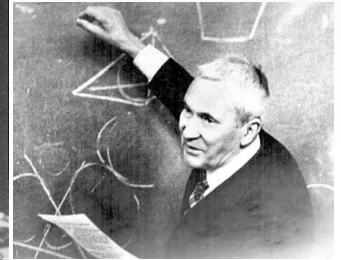
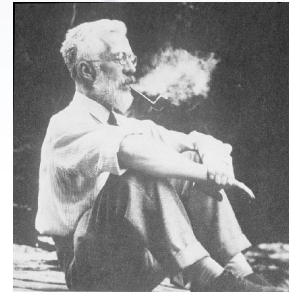
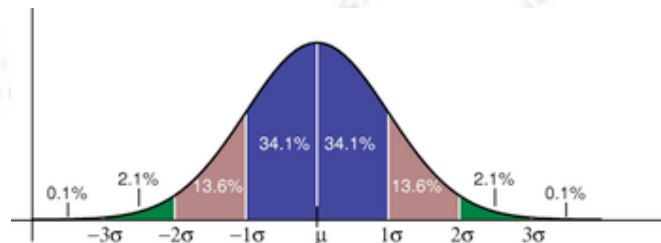


Big Data Analysis

Boosted Decision Trees (BDT) & Random Forests (RF)



Troels C. Petersen (NBI)



"Statistics is merely a quantisation of common sense - Machine Learning is a sharpening of it!"

Decision tree learning

“Tree learning comes closest to meeting the requirements for serving as an off-the-shelf procedure for data mining”,

because it:

- is invariant under scaling and various other transformations of feature values,
- is robust to discontinuous, categorical, and irrelevant features,
- produces inspectable models.

HOWEVER... they are seldom accurate (i.e. most performant)!

[Trevor Hastie, Prof. of Mathematics & Statistics, Stanford]

Decision Trees

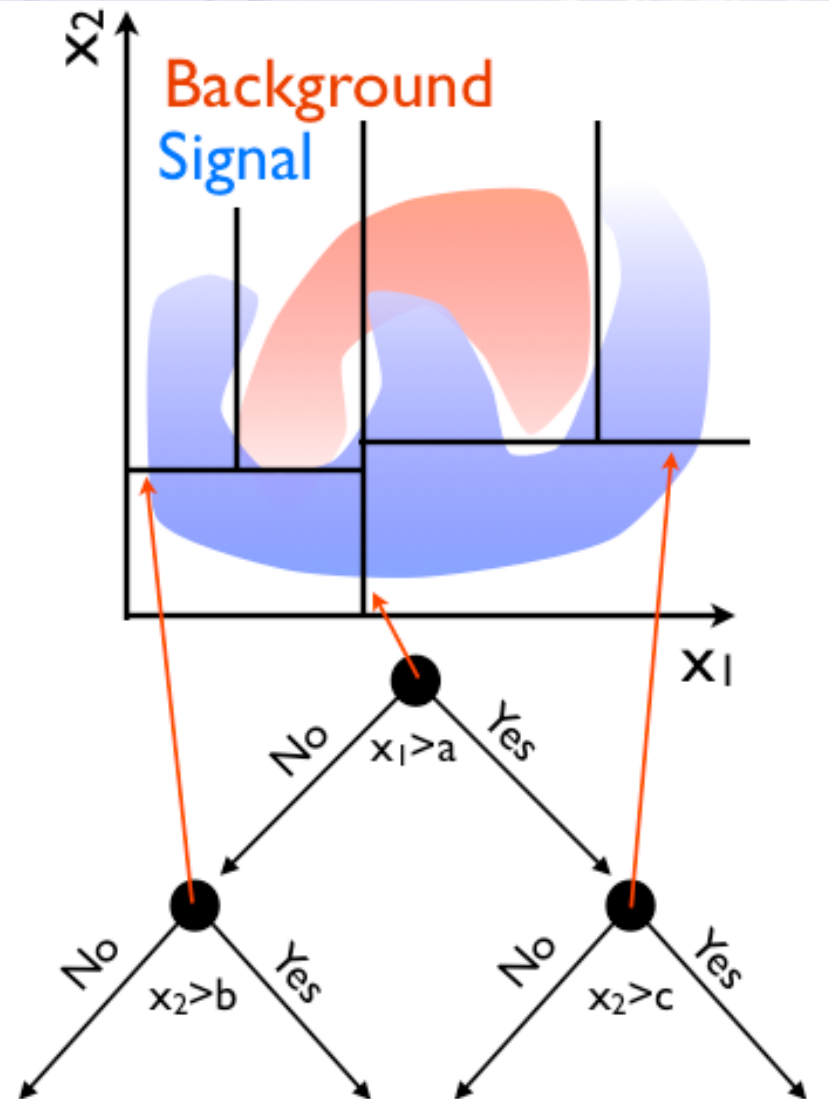
A decision tree divides the parameter space, starting with the maximal separation. In the end each part has a probability of being signal or background.

- Works in 95+% of all problems!
- Fully uses non-linear correlations.

But BDTs require a lot of data for training, and is sensitive to overtraining.

Overtraining can be reduced by limiting the number of nodes and number of trees.

Decision trees are from before 1980!!!



Boosting...

There is no reason, why you can not have more trees. Each tree is a simple classifier, but many can be combined!

To avoid N identical trees, one assigns a higher weight to events that are hard to classify, i.e. boosting:

First classifier

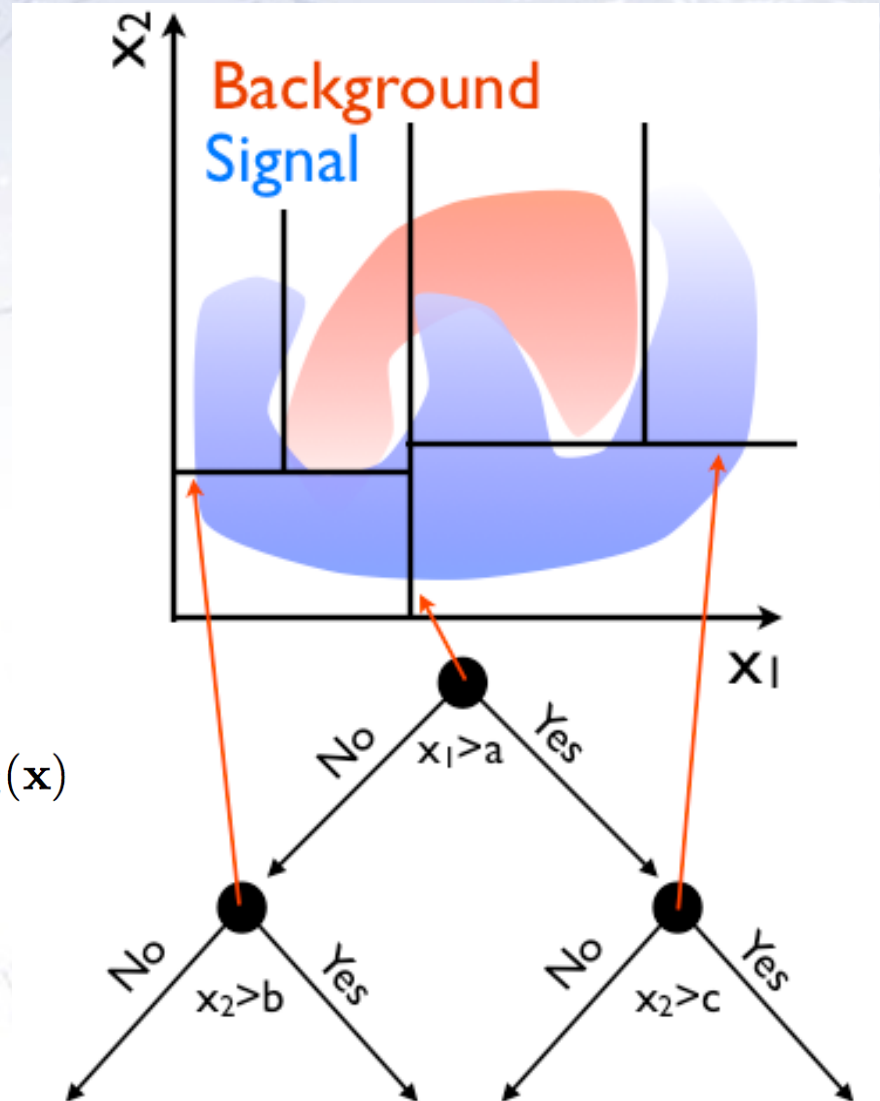
$$y_{\text{Boost}}(\mathbf{x}) = \frac{1}{N_{\text{collection}}} \cdot \sum_i^{N_{\text{collection}}} \ln(\alpha_i) \cdot h_i(\mathbf{x})$$

Parameters in event N

Boost weight $\alpha = \frac{1 - \text{err}}{\text{err}}$

Individual tree

Boosting is from 1997 (AdaBoost).



Boosting...

There is no reason, why you can not have more trees. Each tree is a simple classifier, but many

To avoid N iterations, assign a higher weight to misclassified entries, i.e. boost

Rerun...
increasing the weight of misclassified entries

First classifier

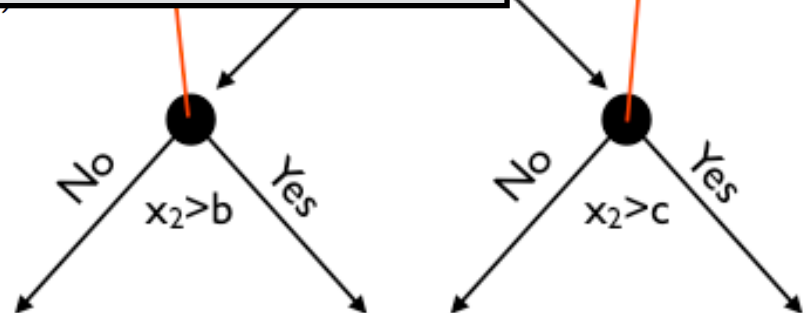
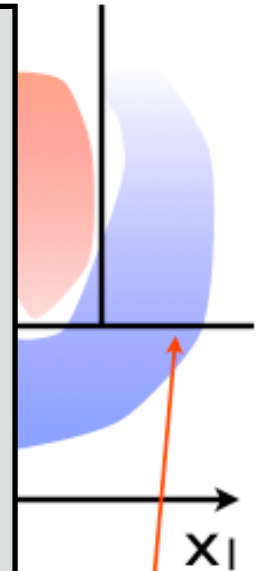
$$y_{\text{Boost}}(\mathbf{x}) = \frac{1}{N_{\text{collection}}} \sum_i$$

Parameters in event N

Individual tree

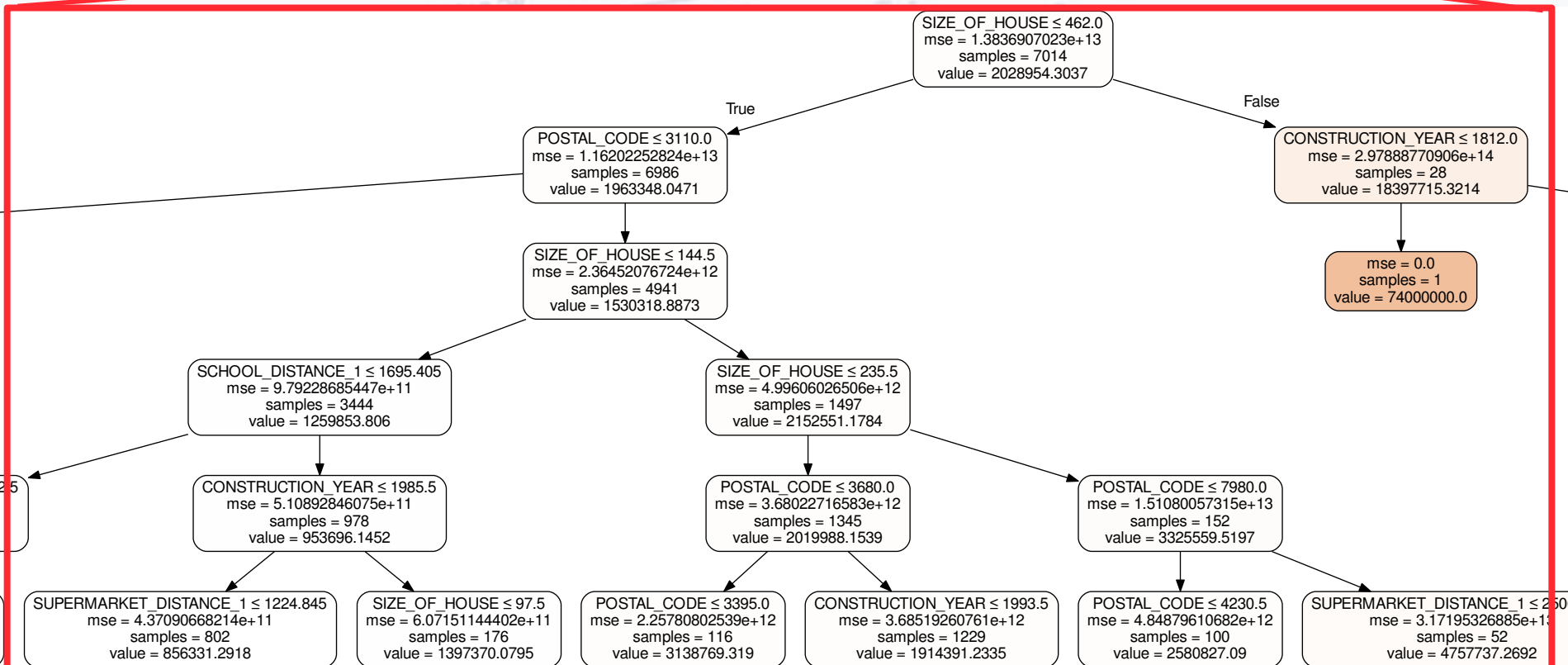
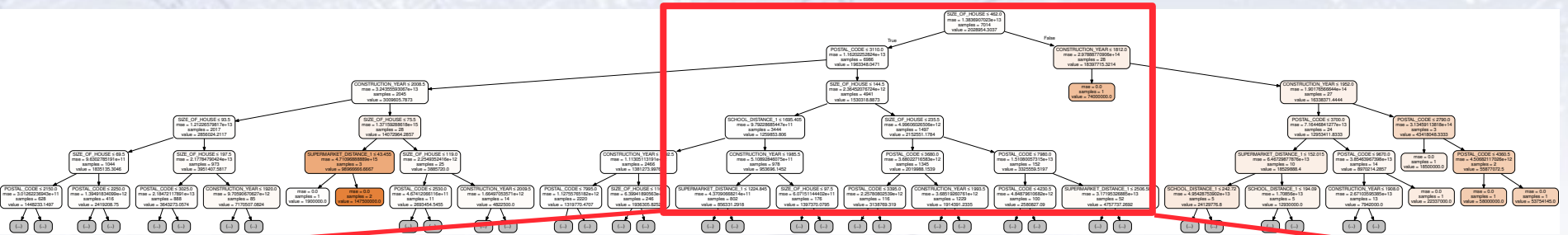
Boosting is from 1997 (AdaBoost).

x_2 ↑
Background



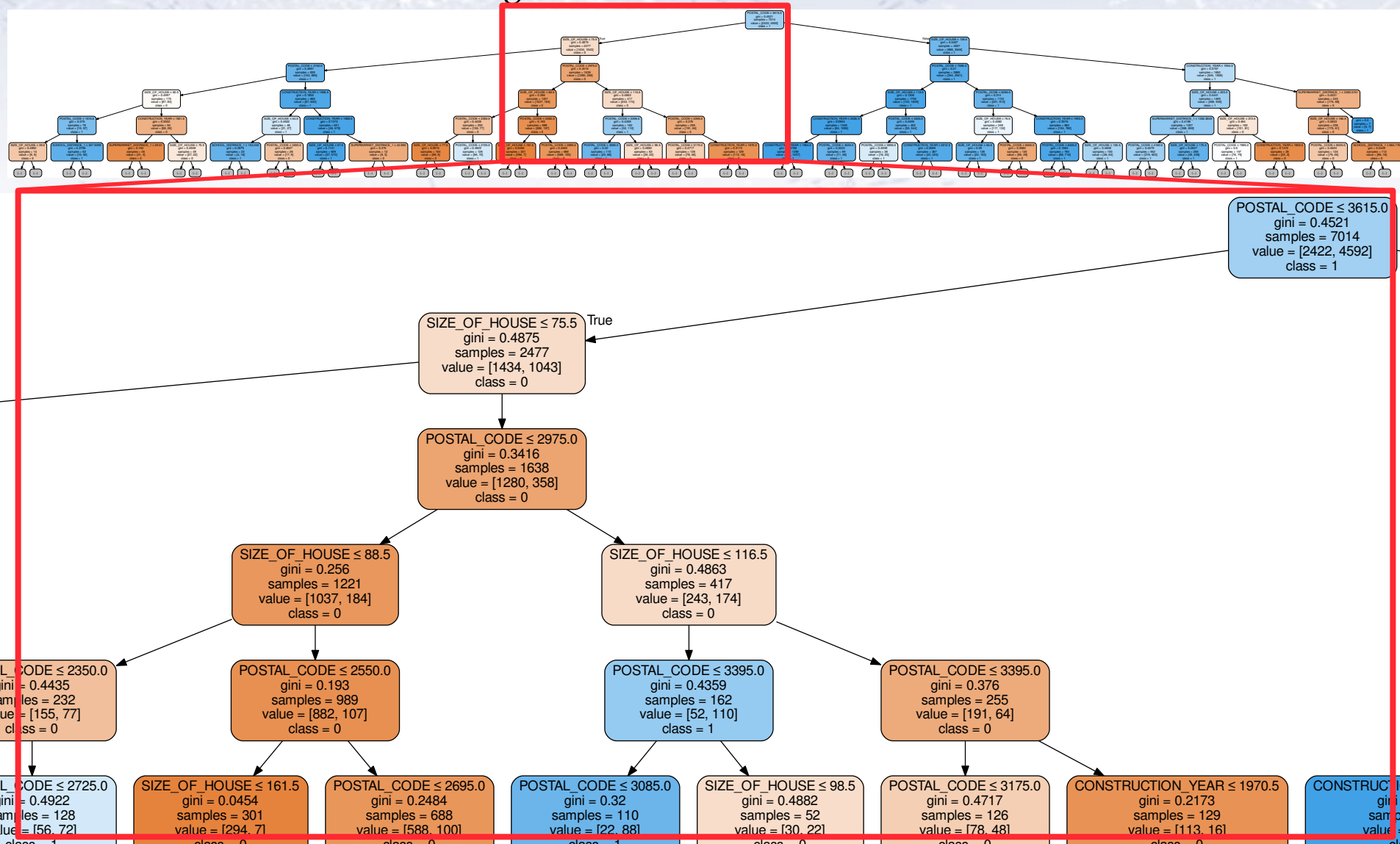
Housing Prices decision tree

Decision tree for estimating the price in the housing prices data set:



Housing Prices decision tree

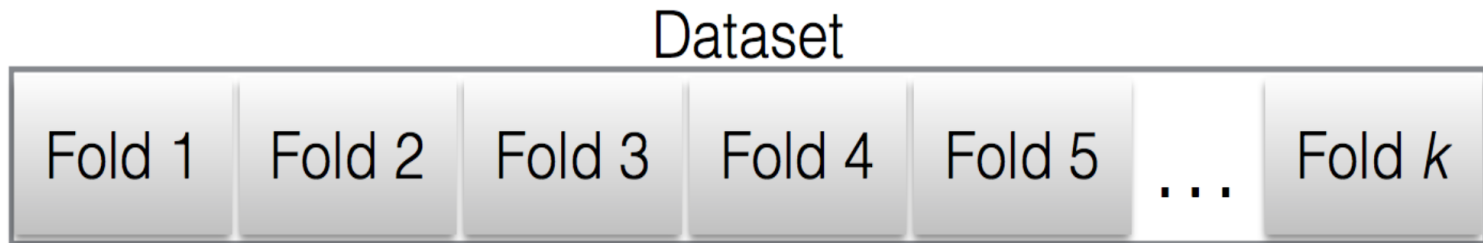
Decision tree for determining, if a house will be sold for more or less than 2Mkr.



Cross Validation

In case your data set is not that large (and perhaps anyhow), one can train on most of it, and then test on the remaining $1/k$ fraction.

This is then repeated for each fold... CPU-intensive, but smart for small data samples.



- ▶ Split the dataset into k randomly sampled independent subsets (folds).
- ▶ Train classifier with $k-1$ folds and test with remaining fold.
- ▶ Repeat k times.

Overtraining...

To test for overtraining, try to increase the number of parameters of your ML. If performance on Cross Validation (CV) sample drops, decrease complexity!



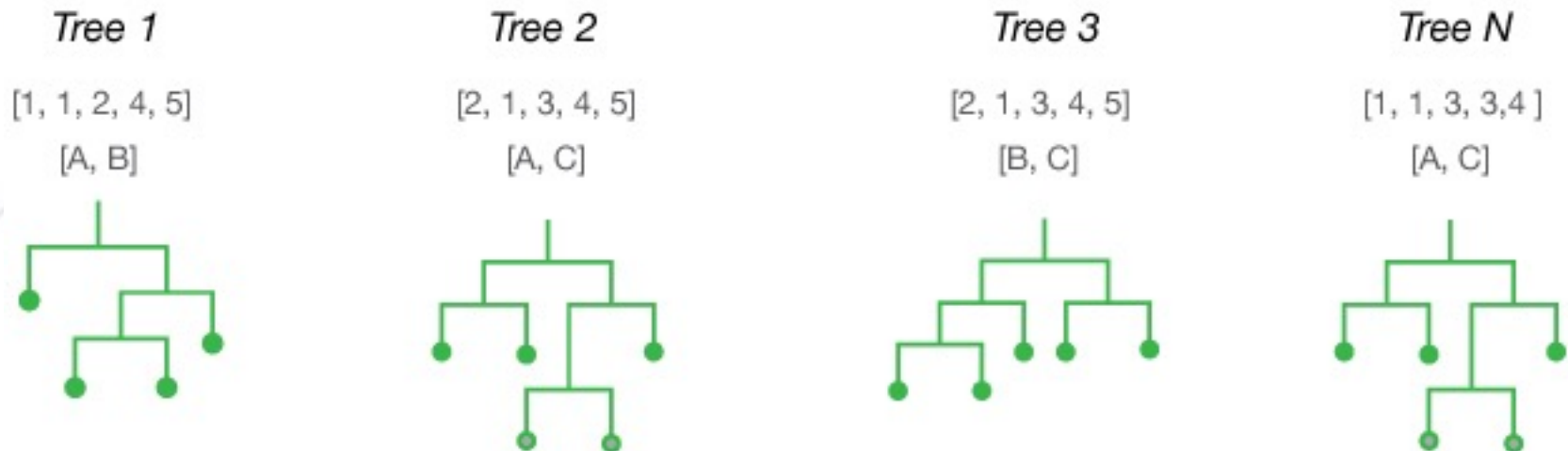
Some overtraining is good!

Random Forests

The many trees in a (forest of) decision trees increases the power of the decision tree algorithm.

To classify a new object from an input vector, give the input vector to each each of the trees in the forest. Each tree gives a classification, and we say the tree "votes" for that class. The forest chooses the classification having the most votes (over all the trees in the forest).

However, in (boosted) decision trees, the output is correlated, which leads to a decreased performance. The solution is to train on a Random Forest!



Random Forests

Each tree is grown as follows:

- If the number of cases in the training set is N , sample N cases at random - but with replacement. This sample will be the training set for growing the tree.
- If there are M input variables, a number $m \ll M$ is specified such that at each node, m variables are selected at random out of the M and the best split on these m is used to split the node. The value of m is held constant.
- Each tree is grown to the largest extent possible. There is no pruning.

The forest error rate depends on two things:

- The correlation between any two trees in the forest. Increasing the correlation increases the forest error rate.
- The strength of each individual tree in the forest. A tree with a low error rate is a strong classifier. Increasing the strength of the individual trees decreases the forest error rate.

Reducing m reduces both the correlation and the strength. Increasing it increases both. Somewhere in between is an "optimal" range of m - usually quite wide. This is the only adjustable parameter to which random forests is somewhat sensitive.

Random Forests

Features of Random Forests:

- It is unexcelled in accuracy among current algorithms.
- It runs efficiently on large data bases.
- It can handle thousands of input variables without variable deletion.
- It gives estimates of what variables are important in the classification.
- It has an effective method for estimating missing data and maintains accuracy when a large proportion of the data are missing.
- It has methods for balancing error in class population unbalanced data sets.
- It computes proximities between pairs of cases that can be used in clustering, locating outliers, or (by scaling) give interesting views of the data.
- The capabilities of the above can be extended to unlabeled data, leading to unsupervised clustering, data views and outlier detection.
- It offers an experimental method for detecting variable interactions.

For these reasons, the Random Forest algorithm has lately been in vogue.



XGboost - a neat little story!

The HiggsML Kaggle Challenge

CERN analyses its data using a vast array of ML methods. CERN is thus part of the community that developpes ML!

After 20 years of using Machine Learning it has now become very widespread (NN, BDT, Random Forest, etc.)

A prime example was the Kaggle “HiggsML Challenge”. Most popular challenge of its time! (1785 teams, 6517 downloads, 35772 solutions, 136 forums)



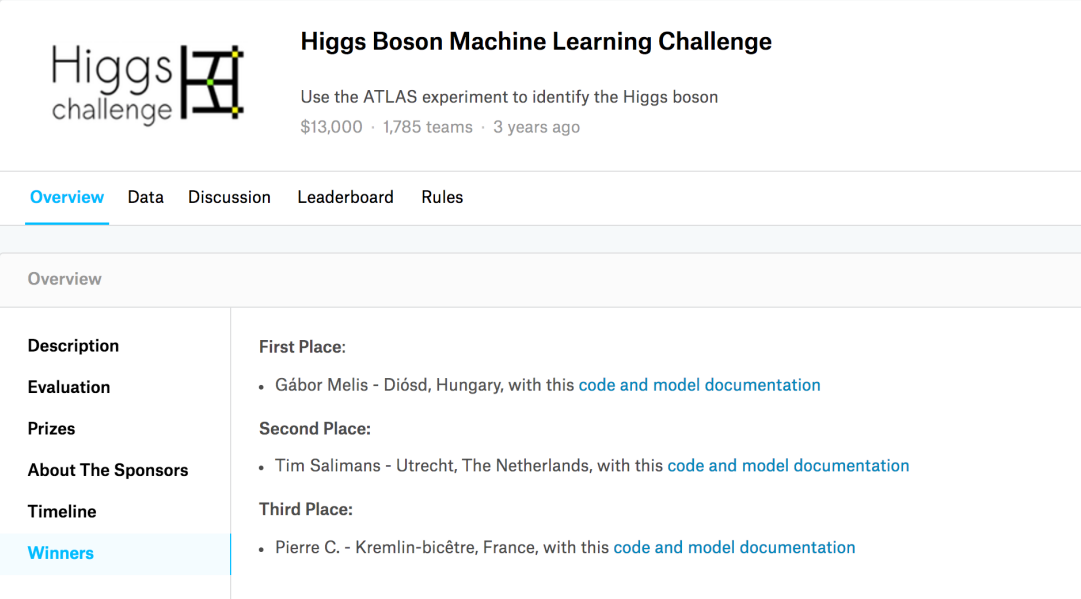
XGBoost history

History [\[edit \]](#)

XGBoost initially started as a research project by Tianqi Chen^[8] as part of the Distributed (Deep) Machine Learning Community (DMLC) group. Initially, it began as a terminal application which could be configured using a libsvm configuration file. After winning the Higgs Machine Learning Challenge, it became well known in the ML competition circles. Soon after, the Python and R packages were built and now it has packages for many other languages like Julia, Scala, Java, etc. This brought the library to more developers and became popular among the [Kaggle](#) community where it has been used for a large number of competitions.^[7]

While Tianqi Chen did not win himself, he provided a method used by about half of the teams, the second place among them!

For this, he got a special award and XGBoost became instantly known in the community.



The screenshot shows the Kaggle page for the Higgs Boson Machine Learning Challenge. The page title is "Higgs Boson Machine Learning Challenge" and the subtitle is "Use the ATLAS experiment to identify the Higgs boson". The prize is "\$13,000" and it was held "3 years ago" with "1,785 teams". The page has tabs for "Overview", "Data", "Discussion", "Leaderboard", and "Rules". The "Overview" tab is selected. The "Overview" section is divided into two columns. The left column contains links for "Description", "Evaluation", "Prizes", "About The Sponsors", "Timeline", and "Winners". The right column contains the following information:

- First Place:**
 - Gábor Melis - Diósd, Hungary, with this [code](#) and [model documentation](#)
- Second Place:**
 - Tim Salimans - Utrecht, The Netherlands, with this [code](#) and [model documentation](#)
- Third Place:**
 - Pierre C. - Kremlin-bicêtre, France, with this [code](#) and [model documentation](#)

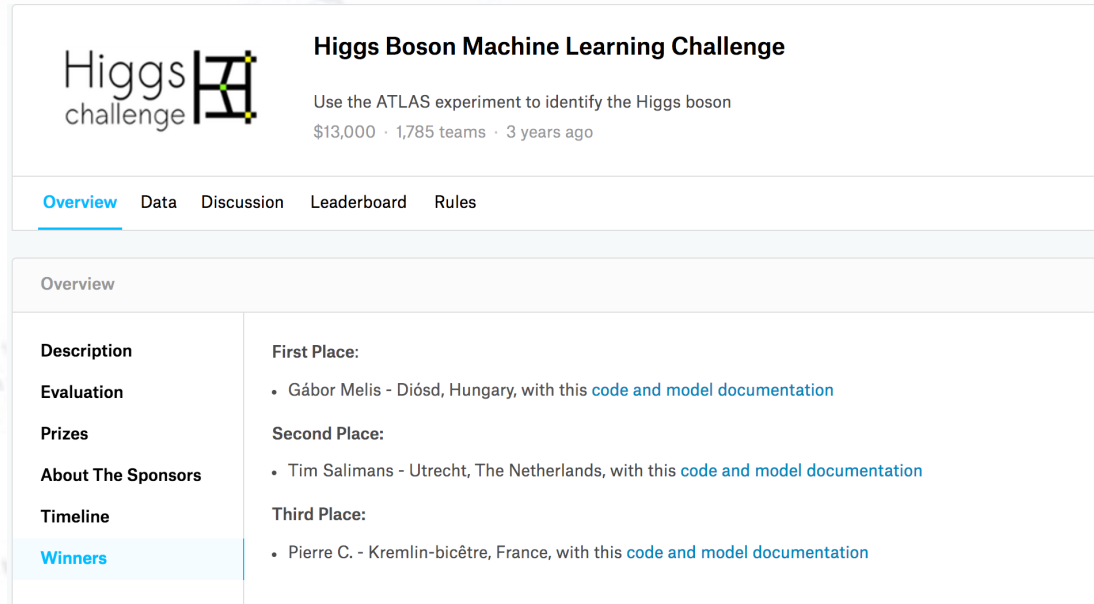
XGBoost history

History [[edit](#)]

XGBoost initially started as a research project by Tianqi Chen^[8] as part of the Distributed (Deep) Machine Learning Community (DMLC) group. Initially, it began as a terminal application which could be configured using a libsvm configuration file. After winning the Higgs Machine Learning Challenge, it became well known in the ML competition circles. Soon after, the Python and R packages were built and now it has packages for many other languages like Julia, Scala, Java, etc. This brought the library to more developers and became popular among the [Kaggle](#) community where it has been used for a large number of competitions.^[7]

While Tianqi Chen did not win himself, he provided a method used by about half of the teams, the second place among them!

For this, he got a special award and XGBoost became instantly known in the community.



The screenshot shows the Kaggle page for the Higgs Boson Machine Learning Challenge. The page title is "Higgs Boson Machine Learning Challenge" and the description is "Use the ATLAS experiment to identify the Higgs boson". The challenge was created 3 years ago with a prize of \$13,000 and 1,785 teams. The page has tabs for Overview, Data, Discussion, Leaderboard, and Rules. The Overview section is active and shows the following information:

Category	Details
Description	First Place:
Evaluation	• Gábor Melis - Diósd, Hungary, with this code and model documentation
Prizes	Second Place:
About The Sponsors	• Tim Salimans - Utrecht, The Netherlands, with this code and model documentation
Timeline	Third Place:
Winners	• Pierre C. - Kremlin-bicêtre, France, with this code and model documentation

XGBoost algorithm

The algorithm is documented on the arXiv: 1603.02754

XGBoost: A Scalable Tree Boosting System

Tianqi Chen
University of Washington
tqchen@cs.washington.edu

Carlos Guestrin
University of Washington
guestrin@cs.washington.edu

ABSTRACT

Tree boosting is a highly effective and widely used machine learning method. In this paper, we describe a scalable end-to-end tree boosting system called XGBoost, which is used widely by data scientists to achieve state-of-the-art results on many machine learning challenges. We propose a novel sparsity-aware algorithm for sparse data and weighted quantile sketch for approximate tree learning. More importantly, we provide insights on cache access patterns, data compression and sharding to build a scalable tree boosting system. By combining these insights, XGBoost scales beyond billions of examples using far fewer resources than existing systems.

Keywords

Large-scale Machine Learning

problems. Besides being used as a stand-alone predictor, it is also incorporated into real-world production pipelines for ad click through rate prediction [15]. Finally, it is the de-facto choice of ensemble method and is used in challenges such as the Netflix prize [3].

In this paper, we describe XGBoost, a scalable machine learning system for tree boosting. The system is available as an open source package². The impact of the system has been widely recognized in a number of machine learning and data mining challenges. Take the challenges hosted by the machine learning competition site Kaggle for example. Among the 29 challenge winning solutions³ published at Kaggle's blog during 2015, 17 solutions used XGBoost. Among these solutions, eight solely used XGBoost to train the model, while most others combined XGBoost with neural nets in ensembles. For comparison, the second most popular method, deep neural nets, was used in 11 solutions. The success

XGBoost algorithm

The algorithm is an extension of the decision tree idea (tree boosting), using regression trees with weighted quantiles and being “sparsity aware” (i.e. knowing about lacking entries and low statistics areas of phase space).

Unlike decision trees, each regression tree contains a continuous score on each leaf:

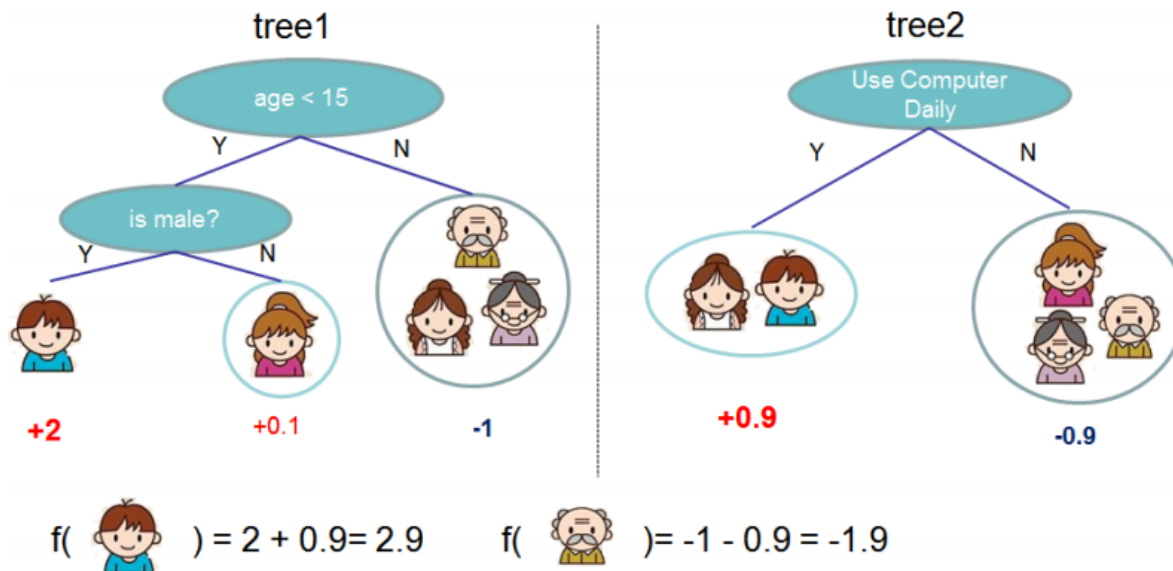
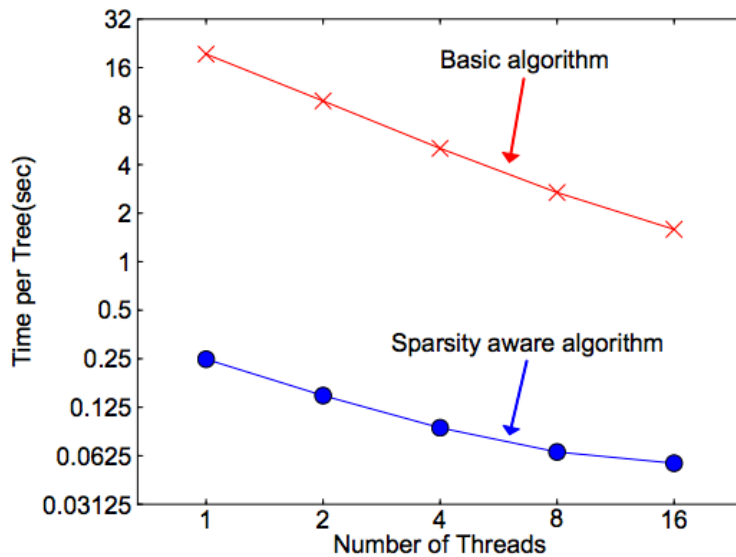


Figure 1: Tree Ensemble Model. The final prediction for a given example is the sum of predictions from each tree.

XGBoost algorithm

The method's speed is partly due to an approximate but fast algorithm to find the best splits.



Algorithm 1: Exact Greedy Algorithm for Split Finding

Input: I , instance set of current node

Input: d , feature dimension

$gain \leftarrow 0$

$G \leftarrow \sum_{i \in I} g_i, H \leftarrow \sum_{i \in I} h_i$

for $k = 1$ **to** m **do**

$G_L \leftarrow 0, H_L \leftarrow 0$

for j **in** $sorted(I, \text{by } \mathbf{x}_{jk})$ **do**

$G_L \leftarrow G_L + g_j, H_L \leftarrow H_L + h_j$

$G_R \leftarrow G - G_L, H_R \leftarrow H - H_L$

$score \leftarrow \max(score, \frac{G_L^2}{H_L + \lambda} + \frac{G_R^2}{H_R + \lambda} - \frac{G^2}{H + \lambda})$

end

end

Output: Split with max score

Algorithm 2: Approximate Algorithm for Split Finding

for $k = 1$ **to** m **do**

 Propose $S_k = \{s_{k1}, s_{k2}, \dots, s_{kl}\}$ by percentiles on feature k .

 Proposal can be done per tree (global), or per split(local).

end

for $k = 1$ **to** m **do**

$G_{kv} \leftarrow \sum_{j \in \{j | s_{k,v} \geq \mathbf{x}_{jk} > s_{k,v-1}\}} g_j$

$H_{kv} \leftarrow \sum_{j \in \{j | s_{k,v} \geq \mathbf{x}_{jk} > s_{k,v-1}\}} h_j$

end

Follow same step as in previous section to find max score only among proposed splits.

XGBoost algorithm

In order to “punish” complexity, the cost-function has a regularised term also:

$$\mathcal{L}(\phi) = \sum_i l(\hat{y}_i, y_i) + \sum_k \Omega(f_k)$$

$$\text{where } \Omega(f) = \gamma T + \frac{1}{2} \lambda \|w\|^2$$

Table 1: Comparison of major tree boosting systems.

System	exact greedy	approximate global	approximate local	out-of-core	sparsity aware	parallel
XGBoost	yes	yes	yes	yes	yes	yes
pGBRT	no	no	yes	no	no	yes
Spark MLlib	no	yes	no	no	partially	yes
H2O	no	yes	no	no	partially	yes
scikit-learn	yes	no	no	no	no	no
R GBM	yes	no	no	no	partially	no

XGBoost

As it turns out, XGBoost is not only very performant but also very fast...

The most important factor behind the success of XGBoost is its scalability in all scenarios. The system runs more than ten times faster than existing popular solutions on a single machine and scales to billions of examples in distributed or memory-limited settings. The scalability of XGBoost is due to several important systems and algorithmic optimizations.

But this will of course only last for so long - new algorithms see the light of day every week... day?

XGBoost

As it turns out, XGBoost is not only very performant but also very fast...

The most important factor behind the success of XGBoost is its scalability in all scenarios. The system runs more than ten times faster than existing popular solutions on a single machine and scales to billions of examples in distributed or memory-limited settings. The scalability of XGBoost is due to several important systems and algorithmic optimizations.

But this will of course only last for so long - new algorithms see the light of day every week... day?

— — — — — — — — — — shortly after — — — — — — — — — —

Meanwhile, LightGBM has seen the light of day, and it is even faster...

Which algorithm takes the crown: Light GBM vs XGBOOST?