Boosted Algorithms

Data Science Immersive

Lesson Overview

Aim: SWBAT 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

Starting off

- Let’s play a game! Wisdom of the crowd vs wisdom of “experts”?
- How many days in a million seconds?
  - We’ll do this one of the two ways:
    - Write down your individual answer on the board and show me at the same time
    - Each tell me your answer one by one (sequentially)

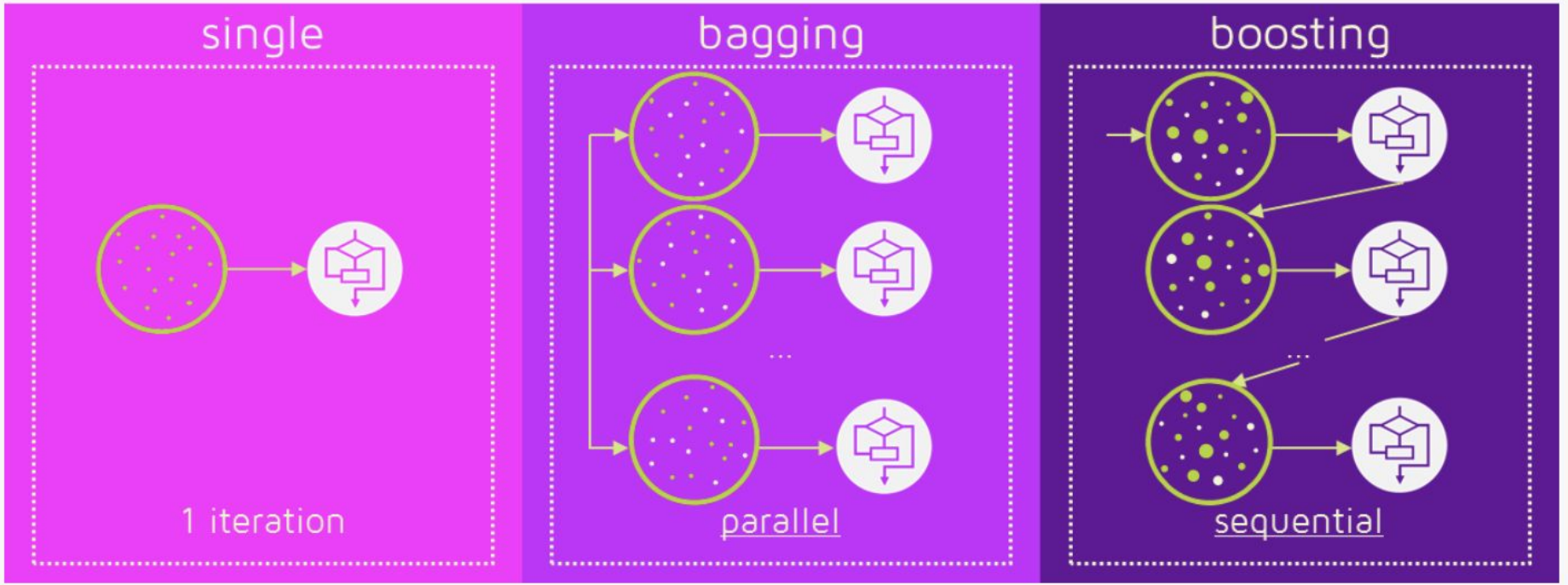
Boosting

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.

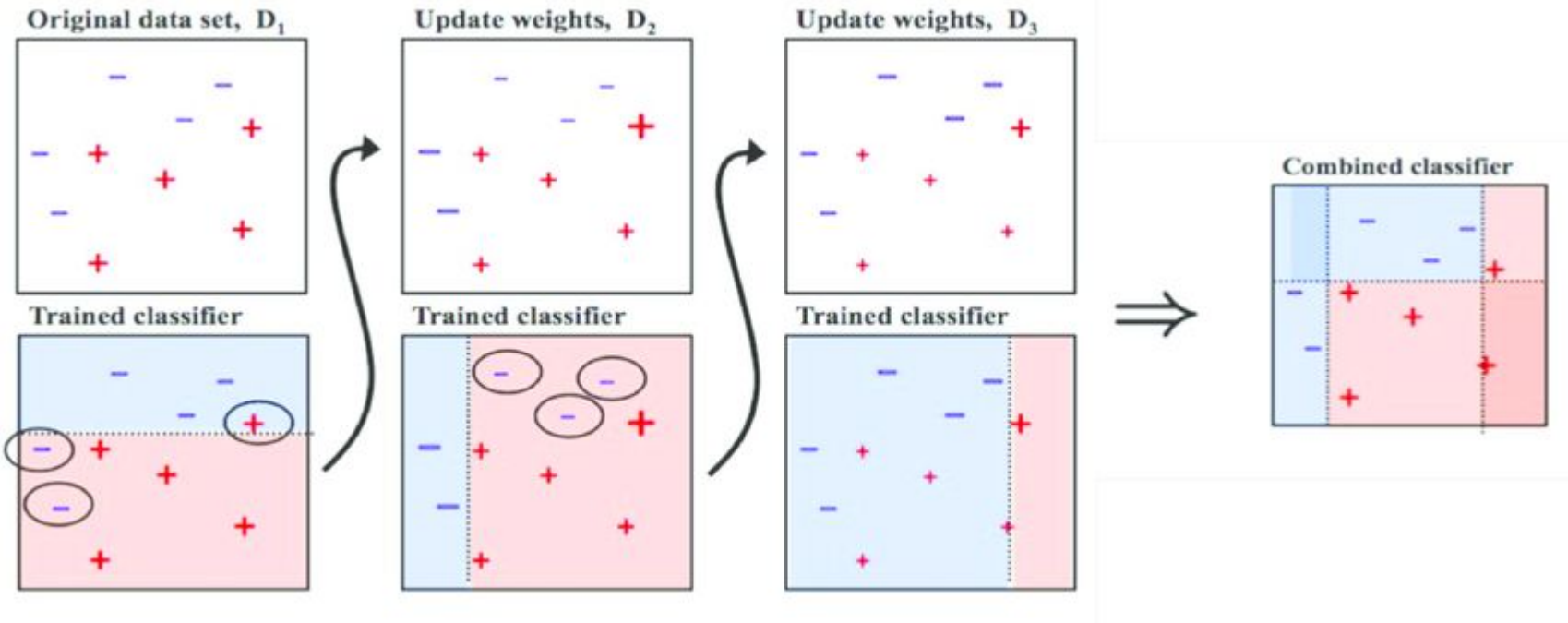
Bagging vs. Boosting



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

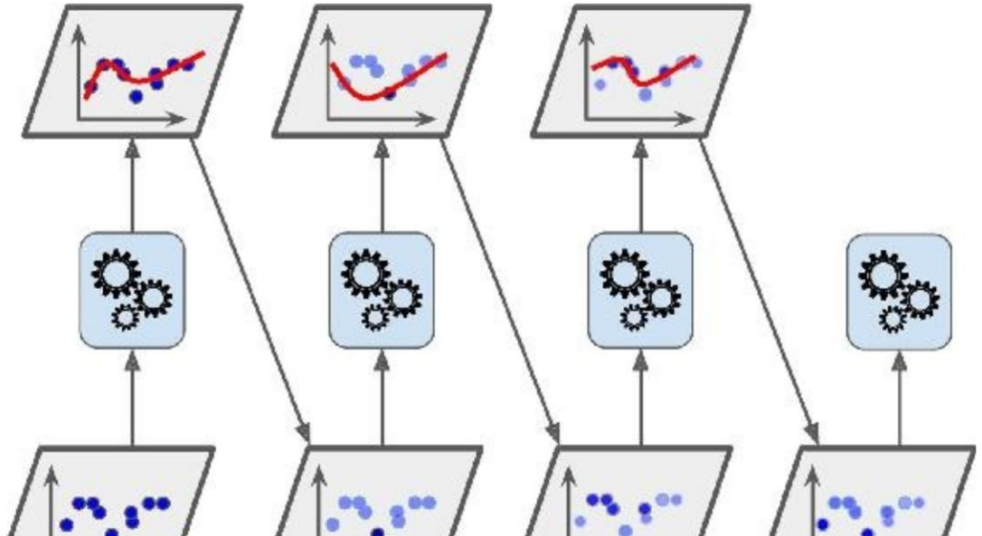
AdaBoost - Intuition



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

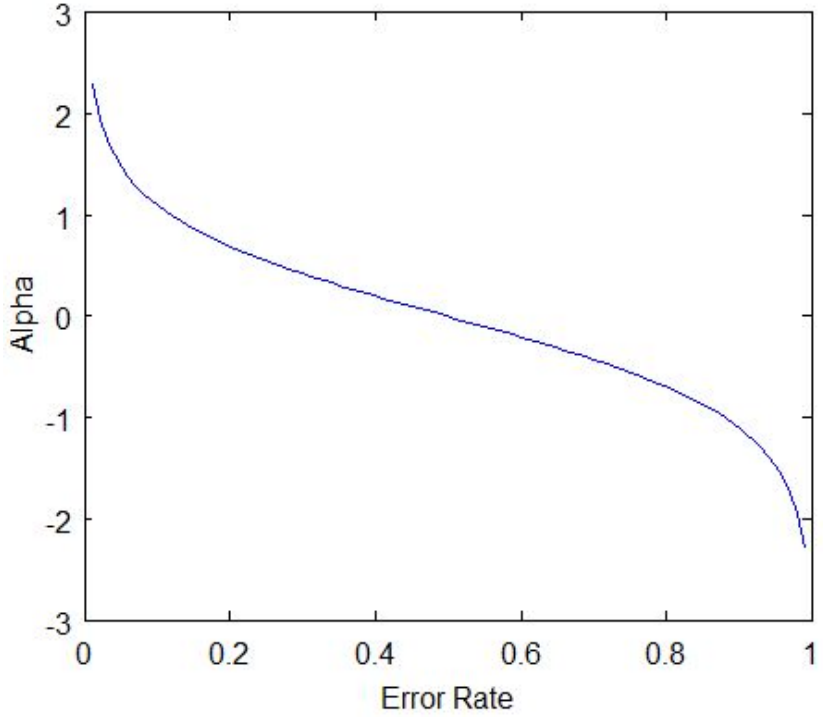
After each classifier is trained, a weight is assigned to the classifier on accuracy. More accurate classifier is assigned higher weight so that it will have more impact in final outcome.

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

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.

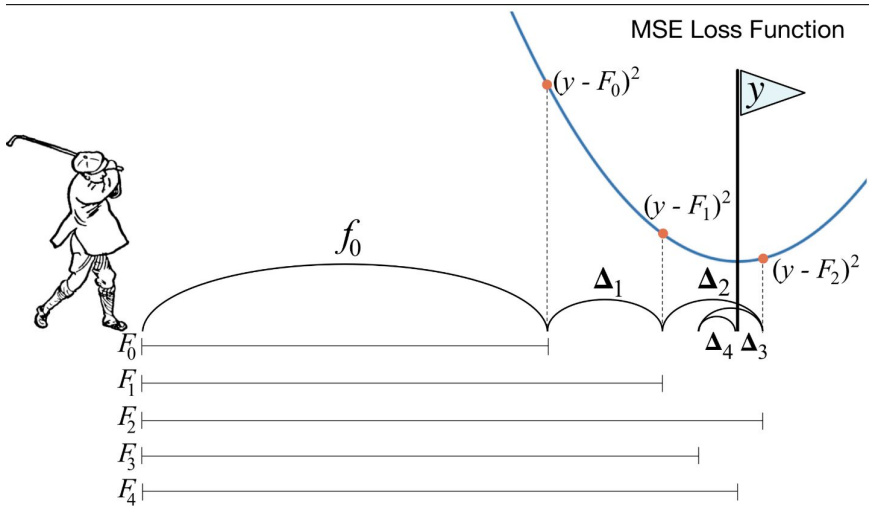
Each classifier is weighted based on their accuracy and then all are aggregated to create the final classifier.

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

1. Fit a simple linear regression or decision tree on data **[call x as input and y as output]**

2. Calculate error residuals. **[e1= y - y_predicted1]**

3. Fit a new model on error residuals as target variable with same input variables **[call it e1_predicted]**

4. Add the predicted residuals to the previous predictions

[y_predicted2 = y_predicted1 + e1_predicted]

5. Fit another model on residuals that is still left. i.e. **[e2 = y - y_predicted2]**

Repeat steps 2 to 5 until it starts overfitting or the sum of residuals become constant.

Overfitting can be controlled by consistently checking accuracy on validation data.

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: $F_0(x) = \underset{\gamma}{\operatorname{argmin}} \sum_{i=1}^{n} L(y_i, \gamma)$

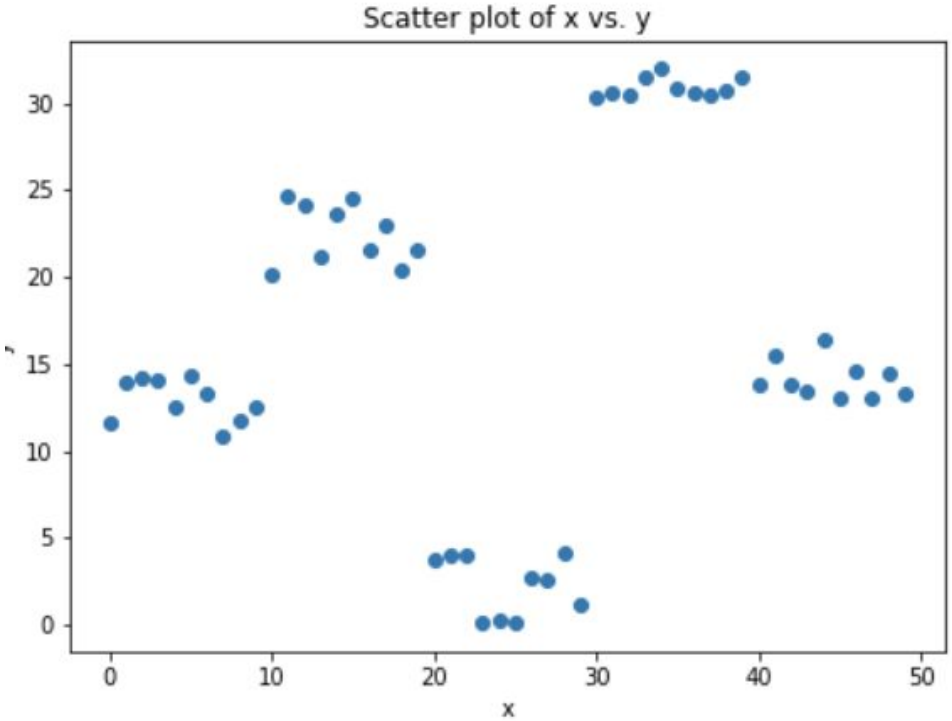
Step 2: for $m = 1$ to M:

(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 $i = 1,\ldots,n$

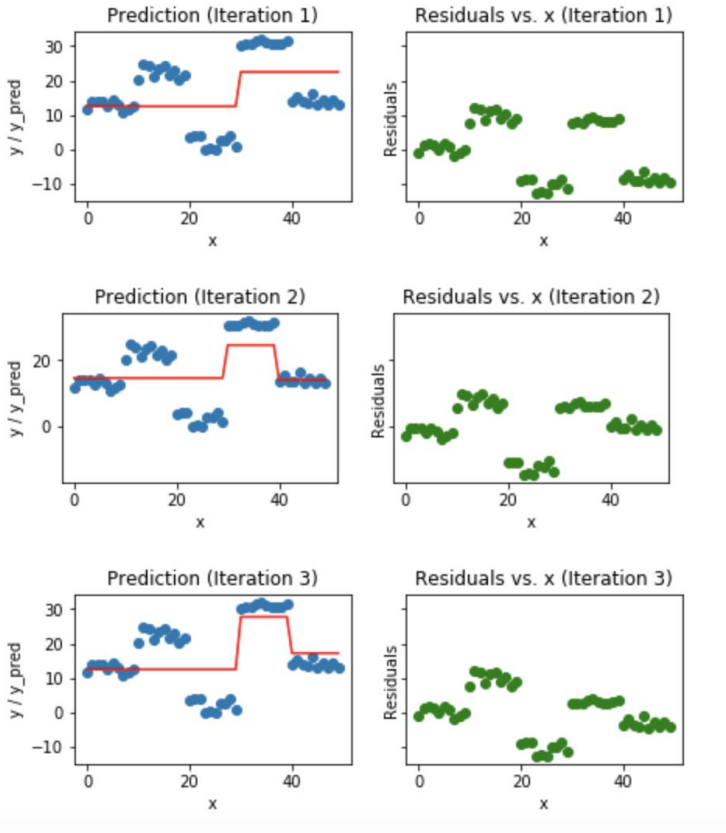
(B) Fit a regression tree to the r_{im} values and create terminal regions R_{jm}, for $j = 1 \ldots J_m$

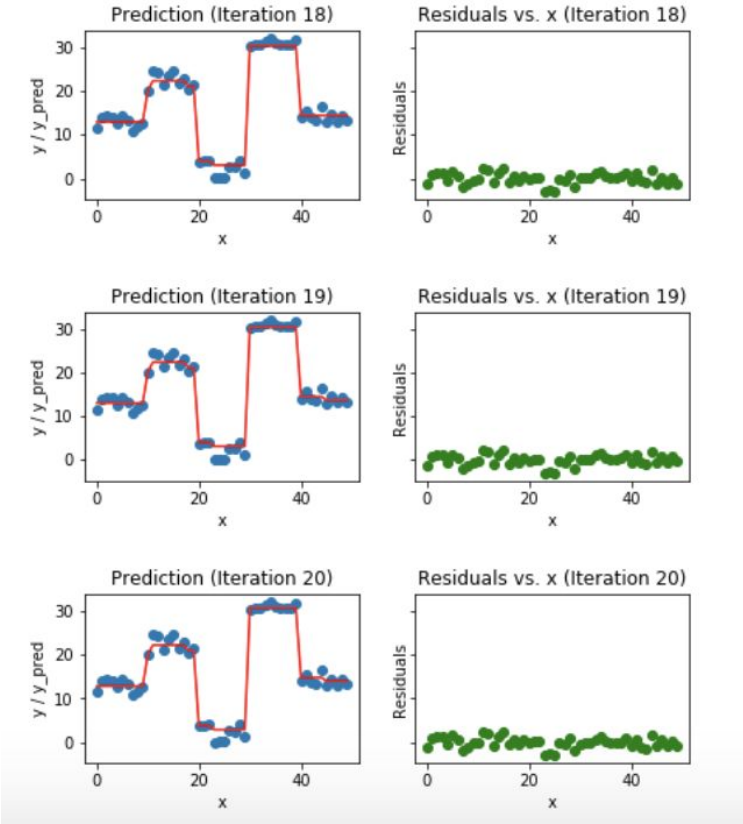
(C) For $j = 1 \ldots J_m$ compute $\gamma_{jm} = \underset{\gamma}{\operatorname{argmin}} \sum_{x_i \in R_{ij}} L(y_i, F_{m-1}(x_i) + \gamma)$

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

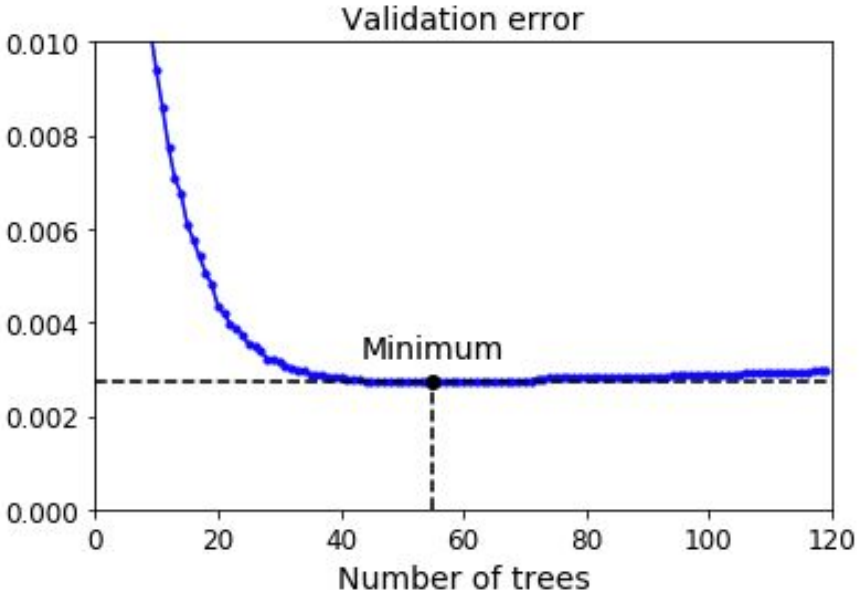

Scatter plot of x vs. y
30
25
20
15
10
5
0
0
10
20
30
40
50
x

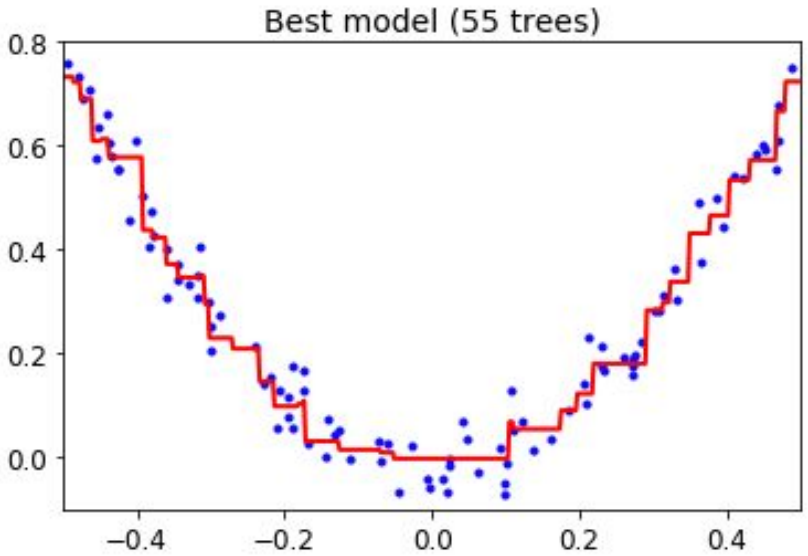
Gradient Boosting over Iterations





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 larger and larger leaves (due to fixed tree size parameters), resulting in a natural regularization effect.

Shrinkage Visualized

https://github.com/fpolchow/boosting_notes

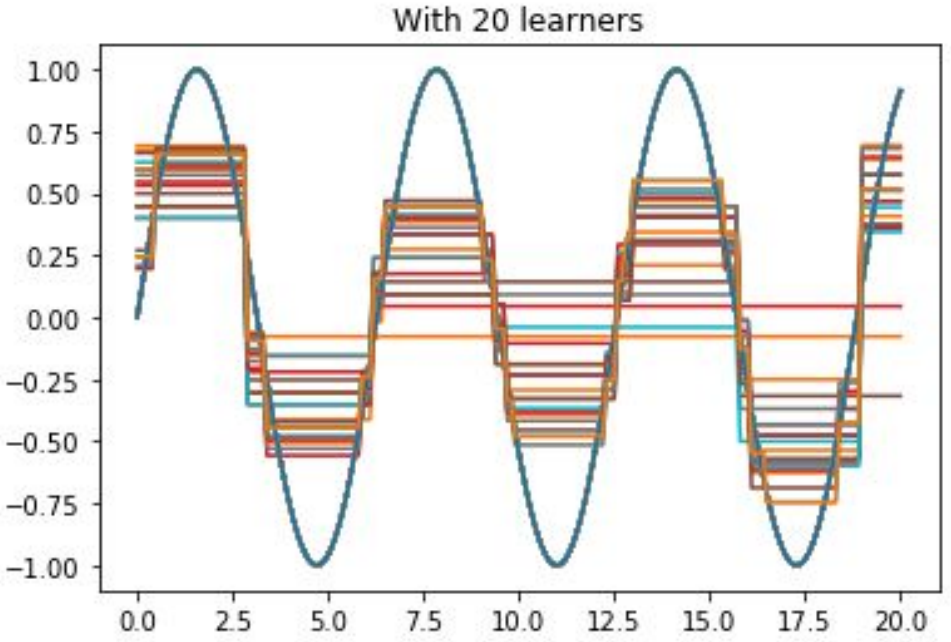
```
def simple_boosting_algorithm(X,y,n_learners,learner,learning_rate,show_each_step = True):
    """Performs a simple ensemble boosting model
    params: show_each_step - if True, will show with each additional learner"""
    f0 = y.mean()
    residuals = y - f0
    ensemble_predictions = np.full(len(y),fill_value=f0)
    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
```

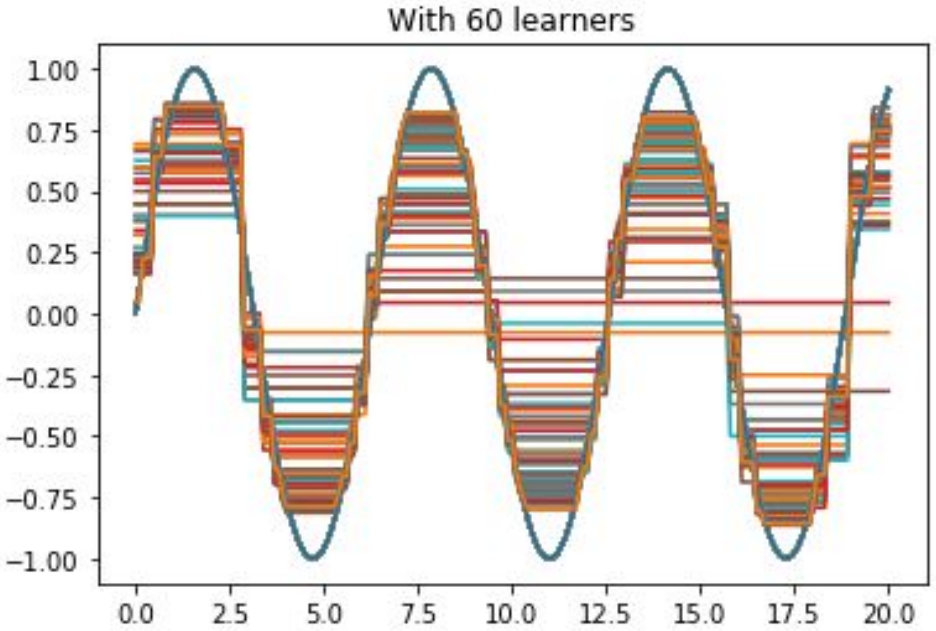
Shrinkage - Slow Convergence

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



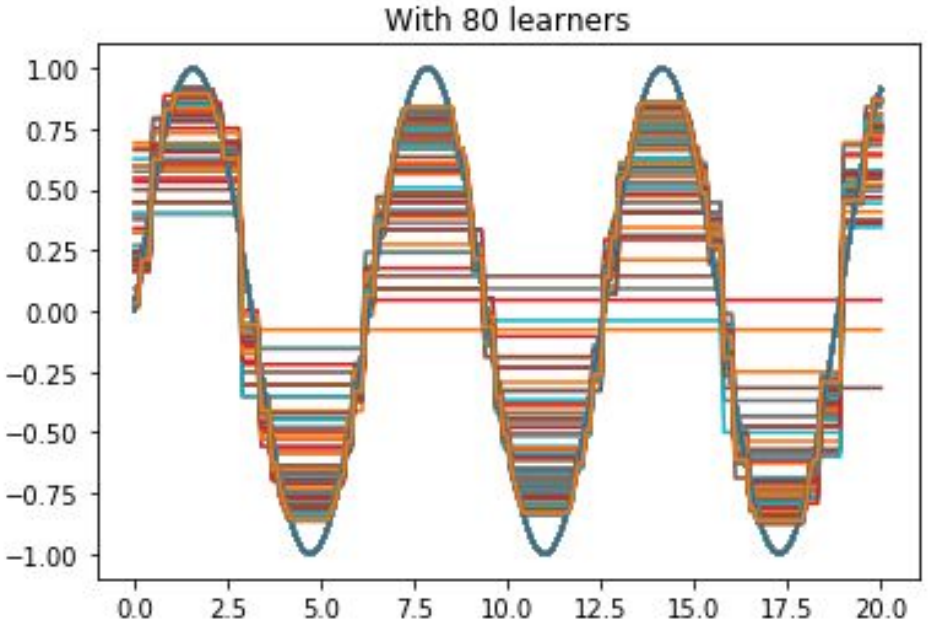
Shrinkage - Slow Convergence

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



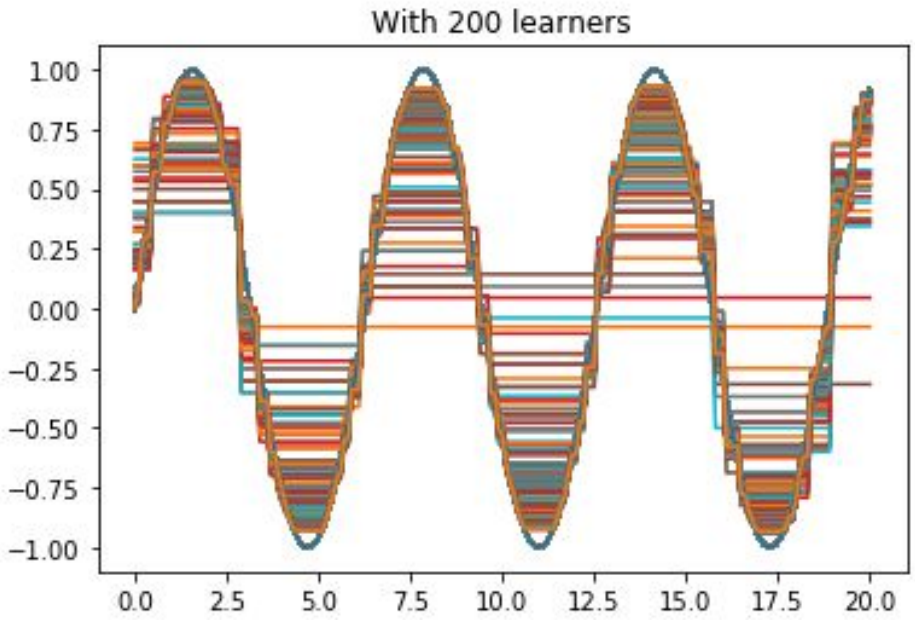
Shrinkage - Slow Convergence

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

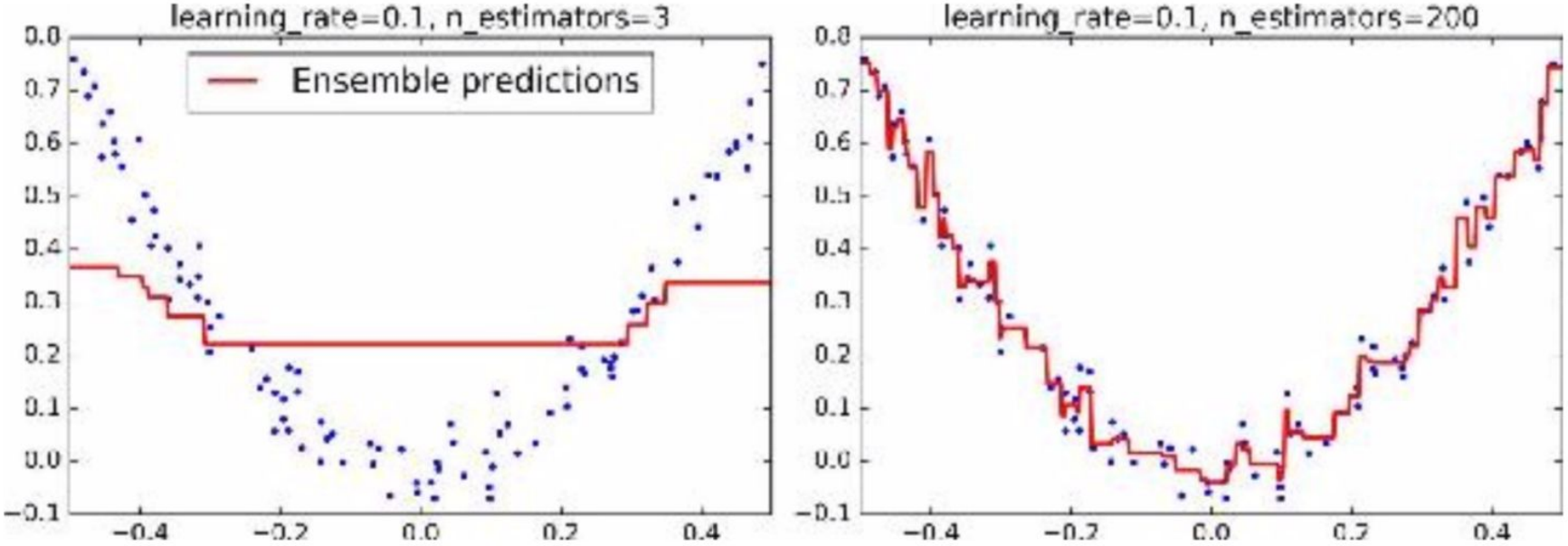


Shrinkage - Slow Convergence

```
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 (instead of exogenous vars) 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. There are however, the difference in modeling details. Specifically, xgboost used 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

Any Questions?