



Support Vector Machines






Support Vector Machine is one of the most performant off-the-shelf supervised machine learning algorithms.

This means that when you have a problem and you try to run a SVM on it, you will often get pretty good results without many tweaks, because it is based on a strong mathematical background.



SVMs are the result of the work of several people over many years. The first SVM algorithm is attributed to Vladimir Vapnik in 1963



SVMs have been successfully used in three main areas:

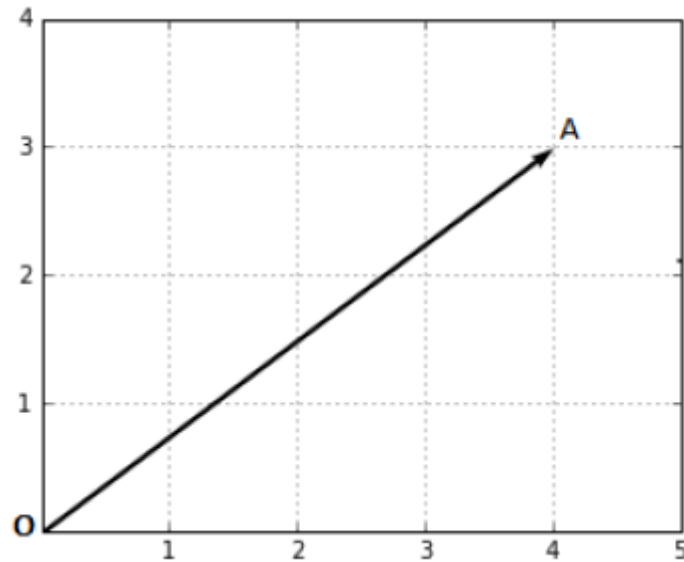
1. Text categorization
2. Image recognition
3. Bioinformatics

Specific examples include classifying

1. News stories
2. Handwritten digit recognition
3. Cancer tissue samples.

What is a vector?

A vector is a mathematical object that can be represented by an arrow.



The magnitude of a vector

The magnitude, or length, of a vector is written , and is called its **norm**.



$$OA^2 = OB^2 + AB^2$$

$$OA^2 = 4^2 + 3^2$$

$$OA^2 = 25$$

$$OA = \sqrt{25}$$

, we can calculate the norm $\|OA\|$ of vector \vec{OA} by using the Pythagorean theorem:

The direction of a vector

The direction is the second component of a vector. By definition it is a new vector for which the coordinates are the initial coordinates of our vector divided by its norm.

The direction of a vector $\mathbf{u} = (u_1, u_2)$ is the vector:

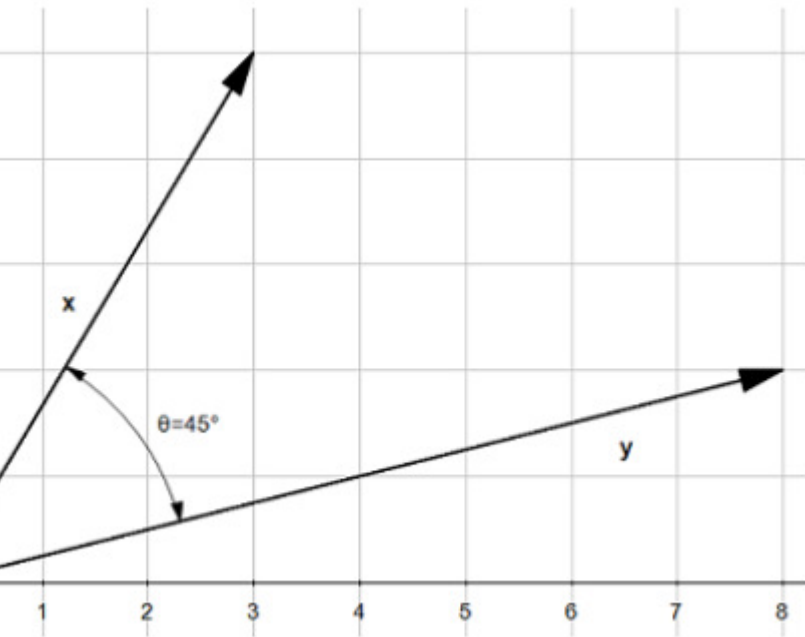
$$\mathbf{w} = \left(\frac{u_1}{\|\mathbf{u}\|}, \frac{u_2}{\|\mathbf{u}\|} \right)$$

The dot product

The dot product is an operation performed on two vectors that returns a number. A number is sometimes called a scalar; that is why the dot product is also called a scalar product.

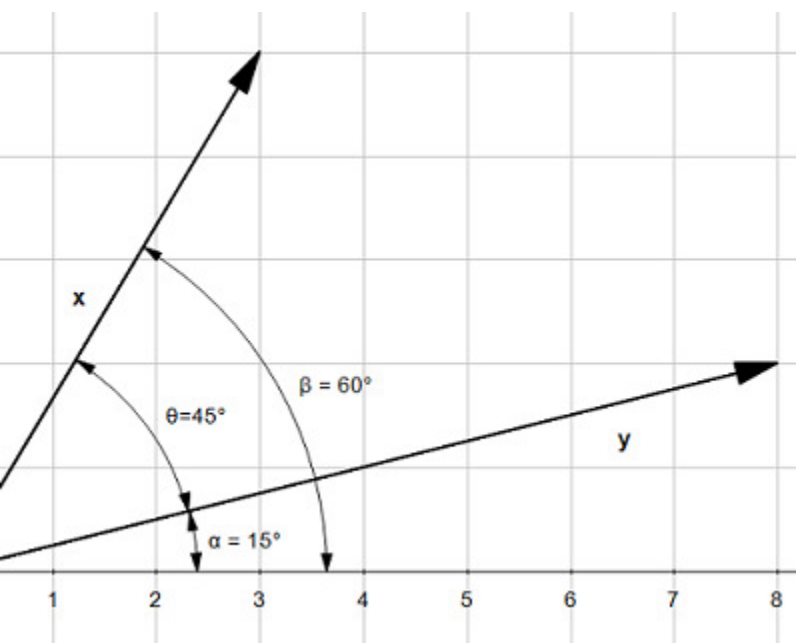
Geometric definition of the dot product

Geometrically, the dot product is the product of the Euclidean magnitudes of the two vectors and cosine of the angle between them.



This means that if we have two vectors, x and y , with an angle θ between them their dot product is.

$$\mathbf{x} \cdot \mathbf{y} = \|\mathbf{x}\| \|\mathbf{y}\| \cos(\theta)$$



$$\theta = \beta - \alpha$$

This means computing $\cos(\theta)$ is the same as computing $\cos(\beta - \alpha)$.

Using the difference identity for cosine we get:

$$\cos(\theta) = \cos(\beta - \alpha) = \cos(\beta)\cos(\alpha) + \sin(\beta)\sin(\alpha)$$

$$\cos(\theta) = \frac{x_1}{\|\mathbf{x}\|} \frac{y_1}{\|\mathbf{y}\|} + \frac{x_2}{\|\mathbf{x}\|} \frac{y_2}{\|\mathbf{y}\|}$$

$$\cos(\theta) = \frac{x_1 y_1 + x_2 y_2}{\|\mathbf{x}\| \|\mathbf{y}\|}$$

If we multiply both sides by $\|\mathbf{x}\| \|\mathbf{y}\|$ we get:

$$\|\mathbf{x}\| \|\mathbf{y}\| \cos(\theta) = x_1 y_1 + x_2 y_2$$

As we already know that:

$$\|\mathbf{x}\| \|\mathbf{y}\| \cos(\theta) = \mathbf{x} \cdot \mathbf{y}$$

This means the dot product can also be written:

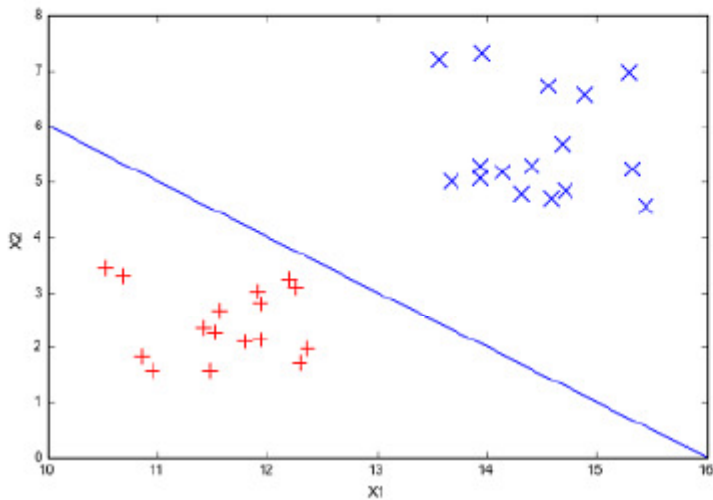
$$\mathbf{x} \cdot \mathbf{y} = x_1 y_1 + x_2 y_2$$

$$\mathbf{x} \cdot \mathbf{y} = \sum_{i=1}^2 (x_i y_i)$$

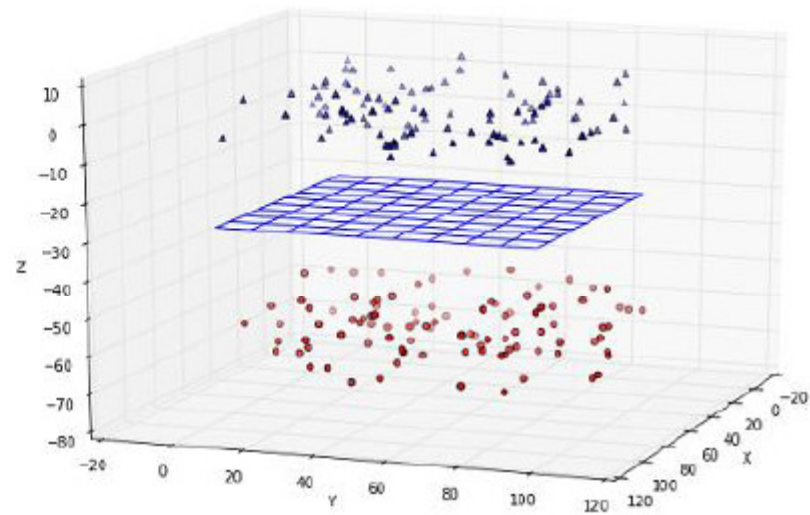
In a more general way, for n -dimensional vectors, we can write:

$$\mathbf{x} \cdot \mathbf{y} = \sum_{i=1}^n (x_i y_i)$$

Linearly separable data

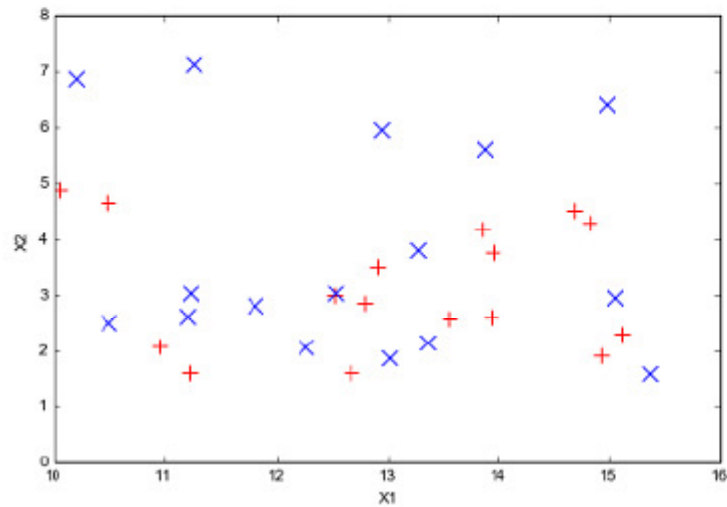


Data separated by a line

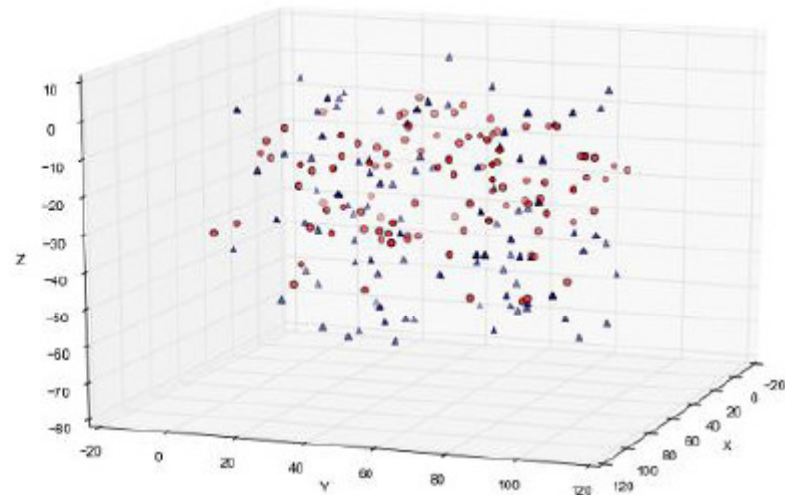


Data separated by a plane

Non-Linearly separable data



Non-linearly separable data in 2D



Non-linearly separable data in 3D

What is a hyperplane?

In geometry, a hyperplane is a subspace of one dimension less than its ambient space.

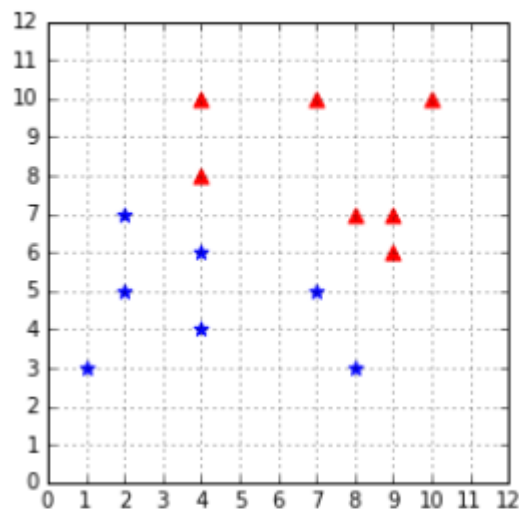
Given vectors $\mathbf{w} = (w_0, w_1)$, $\mathbf{x} = (x, y)$ and b , we can define a hyperplane having the equation:

$$\mathbf{w} \cdot \mathbf{x} + b = 0$$

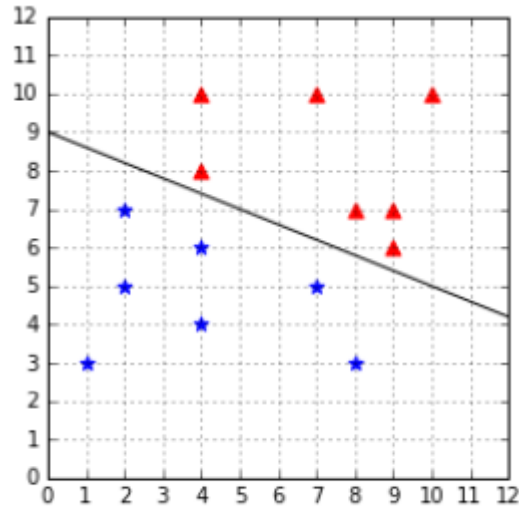
From this equation, we can have another insight into what a hyperplane is: it is the set of points satisfying $\mathbf{w} \cdot \mathbf{x} + b = 0$. And, if we keep just the essence of this definition: a hyperplane is a set of points.

Classifying data with a hyperplane

For instance, with the vector $\mathbf{w} = (0.4, 1.0)$ and $b = -9$ we get the hyperplane in Figure



A linearly separable dataset



We associate each vector \mathbf{x}_i with a label y_i , which can have the value $+1$ or -1 (respectively the triangles and the stars in

We define a hypothesis function h :

$$h(\mathbf{x}_i) = \begin{cases} +1 & \text{if } \mathbf{w} \cdot \mathbf{x}_i + b \geq 0 \\ -1 & \text{if } \mathbf{w} \cdot \mathbf{x}_i + b < 0 \end{cases}$$

which is equivalent to:

$$h(\mathbf{x}_i) = \text{sign}(\mathbf{w} \cdot \mathbf{x}_i + b)$$

Hypothesis function

The SVMs use the same hypothesis function as the Perceptron. The class of an example \mathbf{x}_i is given by:

$$h(\mathbf{x}_i) = \text{sign}(\mathbf{w} \cdot \mathbf{x}_i + b)$$

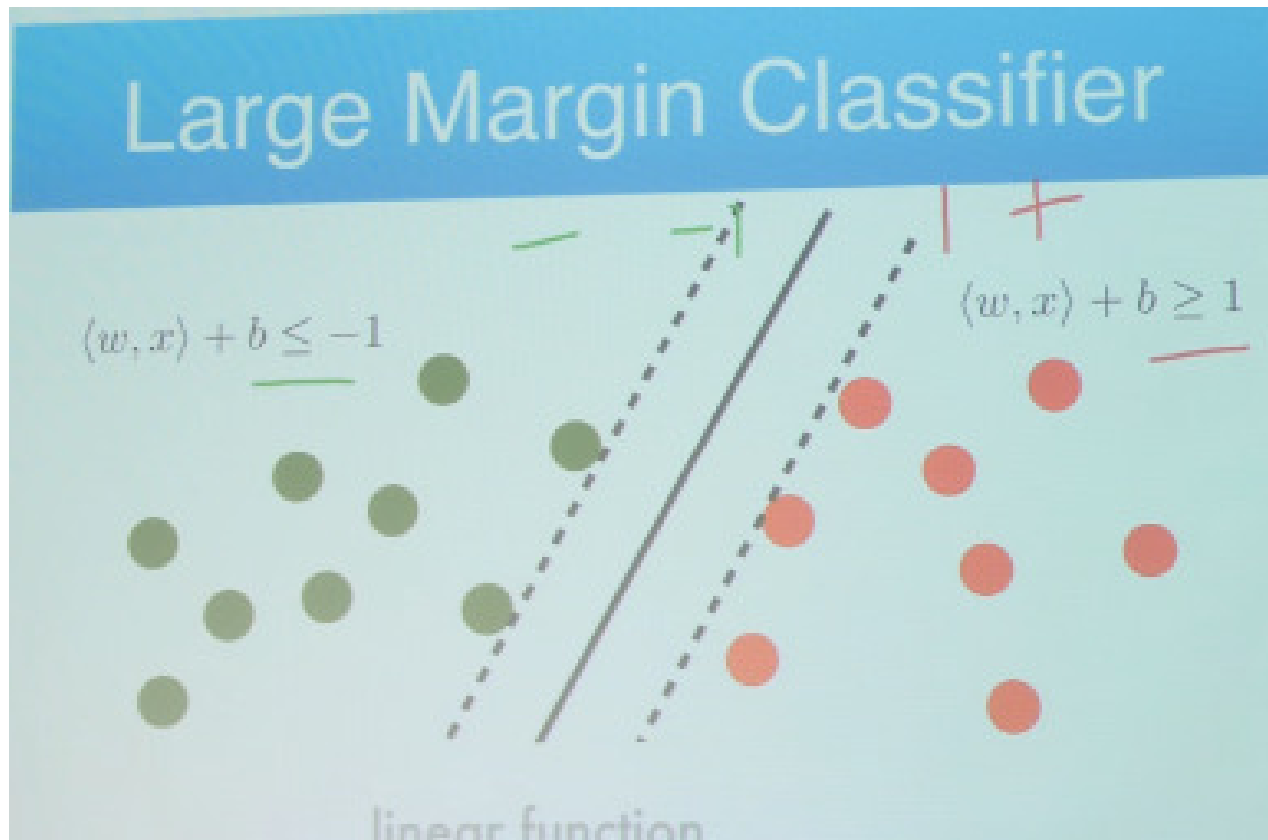
When using the dual formulation, it is computed using only the support vectors:

$$h(\mathbf{x}_i) = \text{sign}\left(\sum_{j=1}^S \alpha_j y_j (\mathbf{x}_j \cdot \mathbf{x}_i) + b\right)$$



SVMs search for the optimal hyperplane

Large Margin Classifier



$$\bar{\omega} \cdot \bar{x}_+ + b \geq 1$$

$$\bar{\omega} \cdot \bar{x}_- + b \leq -1$$

y_i SUCH THAT $y_i = +1$ FOR + SAMPLES
 $y_i = -1$ FOR - SAMPLES

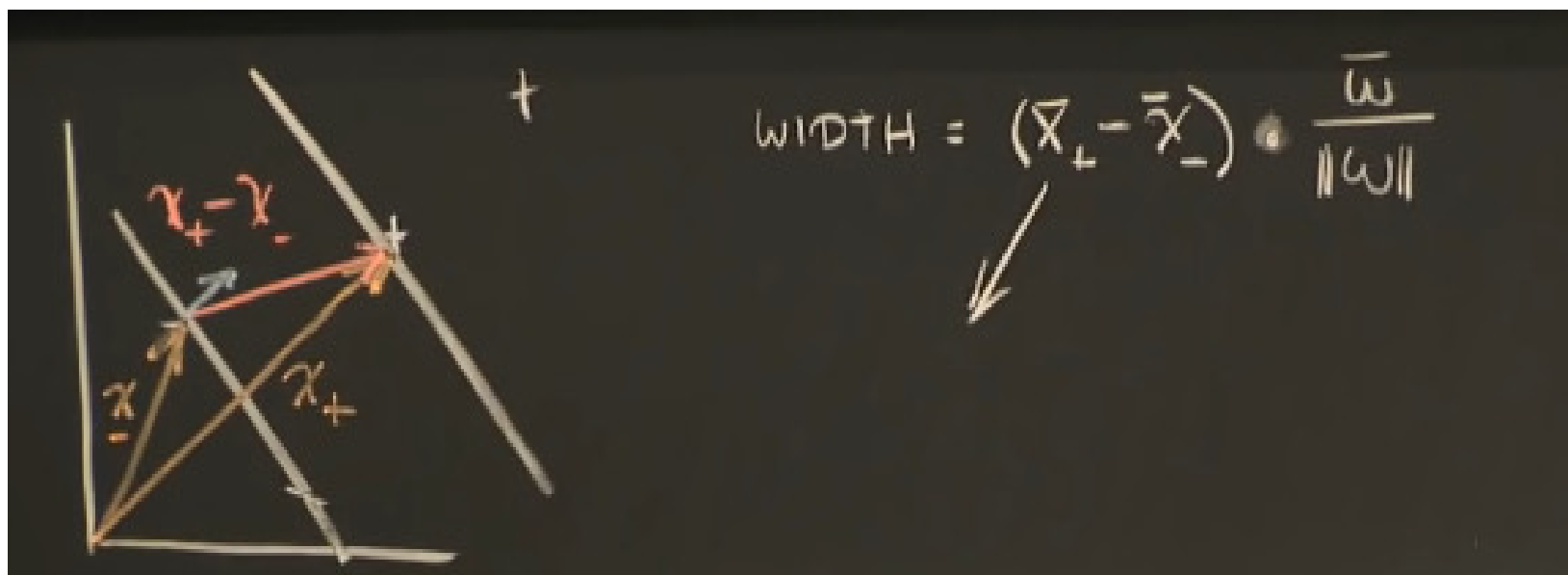
$$\bar{\omega} \cdot \bar{x}_+ + b \geq 1 \quad y_i (\bar{x}_i \bar{\omega} + b) \geq 1 \quad y_i (\bar{x}_i \bar{\omega} + b) - 1 \geq 0$$

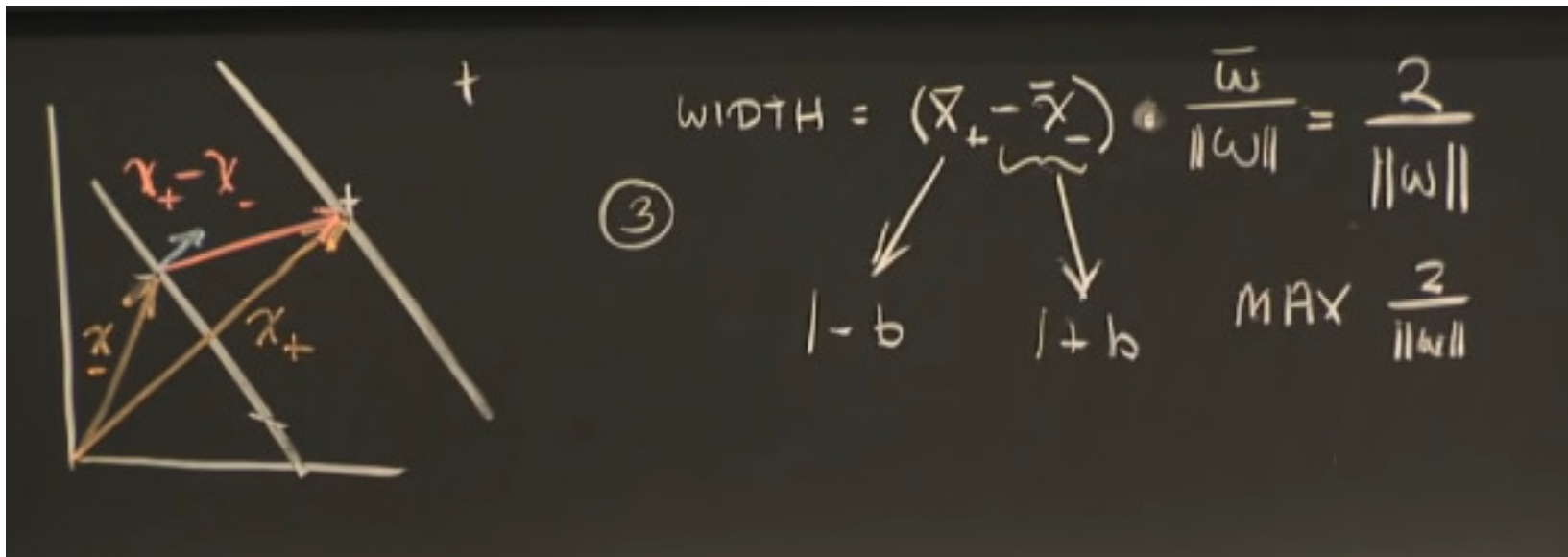
$$\bar{\omega} \cdot \bar{x}_- + b \leq -1 \quad y_i (\bar{x}_i \bar{\omega} + b) \geq 1$$

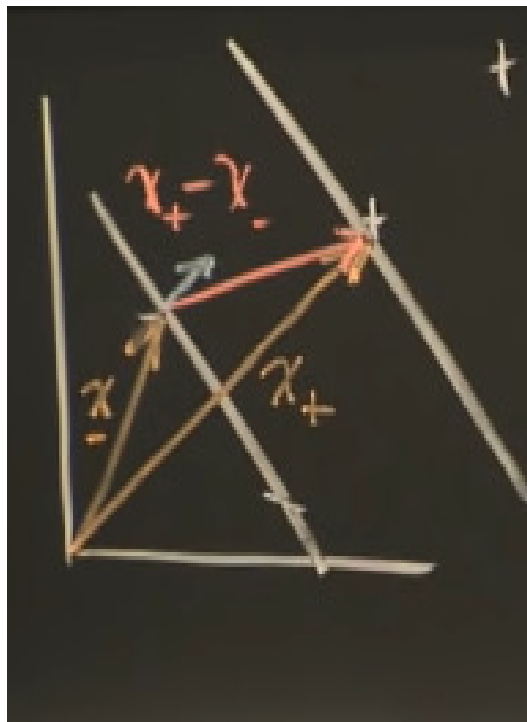
y_i SUCH THAT $y_i = +1$ FOR + SAMPLES
 $= -1$ FOR - SAMPLES

$$\boxed{y_i (\bar{x}_i \bar{\omega} + b) - 1 = 0}$$

FOR x_i IN GUTTER







③

$$\text{WIDTH} = (\bar{x}_+ - \bar{x}_-) \cdot \frac{\bar{w}}{\|\bar{w}\|} = \frac{2}{\|\bar{w}\|}$$

\swarrow \searrow
 $1-b$ $1+b$

$\text{MAX } \frac{2}{\|\bar{w}\|}$
 \swarrow
 $\text{MAX } \frac{1}{\|\bar{w}\|} \rightarrow \text{MIN } \|\bar{w}\| \rightarrow \text{MIN } \frac{1}{2} \|\bar{w}\|^2$

Optimization of a function with given constraints:

Method of **Lagrange Multipliers** will be used to transform the “given function with constraints” to another equivalent “function without constraints” that function and can be optimized (maximize or minimize).

$$L = \frac{1}{2} \|\bar{w}\|^2 - \sum \alpha_i [y_i (\bar{w} \cdot \bar{x}_i + b) - 1]$$

Below is cost function of SVM with **Lagrange Multipliers**

$$L = \frac{1}{2} \|\bar{\omega}\|^2 - \sum \alpha_i [y_i (\bar{\omega} \cdot \bar{x}_i + b) - 1]$$
$$\frac{\partial L}{\partial \bar{\omega}} = \bar{\omega} - \sum \alpha_i y_i \bar{x}_i = 0 \Rightarrow \boxed{\bar{\omega} = \sum_i \alpha_i y_i \bar{x}_i}$$
$$\frac{\partial L}{\partial b} = -\sum \alpha_i y_i = 0 \Rightarrow \boxed{\sum \alpha_i y_i = 0}$$

$$L = \frac{1}{2} \left(\sum \alpha_i y_i \bar{x}_i \right) \left(\sum \alpha_j y_j \bar{x}_j \right) - \left(\sum \alpha_i y_i x_i \right) \left(\sum \alpha_j y_j x_j \right) - \underbrace{\sum \alpha_i y_i}_0 b + \sum \alpha_i$$

$$L = \sum \alpha_i - \frac{1}{2} \sum_i \sum_j \alpha_i \alpha_j y_i y_j x_i \cdot x_j$$

Now aim to find out maximum of above L function.

The solution - quadratic programming

$$\min_{\alpha} \quad \frac{1}{2} \sum_{n=1}^N \sum_{m=1}^N y_n y_m \alpha_n \alpha_m \mathbf{x}_n^T \mathbf{x}_m - \sum_{n=1}^N \alpha_n$$

The solution - quadratic programming

$$\min_{\alpha} \quad \frac{1}{2} \alpha^T \underbrace{\begin{bmatrix} y_1 y_1 \mathbf{x}_1^T \mathbf{x}_1 & y_1 y_2 \mathbf{x}_1^T \mathbf{x}_2 & \dots & y_1 y_N \mathbf{x}_1^T \mathbf{x}_N \\ y_2 y_1 \mathbf{x}_2^T \mathbf{x}_1 & y_2 y_2 \mathbf{x}_2^T \mathbf{x}_2 & \dots & y_2 y_N \mathbf{x}_2^T \mathbf{x}_N \\ \dots & \dots & \dots & \dots \\ y_N y_1 \mathbf{x}_N^T \mathbf{x}_1 & y_N y_2 \mathbf{x}_N^T \mathbf{x}_2 & \dots & y_N y_N \mathbf{x}_N^T \mathbf{x}_N \end{bmatrix}}_{\text{quadratic coefficients}} \alpha + \underbrace{(-\mathbf{1}^T)}_{\text{linear}} \alpha$$

A QP solver is a program used to solve quadratic programming problems.

Python package called CVXOPT

The solution - quadratic programming

$$\min_{\alpha} \quad \frac{1}{2} \alpha^T \underbrace{\begin{bmatrix} y_1 y_1 \mathbf{x}_1^T \mathbf{x}_1 & y_1 y_2 \mathbf{x}_1^T \mathbf{x}_2 & \dots & y_1 y_N \mathbf{x}_1^T \mathbf{x}_N \\ y_2 y_1 \mathbf{x}_2^T \mathbf{x}_1 & y_2 y_2 \mathbf{x}_2^T \mathbf{x}_2 & \dots & y_2 y_N \mathbf{x}_2^T \mathbf{x}_N \\ \dots & \dots & \dots & \dots \\ y_N y_1 \mathbf{x}_N^T \mathbf{x}_1 & y_N y_2 \mathbf{x}_N^T \mathbf{x}_2 & \dots & y_N y_N \mathbf{x}_N^T \mathbf{x}_N \end{bmatrix}}_{\text{quadratic coefficients}} \alpha + \underbrace{(-\mathbf{1}^T)}_{\text{linear}} \alpha$$

subject to $\underbrace{\mathbf{y}^T \alpha}_{\text{linear constraint}} = 0$

$$\underbrace{0}_{\text{lower bounds}} \leq \alpha \leq \underbrace{\infty}_{\text{upper bounds}}$$

QP hands us α

Solution: $\boldsymbol{\alpha} = \alpha_1, \dots, \alpha_N$

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

KKT condition: For $n = 1, \dots, N$

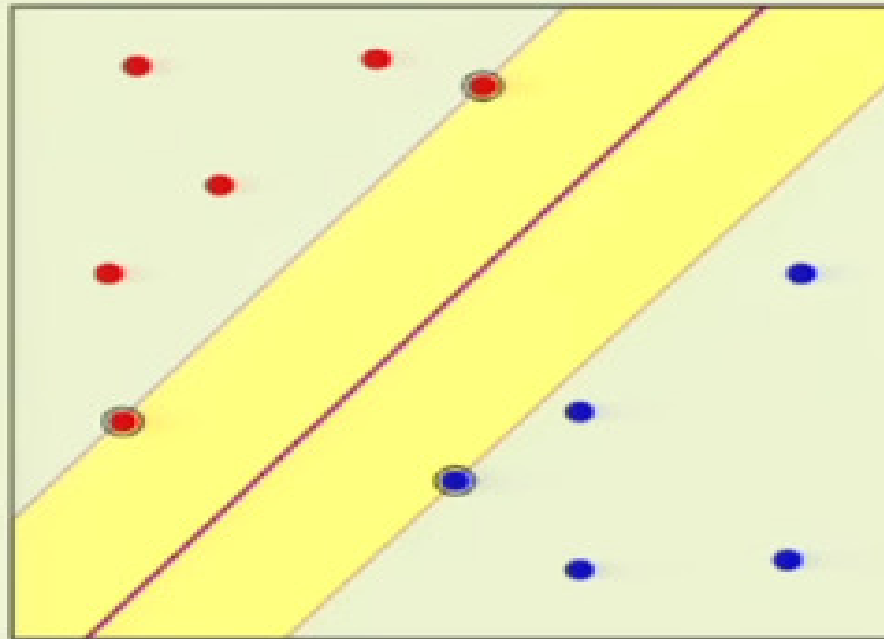
$$\alpha_n (y_n (\mathbf{w}^\top \mathbf{x}_n + b) - 1) = 0$$

We saw this before!

$\alpha_n > 0 \implies \mathbf{x}_n$ is a support vector

Support vectors

Closest \mathbf{x}_n 's to the plane: achieve the margin



Support vectors

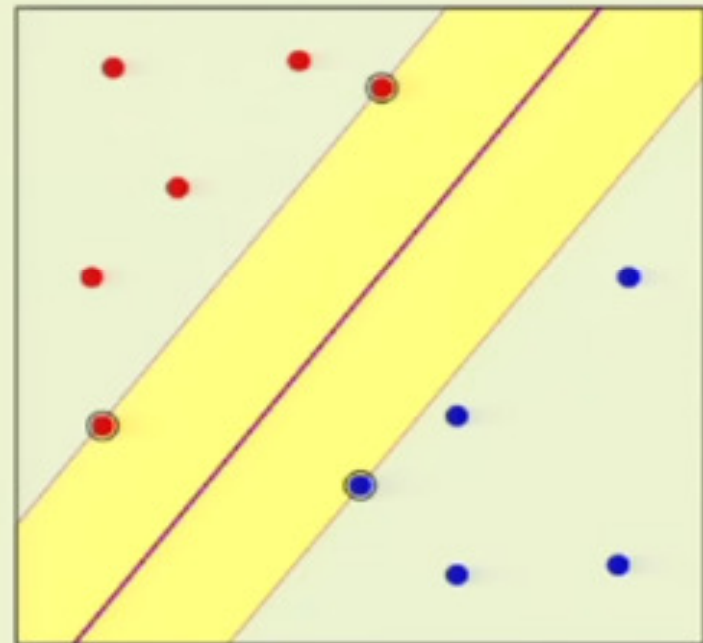
Closest \mathbf{x}_n 's to the plane: achieve the margin

$$\Rightarrow y_n (\mathbf{w}^T \mathbf{x}_n + b) = 1$$

$$\mathbf{w} = \sum_{\mathbf{x}_n \text{ is SV}} \alpha_n y_n \mathbf{x}_n$$

Solve for b using any SV:

$$y_n (\mathbf{w}^T \mathbf{x}_n + b) = 1$$



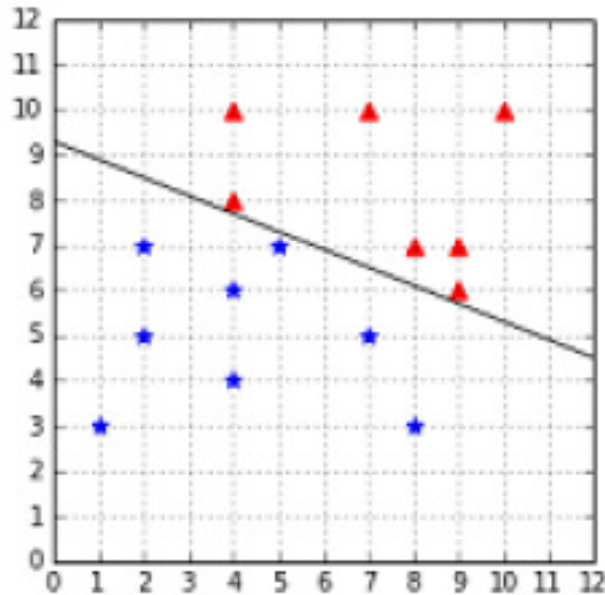
Soft Margin SVM

Dealing with noisy data

The biggest issue with hard margin SVM is that **it requires the data to be linearly separable**. Real-life data is often noisy. Even when the data is linearly separable, a lot of things can happen before you feed it to your model. Maybe someone mistyped a value for an example, or maybe the probe of a sensor returned a crazy value. In **the presence of an outlier** (a data point that seems to be out of its group), **there are two cases: the outlier can be closer to the other examples than most of the examples of its class, thus reducing the margin, or it can be among the other examples and break linear separability.**

Outlier reducing the margin

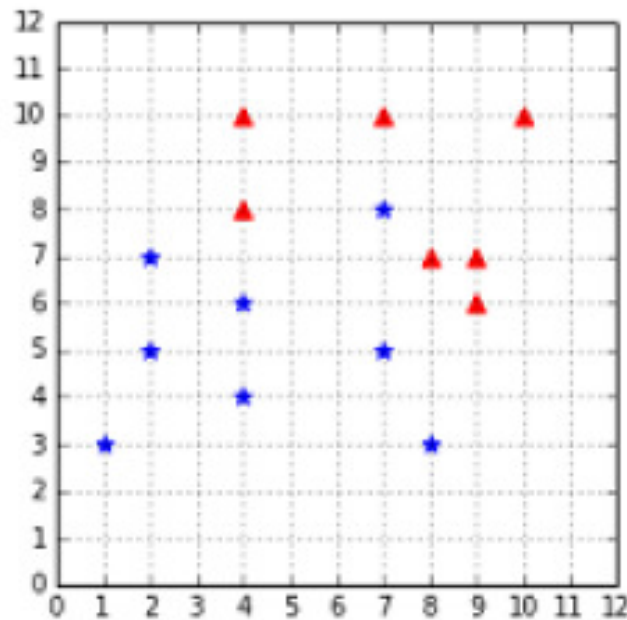
When the data is linearly separable, the hard margin classifier does not behave as we would like in the presence of outliers.



In this case, we can see that the margin is very narrow, and it seems that the outlier is the main reason for this change. Intuitively, we can see that this hyperplane might not be the best at separating the data, and that it will probably generalize poorly.

Outlier breaking linear separability

Even worse, when the outlier breaks the linear separability, the classifier is incapable of finding a hyperplane. We are stuck because of a single data point.



Soft margin to the rescue: Slack variables

The goal is now not to make zero classification mistakes, but to make as few mistakes as possible.

To do so, they modified the constraints of the optimization problem by adding a variable ζ (zeta). So the constraint:

$$y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1$$

becomes:

$$y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1 - \zeta_i$$

As a result, when minimizing the objective function, it is possible to satisfy the constraint even if the example does not meet the original constraint

The problem is that we could choose a huge value of for every example, and all the constraints will be satisfied.

To avoid this, we need to modify the objective function to penalize the choice of a big ζ_i :

$$\begin{aligned} & \underset{\mathbf{w}, b, \zeta}{\text{minimize}} && \frac{1}{2} \|\mathbf{w}\|^2 + \sum_{i=1}^m \zeta_i \\ & \text{subject to} && y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1 - \zeta_i \quad \text{for any } i = 1, \dots, m \end{aligned}$$

We take the sum of all individual ζ_i and add it to the objective function. Adding such a penalty is called **regularization**. As a result, the solution will be the hyperplane that maximizes the margin while having the smallest error possible.

There is still a little problem. With this formulation, one can easily minimize the function by using negative values of ζ_i . We add the constraint $\zeta_i \geq 0$ to prevent this. Moreover, we would like to keep some control over the soft margin. Maybe sometimes we want to use the hard margin—after all, that is why we add the parameter C , which will help us to determine how important the ζ should be (more on that later).

This leads us to the **soft margin formulation**:

$$\begin{aligned} & \underset{\mathbf{w}, b, \zeta}{\text{minimize}} && \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{i=1}^m \zeta_i \\ & \text{subject to} && y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1 - \zeta_i \\ & && \zeta_i \geq 0 \quad \text{for any } i = 1, \dots, m \end{aligned}$$

using the same technique as for the separable case, we find that we need to maximize the same Wolfe dual as before, under a slightly different constraint:

$$\begin{aligned} & \underset{\alpha}{\text{maximize}} && \sum_{i=1}^m \alpha_i - \frac{1}{2} \sum_{i=1}^m \sum_{j=1}^m \alpha_i \alpha_j y_i y_j \mathbf{x}_i \cdot \mathbf{x}_j \\ & \text{subject to} && 0 \leq \alpha_i \leq C, \text{ for any } i = 1, \dots, m \\ & && \sum_{i=1}^m \alpha_i y_i = 0 \end{aligned}$$

Here the constraint $\alpha_i \geq 0$ has been changed to become $0 \leq \alpha_i \leq C$. This constraint is often called the **box constraint** because the vector α is constrained to lie inside the box with side length C in the positive orthant. Note that an orthant is the analog n-dimensional Euclidean space of a quadrant in the plane (Cristianini & Shawe-Taylor, 2000). We will visualize the box constraint in Figure 50 in the chapter about the SMO algorithm.

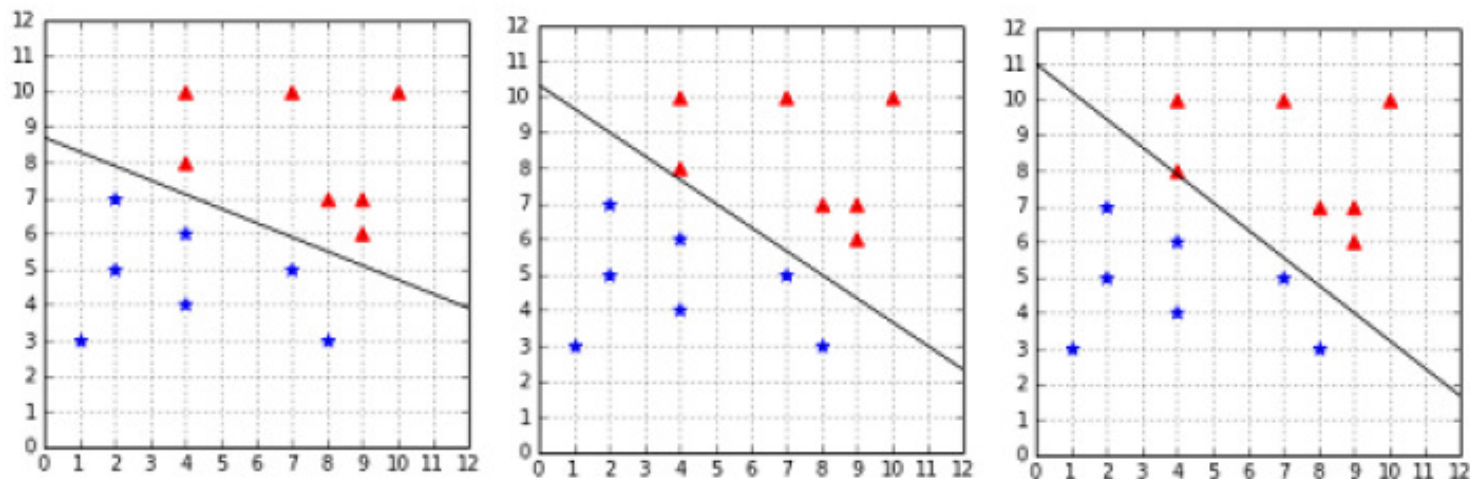
The optimization problem is also called **1-norm soft margin** because we are minimizing the 1-norm of the slack vector ζ .

Understanding what C does

The parameter C gives you control of how the SVM will handle errors. Let us now examine how changing its value will give different hyperplanes.

Figure 35 shows the linearly separable dataset we used throughout this book. On the left, we can see that setting C to $+\infty$ gives us the same result as the hard margin classifier. However, if we choose a smaller value for C like we did in the center, we can see that the hyperplane is closer to some points than others. The hard margin constraint is violated for these examples. Setting $C = 0.01$ increases this behavior as depicted on the right.

What happens if we choose a C very close to zero? Then there is basically no constraint anymore, and we end up with a hyperplane not classifying anything.



It seems that when the data is linearly separable, sticking with a big C is the best choice. But what if we have some noisy outlier? In this case, as we can see in Figure 36, using $C = +\infty$ gives us a very narrow margin. However, when we use $C = 1$, we end up with a hyperplane very close to the one of the hard margin classifier without outlier. The only violated constraint is the constraint of the outlier, and we are much more satisfied with this hyperplane. This time, setting $C = 0.01$ ends up violating the constraint of another example, which was not an outlier. This value of C seems to give too much freedom to our soft margin classifier.

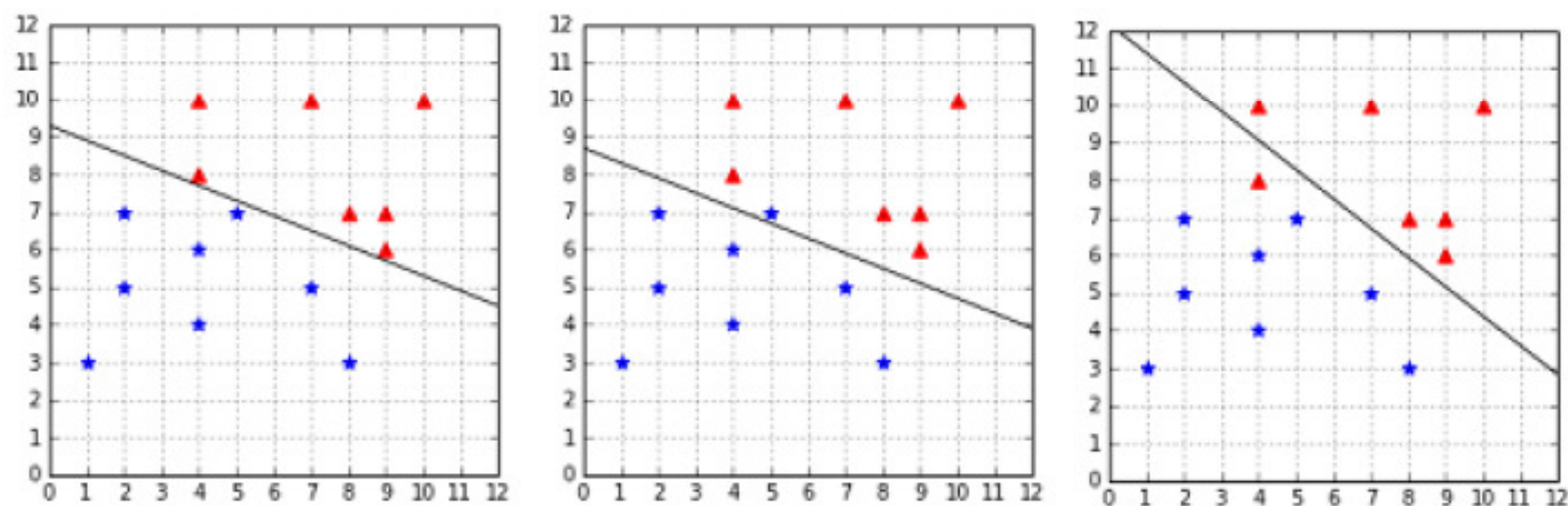


Figure 36: Effect of $C=+\text{Infinity}$, $C=1$, and $C=0.01$ on a linearly separable dataset with an outlier

Eventually, in the case where the outlier makes the data non-separable, we cannot use $C = +\infty$ because there is no solution meeting all the hard margin constraints. Instead, we test several values of C , and we see that the best hyperplane is achieved with $C = 3$. In fact, we get the same hyperplane for all values of C greater than or equal to 3. That is because no matter how hard we penalize it, it is necessary to violate the constraint of the outlier to be able to separate the data. When we use a small C , as before, more constraints are violated.

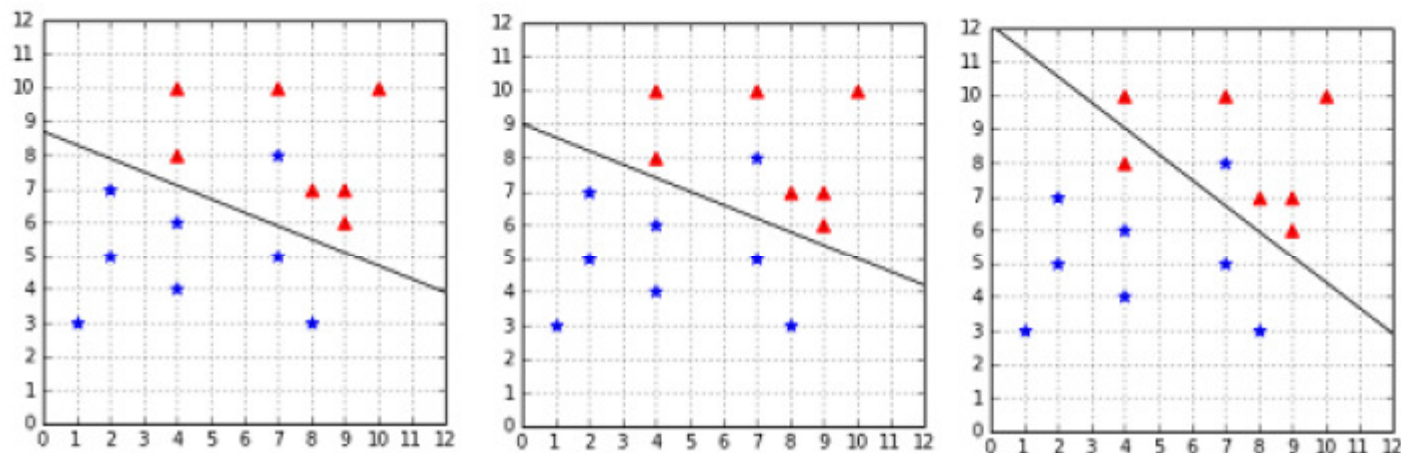


Figure 37: Effect of $C=3$, $C=1$, and $C=0.01$ on a non-separable dataset with an outlier

Rules of thumb:

- A small C will give a wider margin, at the cost of some misclassifications.
- A huge C will give the hard margin classifier and tolerates zero constraint violation.
- The key is to find the value of C such that noisy data does not impact the solution too much.

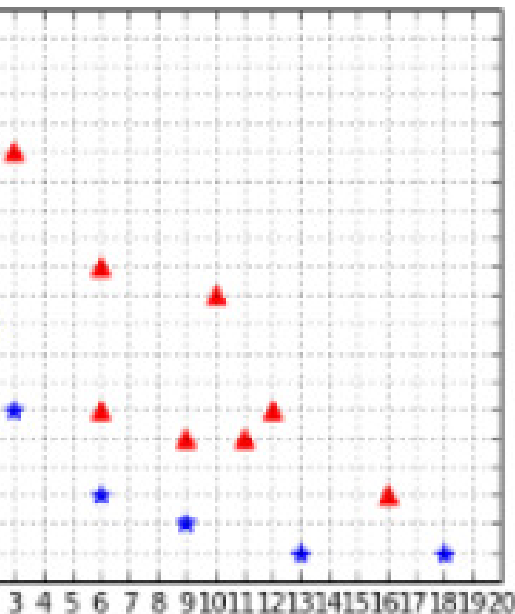
How to find the best C ?

There is no magic value for that will work for all the problems. The recommended approach to select is to use grid search with cross-validation

Can we classify non-linearly separable data?

Imagine you have some data that is not separable and you would like to use SVMs to classify it. We have seen that it is not possible because the data is not linearly separable.

However, this last assumption is not correct. What is important to notice here is that the data is not linearly separable **in two dimensions.**

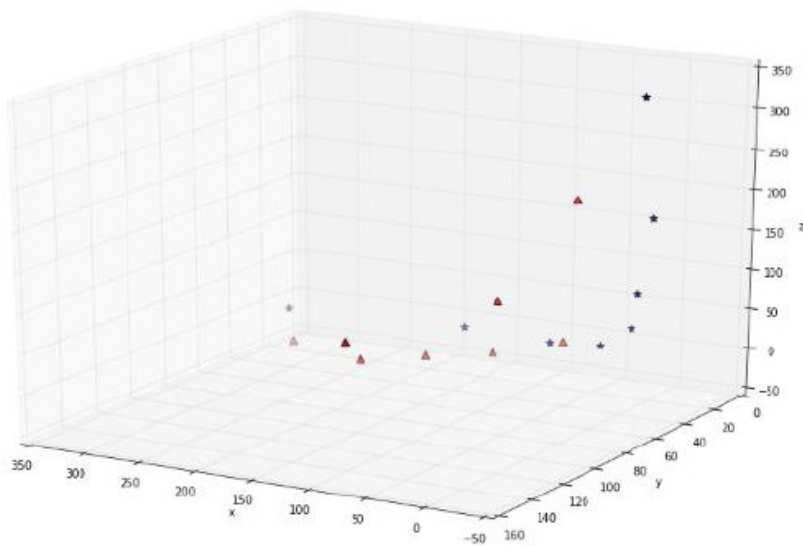


Even if your original data is in two dimensions, nothing prevents you from transforming it before feeding it into the SVM. One possible transformation would be, for instance, to transform every two-dimensional vector (x_1, x_2) into a three-dimensional vector.

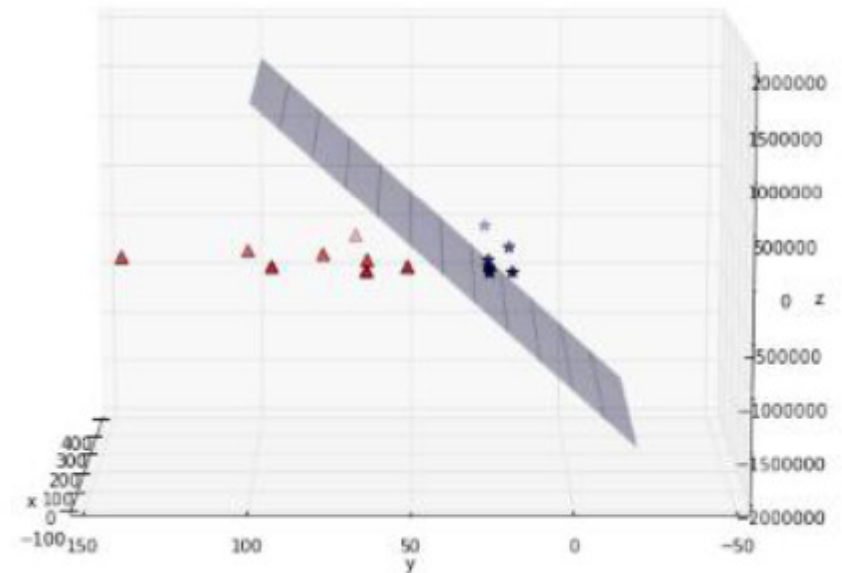
For example, we can do what is called a polynomial mapping by applying the function $\phi : \mathbb{R}^2 \rightarrow \mathbb{R}^3$ defined by:

$$\phi(x_1, x_2) = (x_1^2, \sqrt{2}x_1x_2, x_2^2)$$

If you transform the whole data set of Last Figure and plot the result after transformation you get below Figure, which does not show much improvement. However, after some playing with the graph, we can see that the data is, in fact, separable in three dimensions



The data does not look separable in three dimensions



The data is, in fact, separable by a plane

Here is a basic recipe we can use to classify this dataset:

1. Transform every two-dimensional vector into a three-dimensional vector using the `transform` method of Code Listing 31.
2. Train the SVMs using the 3D dataset.
3. For each new example we wish to predict, transform it using the **`transform` method** before passing it to the **`predict` method**.

Of course, you are not forced to transform the data into three dimensions; it could be five, ten, or one hundred dimensions.

How do we know which transformation to apply?

Choosing which transformation to apply depends a lot on your dataset. Being able to transform the data so that the machine learning algorithm you wish to use performs at its best is probably one key factor of success in the machine learning world. Unfortunately, there is no perfect recipe, and it will come with experience via trial and error.

What is a kernel?

In the last section, we saw a quick recipe to use on the non-separable dataset. One of its **main drawbacks** is that **we must transform every example**.

If we have millions or billions of examples and that transform method is complex, that can **take a huge amount of time**.

This is when kernels come to the rescue.

If you recall, when we search for the KKT multipliers in the Lagrangian function, we do not need the value of a training example ; we only need the value of the dot product between two training examples:

$$W(\alpha) = \sum_{i=1}^m \alpha_i - \frac{1}{2} \sum_{i=1}^m \sum_{j=1}^m \alpha_i \alpha_j y_i y_j \mathbf{x}_i \cdot \mathbf{x}_j$$



The question is this:

Is there a way to compute this value, without transforming the vectors?

And the answer is: Yes, with a kernel!

A kernel is a function that returns the result of a dot product performed in another space.

If we define a kernel as $K(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i \cdot \mathbf{x}_j$ we can then rewrite the soft-margin dual problem:

$$\begin{aligned} & \underset{\alpha}{\text{maximize}} && \sum_{i=1}^m \alpha_i - \frac{1}{2} \sum_{i=1}^m \sum_{j=1}^m \alpha_i \alpha_j y_i y_j K(\mathbf{x}_i, \mathbf{x}_j) \\ & \text{subject to} && 0 \leq \alpha_i \leq C, \text{ for any } i = 1, \dots, m \\ & && \sum_{i=1}^m \alpha_i y_i = 0 \end{aligned}$$

Kernel types

Linear kernel

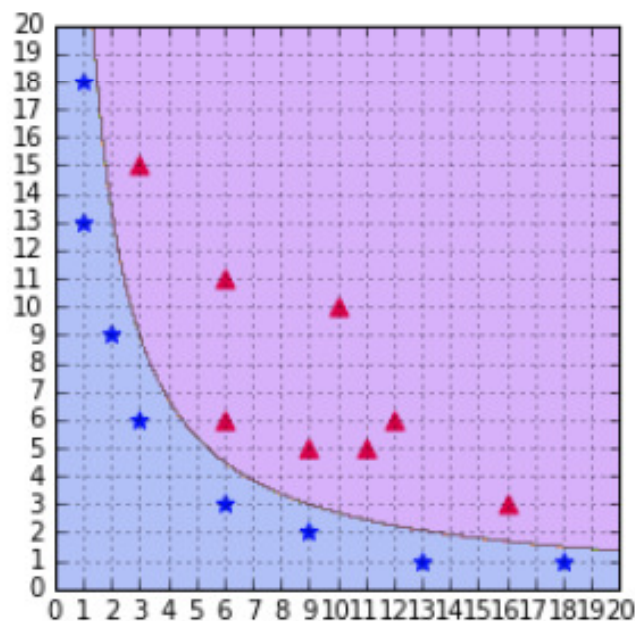
This is the simplest kernel. It is simply defined by:

$$K(\mathbf{x}, \mathbf{x}') = \mathbf{x} \cdot \mathbf{x}'$$

where \mathbf{x} and \mathbf{x}' are two vectors.

Polynomial kernel: It has two parameters which represents a constant term, and , which represents the degree of the kernel.

$$K(\mathbf{x}, \mathbf{x}') = (\mathbf{x} \cdot \mathbf{x}' + c)^d$$



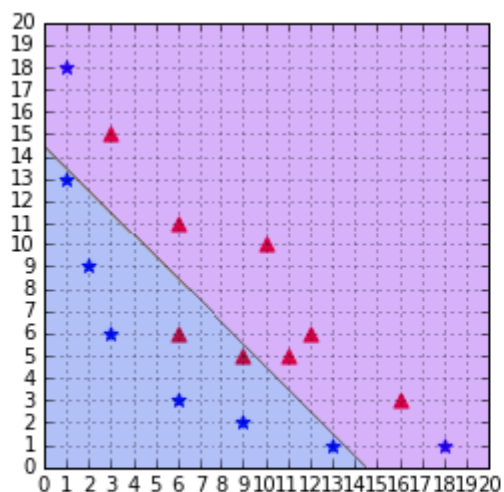
A SVM using a polynomial kernel is able to separate the data (degree=2)

Updating the degree

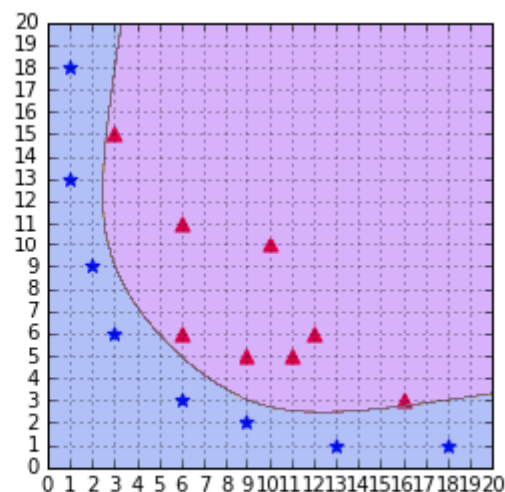
A polynomial kernel with a degree of 1 and no constant is simply the linear kernel.

When you increase the degree of a polynomial kernel, the decision boundary will become more complex and will have a tendency to be influenced by individual data examples.

Using a high-degree polynomial is dangerous because you can often achieve better performance on your test set, but it leads to what is called **overfitting: the model is too close to the data and does not generalize well.**



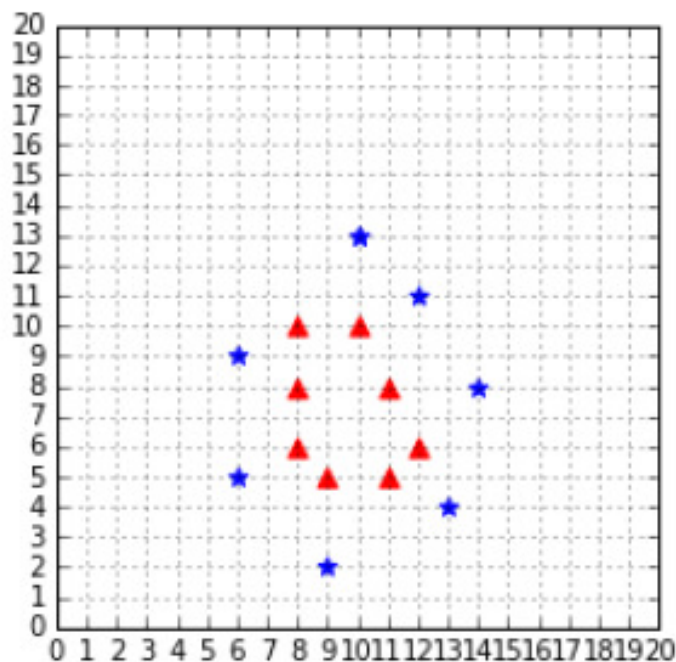
A polynomial kernel with degree = 1



A polynomial kernel with degree = 6

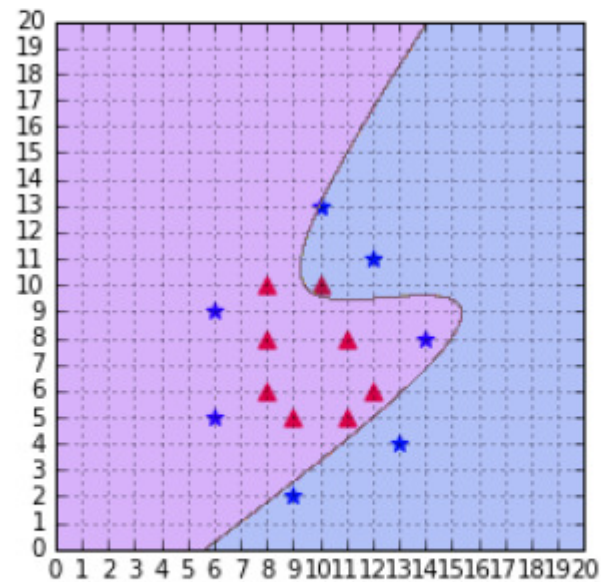
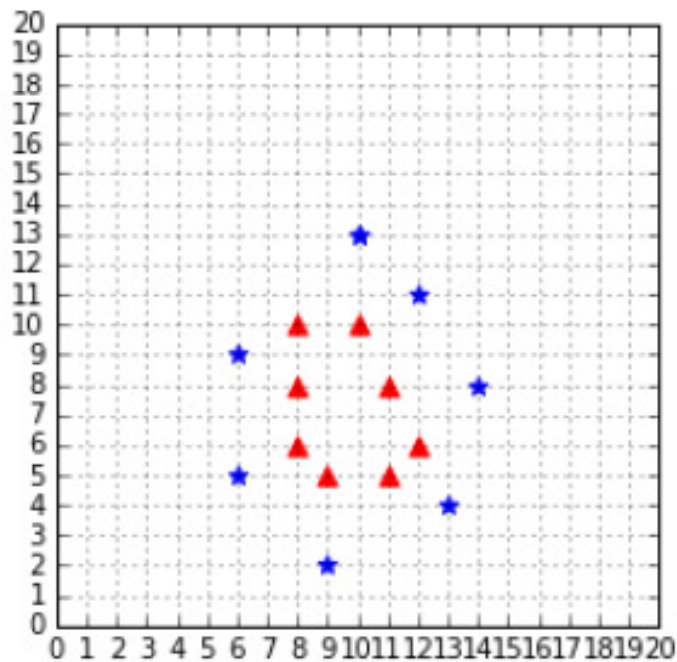
RBF or Gaussian kernel

Sometimes polynomial kernels are not sophisticated enough to work. When you have a difficult dataset like the one depicted Figure. Earlier kernel will show its limitation in these case.



RBF or Gaussian kernel

Sometimes polynomial kernels are not sophisticated enough to work. When you have a difficult dataset like the one depicted Figure. Earlier kernel will show its limitation in these case.



A polynomial kernel is not able to separate the data (degree=3, C=100)

It is also named RBF kernel, where RBF stands for Radial Basis Function. A radial basis function is a function whose value depends only on the distance from the origin or from some point.

The RBF kernel function is:

The RBF kernel function is:

$$K(\mathbf{x}, \mathbf{x}') = \exp(-\gamma \|\mathbf{x} - \mathbf{x}'\|^2)$$

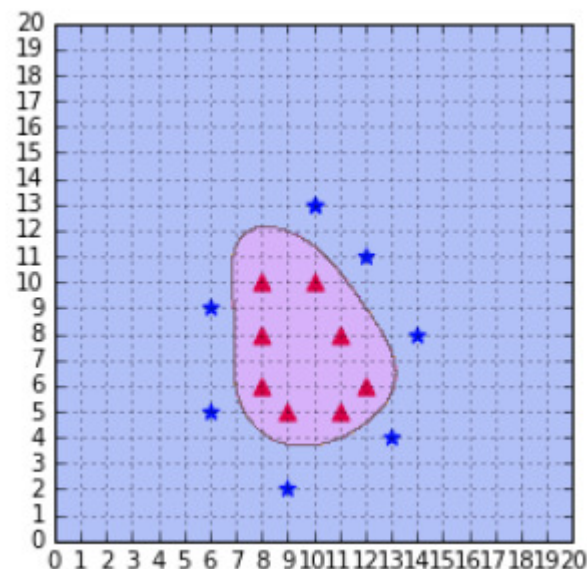
You will often read that it projects vectors into an infinite dimensional space. What does this mean?

Recall this definition: a kernel is a **function** that returns the result of a dot product performed in another space.

In the case of the polynomial kernel example we saw earlier, the kernel returned the result of a dot product performed in \mathbb{R}^3 . As it turns out, the RBF kernel returns the result of a dot product performed in \mathbb{R}^∞ .

Using RBF kernel:

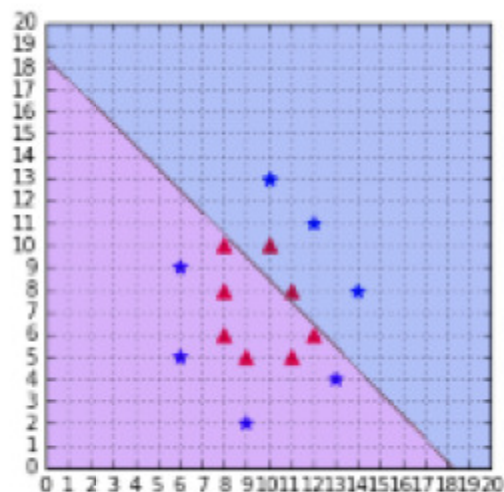
The RBF kernel classifies the data correctly with $\gamma = 0.1$



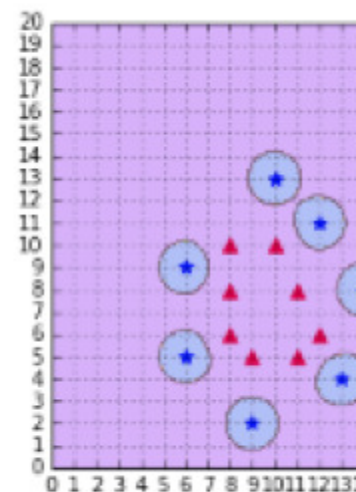
Changing the value of gamma

When gamma is too small, the model behaves like a linear SVM.

When gamma is too large, the model is too heavily influenced by each support vector.

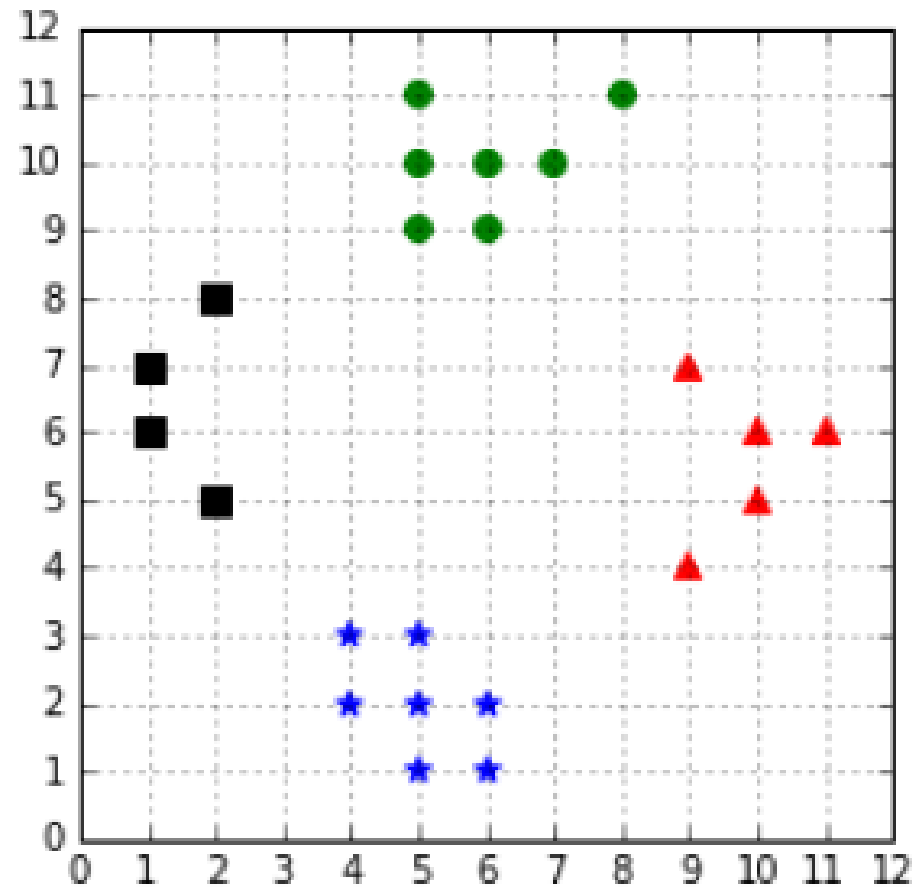


A Gaussian kernel with $\gamma = 1e-5$



A Gaussian kernel with γ too large

Multi-Class Classification Using SVMs :



A four classes classification problem

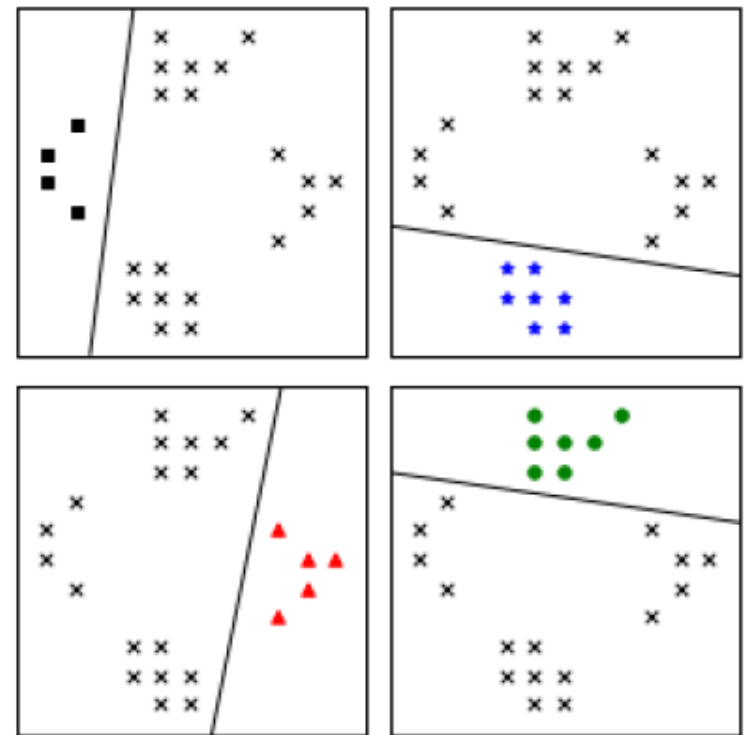
One-against-all

In order to classify K classes, we construct K different binary classifiers.

For a given class, the positive examples are all the points in the class, and the negative examples are all the points not in the class.

We train one binary classifier on each problem

As a result, we obtain one decision boundary per classifier

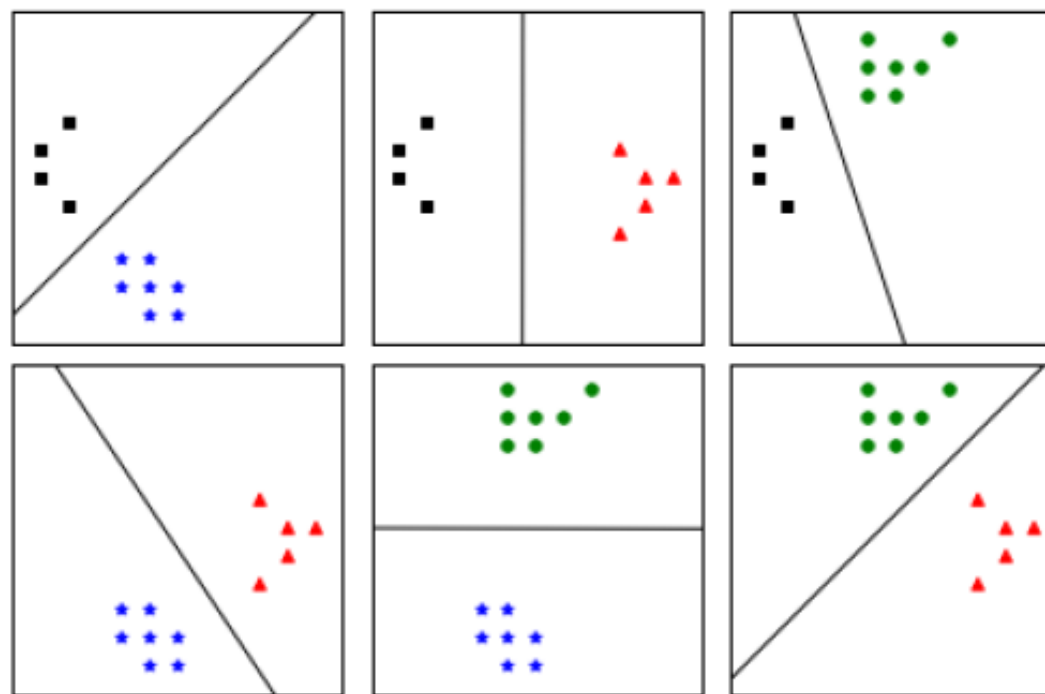


∴ The One-against-all approach creates one classifier per class

One-against-one

In this approach, instead of trying to distinguish one class from all the others, we seek to distinguish one class from another one. As a result, we train one classifier per pair of classes, which leads to $K(K-1)/2$ classifiers for K classes.

Predictions are made using a simple **voting strategy**. Each example we wish to predict is passed to each classifier, and the predicted class is recorded. Then, the class having the most votes is assigned to the example



One-against-one construct with one classifier for each pair of classes