

Stats 102B - Week 7

Miles Chen, PhD

Department of Statistics

Week 7 Monday / Friday

UCLA

Section 1

Classification - Support Vector Machines

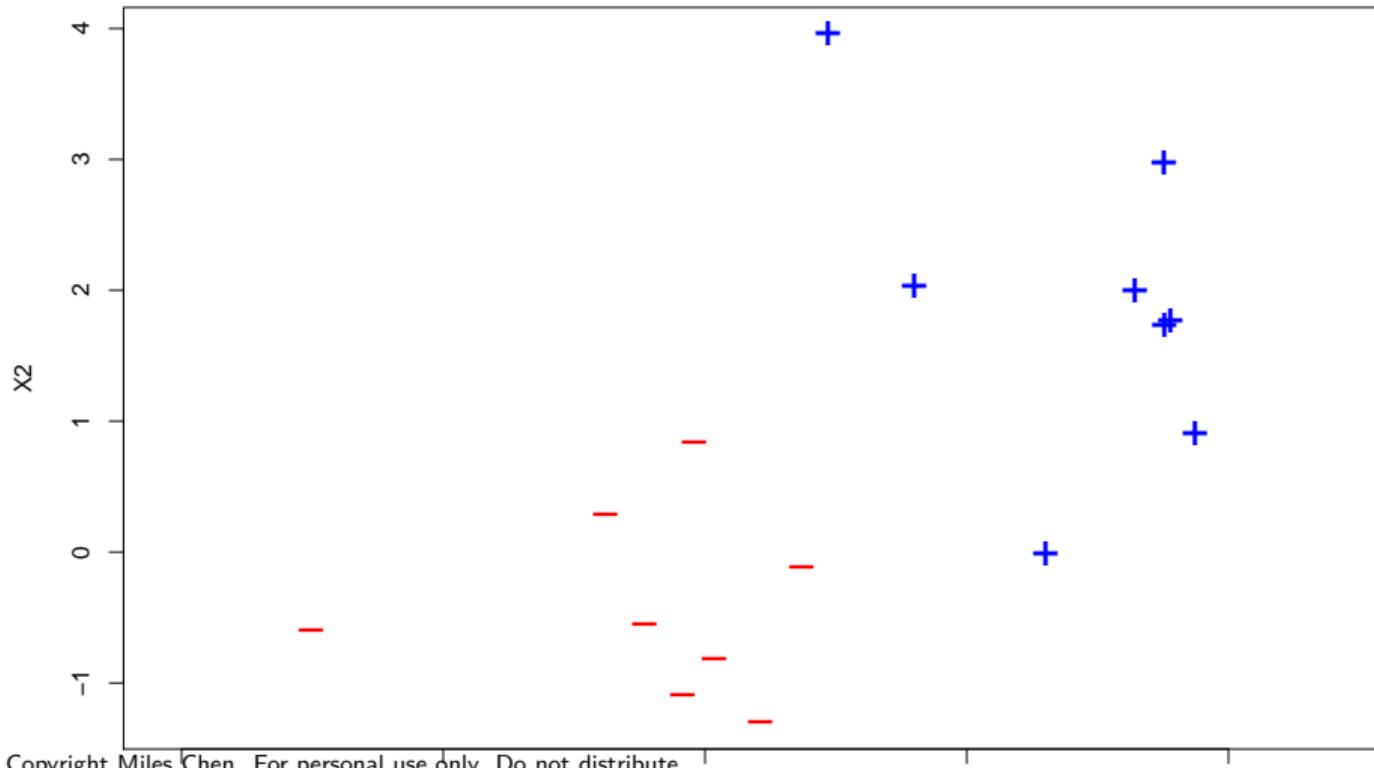
Support Vector Machines

Support Vector Machines is a classification method. The simplest version of the support vector machine is the **maximum-margin linear SVM**.

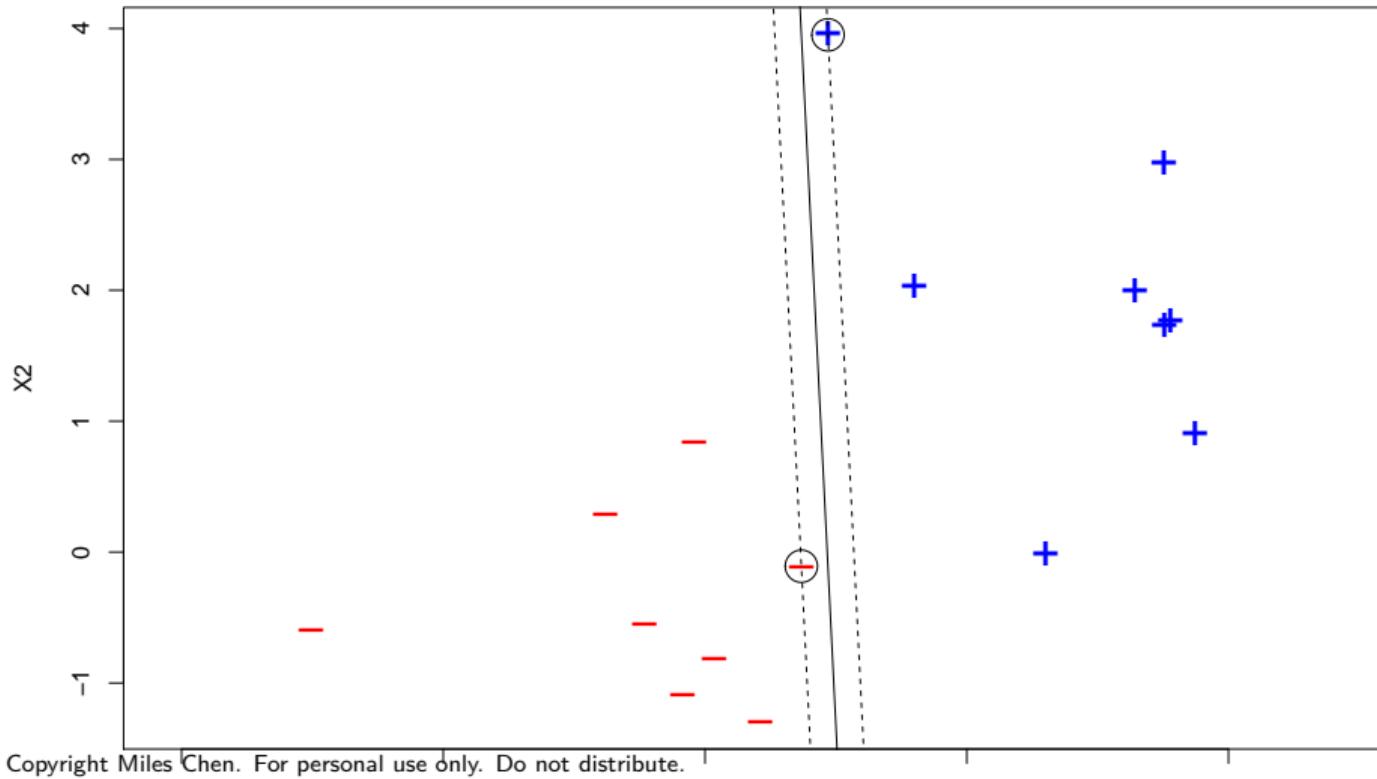
This only works for data that are linearly separable. The idea is to find two parallel hyperplanes (lines in 2D) that separate the two classes from each other and where the distance between these hyperplanes is as wide as possible.

On the next few slides, I'll show some points that are linearly separable. The slides will show several possible pairs of parallel hyperplanes (shown as dotted lines). The idea is to select the lines that have the maximum margin. The region between the dotted lines is the **margin** and the solid line in the middle is the **maximum-margin linear SVM boundary**.

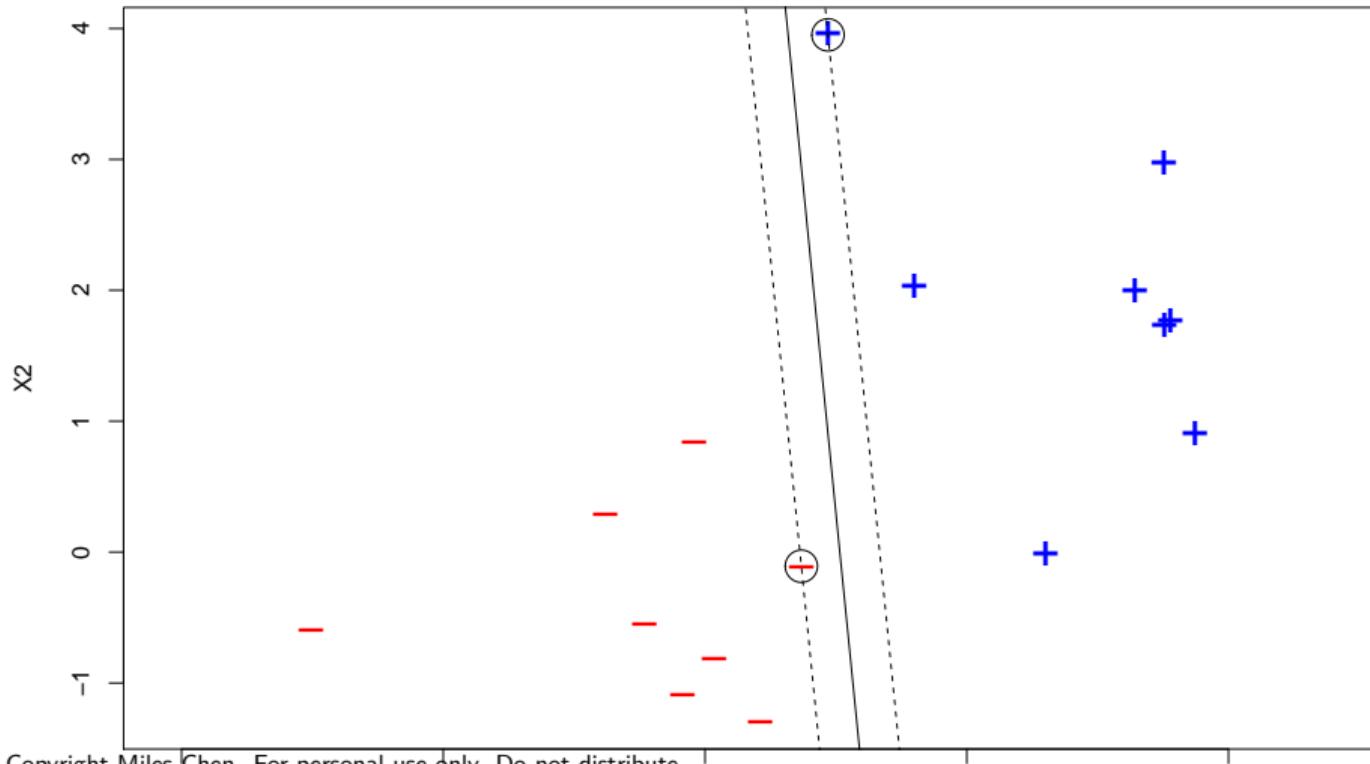
Support Vector Machines



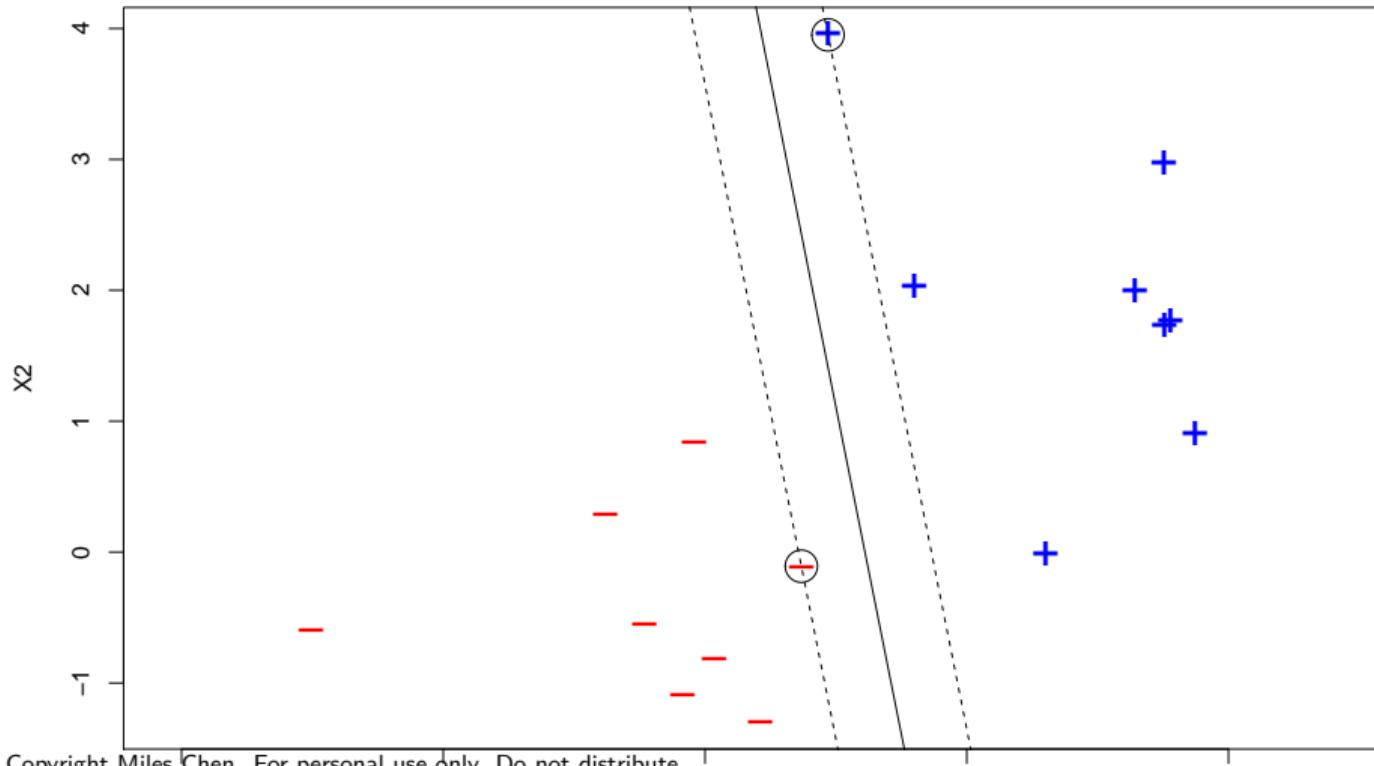
Selecting a boundary with 100% correct classification - narrow margin



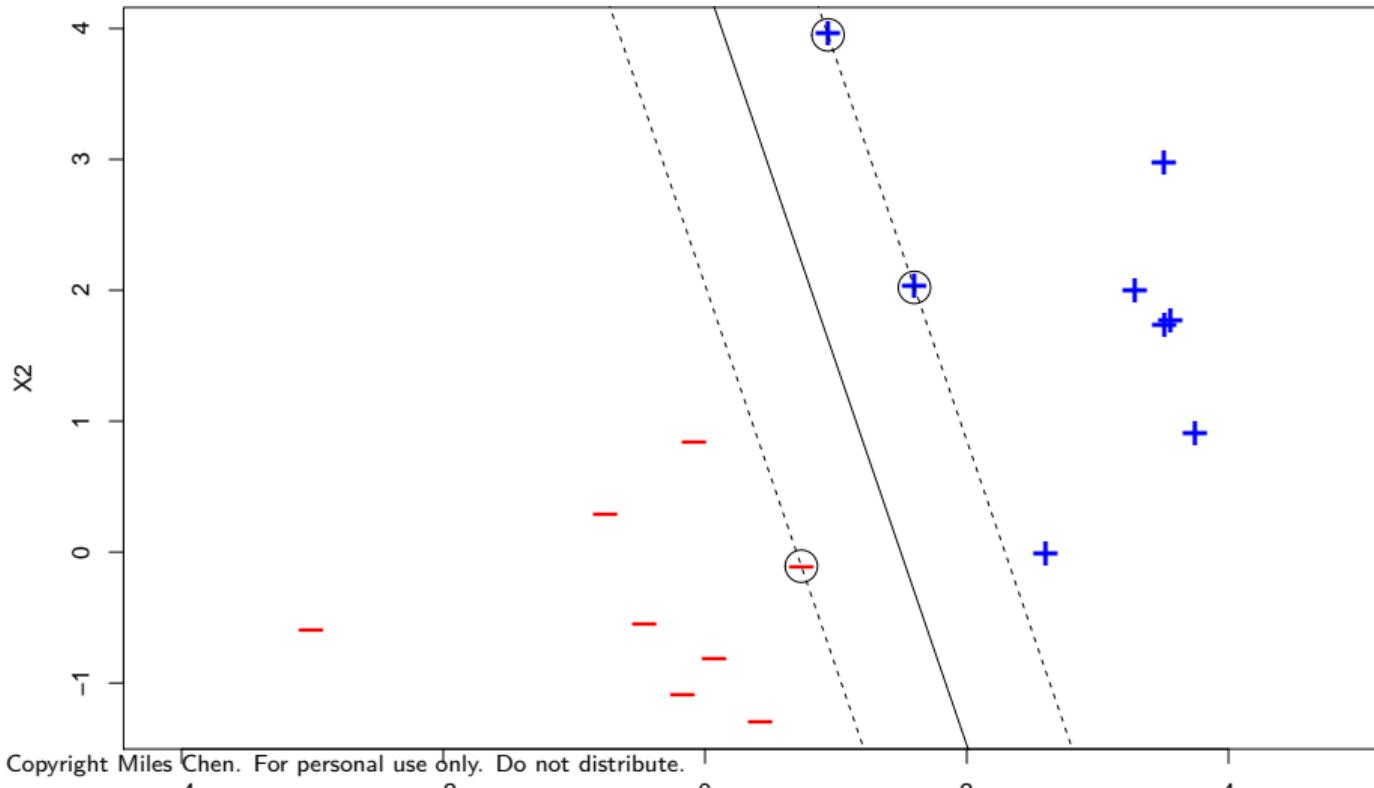
Parallel hyperplanes



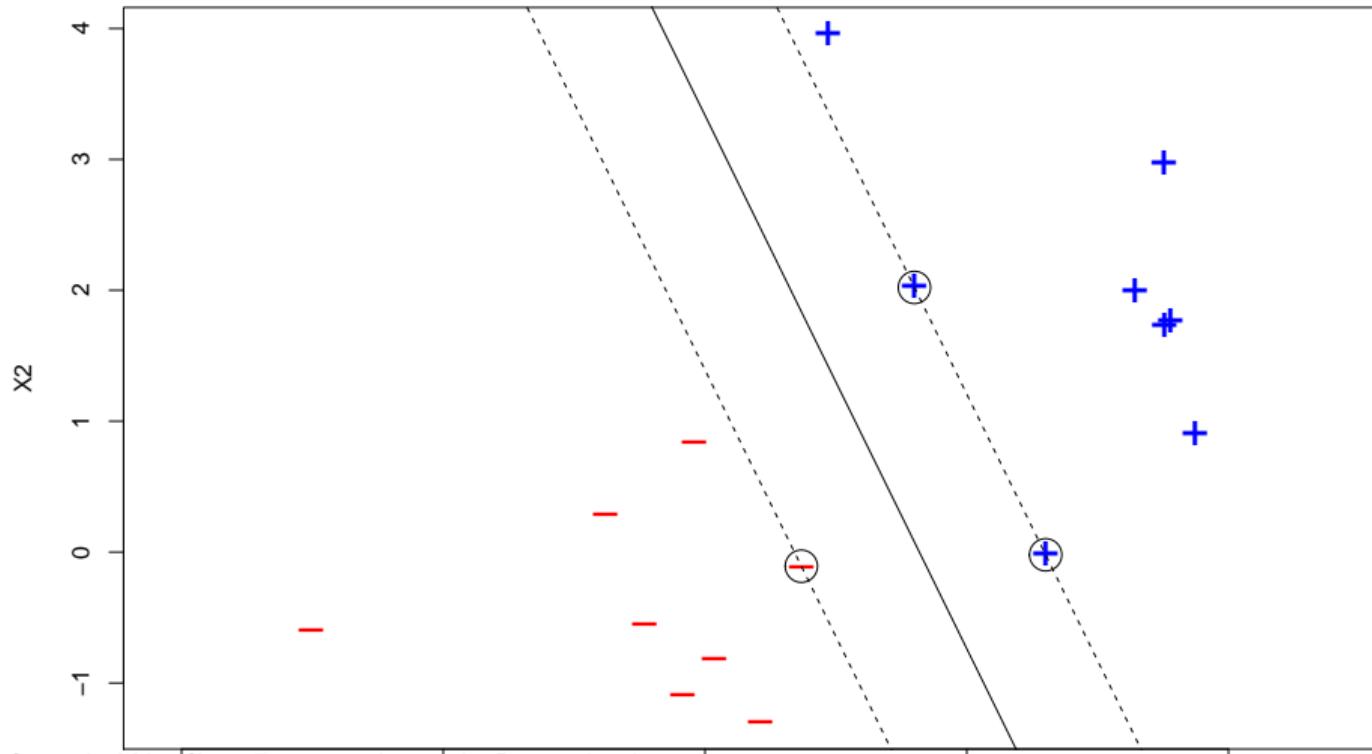
Parallel hyperplanes



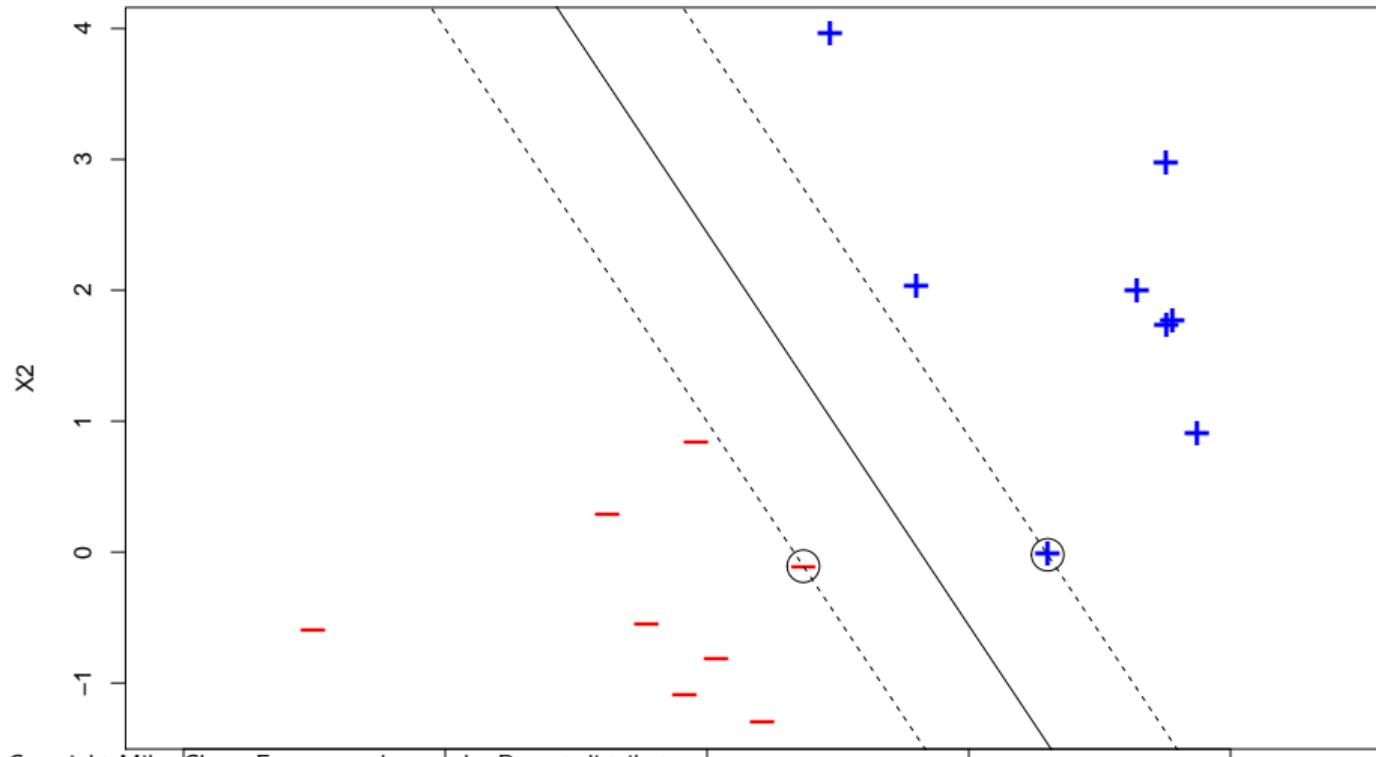
Parallel hyperplanes



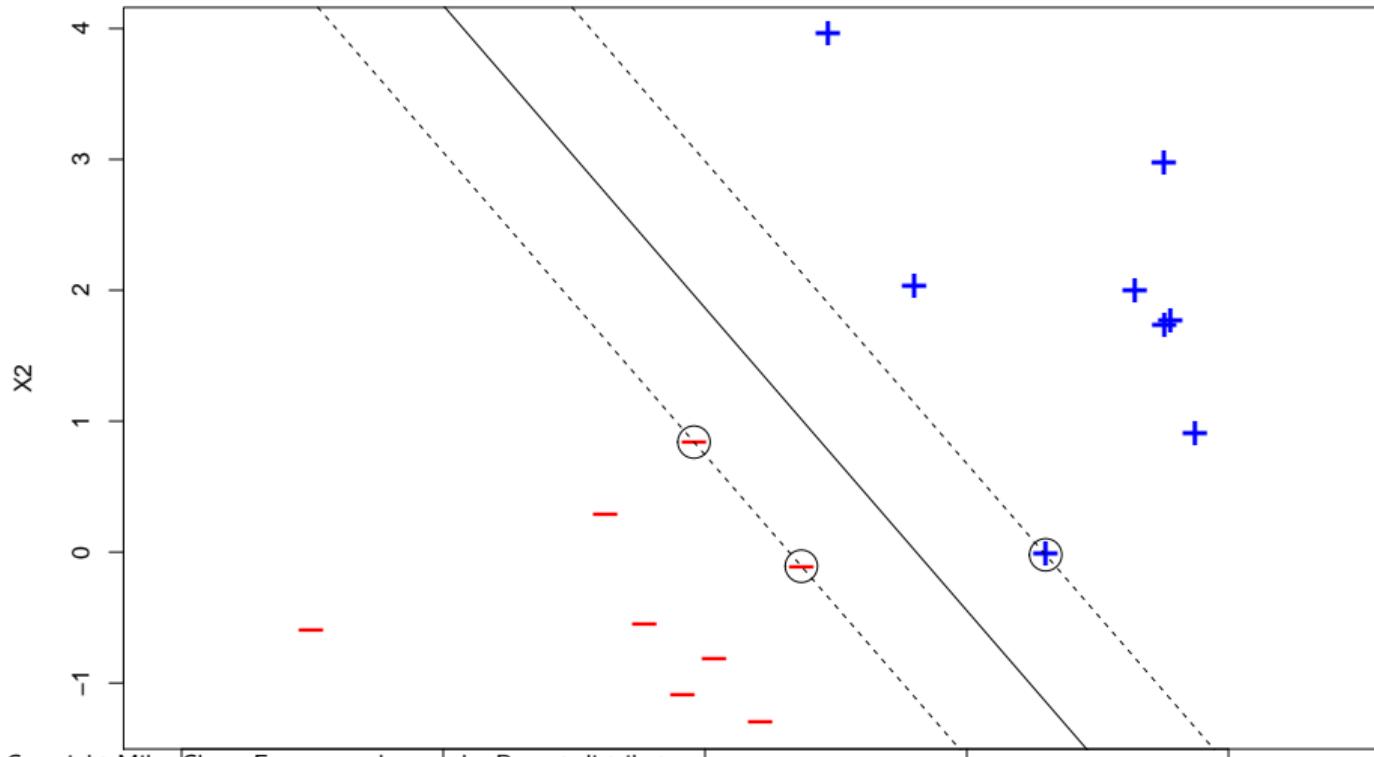
Parallel hyperplanes (widest margin)



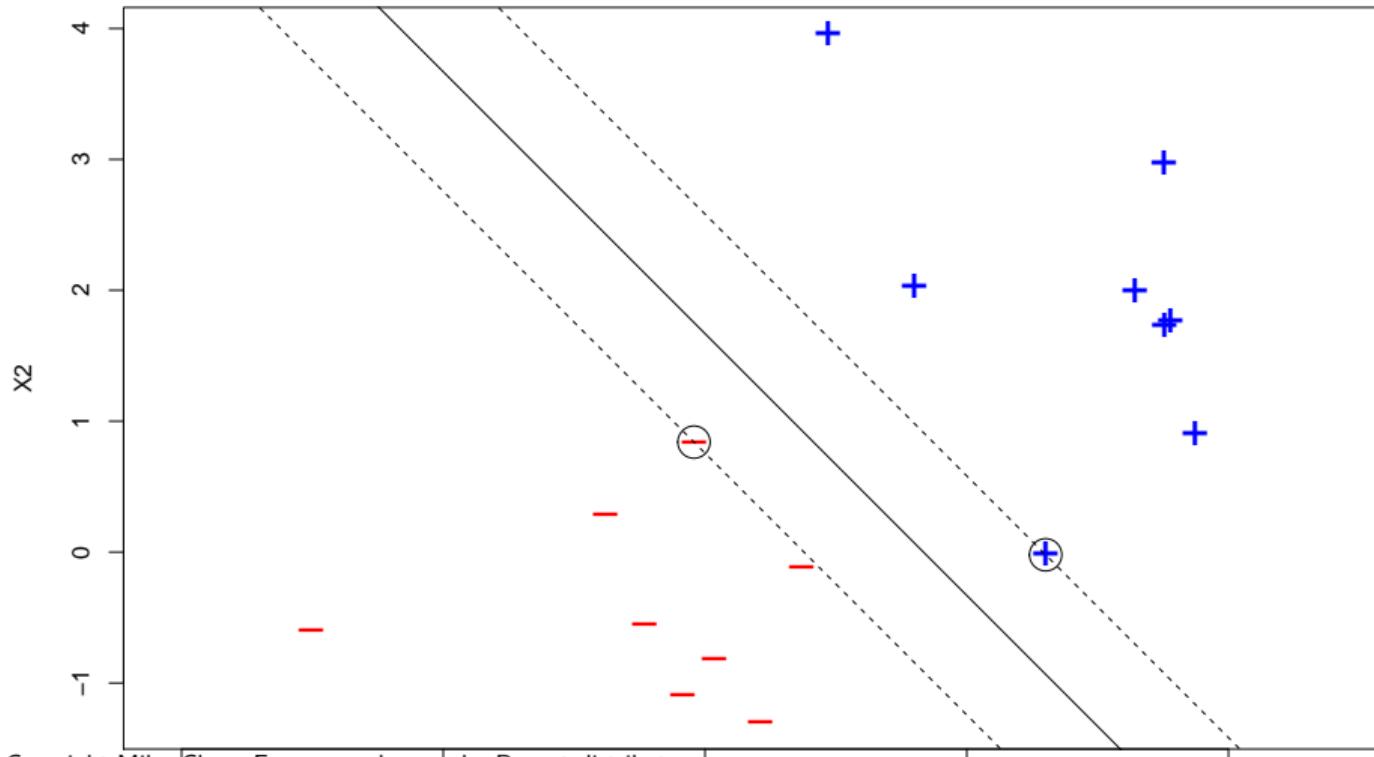
Parallel hyperplanes



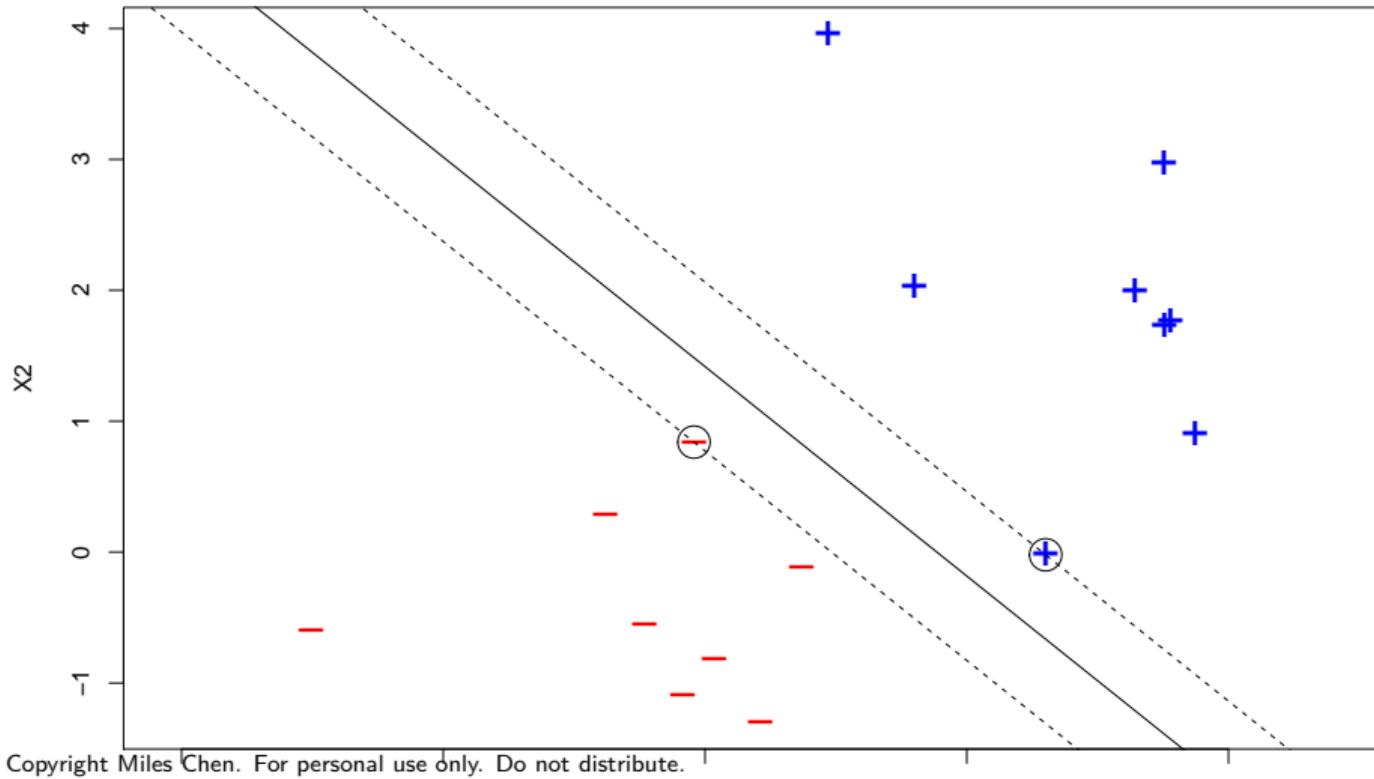
Parallel hyperplanes



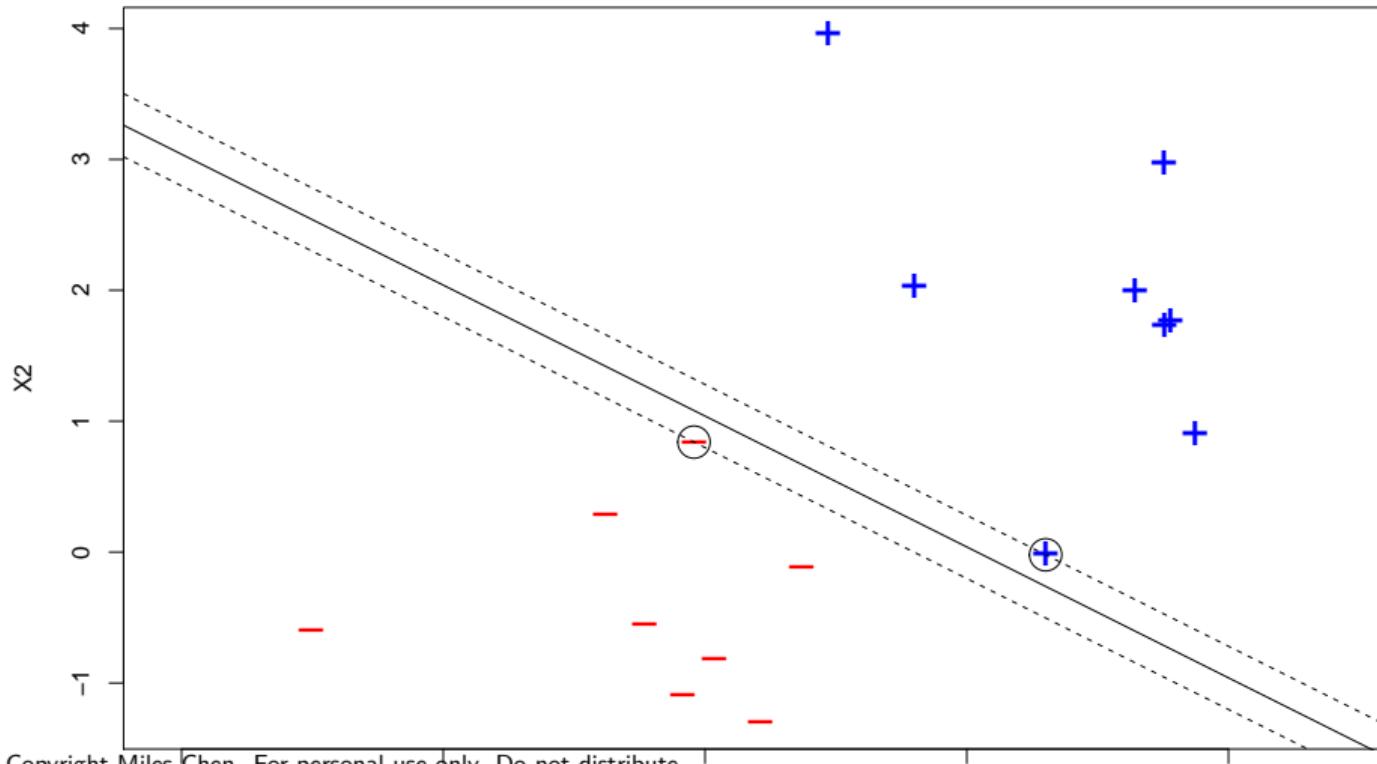
Parallel hyperplanes



Parallel hyperplanes



Parallel hyperplanes



So which hyperplane boundary do we pick?

SVM says the best boundary to pick is the one that maximizes the margin.

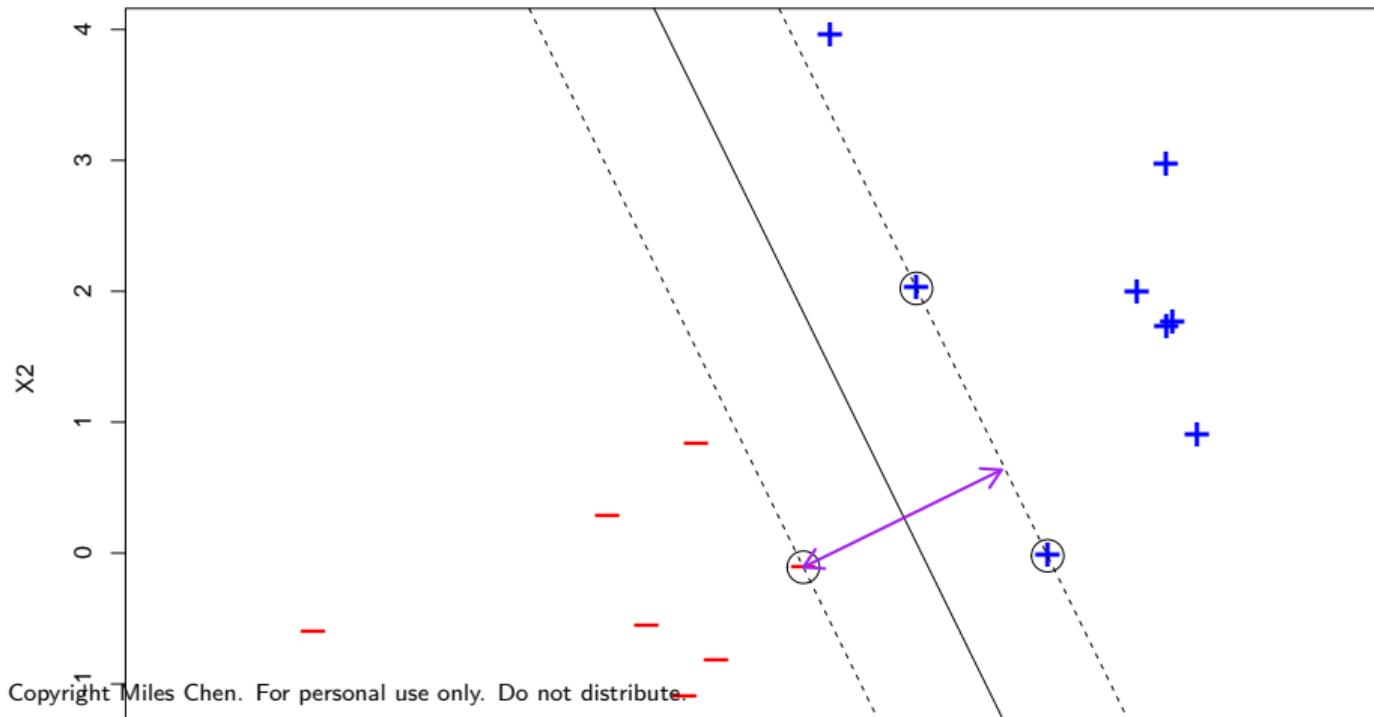
The margin is defined as the perpendicular distance from the decision boundary to the closest points on either side.

The closest points to the decision boundary depend on the decision boundary that is chosen.

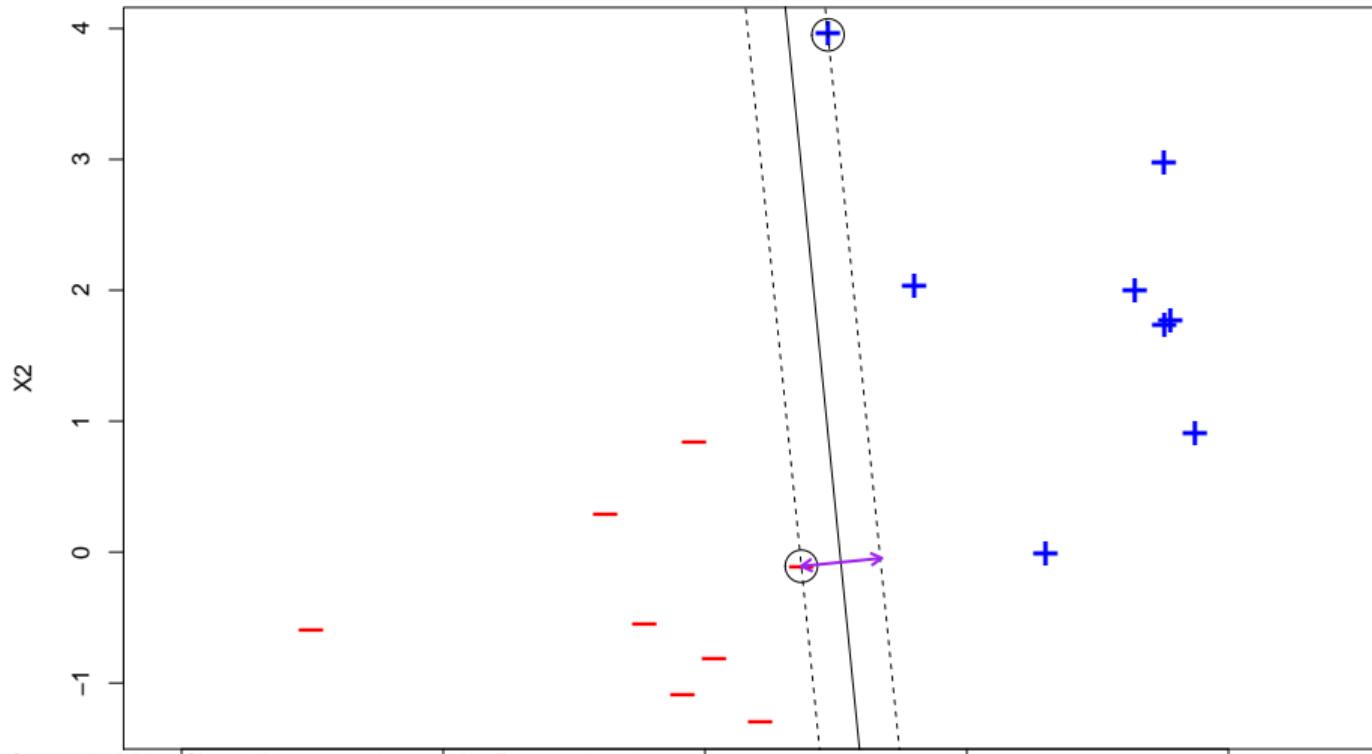
The **closest points are called the support vectors** and will lie on the margin's edges.

They are called support vectors because you can change the location of the other data points and the location of the boundary will be unaffected (as long as the support vectors remain the closest points to the boundary).

Maximum margin linear SVM - support vectors circled, and margin shown in purple



Non-optimal boundary has smaller margin (support vectors circled)



Maximizing the Margin Mathematically

(adapted from Wikipedia and Rogers & Girolami)

We are given a training dataset of n points of the form $(\mathbf{x}_1, t_1), \dots, (\mathbf{x}_n, t_n)$ where t_i are either 1 or -1 indicating the class. Each \mathbf{x}_i is a d-dimensional real vector.

We want to find the “maximum-margin SVM” that divides the group of points with $t_i = 1$ from those with $t_i = -1$ so that the distance from the hyperplane and the nearest point \mathbf{x}_i from either group is maximized.

Any hyperplane can be written as:

The set of points \mathbf{x} that satisfy:

$$\mathbf{w}^T \mathbf{x} - b = 0$$

Where \mathbf{w} is a vector normal (perpendicular) to the hyperplane.

Maximizing the Margin Mathematically

The parallel hyperplanes can be described by the equations

$$\mathbf{w}^T \mathbf{x} - b = 1 \text{ for anything on the “positive” side of the margin}$$

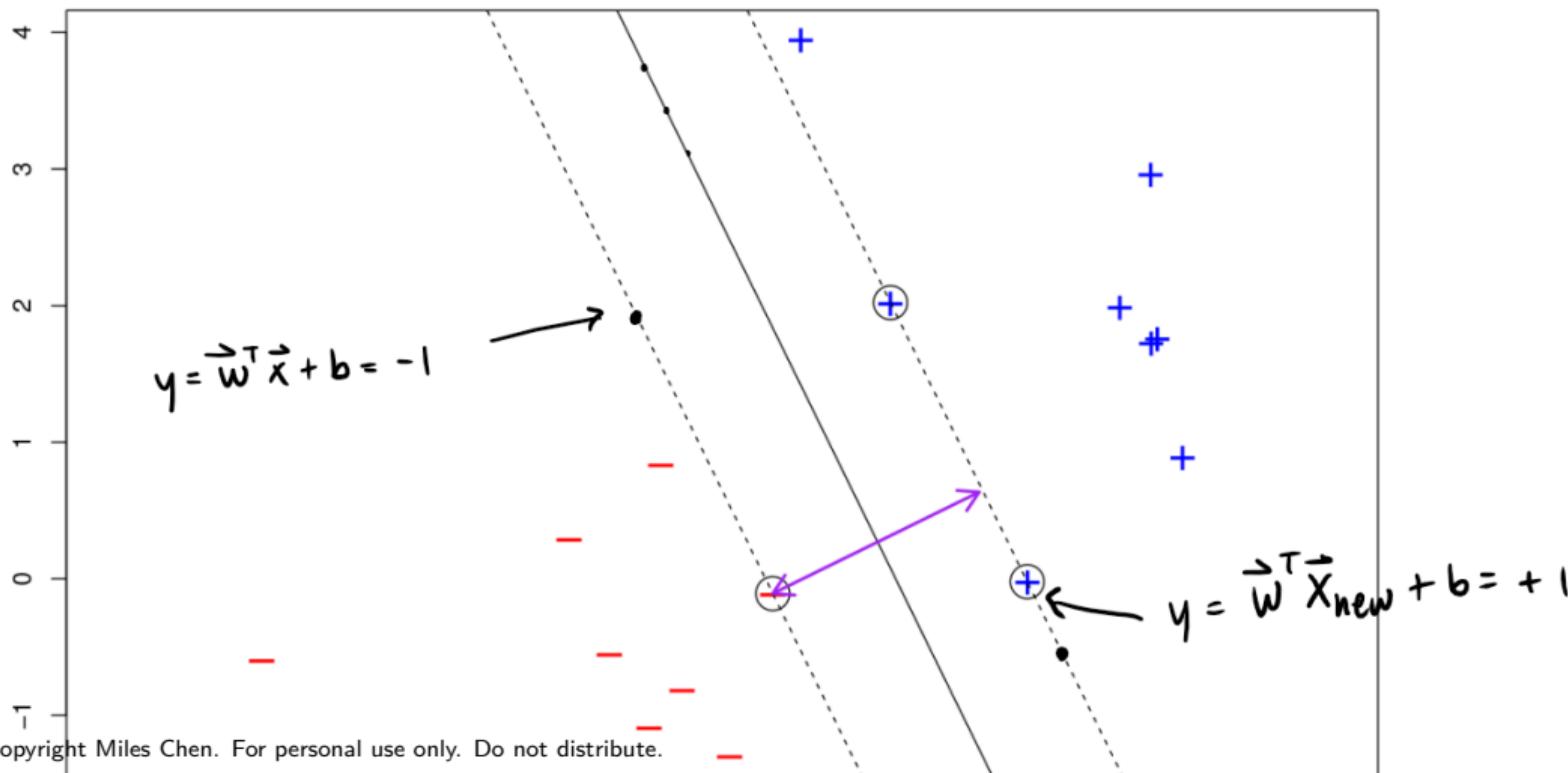
$$\mathbf{w}^T \mathbf{x} - b = -1 \text{ for anything on the “negative” side of the margin}$$

In the next couple pictures, let \mathbf{x}_1 and \mathbf{x}_2 be the closest points from each class. (They lie on the edges of the margin.)

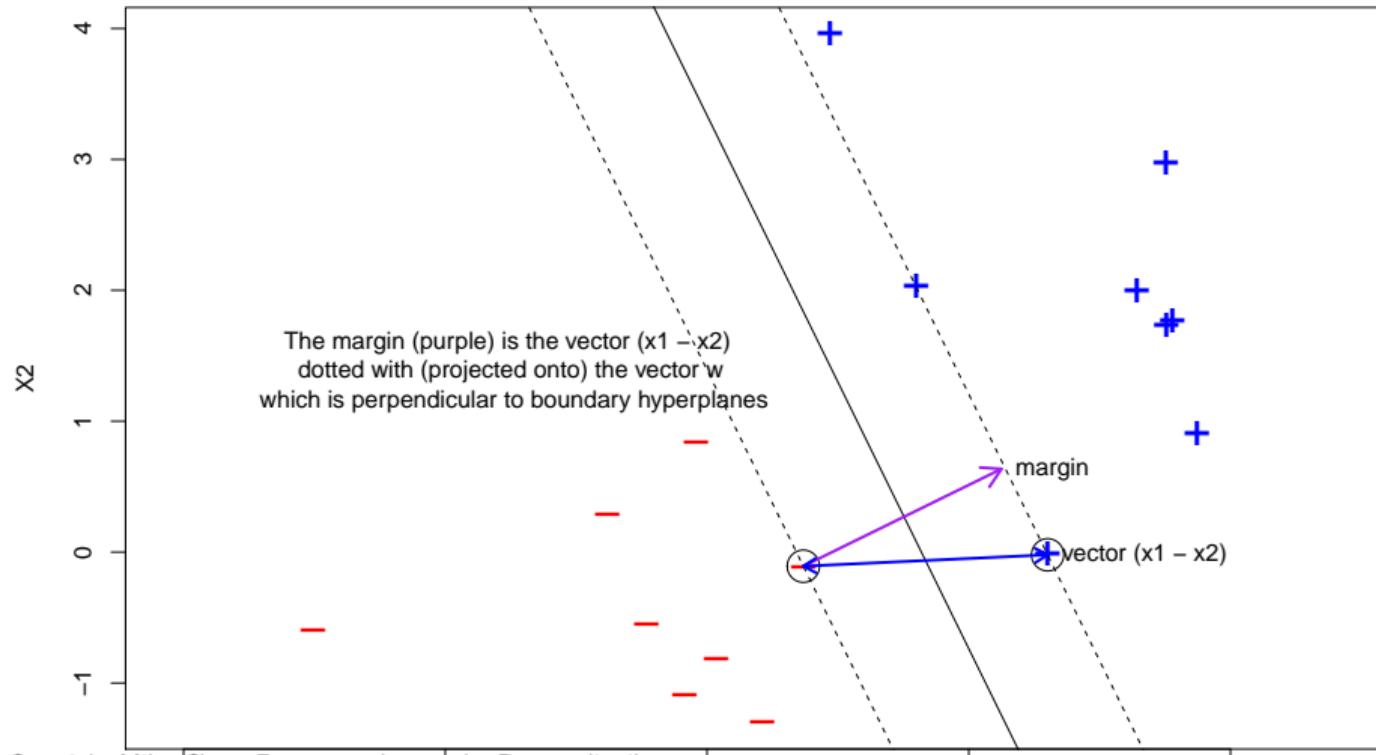
Maximizing the Margin Mathematically

$$y = \vec{w}^T \vec{x}_{\text{new}} + b = 0$$

if \vec{x}_{new} is on the boundary,
the result is 0. not classified as + or -.



Maximizing the Margin Mathematically



Maximizing the Margin Mathematically

Let \mathbf{x}_1 and \mathbf{x}_2 be the closest points from the two classes.

The vector $(\mathbf{x}_1 - \mathbf{x}_2)$ joins the points \mathbf{x}_1 and \mathbf{x}_2 .

With the way the boundary is defined, \mathbf{w} is a vector perpendicular to the boundary. $\frac{\mathbf{w}}{\|\mathbf{w}\|}$ is the direction perpendicular to the boundary.

2γ is equal to the length of the margin.

If we dot the vector $(\mathbf{x}_1 - \mathbf{x}_2)$ with the direction perpendicular to the margin, we will get the length of the margin.

$$2\gamma = \frac{1}{\|\mathbf{w}\|} \mathbf{w}^T (\mathbf{x}_1 - \mathbf{x}_2)$$

Maximizing the Margin Mathematically

We let $\mathbf{w}^T \mathbf{x}_1 - b = 1$ and $\mathbf{w}^T \mathbf{x}_2 - b = -1$

$$2\gamma = \frac{1}{\|\mathbf{w}\|} \mathbf{w}^T (\mathbf{x}_1 - \mathbf{x}_2)$$

$$2\gamma = \frac{1}{\|\mathbf{w}\|} (\mathbf{w}^T \mathbf{x}_1 - \mathbf{w}^T \mathbf{x}_2)$$

$$2\gamma = \frac{1}{\|\mathbf{w}\|} ((\mathbf{w}^T \mathbf{x}_1 - b) - (\mathbf{w}^T \mathbf{x}_2 - b))$$

$$2\gamma = \frac{1}{\|\mathbf{w}\|} (1 + 1)$$

$$\gamma = \frac{1}{\|\mathbf{w}\|}$$

Maximizing the Margin Mathematically

Because $\|\mathbf{w}\|$ is in the denominator, maximizing the distance γ now becomes equivalent to minimizing $\|\mathbf{w}\|$.

We wish to minimize $\|\mathbf{w}\|$, but this is subject to the constraint that

$$\mathbf{w}^T \mathbf{x}_i - b \geq 1 \text{ for all } \mathbf{x}_i \text{ with } t_i = 1$$

$$\mathbf{w}^T \mathbf{x}_i - b \leq -1 \text{ for all } \mathbf{x}_i \text{ with } t_i = -1$$

A clever way to summarize the constraint in one equation is:

$$t_i(\mathbf{w}^T \mathbf{x}_i - b) \geq 1$$

Maximizing the Margin Mathematically

We wish to minimize $\|\mathbf{w}\|$, which will share the same solution as minimizing $\frac{1}{2}\|\mathbf{w}\|^2$, subject to the constraint $t_i(\mathbf{w}^T \mathbf{x}_i - b) \geq 1$ for all i .

Optimizing a function subject to a constraint can be achieved using Lagrange multipliers. Lagrange multipliers add a new term to the objective function for each constraint so that the optimum of the new objective function corresponds to the optimum of the original constrained problem.

For our problem, we will have N Lagrangian terms, one for each observation (as every observation must satisfy the constraint that $t_i(\mathbf{w}^T \mathbf{x}_i - b) \geq 1$)

Maximizing the Margin Mathematically

The new objective function can be written as:

$$\operatorname{argmin}_{\mathbf{w}, \alpha} \frac{1}{2} \mathbf{w}^T \mathbf{w} - \sum_{i=1}^N \alpha_i (t_i (\mathbf{w}^T \mathbf{x}_i - b) - 1)$$

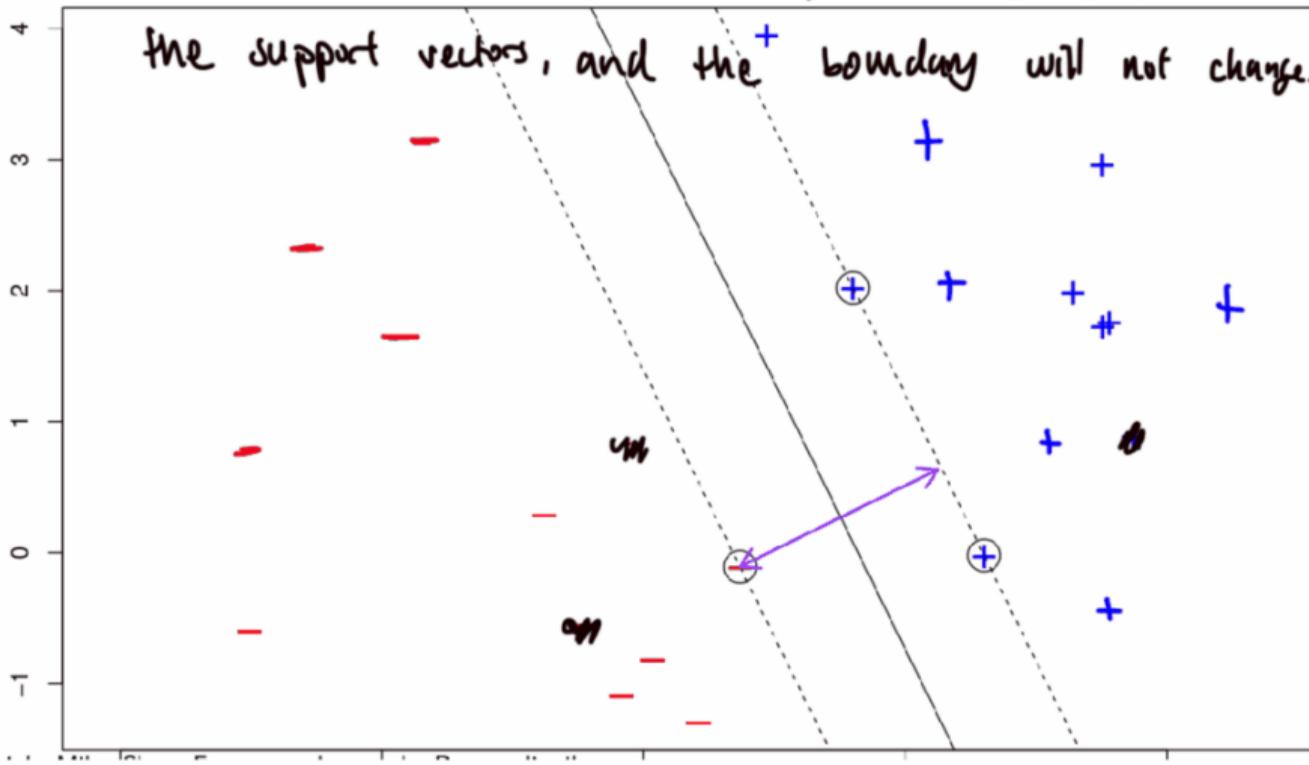
subject to $\alpha_i > 0$, for all i

Keep in mind $\|\mathbf{w}\|^2 = \mathbf{w}^T \mathbf{w}$

Unfortunately, there is no closed form solution to this Lagrange multiplier problem, but apparently it can be solved using quadratic programming. (I don't know anything about quadratic programming.)

We can add or remove points other than

the support vectors, and the boundary will not change.



Maximizing the margin

The function we wish to minimize is:

$$\frac{1}{2} \mathbf{w}^T \mathbf{w} - \sum_{n=1}^N \alpha_n (t_n (\mathbf{w}^T \mathbf{x}_n - b) - 1)$$

At the minimum the derivatives of this function with respect to \mathbf{w} and b will be equal to zero.

$$\frac{\partial}{\partial \mathbf{w}} = \mathbf{w} - \sum_{n=1}^N \alpha_n t_n \mathbf{x}_n = 0$$

$$\mathbf{w} = \sum_{n=1}^N \alpha_n t_n \mathbf{x}_n$$

$$\frac{\partial}{\partial b} = - \sum_{n=1}^N \alpha_n t_n = 0$$

Maximizing the margin

We can plug in the value of \mathbf{w} from the previous page ($\mathbf{w} = \sum \alpha_n t_n \mathbf{x}_n$) into the function we wish to optimize.

$$\begin{aligned}&= \frac{1}{2} \mathbf{w}^T \mathbf{w} - \sum_{n=1}^N \alpha_n (t_n (\mathbf{w}^T \mathbf{x}_n - b) - 1) \\&= \frac{1}{2} \left(\sum_{m=1}^N \alpha_m t_m \mathbf{x}_m^T \right) \left(\sum_{n=1}^N \alpha_n t_n \mathbf{x}_n \right) - \sum_{n=1}^N \alpha_n \left(t_n \left(\sum_{m=1}^N \alpha_m t_m \mathbf{x}_m^T \mathbf{x}_n - b \right) - 1 \right) \\&= \frac{1}{2} \sum_{m=1}^N \sum_{n=1}^N \alpha_m t_m \mathbf{x}_m^T \alpha_n t_n \mathbf{x}_n - \sum_{n=1}^N \sum_{m=1}^N \alpha_n t_n \alpha_m t_m \mathbf{x}_m^T \mathbf{x}_n + \sum_{n=1}^N \alpha_n t_n b + \sum_{n=1}^N \alpha_n \\&= \frac{1}{2} \sum_{m=1}^N \sum_{n=1}^N \alpha_m \alpha_n t_m t_n \mathbf{x}_m^T \mathbf{x}_n - \sum_{n=1}^N \sum_{m=1}^N \alpha_m \alpha_n t_m t_n \mathbf{x}_m^T \mathbf{x}_n + 0 \times b + \sum_{n=1}^N \alpha_n \\&= \sum_{n=1}^N \alpha_n - \frac{1}{2} \sum_{m=1}^N \sum_{n=1}^N \alpha_m \alpha_n t_m t_n \mathbf{x}_m^T \mathbf{x}_n\end{aligned}$$

Maximizing the margin

$$\sum_{n=1}^N \alpha_n - \frac{1}{2} \sum_{m=1}^N \sum_{n=1}^N \alpha_m \alpha_n t_m t_n \mathbf{x}_m^T \mathbf{x}_n$$

We need to find the optimum of the above, subject to:

$$\alpha_n \geq 0, \sum_{n=1}^N \alpha_n t_n = 0$$

Though there is no closed form solution, but the optimum can be found via quadratic programming.

It's worth noting that the optimization problem does not contain \mathbf{w} itself.

Maximizing the margin

At the solution, \mathbf{w} can be expressed as:

$$\mathbf{w} = \sum_{n=1}^N \alpha_n t_n \mathbf{x}_n$$

At the solution, the support vectors (i.e. the \mathbf{x} points that define the margin) will have α_n values that are not 0, while the non-support vectors have $\alpha_n = 0$. (The non-support vectors have no influence on the value of \mathbf{w})

We can think of α as a measure of how much influence a particular value in the training data has on defining \mathbf{w}

Identifying \mathbf{w} is a matter of finding the correct values of α_n .

Soft-Margins

The maximum-margin classifier uses the following constraint.

$$t_n(\mathbf{w}^T \mathbf{x}_n - b) \geq 1$$

The value 1 represents the distance from the boundary (center line) to the nearest point (dotted lines). The constraint is called a “hard margin” because all of the data points in the data set must satisfy the property that its distance is at least 1.

We can relax the constraint and allow for some points to be closer to the boundary (center line) than 1. If we say $\xi_n \geq 0$, we can have the following constraint instead:

$$t_n(\mathbf{w}^T \mathbf{x}_n - b) \geq 1 - \xi_n$$

If $0 < \xi_n \leq 1$, then the point i is within the margins.

If $1 < \xi_n$, then the point i is on the wrong side of the boundary.

Soft Margins

With soft-margins the optimization problem changes slightly. We still want to optimize:

$$\sum_{n=1}^N \alpha_n - \frac{1}{2} \sum_{m=1}^N \sum_{n=1}^N \alpha_m \alpha_n t_m t_n \mathbf{x}_m^T \mathbf{x}_n$$

But there is now an upper bound C on the what α can be in the constraint.

subject to $\sum_{n=1}^N \alpha_n t_n = 0, 0 \leq \alpha_n \leq C$, for all n

Effectively, the upper bound limits the amount of influence that a single point can have on defining the boundary.

Non-linear boundaries with Kernel functions

We can find linear boundaries by maximizing this function.

$$\sum_{n=1}^N \alpha_n - \frac{1}{2} \sum_{m=1}^N \sum_{n=1}^N \alpha_m \alpha_n t_m t_n \mathbf{x}_m^T \mathbf{x}_n$$

We can find non-linear boundaries by optimizing a very similar function that uses a kernel function.

$$\sum_{n=1}^N \alpha_n - \frac{1}{2} \sum_{m=1}^N \sum_{n=1}^N \alpha_m \alpha_n t_m t_n k(\mathbf{x}_m, \mathbf{x}_n)$$

The difference here is that we have replaced the dot product of two points $\mathbf{x}_m^T \mathbf{x}_n$ with a kernel function $k(\mathbf{x}_m, \mathbf{x}_n)$

What is a Kernel function?

A kernel function is an inner product (or dot product) of vectors after we have applied some kind of transformation function $\phi()$ to the original vector.

$$k(\mathbf{x}_m, \mathbf{x}_n) = \langle \varphi(\mathbf{x}_m), \varphi(\mathbf{x}_n) \rangle$$

Transformation Functions

The Kernel function returns the inner product after we apply a transformation ϕ .

For example, the function φ can project a two dimensional point into three dimensions.

$$\varphi(\mathbf{x}) = \varphi([x_1, x_2]) = [x_1, x_2, x_1^2 + x_2^2]$$

Example:

Let's say we have transformation function $\varphi(\mathbf{x}) = \varphi([x_1, x_2]) = [x_1, x_2, x_1^2 + x_2^2]$

and vectors: $\mathbf{x}_n = [1, 1]$ and $\mathbf{x}_m = [5, 1]$

When we apply the transformation function to our vectors, we get:

$$\varphi(\mathbf{x}_n) = [1, 1, 1^2 + 1^2] = [1, 1, 2]$$

$$\varphi(\mathbf{x}_m) = [5, 1, 5^2 + 1^2] = [5, 1, 26]$$

The dot product of these vectors is then:

$$k(\mathbf{x}_m, \mathbf{x}_n) = \langle \varphi(\mathbf{x}_m), \varphi(\mathbf{x}_n) \rangle = \langle [1, 1, 2], [5, 1, 26] \rangle = (1 \cdot 5 + 1 \cdot 1 + 2 \cdot 26) = 58$$

The Kernel function

It is not necessary to actually use φ to project the data into higher dimensions.

We can use the Kernel function directly to get the resulting inner product.

In this case, if $\varphi(\mathbf{x}) = \varphi([x_1, x_2]) = [x_1, x_2, x_1^2 + x_2^2]$, the kernel function is:

$$k(\mathbf{x}_m, \mathbf{x}_n) = \mathbf{x}_m^T \mathbf{x}_n + \|\mathbf{x}_m\|^2 \|\mathbf{x}_n\|^2$$

The Kernel function

$$\mathbf{x}_n = [1, 1], \mathbf{x}_m = [5, 1]$$

We can use φ to transform the vectors first and take the inner product:

$$k(\mathbf{x}_m, \mathbf{x}_n) = \langle \varphi(\mathbf{x}_m), \varphi(\mathbf{x}_n) \rangle = \langle [1, 1, 2], [5, 1, 26] \rangle = (1 \cdot 5 + 1 \cdot 1 + 2 \cdot 26) = 58$$

Or we can use the kernel function directly on the input vectors:

$$k(\mathbf{x}_m, \mathbf{x}_n) = \mathbf{x}_m^T \mathbf{x}_n + \|\mathbf{x}_m\|^2 \|\mathbf{x}_n\|^2 = \langle [1, 1], [5, 1] \rangle + (1^2 + 1^2)(5^2 + 1^2) = (1 \cdot 5 + 1 \cdot 1) + 2 \cdot 26 = 58$$

The Kernel ‘trick’ for SVM

When using a kernel function, it is equivalent to using the φ function to project the data into a higher dimensional space.

After projecting the data, we can use SVM to find a linear hyperplane that separates the points.

Animation:

<https://www.youtube.com/watch?v=3liCbRZPrZA>

Kernel Functions

Many possible kernel functions exist.

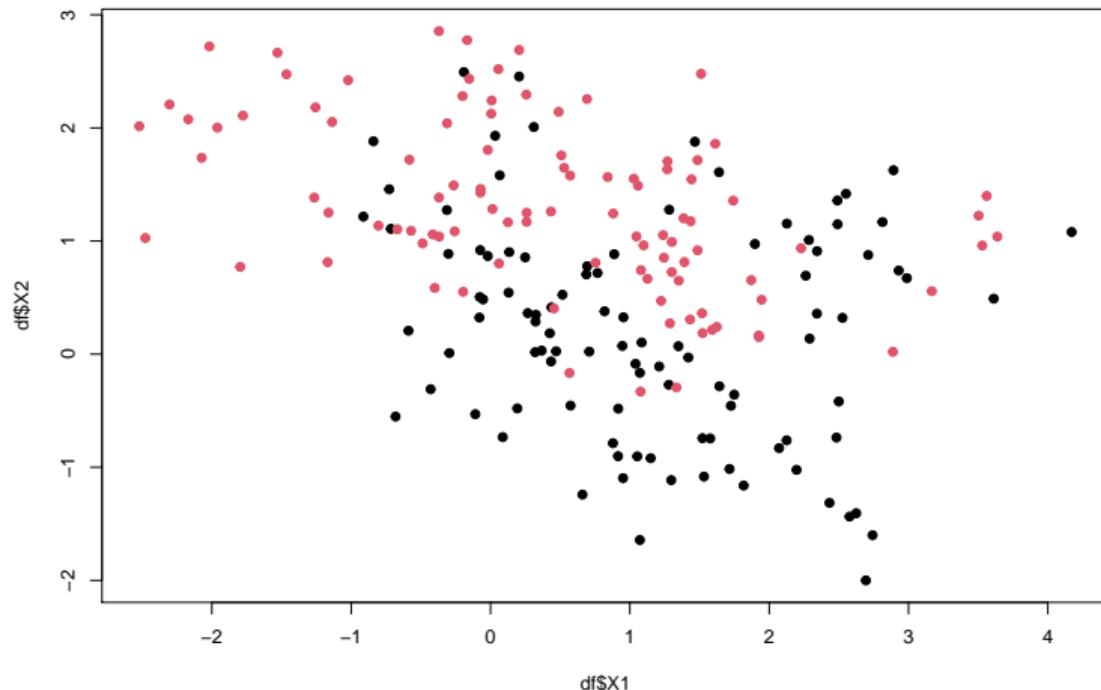
https://en.wikipedia.org/wiki/Positive-definite_kernel

Commonly used ones are the Gaussian and Polynomial Kernel

For some Kernel functions, the corresponding transformation function transforms the 2D vector into a very high dimensional vector. The Gaussian kernel actually transforms the vector into an infinite dimensional vector. (It's based on the Taylor expansion of the exp function.)

Again, it is not necessary to actually transform the original vectors. You can use the Kernel function directly.

Data to be classified (from Elements of Statistical Learning)



svm() in R

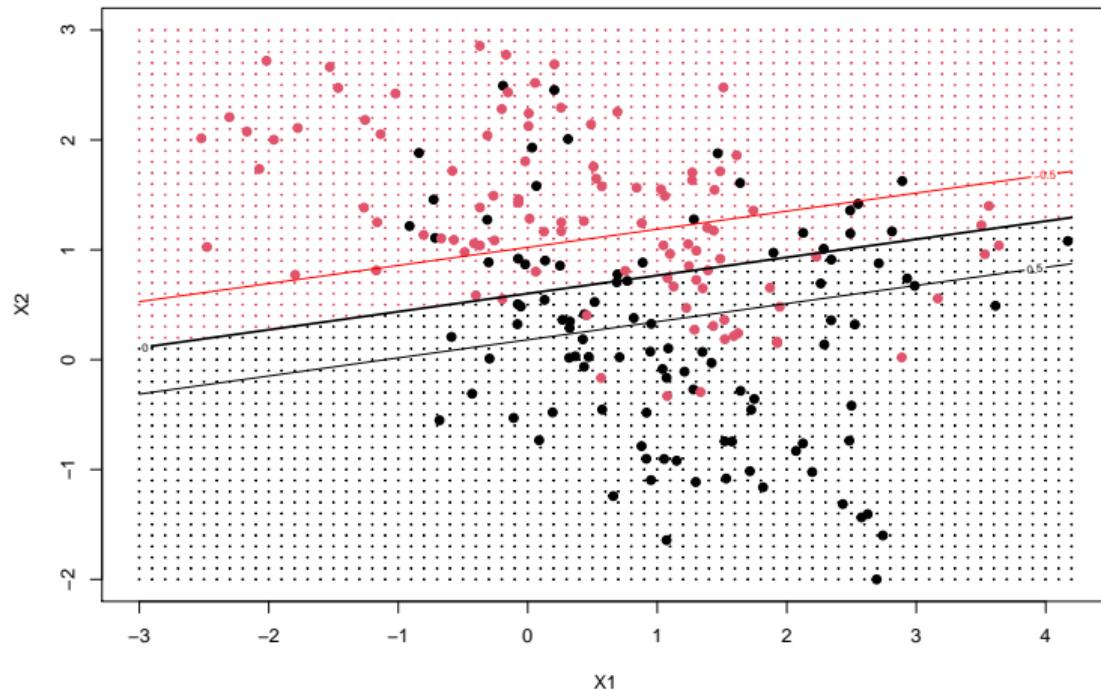
We will use the `svm()` function available in package `e1071`.

There are a few options you can use that will affect the SVM fit.

You can choose the kernel. Here we use a linear kernel.

```
library(e1071)
model <- svm(y ~ . , data = df, scale = FALSE, kernel = "linear")
```

Linear Kernel



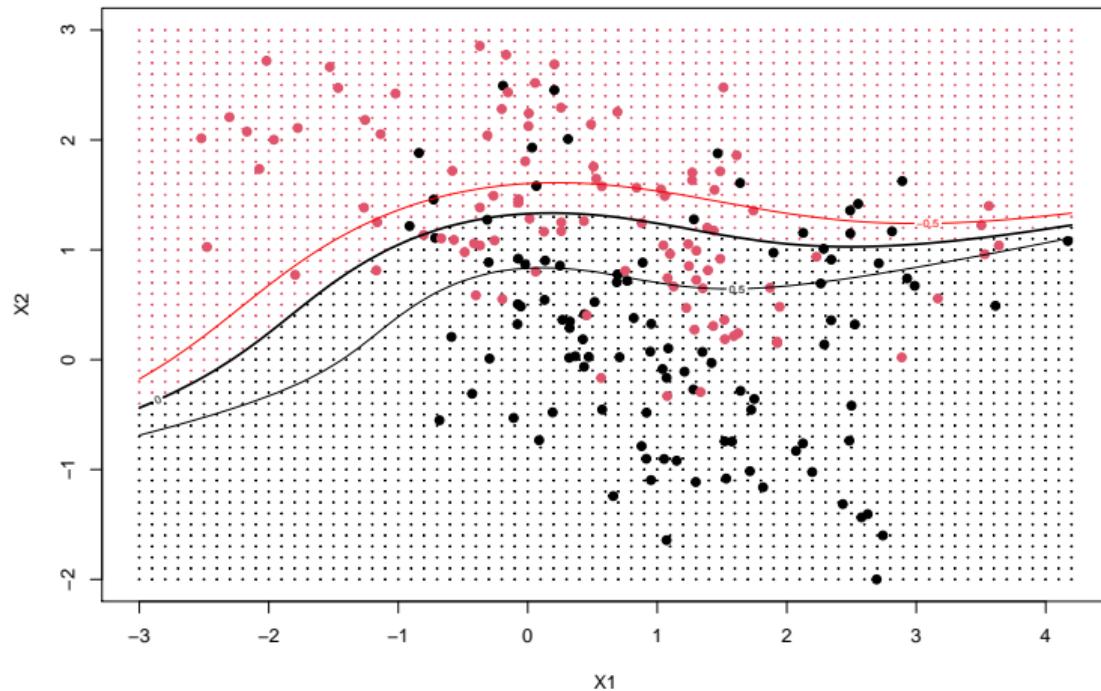
Polynomial Kernel

You can use a polynomial kernel. There will be another argument for the Polynomial degree. The default is 3.

You can also choose a gamma parameter.

```
model <- svm(y ~ . , data = df, scale = FALSE, kernel = "polynomial",
              degree = 3, gamma = 0.5)
```

Polynomial Kernel, degree 3

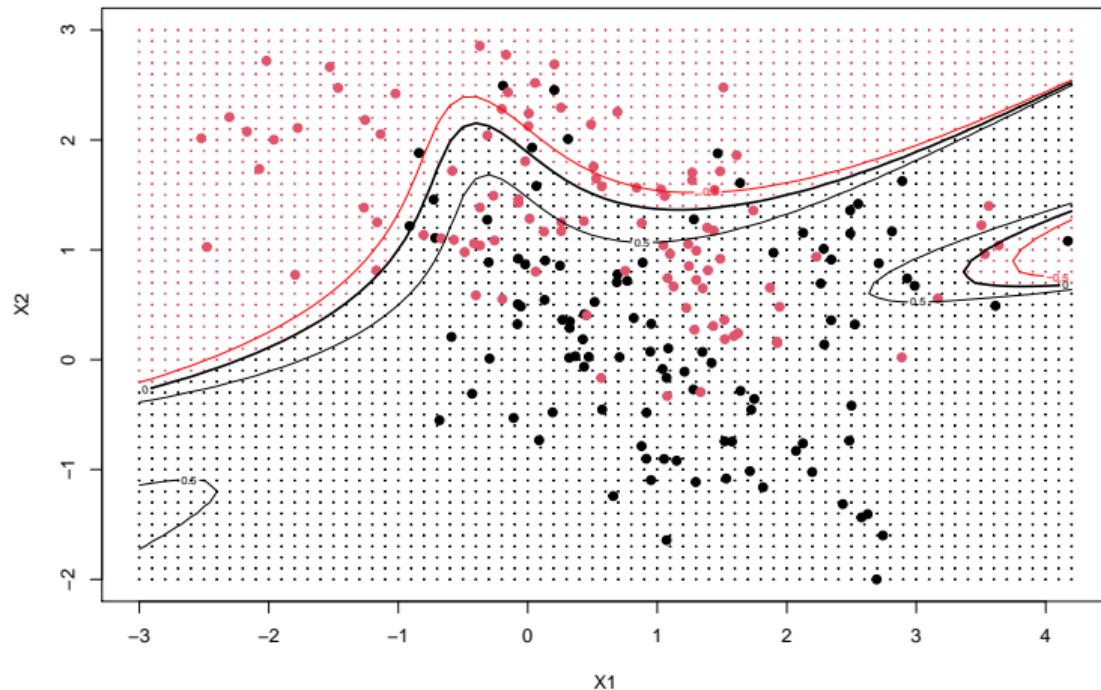


Polynomial Kernel

A degree 5 polynomial

```
model <- svm(y ~ . , data = df, scale = FALSE, kernel = "polynomial",
              degree = 5, gamma = 0.5)
```

Polynomial Kernel, degree 5



Gaussian Kernel

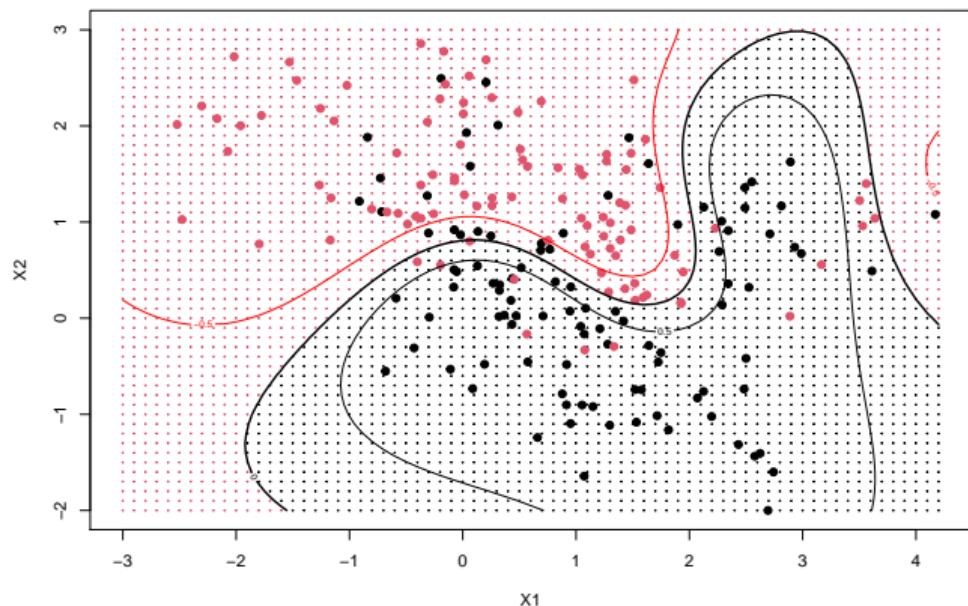
Radial Basis Functions (Gaussian kernels) are very popular for SVM because of the flexibility it offers.

When specifying the Radial basis kernel, you can select the parameters for gamma and cost.

For your homework assignment, I want you to study the effect of using different parameters.

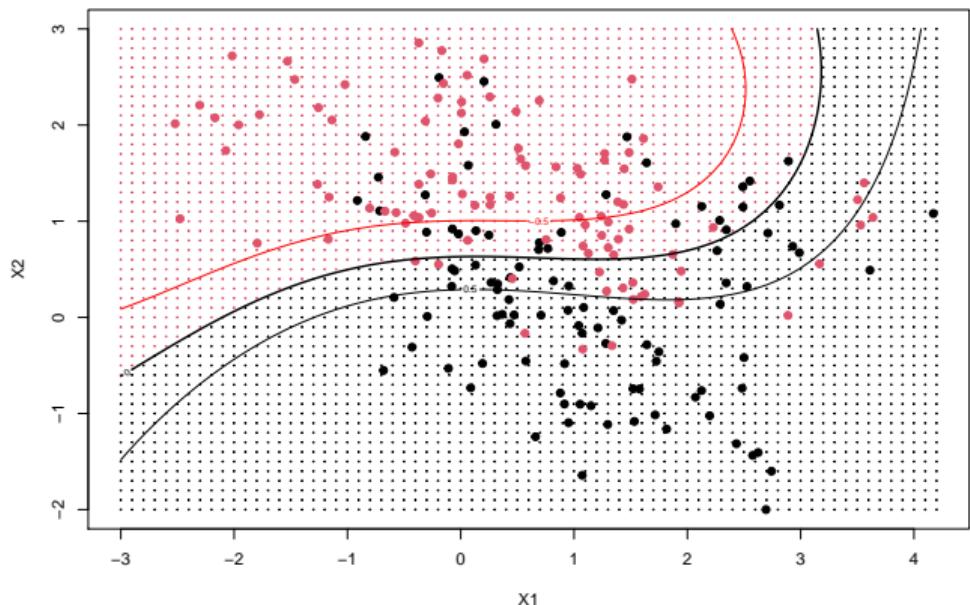
```
model <- svm(y ~ . , data = df, scale = FALSE, kernel = "radial",
              gamma = 0.5, cost = 1)
```

Gaussian Kernel



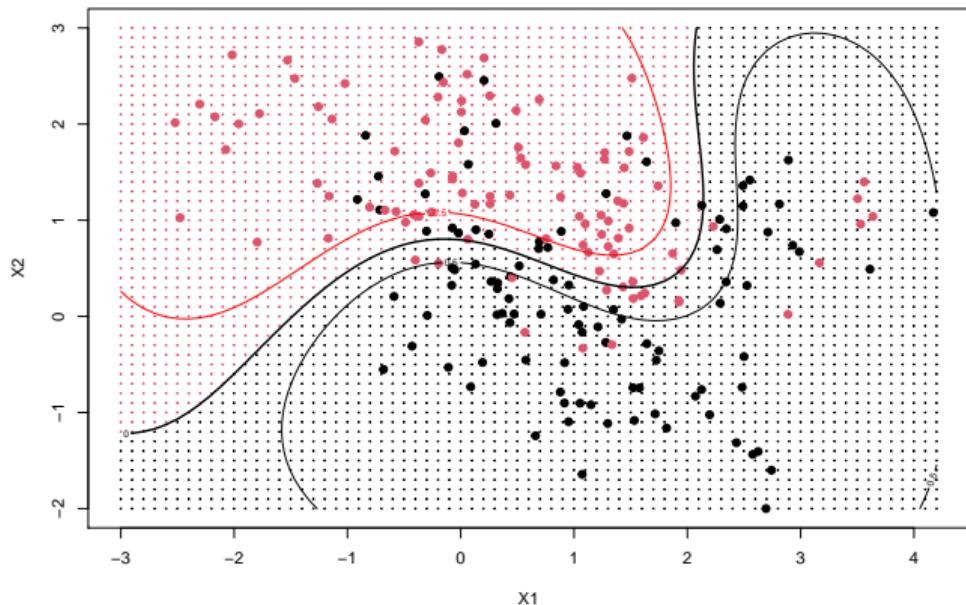
Gaussian Kernel - altering gamma, keep cost = 1

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



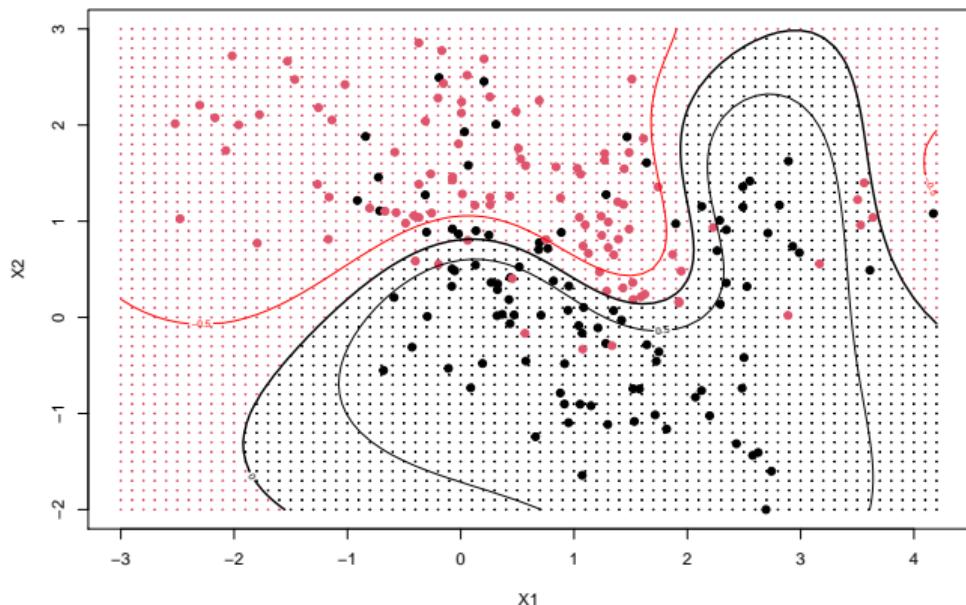
Gaussian Kernel - altering gamma, keep cost = 1

```
model <- svm(y ~ . , data = df, scale = FALSE, kernel = "radial", gamma = 0.3, cost = 1)
```



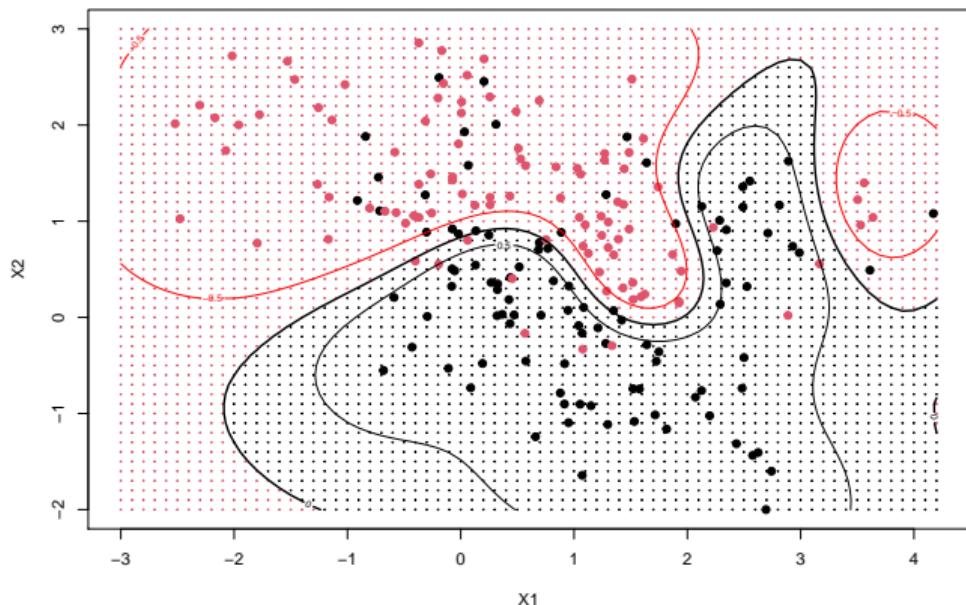
Gaussian Kernel - altering gamma, keep cost = 1

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



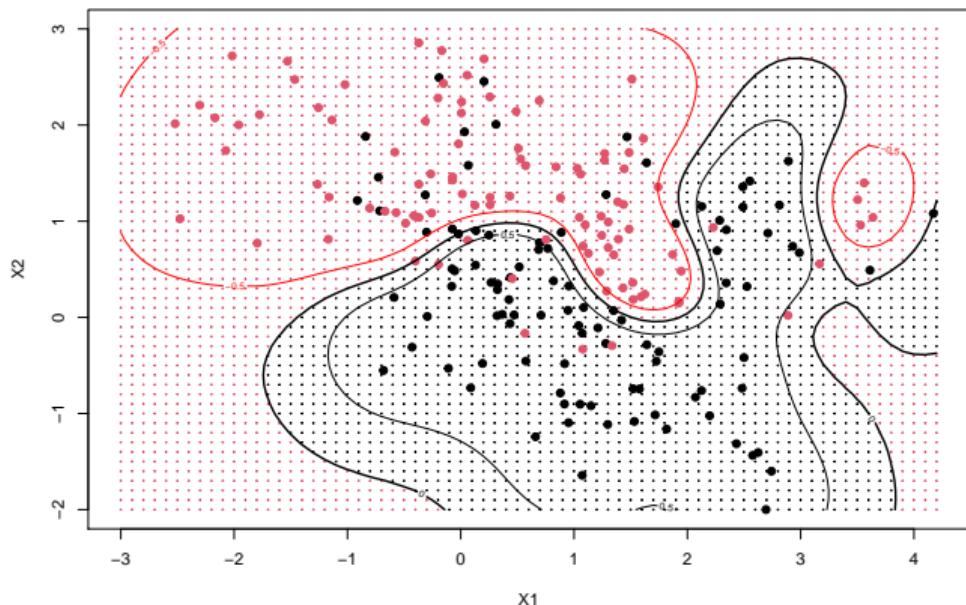
Gaussian Kernel - altering gamma, keep cost = 1

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



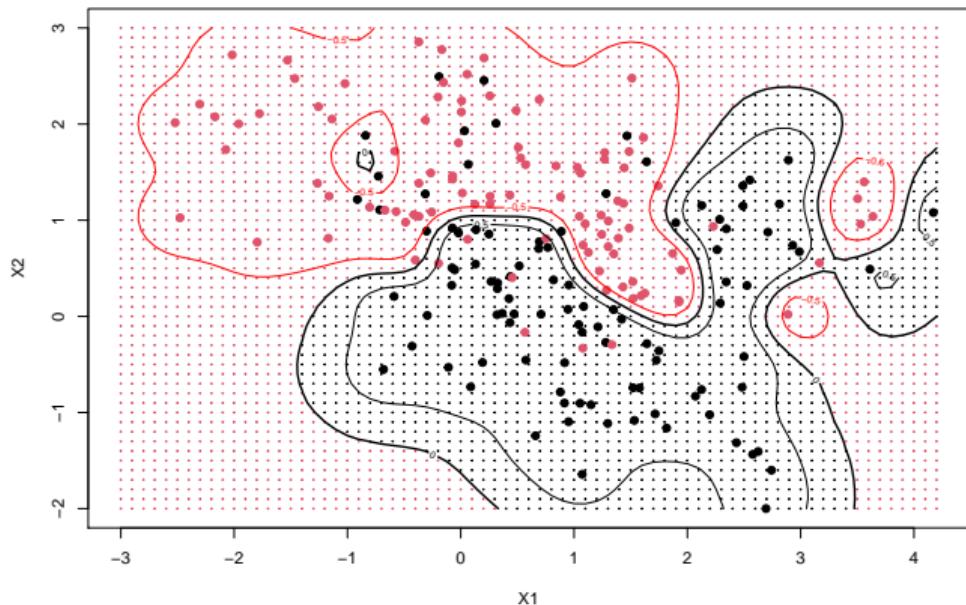
Gaussian Kernel - altering gamma, keep cost = 1

```
model <- svm(y ~ . , data = df, scale = FALSE, kernel = "radial", gamma = 2, cost = 1)
```



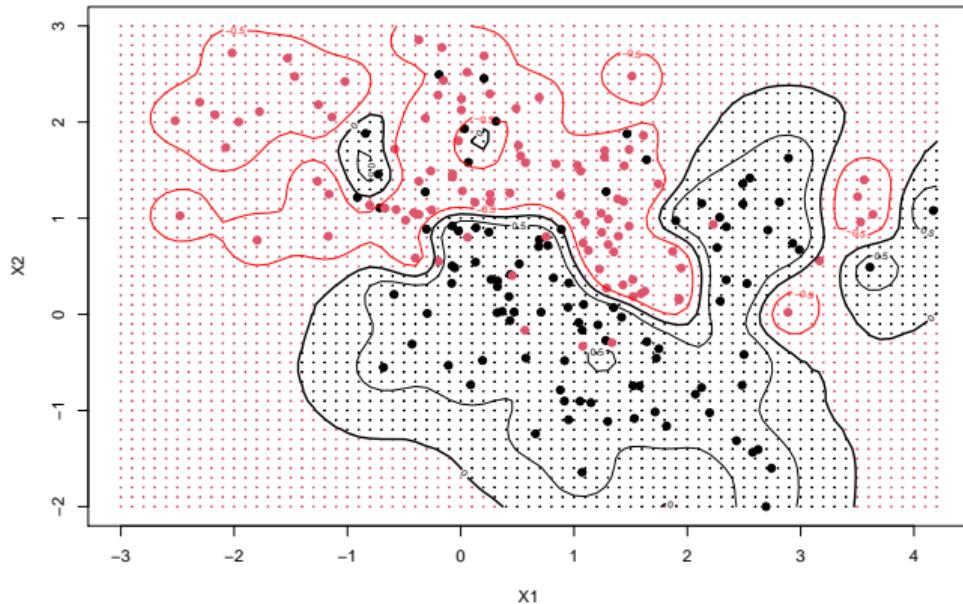
Gaussian Kernel - altering gamma, keep cost = 1

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



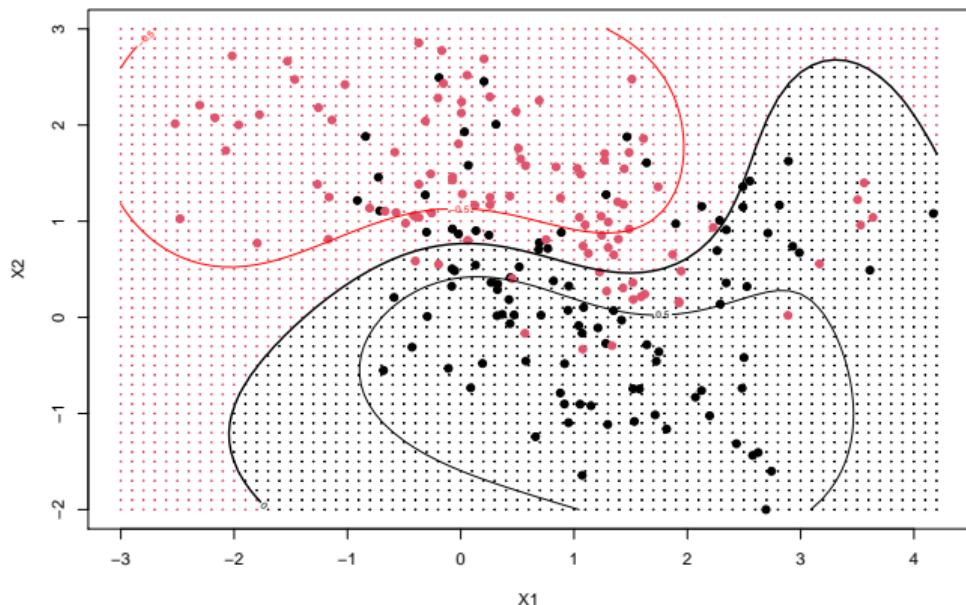
Gaussian Kernel - altering gamma, keep cost = 1

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



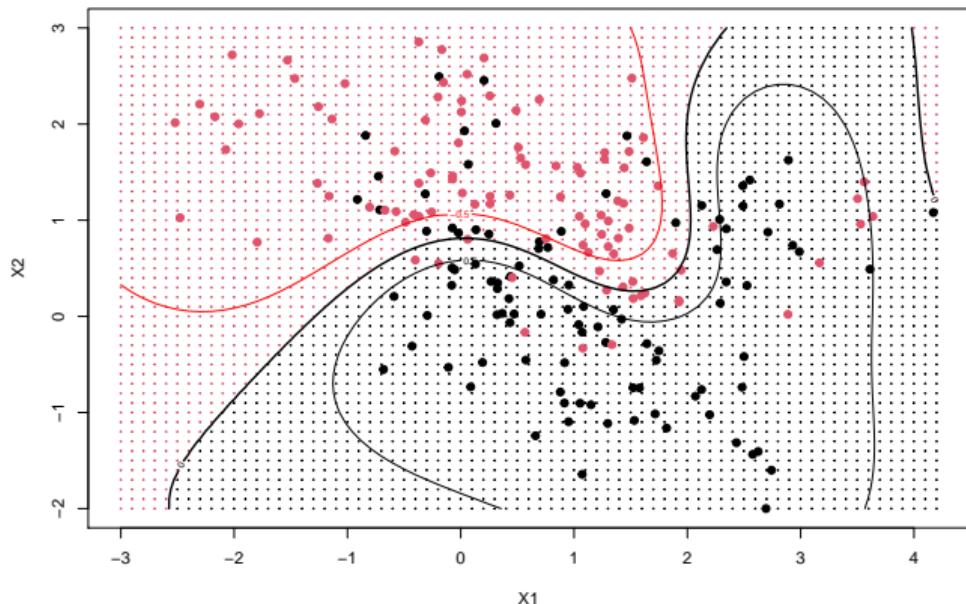
Gaussian Kernel - altering cost, keep gamma = 0.5

```
model <- svm(y ~ . , data = df, scale = FALSE, kernel = "radial", gamma = 0.5, cost = 0.1)
```



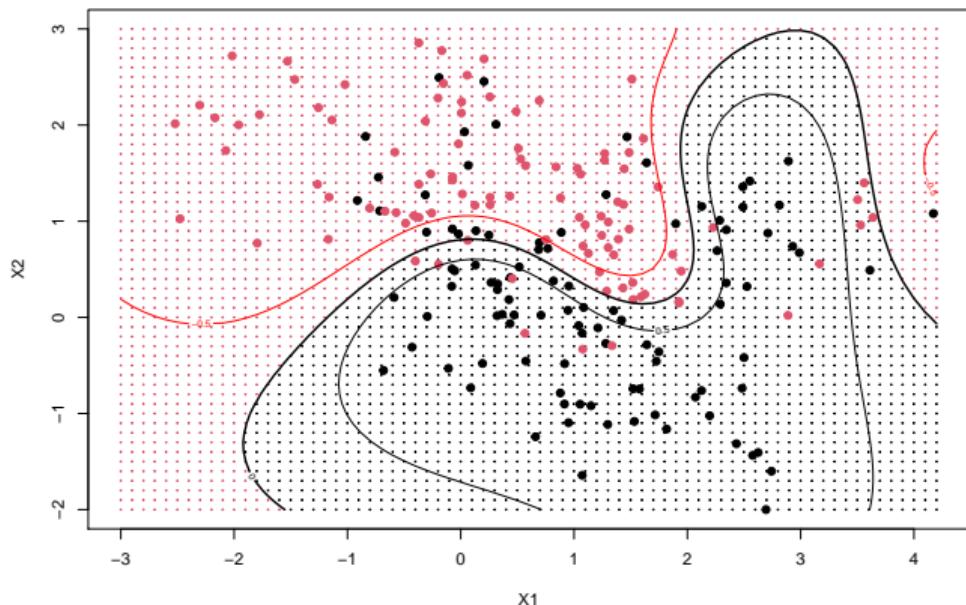
Gaussian Kernel - altering cost, keep gamma = 0.5

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



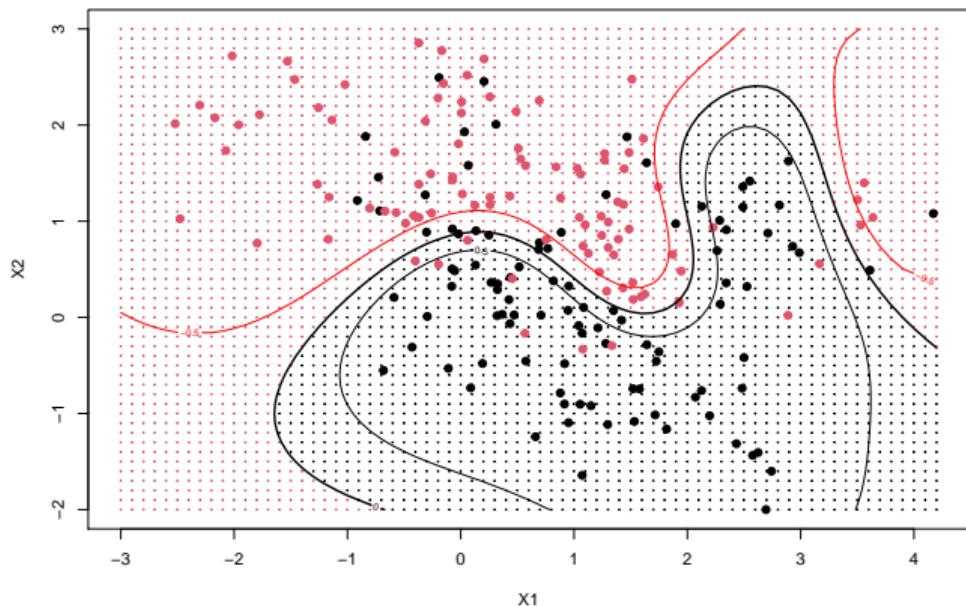
Gaussian Kernel - altering cost, keep gamma = 0.5

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



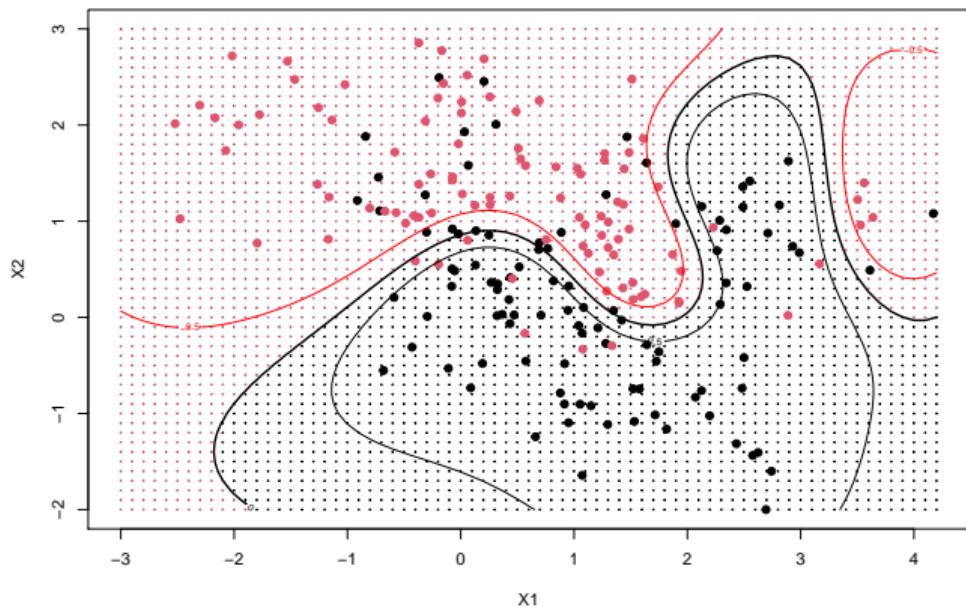
Gaussian Kernel - altering cost, keep gamma = 0.5

```
model <- svm(y ~ . , data = df, scale = FALSE, kernel = "radial", gamma = 0.5, cost = 2)
```



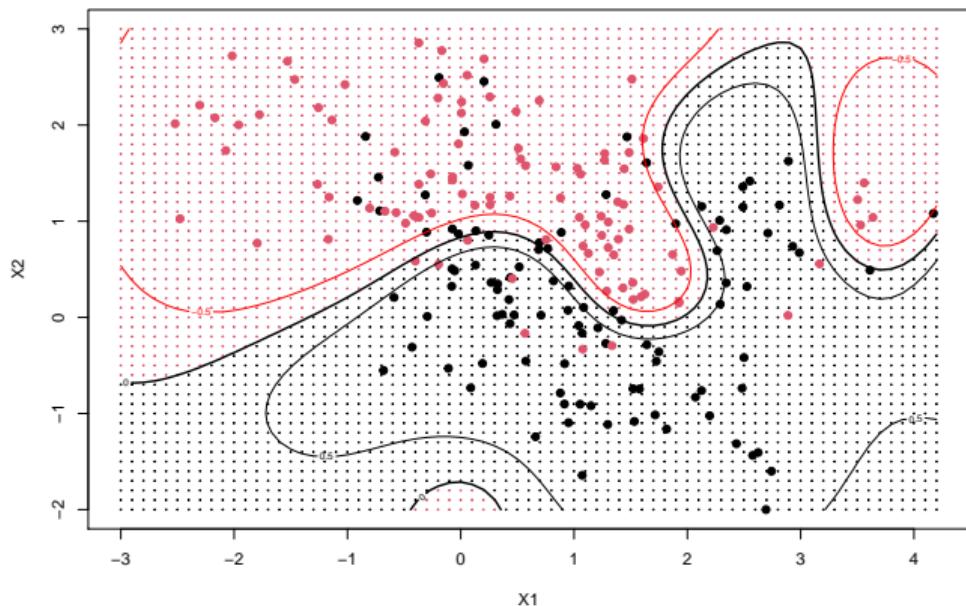
Gaussian Kernel - altering cost, keep gamma = 0.5

```
model <- svm(y ~ . , data = df, scale = FALSE, kernel = "radial", gamma = 0.5, cost = 5)
```



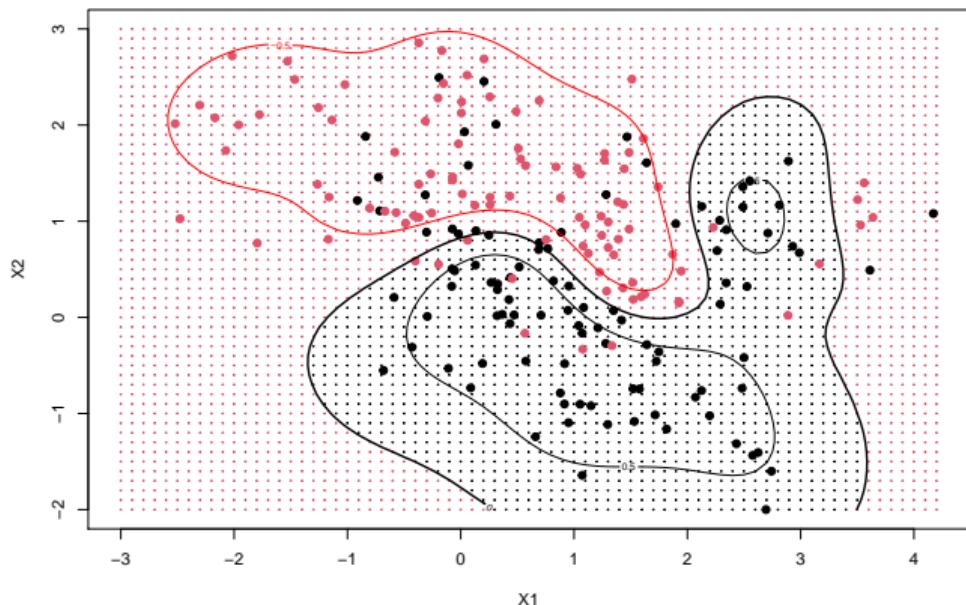
Gaussian Kernel - altering cost, keep gamma = 0.5

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



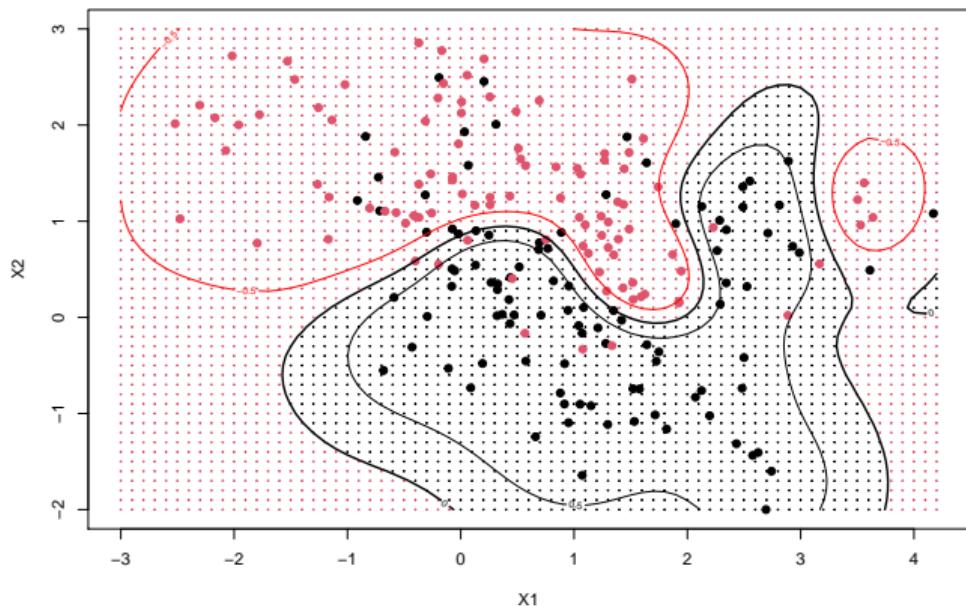
Gaussian Kernel - altering cost, keep gamma = 2

```
model <- svm(y ~ . , data = df, scale = FALSE, kernel = "radial", gamma = 2, cost = 0.1)
```



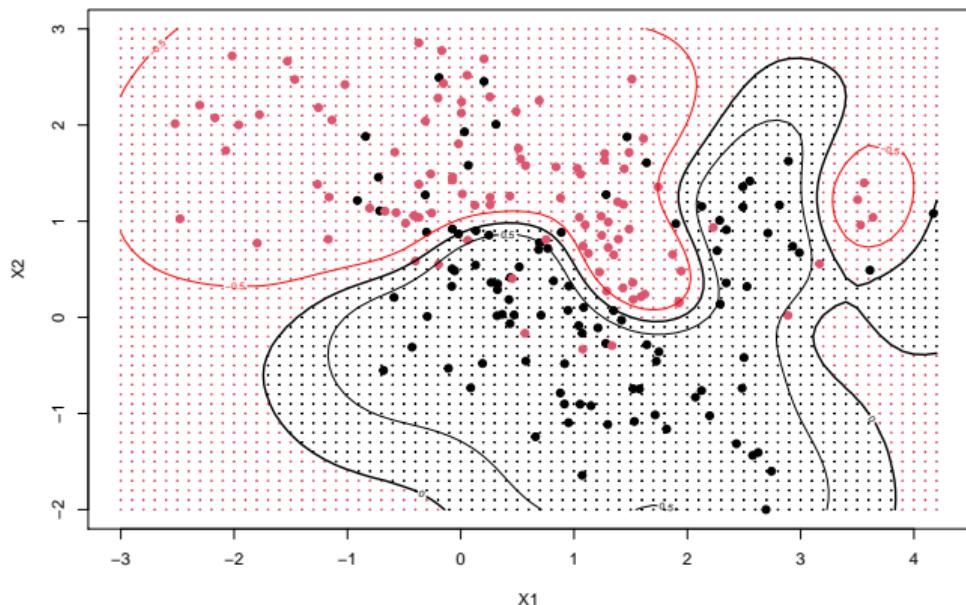
Gaussian Kernel - altering cost, keep gamma = 2

```
model <- svm(y ~ . , data = df, scale = FALSE, kernel = "radial", gamma = 2, cost = 0.5)
```



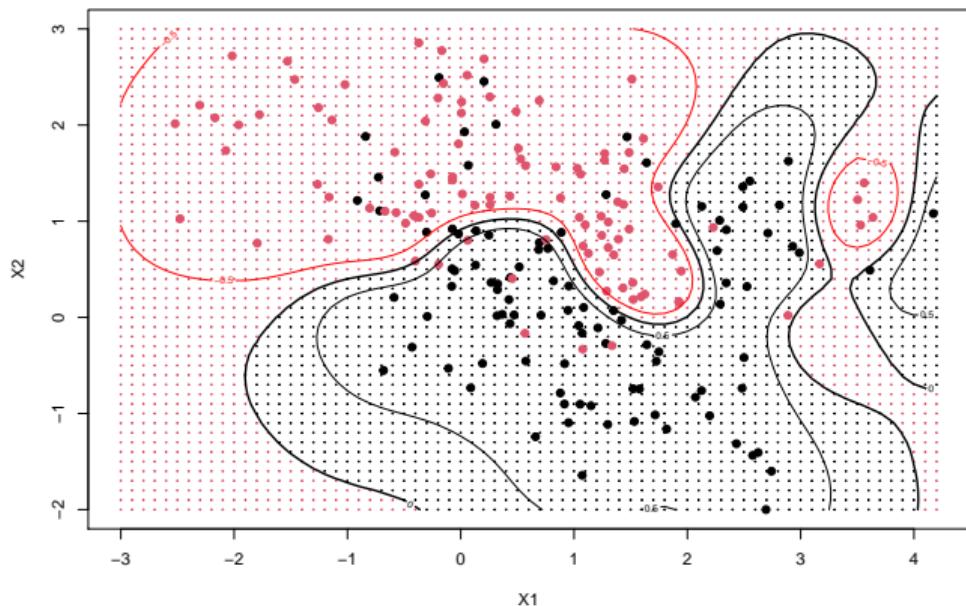
Gaussian Kernel - altering cost, keep gamma = 2

```
model <- svm(y ~ . , data = df, scale = FALSE, kernel = "radial", gamma = 2, cost = 1)
```



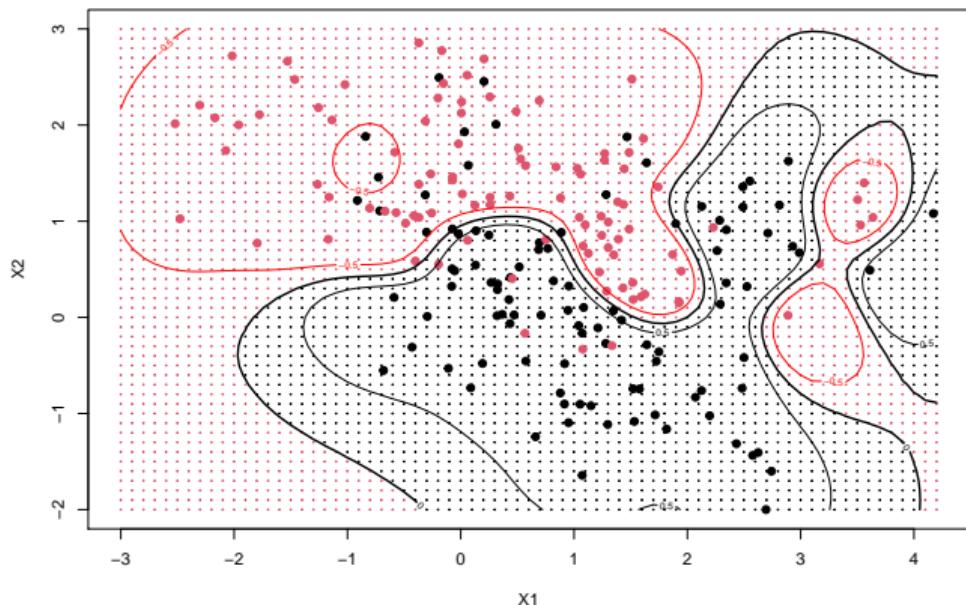
Gaussian Kernel - altering cost, keep gamma = 2

```
model <- svm(y ~ . , data = df, scale = FALSE, kernel = "radial", gamma = 2, cost = 2)
```



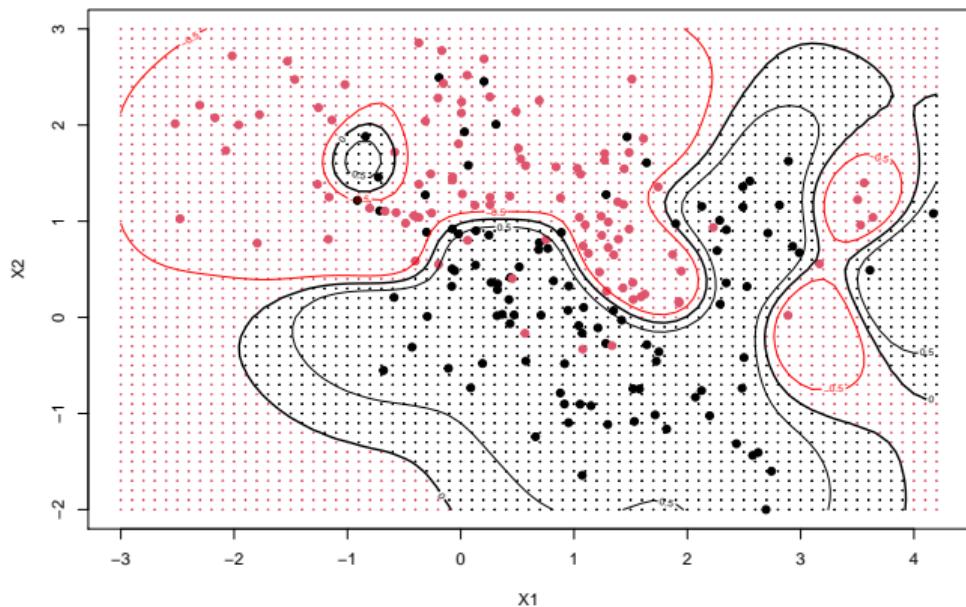
Gaussian Kernel - altering cost, keep gamma = 2

```
model <- svm(y ~ . , data = df, scale = FALSE, kernel = "radial", gamma = 2, cost = 5)
```



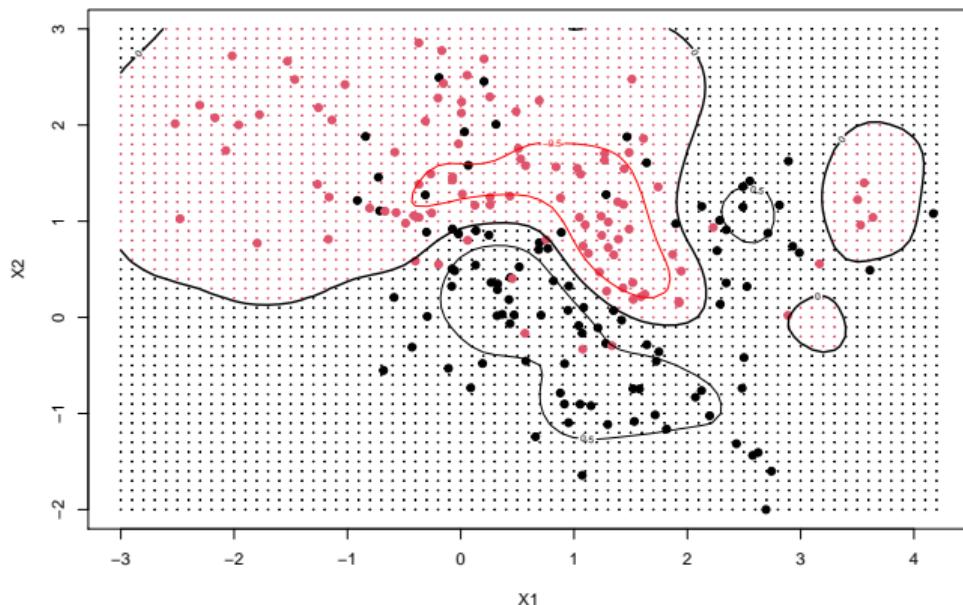
Gaussian Kernel - altering cost, keep gamma = 2

```
model <- svm(y ~ . , data = df, scale = FALSE, kernel = "radial", gamma = 2, cost = 10)
```



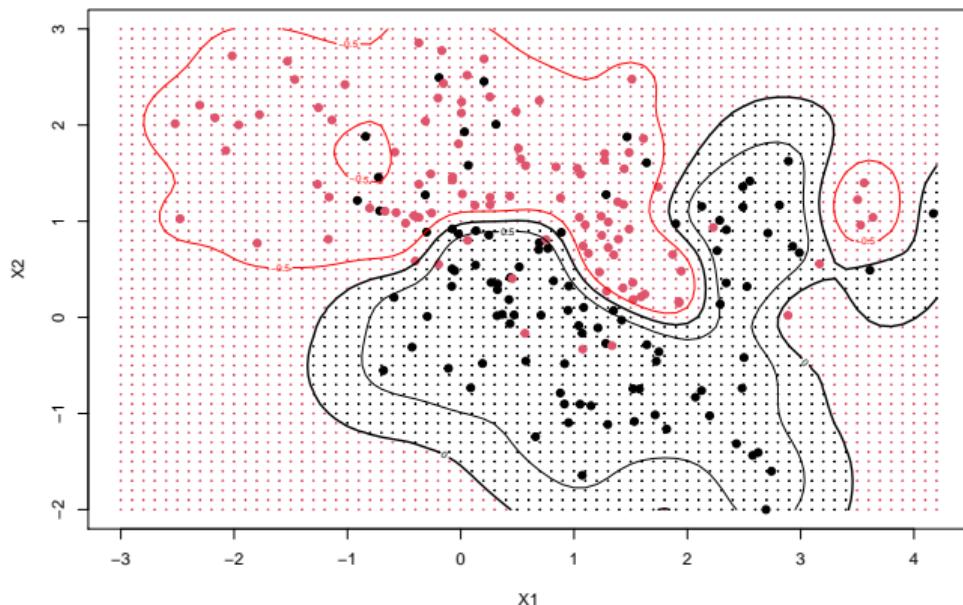
Gaussian Kernel - altering cost, keep gamma = 5

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



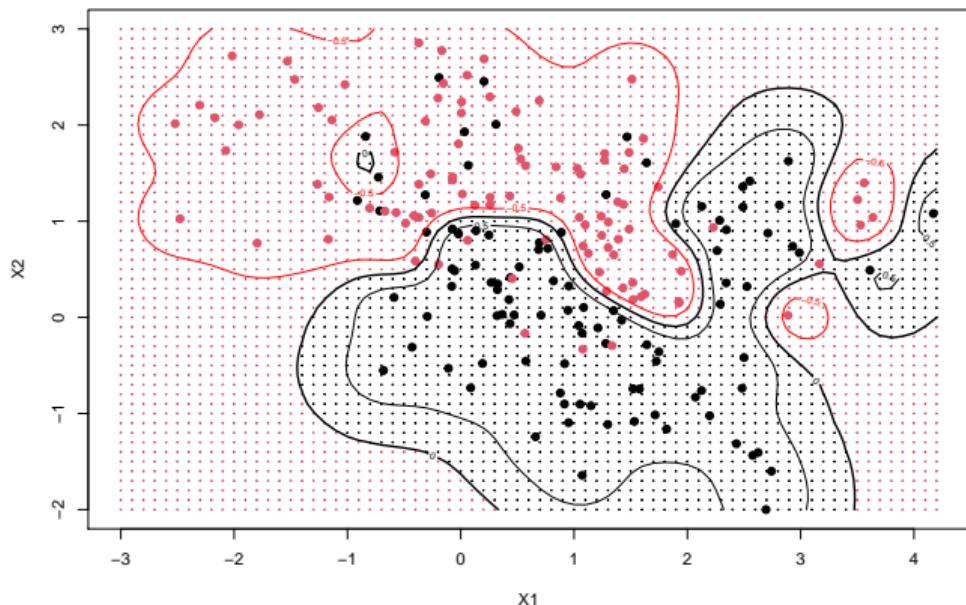
Gaussian Kernel - altering cost, keep gamma = 5

```
model <- svm(y ~ . , data = df, scale = FALSE, kernel = "radial", gamma = 5, cost = 0.5)
```



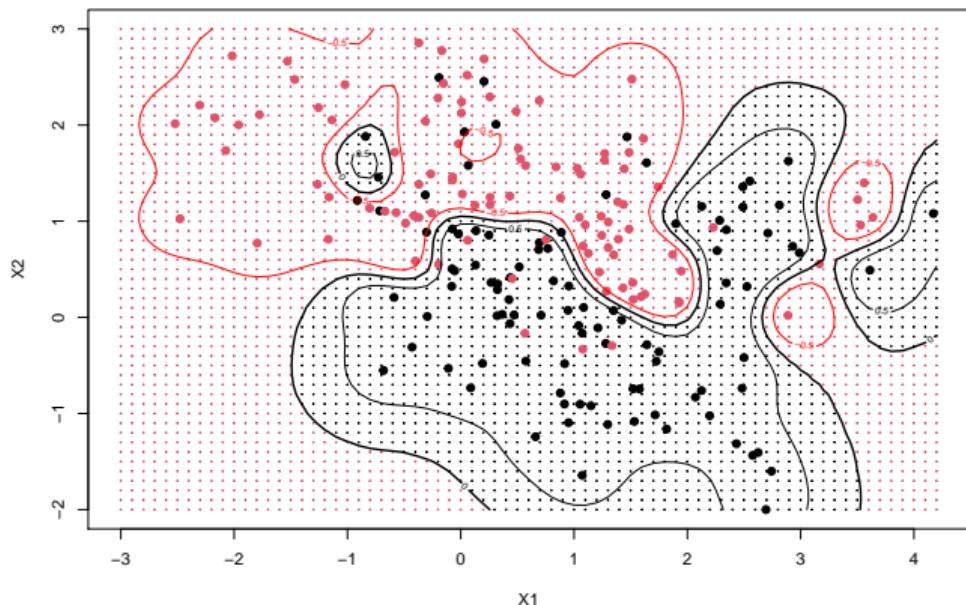
Gaussian Kernel - altering cost, keep gamma = 5

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



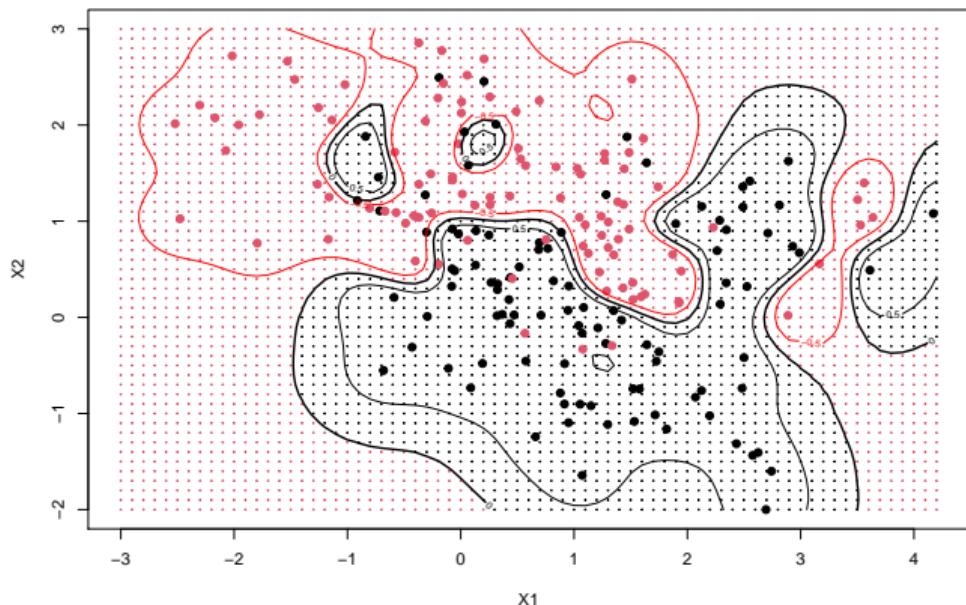
Gaussian Kernel - altering cost, keep gamma = 5

```
model <- svm(y ~ . , data = df, scale = FALSE, kernel = "radial", gamma = 5, cost = 2)
```



Gaussian Kernel - altering cost, keep gamma = 5

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



Gaussian Kernel - altering cost, keep gamma = 5

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

