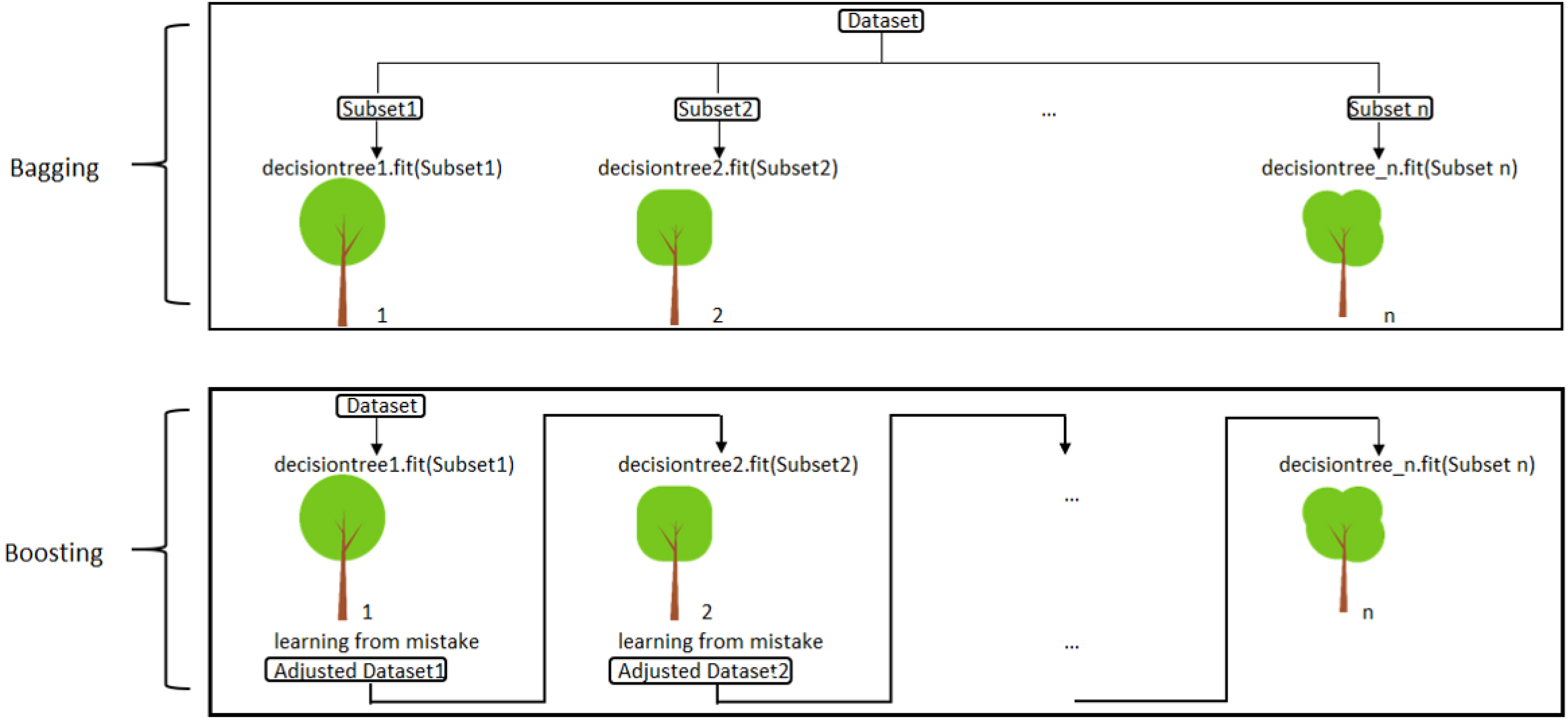
6.04 Boosting

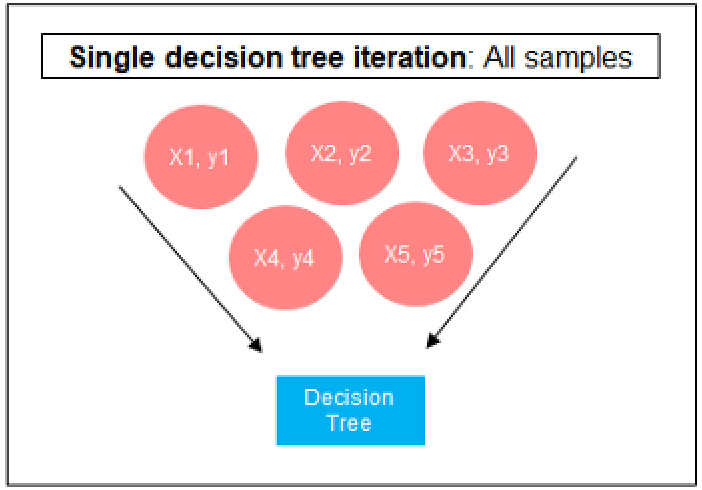
Boosting

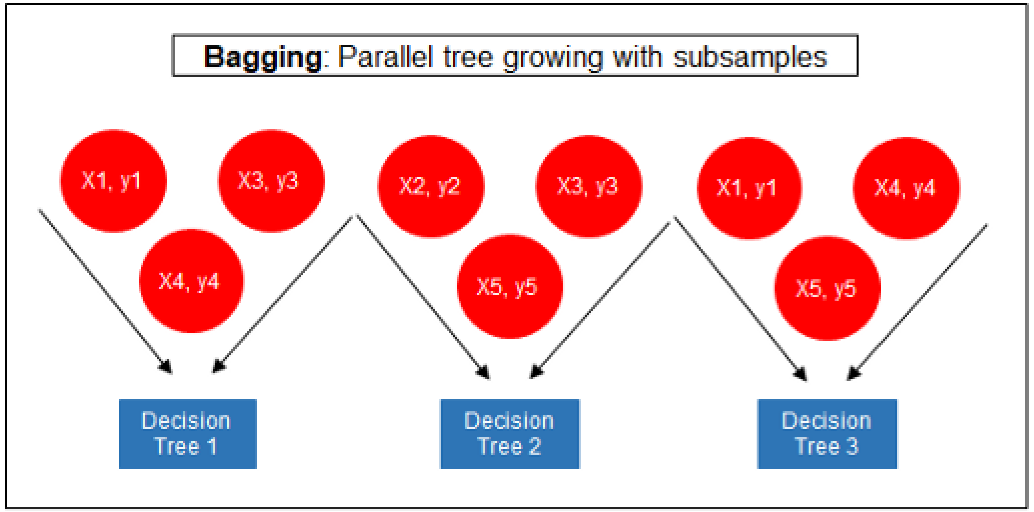
- Ensemble method that aggregates a number of weak models to create one strong model
  - A weak model is one that's only slightly better than random guessing
  - A strong model is one that's strongly correlated w/ true classification
- Boosting effectively learns from its mistakes with each iteration

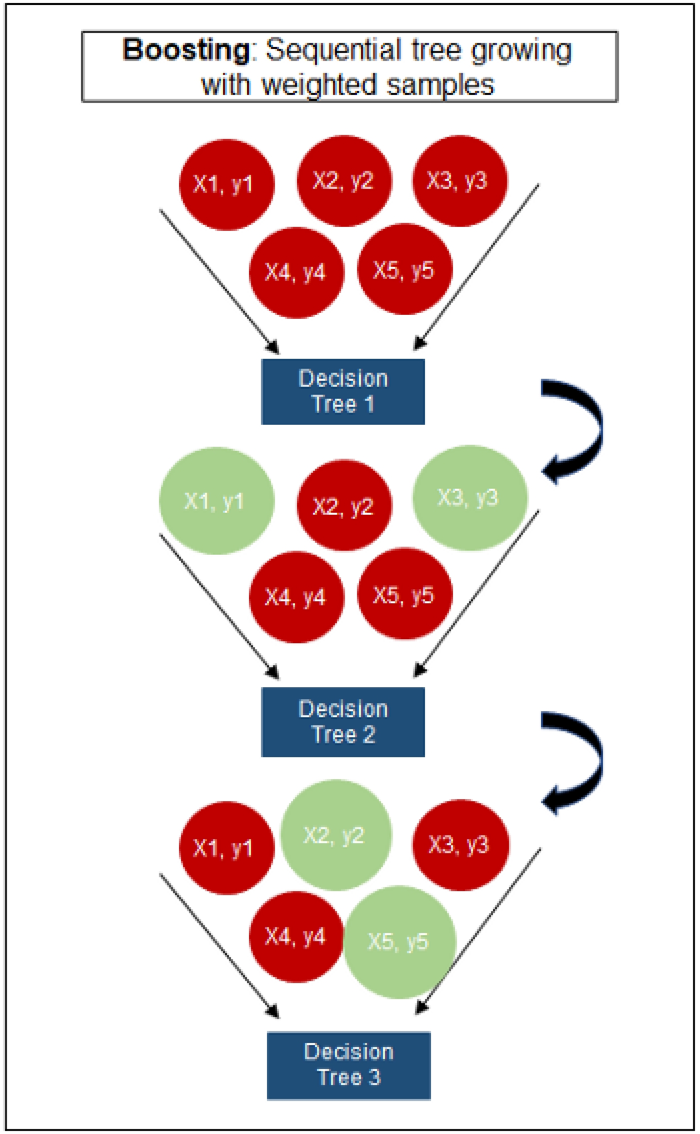
Difference between Bagging and Boosting



Difference between Bagging and Boosting







Boosting

When to Use It?

- Categorical or continuous target variable
- Useful on nearly any type of problem
- Interested in significance of predictors
- Prediction time is important

When Not to Use It?

- Transparency is important
- Training time is important or compute power is limited
- Data is really noisy

Boosting Techniques

1. AdaBoost

2. Gradient Boosting

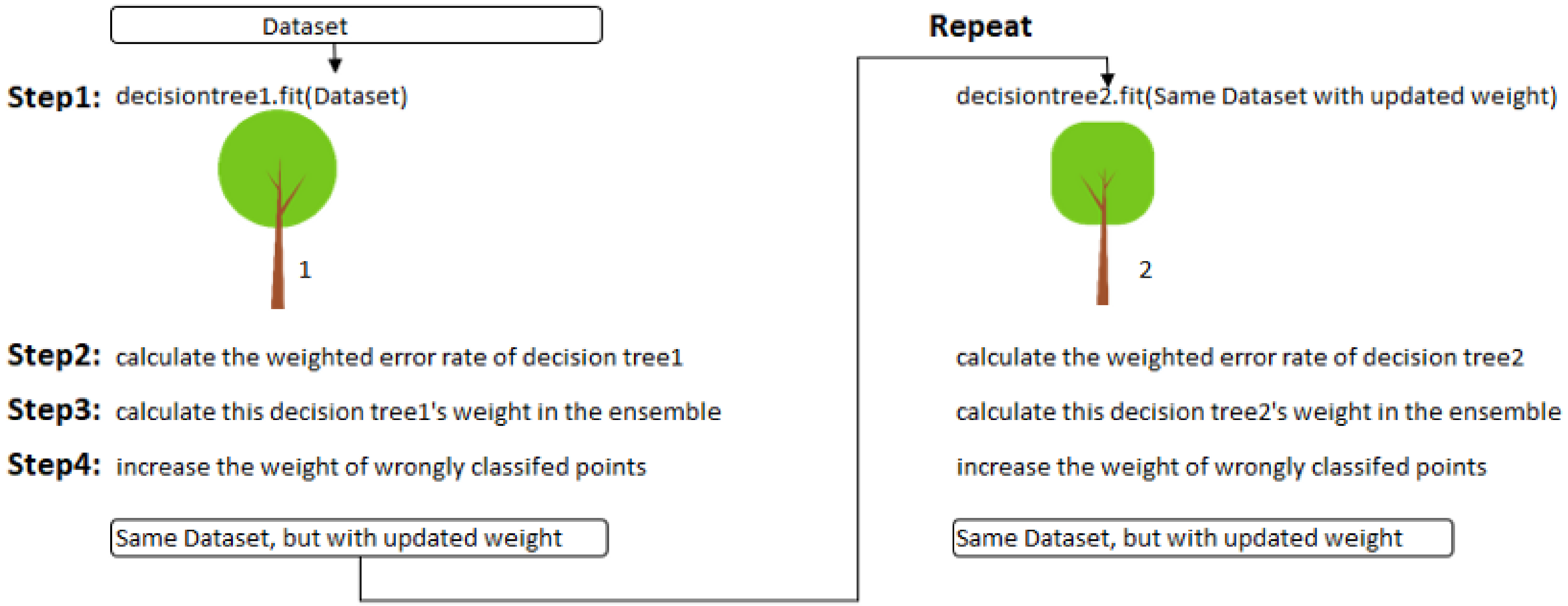
AdaBoost (Adaptive Boosting)

- Boosting ensemble model that works especially well with the decision tree.

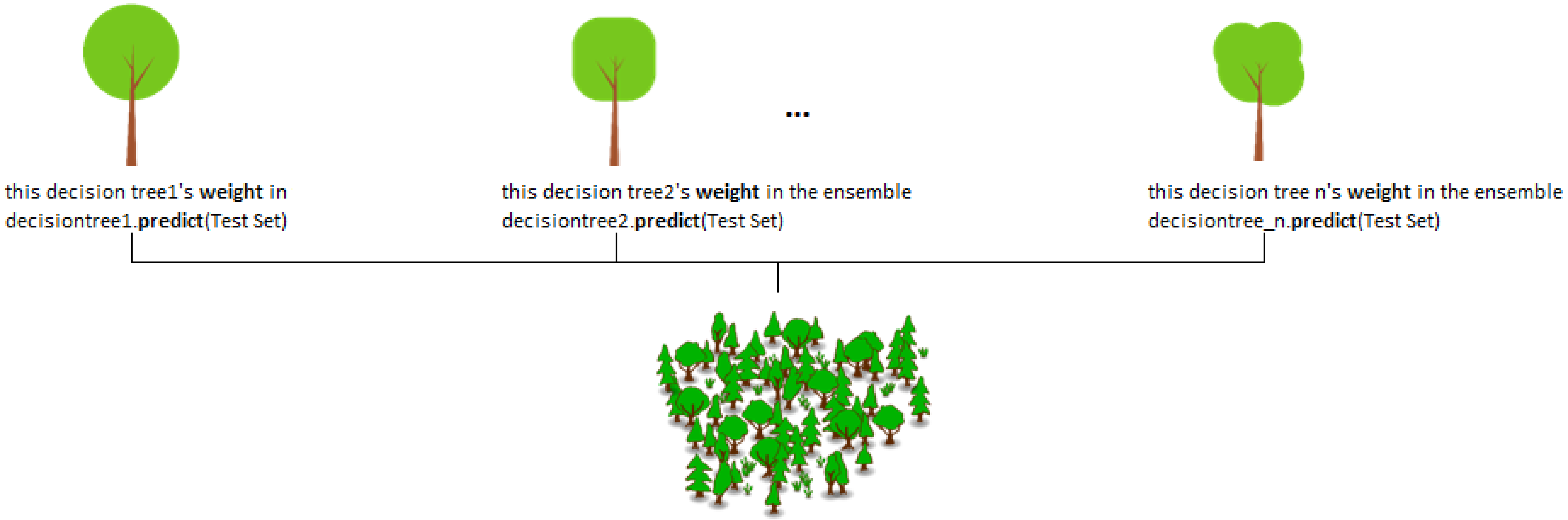
- Boosting model’s key is learning from the previous mistakes, e.g. misclassification data points.

- AdaBoost learns from the mistakes by increasing the weight of misclassified data points.

AdaBoost (Adaptive Boosting)



AdaBoost (Adaptive Boosting)



Final Prediction: **Weighted Majority** Vote for Each Candidate in the Test set

AdaBoost Process

- Step 0: Initialize the weights of data points. if the training set has 100 data points, then each point's initial weight should be 1/100 = 0.01.

- Step 1: Train a decision tree

- Step 2: Calculate the weighted error rate (e) of the decision tree. The weighted error rate (e) is just how many wrong predictions out of total and you treat the wrong predictions differently based on its data point's weight. The higher the weight, the more the corresponding error will be weighted during the calculation of the (e).

AdaBoost Process

- Step 3: **Calculate this decision tree's weight** in the ensemble
- the weight of this tree = learning rate * log((1 — e) / e)
  - the higher weighted error rate of a tree, the less decision power the tree will be given during the later voting
  - the lower weighted error rate of a tree, the higher decision power the tree will be given during the later voting

AdaBoost Process

- Step 4: **Update weights** of wrongly classified points
- the weight of each data point =
  - if the model got this data point correct, the weight stays the same
  - if the model got this data point wrong, the new weight of this point = old weight * np.exp(weight of this tree)

- Note: The higher the weight of the tree (more accurate this tree performs), the more boost (importance) the misclassified data point by this tree will get. The weights of the data points are normalized after all the misclassified points are updated.

AdaBoost Process

- Step 5: **Repeat** Step 1(until the number of trees we set to train is reached)
- Step 6: **Make the final prediction**

- The AdaBoost makes a new prediction by adding up the weight (of each tree) multiply the prediction (of each tree). Obviously, the tree with higher weight will have more power of influence the final decision.

Gradient Boosting

- Ensemble learning method that takes an iterative approach to combining weak learners to create a strong learner by focusing on mistakes of prior iterations

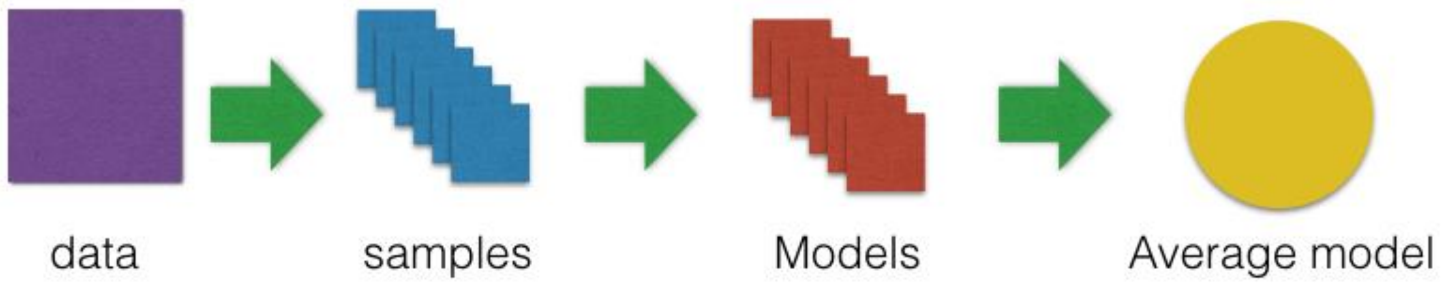
- Takes results from multiple models and combines them to get a final result

- Process overview: create subsets of the original data and run different models on the subsets; run the models sequentially

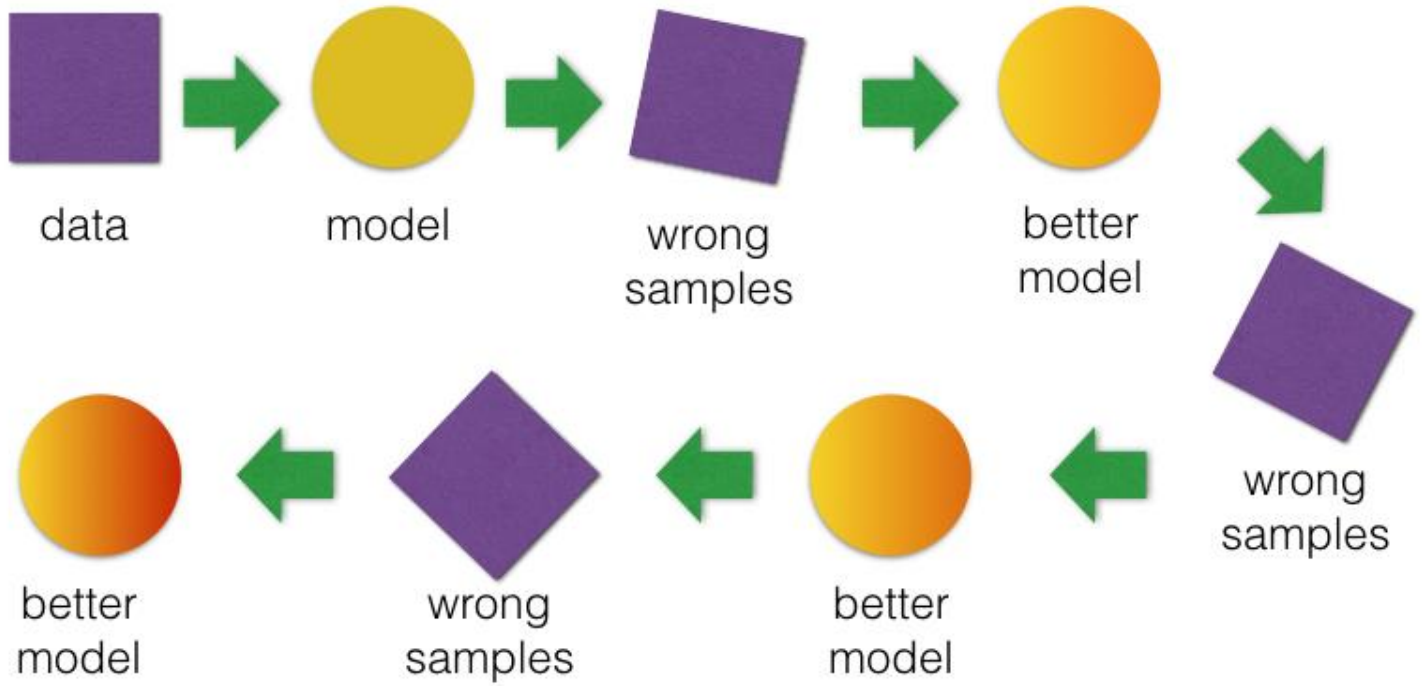
Gradient Boosting Process

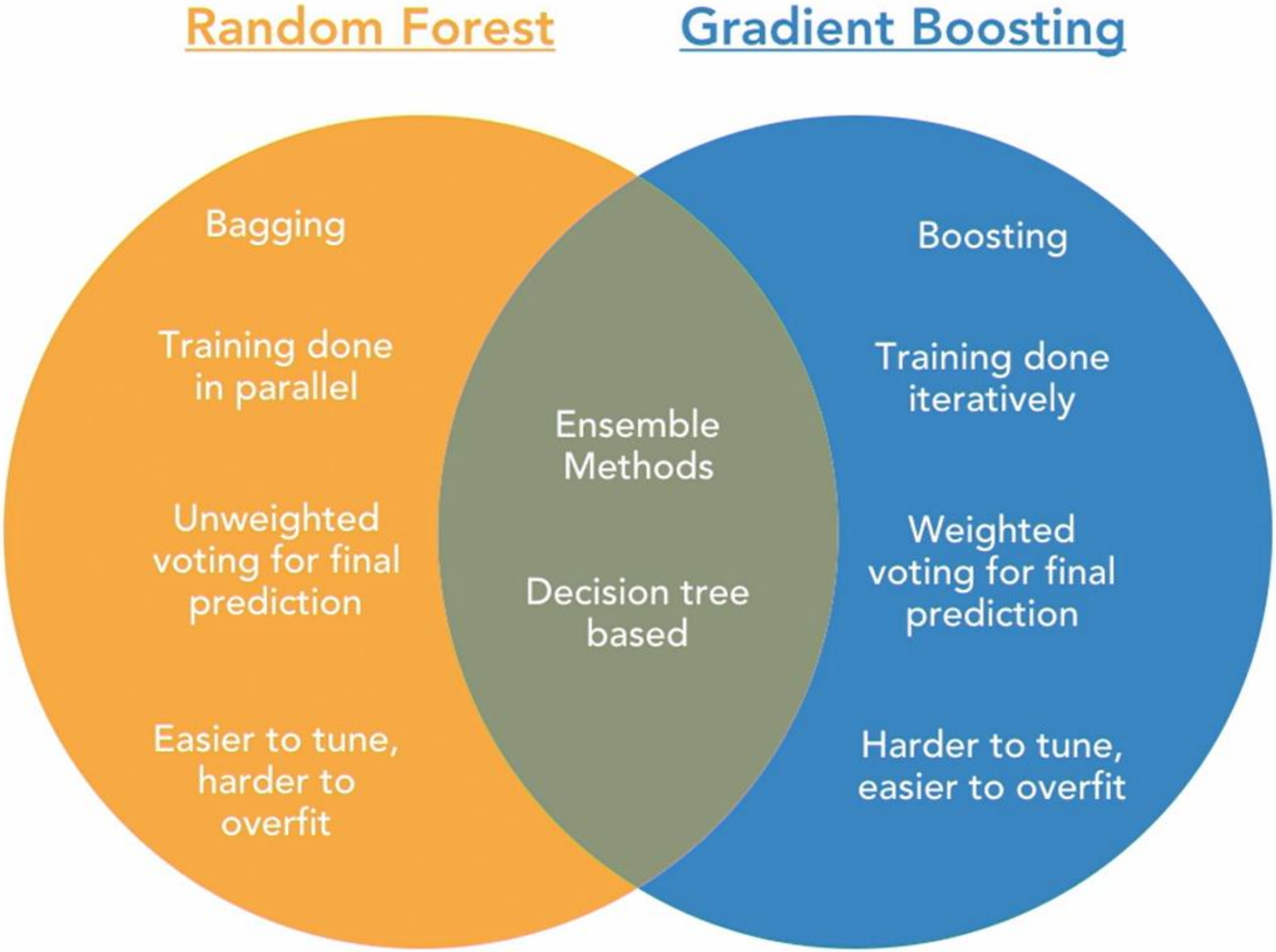
- Create a subset of data
- Run a model on subset of data and get the predictions
- Calculate errors on these predictions
- Assign weights to the incorrect predictions
- Create a better model with the same data
- Repeat cycle until a “strong learner” is created

Bagging



Boosting




Random Forest
Gradient Boosting
Bagging
Training done in parallel
Unweighted voting for final prediction
Easier to tune, harder to overfit
Ensemble Methods
Decision tree based
Boosting
Training done iteratively
Weighted voting for final prediction
Harder to tune, easier to overfit

Gradient Boosting Trade-offs

- Pros
  - Extremely powerful
  - Accepts various types of inputs
  - Can be used for classification or regression
  - Outputs feature importance
- Cons
  - Longer to train (can't parallelize)
  - More likely to overfit
  - More difficult to properly tune