**Support Vector Machines (SVM)**

[[Sandun Dayananda](https://sandundayananda.medium.com/?source=post_page-----db8314e9092d--------------------------------)](https://sandundayananda.medium.com/?source=post_page-----db8314e9092d--------------------------------)

[Sandun Dayananda](https://sandundayananda.medium.com/?source=post_page-----db8314e9092d--------------------------------)

·

[Follow](https://medium.com/m/signin?actionUrl=https%3A%2F%2Fmedium.com%2F_%2Fsubscribe%2Fuser%2Fd03f0cde0742&operation=register&redirect=https%3A%2F%2Fsandundayananda.medium.com%2Fsupport-vector-machines-svm-db8314e9092d&user=Sandun+Dayananda&userId=d03f0cde0742&source=post_page-d03f0cde0742----db8314e9092d---------------------post_header-----------)

13 min read

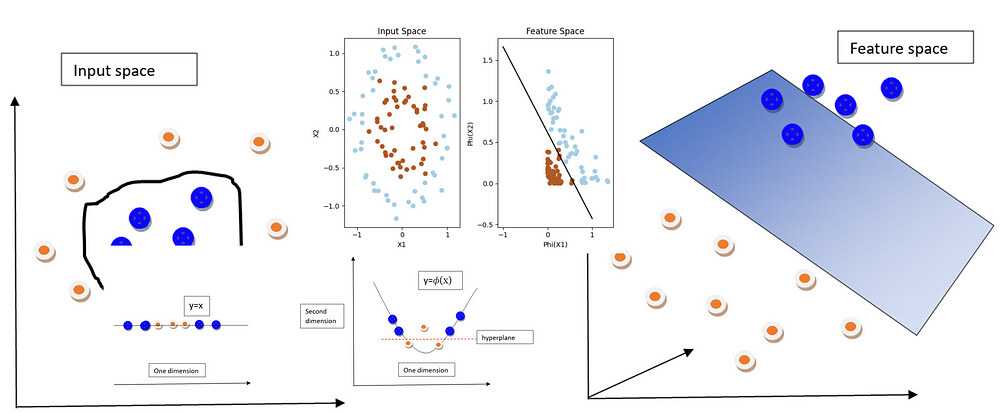
·

Nov 6

15

1

***Mathematical and machine learning approach***

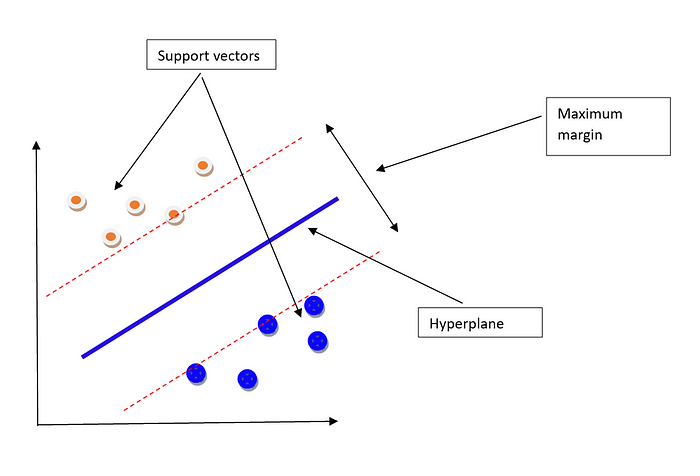


support vector machines

Before dive in to support vector machines, I assume that you have the basic understanding of both [linear regression](https://medium.com/@sandundayananda/linear-regression-4c327a5d5ba7) and [logistic regression](https://medium.com/@sandundayananda/logistic-regression-55512384851b). If you haven’t, I recommend you to go through them first. Support vector machines is a very important machine learning algorithm due to their ability to **handle complex datasets**, **high accuracy** while requiring **less computational power** and also can be used for both regression(specially for non linear regression) and classification tasks. But, they are mainly used for classification tasks

**SVM** is a supervised machine learning algorithm **mainly used for classification tasks**. It aims to find an optimal hyperplane that separates the data points of different classes with the maximum margin. The objective of SVM is to maximize the distance between the hyperplane and the nearest data points of each class, which are called support vectors. SVM can handle both linearly separable and non-linearly separable data by using different **kernel functions** (e.g: linear, polynomial, radial basis function,…) to transform the data into a higher-dimensional space where separation is easier.

**How SVM works?**



maximum margin

SVM operates by mapping the data from its original space to a higher-dimensional space. The goal is to find a hyperplane, a decision boundary, that effectively separates the data points of different classes.

If the data is linearly separable, SVM aims to find the best hyperplane that maximizes the margin. The margin is the region between two parallel hyperplanes, and it represents the distance between the classes. SVM seeks to find the hyperplane with the largest margin, as it provides a wider separation between the classes and can lead to better generalization and classification performance.

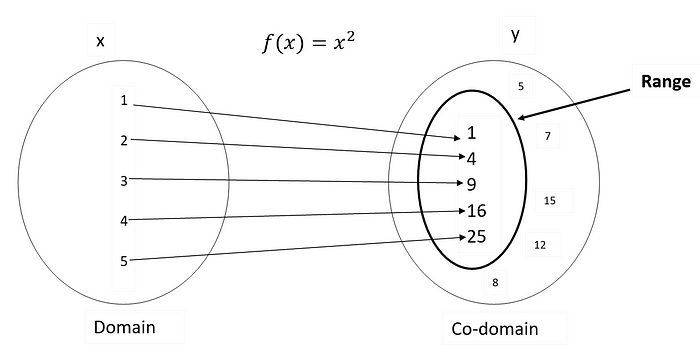
In addition to maximizing the margin, SVM also identifies the support vectors, which are the data points that lie closest to the decision boundary. These support vectors play a crucial role in defining the hyperplane and determining the margin. By considering the support vectors, SVM ensures that the decision boundary is not overly influenced by the entire dataset but rather focuses on the critical points near the boundary.

The hyperplane with the maximum margin is considered the optimal solution, as it provides the best balance between separating the classes and avoiding overfitting. SVM’s ability to find the maximum margin hyperplane and classify new data points based on their position relative to the hyperplane makes it a popular choice for many classification problems.

Overall, SVM leverages the concept of maximizing the margin between classes to create an effective decision boundary, allowing for accurate classification even in cases where the data points are not linearly separable in the original feature space. You will have more understanding when we go through the following explanations.

Okay, now let’s go through the mathematical background of SVM.

When defining a function in real space, we often consider the concepts of domain, range, and co-domain. The domain refers to the set of all possible input values for the function, while the range represents the set of all possible output values. The co-domain, on the other hand, encompasses a broader set that includes the range but may also include values that are not mapped by the function.



data point in space

Now, when we extend this concept to machine learning, specifically in the context of input space and feature space, we encounter a similar notion. In machine learning, the input space refers to the set of all possible input values or data points that can be fed into the model. The feature space, however, is a transformed or derived space where the input data is represented using a set of features or variables.

The domain in the input space refers to the set of all possible input values that the model can receive during training or inference. It defines the range of values over which the model can operate. Similarly, the range in the input space represents the set of possible output values that the model can produce based on the given inputs.

When we move to the feature space, we encounter a new domain and range. The domain in the feature space refers to the set of all possible combinations or transformations of the input features. These features can include different representations, engineered variables, or higher-order interactions. The range in the feature space represents the set of possible feature values or combinations that can be generated by the model.

The co-domain in the feature space is similar to the concept in real space. It includes the range of possible feature values but can extend beyond that, encompassing additional combinations or transformations that may not be directly mapped by the model.

Understanding the domains and ranges in these spaces helps us define the scope of inputs and outputs for our models and provides insights into the transformations and combinations that can be explored to enhance their performance.

**How above concepts used in SVM?**

In SVM, we are given a dataset consisting of data points that need to be classified or separated. Each data point is represented by a feature vector denoted as x, belonging to the input space.

Mathematically, we can represent the input space as: x ∈ ℝ^D

Here, ℝ^D represents a vector space with D dimensions, where x is a D-dimensional feature vector.

To enable SVM to handle complex relationships and nonlinear separations, we apply a mapping function to each input feature vector x. This mapping function transforms the input space into a higher-dimensional space called the feature space. The transformed feature vector is denoted as Φ(x), which belongs to the feature space.

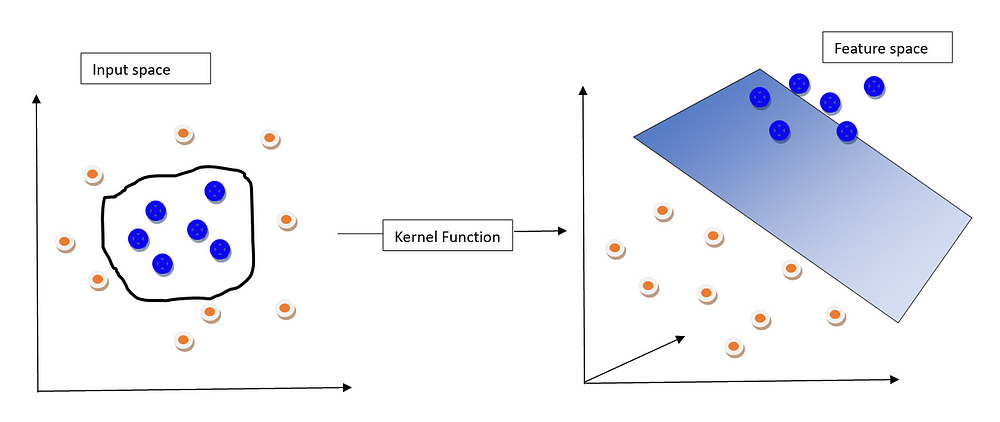
Mathematically, we can represent the feature space as: Φ(x) ∈ ℝ^M

In the feature space, each input feature vector x is mapped to a transformed basis vector Φ(x). The transformation is defined by the mapping function Φ(), which maps the input features to a new set of features in a higher-dimensional space.

It’s important to note that the dimensionality of the feature space, denoted by M, can be different from the dimensionality of the input space (D). The transformation to the feature space is performed in order to find a decision boundary that separates the classes more effectively.

By mapping the data points to a higher-dimensional feature space, SVM seeks to find an optimal hyperplane that maximizes the margin between the classes, facilitating better classification or separation.

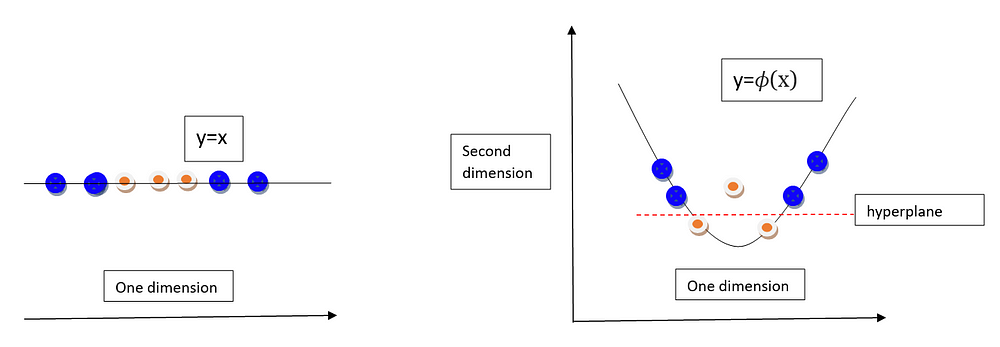
The input space is the original space where the data points reside, represented by the feature vectors x. The feature space is a transformed space obtained by mapping the input features to a higher-dimensional space using the mapping function Φ(), represented by the transformed feature vectors Φ(x). The feature space enables SVM to perform more complex classifications by finding optimal hyperplanes in this transformed space. We can visually represent above scenario as following.



using kernel function

***In the above image, as you can observe, kernel function converts the non linear boundary in the input space in to a linear boundary in the feature space.***

In order to understand this practically we can use one dimension dataset and apply kernel function to it.



applying kernel function to one dimension data

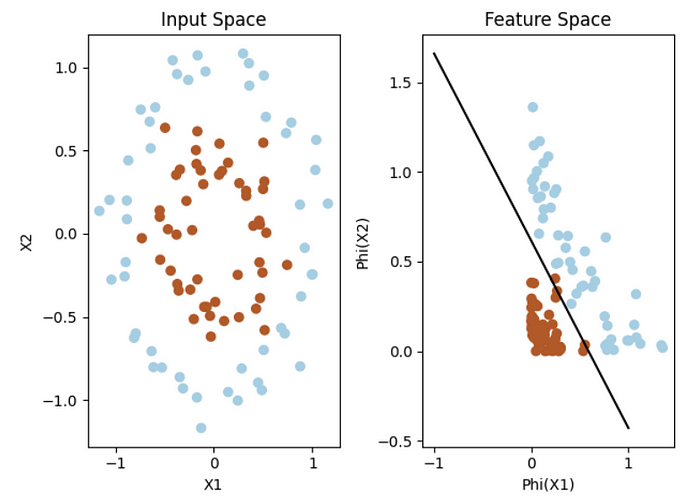
In the above image we can consider we have given one dimension data and it can be defined as**y=x and we can assume we have used a function f(x)=x², here x² is the kernel function**. After applying this kernel function we have divided the data points using a line. When the data is two dimensional the hyperplane is a line. When the data has more dimensions the hyperplane is a plane.

If we genuinely apply kernel function for a lot of features, it is needed a huge computation power. Therefore, in order to overcome this, we use a method called **kernel trick**.

In order to try above example practically we can use following code.

import numpy as np  
import matplotlib.pyplot as plt  
from sklearn import svm, datasets  
  
# Generate some sample data  
X, y = datasets.make\_circles(n\_samples=100, noise=0.1, factor=0.5, random\_state=42)  
  
# Transform input space to feature space  
def transform\_input\_space(X):  
 phi\_X = np.column\_stack((X[:, 0]\*\*2, X[:, 1]\*\*2))  
 return phi\_X # here I used phi\_x is Φ(x)  
  
# Create SVM classifier  
svm\_classifier = svm.SVC(kernel='linear')  
  
# Fit SVM classifier on transformed feature space  
X\_transformed = transform\_input\_space(X)  
svm\_classifier.fit(X\_transformed, y)  
  
# Visualize input space  
plt.subplot(1, 2, 1)  
plt.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.Paired)  
plt.xlabel('X1')  
plt.ylabel('X2')  
plt.title('Input Space')  
  
# Visualize feature space  
plt.subplot(1, 2, 2)  
plt.scatter(X\_transformed[:, 0], X\_transformed[:, 1], c=y, cmap=plt.cm.Paired)  
plt.xlabel('Phi(X1)')  
plt.ylabel('Phi(X2)')  
plt.title('Feature Space')  
  
# Plot separating hyperplane in feature space  
w = svm\_classifier.coef\_[0]  
a = -w[0] / w[1]  
xx = np.linspace(-1, 1)  
yy = a \* xx - (svm\_classifier.intercept\_[0]) / w[1]  
plt.plot(xx, yy, 'k-')  
  
plt.tight\_layout()  
plt.show()

*You can find the full code in my*[*GitHub*](https://github.com/SandunDayananda/support-vector-machines/tree/main)



use of kernel function

In this code, we first generate some sample data in the form of concentric circles using the make\_circles function from scikit-learn.

Next, we define the transform\_input\_space function, which transforms the input space into a feature space by squaring each feature in X. This transformation is applied to the input data using transform\_input\_space(X).

Then, we create an SVM classifier with a linear kernel (SVC(kernel='linear')) and fit it on the transformed feature space (X\_transformed) along with the corresponding labels y.

We visualize the input space on the left subplot and the feature space on the right subplot using scatter plots, with different colours representing the two classes.

Finally, we plot the separating hyperplane in the feature space by calculating the coefficients (w) and intercept of the hyperplane from the SVM classifier and plotting a line (yy = a \* xx - (svm\_classifier.intercept\_[0]) / w[1]) using xx as the x-axis values.

The resulting plot illustrates how the SVM transforms the non-linear boundary in the input space into a linear one in the feature space, allowing for effective separation between the classes.

Feel free to modify the parameters or use your own dataset to experiment with different scenarios.

**Kernel functions**

**choosing a kernel function for classification task**

In the above code, you can see we have used a linear kernel classifier.

# Create SVM classifier  
svm\_classifier = svm.SVC(kernel='linear')

When choosing between a linear or non-linear kernel for SVM, several factors should be considered. Firstly, assess the linearity of the data to determine if a linear kernel is suitable for separating the classes accurately. If the classes can be separated by a straight line or hyperplane, a linear kernel is a good choice. In cases where the data is not linearly separable, a non-linear kernel should be considered to capture complex decision boundaries in a higher-dimensional feature space.

Additionally, examine whether transforming the input features can make the data linearly separable. Feature engineering techniques or domain knowledge may help create new features that enable linear separation. Prior domain knowledge can guide the selection of a specific non-linear kernel if it is deemed more appropriate.

***Consider the computational efficiency, especially for large datasets with many features. Linear kernels are generally faster than non-linear kernels, so if efficiency is a concern, a linear kernel might be preferable.***

To make an informed decision, it is recommended to experiment with both linear and non-linear kernels. Evaluate their performance using appropriate metrics such as **accuracy, precision, recall, or F1-score(***You can learn them*[*here*](https://medium.com/@sandundayananda/logistic-regression-55512384851b)**)**. By comparing the results, you can determine which kernel performs better for your specific dataset.

Remember that the choice of kernel depends on the dataset’s characteristics and the complexity of the required decision boundary. It is advisable to start with a linear kernel and then explore non-linear kernels if needed.

**Kernel trick**

The kernel trick is a powerful concept in support vector machines (SVM) that allows us to implicitly map data points into higher-dimensional feature spaces without explicitly calculating the transformed features. It enables SVM to effectively handle non-linearly separable data by finding complex decision boundaries in the transformed feature space.

In SVM, the decision boundary is determined by a subset of the training data called support vectors. These support vectors are the critical points that lie closest to the decision boundary. The key idea behind the kernel trick is to find a way to compute the dot product between feature vectors in the high-dimensional feature space without actually computing the explicit transformation.

Mathematically, the kernel trick is represented by a kernel function, denoted as K(x, x’), which takes two input feature vectors, x and x’, and calculates their similarity or inner product in the transformed feature space. ***The kernel function effectively replaces the dot product operation in the feature space without explicitly calculating the transformed feature vectors.***Because if we use the dot product it costs a lot of compute power.

By using the kernel function, the SVM algorithm can work directly in the input space while implicitly operating in a higher-dimensional feature space. This avoids the need to compute and store the transformed feature vectors explicitly, which can be computationally expensive, especially in high-dimensional spaces.

Commonly used kernel functions include:

1. Linear Kernel:

* K(x, x’) = x ⋅ x’
* Represents a linear relationship and is suitable for linearly separable data.

2. Polynomial Kernel:

* K(x, x’) = (γx ⋅ x’ + r)^d
* Captures non-linear patterns by raising the dot product of feature vectors to a specified degree (d), with an optional constant term (r) and scaling factor (γ).

3. Radial Basis Function (RBF) Kernel:

* K(x, x’) = exp(-γ ||x — x’||²)
* Measures similarity based on the Euclidean distance between feature vectors. It assigns higher weights to nearby points, controlled by the scaling factor (γ).

4. Sigmoid Kernel:

* K(x, x’) = tanh(γx ⋅ x’ + r)
* Applies a sigmoid activation function to the dot product of feature vectors. It can approximate non-linear decision boundaries.

The choice of kernel depends on the data characteristics and the desired complexity of the decision boundary. The kernel trick allows SVM to efficiently handle non-linearly separable data by implicitly mapping it into a higher-dimensional feature space, enabling accurate classification without explicitly computing the transformed features.

**Hyperplane(decision boundary)**

in the context of SVM, the terms “decision boundary” and “hyperplane” are often used interchangeably and refer to the same concept. In two-dimensional space, a hyperplane is a line, while in three-dimensional space, it becomes a plane. In higher-dimensional feature spaces, hyperplanes become more complex, but the underlying concept remains the same. The hyperplane separates the feature space into different regions corresponding to different classes.

The SVM algorithm aims to find the optimal hyperplane that achieves the maximum margin between the classes. This hyperplane is referred to as the “separating hyperplane” because it effectively separates the data points of different classes. It is positioned such that it maintains the largest possible distance from the nearest data points of each class, which enhances the generalization ability of the classifier.

The key idea is that the decision boundary (or separating hyperplane) created by SVM is chosen to maximize the margin, providing a clear separation between the classes and improving the robustness of the classifier to new, unseen data points.

It’s important to note that in some cases, when the data is not linearly separable, SVM uses techniques like kernel functions to map the data into a higher-dimensional feature space, where a linear decision boundary (hyperplane) can be found. This allows SVM to handle non-linearly separable data by transforming it into a space where linear separation is possible.

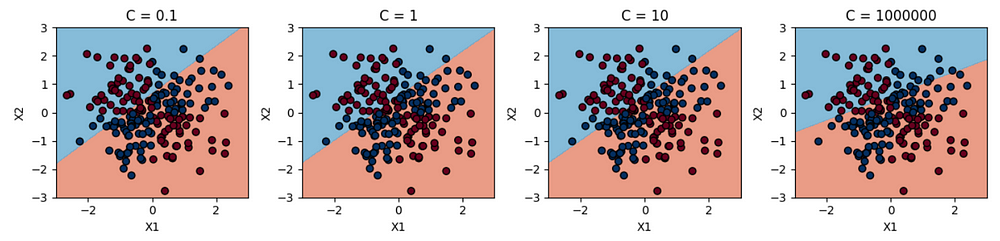
**Hyperparameters and parameters**

In SVM (Support Vector Machines), the following hyperparameters, parameters can be used to optimize the model:

***You can find the full code used to generate following plots on my***[***GitHub***](https://github.com/SandunDayananda/support-vector-machines/tree/main)

**Regularization parameter :**

The regularization parameter (C) in SVM is a **hyperparameter**. It controls the trade-off between allowing the model to classify training points accurately and creating a smooth decision boundary. A smaller value of C leads to a wider margin and potentially more misclassified points (higher bias, lower variance), while a larger value of C results in a narrower margin and fewer misclassified points (lower bias, higher variance). It helps to prevent overfitting by penalizing misclassification.

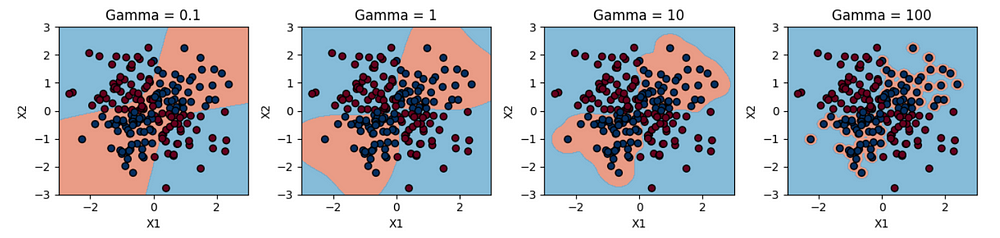


regularization parameter

If we increase the value of C, it will causes to overfitting and if we use a very small C value it will cause to underfitting. You can experiment it yourself using the given [code](https://github.com/SandunDayananda/support-vector-machines/tree/main).

**Gamma parameter:**

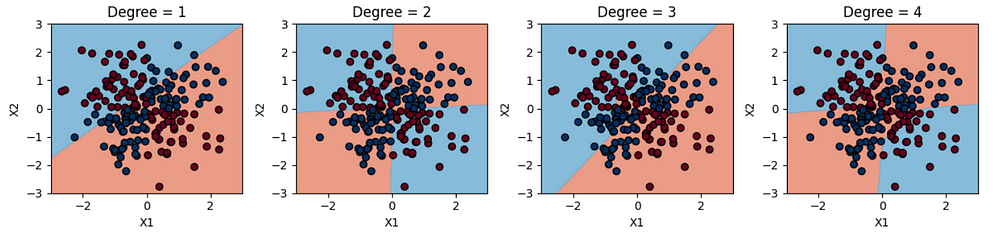
The gamma parameter is another **hyperparameter** in SVM used with RBF kernel. It defines the influence of a single training example. A higher gamma value leads to more complex decision boundaries that can better fit the training data (higher variance). Conversely, a lower gamma value makes the decision boundary smoother and may result in underfitting (higher bias). The choice of gamma depends on the specific dataset and the desired trade-off between complexity and generalization.



gamma

**Degree:**

The degree is a **parameter** specific to the polynomial kernel in SVM. It controls the flexibility of the decision boundary by specifying the degree of the polynomial function used. Higher degrees yield more flexible decision boundaries, but they can also lead to overfitting if not properly tuned. The degree parameter is typically used in combination with the kernel parameter when the kernel is set to “poly”.



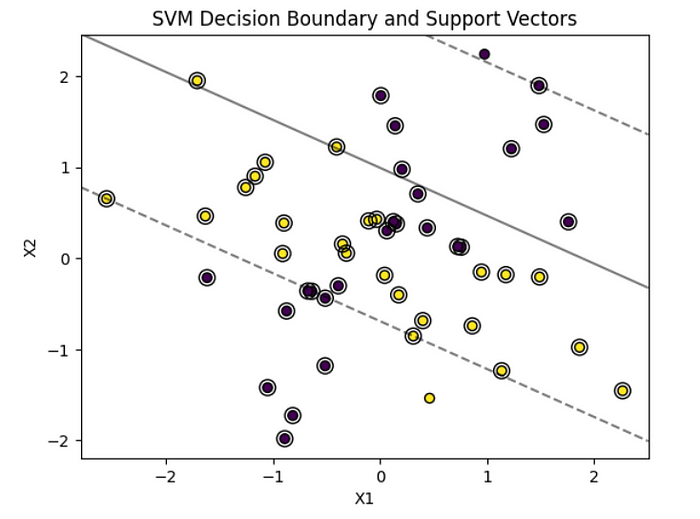
degree

**Random state:**

The random state is not a hyperparameter but a **parameter** used for reproducibility. It is used to set the seed for the random number generator when splitting the data into training and testing sets or when initializing certain random components of the SVM model. Setting a random state ensures that the splits or random initialization can be reproduced, providing consistent results for comparison or debugging purposes. This can be checked using accuracy metric.

**Margin:**

The margin is a **concept** in SVM that refers to the separation between the decision boundary (hyperplane) and the support vectors. It represents the width of the region around the decision boundary where no training examples exist. SVM aims to find the hyperplane that maximizes the margin, as it leads to better generalization and robustness. The margin is an intrinsic characteristic of the SVM model and is not directly set by the user.



margin

The regularization parameter (C) and gamma parameter are hyperparameters that influence the behaviour and performance of the SVM model. The kernel, degree, and random state are parameters that affect the choice of SVM algorithm and its behaviour. The margin, on the other hand, is a concept that describes the separation between the decision boundary and the support vectors.

***Also for the model evaluation the metrices like accuracy, recall, precision, f1-score…etc can be used. (check them***[***here***](https://medium.com/@sandundayananda/logistic-regression-55512384851b)***)***