A Complete Guide To Support Vector Machines(SVMs)

KUSHAL CHAKRABORTY · Follow

12 min read · Jun 29, 2019

1. Introduction

Support Vector Machine is a popular Machine Learning algorithm which became popular in the late 90 s. It is a **supervised** machine learning algorithm which can be used for both **classification** and **regression**.

In terms of classification, it is a discriminative classifier which is used to find the optimal hyper-plane in order to categorize data into different classes. In two dimensional space, this optimal hyper-plane can be thought of as a line dividing the space into two parts : where one part of the space contains datapoints which belong to one class while other part of the space contains datapoints which belong to the other class. The concept of lines acting as a classifier is only true if the data points are linearly separable. SVMs can also be used to find the optimal curve which can be used to classify the data points that are not linearly separable.

In terms of regression, it can be used to fit the curve or the line to a series of datapoints .

We will explain each of these concepts with examples in the following sections.

2. Explanation of Concepts with examples

In this section we will understand the above definitions and concepts in depth with some examples.

For our convenience, we will explain the concepts with respect to two dimensional space and lines which can be extended to greater than two dimensional space and hyper-planes.

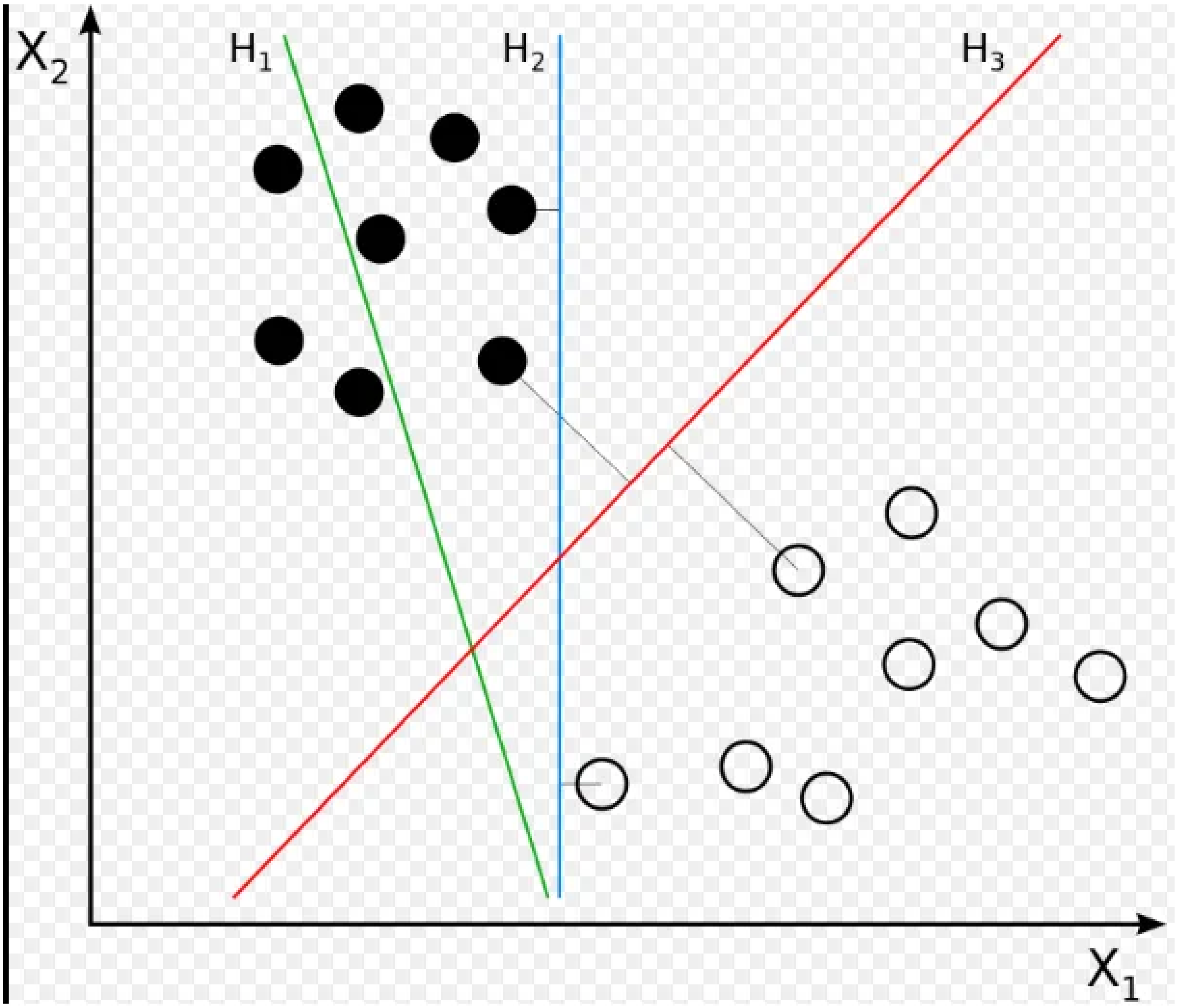


Figure 1: H1 does not separate the classes. H2 does, but only with a small margin. H3 separates them with the maximal margin.(Source **:** www.wikipediacom/SVM**)**

Let us consider the above figure. We have two kinds of datapoints here. Consider that the datapoints marked in black color filled belong to positive class and the datapoints marked with black color border belong to negative class. We have three straight lines which can divide the datapoints two classes, but the main question is : **" Which straight line is the best for the classification task ?"**.

2.1 Geometric Intuition about SVMs

In previous sections we have been using the term **"optimal hyper-plane".** In this section we will be explaining what do we mean by **"optimal hyper-plane".**

By optimal hyper-plane we are referring to that separating hyper-plane from which the data points of classes are at farthest distance away on either side .

The optimal hyper-plane is defined by the plane that maximizes the perpendicular distance between the hyper-plane and the closest samples. This perpendicular distance can be spanned with support vectors.

Now we will be introducing a term called “**margin**”. Let us consider the below figure 2.

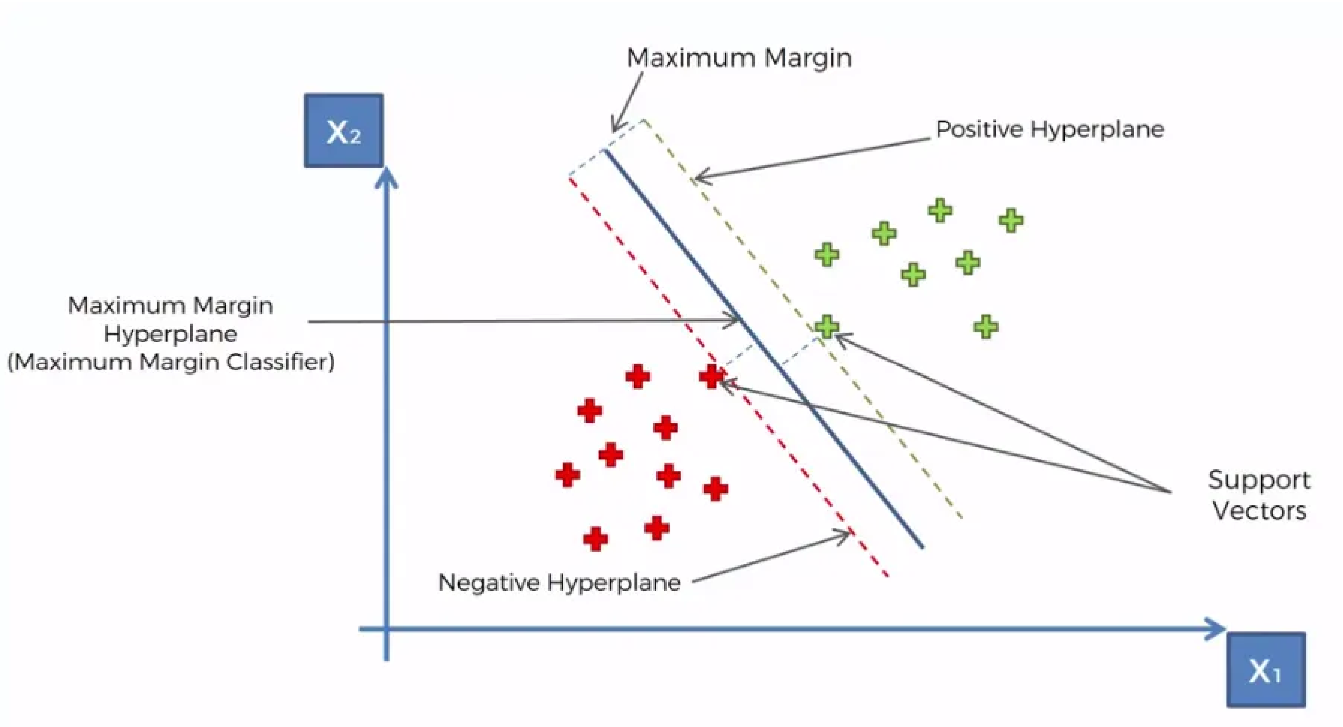


Figure 2 : Figure showing positive ,negative and separating hyper-planes, support vectors and margin. (Source :https://1.bp.com/-K8qVBF8FKpk/WVnU0CDKPzI/AAAAAAAABy)

We can see that there are three hyper-planes:

- The hyper-plane that is touching the points of the positive class is called the **positive hyper-plane.**
- The hyper-plane that is touching the points of the negative class is called the **negative hyper-plane**.
- The hyper-plane that is situated in between the positive and negative class is called the **separating hyper-plane.**
- All these three hyper-planes are parallel to each other.

The distance between the positive and negative hyper-plane is called the “**margin**”. If we maximize the margin, then positive and negative points are quite far away from each other as well as from the separating hyper-plane. Hence the accuracy of the classification task increases. The wider the margin, the better it is for the classification task. As margin increases, the "**generalization accuracy**" (**Accuracy of the model on future unseen data points**) increases.

SVMs try to find a hyper-plane, that maximizes the margin. Hence the optimal or separating hyper-plane is also called "**margin-maximizing hyper-plane**".

Hence in figure 1, the H3 hyper-plane is the best hyper-plane.

2.2 Support Vectors

Support Vectors are those data points that touch the positive and negative hyper-planes. In Figure 2, we can see that there are some data points which first touch the positive and negative hyper-planes, these data points are known as "**Support Vectors**".

Note : *For future unseen data points we will use the position , direction and side of the data point, only with respect to the separating hyper-plane to decide the class of the data points(and not the positive and negative hyper-planes).*

2.3 Alternative Geometric Intuition about SVMs

In this section, we will be using the concept of "**Convex Hull**" in order to find the margin-maximizing hyper-plane.

Convex Hull can be defined as the smallest convex polygon such that all its points are either inside the polygon or on the polygon.

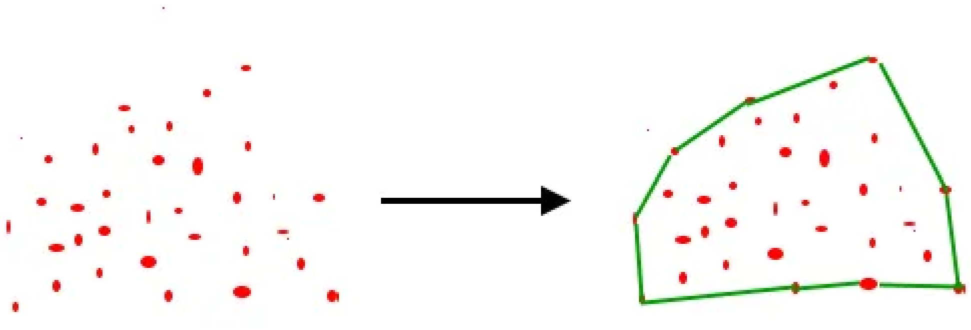


Figure 3 :Diagram of Convex Hull (Source : https://www.geeksforgeeks.org)

Convex polygon is basically a polygon such that all the points on the shortest line connecting any two points (situated inside or on the polygon) lies inside or on the polygon.

In order to find the margin-maximizing hyper-plane using the concept of Convex Hull, we will use the following steps :

- First draw the convex hull for positive data points.
- Second draw the convex hull for negative data points.
- Third find the shortest line or hyper-plane connecting the hulls.
- Finally , draw the line or hyper-plane that bisects the line or hyper-plane (drawn in the third step). This line or hyper-plane obtained is referred to as the **“margin-maximizing hyper-plane”** .

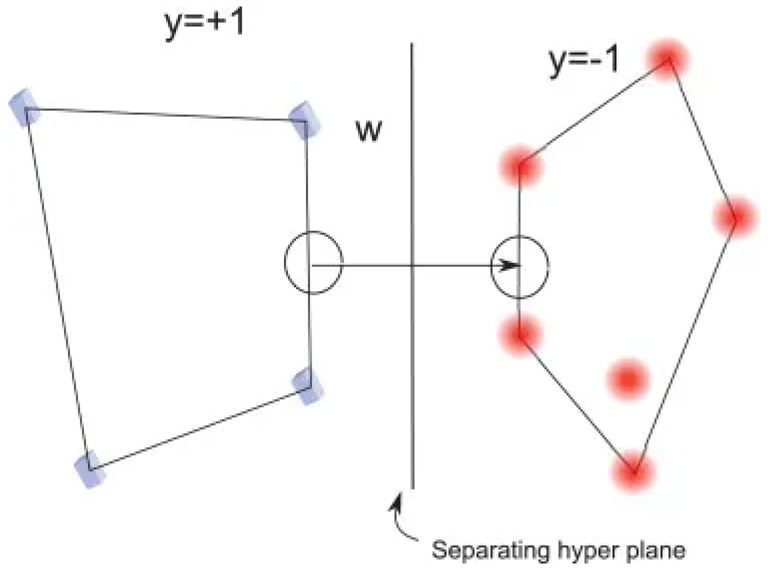


Figure 4 : Separating Hyper-plane obtained using Convex Hull concept (Source: https://sciencedirect.com)

3. Mathematical Formulation of SVMs

Let us consider that the positive and negative hyper-planes are at unit distances away from the separating hyper-plane.

3.1 Hard Margin SVM

Let the equation of the separating hyperplane be :

$$\pi \; : \; w^T . x + b \; = \; 0 \tag{1}$$

$$if \qquad \pi^+ \; : \; w^T . x + b \; = \; 1 \tag{2}$$

be the equation of the positive hyperplane

and

$$\pi^- \; : \; w^T . x + b \; = \; -1 \tag{3}$$

be the equation of the negative hyperplane

$$then \; margin \; = \frac{2}{||w||} \tag{4}$$

For SVMs we can write the constraint optimization problem as :

$$w^*, b^* \; = argmax_{w,b} \frac{2}{||w||} \tag{5}$$

$$such \; that \; \forall i, \; y_i\left(w^T x_i + b\right) \; \geqq 1$$

Applicable if data is linearly separable , all (+)ve pts lie on one side of π *and all* $(-)$*ve pts lie on other side of* π *and there are no datapoints in between the* π^+ *and* π^-.

Equal to 1 is for the Support Vectors.

3.2 Soft Margin SVM

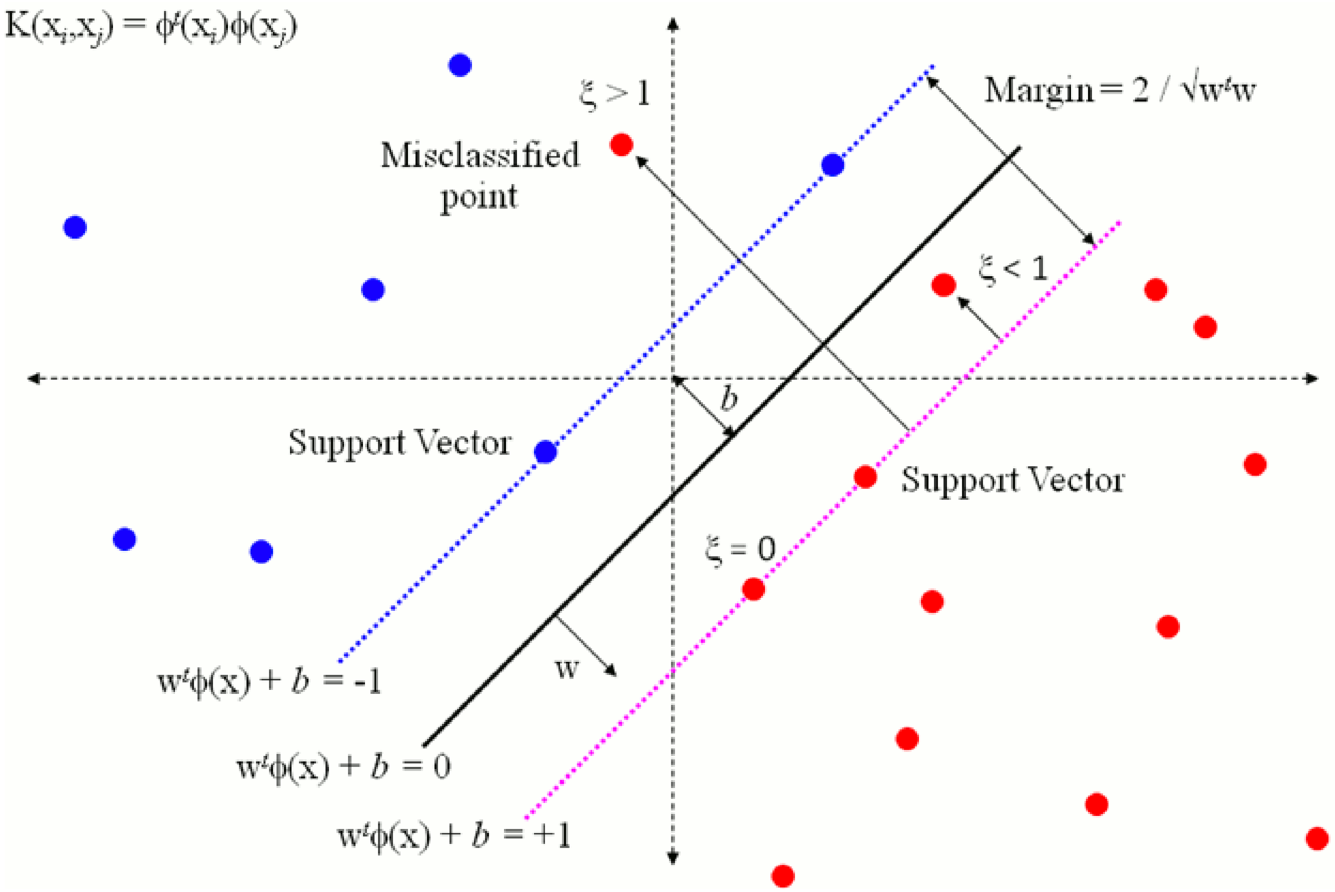
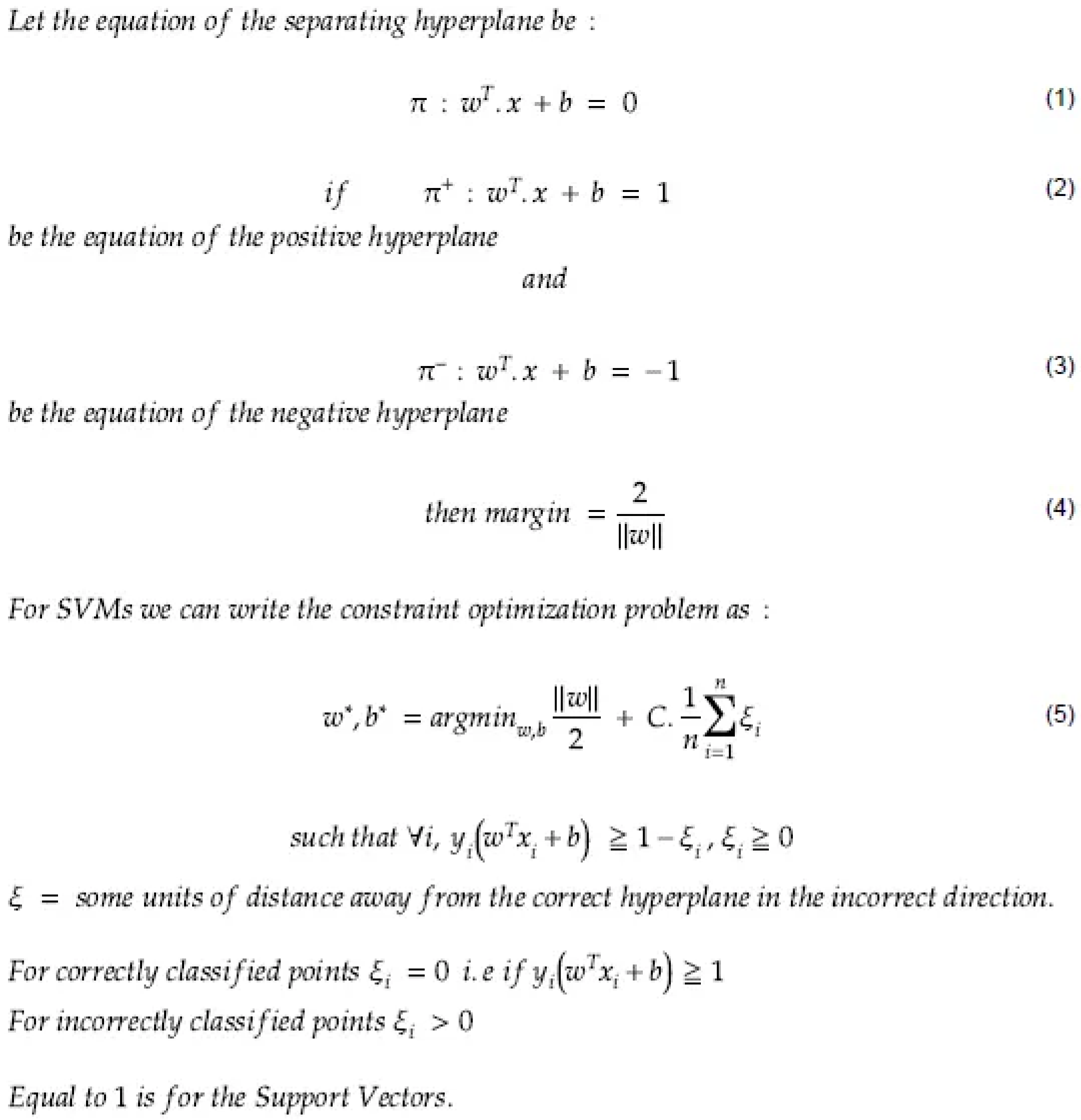


Figure 5: Soft margin SVM



Let the equation of the separating hyperplane be :

$$\pi : w^T.x + b = 0 \tag{1}$$

$$if \qquad \pi^+ : w^T.x + b = 1 \tag{2}$$

be the equation of the positive hyperplane

and

$$\pi^- : w^T.x + b = -1 \tag{3}$$

be the equation of the negative hyperplane

$$then\ margin = \frac{2}{||w||} \tag{4}$$

For SVMs we can write the constraint optimization problem as :

$$w^*, b^* = argmin_{w,b} \frac{||w||}{2} + C.\frac{1}{n}\sum_{i=1}^{n}\xi_i \tag{5}$$

$$such\ that\ \forall i,\ y_i\left(w^T x_i + b\right) \geqq 1 - \xi_i\,,\ \xi_i \geqq 0$$

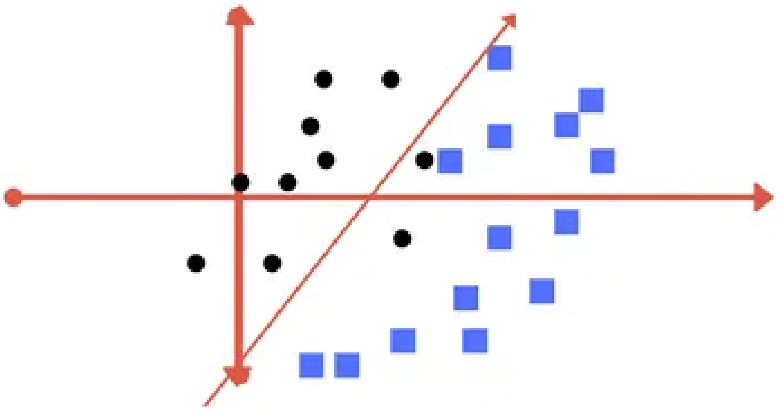
ξ = *some units of distance away from the correct hyperplane in the incorrect direction.*

For correctly classified points $\xi_i = 0$ *i.e if* $y_i\left(w^T x_i + b\right) \geqq 1$
For incorrectly classified points $\xi_i > 0$

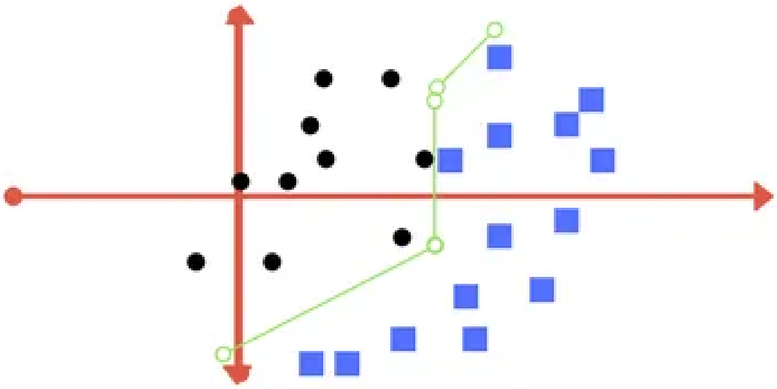
Equal to 1 is for the Support Vectors.

In equation 5, the first portion of the equation before the '+' sign is referred to as the **'regularization'** and the second portion is referred to as the **'Hinge Loss'**.

'C' is the hyper-parameter which is always a positive value. If 'C' increases, then overfitting increases and if 'C' decreases, then underfitting increases. For large values of 'C', the optimization will choose a smaller-margin hyper-plane if that hyper-plane does a better job of getting all the training points classified correctly. Conversely, a very small value of 'C' will cause the optimizer to look for a larger-margin separating hyper-plane, even if that hyper-plane misclassifies more points.



Separating hyper-plane with Low value of 'C'. (Source **:** https://jmlr.com/machine-learning-svm)



Separating hyper-plane with High value of 'C'. (Source **:** https://jmlr.com/machine-learning-svm)

4. Hinge Loss Formulation of SVMs

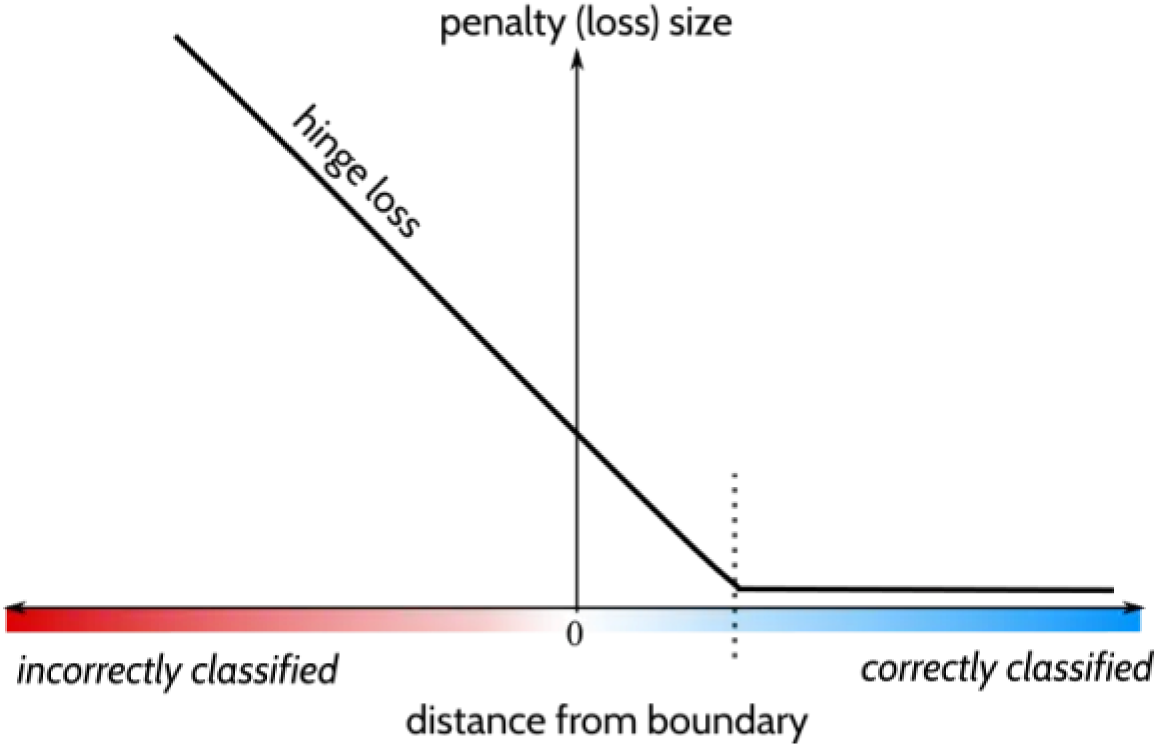
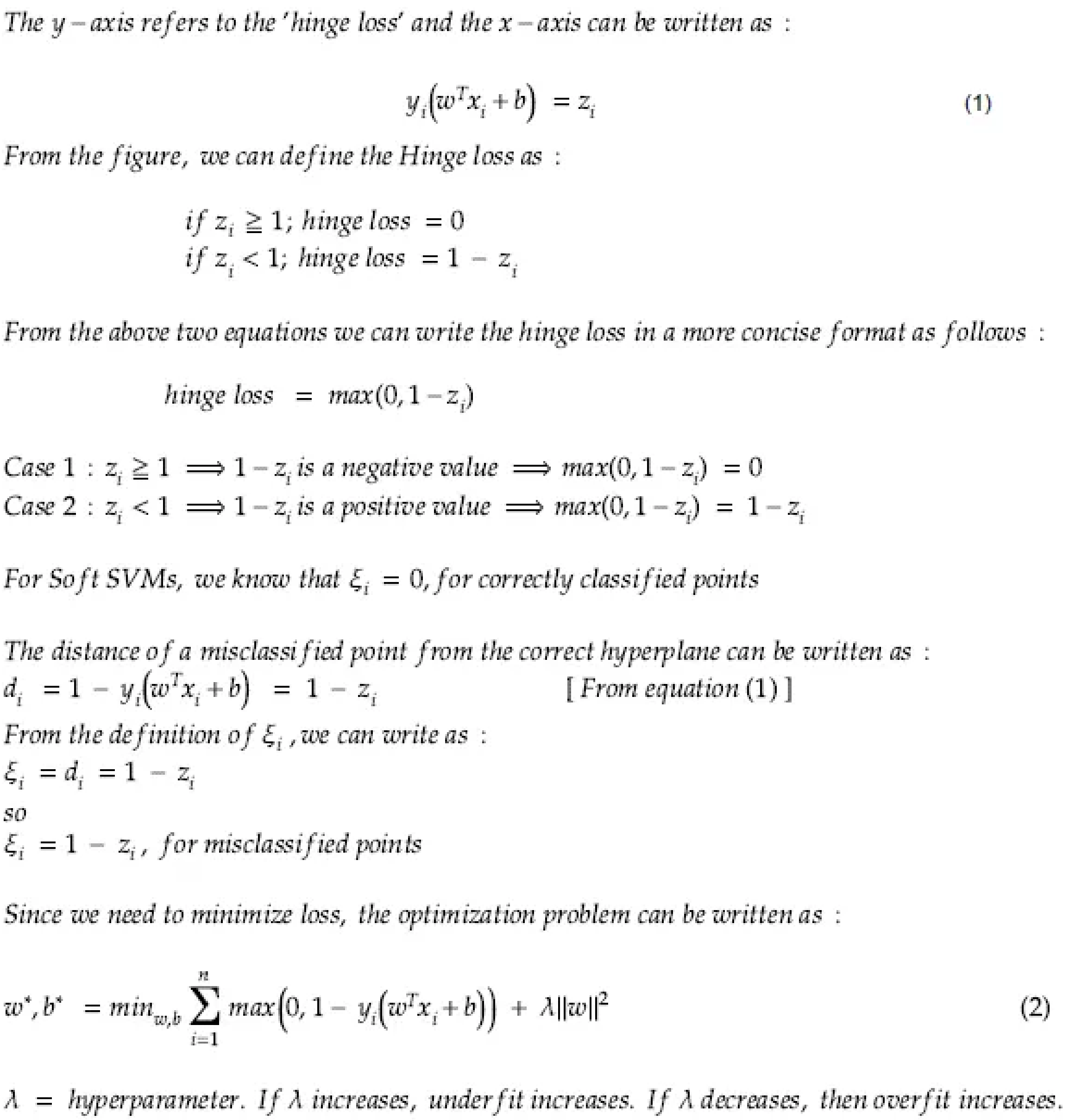


Figure 6 : Graph for Hinge Loss



The $y-$axis refers to the 'hinge loss' and the $x-$axis can be written as :

$$y_i\left(w^T x_i + b\right) \ = z_i \qquad (1)$$

From the figure, we can define the Hinge loss as :

$$\begin{aligned} &if\ z_i \geqq 1;\ hinge\ loss\ = 0 \\ &if\ z_i < 1;\ hinge\ loss\ = 1 - z_i \end{aligned}$$

From the above two equations we can write the hinge loss in a more concise format as follows :

$$hinge\ loss \ = \ max(0, 1 - z_i)$$

$Case\ 1 :\ z_i \geqq 1 \implies 1 - z_i\ is\ a\ negative\ value \implies max(0, 1 - z_i)\ = 0$

$Case\ 2 :\ z_i < 1 \implies 1 - z_i\ is\ a\ positive\ value \implies max(0, 1 - z_i)\ = 1 - z_i$

For Soft SVMs, we know that $\xi_i \ = 0$, for correctly classified points

The distance of a misclassified point from the correct hyperplane can be written as :

$d_i \ = 1 - y_i\left(w^T x_i + b\right) \ = 1 - z_i$ [From equation (1)]

From the definition of ξ_i , we can write as :

$\xi_i \ = d_i \ = 1 - z_i$

so

$\xi_i \ = 1 - z_i$, for misclassified points

Since we need to minimize loss, the optimization problem can be written as :

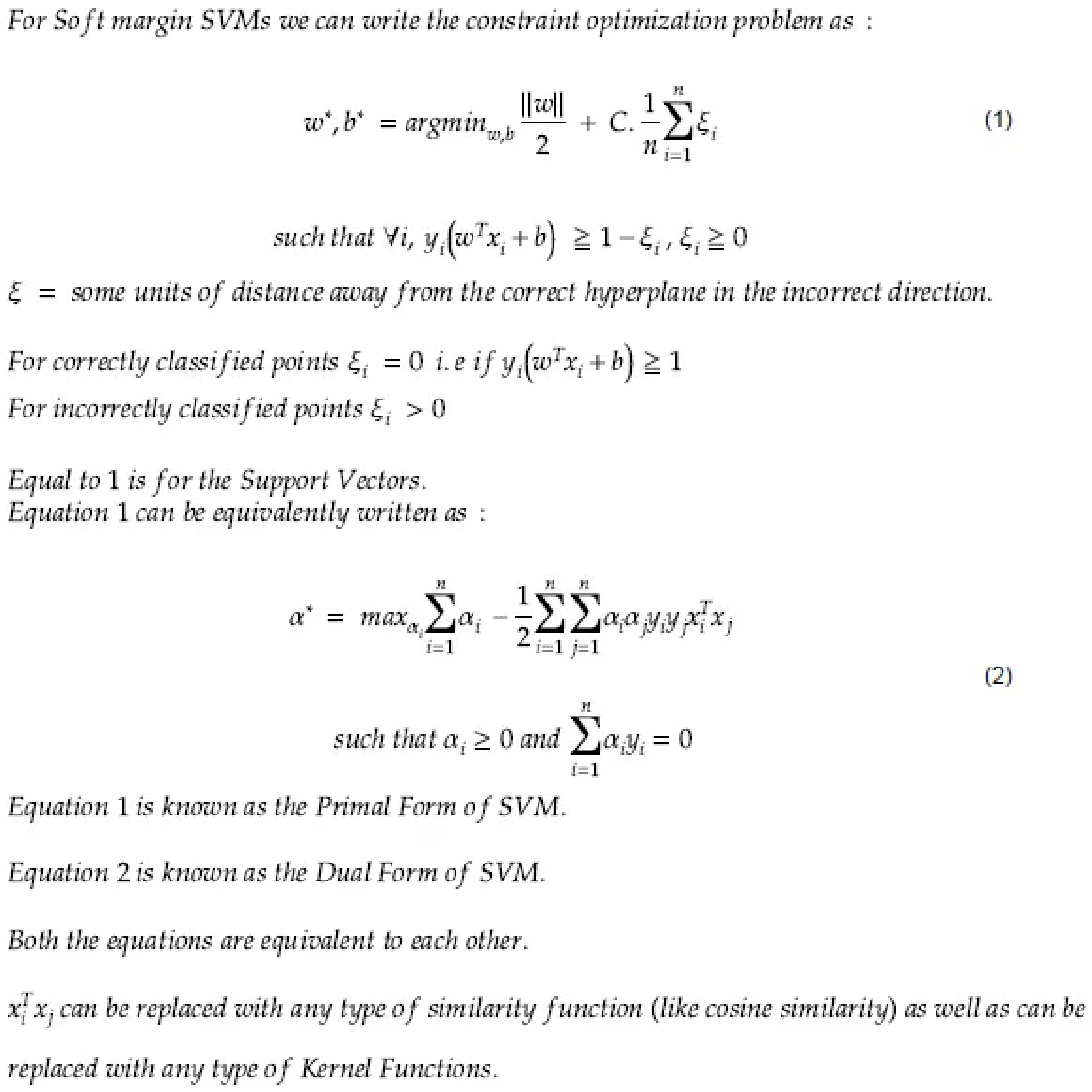
$$w^*, b^* \ = min_{w,b} \sum_{i=1}^{n} max\left(0, 1 - y_i\left(w^T x_i + b\right)\right) \ + \ \lambda\|w\|^2 \qquad (2)$$

λ = hyperparameter. If λ increases, underfit increases. If λ decreases, then overfit increases.

Note : It can be proved that the optimization equation obtained in this section and the optimization equation obtained in the previous section are same. Only C and lambda are inversely related to each other i.e

C = 1/lambda

5. Dual Form of SVM Formulation



For Soft margin SVMs we can write the constraint optimization problem as :

$$w^*, b^* = argmin_{w,b} \frac{||w||}{2} + C.\frac{1}{n}\sum_{i=1}^{n}\xi_i \qquad (1)$$

$$such\ that\ \forall i,\ y_i\left(w^T x_i + b\right) \geqq 1 - \xi_i\ ,\ \xi_i \geqq 0$$

ξ = *some units of distance away from the correct hyperplane in the incorrect direction.*

For correctly classified points $\xi_i = 0$ *i.e if* $y_i\left(w^T x_i + b\right) \geqq 1$

For incorrectly classified points $\xi_i > 0$

Equal to 1 is for the Support Vectors.
Equation 1 can be equivalently written as :

$$\alpha^* = max_{\alpha_i}\sum_{i=1}^{n}\alpha_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\alpha_j y_i y_j x_i^T x_j \qquad (2)$$

$$such\ that\ \alpha_i \geq 0\ and\ \sum_{i=1}^{n}\alpha_i y_i = 0$$

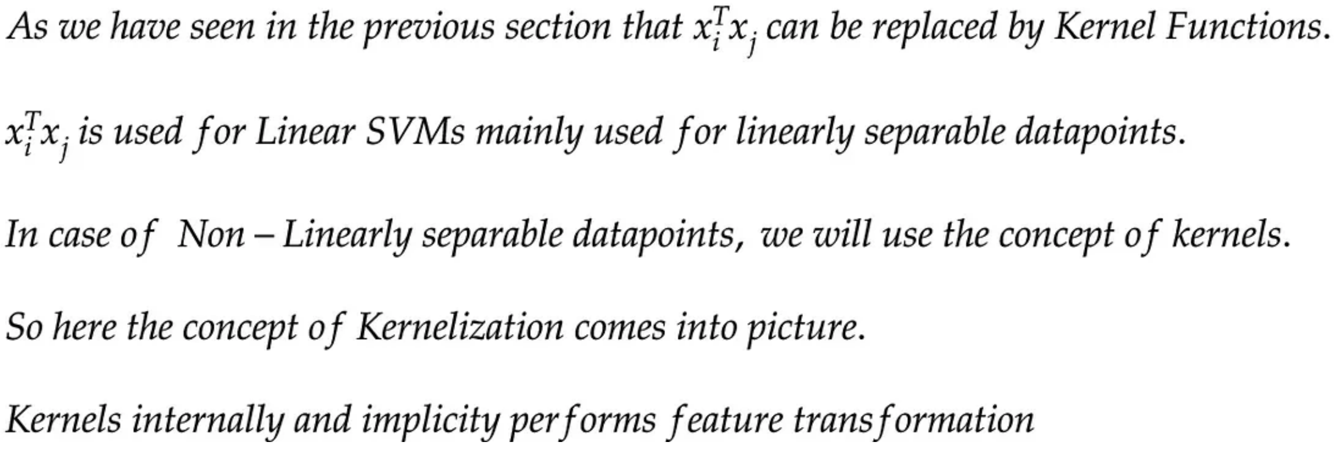
Equation 1 is known as the Primal Form of SVM.

Equation 2 is known as the Dual Form of SVM.

Both the equations are equivalent to each other.

$x_i^T x_j$ *can be replaced with any type of similarity function (like cosine similarity) as well as can be replaced with any type of Kernel Functions.*

6. Kernel Trick in SVM



As we have seen in the previous section that $x_i^T x_j$ *can be replaced by Kernel Functions.*

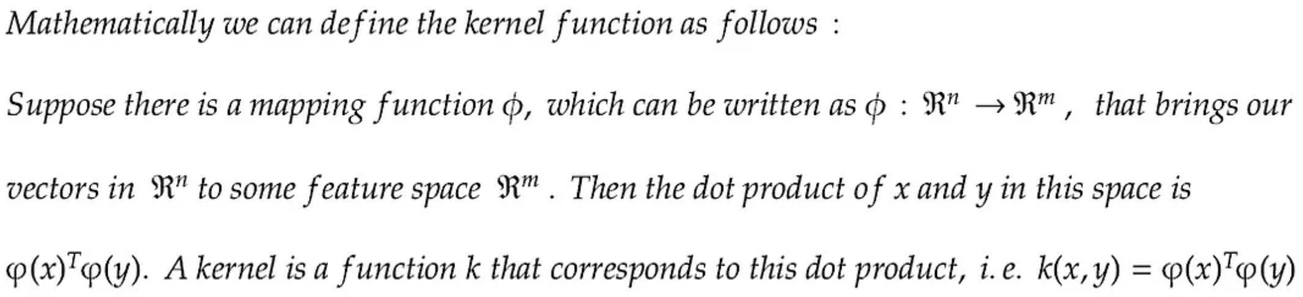
$x_i^T x_j$ *is used for Linear SVMs mainly used for linearly separable datapoints.*

In case of Non – Linearly separable datapoints, we will use the concept of kernels.

So here the concept of Kernelization comes into picture.

Kernels internally and implicity performs feature transformation

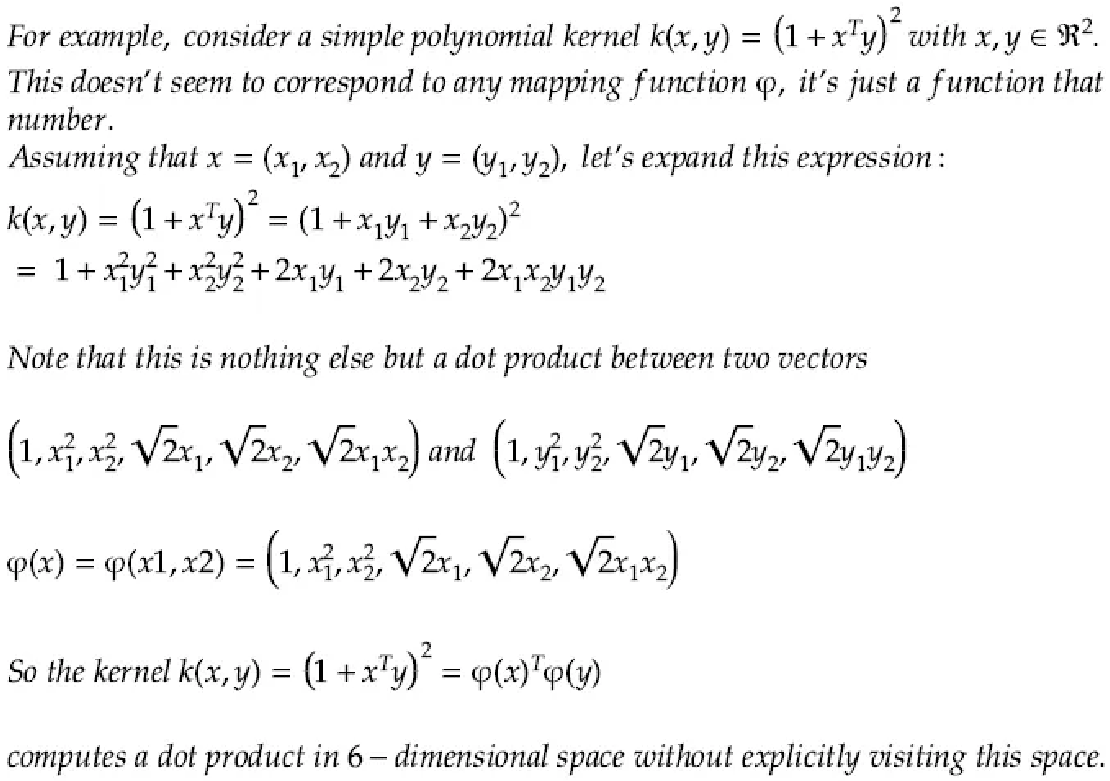
Kernel is a way of computing the dot product of two vectors x and y in some (possibly very high dimensional) feature space, which is why kernel functions are sometimes called “generalized dot product”.



Mathematically we can define the kernel function as follows :

Suppose there is a mapping function ϕ, which can be written as $\phi : \Re^n \to \Re^m$, that brings our vectors in $\Re^n$ to some feature space $\Re^m$. Then the dot product of x and y in this space is $\varphi(x)^T\varphi(y)$. A kernel is a function k that corresponds to this dot product, i.e. $k(x,y) = \varphi(x)^T\varphi(y)$

Kernel is useful because it gives us a way to compute dot products in some feature space without even knowing what this space is and what is phi.



For example, consider a simple polynomial kernel $k(x,y) = \left(1 + x^T y\right)^2$ with $x, y \in \Re^2$.
This doesn't seem to correspond to any mapping function φ, it's just a function that returns a real number.
Assuming that $x = (x_1, x_2)$ and $y = (y_1, y_2)$, let's expand this expression:

$$k(x,y) = \left(1 + x^T y\right)^2 = (1 + x_1 y_1 + x_2 y_2)^2$$
$$= 1 + x_1^2 y_1^2 + x_2^2 y_2^2 + 2x_1 y_1 + 2x_2 y_2 + 2x_1 x_2 y_1 y_2$$

Note that this is nothing else but a dot product between two vectors

$$\left(1, x_1^2, x_2^2, \sqrt{2}x_1, \sqrt{2}x_2, \sqrt{2}x_1 x_2\right) \text{ and } \left(1, y_1^2, y_2^2, \sqrt{2}y_1, \sqrt{2}y_2, \sqrt{2}y_1 y_2\right)$$

$$\varphi(x) = \varphi(x1, x2) = \left(1, x_1^2, x_2^2, \sqrt{2}x_1, \sqrt{2}x_2, \sqrt{2}x_1 x_2\right)$$

So the kernel $k(x,y) = \left(1 + x^T y\right)^2 = \varphi(x)^T\varphi(y)$

computes a dot product in 6 – dimensional space without explicitly visiting this space.

Now consider what if we had data as shown in image below? Clearly, there is no line that can separate the two classes in this x-y plane. So what do we do? We apply transformation and add one more dimension as we call it z-axis. Lets assume value of points on z plane, $w = x^2 + y^2$. In this case we can manipulate it as distance of point from z-origin. Now if we plot in z-axis, a clear separation is visible and a line can be drawn .

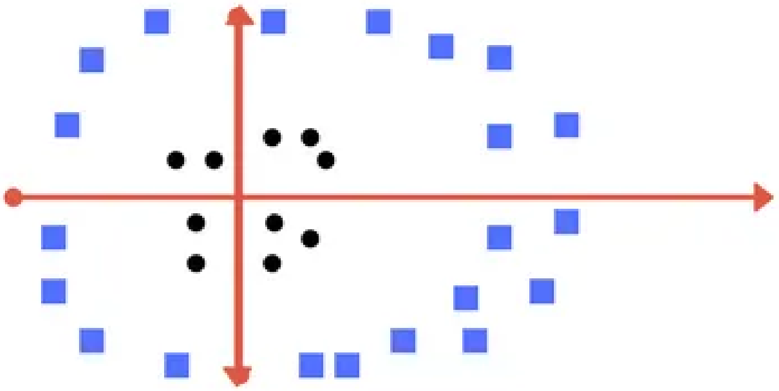


Figure 7 : Can we draw a separating line in this plane? (Source : https://chapter-2-svm-support-vector-machine-theory-f0812effc72)

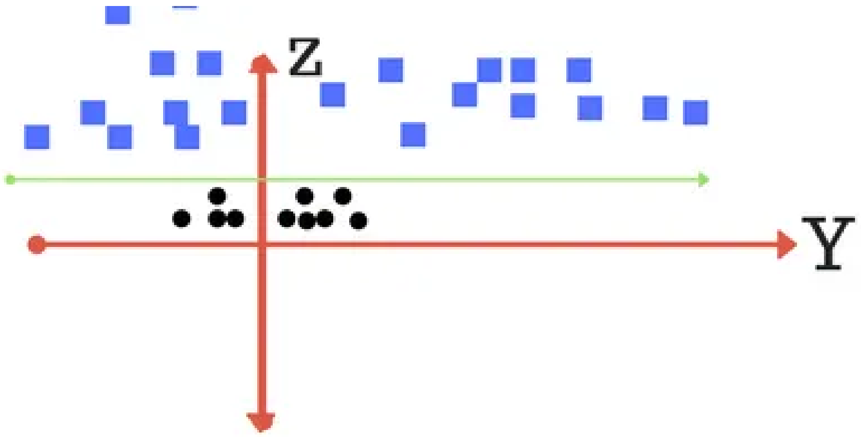


Figure 8 : plot of z-y axis. A separation can be made here.(Source : https://chapter-2-svm-support-vector-machine-theory-f0812effc72)

When we transform back this line to original plane, it maps to circular boundary as shown in below *image* .

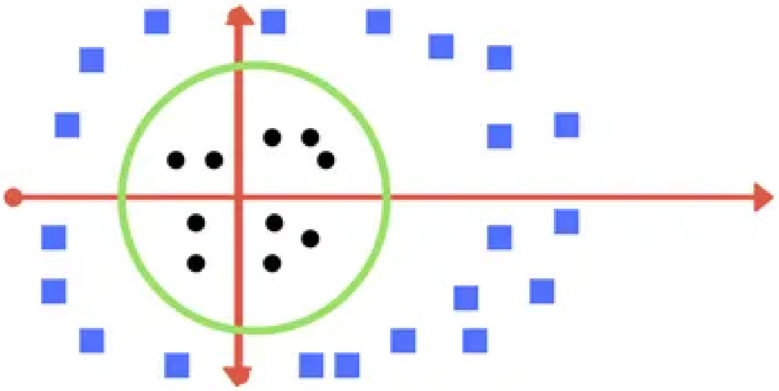
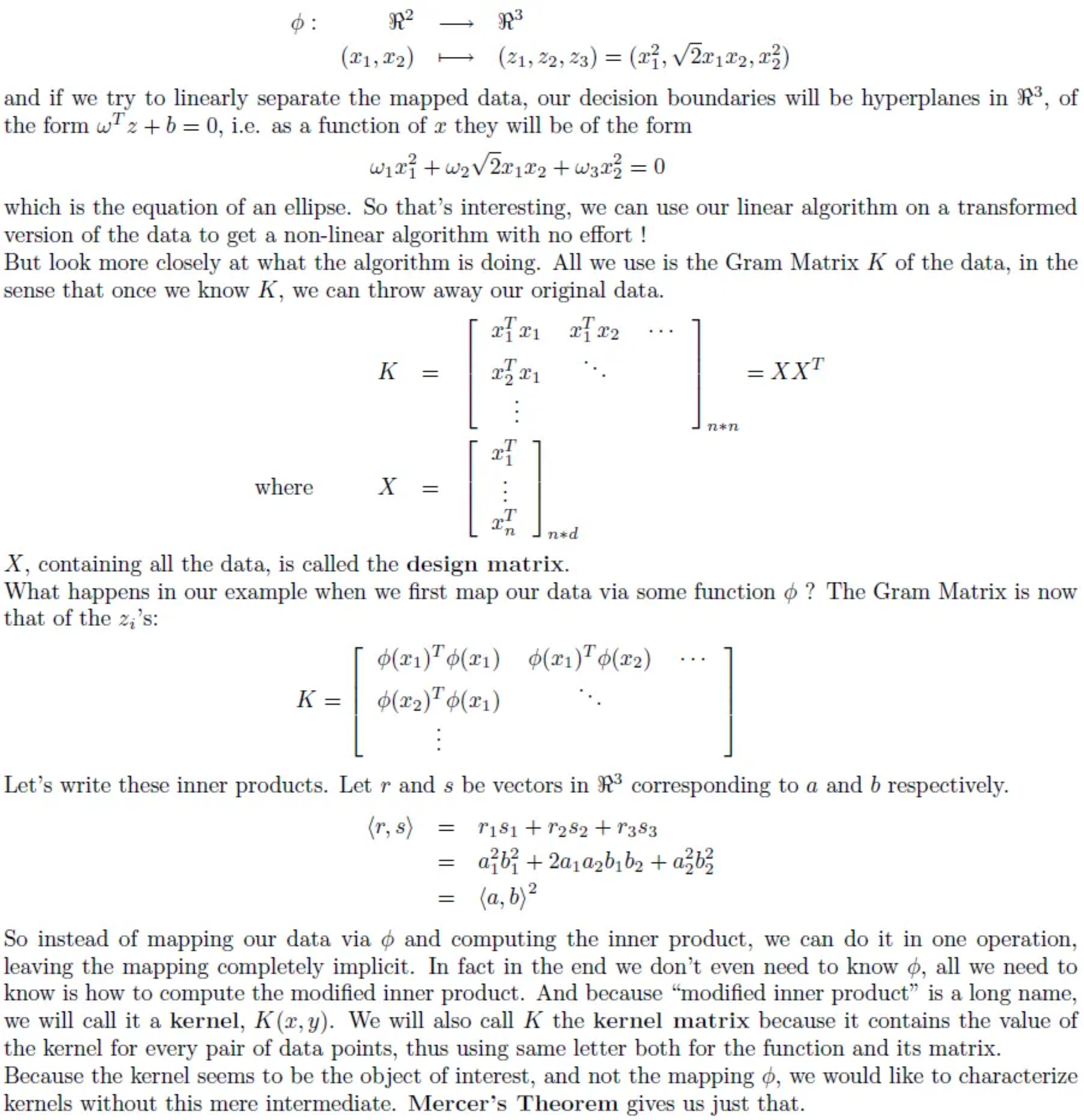


Figure 9 : Transforming back to x-y plane, a line transforms to circle.(Source : https://chapter-2-svm-support-vector-machine-theory-f0812effc72)

For the above example, the kernel will internally and implicitly transform the data which is 2 D to another higher dimensional space where the data will become linearly separable.

What kernalization does is that it takes data which is ‘d’ dimensional and it does a feature transform internally and implicitly using the kernel trick to a dimension ‘d1’ , typically where d1 >d. In ‘d1’ , the data becomes linearly separable.

The figure 7 is clearly not linearly separable, yet if we try to map it to a 3 D space using :



$$\begin{array}{cccc} \phi: & \Re^2 & \longrightarrow & \Re^3 \\ & (x_1, x_2) & \longmapsto & (z_1, z_2, z_3) = (x_1^2, \sqrt{2}x_1x_2, x_2^2) \end{array}$$

and if we try to linearly separate the mapped data, our decision boundaries will be hyperplanes in $\Re^3$, of the form $\omega^T z + b = 0$, i.e. as a function of x they will be of the form

$$\omega_1 x_1^2 + \omega_2\sqrt{2}x_1x_2 + \omega_3 x_2^2 = 0$$

which is the equation of an ellipse. So that’s interesting, we can use our linear algorithm on a transformed version of the data to get a non-linear algorithm with no effort !

But look more closely at what the algorithm is doing. All we use is the Gram Matrix K of the data, in the sense that once we know K, we can throw away our original data.

$$K = \begin{bmatrix} x_1^T x_1 & x_1^T x_2 & \cdots \\ x_2^T x_1 & \ddots & \\ \vdots & & \end{bmatrix}_{n*n} = XX^T$$

$$\text{where} \quad X = \begin{bmatrix} x_1^T \\ \vdots \\ x_n^T \end{bmatrix}_{n*d}$$

X, containing all the data, is called the **design matrix**.

What happens in our example when we first map our data via some function ϕ ? The Gram Matrix is now that of the z_i’s:

$$K = \begin{bmatrix} \phi(x_1)^T\phi(x_1) & \phi(x_1)^T\phi(x_2) & \cdots \\ \phi(x_2)^T\phi(x_1) & \ddots & \\ \vdots & & \end{bmatrix}$$

Let’s write these inner products. Let r and s be vectors in $\Re^3$ corresponding to a and b respectively.

$$\begin{aligned} \langle r, s\rangle &= r_1s_1 + r_2s_2 + r_3s_3 \\ &= a_1^2b_1^2 + 2a_1a_2b_1b_2 + a_2^2b_2^2 \\ &= \langle a, b\rangle^2 \end{aligned}$$

So instead of mapping our data via ϕ and computing the inner product, we can do it in one operation, leaving the mapping completely implicit. In fact in the end we don’t even need to know ϕ, all we need to know is how to compute the modified inner product. And because “modified inner product” is a long name, we will call it a **kernel**, $K(x, y)$. We will also call K the **kernel matrix** because it contains the value of the kernel for every pair of data points, thus using same letter both for the function and its matrix.

Because the kernel seems to be the object of interest, and not the mapping ϕ, we would like to characterize kernels without this mere intermediate. **Mercer’s Theorem** gives us just that.

(Reference : https://people.eecs.berkeley.edu/~jordan/courses/)

If we want to know more about gram matrix , you can refer here.

So if we have a function 'K' defined as follows :

$$K(x_i, x_j) = \phi(x_i) \cdot \phi(x_j)$$

then we just need to know *K* and not the mapping function itself. This function is known as **Kernel function** and it reduces the complexity of finding the mapping function. So*,* **Kernel function defines inner product in the transformed space.**

We have to choose the correct kernel based on our application. There are various types of kernels :

$$1.\ Polynomial\ Kernel\ (Homogeneous): K(x_i, x_j) \ = \ \left(x_i^T x_j\right)^d$$

$$2.\ Polynomial\ Kernel\ (Inhomogeneous)\ :\ K(x_i, x_j) \ = \left(1 \ + \ x_i^T x_j\right)^d$$

$$3.\ Radial\ Basis\ Function\ Kernel\ :\ \ K(x_i, x_j) \ = exp\left(-\ \frac{\|x_i \ - \ x_j\|^2}{2\sigma^2}\right)$$

$$4.\ Domain\ Specific\ Kernel\ \ :Kernel\ chosen\ based\ on\ the\ domain\ of\ the\ application$$

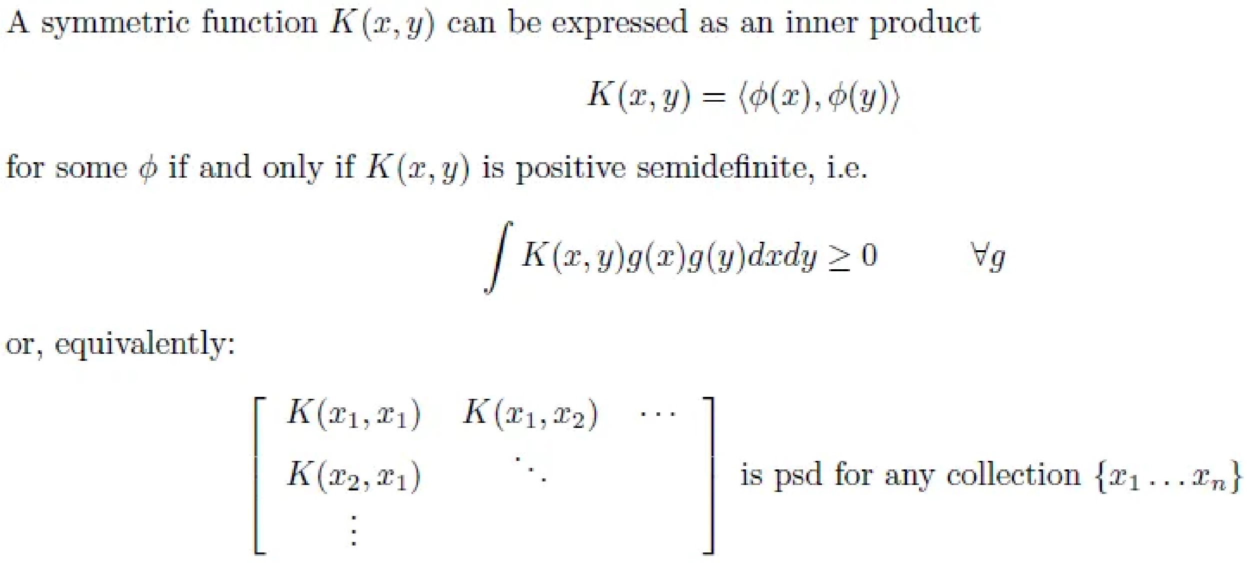
Mercer's Theorem

Apart from this predefined kernels*, what conditions determine which functions can be considered as Kernels ?* This is given by **Mercer's theorem.**

First condition is rather trivial i.e. the **Kernel function must be symmetric.** As a Kernel function is *dot product (inner product)* of the mapping function we can write as below:

$$K\left(\vec{x_i}, \vec{x_j}\right) = \phi\left(\vec{x_i}\right)^T \phi\left(\vec{x_j}\right) \Longrightarrow \phi\left(\vec{x_j}\right)^T \phi\left(\vec{x_i}\right) = K\left(\vec{x_j}, \vec{x_i}\right)$$

Mercer theorem guides us to the **necessary and sufficient condition** for a function to be Kernel function.



A symmetric function $K(x, y)$ can be expressed as an inner product

$$K(x, y) = \langle \phi(x), \phi(y) \rangle$$

for some ϕ if and only if $K(x, y)$ is positive semidefinite, i.e.

$$\int K(x, y) g(x) g(y) dx dy \geq 0 \qquad \forall g$$

or, equivalently:

$$\begin{bmatrix} K(x_1, x_1) & K(x_1, x_2) & \cdots \\ K(x_2, x_1) & \ddots & \\ \vdots & & \end{bmatrix} \text{ is psd for any collection } \{x_1 \ldots x_n\}$$

(Reference : https://people.eecs.berkeley.edu/~jordan/courses/)

7. Train and Runtime Complexities of SVM

The Training time complexities of SVMs is approximately O(n^2). If n is very large, then O(n^2) is also very large, so SVMs are not used in low-latency based applications.

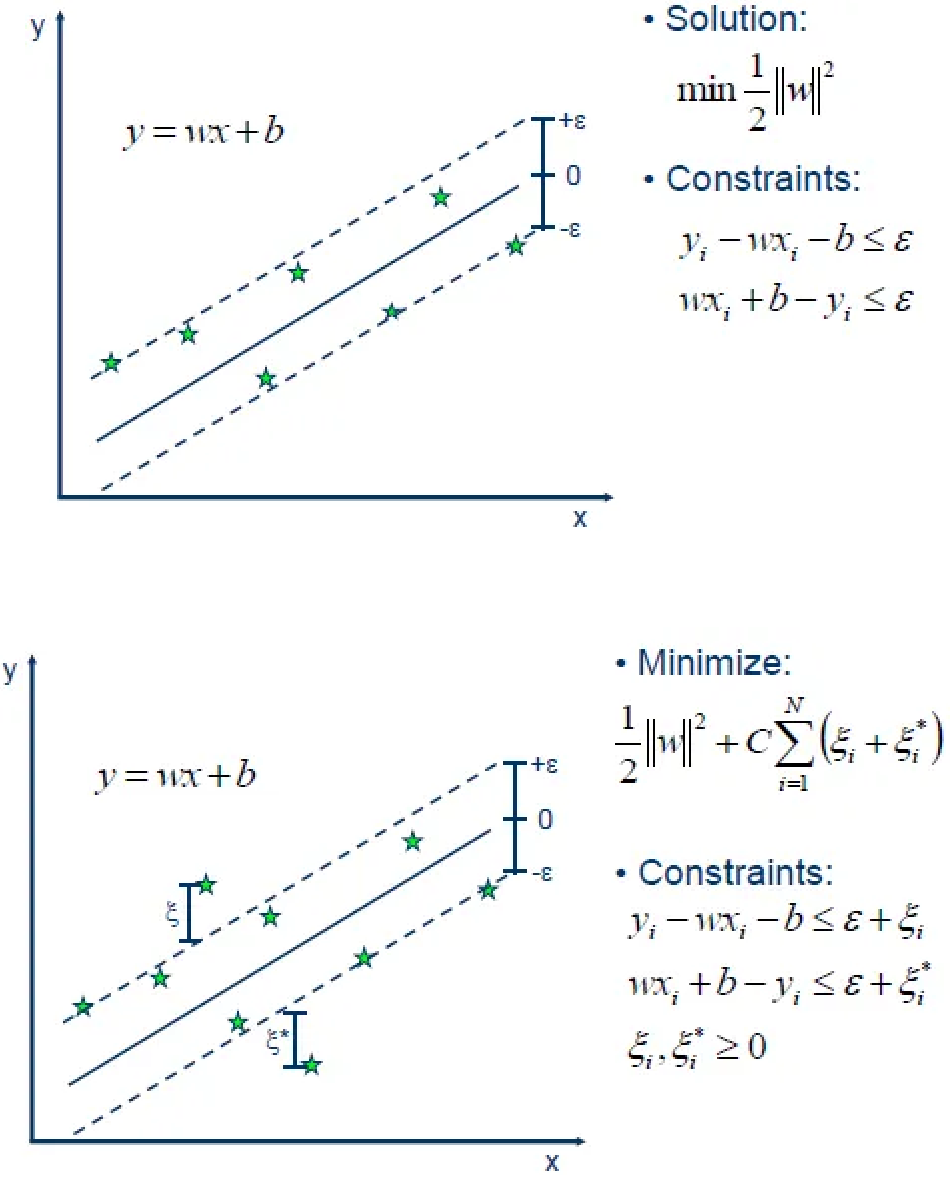
The Runtime Complexity is approximately O(k.d) where

k = Number of Support Vectors

d =Dimensionality of data

8. Support Vector Machines — Regression (SVR)

Support Vector Machine can also be used as a regression method, maintaining all the main features that characterize the algorithm (maximal margin). The Support Vector Regression (SVR) uses the same principles as the SVM for classification, with only a few minor differences. First of all, because output is a real number it becomes very difficult to predict the information at hand, which has infinite possibilities. In the case of regression, a margin of tolerance (epsilon) is set in approximation to the SVM which would have already requested from the problem. But besides this fact, there is also a more complicated reason, the algorithm is more complicated therefore to be taken in consideration. However, the main idea is always the same: to minimize error, individualizing the hyper-plane which maximizes the margin, keeping in mind that part of the error is tolerated.

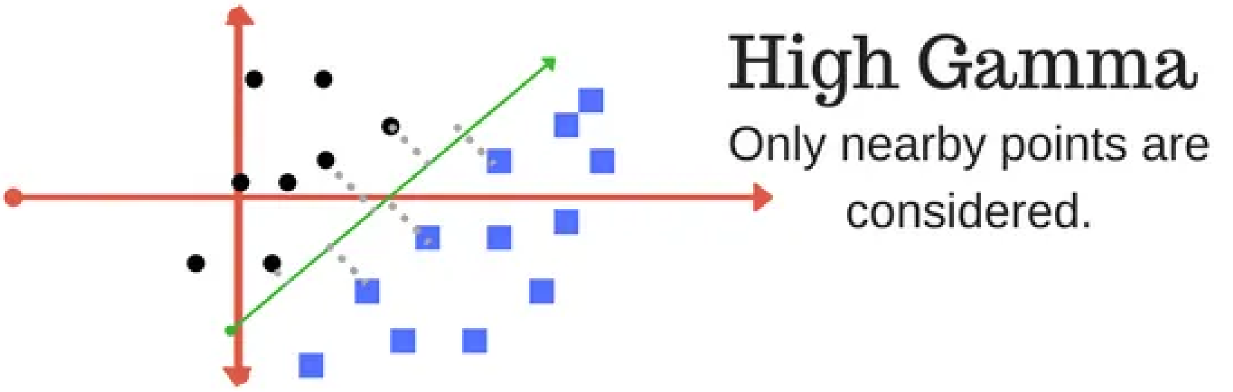

y
x
$y = wx + b$
+ε
0
-ε
• Solution:
$\min \frac{1}{2}\|w\|^2$
• Constraints:
$y_i - wx_i - b \le \varepsilon$
$wx_i + b - y_i \le \varepsilon$
y
x
$y = wx + b$
+ε
0
-ε
ξ
ξ*
• Minimize:
$\frac{1}{2}\|w\|^2 + C\sum_{i=1}^{N}(\xi_i + \xi_i^*)$
• Constraints:
$y_i - wx_i - b \le \varepsilon + \xi_i$
$wx_i + b - y_i \le \varepsilon + \xi_i^*$
$\xi_i, \xi_i^* \ge 0$

Linear SVR (Source **:** https://www.saedsayad.com/support_vector_machine_reg.htm**)**

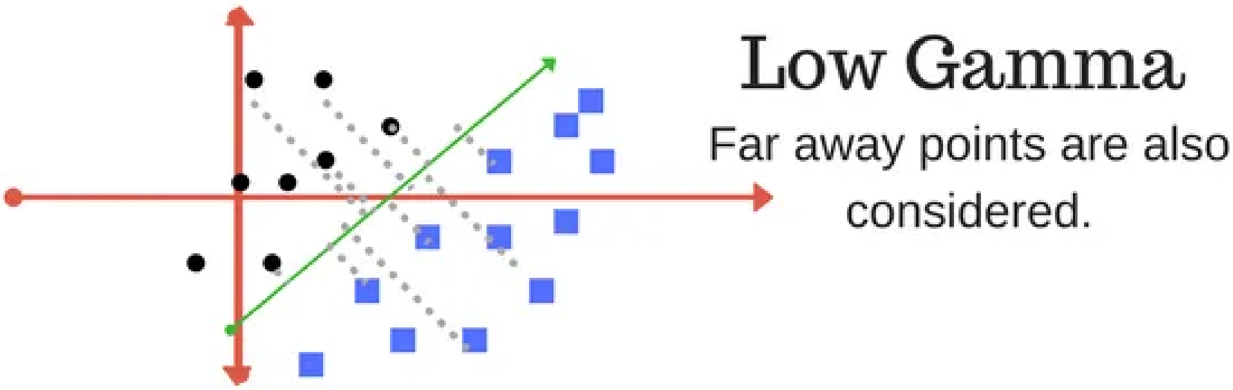
Hyper-parameter : Gamma

There is a very important hyper-parameter in SVC called '**gamma**' which is used very often.

Gamma : The gamma parameter defines how far the influence of a single training example reaches, with low values meaning 'far' and high values meaning 'close'. In other words, with low gamma, points far away from plausible separation line are considered in calculation for the seperation line. Where as high gamma means the points close to plausible line are considered in calculation.



High Gamma (Source :www.datascience.com**)**

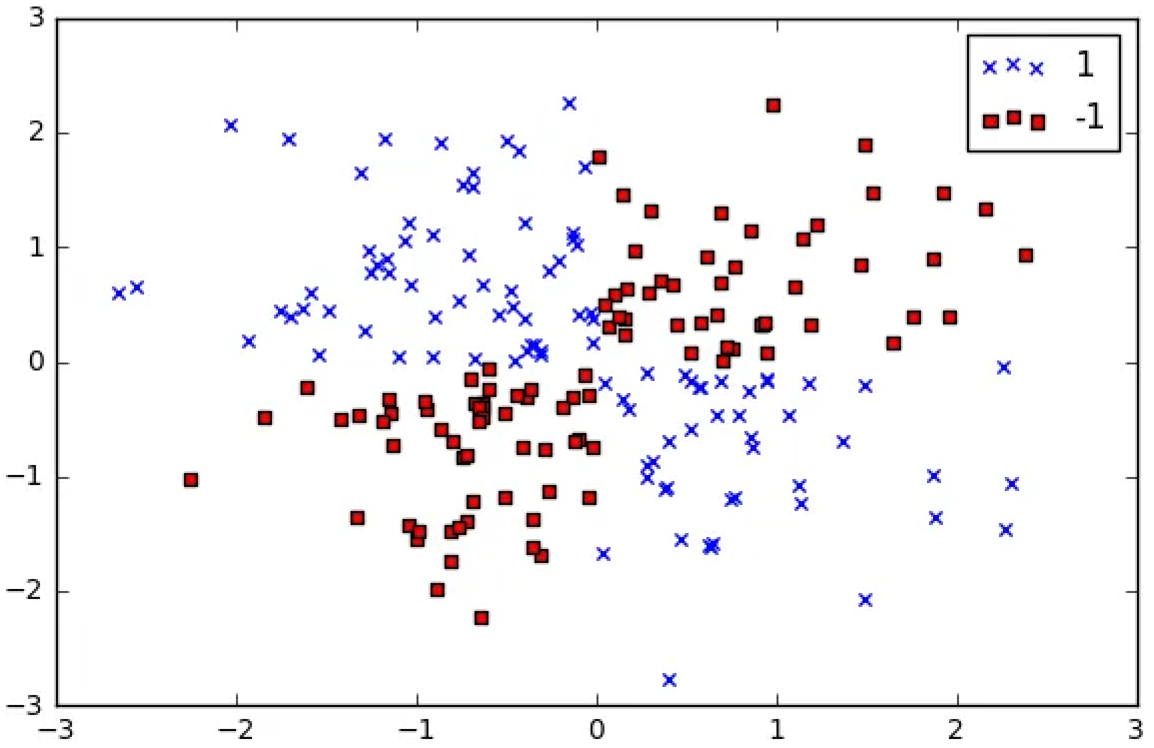


Low Gamma(Source : www.datascience.com**)**

Results of Hyper-parameter Tuning in RBF-kernel

(Reference 1: Excerpt taken from book : Sebastian book Python Machine Learning.

Reference 2: https://chrisalbon.com/machine_learning/)



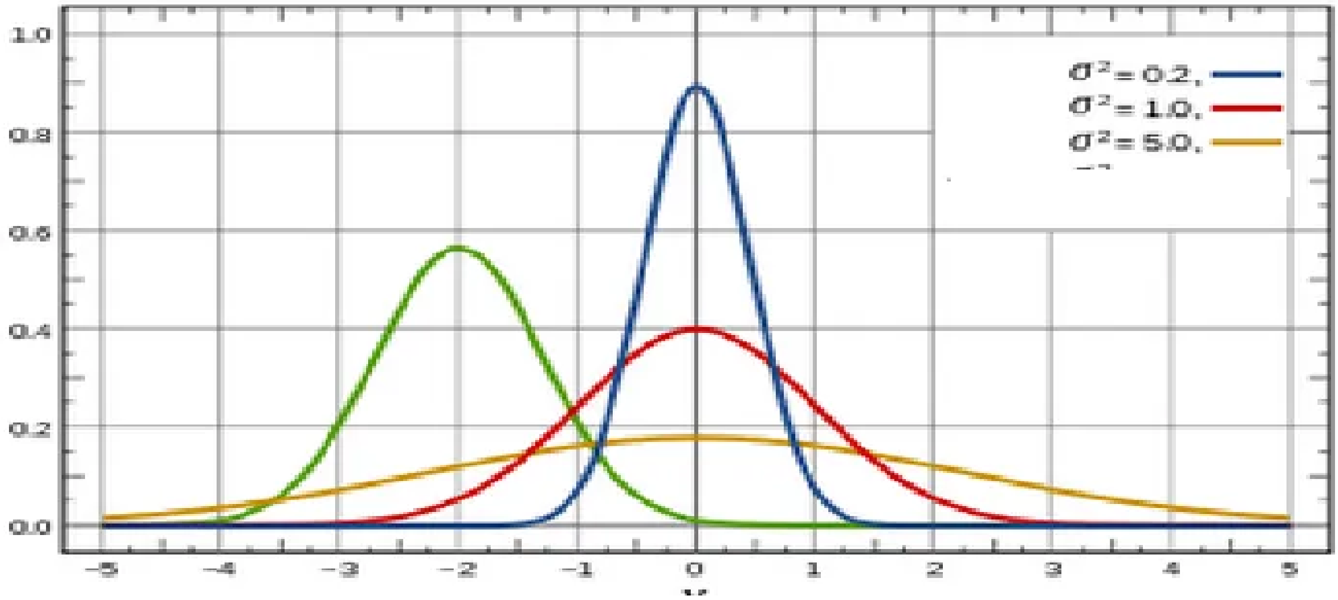
Dataset

Radial Basis Function is a commonly used kernel in SVC:

$$K(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{||\mathbf{x} - \mathbf{x}'||^2}{2\sigma^2}\right)$$

where ||x−x'||^2 is the squared Euclidean distance between two data points x and x'.

If value of sigma is kept constant, as distance between the points increases, the value of K(x,x') decreases exponentially and hence the similarity decreases.



Behavior of RBF kernel function with sigma

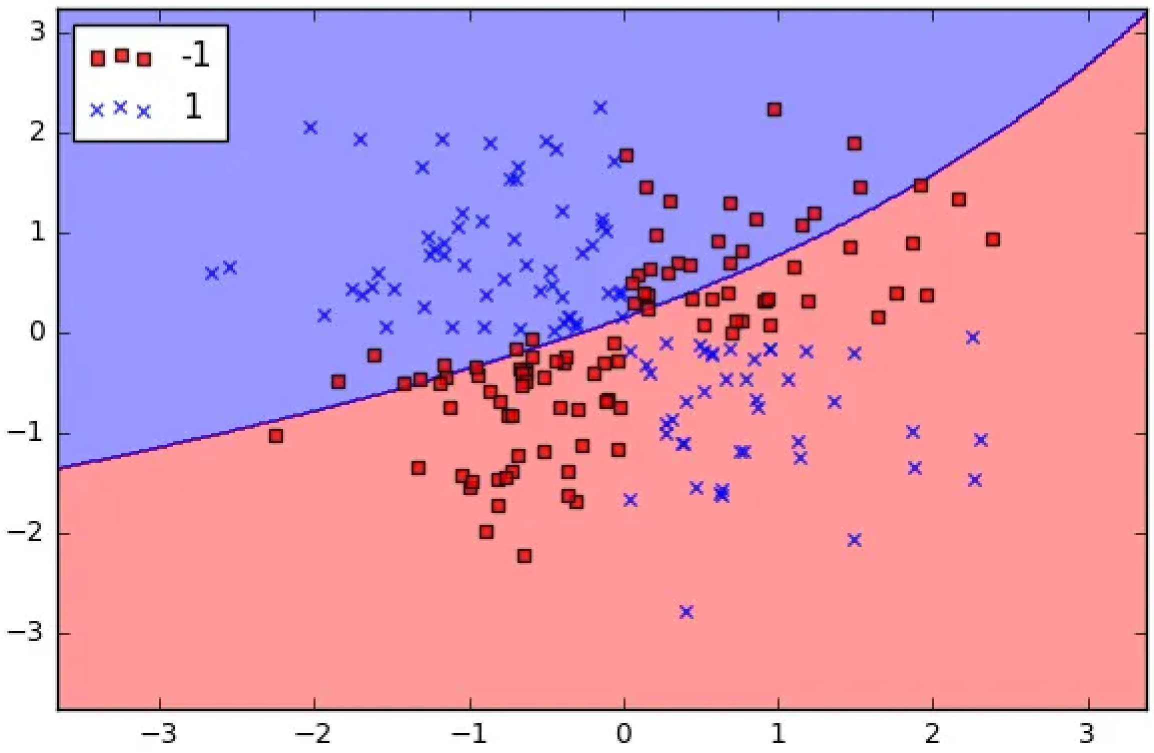
Now if we keep the distance constant and change sigma ,we can observe from the above figure that :

If we increase sigma, the peakedness of the graph decreases and vice-versa. Intuitively it means that, if we increase sigma, the distance within which the points will be considered to have similarity value greater than 0, will increase (Any other points which are present at distance greater than this will have similarity value equal to 0) and vice-versa.

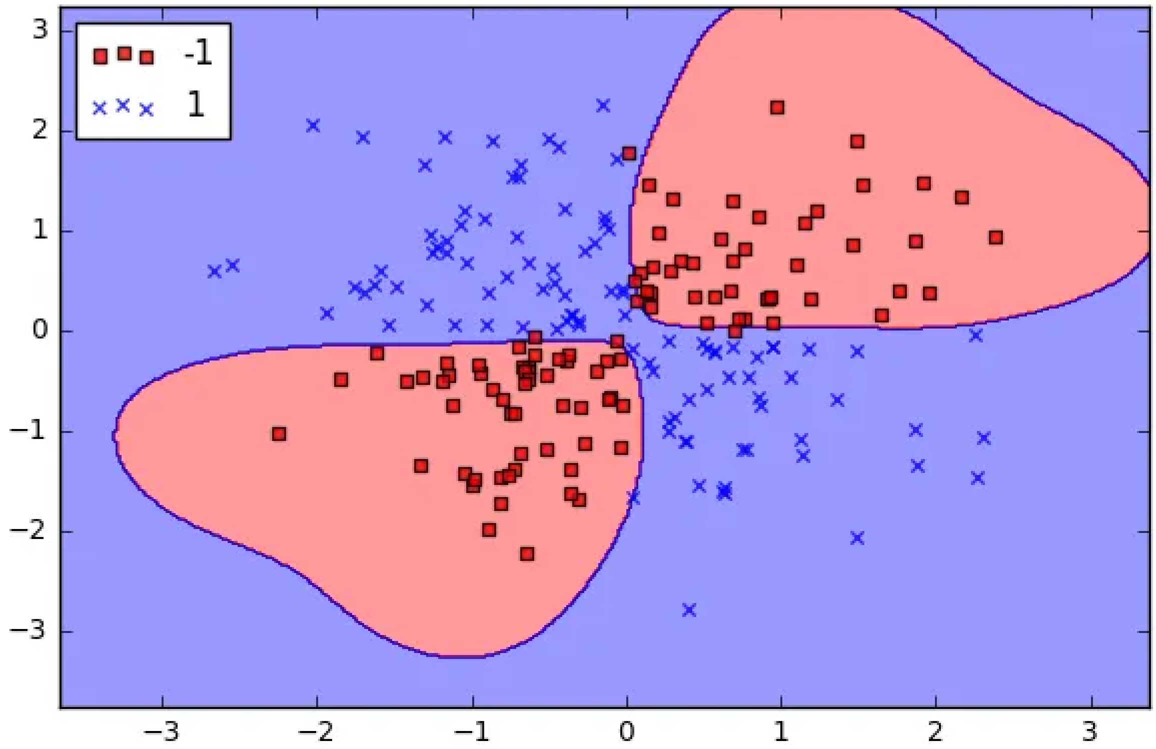
However, it is only important to know that an SVC classifier using an RBF kernel has two parameters: '**gamma**' and '**C**'.

We have explained these two parameters in the above sections.

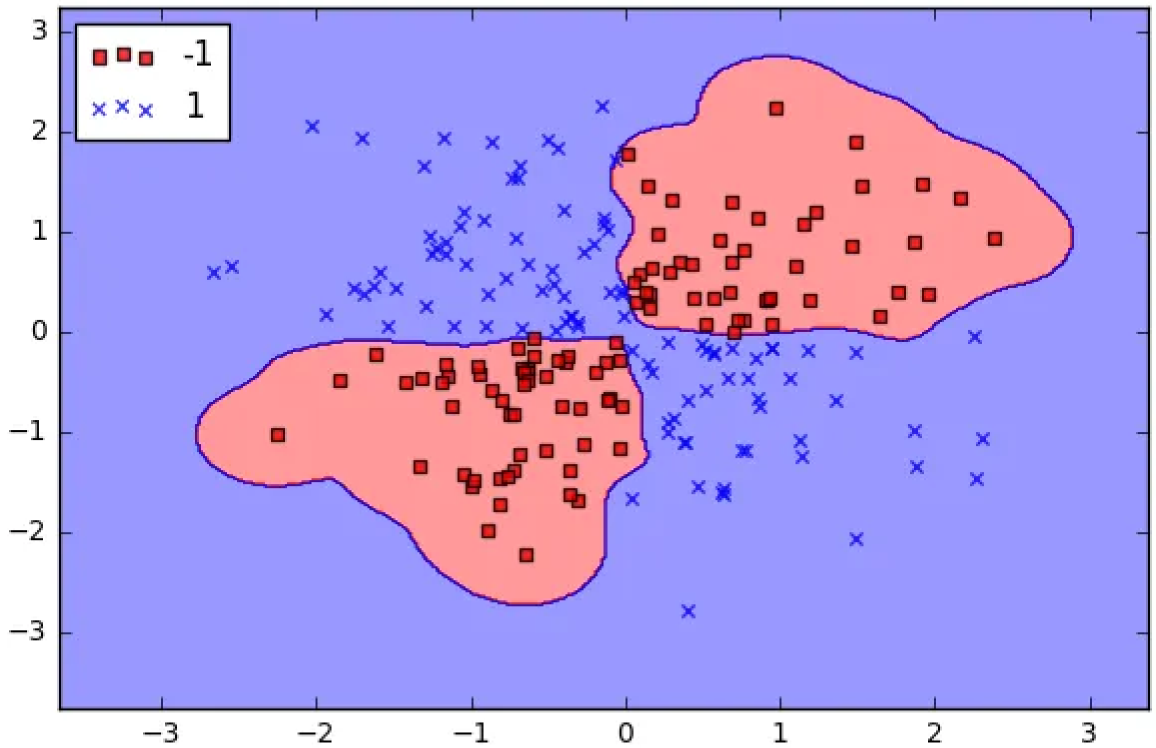
Gamma = 0.01 and C = 1



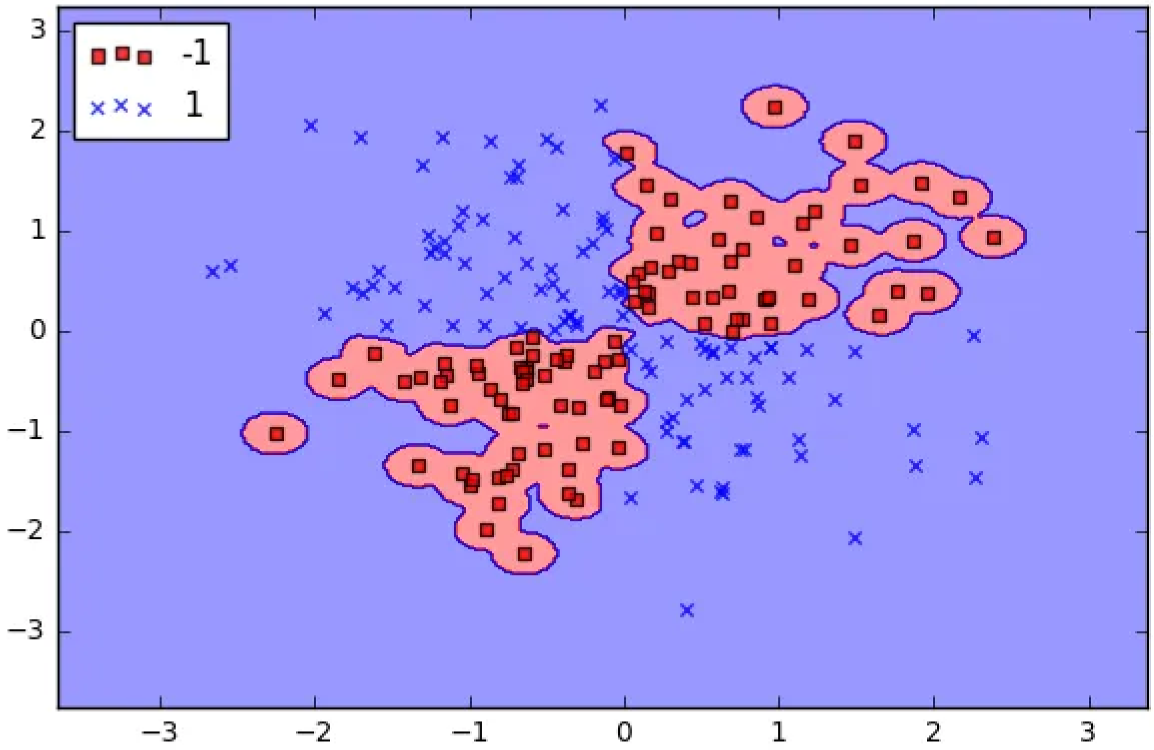
Gamma = 1.0 and C = 1.0



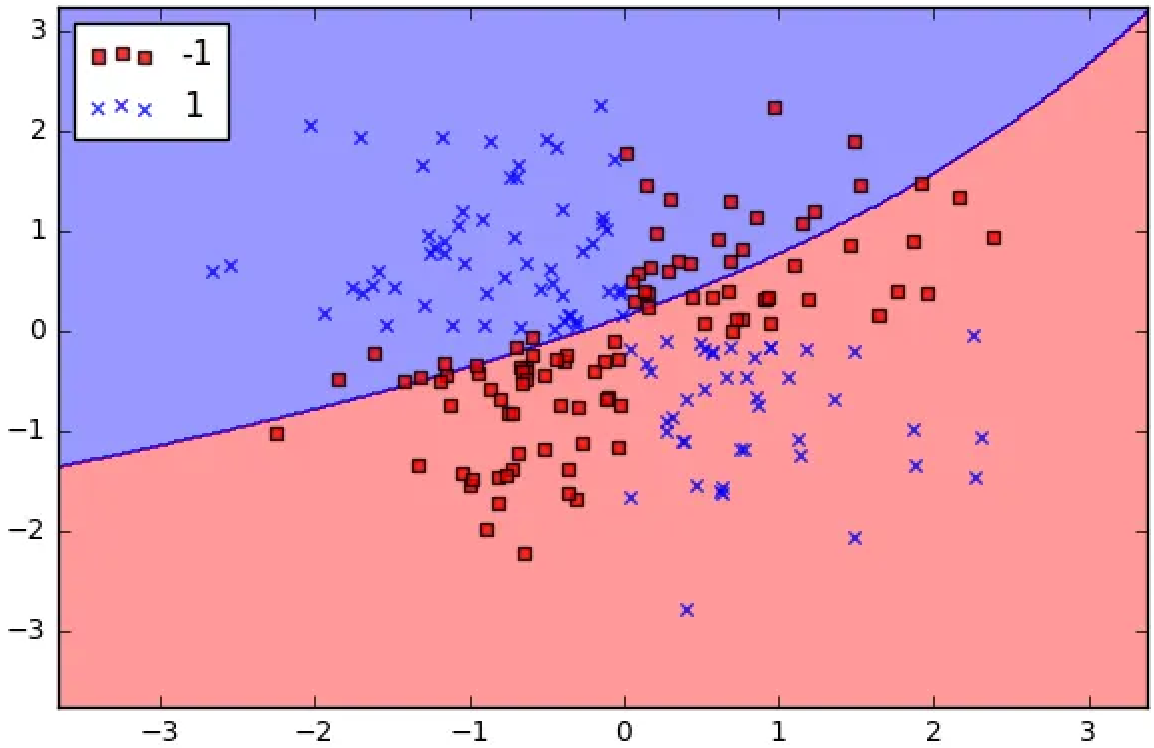
Gamma = 10.0 and C = 1.0



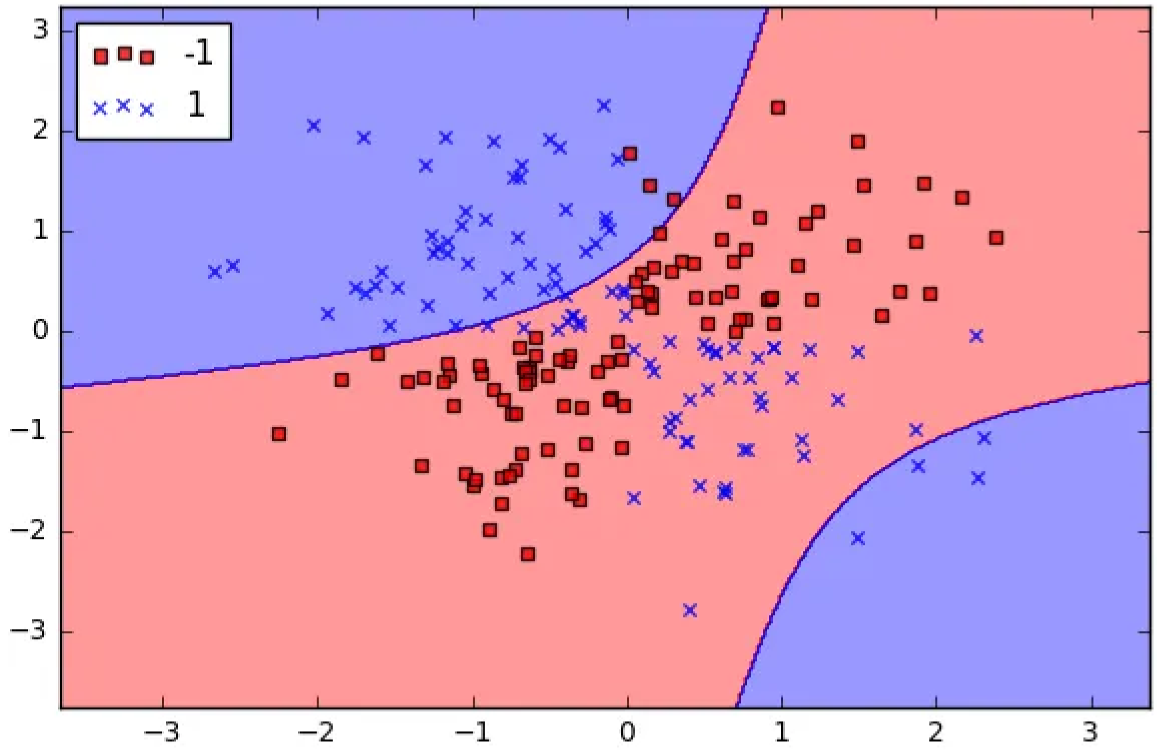
Gamma = 100.0 and C = 1.0



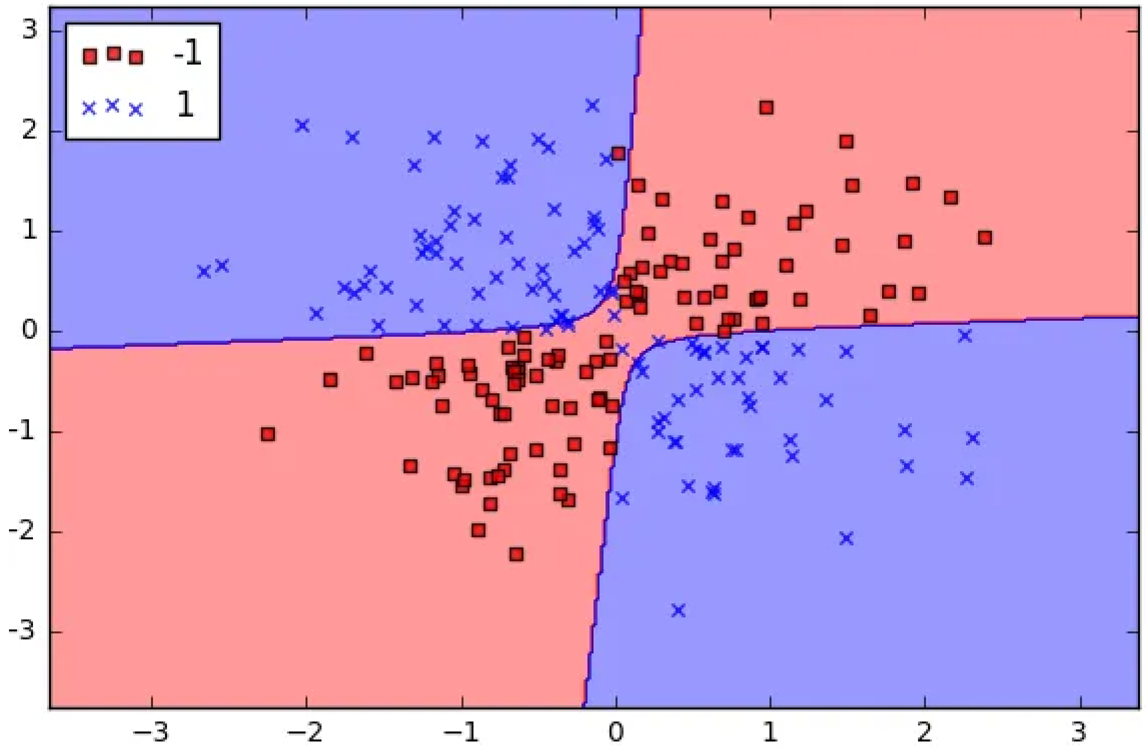
C = 1 and gamma = 0.01



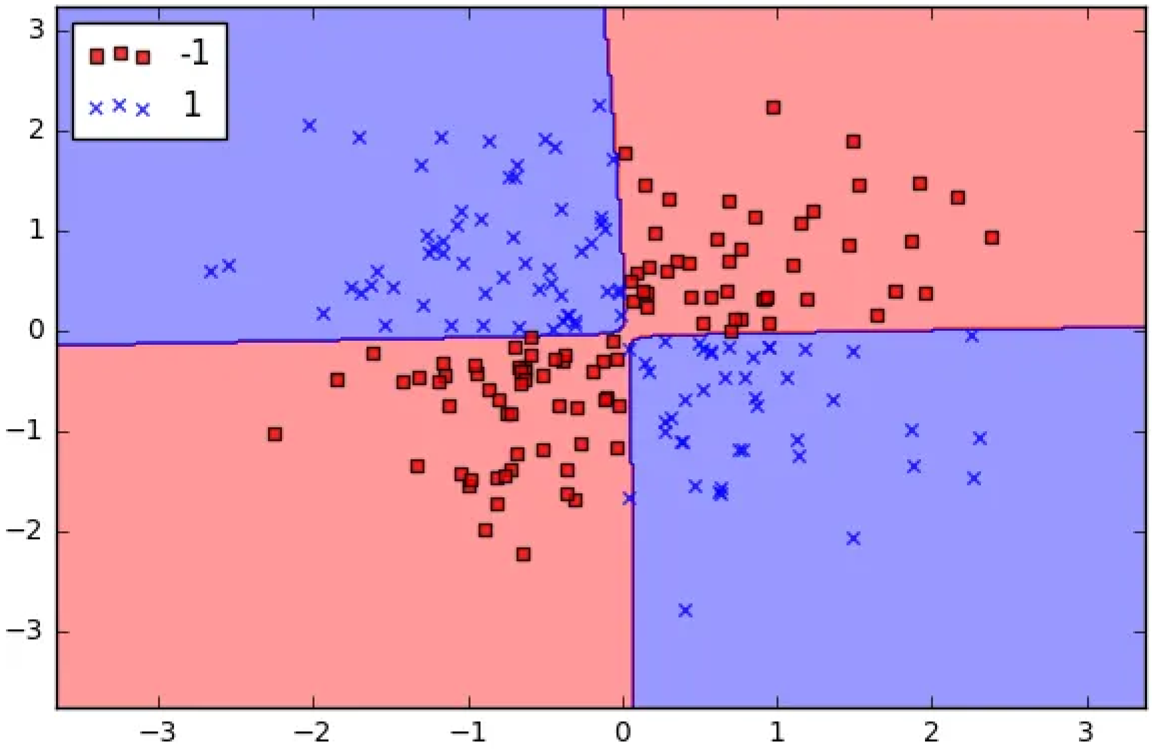
C = 10 and gamma = 0.01



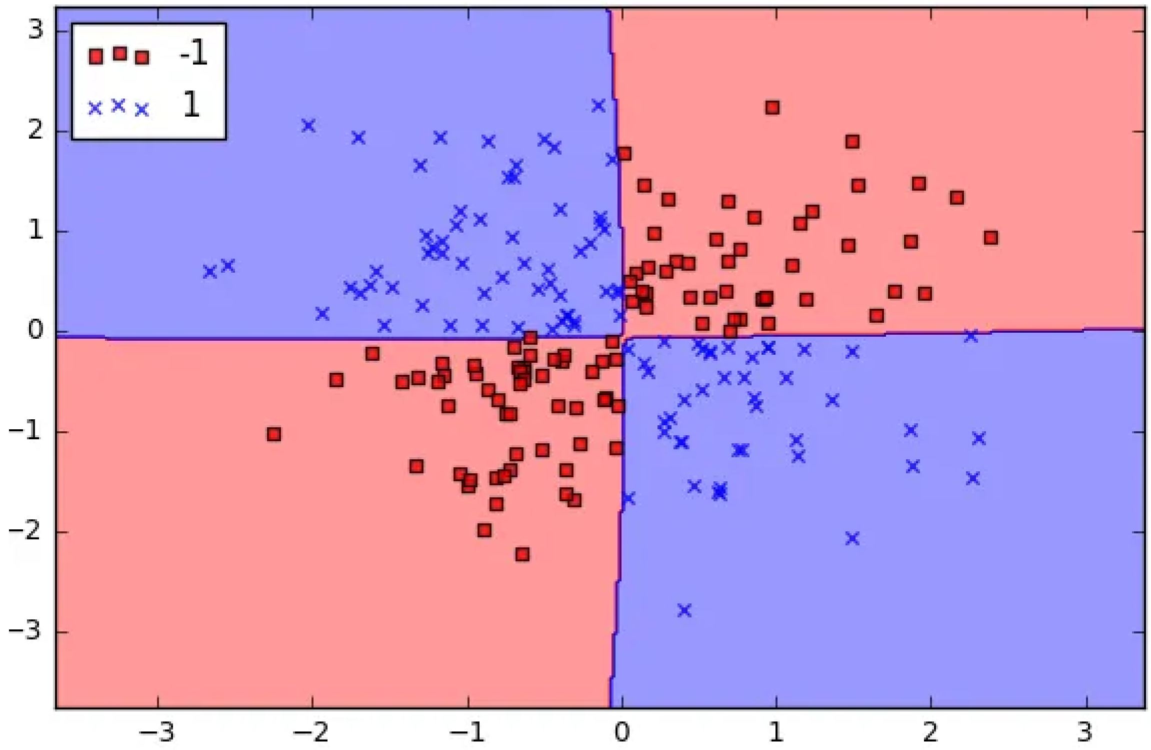
C = 1000 and gamma = 0.01



C = 10000 and gamma = 0.01



C = 100000 and gamma = 0.01



Observation : **'gamma'** is a parameter of the RBF kernel and can be thought of as the 'spread' of the kernel and therefore the decision region. When **'gamma'** is low, the 'curve' of the decision boundary is very low and thus the decision region is very broad. When **'gamma'** is high, the 'curve' of the decision boundary is high, which creates islands of decision-boundaries around data points.

'C' is a parameter of the SVC learner and is the penalty for misclassifying a data point. When **'C'** is small, the classifier is okay with misclassified data points (high bias, low variance). When **'C'** is large, the classifier is heavily penalized for misclassified data and therefore bends over backwards avoid any misclassified data points (low bias, high variance).

9. Application of SVMs

9.1 Implementation of SVMs on Real World Dataset

In this section , we will implement SVM on **Donors Choose Dataset**. In this section, we will use SVM functions provided by ***sklearn library.***

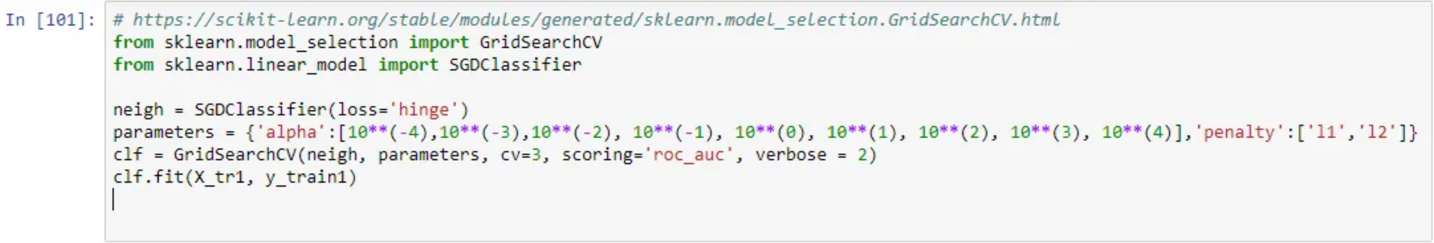
You can find the details of the dataset from here. It is a classification based problem.

For kernel SVMs we can use the SVC( ) provided by sklearn library.You can get a detailed documentation about it from here.

For linear SVMs we can SGDClassifier(loss ='hinge') provided by sklearn library. You can get a detailed documentation about it from here.

We have used the following steps in order to apply linear SVM on Donors Choose dataset :

1. Split the dataset into train and test dataset.
2. Next we have performed the hyper-parameter tuning in order to find the best hyper-parameters using Cross Validation technique. Here we have GridSearchCV( ) provided by sklearn library.



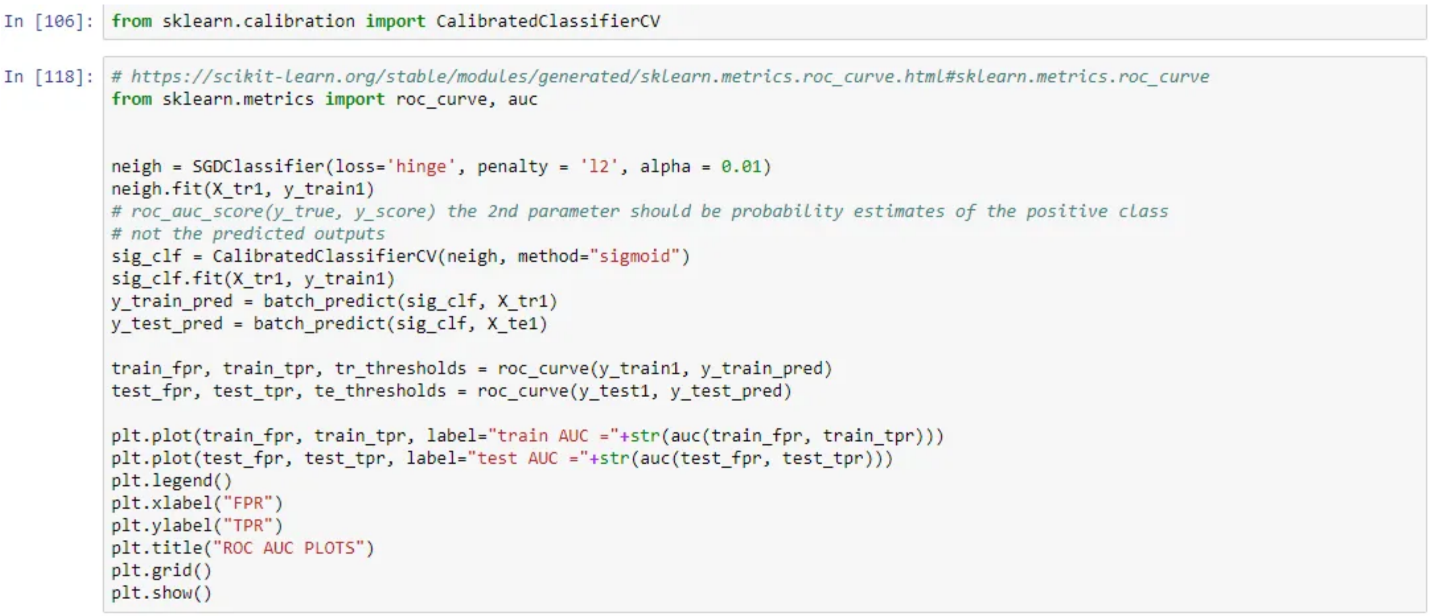
```
In [101]: # https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html
          from sklearn.model_selection import GridSearchCV
          from sklearn.linear_model import SGDClassifier

          neigh = SGDClassifier(loss='hinge')
          parameters = {'alpha':[10**(-4),10**(-3),10**(-2), 10**(-1), 10**(0), 10**(1), 10**(2), 10**(3), 10**(4)],'penalty':['l1','l2']}
          clf = GridSearchCV(neigh, parameters, cv=3, scoring='roc_auc', verbose = 2)
          clf.fit(X_tr1, y_train1)
```

Hyper-parameter Tuning using Grid Search Cross Validation

Although there are various hyper-parameters, we have used two main hyper-parameters : alpha, penalty (or regularization term)

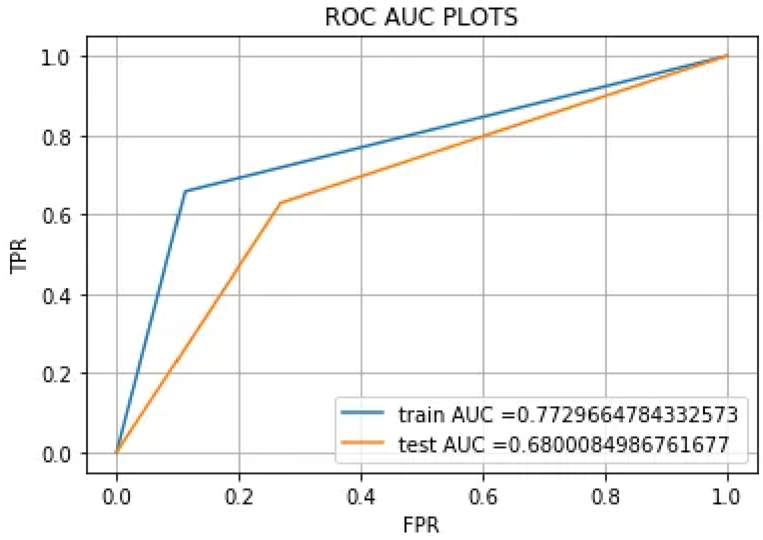
3. Next we create the model with the best hyper-parameters.

4. Now we evaluate the performance of the model using test dataset. The evaluation metric we have used here is the ‘roc auc’ score since it is an imbalanced dataset.



```
In [106]: from sklearn.calibration import CalibratedClassifierCV

In [118]: # https://scikit-learn.org/stable/modules/generated/sklearn.metrics.roc_curve.html#sklearn.metrics.roc_curve
          from sklearn.metrics import roc_curve, auc


          neigh = SGDClassifier(loss='hinge', penalty = 'l2', alpha = 0.01)
          neigh.fit(X_tr1, y_train1)
          # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of the positive class
          # not the predicted outputs
          sig_clf = CalibratedClassifierCV(neigh, method="sigmoid")
          sig_clf.fit(X_tr1, y_train1)
          y_train_pred = batch_predict(sig_clf, X_tr1)
          y_test_pred = batch_predict(sig_clf, X_te1)

          train_fpr, train_tpr, tr_thresholds = roc_curve(y_train1, y_train_pred)
          test_fpr, test_tpr, te_thresholds = roc_curve(y_test1, y_test_pred)

          plt.plot(train_fpr, train_tpr, label="train AUC ="+str(auc(train_fpr, train_tpr)))
          plt.plot(test_fpr, test_tpr, label="test AUC ="+str(auc(test_fpr, test_tpr)))
          plt.legend()
          plt.xlabel("FPR")
          plt.ylabel("TPR")
          plt.title("ROC AUC PLOTS")
          plt.grid()
          plt.show()
```

Code Snippet showing the creation of the model using the best hyper-parameters and evaluation of the model



ROC AUC plots with AUC Scores of Train and Test data

10. References

1. https://medium.com/machine-learning-101/chapter-2-svm-support-vector-machine-theory-f0812effc72

2. https://en.wikipedia.org/wiki/Support-vector_machine

3. https://www.geeksforgeeks.org/convex-hull-set-1-jarviss-algorithm-or-wrapping/