A Comprehensive Introduction to Decision Trees

Jawad ALAOUI M'HAMMEDI

September 2024

Contents

1 Introduction

Decision trees are a class of supervised learning models used for both classification and regression tasks. Their popularity arises from three key characteristics:

1. **Interpretability:** Decision trees produce rules that are relatively simple to understand and explain. For example, we can interpret a decision process as a sequence of questions leading to a final answer.
2. **Flexibility:** Decision trees can approximate complex, non-linear decision boundaries, making them highly versatile.
3. **No Need for Complex Feature Engineering:** Because trees split along feature axes, they can naturally discover interactions and non-linearities between variables without the need for explicit feature engineering.

In this lesson, we will explore the core concepts of decision trees in detail. We will begin by discussing what a decision tree is and how it partitions the feature space. Next, we will delve into the core components of building a tree: thresholds and predictors chosen at each split, and the criteria used to evaluate potential splits (accuracy, Gini index, entropy). We will then consider how to determine when to stop growing a tree, discussing various stopping conditions. Finally, we will examine pruning techniques—both pre- and post-pruning (including early stopping)—to prevent overfitting and improve the generalization capability of the model.

2 What is a Decision Tree?

A decision tree is a flowchart-like structure where each internal node represents a test on a feature, each branch represents an outcome of that test, and each leaf node (terminal node) represents a class label (for classification) or a predicted value (for regression).

For classification problems, imagine you have a dataset $\{(\mathbf{x}_i, y_i)\}_{i=1}^N$ where each $\mathbf{x}_i \in \mathbb{R}^p$ is a p-dimensional feature vector and $y_i \in \{1, 2, \ldots, K\}$ is a class label. A decision tree model attempts to split the feature space into M disjoint regions $R_1, R_2, \ldots, R_M$, where each R_m contains observations that mostly belong to one class. When a new point $\mathbf{x}$ is to be classified, we determine which region R_m it falls into, and assign the class that is most common in R_m among the training points.

3 How Does a Tree Partition the Feature Space?

At each internal node of a decision tree, we pick a feature j and a threshold t_j, and split the data into two subsets:

$$R_{\text{left}} = \{\mathbf{x} : x_j \leq t_j\} \quad \text{and} \quad R_{\text{right}} = \{\mathbf{x} : x_j > t_j\}.$$

This corresponds to a single decision rule: "Is $x_j \leq t_j$?". If yes, go left; if no, go right. By applying such rules recursively, we form a tree structure. Ultimately, these splits define a partition of the space into rectangular (or axis-aligned) regions.

4 Choosing Which Feature and Threshold to Split On

The central question at each internal node is: which feature and threshold should we use for the split? The goal is to find a split that best separates the data into purer subsets. "Purity" here refers to how dominated a node's data is by a single class.

4.1 Common Splitting Criteria

A good split reduces the impurity or, equivalently, increases the purity of the resulting nodes. Several criteria are commonly used:

4.1.1 Classification Accuracy (Error Rate)

The classification error of a node is defined as:

$$\text{Error}(R) = 1 - \max_k \hat{p}_{k|R},$$

where $\hat{p}_{k|R}$ is the proportion of class k points in the region R. This measure is simple but not very sensitive to changes in node purity unless the node is nearly pure.

4.1.2 Gini Index

The Gini index is a more common choice:

$$G(R) = \sum_{k=1}^{K} \hat{p}_{k|R}(1 - \hat{p}_{k|R}) = \sum_{k=1}^{K} \hat{p}_{k|R}(1 - \hat{p}_{k|R}) = \sum_{k=1}^{K} \hat{p}_{k|R} - \sum_{k=1}^{K} \hat{p}_{k|R}^2 = 1 - \sum_{k=1}^{K} \hat{p}_{k|R}^2.$$

For a binary classification ($K = 2$) with $\hat{p} = \hat{p}_{1|R}$, this reduces to:

$$G(R) = 2\hat{p}(1 - \hat{p}).$$

The Gini index is 0 if the node is completely pure (i.e., all samples belong to one class), and it reaches its maximum when classes are equally mixed.

4.1.3 Cross-Entropy (or Entropy)

Another popular criterion is the cross-entropy, also known as the entropy for classification:

$$H(R) = -\sum_{k=1}^{K} \hat{p}_{k|R} \log(\hat{p}_{k|R}).$$

Like the Gini index, entropy is minimized when a node is pure and is maximized when classes are evenly distributed.

4.2 Choosing the Best Split

To select the best feature j and threshold t_j at a node, we consider all possible splits and choose the one that yields the greatest decrease in impurity. Formally, if $I(R)$ represents the impurity measure (Gini, entropy, etc.) of the parent node R, and R_{left}, R_{right} are the child nodes after the split, we define:

$$\Delta I(j, t_j) = I(R) - \left[\frac{N_{\text{left}}}{N} I(R_{\text{left}}) + \frac{N_{\text{right}}}{N} I(R_{\text{right}}) \right],$$

where N is the number of samples in R, and N_{left}, N_{right} are the number of samples in the left and right splits, respectively. We pick (j, t_j) that maximizes $\Delta I(j, t_j)$.

5 Stopping Conditions for Tree Growth

Growing a tree until all leaves are perfectly pure (or until each leaf has only one data point) often results in overfitting. Such a tree perfectly explains the training data but may not generalize well. Therefore, we must decide when to stop adding splits.

Common stopping conditions include:

1. **Maximum Depth:** Stop if the tree reaches a specified maximum depth $d_{\max}$. For example, we might say the tree cannot exceed depth 10.
2. **Minimum Number of Samples in a Leaf:** Stop splitting a node if it would result in a leaf that contains fewer than a certain number of samples, say $m_{\min}$. For instance, do not create a leaf with fewer than 5 samples.

3. **Minimum Improvement in Impurity:** Stop if no split improves impurity by at least a certain threshold (i.e., the gain is too small).

4. **Maximum Number of Leaves:** Stop if adding another split would create more leaves than a predefined maximum.

5. **Pure Leaf Condition:** If a node is already pure (containing only one class), no further splitting is required.

These constraints prevent the tree from growing too complex, forcing it to remain simpler and more generalizable.

6 Pruning Methods: Pre-Pruning, Post-Pruning, and Early Stopping

Even with stopping conditions, it's common for trees to overfit. A remedy is pruning, which involves cutting back a fully grown tree to produce a smaller, less complex tree. Pruning can occur before the tree is fully grown (pre-pruning) or after the tree is fully grown (post-pruning).

6.1 Pre-Pruning (Early Stopping)

Pre-pruning (also known as early stopping) involves stopping the growth of the tree early based on predefined conditions. The stopping conditions discussed above (such as maximum depth or minimum samples per leaf) are forms of pre-pruning. Pre-pruning is simple, fast, and ensures the tree does not become overly complex. However, it may sometimes stop too early, preventing the model from exploring helpful splits.

6.2 Post-Pruning (Cost-Complexity Pruning)

Post-pruning starts with a large, fully grown tree and then prunes it back. The idea is:

1. Grow a very large tree that overfits the training data.

2. Use a pruning criterion to remove subtrees that do not sufficiently reduce errors on validation data.

A common approach is cost-complexity pruning. We define the cost of a subtree T as:

$$\text{Cost}(T) = \sum_{m=1}^{M_T} E(R_m) + \alpha |T|,$$

where $E(R_m)$ is the impurity (or error) at leaf m, M_T is the number of leaves in T, and α is a complexity parameter. By increasing α, we penalize complex trees more. We then find a subtree T that minimizes $\text{Cost}(T)$.

6.2.1 Procedure for Cost-Complexity Pruning

1. Grow a large tree T_0. 2. For a sequence of increasing α, find the subtree T_α that minimizes $\text{Cost}(T_\alpha)$. 3. Use cross-validation to choose α. 4. The chosen α will correspond to an optimally pruned subtree that balances fit and complexity.

6.3 Comparison of Pre- and Post-Pruning

- **Pre-Pruning:** Decides to stop early. Pros: fast and simple. Cons: might stop too soon. - **Post-Pruning:** Grows a large tree and then cuts it back. Pros: often yields better-performing models. Cons: computationally more expensive because you need to grow the full tree first and then prune.

7 Mathematical Formulation of a Decision Tree Prediction

For a K-class classification problem, each leaf R_m of a decision tree corresponds to a region of the feature space. Let $\hat{p}_{k|R_m}$ be the empirical probability of class k in leaf R_m:

$$\hat{p}_{k|R_m} = \frac{1}{|R_m|} \sum_{\mathbf{x}_i \in R_m} \mathbf{1}(y_i = k),$$

where $|R_m|$ is the number of samples in R_m. When predicting the class for a new point $\mathbf{x}$, we find the leaf R_m that $\mathbf{x}$ falls into and pick the class:

$$\hat{y}(\mathbf{x}) = \arg\max_k \hat{p}_{k|R_m}.$$

For regression tasks, each leaf might store the mean response value:

$$\hat{y}(R_m) = \frac{1}{|R_m|} \sum_{\mathbf{x}_i \in R_m} y_i.$$

8 Handling Categorical Variables

Decision trees naturally handle numerical features through inequality splits. For categorical variables with multiple categories, we must find a way to partition the set of categories into two subsets. If a categorical variable has C categories, then there are $2^{C-1} - 1$ possible partitions. Exhaustively searching them can be expensive.

A simple approach is to encode categories as numeric values (e.g., label encoding) and treat them like numeric variables. Although this might not be optimal (since the numeric ordering is arbitrary), it often works well in practice. More sophisticated algorithms can find the best partition of categories without relying on arbitrary numeric encodings, but at a greater computational cost.

9 Computational Complexity and Practical Considerations

The complexity of building a decision tree depends on the search for the best split. For each feature, we may consider many potential thresholds. Naive approaches can be costly for large datasets. Optimizations, such as sorting values once per feature and then evaluating potential splits in linear time, are commonly employed.

Despite these optimizations, decision trees can still be sensitive to small changes in the data. They have high variance because a small perturbation in the training set can lead to a very different tree structure. Ensembling methods like random forests and gradient boosting can reduce this variance and substantially improve predictive performance.

10 Summary

Decision trees are foundational models in machine learning, prized for their interpretability and flexibility:

- They create a hierarchical, piecewise partition of the feature space.
- They rely on criteria like Gini, entropy, or error rate to choose splits that yield purer child nodes.
- Stopping conditions and pruning methods are essential to prevent overfitting.
- Pre-pruning, post-pruning, and early stopping methods each have their advantages and trade-offs.

While a single decision tree may not always be the best-performing model, understanding them is critical because they form the building blocks of many advanced ensemble methods. Decision trees illustrate the key ideas of model interpretability, balancing bias and variance, and controlling model complexity.

11 Further Reading

For more details:

- *Harvard IACS CS109A Advanced Section 7: Decision Trees and Ensemble Methods*, `https://harvard-iacs.github.io/2018-CS109A/a-sections/a-section-7/`
- James, Witten, Hastie, and Tibshirani, *An Introduction to Statistical Learning*, Chapter 8.
- Breiman, Friedman, Olshen, and Stone, *Classification and Regression Trees*.
- Hastie, Tibshirani, and Friedman, *The Elements of Statistical Learning*, Chapter 9.