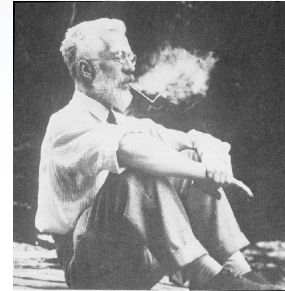
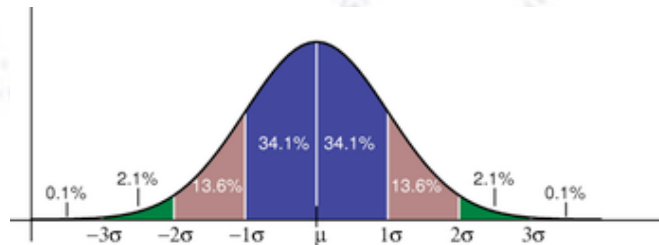


# Applied Statistics

## Introduction to Machine Learning



Troels C. Petersen (NBI)



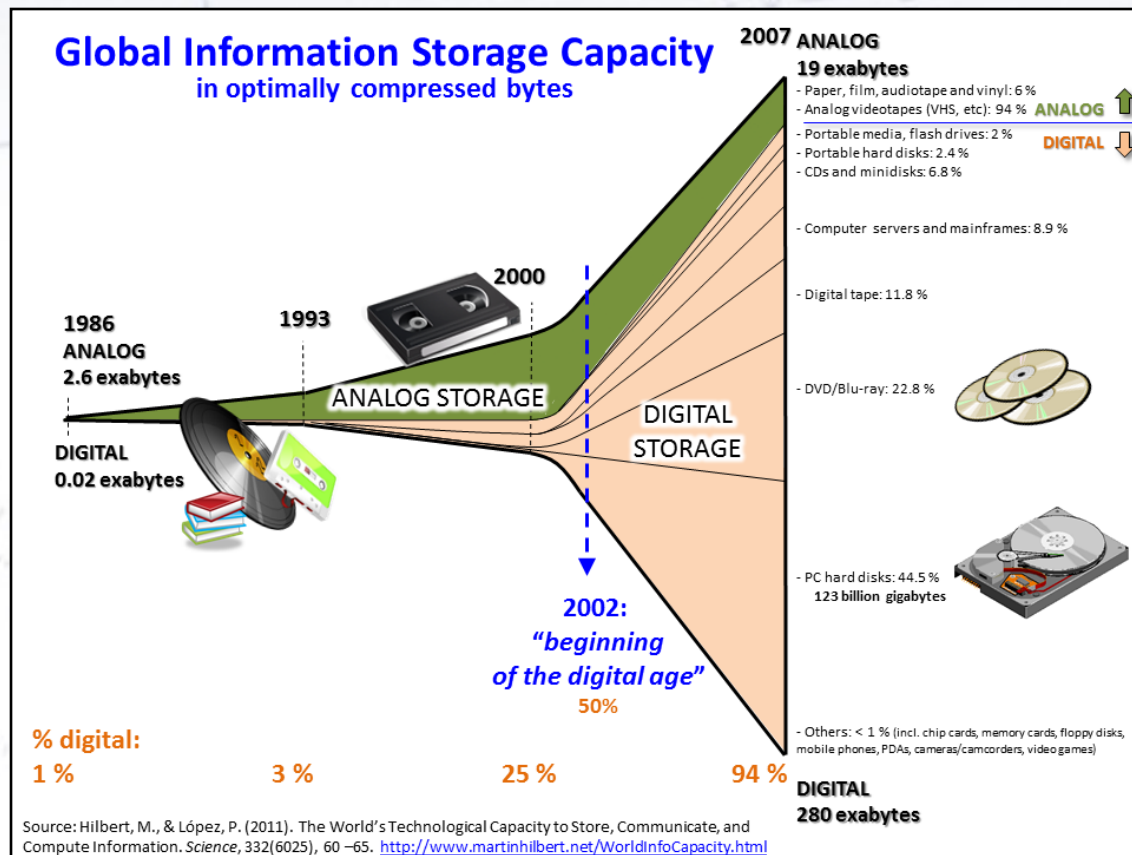
*“Statistics is merely a quantisation of common sense. Machine Learning is a sharpening of it!”*



# Why ML?

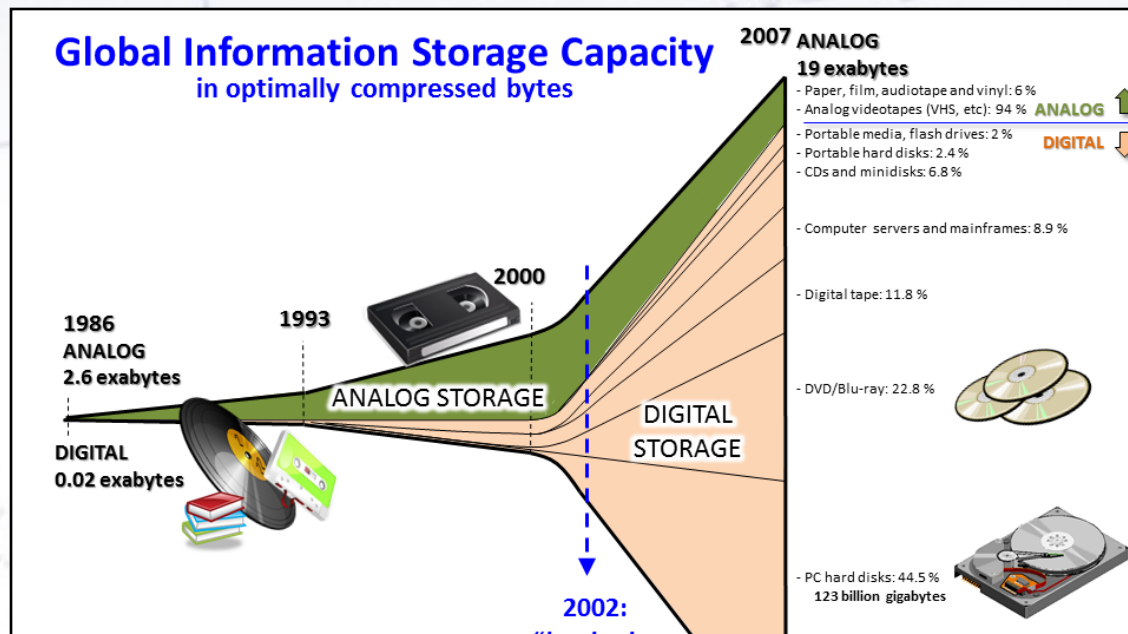
# Why Machine Learning?

Part of the “rising” of Machine Learning has been the explosion in data volume, and the easy access to mine it (i.e. internet-of-things), but also the growth in data storage and processing capabilities.



# Why Machine Learning?

Part of the “rising” of Machine Learning has been the explosion in data volume, and the easy access to mine it (i.e. internet-of-things), but also the growth in data storage and processing capabilities.



**In a digital world, both academia and businesses have an advantage in understanding their (growing) data volumes. Machine Learning is a powerful tool to do exactly that!**

# Dimensionality and Complexity

Humans are good at seeing/understanding data in few dimensions!

However, as dimensionality grows, complexity grows exponentially (“curse of dimensionality”), and humans are generally not geared for such challenges.

	Low dim.	High dim.
Linear	Humans: Computers:	Humans: Computers:
Non-linear	Humans: Computers:	Humans: Computers:

Computers, on the other hand, are OK with high dimensionality, albeit the growth of the challenge, but have a harder time facing non-linear issues.

However, through smart algorithms, computers have learned to deal with it all!

**That is essentially what Machine Learning has enabled!**

# Dimensionality and Complexity

Humans are good at seeing/understanding data in few dimensions!

However, as dimensionality grows, complexity grows exponentially (“curse of dimensionality”), and humans are generally not geared for such challenges.

	Low dim.	High dim.
Linear	Humans: ✓ Computers: ✓	Humans: ÷ Computers: ✓
Non-linear	Humans: Computers:	Humans: Computers:

Computers, on the other hand, are OK with high dimensionality, albeit the growth of the challenge, but have a harder time facing non-linear issues.

However, through smart algorithms, computers have learned to deal with it all!

**That is essentially what Machine Learning has enabled!**

# Dimensionality and Complexity

Humans are good at seeing/understanding data in few dimensions!

However, as dimensionality grows, complexity grows exponentially (“curse of dimensionality”), and humans are generally not geared for such challenges.

	Low dim.	High dim.
Linear	Humans: ✓ Computers: ✓	Humans: ÷ Computers: ✓
Non-linear	Humans: ✓ Computers: (✓)	Humans: ÷ Computers: (✓)

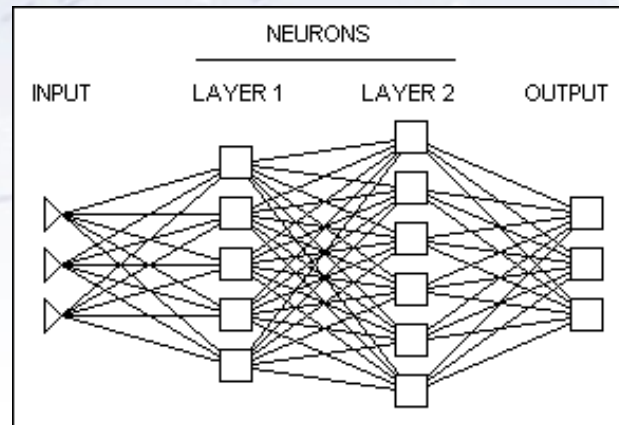
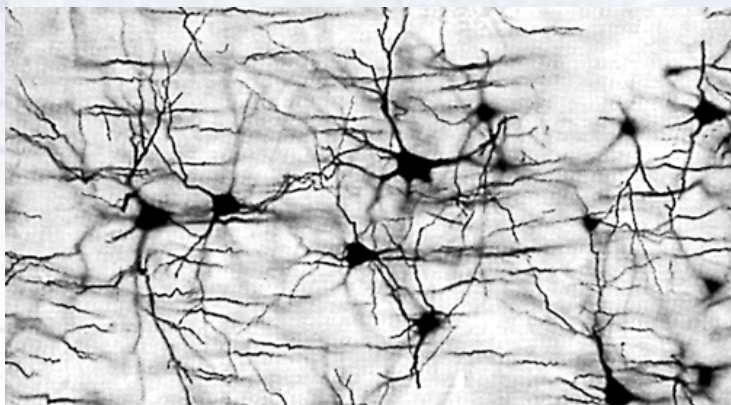
Computers, on the other hand, are OK with high dimensionality, albeit the growth of the challenge, but have a harder time facing non-linear issues.

However, through smart algorithms, computers have learned to deal with it all!

**That is essentially what Machine Learning has enabled!**

# Data Mining

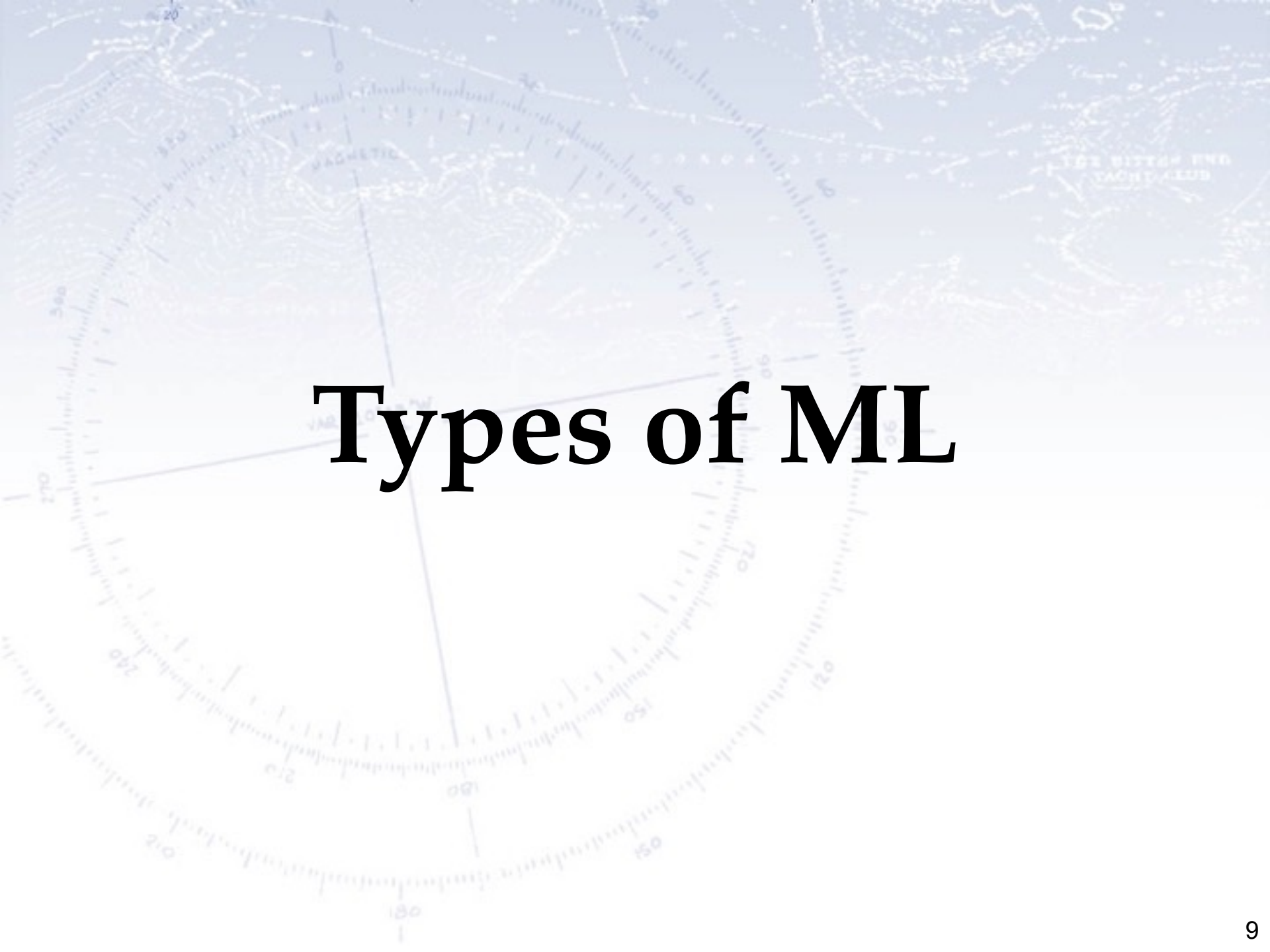
Seeing patterns in data and using it!



*Data mining is the process of extracting patterns from data. As more data are gathered, with the amount of data doubling every three years, data mining is becoming an increasingly important tool to transform these data into information. It is commonly used in a wide range of profiling practices, such as marketing, surveillance, fraud detection and **scientific discovery**.*

[Wikipedia, Introduction to Data Mining]

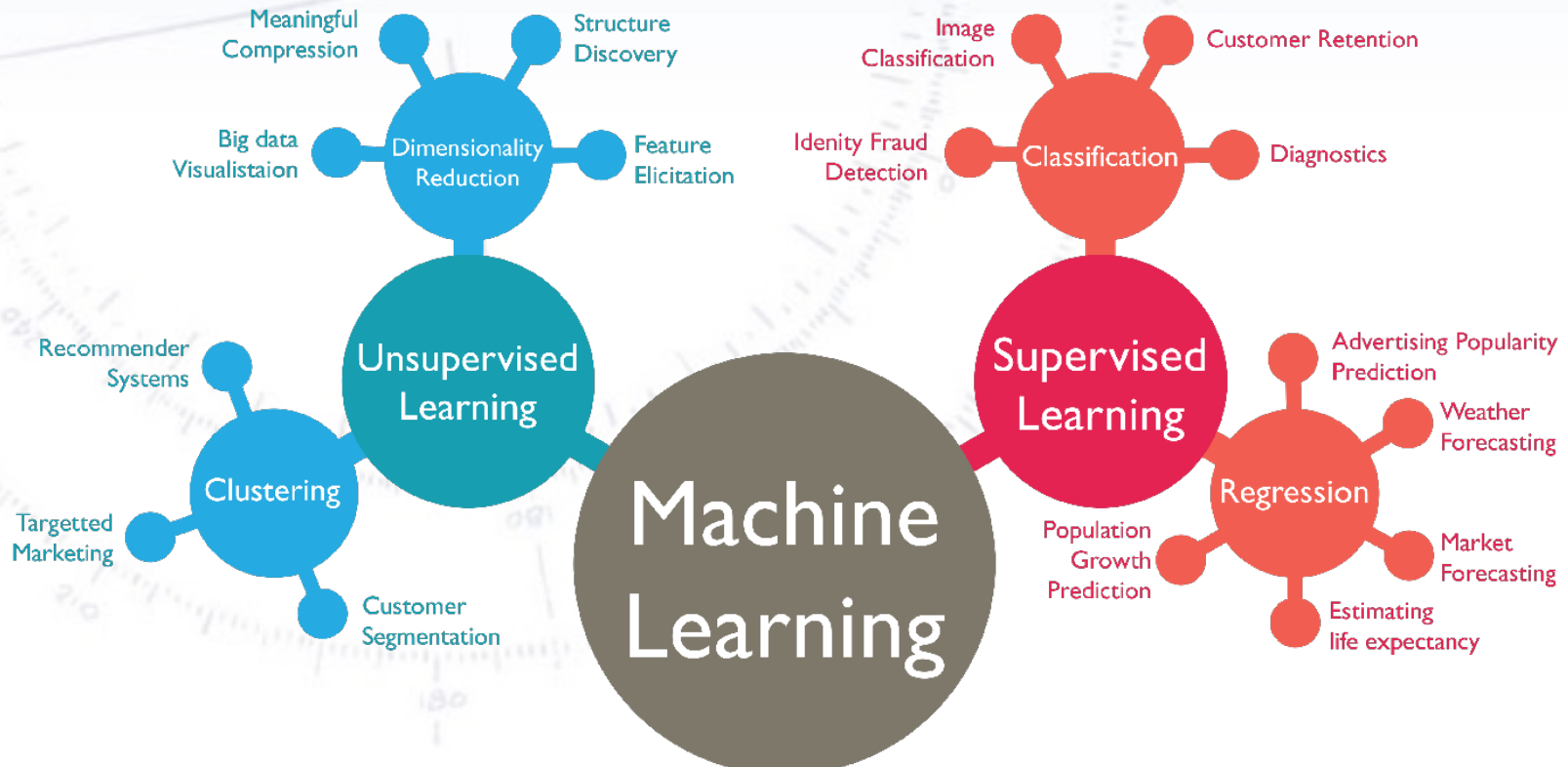


The background is a faded nautical chart. It features concentric lines representing magnetic isotherms, with labels such as 'MAGNETIC' and 'VAR. 10' visible. A compass rose is partially visible, showing degree markings like 0, 30, 60, 90, 120, 150, 180, 210, 240, and 270. The text '1ST BITTER END TACHTALUB' is also present in the upper right quadrant of the chart.

# Types of ML

# Unsupervised vs. Supervised Classification vs. Regression

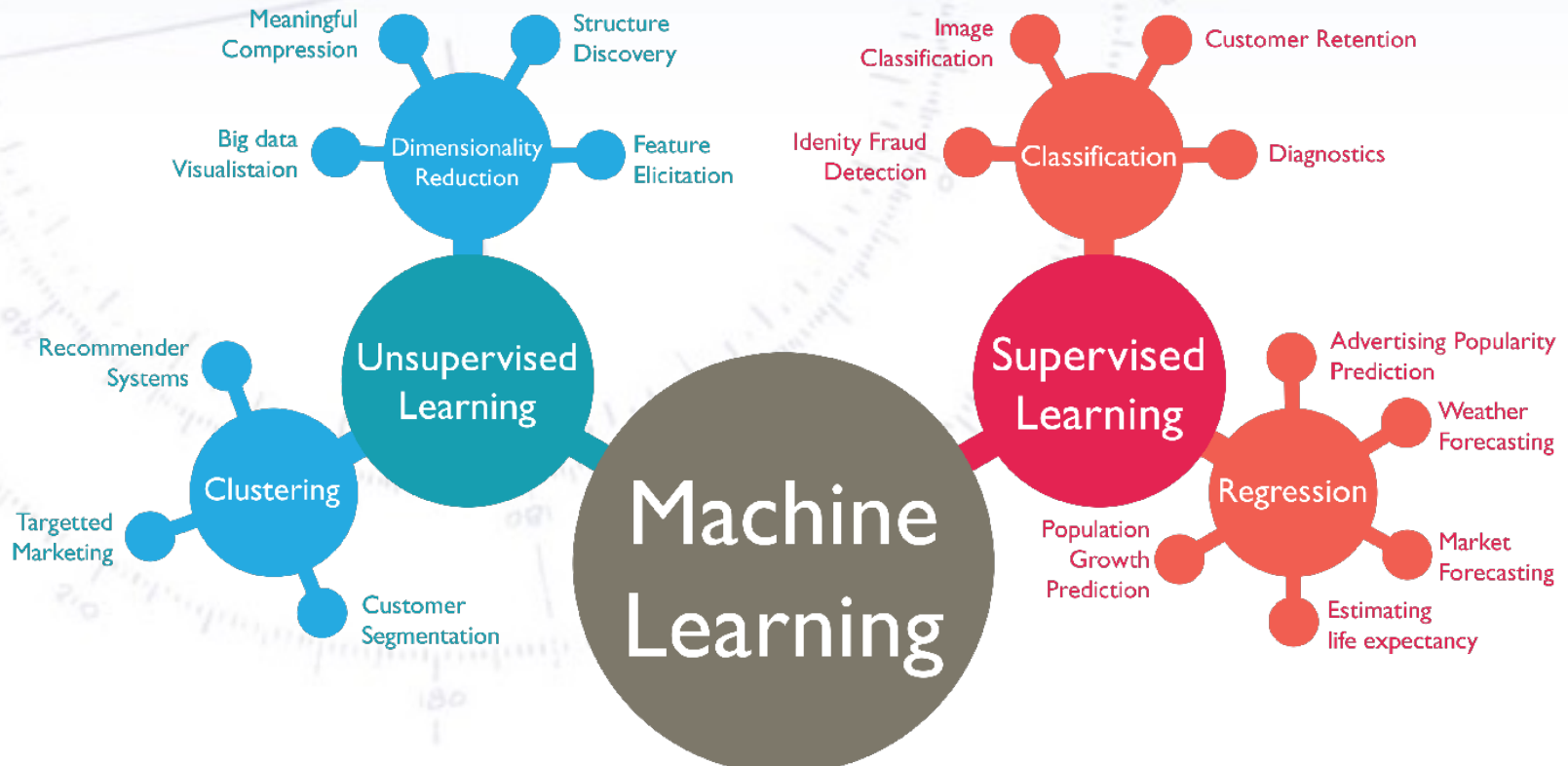
Machine Learning can be supervised (you have correctly labelled examples) or unsupervised (you don't)... [or reinforced]. Following this, one can be using ML to either classify (is it A or B?) or for regression (estimate of X).



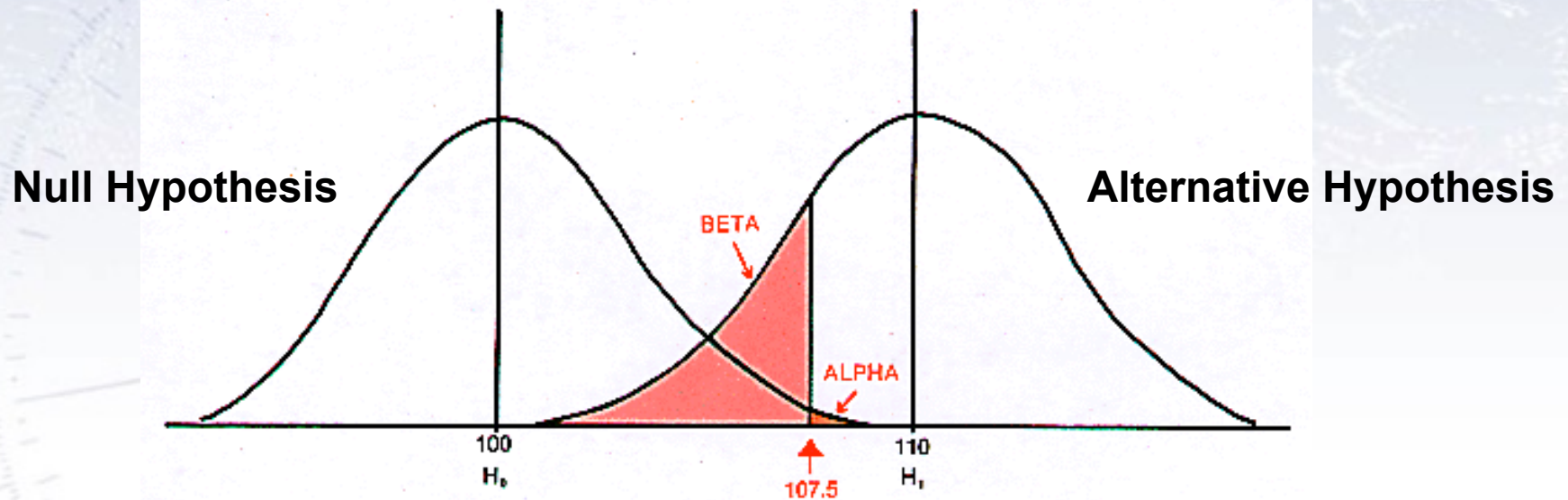
# Unsupervised vs. Supervised Classification vs. Regression

Machine Learning can be supervised (you have correctly labelled examples) or unsupervised (you don't)... [or reinforced]. Following this, one can be using ML to either classify (is it A or B?) or for regression (estimate of X).

**We will be mostly on this side!**

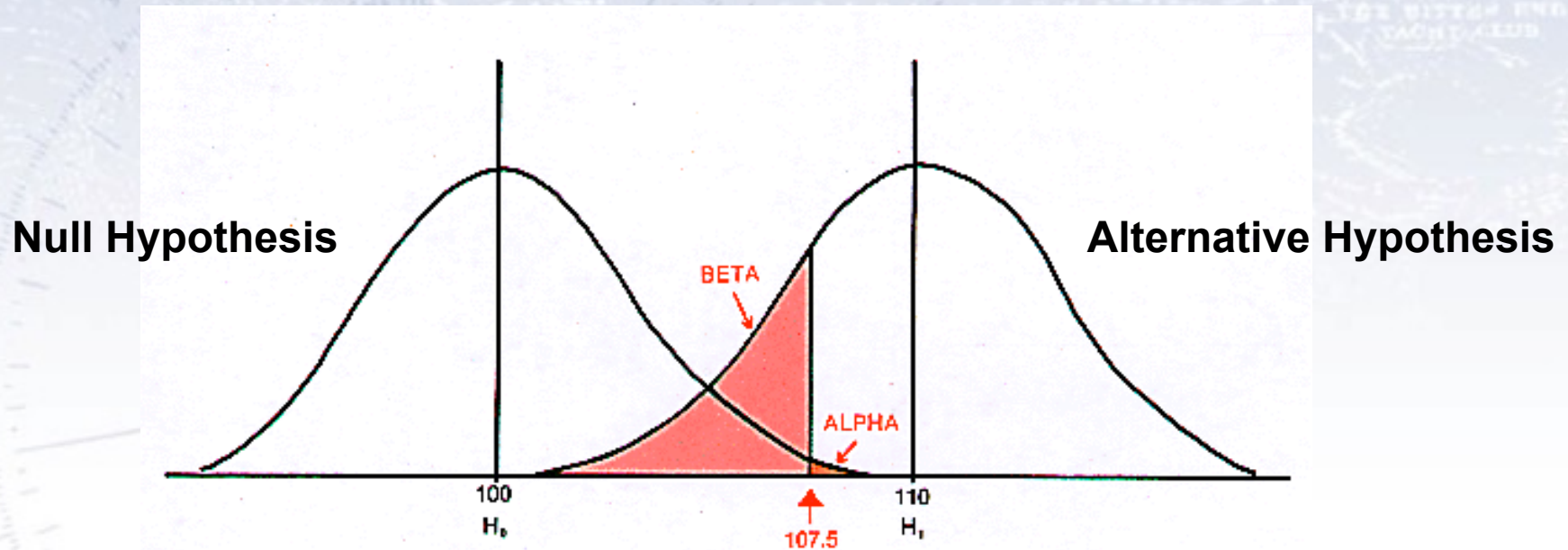


# Classification/Hypothesis



		REALITY	
		Null is True	Null is False
STATISTICAL DECISION:	Do Not Reject Null	$1 - \alpha$ Correct	$\beta$ Type II error
	Reject Null	$\alpha$ Type I error	$1 - \beta$ Correct

# Classification/Hypothesis



Machine Learning typically enables a better separation between hypothesis

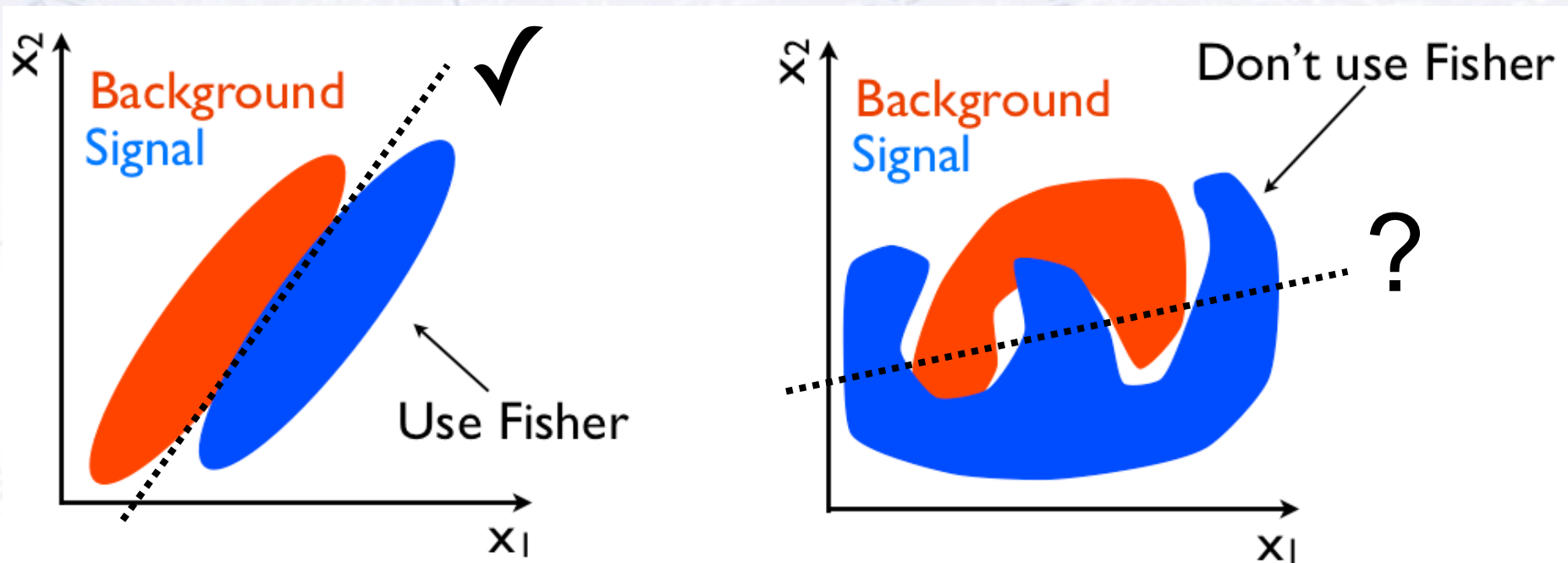
**DECISION:**

Reject Null

$\alpha$ Type I error	$1 - \beta$ Correct
--------------------------	------------------------

# 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...

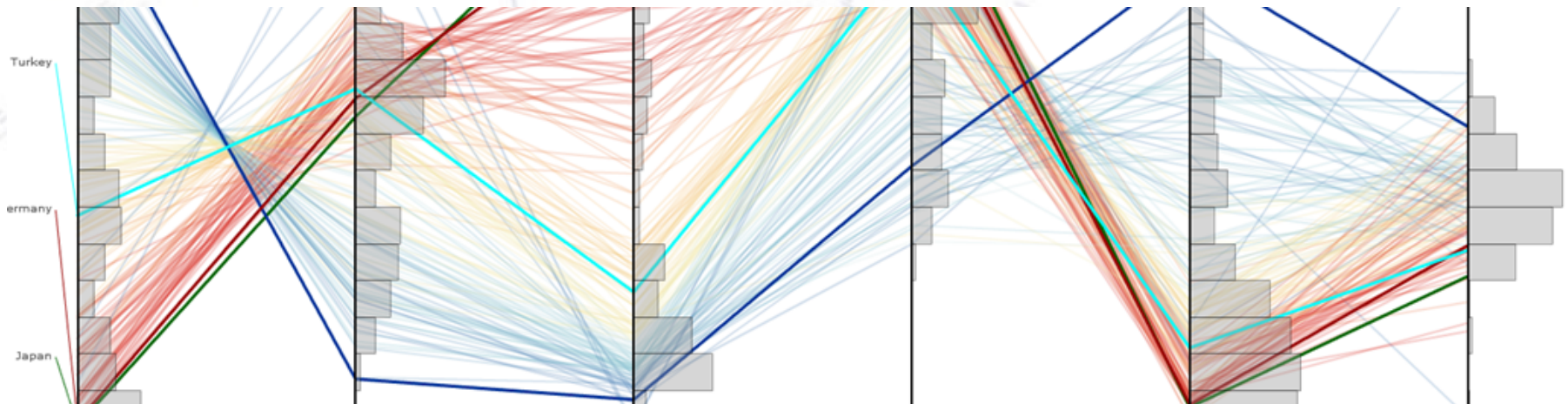
# Today's goal: Introduction

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

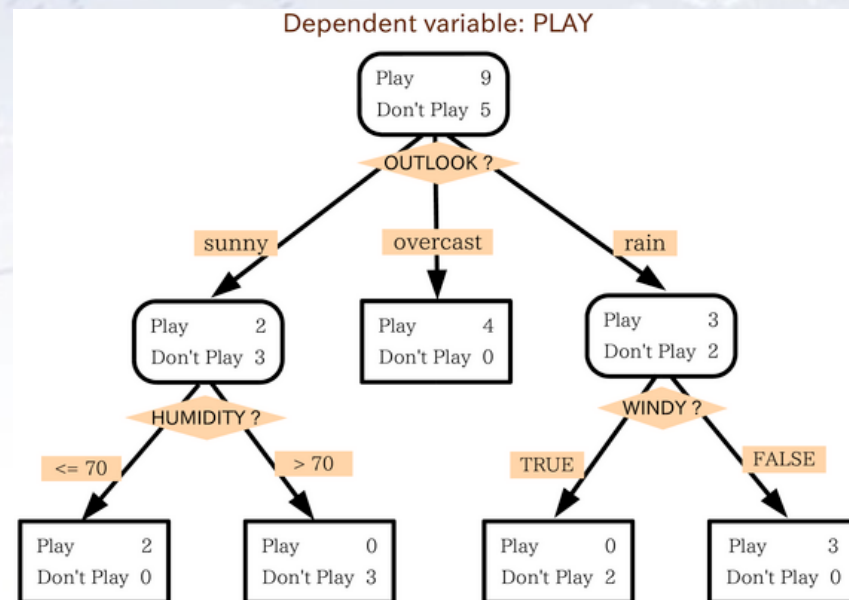
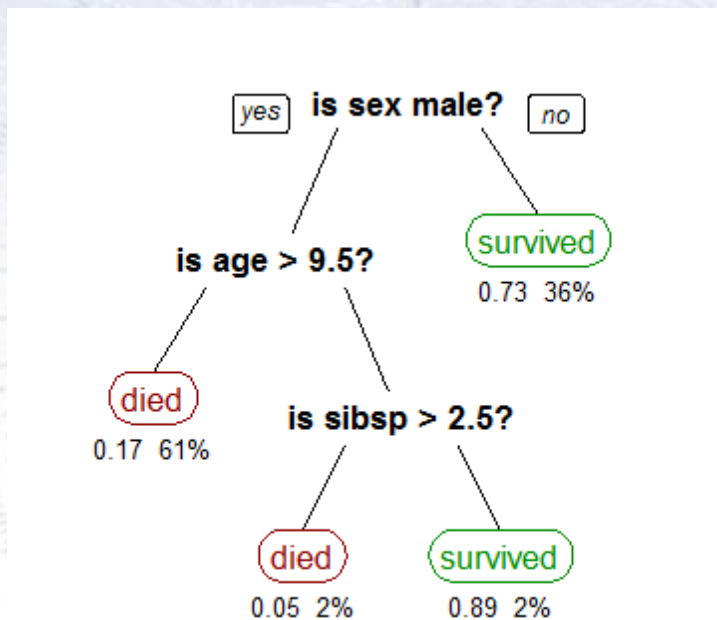
The goal of today's exercise is to:

- Give you an introduction to Machine Learning.
- Be able to recognise problems, where ML is applicable.
- Whet your appetite for using Machine Learning on data.

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



# 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]



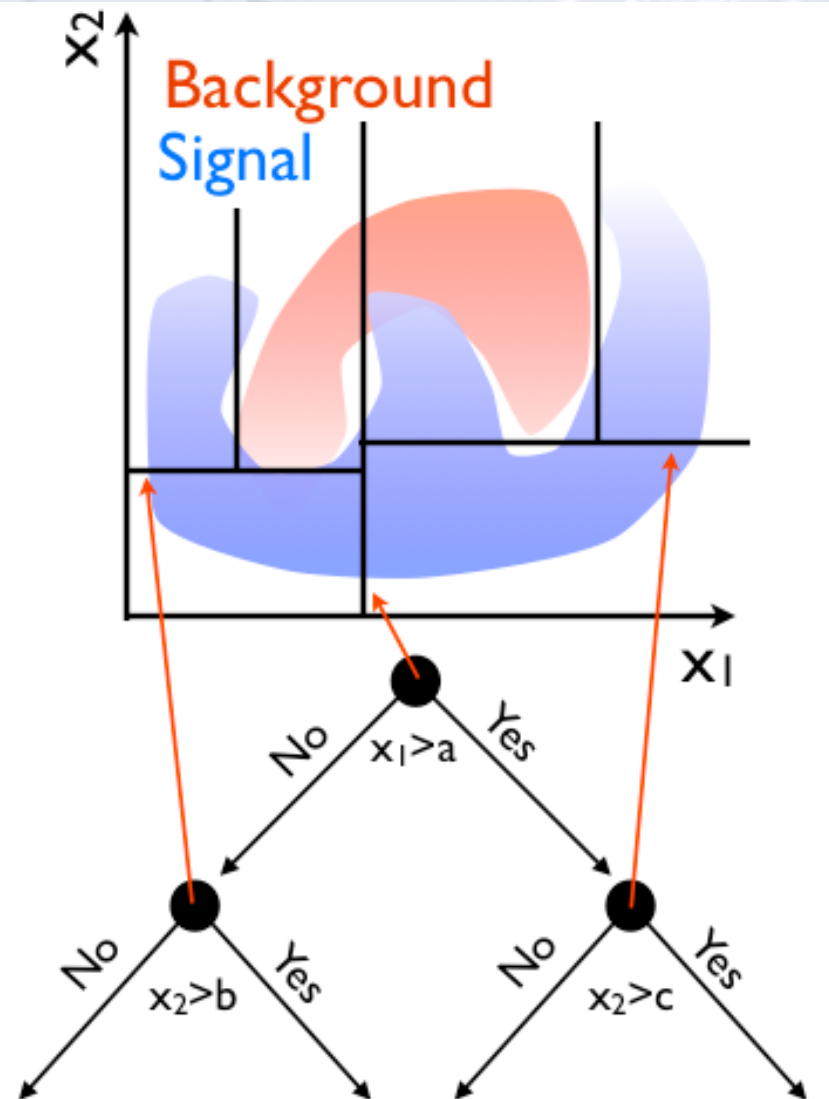
# Boosted Decision Trees

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:

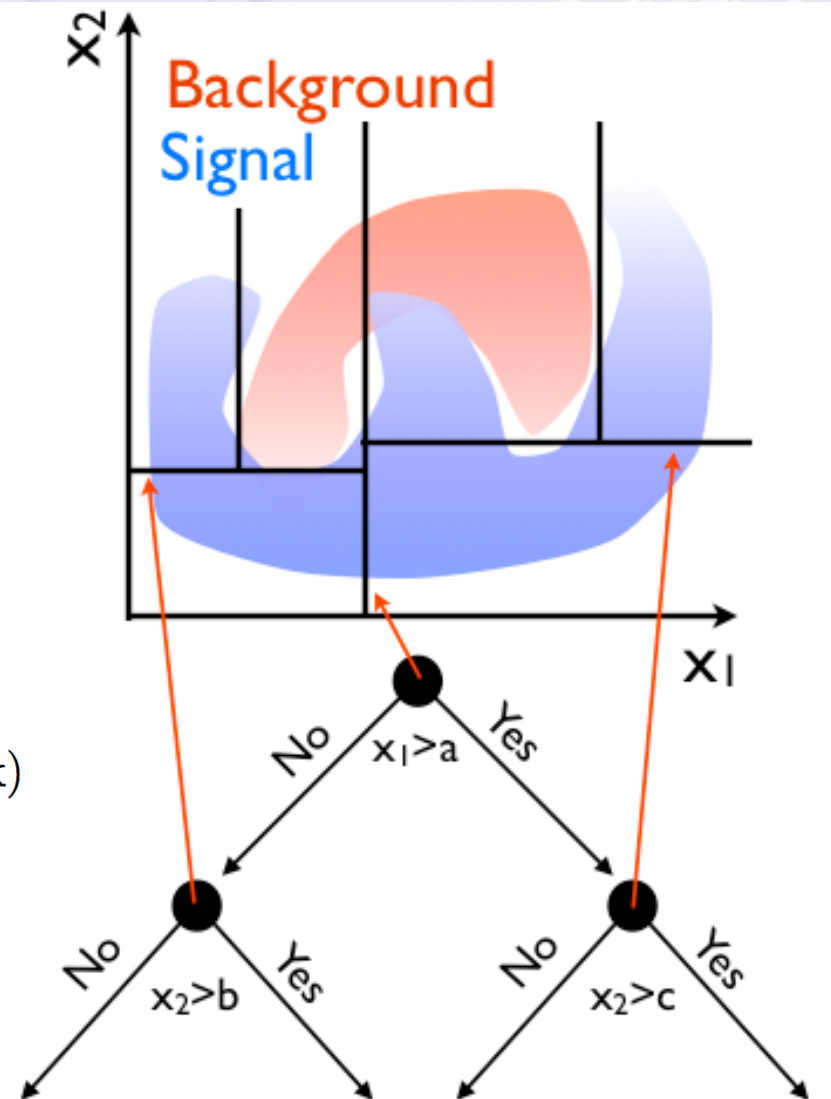
First classifier

Boost weight

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

Parameters in event N

Individual tree



# Boosting...

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

To avoid N identical trees, we assign a higher weight to misclassified entries, i.e. boost

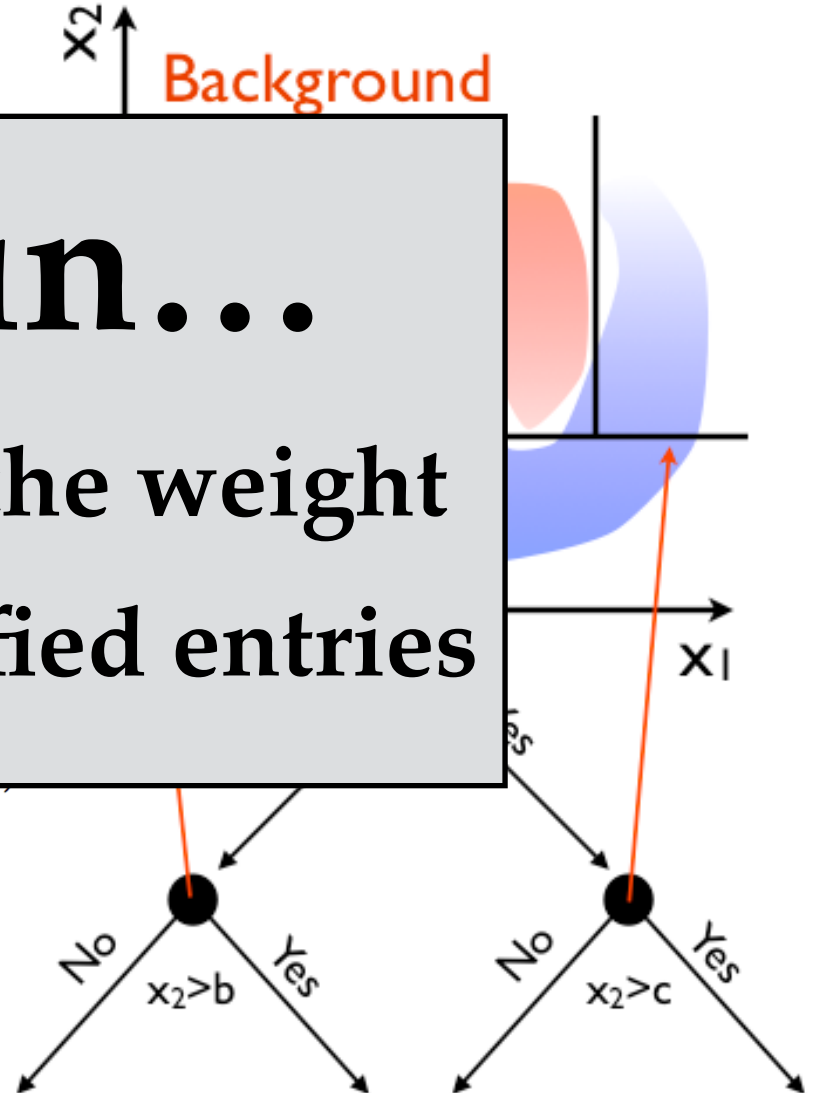
**Rerun...**  
increasing the weight of misclassified entries

First classifier

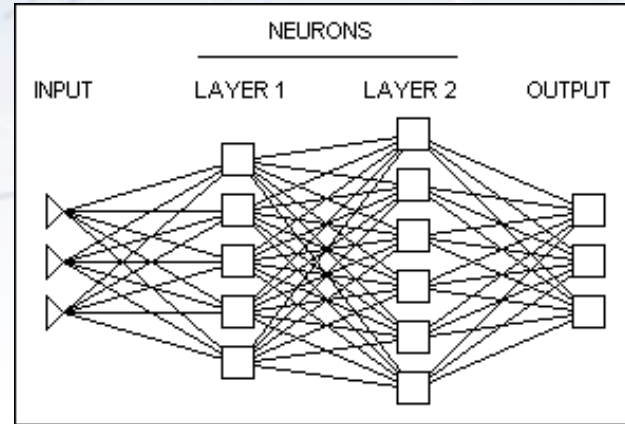
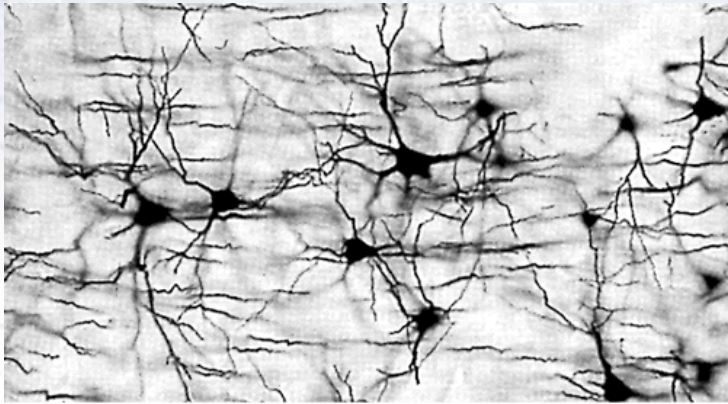
$$y_{\text{Boost}}(\mathbf{x}) = \frac{1}{N_{\text{collection}}} \sum_i$$

Parameters in event N

Individual tree



# 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.

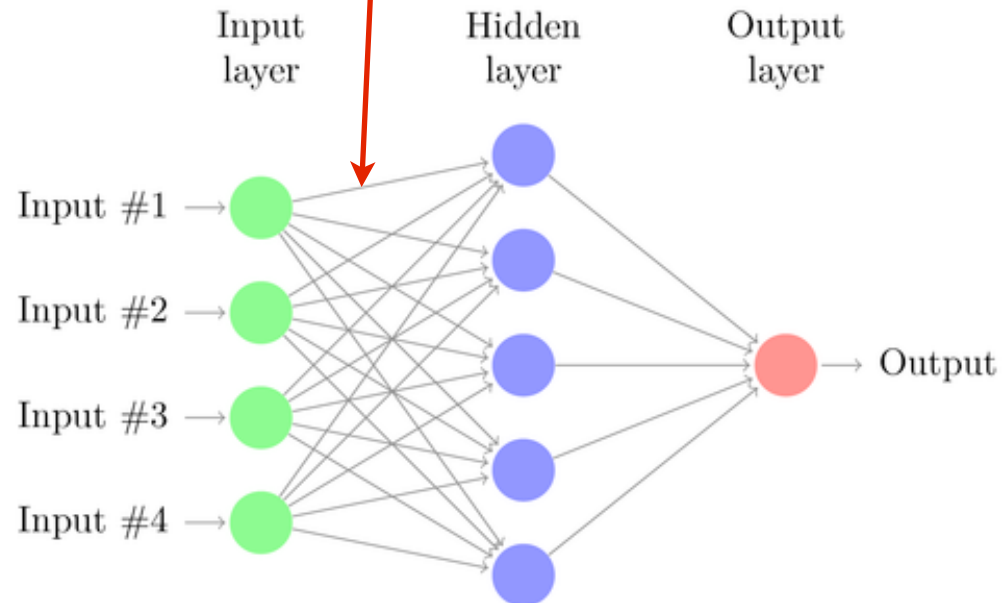
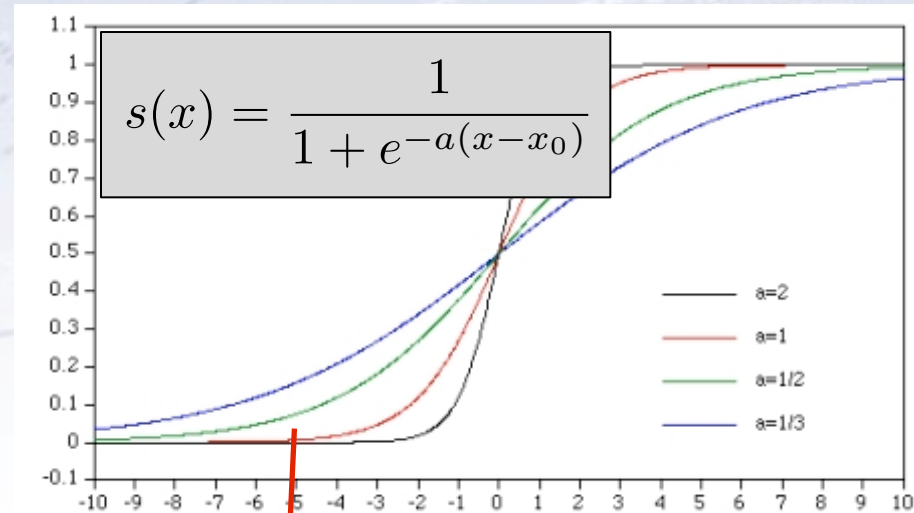
The simplest is a single layer perceptron:

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

This can be generalized to a multilayer perceptron:

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

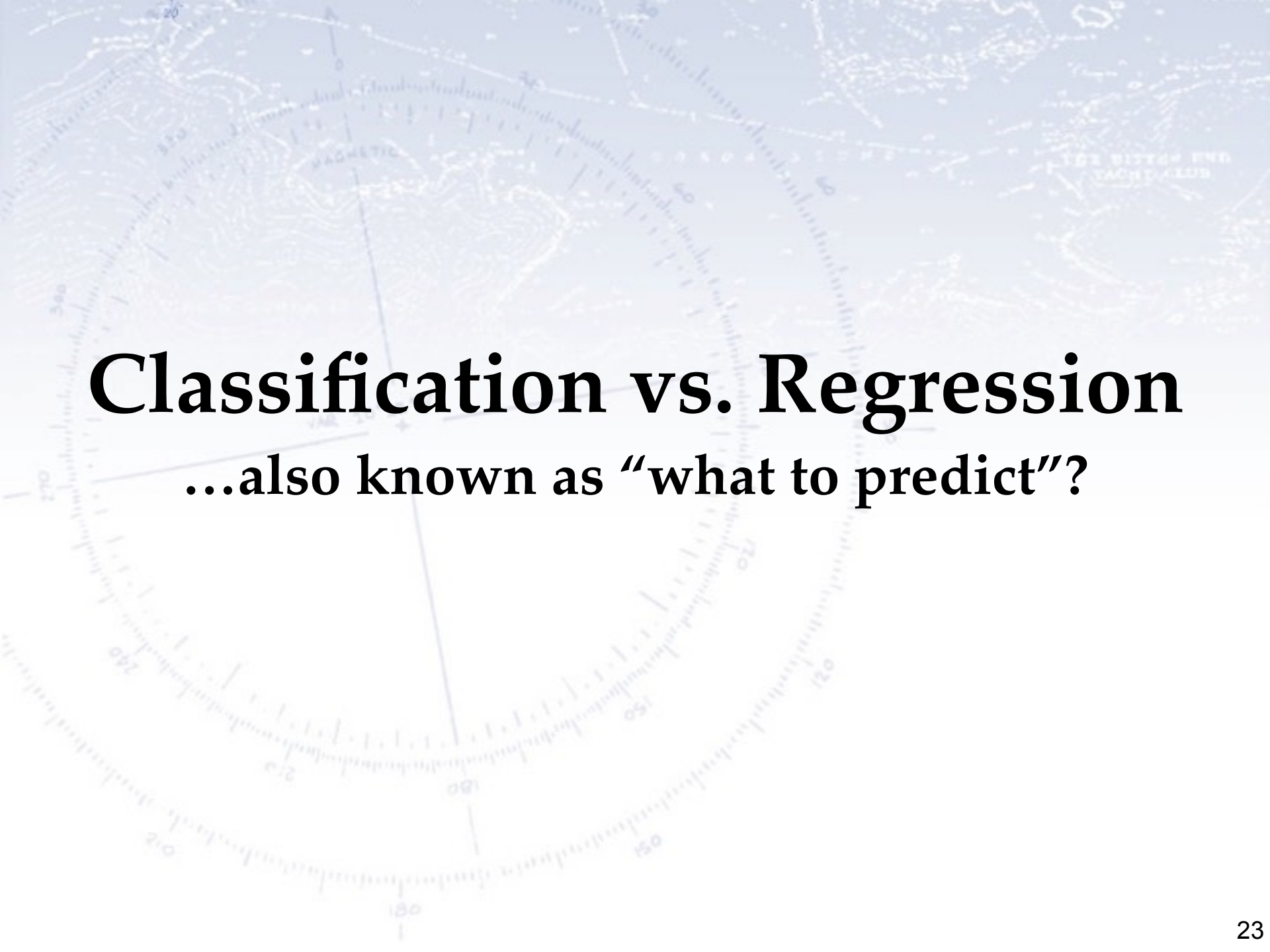
Activation function can be any sigmoid function.



# Method's (dis-)advantages

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

**Table 1:** Assessment of classifier properties. The symbols stand for the attributes “good” (\*\*), “fair” (\*) and “bad” (o). “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.



# Classification vs. Regression

...also known as “what to predict”?

# Classification vs. Regression

The output/predictions of ML algorithms can typically be divided into two:

- Classification: Which **class/category** does the case belong to?
- Regression: What **value** does the case have?

It is possible for ML algorithms to output several values (especially NNs are good at this), but typically not a mixture of classification and regression.

## Classification examples:

- Is the patient healthy or ill?
- Is this case signal or background?
- Is this a photo of dust, volcanic ash, or pollen? (Multi-classification)

## Regression examples:

- At what price will this house sell?
- What is the energy of this particle?
- What is the direction of this particle? (Multi-regression)

This impacts “what you optimise” for your ML algorithm, i.e. the Loss Function.





# Loss functions

...also known as “what to optimise”?

# What loss function to use?

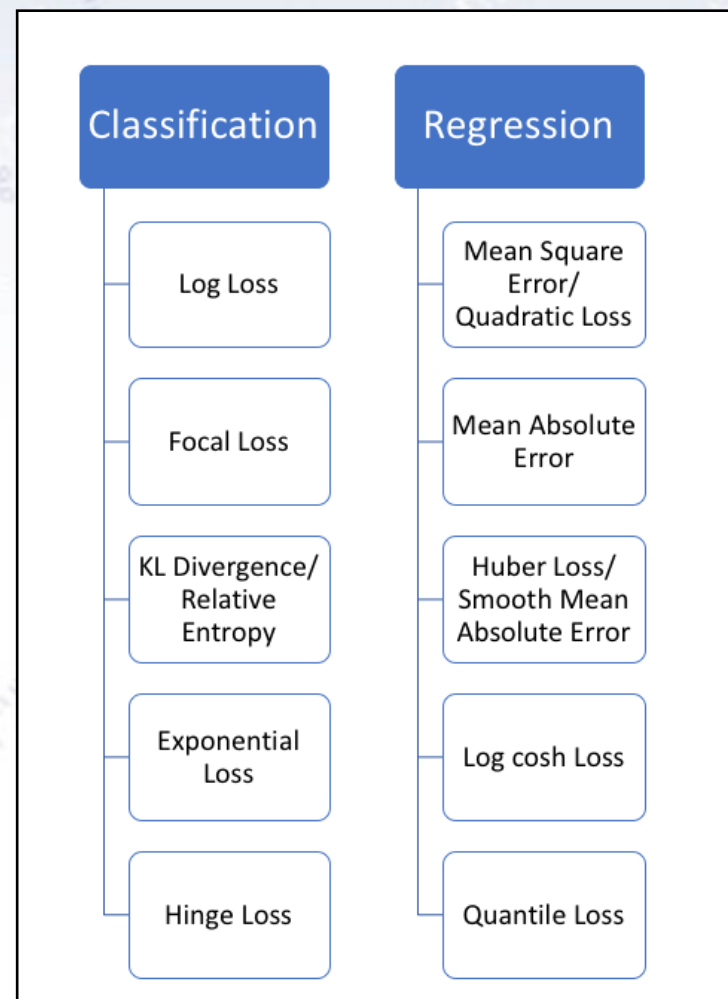
The choice of loss function depends on the problem at hand, and in particular what you find important!

## In classification:

- Do you care how wrong the wrong are?
- Do you want pure signal or high efficiency?
- Does it matter what type of errors you make?

## In regression:

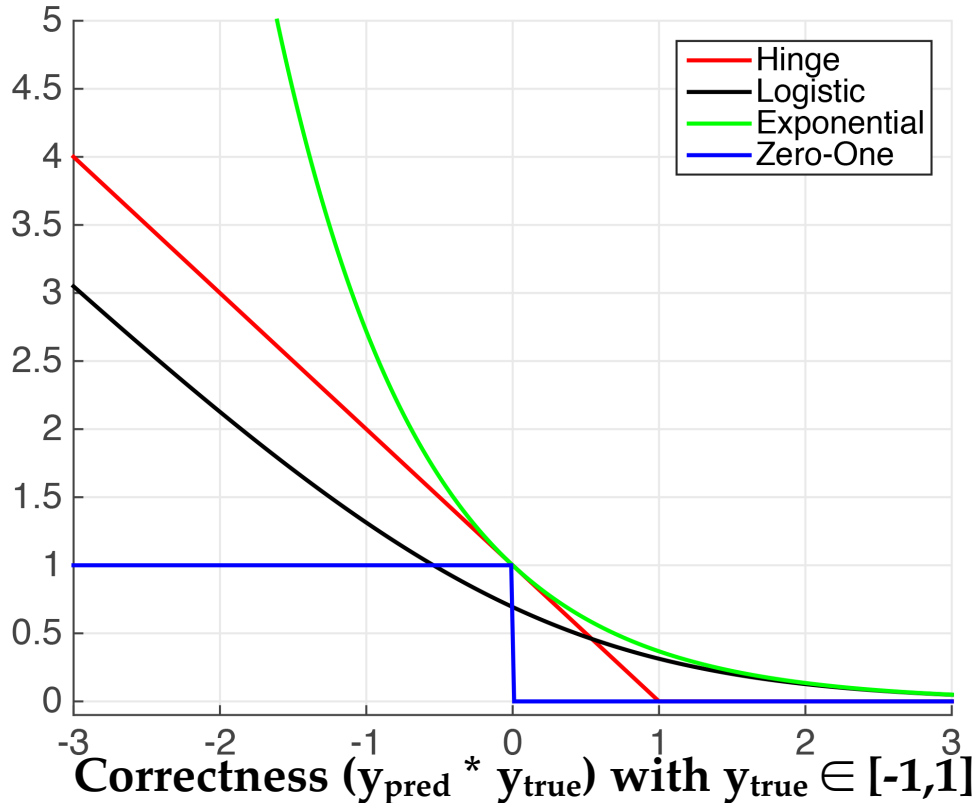
- Do you care about outliers?
- Do you care about size of outliers?
- Is core resolution vital?



# What loss function to use?

The choice of loss function depends on the problem at hand, and in particular what you find important!

## Loss functions for classification



## Classification

Log Loss

Focal Loss

KL Divergence/  
Relative  
Entropy

Exponential  
Loss

Hinge Loss

## Regression

Mean Square  
Error/  
Quadratic Loss

Mean Absolute  
Error

Huber Loss/  
Smooth Mean  
Absolute Error

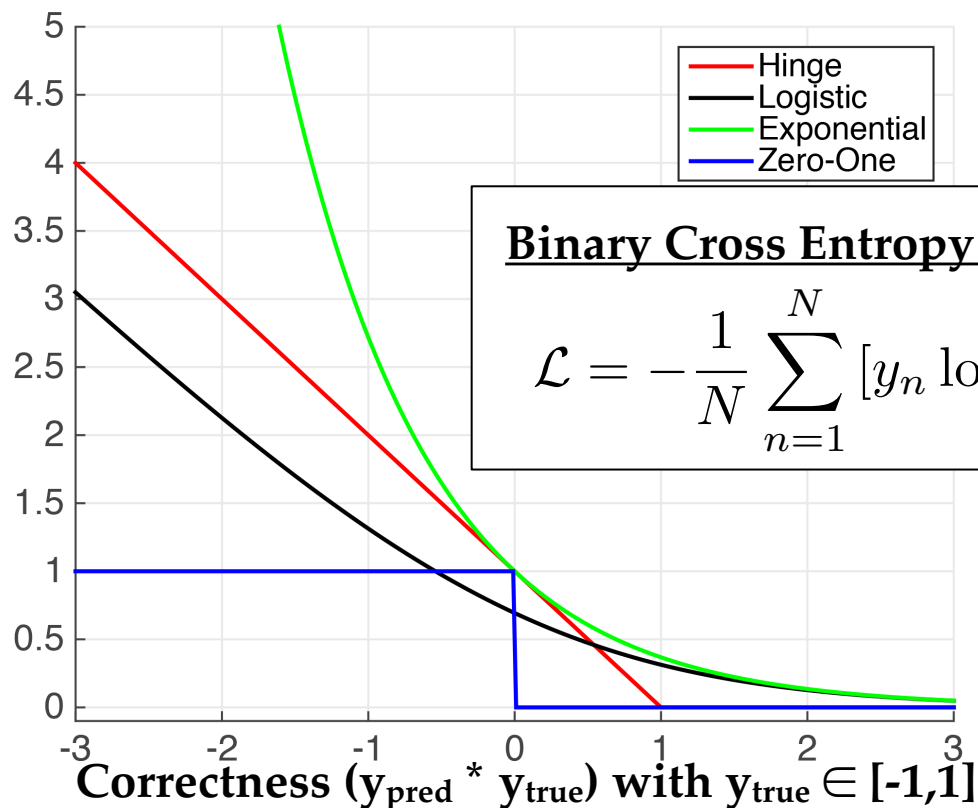
Log cosh Loss

Quantile Loss

# What loss function to use?

The choice of loss function depends on the problem at hand, and in particular what you find important!

Loss functions for classification



**Binary Cross Entropy (aka. LogLoss or Logistic Loss):**

$$\mathcal{L} = -\frac{1}{N} \sum_{n=1}^N [y_n \log \hat{y}_n + (1 - y_n) \log(1 - \hat{y}_n)]$$

Classification

Log Loss

Regression

Mean Square  
Error/  
Quadratic Loss

Exponential  
Loss

Log cosh Loss

Hinge Loss

Quantile Loss

# What loss function to use?

The choice of loss function depends on the problem at hand, and in particular what you find important!

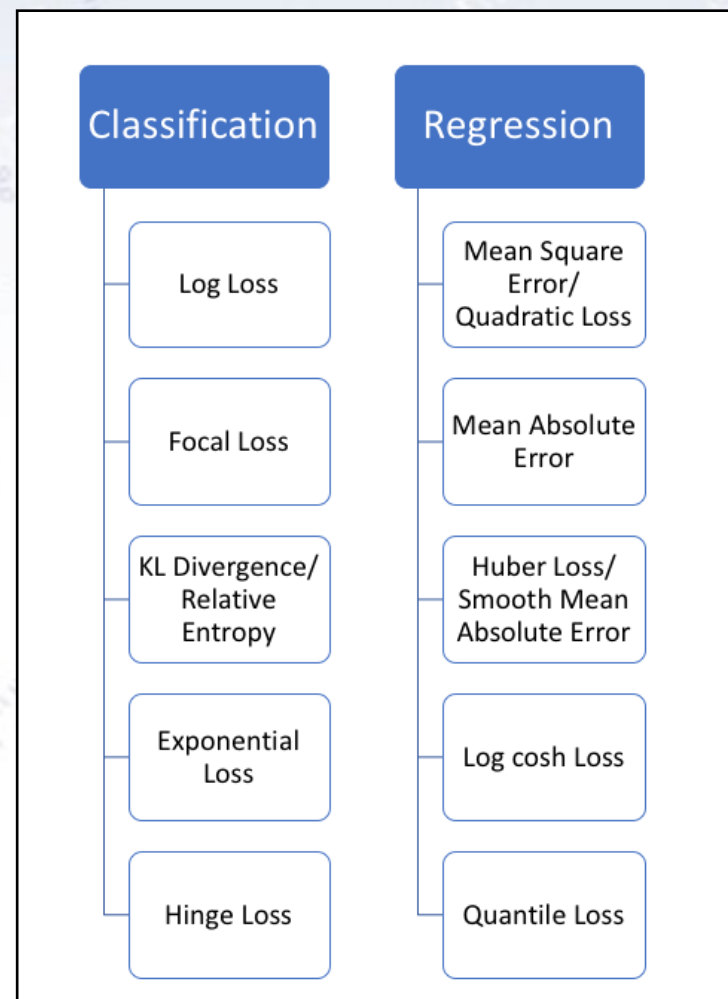
## In classification:

- Do you care how wrong the wrong are?
- Do you want pure signal or high efficiency?
- Does it matter what type of errors you make?

## In regression:

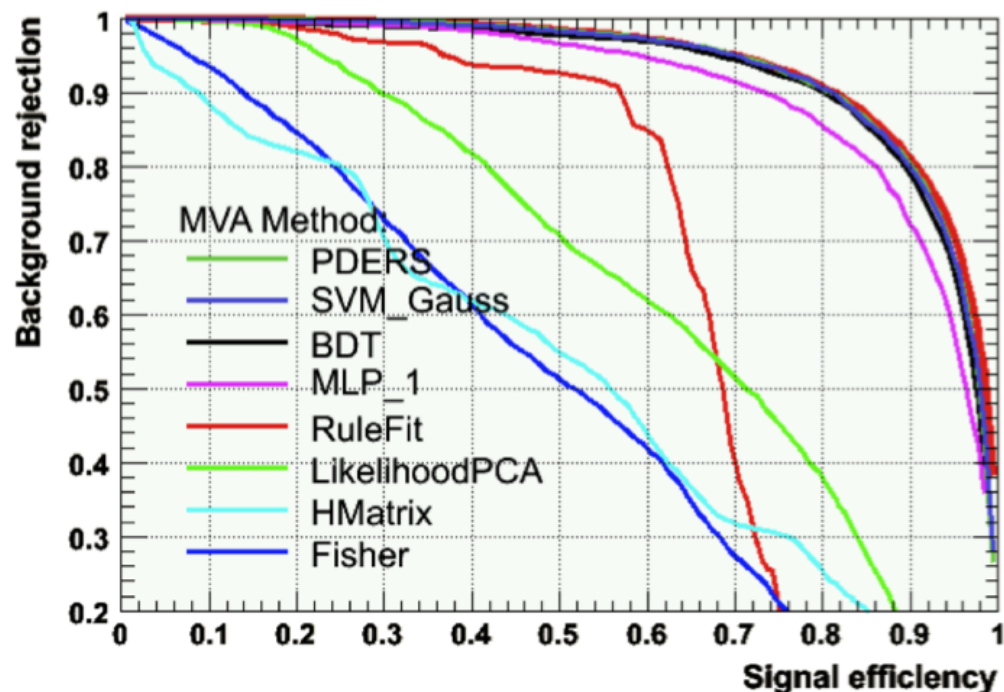
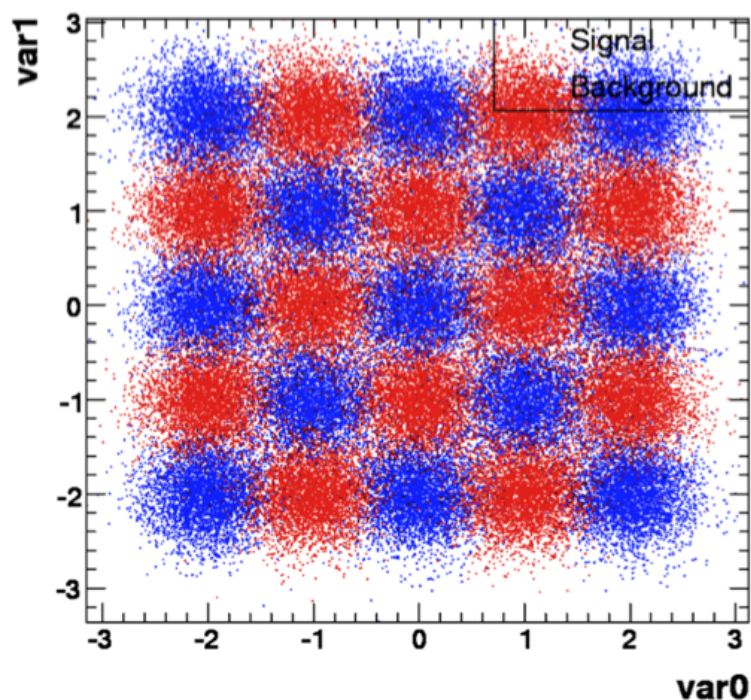
- Do you care about outliers?
- Do you care about size of outliers?
- Is core resolution vital?

Ultimately, the loss function should be tailored to match the wishes of the user. This is however not always that simple, as this might be hard to even know!



# 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...

# 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)!

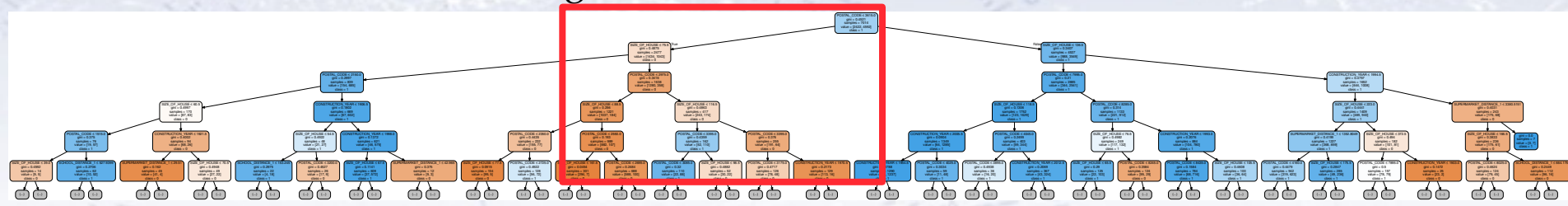
[Trevor Hastie, Prof. of Mathematics & Statistics, Stanford]



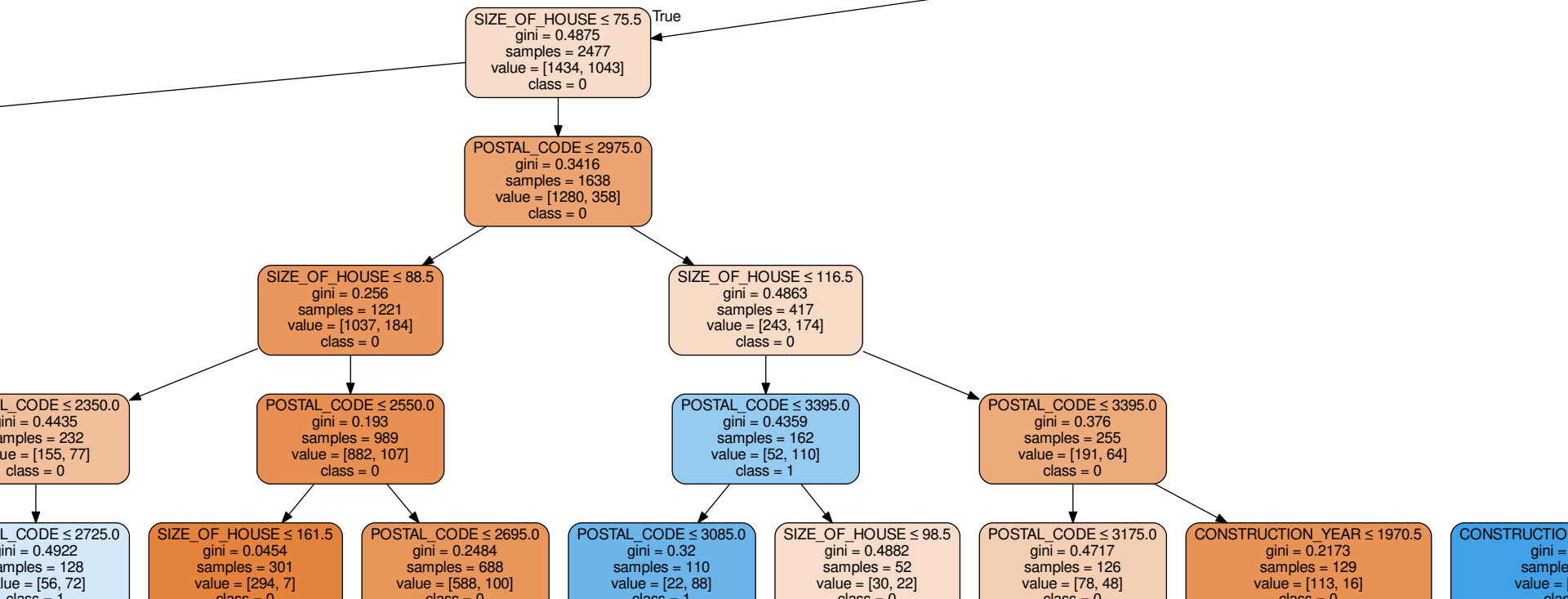


# Housing Prices decision tree

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



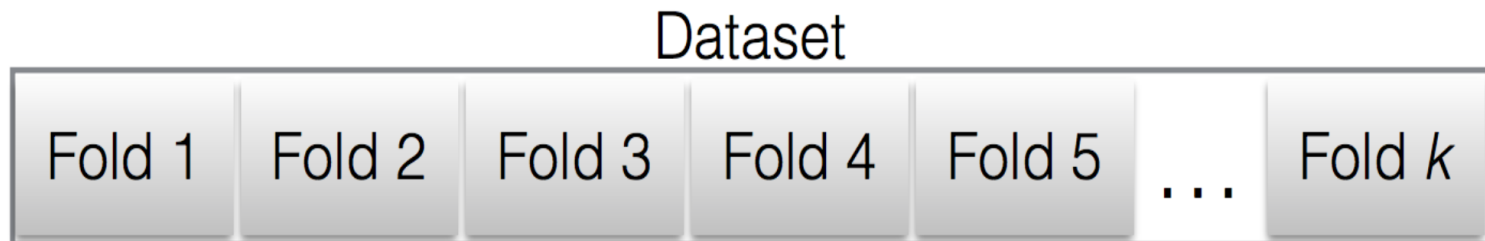
POSTAL\_CODE ≤ 3615.0  
gini = 0.4521  
samples = 7014  
value = [2422, 4592]  
class = 1



# 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.

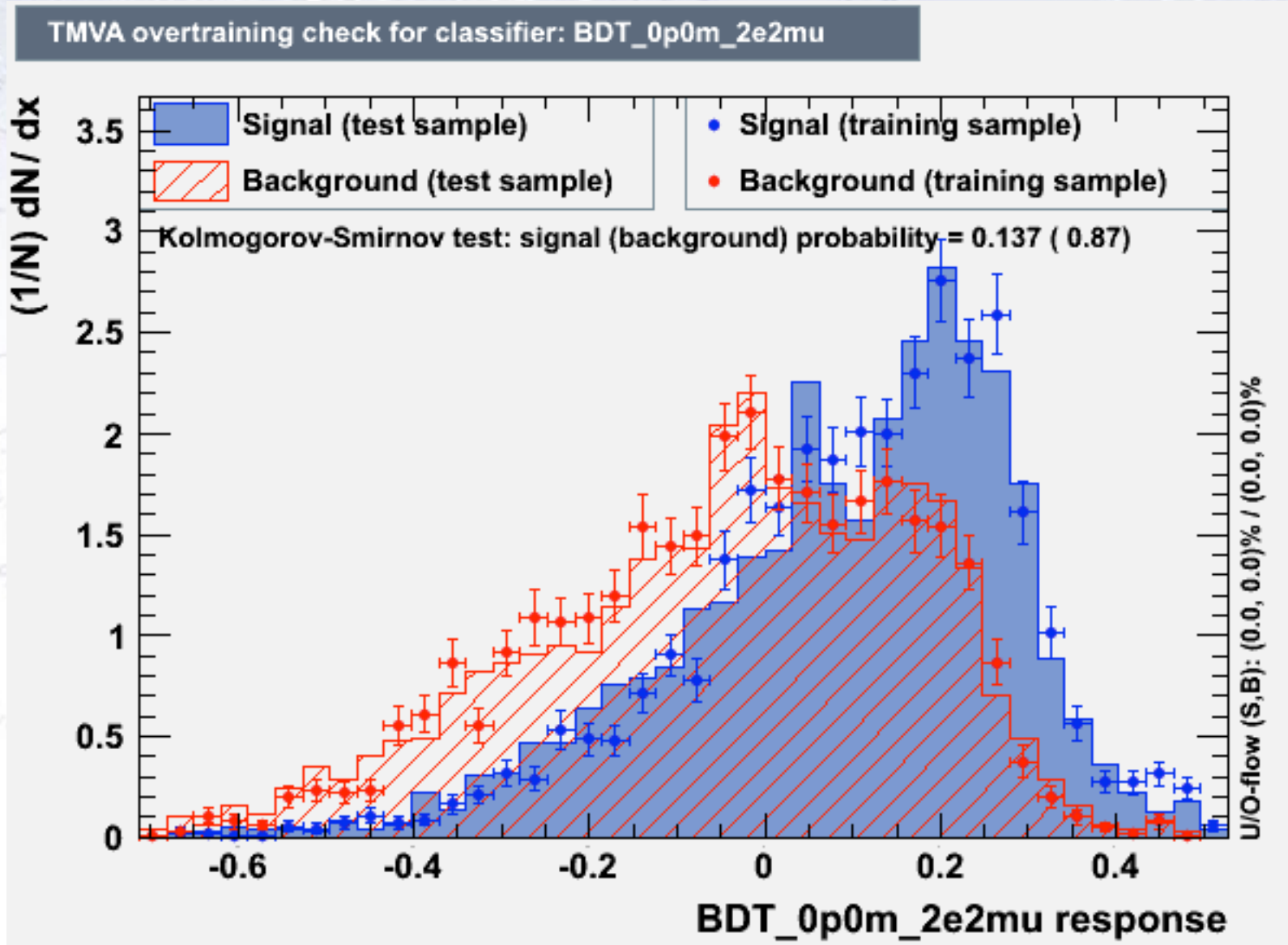
This is then repeated for each fold... CPU-intensive, but smart for small data samples.



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

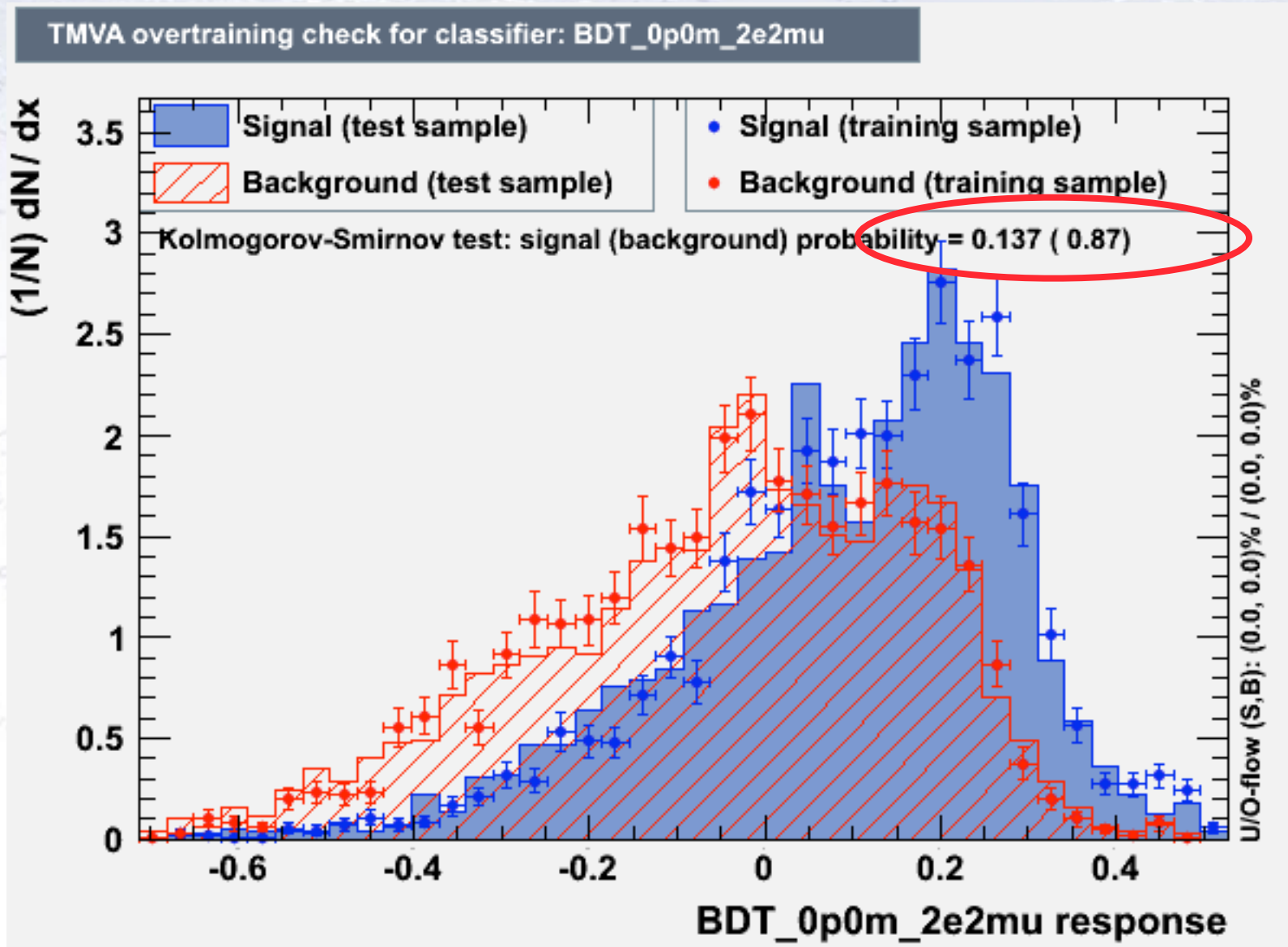
# Test for overtraining

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



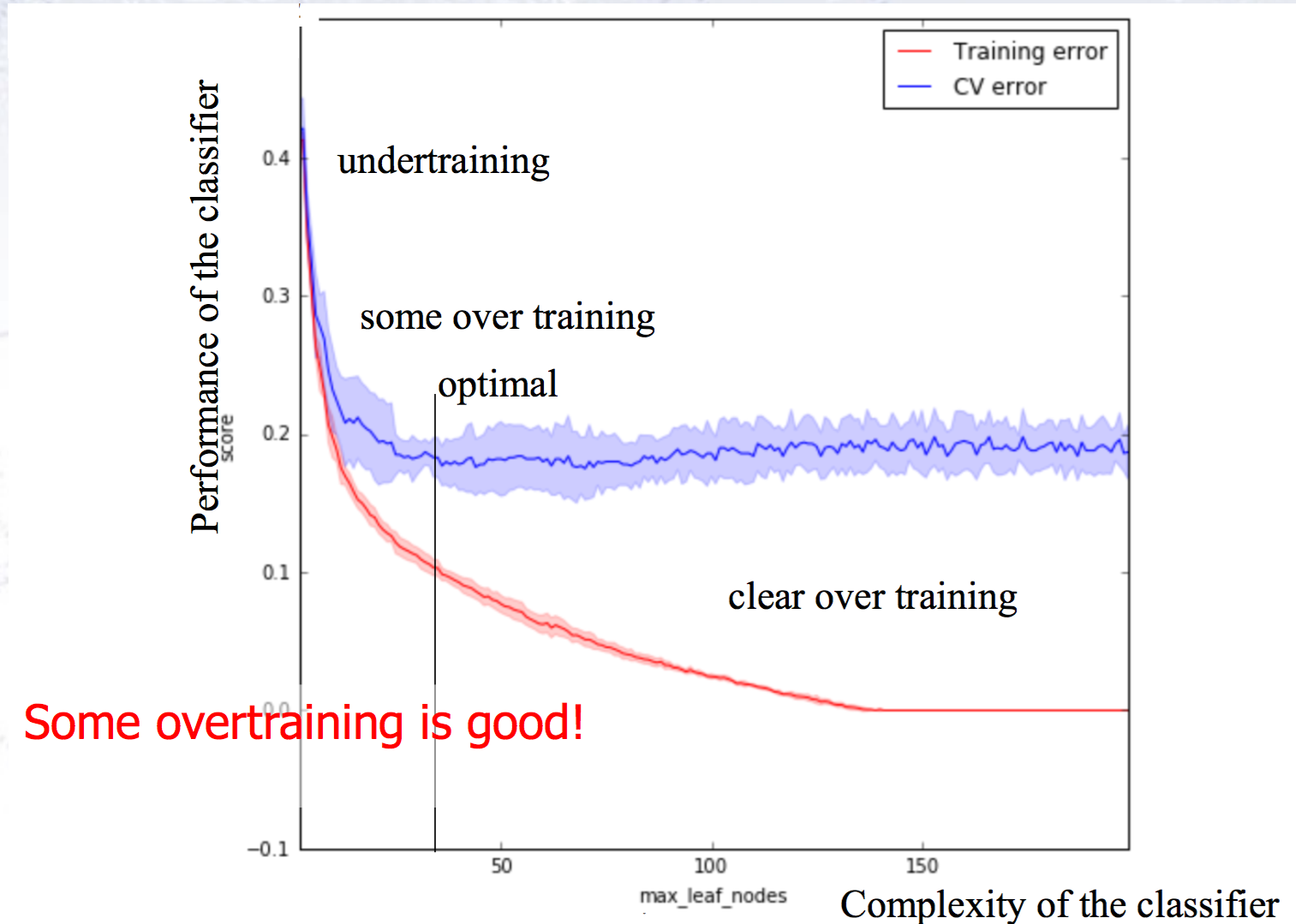
# Test for overtraining

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!



Some overtraining is good!



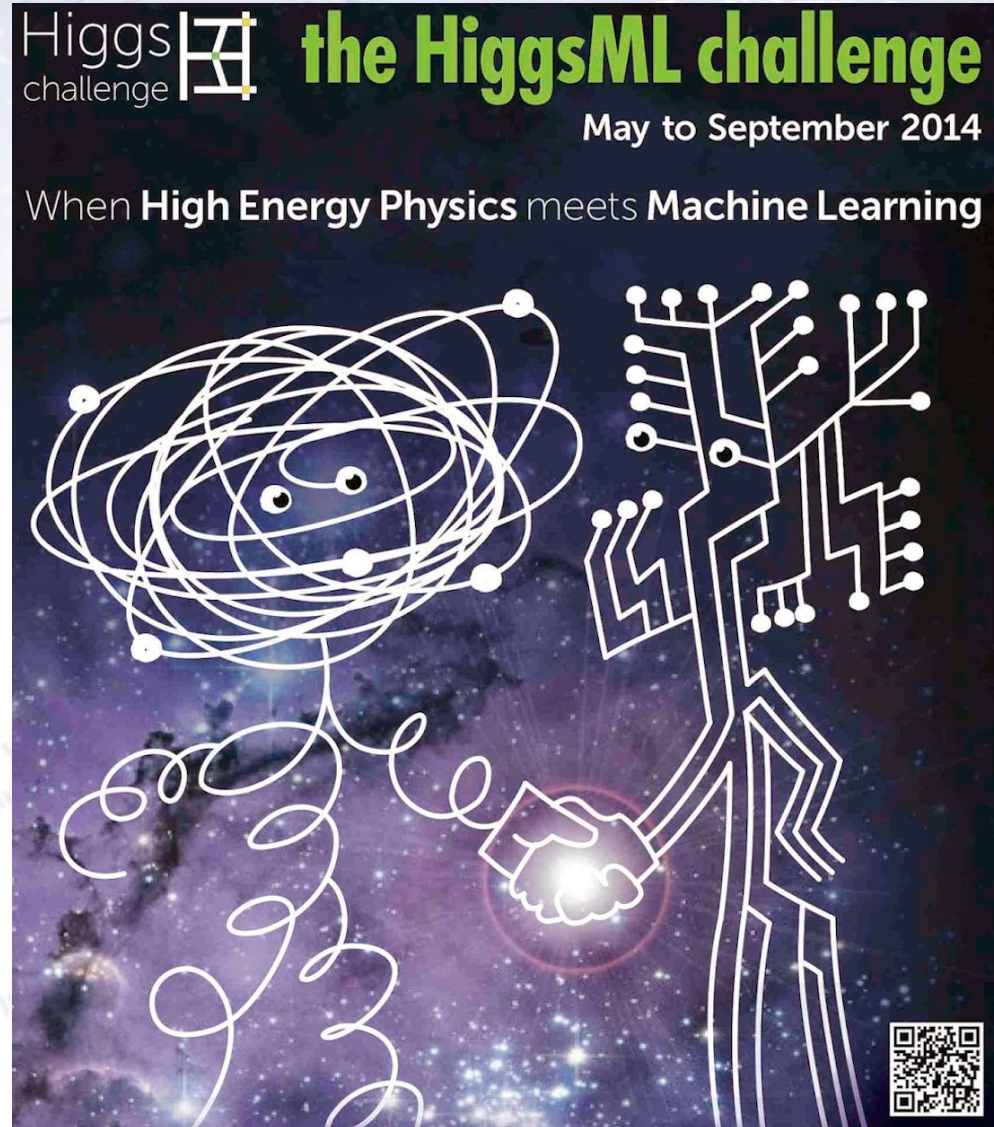
# XGboost - a neat little story!

# 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)



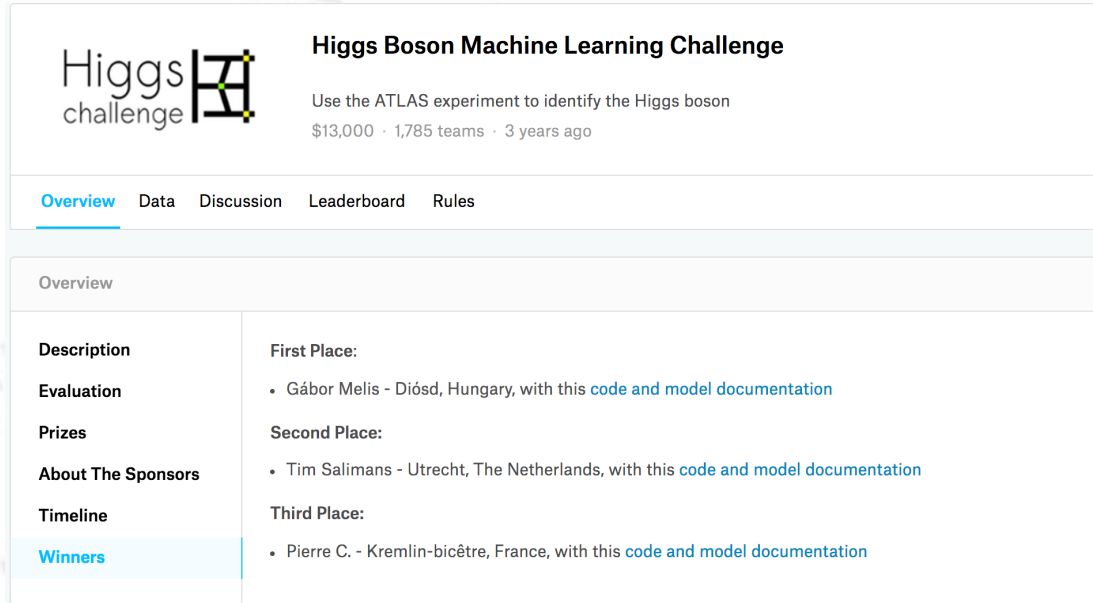
# XGBoost history

## History [\[ edit \]](#)

XGBoost initially started as a research project by Tianqi Chen<sup>[8]</sup> 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.<sup>[7]</sup>

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 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 <a href="#">code and model documentation</a>
Prizes	Second Place:
About The Sponsors	• Tim Salimans - Utrecht, The Netherlands, with this <a href="#">code and model documentation</a>
Timeline	Third Place:
Winners	• Pierre C. - Kremlin-bicêtre, France, with this <a href="#">code and model documentation</a>



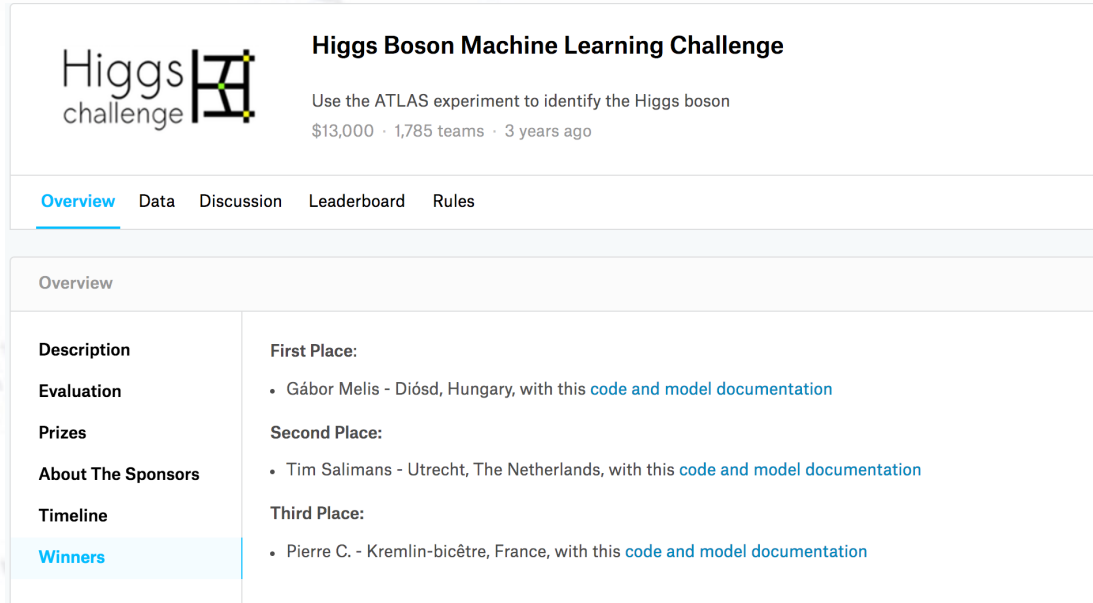
# XGBoost history

## History [\[ edit \]](#)

XGBoost initially started as a research project by Tianqi Chen<sup>[8]</sup> 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.<sup>[7]</sup>

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 a table of winners.

Category	Winner
First Place:	Gábor Melis - Diósd, Hungary, with this <a href="#">code and model documentation</a>
Second Place:	Tim Salimans - Utrecht, The Netherlands, with this <a href="#">code and model documentation</a>
Third Place:	Pierre C. - Kremlin-bicêtre, France, with this <a href="#">code and model documentation</a>

# 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

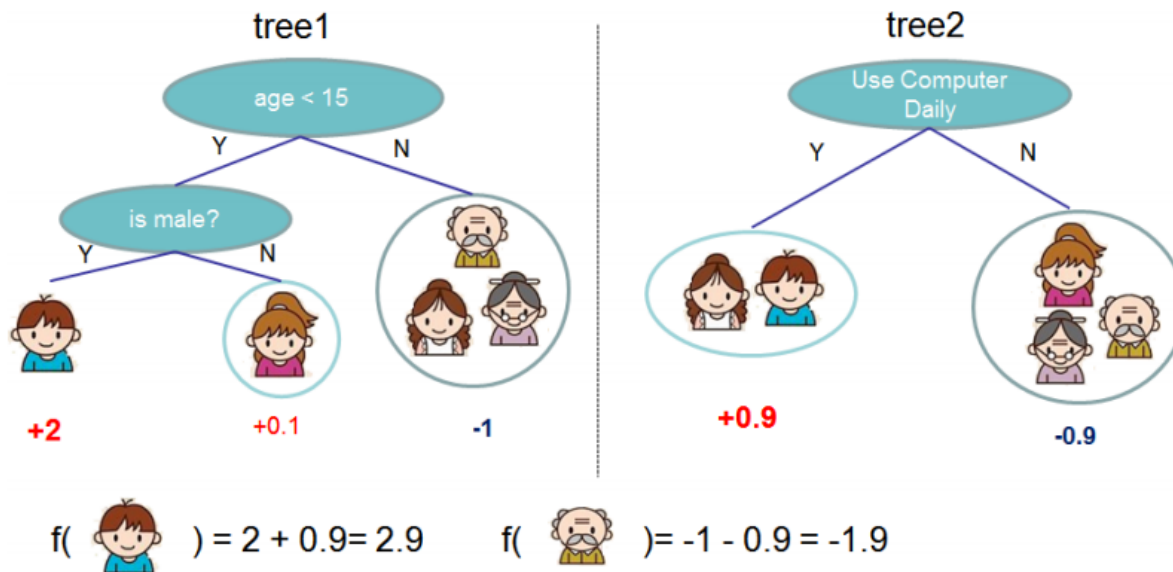
problems. Besides being used as a stand-alone predictor, it is also incorporated into real-world production pipelines for ad click through rate prediction [15]. Finally, it is the de-facto choice of ensemble method and is used in challenges such as the Netflix prize [3].

In this paper, we describe XGBoost, a scalable machine learning system for tree boosting. The system is available as an open source package<sup>2</sup>. 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<sup>3</sup> 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).

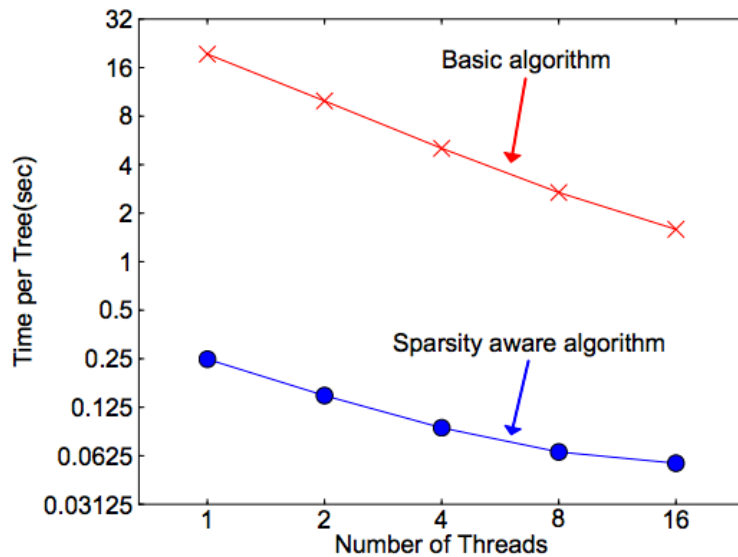
Unlike decision trees, each regression tree contains a continuous score on each leaf:



**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
<b>XGBoost</b>	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?**

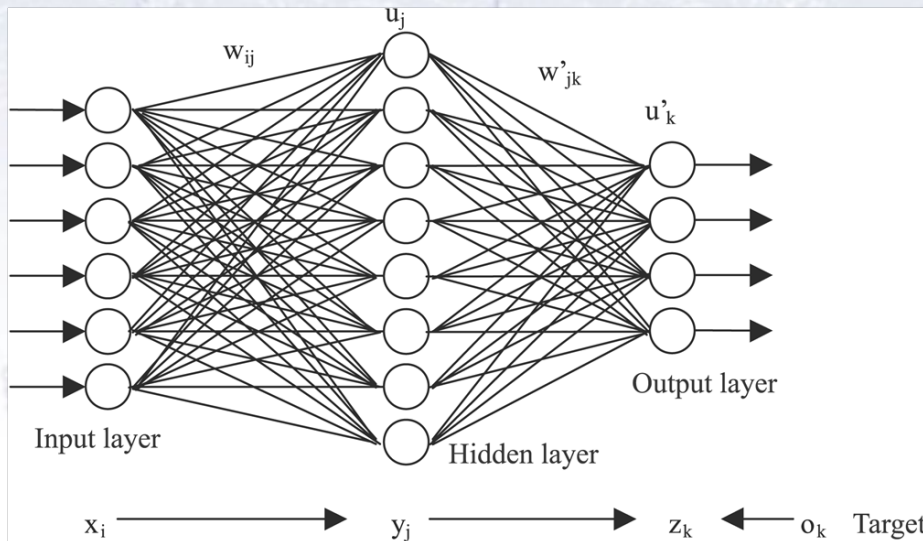


**(Deep) Neural Networks**



# Deep Neural Networks

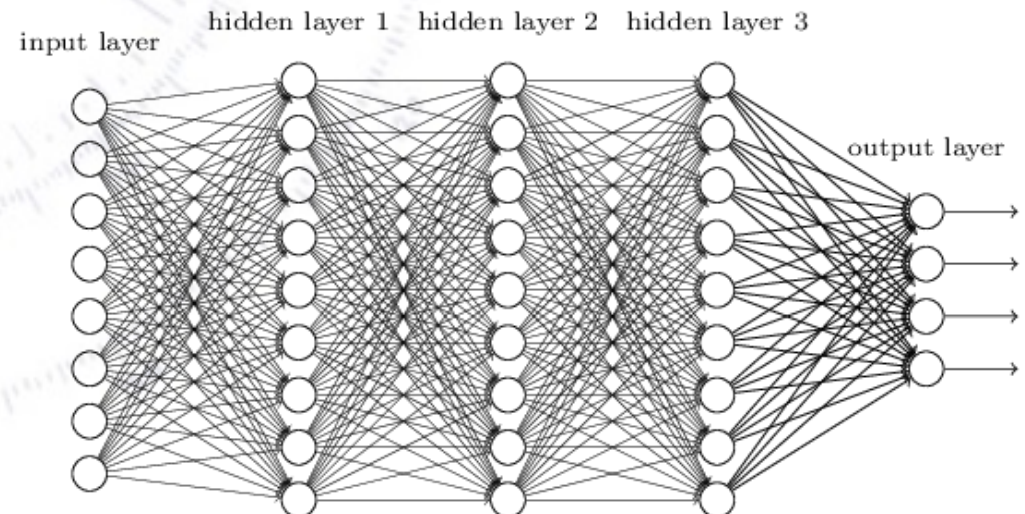
Deep Neural Networks (DNN) are simply (much) extended NNs in terms of layers!



Instead of having just one (or few) hidden layers, many such layers are introduced.

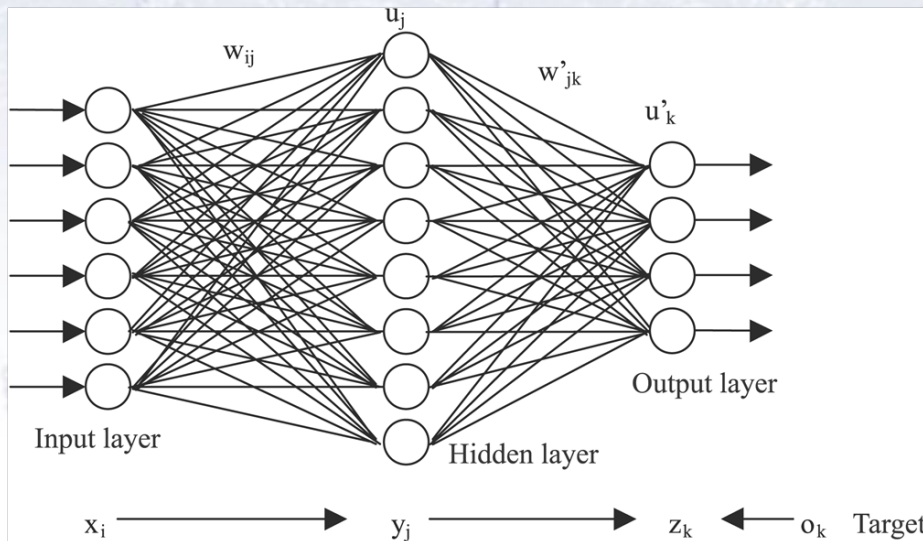
This gives the network a chance to produce key features and use them for many different specialised tasks.

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



# Deep Neural Networks

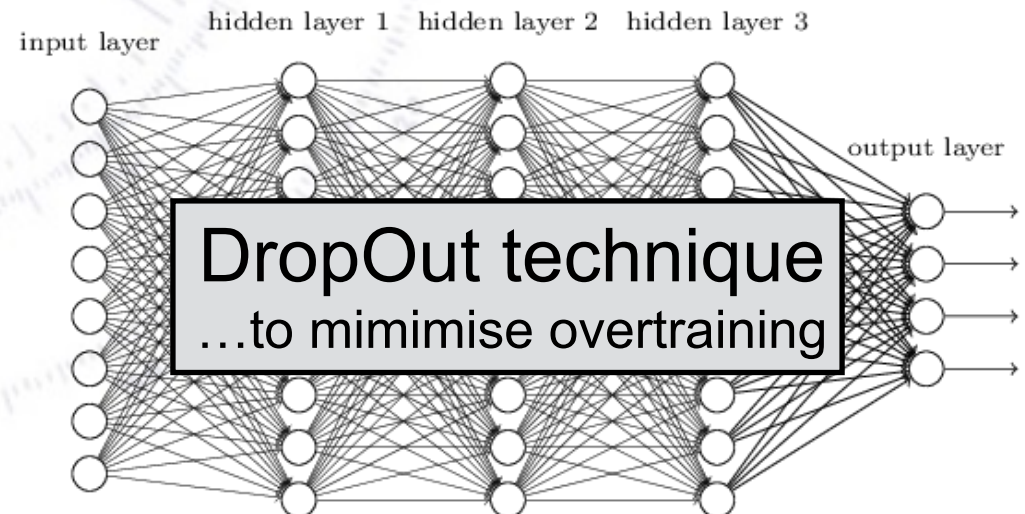
Deep Neural Networks (DNN) are simply (much) extended NNs in terms of layers!



Instead of having just one (or few) hidden layers, many such layers are introduced.

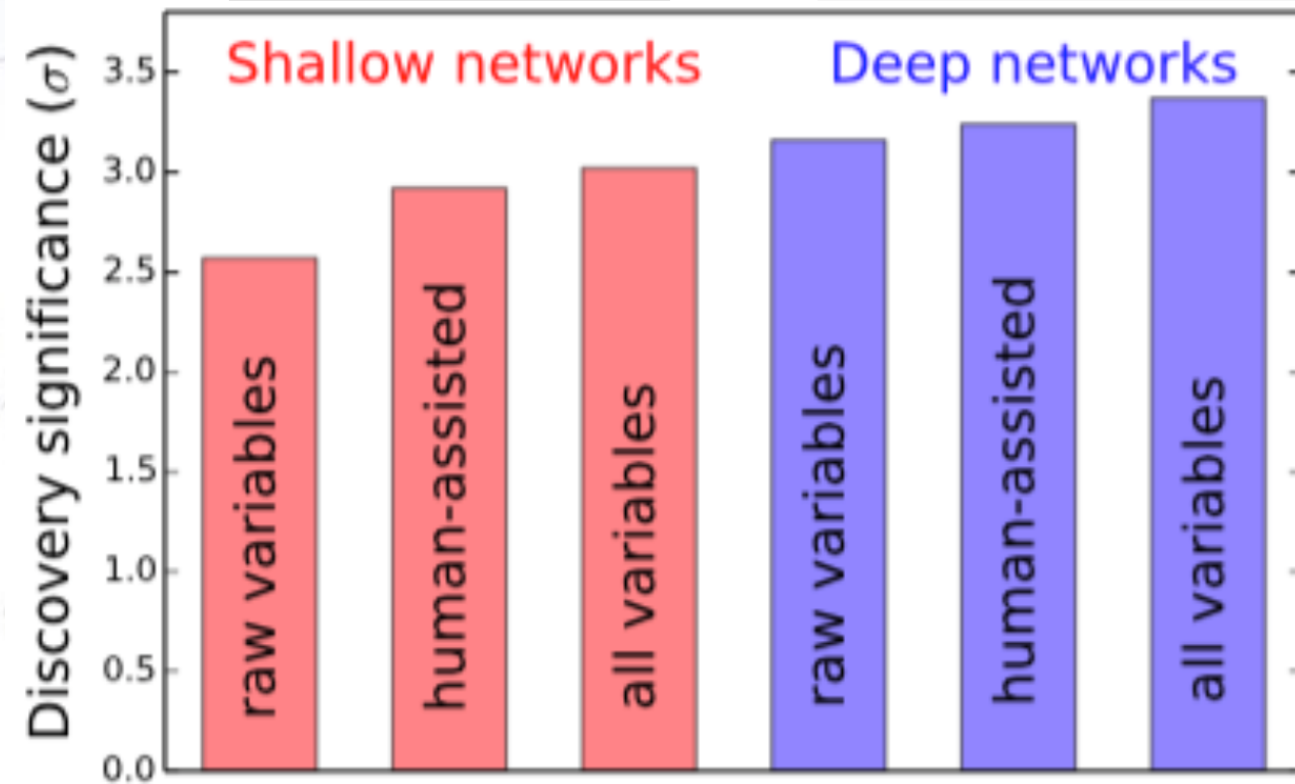
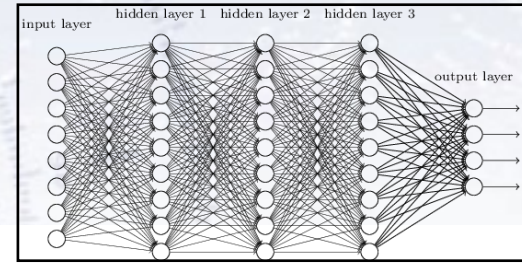
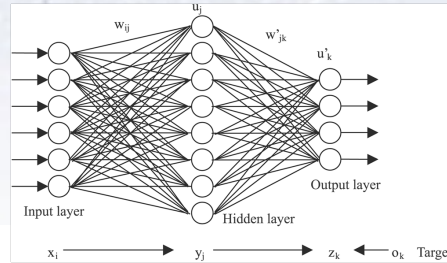
This gives the network a chance to produce key features and use them for many different specialised tasks.

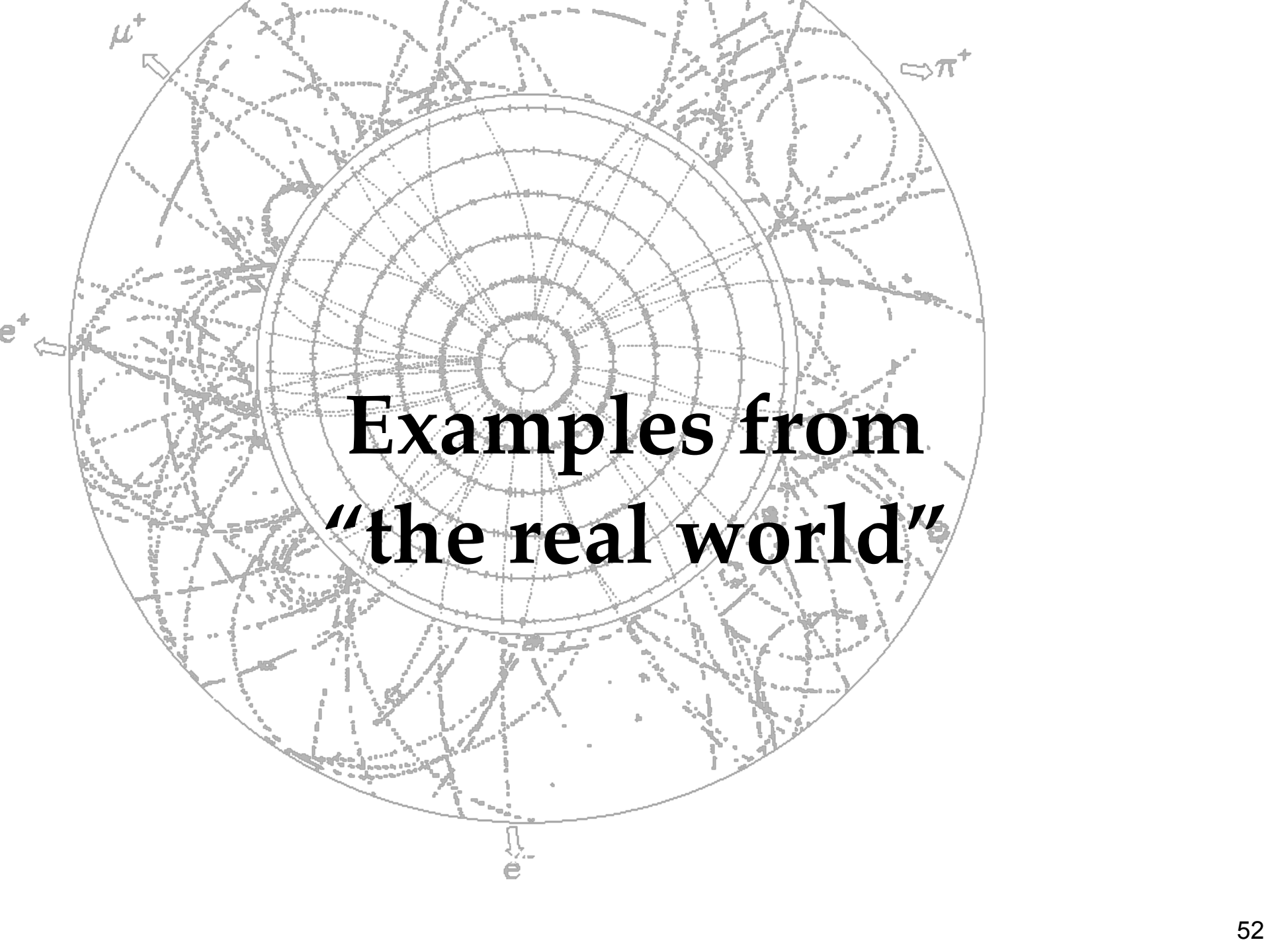
Currently, DNNs can have up to millions of neurons and connections, which compares to about the **brain of a worm**.



# Deep Neural Networks

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

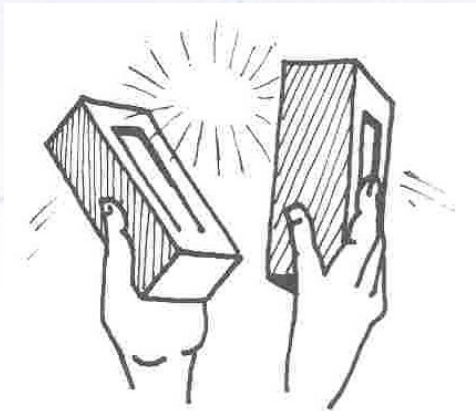




**Examples from  
“the real world”**

# 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?!?**

# 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

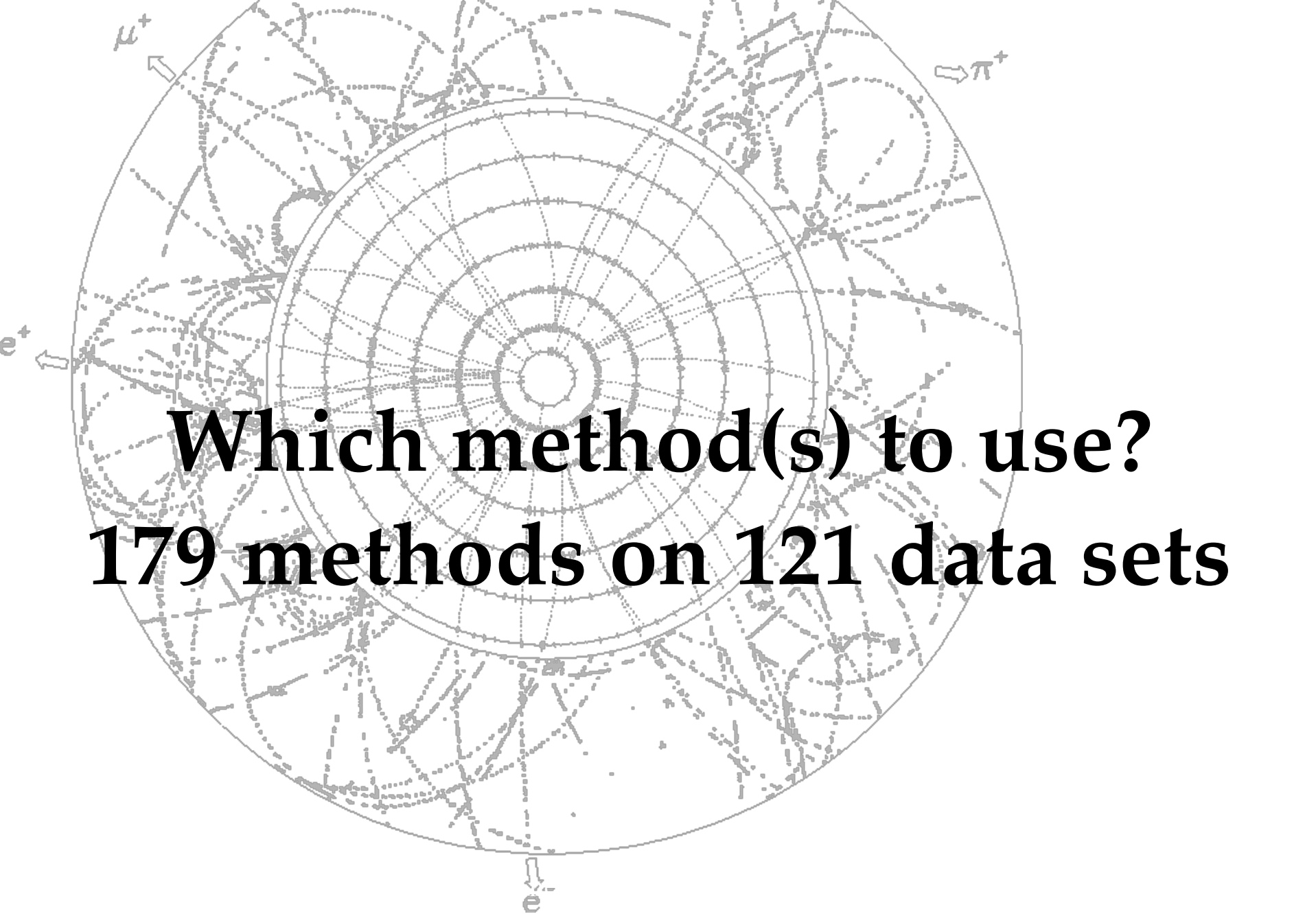
[Heekyung Hellen Kim](#)

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

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

*Dep*  
*Uni*  
*Av.* In a quiet interesting paper, four authors investigated the performance of many ML methods (179 in total) on a large variety of data sets (121 in total).

*Edi* 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.



# 179 methods vs. 121 data sets

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

# 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

# Summary & Conclusions

**Humans are great for problems of low dimensionality.** Linear methods are great for linear problems.

However, real world problems are often high dimensional and non-linear, i.e. “complicated”. Here, Machine Learning (ML) can provide a solution, if good (i.e. many) known cases are available for training.

Large amounts of data with NO known cases can be considered through “unsupervised” learning, but this is hard and typically less powerful.

**ML typically requires high statistics and is not very transparent,** and thus does not apply to simpler and/or low statistics cases.

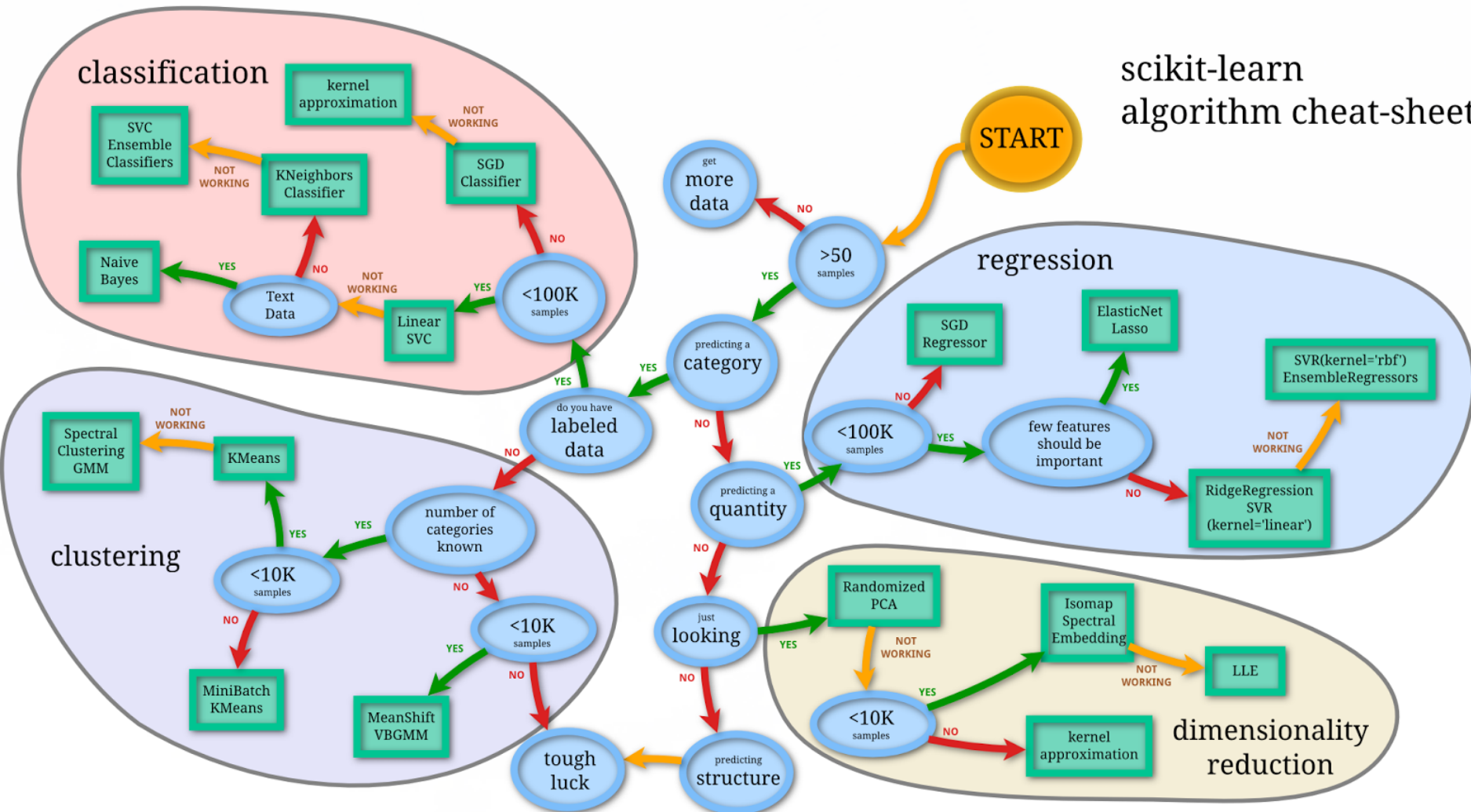
In the end, simple solutions are often great. But if the case is not one such, ML is a great way of “easily extracting” the information and boiling it down to a single/few variable, which summarises the information available.



# Which method to use?

There is no good / simple answer to this, though people have tried, e.g.:

scikit-learn  
algorithm cheat-sheet



# Which method to use?

There is no good / simple answer to this, though people have tried, e.g.:

scikit-learn  
algorithm cheat-sheet

