# **Random Forest Algorithm**

Random Forest is a popular machine learning algorithm that belongs to the supervised learning technique. It can be used for both Classification and Regression problems in ML. It is based on the concept of **ensemble learning,** which is a process of combining multiple classifiers to solve a complex problem and to improve the performance of the model.

As the name suggests, ***"Random Forest is a classifier that contains a number of decision trees on various subsets of the given dataset and takes the average to improve the predictive accuracy of that dataset."*** Instead of relying on one decision tree, the random forest takes the prediction from each tree and based on the majority votes of predictions, and it predicts the final output.

**The greater number of trees in the forest leads to higher accuracy and prevents the problem of overfitting.**

The below diagram explains the working of the Random Forest algorithm:



*Note: To better understand the Random Forest Algorithm, you should have knowledge of the Decision Tree Algorithm.*

Assumptions for Random Forest

Since the random forest combines multiple trees to predict the class of the dataset, it is possible that some decision trees may predict the correct output, while others may not. But together, all the trees predict the correct output. Therefore, below are two assumptions for a better Random forest classifier:

* There should be some actual values in the feature variable of the dataset so that the classifier can predict accurate results rather than a guessed result.
* The predictions from each tree must have very low correlations.

Why use Random Forest?

Below are some points that explain why we should use the Random Forest algorithm:

* It takes less training time as compared to other algorithms.
* It predicts output with high accuracy, even for the large dataset it runs efficiently.
* It can also maintain accuracy when a large proportion of data is missing.

How does Random Forest algorithm work?

Random Forest works in two-phase first is to create the random forest by combining N decision tree, and second is to make predictions for each tree created in the first phase.

The Working process can be explained in the below steps and diagram:

**Step-1:** Select random K data points from the training set.

**Step-2:** Build the decision trees associated with the selected data points (Subsets).

**Step-3:** Choose the number N for decision trees that you want to build.

**Step-4:** Repeat Step 1 & 2.

**Step-5:** For new data points, find the predictions of each decision tree, and assign the new data points to the category that wins the majority votes.

The working of the algorithm can be better understood by the below example:

**Example:** Suppose there is a dataset that contains multiple fruit images. So, this dataset is given to the Random forest classifier. The dataset is divided into subsets and given to each decision tree. During the training phase, each decision tree produces a prediction result, and when a new data point occurs, then based on the majority of results, the Random Forest classifier predicts the final decision. Consider the below image:



Applications of Random Forest

There are mainly four sectors where Random forest mostly used:

1. **Banking:** Banking sector mostly uses this algorithm for the identification of loan risk.
2. **Medicine:** With the help of this algorithm, disease trends and risks of the disease can be identified.
3. **Land Use:** We can identify the areas of similar land use by this algorithm.
4. **Marketing:** Marketing trends can be identified using this algorithm.

Advantages of Random Forest

* Random Forest is capable of performing both Classification and Regression tasks.
* It is capable of handling large datasets with high dimensionality.
* It enhances the accuracy of the model and prevents the overfitting issue.

Disadvantages of Random Forest

* Although random forest can be used for both classification and regression tasks, it is not more suitable for Regression tasks.

Python Implementation of Random Forest Algorithm

Now we will implement the Random Forest Algorithm tree using Python. For this, we will use the same dataset "user\_data.csv", which we have used in previous classification models. By using the same dataset, we can compare the Random Forest classifier with other classification models such as [Decision tree Classifier,](https://www.javatpoint.com/machine-learning-decision-tree-classification-algorithm) [KNN,](https://www.javatpoint.com/k-nearest-neighbor-algorithm-for-machine-learning)[SVM,](https://www.javatpoint.com/machine-learning-support-vector-machine-algorithm) [Logistic Regression,](https://www.javatpoint.com/logistic-regression-in-machine-learning) etc.

Implementation Steps are given below:

* Data Pre-processing step
* Fitting the Random forest algorithm to the Training set
* Predicting the test result
* Test accuracy of the result (Creation of Confusion matrix)
* Visualizing the test set result.

1.Data Pre-Processing Step:

Below is the code for the pre-processing step:

1. # importing libraries
2. **import** numpy as nm
3. **import** matplotlib.pyplot as mtp
4. **import** pandas as pd
6. #importing datasets
7. data\_set= pd.read\_csv('user\_data.csv')
9. #Extracting Independent and dependent Variable
11. # Splitting the dataset into training and test set.
12. from sklearn.model\_selection **import** train\_test\_split
13. x\_train, x\_test, y\_train, y\_test= train\_test\_split(x, y, test\_size= 0.25, random\_state=0)
15. #feature Scaling
16. from sklearn.preprocessing **import** StandardScaler
17. st\_x= StandardScaler()
18. x\_train= st\_x.fit\_transform(x\_train)
19. x\_test= st\_x.transform(x\_test)

In the above code, we have pre-processed the data. Where we have loaded the dataset, which is given as:

2. Fitting the Random Forest algorithm to the training set:

Now we will fit the Random forest algorithm to the training set. To fit it, we will import the**RandomForestClassifier**class from the **sklearn.ensemble** library. The code is given below:

1. #Fitting Decision Tree classifier to the training set
2. from sklearn.ensemble **import** RandomForestClassifier
3. classifier= RandomForestClassifier(n\_estimators= 10, criterion="entropy")
4. classifier.fit(x\_train, y\_train)

In the above code, the classifier object takes below parameters:

* **n\_estimators=** The required number of trees in the Random Forest. The default value is 10. We can choose any number but need to take care of the overfitting issue.
* **criterion=** It is a function to analyze the accuracy of the split. Here we have taken "entropy" for the information gain.

3. Predicting the Test Set result

Since our model is fitted to the training set, so now we can predict the test result. For prediction, we will create a new prediction vector y\_pred. Below is the code for it:

1. #Predicting the test set result
2. y\_pred= classifier.predict(x\_test)

**Output:**

The prediction vector is given as:

By checking the above prediction vector and test set real vector, we can determine the incorrect predictions done by the classifier.

4. Creating the Confusion Matrix

Now we will create the confusion matrix to determine the correct and incorrect predictions. Below is the code for it:

1. #Creating the Confusion matrix
2. from sklearn.metrics **import** confusion\_matrix
3. cm= confusion\_matrix(y\_test, y\_pred)

## Random Forest vs. Other Machine Learning Algorithms

Some of the key-differences are discussed below.

| **Feature** | **Random Forest** | **Other ML Algorithms** |
| --- | --- | --- |
| Ensemble Approach | Utilizes an ensemble of decision trees, combining their outputs for predictions, fostering robustness and accuracy. | Typically relies on a single model (e.g., linear regression, support vector machine) without the ensemble approach, potentially leading to less resilience against noise. |
| Overfitting Resistance | Resistant to overfitting due to the aggregation of diverse decision trees, preventing memorization of training data. | Some algorithms may be prone to overfitting, especially when dealing with complex datasets, as they may excessively adapt to training noise. |
| Handling of Missing Data | Exhibits resilience in handling missing values by leveraging available features for predictions, contributing to practicality in real-world scenarios. | Other algorithms may require imputation or elimination of missing data, potentially impacting model training and performance. |

# Support Vector Machine (SVM) Algorithm

**Last Updated :**10 Oct, 2024

A **Support Vector Machine (SVM)** is a powerful **machine learning algorithm** widely used for both **linear and nonlinear classification**, as well as **regression** and **outlier detection** tasks. SVMs are highly adaptable, making them suitable for various applications such as **text classification**, **image classification**, **spam detection**, **handwriting identification**, **gene expression analysis**, **face detection**, and **anomaly detection**.

SVMs are particularly effective because they focus on finding the **maximum separating hyperplane** between the different classes in the target feature, making them robust for both **binary and multiclass classification**. In this outline, we will explore the **Support Vector Machine (SVM)** algorithm, its applications, and how it effectively handles both **linear and nonlinear classification**, as well as **regression** and **outlier detection** tasks.

## Support Vector Machine

A **Support Vector Machine (SVM)** is a [supervised machine learning](https://www.geeksforgeeks.org/supervised-unsupervised-learning/)**algorithm** used for both **classification** and **regression** tasks. While it can be applied to regression problems, SVM is best suited for **classification** tasks. The primary objective of the **SVM algorithm** is to identify the **optimal hyperplane** in an N-dimensional space that can effectively separate data points into different classes in the feature space. The algorithm ensures that the margin between the closest points of different classes, known as **support vectors**, is maximized.

The dimension of the [hyperplane](https://www.geeksforgeeks.org/separating-hyperplanes-in-svm/) depends on the number of features. For instance, if there are two input features, the hyperplane is simply a line, and if there are three input features, the hyperplane becomes a 2-D plane. As the number of features increases beyond three, the complexity of visualizing the hyperplane also increases.

Consider two independent variables, **x1** and **x2**, and one dependent variable represented as either a blue circle or a red circle.

* In this scenario, the hyperplane is a line because we are working with two features (**x1** and **x2**).
* There are multiple lines (or **hyperplanes**) that can separate the data points.
* The challenge is to determine the **best hyperplane** that maximizes the separation margin between the red and blue circles.



*Linearly Separable Data points*

From the figure above it’s very clear that there are multiple lines (our hyperplane here is a line because we are considering only two input features x1, x2) that segregate our data points or do a classification between red and blue circles. ***So how do we choose the best line or in general the best hyperplane that segregates our data points?***

## How does Support Vector Machine Algorithm Work?

One reasonable choice for the **best hyperplane** in a **Support Vector Machine (SVM)** is the one that maximizes the **separation margin** between the two classes. The **maximum-margin hyperplane**, also referred to as the **hard margin**, is selected based on maximizing the distance between the hyperplane and the nearest data point on each side.



*Multiple hyperplanes separate the data from two classes*

So we choose the hyperplane whose distance from it to the nearest data point on each side is maximized. If such a hyperplane exists it is known as the **maximum-margin hyperplane/hard margin**. So from the above figure, we choose L2. Let’s consider a scenario like shown below



*Selecting hyperplane for data with outlier*

Here we have one blue ball in the boundary of the red ball. So how does SVM classify the data? It’s simple! The blue ball in the boundary of red ones is an outlier of blue balls. The SVM algorithm has the characteristics to ignore the outlier and finds the best hyperplane that maximizes the margin. SVM is robust to outliers.



*Hyperplane which is the most optimized one*

So in this type of data point what SVM does is, finds the maximum margin as done with previous data sets along with that it adds a penalty each time a point crosses the margin. So the margins in these types of cases are called **soft margins**. When there is a soft margin to the data set, the SVM tries to minimize (1/margin+∧(∑penalty)). Hinge loss is a commonly used penalty. If no violations no hinge loss.If violations hinge loss proportional to the distance of violation.

Till now, we were talking about linearly separable data(the group of blue balls and red balls are separable by a straight line/linear line). What to do if data are not linearly separable?



*Original 1D dataset for classification*

Say, our data is shown in the figure above. SVM solves this by creating a new variable using a **kernel**. We call a point xi on the line and we create a new variable yi as a function of distance from origin o.so if we plot this we get something like as shown below



*Mapping 1D data to 2D to become able to separate the two classes*

In this case, the new variable y is created as a function of distance from the origin. A non-linear function that creates a new variable is referred to as a kernel.

## Support Vector Machine Terminology

* **Hyperplane**: The **hyperplane** is the decision boundary used to separate data points of different classes in a feature space. For **linear classification**, this is a linear equation represented as wx+b=0.
* **Support Vectors**: **Support vectors** are the closest data points to the hyperplane. These points are critical in determining the hyperplane and the margin in **Support Vector Machine (SVM)**.
* **Margin**: The **margin** refers to the distance between the **support vector** and the hyperplane. The primary goal of the SVM algorithm is to maximize this margin, as a wider margin typically results in better classification performance.
* **Kernel**: The **kernel** is a mathematical function used in SVM to map input data into a higher-dimensional feature space. This allows the SVM to find a hyperplane in cases where data points are not linearly separable in the original space. Common **kernel functions** include linear, polynomial, radial basis function (RBF), and sigmoid.
* **Hard Margin**: A **hard margin** refers to the maximum-margin hyperplane that perfectly separates the data points of different classes without any misclassifications.
* **Soft Margin**: When data contains **outliers** or is not perfectly separable, SVM uses the **soft margin** technique. This method introduces a **slack variable** for each data point to allow some misclassifications while balancing between maximizing the margin and minimizing violations.
* **C**: The **C parameter** in SVM is a regularization term that balances margin maximization and the penalty for misclassifications. A higher **C** value imposes a stricter penalty for margin violations, leading to a smaller margin but fewer misclassifications.
* **Hinge Loss**: The **hinge loss** is a common loss function in SVMs. It penalizes misclassified points or margin violations and is often combined with a regularization term in the objective function.
* **Dual Problem**: The **dual problem** in SVM involves solving for the **Lagrange multipliers** associated with the support vectors. This formulation allows for the use of the **kernel trick** and facilitates more efficient computation.

## Mathematical Computation: SVM

Consider a binary classification problem with two classes, labeled as +1 and -1. We have a training dataset consisting of input feature vectors X and their corresponding class labels Y.

The equation for the linear hyperplane can be written as:

wTx+b=0*wTx*+*b*=0

The vector W represents the normal vector to the hyperplane. i.e the direction perpendicular to the hyperplane. The parameter **b** in the equation represents the offset or distance of the hyperplane from the origin along the normal vector **w**.

The distance between a data point x\_i and the decision boundary can be calculated as:

di=wTxi+b∣∣w∣∣*di*​=∣∣*w*∣∣*wTxi*​+*b*​

where ||w|| represents the Euclidean norm of the weight vector w. Euclidean norm of the normal vector W

For Linear SVM classifier :

y^={1: wTx+b≥00:  wTx+b <0*y*^​={10​: *wTx*+*b*≥0:  *wTx*+*b* <0​

#### ****Optimization****

* **For Hard margin linear SVM classifier:**

minimizew,b12wTw=minimizeW,b12∥w∥2subject toyi(wTxi+b)≥1fori=1,2,3,⋯,m*w*,*b*minimize​21​*wTw*=*W*,*b*minimize​21​∥*w*∥2subject to*yi*​(*wTxi*​+*b*)≥1*fori*=1,2,3,⋯,*m*

The target variable or label for the ith training instance is denoted by the symbol ti in this statement. And ti=-1 for negative occurrences (when yi= 0) and ti=1positive instances (when yi = 1) respectively. Because we require the decision boundary that satisfy the constraint:  ti(wTxi+b)≥1*ti*​(*wTxi*​+*b*)≥1

* **For Soft margin linear SVM classifier:**

minimize w,b12wTw+C∑i=1mζisubject to yi(wTxi+b)≥ 1−ζiandζi≥0fori=1,2,3,⋯,m*w*,*b*minimize ​21​*wTw*+*C*∑*i*=1*m*​*ζi*​subject to *yi*​(*wTxi*​+*b*)≥ 1−*ζi*​*andζi*​≥0*fori*=1,2,3,⋯,*m*

* **Dual Problem:** A dual Problem of the optimisation problem that requires locating the Lagrange multipliers related to the support vectors can be used to solve SVM. The optimal Lagrange multipliers α(i) that maximize the following dual objective function

maximizeα:12∑i→m∑j→mαiαjtitjK(xi,xj)−∑i→mαi*α*maximize​:21​*i*→*m*∑​*j*→*m*∑​*αi*​*αj*​*ti*​*tj*​*K*(*xi*​,*xj*​)−*i*→*m*∑​*αi*​

where,

* αi is the Lagrange multiplier associated with the ith training sample.
* K(xi, xj) is the kernel function that computes the similarity between two samples xi and xj. It allows SVM to handle nonlinear classification problems by implicitly mapping the samples into a higher-dimensional feature space.
* The term ∑αi represents the sum of all Lagrange multipliers.

The SVM decision boundary can be described in terms of these optimal Lagrange multipliers and the support vectors once the dual issue has been solved and the optimal Lagrange multipliers have been discovered. The training samples that have i > 0 are the support vectors, while the decision boundary is supplied by:

w=∑i→mαitiK(xi,x)+bti(wTxi−b)=1⟺b=wTxi−ti*w*=*i*→*m*∑​*αi*​*ti*​*K*(*xi*​,*x*)+*bti*​(*wTxi*​−*b*)=1⟺*b*=*wTxi*​−*ti*​

## Types of Support Vector Machine

Based on the nature of the decision boundary, Support Vector Machines (SVM) can be divided into two main parts:

* **Linear SVM:**Linear SVMs use a linear decision boundary to separate the data points of different classes. When the data can be precisely linearly separated, linear SVMs are very suitable. This means that a single straight line (in 2D) or a hyperplane (in higher dimensions) can entirely divide the data points into their respective classes. A hyperplane that maximizes the margin between the classes is the decision boundary.
* **Non-Linear SVM:** Non-Linear SVM can be used to classify data when it cannot be separated into two classes by a straight line (in the case of 2D). By using kernel functions, nonlinear SVMs can handle nonlinearly separable data. The original input data is transformed by these kernel functions into a higher-dimensional feature space, where the data points can be linearly separated. A linear SVM is used to locate a nonlinear decision boundary in this modified space.

## Popular kernel functions in SVM

The SVM kernel is a function that takes low-dimensional input space and transforms it into higher-dimensional space, ie it converts nonseparable problems to separable problems. It is mostly useful in non-linear separation problems. Simply put the kernel, does some extremely complex data transformations and then finds out the process to separate the data based on the labels or outputs defined.

Linear : K(w,b)=wTx+bPolynomial : K(w,x)=(γwTx+b)NGaussian RBF: K(w,x)=exp⁡(−γ∣∣xi−xj∣∣nSigmoid :K(xi,xj)=tanh⁡(αxiTxj+b)Linear : *K*(*w*,*b*)Polynomial : *K*(*w*,*x*)Gaussian RBF: *K*(*w*,*x*)Sigmoid :*K*(*xi*​,*xj*​)​=*wTx*+*b*=(*γwTx*+*b*)*N*=exp(−*γ*∣∣*xi*​−*xj*​∣∣*n*=tanh(*αxiT*​*xj*​+*b*)​

1