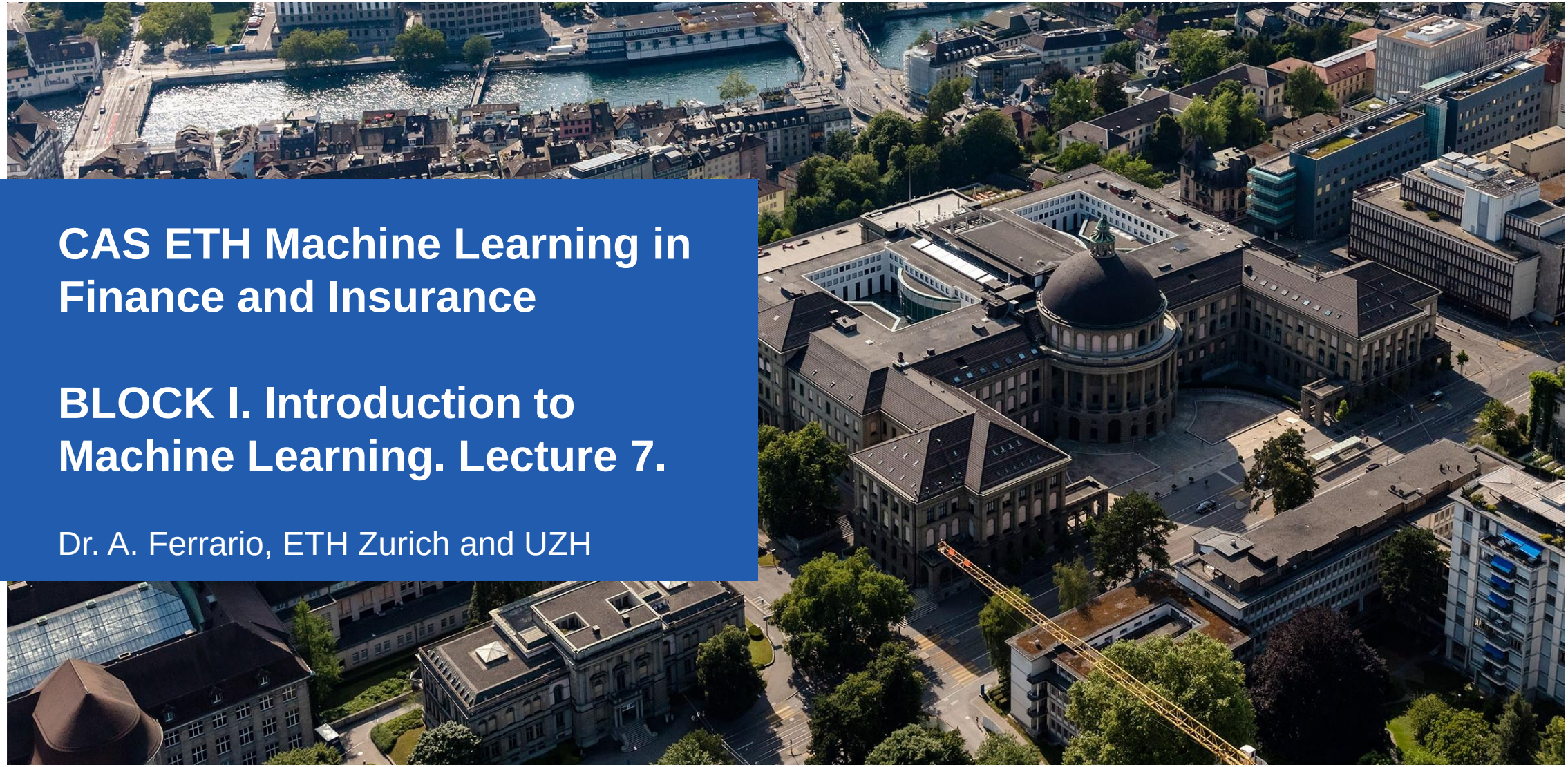
Department of Mathematics
ETH Zurich



CAS ETH Machine Learning in Finance and Insurance

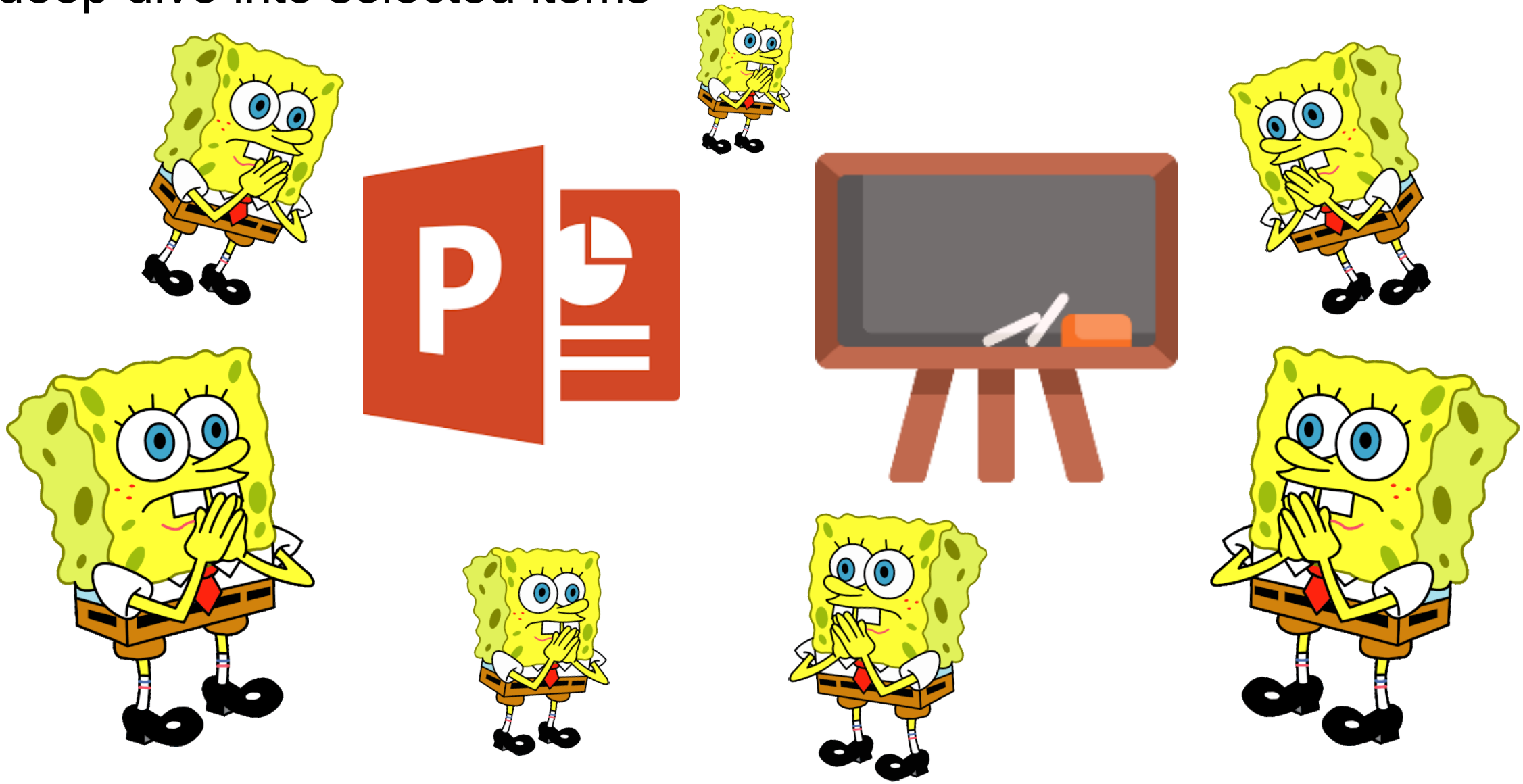
BLOCK I. Introduction to Machine Learning. Lecture 7.

Dr. A. Ferrario, ETH Zurich and UZH

On May 17th…

1	We introduced the decision trees, namely, an “interpretable” method to learn (binary) decisions from data
2	We discussed the famous Classification And Regression Tree algorithm by Breiman et al. (1984) in some detail and learned how to prune trees
3	We showed that decision trees are prone to overfitting and show high variance

We will use slides to introduce our topics and the blackboard to deep-dive into selected items



Machine Learning Methods (Part 7)

- Bagging and Random Forests
- Boosting

Decision trees are easy to train and “interpretable”. However…

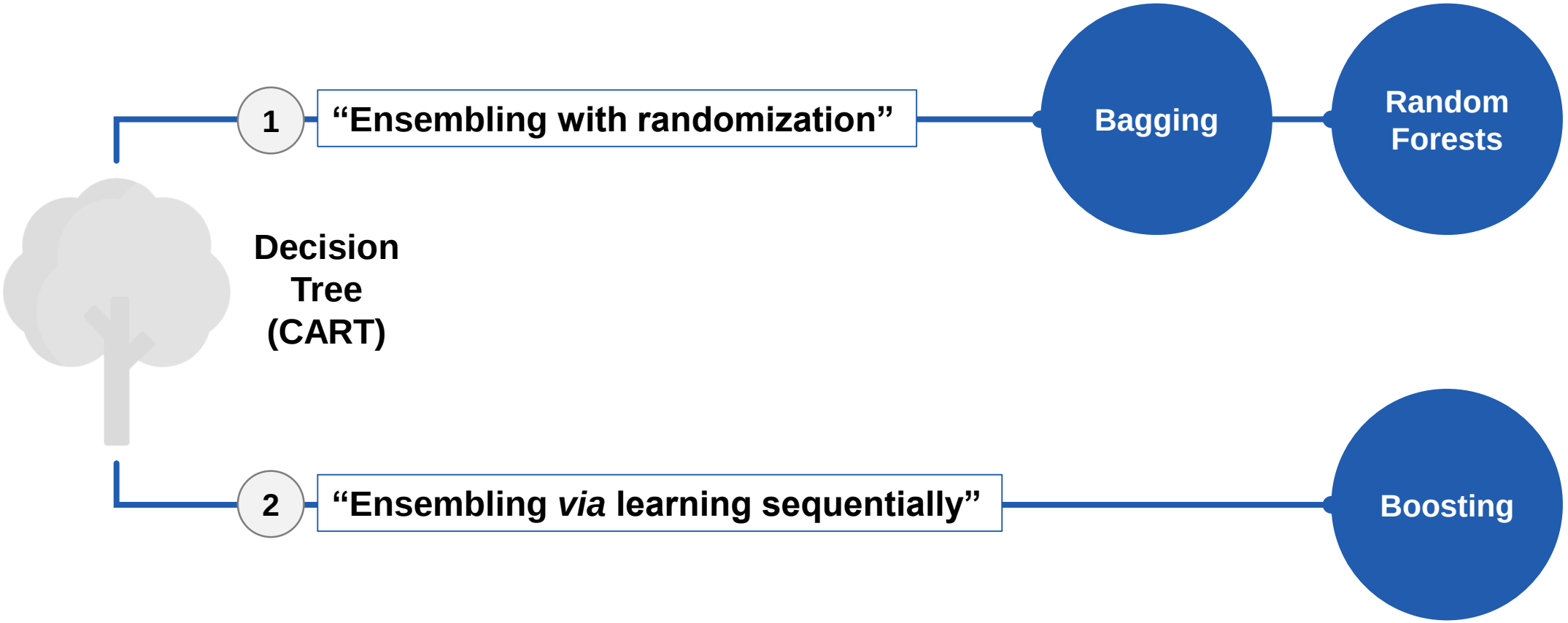
…they are prone to overfitting (large trees are needed to capture complex, nonlinear decision boundaries) and **tend to show high variance**

IDEA: “Wisdom of the crowd”

If we **appropriately combine many trees**, we may end up with a better model


Ensembling

The plan for today: Ensembling



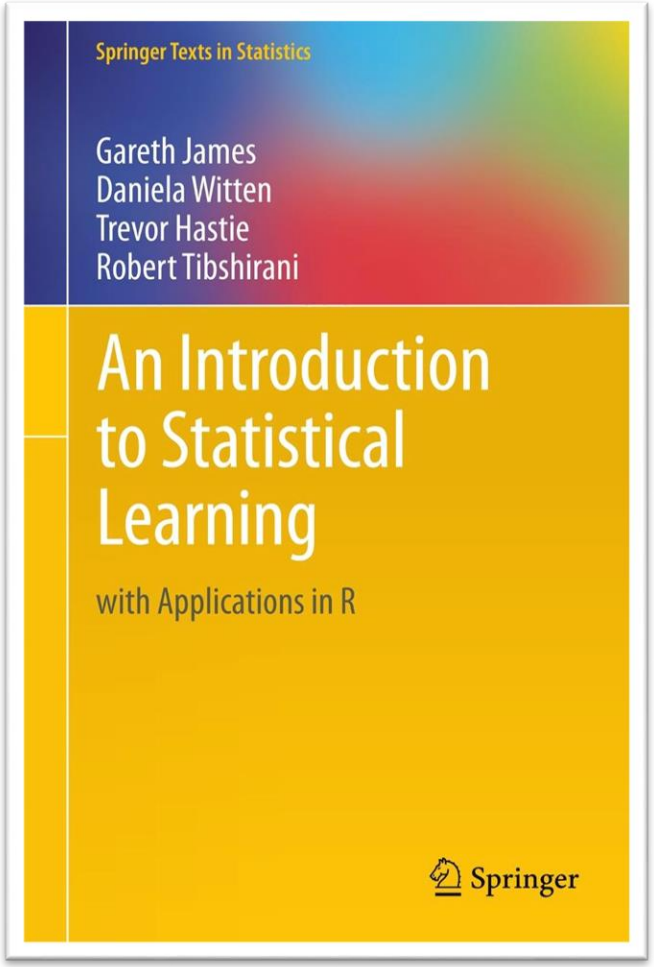
“Ensembling with randomization”

Bagging and Random Forests

Bagging = **B**oostrap **agg**regat**ing**

A simple method to reduce the variance of decision trees

"[Bagging] is a general-purpose procedure for reducing the variance of a statistical learning method; we introduce it here because it is particularly useful and frequently used in the context of decision trees" (pag. 316)



James, G., Witten, D., Hastie, T., & Tibshirani, R. (2013). An introduction to statistical learning. New York: Springer

Bagging = **B**oostrap **agg**regat**ing**

Bagging consists of two steps

We ***bootstrap the original training set*** as generally do not have access to multiple training datasets.

I We generate *B* bootstrapped training datasets, **by randomly sampling with replacement** *B* times **from the original training data** set.

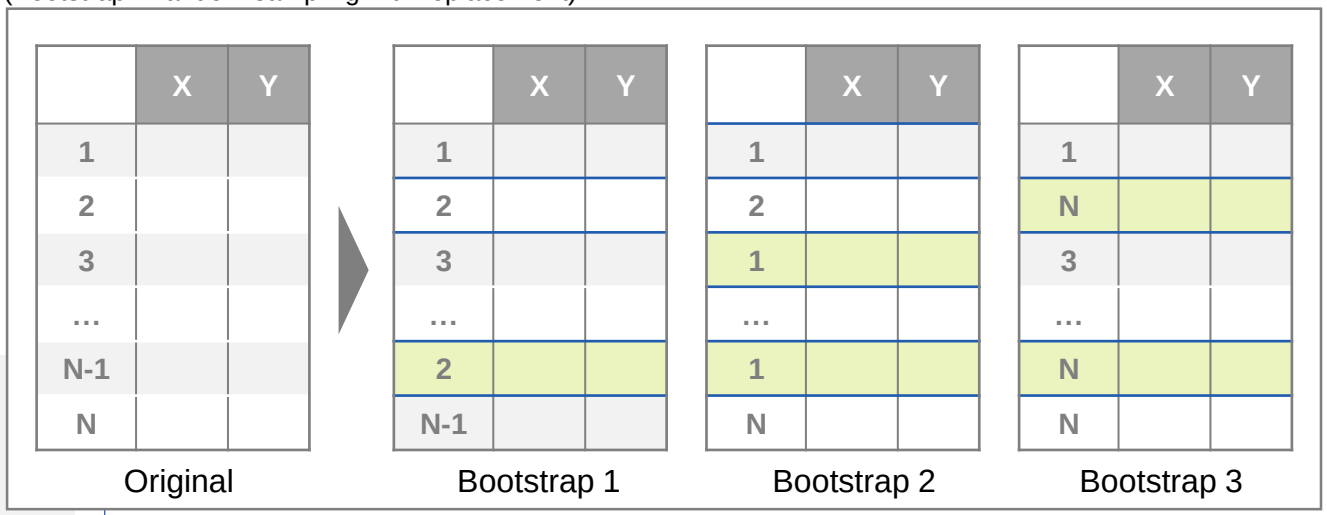
II We train *B* decision trees on the *B* bootstrapped training datasets and we **aggregate their predictions.** The *B* trees are grown deep and are not pruned (high variance).

Formulae for regression and classification

Bagging = **B**oostrap **agg**regat**ing**

Bagging consists of two steps

(Bootstrap = random sampling with replacement)



We ***bootstrap the original training set*** as generally do not have access to multiple training datasets.

I We generate *B* bootstrapped training datasets, **by randomly sampling with replacement** *B* times **from the original training data** set.

II We train *B* decision trees on the *B* bootstrapped training datasets and we **aggregate their predictions.** The *B* trees are grown deep and are not pruned (high variance).

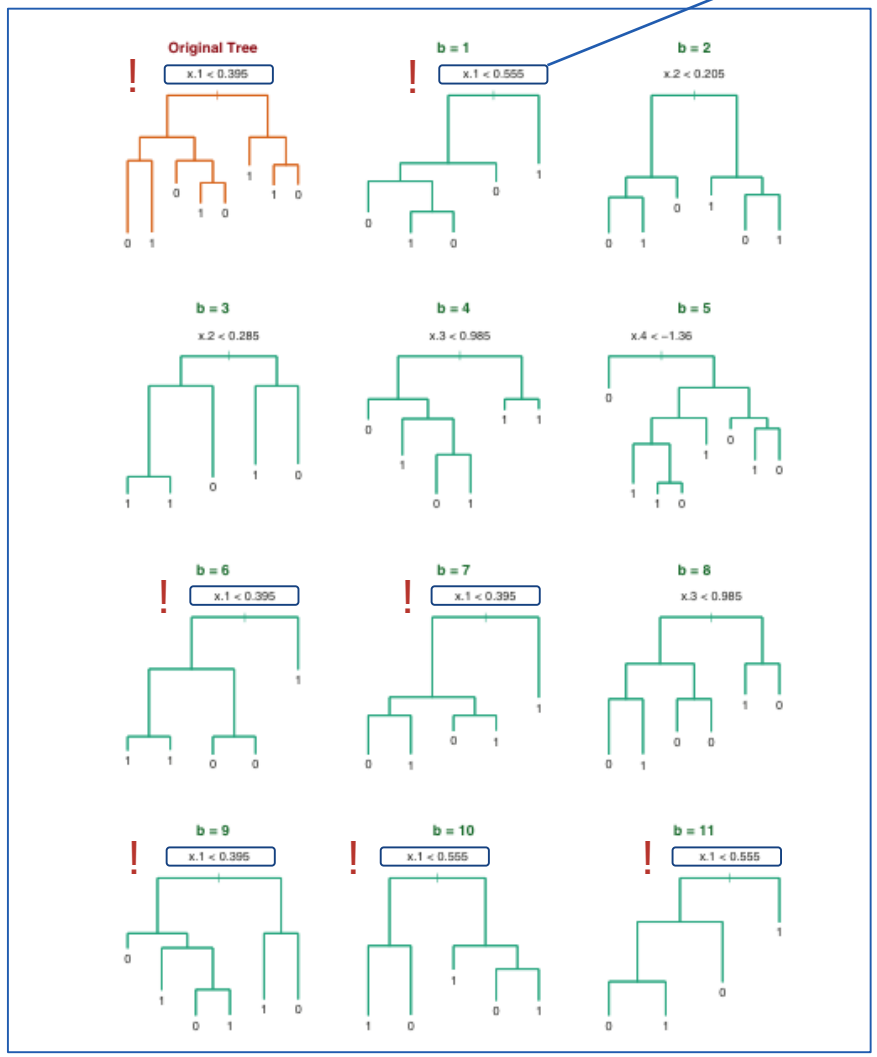
Formulae for regression and classification

Bagging = **B**oostrap **agg**regat**ing**

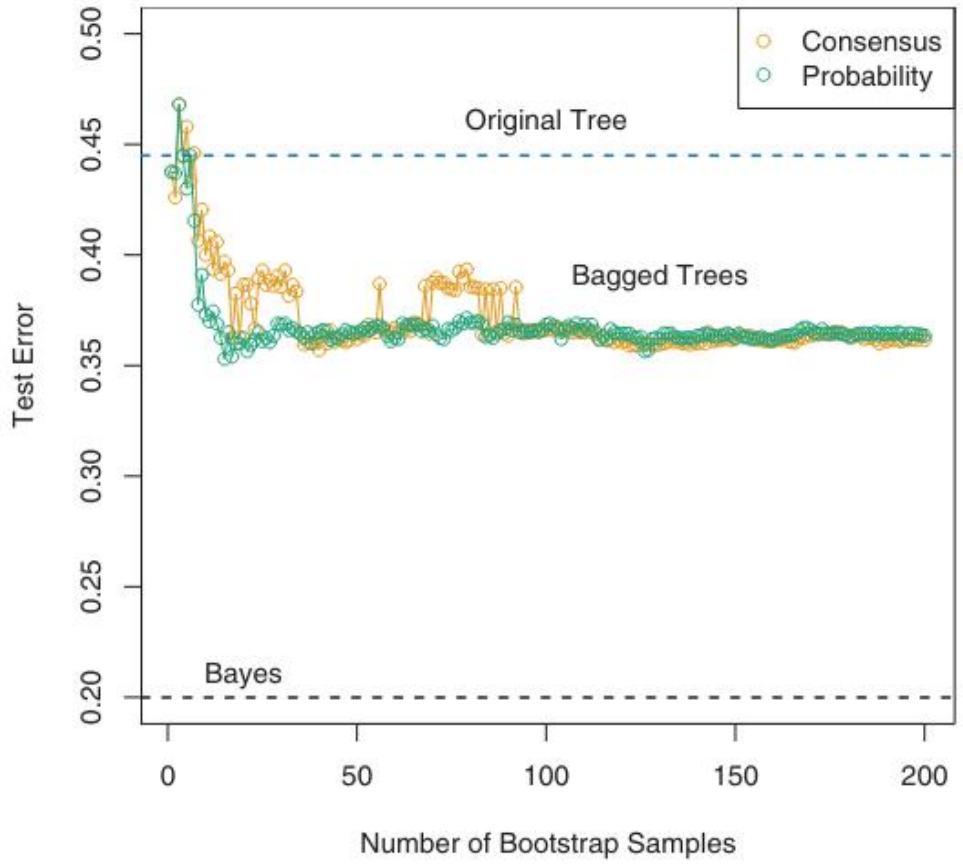
Bagging consists of two steps

Hey! 6 out of 11 bagged tree have a root using **x1**…

Source: Figure 8.9 in Hastie et al. (2009)*



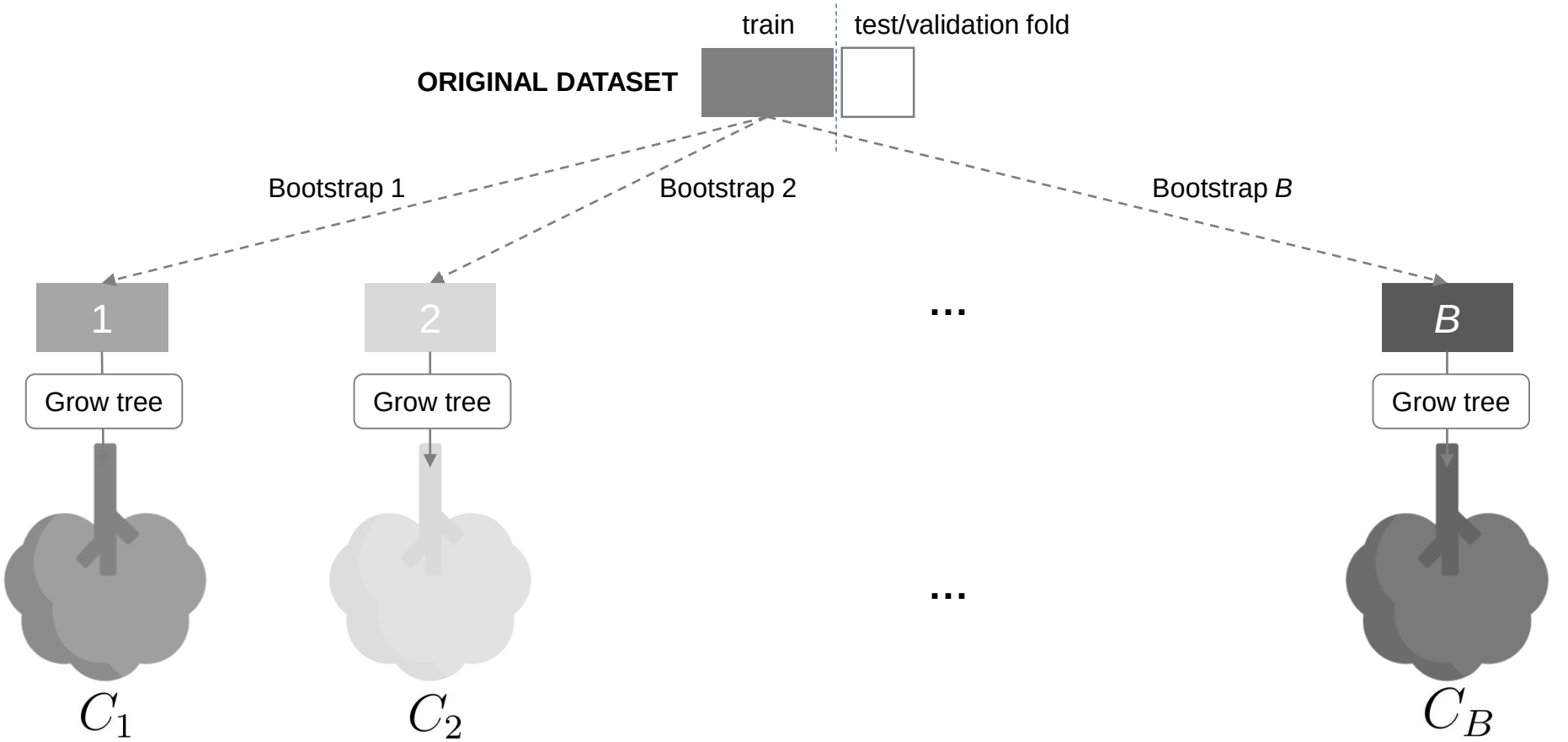
Source: Figure 8.10 in Hastie et al. (2009)



*Hastie, T., Tibshirani, R., Friedman, J. H., & Friedman, J. H. (2009). *The elements of statistical learning: data mining, inference, and prediction* (Vol. 2, pp. 1-758). New York: Springer.

Bagging: A visual summary

Choose how many bootstrapped training datasets: *B*



$$C_{\text{bagging}}(x) = \frac{1}{B}\sum_{i=1}^{B} C_i(x) \qquad \text{(regression)}$$

Bagging: some results from Breiman's original article
Classification trees on medium-sized datasets (in 1996)

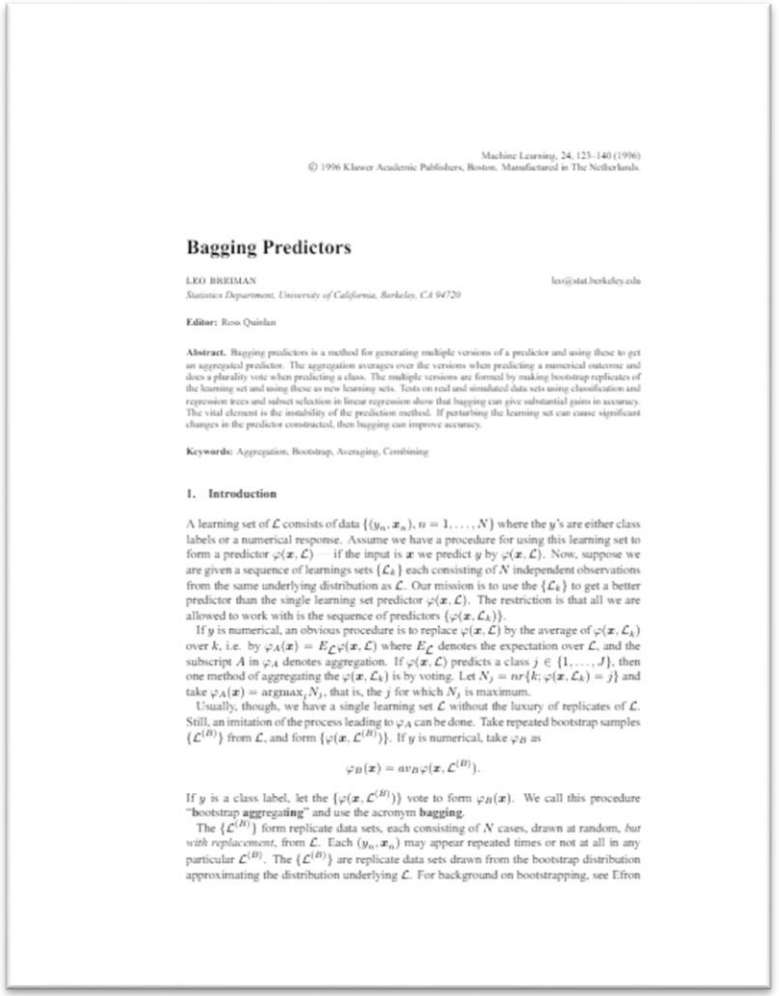
Data Set	# Samples	# Variables	# Classes
waveform	300	21	3
heart	1395	16	2
breast cancer	699	9	2
ionosphere	351	34	2
diabetes	768	8	2
glass	214	9	6
soybean	683	35	19

Table 2. Misclassification Rates (%)

Data Set	$\bar{e}_S$	$\bar{e}_B$	Decrease
waveform	29.1	19.3	34%
heart	4.9	2.8	43%
breast cancer	5.9	3.7	37%
ionosphere	11.2	7.9	29%
diabetes	25.3	23.9	6%
glass	30.4	23.6	22%
soybean	8.6	6.8	21%

Error classification tree

Error bagging ensemble



Machine Learning, 24, 123–140 (1996)
© 1996 Kluwer Academic Publishers, Boston. Manufactured in The Netherlands.

Bagging Predictors

LEO BREIMAN leo@stat.berkeley.edu
Statistics Department, University of California, Berkeley, CA 94720

Editor: Ross Quinlan

Abstract. Bagging predictors is a method for generating multiple versions of a predictor and using these to get an aggregated predictor. The aggregation averages over the versions when predicting a numerical outcome and does a plurality vote when predicting a class. The multiple versions are formed by making bootstrap replicates of the learning set and using these as new learning sets. Tests on real and simulated data sets using classification and regression trees and subset selection in linear regression show that bagging can give substantial gains in accuracy. The vital element is the instability of the prediction method. If perturbing the learning set can cause significant changes in the predictor constructed, then bagging can improve accuracy.

Keywords: Aggregation, Bootstrap, Averaging, Combining

1. Introduction

A learning set of $\mathcal{L}$ consists of data $\{(y_n, \boldsymbol{x}_n), n = 1, \ldots, N\}$ where the y's are either class labels or a numerical response. Assume we have a procedure for using this learning set to form a predictor $\varphi(\boldsymbol{x}, \mathcal{L})$ — if the input is $\boldsymbol{x}$ we predict y by $\varphi(\boldsymbol{x}, \mathcal{L})$. Now, suppose we are given a sequence of learnings sets $\{\mathcal{L}_k\}$ each consisting of N independent observations from the same underlying distribution as $\mathcal{L}$. Our mission is to use the $\{\mathcal{L}_k\}$ to get a better predictor than the single learning set predictor $\varphi(\boldsymbol{x}, \mathcal{L})$. The restriction is that all we are allowed to work with is the sequence of predictors $\{\varphi(\boldsymbol{x}, \mathcal{L}_k)\}$.

If y is numerical, an obvious procedure is to replace $\varphi(\boldsymbol{x}, \mathcal{L})$ by the average of $\varphi(\boldsymbol{x}, \mathcal{L}_k)$ over k, i.e. by $\varphi_A(\boldsymbol{x}) = E_{\mathcal{L}}\varphi(\boldsymbol{x}, \mathcal{L})$ where $E_{\mathcal{L}}$ denotes the expectation over $\mathcal{L}$, and the subscript A in φ_A denotes aggregation. If $\varphi(\boldsymbol{x}, \mathcal{L})$ predicts a class $j \in \{1, \ldots, J\}$, then one method of aggregating the $\varphi(\boldsymbol{x}, \mathcal{L}_k)$ is by voting. Let $N_j = nr\{k; \varphi(\boldsymbol{x}, \mathcal{L}_k) = j\}$ and take $\varphi_A(\boldsymbol{x}) = \text{argmax}_j N_j$, that is, the j for which N_j is maximum.

Usually, though, we have a single learning set $\mathcal{L}$ without the luxury of replicates of $\mathcal{L}$. Still, an imitation of the process leading to φ_A can be done. Take repeated bootstrap samples $\{\mathcal{L}^{(B)}\}$ from $\mathcal{L}$, and form $\{\varphi(\boldsymbol{x}, \mathcal{L}^{(B)})\}$. If y is numerical, take φ_B as

$$\varphi_B(\boldsymbol{x}) = av_B\varphi(\boldsymbol{x}, \mathcal{L}^{(B)}).$$

If y is a class label, let the $\{\varphi(\boldsymbol{x}, \mathcal{L}^{(B)})\}$ vote to form $\varphi_B(\boldsymbol{x})$. We call this procedure "bootstrap **aggregating**" and use the acronym **bagging**.

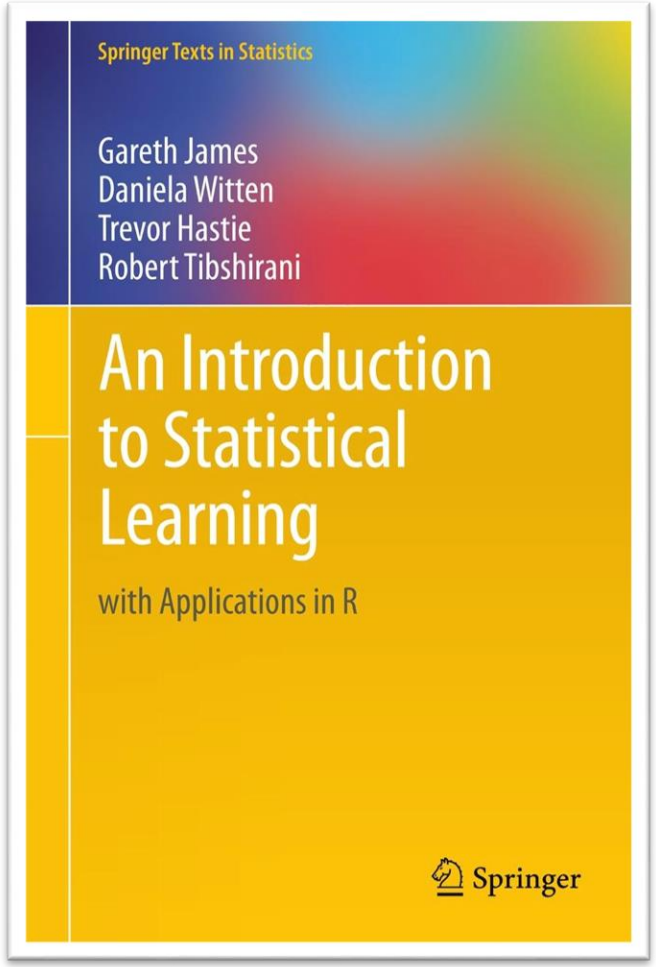
The $\{\mathcal{L}^{(B)}\}$ form replicate data sets, each consisting of N cases, drawn at random, *but with replacement*, from $\mathcal{L}$. Each $(y_n, \boldsymbol{x}_n)$ may appear repeated times or not at all in any particular $\mathcal{L}^{(B)}$. The $\{\mathcal{L}^{(B)}\}$ are replicate data sets drawn from the bootstrap distribution approximating the distribution underlying $\mathcal{L}$. For background on bootstrapping, see Efron

Breiman, L. (1996). Bagging predictors. *Machine learning*, *24*, 123-140.

An important limitation in bagging

The ensembles of trees in bagging tend to be highly correlated

"Suppose there is one very strong predictor in the data set, along with a number of other moderately strong predictors. Then in a collection of bagged trees, most or all of the trees will use this strong predictor in the top split. Consequently, all of the bagged trees will look quite similar to each other. Hence the predictions from the bagged trees will be highly correlated. Unfortunately, averaging many highly correlated quantities does not lead to as large of a reduction in variance as averaging many uncorrelated quantities" (pag. 320)



James, G., Witten, D., Hastie, T., & Tibshirani, R. (2013). An introduction to statistical learning. New York: Springer

A solution: The random forests algorithm
Yet another breakthrough result from Leo Breiman

"Random forests are a combination of tree predictors such that each tree depends on the values of a random vector sampled independently and with the same distribution for all trees in the forest" (pag. 5, Breiman 2001)

"The forests studied here consist of using randomly selected inputs or combinations of inputs at each node to grow each tree" (pag. 10, Breiman 2001)

Machine Learning, 45, 5–32, 2001

Random Forests

LEO BREIMAN
Statistics Department, University of California, Berkeley, CA 94720

Editor: Robert E. Schapire

Abstract. Random forests are a combination of tree predictors such that each tree depends on the values of a random vector sampled independently and with the same distribution for all trees in the forest. The generalization error for forests converges a.s. to a limit as the number of trees in the forest becomes large. The generalization error of a forest of tree classifiers depends on the strength of the individual trees in the forest and the correlation between them. Using a random selection of features to split each node yields error rates that compare favorably to Adaboost (Y. Freund & R. Schapire, *Machine Learning: Proceedings of the Thirteenth International conference*, ∗ ∗ ∗, 148–156), but are more robust with respect to noise. Internal estimates monitor error, strength, and correlation and these are used to show the response to increasing the number of features used in the splitting. Internal estimates are also used to measure variable importance. These ideas are also applicable to regression.

Keywords: classification, regression, ensemble

1. Random forests

1.1. Introduction

Significant improvements in classification accuracy have resulted from growing an ensemble of trees and letting them vote for the most popular class. In order to grow these ensembles, often random vectors are generated that govern the growth of each tree in the ensemble. An early example is bagging (Breiman, 1996), where to grow each tree a random selection (without replacement) is made from the examples in the training set.

Another example is random split selection (Dietterich, 1998) where at each node the split is selected at random from among the K best splits. Breiman (1999) generates new training sets by randomizing the outputs in the original training set. Another approach is to select the training set from a random set of weights on the examples in the training set. Ho (1998) has written a number of papers on "the random subspace" method which does a random selection of a subset of features to use to grow each tree.

In an important paper on written character recognition, Amit and Geman (1997) define a large number of geometric features and search over a random selection of these for the best split at each node. This latter paper has been influential in my thinking.

The common element in all of these procedures is that for the kth tree, a random vector Θ_k is generated, independent of the past random vectors $\Theta_1, \ldots, \Theta_{k-1}$ but with the same distribution; and a tree is grown using the training set and Θ_k, resulting in a classifier $h(\mathbf{x}, \Theta_k)$ where $\mathbf{x}$ is an input vector. For instance, in bagging the random vector Θ is

Breiman, L. Random Forests. Machine Learning 45, 5–32 (2001).

The randomness in the random forest algorithm: overview

1 Randomness 1/2 (same as in bagging)

Random forests = ensemble of trees grown by resampling (with replacement) training data, and selecting a random subset of $1 \leq m \leq p$ predictors at each step, for each tree.

2 Randomness 2/2 (new!)

By adding the randomness in the ensemble, the result is to decrease the variance, i.e., dependence on training data, of the forest. This is in contrast with individual trees, which typically **exhibit high variance** and tend to **overfit data**.

3 Standard choices: (1) $m=p$ (bagging), (2) $m = \sqrt{p}$

Where do we inject randomness into the forest?

sklearn.ensemble.RandomForestClassifier

class sklearn.ensemble.**RandomForestClassifier**(*n_estimators=100, *, criterion='gini', max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='sqrt', max_leaf_nodes=None,*

sklearn.ensemble.RandomForestRegressor

class sklearn.ensemble.**RandomForestRegressor**(*n_estimators=100, *, criterion='squared_error', max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features=1.0, max_leaf_nodes=None,*

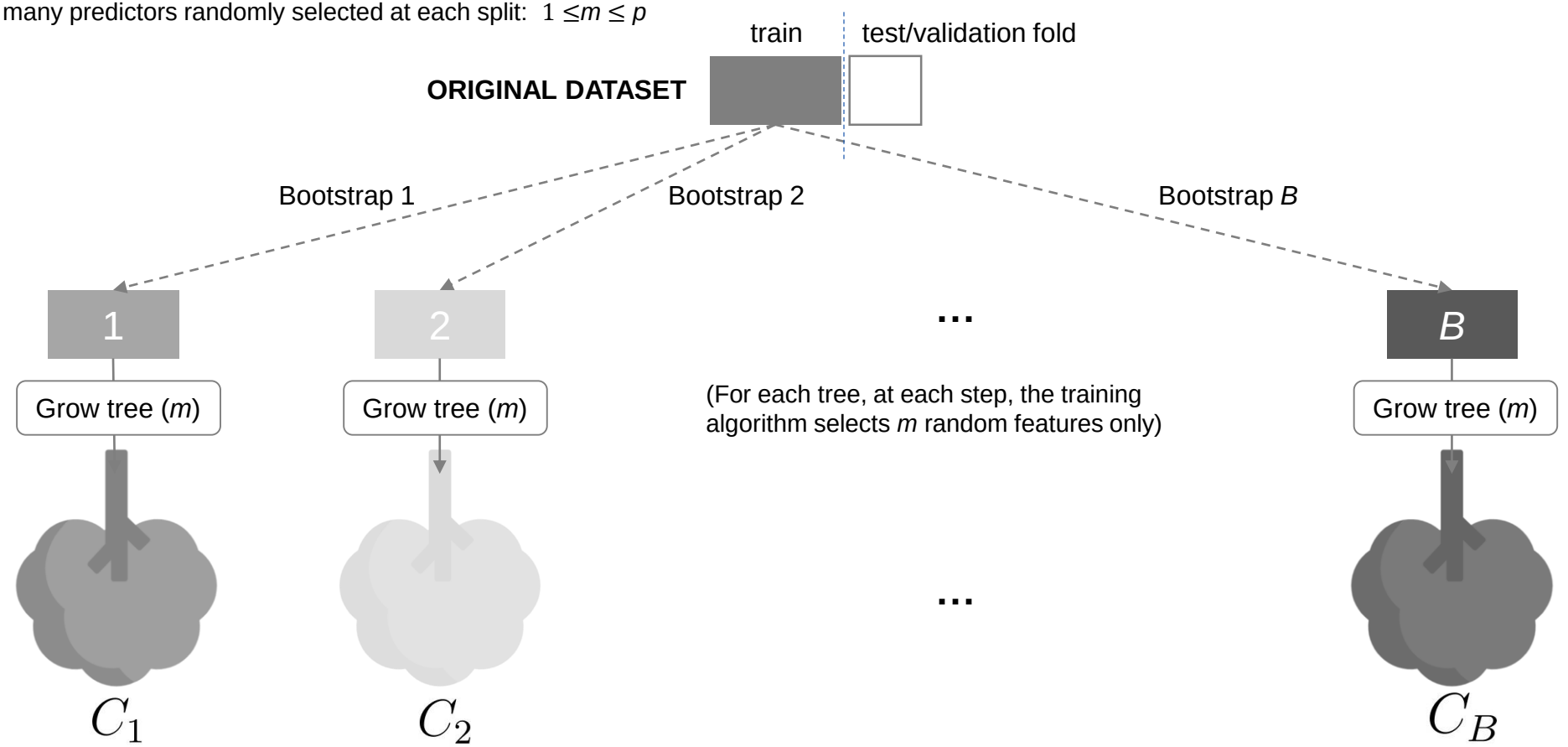
max_features : *{"sqrt", "log2", None}, int or float, default="sqrt"*

The number of features to consider when looking for the best split:

- If int, then consider `max_features` features at each split.
- If float, then `max_features` is a fraction and `max(1, int(max_features * n_features_in_))` features are considered at each split.
- If "sqrt", then `max_features=sqrt(n_features)`.
- If "log2", then `max_features=log2(n_features)`.
- If None, then `max_features=n_features`.

Random forests: A visual summary

Choose how many bootstrapped training datasets: B
Choose how many predictors randomly selected at each split: $1 \leq m \leq p$



$$C_{\mathrm{RF}}(x) = \frac{1}{B}\sum_{i=1}^{B} C_i(x)$$ (regression)

Random forests: An example from James et al. (2013), chapter 8

Gene expression dataset

1

(this is bagging)

"We applied random forests to a high-dimensional biological data set consisting of expression measurements of 4,718 genes measured on tissue samples from 349 patients. There are around 20,000 genes in humans, and individual genes have different levels of activity, or expression, on particular cells, tissues, and biological conditions. In this data set, each of the patient samples has a qualitative label with 15 different levels: either normal or 1 of 14 different types of cancer. Our goal was to use random forests to predict cancer type based on the 500 genes that have the largest variance in the training set." (pag. 321, James et al. (2013))

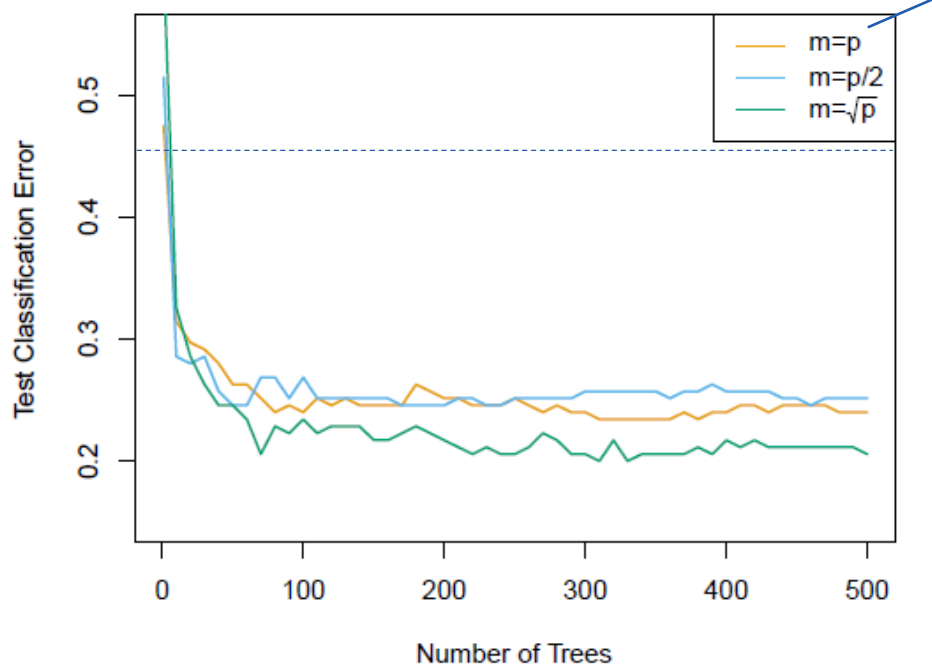


FIGURE 8.10. *Results from random forests for the 15-class gene expression data set with $p = 500$ predictors. The test error is displayed as a function of the number of trees. Each colored line corresponds to a different value of m, the number of predictors available for splitting at each interior tree node. Random forests ($m < p$) lead to a slight improvement over bagging ($m = p$). A single classification tree has an error rate of 45.7%.*

“Ensembling *via* learning sequentially”

Boosting

Two different approaches to ensembling

“Ensembling with randomization”

Train deep trees on bootstrapped copies of training data (and randomly select a subset of predictors at each step for random forests), then aggregate their predictions

“Ensembling *via* learning sequentially”

Ensemble “**weak learners**” sequentially, and turn them into a “**strong learner**”…

Two different approaches to ensembling

“Ensembling with randomization”

Train deep trees on bootstrapped copies of training data (and randomly select a subset of predictors at each step for random forests), then aggregate their predictions

“Ensembling *via* learning sequentially”

Ensemble “**weak learners**” sequentially, and turn them into a “**strong learner**”…

I **Adaptive Boosting (AdaBoost)**

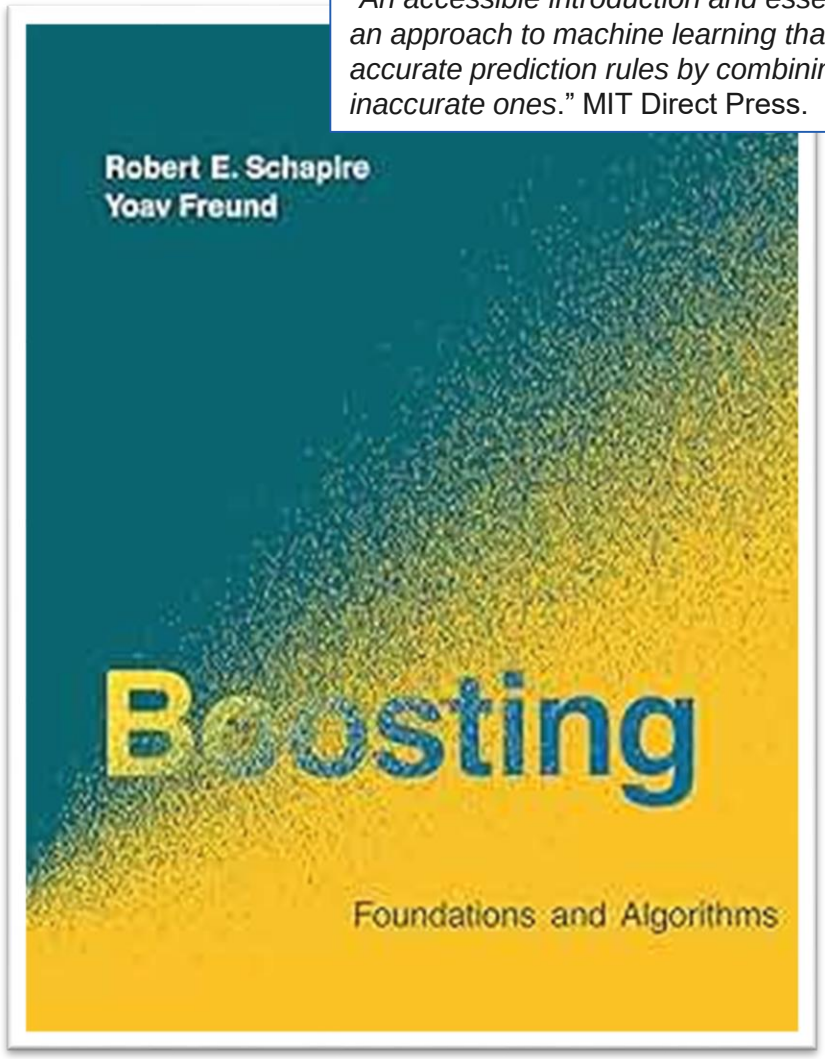
II **Gradient Boosting Machine**

I **Adaptive Boosting (AdaBoost)** – from the works of Schapire and Freund

"*An accessible introduction and essential reference for an approach to machine learning that creates highly accurate prediction rules by combining many weak and inaccurate ones*." MIT Direct Press.

"*In fact, Breiman (1996) (referring to a NIPS workshop) called AdaBoost with trees the "best off-the-shelf classifier in the world*"
(pag. 338) from

Jerome Friedman, Trevor Hastie, Robert Tibshirani, et al. Additive logistic regression: a statistical view of boosting (with discussion and a rejoinder by the authors). The Annals of Statistics, 28 (2):337–407, 2000.



Schapire, R.E., Freund, Y. (2012). Boosting: Foundations and Algorithms. MIT Press.

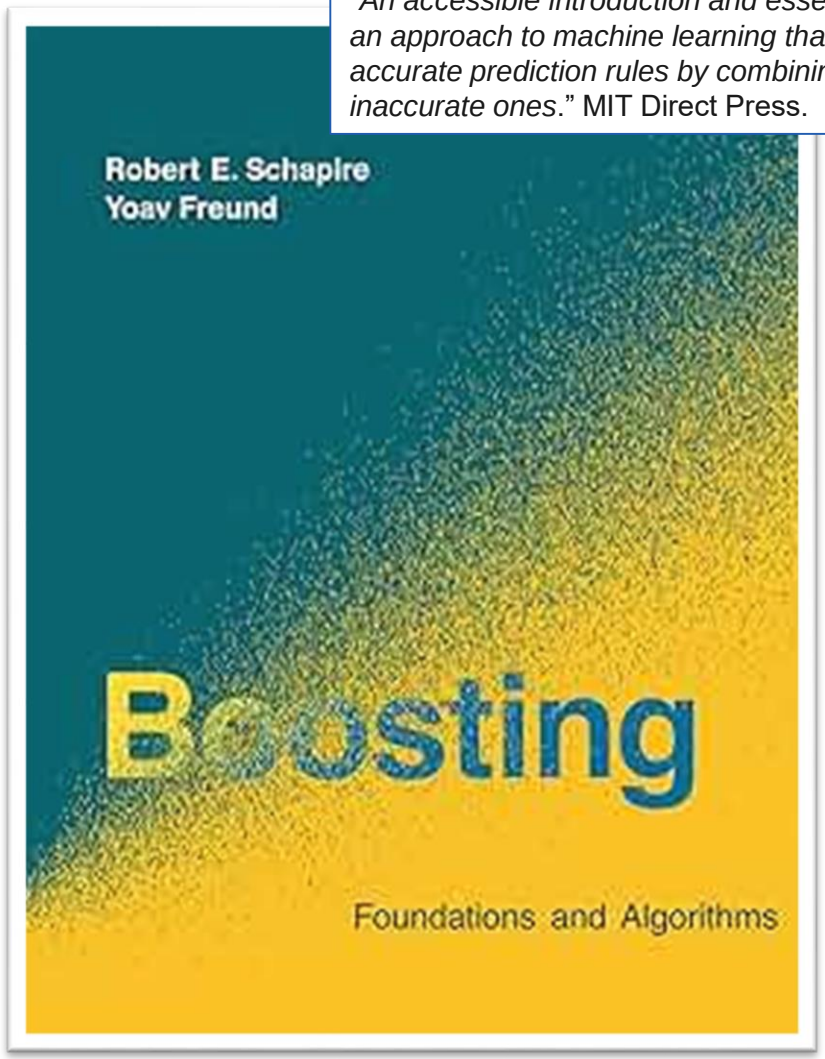
Adaptive Boosting (AdaBoost) – from the works of Schapire and Freund

"'An off-the-shelf' method is one that can be directly applied to the data without requiring a great deal of time-consuming data preprocessing or careful tuning of the learning procedure" (pag. 352, Hastie et al. (2013))

"*In fact, Breiman (1996) (referring to a NIPS workshop) called AdaBoost with trees the "best off-the-shelf classifier in the world*"
(pag. 338) from

Jerome Friedman, Trevor Hastie, Robert Tibshirani, et al. Additive logistic regression: a statistical view of boosting (with discussion and a rejoinder by the authors). The Annals of Statistics, 28 (2):337–407, 2000.

"*An accessible introduction and essential reference for an approach to machine learning that creates highly accurate prediction rules by combining many weak and inaccurate ones*." MIT Direct Press.



Schapire, R.E., Freund, Y. (2012). Boosting: Foundations and Algorithms. MIT Press.

Adaptive Boosting (AdaBoost) – a brief definition from a funny preprint

"AdaBoost is a boosting algorithm that produces a single ensemble learner from a **sequential additive process** which involves **re-weighting of training data** points (therefore the name AdaBoost, or 'Adaptive Boosting' of weights) and **fitting of weak learners**" (pag. 1, emphasis ours)

On Boosting: Theory and Applications

Andrea Ferrario*, Roger Hämmerli†

Prepared for:
Fachgruppe "Data Science"
Swiss Association of Actuaries SAV

Version of June 11, 2019

Abstract

We provide an overview of two commonly used boosting methodologies. We start with the description of different implementations as well as the statistical theory behind selected algorithms which are widely used by the machine learning community, then we discuss a case study focusing on the prediction of car insurance claims in a fixed future time interval. The results of the case study show that, overall, `XGBoost` performs better than `AdaBoost` and it shows best performance when shallow trees, moderate shrinking, the number of iterations increased with respect to default as well as subsampling of both features and training data points are considered.

Keywords. machine learning, boosting, predictive modeling, R, Python, car insurance, Kaggle, Porto Seguro, `AdaBoost`, `XGBoost`.

0 Introduction and overview

This data analytics tutorial has been written for the working group "Data Science" of the Swiss Association of Actuaries SAV, see

`https://www.actuarialdatascience.org/`

The main purpose of this tutorial is to provide an overview of the two most used boosting algorithms, i.e. `AdaBoost` and `XGBoost`. We start with introducing the basic approach of these two algorithms and show various implementations of them. We apply the algorithms to an insurance case study which uses data from the Porto Seguro's Safe Driver Prediction competition[1] hosted on Kaggle's platform[2]. For more statistically oriented literature on machine learning—and in particular on boosting—we refer to the monographs [31], [17], [19] and [5]. A list of references given at the end of this paper can be used to deep dive into the details of boosting.

*Mobiliar Lab for Analytics at ETH and Department of Management, Technology and Economics, ETH Zurich, aferrario@ethz.ch
†Schweizerische Mobiliar Versicherungsgesellschaft, roger.haemmerli@mobiliar.ch
[1] `https://www.kaggle.com/c/porto-seguro-safe-driver-prediction`
[2] `https://www.kaggle.com/`

1

Electronic copy available at: https://ssrn.com/abstract=3402687

Ferrario, Andrea and Hämmerli, Roger, On Boosting: Theory and Applications (June 11, 2019). Available at SSRN: https://ssrn.com/abstract=3402687 or http://dx.doi.org/10.2139/ssrn.3402687

Adaptive Boosting (AdaBoost) – a brief definition from a funny preprint

1

Sequential = information from step *n* is used at step *n+1*
Additive = the "strong learner" is obtained by aggregating the "weak learners" additively (linear combination)

"AdaBoost is a boosting algorithm that produces a single ensemble learner from a **sequential additive process** which involves **re-weighting of training data** points (therefore the name AdaBoost, or 'Adaptive Boosting' of weights) and **fitting of weak learners**" (pag. 1, emphasis ours)

3

Small trees, e.g., a "tree stump" (two leaves)

2

The training data set is **sequentially (re-)weighted** and weights are used to train each "weak learner" in the ensemble. The weights at step *n+1* depend on the performance of the "weak learner" trained at step *n.*

On Boosting: Theory and Applications

Andrea Ferrario*, Roger Hämmerli†

Prepared for:
Fachgruppe "Data Science"
Swiss Association of Actuaries SAV

Version of June 11, 2019

Abstract

We provide an overview of two commonly used boosting methodologies. We start with the description of different implementations as well as the statistical theory behind selected algorithms which are widely used by the machine learning community, then we discuss a case study focusing on the prediction of car insurance claims in a fixed future time interval. The results of the case study show that, overall, `XGBoost` performs better than `AdaBoost` and it shows best performance when shallow trees, moderate shrinking, the number of iterations increased with respect to default as well as subsampling of both features and training data points are considered.

Keywords. machine learning, boosting, predictive modeling, R, Python, car insurance, Kaggle, Porto Seguro, `AdaBoost`, `XGBoost`.

0 Introduction and overview

This data analytics tutorial has been written for the working group "Data Science" of the Swiss Association of Actuaries SAV, see

`https://www.actuarialdatascience.org/`

The main purpose of this tutorial is to provide an overview of the two most used boosting algorithms, i.e. `AdaBoost` and `XGBoost`. We start with introducing the basic approach of these two algorithms and show various implementations of them. We apply the algorithms to an insurance case study which uses data from the Porto Seguro's Safe Driver Prediction competition[1] hosted on Kaggle's platform[2]. For more statistically oriented literature on machine learning—and in particular on boosting—we refer to the monographs [31], [17], [19] and [5]. A list of references given at the end of this paper can be used to deep dive into the details of boosting.

*Mobiliar Lab for Analytics at ETH and Department of Management, Technology and Economics, ETH Zurich, aferrario@ethz.ch
†Schweizerische Mobiliar Versicherungsgesellschaft, roger.haemmerli@mobiliar.ch
[1] `https://www.kaggle.com/c/porto-seguro-safe-driver-prediction`
[2] `https://www.kaggle.com/`

1

Electronic copy available at: https://ssrn.com/abstract=3402687

Ferrario, Andrea and Hämmerli, Roger, On Boosting: Theory and Applications (June 11, 2019). Available at SSRN: https://ssrn.com/abstract=3402687 or http://dx.doi.org/10.2139/ssrn.3402687

AdaBoost: A visual summary

STEP

DATASET WEIGHTS

SEQUENCE

1

$w_i^{(1)}$

Grow a weak learner on training data using $w_i^{(1)}$

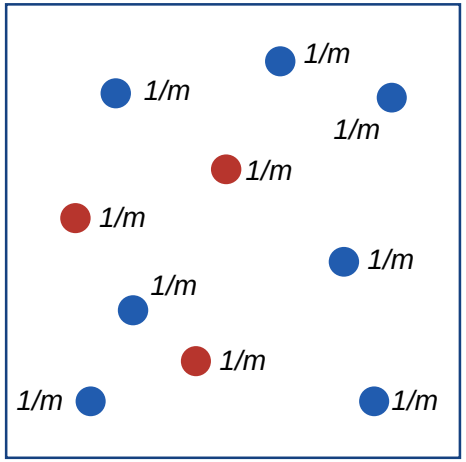
C_1 and α_1

(classical choice in the AdaBoost articles and Schapire and Freund's monograph)

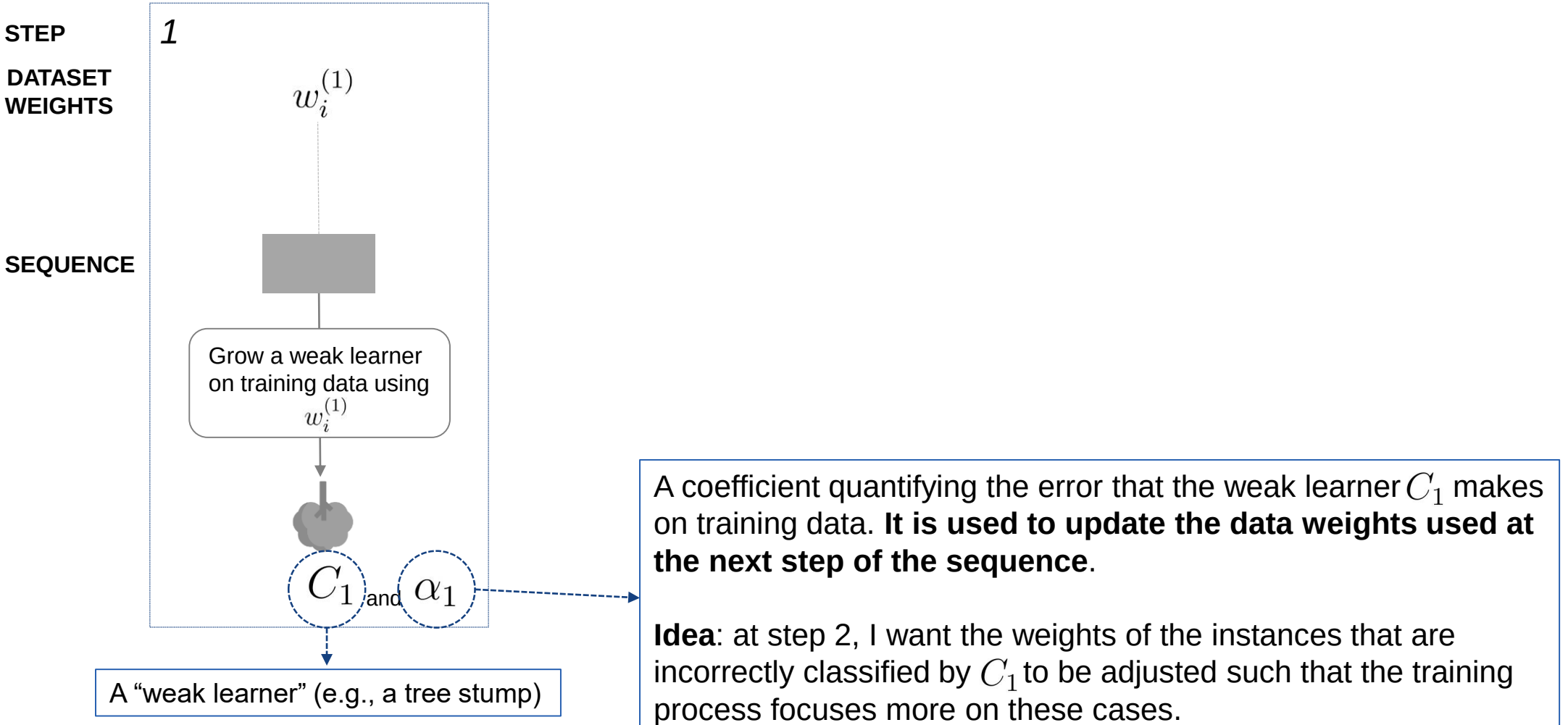
Training data = $\{(x_i, y_i) \in \mathcal{X} \times \{-1, +1\}, 1 \leq i \leq m\}$

Initialized by hand: usually *1/m*

Example: Training data (*m*=10)



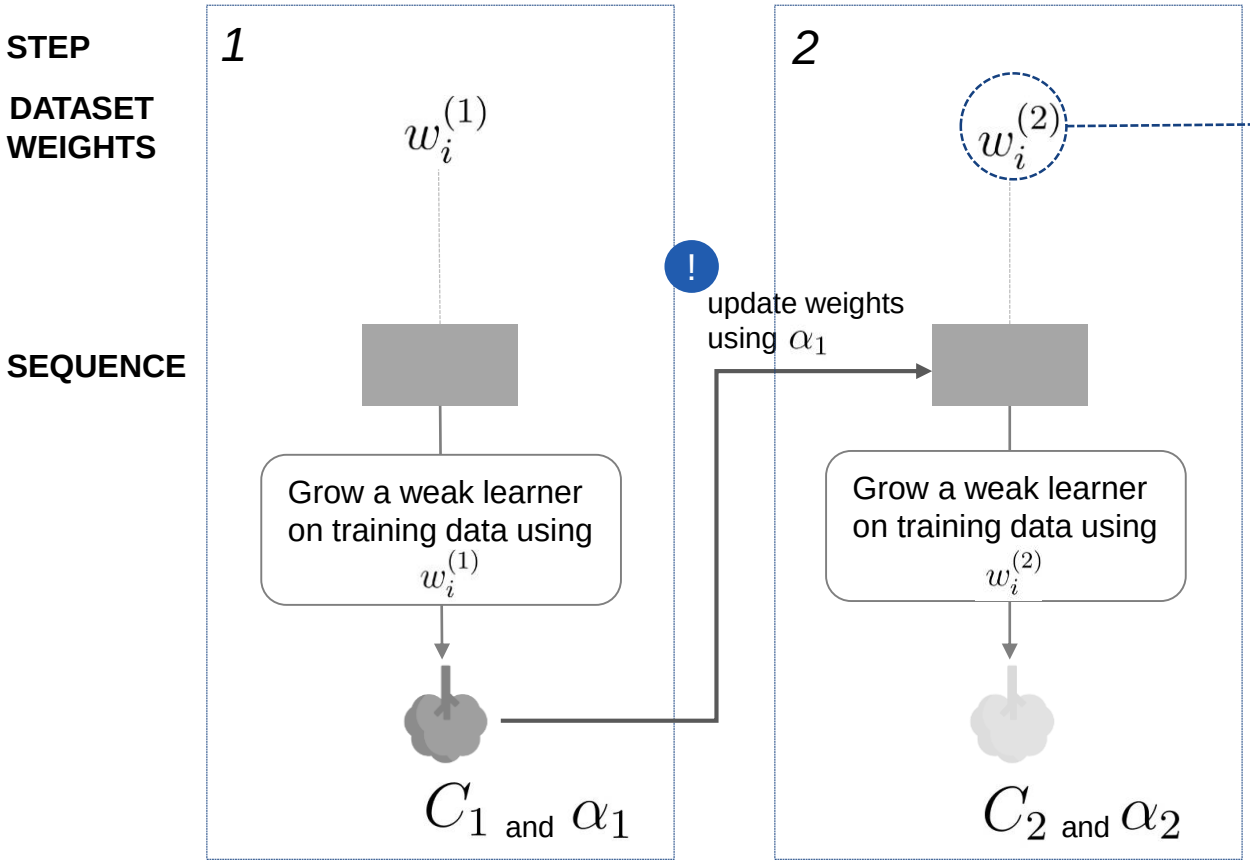
AdaBoost: A visual summary



A coefficient quantifying the error that the weak learner C_1 makes on training data. **It is used to update the data weights used at the next step of the sequence**.

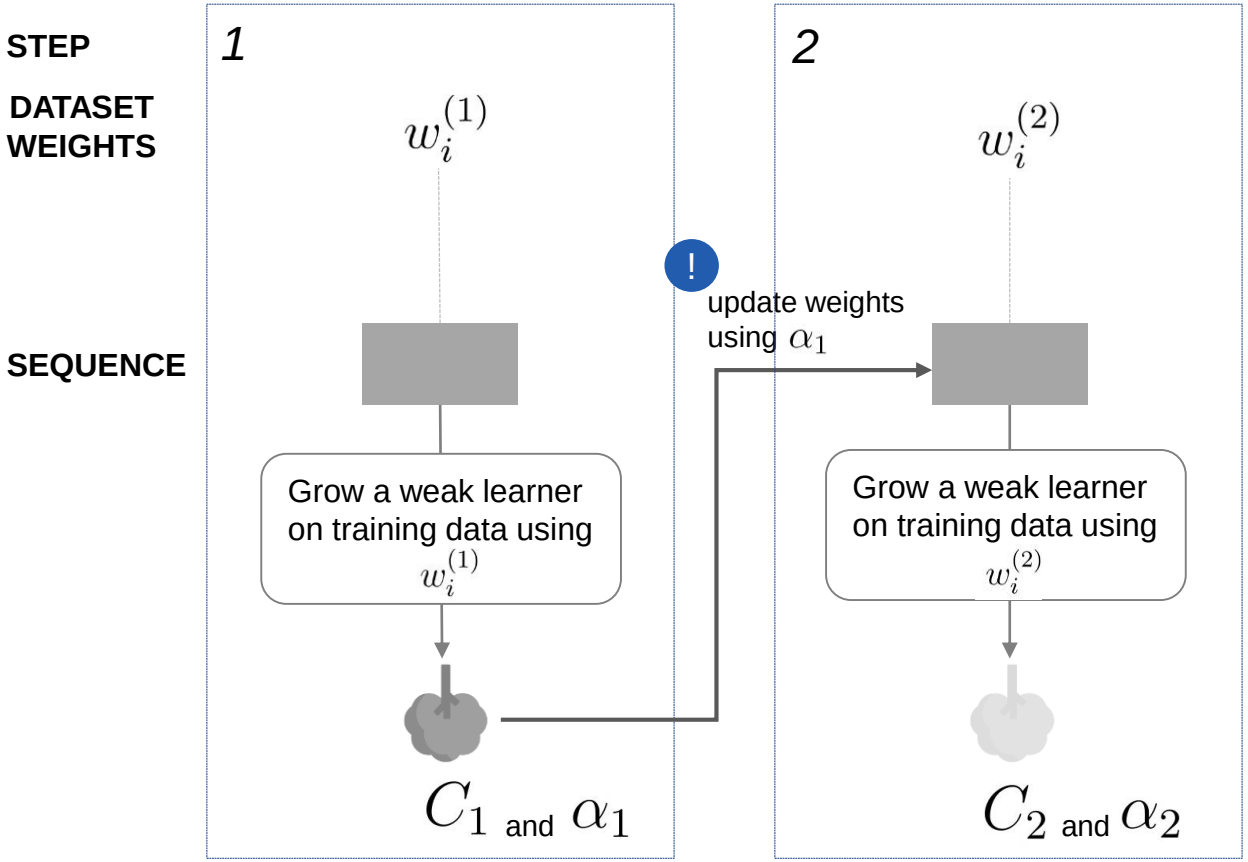
Idea: at step 2, I want the weights of the instances that are incorrectly classified by C_1 to be adjusted such that the training process focuses more on these cases.

AdaBoost: A visual summary

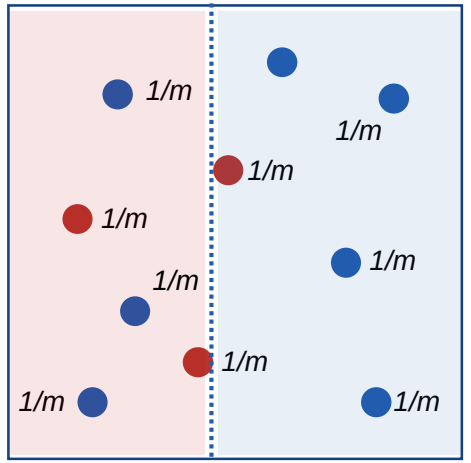


New weights that depend on the performance of the "weak learner" at step 1 that is encoded in the coefficient α_1. Again, the idea is that, at step 2, the misclassified data points have a higher weights than those classified correctly by C_1.

AdaBoost: A visual summary



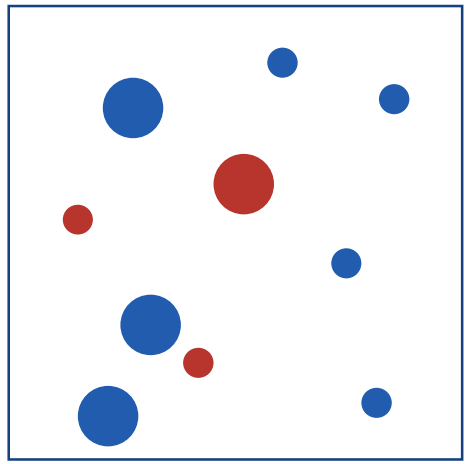
C_1 "weak learner" – a tree stump



Four misclassified training data samples

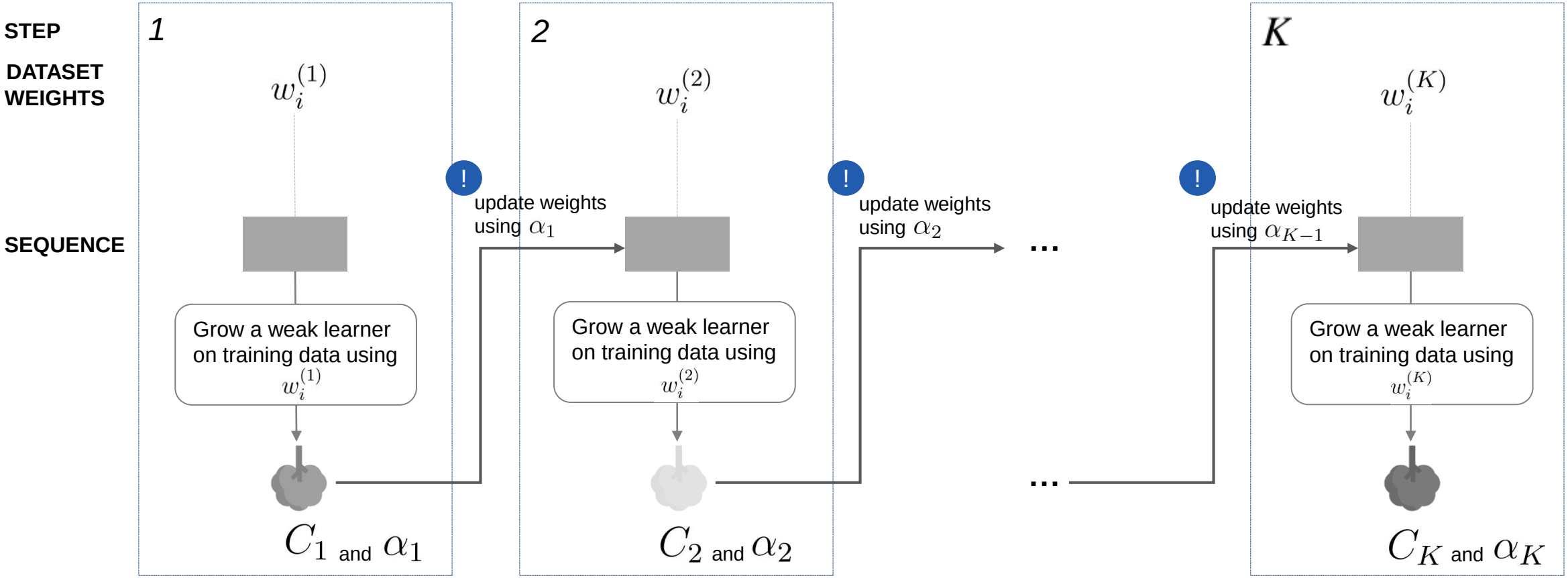
update weights using α_1

$w_i^{(2)}$

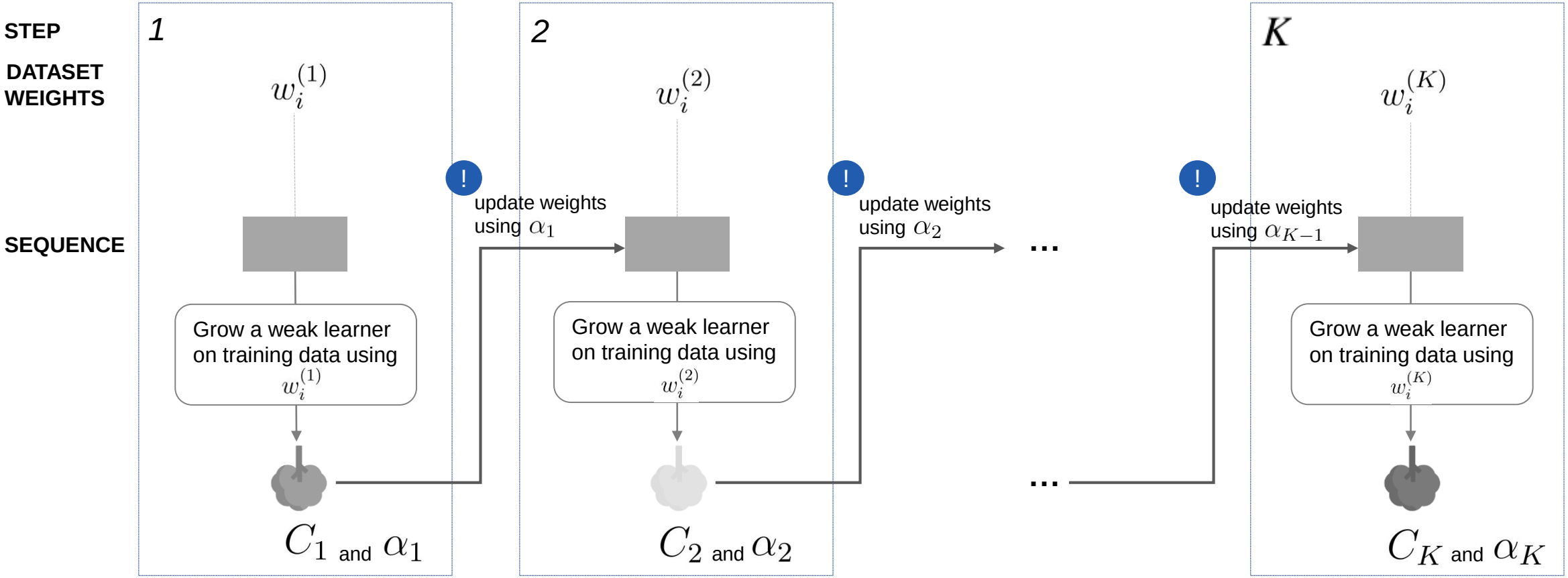


Misclassified data points get higher weights

AdaBoost: A visual summary



AdaBoost: A visual summary

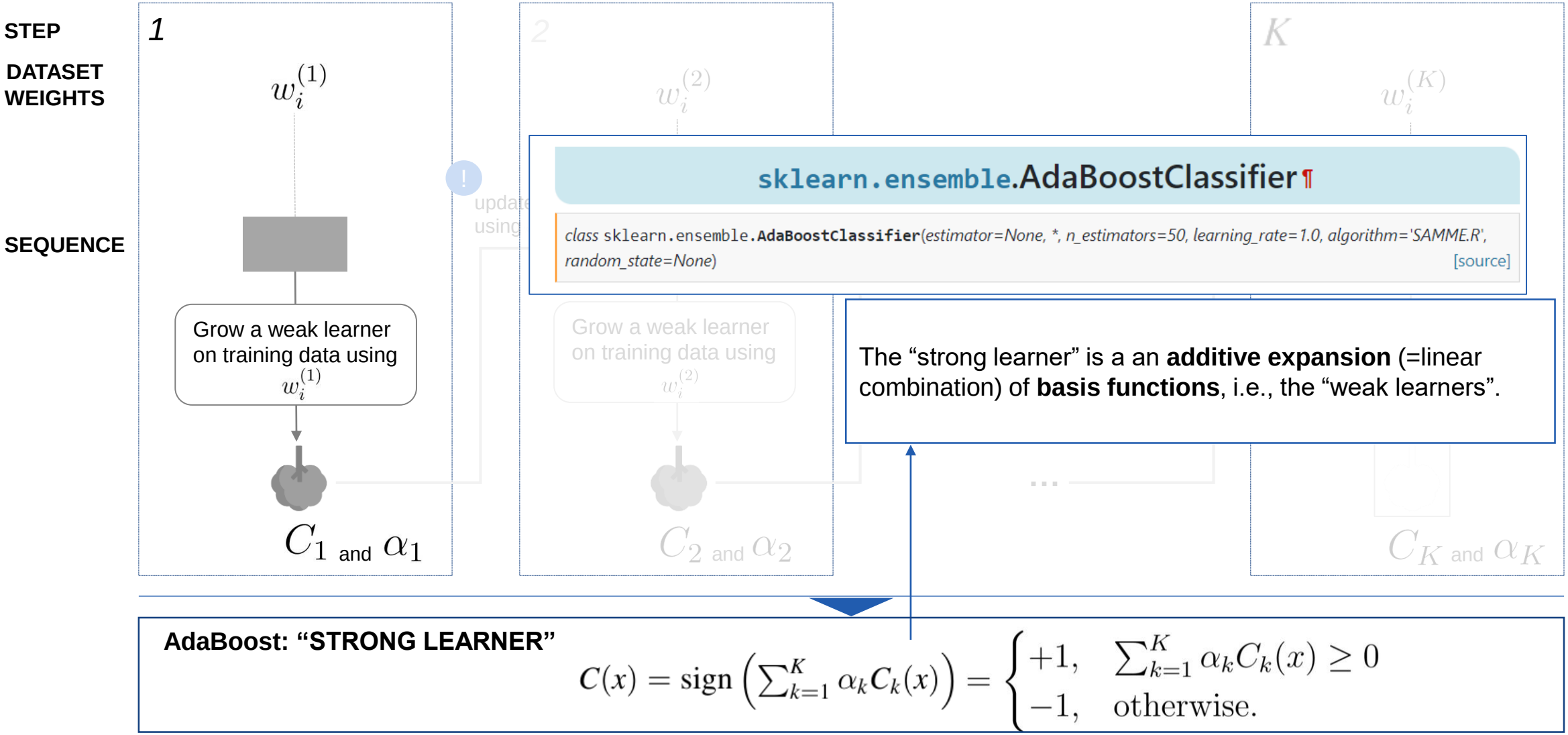


AdaBoost: "STRONG LEARNER"

$$C(x) = \text{sign}\left(\sum_{k=1}^{K} \alpha_k C_k(x)\right) = \begin{cases} +1, & \sum_{k=1}^{K} \alpha_k C_k(x) \geq 0 \\ -1, & \text{otherwise.} \end{cases}$$

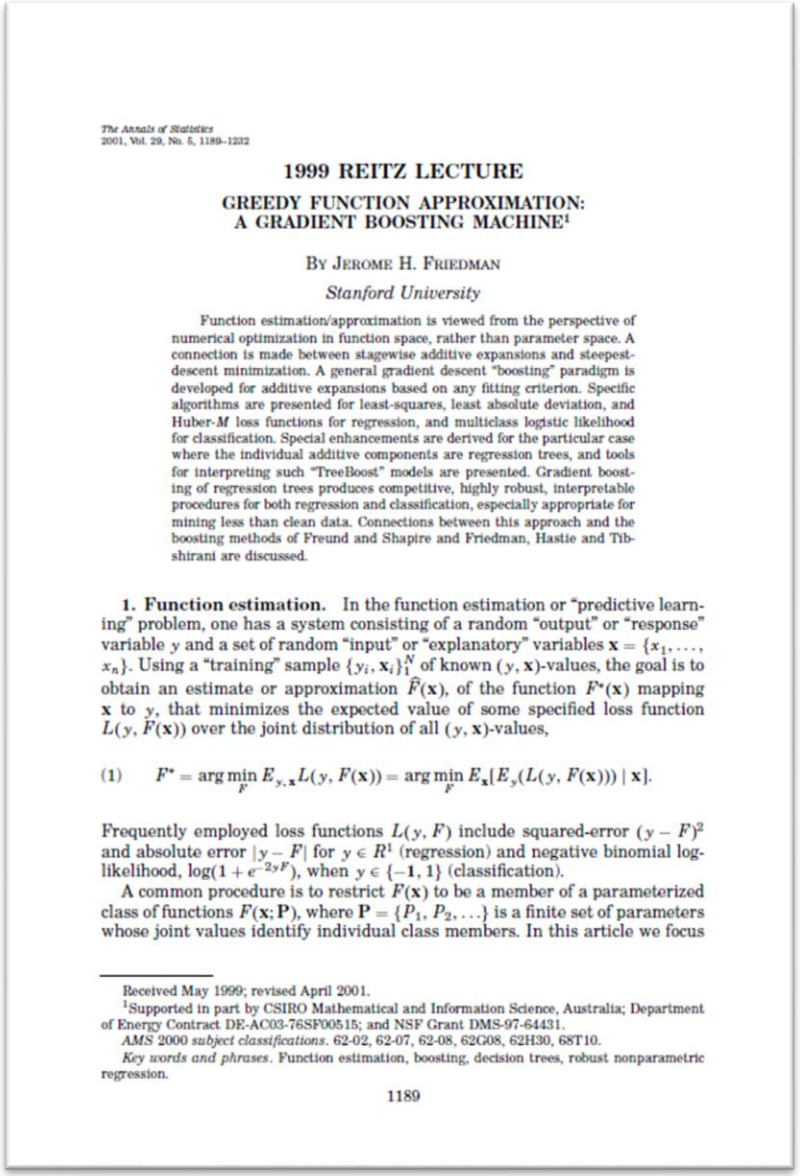
AdaBoost: A visual summary

You can use AdaBoost in Python



The Gradient Boosting Machine
An important idea by Friedman

The idea behind *gradient* boosting is to compute a sequence of "weak learners" and combine them additively ("strong learner") using the ***gradient*** of an **arbitrary (differentiable) loss function** chosen for the problem at hand and in a step-wise fashion.



The Annals of Statistics
2001, Vol. 29, No. 5, 1189–1232

1999 REITZ LECTURE

GREEDY FUNCTION APPROXIMATION:
A GRADIENT BOOSTING MACHINE[1]

BY JEROME H. FRIEDMAN

Stanford University

Function estimation/approximation is viewed from the perspective of numerical optimization in function space, rather than parameter space. A connection is made between stagewise additive expansions and steepest-descent minimization. A general gradient descent "boosting" paradigm is developed for additive expansions based on any fitting criterion. Specific algorithms are presented for least-squares, least absolute deviation, and Huber-M loss functions for regression, and multiclass logistic likelihood for classification. Special enhancements are derived for the particular case where the individual additive components are regression trees, and tools for interpreting such "TreeBoost" models are presented. Gradient boosting of regression trees produces competitive, highly robust, interpretable procedures for both regression and classification, especially appropriate for mining less than clean data. Connections between this approach and the boosting methods of Freund and Shapire and Friedman, Hastie and Tibshirani are discussed.

1. Function estimation. In the function estimation or "predictive learning" problem, one has a system consisting of a random "output" or "response" variable y and a set of random "input" or "explanatory" variables $\mathbf{x} = \{x_1, \ldots, x_n\}$. Using a "training" sample $\{y_i, \mathbf{x}_i\}_1^N$ of known $(y, \mathbf{x})$-values, the goal is to obtain an estimate or approximation $\hat{F}(\mathbf{x})$, of the function $F^*(\mathbf{x})$ mapping $\mathbf{x}$ to y, that minimizes the expected value of some specified loss function $L(y, F(\mathbf{x}))$ over the joint distribution of all $(y, \mathbf{x})$-values,

$$(1) \qquad F^* = \arg\min_F E_{y,\mathbf{x}} L(y, F(\mathbf{x})) = \arg\min_F E_{\mathbf{x}}[E_y(L(y, F(\mathbf{x}))) \mid \mathbf{x}].$$

Frequently employed loss functions $L(y, F)$ include squared-error $(y - F)^2$ and absolute error $|y - F|$ for $y \in R^1$ (regression) and negative binomial log-likelihood, $\log(1 + e^{-2yF})$, when $y \in \{-1, 1\}$ (classification).

A common procedure is to restrict $F(\mathbf{x})$ to be a member of a parameterized class of functions $F(\mathbf{x}; \mathbf{P})$, where $\mathbf{P} = \{P_1, P_2, \ldots\}$ is a finite set of parameters whose joint values identify individual class members. In this article we focus

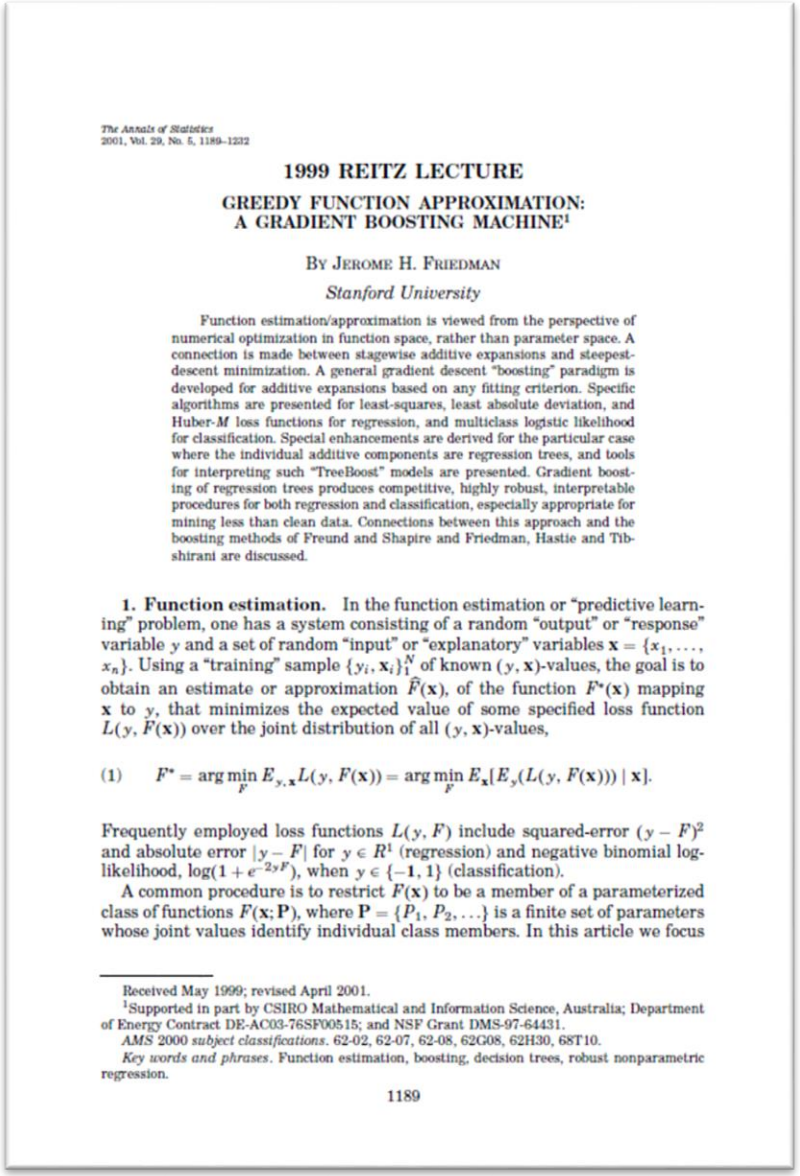
Received May 1999; revised April 2001.
[1]Supported in part by CSIRO Mathematical and Information Science, Australia; Department of Energy Contract DE-AC03-76SF00515; and NSF Grant DMS-97-64431.
AMS 2000 subject classifications. 62-02, 62-07, 62-08, 62G08, 62H30, 68T10.
Key words and phrases. Function estimation, boosting, decision trees, robust nonparametric regression.

Jerome H. Friedman. "Greedy function approximation: A gradient boosting machine.." Ann. Statist. 29 (5) 1189 - 1232, October 2001.

The Gradient Boosting Machine
An important idea by Friedman

The idea behind *gradient* boosting is to compute a sequence of "weak learners" and combine them additively ("strong learner") using the ***gradient*** of an **arbitrary (differentiable) loss function** chosen for the problem at hand and in a step-wise fashion.

Remark: this procedure is what AdaBoost also does…it took some time to researchers to prove this point.



The Annals of Statistics
2001, Vol. 29, No. 5, 1189–1232

1999 REITZ LECTURE

GREEDY FUNCTION APPROXIMATION:
A GRADIENT BOOSTING MACHINE[1]

BY JEROME H. FRIEDMAN

Stanford University

Function estimation/approximation is viewed from the perspective of numerical optimization in function space, rather than parameter space. A connection is made between stagewise additive expansions and steepest-descent minimization. A general gradient descent "boosting" paradigm is developed for additive expansions based on any fitting criterion. Specific algorithms are presented for least-squares, least absolute deviation, and Huber-M loss functions for regression, and multiclass logistic likelihood for classification. Special enhancements are derived for the particular case where the individual additive components are regression trees, and tools for interpreting such "TreeBoost" models are presented. Gradient boosting of regression trees produces competitive, highly robust, interpretable procedures for both regression and classification, especially appropriate for mining less than clean data. Connections between this approach and the boosting methods of Freund and Shapire and Friedman, Hastie and Tibshirani are discussed.

1. Function estimation. In the function estimation or "predictive learning" problem, one has a system consisting of a random "output" or "response" variable y and a set of random "input" or "explanatory" variables $\mathbf{x} = \{x_1, \ldots, x_n\}$. Using a "training" sample $\{y_i, \mathbf{x}_i\}_1^N$ of known $(y, \mathbf{x})$-values, the goal is to obtain an estimate or approximation $\hat{F}(\mathbf{x})$, of the function $F^*(\mathbf{x})$ mapping $\mathbf{x}$ to y, that minimizes the expected value of some specified loss function $L(y, F(\mathbf{x}))$ over the joint distribution of all $(y, \mathbf{x})$-values,

$$(1) \qquad F^* = \arg\min_F E_{y,\mathbf{x}} L(y, F(\mathbf{x})) = \arg\min_F E_{\mathbf{x}}[E_y(L(y, F(\mathbf{x}))) \mid \mathbf{x}].$$

Frequently employed loss functions $L(y, F)$ include squared-error $(y - F)^2$ and absolute error $|y - F|$ for $y \in R^1$ (regression) and negative binomial log-likelihood, $\log(1 + e^{-2yF})$, when $y \in \{-1, 1\}$ (classification).

A common procedure is to restrict $F(\mathbf{x})$ to be a member of a parameterized class of functions $F(\mathbf{x}; \mathbf{P})$, where $\mathbf{P} = \{P_1, P_2, \ldots\}$ is a finite set of parameters whose joint values identify individual class members. In this article we focus

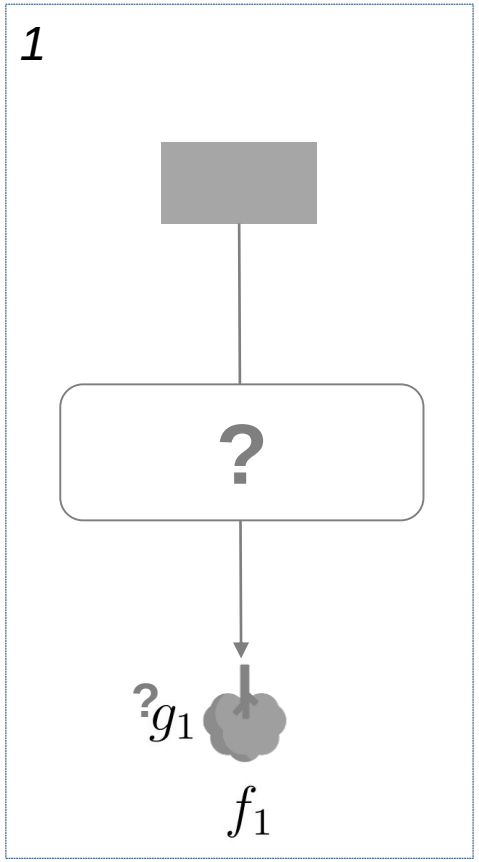
Received May 1999; revised April 2001.
[1]Supported in part by CSIRO Mathematical and Information Science, Australia; Department of Energy Contract DE-AC03-76SF00515; and NSF Grant DMS-97-64431.
AMS 2000 subject classifications. 62-02, 62-07, 62-08, 62G08, 62H30, 68T10.
Key words and phrases. Function estimation, boosting, decision trees, robust nonparametric regression.

Jerome H. Friedman. "Greedy function approximation: A gradient boosting machine.." Ann. Statist. 29 (5) 1189 - 1232, October 2001.

Gradient Boosting: A visual summary

Let us start with the very first step of the sequence

STEP *1*



Key Inputs

Regression and classification

Training data [redacted] $= \{(x_i, y_i) \in \mathcal{X} \times \mathcal{Y}, 1 \leq i \leq m\}$

Loss function $L : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}_+$

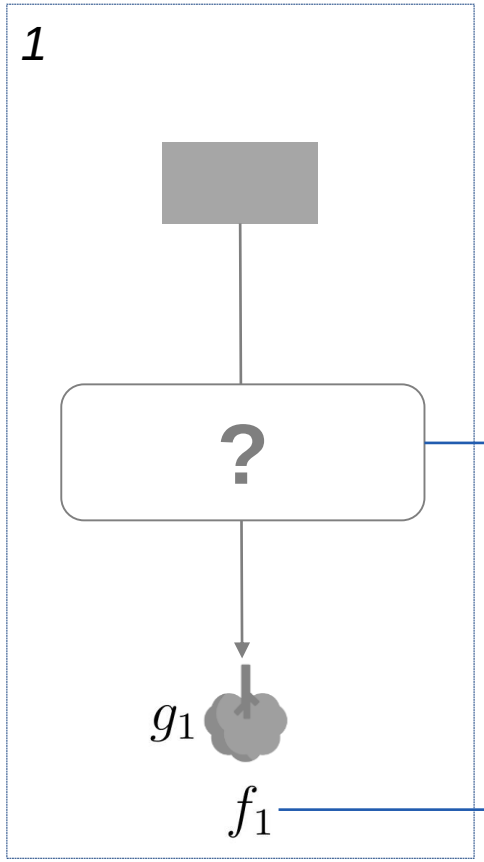
$L(z, w) = (z - w)^2$ (regression)

$f_0(x) = 0$ (convenient initialization – we need it later)

Gradient Boosting: A visual summary

The core of the gradient boosting method: fitting a weak learner to the gradient of the loss function (also known as the “residuals”)

STEP *1*



These are *m* known numbers (one residual per training data point)!

1

We fit a tree (“weak learner”) to the “residuals” of the model from the previous step :

$$r_{1,i} = \left[\frac{\partial L(\hat{y}, y_i)}{\partial \hat{y}}\right]_{\hat{y}=f_0(x_i)}$$

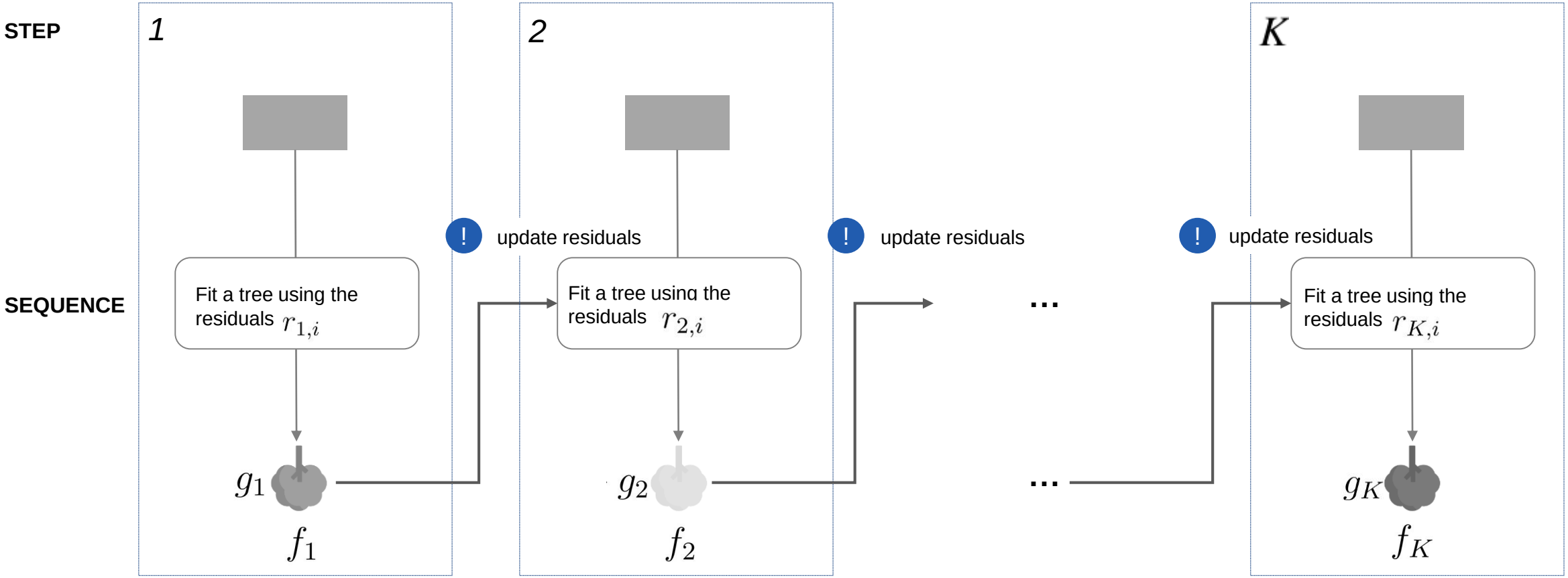
Let us call the resulting tree g_1.

2

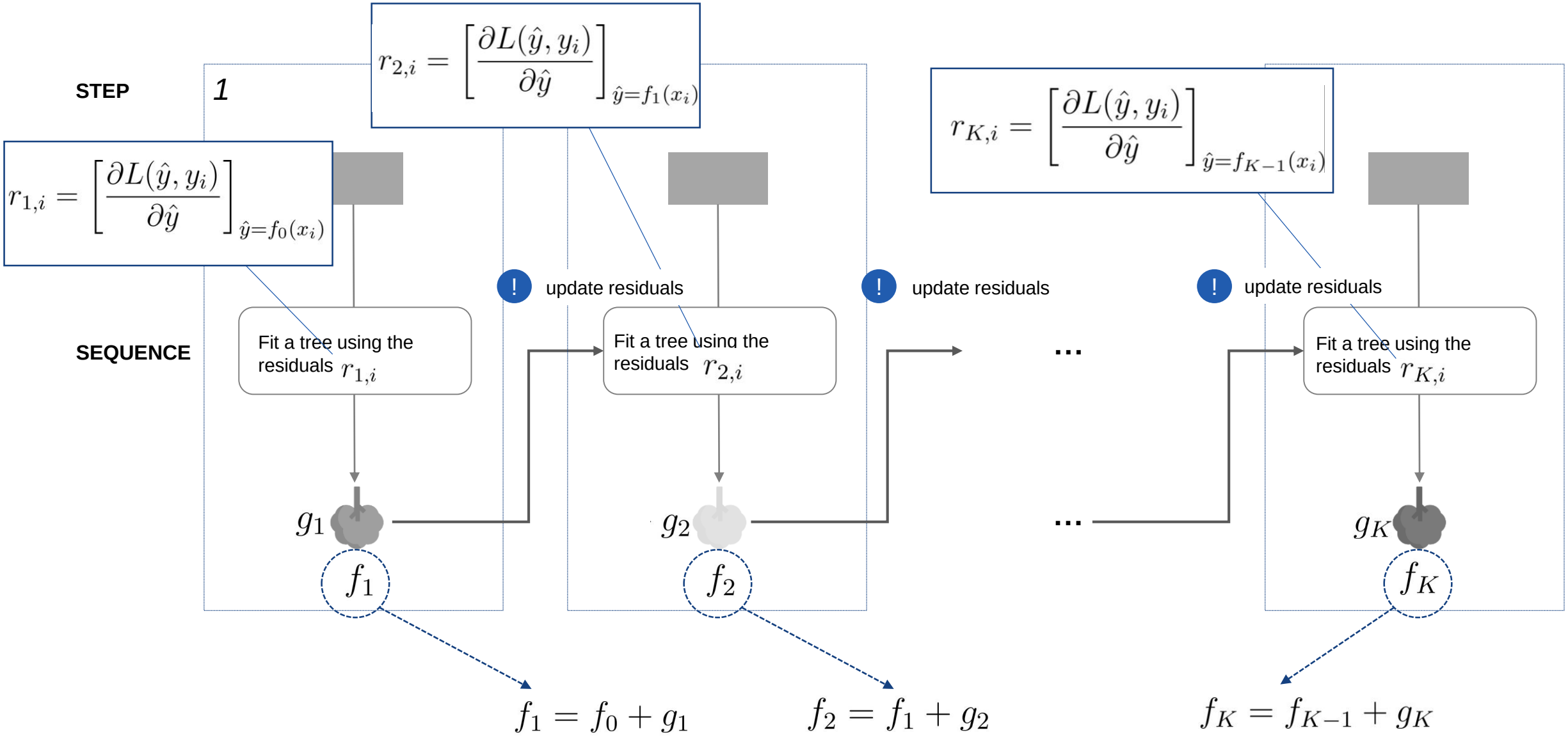
We define: $f_1 = f_0 + g_1$

Idea—like in gradient descent: by fitting the weak learner to the gradient of the function, I can then walk the direction of steepest descent, step-wise.

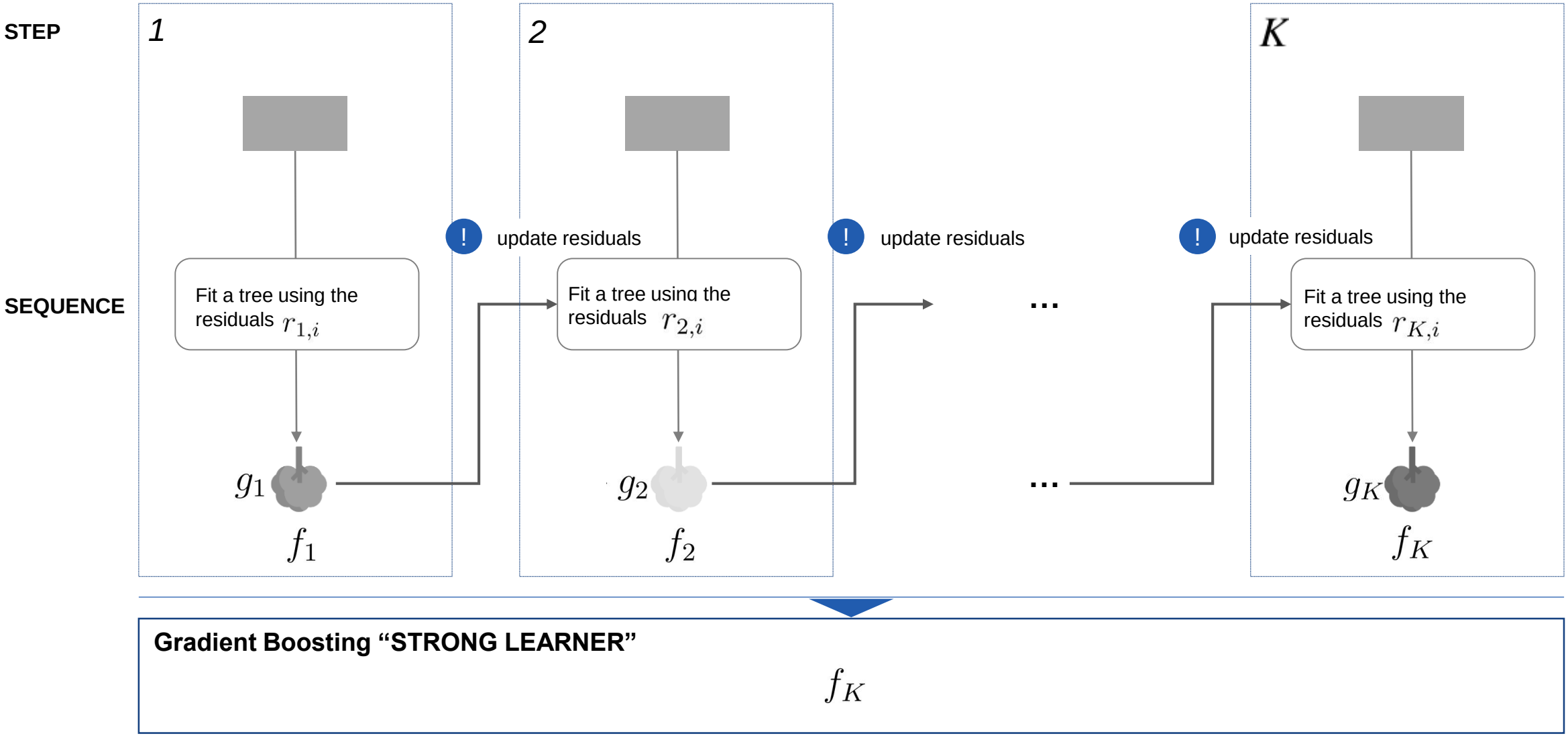
Gradient Boosting: A visual summary



Gradient Boosting: A visual summary



II Gradient Boosting: A visual summary



Two famous implementations of Gradient Boosting

Both are available in Python

Welcome to LightGBM's documentation!

LightGBM is a gradient boosting framework that uses tree based learning algorithms. It is designed to be distributed and efficient with the following advantages:

- Faster training speed and higher efficiency.
- Lower memory usage.
- Better accuracy.
- Support of parallel, distributed, and GPU learning.
- Capable of handling large-scale data.

dmlc XGBoost

XGBoost Documentation

XGBoost is an optimized distributed gradient boosting library designed to be highly **efficient**, **flexible** and **portable**. It implements machine learning algorithms under the Gradient Boosting framework. XGBoost provides a parallel tree boosting (also known as GBDT, GBM) that solve many data science problems in a fast and accurate way. The same code runs on major distributed environment (Hadoop, SGE, MPI) and can solve problems beyond billions of examples.

Deep learning is not all you need: Tree-based ensembling shows high performance on tabular data

When Do Neural Nets Outperform Boosted Trees on Tabular Data?

Duncan McElfresh[1,2], Sujay Khandagale[3], Jonathan Valverde[4], Vishak Prasad C[5], Benjamin Feuer[6], Chinmay Hegde[3], Ganesh Ramakrishnan[5], Micah Goldblum[6], Colin White[2,7]
[1] Abacus.AI, [2] Stanford, [3] Pinterest, [4] University of Maryland,
[5] IIT Bombay, [6] New York University, [7] Caltech

Abstract

Tabular data is one of the most commonly used types of data in machine learning. Despite recent advances in neural nets (NNs) for tabular data, there is still an active discussion on whether or not NNs generally outperform gradient-boosted decision trees (GBDTs) on tabular data, with several recent works arguing either that GBDTs consistently outperform NNs on tabular data, or vice versa. In this work, we take a step back and question the importance of this debate. To this end, we conduct the largest tabular data analysis to date, comparing 19 algorithms across 176 datasets, and we find that the 'NN vs. GBDT' debate is overemphasized: for a surprisingly high number of datasets, either the performance difference between GBDTs and NNs is negligible, or light hyperparameter tuning on a GBDT is more important than choosing between NNs and GBDTs. A remarkable exception is the recently-proposed prior-data fitted network, TabPFN: although it is effectively limited to training sets of size 3000, we find that it outperforms all other algorithms on average, even when randomly sampling 3000 training datapoints. Next, we analyze dozens of metafeatures to determine what *properties* of a dataset make NNs or GBDTs better-suited to perform well. For example, we find that GBDTs are much better than NNs at handling skewed or heavy-tailed feature distributions and other forms of dataset irregularities. Our insights act as a guide for practitioners to determine which technique may work best on their dataset. Finally, with the goal of accelerating tabular data research, we release the TabZilla Benchmark Suite: a collection of the 36 'hardest' of the datasets we study. Our benchmark suite, codebase, and all raw results are available at https://github.com/naszilla/tabzilla.

1 Introduction

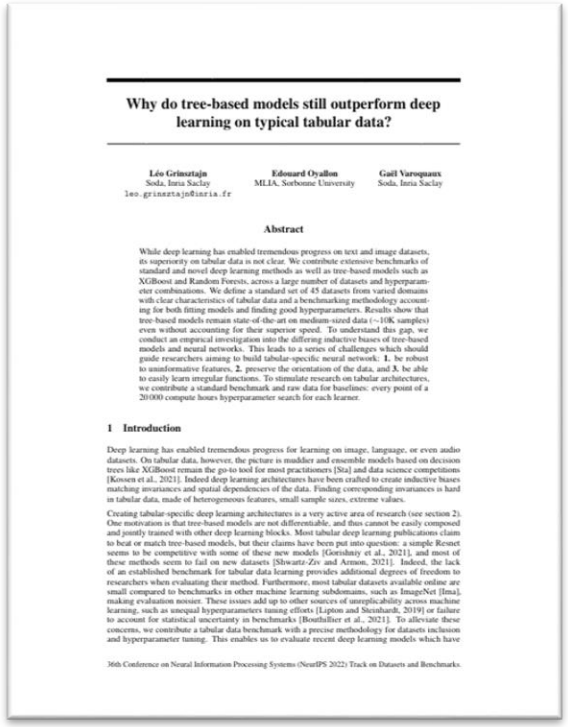
Tabular datasets are data organized into rows and columns, consisting of distinct features that are typically continuous, categorical, or ordinal. They are the oldest and among the most ubiquitous dataset types in machine learning in practice [8, 61], due to their numerous applications across medicine [38, 64], finance [6, 14], online advertising [29, 48, 56], and many other areas [10, 11, 65].

Despite recent advances in neural network (NN) architectures for tabular data [5, 53], there is still an active debate over whether or not NNs generally outperform gradient-boosted decision trees (GBDTs) on tabular data, with multiple works arguing either for [5, 39, 43, 53, 57] or against [8, 27, 28, 61] NNs. This is in stark contrast to other areas such as computer vision and natural language understanding, in which NNs have far outpaced competing methods [9, 19, 20, 42, 67].

*Email: duncan@abacus.ai, crwhite@caltech.edu. Work done while SK and JV were at Abacus.AI.

37th Conference on Neural Information Processing Systems (NeurIPS 2023) Track on Datasets and Benchmarks.

McElfresh, D., Khandagale, S., Valverde, J., Prasad C, V., Ramakrishnan, G., Goldblum, M., & White, C. (2024). When do neural nets outperform boosted trees on tabular data?. *Advances in Neural Information Processing Systems*, *36*.



Why do tree-based models still outperform deep learning on typical tabular data?

Léo Grinsztajn
Soda, Inria Saclay
leo.grinsztajn@inria.fr

Edouard Oyallon
MLIA, Sorbonne University

Gaël Varoquaux
Soda, Inria Saclay

Abstract

While deep learning has enabled tremendous progress on text and image datasets, its superiority on tabular data is not clear. We contribute extensive benchmarks of standard and novel deep learning methods as well as tree-based models such as XGBoost and Random Forests, across a large number of datasets and hyperparameter combinations. We define a standard set of 45 datasets from varied domains with clear characteristics of tabular data and a benchmarking methodology accounting for both fitting models and finding good hyperparameters. Results show that tree-based models remain state-of-the-art on medium-sized data (~10K samples) even without accounting for their superior speed. To understand this gap, we conduct an empirical investigation into the differing inductive biases of tree-based models and neural networks. This leads to a series of challenges which should guide researchers aiming to build tabular-specific neural network: **1.** be robust to uninformative features, **2.** preserve the orientation of the data, and **3.** be able to easily learn irregular functions. To stimulate research on tabular architectures, we contribute a standard benchmark and raw data for baselines: every point of a 20'000 compute hours hyperparameter search for each learner.

1 Introduction

Deep learning has enabled tremendous progress for learning on image, language, or even audio datasets. On tabular data, however, the picture is muddier and ensemble models based on decision trees like XGBoost remain the go-to tool for most practitioners [Sta] and data science competitions [Kossen et al., 2021]. Indeed deep learning architectures have been crafted to create inductive biases matching invariances and spatial dependencies of the data. Finding corresponding invariances is hard on tabular data, made of heterogeneous features, small sample sizes, extreme values.

Creating tabular-specific deep learning architectures is a very active area of research (see section 2). One motivation is that tree-based models are not differentiable, and thus cannot be easily composed and jointly trained with other deep learning blocks. Most tabular deep learning publications claim to beat or match tree-based models, but their claims have been put into question: a simple Resnet seems to be competitive with some of these new models [Gorishniy et al., 2021], and most of these methods seem to fail on new datasets [Shwartz-Ziv and Armon, 2021]. Indeed, the lack of an established benchmark for tabular data learning provides additional degrees of freedom to researchers when evaluating their method. Furthermore, most tabular datasets available online are small compared to benchmarks in other machine learning subdomains, such as ImageNet [Ima], making evaluation noisier. These issues add up to other sources of unreplicability across machine learning, such as unequal hyperparameters tuning efforts [Lipton and Steinhardt, 2019] or failure to account for statistical uncertainty in benchmarks [Bouthillier et al., 2021]. To alleviate these concerns, we contribute a tabular data benchmark with a precise methodology for datasets inclusion and hyperparameter tuning. This enables us to evaluate recent deep learning models which have

36th Conference on Neural Information Processing Systems (NeurIPS 2022) Track on Datasets and Benchmarks.

Grinsztajn, L., Oyallon, E., & Varoquaux, G. (2022). Why do tree-based models still outperform deep learning on typical tabular data?. *Advances in neural information processing systems*, *35*, 507-520.

Information Fusion 81 (2022) 84–90

Contents lists available at ScienceDirect

Information Fusion

journal homepage: www.elsevier.com/locate/inffus

Full length article

Tabular data: Deep learning is not all you need

Ravid Shwartz-Ziv*, Amitai Armon
IT AI Group, Intel, Israel

ARTICLE INFO

Keywords:
Tabular data
Deep neural networks
Tree-based models
Hyperparameter optimization

ABSTRACT

A key element in solving real-life data science problems is selecting the types of models to use. Tree ensemble models (such as XGBoost) are usually recommended for classification and regression problems with tabular data. However, several deep learning models for tabular data have recently been proposed, claiming to outperform XGBoost for some use cases. This paper explores whether these deep models should be a recommended option for tabular data by rigorously comparing the new deep models to XGBoost on various datasets. In addition to systematically comparing their performance, we consider the tuning and computation they require. Our study shows that XGBoost outperforms these deep models across the datasets, including the datasets used in the papers that proposed the deep models. We also demonstrate that XGBoost requires much less tuning. On the positive side, we show that an ensemble of deep models and XGBoost performs better on these datasets than XGBoost alone.

1. Introduction

Deep neural networks have demonstrated great success across various domains, including images, audio, and text [1–3]. There are several canonical architectures for encoding raw data into meaningful representations in these domains. These canonical architectures usually perform well in real-world applications.

In real-world applications, the most common data type is tabular data, comprising samples (rows) with the same set of features (columns). Tabular data is used in practical applications in many fields, including medicine, finance, manufacturing, climate science, and many other applications that are based on relational databases. During the last decade, traditional machine learning methods, such as gradient-boosted decision trees (GBDT) [4], still dominated tabular data modeling and showed superior performance over deep learning. When deep neural networks are applied to tabular data, many challenges arise, such as lack of locality, data sparsity (missing values), mixed feature types (numerical, ordinal, and categorical), and lack of prior knowledge about the dataset structure (unlike with text or images). Although the "no free lunch" principle [5] always applies, tree-ensemble algorithms, such as XGBoost, are considered the recommended option for real-life tabular data problems [4,6,7].

Recently, there have been several attempts to develop deep networks for tabular data [8–10], some of which have been claimed to outperform GBDT. However, each work in this field used different datasets since there is no standard benchmark (such as ImageNet [11] or GLUE [12]). As a result, it is challenging to compare tabular data models accurately, noting that some lack open-source implementations. In addition, papers that attempted to compare these models often did not optimize them equally. Thus, it became unclear whether deep learning models for tabular data surpass GBDT and which deep learning models perform best. These obscurities impede the research and development process and make the conclusions from these papers unclear. As the number of papers on deep learning for tabular data is rising, we believe it is time to rigorously review recent developments in this area and provide more substantiated conclusions that will serve as a basis for future research. The main purpose of this study is to explore whether any of the recently proposed deep models should indeed be a recommended choice for tabular dataset problems. There are two parts to this question: (1) Are the models more accurate, especially for datasets that did not appear in the paper that proposed them? (2) How long do training and hyperparameter search take in comparison to other models?

To answer these questions, we evaluate these recently proposed deep learning models and XGBoost on diverse tabular datasets with the same tuning protocol. We analyze the deep models proposed in four recent papers using eleven datasets, nine of which were used in these papers. We show that in most cases, each deep model performs best on the datasets used in its respective paper but significantly worse on other datasets. Additionally, our study shows that XGBoost usually outperforms deep models on these datasets. Furthermore, we demonstrate that the hyperparameter search process was much shorter for XGBoost. On the positive side, we examine the performance of an

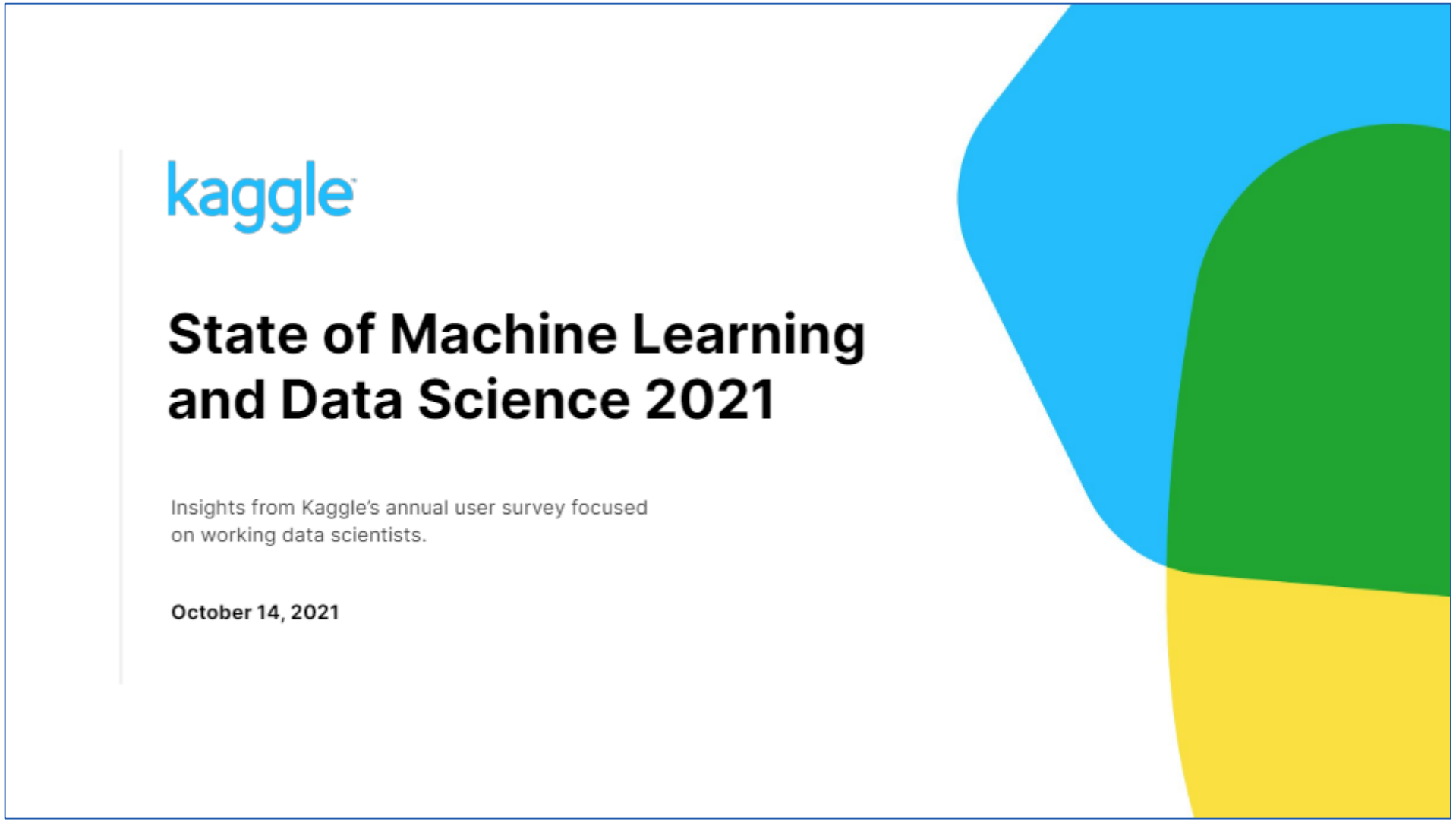
* Corresponding author.
E-mail addresses: ravid.shz@intel.com (R. Shwartz-Ziv), amitai.armon@intel.com (A. Armon).

https://doi.org/10.1016/j.inffus.2021.11.011
Received 22 June 2021; Received in revised form 24 September 2021; Accepted 9 November 2021
Available online 27 November 2021
1566-2535/

Shwartz-Ziv, R., & Armon, A. (2022). Tabular data: Deep learning is not all you need. *Information Fusion*, *81*, 84-90.

Kaggle: tree-based ensemble methods are still a popular choice

The presentation can be found on Moodle



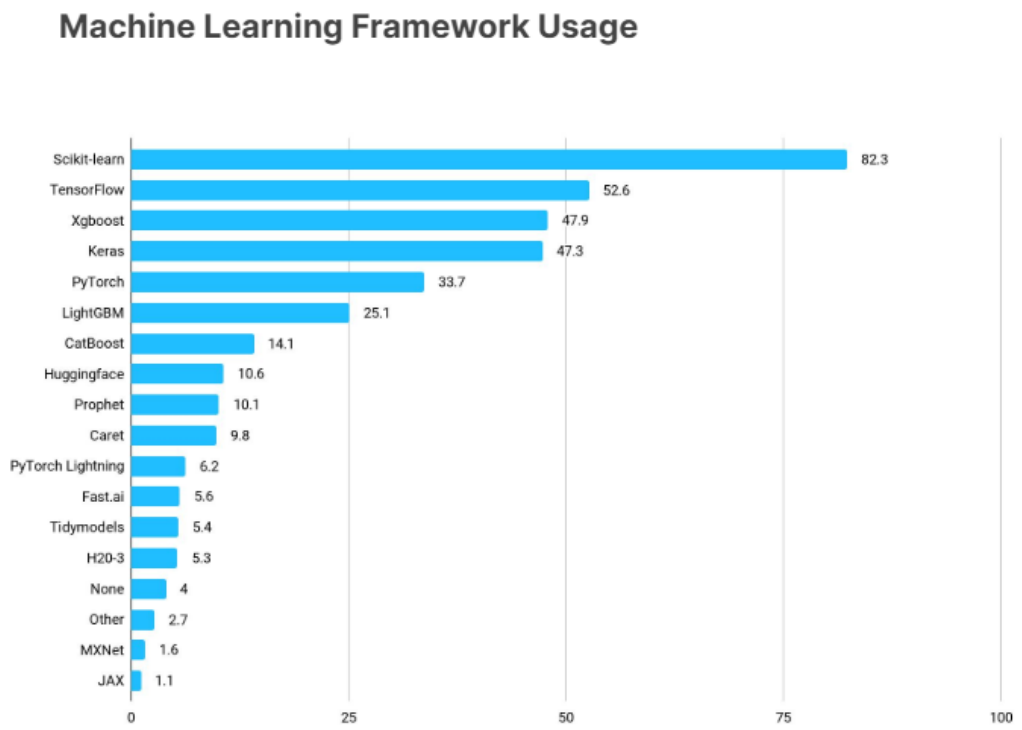
Kaggle: tree-based ensemble methods are still a popular choice

Machine Learning Frameworks

Python-based tools continue to dominate the machine learning frameworks.

Like last year, Scikit-learn, a swiss army knife applicable to most projects, is the top with over 80% of data scientists using it. TensorFlow and Keras, notably used in combination for deep learning, were each selected on about half of the data scientist surveys. Gradient boosting library xgboost is fourth, with about the same usage as 2020 and 2019.

The most popular of the new tools added to the survey this year is Huggingface reaching over 10%.



Kaggle | State of ML & Data Science 2021

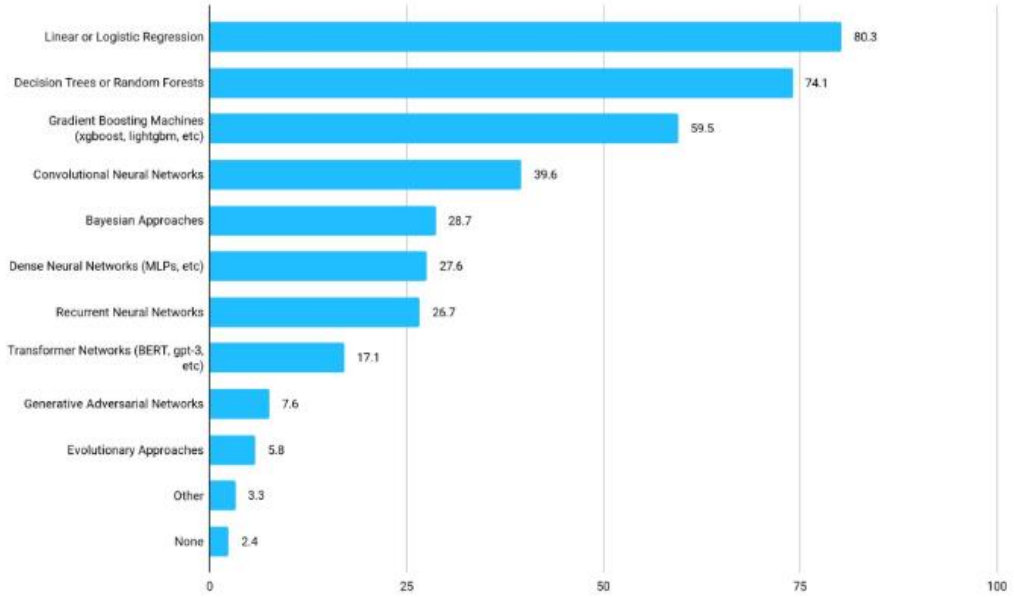
Kaggle: tree-based ensemble methods are still a popular choice

Methods & Algorithms

Like last year, the most commonly used algorithms were linear and logistic regression, followed closely by decision trees and random forests.

Of more complex methods, gradient boosting machines and convolutional neural networks were the most popular approaches.

Methods and Algorithms Usage



Technology 33

Kaggle | State of ML & Data Science 2021

Tree-based ensembling methods: Key takeaways

I	**Ensembling** allows taking care of the limitations that affect decision trees, namely, overfitting and high variance
II	**Bagging** and **random forest algorithms** inject randomness into learning by bootstrapping training data and de-correlating trees
III	**Boosting methods**, such as AdaBoost and gradient boosting machines, train “weak learners” sequentially and combine them additively to return a “strong learner”
IV	Research is actively investigating why **tree-based ensemble methods exhibit performance comparable with the one of deep learning methods** on tabular data. Tree-based methods are still actively used by data scientists across the world.

Lastly, a Google Colab notebook to test what we learned today

```
ensemble.ipynb
```

Reflection

“*Looking back on all the time we spent together…*”*

*La Bouche, Be My Lover (1995) - https://www.youtube.com/watch?v=ViP87WipSm0

Please consider these questions and answer them: We will comment them together

1	What was your understanding of machine learning before starting Block I and how have our lectures together changed that?
2	What was the most engaging/surprising fact that you learned during Block I?
3	What is the topic related to machine learning that is still the most unclear/mysterious to you?
4	Can you specify a business problem you would like to tackle? Which machine learning method would you like to use?
5	How would you continue learning machine learning now?

Andrea’s recommendations

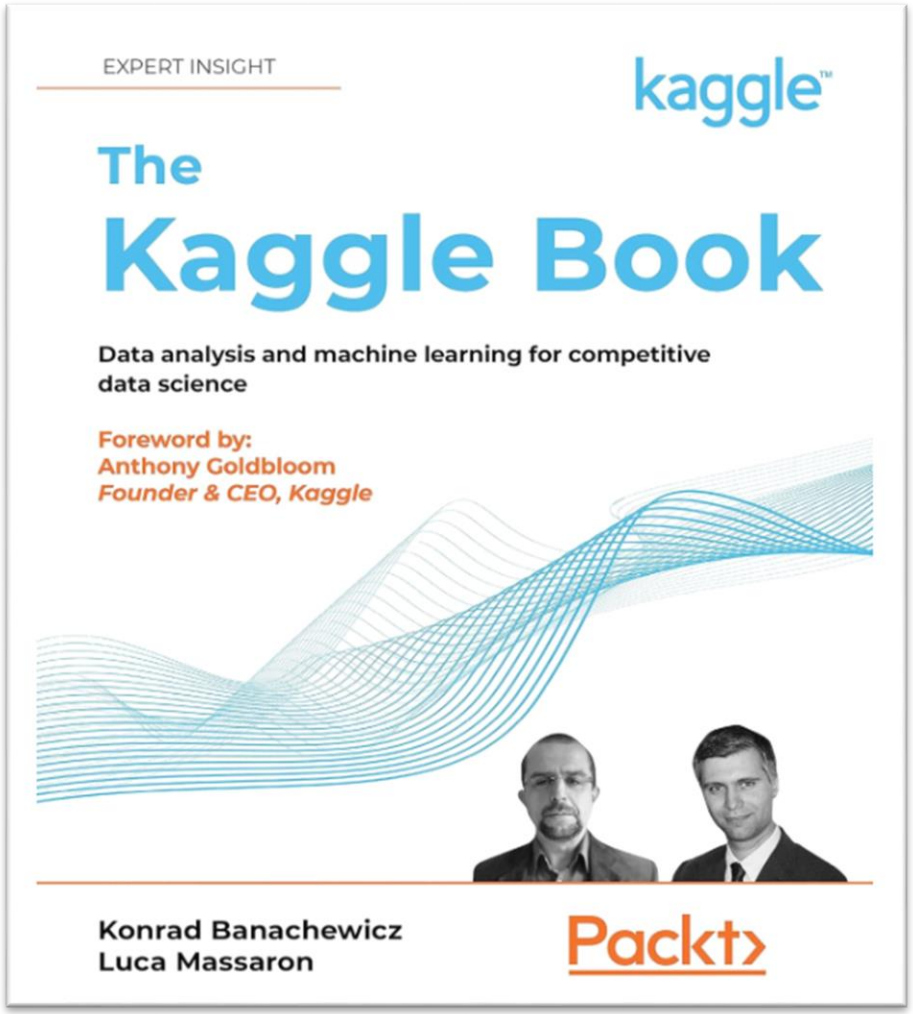
1. **You just started learning machine learning**: now, increase time spent on self-study and improve your current understanding of methods and algorithms

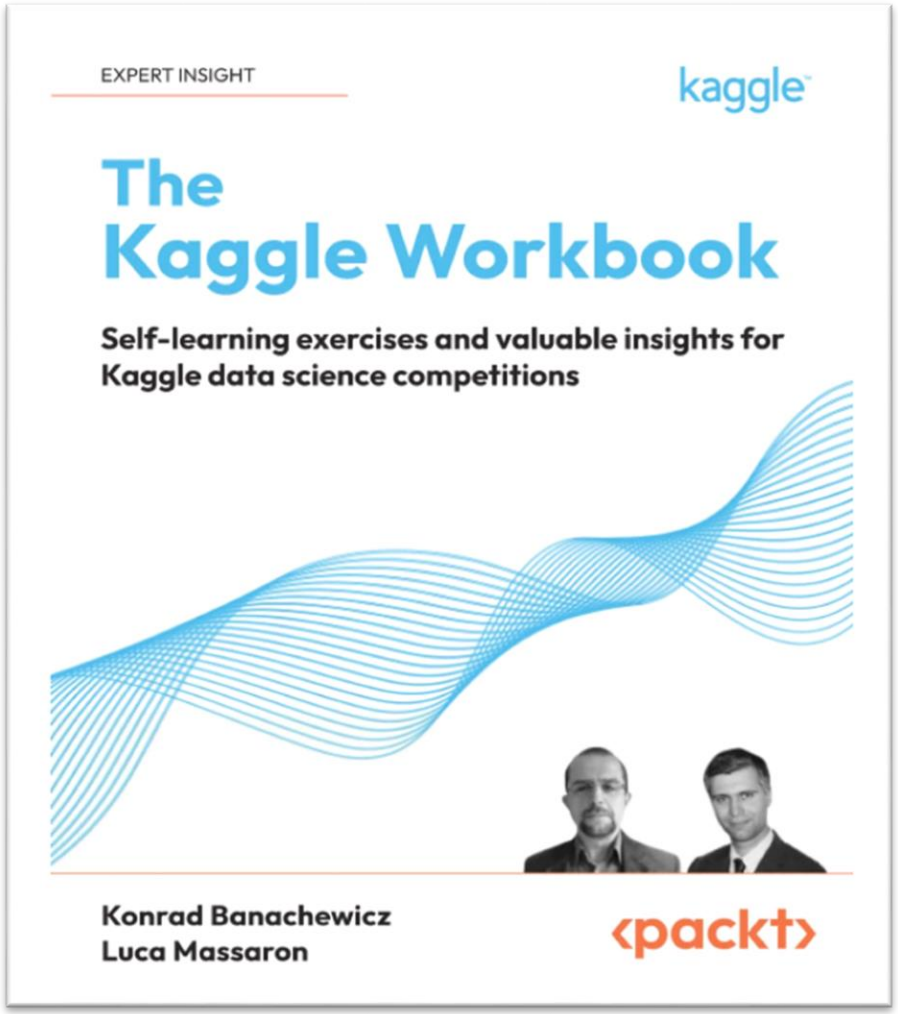
2. **Identify a problem you would like to address**, including a specific data modality such as text, images, or tabular data, and begin coding. Start simple!

3. Most probably, many other people have tried to solve your problem already: **get connected, stay updated**

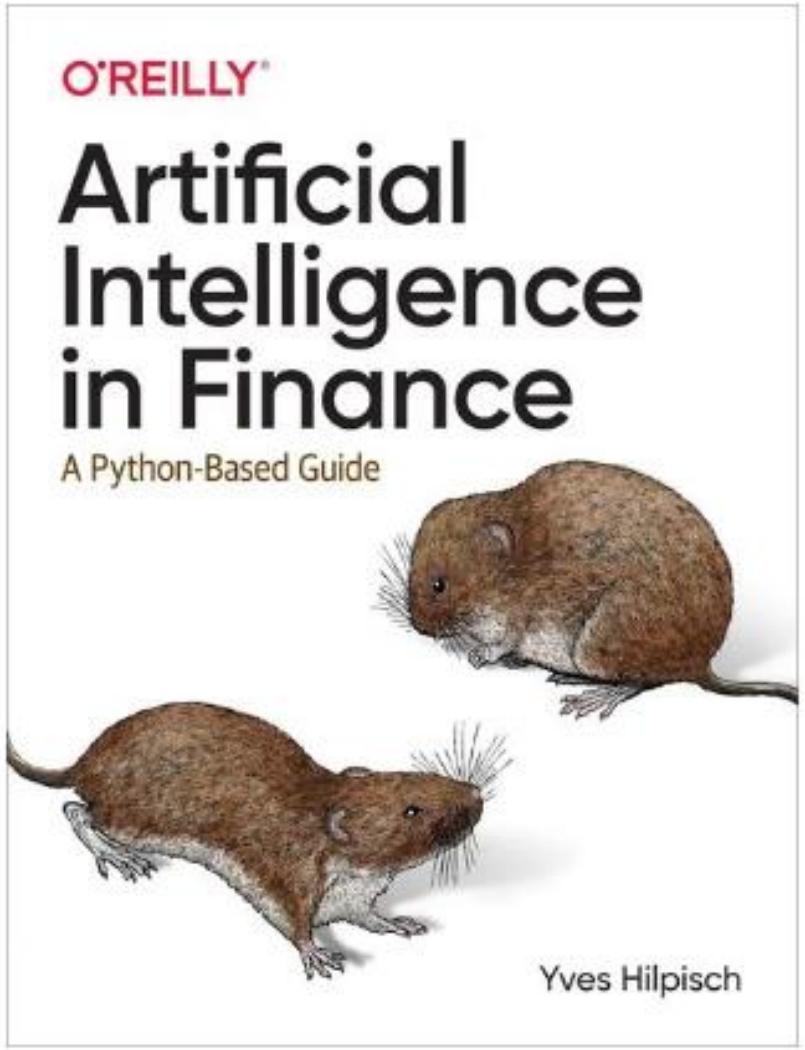
For those of you interested in machine learning, its models and algorithms

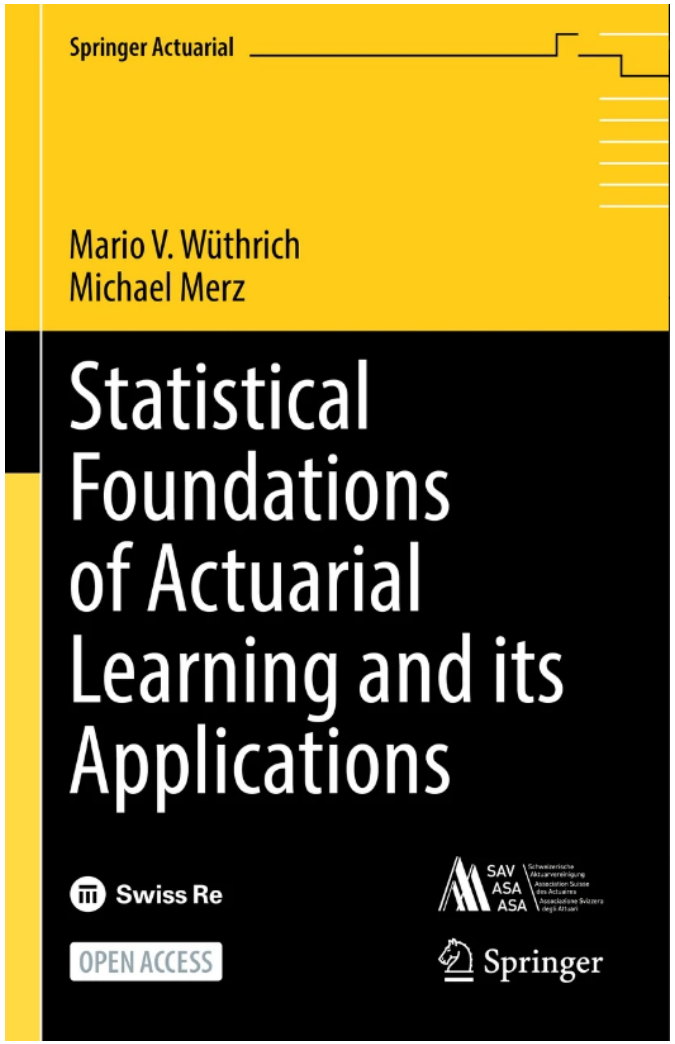
For those of you interested in competitive machine learning





For those of you interested in machine learning applications (finance and insurance)





After 7+1 lectures, 568 slides, 95 mini-exercises, and 7 Google Colab notebooks…

…this is the end of Block 1. Grazie per l' attenzione.