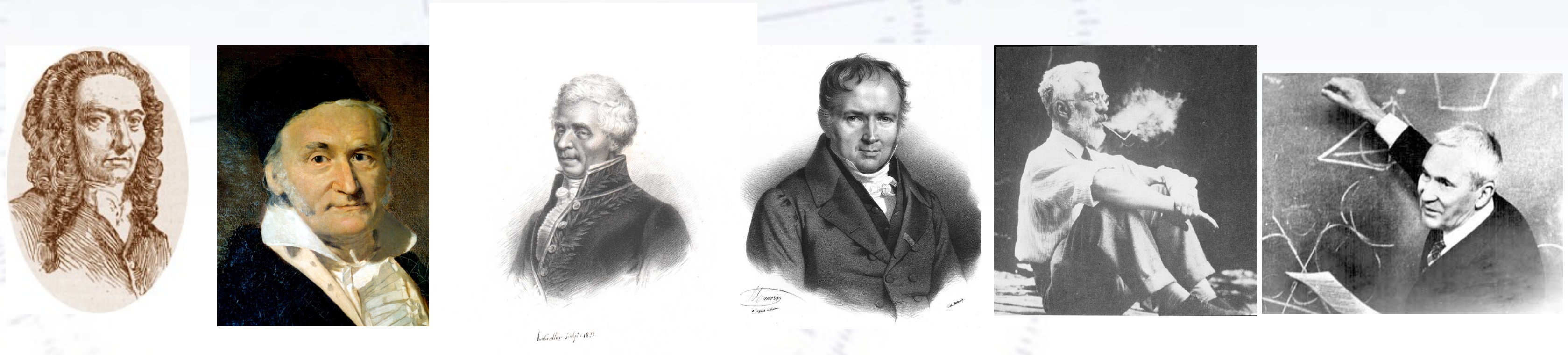
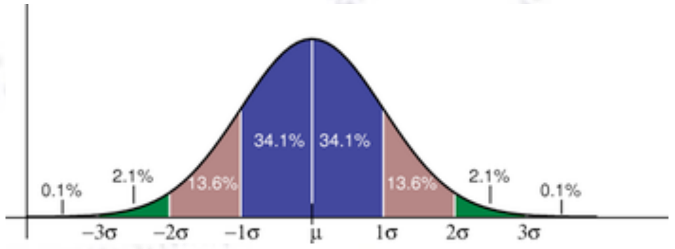
Applied Statistics

Multivariate Analysis - part II



Troels C. Petersen (NBI)



"Statistics is merely a quantisation of common sense"

Fisher Discriminant

You want to separate two types/classes (A and B) of events using several measurements.

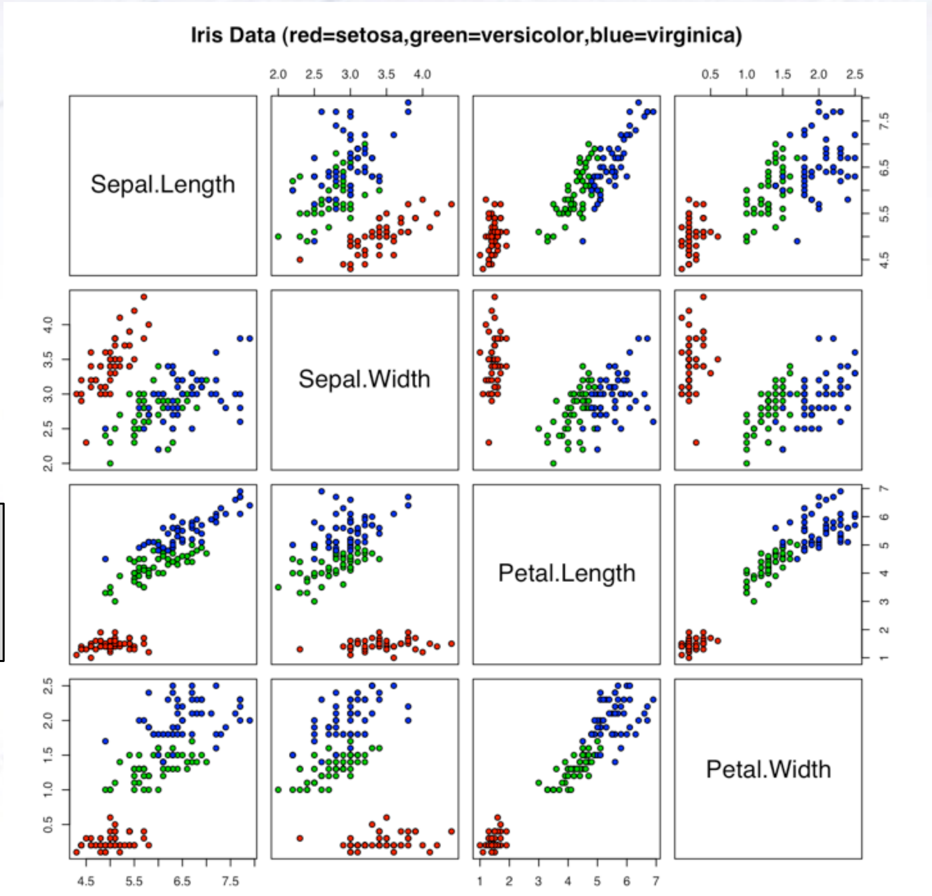
Q: How to combine the variables?
A: Use the Fisher Discriminant:

$$\boxed{\mathcal{F} = w_0 + \vec{w} \cdot \vec{x}}$$

Q: How to choose the values of w?
A: Inverting the covariance matrices:

$$\boxed{\vec{w} = (\mathbf{\Sigma}_A + \mathbf{\Sigma}_B)^{-1} \; (\vec{\mu}_A - \vec{\mu}_B)}$$

This can be calculated analytically, and incorporates the linear correlations into the separation capability.



Fisher Discriminant

The details of the formula are outlined below:

For each input variable (x), you calculate the mean (μ), and form a vector of these.

$$\vec{w} = (\mathbf{\Sigma}_A + \mathbf{\Sigma}_B)^{-1} \ (\vec{\mu}_A - \vec{\mu}_B)$$

Using the input variables (x), you calculate the covariance matrix (Σ) for each species (A/B), add these and invert.

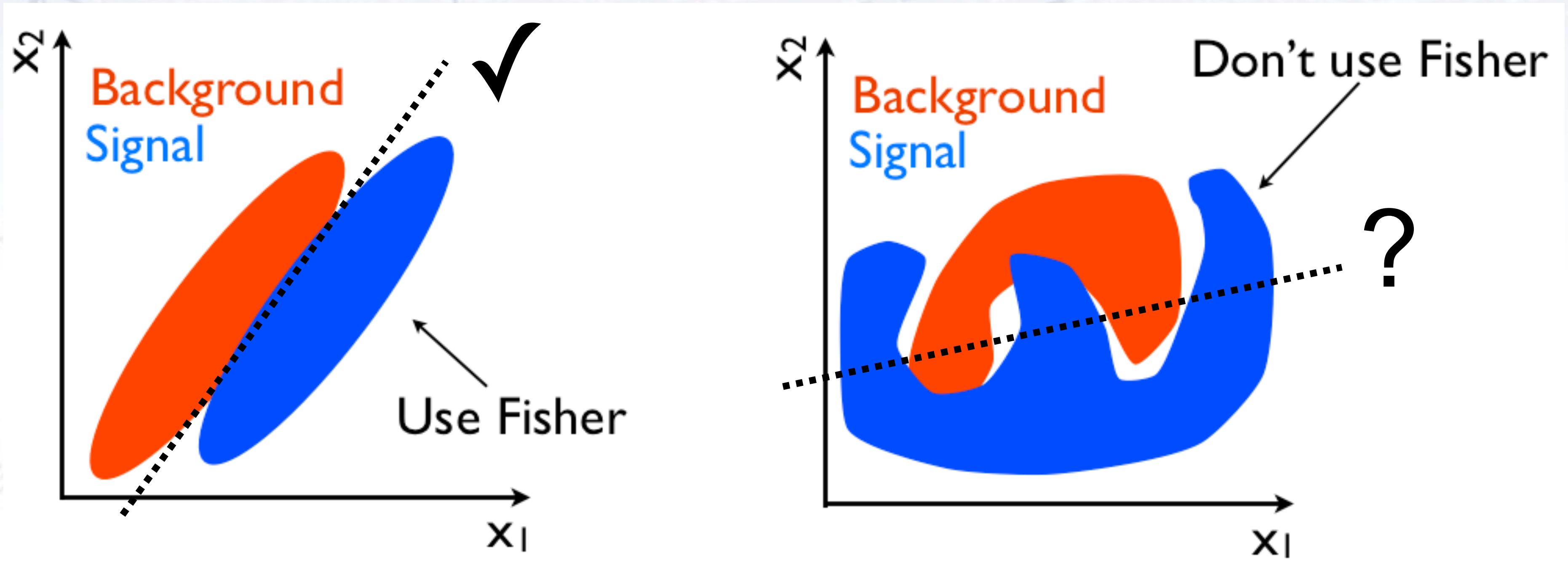
Given weights (w), you take your input variables (x) and combine them linearly as follows:

$$\mathcal{F} = w_0 + \vec{w} \cdot \vec{x}$$

F is what you base your decision on.

Non-linear MVAs

While the Fisher Discriminant uses all separations and **linear correlations**, it does not perform optimally, when there are **non-linear correlations** present:



If the PDFs of signal and background are known, then one can use a likelihood.

But this is **very rarely** the case, and therefore more "tough" methods are needed...

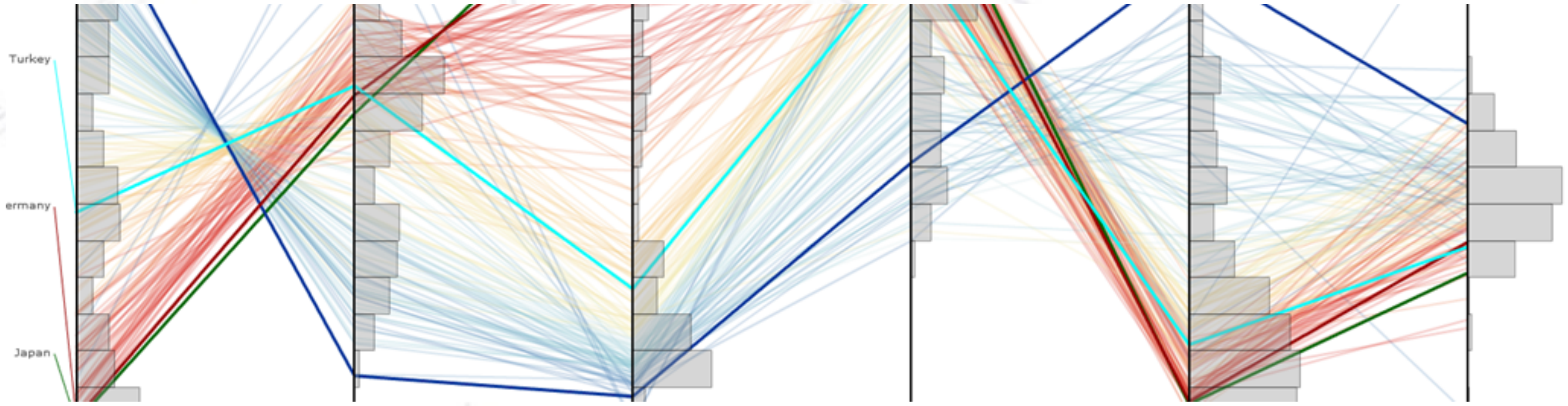
Todays goal: Introduction

MultiVariate Analysis (MVA) is a **huge subject**, and it is **impossible** to go into any detail in one day.

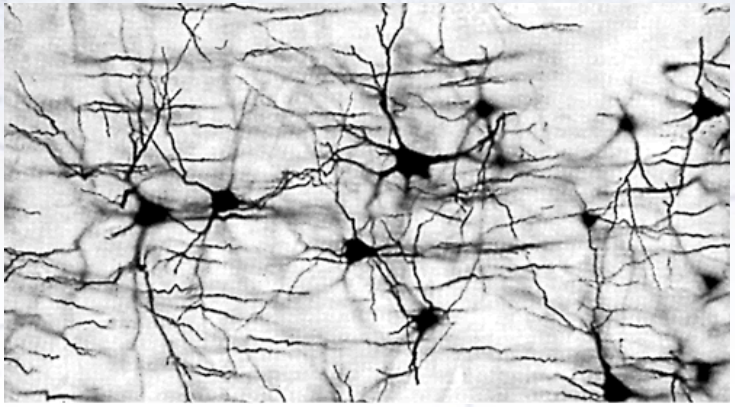
The goal of todays exercise is to:

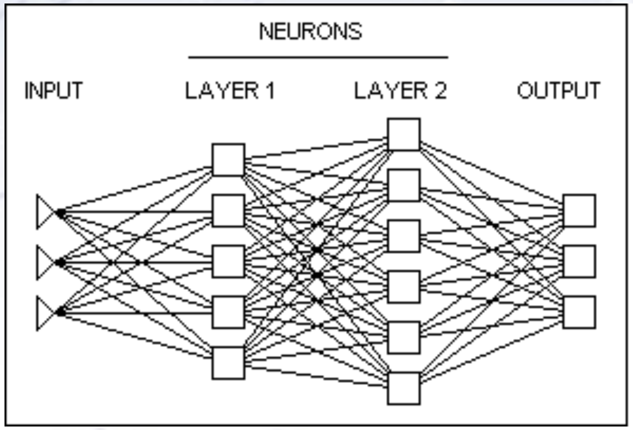
- Give you an introduction to more advanced MVA methods, i.e. Machine Learning.
- Be able to recognise problems, where MVA is applicable.
- Wet your appetite for using Machine Learning on data.

So let us dive into the world of extracting knowledge from information.



Neural Networks (NN)





In machine learning and related fields, artificial neural networks (ANNs) are computational models inspired by an animal's central nervous systems (in particular the brain) which is capable of ***machine learning*** *as well as* ***pattern recognition****.*

Neural networks *have been used to solve a wide variety of tasks that are hard to solve using ordinary rule-based programming, including* ***computer vision*** *and* ***speech recognition****.*

[Wikipedia, Introduction to Artificial Neural Network]

Neural Networks

Neural Networks combine the input variables using a "activation" function s(x) to assign, if the variable indicates signal or background.

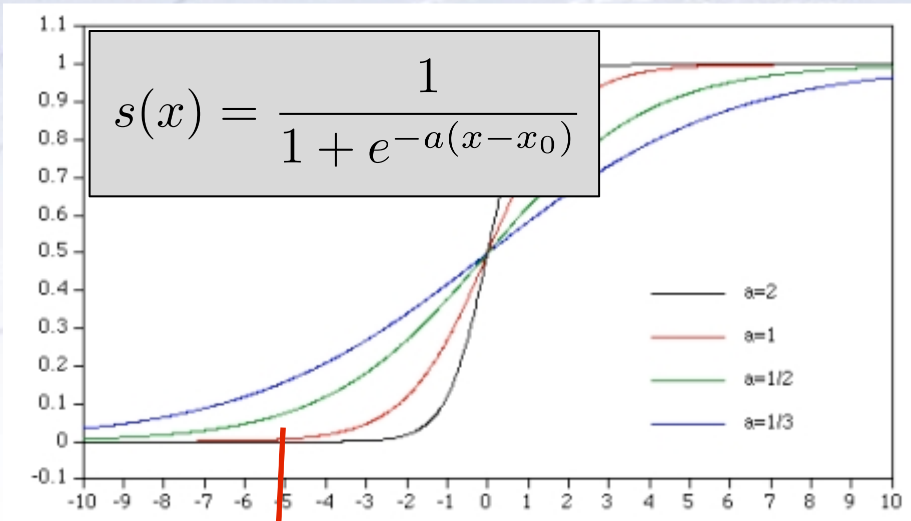
The simplest is a single layer perceptron:

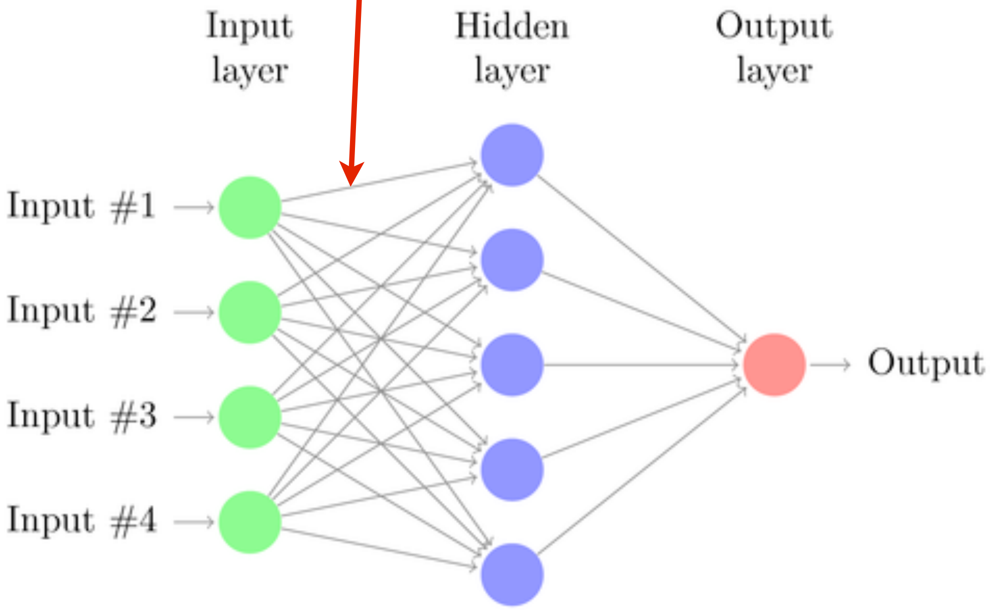
$$t(x) = s\left(a_0 + \sum a_i x_i\right)$$

This can be generalized to a multilayer perceptron:

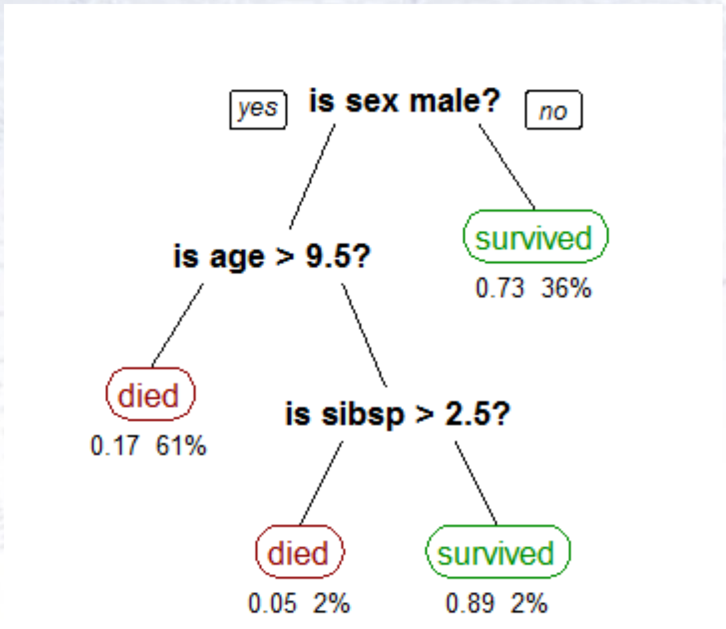
$$t(x) = s\left(a_i + \sum a_i h_i(x)\right)$$
$$h_i(x) = s\left(w_{i0} + \sum w_{ij} x_j\right)$$

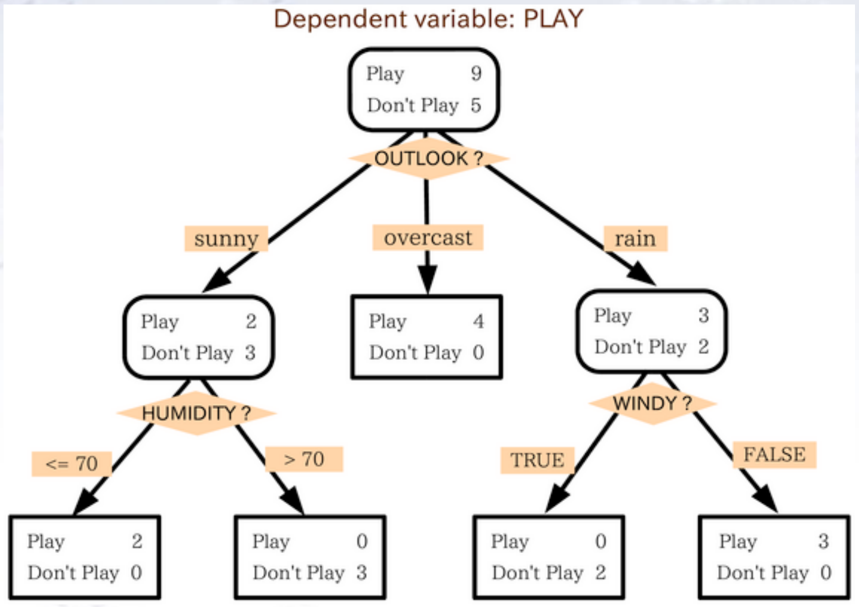
Activation function can be any sigmoid function.





Boosted Decision Trees (BDT)





Decision tree learning *uses a* ***decision tree*** *as a* ***predictive model*** *which maps observations about an item to conclusions about the item's target value. It is one of the predictive modelling approaches used in* ***statistics****,* ***data mining*** *and* ***machine learning****.*

[Wikipedia, Introduction to Decision Tree Learning]

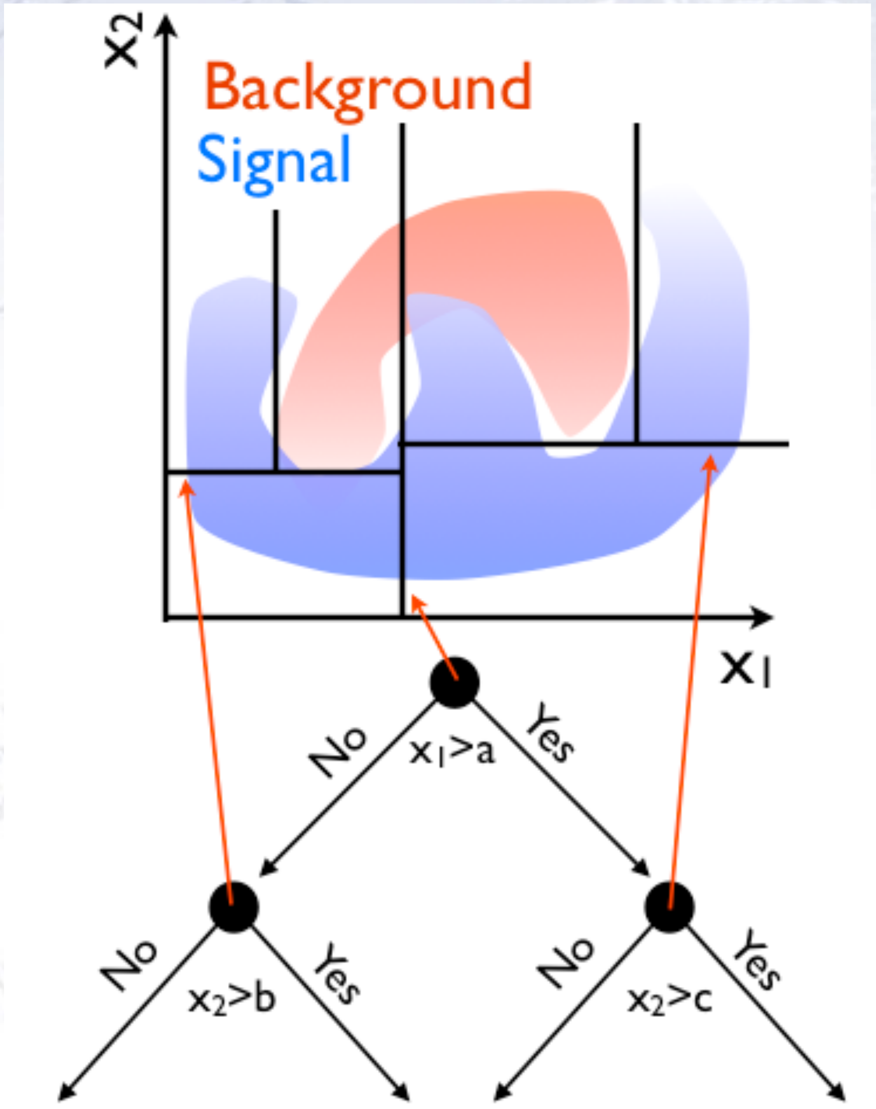
Boosted Decision Trees

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 (see next slide).

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



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:

Boost weight

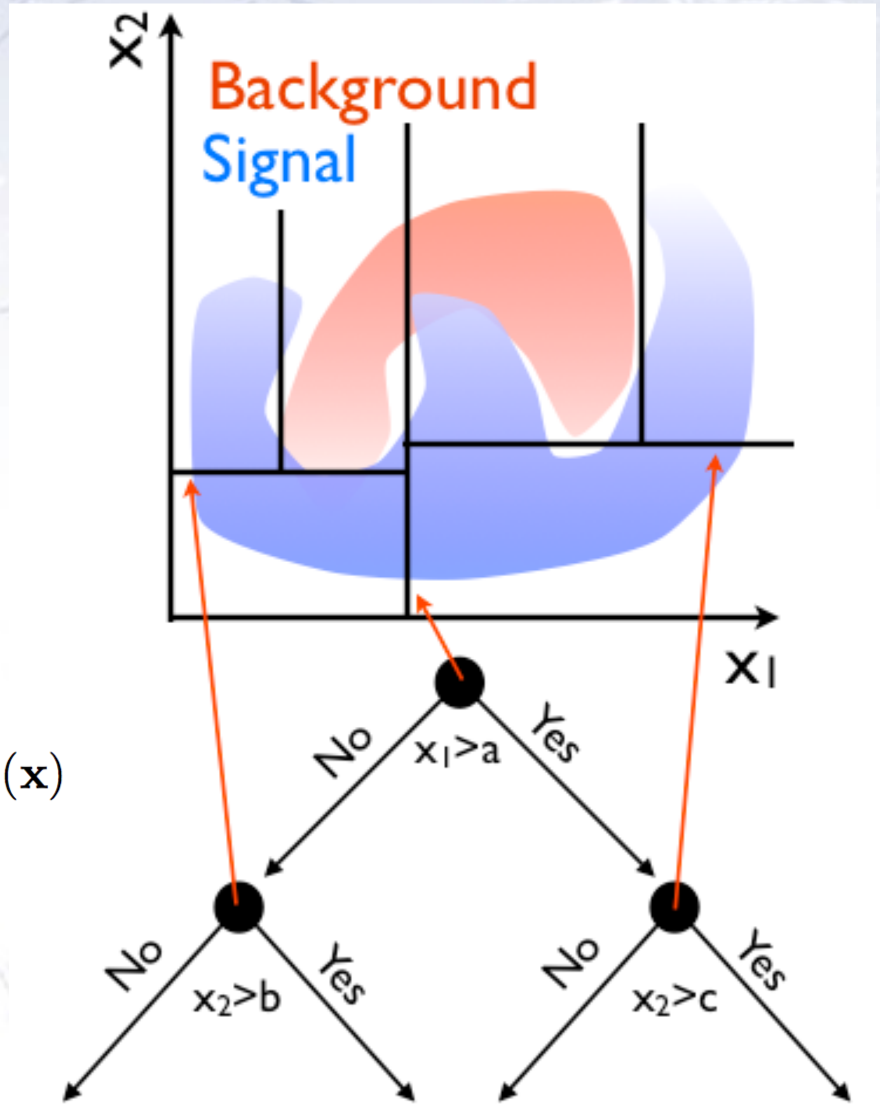
$$\alpha = \frac{1-\mathrm{err}}{\mathrm{err}}$$

First classifier

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

Parameters in event N

Individual tree



Boosting...

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

To avoid N iden a higher weight to classify, i.e. b

Rerun...

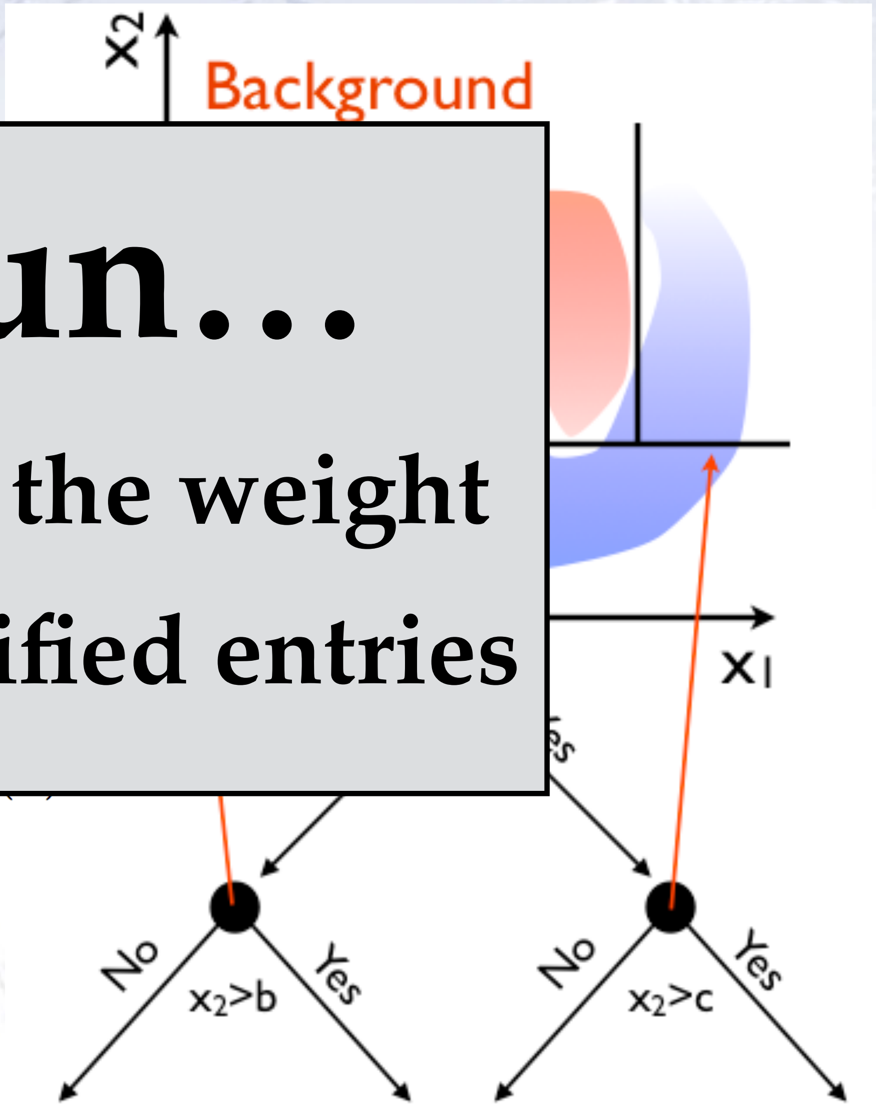
increasing the weight of misclassified entries

First classifier

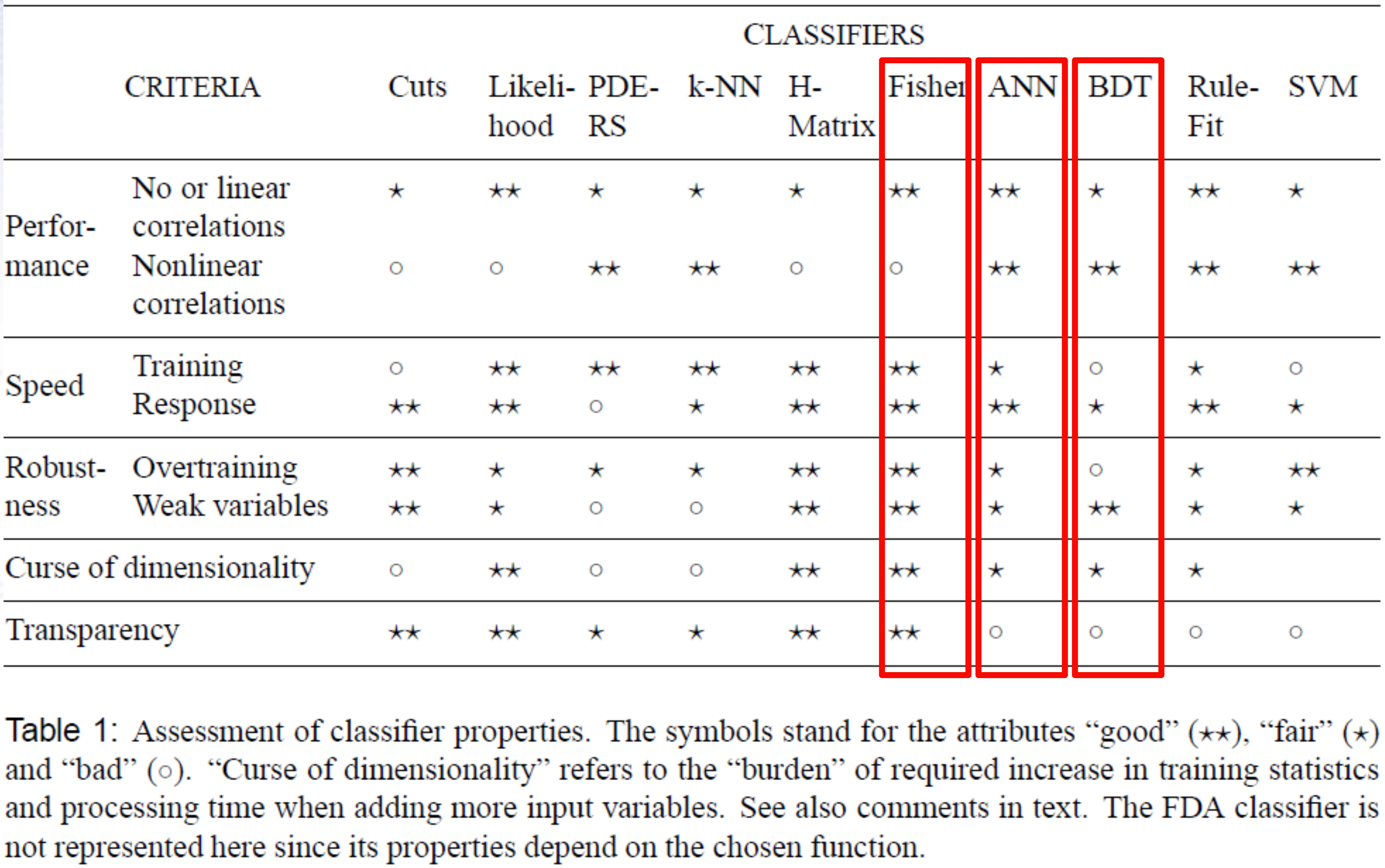
$$y_{\text{Boost}}(\mathbf{x}) = \frac{}{N_{\text{collection}}} \sum_{i}$$

Parameters in event N

Individual tree



Method's (dis-)advantages

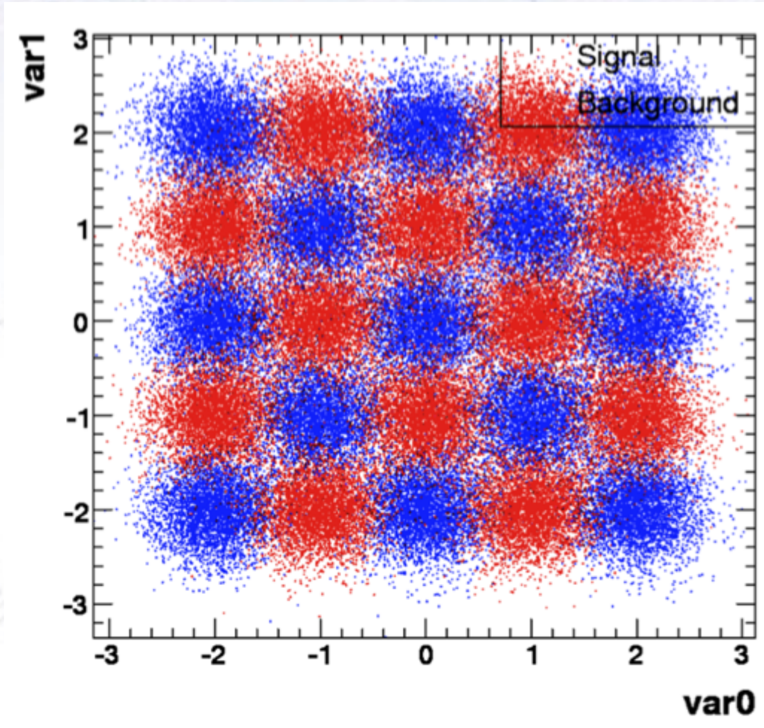


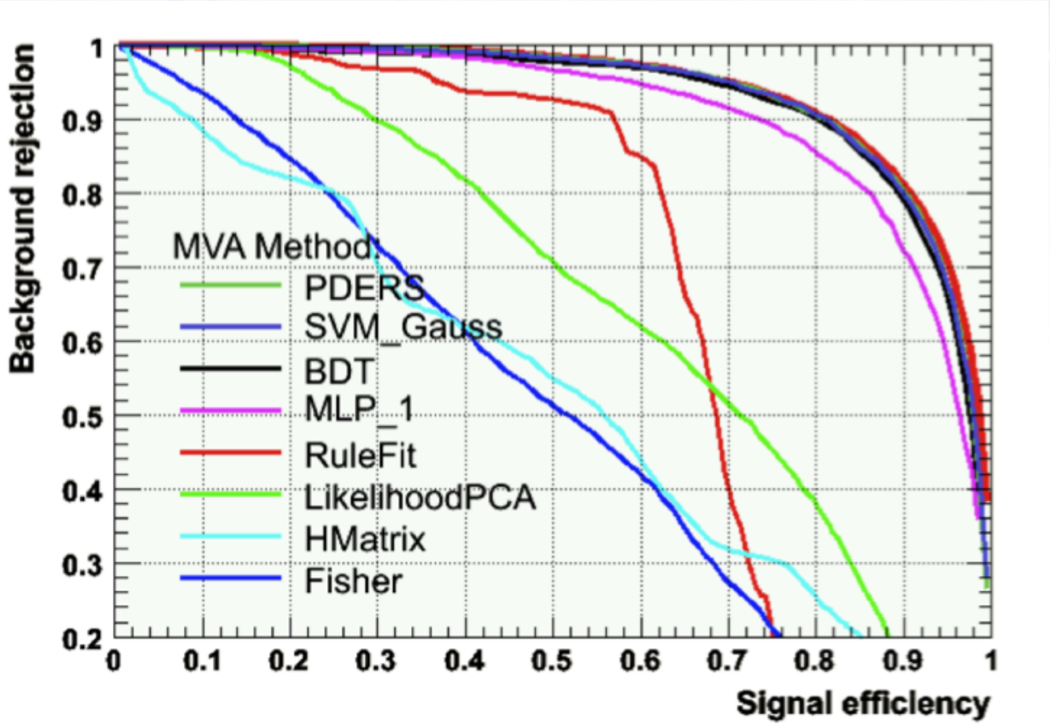
		CLASSIFIERS									
CRITERIA		Cuts	Likeli-hood	PDE-RS	k-NN	H-Matrix	Fisher	ANN	BDT	Rule-Fit	SVM
Perfor-mance	No or linear correlations	⋆	⋆⋆	⋆	⋆	⋆	⋆⋆	⋆⋆	⋆	⋆⋆	⋆
	Nonlinear correlations	○	○	⋆⋆	⋆⋆	○	○	⋆⋆	⋆⋆	⋆⋆	⋆⋆
Speed	Training	○	⋆⋆	⋆⋆	⋆⋆	⋆⋆	⋆⋆	⋆	○	⋆	○
	Response	⋆⋆	⋆⋆	○	⋆	⋆⋆	⋆⋆	⋆⋆	⋆	⋆⋆	⋆
Robust-ness	Overtraining	⋆⋆	⋆	⋆	⋆	⋆⋆	⋆⋆	⋆	○	⋆	⋆⋆
	Weak variables	⋆⋆	⋆	○	○	⋆⋆	⋆⋆	⋆	⋆⋆	⋆	⋆
Curse of dimensionality		○	⋆⋆	○	○	⋆⋆	⋆⋆	⋆	⋆	⋆	
Transparency		⋆⋆	⋆⋆	⋆	⋆	⋆⋆	⋆⋆	○	○	○	○

Table 1: Assessment of classifier properties. The symbols stand for the attributes "good" (⋆⋆), "fair" (⋆) and "bad" (○). "Curse of dimensionality" refers to the "burden" of required increase in training statistics and processing time when adding more input variables. See also comments in text. The FDA classifier is not represented here since its properties depend on the chosen function.

Example of method comparison

Left figure shows the distribution of signal and background used for test.
Right figure shows the resulting separation using various MVA methods.

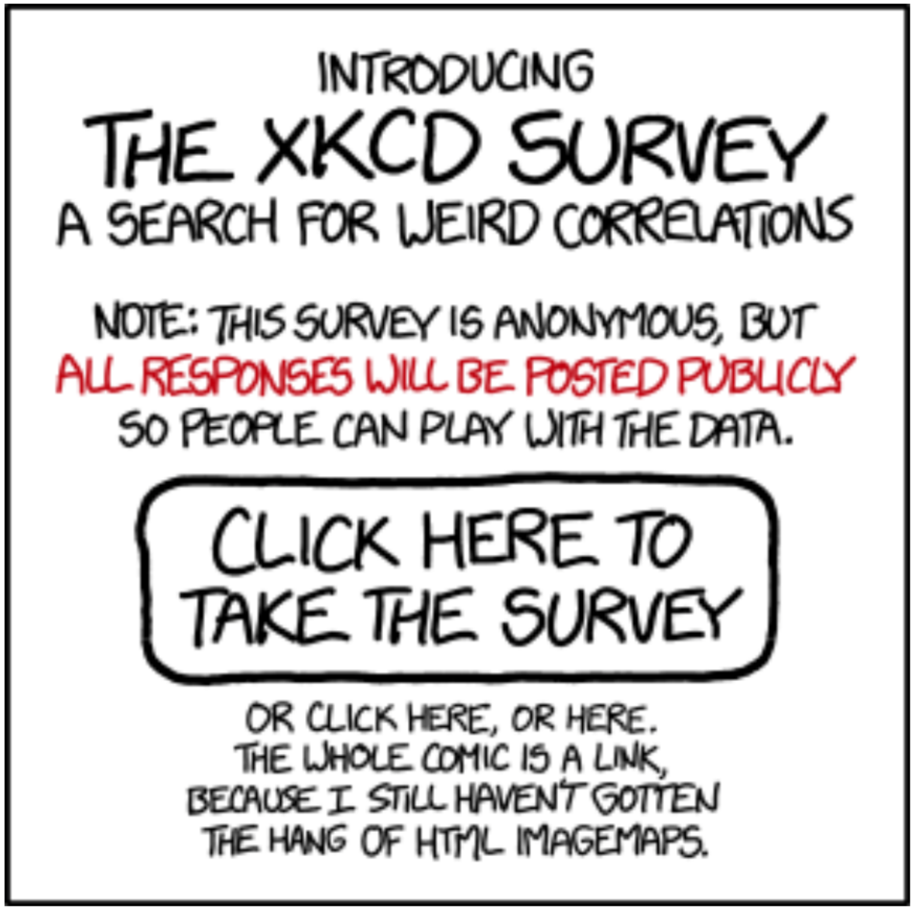




The theoretical limit is known from the Neyman-Pearson lemma using the (known/correct) PDFs in a likelihood.
In all fairness, this is a case that is great for the BDT...

The XKCD survey/sample

The cartoon XKCD had a fun survey…



INTRODUCING

THE XKCD SURVEY

A SEARCH FOR WEIRD CORRELATIONS

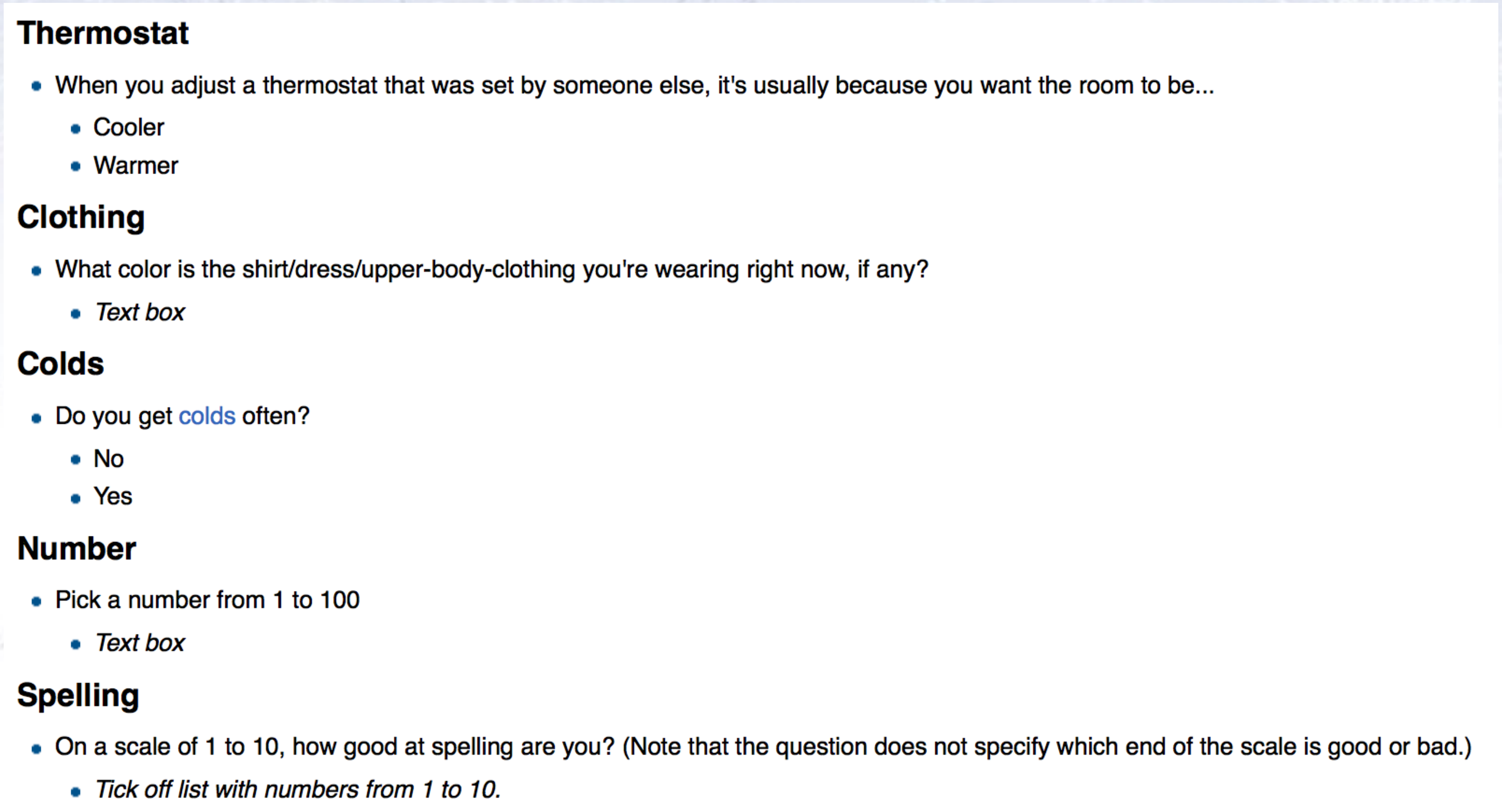
NOTE: THIS SURVEY IS ANONYMOUS, BUT
ALL RESPONSES WILL BE POSTED PUBLICLY
SO PEOPLE CAN PLAY WITH THE DATA.

CLICK HERE TO
TAKE THE SURVEY

OR CLICK HERE, OR HERE.
THE WHOLE COMIC IS A LINK,
BECAUSE I STILL HAVEN'T GOTTEN
THE HANG OF HTML IMAGEMAPS.

I wanted to have included this as an example of using MVA on complex data, but unfortunately it was not made public yet! Next year, I hope…

The XKCD survey/sample



Thermostat

- When you adjust a thermostat that was set by someone else, it's usually because you want the room to be...
  - Cooler
  - Warmer

Clothing

- What color is the shirt/dress/upper-body-clothing you're wearing right now, if any?
  - *Text box*

Colds

- Do you get colds often?
  - No
  - Yes

Number

- Pick a number from 1 to 100
  - *Text box*

Spelling

- On a scale of 1 to 10, how good at spelling are you? (Note that the question does not specify which end of the scale is good or bad.)
  - *Tick off list with numbers from 1 to 10.*

Decision tree learning

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

because it:

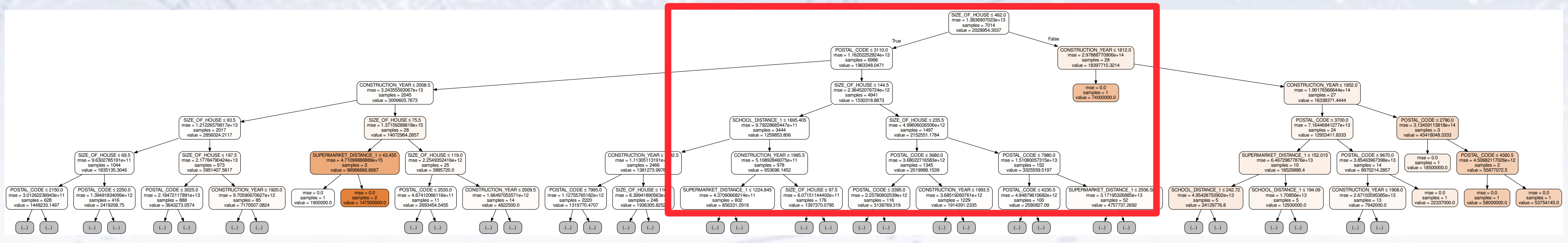
- is invariant under scaling and various other transformations of feature values,
- is robust to inclusion of irrelevant features,
- produces inspectable models.

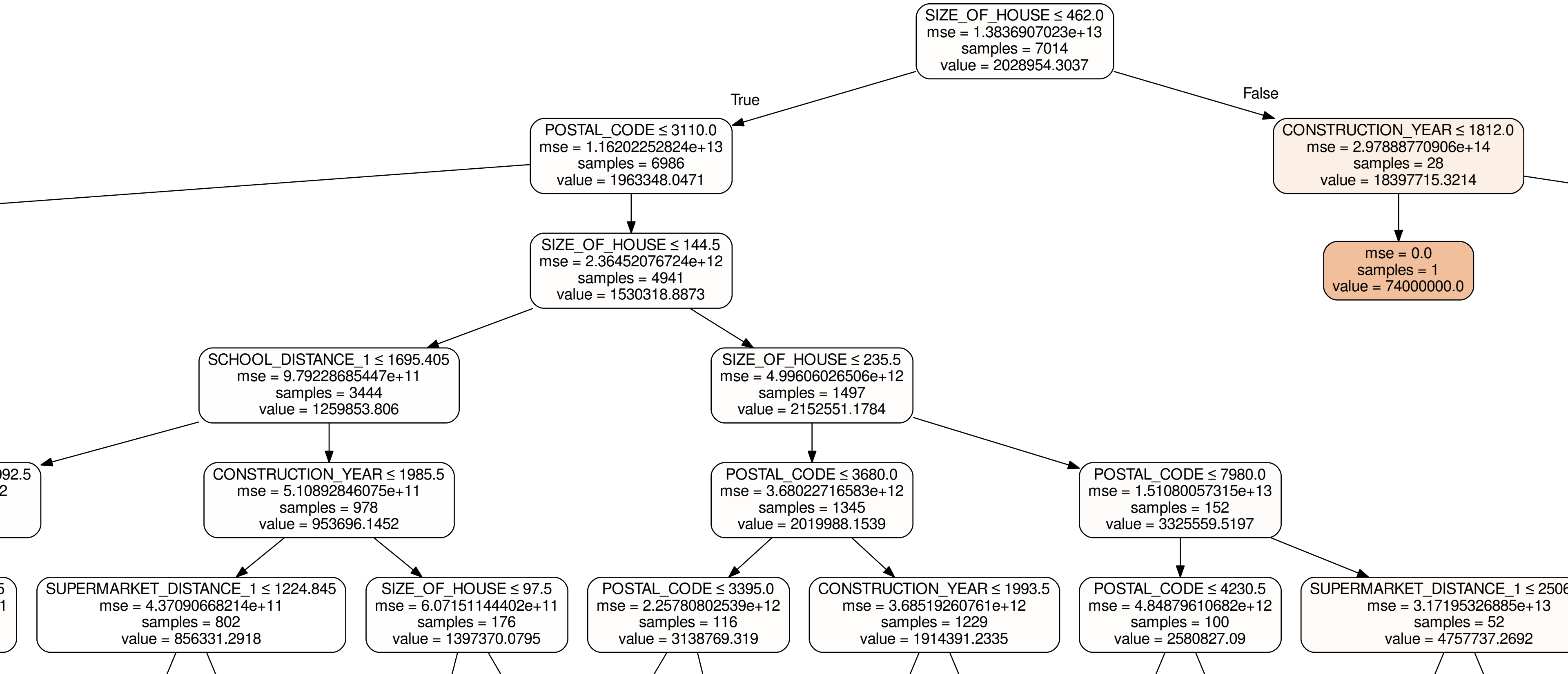
HOWEVER… they are seldom accurate (i.e. most performant)!

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

Housing Prices decision tree

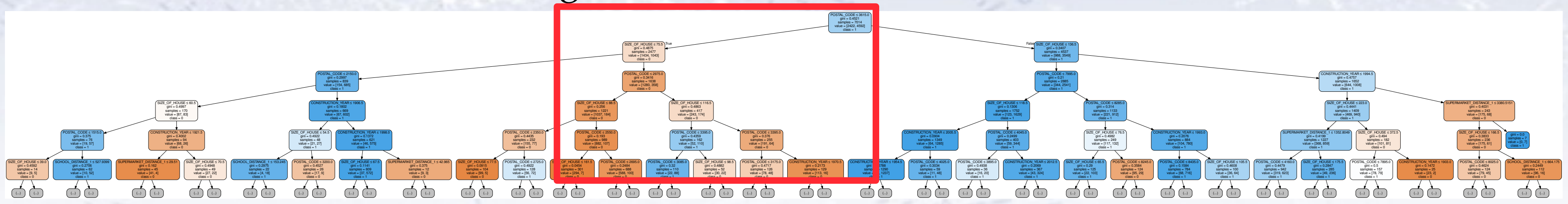
Decision tree for estimating the price in the housing prices data set:

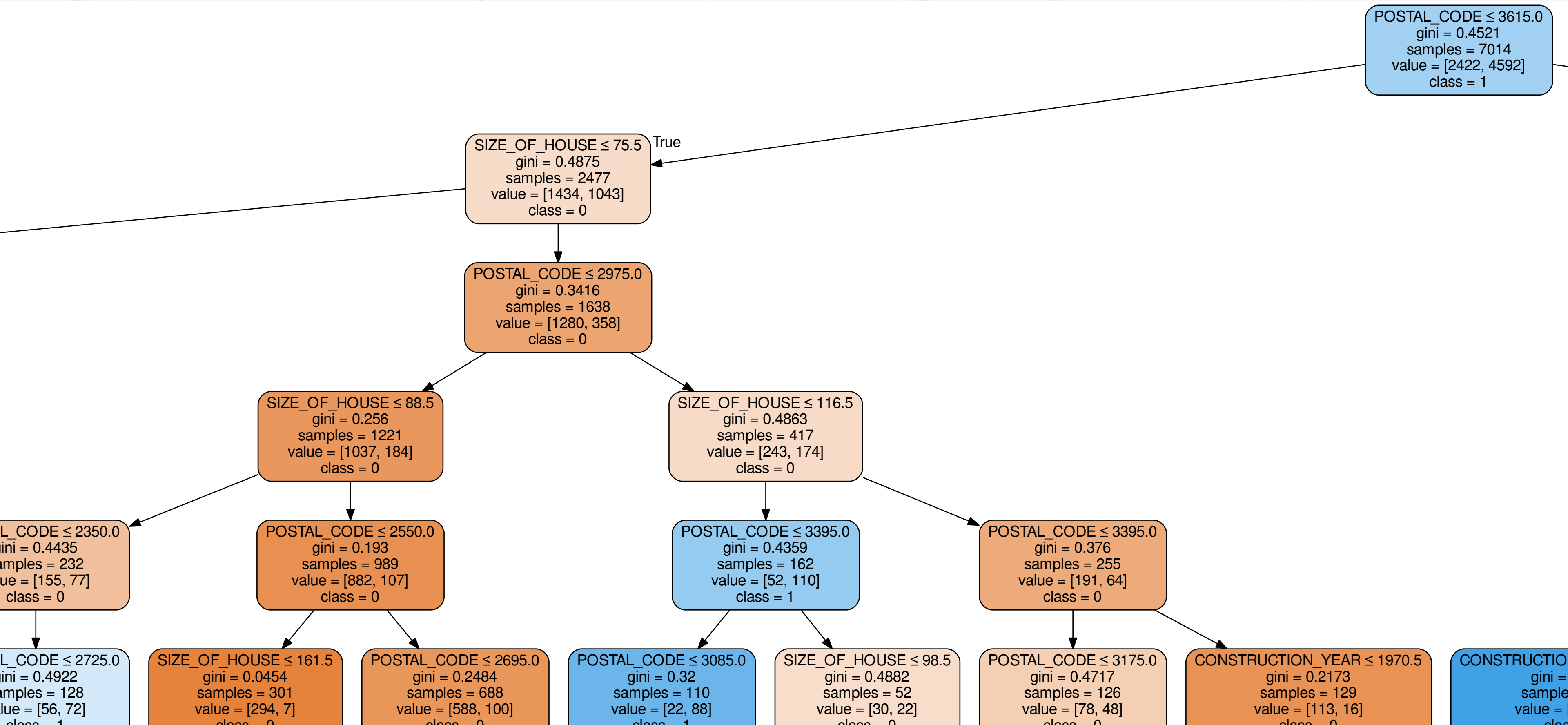




Housing Prices decision tree

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



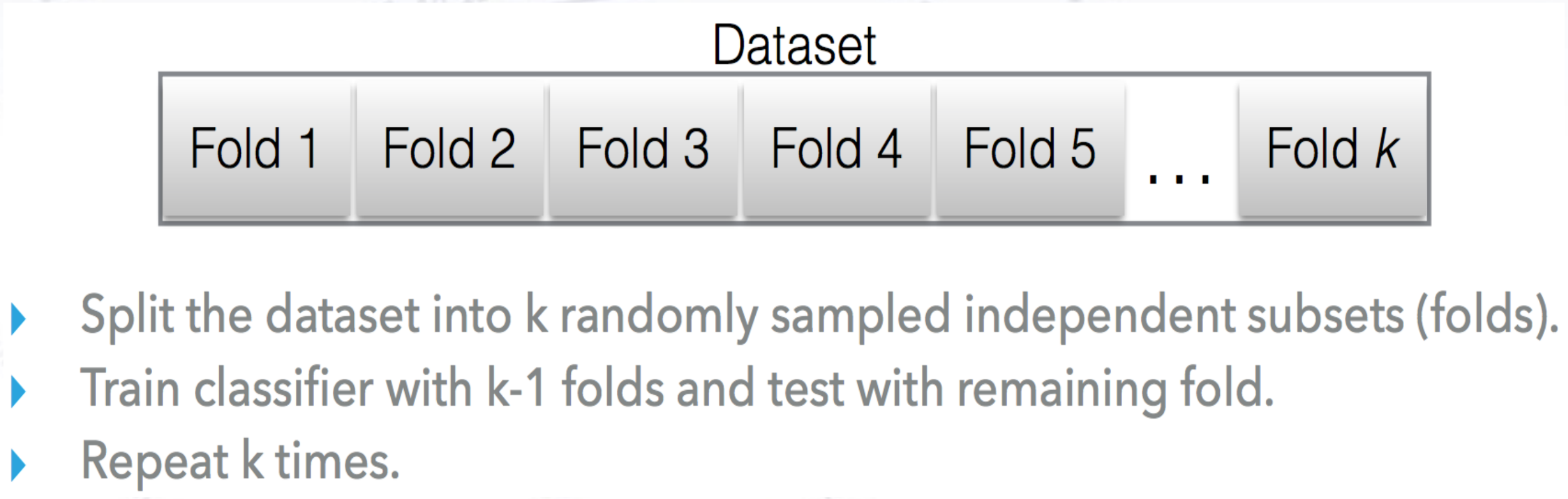


POSTAL_CODE ≤ 3615.0 gini = 0.4521 samples = 7014 value = [2422, 4592] class = 1
True
SIZE_OF_HOUSE ≤ 75.5 gini = 0.4875 samples = 2477 value = [1434, 1043] class = 0
POSTAL_CODE ≤ 2975.0 gini = 0.3416 samples = 1638 value = [1280, 358] class = 0
SIZE_OF_HOUSE ≤ 88.5 gini = 0.256 samples = 1221 value = [1037, 184] class = 0
SIZE_OF_HOUSE ≤ 116.5 gini = 0.4863 samples = 417 value = [243, 174] class = 0
gini = 0.4435 samples = 232 value = [155, 77] class = 0
POSTAL_CODE ≤ 2550.0 gini = 0.193 samples = 989 value = [882, 107] class = 0
POSTAL_CODE ≤ 3395.0 gini = 0.4359 samples = 162 value = [52, 110] class = 1
POSTAL_CODE ≤ 3395.0 gini = 0.376 samples = 255 value = [191, 64] class = 0
gini = 0.4922 samples = 128 value = [56, 72]
SIZE_OF_HOUSE ≤ 161.5 gini = 0.0454 samples = 301 value = [294, 7]
POSTAL_CODE ≤ 2695.0 gini = 0.2484 samples = 688 value = [588, 100]
POSTAL_CODE ≤ 3085.0 gini = 0.32 samples = 110 value = [22, 88]
SIZE_OF_HOUSE ≤ 98.5 gini = 0.4882 samples = 52 value = [30, 22]
POSTAL_CODE ≤ 3175.0 gini = 0.4717 samples = 126 value = [78, 48]
CONSTRUCTION_YEAR ≤ 1970.5 gini = 0.2173 samples = 129 value = [113, 16]

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.

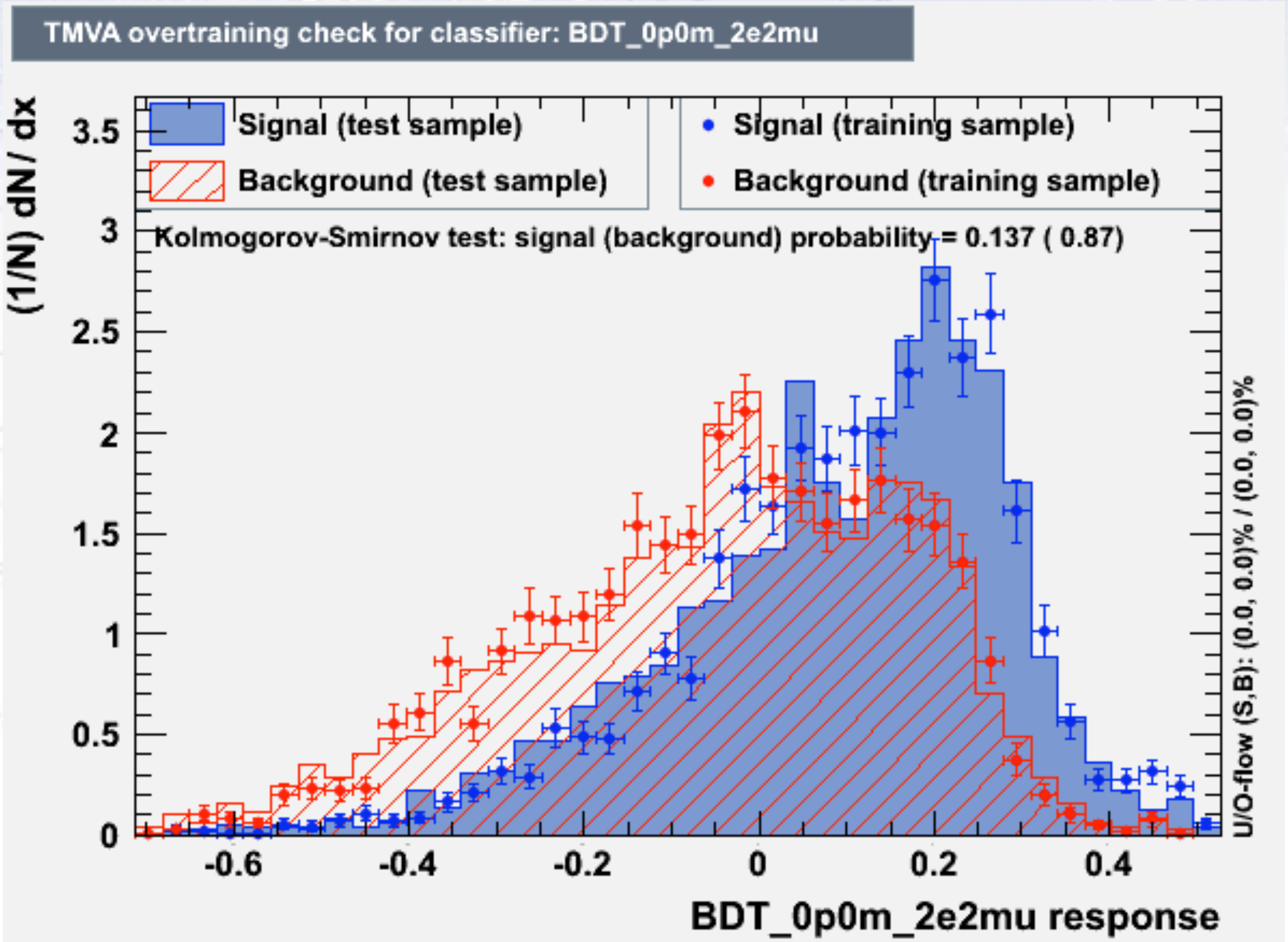
This is then repeated for each fold… CPU-intensive, but smart for small data samples.



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

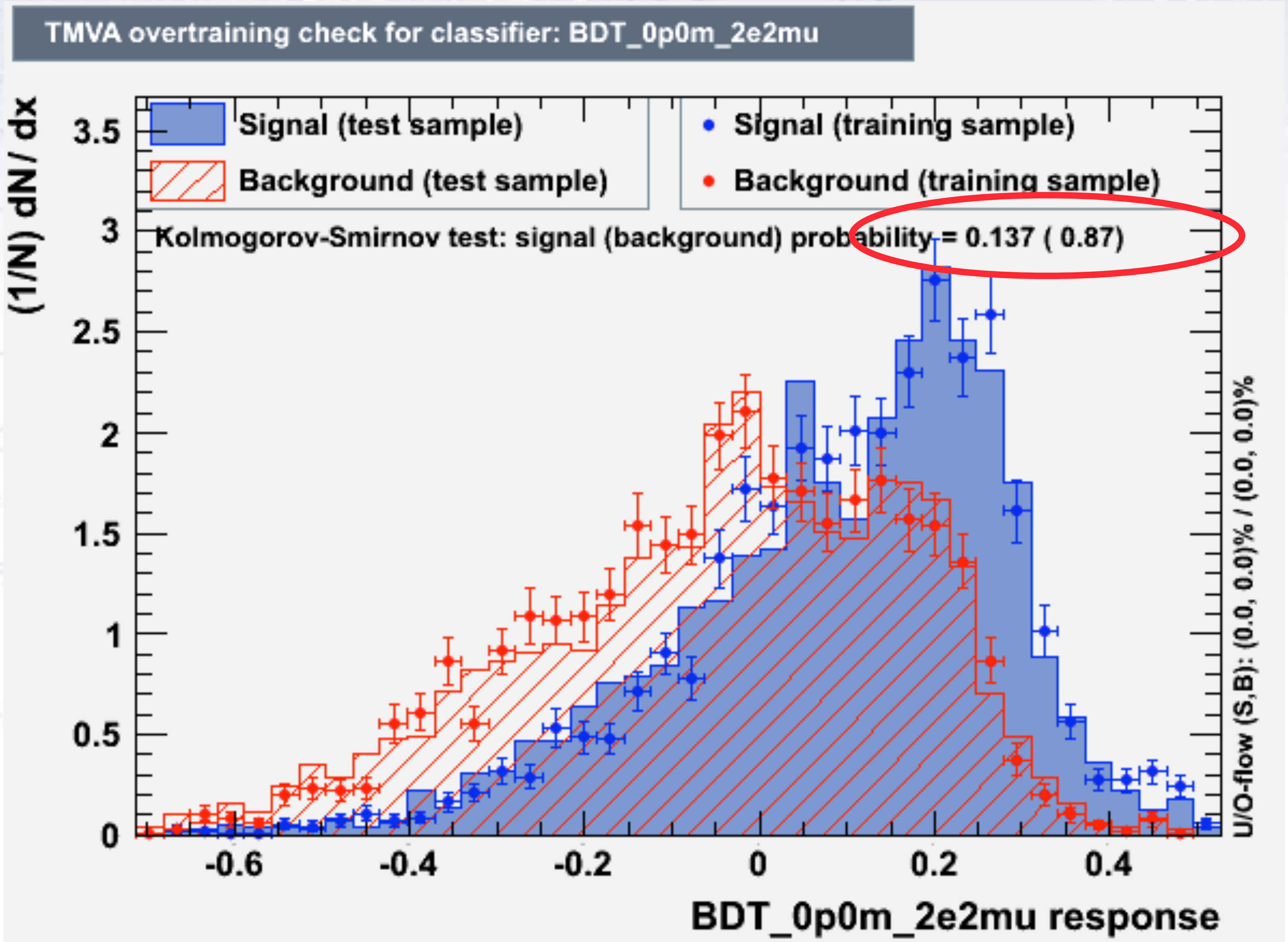
Test for overtraining

In order to test for overtraining, half the sample is used for training, the other for testing:



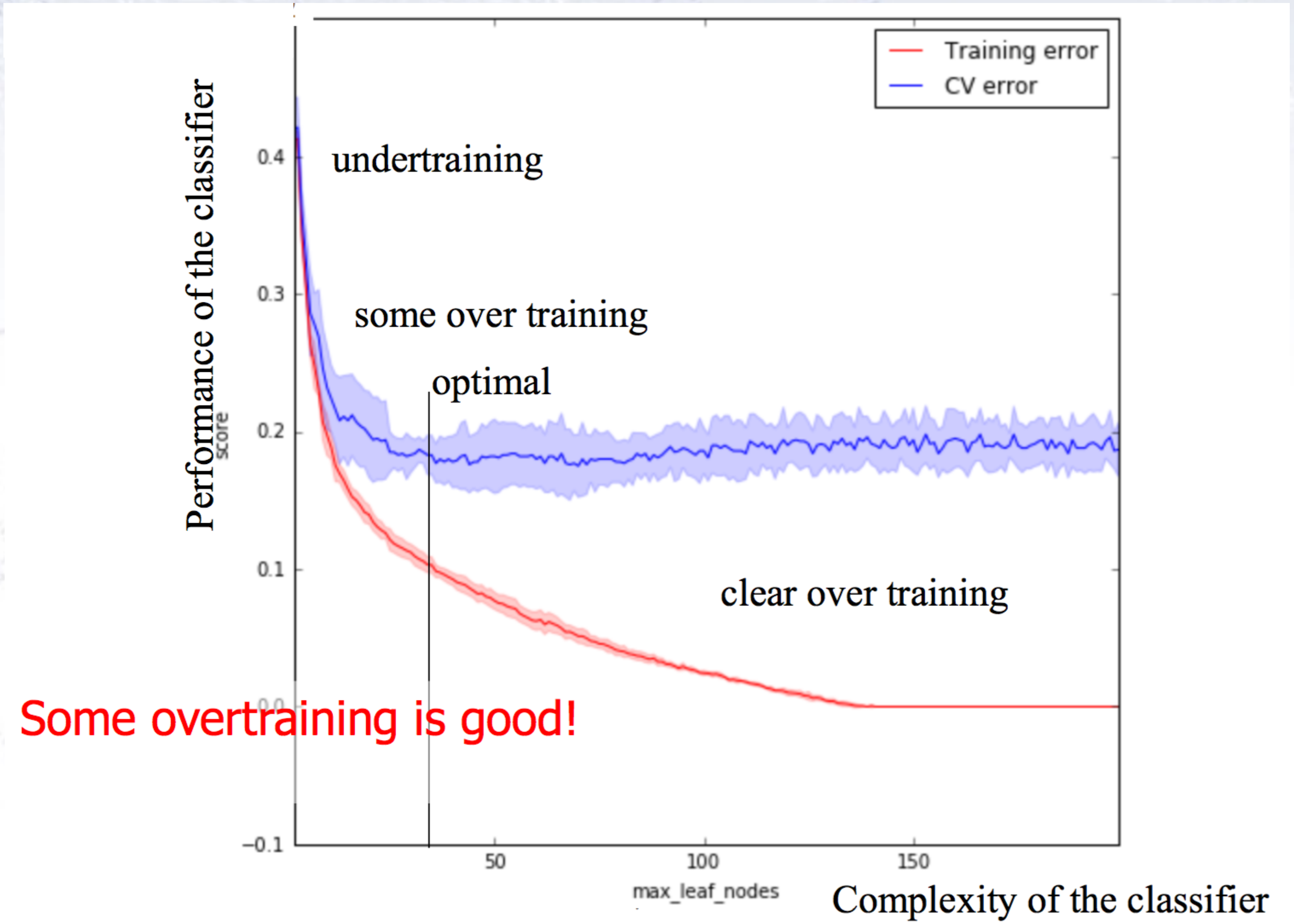
Test for overtraining

In order to test for overtraining, half the sample is used for training, the other for testing:



Overtraining…

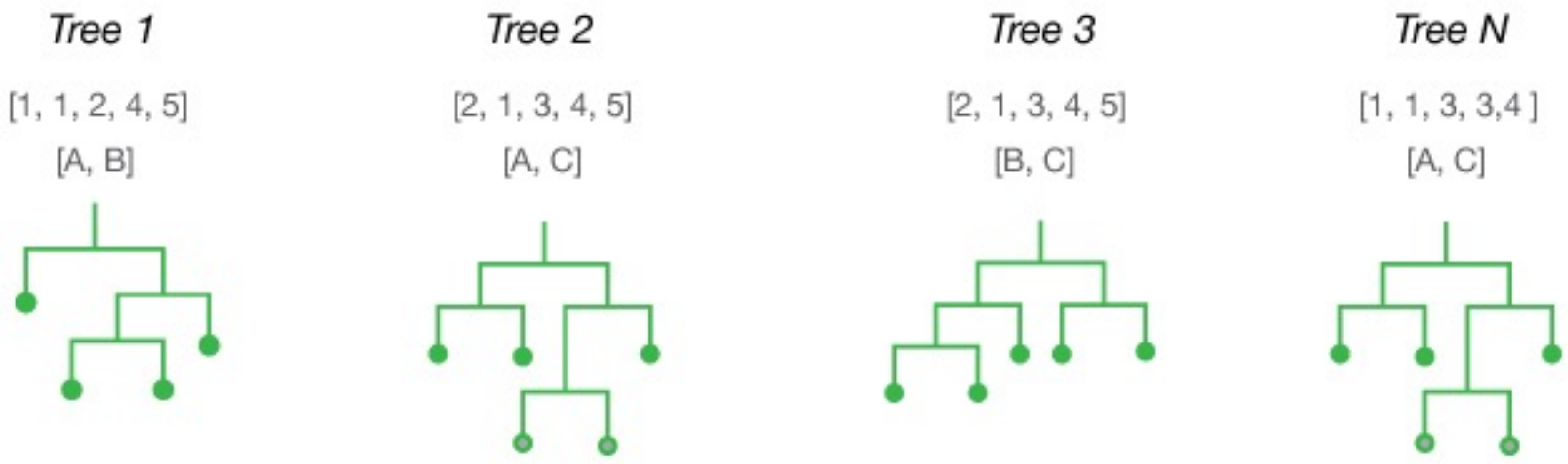
To test for overtraining, try to increase the number of parameters of your ML.
If performance on Cross Validation (CV) sample drops, decrease complexity!



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

<u>Each tree is grown as follows:</u>

- 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<<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.

<u>The forest error rate depends on two things:</u>

- 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

<u>Features of Random Forests:</u>

- 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.
However, when it comes to images, Deep Learning (on GPUs) is “the shit”.

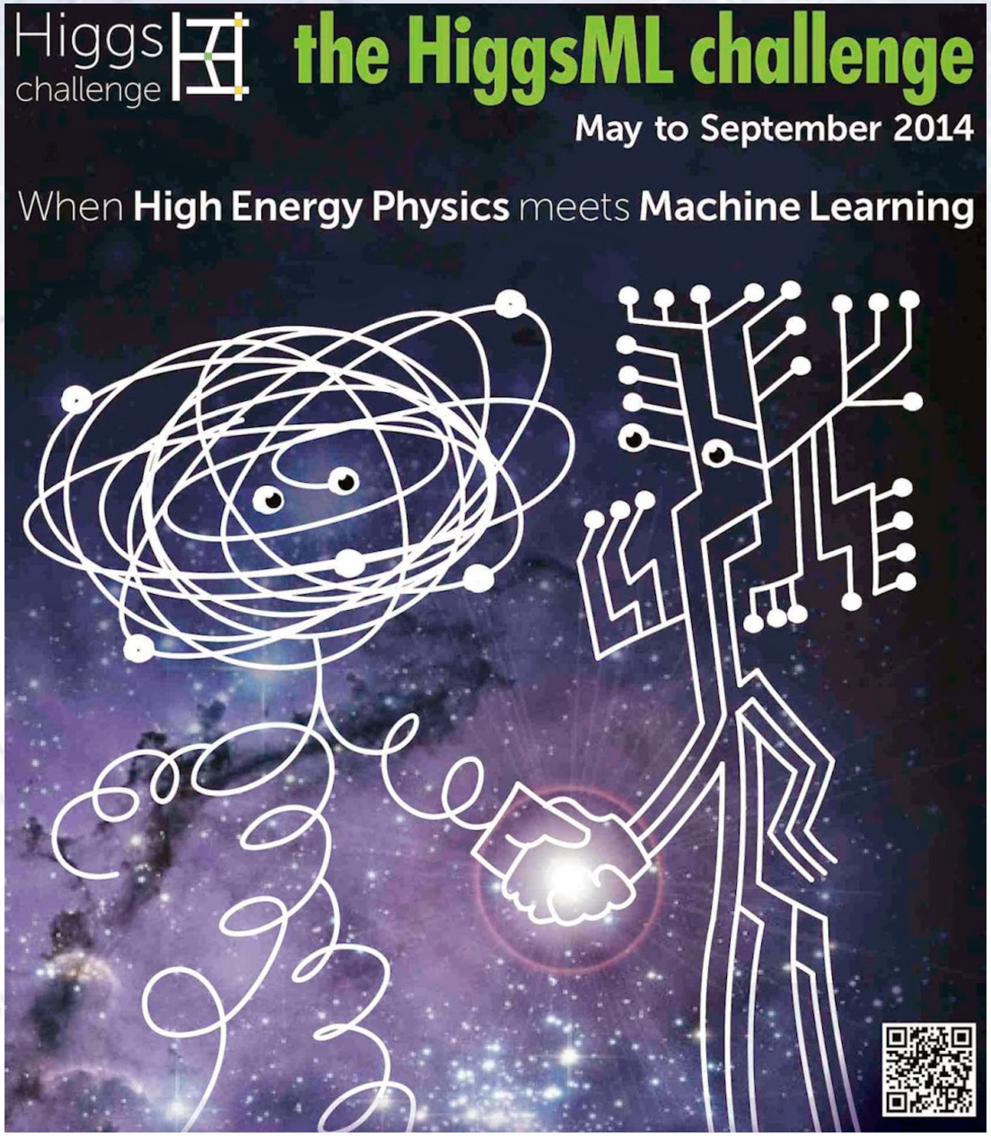
XGboost - a neat little story!

The HiggsML Kaggle Challenge

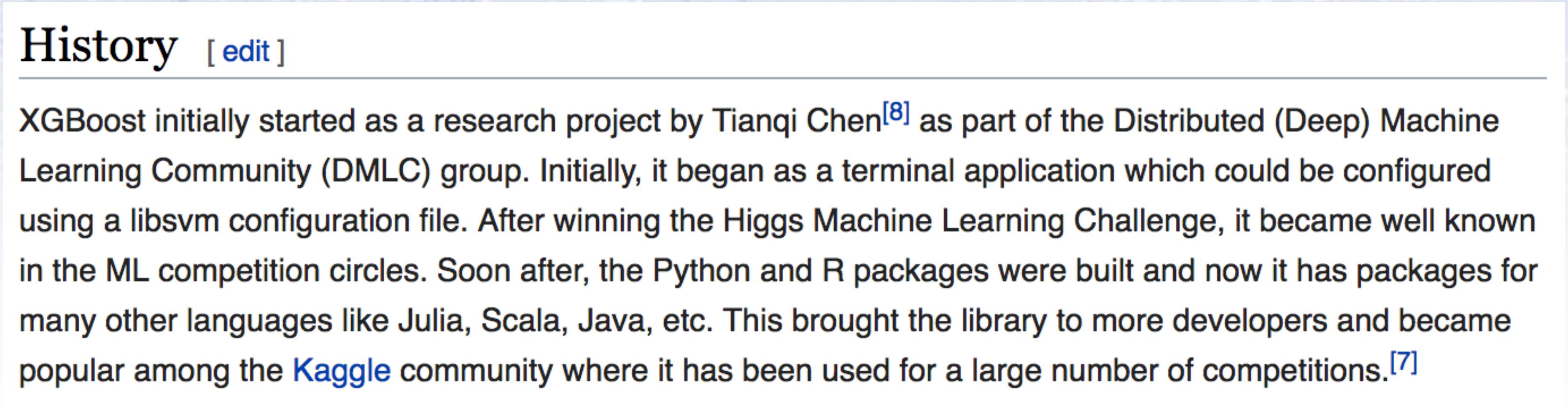
CERN analyses its data using a vast array of ML methods. CERN is thus part of the community that developpes ML!

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

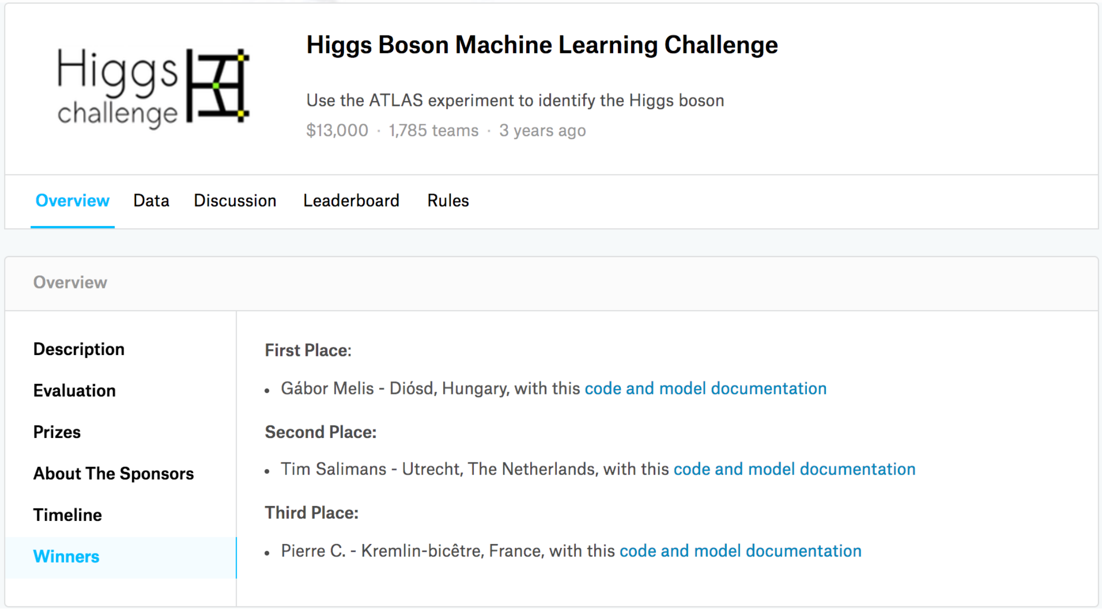


History [edit]

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.



Higgs Boson Machine Learning Challenge

Use the ATLAS experiment to identify the Higgs boson

$13,000 · 1,785 teams · 3 years ago

First Place:

- Gábor Melis - Diósd, Hungary, with this code and model documentation

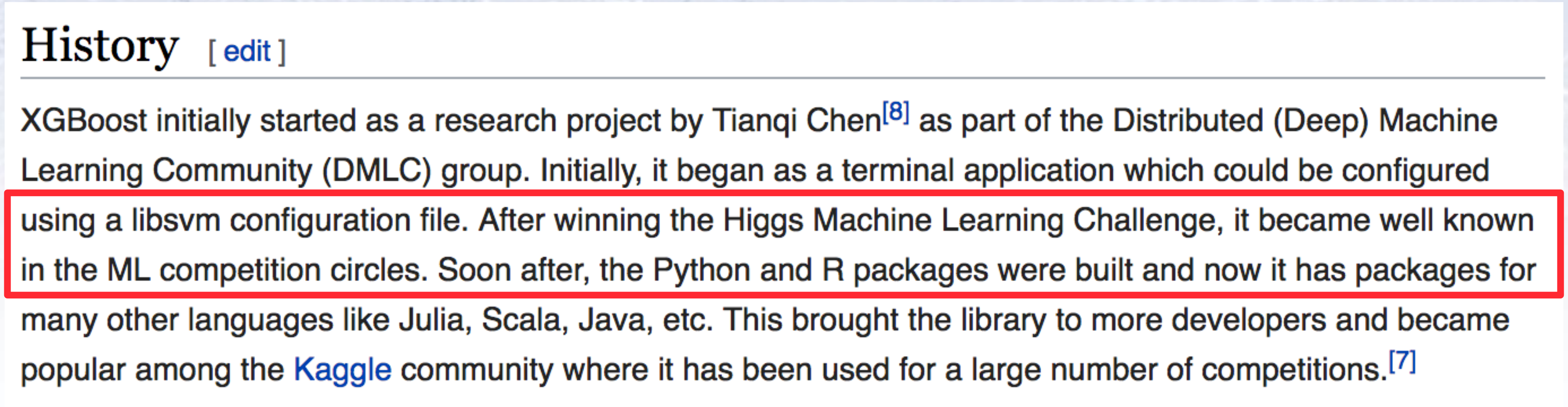
Second Place:

- 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

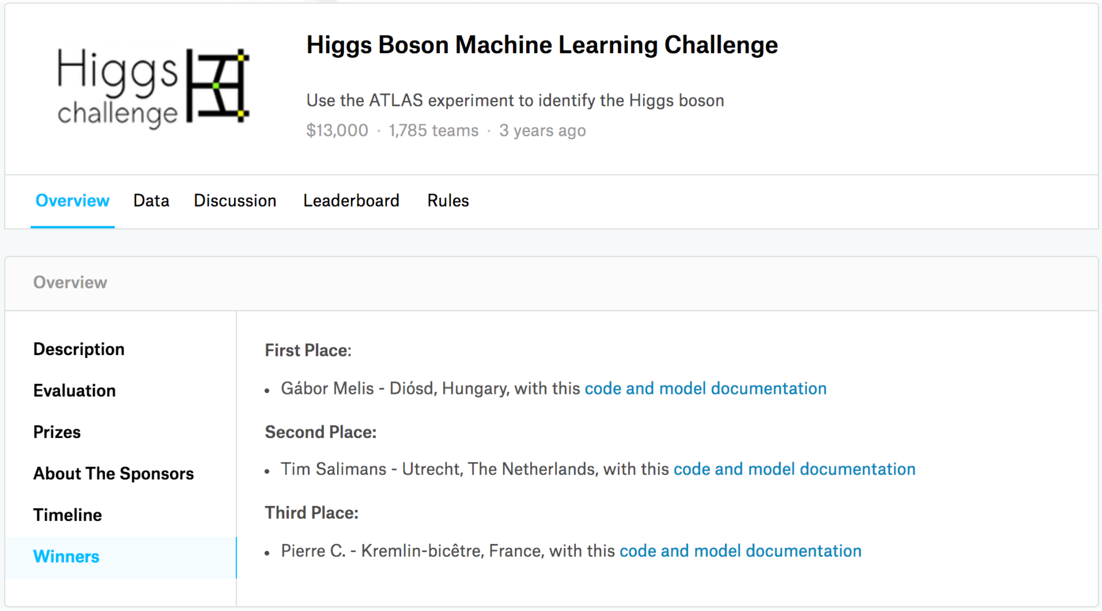


History [edit]

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.



Higgs Boson Machine Learning Challenge

Use the ATLAS experiment to identify the Higgs boson

$13,000 · 1,785 teams · 3 years ago

First Place:

- Gábor Melis - Diósd, Hungary, with this code and model documentation

Second Place:

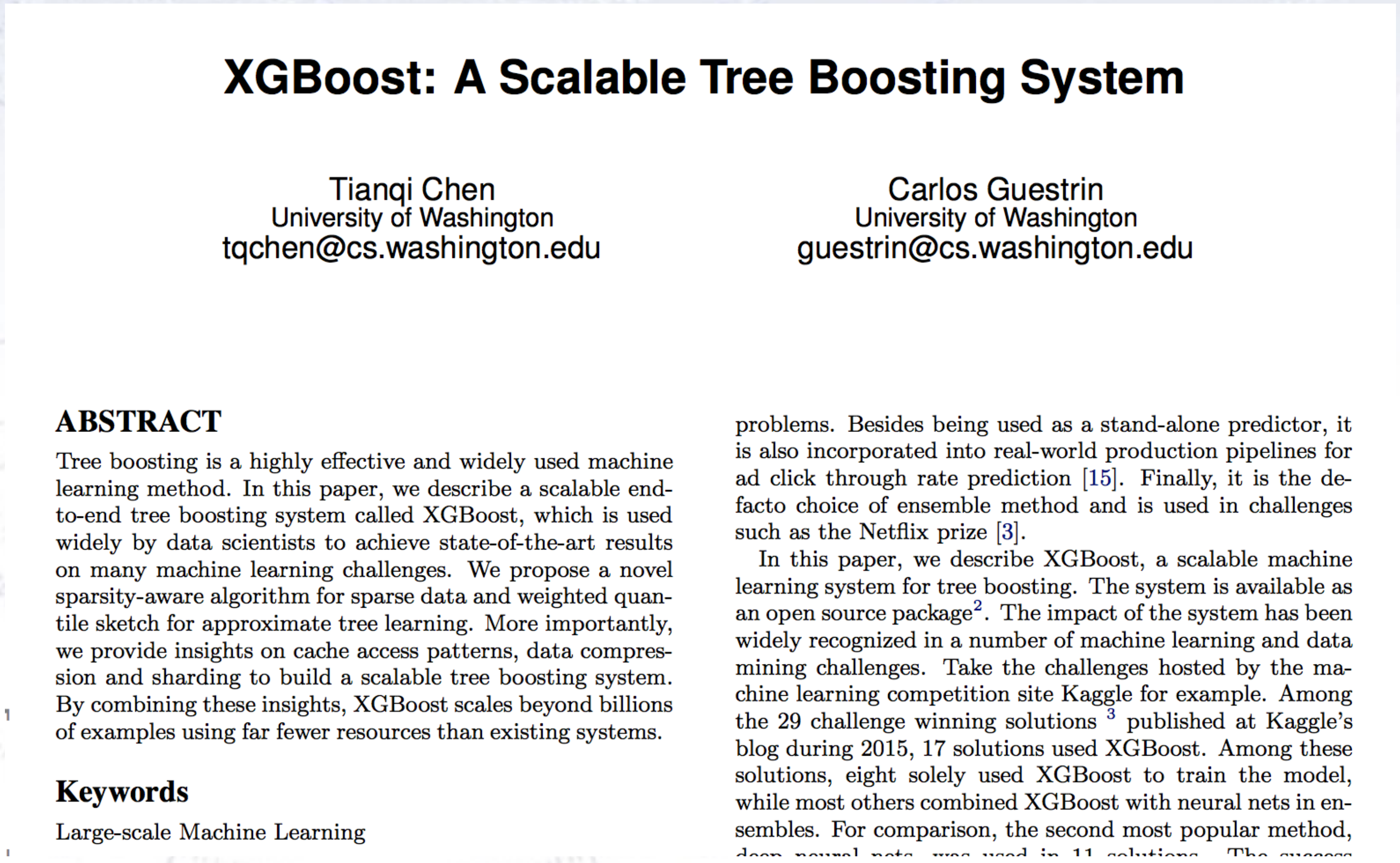
- Tim Salimans - Utrecht, The Netherlands, with this code and model documentation

Third Place:

- Pierre C. - Kremlin-bicêtre, France, with this code and model documentation

XGBoost algorithm

The algorithms 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 package[2]. 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 solutions [3] 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 algorithms is an extension of the decision tree idea (tree boosting), using regression trees with weighted quantiles and being "sparcity 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:

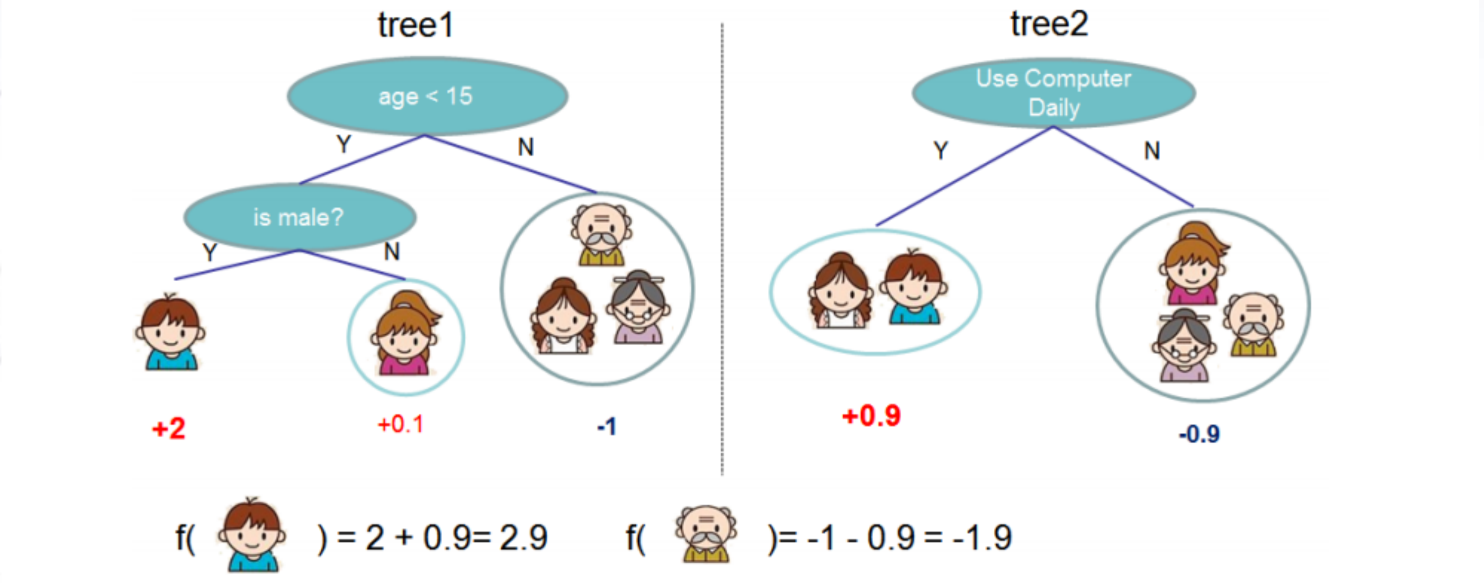
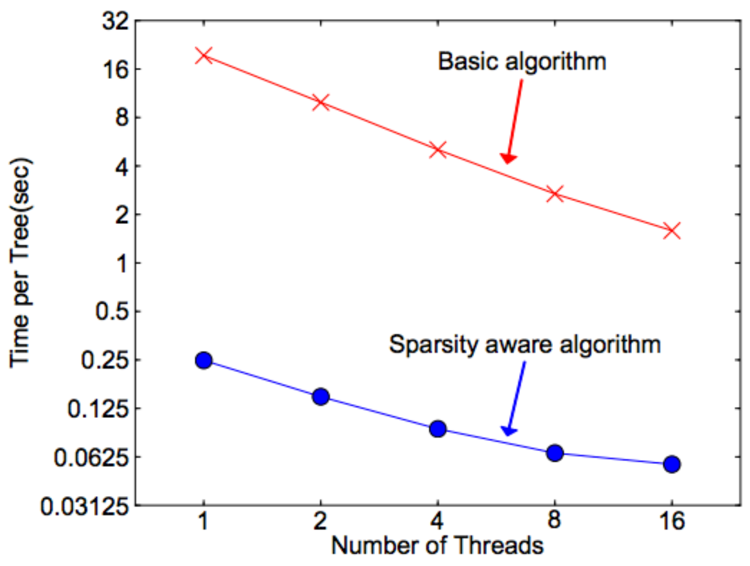
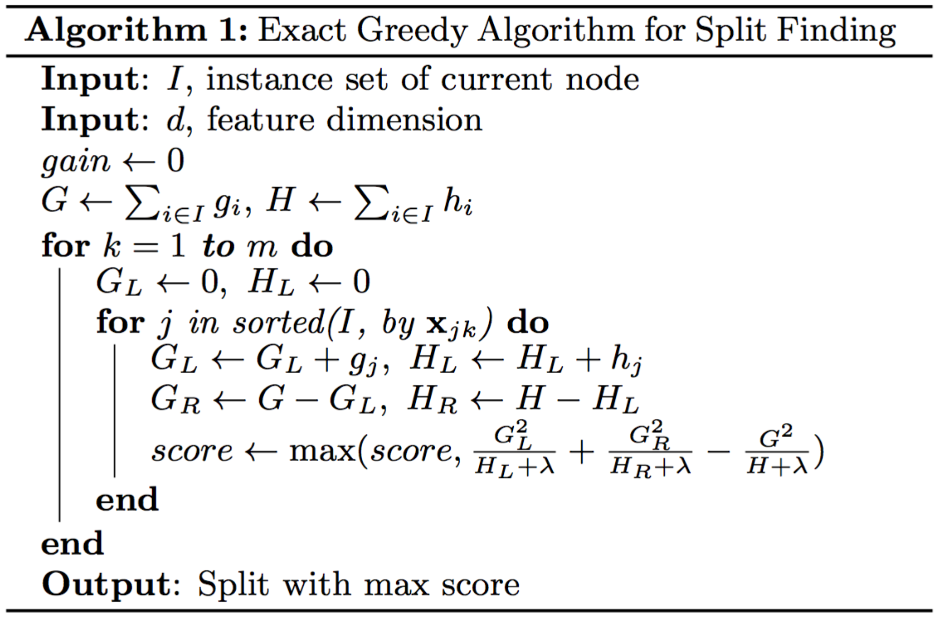


Figure 1: Tree Ensemble Model. The final prediction for a given example is the sum of predictions from each tree.

XGBoost algorithm

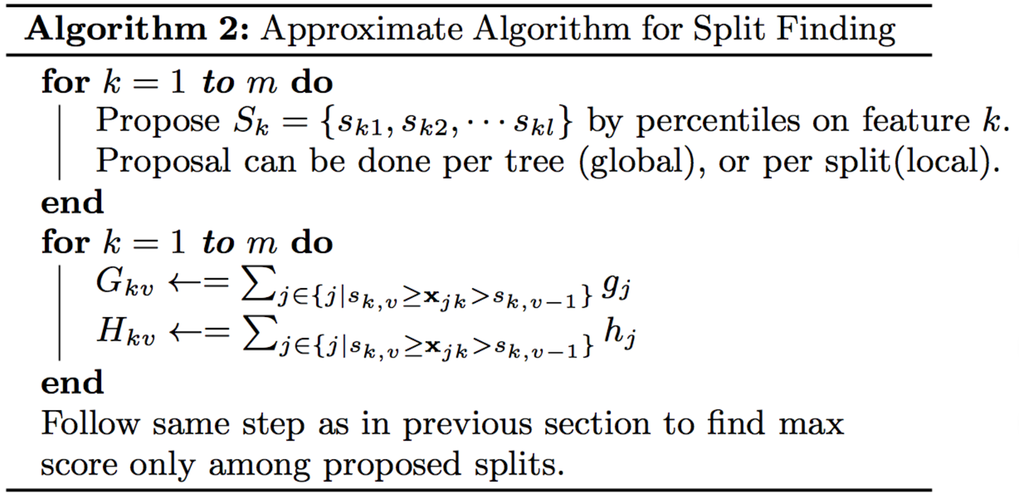
The method's speed is partly due to an approximate but fast algorithm to find the best splits.





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($I$, 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 $S_k = \{s_{k1}, s_{k2}, \cdots s_{kl}\}$ 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:

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

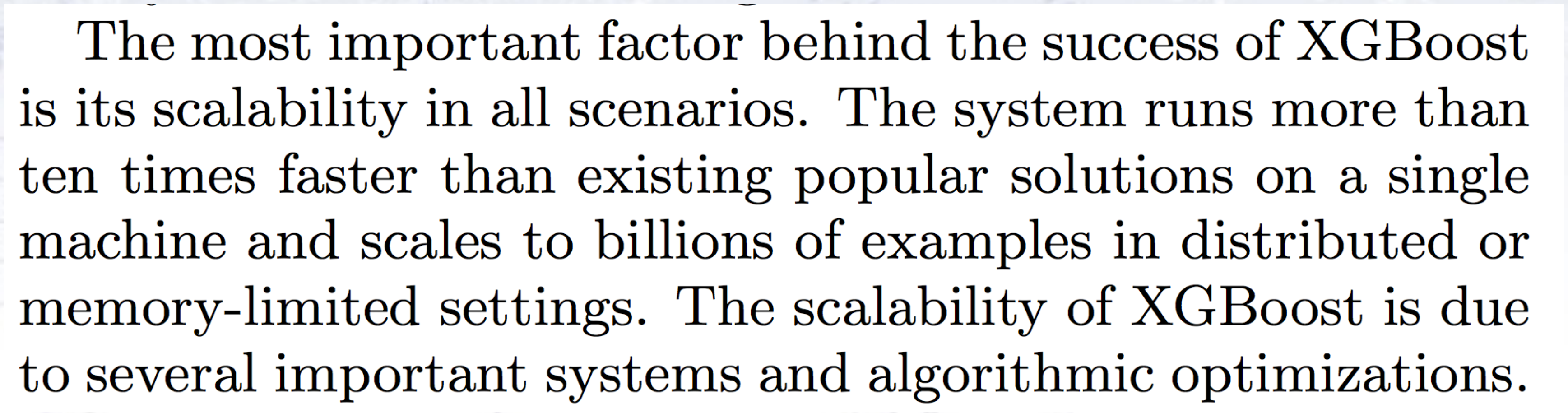
$$\text{where } \Omega(f) = \gamma T + \frac{1}{2}\lambda \|w\|^2$$

Table 1: Comparison of major tree boosting systems.

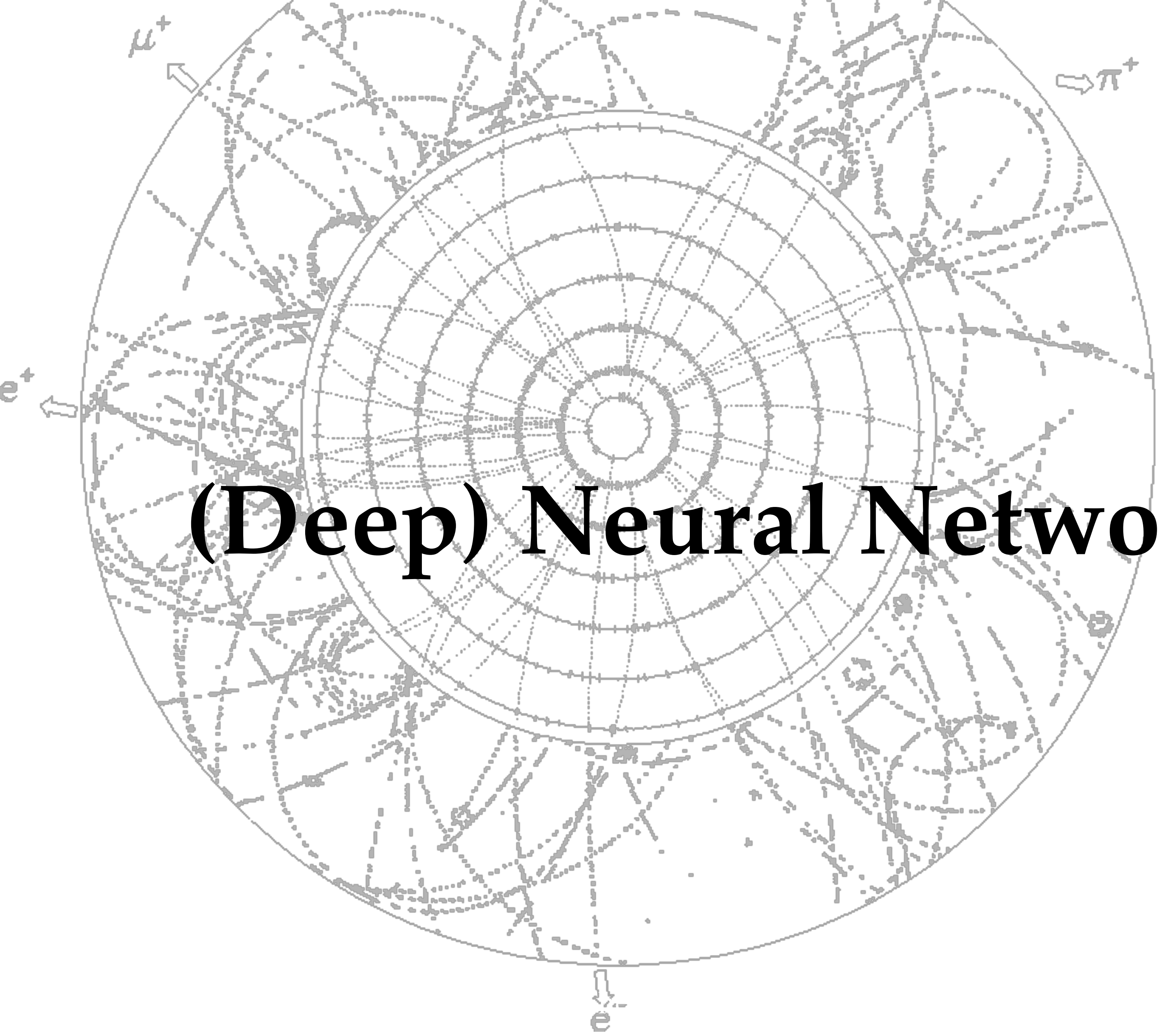
System	exact greedy	approximate global	approximate local	out-of-core	sparsity aware	parallel
XGBoost	yes	yes	yes	yes	yes	yes
pGBRT	no	no	yes	no	no	yes
Spark MLLib	no	yes	no	no	partially	yes
H2O	no	yes	no	no	partially	yes
scikit-learn	yes	no	no	no	no	no
R GBM	yes	no	no	no	partially	no

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?

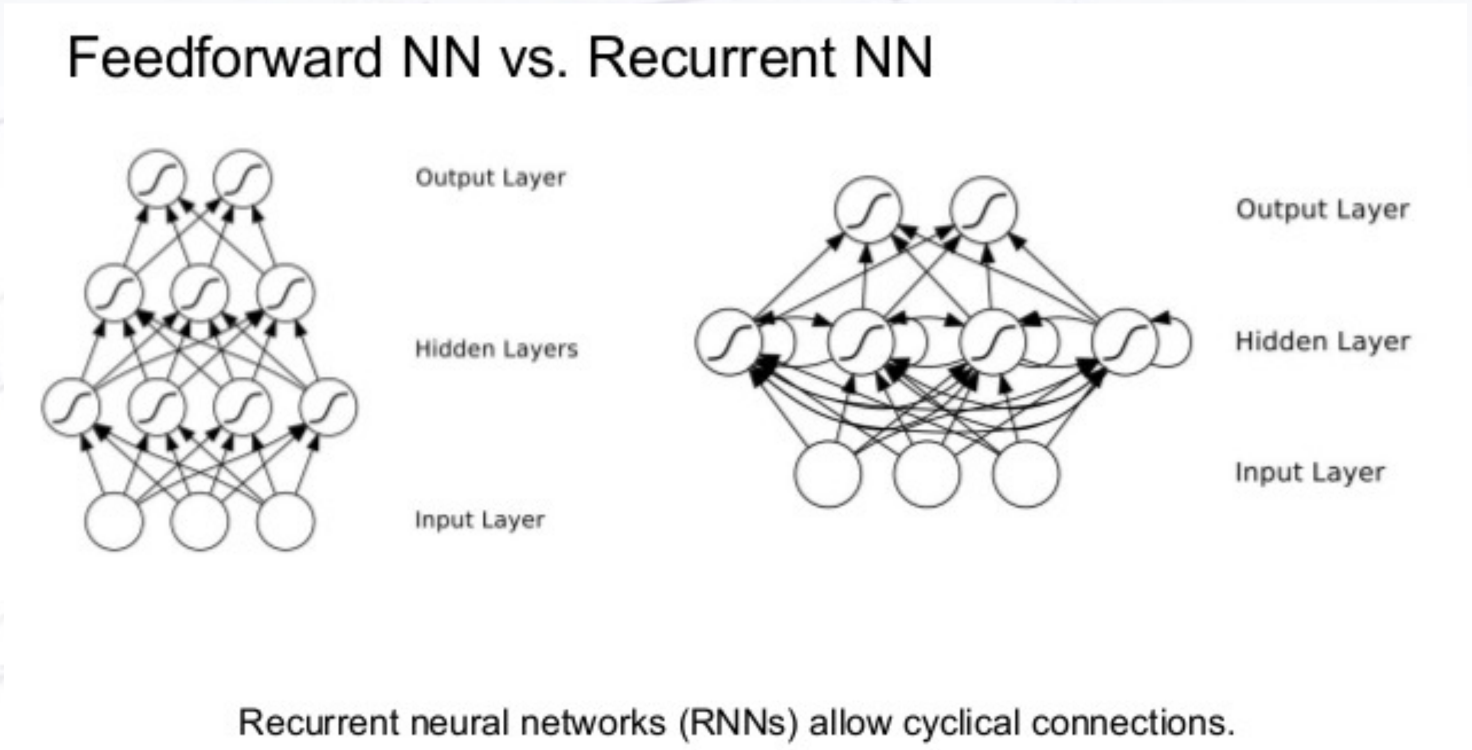


(Deep) Neural Networks

Recurrent NN

Normally, the information from one layer is fed forward to the next layer in a feedforward Neural Network (NN).

However, it may be of advantage to allow a network to give feedback, which is called a recurrent NN:



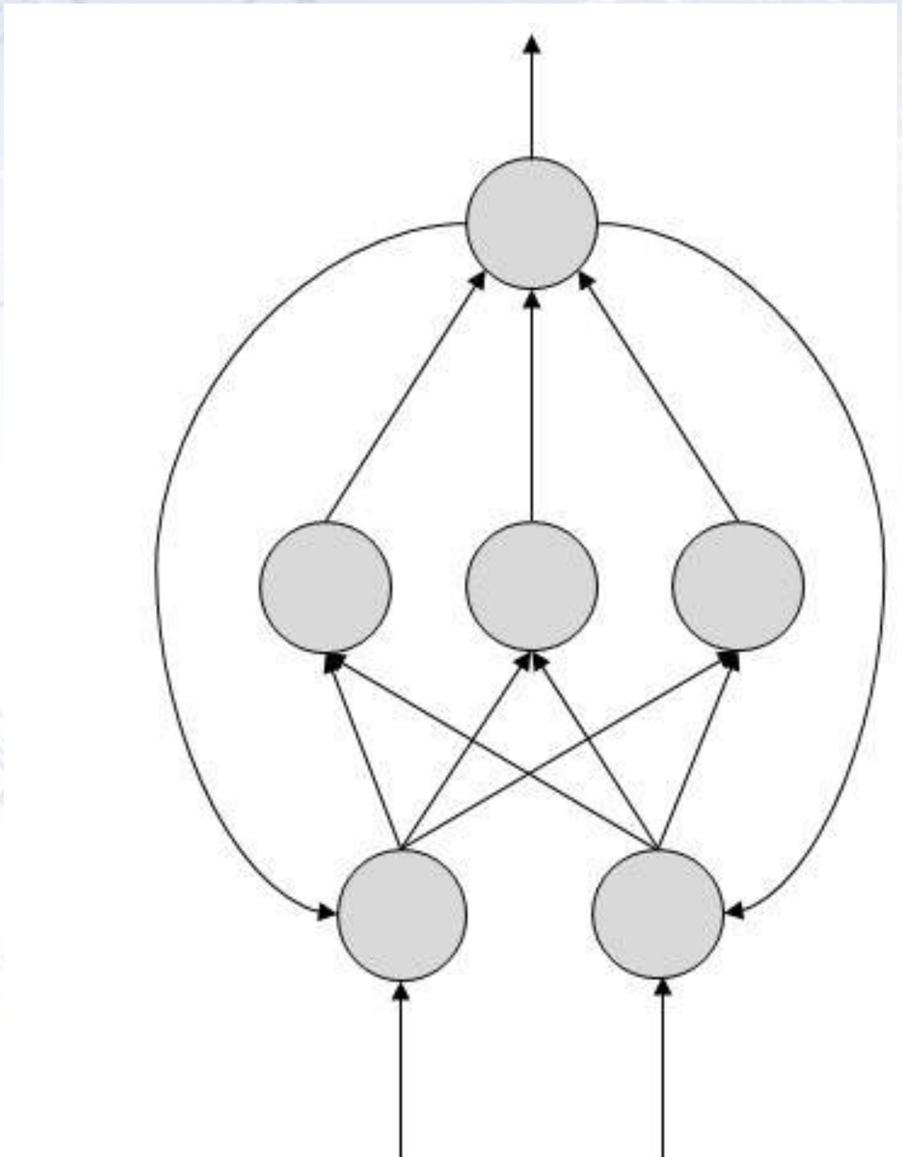
Recurrent neural networks (RNNs) allow cyclical connections.

Feedback network

There is nothing that prohibits the use of feedback in the network.

In this way, one can pass information "back" in the network, allowing for input of "more advanced" neurons to earlier neurons.

Note, that it requires skill and knowledge (and time and hard work) to design the network that suits your problem!



Networks with "memory"

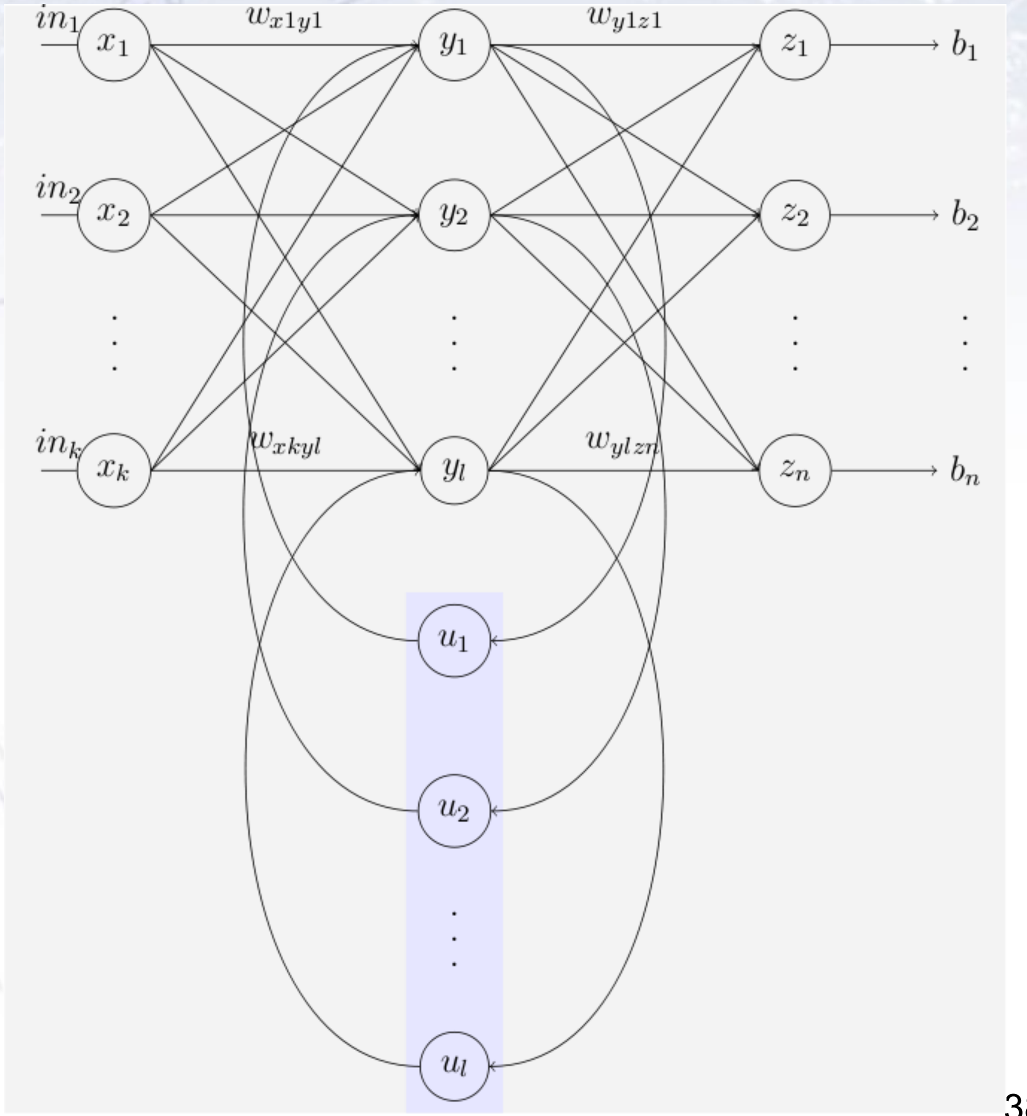
So-called Elman and Jordan networks

Allowing for feedback, one can also use this for providing "memory" of the last state(s) of the network.

This can be used for including "context" or "environment" in the network.

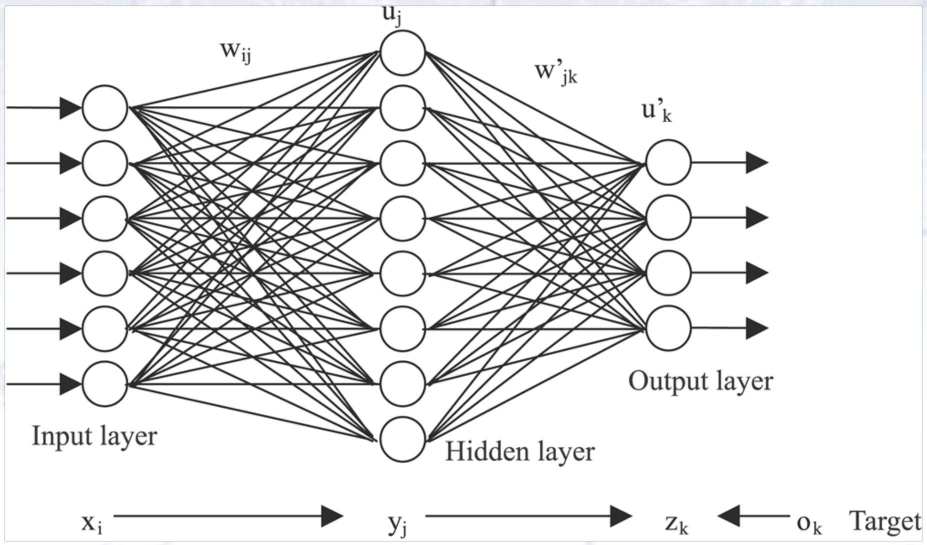
This can be used in case of e.g. a new user regarding adds, a new context regarding translation,

The keyword is Long Short-Term Memory (LSTM), if you want to look for more…

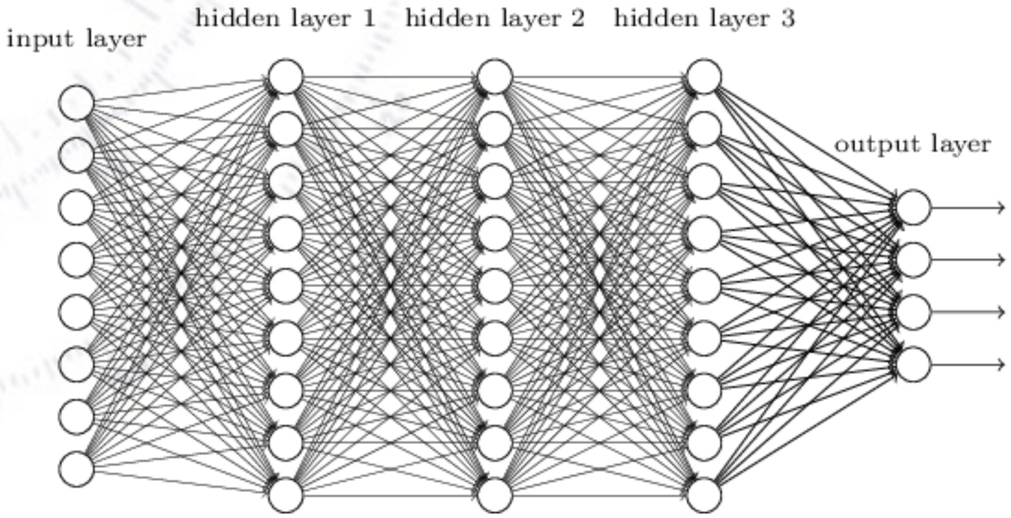


Deep Neural Networks

Deep Neural Networks (DNN) are simply (much) extended NNs in terms of layers!



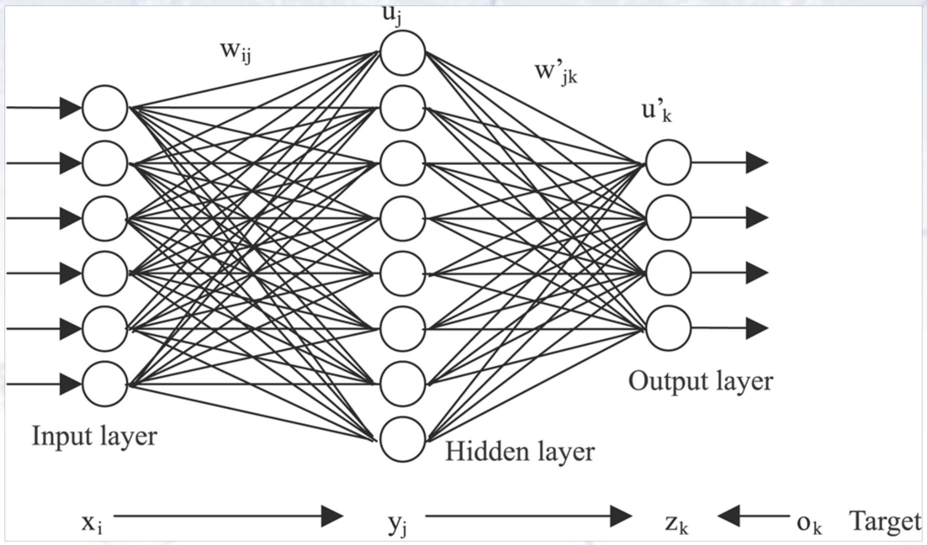
Instead of having just one (or few) hidden layers, many such layers are introduced.
This gives the network a chance to produce key features and use them for many different specialised tasks.



Currently, DNNs can have up to millions of neurons and connections, which compares to about the **brain of a worm**.

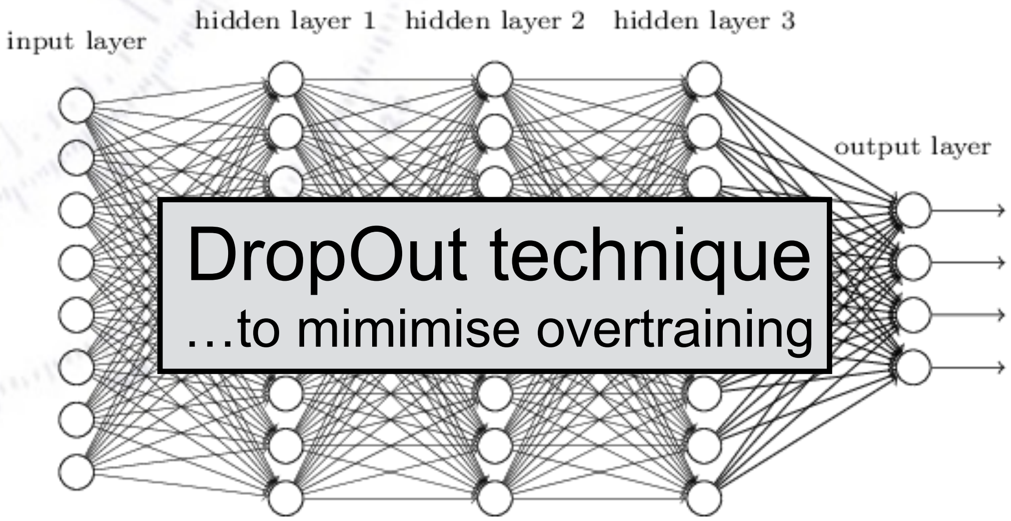
Deep Neural Networks

Deep Neural Networks (DNN) are simply (much) extended NNs in terms of layers!



Instead of having just one (or few) hidden layers, many such layers are introduced.
This gives the network a chance to produce key features and use them for many different specialised tasks.

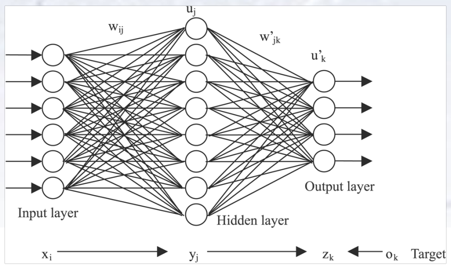
Currently, DNNs can have up to millions of neurons and connections, which compares to about the **brain of a worm**.

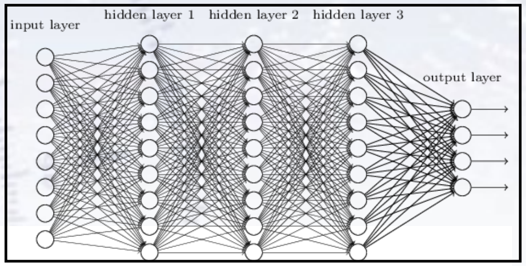


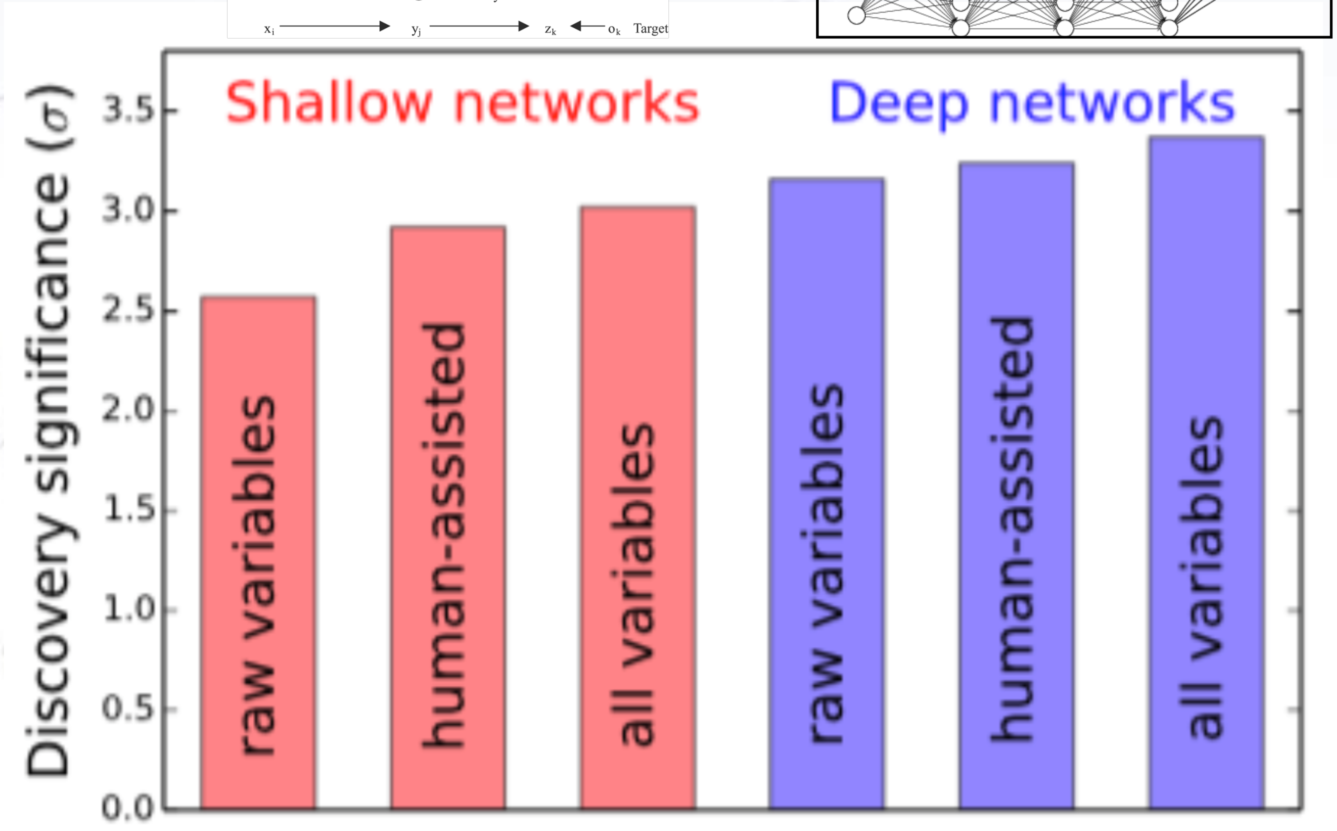
DropOut technique
…to mimimise overtraining

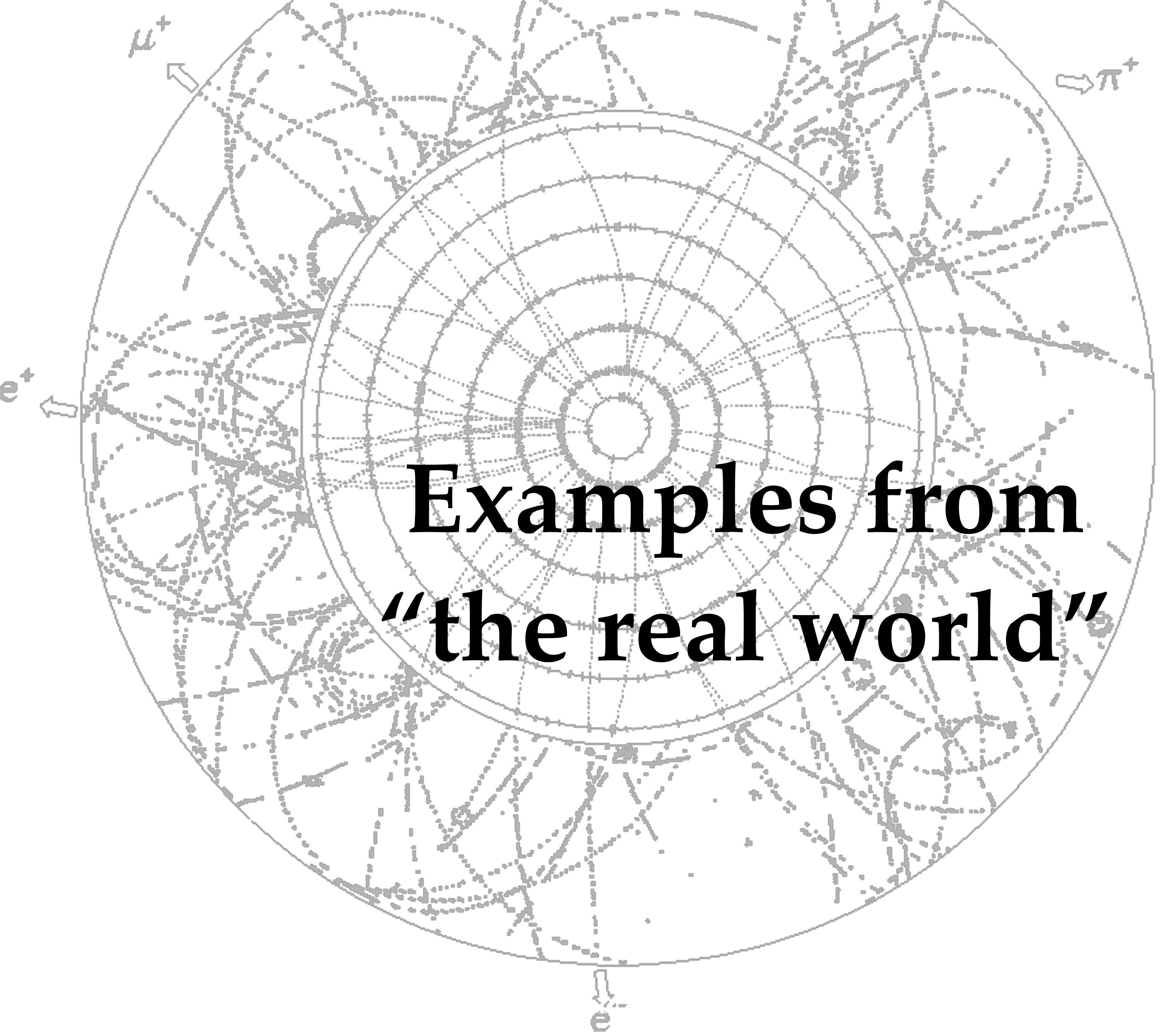
Deep Neural Networks

Deep Neural Networks likes to get both raw and "assisted" variables:





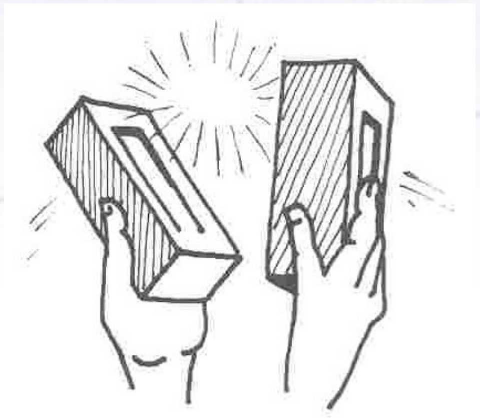




Examples from "the real world"

A great early example

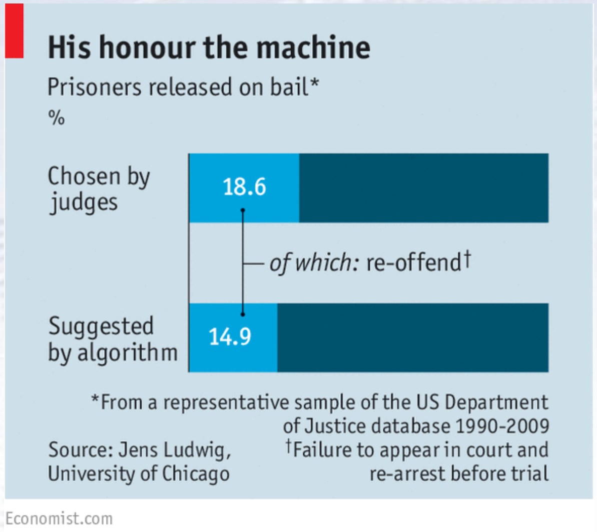
A company producing bricks considered using Machine Learning in the quality control of the bricks. Until now, this had been done manually, with workers **discarding about 4-7% of bricks**.



Based on color, surface and strength of the bricks, a very basic algorithm was trained/optimised and put in place to do the quality control. This worked reasonably well, but unlike the workers, the machine was **discarding 2-30% of bricks!**

How could that be? WHY?!?

Examples:



<u>Who should be released on bail:</u>
In an attempt to find out, who should be released on bail, a group at University of Chicago (CrimeLab) looked into the data of setting bail.
They could decrease the fraction of re-offenders from **18.6%** to **14.9%**. This corresponds (they claim) to what would require 20.000 police officers (2.6 billion $).

<u>Heart attack predictions (4 hours in advance):</u>
A study suggested that succes rate would go from 30% to 80% with Big Data.

<u>Flagging "at risk officers" in US police force:</u>
Using current data, and increase in correct predictions would increase by 12%, while the number of wrong predictions would be reduced by 1/3. (Cop pulling gun at pool).

Chicago is trying to predict, which children have high levels of lead in their blood.

India does actually predict, when it is the best time to sow, and tell farmers.

Ideas:

Turning the Big Data idea with myself and (similar minded) tech friends, I've thought of the following ideas (list not exhaustive):

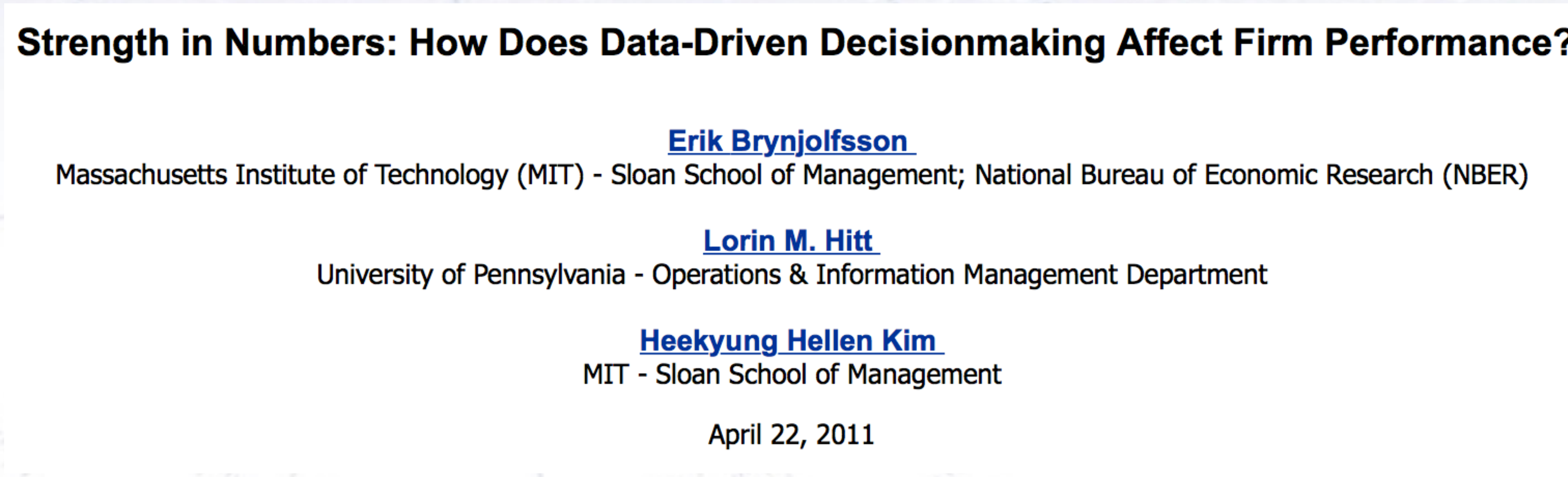
SKAT (ministry of taxation):
SKAT has very large amounts of data on all Danish tax payers going back in time. With supervised learning (i.e. based on experience from known cases), it would be **very interesting** to see, if Big Data could provide SKAT with a list of suspects, to be investigated further!
Also with unsupervised learning, one could divide tax payers into categories, and see if those "alike" (as defined by the clustering) a fraud are also themselves!

Doctors reports:
Doctors have to write a report for every patient visited, and they are typically very standard.
Could one from recording speech (and possibly camera, accelerometer, etc.), create an outline of the necessary report, of course to be checked by the doctor?

Transparency: For transparent answers, use associated Fisher to explain factors.

What to expect?

Typically, businesses are already good at what they are doing (or they would not be in business anymore!), so the improvements one can expect are typically not that large. A study looked into this, by considering 179 businesses:



Strength in Numbers: How Does Data-Driven Decisionmaking Affect Firm Performance

Erik Brynjolfsson
Massachusetts Institute of Technology (MIT) - Sloan School of Management; National Bureau of Economic Research (NBER)

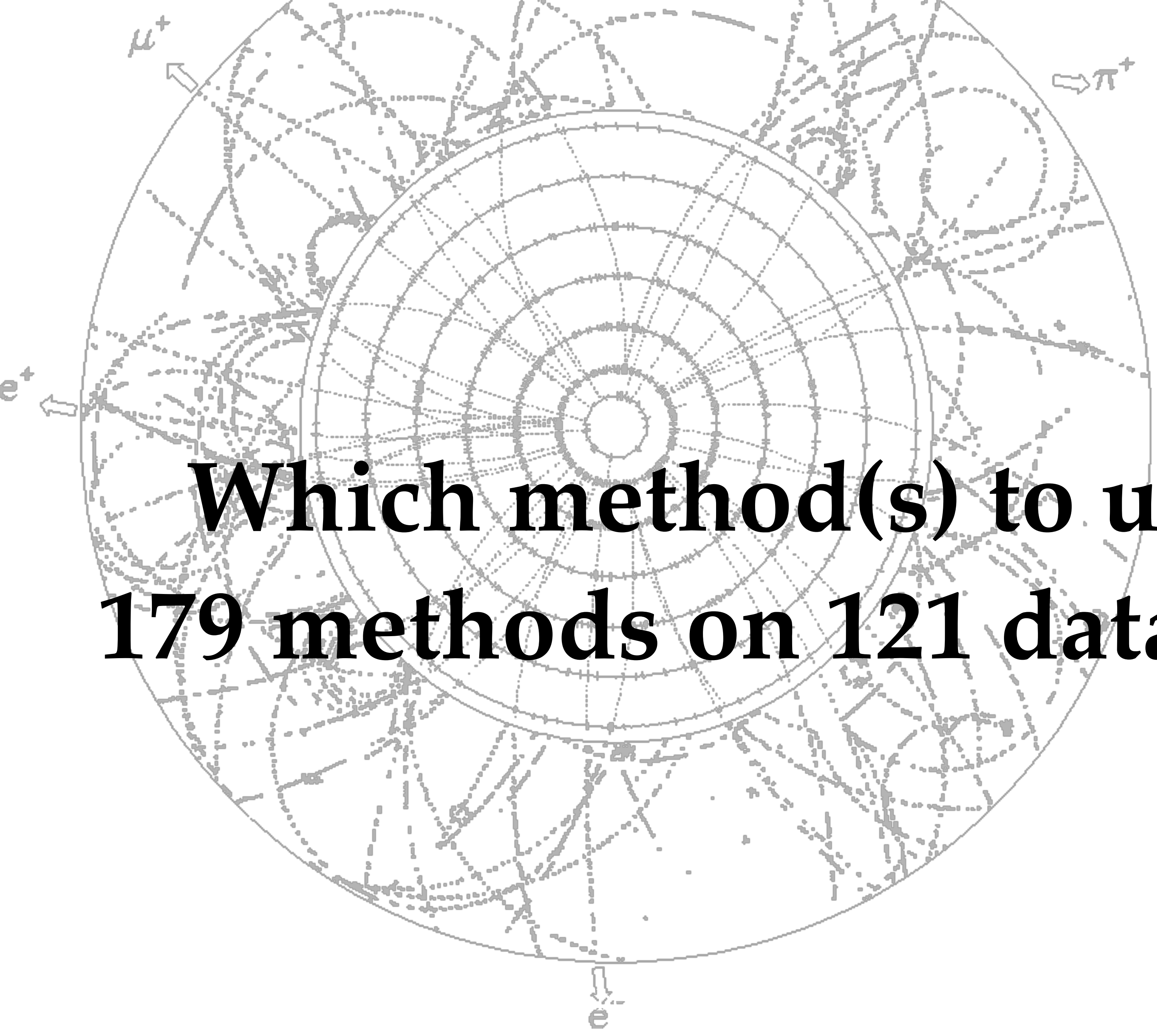
Lorin M. Hitt
University of Pennsylvania - Operations & Information Management Department

Heekyung Hellen Kim
MIT - Sloan School of Management

April 22, 2011

The study found, that there was a significant improvement going data-driven, that it was not due to reverse causality, and that the general level was…

5-6%



Which method(s) to use?
179 methods on 121 data sets

179 methods vs. 121 data sets

“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 inclusion of irrelevant features,
- produces inspectable models.
- HOWEVER… they are seldom accurate (i.e. most performant)!

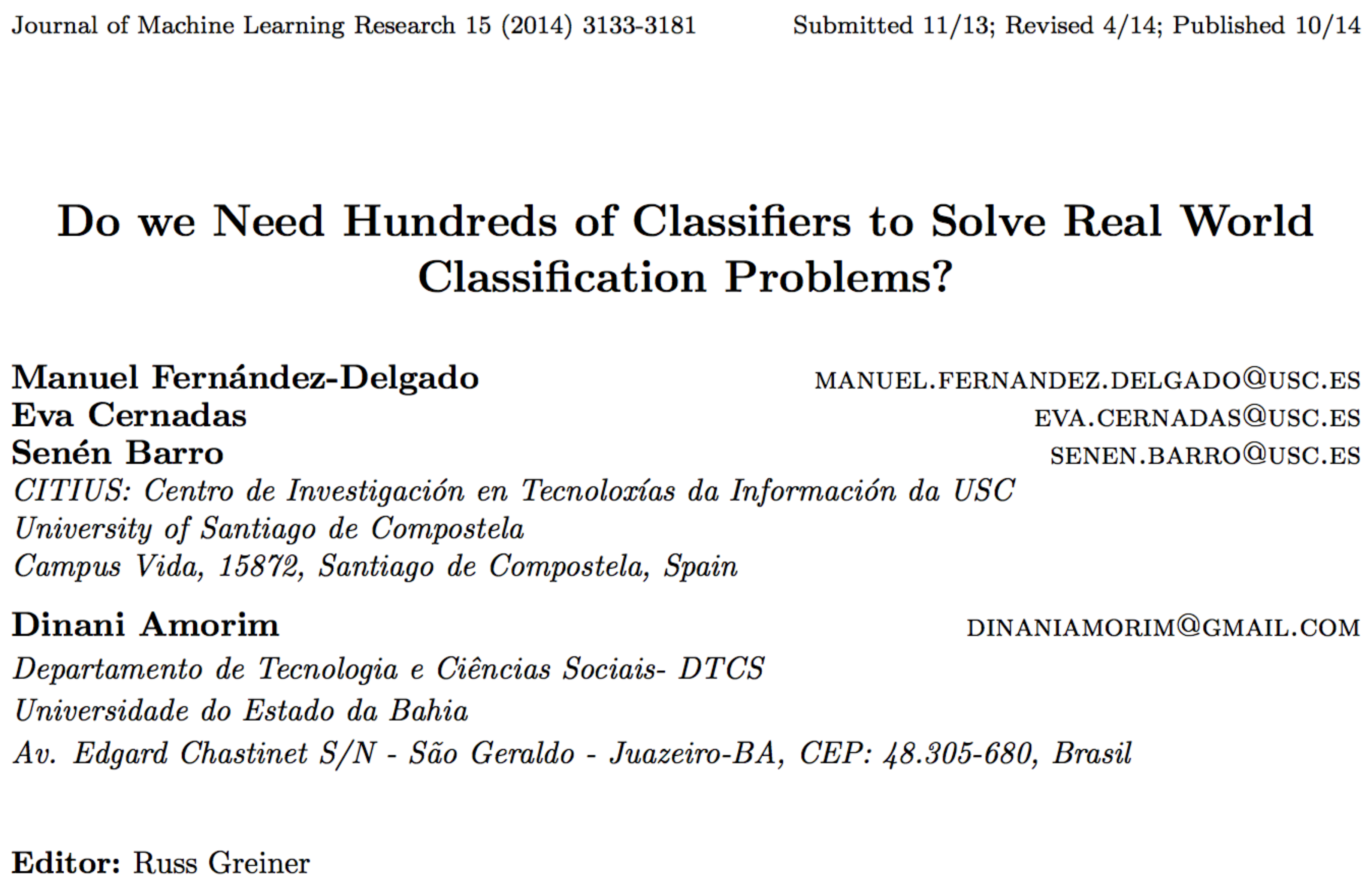
[Trevor Hastie, Professor of Mathematics & Statistics, Stanford University]

In a quite interesting paper, four authors investigated the performance of many Machine Learning (ML) methods (179 in total) on a large variety of data sets (121 in total).

The purpose was to see, if there was any general pattern, and if some type of classifiers were more suited for some problems than others.

Their findings were written up in the following paper…

179 methods vs. 121 data sets



Journal of Machine Learning Research 15 (2014) 3133-3181 Submitted 11/13; Revised 4/14; Published 10/14

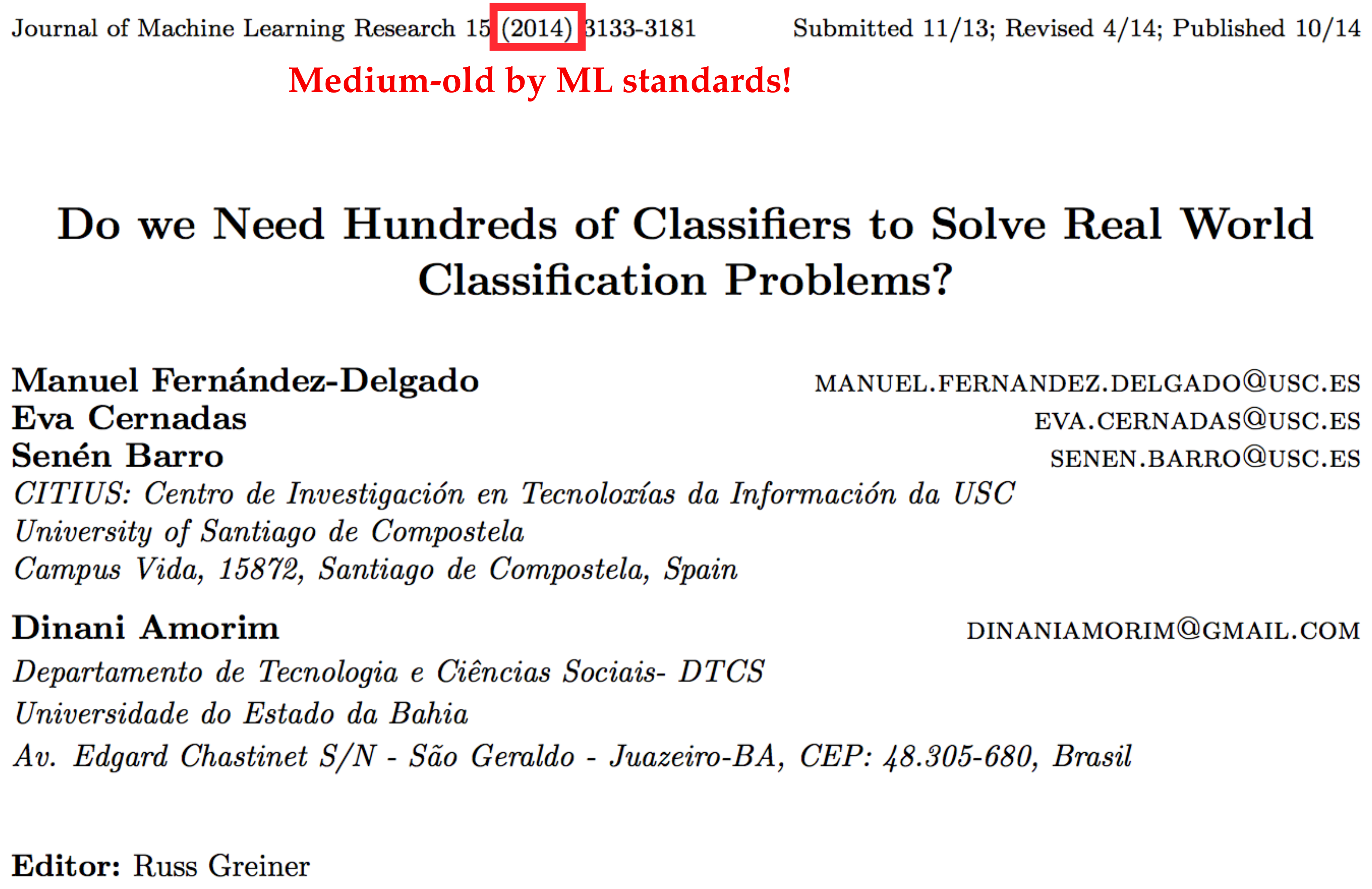
Do we Need Hundreds of Classifiers to Solve Real World Classification Problems?

Manuel Fernández-Delgado MANUEL.FERNANDEZ.DELGADO@USC.ES
Eva Cernadas EVA.CERNADAS@USC.ES
Senén Barro SENEN.BARRO@USC.ES
CITIUS: Centro de Investigación en Tecnoloxías da Información da USC
University of Santiago de Compostela
Campus Vida, 15872, Santiago de Compostela, Spain

Dinani Amorim DINANIAMORIM@GMAIL.COM
Departamento de Tecnologia e Ciências Sociais- DTCS
Universidade do Estado da Bahia
Av. Edgard Chastinet S/N - São Geraldo - Juazeiro-BA, CEP: 48.305-680, Brasil

Editor: Russ Greiner

179 methods vs. 121 data sets



Journal of Machine Learning Research 15 $\boxed{(2014)}$ 3133-3181 Submitted 11/13; Revised 4/14; Published 10/14

Medium-old by ML standards!

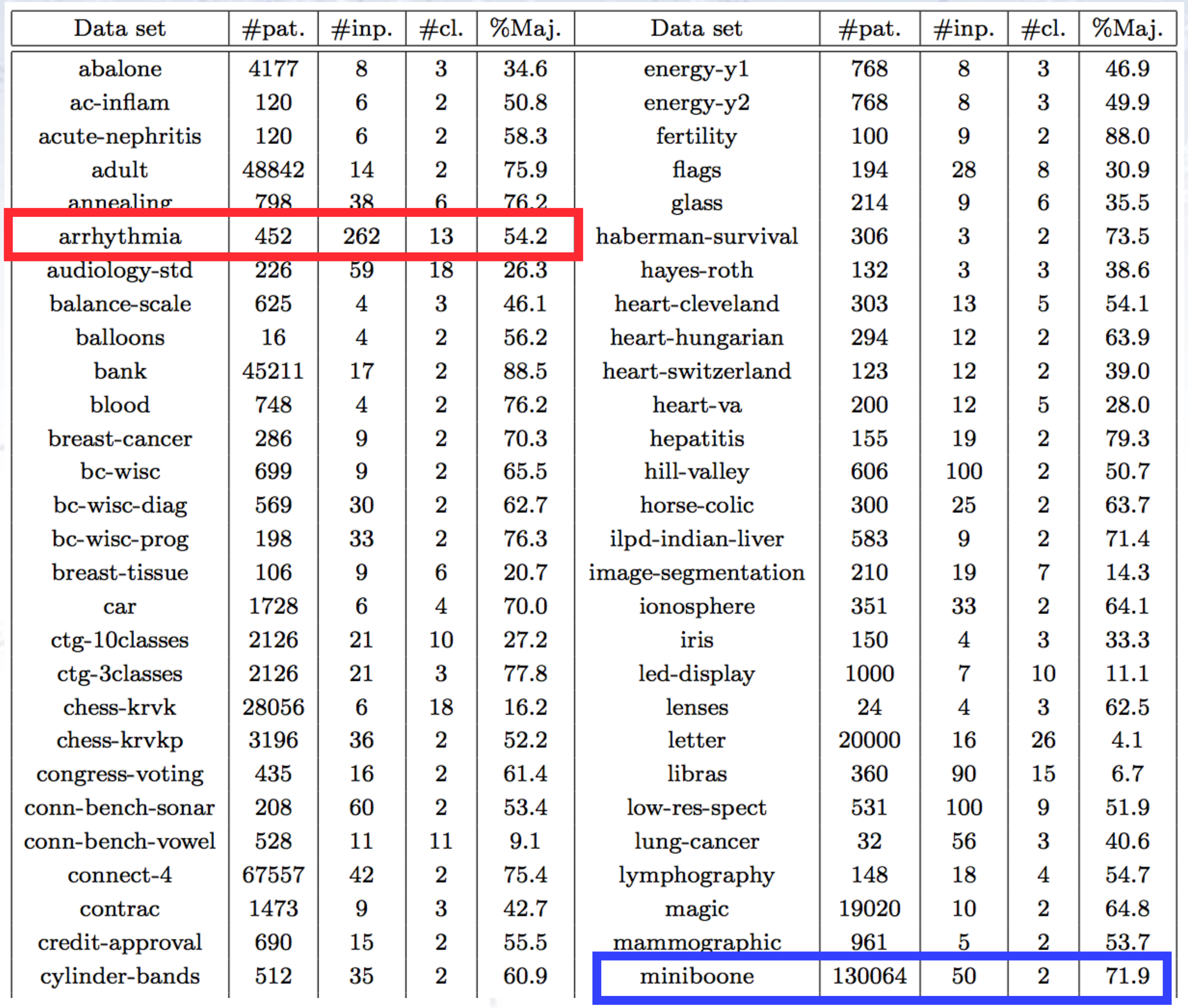
Do we Need Hundreds of Classifiers to Solve Real World Classification Problems?

Manuel Fernández-Delgado MANUEL.FERNANDEZ.DELGADO@USC.ES
Eva Cernadas EVA.CERNADAS@USC.ES
Senén Barro SENEN.BARRO@USC.ES
CITIUS: Centro de Investigación en Tecnoloxías da Información da USC
University of Santiago de Compostela
Campus Vida, 15872, Santiago de Compostela, Spain

Dinani Amorim DINANIAMORIM@GMAIL.COM
Departamento de Tecnologia e Ciências Sociais- DTCS
Universidade do Estado da Bahia
Av. Edgard Chastinet S/N - São Geraldo - Juazeiro-BA, CEP: 48.305-680, Brasil

Editor: Russ Greiner

What are the data sets?



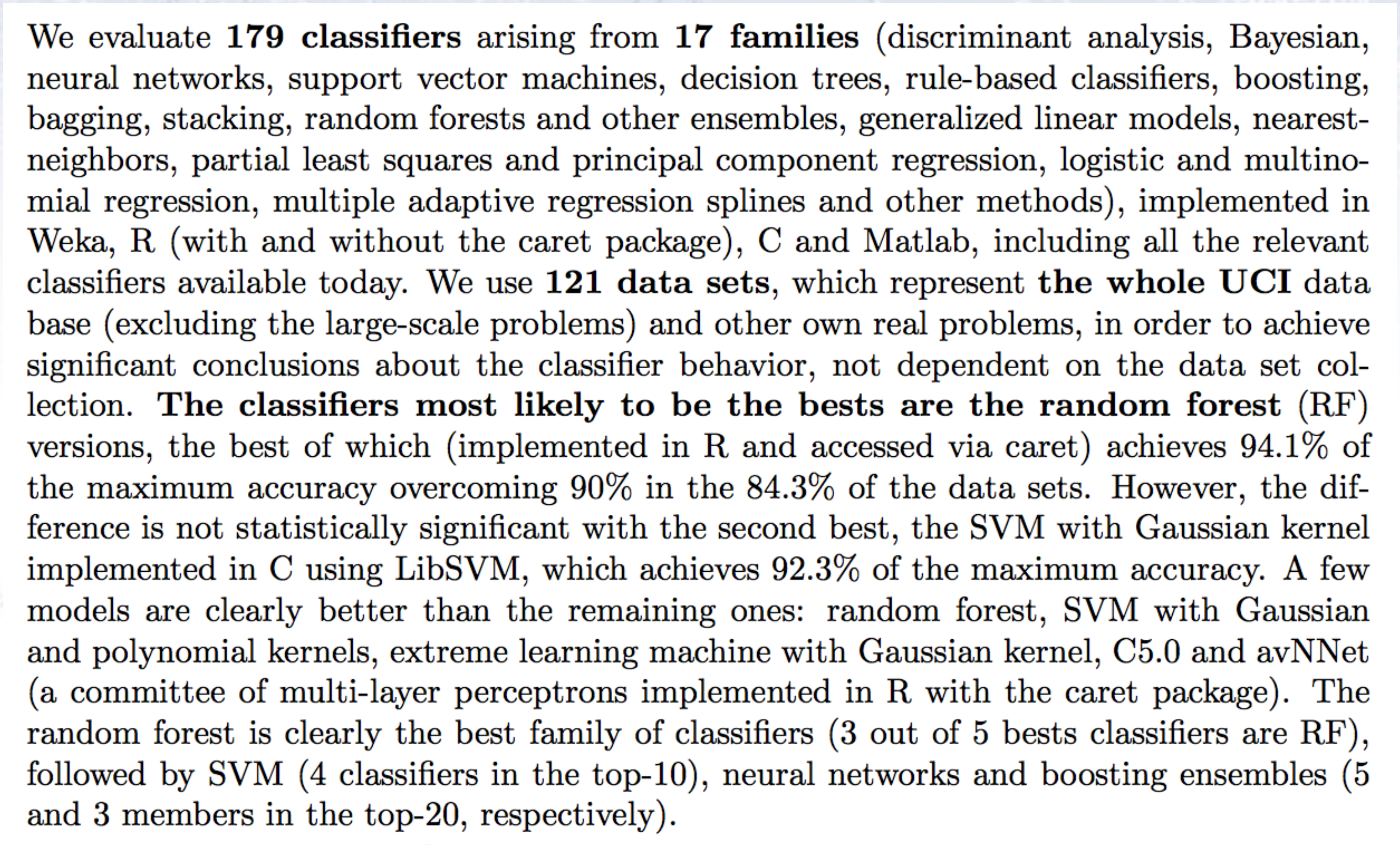
Data set	#pat.	#inp.	#cl.	%Maj.
abalone	4177	8	3	34.6
ac-inflam	120	6	2	50.8
acute-nephritis	120	6	2	58.3
adult	48842	14	2	75.9
annealing	798	38	6	76.2
arrhythmia	452	262	13	54.2
audiology-std	226	59	18	26.3
balance-scale	625	4	3	46.1
balloons	16	4	2	56.2
bank	45211	17	2	88.5
blood	748	4	2	76.2
breast-cancer	286	9	2	70.3
bc-wisc	699	9	2	65.5
bc-wisc-diag	569	30	2	62.7
bc-wisc-prog	198	33	2	76.3
breast-tissue	106	9	6	20.7
car	1728	6	4	70.0
ctg-10classes	2126	21	10	27.2
ctg-3classes	2126	21	3	77.8
chess-krvk	28056	6	18	16.2
chess-krvkp	3196	36	2	52.2
congress-voting	435	16	2	61.4
conn-bench-sonar	208	60	2	53.4
conn-bench-vowel	528	11	11	9.1
connect-4	67557	42	2	75.4
contrac	1473	9	3	42.7
credit-approval	690	15	2	55.5
cylinder-bands	512	35	2	60.9
energy-y1	768	8	3	46.9
energy-y2	768	8	3	49.9
fertility	100	9	2	88.0
flags	194	28	8	30.9
glass	214	9	6	35.5
haberman-survival	306	3	2	73.5
hayes-roth	132	3	3	38.6
heart-cleveland	303	13	5	54.1
heart-hungarian	294	12	2	63.9
heart-switzerland	123	12	2	39.0
heart-va	200	12	5	28.0
hepatitis	155	19	2	79.3
hill-valley	606	100	2	50.7
horse-colic	300	25	2	63.7
ilpd-indian-liver	583	9	2	71.4
image-segmentation	210	19	7	14.3
ionosphere	351	33	2	64.1
iris	150	4	3	33.3
led-display	1000	7	10	11.1
lenses	24	4	3	62.5
letter	20000	16	26	4.1
libras	360	90	15	6.7
low-res-spect	531	100	9	51.9
lung-cancer	32	56	3	40.6
lymphography	148	18	4	54.7
magic	19020	10	2	64.8
mammographic	961	5	2	53.7
miniboone	130064	50	2	71.9

The data sets are all quite small (< 150000 entries).

There are most often between 4-100 input parameters.

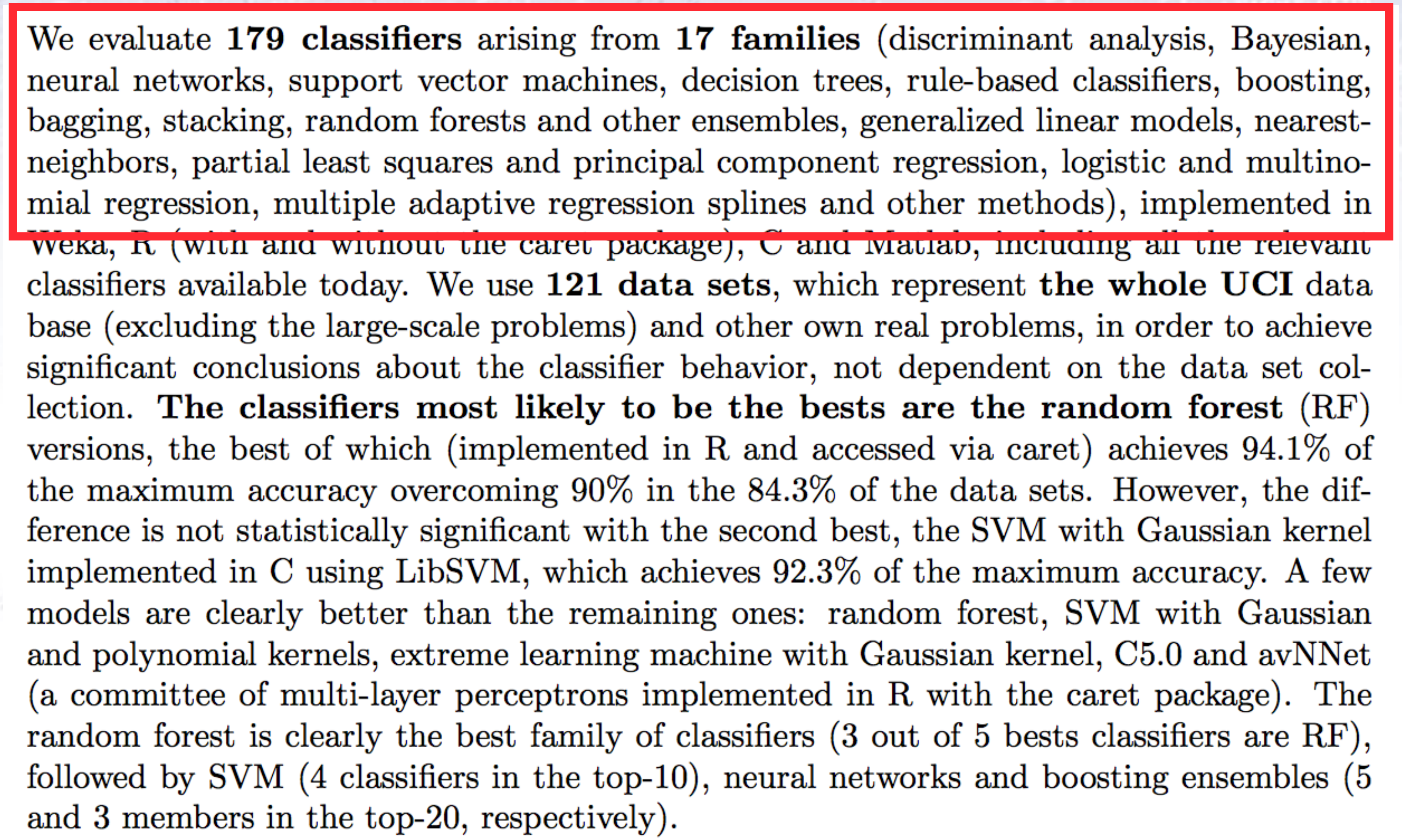
The standard problem is to divide into two classes.

179 methods vs. 121 data sets



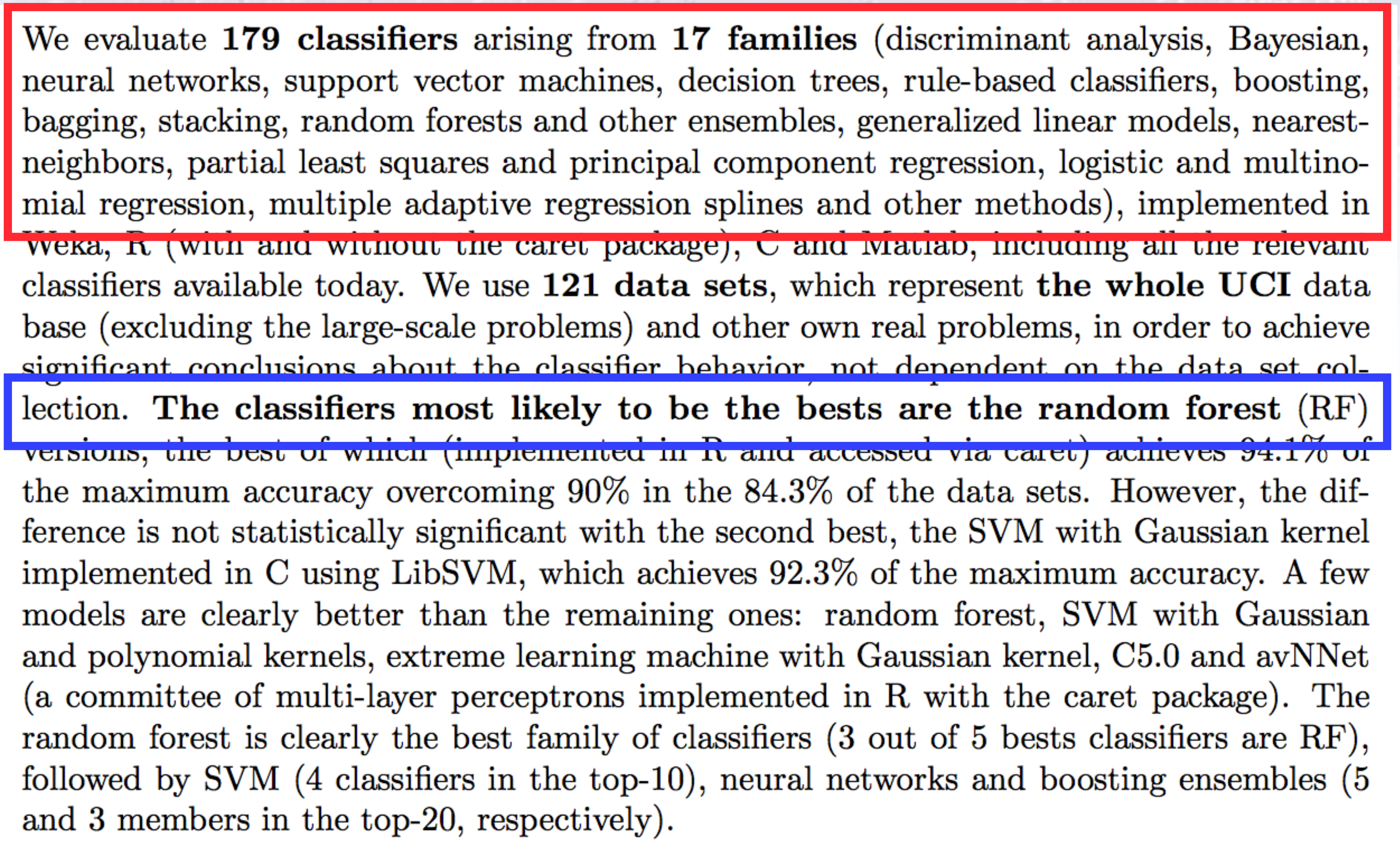
We evaluate **179 classifiers** arising from **17 families** (discriminant analysis, Bayesian, neural networks, support vector machines, decision trees, rule-based classifiers, boosting, bagging, stacking, random forests and other ensembles, generalized linear models, nearest-neighbors, partial least squares and principal component regression, logistic and multinomial regression, multiple adaptive regression splines and other methods), implemented in Weka, R (with and without the caret package), C and Matlab, including all the relevant classifiers available today. We use **121 data sets**, which represent **the whole UCI** data base (excluding the large-scale problems) and other own real problems, in order to achieve significant conclusions about the classifier behavior, not dependent on the data set collection. **The classifiers most likely to be the bests are the random forest** (RF) versions, the best of which (implemented in R and accessed via caret) achieves 94.1% of the maximum accuracy overcoming 90% in the 84.3% of the data sets. However, the difference is not statistically significant with the second best, the SVM with Gaussian kernel implemented in C using LibSVM, which achieves 92.3% of the maximum accuracy. A few models are clearly better than the remaining ones: random forest, SVM with Gaussian and polynomial kernels, extreme learning machine with Gaussian kernel, C5.0 and avNNet (a committee of multi-layer perceptrons implemented in R with the caret package). The random forest is clearly the best family of classifiers (3 out of 5 bests classifiers are RF), followed by SVM (4 classifiers in the top-10), neural networks and boosting ensembles (5 and 3 members in the top-20, respectively).

179 methods vs. 121 data sets



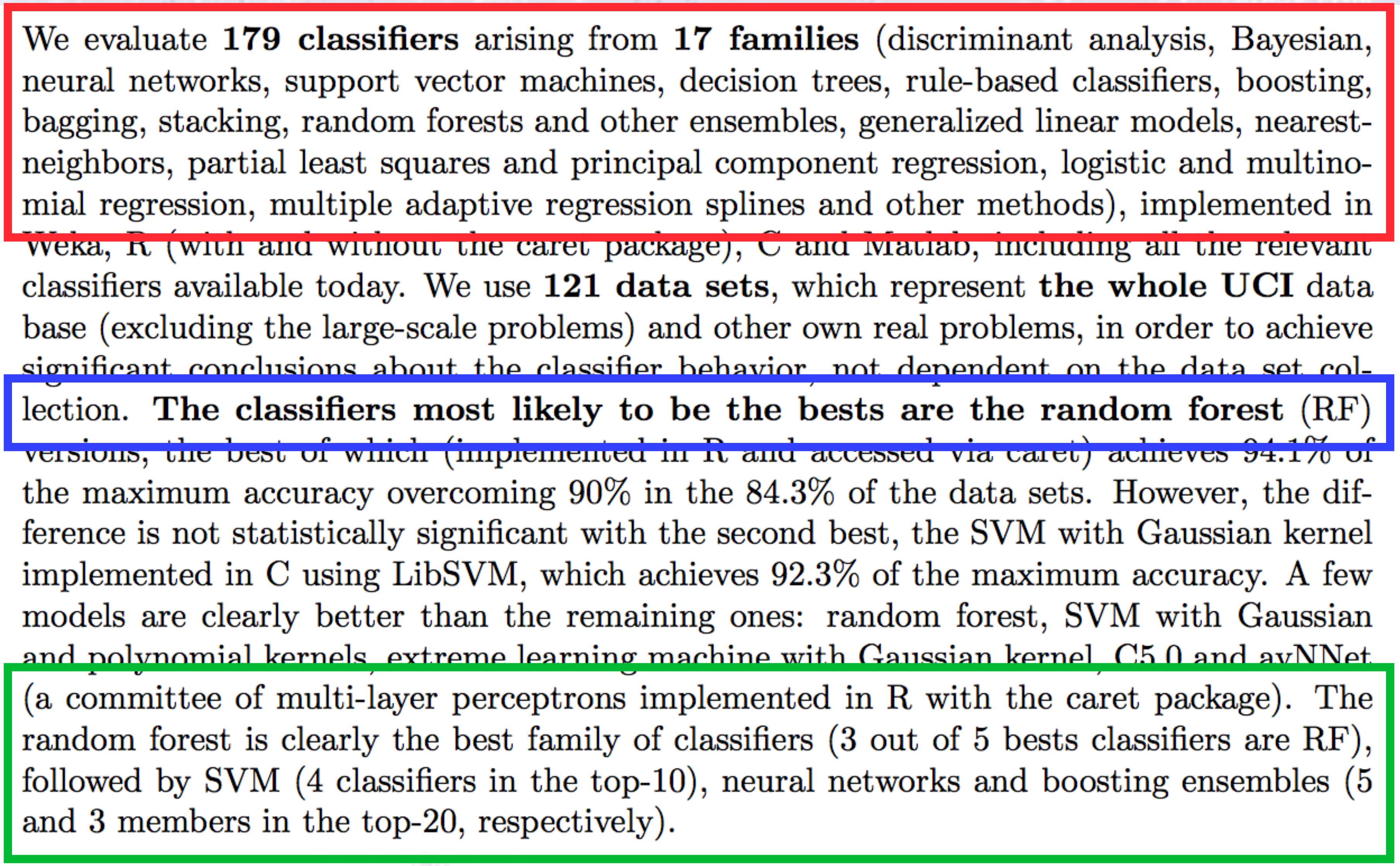
We evaluate **179 classifiers** arising from **17 families** (discriminant analysis, Bayesian, neural networks, support vector machines, decision trees, rule-based classifiers, boosting, bagging, stacking, random forests and other ensembles, generalized linear models, nearest-neighbors, partial least squares and principal component regression, logistic and multinomial regression, multiple adaptive regression splines and other methods), implemented in Weka, R (with and without the caret package), C and Matlab, including all the relevant classifiers available today. We use **121 data sets**, which represent **the whole UCI** data base (excluding the large-scale problems) and other own real problems, in order to achieve significant conclusions about the classifier behavior, not dependent on the data set collection. **The classifiers most likely to be the bests are the random forest** (RF) versions, the best of which (implemented in R and accessed via caret) achieves 94.1% of the maximum accuracy overcoming 90% in the 84.3% of the data sets. However, the difference is not statistically significant with the second best, the SVM with Gaussian kernel implemented in C using LibSVM, which achieves 92.3% of the maximum accuracy. A few models are clearly better than the remaining ones: random forest, SVM with Gaussian and polynomial kernels, extreme learning machine with Gaussian kernel, C5.0 and avNNet (a committee of multi-layer perceptrons implemented in R with the caret package). The random forest is clearly the best family of classifiers (3 out of 5 bests classifiers are RF), followed by SVM (4 classifiers in the top-10), neural networks and boosting ensembles (5 and 3 members in the top-20, respectively).

179 methods vs. 121 data sets



We evaluate **179 classifiers** arising from **17 families** (discriminant analysis, Bayesian, neural networks, support vector machines, decision trees, rule-based classifiers, boosting, bagging, stacking, random forests and other ensembles, generalized linear models, nearest-neighbors, partial least squares and principal component regression, logistic and multinomial regression, multiple adaptive regression splines and other methods), implemented in Weka, R (with and without the caret package), C and Matlab, including all the relevant classifiers available today. We use **121 data sets**, which represent **the whole UCI** data base (excluding the large-scale problems) and other own real problems, in order to achieve significant conclusions about the classifier behavior, not dependent on the data set collection. **The classifiers most likely to be the bests are the random forest** (RF) versions, the best of which (implemented in R and accessed via caret) achieves 94.1% of the maximum accuracy overcoming 90% in the 84.3% of the data sets. However, the difference is not statistically significant with the second best, the SVM with Gaussian kernel implemented in C using LibSVM, which achieves 92.3% of the maximum accuracy. A few models are clearly better than the remaining ones: random forest, SVM with Gaussian and polynomial kernels, extreme learning machine with Gaussian kernel, C5.0 and avNNet (a committee of multi-layer perceptrons implemented in R with the caret package). The random forest is clearly the best family of classifiers (3 out of 5 bests classifiers are RF), followed by SVM (4 classifiers in the top-10), neural networks and boosting ensembles (5 and 3 members in the top-20, respectively).

179 methods vs. 121 data sets



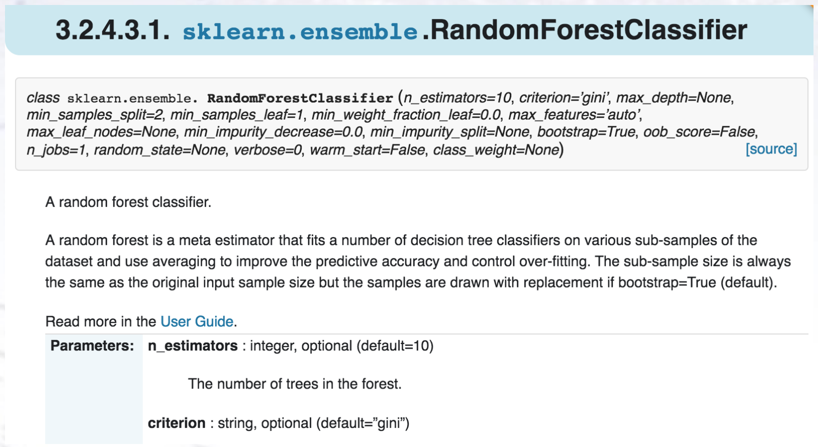
We evaluate **179 classifiers** arising from **17 families** (discriminant analysis, Bayesian, neural networks, support vector machines, decision trees, rule-based classifiers, boosting, bagging, stacking, random forests and other ensembles, generalized linear models, nearest-neighbors, partial least squares and principal component regression, logistic and multinomial regression, multiple adaptive regression splines and other methods), implemented in Weka, R (with and without the caret package), C and Matlab, including all the relevant classifiers available today. We use **121 data sets**, which represent **the whole UCI** data base (excluding the large-scale problems) and other own real problems, in order to achieve significant conclusions about the classifier behavior, not dependent on the data set collection. **The classifiers most likely to be the bests are the random forest** (RF) versions, the best of which (implemented in R and accessed via caret) achieves 94.1% of the maximum accuracy overcoming 90% in the 84.3% of the data sets. However, the difference is not statistically significant with the second best, the SVM with Gaussian kernel implemented in C using LibSVM, which achieves 92.3% of the maximum accuracy. A few models are clearly better than the remaining ones: random forest, SVM with Gaussian and polynomial kernels, extreme learning machine with Gaussian kernel, C5.0 and avNNet (a committee of multi-layer perceptrons implemented in R with the caret package). The random forest is clearly the best family of classifiers (3 out of 5 bests classifiers are RF), followed by SVM (4 classifiers in the top-10), neural networks and boosting ensembles (5 and 3 members in the top-20, respectively).

Random Forests implementations

Given the succes of the RandomForests algorithm, it has naturally been implemented in many languages (the original one being Fortran!!!).

I managed to find it in both Python and R:

Python: scikit-learn package



3.2.4.3.1. `sklearn.ensemble`.RandomForestClassifier

class `sklearn.ensemble.` **`RandomForestClassifier`**(*n_estimators=10, criterion='gini', max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=None, verbose=0, warm_start=False, class_weight=None*) [source]

A random forest classifier.

A random forest is a meta estimator that fits a number of decision tree classifiers on various sub-samples of the dataset and use averaging to improve the predictive accuracy and control over-fitting. The sub-sample size is always the same as the original input sample size but the samples are drawn with replacement if bootstrap=True (default).

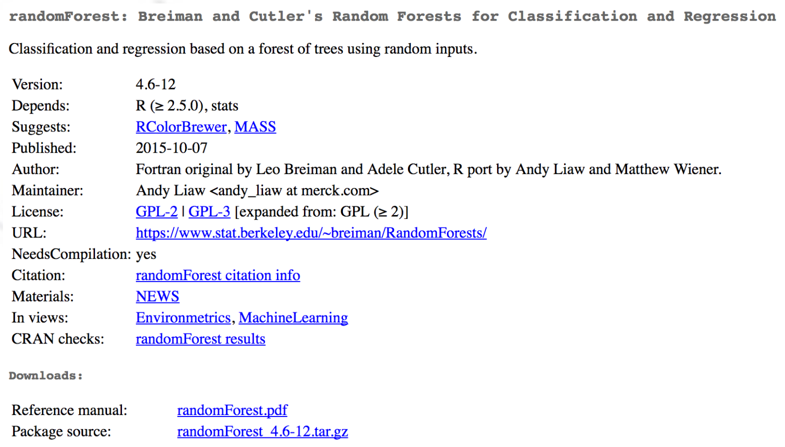
Read more in the User Guide.

Parameters: **n_estimators** : integer, optional (default=10)

The number of trees in the forest.

criterion : string, optional (default="gini")

R: randomForests package



`randomForest: Breiman and Cutler's Random Forests for Classification and Regression`

Classification and regression based on a forest of trees using random inputs.

Version:	4.6-12
Depends:	R (≥ 2.5.0), stats
Suggests:	RColorBrewer, MASS
Published:	2015-10-07
Author:	Fortran original by Leo Breiman and Adele Cutler, R port by Andy Liaw and Matthew Wiener.
Maintainer:	Andy Liaw <andy_liaw at merck.com>
License:	GPL-2 | GPL-3 [expanded from: GPL (≥ 2)]
URL:	https://www.stat.berkeley.edu/~breiman/RandomForests/
NeedsCompilation:	yes
Citation:	randomForest citation info
Materials:	NEWS
In views:	Environmetrics, MachineLearning
CRAN checks:	randomForest results

`Downloads:`

Reference manual:	randomForest.pdf
Package source:	randomForest_4.6-12.tar.gz

Random Forests implementations

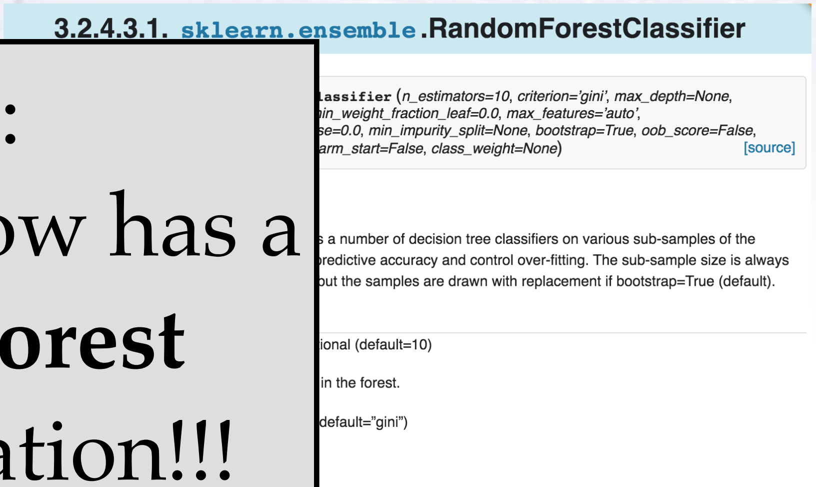
Given the succes of the RandomForests algorithm, it has naturally been implemented in many languages (the original one being Fortran!!!).

I managed to find it in both Python and R:

Python: scikit-learn p

News:
LightGBM now has a
RandomForest
implementation!!!

R: randomForests package



3.2.4.3.1. sklearn.ensemble.RandomForestClassifier

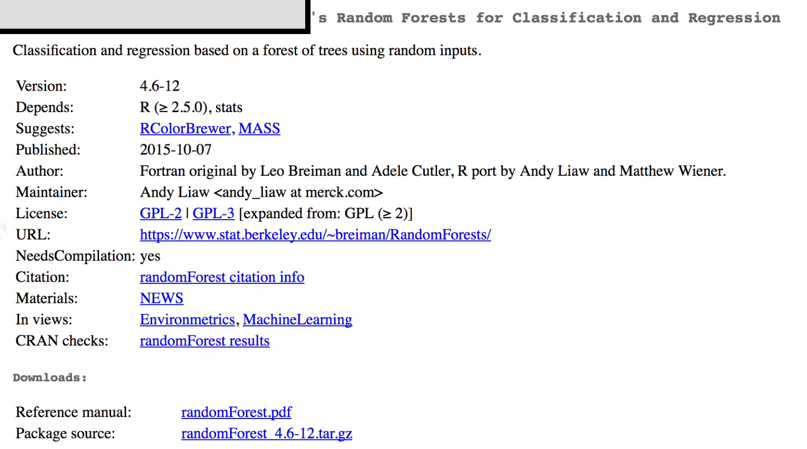
```
lassifier (n_estimators=10, criterion='gini', max_depth=None,
in_weight_fraction_leaf=0.0, max_features='auto',
se=0.0, min_impurity_split=None, bootstrap=True, oob_score=False,
arm_start=False, class_weight=None)                    [source]
```

s a number of decision tree classifiers on various sub-samples of the
predictive accuracy and control over-fitting. The sub-sample size is always
out the samples are drawn with replacement if bootstrap=True (default).

ional (default=10)

in the forest.

default="gini")



's Random Forests for Classification and Regression

Classification and regression based on a forest of trees using random inputs.

Version:	4.6-12
Depends:	R (≥ 2.5.0), stats
Suggests:	RColorBrewer, MASS
Published:	2015-10-07
Author:	Fortran original by Leo Breiman and Adele Cutler, R port by Andy Liaw and Matthew Wiener.
Maintainer:	Andy Liaw <andy_liaw at merck.com>
License:	GPL-2 | GPL-3 [expanded from: GPL (≥ 2)]
URL:	https://www.stat.berkeley.edu/~breiman/RandomForests/
NeedsCompilation:	yes
Citation:	randomForest citation info
Materials:	NEWS
In views:	Environmetrics, MachineLearning
CRAN checks:	randomForest results

Downloads:

Reference manual:	randomForest.pdf
Package source:	randomForest_4.6-12.tar.gz

The results in more detail…

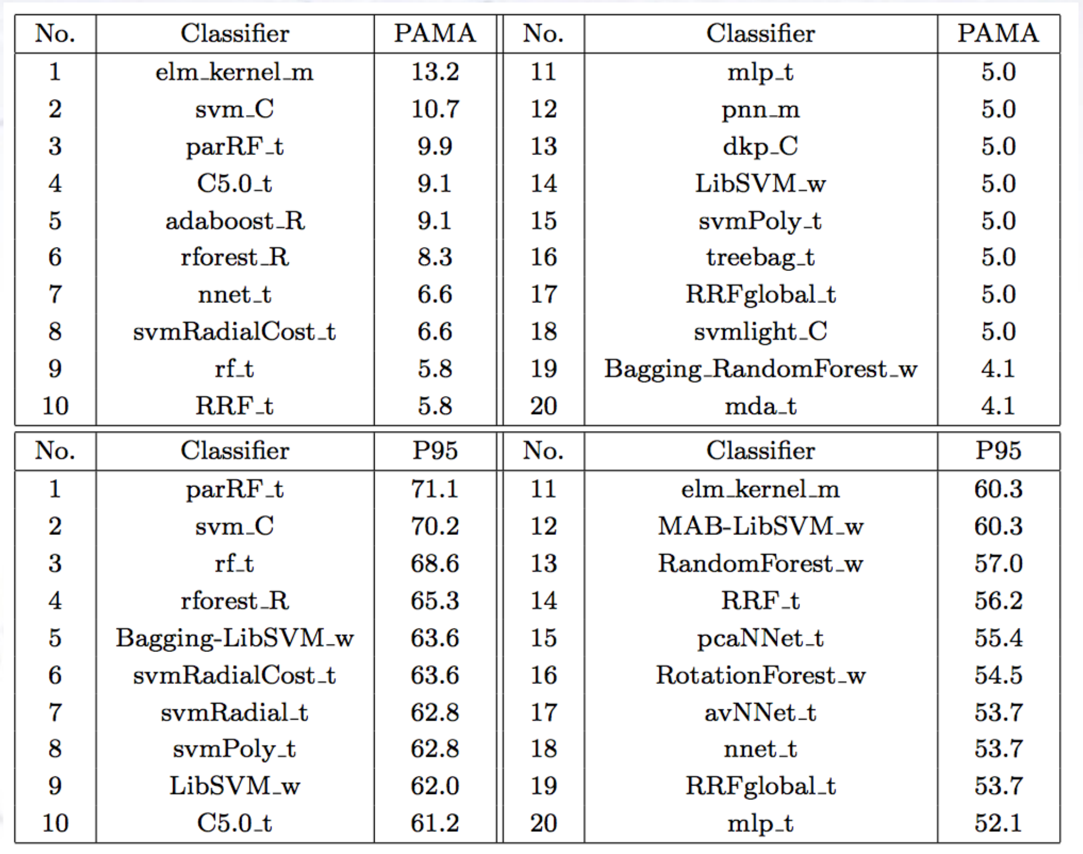
The many good algorithms are ranked according to probability of achieving:

- Maximum Accuracy (PAMA)
- 95% accuracy on all data sets (P95)

As can be seen, the Random Forest (parRF_t) is not the most likely to be the best.
Rather it is the one, which most often is ranked high.

But this just shows, that there is no guarantee that parRF_t is the most powerful method. In fact far from it.

This is a general problem, which must be considered…



No.	Classifier	PAMA	No.	Classifier	PAMA
1	elm_kernel_m	13.2	11	mlp_t	5.0
2	svm_C	10.7	12	pnn_m	5.0
3	parRF_t	9.9	13	dkp_C	5.0
4	C5.0_t	9.1	14	LibSVM_w	5.0
5	adaboost_R	9.1	15	svmPoly_t	5.0
6	rforest_R	8.3	16	treebag_t	5.0
7	nnet_t	6.6	17	RRFglobal_t	5.0
8	svmRadialCost_t	6.6	18	svmlight_C	5.0
9	rf_t	5.8	19	Bagging_RandomForest_w	4.1
10	RRF_t	5.8	20	mda_t	4.1

No.	Classifier	P95	No.	Classifier	P95
1	parRF_t	71.1	11	elm_kernel_m	60.3
2	svm_C	70.2	12	MAB-LibSVM_w	60.3
3	rf_t	68.6	13	RandomForest_w	57.0
4	rforest_R	65.3	14	RRF_t	56.2
5	Bagging-LibSVM_w	63.6	15	pcaNNet_t	55.4
6	svmRadialCost_t	63.6	16	RotationForest_w	54.5
7	svmRadial_t	62.8	17	avNNet_t	53.7
8	svmPoly_t	62.8	18	nnet_t	53.7
9	LibSVM_w	62.0	19	RRFglobal_t	53.7
10	C5.0_t	61.2	20	mlp_t	52.1