Boosting

What Is Boosting?

- Boosting is a general method of using an ensemble of weak learners to produce a strong learner.
  - Weak means the learner is only slightly better than a random guess.
- Boosting is composed of “rounds,” which can be thought of as iterations or new learners.
  - Each round of boosting tries to improve on the previous prediction.
  - Each iteration is sequential to previous iterations—they are not done in parallel.
  - Each iteration tries to learn from the errors of the previous iteration.

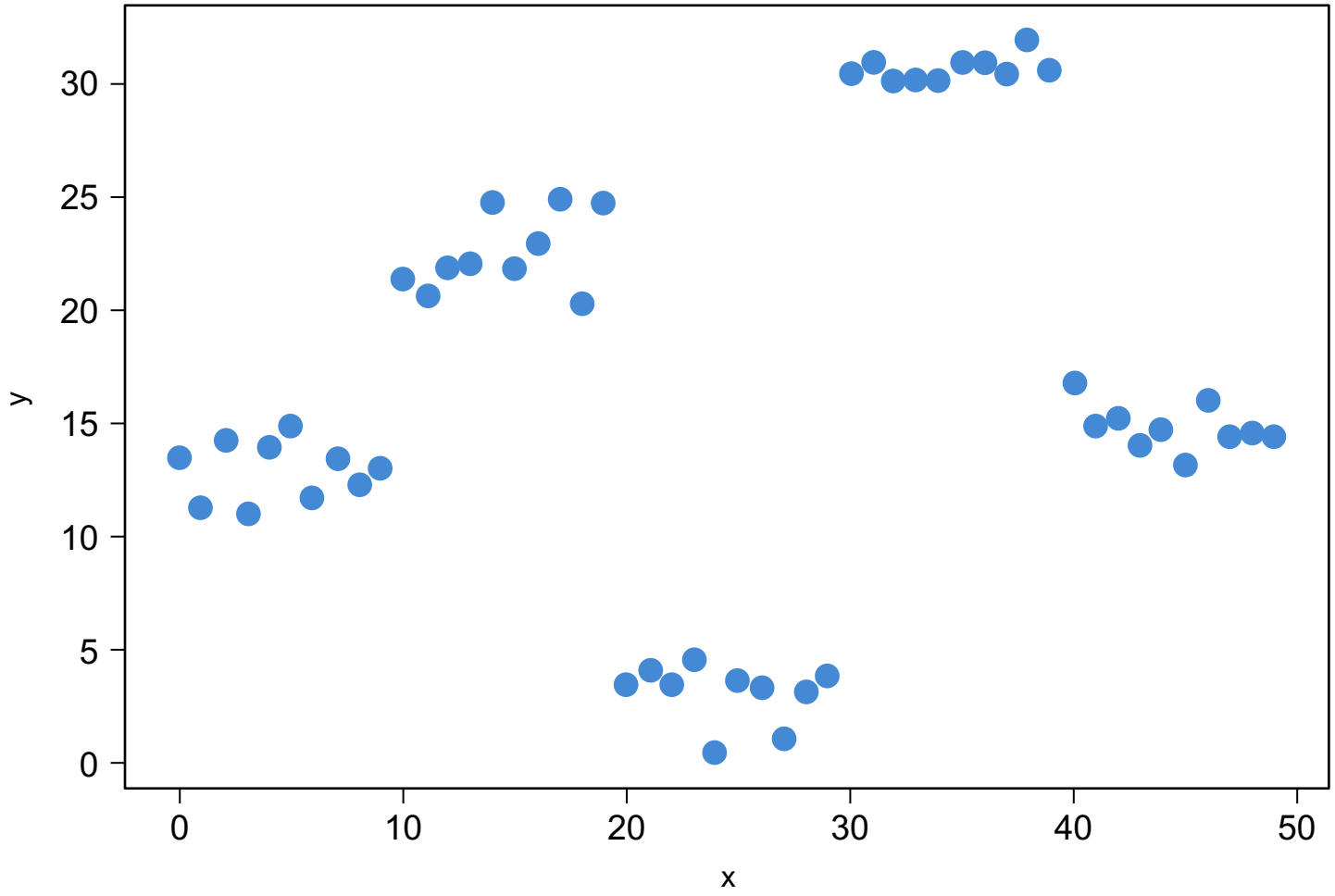
Basic Boosting Algorithm

1. Fit a model to the targets
2. Make predictions with the model
3. Calculate the residuals (targets − predictions)
4. Are the residuals randomly distributed?
    1) Yes = **Stop**
    2) No = Continue
5. Set the residuals as the new targets
6. Return to step (1)

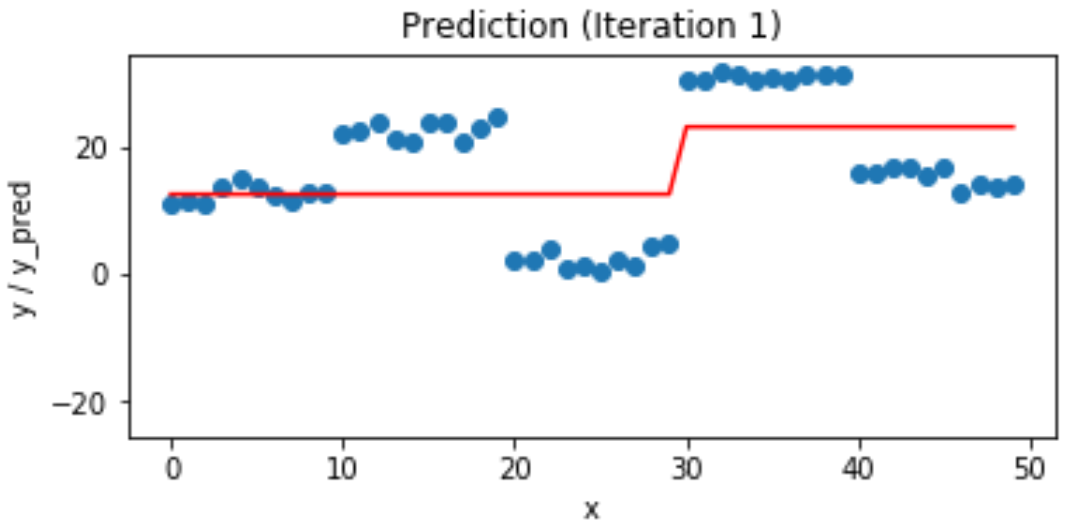
DataScience@SMU

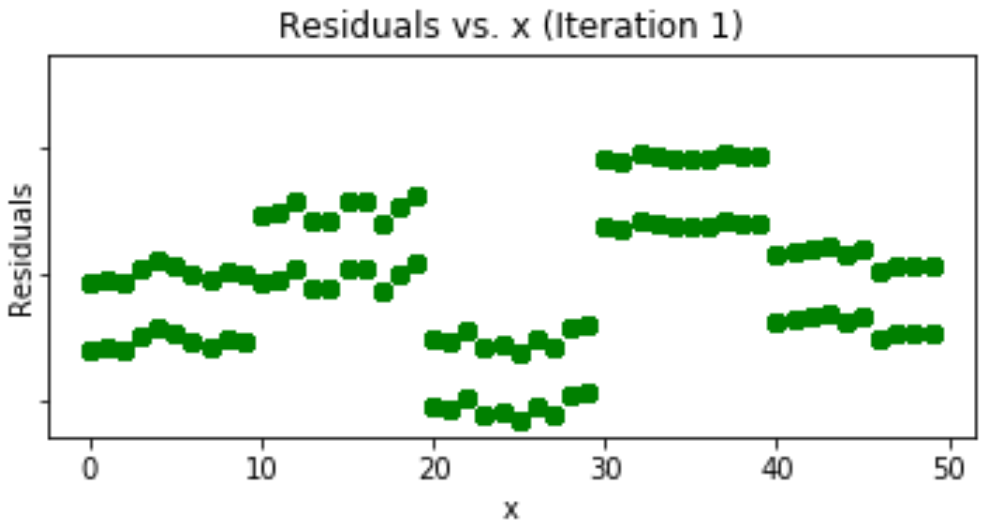
Boosting Walkthrough

Create a Nonlinear Dataset



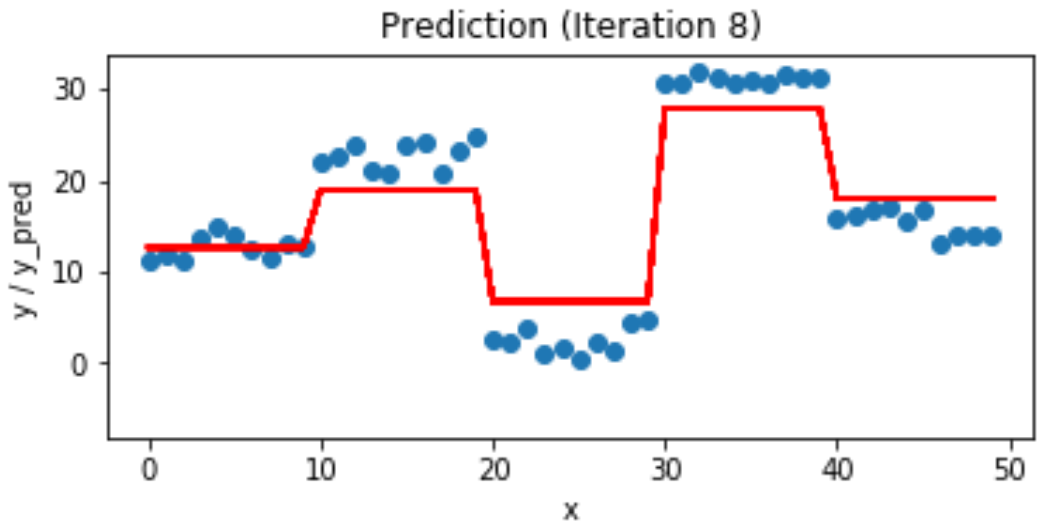
Use a Weak Decision Tree

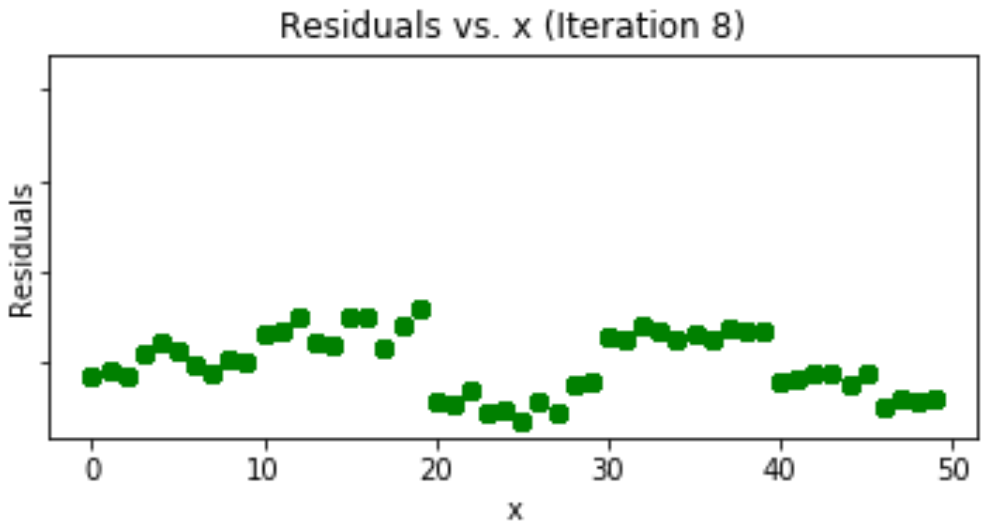




We see this is not a very good prediction, and the residuals are definitely not random. The decision tree was purposely set to a max depth of 1.

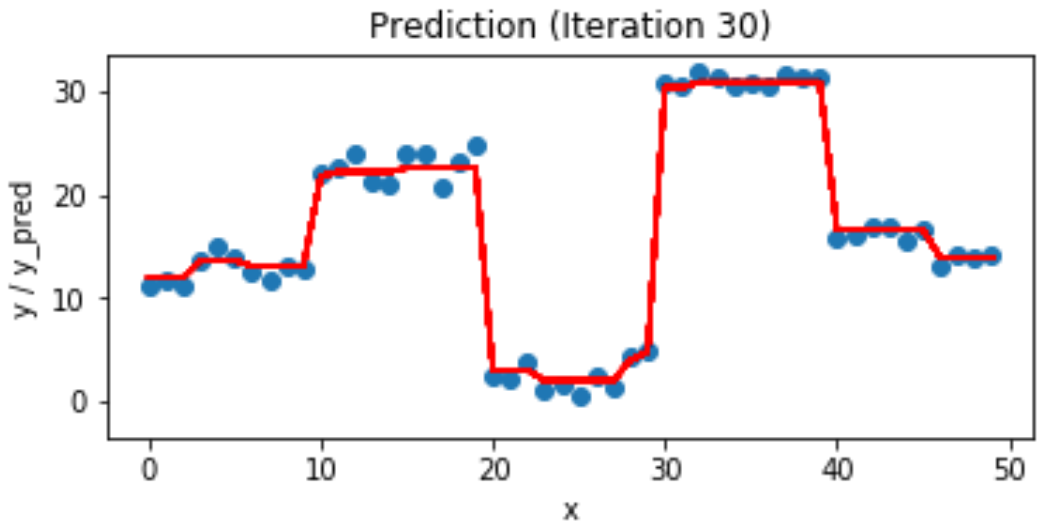
After Iteration 8

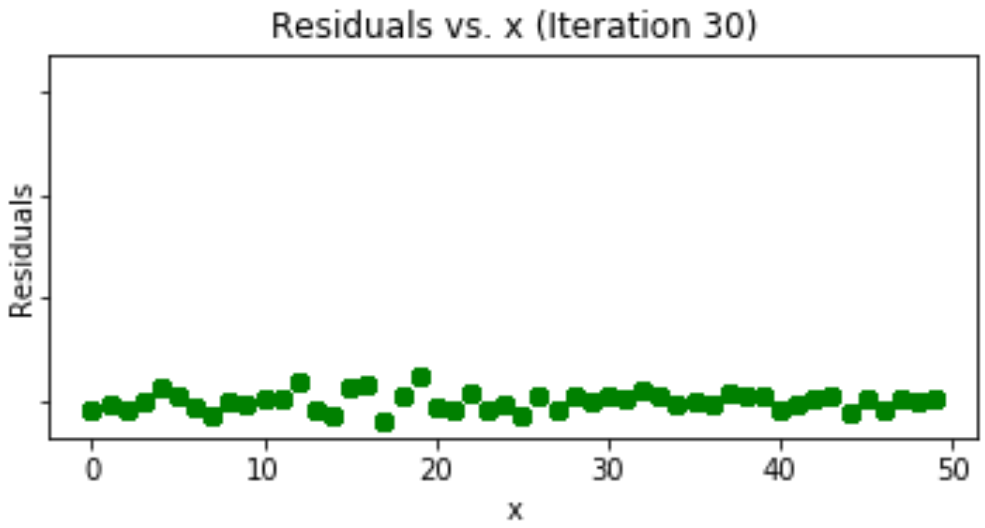




While not perfect, we see that the model is beginning to become a stronger and stronger learner.

After Iteration 30





We now see a very strong, even possibly overfit model.

DataScience@SMU

XGBoost

XGBoost

- XGBoost stands for eXtreme Gradient Boosting.
- The concepts of boosting still apply.
- However, now the gradient is involved.

The Key Concept: Loss

$$J = \sum_i l(p_i, y_i) + \sum_k \Omega(f_k)$$

This looks a lot worse than it is. The first term is just a general loss function of the targets and predictions.

$$\Omega(f) = \gamma T + \frac{1}{2}\lambda ||w||^2$$

The second term is the penalties for terminal nodes/leaves (T) and leaf scores (w). W is very similar to slopes in linear regressors, and T is the total number of leaves. So as before, our regularization term penalizes complexity.

The Second Key Ingredient

- XGBoost uses an **approximate** loss.
- Rather than finding a specific update rule, XGBoost approximates the loss function.
- This means XGBoost can use **any** loss function with a **first and second order partial derivative**.

DataScience@SMU

Hyperparameters

XGBoost Hyperparameters

- eta: the learning rate; smaller eta makes the process more conservative
- gamma: As we saw in the formula for loss, this penalizes creating more nodes; larger gamma is more conservative
- max_depth: the depth of the tree—number of consecutive decisions that can be made
- min_child_weight: A weight needed to exceed for making a partition (analogous to the cp parameter in CART trees)
- max_delta_step: Used primarily for linear boosting, controls the output of each step

XGBoost Hyperparameters (cont.)

- subsample: At each boosting round, the algorithm can pick a subsample fraction of the training data; this helps prevent overfitting; this is **row** subsampling
- colsample_byX: a group of parameters that allows subsampling of columns at various points in the algorithm, where X can by tree, level, and node
- lambda: L2 regularization (on by default)
- alpha: L1 regularization (off by default)

XGBoost Tuning

- The previous slide mentioned all the typically used hyperparameters
- The reason it is important to understand their role is XGBoost scales can be time consuming; consider three splits for each parameter and a grid search; that would be 3^{11} or 177147 runs

DataScience@SMU