Lecture 5

Decision Trees
Random Forest

Creating a **decision tree** by hand or using a programming language like R involves several steps, including data preprocessing, selecting a splitting criterion, and recursively building the tree. Here, I'll provide a high-level overview of the process, but keep in mind that decision tree algorithms in R are typically automated, and manual construction is uncommon. You can use libraries like "rpart" or "tree" in R to build decision trees from data. However, manual construction helps to illustrate how the algorithm works. We'll look at coding it up in R later.

Manual Decision Tree Construction:
Data Preparation: Gather and preprocess your dataset, ensuring that it's clean and organized. This may involve handling missing values, encoding categorical variables, and splitting the data into training and testing sets.

Select the Root Node: Choose a feature (attribute) that will serve as the root node for your decision tree. The selection can be based on domain knowledge, or you can calculate a measure like Gini impurity or entropy to determine the best attribute to split on.

Create Child Nodes: For each possible outcome of the root node feature, create child nodes representing the next set of decisions. These child nodes are typically created based on a condition that maximizes information gain (minimizes impurity) using metrics like Gini impurity or entropy.

Repeat for Child Nodes: For each child node, continue the process recursively, selecting the best attribute to split on and creating further child nodes. This process continues until a stopping criterion is met. Stopping criteria may include reaching a maximum depth, having a node with pure class labels, or achieving a minimum number of data points in a leaf node.

Assign Class Labels: For each leaf node (terminal node), assign a class label based on the majority class of the data points in that node.

Pruning (Optional): After constructing the full tree, you can perform pruning to reduce the tree's size and improve generalization by removing nodes that don't contribute significantly to prediction accuracy. Not pruning can result in overfitting, making any test data perform more poorly.

Let's consider a simple decision tree problem for classifying whether to play tennis based on weather conditions. This is a classic example often used to illustrate decision tree concepts.

Problem Statement: You want to decide whether to play tennis or not, and your decision is based on the following three weather conditions:

- Outlook (Sunny, Overcast, Rainy)
- Temperature (Hot, Mild, Cool)
- Humidity (High, Normal)

You have historical data with labels indicating whether you played tennis or not in the past for different combinations of these conditions.

Here's a simplified dataset:

Outlook	**Temperature**	**Humidity**	**PlayTennis**
Sunny	Hot	High	No
Sunny	Hot	Normal	No
Overcast	Hot	High	Yes
Rainy	Mild	High	Yes
Rainy	Cool	Normal	Yes
Overcast	Cool	Normal	Yes
Sunny	Mild	High	No
Sunny	Cool	Normal	Yes
Rainy	Mild	Normal	No

Now, let's manually construct a decision tree to decide whether to play tennis:

Step 1: Choose the Root Node: Select the feature that provides the best split based on information gain, Gini impurity, or entropy. In this case, we'll use Gini impurity. The "Outlook" attribute has the lowest Gini impurity, so it becomes the root node. (We'll look at how to calculate this later.)

Step 2: Create Child Nodes: For each possible value of "Outlook" (Sunny, Overcast, Rainy), create child nodes:

For Sunny: Check the Gini impurity for "Temperature" and "Humidity." "Temperature" (Hot, Mild, Cool) has the lowest Gini impurity, so it becomes the next decision node for Sunny.

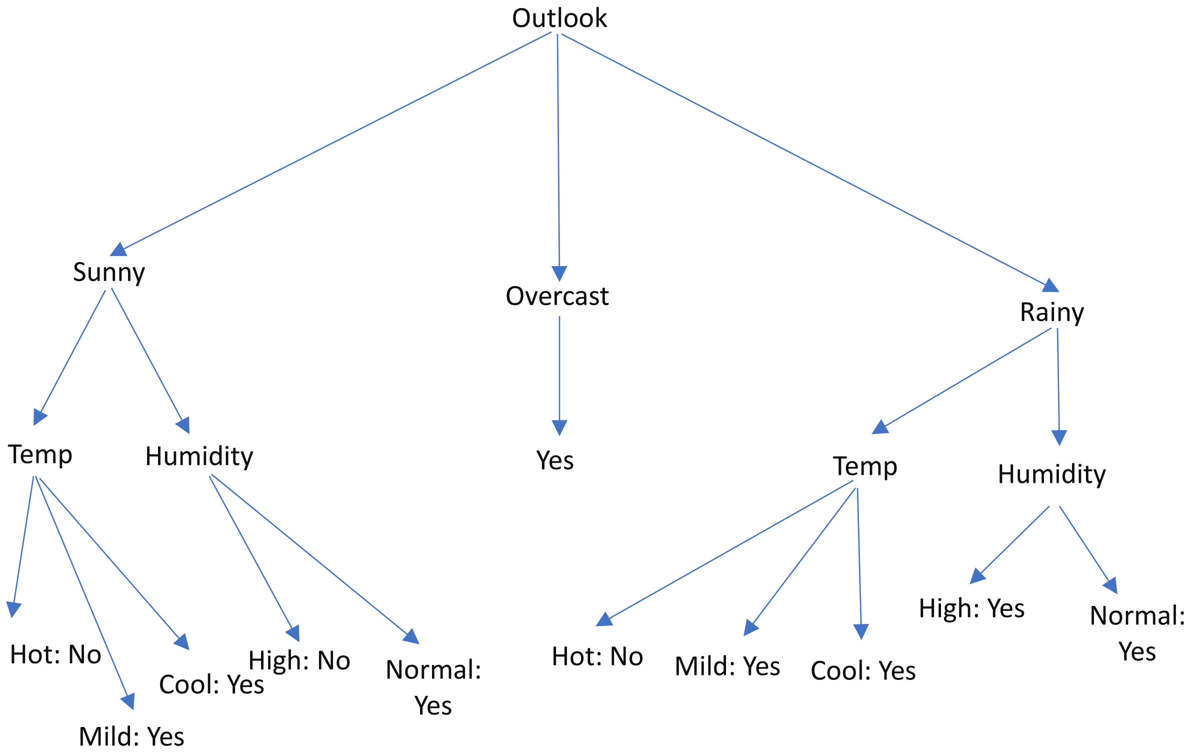
For Overcast: Play tennis (Yes) since it's always Yes when it's overcast.

For Rainy: Check the Gini impurity for "Temperature" and "Humidity." "Temperature" has the lowest Gini impurity, so it becomes the next decision node for Rainy.

Step 3: Continue Creating Child Nodes: Continue creating child nodes based on Gini impurity until you reach a stopping criterion. In this simple example, we've reached the stopping criterion when we've classified all instances.

Here's the resulting decision tree:

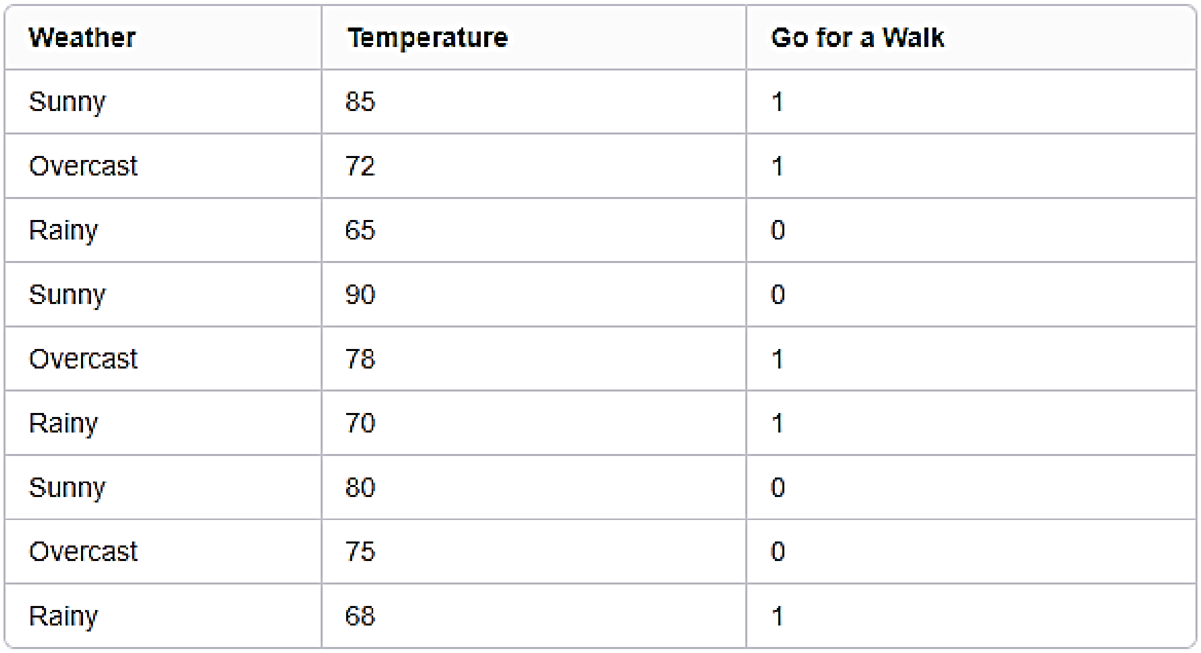
```
Outlook (Root)
  |
  +-- Sunny
  |   |
  |   +-- Temperature
  |   |   |
  |   |   +-- Hot: No
  |   |   +-- Mild: Yes
  |   |   +-- Cool: Yes
  |   |
  |   +-- Humidity
  |       |
  |       +-- High: No
  |      +-- Normal: Yes
  |
  +-- Overcast: Yes
  |
  +-- Rainy
     |
     +-- Temperature
     |   |
     |   +-- Hot: No
     |   +-- Mild: Yes
     |   +-- Cool: Yes
     |
     +-- Humidity
         |
         +-- High: Yes
         +-- Normal: Yes
```



In this decision tree, you can follow the path from the root node to make a decision on whether to play tennis or not based on the given weather conditions. The leaf nodes represent the final decisions, with "Yes" indicating playing tennis and "No" indicating not playing tennis.

We can also build decisions trees from numerical examples.

In this example, we'll use a small dataset related to whether people go for a walk based on two features: "Weather" and "Temperature." The target variable is binary: "Go for a walk" (1) or "Not go for a walk" (0).



Weather	Temperature	Go for a Walk
Sunny	85	1
Overcast	72	1
Rainy	65	0
Sunny	90	0
Overcast	78	1
Rainy	70	1
Sunny	80	0
Overcast	75	0
Rainy	68	1

Let's manually create a decision tree based on this data:

Step 1: *Choose the Root Node:* We choose the root node based on the feature that best splits the data. Let's start with the "Weather" feature.

Step 2: *Split the Data Based on the Chosen Feature:* We split the data into three subsets: Sunny, Overcast, and Rainy.

Step 3: *Evaluate Each Subset*: For the Sunny subset:

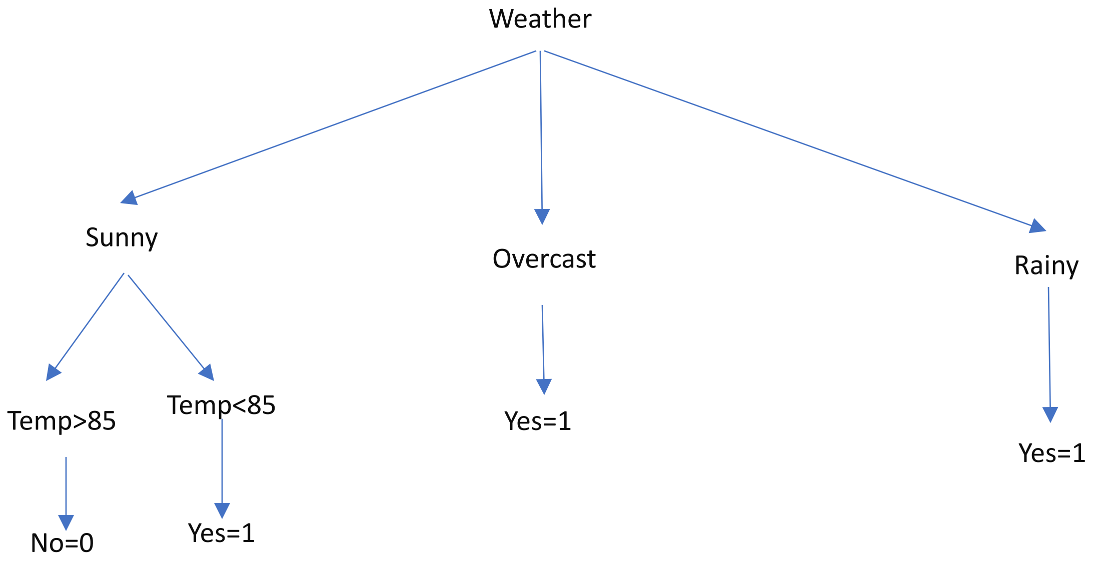
If Temperature <= 85, then Go for a Walk (1).
If Temperature > 85, then Not go for a Walk (0).

For the Overcast subset:
Always Go for a Walk (1).

For the Rainy subset:
Always Go for a Walk (1).

```
If Weather = Sunny
├── If Temperature <= 85
|   └── Go for a Walk (1)
├── If Temperature > 85
|   └── Not go for a Walk (0)
```

If Weather = Overcast
└── Go for a Walk (1)
If Weather = Rainy
└── Go for a Walk (1)



Creating a Decision Tree in R:
In R, you can create decision trees using specialized packages like "rpart" and "tree." Here's a basic example using the "rpart" package:

```
# Load the rpart package
library(rpart)

# Create a decision tree model
model <- rpart(Class ~ Feature1 + Feature2 + Feature3, data =
your_data)

# Visualize the decision tree
plot(model)
text(model)
```

In the above code, "Class" is the target variable you want to predict, and "Feature1," "Feature2," and "Feature3" are the input features. You can load your dataset into R, replace "your_data" with your dataset's name, and adjust the target variable and features accordingly.

To further customize the tree-building process or modify stopping criteria, you can use additional arguments in the "rpart" function.

The **Random Forest** algorithm is an ensemble learning method used for classification and regression tasks. It builds a "forest" of decision trees, where each tree is trained on a random subset of the data, and the final prediction is determined by aggregating the results of these individual trees. Random Forest is known for its high predictive accuracy and robustness. Here's how the Random Forest algorithm works:

Algorithm Steps:
Bootstrap Aggregating (Bagging): Random Forest uses a technique called bagging. It creates multiple subsets (samples) of the training data by random sampling with replacement. This means that some data points may appear more than once in a subset, while others may not appear at all. These subsets are called "bootstrapped samples."

Random Feature Selection: For each decision tree in the forest, a random subset of features (input attributes) is selected for training. This is typically done to decorrelate the trees and promote diversity in the forest.

Tree Construction: Each decision tree is constructed independently using the bootstrapped sample and the selected subset of features. At each node of the tree, a feature is chosen for splitting based on some criterion (e.g., Gini impurity or entropy for classification, mean squared error for regression). However, only a random subset of features is considered at each node.

Tree Growth: The decision trees continue to grow until a stopping criterion is met. Common stopping criteria include reaching a maximum depth, having a minimum number of data points in a leaf node, or when further splitting does not improve the tree's performance significantly.

Voting (Classification) or Averaging (Regression): For classification tasks, each tree "votes" for a class, and the class with the majority of votes becomes the predicted class for the Random Forest. For regression tasks, the predictions of all trees are averaged to obtain the final prediction. (It’s uncommon to use Random Forest for regression as it tends to product piecewise graphs.)

Key Characteristics and Advantages:
Diversity of Trees: By training each tree on a different random subset of data and a random subset of features, Random Forest introduces diversity among the trees. This diversity reduces overfitting and increases the model's robustness.

Handling Overfitting: The aggregation of multiple trees through majority voting or averaging helps reduce overfitting and improves the model's generalization performance.

Feature Importance: Random Forest provides a measure of feature importance. It assesses the importance of each feature by examining how much it contributes to the reduction in the Gini impurity or mean squared error across the forest.

Robustness: Random Forest is robust to noisy data and outliers, making it a good choice for real-world, imperfect datasets.

Parallelization: The construction of each decision tree in the forest is independent, making it well-suited for parallel processing, which can be important when the number of features to be considered is large, or the dataset itself has many observations.

Random Forest is a versatile algorithm used in various fields, including classification, regression, and feature selection. It is a popular choice in data mining and machine learning due to its strong performance and ease of use.

How do we determine the feature to split on? One way is the Gini index, and we'll look at how to calculate this value.

The Gini index (also known as the *Gini coefficient* or *Gini ratio*) is a measure of statistical dispersion used to quantify the level of inequality in a dataset, often applied to income distribution, wealth distribution, or other types of inequality. It varies between 0 and 1, with 0 representing perfect equality (everyone has the same income or wealth) and 1 representing perfect inequality (one individual or group has all the income or wealth).

The formula to calculate the Gini index is as follows:

$$G = 1 - \sum_{i=1}^{n} p_i^2$$

Where:
G is the Gini index.
n is the number of groups or categories (e.g., individuals or households).
p_i is the proportion of the total income, wealth, or other relevant variable held by the ith group.

To calculate the Gini index, follow these steps:

Determine the groups: Identify the groups or categories of interest. For example, if you're calculating income inequality, each group might represent an income bracket.

Calculate the proportion: For each group, calculate the proportion of the total income, wealth, or other variable that it holds. This proportion is represented by p_i.
Square the proportions: Square the proportions calculated in step 2 (p_i^2) for each group.

Sum the squared proportions: Add up all the squared proportions.

Calculate the Gini index: Subtract the sum of the squared proportions from 1 to obtain the Gini index.

Here's a step-by-step example of how to calculate the Gini index for a simple income distribution:
Suppose you have the following income distribution for a population with 5 groups:

Group	A	B	C	D	E
Income Percentage p_i	10%	20%	25%	30%	15%
p_i^2	0.01	0.04	0.0625	0.09	0.0225

$$\sum_{i=1}^{5} p_i^2 = 0.01 + 0.04 + 0.0625 + 0.09 + 0.025 = 0.285$$

Calculate the Gini index:

$$G = 1 - 0.285 = 0.715$$

So, in this example, the Gini index for this income distribution is 0.715, indicating relatively high income inequality. In the random forest example, we compare the Gini index for each possible choice of splitting criterion and then choose the highest Gini index value as the property to split on. With a numerical variable, you want to split at a point where one group is (ideally) 100% in one category so that you don't have to keep splitting it further, but this may not always be possible.

Resources:

1. https://www.datacamp.com/tutorial/decision-trees-R
2. https://www.guru99.com/r-decision-trees.html
3. https://www.r-bloggers.com/2021/04/decision-trees-in-r/
4. https://community.rstudio.com/t/decision-tree-in-r/5561
5. https://www.geeksforgeeks.org/decision-tree-in-r-programming/
6. https://www.tutorialspoint.com/r/r_decision_tree.htm
7. https://fderyckel.github.io/machinelearningwithr/trees-and-classification.html
8. https://www.listendata.com/2015/04/decision-tree-in-r.html
9. https://ademos.people.uic.edu/Chapter24.html
10. https://www.r-bloggers.com/2021/04/random-forest-in-r/
11. https://www.geeksforgeeks.org/random-forest-approach-in-r-programming/
12. https://www.simplilearn.com/tutorials/data-science-tutorial/random-forest-in-r
13. https://www.listendata.com/2014/11/random-forest-with-r.html
14. https://www.projectpro.io/recipes/perform-random-forest-r