



Institute of
Data

2022



Data Science and AI

Module 7 Part 1

Decision Trees and Ensemble Methods



Agenda: Module 7

- Overview
- Decision Trees
- Random Forests
- Ensemble Methods
 - Introduction – Bias vs Variance
 - Bagging
 - Boosting
 - Stacking



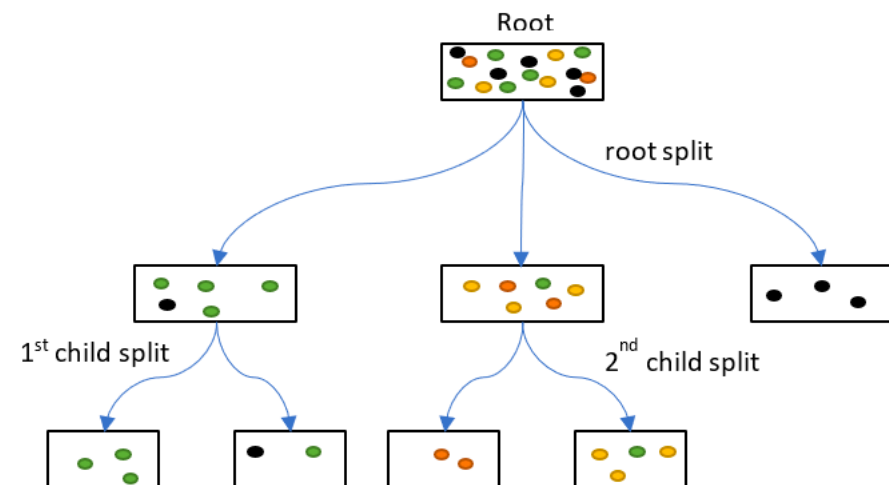
Decision Trees

- A type of **supervised** Machine Learning algorithm.
- It is one of the **most widely used** and practical methods for classification. It is the **simplest** and yet the **most powerful** algorithm!
- A decision tree is a tree in which each branch node represents **a choice between a number of alternatives**, and each **leaf node** represents an **outcome**.
- The type of the **target variable** defines the type of Decision Tree
 - **Categorical** variable Decision Tree: Has categorical target variable
 - **Continuous** variable Decision Tree: Has continuous target variable



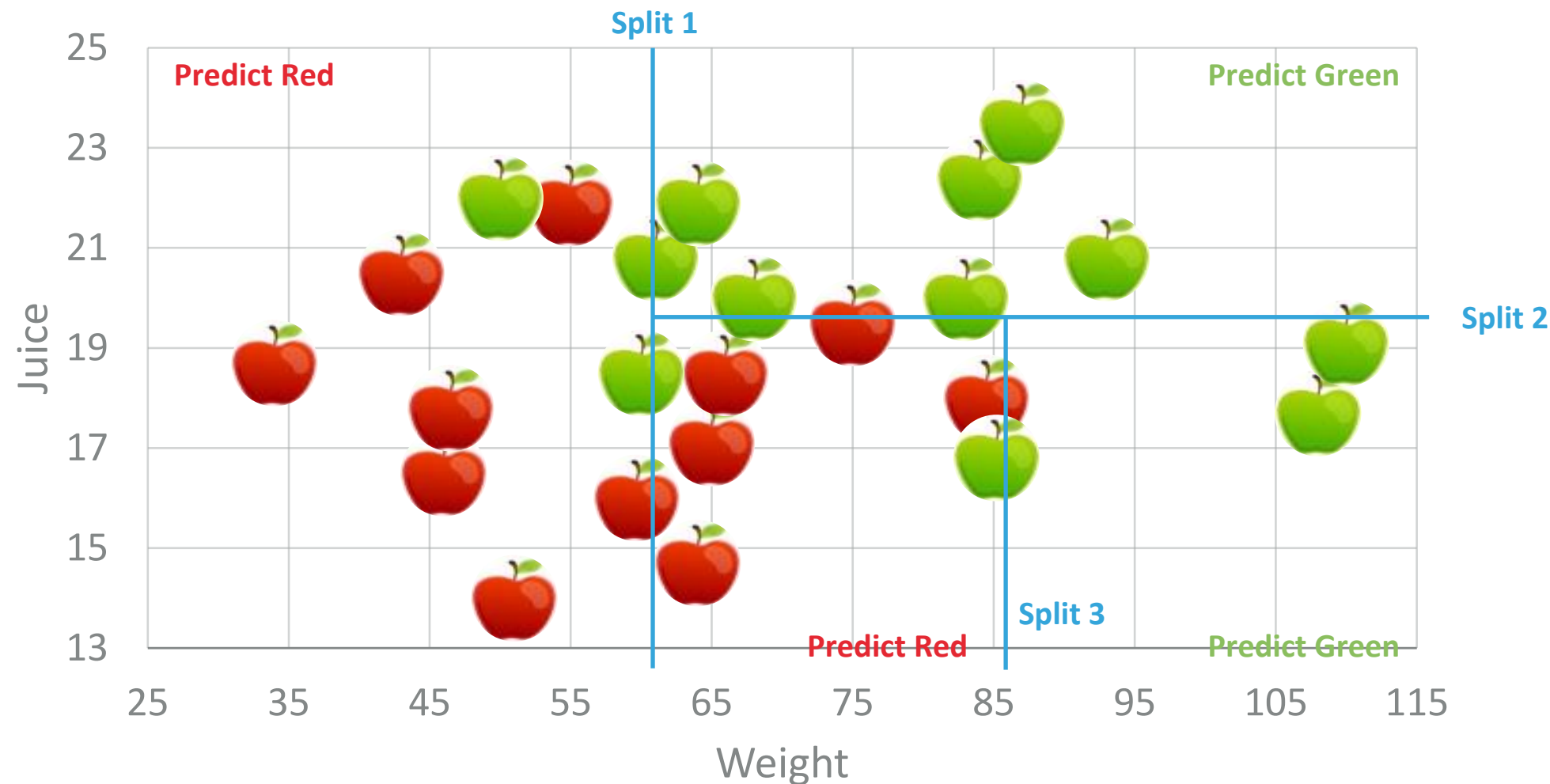
Constructing Decision Trees

- The approach is to: **Iteratively** separate data into **homogeneous groups**
 - Split data by category or range, **one feature at a time**
 - Into groups as **pure** as possible
 - repeating the process until a certain stop condition is reached



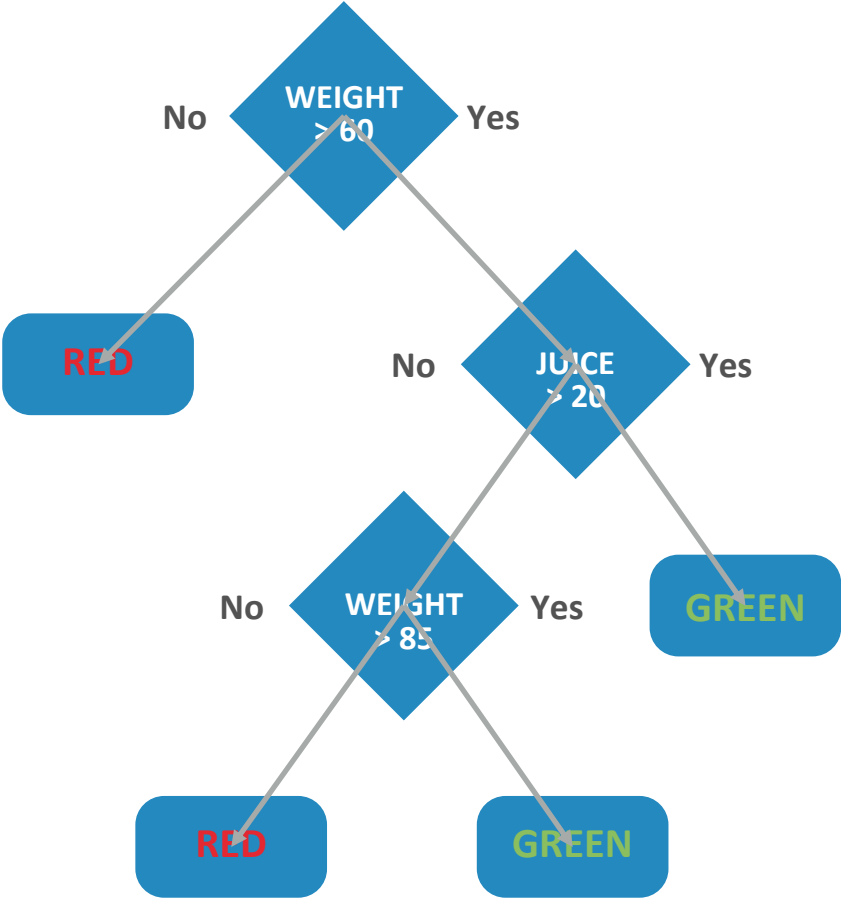
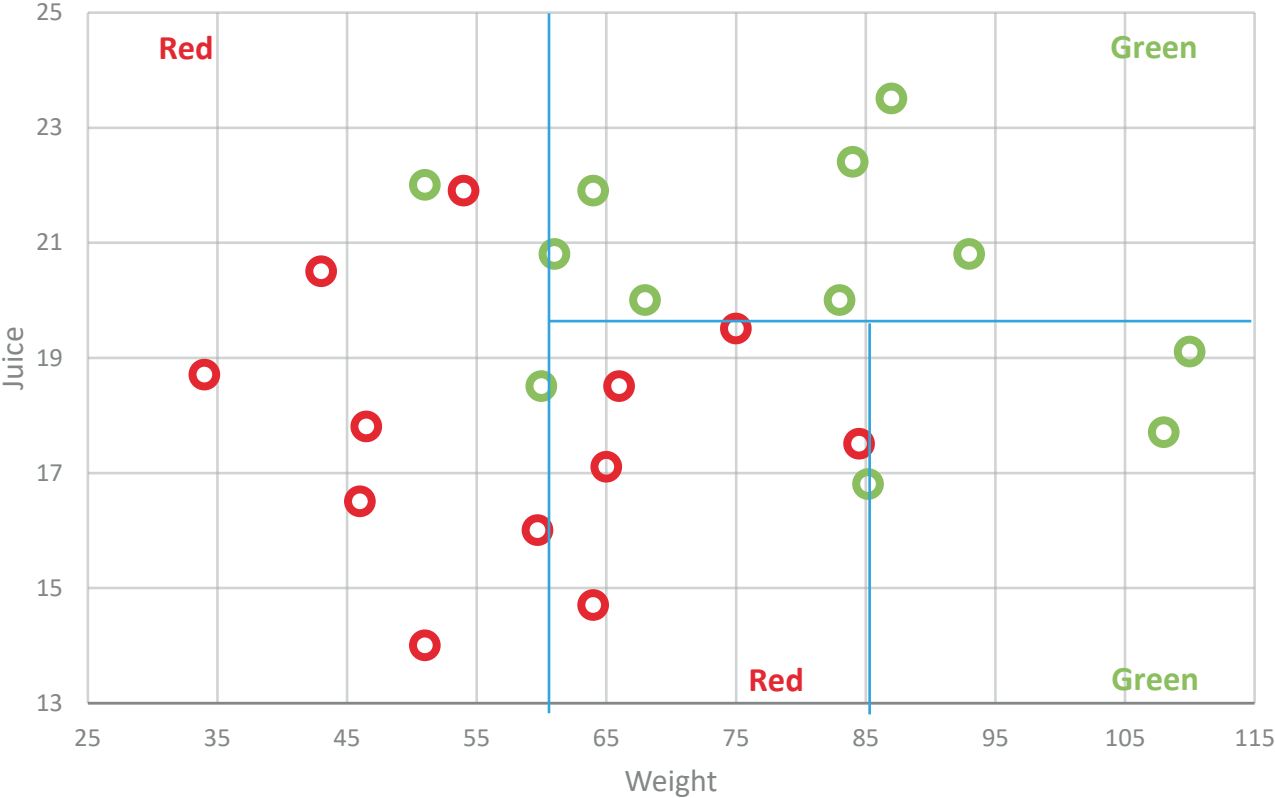


Iteratively separate data into homogeneous groups





Mapping the Decision Tree





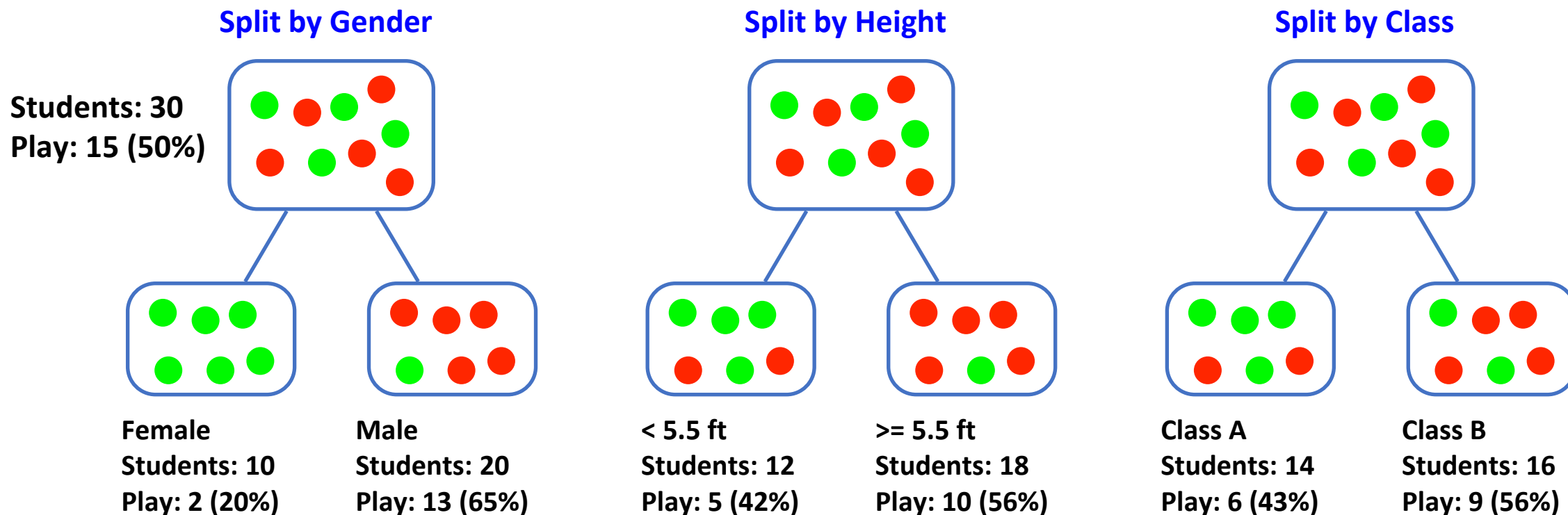
Constructing Decision Trees - Example 2

- Given 30 students with variables **Gender** (Boy, Girl), **Height** (5-6 ft) and **Class**(A, B). 15 out of these 30 play cricket in leisure time. **Predict** who plays cricket during leisure period?
- Need to segregate students who play cricket in their leisure time based on **highly significant** input variable among all three (gender, height, class).
- Decision Tree segregates students based on all values of the three variables and identifies the variable, which creates **the best homogeneous** sets of students (heterogeneous to each other).
- Decision tree identifies the **most significant variable** and its value which gives the best homogeneous sets of population.



Constructing Decision Trees - Example 2

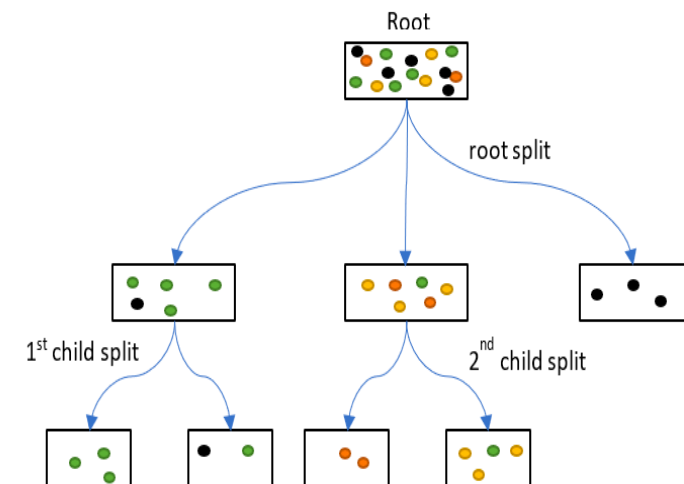
- The variable Gender can identify the **most homogeneous** sets compared to the other two variables





Terminology

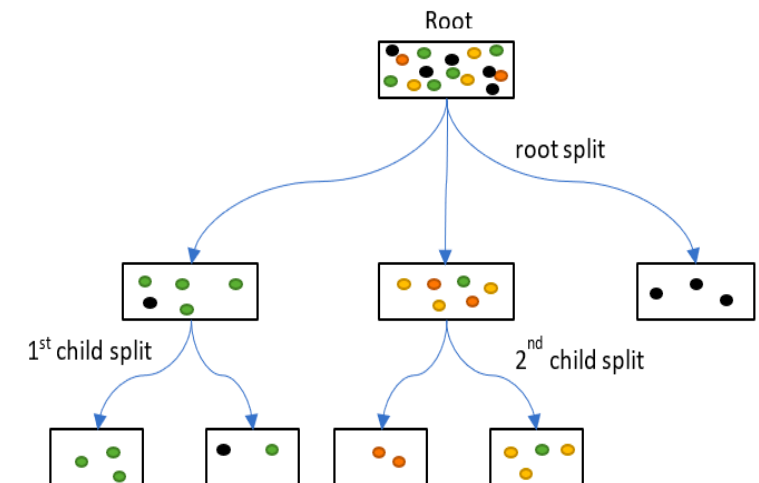
- **Root Node:** Represents all the data and gets divided into two or more homogeneous sets
- **Splitting:** Process of dividing a node into two or more sub-nodes
- **Decision Node:** When a sub-node splits into further sub-nodes
- **Leaf or Terminal Node:** Nodes that do not split





Terminology

- **Pruning**: Remove of sub-nodes of a decision node
- **Branch** or **Sub-Tree**: A subsection of the entire tree
- **Parent Node**: A node divided into sub-nodes is the parent of those sub-nodes
- **Child Node**: Sub-nodes resulting of a node division are children of such node





Constructing Decision Trees - Greedy Decision Tree Learning

1. Start with an *empty* tree
2. Select a *feature* to split the data
 - (Look for the feature that gives the *smallest classification error*)
3. For *each split* of the tree
 - If reached *stop condition* (purity, # elements, depth) then make predictions
 - Otherwise, go to 2. and continue recursively on this *split*



Advantages of Decision Trees

- **Less data cleaning required**: it can deal with **categorical features** directly without encoding and is not confused too much by **outliers** and **missing values** to a fair degree.
- Non-Parametric Method: Have **no assumptions on space distribution** and classifier structure.
- **Easy to Understand**: It does not require any statistical knowledge to read and interpret.
- Its **graphical representation is very intuitive**, and users can easily relate their hypothesis.
- **Useful in Data Exploration**: Good way to identify the most significant variables and their relations;
- Can help to identify the most **significant variable** when working with hundreds of them.



Disadvantages of Decision Trees

- **Overfitting**: One of the most practical difficulty for decision tree models. Decision-tree can create **over-complex** trees that do not generalise the data well.
- Decision trees can be **unstable** because small variations in the data might result in a completely different tree being generated.
- There are concepts that **are hard to learn** because decision trees do not express them easily, such as XOR concept.
- Decision tree learners create **biased trees** if some classes dominate. It is therefore recommended to **balance** the dataset prior to fitting with the decision tree.



Decision Tree Algorithms (most common)

- CART (Classification And Regression Tree)
- **ID3** (Iterative Dichotomiser 3)
- **C4.5** (Successor of ID3)
- CHAID (CHi-squared Automatic Interaction Detector): Multi-level splits are performed for classification trees
- Conditional Inference Trees: This method results in unbiased feature selection and does not require pruning, being based on a statistical approach using non-parametric tests to split and to avoid overfitting
- MARS: Multivariate Adaptive Regression Splines



Splitting strategies

- The decision of how to **split nodes** affects a tree's accuracy heavily
- Decision Trees use different **algorithms** to decide to split a node into two or more sub-nodes
- The sub-nodes are required to be **more homogeneous**. i.e. The purity of the node increases in relation to the **target variable**
- Decision trees pick the split that results in the **most homogeneous** sub-nodes after calculating splits on **all available variables**.
- Common splitting strategies
 - **GINI**
 - The Gini index indicates the **degree of heterogeneity**. The higher the value of Gini, the higher the heterogeneity. (i.e. Gini = 0 means fully homogeneous)
 - **Information gain**
 - It chooses the split which has the lowest **entropy** (degree of disorder) compared to the parent node and other splits



Using decision trees in Scikit Learn

- Scikit-learn **DecisionTreeClassifier** is a class capable of performing multi-class classification on a dataset.
- Decision tree class works **the same as other classifiers** by taking a set of features and target variable.
- It does not require complex data preparation as it can deal with **categorical variables**, **missing data** and **outliers**. However, cleaning the data does not hurt and will help in improving results. Especially if you are going to use decision trees side by side with other models.
- Once trained, you can **plot the tree** with the `plot_tree` function.
- Note that the default values for the parameters controlling the size of the trees (e.g. `max_depth`, `min_samples_leaf`, etc.) lead to **fully grown and unpruned trees** which can potentially be very large on some data sets. To reduce memory consumption, the complexity and size of the trees should be controlled by setting those hyperparameter values.



Decision Trees hyperparameters

- **criterion** : string, optional (default="gini")
 - The function to measure the quality of a split. Supported criteria are "gini" for the Gini impurity and "entropy" for the information gain.
- **max_depth** : int or None, optional (default=None)
 - The maximum depth of the tree. If None, then nodes are expanded until all leaves are pure or until all leaves contain less than min_samples_split samples.
- **min_samples_split** : int, float, optional (default=2)
 - The minimum number of samples required to split an internal node
- **max_leaf_nodes** : int or None, optional (default=None)
 - Grow a tree with max_leaf_nodes in best-first fashion. Best nodes are defined as relative reduction in impurity. If None then unlimited number of leaf nodes.
- **class_weight** : dict, list of dicts, "balanced" or None, default=None
 - Weights associated with classes in the form {class_label: weight}. If not given, all classes are supposed to have weight one.



Decision trees - Summary

- Decision tree is one of the **most widely used** and **practical** supervised learning algorithm.
- Motivations for using decision trees include:
 - **Less data cleaning required**
 - **Easy to Understand and explain**
- However, decisions trees tend to **overfit** training data without the appropriate hyperparameters tuning



Lab 7.1.1: Decision Trees

- Purpose
 - Practice with Decision Tree Modelling
 - Practice coding in Python
- Resources
 - Your skills and resourcefulness
- Material
 - Jupyter Notebook (Lab-7_1_1)



Random Forests

- Overview
- Algorithm
- Pros and Cons



Random Forest Overview

- Random Forest is a learning approach that uses an **ensemble** method for regression and classification.
- It uses “**Bagging**” (aka bootstrapping) where a set of relatively weak learners are combined to create a stronger learner. These techniques sample the training dataset with **replacement** and train a model of each subset then aggregate these models.
- Constructs a **large list of uncorrelated decision trees** at training time and chooses the **mean** prediction (regression) or the **mode** label of the classes (classification) of the individual trees.
- Random decision forests address the tendency of decision trees to **overfit** on the training data.



Random Forest Algorithm

- Let N be the number of cases in the training set
 - A random sample of N with replacement is the training set for growing the tree
- Let M be the number of input variables
 - Specify a number $m < M$ such that at each node, m variables are randomly selected from M
 - Use m to split the node. While the forest grows the value of m is held constant
- Trees grow to their terminal nodes without pruning
- Makes a prediction of data by aggregating the predictions of the n -tree trees (i.e., the majority votes for classification, the average for regression)



Random Forest Pros

- Can handle large datasets with **higher dimensionality**
 - Can identify most significant variables, so it is considered as one of the dimensionality reduction methods.
 - The model outputs the importance of variables
- Smaller prediction **variance** and therefore usually a **better general performance**.
- Keeps accuracy when a large fraction of the data are **missing**
- Has methods for balancing errors in datasets where classes are imbalanced
- Can solve both classification and regression problems



Random Forest Cons

- A **black box** approach for statistical modellers
 - Almost no visibility of what the model does
- Not as good for **regression** problems as it does not give precise, continuous nature predictions
 - In case of regression, it does not predict beyond the range in the training data, and that they may overfit datasets that are particularly noisy



Random Forest hyperparameters

- **n_estimators** : integer, optional (default=10)
 - The number of trees in the forest.
- **criterion** : string, optional (default="gini")
 - The function to measure the quality of a split. Supported criteria are "gini" for the Gini impurity and "entropy" for the information gain.
- **max_depth** : int or None, optional (default=None)
 - The maximum depth of the tree. If None, then nodes are expanded until all leaves are pure or until all leaves contain less than min_samples_split samples.
- **min_samples_split** : int, float, optional (default=2)
 - The minimum number of samples required to split an internal node
- **max_leaf_nodes** : int or None, optional (default=None)
 - Grow a tree with max_leaf_nodes in best-first fashion. Best nodes are defined as relative reduction in impurity. If None then unlimited number of leaf nodes.
- **class_weight** : dict, list of dicts, "balanced" or None, default=None
 - Weights associated with classes in the form {class_label: weight}. If not given, all classes are supposed to have weight one.



Lab 7.1.2: Random Forests

- Purpose
 - Practice with Decision Random Forests
 - Practice coding in Python
- Resources
 - Your skills and resourcefulness
- Material
 - Jupyter Notebook (Lab-7_1_2)



Questions?



Appendices



Common splitting algorithms: Gini Index

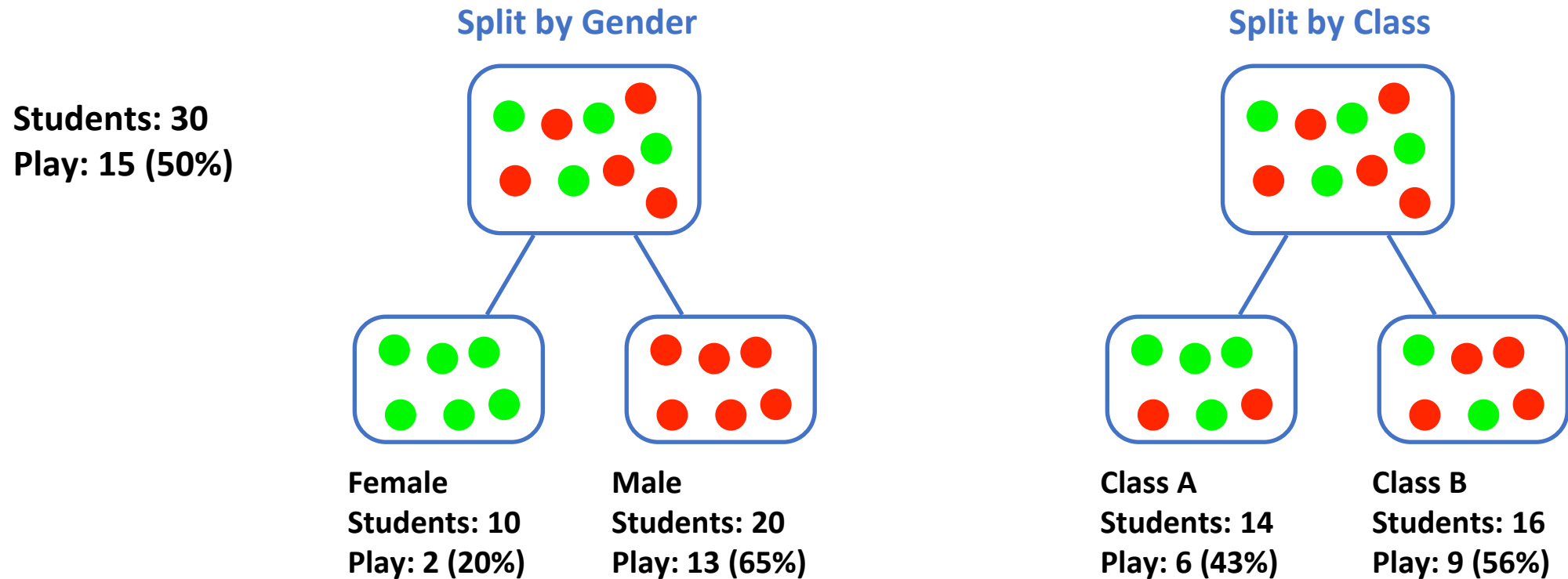
- The Gini index indicates the **degree of homogeneity**. The higher the value of Gini, the higher the homogeneity.
- Gini is used only for Binary splits
- Steps to calculate Gini for a split
 - Calculate Gini for sub-nodes, using the **Sum of Square of probability for success and failure**

$$p^2 + q^2$$

- Calculate Gini for split using weighted Gini score of each node of that split



Common splitting algorithms: Gini Index





Common splitting algorithms: Gini Index

- Split on Gender
 - Gini for sub-node Female = $0.2^2 + 0.8^2 = 0.68$
 - Gini for sub-node Male = $0.65^2 + 0.35^2 = 0.55$
- Split on Class
 - Gini for sub-node A = $0.43^2 + 0.57^2 = 0.51$
 - Gini for sub-node B = $0.56^2 + 0.44^2 = 0.51$



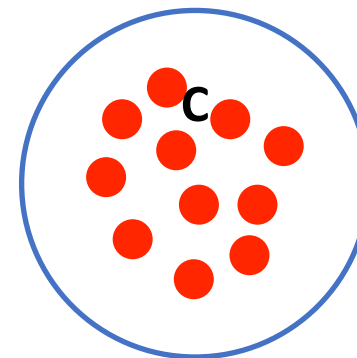
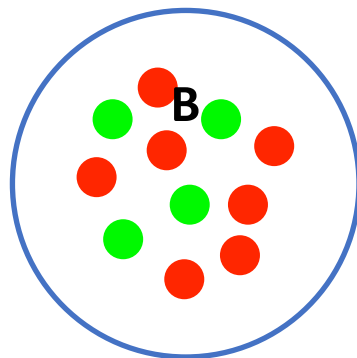
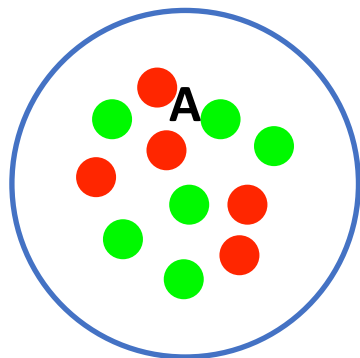
Common splitting algorithms: Gini Index

- Split on Gender
 - Gini Weight on Split Gender = $(10/30 * 0.68) + (20/30 * 0.55) = 0.59$
- Split on Class
 - Gini Weight on Split Class = $(14/30 * 0.51) + (16/30 * 0.51) = 0.51$
- The Gini score for on Gender (**0.59**) is higher than on Class (**0.51**) so it splits on Gender



Common splitting algorithms: Information Gain

- Which image is easier to describe?
 - A requires the maximum information (A is more impure)
 - B requires more information to describe it (B is less Pure than C)
 - C requires less information as all values are similar or the same (C is a Pure node)





Common splitting algorithms: Information Gain

- Used with splitting the data using entropy
- Calculated as the entropy reduction after the dataset is branched on some attribute

$$\text{Gain}(T, X) = \text{Entropy}(T) - \text{Entropy}(T, X)$$

where:

T: Target variable

X: Feature to be split on

Entropy(T, X): The entropy calculated after the data is split on feature X



Common splitting algorithms: Information Gain

- So less impure node requires less information to describe it
- A more impure node requires more information
- Information theory is a measure to define this degree of **disorganisation** in a system known as Entropy



Common splitting algorithms: Information Gain

- Categorical target variables also use entropy
 - It chooses the split which has the lowest entropy compared to the parent node and other splits
- Entropy can be calculated using the formula:

$$Entropy = -p \log_2 p - q \log_2 q$$



Common splitting algorithms: Information Gain

- If the sample is entirely homogeneous, then the entropy is zero
- If the sample is an equally divided (50% / 50%), it has an entropy of one
- Where p and q are the probability of success and failure respectively in that node
- The lesser the entropy, the better it is



Common splitting algorithms: Information Gain

- Steps to calculate Entropy for a split
 1. Calculate entropy of the parent node
 2. Calculate entropy of each node of split and calculate the weighted average of all sub-nodes available in split



Common splitting algorithms: Information Gain

1. Parent node's entropy = $-(15/30) \log_2 (15/30) - (15/30) \log_2 (15/30) = 1$
 - The result of 1 shows an impure node
2. Gender's entropy
 - Female = $-(2/10) \log_2 (2/10) - (8/10) \log_2 (8/10) = 0.72$
 - Male = $-(13/20) \log_2 (13/20) - (7/20) \log_2 (7/20) = 0.93$
3. Entropy for split Gender (Weighted entropy) = $(10/30) * 0.72 + (20/30) * 0.93 = \underline{0.86}$



Common splitting algorithms: Information Gain

1. Entropy for Class

1. $A = -(6/14) \log_2 (6/14) - (8/14) \log_2 (8/14) = 0.99$

2. $B = -(9/16) \log_2 (9/16) - (7/16) \log_2 (7/16) = 0.99$

2. Entropy for split Class = $(14/30) * 0.99 + (16/30) * 0.99 = \underline{0.99}$

- The tree splits on Gender as the Entropy for Split on Gender is the lowest among all



End of Presentation