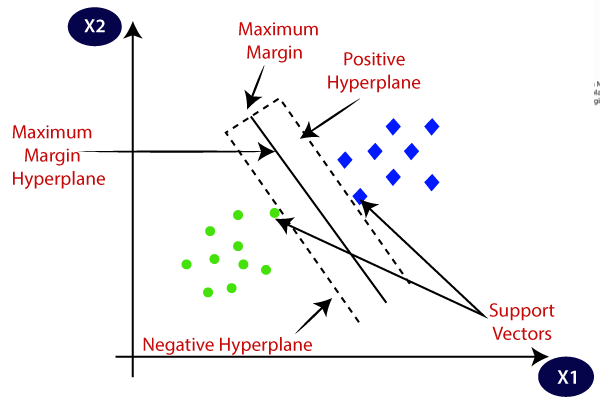
**SUPPORT VECTOR MACHINE ALGORITHM**

Support Vector Machine or SVM is one of the most popular Supervised Learning algorithms, which is used for Classification as well as Regression problems.

The goal of the SVM algorithm is to create the best line or decision boundary that can segregate n-dimensional space into classes so that we can easily put the new data point in the correct category in the future. This best decision boundary is called a hyperplane.

SVM chooses the extreme points/vectors that help in creating the hyperplane. These extreme cases are called support vectors, and hence the algorithm is termed as Support Vector Machine. Consider the below diagram in which there are two different categories that are classified using a decision boundary or hyperplane:



SVM algorithms can be used for Face detection, image classification, text categorization, etc.

**Types of Support Vector Machine Algorithm**

SVM can be of two types:

Linear SVM: Linear SVM is used for linearly separable data, which means if a dataset can be classified into two classes by using a single straight line, then such data is termed as linearly separable data, and classifier is used called as Linear SVM classifier.

Non-linear SVM: Non-Linear SVM is used for non-linearly separated data, which means if a dataset cannot be classified by using a straight line, then such data is termed as non-linear data and classifier used is called as Non-linear SVM classifier.

Hyperplane and Support Vectors in the SVM algorithm:

Hyperplane: There can be multiple lines/decision boundaries to segregate the classes in n-dimensional space, but we need to find out the best decision boundary that helps to classify the data points. This best boundary is known as the hyperplane of SVM.

The dimensions of the hyperplane depend on the features present in the dataset, which means if there are 2 features (as shown in image), then the hyperplane will be a straight line. And if there are 3 features, then the hyperplane will be a 2-dimension plane.

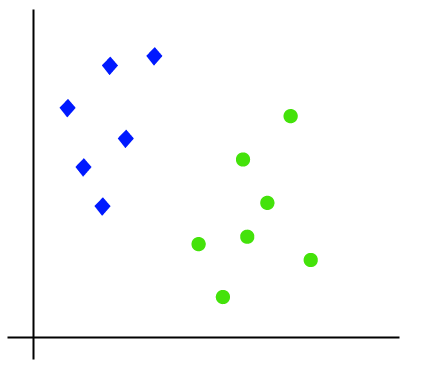
We always create a hyperplane that has a maximum margin, which means the maximum distance between the data points.

**Support Vectors:**

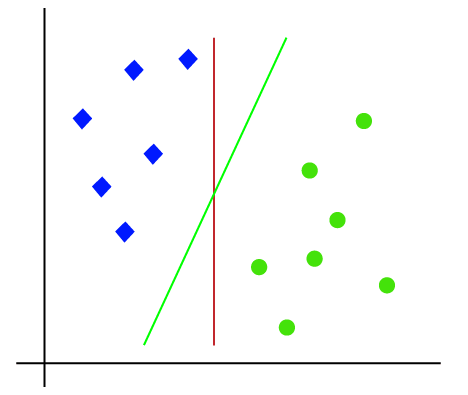
The data points or vectors that are the closest to the hyperplane and which affect the position of the hyperplane are termed as Support Vector. Since these vectors support the hyperplane, hence called a Support vector.

Linear SVM:

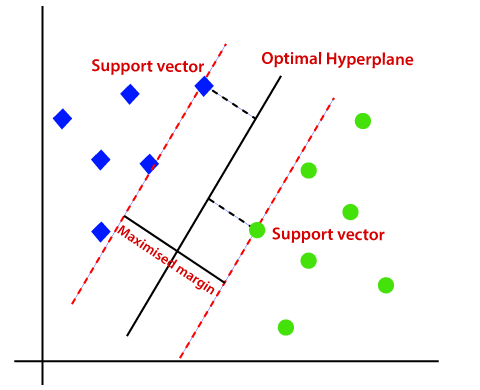
The working of the SVM algorithm can be understood by using an example. Suppose we have a dataset that has two tags (green and blue), and the dataset has two features x1 and x2. We want a classifier that can classify the pair(x1, x2) of coordinates in either green or blue. Consider the below image:



So as it is 2-d space so by just using a straight line, we can easily separate these two classes. But there can be multiple lines that can separate these classes. Consider the below image:

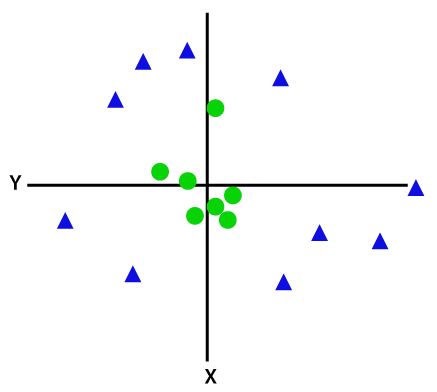


Hence, the SVM algorithm helps to find the best line or decision boundary; this best boundary or region is called as a hyperplane. SVM algorithm finds the closest point of the lines from both the classes. These points are called support vectors. The distance between the vectors and the hyperplane is called as margin. And the goal of SVM is to maximize this margin. The hyperplane with maximum margin is called the optimal hyperplane.



Non-Linear SVM:

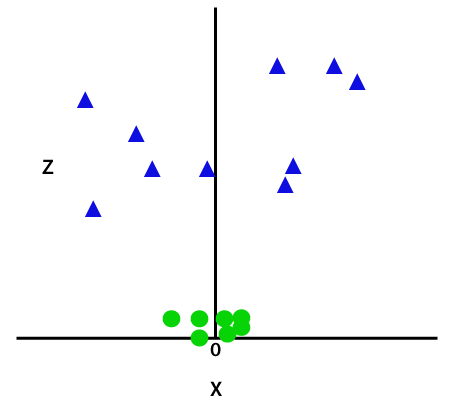
If data is linearly arranged, then we can separate it by using a straight line, but for non-linear data, we cannot draw a single straight line. Consider the below image:



So to separate these data points, we need to add one more dimension. For linear data, we have used two dimensions x and y, so for non-linear data, we will add a third dimension z. It can be calculated as:

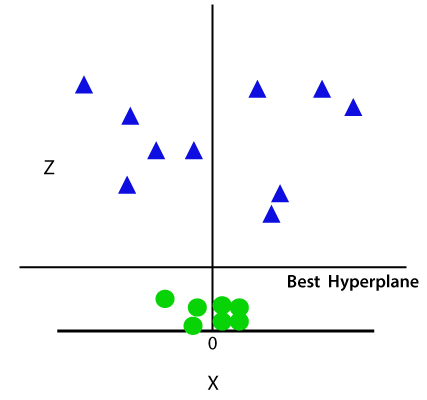
z=x2 +y2

By adding the third dimension, the sample space will become as below image:



So now, SVM will divide the datasets into classes in the following way.

Since we are in 3-d Space, hence it is looking like a plane parallel to the x-axis. If we convert it in 2d space with z=1, then it will become as:



Hence we get a circumference of radius 1 in case of non-linear data.

**MATHEMATICS BEHIND SUPPORT VECTOR MACHINE**

Equation of Line : **mx + b**

Where m indicates the slope of the line and b is the y-intercept

General form of this equation 👍

Ax1 + bx2 + c = 0

In many vectors form

W1x1 + w2x2 + w3x3+.....wnxn + w0 = 0

**Using dot product**

W . x = 0

In Vector Form

w = [w1,

w2,

w3,

.

.

.

wn]

x = [x1,

x2,

x3,

.

.

.

xn]

Not Multiplication

[w1,w2,w3,w4....+wn][x1,

x2,

x3,

.

.

.

Xn]

**Final Equation**

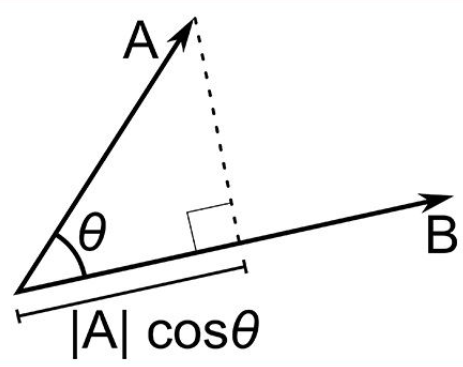
**wTx + w0 = 0**

If Consider

W0 = 0

Then eq

wTx = 0



If wTx = 0 and θ = 90

|W| |X| cosθ

#### 

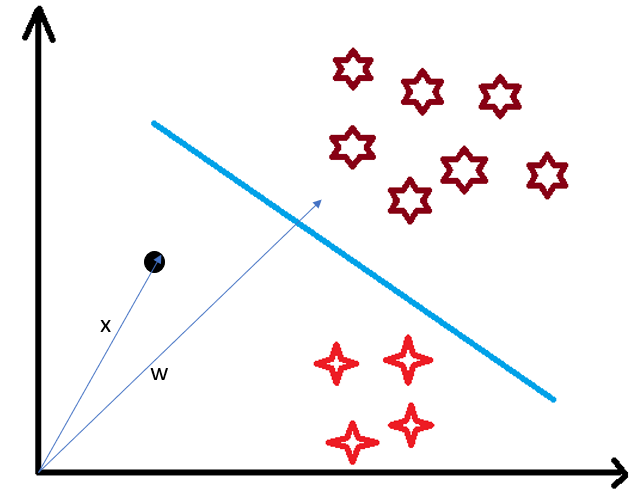
#### 

#### 

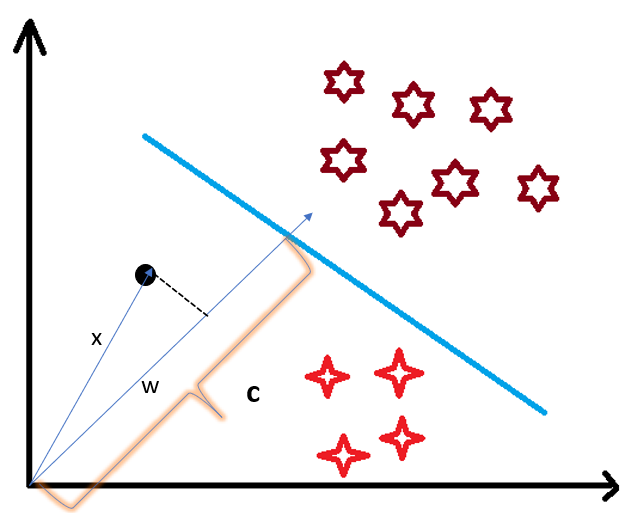
#### 

#### **Use of Dot Product in SVM**

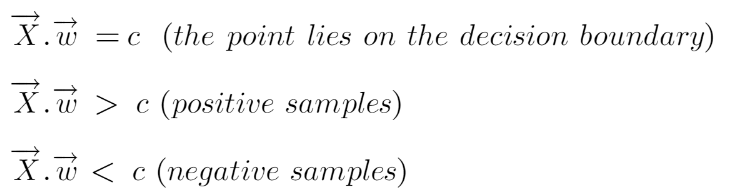
Consider a random point X and we want to know whether it lies on the right side of the plane or the left side of the plane (positive or negative).



To find this first we assume this point is a vector (X) and then we make a vector (w) which is perpendicular to the hyperplane. Let’s say the distance of vector w from origin to decision boundary is ‘c’. Now we take the projection of X vector on w.



We already know that projection of any vector or another vector is called dot-product. Hence, we take the dot product of x and w vectors. If the dot product is greater than ‘c’ then we can say that the point lies on the right side. If the dot product is less than ‘c’ then the point is on the left side and if the dot product is equal to ‘c’ then the point lies on the decision boundary.

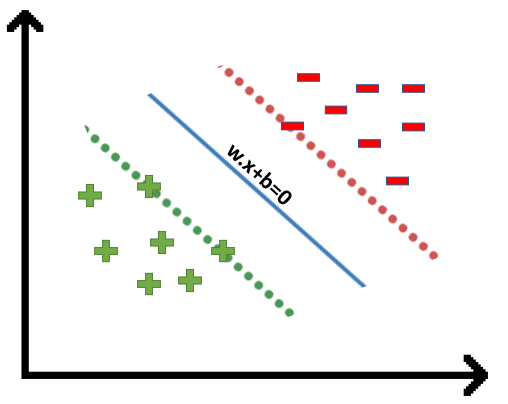


You must be having this doubt that why did we take this perpendicular vector w to the hyperplane? So what we want is the distance of vector X from the decision boundary and there can be infinite points on the boundary to measure the distance from. So that’s why we come to standard, we simply take perpendicular and use it as a reference and then take projections of all the other data points on this perpendicular vector and then compare the distance.

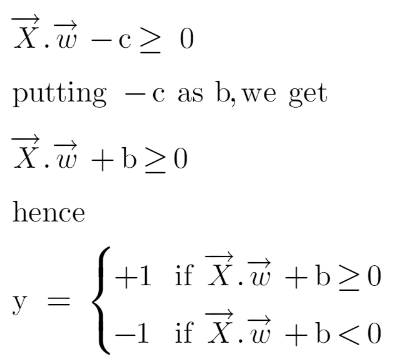
In SVM we also have a concept of margin. In the next section, we will see how we find the equation of a hyperplane and what exactly do we need to optimize in SVM.

**Margin in Support Vector Machine**

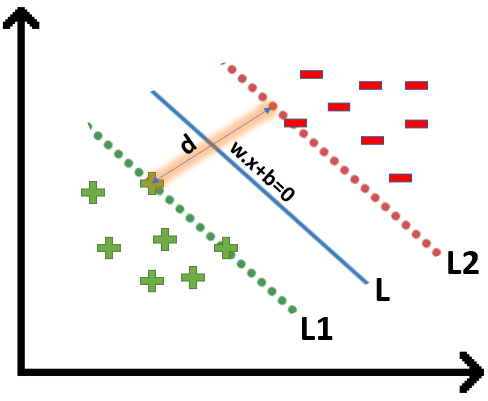
We all know the equation of a hyperplane is w.x+b=0 where w is a vector normal to hyperplane and b is an offset.



To classify a point as negative or positive we need to define a decision rule. We can define decision rule as:



If the value of w.x+b>0 then we can say it is a positive point otherwise it is a negative point. Now we need (w,b) such that the margin has a maximum distance. Let’s say this distance is ‘d’.



To calculate ‘d’ we need the equation of L1 and L2. For this, we will take few assumptions that the equation of L1 is w.x+b=1 and for L2 it is w.x+b=-1.

**Now the question comes**

Why the magnitude is equal, why didn’t we take 1 and -2?

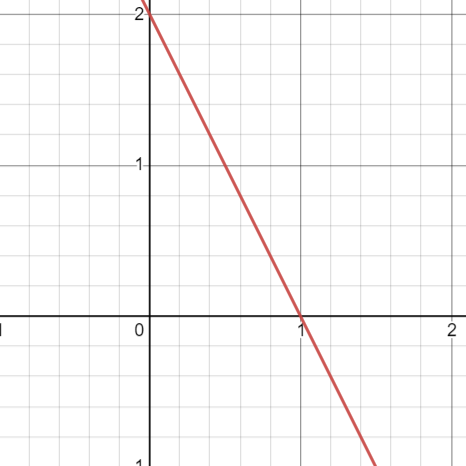
Why did we only take 1 and -1, why not any other value like 24 and -100?

Why did we assume this line?

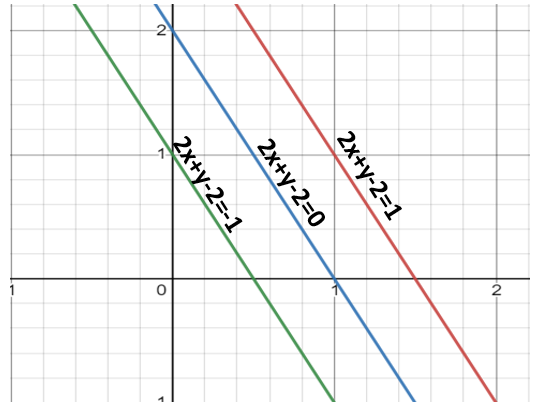
**Let’s try to answer these questions**

* We want our plane to have equal distance from both the classes that means L should pass through the center of L1 and L2 that’s why we take magnitude equal.
* Let’s say the equation of our hyperplane is 2x+y=2, we observe that even if we multiply the whole equation with some other number the line doesn’t change (try plotting on a graph). Hence for mathematical convenience, we take it as 1.
* Now the main question is exactly why there’s a need to assume only this line? To answer this, I’ll try to take the help of graphs.

Suppose the equation of our hyperplane is 2x+y=2:



Let’s create margin for this hyperplane,



If you multiply these equations by 10, we will see that the parallel line (red and green) gets closer to our hyperplane

We also observe that if we divide this equation by 10 then these parallel lines get bigger.

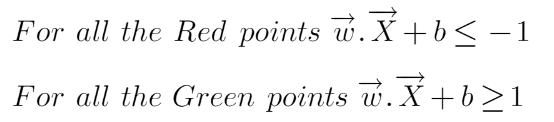
By this I wanted to show you that the parallel lines depend on (w,b) of our hyperplane, if we multiply the equation of hyperplane with a factor greater than 1 then the parallel lines will shrink and if we multiply with a factor less than 1, they expand.

We can now say that these lines will move as we do changes in (w,b) and this is how this gets optimized. But what is the optimization function? Let’s calculate it.

We know that the aim of SVM is to maximize this margin that means distance (d). But there are few constraints for this distance (d). Let’s look at what these constraints are.

**Optimization Function and its Constraints**

In order to get our optimization function, there are few constraints to consider. That constraint is that “We’ll calculate the distance (d) in such a way that no positive or negative point can cross the margin line”. Let’s write these constraints mathematically:



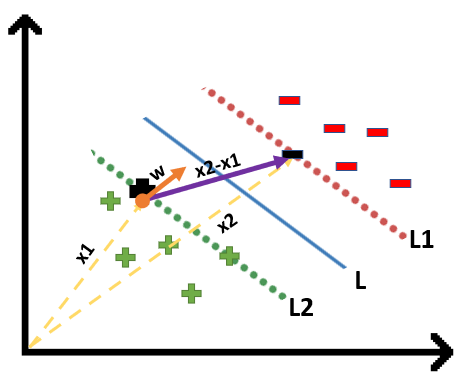
Rather than taking 2 constraints forward, we’ll now try to simplify these two constraints into 1. We assume that negative classes have *y=-1* and positive classes have *y=1.*

We can say that for every point to be correctly classified this condition should always be true:



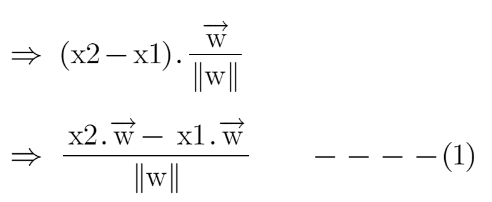
Suppose a green point is correctly classified that means it will follow w.x+b>=1, if we multiply this with y=1 we get this same equation mentioned above. Similarly, if we do this with a red point with y=-1 we will again get this equation. Hence, we can say that we need to maximize (d) such that this constraint holds true.

We will take 2 support vectors, 1 from the negative class and 2nd from the positive class. The distance between these two vectors x1 and x2 will be (x2-x1) vector. What we need is, the shortest distance between these two points which can be found using a trick we used in the dot product. We take a vector ‘w’ perpendicular to the hyperplane and then find the projection of (x2-x1) vector on ‘w’. Note: this perpendicular vector should be a unit vector then only this will work.

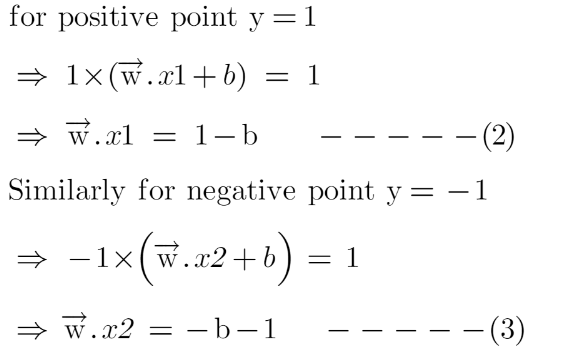


#### **Finding Projection of a Vector on Another Vector Using Dot Product**

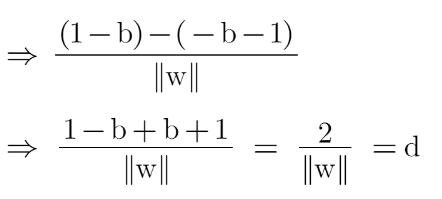
To find the projection of a vector on another vector. We do this by dot-product of both vectors. So let’s see how



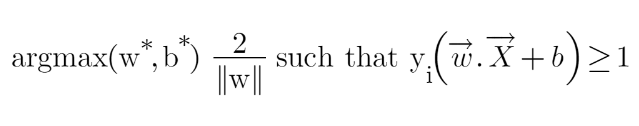
Since x2 and x1 are support vectors and they lie on the hyperplane, hence they will follow yi\* (2.x+b)=1 so we can write it as:



Putting equations (2) and (3) in equation (1) we get:



Hence the equation which we have to maximize is:



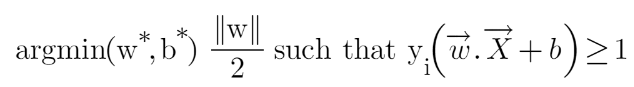
We have now found our optimization function but there is a catch here that we don’t find this type of perfectly linearly separable data in the industry, there is hardly any case we get this type of data and hence we fail to use this condition we proved here. The type of problem which we just studied is called Hard Margin SVM now we shall study soft margin which is similar to this but there are few more interesting tricks we use in Soft Margin SVM.

**Soft Margin SVM**

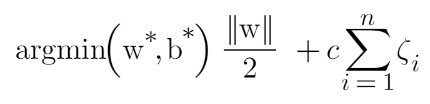
In real-life applications, we rarely encounter datasets that are perfectly linearly separable. Instead, we often come across datasets that are either nearly linearly separable or entirely non-linearly separable. Unfortunately, the trick demonstrated above for linearly separable datasets is not applicable in these cases. This is where Support Vector Machines (SVM) come into play. These are a powerful tool in machine learning that can effectively handle both almost linearly separable and non-linearly separable datasets, providing a robust solution to classification problems in diverse real-world scenarios.

To tackle this problem what we do is modify that equation in such a way that it allows few misclassifications that means it allows few points to be wrongly classified.

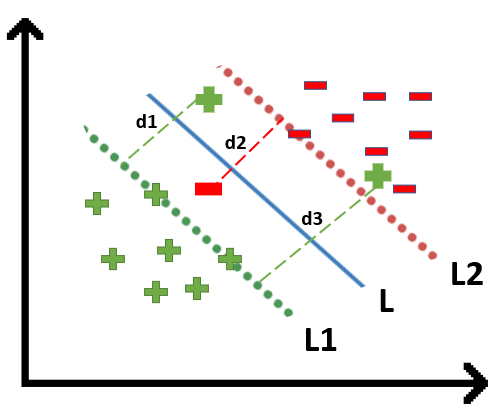
We know that max[f(x)] can also be written as min[1/f(x)], it is common practice to minimize a cost function for optimization problems; therefore, we can invert the function.



To make a soft margin equation we add 2 more terms to this equation which is zeta and multiply that by a hyperparameter ‘c’



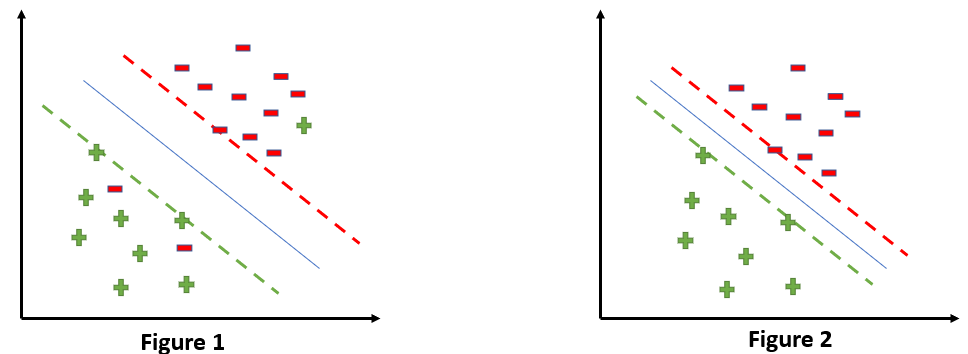
For all the correctly classified points our zeta will be equal to 0 and for all the incorrectly classified points the zeta is simply the distance of that particular point from its correct hyperplane that means if we see the wrongly classified green points the value of zeta will be the distance of these points from L1 hyperplane and for wrongly classified redpoint zeta will be the distance of that point from L2 hyperplane.



So now we can say that our that are SVM Error = Margin Error + Classification Error. The higher the margin, the lower would-be margin error, and vice versa.

Let’s say you take a high value of ‘c’ =1000, this would mean that you don’t want to focus on margin error and just want a model which doesn’t misclassify any data point.

Look at the figure below:

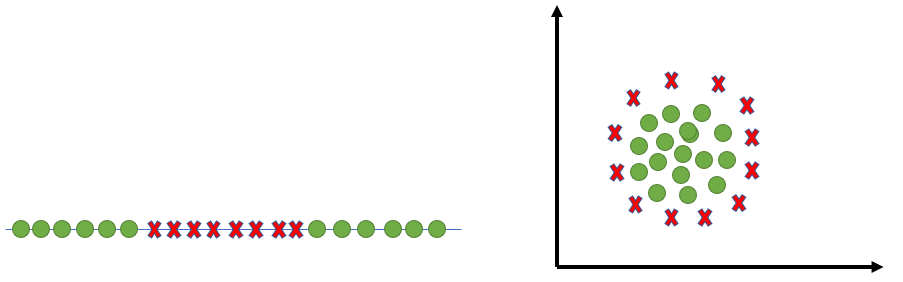


If someone asks you which is a better model, the one where the margin is maximum and has 2 misclassified points or the one where the margin is very less, and all the points are correctly classified?

Well, there’s no correct answer to this question, but rather we can use SVM Error = Margin Error + Classification Error to justify this. If you don’t want any misclassification in the model then you can choose figure 2. That means we’ll increase ‘c’ to decrease Classification Error but if you want that your margin should be maximized then the value of ‘c’ should be minimized. That’s why ‘c’ is a hyperparameter and we find the optimal value of ‘c’ using GridsearchCV and cross-validation.

Kernels in Support Vector Machine

SVM is that it can even work with a non-linear dataset and for this, we use “Kernel Trick” which makes it easier to classifies the points. Suppose we have a dataset like this:



Here we see we cannot draw a single line or say hyperplane which can classify the points correctly. So what we do is try converting this lower dimension space to a higher dimension space using some quadratic functions which will allow us to find a decision boundary that clearly divides the data points. These functions which help us do this are called Kernels and which kernel to use is purely determined by hyperparameter tuning.



**Different Kernel Functions**

Some kernel functions which you can use in SVM are given below:

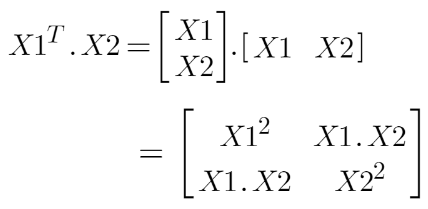
**1. Polynomial Kernel**

Following is the formula for the polynomial kernel:

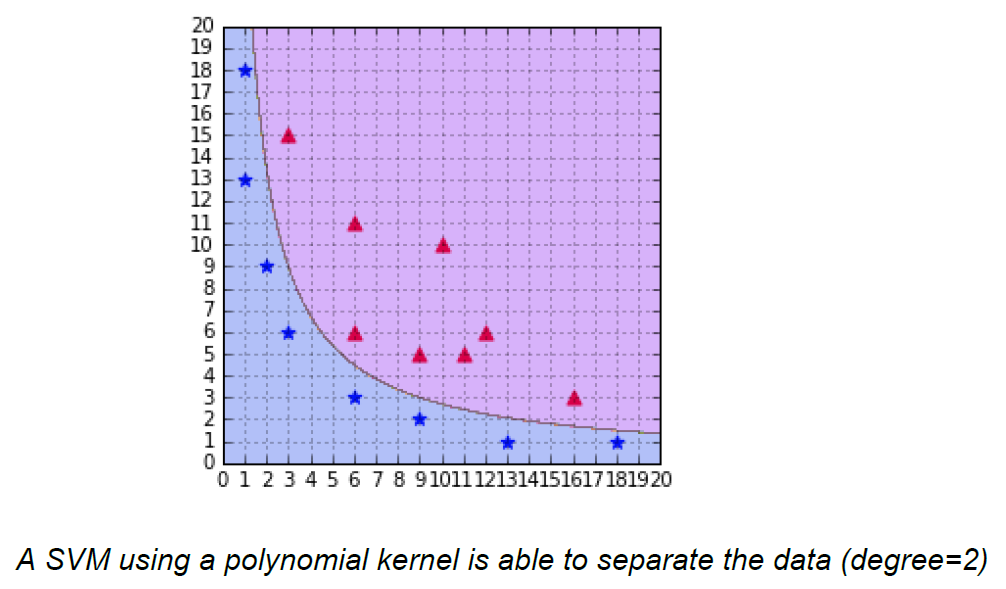


Here d is the degree of the polynomial, which we need to specify manually.

Suppose we have two features X1 and X2 and output variable as Y, so using polynomial kernel we can write it as:

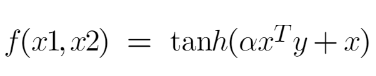


So we basically need to find X12 , X22 and X1.X2, and now we can see that 2 dimensions got converted into 5 dimensions.

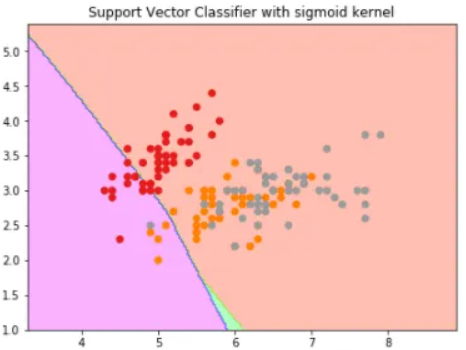


#### **2. Sigmoid Kernel**

We can use it as the proxy for neural networks. Equation is:

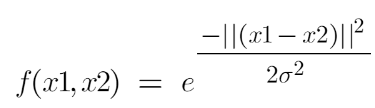


It is just taking your input, mapping them to a value of 0 and 1 so that they can be separated by a simple straight line.



**3. RBF Kernel**

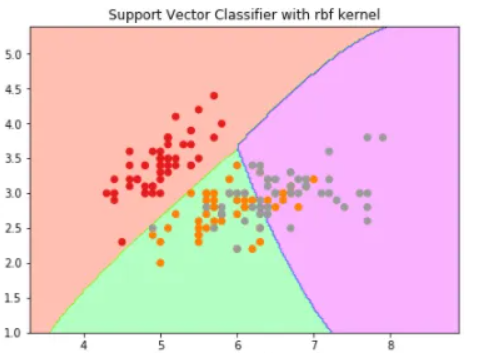
What it actually does is to create non-linear combinations of our features to lift your samples onto a higher-dimensional feature space where we can use a linear decision boundary to separate your classes It is the most used kernel in SVM classifications, the following formula explains it mathematically:



where,

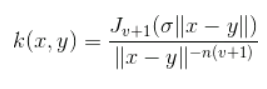
1. ‘σ’ is the variance and our hyperparameter

2. ||X₁ – X₂|| is the Euclidean Distance between two points X₁ and X₂



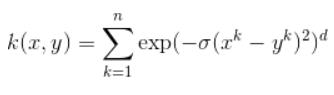
**4. Bessel function kernel**

It is mainly used for eliminating the cross term in mathematical functions. Following is the formula of the Bessel function kernel:



**5. Anova Kernel**

It performs well on multidimensional regression problems. The formula for this kernel function is:



**Advantages of SVM**

SVM works better when the data is Linear

It is more effective in high dimensions

With the help of the kernel trick, we can solve any complex problem

SVM is not sensitive to outliers

Can help us with Image classification

**Disadvantages of SVM**

Choosing a good kernel is not easy

It doesn’t show good results on a big dataset

The SVM hyperparameters are Cost -C and gamma. It is not that easy to fine-tune these hyper-parameters. It is hard to visualize their impact

**Difference between Dot Product and Cross Product**

