Boosting Models

upGrad knowledge hut

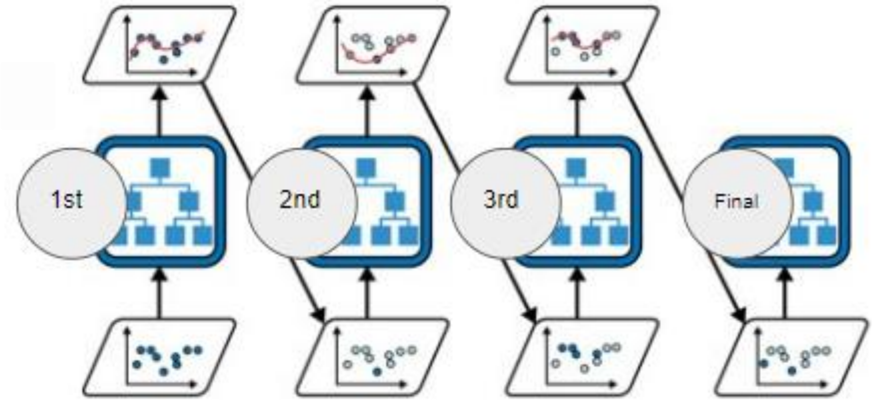
Learning Objectives

- Describe the boosting
- Discuss the concept of AdaBoost
- Explain gradient boosting and demonstrate it in Python
- Differentiate between AdaBoost and gradient boosting
- Explore XGBoost and demonstrate how XGBoost is used

Introduction to Boosting

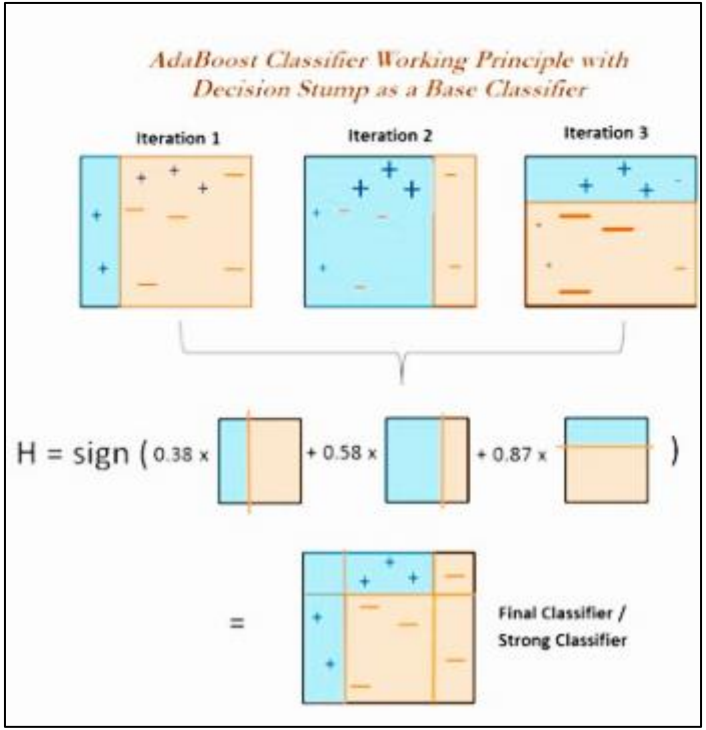
Boosting

Add models sequentially so that each one corrects the mistakes of the previous one.



AdaBoost

Construction Random Forest

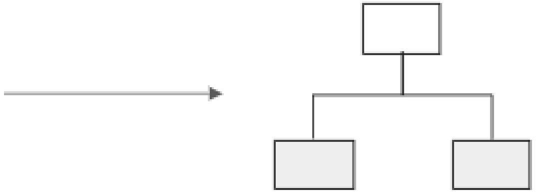


AdaBoost = Adaptive Boosting

How? Adding more weight to them.
Effect: Additional models focus more on the "difficult" cases.

Controlling Variance in Decision Trees

	debt	assets	price	status
0	500	2500	1100	OK
1	250	4500	1500	OK
2	500	2500	1250	default
3	1000	4000	4500	default

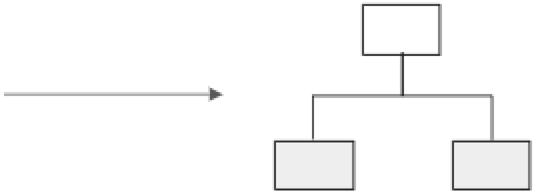


Stump = Decision tree with one node and two leaves (weak learner).

Can only pick one feature to make the split.

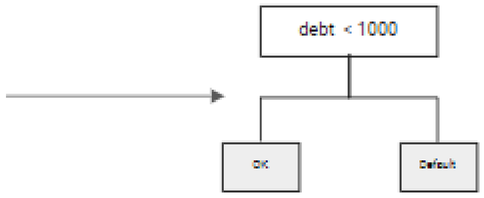
Initial Weight Sample

	debt	assets	price	status	Sample weight
0	500	2500	1100	OK	1/4
1	250	4500	1500	OK	1/4
2	500	2500	1250	default	1/4
3	1000	4000	4500	default	1/4



First Stump

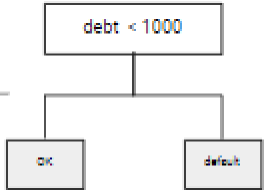
	debt	assets	price	status	Sample weight
0	500	2500	1100	OK	1/4
1	250	4500	1500	OK	1/4
2	500	2500	1250	default	1/4
3	1000	4000	4500	default	1/4



How can you weight the tree according to this error?
First step: Calculate total error.

Total Error

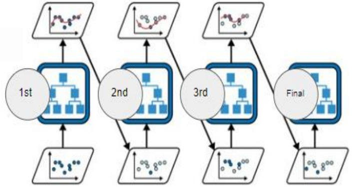
	debt	assets	price	status	Sample weight
0	500	2500	1100	OK	1/4
1	250	4500	1500	OK	1/4
2	500	2500	1250	default	1/4
3	1000	4000	4500	default	1/4



Total Error r_j= Sum of the weights for incorrectly labeled samples = 1/4

$$r_j = \sum_{\substack{i=1 \\ \hat{y}_j^{(i)} \neq y^{(i)}}}^{m} w^{(i)}$$

_j_-th tree

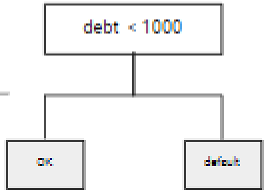


Second step: Use Total Error to calculate model weight alpha.

Model Weight

$$\alpha_j = \eta \log \frac{1 - r_j}{r_j}$$

	debt	assets	price	status	Sample weight
0	500	2500	1100	OK	1/4
1	250	4500	1500	OK	1/4
2	500	2500	1250	default	1/4
3	1000	4000	4500	default	1/4



Model weight = log(1- Total Error / Total Error)

=log(1 - 0.25 / 0.25) = **0.477**

Update the Weight

$$\text{for } i = 1, 2, \cdots, m$$

$$w^{(i)} \leftarrow \begin{cases} w^{(i)} & \text{if } \widehat{y_j}^{(i)} = y^{(i)} \\ w^{(i)} \exp(\alpha_j) & \text{if } \widehat{y_j}^{(i)} \neq y^{(i)} \end{cases}$$

	debt	assets	price	status	Sample weight
0	500	2500	1100	OK	1/4
1	250	4500	1500	OK	1/4
2	500	2500	1250	default	1/4
3	1000	4000	4500	default	1/4

Updated sample weight = sample weight * exp(*alpha*) = 0.25 * exp0.477 = **0.40**

Increase Weight of Misclassified Example

$$\text{for } i = 1, 2, \cdots, m$$

$$w^{(i)} \leftarrow \begin{cases} w^{(i)} & \text{if } \widehat{y_j}^{(i)} = y^{(i)} \\ \boxed{w^{(i)} \exp(\alpha_j) \quad \text{if } \widehat{y_j}^{(i)} \neq y^{(i)}} \end{cases}$$

	debt	assets	price	status	Sample weight
0	500	2500	1100	OK	1/4
1	250	4500	1500	OK	1/4
2	500	2500	1250	default	1/4
3	1000	4000	4500	default	1/4

	debt	assets	price	status	Sample weight
0	500	2500	1100	OK	1/4
1	250	4500	1500	OK	1/4
2	500	2500	1250	default	0.4
3	1000	4000	4500	default	1/4

Updated sample weight = sample weight * exp(*alpha*) = 0.25 * exp0.477 = **0.40**

Weights get **boosted:** (High boosting when alpha is large, low boosting when alpha is small)

Normalize Weight

	debt	assets	price	status	Sample weight
0	500	2500	1100	OK	0.25
1	250	4500	1500	OK	0.25
2	500	2500	1250	default	0.4
3	1000	4000	4500	default	0.25

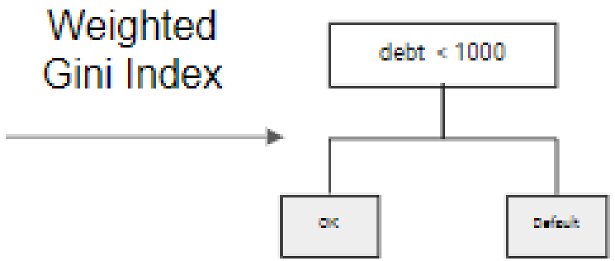
Total = 1.15

	debt	assets	price	status	Updated Sample weight
0	500	2500	1100	OK	0.25 / 1.15 = 0.22
1	250	4500	1500	OK	0.25 / 1.15 = 0.22
2	500	2500	1250	default	0.4 / 1.15 = 0.35
3	1000	4000	4500	default	0.25 / 1.15 = 0.22

Total ≈ 1

Train Next Stump on Modified Weights

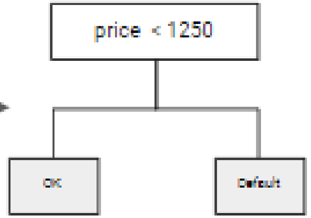
	debt	assets	price	status	Updated Sample weight
0	500	2500	1100	OK	0.22
1	250	4500	1500	OK	0.22
2	500	2500	1250	default	0.34
3	1000	4000	4500	default	0.22



New Sample Based on New Weights

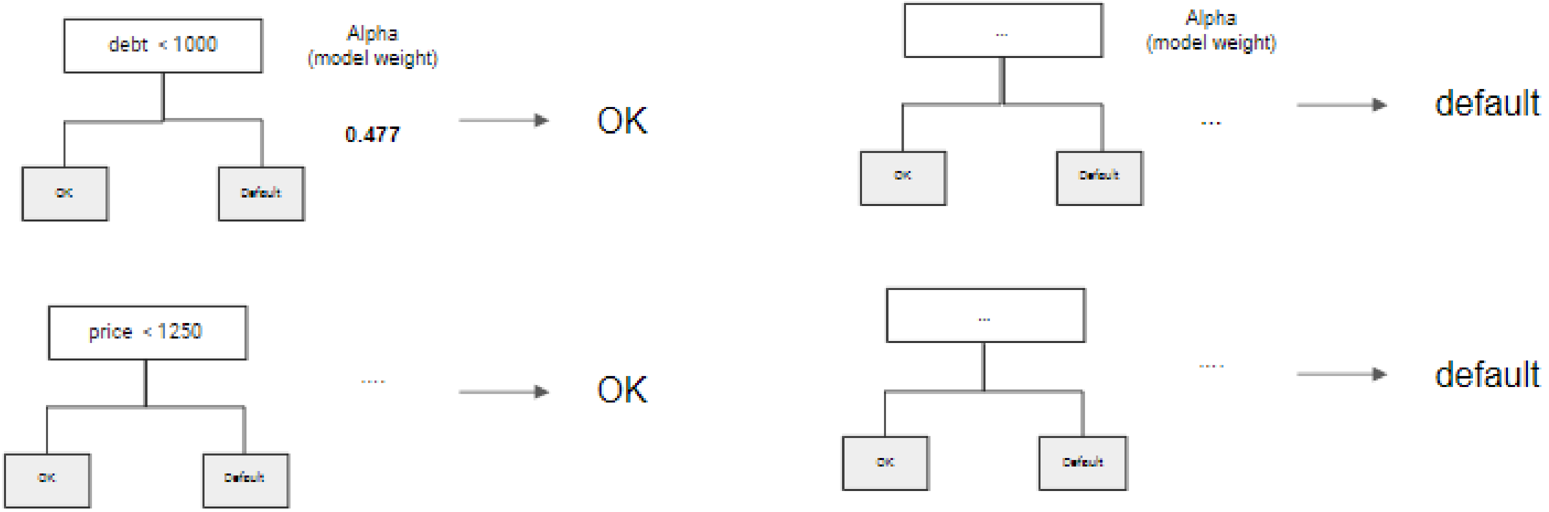
	debt	assets	price	status	Updated Sample weight
0	500	2500	1100	OK	0.22
1	250	4500	1500	OK	0.22
2	500	2500	1250	default	0.34
3	1000	4000	4500	default	0.22

	debt	assets	price	status	Sample weight
0	500	2500	1100	OK	1/4
2	500	2500	1250	default	1/4
2	500	2500	1250	default	1/4
3	1000	4000	4500	default	1/4



Prediction

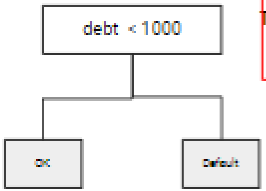
	debt	assets	price	status
4	250	5000	3000	??



Add Weights

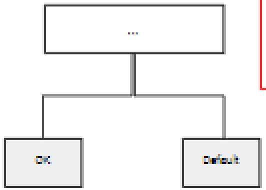
	debt	assets	price	status
4	250	5000	3000	??

$$\hat{y}(\mathbf{x}) = \underset{k}{\operatorname{argmax}} \boxed{\sum_{\substack{j=1 \\ \hat{y}_j(\mathbf{x})=k}}^{N} \alpha_j}$$



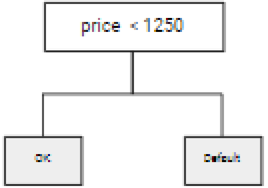
Total Weight = 1.4

→ OK



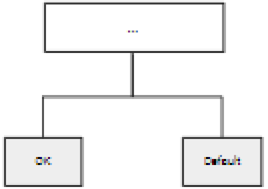
Total Weight = 0.2

→ default



....

→ OK



....

→ default

Choose Class Using argmax

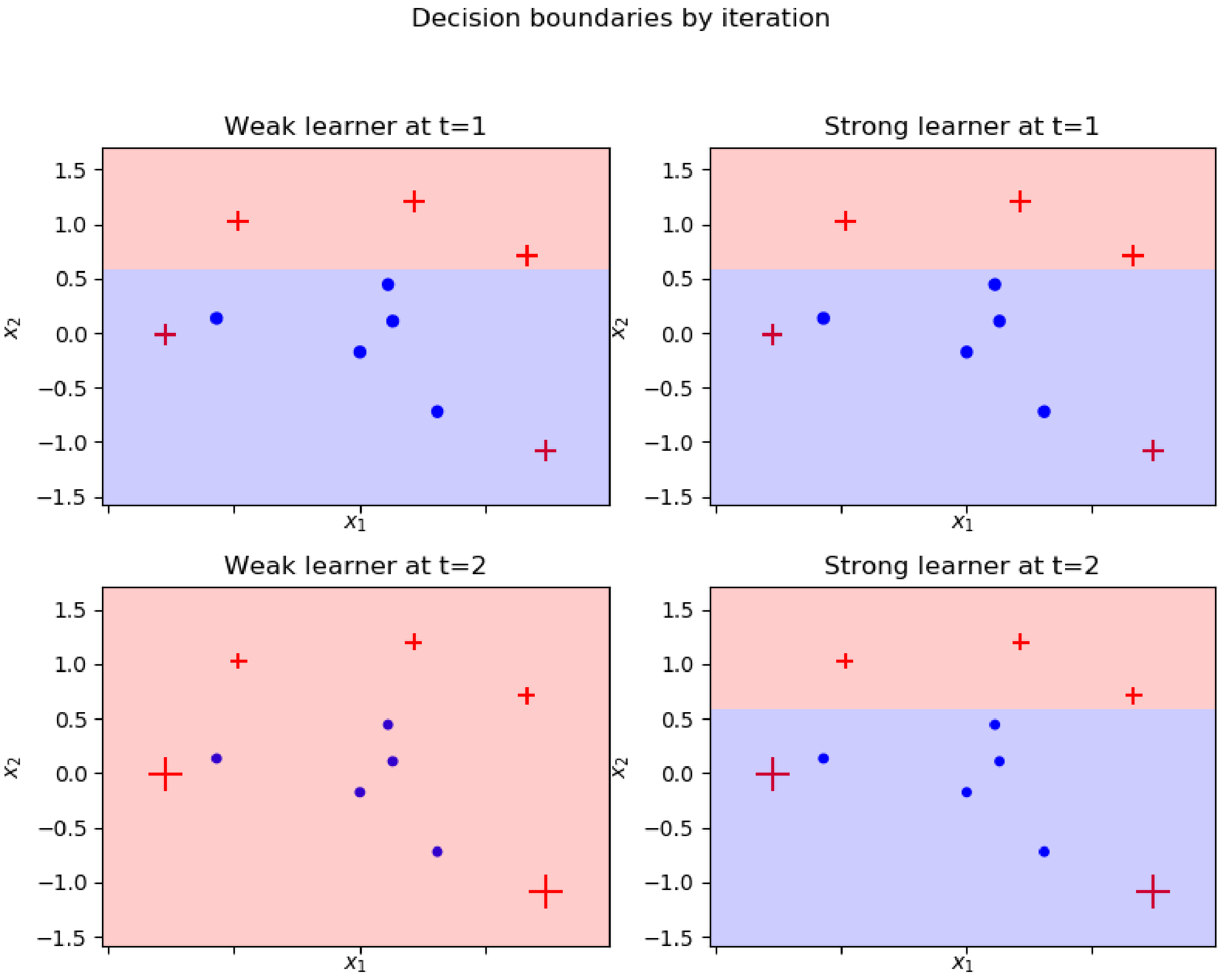
	debt	assets	price	status
4	250	5000	3000	OK

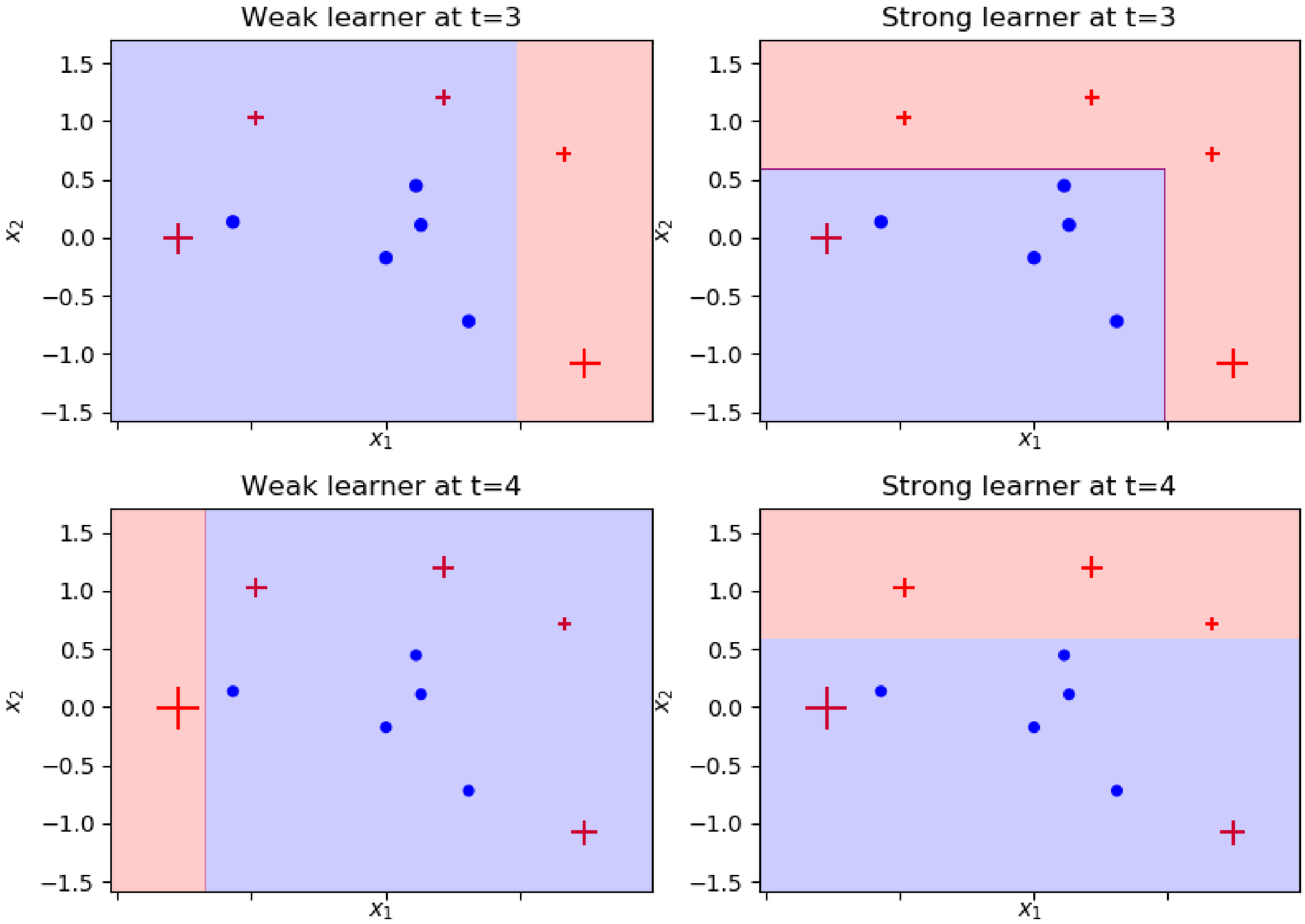
$$\hat{y}(\mathbf{x}) = \boxed{\underset{k}{\operatorname{argmax}}} \sum_{\substack{j=1 \\ \hat{y}_j(\mathbf{x})=k}}^{N} \alpha_j$$

$\boxed{\text{Total Weight} = 1.4} \longrightarrow$ OK > $\boxed{\text{Total Weight} = 0.2} \longrightarrow$ default

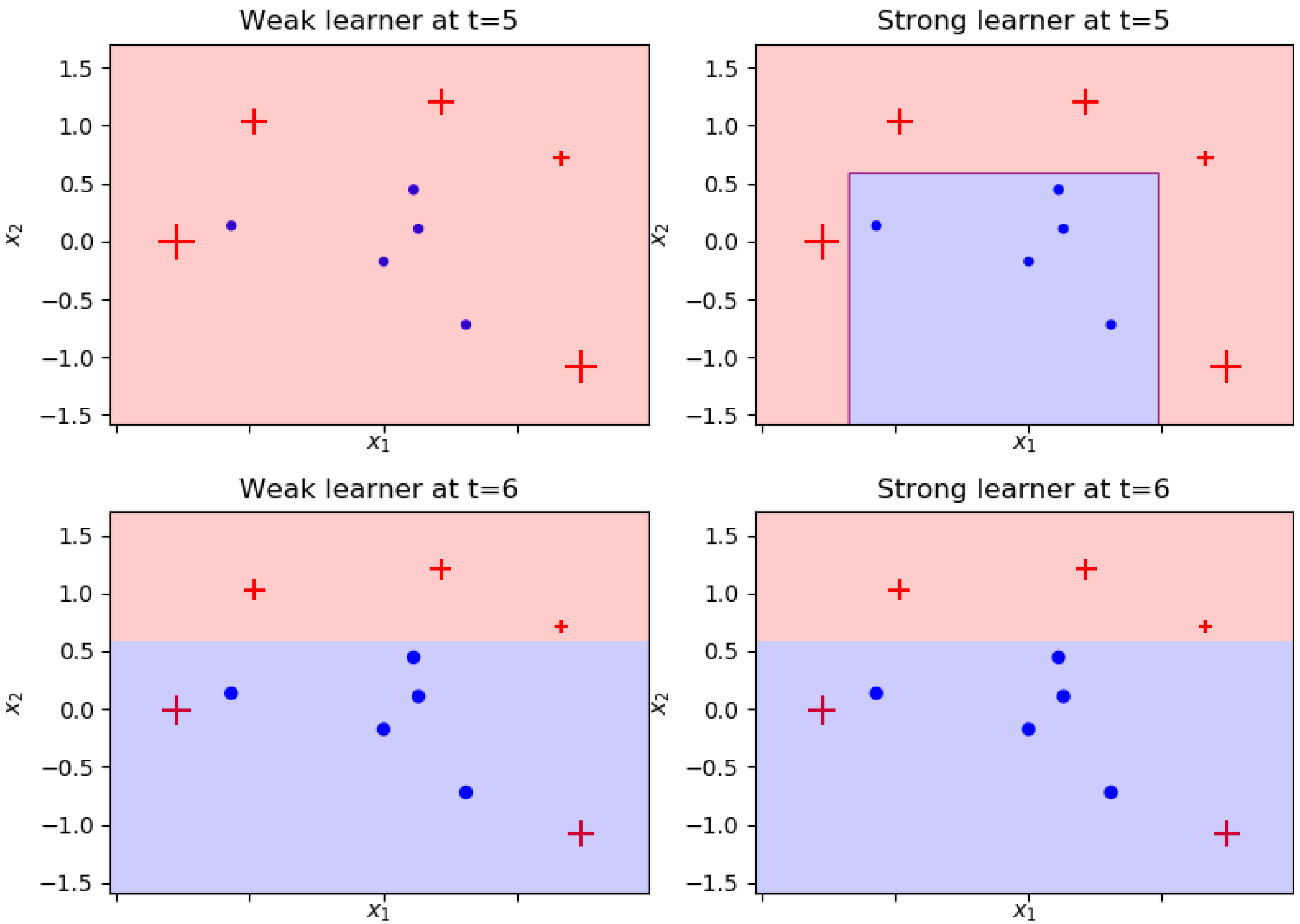
Compute the predictions from all models and weigh them according to their model weights *α*.

Predicted class is the one that receives the majority of weighted votes.

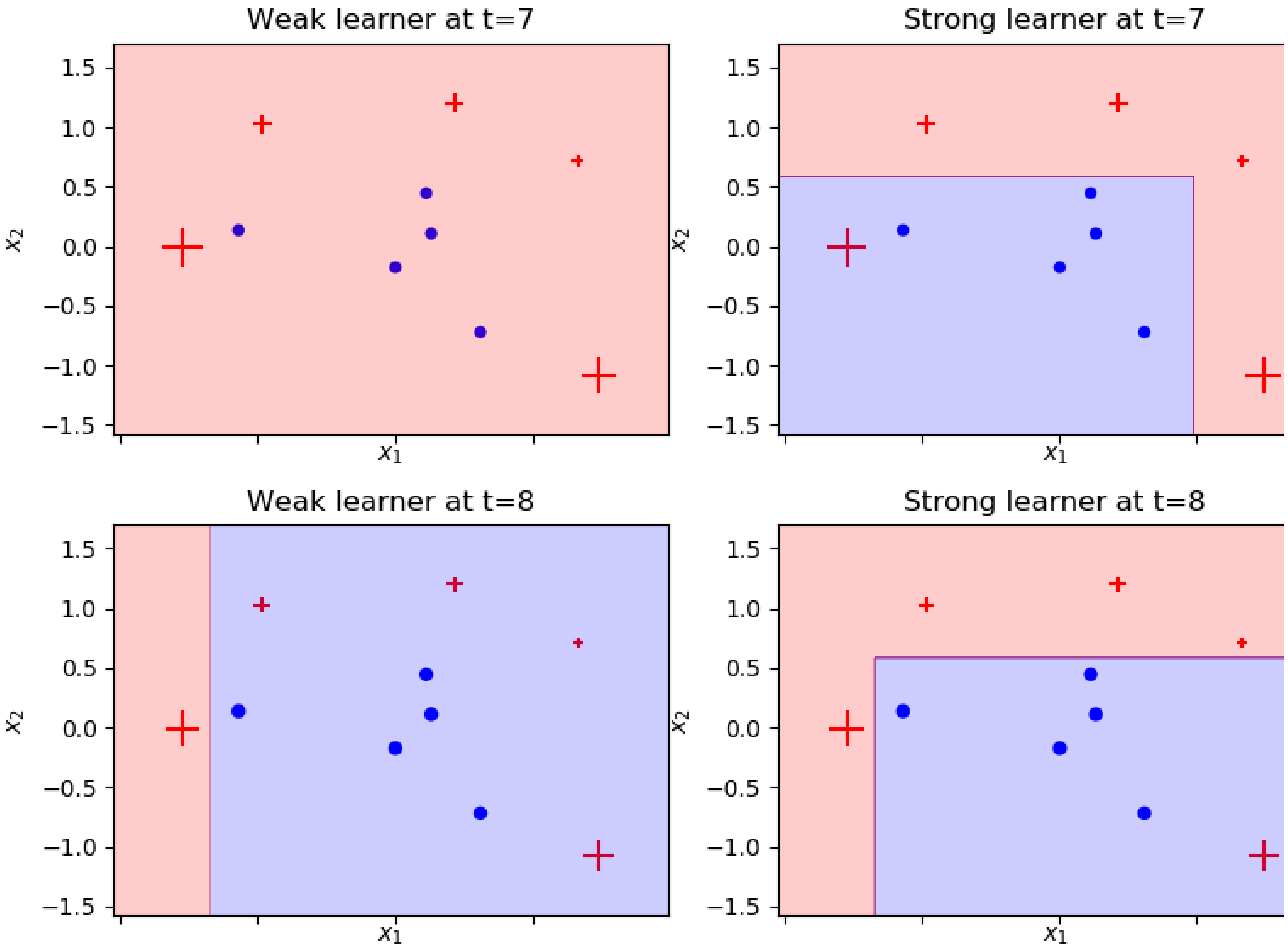

Decision boundaries by iteration
Weak learner at t=1
Strong learner at t=1
Weak learner at t=2
Strong learner at t=2
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
x_1
x_2

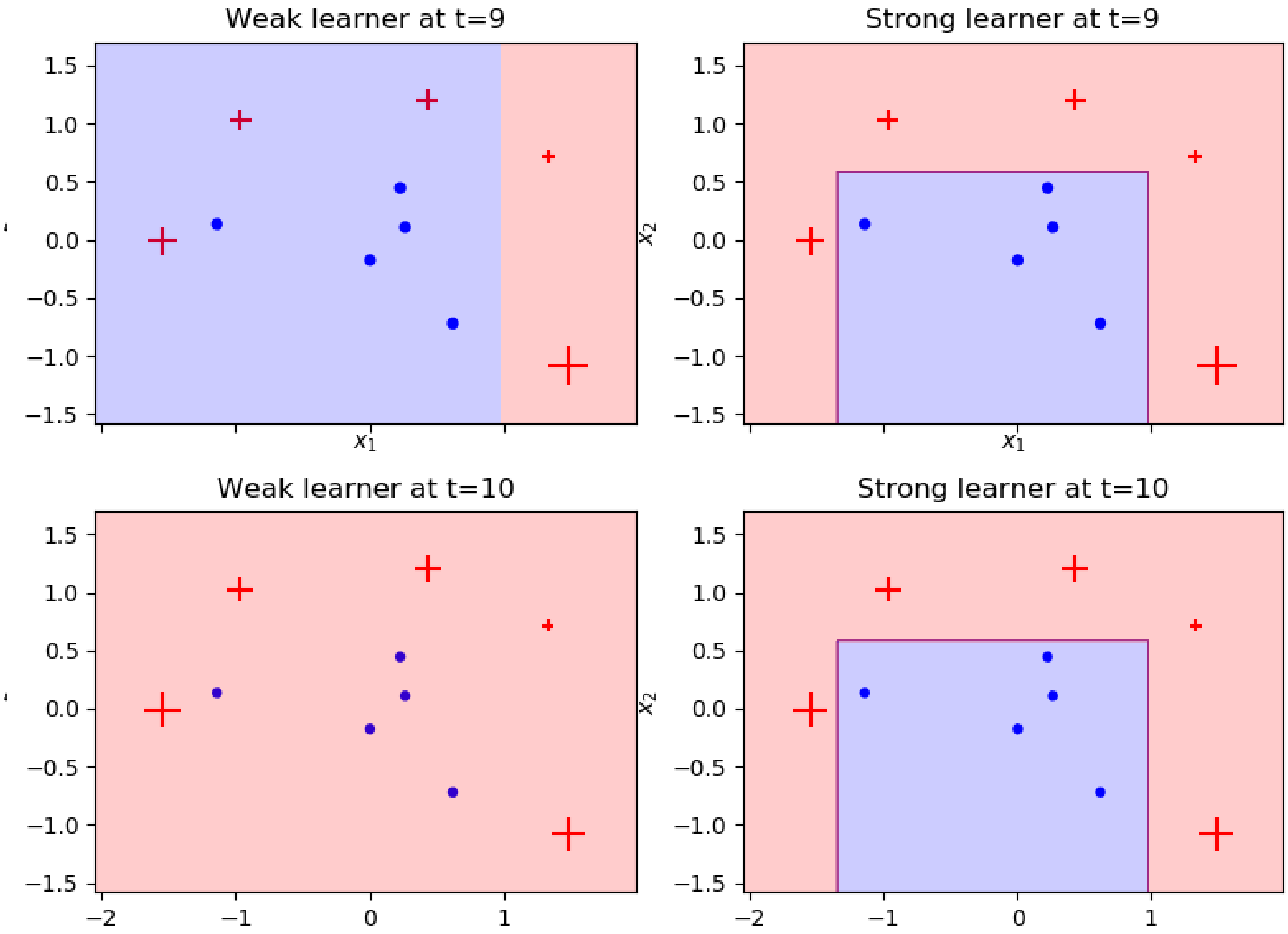


Weak learner at t=3
Strong learner at t=3
Weak learner at t=4
Strong learner at t=4
x_1
x_2
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5



Weak learner at t=5
Strong learner at t=5
Weak learner at t=6
Strong learner at t=6
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
x_1
x_2


Weak learner at t=7
Strong learner at t=7
Weak learner at t=8
Strong learner at t=8
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
x_1
x_2



Weak learner at t=9
Strong learner at t=9
Weak learner at t=10
Strong learner at t=10
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2
−1
0
1
x_1
x_2

Boosting Advantages and Disadvantages

Advantages

- Very good performance, especially for classification problems
- Robust to different data types
- Can handle missing data

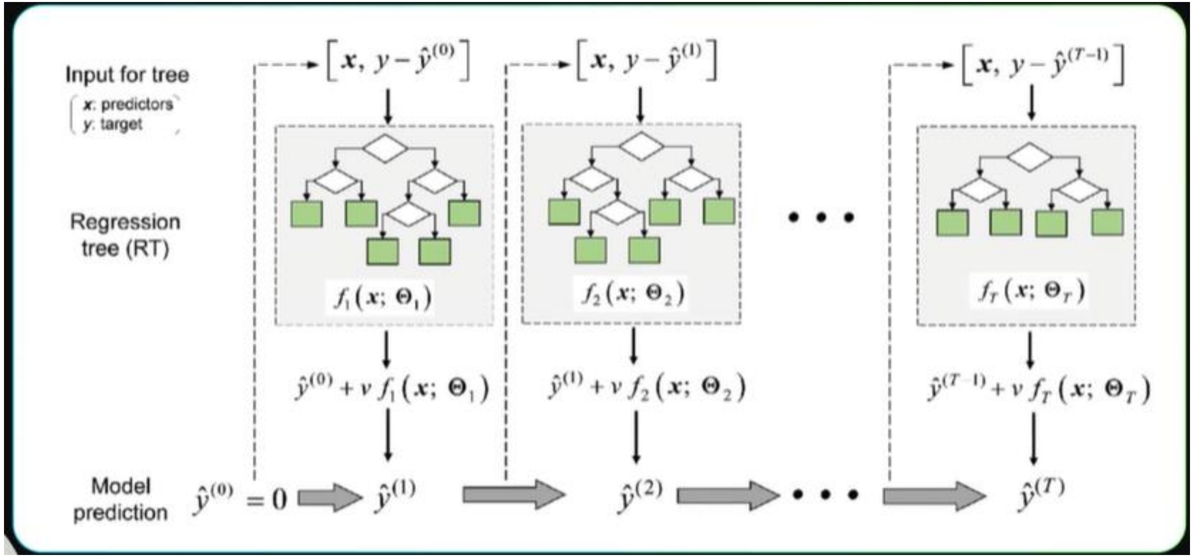
Disadvantages

- Computationally expensive
- Training cannot be parallelized (models must be trained sequentially)
- Harder to implement for real-time predictions
- Overfitting with poor parametrization

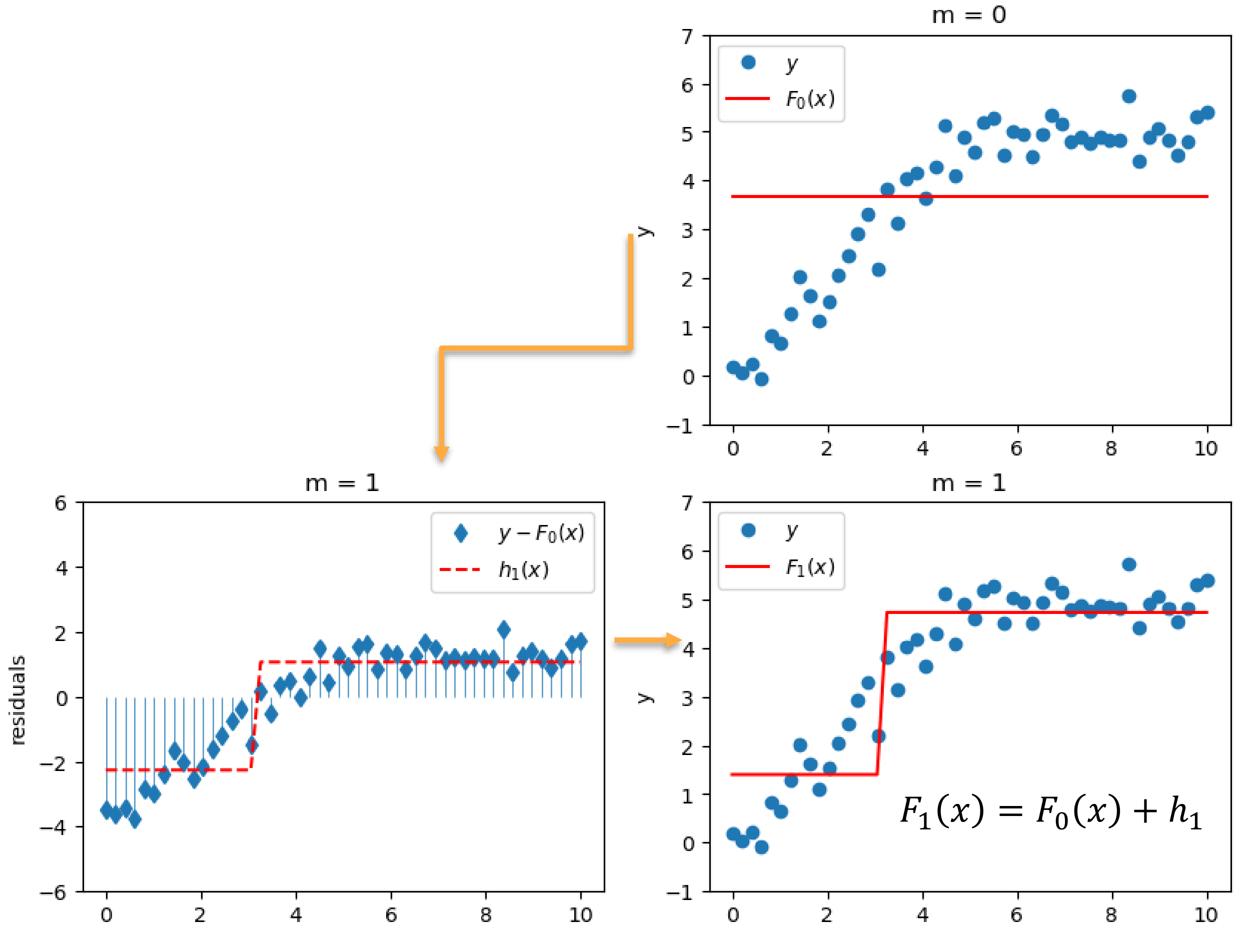
Gradient Boosting SKLearn

Intuition of Gradient Boosting

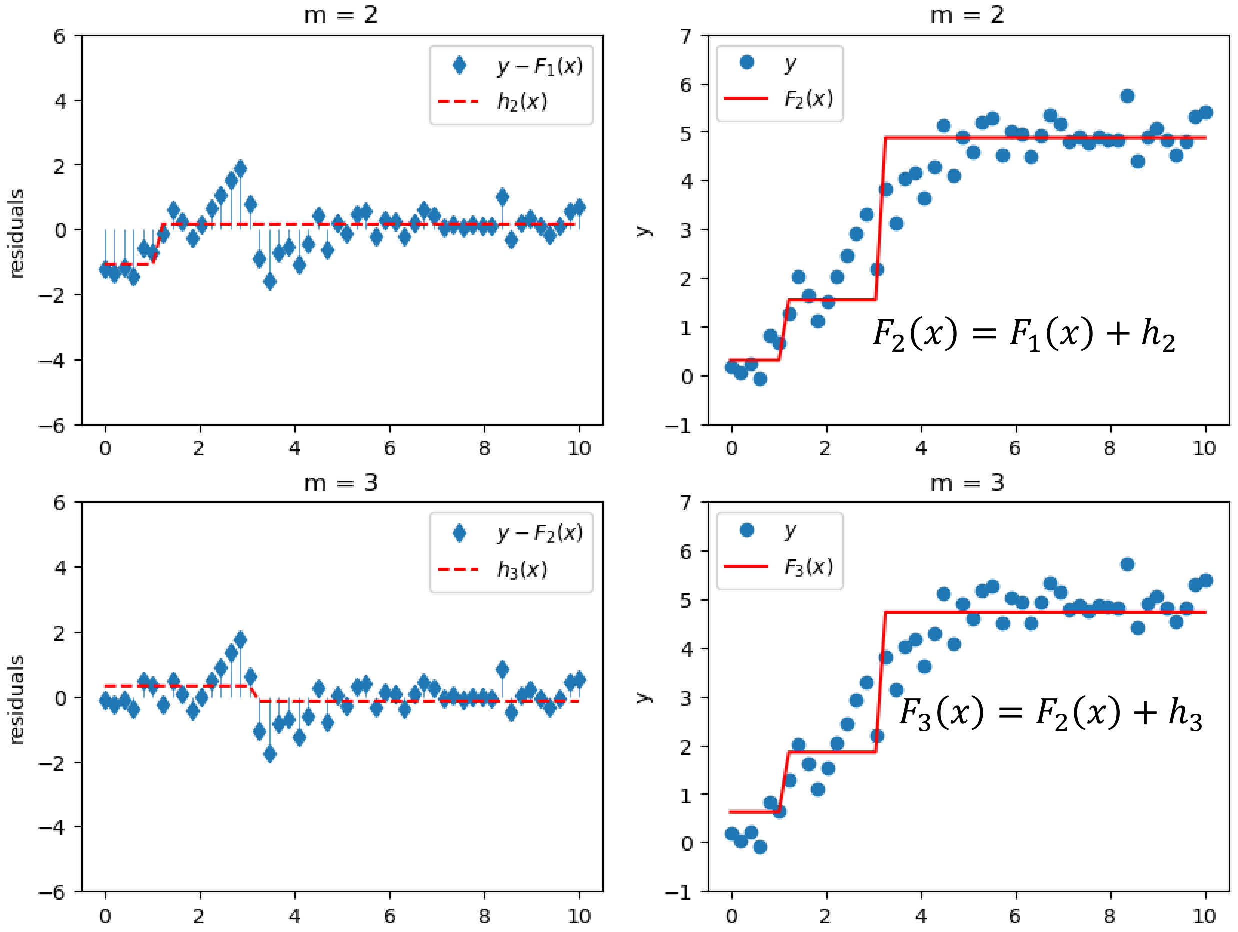
Works sequentially, trying to correct previous models' errors (just like AdaBoost)
Instead of tweaking weights, it trains new models based on the residuals.



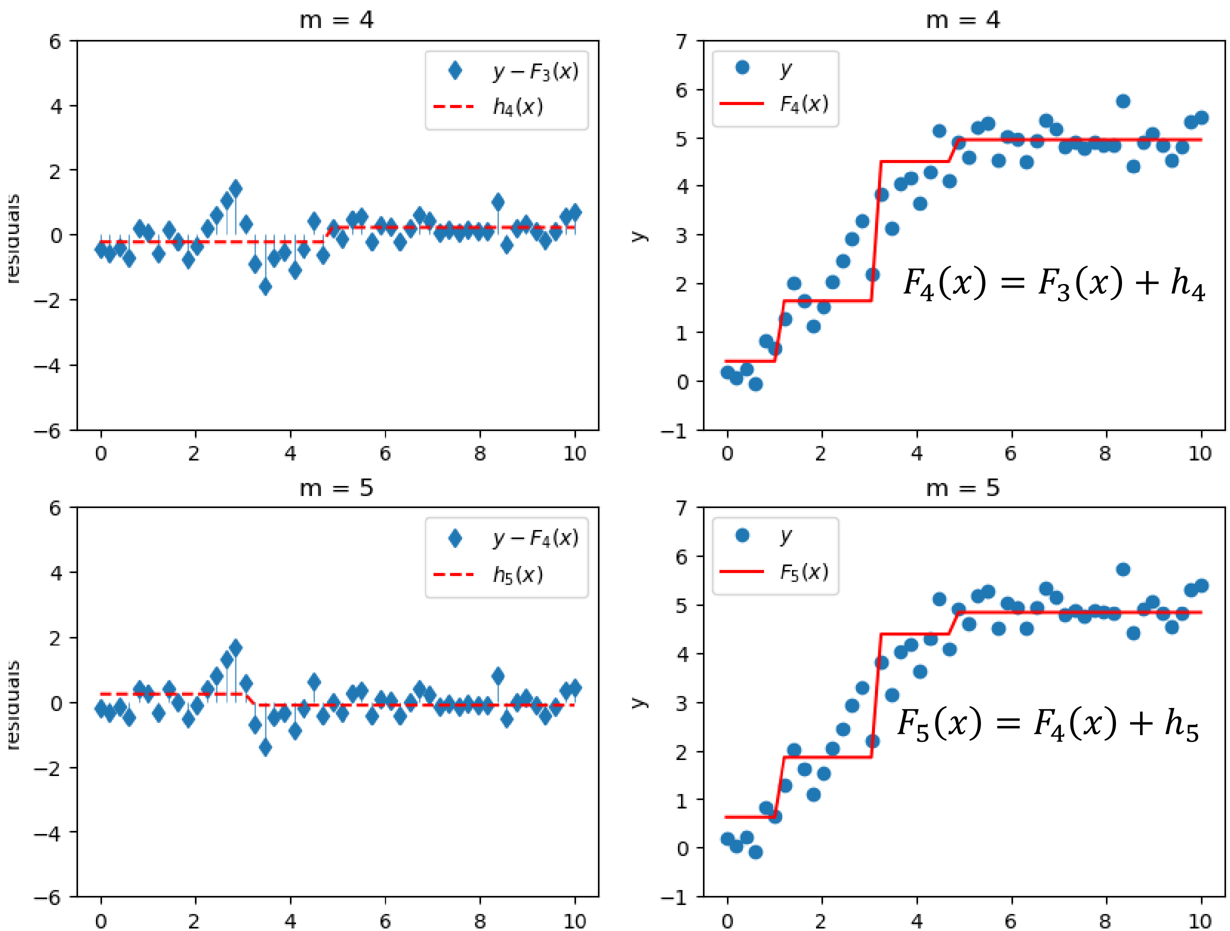
Intuition of Gradient Boosting



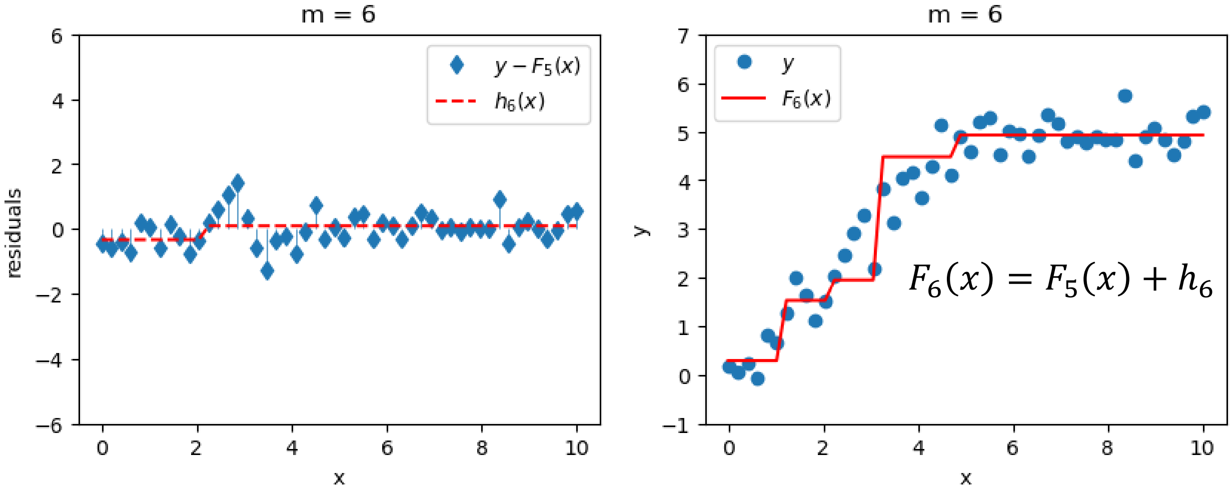
Intuition of Gradient Boosting



Intuition of Gradient Boosting



Intuition of Gradient Boosting



Gradient Boosting

With the learning rate η, the update step will then look like

$$F_m(x) = F_{m-1}(x) + \eta h_m(x),$$

and our composite model will look like

$$F_M(x) = F_0(x) + \eta \sum_{m=1}^{M} h_m(x)$$

AdaBoost vs. Gradient Boosting

AdaBoost and Gradient Boosting

	AdaBoost	Gradient Boosting
Models	Typically, “stumps” with low depth (2 - 8 leaves)	Stronger learners with typically 8 - 30 leaves
Weights	Each model gets a different weight	All models are weighed equally, impact controlled by learning rate
Variance	Assigns different weights to all training examples - capture maximum variance	Tries to capture variance based on previous models’ errors (residuals)
Hyperparameters	Limited hyper parameters such as learning rate and number of models, base learner	Various parameters such as learning rate, loss function, or tolerance threshold for early stopping

Algorithm Used

AdaBoost	Gradient Boosting
Longest legacy → not used as much anymore for new projects, but can be still found in existing systems	Generally stronger performance than AdaBoost
Faster than Gradient Boosting	More complex / longer runtine
Mostly lower performance	Requires more hyper parameter tuning
	Higher risk of overfitting when parameters are not optimized
	Foundation for more advanced algorithms like XGBoost

XGBoost

XGBoost Background

*XGBoost is an optimized **distributed** gradient boosting library designed to be highly **efficient**, **flexible** and **portable.***

XGBoost provides a parallel tree boosting (also known as GBDT, GBM) that solve many data science problems in a fast and accurate way.

Available for R, Python, Ruby, Julia, C, and many more.

Regularization

XGBoost offers different techniques for regularization that all aim to reduce the potential of overfitting.

- L1 and L2 Regularization to penalize additional features in the dataset that do not add significantly value.
- Gamma parameter to control tree complexity and set a limit for minimum loss required to make further splits (similar min_impurity_decrease).

Parallelization

XGBoost offers ability to parallelize computation - increase training speed

- Trees are still trained sequentially, but the computation of a single tree can be parallelized with specific pre-sorting and block storage methods.
- Result: Even very large datasets can be processed faster, and the individual trees can be created in less time.

Dropouts

Applies concept of dropout to the training process of boosted trees.

Randomly remove trees from the ensemble during training to make the model generalize better and prevent overfitting.

DART = DART: Dropout Additive Regression Trees (DART)

Thank you