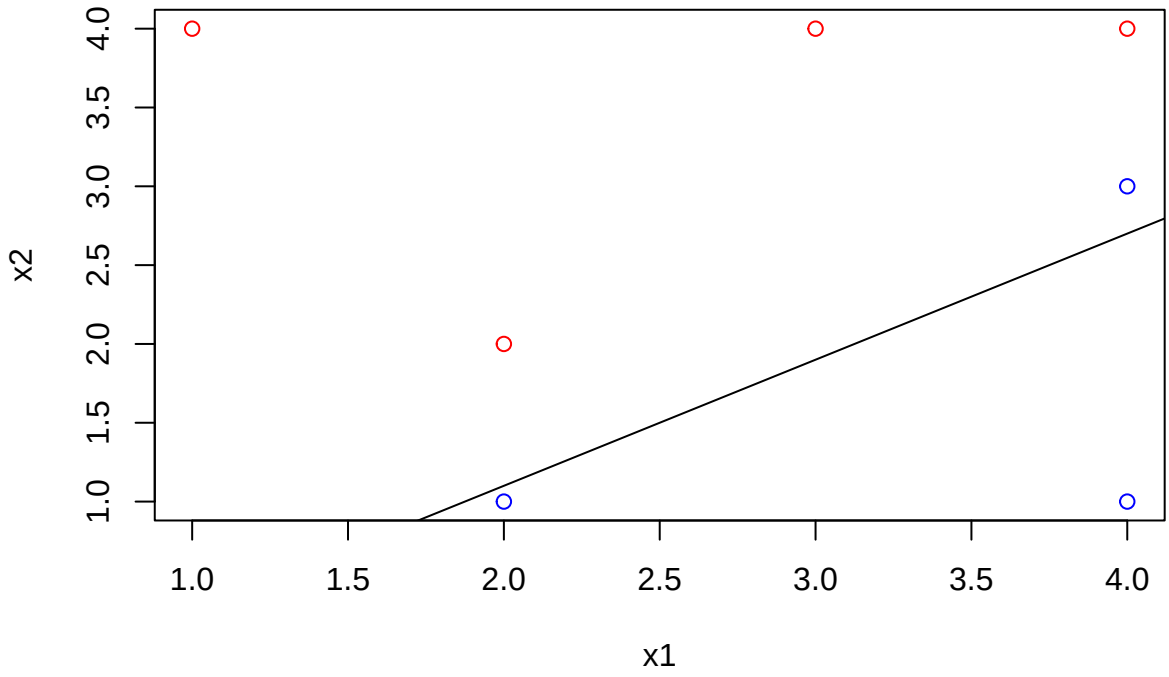

**Answer:**
### (f) Argue that a slight movement of the seventh observation would not affect the maximal margin hyperplane.

**Answer:** The seventh observation lies outside of the margin for the maximal margin hyperplane.

### (g)

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

```
plot(x1, x2, col = colors)
abline(b, 0.8)
```

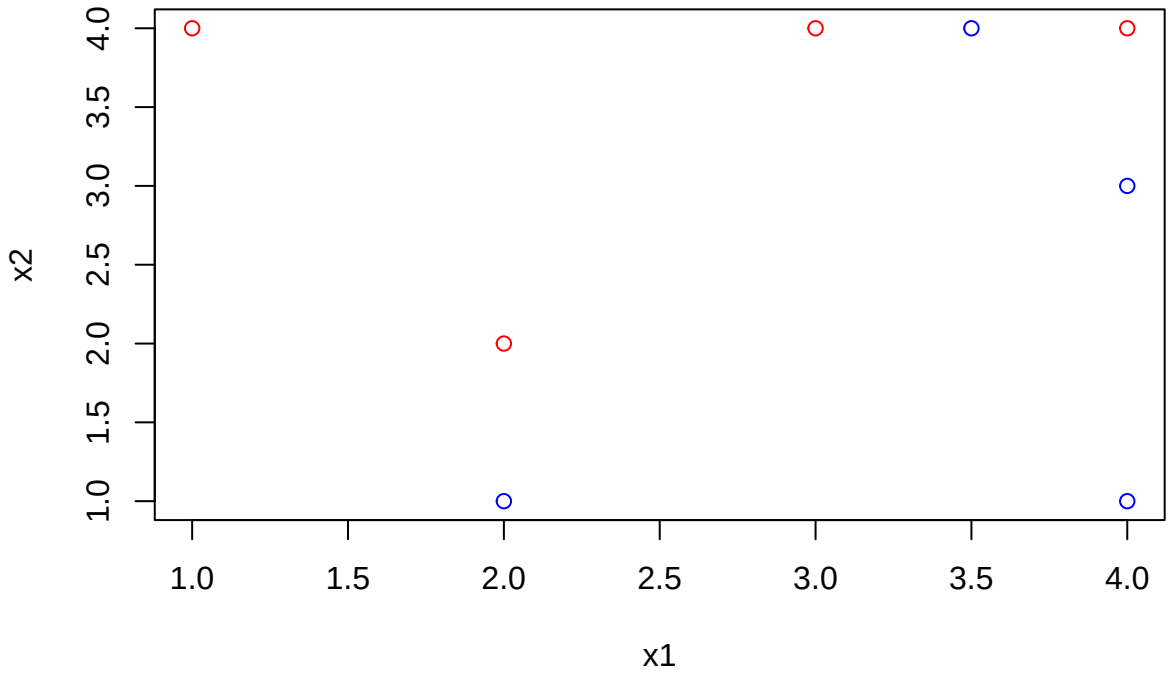



**Answer:**

**(h)**

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

```
x1 <- append(x1,3.5)
x2 <- append(x2,4)
colors <- append(colors,"blue")
plot(x1,x2,col=colors)
```




**Answer:**

## 9.5

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.

### (a)

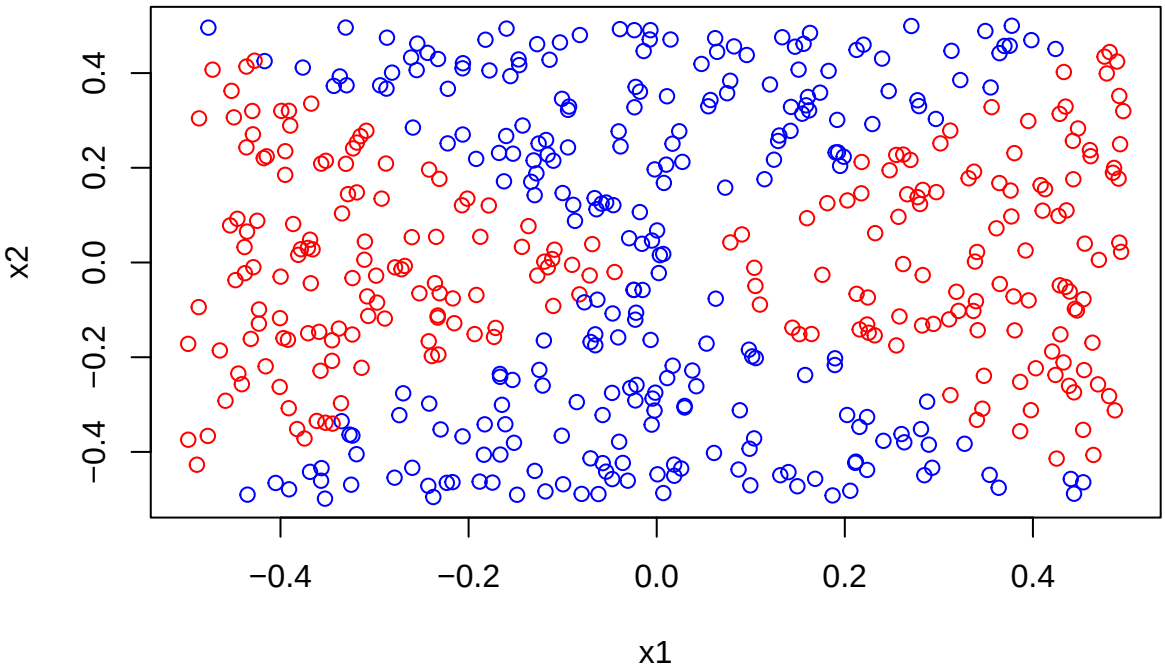
```
set.seed(1)
x1 = runif(500) - 0.5
x2 = runif(500) - 0.5
y = 1 * (x1^2 - x2^2 > 0)
```

**Answer:**

### (b)

Plot the observations, colored according to their class labels. Your plot should display X1 on the x-axis, and X2 on the y- axis.

```
color <- rep(NA,length(y))
color[y==0] <- "blue"
color[y==1] <- "red"
plot(x1,x2,col=color)
```




**Answer:**

### (c) Fit a logistic regression model to the data, using X1 and X2 as predictors.

```
fit.logis <- glm(y~x1+x2, data=data.frame(x1,x2,y),family = binomial)
summary(fit.logis)
```

**Answer:**

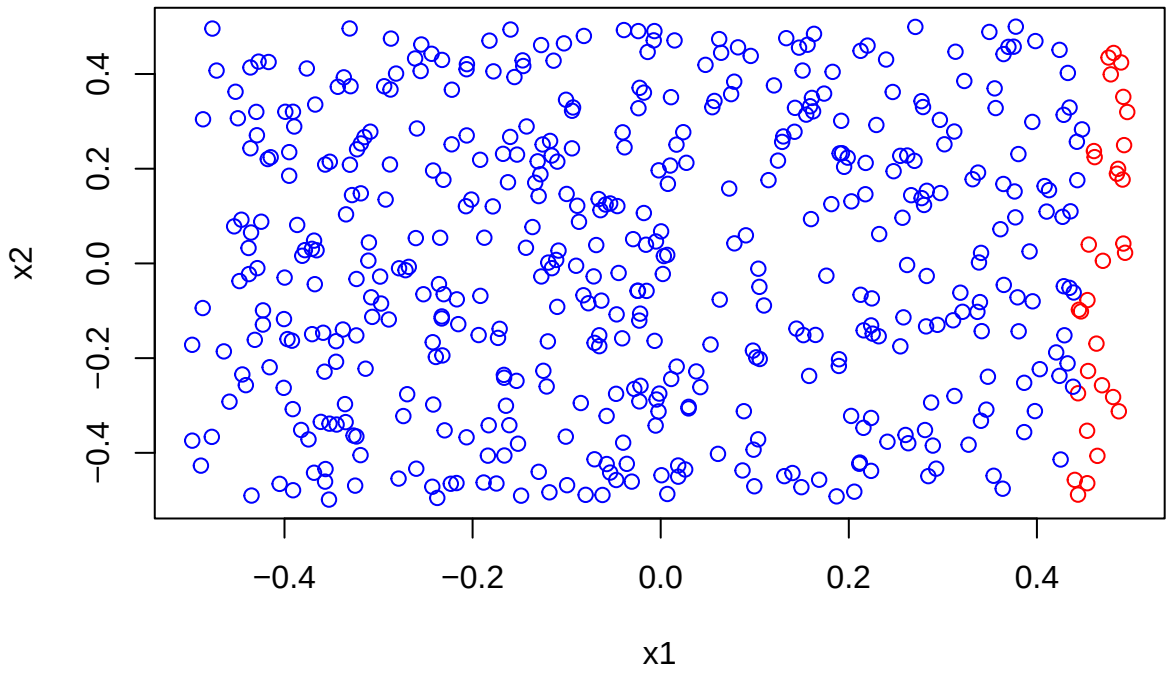
```
##
## Call:
## glm(formula = y ~ x1 + x2, family = binomial, data = data.frame(x1,
##     x2, y))
##
## Deviance Residuals:
##    Min      1Q  Median      3Q     Max
## -1.179  -1.139  -1.112   1.206   1.257
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) -0.087260   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
```

The coefficients are all not significant.

### (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.

```
data=data.frame(x1,x2,y)
pred.fit <- predict(fit.logis, data, type="response")
color.pred <- ifelse(pred.fit>0.5, "red", "blue")
plot(x1,x2,col=color.pred)
```




**Answer:**
The prediction is far from the true data. The decision boundary is linear.

**(e)**

Now fit a logistic regression model to the data using non-linear functions of X1 and X2 as predictors (e.g. X1^2, X1*X2, log(X2), and so forth).

```
fit.logis2 = glm(y ~ I(x1^2) + I(x2^2), data = data, family = binomial)
```

**Answer:**

```
## Warning: glm.fit: algorithm did not converge
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
summary(fit.logis2)
```

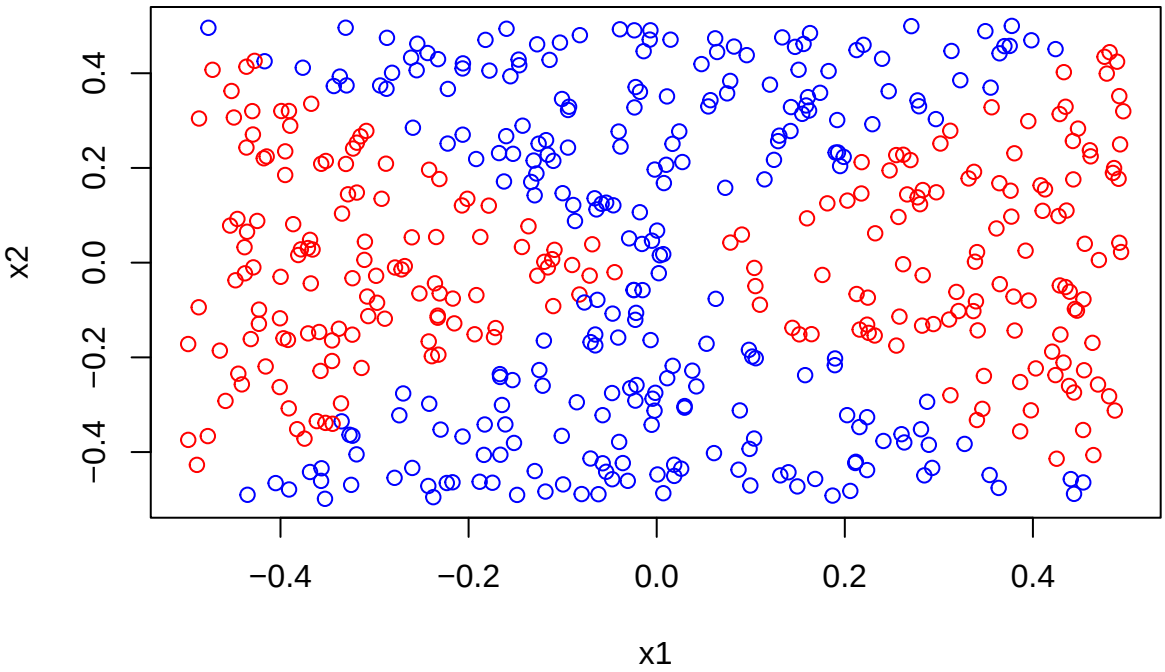
```
##
## Call:
## glm(formula = y ~ I(x1^2) + I(x2^2), family = binomial, data = data)
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -0.00667   0.00000   0.00000   0.00000   0.00565
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept)      7.285    173.960   0.042    0.967
## I(x1^2)      93153.509 860924.357   0.108    0.914
## I(x2^2)     -93265.460 863556.384  -0.108    0.914
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 6.9218e+02  on 499  degrees of freedom
```

```
## Residual deviance: 7.9869e-05  on 497  degrees of freedom
## AIC: 6.0001
##
## Number of Fisher Scoring iterations: 25
```

**(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.

```
pred.fit2 <- predict(fit.logis2, data, type="response")
color.pred2 <- ifelse(pred.fit2>0.5, "red", "blue")
plot(x1,x2,col=color.pred2)
```

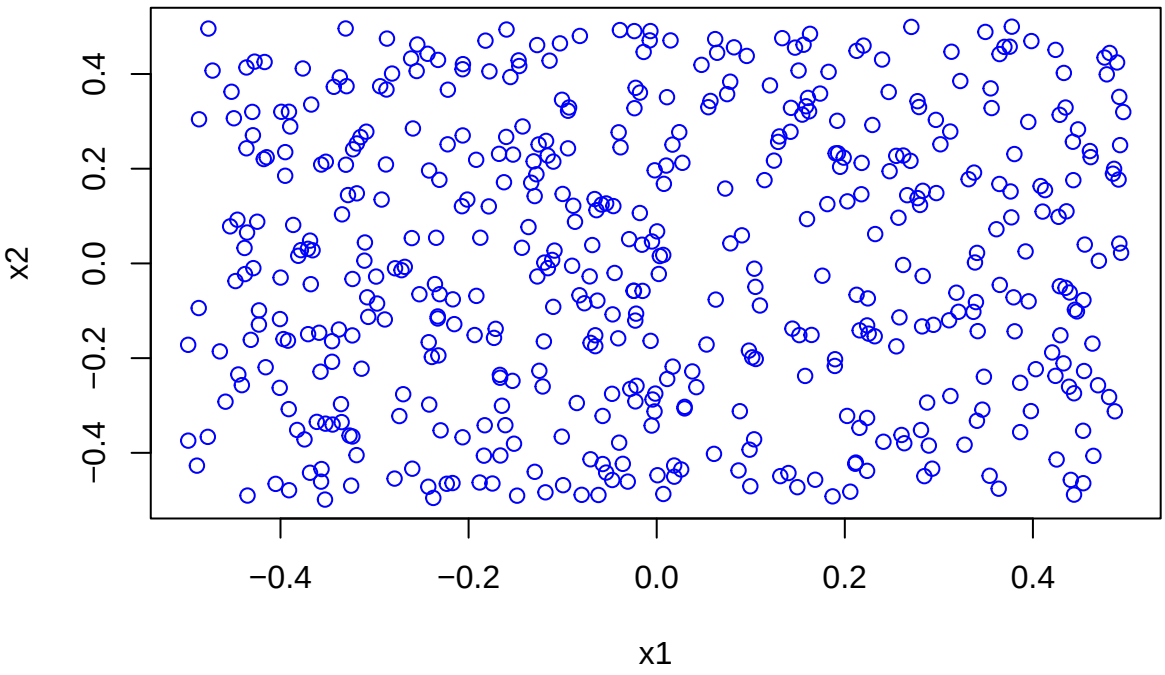



**Answer:**

**(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.

```
fit.svm = svm(as.factor(y) ~ x1 + x2, data, kernel = "linear", cost = 0.1)
pred.svm = predict(fit.svm, data)
color.svm <- ifelse(pred.svm==1, "red", "blue")
plot(x1,x2,col=color.svm)
```

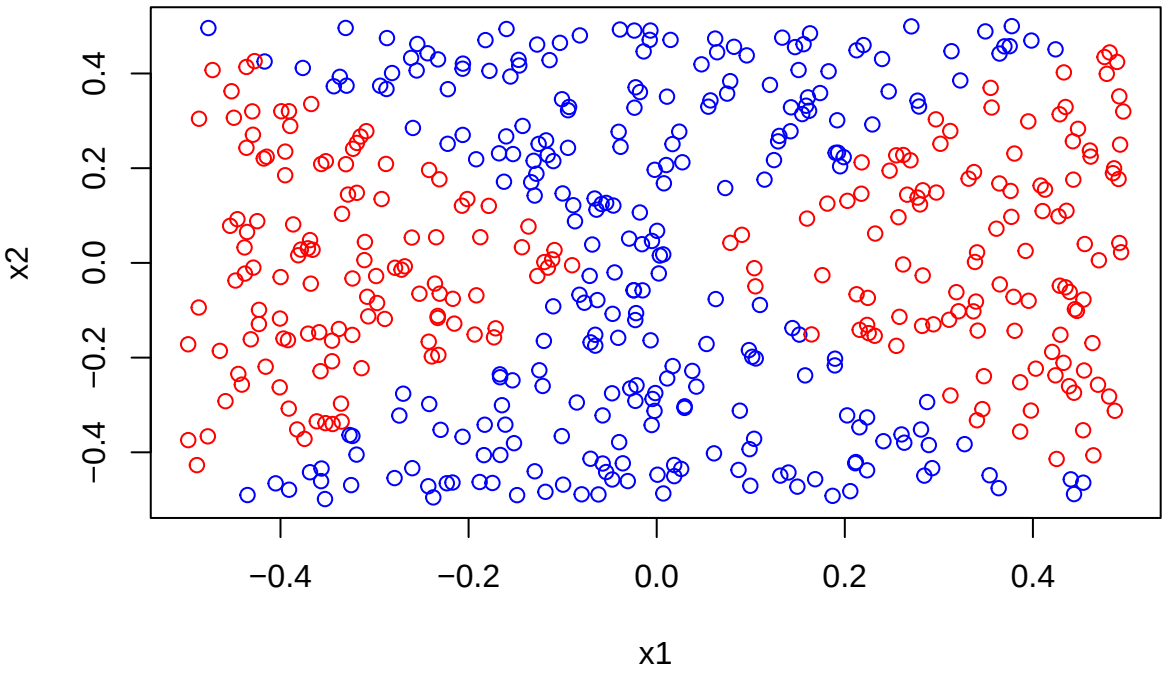



**Answer:**

**(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.

```
fit.svm2 = svm(as.factor(y) ~ x1 + x2, data, kernel = "polynomial", degree = 2)
pred.svm2 = predict(fit.svm2, data)
color.svm2 <- ifelse(pred.svm2==1, "red", "blue")
plot(x1,x2,col=color.svm2)
```




**Answer:**

### (i)

Comment on your results.

**Answer:** It is very convenient to use SVM to find non-linear decision boundary without setting much parameters and predictors.

## 9.7

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.

```
new.var = ifelse(Auto$mpg > median(Auto$mpg), 1, 0)
Auto$mpglevel = as.factor(new.var)
```

**Answer:**

### (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.

```
set.seed(1)
tune1 = tune(svm, mpglevel ~ ., data = Auto, kernel = "linear", ranges = list(cost = c(0.01, 0.1, 1, 5,
summary(tune1)
```

**Answer:**

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##     1
##
## - best performance: 0.01025641
##
## - Detailed performance results:
##    cost      error dispersion
## 1 1e-02 0.07653846 0.03617137
## 2 1e-01 0.04596154 0.03378238
## 3 1e+00 0.01025641 0.01792836
## 4 5e+00 0.02051282 0.02648194
## 5 1e+01 0.02051282 0.02648194
## 6 1e+02 0.03076923 0.03151981
```

The best cost is 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.

```
tune2 = tune(svm, mpglevel ~ ., data = Auto, kernel = "polynomial", ranges = list(cost = c(0.1,
    1, 5, 10, 100), degree = c(2, 3, 4)))
summary(tune2)
```

**Answer:**

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost degree
##   100      2
##
## - best performance: 0.3060897
##
## - Detailed performance results:
##     cost degree     error dispersion
## 1    0.1      2 0.6019231 0.06346118
## 2    1.0      2 0.6019231 0.06346118
## 3    5.0      2 0.6019231 0.06346118
## 4   10.0      2 0.5841667 0.07806609
## 5  100.0      2 0.3060897 0.07318010
## 6    0.1      3 0.6019231 0.06346118
## 7    1.0      3 0.6019231 0.06346118
## 8    5.0      3 0.6019231 0.06346118
## 9   10.0      3 0.6019231 0.06346118
## 10 100.0      3 0.3413462 0.14973473
## 11   0.1      4 0.6019231 0.06346118
## 12   1.0      4 0.6019231 0.06346118
## 13   5.0      4 0.6019231 0.06346118
## 14  10.0      4 0.6019231 0.06346118
## 15 100.0      4 0.6019231 0.06346118
```

```
tune3 = tune(svm, mpglevel ~ ., data = Auto, kernel = "radial", ranges = list(cost = c(0.1,
    1, 5, 10, 100), gamma = c(0.01, 0.1, 1, 5, 10, 100)))
summary(tune3)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost gamma
##    10  0.01
##
```

```
## - best performance: 0.02044872
##
## - Detailed performance results:
##     cost gamma      error dispersion
## 1    0.1 1e-02 0.08698718 0.04560056
## 2    1.0 1e-02 0.07160256 0.03373099
## 3    5.0 1e-02 0.05121795 0.02967002
## 4   10.0 1e-02 0.02044872 0.01077927
## 5  100.0 1e-02 0.02044872 0.02020886
## 6    0.1 1e-01 0.07673077 0.03419344
## 7    1.0 1e-01 0.05121795 0.03203768
## 8    5.0 1e-01 0.02557692 0.01709522
## 9   10.0 1e-01 0.02301282 0.02244393
## 10 100.0 1e-01 0.03070513 0.03153205
## 11   0.1 1e+00 0.59461538 0.08083319
## 12   1.0 1e+00 0.06141026 0.03026776
## 13   5.0 1e+00 0.06397436 0.02789391
## 14  10.0 1e+00 0.06397436 0.02789391
## 15 100.0 1e+00 0.06397436 0.02789391
## 16   0.1 5e+00 0.59461538 0.08083319
## 17   1.0 5e+00 0.52051282 0.09421163
## 18   5.0 5e+00 0.51538462 0.10051415
## 19  10.0 5e+00 0.51538462 0.10051415
## 20 100.0 5e+00 0.51538462 0.10051415
## 21   0.1 1e+01 0.59461538 0.08083319
## 22   1.0 1e+01 0.55384615 0.09432787
## 23   5.0 1e+01 0.54358974 0.09085645
## 24  10.0 1e+01 0.54358974 0.09085645
## 25 100.0 1e+01 0.54358974 0.09085645
## 26   0.1 1e+02 0.59461538 0.08083319
## 27   1.0 1e+02 0.59461538 0.08083319
## 28   5.0 1e+02 0.59461538 0.08083319
## 29  10.0 1e+02 0.59461538 0.08083319
## 30 100.0 1e+02 0.59461538 0.08083319
```

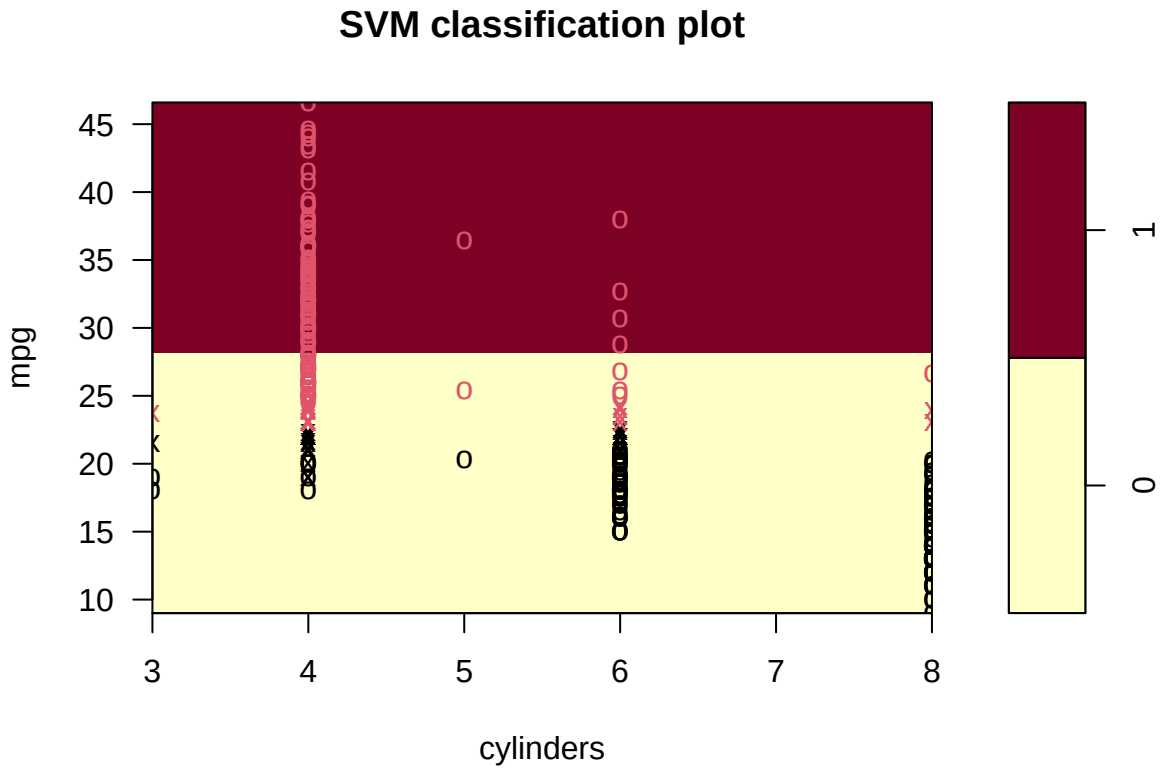
For polynomial, the best cost is 100, best degree is 2.

For radial, the best cost is 10, best gamma is 0.01.

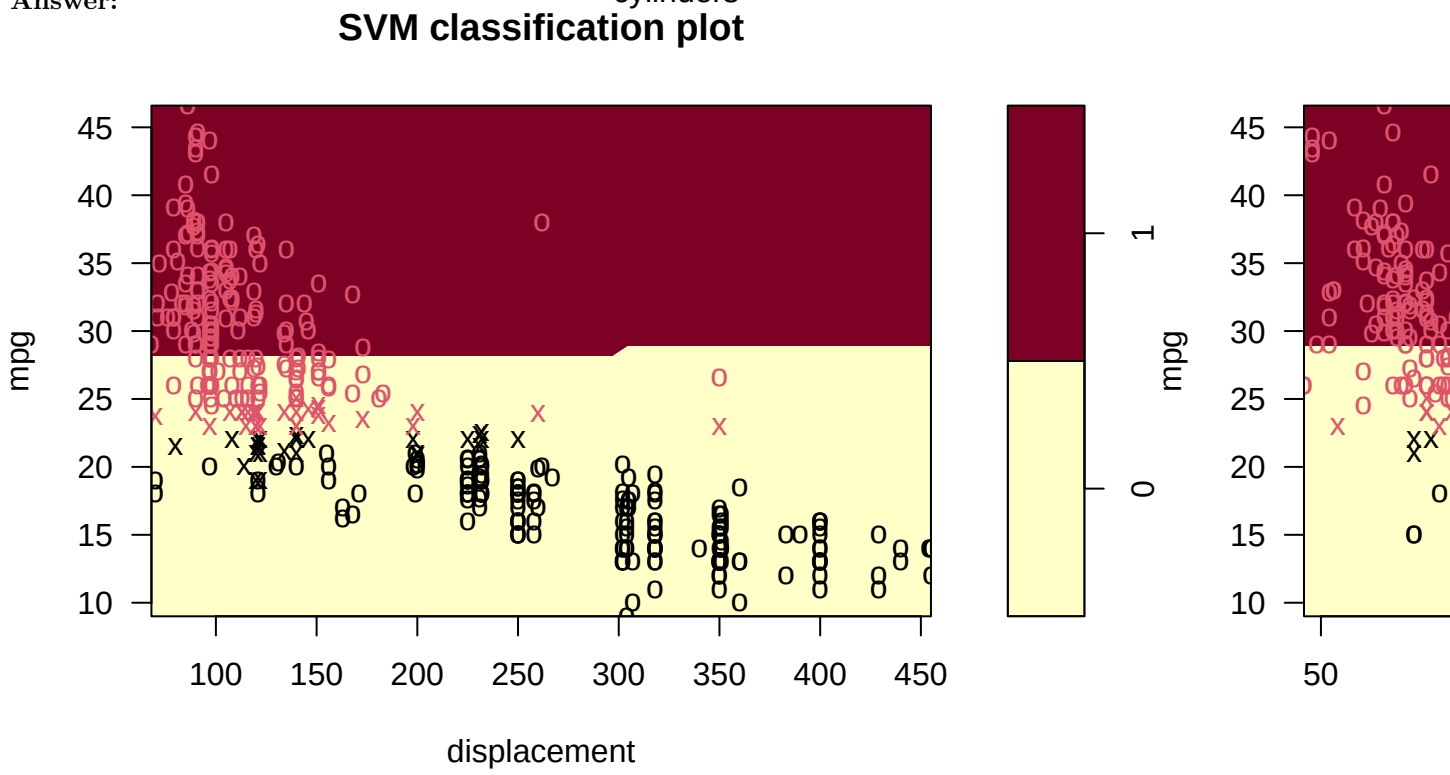
### (d)

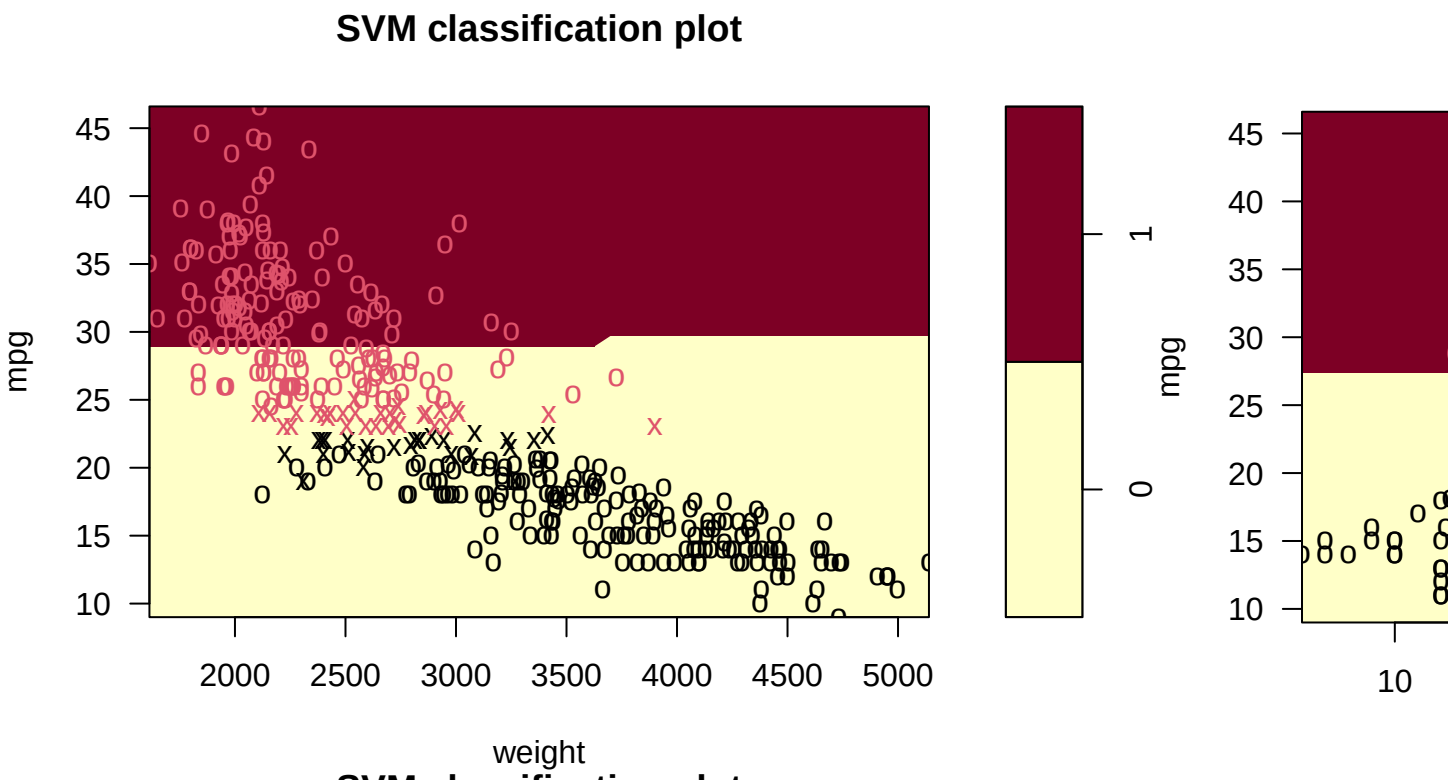
Make some plots to back up your assertions in (b) and (c).

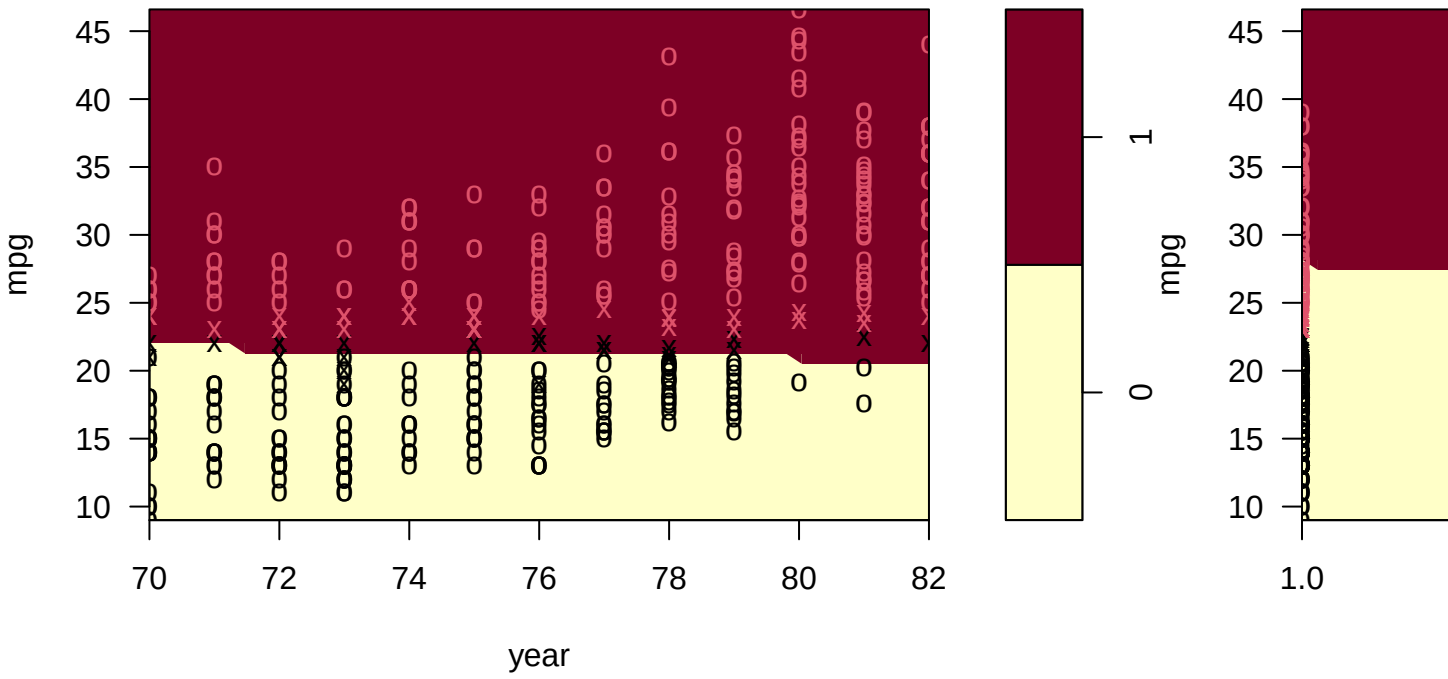
```
svm.linear = svm(mpglevel ~ ., data = Auto, kernel = "linear", cost = 1)
svm.poly = svm(mpglevel ~ ., data = Auto, kernel = "polynomial", cost = 100, degree = 2)
svm.radial = svm(mpglevel ~ ., data = Auto, kernel = "radial", cost = 10, gamma = 0.01)
plotsvm = function(fit) {
    for (name in names(Auto)[!(names(Auto) %in% c("mpg", "mpglevel", "name"))]) {
        plot(fit, Auto, as.formula(paste("mpg~", name, sep = "")))
    }
}
plotsvm(svm.linear)
```


SVM classification plot
mpg
cylinders
45
40
35
30
25
20
15
10
3
4
5
6
7
8
1
0

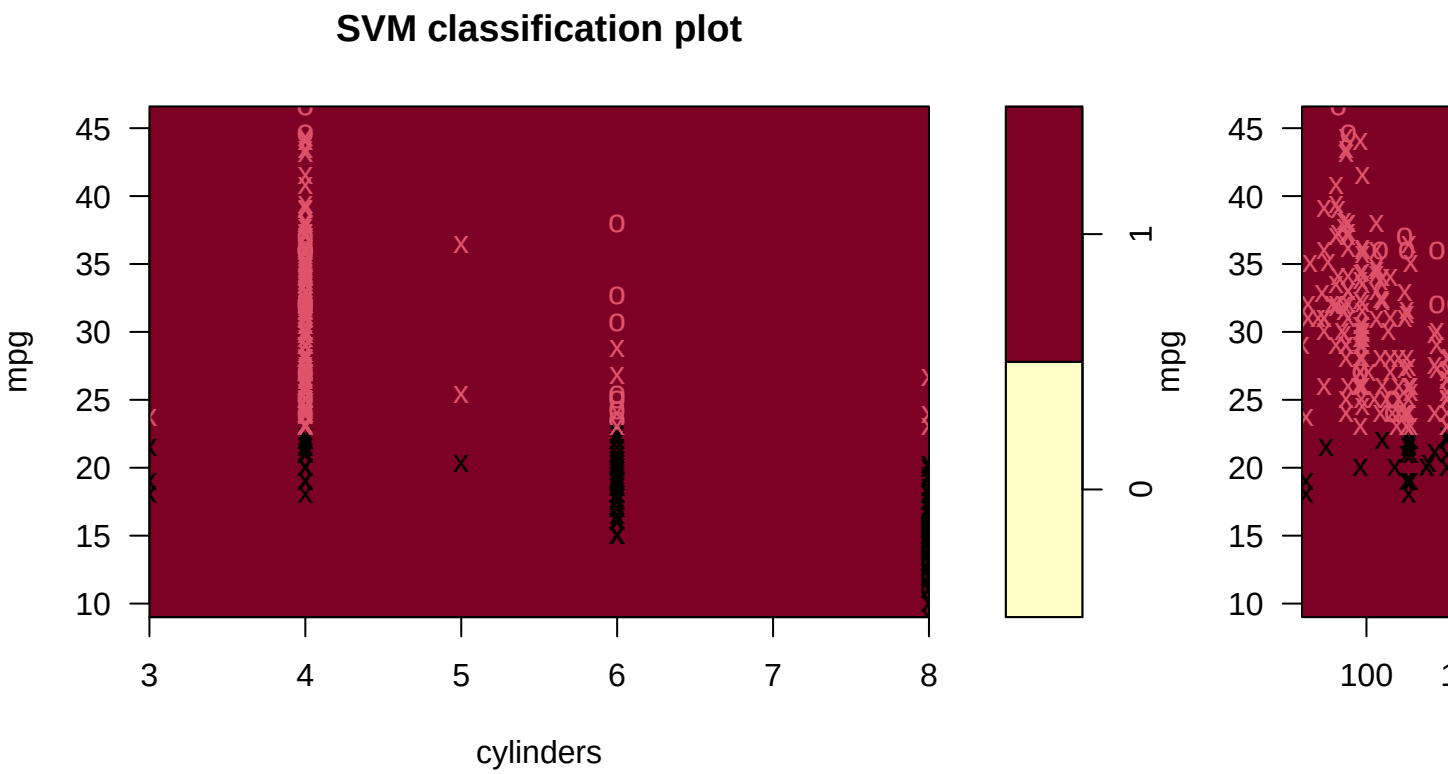
**Answer:**

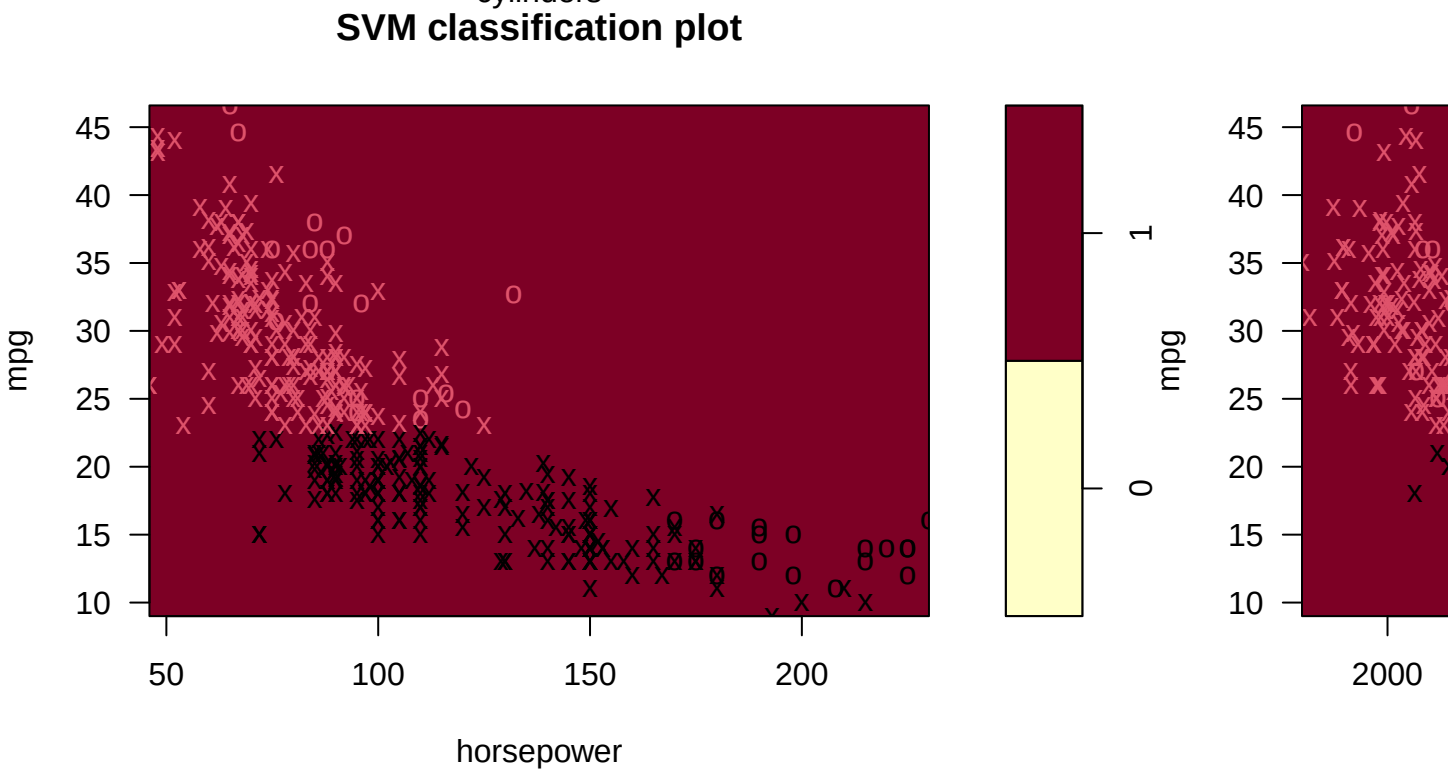

SVM classification plot
mpg
displacement
45
40
35
30
25
20
15
10
100
150
200
250
300
350
400
450
1
0
mpg
50

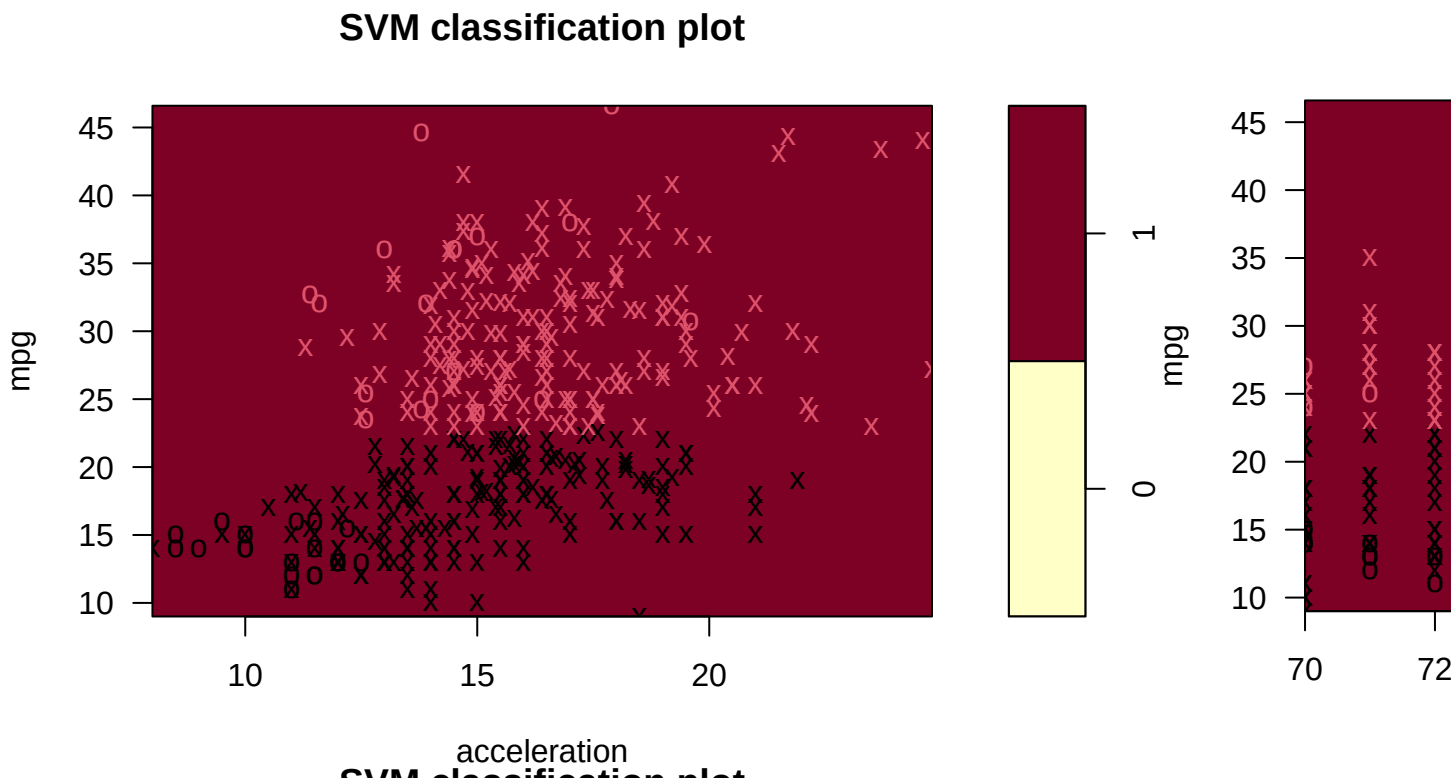


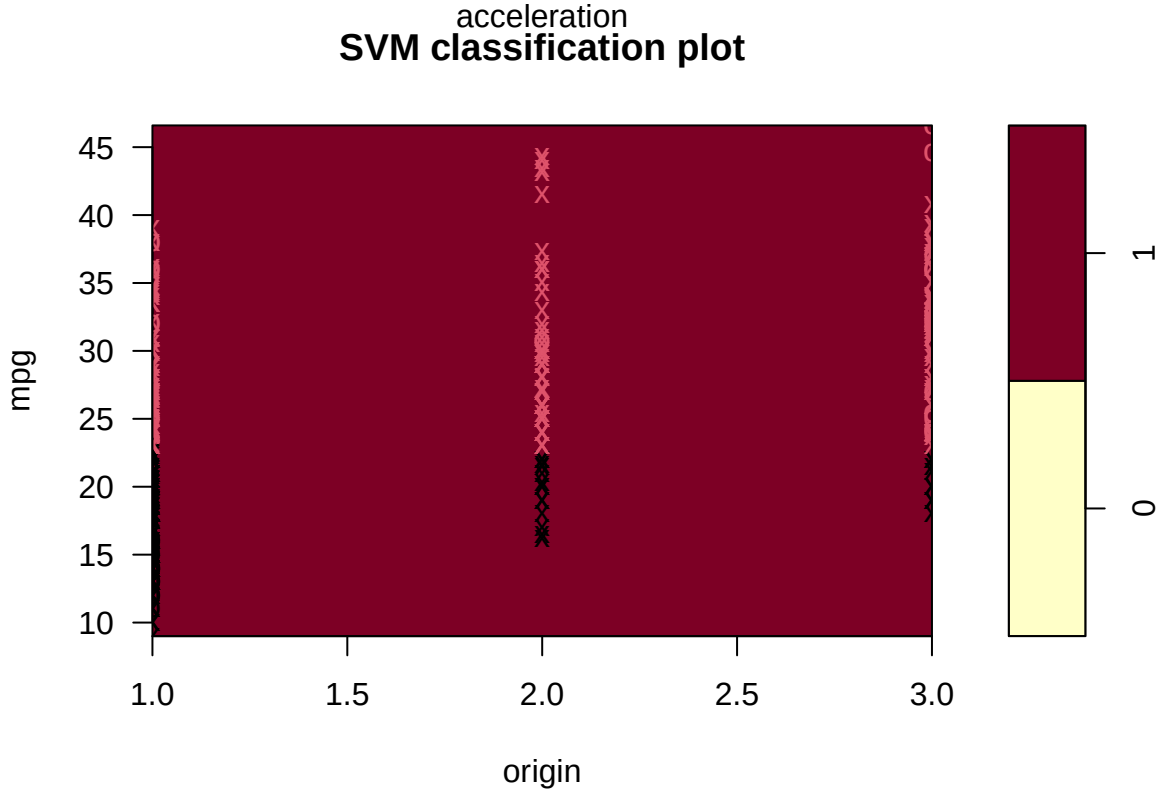




```
plotsvm(svm.poly)
```

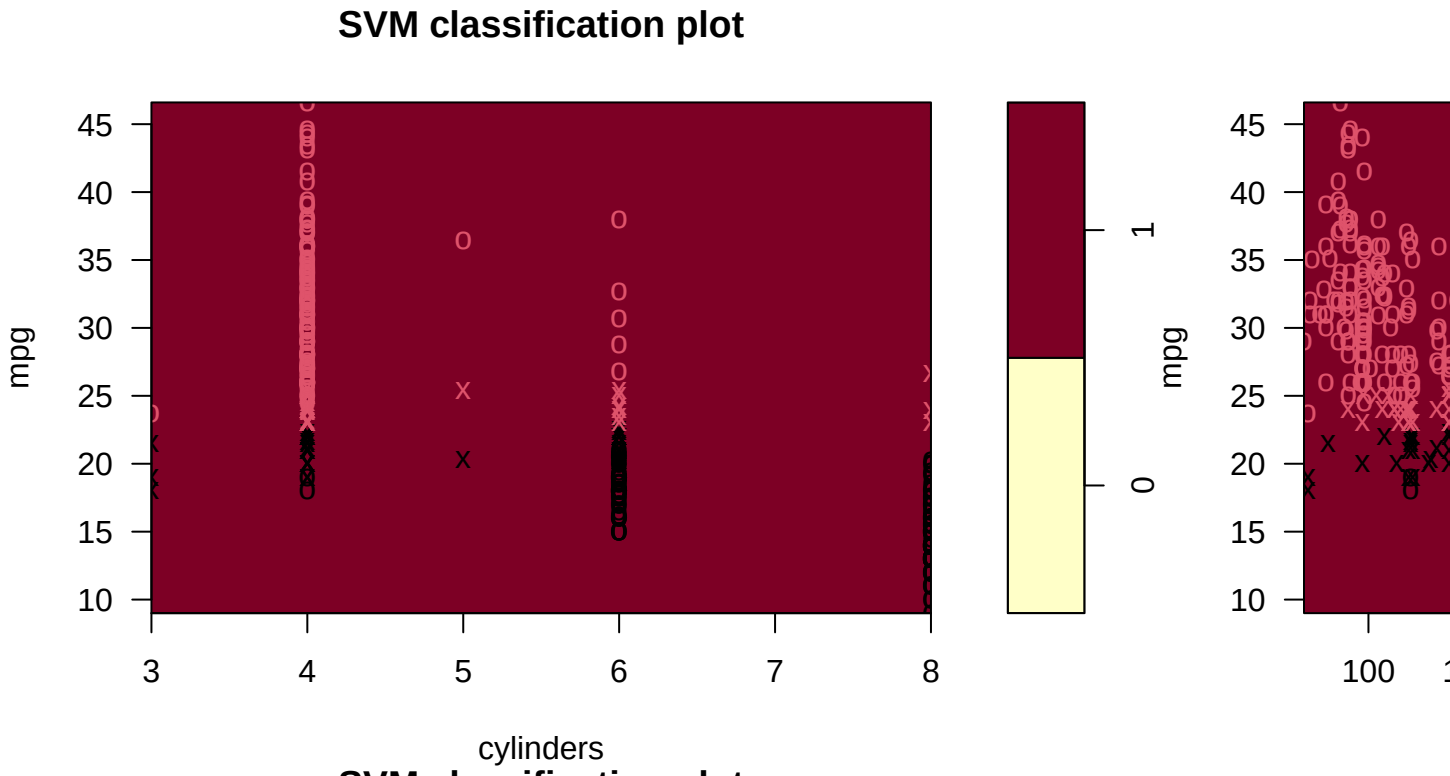

SVM classification plot
mpg
cylinders
mpg

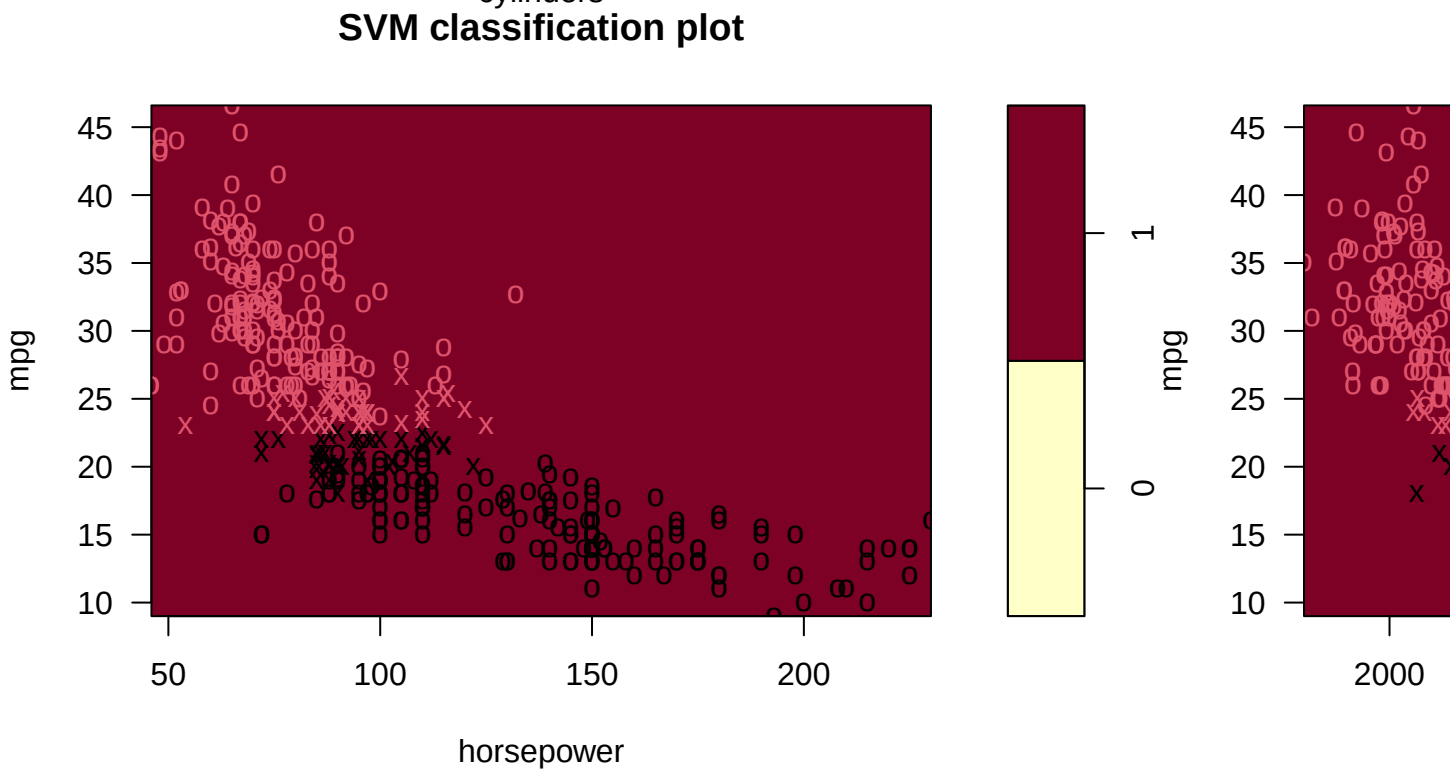

SVM classification plot
mpg
horsepower
mpg

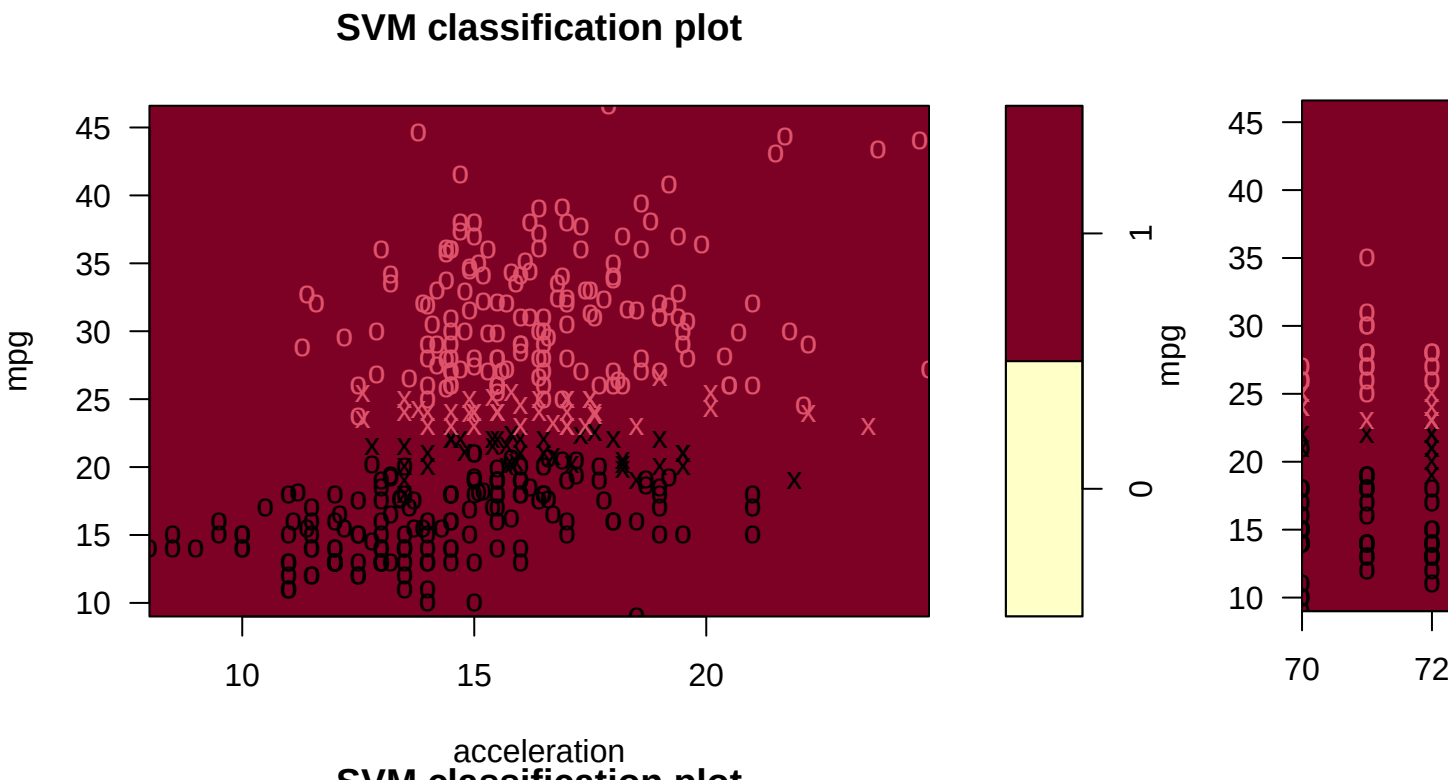


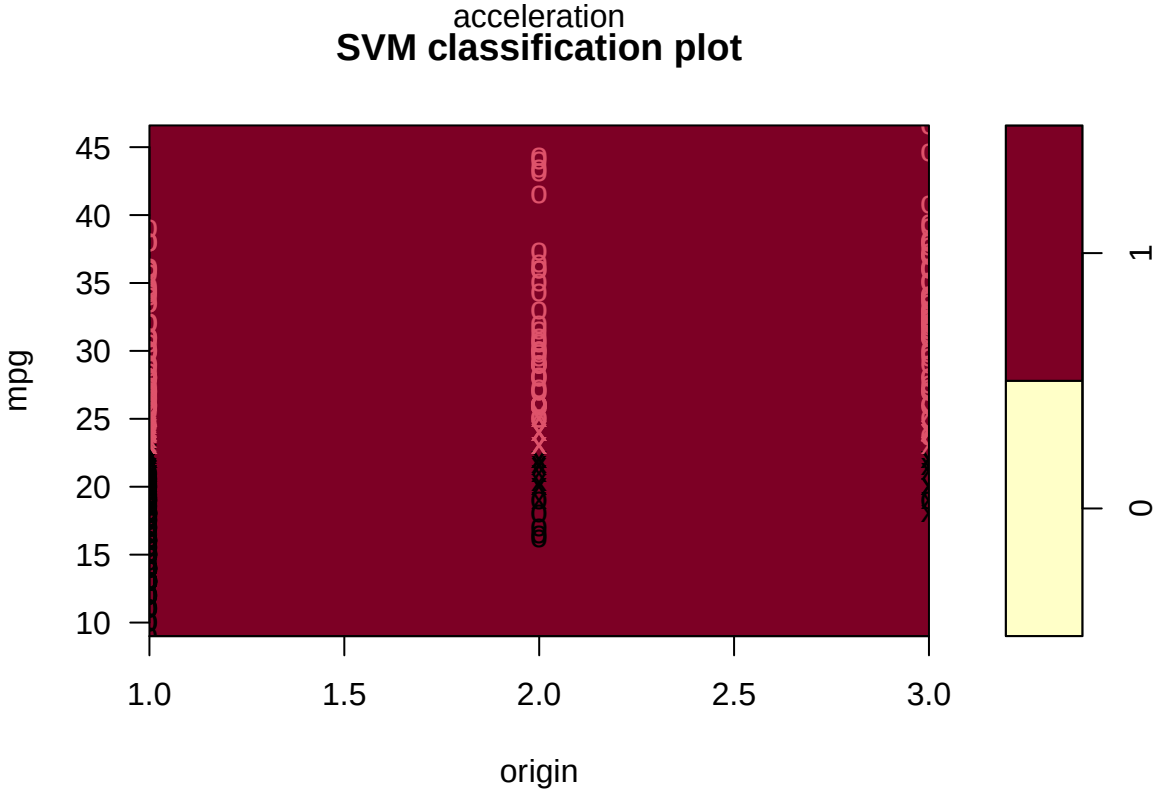




```
plotsvm(svm.radial)
```


SVM classification plot
mpg
cylinders
mpg


SVM classification plot
mpg
horsepower
mpg




**9.8**

**(a)**

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

```
set.seed(1)
train <- sample(dim(OJ)[1], 800)
OJ.train <- OJ[train, ]
OJ.test <- OJ[-train, ]
```

**Answer:**

### (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.

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

**Answer:**

```
##
## Call:
## svm(formula = Purchase ~ ., data = OJ.train, kernel = "linear", cost = 0.01)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  0.01
##
## Number of Support Vectors:  435
##
##  ( 219 216 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

### (c)

What are the training and test error rates?

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

**Answer:**

```
##     train.pred
##       CH  MM
##   CH 420  65
##   MM  75 240
```

```
(train.errorRate = (75+65)/(75+65+420+240))
```

```
## [1] 0.175
```

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

```
##     test.pred
##       CH  MM
##   CH 153  15
##   MM  33  69
```

```
(test.errorRate = (15+33)/(15+33+153+69))
```

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

### (d)

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

```
set.seed(2)
tune.linear = tune(svm, Purchase ~ ., data = OJ.train, kernel = "linear", ranges = list(cost = seq(0.01
tune.linear$best.parameters$cost
```

**Answer:**

```
## [1] 4.01
```

### (e)

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

```
svm.linear2 = svm(Purchase ~ ., kernel = "linear", data = OJ.train, cost = tune.linear$best.parameters$

train.pred2 = predict(svm.linear2, OJ.train)
table(OJ.train$Purchase, train.pred2)
```

**Answer:**

```
##     train.pred2
##       CH  MM
##   CH 423  62
##   MM  70 245
```

```
(train.errorRate2 = (61+70)/(61+70+424+245))
```

```
## [1] 0.16375
```

```
test.pred2 = predict(svm.linear2, OJ.test)
table(OJ.test$Purchase, test.pred2)
```

```
##     test.pred2
##       CH  MM
##   CH 155  13
##   MM  29  73
```

```
(test.errorRate2 = (13+29)/(13+29+155+73))
```

```
## [1] 0.1555556
```

### (f)

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

```
set.seed(3)
tune.radial = tune(svm, Purchase ~ ., data = OJ.train, kernel = "radial", ranges = list(cost = seq(0.01
svm.radial = svm(Purchase ~ ., data = OJ.train, kernel = "radial", cost = tune.radial$best.parameters$c
train.pred.radial = predict(svm.radial, OJ.train)
table(OJ.train$Purchase, train.pred.radial)
```

**Answer:**

```
##     train.pred.radial
##       CH  MM
##   CH 438  47
##   MM  72 243
```

```
(47+72)/dim(OJ.train)[1]
```

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

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

```
##     test.pred.radial
##       CH  MM
##   CH 150  18
##   MM  30  72
```

```
(30+18)/dim(OJ.test)[1]
```

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

### (g)

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

```
set.seed(4)
tune.poly = tune(svm, Purchase ~ ., data = OJ.train, kernel = "poly", ranges = list(cost = seq(0.01, 10
svm.poly = svm(Purchase ~ ., data = OJ.train, kernel = "poly", cost = tune.poly$best.parameters$cost)
train.pred.poly = predict(svm.poly, OJ.train)
table(OJ.train$Purchase, train.pred.poly)
```

**Answer:**

```
##     train.pred.poly
##       CH  MM
##   CH 454  31
##   MM  97 218
```

```
(31+97)/dim(OJ.train)[1]
```

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

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

```
##     test.pred.poly
##       CH  MM
##   CH 154  14
##   MM  47  55
```

```
(47+14)/dim(OJ.test)[1]
```

```
## [1] 0.2259259
```

**(h)**

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

**Answer:** Linear kernel has lowest test error rate.