Ensemble Techniques

Li, Jia

DSAA 5002
HKUST Guangzhou

Sep 29, 2025

Ensemble Methods

- Construct a set of base classifiers learned from the training data

- Predict class label of test records by combining the predictions made by multiple classifiers (e.g., by taking majority vote)

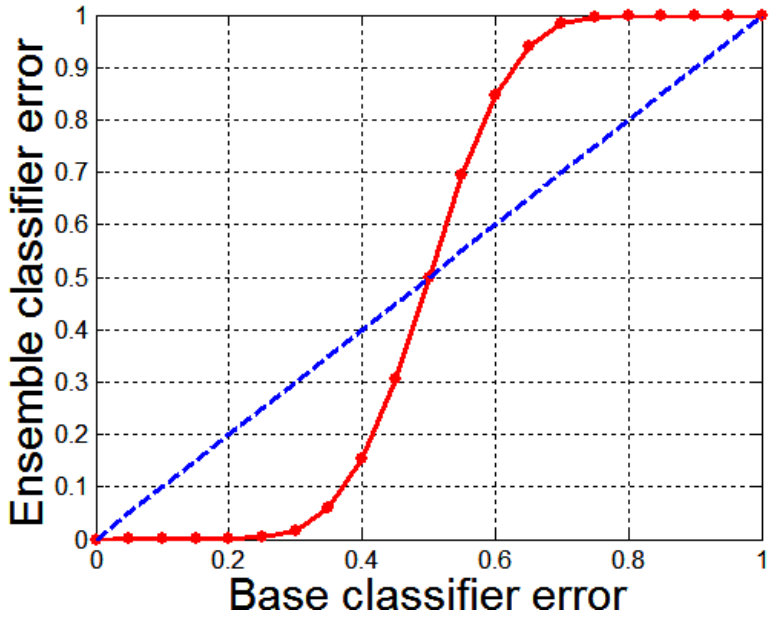
Example: Why Do Ensemble Methods Work?

- Suppose there are 25 base classifiers
  - Each classifier has error rate, $\epsilon = 0.35$
  - Majority vote of classifiers used for classification
  - If all classifiers are identical:
    - Error rate of ensemble = ϵ (0.35)
  - If all classifiers are independent (errors are uncorrelated):
    - Error rate of ensemble = probability of having more than half of base classifiers being wrong

$$e_{\text{ensemble}} = \sum_{i=13}^{25} \binom{25}{i} \epsilon^i (1-\epsilon)^{25-i} = 0.06$$

Necessary Conditions for Ensemble Methods

- **Ensemble Methods work better than a single base classifier if:**
  1. All base classifiers are independent of each other
  2. All base classifiers perform better than random guessing (error rate < 0.5 for binary classification)



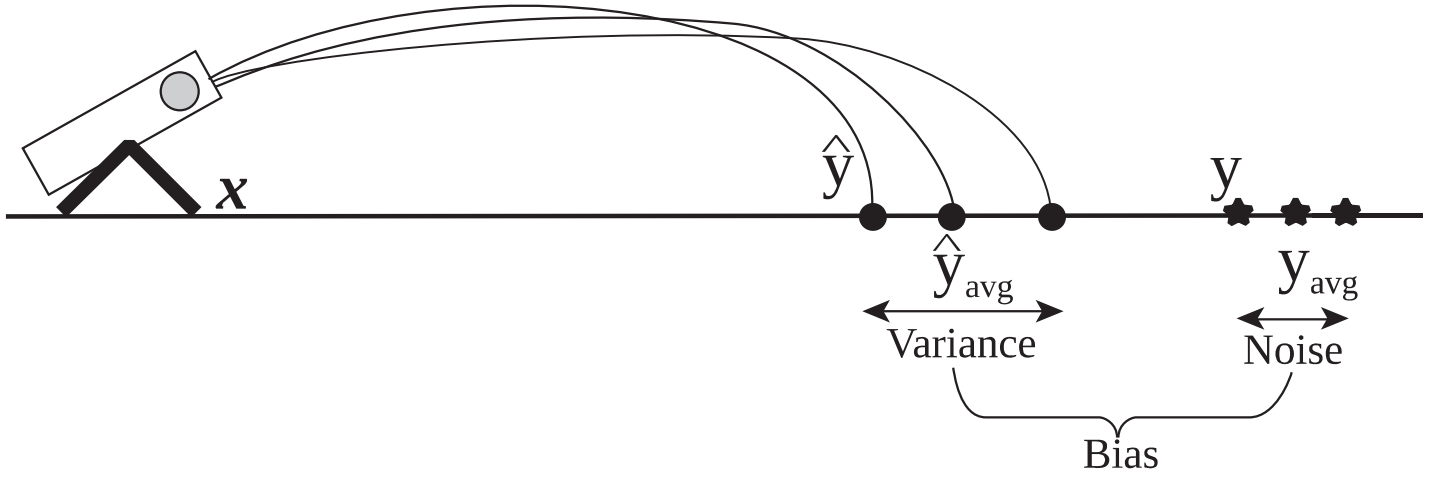
Classification error for an ensemble of 25 base classifiers, assuming their errors are uncorrelated.

Rationale for Ensemble Learning

- **Ensemble Methods work best with unstable base classifiers**
  - Classifiers that are sensitive to minor perturbations in training set, due to high model complexity
  - Examples: Unpruned decision trees, ANNs, …

Bias-Variance Decomposition

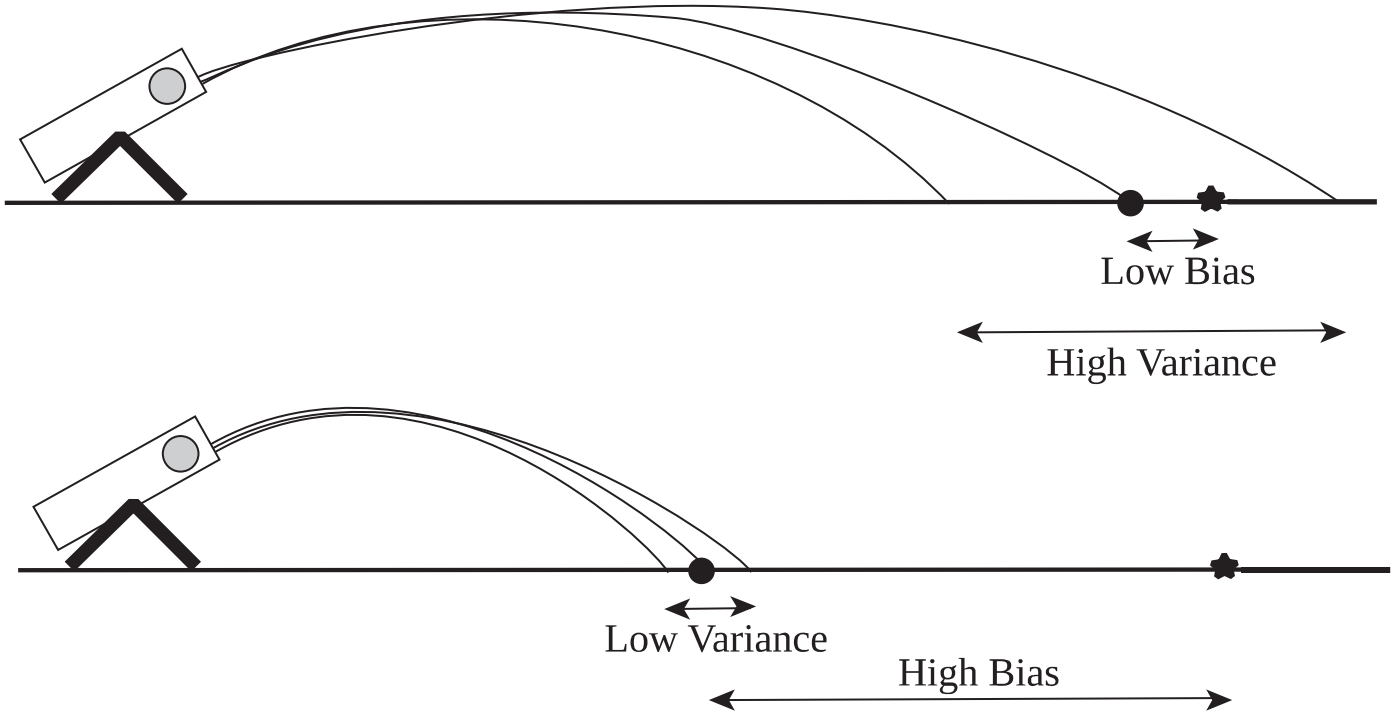
- Analogous problem of reaching a target y by firing projectiles from x (regression problem)



- For classification, the generalization error of model m can be given by:

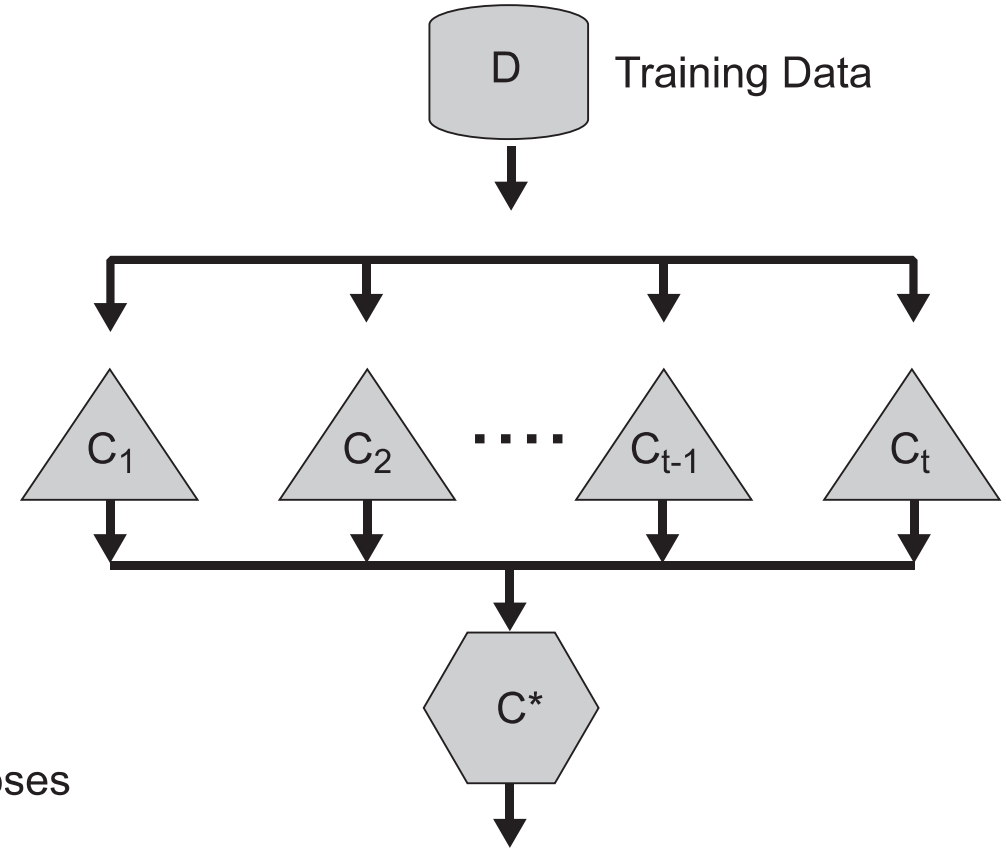
$$gen.error(m)=c_1+bias(m)+c_2\times variance(m)$$

Bias-Variance Trade-off and Overfitting



Ensemble methods try to reduce the variance of complex models (with low bias) by aggregating responses of multiple base classifiers

General Approach of Ensemble Learning



Using majority vote or weighted majority vote (weighted according to their accuracy or relevance)

Constructing Ensemble Classifiers

- **By manipulating training set**
  - Example: bagging, boosting

- **By manipulating input features**
  - Example: random forests

- **By manipulating learning algorithm**
  - Example: injecting randomness in the initial weights of ANN

Bagging (Bootstrap AGGregatING)

- **Bootstrap sampling: sampling with replacement**

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

- **Build classifier on each bootstrap sample**

- **Probability of a training instance being selected in a bootstrap sample is:**
  - $1 - (1 - 1/n)^n$ (n: number of training instances)
  - ~0.632 when n is large

Bagging Algorithm

Algorithm 4.5 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: $C^*(x) = \underset{y}{\operatorname{argmax}} \sum_i \delta\big(C_i(x) = y\big)$.

$\{\delta(\cdot) = 1$ if its argument is true and 0 otherwise.$\}$

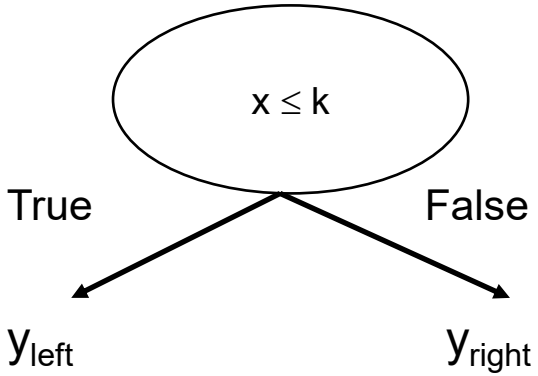
Bagging Example

- **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 tree of size 1)**
  - Decision rule: $x \leq k$ versus $x > k$
  - Split point k is chosen based on entropy



Bagging 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

Bagging 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

Bagging Round 2:

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

x <= 0.7 ➔ y = 1
x > 0.7 ➔ y = 1

Bagging Round 3:

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

x <= 0.35 ➔ y = 1
x > 0.35 ➔ y = -1

Bagging Round 4:

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

x <= 0.3 ➔ y = 1
x > 0.3 ➔ y = -1

Bagging Round 5:

x	0.1	0.1	0.2	0.5	0.6	0.6	0.6	1	1	1
y	1	1	1	-1	-1	-1	-1	1	1	1

x <= 0.35 ➔ y = 1
x > 0.35 ➔ y = -1

Bagging Example

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

- **Summary of Trained Decision Stumps:**

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

Bagging Example

- Use majority vote (sign of sum of predictions) to determine class of ensemble classifier

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
Sign	1	1	1	-1	-1	-1	-1	1	1	1

Predicted Class

- Bagging can also increase the complexity (representation capacity) of simple classifiers such as decision stumps

Random Forest Algorithm

- **Construct an ensemble of decision trees by manipulating training set as well as features**

  - Use bootstrap sample to train every decision tree (similar to Bagging)
  - Use the following tree induction algorithm:
    - At every internal node of decision tree, randomly sample p attributes for selecting split criterion
    - Repeat this procedure until all leaves are pure (unpruned tree)

Characteristics of Random Forest

- Base classifiers are unpruned trees and hence are *unstable classifiers*
- Base classifiers are *decorrelated* (due to randomization in training set as well as features)
- Random forests reduce variance of unstable classifiers without negatively impacting the bias
- Selection of hyper-parameter p
  - Small value ensures lack of correlation
  - High value promotes strong base classifiers
  - Common default choices: $\sqrt{d}$, $\log_2(d+1)$

Bags and Forests of Trees

Bagging:

- create an ensemble of trees, each trained on a bootstrap sample of the training set
- average the predictions.

Random forest:

- create an ensemble of trees, each trained on a bootstrap sample of the training set
- in each tree and each split, randomly select a subset of predictors, choose a predictor from this subset for splitting
- average the predictions

Note that the ensemble building aspects of both methods are embarrassingly parallel!

Tuning Random Forests

Random forest models have multiple hyper-parameters to tune:

1. the number of predictors to randomly select at each split
2. the total number of trees in the ensemble
3. the minimum leaf node size

In theory, each tree in the random forest is full, but in practice this can be computationally expensive (and added redundancies in the model), thus, imposing a minimum node size is not unusual.

Variable Importance for RF

Same as with Bagging:

Calculate the total amount that the Gini index (for classification) is decreased due to splits over a given predictor, averaged over all trees.

Final Thoughts on Random Forests

When the number of predictors is large, but the number of relevant predictors is small, random forests can perform poorly.

Question: Why?

In each split, the chances of selected a relevant predictor will be low and hence most trees in the ensemble will be weak models.

Final Thoughts on Random Forests (cont.)

Increasing the number of trees in the ensemble generally does not increase the risk of overfitting.

Again, by decomposing the generalization error in terms of bias and variance, we see that increasing the number of trees produces a model that is at least as robust as a single tree.

However, if the number of trees is too large, then the trees in the ensemble may become more correlated, increase the variance.

Boosting

- **An iterative procedure to adaptively change distribution of training data by focusing more on previously misclassified records**
  - Initially, all N records are assigned equal weights (for being selected for training)
  - Unlike bagging, weights may change at the end of each boosting round

Boosting

- **Records that are wrongly classified will have their weights increased in the next round**
- **Records that are classified correctly will have their weights decreased in the next round**

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

AdaBoost

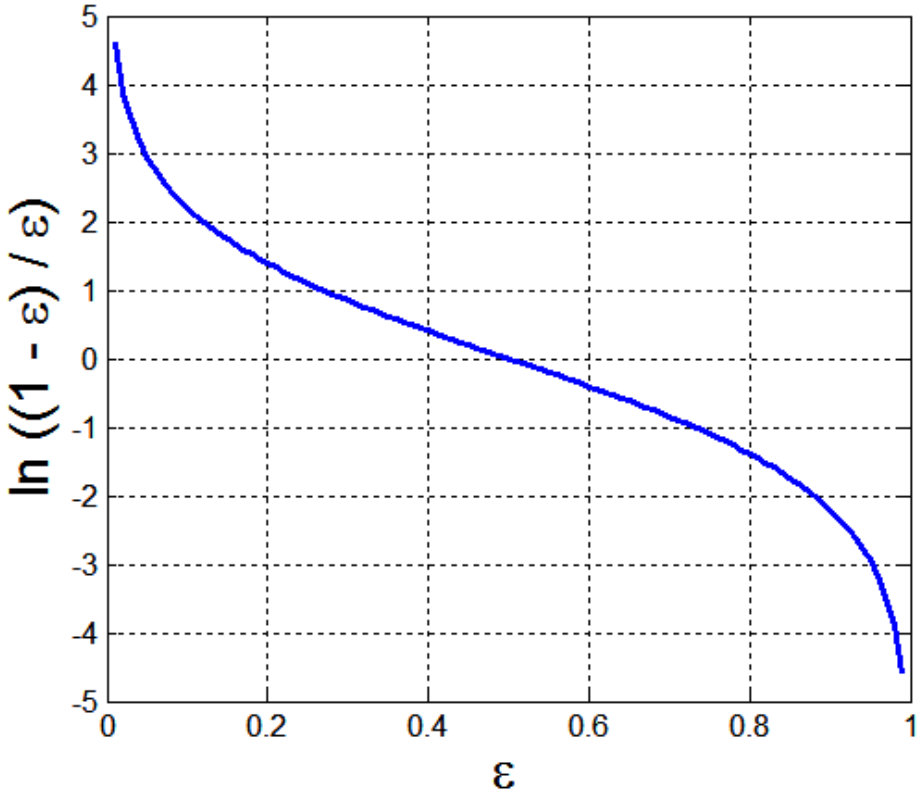
- Base classifiers: C_1, C_2, …, C_T

- Error rate of a base classifier:

$$\epsilon_i = \frac{1}{N}\sum_{j=1}^{N} w_j^{(i)}\, \delta(C_i(x_j) \neq y_j)$$

- Importance of a classifier:

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



AdaBoost Algorithm

- **Weight update:**

$$w_j^{(i+1)} = \frac{w_j^{(i)}}{Z_i} \times \begin{cases} e^{-\alpha_i} & \text{if } C_i(x_j) = y_j \\ e^{\alpha_i} & \text{if } C_i(x_j) \neq y_j \end{cases}$$

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:**

$$C^*(x) = \arg\max_y \sum_{i=1}^{T} \alpha_i \delta(C_i(x) = y)$$

AdaBoost Algorithm

Algorithm 4.6 AdaBoost algorithm.

1: $\mathbf{w} = \{w_j = 1/N \mid j = 1, 2, \ldots, N\}$. {Initialize the weights for all N examples.}
2: Let k be the number of boosting rounds.
3: **for** $i = 1$ to k **do**
4: Create training set D_i by sampling (with replacement) from D according to $\mathbf{w}$.
5: Train a base classifier C_i on D_i.
6: Apply C_i to all examples in the original training set, D.
7: $\epsilon_i = \frac{1}{N}\left[\sum_j w_j \, \delta\big(C_i(x_j) \neq y_j\big)\right]$ {Calculate the weighted error.}
8: **if** $\epsilon_i > 0.5$ **then**
9: $\mathbf{w} = \{w_j = 1/N \mid j = 1, 2, \ldots, N\}$. {Reset the weights for all N examples.}
10: Go back to Step 4.
11: **end if**
12: $\alpha_i = \frac{1}{2} \ln \frac{1-\epsilon_i}{\epsilon_i}$.
13: Update the weight of each example according to Equation 4.103.
14: **end for**
15: $C^*(\mathbf{x}) = \underset{y}{\operatorname{argmax}} \sum_{j=1}^{T} \alpha_j \delta(C_j(\mathbf{x}) = y)\big)$.

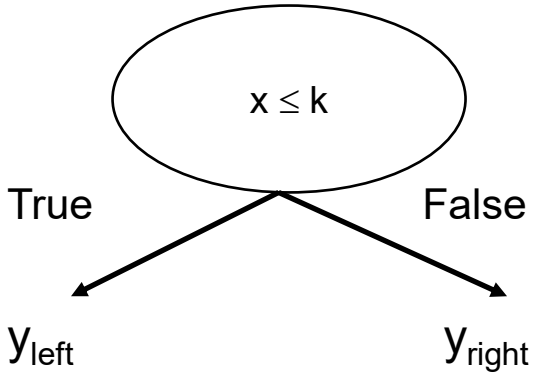
AdaBoost Example

- **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: $x \leq k$ 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

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.

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, $T^{(0)}$, to the current ***residuals***, i.e. train using

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

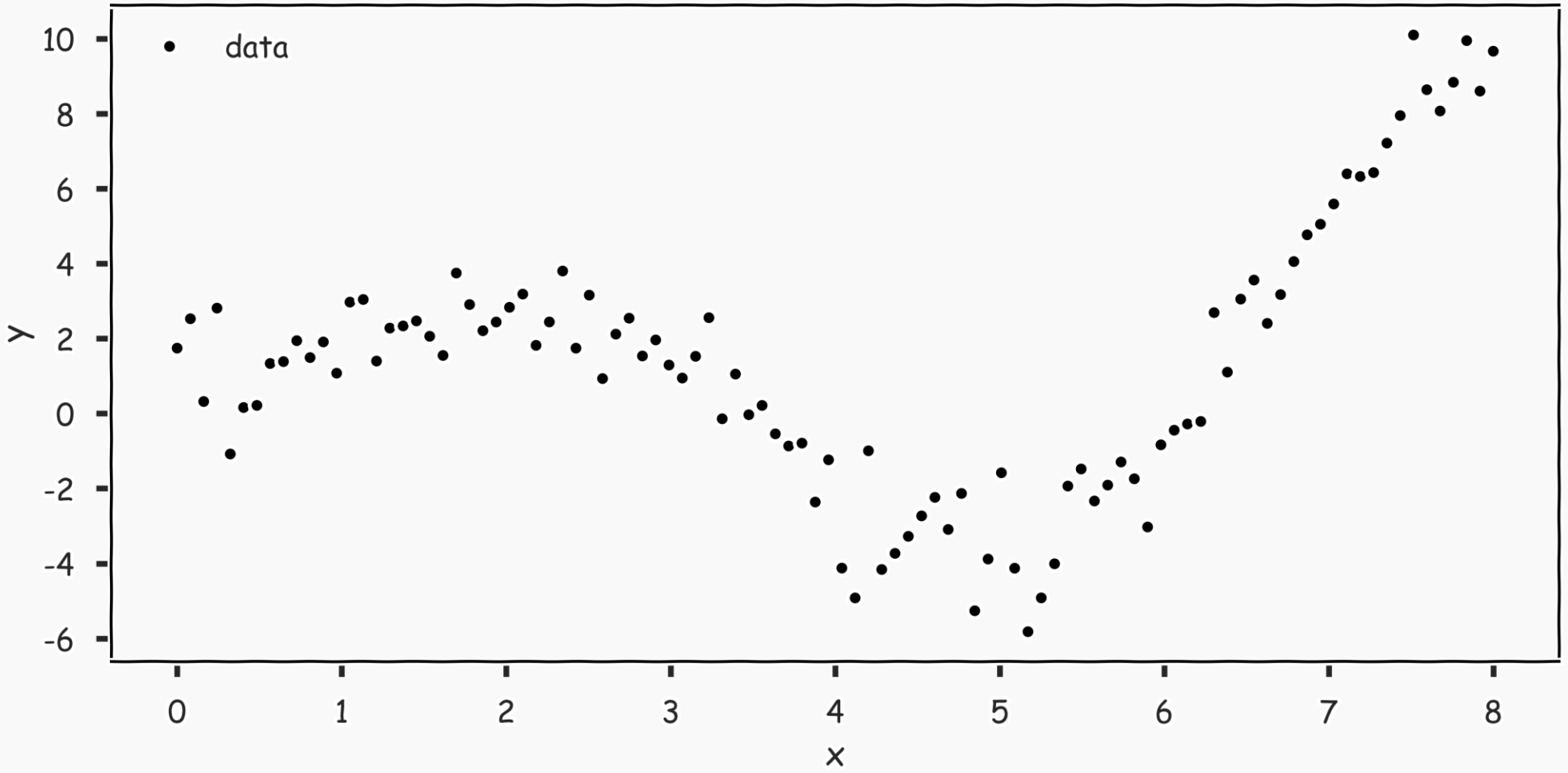
3. Set $T \leftarrow T + \lambda T^{(0)}$

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

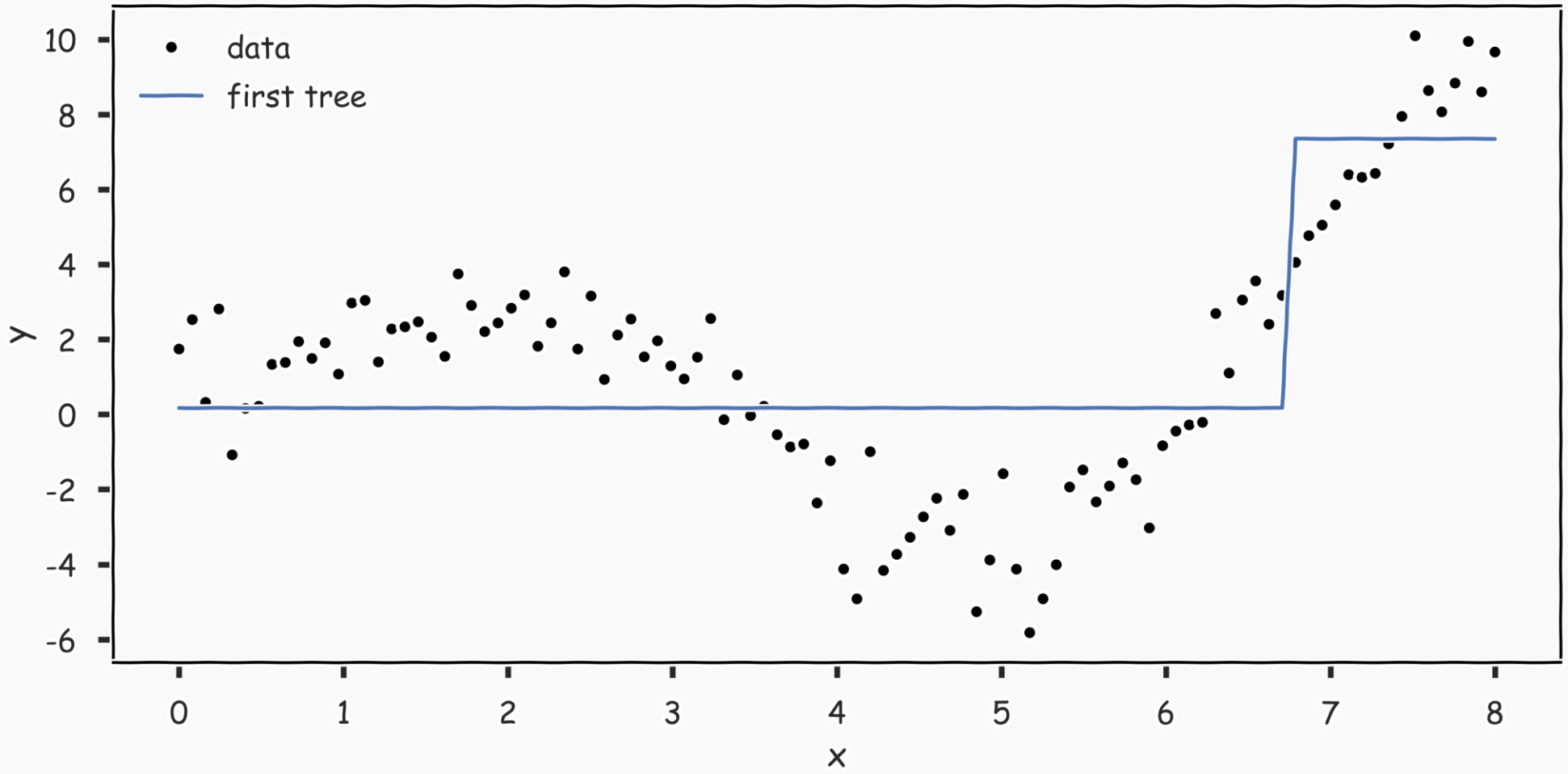
5. Repeat steps 2-4 until ***stopping*** condition met.

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

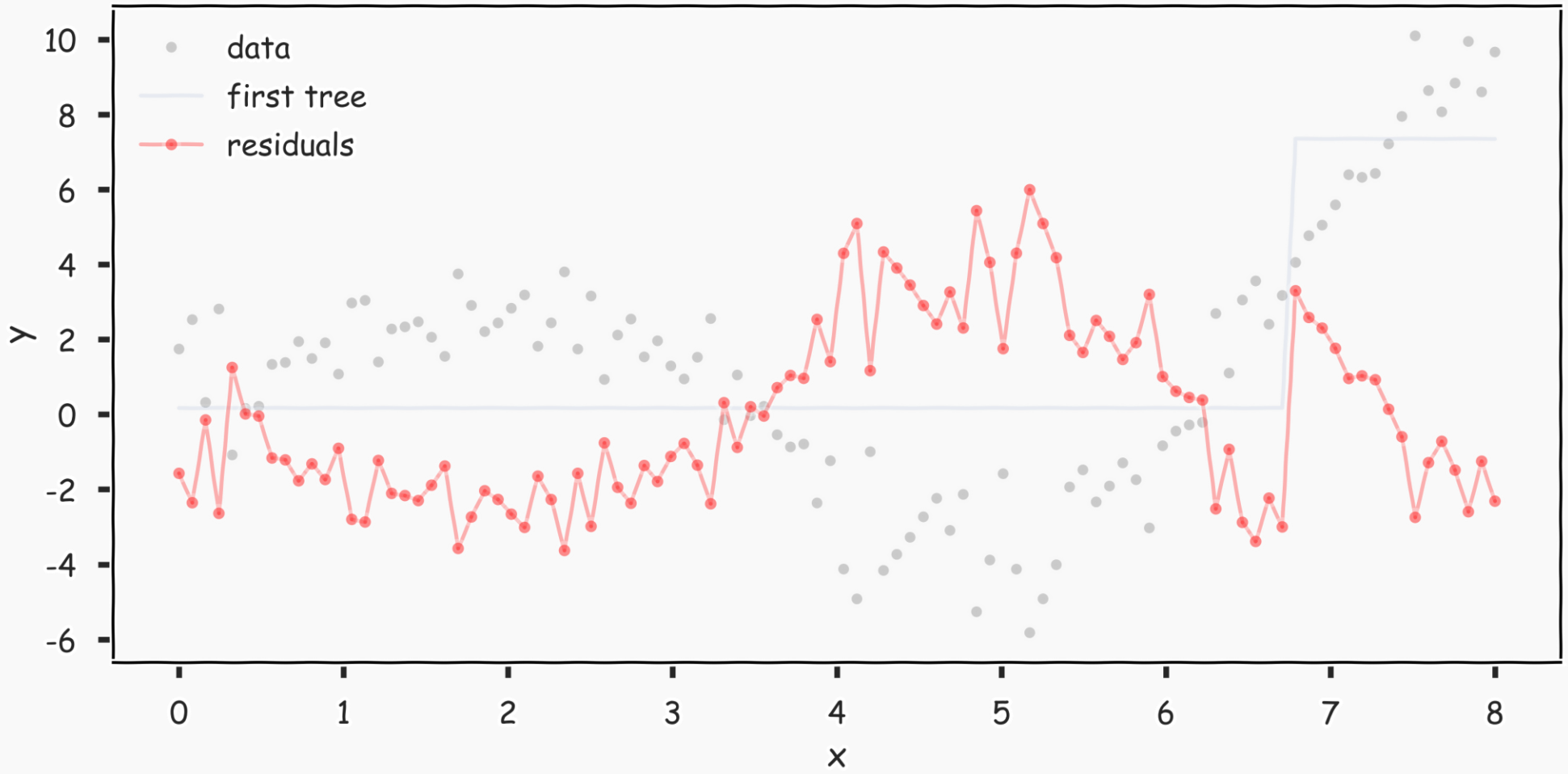
Gradient Boosting: illustration



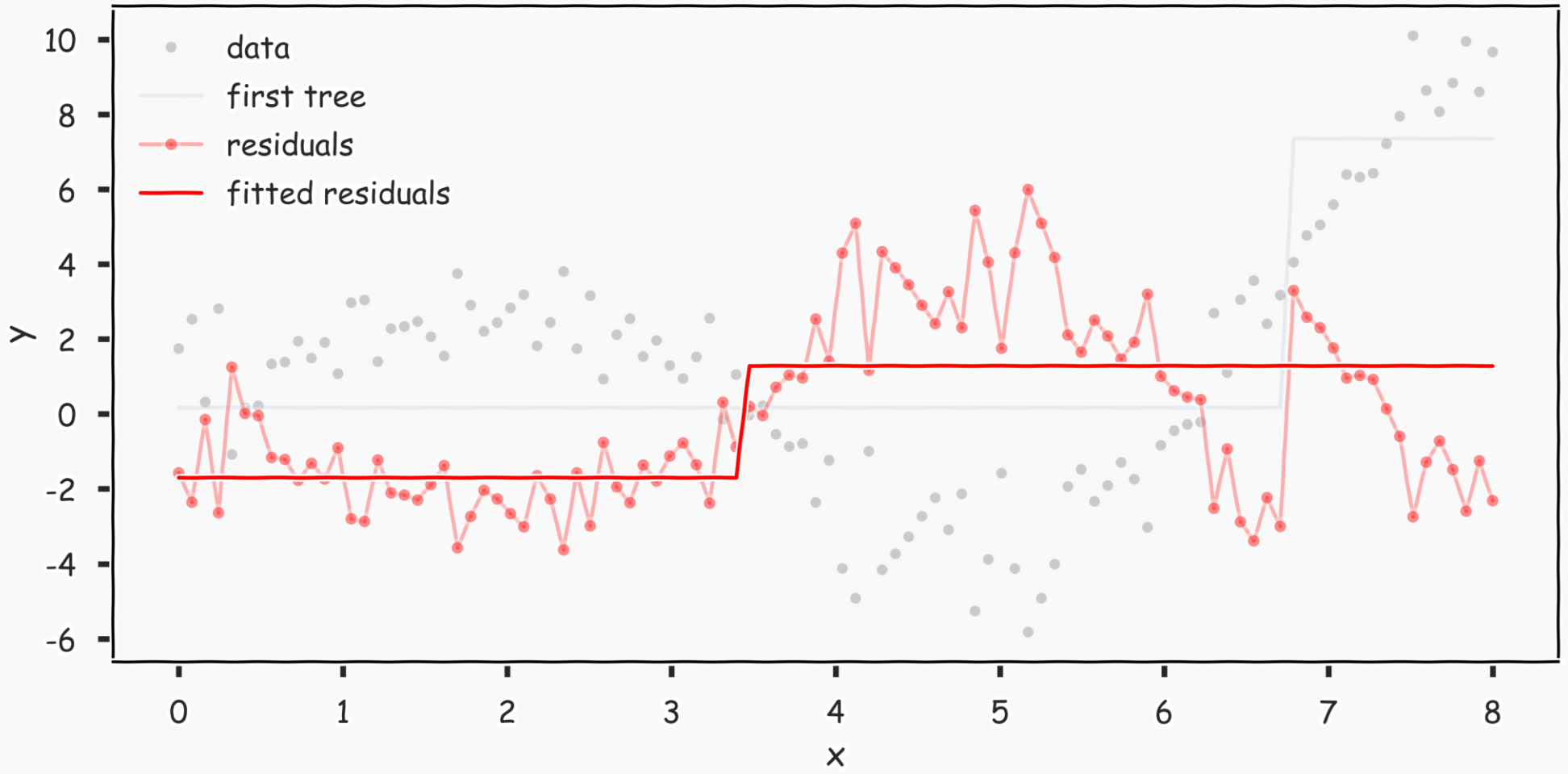
Gradient Boosting: illustration



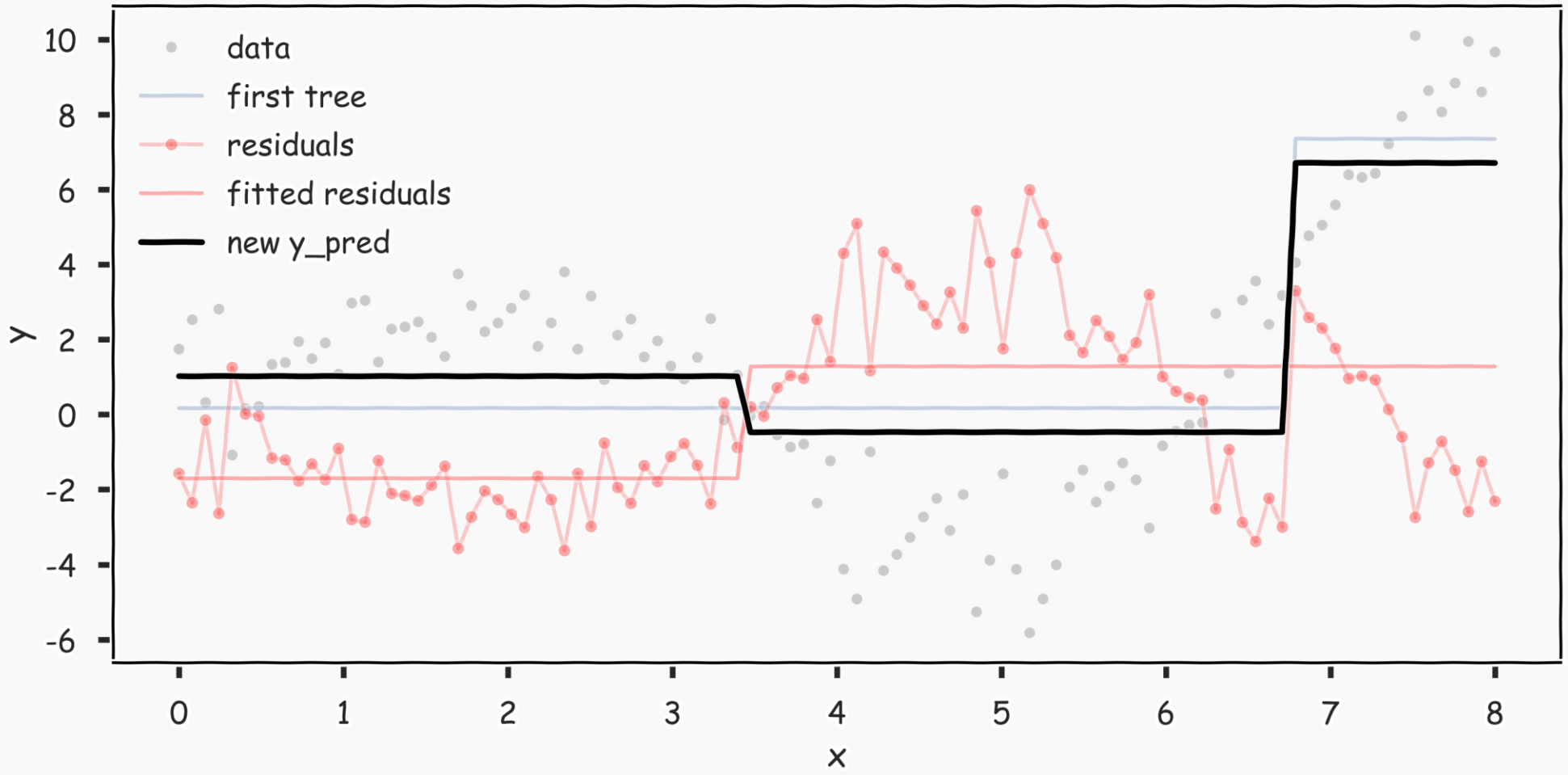
Gradient Boosting: illustration



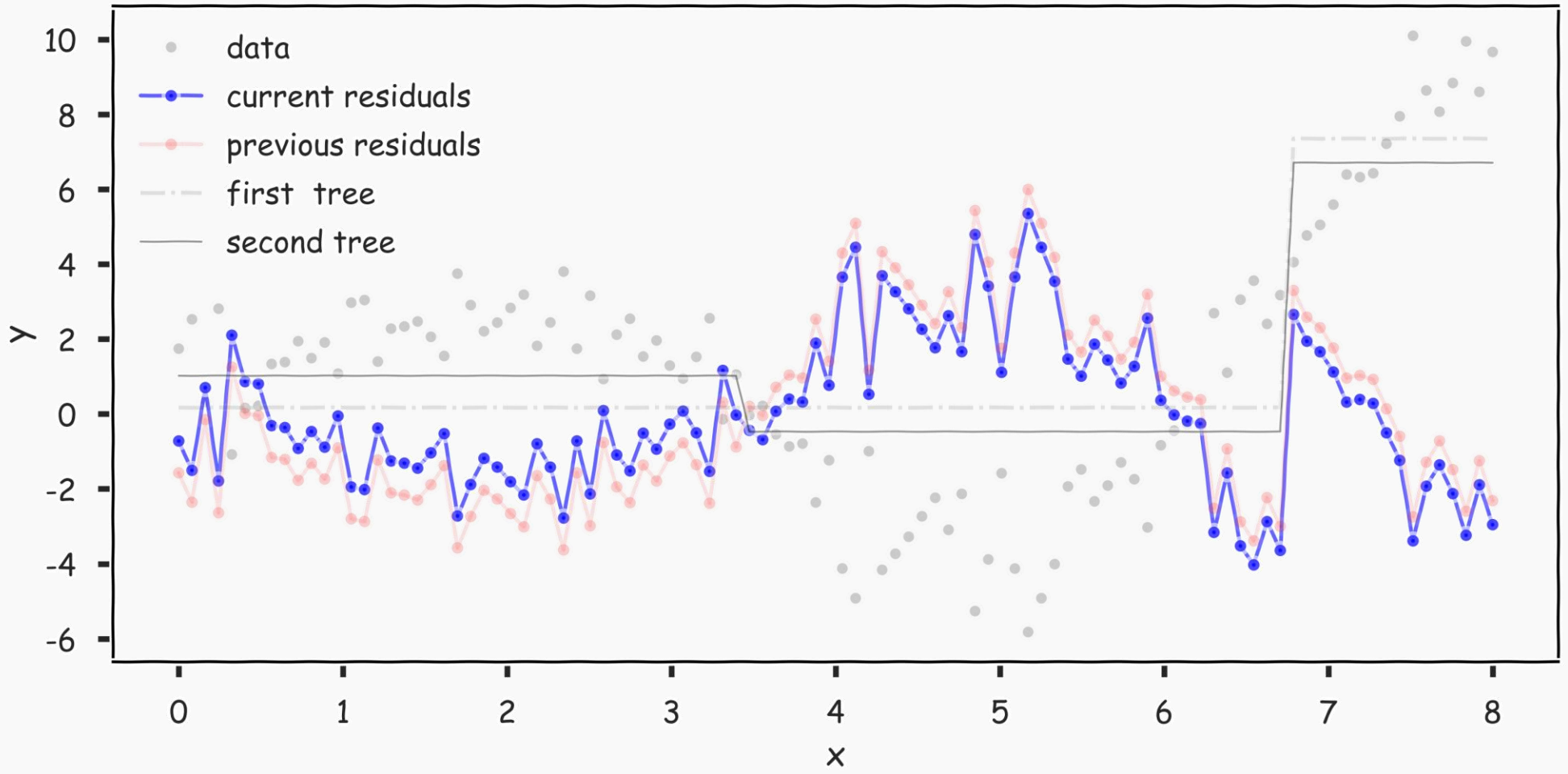
Gradient Boosting: illustration



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***.

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 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:

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

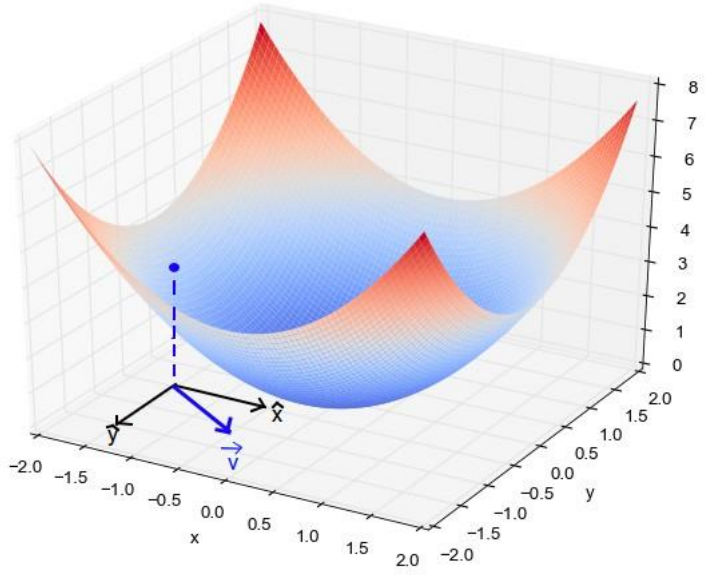
3. Adjust the variables values by some negative multiple of the gradient:

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

The factor λ is often called the learning rate.

Why Does Gradient Descent Work?

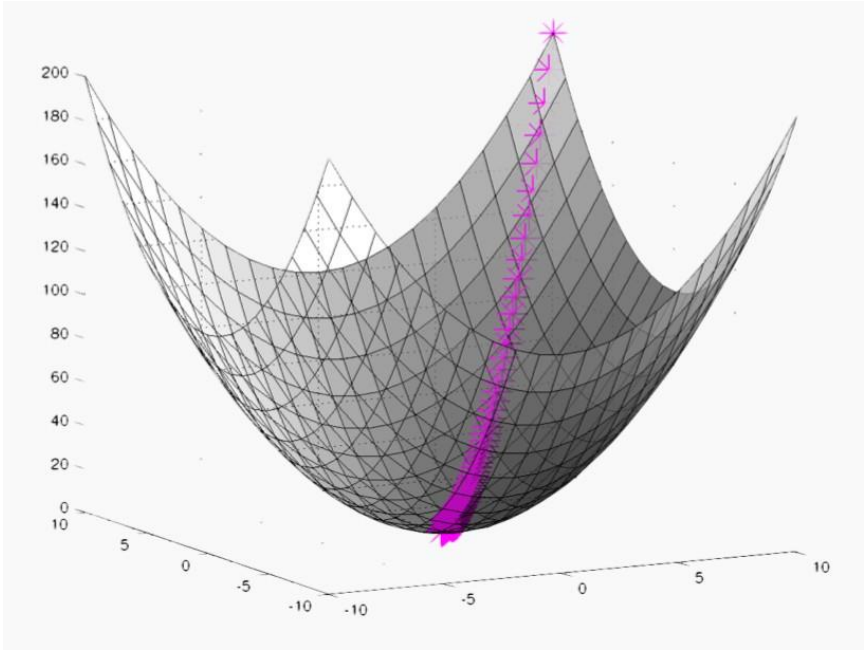
- **Claim:** If the function is convex, this iterative methods will eventually move x close enough to the minimum, for an appropriate choice of λ.
- **Why does this work?** Recall, that as a vector, the gradient at at point gives the direction for the greatest possible rate of increase.



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.)

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 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.

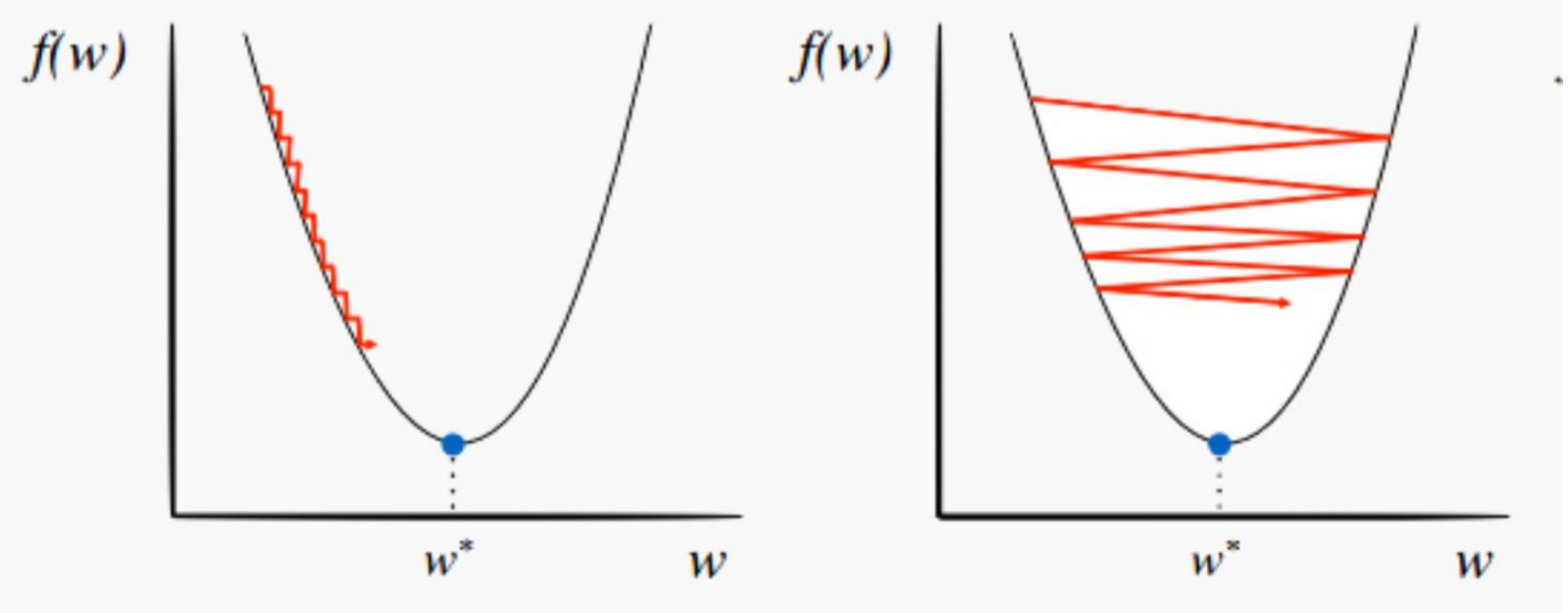
When do we terminate gradient descent?

- 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

Final thoughts on Boosting

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

Boosting in sklearn

Python has boosting algorithms implemented for you:

- **sklearn.ensemble.AdaBoostClassifier**
- **sklearn.ensemble.AdaBoostRegressor**
- With arguments of **base_estimator** (what models to use), **n_estimators** (max number of models to use), **learning_rate** (λ), etc…

Slides Credit

[1] Chapter 4 Introduction to Data Mining By Tan, Steinbach, Kumar.
[2] CS109A Introduction to Data Science by Protopapas and Rader.