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 problem
- 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
- 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.

Types of Decision Trees:

1. Categorical variable decision tree

A categorical variable decision tree includes categorical target variables that are divided into categories. For example, the categories can be yes or no. The categories mean that every stage of the decision process falls into one category, and there are no in-betweens.

2. Continuous variable decision tree

A continuous variable decision tree is a decision tree with a continuous target variable. For example, the income of an individual whose income is unknown can be predicted based on available information such as their occupation, age, and other continuous variables.

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.
- 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.

Algorithm:

The complete process can be better understood using the below algorithm:

Step-1: Begin the 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).

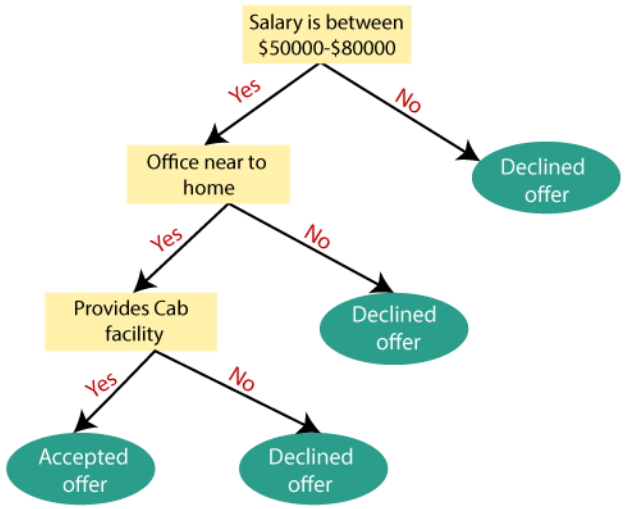
Step-3: Divide the S into subsets that contains possible values for the best attributes.

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



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.

Measuring consistency in a decision tree:

Measuring consistency in machine learning refers to the extent to which a decision tree (or any other model) produces consistent results when applied to different subsets of the same dataset or to new, unseen data.

In the context of decision trees, consistency is important because a decision tree that is consistent will provide reliable predictions for new data. If a decision tree is inconsistent, it may produce

conflicting or unreliable predictions, which can lead to poor performance on new data.

There are some techniques for measuring consistency somee of them are as follows:

Gini impurity:

The division is called pure if all elements are accurately separated into different classes.

The Gini impurity is used to predict that a randomly selected example would be incorrectly classified by a specific node.

The degree of Gini impurity ranges from 0 to 1

An attribute with the low Gini index should be preferred as compared to the high Gini index.

Formula:

Gini = 1- ∑jPj2

Information gain:

The term "information gain" refers to the process of selecting the best features/attributes that provide the most information about a class.

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.

Formula:

Information Gain = 1 – Entropy

Entropy: Entropy is a metric to measure the impurity in a given attribute.

Decision tree using recursion:

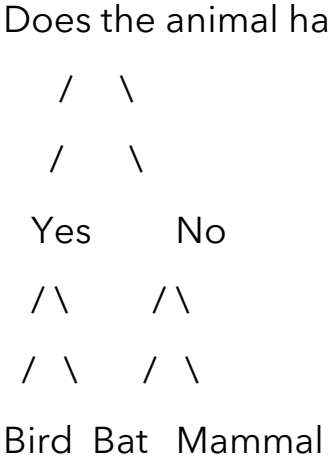
- Decision tree algorithms often use recursion to construct the decision tree
- Recursion is a powerful tool for decision tree algorithms because it allows the algorithm to handle datasets with a large number of features and to construct complex decision boundaries that can capture non-linear relationships between features.
- The algorithm starts by evaluating the entire dataset, and then chooses the best split to divide the data into two subsets.
- At each recursive step, the algorithm needs to evaluate the best split for the subset of data.
- The same process is then applied to each subset, and continues recursively until some stopping criteria is met
- It can also lead to overfitting if the stopping criteria are not chosen carefully

Constructing the decision tree using recursion

To construct a decision tree using recursion, we can follow these general steps:

1. Define a recursive function that takes in a dataset and a set of criteria or rules to split the data.
2. Within the function, check if the criteria for splitting the data are met. If not, return a leaf node with a default decision or prediction.
3. If the criteria are met, split the dataset into smaller subsets based on the criteria.
4. Call the recursive function on each subset of data, passing in the new criteria for each subset.
5. Repeat steps 2-4 until a stopping condition is met

Let's take an example of a decision tree for classifying animals based on their characteristics. The tree starts with a root node that represents the first decision, which is whether the animal has feathers or not. If the animal has feathers, it goes down the left branch, and if it doesn't have feathers, it goes down the right branch. The tree continues to split based on different features until it reaches the final decision.



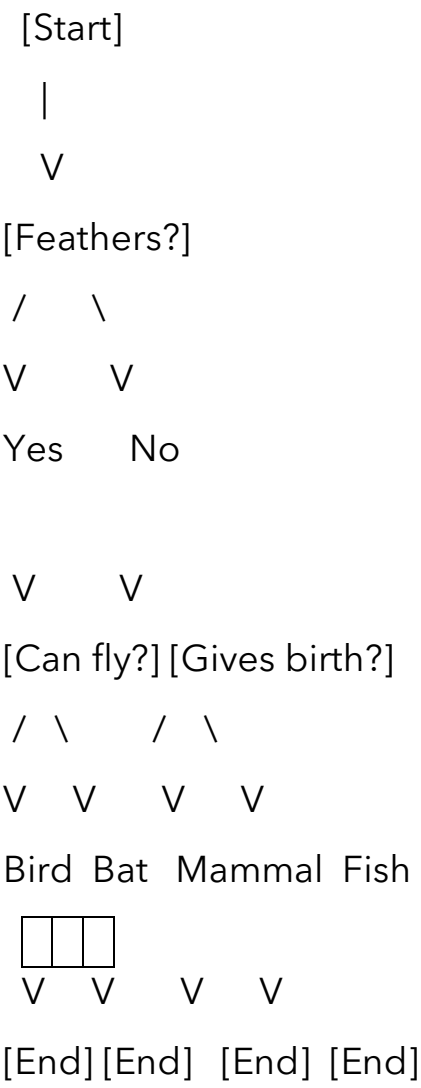
To implement this decision tree using recursion, we can define a function that takes in the animal's features as input and returns the predicted class as output. The function checks the value of the current feature and decides which branch of the tree to follow. It then calls itself recursively with the new set of features until it reaches a leaf node, which represents the final decision.

Here Is the Python code for the recursive function:

```
Def classify_animal(features):
    If features['feathers'] == 'yes':
        If features['can_fly'] == 'yes':
            Return 'bird'
        Else:
            Return 'bat'
    Else:
        If features['gives_birth'] == 'yes':
            Return 'mammal'
        Else:
            Return 'fish'
```

The function takes in a dictionary of features and checks the value of the 'feathers' feature to decide which branch to follow. If the animal has feathers, it checks the value of the 'can_fly' feature to decide whether it is a bird or a bat. If the animal doesn't have feathers, it checks the value of the 'gives_birth' feature to decide whether it is a mammal or a fish.

Here is the flowchart of the decision tree:



Code in python:

```
def create_decision_tree(data):
    # Base case: if all labels are the same, return a leaf node with that label
```

```
    if len(set([d[-1] for d in data])) == 1:
        return data[0][-1]

    # Recursive case: find the best feature to split the data and create child nodes
    best_feature = find_best_feature(data)
    tree = {best_feature: {}}
    for value in unique_values(data, best_feature):
        subset = get_subset(data, best_feature, value)
        tree[best_feature][value] = create_decision_tree(subset)

    return tree
```

Plotting decision trees using matplotlib:

To plot a decision tree using matplotlib, you can use the sklearn.tree library to create a visualization of the tree, and then use matplotlib.pyplot to plot the tree.

Sklearn.tree:

sklearn.tree is a module in the Scikit-learn library that provides classes for constructing and analyzing decision trees. Some of the key classes in **sklearn.tree** include:

- **DecisionTreeClassifier**: A class for constructing a decision tree classifier, which can be used for classification tasks.
- **plot_tree**: A function for visualizing a decision tree using matplotlib

Matplotlib.pyplot:

matplotlib.pyplot is a module in the Matplotlib library that provides a collection of functions for creating a wide variety of visualizations, including decision trees. In particular, the sklearn.tree library provides a function called plot_tree that uses matplotlib.pyplot to plot a decision tree.

Here are the steps:

Import the necessary libraries:

```
import matplotlib.pyplot as plt
from sklearn.tree import DecisionTreeClassifier, plot_tree
```

Create an instance of the DecisionTreeClassifier class from sklearn.tree and fit the model to your data:

```
clf = DecisionTreeClassifier()
clf.fit(X, y)
```

where **X** is your feature matrix and **y** is your target vector.

Use the plot_tree function to plot the decision tree:

```
fig, ax = plt.subplots(figsize=(12, 12))
plot_tree(clf, ax=ax)
plt.show()
```

You can customize the appearance of the plot using the figsize parameter to adjust the size of the plot.

Here's an example code to create and display a decision tree:

```
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier, plot_tree

# Load the iris dataset
iris = load_iris()
X = iris.data[:, :2]
y = iris.target
```

```
# Create a decision tree
clf = DecisionTreeClassifier()
clf.fit(X, y)

# Plot the decision tree
fig, ax = plt.subplots(figsize=(12, 12))
plot_tree(clf, ax=ax)
plt.show()
```