# Homework 4 - Classifiers

## Homework 4 Requirements

There is no separate instruction file. This file is the instructions and you will modify it for your submission.

You will submit two files.

The files you submit will be:

1. `102b_hw4_output_First_Last.Rmd` Take this R Markdown file and make the necessary edits so that it generates the requested output.
2. `102b_hw4_output_First_Last.pdf` Your output file. This is the primary file that will be graded. **Make sure all requested output is visible in the output file.**

### Academic Integrity

Modifying the following statement with your name.

"By including this statement, I, Neftali Lemus, declare that all of the work in this assignment is my own original work. At no time did I look at the code of other students nor did I search for code solutions online. I understand that plagiarism on any single part of this assignment will result in a 0 for the entire assignment and that I will be referred to the dean of students."

If you collaborated verbally with other students, please also include the following line to credit them.

"I did discuss ideas related to the homework with Josephine Bruin for parts 2 and 3, with John Wooden for part 2, and with Gene Block for part 1. At no point did I show another student my code, nor did I look at another student's code." Homework questions and text copyright Miles Chen. For personal use only. Do not distribute. Do not post or share your solutions.

# Part 1 Naive Bayes Classifier for Iris data

Task: Write a function that performs Naive Bayes classification for the iris data. The function will output probability estimates of the species for a test case.

The function will accept three inputs: a row matrix for the x values of the test case, a matrix of x values for the training data, and a vector of class labels for the training data.

The function will create the probability estimates based on the training data it has been provided.

Within the function use a Gaussian model and estimate the mean and standard deviation of the Gaussian populations based on the training data provided. (Hint: You have 24 parameters to estimate: the mean and standard deviation of each of the 4 variables for each of the three species. With the naive assumption, you do not have to estimate any covariances.)

```
library(dplyr)
```

```
##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
iris_nb <- function(testx, trainx, trainy){
  # 3 SPECIES * 4 VARIABLES * 2 MEASUREMENTS = 24 PARAMETERS
  #MEANS
  #Setosa
  x1bar_a <- mean(trainx[which(trainy == "setosa"),1]) #SL
  x1bar_b <- mean(trainx[which(trainy == "setosa"),2]) # SW
  x1bar_c <- mean(trainx[which(trainy == "setosa"),3]) # PL
  x1bar_d <- mean(trainx[which(trainy == "setosa"),4]) # PW
  #Versicolor
  x2bar_a <- mean(trainx[which(trainy == "versicolor"),1]) #SL
  x2bar_b <- mean(trainx[which(trainy == "versicolor"),2]) # SW
  x2bar_c <- mean(trainx[which(trainy == "versicolor"),3]) # PL
  x2bar_d <- mean(trainx[which(trainy == "versicolor"),4]) # PW
  #Virginica
  x3bar_a <- mean(trainx[which(trainy == "virginica"),1]) #SL
  x3bar_b <- mean(trainx[which(trainy == "virginica"),2]) # SW
  x3bar_c <- mean(trainx[which(trainy == "virginica"),3]) # PL
  x3bar_d <- mean(trainx[which(trainy == "virginica"),4]) # PW
  # STANDARD DEVIATIONS
  #Setosa
  sd1_a <- sd(trainx[which(trainy == "setosa"),1]) #SL
  sd1_b <- sd(trainx[which(trainy == "setosa"),2]) # SW
  sd1_c <- sd(trainx[which(trainy == "setosa"),3]) # PL
  sd1_d <- sd(trainx[which(trainy == "setosa"),4]) # PW
  #Versicolor
  sd2_a <- sd(trainx[which(trainy == "versicolor"),1]) #SL
  sd2_b <- sd(trainx[which(trainy == "versicolor"),2]) # SW
  sd2_c <- sd(trainx[which(trainy == "versicolor"),3]) # PL
  sd2_d <- sd(trainx[which(trainy == "versicolor"),4]) # PW
  #Virginica
  sd3_a <- sd(trainx[which(trainy == "virginica"),1]) #SL
  sd3_b <- sd(trainx[which(trainy == "virginica"),2]) # SW
  sd3_c <- sd(trainx[which(trainy == "virginica"),3]) # PL
  sd3_d <- sd(trainx[which(trainy == "virginica"),4]) # PW

  # Likelihood
  like_setosa <-
    dnorm(testx[1], mean = x1bar_a, sd = sd1_a) *
    dnorm(testx[2], mean = x1bar_b, sd = sd1_b) *
    dnorm(testx[3], mean = x1bar_c, sd = sd1_c) *
    dnorm(testx[4], mean = x1bar_d, sd = sd1_d);like_setosa # Setosa
  like_versicolor <-
    dnorm(testx[1], mean = x2bar_a, sd = sd2_a) *
    dnorm(testx[2], mean = x2bar_b, sd = sd2_b) *
    dnorm(testx[3], mean = x2bar_c, sd = sd2_c) *
    dnorm(testx[4], mean = x2bar_d, sd = sd2_d);like_versicolor # Versicolor
  like_virginica <-
    dnorm(testx[1], mean = x3bar_a, sd = sd3_a) *
    dnorm(testx[2], mean = x3bar_b, sd = sd3_b) *
    dnorm(testx[3], mean = x3bar_c, sd = sd3_c) *
    dnorm(testx[4], mean = x3bar_d, sd = sd3_d);like_virginica # Virginica
  #Prior: training_y's frequency is 40 for each specie
    prior_setosa = prior_versicolor = prior_virginica = 1/3
  # Marginal = product of likelihood and prior
  marginal <- like_setosa * prior_setosa + like_versicolor * prior_versicolor + like_virginica * prior_v
  # Posterior Class Probability
  a <- like_setosa * prior_setosa / marginal
  b <- like_versicolor * prior_versicolor / marginal
  c <- like_virginica * prior_virginica / marginal
  # Output vector
  output <- c("Setosa" = a , "Versicolor" = b , "Virginica" = c )
  output

}
```

```
### output should be a named vector that looks something like this:
## [these numbers are completely made up btw]
    setosa versicolor  virginica
 0.9518386  0.0255936  0.0225678
```

```
set.seed(1)
training_rows <- sort(c(sample(1:50, 40), sample(51:100, 40), sample(101:150, 40)))
training_x <- as.matrix(iris[training_rows, 1:4])
training_y <- iris[training_rows, 5]

# test cses
test_case_a <- as.matrix(iris[24, 1:4]) # true class setosa
```

```
test_case_b <- as.matrix(iris[73, 1:4]) # true class versicolor
test_case_c <- as.matrix(iris[124, 1:4]) # true class virginica

# class predictions of test cases
iris_nb(test_case_a, training_x, training_y) # Predicts 100 % Setosa
```

### Testing it out

```
##       Setosa   Versicolor    Virginica
## 1.000000e+00 1.029887e-13 4.098385e-18
```

```
iris_nb(test_case_b, training_x, training_y) # Predicts 90.3 % Versicolor
```

```
##        Setosa    Versicolor     Virginica
## 2.980587e-115  9.034742e-01  9.652578e-02
```

```
iris_nb(test_case_c, training_x, training_y) # Predicts  90.4  % Virginica
```

```
##        Setosa    Versicolor     Virginica
## 6.078393e-136  9.540725e-02  9.045928e-01
```

```
# should work and produce slightly different estimates based on new training data
set.seed(10)
training_rows2 <- sort(c(sample(1:50, 25), sample(51:100, 25), sample(101:150, 25)))
training_x2 <- as.matrix(iris[training_rows2, 1:4])
training_y2 <- iris[training_rows2, 5]

iris_nb(test_case_a, training_x2, training_y2) # Predicts setosa 100 %
```

```
##       Setosa   Versicolor    Virginica
## 1.000000e+00 2.600180e-10 4.610354e-17
```

```
iris_nb(test_case_b, training_x2, training_y2) # Predicts versicolor 91.9 %
```

```
##        Setosa    Versicolor     Virginica
## 1.735964e-131  9.195738e-01  8.042615e-02
```

```
iris_nb(test_case_c, training_x2, training_y2) # Predicts virginica 81.2%
```

```
##        Setosa    Versicolor     Virginica
## 3.400709e-149  1.887116e-01  8.112884e-01
```

### Naive Bayes with R

While instructive and education (I hope) to write your own NaiveBayes function, in practical settings, I recommend using the production ready code from some time-tested packages.

I've included some code for using the `naiveBayes()` function that is part of the `e1071` package. No need to modify anything. The results prediced by `naiveBayes()` should match the results from the function you wrote.

```r
# code provided. no need to edit. These results should match your results above.
library(e1071)
nb_model1 <- naiveBayes(training_x, training_y)
predict(nb_model1, newdata = test_case_a, type = 'raw')
```

```
##      setosa   versicolor    virginica
## [1,]      1 1.029887e-13 4.098385e-18
```

```r
predict(nb_model1, newdata = test_case_b, type = 'raw')
```

```
##             setosa versicolor  virginica
## [1,] 2.980587e-115  0.9034742 0.09652578
```

```r
predict(nb_model1, newdata = test_case_c, type = 'raw')
```

```
##             setosa versicolor virginica
## [1,] 6.078393e-136 0.09540725 0.9045928
```

```r
nb_model2 <- naiveBayes(training_x2, training_y2)
predict(nb_model2, newdata = test_case_a, type = 'raw')
```

```
##      setosa  versicolor    virginica
## [1,]      1 2.60018e-10 4.610354e-17
```

```r
predict(nb_model2, newdata = test_case_b, type = 'raw')
```

```
##             setosa versicolor  virginica
## [1,] 1.735964e-131  0.9195738 0.08042615
```

```r
predict(nb_model2, newdata = test_case_c, type = 'raw')
```

```
##             setosa versicolor virginica
## [1,] 3.400709e-149  0.1887116 0.8112884
```

## Part 2: K-nearest neighbors Classifier for the Iris data

Task: Write a classifier using the K-nearest neighbors algorithm for the iris data set.

First write a function that will calculate the euclidean distance from a vector A (in 4-dimensional space) to another vector B (also in 4-dimensional space). If you're struggling with four dimensions, think about how you might calculate it in two dimensions, then three, then four.

Use that function to find the k nearest neighbors to then make a classification.

The function will accept four inputs: a row matrix for the x values of the test case, a matrix of x values for the training data, a vector of class labels for the training data, and the k parameter.

The function will return a single label.

```
#Mode Function for Labels
mode_function <- function(x) {
  uniqx <- unique(na.omit(x))
  uniqx[which.max(tabulate(match(x, uniqx)))]
}

#Euclidean Distance
distance <- function(a, b) {
  vet <- numeric(nrow(b))
  for( i in 1:nrow(b))
vet[i]<- sqrt( (a[1] - b[i,1])^2 + (a[2] - b[i,2])^2 + (a[3] -b[i,3])^2 + (a[4] - b[i,4])^2 )
vet
}

#Iris Knn
iris_knn <- function(testx, trainx, trainy, k) {
  #Ordered in decreasing Order
  dec_or <- order(distance(testx,trainx))
  closest_dp <- dec_or[1:k]
  lbs_closest <- trainy[closest_dp]
  mode_function(lbs_closest)
}
```

```
iris_knn(test_case_a, training_x, training_y, 5)
```

```
## [1] setosa
## Levels: setosa versicolor virginica
```

```
iris_knn(test_case_b, training_x, training_y, 5)
```

```
## [1] virginica
## Levels: setosa versicolor virginica
```

```
iris_knn(test_case_c, training_x, training_y, 5)
```

```
## [1] virginica
## Levels: setosa versicolor virginica
```

```
iris_knn(test_case_a, training_x2, training_y2, 5)
```

```
## [1] setosa
## Levels: setosa versicolor virginica
```

```
iris_knn(test_case_b, training_x2, training_y2, 5)
```

```
## [1] versicolor
## Levels: setosa versicolor virginica
```

```r
iris_knn(test_case_c, training_x2, training_y2, 5)
```

```
## [1] virginica
## Levels: setosa versicolor virginica
```

### KNN with R

Again, if you plan on using KNN in real-life, use a function from a package.

I've included some code for using the `knn()` function that is part of the `class` package. No need to modify anything. The results prediced by `knn()` should match the results from the function you wrote, including the misclassification of some of the test cases based on the training data.

```r
library(class)
knn(train = training_x, cl = training_y, test = test_case_a, k = 5)
```

```
## [1] setosa
## Levels: setosa versicolor virginica
```

```r
knn(train = training_x, cl = training_y, test = test_case_b, k = 5)
```

```
## [1] virginica
## Levels: setosa versicolor virginica
```

```r
knn(train = training_x, cl = training_y, test = test_case_c, k = 5)
```

```
## [1] virginica
## Levels: setosa versicolor virginica
```

```r
knn(train = training_x2, cl = training_y2, test = test_case_a, k = 5)
```

```
## [1] setosa
## Levels: setosa versicolor virginica
```

```r
knn(train = training_x2, cl = training_y2, test = test_case_b, k = 5)
```

```
## [1] versicolor
## Levels: setosa versicolor virginica
```

```r
knn(train = training_x2, cl = training_y2, test = test_case_c, k = 5)
```

```
## [1] virginica
## Levels: setosa versicolor virginica
```

## Part 3: SVM

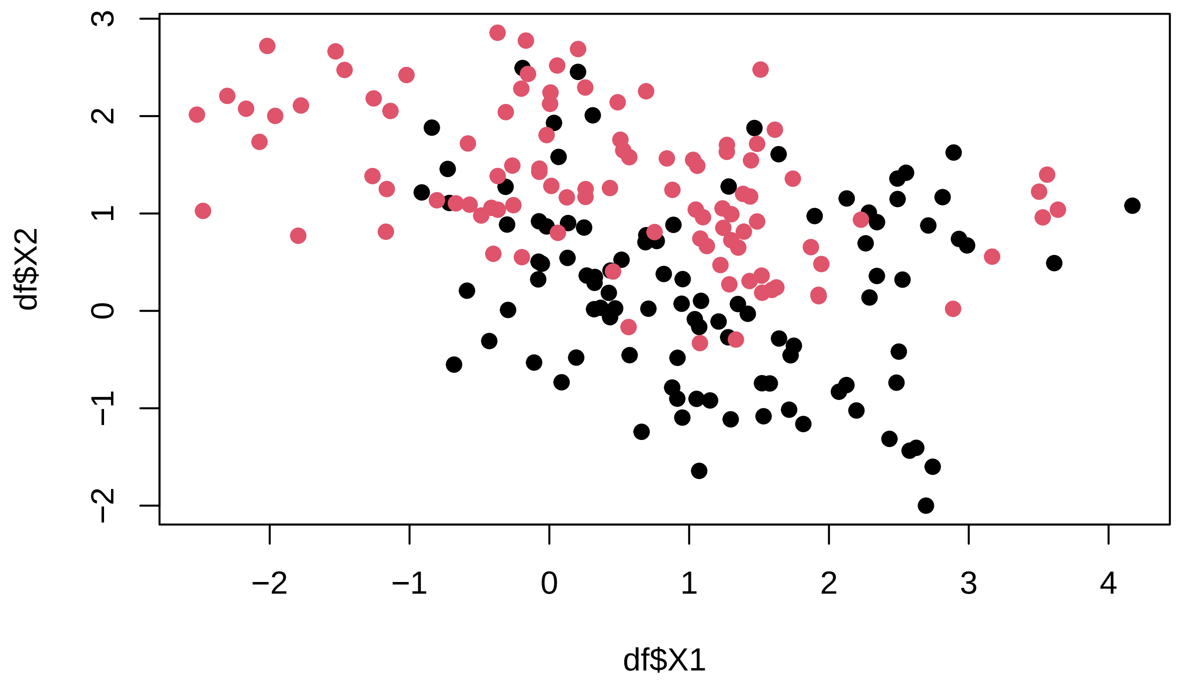
Manual implementation of SVM is a bit of a pain (quadratic programming is hard), and I will not include it in the hw.

For the interested student, I refer them to the code from book's companion github repository: https://github.com/sdrogers/fcmlcode/blob/master/R/chapter5/svmhard.R and this post on stackexchange: https://stats.stackexchange.com/questions/179900/optimizing-a-support-vector-machine-with-quadratic-programming

Instead, I will use an example of a mixture model that can be separated via SVM.

The mixture model comes from the excellent (but advanced) textbook, *The Elements of Statistical Learning*, which the authors have made to be freely available at: https://hastie.su.domains/ElemStatLearn/

```r
load("df.Rdata")
plot(df$X1,df$X2, col = df$y, pch = 19) # create a plot of the mixture
```



We will use the `svm()` function available in package `e1071`.

Read the documentation on the function `svm()`.

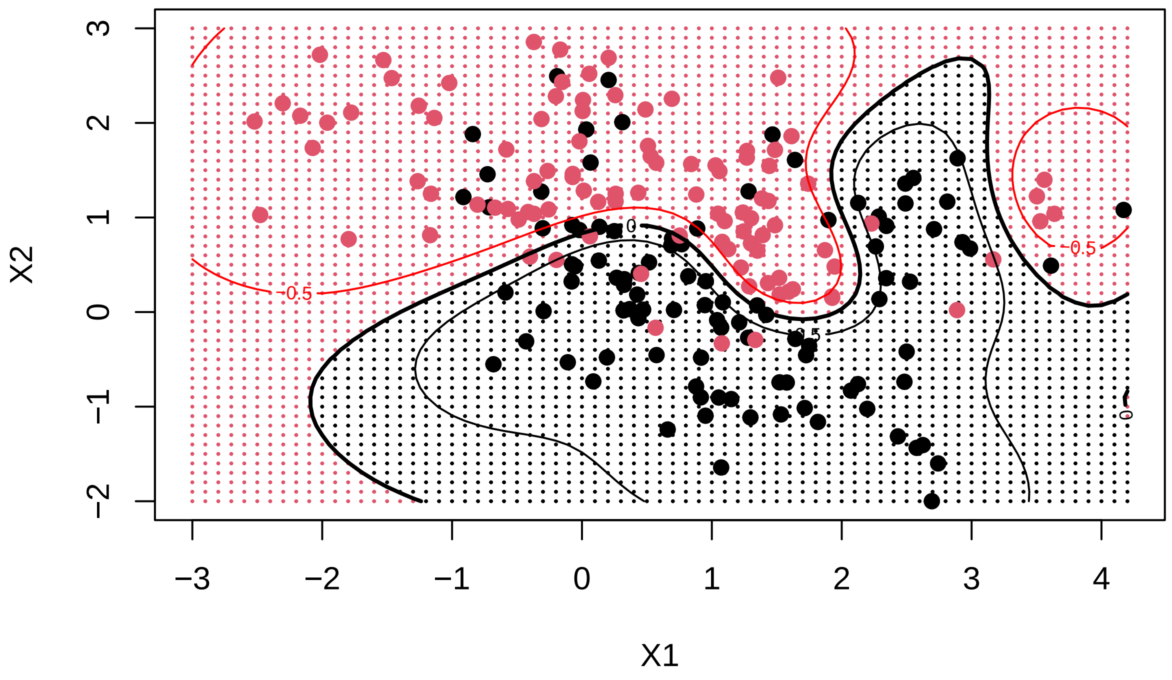
For the following models, we will use a **radial-basis function**, which is equivalent to using a Gaussian Kernel function. (The Gaussian Kernel function projects the 2-dimensional data into infinite dimensional space and takes the inner product of these infinite dimensional vectors. It doesn't actually do this, but the resulting inner product can be found and used to draw a decision boundary.)

The svm function allows for multiple arguments, but we will focus on the effect of the arguments for gamma and cost.

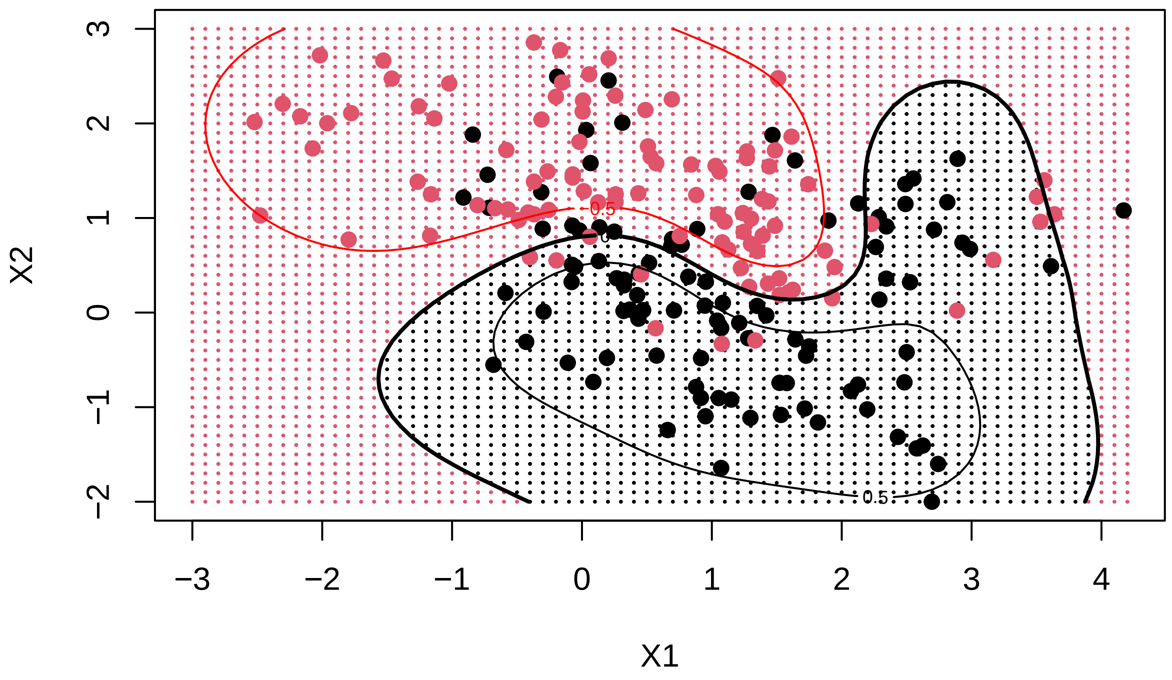
I have created 9 classification models using SVM and different values of gamma and cost.

Pay attention to the values of `gamma` and `cost`. At the very end comment on the effect of each parameter on the resulting model.

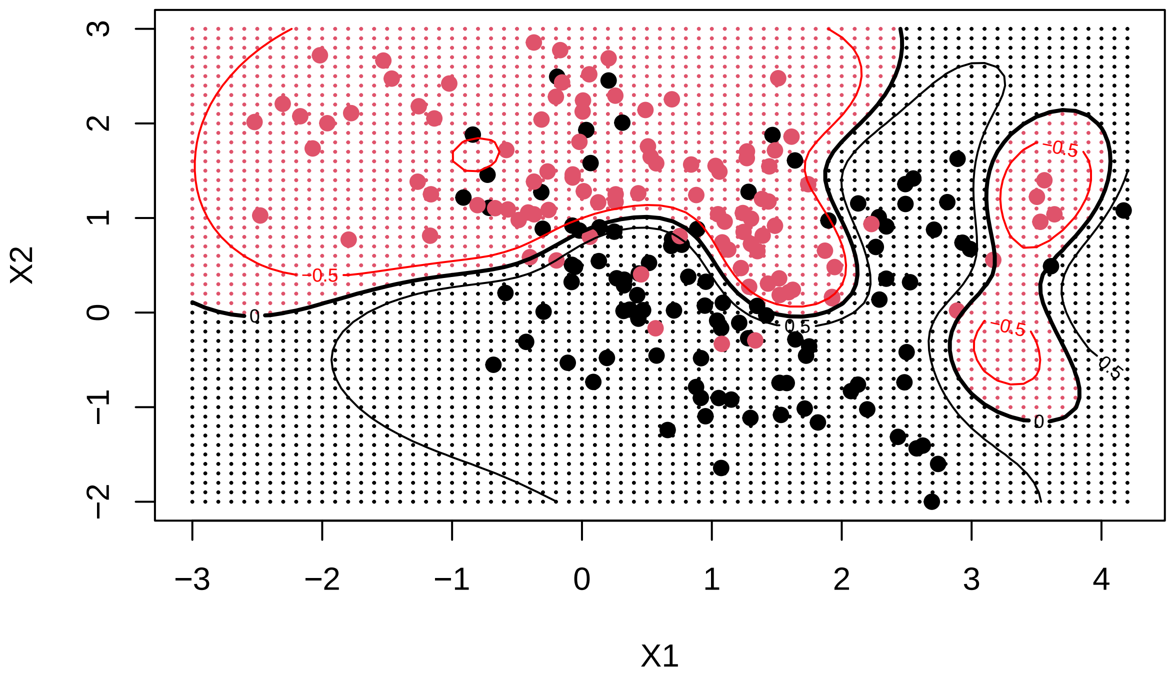
```r
library(e1071)
model <- svm(y ~ . , data = df, scale = FALSE, kernel = "radial", gamma = 1, cost = 1)
```



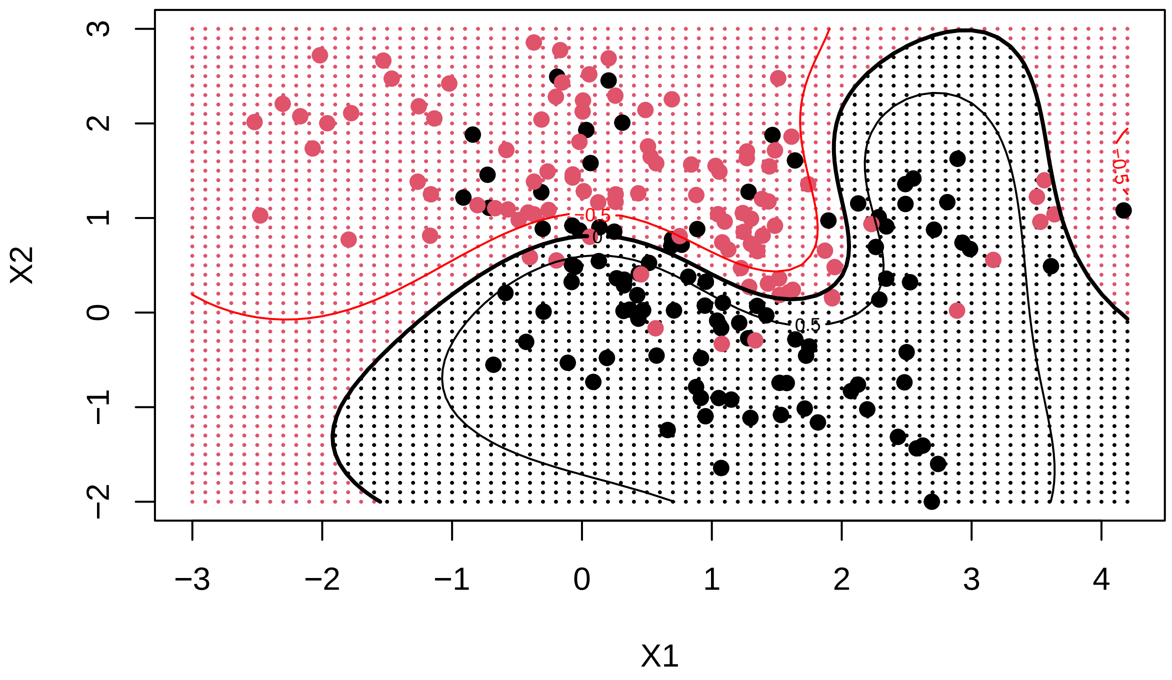
```r
model <- svm(y ~ . , data = df, scale = FALSE, kernel = "radial", gamma = 1, cost = 0.1)
```



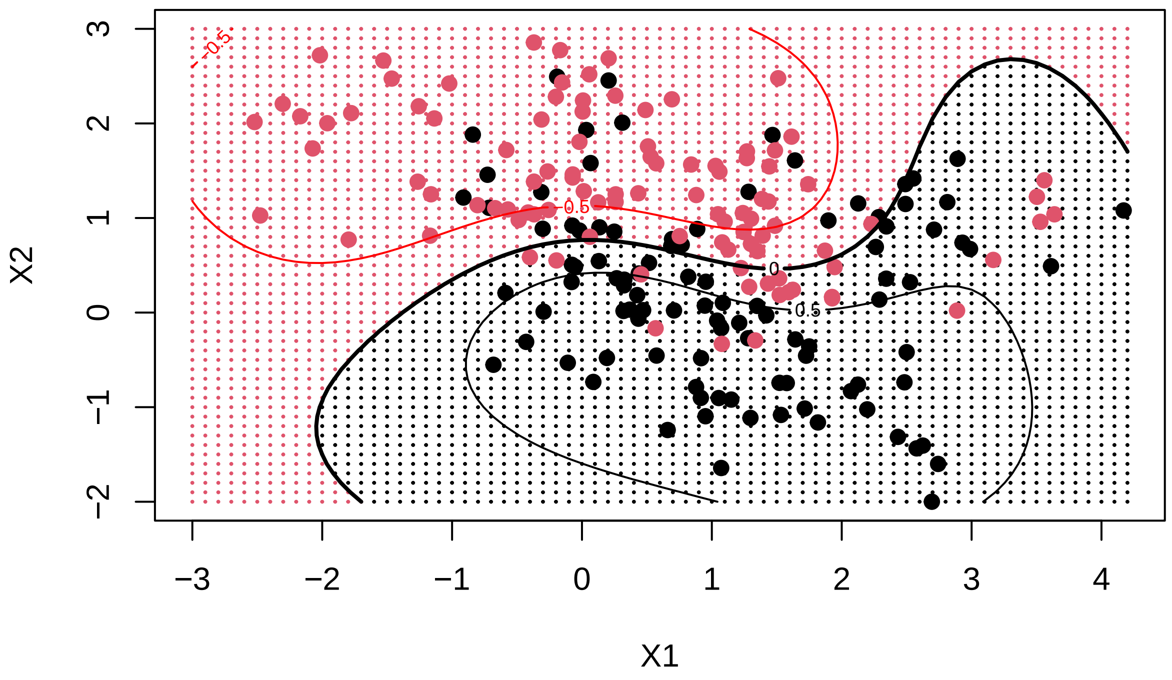
```r
model <- svm(y ~ . , data = df, scale = FALSE, kernel = "radial", gamma = 1, cost = 10)
```



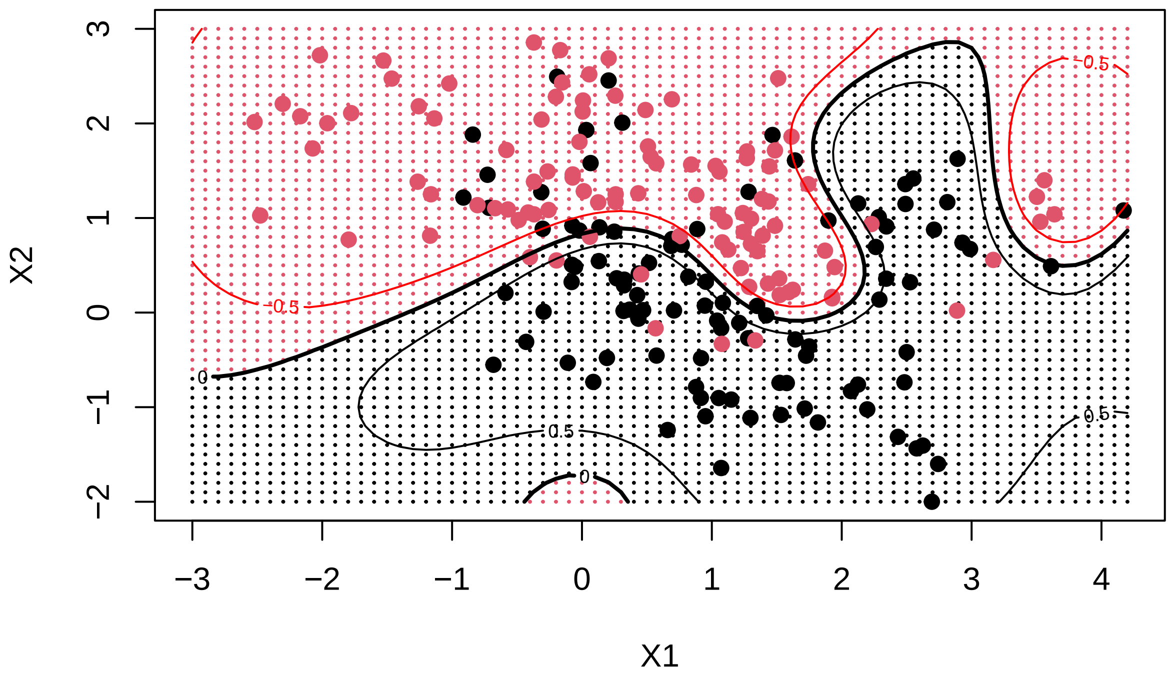
```
model <- svm(y ~ . , data = df, scale = FALSE, kernel = "radial", gamma = 0.5, cost = 1)
```



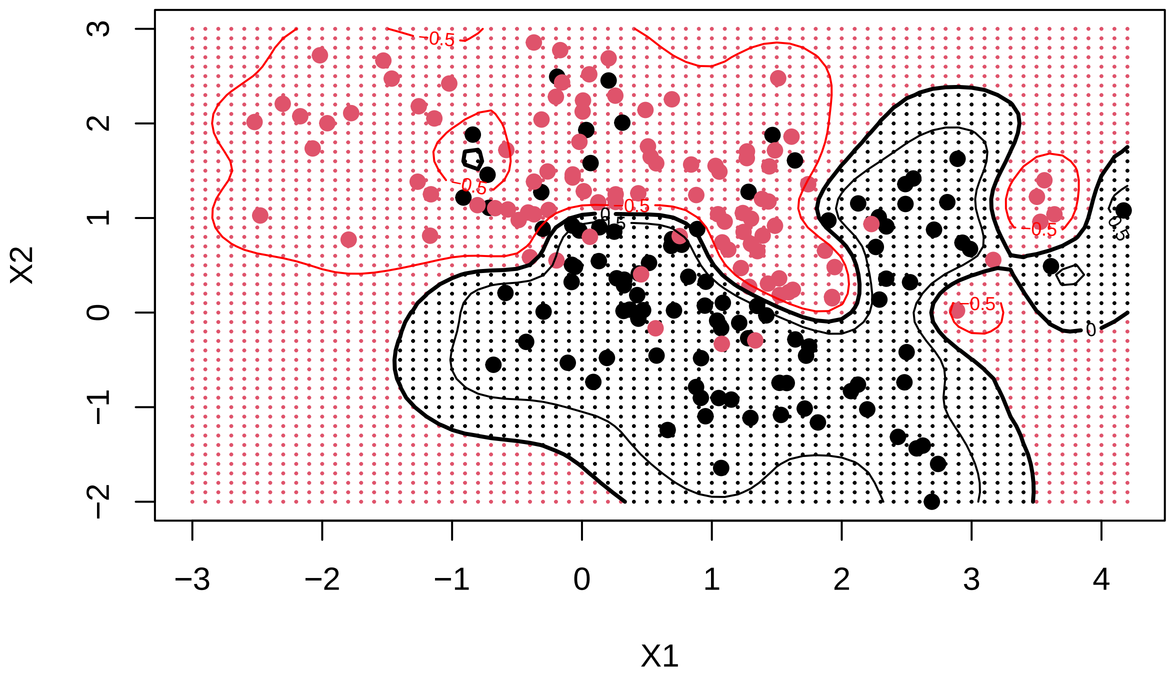
```
model <- svm(y ~ . , data = df, scale = FALSE, kernel = "radial", gamma = 0.5, cost = 0.10)
```



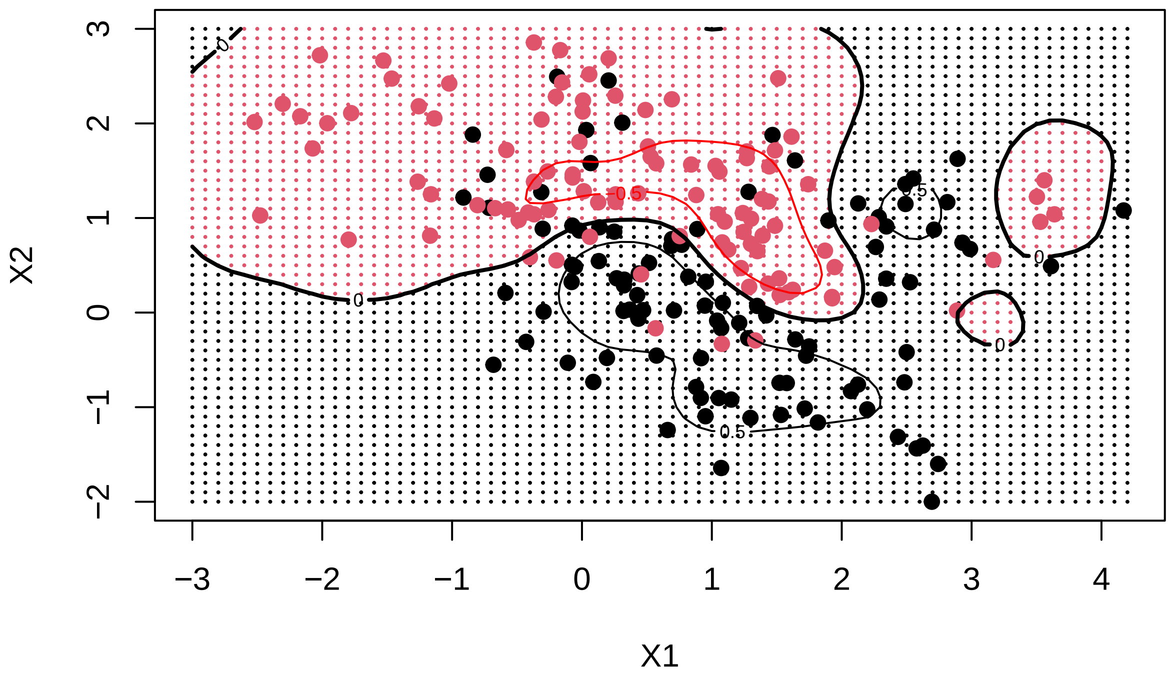
```
model <- svm(y ~ . , data = df, scale = FALSE, kernel = "radial", gamma = 0.5, cost = 10)
```



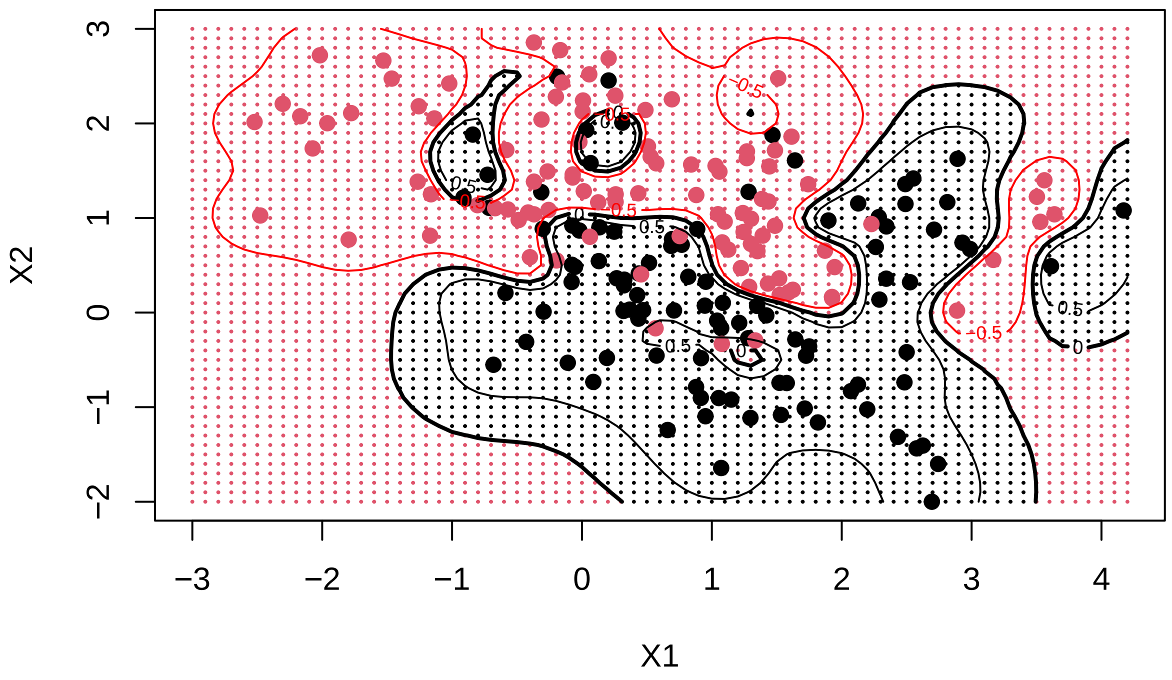
```r
model <- svm(y ~ . , data = df, scale = FALSE, kernel = "radial", gamma = 5, cost = 1)
```



```r
model <- svm(y ~ . , data = df, scale = FALSE, kernel = "radial", gamma = 5, cost = 0.1)
```



```r
model <- svm(y ~ . , data = df, scale = FALSE, kernel = "radial", gamma = 5, cost = 10)
```



**Write about the effect of the cost paramter:** Cost is a tuning parameter that takes care of outliers, as it increases, the graph's complexity increases, this leads to complex regions of classification for the distribution of the data. An example of this is the small red region that forms at the top right corner of the graph as cost increases.

**Write about the effect of the gamma parameter:** The Gamma parameter is the maximum distance between the separating line and the data points. As it increases, the separating lines get closer to the actual data points and the boundaries become well defined.