Lecture Notes

Support Vector Machine

Support Vector Machine (SVM) is an advanced machine learning technique which has a unique way of solving complex problems such as image recognition, face detection, voice detection etc. As you will learn in this session, SVMs solves the problem of nonlinearity through kernels.

For instance, if you have a data as shown in the figure below, SVMs can handle it easily and that's how SVM distinguishes from logistic regression.

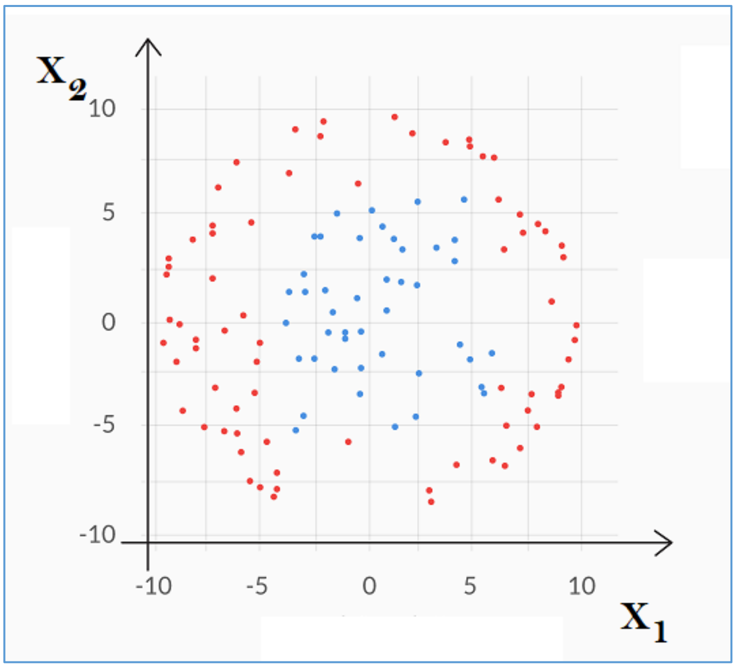

Figure 1: Nonlinear Data

It is important to remember that SVMs belong to the class of linear machine learning models (logistic regression is also a linear model).

A linear model uses a linear function (i.e. of the form y = ax + b) to model the relationship between the input x and output y. For example, in logistic regression, the log(odds) of an outcome (say, defaulting on a credit card) is linearly related to the attributes x1, x2, etc.

$$\boxed{log(odds\ of\ default) = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots \beta_n X_n}$$

Similarly, SVMs are also linear models and It needs attributes in the numeric form.

Concept of Hyperplane in 2D

Before you move on to support vector machines, you need to understand the concept of hyperplanes. Essentially, it is a boundary which classifies the data set (classifies Spam email from the ham ones). It could be lines, 2D planes, or even n-dimensional planes that are beyond our imagination.

A line that is used to classify one class from another is called a hyperplane. In fact, it is the model you're trying to build as shown in the figure below:

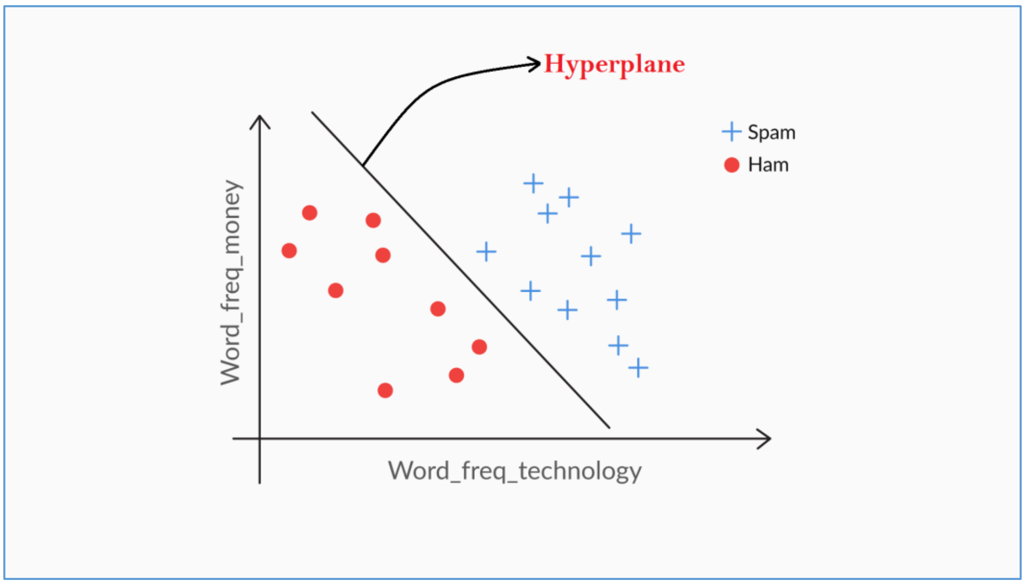

Figure 2: Hyperplane

The standard equation of a line is given by **ax + by + c = 0**. You could generalise it as $W_0 + W_1X_1 + W_2X_2 = 0$, where X_1 and X_2 are the features — such as '*word_freq_technology*' and '*word_freq_money*' — and W1 and W2 are the coefficients.

For any line with W coefficients, substituting the value of features x1 and x2 in the equation of the line determined by its W coefficients, will return a value.

A **positive value** (**blue points** in the plot above) would mean that the set of values of the features is in one class; however, a **negative value** (**red points** in the plot above) would imply it belongs to the other class. A value of zero would imply that the point lies on the line (hyperplane) because any point on the line will satisfy the equation $W_0 + W_1X_1 + W_2X_2 = 0$.

Concept of Hyperplane in 3D

In 3 dimensions (refer to the figure below), the hyperplane (light orange) will be a plane with an expression of **ax+by+cz+d = 0**. The plane divides the data set into two halves. Data points above the plane represent one class (red), while data points below the plane represent the other class (blue).

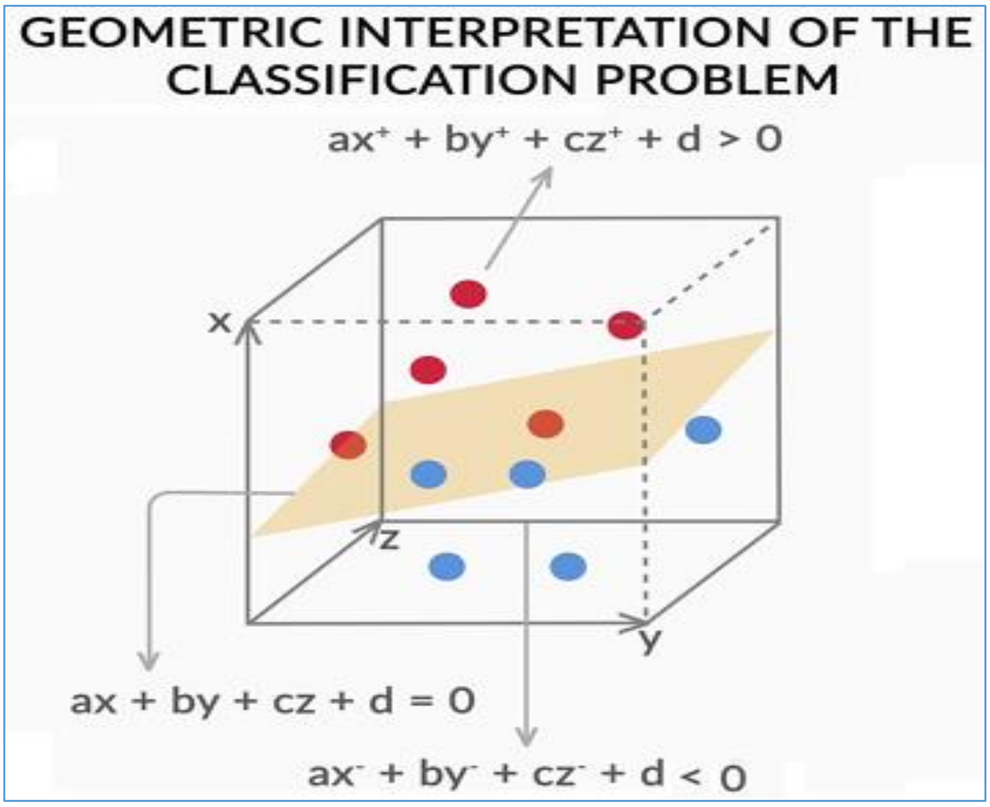

Figure 3: 3D Hyperplane - Plane

In general, if the hyperplane from d attributes, the expression in d-dimensional can be written as follows:

$$\boxed{\sum_{i=1}^{d}(W_i . X_i + W_0) = 0}$$

The model denoted by the expression given above is called **a linear discriminator**. Similar to the 2D and 3D expressions, an n-dimensional hyperplane also follows the general rule: all points above the plane will yield a value **greater than 0**, and those below it will yield **lesser than 0** when plugged into the expression of the hyperplane.

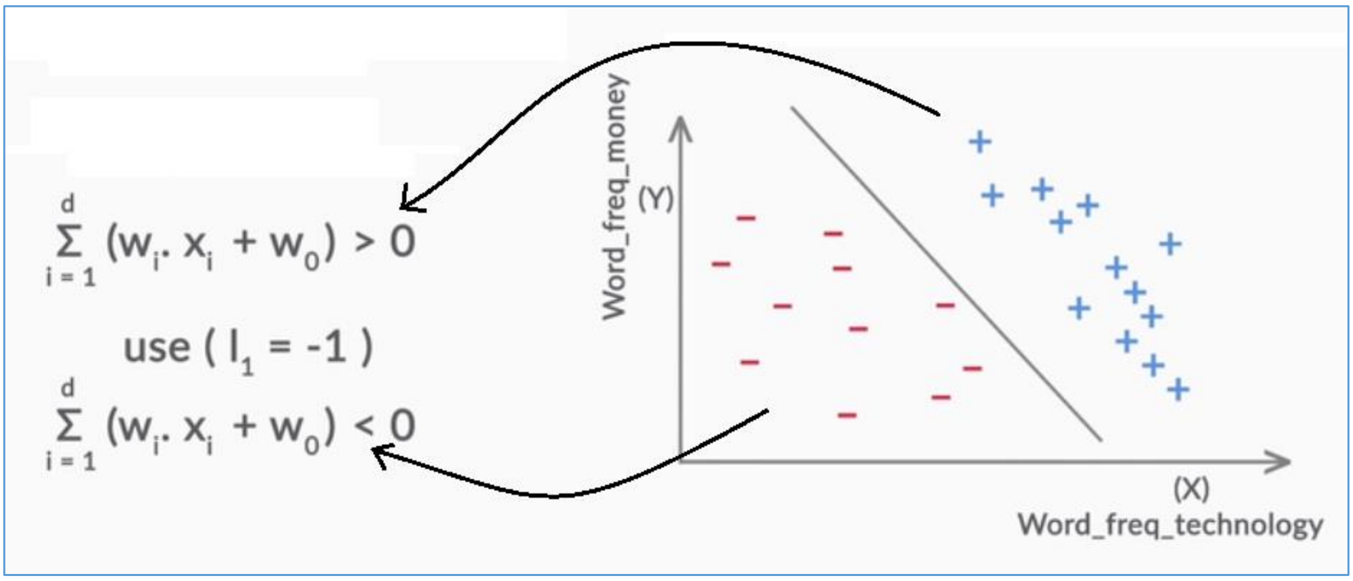

Figure 4: Linear Discriminator

Maximal Margin Classifier

There could be multiple lines (hyperplanes) possible, which perfectly separate the two classes as shown in the figure below. But the best line is the one that maintains the largest possible equal distance from the nearest points of both the classes. So for the separator to be optimal, the margin — or the distance of the nearest point to the separator — should be maximum. This is called the **Maximal Margin Classifier**.

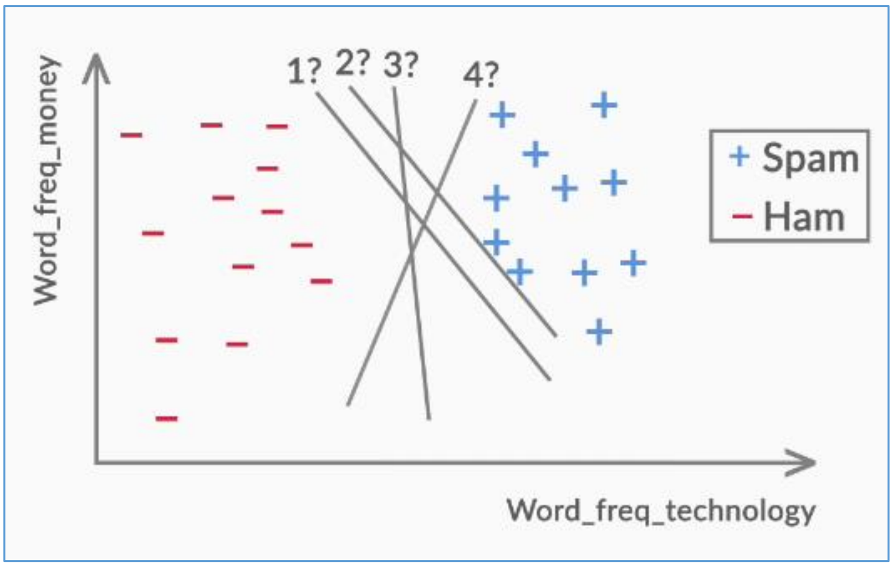

Figure 5: Maximal Margin Classifier

The third line (hyperplane) should be considered as the Maximal Margin Classifier in the figure above.

The mathematical formulation for the Maximal Margin Classifier requires two major constraints that need to be taken into account while maximising the margin. They are

- The standardisation of coefficients such that the summation of the square of the coefficients of all the attributes is equal to 1. For example, if you have 20 attributes, then the summation of the coefficients should be

$$\sum_{i=0}^{20}(W_i^2) = 1$$

- Along with the first constraint, the maximal margin hyperplane should also follow the constraint given below:

$$(l_i X(W_i . Y_i)) \geqslant M$$

where,

l_i = label (1, -1)

W_i = coefficient of attributes

Y_i = data points of all the attributes in each row

The maximal margin line (hyperplane), although it separates the two classes perfectly, is very sensitive to the **training data**. This means that **the Maximal Margin Classifier** will perform perfectly on the training data set. But on the unseen data, it may perform poorly. Also, there are cases where the classes cannot be perfectly separated.

But the **Soft Margin Classifier** helps in solving this problem.

Soft Margin Classifier

The **Support Vector Classifier** essentially allows certain points to be deliberately misclassified. By doing this, it is able to classify most of the points correctly in the unseen data and is also more robust.

The Support Vector Classifier is also called the **Soft Margin Classifier** because instead of searching for the margin that exactly classifies each and every data point to the correct class, the Soft Margin Classifier allows some observations to fall on the wrong side. The points that are close to the hyperplane are only considered for constructing the hyperplane, and they are called **support vectors**.

The Support Vector Classifier works well when the data is partially intermingled (i.e. the data can be classified by minimal misclassifications). But what if the distribution looks completely intermingled and follows some pattern, something like the circular distribution of labels (+ and -), as shown in the figure below.

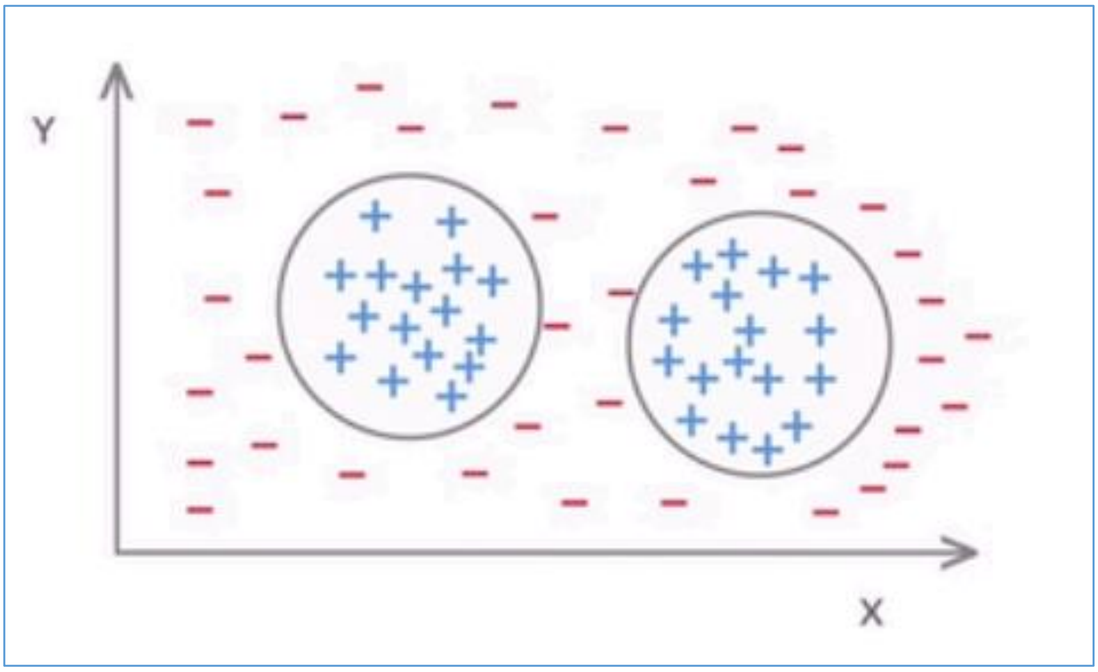

Figure 6: Intermingled Data

Obviously, the **Support Vector Classifier** can't classify the data above correctly because it divides the data set into two halves, which misclassifies a lot of data points. But this doesn't mean that this problem cannot be solved. There is a way to solve such problems, which you will learn later.

Like the Maximal Margin Classifier, the **Support Vector Classifier** also maximises the margin; but the margin, here, will allow some points to be misclassified, as shown in the figure below.

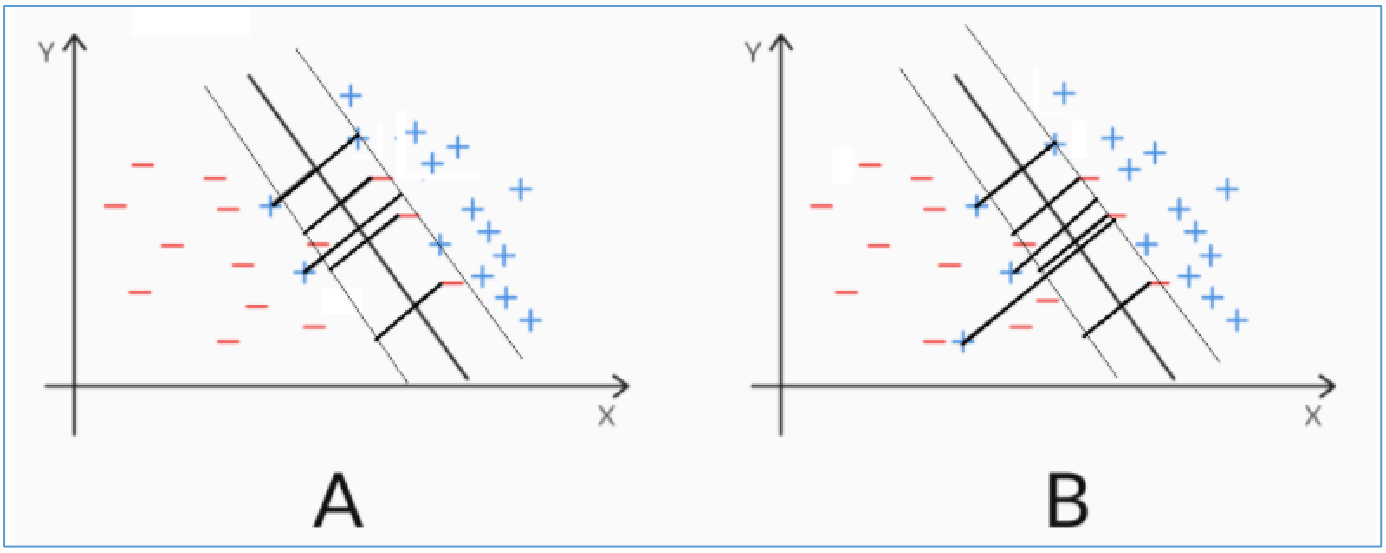

Figure 7: Soft Margin Classifier

So to select **the best-fit Support Vector Classifier**, the notion of slack variables (epsilons(ε)) can help by comparing the classifiers.

So let’s understand the concept of the slack variable(ϵ). A slack variable is used to control misclassifications. It tells you where an observation is located relative to the margin and hyperplane.

There are three different conditions applied if any new data point comes into play. Suppose you draw a Support Vector Classifier in such a way that it doesn't allow any misclassification, i.e. Epsilon(ϵ) = 0; then each observation is on the correct side of the margin as shown in the figure below.

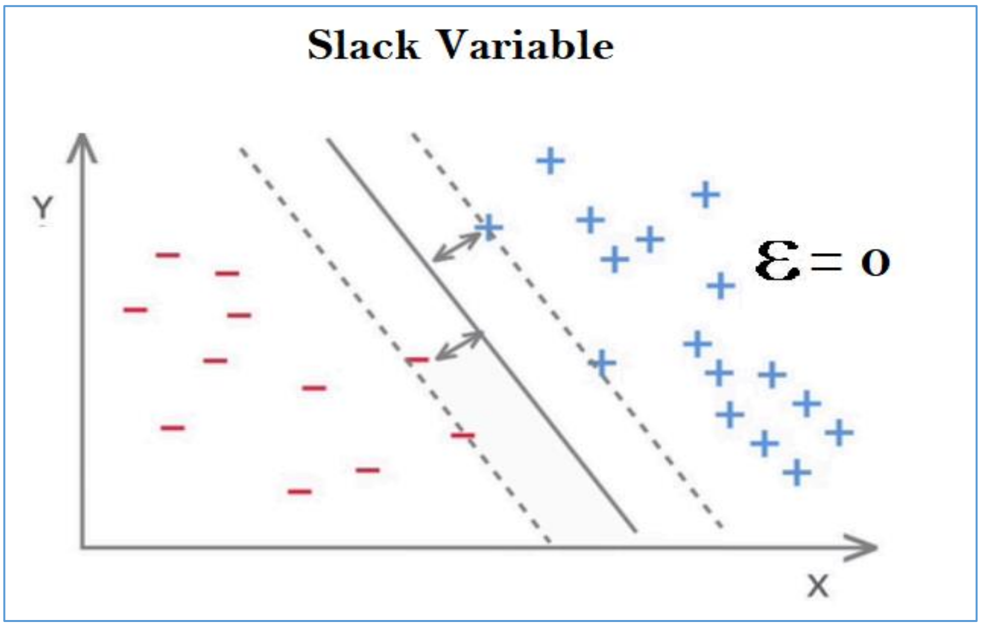

Figure 8: Slack Variable

But if you draw a Support Vector Classifier in such a way that it only violates the margin, i.e. 0 < Epsilon(ϵ) < 1, the observations classify correctly as shown in the figure below.

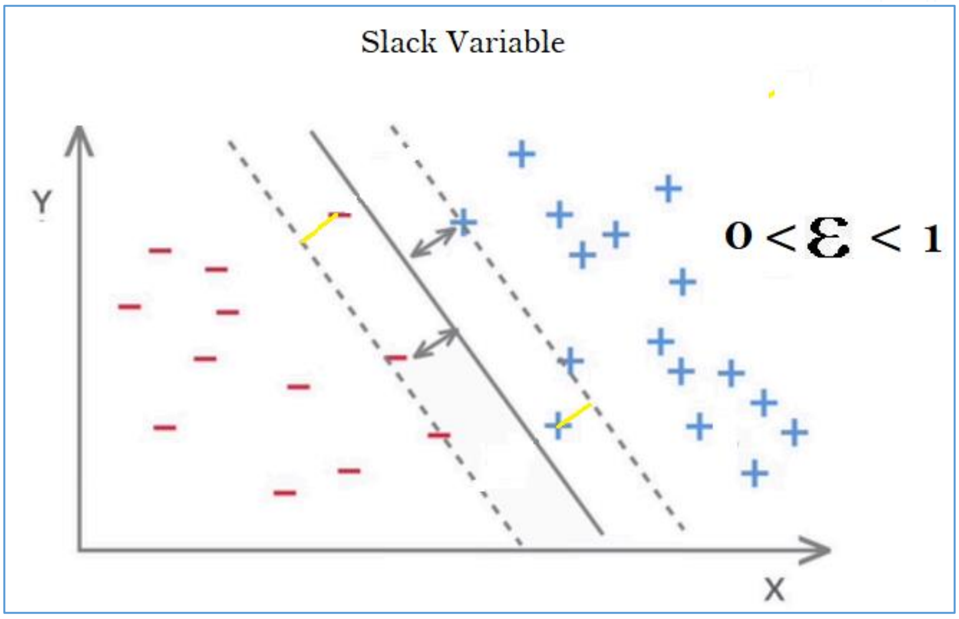

Figure 9: Slack Variables

But if the data points violate the hyperplane, i.e. Epsilon(ϵ) > 1, then the observation is on the wrong side of the hyperplane as shown in the figure below.

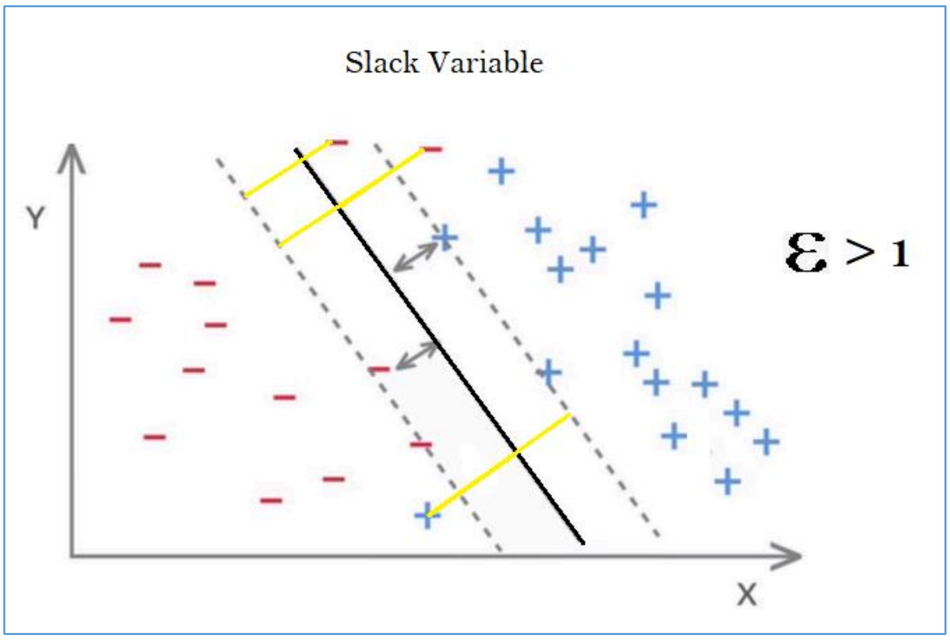

Figure 10: Slack Variable

So you can see that

- Each data point has a slack value associated to it according to where the point is located.
- The value of slack lies between 0 and +infinity.

Lower values of slack are better than higher values (slack = 0 implies a correct classification, but slack > 1 implies an incorrect classification, whereas slack within 0 and 1 classifies correctly but violates the margin).

Cost of Misclassification

The cost of misclassification is greater than or equal to the summation of all the epsilons of each data point, and is denoted by cost or 'C'.

$$\boxed{\sum \epsilon_i \leq C.}$$

Once you understand the **notion of the slack variable**, you can easily compare the two **Support Vector Classifiers**. You can measure the summation of all the epsilons(ϵ) of both the hyperplanes and choose the best one that gives you the least sum of epsilons(ϵ). The summation of all the epsilons of each data point is denoted by cost or 'C', i.e. cost of misclassification.

When **C is large**, the **slack variables** can be large, i.e. you allow a larger number of data points to be misclassified or to violate the margin. So you get a hyperplane where the margin is wide and misclassifications are allowed. In this case, the model is flexible, more generalisable, and less likely to overfit. In other words, it has a high bias.

On the other hand, when **C is small**, you force the individual slack variables to be small, i.e. you do not allow many data points to fall on the wrong side of the margin or the hyperplane. So, the margin is narrow and there are few **misclassifications**. In this case, the model is less flexible, less generalisable, and more likely to overfit. In other words, it has a **high variance**.

Kernels

Kernels are one of the most interesting inventions in machine learning, partly because they were born through the creative imagination of mathematicians, and partly because of their utility in **dealing with nonlinear data sets.**

So far, you learnt about hyperplanes, the Maximal Margin Classifier, and the Support Vector Classifier. All of these are linear models (since they use linear hyperplanes to separate the classes). However, many real-world data sets **are not separable by linear boundaries**. For instance, what if the distribution of data points looks like the figure given below?

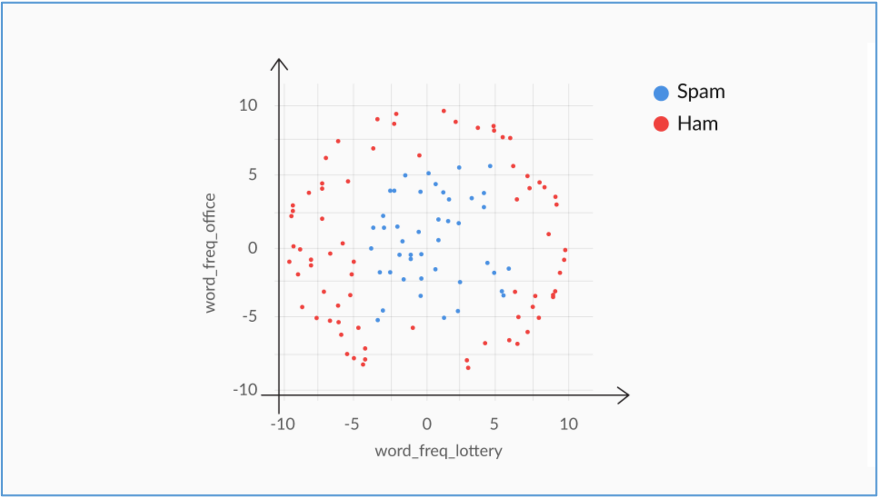

Figure-11: Intermingled Data

You'll agree that it is not possible to imagine a linear hyperplane (a line in 2D) that separates the red and blue points reasonably well. Thus, you need to tweak the linear SVM model and enable it to incorporate nonlinearity in some way. Kernels serve this purpose — they enable the linear SVM model to separate nonlinearly separable data points.

Mapping Nonlinear Data to Linear Data

You can transform nonlinear boundaries to linear boundaries by applying certain functions to the original attributes. The original space (X, Y) is called the **attribute space**, and the transformed space (X', Y') is called the **feature space**.

Let's say you want to classify emails into 'spam' or 'ham' on the basis of two attributes — *'word_freq_office'* (X) and '*word_freq_lottery*' (Y). The following plot shows the data set which is clearly nonlinear.

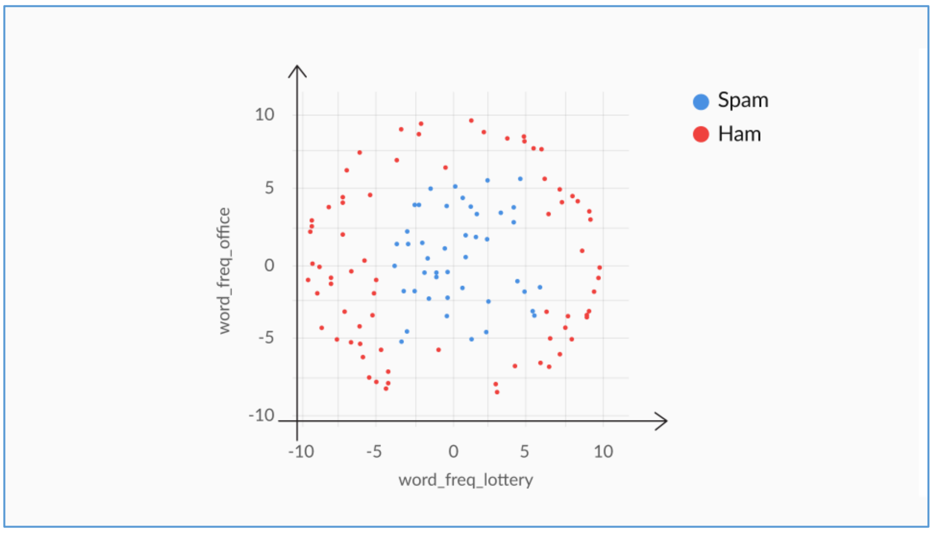

Figure 12: Nonlinear Distribution

To convert this data set into a linearly separable one, a simple transformation into a new feature space (X', Y') can be made. For now, don't worry about the math behind the transformation. You may almost never need to manually transform data sets. Just assume that some appropriate transformation from (X, Y) to (X', Y') can make the data linearly separable.

In the original attribute space, notice that the observations are distributed in a circular fashion. This gives you a hint that the transformation should convert the circular distribution to a linear distribution.

- word_freq_office(X') = (word_freq_office(X)−a)^2
- word_freq_office'(Y') = (word_freq_office(Y)−b)^2

Feature Transformation

The process of transforming the original attributes into a new feature space is called 'feature transformation'. However, as the number of attributes increases, there is an exponential increase in the number of dimensions in the transformed feature space. Suppose you have four variables in your data set; then, considering only a polynomial transformation of degree 2, you end up making **15 features** in the new feature space, as shown in the figure below.

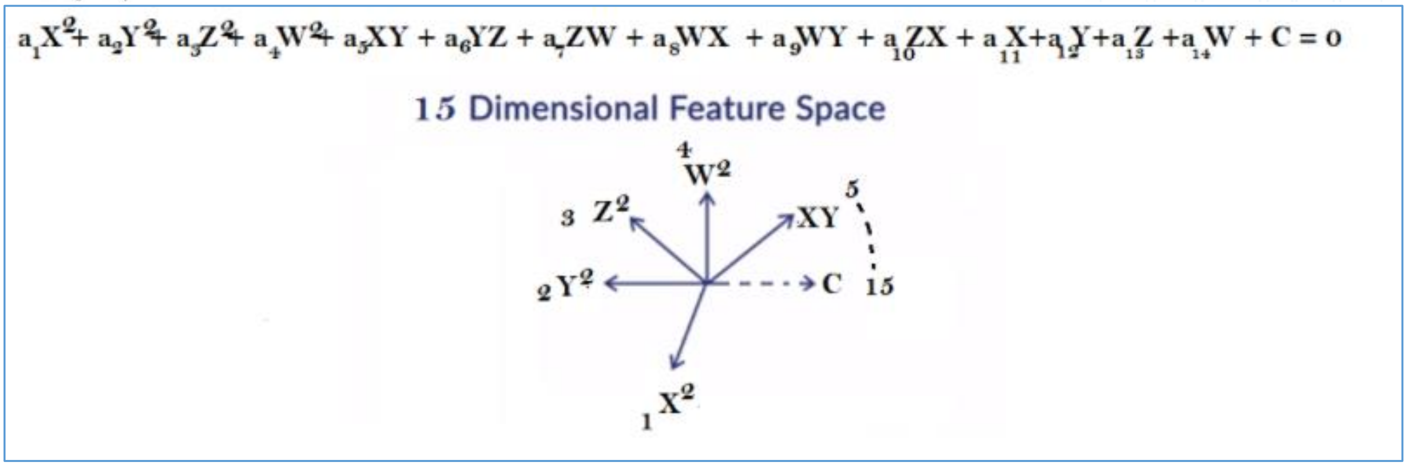
$a_1X^2 + a_2Y^2 + a_3Z^2 + a_4W^2 + a_5XY + a_6YZ + a_7ZW + a_8WX + a_9WY + a_{10}ZX + a_{11}X + a_{12}Y + a_{13}Z + a_{14}W + C = 0$
15 Dimensional Feature Space
4 W^2
3 Z^2
5 XY
2 Y^2
C 15
1 X^2

Figure 13: 4D Feature Space

Kernel Trick

As feature transformation results in a large number of features, it makes the modelling (i.e. the learning process) computationally expensive. The use of kernels resolves this issue. The key fact that makes the kernel trick possible is that to find the best-fit model, the learning algorithm only needs the inner products of the observations ($X^T_i.X_j$). It never uses the individual data points X1, X2, etc. in isolation.

Think of a kernel as a **black box** as shown in the figure below. The attributes are passed on the black box, and it returns the linear boundaries for the classification of the nonlinear data implicitly.

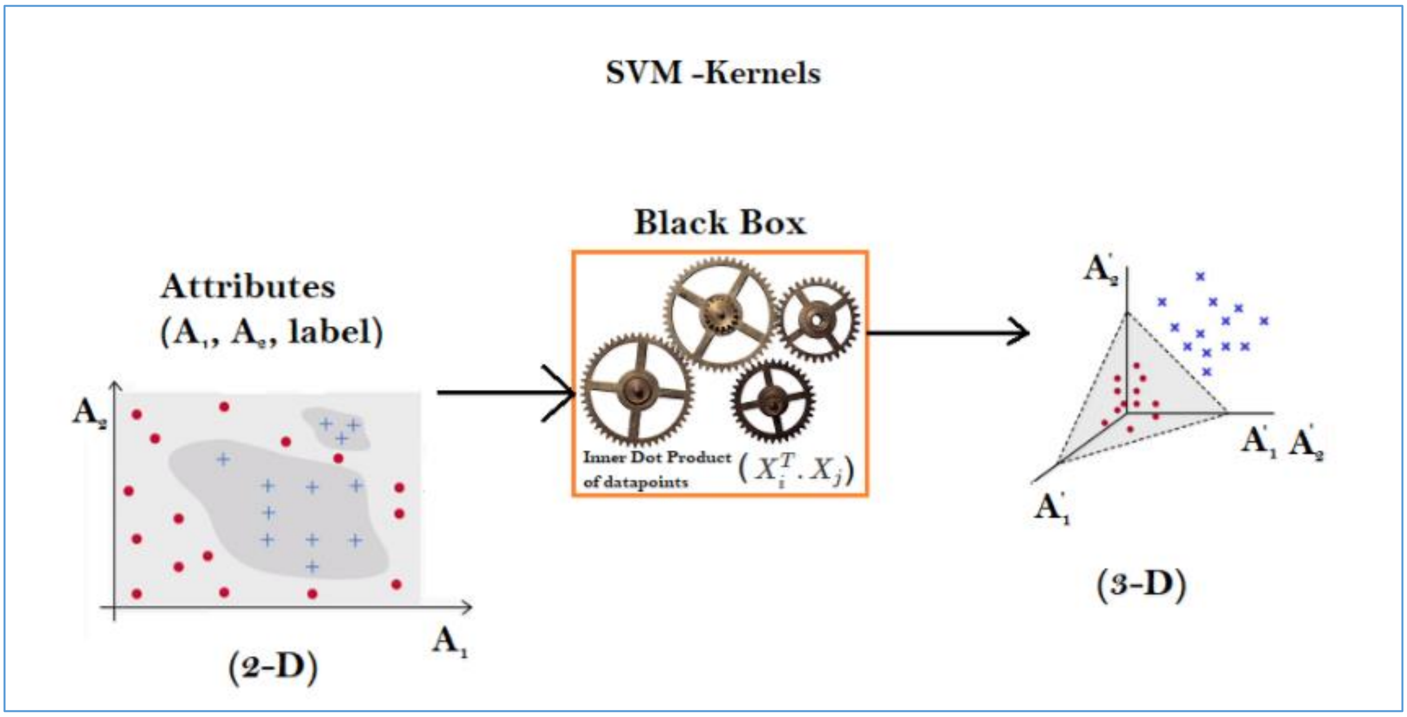

Figure 14: Black Box

Kernel functions use this fact to bypass the explicit transformation process from the attribute space to the feature space, and they rather do it implicitly. The benefit of an implicit transformation is that now, you do not need to

- Manually find the mathematical transformation needed to convert a nonlinear feature space to a linear feature space
- Perform computationally heavy transformations

In practice, you only need to know that kernels are functions that help you transform nonlinear data sets. Given a data set, you can try various kernels and choose the one that produces the best model. The **three most popular** types of kernel functions are

- **The linear kernel**: This is the same as the Support Vector Classifier or the hyperplane, without any transformation at all.
- **The polynomial kernel**: This kernel function is capable of creating nonlinear, polynomial decision boundaries.
- **The radial basis function (RBF) kernel**: This is the most complex kernel function; it is capable of transforming highly nonlinear feature spaces to linear ones. It is even capable of creating elliptical (i.e. enclosed) decision boundaries.

The three types of kernel functions, shown in the figures below, represent the typical decision boundaries they are capable of creating. Notice that the linear kernel is same as the vanilla hyperplane, the polynomial kernel can produce polynomial shaped nonlinear boundaries, and the RBF kernel can produce highly nonlinear, ellipsoid boundaries.

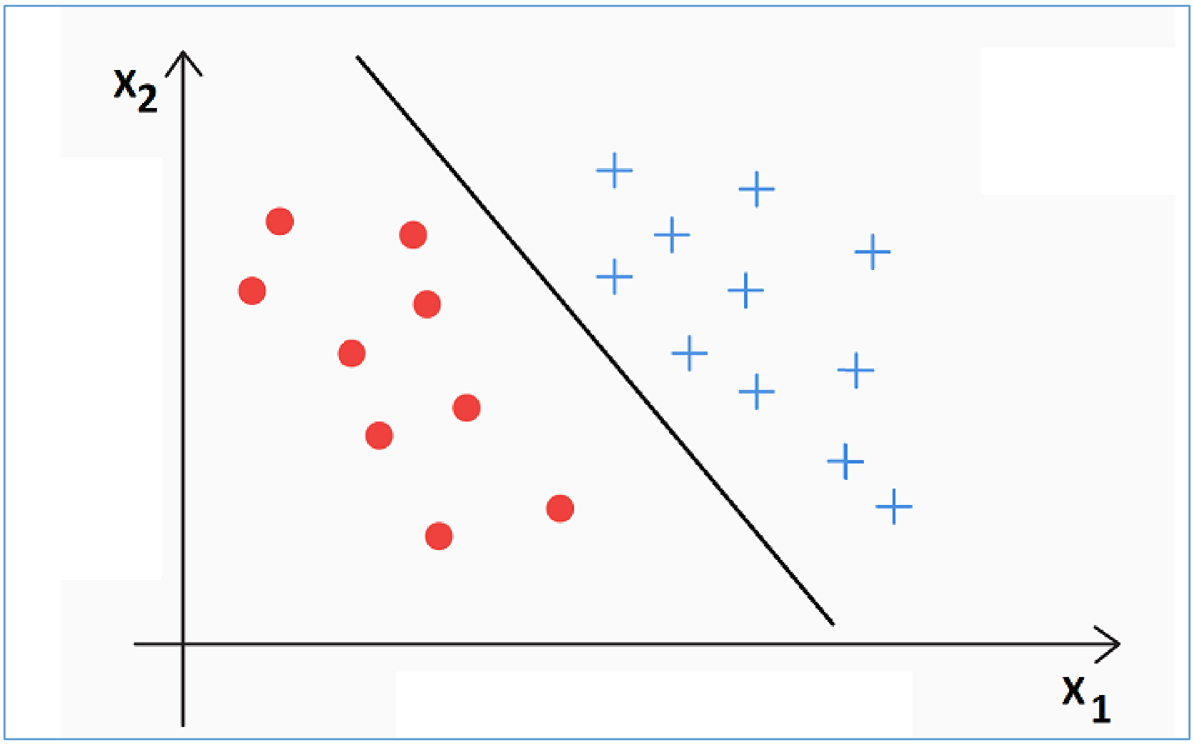

Figure 15:Linear Kernel (Hyperplane)

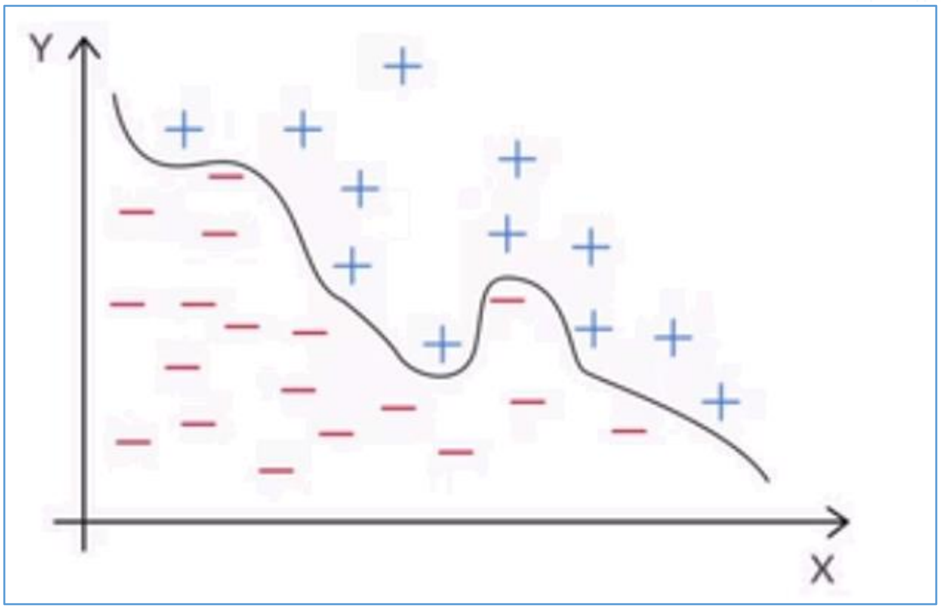

Figure 16:Polynomial Kernel

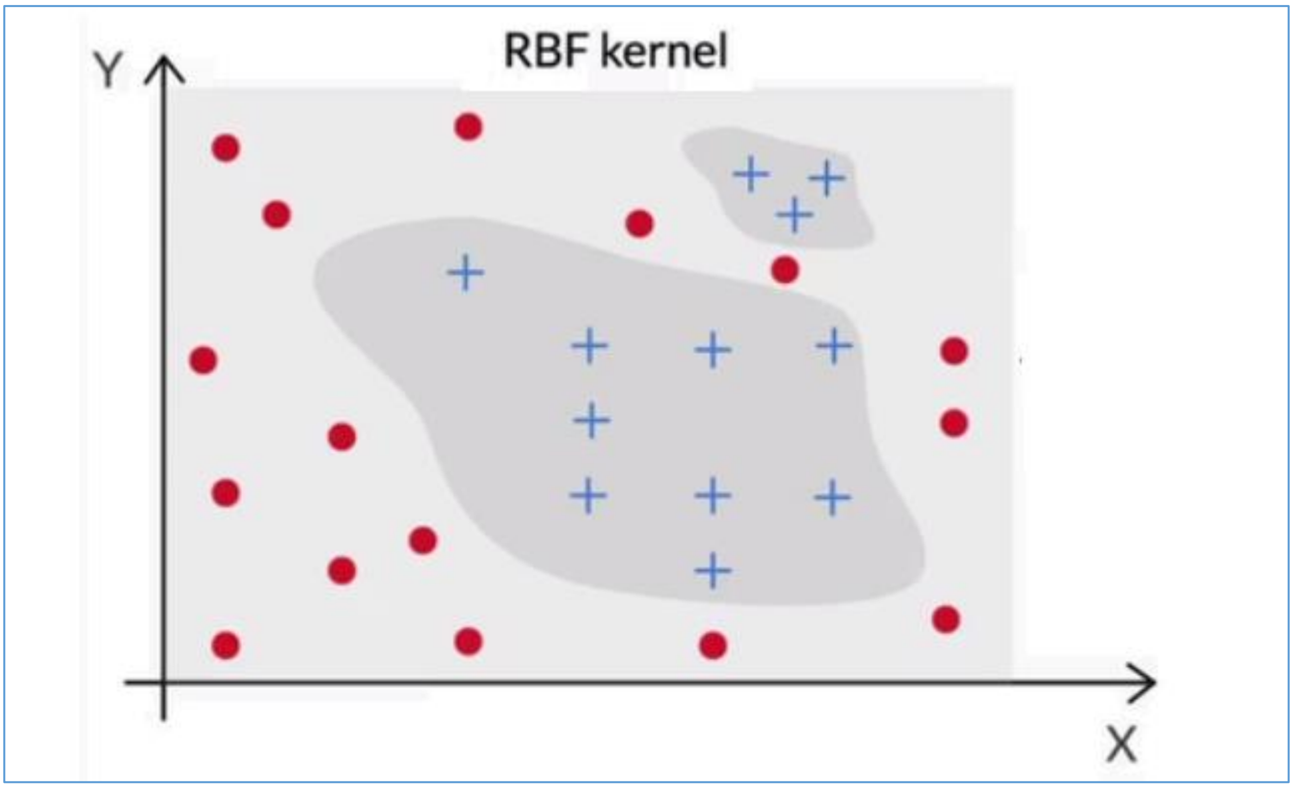

Figure 17: RBF Kernel

Kernel Parameter

In nonlinear kernels such as **the RBF**, you use the parameter **sigma** to control the amount of nonlinearity in the model. The higher the value of sigma, the more is the nonlinearity introduced; the lower the value of sigma, the lesser is the nonlinearity. It is also denoted as gamma in some texts and packages. Apart from sigma, you also have the hyperparameter C or the cost (with all types of kernels).

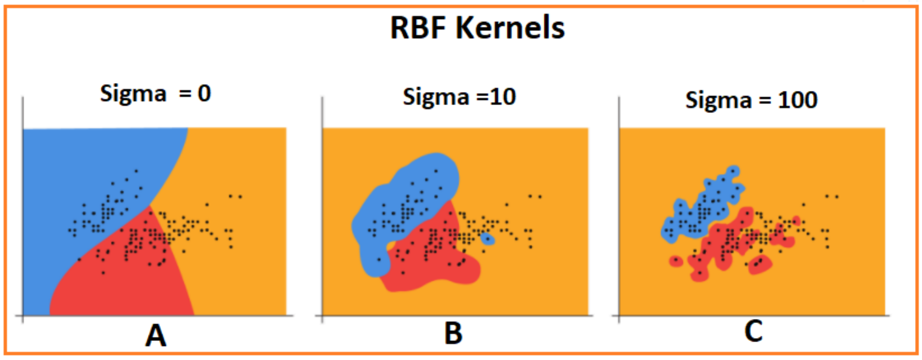

Figure 18: RBF Kernels

The plot above shows three RBF kernels with different values of sigma. When sigma is high, more nonlinearity is added. When you increase the sigma from 10 to 100, the nonlinearity is further increased, and all the training points are correctly mapped, as is visible in the highly complex decision boundaries in figure 1(C). However, this results in a highly biased model that may overfit the training set. Like most other hyperparameters, it is advisable to tune the value of sigma using cross-validation.

Choosing the appropriate kernel is important for building a model of optimum complexity. If the kernel is highly nonlinear, the model is likely to overfit. On the other hand, if the kernel is too simple, then it may not fit the training data well.

Usually, it is difficult to choose the appropriate kernel by visualising the data or using an exploratory analysis. Thus, cross-validation (or hit-and-trial, if you are only choosing from 2-3 types of kernels) is often a good strategy.