NAME	**SAIVARDHAN JETHIWALA**
STUDENT ID	**24071402**
SUBJECT	**MACHINE LEARNING AND NEURAL NETWORKS**
SUBJECT ID	**7PAM2021-0901-2025**
ASSIGNMENT NAME	**INDIVIDUAL ASSIGNMENT: MACHINE LEARNING TUTORIAL**

DECISION TREES FOR CLASSIFICATION: A COMPREHENSIVE TUTORIAL

INTRODUCTION

Decision Trees Classification: An All-Inclusive Guide Overview for tasks involving regression and classification, decision trees are a well-liked supervised learning technique. By learning basic decision rules from data and organizing them into a tree-like structure of if-then conditions, a decision tree model forecasts target values. They are among the most popular machine learning algorithms because they are easy to comprehend and visualize, even for non-statisticians. For instance, a classification tree represents internal nodes as feature tests, branches as the outcomes of such tests, and leaf nodes as class labels or class probabilities. In practice, decision trees are a very interpretable model since their white-box nature provides tractability in the decision-making process behind every prediction [1][2].

The main point is that the target classes can be grouped in a homogeneous way by recursively dividing the dataset according to feature values [5]. The method continues to create new nodes from the leftover data after each division, thus it splits the data by the feature which separates the classes best. This operation, called recursive partitioning, terminates when it reaches the stopping conditions [7]. Such conditions are, for instance, when all the data points at a node are of one class or when the maximum tree depth has been achieved. The outcome is a nested set of conditions. Looking at a route from the root to a leaf in the decision tree is like seeing how a sequence of decisions results in a certain classification. leads to a specific classification [1].

HOW DECISION TREES WORK

At a high level, training a decision tree involves:

- **Splitting the Data:** At each node, the algorithm evaluates possible splits on all features and finds the one that yields the most "information gain" (i.e. best separates the classes). Depending on the response, the data is split into branches (yes/no for a binary split, for example). After deciding on the split, the data is separated into branches based on the answer. For instance, a binary split would have branches yes and no [1].
- **Recursive Partitioning**: Each branch resulting from a split is recursively split further, thus creating a binary tree structure and subsets of data. The top-down induction continues until certain criteria are met, for example, a node being pure that is all the samples in that node belong to the same class or no split yielding an improvement [7]. (terminal nodes) then assign a class label (or a distribution of labels) to any sample that falls into that leaf[7].

Each path from the root to a leaf can be read as a series of if-then rules that classify an instance. For example, A small decision tree trained to predict survival of Titanic passengers. It illustrates how a tree represents decision logic: the root node splits on Sex, then subsequent nodes (e.g., checking Age and No of Siblings/Spouses (sibsp)) lead to predictions of Survived or Died. This decision tree might be seen as being consistent with the Titanic data trends that have been confirmed: "If the person is a female, then they survived; if the person is a male but his age is 95 and sibsp is 3, then he survived; in all other cases, the person did not survive." Internal nodes query a feature (such as "Sex = male?"), and branches divide the data appropriately. Each leaf node outputs a prediction (e.g., survival probability and class) [2].

A great exercise is to manually walk through a decision path for a sample data point to see how the classification is made. This reinforces how the tree's conditional logic works in a human-interpretable way.

Behind the scenes, the crucial component is how the "best" feature and threshold are selected at each split. This is determined by a split criterion that measures how well a feature divides the data into homogeneous subsets (i.e., subsets containing mostly a single class). The most common criteria are based on impurity measures like entropy or Gini impurity, which we discuss next [1][2].

SPLITTING CRITERIA: ENTROPY, INFORMATION GAIN, AND GINI

Impurity of a node refers to how mixed the classes are in that subset of data. A node with all samples from one class is pure (impurity 0), whereas a node with a perfectly even mix of classes is highly impure. Decision tree algorithms aim to reduce impurity with each split. Two widely used impurity measures are information entropy (with information gain) and Gini index.

- **Entropy:** In information theory, entropy is understood as a measure of disorder or uncertainty.

$$E = -\sum_{i=1}^{C} p(i) \log_2 p(i),$$

Here, p(i) is the proportion of class (i) in that node. Entropy is 0 when one class contains all examples (no uncertainty), while it is maximum when classes are perfectly mixed. For instance, the entropy of a node with 50% positive and 50% negative instances will be higher than that of a node containing 90% of one class and 10% of the other.

- **Information Gain**

Information Gain works hand-in-hand with entropy and is the main criteria used for deciding splits. Information gain is the difference in entropy between the original node and the children nodes after a branch division. In other words:

$$\text{Information Gain} = E_{\text{parent}} - \sum_{\text{children}} \frac{n_i}{n_{\text{parent}}} E_{\text{child}_i},$$

Where parent is the entropy of the node before the split and the summation stands for the entropy of the child nodes weighted by their proportions. The algorithm evaluates each possible split by calculating the information gain for that split and then, it selects the split for which this gain is the highest, i.e. the entropy decrease is the largest (the increase in purity is the greatest). In fact, the best split would generate children nodes each consisting of only one class (entropy 0), thus, attaining the highest information gain.

- **Gini Impurity:** The Gini index is the second most common impurity measure (used by the CART algorithm). The Gini impurity $G(t)$ of a node is defined as:

$$G = \sum_{i=1}^{C} p(i)\,(1 - p(i)) = 1 - \sum_{i=1}^{C} [p(i)]^2.$$

This formula gives the probability of misclassifying a random sample from the node if we label it according to the node's class distribution. Like entropy, Gini is 0 for a pure node and higher when classes are more mixed (for a 50/50 split between two classes, $G=0.5$). The **Gini gain** from a split is computed analogously to information gain (parent Gini minus weighted child Gini). The tree algorithm will choose the split that minimizes the Gini impurity of the children (equivalently, maximizes the reduction in Gini) [2].

Entropy vs. Gini: Both criteria often yield similar results in practice. Gini is somewhat faster to compute (no logarithm) and is the default in many implementations (including scikit-learn's DecisionTreeClassifier). Entropy (used in the ID3 and C4.5 algorithms by Quinlan) has a more theoretical foundation in information theory [1]. A minor difference is that entropy tends to penalize very impure nodes a bit more than Gini, but the choice rarely leads to dramatically different trees. In practice:

- Gini is often preferred for its computational efficiency and is the default for CART-based algorithms.
- Entropy is conceptually intuitive (related to information gain) and is sometimes favored in educational contexts or when following the ID3/C4.5 tradition.

To illustrate these concepts,we calculate entropy and information gain for a simple split by hand. For example, given a small dataset with two classes, they can compute the parent entropy and the entropy after splitting on a particular feature. This exercise builds intuition on how a decision tree "decides" on a split based on impurity reduction.

HYPERPARAMETERS AND AVOIDING OVERFITTING

One of the issues with the decision trees is that they tend to overfit the training data if the trees grow too complex. A fully grown tree will keep splitting until every leaf is pure, or until there aren't any more data points to split. This can often result in a really deep tree that memorizes training samples but fails to generalize to new data, which is what high variance models do. To reduce overfitting, we are able to constrain growth in the tree using a hyperparameter or use pruning techniques [3][4][7].

Important hyperparameters for the control of the complexity of decision trees include:

- **max_depth:** The tree's maximum depth (the count of splits from root to a leaf) is limited by depth stopping the tree from becoming arbitrarily complex. So, with max_depth=3 we have at most 3 consecutive splits, which can lead to underfitting, but the variance of an unbounded tree is reduced.[7]
- **min_samples_split:** A node with a smaller number of samples than this value will be a leaf. The tree will be more conservative for higher values of min_samples_split when it splits on small subsets if min_samples_split is increased.
- **min_samples_leaf:** The least number of samples that must be in a leaf node. As an example, min_samples_leaf=5 means each leaf will have at least 5 samples, thus it could be that the tree is stopped from making leaves with very few samples - which are possibly noise or outliers. This leads to wider splits that have better generalization capabilities.
- **max_leaf_nodes:** Restricts the number of leaves. The procedure will continue to select the most significant splits up to this point. It is, thus, another method of controlling a tree's size.
- **criterion:** The split criterion can be "gini" or "entropy" for classification, as decided in the discussion. This, however, does not avert overfitting directly, but the tree structure can be slightly influenced [7].
- **max_features:** In classification trees-especially when using ensembling methods such as Random Forests-you can restrict how many features are considered at each split. While usually used for ensembling, the usage of max_features < total_features adds randomness that can reduce variance, though a single tree with feature bagging is uncommon outside of ensembles[6].
- **CCP_alpha :** the new hyperparameter introduced in Scikit-learn for post-pruning in cost complexity pruning. The penalty value for tree complexity: higher ccp_alpha values will prune more of the tree. One can search an optimal number of this parameter by using cross-validation and obtain an optimal trade-off between complexity and accuracy[3][4].

In practice, one can get a tree's cost-complexity pruning path and select an alpha which leads to the best validation score.

These hyperparameters will need tuning. An over-constrained tree may underfit-for example, a depth-1 tree can be too simple to capture patterns-whereas an under-constrained tree can overfit. In general, the way one goes about determining appropriate values is by using techniques such as cross-validation, perhaps coupled with grid-search for the best max_depth or ccp_alpha that maximizes validation accuracy.

Pruning: This is the process of trimming a fully grown tree to avoid overfitting. They are of two types:

- **pre-pruning:** It stops growing the tree well before it overfits; criteria for that are the same as the above, such as limiting depth, minimum number of samples, etc. That's mainly what scikit-learn does by setting those limits - essentially never letting the tree overfit in the first place.
- **Post-pruning:** The approach starts by developing the complete tree, then pruning the branches that contribute very little. One heuristic is cost-complexity pruning, used by CART, which prunes the branches that have low improvement relative to the added complexity [3][4]. Scikit-learn's ccp_alpha provides an implementation of that by adding the size penalization to the impurity of the tree. By increasing alpha, we progressively prune more branches. Other libraries or algorithms such as C4.5 perform post-pruning using alternative approaches like evaluating the splits on a validation set or heuristically removing branches that do not generalize [1].

In other words, the growth of the tree should be controlled in order to obtain a model that generalizes well. Usually a small depth and a sufficient number of minimum samples per leaf are taken at the start, and complexity is increased later if needed. The model's performance should always be checked on unseen data to ensure that it is not overfitting.

REFERENCES

[1] J. R. Quinlan, "Decision trees and decision-making," IEEE Transactions on Systems, Man, and Cybernetics, vol. 20, no. 2, pp. 339–346, 1990.

[2] S. R. Safavian and D. Landgrebe, "A survey of decision tree classifier methodology," IEEE Transactions on Systems, Man, and Cybernetics, vol. 21, no. 3, pp. 660–674, 1991.

[3] S. B. Gelfand, C. S. Ravishankar and E. J. Delp, "An iterative growing and pruning algorithm for classification tree design," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 13, no. 2, pp. 163–174, 1991.

[4] F. Esposito, D. Malerba and G. Semeraro, "A comparative analysis of methods for pruning decision trees," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 19, no. 5, pp. 476–491, 1997.

[5] B. Kim and D. A. Landgrebe, "Hierarchical classifier design in high-dimensional numerous class cases," IEEE Transactions on Geoscience and Remote Sensing, vol. 29, no. 4, pp. 518–528, 1991.

[6] T. K. Ho, "The random subspace method for constructing decision forests," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 20, no. 8, pp. 832–844, 1998.

[7] L. Rokach and O. Maimon, "Top-down induction of decision trees classifiers — a survey," IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews, vol. 35, no. 4, pp. 476–487, 2005.