Random Forest

Random Forest is a machine learning algorithm that uses many decision trees to make better predictions. Each tree looks at different random parts of the data and their results are combined by voting for classification or averaging for regression which makes it as ensemble learning technique. This helps in improving accuracy and reducing errors.

Working of Random Forest Algorithm

- Create Many Decision Trees: The algorithm makes many decision trees each using a random part of the data. So every tree is a bit different.
- Pick Random Features: When building each tree it doesn't look at all the features (columns) at once. It picks a few at random to decide how to split the data. This helps the trees stay different from each other.
- Each Tree Makes a Prediction: Every tree gives its own answer or prediction based on what it learned from its part of the data.
- Combine the Predictions: For classification we choose a category as the final answer is the one that most trees agree on i.e majority voting and for regression we predict a number as the final answer is the average of all the trees predictions.
- Why It Works Well: Using random data and features for each tree helps avoid overfitting and makes the overall prediction more accurate and trustworthy.

Key Features of Random Forest

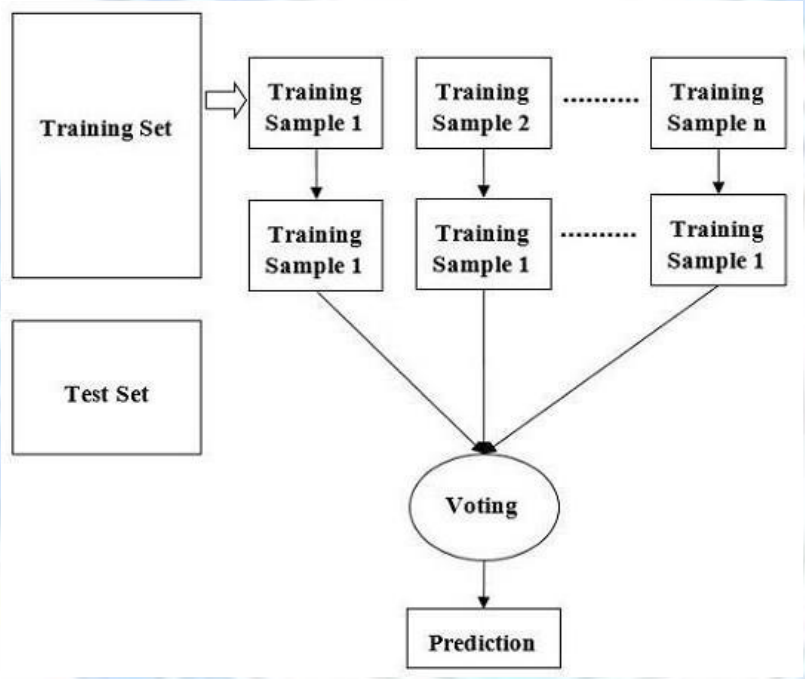
- Handles Missing Data: It can work even if some data is missing so you don't always need to fill in the gaps yourself.
- Shows Feature Importance: It tells you which features (columns) are most useful for making predictions which helps you understand your data better.
- Works Well with Big and Complex Data: It can handle large datasets with many features without slowing down or losing accuracy.
- Used for Different Tasks: You can use it for both classification like predicting types or labels and regression like predicting numbers or amounts.

Advantages of Random Forest

- Random Forest provides very accurate predictions even with large datasets.
- Random Forest can handle missing data well without compromising with accuracy.
- It doesn't require normalization or standardization on dataset.
- When we combine multiple decision trees it reduces the risk of overfitting of the model.

Limitations of Random Forest

- It can be computationally expensive especially with a large number of trees.
- It's harder to interpret the model compared to simpler models like decision trees.




```
import pandas as pd

from sklearn.ensemble import RandomForestClassifier

# Loading the iris dataset

iris = pd.read_csv('https://archive.ics.uci.edu/ml/machine-learningdatabases/iris/iris.data', header=None)

iris.columns = ['sepal_length', 'sepal_width', 'petal_length', 'petal_width', 'species']

# Separating the features and target variable

X = iris.iloc[:, :-1]
```

```
y = iris.iloc[:, -1]

# Splitting the data into training and testing sets

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y,

test_size=0.35, random_state=42)

# Creating the Random Forest classifier object

rfc = RandomForestClassifier(n_estimators=100)

# Training the model on the training data

rfc.fit(X_train, y_train)

# Making predictions on the test data

y_pred = rfc.predict(X_test)

# Importing the metrics library

from sklearn.metrics import accuracy_score, precision_score,

recall_score, f1_score

# Calculating the accuracy, precision, recall, and F1-score

accuracy = accuracy_score(y_test, y_pred)

precision = precision_score(y_test, y_pred, average='weighted')

recall = recall_score(y_test, y_pred, average='weighted')

f1 = f1_score(y_test, y_pred, average='weighted')

print("Accuracy:", accuracy)

print("Precision:", precision)

print("Recall:", recall)

print("F1-score:", f1)
```

Naive Bayes

Naive Bayes is a machine learning classification algorithm that predicts the category of a data point using probability. It assumes that all features are independent of each other. Naive Bayes performs well in many real-world applications such as spam filtering, document categorisation and sentiment analysis.

Key Features of Naive Bayes Classifiers

The main idea behind the Naive Bayes classifier is to use Bayes' Theorem to classify data based on the probabilities of different classes given the features of the data. It is used mostly in high-dimensional text classification

- The Naive Bayes Classifier is a simple probabilistic classifier and it has very few number of parameters which are used to build the ML models that can predict at a faster speed than other classification algorithms.
- It is a probabilistic classifier because it assumes that one feature in the model is independent of existence of another feature. In other words, each feature contributes to the predictions with no relation between each other.
- Naive Bayes Algorithm is used in spam filtration, Sentimental analysis, classifying articles and many more.

Introduction to Bayes' Theorem

Bayes' Theorem provides a principled way to reverse conditional probabilities. It is defined as:

$P(y|X) = P(X|y) \cdot P(y) / P(X)$

Where:

- $P(y|X)$: Posterior probability, probability of class y given features X
- $P(X|y)$: Likelihood, probability of features X given class y
- $P(y)$: Prior probability of class y
- $P(X)$: Marginal likelihood or evidence

Confusion Matrix in Machine Learning

Confusion matrix is a simple table used to measure how well a classification model is performing. It compares the predictions made by the model with the actual results and shows where the model was right or wrong. This helps you understand where the model is making mistakes so you can improve it. It breaks down the predictions into four categories:

- True Positive (TP): The model correctly predicted a positive outcome i.e the actual outcome was positive.
- True Negative (TN): The model correctly predicted a negative outcome i.e the actual outcome was negative.
- False Positive (FP): The model incorrectly predicted a positive outcome i.e the actual outcome was negative. It is also known as a Type I error.
- False Negative (FN): The model incorrectly predicted a negative outcome i.e the actual outcome was positive. It is also known as a Type II error.

	Predicted Positive	Predicted Negative
Actual Positive	True Positive (TP)	False Negative (FN)
Actual Negative	False Positive (FP)	True Negative (TN)

Confusion Matrix For Binary Classification

A 2x2 Confusion matrix is shown below for the image recognition having a Dog image or Not Dog image:

	Predicted	Predicted
Actual	True Positive (TP)	False Negative (FN)
Actual	False Positive (FP)	True Negative (TN)

- True Positive (TP): It is the total counts having both predicted and actual values are Dog.
- True Negative (TN): It is the total counts having both predicted and actual values are Not Dog.
- False Positive (FP): It is the total counts having prediction is Dog while actually Not Dog.
- False Negative (FN): It is the total counts having prediction is Not Dog while actually, it is Dog.

Example: Confusion Matrix for Dog Image Recognition with Numbers

Index	1	2	3	4	5	6	7	8	9	10
Actual	Dog	Dog	Dog	Not Dog	Dog	Not Dog	Dog	Dog	Not Dog	Not Dog
Predicted	Dog	Not Dog	Dog	Not Dog	Dog	Dog	Dog	Dog	Not Dog	Not Dog
Result	TP	FN	TP	TN	TP	FP	TP	TP	TN	TN

- Actual Dog Counts = 6
- Actual Not Dog Counts = 4
- True Positive Counts = 5
- False Positive Counts = 1
- True Negative Counts = 3
- False Negative Counts = 1

		Predicted	
		Dog	Not Dog
Actual	Dog	True Positive (TP =5)	False Negative (FN =1)
	Not Dog	False Positive (FP=1)	True Negative (TN=3)