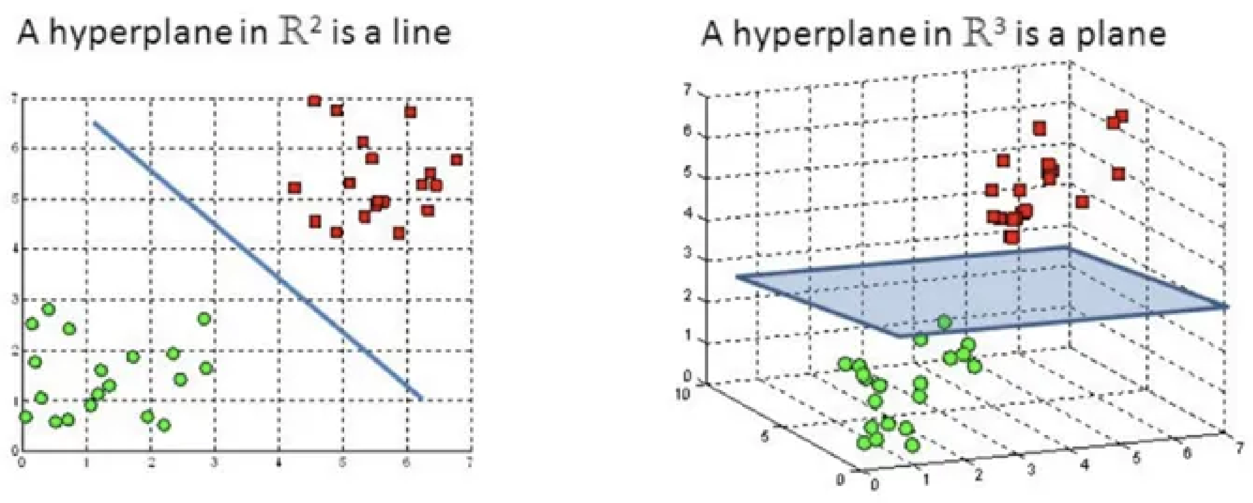
Support Vector Machine (SVM)

om pramod · Follow
10 min read · Jan 21

Support Vector Machine or SVM is one of the most popular Supervised Learning algorithms, which is used for Classification as well as Regression problems. However, primarily, it is used for Classification problems in Machine Learning.

> *Terminologies used in SVM:*

· **Hyperplane**: In SVM, a hyperplane is a decision boundary that separates the data into different classes. The goal of SVM is to find the best hyperplane that maximally separates the different classes. In a two-dimensional space, a hyperplane is a straight line, and in a three-dimensional space, it is a plane, and so on. A good rule of thumb is that for an n-dimensional space, the hyperplane will generally have an n-1 dimension.



reference

· **Margin:** The margin is the distance between the hyperplane and the closest data points from each class. The goal of SVM is to find the hyperplane that maximizes the margin, which helps to ensure that the decision boundary is as far away as possible from the data points. The larger the margin, the better the separation of the classes.

Here’s a simple mathematical example of how an SVM classifies data with two features (x1 and x2) and two classes (A and B):

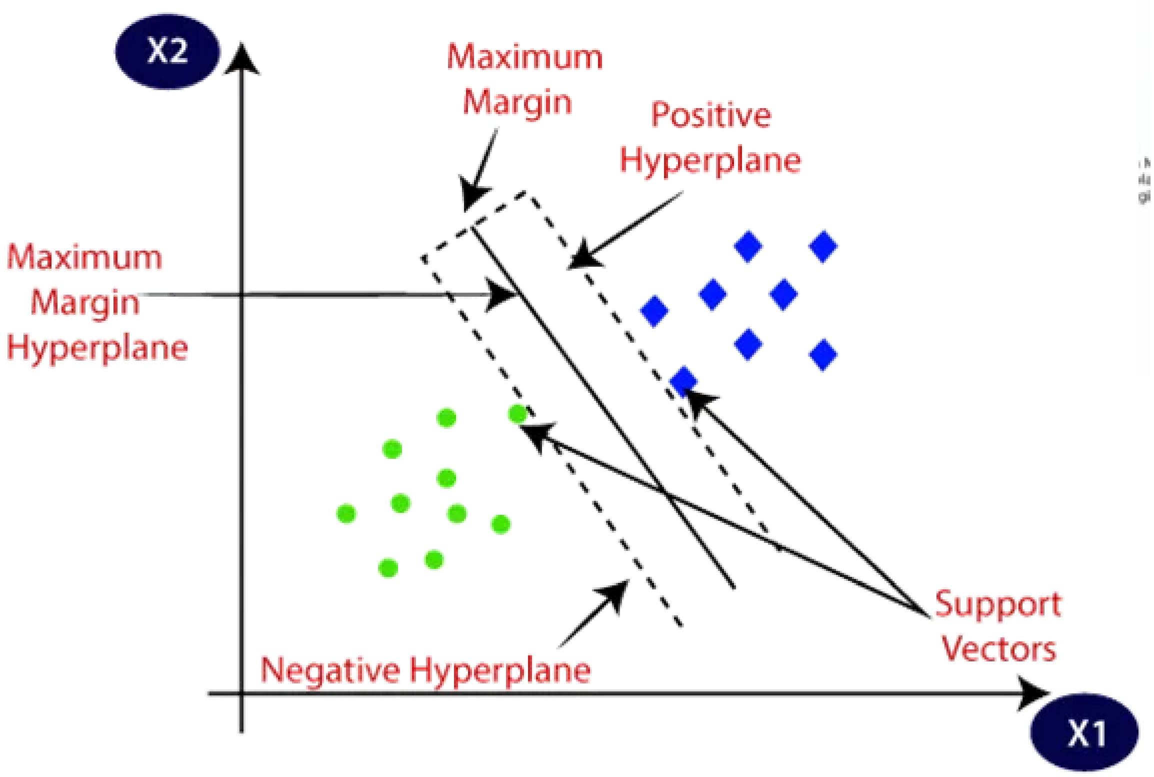
The goal is to find the best boundary (or hyperplane) that separates the two classes. The equation for the hyperplane is given by:

w1*x1 + w2*x2 + b = 0

where w1 and w2 are the weights assigned to the features x1 and x2, and b is the bias term.

The SVM algorithm finds the weights and bias that maximize the margin.

· **Support Vectors:** Support vectors are the data points that are closest to the decision boundary (or hyperplane). These points have a special role in determining the location of the decision boundary because the goal of SVM is to find the hyperplane that maximally separates the different classes by maximizing the margin, which is the distance between the hyperplane and the closest data points from each class.



Reference

· **Soft Margin:** In a hard-margin SVM, the goal is to find a decision boundary that separates the two classes with the largest possible margin However, in some cases, the data may not be perfectly separable and there may be some overlap between the two classes. In these cases, a hard-margin SVM may not be able to find a decision boundary that separates the two classes perfectly. A soft-margin SVM, on the other hand refers to the ability of SVM to tolerate misclassifications by allowing some data points to be on the wrong side of the decision boundary. This is achieved by introducing a regularization parameter.

· **Regularization parameter(C):** The regularization parameter is a tunable parameter that controls the trade-off between maximizing the margin and minimizing the number of misclassifications. A smaller value of C will result in a larger margin but more misclassification, while a larger value of C will result in a smaller margin but less misclassification. By default, the value of C is set to 1, which corresponds to a hard-margin SVM. To use a soft-margin SVM, you can set a smaller value of C, such as 0.1.

Types of SVM

There are two main types of SVM: linear SVM and non-linear SVM.

1. Linear SVM

The Linear Support Vector Machine algorithm is used when we have linearly separable data. In simple language, if we have a dataset that can be classified into two groups using a simple straight line, we call it linearly separable data, and the classifier used for this is known as Linear SVM Classifier.

2. Kernel or Non-Linear SVM

The non-linear support vector machine algorithm is used when we have non-linearly separable data. In simple language, if we have a dataset that cannot be classified into two groups using a simple straight line, we call it non-linear separable data, and the classifier used for this is known as a Non-Linear SVM classifier.

Kernel functions:

In some cases, the data may not be linearly separable and a linear decision boundary may not be able to separate the two classes correctly. Kernel SVM addresses this issue by introducing a kernel function, which is a mathematical function that transforms the input data into a higher-dimensional space, where it may be easier to find a linear decision boundary. Thus kernel function is used to move to new dimensions. The goal of the kernel function is to map the input features in such a way that the data becomes linearly separable in the transformed space. Common kernel functions include linear, polynomial, radial basis function (RBF), and sigmoid.

1. **Linear kernel**: The linear kernel is the simplest kernel function. It computes the dot product of the input features, which is equivalent to a linear SVM. A linear kernel is used when the data is linearly separable.

```
from sklearn.svm import SVC
clf = SVC(kernel='linear', C=1)
```

The linear kernel can be defined as:

K(x, x') = xT*x'

where x and x' are input features, and T denotes the transpose of a matrix.

2. Polynomial kernel: The polynomial kernel is used to capture non-linear relationships between the input features. It computes the dot product of the input features raised to a specified power.

```
from sklearn.svm import SVC
clf = SVC(kernel='poly', degree=2, coef0=1, C=1, gamma='auto')
```

The polynomial kernel can be defined as:

K(x, x') = (gamma * xT * x' + coef0)^d

where x and x’ are input features, T denotes the transpose of a matrix, gamma is a kernel coefficient for ‘rbf’, ‘poly’ and ‘sigmoid’. if gamma=’scale’ (default) is passed then it uses 1 / (n_features * X.var()) as value of gamma, if ‘auto’, uses 1 / n_features coef0 is a constant term, and d is the degree of the polynomial, c is a constant

note: The default value of gamma is ‘scale’, this value is chosen to be optimal for most of the datasets, but it’s worth trying different values if the performance is not optimal.

coef0 is a parameter that is used in the polynomial and sigmoid kernel functions in Support Vector Machine (SVM) algorithm.

In the polynomial kernel function, coef0 is a constant term that is added to the dot product of the input features before raising it to the power of the degree. It controls the position of the decision boundary. A positive value of coef0 shifts the decision boundary to the positive side and a negative value of coef0 shifts the decision boundary to the negative side.

3. Sigmoid kernel: It computes the dot product of the input features after applying a sigmoid function to the input features.

```
from sklearn.svm import SVC
clf = SVC(kernel='sigmoid', gamma=1, coef0=1, C=1)
```

The sigmoid kernel function can be defined as:

K(x, x’) = tanh(gamma * xT * x’ + coef0)

where x and x’ are input features, T denotes the transpose of a matrix, gamma is a parameter that controls the slope of the sigmoid function, and In the sigmoid kernel function, coef0 is a constant that is added to the dot product of the input features before applying the sigmoid function.

4. Gaussian Radial Basis Function (RBF): In scikit-learn library, the Gaussian RBF kernel can be set by passing ‘rbf’ as the value of the kernel parameter of the SVC class and specifying the value of gamma using gamma parameter :

```
from sklearn.svm import SVC
clf = SVC(kernel='rbf', C=1, gamma=1)
```

The Gaussian RBF is defined as follows:

$K(x, x') = \exp(-\text{gamma} * ||x — x'||^2)$

Where x and x' are input features, ||x — x'|| is the Euclidean distance between the input features

Note: In Support Vector Machine (SVM), a kernel function, also known as a similarity function. A similarity function maps the input data into a higher-dimensional feature space by computing a dot product between the input data and a set of basis functions. The dot product is a measure of similarity between the input data and the basis functions. The dot product between two vectors is high when they are similar and low when they are dissimilar.

To summarize, The are two main types of classification SVM algorithms Hard Margin and Soft Margin:

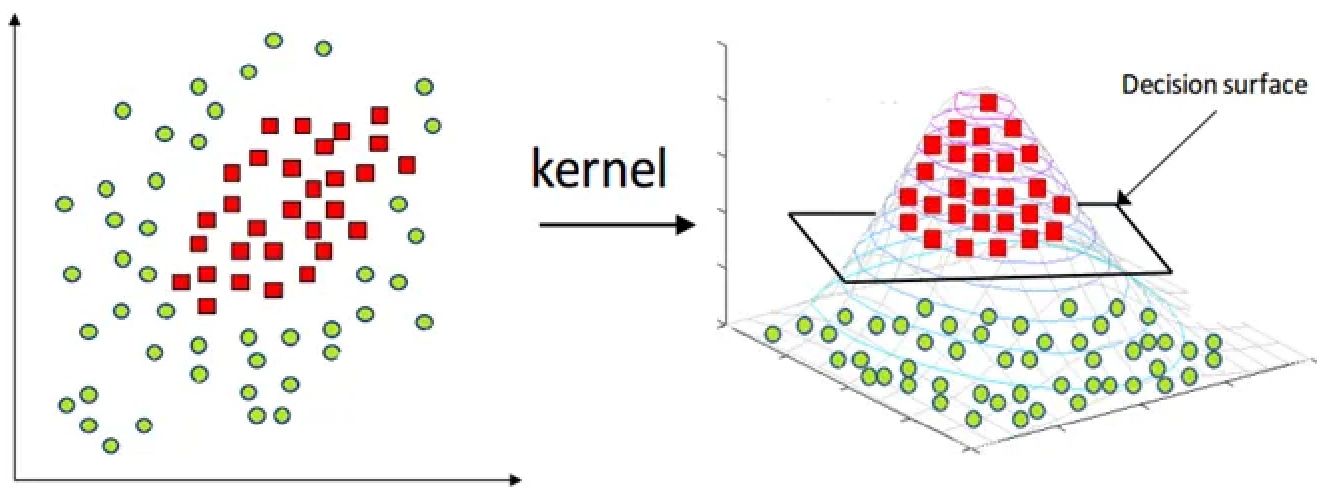
Hard Margin: aims to find the best hyperplane without tolerating any form of misclassification.

Soft Margin: we add a degree of tolerance in SVM. In this way we allow the model to voluntarily misclassify a few data points if that can lead to identifying a hyperplane able to generalise better to unseen data.

Soft Margin SVM can be implemented in Scikit-Learn by adding a C penalty term in svm.SVC. The bigger C and the more penalty the algorithm gets when making a misclassification.

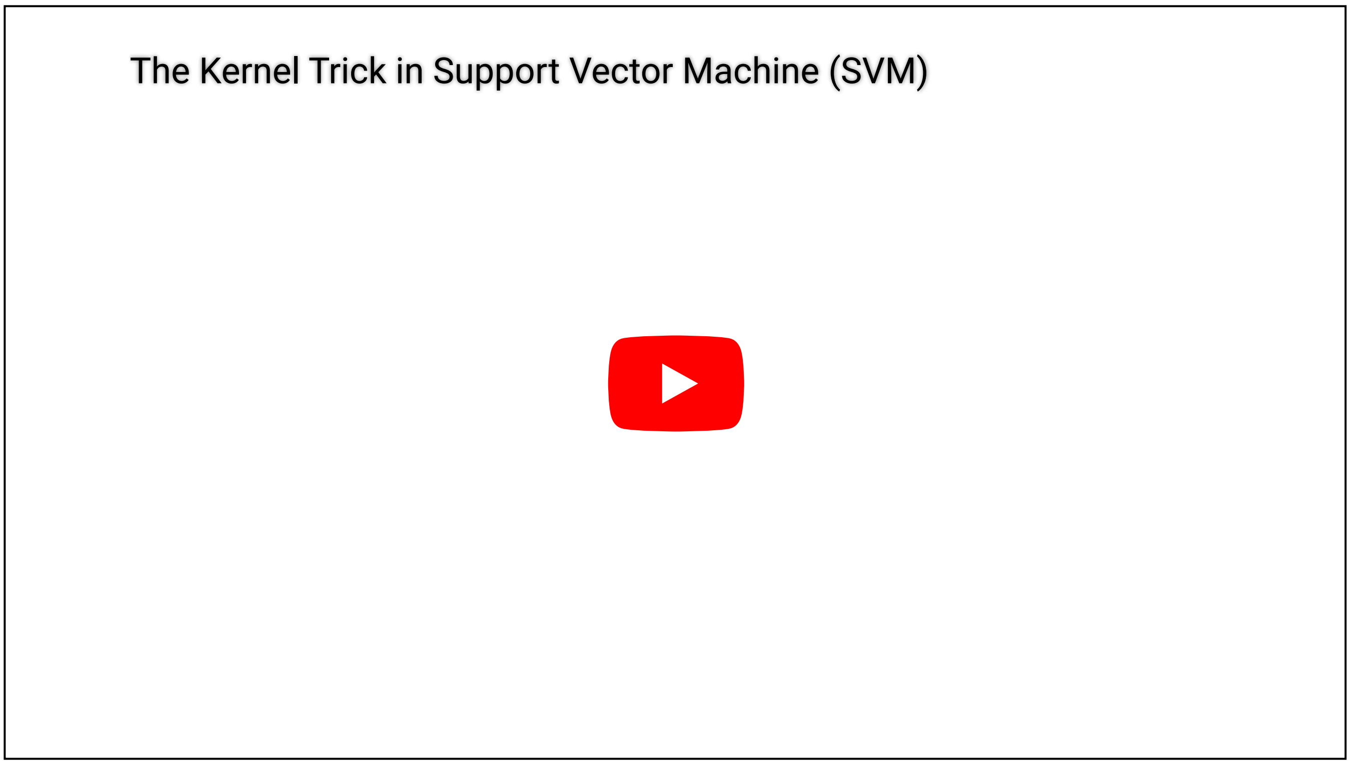
> *Finding Non-Linear Decision Boundary:*

One way to find a non-linear decision boundary in SVM is to use a kernel trick. For example, let's say we have a dataset with two classes, class A and class B, that are not linearly separable in their original feature space. By using an RBF kernel, we can map the input data into a higher-dimensional space where a linear decision boundary can be found to separate class A from class B.



Reference

For a better understanding of Kernel Trick watch these videos:



Choosing Landmark Points in SVM:

In Support Vector Machine (SVM) algorithm, landmark points, also known as support vectors, are the points in the input data that are closest to the decision boundary and have the greatest impact on determining the location of the decision boundary. The kernel trick is accomplished by choosing "landmark points" in the data. It's worth noting that, the landmark points are chosen by the kernel function in SVM automatically, so you don't need to choose them explicitly. In scikit-learn, once an SVM model is trained, the support vectors can be accessed through the support_vectors_ attribute of the model object. Here is an example:

```python
from sklearn.svm import SVC
from sklearn.datasets import make_classification

# create a synthetic data set
X, y = make_classification(n_samples=1000, n_features=2, n_informative=2, n_rec

# fit the model
model = SVC(kernel='linear', C=1, random_state=42)
model.fit(X, y)

# access the support vectors
support_vectors = model.support_vectors_
print("Support Vectors: \n", support_vectors)
```

A good choice of landmark points will lead to a well-separated boundary, while a bad choice will result in a poor separation.

> *Multiclass Classification Using SVM:*

In its most simple type, SVM doesn't support multiclass classification natively. It supports binary classification and separating data points into two classes. Support Vector Machines (SVMs) can be used for multi-class classification by using various techniques such as One-vs-All (also known as One-vs-Rest), and One-vs-One.

one-vs-All method is a way to perform multi-class classification by training multiple binary classifiers, where each classifier is trained to separate one class from all other classes.

For example, let's say we want to classify images of animals into 4 different classes: cats, dogs, birds and fish. Using the One-vs-All method, we would train 4 different binary classifiers:

A classifier that separates cats from non-cats (dogs, birds, fish)

A classifier that separates dogs from non-dogs (cats, birds, fish)

A classifier that separates birds from non-birds (cats, dogs, fish)

A classifier that separates fish from non-fish (cats, dogs, birds)

When we want to classify a new image, we would input it into all 4 classifiers, and the classifier that outputs the highest score would be the final prediction for the class of the image.

```
from sklearn.svm import SVC
from sklearn.datasets import make_classification
from sklearn.multiclass import OneVsRestClassifier

# create a synthetic data set
X, y = make_classification(n_samples=1000, n_features=2, n_informative=2, n_rec

# fit the model using the One-vs-All method
```

```
model = OneVsRestClassifier(SVC(kernel='linear', C=1, random_state=42))
model.fit(X, y)
```

One-vs-One is another method for performing multi-class classification using Support Vector Machines (SVMs) or other binary classifiers. The basic idea is to train a binary classifier for each pair of classes. For example, if you have three classes, you would train three classifiers:

Classifier 1: separates class 1 from class 2

Classifier 2: separates class 1 from class 3

Classifier 3: separates class 2 from class 3

When a new example is encountered, all the classifiers are used to make a prediction. The class that is predicted by the most classifiers wins the final prediction.

```
from sklearn.multiclass import OneVsOneClassifier

# fit the model using the One-vs-One method
model = OneVsOneClassifier(SVC(kernel='linear', C=1, random_state=42))
model.fit(X, y)
```

note that, One-vs-One method is more computationally expensive than One-vs-All method, as it requires training n(n-1)/2 classifiers for n classes, but it can lead to better performance for some datasets.

Here is the implementation of SVM to perform multiclass classification on iris dataset using Grid Search:

```
from sklearn.datasets import load_iris
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score
```

```
iris = load_iris()
X, y = iris.data, iris.target

# split the data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, randon

params_grid = [{'kernel': ['rbf'], 'gamma': [1e-3, 1e-4],
                     'C': [1, 10, 100, 1000]},
                    {'kernel': ['linear'], 'C': [1, 10, 100, 1000]}]

# Performing CV to tune parameters for best SVM fit
svm_model = GridSearchCV(SVC(), params_grid, cv=5)
svm_model.fit(X_train, y_train)

# View the accuracy score
print('Best score for training data:', svm_model.best_score_,"\n")

# View the best parameters for the model found using grid search
print('Best C:',svm_model.best_estimator_.C,"\n")
print('Best Kernel:',svm_model.best_estimator_.kernel,"\n")
print('Best Gamma:',svm_model.best_estimator_.gamma,"\n")

final_model = svm_model.best_estimator_
# make predictions on the test set
y_pred = final_model.predict(X_test)

# evaluate the model's performance
acc = accuracy_score(y_test, y_pred)
print('Accuracy:', acc)
```

output may look like this-

```
Best score for training data: 0.9583333333333334

Best C: 1000

Best Kernel: rbf

Best Gamma: 0.001

Accuracy: 1.0
```

Note: params_grid can be written in another way as-

```
params_grid = {'kernel': {'rbf': {'gamma': [1e-3, 1e-4], 'C': [1, 10, 100, 1000
'linear': {'C': [1, 10, 100, 1000]}}}
```

Here, The grid search is set to explore different combinations of two types of kernel functions: 'rbf' (Radial basis function) and 'linear'.

For the 'rbf' kernel, the grid search will try different combinations of two hyperparameters: gamma and C

For the 'linear' kernel, the grid search will try different combinations of one hyperparameter: C

Final Note: Thanks for reading! I hope you find this article informative.

Wanna connect with me? Hit me up on LinkedIn

Machine Learning Svm Support Vector Machine

Follow

Written by om pramod

35 Followers

AIML enthusiast