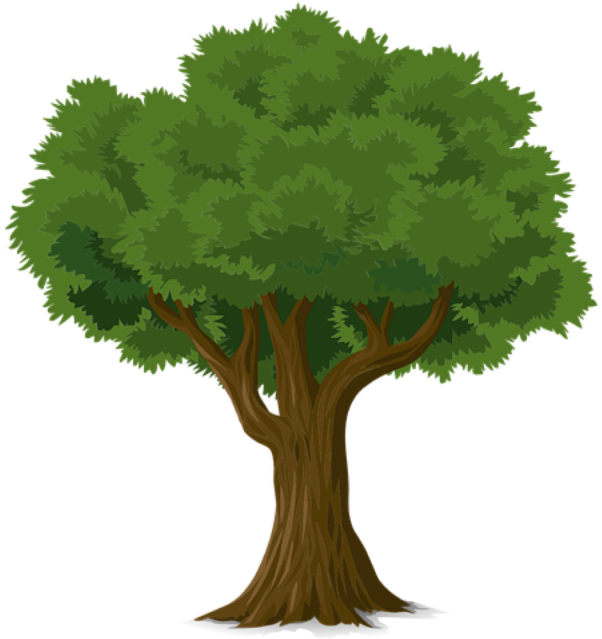
2022 Data Science Camp

Tree-Based Models



Motivation

- How does you make a decision?
  - Series of **Yes/No** questions (alternatively, True/False)
- When you decide what you want for lunch – what kinds of questions do you ask yourself?
  - Am I hungry?
  - If yes, do I want to make something?
    - If yes, make a sandwich.
    - If no, buy food.
  - If no, don't eat.
- Think of it like a (complicated) **flow chart**.
- Think of it like **alternate realities** in the **multiverse**… from a single starting point, each choice you make creates branches that lead to different outcomes
  - Different decisions lead you to different variants of the same response
  - In "Loki," different decisions lead to different timelines, which lead to different Loki **variants**

Source: https://disney.fandom.com/wiki/Variants/Gallery?file=Loki_Variants_promo_image.png

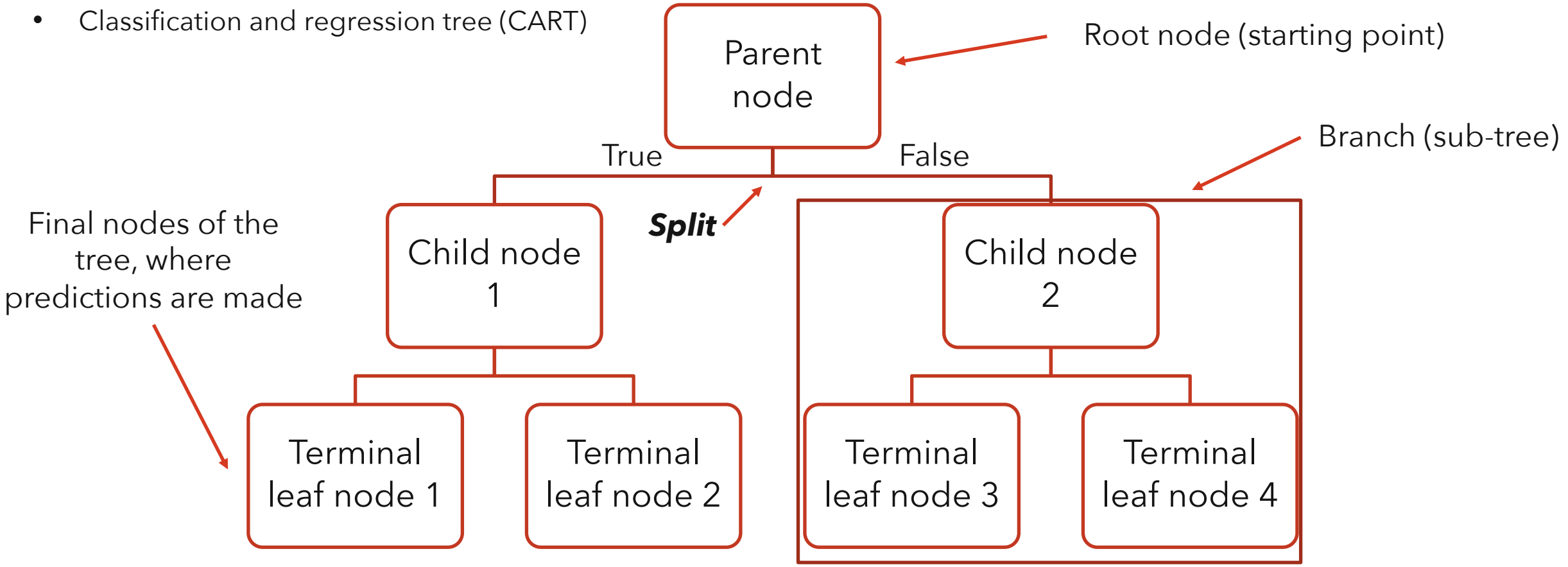
Motivation

- Can you think of situations where you've used a decision tree?
- Volunteers – describe the situation. What decision were you making?
- What questions did you ask yourself?
- What were the possible outcomes?

Decision Trees

Decision Trees: Overview

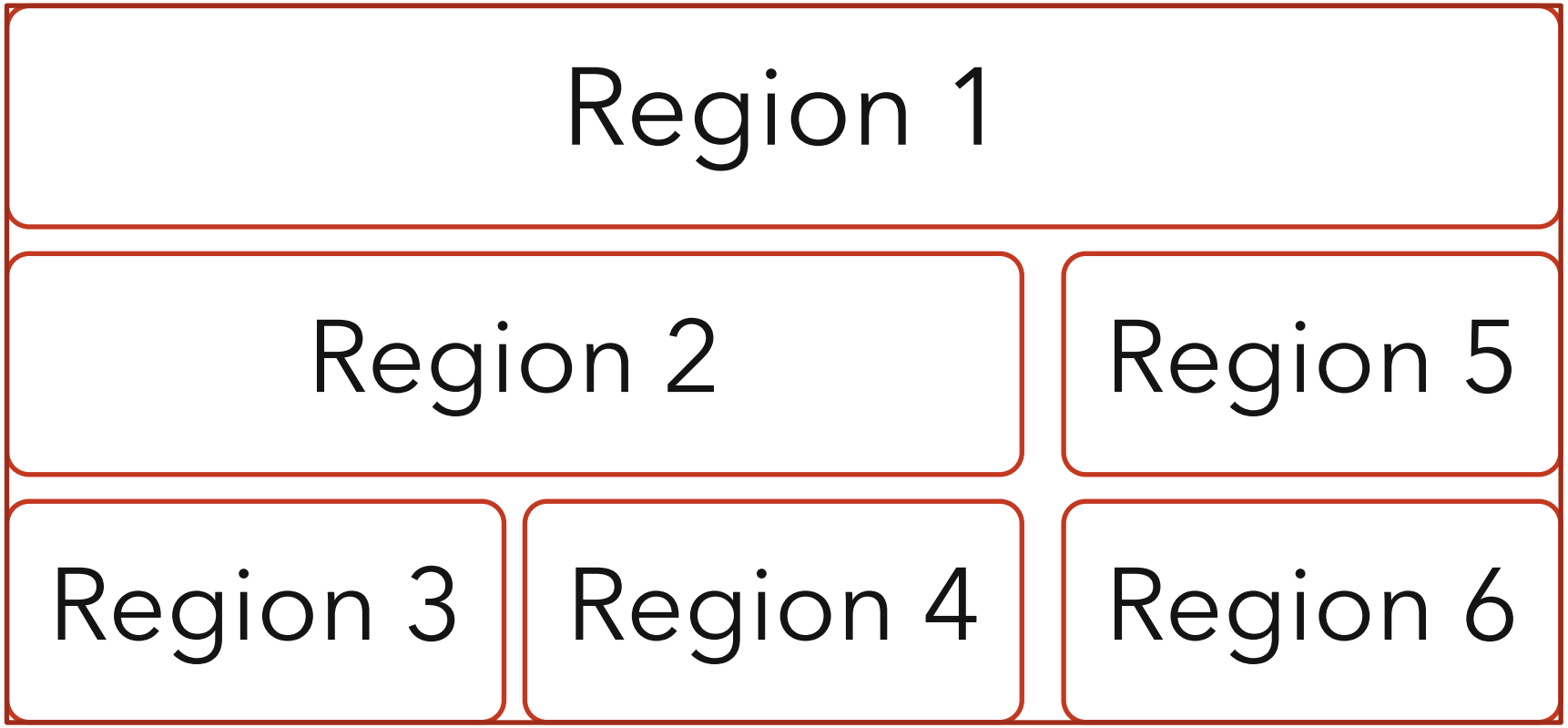
- Decision trees are **non-parametric** models (they're flexible!)
- Can either output a numerical prediction (**regression**) or a categorical prediction (**classification**)
- Classification and regression tree (CART)



Decision Trees: Alternate View

- When you create these trees, what is actually happening?
- You have your response space (the entire area that includes all possible outputs from your data)

Entire response space, split into regions from the tree



Decision Trees: Splitting

- How are the splits made?
- **GOAL**: create nodes that include homogeneous observations (i.e., most similar to one another)
- For regression, we're trying to minimize the variance of the response values
  - **Variance**: the spread of the response y around the mean value
- For classification, we're trying to group like labels with like labels
  - **Gini impurity**: how "impure" a node based on frequency of classes
    - If the node is 100% composed of class 1, then it is maximally pure.
  - **Entropy**: how much randomness is in processing the information (i.e., how uncertain the outcome is)
- Which variables do we use?
  - Use all possible variables for each split → choose the one that creates the most homogeneous nodes
  - This is known as **recursive binary splitting**, as multiple variables are considered at each split

Source: https://towardsdatascience.com/decision-trees-in-machine-learning-641b9c4e8052

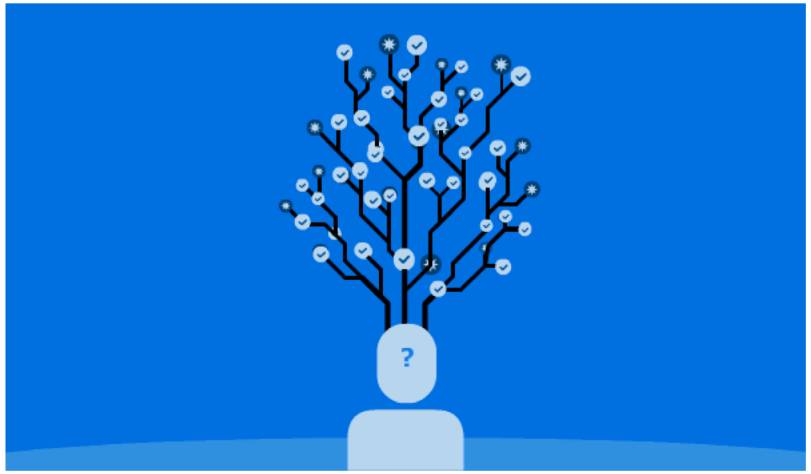
Decision Trees: Advantages

- Provides **visual interpretation** to the data
- Handles a combination of **numerical** and **categorical** data
- Defined **rules** are straightforward
- Can more or less "**dump and fit**" data (requires little preprocessing)

Decision Trees: Potential Problems?

- You've built a tree – now what?
- Your tree was built to have homogeneous nodes → what problem could this lead to?
- **POTENTIAL PROBLEM**: your tree is very deep (many levels), which makes interpretability difficult
  - **Solution**: **prune** the tree, to help reduce the depth of the tree
    - Remove nodes that aren't that helpful
    - Reduces tree complexity
  - **Solution**: control the depth of the tree
- **POTENTIAL PROBLEM**: your tree might predict well for the training data set
  - What happens if you introduce a testing data set?
  - Might not predict as well (your tree has overfit!)
- Your tree **low bias**, but has **high variance** → how do we fix this?
  - **Solution** : combine trees to create an **ensemble**

Source: https://www.kdnuggets.com/2020/01/decision-tree-algorithm-explained.html



Decision Trees: Terminology

Term	Definition
Root node	Starting point of the tree, includes all data
Internal or intermediate node	Nodes that occur in the middle of the tree
Terminal node or leaf	Final nodes of the tree, where predictions are made This node is not split
Parent node	A node that is split to create two child nodes
Child node	The sub-node of a parent node
Branch or sub-tree	A subsection or part of a
Split	The process of dividing the parent node into two child nodes
Prune	Opposite of "split," where branches or child nodes are removed

Decision Trees: Versions

- There are many ways to create decision trees, but they all follow the same general principle – create homogeneous nodes

CART

- Classification and regression trees
- Arguably the most popular algorithm

ID3

- Iterative dichotomiser 3
- Multiway tree (more than one split is possible for a single parent node)
- Requires categorical variables

C4.5

- Converts continuous variables to a set of if-then rules
- Allows for both categorical and numerical variables

CHAID

- Chi-squared automatic interaction detection
- Uses chi-squared statistics to choose best splits

Decision Trees: Code

```
from sklearn import datasets
from sklearn.ensemble import DecisionTreeClassifier, DecisionTreeRegressor


mnist = datasets.load_digits()
X = mnist.data
Y = mnist.target
clf = DecisionTreeClassifier(criterion = 'gini', max_depth = None)
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
```

Import packages

Load data

Fit the model

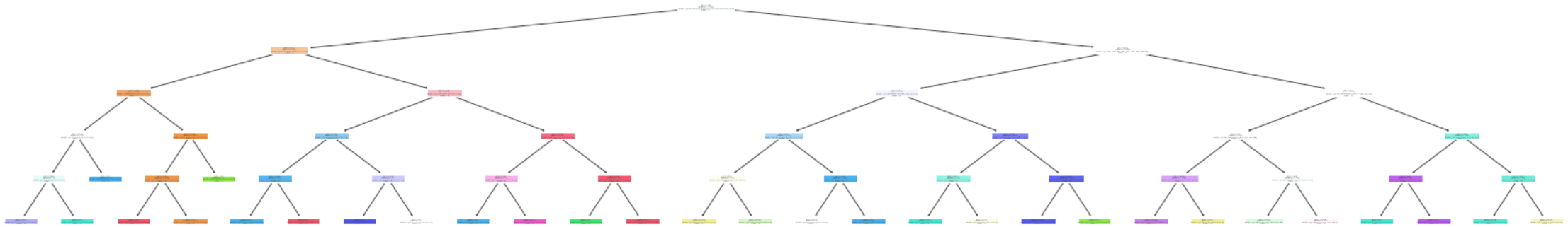
How splits are determined

How deep trees are grown

Make predictions

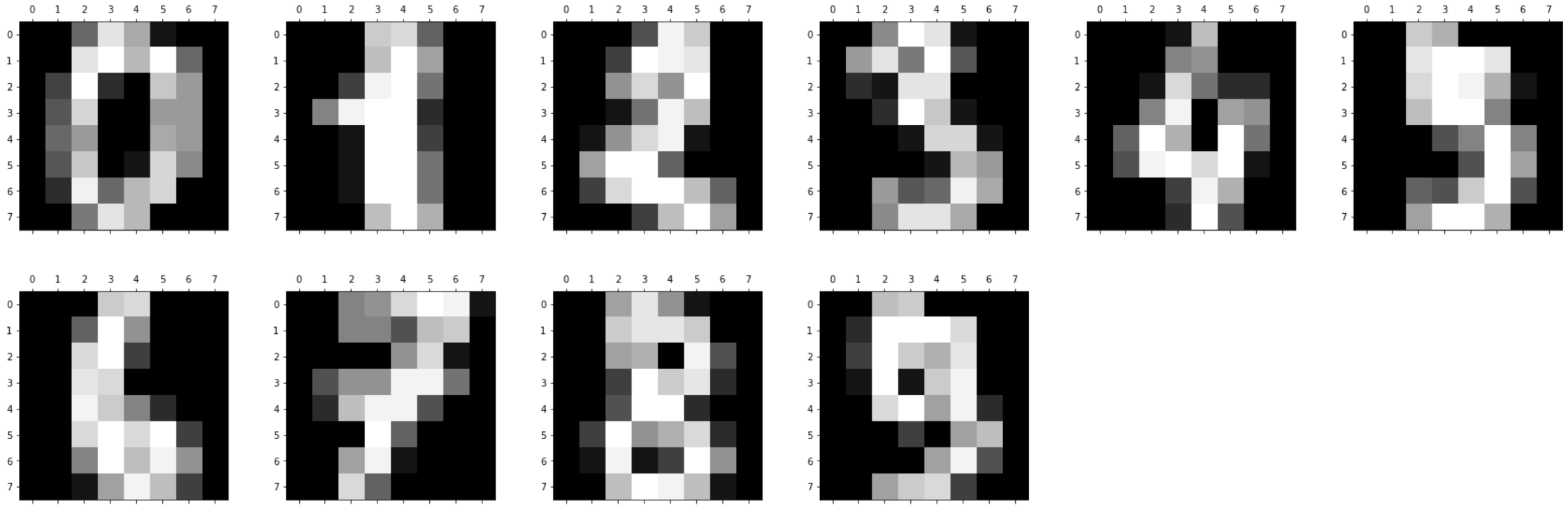
Decision Trees: Code

```
from sklearn.tree import plot_tree


plt.figure(figsize = (30, 10))

plot_tree(dt_clf, feature_names = X.columns, class_names = [str(i) for i in
y.cat.categories], filled = True)

plt.show()
```



Try fitting a decision tree yourself!

First… let's play around with the MNIST data set.



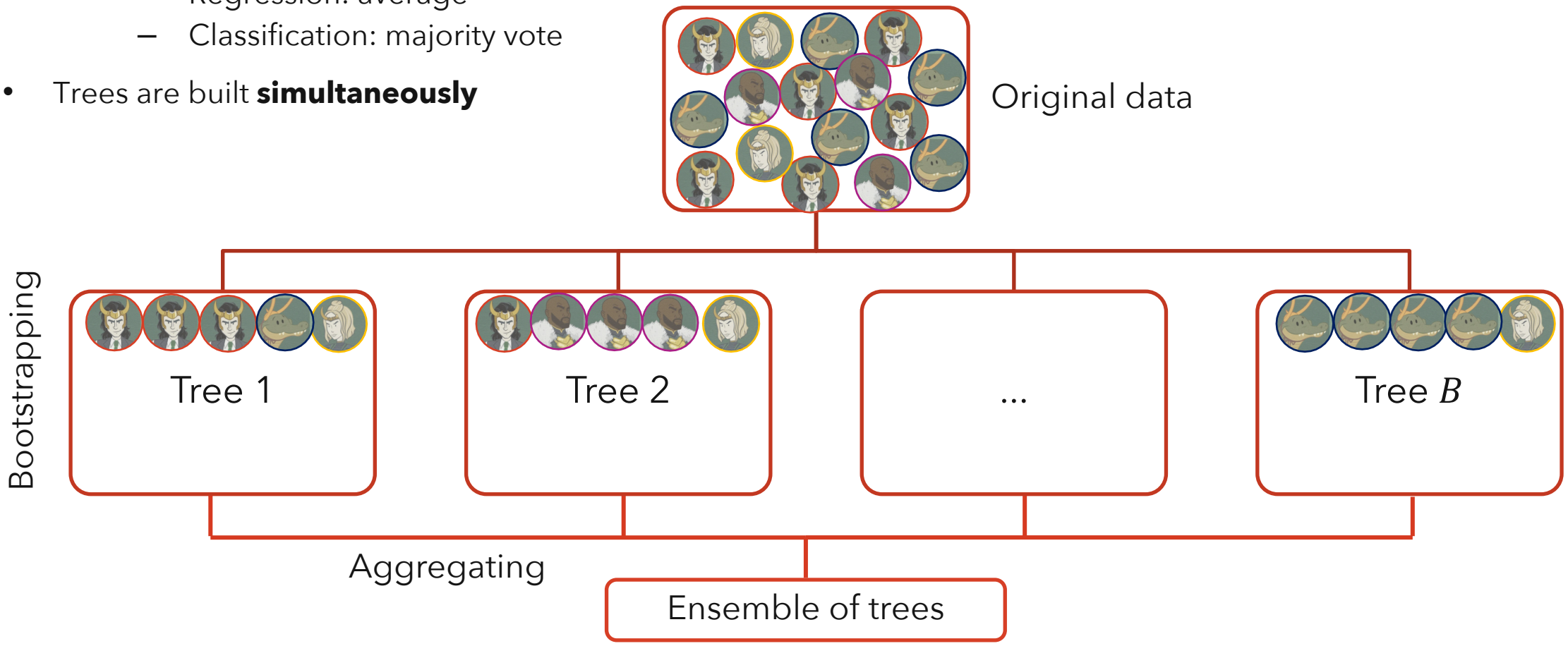
Random Forest (RF)

Random Forest: Overview

- Combine trees (**weak learners**) to create a **forest** (an **ensemble** of weak learners)
- "Random"
  - Create **different samples** of training data for each tree (row sampling)
  - Use random subsets of predictors for each split (column sampling)
- "Forest"
  - Many decision trees

Random Forest: Algorithm

- **Bagging**: bootstrap aggregation
  - Create B bootstrap (sampling with replacement) samples
  - Create B trees for each of the B samples
  - Predictions are aggregated across trees to get final output
    - Regression: average
    - Classification: majority vote
- Trees are built **simultaneously**



Random Forest: Pros and Cons

Pros

- Model is less likely to **overfit**
  - Averages across multiple trees
  - Uses subsets of all variables, so the trees are uncorrelated (not associated) with another
- Easy to use – it's a **combination** of decision trees
- Inherently provides **variable importance** (which features contribute most to the predictions)
  - Not all variables are used at every split, so the impact of leaving a variable out is easily obtained

Cons

- Since so many decision trees are fit, need a lot of **memor**y and **computation resources** – problems when the data is large
- Can be difficult to interpret – what does an average of many decision trees mean?

Random Forest: Code

```
from sklearn import datasets
from sklearn.ensemble import RandomForestClassifier, RandomForestRegressor


mnist = datasets.load_digits()
X = mnist.data
Y = mnist.target
clf = RandomForestClassifier(n_estimators = 100, criterion = 'gini', max_depth = None,
      max_features = 'sqrt')
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
```

Import packages

Load data

Fit the model

Number of trees

How splits are determined

How deep trees are grown

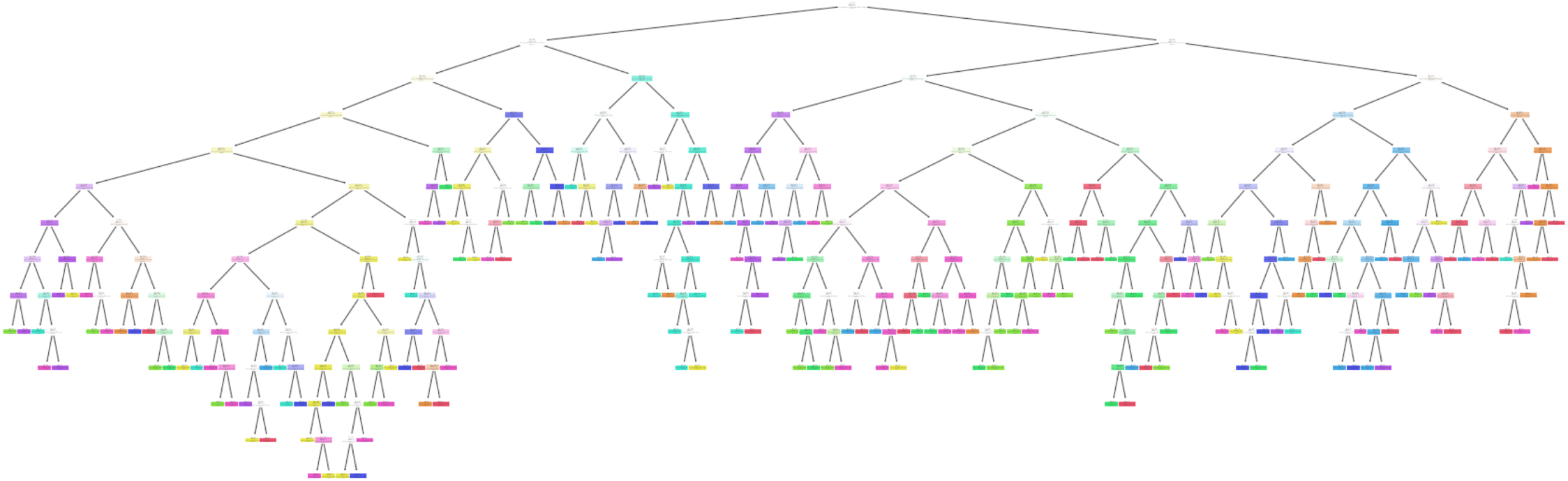
Size of feature subset used at each split

Make predictions

Random Forest: Code

```
from sklearn.tree

import plot_tree plt.figure(figsize = (30, 10))

plot_tree(clf.estimators_[1], feature_names = , filled = True)
```

Plot one of the trees in the forest



Random Forest: Code

```
from sklearn.metrics import accuracy_score
```

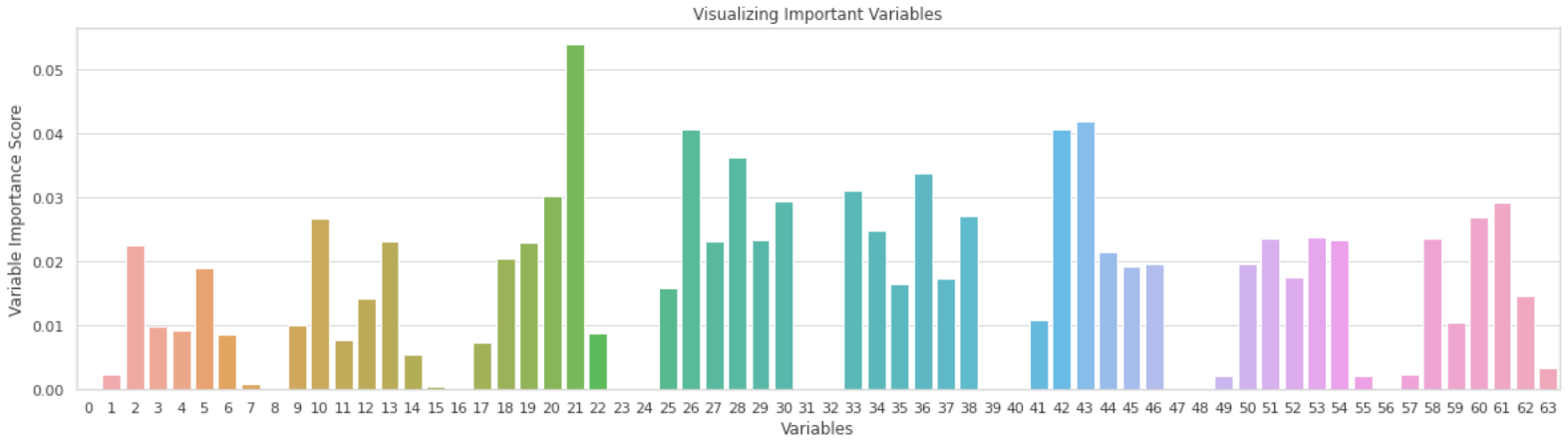
Import packages

```
print("Accuracy: ", accuracy_score(y_test, y_pred))
```

Output accuracy

```
feature_imp = pd.Series(clf.feature_importances_).sort_values(ascending = False)
```

Variable importance of the variables

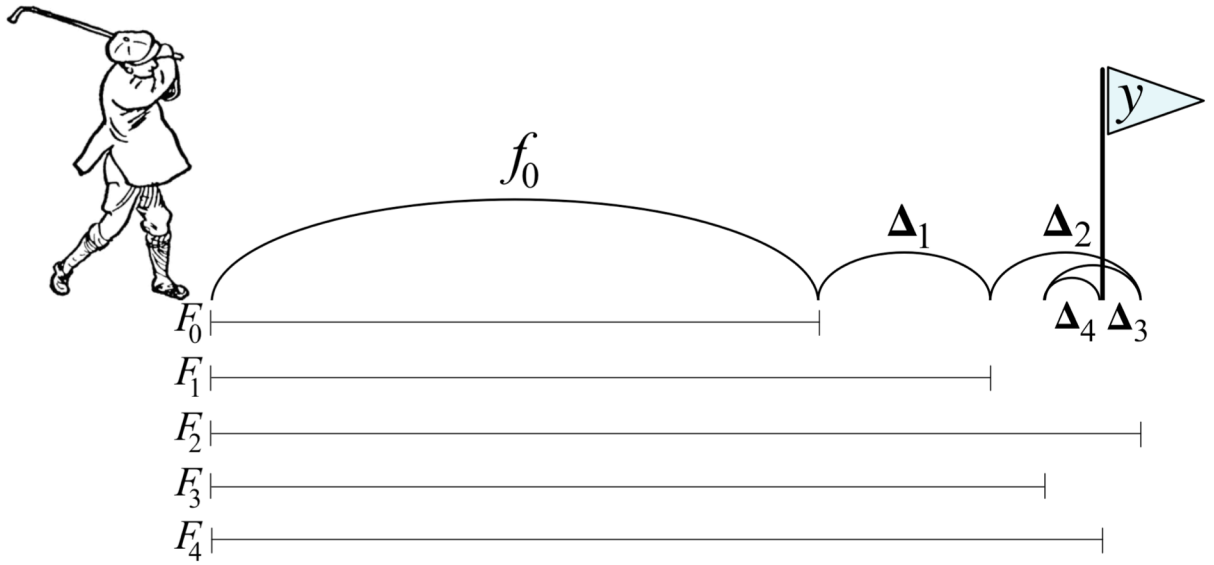


Try fitting a random forest yourself!

Gradient Boosting Machines (GBMs)

Gradient Boosting Machines: Intuition

- A golfer has a **general** idea of where the hole is
- They'll give the golf ball a whack, and (generally) it will land somewhere closer to the hole
- Each subsequent hit, the golfer will get **closer and closer** to the hole
- Each hit gives the golfer a chance to **reassess** how close they are to the hole (**distance**) and which **direction** they need to go
- Some hits may overshoot the hole, but subsequent hits will slowly get closer to the goal



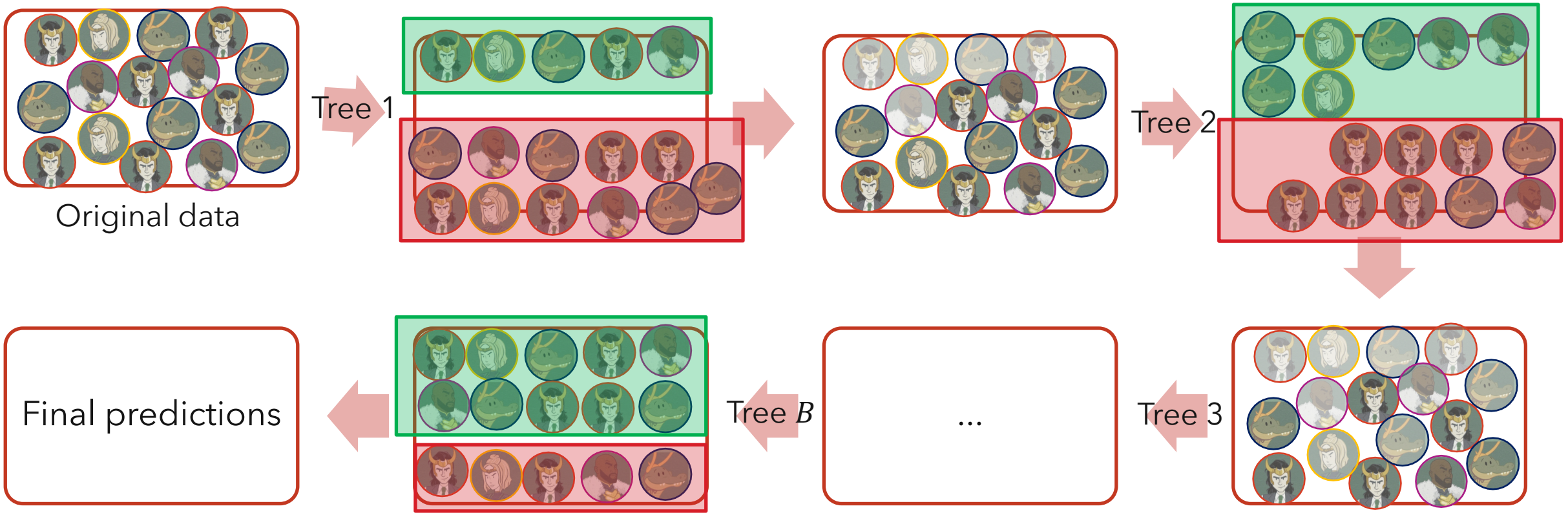
Source: https://explained.ai/gradient-boosting/L2-loss.html#sec:2.3

Gradient Boosting Machines: Overview

- Combine trees (**weak learners**) to create a sequential machine (an **ensemble** of weak learners)
- Instead of fitting trees simultaneously, we fit trees **sequentially**, improving model fit with every iteration
- "Gradient"
  - With each model, we assign difficult observations **higher weights** to improve prediction performance
  - Each sequential model learns from the **mistakes** of the previous model
  - These errors are also known as **residuals**
- "Boosting"
  - The above idea is known as "boosting"
  - We **boost** the signal of the most wrong observations
- "Machines"
  - Final prediction is a weighted sum of all "machines" in the model

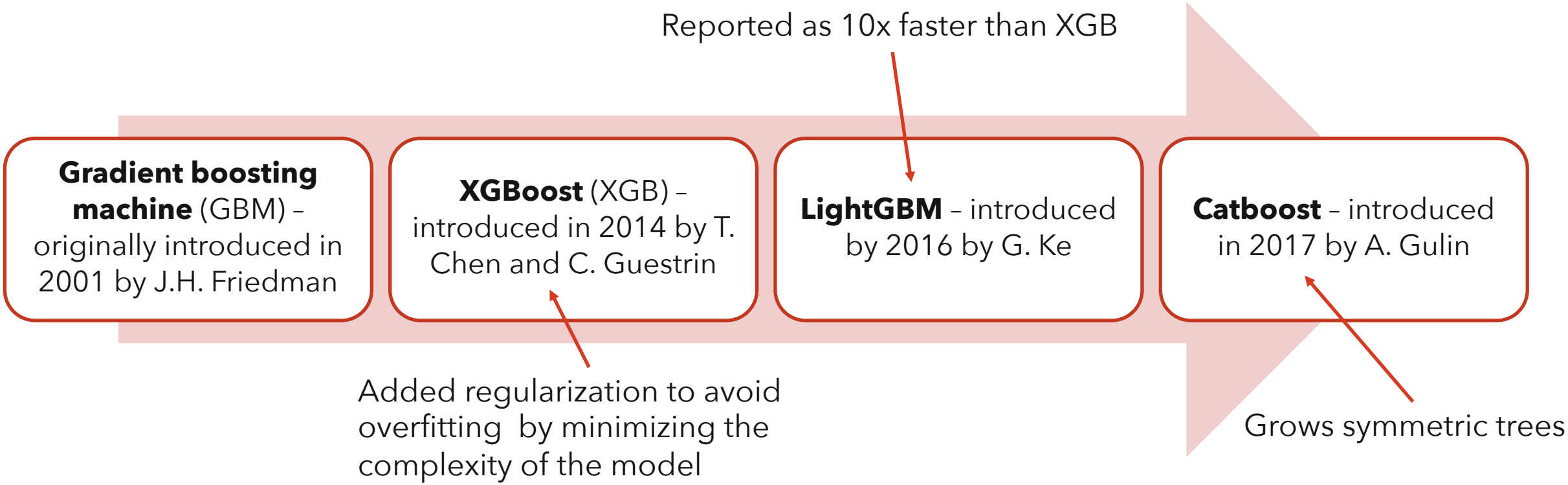
Gradient Boosting Machines: Algorithm

- Fit B **sequential**, **shallow** trees
- If a tree is fit with only one split, it's called a **stump**



Gradient Boosting Machines: Versions

- Unlike the Random Forest, there are many ways to implement GBMs
- Key differences in methods are in the algorithm (efficiency)
- Popular methods:



Gradient Boosting Machines: Code

```
from sklearn import datasets
from sklearn.ensemble import GradientBoostingClassifier, GradientBoostingRegressor


mnist = datasets.load_digits()
X = mnist.data
Y = mnist.target
clf = GradientBoostingClassifier(n_estimators = 100, criterion = 'squared_error',
      max_depth = 3, max_features = None)
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
```

Import packages

Load data

Fit the model

Number of trees

How splits are determined

Size of feature subset used at each split

How deep trees are grown

Make predictions

Try fitting a gradient boosting machine yourself!

Discussion:

Which tree-based method is the "best" in your opinion?

Why?