Random Forests

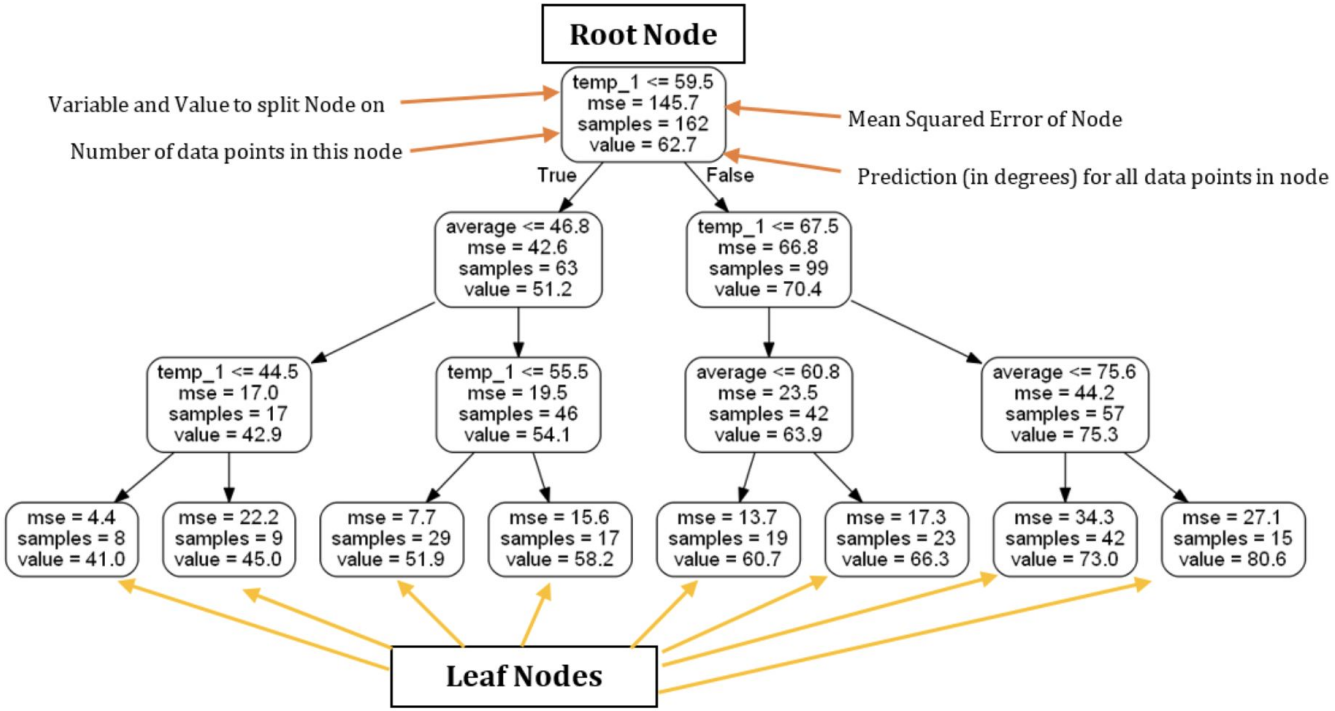
Data Science Immersive

FLATIRON SCHOOL

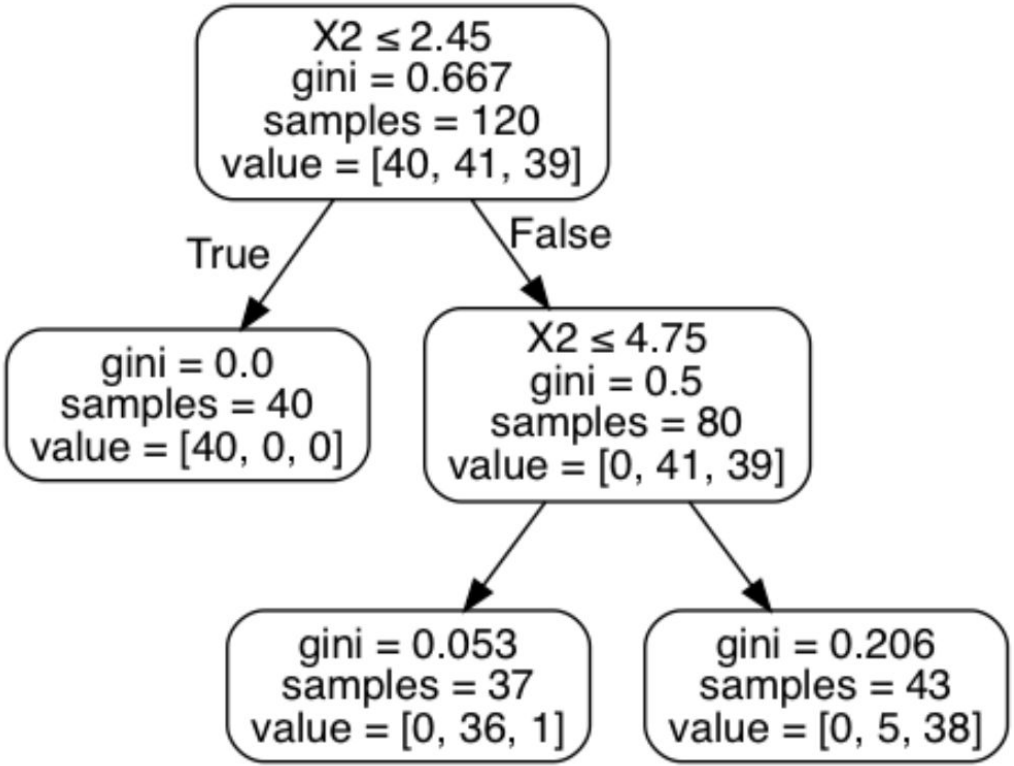
Agenda today

- Decision Trees Review
- Learn Bootstrapping and Aggregating (Bagging) to Decision Trees
- Learn the Random Forests algorithm

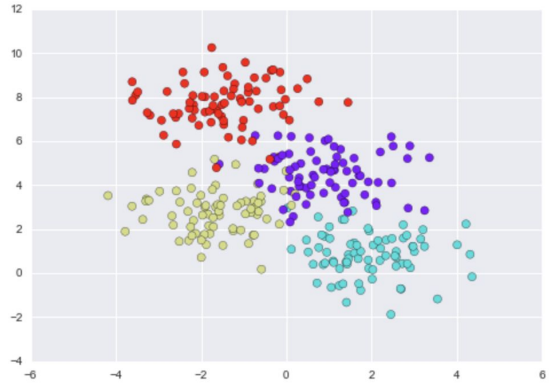
Quick Review of Regression Trees

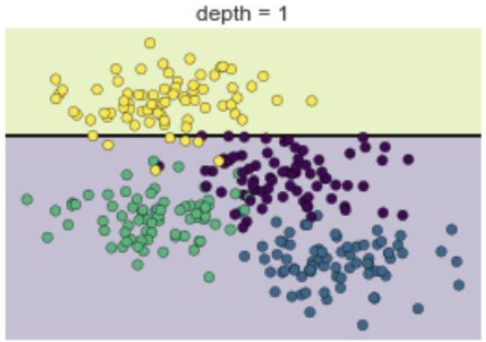


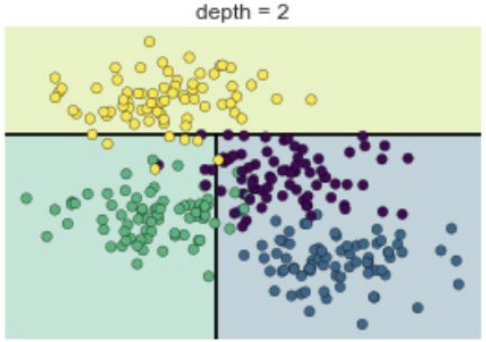
Quick Review of Classification Trees

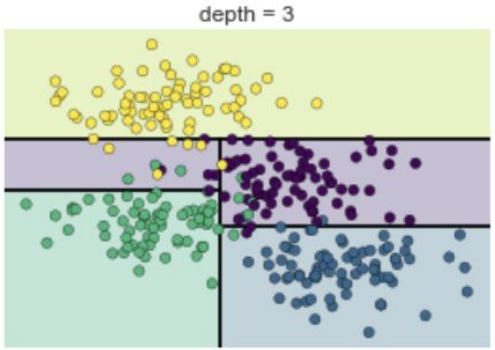


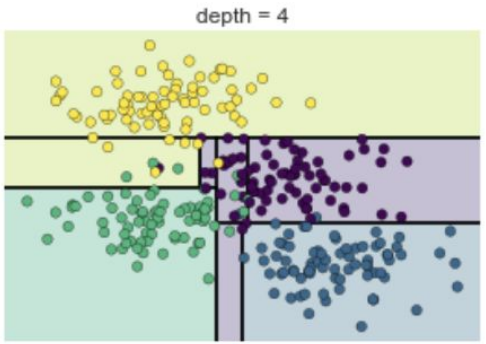
Visualizing Classification Decision Trees





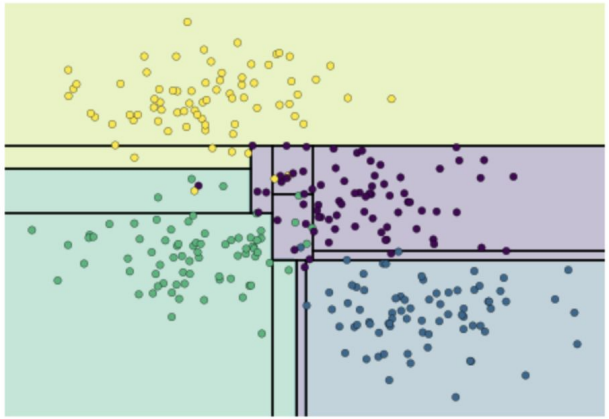


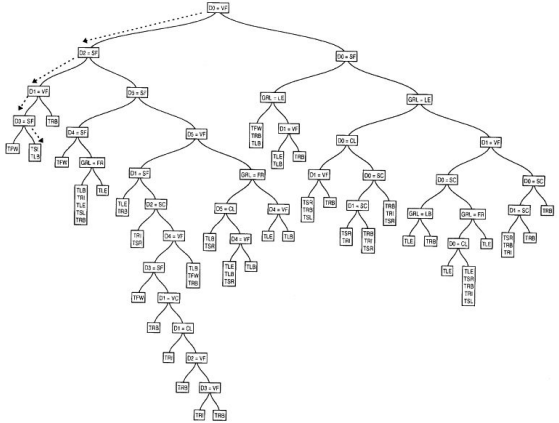


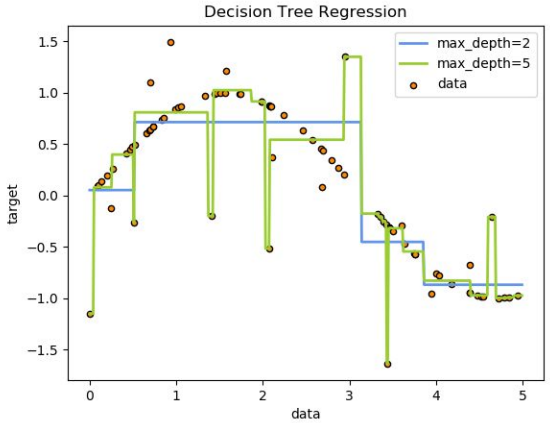


Tree Depth and Overfitting

The deeper our trees go, the higher the chance that we overfit our data







Trees are pretty great :)

Decision Tree Review

Advantages of Decision Trees:

- Work well with non-linear relationships
- Easy to interpret
- Implicit feature selection
- Account for interaction

Disadvantages of Decision Trees:

- Decision trees tend to overfit, especially if they are completely pure.
- Instability: Small changes in the input data can cause large changes to the structure of the tree.
- “Trees have one aspect that prevents them from being the ideal tool for predictive modeling, namely inaccuracy” -- Elements of Statistical Learning

Wisdom of the Crowd - expert or the crowd?

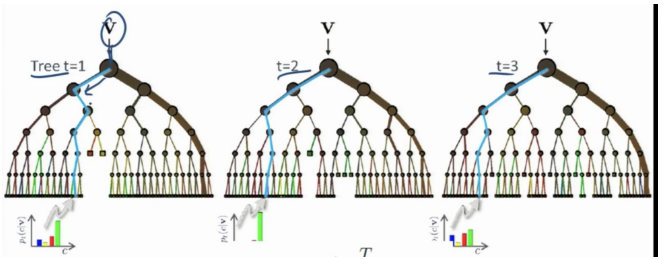
At a 1906 fair in Plymouth, England, statistician Francis Galton noticed how when 800 people guessed how much a “dressed” ox weighed. It turns out that the actual weight had only a 1% error from the median guess

Takeaway

We can learn a lot by combining many different learners. This is called an *ensemble method*.

In essence, we can combine the output of models that are not accurate enough on their own. We can get wisdom from the crowd!

How can we grow a forest of trees?



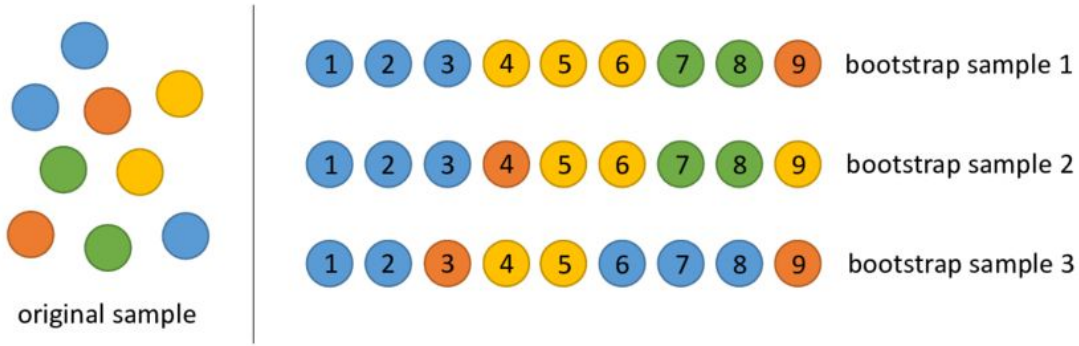
Bagged Tree: Recipe

To construct a Bagged Tree estimator, what we need is:

1) **Bootstrap** the entire dataset
2) Build a tree using a **the bootstrapped dataset**
3) Repeat step 1 and 2 many, many, many times, and **aggregate** all the trees
4) Output prediction through each tree
5) - For regression, take the average of the prediction
   - For classification, take the majority predicted value

Step 1: Bootstrapping

- Trees are prone to overfitting. (They have a high variance). This is especially true if the trees are built out to full “purity” in each of the leaves.
- To help prevent overfitting we take bootstrap samples **with replacement** from our training data that is the same size as our training data.



Step 1: Bootstrapping

Original Dataset

	Chest Pain	Good Blood Circ.	Blocked Arteries	Weight	Heart Disease
1	No	No	No	125	No
2	Yes	Yes	Yes	180	Yes
3	Yes	Yes	No	210	No
4	Yes	No	Yes	167	Yes

Bootstrapped Dataset

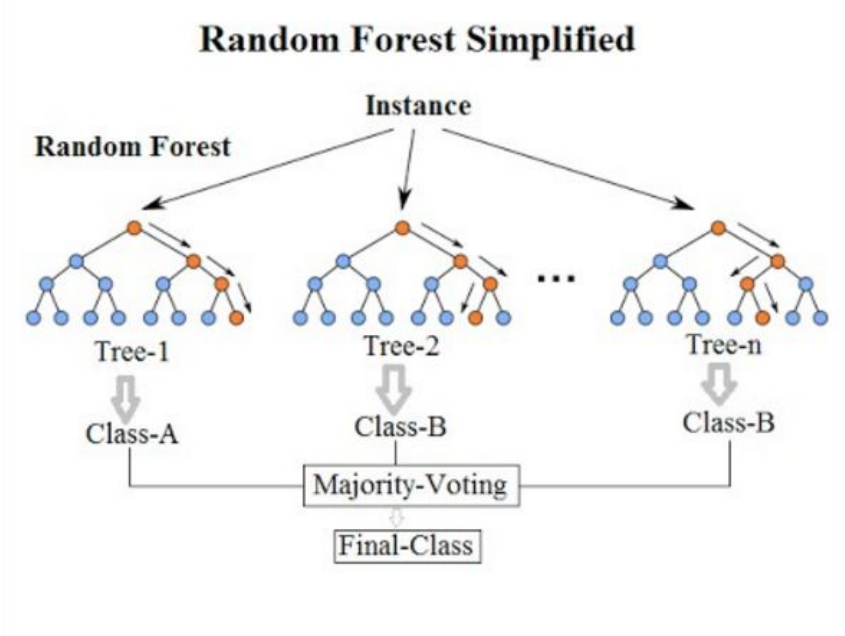
	Chest Pain	Good Blood Circ.	Blocked Arteries	Weight	Heart Disease
2	Yes	Yes	Yes	180	Yes
1	No	No	No	125	No
4	Yes	No	Yes	167	Yes
4	Yes	No	Yes	167	Yes

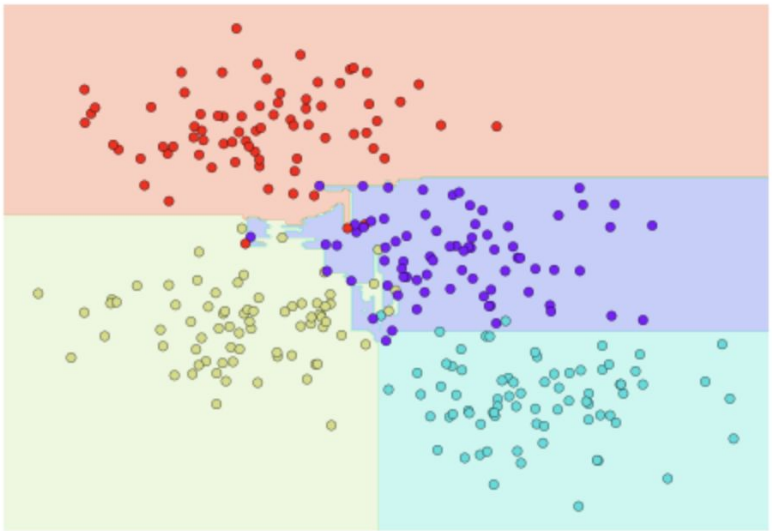
Step 2: Aggregating

- Once we have taken bootstrapped samples, we fit a decision tree to each sample. This tree will have a **low bias and high variance**
- Repeat this process for however many trees you want in your model
- Now, we can feed data through all of the bootstrapped trees and take
    - Classification: whichever class is predicted most by the bootstrapped decision trees
    - Regression: take an average of the predicted values for each decision tree

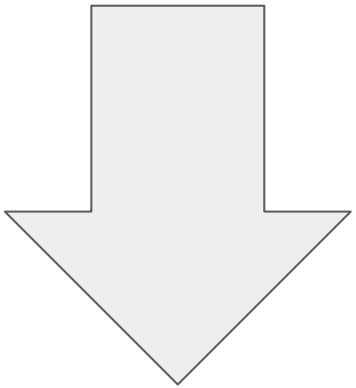
Voting Method

Majority Class Labels (Majority/Hard Voting)



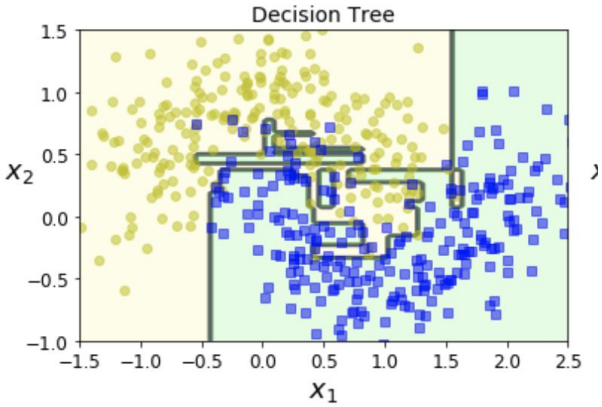


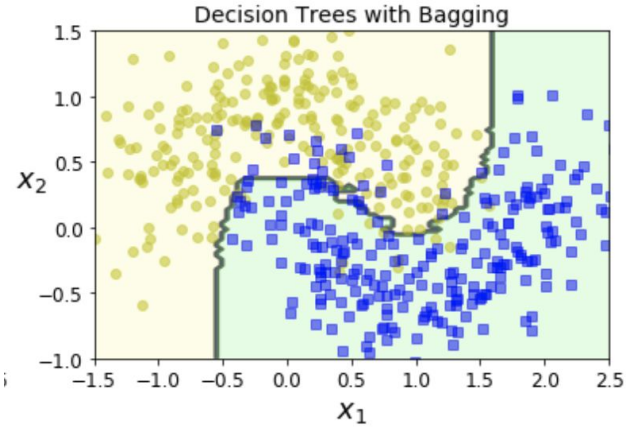
Bootstrapping + **Agg**regating



Bagging

Decision Tree v. Decision Trees with Bagging



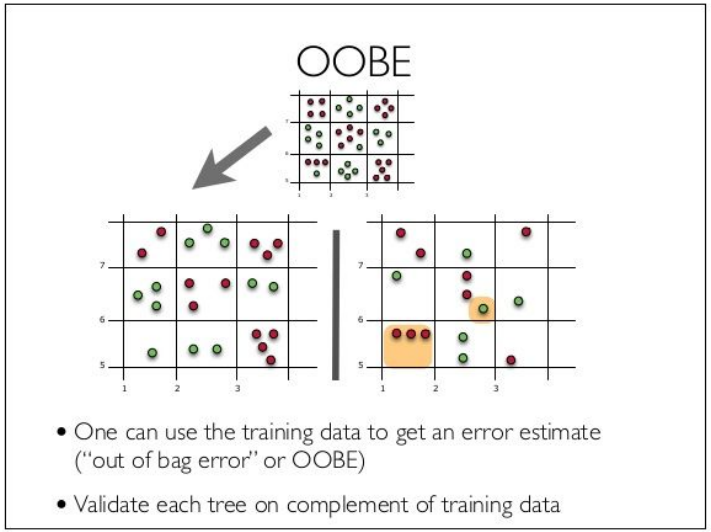


Out-of-Bag Error

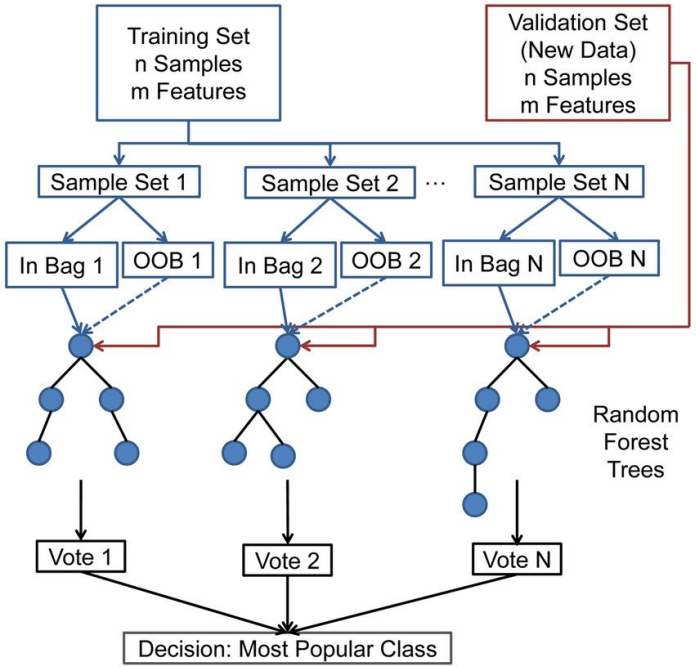
In general only ⅔ of the observations will be trained on each model. (Why 2/3?)

Performing cross validation on bagged classifiers/regressors is challenging because it can be computationally expensive.

Rather, we can look at every observation and make a prediction for each of the data points for which that data points was not used to create the tree.



Out-of-Bag Error



It's essentially just a form of cross-validation!

The Issue with Bagging

The issue with bagging is that each one of the trees might be correlated to each other. There might be a feature that is powerful in generating a separation between different categories, which results in trees that are correlated to one another despite being from bootstrapped samples.

We need to do something to ensure that the bootstrapped samples are not correlated with one another…..

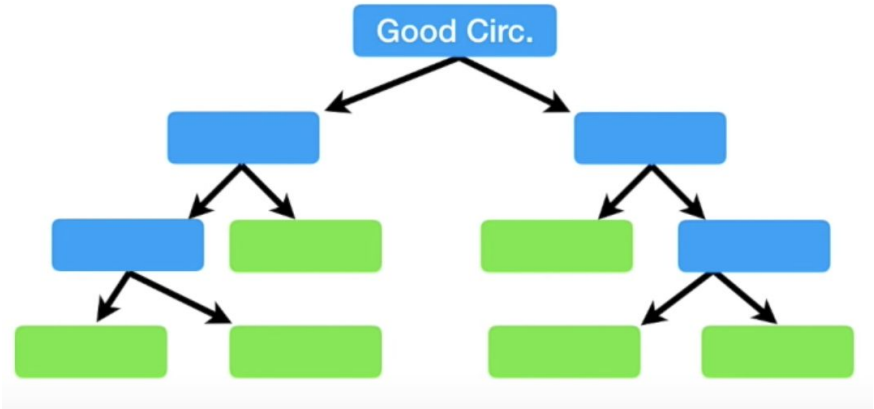
Random Forest: Recipe

To construct a Random Forest estimator, what we need is:

1) **Bootstrap** the entire dataset
2) Build a tree using only a **random subset of the features at each node** from bootstrapped dataset
3) Repeat step 1 and 2 many, many, many times, and **aggregate** all the trees
4) Output prediction through each tree
5) - For regression, take the average of the prediction
   - For classification, take the majority predicted value

Random Forest: Recipe

Step 2--selecting a subset of the feature at each node

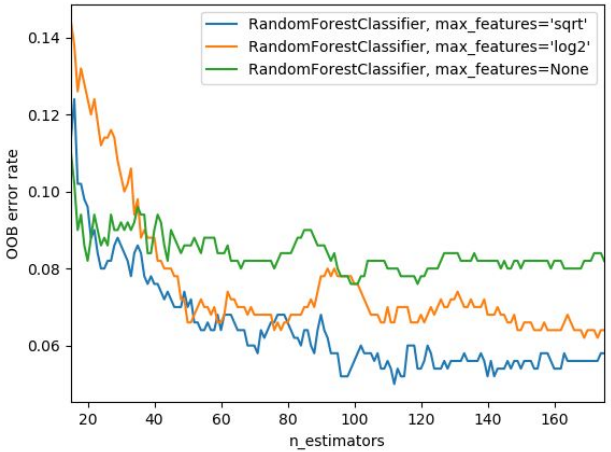


Bootstrapped Dataset

Chest Pain	Good Blood Circ.	Blocked Arteries	Weight	Heart Disease
Yes	Yes	Yes	180	Yes
No	No	No	125	No
Yes	No	Yes	167	Yes
Yes	No	Yes	167	Yes

Random Forests

Random Forests de-correlate each of the decision trees created in bagging by ensuring that at each split, only *m* features are considered for a given split. (typically *m* = sqrt(p))



This means that on average *(p-m)/p* splits will not even consider a given strong predictor. As we increase the number of trees, this will not lead to overfitting, meaning that we should make as many trees as possible until we have achieved an acceptable error rate.

Random Subspace Sampling Method

Important note: The *m* features are randomly chosen at each ***node*** not for the entire decision tree.

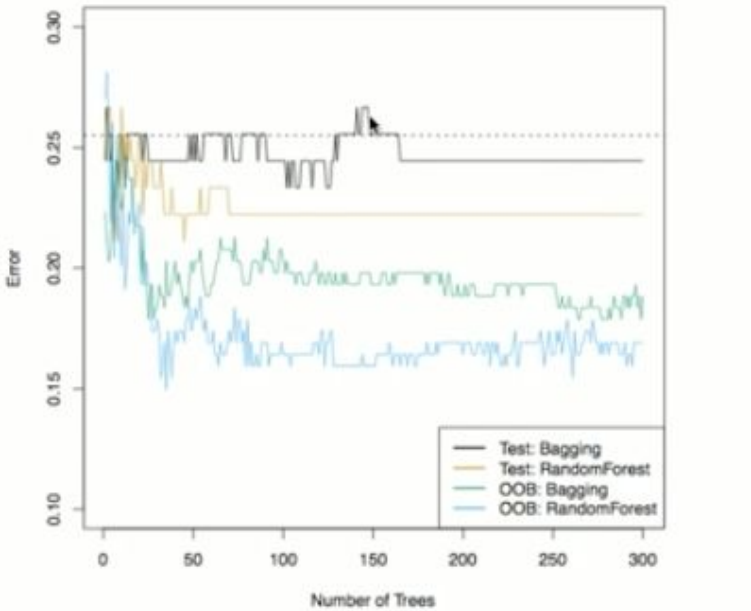
This is to ensure that our trees need to be decorrelated with one another

Our trees should have ***diverse*** opinions

Comparing Performances

In general:

Random Forest > Bagging > Decision Trees



Random Forest Advantages/Disadvantages

Advantages

- A very powerful model. Will nearly always outperform decision trees
- Able to detect non-linear relationships well
- Harder than other models to overfit

Disadvantages

- Not as interpretable as decision trees
- Many hyperparameters to tune (GridSearch is your friend!)

Random Forest Hyperparameters

n_estimators : the number of trees in the forest

criterion: “gini”,”entropy”

max_features: the number of random features to be considered when looking for the best split

max_depth: the maximum number of levels of a tree

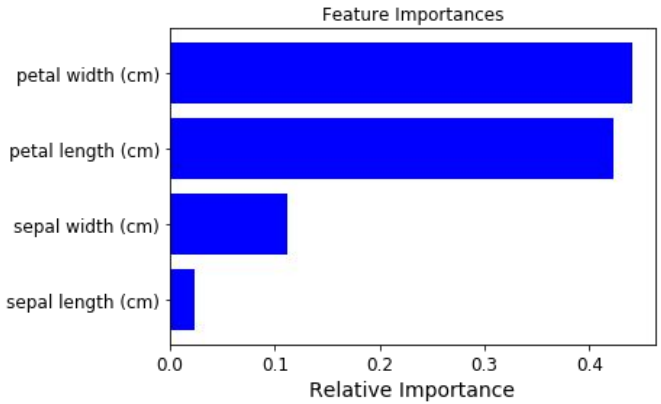
bootstrap: whether or not bootstrap samples are used to build trees

oob_score: whether or not to use out-of-bag samples to estimate the generalization accuracy

n_jobs: how many cores you want to use when training your trees

Feature Importances

The more the accuracy of the random forest decreases due to the exclusion (or permutation) of a single variable, the more important that variable is, and therefore variables with a large mean decrease in accuracy are more important for classification of the data.



There are many other ways to determine the feature importances of Random Forest. Check them out here https://papers.nips.cc/paper/4928-understanding-variable-importances-in-forests-of-randomized-trees.pdf

On another note

In sklearn, you can create custom ensemble models by making use of the VotingClassifier/VotingRegressor. This is intended to be used with conceptually different models.

- Within the VotingClassifier, you can specify “Majority Class” vote or “Soft” vote.
  - Majority Class: Select the class that is predicted most
  - Soft: Take an average of the probabilities for each class, make decision based off of that

Question to ponder…….

How do Random Forests handle the bias-variance tradeoff? What would be another way of using ensembling methods to tackle the bias-variance tradeoff?

Additional Resources

https://www.stat.berkeley.edu/~breiman/randomforest2001.pdf

https://www.stat.berkeley.edu/~breiman/RandomForests/cc_home.htm

Boosted Algorithms

Data Science Immersive

Lesson Overview

Aim: differentiate between bagging and boosting ensemble methods, as well as explain how a two specific boosting methods (AdaBoost and Gradient Boosting) differ, and how they each make predictions.

Agenda:

- Review Ensemble Methods
- Learn about the AdaBoost method
- Learn about the Gradient Boosting

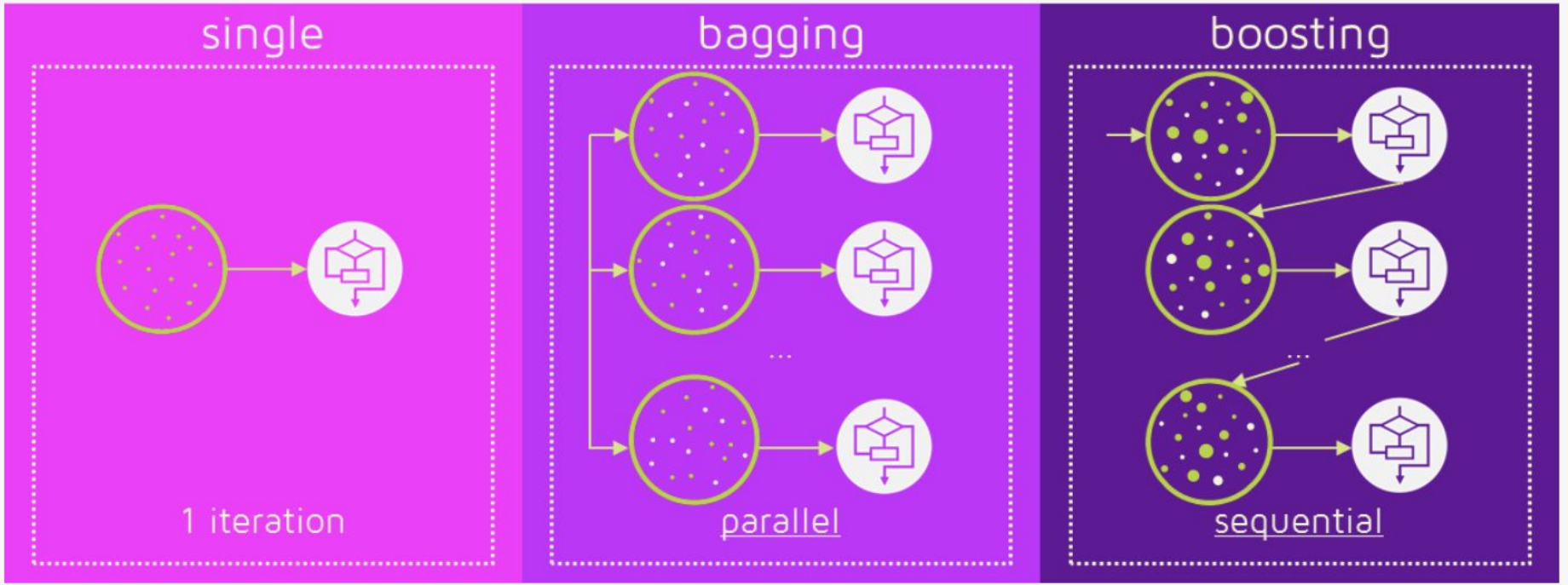
Boosting

Boosting (originally called hypothesis boosting) refers to any ensemble method that can combine several weak learners and their mistakes into a strong learner.
What is a weak learner?

The general idea of most boosting methods is to train predictors sequentially, each trying to correct its predecessor.

There are many boosting methods available, but by far the most popular are AdaBoost(short for Adaptive Boosting) and Gradient Boosting.

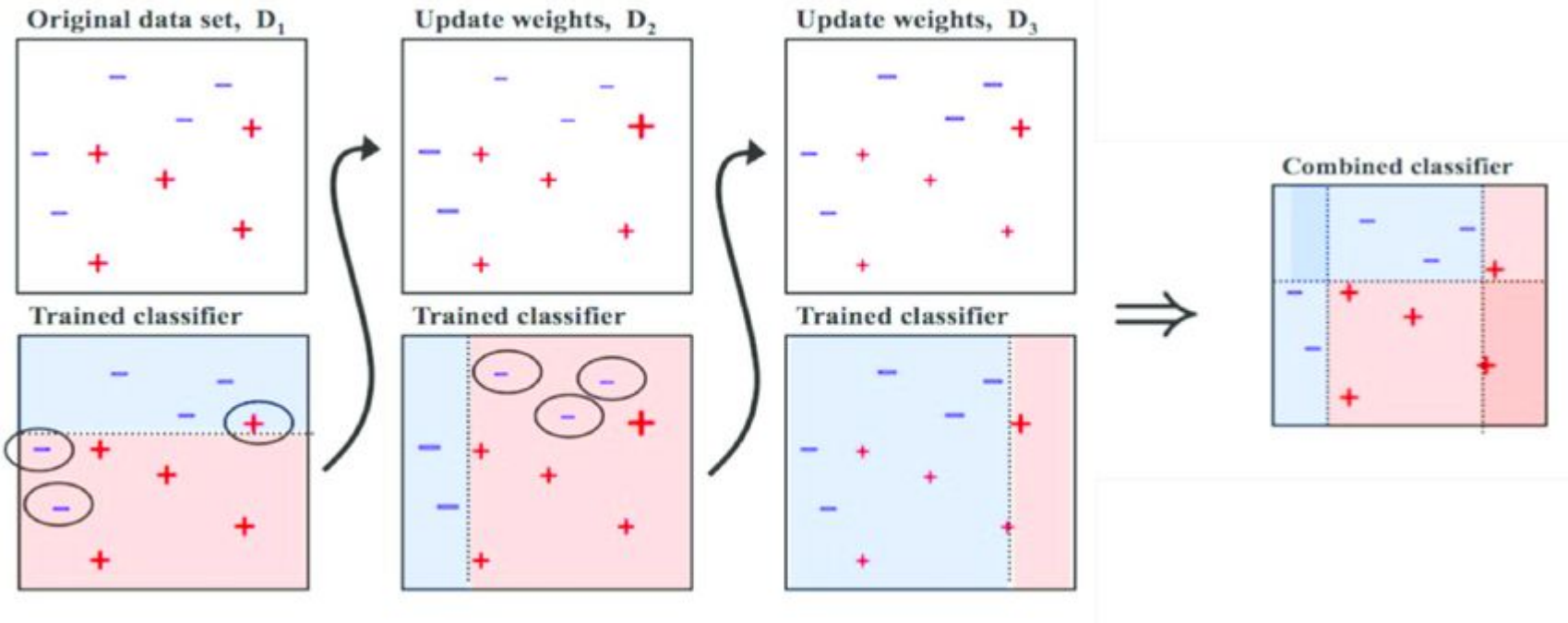
Bagging vs. Boosting



Boosting Models

- **Boosting** is a statistical framework where the objective is to minimize the loss of the model by adding weak learners using a **gradient descent** like procedure.
- This allowed different loss functions to be used, expanding the technique beyond binary classification problems to support regression, multi-class classification and more.
- Boosted algorithm not only exists for trees, but is also a general procedures other classifiers can performan

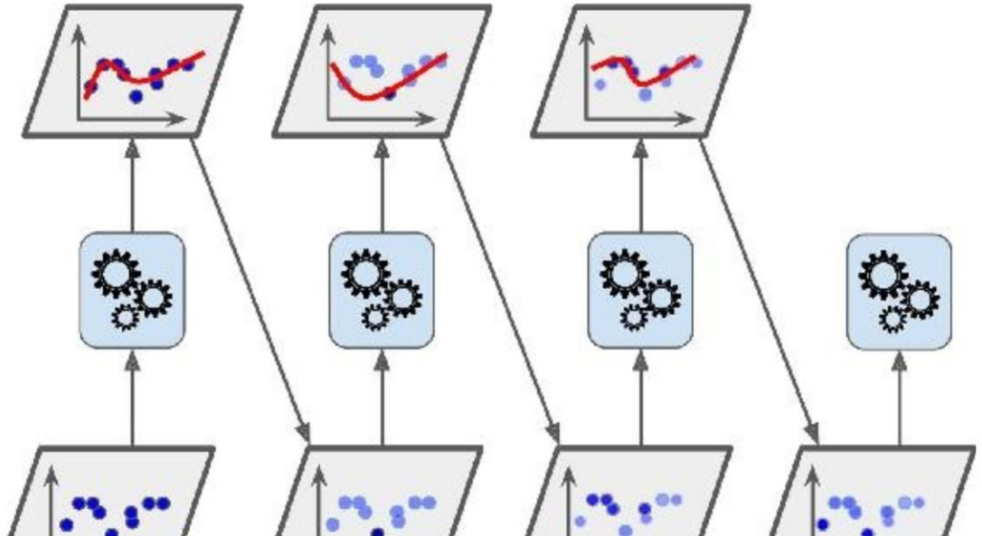
AdaBoost - Intuition



AdaBoost - Adaptive Boosting

AdaBoost is a ensemble method used to solve classification problems.

0. Initialize the weight of each of the observations
1. Fit a base classifier like a Decision tree.
2. Use that classifier to make predictions on the training set.
3. Increase the relative weight of the of the instances that were misclassified
4. Train another classifier using the updated weights.
5. Aggregate all of the classifiers into one, weighting them by their accuracy.



AdaBoost - Adaptive Boosting

1. Initialize the observation weights $w_i = 1/N,\ i = 1, 2, \ldots, N$.
2. For $m = 1$ to M:
   (a) Fit a classifier $G_m(x)$ to the training data using weights w_i.
   (b) Compute
   $$\text{err}_m = \frac{\sum_{i=1}^{N} w_i I(y_i \neq G_m(x_i))}{\sum_{i=1}^{N} w_i}.$$
   (c) Compute $\alpha_m = \log((1 - \text{err}_m)/\text{err}_m)$.
   (d) Set $w_i \leftarrow w_i \cdot \exp[\alpha_m \cdot I(y_i \neq G_m(x_i))],\ i = 1, 2, \ldots, N$.
3. Output $G(x) = \text{sign}\left[\sum_{m=1}^{M} \alpha_m G_m(x)\right]$.

Classification Models

The first classifier is trained using a ***random subset*** of overall training set...

Misclassified item is assigned higher weight so that it appears in the training subset of next classifier with higher probability.

Weighting the Classifiers

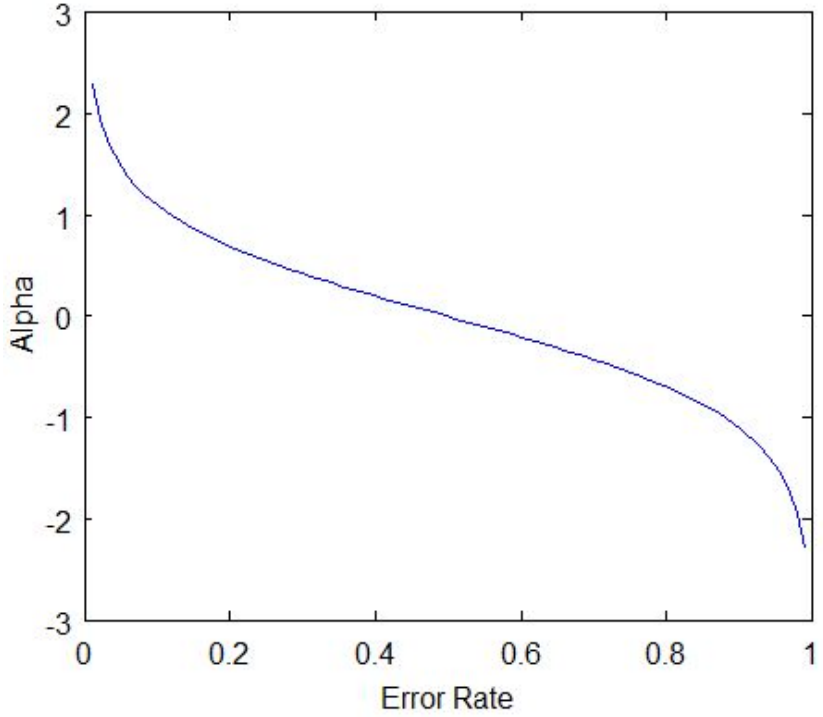
After each classifier is trained, a weight is assigned to the classifier on accuracy. More accurate classifier is assigned higher weight so that it will have more impact in final outcome.

$$H(x) = sign\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right)$$

Equation for the final classifier

$$\alpha_t = \frac{1}{2} ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right)$$

Calculation of weight for each individual classifier



Regression vs. Classification for AdaBoost

Regression	Classification
• Start with a weak tree • Use loss function to determine weight to modify predicted values • Make a new tree • Use these trees together	• Start with a weak tree • Use misclassification penalty to determine weight to modify predicted values • Make a new tree • Use these trees together

Data Preparation for AdaBoost

This section lists some heuristics for best preparing your data for AdaBoost.

- **Quality Data**: Because the ensemble method continues to attempt to correct misclassifications in the training data, you need to be careful that the training data is of a high-quality.
- **Outliers**: Outliers will force the ensemble down the rabbit hole of working hard to correct for cases that are unrealistic. These could be removed from the training dataset.
- **Noisy Data**: Noisy data, specifically noise in the output variable can be problematic. If possible, attempt to isolate and clean these from your training dataset.

AdaBoost Summary

AdaBoost sequentially trains classifiers.

Tries to improve classifier by looking at the misclassified instances.

The algorithm weights misclassified instance more so they are more likely to be included in the training data subset.

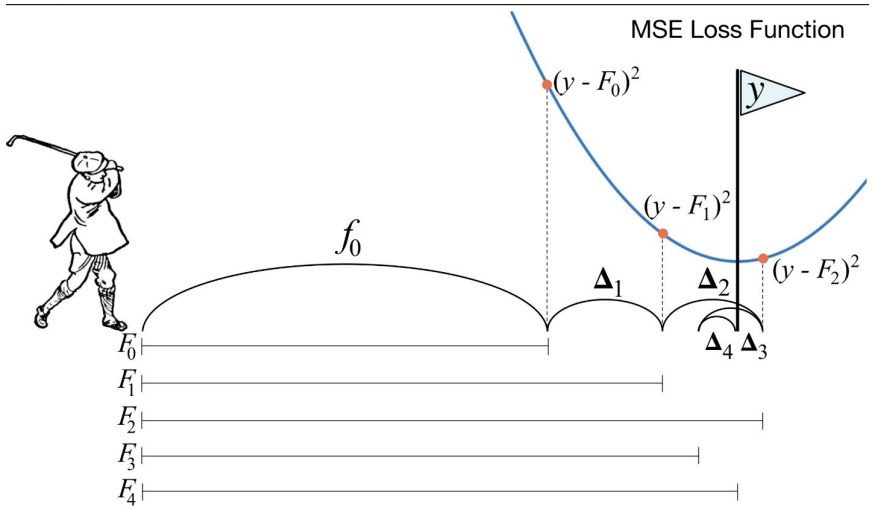
Each classifier is weighted based on their accuracy and then all are aggregated to create the final classifier.

Doesn’t perform well on very noisy data or data with outliers.

AdaBoost can use any type of classification model, not just a decision tree.

Part II: Gradient Boosting

- Just like AdaBoost, Gradient Boost evaluates the collection of trees sequentially, and gradually ensembles a collection of trees that model the underlying behavior of our data
- Unlike Adaboost, Gradient Boost evaluates the residuals of the previous tree and try to predict the residual



Steps in Gradient Boosting

Input: Data $\{(x_i, y_i)\}_{i=1}^{n}$, and a differentiable **Loss Function** $L(y_i, F(x))$

Step 1: Initialize model with a constant value: $F_0(x) = \underset{\gamma}{\operatorname{argmin}} \sum_{i=1}^{n} L(y_i, \gamma)$

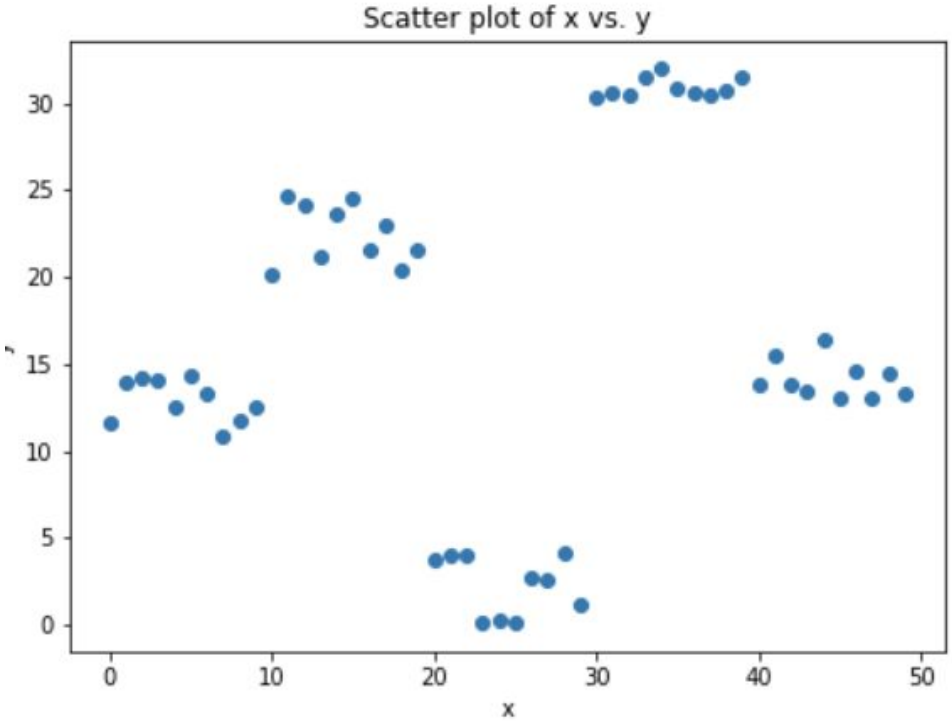
Step 2: for $m = 1$ to M:

(A) Compute $r_{im} = -\left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{m-1}(x)}$ for $i = 1,\ldots,n$

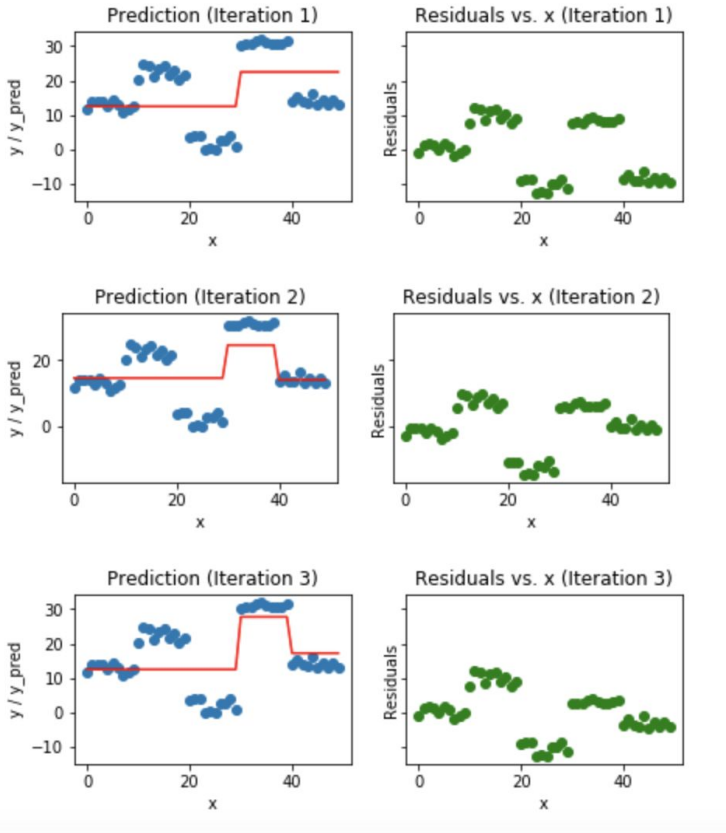
(B) Fit a regression tree to the r_{im} values and create terminal regions R_{jm}, for $j = 1 \ldots J_m$

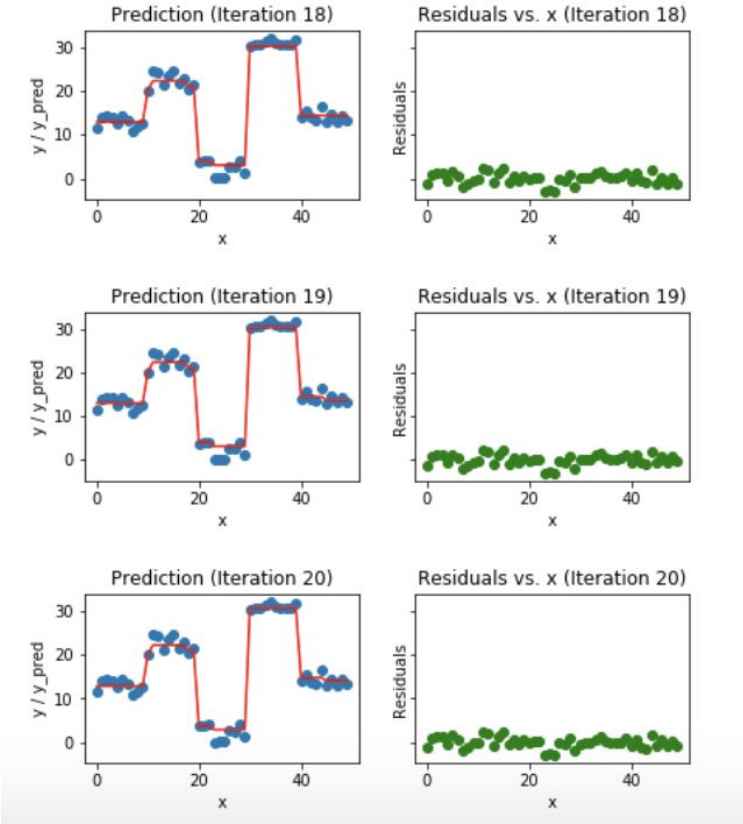
(C) For $j = 1 \ldots J_m$ compute $\gamma_{jm} = \underset{\gamma}{\operatorname{argmin}} \sum_{x_i \in R_{ij}} L(y_i, F_{m-1}(x_i) + \gamma)$

(D) Update $F_m(x) = F_{m-1}(x) + \nu \sum_{j=1}^{J_m} \gamma_{jm} I(x \in R_{jm})$

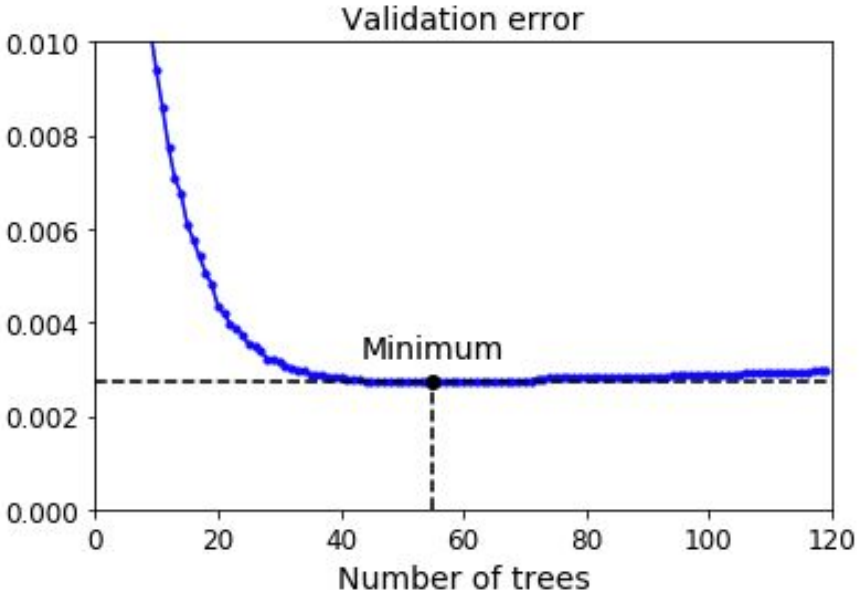

Scatter plot of x vs. y
30
25
20
15
10
5
0
0
10
20
30
40
50
x

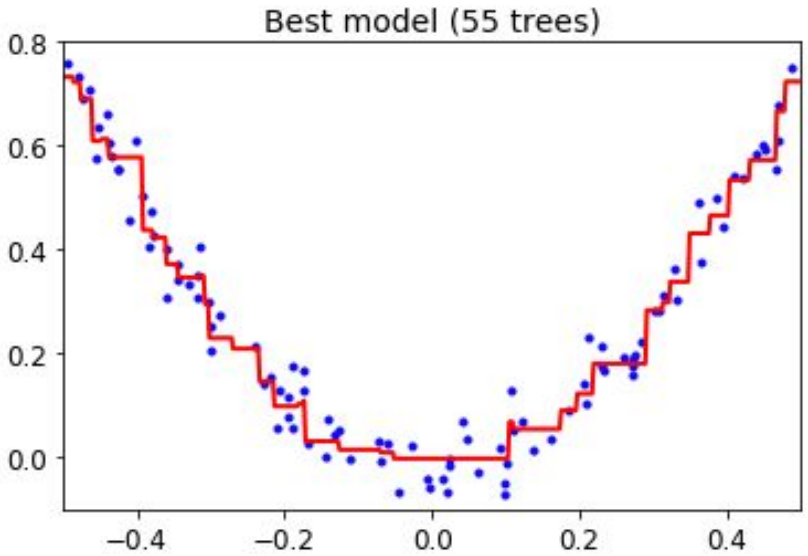
Gradient Boosting over Iterations





Validation Error over Iterations





Shrinkage - Learning Rate

The contribution of each tree to this sum can be weighted to slow down the learning by the algorithm. This weighting is called a **shrinkage** or a **learning rate**.

For each gradient step, the step magnitude is multiplied by a factor between 0 and 1

Shrinkage causes sample-predictions to slowly converge toward observed values.

As this slow convergence occurs, samples that get closer to their target end up being grouped together into leaves, resulting in a natural regularization effect.

Shrinkage Visualized

https://github.com/fpolchow/boosting_notes

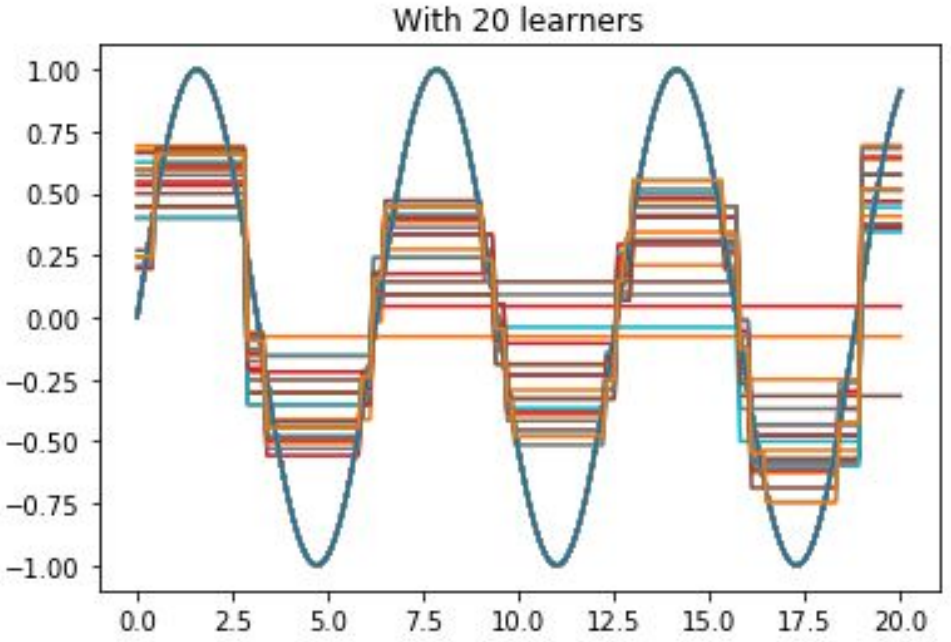
```
def simple_boosting_algorithm(X,y,n_learners,learner,learning_rate,show_each_step = True):
    """Performs a simple ensemble boosting model
    params: show_each_step - if True, will show with each additional learner"""
    f0 = y.mean()
    residuals = y - f0
    ensemble_predictions = np.full(len(y),fill_value=f0)
    plt.figure(figsize=(20,10))
    for i in range(n_learners):
        residuals = y - ensemble_predictions
        f = learner.fit(X.reshape(-1,1),residuals)
        ensemble_predictions = learning_rate * f.predict(X.reshape(-1,1)) + ensemble_predictions
        if show_each_step:
            plt.plot(X,y)
            plt.plot(X,ensemble_predictions)

    plt.plot(X,y)
    plt.plot(X,ensemble_predictions)

    plt.title('With ' + str(n_learners) + ' learners with a depth of '+ str(learner.max_depth) + ' and a lea
```

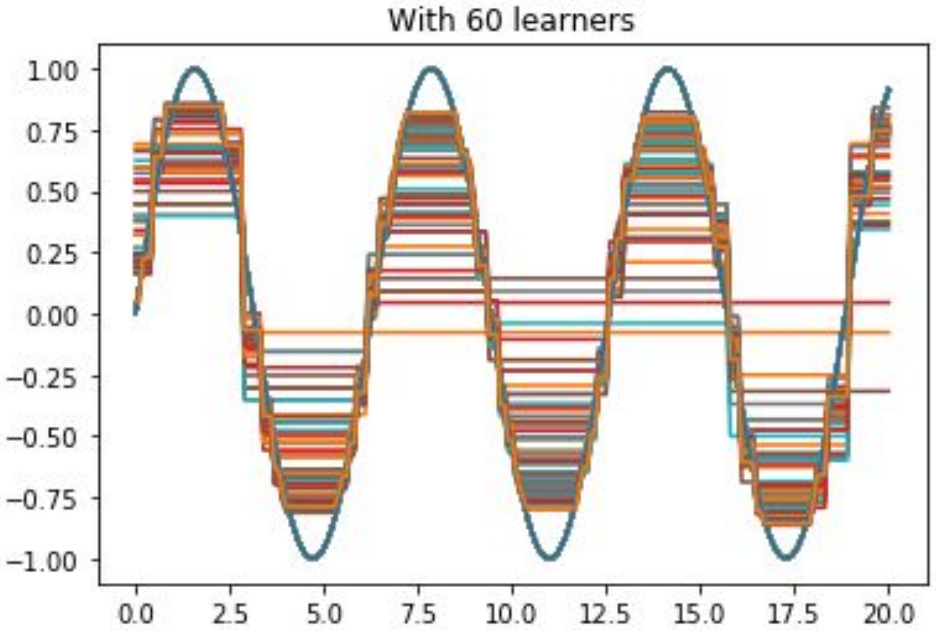
Shrinkage - Slow Convergence

```
simple_boosting_algorithm(X,np.sin(X),20,tree.DecisionTreeRegressor(max_depth=1),0.001)
```



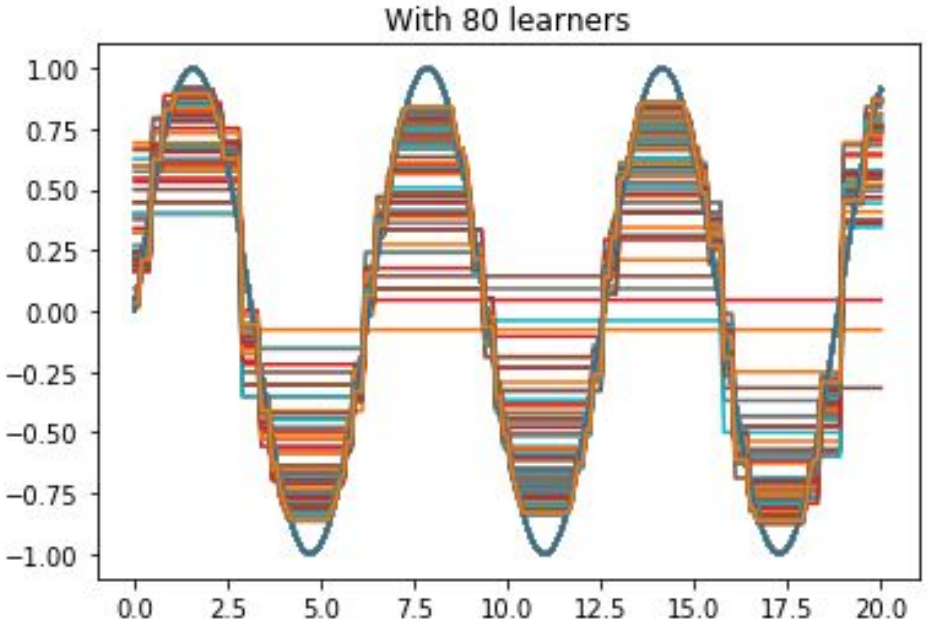
Shrinkage - Slow Convergence

```
simple_boosting_algorithm(X,np.sin(X),60,tree.DecisionTreeRegressor(max_depth=1),0.001)
```



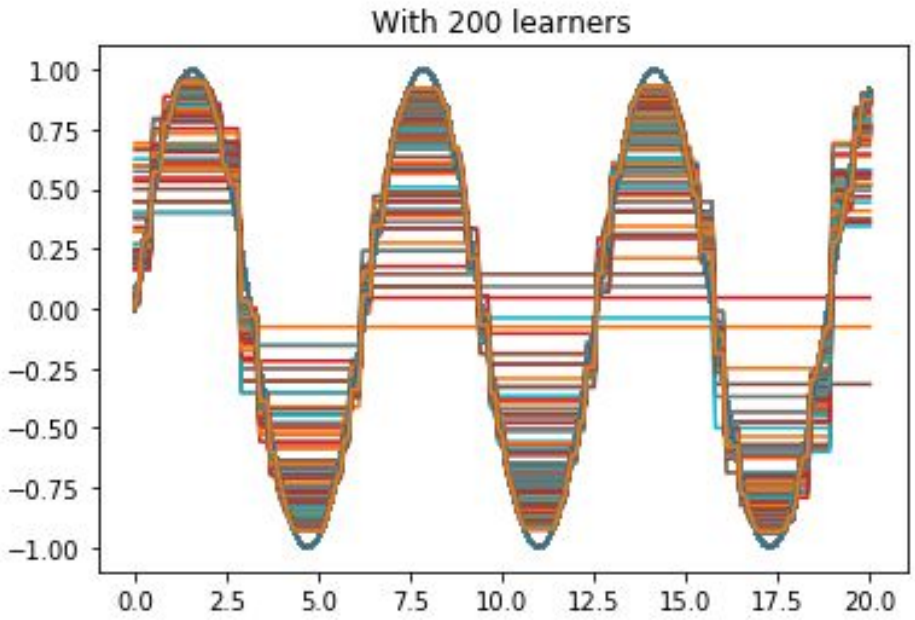
Shrinkage - Slow Convergence

```
simple_boosting_algorithm(X,np.sin(X),80,tree.DecisionTreeRegressor(max_depth=1),0.001)
```

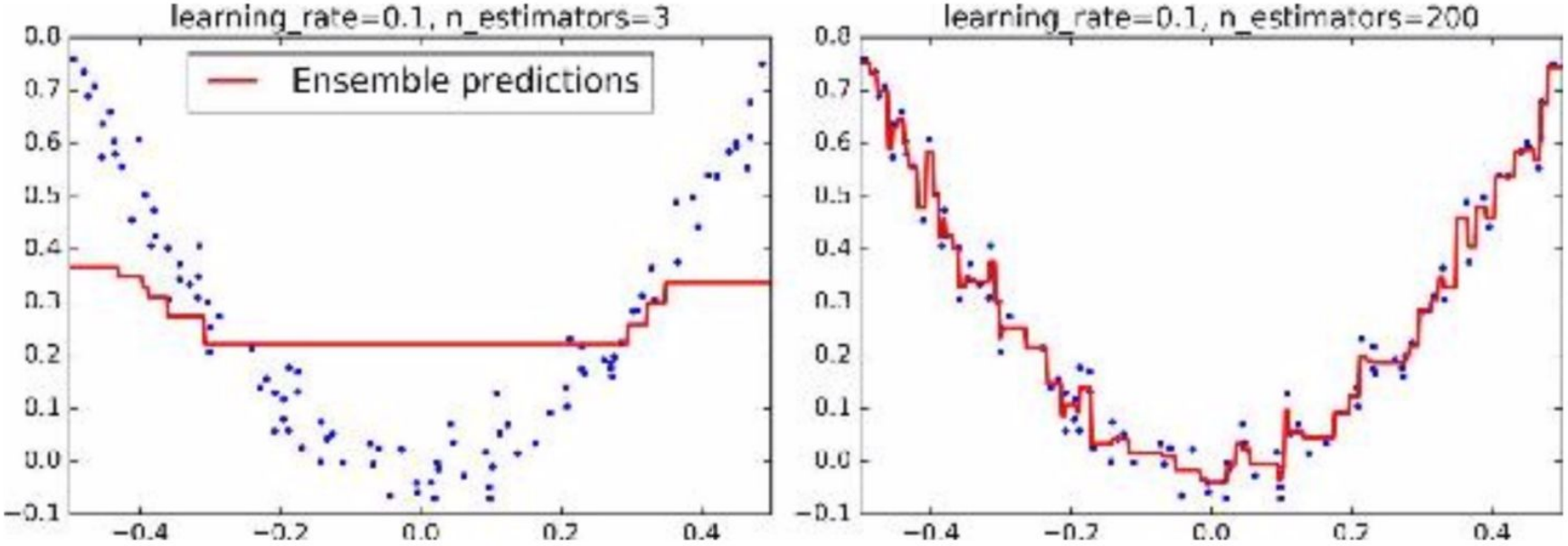


Shrinkage - Slow Convergence

```
simple_boosting_algorithm(X,np.sin(X),200,tree.DecisionTreeRegressor(max_depth=1),0.001)
```



The Effect of Estimators



Regression vs. Classification in Gradient Boosting

Regression	Classification
• Start with a weak tree • Minimize objective function (cost function) by finding residuals of prior tree • Create new tree using gradients from prior tree	• Start with a weak tree • Maximize log likelihood by calculating the partial derivative of log probability for all objects in the dataset. • Create a new tree using these gradients and add it to your ensemble using a pre-determined weight, which is the gradient descent learning rate.

XGBoost

Both xgboost and gbm follows the principle of gradient boosting. However, there is a difference in the modeling process. Specifically, xgboost uses a more regularized model formalization to control over-fitting, which gives it better performance.

The name xgboost, though, actually refers to the engineering goal to push the limit of computations resources for boosted tree algorithms. Which is the reason why many people use xgboost.

- Has its own library -- not part of SKLearn
- Harder, better, faster, stronger
- ANY cost function, and more parameters
- Must be twice-differentiable because it uses something called a “Hessian” in the gradient descent algorithm. This is like another weight in addition to the learning rate, just like Shrinkage
- Built-in regularization (L2 by default) based on node weights.

Great Resources

Boosting Models:
https://machinelearningmastery.com/gentle-introduction-gradient-boosting-algorithm-machine-learning/

AdaBoost: http://mccormickml.com/2013/12/13/adaboost-tutorial/

Gradient Descent: https://medium.com/mlreview/gradient-boosting-from-scratch-1e317ae4587d

Comparison of Gradient Boosting and XGBoost:
https://medium.com/@gabrieltseng/gradient-boosting-and-xgboost-c306c1bcfaf5