Support Vector Machine(SVM)

It is a supervised machine learning problem where we try to find a hyperplane that best separates the two classes.

Note: Don’t get confused between SVM and logistic regression. Both the algorithms try to find the best hyperplane, but the main difference is logistic regression is a probabilistic approach whereas support vector machine is based on statistical approaches.

Now the question is which hyperplane does it select? There can be an infinite number of hyperplanes passing through a point and classifying the two classes perfectly. So, which one is the best?

Well, SVM does this by finding the maximum margin between the hyperplanes that means maximum distances between the two classes.

Logistic Regression vs Support Vector Machine (SVM)

Depending on the number of features you have you can either choose Logistic Regression or SVM.

SVM works best when the dataset is small and complex. It is usually advisable to first use logistic regression and see how does it performs, if it fails to give a good accuracy you can go for SVM without any kernel (will talk more about kernels in the later section). Logistic regression and SVM without any kernel have similar performance but depending on your features, one may be more efficient than the other.

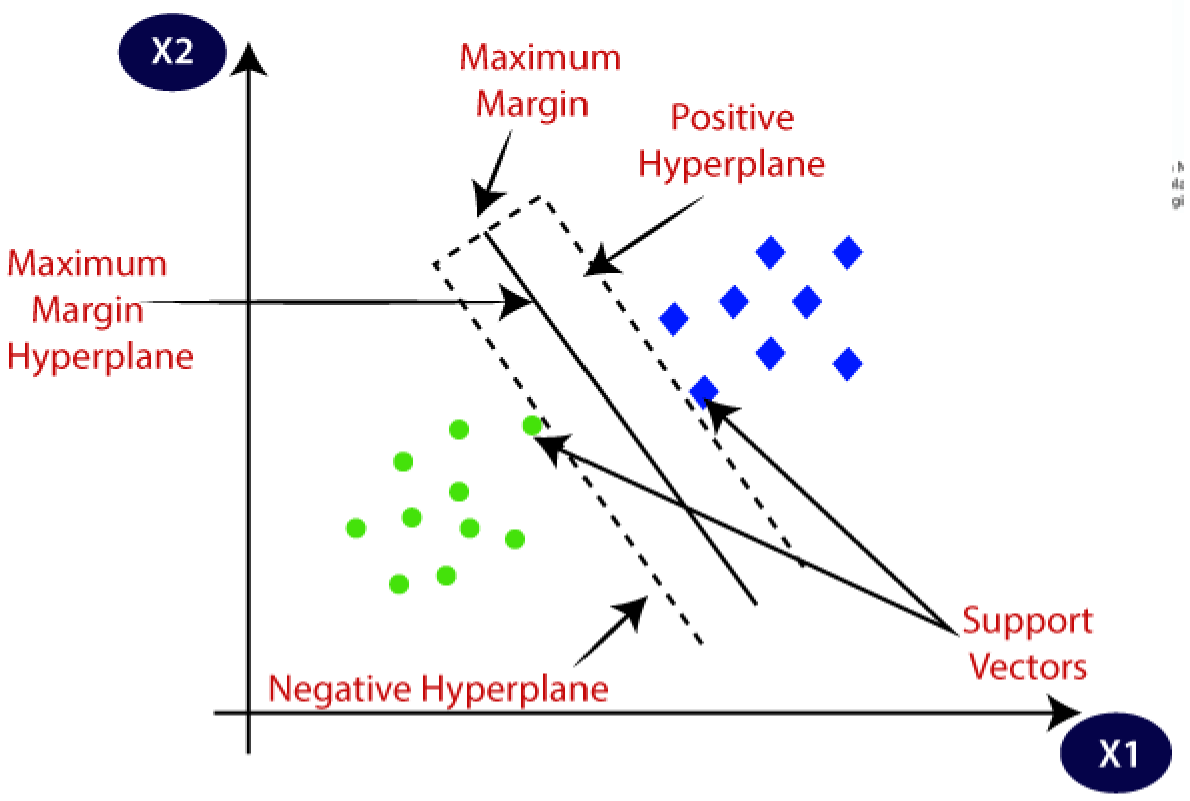
Types of Support Vector Machine (SVM) Algorithms

- **Linear SVM**: When the data is perfectly linearly separable only then we can use Linear SVM. Perfectly linearly separable means that the data points can be classified into 2 classes by using a single straight line(if 2D).
- **Non-Linear SVM**: When the data is not linearly separable then we can use Non-Linear SVM, which means when the data points cannot be separated into 2 classes by using a straight line (if 2D) then we use some advanced techniques like kernel tricks to classify them. In most real-world applications we do not find linearly separable datapoints hence we use kernel trick to solve them.

Important Terms

Now let’s define two main terms which will be repeated again and again in this article:

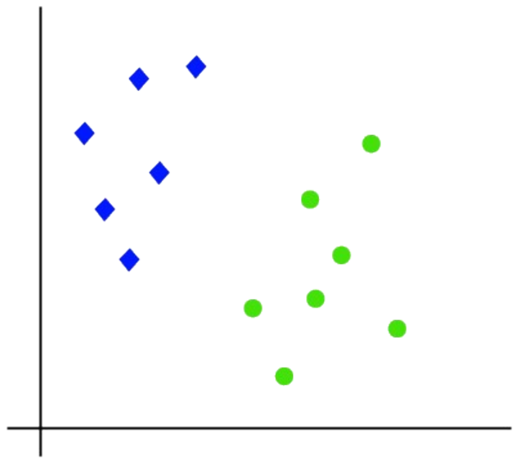
- **Support Vectors:** These are the points that are closest to the hyperplane. A separating line will be defined with the help of these data points.
- **Margin:** it is the distance between the hyperplane and the observations closest to the hyperplane (support vectors). In SVM large margin is considered a good margin. There are two types of margins **hard margin** and **soft margin.**



How Does Support Vector Machine Work?

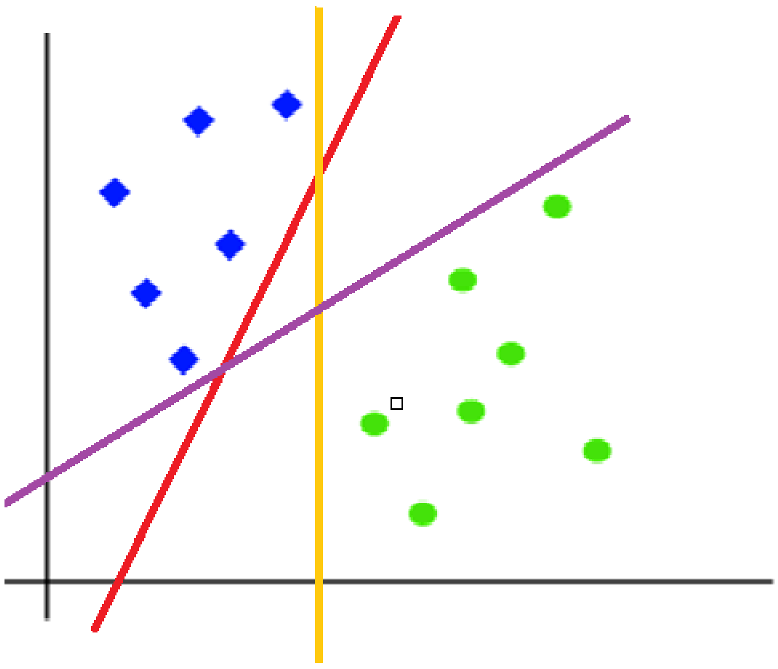
SVM is defined such that it is defined in terms of the support vectors only, we don’t have to worry about other observations since the margin is made using the points which are closest to the hyperplane (support vectors), whereas in logistic regression the classifier is defined over all the points. Hence SVM enjoys some natural speed-ups.

Let’s understand the working of SVM using an example. Suppose we have a dataset that has two classes (green and blue). We want to classify that the new data point as either blue or green.

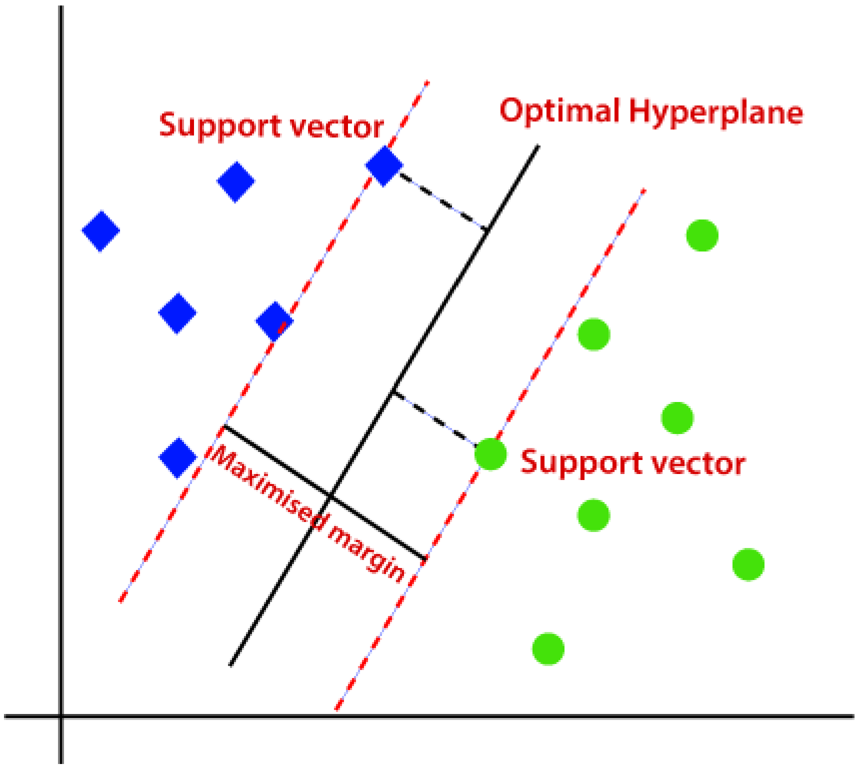


To classify these points, we can have many decision boundaries, but the question is which is the best and how do we find it?

NOTE: Since we are plotting the data points in a 2-dimensional graph we call this decision boundary a **straight line** but if we have more dimensions, we call this decision boundary a "**hyperplane**"



The best hyperplane is that plane that has the maximum distance from both the classes, and this is the main aim of SVM. This is done by finding different hyperplanes which classify the labels in the best way then it will choose the one which is farthest from the data points or the one which has a maximum margin.



Mathematical Intuition Behind Support Vector Machine

Many people skip the math intuition behind this algorithm because it is pretty hard to digest. Here in this section, we'll try to understand each and every step working under the hood. SVM is a broad topic and people are still doing research on this algorithm. If you are planning to do research, then this might not be the right place for you.

Here we will understand only that part that is required in implementing this algorithm. You must have heard about the **primal formulation, dual formulation, Lagranges multiplier** etc.

Before getting into the nitty-gritty details of this topic first let's understand what a dot product is.

Understanding Dot-Product

We all know that a vector is a quantity that has magnitude as well as direction and just like numbers we can use mathematical operations such as addition, multiplication. In this section, we will try to learn about the multiplication of vectors which can be done in two ways, dot product, and cross product. The difference is only that the dot product is used to get a scalar value as a resultant whereas cross-product is used to obtain a vector again.

The dot product can be defined as the projection of one vector along with another, multiply by the product of another vector.

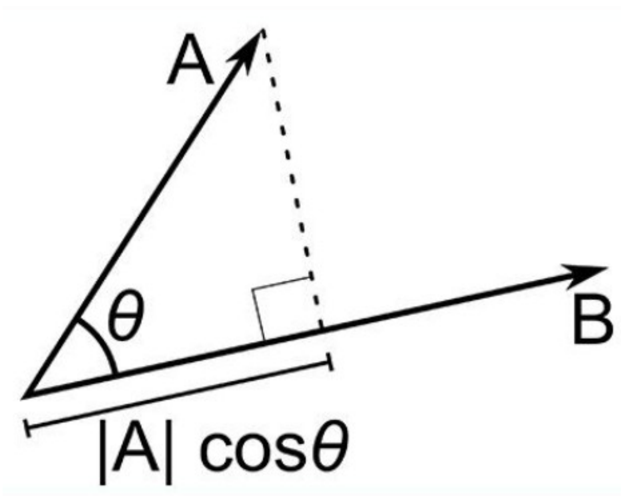


Image 2

Here a and b are 2 vectors, to find the dot product between these 2 vectors we first find the magnitude of both the vectors and to find magnitude we use the Pythagorean theorem or the distance formula.

After finding the magnitude we simply multiply it with the cosine angle between both the vectors. Mathematically it can be written as:

A . B = |A| cosθ * |B|

Where |A| cosθ is the projection of A on B

And |B| is the magnitude of vector B

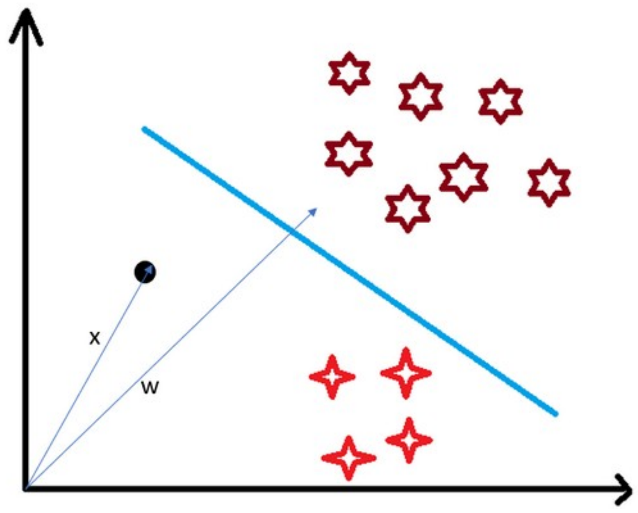
Now in SVM we just need the projection of A not the magnitude of B, I'll tell you why later. To just get the projection we can simply take the unit vector of B because it will be in the direction of B but its magnitude will be 1. Hence now the equation becomes:

A.B = |A| cosθ * unit vector of B

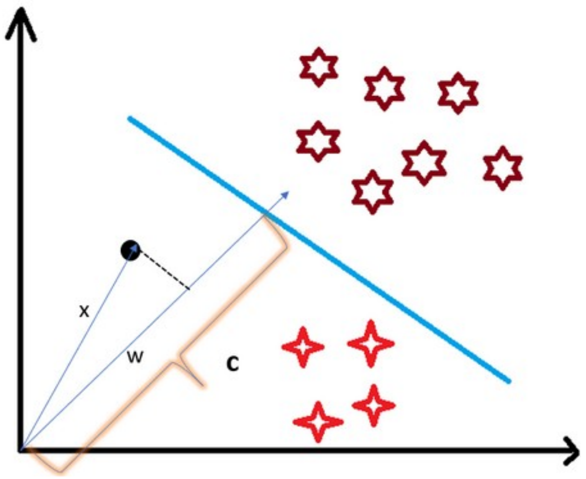
Now let's move to the next part and see how we will use this in SVM.

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.

$$\vec{X}.\vec{w} = c \quad (the\ point\ lies\ on\ the\ decision\ boundary)$$

$$\vec{X}.\vec{w} > c\ (positive\ samples)$$

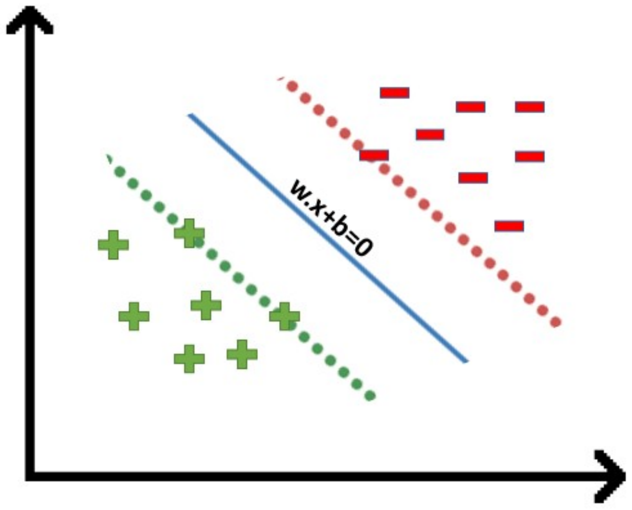
$$\vec{X}.\vec{w} < c\ (negative\ samples)$$

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:

$$\vec{X}.\vec{w} - \mathrm{c} \geq 0$$

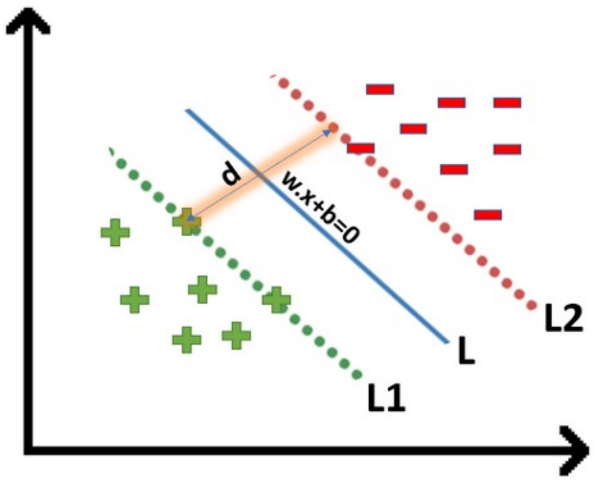
$$\text{putting } -\mathrm{c} \text{ as b, we get}$$

$$\vec{X}.\vec{w} + \mathrm{b} \geq 0$$

$$\text{hence}$$

$$\mathrm{y} = \begin{cases} +1 & \text{if } \vec{X}.\vec{w} + \mathrm{b} \geq 0 \\ -1 & \text{if } \vec{X}.\vec{w} + \mathrm{b} < 0 \end{cases}$$

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**.

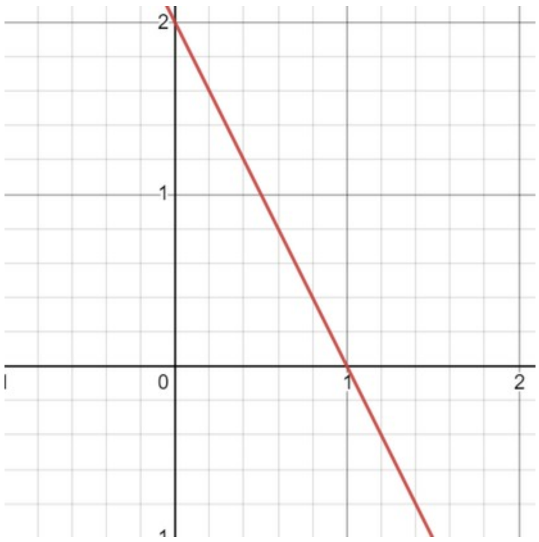
Now the question comes

- 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?

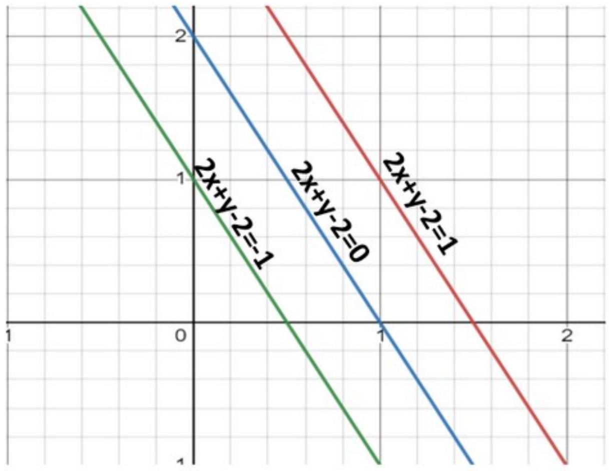
Let’s try to answer these questions

- 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. For more clarity look at this graph (https://www.desmos.com/calculator/dvjo3vacyp)

We also observe that if we divide this equation by 10 then these parallel lines get bigger. Look at this graph (https://www.desmos.com/calculator/15dbwehq9g).

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:

$$For\ all\ the\ Red\ points\ \overrightarrow{w}.\overrightarrow{X}+b\leq -1$$

$$For\ all\ the\ Green\ points\ \overrightarrow{w}.\overrightarrow{X}+b\geq 1$$

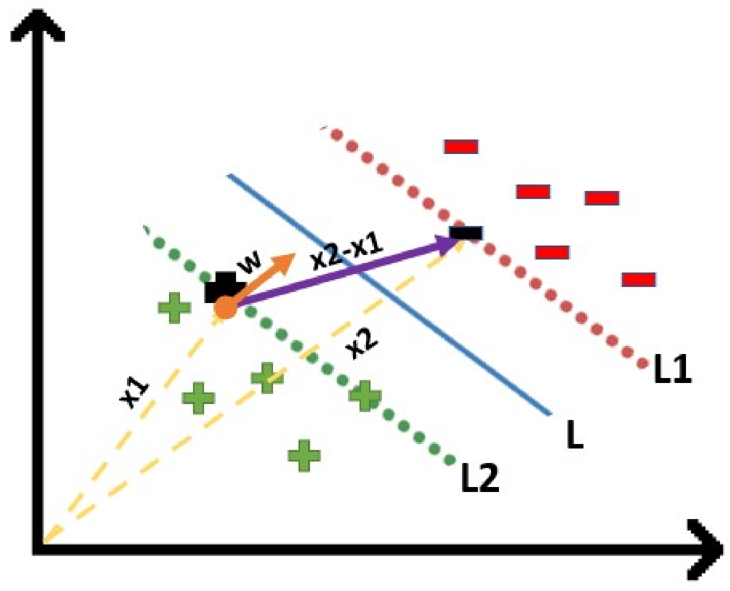
Rather than taking 2 constraints forward, we'll now try to simplify these two constraints into 1. We assume that negative classes have ***y=-1*** and positive classes have ***y=1.***

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

$$y_i(\overrightarrow{w}.\overrightarrow{X}+b)\geq 1$$

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. Why this should be a unit vector? This has been explained in the dot-product section. To make this 'w' a unit vector we divide this with the norm of 'w'.



Finding Projection of a Vector on Another Vector Using Dot Product

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

$$\Rightarrow\ (\mathrm{x2}-\mathrm{x1}) . \frac{\overrightarrow{\mathrm{w}}}{\|\mathrm{w}\|}$$

$$\Rightarrow\ \frac{\mathrm{x2}.\overrightarrow{\mathrm{w}}-\ \mathrm{x1}.\overrightarrow{\mathrm{w}}}{\|\mathrm{w}\|} \qquad ----(1)$$

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

$$\text{for positive point } \mathrm{y}=1$$

$$\Rightarrow\ 1\times(\overrightarrow{\mathrm{w}}.x1+b)\ =\ 1$$

$$\Rightarrow\ \overrightarrow{\mathrm{w}}.x1\ =\ 1-\mathrm{b} \qquad -----(2)$$

$$\text{Similarly for negative point } \mathrm{y}=\ -1$$

$$\Rightarrow\ -1\times\left(\overrightarrow{\mathrm{w}}.x2+b\right)\ =\ 1$$

$$\Rightarrow\ \overrightarrow{\mathrm{w}}.x2\ =\ -\mathrm{b}-1 \qquad -----(3)$$

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

$$\Rightarrow\ \frac{(1-\mathrm{b})-(-\mathrm{b}-1)}{\|\mathrm{w}\|}$$

$$\Rightarrow\ \frac{1-\mathrm{b}+\mathrm{b}+1}{\|\mathrm{w}\|}\ =\ \frac{2}{\|\mathrm{w}\|}\ =\mathrm{d}$$

Hence the equation which we have to maximize is:

$$\operatorname{argmax}(\mathrm{w}^*,\mathrm{b}^*)\ \frac{2}{\|\mathrm{w}\|} \text{ such that } \mathrm{y}_{\mathrm{i}}\left(\vec{w}.\vec{X}+b\right)\geq 1$$

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.

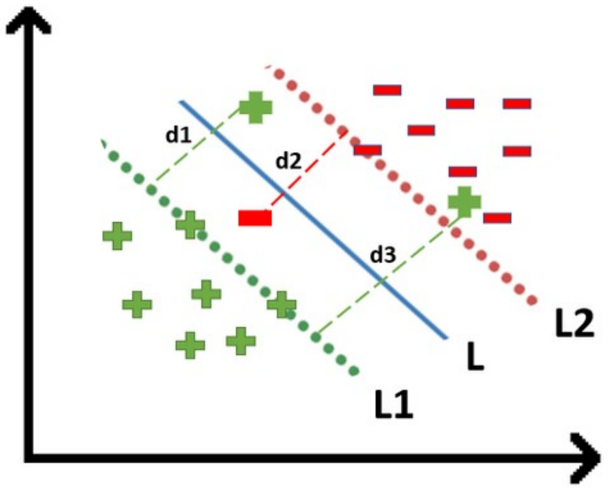
We know that *max[f(x)]* can also be written as *min[1/f(x)]*, it is common practice to minimize a cost function for optimization problems; therefore, we can invert the function.

$$\mathrm{argmin}(\mathrm{w}^*, \mathrm{b}^*)\ \frac{\|\mathrm{w}\|}{2}\ \text{such that}\ \mathrm{y}_\mathrm{i}\left(\vec{w}.\vec{X} + b\right) \geq 1$$

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

$$\mathrm{argmin}\left(\mathrm{w}^*, \mathrm{b}^*\right)\ \frac{\|\mathrm{w}\|}{2}\ + c\sum_{i=1}^{n}\zeta_i$$

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:

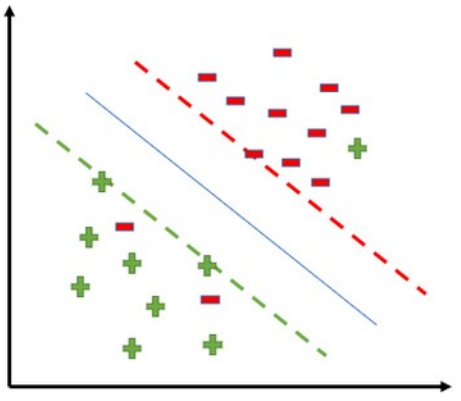


Figure 1

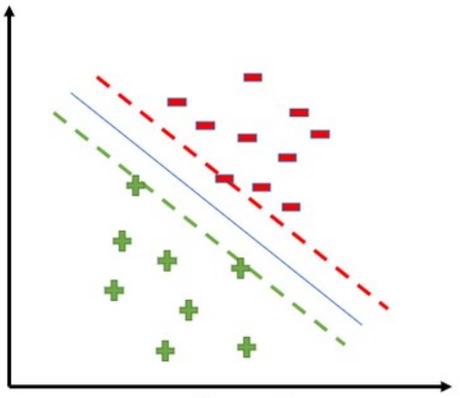


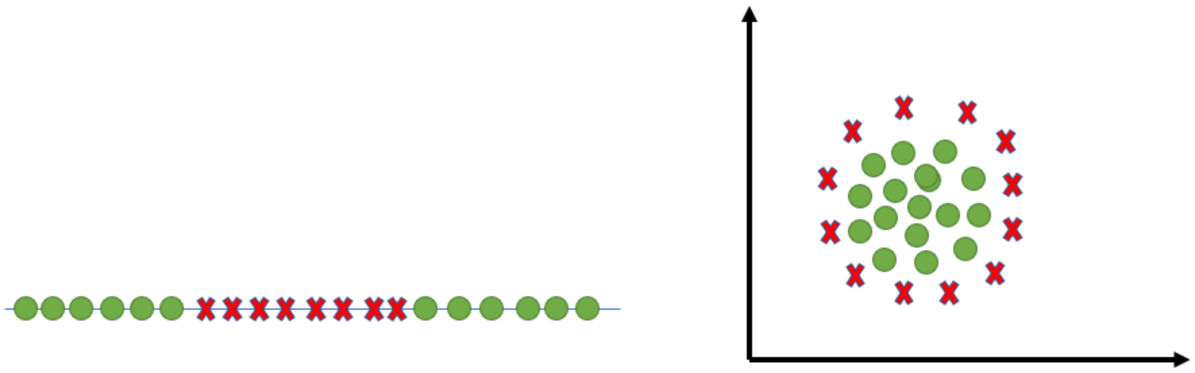
Figure 2

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?

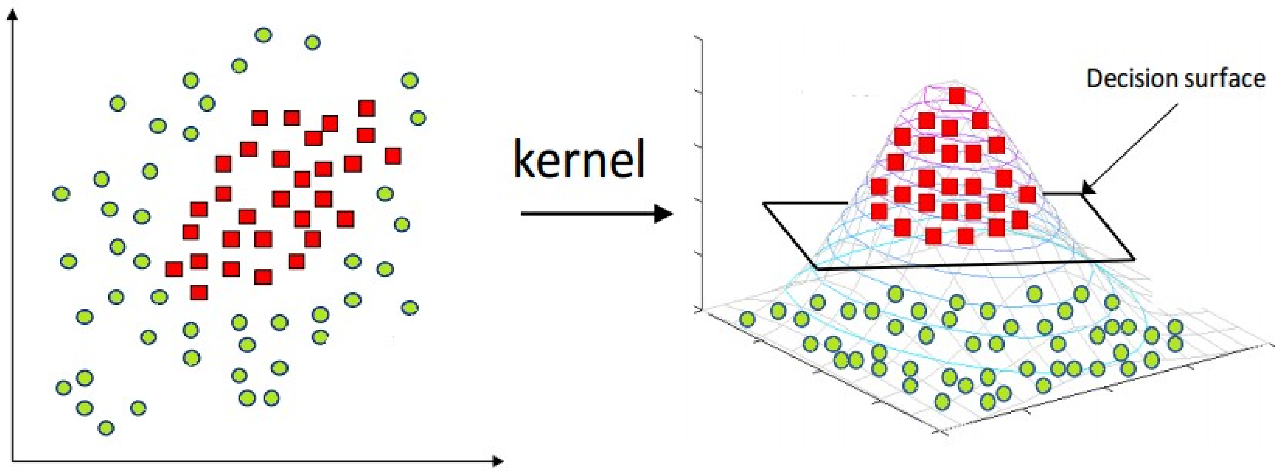
Well, there’s no correct answer to this question, but rather we can use *SVM Error = Margin Error + Classification Error to* justify this. If you don’t want any misclassification in the model then you can choose *figure 2*. That means we’ll increase ‘c’ to decrease Classification Error but if you want 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

The most interesting feature of 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:

$$f(X1, X2) = (X1^T . X2 + 1)^d$$

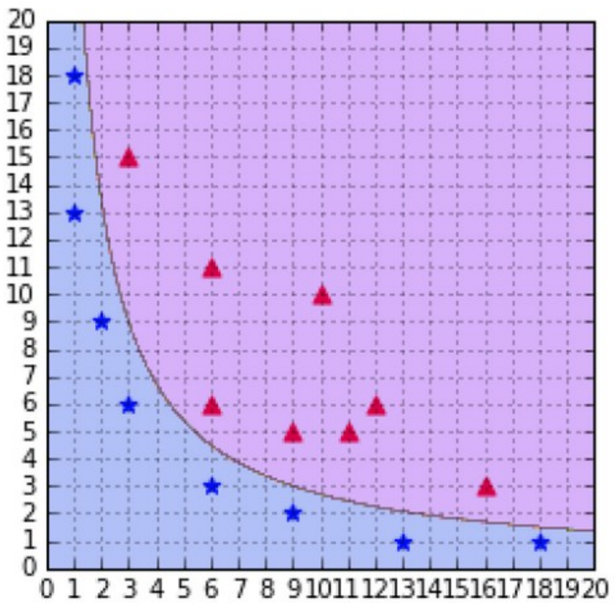
Here d is the degree of the polynomial, which we need to specify manually.

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

$$X1^T . X2 = \begin{bmatrix} X1 \\ X2 \end{bmatrix} . \begin{bmatrix} X1 & X2 \end{bmatrix}$$

$$= \begin{bmatrix} X1^2 & X1 . X2 \\ X1 . X2 & X2^2 \end{bmatrix}$$

So we basically need to find X_1^2 , X_2^2 and X1.X2, and now we can see that 2 dimensions got converted into 5 dimensions.



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

2. Sigmoid Kernel

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

$$f(x1, x2) \;=\; \tanh(\alpha x^T y + x)$$

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.

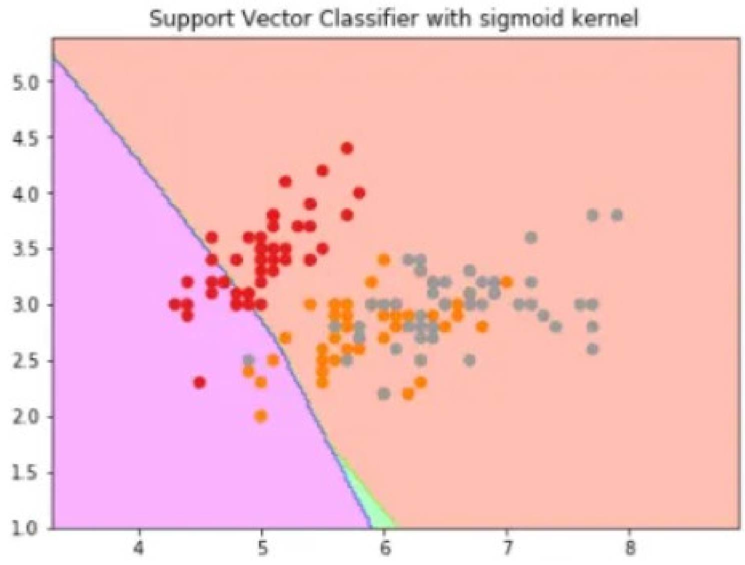


Image Source: https://dataaspirant.com/svm-kernels/#t-1608054630725

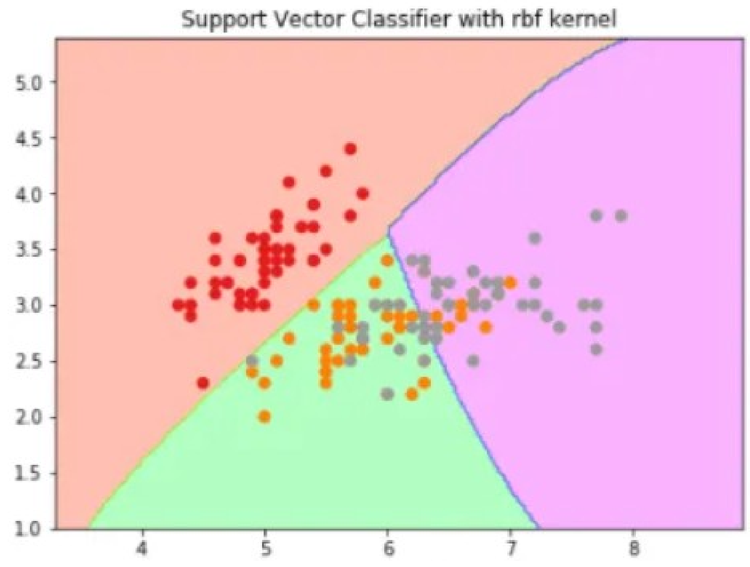
3. RBF Kernel

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:

$$f(x1, x2) = e^{\frac{-||(x1 - x2)||^2}{2\sigma^2}}$$

where,

1. 'σ' is the variance and our hyperparameter
2. $||X_1 - X_2||$ is the Euclidean Distance between two points X_1 and X_2



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:

$$k(x, y) = \frac{J_{v+1}(\sigma||x - y||)}{||x - y||^{-n(v+1)}}$$

5. Anova Kernel

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

$$k(x, y) = \sum_{k=1}^{n} \exp(-\sigma(x^k - y^k)^2)^d$$

How to Choose the Right Kernel?

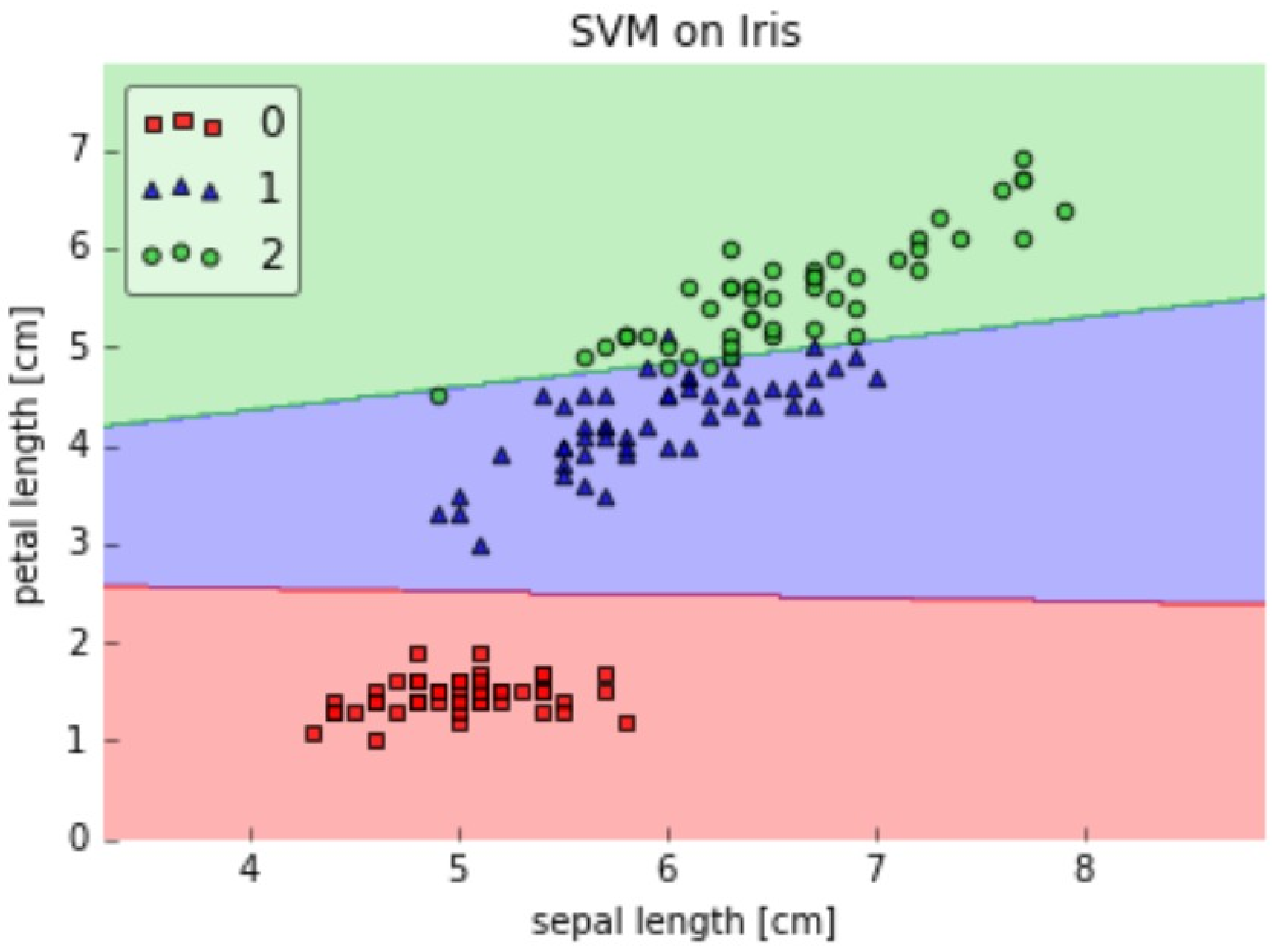
I am well aware of the fact that you must be having this doubt about how to decide which kernel function will work efficiently for your dataset. It is necessary to choose a good kernel function because the performance of the model depends on it.

Choosing a kernel totally depends on what kind of dataset are you working on. If it is linearly separable then you must opt. for linear kernel function since it is very easy to use and the complexity is much lower compared to other kernel functions. I'd recommend you start with a hypothesis that your data is linearly separable and choose a linear kernel function.

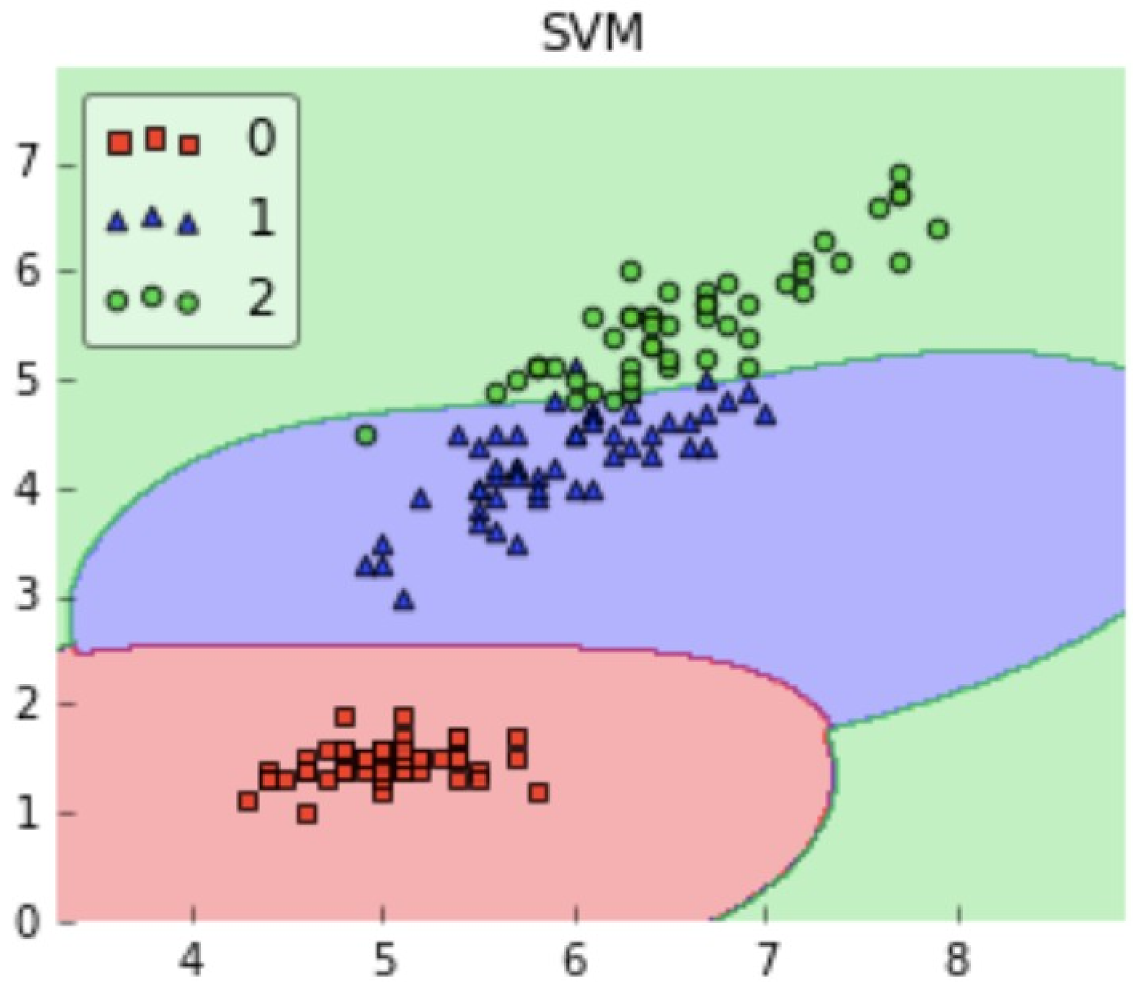
You can then work your way up towards the more complex kernel functions. Usually, we use SVM with RBF and linear kernel function because other kernels like polynomial kernel are rarely used due to poor efficiency. But what if linear and RBF both give approximately similar results? Which kernel do we choose now?

Example

Let's understand this with the help of an example, for simplicity I'll only take 2 features that mean 2 dimensions only. In the figure below I have plotted the decision boundary of a linear SVM on 2 features of the iris dataset:



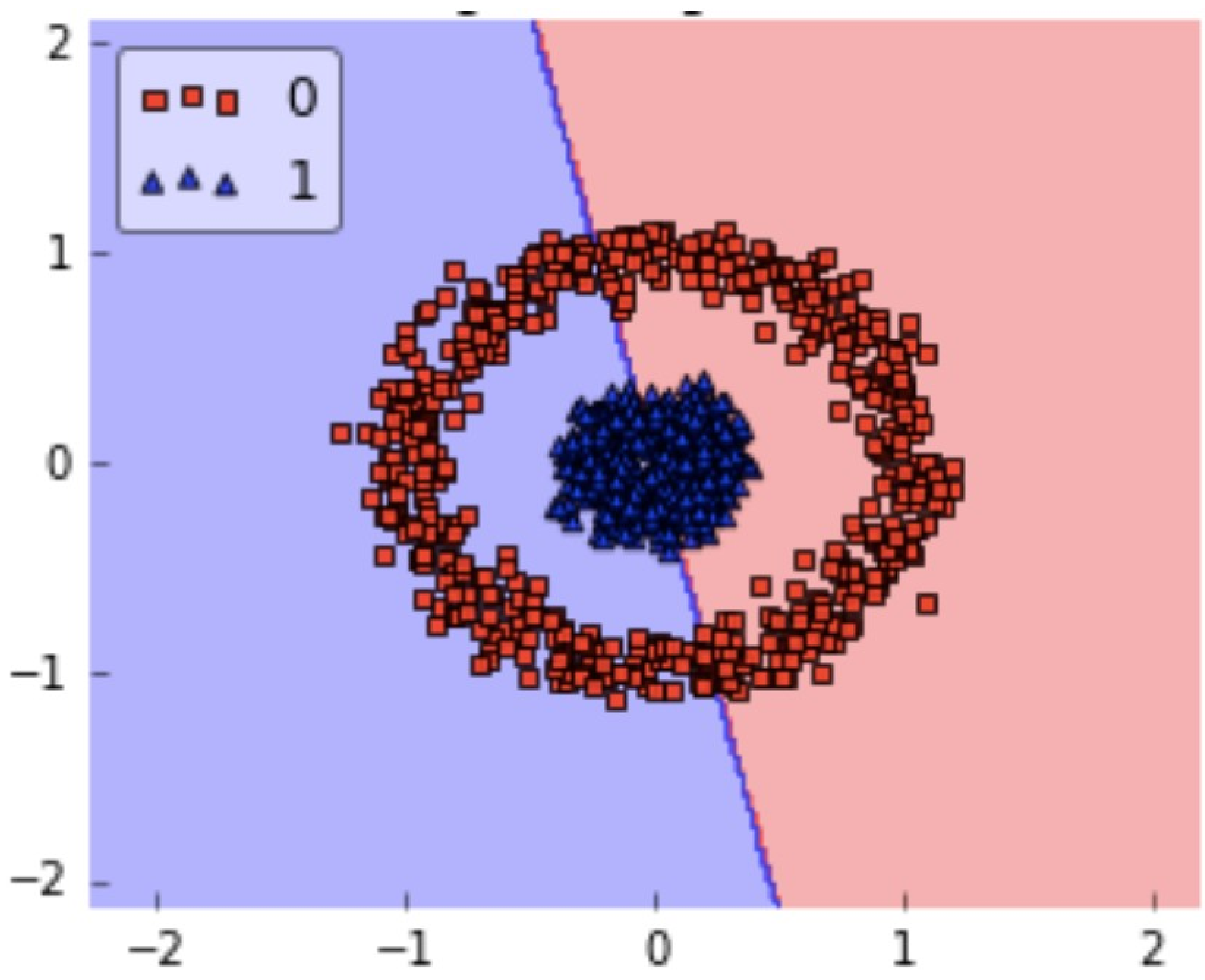
Here we see that a linear kernel works fine on this dataset, but now let’s see how will RBF kernel work.



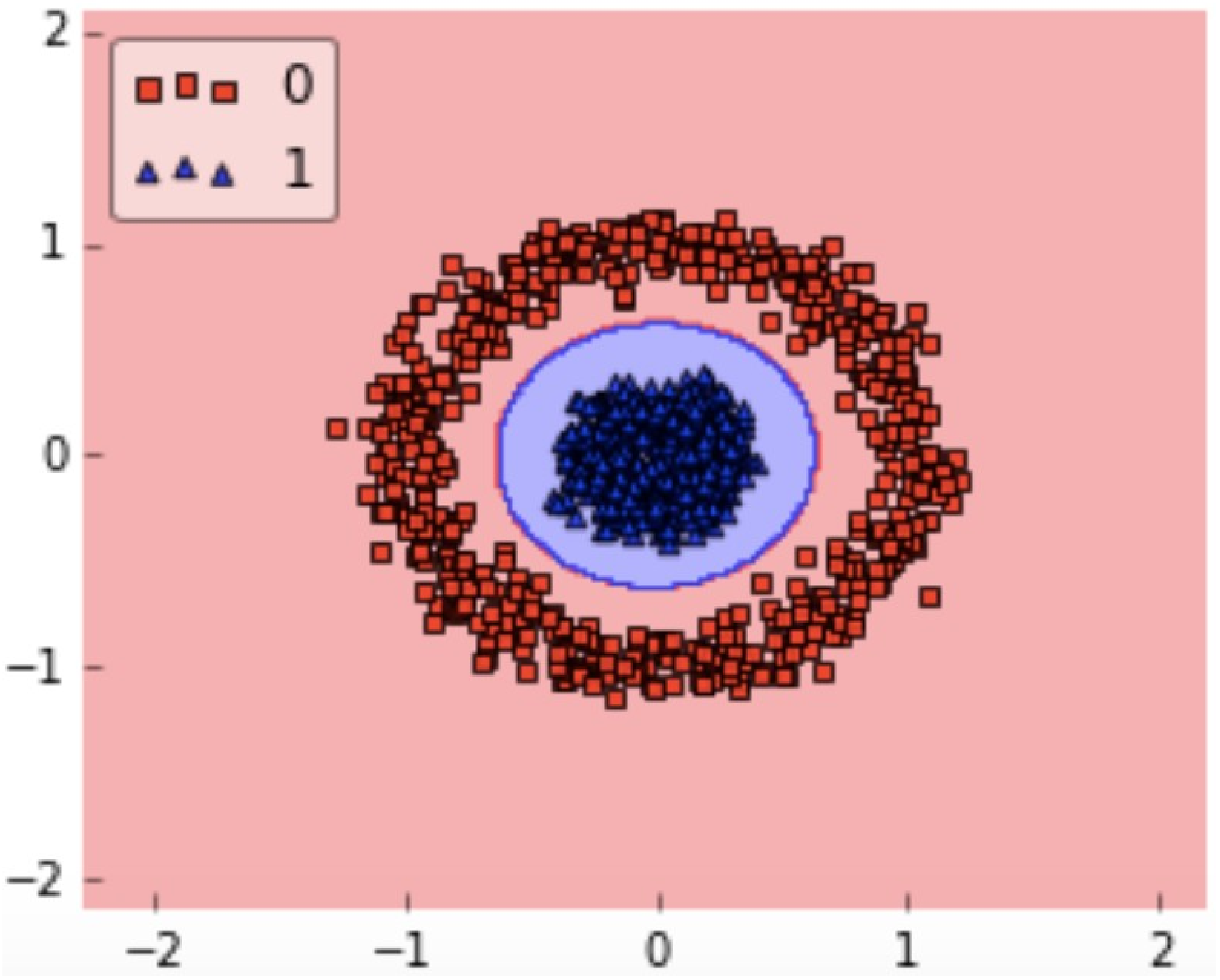
We can observe that both the kernels give similar results, both work well with our dataset but which one should we choose?

Linear SVM is a parametric model. A Parametric Model is a concept used to describe a model in which all its data is represented within its parameters. In short, the only information needed to predict the future from the current value is the parameters.

The complexity of the RBF kernel grows as the training data size increases. In addition to the fact that it is more expensive to prepare RBF kernel, we also have to keep the kernel matrix around, and the projection into this “infinite” higher dimensional space where the data becomes linearly separable is more expensive as well during prediction. If the dataset is not linear then using linear kernel doesn’t make sense we’ll get a very low accuracy if we do so.



So for this kind of dataset, we can use RBF without even a second thought because it makes decision boundary like this:


0
1
2
1
0
−1
−2
−2
−1
0
1
2