### Decision Tree in Machine Learning

# Introduction

For this assignment, I chose Decision Tree Classification as the topic. In this report, I will discuss the working of decision trees in detail with respect to their theoretical background and then demonstrate the working of decision trees on a given data set using programming language Python from the scratch.

Decision Tree is a tree like structure used in machine learning to classify as well as regression. It divides data using several decision rules that are made depending on feature values leading to an output (Costa et al., 2023). Internal nodes are decision nodes and branches are outcomes and the end nodes are the classification or prediction. This decision-making process is sequential, resembling human thinking hence decision trees are easy to interpret.

Decision trees are especially useful where it is important to explain why a particular decision has been made (Kotsiantis, 2011). When performing the Decision Tree Classifier, I used the Adult dataset which is one of the most popular datasets in the machine learning domain. This dataset is obtained from the UCI Machine Learning Repository and includes demographic data and income level, with target variable of whether the income is more than $50K per year.

The present work is devoted to the description of the theoretical background of decision trees and such concepts as splitting criteria, recursive construction of the tree, and processes of the prediction. Further, I illustrate their usage on a practical dataset with a Python implementation of the algorithm.

# Decision Tree Workflow

A decision tree is constructed through a step-by-step process that involves selecting the best splits at each node to create a hierarchical structure of decisions. The key steps in constructing a decision tree are as follows:

* **Evaluate Node Impurity**: Compute the impurity of the current node to assess its class heterogeneity.
* **Find the Best Split**: For every feature and its possible values, calculate the reduction in impurity achieved by splitting the node.
* **Split the Node**: Choose the split with the lowest impurity and divide the data into left and right child nodes.
* **Recursively Apply the Process**: Repeat the above steps for the child nodes until a stopping condition, such as max depth or minimum sample size, is met.
* **Assign Terminal Values**: When a node cannot be split further, assign it a terminal class based on the majority label.

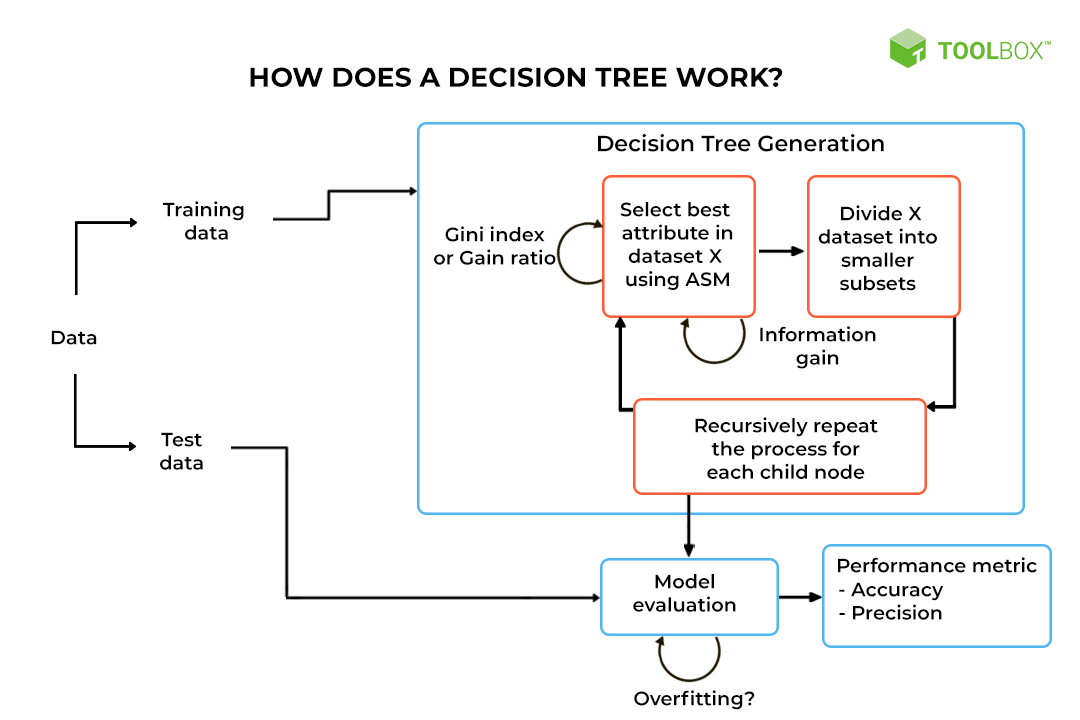


Figure : Decision Tree Working

Next section delves into the details of impurity calculation, splitting criteria, and how the optimal split is selected, with a focus on Gini Impurity as the splitting metric.

# Gini Impurity and Splitting

Splitting is the cornerstone of decision tree construction. It divides a dataset into subsets by finding the most meaningful splits in the feature space, thereby maximizing the separation of different class labels (Costa and Pedreira, 2022). The objective of splitting is to create child nodes that are purer than the parent node. A pure node contains data from a single class, whereas an impure node has a mix of class labels.

To achieve this, the algorithm uses a splitting metric to evaluate the quality of each possible split. The metric quantifies how "pure" the resulting child nodes are. One common metric is Gini Impurity, which minimizes impurity at each split.

## Gini Impurity Calculation

Gini Impurity measures the likelihood of misclassification at a node. The formula for Gini Impurity is:

**Where**:

* G: Gini Impurity.
* pi​: Proportion of class ii instances at the node.
* n: Total number of classes.

A node with only one class has G=0G=0, meaning it is pure. Higher values of GG indicate higher impurity.

**Example**: Consider a node with 10 samples:

* 7 samples belong to class 00 (p0=7/10=0.7).
* 3 samples belong to class 11 (p1=3/10=0.3).

The Gini Impurity is:

*G*=1−(0.72+0.32) = 1−(0.49+0.09) = 0.42

This node is moderately impure because it contains samples from both classes.

## Splitting and Weighted Gini ****Impurity****

When a node is split, the Gini Impurity of the child nodes is weighted by their sizes to compute the Weighted Gini Impurity of the split:

​

Where:

* nleft, nright​: Sizes of the left and right child nodes.
* N: Total size of the parent node.

**Example**: Split the above node into two child nodes:

* Left node: [<=50K, <=50K] (nleft=2, Gleft=0).
* Right node: [<=50K, >50K, >50K] (nright=3, Gright=0.44).

Gsplit = 2/5⋅0 + 3/5⋅0.44 = 0.264

Since Gsplit ​=0.264 is lower than G=0.48 of the parent node, the split improves purity.

## Optimal Split Selection

The algorithm selects the split with the lowest Gsplit​value by evaluating all features and their unique values. This process involves:

* Calculating Gini Impurity for each possible split.
* Selecting the split that minimizes Gsplit.

**Computational Complexity:** The process requires evaluating O(m⋅v) splits, where:

* m: Number of features.
* v: Average number of unique values per feature.

# Alternatives to Gini Impurity

## Entropy

Entropy is another widely used metric to evaluate the impurity of a node. It measures the level of randomness or disorder in the data. A pure node, where all samples belong to the same class, has an entropy of 0, while a node with a uniform distribution of classes has maximum entropy.

The formula for entropy is:

Where:

* E: Entropy of the node.
* pi​: Proportion of samples belonging to class ii in the node.
* n: Total number of classes.

## Information Gain (IG)

Information Gain is the reduction in entropy achieved by splitting a node into child nodes. It quantifies how much the split has improved the purity of the data. The formula for Information Gain is:

Where:

* IG: Information Gain.
* Eparent​: Entropy of the parent node.
* Eleft, Eright​: Entropy of the left and right child nodes.
* nleft, nright​: Number of samples in the left and right child nodes.
* N: Total number of samples in the parent node.

## When to Use Entropy and Information Gain

Use Entropy and Information Gain when a probabilistic interpretation of impurity is needed, as entropy directly quantifies the uncertainty in the dataset.

It is often used in datasets with categorical variables where probabilistic outcomes are more intuitive.

Table : Comparison with Gini Impurity

|  |  |
| --- | --- |
| Metric | Characteristics |
| Gini Impurity | Easier to compute, often leads to similar splits as IG. |
| Entropy | Provides probabilistic interpretation of impurity. |
| Information Gain | Measures entropy reduction after a split. |

Gini Impurity is computationally faster and works well for large datasets, while entropy provides deeper insights into probabilistic distributions but is slightly more computationally intensive. Choose based on interpretability needs and dataset size.

# Implementation of Decision Tree Classifier

The Decision Tree Classifier (DTR) was implemented from scratch in Python, allowing for a deep understanding of how the algorithm operates under the hood. The process involved creating a class-based implementation that encapsulates the core functionalities of a decision tree, from splitting nodes to making predictions and visualizing the tree structure. Here's a detailed breakdown of the implementation process:

**1. Initialization**

The classifier was initialized with two key parameters:

* max\_depth: The maximum depth of the tree to control its complexity and prevent overfitting.
* min\_size: The minimum number of samples required to form a terminal node, ensuring meaningful splits.

These parameters provide flexibility to balance model complexity and performance.

**2. Gini Impurity Calculation**

The Gini Impurity metric was implemented to measure the impurity at each node. The following steps were followed:

* For a given split, the child nodes were analyzed to calculate their impurity.
* The impurity was weighted by the size of each child node relative to the parent node.
* The weighted Gini Impurity was returned, providing a quantitative measure of split quality.

**3. Splitting the Data**

To split the data:

* For categorical features, data was divided based on equality conditions (e.g., workclass == Private).
* For numerical features, splits were based on thresholds (e.g., age < 35).
* The test\_split method iterated over all feature-value combinations, creating left and right child nodes for evaluation.

**4. Finding the Best Split**

The get\_best\_split method iteratively evaluated all possible splits by:

* Calculating Gini Impurity for each potential split.
* Selecting the split with the lowest Gini Impurity as the optimal division for the current node.

This process ensured that each node division maximized purity and minimized misclassification.

**5. Recursive Node Splitting**

Using recursion, the dataset was repeatedly divided into smaller subsets:

1. **Base Cases**: The recursion terminated when one of the following conditions was met:
   * The maximum depth (max\_depth) was reached.
   * The number of samples in a node was below min\_size.
   * The node achieved complete purity (Gini Impurity = 0).
2. **Recursive Calls**: If none of the stopping criteria were met, the node was split further, and the process was repeated for the resulting child nodes.

This recursive approach enabled the tree to grow dynamically, adapting to the dataset's structure.

**6. Terminal Nodes**

For terminal nodes:

* If a node could not be split further, it was assigned a majority class label from its samples.
* This ensured that every branch of the tree ended with a definitive prediction.

**7. Prediction Mechanism**

To make predictions:

* Each sample was passed through the tree, starting at the root node.
* At each internal node, the sample was evaluated against the split condition to determine whether to proceed to the left or right child node.
* The traversal continued until reaching a terminal node, where the class label was returned as the prediction.

This process closely mimicked the step-by-step decision-making flow of the tree.

# Demonstration

## Features Used

The following features were selected from the Adult dataset for the demonstration:

* **Numerical Features:**
  + age: Represents the individual's age.
  + hours-per-week: Number of hours the individual works weekly.
* **Categorical Features:**
  + workclass: The type of employment (e.g., Private, Self-employed).
  + education: Educational qualification (e.g., Bachelors, Masters).
  + marital-status: Marital status (e.g., Married-civ-spouse, Never-married).
  + occupation: Type of occupation (e.g., Prof-specialty, Exec-managerial).
* **Target Variable:**
  + income: Binary classification indicating whether the individual's income is <=50K or >50K.

## Data Preparation

The dataset was prepared by first removing all rows with missing values to ensure data quality. Since the data was imbalanced, an under-sampling technique was applied to balance the class distribution between <=50K and >50K. Relevant numerical and categorical features, such as age, hours-per-week, workclass, and marital-status, were selected for simplicity and interpretability. Finally, a subset of 3000 samples was randomly selected, with 80% allocated for training and 20% for testing.

## Parameter Tuning and Model Implementation

To optimize the decision tree’s performance, hyperparameters such as max\_depth (maximum depth of the tree) and min\_size (minimum samples per leaf) were tuned. A grid search was performed over different values, and the F1 score was calculated for each combination.

|  |  |  |
| --- | --- | --- |
| max\_depth | min\_size | f1\_score |
| 1 | 1 | 0.766021 |
| 1 | 3 | 0.766021 |
| 1 | 5 | 0.766021 |
| 2 | 1 | 0.765507 |
| 2 | 3 | 0.765507 |
| 2 | 5 | 0.765507 |
| 3 | 1 | 0.789773 |
| 3 | 3 | 0.789773 |
| 3 | 5 | 0.789773 |
| 4 | 1 | 0.786227 |
| 4 | 3 | 0.786227 |
| 4 | 5 | 0.786227 |

The best parameters were:

* Max Depth: 3
* Min Samples: 1
* F1 Score: 0.7898

**Evaluation of the Final Decision Tree**

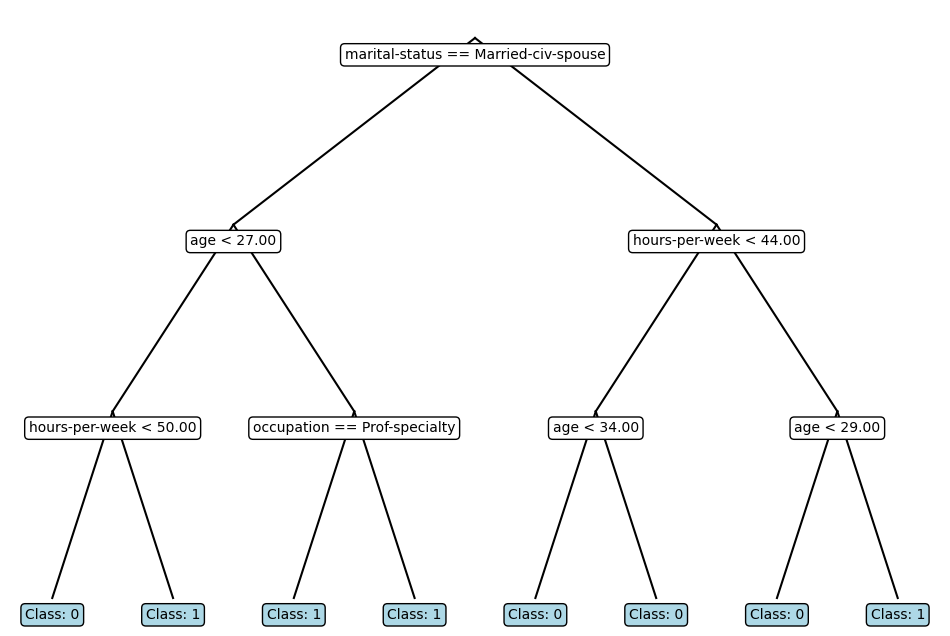
The final tree was trained using the best parameters and evaluated on the test dataset. The following metrics were calculated:

* Accuracy: 0.7533
* Precision: 0.6985
* Recall: 0.9085
* F1 Score: 0.7898

These results indicate that the decision tree performed well in predicting higher-income individuals (>50K), achieving high recall and a balanced F1 score.

**Interpretation of the Final Decision Tree**

The final decision tree is depicted above. Each node represents a decision rule, and the leaf nodes indicate the class predictions. Key decision rules from the tree include:



## Scenarios

1. A young, married individual working part-time (less than 50 hours/week).
2. A young, married individual working overtime (more than 50 hours/week).
3. A married professional with a high-skill job.
4. A married individual working in a non-professional occupation.
5. A young, single individual working fewer hours.
6. An older, single individual working fewer hours.
7. A young, single individual working long hours.
8. An experienced, single individual working long hours.

This comprehensive list shows how the decision tree integrates demographic and work-related features to make income predictions.

# References

Kotsiantis, S.B. (2011) 'Decision trees: a recent overview,' *Artificial Intelligence Review*, 39(4), pp. 261–283. https://doi.org/10.1007/s10462-011-9272-4.

Costa, V.G. *et al.* (2023) 'Evolving interpretable decision trees for reinforcement learning,' *Artificial Intelligence*, 327, p. 104057. https://doi.org/10.1016/j.artint.2023.104057.

Costa, V.G. and Pedreira, C.E. (2022) 'Recent advances in decision trees: an updated survey,' *Artificial Intelligence Review*, 56(5), pp. 4765–4800. https://doi.org/10.1007/s10462-022-10275-5.

Demirović, E. *et al.* (2020) 'MurTree: Optimal Classification Trees via Dynamic Programming and Search,' *arXiv (Cornell University)* [Preprint]. https://doi.org/10.48550/arxiv.2007.12652.