Ensemble Methods: Boosting

Rishabh Iyer

University of Texas at Dallas

Based on the slides of Nick Rouzzi, Vibhav Gogate and Rob Schapire

Last Time

- Variance reduction via bagging
  - Generate “new” training data sets by sampling with replacement from the empirical distribution
    Bootstrap Sampling
  - Learn a classifier for each of the newly sampled sets
  - Combine the classifiers for prediction
- Today: how to reduce bias for binary classification problems
  - Adaptive Boosting (AdaBoost)
  - Gradient Boosting

Boosting

- How to translate rules of thumb (i.e., good heuristics) into good learning algorithms

High Bias & Low Variance.

- For example, if we are trying to classify email as spam or not spam, a good rule of thumb may be that emails containing "Click" or "FREE" are likely to be spam most of the time

Boosting

- Freund & Schapire
  - Theory for “weak learners” in late 80’s
- Weak Learner: performance on ***any*** training set is slightly better than chance prediction
- Intended to answer a theoretical question, not as a practical way to improve learning
  - Tested in mid 90’s using not-so-weak learners
  - Works anyway!

PAC Learning

- Given i.i.d samples from an unknown, arbitrary distribution
  - "Strong" learning algorithm
    - For any distribution with high probability given polynomially many samples (and polynomial time) can find classifier with arbitrarily small error
  - "Weak" learning algorithm
    - Same, but error only needs to be slightly better than random guessing (e.g., accuracy only needs to exceed 50% for binary classification)
  - **Does weak learnability imply strong learnability?**

Boosting

1. Weight all training samples equally
2. Train model on training set
3. Compute error of model on training set
4. Increase weights on training cases model gets wrong
5. Train new model on re-weighted training set
6. Re-compute errors on weighted training set
7. Increase weights again on cases model gets wrong

Repeat until tired

Final model: weighted prediction of each model

feat > x

yes → Spam (+1)

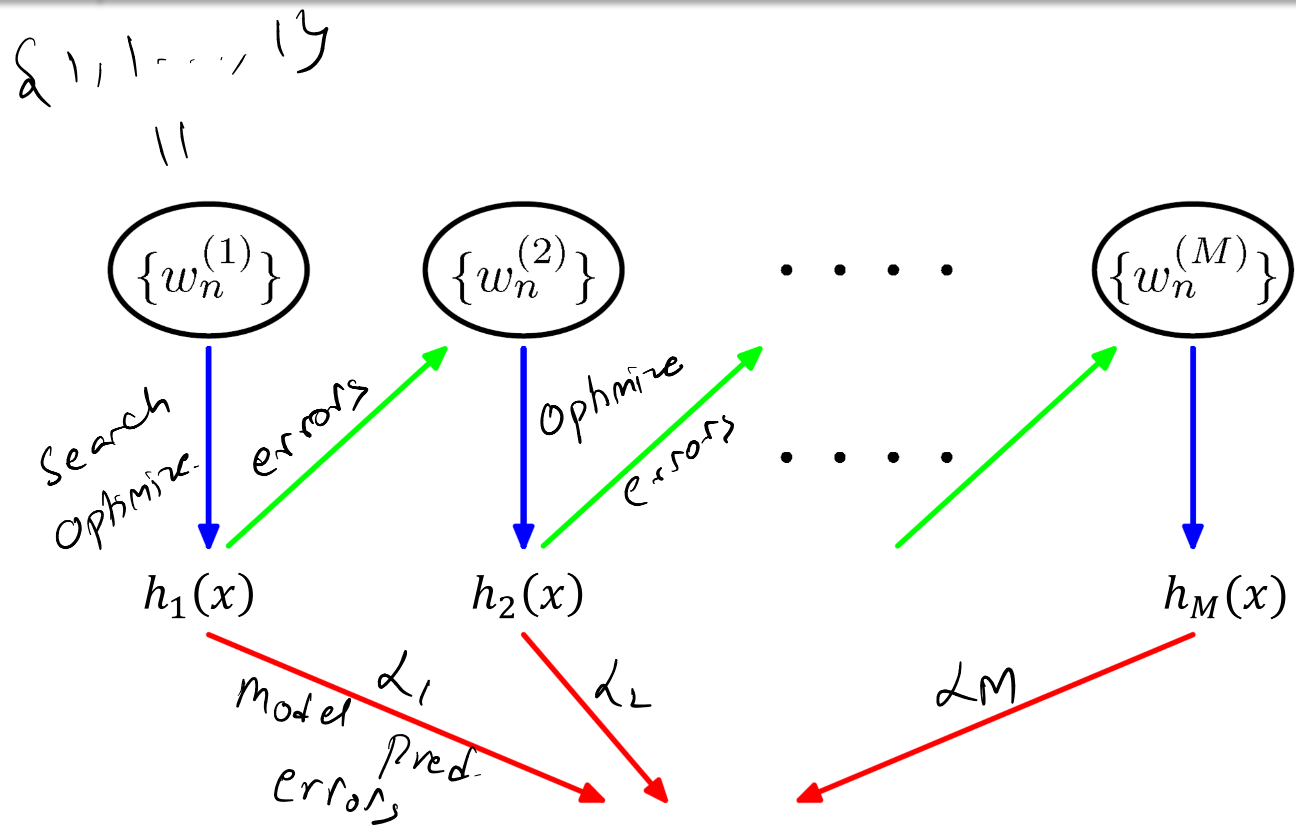
no → Not Spam (-1)

$$\min_{\text{feat},\ \text{thresh}} \ Err(\text{Train} \mid \text{feat}, \text{thresh})$$

$$x_1^{(1)} \geq x_1^{(2)} \geq \cdots \geq x_1^{(M)} \leftarrow \text{Feat } 1$$

$$x_1 \leq \text{thresh}$$

Boosting: Graphical Illustration



$$h(x) = \text{sign}\left(\sum_m \alpha_m h_m(x)\right)$$

AdaBoost

1. Initialize the data weights $w_1, \dots, w_M$ for the first round as $w_1^{(1)}, \dots, w_M^{(1)} = \frac{1}{M}$
2. For $t = 1, \dots, T$
   a) Select a classifier h_t for the T^{th} round by minimizing the weighted error

$$\epsilon_t = \sum_m w_m^{(t)} 1_{h_t(x^{(m)}) \neq y^{(m)}}$$

0/1 error

   b) Compute

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

if $\mathrm{Sign}(h_t(x^{(m)})) = y^{(m)}$ then $w_m^{t+1} \downarrow$

   c) Update the weights $\forall$ Data points m

$$w_m^{(t+1)} = \frac{w_m^{(t)} \exp\left(-y^{(m)} h_t(x^{(m)}) \alpha_t\right)}{2\sqrt{\epsilon_t \cdot (1 - \epsilon_t)}}$$

if $\mathrm{Sign}(h_t(x^{(m)})) \neq y^{(m)}$ then $w_m^{t+1} \uparrow$

Step 1 is solved the same way as decision trees. Search over splits and features that minimizes the error

AdaBoost

1. Initialize the data weights $w_1, \ldots, w_M$ for the first round as $w_1^{(1)}, \ldots, w_M^{(1)} = \frac{1}{M}$
2. For $t = 1, \ldots, T$
   a) Select a classifier h_t for the T^{th} round by minimizing the weighted error

$$\epsilon_t = \sum_m w_m^{(t)} 1_{h_t(x^{(m)}) \neq y^{(m)}}$$

   b) Compute

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

Weighted number of incorrect classifications of the t^{th} classifier

   c) Update the weights

$$w_m^{(t+1)} = \frac{w_m^{(t)} \exp\left(-y^{(m)} h_t\left(x^{(m)}\right) \alpha_t\right)}{2\sqrt{\epsilon_t \cdot (1 - \epsilon_t)}}$$

AdaBoost

1. Initialize the data weights $w_1, \ldots, w_M$ for the first round as $w_1^{(1)}, \ldots, w_M^{(1)} = \frac{1}{M}$
2. For $t = 1, \ldots, T$
   a) Select a classifier h_t for the T^{th} round by minimizing the weighted error

$$\epsilon_t = \sum_m w_m^{(t)} 1_{h_t(x^{(m)}) \neq y^{(m)}}$$

   b) Compute

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

$\epsilon_t \to 0$
$\alpha_t \to \infty$

   c) Update the weights

$$w_m^{(t+1)} = \frac{w_m^{(t)} \exp\left(-y^{(m)} h_t\left(x^{(m)}\right) \alpha_t\right)}{2\sqrt{\epsilon_t \cdot (1 - \epsilon_t)}}$$

AdaBoost

1. Initialize the data weights $w_1, \dots, w_M$ for the first round as $w_1^{(1)}, \dots, w_M^{(1)} = \frac{1}{M}$
2. For $t = 1, \dots, T$
   a) Select a classifier h_t for the T^{th} round by minimizing the weighted error

$$\epsilon_t = \sum_m w_m^{(t)} 1_{h_t(x^{(m)}) \neq y^{(m)}}$$

   b) Compute

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

$\epsilon_t \to .5$
$\alpha_t \to 0$

   c) Update the weights

$$w_m^{(t+1)} = \frac{w_m^{(t)} \exp\left(-y^{(m)} h_t\left(x^{(m)}\right) \alpha_t\right)}{2\sqrt{\epsilon_t \cdot (1-\epsilon_t)}}$$

$\alpha_t \geq 0$

for $\varepsilon_t \leq 0.5$

AdaBoost

1. Initialize the data weights $w_1, \dots, w_M$ for the first round as $w_1^{(1)}, \dots, w_M^{(1)} = \frac{1}{M}$
2. For $t = 1, \dots, T$
   a) Select a classifier h_t for the T^{th} round by minimizing the weighted error

$$\epsilon_t = \sum_m w_m^{(t)} 1_{h_t(x^{(m)}) \neq y^{(m)}}$$

   b) Compute

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

$\epsilon_t \to 1$

$\alpha_t \to -\infty$

   c) Update the weights

$$w_m^{(t+1)} = \frac{w_m^{(t)} \exp\left(-y^{(m)} h_t\left(x^{(m)}\right)\alpha_t\right)}{2\sqrt{\epsilon_t \cdot (1-\epsilon_t)}}$$

AdaBoost

1. Initialize the data weights $w_1, \ldots, w_M$ for the first round as $w_1^{(1)}, \ldots, w_M^{(1)} = \frac{1}{M}$
2. For $t = 1, \ldots, T$
   a) Select a classifier h_t for the T^{th} round by minimizing the weighted error

$$\epsilon_t = \sum_m \underbrace{w_m^{(t)}} 1_{h_t(x^{(m)}) \neq y^{(m)}}$$

   b) Compute

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

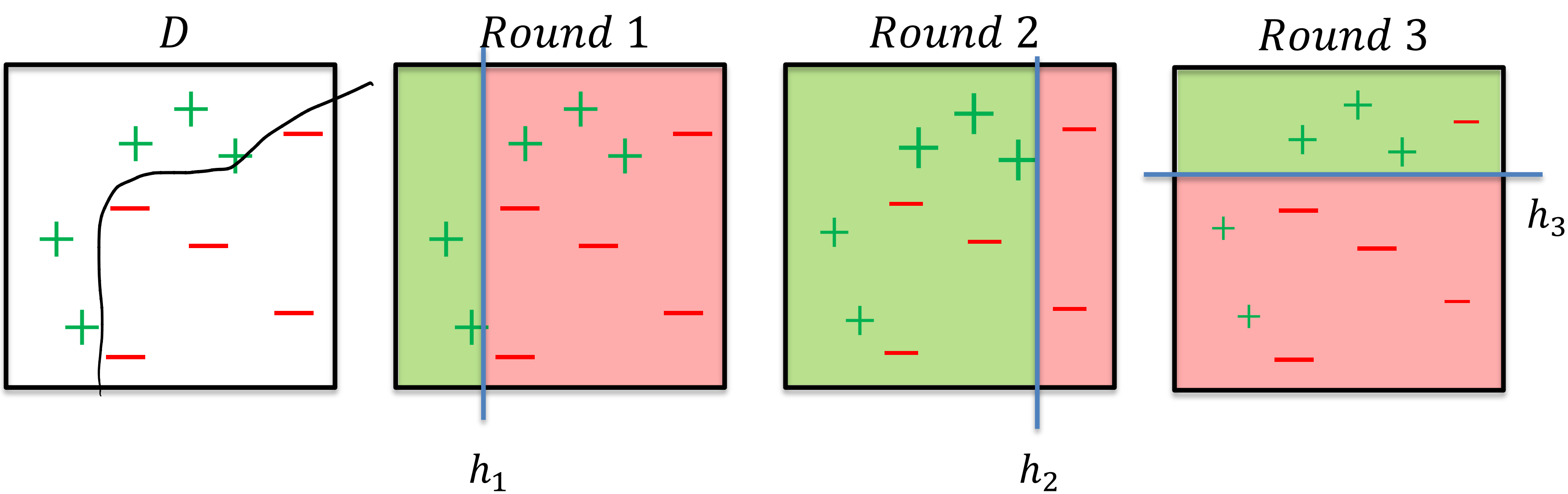
   c) Update the weights

$$\underbrace{w_m^{(t+1)}} = \frac{w_m^{(t)} \exp\left(-y^{(m)} h_t\left(x^{(m)}\right) \alpha_t\right)}{2\sqrt{\epsilon_t \cdot (1-\epsilon_t)}}$$

Normalization constant

Example

- Consider a classification problem where vertical and horizontal lines (and their corresponding half spaces) are the weak learners

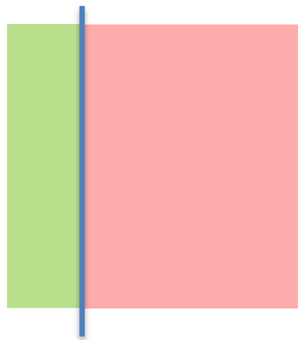
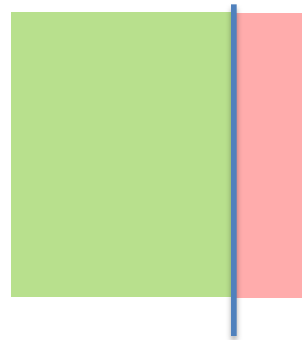
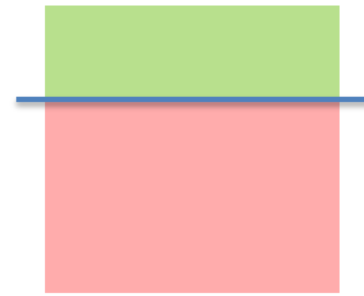


$\epsilon_1 = .3$
$\alpha_1 = .42$

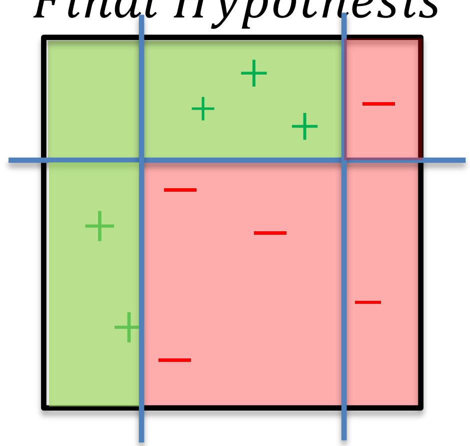
$\epsilon_2 = .21$
$\alpha_2 = .65$

$\epsilon_3 = .14$
$\alpha_3 = .92$

Final Hypothesis

$h(x) = sign$ [$.42$  $+.65$  $+.92$ ]

Final Hypothesis



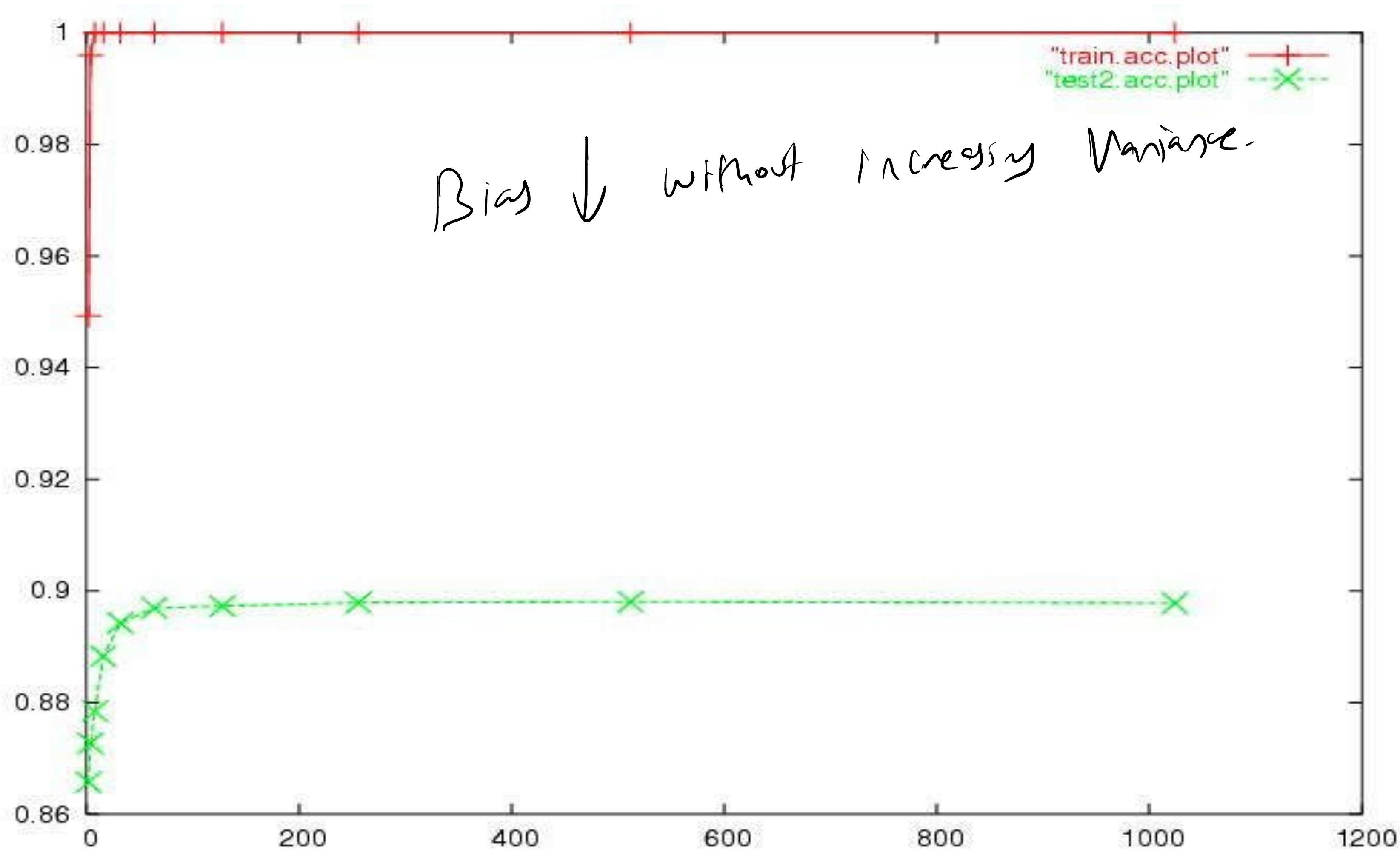
Boosting

Theorem: Let $Z_t = 2\sqrt{\epsilon_t \cdot (1 - \epsilon_t)}$ and $\gamma_t = \frac{1}{2} - \epsilon_t$.

$$\frac{1}{M} \sum_m 1_{h(x^{(m)}) \neq y^{(m)}} \leq \prod_{t=1}^{T} Z_t = \prod_{t=1}^{T} \sqrt{1 - 4\gamma_t^2}$$

So, even if all of the γ's are small positive numbers (i.e., can always find a weak learner), the training error goes to zero as T increases

Boosting Performance



AdaBoost in Practice

- Our description of the algorithm assumed that a set of possible hypotheses was given
  - In practice, the set of hypotheses can be built as the algorithm progress
- Example: build new decision tree at each iteration for the data set in which the m^{th} example has weight $w_m^{(t)}$
  - When computing information gain, compute the empirical probabilities using the weights

Summary of AdaBoost

- Algorithm is inherently sequential: the next tree is learnt based on the errors of the previous trees
- Data instances are weighed based on the errors of the samples
  - Training data samples that most models so far have gotten wrong will have a higher weight
  - Samples that most models have gotten correct will have a lower weights
- The model is trained on the weighted training set
- Subsequent trees will focus more on “hard” samples models so far have gotten wrong
- Each weak model/tree is set a weight based on its performance.

Gradient Boosting

- The idea is the same except that the errors here are propogated in a different way.
- First step: Fit a tree on the full training set (x = Features, y = Labels)
- Evaluate the model on the training set and compute "residuals":

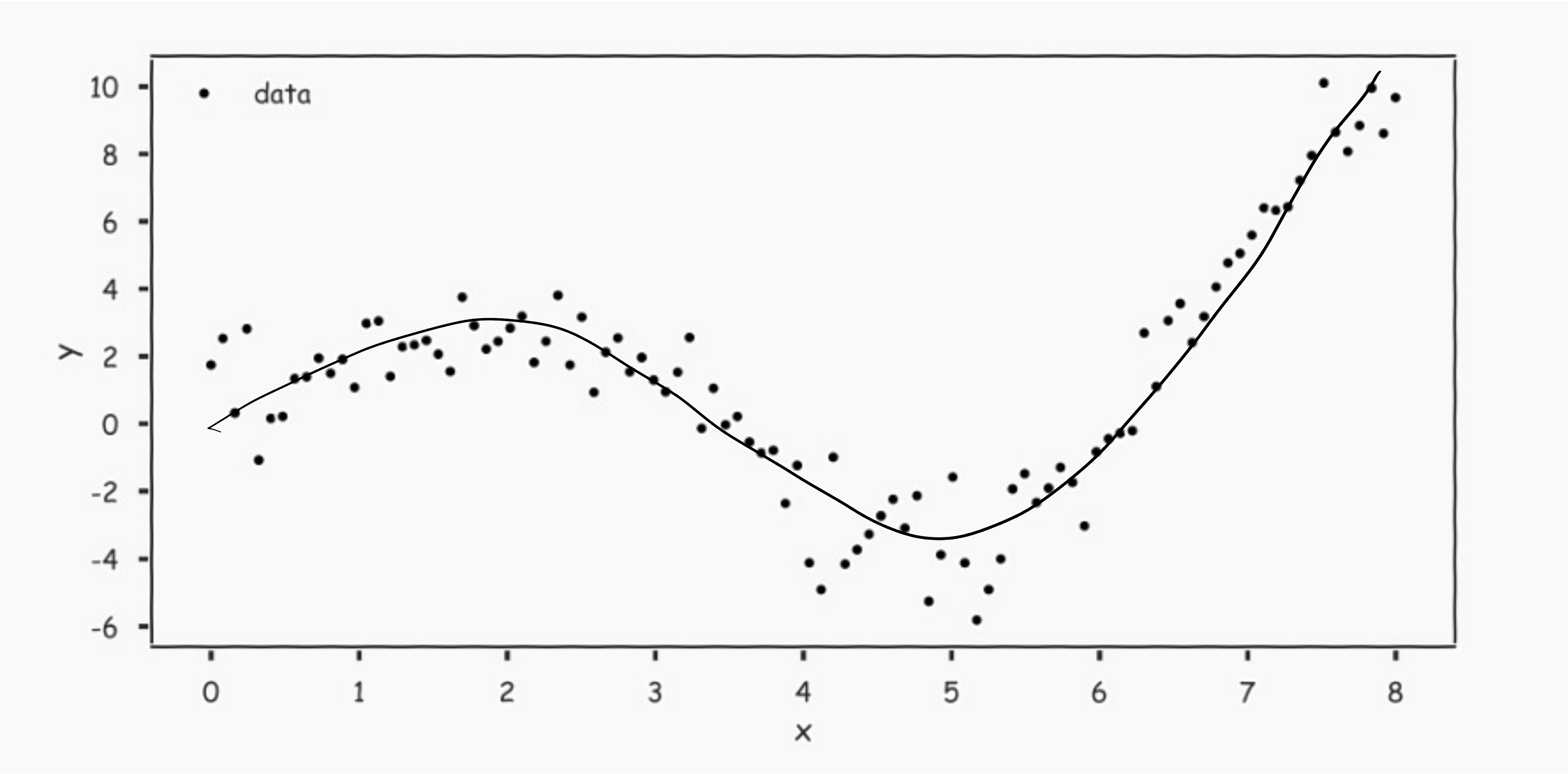
$$y' = y - \alpha f(x)$$

Where f(x) is the model prediction on x and α is the learning rate

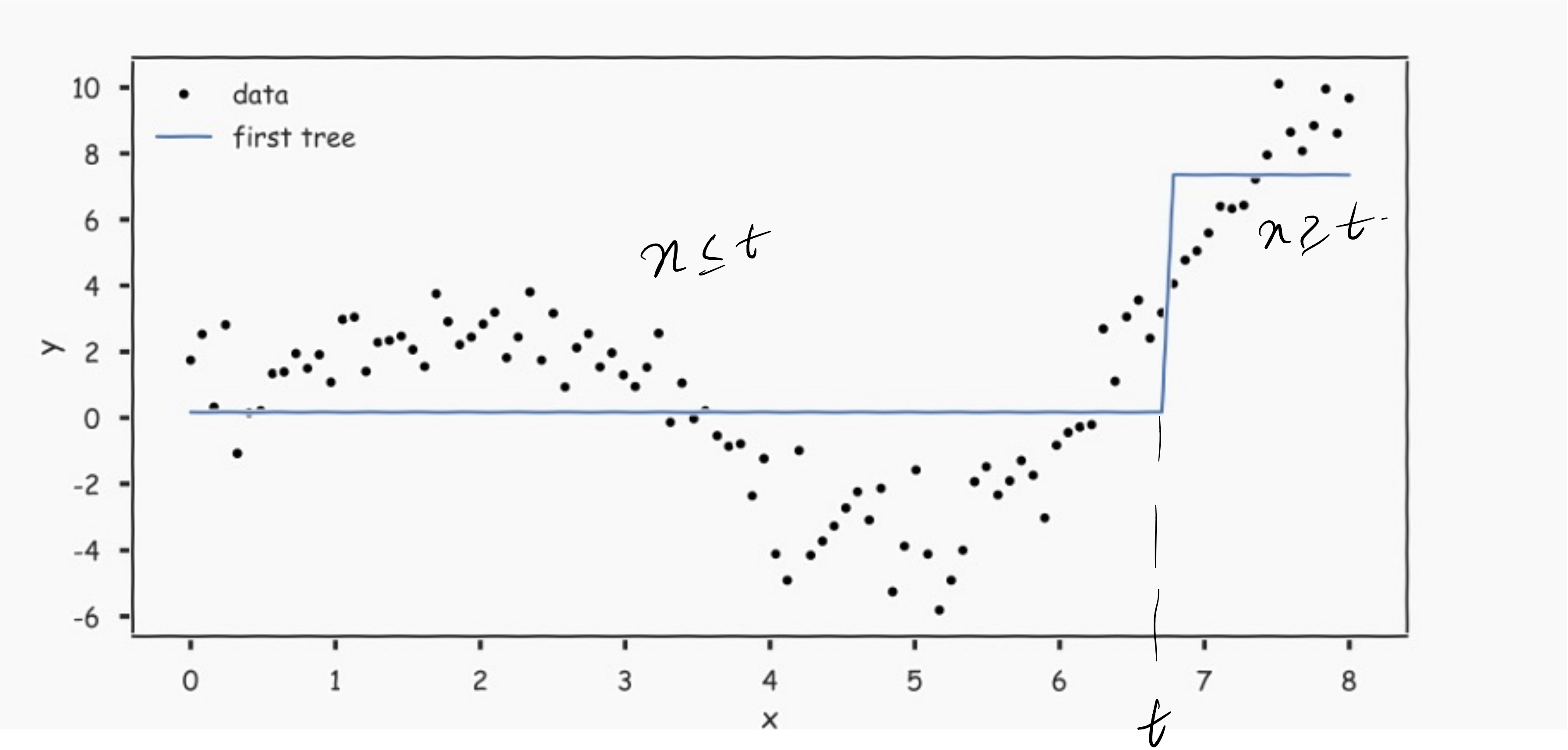
- Set the label $y = y'$ and train the model on the training set with x as features and y as labels.
- Repeat until a stopping condition

$$Loss = (y_{pred} - y_{true})^2$$

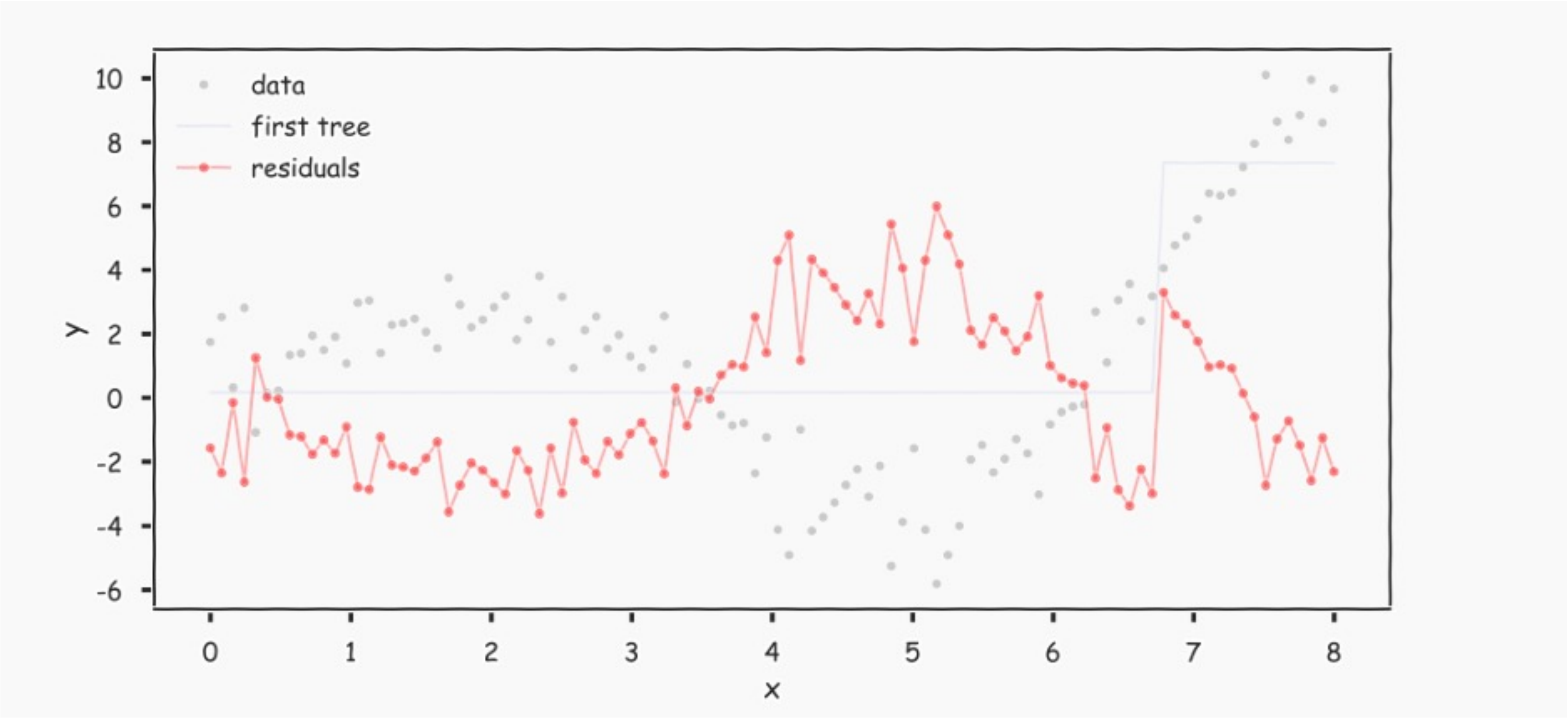
Illustrating Gradient Boosting



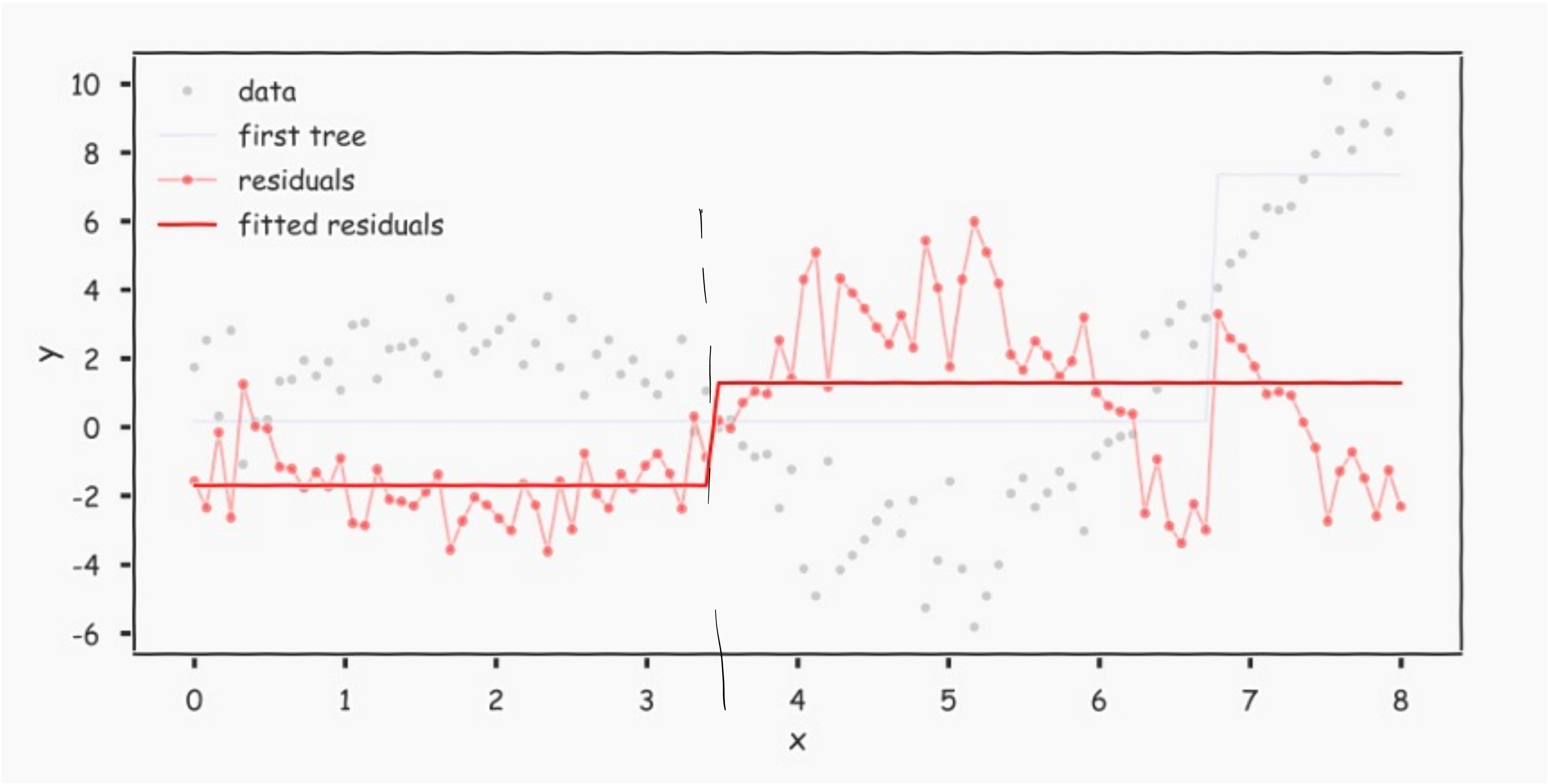
Illustrating Gradient Boosting



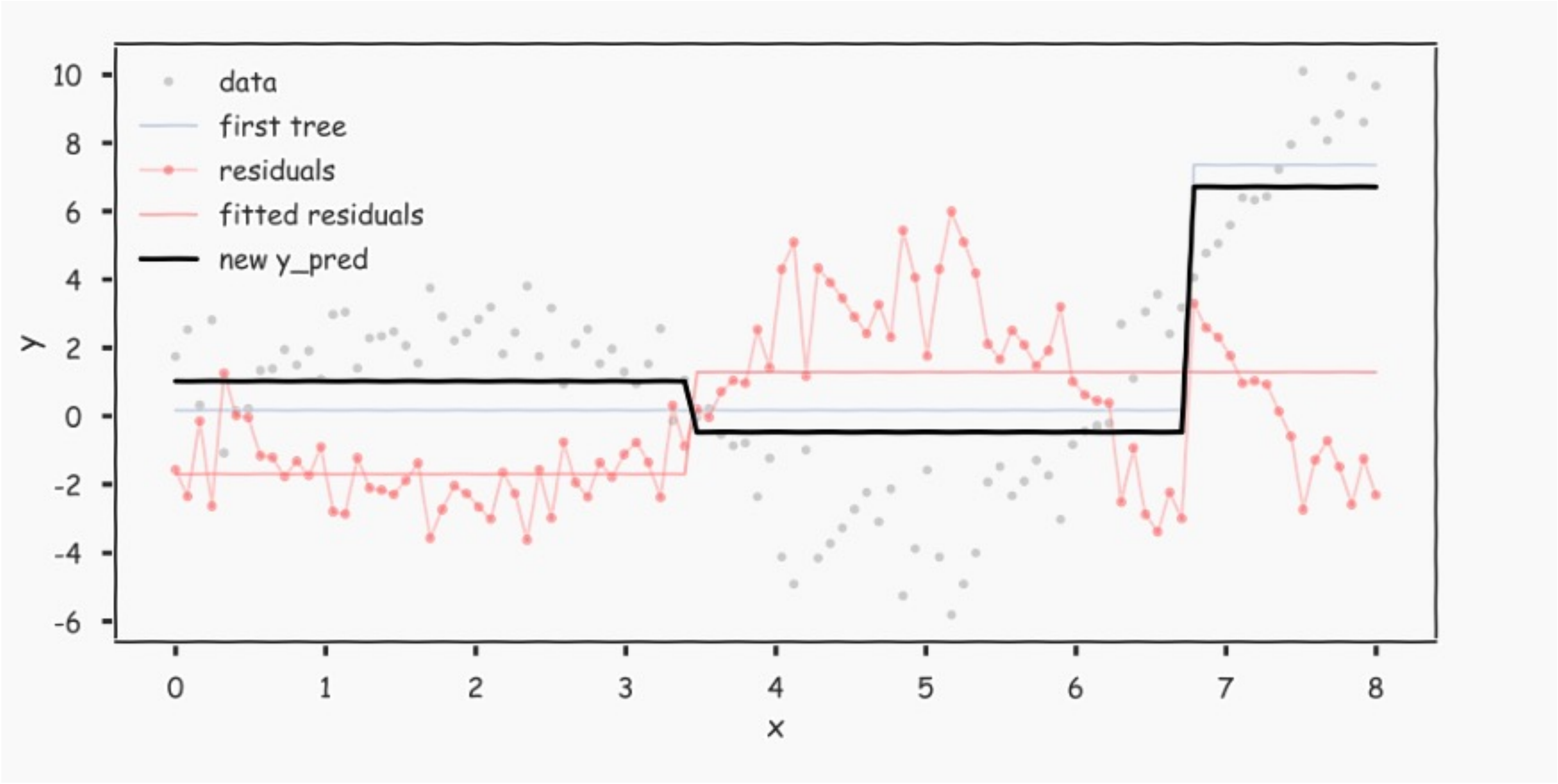
Illustrating Gradient Boosting



Illustrating Gradient Boosting



Illustrating Gradient Boosting



Why does gradient boosting work?

- We are training a simple model at every step so the model inherently as a low ~~bias~~ variance.
- Over time we seek to reduce the "residual error" and fit the data on the residual
- Focus on what previous models have not modeled properly.
- Why Gradient Boosting?
  - Essentially we are minimizing a loss function $L(f(x), y)$ iteratively
  - At every round the residual $y' = -\partial L/\partial f$
  - In the case of the Squared loss, this is exactly the difference between y and f(x)
  - Can be different for other losses (e.g, Logistic Loss)

$[f(x) - y]^2$

$\rightarrow (f(x) - y)$

$= y - f(x)$

Gradient Boosting vs Adaptive Boosting

- Both Gradient Boosting and AdaBoost try to fit models which can fix weaknesses of previous models
- AdaBoost does it by weighing samples, Gradient Boosting does it by computing residuals
- AdaBoost weighs the weak models at the end. Gradient Boosting does not assign any weights (weights are just 1)
- Adaboost uses exponential loss, which makes it very sensitive to outliers since misclassified points are exponentially more influential due to increasing weights.
- Gradient Boosting can use any differentiable loss function, not just exponential loss. This includes logistic loss for classification or squared error for regression, making it more flexible in handling a variety of error distributions and less sensitive to outliers than Adaboost.

When to Use One vs the Other

- **Adaboost**: ← only classifⁿ (Binary)
  - When you have a binary classification problem, especially if you believe that your problem can benefit from more focus on misclassified instances.
  - Useful when the decision boundary is very irregular, as the focus on misclassified instances can aggressively adapt to it.
  - More sensitive to outliers and noise since weights are increased on misclassified points
- **Gradient Boosting**: ← classfⁿ & Regression
  - When you need a model for both regression and classification tasks, or you need to minimize a specific loss function.
  - More suitable for problems where the predictive power can be incrementally improved by focusing on residuals or mistakes of previous models.
  - Typically performs better than Adaboost when configured correctly but requires careful tuning (more hyper-parameters).
  - Gradient Boosting more robust to AdaBoost in noise and outliers.

Boosting vs. Bagging

- Bagging doesn't work well with stable models
  - Boosting might still help

Variance Low.

- Boosting (specifically adaBoost) might hurt performance on noisy datasets
  - Bagging doesn't have this problem
- On average, boosting improves classification accuracy more than bagging, but it is also more common for boosting to hurt performance
- Bagging is easier to parallelize
- Both ensemble methods have added overhead required to train multiple classifiers

Boosting Beyond Binary Classification

AdaBoost

- Slightly more complicated
  - Want to select weak learners that are better than random guessing, but there are many different ways to do better than random
  - A hypothesis space is boostable if there exists a baseline measure, that is slightly better than random, such that you can always find a hypothesis that outperforms the baseline
  - Can be boostable with respect to some baselines but not others

Gradient Boosting can be adapted ← Multiclass loss

Additional Reading

- http://cs229.stanford.edu/extra-notes/boosting.pdf
- https://web.stanford.edu/~hastie/Papers/buehlmann.pdf
- http://web.stanford.edu/~hastie/TALKS/boost.pdf
- https://cseweb.ucsd.edu/~yfreund/papers/IntroToBoosting.pdf