Random Forest Classifier using Scikit-learn

Last Updated : 31 Jan, 2024

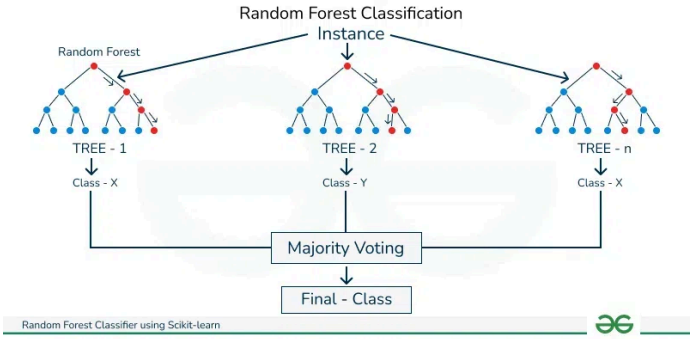
In this article, we will see how to build a **Random Forest Classifier** using the **Scikit-Learn library** of Python programming language and to do this, we use the **IRIS dataset** which is quite a common and famous dataset.

Random Forest

The Random forest or Random Decision Forest is a supervised Machine learning algorithm used for classification, regression, and other tasks using decision trees. Random Forests are particularly well-suited for handling large and complex datasets, dealing with high-dimensional feature spaces, and providing insights into feature importance. This algorithm's ability to maintain high predictive accuracy while minimizing overfitting makes it a popular choice across various domains, including finance, healthcare, and image analysis, among others.

Random Forest Classifier

The Random forest classifier creates a set of decision trees from a randomly selected subset of the training set. It is a set of decision trees (DT) from a randomly selected subset of the training set and then It collects the votes from different decision trees to decide the final prediction.



Random Forest Classifier

Additionally, the random forest classifier can handle both classification and regression tasks, and its ability to provide feature importance scores makes it a valuable tool for understanding the significance of different variables in the dataset.

How Random Forest Classification works

Random Forest Classification is an ensemble learning technique designed to enhance the accuracy and robustness of classification tasks. The algorithm builds a multitude of decision

the classification classes. Each decision tree in the random forest is constructed using a subset of the training data and a random subset of features introducing diversity among the trees, making the model more robust and less prone to overfitting.

The random forest algorithm employs a technique called bagging (Bootstrap Aggregating) to create these diverse subsets.

During the training phase, each tree is built by recursively partitioning the data based on the features. At each split, the algorithm selects the best feature from the random subset, optimizing for information gain or Gini impurity. The process continues until a predefined stopping criterion is met, such as reaching a maximum depth or having a minimum number of samples in each leaf node.

Once the random forest is trained, it can make predictions, using each tree “votes” for a class, and the class with the most votes becomes the predicted class for the input data.

Feature Selection in Random Forests

Feature selection in Random Forests is inherently embedded in the construction of individual decision trees and the aggregation process.

During the training phase, each decision tree is built using a random subset of features, contributing to diversity among the trees. The process is, known as feature bagging, helps prevent the dominance of any single feature and promotes a more robust model.

The algorithm evaluates various subsets of features at each split point, selecting the best feature for node splitting based on criteria such as information gain or Gini impurity. Consequently, Random Forests naturally incorporate a form of feature selection, ensuring that the ensemble benefits from a diverse set of features to enhance generalization and reduce overfitting.

Random Forest Classifier in Machine Learning

Step 1: Loading dataset

python3

```
# importing required libraries
# importing Scikit-learn library and datasets package
from sklearn import datasets

# Loading the iris plants dataset (classification)
iris = datasets.load_iris()
```

Step 2: Checking dataset content and features names present in it.

python3

```
print(iris.target_names)
```

Output:

```
['setosa' 'versicolor' 'virginica']
```

python3

```
print(iris.feature_names)
```

Output:

```
['sepal length (cm)', 'sepal width (cm)', 'petal
length (cm)', 'petal width (cm)']
```

Step 3: Train Test Split

python3

```
# dividing the datasets into two parts i.e. training da
X, y = datasets.load_iris( return_X_y = True)

# Splitting arrays or matrices into random train and te
from sklearn.model_selection import train_test_split
# i.e. 70 % training dataset and 30 % test datasets
X_train, X_test, y_train, y_test = train_test_split(X,
```

Step 4: Import Random Forest Classifier module.

python3

```
# importing random forest classifier from assemble modu
from sklearn.ensemble import RandomForestClassifier
import pandas as pd
# creating dataframe of IRIS dataset
data = pd.DataFrame({'sepallength': iris.data[:, 0], 's
                     'petallength': iris.data[:, 2], 'p
                     'species': iris.target})
```

Overview of the Dataset

python3

```
# printing the top 5 datasets in iris dataset
print(data.head())
```

Output:

```
   sepallength   sepalwidth   petallength
petalwidth   species
0          5.1            3.5              1.4
           0.2          0
1          4.9            3.0              1.4
           0.2          0
2          4.7            3.2              1.3
           0.2          0
3          4.6            3.1              1.5
          0.2           0
4          5.0            3.6              1.4
          0.2           0
```

Step 5: Training of Model

python3

```
# creating a RF classifier
clf = RandomForestClassifier(n_estimators = 100)

# Training the model on the training dataset
# fit function is used to train the model using the tra
clf.fit(X_train, y_train)

# performing predictions on the test dataset
y_pred = clf.predict(X_test)

# metrics are used to find accuracy or error
from sklearn import metrics
```

```
# using metrics module for accuracy calculation
print("ACCURACY OF THE MODEL:", metrics.accuracy_score(
```

Output:

```
ACCURACY OF THE MODEL: 0.9238095238095239
```

Step 6: Predictions

Python3

```
# predicting which type of flower it is.
clf.predict([[3, 3, 2, 2]])
```

Output:

```
array([0])
```

This implies it is **setosa** flower type as we got the three species or classes in our data set: **Setosa, Versicolor, and Virginia.**

Check the important features

Now we will also find out the important features or selecting features in the IRIS dataset by using the following lines of code.

python3

```
# using the feature importance variable
import pandas as pd
feature_imp = pd.Series(clf.feature_importances_, index
feature_imp
```

Output:

```
petal length (cm)    0.440050
petal width (cm)     0.423437
sepal length (cm)    0.103293
sepal width (cm)     0.033220
dtype: float64
```

Random Forests in Python's Scikit-Learn library come with a set of hyperparameters that allow you to fine-tune the behavior of the model. Understanding and selecting appropriate hyperparameters is crucial for optimizing model performance.

Random Forest Classifier Parameters

- **n_estimators:** Number of trees in the forest.
    - More trees generally lead to better performance, but at the cost of computational time.
    - Start with a value of 100 and increase as needed.
- **max_depth:** Maximum depth of each tree.
    - Deeper trees can capture more complex patterns, but also risk overfitting.
    - Experiment with values between 5 and 15, and consider lower values for smaller datasets.
- **max_features:** Number of features considered for splitting at each node.
    - A common value is 'sqrt' (square root of the total number of features).
    - Adjust based on dataset size and feature importance.
- **criterion:** Function used to measure split quality ('gini' or 'entropy').
    - Gini impurity is often slightly faster, but both are generally similar in performance.
- **min_samples_split:** Minimum samples required to split a node.
    - Higher values can prevent overfitting, but too high can hinder model complexity.
    - Start with 2 and adjust as needed.
- **min_samples_leaf:** Minimum samples required to be at a leaf node.
    - Similar to min_samples_split, but focused on leaf nodes.
    - Start with 1 and adjust as needed.
- **bootstrap:** Whether to use bootstrap sampling when building trees (True or False).
    - Bootstrapping can improve model variance and generalization, but can slightly increase bias.

Advantages of Random Forest Classifier

- The ensemble nature of Random Forests, combining multiple trees, makes them less prone to overfitting compared to individual decision trees.
- Effective on datasets with a large number of features, and it can handle irrelevant variables well.

- Random Forests can provide insights into feature importance, helping in feature selection and understanding the dataset.

Disadvantages of Random Forest Classifier

- Random Forests can be computationally expensive and may require more resources due to the construction of multiple decision trees.
- The ensemble nature makes it challenging to interpret the reasoning behind individual predictions compared to a single decision tree.
- In imbalanced datasets, Random Forests may be biased toward the majority class, impacting the predictive performance for minority classes.

Conclusion

In conclusion, Random Forests, with their ensemble of decision trees, stand out as a robust solution for various machine learning tasks, showcasing their versatility and effectiveness.

Frequently Asked Questions(FAQs)

Q. What is the random forest classifier?

Random Forest Classifier is an ensemble learning method using multiple decision trees for classification tasks, improving accuracy. It excels in handling complex data, mitigating overfitting, and providing robust predictions with feature importance.

Q. Can random forest be used for regression ?

Random Forest can be used for both regression and classification tasks, making it a versatile machine learning algorithm.

Q. What is the principle of random forest?

Random Forest builds multiple decision trees using random subsets of the dataset and combines their outputs for improved accuracy.

Q. What are the applications of random forest?