Support Vector Machines

Introduction to Statistical Learning Chapter 9

Recall: Logistic Regression

A classification algorithm which assigns **probabilities** in a way which rewards high confidence on correct predictions, and lower confidence on incorrect predictions.

These probabilities can be translated into predictions. Typically, probability >= 0.5 corresponds to a prediction of "True".

It is a linear method - meaning that it creates a (hyper)plane decision boundary.

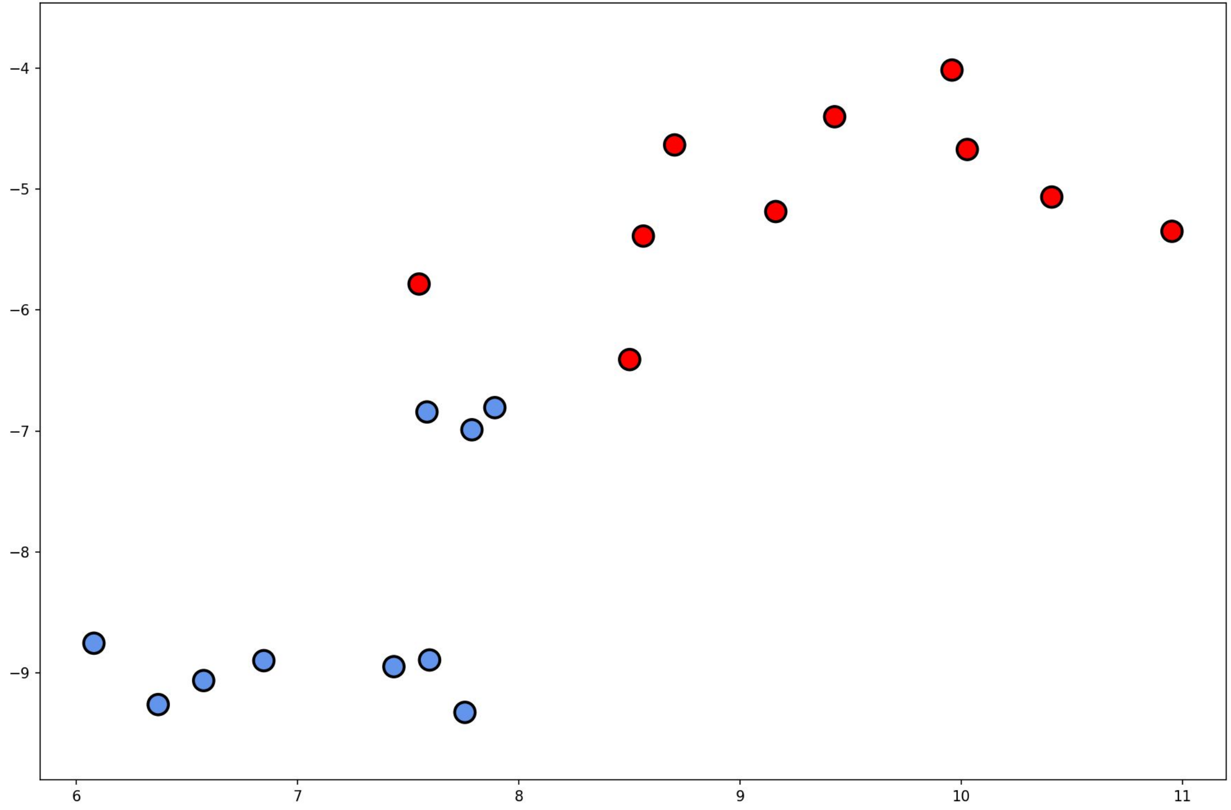
Support Vector Machines

Support Vector Machines, do not predict a probability, but simply predict a class (True/False, for example).

They do this by creating a decision function, which determines a decision boundary.

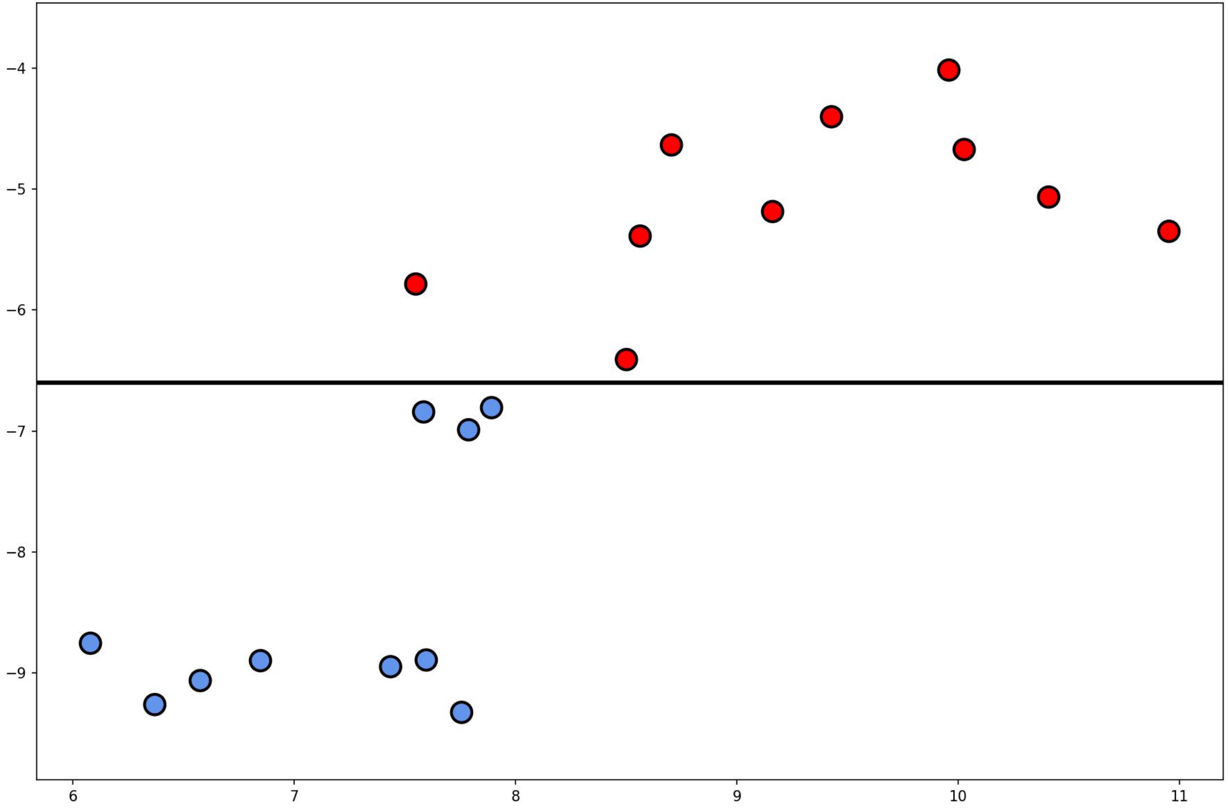
Predicts “True” if the value of the decision function is positive and “False” if it is negative.

Support Vector Machines



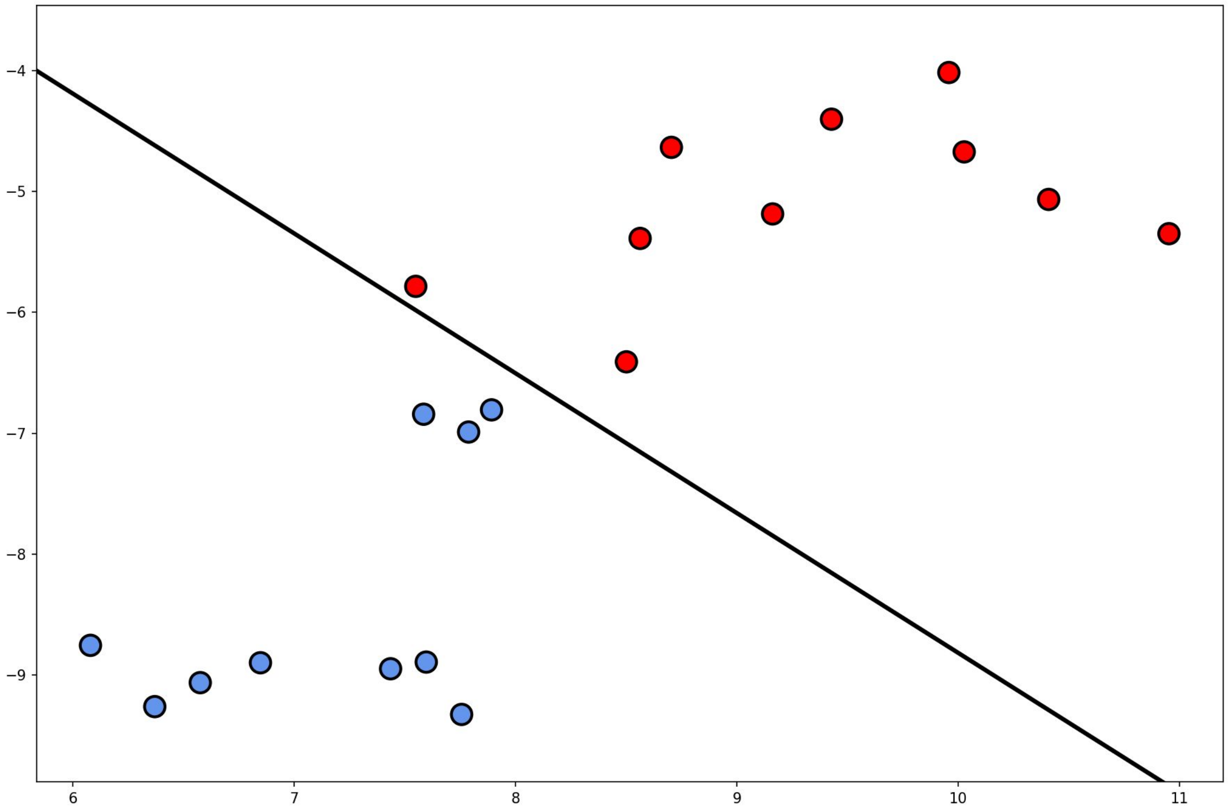
Goal: Find a line (or hyperplane in higher dimensions) that separates the two classes.

Support Vector Machines



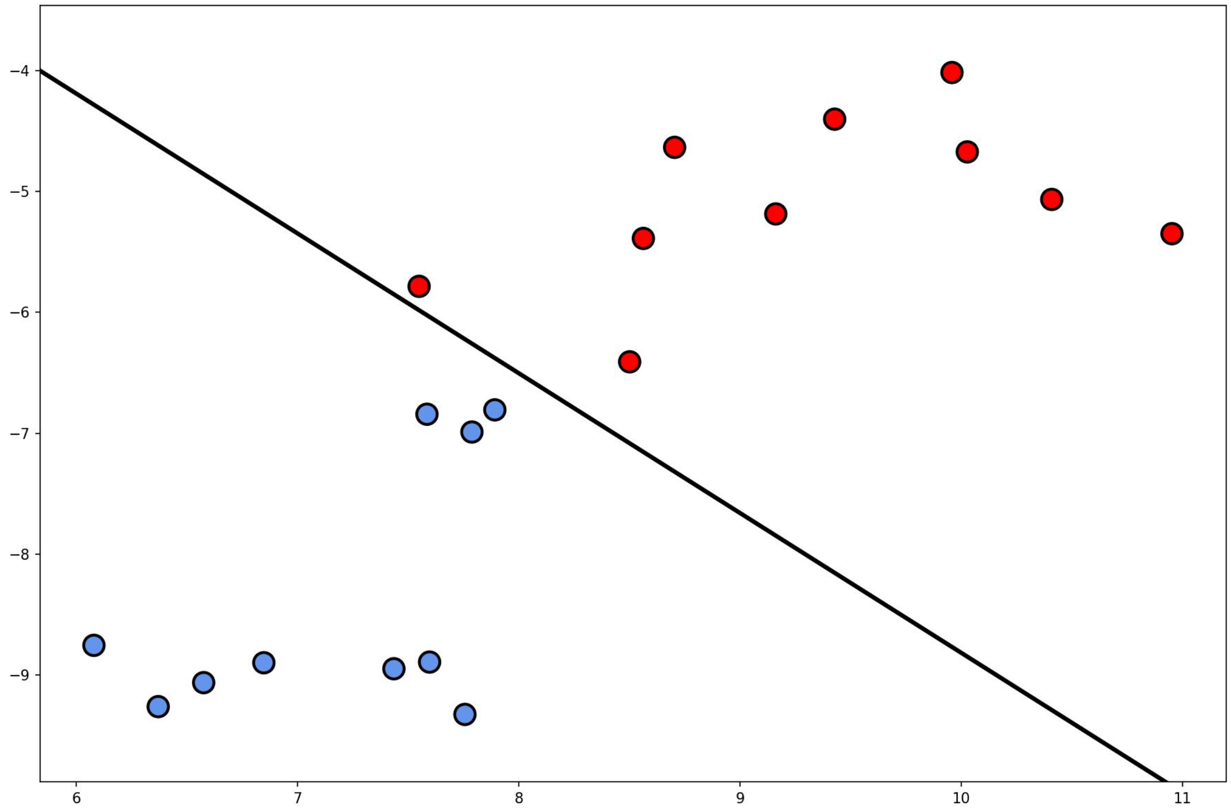
Here is one possible line.

Support Vector Machines



Here is another.

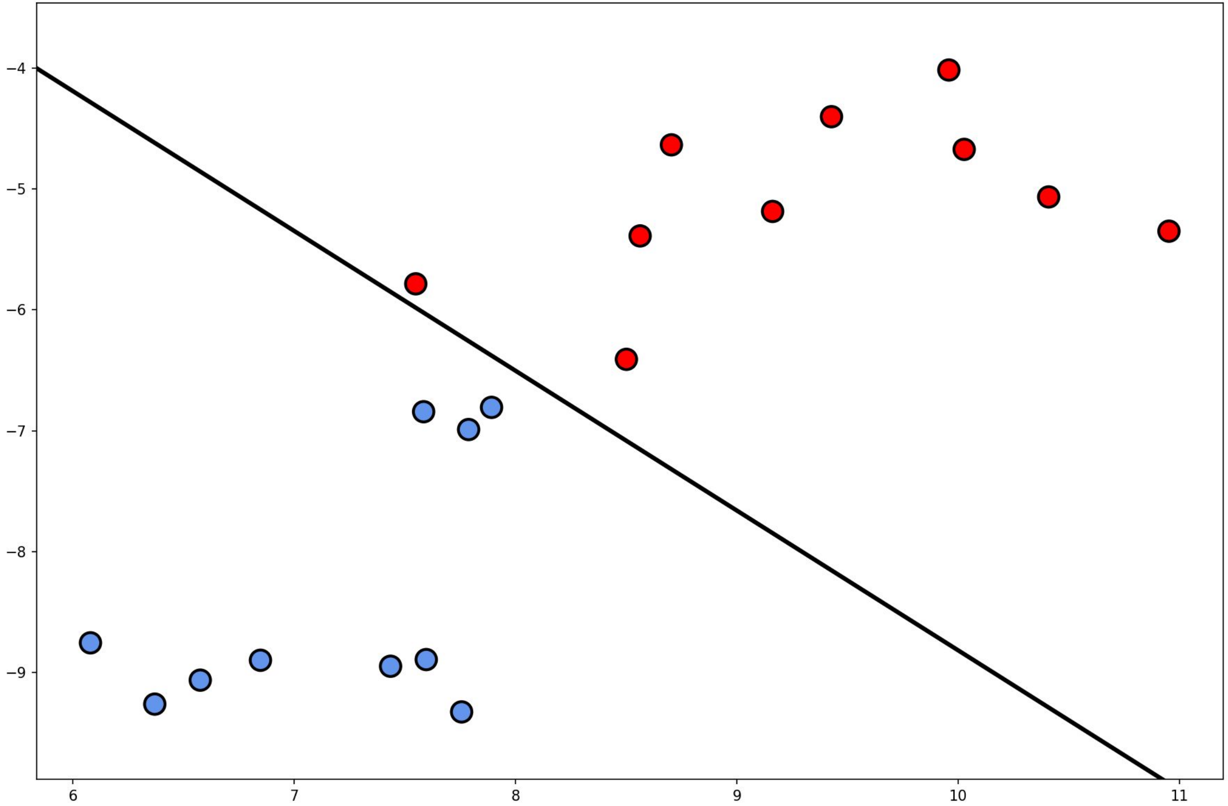
Support Vector Machines



Here is another.

How do we decide?

Support Vector Machines

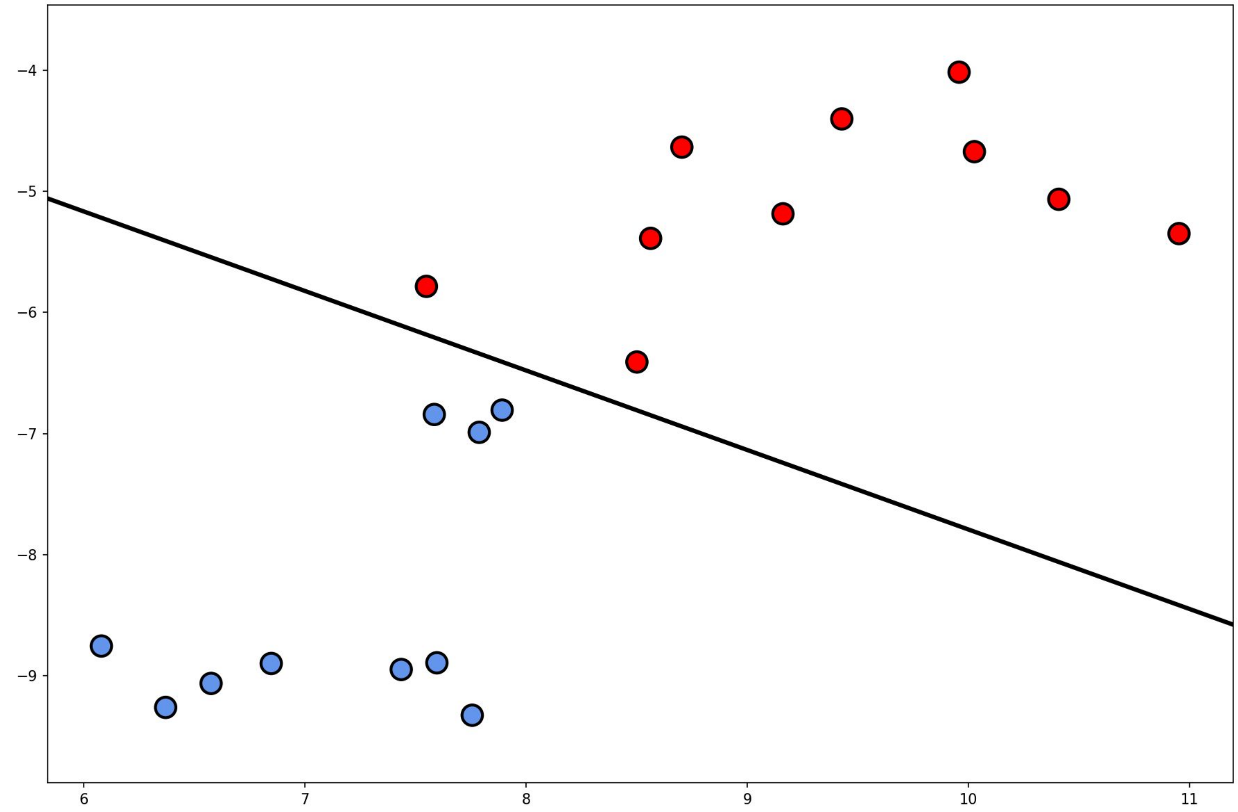


Here is another.

How do we decide?

Support vector machines find the line that maximizes the distance to the points closest to the boundary.

Support Vector Machines



This is the line that has the maximum “margin”, or distance to the closest points.

Support Vector Machines

To determine this line/hyperplane, we use a “decision function” that looks like:

$$\hat{f}(\vec{x}) = \beta_0 + \beta_1 x^{(1)} + \cdots + \beta_k x^{(k)}$$

Support Vector Machines

To determine this line/hyperplane, we use a “decision function” that looks like:

$$\hat{f}(\vec{x}) = \beta_0 + \beta_1 x^{(1)} + \cdots + \beta_k x^{(k)}$$

The sign of f(x) (which corresponds to which side of the hyperplane the point is on) determines the predicted class

f(x) > 0: Predict class A/True

f(x) <0: Predict class B/False

Support Vector Machines

To determine this line/hyperplane, we use a “decision function” that looks like:

$$\hat{f}(\vec{x}) = \beta_0 + \beta_1 x^{(1)} + \cdots + \beta_k x^{(k)}$$

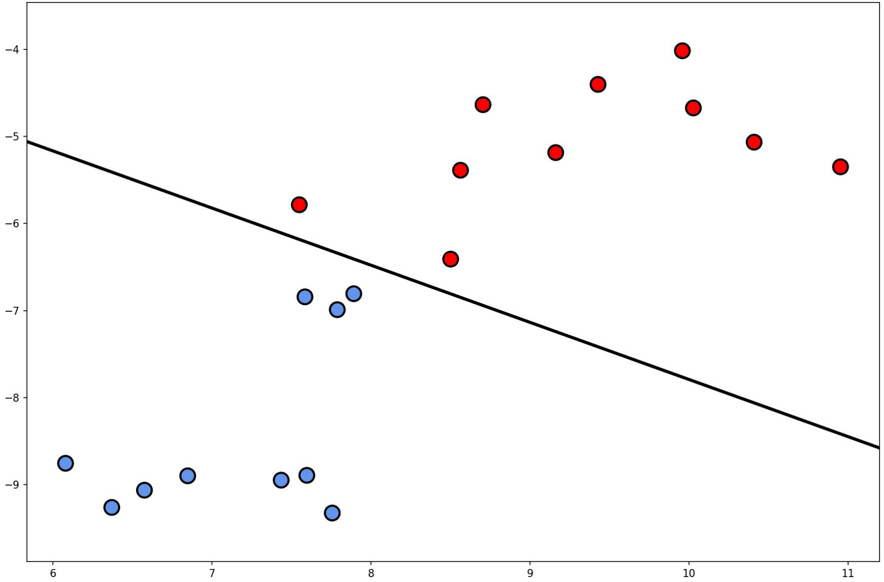
The sign of f(x) (which corresponds to which side of the hyperplane the point is on) determines the predicted class

f(x) > 0: Predict class A/True

f(x) <0: Predict class B/False

The boundary is formed by all points such that f(x) = 0

Support Vector Machines



Here, the decision function is given by

$$\hat{f}(\vec{x}) = 3.07 + 1.647x^{(1)} + 2.508x^{(2)}$$

Warning: Math Incoming

Recall: Dot Products

Given two vectors

$$\vec{x} = \langle x^{(1)}, x^{(2)}, \dots, x^{(k)} \rangle$$

$$\vec{y} = \langle y^{(1)}, y^{(2)}, \dots, y^{(k)} \rangle$$

The dot product is given by:

$$\langle \vec{x}, \vec{y} \rangle = \sum_{i=1}^{k} x^{(i)} \cdot y^{(i)} = x^{(1)}y^{(1)} + \cdots x^{(k)}y^{(k)}$$

Geometrically, $\langle \vec{x}, \vec{y} \rangle = ||\vec{x}|| \cdot ||\vec{y}|| \cdot \cos(\theta)$

where θ is the angle between the vectors.

Support Vector Machines

$$\hat{f}(\vec{x}) = \beta_0 + \beta_1 x^{(1)} + \cdots + \beta_k x^{(k)}$$

Support Vector Machines

$$\hat{f}(\vec{x}) = \beta_0 + \beta_1 x^{(1)} + \cdots + \beta_k x^{(k)} \quad = \beta_0 + \sum_{i=1}^{k} \beta_i x^{(i)}$$

Support Vector Machines

$$\hat{f}(\vec{x}) = \beta_0 + \beta_1 x^{(1)} + \cdots + \beta_k x^{(k)} \quad = \beta_0 + \sum_{i=1}^{k} \beta_i x^{(i)}$$

Given training data: $\vec{x}_1, \ldots, \vec{x}_n$

Find α_j so that for all *i >=1*, $\beta_i = \sum_{j=1}^{n} \alpha_j x_j^{(i)}$

Support Vector Machines

$$\hat{f}(\vec{x}) = \beta_0 + \beta_1 x^{(1)} + \cdots + \beta_k x^{(k)} \quad = \beta_0 + \sum_{i=1}^{k} \beta_i x^{(i)}$$

Given training data: $\vec{x}_1, \ldots, \vec{x}_n$

Find α_j so that for all *i >=1*, $\beta_i = \sum_{j=1}^{n} \alpha_j x_j^{(i)}$

$$\hat{f}(\vec{x}) = \beta_0 + \sum_{i=1}^{k} \sum_{j=1}^{n} \alpha_j x_j^{(i)} x^{(i)}$$

Support Vector Machines

$$\hat{f}(\vec{x}) = \beta_0 + \beta_1 x^{(1)} + \cdots + \beta_k x^{(k)} \;\; = \beta_0 + \sum_{i=1}^{k} \beta_i x^{(i)}$$

Given training data: $\vec{x}_1, \ldots, \vec{x}_n$

Find α_j so that for all *i >=1*, $\beta_i = \sum_{j=1}^{n} \alpha_j x_j^{(i)}$

$$\hat{f}(\vec{x}) = \beta_0 + \sum_{i=1}^{k} \sum_{j=1}^{n} \alpha_j x_j^{(i)} x^{(i)} = \beta_0 + \sum_{j=1}^{n} \alpha_j \sum_{i=1}^{k} x_j^{(i)} x^{(i)}$$

Support Vector Machines

$$\hat{f}(\vec{x}) = \beta_0 + \beta_1 x^{(1)} + \cdots + \beta_k x^{(k)} \; = \beta_0 + \sum_{i=1}^{k} \beta_i x^{(i)}$$

Given training data: $\vec{x}_1, \ldots, \vec{x}_n$

Find α_j so that for all *i >=1*, $\beta_i = \sum_{j=1}^{n} \alpha_j x_j^{(i)}$

$$\hat{f}(\vec{x}) = \beta_0 + \sum_{i=1}^{k} \sum_{j=1}^{n} \alpha_j x_j^{(i)} x^{(i)} = \beta_0 + \sum_{j=1}^{n} \alpha_j \sum_{i=1}^{k} x_j^{(i)} x^{(i)}$$

$$= \beta_0 + \sum_{j=1}^{n} \alpha_j \cdot \langle \vec{x}_j, \vec{x} \rangle$$

Support Vector Machines

$$\hat{f}(\vec{x}) = \beta_0 + \beta_1 x^{(1)} + \cdots + \beta_k x^{(k)}$$

$$= \beta_0 + \sum_{j=1}^{n} \alpha_j \cdot \langle \vec{x}_j, \vec{x} \rangle$$

So, the decision boundary can be determined by a linear combination of dot products with the training data!

Support Vector Machines

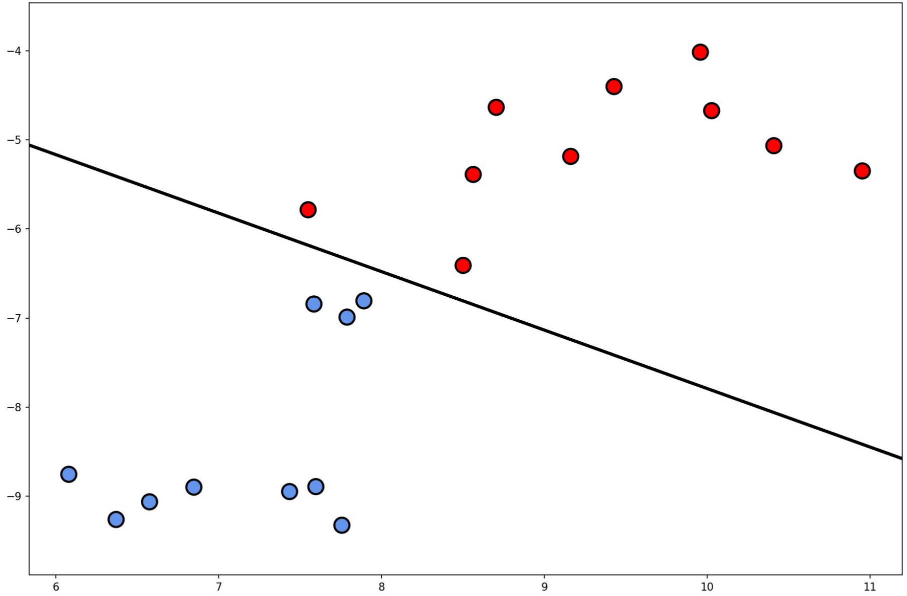
$$\hat{f}(\vec{x}) = \beta_0 + \beta_1 x^{(1)} + \cdots + \beta_k x^{(k)}$$

$$= \beta_0 + \sum_{j=1}^{n} \alpha_j \cdot \langle \vec{x}_j, \vec{x} \rangle$$

So, the decision boundary can be determined by a linear combination of dot products with the training data!

Bonus: α_j = 0 for all except for the points closest to the boundary (the **support vectors)**

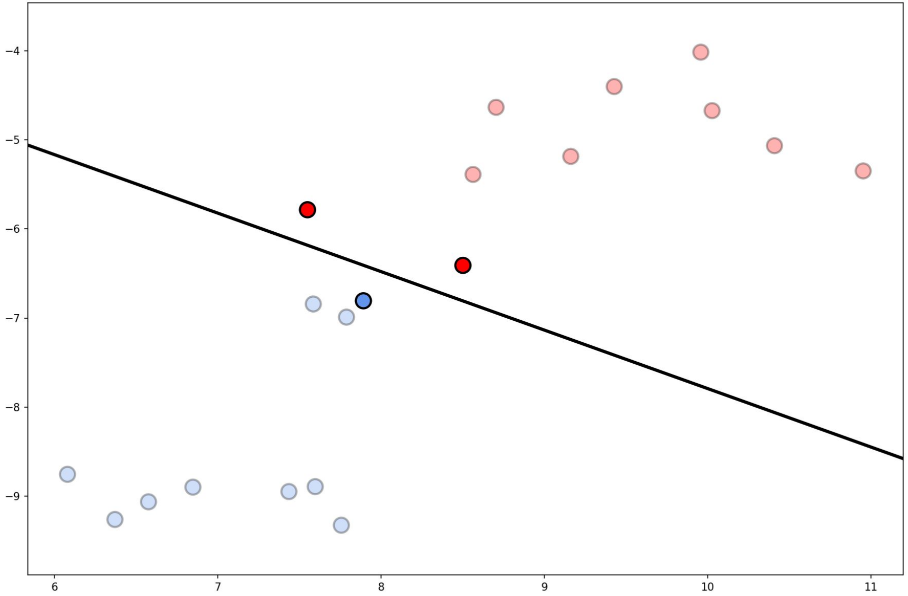
Support Vector Machines



Here, the decision function is given by

$$\hat{f}(\vec{x}) = 3.07 + 1.647x^{(1)} + 2.508x^{(2)}$$

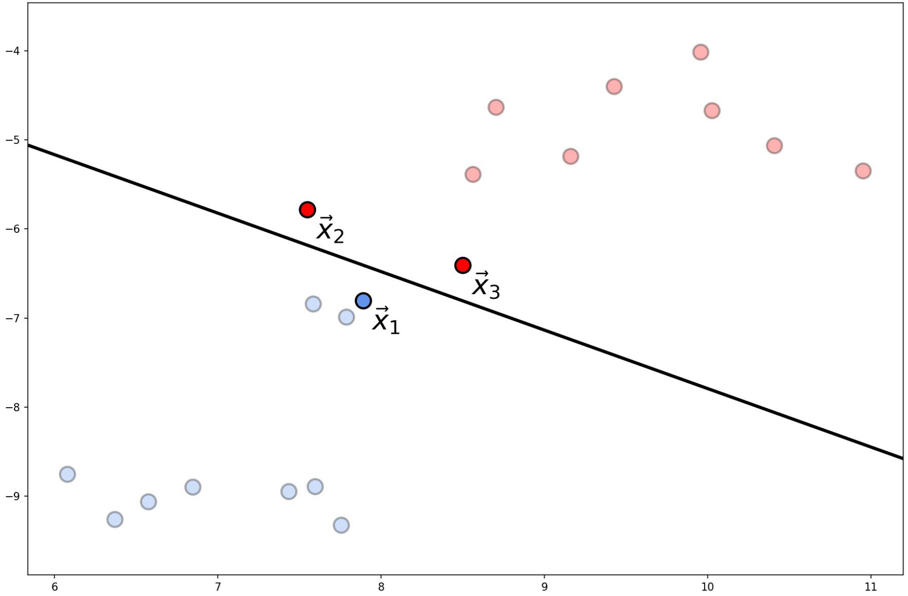
Support Vector Machines



Here, the decision function is given by

$$\hat{f}(\vec{x}) = 3.07 + 1.647x^{(1)} + 2.508x^{(2)}$$

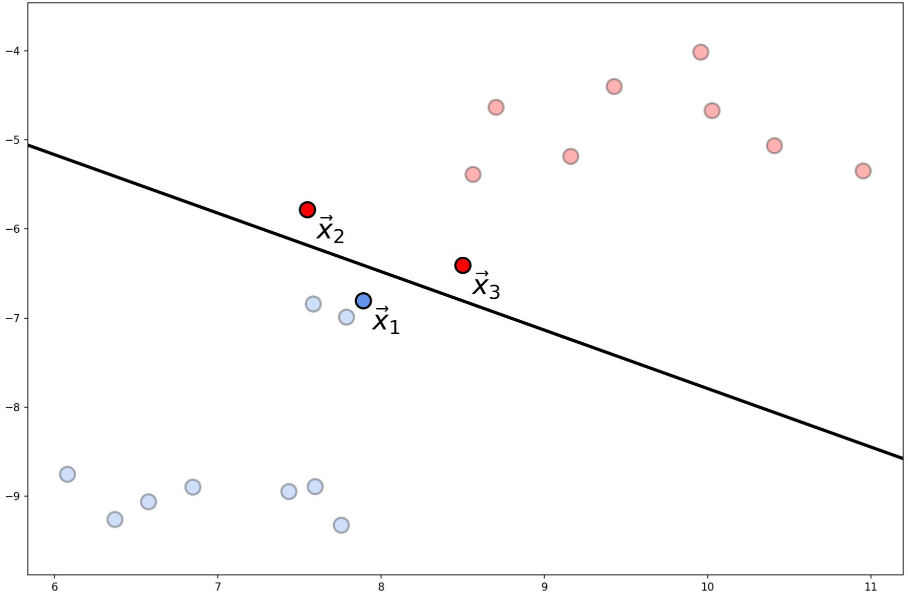
Support Vector Machines



Here, the decision function is given by

$$\hat{f}(\vec{x}) = 3.07 + 1.647x^{(1)} + 2.508x^{(2)}$$

Support Vector Machines



Here, the decision function is given by

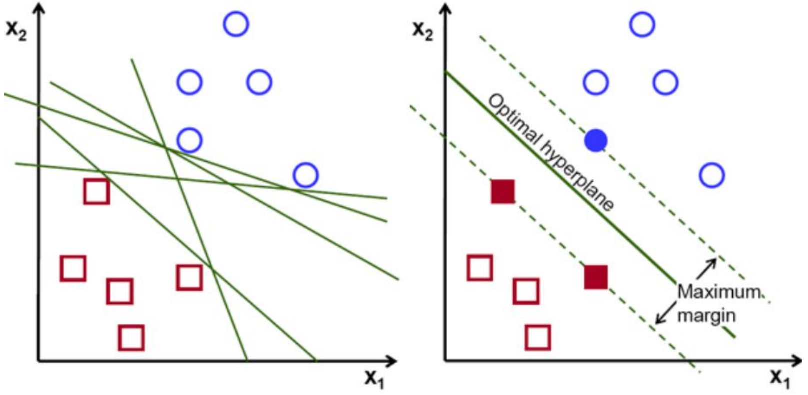
$$\hat{f}(\vec{x}) = 3.07 + 1.647x^{(1)} + 2.508x^{(2)}$$

$$= 3.07 - 4.50 \cdot \langle \vec{x}_1, \vec{x} \rangle + 1.15 \cdot \langle \vec{x}_2, \vec{x} \rangle + 3.35 \cdot \langle \vec{x}_3, \vec{x} \rangle$$

Support Vector Machines

SVMs search for the “best” separating hyperplane.

Goal: find one which maximizes the “margin” - the distance from the nearest points to the decision boundary.

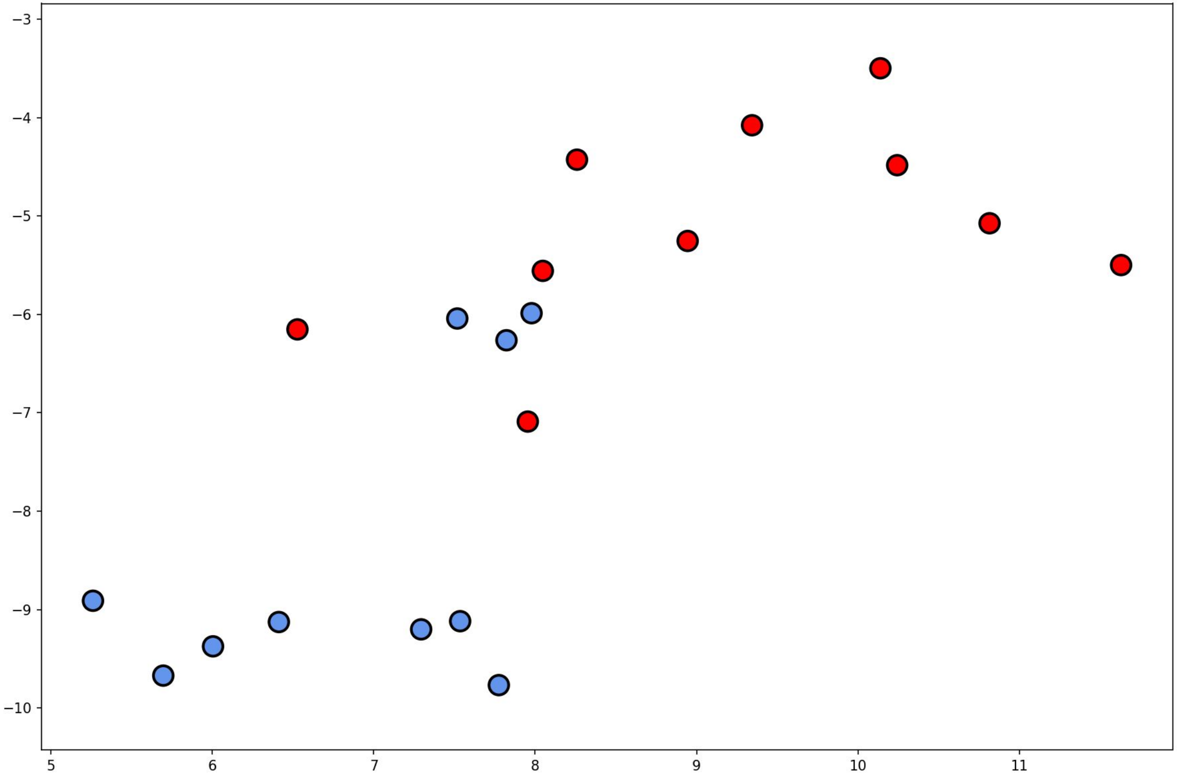


Stolen from https://medium.com/@george.drakos62/support-vector-machine-vs-logistic-regression-94cc2975433f

Support Vector Machines vs. Logistic Regression

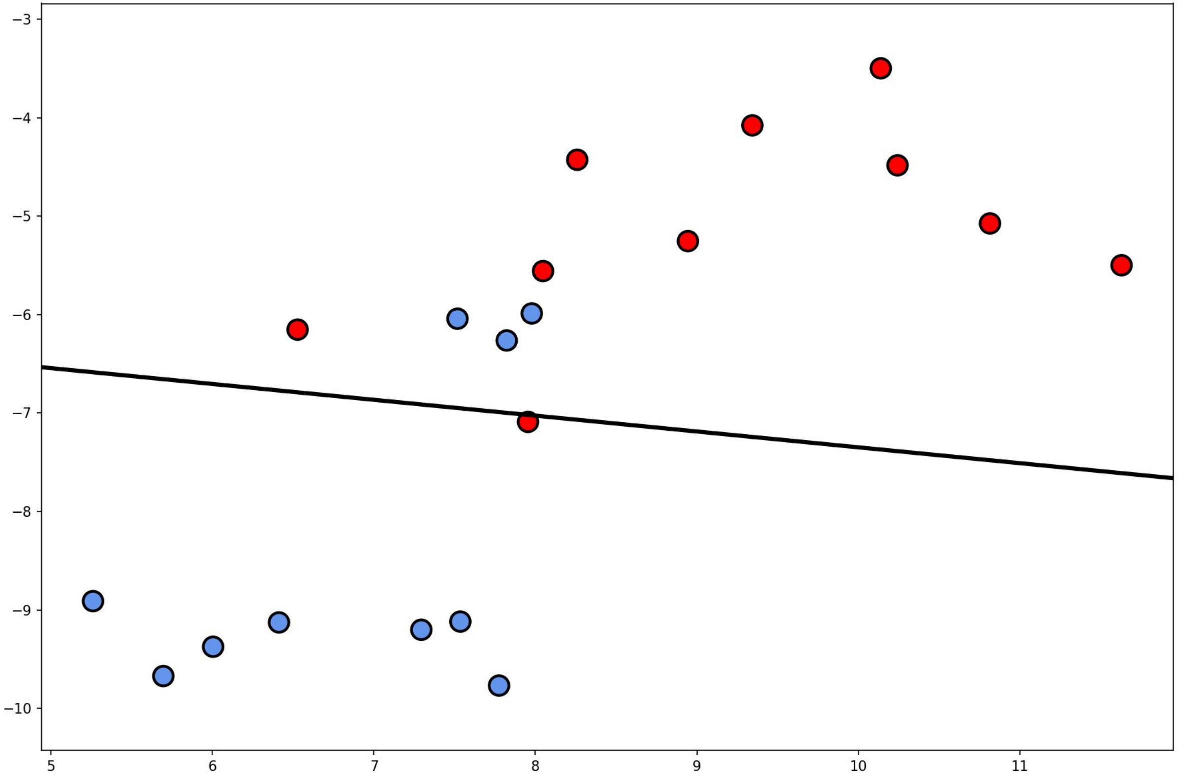
- Logistic Regression returns probabilities, which can be translated to predictions.
- SVMs only return predictions.
- SVMs rely on a smaller set of points (the support vectors) for determining the boundary.
- Logistic regression uses all the training points (they all contribute to the likelihood function)

What about non-linearly separable?



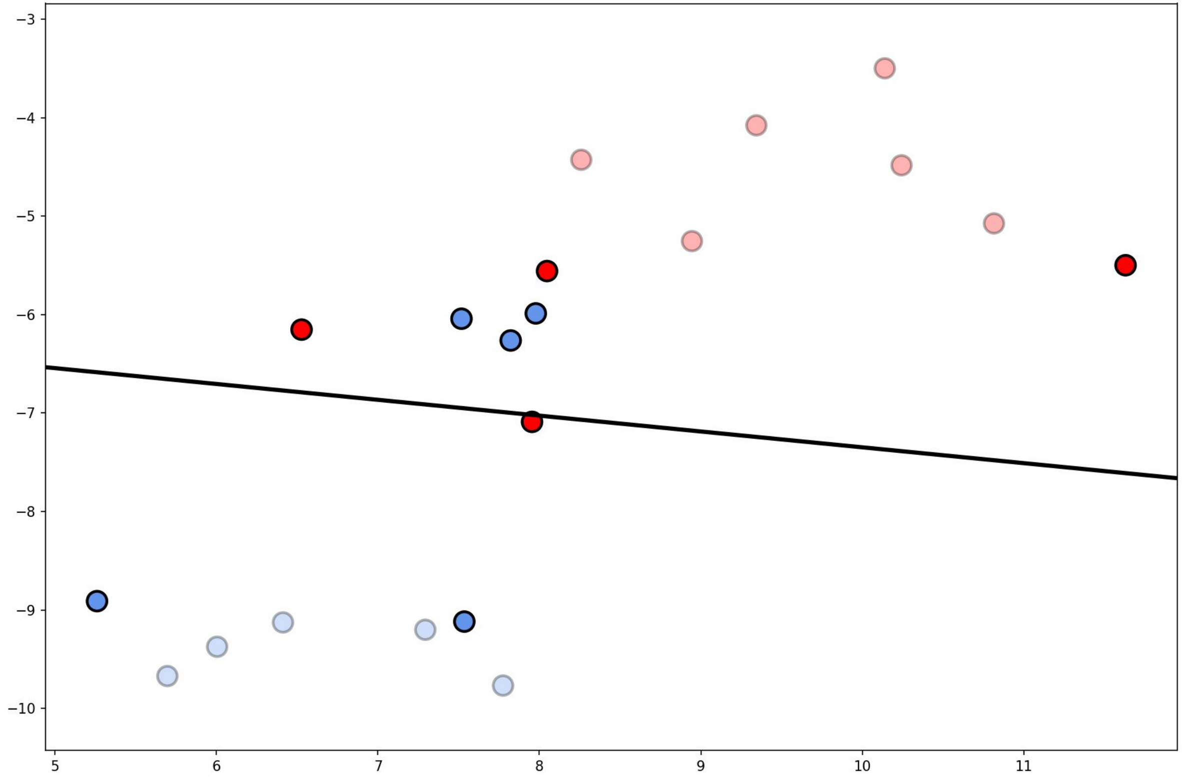
We still want the maximum margin, but allow a little bit of slack for points that cannot be classified correctly.

What about non-linearly separable?



We still want the maximum margin, but allow a little bit of slack for points that cannot be classified correctly.

What about non-linearly separable?



We still want the maximum margin, but allow a little bit of slack for points that cannot be classified correctly.

Support Vector Machines

SVMs can be used on problems where there is a nonlinear decision boundary, by applying a kernel function to transform the original space:

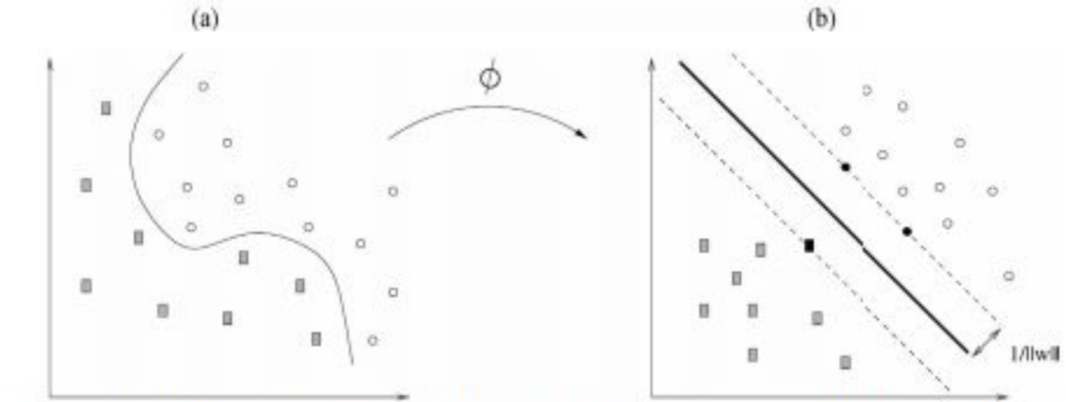


FIGURE 1: An illustration of a SVM model for two groups modified from Moguerza & Muñoz (2006). Panel (a) shows the data and a non-linear discriminant function; (b) how the data becomes separable after a kernel function Φ is applied.

Image Source: http://www.scielo.org.co/pdf/rce/v35nspe2/v35nspe2a03.pdf

Kernels

Idea: we can transform our linearly inseparable data into a higher-dimensional space, where it is linearly separable.

Example: we cannot separately the two different colors of points linearly:

Kernels

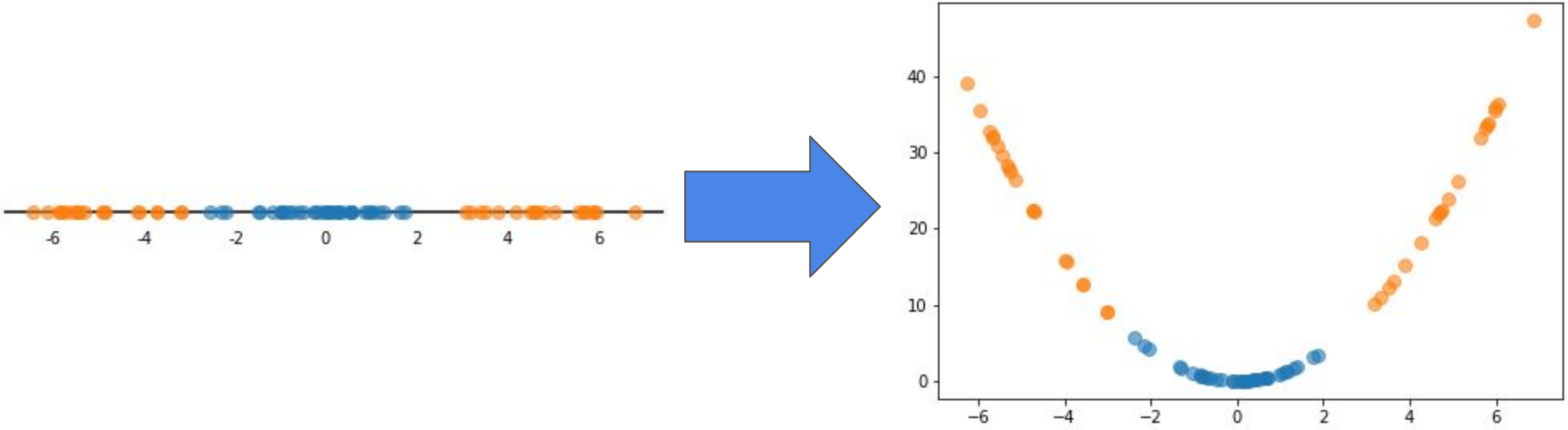
Since we only have a single variable, our decision function looks like:

$$f(x) = \beta_0 + \beta_1 \cdot x$$

Kernels

But, if we embed each point into a 2-dimensional space, we can easily separate the two colors with a line:

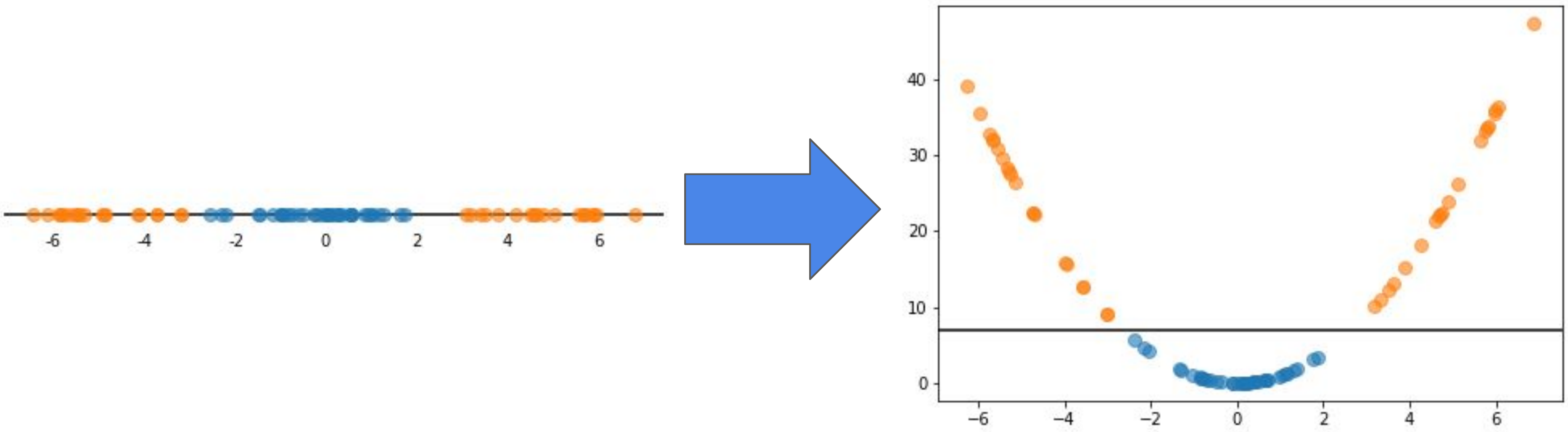
$$x \mapsto (x, x^2)$$



Kernels

But, if we embed each point into a 2-dimensional space, we can easily separate the two colors with a line:

$$x \mapsto (x, x^2)$$



Kernels

Once we embed into higher-dimensional space, our decision function now looks like:

$$f(x) = \beta_0 + \beta_1 \cdot x + \beta_2 \cdot x^2$$

Kernels

Given an embedding map φ, the associated **kernel** *K* is given by

$$K(\vec{x}, \vec{y}) = \langle \phi(\vec{x}), \phi(\vec{y}) \rangle$$

Kernels

Given an embedding map φ, the associated **kernel** *K* is given by

$$K(\vec{x}, \vec{y}) = \langle \phi(\vec{x}), \phi(\vec{y}) \rangle$$

So what?

Kernels

Given an embedding map φ, the associated **kernel** K is given by

$$K(\vec{x}, \vec{y}) = \langle \phi(\vec{x}), \phi(\vec{y}) \rangle$$

So what?

If we only care about dot products, we don't have to explicitly compute the embedding. We can instead use a kernel function!

Kernels

By mathematics black magic, we don’t really need the higher-dimensional coordinates, just a way to compute similarity (i.e. dot products) in the higher-dimensional space.

A **kernel** is a shortcut to finding the similarity without having to compute the new coordinates.

Rather than specifying a transformation map, instead specify a kernel, or “similarity” function, which is often computationally cheaper.

Kernels

Original version of decision function:

$$f(\vec{x}) = \beta_0 + \beta_1 \cdot x_1 + \beta_2 \cdot x_2 + \cdots + \beta_p \cdot x_p$$

By math, this is equivalent to one of the following form (where $x_1, \ldots, x_n$ are the data points):

dot product

$$f(\vec{x}) = \beta_0 + \sum_{i=1}^{n} \alpha_i \cdot (\vec{x} \cdot \vec{x_i})$$

Support Vector Machines

Original: $$\hat{f}(\vec{x}) = \beta_0 + \sum_{j=1}^{n} \alpha_j \cdot \langle \vec{x}_j, \vec{x} \rangle$$

Support Vector Machines

Original:

$$\hat{f}(\vec{x}) = \beta_0 + \sum_{j=1}^{n} \alpha_j \cdot \langle \vec{x}_j, \vec{x} \rangle$$

Using an embedding:

$$\hat{f}(\vec{x}) = \beta_0 + \sum_{j=1}^{n} \alpha_j \cdot \langle \phi(\vec{x}_j), \phi(\vec{x}) \rangle$$

Support Vector Machines

Original:

$$\hat{f}(\vec{x}) = \beta_0 + \sum_{j=1}^{n} \alpha_j \cdot \langle \vec{x}_j, \vec{x} \rangle$$

Using an embedding:

$$\hat{f}(\vec{x}) = \beta_0 + \sum_{j=1}^{n} \alpha_j \cdot \langle \phi(\vec{x}_j), \phi(\vec{x}) \rangle$$

Using a kernel:

$$\hat{f}(\vec{x}) = \beta_0 + \sum_{j=1}^{n} \alpha_j \cdot K(\vec{x}_j, \vec{x})$$

Kernels

A kernel replaces the dot product with some kernel function *K*.

$$f(\vec{x}) = \beta_0 + \sum_{i=1}^{n} \alpha_i \cdot K(\vec{x}, \vec{x_i})$$

Kernels

The two most common types of kernels:

Polynomial: Let us work with polynomials of our features, up to a specified degree (eg. working with squared mean radius instead of just mean radius)

The degree d polynomial kernel is given by

$$K(\vec{x}, \vec{y}) = (\vec{x} \cdot \vec{y} + c)^d$$

Kernels

The two most common types of kernels:

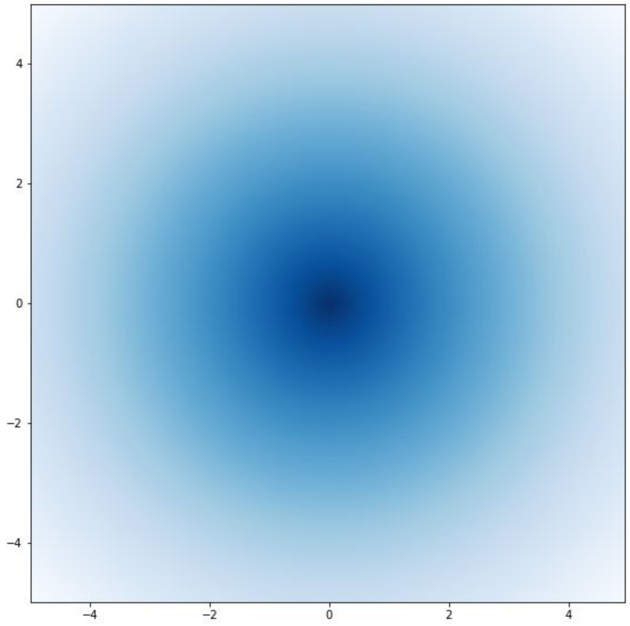
Radial Basis Functions: Each point is like a Gaussian probability distribution.

$$K(\vec{x}, \vec{y}) = e^{(-\gamma||\vec{x} - \vec{y}||)}$$

Here, K is large if the distance between x and y is small. It has a maximum value of 1 when x = y.

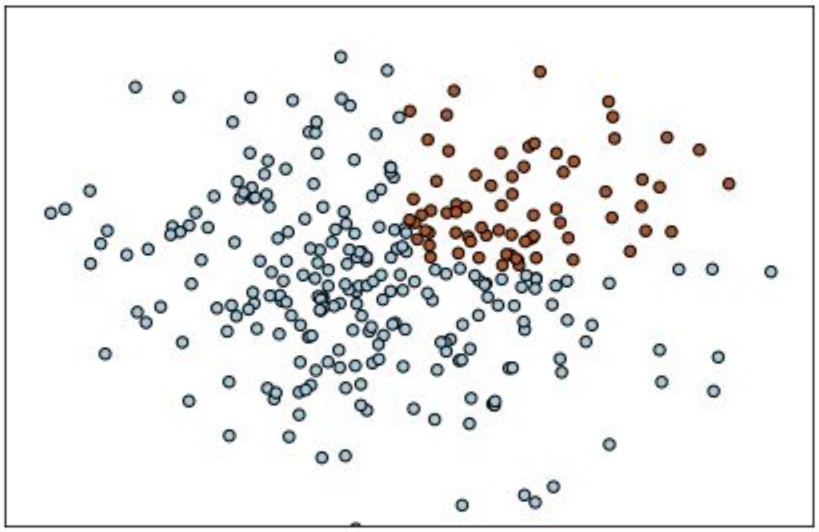
Kernels

Radial Basis Function with y = (0,0) and gamma = 0.1 as we vary x.
Here, darker corresponds to higher values of K(x, y).



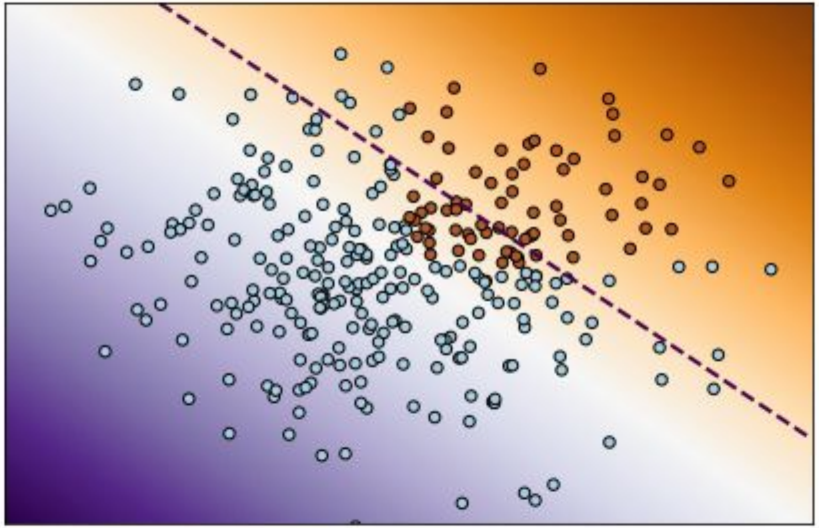
Kernels

Kernels can allow us to fit highly nonlinear decision boundaries.



Kernels

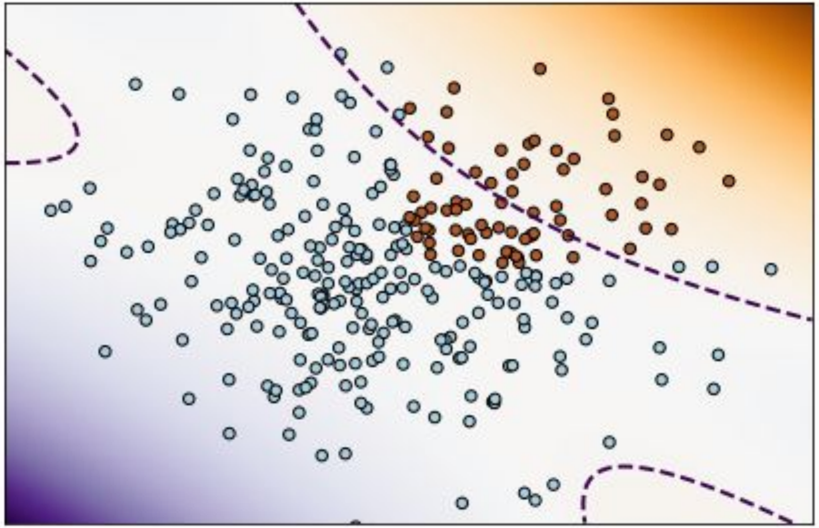
Kernels can allow us to fit highly nonlinear decision boundaries.



Linear Kernel (regular SVC)

Kernels

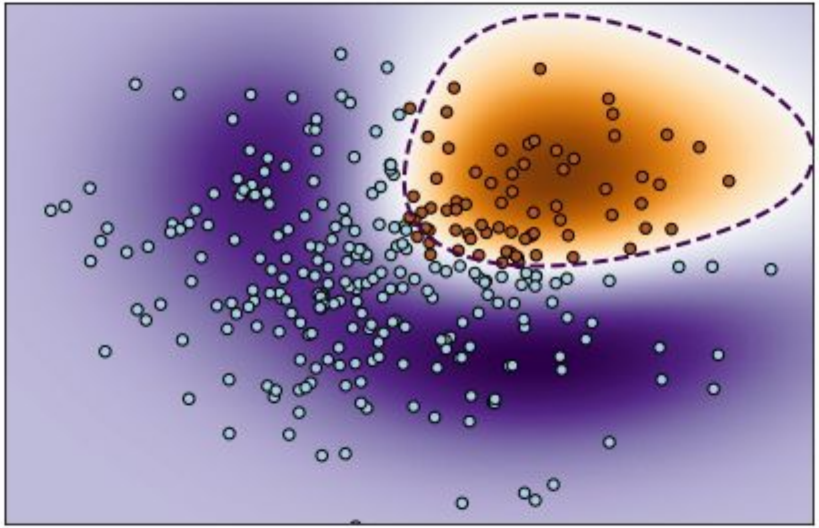
Kernels can allow us to fit highly nonlinear decision boundaries.



Polynomial Kernel (degree 3)

Kernels

Kernels can allow us to fit highly nonlinear decision boundaries.



Radial Basis Function (Gaussian) Kernel

Logistic Regression vs. Support Vector Machines

SVM can fit nonlinear decision boundaries, unlike logistic regression.

SVM does not output probabilities.

SVM is much slower to train the logistic regression, especially with a large number of observations.