Decision Trees in Machine Learning

Prashant Gupta May 17, 2017 · 6 min read

A tree has many analogies in real life, and turns out that it has influenced a wide area of **machine learning**, covering both **classification and regression**. In decision analysis, a decision tree can be used to visually and explicitly represent decisions and decision making. As the name goes, it uses a tree-like model of decisions. Though a commonly used tool in data mining for deriving a strategy to reach a particular goal, its also widely used in machine learning, which will be the main focus of this article.

How can an algorithm be represented as a tree?

For this let's consider a very basic example that uses titanic data set for predicting whether a passenger will survive or not. Below model uses 3 features/attributes/columns from the data set, namely sex, age and sibsp (number of spouses or children along).

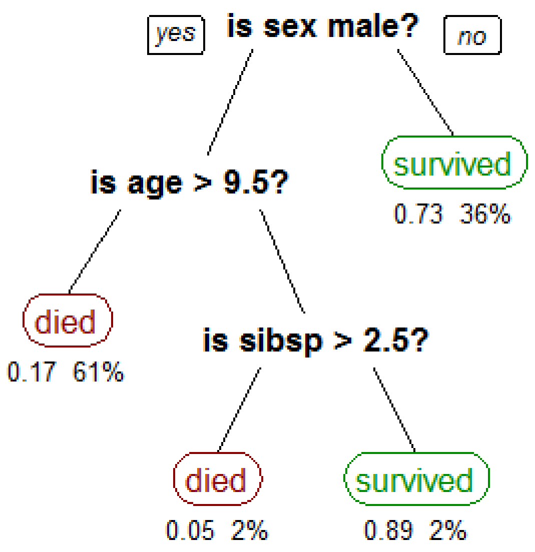


Image taken from wikipedia

A decision tree is drawn upside down with its root at the top. In the image on the left, the bold text in black represents a condition/**internal node**, based on which the tree splits into branches/ **edges**. The end of the branch that doesn't split anymore is the decision/**leaf**, in this case, whether the passenger died or survived, represented as red and green text respectively.

Although, a real dataset will have a lot more features and this will just be a branch in a much bigger tree, but you can't ignore the simplicity of this algorithm. The **feature importance is clear** and relations can be viewed easily. This methodology is more commonly known as **learning decision tree from data** and above tree is called **Classification tree** as the target is to classify passenger as survived or died. **Regression trees** are represented in the same manner, just they predict continuous values like price of a house. In general, Decision Tree algorithms are referred to as CART or Classification and Regression Trees.

So, what is actually going on in the background? Growing a tree involves deciding on **which features to choose** and **what conditions to use** for splitting, along with knowing when to stop. As a tree generally grows arbitrarily, **you will need to trim it down** for it to look beautiful. Lets start with a common technique used for splitting.

Recursive Binary Splitting

In this procedure all the features are considered and different split points are tried and tested using a cost function. The split with the best cost (or lowest cost) is selected.

Consider the earlier example of tree learned from titanic dataset. In the first split or the root, all attributes/features are considered and the training data is divided into groups based on this split. We have 3 features, so will have 3 candidate splits. Now we will ***calculate how much accuracy each split will cost us, using a function***. ***The split that costs least is chosen***, which in our example is sex of the passenger. This ***algorithm is recursive in nature*** as the groups formed can be sub-divided using same strategy. Due to this procedure, this algorithm is also known as the **greedy algorithm**, as we have an excessive desire of lowering the cost. **This makes the root node as best predictor/classifier.**

Cost of a split

Lets take a closer look at **cost functions used for classification and regression**. In both cases the cost functions try to **find most homogeneous branches, or branches having groups with similar responses**. This makes sense we can be more sure that a test data input will follow a certain path.

> Regression : sum(y — prediction)^2

Lets say, we are predicting the price of houses. Now the decision tree will start splitting by considering each feature in training data. The mean of responses of the training data inputs of particular group is considered as prediction for that group. The above function is applied to all data points and cost is calculated for all candidate splits. *Again the split with lowest cost is chosen*. Another cost function involves reduction of standard deviation, more about it can be found here.

Classification : G = sum(pk * (1 — pk))

A Gini score gives an idea of how good a split is by how mixed the response classes are in the groups created by the split. Here, pk is proportion of same class inputs present in a particular group. A perfect class purity occurs when a group contains all inputs from the same class, in which case pk is either 1 or 0 and G = 0, where as a node having a 50–50 split of classes in a group has the worst purity, so for a binary classification it will have pk = 0.5 and G = 0.5.

When to stop splitting?

You might ask ***when to stop growing a tree?*** As a problem usually has a large set of features, it results in large number of split, which in turn gives a huge tree. Such trees are *complex and can lead to overfitting.* So, we need to know when to stop? One way of doing this is to **set a minimum number of training inputs to use on each leaf.** For example we can use a minimum of 10 passengers to reach a decision(died or survived), and ignore any leaf that takes less than 10 passengers. Another way is to set **maximum depth** of your model. **Maximum depth refers to the the length of the longest path from a root to a leaf.**

Pruning

The performance of a tree can be further increased by ***pruning***. *It involves* ***removing the branches that make use of features having low importance***. This way, we reduce the complexity of tree, and thus increasing its predictive power by reducing overfitting.

Pruning can start at either root or the leaves. The simplest method of pruning starts at leaves and removes each node with most popular class in that leaf, this change is kept if it doesn't deteriorate accuracy. Its also called **reduced error pruning**. More sophisticated pruning methods can be used such as **cost complexity pruning** where a learning parameter (alpha) is used to weigh whether nodes can be removed based on the size of the sub-tree. This is also known as **weakest link pruning.**

Advantages of CART

- Simple to understand, interpret, visualize.
- Decision trees *implicitly perform variable screening or feature selection.*
- Can *handle both numerical and categorical data*. Can also *handle multi-output problems.*

- Decision trees require relatively *little effort from users for data preparation.*
- *Nonlinear relationships between parameters do not affect tree performance.*

Disadvantages of CART

- Decision-tree learners *can create over-complex trees* that do not generalize the data well. This is called *overfitting*.
- Decision trees can be unstable because *small variations in the data might result in a completely different tree being generated.* This is called *variance*, which needs to be *lowered by methods like bagging and* ***boosting***.
- Greedy algorithms cannot guarantee to return the globally optimal decision tree. This can be mitigated by training multiple trees, where the features and samples are randomly sampled with replacement.
- Decision tree learners create *biased trees if some classes dominate*. It is therefore recommended to balance the data set prior to fitting with the decision tree.

This is all the basic, to get you at par with decision tree learning. An improvement over decision tree learning is made using technique of **boosting**. A popular library for implementing these algorithms is **Scikit-Learn**. It has a wonderful api that can get your model up an running with **just a few lines of code in python**.

If you liked this article, be sure to click ❤ below to recommend it and if you have any questions, **leave a comment** and I will do my best to answer.

For being more aware of the world of machine learning, **follow me**. It's the best way to find out when I write more articles like this.

You can also follow me on **Twitter**, **email me directly** or **find me on linkedin**. I'd love to hear from you.

That's all folks, Have a nice day :)

Sign up for The Variable

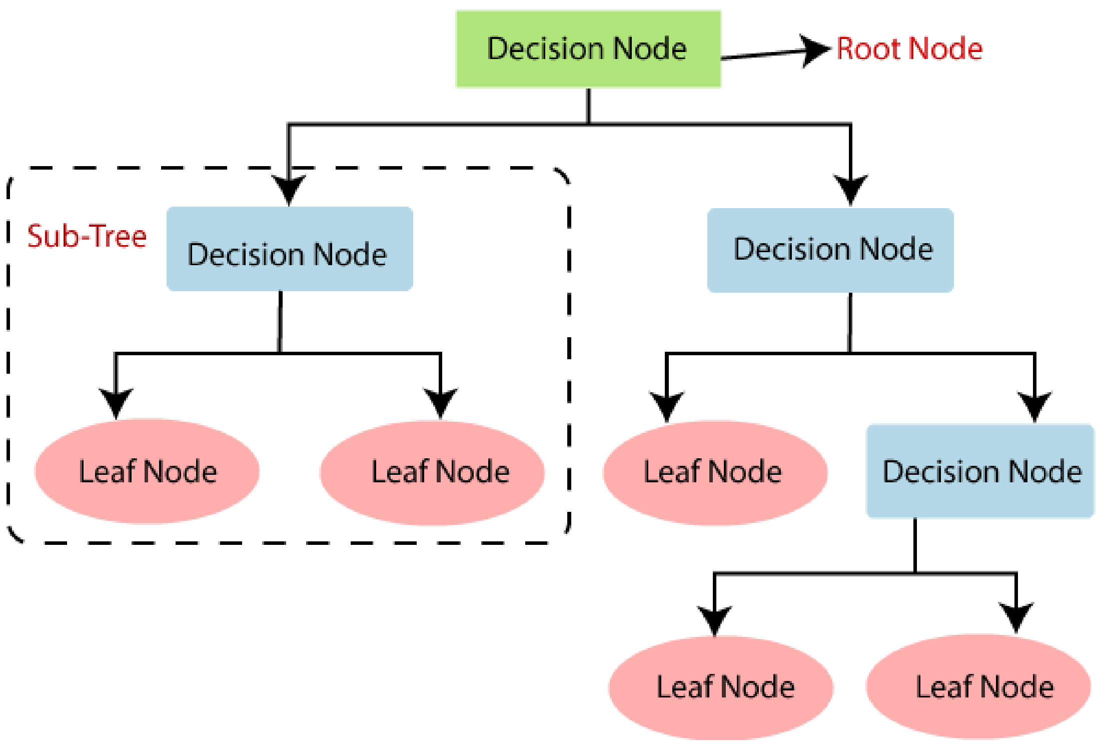
Machine Learning Decision Tree Deep Learning Artificial Intelligence Data Science

⇧ SCROLL TO TOP

Decision Tree Classification Algorithm

- Decision Tree is a **Supervised learning technique** that can be used for both classification and Regression problems, but mostly it is preferred for solving Classification problems. It is a tree-structured classifier, where **internal nodes represent the features of a dataset, branches represent the decision rules** and **each leaf node represents the outcome.**
- In a Decision tree, there are two nodes, which are the **Decision Node** and **Leaf Node.** Decision nodes are used to make any decision and have multiple branches, whereas Leaf nodes are the output of those decisions and do not contain any further branches.
- The decisions or the test are performed on the basis of features of the given dataset.
- ***It is a graphical representation for getting all the possible solutions to a problem/decision based on given conditions.***
- It is called a decision tree because, similar to a tree, it starts with the root node, which expands on further branches and constructs a tree-like structure.
- In order to build a tree, we use the **CART algorithm,** which stands for **Classification and Regression Tree algorithm.**
- A decision tree simply asks a question, and based on the answer (Yes/No), it further split the tree into subtrees.
- Below diagram explains the general structure of a decision tree:

Note: A decision tree can contain categorical data (YES/NO) as well as numeric data.



⇧ SCROLL TO TOP ›n Trees?

There are various algorithms in Machine learning, so choosing the best algorithm for the given dataset and problem is the main point to remember while creating a machine learning model. Below are the two reasons for using the Decision tree:

- Decision Trees usually mimic human thinking ability while making a decision, so it is easy to understand.
- The logic behind the decision tree can be easily understood because it shows a tree-like structure.

Decision Tree Terminologies

- **Root Node:** Root node is from where the decision tree starts. It represents the entire dataset, which further gets divided into two or more homogeneous sets.
- **Leaf Node:** Leaf nodes are the final output node, and the tree cannot be segregated further after getting a leaf node.
- **Splitting:** Splitting is the process of dividing the decision node/root node into sub-nodes according to the given conditions.
- **Branch/Sub Tree:** A tree formed by splitting the tree.
- **Pruning:** Pruning is the process of removing the unwanted branches from the tree.
- **Parent/Child node:** The root node of the tree is called the parent node, and other nodes are called the child nodes.

How does the Decision Tree algorithm Work?

In a decision tree, for predicting the class of the given dataset, the algorithm starts from the root node of the tree. This algorithm compares the values of root attribute with the record (real dataset) attribute and, based on the comparison, follows the branch and jumps to the next node.

For the next node, the algorithm again compares the attribute value with the other sub-nodes and move further. It continues the process until it reaches the leaf node of the tree. The complete process can be better understood using the below algorithm:

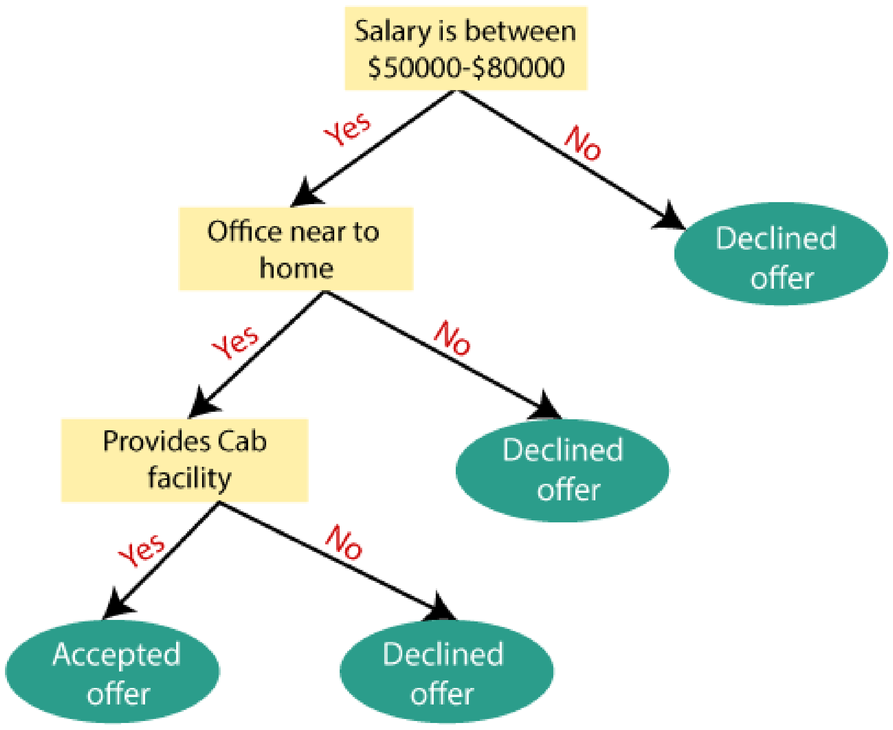
[]

⇧ SCROLL TO TOP

he tree with the root node, says S, which contains the complete dataset.

- **Step-2:** Find the best attribute in the dataset using **Attribute Selection Measure (ASM).**
- **Step-3:** Divide the S into subsets that contains possible values for the best attributes.
- **Step-4:** Generate the decision tree node, which contains the best attribute.
- **Step-5:** Recursively make new decision trees using the subsets of the dataset created in step -3. Continue this process until a stage is reached where you cannot further classify the nodes and called the final node as a leaf node.

Example: Suppose there is a candidate who has a job offer and wants to decide whether he should accept the offer or Not. So, to solve this problem, the decision tree starts with the root node (Salary attribute by ASM). The root node splits further into the next decision node (distance from the office) and one leaf node based on the corresponding labels. The next decision node further gets split into one decision node (Cab facility) and one leaf node. Finally, the decision node splits into two leaf nodes (Accepted offers and Declined offer). Consider the below diagram:



Attribute Selection Measures

While implementing a Decision tree, the main issue arises that how to select the best attribute for the root node and for sub-nodes. So, to solve such problems there is a technique which is called as **Attribute selection measure or ASM.** By this measurement, we can easily select the best attribute for the nodes of the tree. There are two popular techniques for ASM, which are:

- **Information Gain**
- **Gini Index**

1. Information Gain:

⇧ SCROLL TO TOP

- Information gain is the measurement of changes in entropy after the segmentation of a dataset based on an attribute.
- It calculates how much information a feature provides us about a class.
- According to the value of information gain, we split the node and build the decision tree.
- A decision tree algorithm always tries to maximize the value of information gain, and a node/attribute having the highest information gain is split first. It can be calculated using the below formula:

> Information Gain= Entropy(S)- [(Weighted Avg) *Entropy(each feature)

Entropy: Entropy is a metric to measure the impurity in a given attribute. It specifies randomness in data. Entropy can be calculated as:

```
Entropy(s)= -P(yes)log2 P(yes)- P(no) log2 P(no)
```

Where,

- **S= Total number of samples**
- **P(yes)= probability of yes**
- **P(no)= probability of no**

2. Gini Index:

- Gini index is a measure of impurity or purity used while creating a decision tree in the CART(Classification and Regression Tree) algorithm.
- An attribute with the low Gini index should be preferred as compared to the high Gini index.
- It only creates binary splits, and the CART algorithm uses the Gini index to create binary splits.
- Gini index can be calculated using the below formula:

> Gini Index= $1- \sum_j P_j^2$

Pruning: Getting an Optimal Decision tree

Pruning is a process of deleting the unnecessary nodes from a tree in order to get the optimal decision tree.

A too-large tree increases the risk of overfitting, and a small tree may not capture all the important features of the dataset. Therefore, a technique that decreases the size of the learning tree without reducing accuracy is known as Pruning. There are mainly two types of tree **pruning** technology used:

⇧ SCROLL TO TOP `**ty Pruning**

- **Reduced Error Pruning.**

Advantages of the Decision Tree

- It is simple to understand as it follows the same process which a human follow while making any decision in real-life.
- It can be very useful for solving decision-related problems.
- It helps to think about all the possible outcomes for a problem.
- There is less requirement of data cleaning compared to other algorithms.

Disadvantages of the Decision Tree

- The decision tree contains lots of layers, which makes it complex.
- It may have an overfitting issue, which can be resolved using the **Random Forest algorithm.**
- For more class labels, the computational complexity of the decision tree may increase.

Python Implementation of Decision Tree

Now we will implement the Decision tree using Python. For this, we will use the dataset "**user_data.csv**," which we have used in previous classification models. By using the same dataset, we can compare the Decision tree classifier with other classification models such as KNN SVM, LogisticRegression, etc.

Steps will also remain the same, which are given below:

- **Data Pre-processing step**
- **Fitting a Decision-Tree algorithm to the Training set**
- **Predicting the test result**
- **Test accuracy of the result(Creation of Confusion matrix)**
- ⇧ SCROLL TO TOP **test set result.**

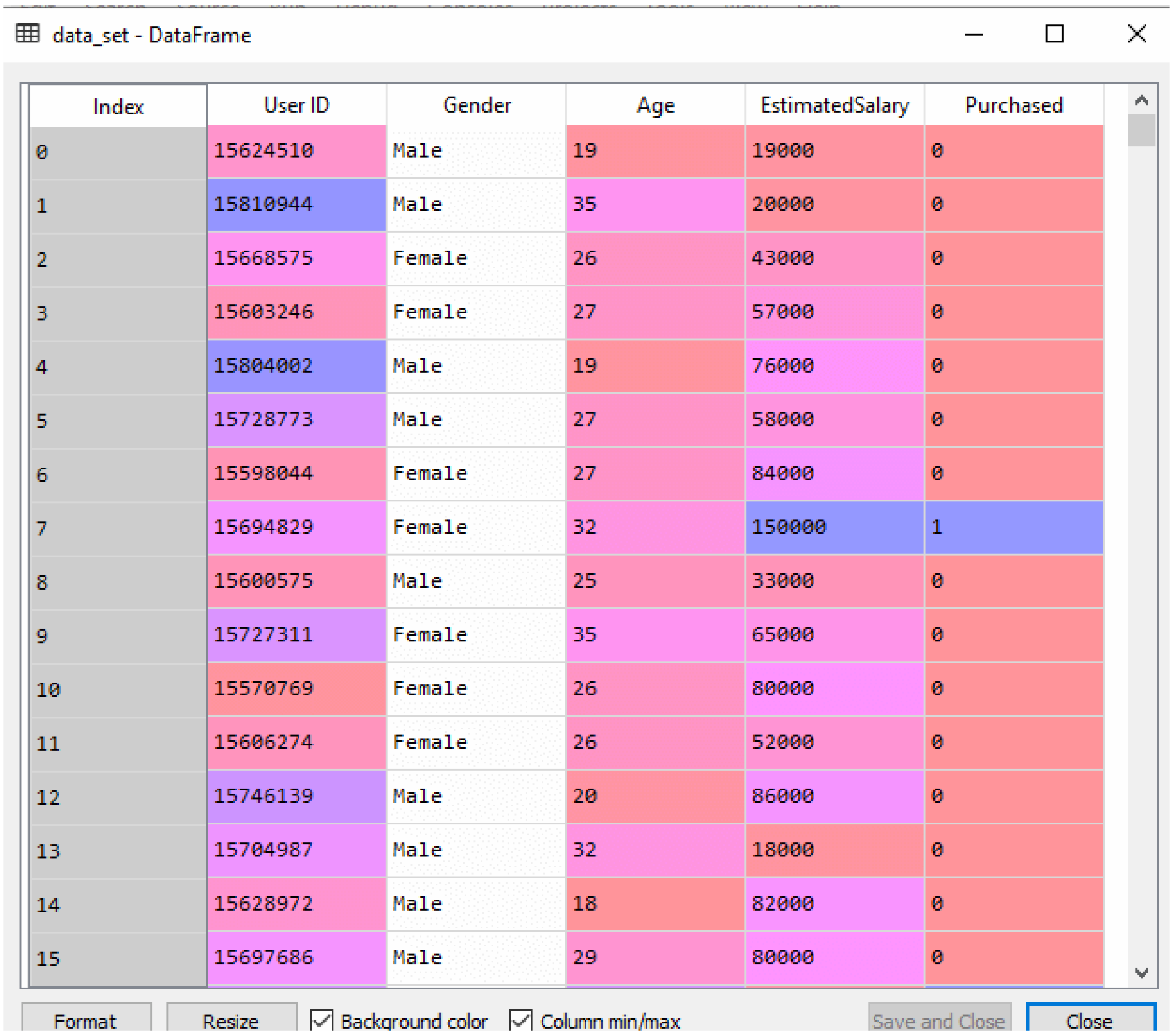
1. Data Pre-Processing Step:

Below is the code for the pre-processing step:

```
# importing libraries
import numpy as nm
import matplotlib.pyplot as mtp
import pandas as pd

#importing datasets
data_set= pd.read_csv('user_data.csv')

#Extracting Independent and dependent Variable
x= data_set.iloc[:, [2,3]].values
y= data_set.iloc[:, 4].values

# Splitting the dataset into training and test set.
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test= train_test_split(x, y, test_size= 0.25, random_state=0)

#feature Scaling
from sklearn.preprocessing import StandardScaler
st_x= StandardScaler()
x_train= st_x.fit_transform(x_train)
x_test= st_x.transform(x_test)
```

In the above code, we have pre-processed the data. Where we have loaded the dataset, which is given as:

⇧ SCROLL TO TOP



Index	User ID	Gender	Age	EstimatedSalary	Purchased
0	15624510	Male	19	19000	0
1	15810944	Male	35	20000	0
2	15668575	Female	26	43000	0
3	15603246	Female	27	57000	0
4	15804002	Male	19	76000	0
5	15728773	Male	27	58000	0
6	15598044	Female	27	84000	0
7	15694829	Female	32	150000	1
8	15600575	Male	25	33000	0
9	15727311	Female	35	65000	0
10	15570769	Female	26	80000	0
11	15606274	Female	26	52000	0
12	15746139	Male	20	86000	0
13	15704987	Male	32	18000	0
14	15628972	Male	18	82000	0
15	15697686	Male	29	80000	0

2. Fitting a Decision-Tree algorithm to the Training set

Now we will fit the model to the training set. For this, we will import the **DecisionTreeClassifier** class from **sklearn.tree** library. Below is the code for it:

```
#Fitting Decision Tree classifier to the training set
From sklearn.tree import DecisionTreeClassifier
classifier= DecisionTreeClassifier(criterion='entropy', random_state=0)
classifier.fit(x_train, y_train)
```

In the above code, we have created a classifier object, in which we have passed two main parameters;

- **"criterion='entropy':** Criterion is used to measure the quality of split, which is calculated by information gain given by entropy.
- **random_state=0":** For generating the random states.

Below is the output for this:

⇧ SCROLL TO TOP

```
Out[8]:
DecisionTreeClassifier(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=False,
                       random_state=0, splitter='best')
```

3. Predicting the test result

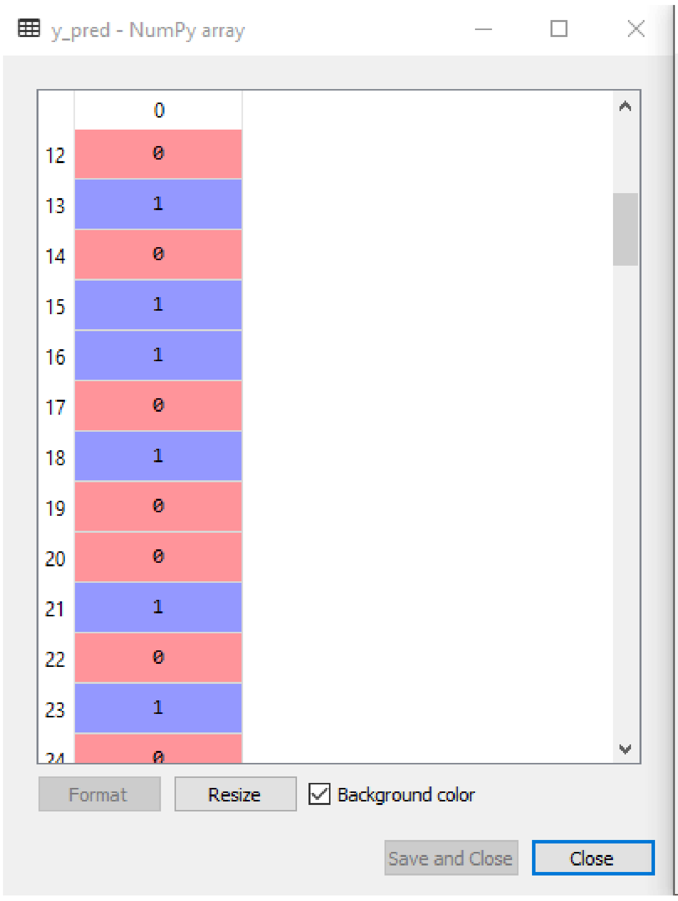
Now we will predict the test set result. We will create a new prediction vector **y_pred.** Below is the code for it:

```
#Predicting the test set result
y_pred= classifier.predict(x_test)
```

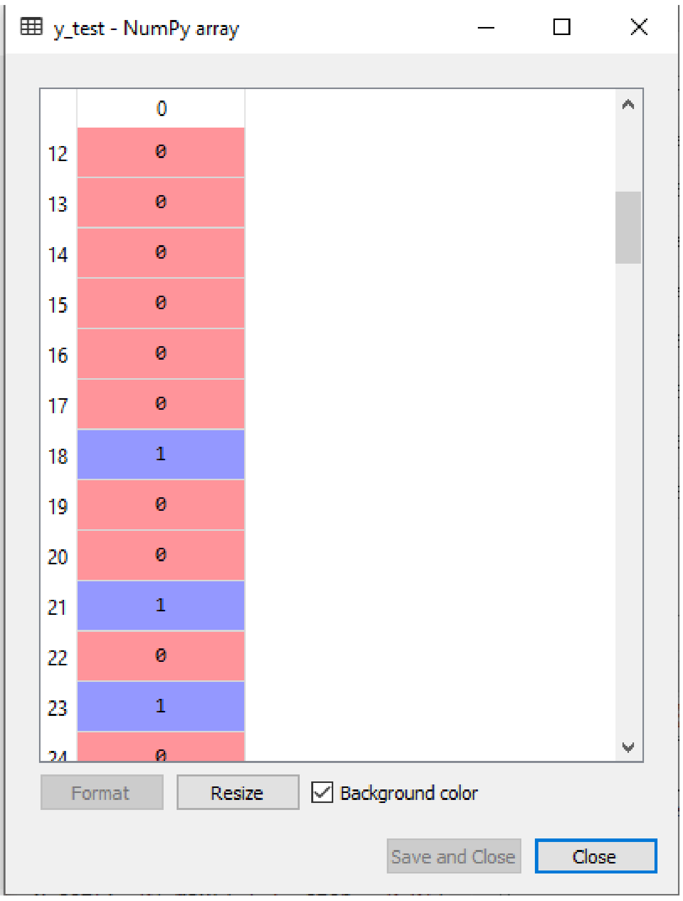
Output:

In the below output image, the predicted output and real test output are given. We can clearly see that there are some values in the prediction vector, which are different from the real vector values. These are prediction errors.

⇧ SCROLL TO TOP


y_pred - NumPy array

	0
12	0
13	1
14	0
15	1
16	1
17	0
18	1
19	0
20	0
21	1
22	0
23	1
24	0


y_test - NumPy array

	0
12	0
13	0
14	0
15	0
16	0
17	0
18	1
19	0
20	0
21	1
22	0
23	1
24	0

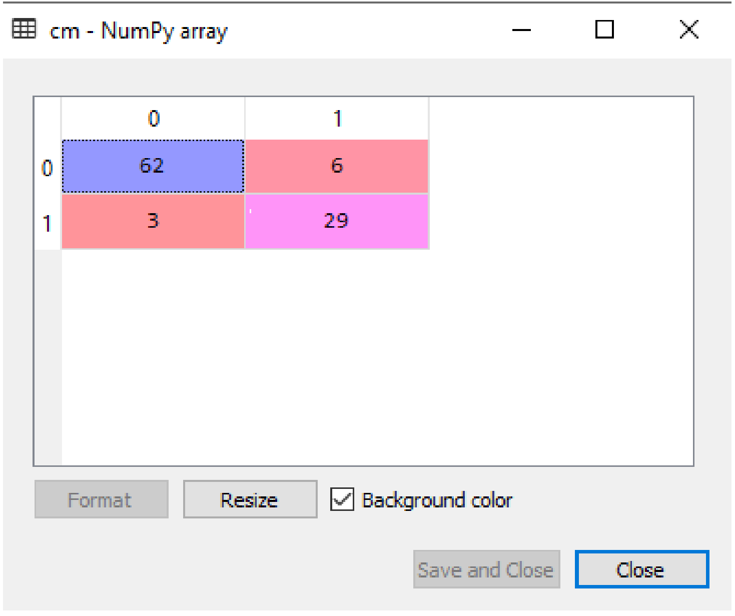
4. Test accuracy of the result (Creation of Confusion matrix)

In the above output, we have seen that there were some incorrect predictions, so if we want to know the number of correct and incorrect predictions, we need to use the confusion matrix. Below is the code for it:

```
#Creating the Confusion matrix
from sklearn.metrics import confusion_matrix
cm= confusion_matrix(y_test, y_pred)
```

Output:

⇧ SCROLL TO TOP



	0	1
0	62	6
1	3	29

In the above output image, we can see the confusion matrix, which has **6+3= 9 incorrect predictions** and**62+29=91 correct predictions. Therefore, we can say that compared to other classification models, the Decision Tree classifier made a good prediction.**

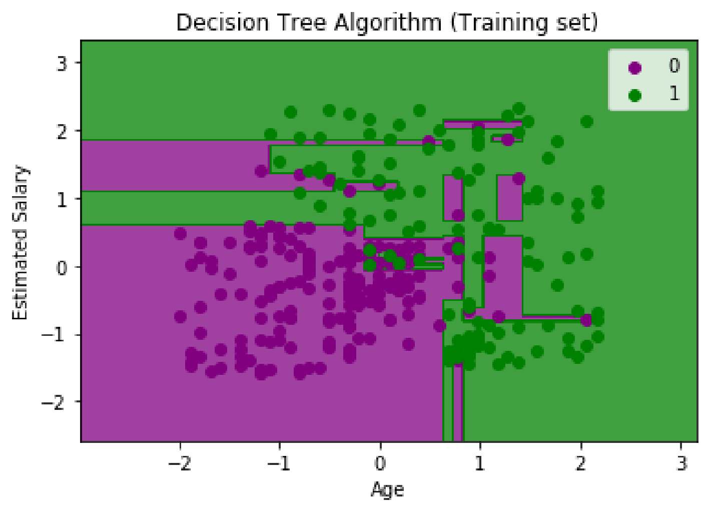
5. Visualizing the training set result:

Here we will visualize the training set result. To visualize the training set result we will plot a graph for the decision tree classifier. The classifier will predict yes or No for the users who have either Purchased or Not purchased the SUV car as we did in Logistic Regression. Below is the code for it:

```
#Visulaizing the trianing set result
from matplotlib.colors import ListedColormap
x_set, y_set = x_train, y_train
x1, x2 = nm.meshgrid(nm.arange(start = x_set[:, 0].min() - 1, stop = x_set[:, 0].max() + 1, step  =0.01),
nm.arange(start = x_set[:, 1].min() - 1, stop = x_set[:, 1].max() + 1, step = 0.01))
mtp.contourf(x1, x2, classifier.predict(nm.array([x1.ravel(), x2.ravel()]).T).reshape(x1.shape),
alpha = 0.75, cmap = ListedColormap(('purple','green' )))
mtp.xlim(x1.min(), x1.max())
mtp.ylim(x2.min(), x2.max())
fori, j in enumerate(nm.unique(y_set)):
mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
        c = ListedColormap(('purple', 'green'))(i), label = j)
mtp.title('Decision Tree Algorithm (Training set)')
mtp.xlabel('Age')
mtp.ylabel('Estimated Salary')
mtp.legend()
```

⇧ SCROLL TO TOP

Output:



The above output is completely different from the rest classification models. It has both vertical and horizontal lines that are splitting the dataset according to the age and estimated salary variable.

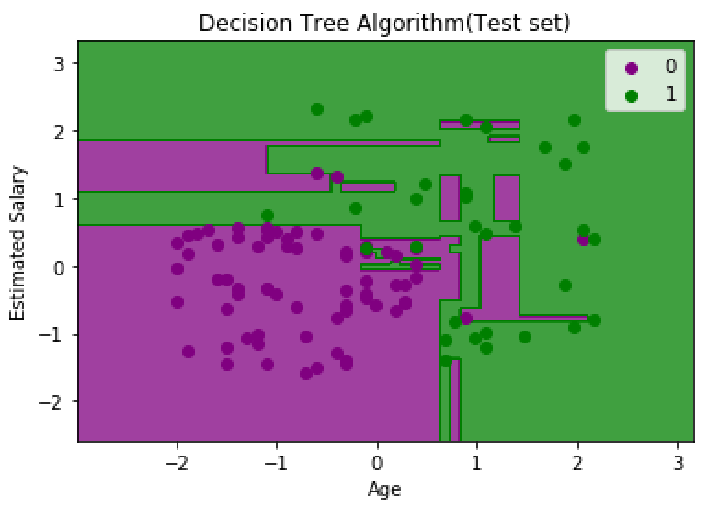
As we can see, the tree is trying to capture each dataset, which is the case of overfitting.

6. Visualizing the test set result:

Visualization of test set result will be similar to the visualization of the training set except that the training set will be replaced with the test set.

```
#Visulaizing the test set result
from matplotlib.colors import ListedColormap
x_set, y_set = x_test, y_test
x1, x2 = nm.meshgrid(nm.arange(start = x_set[:, 0].min() - 1, stop = x_set[:, 0].max() + 1, step  =0.01),
nm.arange(start = x_set[:, 1].min() - 1, stop = x_set[:, 1].max() + 1, step = 0.01))
mtp.contourf(x1, x2, classifier.predict(nm.array([x1.ravel(), x2.ravel()]).T).reshape(x1.shape),
alpha = 0.75, cmap = ListedColormap(('purple','green' )))
mtp.xlim(x1.min(), x1.max())
mtp.ylim(x2.min(), x2.max())
fori, j in enumerate(nm.unique(y_set)):
mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
        c = ListedColormap(('purple', 'green'))(i), label = j)
mtp.title('Decision Tree Algorithm(Test set)')
mtp.xlabel('Age')
mtp.ylabel('Estimated Salary')
mtp.legend()
mtp.show()
```

⇧ SCROLL TO TOP



As we can see in the above image that there are some green data points within the purple region and vice versa. So, these are the incorrect predictions which we have discussed in the confusion matrix.

← Prev

Next →

⇧ SCROLL TO TOP

KDnuggets Home » News » 2020 » Jan » Tutorials, Overviews » Decision Tree Algorithm, Explained (20:n02)

Decision Tree Algorithm, Explained

Jan 2020 Gold KDnuggets Blog

2020 Gold KDnuggets Blog

<= Previous post
Next post =>

Like 83 Share 83 Tweet Share Share 520

Tags: Algorithms, Decision Trees, Explained

All you need to know about decision trees and how to build and optimize decision tree classifier.

By Nagesh Singh Chauhan, Data Science Enthusiast.

comments

Introduction

Classification is a two-step process, learning step and prediction step, in machine learning. In the learning step, the model is developed based on given training data. In the prediction step, the model is used to predict the response for given data. Decision Tree is one of the easiest and popular classification algorithms to understand and interpret.

Decision Tree Algorithm

Decision Tree algorithm belongs to the family of supervised learning algorithms. Unlike other supervised learning algorithms, the decision tree algorithm can be used for solving **regression and classification problems** too.

The goal of using a Decision Tree is to create a training model that can use to predict the class or value of the target variable by **learning simple decision rules** inferred from prior data(training data).

In Decision Trees, for predicting a class label for a record we start from the **root** of the tree. We compare the values of the root attribute with the record's attribute. On the basis of comparison, we follow the branch corresponding to that value and jump to the next node.

Types of Decision Trees

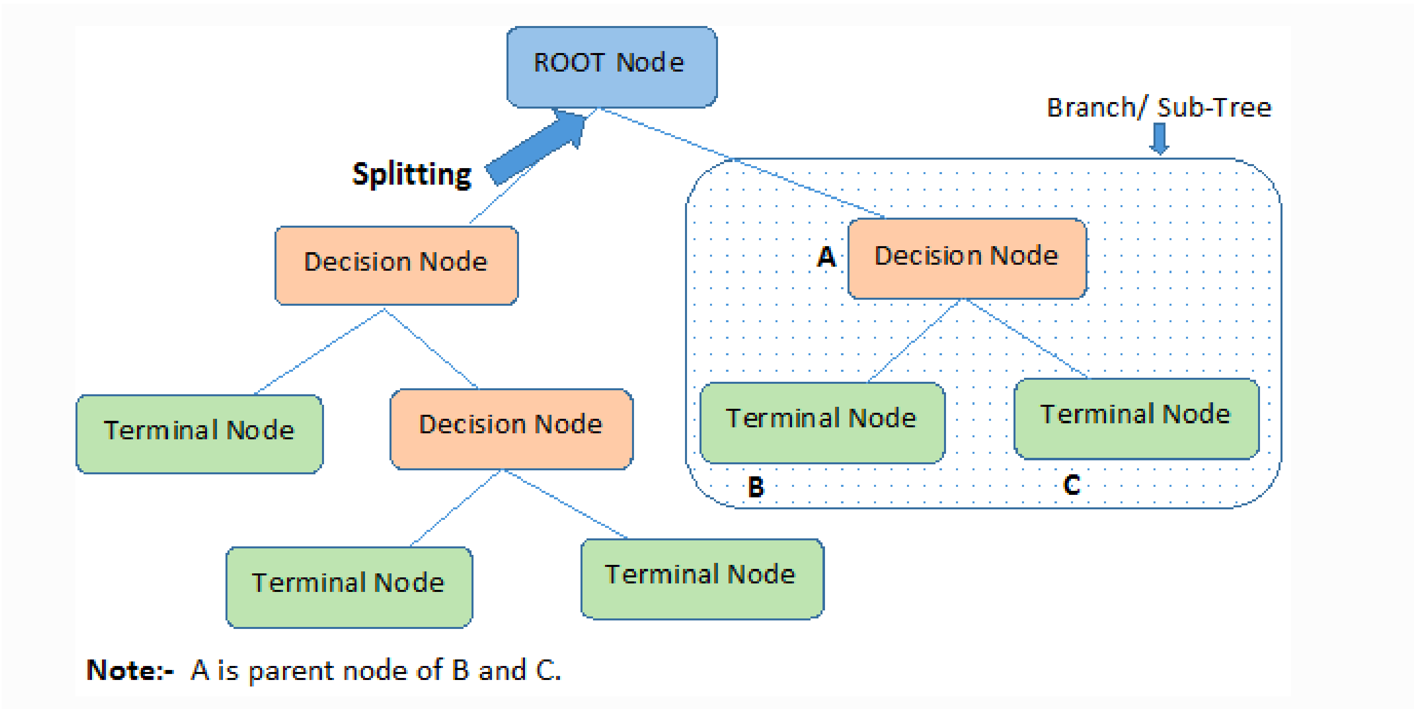
Types of decision trees are based on the type of target variable we have. It can be of two types:

1. **Categorical Variable Decision Tree:** Decision Tree which has a categorical target variable then it called a **Categorical variable decision tree.**
2. **Continuous Variable Decision Tree:** Decision Tree has a continuous target variable then it is called **Continuous Variable Decision Tree.**

Example:- Let's say we have a problem to predict whether a customer will pay his renewal premium with an insurance company (yes/ no). Here we know that the income of customers is a significant variable but the insurance company does not have income details for all customers. Now, as we know this is an important variable, then we can build a decision tree to predict customer income based on occupation, product, and various other variables. In this case, we are predicting values for the continuous variables.

Important Terminology related to Decision Trees

1. **Root Node:** It represents the entire population or sample and this further gets divided into two or more homogeneous sets.
2. **Splitting:** It is a process of dividing a node into two or more sub-nodes.
3. **Decision Node:** When a sub-node splits into further sub-nodes, then it is called the decision node.
4. **Leaf / Terminal Node:** Nodes do not split is called Leaf or Terminal node.
5. **Pruning:** When we remove sub-nodes of a decision node, this process is called pruning. You can say the opposite process of splitting.
6. **Branch / Sub-Tree:** A subsection of the entire tree is called branch or sub-tree.
7. **Parent and Child Node:** A node, which is divided into sub-nodes is called a parent node of sub-nodes whereas sub-nodes are the child of a parent node.



Decision trees classify the examples by sorting them down the tree from the root to some leaf/terminal node, with the leaf/terminal node providing the classification of the example.

Each node in the tree acts as a test case for some attribute, and each edge descending from the node corresponds to the possible answers to the test case. This process is recursive in nature and is repeated for every subtree rooted at the new node.

Assumptions while creating Decision Tree

Below are some of the assumptions we make while using Decision tree:

- In the beginning, the whole training set is considered as the **root.**
- Feature values are preferred to be categorical. If the 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 Trees follow **Sum of Product (SOP) r**epresentation. The Sum of product (SOP) is also known as **Disjunctive Normal Form**. For a class, every branch from the root of the tree to a leaf node having the same class is conjunction (product) of values, different branches ending in that class form a disjunction (sum).

The primary challenge in the decision tree implementation is to identify which attributes do we need to consider as the root node and each level. Handling this is to know as the attributes selection. We have different attributes selection measures to identify the attribute which can be considered as the root note at each level.

How do Decision Trees work?

The decision of making strategic splits heavily affects a tree's accuracy. The decision criteria are different for classification and regression trees.

Decision trees use multiple algorithms to decide to split a node into two or more sub-nodes. The creation of sub-nodes increases the homogeneity of resultant sub-nodes. In other words, we can say that the purity of the node increases with respect to the target variable. The decision tree splits the nodes on all available variables and then selects the split which results in most homogeneous sub-nodes.

The algorithm selection is also based on the type of target variables. Let us look at some algorithms used in Decision Trees:

ID3 → (extension of D3)
C4.5 → (successor of ID3)
CART → (Classification And Regression Tree)
CHAID → (Chi-square automatic interaction detection Performs multi-level splits when computing classification trees)
MARS → (multivariate adaptive regression splines)

The ID3 algorithm builds decision trees using a top-down greedy search approach through the space of possible branches with no backtracking. A greedy algorithm, as the name suggests, always makes the choice that seems to be the best at that moment.

Steps in ID3 algorithm:

1. It begins with the original set S as the root node.
2. On each iteration of the algorithm, it iterates through the very unused attribute of the set S and calculates **Entropy(H)** and **Information gain(IG)** of this attribute.

3. It then selects the attribute which has the smallest Entropy or Largest Information gain.
4. The set S is then split by the selected attribute to produce a subset of the data.
5. The algorithm continues to recur on each subset, considering only attributes never selected before.

Attribute Selection Measures

If the dataset consists of **N** attributes then deciding which attribute to place at the root or at different levels of the tree as internal nodes is a complicated step. By just randomly selecting any node to be the root can't solve the issue. If we follow a random approach, it may give us bad results with low accuracy.

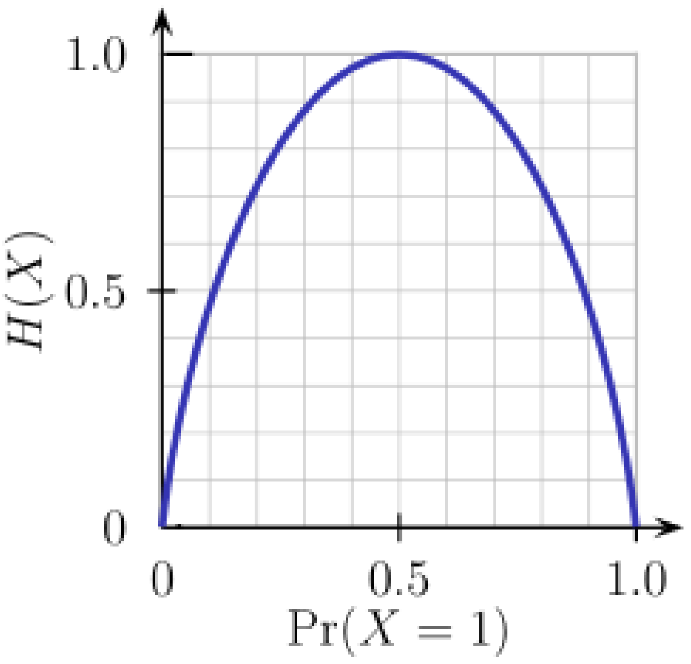
For solving this attribute selection problem, researchers worked and devised some solutions. They suggested using some *criteria* like :

Entropy,
Information gain,
Gini index,
Gain Ratio,
Reduction in Variance
Chi-Square

These criteria will calculate values for every attribute. The values are sorted, and attributes are placed in the tree by following the order i.e, the attribute with a high value(in case of information gain) is placed at the root.
While using Information Gain as a criterion, we assume attributes to be categorical, and for the Gini index, attributes are assumed to be continuous.

Entropy

Entropy is a measure of the randomness in the information being processed. The higher the entropy, the harder it is to draw any conclusions from that information. Flipping a coin is an example of an action that provides information that is random.

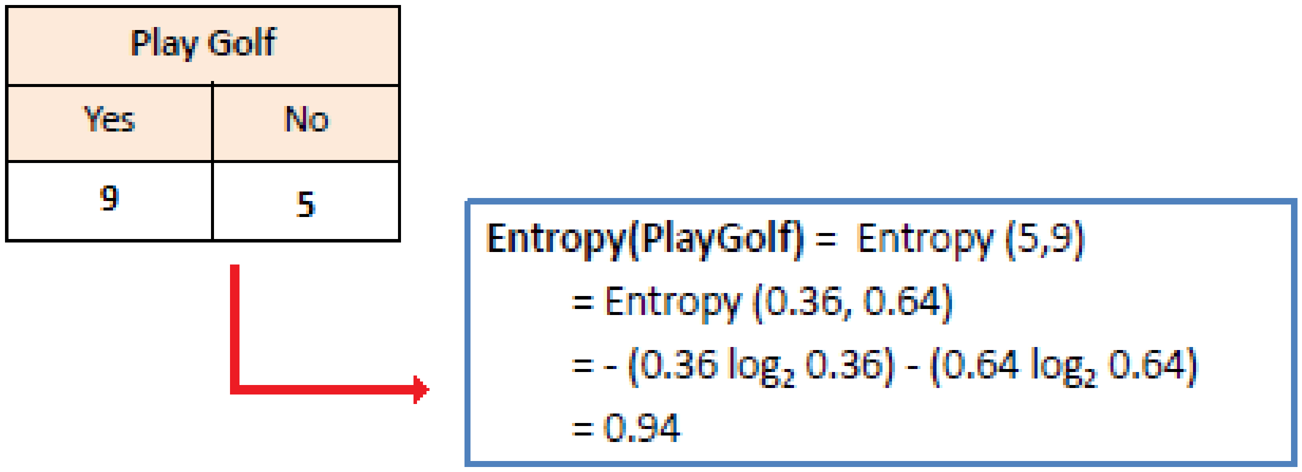


From the above graph, it is quite evident that the entropy H(X) is zero when the probability is either 0 or 1. The Entropy is maximum when the probability is 0.5 because it projects perfect randomness in the data and there is no chance if perfectly determining the outcome.

> **ID3 follows the rule — A branch with an entropy of zero is a leaf node and A brach with entropy more than zero needs further splitting.**

Mathematically Entropy for 1 attribute is represented as:

$$E(S) = \sum_{i=1}^{c} -p_i \log_2 p_i$$



Play Golf	
Yes	No
9	5

Entropy(PlayGolf) = Entropy (5,9)

$= \text{Entropy}(0.36, 0.64)$

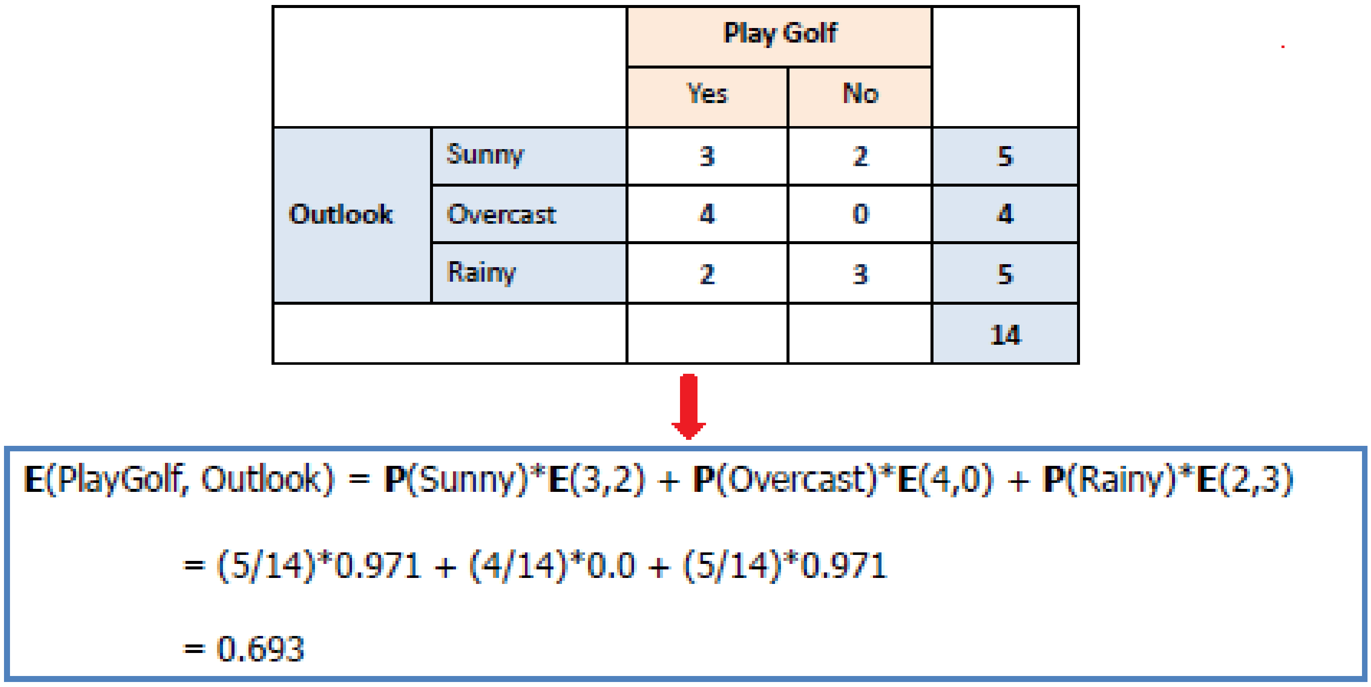
$= -(0.36 \log_2 0.36) - (0.64 \log_2 0.64)$

$= 0.94$

Where S → **Current state, and Pi → Probability of an event *i* of state S or Percentage of class *i* in a node of state S.**

Mathematically Entropy for multiple attributes is represented as:

$$E(T,X) = \sum_{c \in X} P(c)E(c)$$



		Play Golf		
		Yes	No	
Outlook	Sunny	3	2	5
	Overcast	4	0	4
	Rainy	2	3	5
				14

E(PlayGolf, Outlook) = **P**(Sunny)***E**(3,2) + **P**(Overcast)***E**(4,0) + **P**(Rainy)***E**(2,3)

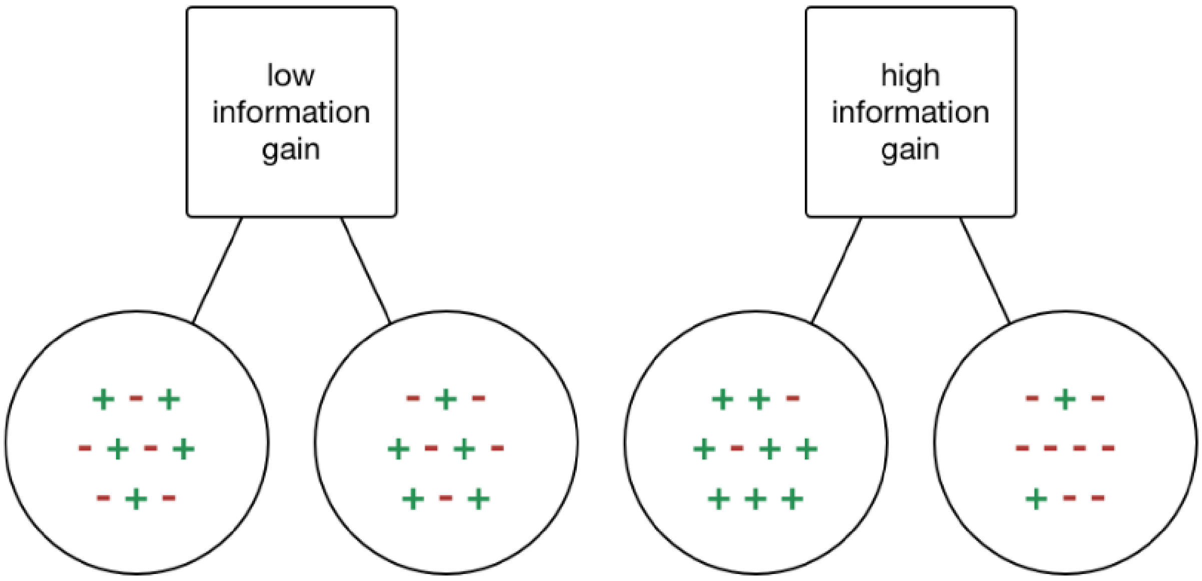
$= (5/14)*0.971 + (4/14)*0.0 + (5/14)*0.971$

$= 0.693$

where **T→ Current state and X → Selected attribute**

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. Constructing a decision tree is all about finding an attribute that returns the highest information gain and the smallest entropy.



Information Gain

Information gain is a decrease in entropy. It computes the difference between entropy before split and average entropy after split of the dataset based on given attribute values. ID3 (Iterative Dichotomiser) decision tree algorithm uses information gain.

Mathematically, IG is represented as:

$$\text{Information Gain}(T,X) = \text{Entropy}(T) - \text{Entropy}(T, X)$$

$$\begin{aligned} \text{IG(PlayGolf, Outlook)} &= \text{E(PlayGolf)} - \text{E(PlayGolf, Outlook)} \\ &= 0.940 - 0.693 \\ &= 0.247 \end{aligned}$$

In a much simpler way, we can conclude that:

$$Information\ Gain = Entropy(before) - \sum_{j=1}^{K} Entropy(j,\ after)$$

Information Gain

Where “before” is the dataset before the split, K is the number of subsets generated by the split, and (j, after) is subset j after the split.

Gini Index

You can understand the Gini index as a cost function used to evaluate splits in the dataset. It is calculated by subtracting the sum of the squared probabilities of each class from one. It favors larger partitions and easy to implement whereas information gain favors smaller partitions with distinct values.

$$Gini = 1 - \sum_{i=1}^{C} (p_i)^2$$

Gini Index

Gini Index works with the categorical target variable "Success" or "Failure". It performs only Binary splits.

> Higher value of Gini index implies higher inequality, higher heterogeneity.

Steps to Calculate Gini index for a split

1. Calculate Gini for sub-nodes, using the above formula for success(p) and failure(q) (p^2+q^2).
2. Calculate the Gini index for split using the weighted Gini score of each node of that split.

CART (Classification and Regression Tree) uses the Gini index method to create split points.

Gain ratio

Information gain is biased towards choosing attributes with a large number of values as root nodes. It means it prefers the attribute with a large number of distinct values.

C4.5, an improvement of ID3, uses Gain ratio which is a modification of Information gain that reduces its bias and is usually the best option. Gain ratio overcomes the problem with information gain by taking into account the number of branches that would result before making the split. It corrects information gain by taking the intrinsic information of a split into account.

> Let us consider if we have a dataset that has users and their movie genre preferences based on variables like gender, group of age, rating, blah, blah. With the help of information gain, you split at 'Gender' (assuming it has the highest information gain) and now the variables 'Group of Age' and 'Rating' could be equally important and with the help of gain ratio, it will penalize a variable with more distinct values which will help us decide the split at the next level.

$$Gain\ Ratio = \frac{Information\ Gain}{SplitInfo} = \frac{Entropy\ (before) - \sum_{j=1}^{K} Entropy(j, after)}{\sum_{j=1}^{K} w_j \log_2 w_j}$$

Gain Ratio

Where "before" is the dataset before the split, K is the number of subsets generated by the split, and (j, after) is subset j after the split.

Reduction in Variance

Reduction in variance is an algorithm used for continuous target variables (regression problems). This algorithm uses the standard formula of variance to choose the best split. The split with lower variance is selected as the criteria to split the population:

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

Above X-bar is the mean of the values, X is actual and n is the number of values.

Steps to calculate Variance:

1. Calculate variance for each node.
2. Calculate variance for each split as the weighted average of each node variance.

Chi-Square

The acronym CHAID stands for *Chi*-squared Automatic Interaction Detector. It is one of the oldest tree classification methods. It finds out the statistical significance between the differences between sub-nodes and parent node. We measure it by the sum of squares of standardized differences between observed and expected frequencies of the target variable.

It works with the categorical target variable “Success” or “Failure”. It can perform two or more splits. Higher the value of Chi-Square higher the statistical significance of differences between sub-node and Parent node.

It generates a tree called CHAID (Chi-square Automatic Interaction Detector).

Mathematically, Chi-squared is represented as:

$$\chi^2 = \sum \frac{(O-E)^2}{E}$$

Where:

χ^2 = Chi Square obtained
$\sum$ = the sum of
O = observed score
E = expected score

Steps to Calculate Chi-square for a split:

1. Calculate Chi-square for an individual node by calculating the deviation for Success and Failure both
2. Calculated Chi-square of Split using Sum of all Chi-square of success and Failure of each node of the split

How to avoid/counter Overfitting in Decision Trees?

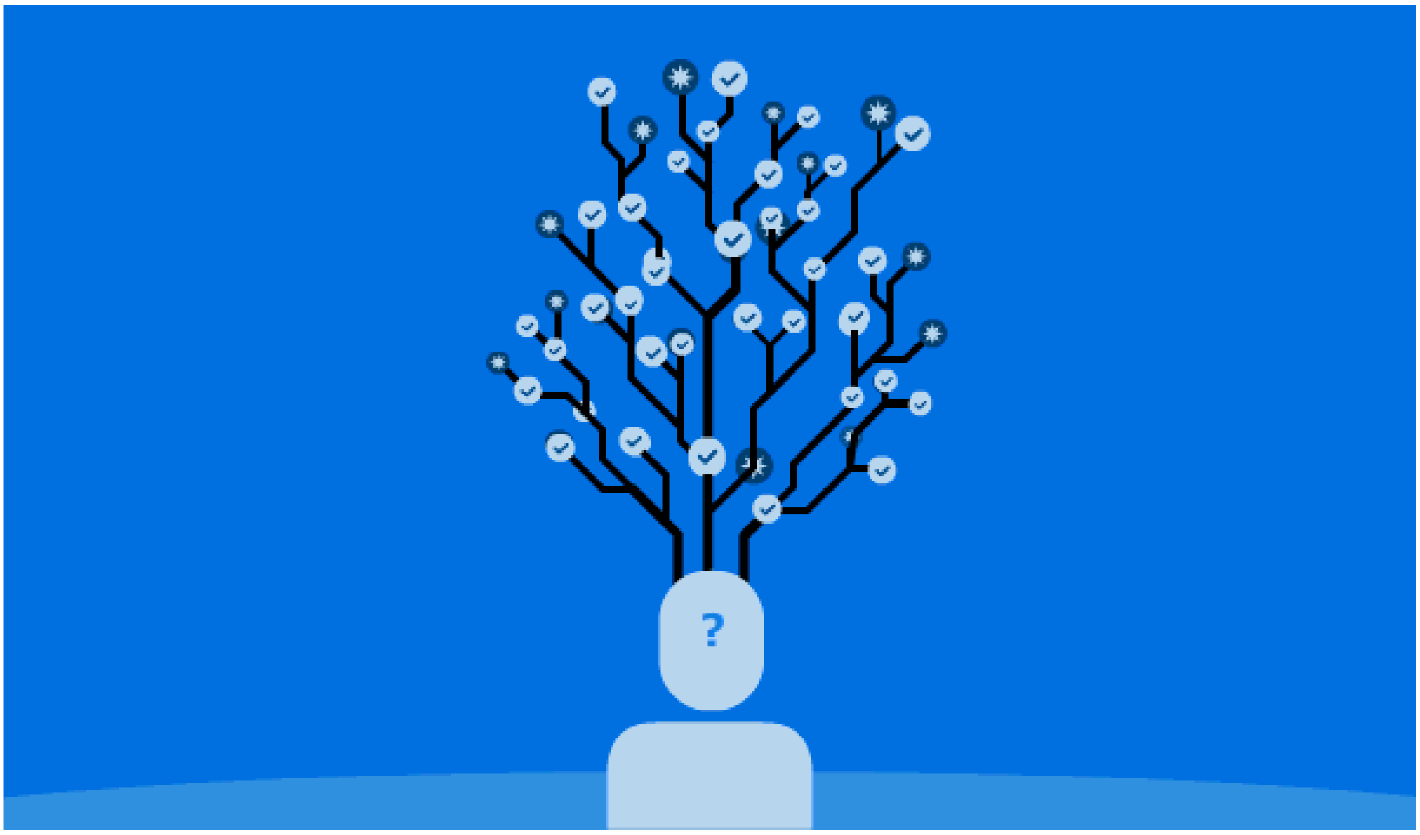
The common problem with Decision trees, especially having a table full of columns, they fit a lot. Sometimes it looks like the tree memorized the training data set. If there is no limit set on a decision tree, it will give you 100% accuracy on the training data set because in the worse case it will end up making 1 leaf for each observation. Thus this affects the accuracy when predicting samples that are not part of the training set.

Here are two ways to remove overfitting:

1. Pruning Decision Trees.
2. Random Forest

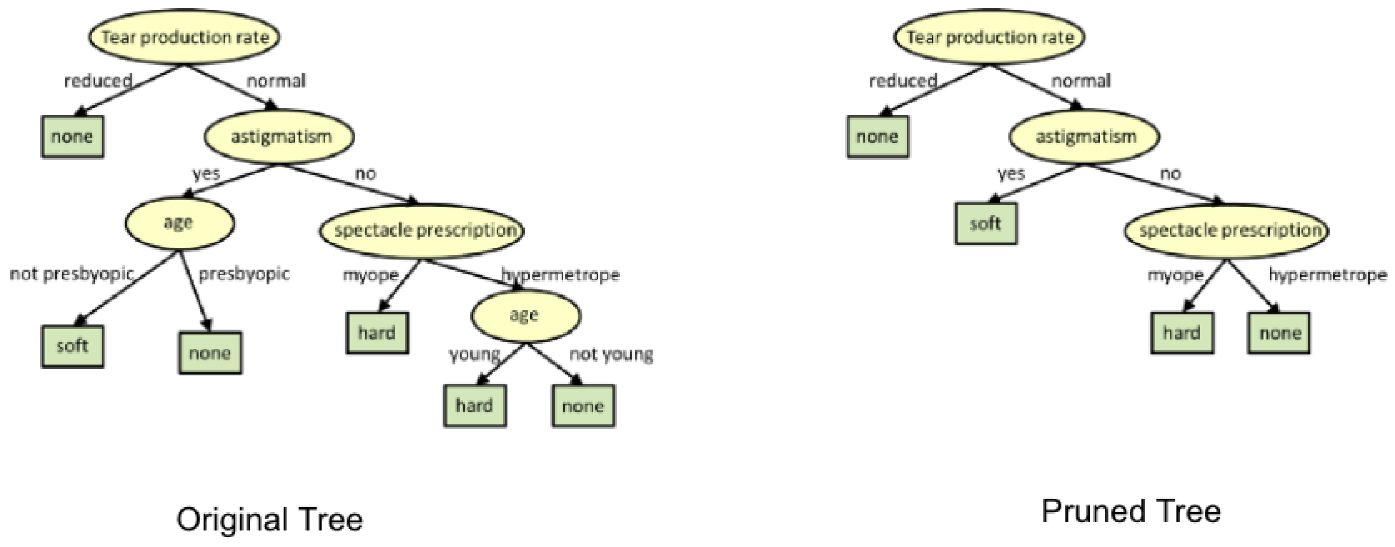
Pruning Decision Trees

The splitting process results in fully grown trees until the stopping criteria are reached. But, the fully grown tree is likely to overfit the data, leading to poor accuracy on unseen data.



Pruning in action

In **pruning**, you trim off the branches of the tree, i.e., remove the decision nodes starting from the leaf node such that the overall accuracy is not disturbed. This is done by segregating the actual training set into two sets: training data set, D and validation data set, V. Prepare the decision tree using the segregated training data set, D. Then continue trimming the tree accordingly to optimize the accuracy of the validation data set, V.



Pruning

In the above diagram, the 'Age' attribute in the left-hand side of the tree has been pruned as it has more importance on the right-hand side of the tree, hence removing overfitting.

Random Forest

Random Forest is an example of ensemble learning, in which we combine multiple machine learning algorithms to obtain better predictive performance.

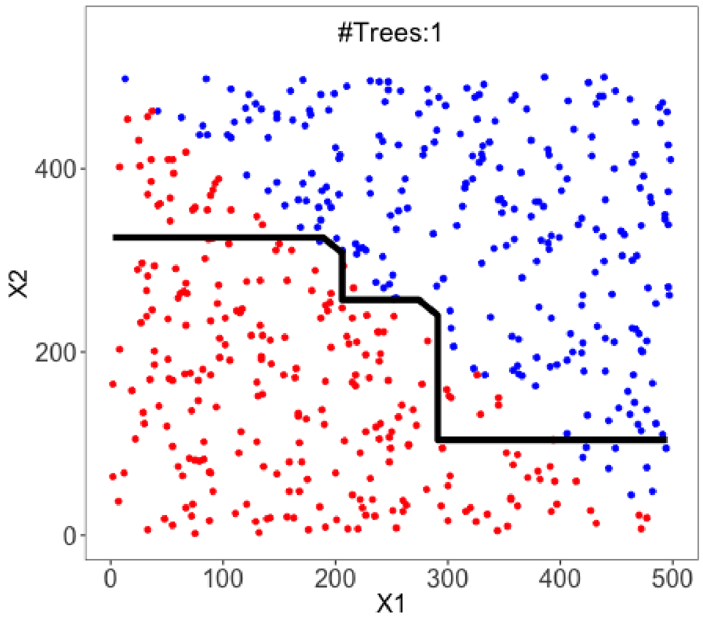
Why the name "Random"?

Two key concepts that give it the name random:

1. A random sampling of training data set when building trees.
2. Random subsets of features considered when splitting nodes.

A technique known as bagging is used to create an ensemble of trees where multiple training sets are generated with replacement.

In the bagging technique, a data set is divided into **N** samples using randomized sampling. Then, using a single learning algorithm a model is built on all samples. Later, the resultant predictions are combined using voting or averaging in parallel.



Random Forest in action

Which is better Linear or tree-based models?

Well, it depends on the kind of problem you are solving.

1. If the relationship between dependent & independent variables is well approximated by a linear model, linear regression will outperform the tree-based model.
2. If there is a high non-linearity & complex relationship between dependent & independent variables, a tree model will outperform a classical regression method.
3. If you need to build a model that is easy to explain to people, a decision tree model will always do better than a linear model. Decision tree models are even simpler to interpret than linear regression!

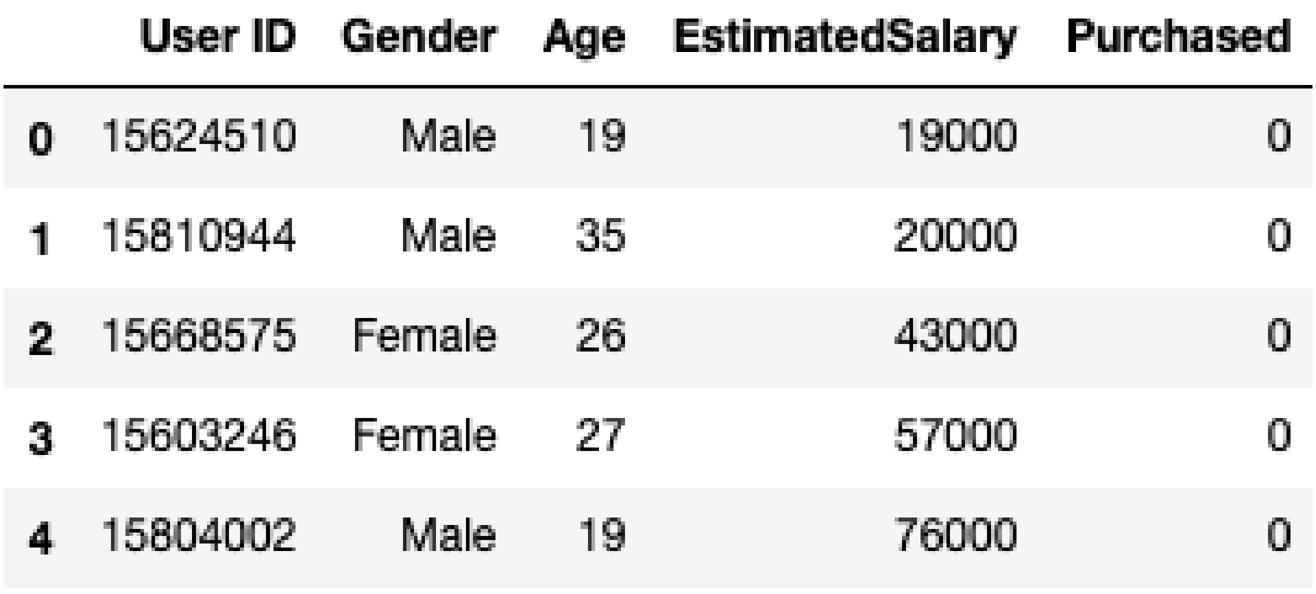
Decision Tree Classifier Building in Scikit-learn

The dataset that we have is a supermarket data which can be downloaded from here.
Load all the basic libraries.

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

Load the dataset. It consists of 5 features, `UserID`, `Gender`, `Age`, `EstimatedSalary` and `Purchased`.

```
data = pd.read_csv('/Users/ML/DecisionTree/Social.csv')
data.head()
```



	User ID	Gender	Age	EstimatedSalary	Purchased
0	15624510	Male	19	19000	0
1	15810944	Male	35	20000	0
2	15668575	Female	26	43000	0
3	15603246	Female	27	57000	0
4	15804002	Male	19	76000	0

Dataset

We will take only `Age` and `EstimatedSalary` as our independent variables `X` because of other features like `Gender` and `User ID` are irrelevant and have no effect on the purchasing capacity of a person. Purchased is our dependent variable `y`.

```
feature_cols = ['Age','EstimatedSalary' ]X = data.iloc[:,[2,3]].values
y = data.iloc[:,4].values
```

The next step is to split the dataset into training and test.

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test =  train_test_split(X,y,test_size = 0.25, random_state= 0)
```

Perform feature scaling

```
#feature scaling
from sklearn.preprocessing import StandardScaler
sc_X = StandardScaler()
X_train = sc_X.fit_transform(X_train)
X_test = sc_X.transform(X_test)
```

Fit the model in the Decision Tree classifier.

```
from sklearn.tree import DecisionTreeClassifier
classifier = DecisionTreeClassifier()
classifier = classifier.fit(X_train,y_train)
```

Make predictions and check accuracy.

```
#prediction
y_pred = classifier.predict(X_test)#Accuracy
from sklearn import metricsprint('Accuracy Score:', metrics.accuracy_score(y_test,y_pred))
```

The decision tree classifier gave an accuracy of 91%.

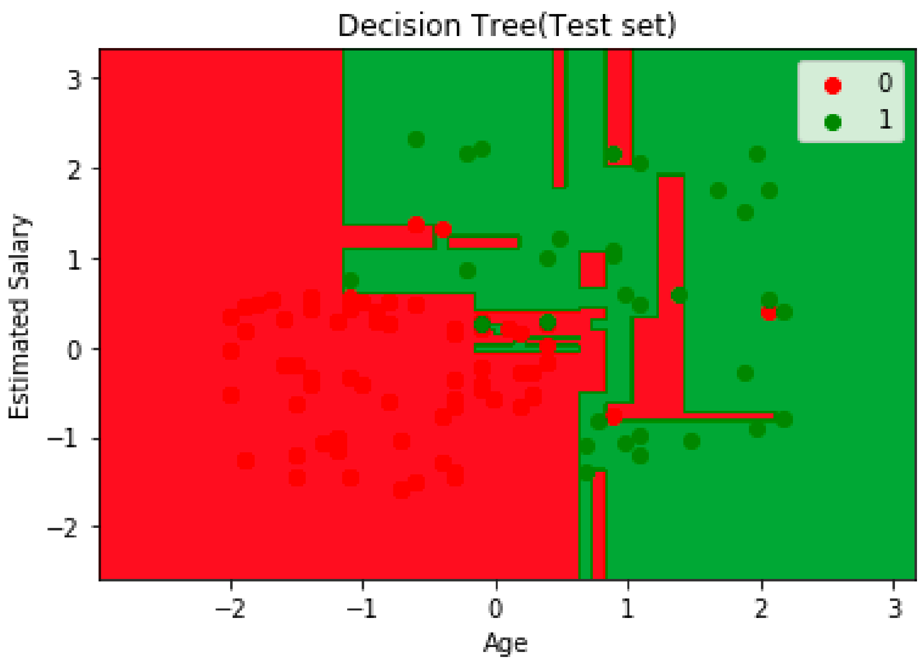
Confusion Matrix

```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)Output:
array([[64,  4],
       [ 2, 30]])
```

It means 6 observations have been classified as false.

Let us first visualize the model prediction results.

```
from matplotlib.colors import ListedColormap
X_set, y_set = X_test, y_test
X1, X2 = np.meshgrid(np.arange(start = X_set[:,0].min()-1, stop= X_set[:,0].max()+1, step = 0.01),np.arange(start = X_set[:,1].min()-1,
plt.contourf(X1,X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape), alpha=0.75, cmap = ListedColormap(("red
plt.ylim(X2.min(), X2.max())for i,j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set==j,0],X_set[y_set==j,1], c = ListedColormap(("red","green"))(i),label = j)
plt.title("Decision Tree(Test set)")
plt.xlabel("Age")
plt.ylabel("Estimated Salary")
plt.legend()
plt.show()
```



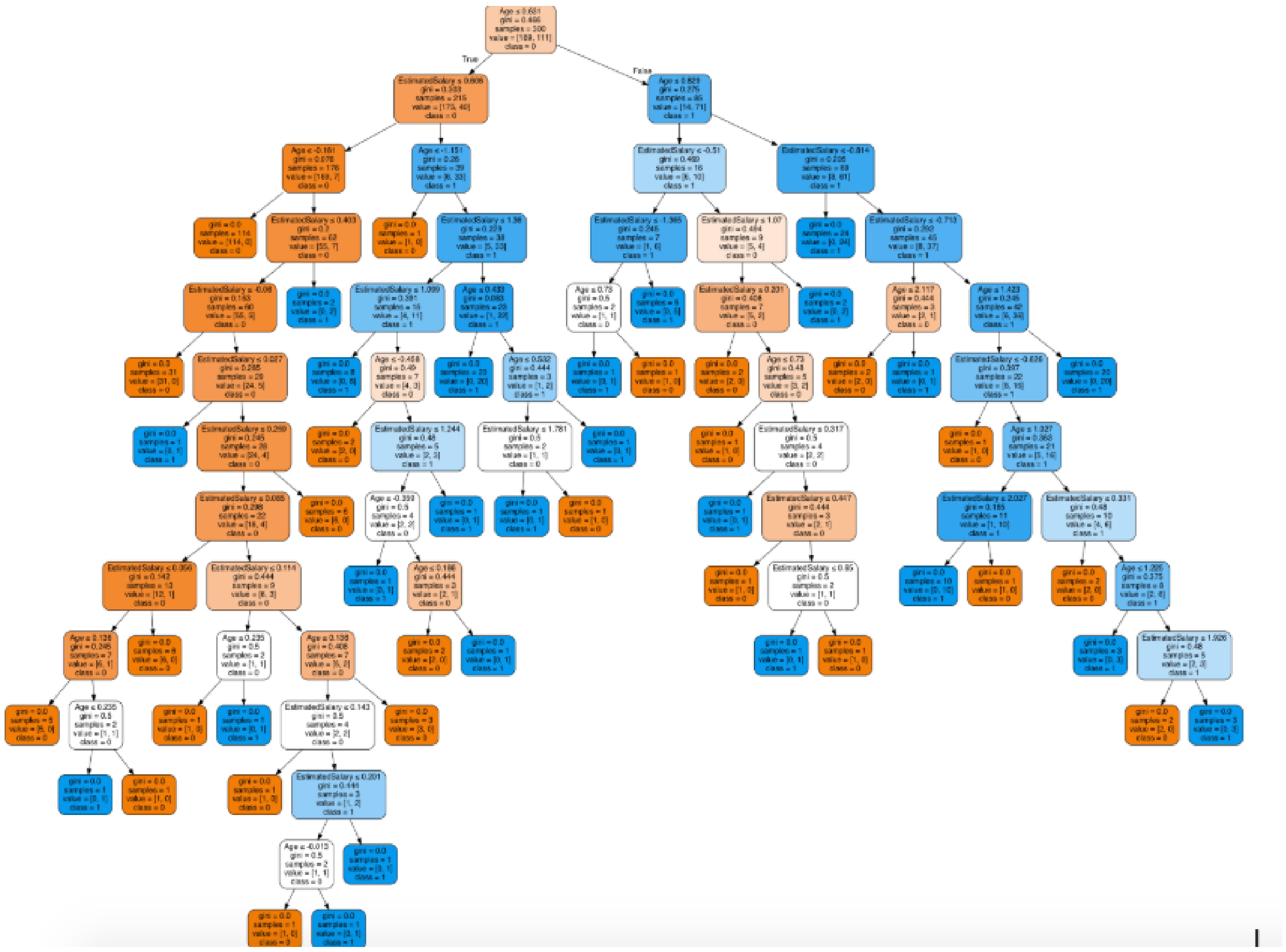
Let us also visualize the tree:

You can use Scikit-learn's *export_graphviz* function to display the tree within a Jupyter notebook. For plotting trees, you also need to install Graphviz and pydotplus.

```
conda install python-graphviz
pip install pydotplus
```

export_graphviz function converts decision tree classifier into dot file and pydotplus convert this dot file to png or displayable form on Jupyter.

```
from sklearn.tree import export_graphviz
from sklearn.externals.six import StringIO
from IPython.display import Image
import pydotplusdot_data = StringIO()
export_graphviz(classifier, out_file=dot_data,
                filled=True, rounded=True,
                special_characters=True,feature_names = feature_cols,class_names=['0','1'])
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
Image(graph.create_png())
```


Decision Tree.

In the decision tree chart, each internal node has a decision rule that splits the data. Gini referred to as the Gini ratio, which measures the impurity of the node. You can say a node is pure when all of its records belong to the same class, such nodes known as the leaf node.

Here, the resultant tree is unpruned. This unpruned tree is unexplainable and not easy to understand. In the next section, let’s optimize it by pruning.

Optimizing the Decision Tree Classifier

criterion: optional (default=”gini”) or Choose attribute selection measure: This parameter allows us to use the different-different attribute selection measure. Supported criteria are “gini” for the Gini index and “entropy” for the information gain.

splitter: string, optional (default=”best”) or Split Strategy: This parameter allows us to choose the split strategy. Supported strategies are “best” to choose the best split and “random” to choose the best random split.

max_depth: int or None, optional (default=None) or Maximum Depth of a Tree: The maximum depth of the tree. If None, then nodes are expanded until all the leaves contain less than min_samples_split samples. The higher value of maximum depth causes overfitting, and a lower value causes underfitting (Source).

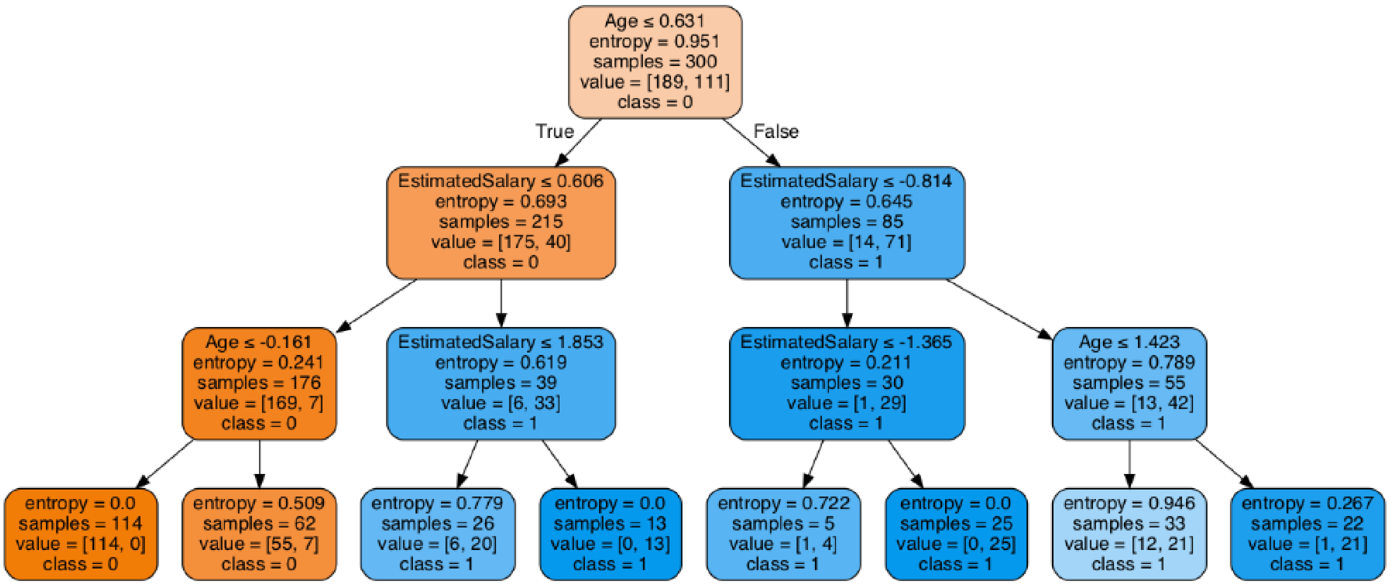
In Scikit-learn, optimization of decision tree classifier performed by only pre-pruning. The maximum depth of the tree can be used as a control variable for pre-pruning.

```
# Create Decision Tree classifer object
classifier = DecisionTreeClassifier(criterion="entropy", max_depth=3)# Train Decision Tree Classifer
classifier = classifier.fit(X_train,y_train)#Predict the response for test dataset
y_pred = classifier.predict(X_test)# Model Accuracy, how often is the classifier correct?
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

Well, the classification rate increased to 94%, which is better accuracy than the previous model.

Now let us again visualize the pruned Decision tree after optimization.

```
dot_data = StringIO()
export_graphviz(classifier, out_file=dot_data,
                filled=True, rounded=True,
                special_characters=True, feature_names = feature_cols,class_names=['0','1'])
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
Image(graph.create_png())
```



Decision Tree after pruning

This pruned model is less complex, explainable, and easy to understand than the previous decision tree model plot.

Conclusion

In this article, we have covered a lot of details about Decision Tree; It's working, attribute selection measures such as Information Gain, Gain Ratio, and Gini Index, decision tree model building, visualization and evaluation on supermarket dataset using Python Scikit-learn package and optimizing Decision Tree performance using parameter tuning.

Well, that's all for this article hope you guys have enjoyed reading it, feel free to share your comments/thoughts/feedback in the comment section.

Source