Boosting (morning)

Objectives

By the end of this lecture you will be able to answer the following questions:

- What is the general idea of boosting and how it compares to other ensemble methods (e.g., Random Forests)?
- What is the general form of the gradient boosting algorithm?
- What are each of the tunable hyper-parameters when decision trees are used as the weak learner in boosting? What's the general strategy for fitting boosted trees
- How to implement a grid search over the tunable hyper-parameters when fitting boosted trees? How to interpret the results?
- What are advantages and disadvantages of boosting?

Why does this morning matter?

- Boosting algorithms perform extremely well in prediction settings; XGBoost in particular (a variant of boosting) is one of the top performing algorithms on Kaggle
- If you need to squeeze out that last ounce of performance, boosting is a good way to go

Boosting Motivation

Boosting - Example 1

It's exam time!

Figure 1:Exam time!

Normally, this would stink...

Figure 2:Exams stink, right?

Boosting - Example 1

But, today you get to take the exam as a class (kind of)! So, now you're all in it together...

Figure 3:Exams, all in it together...

Boosting - Example 1

The way it's going to work is as follows:

1. The first person will take the test, from start to finish
2. We'll then see how that person did, calculating what he/she got right and wrong
3. After this, we'll pass the test on to the next person, who will take it from start to finish. **But**, this person will also get told what the last person got right and wrong (so in essence they're taking a slightly different test)
4. We'll iterate over steps 2 and 3 until everybody has had a chance to take a **version** of the test, and use the final error rate (how many questions were incorrect) as the grade for each person in the class

Boosting - Example 2

Let's predict the stock market (Step n... profit $$$), specifically how much the market will change for a handful of stocks

Figure 4:How much will it change?

Boosting - Example 2

For predicting, though, we'll have each person predict based off a slighty different version of the data:

1. The first person will look over the original data, and then make predictions on how each stock's price will change for the next day
2. We'll look at tomorrow's prices (which we have) and figure out how much the first person missed by for each stock (maybe it's \$1 for Apple, and \$0 for Microsoft)
3. Next, we'll give the next person a shot at predicting how the stock's price will change for the next day. **But**, this person is going to know how much the person before missed by. This means he/she will be able to focus more on where we're predicting poorly
4. Iterate over steps 2 and 3 until everybody has had one pass over a **version** of the data, and aggregate each of the predictions to get our final result

Gradient Boosting

Gradient Boosting - High Level Overview

Our goal is to construct a function f so that

$$y_i \approx f(x_i) \, \forall i$$

Let's look for an f so that $\sum_i (y_i - f(x))^2$ is small

Gradient Boosting - High Level Overview

As we saw earlier, we would like to build up f in stages, making small adjustments as we go along

In the end, f will be a sum of smaller models (called *weak learners*)

$$f(x) = \phi_0(x) + \phi_1(x) + \phi_2(x) + ... + \phi_M(x)$$

The process of building up the model looks this:

- $G_0(x) = \phi_0(x)$
- $G_1(x) = G_0(x) + \phi_1(x) = \phi_0(x) + \phi_1(x)$
- $G_2(x) = G_1(x) + \phi_2(x) = \phi_0(x) + \phi_1(x) + \phi_2(x)$

...

- $G_M(x) = G_{M-1}(x) + \phi_M(x) =$
  $\phi_0(x) + \phi_1(x) + \phi_2(x) + ... + \phi_M(x)$

Training data

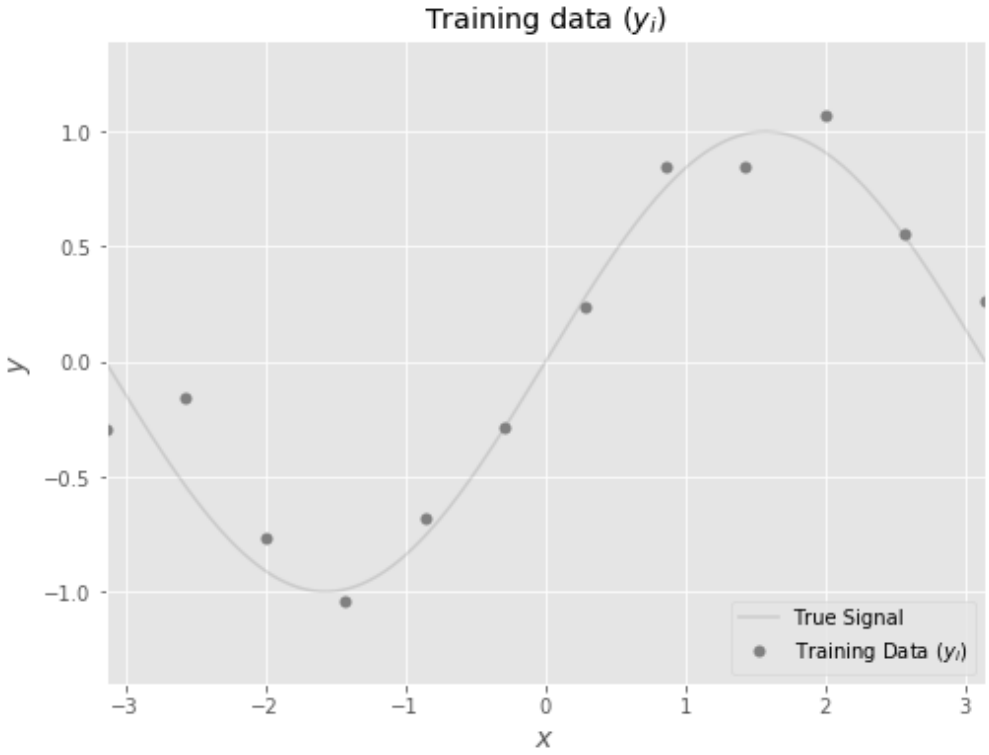

Figure 5:Training data

ϕ_0

We want to make the simplest possible choice for our first stage ϕ_0

$$\phi_0(x) = \textit{constant}$$

ϕ_0

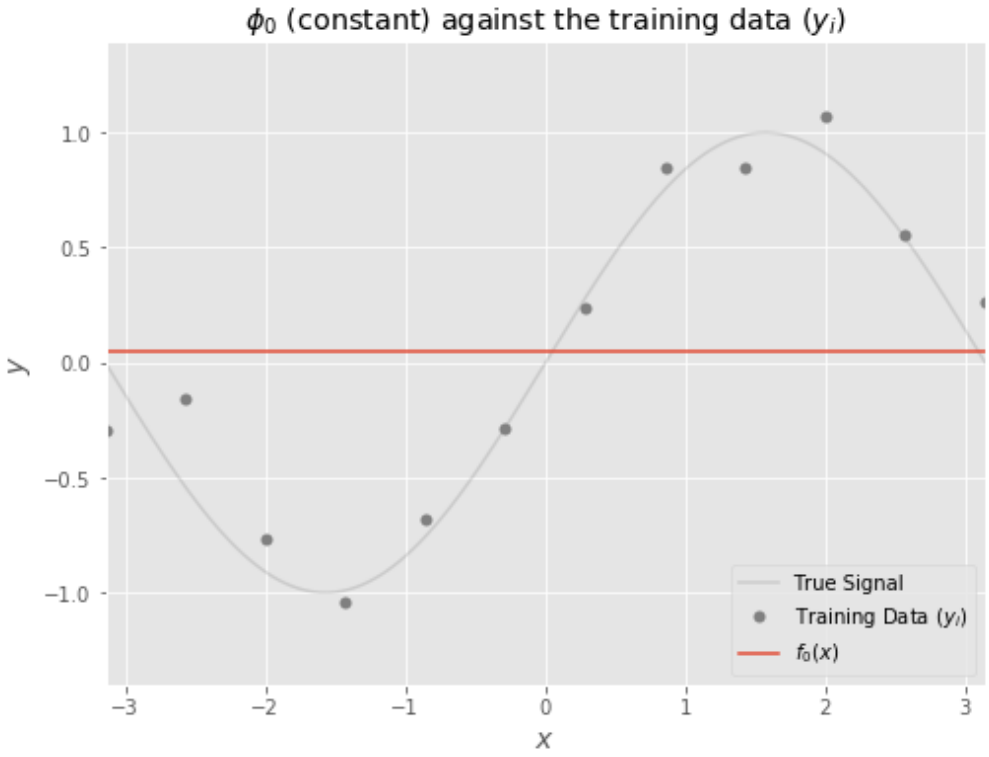

Figure 6:ϕ_0 (constant)

G_0

At this point our model is

$$G_0(x) = \phi_0(x)$$

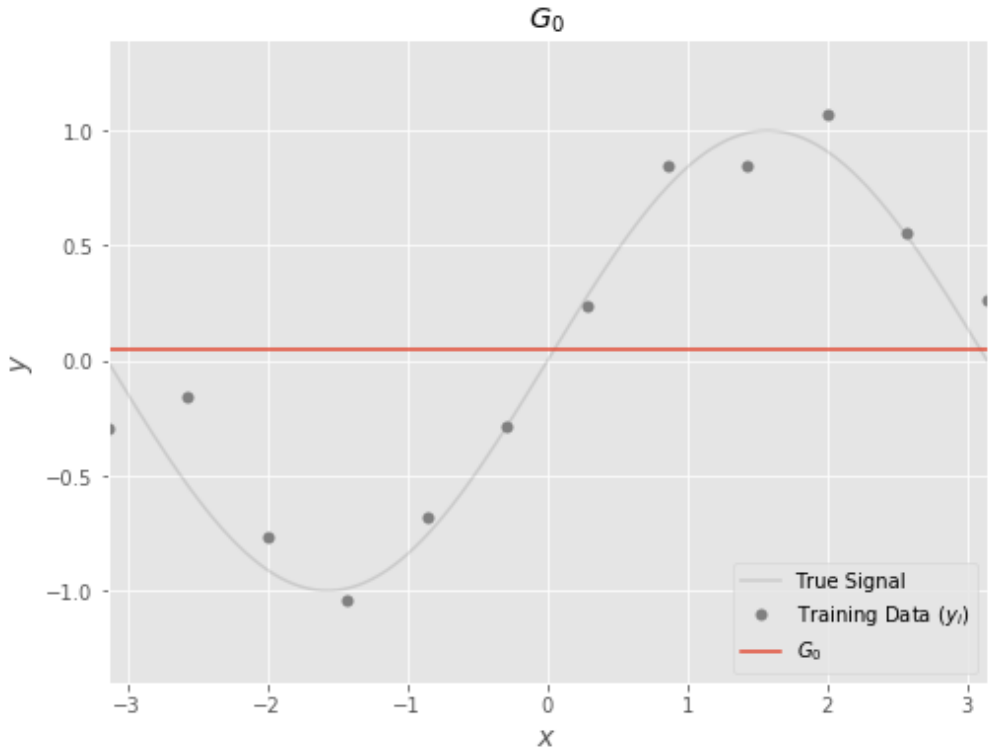

Figure 7: G_0

$y_i - G_0(x_i)$

Answer: We are off by $y_i - G_0(x_i)$ (the **residuals**)

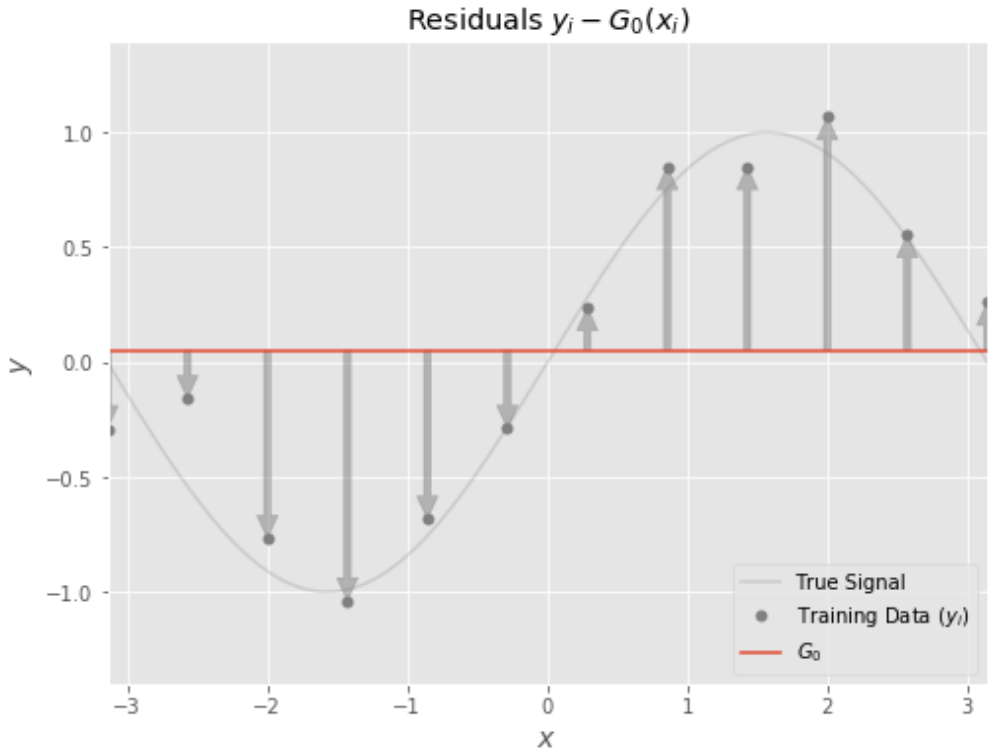

Figure 8:Residuals: $y_i - G_0(x_i)$

Question: How should we proceed next?

How should we proceed next?

Answer: Fit a model to the residuals

This is a new **version** of the training data:

- x_i are the same as before,
- but the response values y_i become the **residuals**: $y_i \leftarrow y_i - G_0(x_i)$

New version of the training data: $y_i - G_0(x_i)$

Figure 9:New version of the training data: $y_i - G_0(x_i)$

ϕ_1

Here's a very simple model fit to this new version of the training set

Figure 10:ϕ_1 (simple tree)

G_1

Now, we can update the model:

$$G_1(x) = G_0(x) + \phi_1(x) = \phi_0(x) + \phi_1(x)$$

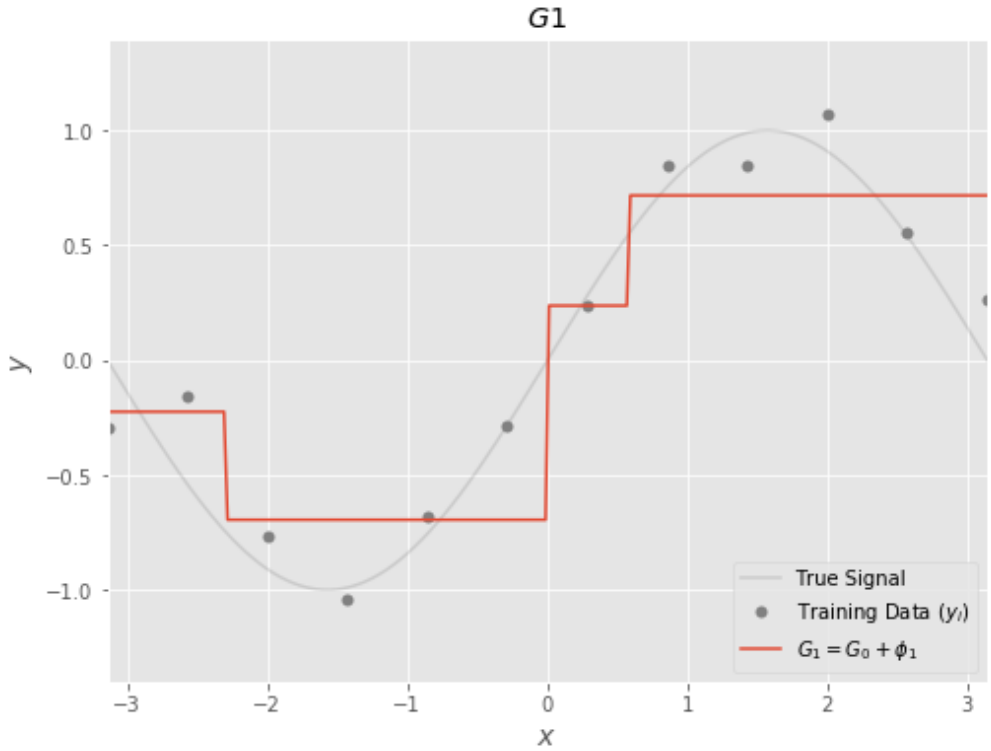

Figure 11: G_1

Let's keep going...

Let's go one more step so the idea is clear

$y_i - G_1(x_i)$

We are off by $y_i - G_1(x_i)$ (the **residuals**)

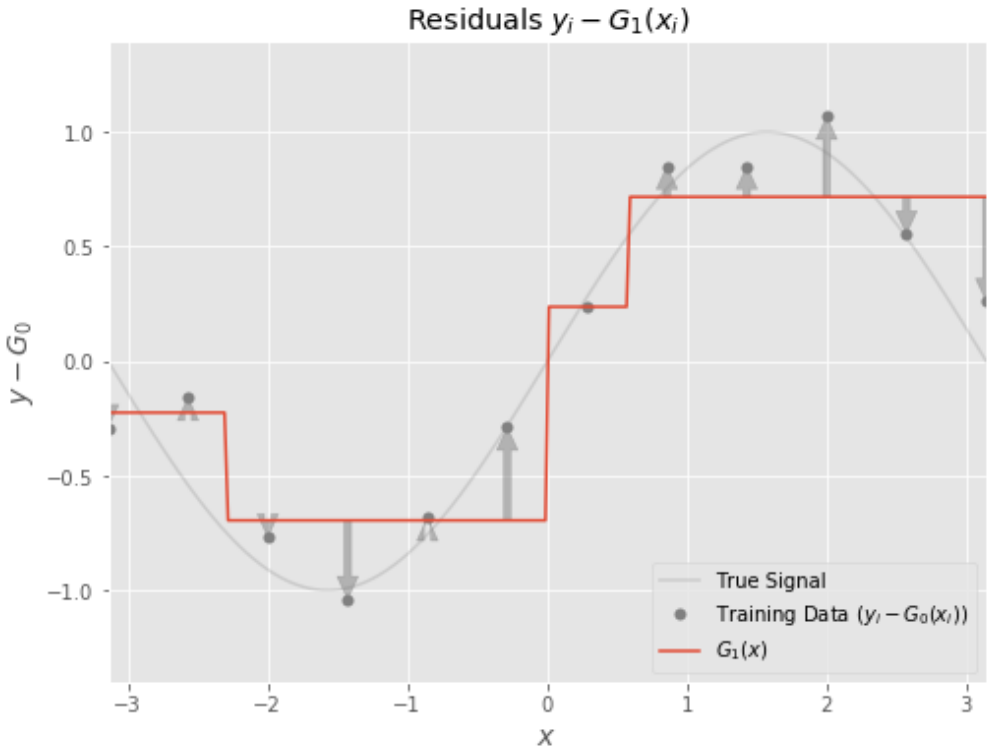

Figure 12:Residuals: $y_i - G_1(x_i)$

New version of the training data

This is a new **version** of the training data:

- x_i are the same as before,
- but the response values y_i become the **residuals**: $y_i \leftarrow y_i - G_1(x_i)$

Figure 13:New version of the training data: $y_i - G_1(x_i)$

ϕ_2

Here's a very simple model fit to this new version of the training set

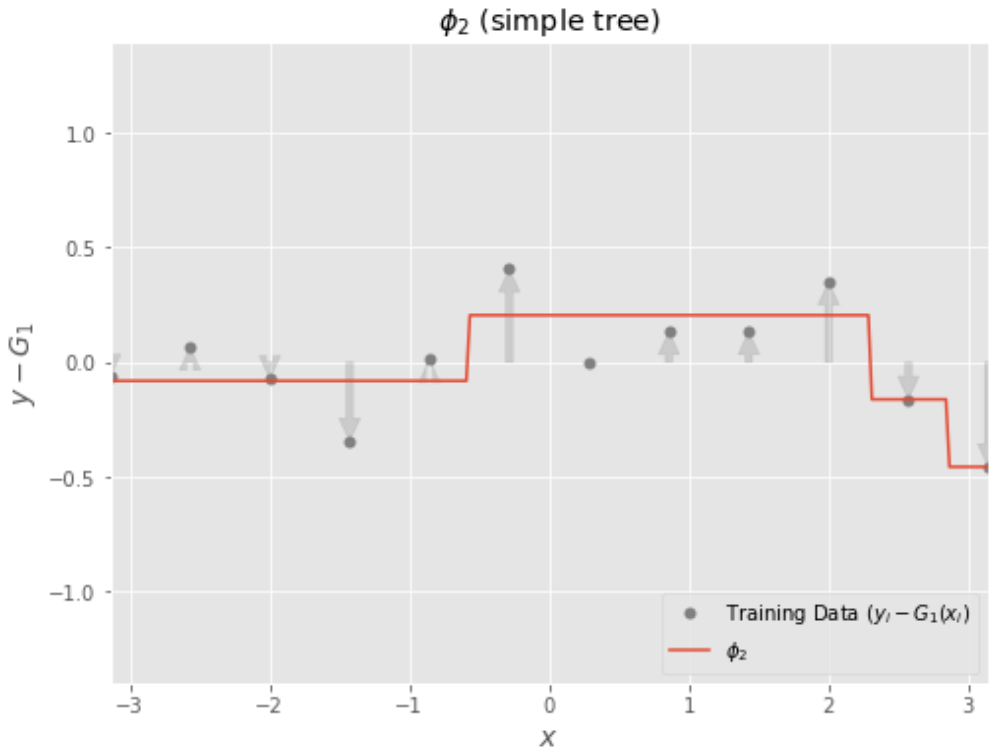

Figure 14:ϕ_2 (simple tree)

G_2

Now, update the model:

$$G_2(x) = G_1(x) + \phi_2(x) = \phi_0(x) + \phi_1(x) + \phi_2(x)$$

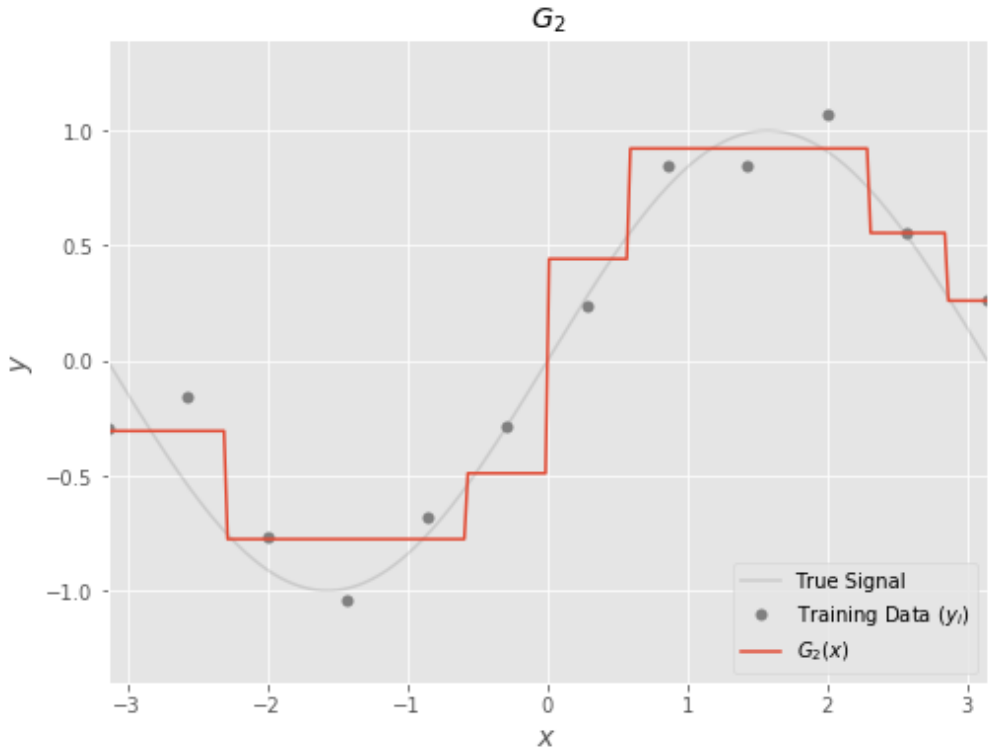

Figure 15: G_2

Boosting Over Time

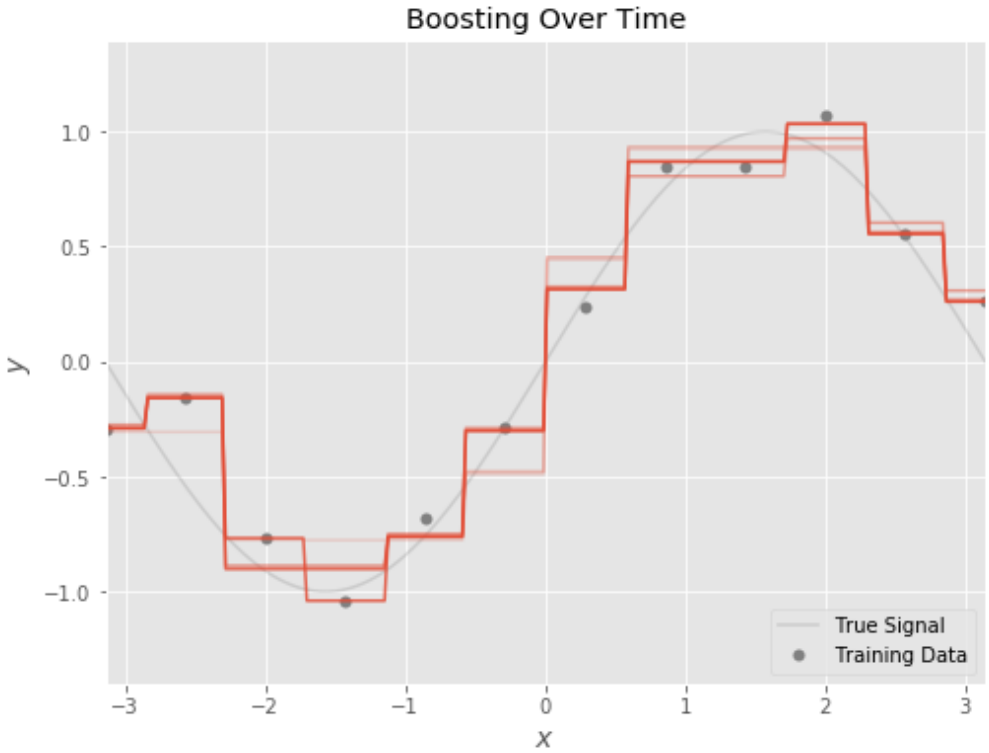

Figure 16:Boosting Over Time

But, What Happened to Growing Slowly Over Time?

Instead of adding in the entirety of the residual fitted tree during an update, $G_{k+1}(x) = G_k(x) + \phi_{k+1}(x)$, we add some *small fraction* of ϕ_{k+1}: $G_{k+1}(x) = G_k(x) + \alpha\phi_{k+1}(x)$

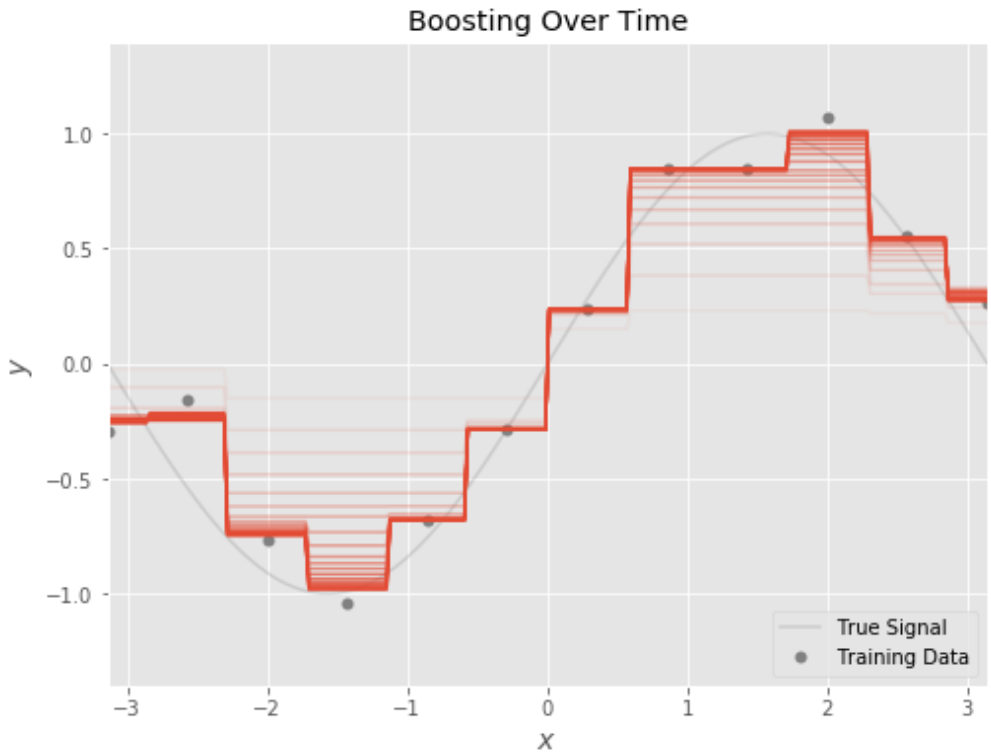

Figure 17:Boosting Slowly Over Time

Boosting - Recap

The motivation for boosting was to build an algorithm that combined the outputs of many **weak learners** to produce a powerful committee

- A **weak learner** is defined as an algorithm (model) whose error rate is only slightly better than we would achieve through intelligent random guessing (think **high bias, low variance**)

Figure 18:Random Guessing; because why not?

Boosting - Recap

We end up fitting a **sequence of weak learners**, where each **weak learner** builds on the previous one:

$$G_k(X) = G_{k-1}(X) + \alpha_k \phi_k(X, \gamma_k)$$

Notation:

- G_k, the boosted model after k steps
- ϕ_k, the weak learner trained at step k
- α_k, a weight that we apply to each of our weak learners
- γ_k denotes the hyper-parameters of whatever weak learner we are using (e.g., maximum depth, minimum samples per leaf for a decision tree)

Ensemble Fitting - Bagging and Random Forests Review

With **bagged trees**, we fit many deep trees (**low bias, high variance**) and then average them in the hopes that we could decrease the variance

- Outperforms an individual decision tree, but doesn't give us as good a decrease in variance as we would like
- With **random forests**, we still fit many deep trees, but take a random subset of the input variables at each split in order to **decorrelate** the trees
- Outperforms individual decision trees and bagged trees, as we're able to more readily leverage a decrease in variance when averaging the trees

In both cases, each of the trees are fit independently of each other (e.g. the fitting of one tree doesn't affect the fitting of another tree, and there is no **sequential** nature to it)

Ensemble Fitting - Boosting

With **boosting**, we fit the trees in a **sequential** manner, where each tree is dependent upon the last

Each subsequent tree is able to kind of focus on where the previous tree didn't do so well

Remember that we want to use the residuals to update the model:

$$G_{k+1}(x_i) = G_k(x_i) + \alpha(y_i - G_k(x_i))$$

Removing the details of the specific situation, this looks like

$$\xi_{k+1} = \xi_k + \alpha(y - \xi_k)$$

Question: Does this look... familiar?

Gradient Descent - Review

Recall that with **gradient descent**, we are trying to intelligently update our parameters such that we decrease our loss with each update:

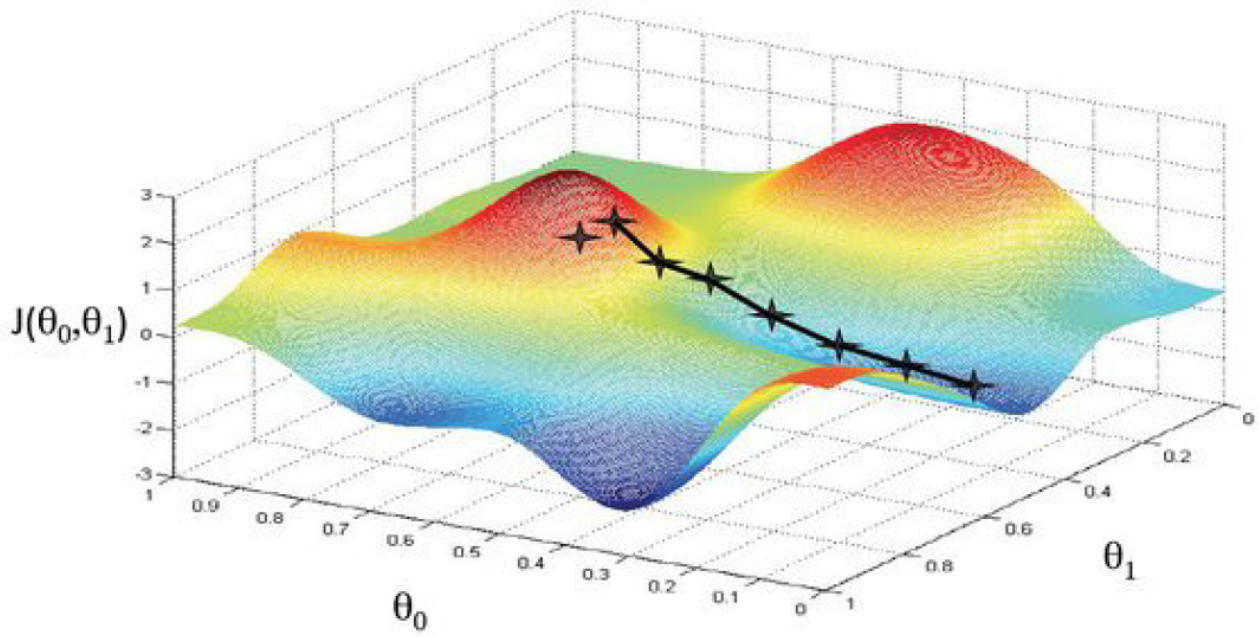

Figure 19:Gradient Descent :)

Gradient Descent Algorithm - Review

Input: A *loss* function $J(\theta)$

Output: A point θ^* that minimizes J

- Calculate analytically $\nabla_\theta J(\theta)$
- Initialize $\theta_0 = 0$ (arbitrary; there may be more principled choices).
- Iterate until convergence:
  - Set $\theta_{i+1} = \theta_i - \alpha \nabla_\theta J(\theta_i)$

Gradient Descent to Minimize the Squared Error Loss

What happens when we apply gradient descent to minimize the squared error loss?

$$J(\xi, y) = \frac{1}{2}(y - \xi)^2$$

$$\nabla_\xi J(\xi, y) = -\frac{1}{2}\frac{\partial}{\partial \xi}(y - \xi)^2 = y - \xi$$

So gradient descent for least squares looks like:

$$\xi_{k+1} = \xi_k + \alpha(y - \xi_k)$$

Gradient Boosting

$$\xi_{k+1} = \xi_k + \alpha(y - \xi_k)$$

That is, **least squares gradient descent iteratively moves in the direction of the residual**

This is why this algorithm is called **Gradient Boosted Regression**

- We "boost" the current estimate by adding an **approximation to the gradient of the squared error loss function**

Gradient Boosting - Recap

We fit a weak learner (here a tree) in a **sequential manner**, where each one (except the first) is fit on the gradient of the loss function from the previous one

We fit each model sequentially:

$$G_m(X) = G_{m-1}(X) + \alpha_m \phi(X, \gamma_m)$$

Gradient Boosting - Algorithm

Gradient boosting algorithm

- **Inputs**: A training dataset x_i, y_i and a learning rate α
- **Output**: A function f such that $f(x_i) \approx y_i$

1. Initialize $G_0(x) = \phi_0(x) = \frac{1}{N} \sum_{i=1}^{N} y_i$
2. For $k = 1$ to M, do:
   - 2.1 Create a working training dataset $W_k = x_i, y_i - G_k(x_i)$
   - 2.2 Fit a regression tree ϕ_k, minimizing the least squares
   - 2.3 Set $G_k(x) = G_{k-1}(x) + \alpha\phi_k(x)$
3. Return $f(x) = G_M(x)$

Gradient Boosting - Revisiting Examples

In our testing scenario earlier, the "gradient" would be the knowledge of what questions the previous tester got right or wrong

Figure 20:Testing Again!?

In our stock market example, the "gradient" would be the knowledge of how far off the previous person's predictions were for each stock

Figure 21:$$$

Gradient Boosted Trees - Hyper-parameters

`GradientBoostingRegressor` has **many knobs** to turn:

```
GradientBoostingRegressor(
    loss='ls',

    n_estimators=100,
    learning_rate=0.1,
    max_depth=3,
    subsample=1.0,

    min_samples_split=2,
    min_samples_leaf=1,
    min_weight_fraction_leaf=0.0,
    ...)
```

Gradient Boosted Trees - Hyper-parameters

The most important knobs to `GradientBoostedRegressor` are

- `loss` controls the loss function to minimize. `ls` is the least squares minimization we just discussed
- `n_estimators` is how many boosting stages to compute, i.e., how many regression trees to grow
- `learning_rate` is the learning rate for the gradient update
- `max_depth` controls how deep to grow each individual tree
- `subsample` allows to fit each tree on a random sample of the training data (similar to bagging in random forests)

When dealing with these hyper-parameters, keep in mind that we want to make **weak learners**, i.e., we want trees that are **high bias, low variance**...

Gradient Boosted Trees - `n_estimators` (M)

Too many trees leads to overfitting

One way to tune the number of trees is to make use of a validation set, held out from training the model

Another way is to use cross-validation. We then generally choose the number of trees minimizing the average out validation fold error

Gradient Boosted Trees - `learning_rate` (α)

The learning rate allows us to grow our boosted model slowly

A large learning rate will cause the model to fit hard to the training data, which creates a **high variance** situation

On the other end, a smaller learning rate reduces the boosted models sensitivity to the training data

- A smaller learning rate means more trees are needed to reach the optimal point but rate eventually (i.e., given enough time) results in a superior model

Gradient Boosted Trees - `learning_rate` (α)

Strategy for Tuning the Learning Rate

- In the initial exploratory phases of modeling, set the learning rate to some large value, say .1. This allows you to iterate through ideas quickly
- When tuning other parameters using grid search, decrease the learning rate to a more sensible value; .01 works well
- When fitting the *final* production model, set the learning rate to a very small value, .001 or .0005; smaller is better

General Advice

- Run the final model overnight! It will fit while you are sleeping!
- Make sure you've written all your analysis code up front, based on your initial models. Then all that's left is to run your final model through and see your final plots and statistics.

Gradient Boosted Trees - `max_depth`

A deeper tree depth allows the model to capture deeper interactions between the predictors, resulting in lower bias but it also

- Causes the model to fit faster, increasing the variance and somewhat combating the effect of the learning rate
- Allows the model to assume a more complex for at the same number of trees. This is a blessing (low bias) and curse (high variance)

It's never obvious up front what tree depth is best for a given problem, so a grid search is needed to determine the best value

Strategy for Tuning Tree Depth

- Tune with a grid search and cross validation

Gradient Boosted Trees - `subsample`

The `subsample` parameter allows to train each tree on a subsample of the training data. This is similar to bagging in the random forest algorithm, and has the same result: it lowers the variance of the resulting model

Not subsampling at all, or over subsampling are both bad ideas

- Between these two extremes, different levels of subsampling generally give the same performance

Strategy For Tuning Subsample

- Set to .5, it almost always works well
- If you have a massive amount of data and want the model to fit more quickly, decrease this value
- The default rate in sklearn is 1.0, so make sure you **always** change it

Gradient Boosted Trees - Tuning Other Parameters

The other parameters to `GradientBoostingRegressor` are less important, but can be tuned with grid search for additional improvements in importance

- `min_samples_split`: Any node with less samples than this will not be considered for splitting
- `min_samples_leaf`: All leaf nodes must contain more samples than this
- `min_weight_fraction_leaf`: Same as above but a expressed as a fraction of the total number of training samples

Generally these are less important because **you shouldn't be growing super gigantic trees!**

Gradient Boosted Trees - Tuning Other Parameters

If you decide to include these in a grid search, **be wary of overfitting to your validation set**

Also, the number of model comparisons grows **exponentially** in the number of parameters tuned

Gradient Boosted Trees - Advantages

Gradient boosted trees often outperform most (if not all) other models

Figure 22:Happy Dance

Gradient Boosted Trees - Disadvantages

Typically, to get the level of performance that gradient boosted trees offer, fairly extensive tuning is required

They also aren't parallelizeable, since they are fit **sequentially**

Figure 23:So much less time...