Random Forest Algorithm with Python and Scikit-Learn

Usman Malik

•

Random forest is a type of supervised machine learning algorithm based on ensemble learning. Ensemble learning is a type of learning where you join different types of algorithms or same algorithm multiple times to form a more powerful prediction model. The random forest algorithm combines multiple algorithm of the same type i.e. multiple decision *trees*, resulting in a *forest of trees*, hence the name "Random Forest". The random forest algorithm can be used for both regression and classification tasks.

How the Random Forest Algorithm Works

The following are the basic steps involved in performing the random forest algorithm:

1. Pick N random records from the dataset.
2. Build a decision tree based on these N records.
3. Choose the number of trees you want in your algorithm and repeat steps 1 and 2.
4. In case of a regression problem, for a new record,

each tree in the forest predicts a value for Y (output). The final value can be calculated by taking the average of all the values predicted by all the trees in forest. Or, in case of a classification problem, each tree in the forest predicts the category to which the new record belongs. Finally, the new record is assigned to the category that wins the majority vote.

Advantages of using Random Forest

As with any algorithm, there are advantages and disadvantages to using it. In the next two sections we'll take a look at the pros and cons of using random forest for classification and regression.

1. The random forest algorithm is not biased, since, there are multiple trees and each tree is trained on a subset of data. Basically, the random forest algorithm relies on the power of "the crowd"; therefore the overall biasedness of the algorithm is reduced.
2. This algorithm is very stable. Even if a new data point is introduced in the dataset the overall algorithm is not affected much since new data may impact one tree, but it is very hard for it to impact all the trees.
3. The random forest algorithm works well when you have both categorical and numerical features.
4. The random forest algorithm also works well when data has missing values or it has not been scaled well (although we have performed feature scaling in

this article just for the purpose of demonstration).

Disadvantages of using Random Forest

1. A major disadvantage of random forests lies in their complexity. They required much more computational resources, owing to the large number of decision trees joined together.
2. Due to their complexity, they require much more time to train than other comparable algorithms.

Throughout the rest of this article we will see how Python's Scikit-Learn library can be used to implement the random forest algorithm to solve regression, as well as classification, problems.

Part 1: Using Random Forest for Regression

In this section we will study how random forests can be used to solve regression problems using Scikit-Learn. In the next section we will solve classification problem via random forests.

Problem Definition

The problem here is to predict the gas consumption (in millions of gallons) in 48 of the US states based on petrol tax (in cents), per capita income (dollars), paved highways (in miles) and the proportion of population with the driving license.

Solution

To solve this regression problem we will use the random forest algorithm via the Scikit-Learn Python library. We will follow the traditional machine learning pipeline to solve this problem. Follow these steps:

1. Import Libraries

Execute the following code to import the necessary libraries:

```
import pandas as pd
import numpy as np
```

2. Importing Dataset

The dataset for this problem is available at:

https://drive.google.com/file/d/1mVmGNx6cbfvRHC_DvF12ZL3wGLSHD9f_/view

For the sake of this tutorial, the dataset has been downloaded into the "Datasets" folder of the "D" Drive. You'll need to change the file path according to your own setup.

Execute the following command to import the dataset:

```
dataset = pd.read_csv('D:\Datasets\petrol_consumption.csv')
```

To get a high-level view of what the dataset looks like, execute the following command:

```
dataset.head()
```

	Petrol_tax	Average_income	Paved_Highways	Popu
0	9.0	3571	1976	0.525
1	9.0	4092	1250	0.572
2	9.0	3865	1586	0.580
3	7.5	4870	2351	0.529
4	8.0	4399	431	0.544

We can see that the values in our dataset are not very well scaled. We will scale them down before training the algorithm.

3. Preparing Data For Training

Two tasks will be performed in this section. The first task is to divide data into 'attributes' and 'label' sets. The resultant data is then divided into training and test sets.

The following script divides data into attributes and labels:

```
X = dataset.iloc[:, 0:4].values
y = dataset.iloc[:, 4].values
```

Finally, let's divide 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, t
```

4. Feature Scaling

We know our dataset is not yet a scaled value, for instance the Average_Income field has values in the range of thousands while Petrol_tax has values in range of tens. Therefore, it would be beneficial to scale our data (although, as mentioned earlier, this step isn't as important for the random forests algorithm). To do so, we will use Scikit-Learn's `StandardScaler` class. Execute the following code to do so:

```
# Feature Scaling
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

5. Training the Algorithm

Now that we have scaled our dataset, it is time to train our random forest algorithm to solve this regression problem. Execute the following code:

```
from sklearn.ensemble import RandomForestRegressor
```

```
regressor = RandomForestRegressor(n_estimators=20, random_s
regressor.fit(X_train, y_train)
y_pred = regressor.predict(X_test)
```

The `RandomForestRegressor` class of the `sklearn.ensemble` library is used to solve regression problems via random forest. The most important parameter of the `RandomForestRegressor` class is the `n_estimators` parameter. This parameter defines the number of trees in the random forest. We will start with `n_estimator=20` to see how our algorithm performs. You can find details for all of the parameters of `RandomForestRegressor` here.

6. Evaluating the Algorithm

The last and final step of solving a machine learning problem is to evaluate the performance of the algorithm. For regression problems the metrics used to evaluate an algorithm are mean absolute error, mean squared error, and root mean squared error. Execute the following code to find these values:

```
from sklearn import metrics

print('Mean Absolute Error:', metrics.mean_absolute_error(y
print('Mean Squared Error:', metrics.mean_squared_error(y_t
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squa
```

The output will look something like this:

```
Mean Absolute Error: 51.765
Mean Squared Error: 4216.16675
Root Mean Squared Error: 64.932016371
```

With 20 trees, the root mean squared error is 64.93 which is greater than 10 percent of the average petrol consumption i.e. 576.77. This may indicate, among other things, that we have not used enough estimators (trees).

If the number of estimators is changed to 200, the results are as follows:

```
Mean Absolute Error: 47.9825
Mean Squared Error: 3469.7007375
Root Mean Squared Error: 58.9041657058
```

The following chart shows the decrease in the value of the root mean squared error (RMSE) with respect to number of estimators. Here the **X-axis contains the number of estimators** while the **Y-axis contains the value for root mean squared error**.

You can see that the error values decreases with the increase in number of estimator. After 200 the rate of decrease in error diminishes, so therefore 200 is a good number for `n_estimators`. You can play around with the number of trees and other parameters to see if you can get better results on your own.

Part 2: Using Random Forest for Classification

Problem Definition

The task here is to predict whether a bank currency note is authentic or not based on four attributes i.e. variance of the image wavelet transformed image, skewness, entropy, and curtosis of the image.

Solution

This is a binary classification problem and we will use a

random forest classifier to solve this problem. Steps followed to solve this problem will be similar to the steps performed for regression.

1. Import Libraries

```
import pandas as pd
import numpy as np
```

2. Importing Dataset

The dataset can be downloaded from the following link:

https://drive.google.com/file/d/13nw-uRXPY8XIZQxKRNZ3yYlho-CYm_Qt/view

The detailed information about the data is available at the following link:

https://archive.ics.uci.edu/ml/datasets/banknote+authentication

The following code imports the dataset:

```
dataset = pd.read_csv("D:/Datasets/bill_authentication.csv"
```

To get a high level view of the dataset, execute the following command:

```
dataset.head()
```

	Variance	Skewness	Curtosis	Entropy	Class
0	3.62160	8.6661	-2.8073	-0.44699	0
1	4.54590	8.1674	-2.4586	-1.46210	0
2	3.86600	-2.6383	1.9242	0.10645	0
3	3.45660	9.5228	-4.0112	-3.59440	0
4	0.32924	-4.4552	4.5718	-0.98880	0

As was the case with regression dataset, values in this dataset are not very well scaled. The dataset will be scaled before training the algorithm.

3. Preparing Data For Training

The following code divides data into attributes and labels:

```
X = dataset.iloc[:, 0:4].values
y = dataset.iloc[:, 4].values
```

The following code divides 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, t
```

4. Feature Scaling

As with before, feature scaling works the same way:

```
# Feature Scaling
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

5. Training the Algorithm

And again, now that we have scaled our dataset, we can train our random forests to solve this classification problem. To do so, execute the following code:

```
from sklearn.ensemble import RandomForestRegressor

regressor = RandomForestRegressor(n_estimators=20, random_s
regressor.fit(X_train, y_train)
y_pred = regressor.predict(X_test)
```

In case of regression we used the `RandomForestRegressor` class of the sklearn.ensemble library. For classification, we will `RandomForestClassifier` class of the sklearn.ensemble library. `RandomForestClassifier` class also takes `n_estimators` as a parameter. Like before, this parameter defines the number of trees in our random forest. We will start with 20 trees again. You can find details for all of the parameters of `RandomForestClassifier` here.

6. Evaluating the Algorithm

For classification problems the metrics used to evaluate an algorithm are accuracy, confusion matrix, precision recall, and F1 values. Execute the following script to find these values:

```
from sklearn.metrics import classification_report, confusio

print(confusion_matrix(y_test,y_pred))
print(classification_report(y_test,y_pred))
print(accuracy_score(y_test, y_pred))
```

The output will look something like this:

```
[[155    2]
    1  117]]
             precision   recall   f1-score   support

          0       0.99     0.99       0.99       157
          1       0.98     0.99       0.99       118

 avg / total       0.99     0.99       0.99       275

0.989090909091
```

The accuracy achieved for by our random forest classifier with 20 trees is 98.90%. Unlike before, changing the number of estimators for this problem didn't significantly improve the results, as shown in the following chart. Here

the X-axis contains the number of estimators while the Y-axis shows the accuracy.

98.90% is a pretty good accuracy, so there isn't much point in increasing our number of estimators anyway. We can see that increasing the number of estimators did not further improve the accuracy.

To improve the accuracy, I would suggest you to play around with other parameters of the `RandomForestClassifier` class and see if you can improve on our results.

Resources

Want to learn more about Scikit-Learn and other useful machine learning algorithms like random forests? You can check out some more detailed resources, like an online course:

- Data Science in Python, Pandas, Scikit-learn, Numpy, Matplotlib
- Python for Data Science and Machine Learning Bootcamp
- Machine Learning A-Z: Hands-On Python & R In Data Science

Courses like these give you the resources and quality of instruction you'd get in a university setting, but at your own pace, which is great for difficult topics like machine

learning.