**Basics of decision tree algorithms: construction, splitting criteria (e.g., Gini index, information gain).**

A decision tree is a type of supervised learning algorithm that is commonly used in machine learning to model and predict outcomes based on input data. It is a tree-like structure where each internal node tests on attribute, each branch corresponds to attribute value and each leaf node represents the final decision or prediction. The decision tree algorithm falls under the category of supervised learning. They can be used to solve both regression and classification problems.

**Decision Tree Terminologies**

There are specialized terms associated with decision trees that denote various components and facets of the tree structure and decision-making procedure. :

* Root Node: A decision tree’s root node, which represents the original choice or feature from which the tree branches, is the highest node.
* Internal Nodes (Decision Nodes): Nodes in the tree whose choices are determined by the values of particular attributes. There are branches on these nodes that go to other nodes.
* Leaf Nodes (Terminal Nodes): The branches’ termini, when choices or forecasts are decided upon. There are no more branches on leaf nodes.
* Branches (Edges): Links between nodes that show how decisions are made in response to particular circumstances.
* Splitting: The process of dividing a node into two or more sub-nodes based on a decision criterion. It involves selecting a feature and a threshold to create subsets of data.
* Parent Node: A node that is split into child nodes. The original node from which a split originates.
* Child Node: Nodes created as a result of a split from a parent node.
* Decision Criterion: The rule or condition used to determine how the data should be split at a decision node. It involves comparing feature values against a threshold.
* Pruning: The process of removing branches or nodes from a decision tree to improve its generalisation and prevent overfitting.

Understanding these terminologies is crucial for interpreting and working with decision trees in machine learning applications.

How Decision Tree is formed?

The process of forming a decision tree involves recursively partitioning the data based on the values of different attributes. The algorithm selects the best attribute to split the data at each internal node, based on certain criteria such as information gain or Gini impurity. This splitting process continues until a stopping criterion is met, such as reaching a maximum depth or having a minimum number of instances in a leaf node.

Why Decision Tree?

Decision trees are widely used in machine learning for a number of reasons:

* Decision trees are so versatile in simulating intricate decision-making processes, because of their interpretability and versatility.
* Their portrayal of complex choice scenarios that take into account a variety of causes and outcomes is made possible by their hierarchical structure.
* They provide comprehensible insights into the decision logic, decision trees are especially helpful for tasks involving categorisation and regression.
* They are proficient with both numerical and categorical data, and they can easily adapt to a variety of datasets thanks to their autonomous feature selection capability.
* Decision trees also provide simple visualization, which helps to comprehend and elucidate the underlying decision processes in a model.

Decision Tree Approach

Decision tree uses the tree representation to solve the problem in which each leaf node corresponds to a class label and attributes are represented on the internal node of the tree. We can represent any boolean function on discrete attributes using the decision tree.

A diagram of a person

Description automatically generated

 Below are some assumptions that we made while using the decision tree:

At the beginning, we consider the whole training set as the root.

* Feature values are preferred to be categorical. If the values are continuous then they are discretized prior to building the model.
* On the basis of attribute values, records are distributed recursively.
* We use statistical methods for ordering attributes as root or the internal node.

A screenshot of a computer screen

Description automatically generated

 As you can see from the above image the Decision Tree works on the Sum of Product form which is also known as *Disjunctive Normal Form*. In the above image, we are predicting the use of computer in the daily life of people. In the Decision Tree, the major challenge is the identification of the attribute for the root node at each level. This process is known as attribute selection. We have two popular attribute selection measures:

1. **Information Gain**
2. **Gini Index**

1. Information Gain:

When we use a node in a decision tree to partition the training instances into smaller subsets the entropy changes. Information gain is a measure of this change in entropy.

* Suppose S is a set of instances,
* A is an attribute
* Sv is the subset of S
* *v* represents an individual value that the attribute *A* can take and Values (A) is the set of all possible values of A, then  
   Gain(S,A)=Entropy(S)–∑vA∣Sv∣∣S∣.Entropy(Sv)*Gain*(*S*,*A*)=*Entropy*(*S*)–∑*vA*​∣*S*∣∣*Sv*​∣​.*Entropy*(*Sv*​)

Entropy: is the measure of uncertainty of a random variable, it characterizes the impurity of an arbitrary collection of examples. The higher the entropy more the information content.

Suppose S is a set of instances, A is an attribute, Sv is the subset of S with A = v, and Values (A) is the set of all possible values of A, then

Gain(S,A)=Entropy(S)–∑vϵValues(A)∣Sv∣∣S∣.Entropy(Sv)  *Gain*(*S*,*A*)=*Entropy*(*S*)–∑*vϵValues*(*A*)​∣*S*∣∣*Sv*​∣​.*Entropy*(*Sv*​)

Example:

For the set X = {a,a,a,b,b,b,b,b}  
Total instances: 8  
Instances of b: 5  
Instances of a: 3

Entropy H(X)=[(38)log⁡238+(58)log⁡258]=−[0.375(−1.415)+0.625(−0.678)]=−(−0.53−0.424)=0.954Entropy *H*(*X*)​=[(83​)log2​83​+(85​)log2​85​]=−[0.375(−1.415)+0.625(−0.678)]=−(−0.53−0.424)=0.954​

Building Decision Tree using Information Gain The essentials:

* Start with all training instances associated with the root node
* Use info gain to choose which attribute to label each node with
* *Note:* No root-to-leaf path should contain the same discrete attribute twice
* Recursively construct each subtree on the subset of training instances that would be classified down that path in the tree.
* If all positive or all negative training instances remain, the label that node “yes” or “no” accordingly
* If no attributes remain, label with a majority vote of training instances left at that node
* If no instances remain, label with a majority vote of the parent’s training instances.

Example: Now, let us draw a Decision Tree for the following data using Information gain. Training set: 3 features and 2 classes

| X | Y | Z | C |
| --- | --- | --- | --- |
| 1 | 1 | 1 | I |
| 1 | 1 | 0 | I |
| 0 | 0 | 1 | II |
| 1 | 0 | 0 | II |

Here, we have 3 features and 2 output classes. To build a decision tree using Information gain. We will take each of the features and calculate the information for each feature.A green circles and black lines with white text

Description automatically generated

Split on feature X

A diagram of a mathematical equation

Description automatically generated

Split on feature Y

A diagram of a mathematical equation

Description automatically generated

Split on feature Z

From the above images, we can see that the information gain is maximum when we make a split on feature Y. So, for the root node best-suited feature is feature Y. Now we can see that while splitting the dataset by feature Y, the child contains a pure subset of the target variable. So we don’t need to further split the dataset. The final tree for the above dataset would look like this:

A diagram of a network

Description automatically generated

 2. Gini Index

* Gini Index is a metric to measure how often a randomly chosen element would be incorrectly identified.
* It means an attribute with a lower Gini index should be preferred.
* Sklearn supports “Gini” criteria for Gini Index and by default, it takes “gini” value.
* The Formula for the calculation of the Gini Index is given below.

The Formula for Gini Index is given by :

Screenshot-from-2023-12-26-08-40-38

*Gini Impurity*

The Gini Index is a measure of the inequality or impurity of a distribution, commonly used in decision trees and other machine learning algorithms. It ranges from 0 to 0.5, where 0 indicates a pure set (all instances belong to the same class), and 0.5 indicates a maximally impure set (instances are evenly distributed across classes).

Some additional features and characteristics of the Gini Index are:

* It is calculated by summing the squared probabilities of each outcome in a distribution and subtracting the result from 1.
* A lower Gini Index indicates a more homogeneous or pure distribution, while a higher Gini Index indicates a more heterogeneous or impure distribution.
* In decision trees, the Gini Index is used to evaluate the quality of a split by measuring the difference between the impurity of the parent node and the weighted impurity of the child nodes.
* Compared to other impurity measures like entropy, the Gini Index is faster to compute and more sensitive to changes in class probabilities.
* One disadvantage of the Gini Index is that it tends to favour splits that create equally sized child nodes, even if they are not optimal for classification accuracy.
* In practice, the choice between using the Gini Index or other impurity measures depends on the specific problem and dataset, and often requires experimentation and tuning.

Example of a Decision Tree Algorithm

Forecasting Activities Using Weather Information

* Root node: Whole dataset
* Attribute : “Outlook” (sunny, cloudy, rainy).
* Subsets: Overcast, Rainy, and Sunny.
* Recursive Splitting: Divide the sunny subset even more according to humidity, for example.
* Leaf Nodes: Activities include “swimming,” “hiking,” and “staying inside.”

Beginning with the entire dataset as the root node of the decision tree:

* Determine the best attribute to split the dataset based on information gain, which is calculated by the formula: Information gain = Entropy(parent) – [Weighted average] \* Entropy(children), where entropy is a measure of impurity or disorder of a set of examples, and the weighted average is based on the number of examples in each child node.
* Create a new internal node that corresponds to the best attribute and connects it to the root node. For example, if the best attribute is “outlook” (which can have values “sunny”, “overcast”, or “rainy”), we create a new node labeled “outlook” and connect it to the root node.
* Partition the dataset into subsets based on the values of the best attribute. For example, we create three subsets: one for instances where the outlook is “sunny”, one for instances where the outlook is “overcast”, and one for instances where the outlook is “rainy”.
* Recursively repeat steps 1-4 for each subset until all instances in a given subset belong to the same class or no further splitting is possible. For example, if the subset of instances where the outlook is “overcast” contains only instances where the activity is “hiking”, we assign a leaf node labeled “hiking” to this subset. If the subset of instances where the outlook is “sunny” is further split based on the humidity attribute, we repeat steps 2-4 for this subset.
* Assign a leaf node to each subset that contains instances that belong to the same class. For example, if the subset of instances where the outlook is “rainy” contains only instances where the activity is “stay inside”, we assign a leaf node labeled “stay inside” to this subset.
* Make predictions based on the decision tree by traversing it from the root node to a leaf node that corresponds to the instance being classified. For example, if the outlook is “sunny” and the humidity is “high”, we traverse the decision tree by following the “sunny” branch and then the “high humidity” branch, and we end up at a leaf node labeled “swimming”, which is our predicted activity.

Advantages of Decision Tree

* Easy to understand and interpret, making them accessible to non-experts.
* Handle both numerical and categorical data without requiring extensive preprocessing.
* Provides insights into feature importance for decision-making.
* Handle missing values and outliers without significant impact.
* Applicable to both classification and regression tasks.

Disadvantages of Decision Tree

* Disadvantages include the potential for overfitting
* Sensitivity to small changes in data, limited generalization if training data is not representative
* Potential bias in the presence of imbalanced data.

**Tree pruning techniques: pre-pruning, post-pruning.**

Decision tree pruning is a critical technique in machine learning used to optimize decision tree models by reducing overfitting and improving generalization to new data. In this guide, we’ll explore the importance of decision tree pruning, its types, implementation, and its significance in machine learning model optimization.

**What is Decision Tree Pruning?**

[Decision tree](https://www.geeksforgeeks.org/decision-tree/) pruning is a technique used to prevent decision trees from[overfitting](https://www.geeksforgeeks.org/underfitting-and-overfitting-in-machine-learning/) the training data. Pruning aims to simplify the decision tree by removing parts of it that do not provide significant predictive power, thus improving its ability to generalize to new data.

Decision Tree Pruning removes unwanted nodes from the overfitted decision tree to make it smaller in size which results in more fast, more accurate and more effective predictions.

**Types Of Decision Tree Pruning**

There are two main types of decision tree pruning: **Pre-Pruning** and **Post-Pruning**.

**Pre-Pruning (Early Stopping)**

Sometimes, the growth of the decision tree can be stopped before it gets too complex, this is called pre-pruning. It is important to prevent the overfitting of the training data, which results in a poor performance when exposed to new data.

Some common pre-pruning techniques include:

* **Maximum Depth**: It limits the maximum level of depth in a decision tree.
* **Minimum Samples per Leaf**: Set a minimum threshold for the number of samples in each leaf node.
* **Minimum Samples per Split**: Specify the minimal number of samples needed to break up a node.
* **Maximum Features:** Restrict the quantity of features considered for splitting.

By pruning early, we come to be with a simpler tree that is less likely to overfit the training facts.

**Post-Pruning (Reducing Nodes)**

After the­ tree is fully grown, post-pruning involves re­moving branches or nodes to improve the­ model’s ability to generalize­. Some common post-pruning techniques include­:

* **Cost-Complexity Pruning (CCP)**: This method assigns a price to each subtre­e primarily based on its accuracy and complexity, the­n selects the subtre­e with the lowest fee.
* **Re­duced Error Pruning**: Removes branche­s that do not significantly affect the overall accuracy.
* **Minimum Impurity De­crease**: Prunes node­s if the decrease­ in impurity (Gini impurity or entropy) is beneath a ce­rtain threshold.
* **Minimum Leaf Size**: Re­moves leaf nodes with fe­wer samples than a specifie­d threshold.

Post-pruning simplifies the tre­e while preserving its Accuracy. Decision tree pruning helps to improve the performance and interpretability of decision trees by reducing their complexity and avoiding overfitting. Proper pruning can lead to simpler and more robust models that generalize better to unseen data.

**Decision Tree Implementation in Python**

Here we are going to create a decision tree using preloaded dataset **breast\_cancer**in sklearn library.

The Decision Tree model is using pre-pruning technique, specifically, the default approach of scikit-learn’s DecisionTreeClassifier, which employs the Gini impurity criterion for making splits. This is evident from the parameter criterion="gini" passed to the [DecisionTreeClassifier()](https://www.geeksforgeeks.org/decision-tree-implementation-python/" \t "_blank)constructor. Gini impurity is a measure of how often a randomly chosen element from the set would be incorrectly labeled if it were randomly labeled according to the distribution of labels in the set.

Python3

**from** **sklearn.datasets** **import** load\_breast\_cancer

**from** **sklearn.model\_selection** **import** train\_test\_split

**from** **sklearn.tree** **import** DecisionTreeClassifier

**from** **sklearn.tree** **import** plot\_tree

**import** **matplotlib.pyplot** **as** **plt**

*# Load breast cancer dataset*

X, y = load\_breast\_cancer(return\_X\_y=**True**)

*# Separating Training and Testing data*

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, train\_size=0.2, random\_state=42)

*# Train decision tree model*

model = DecisionTreeClassifier(criterion="gini")

model.fit(X\_train, y\_train)

*# Plot original tree*

plt.figure(figsize=(15, 10))

plot\_tree(model, filled=**True**)

plt.title("Original Decision Tree")

plt.show()

*# Model Accuracy before pruning*

accuracy\_before\_pruning = model.score(X\_test, y\_test)

print("Accuracy before pruning:", accuracy\_before\_pruning)

**Output:**

Accuracy before pruning: 0.8793859649122807

**Decision Tree Pre-Pruning Implementation**

In the implementation, we pruning technique is hyperparameter tuning through cross-validation using GridSearchCV. Hyperparameter tuning involves searching for the optimal hyperparameters for a machine learning model to improve its performance. It does not directly prune the decision tree, but it helps in finding the best combination of hyperparameters, such as max\_depth, max\_features, criterion, and splitter, which indirectly controls the complexity of the decision tree and prevents overfitting. Therefore, it’s a form of post-pruning technique.

Python3

**from** **sklearn.tree** **import** DecisionTreeClassifier

parameter = {

'criterion' :['entropy','gini','log\_loss'],

'splitter':['best','random'],

'max\_depth':[1,2,3,4,5],

'max\_features':['auto','sqrt','log2']

}

model = DecisionTreeClassifier()

**from** **sklearn.model\_selection** **import** GridSearchCV

cv = GridSearchCV(model,param\_grid = parameter,cv = 5)

cv.fit(X\_train,Y\_train)

**Visualizing**

Python3

**from** **sklearn.tree** **import** export\_graphviz

**import** **graphviz**

best\_estimator = cv.best\_estimator\_

feature\_names = features

dot\_data = export\_graphviz(best\_estimator, out\_file=**None**, filled=**True**, rounded=**True**,

feature\_names=feature\_names, class\_names=['0', '1', '2'])

graph = graphviz.Source(dot\_data)

graph.render("decision\_tree", format='png', cleanup=**True**)

graph

**Output:**

**Best Parameters**

Python3

cv.score(X\_test,Y\_test)

cv.best\_params\_

**Output:**

0.9736842105263158

{'criterion': 'gini',

'max\_depth': 4,

'max\_features': 'sqrt',

'splitter': 'best'}

**Decision Tree Post-Pruning Implementation**

Python3

*# Cost-complexity pruning (Post-pruning)*

path = model.cost\_complexity\_pruning\_path(X\_train, y\_train)

ccp\_alphas, impurities = path.ccp\_alphas, path.impurities

*# Train a series of decision trees with different alpha values*

pruned\_models = []

**for** ccp\_alpha **in** ccp\_alphas:

pruned\_model = DecisionTreeClassifier(criterion="gini", ccp\_alpha=ccp\_alpha)

pruned\_model.fit(X\_train, y\_train)

pruned\_models.append(pruned\_model)

*# Find the model with the best accuracy on test data*

best\_accuracy = 0

best\_pruned\_model = **None**

**for** pruned\_model **in** pruned\_models:

accuracy = pruned\_model.score(X\_test, y\_test)

**if** accuracy > best\_accuracy:

best\_accuracy = accuracy

best\_pruned\_model = pruned\_model

*# Model Accuracy after pruning*

accuracy\_after\_pruning = best\_pruned\_model.score(X\_test, y\_test)

print("Accuracy after pruning:", accuracy\_after\_pruning)

**Output:**

Accuracy after pruning: 0.918859649122807

Python3

*# Plot pruned tree*

plt.figure(figsize=(15, 10))

plot\_tree(best\_pruned\_model, filled=**True**)

plt.title("Pruned Decision Tree")

plt.show()

**Output:**

#Displayed by the associates

**Why Pruning decision trees is Important?**

Decision Tree Pruning has an important role in optimizing the decision tree model. It involves the removal of certain parts of the tree which can potentially reduce its performance. Here is why decision tree pruning is important:

1. **Prevents Overfitting:**Decision trees are prone to overfitting, where the model memorizes the training data rather than learning generalizable patterns. Pruning helps prevent overfitting by simplifying the tree structure, removing branches that capture noise or outliers in the training data.
2. **Improves Generalization:** By reducing the complexity of the decision tree, pruning enhances the model’s ability to generalize to unseen data. A pruned decision tree is more likely to capture underlying patterns in the data rather than memorizing specific instances, leading to better performance on new data.
3. **Reduces Model Complexity**: Pruning results in a simpler decision tree with fewer branches and nodes. This simplicity not only makes the model easier to interpret but also reduces computational requirements during both training and inference. A simpler model is also less prone to overfitting and more robust to changes in the data.
4. **Enhances Interpretability**: Pruning produces decision trees with fewer branches and nodes, which are easier to interpret and understand. This is particularly important in applications where human insight into the decision-making process is valuable, such as in medical diagnosis or financial decision-making.
5. **Speeds Up Training and Inference**: Pruned decision trees require less computational resources during both training and inference phases. With fewer branches and nodes, the decision-making process becomes more efficient, resulting in faster predictions without sacrificing accuracy.
6. **Facilitates Model Maintenance**: Pruning helps maintain decision tree models over time by keeping them lean and relevant. As new data becomes available or the problem domain evolves, pruned decision trees are easier to update and adapt compared to overly complex, unpruned trees.

**Handling categorical predictors and missing values.**

<https://www.geeksforgeeks.org/how-to-handle-missing-values-of-categorical-variables-in-python/>

<https://www.scaler.com/topics/data-science/categorical-missing-values/>

***Thank you***