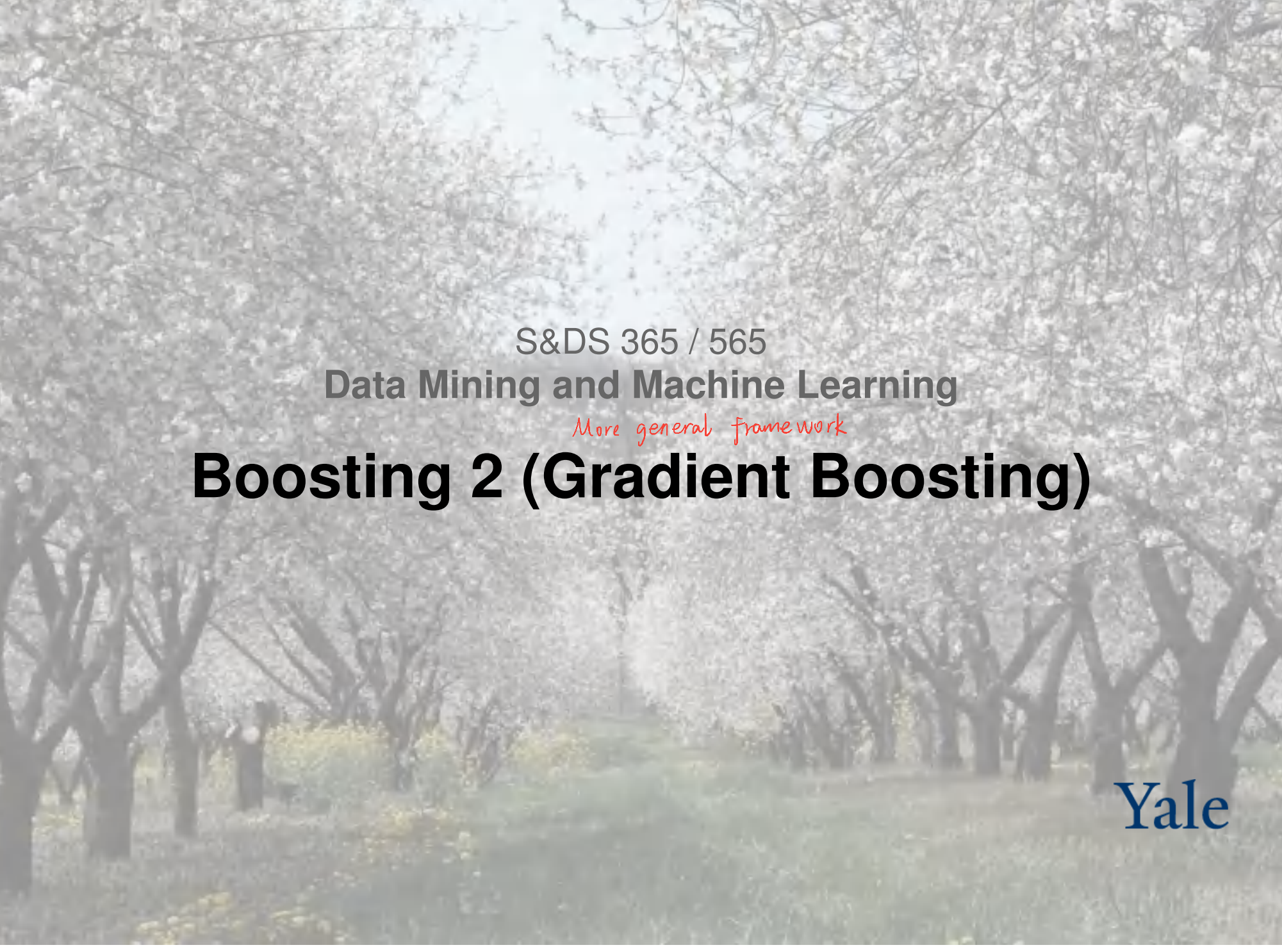


S&DS 365 / 565

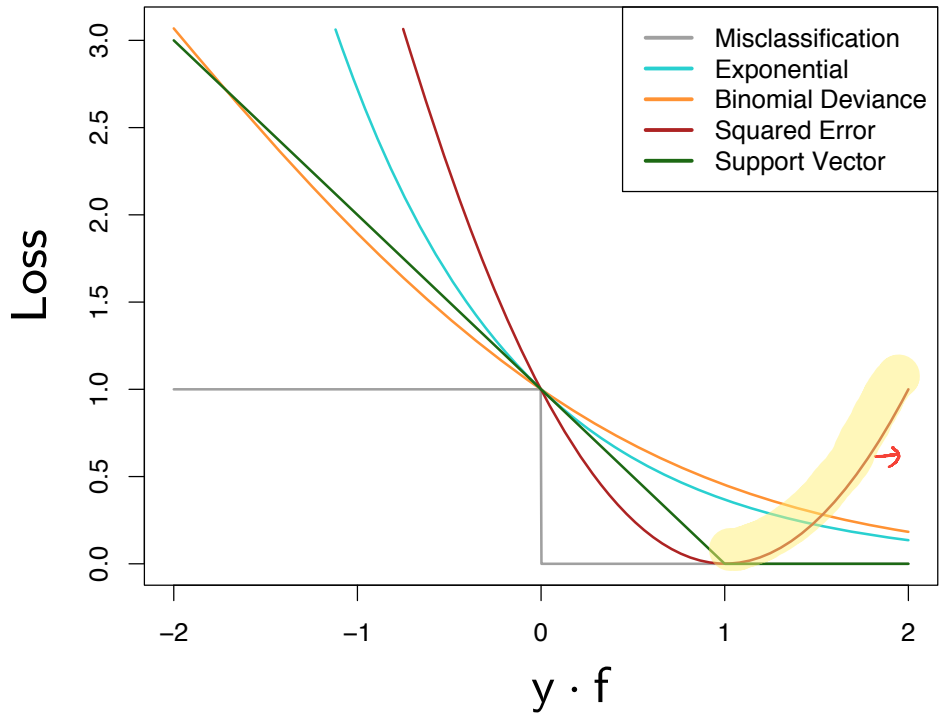
Data Mining and Machine Learning

More general framework

Boosting 2 (Gradient Boosting)

Yale

Loss functions



problem with square error
if you are very correct
loss are very large
use left-side square error instead

FIGURE 10.4. *Loss functions for two-class classification. The response is* $y = \pm 1$*; the prediction is* f*, with class prediction* $\text{sign}(f)$*. The losses are misclassification:* $I(\text{sign}(f) \neq y)$*; exponential:* $\exp(-yf)$*; binomial deviance:* $\log(1 + \exp(-2yf))$*; squared error:* $(y - f)^2$*; and support vector:* $(1 - yf)_+$ *(see Section 12.3). Each function has been scaled so that it passes through the point* $(0, 1)$*.*

Gradient Boosting

Like AdaBoost, Gradient Boosting iteratively combines weak learners to produce a strong learner. They differ on how they create weak learners.

- AdaBoost changes the weights of the data to create different weak learners. because of structure of exponential loss
- Gradient Boosting considers the "residuals". because of squared loss

Gradient Boosting

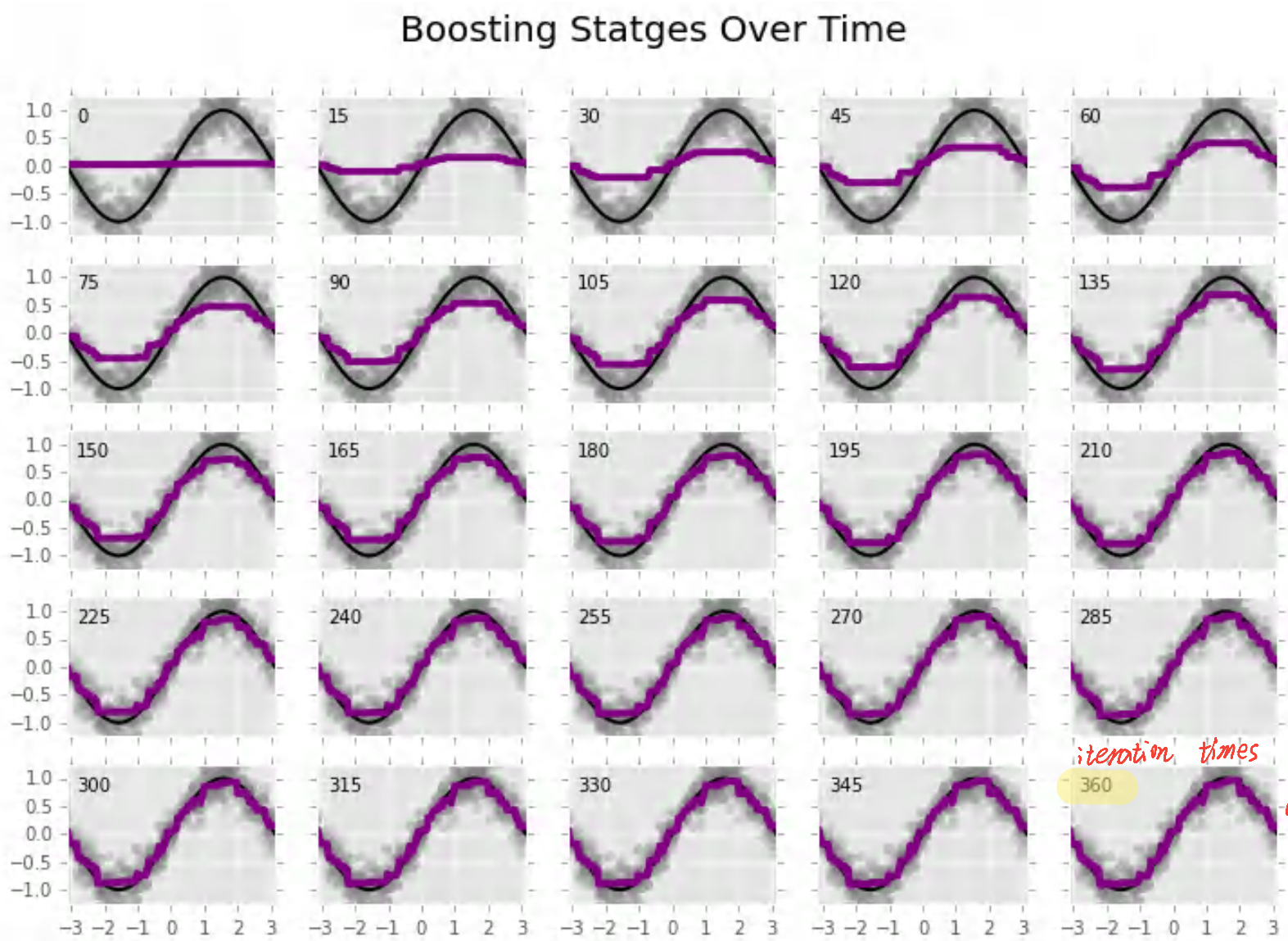
Setup: $\{x_i, y_i\}_{i=1}^n$, regression setting.

Goal: find f that minimizes the squared loss:

$$\arg\min_f \sum_i (y_i - f(x))^2$$

We would like to build up f in stages, making small adjustments as we go along.

紫：$\hat{f}$ 黑：true f



In the end, f will be a sum of *weak* learners

(decision stumps)

$$f(x) = f_0(x) + f_1(x) + f_2(x) + \cdots + f_M(x)$$

The process of building up the model looks like

$$\begin{aligned}
S_0(x) &= f_0(x) \\
S_1(x) &= f_0(x) + f_1(x) \\
S_2(x) &= f_0(x) + f_1(x) + f_2(x) \\
&\vdots \\
S_M(x) &= f_0(x) + f_1(x) + f_2(x) + \cdots + f_M(x)
\end{aligned}$$

Where Should We Start?

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

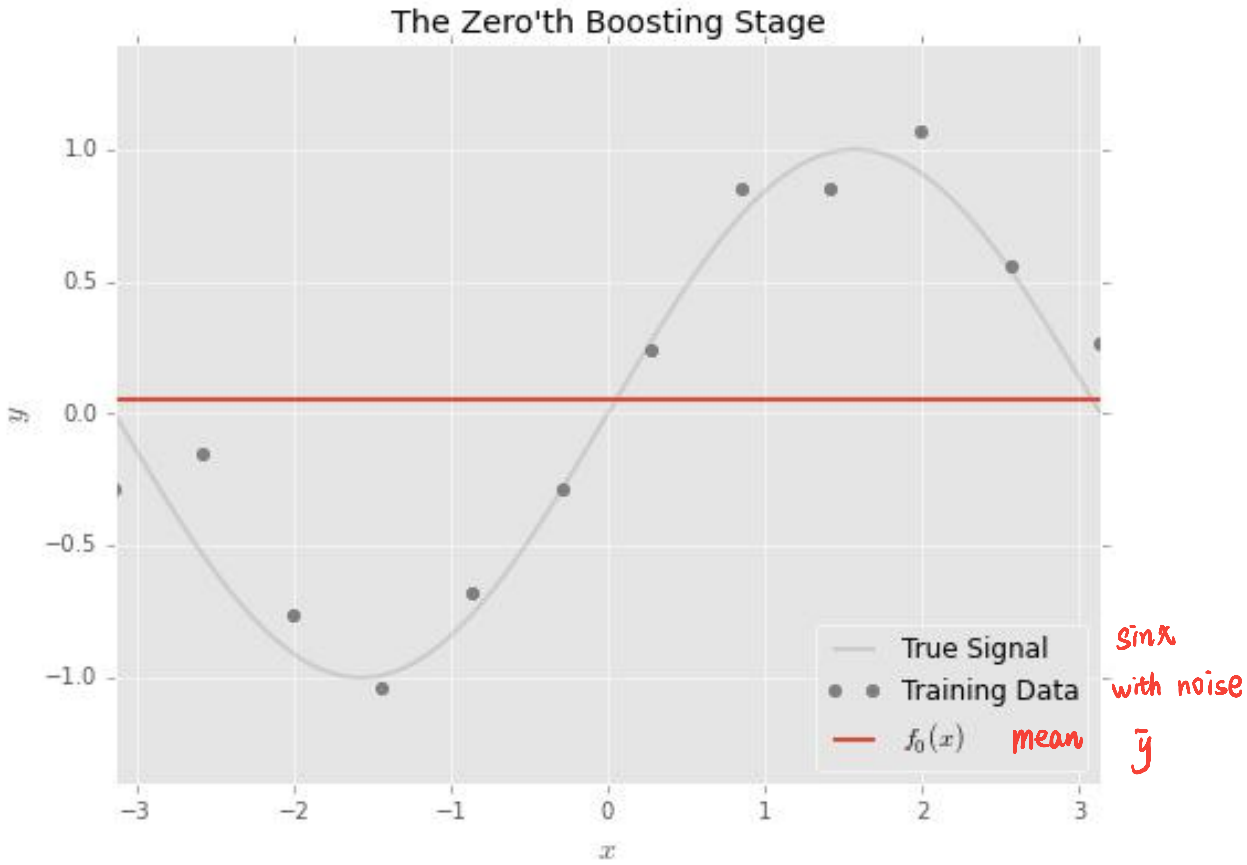
$$f_0(x) = \text{constant}$$

A natural choice is to pick the mean of our responses. That is,

$$f_0(x) = \overline{y}.$$

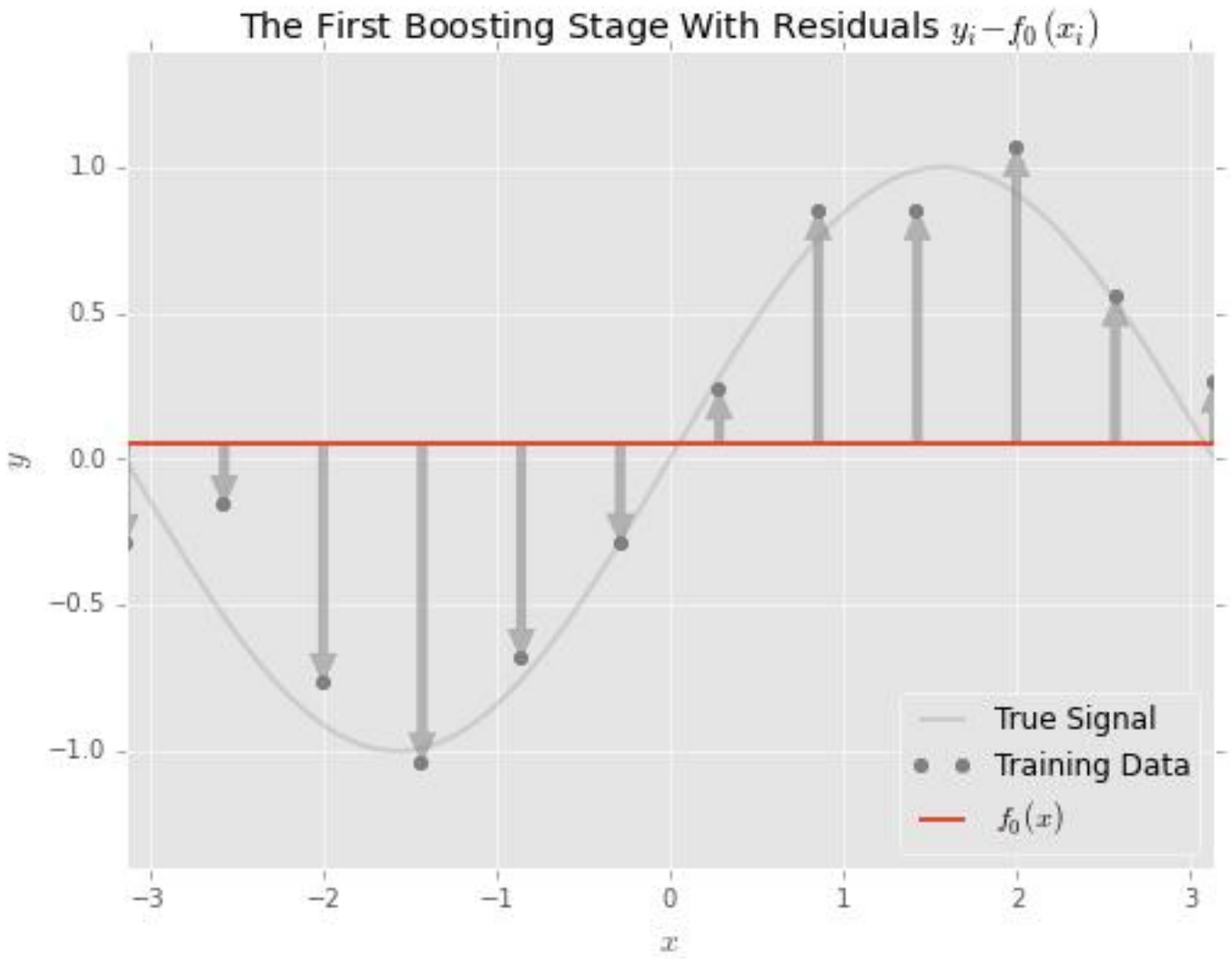
How To Update

We are now in the situation illustrated below



Our next task is to update our simple f_0 to $S_1(x) = f_0(x) + f_1(x)$.

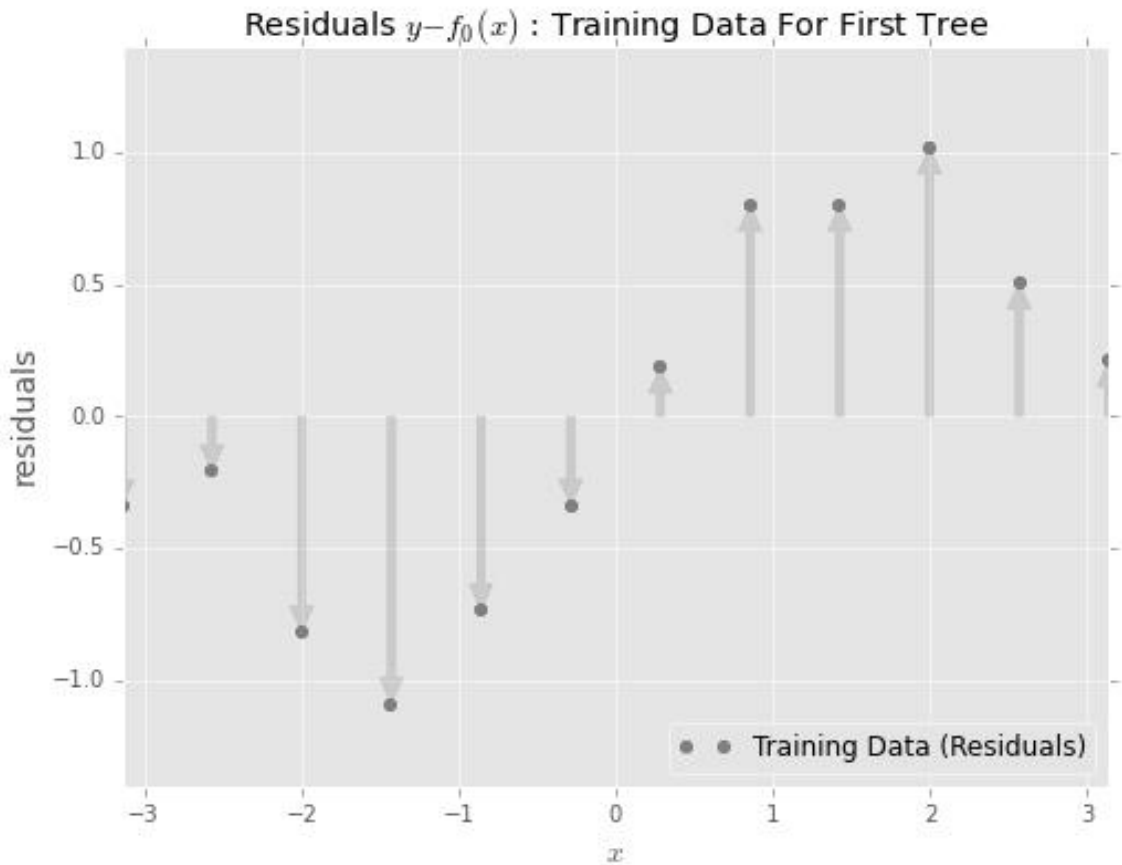
How to pick f_1? The *residuals*, $y_i - f_0(x_i)$ answer this question



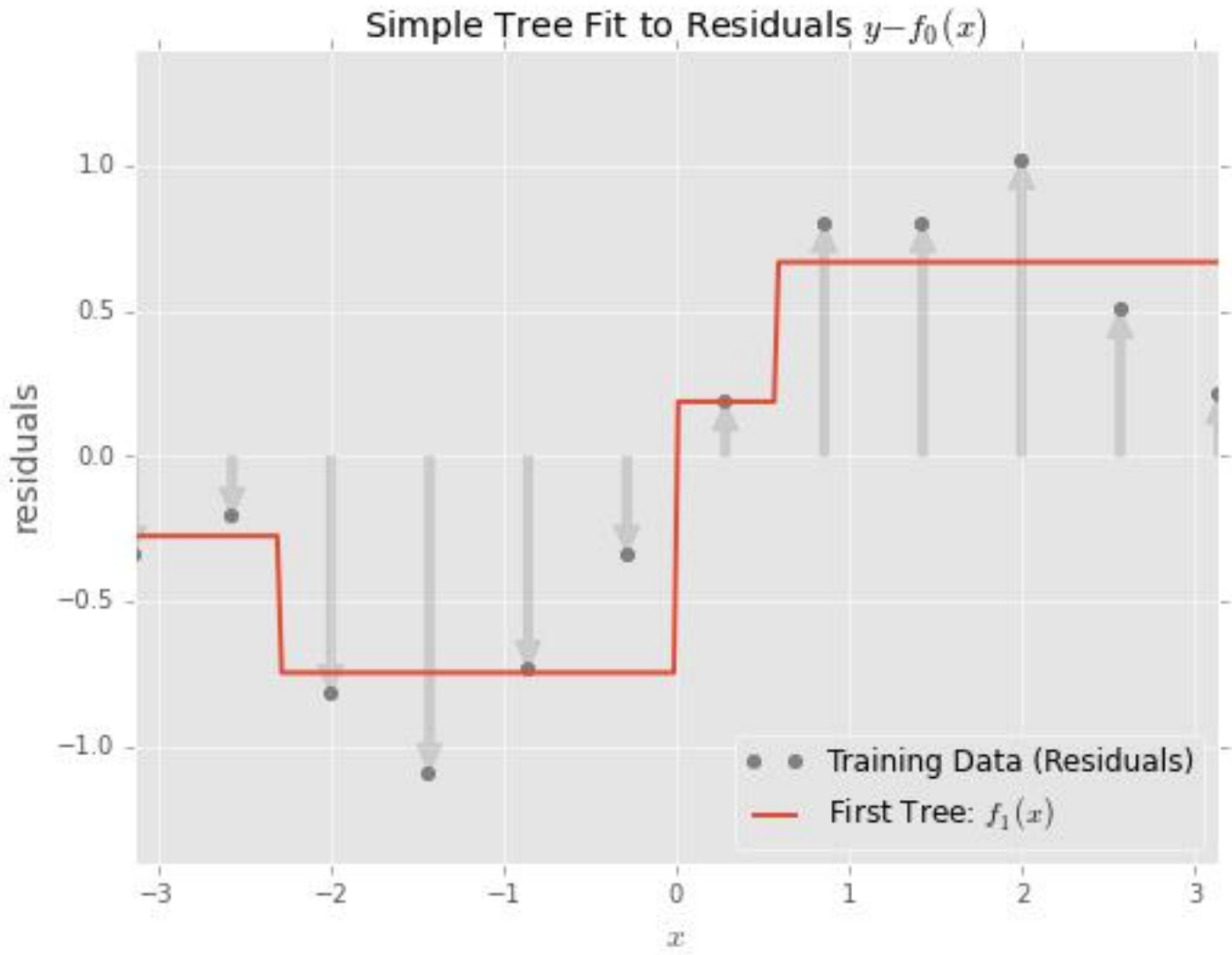
Here is a *new* dataset.

- The values of x are the same as before, taken directly from the training data.
- The response values are the *residuals*: $y_{\text{new}} = y_i - f_0(x_i)$.

$y_{\text{new},i}$

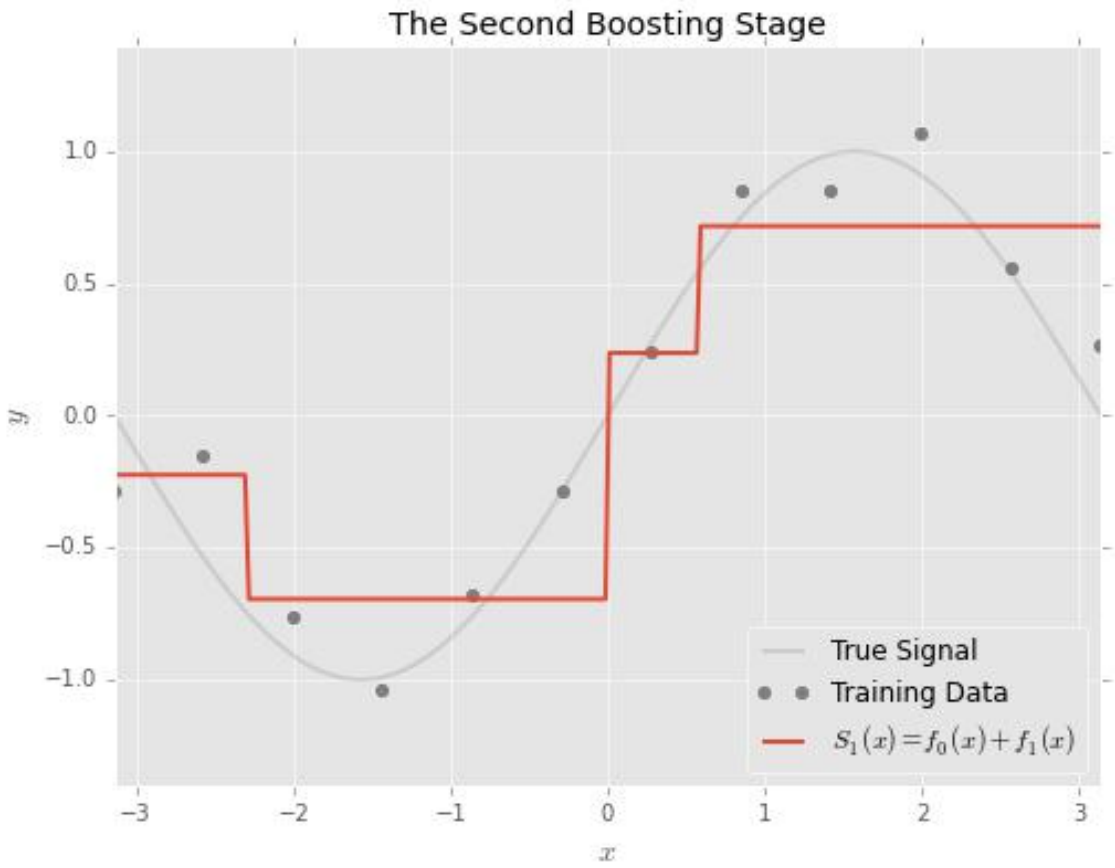


Here is a very simple model fit to this *working data set*



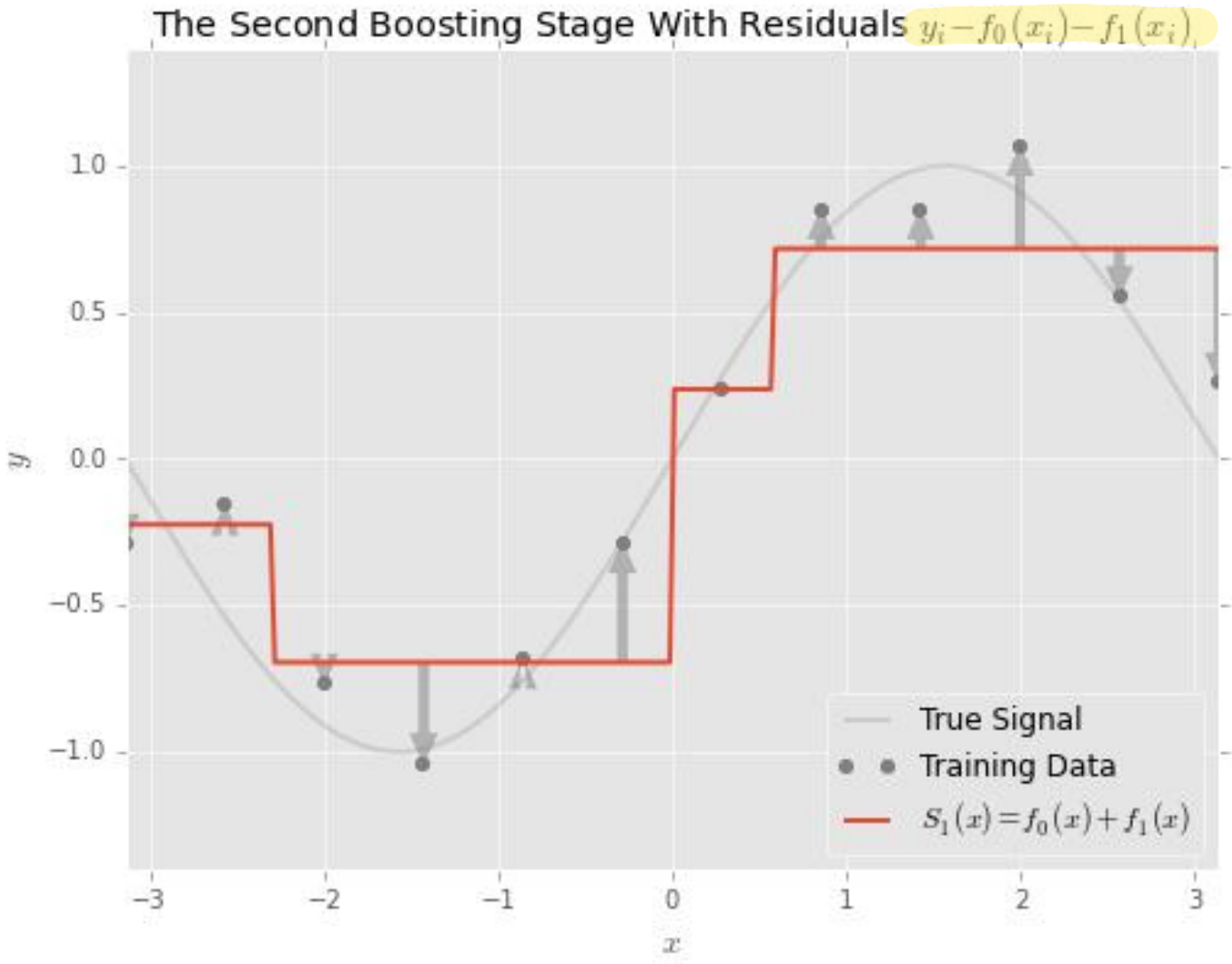
Now we can update the model.

$$S_1(x) = f_0(x) + f_1(x) \leftarrow \text{Model fit to residuals!}$$

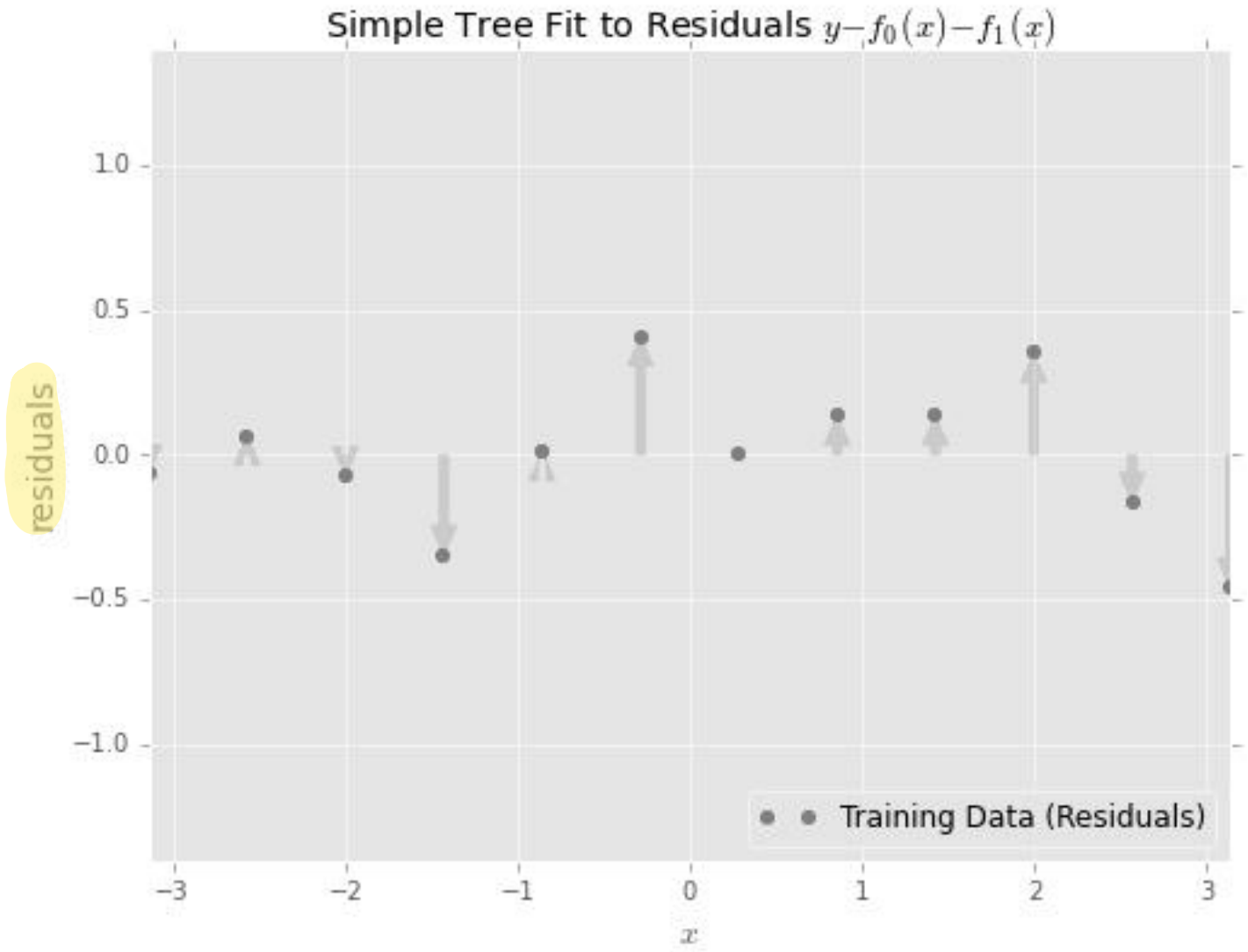


Let’s go one more step, so that the idea is clear.

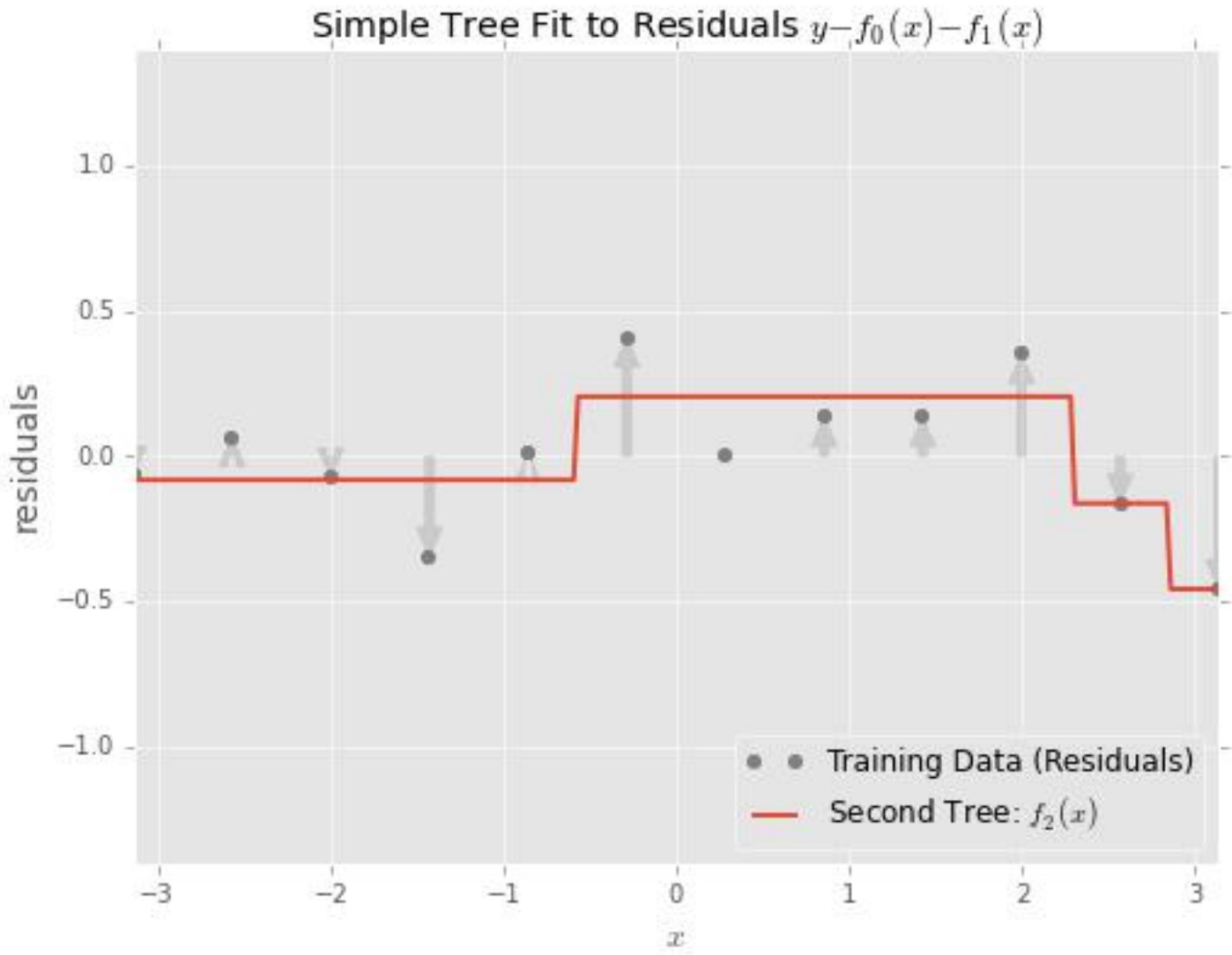
Calculate the residuals of the current model...



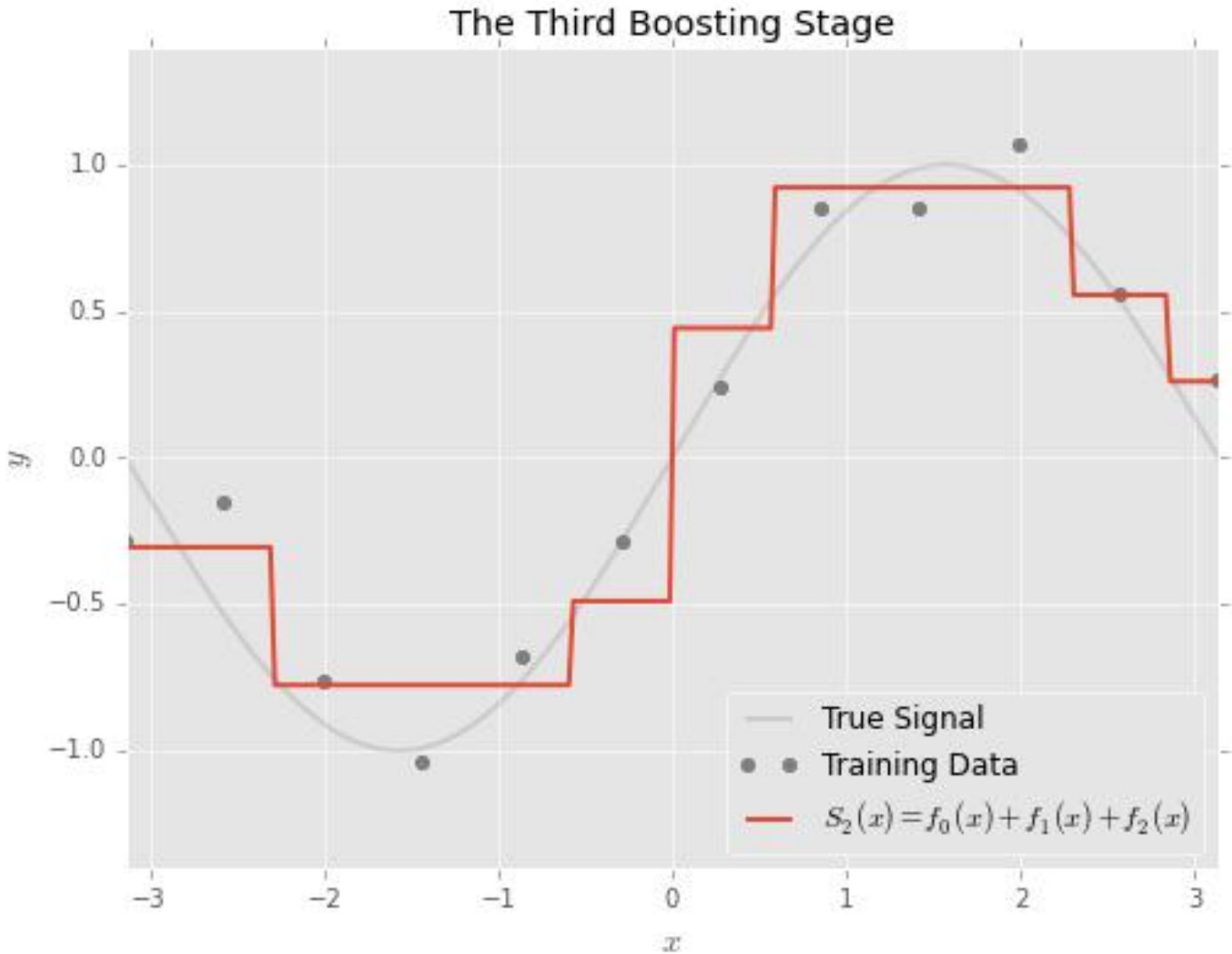
Create a training data set with the residuals as the response...

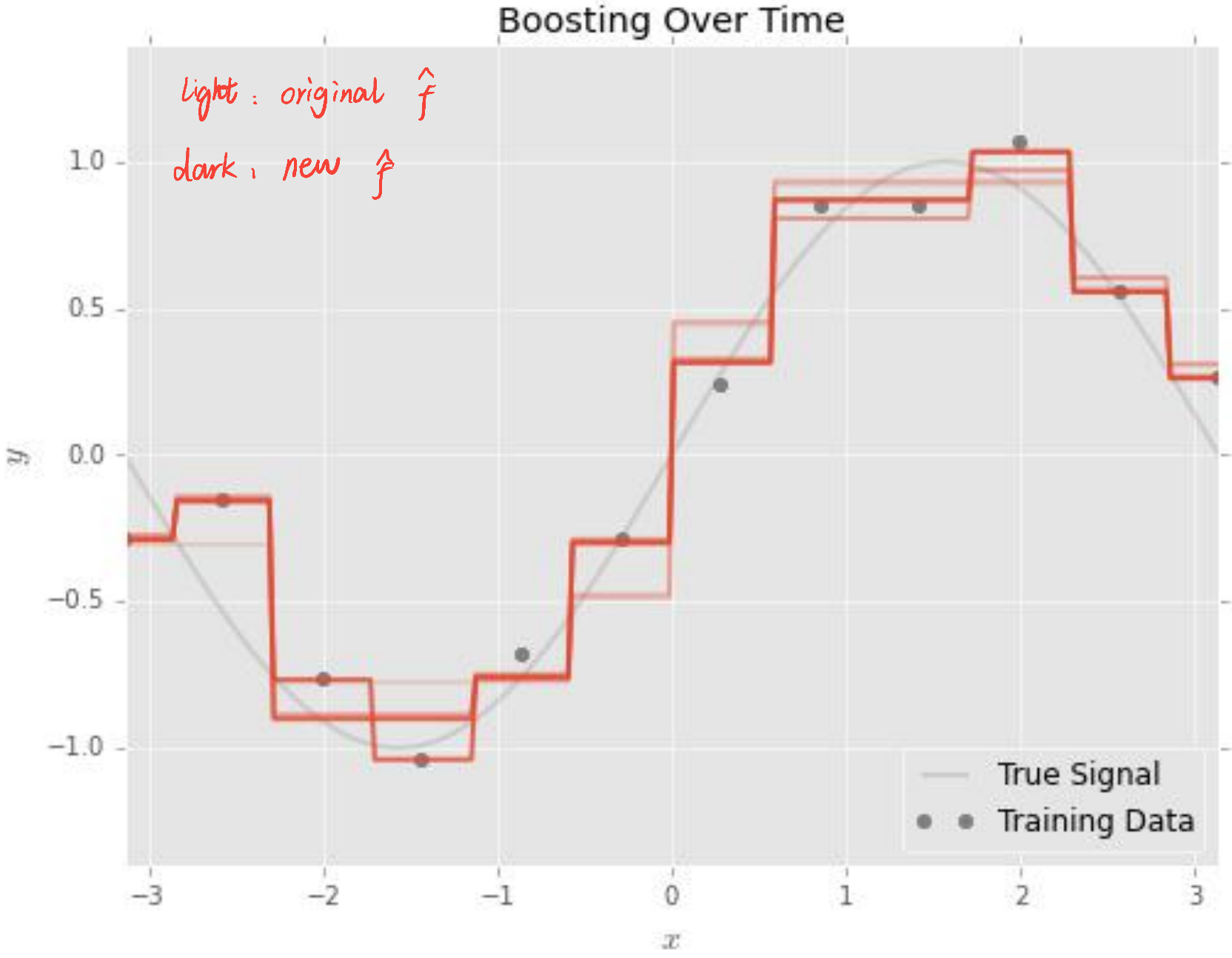


Create a training data set with the residuals as the response...



Update the model!




Boosting Over Time
light : original $\hat{f}$
dark : new $\hat{f}$
1.0
0.5
0.0
−0.5
−1.0
y
−3
−2
−1
0
1
2
3
x
True Signal
Training Data

But, What Happened to Growing Slowly Over Time?

Instead of adding in the entirety of the residual fitted tree during an update

$$S_{m+1}(x) = S_m(x) + f_{m+1}(x)$$

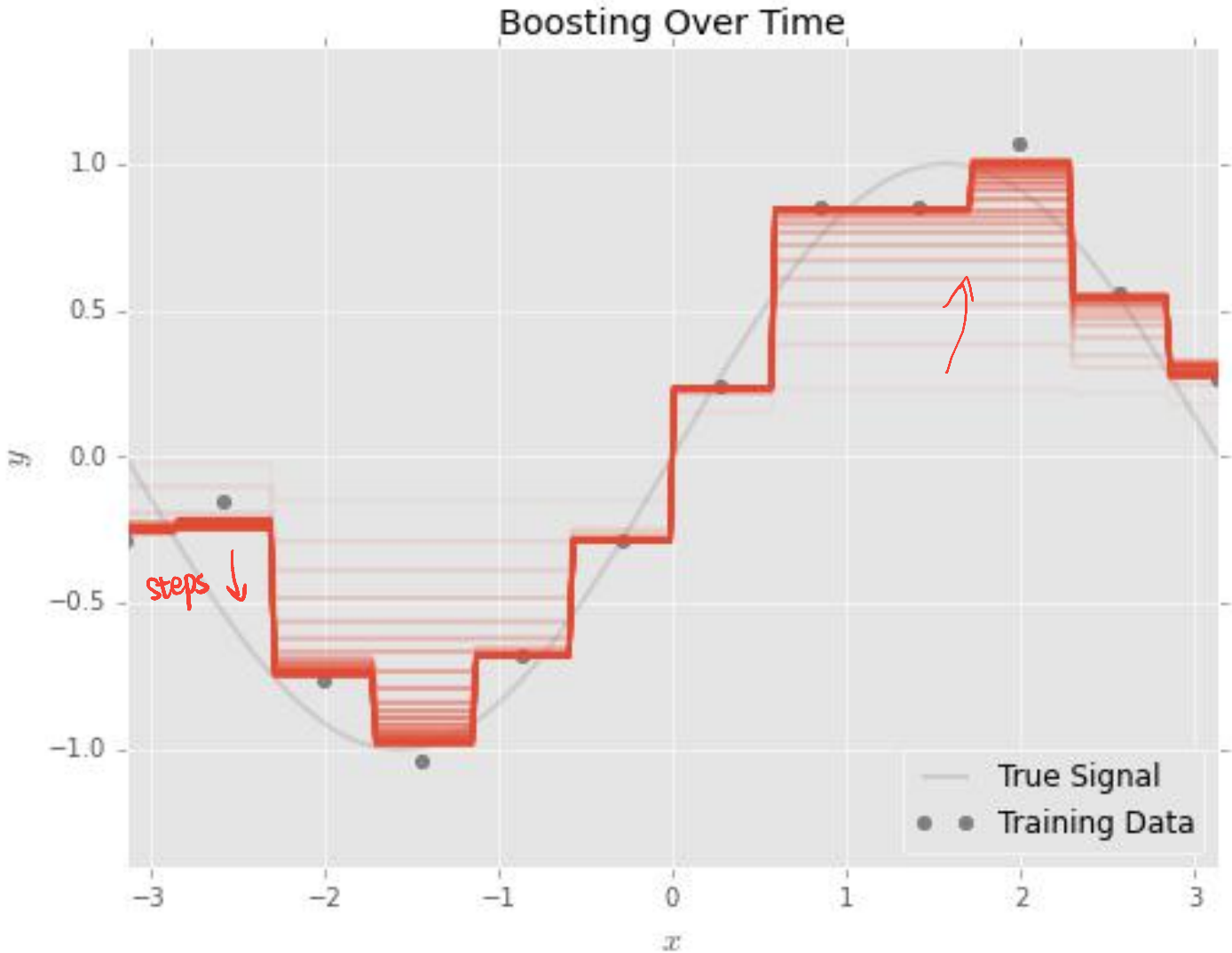
Instead we added some *small fraction* of f_{m+1}

Like Gradient Descent

$$S_{m+1}(x) = S_m(x) + \lambda f_{m+1}(x)$$

↑ Learning Rate (step-size)

$\lambda = 0.01$


Boosting Over Time
1.0
0.5
0.0
−0.5
−1.0
y
steps
True Signal
Training Data
−3
−2
−1
0
1
2
3
x

Gradient Boosting to Minimize Sum of Squared Errors

mean

- Initialize $S_0(x) = f_0(x) = \frac{1}{N}\sum_i y_i$.
- Iterate (m) until satisfied:
  - Create the working data set $W_m = \{x_i, y_i - S_m(x_i)\}$.
  - Fit a regression tree to W_m, minimizing least squares. Call this tree f_m.
  - Set $S_{m+1}(x) = S_m(x) + \lambda f_m(x)$.
- Return $S_M(x) = \lambda f_0(x) + \lambda f_1(x) + \cdots + \lambda f_M(x)$.

residual of ith observ

$$f_m = \arg\min_f \sum_i \left[y_i - S_m(x_i) - f(x_i)\right]^2$$

Model selection: pick optimal λ

Why is it called Gradient Boosting

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

$$L(y, S_m(x)) = \frac{1}{2}(y - S_m(x))^2$$

Why is it called Gradient Boosting

neg Gradient of squared loss is just residual

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

$$L(y, S_m(x)) = \frac{1}{2}(y - S_m(x))^2$$

Easier to rewrite it as

$$L(y, \hat{y}) = \frac{1}{2}(y - \hat{y})^2$$

to know what's direction $\hat{y}$ should go to minimize loss
so compute gradient of Loss with respect to $\hat{y}$ → direction is residual

$$-\nabla_{\hat{y}} L(y, \hat{y}) = -\frac{1}{2}\frac{\partial}{\partial \hat{y}}(y - \hat{y})^2 = y - \hat{y} = y - S_m(x)$$

Why is it called Gradient Boosting

For regression with square loss:

$$\text{residual} \iff \text{negative gradient}$$
$$\text{fit model to residual} \iff \text{fit model to negative gradient}$$

So, in some loose sense, we're fitting our model using *functional* gradient descent.

with respect to $\hat{y}$ rather than param

Gradients turn out to be a better paradigm, and allow us to generalize to other loss functions[1].

[1] We might want loss functions that are more robust to outliers, like the Huber loss

Thus, the generalized algorithm is as follows.

- Initialize $S_0(x) = f_0(x) = \frac{1}{N}\sum_i y_i$.
- For m to 1 to M:
  - Calculate the negative gradient (or pseudo-residuals):

HW5

move $\hat{y}$ + little step × r_m can decrease loss

$$r_m = -\frac{\partial L(y, \hat{y})}{\partial \hat{y}}$$

For logit model gradient is error of label with what the expected value of label under the current form of guess params

  - Create the working data set $W_m = (x, r_m)$. Fit a model (weak learner) to W_m. That is, $f_m(x) = \arg\min_f L(r_m, f)$.

some loss

  - Pick

Sometimes minimizing $L(r_m, f)$ is computationally prohibitable instead we use $\operatorname{argmin}_f \|r_m - f\|^2$ L_2-norm of r_m and f

$$\lambda_m = \arg\min_{\lambda} L(y_i, S_{m-1}(x_i) + \lambda f_m(x_i))$$

let f to be align with r_m

sometimes just pick λ that will decrease Loss rather than minimize loss

  - Set $S_{m+1}(x) = S_m(x) + \lambda_m f_m(x)$.

do binary search until λ is sufficiently small

- Return $S_M(x)$

can do Newton Gradient Descent too

Gradient tree boosting

Here's another version: Gradient Tree Boosting, devised by Friedman. The key is to pick a λ for each leaf of the tree, and not just a λ for the whole tree:

1. Compute gradients at m-th step:

$$r_{im} = -\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}\Big|_{f(x_i)=f_m(x_i)}$$

2. Fit a regression tree to the negative gradients (or psuedo-residuals):

$$f_m = \arg\min_{f} \sum_{i=1}^{n} \Big(r_{im} - f(x_i)\Big)^2$$

Gradient tree boosting

The tree now acts like a kernel, dividing up the space into cells.

3. Fit the values at the leaves to minimize the loss function:

$$\theta_\ell = \arg\min_\theta \sum_{x_i \in \text{leaf } \ell} L(y_i, f_m(x_i) + \theta)$$

4. Update the model:

$$S_{m+1}(x) = S_m(x) + \theta_{\ell(x)}$$

where $\ell(x)$ is the leaf that x falls to.

Conclusion (trees and boosting)

- Small trees are interpretable. But need more control over bias/variance tradeoff
- Boosting combines strengths of trees and additive models
- Component functions are added in in a greedy fashion, fitting a new model on a reweighted version of the data