Decision Tree vs. Random Forest - Which Algorithm Should you Use?

ALGORITHM BEGINNER CLASSIFICATION MACHINE LEARNING PYTHON STRUCTURED DATA SUPERVISED

A Simple Analogy to Explain Decision Tree vs. Random Forest

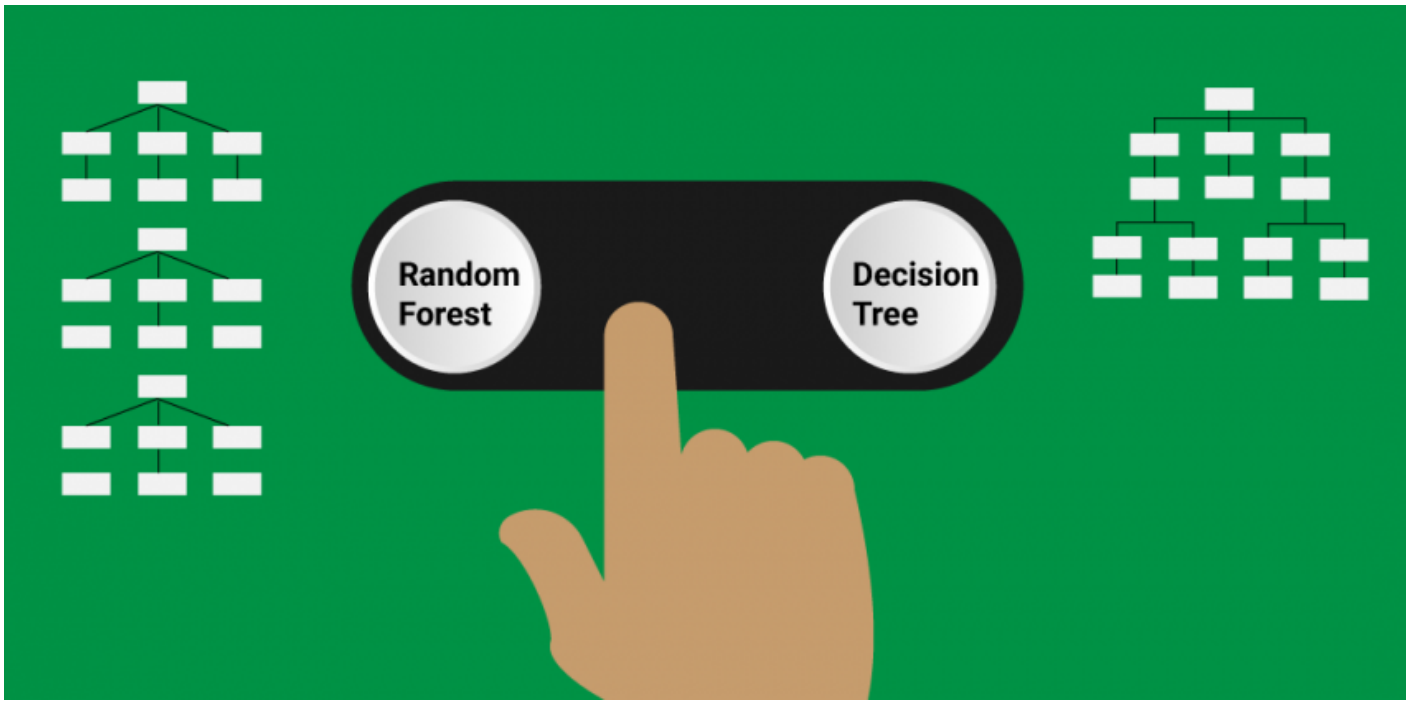
Let's start with a thought experiment that will illustrate the difference between a decision tree and a random forest model.

Suppose a bank has to approve a small loan amount for a customer and the bank needs to make a decision quickly. The bank checks the person's credit history and their financial condition and finds that they haven't re-paid the older loan yet. Hence, the bank rejects the application.

But here's the catch – the loan amount was very small for the bank's immense coffers and they could have easily approved it in a very low-risk move. Therefore, the bank lost the chance of making some money.

Now, another loan application comes in a few days down the line but this time the bank comes up with a different strategy – multiple decision-making processes. Sometimes it checks for credit history first, and sometimes it checks for customer's financial condition and loan amount first. Then, the bank combines results from these multiple decision-making processes and decides to give the loan to the customer.

Even if this process took more time than the previous one, the bank profited using this method. This is a classic example where collective decision making outperformed a single decision-making process. Now, here's my question to you – do you know what these two processes represent?

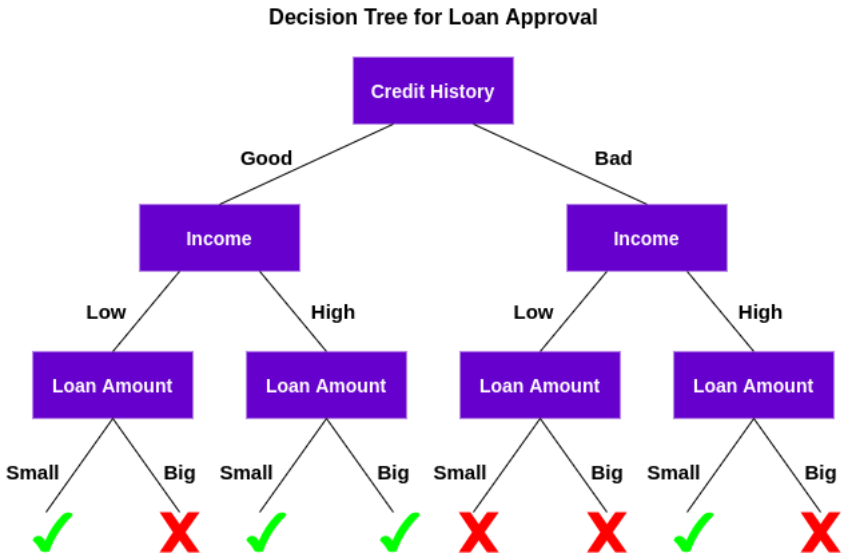


These are decision trees and a random forest! We'll explore this idea in detail here, dive into the major differences between these two methods, and answer the key question – which machine learning algorithm should you go with?

Table of Contents

Brief Introduction to Decision Trees

A decision tree is a supervised machine learning algorithm that can be used for both classification and regression problems. A decision tree is simply a series of sequential decisions made to reach a specific result. Here's an illustration of a decision tree in action (using our above example):



Let's understand how this tree works.

First, it checks if the customer has a good credit history. Based on that, it classifies the customer into two groups, i.e., customers with good credit history and customers with bad credit history. Then, it checks the income of the customer and again classifies him/her into two groups. Finally, it checks the loan amount requested by the customer. Based on the outcomes from checking these three features, the decision tree decides if the customer's loan should be approved or not.

The features/attributes and conditions can change based on the data and complexity of the problem but the overall idea remains the same. So, a decision tree makes a series of decisions based on a set of features/attributes present in the data, which in this case were credit history, income, and loan amount.

Now, you might be wondering:

> Why did the decision tree check the credit score first and not the income?

This is known as feature importance and the sequence of attributes to be checked is decided on the basis of criteria like **Gini Impurity Index** or **Information Gain.** The explanation of these concepts is outside the scope of our article here but you can refer to either of the below resources to learn all about decision trees:

- Tree-Based Algorithms: A Complete Tutorial from Scratch (in R & Python)
- Getting Started with Decision Trees (Free Course)

Note: The idea behind this article is to compare decision trees and random forests. Therefore, I will not go into the details of the basic concepts, but I will provide the relevant links in case you wish to explore further.

An Overview of Random Forest

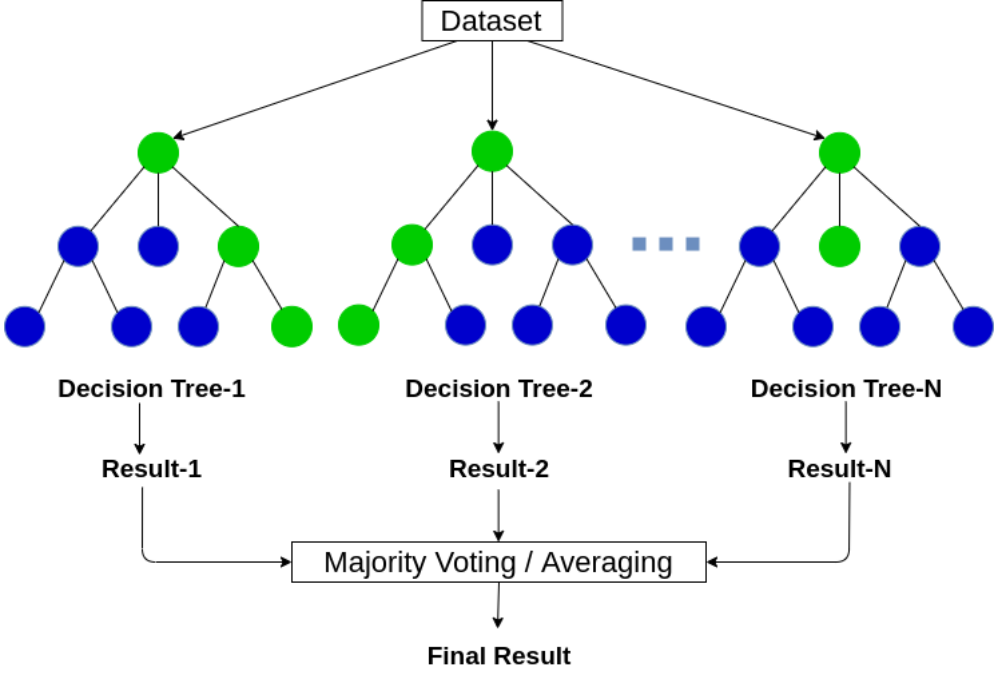
The decision tree algorithm is quite easy to understand and interpret. But often, a single tree is not sufficient for producing effective results. This is where the Random Forest algorithm comes into the picture.

Random Forest is a tree-based machine learning algorithm that leverages the power of multiple decision trees for making decisions. As the name suggests, it is a "forest" of trees!

But why do we call it a "random" forest? That's because it is a forest of **randomly created decision trees**. Each node in the decision tree works on a random subset of features to calculate the output. The random forest then combines the output of individual decision trees to generate the final output.

In simple words:

> The Random Forest Algorithm combines the output of multiple (randomly created) Decision Trees to generate the final output.



This process of combining the output of multiple individual models (also known as weak learners) is called **Ensemble Learning**. If you want to read more about how the random forest and other ensemble learning algorithms work, check out the following articles:

- Building a Random Forest from Scratch & Understanding Real-World Data Products
- A Beginner's Guide to Random Forest Hyperparameter Tuning
- A Comprehensive Guide to Ensemble Learning (with Python codes)
- How to build Ensemble Models in Machine Learning? (with code in R)

Now the question is, how can we decide which algorithm to choose between a decision tree and a random forest? Let's see them both in action before we make any conclusions!

Clash of Random Forest and Decision Tree (in Code!)

In this section, we will be using Python to solve a binary classification problem using both a decision tree as well as a random forest. We will then compare their results and see which one suited our problem the best.

We'll be working on the Loan Prediction dataset from Analytics Vidhya's DataHack platform. This is a binary classification problem where we have to determine if a person should be given a loan or not based on a certain set of features.

*Note: You can go to the **DataHack** platform and compete with other people in various online machine learning competitions and stand a chance to win exciting prizes.*

Ready to code?

Step 1: Loading the Libraries and Dataset

Let's start by importing the required Python libraries and our dataset:

```
1  import pandas as pd
```

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import f1_score
from sklearn.model_selection import train_test_split

# Importing dataset
df=pd.read_csv('dataset.csv')
df.head()
```

view raw

rfc_vs_dt-1.py hosted with ❤ by GitHub

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	C
0	LP001002	Male	No	0	Graduate	No	5849	0.0	NaN	360.0	
1	LP001003	Male	Yes	1	Graduate	No	4583	1508.0	128.0	360.0	
2	LP001005	Male	Yes	0	Graduate	Yes	3000	0.0	66.0	360.0	
3	LP001006	Male	Yes	0	Not Graduate	No	2583	2358.0	120.0	360.0	
4	LP001008	Male	No	0	Graduate	No	6000	0.0	141.0	360.0	

The dataset consists of 614 rows and 13 features, including credit history, marital status, loan amount, and gender. Here, the target variable is *Loan_Status*, which indicates whether a person should be given a loan or not.

Step 2: Data Preprocessing

Now, comes the most crucial part of any data science project – **data preprocessing** and **feature engineering**. In this section, I will be dealing with the categorical variables in the data and also imputing the missing values.

I will impute the missing values in the categorical variables with the mode, and for the continuous variables, with the mean (for the respective columns). Also, we will be label encoding the categorical values in the data. You can read this article for learning more about **Label Encoding**.

```
# Data Preprocessing and null values imputation
# Label Encoding
df['Gender']=df['Gender'].map({'Male':1,'Female':0})
df['Married']=df['Married'].map({'Yes':1,'No':0})
df['Education']=df['Education'].map({'Graduate':1,'Not Graduate':0})
df['Dependents'].replace('3+',3,inplace=True)
df['Self_Employed']=df['Self_Employed'].map({'Yes':1,'No':0})
df['Property_Area']=df['Property_Area'].map({'Semiurban':1,'Urban':2,'Rural':3})
df['Loan_Status']=df['Loan_Status'].map({'Y':1,'N':0})

#Null Value Imputation
rev_null=['Gender','Married','Dependents','Self_Employed','Credit_History','LoanAmount','Loan_Amount_Term']
df[rev_null]=df[rev_null].replace({np.nan:df['Gender'].mode(),
                                   np.nan:df['Married'].mode(),
                                   np.nan:df['Dependents'].mode(),
                                   np.nan:df['Self_Employed'].mode(),
                                   np.nan:df['Credit_History'].mode(),
                                   np.nan:df['LoanAmount'].mean(),
                                   np.nan:df['Loan_Amount_Term'].mean()})
```

view raw

rfc_vs_dt-2.py hosted with ❤ by GitHub

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_Hist
0	LP001002	1.0	0.0	0.0	1	0.0	5849	0.0	342.0	360.0	
1	LP001003	1.0	1.0	1.0	1	0.0	4583	1508.0	128.0	360.0	
2	LP001005	1.0	1.0	0.0	1	1.0	3000	0.0	66.0	360.0	
3	LP001006	1.0	1.0	0.0	0	0.0	2583	2358.0	120.0	360.0	
4	LP001008	1.0	0.0	0.0	1	0.0	6000	0.0	141.0	360.0	

Step 3: Creating Train and Test Sets

Now, let's split the dataset in an **80:20** ratio for training and test set respectively:

```
X=df.drop(columns=['Loan_ID','Loan_Status']).values
Y=df['Loan_Status'].values
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.2, random_state = 42)
```

view raw

rfc_vs_dt-3.py hosted with ❤ by GitHub

Let's take a look at the shape of the created train and test sets:

```
print('Shape of X_train=>',X_train.shape)
print('Shape of X_test=>',X_test.shape)
print('Shape of Y_train=>',Y_train.shape)
print('Shape of Y_test=>',Y_test.shape)
```

view raw

rfc_vs_dt-4.py hosted with ❤ by GitHub

```
Shape of X_train=> (491, 11)
Shape of X_test=> (123, 11)
Shape of Y_train=> (491,)
Shape of Y_test=> (123,)
```

Great! Now we are ready for the next stage where we'll build the decision tree and random forest models!

Step 4: Building and Evaluating the Model

Since we have both the training and testing sets, it's time to train our models and classify the loan applications. First, we will train a decision tree on this dataset:

```
# Building Decision Tree
from sklearn.tree import DecisionTreeClassifier
dt = DecisionTreeClassifier(criterion = 'entropy', random_state = 42)
dt.fit(X_train, Y_train)
dt_pred_train = dt.predict(X_train)
```

view raw

rfc_vs_dt-5.py hosted with ❤ by GitHub

Next, we will evaluate this model using F1-Score. F1-Score is the harmonic mean of precision and recall given by the formula:

$$F1 - score = \frac{2 * Precision * Recall}{Precision + Recall}$$

You can learn more about this and various other evaluation metrics here:

- 11 Important Model Evaluation Metrics for Machine Learning Everyone should know

Let's evaluate the performance of our model using the F1 score:

```
# Evaluation on Training set
dt_pred_train = dt.predict(X_train)
print('Training Set Evaluation F1-Score=>',f1_score(Y_train,dt_pred_train))
```

view raw

rfc_vs_dt-6.py hosted with ❤ by GitHub

```
Training Set Evaluation F1-Score=> 1.0
```

```
# Evaluating on Test set
dt_pred_test = dt.predict(X_test)
print('Testing Set Evaluation F1-Score=>',f1_score(Y_test,dt_pred_test))
```

view raw

rfc_vs_dt-7.py hosted with ❤ by GitHub

```
Testing Set Evaluation F1-Score=> 0.7953216374269005
```

Here, you can see that the decision tree performs well on in-sample evaluation, but its performance decreases drastically on out-of-sample evaluation. Why do you think that's the case? Unfortunately, our decision tree model is overfitting on the training data. Will random forest solve this issue?

Building a Random Forest Model

Let's see a random forest model in action:

```
# Building  Random Forest Classifier
from sklearn.ensemble import RandomForestClassifier
rfc = RandomForestClassifier(criterion = 'entropy', random_state = 42)
rfc.fit(X_train, Y_train)

# Evaluating on Training set
rfc_pred_train = rfc.predict(X_train)
print('Training Set Evaluation F1-Score=>',f1_score(Y_train,rfc_pred_train))
```

view raw

rfc_vs_dt-8.py hosted with ❤ by GitHub

```
Training Set Evaluation F1-Score=> 1.0
```

```
# Evaluating on Test set
rfc_pred_test = rfc.predict(X_test)
print('Testing Set Evaluation F1-Score=>',f1_score(Y_test,rfc_pred_test))
```

view raw

rfc_vs_dt-9.py hosted with ❤ by GitHub

```
Testing Set Evaluation F1-Score=> 0.8461538461538461
```

Here, we can clearly see that the random forest model performed much better than the decision tree in the out-of-sample evaluation. Let's discuss the reasons behind this in the next section.

Why Did Our Random Forest Model Outperform the Decision Tree?

Random forest leverages the power of multiple decision trees. It does *not* rely on the feature importance given by a single decision tree. Let's take a look at the feature importance given by different algorithms to different features:

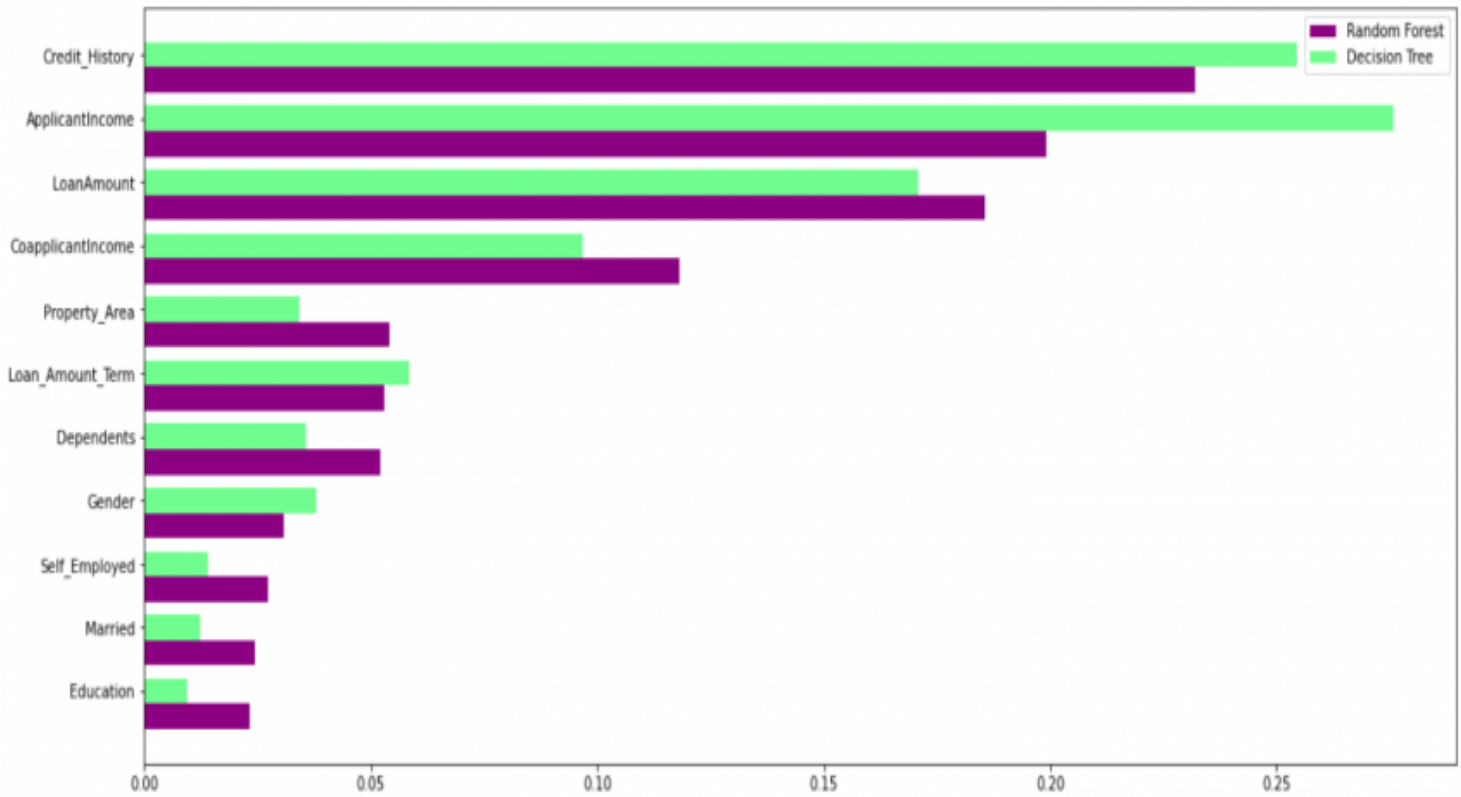
```
feature_importance=pd.DataFrame({
    'rfc':rfc.feature_importances_,
    'dt':dt.feature_importances_
},index=df.drop(columns=['Loan_ID','Loan_Status']).columns)
feature_importance.sort_values(by='rfc',ascending=True,inplace=True)

index = np.arange(len(feature_importance))
fig, ax = plt.subplots(figsize=(18,8))
rfc_feature=ax.barh(index,feature_importance['rfc'],0.4,color='purple',label='Random Forest')
dt_feature=ax.barh(index+0.4,feature_importance['dt'],0.4,color='lightgreen',label='Decision Tree')
ax.set(yticks=index+0.4,yticklabels=feature_importance.index)

ax.legend()
plt.show()
```

view raw

rfc_vs_dt-10.py hosted with ❤ by GitHub



As you can clearly see in the above graph, the decision tree model gives high importance to a particular set of features. But the random forest chooses features randomly during the training process. Therefore, it does not depend highly on any specific set of features. This is a special characteristic of random forest over bagging trees. You can read more about the bagging trees classifier here.

Therefore, the random forest can generalize over the data in a better way. This randomized feature selection makes random forest much more accurate than a decision tree.

So Which One Should You Choose – Decision Tree or Random Forest?

> Random Forest is suitable for situations when we have a large dataset, and interpretability is not a major concern.

Decision trees are much easier to interpret and understand. Since a random forest combines multiple decision trees, it becomes more difficult to interpret. Here's the good news – it's not impossible to interpret a random forest. Here is an article that talks about interpreting results from a random forest model:

- Decoding the Black Box: An Important Introduction to Interpretable Machine Learning Models in Python.

Also, Random Forest has a higher training time than a single decision tree. You should take this into consideration because as we increase the number of trees in a random forest, the time taken to train each of them also increases. That can often be crucial when you're working with a tight deadline in a machine learning project.

But I will say this – despite instability and dependency on a particular set of features, decision trees are really helpful because they are easier to interpret and faster to train. Anyone with very little knowledge of data science can also use decision trees to make quick data-driven decisions.

End Notes

That is essentially what you need to know in the decision tree vs. random forest debate. It can get tricky when you're new to machine learning but this article should have cleared up the differences and similarities for you.

You can reach out to me with your queries and thoughts in the comments section below.

Article Url - https://www.analyticsvidhya.com/blog/2020/05/decision-tree-vs-random-forest-algorithm/

Abhishek Sharma

He is a data science aficionado, who loves diving into data and generating insights from it. He is inspired by neural networks, learns and trains himself every day. He is always ready for making machines to learn through code and writing technical blogs.