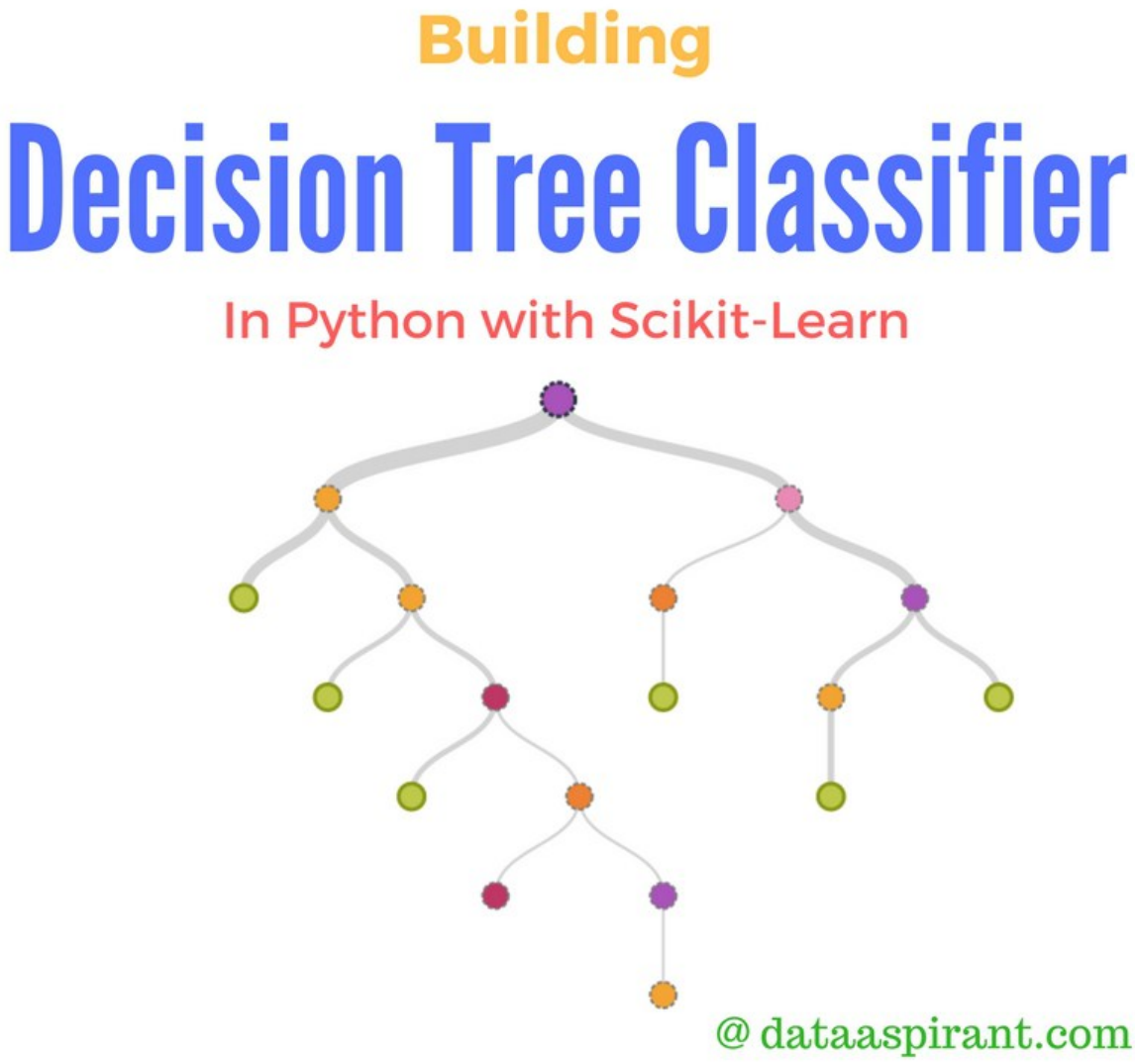
BUILDING DECISION TREE ALGORITHM IN PYTHON WITH SCIKIT LEARN

February 1, 2017 Rahul Saxena 13 Comments Data Science, Machine Learning, python



Decision tree algorithm in python

Decision Tree Algorithm implementation with scikit learn

One of the cutest and lovable supervised algorithms is Decision Tree Algorithm. It can be used for both the classification as well as regression purposes also.

As in the previous article how the decision tree algorithm works we have given the enough introduction to the working aspects of decision tree algorithm. In this article, we are going to build a decision tree classifier in python using scikit-learn machine learning packages for balance scale dataset.

The summarizing way of addressing this article is to explain how we can implement Decision Tree classifier on Balance scale data set. We will program our classifier in Python language and will use its **sklearn library**.

> How we can implement Decision Tree classifier in Python with Scikit-learn
>
> CLICK TO TWEET

Decision tree algorithm prerequisites

Before get start building the decision tree classifier in Python, please gain enough knowledge on how the decision tree algorithm works. If you don't have the basic understanding of how the Decision Tree algorithm. You can spend some time on **how the Decision Tree Algorithm works** article.

Once we completed modeling the Decision Tree classifier, we will use the trained model to predict whether the balance scale tip to the **right** or tip to the **left** or be **balanced**. The greatness of using Sklearn is that. It provides the functionality to implement machine learning algorithms in a few lines of code.

Before get started let's quickly look into the assumptions we make while creating the decision tree and the decision tree algorithm pseudocode.

Assumptions we make while using Decision tree

- In the beginning, the whole training set is considered at the root.

- Feature values are preferred to be categorical. If values are continuous then they are discretized prior to building the model.
- Records are distributed recursively on the basis of attribute values.
- Order to placing attributes as root or internal node of the tree is done by using some statistical approach.

Decision Tree Algorithm Pseudocode

1. Place the best attribute of our dataset at the root of the tree.
2. Split the training set into subsets. Subsets should be made in such a way that each subset contains data with the same value for an attribute.
3. Repeat step 1 and step 2 on each subset until you find leaf nodes in all the branches of the tree.

While building our decision tree classifier, we can improve its accuracy by tuning it with different parameters. But this tuning should be done carefully since by doing this our algorithm can overfit on our training data & ultimately it will build bad generalization model.

Sklearn Library Installation

Python's sklearn library holds tons of modules that help to build predictive models. It contains tools for data splitting, pre-processing, feature selection, tuning and supervised – unsupervised learning algorithms, etc. It is similar to Caret library in R programming.

For using it, we first need to install it. The best way to install data science libraries and its dependencies is by installing Anaconda package. You can also install only the most popular machine learning Python libraries.

Sklearn library provides us direct access to a different module for training our model with different machine learning algorithms like K-nearest neighbor classifier, Support vector machine classifier, decision tree, linear regression,

etc.

Balance Scale Data Set Description

Balance Scale data set consists of 5 attributes, 4 as feature attributes and 1 as the target attribute. We will try to build a classifier for predicting the Class attribute. The index of target attribute is 1st.

1.: 3 (L, B, R)
2. Left-Weight: 5 (1, 2, 3, 4, 5)
3. Left-Distance: 5 (1, 2, 3, 4, 5)
4. Right-Weight: 5 (1, 2, 3, 4, 5)
5. Right-Distance: 5 (1, 2, 3, 4, 5)

Index	Variable Name	Variable Values
1.	Class Name(Target Variable)	“R” : balance scale tip to the rig “L” : balance scale tip to the le “B” : balance scale be balance
2.	Left-Weight	1, 2, 3, 4, 5
3.	Left-Distance	1, 2, 3, 4, 5
4.	Right-Weight	1, 2, 3, 4, 5
5.	Right-Distance	1, 2, 3, 4, 5

The above table shows all the details of data.

Balance Scale Problem Statement

The problem we are going to address is To model a classifier for evaluating balance tip’s direction.

Decision Tree classifier implementation in Python with sklearn Library

The modeled Decision Tree will compare the new records metrics with the prior records(training data) that correctly classified the balance scale's tip direction.

Python packages used

- **NumPy**
  - NumPy is a Numeric Python module. It provides fast mathematical functions.
  - Numpy provides robust data structures for efficient computation of multi-dimensional arrays & matrices.
  - We used numpy to read data files into numpy arrays and data manipulation.
- **Pandas**
  - Provides DataFrame Object for data manipulation
  - Provides reading & writing data b/w different files.
  - DataFrames can hold different types data of multidimensional arrays.
- **Scikit-Learn**
  - It's a machine learning library. It includes various machine learning algorithms.
  - We are using its
    - train_test_split,
    - DecisionTreeClassifier,
    - accuracy_score algorithms.

> If you haven't setup the machine learning setup in your system the below posts will helpful.

Python Machine learning setup in ubuntu

Python machine learning virtual environment setup

Importing Python Machine Learning Libraries

This section involves importing all the libraries we are going to use. We are importing numpy and sklearn train_test_split, DecisionTreeClassifier & accuracy_score modules.

```
1 import numpy as np
2 import pandas as pd
3 from sklearn.cross_validation import train_test_split
4 from sklearn.tree import DecisionTreeClassifier
5 from sklearn.metrics import accuracy_score
6 from sklearn import tree
```

Numpy arrays and pandas dataframes will help us in manipulating data. As discussed above, sklearn is a machine learning library. The cross_validation's **train_test_split()** method will help us by splitting data into train & test set.

The tree module will be used to build a Decision Tree Classifier. Accutacy_score module will be used to calculate accuracy metrics from the predicted class variables.

Data Import

For importing the data and manipulating it, we are going to use pandas dataframes. First of all, we need to download the dataset. You can download the dataset from here. All the data values are separated by commas.

After downloading the data file, we will use Pandas **read_csv()** method to import data into pandas dataframe. Since our data is separated by commas "," and there is no header in our data, so we will put header parameter's value **"None"** and sep parameter's value as ",".

```
balance_data = pd.read_csv(
'https://archive.ics.uci.edu/ml/machine-learning-databases/balance-scale/balar
                           sep= ',', header= None)
```

We are saving our data into “balance_data” dataframe.

For checking the length & dimensions of our dataframe, we can use len() method & “.shape”.

```
print "Dataset Lenght:: ", len(balance_data)
print "Dataset Shape:: ", balance_data.shape
```

Output:

```
Dataset Lenght::  625
Dataset Shape::  (625, 5)
```

We can print head .e, top 5 lines of our dataframe using **head()** method.

```
print "Dataset:: "
balance_data.head()
```

```
Dataset::
```

Output:

	0	1	2	3	4
0	B	1	1	1	1
1	R	1	1	1	2
2	R	1	1	1	3
3	R	1	1	1	4
4	R	1	1	1	5

Data Slicing

Data slicing is a step to split data into train and test set. Training data set can be used specifically for our model building. Test dataset should not be mixed up while building model. Even during standardization, we should not standardize our test set.


```
1 X = balance_data.values[:, 1:5]
2 Y = balance_data.values[:,0]
```


The above snippet divides data into feature set & target set. The "X " set consists of predictor variables. It consists of data from 2nd column to 5th column. The "Y" set consists of the outcome variable. It consists of data in the 1st column. We are using ".values" of numpy converting our dataframes into numpy arrays.

Let's split our data into training and test set. We will use sklearn's train_test_split() method.

```
1 X_train, X_test, y_train, y_test = train_test_split( X, Y, test_size = 0.3, r
```

The above snippet will split data into training and test set. **X_train**, **y_train** are training data & **X_test**, **y_test** belongs to the test dataset.

The parameter test_size is given value **0.3**; it means test sets will be **30%** of whole dataset & training dataset's size will be **70%** of the entire dataset. random_state variable is a pseudo-random number generator state used for random sampling. If you want to replicate our results, then use the same value of random_state.

Decision Tree Training

Now we fit Decision tree algorithm on training data, predicting labels for validation dataset and printing the accuracy of the model using various parameters.

DecisionTreeClassifier(): This is the classifier function for DecisionTree. It is the main function for implementing the algorithms. Some important

parameters are:

- **criterion:** It defines the function to measure the quality of a split. Sklearn supports “gini” criteria for Gini Index & “entropy” for Information Gain. By default, it takes “gini” value.
- **splitter:** It defines the strategy to choose the split at each node. Supports “best” value to choose the best split & “random” to choose the best random split. By default, it takes “best” value.
- **max_features:** It defines the no. of features to consider when looking for the best split. We can input integer, float, string & None value.
    - If an integer is inputted then it considers that value as max features at each split.
    - If float value is taken then it shows the percentage of features at each split.
    - If “auto” or “sqrt” is taken then max_features=sqrt(n_features).
    - If “log2” is taken then max_features= log2(n_features).
    - If None, then max_features=n_features. By default, it takes “None” value.
- **max_depth:** The max_depth parameter denotes maximum depth of the tree. It can take any integer value or None. If None, then nodes are expanded until all leaves are pure or until all leaves contain less than min_samples_split samples. By default, it takes “None” value.
- **min_samples_split:** This tells above the minimum no. of samples reqd. to split an internal node. If an integer value is taken then consider min_samples_split as the minimum no. If float, then it shows percentage. By default, it takes “2” value.
- **min_samples_leaf:** The minimum number of samples required to be at a leaf node. If an integer value is taken then consider min_samples_leaf as the minimum no. If float, then it shows percentage. By default, it takes “1” value.
- **max_leaf_nodes:** It defines the maximum number of possible leaf nodes. If None then it takes an unlimited number of leaf nodes. By

default, it takes “None” value.

- **min_impurity_split:** It defines the threshold for early stopping tree growth. A node will split if its impurity is above the threshold otherwise it is a leaf.

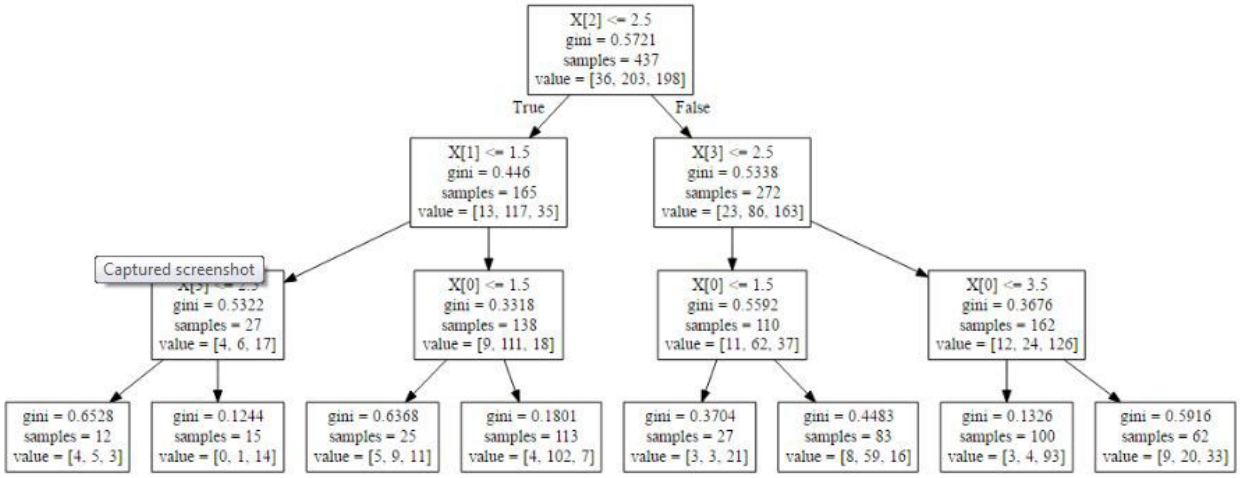
Let’s build classifiers using criterion as gini index & information gain. We need to fit our classifier using fit(). We will plot our decision tree classifier’s visualization too.

Decision Tree Classifier with criterion gini index

```
clf_gini = DecisionTreeClassifier(criterion = "gini", random_state = 100,
                                  max_depth=3, min_samples_leaf=5)
clf_gini.fit(X_train, y_train)
```

Output:

```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=3,
            max_features=None, max_leaf_nodes=None, min_samples_leaf=5,
            min_samples_split=2, min_weight_fraction_leaf=0.0,
            presort=False, random_state=100, splitter='best')
```

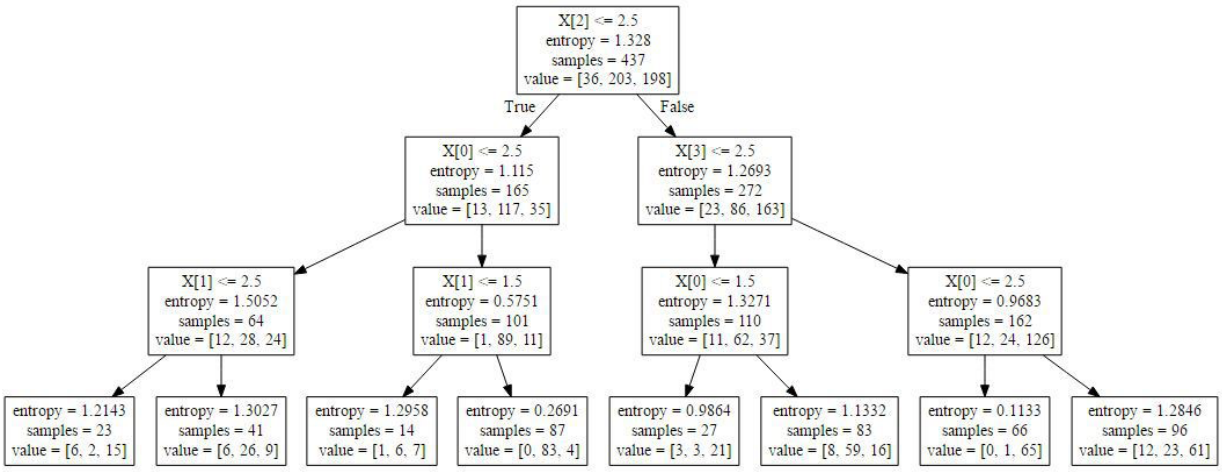


Decision Tree Classifier with criterion information gain

```
clf_entropy = DecisionTreeClassifier(criterion = "entropy", random_state = 100
 max_depth=3, min_samples_leaf=5)
clf_entropy.fit(X_train, y_train)
```

Output

```
DecisionTreeClassifier(class_weight=None, criterion='entropy', max_depth=3,
            max_features=None, max_leaf_nodes=None, min_samples_leaf=5,
            min_samples_split=2, min_weight_fraction_leaf=0.0,
            presort=False, random_state=100, splitter='best')
```



Prediction

Now, we have modeled 2 classifiers. One classifier with gini index & another one with information gain as the criterion. We are ready to predict classes for our test set. We can use predict() method. Let's try to predict target variable for test set's 1st record.

```
clf_gini.predict([[4, 4, 3, 3]])
```

Output

```
array(['R'], dtype=object)
```

This way we can predict class for a single record. It's time to predict target variable for the whole test dataset.

Prediction for Decision Tree classifier with criterion as gini index

```
y_pred = clf_gini.predict(X_test)
```

```
y_pred = clf_gini.predict(X_test)
y_pred
```

Output

```
array(['R', 'L', 'R', 'R', 'R', 'L', 'R', 'L', 'L', 'L', 'R', 'L', 'L',
       'L', 'R', 'L', 'R', 'L', 'L', 'R', 'L', 'R', 'L', 'L', 'R', 'L',
       'L', 'L', 'R', 'L', 'L', 'L', 'R', 'L', 'L', 'L', 'L', 'R', 'L',
       'L', 'R', 'L', 'R', 'L', 'R', 'R', 'L', 'L', 'R', 'L', 'R', 'R',
       'L', 'R', 'R', 'L', 'R', 'R', 'L', 'L', 'R', 'R', 'L', 'L', 'L',
       'L', 'L', 'R', 'R', 'L', 'L', 'R', 'R', 'L', 'R', 'L', 'R', 'R',
       'R', 'L', 'R', 'L', 'L', 'L', 'L', 'R', 'R', 'L', 'R', 'L', 'R',
       'R', 'L', 'L', 'L', 'R', 'R', 'L', 'L', 'L', 'R', 'L', 'R', 'R',
       'R', 'R', 'R', 'R', 'R', 'L', 'R', 'L', 'R', 'R', 'L', 'R', 'R',
       'R', 'R', 'R', 'L', 'R', 'L', 'L', 'L', 'L', 'L', 'L', 'L', 'R',
       'R', 'R', 'R', 'L', 'R', 'R', 'R', 'L', 'L', 'R', 'L', 'R', 'L',
       'R', 'L', 'L', 'R', 'L', 'L', 'R', 'L', 'R', 'L', 'R', 'R', 'R',
       'L', 'R', 'R', 'R', 'R', 'R', 'L', 'L', 'R', 'R', 'R', 'R', 'L',
       'R', 'R', 'R', 'L', 'R', 'L', 'L', 'L', 'L', 'R', 'R', 'L', 'R',
       'R', 'L', 'L', 'R', 'R', 'R'], dtype=object)
```

Prediction for Decision Tree classifier with criterion as information gain

```
y_pred_en = clf_entropy.predict(X_test)
y_pred_en
```

Output

```
array(['R', 'L', 'R', 'L', 'R', 'L', 'R', 'L', 'R', 'R', 'R', 'R', 'L',
       'L', 'R', 'L', 'R', 'L', 'L', 'R', 'L', 'R', 'L', 'L', 'R', 'L',
       'R', 'L', 'R', 'L', 'R', 'L', 'L', 'L', 'L', 'L', 'R',
       'L', 'R', 'L', 'R', 'L', 'R', 'R', 'L', 'L', 'R', 'L', 'L', 'R',
       'L', 'L', 'R', 'L', 'R', 'R', 'L', 'R', 'R', 'R', 'L', 'L', 'R',
       'L', 'L', 'R', 'L', 'L', 'L', 'R', 'R', 'L', 'R', 'L', 'R', 'R',
       'R', 'L', 'R', 'L', 'L', 'L', 'L', 'R', 'R', 'L', 'R', 'L', 'R',
       'R', 'L', 'L', 'L', 'R', 'R', 'L', 'L', 'L', 'R', 'L', 'L', 'R',
       'R', 'R', 'R', 'R', 'R', 'L', 'R', 'L', 'R', 'R', 'L', 'R', 'R',
       'L', 'R', 'R', 'L', 'R', 'R', 'R', 'L', 'L', 'L', 'L', 'L', 'R',
       'R', 'R', 'R', 'L', 'R', 'R', 'R', 'L', 'L', 'R', 'L', 'R', 'L',
       'R', 'L', 'R', 'R', 'L', 'L', 'R', 'L', 'R', 'R', 'R', 'R', 'R',
       'L', 'R', 'R', 'R', 'R', 'R', 'R', 'L', 'R', 'L', 'R', 'R', 'L',
       'R', 'L', 'R', 'L', 'R', 'L', 'L', 'L', 'L', 'L', 'R', 'R', 'R',
       'L', 'L', 'L', 'R', 'R', 'R'], dtype=object)
```

Calculating Accuracy Score

The function **accuracy_score()** will be used to print accuracy of Decision Tree

algorithm. By accuracy, we mean the ratio of the correctly predicted data points to all the predicted data points. Accuracy as a metric helps to understand the effectiveness of our algorithm. It takes 4 parameters.

- y_true,
- y_pred,
- normalize,
- sample_weight.

Out of these 4, normalize & sample_weight are optional parameters. The parameter y_true accepts an array of correct labels and y_pred takes an array of predicted labels that are returned by the classifier. It returns accuracy as a float value.

Accuracy for Decision Tree classifier with criterion as gini index

```
1 print "Accuracy is ", accuracy_score(y_test,y_pred)*100
```

Output

```
1 Accuracy is  73.4042553191
```

Accuracy for Decision Tree classifier with criterion as information gain

```
1 print "Accuracy is ", accuracy_score(y_test,y_pred_en)*100
```

Output

```
1 Accuracy is  70.7446808511
```

Conclusion

In this article, we have learned how to model the decision tree algorithm in Python using the Python machine learning library scikit-learn. In the process, we learned how to split the data into train and test dataset. To model decision tree classifier we used the information gain, and gini index split criteria. In the

end, we calucalte the accuracy of these two decision tree models.

Follow us:

FACEBOOK| QUORA |TWITTER| GOOGLE+ | LINKEDIN| REDDIT | FLIPBOARD | MEDIUM | G

I hope you like this post. If you have any questions, then feel free to comment below. If you want me to write on one particular topic, then do tell it to me in the comments below.

Related Courses:

Do check out unlimited data science courses

Title of the course	Course Link	Course Lir
Machine Learning: Classification	Machine Learning: Classification	• Will le intro classi • This c intro popu classi algor • You v to im pytho pytho learn

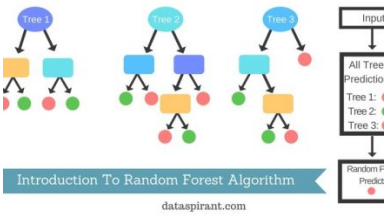
Data Mining with Python: Classification and Regression	Data Mining with Python: Classification and Regression	• Unde conce minir how t conce real v • Will g expe pytho langu • Hand with matp (Pyth
Machine learning with Scikit-learn	Machine learning with Scikit-learn	• Load learn mach algor unsu supe • Asses accur perfo • Being what' mode scena

Share this:

Related

visualize decision tree in python with graphviz
April 21, 2017
In "Machine Learning"

How to save Scikit Learn models with Python Pickle library
February 13, 2017
In "python"



How the random forest algorithm works in machine learning
May 22, 2017
In "Machine Learning"

classification algorithms Decision tree python