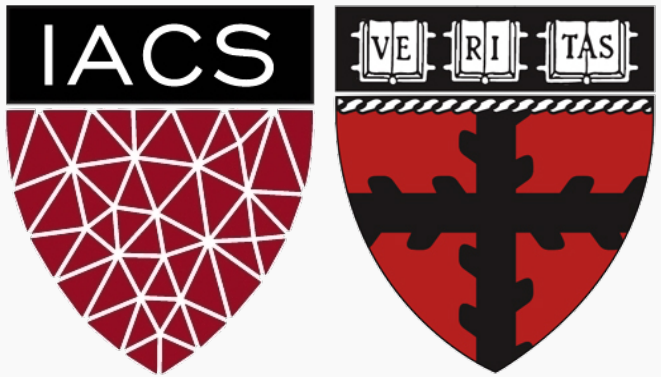
Gradient Boosting

CS109A Introduction to Data Science

Pavlos Protopapas, Natesh Pillai



Can boosting still have problems with overfitting or underfitting?

When should we choose boosting over random forest?

How do we differentiate between using bagging and boosting?

Does random forest deal well with imbalanced data?

Does having feature importance mean that the model is an interpretable one?

When should we use boosting over RFs?

Why is using false-positive rate and true-positive rates not as good as precision and recall for evaluating models with unbalanced data?

How do we overcome the challenge of overfitting with boosting given that several learners are chained together to reduce bias?

How do adaboost and gradient boost differ?

Comparison of Models:

Choosing the right model isn't just about minimizing the test errors.
We want extra insights from our models:

	Has a fixed form f(x) parametric	easy to interpret	computational complexity
Linear Regression	YES	YES	LOW
Polynomial Regression	YES	NO	LOW
Regression Trees	NO	YES	LOW
Bagging and RF	NO	YES/NO	MEDIUM
K-nearest Neighbors	NO	YES	HIGH

"Can a set of weak learners create a single strong learner?"

Leslie Gabriel Valiant

How many jelly beans do you see?

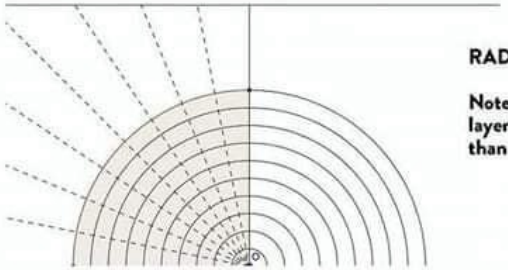

THEJENKINSCOMIC
> do or do not:
()myMethod
cept:
pass
finally:
SYNTAX ERROR

CUTTING ONIONS

(ALL VIEWS ARE CROSS-SECTIONS OF AN ONION HALF)

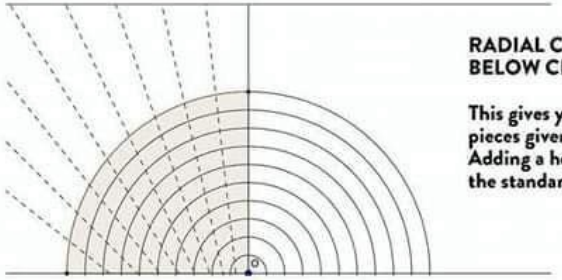
USING ALL VERTICAL CUTS

Note how long the pieces towards the outside edge of the onion get. This is why horiziontal cuts are also necessary, especially when the initial cuts are perfectly vertical.



RADIAL CUTS AIMED AT CENTER.

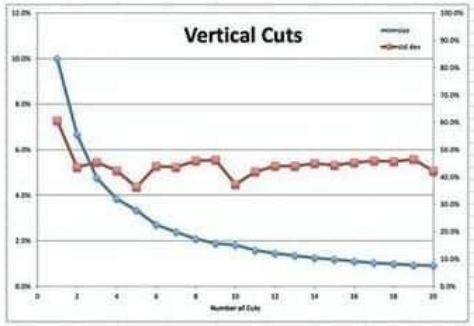
Note that the pieces in the center layer of onion are much smaller than those in the outer layers.

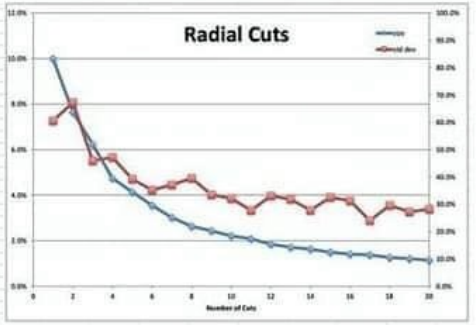


RADIAL CUTS AIMED 60% BELOW CENTER.

This gives you the most evenly-sized onion pieces given the same number of cuts. Adding a horitzonal slice further reduces the standard deviation of piece size.

STANDARD DEVIATION IN PIECE SIZE WITH VERTICAL CUTS VS. 60% RADIAL CUTS





Motivation for Boosting

Question: Could we address the shortcomings of single decision trees models in some other way?

For example, rather than performing variance reduction on complex trees, can we decrease the bias of simple trees - make them more expressive?

Can we learn from our mistakes?

A solution to this problem, making an expressive model from simple trees, is another class of ensemble methods called ***boosting***.

Gradient Boosting

The key intuition behind boosting is that one can take an ensemble of simple models $\{T_h\}_{h \in H}$ and additively combine them into a single, more complex model.

Each model T_h might be a poor fit for the data, but a linear combination of the ensemble:

$$T = \sum_h \lambda_h T_H$$

can be expressive/flexible.

Question: But which models should we include in our ensemble? What should the coefficients or weights in the linear combination be?

Gradient Boosting: the algorithm

Gradient boosting is a method for iteratively building a complex regression model T by adding simple models.

Each new simple model added to the ensemble compensates for the weaknesses of the current ensemble.

Gradient Boosting: the algorithm

1. Fit a simple model $T^{(0)}$ on the training data

$$\{(x_1, y_1), \ldots, (x_N, y_N)\}$$

Set $T \leftarrow T^{(0)}$.

Compute the residuals $\{r_1, \ldots, r_N\}$ for T.

2. Fit a simple model, $T^{(1)}$, to the current ***residuals***, i.e. train using

$$\{(x_1, r_1), \ldots, (x_N, r_N)\}$$

3. Set $T \leftarrow T + \lambda T^{(1)}$

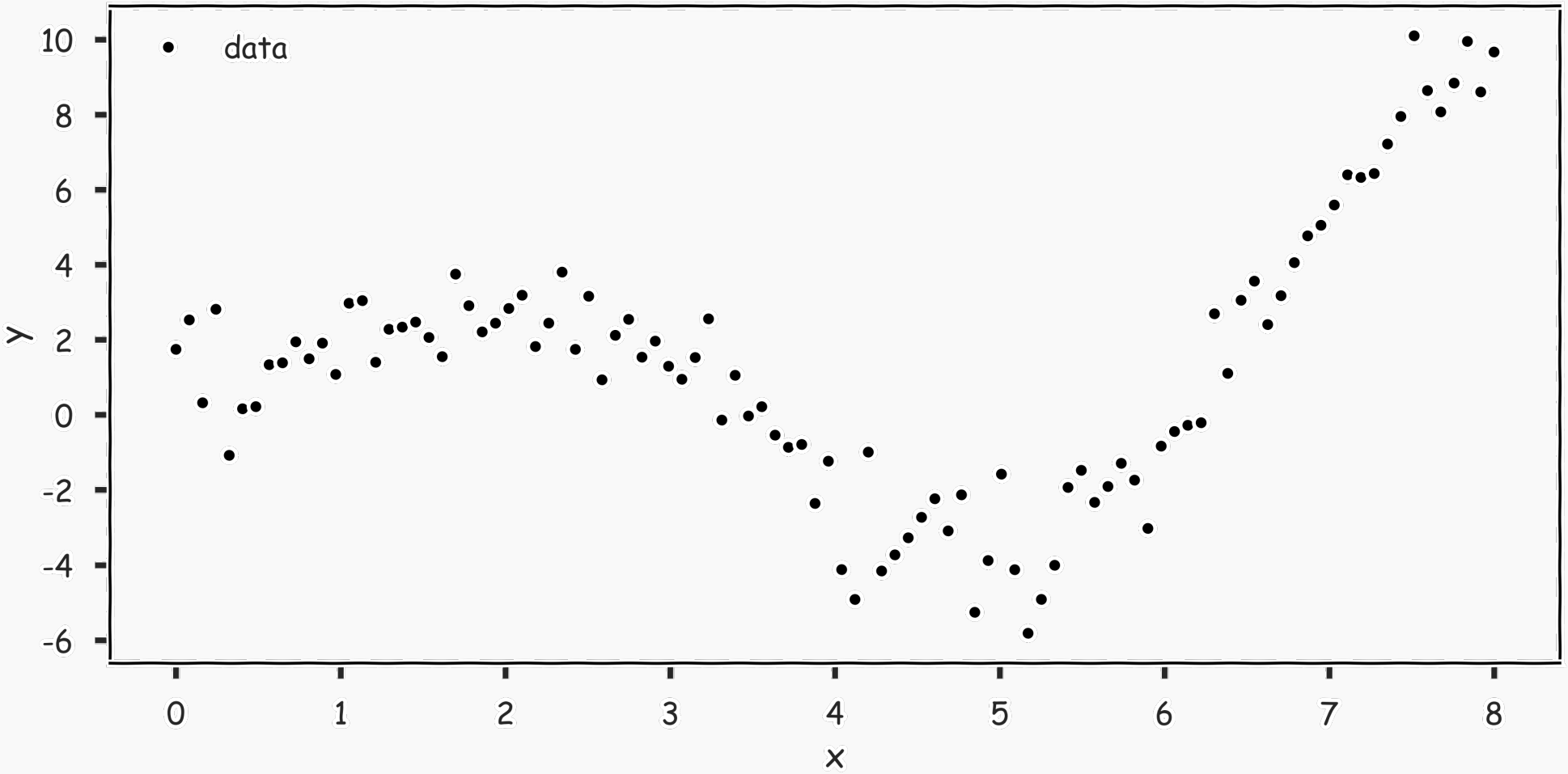
4. Compute residuals, set $r_n \leftarrow r_n - \lambda T^i(x_n), \ n = 1, \ldots, N$

5. Repeat steps 2-4 until ***stopping*** condition met.

where λ is a constant called the ***learning rate***.

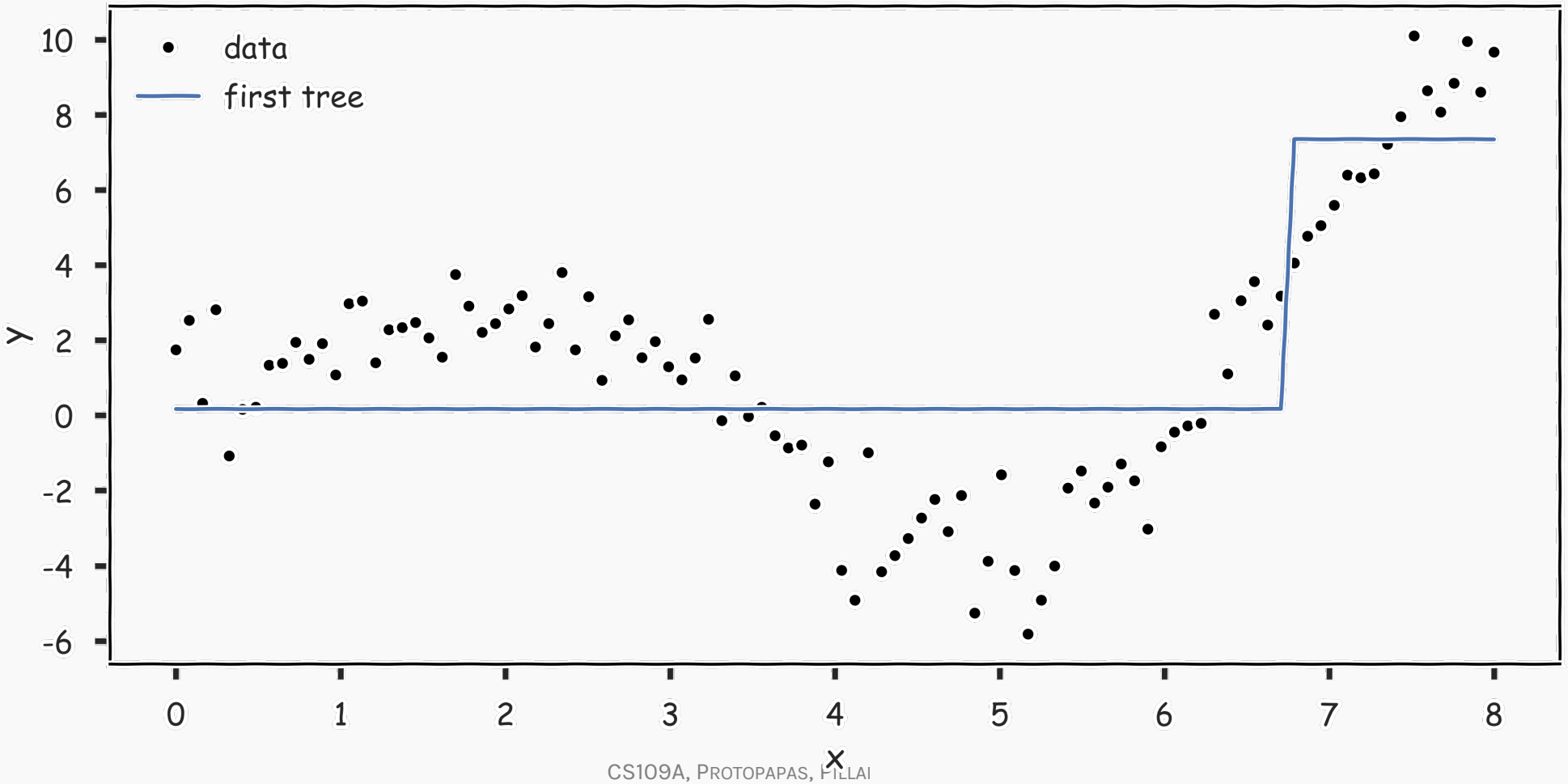
Gradient Boosting: illustration

training data: $\{(x_1, y_1), \ldots, (x_N, y_N)\}$



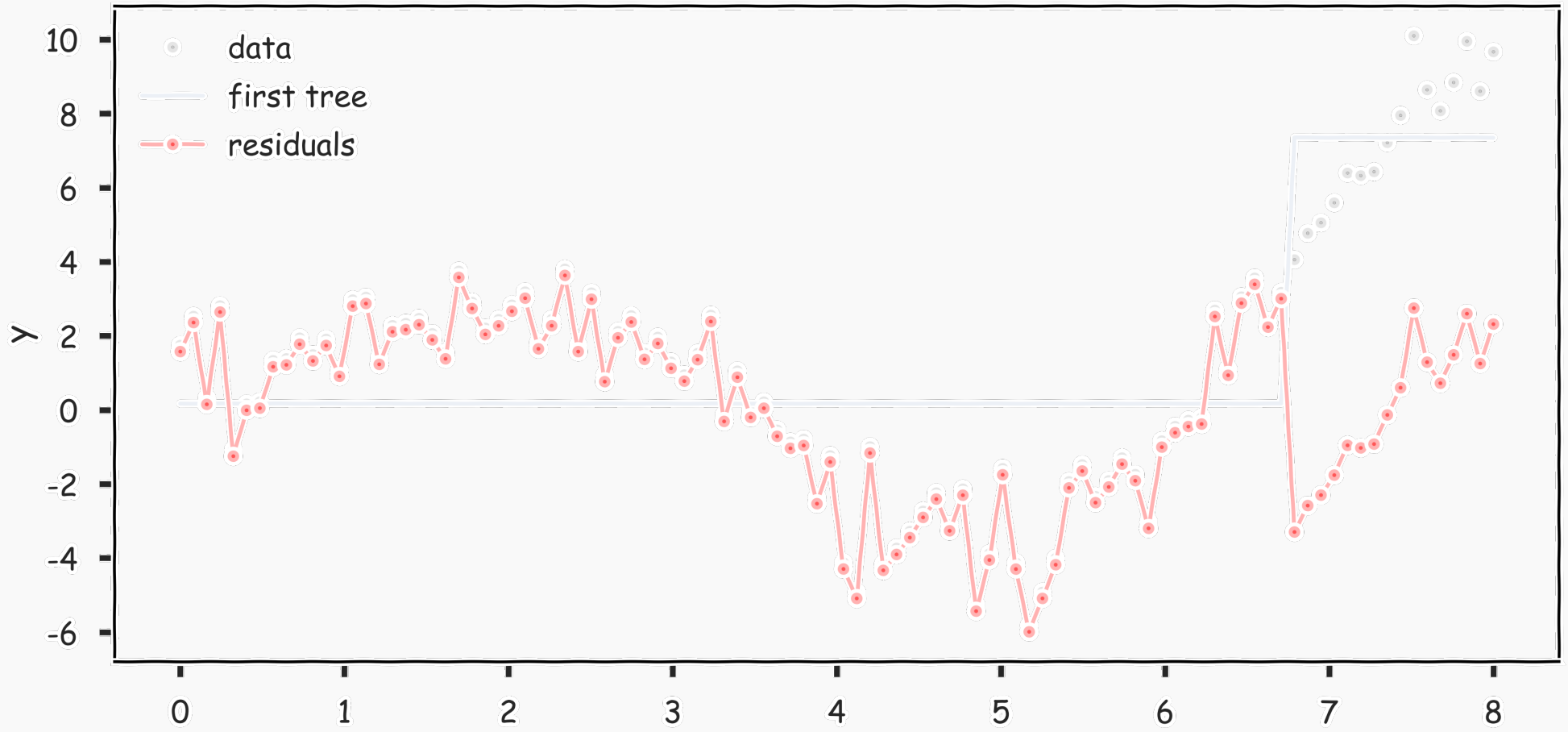
Gradient Boosting: illustration

Fit a simple model $T^{(0)}$



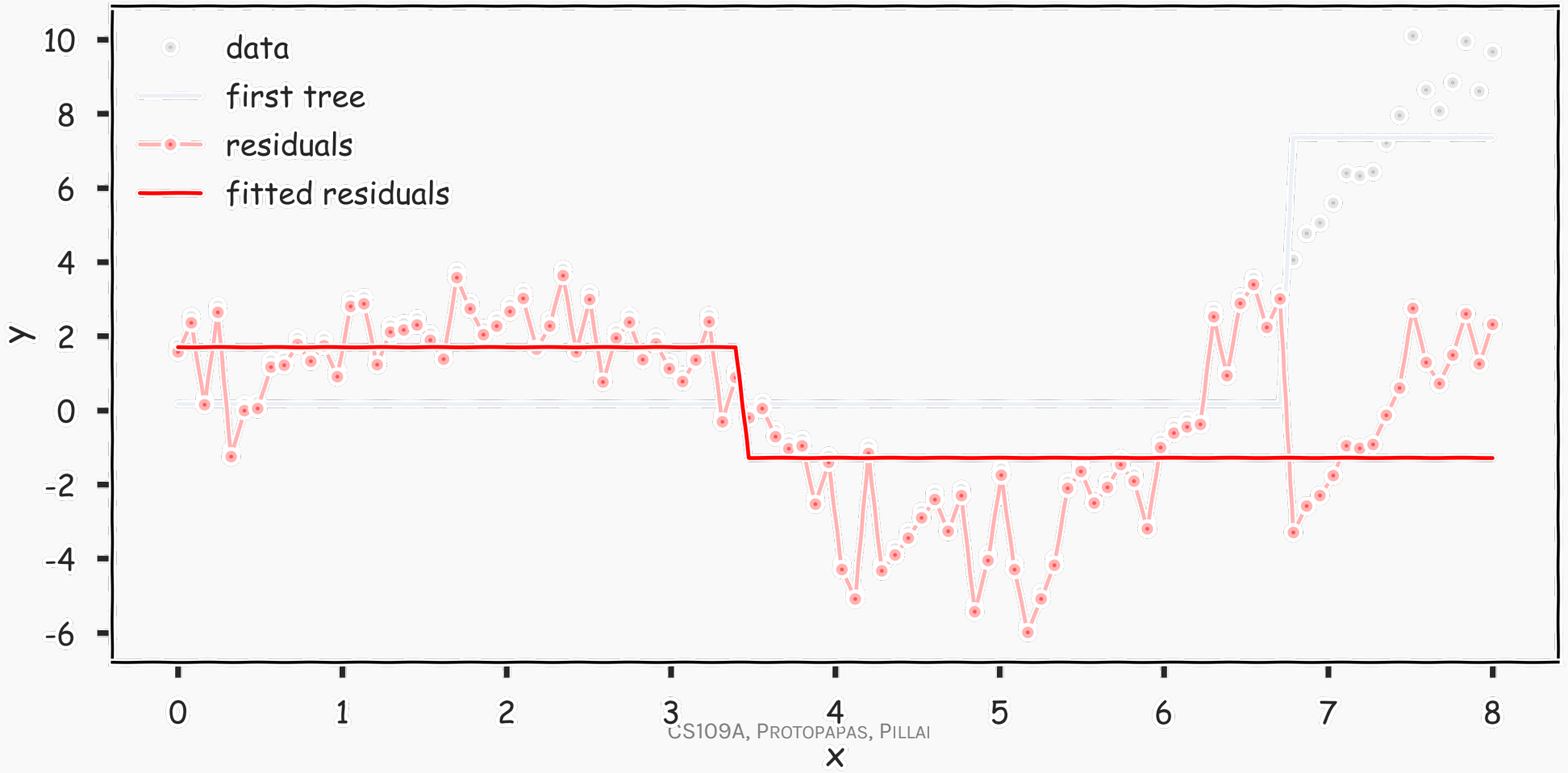
Gradient Boosting: illustration

Compute the residuals $\{r_1, \ldots, r_N\}$ for T.



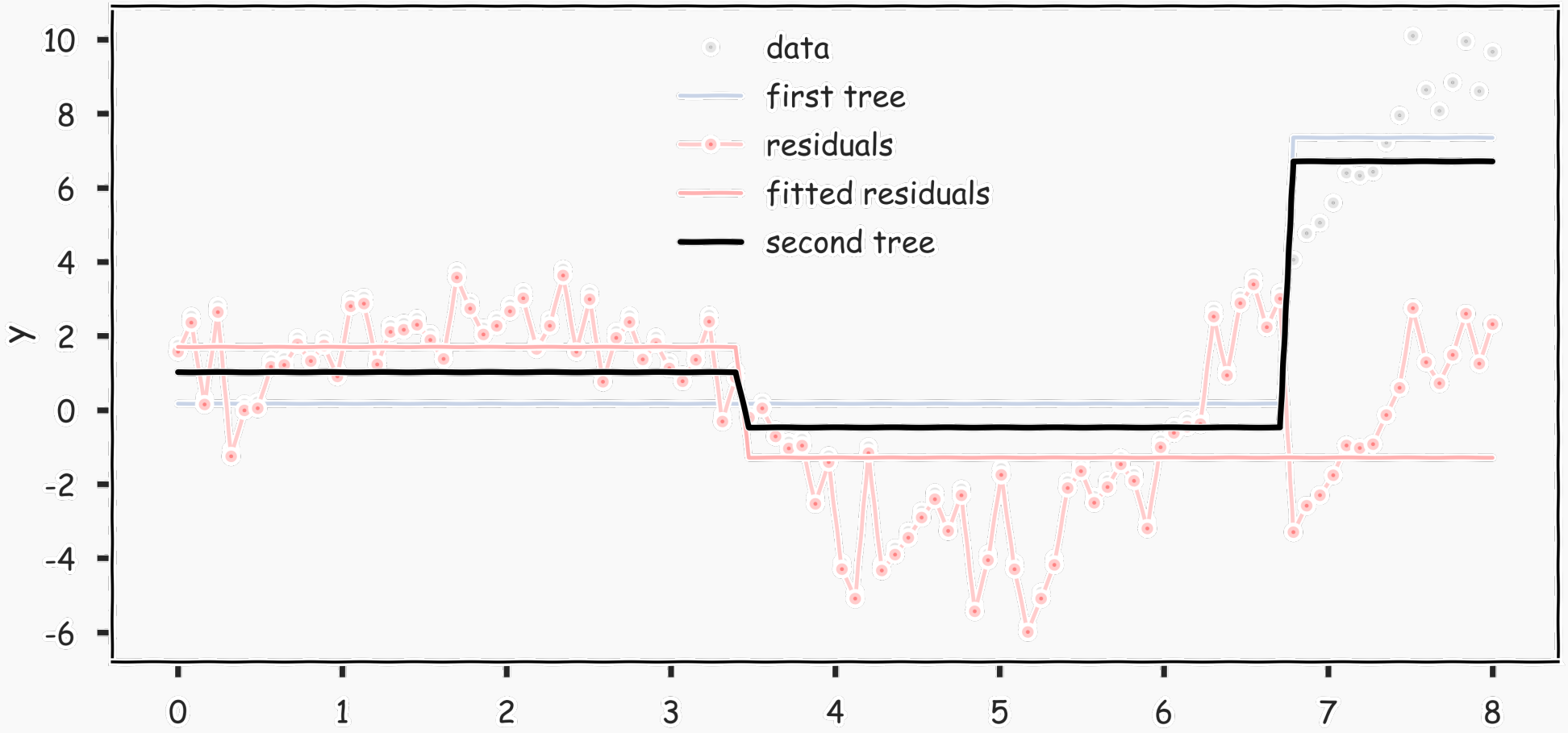
Gradient Boosting: illustration

train using: $\{(x_1, r_1), \ldots, (x_N, r_N)\}$



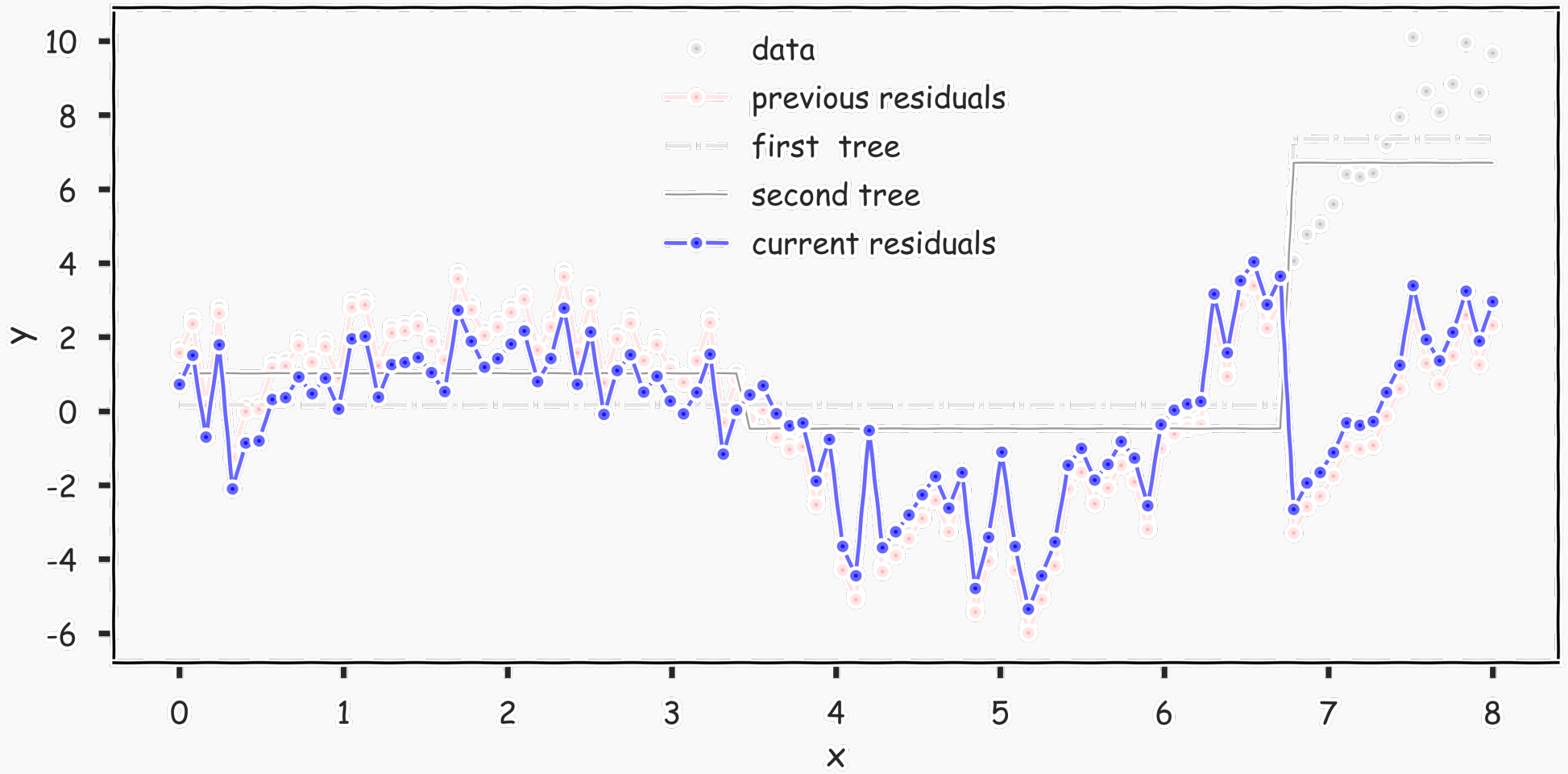
Gradient Boosting: illustration

Set $T \leftarrow T + \lambda T^{(1)}$

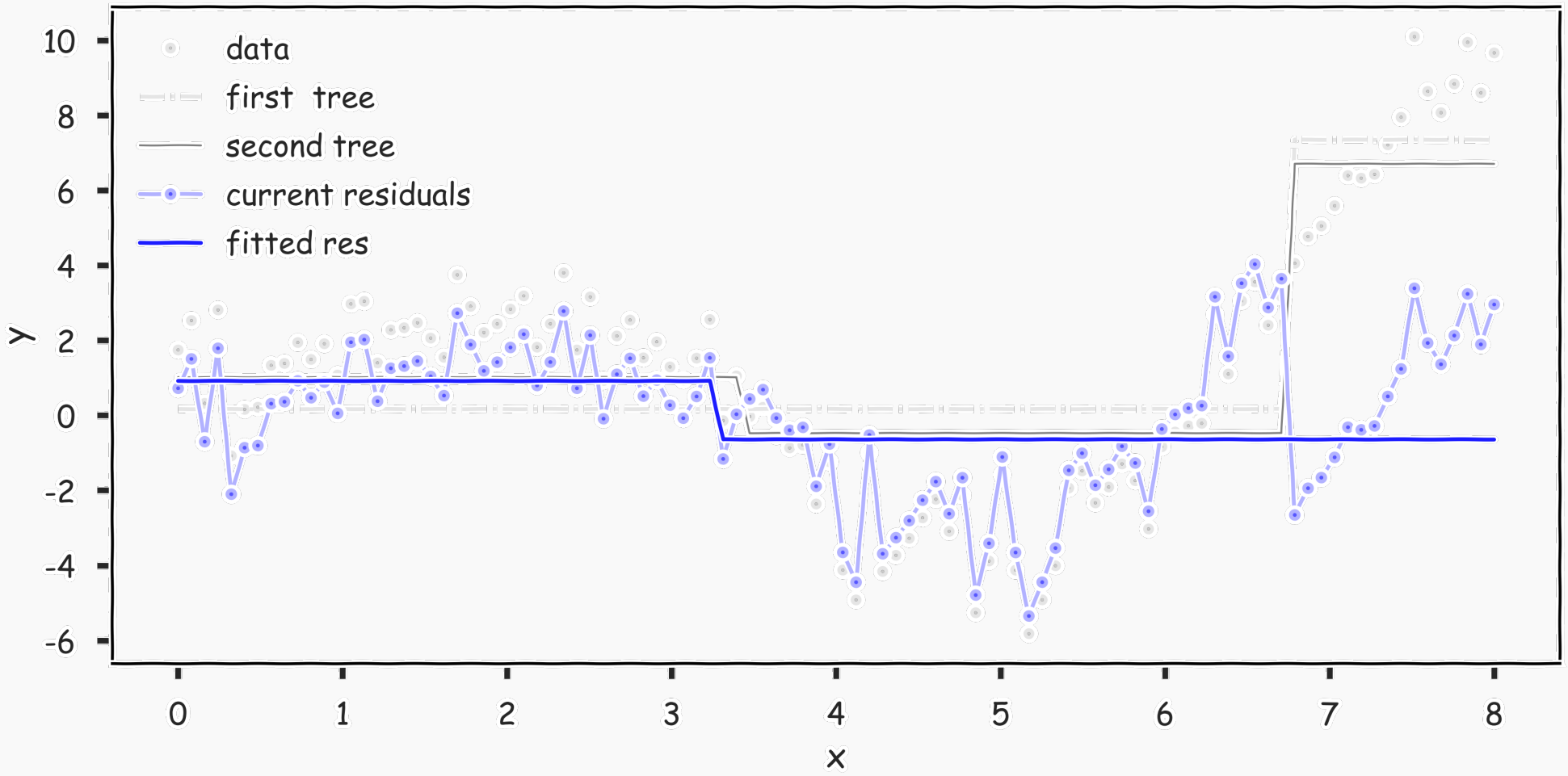


Gradient Boosting: illustration

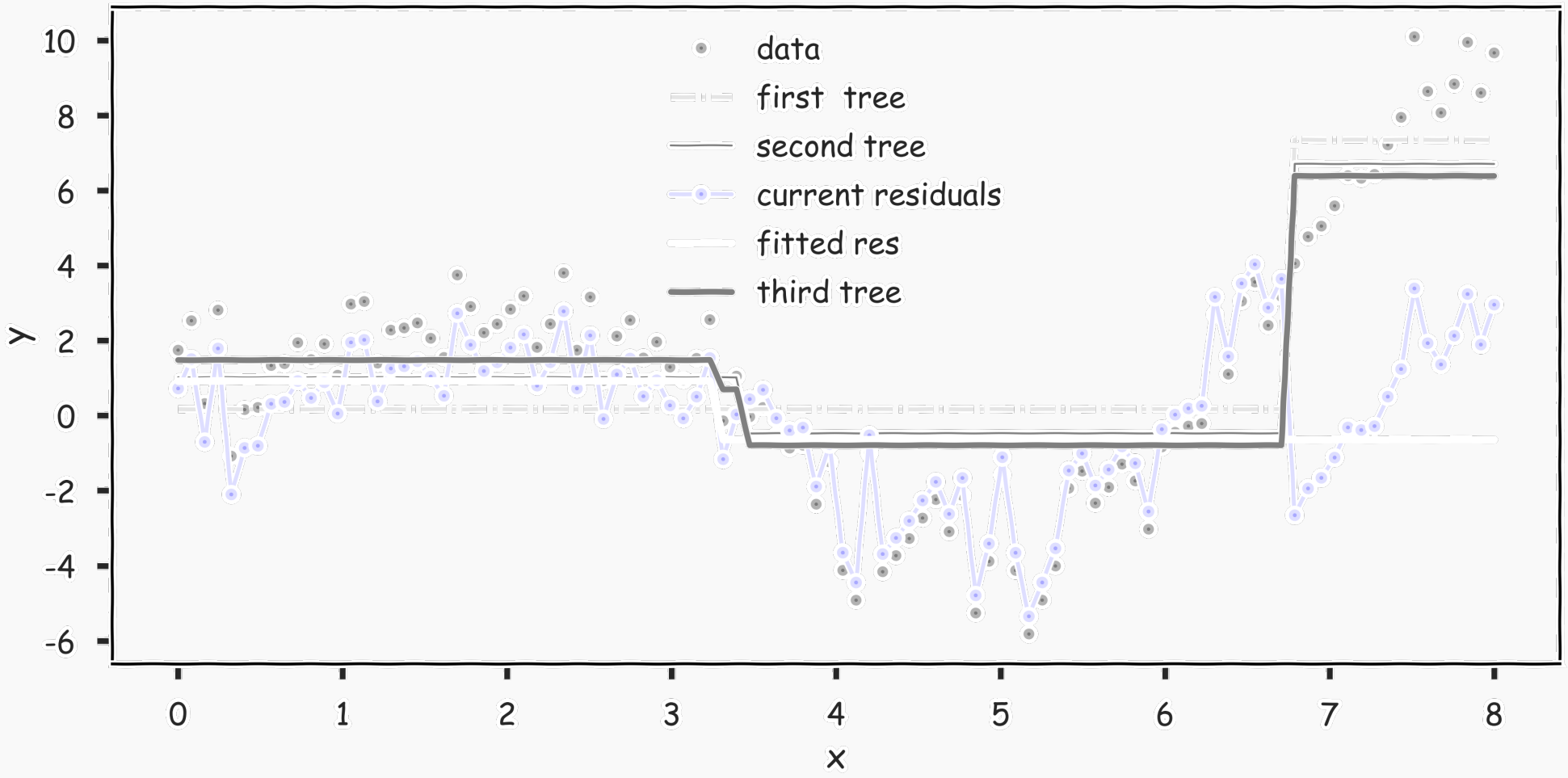
$$r_n \leftarrow r_n - \lambda T^i(x_n)$$

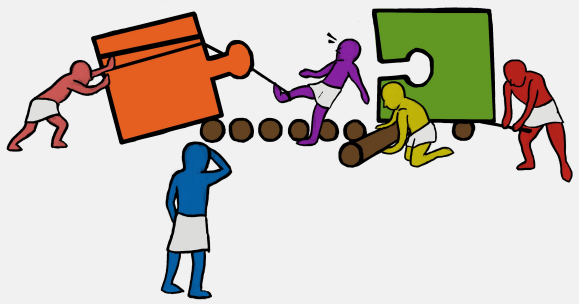


Gradient Boosting: illustration



Gradient Boosting: illustration

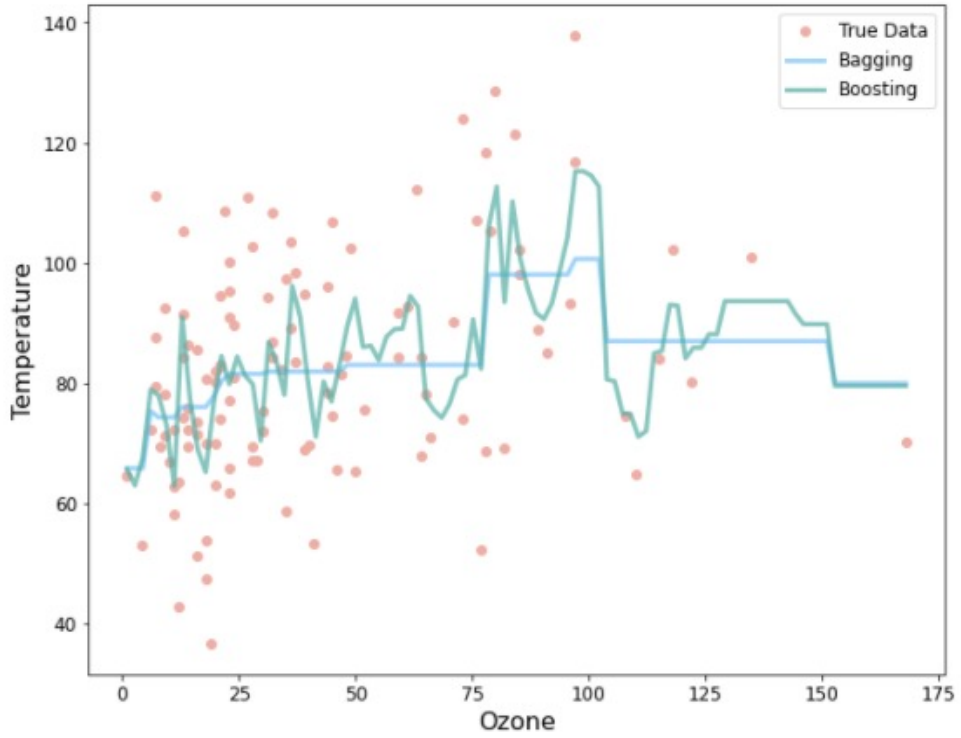




Description

🏋️‍♀️👨‍🏫 Exercise: Regression with Boosting

The goal of this exercise is to understand *Gradient Boosting Regression.*



Instructions:

Why Does Gradient Boosting Work?

Intuitively, each simple model $T^{(i)}$ we add to our ensemble model T, models the errors of T.

Thus, with each addition of $T^{(i)}$, the residual is reduced

$$r_n - \lambda T^{(i)}(x_n)$$

Note that gradient boosting has a tuning parameter, λ.

If we want to easily reason about how to choose λ and investigate the effect of λ on the model T, we need a bit more mathematical formalism.

In particular, how can we effectively descend through this optimization via an iterative algorithm?

We need to formulate gradient boosting as a type of ***gradient descent***.

Review: A Brief Sketch of Gradient Descent

In optimization, when we wish to minimize a function, called the ***objective function (or loss function)***, over a set of variables, we compute the partial derivatives of this function with respect to the variables.

If the partial derivatives are sufficiently simple, one can analytically find a common root - i.e. a point at which all the partial derivatives vanish; this is called a ***stationary point.***

If the objective function has the property of being ***convex***, then the stationary point is precisely the min.

Review: A Brief Sketch of Gradient Descent the Algorithm

In practice, our objective functions are complicated and analytically find the stationary point is intractable.

Instead, we use an iterative method called ***gradient descent***

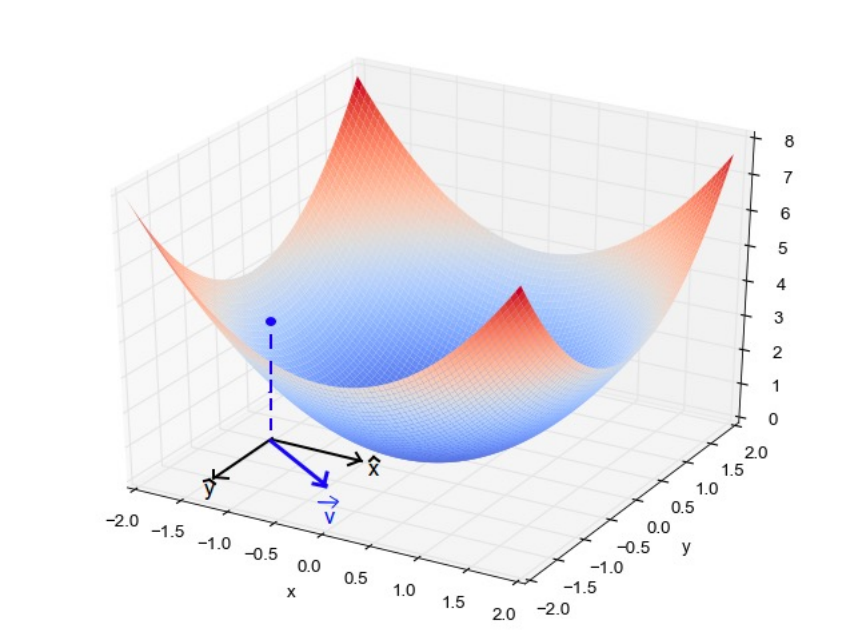
1. Initialize the variables at any value:
$$x = [x_1, ..., x_J]$$
2. Take the gradient of the objective function at the current variable values:
$$\nabla f(x) = \left[\frac{\partial f}{\partial x_1}(x), \ldots, \frac{\partial f}{\partial x_J}(x)\right]$$
3. Adjust the variables values by some negative multiple of the gradient:
$$x \leftarrow x - \lambda \nabla f(x)$$

   The factor λ is often called the learning rate.

Why Does Gradient Descent Work?

Claim: If the function is convex, this iterative methods will eventually move x close enough to the minimum, for an appropriate choice of λ.

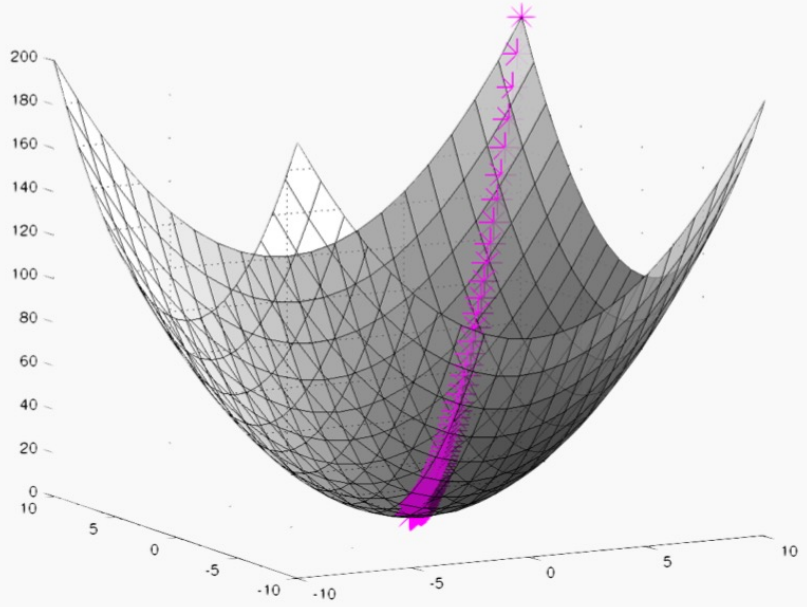
Why does this work? Recall, that as a vector, the gradient at at point gives the direction for the greatest possible rate of increase.



Why Does Gradient Descent Work?

Subtracting a λ multiple of the gradient from x, moves x in the ***opposite*** direction of the gradient (hence towards the steepest decline) by a step of size λ.

If f is convex, and we keep taking steps descending on the graph of f, we will eventually reach the minimum.



Gradient Boosting as Gradient Descent

Often in regression, our objective is to minimize the MSE

$$\text{MSE}(\hat{y}_1, \ldots, \hat{y}_N) = \frac{1}{N} \sum_{i=1}^{N} (y_i - \hat{y}_i)^2$$

Treating this as an optimization problem, we can try to directly minimize the MSE with respect to the predictions

$$\begin{aligned} \nabla \text{MSE} &= \left[\frac{\partial \text{MSE}}{\partial \hat{y}_1}, \ldots, \frac{\partial \text{MSE}}{\partial \hat{y}_N} \right] \\ &= -2 \left[y_1 - \hat{y}_1, \ldots, y_N - \hat{y}_N \right] \\ &= -2 \left[r_1, \ldots, r_N \right] \end{aligned}$$

The update step for gradient descent would look like

$$\hat{y}_n \leftarrow \hat{y}_n + \lambda r_n, \quad n = 1, \ldots, N$$

Gradient Boosting as Gradient Descent (cont.)

There are two reasons why minimizing the MSE with respect to $\hat{y}_n$'s is not interesting:

- We know where the minimum MSE occurs: $\hat{y}_n = y_n$, for every n.
- Learning sequences of predictions, $\hat{y}_n^1, \ldots, \hat{y}_n^i, \ldots$, does not produce a model. The predictions in the sequences do not depend on the predictors!

Gradient Boosting as Gradient Descent (cont.)

$$\hat{y}_n \leftarrow \hat{y}_n + \lambda r_n$$

The solution is to change the update step in gradient descent. Instead of using the gradient - the residuals - we use an ***approximation*** of the gradient that depends on the predictors:

$$\hat{y} \leftarrow \hat{y}_n + \lambda\, \hat{r}_n(x_n), \qquad n = 1, \ldots, N$$

In gradient boosting, we use a simple model to approximate the residuals, $\hat{r}_n(x_n)$, in each iteration.

Motto: gradient boosting is a form of gradient descent with the MSE as the loss (objective) function.

Technical note: note that gradient boosting is descending in a space of models or functions relating x_n to y_n!

Gradient Boosting as Gradient Descent (cont.)

But why do we care that gradient boosting is gradient descent?

By making this connection, we can import the massive amount of techniques for studying gradient descent to analyze gradient boosting.

For example, we can easily reason about how to choose the learning rate λ in gradient boosting.

Choosing a Learning Rate

Under ideal conditions, gradient descent iteratively approximates and converges to the optimum.

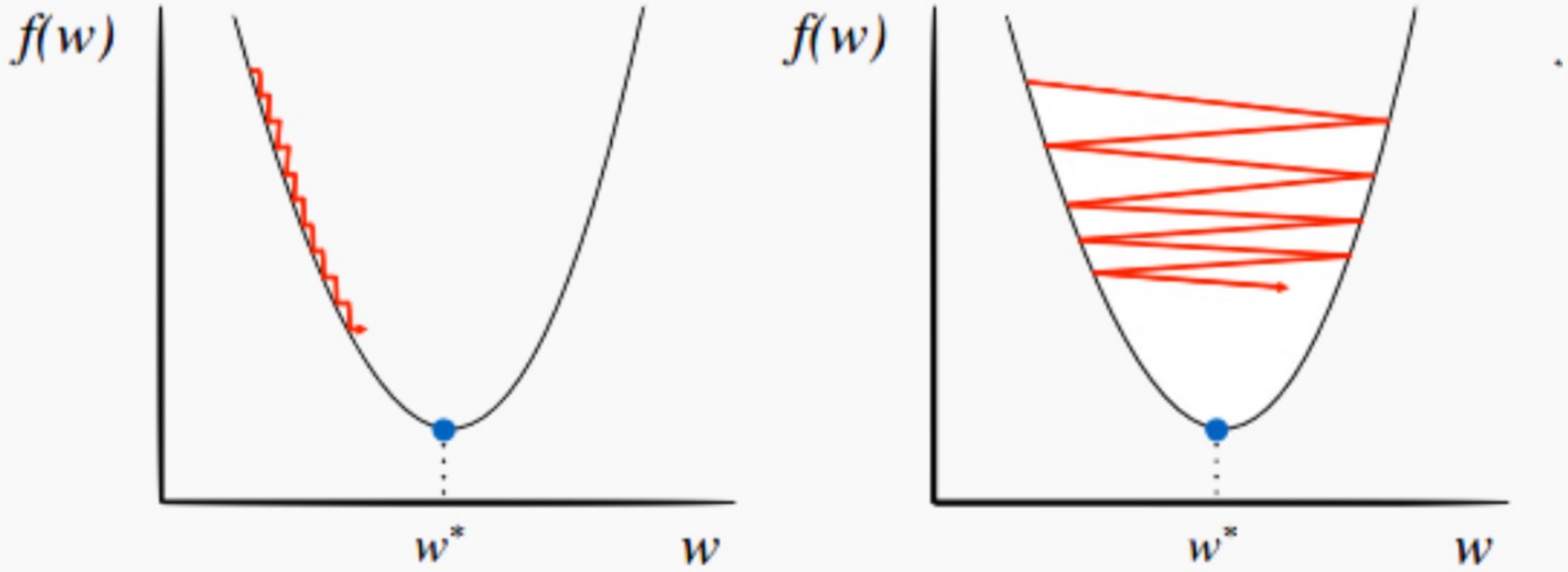
When do we terminate gradient descent?

- We can limit the number of iterations in the descent. But for an arbitrary choice of maximum iterations, we cannot guarantee that we are sufficiently close to the optimum in the end.
- If the descent is stopped when the updates are sufficiently small (e.g. the residuals of T are small), we encounter a new problem: the algorithm may never terminate!

Both problems have to do with the magnitude of the learning rate, λ.

Choosing a Learning Rate

For a constant learning rate, λ, if λ is too small, it takes too many iterations to reach the optimum.



If λ is too large, the algorithm may 'bounce' around the optimum and never get sufficiently close.

Choosing a Learning Rate

Choosing λ:

- If λ is a constant, then it should be tuned through cross validation.
- For better results, use a variable λ. That is, let the value of λ depend on the gradient

$$\lambda = h(\|\nabla f(x)\|),$$

where $\|\nabla f(x)\|$ is the magnitude of the gradient, $\nabla f(x)$. So

- around the optimum, when the gradient is small, λ should be small
- far from the optimum, when the gradient is large, λ should be larger

Thank you