# Boosting

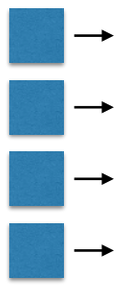
# Outline

- **Understand what Boosting (trees) is**
  - More trees, but differently
  - Learning to the ‘mistakes’
  - Combining functions into super-function
  - Parameters: B, d, $\lambda$
    - Learning rates
    - Tree shapes
- **Compare Boosting to other (tree) approaches**
  - Computation/ops
  - Bias/Variance
  - Metaphors
- **Compare types of Boosting: computation/ speed, generality, flexibility, nomenclature**
  - Learning to the residuals
  - Weighted sampling
  - Differentiable loss functions
- **Why Boosting?**
  - Interpretability
  - n << p
- **Assignment**
  - Compare AdaBoost, GradientBoosting, RandomForest regressors
  - Investigate effects of B and $\lambda$, and GridSearch over other hyperparameters
  - GridSearch hyperparameters

# What is Boosting?

- More trees, but differently
  - Instead of parallel, we have serial/stacked/nested
  - Instead of global loss/error, we have “error so far”

Bagging/RF



pred=consensus(f1(x), f2(x), f3(x), f4(x))

*consensus() is via majority vote, mean, etc

Boosting

pred=f1(x) + f2(x) + f3(x) + f4(x)

# What is Boosting?

- Boosting Algorithm

**Algorithm 8.2** *Boosting for Regression Trees*

1. Set $\hat{f}(x) = 0$ and $r_i = y_i$ for all $i$ in the training set.

2. For $b = 1, 2, \ldots, B$, repeat:

   (a) Fit a tree $\hat{f}^b$ with $d$ splits ($d+1$ terminal nodes) to the training data $(X, r)$. ← The (view of the) 'training data' changes below

   (b) Update $\hat{f}$ by adding in a shrunken version of the new tree:

$$\hat{f}(x) \leftarrow \hat{f}(x) + \lambda \hat{f}^b(x). \tag{8.10}$$

   (c) Update the residuals,

$$r_i \leftarrow r_i - \lambda \hat{f}^b(x_i). \tag{8.11}$$

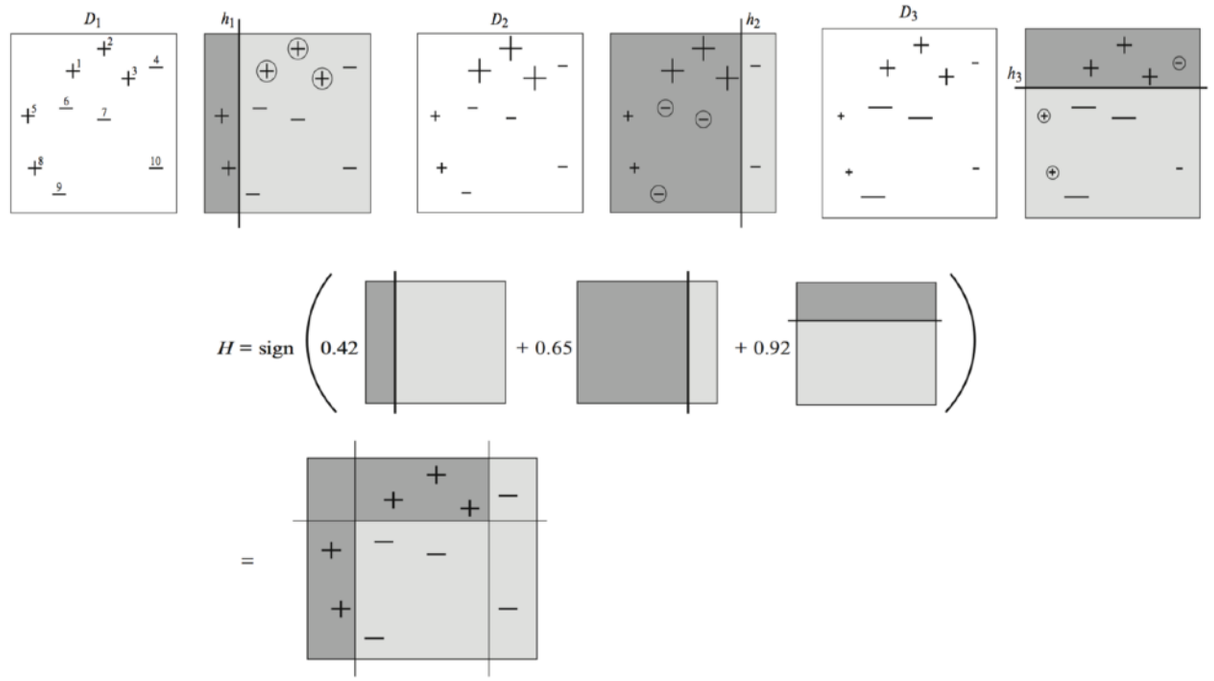
3. Output the boosted model,

$$\hat{f}(x) = \sum_{b=1}^{B} \lambda \hat{f}^b(x). \tag{8.12}$$

# What is Boosting?

- Boosting Algorithm
  - Start with a null-op function, *f*
  - Set your residuals to just be *y*
  - So far, your function does nothing and your errors are *y*
  - Now, for each of the *b* trees you want to build:
    - Fit a new tree, *f**, to the residuals ("error so far"), NOT the outcome *y*
      - Note: we limit this tree's depth
    - Combine *f* and *f**, but with less weight on the new *f**: $f = f(x) + \lambda f^*(x)$
    - Update the residuals by using the new 'combo' function
      - Note: could re-run *f* again on *y*, but less duplicate work to track and update residuals
  - Return *f*, the combination of all *b* functions

# What is Boosting?




Figures from AdaBoost: Schapire and Freund, 2012

# What is Boosting?

- Parameters
  - B, the number of trees
  - d, depth control on the trees
  - $\lambda$, the learning rate
- Why depth controls and slow learning?
  - Instead of ‘hard fit’ which can overfit, we want to learn slowly
  - Each tree takes into account all that came before, so small, focused trees are sufficient
    - (You don’t have to be a genius, you just have to stand on the shoulders of giants)

# Boosting v Bagging/RF

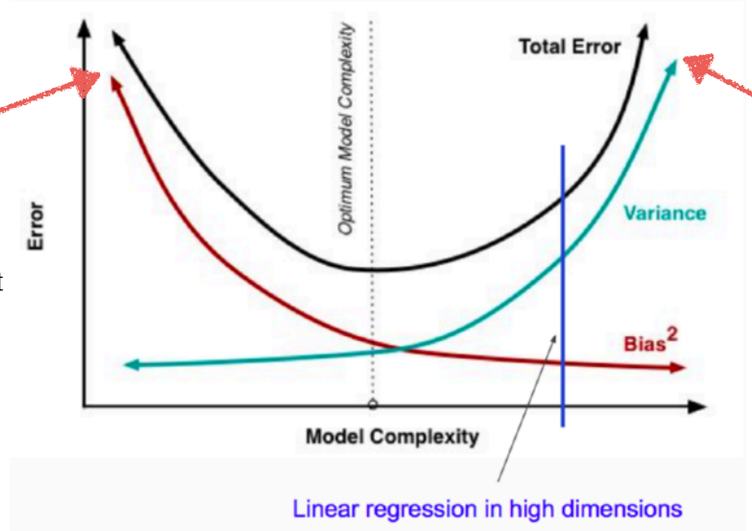
- Compare Boosting to other (tree) approaches
  - Computation/ops
    - Serial training, not parallel
    - Possibly compressible (ie as additive model if d=1)
      - Often keeps around more info than strictly necessary (ie staged_predict ordering)

# Boosting v Bagging/RF

- Compare Boosting to other (tree) approaches
  - Bias/Variance
    - With trees, we can easily traverse the spectrum of bias-variance tradeoff

Boosting:

start here and move right: start with high Bias low Variance, and chip away at Bias with sequential additions to weakest areas

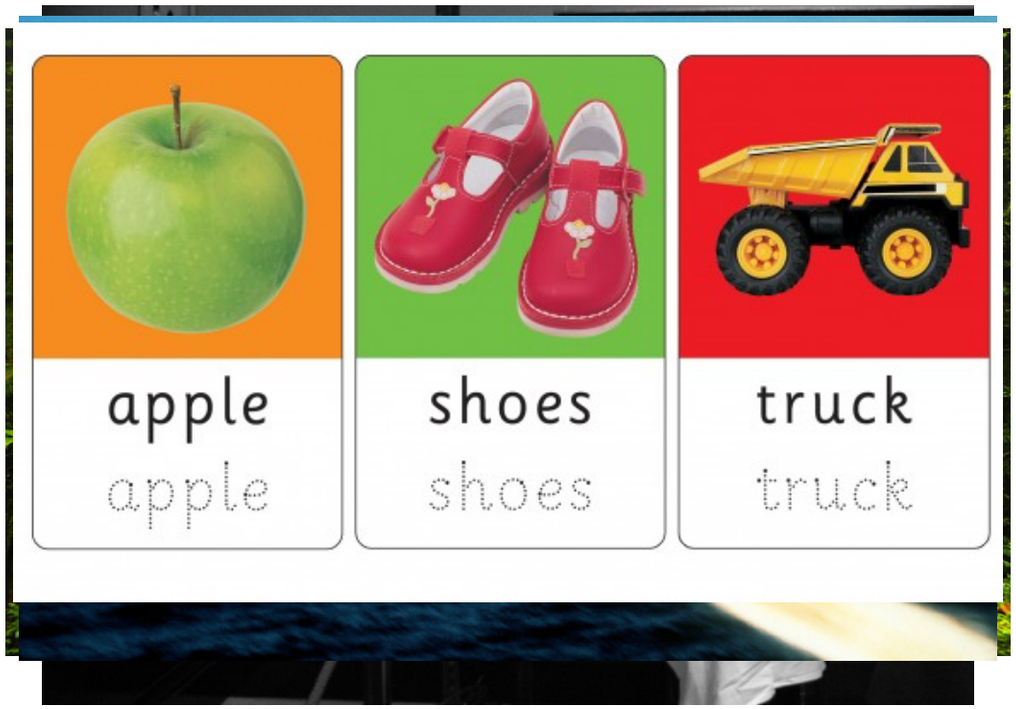



Bagging, Random Forests, and, actually, most models:

start here and move left: start with high Variance low Bias, reign Variance in by averaging. RF: averaging highly correlated quantities doesn't reduce variance as much as averaging uncorrelated quantities, so de-correlate the trees by restricting predictors available

# Boosting v Bagging/RF

- Compare Boosting to other (tree) approaches



# Boosting v Bagging/RF

- Compare Boosting to other (tree) approaches
  - Metaphors: take ‘em with a grain of salt!

| | Bagging | RF | Boosting |
|---|---|---|---|
| **Reproduce a painting** | Have many artists look at a painting from exactly 10m away for a while, afterwards hide it and ask them to try to reproduce it together, equally valuing everyone’s input | …have the artists look at the painting from various different distances/ points of view | Ask one artist to copy the painting. S/he can look back and forth from their canvas to the painting many times to learn about new areas, but is only allowed one brushstroke at a time, and it has to dry before s/he can continue |
| **Grow lush vegetation** | Pine forest | Bonsai forest? | Jungle |
| **Explode something** | Fire lots of rockets | …lots of different rockets | One self-guiding missile |
| **Learn a Language** | Review your Flashcard deck a lot at your desk | Review the deck throughout the day when you’re variously distracted | As you review, remove the cards you already know well so you can focus on the others |

# Types of Boosting

- Compare types of Boosting: computation/speed, generality, flexibility, nomenclature
  - Boosting writ-large:
    - Generally applicable
    - Train to the “error so far”: fit to a modified version of the original data set
    - Can be computationally expensive, must be serially trained (not parallel)
    - (No requirements on loss function)
  - AdaBoost
    - Upweight the points you’re getting wrong, focus more on them in the future
      - Some trainers accept weighted points, or you can do weighted resampling
    - Weight each individual weak learner based on its performance
  - Gradient Boosting
    - Improve handling of loss functions for robustness and speed
    - Differentiable loss functions
      - Squared loss isn’t robust to outliers, might want to use absolute loss instead (similarities to ridge v lasso)
      - With Square Loss, residual and negative gradient are the same, but this is not always the case
    - XGBoost

# Why Boosting?

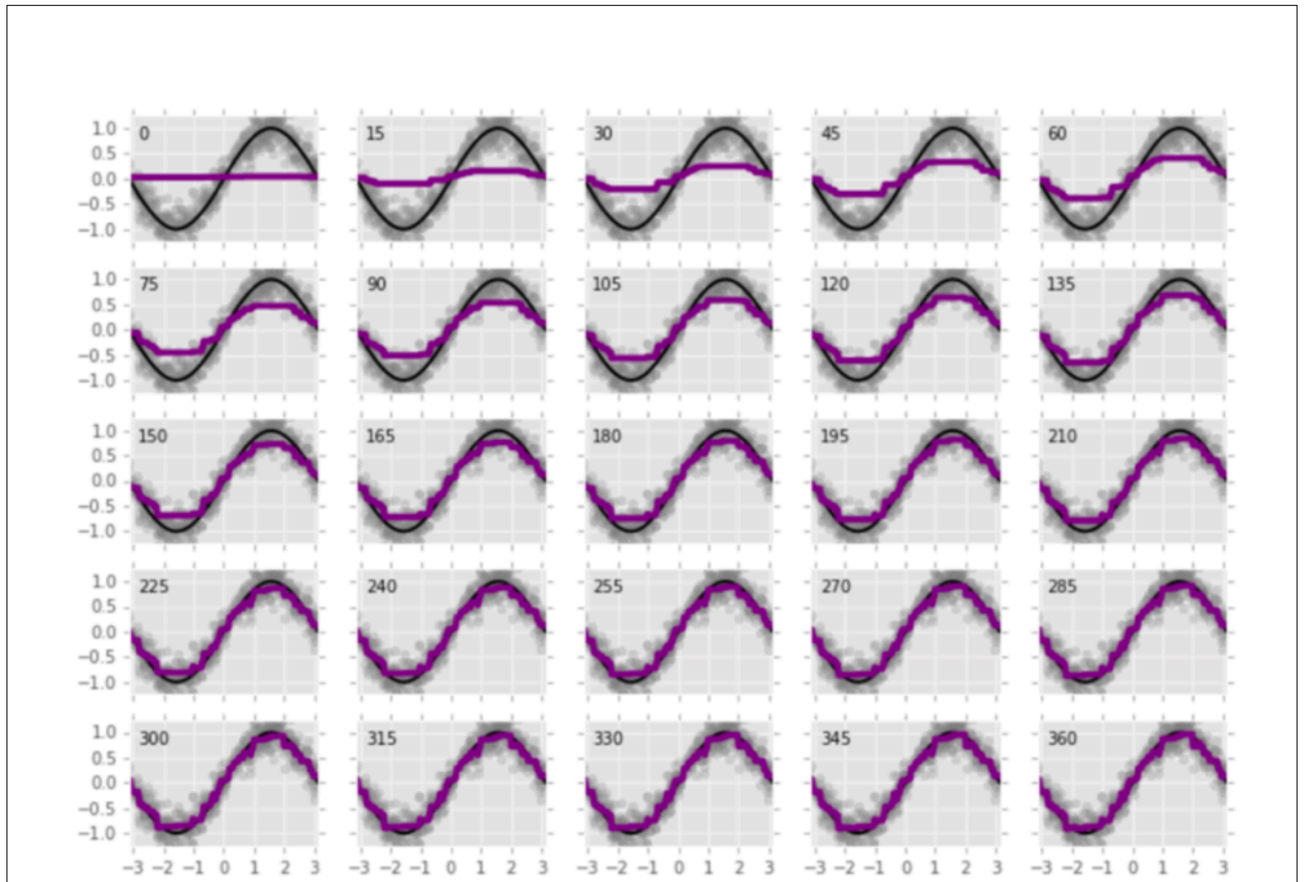
- Performance
  - Powerful off-the-shelf
  - General purpose
  - Wins competitions (Kaggle, Microsoft, Yahoo)
  - Adds cool features (face detection in Viola & Jones, 2001)
- Interpretability
  - Feature Interactions
    - Create a linear model if you want (d=1)
    - Create an order-2 model, look at interactions
  - Valuable to be able to determine whether interactions are important
    - Can even help tell which interactions are important
- n << p
  - Common these days, LR/LDA will be painful

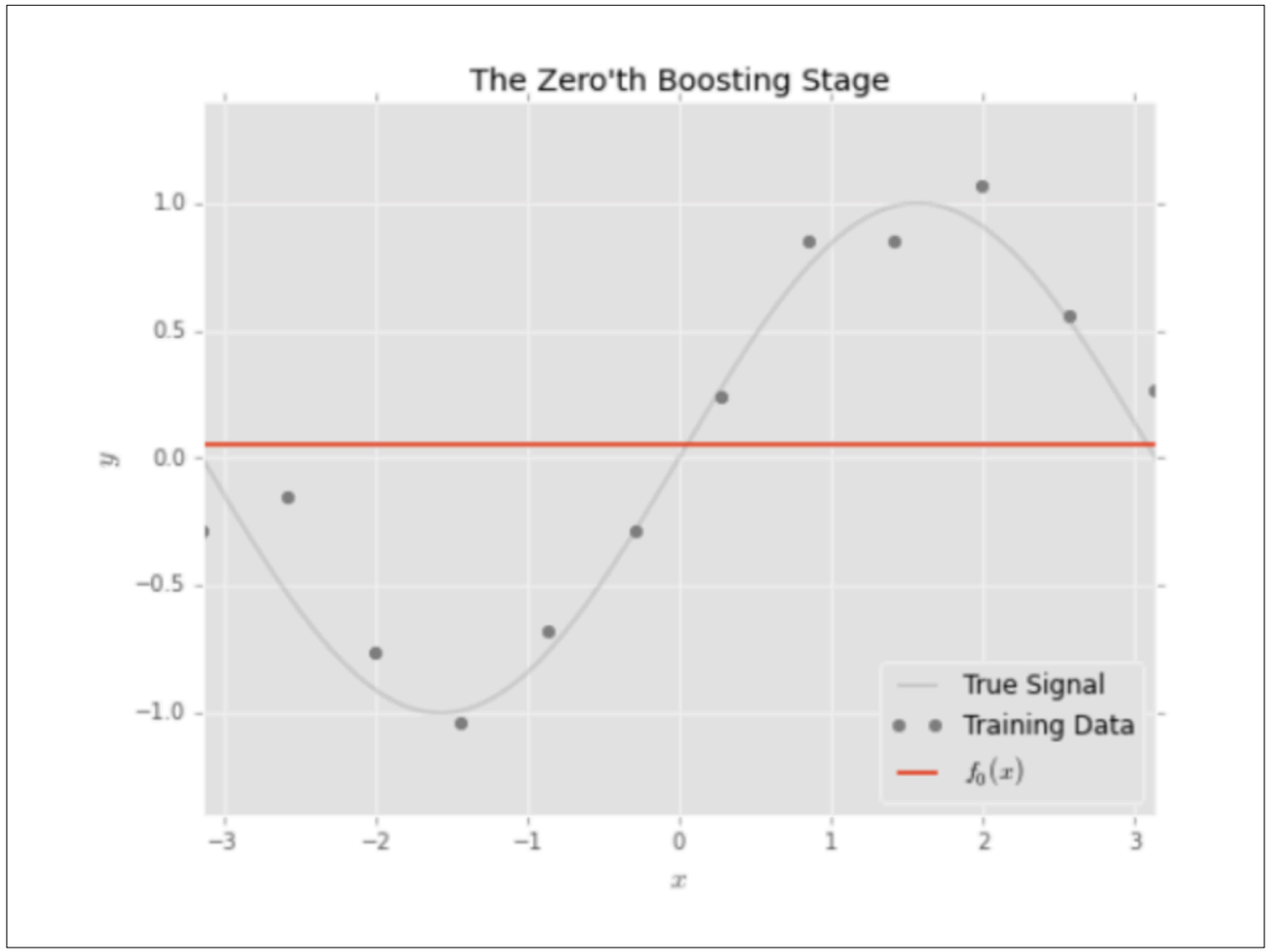
# Assignment

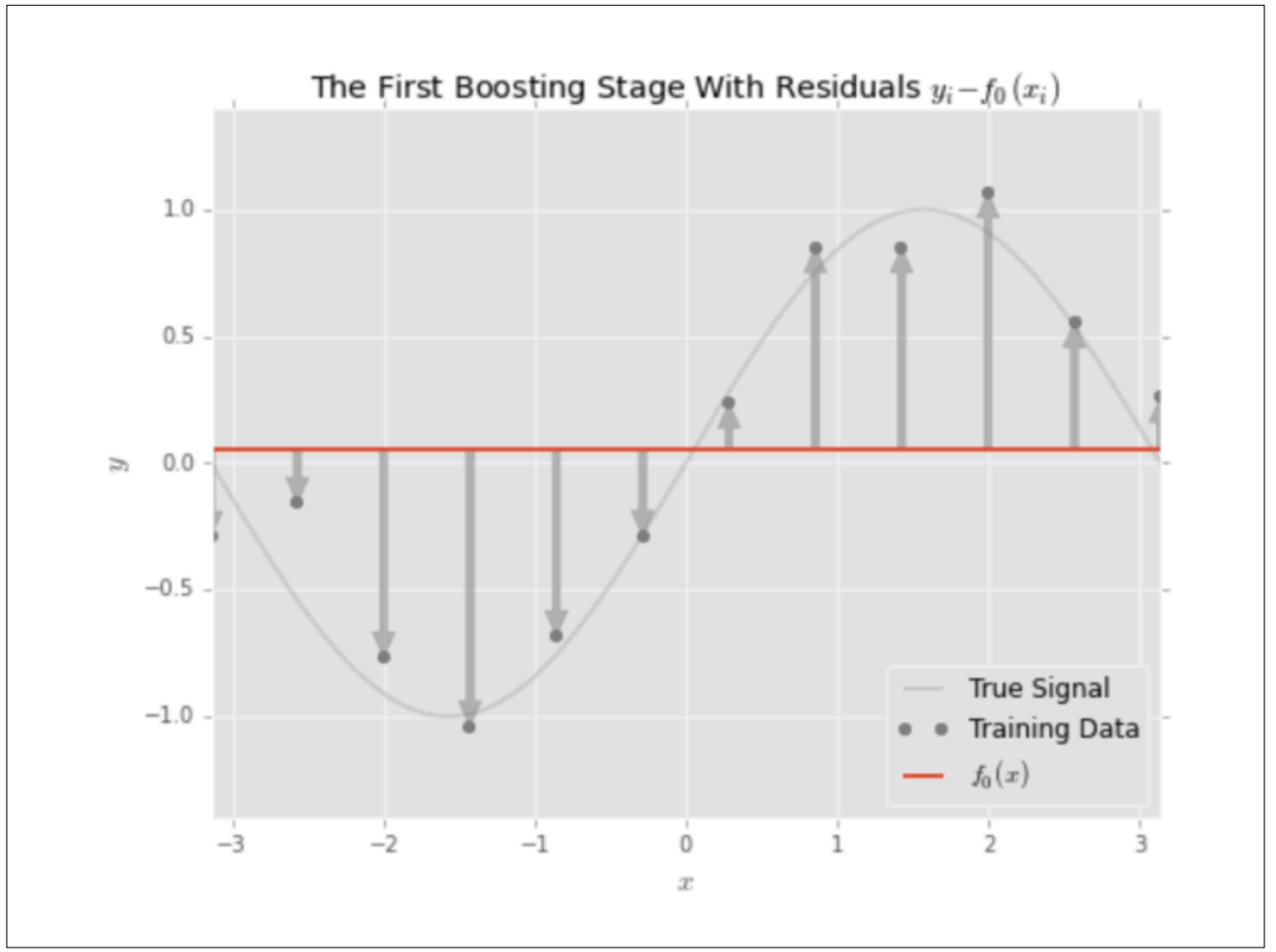
- Compare [AdaBoost, GradientBoosting, RandomForest] regressors
- Investigate effects of B and $\lambda$
- GridSearch over other hyperparameters
- Notes:
  - stage_predict
  - parallelization

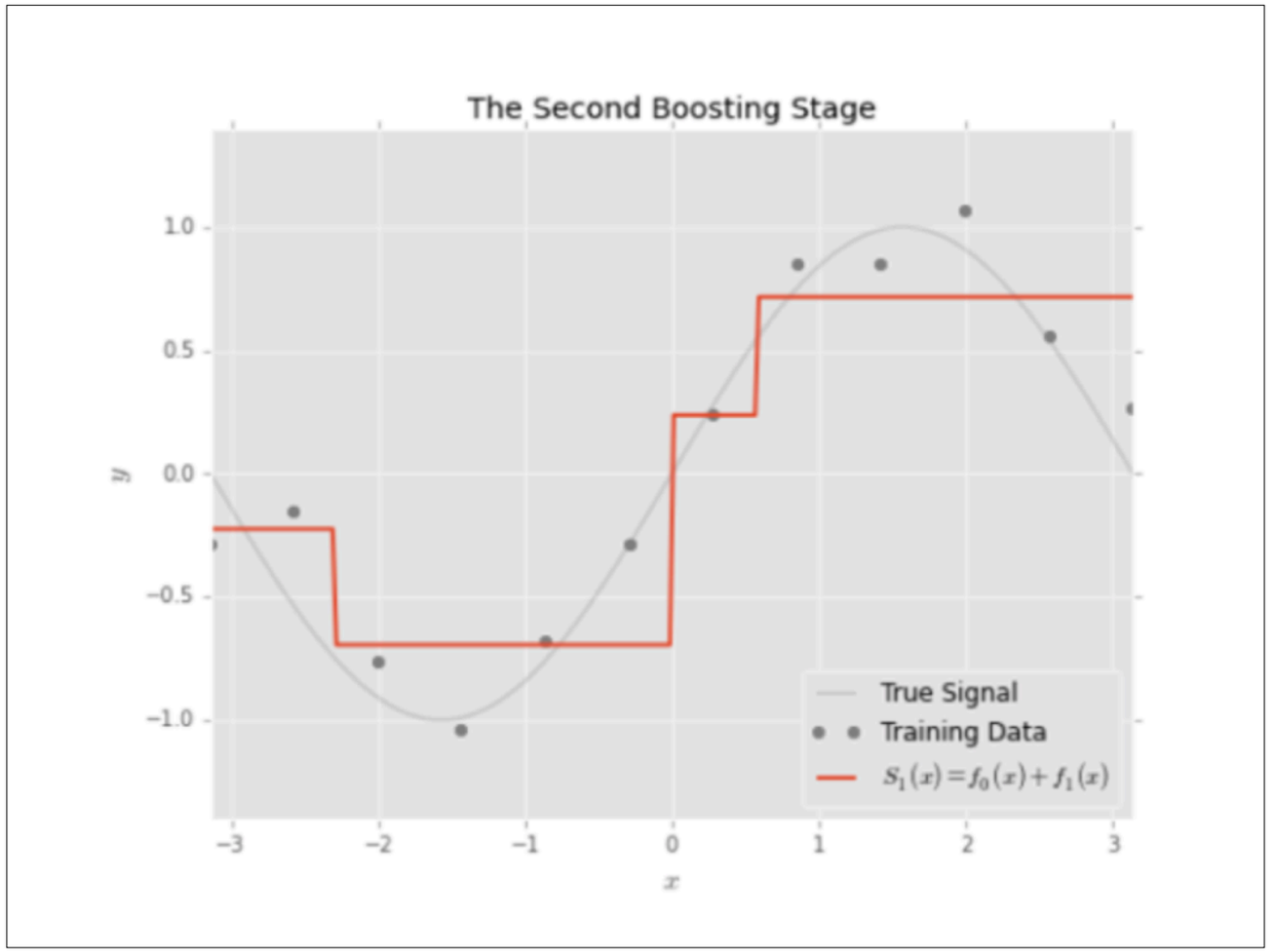

BOOSTED

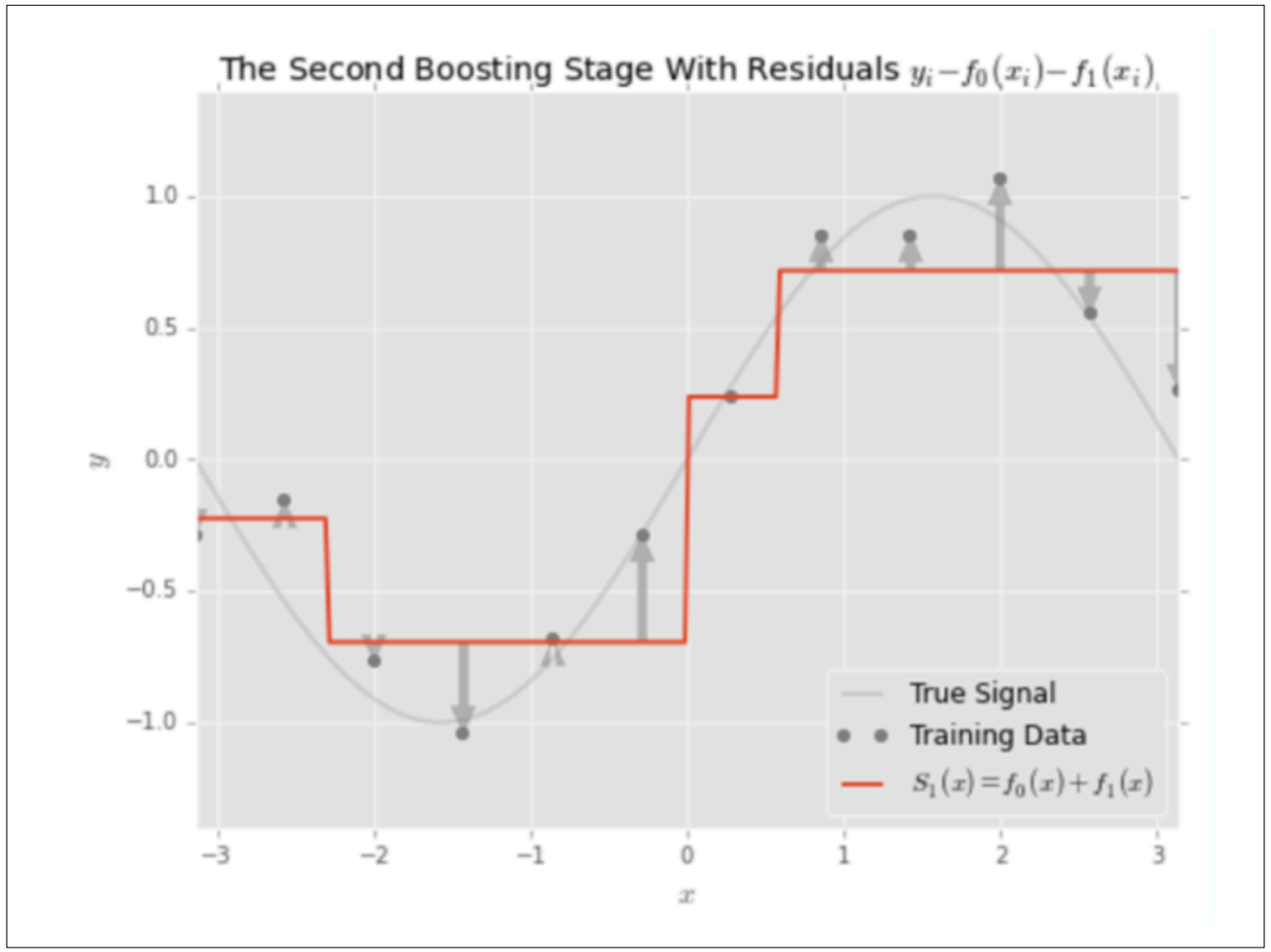
<boosting
visualizations>

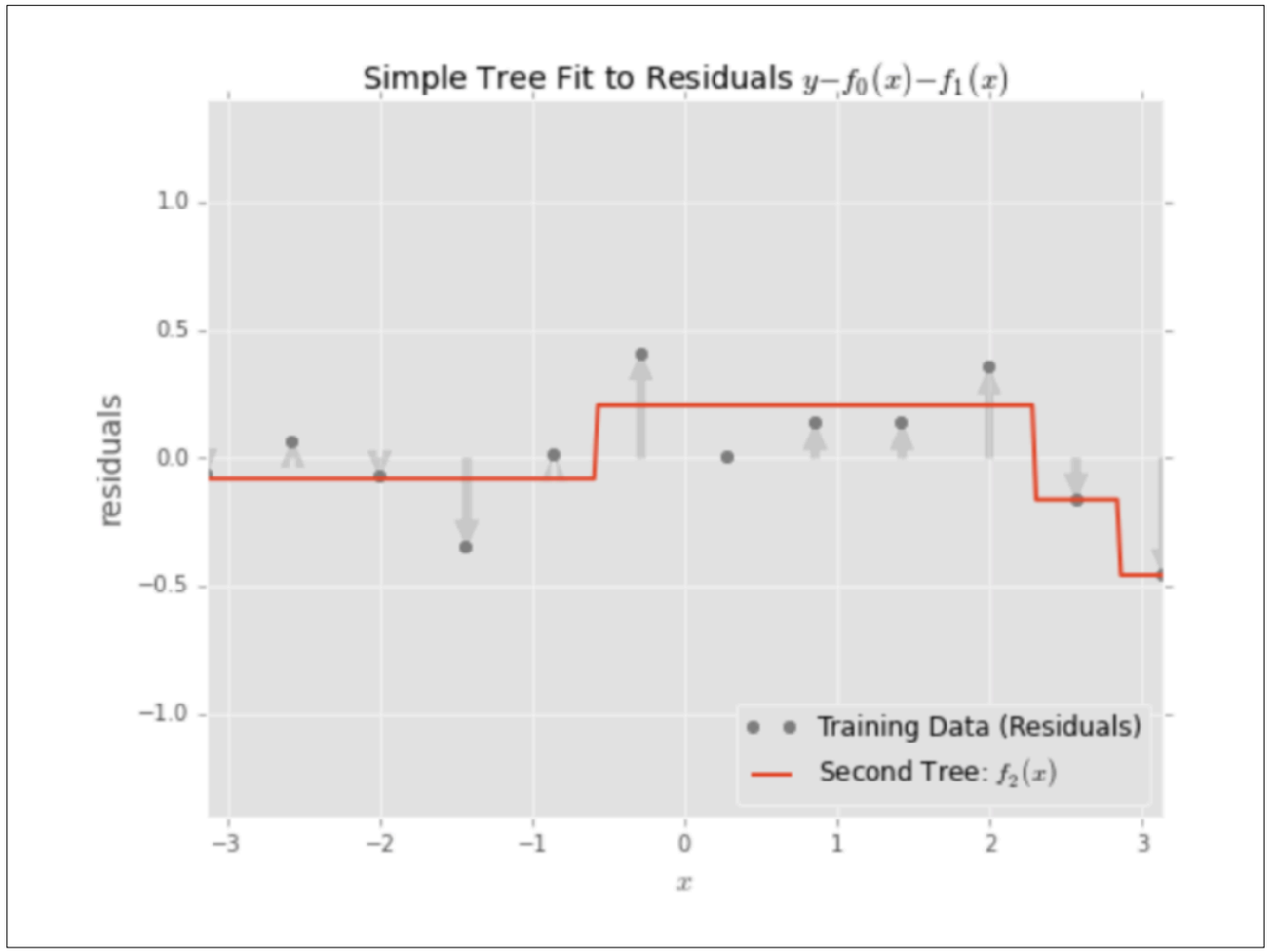

0
15
30
45
60
75
90
105
120
135
150
165
180
195
210
225
240
255
270
285
300
315
330
345
360
1.0
0.5
0.0
−0.5
−1.0
−3 −2 −1 0 1 2 3

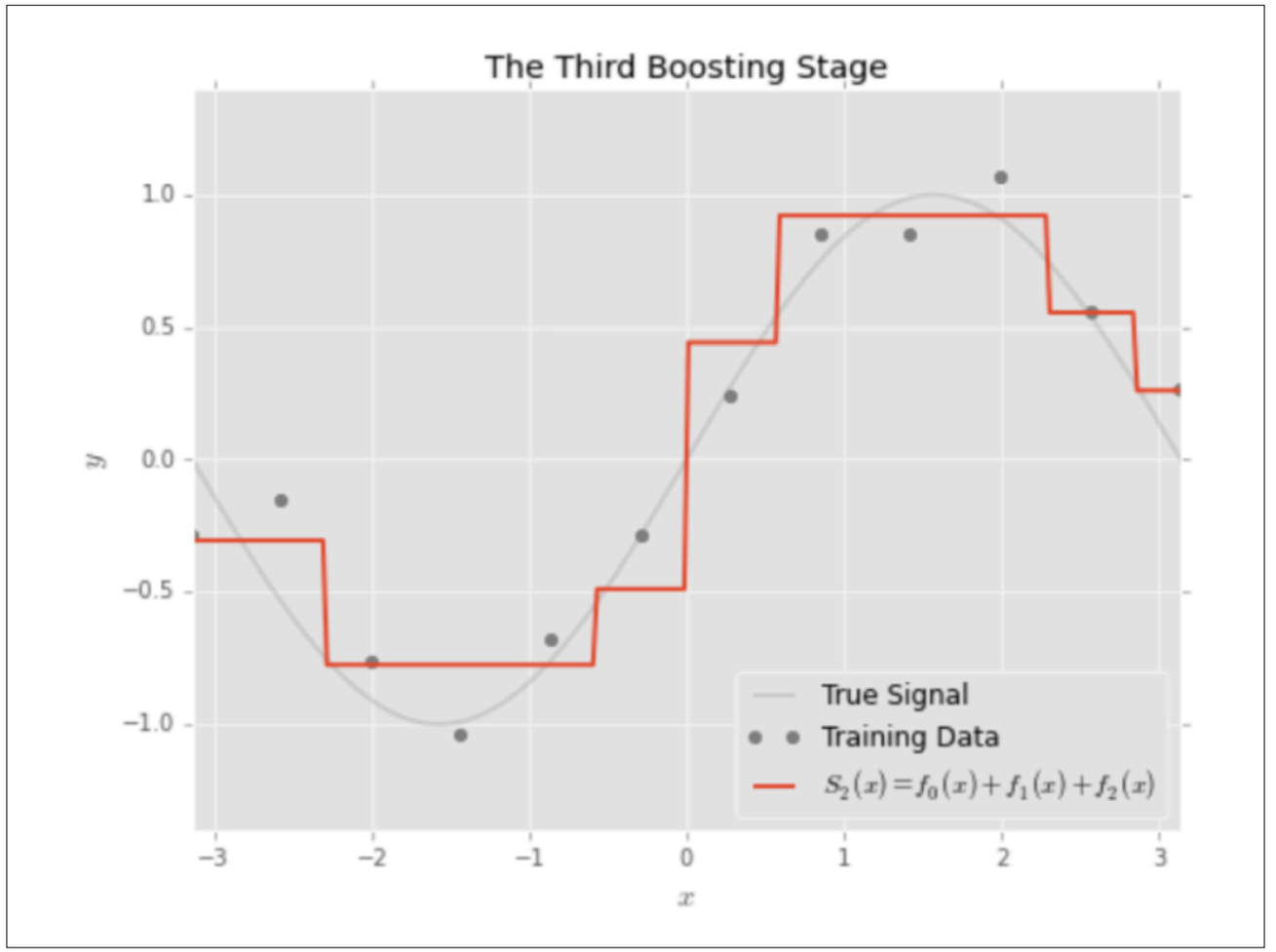

The Zero'th Boosting Stage
1.0
0.5
0.0
−0.5
−1.0
y
−3
−2
−1
0
1
2
3
x
True Signal
Training Data
$f_0(x)$

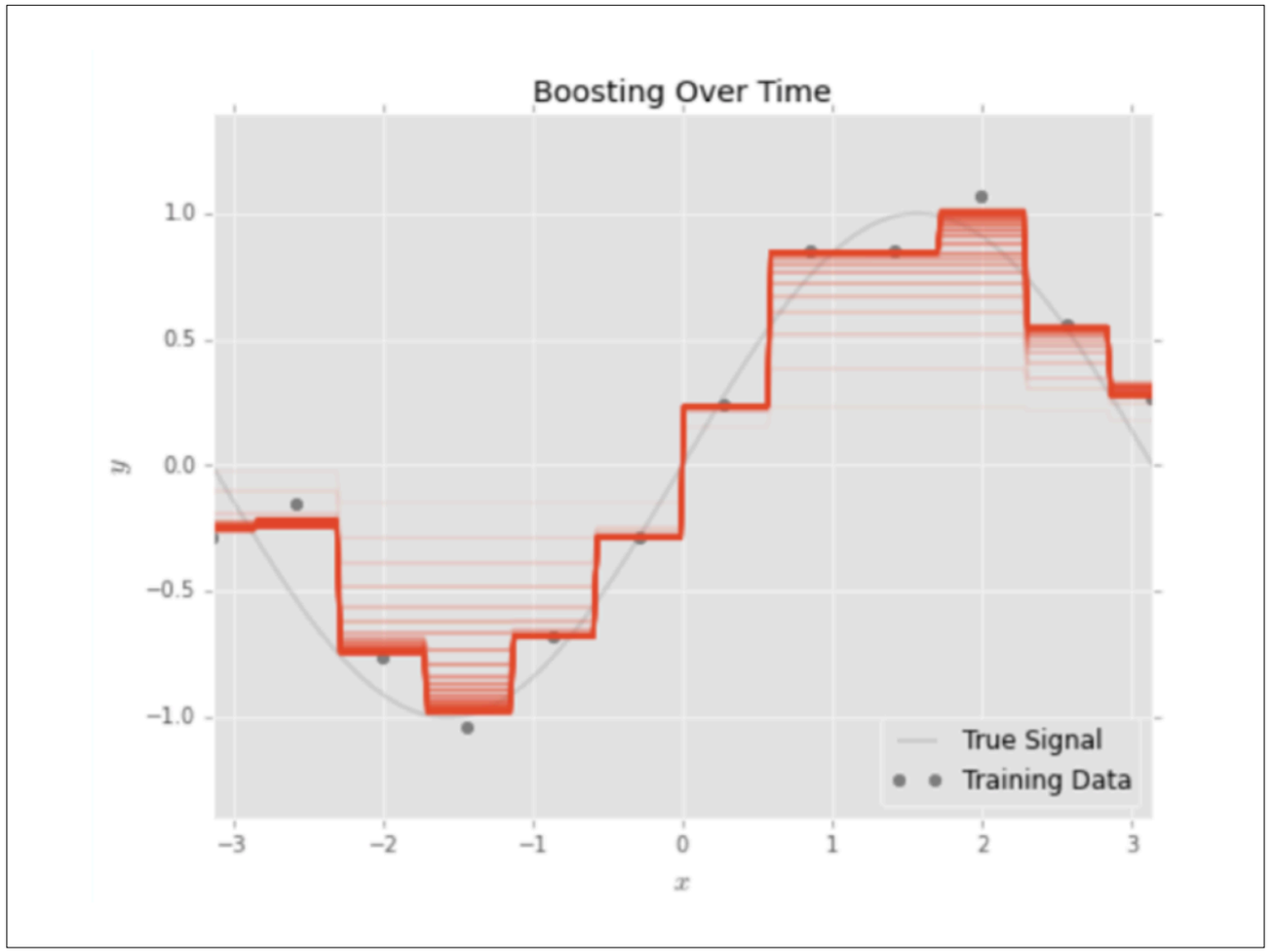

The First Boosting Stage With Residuals $y_i - f_0(x_i)$
1.0
0.5
0.0
−0.5
−1.0
y
−3
−2
−1
0
1
2
3
x
True Signal
Training Data
$f_0(x)$


The Second Boosting Stage
1.0
0.5
0.0
−0.5
−1.0
y
−3
−2
−1
0
1
2
3
x
True Signal
Training Data
$S_1(x) = f_0(x) + f_1(x)$


The Second Boosting Stage With Residuals $y_i - f_0(x_i) - f_1(x_i)$
1.0
0.5
0.0
−0.5
−1.0
y
−3
−2
−1
0
1
2
3
x
True Signal
Training Data
$S_1(x) = f_0(x) + f_1(x)$


Simple Tree Fit to Residuals $y - f_0(x) - f_1(x)$
residuals
1.0
0.5
0.0
−0.5
−1.0
−3
−2
−1
0
1
2
3
$x$
Training Data (Residuals)
Second Tree: $f_2(x)$

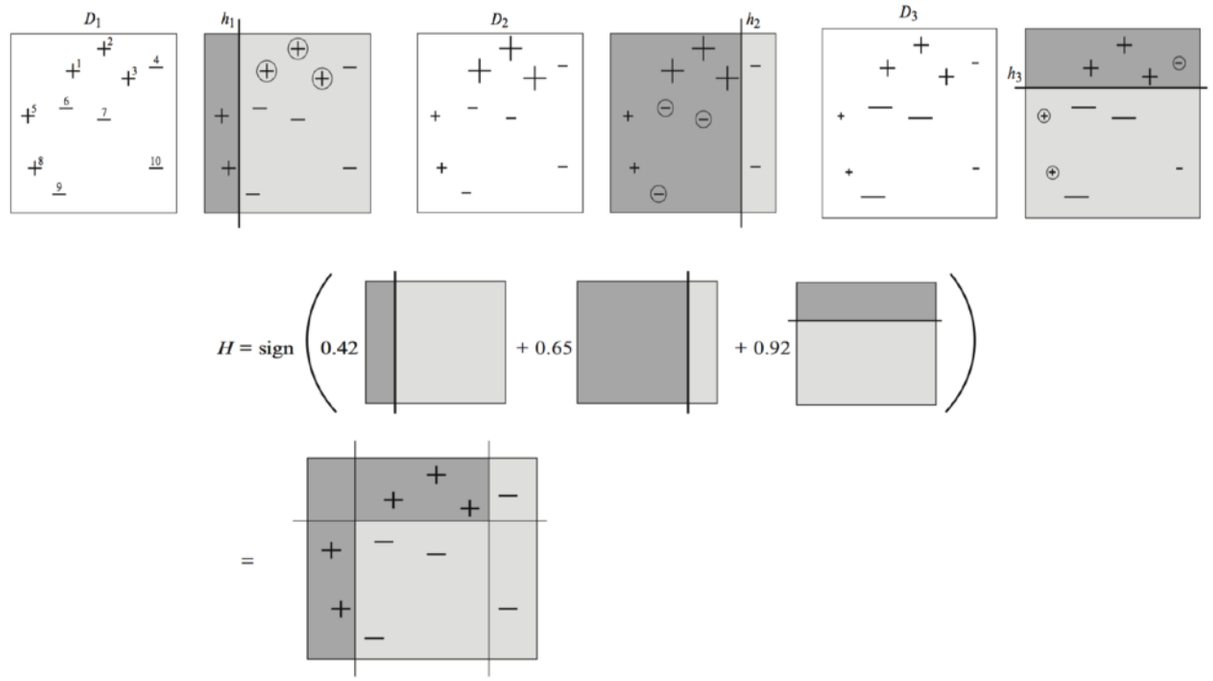

The Third Boosting Stage
1.0
0.5
0.0
−0.5
−1.0
y
−3
−2
−1
0
1
2
3
x
True Signal
Training Data
$S_2(x) = f_0(x) + f_1(x) + f_2(x)$


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

# AdaBoost

- “Adaptive Boosting”
  - Upweight points you get wrong
    - Intuitively, choose the training set for each new classifier that you train based on the results of the previous classifier
  - Weight each individual weak learner based on its performance
- “Weak Learners”
  - Anything not random is valuable
    - Even if it’s consistently wrong.  (Do the opposite!)

# AdaBoost




Figures from AdaBoost: Schapire and Freund, 2012

# AdaBoost (Schapire)

Given: $(x_1, y_1), \ldots, (x_m, y_m)$ where $x_i \in \mathcal{X}$, $y_i \in \{-1, +1\}$.
Initialize: $D_1(i) = 1/m$ for $i = 1, \ldots, m$.
For $t = 1, \ldots, T$:

- Train weak learner using distribution $D_t$.
- Get weak hypothesis $h_t : \mathcal{X} \to \{-1, +1\}$.
- Aim: select $h_t$ with low weighted error:

$$\varepsilon_t = \Pr_{i \sim D_t} [h_t(x_i) \neq y_i].$$

- Choose $\alpha_t = \frac{1}{2} \ln \left( \frac{1 - \varepsilon_t}{\varepsilon_t} \right)$.
- Update, for $i = 1, \ldots, m$:

$$D_{t+1}(i) = \frac{D_t(i) \exp(-\alpha_t y_i h_t(x_i))}{Z_t}$$

where $Z_t$ is a normalization factor (chosen so that $D_{t+1}$ will be a distribution).

Output the final hypothesis:

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

**Fig. 1** The boosting algorithm AdaBoost.

Start w equation for final classifier, H(x):

We output T weak classifiers

h_t(x) is output of weak classifier t

Note that outputs are {-1, 1} not {0, 1}

Final decision based on sign of the sum of all weak classifiers

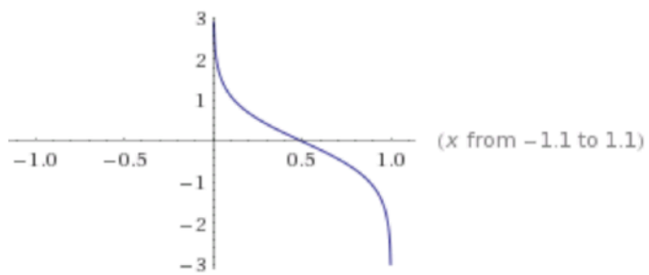
# Let's talk about $\alpha$

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

Input:

$$0.5 \log\left(\frac{1-x}{x}\right)$$

Plots:

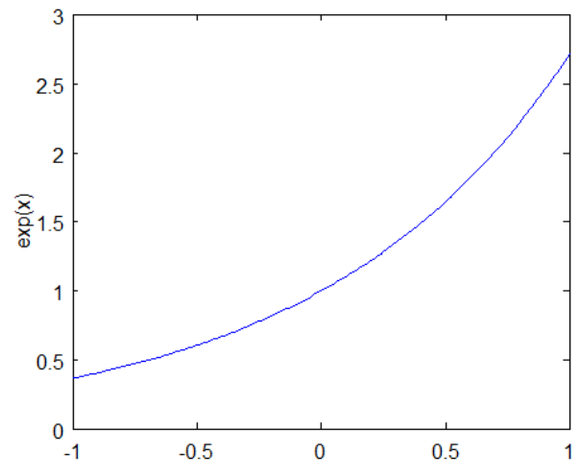



First classifier, we start with uniform/equal weights for all training examples
e is the number of misclassifications on the training set divided by the training set size
What happens as error rate changes?  See plot: error rate on x, alpha on y

- classifier weight grows exponentially as the error approaches 0
- weight is zero if error rate is 0.5.  50% accuracy is no better than random guessing, so ignore it
- classifier weight grows exponentially negative as error approaches 1.  why not ignore it?

# Weight update

$$D_{t+1}(i) = \frac{D_t(i)\exp(-\alpha_t y_i h_t(x_i))}{Z_t}$$




D_t is often described (indeed, in original AdaBoost paper) as a distribution: each weight D_i represents the probability of training example i being selected as part of the training set.

To make it a distribution, all the probabilities have to add up to 1. That's why we divide by the sum (Z_t)

This has to be figured out for each training example i. The weight is updated for each training example. (This is *in addition* to each weak learner getting an alpha weight)

How does the exponential term behave? Let's look at how exp(x) behaves:

will return a fraction for negative values of x

will return a value greater than one for positive values of x

So… the weight for training sample i will be either increased or decreased depending on the final sign of the term inside exp()

for binary classifiers that output {-1, 1} the terms y and h(x) only contribute to the sign, not the magnitude

if predicted and actual agree, y*h(x) will be +1

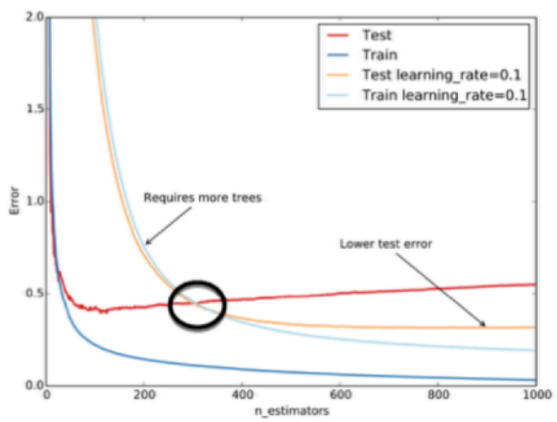
else, will be -1

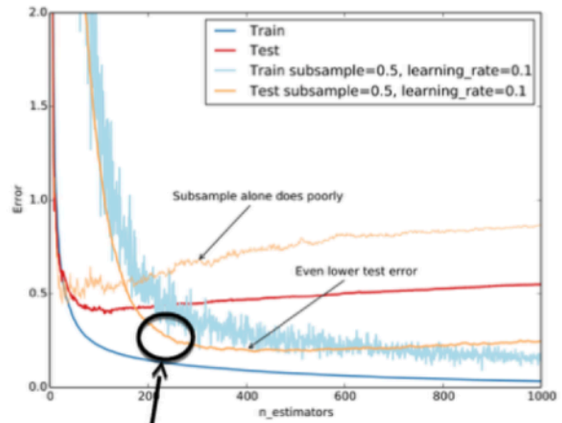
So… misclassifications by a classifier with a positive alpha will cause a particular training example to be given a larger weight.

Alpha term's effect: if a weak classifier misclassifies an input, we don't take that as seriously as a strong classifier's mistake

# Stochastic Boosting

- Just like in Random Forests!
  - max_features
  - sub_sample




Lower test error!
Fewer trees to get there!

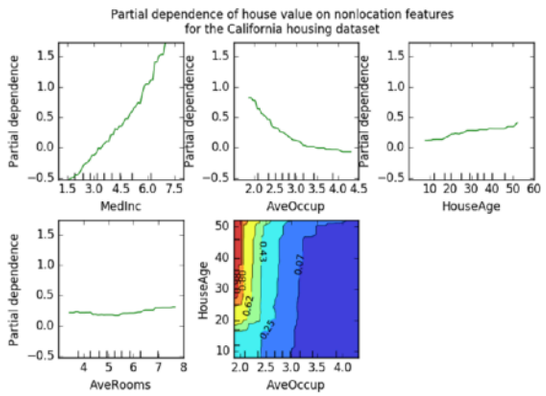
# Grid Search

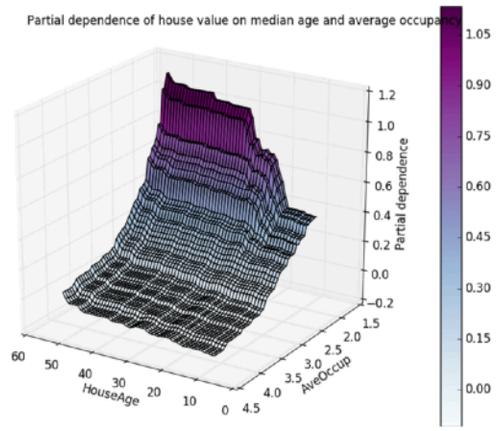
- Stupidly try all the things
  - For each parameter value, try each other parameter value…
  - Combinatorial growth

```
from sklearn.grid_search import GridSearchCV
param_grid = {'learning_rate': [0.1, 0.05, 0.02, 0.01]
              'max_depth': [4, 6],
              'min_samples_leaf': [3, 5, 9, 17],
              'max_features': [1.0, 0.3, 0.1]}
est = GradientBoostingRegressor(n_estimators=3000)
gs_cv = GridSearchCV(est, param_grid).fit(X, y)
# best hyperparameter setting
gs_cv.best_params_
```

# Partial Dependency Plots and Feature Importance

- Feature Importance
  - Same as for Random Forests: focus on the information gain given to you by a feature
- Partial Dependency Plots
  - Determine the effect of one feature, fixing all the others constant
  - Determine the interactions between features (in low, human-interpretable dimensions)



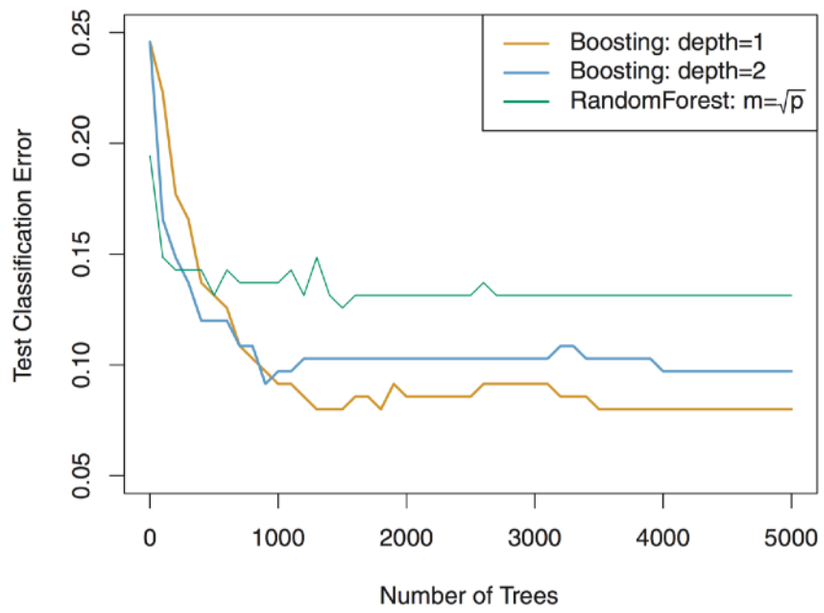
# Appendix

# AdaBoost (Assignment)

**Algorithm 10.1** *AdaBoost.M1.*

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 $G(x) = \text{sign}\left[\sum_{m=1}^{M} \alpha_m G_m(x)\right]$.

# Boosting Example
# From: ISLR



# Loss Functions

Machine Learning, Murphy

| Name | Loss | Derivative | $f^*$ | Algorithm |
|---|---|---|---|---|
| Squared error | $\frac{1}{2}(y_i - f(\mathbf{x}_i))^2$ | $y_i - f(\mathbf{x}_i)$ | $\mathbb{E}[y\|\mathbf{x}_i]$ | L2Boosting |
| Absolute error | $\|y_i - f(\mathbf{x}_i)\|$ | $\text{sgn}(y_i - f(\mathbf{x}_i))$ | $\text{median}(y\|\mathbf{x}_i)$ | Gradient boosting |
| Exponential loss | $\exp(-\tilde{y}_i f(\mathbf{x}_i))$ | $-\tilde{y}_i \exp(-\tilde{y}_i f(\mathbf{x}_i))$ | $\frac{1}{2}\log\frac{\pi_i}{1-\pi_i}$ | AdaBoost |
| Logloss | $\log(1 + e^{-\tilde{y}_i f_i})$ | $y_i - \pi_i$ | $\frac{1}{2}\log\frac{\pi_i}{1-\pi_i}$ | LogitBoost |

**Table 16.1** Some commonly used loss functions, their gradients, their population minimizers $f^*$, and some algorithms to minimize the loss. For binary classification problems, we assume $\tilde{y}_i \in \{-1, +1\}$, $y_i \in \{0, 1\}$ and $\pi_i = \text{sigm}(2f(\mathbf{x}_i))$. For regression problems, we assume $y_i \in \mathbb{R}$. Adapted from (Hastie et al. 2009, p360) and (Buhlmann and Hothorn 2007, p483).

# Gradient Boosting
# From: A Gentle Introduction to Gradient Boosting, Cheng Li

## Gradient Boosting for Regression

### Gradient Descent

Minimize a function by moving in the opposite direction of the gradient.

$$\theta_i := \theta_i - \rho \frac{\partial J}{\partial \theta_i}$$

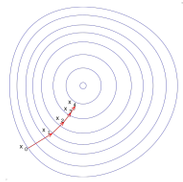


Figure : Gradient Descent. Source: `http://en.wikipedia.org/wiki/Gradient_descent`

## Gradient Boosting for Regression

### How is this related to gradient descent?

Loss function $L(y, F(x)) = (y - F(x))^2/2$
We want to minimize $J = \sum_i L(y_i, F(x_i))$ by adjusting $F(x_1), F(x_2), ..., F(x_n)$.
Notice that $F(x_1), F(x_2), ..., F(x_n)$ are just some numbers. We can treat $F(x_i)$ as parameters and take derivatives

$$\frac{\partial J}{\partial F(x_i)} = \frac{\partial \sum_i L(y_i, F(x_i))}{\partial F(x_i)} = \frac{\partial L(y_i, F(x_i))}{\partial F(x_i)} = F(x_i) - y_i$$

So we can interpret residuals as negative gradients.

$$y_i - F(x_i) = -\frac{\partial J}{\partial F(x_i)}$$

# Loss Functions
# From: A Gentle Introduction to Gradient Boosting, Cheng Li

## Gradient Boosting for Regression

### Loss Functions for Regression Problem

- Absolute loss (more robust to outliers)

$$L(y, F) = |y - F|$$

- Huber loss (more robust to outliers)

$$L(y, F) = \begin{cases} \frac{1}{2}(y - F)^2 & |y - F| \leq \delta \\ \delta(|y - F| - \delta/2) & |y - F| > \delta \end{cases}$$

| $y_i$ | 0.5 | 1.2 | 2 | 5* |
|---|---|---|---|---|
| $F(x_i)$ | 0.6 | 1.4 | 1.5 | 1.7 |
| Square loss | 0.005 | 0.02 | 0.125 | 5.445 |
| Absolute loss | 0.1 | 0.2 | 0.5 | 3.3 |
| Huber loss($\delta = 0.5$) | 0.005 | 0.02 | 0.125 | 1.525 |