Support Vector Machine

Introduction:

Being a data science practitioner, you must be aware of the different algorithms available at our end. The important point is the awareness of when to use which algorithm. As an analogy, think of ‘Regression’ as a sword capable of slicing and dicing data efficiently, but incapable of dealing with highly complex data.

On the contrary, the ‘Support Vector Machine’ is like a sharp knife – it works on smaller datasets, but on complex ones, it can be much stronger and powerful in building machine learning models.

SVM is a supervised learning algorithm that can be used for both classifications as well as regression problems. However, mostly it is used for classification problems. It is a highly efficient and preferred algorithm due to significant accuracy with less computation power.

What is SVM?

In the SVM algorithm, we plot each observation as a point in an n-dimensional space (where n is the number of features in the dataset). Our task is to find an optimal hyperplane that successfully classifies the data points into their respective classes.

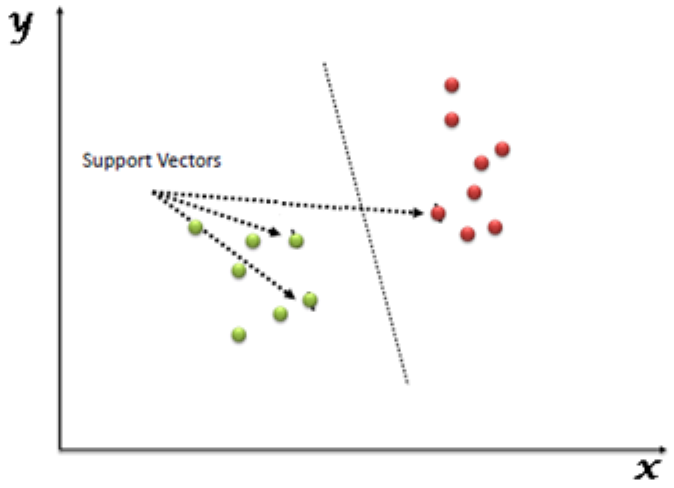
Before diving into the working of SVM let’s first understand the two basic terms used in the algorithm “The support vector ” and ” Hyper-Plane”.

Hyper-Plane

A hyperplane is a decision boundary that differentiates the two classes in SVM. A data point falling on either side of the hyperplane can be attributed to different classes. The dimension of the hyperplane depends on the number of input features in the dataset. If we have 2 input features the hyper-plane will be a line. likewise, if the number of features is 3, it will become a two-dimensional plane.

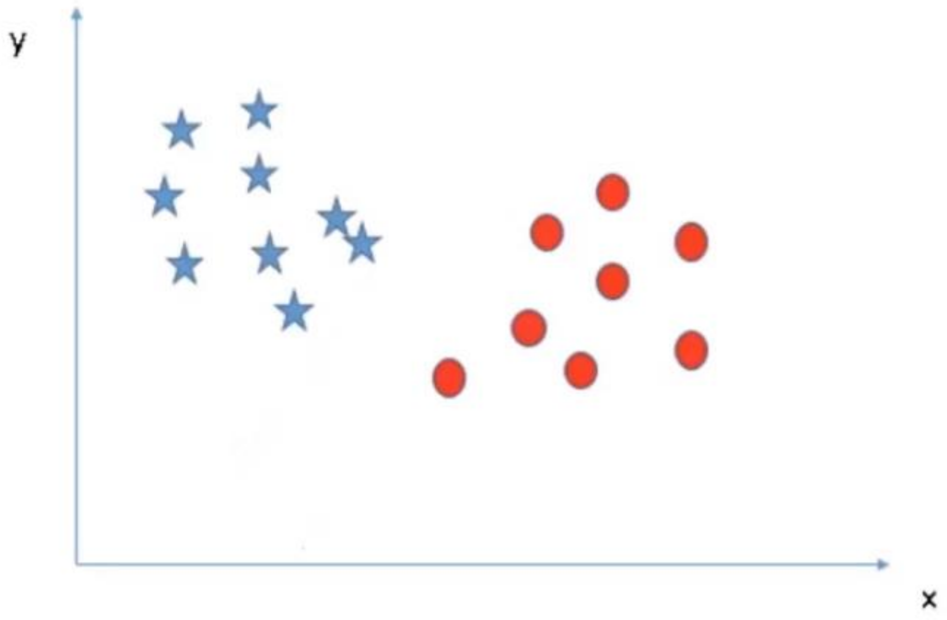
Support-Vectors

Support vectors are the data points that are nearest to the hyper-plane and affect the position and orientation of the hyper-plane. We have to select a hyperplane, for which the margin, i.e the distance between support vectors and hyper-plane is maximum. Even a little interference in the position of these support vectors can change the hyper-plane.

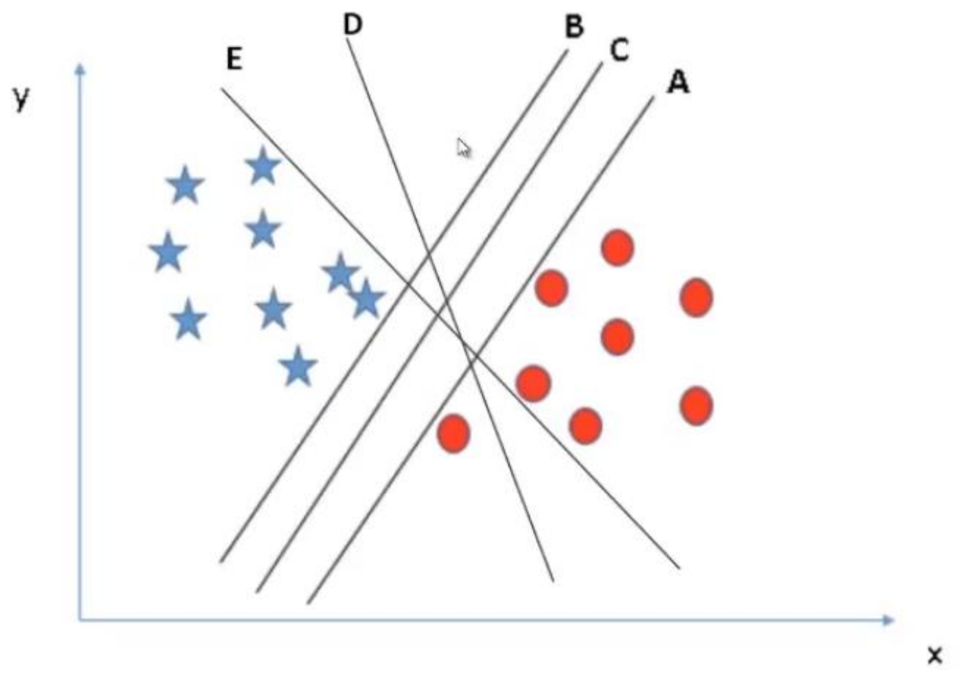


How SVM works?

As we have a clear idea of the terminologies related to SVM, let’s now see how the algorithm works. For example, we have a classification problem where we have to separate the red data points from the blue ones.



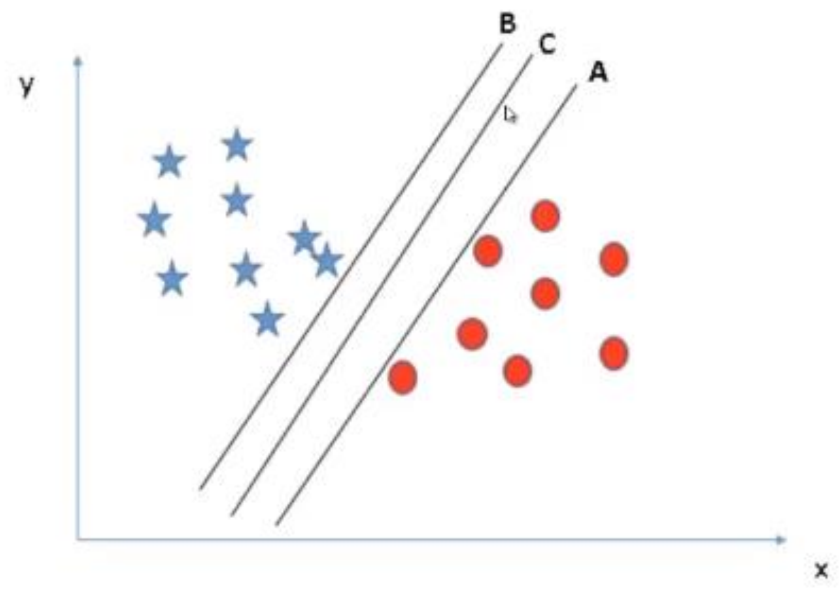
Since it is a two-dimensional problem, our decision boundary will be a line, for the 3-dimensional problem we have to use a plane, and similarly, the complexity of the solution will increase with the rising number of features.



As shown in the above image, we have multiple lines separating the data points successfully. But our objective is to look for the best solution.

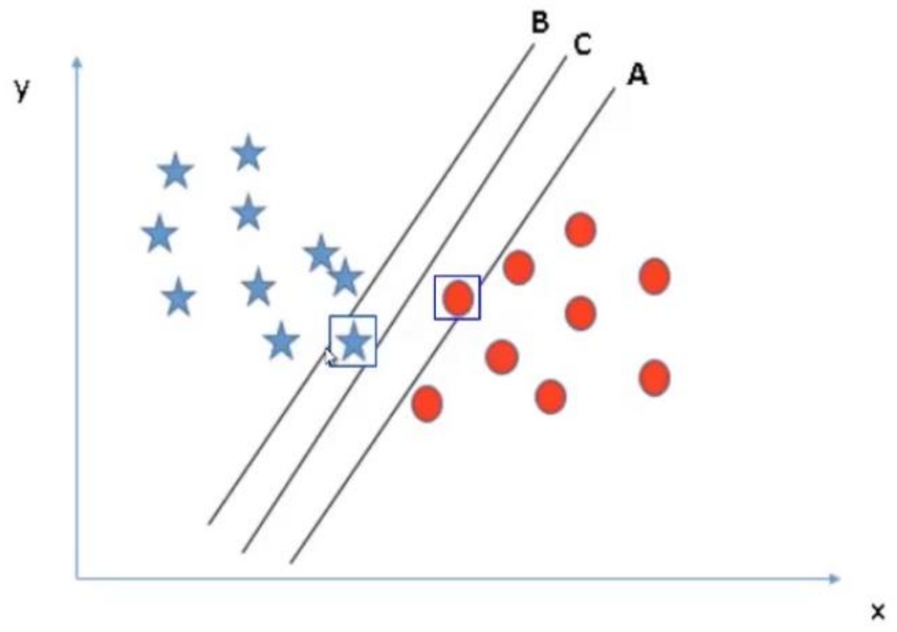
There are few rules that can help us to identify the best line.

Maximum classification, i.e the selected line must be able to successfully segregate all the data points into the respective classes. In our example, we can clearly see lines E and D are miss classifying a red data point. Hence, for this problem lines A, B, C is better than E and D. So we will drop them.



The second rule is **Best Separation**, which means, we must choose a line such that it is perfectly able to separate the points.

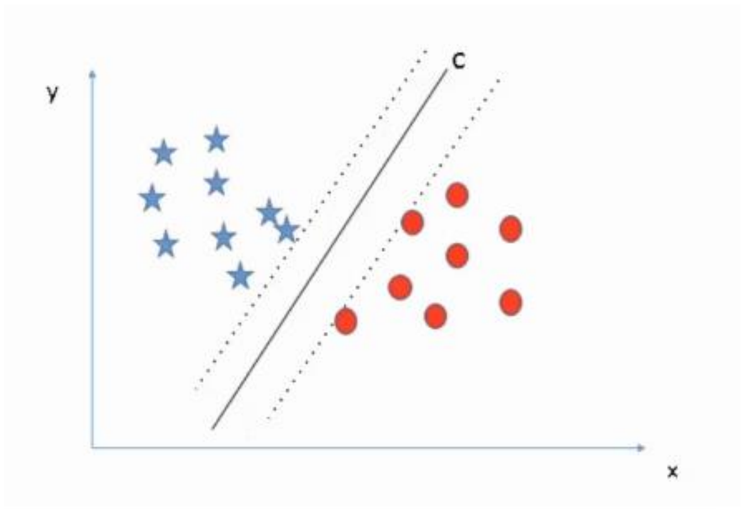
If we talk about our example, if we get a new red data point closer to line A as shown in the image below, line A will miss classifying that point. Similarly, if we got a new blue instance closer to line B, then line A and C will classify the data successfully, whereas line B will miss classifying this new point.

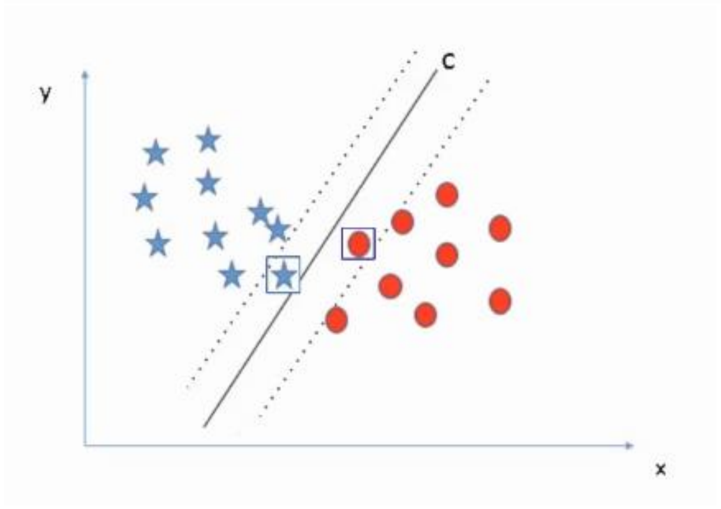


The point to be noticed here, In both the cases line C is successfully classifying all the data points why? To understand this let’s take all the lines one by one.

Why Line C?

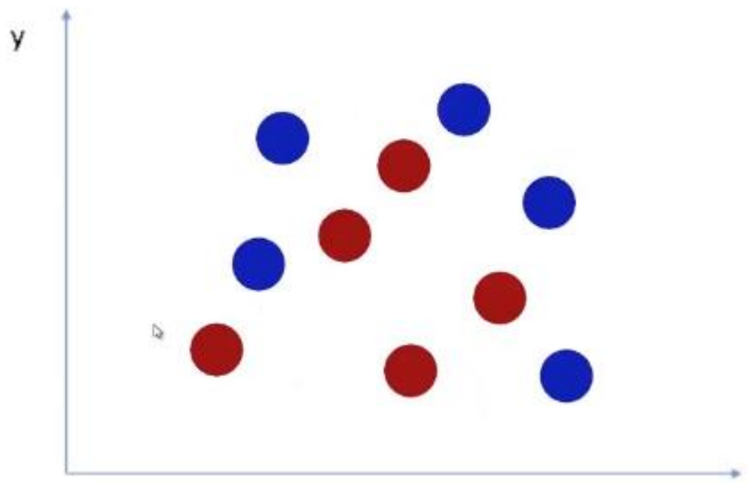
In the case of line C, It has sufficient margin on the left as well as the right side. This maximum margin makes line C more robust for the new data points that might appear in the future. Hence, C is the best fit in that case that successfully classifies all the data points with the maximum margin.



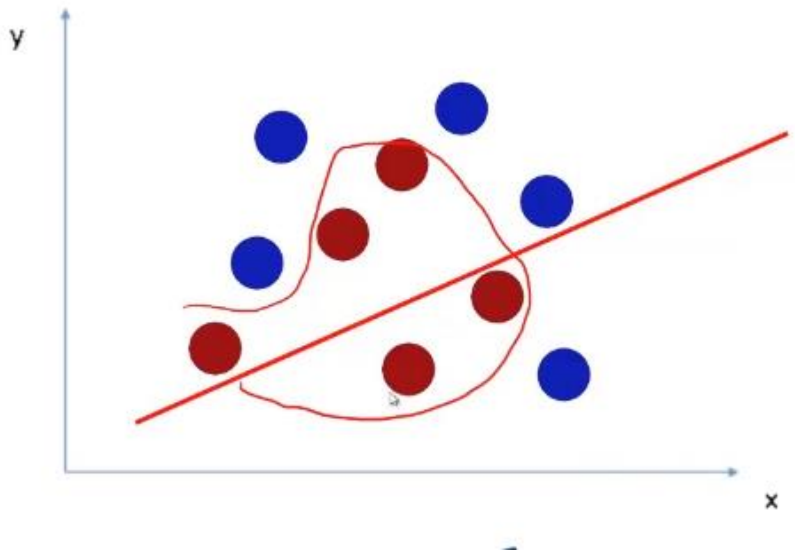


This is what SVM looks for, it aims for the maximum margin and creates a line that is equidistant from both sides, which is line C in our case. so we can say C represents the SVM classifier with the maximum margin.

Now let’s look at the data below, As we can see this is not linearly separable data, so SVM will not work in this situation. If anyhow we try to classify this data with a line, the result will not be promising.

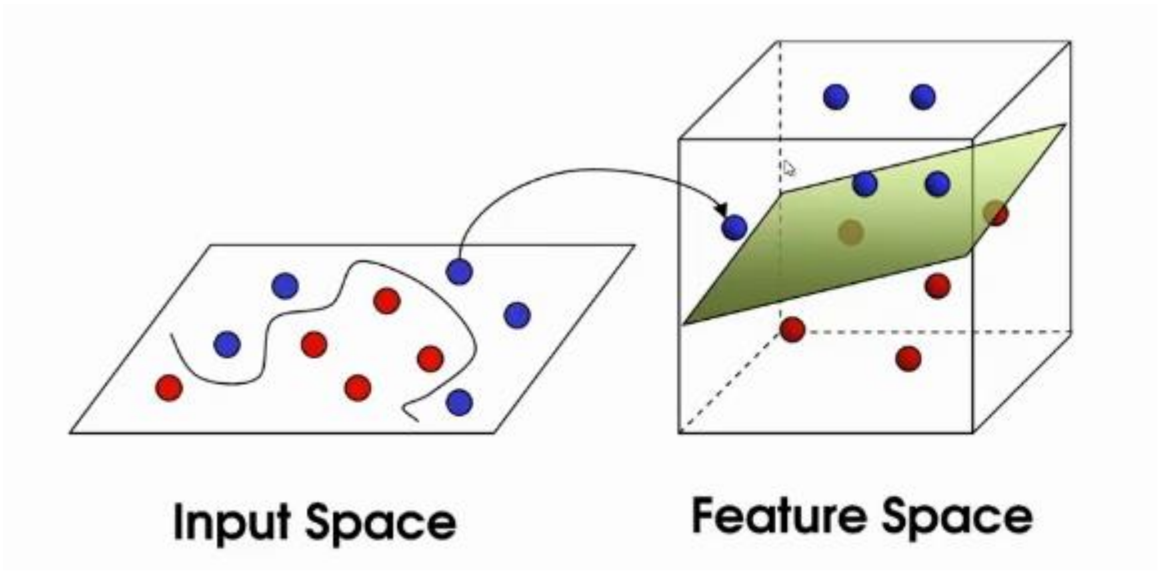


So, is there any way that SVM can classify this kind of data? For this problem, we have to create a decision boundary that looks something like this.



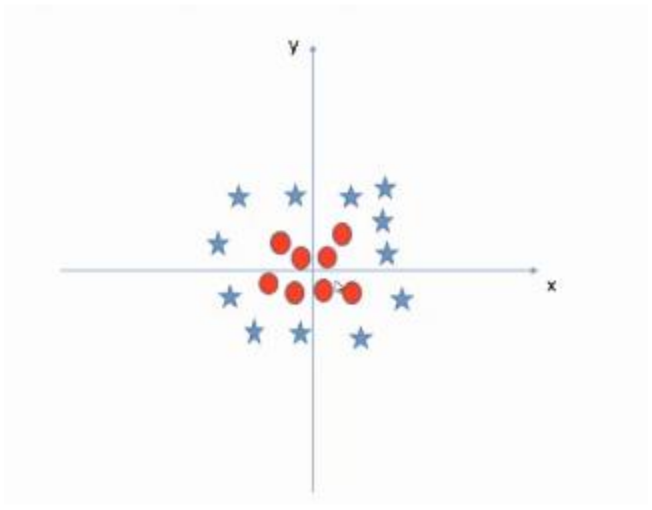
The question is, is it possible to create such a decision boundary using SVM. Well, the answer is **Yes**. SVM does this by projecting the data in a higher dimension. As shown in the following image. In the first case, data is not linearly separable, hence, we project into a higher dimension.

If we have more complex data then SVM will continue to project the data in a higher dimension till it becomes linearly separable. Once the data become linearly separable, we can use SVM to classify just like the previous problems.



Projection into Higher Dimension

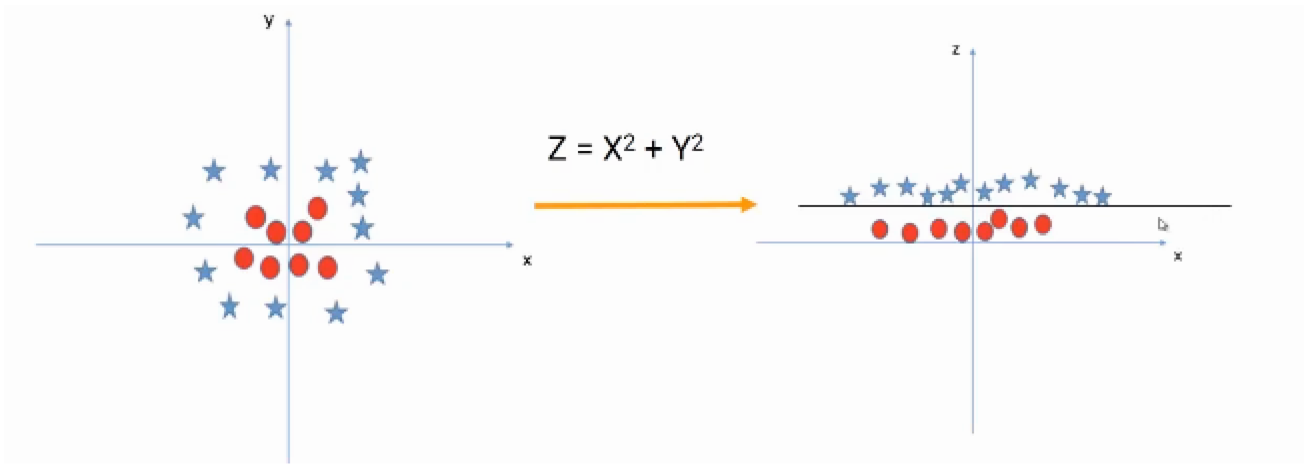
Now let’s understand how SVM projects the data into a higher dimension. Take this data, it is a circular non linearly separable dataset.



To project the data in a higher dimension, we are going to create another dimension z, where

$$Z = X^2 + Y^2$$

Now we will plot this feature Z with respect to x, which will give us linearly separable data that looks like this.



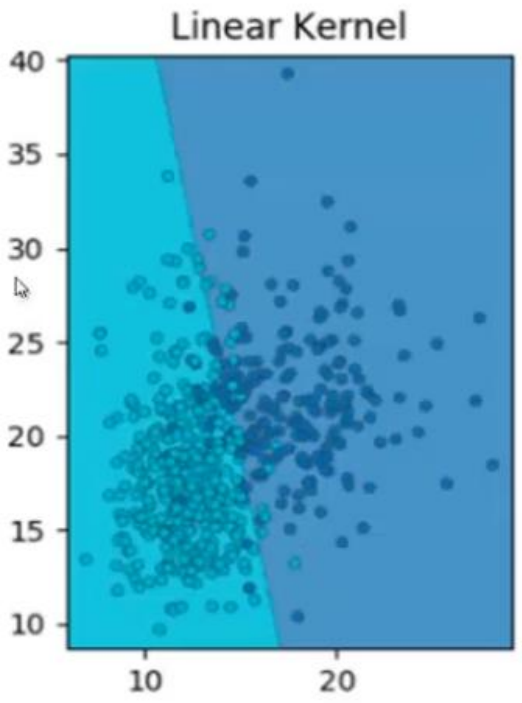
Here, we have created a mapping Z using the base features X and Y, this process is known as **kernel transformation**. Precisely, a kernel takes the features as input and creates the linearly separable data in a higher dimension.

Now the question is, do we have to perform this transformation manually? The answer is no. SVM handles this process itself, just we have to choose the kernel type.

Let’s quickly go through the different types of kernels available.

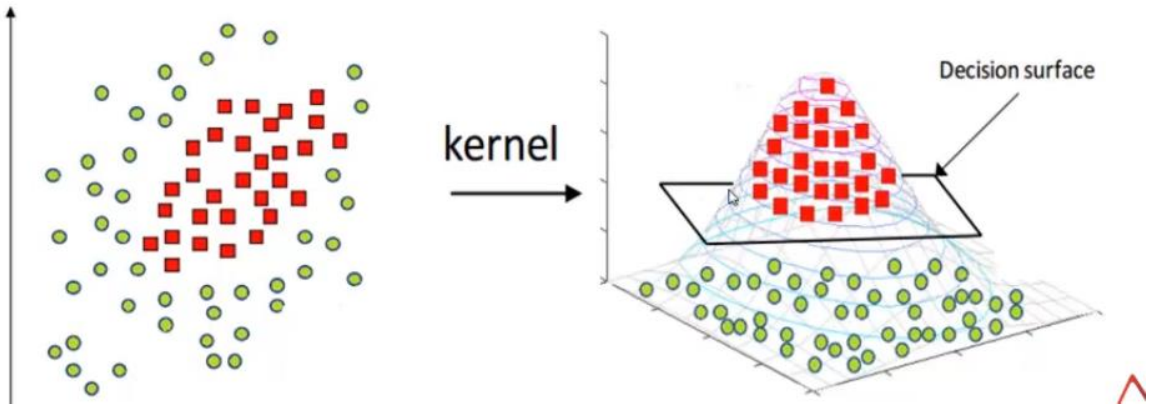
Linear Kernel

To start with, in the linear kernel, the decision boundary is a straight line. Unfortunately, most of the real-world data is not linearly separable, this is the reason the linear kernel is not widely used in SVM.



Gaussian / RBF kernel

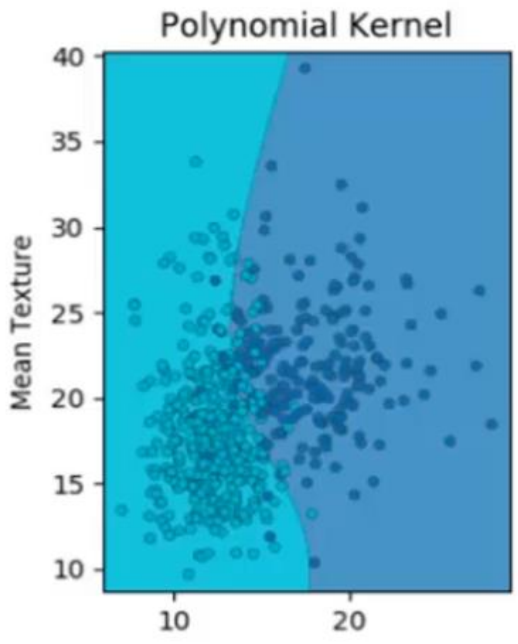
It is the most commonly used kernel. It projects the data into a Gaussian distribution, where the red points become the peak of the Gaussian surface and the green data points become the base of the surface, making the data linearly separable.



But this kernel is prone to overfitting and it captures the noise.

Polynomial kernel

At last, we have a polynomial kernel, which is non-uniform in nature due to the polynomial combination of the base features. It often gives good results.



But the problem with the polynomial kernel is, the number of higher dimension features increases exponentially. As a result, this is computationally more expensive than RBF or linear kernel.

SVM Use Cases

Some use-cases of SVM are as below.

- Face Detection

- Bioinformatics
- Classification of Images
- Remote Homology Detection
- Handwriting Detection
- Generalized Predictive Control
- Text and Hypertext Categorization

A very interesting fact is that SVM does not actually have to perform this actual transformation on the data points to the new high dimensional feature space. This is called the **kernel trick**.

The Kernel Trick: Internally, the kernelized SVM can compute these complex transformations just in terms of similarity calculations between pairs of points in the higher dimensional feature space where the transformed feature representation is implicit. This similarity function, which is mathematically a kind of complex dot product is actually the kernel of a kernelized SVM. This makes it practical to apply SVM when the underlying feature space is complex or even infinite-dimensional. The kernel trick itself is quite complex and is beyond the scope of this article.

Important Parameters in Kernelized SVC (Support Vector Classifier)

1. **The Kernel**: The kernel, is selected based on the type of data and also the type of transformation. By default, the kernel is Radial Basis Function Kernel (RBF).

2. **Gamma** : This parameter decides how far the influence of a single training example reaches during transformation, which in turn affects how tightly the decision boundaries end up surrounding points in the input space. If there is a small value of gamma, points farther apart are considered similar. So more points are grouped together and have smoother decision boundaries (maybe less

accurate). Larger values of gamma cause points to be closer together (may cause overfitting).

3. **The ‘C’ parameter**: This parameter controls the amount of regularization applied to the data. Large values of C mean low regularization which in turn causes the training data to fit very well (may cause overfitting). Lower values of C mean higher regularization which causes the model to be more tolerant of errors (may lead to lower accuracy).

   Note: The regularization parameter (lambda) **serves as a degree of importance that is given to misclassifications**.

Pros of Kernelized SVM:

1. They perform very well on a range of datasets.
2. They are versatile: different kernel functions can be specified, or custom kernels can also be defined for specific datatypes.
3. They work well for both high and low dimensional data.

Cons of Kernelized SVM:

1. Efficiency (running time and memory usage) decreases as the size of the training set increases.
2. Needs careful normalization of input data and parameter tuning.
3. Does not provide a direct probability estimator.
4. Difficult to interpret why a prediction was made.