CS550: Advanced Data Analytics

ML using Ensembles

Based on lecture slides for
Stanford CS226, Harvard CS109
Data Mining Book etc.

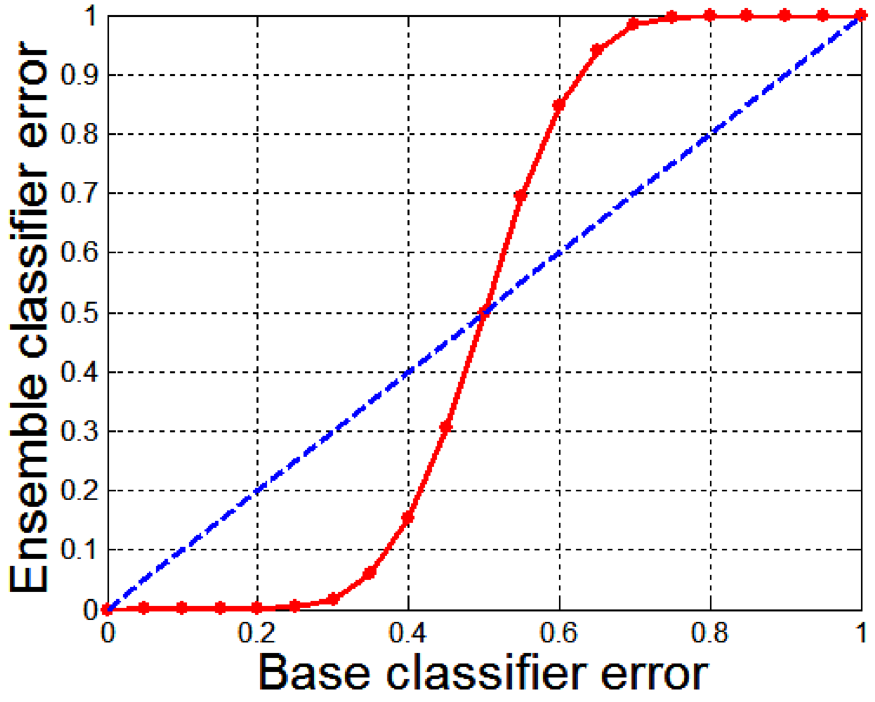
Instructor: Dr. Gagan Raj

Today's Class: Ensembles Motivation

- Imagine a team of engineers working on solving a problem
  - Each of them have different strengths and together they can do better
  - This is roughly the motivation in building the ensemble of ML models
- If we have multiple weak classifiers, can we expect their combination to do better?
  - Yes, if they are good in different areas. We combine their predictions
  - Combination of learners to increase accuracy and reduce overfitting.
  - Main causes of error in learning: noise, bias, variance. Ensembles help reduce those factors.
  - Improves stability of machine learning models.

Why Ensemble Methods work?

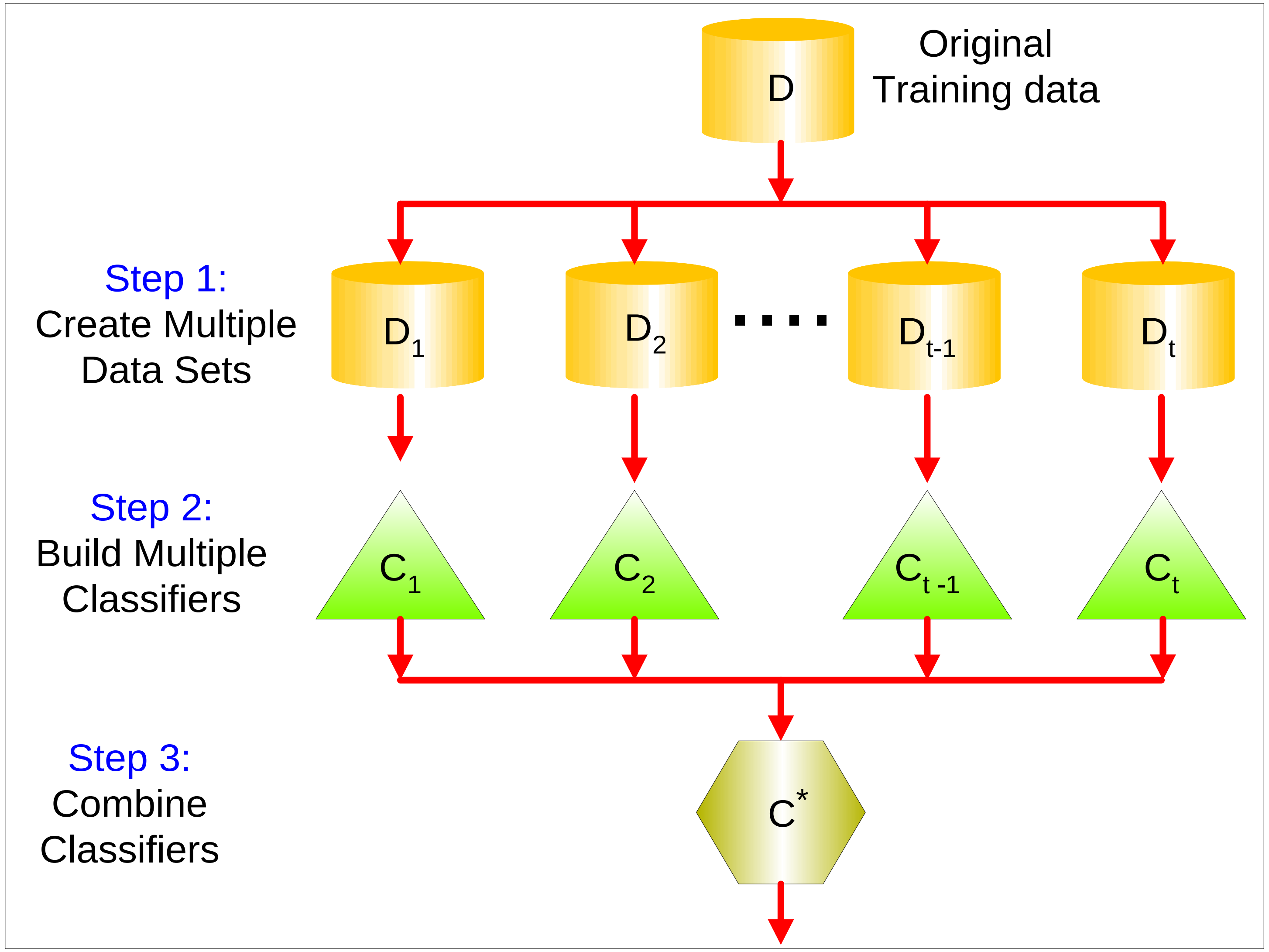
- Suppose there are 25 base classifiers
  - Each classifier has error rate, ε = 0.35
  - Assume errors made by classifiers are uncorrelated
  - Probability that the ensemble classifier makes a wrong prediction:



$$P(X \geq 13) = \sum_{i=13}^{25} \binom{25}{i} \varepsilon^{i} (1-\varepsilon)^{25-i} = 0.06$$

General Approach

- Classifiers are independently learnt over independent samples of the training data
- Predictions are averaged or a majority vote is taken etc.
- Weighted average can be taken as well


D
Original Training data
Step 1: Create Multiple Data Sets
D_1
D_2
....
D_{t-1}
D_t
Step 2: Build Multiple Classifiers
C_1
C_2
C_{t-1}
C_t
Step 3: Combine Classifiers
C^*

Types of Ensemble Methods

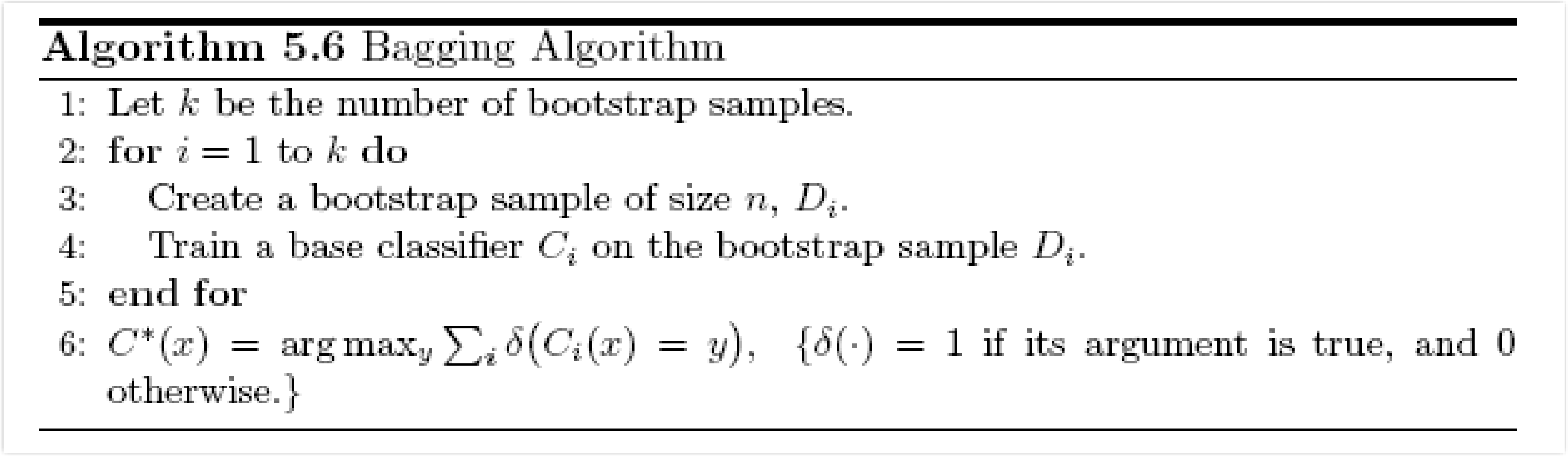
- Manipulate data distribution
  - Bagging: resample original data with replacement
  - Pasting: resample original data without replacement
  - Boosting: ‘resample’ original data by choosing troublesome points more often
  - The learners can also be retrained on modified versions of the original data (gradient boosting).
- Manipulate input features
  - Example: random forests
- Manipulate class labels
  - Example: error-correcting output coding

Bagging

- Learns multiple trees over independent samples of the training data
- For a dataset **D** on **n** data points: Create dataset **D'** of **n** points but sample from **D** with replacement:
  - Each data instance has probability $1-(1 - 1/n)^n$ of being selected as part of the bootstrap sample
  - E.g. 33% points in D' will be duplicates, 66% will be unique
- Predictions from each tree are averaged to compute the final model prediction

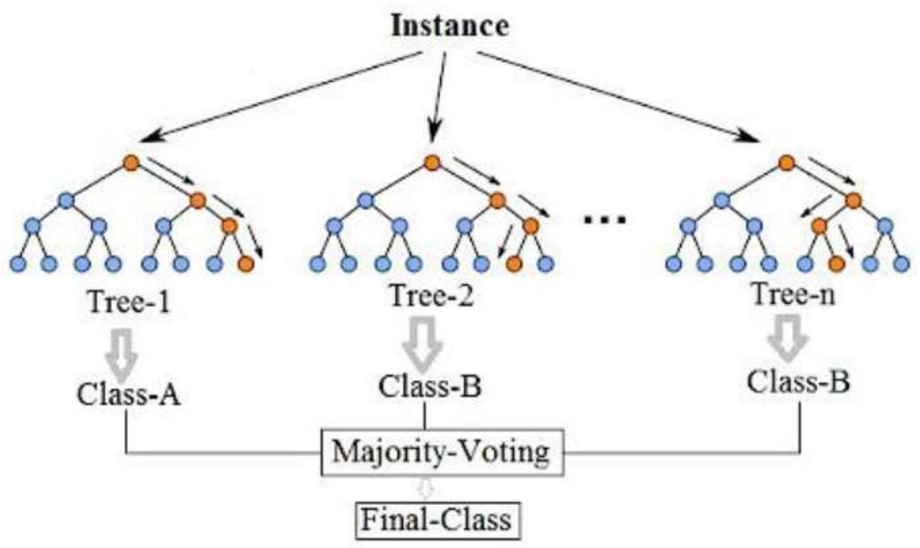
Original Data	1	2	3	4	5	6	7	8	9	10
Bagging (Round 1)	7	8	10	8	2	5	10	10	5	9
Bagging (Round 2)	1	4	9	1	2	3	2	7	3	2
Bagging (Round 3)	1	8	5	10	5	5	9	6	3	7

Bagging Algorithm



Algorithm 5.6 Bagging Algorithm

```
1: Let $k$ be the number of bootstrap samples.
2: for $i = 1$ to $k$ do
3:     Create a bootstrap sample of size $n$, $D_i$.
4:     Train a base classifier $C_i$ on the bootstrap sample $D_i$.
5: end for
6: $C^*(x) = \arg\max_y \sum_i \delta(C_i(x) = y)$, $\{\delta(\cdot) = 1$ if its argument is true, and 0
   otherwise.$\}$
```



- We can also use hash functions (of data sample id, tree id) to create random samples

https://people.csail.mit.edu/rivest/pubs/Riv18e.pdf

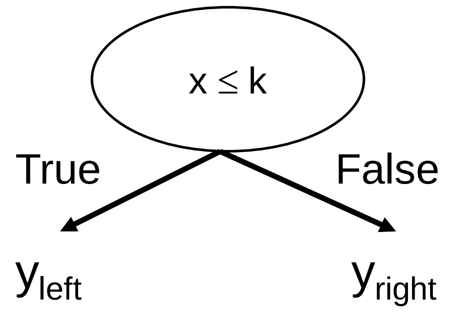
Bagging Example

- Consider 1-dimensional data

Original Data:

x	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
y	1	1	1	-1	-1	-1	-1	1	1	1

- Classifier is a decision stump
  - Decision rule: $x \le k$ versus $x > k$
  - Split point k is chosen based on entropy
  - **Decision stump** is a **weak learner**



Example

Bagging Round 1:

x	0.1	0.2	0.2	0.3	0.4	0.4	0.5	0.6	0.9	0.9
y	1	1	1	1	-1	-1	-1	-1	1	1

x <= 0.35 ➔ y = 1
x > 0.35 ➔ y = -1

Example Continued

Bagging Round 6:

x	0.2	0.4	0.5	0.6	0.7	0.7	0.7	0.8	0.9	1
y	1	-1	-1	-1	-1	-1	-1	1	1	1

x <= 0.75 ➔ y = -1
x > 0.75 ➔ y = 1

Bagging Round 7:

x	0.1	0.4	0.4	0.6	0.7	0.8	0.9	0.9	0.9	1
y	1	-1	-1	-1	-1	1	1	1	1	1

x <= 0.75 ➔ y = -1
x > 0.75 ➔ y = 1

Bagging Round 8:

x	0.1	0.2	0.5	0.5	0.5	0.7	0.7	0.8	0.9	1
y	1	1	-1	-1	-1	-1	-1	1	1	1

x <= 0.75 ➔ y = -1
x > 0.75 ➔ y = 1

Bagging Round 9:

x	0.1	0.3	0.4	0.4	0.6	0.7	0.7	0.8	1	1
y	1	1	-1	-1	-1	-1	-1	1	1	1

x <= 0.75 ➔ y = -1
x > 0.75 ➔ y = 1

Bagging Round 10:

x	0.1	0.1	0.1	0.1	0.3	0.3	0.8	0.8	0.9	0.9
y	1	1	1	1	1	1	1	1	1	1

x <= 0.05 ➔ y = 1
x > 0.05 ➔ y = 1

Bagging Example

Round	Split Point	Left Class	Right Class
1	0.35	1	-1
2	0.7	1	1
3	0.35	1	-1
4	0.3	1	-1
5	0.35	1	-1
6	0.75	-1	1
7	0.75	-1	1
8	0.75	-1	1
9	0.75	-1	1
10	0.05	1	1

Example conclusion

Round	x=0.1	x=0.2	x=0.3	x=0.4	x=0.5	x=0.6	x=0.7	x=0.8	x=0.9	x=1.0
1	1	1	1	-1	-1	-1	-1	-1	-1	-1
2	1	1	1	1	1	1	1	1	1	1
3	1	1	1	-1	-1	-1	-1	-1	-1	-1
4	1	1	1	-1	-1	-1	-1	-1	-1	-1
5	1	1	1	-1	-1	-1	-1	-1	-1	-1
6	-1	-1	-1	-1	-1	-1	-1	1	1	1
7	-1	-1	-1	-1	-1	-1	-1	1	1	1
8	-1	-1	-1	-1	-1	-1	-1	1	1	1
9	-1	-1	-1	-1	-1	-1	-1	1	1	1
10	1	1	1	1	1	1	1	1	1	1
Sum	2	2	2	-6	-6	-6	-6	2	2	2
Predicted Class Sign	1	1	1	-1	-1	-1	-1	1	1	1

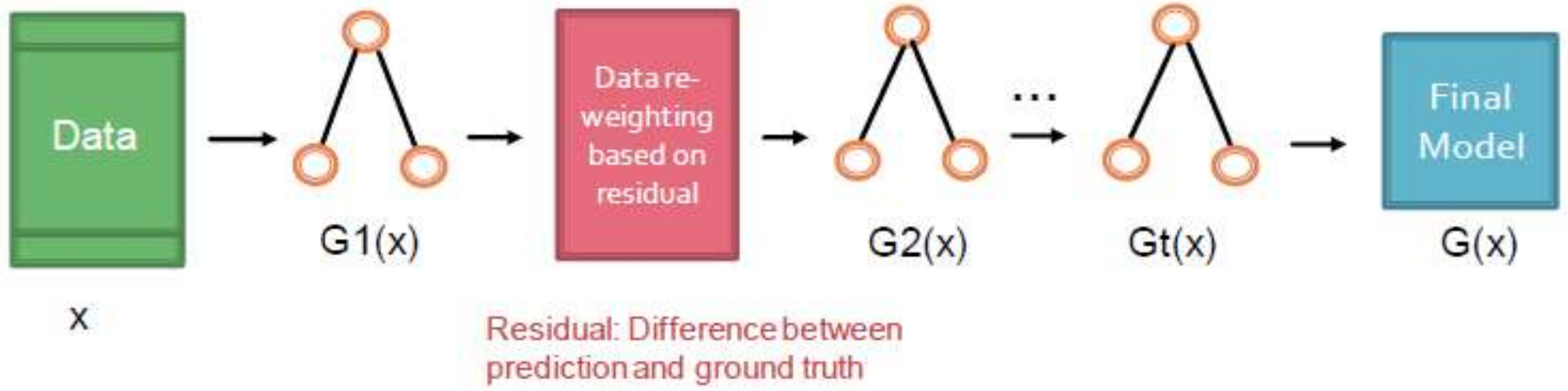
- Assume test set is the same as the original data
- Use majority vote to determine class of ensemble classifier

Random Forests

- Extension of bagging of decision trees: generate bootstrap training samples, and a tree based on each, but also randomize subset of predictors (features) allowed each time a split is considered.
- Typically, # of random predictors chosen to be of order the square root of the total number of predictors.
- **This is called: Feature bagging**
    - **Benefit:** Breaks correlation between trees
        - If one feature is very strong predictor, then every tree will select it, causing trees to be correlated.
- All trees are fully grown, No pruning
- Two parameters
    - Number of trees
    - Number of features
- **Random Forests achieve state-of-the-art results in many classification problems!**

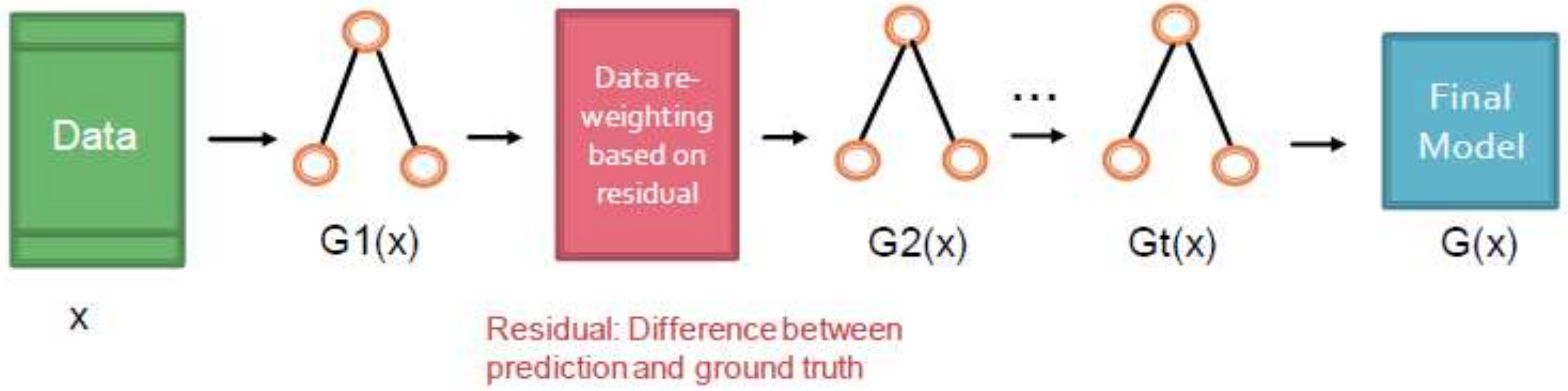
Boosting: Learning from Mistakes

- **Boosting:** Another ensemble learning algorithm
- Combines the outputs of many "weak" classifiers to produce a powerful "committee"
- Learns multiple trees sequentially, each trying to improve upon its predecessor
- Adaptively change distribution of training data by focusing more on previously misclassified records
  - Initially, all N records are assigned equal weights
  - Unlike bagging, weights may change at the end of each boosting round
- Final classifier is weighted sum of the individual classifiers


Data
X
G1(x)
Data re-weighting based on residual
G2(x)
...
Gt(x)
Final Model
G(x)
Residual: Difference between prediction and ground truth

Examples of Boosting

- AdaBoost : Where each $Gt(x)$is a one–level decision tree
- Gradient Boosted Decision Trees: Where each $Gt(x)$is a multi–level decision tree
- There are few implementations on boosting:
  - XGBoost: An efficient Gradient Boosting Decision
  - LGBM: Light Gradient Boosted Machines. It is a library for training GBMs developed by Microsoft, and it competes with XGBoost
  - CatBoost: A new library for Gradient Boosting Decision Trees, offering appropriate handling of categorical features


Data
x
G1(x)
Data re-weighting based on residual
G2(x)
...
Gt(x)
Final Model
G(x)
Residual: Difference between prediction and ground truth

AdaBoost Procedure

- Suppose we have training data $\{(x_i, y_i)\}_{i=1}^{N}$, $y_i \in \{1, -1\}$
- Initialize equal weights for all observations $w_i = \frac{1}{N}$
- At each iteration t:
    1. **Train a stump $\boldsymbol{G_t}$** using data weighted by w_i
    2. **Compute the misclassification error** adjusted by w_i
    3. **Compute the weight of the current tree $\boldsymbol{\alpha_t}$**
    4. **Reweight each observation** based on prediction accuracy

AdaBoost: Weak Learner

- Records that are wrongly classified will have their weights increased
- Records that are classified correctly will have their weights decreased

Original Data	1	2	3	4	5	6	7	8	9	10
Boosting (Round 1)	7	3	2	8	7	9	4	10	6	3
Boosting (Round 2)	5	4	9	4	2	5	1	7	4	2
Boosting (Round 3)	4	4	8	10	4	5	4	6	3	4

- Example 4 is hard to classify
- Its weight is increased, therefore it is more likely to be chosen again in subsequent rounds

Training one decision tree

- **How to split?**
  - Apply weighting to the splitting criterion function and optimize the function to find the best split
  - We'll use information gain as an example
- Information gain: $IG = H(S) - \frac{|S_1|}{|S|} H(S_1) - \frac{|S_2|}{|S|} H(S_2)$
- Where $\mathrm{H}(\boldsymbol{S}) = -\sum_{i=1}^{N} p(X_i) \log(p(X_i))$
- After weighting: $\mathrm{H}_w(\boldsymbol{S}) = -\frac{\sum_{i=1}^{N} w_i p(X_i) \log(p(X_i))}{\sum_{i=1}^{N} w_i}$

Ada Boost

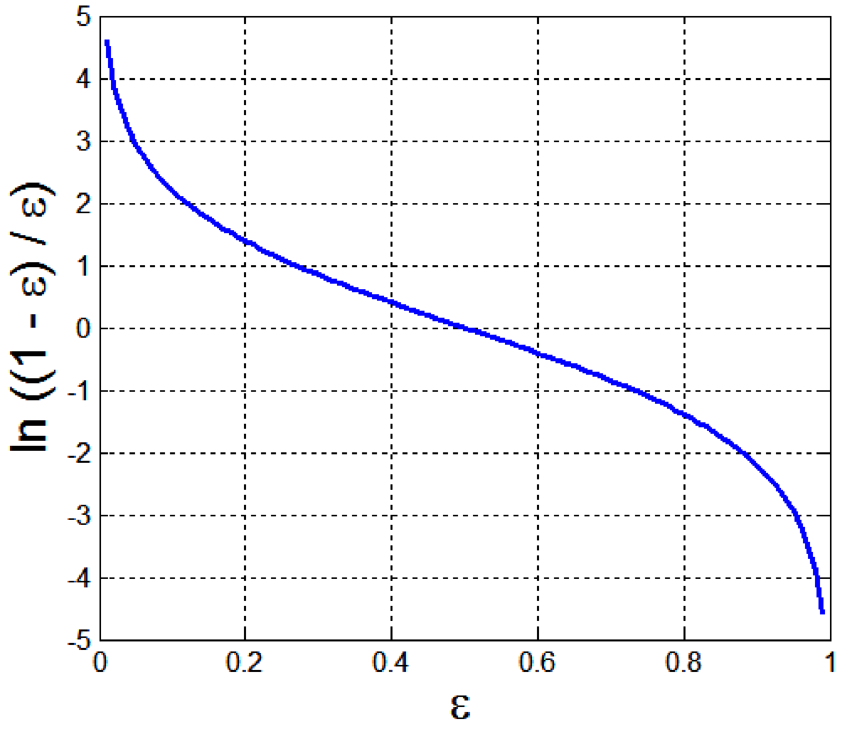
- Base classifiers: C_1, C_2, …, C_T

- Calculate the weighted mis-classification error rate:

$$\varepsilon_i = \frac{1}{N}\sum_{j=1}^{N} w_j \delta\left(C_i(x_j) \neq y_j\right)$$

- Use the error to weight the current tree in the final classifier:

$$\alpha_i = \frac{1}{2}\ln\left(\frac{1-\varepsilon_i}{\varepsilon_i}\right)$$



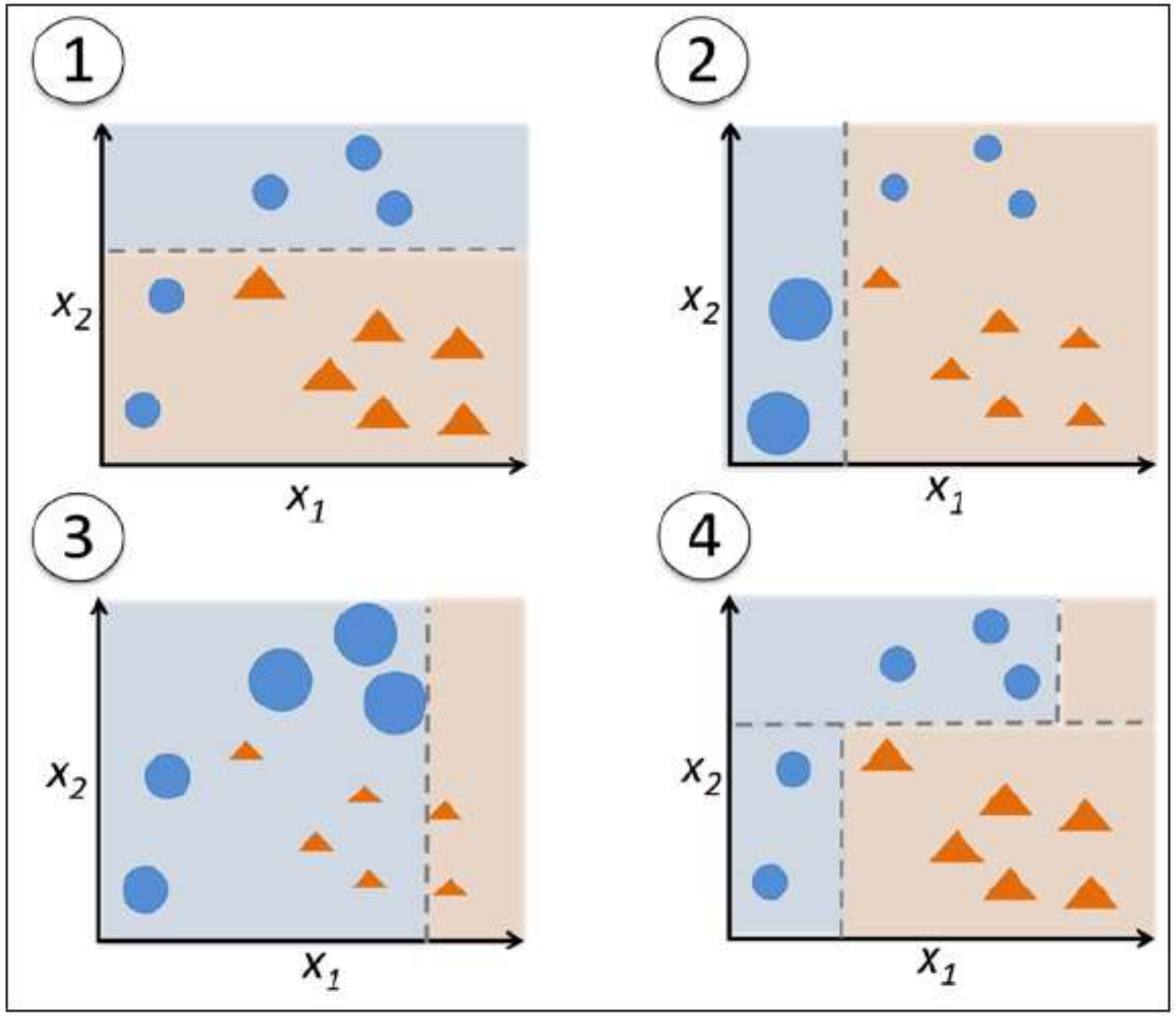
AdaBoost Algorithm Continued

- Weight update of training data: $$w_j^{(i+1)} = \frac{w_j^{(i)}}{Z_i}\begin{cases}\exp^{-\alpha_i} & \text{if } C_i(x_j) = y_j \\ \exp^{\alpha_i} & \text{if } C_i(x_j) \neq y_j\end{cases}$$
  where Z_i is the normalization factor

- If any intermediate rounds produce error rate higher than 50%, the weights are reverted back to 1/n and the resampling procedure is repeated

- Classification:
$$C^*(x) = \underset{y}{\operatorname{argmax}} \sum_{i=1}^{T} \alpha_i \delta(C_i(x) = y)$$

An Illustration



Raschka, Python Machine Learning

- Misclassified points become more important in the next round
- Together, simple classifiers can make complex decisions

AdaBoost Algorithm Summary

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
   $$\mathrm{err}_m = \frac{\sum_{i=1}^{N} w_i I(y_i \neq G_m(x_i))}{\sum_{i=1}^{N} w_i}.$$
   (c) Compute $\alpha_m = \log((1 - \mathrm{err}_m)/\mathrm{err}_m)$.
   (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 $G(x) = \mathrm{sign}\left[\sum_{m=1}^{M} \alpha_m G_m(x)\right]$.

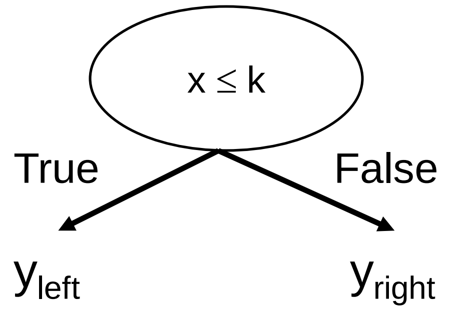
AdaBoost Example

- Consider 1-dimensional data set:

Original Data:

x	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
y	1	1	1	-1	-1	-1	-1	1	1	1

- Classifier is a decision stump
  - Decision rule: $x \le k$ versus $x > k$
  - Split point k is chosen based on entropy



AdaBoost Example

- Training sets for the first 3 boosting rounds:

Boosting Round 1:

x	0.1	0.4	0.5	0.6	0.6	0.7	0.7	0.7	0.8	1
y	1	-1	-1	-1	-1	-1	-1	-1	1	1

Boosting Round 2:

x	0.1	0.1	0.2	0.2	0.2	0.2	0.3	0.3	0.3	0.3
y	1	1	1	1	1	1	1	1	1	1

Boosting Round 3:

x	0.2	0.2	0.4	0.4	0.4	0.4	0.5	0.6	0.6	0.7
y	1	1	-1	-1	-1	-1	-1	-1	-1	-1

- Summary:

Round	Split Point	Left Class	Right Class	alpha
1	0.75	-1	1	1.738
2	0.05	1	1	2.7784
3	0.3	1	-1	4.1195

AdaBoost Example

- Weights

Round	x=0.1	x=0.2	x=0.3	x=0.4	x=0.5	x=0.6	x=0.7	x=0.8	x=0.9	x=1.0
1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
2	0.311	0.311	0.311	0.01	0.01	0.01	0.01	0.01	0.01	0.01
3	0.029	0.029	0.029	0.228	0.228	0.228	0.228	0.009	0.009	0.009

- Classification

Round	x=0.1	x=0.2	x=0.3	x=0.4	x=0.5	x=0.6	x=0.7	x=0.8	x=0.9	x=1.0
1	-1	-1	-1	-1	-1	-1	-1	1	1	1
2	1	1	1	1	1	1	1	1	1	1
3	1	1	1	-1	-1	-1	-1	-1	-1	-1
Sum	5.16	5.16	5.16	-3.08	-3.08	-3.08	-3.08	0.397	0.397	0.397
Sign	1	1	1	-1	-1	-1	-1	1	1	1

Predicted Class

Gradient Boosting

- Idea: Perform additive training with simple models $\{T_h\}_{h \in H}$
- Add a new model (decision tree) each time
- Here the model can be multi-level

$$T = \sum_h \lambda_h T_H$$

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

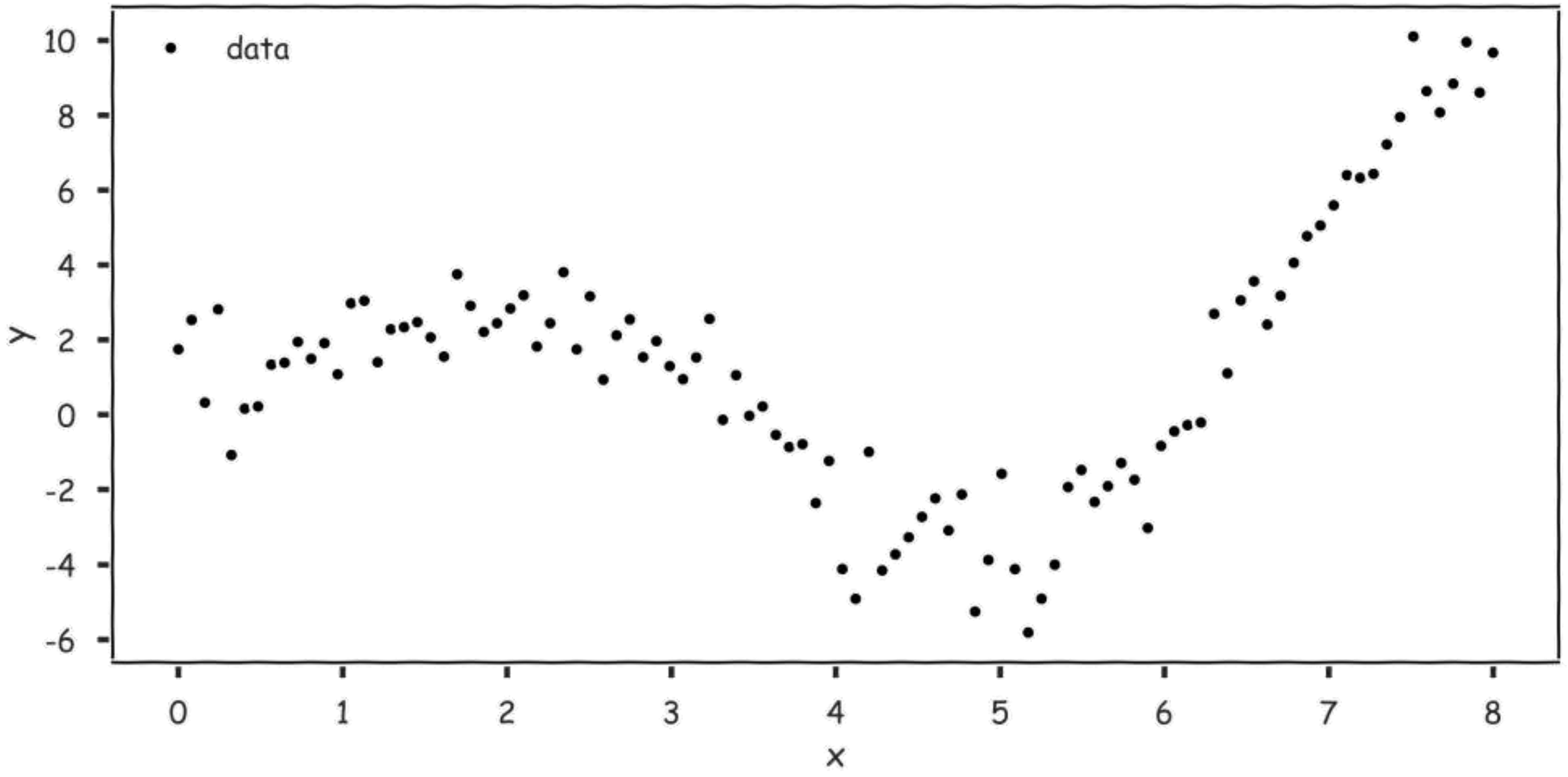
Gradient Boosting: the algorithm

- ***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.

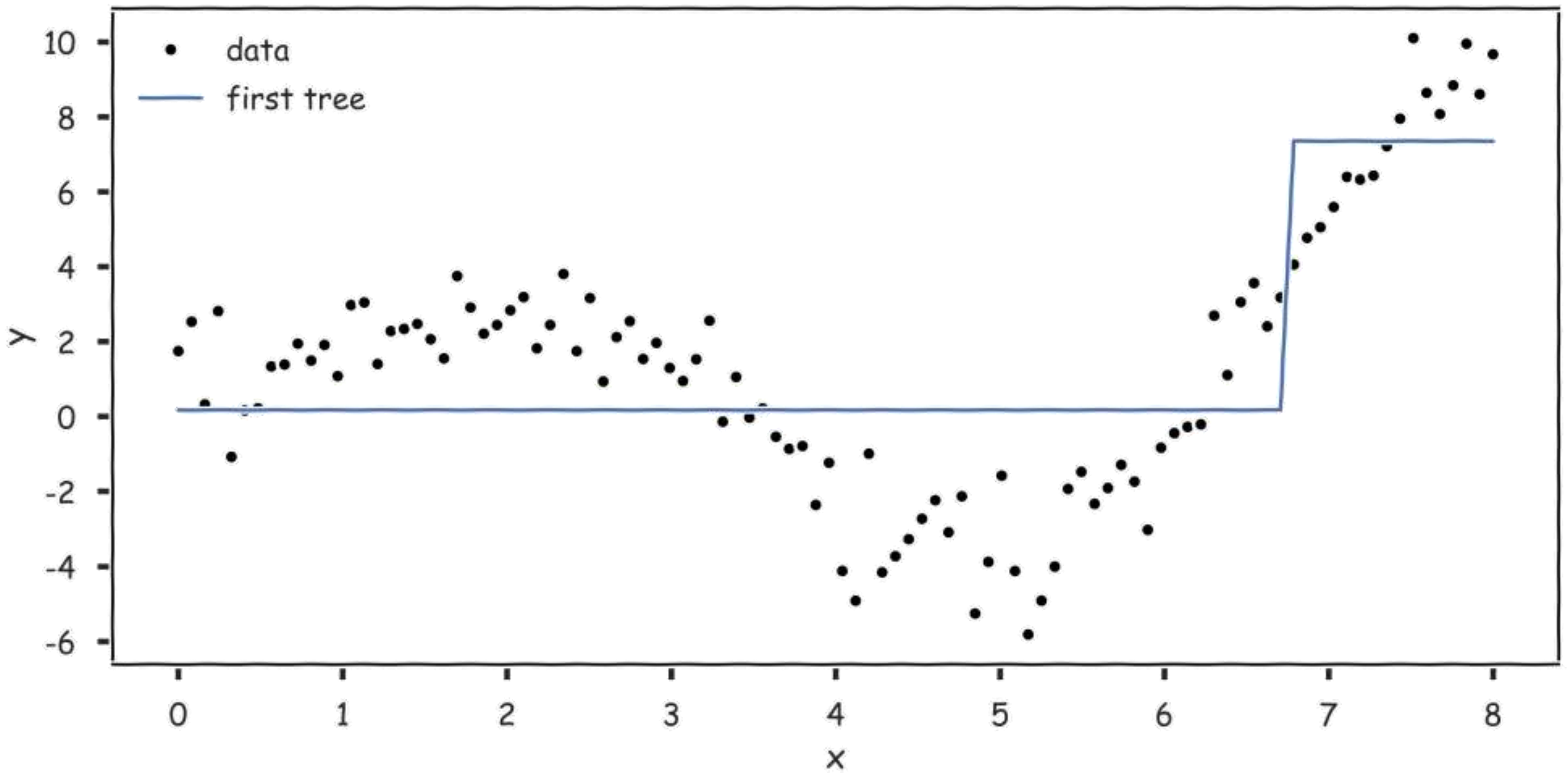
1. Fit a simple model $T^{(0)}$ on the training data $\{(x_1, y_1), \dots, (x_N, y_N)\}$

   Set $T \leftarrow T^{(0)}$. Compute the residuals $\{r_1, . . . , r_N\}$ for T.
2. Fit a simple model, $T^{(1)}$, to the current ***residuals***, i.e. train using $\{(x_1, r_1), \dots, (x_N, r_N)\}$
3. Set $T \leftarrow T + \lambda T^{(1)}$
4. Compute residuals, set $r_n \leftarrow r_n - \lambda T^i(x_n),\ n = 1, \dots, N$
5. Repeat steps 2-4 until ***stopping*** condition met, where λ is a constant called the ***learning rate***.

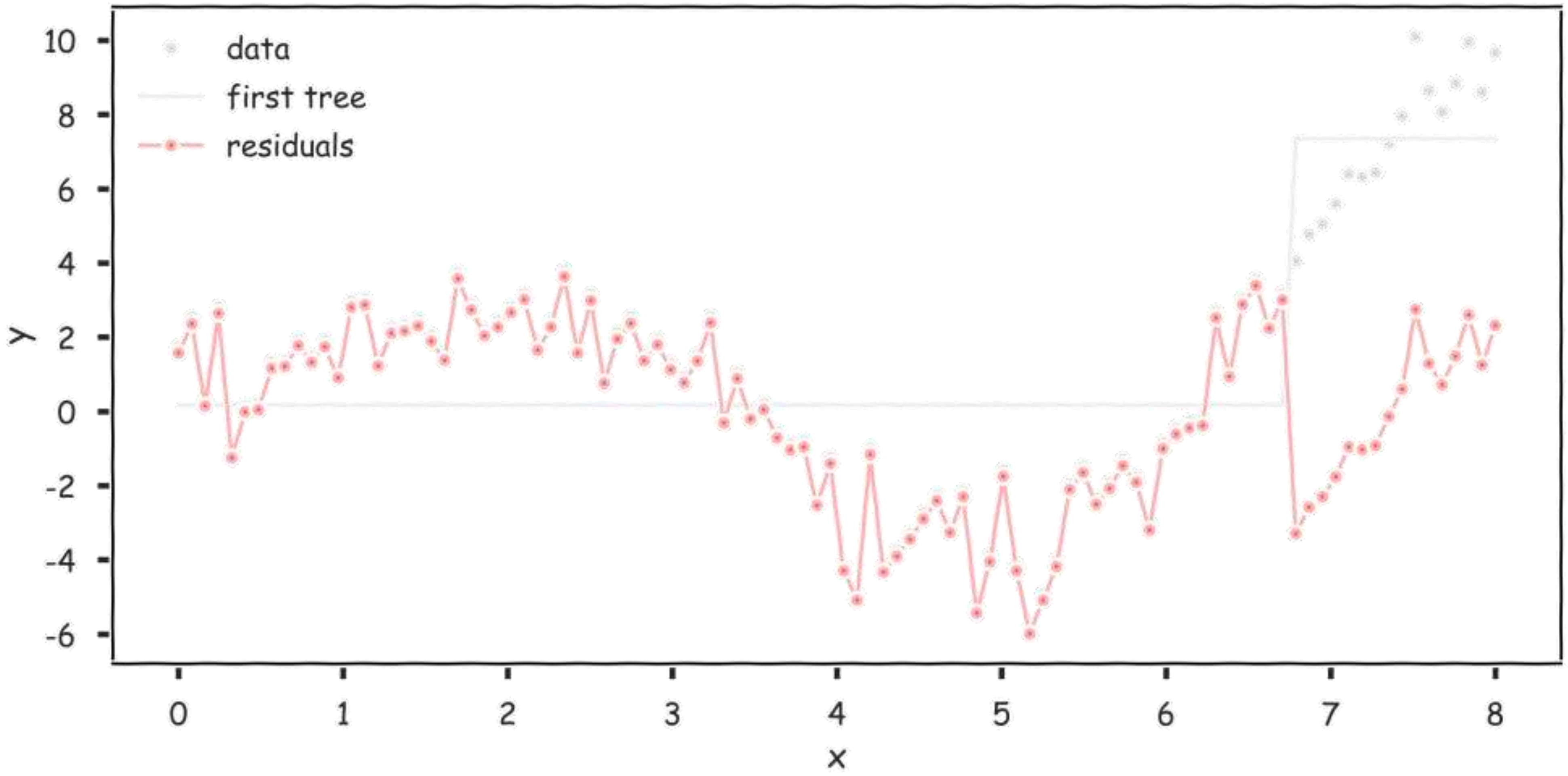
Gradient Boosting: illustration



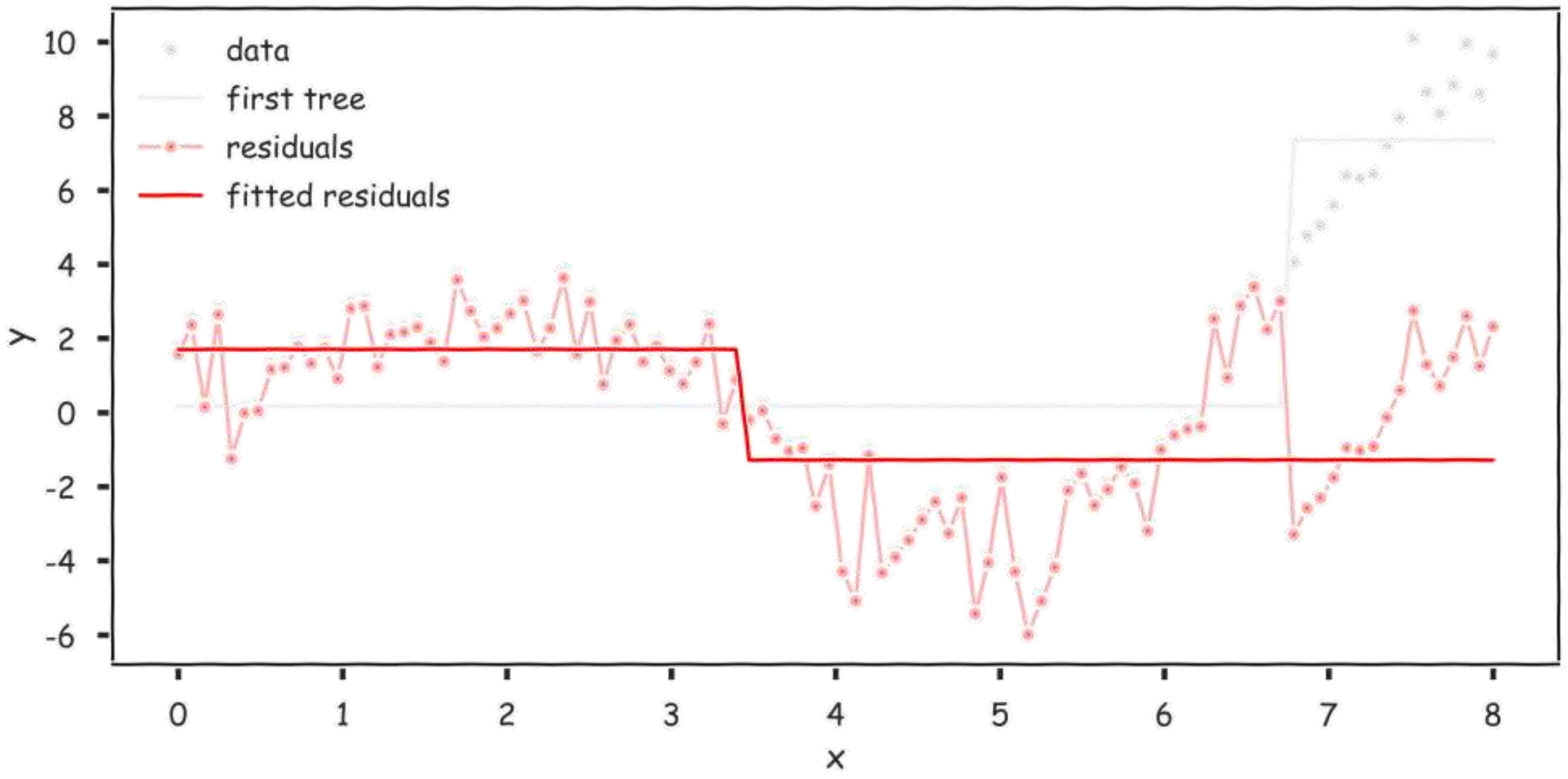
Gradient Boosting: illustration



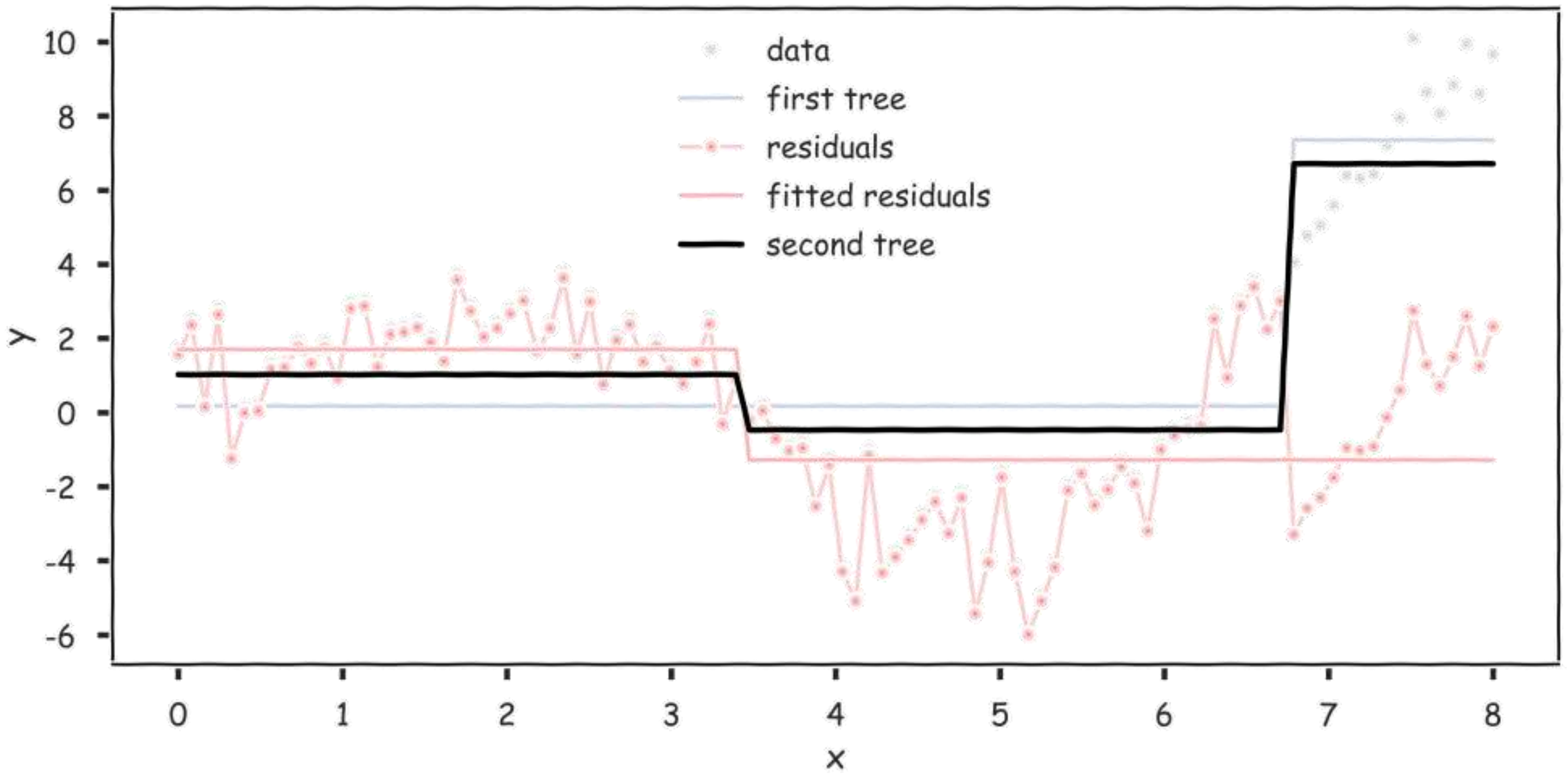
Gradient Boosting: illustration



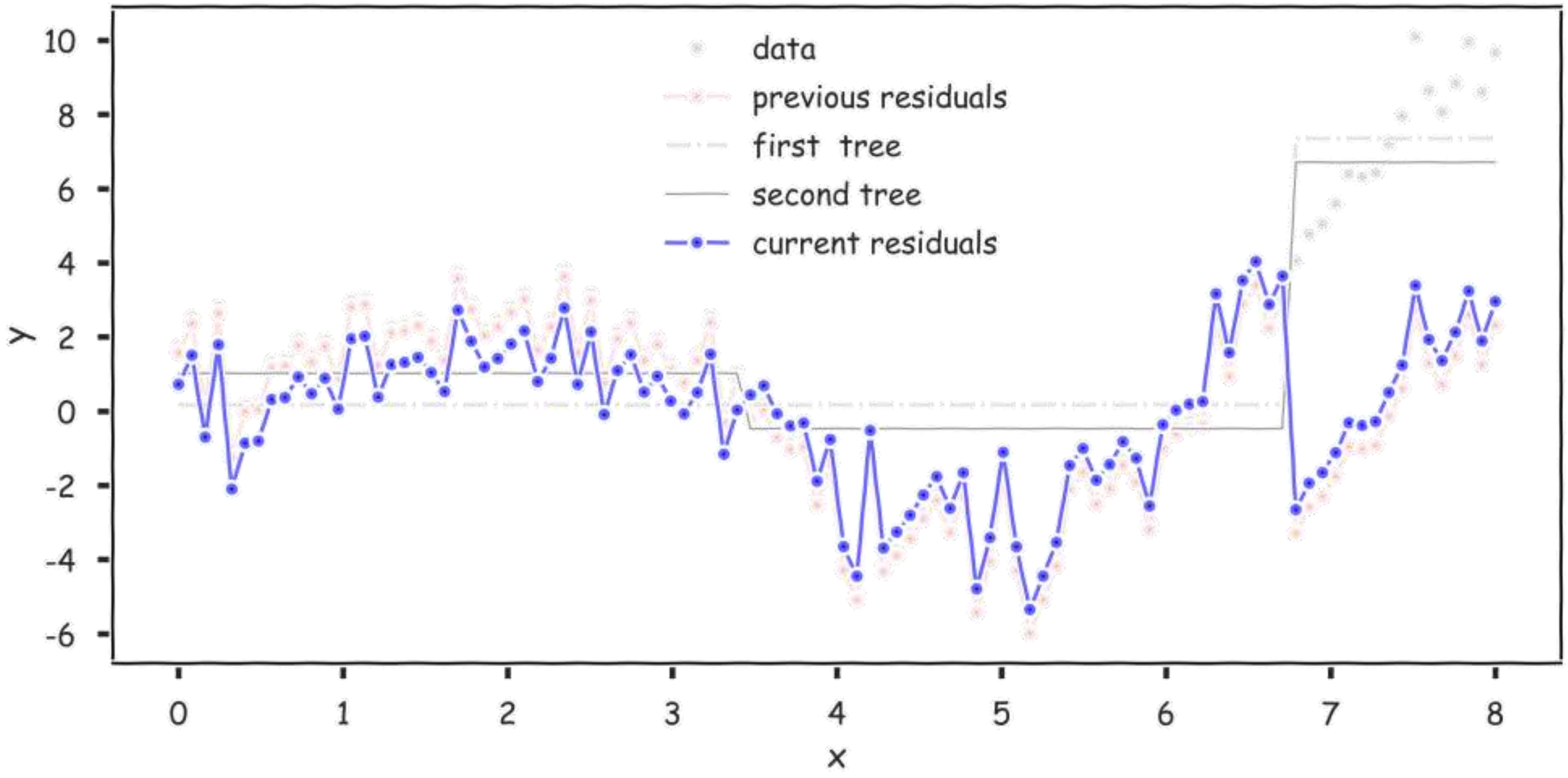
Gradient Boosting: illustration



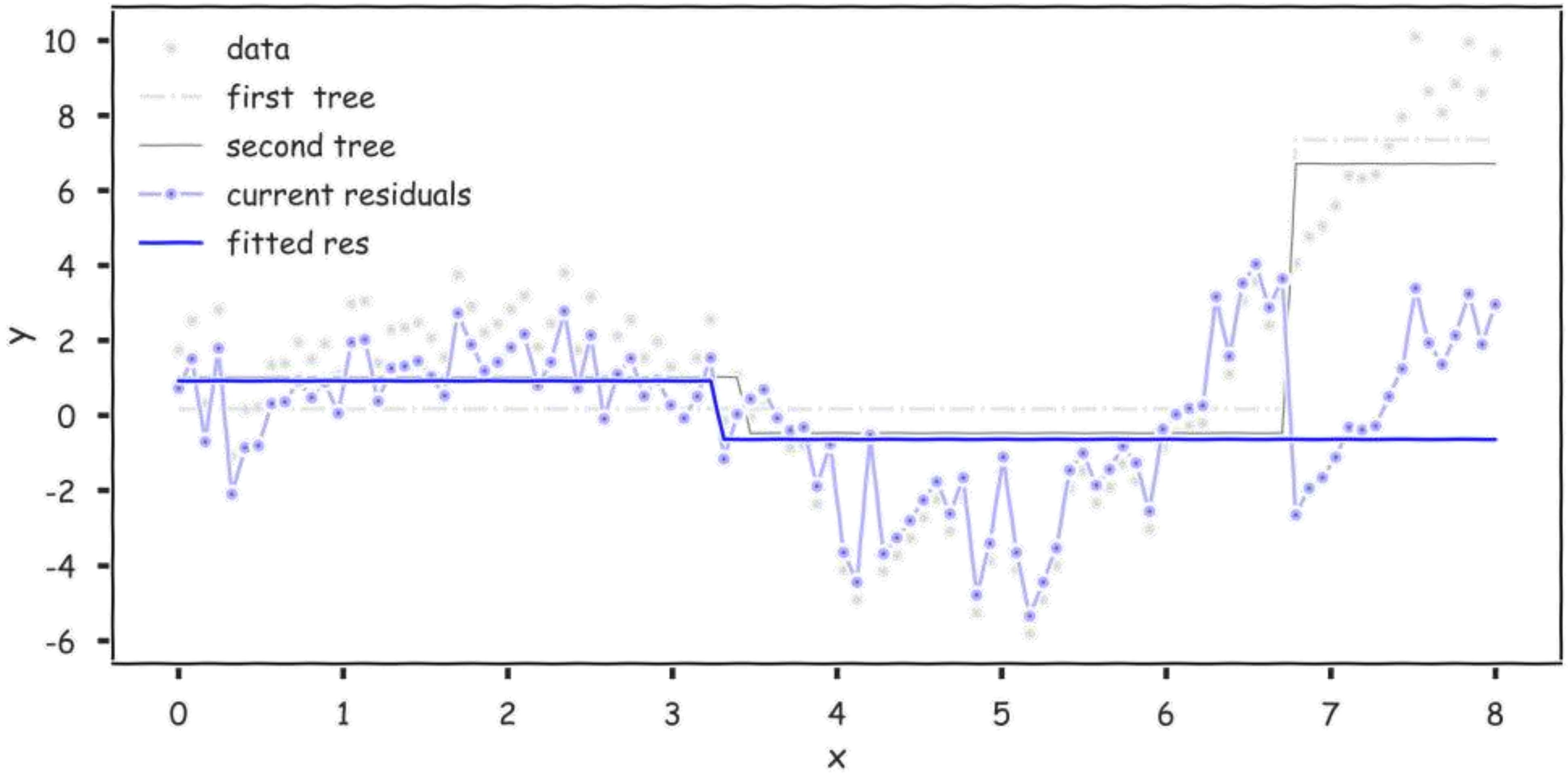
Gradient Boosting: illustration



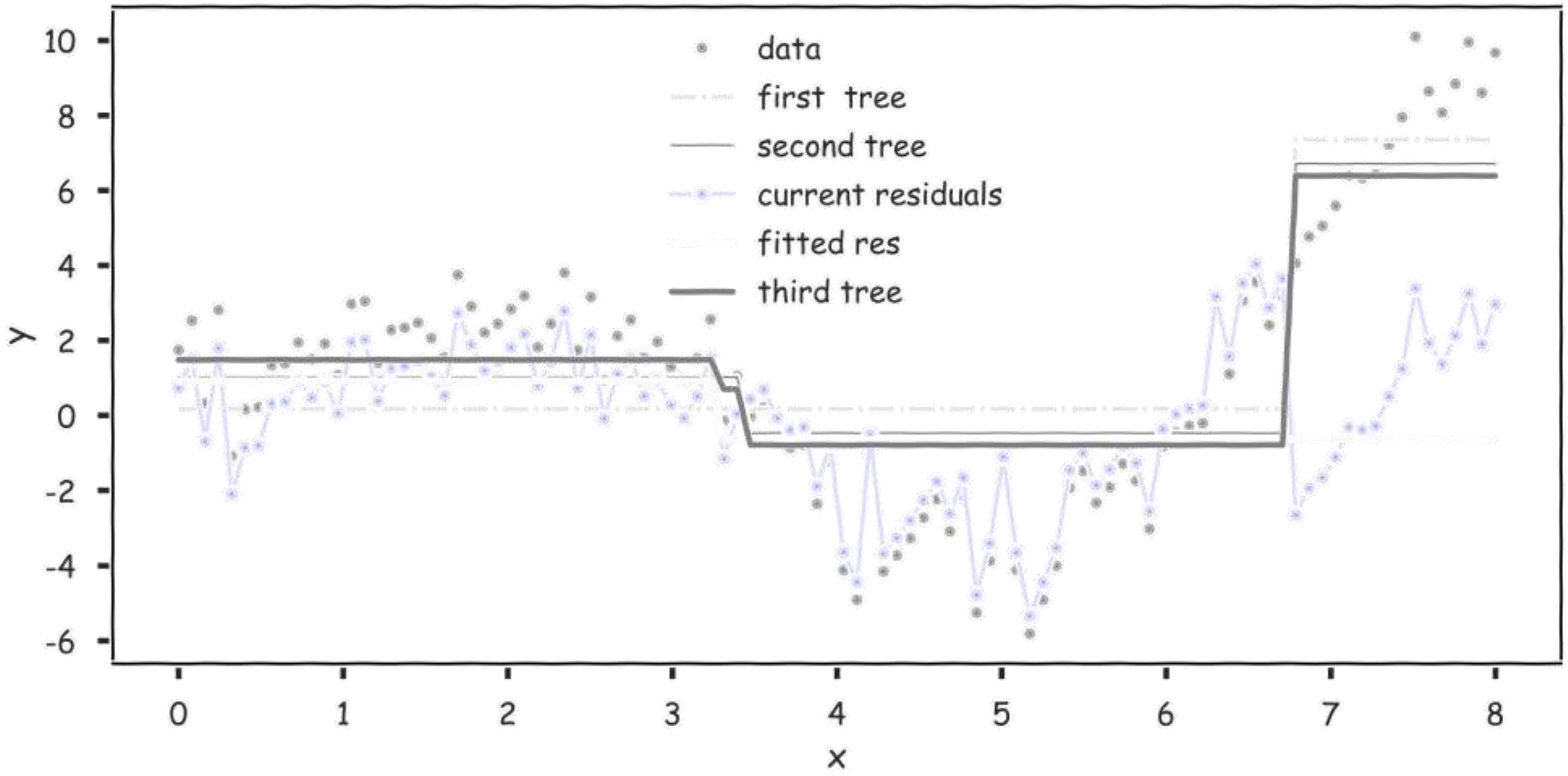
Gradient Boosting: illustration



Gradient Boosting: illustration



Gradient Boosting: illustration



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***.

Gradient Boosting with Regularization

- Prediction at round *n* is: $T^n = T^{n-1} + \lambda f_n(x_i)$
  - We need to decide what $f_n()$ to add
- Goal: Find a tree $f_n(.)$ that minimizes the loss l():

$$\sum_i l(y_i, \hat{y_i}^{n-1} + f_n(x_i)) + \Omega(f_n)$$

  - y_i : The ground-truth label
  - $\hat{y_i}^{n-1} + f_n(x_i)$: The prediction made at round n
  - $\Omega(f_n)$: Model complexity

Regularized Gradient Boosting

- Objective: $\sum_i l(y_i, \hat{y}_i^{\,n-1} + f_n(x_i)) + \Omega(f_n)$
- Take the Taylor expansion of the objective:
  - $g(x+\Delta) \approx g(x) + g'(x)\Delta + \frac{1}{2}g''(x)\Delta^2$
- So, we get the approximate objective:
  - $\sum_i [l(y_i, \hat{y}_i^{\,n-1}) + g_i f_n(x_i)) + \frac{1}{2}\mathrm{h}_i f_n^2(x_i)] + \Omega(f_n)$
- Where $g_i = \partial_{\hat{y}_i^{n-1}} l(y_i, \hat{y}_i^{\,n-1})$ $\; and \;$ $h_i = \partial^2_{\hat{y}_i^{n-1}} l(y_i, \hat{y}_i^{\,n-1})$

How to decide which f to add?

Model Complexity

Reordering the objective

Finding optimal w_j^*

Greedy solution

- In practice, we grow tree greedily:
  - Start with tree with depth 0
  - For each leaf node in the tree, try to add a split
  - The change of the objective after adding a split is:

Take the split that gives the best gain.

Summary: GBDT Algorithm

- **Add a new tree $f_t(x)$ in each iteration**
  - Compute necessary statistics for our objective

$$g_i = \partial_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)}), \quad h_i = \partial^2_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)})$$

  - Greedily grow the tree that minimizes the objective:

$$Obj = -\frac{1}{2}\sum_{j=1}^{T} \frac{G_j^2}{H_j + \lambda} + \gamma T$$

- **Add $f_t(x)$ to our ensemble model**

$$y^{(t)} = y^{(t-1)} + \epsilon f_t(x_i)$$

ϵ is called step-size or shrinkage, usually set around 0.1
Goal: prevent overfitting

- **Repeat until we user M ensemble of trees**

AdaBoost and Gradient Boosting

- Using the language of gradient descent also allow us to connect gradient boosting for regression to a boosting algorithm often used for classification, AdaBoost.
- In classification, we *typically* want to minimize the classification error:
- **Our solution:** we replace the Error function with a differentiable function that is a good indicator of classification error.
- The function we choose is called ***exponential loss***

$$\text{ExpLoss} = \frac{1}{N}\sum_{n=1}^{N} \exp\left(-y_n \hat{y}_n\right), \;\; y_n \in \{-1, 1\}$$

- Exponential loss is differentiable with respect to $\hat{y}_n$

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

Review: A Brief Sketch of Gradient Descent

- In optimization, when we wish to minimize a function, called the ***objective 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 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:

$$x = [x_1, \ldots, x_J]$$

2. Take the gradient of the objective function at the current variable values:

$$\nabla f(x) = \left[\frac{\partial f}{\partial x_1}(x), \ldots, \frac{\partial f}{\partial x_J}(x)\right]$$

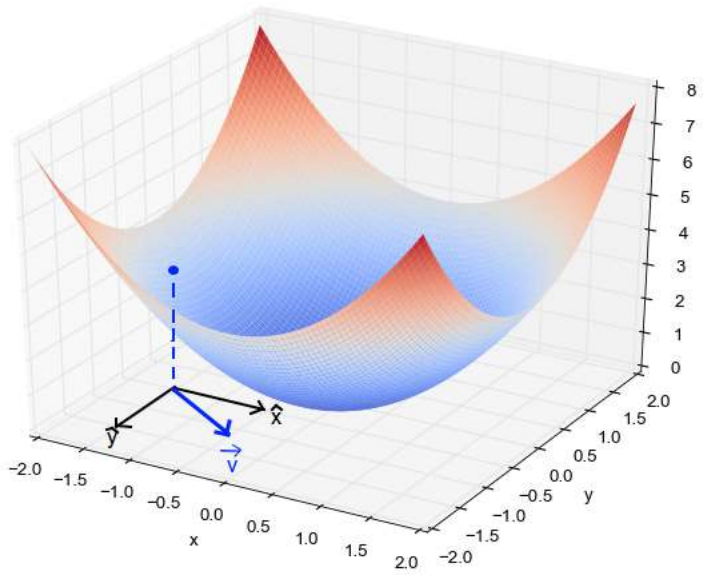
3. Adjust the variables values by some negative multiple of the gradient:

$$x \leftarrow x - \lambda \nabla f(x)$$

- The factor λ is often called the learning rate.

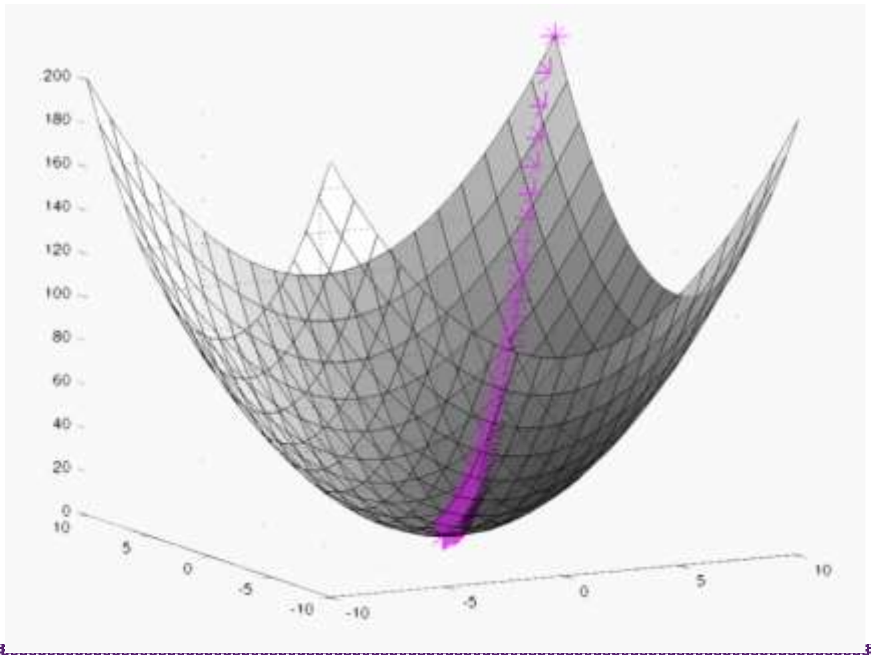
Why Does Gradient Descent Work?

- **Claim:** If the function is convex, this iterative method 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

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

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

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

- 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:

- $\hat{y} \leftarrow \hat{y}_n + \lambda\, \hat{r}_n(x_n), \quad n = 1, \dots, N$
  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 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.)

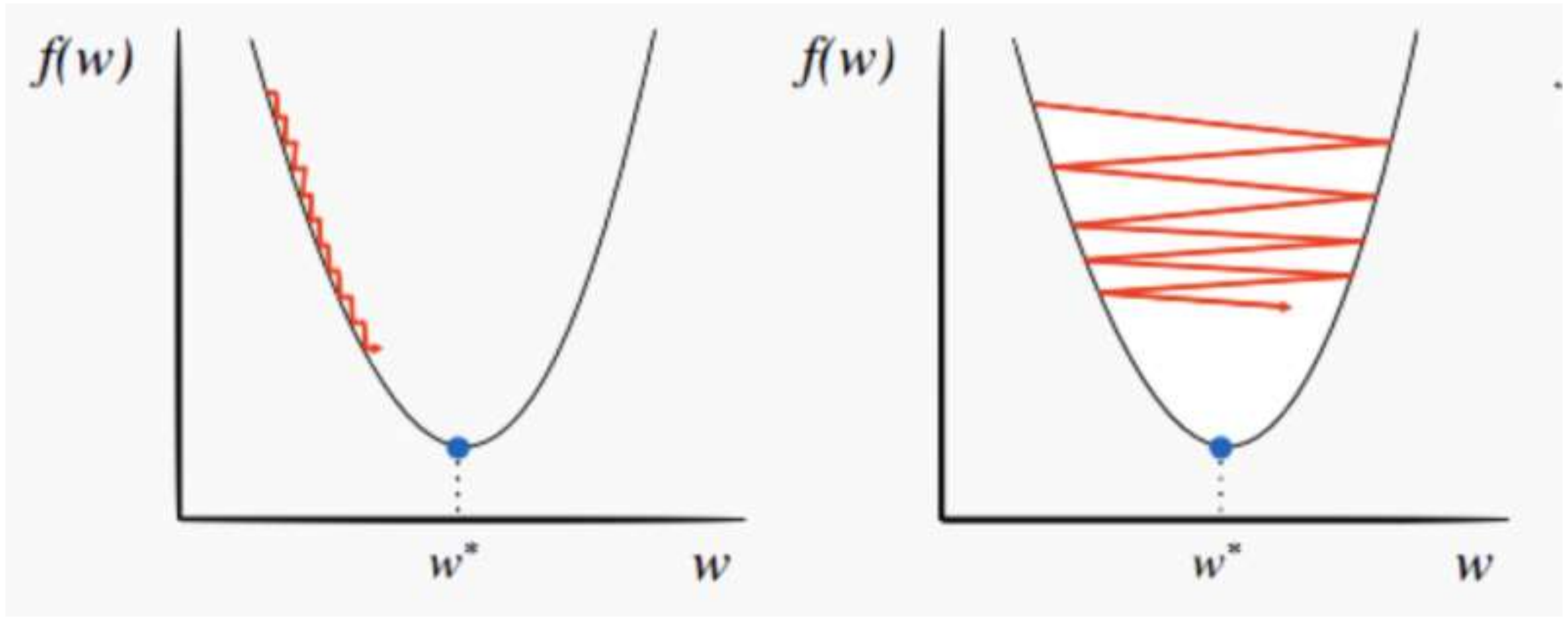
- But why do we care that gradient boosting is gradient descent?
- By making this connection, we can import the massive amount of techniques for studying gradient descent to analyze gradient boosting.
- **For example**, we can easily reason about how to choose the learning rate λ in gradient boosting.

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.