An Exhaustive Guide to Decision Tree Classification

An End-to-End Tutorial for Classification Using Decision Tree

Ashwin Raj
Do Cool Things, That Matter!

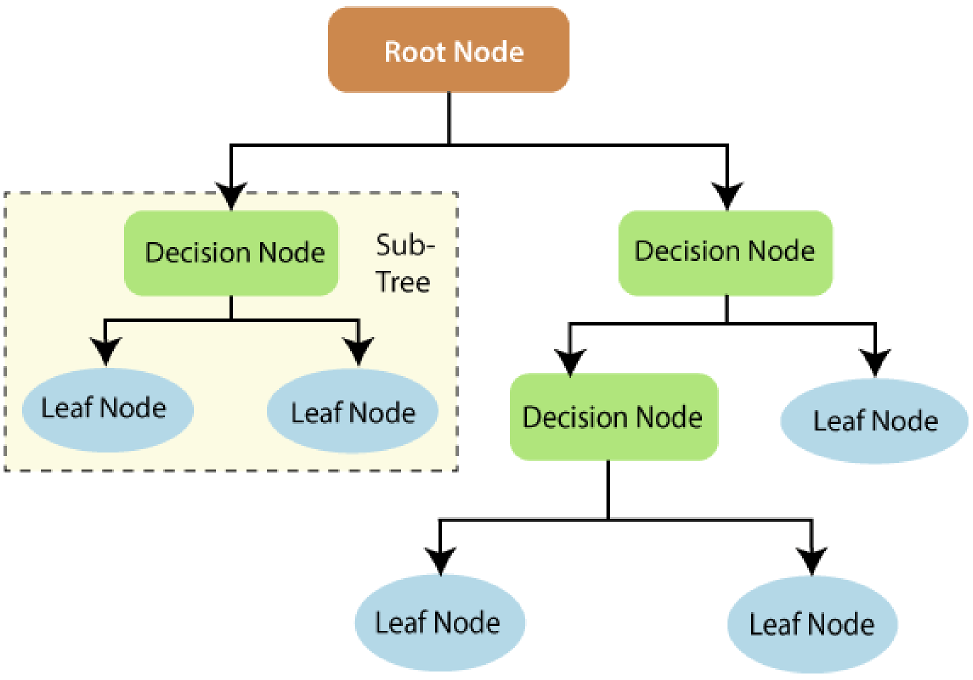
There are various machine learning algorithms that can be put into use for dealing with classification problems. One such algorithm is the Decision Tree algorithm, that apart from classification can also be used for solving regression problems. Though considered to be one of the simplest classification algorithms, if its parameters are tuned properly can yield incredibly accurate predictions.

In this article, I will first try to give you an intuition of what the algorithm is, and how it makes predictions. Then I will try to break down certain important terms in affinity with this algorithm and finally by the end of this article, I will be designing our very own classifier using decision trees.

Before starting this article, I would recommend you to take a look at my previous articles where I have discussed various learning algorithms. Also if you like this article, do consider clapping for this article, and make sure to follow me for more #MachineLearningRecipes.

How are Decisions Made in a Decision Tree?

As I have discussed in my previous articles, the Decision tree algorithm is a simple yet efficient supervised learning algorithm wherein the data points are continuously split according to certain parameters and/or the problem that the algorithm is trying to solve. Decision trees are also popularly referred to as CART (that stands for Classification and Regression Trees)



Every decision tree includes a root node, some branches, and leaf nodes. The internal nodes present within the tree describe the various test cases. Decision Trees can be used to solve both classification and regression problems. The algorithm can be thought of as a graphical tree-like structure that uses various tuned parameters to predict the results. The decision trees apply a top-down approach to the data that it is fed during training.

To understand how the algorithm actually works, Assume that a predictive model needs to be developed that can predict if a student's application for securing admission into a particular course gets accepted or not. Consider the following set of data that is provided to the predictive model.

Student Id Number	Marks in Class 10	Score in class 12	Score in Graduation	Score in GATE	Work Experience	Admission Result
Std01	85	75	55	70	No	Yes
Std02	70	80	75	40	Yes	No
Std03	80	60	65	75	No	No
Std04	65	55	70	60	Yes	Yes

An application from a particular student will be accepted for the course at the university only if it satisfies the conditions that are as described below:

- Score in the GATE examination shall be equal to or more than 60.
- Marks in Graduation, Class 10, and Class 12 shall be more than 60.

In this case, there may be certain exceptions to the aforementioned conditions. In such conditions, the application will be put on a waiting list.

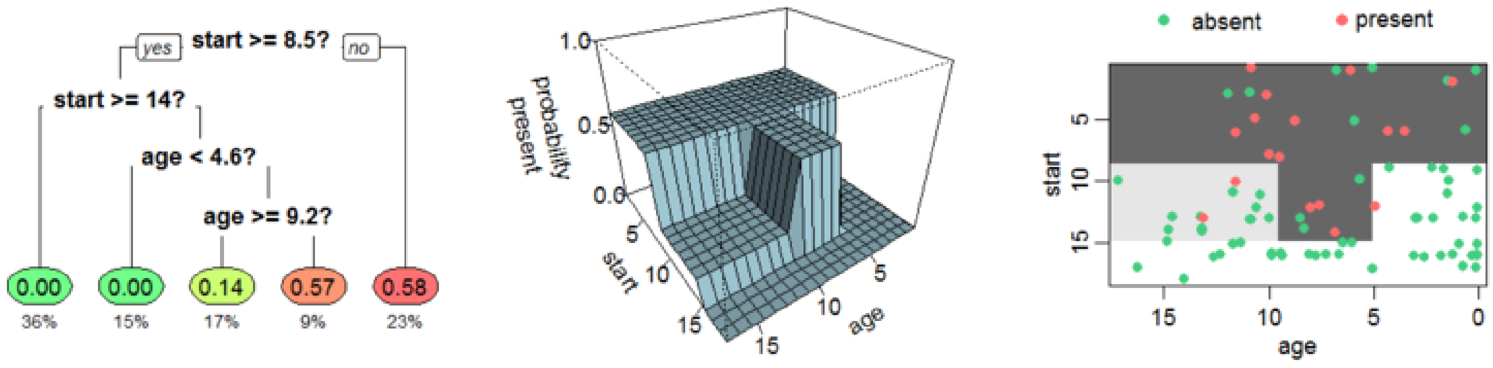
- If the applicant has less than the threshold score in GATE/Graduation but has work experience then the application shall be put on the waiting list
- If the applicant has a score more than 75 in Class 12 but less than the min required score in class 10 then their application shall be on the waiting list.

The problem considered in the above example can be considered in the graphical form as a decision tree or a flow chart. A tree would satisfy all the possible situations that are provided in the problem. The decision tree algorithm works like a bunch of nested if-else statements wherein successive conditions are checked unless the model reaches a conclusion.

The decision nodes or simply nodes of the tree are the questions that are presented by the tree after passing each node(starting from the root node). A branch or sub-tree is a subsection of the entire tree. Each edge of the tree corresponds to the outcome of the question and the outcome is represented by a leaf node or a terminal node which represents the class distribution.

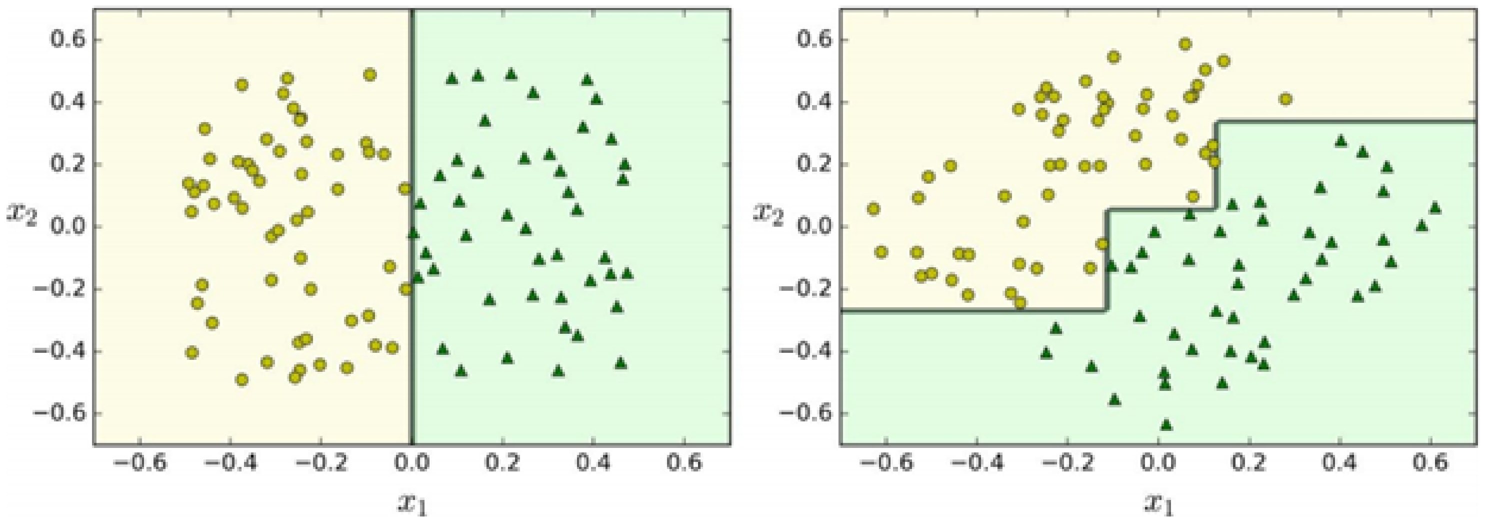
How are Decision Trees used in Classification?

The Decision Tree algorithm uses a data structure called a tree to predict the outcome of a particular problem. Since the decision tree follows a supervised approach, the algorithm is fed with a collection of pre-processed data. This data is used to train the algorithm. Learn more about this here.



Decision trees follow a top-down approach meaning that the root node of the tree is always at the top of the structure while the outcomes are represented by the tree leaves. Decision trees are built using a heuristic called recursive partitioning (commonly referred to as Divide and Conquer). Each node following the root node is split into several nodes.

The key idea is to use a decision tree to partition the data space into dense regions and sparse regions. The splitting of a binary tree can either be binary or multiway. The algorithm keeps on splitting the tree until the data is sufficiently homogeneous. At the end of the training, a decision tree is returned that can be used to make optimal categorized predictions.



An important term in the development of this algorithm is Entropy. It can be considered as the measure of uncertainty of a given dataset and its value describes the degree of randomness of a particular node. Such a situation occurs when the margin of difference for a result is very low and the model thereby doesn't have confidence in the accuracy of the prediction.

The higher the entropy, the higher will be the randomness in the dataset. While building a decision tree, a lower entropy shall be preferred The expression for calculating the entropy of a decision tree is as described:

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

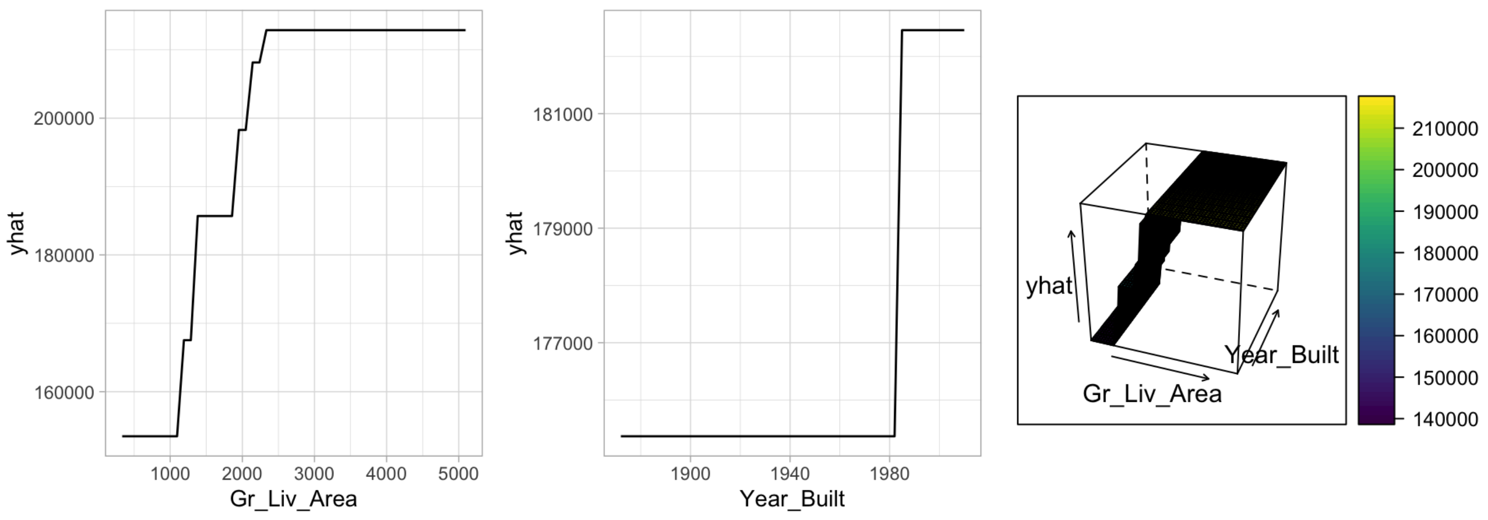
Another metric used for a similar purpose is the Gini Index. It uses the Gini method to create split points. Information Gain is the metric that is generally used for measuring the reduction of uncertainty in the dataset. Information gain in decision trees is generally described by the formulae:

$$IG(D_p, f) = I(D_p) - \frac{N_{left}}{N} I(D_{left}) - \frac{N_{right}}{N} I(D_{right})$$

This metric can further be used to determine the root node of the decision tree and the number of splits that are to be made. The root node of a decision tree is often referred to as the decision node. Each tree has only one root node. The root node is often considered the most important feature in rapport with all other features. Generally, the feature with the highest accuracy amongst all others is chosen as the root node.

Splits in a Decision Tree

As the number of splits increases in a decision tree, the time required to build the tree also increases. Trees with a large number of splits are however prone to overfitting resulting in poor accuracy. This can however be managed by deciding an optimal value for the max_depth parameter. As the value of this parameter increases, the number of splits also increases.



Other parameters that can be used to control the splitting of a decision tree include min_samples_split, min_samples_leaf, and max_features.

Another method by which over-fitting can be avoided to a great extent is by removing branches that have little or no significance in the decision-making process. This is referred to as Pruning. There are two different types of pruning — pre-pruning and post-pruning.

Now that I have provided a better understanding of the theoretical concepts surrounding the algorithm and its working, we shall try applying our knowledge to build our very own classifier. The code and other resources that are used to build the classifier are available in my Github.

Step 1: Importing the Required Libraries and Datasets

Libraries are a set of useful functions that eliminate the need for writing codes from scratch and play a vital role in developing machine learning models and other applications. Python offers a wide array of libraries that can be leveraged to develop highly sophisticated learning models.

```
import numpy as np
import pandas as pd
import seaborn as sns

import matplotlib.pyplot as plt
%matplotlib inline
```

To start with, we shall import popular libraries such as Pandas and NumPy. Pandas is a fast, powerful, flexible, and easy-to-use open-source data analysis and manipulation tool, built on top of the Python programming language. NumPy on the other hand consists of a collection of multi-dimensional array objects and routines for processing those NumPy arrays.

Unlike C/C++, Python gives us the flexibility to import libraries as we move forward in the program. Matplotlib is a multi-platform data visualization library built on NumPy, designed to work with SciPy.

Another library that fulfills similar needs is the Seaborn library that is built on top of matplotlib and is closely integrated with pandas data structures. Visualization is the central part of Seaborn which helps in the exploration and understanding of data and exploratory data analysis and insight study.

```
import warnings

warnings.filterwarnings('ignore')data = 'car_evaluation.csv'
df = pd.read_csv(data, header=None)
```

Once all these libraries have been imported, the next step is to fetch the dataset required to train and test the predictive model. read_csv() method is used to load the dataset into a python file/notebook. The dataset used for this model can be downloaded from here.

Step 2: Exploratory Data Analysis and Feature Engineering

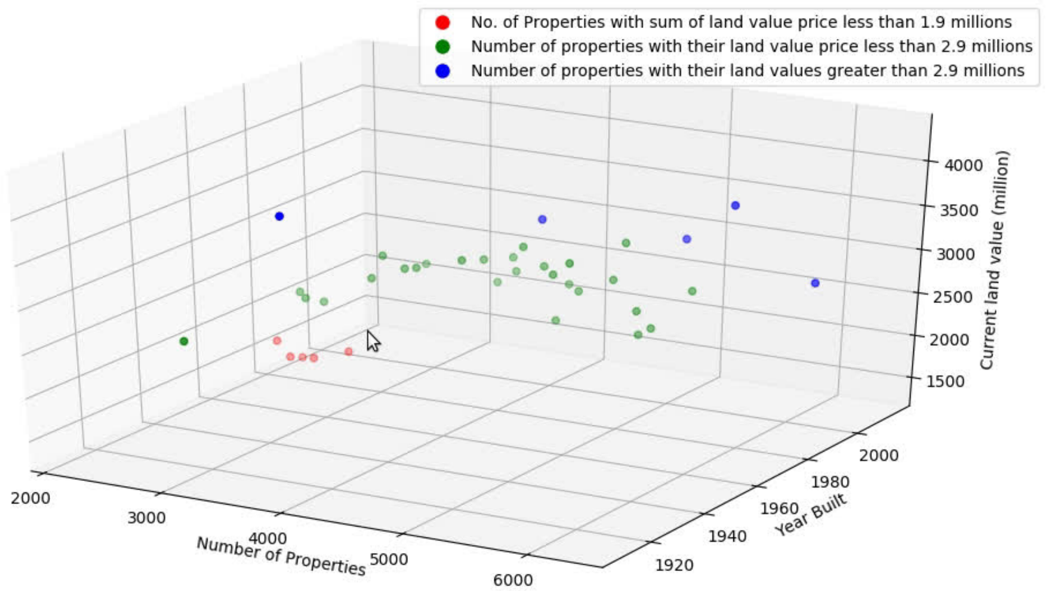
After we have loaded the data into a pandas data frame, the next step in developing the model is the exploratory data analysis. Exploratory data analysis is an approach to analyzing data so as to gain insights into the hidden facts & patterns within the data that are often not visible to us.

This step also includes cleaning and preprocessing the data. During this step, we get an insight into the type of data that we'll be working on. In this step, we also make some changes in the data such as removing missing values, dropping certain columns, reviewing fields, exploring certain variables and defining the relation between various different variables.

```
    from sklearn.model_selection import train_test_split
     X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.33, random_state = 42)encoder
=  ce.OrdinalEncoder(cols=['buying', 'maint', 'doors', 'persons', 'lug_boot', 'safety'])
    X_train = encoder.fit_transform(X_train)

    X_test = encoder.transform(X_test)
```

Once the data has been properly pre-processed, the next step will be to split the data into training and testing data. 70–80% of the data is usually taken as the training data, while the remaining data is taken as the test data. Sometimes the test data is further classified into an additional segment referred to as the validation data, used for the evaluation of the model.



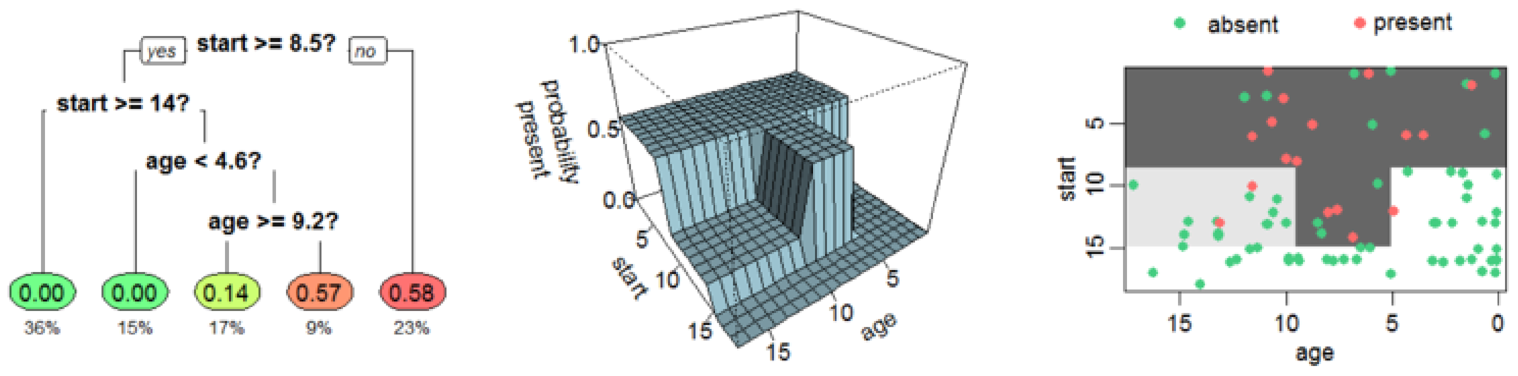
Our final step before training the model is Feature Engineering. It is the process of transforming raw data into useful features that reveal useful insights about the model and thus, increasing its predictive power. During this step, the categorical values are encoded and other suitable changes are made to the data. By the end of this step, the predictive model is ready.

Step 3: Fitting the Model, Evaluating Result & Visualizing Trees

Now that the data is totally prepared, the classifier is instantiated and the model is fit onto the data. The criterion chosen for this classifier is entropy, though the Gini index can also be used. Once our model fits the data, we try predicting values using the classifier model. This is done in order to perform an unbiased evaluation and get the accuracy score of the model.

```
       X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.33, random_state = 42)from
sklearn.tree import DecisionTreeClassifier
    clf_en = DecisionTreeClassifier(criterion='entropy', max_depth=3, random_state=0)
    clf_en.fit(X_train, y_train)y_pred_en = clf_en.predict(X_test)
```

It shall be ensured that the model is neither overfitting nor underfitting the data. This can be done by calculating the accuracy score of both the train and test data. If the values are comparable then the model isn't overfitting.



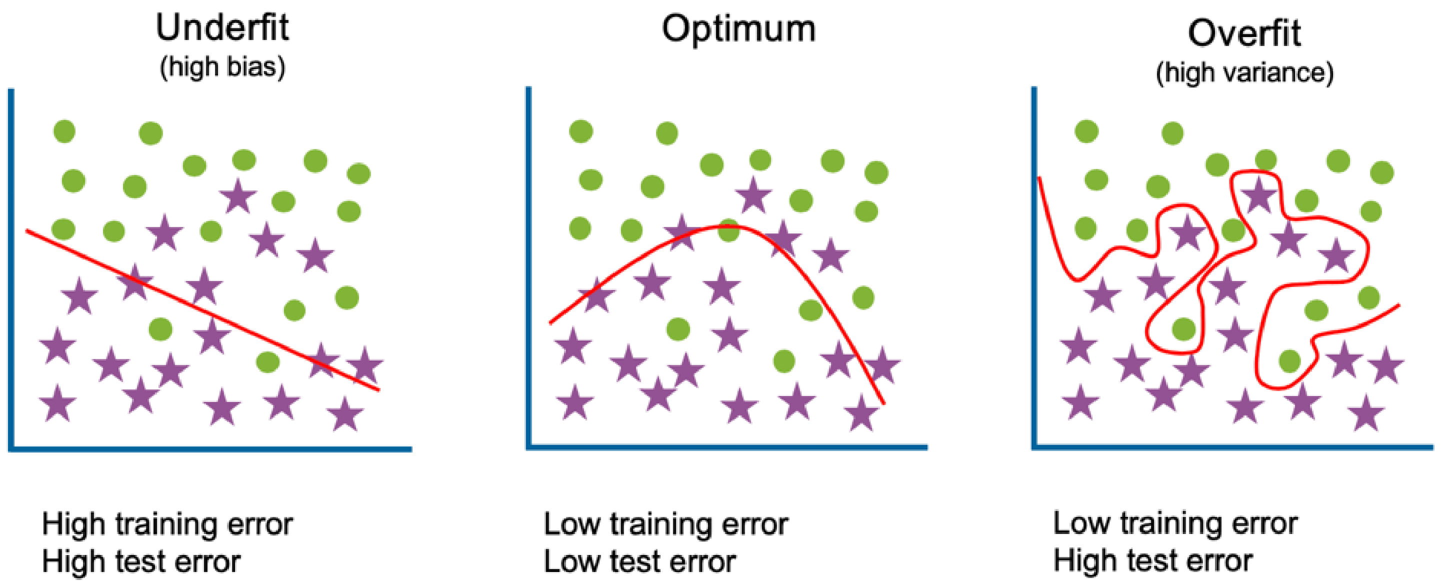
Once the model is fit over the data and the predictions are made, our final step would be to evaluate the classifier. One of the most popular tools for this method to do is to calculate the confusion matrix of the classifier. A confusion matrix is a tool for summarizing the performance of the model.

```
   from sklearn.metrics import accuracy_score
   print('Model accuracy score with criterion entropy: {0:0.4f}'. format(accuracy_score(y_test, y_pred_en)))

   y_pred_train_en = clf_en.predict(X_train)
   y_pred_train_enplt.figure(figsize=(12,8))

   from sklearn import tree
   tree.plot_tree(clf_en.fit(X_train, y_train))from sklearn.metrics import confusion_matrix
   cm = confusion_matrix(y_test, y_pred_en)
   print('Confusion matrix\n\n', cm)
```

The Confusion Matrix gives us a complete summary of correct and incorrect predictions broken down by each category, throwing light on the performance of the classification models and the errors made.



A confusion matrix has four possible outcomes namely True Positive, True Negative, False Positive, and False Negative. The confusion matrix is usually presented as an output in a tabular form. Other evaluation techniques in AI/ML include precision score, f1 score, recall, and support scores.

Advantages of Decision Tree Algorithm

Decision Tree classifiers are amongst the most widely used predictive algorithms for classification.

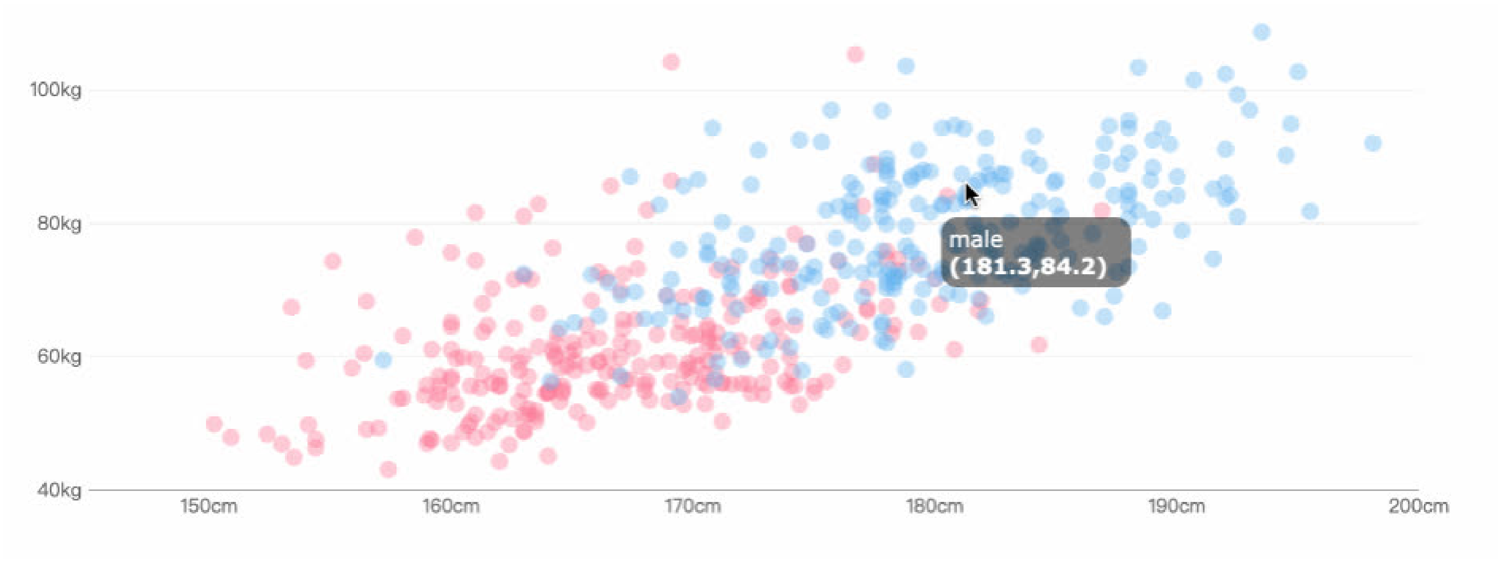
Some features that make it so popular are:

- Extremely fast classification of unknown records
- Disregards features that are of little or no importance in prediction.
- Extremely efficient, provided the parameters are tuned optimally.
- Inexpensive to construct with an easy to interpret logic.

Limitations of Decision Tree Algorithm

Though the Decision Tree classifier is one of the most sophisticated classification algorithms, it may have certain limitations, especially in real-world scenarios. Some of its deterrents are as below:

- Decision Tree Classifiers tends to overfit on the data
- Changes in data may lead to unnecessary changes in the result
- Large trees can be difficult to interpret
- These are biased toward splits on features having a number of levels.



Since a Decision tree classifier tends to overfit in many cases, it is advantageous to replace a Decision Tree classifier with Principal Component Analysis for datasets with a large number of features.

Applications of Decision Tree Classifiers

Having discussed the advantages & limitations of the Decision Tree algorithm, it's time to shed some light on the application of Decision Tree classifiers. One of the most popular uses of the Decision Tree algorithm is in Biomedical Engineering, wherein it is used for identifying features that can be used in implantable devices and for exploring potential medicines.

Decision tree classifiers also find their use in DA, financial analysis, and economic product development wherein they are used to understand the customer satisfaction level, business, and related behavior.

Amongst other applications, Decision tree classifiers are also used in system design and Physics, especially in particle detection. It also finds its application in Manufacturing and Production wherein it can be put in quality control, anomaly detection, and semiconductor manufacturing.

With that, we have reached the end of this article. I hope this article would have helped you get a hunch of how the Decision Tree classifier works. If you have any questions or if you believe I have made any mistake, feel free to contact me. Get in touch with me via EMail or LinkedIn. Happy Learning!