Support Vector Machines

W. Evan Johnson, Ph.D.
Professor, Division of Infectious Disease
Director, Center for Data Science
Rutgers University – New Jersey Medical School

10/30/2023

Support Vector Machines

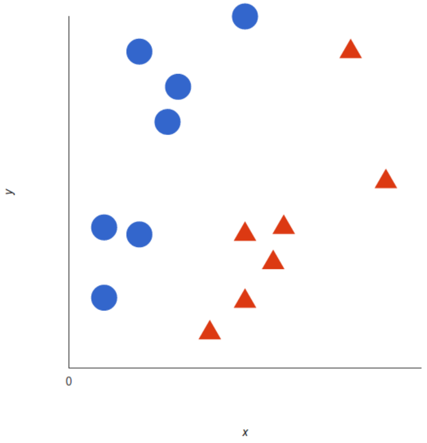
In machine learning, **support vector machines** or **SVM**s are supervised learning models with associated learning algorithms that analyze data used for classification and regression analysis. However, they are mostly used in classification problems. Here we will focus on developing intuition rather than rigor and develop a basic understanding of the working principles.

Material for this lecture was obtained and adapted from:

- https://www.datacamp.com/community/tutorials/support-vector-machines-r
- *The Elements of Statistical Learning*, Hastie, et al., Springer
- https://www.geeksforgeeks.org/classifying-data-using-support-vector-machinessvms-in-r/amp/

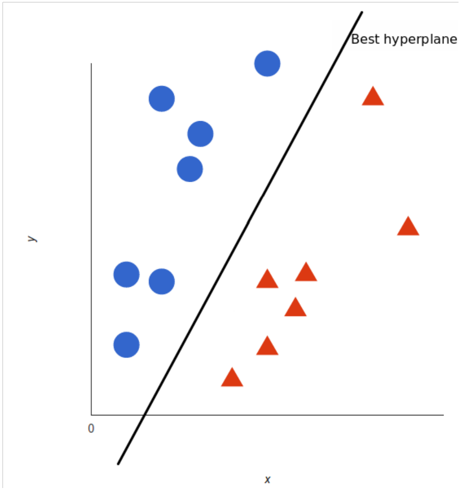
Support Vector Machines–Linear Data

Let's imagine we have two tags: *red* and *blue*, and our data has two features: x and y. We can plot our training data on a plane:



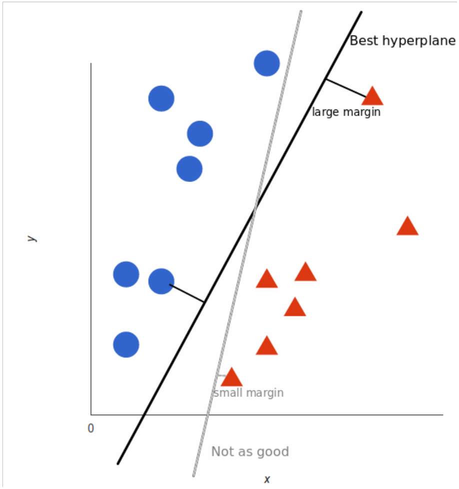
Support Vector Machines

An **SVM** identifies the **decision boundary** or **hyperplane** (two dimensions: line) that best separates the tags:



Support Vector Machines

But, what exactly is the best hyperplane? For SVM, it's the one that maximizes the margins from the data from both tags:



A Look into SVM Methodology

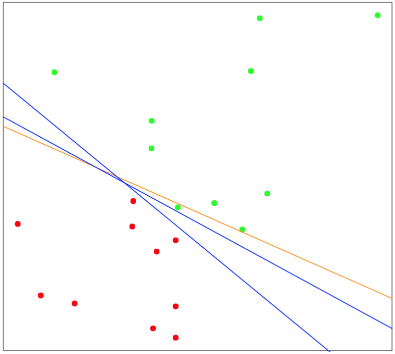


FIGURE 4.14. *A toy example with two classes separable by a hyperplane. The orange line is the least squares solution, which misclassifies one of the training points. Also shown are two blue separating hyperplanes found by the* perceptron learning algorithm *with different random starts.*

A Look into SVM Methodology

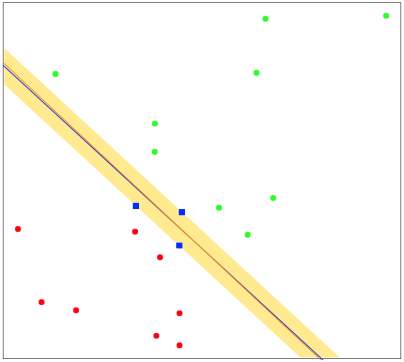


FIGURE 4.16. *The same data as in Figure 4.14. The shaded region delineates the maximum margin separating the two classes. There are three support points indicated, which lie on the boundary of the margin, and the optimal separating hyperplane (blue line) bisects the slab. Included in the figure is the boundary found using logistic regression (red line), which is very close to the optimal separating hyperplane (see Section 12.3.3).*

A Look into SVM Methodology

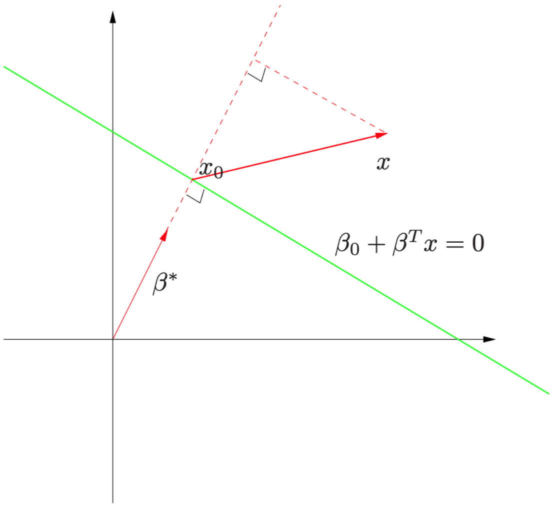


FIGURE 4.15. *The linear algebra of a hyperplane (affine set).*

A Look into SVM Methodology

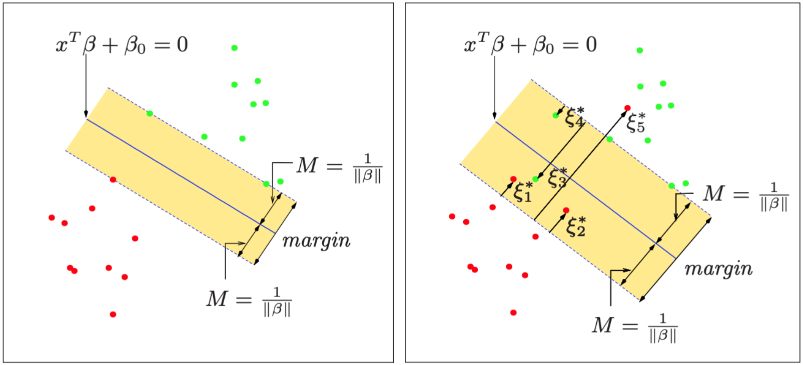
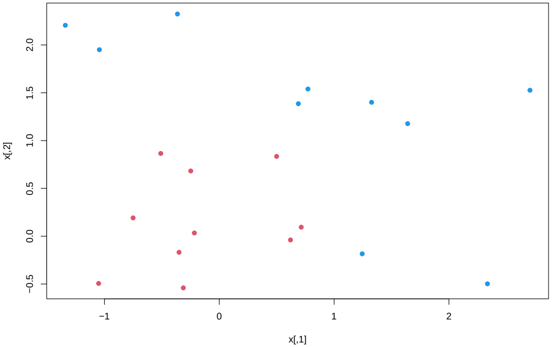


FIGURE 12.1. *Support vector classifiers. The left panel shows the separable case. The decision boundary is the solid line, while broken lines bound the shaded maximal margin of width $2M = 2/\|\beta\|$. The right panel shows the nonseparable (overlap) case. The points labeled ξ_j^* are on the wrong side of their margin by an amount $\xi_j^* = M\xi_j$; points on the correct side have $\xi_j^* = 0$. The margin is maximized subject to a total budget $\sum \xi_i \leq$ constant. Hence $\sum \xi_j^*$ is the total distance of points on the wrong side of their margin.*

Support Vector Machines in R

First generate some data in 2 dimensions, and make them a little separated:

```r
set.seed(10111)
x = matrix(rnorm(40), 20, 2)
y = rep(c(-1, 1), c(10, 10))
x[y == 1,] = x[y == 1,] + 1
plot(x, col = y + 3, pch = 19)
```



Support Vector Machines in R

We will use the **e1071** package which contains the svm function and make a dataframe of the data, turning y into a factor variable.

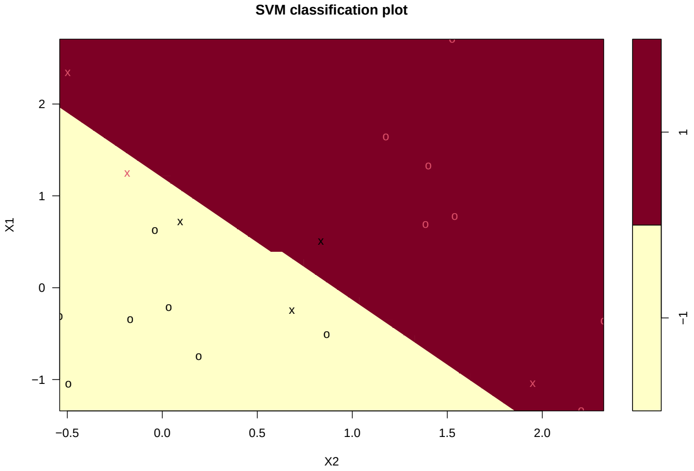
```r
library(e1071)
dat = data.frame(x, y = as.factor(y))
svmfit = svm(y ~ ., data = dat, kernel = "linear", cost = 10, scale = FALSE)
print(svmfit)
```

```
##
## Call:
## svm(formula = y ~ ., data = dat, kernel = "linear", cost = 10, scale = FALSE)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  10
##
## Number of Support Vectors:  6
```

Support Vector Machines in R

There's a plot function for SVM that shows the decision boundary

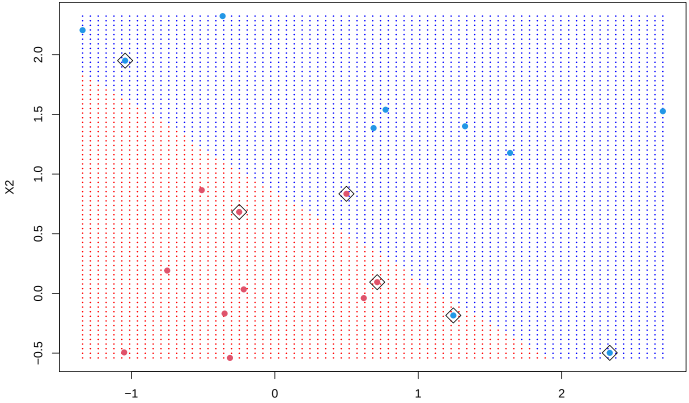
```
plot(svmfit, dat)
```



Support Vector Machines in R

Or plotting it more cleanly:

```
ygrid = predict(svmfit, xgrid)
plot(xgrid, col = c("red","blue")[as.numeric(ygrid)], pch = 20, cex = .2)
points(x, col = y + 3, pch = 19)
points(x[svmfit$index,], pch = 5, cex = 2)
```



Support Vector Machines in R

Unfortunately, the svm function is not too friendly, in that you have to do some work to get back the linear coefficients. The reason is probably that this only makes sense for linear kernels, and the function is more general. So let's use a formula to extract the coefficients more efficiently. You extract β and β_0, which are the linear coefficients.

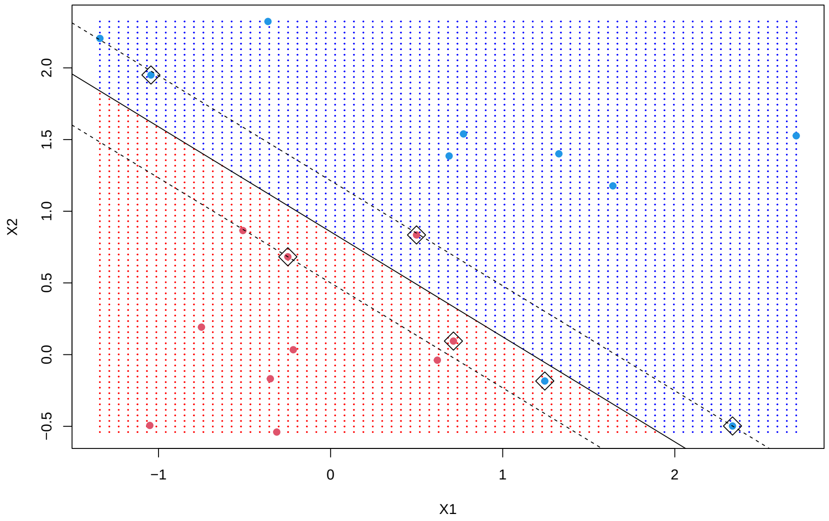
```r
beta = drop(t(svmfit$coefs)%*%x[svmfit$index,])
beta0 = svmfit$rho
```

Now you can replot the points on the grid, then put the points back in (including the support vector points). Then you can use the coefficients to draw the decision boundary using a simple equation of the form:

$$\beta_0 + x_1\beta_1 + x_2\beta_2 = 0$$

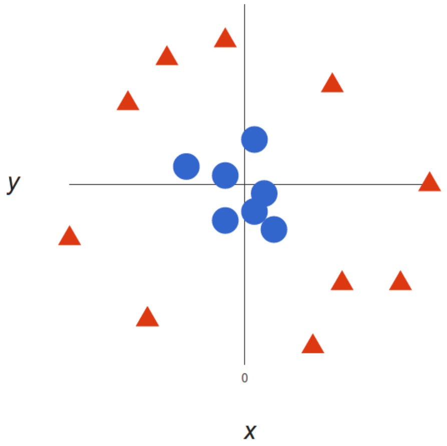
Support Vector Machines in R

Now plotting the lines on the graph:



Support Vector Machines–Non-Linear Data

Now the examples before were easy since clearly, the data was linearly separable. Often things aren't that simple. Take a look at this case:



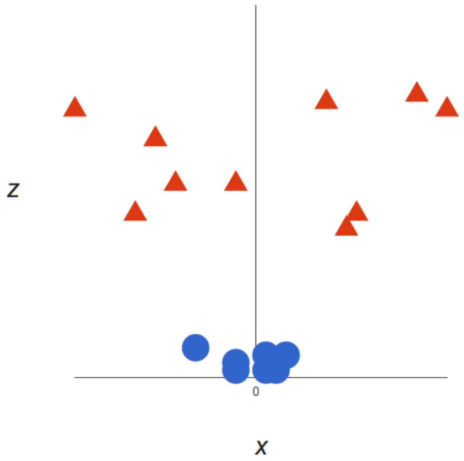
Support Vector Machines

It's pretty clear that there's not a linear decision boundary (a single straight line that separates both tags). However, the vectors are very clearly segregated, and it looks as though it should be easy to separate them.

So here's what we'll do: we will add a third dimension. Up until now, we had two dimensions: x and y. We create a new z dimension, and we rule that it be calculated a certain way that is convenient for us: $z = x^2 + y^2$ (you'll notice that's the equation for a circle).

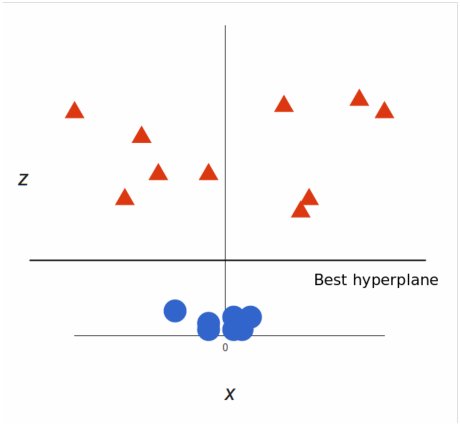
Support Vector Machines

his will give us a three-dimensional space. Taking a slice of that space:



Support Vector Machines

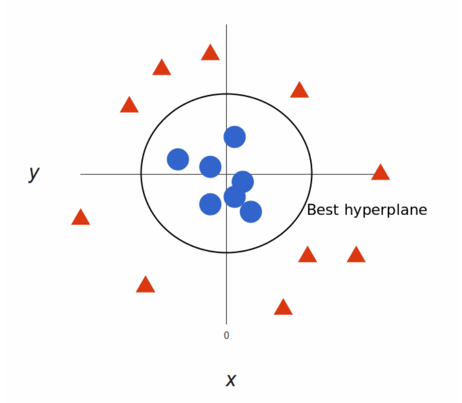
What can SVM do with this? Let's see:



That's great! Note that since we are in three dimensions now, the hyperplane is a plane parallel to the x axis at a certain z (let's say $z=1$).

Support Vector Machines

What's left is mapping it back to two dimensions:



And there we go! Our decision boundary is a circumference of radius 1, which separates both tags using SVM.

Kernel Trick

In this example, we found a way to classify nonlinear data by cleverly mapping our space to a higher dimension. However, it turns out that calculating this transformation can get pretty computationally expensive: there can be a lot of new dimensions, each one of them possibly involving a complicated calculation. Doing this for every vector in the dataset can be a lot of work, so it'd be great if we could find a cheaper solution.

Kernel Trick

Here's a trick: SVM doesn't need the actual vectors to work its magic, it actually can get by only with the dot products between them. This means that we can sidestep the expensive calculations of the new dimensions! This is what we do instead:

- Imagine the new space we want:

$$z = x^2 + y^2$$

- Figure out what the dot product in that space looks like:

$$a \cdot b = x_a \cdot x_b + y_a \cdot y_b + z_a \cdot z_b = x_a \cdot x_b + y_a \cdot y_b + (x_a^2 + y_a^2) \cdot (x_b^2 + y_b^2)$$

- Tell SVM to do its thing, but using the new dot product—we call this a **kernel function**.

Kernel Trick

This often called the **kernel trick**, which enlarges the feature space in order to accommodate a non-linear boundary between the classes.

Common types of kernels used to separate non-linear data are *polynomial* kernels, *radial basis* kernels, and *linear* kernels (which are the same as support vector classifiers). Simply, these kernels transform our data to pass a linear hyperplane and thus classify our data.

Support Vector Machines in R: Non-linear SVM

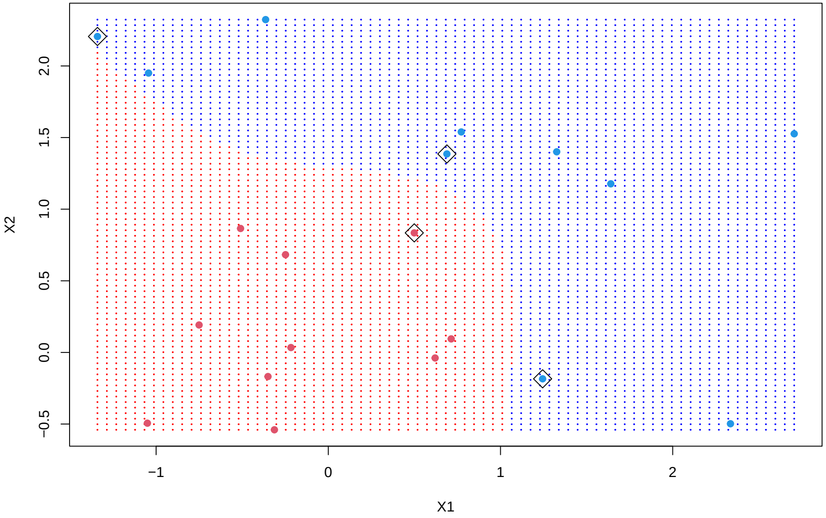
Now let's apply a non-linear (polynomial) SVM to our prior simulated dataset.

```r
library(e1071)
dat = data.frame(x, y = as.factor(y))
svmfit = svm(y ~ ., data = dat,
             kernel = "polynomial", cost = 10, scale = FALSE)
print(svmfit)
```

```
##
## Call:
## svm(formula = y ~ ., data = dat, kernel = "polynomial", cost = 10,
##     scale = FALSE)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  polynomial
##        cost:  10
##      degree:  3
##      coef.0:  0
##
## Number of Support Vectors:  4
```

Support Vector Machines in R: Non-linear SVM

Plotting the result:



Support Vector Machines in R: Non-linear SVM

Here is a more complex example from *Elements of Statistical Learning*, where the decision boundary needs to be non-linear and there is no clear separation.

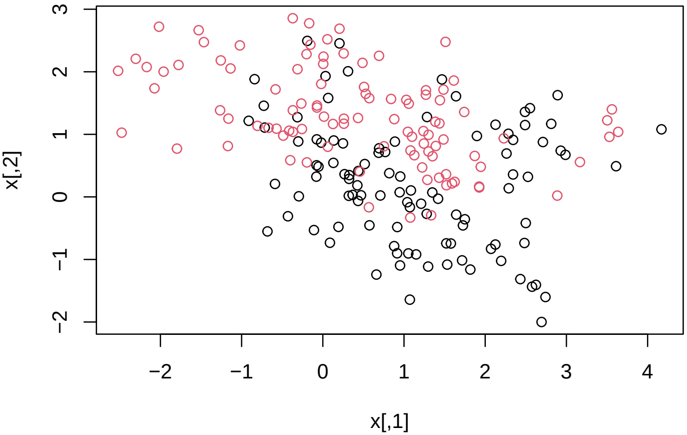
```
#download.file(
#  "http://www-stat.stanford.edu/~tibs/ElemStatLearn/datasets/ESL.mixture.rda",
#  destfile='ESL.mixture.rda')
rm(x,y)
load(file = "ESL.mixture.rda")
attach(ESL.mixture)
names(ESL.mixture)
```

```
## [1] "x"          "y"          "xnew"       "prob"       "marginal" "px1"        "px2"
## [8] "means"
```

Support Vector Machines in R: Non-linear SVM

Plotting the data:

```r
plot(x, col = y + 1)
```



Support Vector Machines in R: Non-linear SVM

Now make a data frame with the response y, and turn that into a factor. We will fit an SVM with radial kernel.

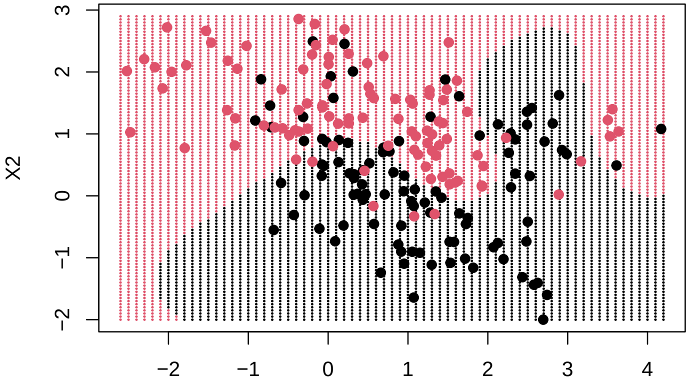
```
dat = data.frame(y = factor(y), x)
fit = svm(factor(y) ~ ., data = dat, scale = FALSE,
          kernel = "radial", cost = 5)
print(fit)
```

```
##
## Call:
## svm(formula = factor(y) ~ ., data = dat, kernel = "radial", cost = 5,
##     scale = FALSE)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  5
##
## Number of Support Vectors:  103
```

Support Vector Machines in R: Non-linear SVM

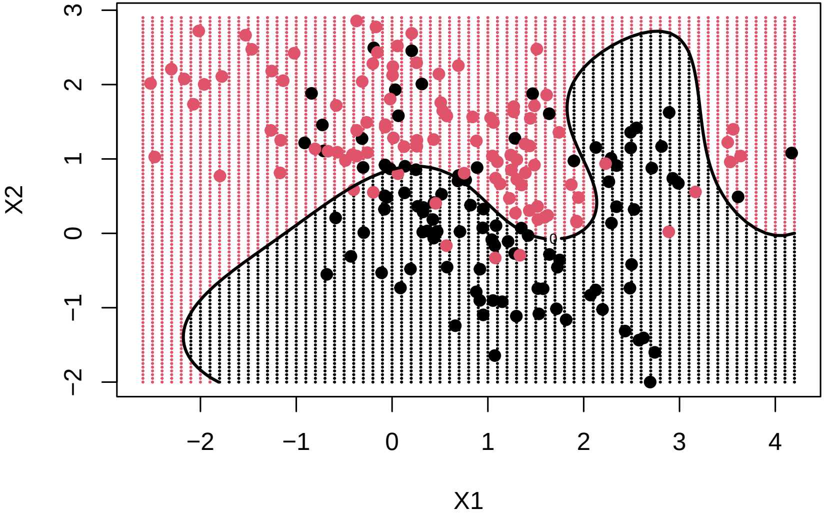
It's time to create a grid and predictions. We use `expand.grid` to create the grid, predict each of the values on the grid, and plot them:

```r
xgrid = expand.grid(X1 = px1, X2 = px2)
ygrid = predict(fit, xgrid)
plot(xgrid, col = as.numeric(ygrid), pch = 20, cex = .2)
points(x, col = y + 1, pch = 19)
```



Support Vector Machines in R: Non-linear SVM

Plotting with a contour:



Advantages and Disadvantages of SVMs

Advantages:

- **High Dimensionality:** SVM is an effective tool in high-dimensional spaces, which is particularly applicable to document classification and sentiment analysis where the dimensionality can be extremely large.
- **Memory Efficiency:** Since only a subset of the training points are used in the actual decision process of assigning new members, just these points need to be stored in memory (and calculated upon) when making decisions.
- **Versatility:** Class separation is often highly non-linear. The ability to apply new kernels allows substantial flexibility for the decision boundaries, leading to greater classification performance.

Advantages and Disadvantages of SVMs

Disadvantages:

- **Kernel Selection:** SVMs are very sensitive to the choice of the kernel parameters. In situations where the number of features for each object exceeds the number of training data samples, SVMs can perform poorly. This can be seen intuitively as if the feature space is much larger than the samples. Then there are less effective support vectors on which to support the optimal linear hyperplanes.
- **Non-Probabilistic:** Since the classifier works by placing objects above and below a classifying hyperplane, there is no direct probabilistic interpretation for group membership. However, one potential metric to determine the "effectiveness" of the classification is how far from the decision boundary the new point is.

Session Info

```r
sessionInfo()
```

```
## R version 4.3.1 (2023-06-16)
## Platform: aarch64-apple-darwin20 (64-bit)
## Running under: macOS Ventura 13.5.1
##
## Matrix products: default
## BLAS:   /Library/Frameworks/R.framework/Versions/4.3-arm64/Resources/lib/libRblas.0.dylib
## LAPACK: /Library/Frameworks/R.framework/Versions/4.3-arm64/Resources/lib/libRlapack.dylib;  LAPACK version 3
##
## locale:
## [1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
##
## time zone: America/Denver
## tzcode source: internal
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
##  [1] e1071_1.7-13    lubridate_1.9.3 forcats_1.0.0   stringr_1.5.0
##  [5] dplyr_1.1.3     purrr_1.0.2     readr_2.1.4     tidyr_1.3.0
##  [9] tibble_3.2.1    tidyverse_2.0.0 caret_6.0-94    lattice_0.22-5
## [13] ggplot2_3.4.4
##
## loaded via a namespace (and not attached):
##  [1] gtable_0.3.4         xfun_0.40            recipes_1.0.8
##  [4] tzdb_0.4.0           vctrs_0.6.4          tools_4.3.1
##  [7] generics_0.1.3       stats4_4.3.1         parallel_4.3.1
## [10] proxy_0.4-27         fansi_1.0.5          pkgconfig_2.0.3
## [13] ModelMetrics_1.2.2.2 Matrix_1.6-1.1       data.table_1.14.8
## [16] lifecycle_1.0.3      compiler_4.3.1       munsell_0.5.0
```