

Boosting

ML2: AI Concepts and Algorithms (SS2025)

Faculty of Computer Science and Applied Mathematics

University of Applied Sciences Technikum Wien

Lecturer: Rosana de Oliveira Gomes

Authors: B. Knapp, S. Lackner, S. Rezagholi, R.O. Gomes



Boosting: Intro

*Boosting belong to a family of machine learning algorithms that allow the construction of a **strong learner from many weak learners**. It **combines models iteratively such that the next model depends on the errors of the previous model**.*

History

Kearns and Valiant 1989: Theoretical developments (*hypothesis boosting problem*).

Freund and Schapire 1997: First practical algorithm (**AdaBoost**).

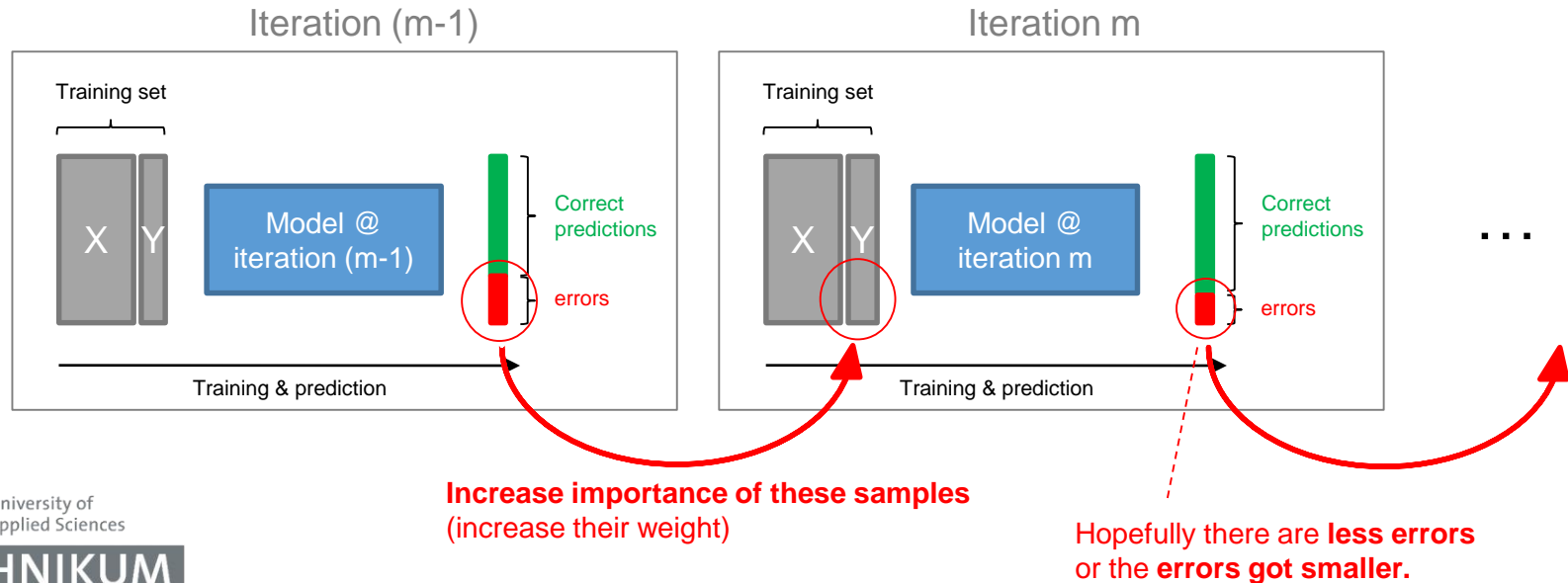
Friedman et al. 2001: Further developments (**gradient boosting**).

Friedman 2002: Generalization of gradient boosting (**stochastic gradient boosting**).

...Since 2016: Highly scalable versions (**e.g. XGBoost, LightGBM**).

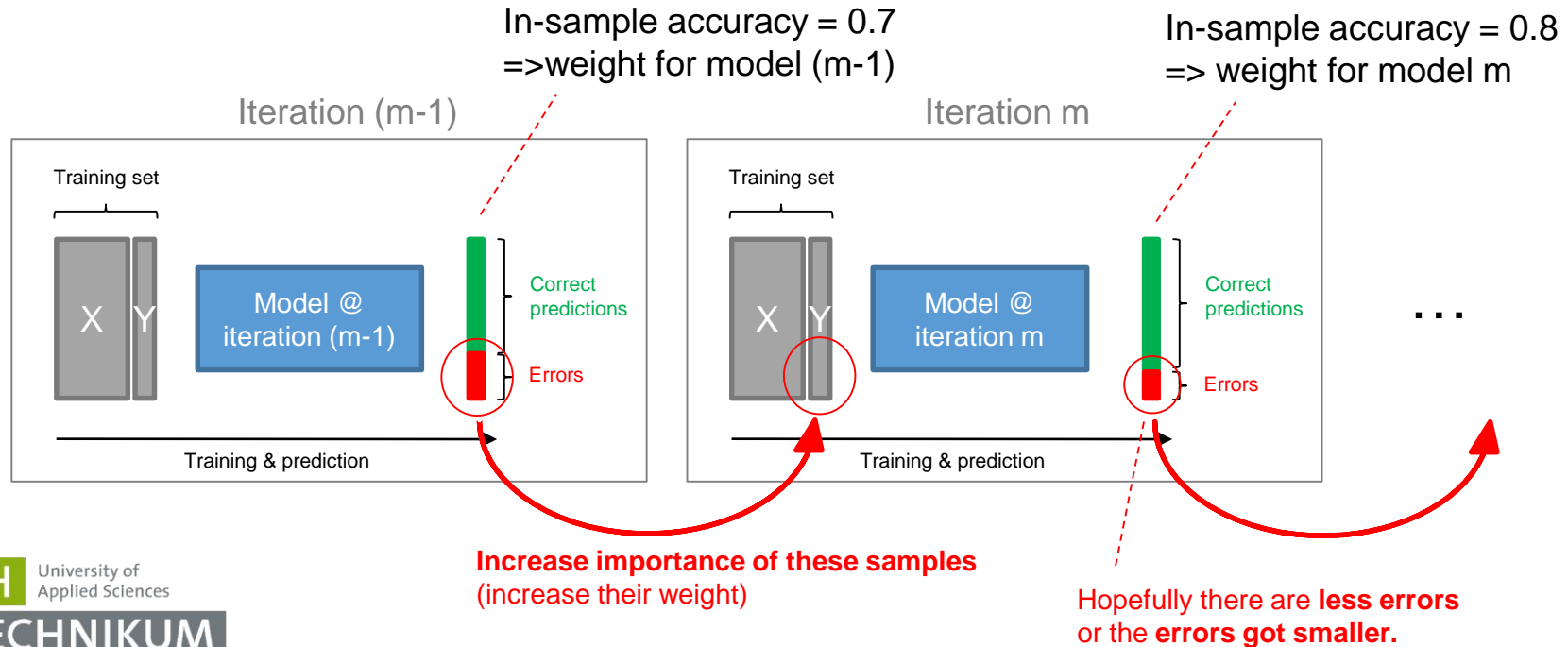
Boosting: Overview

- Boosting is an ensemble method in which we **build predictors in a stage-wise fashion** and each predictor is **informed by the errors of the previous predictor**.
- General idea: **The next learner focuses on the cases that the learners before it could not handle.**



Boosting: Overview

- Also each learner has a **weight**, e.g. its relative performance. The learners' individual predictions are combined by the respective **weighted sum**.



Boosting: Overview

There are 2 primary boosting methodologies:

1. Increase the **weight** of observations for which the previous models failed (Example: AdaBoost).
2. Fit the next model to the **errors** of the previous models (Examples: Gradient boosting, XGBoost).

Adaptive Boosting: AdaBoost

- Perhaps the most popular boosting algorithm.
- May be used for **classification or regression**.

A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting

Yoav Freund

Robert E. Schapire

AT&T Bell Laboratories
600 Mountain Avenue

Murray Hill, NJ 07974-0636
{yoav, schapire}@research.att.com

1995

Explaining AdaBoost

Robert E. Schapire

Abstract Boosting is an approach to machine learning based on the idea of creating a highly accurate prediction rule by combining many relatively weak and inaccurate rules. The AdaBoost algorithm of Freund and Schapire was the first practical boosting algorithm, and remains one of the most widely used and studied, with applications in numerous fields. This chapter aims to review some of the many perspectives and analyses of AdaBoost that have been applied to explain or understand it as a learning method, with comparisons of both the strengths and weaknesses of the various approaches.

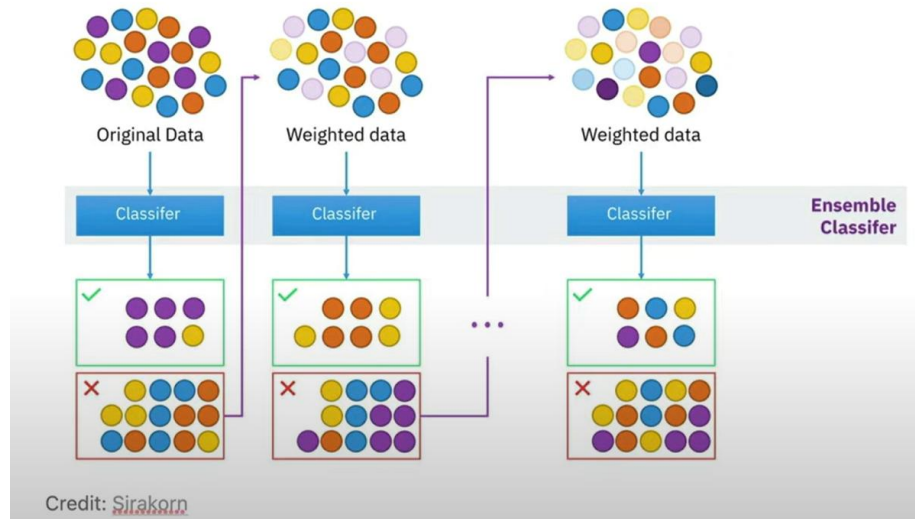
Abstract. We consider the problem of dynamically apportioning resources among a set of options in a worst-case on-line framework. The model we study can be interpreted as a broad, abstract extension of the well-studied on-line prediction model to a general decision-theoretic setting. We show that the multiplicative weight-update rule of Littlestone and Warmuth [10] can be adapted to this model yielding bounds that are slightly weaker in some cases, but applicable to a considerably more general class of learning problems. We show how the resulting learning algorithm can be applied to a variety of problems, including gambling, multiple-outcome prediction, repeated games and prediction of points in \mathbb{R}^n . We also show how the weight-update rule can be used to derive a new boosting algorithm which does not require prior knowledge about the performance of the weak learning algorithm.

2003: Gödel Prize



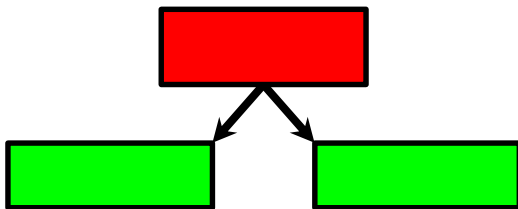
AdaBoost: Idea

- In each iteration AdaBoost identifies misclassified datapoints.
- AdaBoost **increases the relative weights of misclassified datapoints** (and thereby decreases the relative weights of correctly classified datapoints).
- Result: The next classifier will “pay more attention” to the datapoints that the ensemble can not yet classify correctly.



AdaBoost: Method

- Base learners are often decision trees (depth 1 for classification). These are called **stumps**. But any other base learners can be used.
- Stumps are usually very weak learners (because they only make one split). But a properly constructed ensemble of stumps can be a strong learner.



Documentation

Parameters:

estimator : object, default=None

The base estimator from which the boosted ensemble is built. Support for sample weighting is required, as well as proper `classes_` and `n_classes_` attributes. If `None`, then the base estimator is `DecisionTreeClassifier` initialized with `max_depth=1`.

! Added in version 1.2: `base_estimator` was renamed to `estimator`.

n_estimators : int, default=50

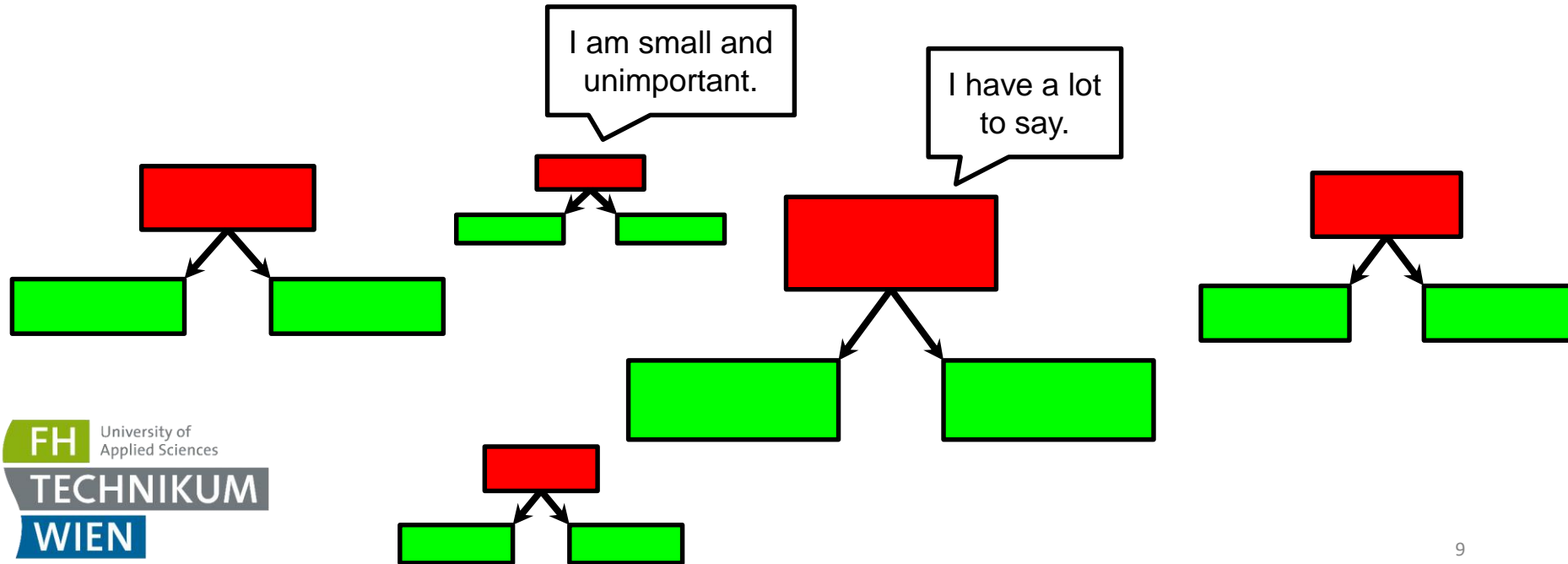
The maximum number of estimators at which boosting is terminated. In case of perfect fit, the learning procedure is stopped early. Values must be in the range `[1, inf)`.

learning_rate : float, default=1.0

Weight applied to each classifier at each boosting iteration. A higher learning rate increases the contribution of each classifier. There is a trade-off between the `learning_rate` and `n_estimators` parameters. Values must be in the range `(0.0, inf)`.

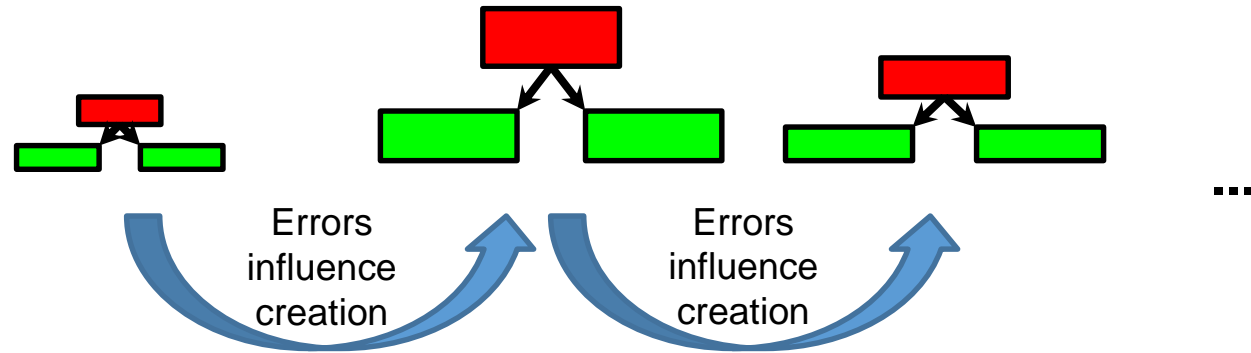
AdaBoost: Method

- In classical plurality-voting ensembles every learner has equal impact.
- In AdaBoost learners have different impacts on the final prediction (since the learners have different weights).



AdaBoost: Method

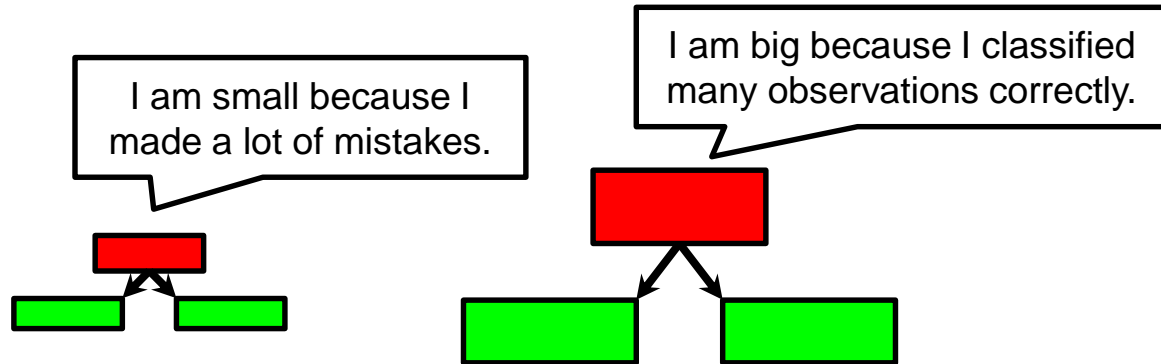
- In a classical **random forest** each **tree** is made **independently**.
- In AdaBoost the previous stump (more precisely its mistakes) influences how the next stump is created (by influencing the weights of its input sample).



AdaBoost: Method

- 1) Initially each observation in the sample gets the same weight and weights sum to 1 (uniform distribution).
- 2) We decide which feature to use for the **first stump** based on the **lowest Gini index** (or cross-entropy).
- 3) We **calculate** the amount of **influence** of a stump on the final decision. It depends on the fraction of observations correctly classified. Stumps which are worse than random will have an influence of 0 while stumps that classify well will have much influence.
- 4) Formula:

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



AdaBoost: Method

4) We **reweight the observations**:

If our first stump misclassified an observation we increase the weight of this misclassified observation:

weight <- *weight* * *exp(influence)*



At the same time we decrease the weights of the correctly classified observations:

weight <- *weight* * *exp(- influence)*



5) We **rescale** the new weights such that they **sum up to 1**.

AdaBoost: Method

6) We use the **sample weights** to **make the second tree stump** by one of the two following methods:

- a) We could **calculate a weighted Gini index** (or cross-entropy) such that observations with a higher weight count more in the calculation of the Gini index.
- b) We **bootstrap** (randomly draw observations from the sample according to their weights, the **probability** of an observation to be chosen equals its **weight**). This means that observations where we failed before are more likely to appear in the bootstrapped dataset.

id	Features ...	Weight
1	...	0.2
2	...	0.4
3	...	0.2
4	...	0.2

Weighted sample

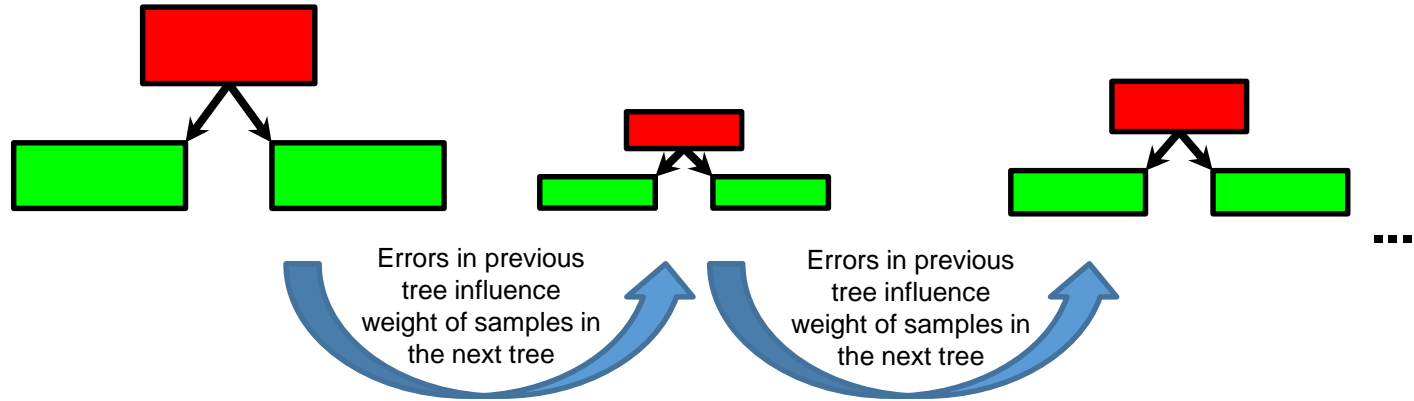
Weighted bootstrapping



id	Features ...	Weight
2	...	
4	...	
1	...	
2	...	

AdaBoost: Method

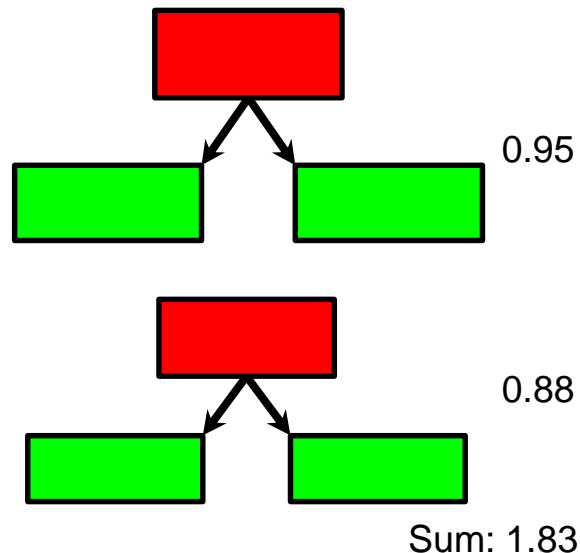
7) We repeat the whole procedure.



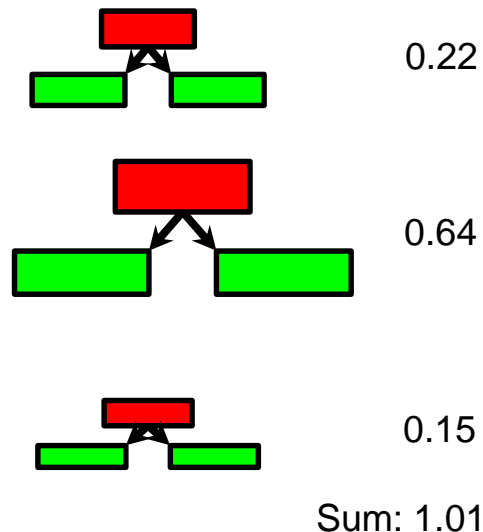
AdaBoost: Method

Prediction by weighted voting (weighted by stump influence).

Trees prediction **YES**

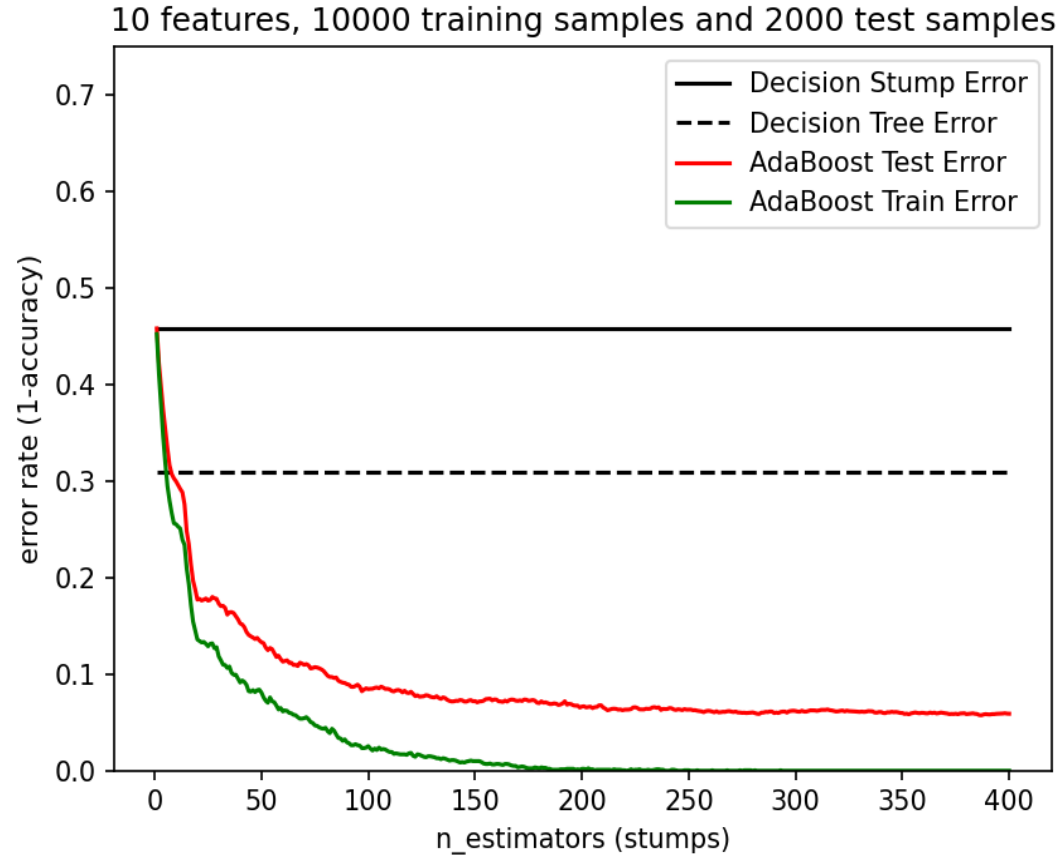


Trees predicting **NO**



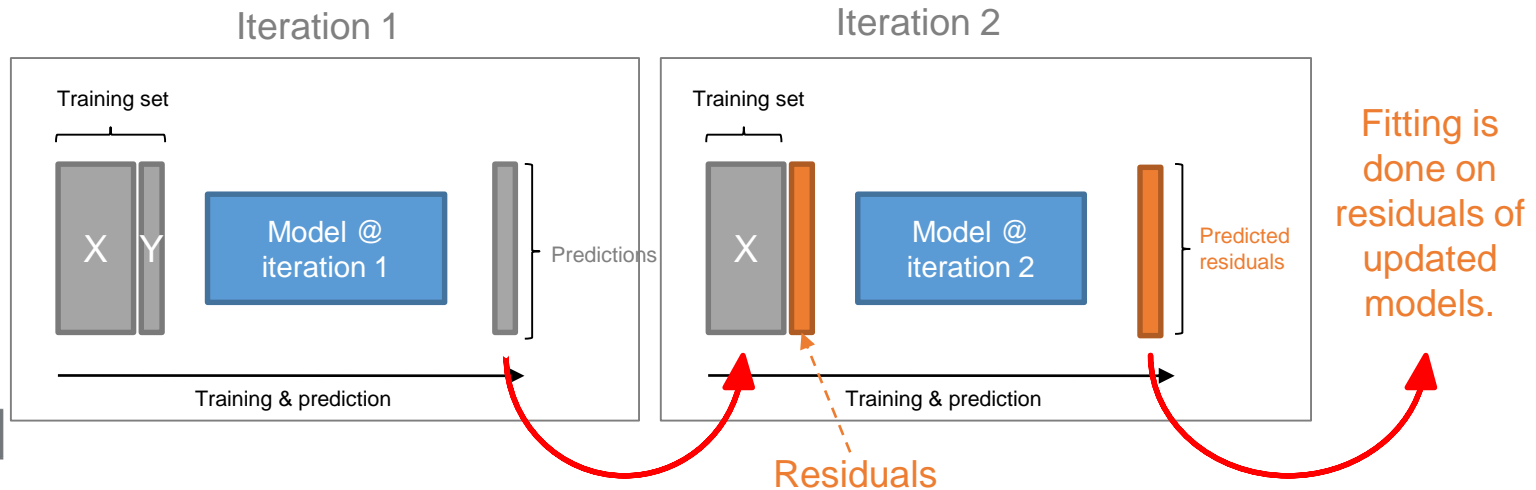
$1.83 > 1.01 \Rightarrow \text{Prediction} = \text{YES}$

AdaBoost: Performance



Gradient Boosting: Basic Idea

- Same motivation as AdaBoost: **Focus the ensemble on difficult cases.**
- But **gradient boosting does not use reweighting** to achieve this.
- Gradient boosting **fits models to the errors (residuals) of the previous models.** Therefore observations leading to large errors immediately become more important.
- The **base learners are regressions.** (There is also a version for classification. We do not cover it in this course.)

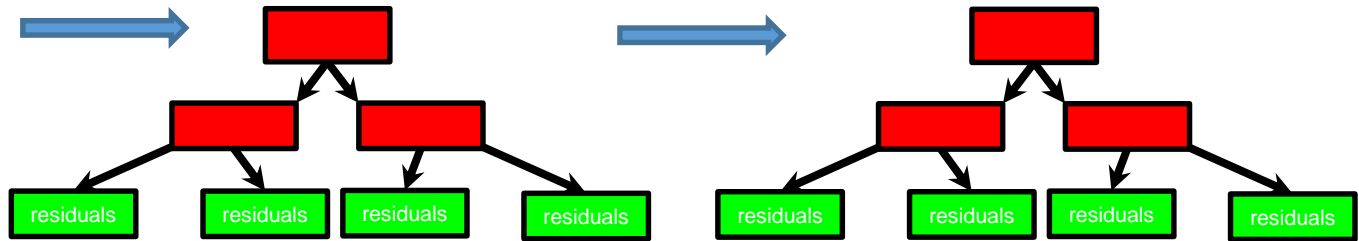


Gradient Boosting: Method

- Gradient boosting starts with a **single leaf** which predicts the **mean value** of the target variable in the training set. This is a rather naive empirical estimate.
- Gradient boosting then **grows** small **trees** (but not stumps) **using the errors** (residuals) of the **previous estimations** as new target variables (instead of the original target variable).
- An example is given in the next slides.

42 508

Single leaf that just predicts the mean value.



Gradient Boosting: Method


Step 1: Predict the mean and calculate the residuals.

Average Weight

71.2

Height (m)	Favorite Color	Gender	Weight (kg)	Residual
1.6	Blue	Male	88	16.8
1.6	Green	Female	76	4.8
1.5	Blue	Female	56	-15.2
1.8	Red	Male	73	1.8
1.5	Green	Male	77	5.8
1.4	Blue	Female	57	-14.2

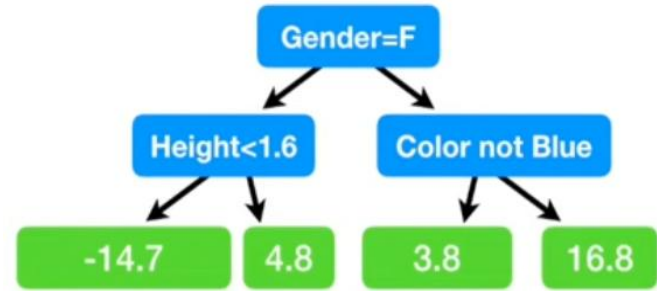
Now do the same thing
for the remaining
Weights...


$$(57 - 71.2) = -14.2$$

Gradient Boosting: Method

Step 2: Built a new tree to predict the residuals.

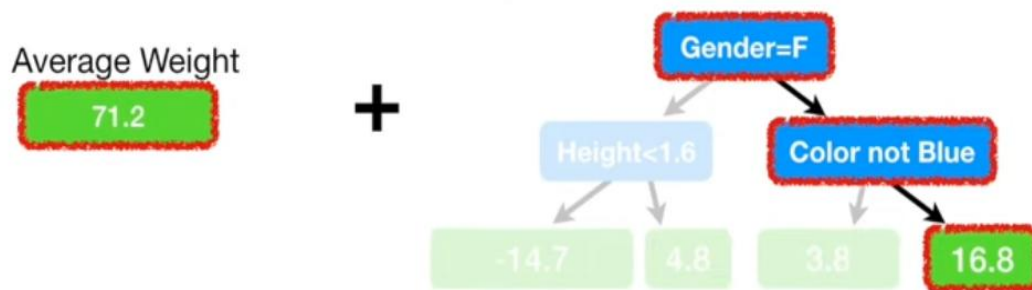
Height (m)	Favorite Color	Gender	Weight (kg)	Residual
1.6	Blue	Male	88	16.8
1.6	Green	Female	76	4.8
1.5	Blue	Female	56	-15.2
1.8	Red	Male	73	1.8
1.5	Green	Male	77	5.8
1.4	Blue	Female	57	-14.2



If you do not remember the algorithm for decision tree construction, review the respective slides.

Gradient Boosting: Method

Step 3: The new prediction is obtained by adding the tree result to the last prediction (the mean).



$$\text{Predicted Weight} = 71.2 + 16.8 = 88$$

Now the prediction exactly equals the observation on the training data.

The model has been overfit: Very low bias.

Gradient Boosting: Method

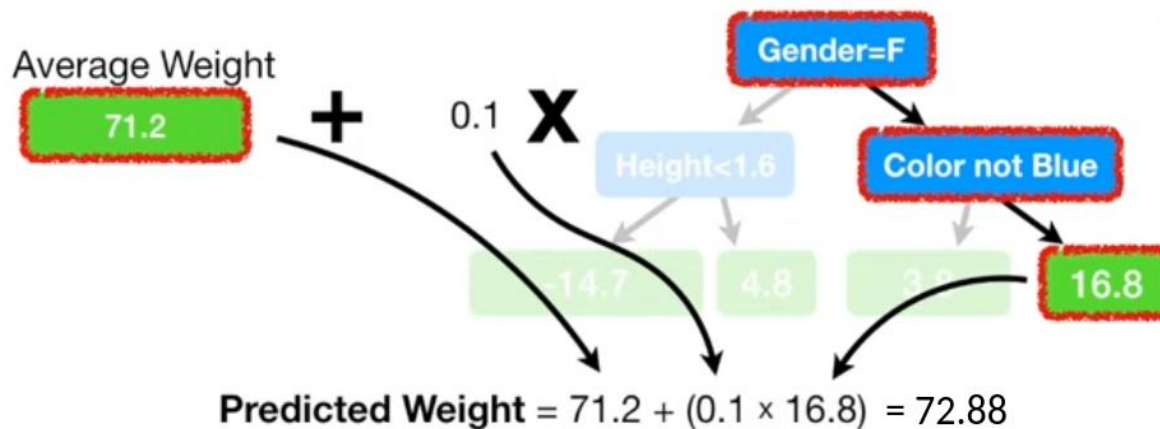
We use a learning rate to reduce the danger of overfitting.



Gradient Boost deals with this problem by using a **Learning Rate** to scale the contribution from the new tree.

The **Learning Rate** is a value between **0** and **1**.

Gradient Boosting: Method



This prediction is a little better than the mean prediction.

Using a small learning rate, we iteratively improve the prediction by adding more trees.

At every iteration we make a small step in the right direction.

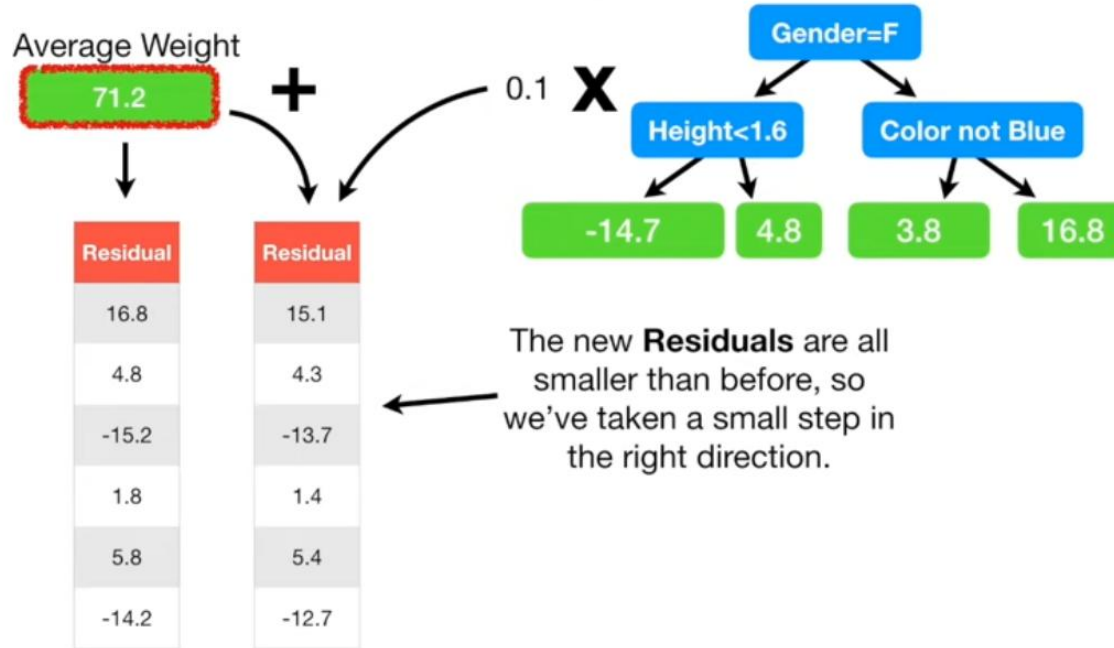
Empirical evidence shows that a lot of small steps in the right direction result in better generalization (performance on new test data), since variance is reduced.

Gradient Boosting: Method

Step 4: We calculate new residuals with respect to the new predictions.

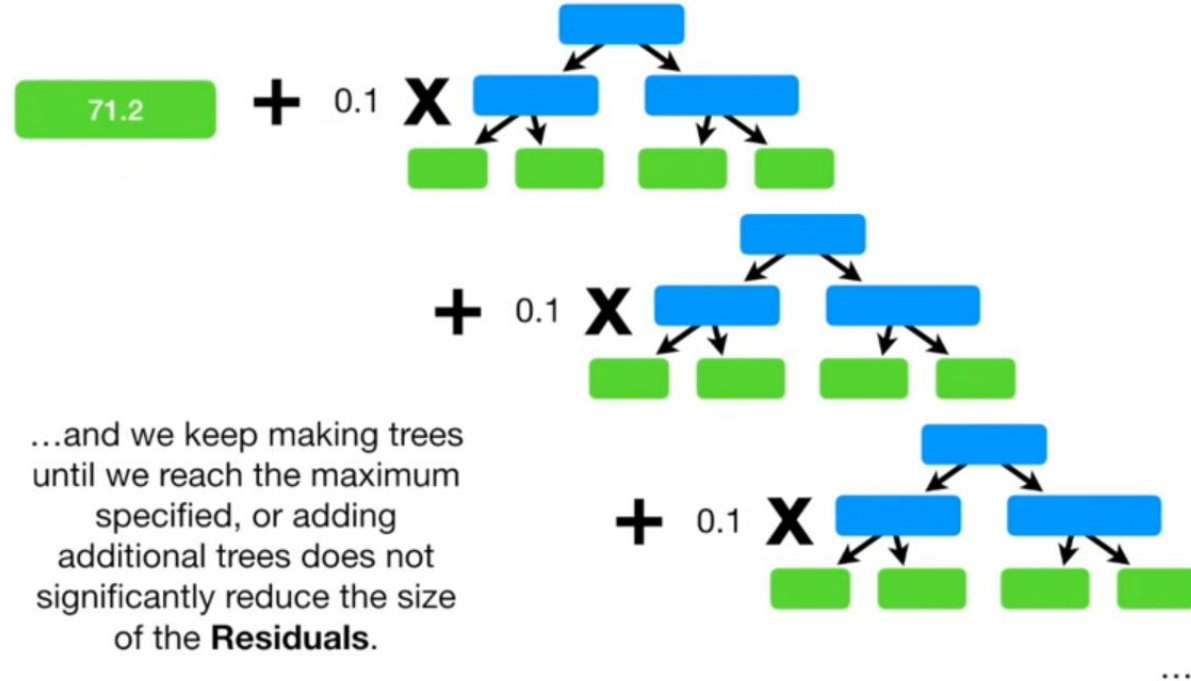


Gradient Boosting: Method



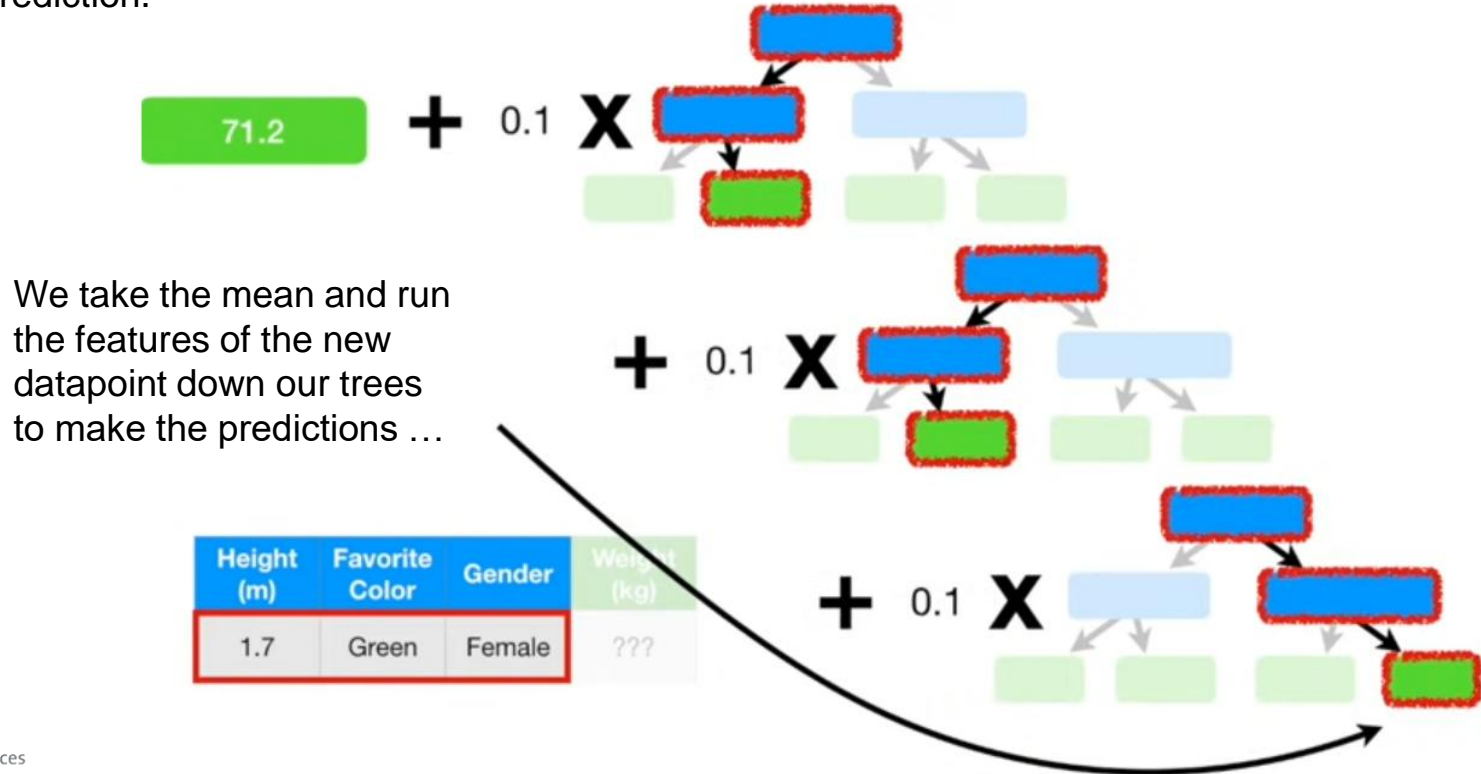
Gradient Boosting: Method

Then: Repeat as often as desired.



Gradient Boosting: Method

Application: Prediction.



Extreme Gradient boosting: XGBoost

- A highly **scalable** and **well-performing** variation of gradient boosting.
- XGBoost differs from gradient boosting in several ways:
 - Residual-similarity-based **XGBoost-trees**.
 - **Regularization** parameter in tree-building controls tree **pruning**.
- Can be used for regression and classification.
- Not covered in this course.

Scikit-Learn API

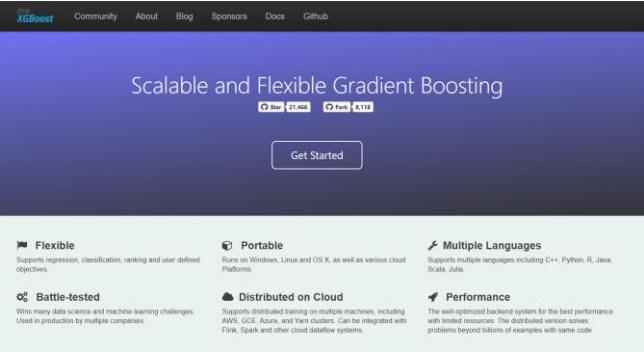
Scikit-Learn Wrapper interface for XGBoost.

```
class xgboost.XGBRegressor(object, booster='gbtree', **kwargs)
```

Bases: `xgboost.sklearn.XGBModel`, `object`

Implementation of the scikit-learn API for XGBoost regression.

- Parameters:
- **n_estimators** (`int`) – Number of gradient boosted trees. Equivalent to number of boosting rounds.
 - **max_depth** (`Optional[int]`) – Maximum tree depth for base learners.
 - **learning_rate** (`Optional[float]`) – Boosting learning rate (xgb's "eta")
 - **verbosity** (`Optional[int]`) – The degree of verbosity. Valid values are 0 (silent) - 3 (debug).
 - **objective** (`typing.Union[str, typing.Callable[[numpy.ndarray, numpy.ndarray], typing.Tuple[numpy.ndarray, numpy.ndarray]], NoneType]`) – Specify the learning task and the corresponding learning objective or a custom objective function to be used (see note below).
 - **booster** (`Optional[str]`) – Specify which booster to use: `gbtree`, `gblinear` or `dart`.
 - **tree_method** (`Optional[str]`) – Specify which tree method to use. Default to



The screenshot shows the XGBoost website with a dark blue header and a light blue main content area. The header includes navigation links: XGBoost, Community, About, Blog, Sponsors, Docs, and Github. The main content area features the title "Scalable and Flexible Gradient Boosting" with download statistics (21,486 stars, 3,118 forks) and a "Get Started" button. Below this, there are four feature sections: Flexible (Supports regression, classification, ranking and user defined objectives), Portable (Runs on Windows, Linux and OS X, as well as various cloud Platforms), Distributed on Cloud (Supports distributed training on multiple machines, including AWS, GCE, Azure, and Yarn clusters. Can be integrated with Hadoop, Spark and other cloud dataflow systems), and Multiple Languages (Supports multiple languages including C++, Python, R, Java, Scala, Julia). A "Performance" section at the bottom right mentions a well-optimized backend system for the best performance with limited resources.

Boosting: Pros & Cons

- Boosting allow for the construction of a strong learner from many weak learners.
- Boosting **combines models iteratively such that the next model depends on the errors of the previous model.**
- Boosting algorithms perform very well, especially on traditional tabular data.
- Boosting is a **sequential ensemble approach**. Therefore it **can not be parallelized** and scaling is not as easy as in bagging.
- Boosting **increases training time (by a lot) and test time (by a bit).**
- Ensemble methods turn interpretable base learners into **black box models.**

Takeaway

- Boosting is an ensemble methodology that builds **a strong learner from many weak weak learners, by training models sequentially.**
- The mathematical analysis of boosting procedures can be quite difficult, but their idea and their implementation is straightforward.
- **AdaBoost:** works by reweighting misclassified samples
- **Gradient boosting:** each learner fits the residuals (gradients of loss) of the previous ensemble; uses a loss function for learning (gradient descent)
- **XGBoost:** advanced gradient boosting with regularization and other optimizations.

Next up: DBScan

Assignment: Boosting

a) Explain AdaBoost and gradient boosting using 1 page/slide each.

Use self-made images or even hand drawings (of which you take a photo).

Use self-written explanations.

Do not copy from the lecture slides or the internet (neither text nor images).

b) Implement AdaBoost.

Implement the simplest version of AdaBoost as described in the lecture.

Run it on test data of your choice.

Use a library for decision trees and random forests on the same dataset.

Is your AdaBoost performing better?

Online Resources

- **AdaBoost Clearly Explained**
<https://www.youtube.com/watch?v=LsK-xG1cLYA>
- **Gradient Boost: Regression Main Ideas**
<https://www.youtube.com/watch?v=3CC4N4z3GJc>
- **Gradient Boost Part: Classification**
<https://www.youtube.com/watch?v=jxuNLH5dXCs>
- **XGBoost Part 1 (of 4): Regression**
<https://www.youtube.com/watch?v=OtD8wVaFm6E>
- **XGBoost Part 2 (of 4): Classification**
<https://www.youtube.com/watch?v=8b1JEDvenQU>

References

- Géron A. (2017): Hands-On Machine Learning with Scikit-Learn & Tensorflow. – O'Reilly.
- James G., Witten D., Hastie T., Tibshirani R. (2017): An introduction to Statistical Learning. – Springer.
- Kuhn M., Johnson K. (2016): Applied Predictive Modeling. – Springer.
- Berk R. (2016): Statistical Learning from a Regression Perspective. – Springer.
- Freund Y, Schapire R (1996): „Experiments with a new Boosting Algorithm.“ Machine Learning: Proceedings of the Thirteenth International Conference, pp. 148-156
- Friedman J (2002): „Stochastic Gradient Boosting.“ Computational Statistics and Data Analysis, 38(4), 367-378
- Friedman J, Hastie T, Tibshirani R (2000): „Additive Logistic Regression: A Statistical View of Boosting.“ Annals of Statistics, 38, 337-374