Module D10: Introduction to Artificial Intelligence

Adam Darmanin

February 20, 2024

MASTERS OF BUSINESS ADMINISTRATION (MBA) in Leadership Excellence
Lecture Manuscript

Contents

1 Decision Trees

In this lesson we're going to talk about Decision Trees, a model that is as intuitive as it is powerful. Decision Trees are one of the most interpretable machine learning algorithms. They mimic the human way of thinking, breaking down decisions into a series of steps, each determined by a clear rule. Imagine trying to decide whether to play tennis based on the weather. You might first ask, "Is it sunny?" If it is, you might then ask, "Is it hot?" Each answer narrows your options until you reach a conclusion. This is exactly how a Decision Tree operates. Look at the image from "TEACHING AND LEARNING DATA-DRIVEN MACHINE LEARNING WITH EDUCATIONALLY DESIGNED JUPYTER NOTEBOOKS" from Fleischer et al. (2022).

NOTE TO EDITOR: Transition to Image

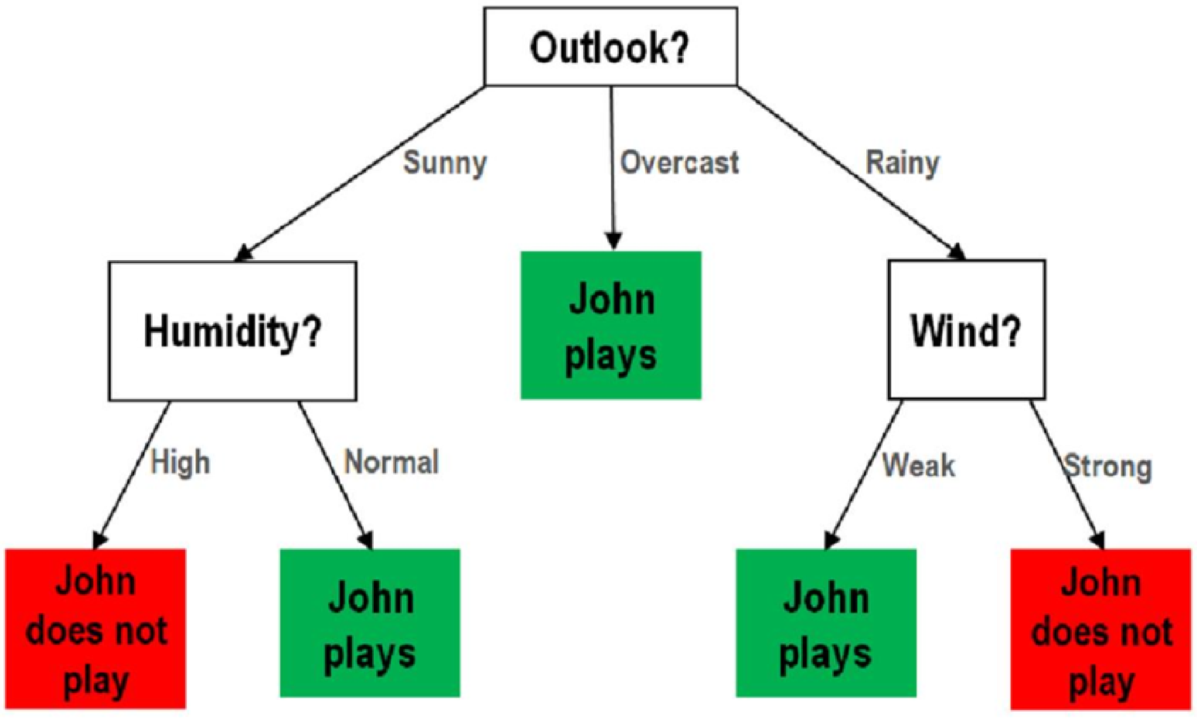

Figure 1: John Plays Tennis Decisions

In a decision tree, we start with a dataset that needs classification. At the very top of the tree, we have what's called the root node—this represents the entire dataset. The model then asks a question about one of the features and, based on the answer, splits the data into two or more groups. Each group is assigned to a branch, which leads to another node that continues this process of splitting. These splits continue until we reach leaf nodes, where the model makes its final decision. This is show on the next image, defining the tree structure:

NOTE TO EDITOR: Transition to Image

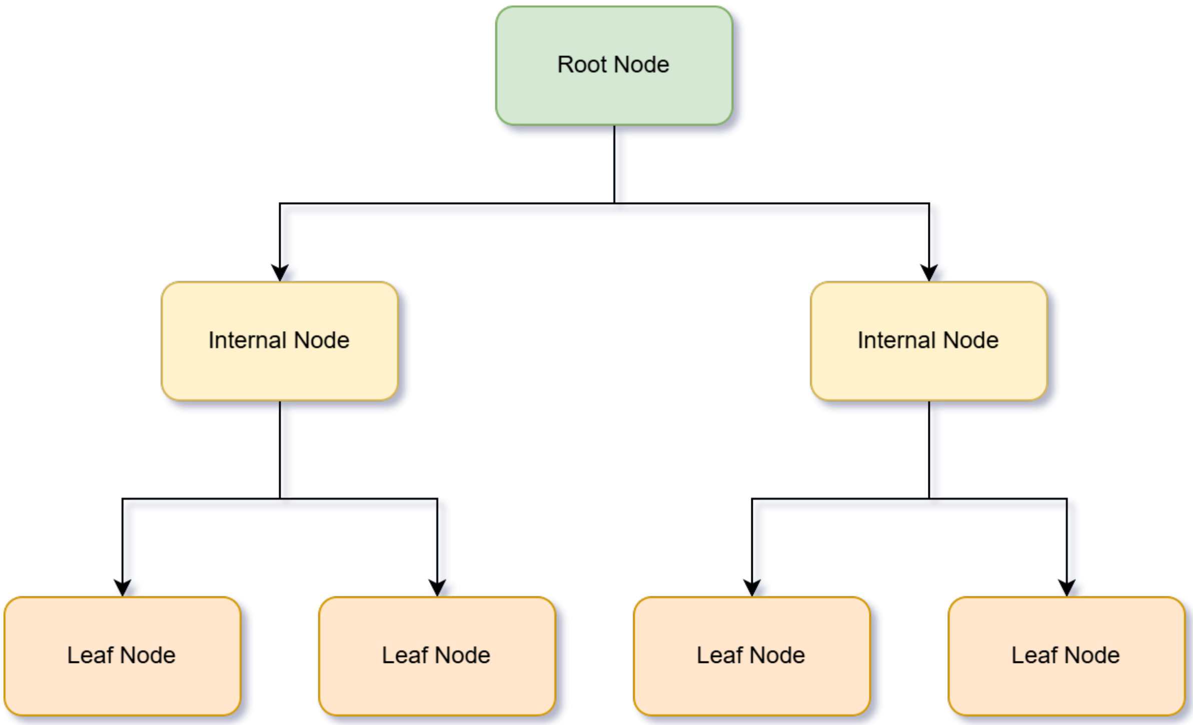

Figure 2: Tree Structure

This tree-like structure makes decision trees highly interpretable. In the previous tree we are trying to determine whether John will play tennis based on the weather. The tree might first check if the outlook is sunny, overcast, or rainy. If it's overcast, the decision is immediately "Yes." If it's sunny, the tree may then check another feature, like humidity. If the humidity is high, John won't play, but if it's normal, he will. The tree repeats this process until all possible cases have been considered, and each decision path leads to a final prediction.

Each internal node in the tree represents a Boolean test on a feature, meaning that it asks a yes/no question (or equivalent categorical question). The edges of the tree, the branches, are labeled with the possible values of the tested feature. When we reach a leaf node, we find the predicted class or outcome.

Decision trees can handle both classification and regression tasks. When used for classification, they predict discrete class labels. For regression, the leaf nodes contain a numerical value instead of a class label, which represents the predicted output.

But how does the tree decide what question to ask at each split? The key lies in impurity, a measure of how mixed the data is at a node. If all data points at a node belong to the same class, the node is pure, and no further splitting is needed. To quantify impurity, we use metrics like the Gini Index or Entropy. The Gini Index measures the likelihood that a randomly chosen data point would be misclassified if it were labeled according to the majority class at the node. It is calculated using the formula shown:

NOTE TO EDITOR: Transition to formula

$$G = 1 - \sum_{k=1}^{K} p_{i,k^2}$$

The Gini Index, denoted as G, measures impurity in a dataset. It is calculated as one minus the sum of the squared class probabilities within a given node. The summation runs from class index one to class index K, where each probability, denoted as p sub i comma k, represents the proportion of samples that belong to class k in node i. Squaring these probabilities gives greater weight to larger proportions and reduces the influence of smaller ones. When a node contains only one class, meaning all samples belong to the same category, the sum of squared probabilities is one, making the Gini Index equal to zero, which represents perfect purity. If multiple classes are evenly distributed, the sum of squared probabilities decreases, and the Gini Index approaches its maximum, indicating a high level of impurity. Entropy, another impurity measure, comes from information theory. It captures the uncertainty or randomness of a dataset and is defined as the formula on your screen:

NOTE TO EDITOR: Transition to formula

$$H = -\sum_{k=1}^{K} p_{i,k} \log_2(p_i, k)$$

Entropy, denoted as H, is calculated as the negative sum of the probability of each class multiplied by the logarithm base two of that same probability. The summation runs across all classes in the dataset. The logarithm function helps quantify how much information is contained in a given probability distribution. When a node is pure, meaning all samples belong to a single class, the probability for that class is one, and the logarithm of one is zero, leading to an entropy value of zero. This indicates no uncertainty. However, when multiple classes are equally represented, the entropy reaches its maximum value, meaning the dataset is highly uncertain. The goal in a decision tree is to minimize entropy at each split, reducing disorder and increasing the homogeneity of the resulting nodes.

Since probabilities always lie between zero and one, their logarithm values are negative. To keep entropy values non-negative, a negative sign is included in the formula. This ensures entropy represents a measure of disorder where a pure dataset has an entropy of zero, and a maximally mixed dataset has the highest entropy.

Entropy is a fundamental concept in information theory, where it measures the amount of uncertainty or disorder in a system. The choice of logarithm base determines the unit of measurement. Using a logarithm base two, as we do in decision trees, expresses entropy in bits, which aligns with how we quantify information in binary systems. However, in some algorithms and theoretical applications, you might encounter the natural logarithm, LN, which uses base e and expresses entropy in nats. This difference only affects the scale of entropy values and does not change the relative ranking of splits in a decision tree. Since decision trees and machine learning typically deal with information gain in bits, log base two is the standard choice.

Back to the problem of dividing the data in the nodes, the tree calculates the reduction in impurity for every possible split. This reduction, called information gain, is the difference in impurity between the parent node and the weighted sum of the child nodes. The figure from Wikipedia will help us visualize this concept.

NOTE TO EDITOR: TRANSITION TO IMAGE

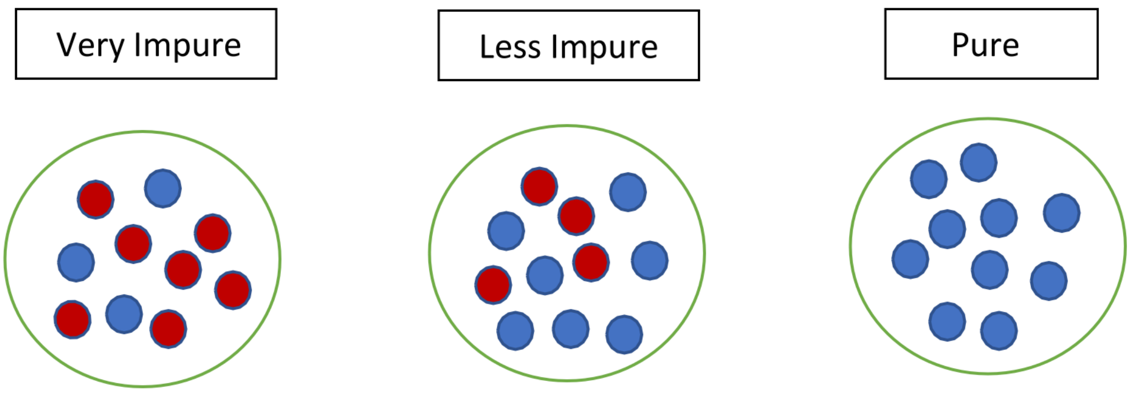

Figure 3: Entropy Examples

The formula for quantifying information gain is on your screen:

NOTE TO EDITOR: Transition to formula

$$\text{Information Gain} = H_{\text{parent}} - \sum_j \frac{N_j}{N_{\text{total}}} H_j$$

This quantifies the reduction in impurity when splitting a dataset into child nodes. It is defined as the impurity of the parent node, denoted as H sub Parent, minus the weighted sum of the impurities of the child nodes. The summation iterates over all possible child nodes, indexed by j. For each child node, its impurity, denoted as H J , is multiplied by the fraction of data points in that node relative to the total dataset in the parent node. This fraction is represented by N J divided by H Sub Total, where N J is the number of samples in the child node, and N sub Total is the total number of samples in the parent node before the split.

From an information-theoretic perspective, information gain is rooted in the concept of entropy, which measures the uncertainty in a dataset. It is closely linked to the Kullback-Leibler Divergence, also known as KLD, which quantifies the difference between two probability distributions. In the case of decision trees, KLD helps us understand how much the probability distribution of class labels changes before and after a split. The intuition is that a well-chosen split significantly reduces uncertainty, making the class distributions in the child nodes more predictable. KLD between two probability distributions, denoted as P and Q, is given by the formula on your screen:

NOTE TO EDITOR: Transition to formula

$$D_{KL}(P||Q) = \sum_k p_k \log_2 \frac{p_k}{q_k}$$

where P K represents the true distribution of class labels before the split, and Q K represents the new distribution after splitting. This divergence quantifies the inefficiency of assuming that the post-split distribution Q approximates the original distribution P. When a decision tree chooses an optimal split, it effectively reduces this divergence, making the post-split distribution more concentrated and predictable.

By maximizing information gain, decision trees effectively select the split that results in the most homogeneous child nodes, ensuring that each step in the tree structure meaningfully reduces the uncertainty in classification. This aligns with the principles of information theory, where the goal is to minimize entropy and maximize predictability in a dataset.

NOTE TO EDITOR: Transition to Image ”Impurity Measures”

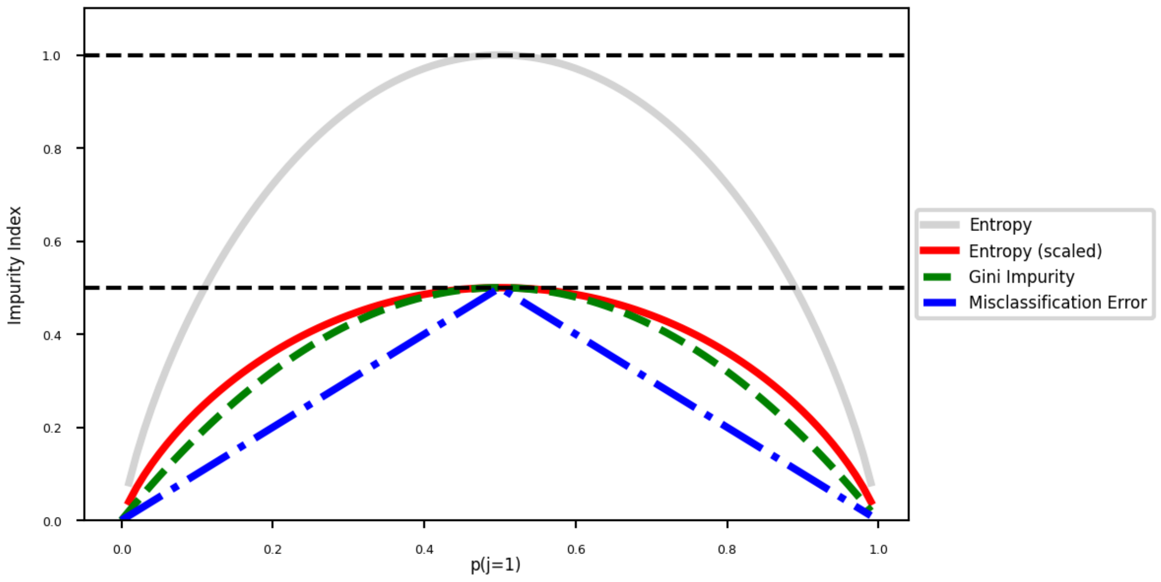

Figure 4: Impurity Measures

The graph on your screen, taken from the online archive: ”ML for Engineers”, compares different impurity measures used in decision trees, plotting impurity against P of J, the probability of an instance belonging to class 1. The x-axis represents this probability, while the y-axis denotes the impurity index, showing how impurity changes as the class distribution varies. The gray curve, representing unscaled entropy, follows a logarithmic shape, peaking at 1. This reflects maximum disorder when class labels are evenly split, meaning there is the highest uncertainty in classification. As P of J moves toward 0 or 1, entropy declines sharply, showing reduced impurity as one class dominates. The red curve is the same Entropy but scaled for direct comparison with other impurity measures, remember this technique from our lecture on Data Engineering? The green dashed curve represents the Gini Index. It follows a parabolic shape, also peaking at 0.5 but displaying a slightly lower maximum than entropy. The symmetry of the Gini curve emphasizes its focus on pairwise class probability differences rather than overall class uncertainty. Since Gini impurity avoids logarithmic computations, its curve is computationally smoother, making it an efficient criterion in decision tree learning. The blue curve represents Misclassification Error, which appears flatter compared to Gini and Entropy. Still it peaks at 0.5, like the other measures, but decreases more gradually. This lower sensitivity to probability changes indicates that it does not capture fine-grained class distinctions as effectively as Gini or Entropy. Consequently, while Misclassification Error provides an intuitive impurity measure, its relatively broad shape makes it less ideal for decision tree splitting. A key observation from the graph is that Entropy decreases more sharply than Gini when moving away from the peak, making it more sensitive to class imbalances. Gini, in contrast, remains computationally efficient and closely follows entropy’s shape. Misclassification Error, with its flatter trajectory, is less commonly used for split selection but can be useful in evaluating final model accuracy. The visualization effectively illustrates how impurity measures behave and guides the selection of splitting criteria in decision trees. Depending on the dataset and computational constraints, one may prefer entropy for its sensitivity, Gini for its efficiency, or Misclassification Error for simpler post-pruning decisions. With this knowledge, the choice between impurity measures in decision trees such as Gini Index and Entropy is guided by practical considerations, including computational efficiency, tree balance, and interpretability. The key idea is to select the criterion that maximizes purity gain, often measured by Information Gain or Gini Gain. The best criterion is chosen by evaluating each split and selecting the feature that maximizes information gain. Entropy is preferred when a probabilistic interpretation is needed, whereas Gini index is computationally more efficient due to the absence of logarithms.

Let's try to build a tree using our toy dataset of salaries. We determine which feature to split on first by evaluating impurity reduction using entropy and Gini index. A reminder of the toy dataset is on your screen:

NOTE TO EDITOR: Transition to Table

ID	Age	Salary	Gender
1	25	$50,000$	Male
2	35	$60,000$	Female
3	40	$80,000$	Male
4	35	$70,000$	Female
5	35	$1,000,000$	Male

To start, we compute the entropy of the dataset. This tells us how mixed the data is before any split is made. Since we are predicting Gender, we calculate entropy based on the proportion of Male and Female samples. On your screen are the steps:

NOTE TO EDITOR: Transition to Formula

$$H(S) = -\sum_{k=1}^{K} p_k \log_2(p_k)$$

$$p_{\text{Male}} = \frac{3}{5}, \quad p_{\text{Female}} = \frac{2}{5}$$

$$H(S) = -\left(\frac{3}{5}\log_2\frac{3}{5} + \frac{2}{5}\log_2\frac{2}{5}\right)$$

$$H(S) = -\left(0.6 \times -0.736 + 0.4 \times -1.321\right)$$

$$H(S) = -(-0.4416 - 0.5284) = 0.970$$

To compute entropy, we use a formula that quantifies uncertainty in a dataset. The formula works by taking the proportion of each class and multiplying it by the logarithm of that proportion. The negative sign ensures that the entropy value remains positive.

First, we determine the proportion of Male and Female samples in the dataset. There are three Male and two Female instances, so the probabilities are three-fifths for Male and two-fifths for Female.

Next, we substitute these values into the entropy equation. For each class, we multiply its probability by the base two logarithm of the same probability. The values from the logarithms are precomputed, and their multiplications are carried out.

Finally, summing the computed values and applying the negative sign gives us an entropy of zero point nine seven. This value tells us that the dataset has a relatively high level of uncertainty, meaning it is not entirely pure. If all instances belonged to the same class, entropy would be zero, indicating no uncertainty. Since we have both Male and Female instances, the entropy is high, and we must perform a split to reduce the uncertainty.

Now we compute entropy after splitting on each feature. We begin with Age.

NOTE TO EDITOR: Transition to Formula

For Age = 35:

$$H(35) = -\left(\frac{1}{3}\log_2\frac{1}{3} + \frac{2}{3}\log_2\frac{2}{3}\right)$$

$$H(35) = -\left(0.333 \times -1.585 + 0.666 \times -0.585\right)$$

$$H(35) = 0.918$$

For Age = 25 and Age = 40:

$$H(25) = H(40) = 0$$

Weighted entropy after splitting on Age:

$$H(S|\text{Age}) = \frac{1}{5}(0) + \frac{3}{5}(0.918) + \frac{1}{5}(0) = 0.551$$

Information gain:

$$\text{Gain}(S, \text{Age}) = H(S) - H(S|\text{Age}) = 0.970 - 0.551 = 0.419$$

To assess whether Age is a good splitting feature, we compute entropy for each subset formed by splitting on Age. If all the instances in a subset belong to the same class, the entropy for that subset is zero, indicating no uncertainty.

We begin with the subset where Age is thirty-five. This group contains three individuals, one male and two females. Using the entropy formula, we calculate the uncertainty within this subset by applying the probability of each class and computing the logarithmic terms. After performing the calculations, the entropy for this subset is approximately zero point nine one eight.

Next, we analyze the subsets where Age is twenty-five and forty. Since each of these groups contains only one individual, they are pure, meaning their entropy is zero.

To determine the overall entropy after the split, we compute the weighted sum of the individual entropies. Each subset contributes to the final entropy proportionally to its size in the dataset. The result of this weighted sum is zero point five five one, showing that the data is more homogeneous after splitting.

Finally, we compute the information gain, which tells us how much uncertainty is reduced by splitting on Age. This is obtained by subtracting the new entropy from the original dataset entropy. The information gain is zero point four one nine. The higher this value, the better the split, as it results in greater purity in the resulting subsets.

Now we compute entropy after splitting on Salary.

NOTE TO EDITOR: Transition to Formula

Splitting on Salary:

$$H(S|\text{Salary}) = 0.550$$

$$\text{Gain}(S, \text{Salary}) = H(S) - H(S|\text{Salary}) = 0.970 - 0.550 = 0.420$$

To evaluate whether Salary is a better splitting feature, we measure how entropy changes when the dataset is divided based on different salary levels. The goal is to determine whether splitting on Salary results in purer subsets compared to other features.

After performing the entropy calculation for subsets formed by splitting on Salary, the total weighted entropy is found to be zero point five five zero. This means that the data is less mixed than before, as entropy has decreased.

To quantify the improvement, we compute the information gain. This is done by subtracting the new entropy from the original dataset entropy. The result is an information gain of zero point four two zero. This tells us how much uncertainty has been removed by using Salary as a split.

Since the information gain for Salary is slightly higher than that for Age, this means Salary results in the purest possible subsets at this stage of the decision tree. Therefore, the best first split is made on Salary. Once this split is established, further divisions can be made based on the remaining features.

By using entropy and information gain, we systematically ensure that each decision in the tree-building process leads to the greatest possible reduction in uncertainty, improving the tree's ability to classify future data accurately.

On screen is the visualization you can get from the Scikit Learn Decision Trees API, doing the same splits we did earlier, though their alogrithm split on the salary only because as we calculated, it had the best information gain:

NOTE TO EDITOR: Transition to Image "Salary Decision Tree"

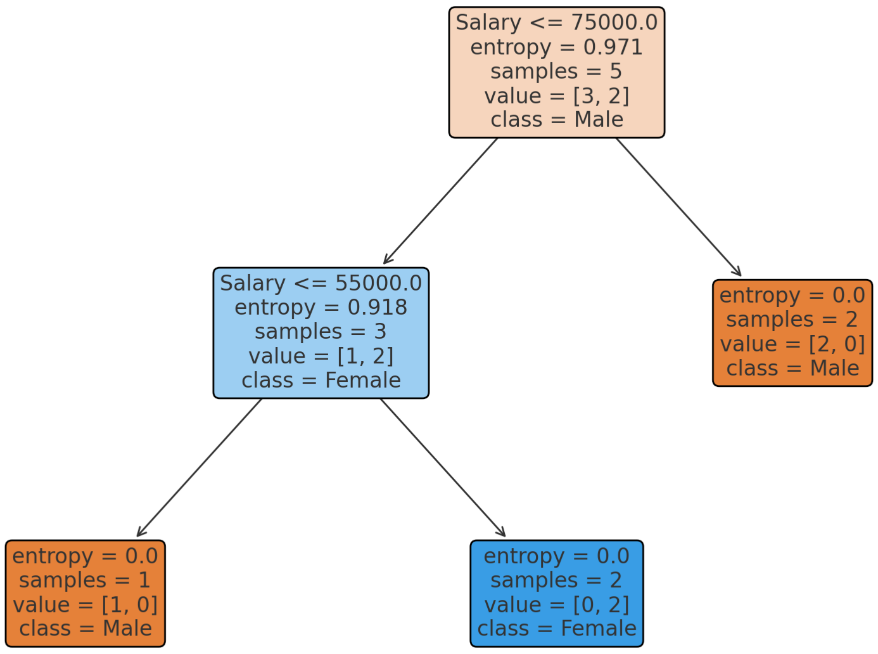

Figure 5: Salary Decision Tree

We'd also show you the decision boundaries, but for you who have paid attention to our lessons, you know it will be very skewed and why. Hint, the millionaire outlier.

Another question you should be asking is when do we stop splitting. From the toy dataset its obvious, when we run out of data but in general its when the maximum depth of the tree is reached or when the maximum number of leaves are achieved, or when a minimum number of observations are left in a node. On that last one, if the minimum number of observations left is 1, then we have probably overfit.

The process we showed you is called recursive partitioning, and a commonly used algorithm for this is CART short for Classification And Regression Trees. The CART algorithm follows a structured approach to decision tree construction by selecting the best feature at each node and determining the optimal split point based on an evaluation criterion. The tree is built recursively until a stopping criterion is met, ensuring that the model does not become overly complex. A key feature of CART is that it always produces binary trees, meaning each node splits into exactly two branches.

NOTE TO EDITOR: Transition to Algorithm Steps

1. Start at the root node containing the full dataset.
2. Identify the best feature to split on by evaluating all possible splits using a cost function.
3. Perform recursive binary splitting, dividing the dataset into two subsets at each step.
4. Continue splitting nodes until a predefined stopping criterion is met.
5. Apply pruning techniques to simplify the tree and prevent overfitting.
6. Finalize the model, ensuring it generalizes well to unseen data.

The process begins at the root node, where all data points are initially grouped together. The algorithm evaluates each feature and determines an optimal split point by assessing all possible values for continuous variables or distinct categories for categorical variables. This is done using a cost function that measures impurity reduction. For classification problems, impurity can be measured using entropy, the Gini index, or information gain. For regression problems, impurity is commonly assessed using residual sum of squares. The split that minimizes impurity the most is selected, and the dataset is divided into two subsets accordingly. This recursive binary splitting continues at each child node, repeating the process of selecting the best feature and splitting the data further.

As the tree grows, it becomes increasingly complex, which can lead to overfitting. To address this, the algorithm applies stopping criteria such as setting a minimum number of samples required to split a node or defining a maximum depth for the tree. If further splitting does not significantly improve the model, the node is designated as a leaf, where final classifications or predictions are made. Once the tree is fully built, pruning techniques are used to refine its structure. The goal of pruning is to remove unnecessary splits that do not contribute significantly to predictive accuracy. This is typically done by evaluating the tree's performance on a validation set and systematically removing nodes that do not improve generalization.

Unlike some machine learning models, CART does not require extensive preprocessing. It can handle both categorical and continuous features without transformation. However, categorical features may need to be encoded numerically before being input into the model. By following this structured process, CART produces decision trees that are not only effective at making predictions but also interpretable, making them widely used in applications where model transparency is essential. The final tree structure depends on the selected evaluation criteria, stopping conditions, and pruning strategy, ensuring that the model balances complexity with predictive performance.

Regression trees in the CART framework function similarly to classification trees but differ in their splitting criteria. Instead of minimizing impurity measures like entropy or the Gini index, regression trees aim to minimize the Mean Squared Error (MSE). The goal is to partition the training set in a way that reduces the variance of the target variable within each subset, leading to more precise numerical predictions.

NOTE TO EDITOR: Transition to Formula

$$J(k, t_k) = \frac{m_{\text{left}}}{m}\text{MSE}_{\text{left}} + \frac{m_{\text{right}}}{m}\text{MSE}_{\text{right}}$$

where

$$\text{MSE}_{\text{node}} = \sum_{i\in\text{node}} \left(\hat{y}_{\text{node}} - y^{(i)}\right)^2$$

$$\hat{y}_{\text{node}} = \frac{1}{m_{\text{node}}} \sum_{i\in\text{node}} y^{(i)}$$

The algorithm starts at the root node and iteratively selects the best feature and split point to minimize the MSE across all possible partitions. The function J of K and T represents the cost function used to evaluate each potential split, where k is the feature index and T K is the split threshold. The algorithm chooses the split that results in the lowest weighted sum of the MSE for the left and right child nodes.

MSE for each node is calculated by summing the squared differences between the predicted value at that node and the actual target values. The predicted value at each node Y Hat, is simply the mean of the target values in that node. This ensures that each split creates subgroups that are as homogeneous as possible in terms of their target values.

Once the best split is determined, the dataset is divided into two subsets, and the process repeats recursively for each child node. This continues until a stopping criterion is met, such as a minimum number of samples in a node or a maximum tree depth.

The primary advantage of regression trees is their ability to capture nonlinear relationships in the data without requiring feature transformations. However, they are prone to overfitting if the tree grows too deep. To address this, pruning techniques are applied, where branches that do not significantly improve prediction accuracy on a validation set are removed. This helps maintain a balance between model complexity and predictive performance.

By structuring the decision-making process around minimizing variance, regression trees provide an intuitive way to model numerical outcomes while maintaining interpretability.

To build a regression tree using the CART algorithm, we need to determine the best splits based on minimizing the Mean Squared Error (MSE). We will go through the step-by-step process of constructing the tree, carefully explaining each decision along the way.

NOTE TO EDITOR: Transition to Table

ID	Age	Salary	Gender
1	25	$50,000$	Male
2	35	$60,000$	Female
3	40	$80,000$	Male
4	35	$70,000$	Female

The goal of regression trees is to split the data in a way that minimizes the variance of the target variable within each node. In this case, we assume the target variable is Salary, and we aim to create splits that reduce the spread of Salary values in each resulting subset.

To evaluate the quality of a split, we compute the Mean Squared Error (MSE) before and after the split. The formula for MSE is:

NOTE TO EDITOR: Transition to Formula

$$\text{MSE}_{\text{node}} = \frac{1}{m_{\text{node}}} \sum_{i \in \text{node}} (y^{(i)} - \hat{y}_{\text{node}})^2$$

$$\hat{y}_{\text{node}} = \frac{1}{m_{\text{node}}} \sum_{i \in \text{node}} y^{(i)}$$

To begin, we calculate the overall MSE before any splits. The mean Salary across all data points is:

NOTE TO EDITOR: Transition to Formula

$$\hat{y}_{\text{root}} = \frac{50,000 + 60,000 + 80,000 + 70,000}{4} = 65,000$$

Now, we compute the variance of Salary around this mean:

NOTE TO EDITOR: Transition to Formula

$$\begin{aligned}\text{MSE}_{\text{root}} = \frac{1}{4}\Big(&(50,000 - 65,000)^2 + \\ &(60,000 - 65,000)^2 + \\ &(80,000 - 65,000)^2 + \\ &(70,000 - 65,000)^2\Big)\end{aligned} \tag{1}$$

$$\text{MSE}_{\text{root}} = \frac{1}{4}(225,000,000 + 25,000,000 + 225,000,000 + 25,000,000)$$

$$\text{MSE}_{\text{root}} = \frac{500,000,000}{4} = 125,000,000$$

This is the overall error if we do not split the data. Our goal is to find a feature and a threshold that will split the dataset into two subsets, reducing this error.

We now evaluate possible splits on Age. If we split at Age less than or equal to 35 and Age greater than 35, we get two groups:

NOTE TO EDITOR: Transition to Formula

$$\text{Left subset:} \quad \{(25, 50,000), (35, 60,000), (35, 70,000)\}$$

$$\text{Right subset:} \quad \{(40, 80,000)\}$$

We compute the mean salary for each subset:

NOTE TO EDITOR: Transition to Formula

$$\hat{y}_{\text{left}} = \frac{50,000 + 60,000 + 70,000}{3} = 60,000$$

$$\hat{y}_{\text{right}} = 80,000$$

Then, we calculate the MSE for each subset:

NOTE TO EDITOR: Transition to Formula

$$\text{MSE}_{\text{left}} = \frac{1}{3}\left((50,000 - 60,000)^2 + (60,000 - 60,000)^2 + (70,000 - 60,000)^2\right)$$

$$= \frac{1}{3}(100,000,000 + 0 + 100,000,000) = \frac{200,000,000}{3} = 66,666,667$$

$$\text{MSE}_{\text{right}} = 0$$

Now, we compute the weighted average MSE:

NOTE TO EDITOR: Transition to Formula

$$J(Age \leq 35) = \frac{3}{4} \times 66,666,667 + \frac{1}{4} \times 0$$

$$= 50,000,000$$

Since the weighted MSE after the split is lower than the original root MSE, this suggests that splitting at Age less than or equal to 35 provides a reduction in error, making it a strong candidate for the first split in the regression tree. This step-by-step calculation shows how a regression tree selects the best split by minimizing variance within each subset. Each candidate feature is evaluated, and the split that results in the greatest reduction in error is chosen. The process is then repeated recursively until a stopping condition, such as a minimum node size, is reached.

By iteratively applying this process, the regression tree can effectively model relationships within the data, ensuring that each subset has minimal variance in the target variable.

Using Scikit learn APIs, the regression tree plotted is shown on your screen that reflects the steps we just went through:

NOTE TO EDITOR: Transition to Image "Salary Decision Tree Regression"

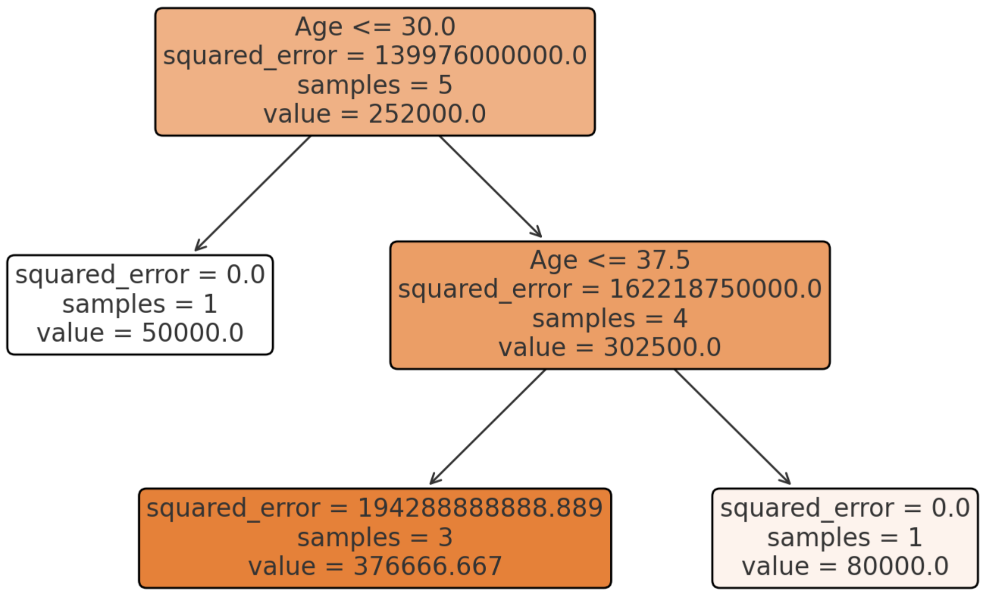

Figure 6: Salary Decision Tree Regression

A word of warning though, a regression tree yields non-smooth step-wise predictions, which means it cannot extrapolate as every leaf produces a mean. Have a look at the plot from Scikit Learn's API documentation on a regression tree on your screen:

NOTE TO EDITOR: Transition to Image "Regression Tree Line"

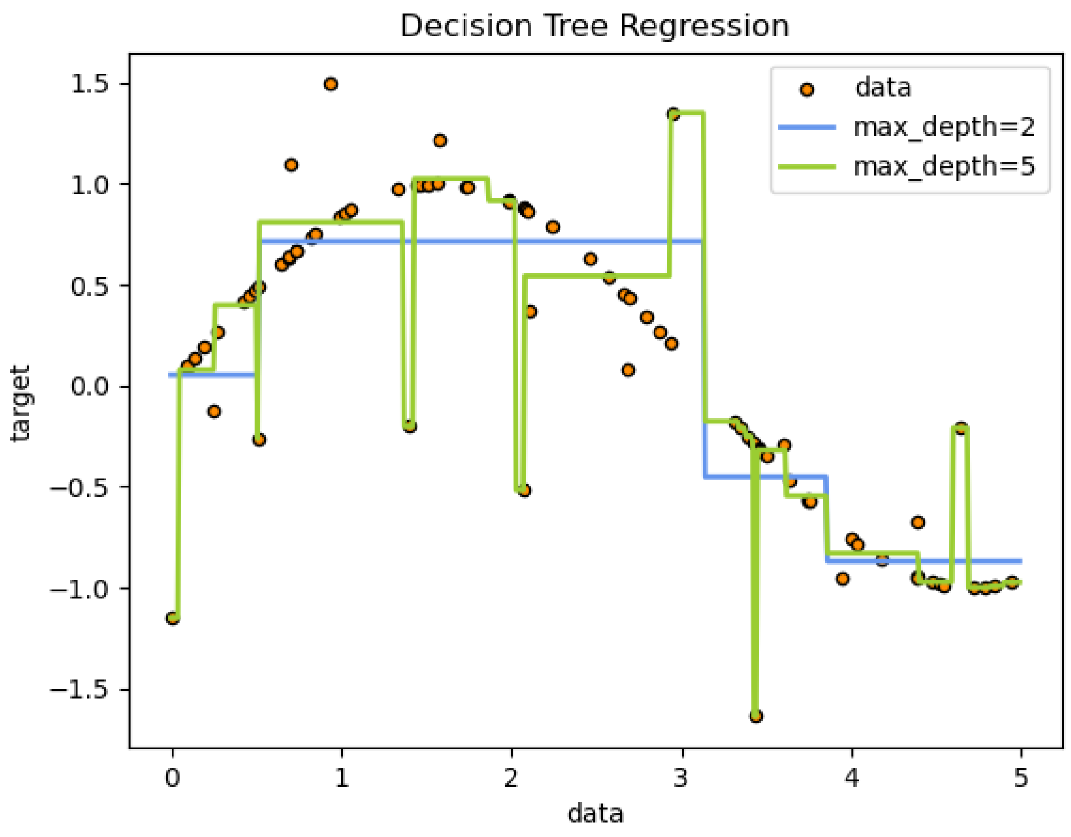

Figure 7: Regression Tree Line

The regression line is not your standard continuous regression line, but steps and angles. In truth, scitkit learn uses this plot to show case overfitting in tree as a model with a depth of 5 (yellow line) learns the details of the training data to the point that it overfits to the noise and the other with a depth of 2 (blue line) learns the major tendencies in the data well. As the CART algorithm constructs decision trees by recursively splitting the data to minimize impurity in classification tasks or variance in regression tasks it can allow the tree to grow indefinitely, becoming too complex, and capturing noise rather than meaningful patterns in the data. This problem is as you know: overfitting. To address this issue, the CART algorithm applies a technique known as pruning, which simplifies the tree while maintaining its predictive power.

Pruning is the process of reducing the size of a decision tree by removing sections that provide little predictive value. The objective is to improve generalization by eliminating unnecessary splits that do not significantly improve model performance on new data. When a decision tree is pruned, it becomes less sensitive to minor fluctuations in the dataset, making it more robust and capable of handling unseen data.

CART uses two main types of pruning: pre-pruning and post-pruning.

Pre-pruning, also called early stopping, halts the tree growth before it reaches full depth. This is done by setting predefined conditions, such as a maximum tree depth, a minimum number of samples per node, or a minimum reduction in impurity required for a split. If further splitting does not meet these conditions, the node is converted into a leaf. The advantage of pre-pruning is that it prevents the tree from growing unnecessarily large, reducing computational cost and avoiding extreme overfitting. However, the risk of pre-pruning is that the tree may be stopped too early, leading to an underfitting model that fails to capture important patterns in the data.

Post-pruning, also called cost-complexity pruning, first allows the tree to grow fully and then removes branches that do not significantly improve performance. This method evaluates the contribution of each node to the overall accuracy and prunes those that introduce unnecessary complexity. Unlike pre-pruning, post-pruning ensures that the model captures all possible patterns before making decisions about reducing tree complexity.

To quantify whether a branch should be pruned, CART uses a cost-complexity measure, which balances model accuracy and simplicity. The cost function is defined as the function on the screen:

NOTE TO EDITOR: Transition to Formula

$$R_\alpha(T) = R(T) + \alpha|L(T)|$$

where

$$R(T) = \sum_{t \in L(T)} r(t) \cdot p(t) = \sum_{t \in L(T)} R(t)$$

In this formula, the term R of T represents the total error of the tree. If we are dealing with a classification problem, this error is measured as the misclassification rate, which tells us how often the model predicts the wrong class. If we are working with a regression tree, the error is measured using the mean squared error, which calculates how far the predicted values deviate from the actual values on average.

The function L of T represents the set of leaf nodes in the tree. Leaf nodes are the final decision points where predictions are made. The larger the number of leaf nodes, the more complex the tree, meaning it has more decision rules and captures finer patterns in the data.

The equation for R of T provides a more detailed breakdown of how the total error is computed. The summation iterates over all leaf nodes in L of T, where each leaf node t contributes a weighted error term. The function $r(t)$ represents the misclassification rate (or the mean squared error in regression), while $p(t)$ represents the probability that a sample reaches that particular leaf. The expression R of T Dot P of T gives the contribution of each node to the overall error, and summing over all leaf nodes gives the total error. This can also be rewritten using R of T, where R of T is simply the weighted error at node t, ensuring consistency in notation.

The term Alpha is a regularization parameter that controls how much we penalize tree complexity. If Alpha is set to zero, the model does not penalize complexity, meaning it will keep all the branches and grow as deep as possible. However, if Alpha is increased, the penalty for having too many leaves becomes stronger, encouraging the algorithm to remove unnecessary branches and create a simpler tree.

By adding Alpha times the norm of L of T to the cost function, the model is forced to balance two objectives: minimizing error while keeping the tree as simple as possible. If the tree is too complex, it may memorize the training data and fail to generalize to new examples, a problem known as overfitting. On the other hand, if the tree is too simple, it may fail to capture important patterns, leading to underfitting. The goal of pruning is to find the right balance by selecting an appropriate value of Alpha.

To understand how post-pruning works, consider the process of pruning subtrees. At each step, the cost-complexity function is evaluated before and after removing a subtree T_t. The variation in the cost function due to pruning is given by the equations on your screen:

NOTE TO EDITOR: Transition to Formula

$$R_\alpha(T - T_t) - R_\alpha(T) = R(T - T_t) - R(T) + \alpha(|L(T - T_t)| - |L(T)|)$$

$$g(t) = \frac{R(t) - R(T_t)}{|L(T_t)| - 1}$$

$$\alpha_{t+1} = g(t_1) = \frac{R(t_1) - R(T_{t_1})}{|L(T_{t_1})| - 1}$$

The first equation calculates how much the cost-complexity function changes when a subtree is removed. The left-hand side of the equation represents the difference in cost-complexity before and after pruning. The right-hand side consists of two terms: the first term represents the reduction in total error after pruning, while the second term accounts for the complexity penalty associated with having fewer leaves in the tree. If removing a subtree reduces the cost-complexity function, it means that the subtree did not contribute substantial predictive value, making pruning a justified step.

The second equation defines the function G of T, which determines which subtree should be pruned first. This function measures how much the error is reduced per removed leaf node. It is calculated as the difference between the error at node t and the total error of the subtree rooted at t, divided by the number of leaf nodes removed. A smaller G of T value suggests that pruning this subtree simplifies the tree structure without significantly increasing error.

A key part of the pruning process is determining the regularization parameter Alpha sub T plus 1, which is given by the third equation. This parameter is set to the smallest G of T value across all nodes in the tree at a given iteration. It acts as a threshold that decides whether a subtree should be pruned. If a subtree has an error reduction value G of T equal to Alpha sub T plus 1, it is pruned because its contribution to reducing error is minimal compared to the complexity it introduces. This pruning process is applied iteratively, selecting and pruning the subtree with the smallest G of T at each step until no further pruning improves the model.

The pruning process ensures that the final decision tree remains interpretable and generalizable, preventing overfitting while preserving the most meaningful patterns in the data. If pruning a subtree decreases the cost-complexity function, it means that removing the subtree improves generalization without significantly compromising predictive accuracy. The process is repeated iteratively, trimming unnecessary branches while maintaining the tree's ability to make accurate predictions.

The pruning process follows an iterative algorithm:

NOTE TO EDITOR: Transition to Algo

1. Start with the fully grown tree T_1, which was obtained by minimizing the error function $R(T)$.
2. Identify the node t that minimizes the error reduction function $g(t)$.
3. Let $\alpha_{t+1} = g(t_1)$, which defines the threshold for pruning, and remove the corresponding subtree.
4. Repeat this process iteratively, selecting the next node for pruning until either the root is reached or further pruning does not improve performance.
5. This results in a sequence of trees $T_1 \supseteq T_2 \supseteq \cdots \supseteq T_k$.

The result of this algorithm is a series of pruned trees of varying sizes, each representing a different level of model complexity. To select the best tree, the algorithm determines the optimal value of Alpha using cross-validation. This involves computing the validation error for each pruned tree and selecting the one with the lowest error.

In practice, pruning is particularly useful when dealing with small datasets or noisy data, where overfitting is a frequent issue. Deep decision trees tend to memorize specific details of the training set instead of learning broader patterns. By applying pruning, CART ensures that decision trees remain powerful and generalizable for both classification and regression tasks, avoiding the drawbacks of excessive complexity.

Pruning also improves interpretability. A fully grown tree may contain many unnecessary splits, making it difficult to understand the decision-making process. By simplifying the tree, pruning highlights only the most important decision rules, ensuring greater transparency. This is particularly valuable in fields such as healthcare and finance, where explainability is critical.

By balancing model complexity and accuracy, pruning enhances generalization, ensuring that decision trees perform well not only on training data but also on unseen examples.

While trees are simple, they're very powerful, especially when combined together into an ensemble, which is the topic of the next lesson.

A small notbook is available for you to build a tree from ground up. the URL is on your screen:

NOTE TO EDITOR: Show the URL

`https://colab.research.google.com/drive/12zX5SBlhiieEFAO39m-qzrw3_YTIyauT?usp=sharing`