CaRT (Decision Trees)

Classification and Regression Trees

Derek Andersen and Joanne Chau

Agenda

- Introduction to CaRT
- Classification trees
- Python implementation of classification tree
- Regression trees
- Python implementation of regression tree
- Conclusion

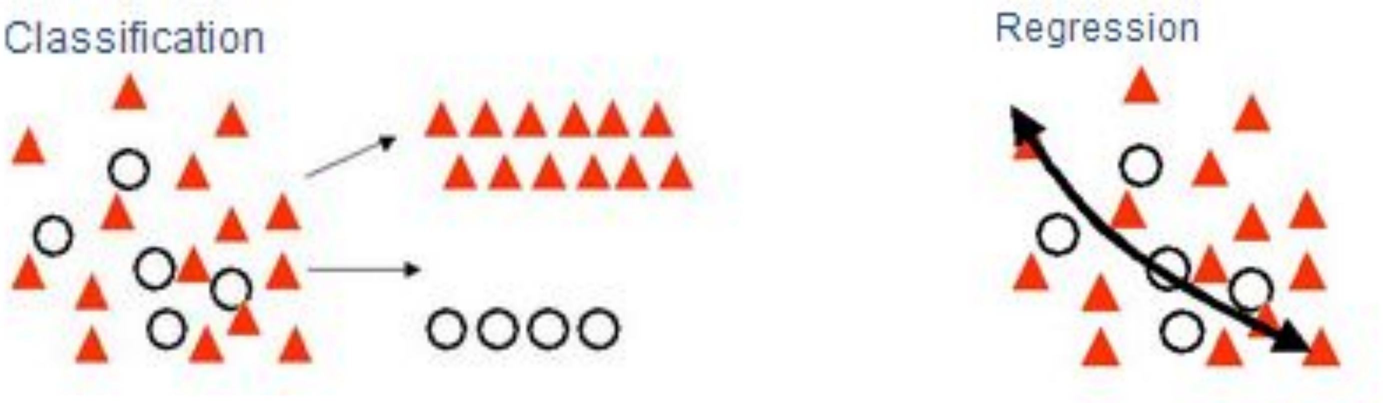
Decision Trees

- An important type of algorithm for predictive modeling machine learning.
- Their purpose is to serve as a model that predicts the value of a target (dependent) variable based on the values of several input (independent) variables.
- A tree model is built by being trained on a training dataset. The resulting tree can be stored and used later on for data analysis / classification.
- Constructed in a top-down fashion through recursive partitioning.

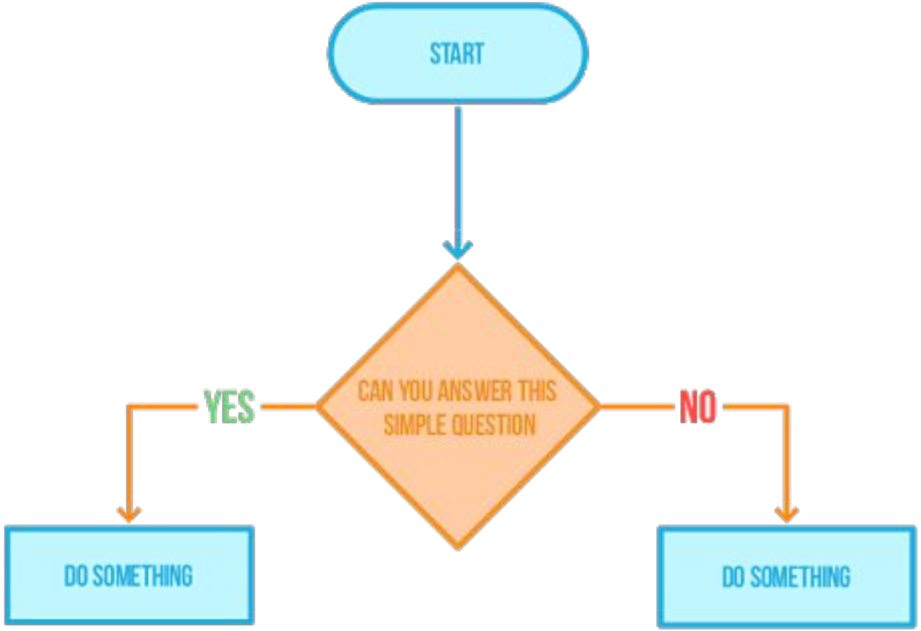
What is CaRT?

- CaRT stands for ‘Classification and Regression trees.’
- It was introduced in 1984 by Leo Breiman to refer to Decision Tree algorithms that are used for **classification** or **regressive** modeling problems.
- CaRT is an umbrella term that refers to the following types of decision trees:
  - **Classification Trees**: The target variable is categorical and the tree is used to identify the "class" within which a target variable would likely fall.
  - **Regression Trees**: The target variable is continuous and the tree is used to predict its value.

Classification vs. Regression



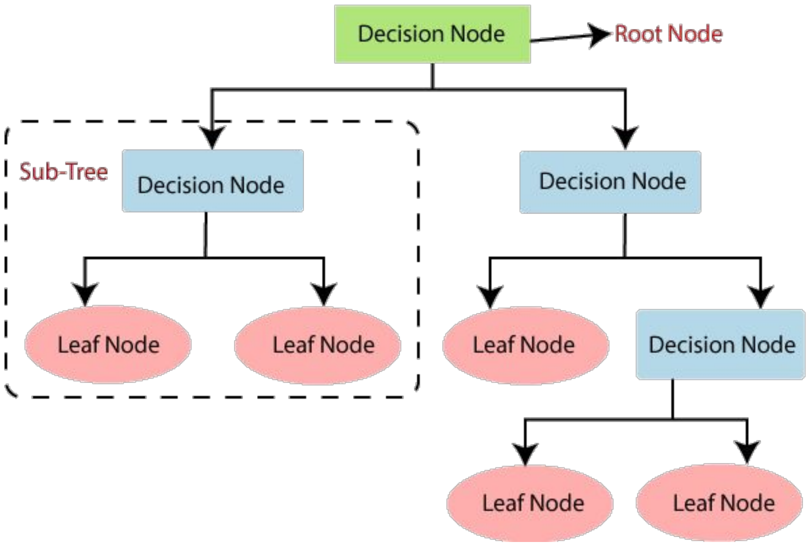
A visual example of Classification vs. Regression.



A very simple decision tree.

Components of CaRT Model Representation

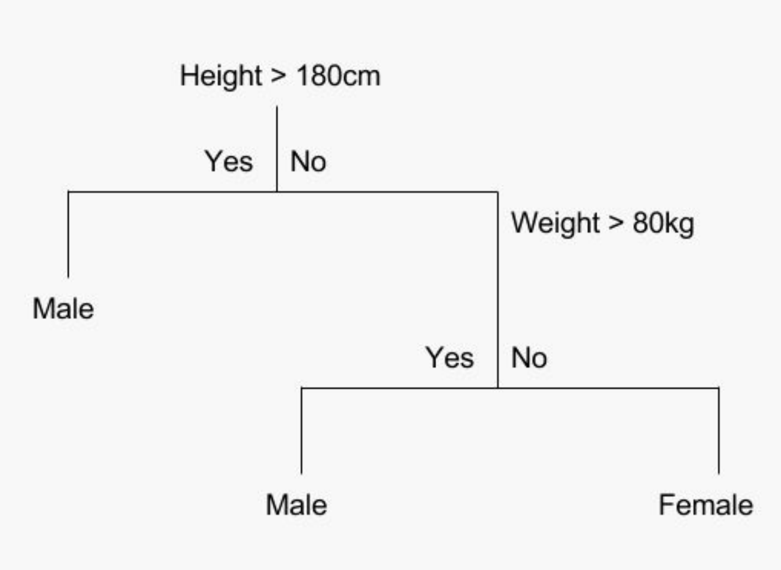
- **Nodes**
  - **Root Node**: The starting point of the tree, contains some criterion or question that has to be answered
  - **Decision Nodes**: Contain criteria or questions that have to be answered
  - **Leaf Nodes**: Also commonly known as terminal nodes. Contain the final category or predicted value for the data
- **Branches**
  - Arrows connecting nodes, showing the flow from question to answer



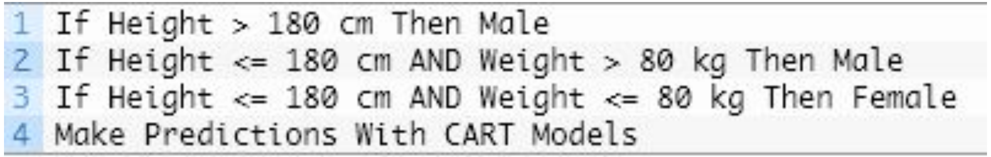
How is a CaRT model stored?

- Once a CaRT model is trained, it can be stored as a set of **rules** (in the form of conditional statements).
- These rules act to control the flow of a test instance being evaluated via the trained model.
- Given an test instance (with parameters), the model will evaluate it according to this set of rules and a decision (or estimated value in the case of regression trees) will be output.
- This stored model can be retrieved and used later for future data analysis or classification tasks.

How is a CaRT model stored?



Tree



```
If Height > 180 cm Then Male
If Height <= 180 cm AND Weight > 80 kg Then Male
If Height <= 180 cm AND Weight <= 80 kg Then Female
Make Predictions With CART Models
```

Set of rules

Question:
What gender is a person with a height of 150cm and a weight of 90kg?

Feature Choosing

Creating a CaRT model involves selecting **features** of input instances and **split conditions** according to those features until a suitable tree is constructed.

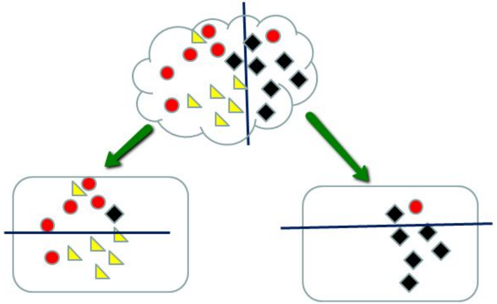
Feature Selection → The process of deciding which variables of the data are important enough to include in the model. Feature selection is done by the algorithm itself, not manually.

The selected features are what dictate the construction of the tree, and determine where its split points will be.

Conditions for Splitting

To decide how to split nodes, a process called **recursive partitioning** is used.

This is a process where all the data is split into partitions, according to different split conditions using a **cost function**. The condition with the best cost is selected. This process is repeated until the data is sufficiently split.



Note: This is considered a **greedy** approach because the best split condition is chosen at each node, without concern for future split choices.

How are split conditions decided for classification problems?

Cost function for classification problems

- For classification problems, the **gini index** function, which ranges between 0 and 1, is used as a cost function.
- This provides an indication of how “pure” the leaf nodes are (how mixed the training data partitioned to each node is).
- If at some node, all the data belongs to a single class, then the node is considered **pure** (a gini index of 0) → this will lead to a terminal node.
- A node that has a 50/50 split of classes for a binary classification problem (worst purity) will have a gini index of 0.5.

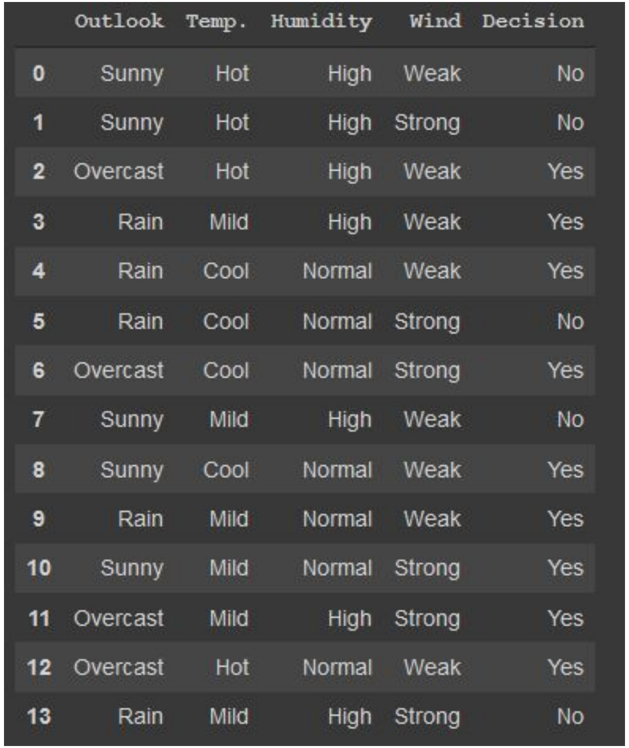
Gini index formula

$$G = 1 - \sum_{i=1}^{C} (p_i)^2$$

Where:

- *G* is the gini index over all classes (for a given parameter),
- p_i is the probability of an object being classified to a particular class *i*,
- *C* is the number of possible classes for each instance (e.g. for {yes, no}, *C* = 2).

Example dataset: Golf playing decisions



	Outlook	Temp.	Humidity	Wind	Decision
0	Sunny	Hot	High	Weak	No
1	Sunny	Hot	High	Strong	No
2	Overcast	Hot	High	Weak	Yes
3	Rain	Mild	High	Weak	Yes
4	Rain	Cool	Normal	Weak	Yes
5	Rain	Cool	Normal	Strong	No
6	Overcast	Cool	Normal	Strong	Yes
7	Sunny	Mild	High	Weak	No
8	Sunny	Cool	Normal	Weak	Yes
9	Rain	Mild	Normal	Weak	Yes
10	Sunny	Mild	Normal	Strong	Yes
11	Overcast	Mild	High	Strong	Yes
12	Overcast	Hot	Normal	Weak	Yes
13	Rain	Mild	High	Strong	No

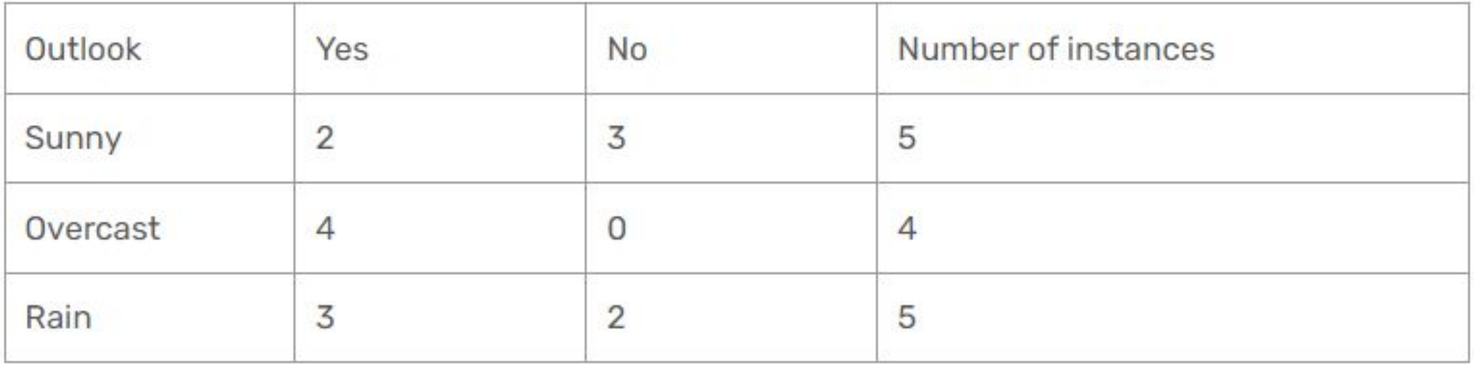
Features

- **Outlook** → Sunny, Overcast, Rain
- **Temperature** → Cool, Mild, Hot
- **Humidity** → High, Normal
- **Wind** → Weak, Strong

Let’s calculate the gini index

First, we calculate the gini index for each individual feature. The one with the lowest cost will be used as the feature to take into consideration for the first decision node.

‘Outlook’ feature



Outlook	Yes	No	Number of instances
Sunny	2	3	5
Overcast	4	0	4
Rain	3	2	5

Gini index for ‘outlook’ feature

$$G(Sunny) = 1 - \overset{\text{yes}}{\left(\frac{2}{5}\right)^2} - \overset{\text{no}}{\left(\frac{3}{5}\right)^2} = 1 - 0.16 - 0.36 = 0.48$$

$$G(Overcast) = 1 - \left(\frac{4}{4}\right)^2 - \left(\frac{0}{4}\right)^2 = 1 - 1 - 0 = 0$$

$$G(Rain) = 1 - \left(\frac{3}{5}\right)^2 - \left(\frac{2}{5}\right)^2 = 1 - 0.36 - 0.16 = 0.48$$

Weighted sum of gini indexes for ‘outlook’ feature:

$$G(\textbf{Outlook}) = \left(\frac{5}{14}\times 0.48\right) + \left(\frac{4}{14}\times 0\right) + \left(\frac{5}{14}\times 0.48\right) = 0.171 + 0 + 0.171 = \boxed{0.342}$$

The winning feature

$$G(\textbf{Outlook}) = \boxed{0.342}$$

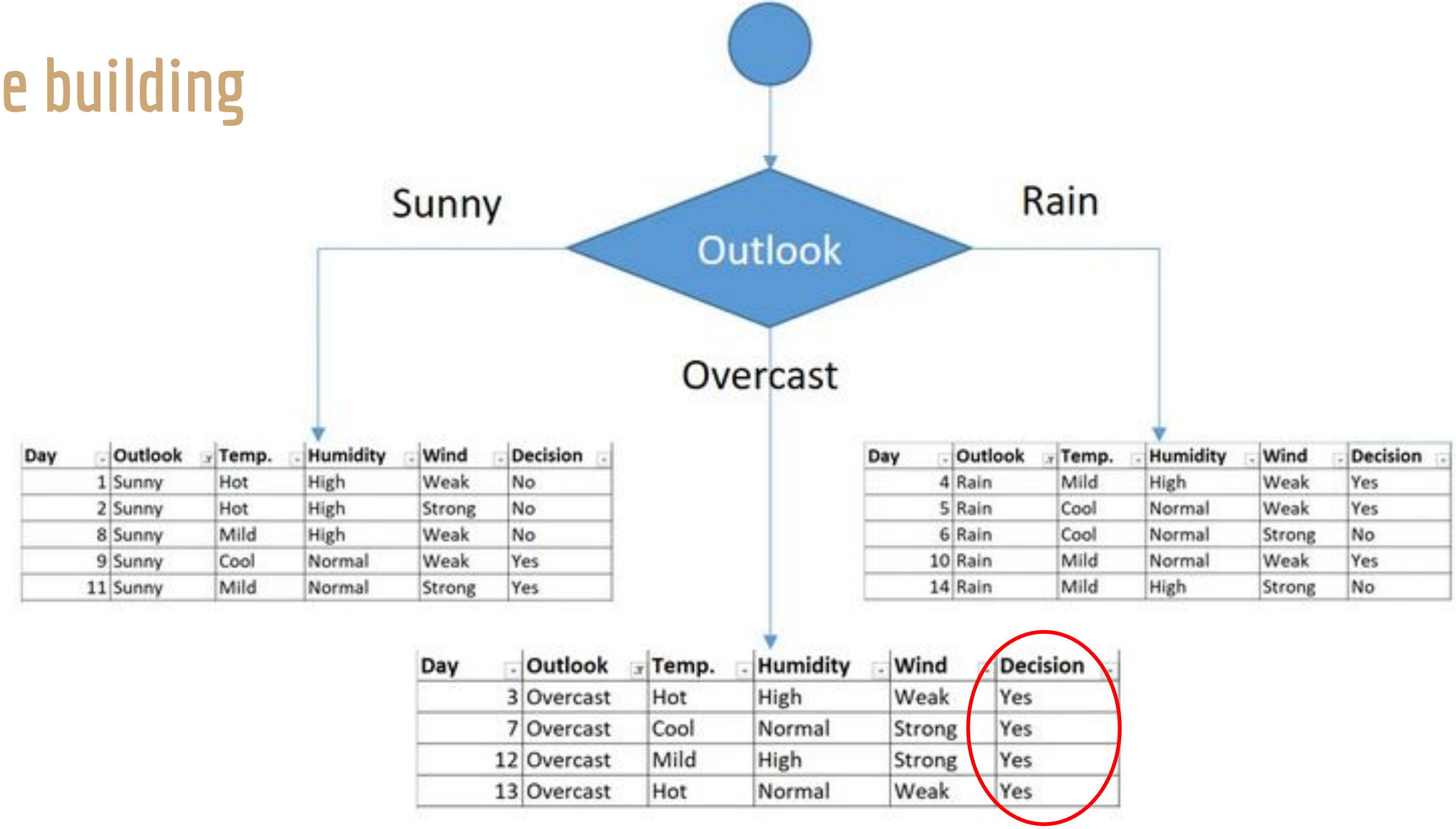
Outlook has the lowest gini index, so it’s our first split condition.

$$G(\textbf{Temperature}) = 0.439$$

$$G(\textbf{Humidity}) = 0.367$$

$$G(\textbf{Wind}) = 0.428$$

Tree building

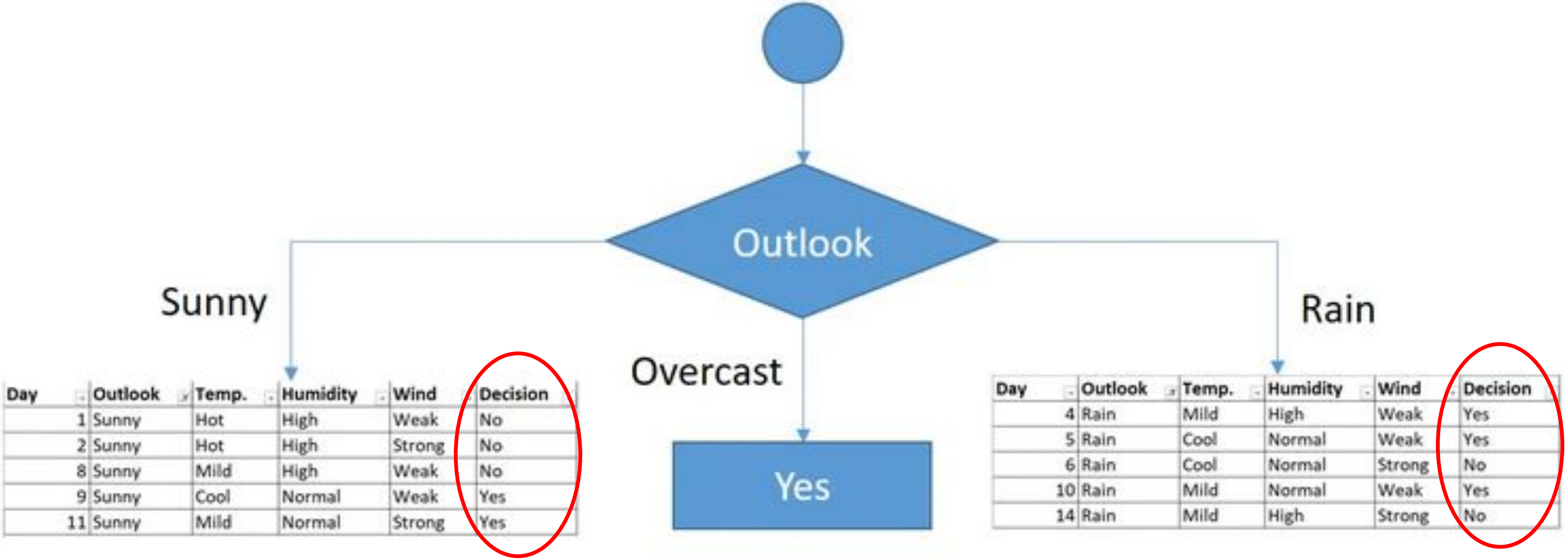


Day	Outlook	Temp.	Humidity	Wind	Decision
1	Sunny	Hot	High	Weak	No
2	Sunny	Hot	High	Strong	No
8	Sunny	Mild	High	Weak	No
9	Sunny	Cool	Normal	Weak	Yes
11	Sunny	Mild	Normal	Strong	Yes

Day	Outlook	Temp.	Humidity	Wind	Decision
4	Rain	Mild	High	Weak	Yes
5	Rain	Cool	Normal	Weak	Yes
6	Rain	Cool	Normal	Strong	No
10	Rain	Mild	Normal	Weak	Yes
14	Rain	Mild	High	Strong	No

Day	Outlook	Temp.	Humidity	Wind	Decision
3	Overcast	Hot	High	Weak	Yes
7	Overcast	Cool	Normal	Strong	Yes
12	Overcast	Mild	High	Strong	Yes
13	Overcast	Hot	Normal	Weak	Yes

Purity = 0 → Terminal node ✓



Day	Outlook	Temp.	Humidity	Wind	Decision
1	Sunny	Hot	High	Weak	No
2	Sunny	Hot	High	Strong	No
8	Sunny	Mild	High	Weak	No
9	Sunny	Cool	Normal	Weak	Yes
11	Sunny	Mild	Normal	Strong	Yes

Day	Outlook	Temp.	Humidity	Wind	Decision
4	Rain	Mild	High	Weak	Yes
5	Rain	Cool	Normal	Weak	Yes
6	Rain	Cool	Normal	Strong	No
10	Rain	Mild	Normal	Weak	Yes
14	Rain	Mild	High	Strong	No

Purity ≠ 0 → Repeat feature selection process

$$G(\mathbf{Temperature}) = ?$$

$$G(\mathbf{Humidity}) = ?$$

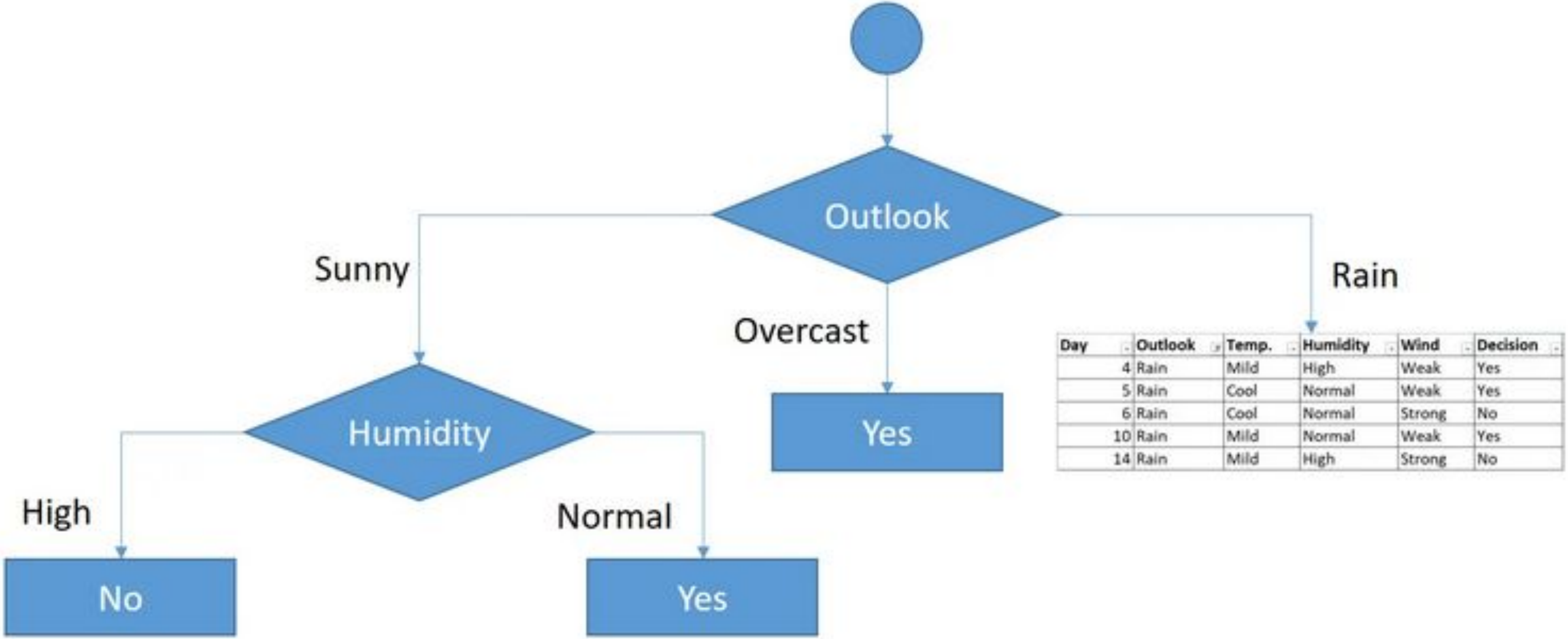
$$G(\mathbf{Wind}) = ?$$

Purity ≠ 0 → Repeat feature selection process

$$G(\mathbf{Temperature}) = ?$$

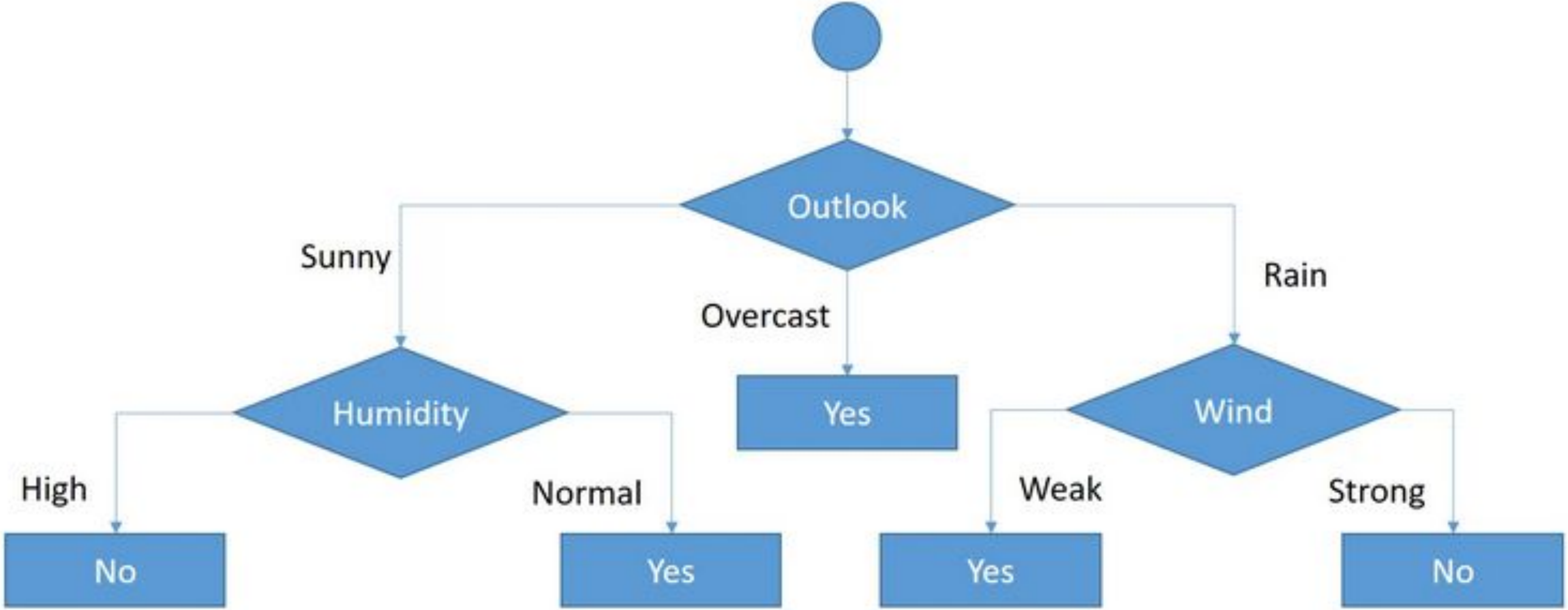
$$G(\mathbf{Humidity}) = ?$$

$$G(\mathbf{Wind}) = ?$$



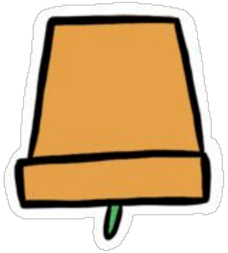
Day	Outlook	Temp.	Humidity	Wind	Decision
4	Rain	Mild	High	Weak	Yes
5	Rain	Cool	Normal	Weak	Yes
6	Rain	Cool	Normal	Strong	No
10	Rain	Mild	Normal	Weak	Yes
14	Rain	Mild	High	Strong	No

Final tree


Outlook
Sunny
Overcast
Rain
Humidity
Yes
Wind
High
Normal
Weak
Strong
No
Yes
Yes
No

Classification Tree Implementation in Python

How a CaRT model is grown



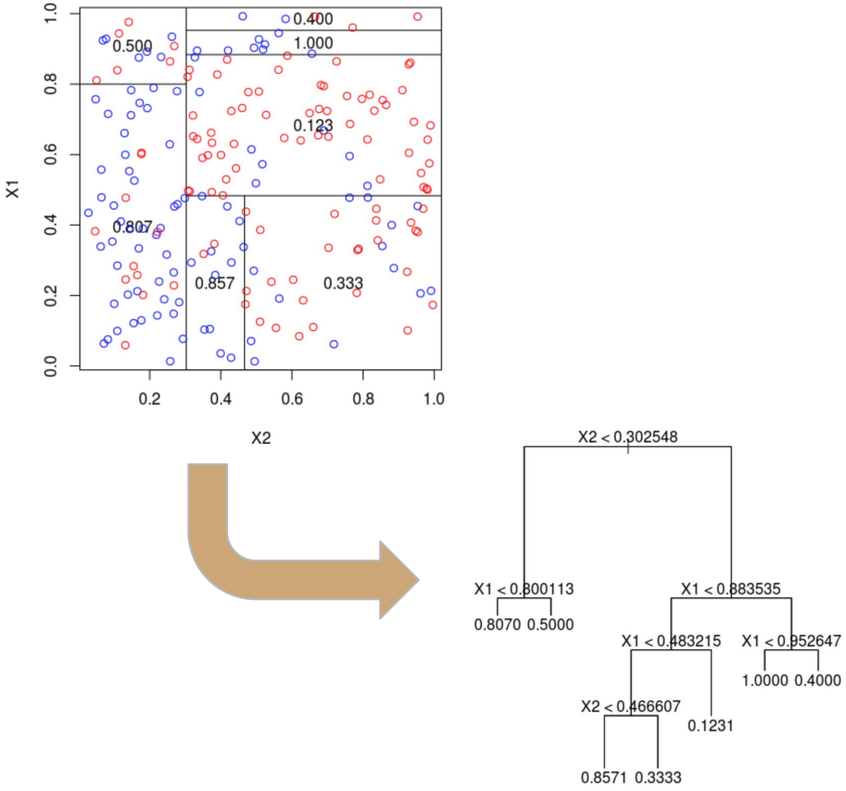
The CaRT model provides a recipe in order to grow the tree representation.

1. Features to choose
2. Conditions for splitting
3. Stopping rules for deciding when a branch is terminal
4. Pruning

How are split conditions decided for regression problems?

What are Regression Trees?

- Regression trees are a variant of decision trees.
- Unlike classification trees, they are designed to approximate real-valued functions.
- It is built utilizing recursive partitioning, which is an iterative process that splits the data into partitions or branches and continues splitting each partition into smaller groups.



Feature choosing for regression problems

- There are many splitting criteria for regression problems.
- One of the most common ones are **weighted variance** of the nodes because we want minimum variation in the nodes after the split.

$$\text{Variance} = \frac{\Sigma(X - \overline{X})^2}{n}$$

- The algorithm selects the split that minimizes the sum of the standard deviations from the mean in two separate partitions.
- The splitting rule continues until it hits a user-specified minimum node size or if sum of standard deviation to the node is zero.

Stopping Criteria

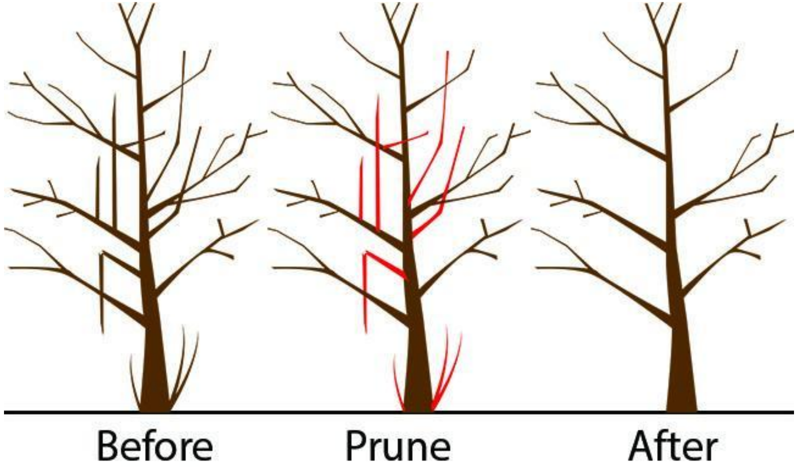
- Because CaRT proceeds recursively, it is important to have a stopping criterion. Imagine a dead end to let them know to not go further.
- Stopping criteria controls if the tree growing process should stop or continue.
- The below are examples of stopping rules that are used:
    - If a node becomes pure: If all cases result in the same identical value
    - If the current tree depth reaches the user-specified maximum node size value

Pruning the Tree

Pruning → Optimizing a decision tree. This is accomplished by simplifying or compressing by removing sections of the tree that are redundant to classify instances. This process came about from an attempt to prevent overfitting in trees.

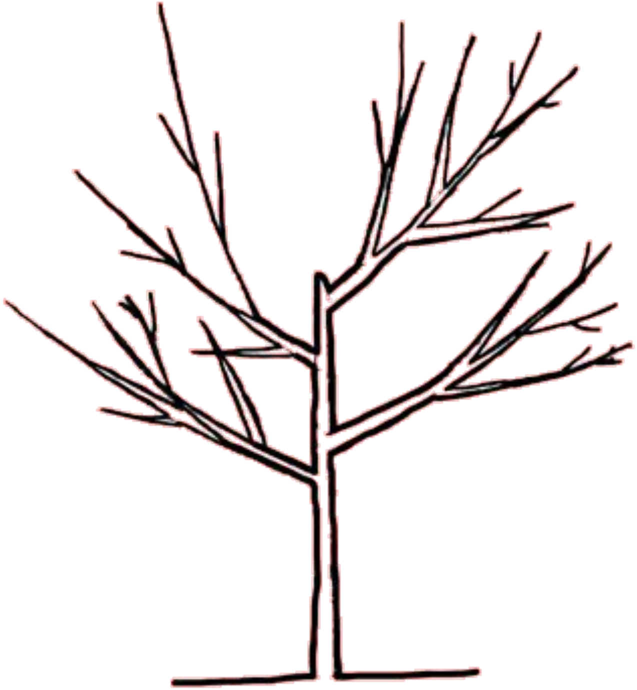
There are two types of pruning.

- Pre-pruning
- Post-pruning



Pre-Pruning

- We know it as the stopping criterion
- It stops the tree-building process early, before it produces leaves with very small sample sizes.
- It prevents a complete induction of the training set by introducing a stop criterion



Post-Pruning

- Post-Pruning is what people normally refer to when they say pruning.
- Nodes and subtrees are replaced with leaves to improve complexity. Pruning can not only reduce the size but also improve the classification accuracy of unseen objects.
- It is broken up into:
  - Bottom-Up Pruning
  - Top-Down Pruning

Post-Pruning

Bottom-Up Pruning

- Start from the terminal nodes and follow the recursive upwards and then determine the relevance of each individual node. If the relevance for the classification is not given, the node is dropped or replaced by a leaf.
- **Advantage**: No relevant sub-trees can be lost with this method.
- **Methods**: Reduced Error Pruning, Minimum Cost Complexity Pruning, Minimum Error Pruning

Top-Down Pruning

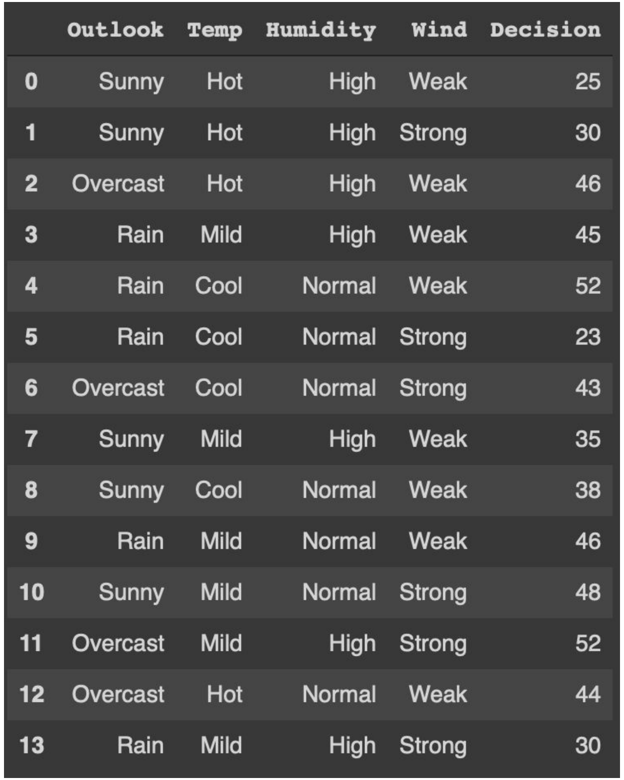
- Starts from the root node. Following the structure, it checks and decides whether a node is relevant for the classification for all *n* items. If it is irrelevant, the node is replaced by a leaf or dropped.
- **Disadvantage**: By pruning an inner node, an entire subtree can be dropped.
- **Methods**: Pessimistic Error Pruning

Regression example

- We will be utilizing a similar dataset to the classification example. The only difference between the two datasets are in the decision columns. Regression trees require numbers.

Features:

- **Outlook** → Sunny, Overcast, Rain
- **Temperature** → Cool, Mild, Hot
- **Humidity** → High, Normal
- **Wind** → Weak, Strong



	Outlook	Temp	Humidity	Wind	Decision
0	Sunny	Hot	High	Weak	25
1	Sunny	Hot	High	Strong	30
2	Overcast	Hot	High	Weak	46
3	Rain	Mild	High	Weak	45
4	Rain	Cool	Normal	Weak	52
5	Rain	Cool	Normal	Strong	23
6	Overcast	Cool	Normal	Strong	43
7	Sunny	Mild	High	Weak	35
8	Sunny	Cool	Normal	Weak	38
9	Rain	Mild	Normal	Weak	46
10	Sunny	Mild	Normal	Strong	48
11	Overcast	Mild	High	Strong	52
12	Overcast	Hot	Normal	Weak	44
13	Rain	Mild	High	Strong	30

Let’s Calculate the Global Standard Deviation (all golfers)

$$\mathrm{SD} = \sqrt{\frac{\sum |x - \bar{x}|^2}{n}}$$

- Where *x* ¯ is the average of all *x*’s
- *x* is each individual instance in the set of all *x*’s
- *n* is the total number of *x*’s

golf_players = {25, 30, 46, 45, 52, 23, 43, 35, 38, 46, 48, 52, 44, 30}

avg(golf_players) = $\frac{25+30+46+45+52+23+42+35+38+46+48+52+44+30}{14} = 39.78$

SD(golf_players) = $\sqrt{\frac{(25-39.78)^2 + (30-39.78)^2 + (46-39.18)^2 + \dots + (30-39.78)^2}{14}} = 9.32$

Feature Splitting

We must now calculate the standard deviation for each of the features.

The feature with the highest standard deviation reduction would be the optimal feature. We continue calculating the Standard Deviation Reduction until we reach terminal nodes.

- To calculate the **Standard Deviation Reduction (SDR)** first calculate the **Standard Deviation (SD)** of the full dataset (shown in the slide above), known as the **global SD**. Then calculate each features’ **Weighted Standard Deviation (WSD)** and then subtract the WSD from the global SD.

Standard Deviation Reduction Calculation

- First we need to calculate the WSD for each of the features.
- Looking at the Outlook feature, it divides into three categories, Sunny, Overcast and Rain.
- We break the Outlook feature up into each of the categories because we need to calculate the SD for each of the outlook candidates.

Sunny Outlook

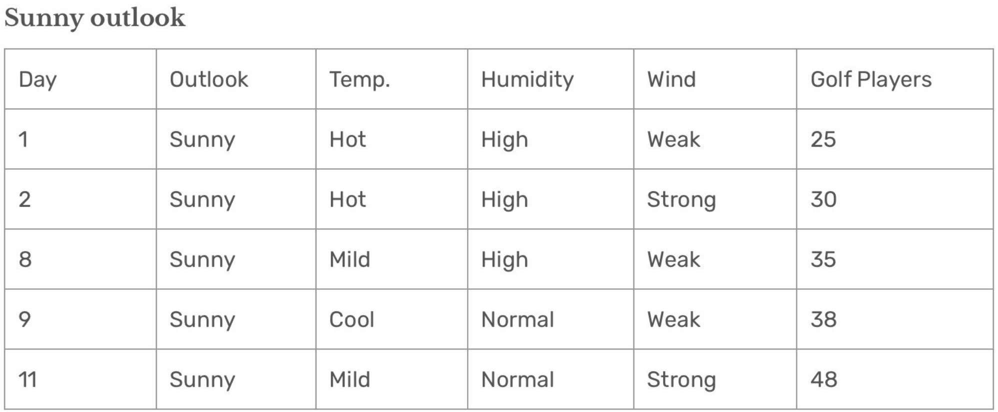
golf_players = {25, 30, 35, 38, 48}

avg(golf_players |sunny) =

$$\frac{25+30+35+38+48}{2} = 35.2$$

SD(golf_players | sunny) =

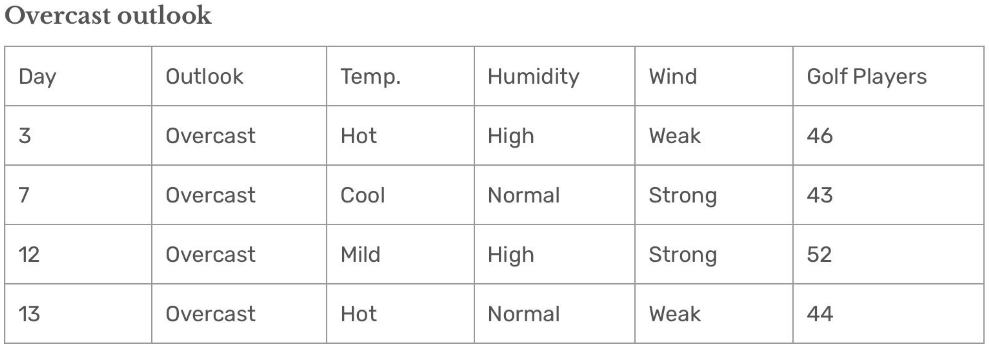
$$\sqrt{\frac{(25-35.2)^2+(30-35.2)^2+...+(48-35.2)^2}{5}} = 7.78$$



Sunny outlook

Day	Outlook	Temp.	Humidity	Wind	Golf Players
1	Sunny	Hot	High	Weak	25
2	Sunny	Hot	High	Strong	30
8	Sunny	Mild	High	Weak	35
9	Sunny	Cool	Normal	Weak	38
11	Sunny	Mild	Normal	Strong	48

Overcast Outlook



Overcast outlook

Day	Outlook	Temp.	Humidity	Wind	Golf Players
3	Overcast	Hot	High	Weak	46
7	Overcast	Cool	Normal	Strong	43
12	Overcast	Mild	High	Strong	52
13	Overcast	Hot	Normal	Weak	44

golf_players = {46, 43, 52, 44}

avg(golf_players | overcast) =

$$\frac{46+43+52+44}{4} = 46.25$$

SD(golf_players | overcast) =

$$\sqrt{\frac{(46-46.25)^2+\ldots+(44-46.25)^2}{4}} = 3.49$$

Rainy Outlook

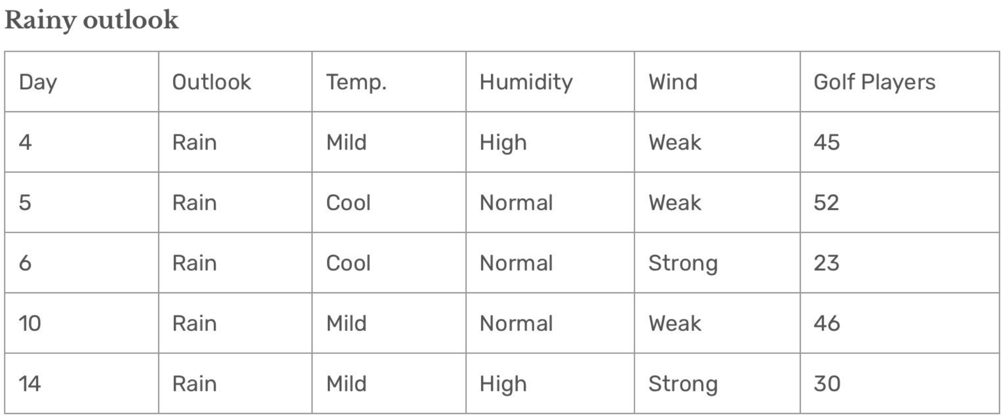
golf_players = {45, 52, 23, 46, 30}

avg(golf_players | rainy) =

$$\frac{45+52+23+46+30}{5} = 39.2$$

SD(golf_players | rainy) =

$$\sqrt{\frac{(45-39.2)^2+\ldots+(30-39.2)^2}{5}} = 10.87$$

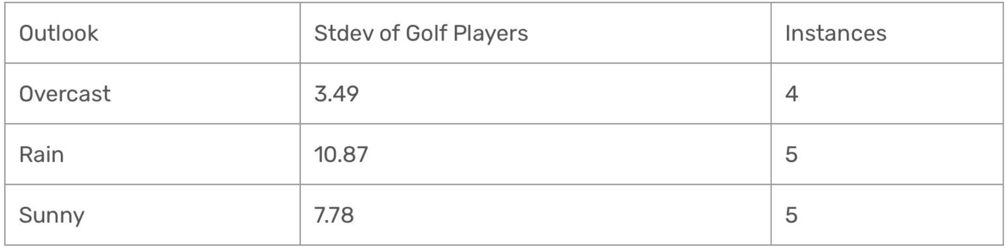


Rainy outlook

Day	Outlook	Temp.	Humidity	Wind	Golf Players
4	Rain	Mild	High	Weak	45
5	Rain	Cool	Normal	Weak	52
6	Rain	Cool	Normal	Strong	23
10	Rain	Mild	Normal	Weak	46
14	Rain	Mild	High	Strong	30

Calculating the Weighted Standard Deviation

To calculate the WSD, we take each candidates’ number of instances, divide it by total instances and then multiply it by their SD. For this category, there are 14 total instances.

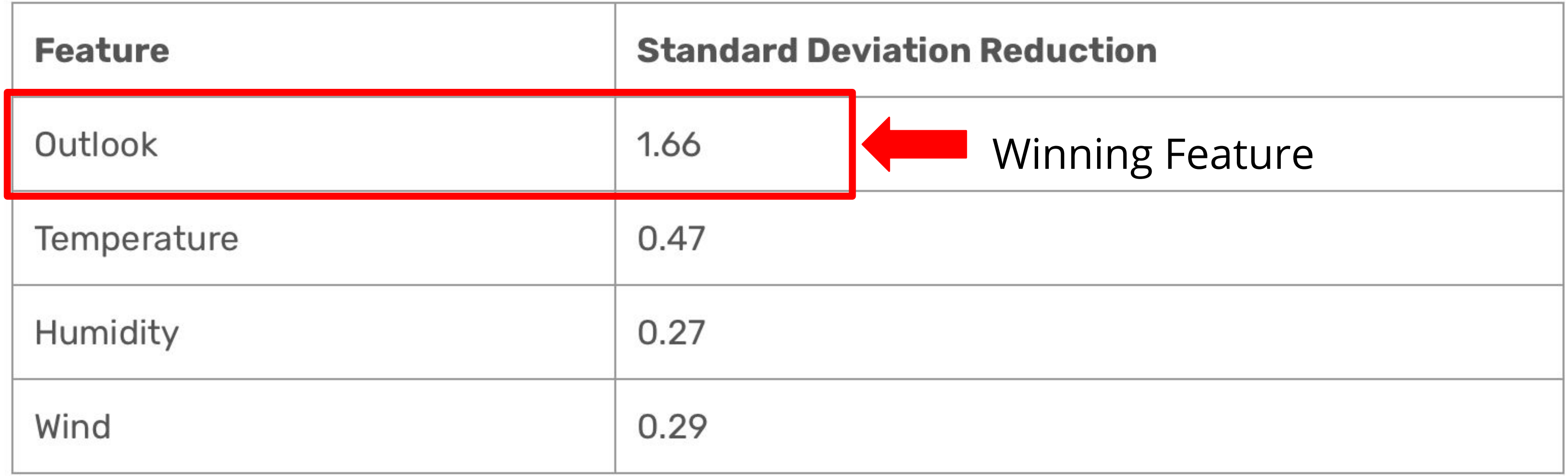


Outlook	Stdev of Golf Players	Instances
Overcast	3.49	4
Rain	10.87	5
Sunny	7.78	5

WSD(outlook) = (4/14)x3.49 + (5/14)x10.87 + (5/14)x7.78 = 7.66

SDR(outlook) = 9.32 - 7.66 = 1.66

Repeat for all Features

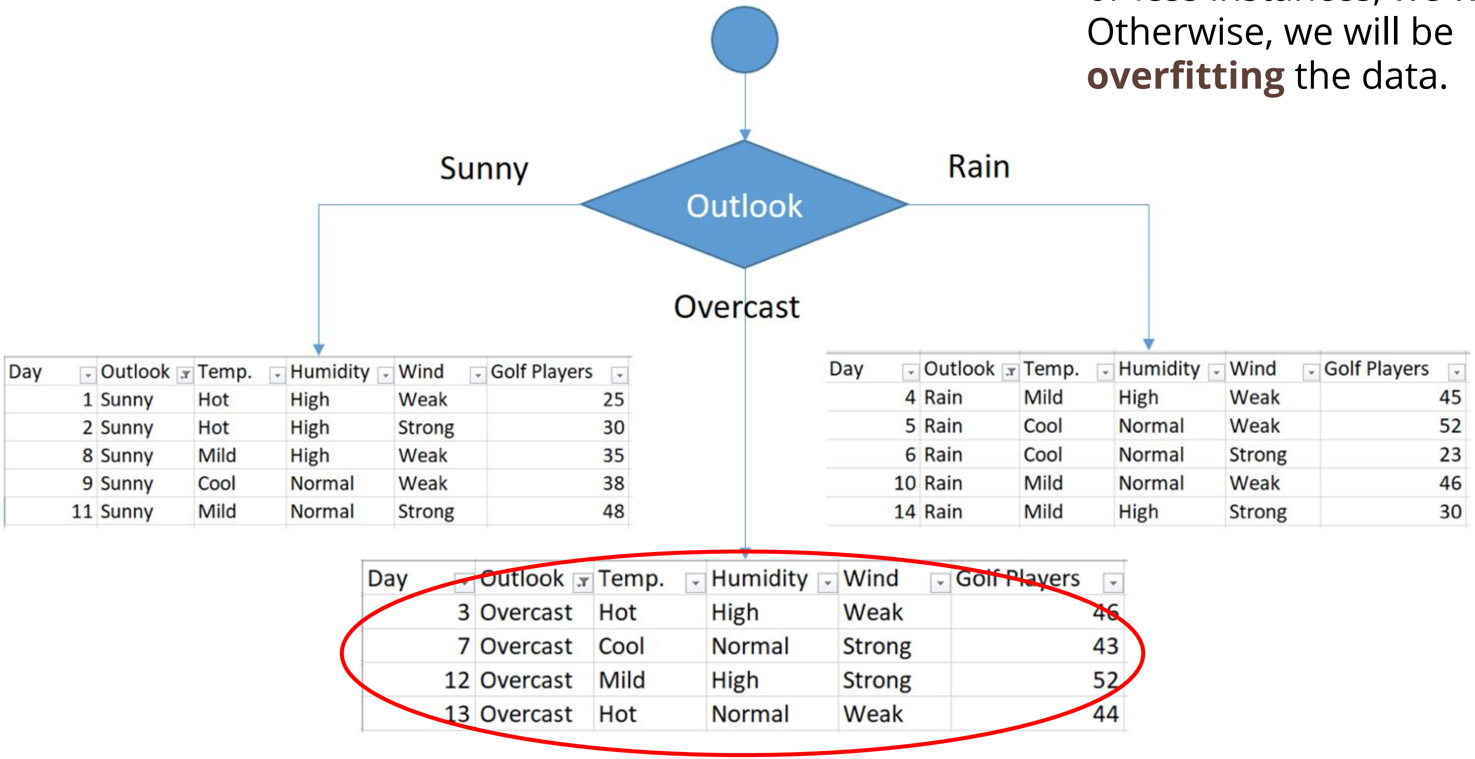


Feature	Standard Deviation Reduction
Outlook	1.66
Temperature	0.47
Humidity	0.27
Wind	0.29

Winning Feature

Tree Building

We can apply a **stopping criterion** here. Since this is a small data set, if there are 4 or less instances, we will stop. Otherwise, we will be **overfitting** the data.



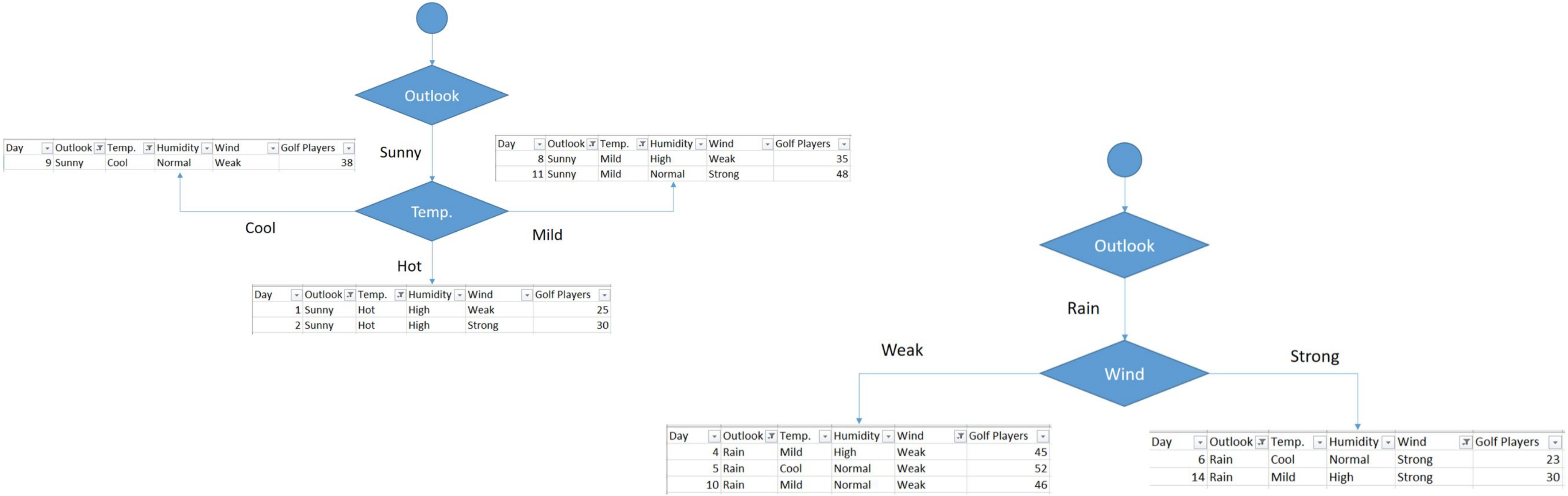
Day	Outlook	Temp.	Humidity	Wind	Golf Players
1	Sunny	Hot	High	Weak	25
2	Sunny	Hot	High	Strong	30
8	Sunny	Mild	High	Weak	35
9	Sunny	Cool	Normal	Weak	38
11	Sunny	Mild	Normal	Strong	48

Day	Outlook	Temp.	Humidity	Wind	Golf Players
4	Rain	Mild	High	Weak	45
5	Rain	Cool	Normal	Weak	52
6	Rain	Cool	Normal	Strong	23
10	Rain	Mild	Normal	Weak	46
14	Rain	Mild	High	Strong	30

Day	Outlook	Temp.	Humidity	Wind	Golf Players
3	Overcast	Hot	High	Weak	46
7	Overcast	Cool	Normal	Strong	43
12	Overcast	Mild	High	Strong	52
13	Overcast	Hot	Normal	Weak	44

Recursive Partitioning

Since the stopping criterion is not applied to the other two outlook features, we now have to work on those partitions.



Day	Outlook	Temp.	Humidity	Wind	Golf Players
9	Sunny	Cool	Normal	Weak	38

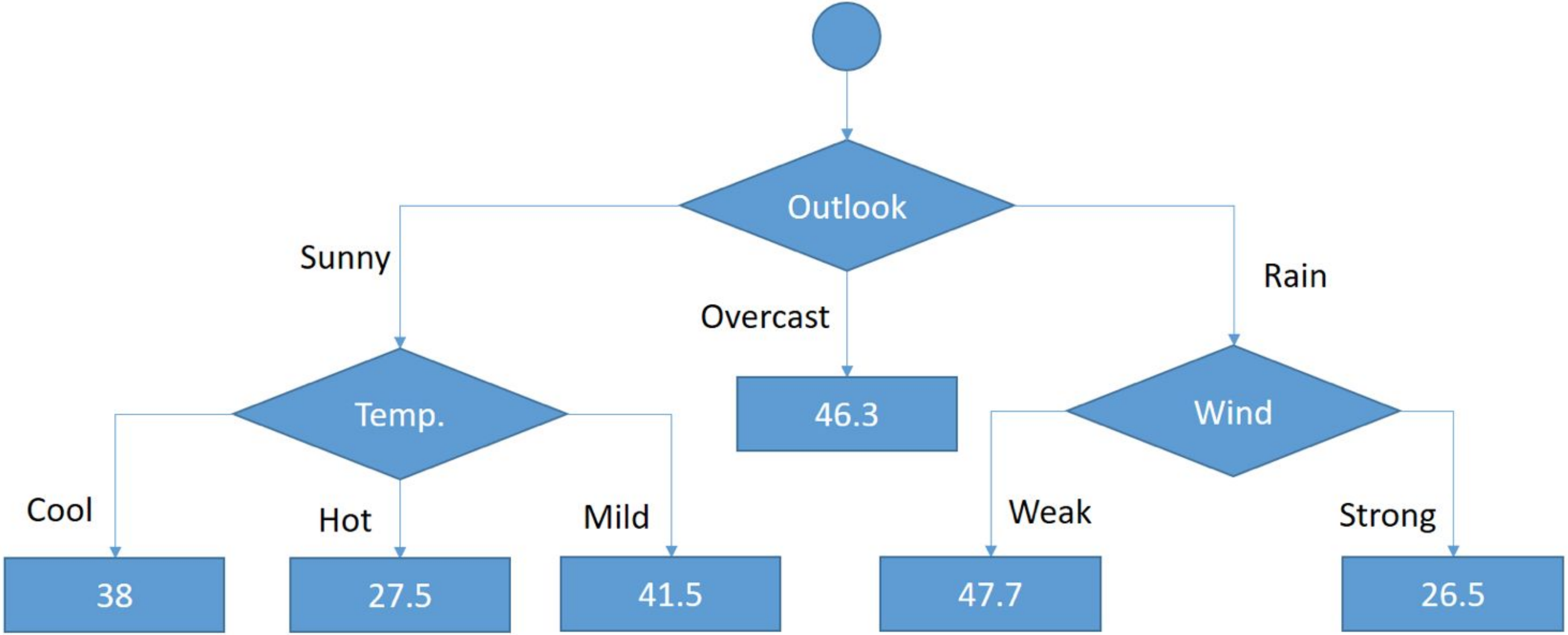
Day	Outlook	Temp.	Humidity	Wind	Golf Players
8	Sunny	Mild	High	Weak	35
11	Sunny	Mild	Normal	Strong	48

Day	Outlook	Temp.	Humidity	Wind	Golf Players
1	Sunny	Hot	High	Weak	25
2	Sunny	Hot	High	Strong	30

Day	Outlook	Temp.	Humidity	Wind	Golf Players
4	Rain	Mild	High	Weak	45
5	Rain	Cool	Normal	Weak	52
10	Rain	Mild	Normal	Weak	46

Day	Outlook	Temp.	Humidity	Wind	Golf Players
6	Rain	Cool	Normal	Strong	23
14	Rain	Mild	High	Strong	30

Final tree


Outlook
Sunny
Overcast
Rain
Temp.
46.3
Wind
Cool
Hot
Mild
Weak
Strong
38
27.5
41.5
47.7
26.5

Regression Tree Implementation in Python

Quote from stackexchange

“Any multi-way split can be represented as a series of two-way splits. For a three-way split, you can split into A, B, and C by first splitting into A&B versus C and then splitting out A from B.

A given algorithm might not choose that particular sequence (especially if, like most algorithms, it's greedy), but it certainly could. And if any randomization or stagewise procedures are done like in random forests or boosted trees, the chances of finding the right sequence of splits goes up. As others have pointed out, multi-way splits are computationally costly, so given these alternatives, most researchers seem to have chosen binary splits.”

Advantages and Disadvantages of Decision Trees

Advantages

- Simple to understand and interpret
- Follows the same process by which humans make decisions in real life
- Useful for decision-related problems
- Helps in visualizing all possible outcomes
- Less data cleanup in comparison to other algorithms

Disadvantages

- Overfitting issues
- They are unstable, when there is a small change in data, it can cause a large change in the structure of the decision tree
- For classification trees, the information gained can be biased

CaRT is more connected to reality than we realize.

Decision trees are important to know and understand because they are used for more than just computer science!

Some examples where decision trees are used:

- Medical practices: used for medical diagnosis
- Business analytics: expansion opportunities based on sales
- Marketing strategies: using demographics to determine popular products
- Automated telephone systems: “For customer representative press 1”

Further Study: Language Modeling Applications

- Application of Semantic Classification Trees to Natural Language Understanding
- Part of Speech Tags and Decision Trees for Language Modeling
- Automatic Classification of Dialogue Acts of Semantic Classification Trees and Polygrams
- Multilingual Prosody Modeling Using Cascades of Regression Trees and Neural Networks

Thank you for listening!

References

- https://www.datasciencecentral.com/profiles/blogs/introduction-to-classification-regression-trees-cart
- https://machinelearningmastery.com/classification-and-regression-trees-for-machine-learning/
- https://en.wikipedia.org/wiki/Decision_tree_pruning
- https://sefiks.com/2018/08/27/a-step-by-step-cart-decision-tree-example/
- https://sefiks.com/2018/08/28/a-step-by-step-regression-decision-tree-example/
- https://github.com/serengil/chefboost