ML – Supervised Learning

CPSC 383: Explorations in Artificial Intelligence and Machine Learning
Fall 2025

Jonathan Hudson, Ph.D
Associate Professor (Teaching)
Department of Computer Science
University of Calgary

August 27, 2025

UNIVERSITY OF CALGARY

Categories

In **supervised learning**, the learning system is given pairs of input/output values as training data.

- e.g. (x, y), where x is an input value and y is the "correct" or "target" output
- y is also sometimes called a **label**

Categories

In **classification** problems, y is one of a fixed number of discrete values.

- If $y \in \{0,1\}$, or has two values, then it is called **binary classification**.

In **regression problems**, y is a continuous numeric value in $\mathbb{R}$.

Exercise

- Suppose you are building a machine learning algorithm that takes a photo as input and determines whether or not it is an image of a frog.

- Is this an example of classification or regression?

Exercise

- Suppose you are building a machine learning algorithm that takes a photo as input and determines the **probability** that it is an image of a frog.

- Is this an example of classification or regression?

Exercise

Suppose you are designing a system to estimate the number of frogs in your neighbourhood pond based on factors like temperature, surface area, algae count, proximity to humans, and so on.

Is this an example of a classification problem or a regression problem?

MNIST

MNIST is a dataset of labeled handwritten digits collected from highschool students and the US census bureau.

- Consists of 60,000 training and 10,000 testing images
- Each image is 28x28 grayscale pixels (values 0-255)

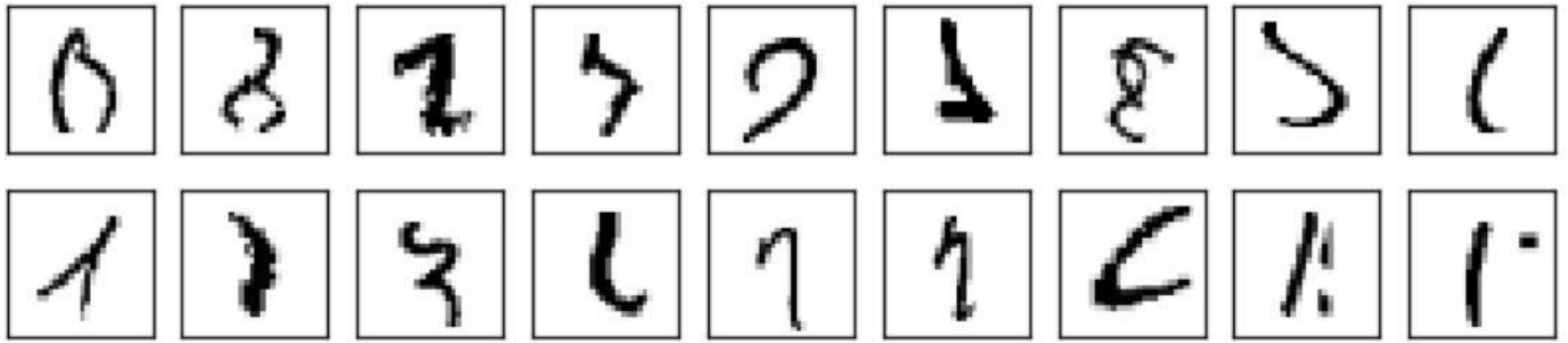
Question

Is this an example of a classification or a regression problem?

MNIST

In the paper MNIST was originally published with, the authors used a support vector machine to obtain an error rate of 0.8% on the test data.

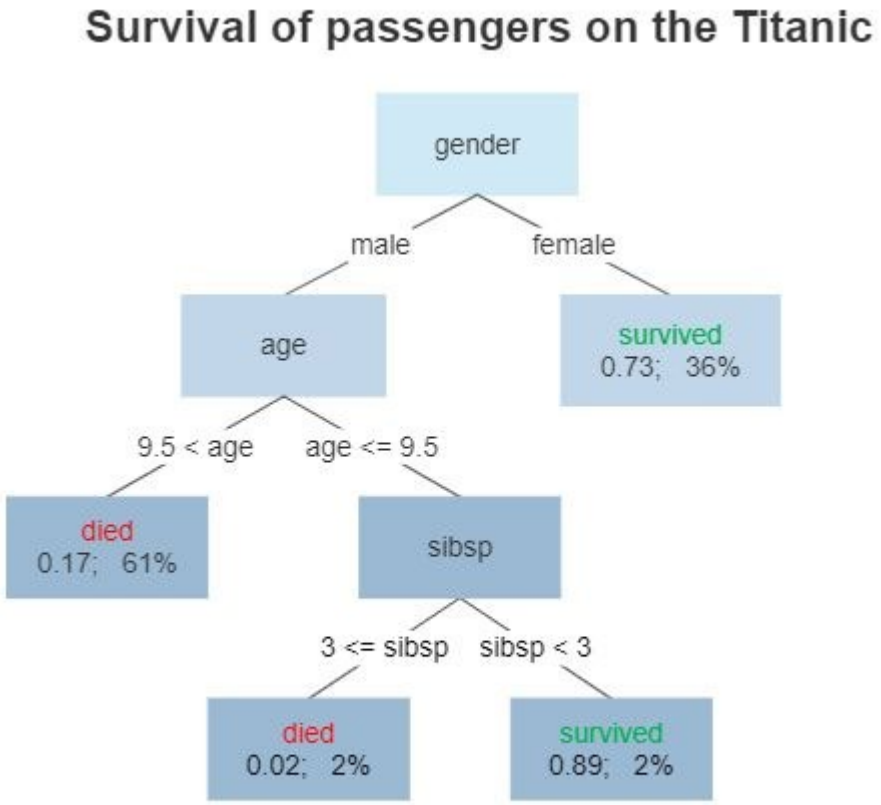
Since then, it has become the de facto dataset for introducing machine learning techniques.



Decision Trees

Decision Trees

- A decision tree is a tree where you enter at the root
- You are presented with decisions at each node in which you pick one path to a lower node in tree
  - Another internal node, or a leaf node
- You continue this making a choice at each node based on your data (an attribute test)
- At the bottom of the tree are leaf/terminal nodes which are the decision the tree makes based on the data

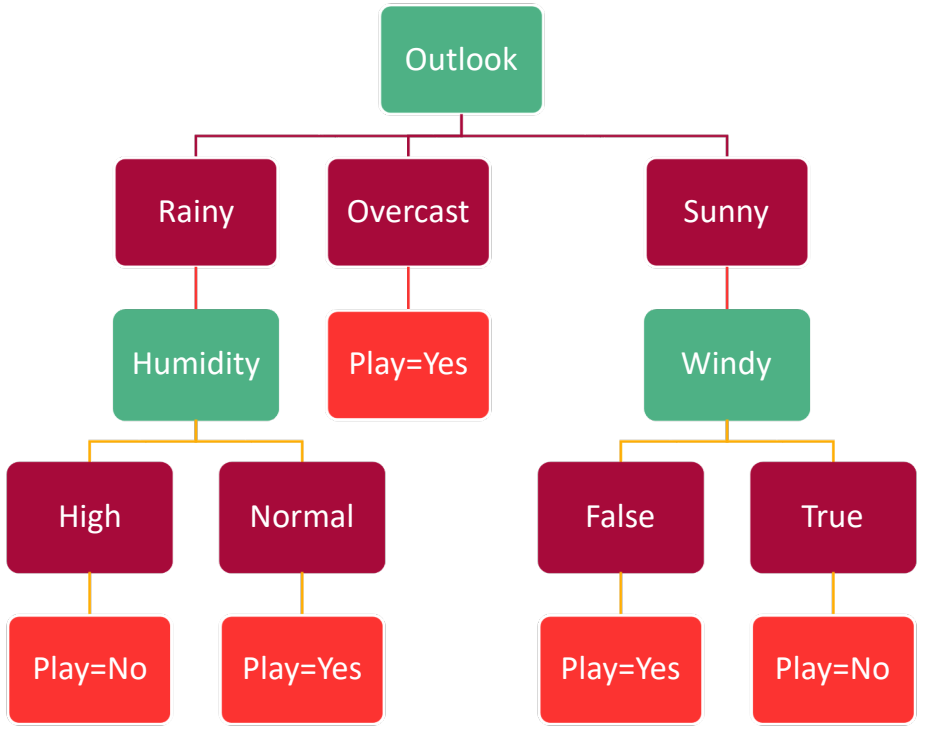


Example

Predictors | Target

Outlook	Temp	Humidity	Windy	Play Golf
Rainy	Hot	High	False	No
Rainy	Hot	High	True	No
Overcast	Hot	High	False	Yes
Sunny	Mild	High	False	Yes
Sunny	Cool	Normal	False	Yes
Sunny	Cool	Normal	True	No
Overcast	Cool	Normal	True	Yes
Rainy	Mild	High	False	No
Rainy	Cool	Normal	False	Yes
Sunny	Mild	Normal	False	Yes
Rainy	Mild	Normal	True	Yes
Overcast	Mild	High	True	Yes
Overcast	Hot	Normal	False	Yes
Sunny	Mild	High	True	No

Decision Tree



Usages

- Used for classification
  - Example: based on the game state, is the team going to win or lose the game
- Used for regression
  - Example: based on the game state, how many points is the team going to score before they lose possession

How to make

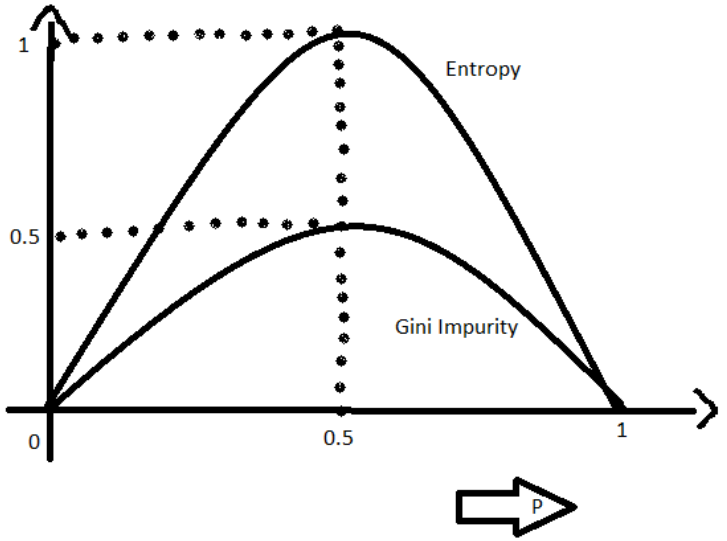
1. All dataset outcomes are at root
2. Select an attribute with the most **benefit**
3. Split dataset on attribute to make leaf nodes
4. Now you have a root with 2 or more leaves (often 2)
5. Now return to step 1 for each new leaf node

- Repeat above until a designate stopping point

What do we mean by benefit on split

1. Gini Impurity
   - If you took an input (a row) and classified it randomly, how likely is it to be in the wrong class
   - Used in CART
2. Entropy
   - The amount of impurity in the dataset
   - Used in ID3 and C4.5

- Gini often preferred as it measures the same relative idea without need of a logarithm (computationally more work)
  - There are other differences we won't go into
- Information Gain
  - Reduction in either Gini or Entropy after being split on an attribute



$$E(S) = \sum_{i=1}^{c} - p_i \log_2 p_i$$

$$Gini(E) = 1 - \sum_{j=1}^{c} p_j^2$$

ID3 Example: What is entropy of result

- Entropy of Play Golf

Play Golf	
Yes	No
9	5

- $p_{yes} = \frac{9}{14} \approx 0.64$
- $p_{no} = \frac{5}{14} \approx 0.36$
- $Entropy(PlayGolf)$
- $= -(0.64 \log_2 0.64) - (0.36 \log_2 0.36)$
- =0.94

Outlook	Temp	Humidity	Windy	Play Golf
Rainy	Hot	High	False	No
Rainy	Hot	High	True	No
Overcast	Hot	High	False	Yes
Sunny	Mild	High	False	Yes
Sunny	Cool	Normal	False	Yes
Sunny	Cool	Normal	True	No
Overcast	Cool	Normal	True	Yes
Rainy	Mild	High	False	No
Rainy	Cool	Normal	False	Yes
Sunny	Mild	Normal	False	Yes
Rainy	Mild	Normal	True	Yes
Overcast	Mild	High	True	Yes
Overcast	Hot	Normal	False	Yes
Sunny	Mild	High	True	No

ID3 Example: What is entropy of attribute

- Entropy of Outlook

		Play Golf		
		Yes	No	
Outlook	Sunny	3	2	5
	Overcast	4	0	4
	Rainy	2	3	5
				14

- $Entropy(PlayGolf, Outlook)$
- $= \sum_{x \in Outlook} P(x) Entropy(x)$
- $= P(sunny)E(3,2) + P(overcast)E(4,0) + P(Rainy)E(2,3)$
- =0.693

Outlook	Temp	Humidity	Windy	Play Golf
Rainy	Hot	High	False	No
Rainy	Hot	High	True	No
Overcast	Hot	High	False	Yes
Sunny	Mild	High	False	Yes
Sunny	Cool	Normal	False	Yes
Sunny	Cool	Normal	True	No
Overcast	Cool	Normal	True	Yes
Rainy	Mild	High	False	No
Rainy	Cool	Normal	False	Yes
Sunny	Mild	Normal	False	Yes
Rainy	Mild	Normal	True	Yes
Overcast	Mild	High	True	Yes
Overcast	Hot	Normal	False	Yes
Sunny	Mild	High	True	No

ID3 Example: Which attribute to use

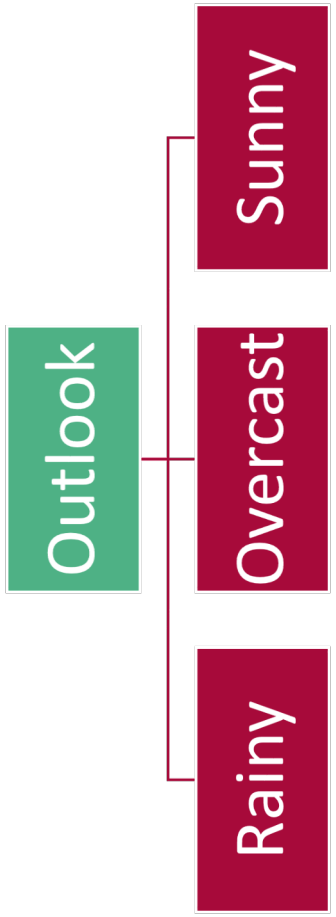
- Pick attribute with best gain
- Gain(PlayGolf, X) = Entropy(PlayGolf) – Entropy(PlayGolf,X)
- Therefore Gain(PlayGolf, Outlook) = 0.940-0.693 = 0.247
- We can do this for all 4 attributes

Attribute	Gain
Outlook	0.247
Temperature	0.029
Humidity	0.152
Windy	0.048

- We will split on outlook first

ID3 Example: First split

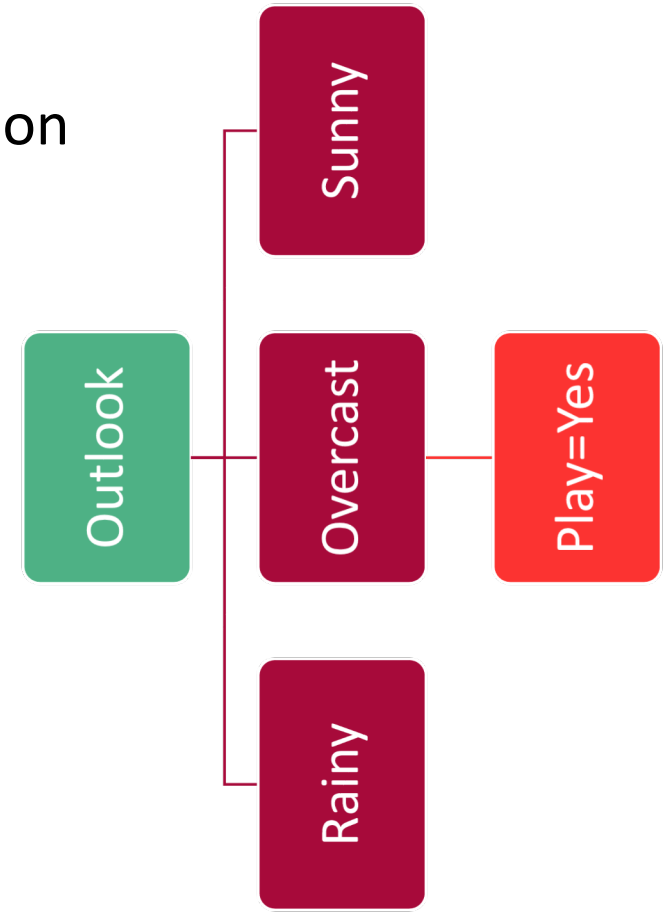
- We will split on outlook first



Outlook	Temp	Humidity	Windy	Play Golf
Sunny	Mild	High	False	Yes
Sunny	Cool	Normal	False	Yes
Sunny	Mild	Normal	False	Yes
Sunny	Cool	Normal	True	No
Sunny	Mild	High	True	No
Overcast	Hot	High	False	Yes
Overcast	Cool	Normal	True	Yes
Overcast	Mild	High	True	Yes
Overcast	Hot	Normal	False	Yes
Rainy	Hot	High	False	No
Rainy	Hot	High	True	No
Rainy	Mild	High	False	No
Rainy	Cool	Normal	False	Yes
Rainy	Mild	Normal	True	Yes

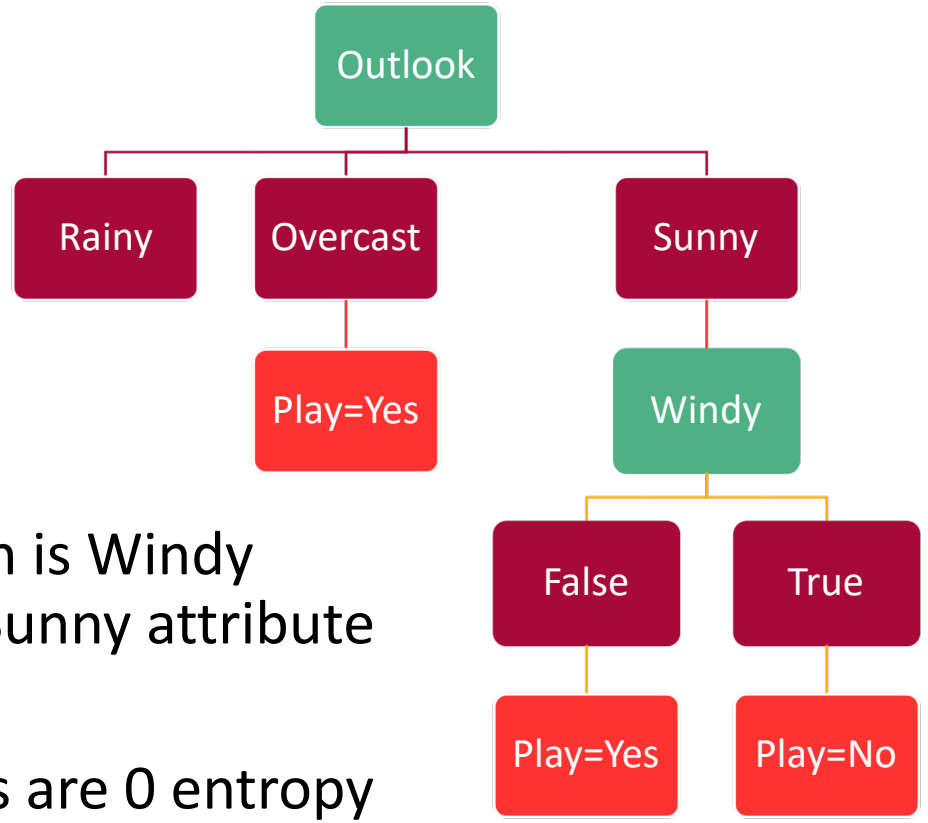
ID3 Example: Entropy 0

- Now we can see all of Overcast have 0 entropy
- So we can point to decision leaf node



Outlook	Temp	Humidity	Windy	Play Golf
Sunny	Mild	High	False	Yes
Sunny	Cool	Normal	False	Yes
Sunny	Mild	Normal	False	Yes
Sunny	Cool	Normal	True	No
Sunny	Mild	High	True	No
Overcast	Hot	High	False	Yes
Overcast	Cool	Normal	True	Yes
Overcast	Mild	High	True	Yes
Overcast	Hot	Normal	False	Yes
Rainy	Hot	High	False	No
Rainy	Hot	High	True	No
Rainy	Mild	High	False	No
Rainy	Cool	Normal	False	Yes
Rainy	Mild	Normal	True	Yes

ID3 Example: Loop until done

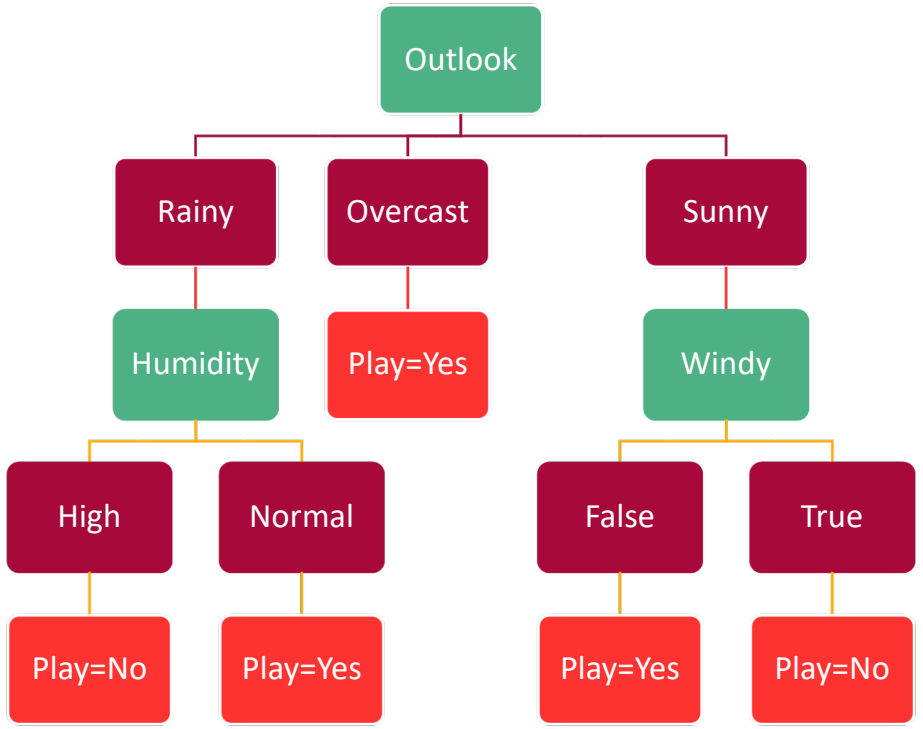


- Next best Gain is Windy Attribute for Sunny attribute sub-tree
- Both branches are 0 entropy
- Rainy will split on Humidity

Outlook	Temp	Humidity	Windy	Play Golf
Sunny	Mild	High	False	Yes
Sunny	Cool	Normal	False	Yes
Sunny	Mild	Normal	False	Yes
Sunny	Cool	Normal	True	No
Sunny	Mild	High	True	No
Overcast	Hot	High	False	Yes
Overcast	Cool	Normal	True	Yes
Overcast	Mild	High	True	Yes
Overcast	Hot	Normal	False	Yes
Rainy	Hot	High	False	No
Rainy	Hot	High	True	No
Rainy	Mild	High	False	No
Rainy	Cool	Normal	False	Yes
Rainy	Mild	Normal	True	Yes

ID3 Example: Loop until done

- Complete tree



Outlook	Temp	Humidity	Windy	Play Golf
Sunny	Mild	High	False	Yes
Sunny	Cool	Normal	False	Yes
Sunny	Mild	Normal	False	Yes
Sunny	Cool	Normal	True	No
Sunny	Mild	High	True	No
Overcast	Hot	High	False	Yes
Overcast	Cool	Normal	True	Yes
Overcast	Mild	High	True	Yes
Overcast	Hot	Normal	False	Yes
Rainy	Hot	High	False	No
Rainy	Hot	High	True	No
Rainy	Mild	High	False	No
Rainy	Cool	Normal	False	Yes
Rainy	Mild	Normal	True	Yes

ID3 Example: Loop until done

- Change to rules by following from root to each leaf

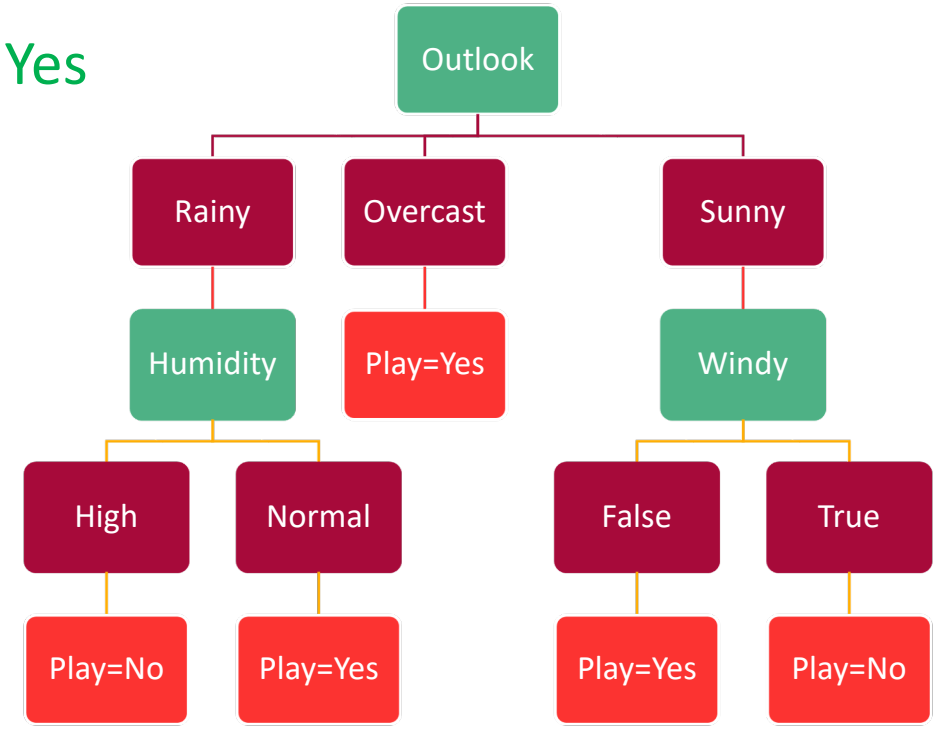
If Outlook is Rainy and Humidity is High, then Play is No

If Outlook is Rainy and Humidity is Normal, then Play is Yes

If Outlook is Overcast, then Play is Yes

If Outlook is Sunny and Windy is False, then Play is Yes

If Outlook is Sunny and Windy is True, then Play is No



Optimal Decision Trees

- Too large trees will overfit
- Pruning will delete nodes as long as accuracy remains relatively the same as before deletion

1. Cost Complexity Pruning
2. Reduced Error Pruning

Applications

- Decision Making
- Should I buy something, make a deal, go for it on 4th down

- Business Management
- Customer Relationship Management
- Fraudulent Statement Detection
- Energy Consumption
- Healthcare Management
- Fault Diagnosis

Advantages

1. Compared to other algorithms decision trees requires less effort for data preparation during pre-processing.
2. A decision tree does not require normalization of data.
3. A decision tree does not require scaling of data as well.
4. Missing values in the data also do NOT affect the process of building a decision tree to any considerable extent.
5. A Decision tree model is very intuitive and easy to explain to technical teams as well as stakeholders.

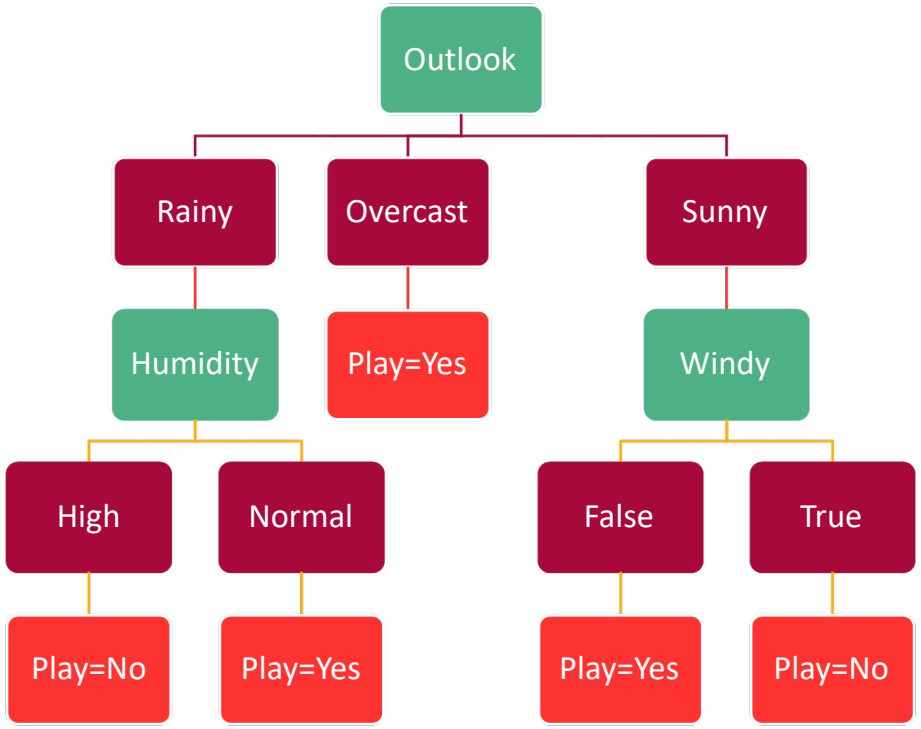
Disadvantages

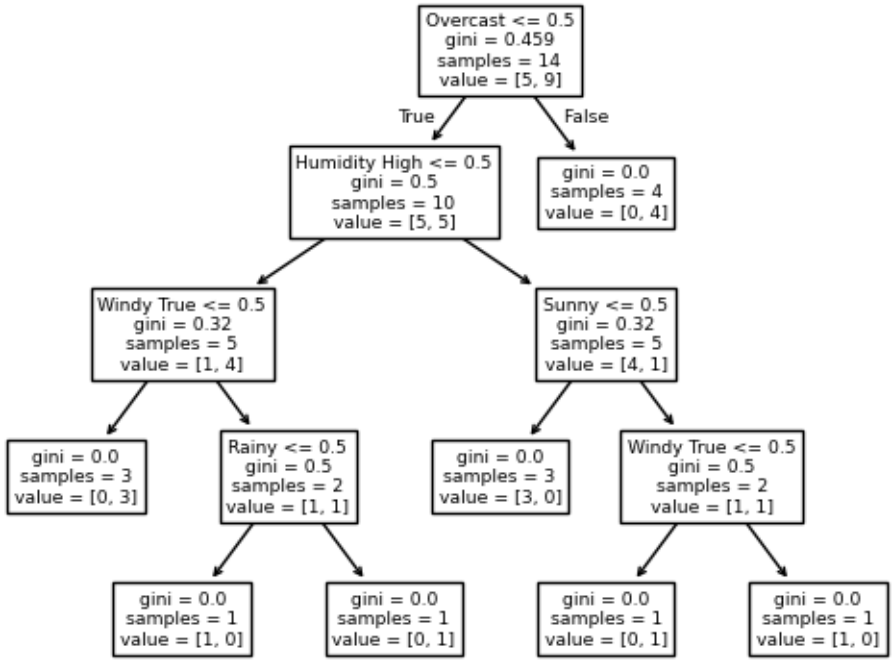
1. A small change in the data can cause a large change in the structure of the decision tree causing instability.
2. The decision tree contains lots of layers, which makes it complex.
3. Decision tree often involves higher time to train the model.
4. It may have an overfitting issue, which can be resolved using the **Random Forest algorithm (next section).**

Sci-Kit

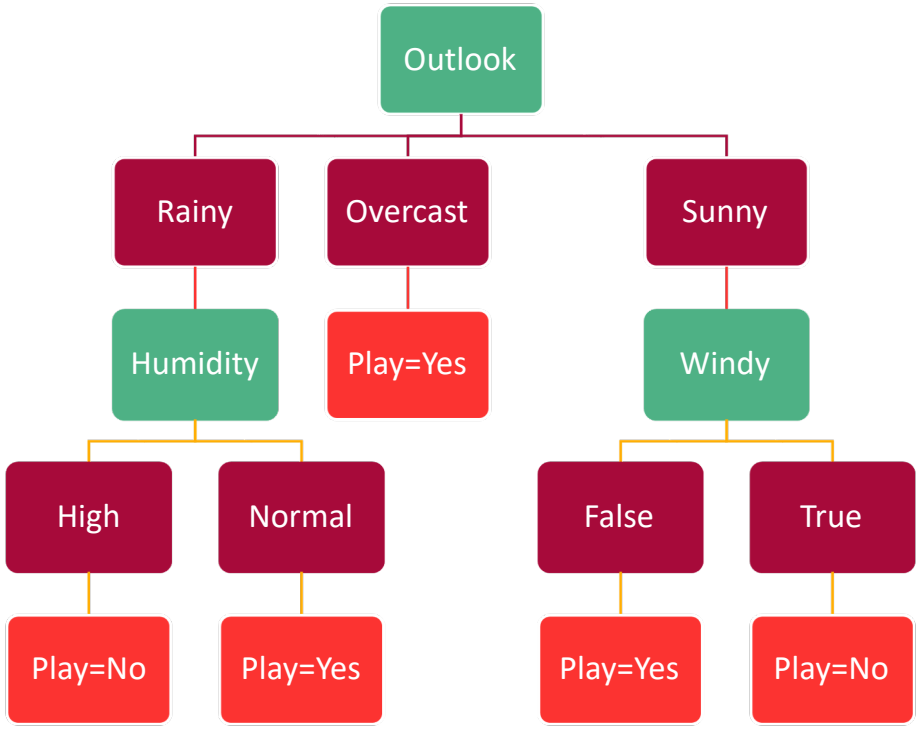
```
data = []
data.append(["Sunny", "Mild", "High", "False", "Yes"])
data.append(["Sunny", "Cool", "Normal", "False", "Yes"])
data.append(["Sunny", "Mild", "Normal", "False", "Yes"])
data.append(["Sunny", "Cool", "Normal", "True", "No"])
data.append(["Sunny", "Mild", "High", "True", "No"])
data.append(["Overcast", "Hot", "High", "False", "Yes"])
data.append(["Overcast", "Cool", "Normal", "True", "Yes"])
data.append(["Overcast", "Mild", "High", "True", "Yes"])
data.append(["Overcast", "Hot", "Normal", "False", "Yes"])
data.append(["Rainy", "Hot", "High", "False", "No"])
data.append(["Rainy", "Hot", "High", "True", "No"])
data.append(["Rainy", "Mild", "High", "False", "No"])
data.append(["Rainy", "Cool", "Normal", "False", "Yes"])
data.append(["Rainy", "Mild", "Normal", "True", "Yes"])
```

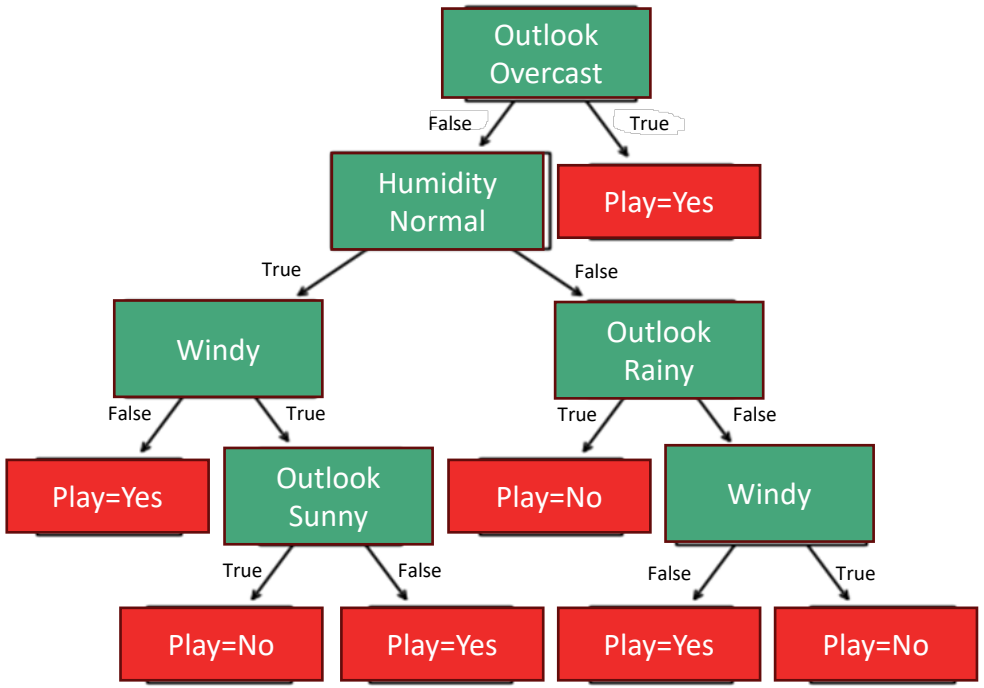
Sci-Kit





Sci-Kit





UNIVERSITY OF CALGARY

Random Forest

Random Forest

- Decision trees and Random Forests take the same input and produce the same output
- A structure you can create with data, and that when you feed it future data it either
  - Classifies it into a category label (Play Golf or Not)
  - Or produces a regression (a predicated value)

- What is different
- Random Forests are an ensemble method
- Ensemble methods bring together multiple models/classifiers at one time and combines their results
- In random FOREST is made up of many decision TREES

Random Forest

- Essentially
- Build many decisions trees on data
- For each decision tree allow internal node creation to be made with stochastic factors around indeterminate/unclear choices (which attribute next, which exact value for attribute is best split point)
- Make multiple trees (200!, more?)
- When we predict something we average the combination of all the outputs

Advantages

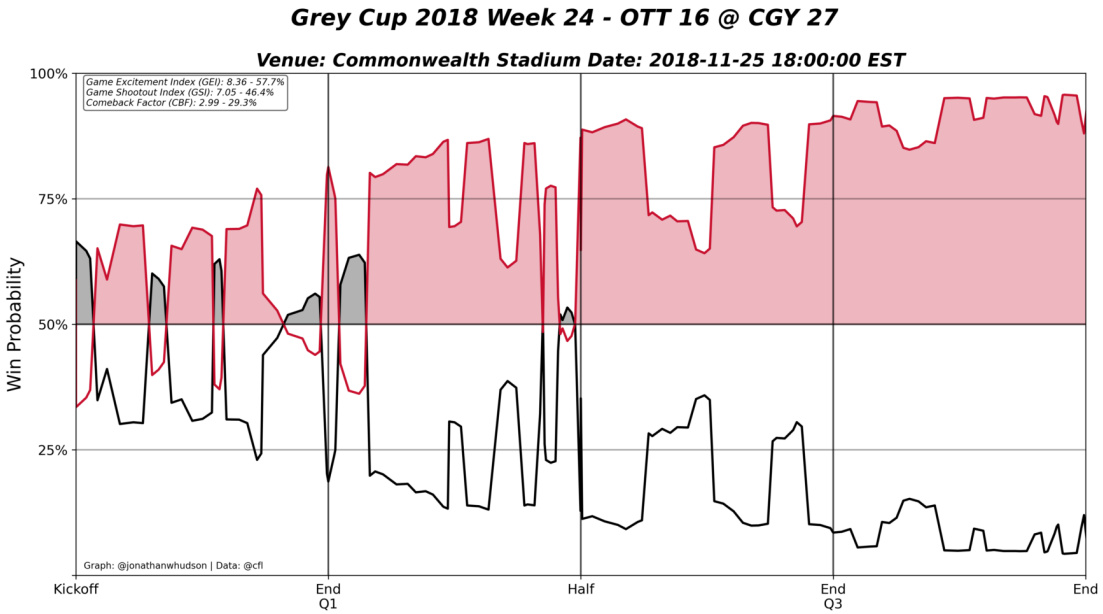
- Less prone to overfitting due to ensemble average, each individual tree could be overfit, but we don't care as we expect some alternatives overfit counter to that
- Often rather stable if data changes slightly
- Maybe some % of ensemble trees are less accurate with a new data point, but others are likely robust to that change
- Preferred for large complex datasets (where you often wouldn't try to interpret tree visually anyway)
- Good with outliers

Disadvantages

- Much harder to interpret as you can't just follow one tree, you need to interpret 200!, more!?!
- Training time can be 200X for 200 trees (although often the measures that slow down one decision tree training when preventing overfitting can be relaxed to speed up time)
  - However there are often methods used to make sure ensemble trees are diverse which add their own training time
- Long time to predict (just like training) run through 200! trees
- Hard to nail down how important one feature is to method, as it is dependent on each tree itself

Usage Scenario

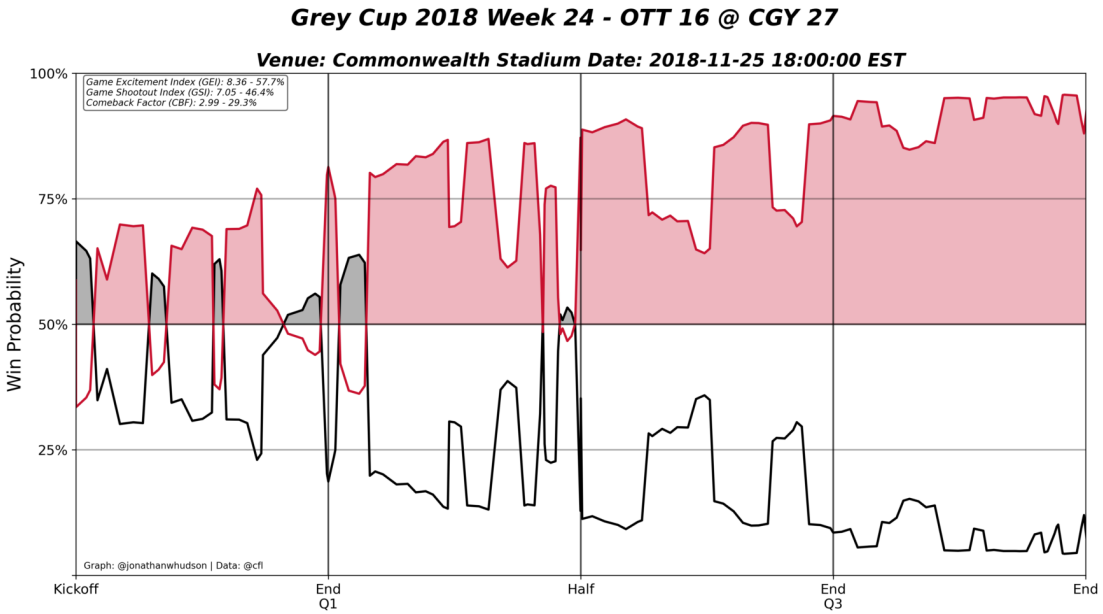
- It is popular in sports to make win predications
- i.e. this is the state of game, will a team win the game
- You do this my taking game states from past games, and then whether or not the team won, and training a classifier on it (yes/no)
- With this you can then either ask single questions about a game state, Will my team come back? Or repetitive questions to chart the back and forth of a competition.



UNIVERSITY OF CALGARY

Usage Scenario

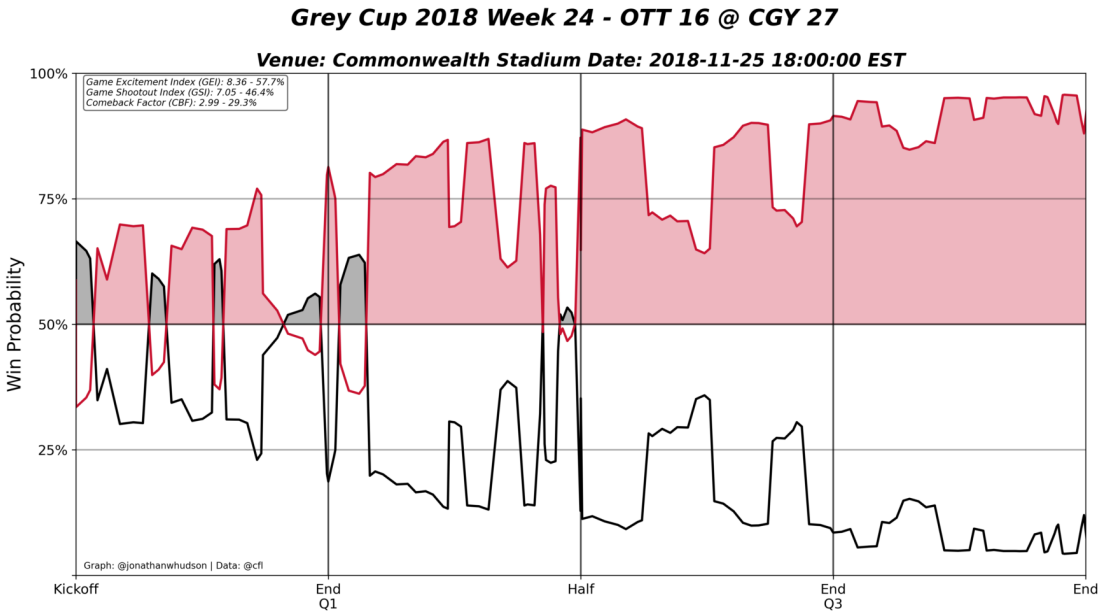
- You can do this with a neural network
  - High uninterpretable and often takes a reasonable large amount of computer power for training
  - The neural network has no reason not to learn weird ideas when data is sparse
    - In general if you are losing my more points ,you are less likely to win
    - But if there is one data point where a team won at -25 points with 18 minutes left, maybe a neural network will learn this input means it should predict a win, despite rest of relationship not following that



UNIVERSITY OF CALGARY

Usage Scenario

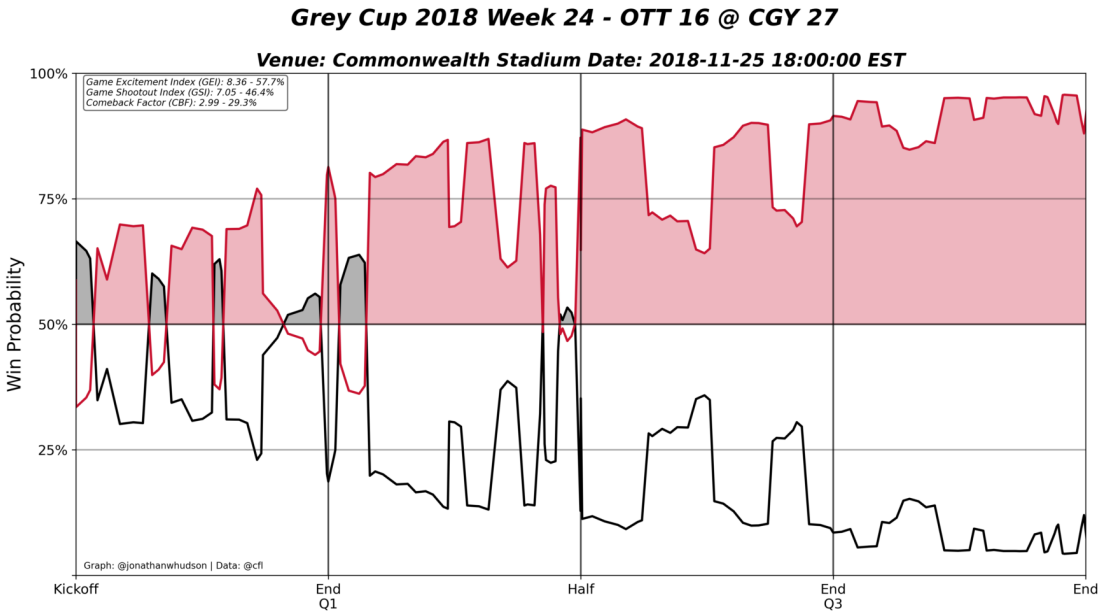
- A decision tree could be used
- However one decision tree would be highly susceptible again to outliers, and would likely easily overfit
- But it would be interpretable to explore why it thinks your team will win
- It also wouldn't take a lot of compute power, one decision trees can be very very efficient (relative to neural network), train/predict



UNIVERSITY OF CALGARY

Usage Scenario

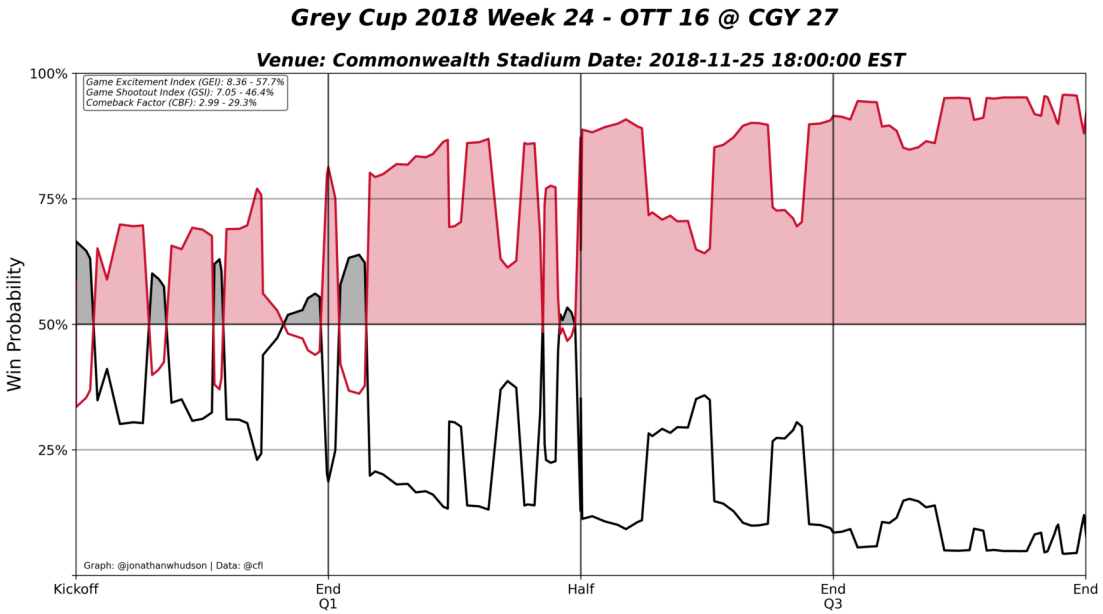
- Or you could use random forest
- Robustness to overfitting and outliers
- Still relatively simple to train
- Treats numerical data sanely at decision points that match how games actually work on average
  - -25 points with 18 minutes left is not a secret to winning football games
- In fact those that make win prediction models will prefer a random forest
- https://journalofbigdata.springeropen.com/articles/10.1186/s40537-024-01008-2



UNIVERSITY OF CALGARY

Usage Scenario

- Or you could use random forest
- Robustness to overfitting and outliers
- Still relatively simple to train
- Treats numerical data sanely at decision points that match how games actually work on average
  - -25 points with 18 minutes left is not a secret to winning football games
- In fact those that make win prediction models will prefer a random forest
- https://journalofbigdata.springeropen.com/articles/10.1186/s40537-024-01008-2

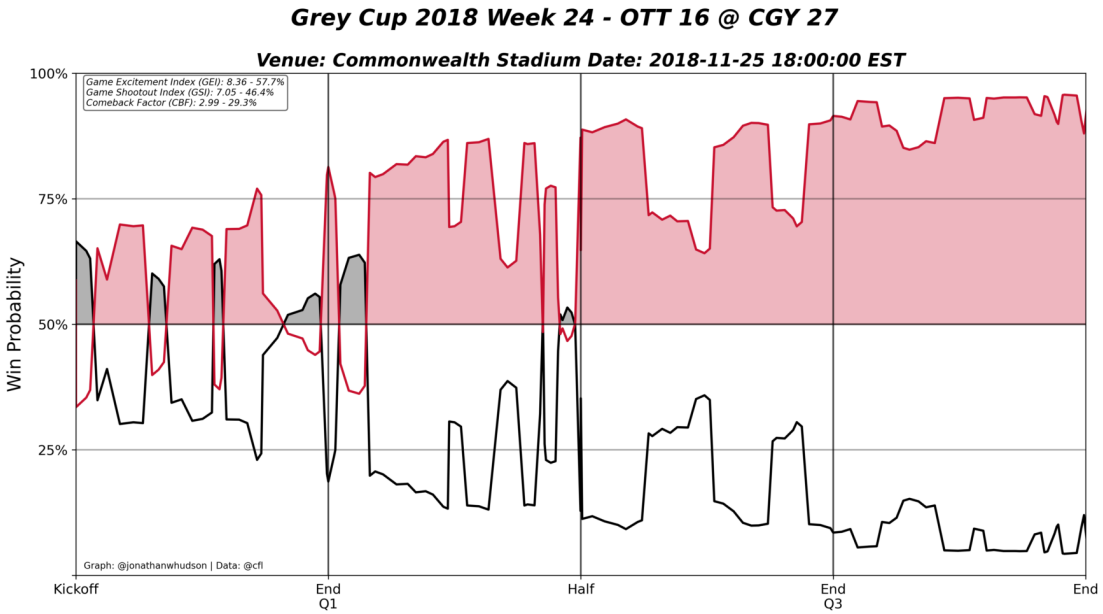


UNIVERSITY OF CALGARY

Usage Scenario

- What's great in sklearn the models are swappable with one line adjustment

```
model_wp = RandomForestClassifier(
        n_estimators=500,
        max_leaf_nodes=200,)
```

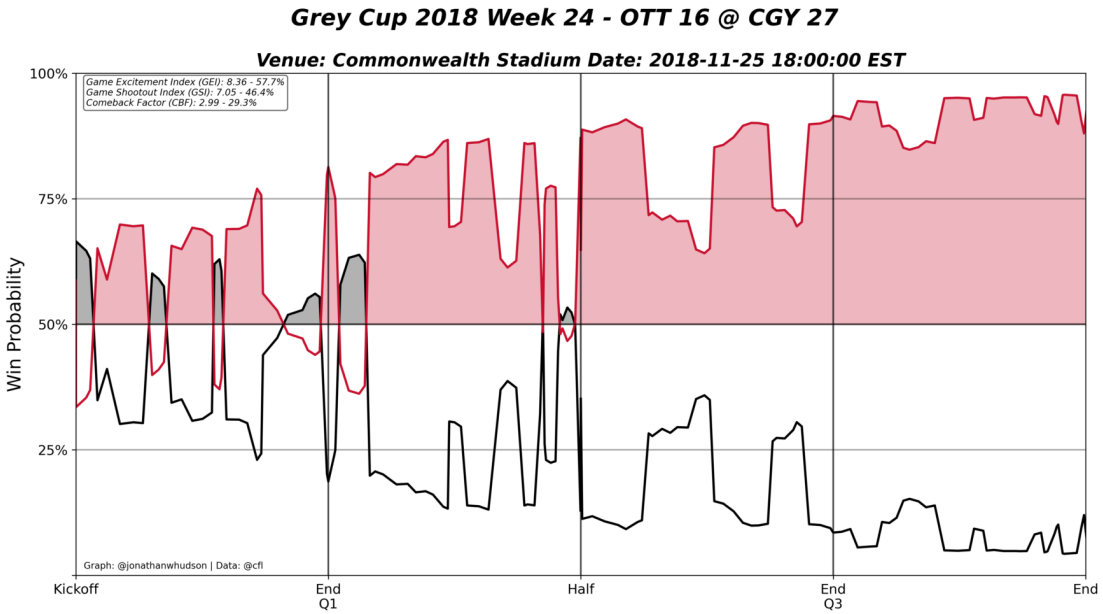


UNIVERSITY OF CALGARY

Usage Scenario

- What's great in sklearn the models are swappable with one line adjustment

```
model_wp = MLPClassifier(
        hidden_layer_sizes=(100,100,100,),
        activation='relu',
        solver='adam')
```

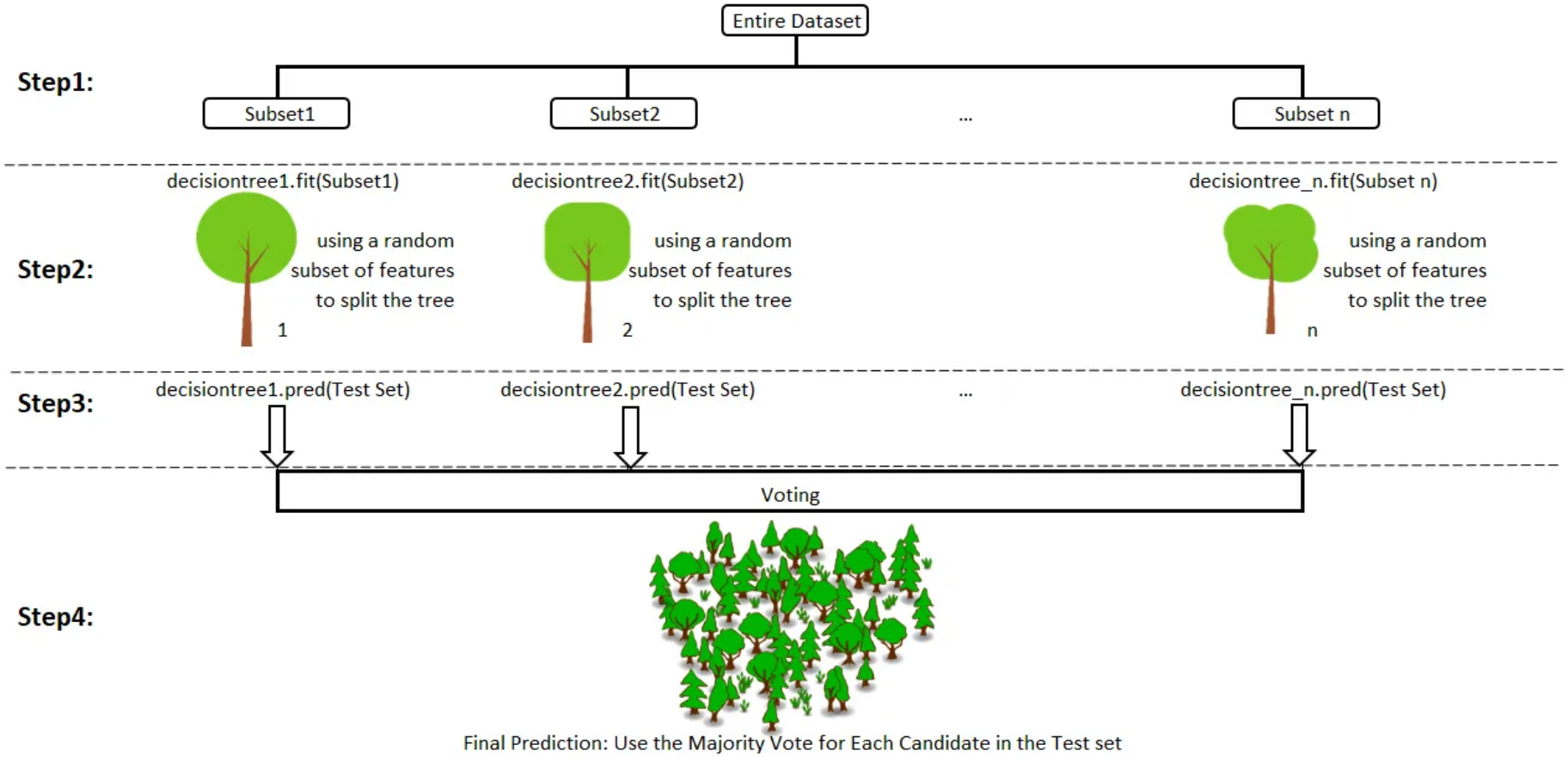


UNIVERSITY OF CALGARY

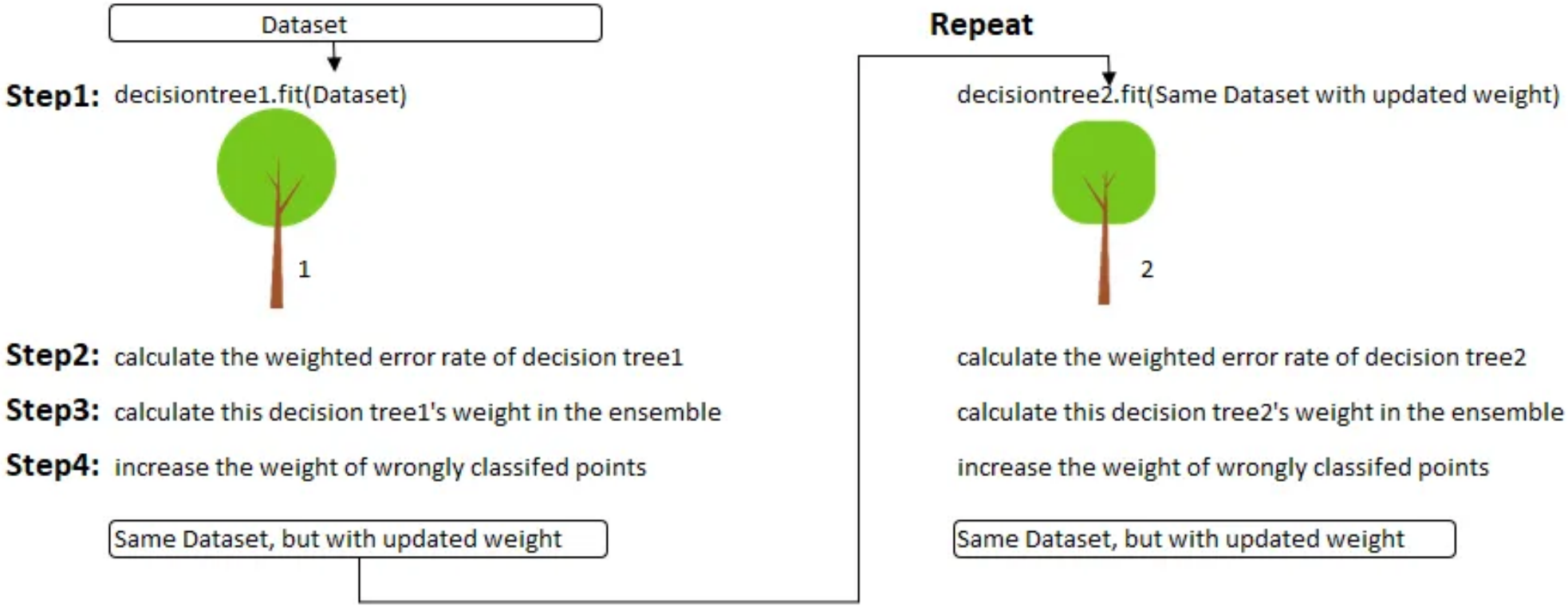
Bagging and boosting

- There are two main ways to make an ensemble method
- Bagging
  - Make each model independently, with likely only stochastic factors that result in each being different (data used, decisions varied or randomized, etc.)
- Boosting
  - Make one model, then to make the next look at what that model is bad at, design next model to be good at those things first

Bagged method of random forest



Adaboost (boosting method)



Next…reinforcement learning

Jonathan Hudson, Ph.D.
jwhudson@ucalgary.ca
https://cspages.ucalgary.ca/~jwhudson/

UNIVERSITY OF CALGARY