Understanding Decision Trees with Python

Author, Mayank Tripathi

A Data Science Foundation White Paper

May 2020

---

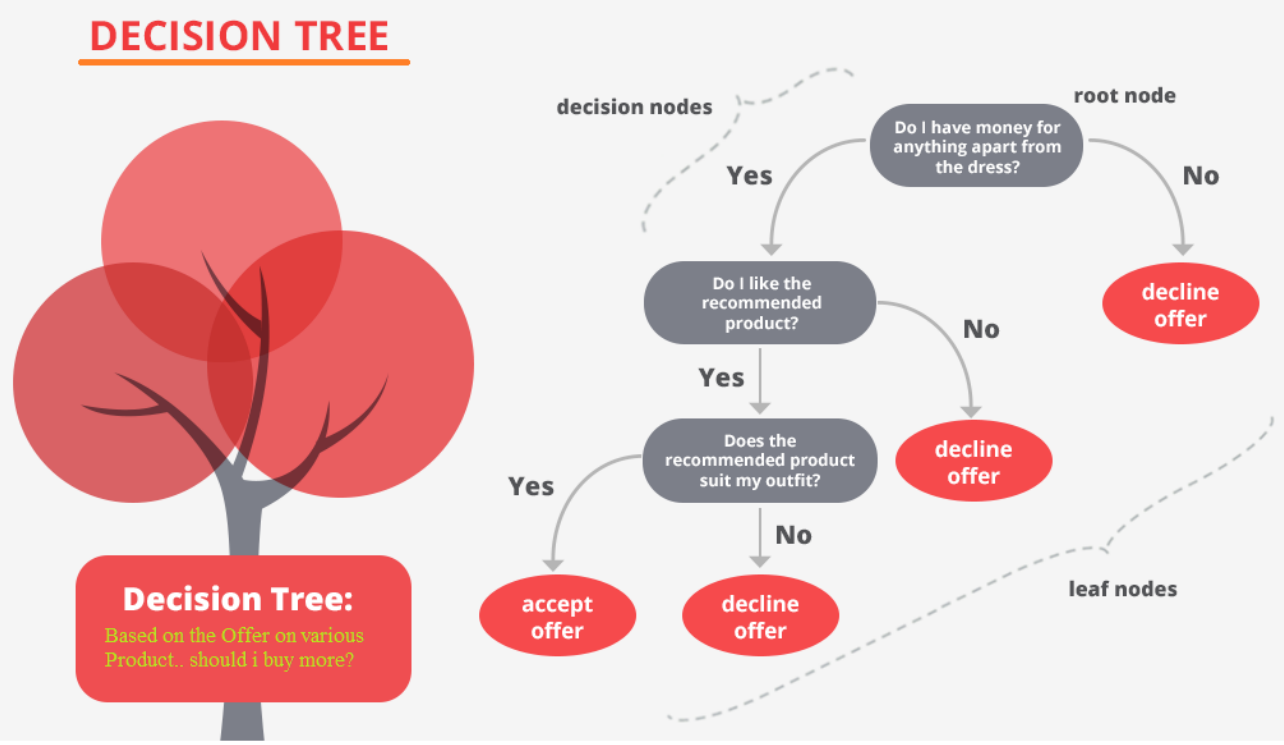
www.datascience.foundation

Data Science Foundation
Data Science Foundation, Atlantic Business Centre, Atlantic Street, Altrincham, WA14 5NQ
Tel: 0161 926 3641 Email: admin@datascience.foundation Web: www.datascience.foundation
Registered in England and Wales 4th June 2015, Registered Number 9624670

Decision Trees, the popular and time-tested method of applying logic to complex problems, where the variables are many and the options specific and dependent, have an important role to play within Machine Learning.

We will dedicate this paper to understanding why this reasonably humble technique has become such an important tool for data scientists. And we will start the debate by suggesting that Decision Trees are popular because they have two key properties:

1. Simplicity: Decision Trees are simple, visually appealing and are easy to interpret.
2. Accuracy: Advance Decision Tree models show exceptional performance in predicting patterns in complex data.



Definition

As per Wikipedia, A decision tree is a decision support tool that uses a tree-like model of decisions and their possible consequences, including chance event outcomes, resource costs, and utility. It is one way to display an algorithm that only contains conditional control

statements. https://en.wikipedia.org/wiki/Decision_tree

Decision trees are commonly used in operations research, specifically in decision analysis, to help identify a strategy most likely to reach a goal but are also a popular tool in machine learning. Generally, a decision tree is drawn upside down with its root at the top (recommended) and it is known as Top-Down Approach. But it can also be drawn from Left - To - Right as well.

A decision tree consists of three types of nodes

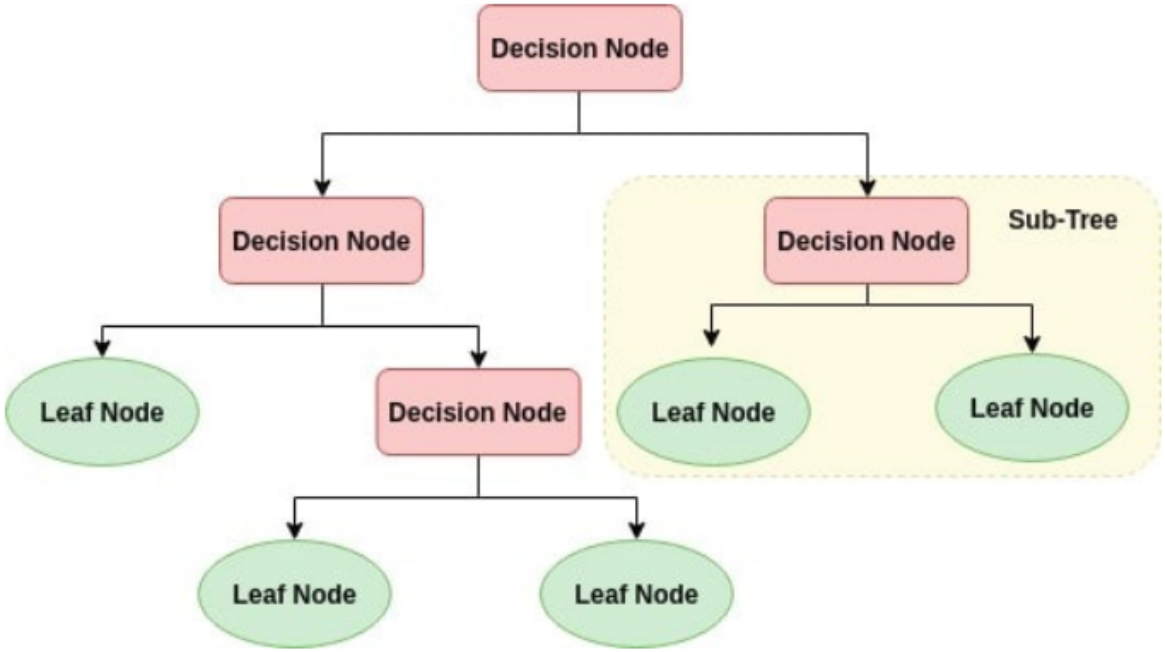
- Root Nodes
- Decision Nodes
- Leaf Nodes

Each node in the tree acts as a test case for some attribute and each edge descending from that node corresponds to one of the possible answers to the test case.

This process is recursive and is repeated for every subtree rooted at the new nodes.

Refer to below image for more visual representation.

1. Root Node – the very top node is called as Root Node or just a Node. Alternatively, it is also called as Top Decision Node. It represents the entire population or sample, and this further gets pided into two or more homogeneous sets.
2. Decision Node – When a sub-node splits into further sub-nodes, then it is called a decision node. These are also called as Internal nodes, or at-times just a Node(s). Internal nodes have arrows pointing to them, and they have arrows pointing away from them.
3. Leaf / Terminal Node – Nodes with no children (no further split) is called Leaf or Terminal node or just leaves. Leaf noes have arrows pointing to them, but there are no arrows pointing away from them.



Now that we know about nodes, let's have a look at other related terminology

4. Splitting – It is a process of piding a node into two or more sub-nodes. See the line with arrows in the above image.
5. Pruning – When we reduce the size of decision trees by removing nodes (opposite of Splitting), the process is called pruning. We will see some examples of this later.
6. Branch / Sub-Tree – A sub section of the decision tree is called branch or sub-tree.
7. Parent and Child Node – A node, which is pided into sub-nodes is called a parent node, whereas sub-nodes are the child of a parent node. Don't get confuse, this is same as Root Node and Decision Node. We are defining it with Parent and Chile Node.

Types

In Machine Learning, we have two types of Model, these are Regression and Classification. With Decision Trees we have similar models. We can say that Decision Trees can be applied to both Regression and Classification Problems.

Regression Tree

Regression Trees are used for continuous quantitative target variables.
Example: Predicting rainfall; Predicting revenue; Predicting marks etc.

Classification Tree

Classification Tree are used for discrete categorical target variables.

Example: Predicting if the temperature will be High or Low; Predicting if a team will Win the match or not; Predicting the health of a person, is that Person Healthy or Unhealthy. etc.

How does the Decision Tree algorithm work?

Generally we know they work in a stepwise manner, that is step-by-step rather than a continuous process, and have a tree structure where we split a node using a feature based on some criterion.

But how do these features get selected and how does a threshold or value get chosen for a feature?

There are many splitting criteria used in Decision trees I will not be going into those theoretical parts in this paper, as this would require a lot of information on theory. The 3 main splitting criteria used in Decision trees are

1. **Gini Impurity –** As per Wikipedia, Gini impurity is a measure of how often a randomly chosen element from the set would be incorrectly labeled if it was randomly labeled according to the distribution of labels in the subset.

   In simple terms, Gini impurity is the measure of impurity in a node. Its formula is:

$$I_G(n) = 1 - \sum_{i=1}^{J} (p_i)^2$$

2. **Entropy –**Another very popular way to split nodes in the decision tree is Entropy. Entropy is the measure of Randomness in the system. The formula for Entropy is:

$$Entropy = \sum_{i=1}^{C} -p_i * \log_2(p_i)$$

3. **Variance –**– Gini Impurity and Entropy work well for the classification scenario. But what about regression?

   In the case of regression, the most common split measure used is just the weighted variance of the nodes. It makes sense too: We want minimum variation in the nodes after the split.

$$\text{Variance} = \frac{\Sigma(X - \overline{X})^2}{n}$$

4. **Information Gain –** Information gain or IG is a statistical property that measures how well a given attribute separates the training examples according to their target classification.

The basic idea behind any decision tree algorithm is as follows:

1. Select the best attribute using Attribute Selection Measures (one of the above splitting criteria) to split the records.
2. Make that attribute a decision node and break the dataset into smaller subsets.
3. Start tree building by repeating this process recursively for each child until there are no more remaining attributes.

If you have read my other articles on Python, then you would have noticed that I have provided less theory but more practical examples. If you would like more details on the **behind Decision Trees,** add a comment, and I will share the details. Including the theory part here will make this article boring. As these topics are big in themselves and cannot be covered in

one go.

Requirements

Before you proceed, please make sure you have Python installed on your workstation.

And open your favorite Editor / tool, or you can use Python Jupyter / Google Colab which is free to use and doesn't require installation. If not sure please check https://datascience.foundation/datatalk/setting-up-a-python-jupyter-notebook-online-working-with-python-on-the-cloud

Hands-On

Let us apply Decision Tree Algorithms in Python using below practical examples provided below using a real-world dataset.

Note: The intention here is to understand Decision Trees, so I will not spend time on data cleaning or accuracy score.

These examples will incorporate the following steps: (steps may vary from person to person or example to example).

Step 1: Gather the data / dataset

Step 2: Import the required Python packages (as we are using Python here)

Step 3: Build a data frame

Step 4: Create the Model in Python (In this example Decision Tree)

Step 5: Predict using Test Dataset and Check the score

Step 6: Prediction with a New Set of Data / unseen data (if required)

Practical 1 - Classification

This is a simple hands-on example.

We will use the scikit-learn library to build the decision tree model. We will be using the **iris dataset** to build a decision tree classifier. The data set contains information of 3 classes of the

iris plant with the following attributes: - sepal length - sepal width - petal length - petal width.

class: Iris Setosa, Iris Versicolour, Iris Virginica

The task is to predict the class of the iris plant based on the attributes.

The scikit-learn dataset library already has the iris dataset. You can either use the dataset from the source or import it from the scikit-learn dataset library.

Note : This dataset can also be downloaded from https://archive.ics.uci.edu/ml/datasets/iris
As now we have a Dataset, we will start importing the basic library, as below.

```
#Importing required libraries
import pandas as pd
import numpy as np
from sklearn.datasets import load_iris
```

As mentioned above we will be using the iris dataset from sklearn, and its very easy to load the data.

```
#Loading the iris data
data = load_iris()
```

There are 150 examples / samples in the dataset, it contains columns as shown below.

```
[31] data.data.shape

     (150, 4)
```

There are three classes of iris plants: 'setosa', 'versicolor' and 'virginica'. Now, we have

imported the iris data in the variable 'data'. We will now extract the attribute data and the corresponding labels. We can extract the attributes and labels by calling .data and .target as shown below:

```
print('Classes to predict: ', data.target_names)

Classes to predict:  ['setosa' 'versicolor' 'virginica']
```

```
[33] print('Features: ', data.feature_names)

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

Extracting the Attributes and Target from data.

```
[36] # Extracting data attributes / features
     X = data.data

     # Extracting target/ class labels
     y = data.target
```

```
[37] #Shape of the dataset
     display(X.shape, y.shape)

     (150, 4)
     (150,)
```

Now that we have extracted the data attributes and corresponding labels, we will split them to train and test datasets from the variable X and y.

For this purpose, we will use the **scikit-learn's model_selection** library, and **'train_test_split'** function, which takes in the attributes and labels as inputs and produces the train and test sets as below.

```
# Import library for splitting the dataset into train and test.
from sklearn.model_selection import train_test_split
```

```
[39] #Using the train_test_split to create train and test sets.
     X_train, X_test, y_train, y_test = train_test_split(X, y, random_state = 50, test_size = 0.25)
```

train_test_split function also optionally takes **random_state**, which we can set it to any random value, here I have taken it as 50. You can have it as 10, 20, 30, 100, 3, 5, etc… any random value should work fine.

Also **test_size** I have used 0.25 which indicates that we want to split the test data as 25% of total dataset and remaining 75% will assign as Train data.

Since, this is a classification problem, we will import the **DecisionTreeClassifier** function from the sklearn library.

When we initiate the Decision Tree Classifier, by-default the criterion parameters is set to 'gini', and there are various other parameters as well, which we can go with the default or change it if required. Will see what impact of it when change one or two parameters.

First will proceed with the default criterion as 'gini'.

```
clf = DecisionTreeClassifier()
clf.fit(X_train,y_train)
```

```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                       max_depth=None, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=None, splitter='best')
```

Next, we will fit the classifier on the train attributes and labels using **fit** function or method.

Now, we will use the trained classifier / model to predict the labels of the test attributes.

```
#prediction
y_pred = clf.predict(X_test)
```

We will now evaluate the predicted classes using some metrics. For this case, I will be using **'accuracy_score'** to calculate the accuracy of the predicted labels.

```
[19] print('Accuracy Score on train data (using Default criterionas gini): ', accuracy_score(y_true=y_train, y_pred=clf.predict(X_train)))
     print('Accuracy Score on test data (using Default criterionas gini): ', accuracy_score(y_true=y_test, y_pred=y_pred))

Accuracy Score on train data (using Default criterionas gini):  1.0
Accuracy Score on test data (using Default criterionas gini):  0.9473684210526315
```

From the score it seems using Gini as Criterion we found the Accuracy on Train Data as 100 which is represented as 1.0.

And on Test Data it is ~94% which is represented by ~ 0.94

Next, we will set the 'criterion' to 'entropy', which sets the measure for splitting the attribute to information gain.

Now let's change the Criterion from default Gini to Entropy. Which is very simple and straight forward, we just need to set a parameter criterion as entropy. Make a note the entropy is in lower case, even gini was in lower case.

```
[47] # Changing the Criterion to Entropy
     clf_entropy = DecisionTreeClassifier(criterion = 'entropy')
```

```
#Training the decision tree classifier.
clf_entropy.fit(X_train, y_train)

DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='entropy',
                       max_depth=None, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=None, splitter='best')
```

Check the output of fit method, from here we can see now that the criterion is changed to entropy.

Now, we will use the trained classifier / model with entropy to predict the labels of the test attributes.

```
[49] #Predicting labels on the test set.
     y_pred_entropy =  clf_entropy.predict(X_test)
```

Will use the same evaluation method to evaluate the predicted classes. For this case, I will be using **'accuracy_score'** to calculate the accuracy of the predicted labels.

```
[50] print('Accuracy Score on train data (using Default criterion as Entropy): ', accuracy_score(y_true=y_train, y_pred =clf_entropy.predict(X_train)))
     print('Accuracy Score on test data (using Default criterion as Entropy): ', accuracy_score(y_true=y_test, y_pred =y_pred_entropy))

Accuracy Score on train data (using Default criterion as Entropy):  1.0
Accuracy Score on test data (using Default criterion as Entropy):  0.9473684210526315
```

From the score it seems using Entropy as Criterion we got the same Accuracy as we received with Gini.

On Train Data it is 100 which is represented as 1.0.

And on Test Data it is ~94% which is represented by ~ 0.94

Next, we will tune the parameters of the decision tree to increase its accuracy.

One of those parameters is **'min_samples_split',** which is the minimum number of samples required to split an internal node. Its default value is equal to 2 because we cannot split a node containing only one example / sample.

Per the documentation from scikit-learn.org, the parameter **'min_samples_split'**

min_samples_split : int or float, default=2
The minimum number of samples required to split an internal node:
If int, then consider min_samples_split as the minimum number.
If float, then min_samples_split is a fraction and ceil (min_samples_split * n_samples) are the minimum number of samples for each split.

Data Science Foundation, Atlantic Business Centre, Atlantic Street, Altrincham, WA14 5NQ
Tel: 0161 926 3641 Email: admin@datascience.foundation Web: www.datascience.foundation
Registered in England and Wales 4th June 2015, Registered Number 9624670

```
[56] # After tune the parameters of the decision tree to increase its accuracy
     clf2 = DecisionTreeClassifier(criterion='entropy', min_samples_split=50)
     clf2.fit(X_train, y_train)
```

```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='entropy',
                       max_depth=None, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=50,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=None, splitter='best')
```

From above we can see that now the min_samples_split was set to 50, and now let's see the Accuracy Score with a new parameter.

```
[57] print('Accuracy Score on train data (using Default criterion as Entropy & min_samples_split): ', accuracy_score(y_true=y_train, y_pred=clf2.predict(X_train)))
     print('Accuracy Score on the test data (using Default criterion as Entropy & min_samples_split): ', accuracy_score(y_true=y_test, y_pred=clf2.predict(X_test)))
```

```
Accuracy Score on train data (using Default criterion as Entropy & min_samples_split):  0.9642857142857143
Accuracy Score on the test data (using Default criterion as Entropy & min_samples_split):  0.9473684210526315
```

We can see that the accuracy on the test set remained same, while it decreased on the training set from 1 to ~0.96. The score will not necessarily remain the same, it may increase or decrease.

This is because increasing the value of the min_sample_split smooths the decision boundary and thus prevents it from overfitting.

You may tune other parameters of the decision tree and check how they affect the decision boundary in a similar way.

Now we have seen the Model; Predicted the values, and evaluated the model, but where is the Decision Tree. Is there a way we can visualize the Tree showing which Model has been used?

The answer is yes. We can.
To visualize the Decision Tree, one has to install the package called "pydotplus".

```
[54] !pip install pydotplus
```

```
Requirement already satisfied: pydotplus in /usr/local/lib/python3.6/dist-packages (2.0.2)
Requirement already satisfied: pyparsing>=2.0.1 in /usr/local/lib/python3.6/dist-packages (from pydotplus) (2.4.7)
```

Note: If you are using Google Colab, you will not be required to install it, as this package is

pre-installed. But still if you get an error, then please install it. For me it is already there, so it says Requirement already satisfied.

There are few packages which we need to import these are:

```
from sklearn.tree import export_graphviz
from sklearn.externals.six import StringIO
from IPython.display import Image
import pydotplus
```

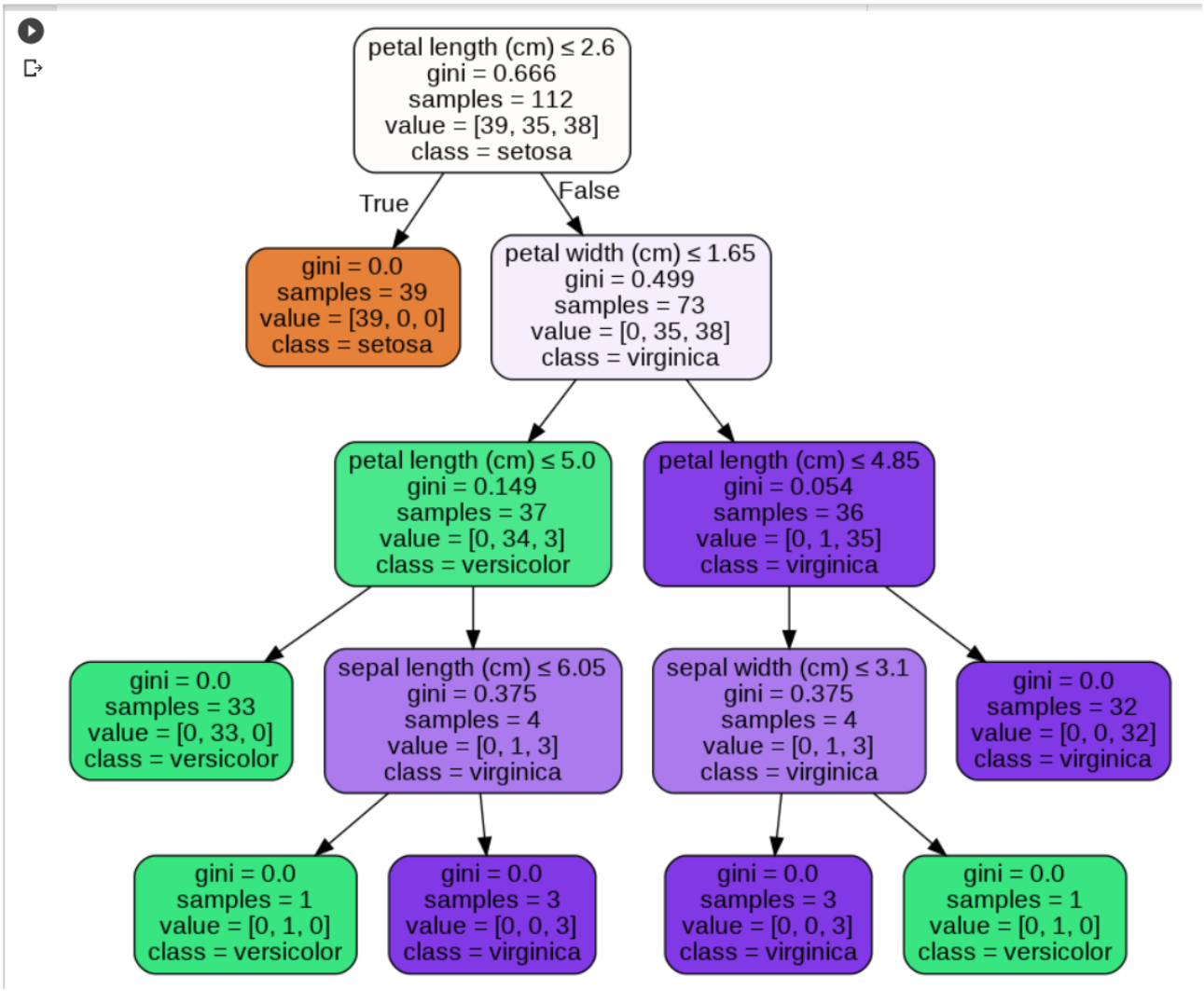
and the code to visualize the Tree.

```
dot_data = StringIO()
export_graphviz(clf, out_file=dot_data,
                filled=True, rounded=True,
                special_characters=True,feature_names  = data.feature_names,class_names=data.target_names)
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
Image(graph.create_png())
```

```
[55] from sklearn.tree import export_graphviz
     from sklearn.externals.six import StringIO
     from IPython.display import Image
     import pydotplus

     dot_data = StringIO()
     export_graphviz(clf, out_file=dot_data,
                     filled=True, rounded=True,
                     special_characters=True,feature_names = data.feature_names,class_names=data.target_names)
     graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
     Image(graph.create_png())
```

And there comes the Decision Tree…
Note: Here I have used the default classifier which is clf, thus in the tree below we see the gini values.



Practical 2 - Regression

Decision trees can also be applied to regression problems, using the **DecisionTreeRegressor class.**

As in the classification setting, the fit method will take argument arrays X and y, only in this case y is expected to have floating point values instead of integer values.

Similar to what we did for **DecisionTreeClassifier,** in same way we will be using **DecisionTreeRegressor** method.

```
[60] from sklearn import tree

[61] X = [[0, 0], [2, 2]]
     y = [0.5, 2.5]

[62] clf = tree.DecisionTreeRegressor()

[63] clf = clf.fit(X, y)

[64] clf.predict([[1, 1]])

     array([0.5])
```

This is the end of this paper on Decision Trees, but please do not stop here. Please go and do more hands-on by changing various parameters in the Classifier; try to understand the math behind the tree and work with other datasets.

If you are interested you can obtain various datasets from https://archive.ics.uci.edu/ml/datasets.php this is a good site and has various domain related datasets for free. So what are you waiting for?

Code for reference

You can also download the code or refer it from https://colab.research.google.com/drive/1pBCYRM8IJEAd1d7FC9vsWuBV7qS4vjl1?usp=sharing

Thank You!
Happy Learning :)

References

- https://en.wikipedia.org/wiki/Decision_tree
- https://en.wikipedia.org/wiki/Decision_tree_learning

- https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html
- https://scikit-learn.org/stable/modules/generated/sklearn.tree.export_graphviz.html
- https://scikit-learn.org/stable/modules/tree.html#tree
- https://archive.ics.uci.edu/ml/datasets.php

Background Reading

For further reading on Regression using Python, please refer to these papers: https://datascience.foundation/sciencewhitepaper/understanding-logistic-regression-with-python-practical-guide-1

And

https://datascience.foundation/sciencewhitepaper/understanding-linear-regression-with-python-practical-guide-2

About the Data Science Foundation

The Data Science Foundation is a professional body representing the interests of the Data Science Industry. Its membership consists of suppliers who offer a range of big data analytical and technical services and companies and individuals with an interest in the commercial advantages that can be gained from big data. The organisation aims to raise the profile of this developing industry, to educate people about the benefits of knowledge based decision making and to encourage firms to start using big data techniques.

Contact Data Science Foundation

Email:admin@datascience.foundation
Telephone: 0161 926 3641
Atlantic Business Centre
Atlantic Street
Altrincham
WA14 5NQ
web: www.datascience.foundation