Assignment 4

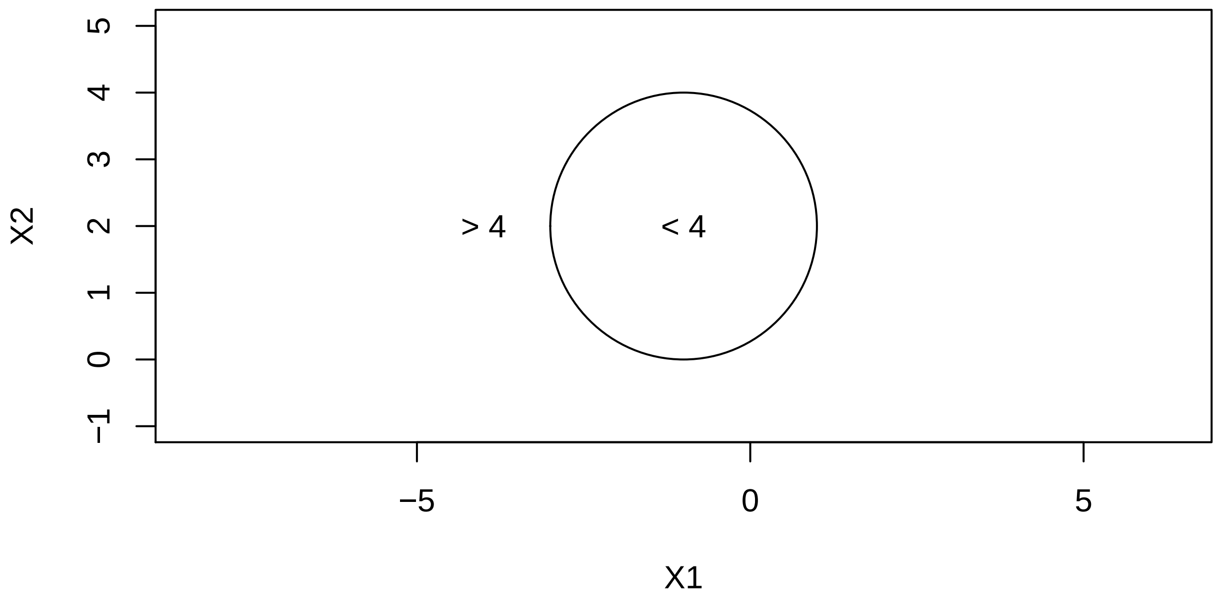
Exercise 1

We have seen that in $p = 2$ dimensions, a linear decision boundary takes the form $\beta_0 + \beta_1 X_1 + \beta_2 X_2 = 0$. We now investigate a non-linear decision boundary.

(a-c) Sketch the curve $(1 + X_1)^2 + (2 - X_2)^2 = 4$. On your sketch, indicate the set of points for which $(1 + X_1)^2 + (2 - X_2)^2 > 4$, as well as the set of points for which $(1 + X_1)^2 + (2 - X_2)^2 \leq 4$. Suppose that a classifier assigns an observation to the blue class if $(1 + X_1)^2 + (2 - X_2)^2 > 4$, and to the red class otherwise. To what class is the observation $(0, 0)$ classified? $(-1, 1)$? $(2, 2)$? $(3, 8)$?

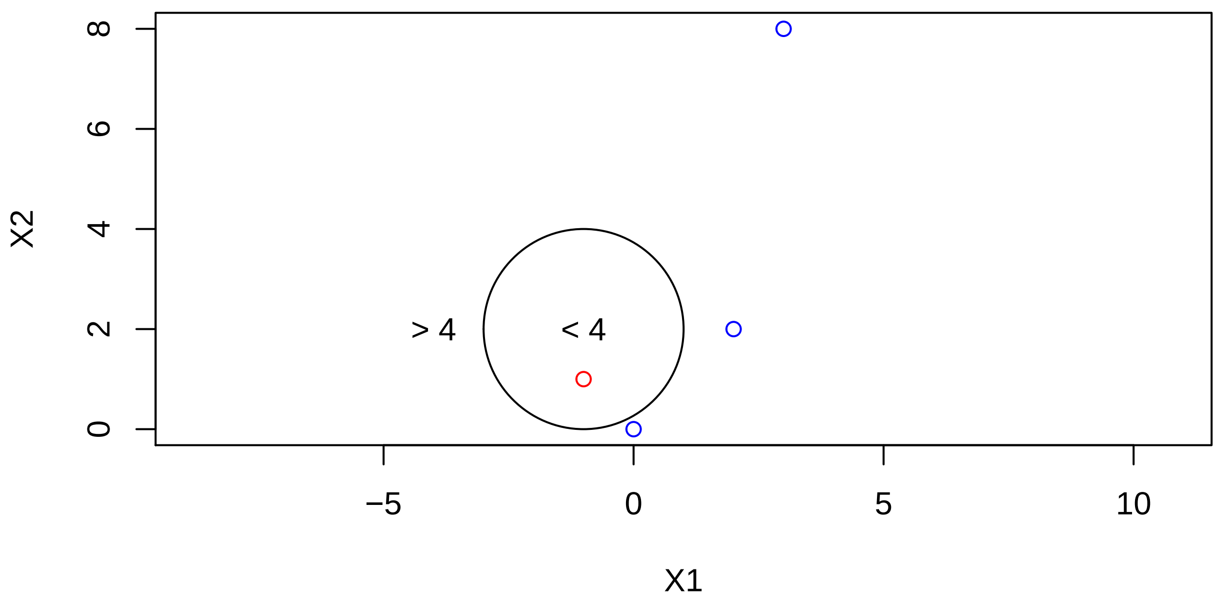
The given equation is the definition of a circle with radius of 2 and is exactly 4 on the circumference. The corresponding dots have been colored according to in which class the observation belongs.

```
# Radius of the circle
rad = 2

# Plotting the circle and restrictive areas
plot(NA, NA, type = "n", xlim = c(-4, 2), ylim = c(-1, 5), asp = 1, xlab = "X1",
    ylab = "X2")
symbols(c(-1), c(2), circles = c(rad), add = TRUE, inches = FALSE)

# Adding text on the area according to given restrictive equations
text(c(-1), c(2), "< 4")
text(c(-4), c(2), "> 4")
```



```
# Plotting the dots (note that for fixing the colors you should already know
# if the dots belong inside or outside of the given circle)
plot(c(0, -1, 2, 3), c(0, 1, 2, 8), col = c("blue", "red", "blue", "blue"),
    type = "p", asp = 1, xlab = "X1", ylab = "X2")
```

```r
# Adding the circle and restrictive areas
symbols(c(-1), c(2), circles = c(rad), add = TRUE, inches = FALSE)
text(c(-1), c(2), "< 4")
text(c(-4), c(2), "> 4")
```



(d) Argue that while the decision boundary in (c) is not linear in terms of X_1 and X_2, it is linear in terms of X_1, X_1^2 , X_2, and X_2^2.

If the decision boundary is shown in it's expanded form we get

$$(1 + X_1)^2 + (2 - X_2)^2 > 4 \iff 1 + 2X_1 + X_1^2 + 4 - 4X_2 + X_2^2 > 4 \iff 5 + 2X_1 - 4X_2 + X_1^2 + X_2^2 > 4.$$

The last inequality shows that the terms X_1 and X_2 are not in deed linear due to fact that they exist both in linear and in quadratic form. But if the terms are treated separately i.e. they "represent themselves", then they're linear – multiplying with a constant retrains the linear property of the term.

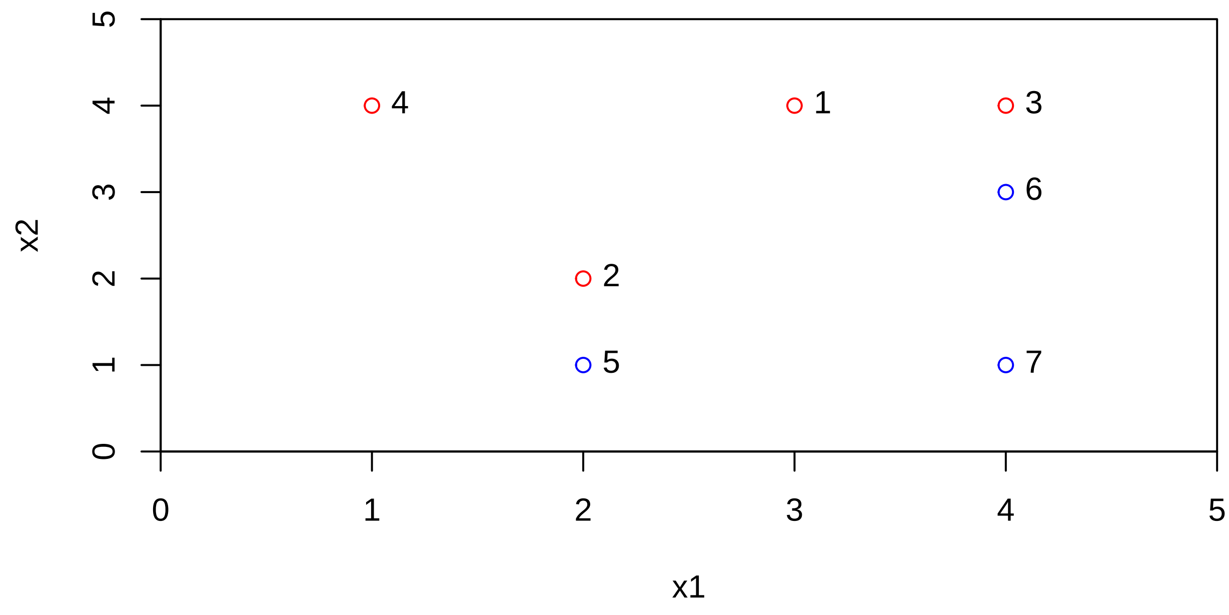
Exercise 2

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.

```r
# Fixing origin of the plot
par(xaxs = "i", yaxs = "i")

# Creating the data vectors
x1 = c(3, 2, 4, 1, 2, 4, 4)
x2 = c(4, 2, 4, 4, 1, 3, 1)

# Storing the given colors (2 = red, 4 = blue)
colors = rep(c(2, 4), c(4, 3))
```

```r
# Scatter plotting the data with labels
plot(x2 ~ x1, col = colors, xlim = c(0, 5), ylim = c(0, 5))
text(x2 ~ x1, labels = c(1:7), pos = 4)
```



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

The maximal margin classifier has to be in between observations 2, 5 and 3, 6. The coordinates (x_1, x_2)in corresponding order are $(2, 2)$, $(4, 4)$, $(2, 1)$ and $(4, 3)$. The middle point between observations 2 and 5 is $(2, 1.5)$ and the middle point for 3 and 6 is $(4, 3.5)$. The hyperplane goes through these points and the distance from the hyperplane to observations is equal i.e. it goes right between the observations. Because the original dimension $p = 2$, hyperplane's dimension is $p - 1 = 1$ which is a line.

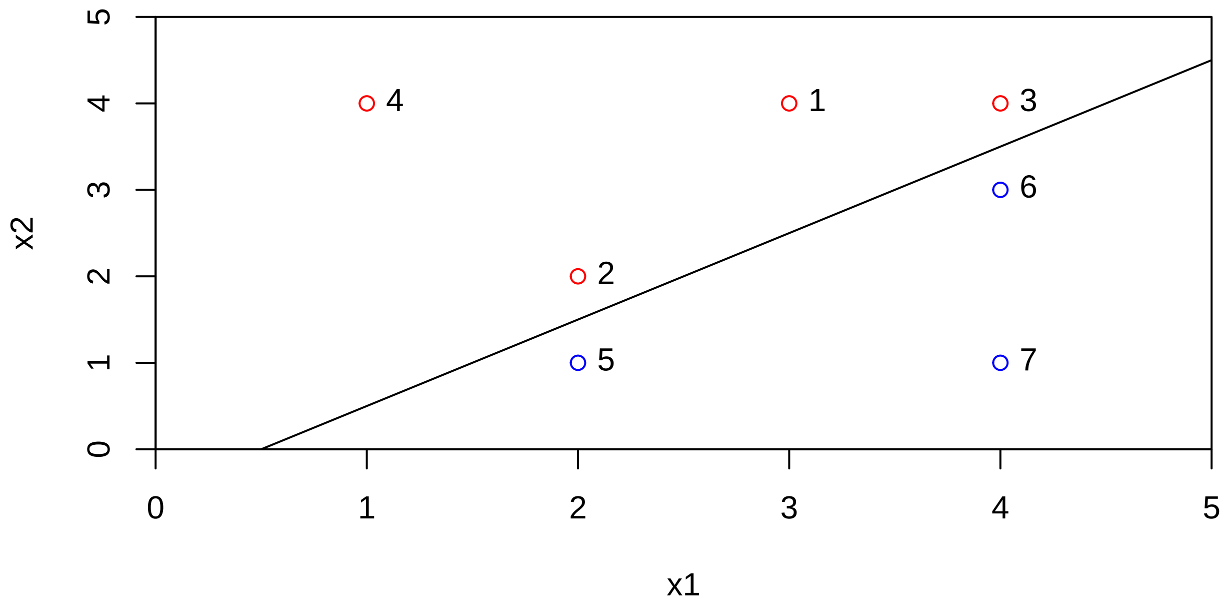
$$\beta_1 = (2 - 4)/(1.5 - 3.5) = 1$$

and

$$\begin{aligned} x_2 - 3.5 &= \beta_1(x_1 - 4) \\ x_2 &= (x_1 - 4) + 3 \\ x_2 &= x_1 - 0.5. \end{aligned}$$

The hyperplane is $\beta_0 + \beta_1 x_1 + \beta_2 x_2 = 0$, where $\beta_0 = 0.5$, $\beta_1 = -1$ and $\beta_2 = 1$.

```r
# Fixing origin of the plot
par(xaxs = "i", yaxs = "i")

# Scatter plotting the data with labels
plot(x2 ~ x1, col = colors, xlim = c(0, 5), ylim = c(0, 5))
text(x2 ~ x1, labels = c(1:7), pos = 4)

# Adding the hyperplane to the plot
abline(-0.5, 1)
```

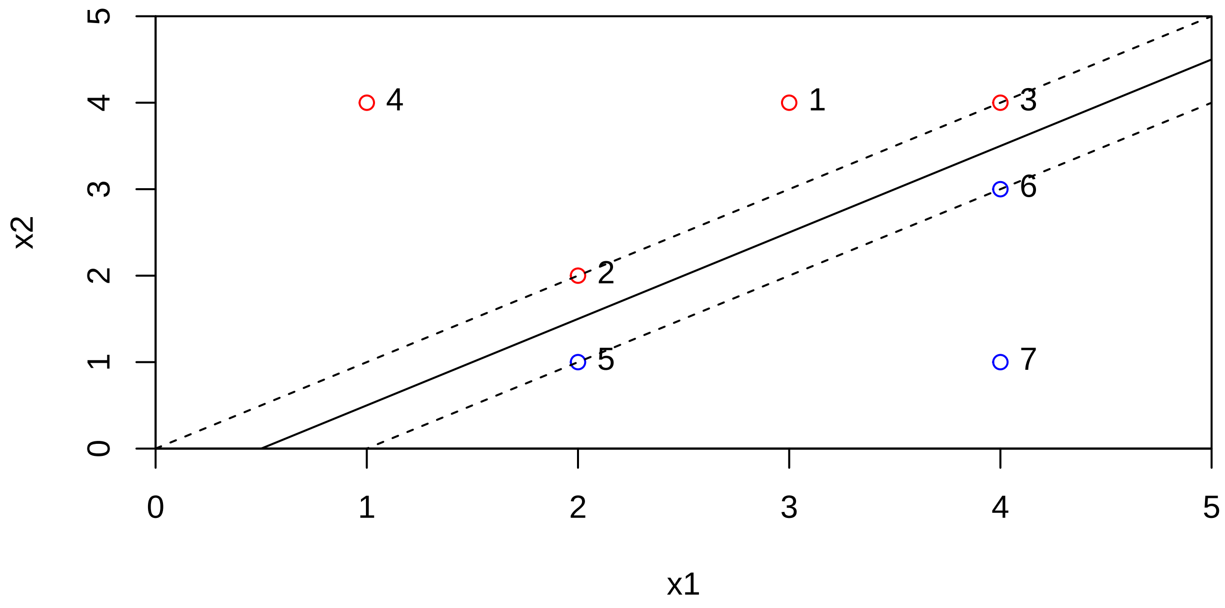


(c) Describe the classification rule for the maximal margin classifier. It should be something along the lines of "Classify to Red if $\beta_0 + \beta_1 X_1 + \beta_2 2X_2 > 0$, and classify to Blue otherwise." Provide the values for β_0, β_1 and β_2.

If $\beta_0 + \beta_1 x_1 + \beta_2 x_2 > 0$, where β_0, β_1 and β_2 are the same as above, then classify to red, otherwise classify to blue.

(d-3) On your sketch, indicate the margin for the maximal margin hyperplane. Indicate the support vectors for the maximal margin classifier.

The maximal margin goes through observations 2,3 and 5,6 and the corresponding coordinates are $(2, 2)$, $(4, 4)$ and $(2, 1)$, $(4, 3)$, which are also the support vectors for the maximal margin classifier.

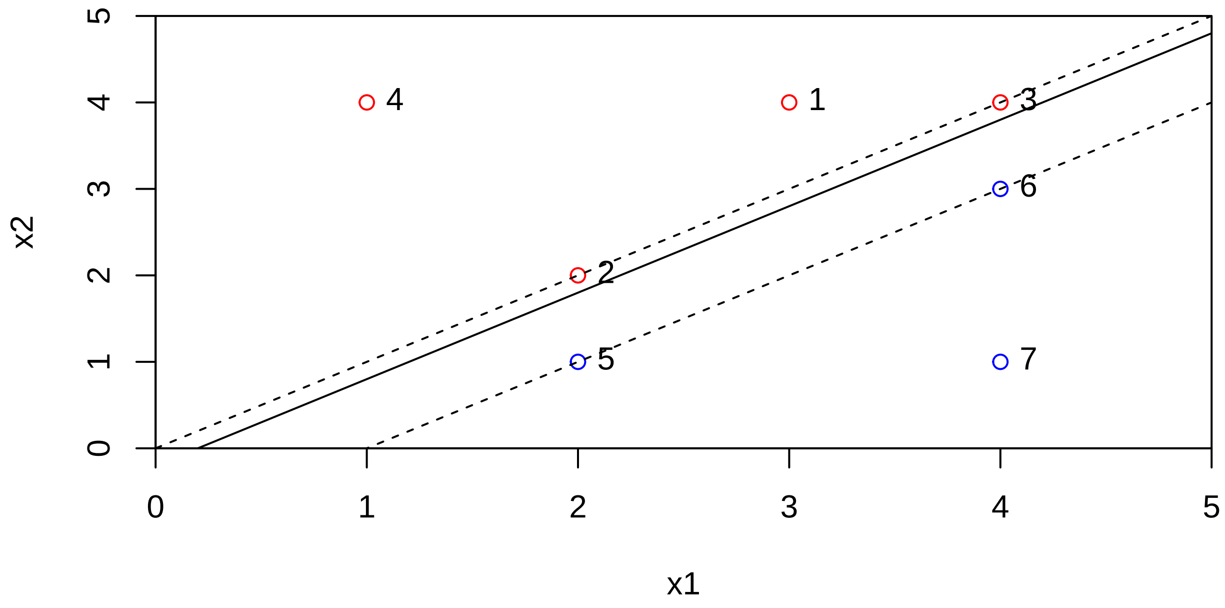


(f) Argue that a slight movement of the seventh observation would not affect the maximal margin hyperplane.

The seventh observation is so far away from the margins and the hyperplane that moving it anywhere below the lower margin doesen't affect the margin.

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

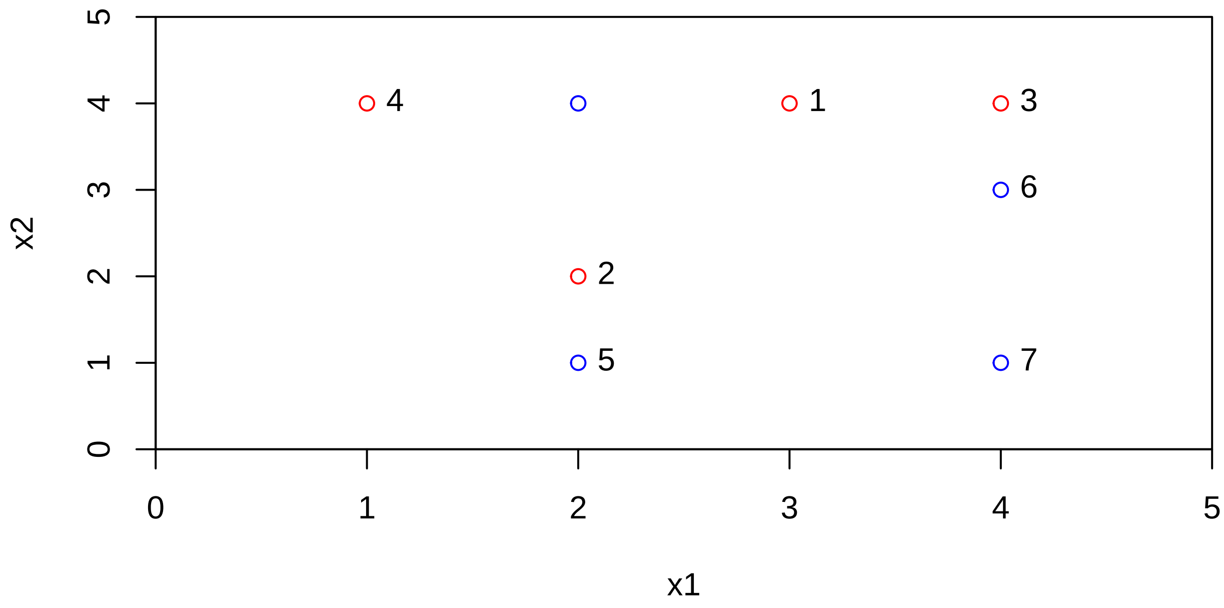
A hyperplane in which the constant is a bit less or a bit more than 0.5 isn't optimal because it cannot maximize both of the margins anymore. For example $\beta_0 + \beta_1 x_1 + \beta_2 x_2 = 0$, where $\beta_0 = 0.2$, $\beta_1 = -1$ and $\beta_2 = 1$



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

For example if the additional observation is in between two opposite colors, then the two classes cannot be separated by a hyperplane

```r
# Fixing origin of the plot
par(xaxs = "i", yaxs = "i")

# Scatter plotting the data with labels
plot(x2 ~ x1, col = colors, xlim = c(0, 5), ylim = c(0, 5))
text(x2 ~ x1, labels = c(1:7), pos = 4)

# Adding the dot
points(2, 4, col = 4)
```



Exercise 3

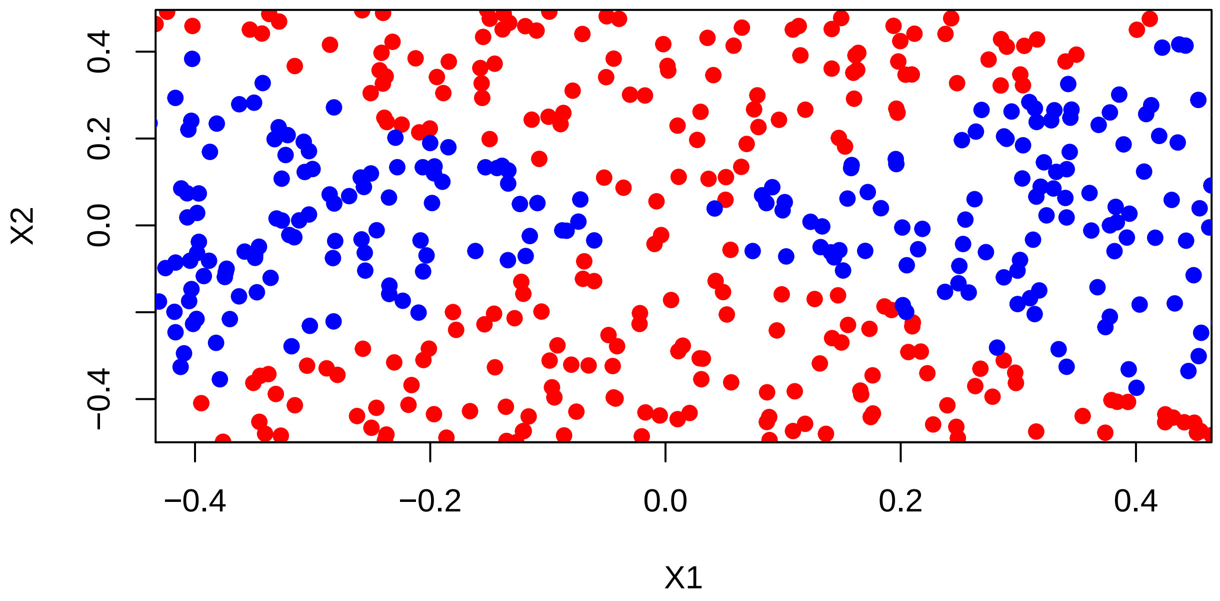
We have seen that we can fit an SVM with a non-linear kernel in order to perform classification using a non-linear decision boundary. We will now see that we can also obtain a non-linear decision boundary by performing logistic regression using non-linear transformations of the features.

(a) 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:

```
set.seed(12)
x1 = runif(500) - 0.5
x2 = runif(500) - 0.5
y = 1 * (x1^2 - x2^2 > 0)
```

(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 $y - axis$.

```
# If the observation is classified to 0 = red, otherwise blue.
plot(x1[y == 0], x2[y == 0], col = 2, xlab = "X1", ylab = "X2", pch = 19)
points(x1[y == 1], x2[y == 1], col = 4, pch = 19)
```



The boundary separating the classes is clearly non-linear and resembles the shape of an X.

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

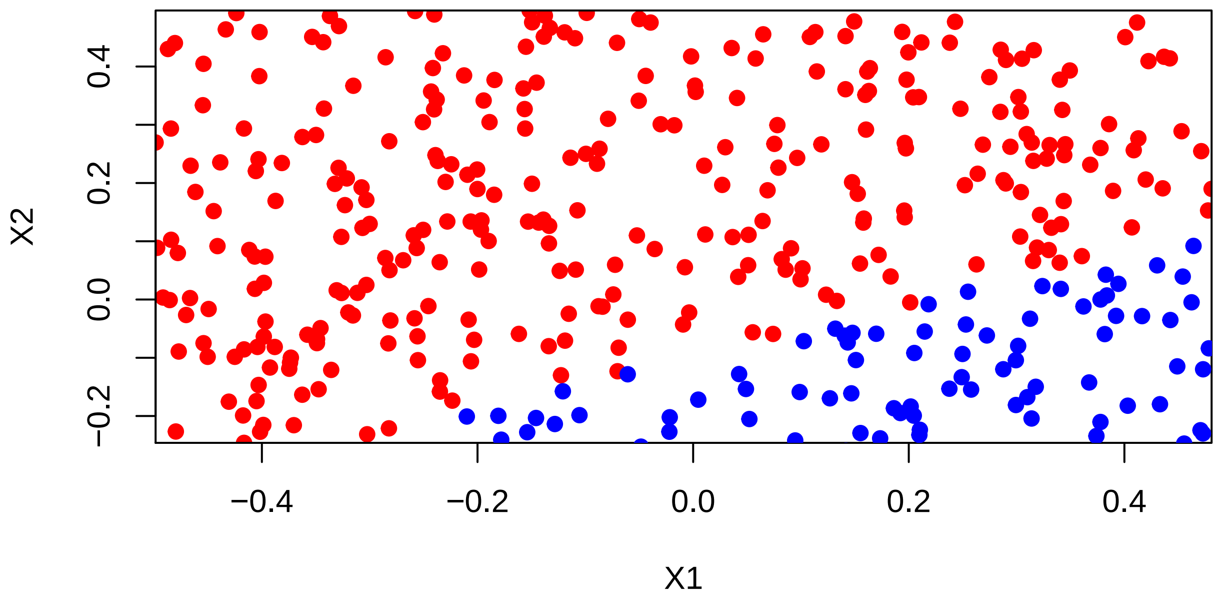
```
# Fitting logistic regression
glm.fit = glm(y ~ x1 + x2, family = "binomial")
summary(glm.fit)
```

```

Call:
glm(formula = y ~ x1 + x2, family = "binomial")

Deviance Residuals:
   Min       1Q   Median       3Q      Max
-1.350   -1.165    1.050    1.151    1.291

Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept)  0.04927    0.08978   0.549    0.583
x1          -0.23002    0.31534  -0.729    0.466
x2           0.51072    0.31560   1.618    0.106

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 692.86  on 499  degrees of freedom
Residual deviance: 689.58  on 497  degrees of freedom
AIC: 695.58

Number of Fisher Scoring iterations: 3
```

Neither of the variables predict y significantly.

(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
# Creating a dataframe of simulated data
data = data.frame(x1 = x1, x2 = x2, y = y)
dataX = data.frame(x1 = x1, x2 = x2)
dataY = data.frame(y = y)

# Predicting responses with the given model
glm.prob = predict(glm.fit, data, type = "response")

# Dividing the predictions into two classes
glm.pred = ifelse(glm.prob > 0.5, 1, 0)

# Creating new variables into dataframe acoording to predicted class
data.sq = data[glm.pred == 1, ]
data.tr = data[glm.pred == 0, ]

# Plotting the predicted classes
plot(data.sq$x1, data.sq$x2, col = 2, xlab = "X1", ylab = "X2", pch = 19)
points(data.tr$x1, data.tr$x2, col = 4, pch = 19)
```



The decision boundary varies with different data sets: for some sets the model predicts that there's only one class and for others there's a linear decision boundary but which varies as well.

(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 , $X_1 \times X_2$, $log(X_2)$, and so forth).

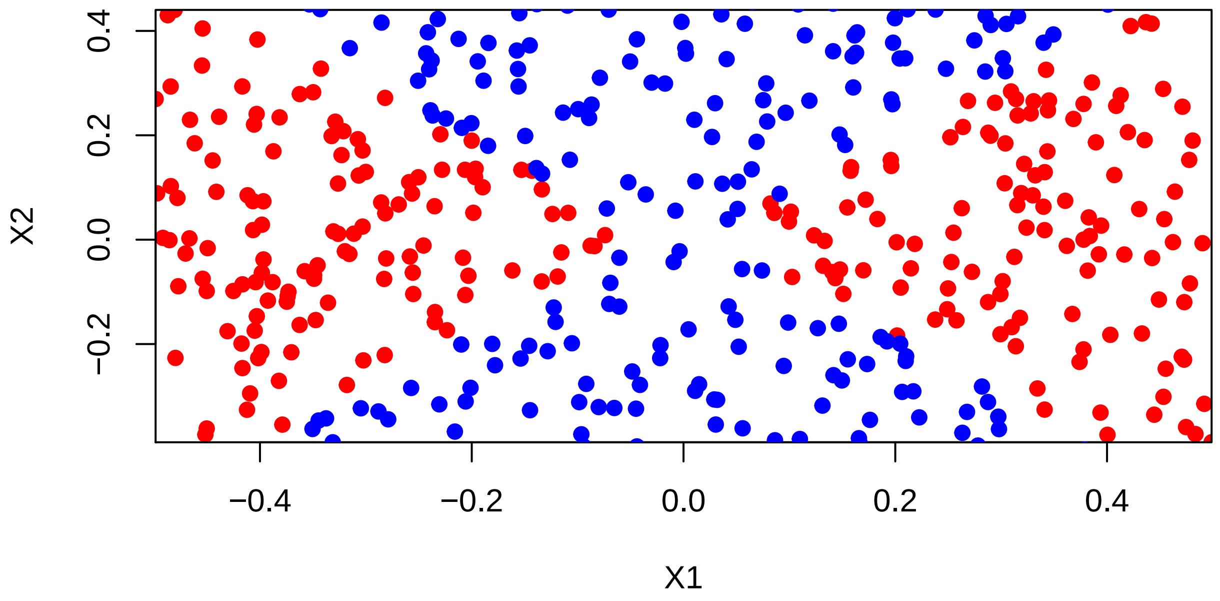
Let's use the following functios in our model: X_1, X_2, X_1^2, X_2^2 and $X_1 \times X_2$.

```r
# Fitting a model with non-linear terms
glm.fit2 = glm(y ~ x1 + I(x1^2) + x2 + I(x2^2) + I(x1 * x2), data = data, family = binomial)
summary(glm.fit2)
```

```
Call:
glm(formula = y ~ x1 + I(x1^2) + x2 + I(x2^2) + I(x1 * x2), family = binomial,
    data = data)

Deviance Residuals:
   Min       1Q   Median       3Q      Max
  0.00     0.00     0.00     0.00     8.49

Coefficients:
              Estimate Std. Error    z value Pr(>|z|)
(Intercept) -9.495e+13  5.833e+06  -16277789   <2e-16 ***
x1           2.075e+14  1.056e+07   19654425   <2e-16 ***
I(x1^2)      3.333e+16  4.324e+07  770827890   <2e-16 ***
x2           3.598e+14  1.055e+07   34107094   <2e-16 ***
I(x2^2)     -3.291e+16  3.978e+07 -827363879   <2e-16 ***
I(x1 * x2)   1.487e+15  3.896e+07   38154411   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 692.86  on 499  degrees of freedom
Residual deviance: 792.96  on 494  degrees of freedom
AIC: 804.96

Number of Fisher Scoring iterations: 22
```

(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.

```r
# Predicting responses with the given model
glm.prob2 = predict(glm.fit2, data, type = "response")

# Dividing the predictions into two classes
glm.pred2 = ifelse(glm.prob2 > 0.5, 1, 0)

# Creating new variables into dataframe acoording to predicted class
data.sq2 = data[glm.pred2 == 1, ]
data.tr2 = data[glm.pred2 == 0, ]

# Fixing margins and origin of the plot
par(mar = c(4, 4, 1, 1))

# Plotting the predicted classes
plot(data.sq2$x1, data.sq2$x2, col = 2, xlab = "X1", ylab = "X2", pch = 19)
points(data.tr2$x1, data.tr2$x2, col = 4, pch = 19)
```



The non-linear decision boudary $\mathbb{P}(\beta_0 + \beta_1 x_1 + \beta_2 x_1^2 + \beta_3 x_2 + \beta_4 x_2^2 + \beta_5 x_1 x_2) > 0.5$ seems to predict the real simuated classes pretty well.

(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.

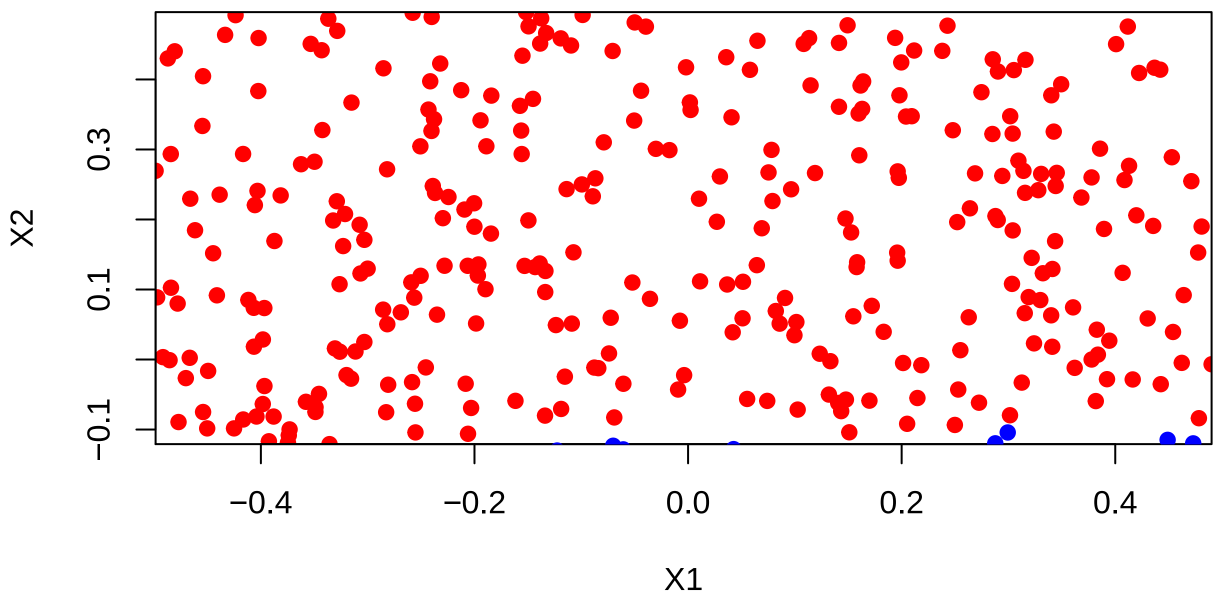
Let's use `tune()` function to do 10-fold cross validation to figure out the best linear kernel svm.

```r
library(e1071)

# Changing y to factor
data$y = as.factor(data$y)

# 10-fold CV
set.seed(12)
tune.out=tune(svm, y~x1+x2, data=data, kernel ="linear",
              ranges=list(cost=c(0.001, 0.01, 0.1, 1,5,10,100)))

summary(tune.out)
```

```
Parameter tuning of 'svm':

- sampling method: 10-fold cross validation

- best parameters:
 cost
    1

- best performance: 0.36

- Detailed performance results:
   cost error dispersion
```

```
1 1e-03 0.488 0.07899367
2 1e-02 0.486 0.08329332
3 1e-01 0.364 0.07530678
4 1e+00 0.360 0.08000000
5 5e+00 0.360 0.08000000
6 1e+01 0.360 0.08000000
7 1e+02 0.362 0.07913420
```

```r
svm.bestlin = tune.out$best.model

# Predictions
svm.pred1 = predict(svm.bestlin, data)

data.sq3 = data[svm.pred1 == 1, ]
data.tr3 = data[svm.pred1 == 0, ]

# Plotting
plot(data.sq3$x1, data.sq3$x2, col=2, xlab="X1", ylab="X2", pch=19)
points(data.tr3$x1, data.tr3$x2, col=4, pch=19)
```



(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.

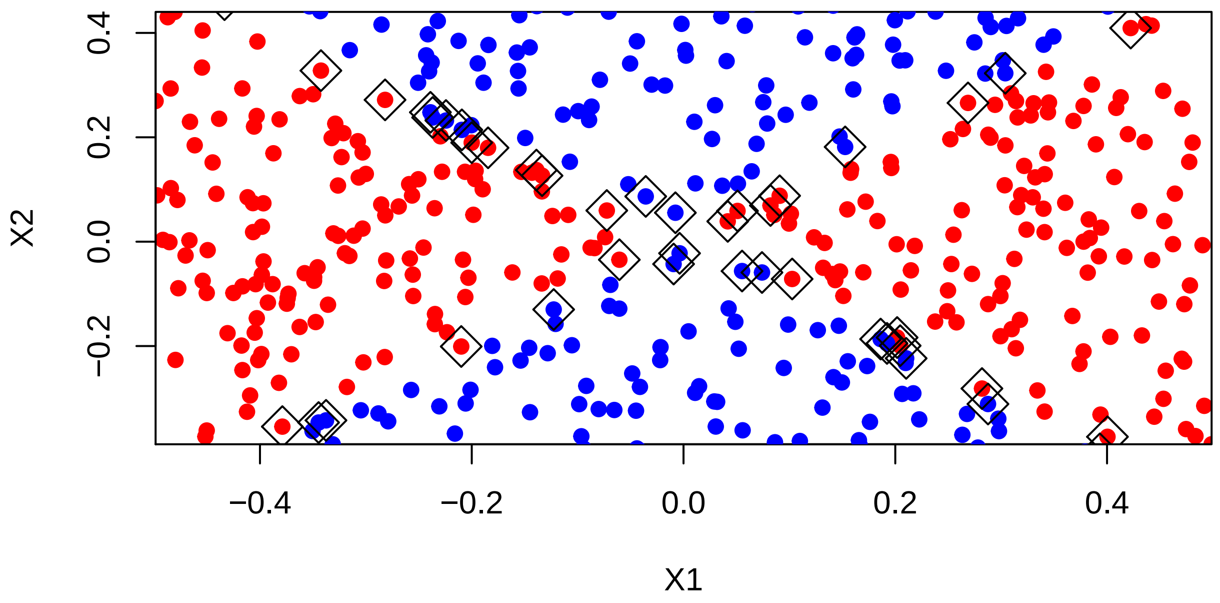
```r
# 10-fold CV
set.seed(12)
tune.out2=tune(svm, y~x1+x2, data=data, kernel ="radial",
               ranges=list(cost=c(0.001, 0.01, 0.1, 1,5,10,100)))

summary(tune.out2)
```

```
Parameter tuning of 'svm':

- sampling method: 10-fold cross validation
- best parameters:
 cost
  100

- best performance: 0.022

- Detailed performance results:
   cost error dispersion
1 1e-03 0.488 0.07899367
2 1e-02 0.488 0.07899367
3 1e-01 0.088 0.03425395
4 1e+00 0.060 0.03887301
5 5e+00 0.044 0.03373096
6 1e+01 0.026 0.02118700
7 1e+02 0.022 0.02201010
```

```
svm.bestunlin = tune.out2$best.model
svm.bestunlin
```

```


Call:
best.tune(method = svm, train.x = y ~ x1 + x2, data = data, ranges = list(cost = c(0.001,
    0.01, 0.1, 1, 5, 10, 100)), kernel = "radial")


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  radial
       cost:  100
      gamma:  0.5

Number of Support Vectors:  42
```

```
# Predictions
svm.pred2 = predict(svm.bestunlin, data)

data.sq4 = data[svm.pred2 == 1, ]
data.tr4 = data[svm.pred2 == 0, ]

# Plotting
plot(data.sq4$x1, data.sq4$x2, col=2, xlab="X1", ylab="X2", pch=19)
points(data.tr4$x1, data.tr4$x2, col=4, pch=19)
points(data$x1[svm.bestunlin$index], data$x2[svm.bestunlin$index], pch = 5, cex = 2)
```



(i) Comment on your results.

With this spesific simulated data, the logistic regression model fails to predict the data because the decision boundary is clearly non-linear which could be characterized as shape of an X. Neither one of the coefficients in the logistic regression model is statistically significant and thus also fail to predict the data. The non linear logistic regression model seems to do pretty well in predicting the true classes if only visual inspection of the plot is used. The support vector classifier (linear kernel) classifies the observations into two class like logistic regression model, though the other class is more underrepresented than with logistic regression. The support vector machine (non-linear kernel) predicts the real classes pretty accurately if only visual inspection is used. It has 42 support vectors which are marked in the plot above.

Exercise 4

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.

```
library(ISLR)

# Calculating median of the gas variable which is used to be te divisive
# criterion
gas.med = median(Auto$mpg)

# Creating a vector of binary variables based on the divisive criterion
new.var = ifelse(Auto$mpg > gas.med, 1, 0)

# Storing the new variable into data set as a factor
Auto$mpglevel = as.factor(new.var)
```

(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.

```r
# 10-fold CV
set.seed(12)
tune.out3 = tune(svm, mpglevel ~ ., data = Auto, kernel = "linear", ranges = list(cost = c(0.01,
    0.1, 1, 5, 10, 100)))
summary(tune.out3)
```

```
Parameter tuning of 'svm':

- sampling method: 10-fold cross validation

- best parameters:
 cost
    1

- best performance: 0.01275641

- Detailed performance results:
   cost      error dispersion
1 1e-02 0.07416667 0.03319890
2 1e-01 0.04596154 0.02651304
3 1e+00 0.01275641 0.01808165
4 5e+00 0.02038462 0.01617396
5 1e+01 0.02038462 0.01617396
6 1e+02 0.03576923 0.02761755
```

The cross-validation error is minimized for cost parameter value of 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.

```r
# Radial kernel
set.seed(12)
tune.outradial = tune(svm, mpglevel ~ ., data = Auto, kernel = "radial", ranges = list(cost = c(0.01,
    0.1, 1, 5, 10, 100), gamma = c(0.01, 0.1, 1, 5, 10, 100)))
summary(tune.outradial)
```

```
Parameter tuning of 'svm':

- sampling method: 10-fold cross validation

- best parameters:
 cost gamma
  100  0.01

- best performance: 0.01532051

- Detailed performance results:
    cost gamma      error dispersion
1  1e-02 1e-02 0.57403846 0.03055966
```

```
2  1e-01 1e-02 0.08942308 0.04057270
3  1e+00 1e-02 0.07416667 0.03319890
4  5e+00 1e-02 0.04852564 0.02826665
5  1e+01 1e-02 0.03070513 0.02357884
6  1e+02 1e-02 0.01532051 0.02474291
7  1e-02 1e-01 0.18096154 0.07700141
8  1e-01 1e-01 0.07929487 0.03737063
9  1e+00 1e-01 0.05621795 0.02921347
10 5e+00 1e-01 0.03064103 0.01626404
11 1e+01 1e-01 0.02801282 0.01875978
12 1e+02 1e-01 0.02794872 0.02510152
13 1e-02 1e+00 0.57403846 0.03055966
14 1e-01 1e+00 0.57403846 0.03055966
15 1e+00 1e+00 0.06121795 0.03238466
16 5e+00 1e+00 0.06371795 0.03227585
17 1e+01 1e+00 0.06371795 0.03227585
18 1e+02 1e+00 0.06371795 0.03227585
19 1e-02 5e+00 0.57403846 0.03055966
20 1e-01 5e+00 0.57403846 0.03055966
21 1e+00 5e+00 0.51525641 0.06957404
22 5e+00 5e+00 0.51275641 0.07072646
23 1e+01 5e+00 0.51275641 0.07072646
24 1e+02 5e+00 0.51275641 0.07072646
25 1e-02 1e+01 0.57403846 0.03055966
26 1e-01 1e+01 0.57403846 0.03055966
27 1e+00 1e+01 0.53570513 0.06113655
28 5e+00 1e+01 0.52544872 0.05714688
29 1e+01 1e+01 0.52544872 0.05714688
30 1e+02 1e+01 0.52544872 0.05714688
31 1e-02 1e+02 0.57403846 0.03055966
32 1e-01 1e+02 0.57403846 0.03055966
33 1e+00 1e+02 0.57403846 0.03055966
34 5e+00 1e+02 0.57403846 0.03055966
35 1e+01 1e+02 0.57403846 0.03055966
36 1e+02 1e+02 0.57403846 0.03055966
```

```r
# Polynomial kernel
set.seed(21)
tune.outpoly = tune(svm, mpglevel ~ ., data = Auto, kernel = "polynomial", ranges = list(cost = c(0.01,
    0.1, 1, 5, 10, 100), degree = c(2, 3, 4)))
summary(tune.outpoly)
```

```

Parameter tuning of 'svm':

- sampling method: 10-fold cross validation

- best parameters:
 cost degree
  100      2

- best performance: 0.3338462

- Detailed performance results:
    cost degree     error dispersion
```

```
1   1e-02      2 0.5867308 0.07310319
2   1e-01      2 0.5867308 0.07310319
3   1e+00      2 0.5867308 0.07310319
4   5e+00      2 0.5867308 0.07310319
5   1e+01      2 0.5508974 0.11697667
6   1e+02      2 0.3338462 0.08816986
7   1e-02      3 0.5867308 0.07310319
8   1e-01      3 0.5867308 0.07310319
9   1e+00      3 0.5867308 0.07310319
10  5e+00      3 0.5867308 0.07310319
11  1e+01      3 0.5867308 0.07310319
12  1e+02      3 0.3514103 0.13939284
13  1e-02      4 0.5867308 0.07310319
14  1e-01      4 0.5867308 0.07310319
15  1e+00      4 0.5867308 0.07310319
16  5e+00      4 0.5867308 0.07310319
17  1e+01      4 0.5867308 0.07310319
18  1e+02      4 0.5867308 0.07310319
```

It seems that the radial kernel performs better than polynomial kernel even with the same cost parameter, though it may be due to limiting the degree of the polynomial only up to 4. The cost parameter value which minimizes the cross-validation error is 100 for both radial and polynomial kernels. The gamma for radial basis kernel is 0.01 and the degree of the polynomial is 2 for polynomial basis kernel svm.

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

Hint: In the lab, we used the plot() function for svm objects only in cases with $p = 2$. When $p > 2$, you can use theplot() function to create plots displaying pairs of variables at a time. Essentially, instead of typing

```
plot(svmfit, dat)
```

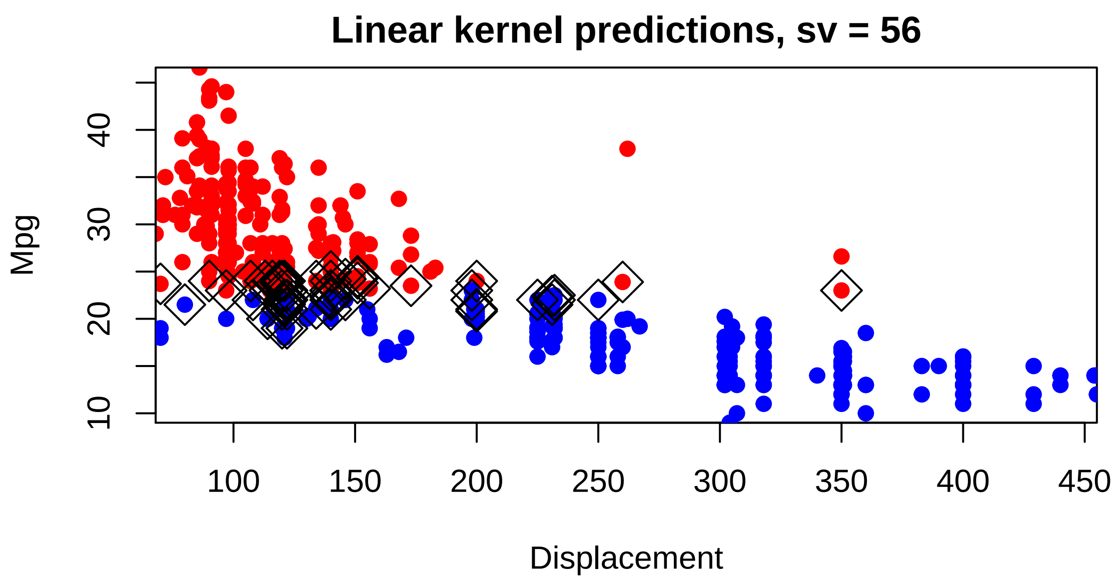
where svmfit contains your fitted model and dat is a data frame containing your data, you can type

```
plot(svmfit, dat, x1∼x4)
```

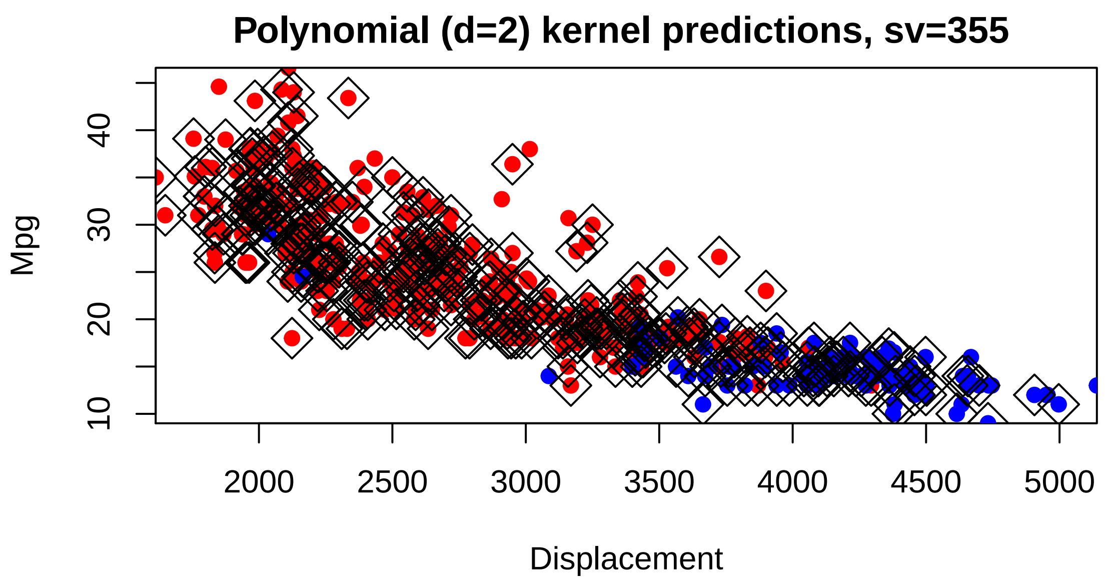
in order to plot just the first and fourth variables. However, you must replace x1 and x4 with the correct variable names. To findout more, type ?plot.svm.

```r
# Forming the models according to minimum cv error
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 = 100, gamma = 0.01)

# Predictions
svm.predlin = predict(svm.linear, Auto)
svm.predpoly = predict(svm.poly, Auto)
svm.predrad = predict(svm.radial, Auto)

Auto$lin = ifelse(svm.predlin == 1, 1, 0)
Auto$poly = ifelse(svm.predpoly == 1, 1, 0)
Auto$radial = ifelse(svm.predrad == 1, 1, 0)

# Plotting
par(mar = c(5, 4, 2, 4))

# Linear kernel and displacement
```

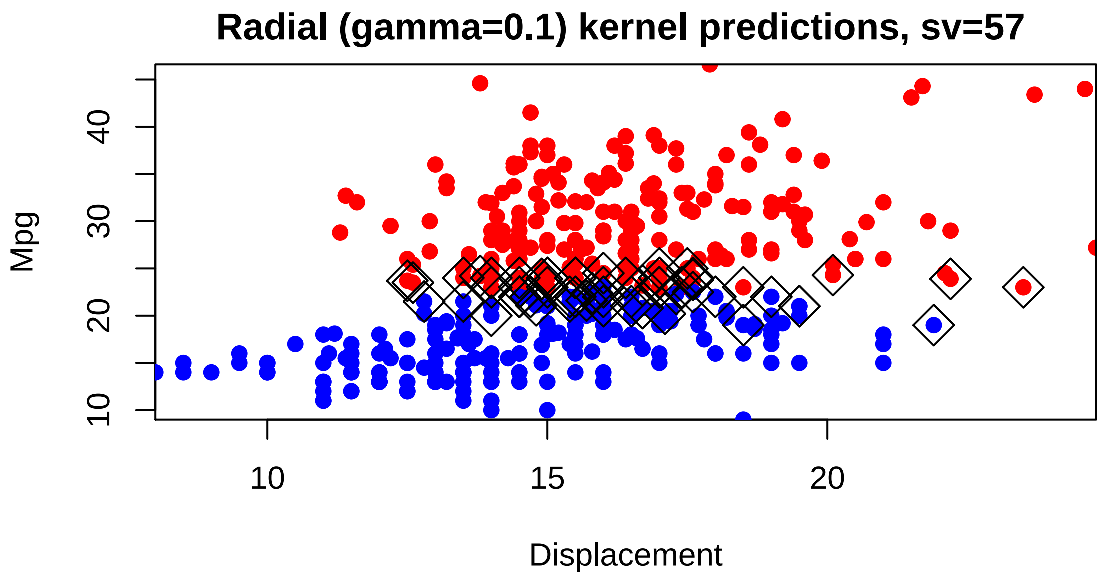
```r
plot(Auto$displacement[Auto$lin == 1], Auto$mpg[Auto$lin == 1], col = 2, xlab = "Displacement",
    ylab = "Mpg", main = "Linear kernel predictions, sv = 56", pch = 19, ylim = c(min(Auto$mpg),
        max(Auto$mpg)), xlim = c(min(Auto$displacement), max(Auto$displacement)))
points(Auto$displacement[Auto$lin == 0], Auto$mpg[Auto$lin == 0], col = 4, pch = 19)
points(Auto$displacement[svm.linear$index], Auto$mpg[svm.linear$index], pch = 5,
    cex = 2)
```



```r
# Polynomial kernel and weight
plot(Auto$weight[Auto$poly == 1], Auto$mpg[Auto$poly == 1], col = 2, xlab = "Displacement",
    ylab = "Mpg", main = "Polynomial (d=2) kernel predictions, sv=355 ", pch = 19,
    ylim = c(min(Auto$mpg), max(Auto$mpg)), xlim = c(min(Auto$weight), max(Auto$weight)))
points(Auto$weight[Auto$poly == 0], Auto$mpg[Auto$poly == 0], col = 4, pch = 19)
points(Auto$weight[svm.poly$index], Auto$mpg[svm.poly$index], pch = 5, cex = 2)
```



```
# Radial kernel and acceleration
plot(Auto$acceleration[Auto$radial == 1], Auto$mpg[Auto$radial == 1], col = 2,
    xlab = "Displacement", ylab = "Mpg", main = "Radial (gamma=0.1) kernel predictions, sv=57 ",
    pch = 19, ylim = c(min(Auto$mpg), max(Auto$mpg)), xlim = c(min(Auto$acceleration),
        max(Auto$acceleration)))
points(Auto$acceleration[Auto$radial == 0], Auto$mpg[Auto$radial == 0], col = 4,
    pch = 19)
points(Auto$acceleration[svm.radial$index], Auto$mpg[svm.radial$index], pch = 5,
    cex = 2)
```



Exercise 5

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.

```
set.seed(31)
train = sample(dim(OJ)[1], 800)
OJ.train = OJ[train, ]
OJ.test = OJ[-train, ]
```

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

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


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  linear
       cost:  0.01
      gamma:  0.05555556

Number of Support Vectors:  440

 ( 220 220 )


Number of Classes:  2

Levels:
 CH MM
```

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

```r
# Training error
train.pred = predict(svm.linearPur, OJ.train)
table(OJ.train$Purchase, train.pred)
```

```
    train.pred
      CH  MM
  CH 436  53
  MM  84 227
```

```r
paste("Training error is", round((84 + 53)/(436 + 53 + 84 + 227), 2))
```

```
[1] "Training error is 0.17"
```

```r
# Test error
test.pred = predict(svm.linearPur, OJ.test)
table(OJ.test$Purchase, test.pred)
```

```
    test.pred
      CH  MM
  CH 145  19
  MM  24  82
```

```r
paste("Test error is", round((19 + 24)/(145 + 19 + 24 + 82), 2))
```

```
[1] "Test error is 0.16"
```

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

```r
set.seed(54)
tune.outOJ = tune(svm, Purchase ~ ., data = OJ.train, kernel = "linear", ranges = list(cost = 10^seq(-2
    1, by = 0.25)))
summary(tune.outOJ)
```

```
Parameter tuning of 'svm':

- sampling method: 10-fold cross validation

- best parameters:
       cost
 0.05623413

- best performance: 0.17875

- Detailed performance results:
          cost   error dispersion
1   0.01000000 0.18000 0.04609772
2   0.01778279 0.18125 0.05628857
3   0.03162278 0.18125 0.05628857
4   0.05623413 0.17875 0.05337563
5   0.10000000 0.17875 0.04966904
6   0.17782794 0.18000 0.05075814
7   0.31622777 0.18125 0.04497299
8   0.56234133 0.17875 0.04604120
9   1.00000000 0.17875 0.05402224
10  1.77827941 0.17875 0.05653477
11  3.16227766 0.17875 0.05923412
12  5.62341325 0.18000 0.05839283
13 10.00000000 0.18000 0.05688683
```

```r
paste("The cv error is smallest for cost =", round(tune.outOJ$best.parameters[1,
    1], 2))
```

```
[1] "The cv error is smallest for cost = 0.06"
```

```r
bestmodOJ = tune.outOJ$best.model
paste("Number of support vectors is", length(bestmodOJ$index))
```

```
[1] "Number of support vectors is 367"
```

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

```r
# Training error
train.pred2 = predict(bestmodOJ, OJ.train)
table(OJ.train$Purchase, train.pred2)
```

```
    train.pred2
      CH  MM
  CH 433  56
  MM  78 233
```

```r
paste("Training error is", round((78 + 56)/(433 + 56 + 78 + 233), 4))
```

```
[1] "Training error is 0.1675"
```

```r
# Test error
test.pred2 = predict(bestmodOJ, OJ.test)
table(OJ.test$Purchase, test.pred2)
```

```
    test.pred2
      CH  MM
```

```
  CH 145  19
  MM  20  86
```

```
paste("Test error is", round((19 + 20)/(145 + 19 + 20 + 86), 4))
```

```
[1] "Test error is 0.1444"
```

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

```
# Fixed cost = 0.01
svm.radPur = svm(Purchase ~ ., kernel = "radial", data = OJ.train, cost = 0.01)
summary(svm.radPur)
```

```


Call:
svm(formula = Purchase ~ ., data = OJ.train, kernel = "radial",
    cost = 0.01)


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  radial
       cost:  0.01
      gamma:  0.05555556

Number of Support Vectors:  627

 ( 316 311 )


Number of Classes:  2

Levels:
 CH MM
```

```
# 10-fold cv for cost parameter and fixed gamma
set.seed(62)
tune.outOJrad = tune(svm, Purchase ~ ., data = OJ.train, kernel = "radial",
    ranges = list(cost = 10^seq(-2, 1, by = 0.25)))
summary(tune.outOJrad)
```

```

Parameter tuning of 'svm':

- sampling method: 10-fold cross validation

- best parameters:
      cost
 0.5623413

- best performance: 0.17375

- Detailed performance results:
          cost   error dispersion
1   0.01000000 0.38875 0.05382908
```

```
2   0.01778279 0.38875 0.05382908
3   0.03162278 0.38375 0.05466120
4   0.05623413 0.20625 0.06187184
5   0.10000000 0.18750 0.05432669
6   0.17782794 0.18000 0.04721405
7   0.31622777 0.17875 0.05172376
8   0.56234133 0.17375 0.05573063
9   1.00000000 0.17625 0.05846711
10  1.77827941 0.17875 0.05239076
11  3.16227766 0.19250 0.05533986
12  5.62341325 0.19500 0.06297045
13 10.00000000 0.19375 0.06434769
```

```
paste("The cv error is smallest for cost =", round(tune.outOJrad$best.parameters[1,
    1], 2))
```

```
[1] "The cv error is smallest for cost = 0.56"
```

```
bestmodOJrad = tune.outOJrad$best.model
paste("Number of support vectors is", length(bestmodOJrad$index))
```

```
[1] "Number of support vectors is 408"
```

```
# Training error
train.pred3 = predict(bestmodOJrad, OJ.train)
table(OJ.train$Purchase, train.pred3)
```

```
    train.pred3
      CH  MM
  CH 445  44
  MM  84 227
```

```
paste("Training error is", round((44 + 84)/(445 + 44 + 84 + 227), 4))
```

```
[1] "Training error is 0.16"
```

```
# Test error
test.pred3 = predict(bestmodOJrad, OJ.test)
table(OJ.test$Purchase, test.pred3)
```

```
    test.pred3
      CH  MM
  CH 146  18
  MM  25  81
```

```
paste("Test error is", round((18 + 25)/(146 + 18 + 25 + 81), 4))
```

```
[1] "Test error is 0.1593"
```

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

```
# Fixed cost = 0.01
svm.polPur = svm(Purchase ~ ., kernel = "polynomial", data = OJ.train, cost = 0.01,
    degree = 2)
summary(svm.polPur)
```

```


Call:
```

```
svm(formula = Purchase ~ ., data = OJ.train, kernel = "polynomial",
    cost = 0.01, 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:  627

 ( 316 311 )


Number of Classes:  2

Levels:
 CH MM
```

```
# 10-fold cv for cost parameter and fixed gamma
set.seed(62)
tune.outOJpoly = tune(svm, Purchase ~ ., data = OJ.train, kernel = "polynomial",
    ranges = list(cost = 10^seq(-2, 1, by = 0.25)), degree = 2)
summary(tune.outOJpoly)
```

```

Parameter tuning of 'svm':

- sampling method: 10-fold cross validation

- best parameters:
 cost
   10

- best performance: 0.1825

- Detailed performance results:
          cost   error dispersion
1   0.01000000 0.38875 0.05382908
2   0.01778279 0.37250 0.05767485
3   0.03162278 0.36375 0.05447030
4   0.05623413 0.34000 0.04518481
5   0.10000000 0.33500 0.04632314
6   0.17782794 0.26375 0.03928617
7   0.31622777 0.21625 0.04931827
8   0.56234133 0.21250 0.06236096
9   1.00000000 0.20125 0.06652328
10  1.77827941 0.19500 0.06324555
11  3.16227766 0.19875 0.06520534
12  5.62341325 0.18375 0.05138701
13 10.00000000 0.18250 0.05109903
```

```r
paste("The cv error is smallest for cost =", round(tune.outOJpoly$best.parameters[1,
    1], 2))
```

```
[1] "The cv error is smallest for cost = 10"
```

```r
bestmodOJpoly = tune.outOJpoly$best.model
paste("Number of support vectors is", length(bestmodOJpoly$index))
```

```
[1] "Number of support vectors is 357"
```

```r
# Training error
train.pred4 = predict(bestmodOJpoly, OJ.train)
table(OJ.train$Purchase, train.pred4)
```

```
    train.pred4
      CH  MM
  CH 452  37
  MM  89 222
```

```r
paste("Training error is", round((37 + 89)/(452 + 37 + 89 + 222), 4))
```

```
[1] "Training error is 0.1575"
```

```r
# Test error
test.pred4 = predict(bestmodOJpoly, OJ.test)
table(OJ.test$Purchase, test.pred4)
```

```
    test.pred4
      CH  MM
  CH 148  16
  MM  28  78
```

```r
paste("Test error is", round((16 + 28)/(148 + 16 + 28 + 78), 4))
```

```
[1] "Test error is 0.163"
```

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

The cross-validation method is overall the best method for each base of kernel. Both training and test set errors are pretty similar with all approaches but the radial linear kernel seems to perform best on this data and it needs fewer support vectors than the other methods. Note that the results may vary with different random seeds.