Machine Learning: Optional Other SciKit

CPSC 501: Advanced Programming Techniques
Winter 2025

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

Friday, February 21, 2025

UNIVERSITY OF CALGARY

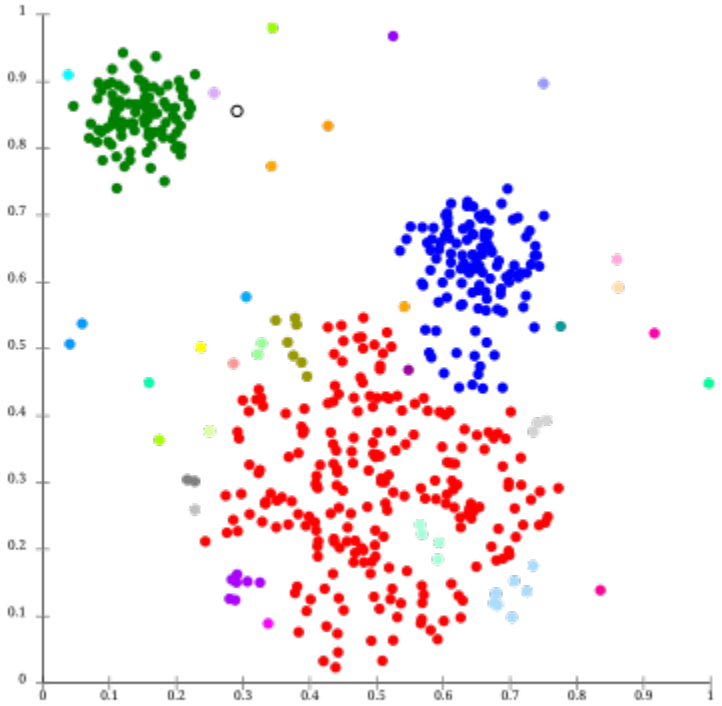
Unsupervised: Clustering

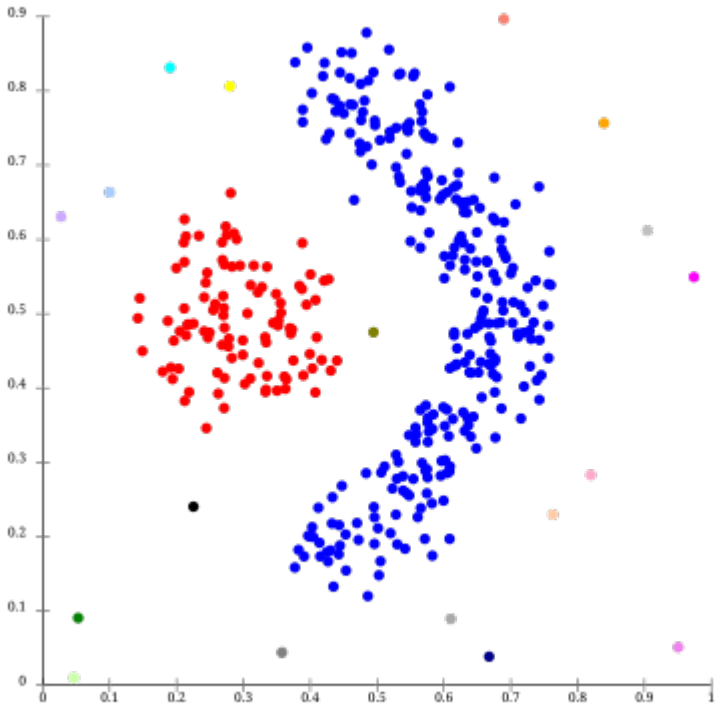
Clustering

- Grouping unlabeled data based on similarities
- Find patterns
- Sometimes we have minor direction like
  - How many clusters
  - Quality of a good cluster
  - Relationship of clusters

Clustering Goals

- Clusters goal may have a desired shape
  - Ex. Circular
- Or maybe not be restrained to a particular shape
- Goal can also be to force every point to be part of a cluster (hard clustering), or soft where goal is points end up with likelihood of membership





Clustering

- Most common intro example
  - K-means clustering
    - Ask for k clusters of data
    - algorithm attempts to create them
  - Often usage has you run k-means with differing sizes and contrasting results

K-Means

Initialize k **means** with random values

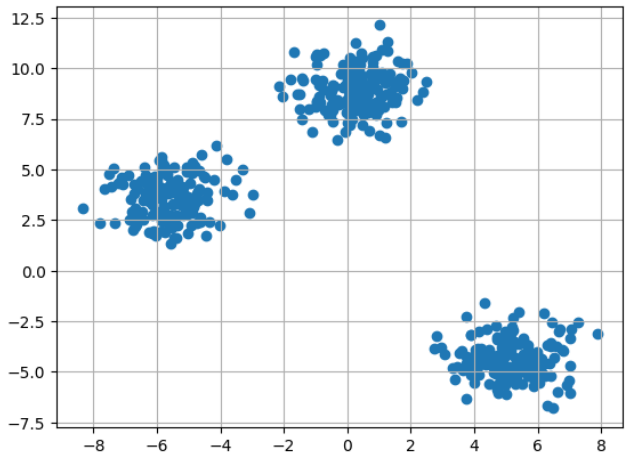
While more iterations left:

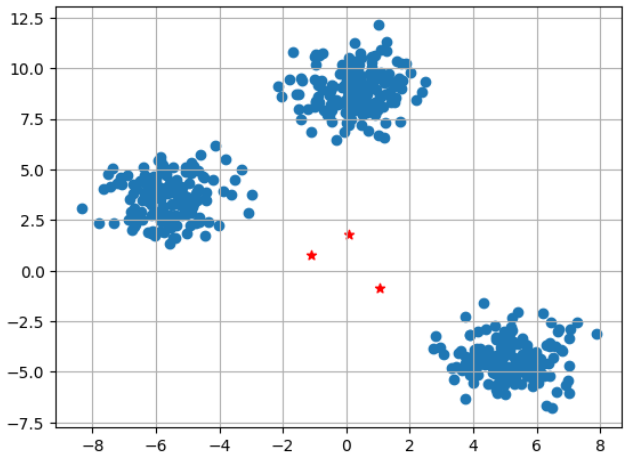
Loop through all the items

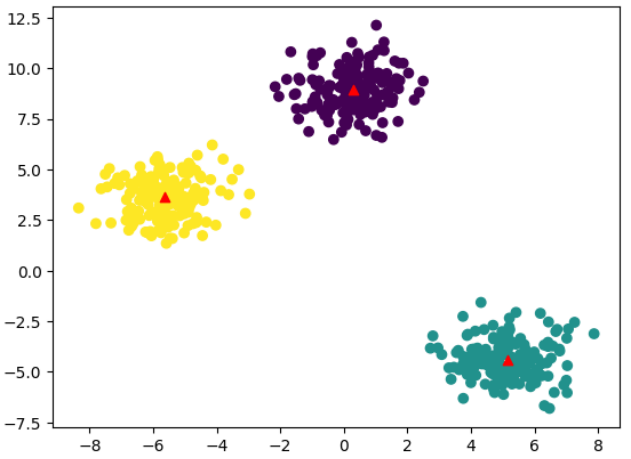
Find the **mean** closest to that item

Assign item to cluster of that **mean**

Update **mean** by shifting it to average of its cluster







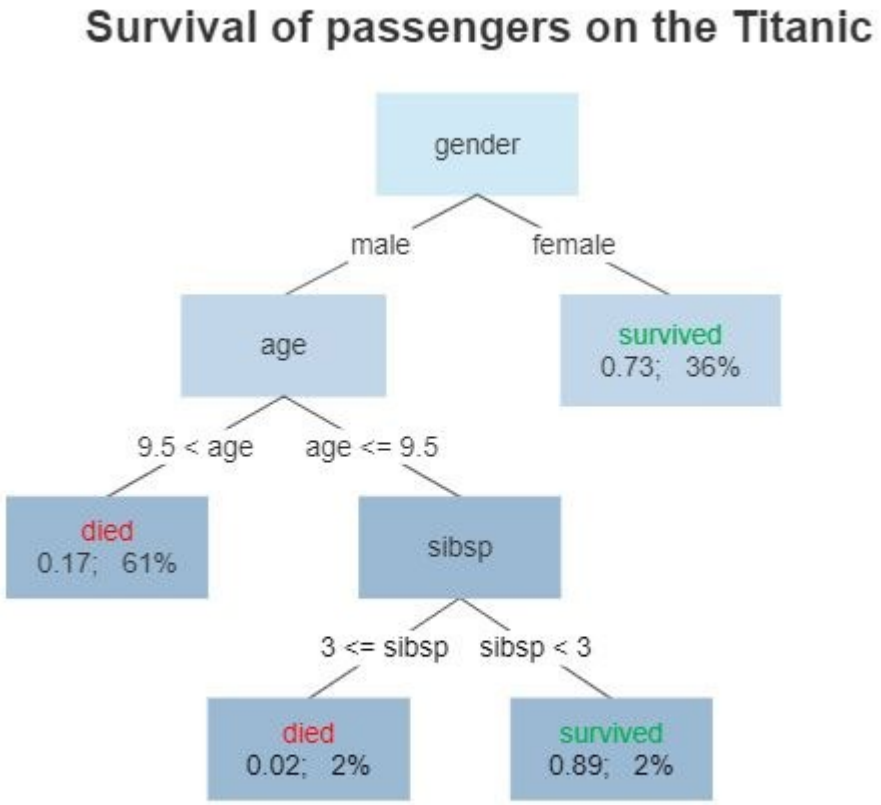
Sci Kit

```
from sklearn.cluster import Kmeans

kmeans = KMeans(n_clusters = 3, random_state = 2)
kmeans.fit(X)

pred = kmeans.fit_predict(X)

#Plot data is the same
plt.scatter(X[:,0],X[:,1],c = pred)
#We use use kmeans cluster data
for i in kmeans.cluster_centers_:
    plt.scatter(i[0],i[1],marker = '^',c = 'red')
plt.show()
```

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

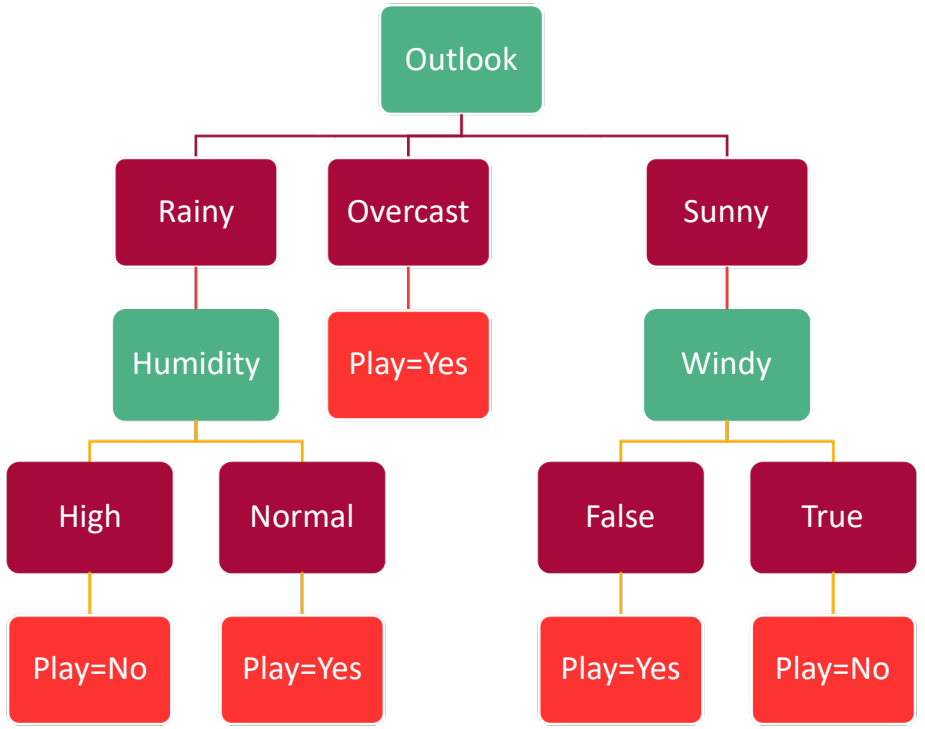


Example

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



How to make

1. All dataset outcomes are at root
2. Select an attribute with the most **benefit (use info theory concept called entropy -> information gained (ex. Best split of groups, think binary search))**
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

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

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

```
import numpy as np
from sklearn import preprocessing
from sklearn.tree import DecisionTreeRegressor
from sklearn.tree import plot_tree
oe = preprocessing.OrdinalEncoder()
oe.fit(data)
data = oe.transform(data)
dataset = np.array(data)
```

```
[[2. 2. 0. 0. 1.]
[2. 0. 1. 0. 1.]
[2. 2. 1. 0. 1.]
[2. 0. 1. 1. 0.]
[2. 2. 0. 1. 0.]
[0. 1. 0. 0. 1.]
[0. 0. 1. 1. 1.]
[0. 2. 0. 1. 1.]
[0. 1. 1. 0. 1.]
[1. 1. 0. 0. 0.]
[1. 1. 0. 1. 0.]
[1. 2. 0. 0. 0.]
[1. 0. 1. 0. 1.]
[1. 2. 1. 1. 1.]]
```

Sci-Kit

```
from sklearn import preprocessing
enc = preprocessing.OneHotEncoder()
enc.fit(data)
data = enc.transform(data).toarray()
#For binary categories drop one column
data = np.delete(data,[7,8,10],1)
import numpy as np
dataset = np.array(data)
```

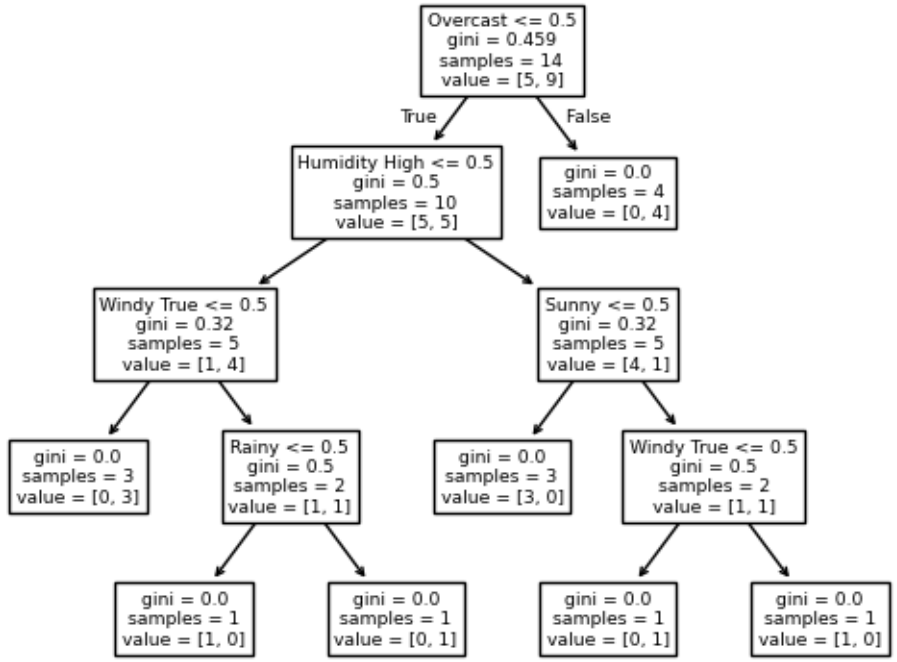
```
[[0. 0. 1. 0. 0. 1. 1. 0. 1.]
 [0. 0. 1. 1. 0. 0. 0. 0. 1.]
 [0. 0. 1. 0. 0. 1. 0. 0. 1.]
 [0. 0. 1. 1. 0. 0. 0. 1. 0.]
 [0. 0. 1. 0. 0. 1. 1. 1. 0.]
 [1. 0. 0. 0. 1. 0. 1. 0. 1.]
 [1. 0. 0. 1. 0. 0. 0. 1. 1.]
 [1. 0. 0. 0. 0. 1. 1. 1. 1.]
 [1. 0. 0. 0. 1. 0. 0. 0. 1.]
 [0. 1. 0. 0. 1. 0. 1. 0. 0.]
 [0. 1. 0. 0. 1. 0. 1. 1. 0.]
 [0. 1. 0. 0. 0. 1. 1. 0. 0.]
 [0. 1. 0. 1. 0. 0. 0. 0. 1.]
 [0. 1. 0. 0. 0. 1. 0. 1. 1.]]
```

OneHot coding is the only way sklearn lets us do categories with inaccurate implication that the categories are numerical related

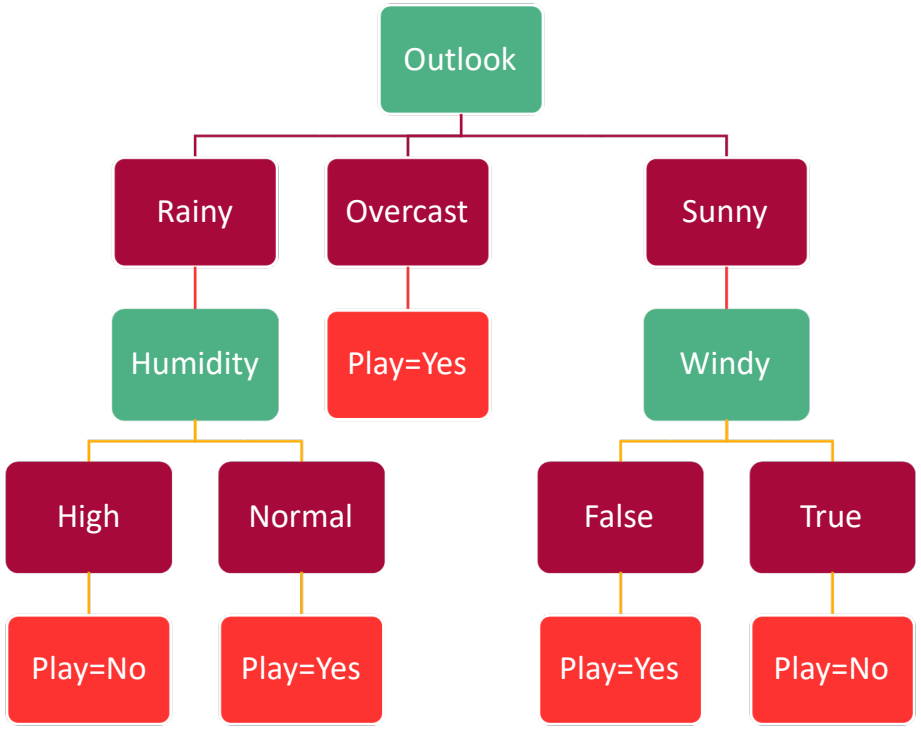
We should not use the prior label encoder as it relates Sunny to Rainy to Overcast as relative numbers

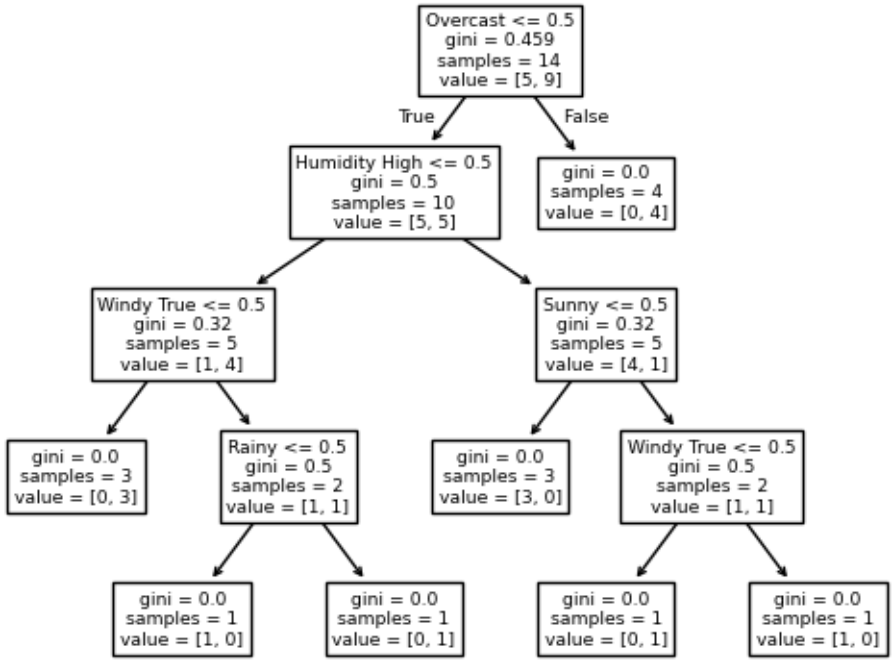
Sci-Kit: CART Decision Trees

```
import numpy as np
from sklearn import preprocessing
from sklearn.tree import DecisionTreeRegressor
from sklearn.tree import plot_tree
oe = preprocessing.OrdinalEncoder()
oe.fit(data)
data = oe.transform(data)
dataset = np.array(data)
X = dataset[:, 0:8].astype(int)
y = dataset[:, 8:9].astype(int)
regressor = DecisionTreeClassifier(random_state = 0)
regressor.fit(X, y)
plot_tree(regressor, feature_names
=['Overcast','Rainy','Sunny','Cool','Hot','Mild','Humidity High','Windy True'])
```



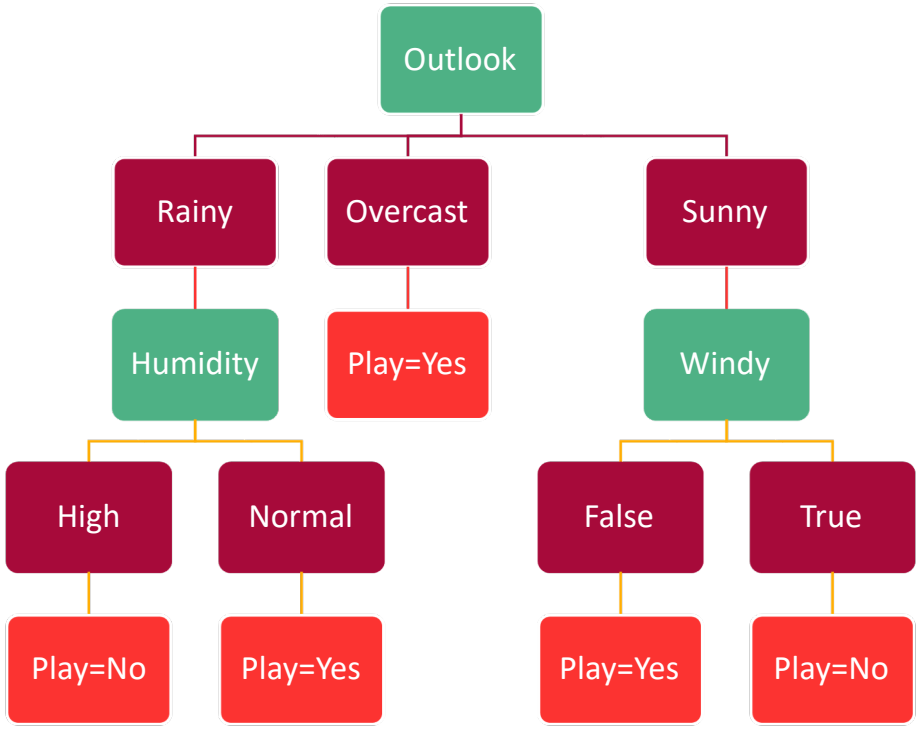
Sci-Kit

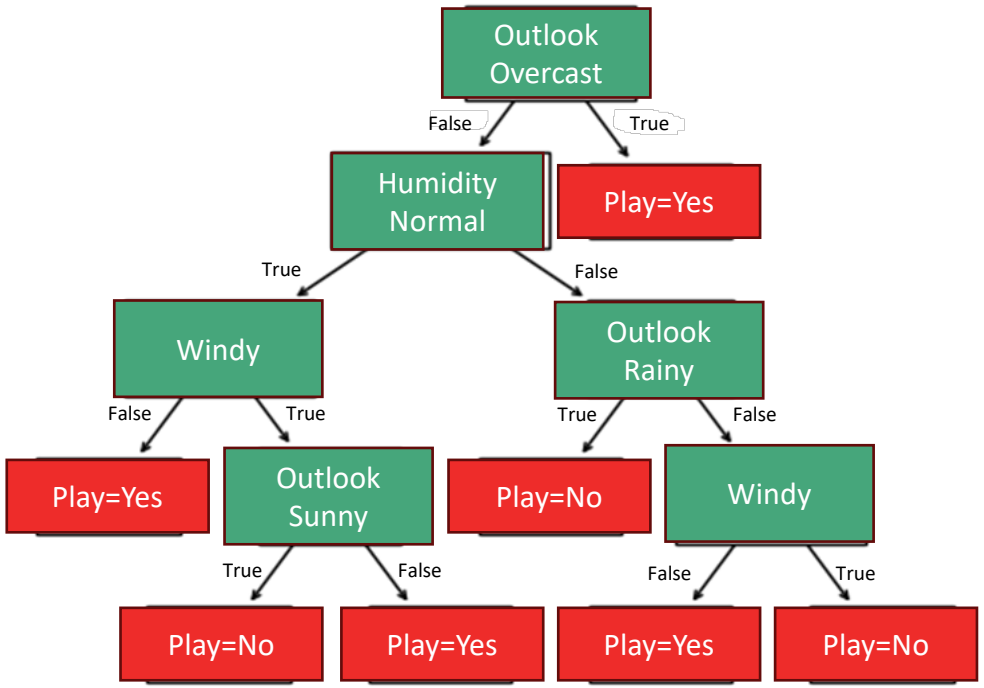




UNIVERSITY OF CALGARY

Sci-Kit



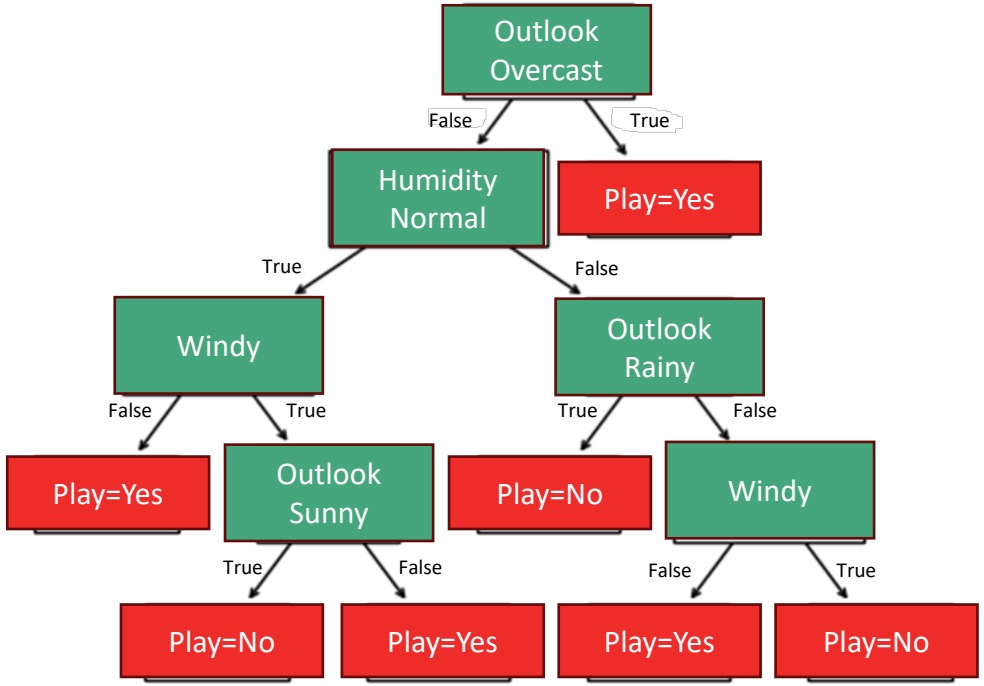


Windy True <= 0.5
Windy False

Rainy <= 0.5
Means Overcast or Sunny
but we've eliminated
Overcast already

Sci-Kit

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



If Outlook is Overcast, then Play is Yes
If Outlook is Sunny or Rainy and Humidity is Normal and Windy is False, then Play is Yes
If Outlook is Sunny and Humidity is Normal and Windy is True, then Play is No
If Outlook is Rainy and Humidity is Normal and Windy is True, then Play is Yes
If Outlook is Rainy and Humidity is High, then Play is No
If Outlook is Sunny and Humidity is High and Windy is False, then Play is Yes
If Outlook is Sunny and Humidity is High and Windy is True, then Play is No

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)

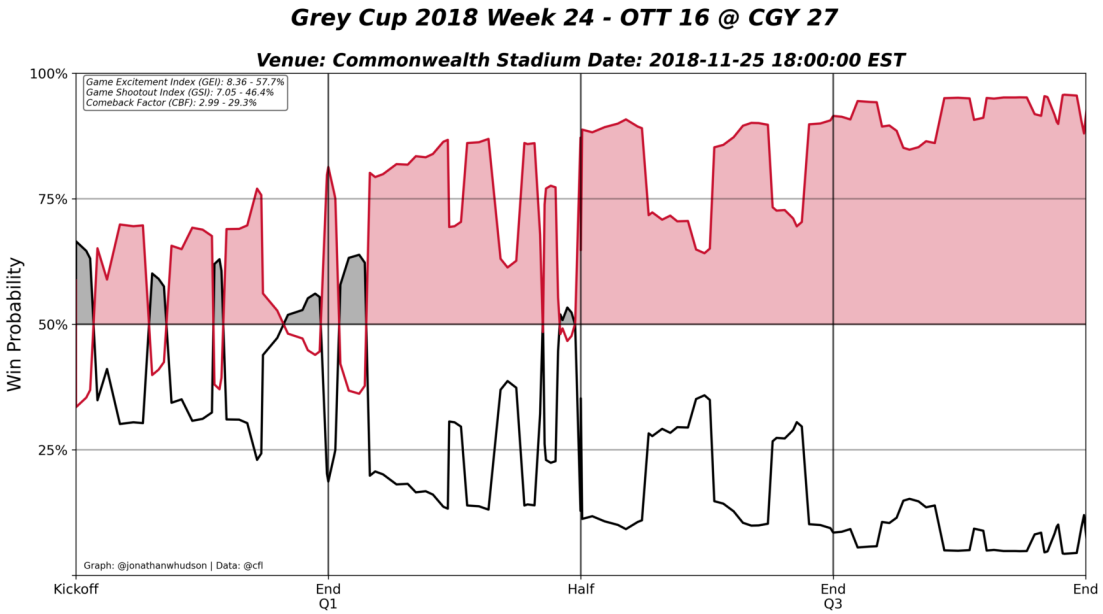
- What is different
- Random Forests are an ensemble method
- Ensemble methods bring together multiple models/classifiers at one time and combines their results
- In random FOREST is made up of many decision TREES

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

Usage Scenario

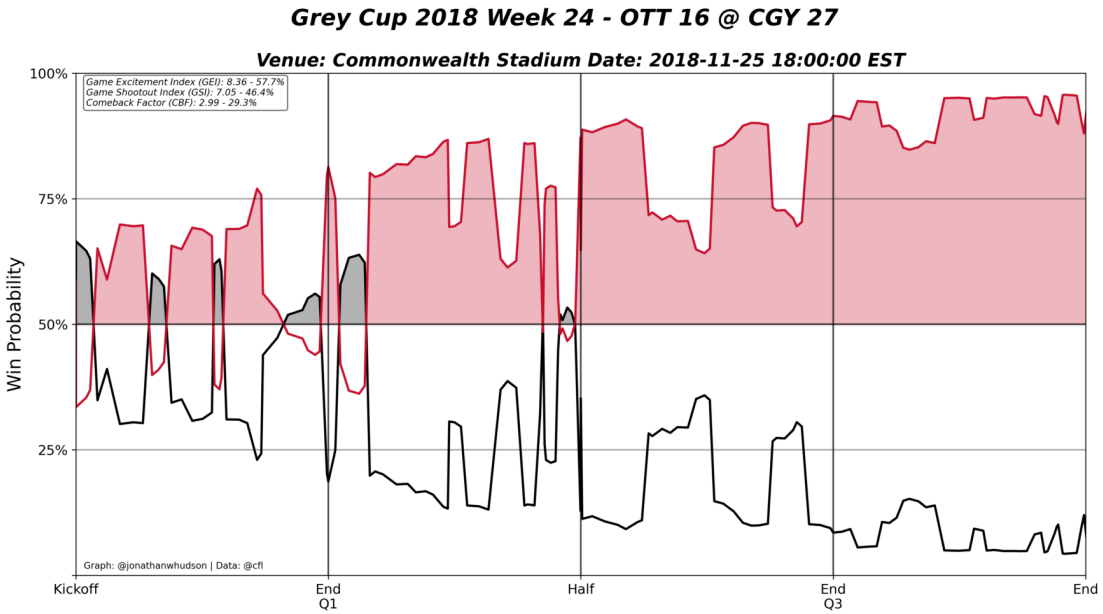
- It is popular in sports to make win predications
- i.e. this is the state of game, will a team win the game
- You do this my taking game states from past games, and then whether or not the team won, and training a classifier on it (yes/no)
- With this you can then either ask single questions about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

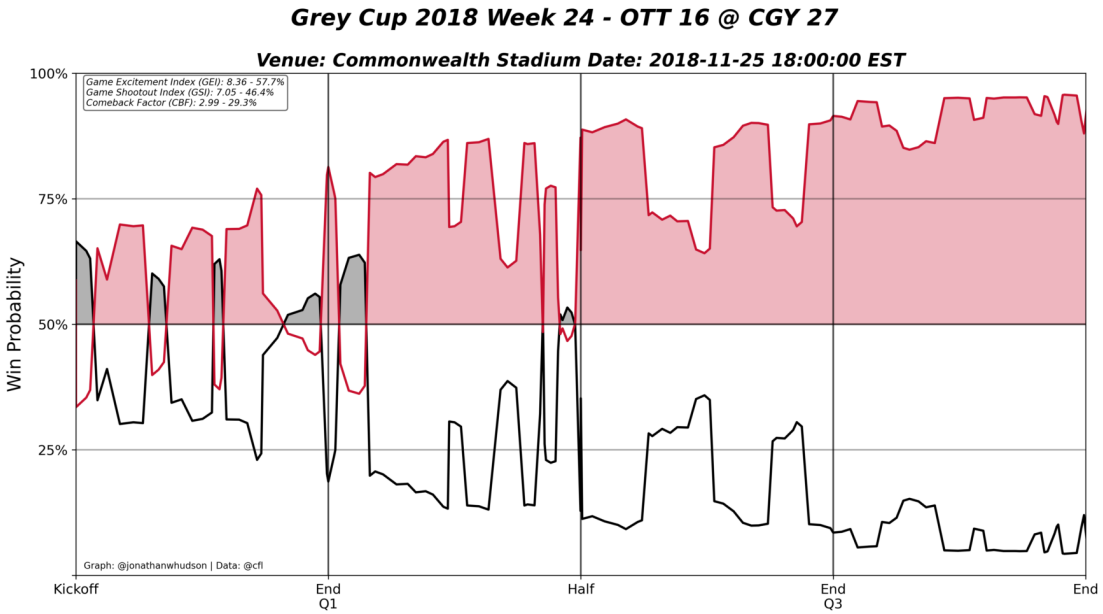
- You can do this with a neural network
  - High uninterpretable and often takes a reasonable large amount of computer power for training
  - The neural network has no reason not to learn weird ideas when data is sparse
    - In general if you are losing my more points ,you are less likely to win
    - 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

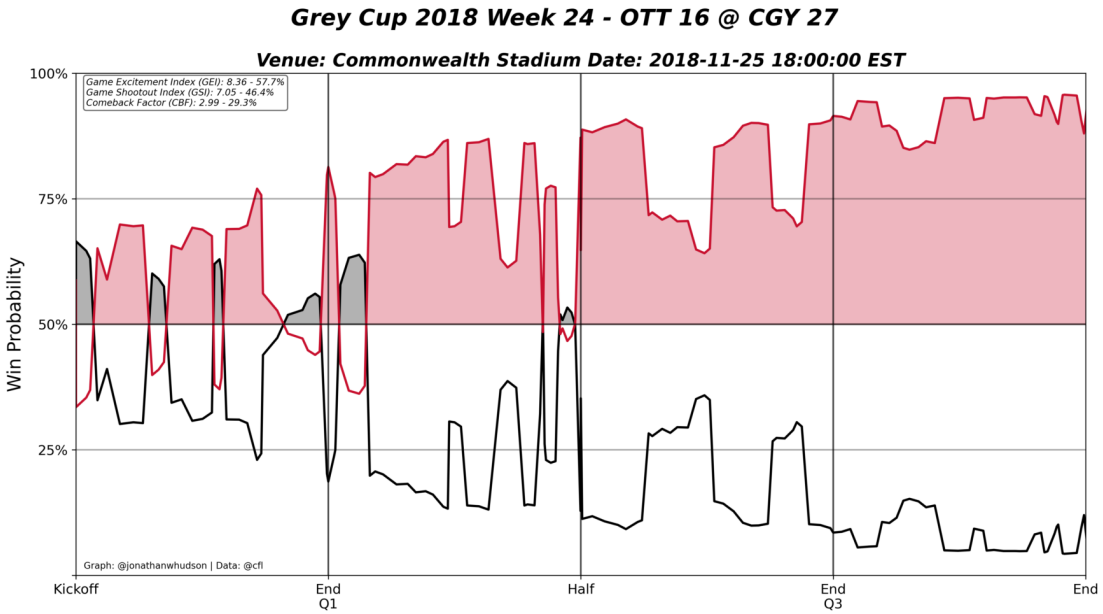
- A decision tree could be used
- However one decision tree would be highly susceptible again to outliers, and would likely easily overfit
- But it would be interpretable to explore why it thinks your team will win
- It also wouldn't take a lot of compute power, one decision trees can be very very efficient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

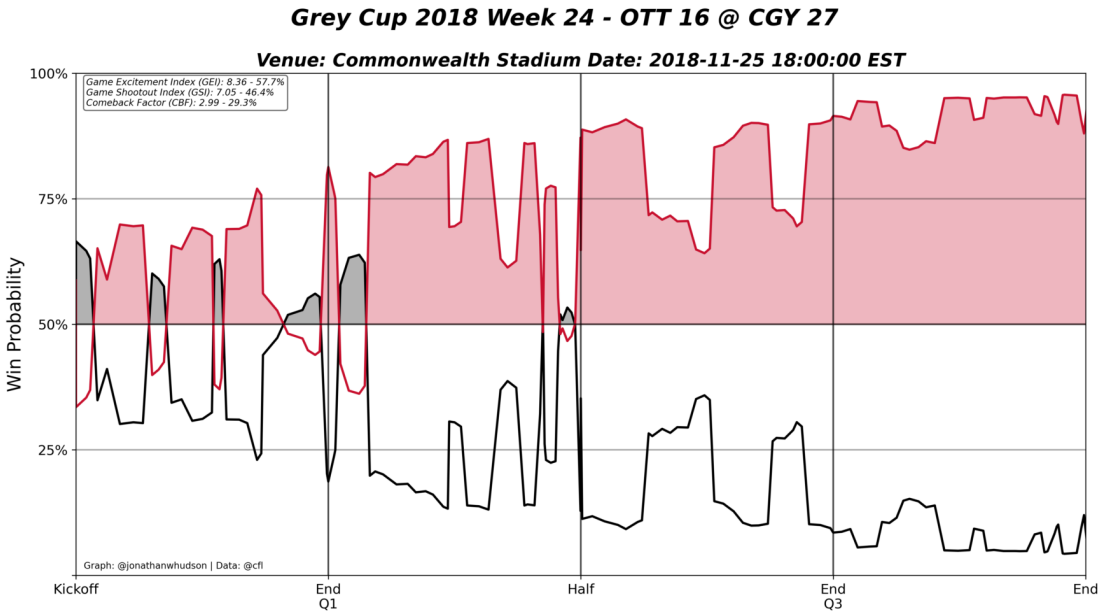


UNIVERSITY OF CALGARY

Usage Scenario

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

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

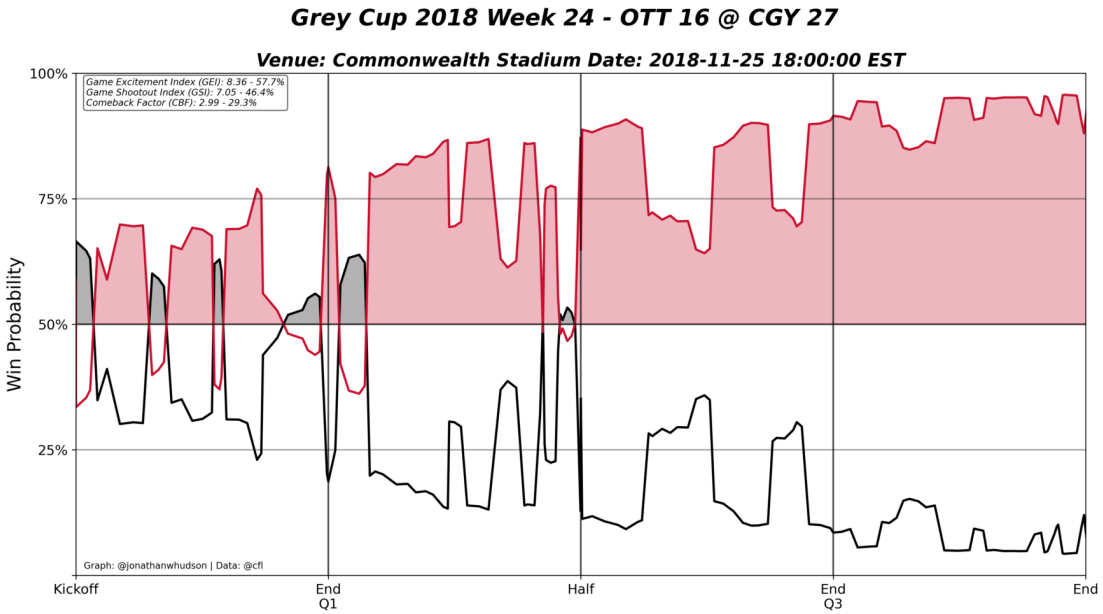


UNIVERSITY OF CALGARY

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')
```

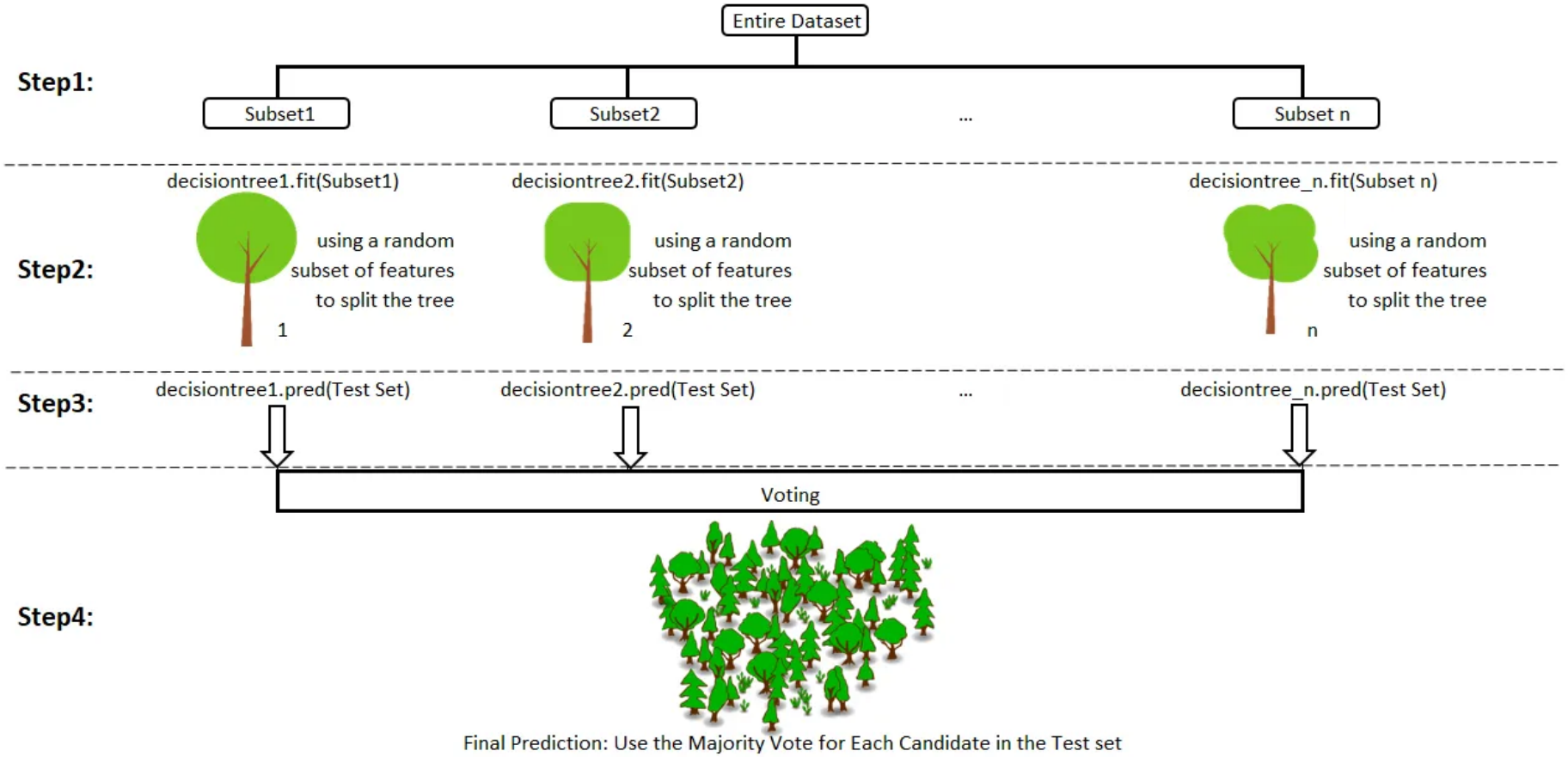


UNIVERSITY OF CALGARY

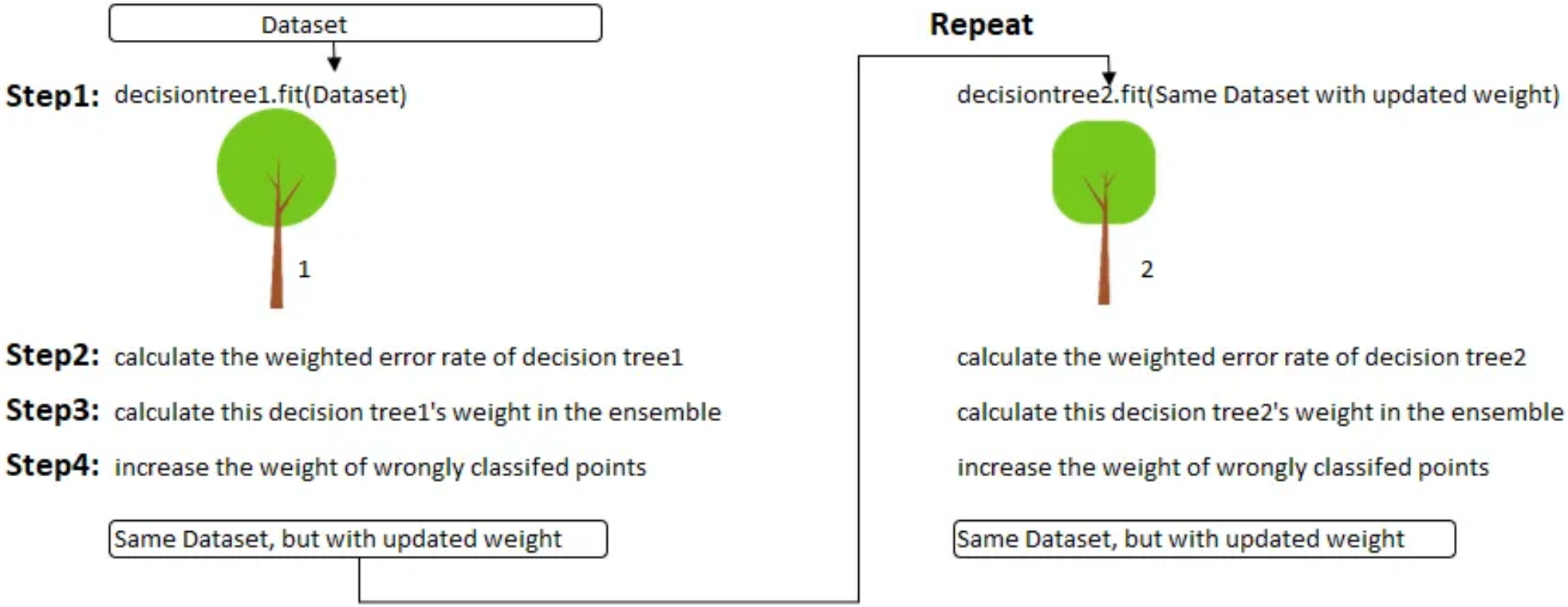
Bagging and boosting

- 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)



Onward to … neural networks

Jonathan Hudson
jwhudson@ucalgary.ca
https://pages.cpsc.ucalgary.ca/~jwhudson/

UNIVERSITY OF CALGARY