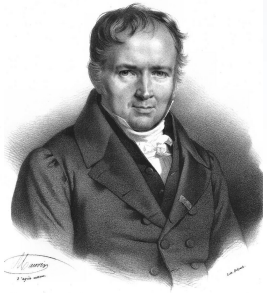
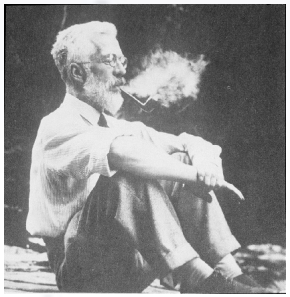
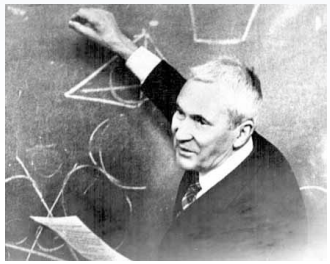
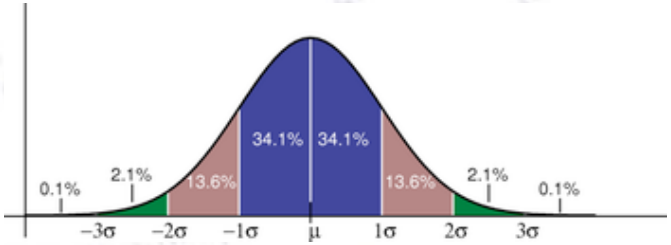
Applied Statistics

Introduction to Machine Learning



Troels C. Petersen (NBI)

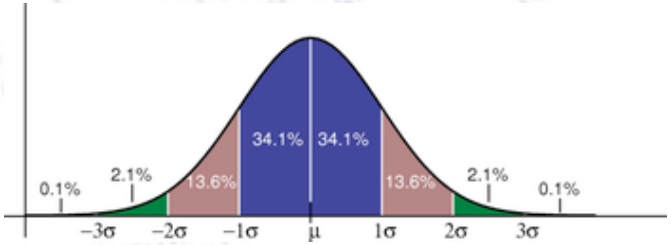


"Statistics is merely a quantisation of common sense. Machine Learning is a sharpening of it!"

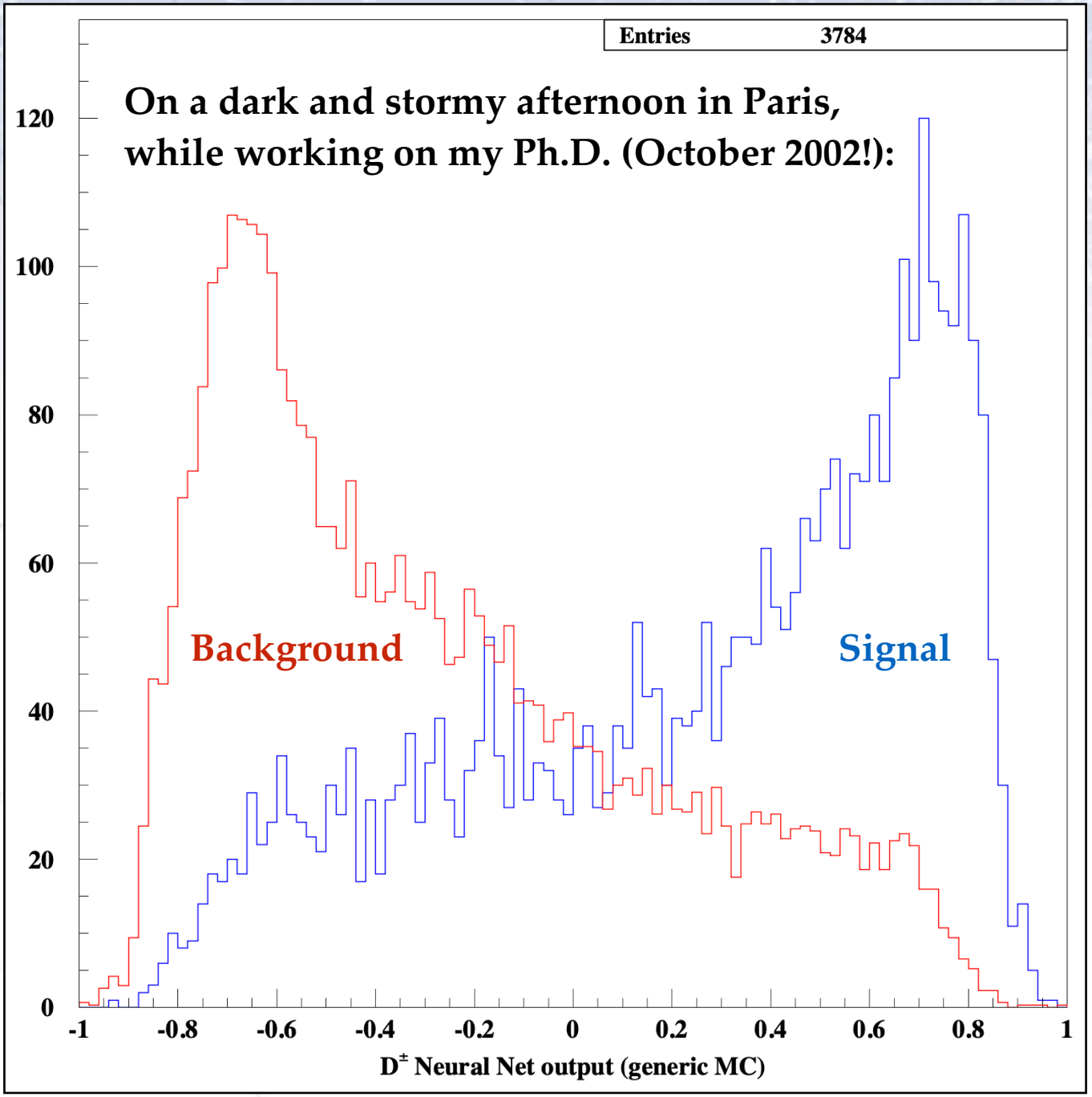
Applied Statistics

Introduction to Machine Learning

Troels C. Petersen (NBI)



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


Entries 3784
On a dark and stormy afternoon in Paris,
while working on my Ph.D. (October 2002!):
Background
Signal
120
100
80
60
40
20
0
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1
D± Neural Net output (generic MC)

Comment on "The AI Hype"

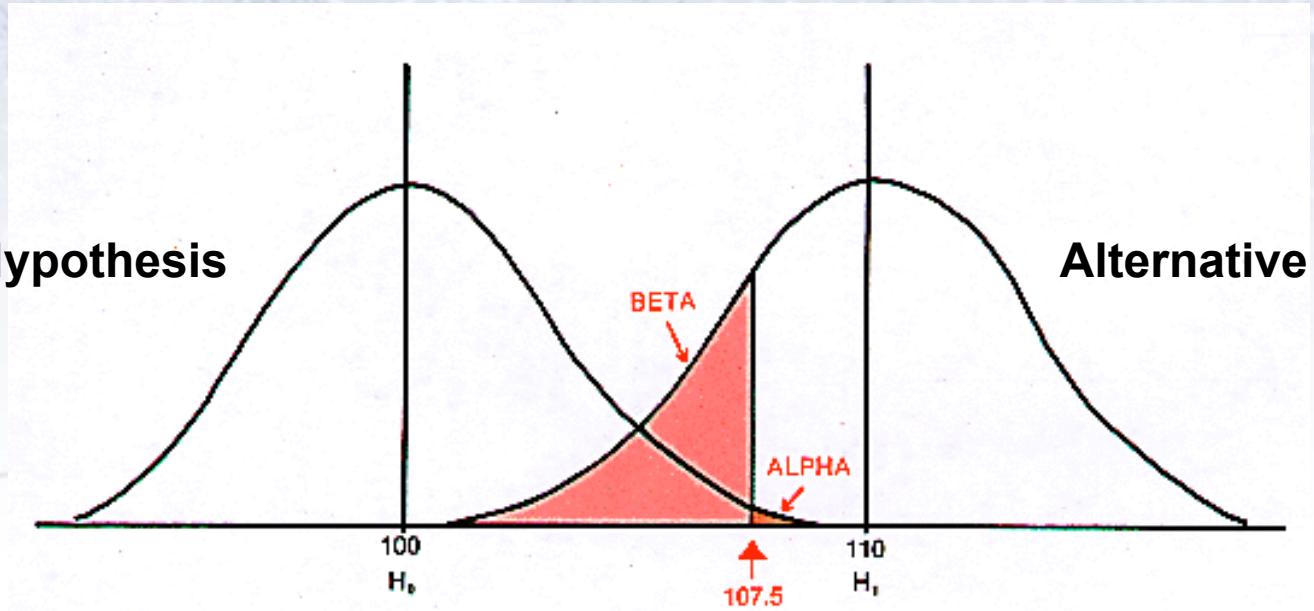
Machine Learning is a tool like all others (logic, math, computers, statistics, etc.)

> *Despite the connotations of machine learning and artificial intelligence as a mysterious and radical departure from traditional approaches, we stress that machine learning has a mathematical formulation that is closely tied to statistics, the calculus of variations, approximation theory, and optimal control theory.*
>
> [PDG 2024, Review of Machine Learning]

So this is just a sharpening of our tools… albeit a cool sharpening!

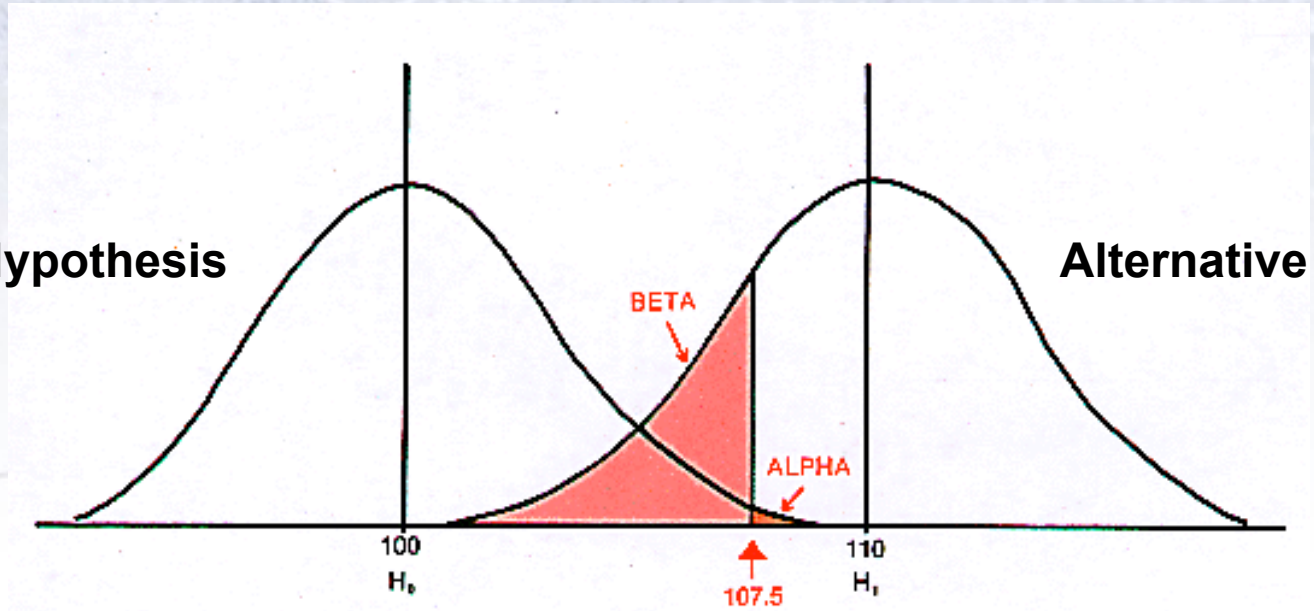
Why ML?

Classification



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

Classification



Machine Learning typically enables a better separation between hypothesis

DECISION:		
Reject Null	α Type I error	1 – β Correct

What is ML?

What is Machine Learning?

While there is no formal definition, an early attempt is the following intuition:

"Machine learning programs can perform tasks without being explicitly programmed to do so."

[Arthur Samuel, US computer pioneer 1901-1990]

"Little Peter is capable of finding his way home without being explicitly taught to do so."

What is Machine Learning?

While there is no formal definition, an early attempt is the following intuition:

"A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P if its performance at tasks in T, as measured by P, improves with experience E."

[T. Mitchell, "Machine Learning" 1997]

"Little Peter is said to learn from traveling around with respect to finding his way home and the time it takes, if his ability to find his way home, as measured by the time it takes, improves as he travels around."

Humans vs. ML

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:

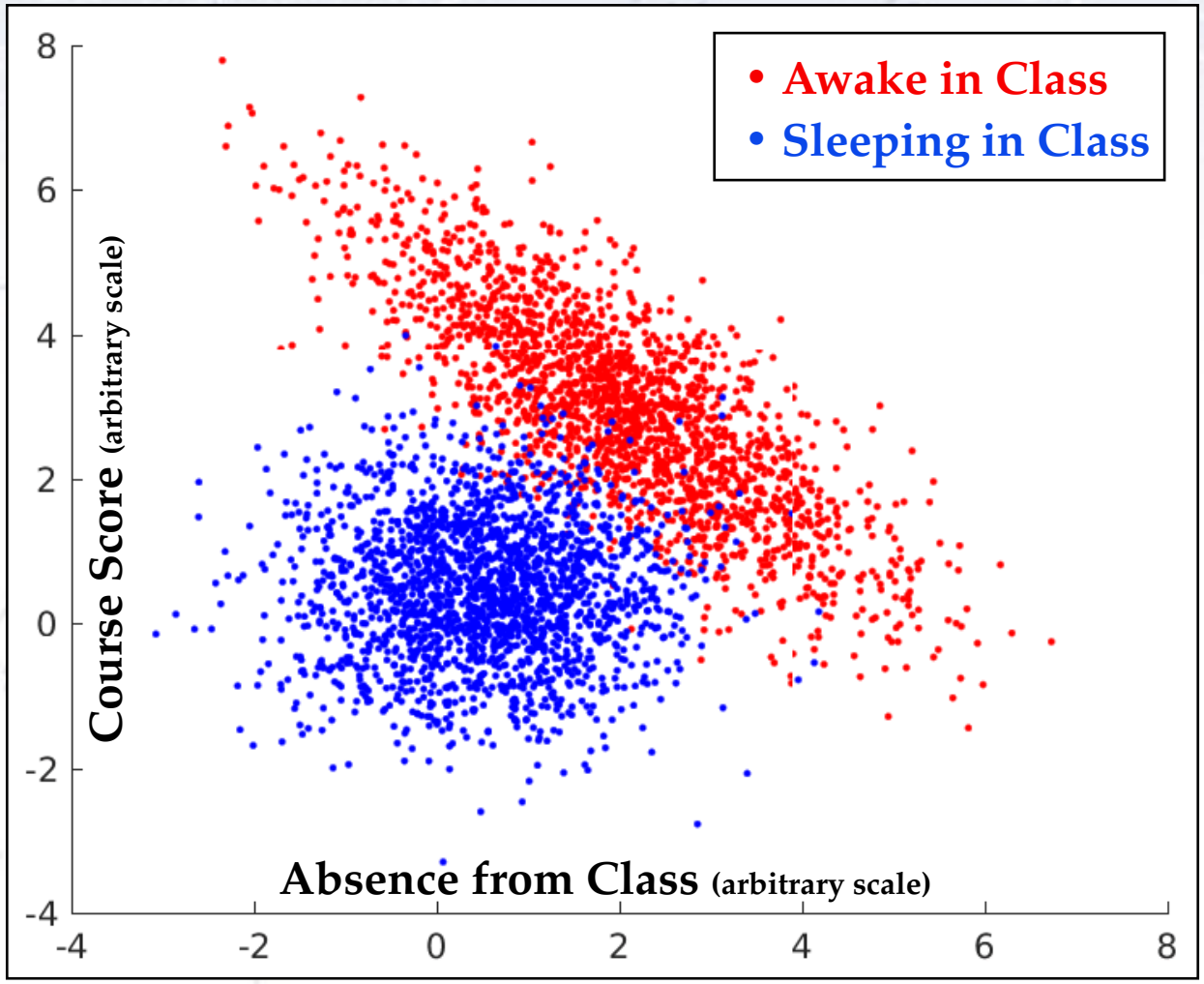
Dimensionality and Complexity

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:

Dimensionality and Complexity

Humans & Computers are good at seeing/understanding linear data in few dimensions:



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.

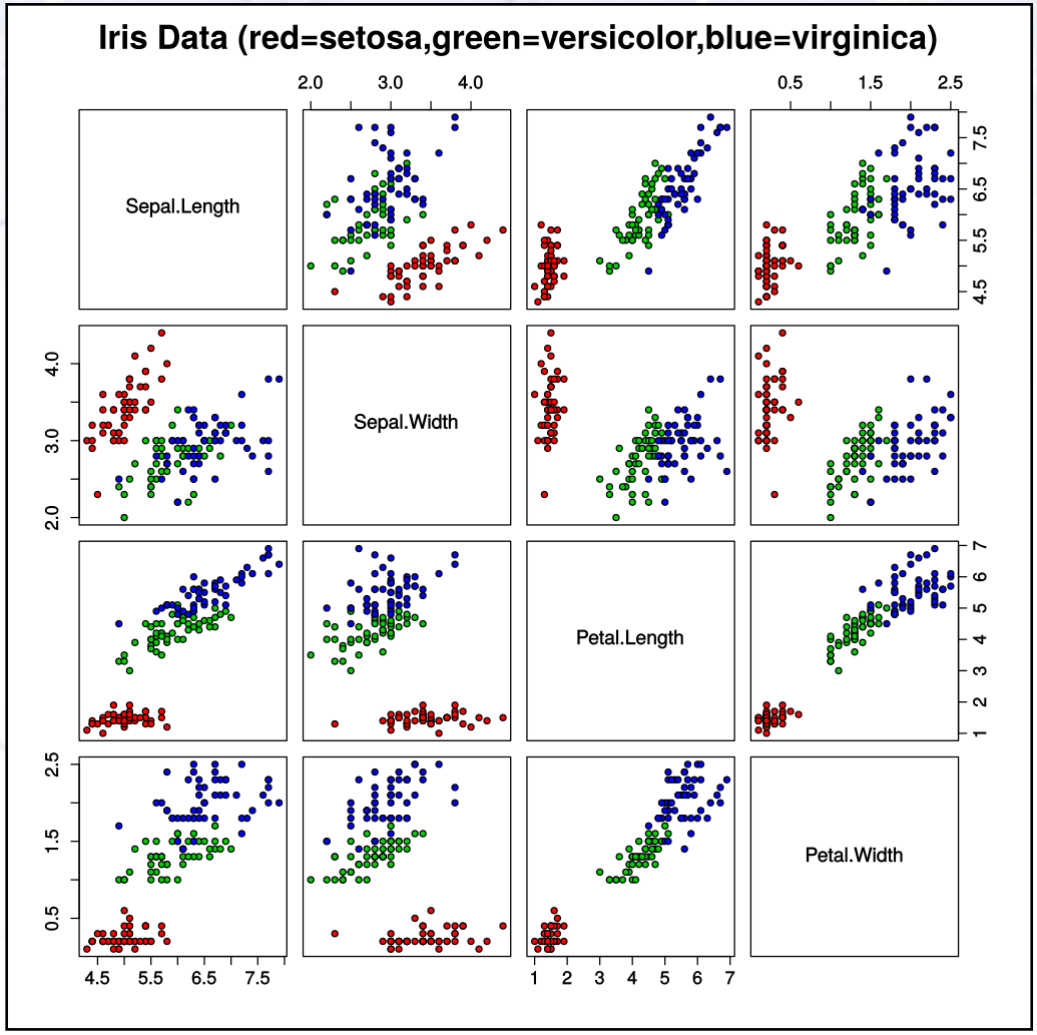
	Low dim.	High dim.
Linear	Humans: ✓ Computers: ✓	Humans: Computers:
Non-linear	Humans: Computers:	Humans: Computers:

Dimensionality and Complexity

However, when the dimensionality goes beyond 3D, we are lost, even for simple linear data. Computers are not…

Shown is the famous Fisher Iris dataset:
150 irises (3 kinds) with
4 measurements for each.

4 dimensional data!



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.

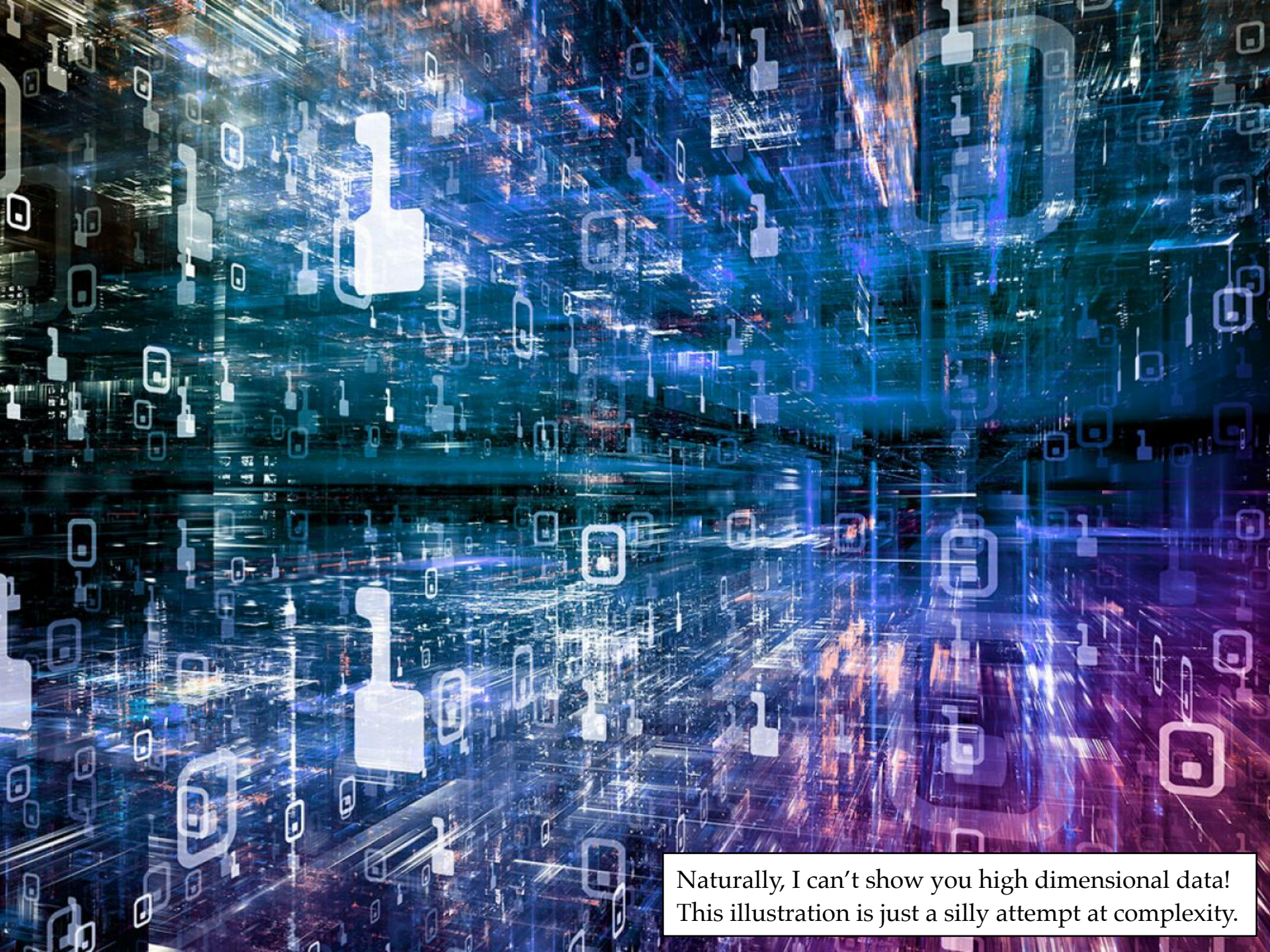

Jackson Pollock

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.


Naturally, I can’t show you high dimensional data!
This illustration is just a silly attempt at complexity.

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!

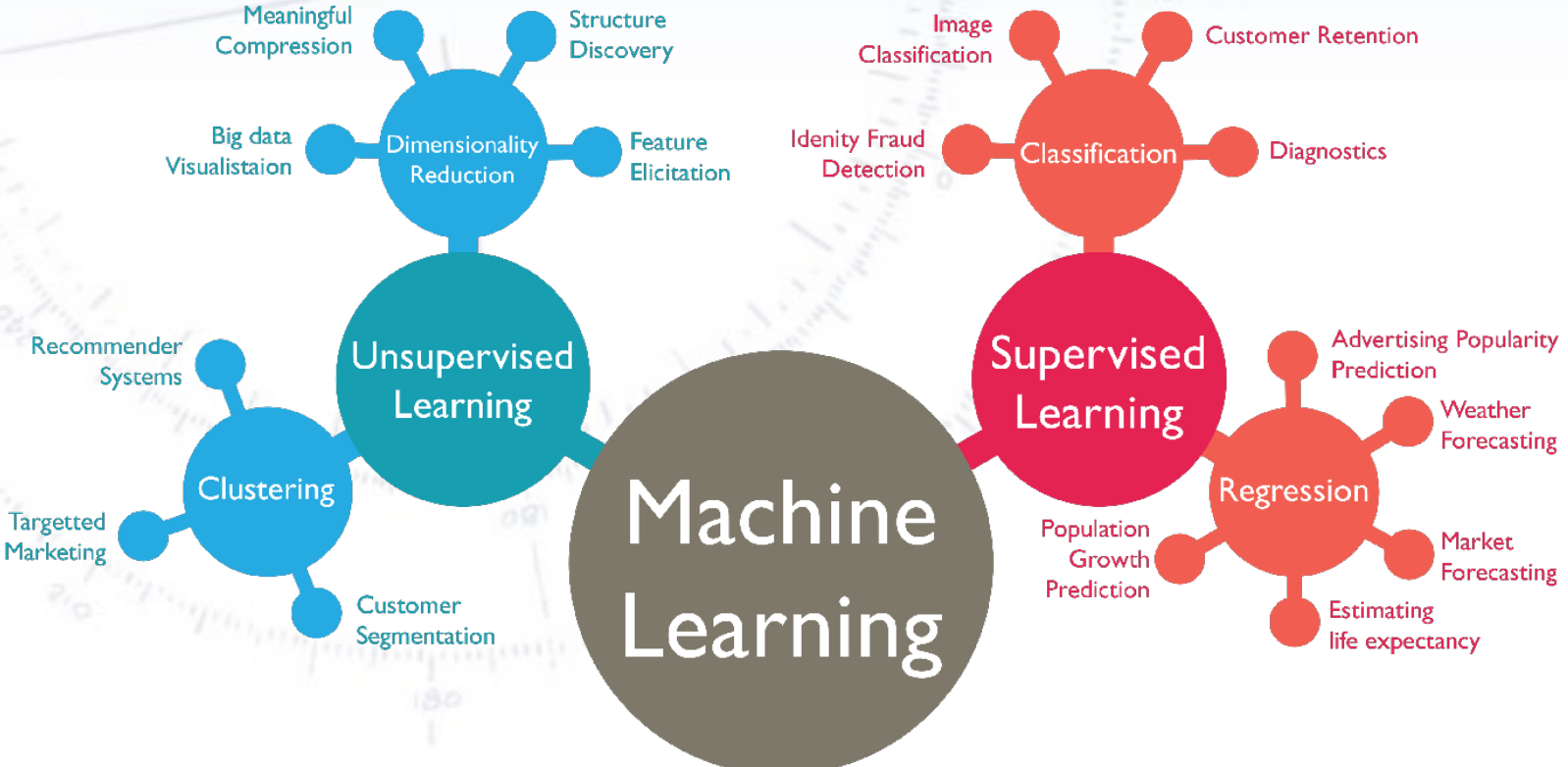
That is essentially what Machine Learning has enabled!

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).

We will be mostly on this side!

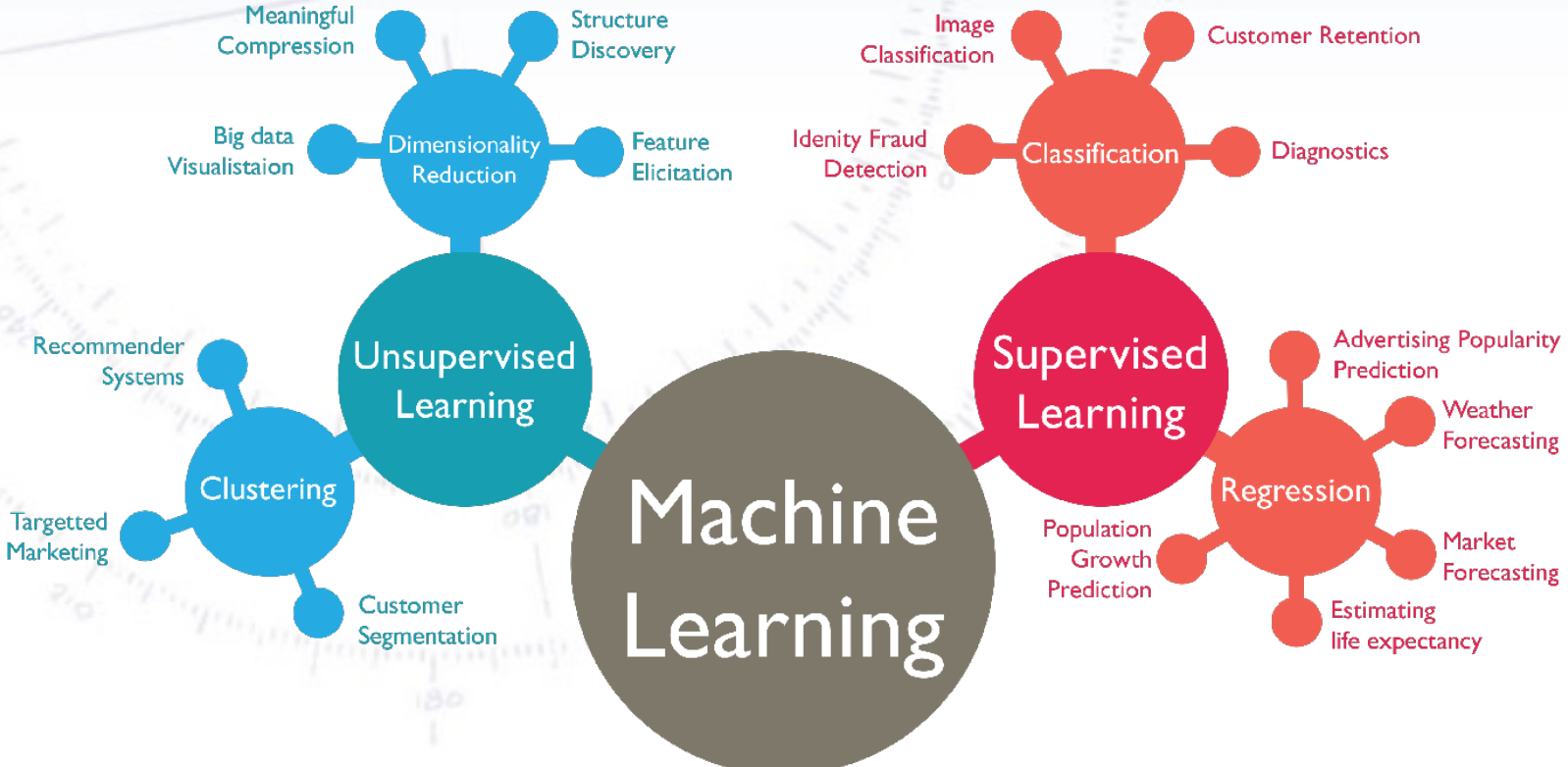


Unsupervised vs. Supervised
Classification vs. Regression

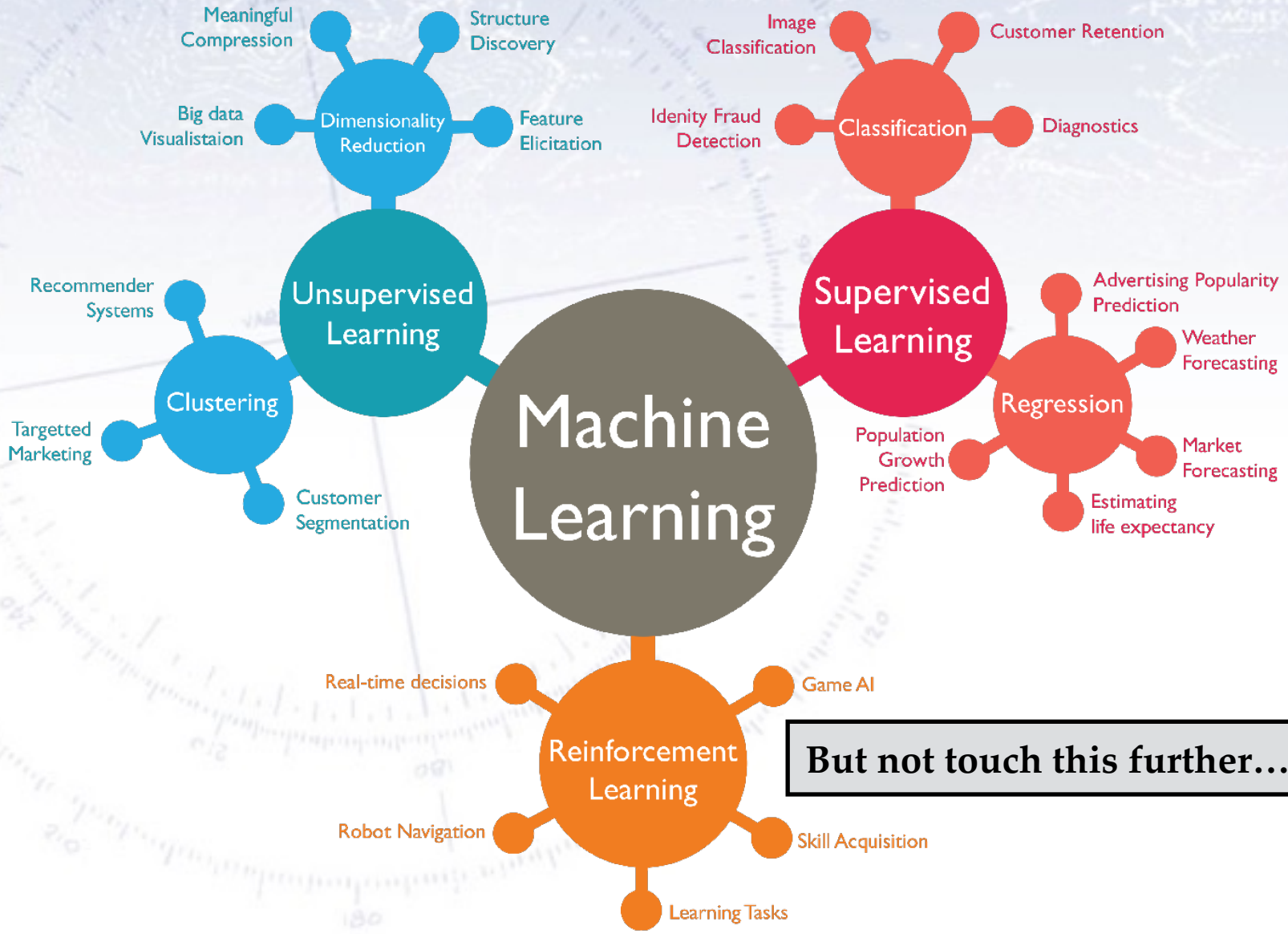
Machine Learning can be supervised (you have correctly labelled examples) or unsupervised (you don't)… [or reinforced]. Following this, one can be using ML to either classify (is it A or B?) or for regression (estimate of X).

And only briefly mention this side!

We will be mostly on this side!



Reinforcement Learning



But not touch this further…

Two main ingredients:

1. Solutions exists

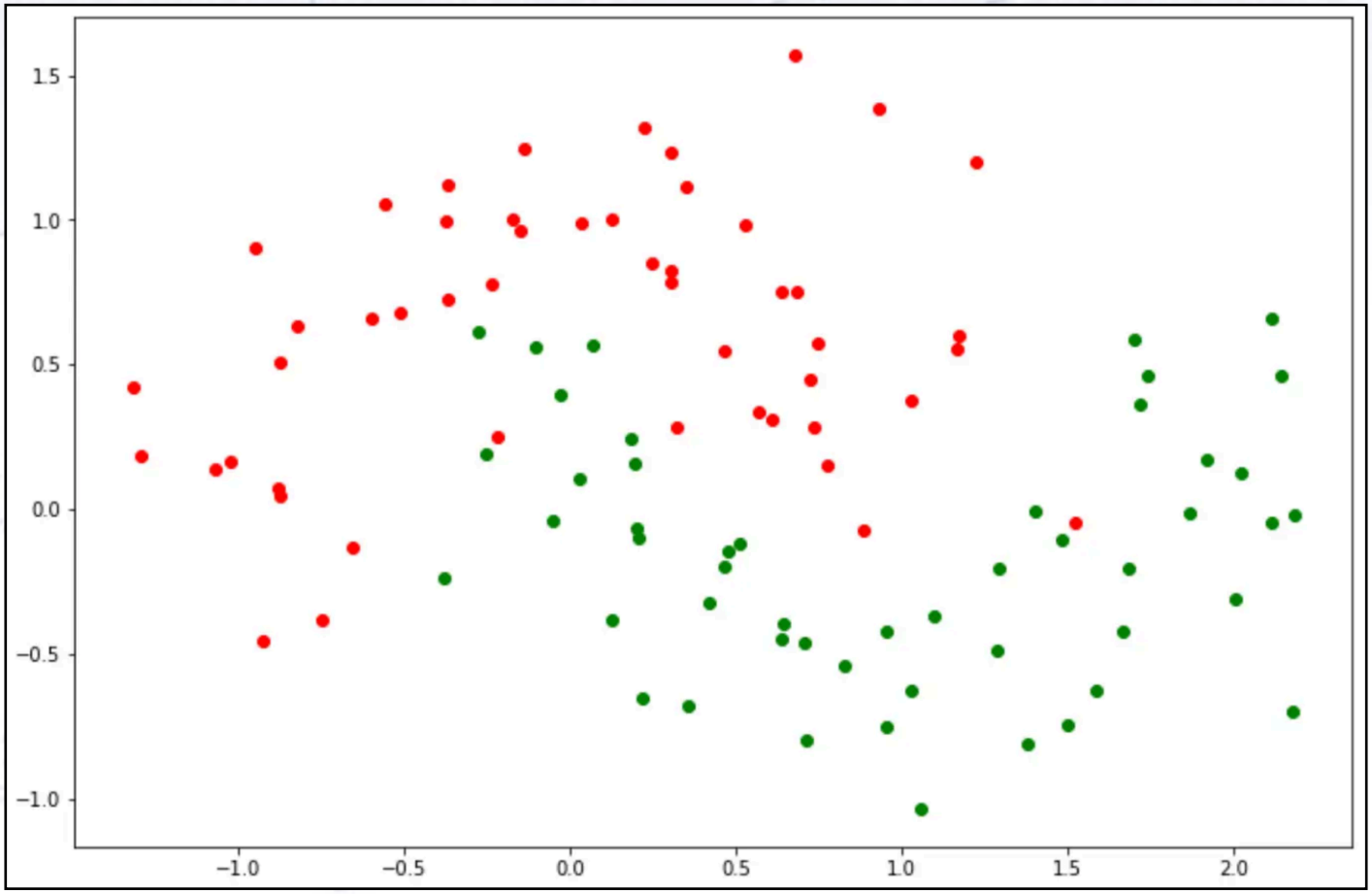
2. How to find them

Solutions exists

(Technically called Universal Approximation Theorems)

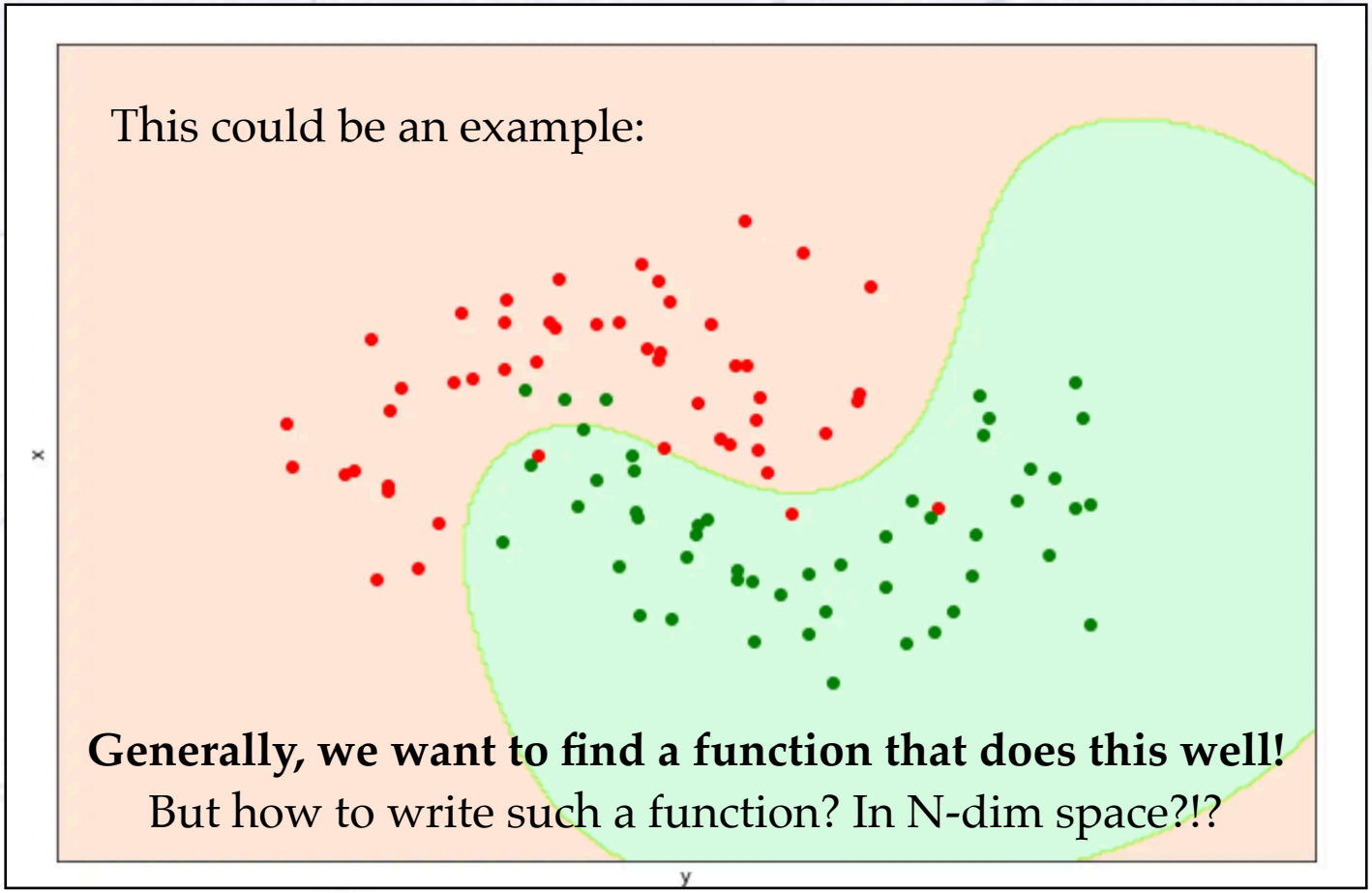
Where to separate?

Look at the red and green points, and imagine that you wanted to draw a curve that separates these.



Where to separate?

Look at the red and green points, and imagine that you wanted to draw a curve that separates these.

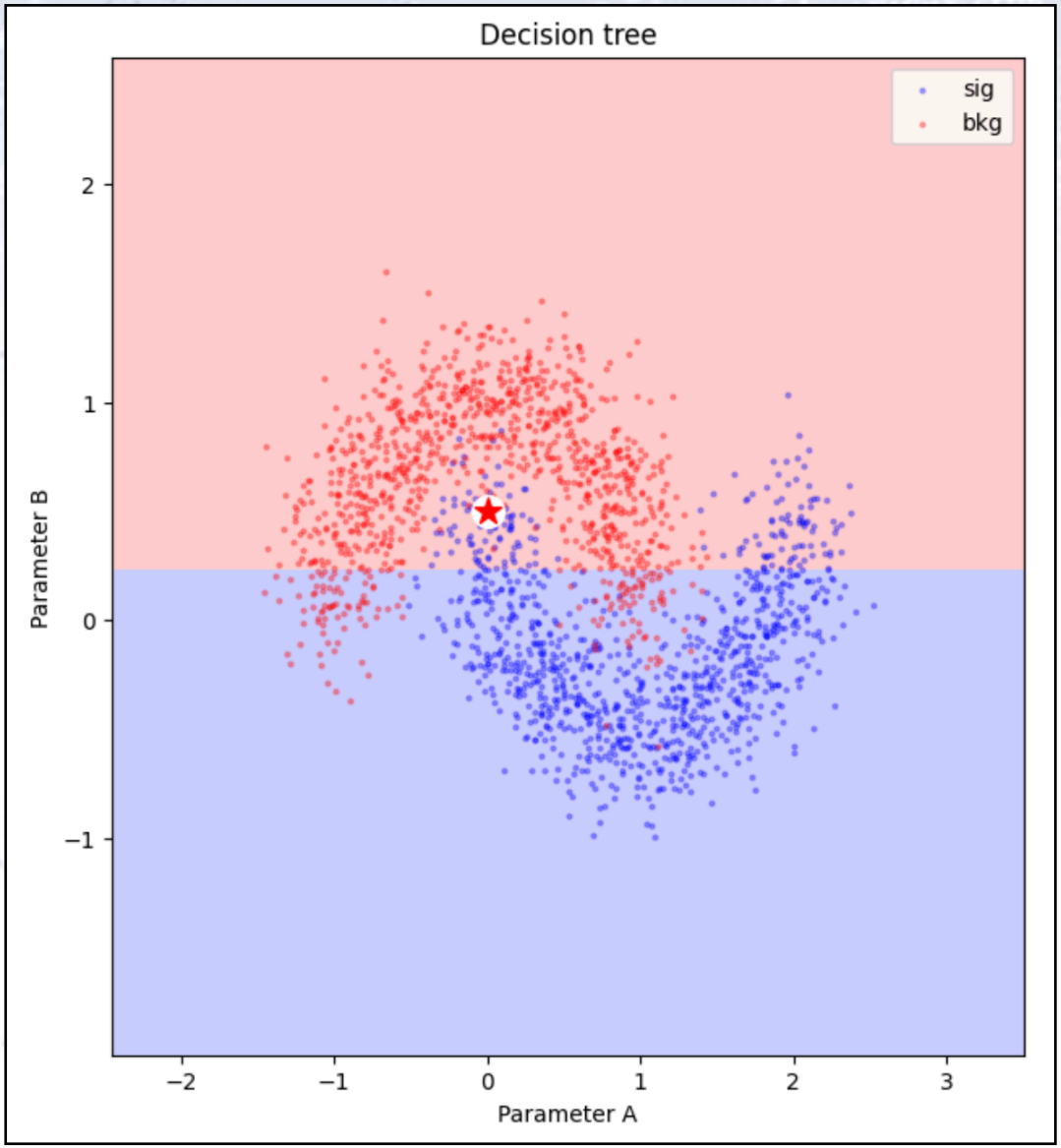


Universal Approx. Theorems

A simple function can be obtained simply by asking a lot of questions:
Question: Is B > 0.23?
Answer: Yes → Red
Answer: No → Blue

This question is illustrated in the drawing by the horizontal line with red and blue on the sides.

A Decision Tree consists of asking many such questions, corresponding to setting a lot of lines.



Universal Approx. Theorems

A simple function can be obtained simply by asking a lot of questions:
Question: Is B > 0.23?
Answer: Yes → Red
Answer: No → Blue

This question is illustrated in the drawing by the horizontal line with red and blue on the sides.

A Decision Tree consists of asking many such questions, corresponding to setting a lot of lines.

Universal Approx. Theorems

A simple function can be obtained simply by asking a lot of questions:
Question: Is B > 0.23?
Answer: Yes → Red
Answer: No → Blue

This question is illustrated in the drawing by the horizontal line with red and blue on the sides.

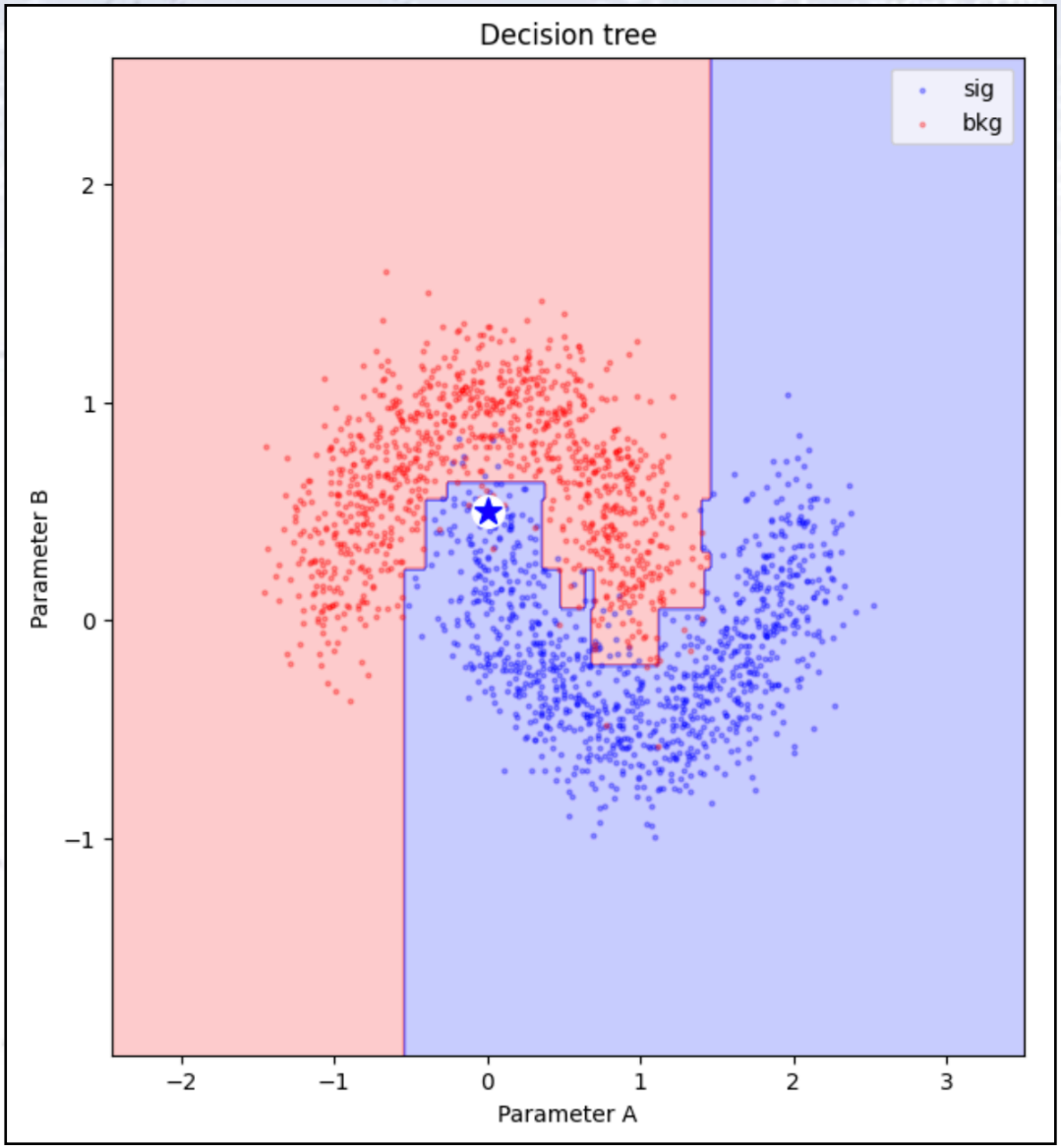
A Decision Tree consists of asking many such questions, corresponding to setting a lot of lines.

Universal Approx. Theorems

A simple function can be obtained simply by asking a lot of questions:
Question: Is B > 0.23?
Answer: Yes → Red
Answer: No → Blue

This question is illustrated in the drawing by the horizontal line with red and blue on the sides.

A Decision Tree consists of asking many such questions, corresponding to setting a lot of lines.

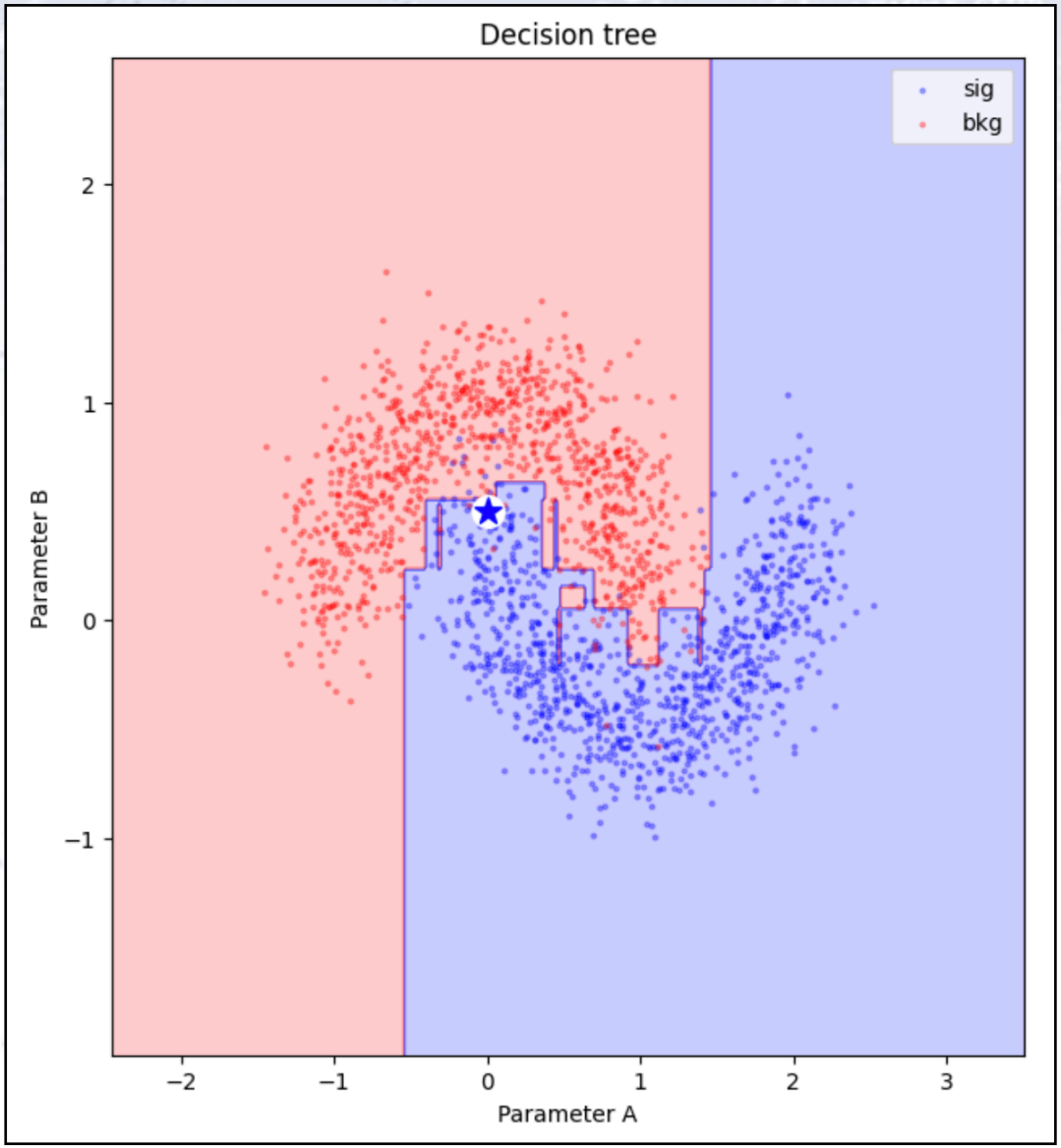


Universal Approx. Theorems

A simple function can be obtained simply by asking a lot of questions:
Question: Is B > 0.23?
Answer: Yes → Red
Answer: No → Blue

This question is illustrated in the drawing by the horizontal line with red and blue on the sides.

A Decision Tree consists of asking many such questions, corresponding to setting a lot of lines.

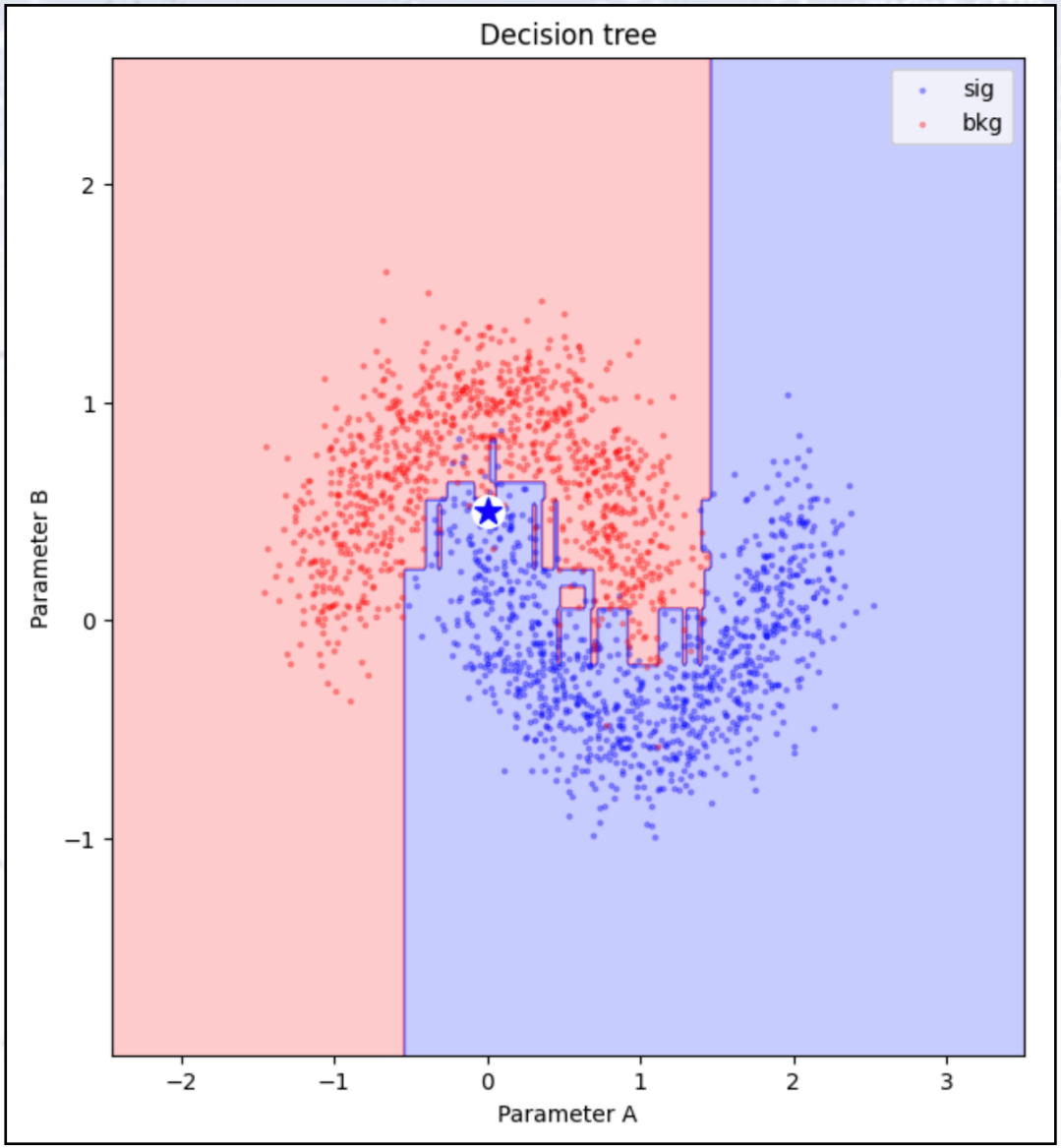


Universal Approx. Theorems

A simple function can be obtained simply by asking a lot of questions:
Question: Is B > 0.23?
Answer: Yes → Red
Answer: No → Blue

This question is illustrated in the drawing by the horizontal line with red and blue on the sides.

A Decision Tree consists of asking many such questions, corresponding to setting a lot of lines.



Universal Approximation Theorems

Theorem 5.1.1 (Universal Approximation Theorem) [10] *Let σ be a non-constant, bounded, and monotone-increasing continuous function. Let I_{m_0} denote the m_0-dimensional unit hypercube $[0,1]^{m_0}$. The space of continuous functions on I_{m_0} is denoted as $C(I_{m_0})$. Then given any function $f \in C(I_{m_0})$ and $\epsilon > 0$ there exists a set of real constants a_i, b_i and w_{ij}, where $i = 1, \ldots, m_1$ and $j = 1, \ldots, m_0$ such that we may define*

$$F(x_1, \ldots, x_{m_0}) = \sum_{i=1}^{m_1} a_i \sigma \left(\sum_{j=1}^{m_0} w_{ij} x_j + b_i \right) \tag{5.6}$$

as an approximate realization of the function f; that is,

$$|F(x_1, \ldots, x_{m_0}) - f(x_1, \ldots, x_{m_0})| < \epsilon \tag{5.7}$$

for all $x_1, x_2, \ldots, x_{m_0}$ that lie in the input space.

Universal Approximation Theorems

Theorem 5.1.1 (Universal Approximation Theorem) [10] *Let σ be a non-*

Summary:

Neural Networks etc. can approximate functions in any dimension very well!

$$F(x_1, \ldots, x_{m_0}) = \sum_{i=1} a_i \sigma \left(\sum_{j=1} w_{ij} x_j + b_i \right) \tag{5.6}$$

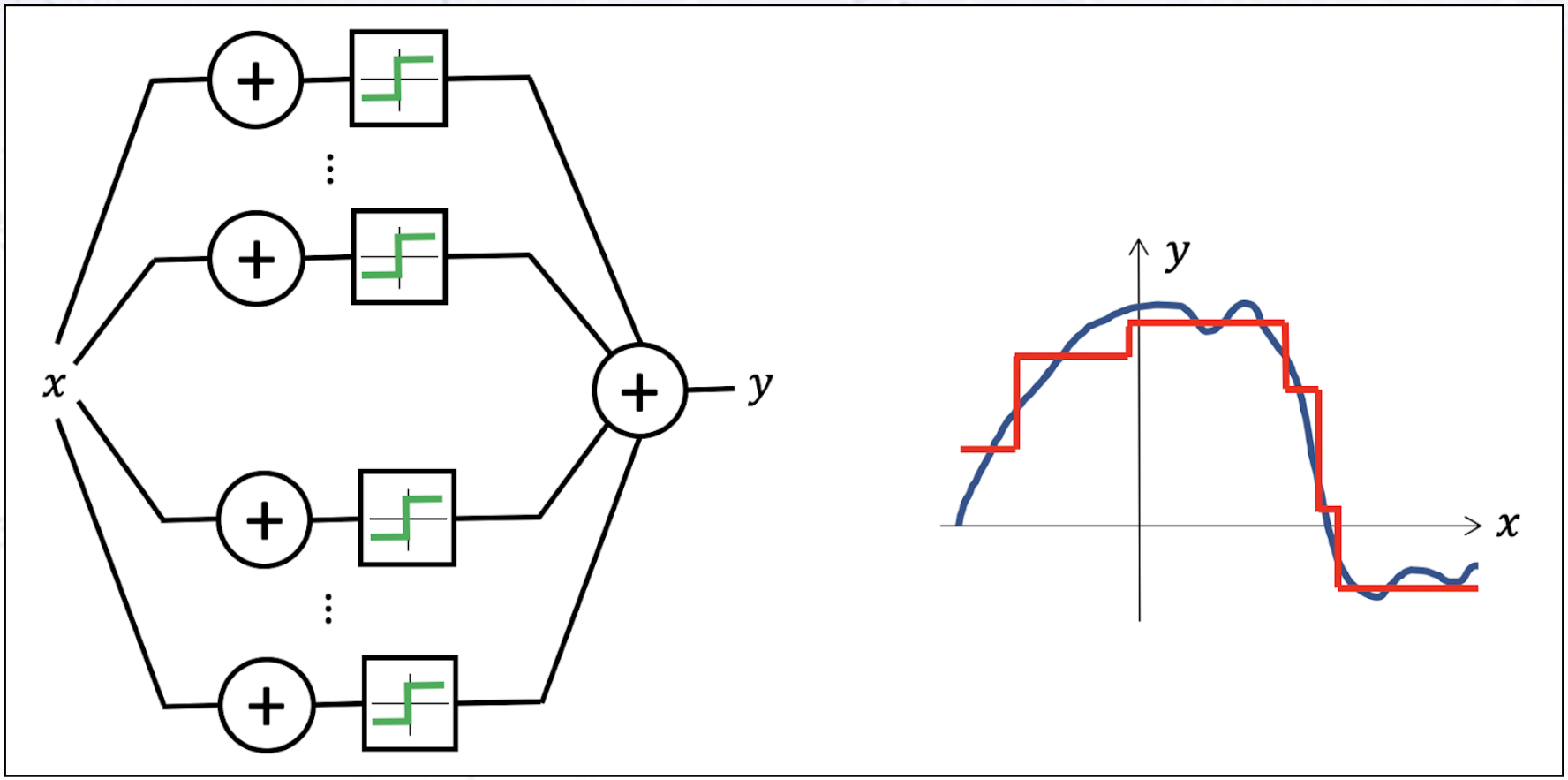
as an approximate realization of the function f; that is,

$$|F(x_1, \ldots, x_{m_0}) - f(x_1, \ldots, x_{m_0})| < \epsilon \tag{5.7}$$

for all $x_1, x_2, \ldots, x_{m_0}$ that lie in the input space.

Universal Approx. Theorems

Such approximations typically entails a **large amount of parameters**, for which the UATs give no recipe on how to find - only that such a construction is possible.



How to find these

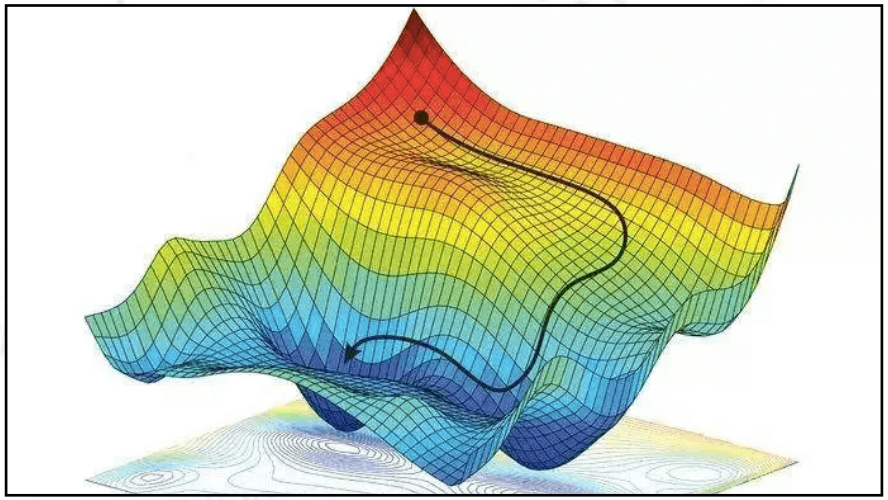
(Technically called Stochastic Gradient Descent)

Stochastic Gradient Descent

The way to obtain the parameters/weights of ML algorithms, is generally by **Stochastic Gradient Descent**.

This "back propagation" algorithm works by computing the gradient of the loss function (to be optimised) with respect to each weight using the chain rule.

One thus computes the gradient one layer at a time, iterating backwards from the last layer (avoiding redundancies). See Goodfellow et al. for details.



Goodfellow, Ian; Bengio, Yoshua; Courville, Aaron (2016). "6.5 Back-Propagation and Other Differentiation Algorithms". *Deep Learning*. MIT Press. pp. 200–220. ISBN 9780262035613.

(Normal) Gradient Descent

The choice of loss function, L, depends on the problem at hand, and in particular what you find important! You want to minimise this with respect to the model parameters θ:

$$L(\theta) = \frac{1}{N}\sum_{i}^{N} L_i(\theta)$$

In order to find the optimal solution, one can use Gradient Descent, typically **based on the whole dataset**:

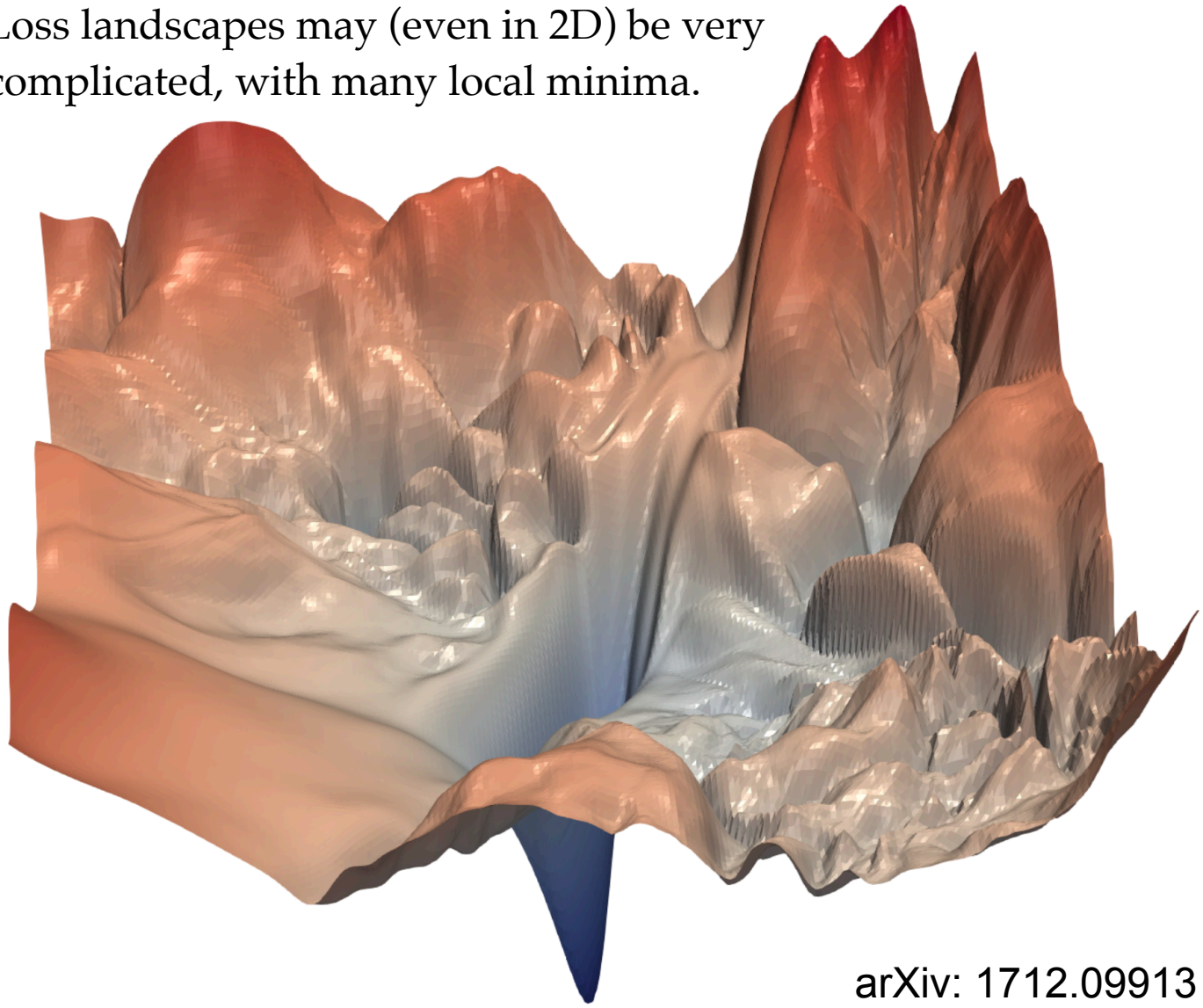
$$\theta_{j+1} = \theta_j - \eta \nabla L(\theta) = \theta_j - \frac{\eta}{N}\sum_{i}^{N} \nabla L_i(\theta)$$

This is the procedure used by e.g. Minuit and other minimisation routines.

Note the very important parameter: **Learning rate η**.

(Nasty) Loss Landscapes

Loss landscapes may (even in 2D) be very complicated, with many local minima.



arXiv: 1712.09913

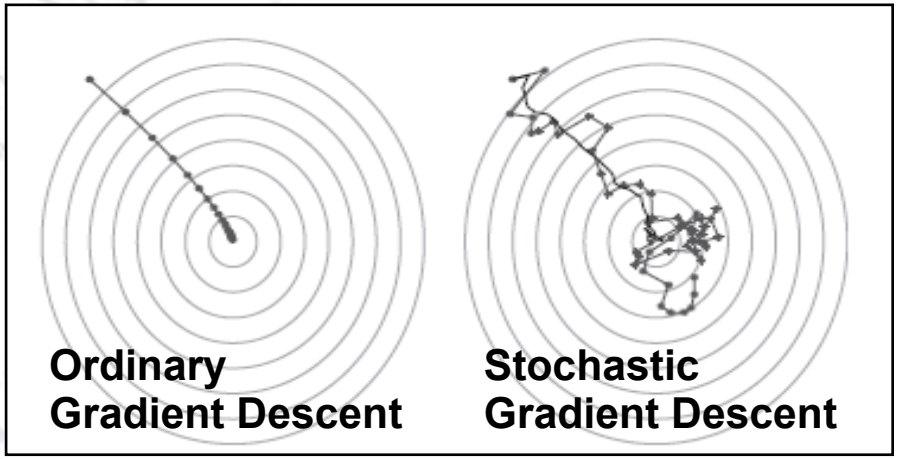
Stochastic Gradient Descent

The way to obtain the parameters/weights of ML algorithms, is generally by **Stochastic Gradient Descent**.

This "back propagation" algorithm works by computing the gradient of the loss function (to be optimised) with respect to each weight using the chain rule.

One thus computes the gradient one layer at a time, iterating backwards from the last layer (avoiding redundancies). See Goodfellow et al. for details.

The gradient descent is made stochastic (and fast) by only considering a fraction (called a "batch") of the data, when calculating the step in the search for optimal parameters for the algorithm. This allow for stochastic jumping, that avoids local (false) minima.



Goodfellow, Ian; Bengio, Yoshua; Courville, Aaron (2016). "6.5 Back-Propagation and Other Differentiation Algorithms". *Deep Learning*. MIT Press. pp. 200–220. ISBN 9780262035613.

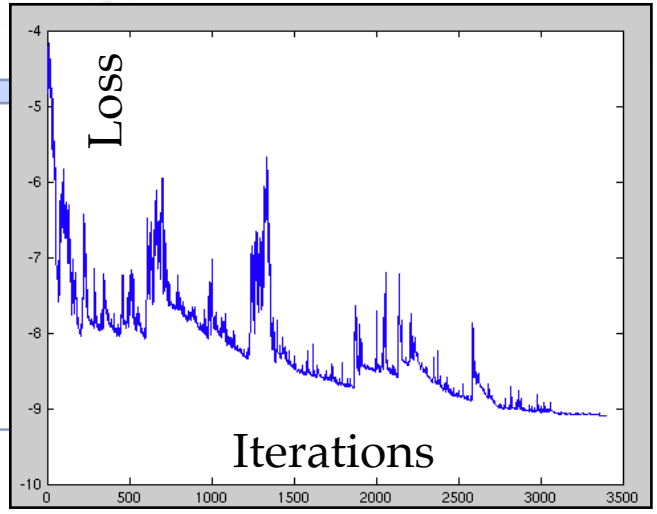
Stochastic Gradient Descent

In order to give the gradient descent some degree of "randomness" (stochastic), one evaluates the below function **for small batches** instead of the full dataset.

$$\theta_{j+1} = \theta_j - \eta \nabla L(\theta) = \theta_j - \frac{\eta}{N} \sum_{i}^{N} \nabla L_i(\theta)$$

The algorithm thus becomes:

- Choose an initial vector of parameters w and learning rate η.
- Repeat until an approximate minimum is obtained:
  - Randomly shuffle examples in the training set.
  - For $i = 1, 2, \ldots, n$, do:
    - $w := w - \eta \nabla Q_i(w)$.



Not only does this vectorise well and gives smoother descents, but with decreasing learning rate, **it "almost surely" finds the global minimum** (Robbins-Siegmund theorem).

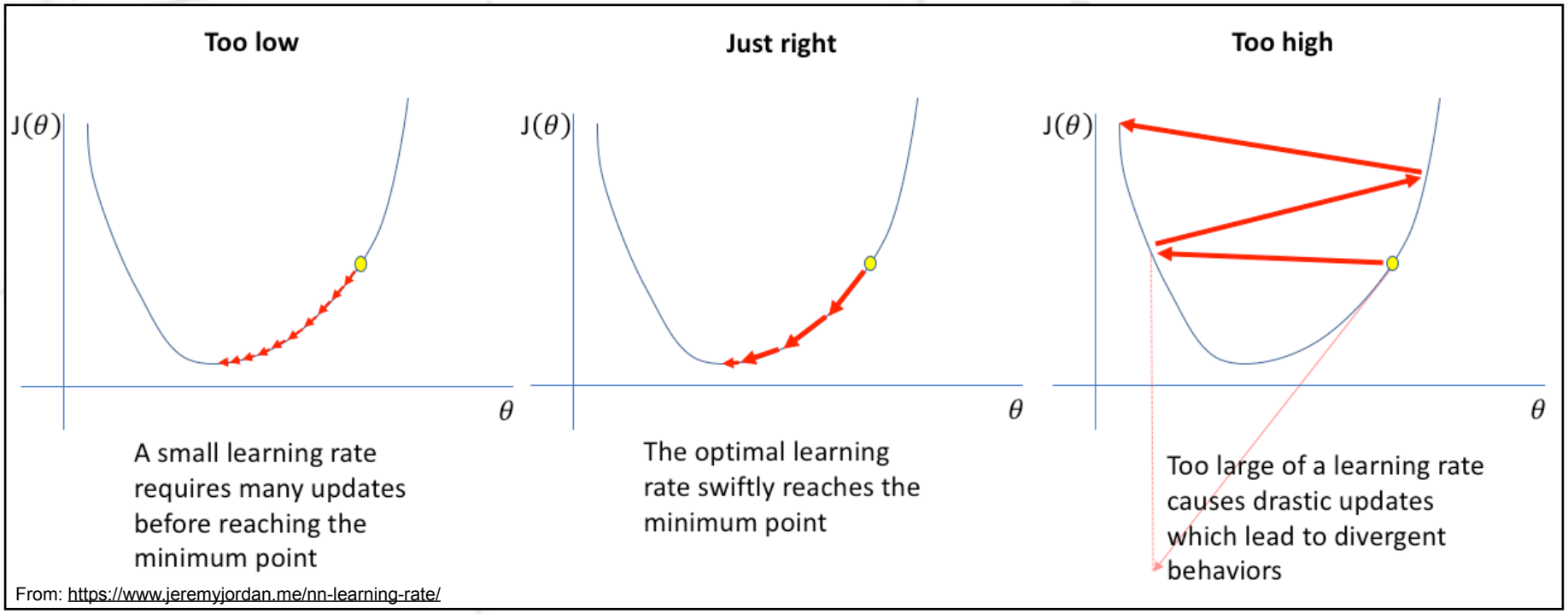
Learning Rate Schedulers

But, there is **no reason to consider a fixed value for the learning rate**!

More practically, one would typically adapt the learning rate to the situation:

- When exploring: Use larger learning rate.
- When exploiting: Use lower learning rate (when converging).

Below is illustrated what happens, when the learning rate is right/wrong.



From: https://www.jeremyjordan.me/nn-learning-rate/

Ingredients for ML

So now we know that at least in principle:

- a solution exists (Universal Approximation Theorem) and
- that it can be found (Stochastic Gradient Descent).

But this does not in reality make us capable of getting ML results.

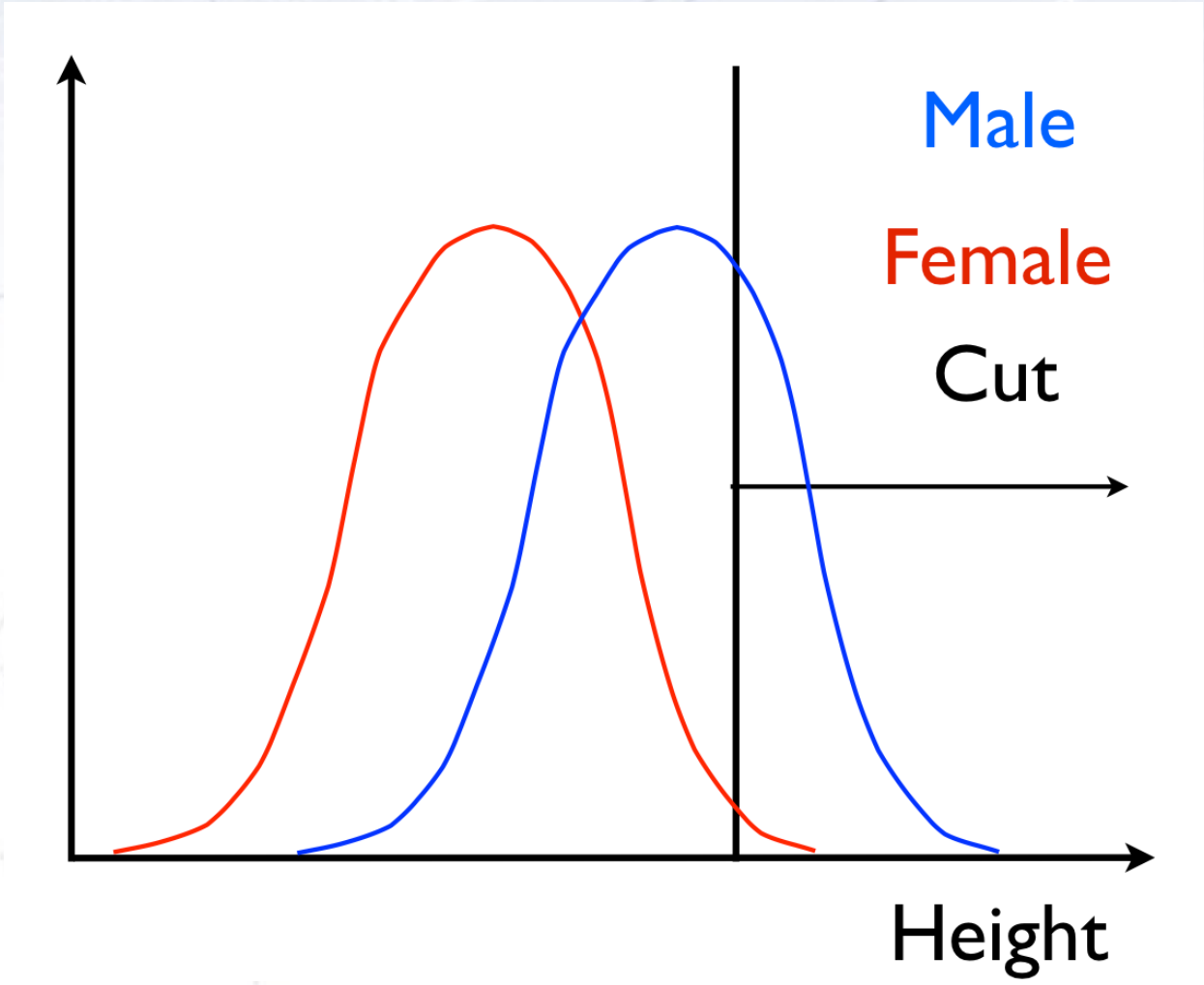
We (at least) also need:

- actual functions/algorithms for making approximations
  Boosted Decision Trees (BDTs) & Neural Networks (NNs)
- knowledge about how to tell them what to learn
  Loss functions (and how to minimise these)
- a scheme for how to use the data we have available
  Training, validation, and testing samples & Cross Validation

The linear analysis case

Simple Example

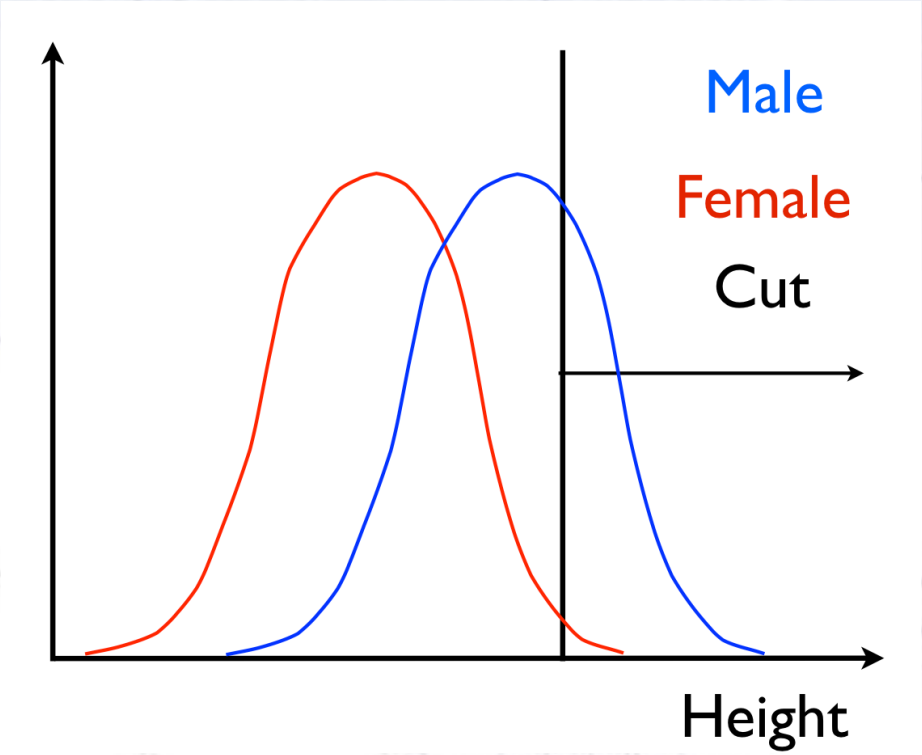
Problem: You want to figure out a method for getting sample that is mostly male!
Solution: Gather height data from 10000 people, Estimate cut with 95% purity!

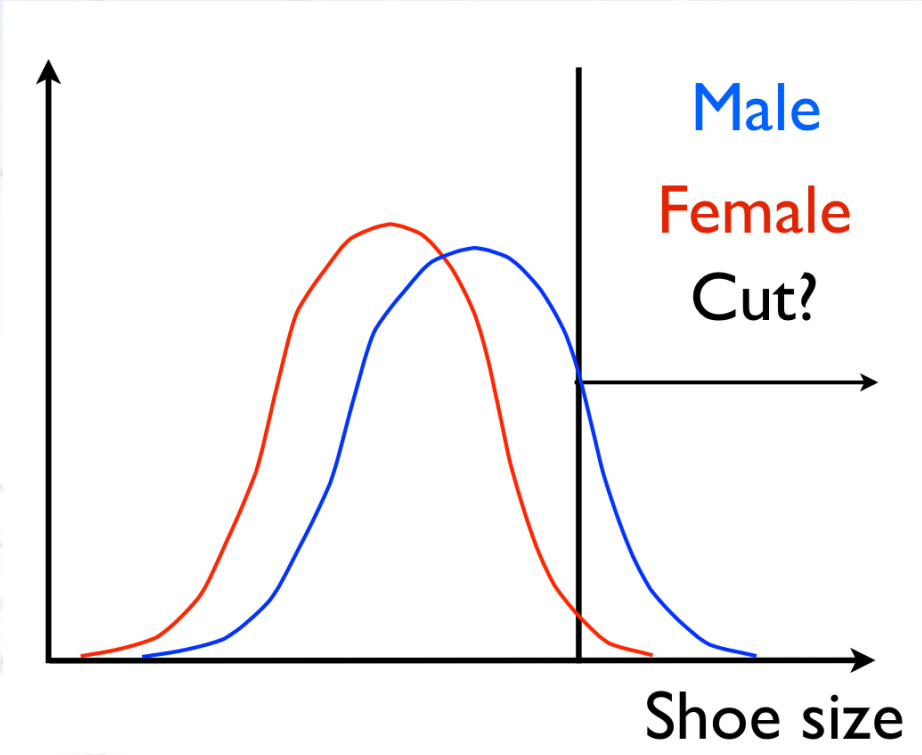


Simple Example

Additional data: The data you find also contains shoe size!

How to use this? Well, it is more information, but should you cut on it?





The question is, what is the best way to use this (possibly correlated) information!

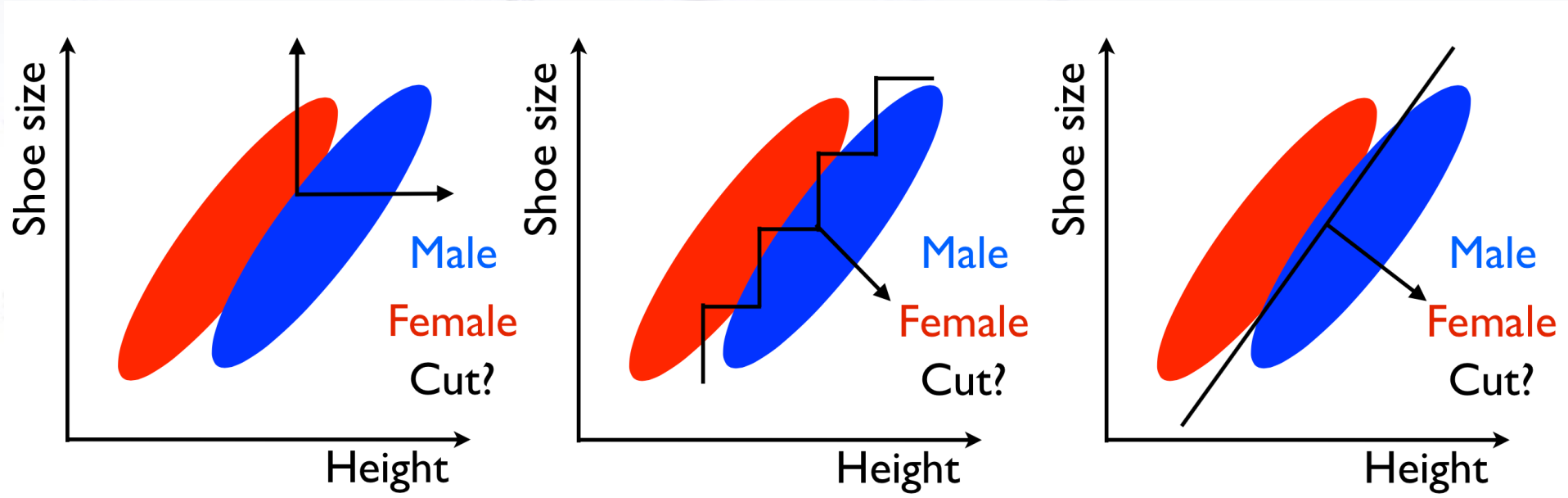
Simple Example

So we look if the data is correlated, and consider the options:

Cut on each var?
Poor efficiency!

Advanced cut?
Clumsy and hard to implement

Combine var?
Smart and promising



The latter approach is the Fisher discriminant!
It has the advantage of being simple and applicable in many dimensions easily!

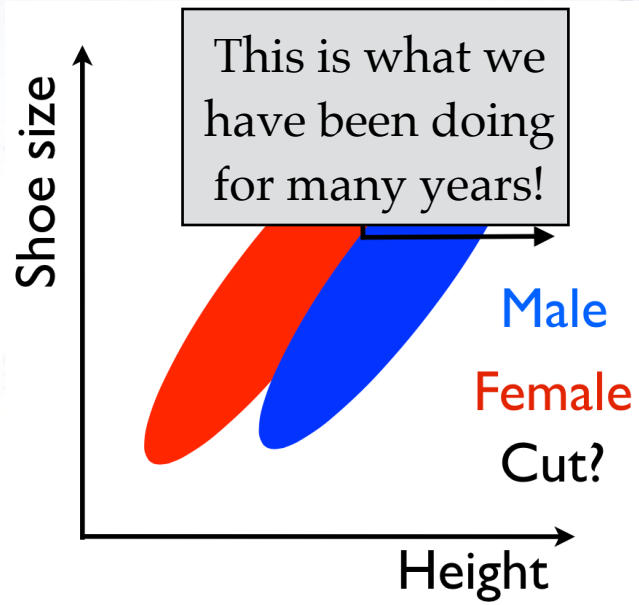
Simple Example

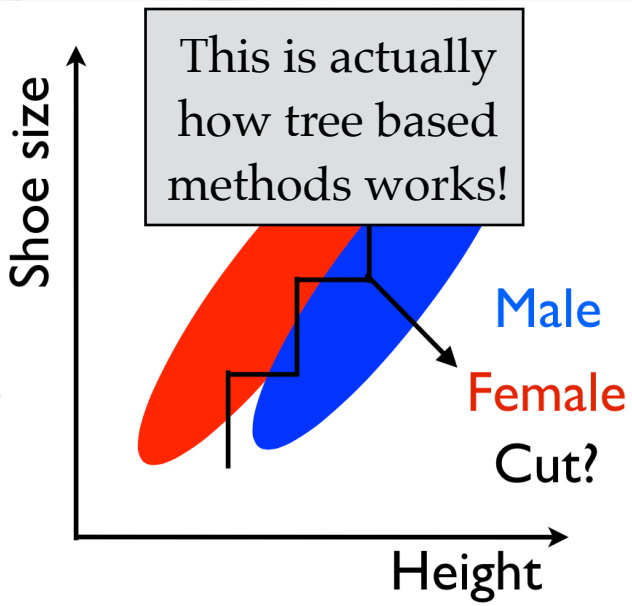
So we look if the data is correlated, and consider the options:

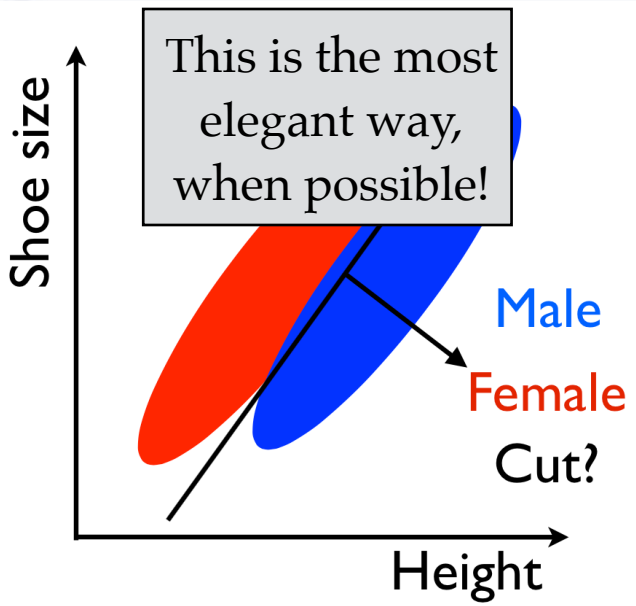
Cut on each var?
Poor efficiency!

Advanced cut?
Clumsy and hard to implement

Combine var?
Smart and promising



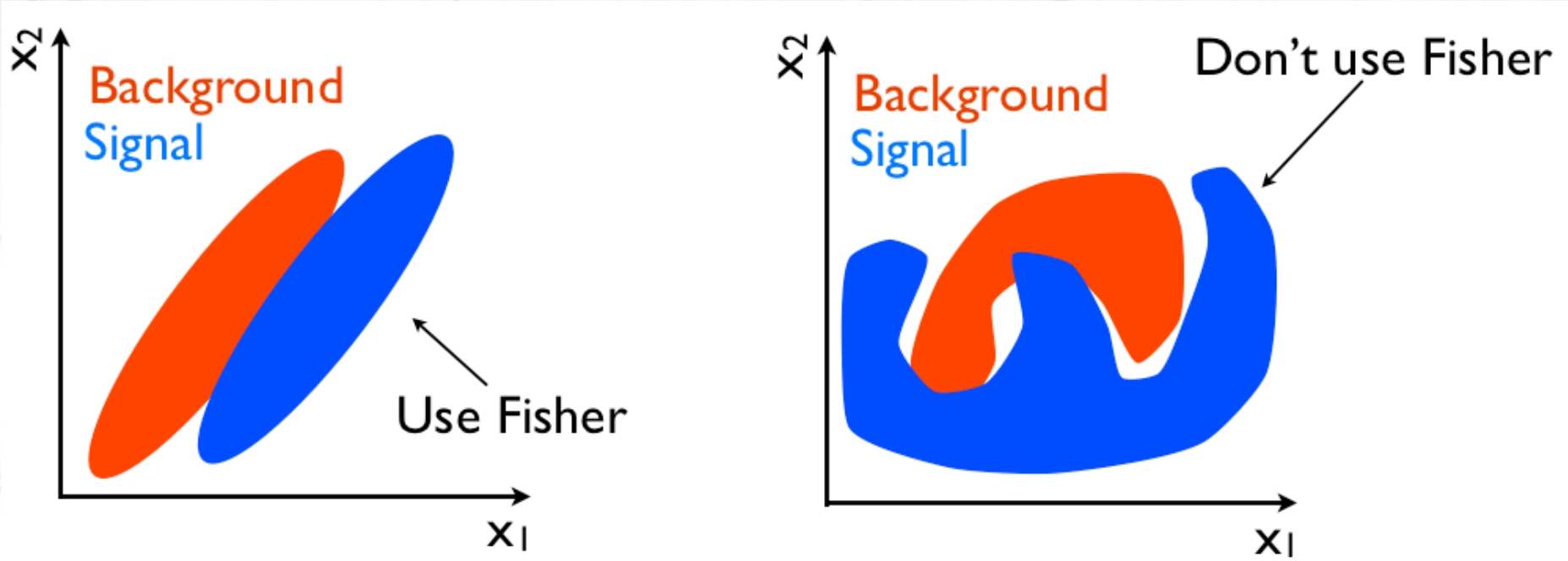




The latter approach is the Fisher discriminant!
It has the advantage of being simple and applicable in many dimensions easily!

Non-linear cases

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 hence one should move on to the Fisher. However, if correlations are non-linear, more "tough" methods are needed...

Tree based models

Decision tree learning

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

because it:

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

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

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

Again, for tabular data, I tend to disagree with the last statement!

Decision Trees

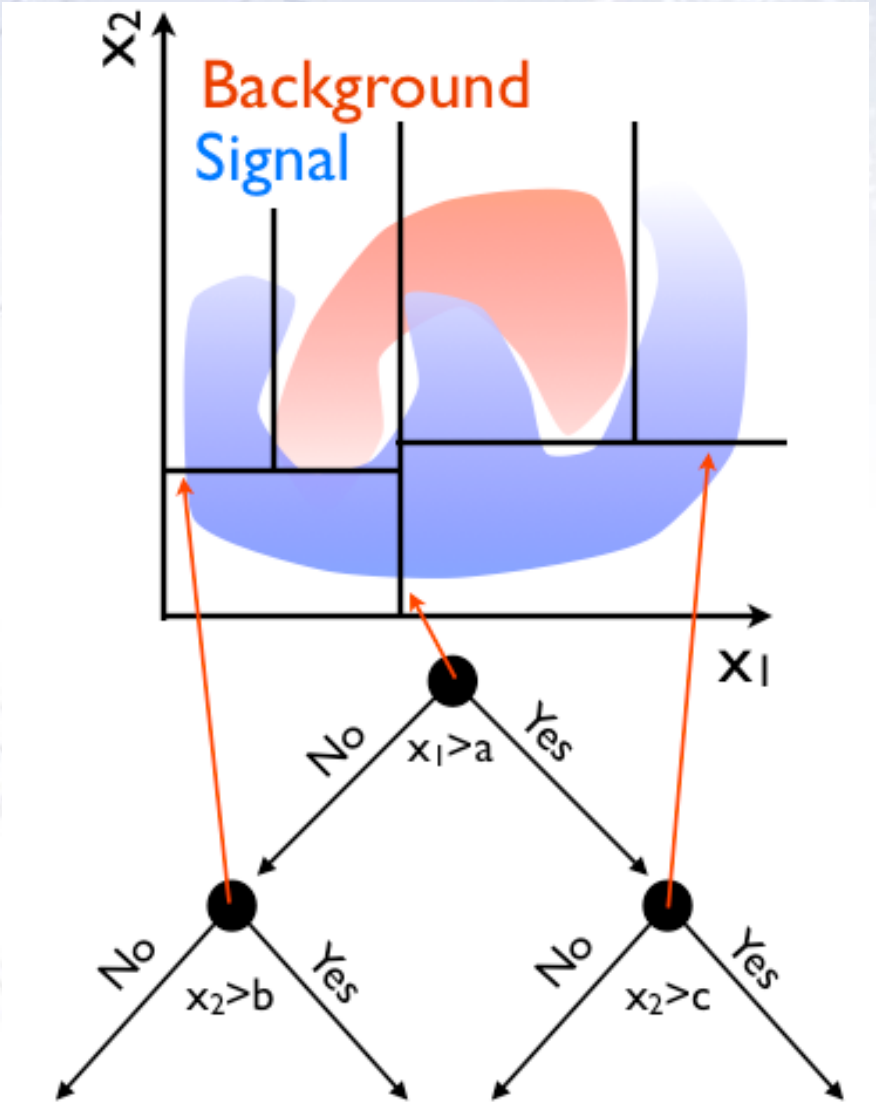
A decision tree divides the parameter space, starting with the maximal separation. In the end each part has a probability of being signal or background.

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

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

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

Decision trees are from before 1980!!!



Boosting… using many trees!

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

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

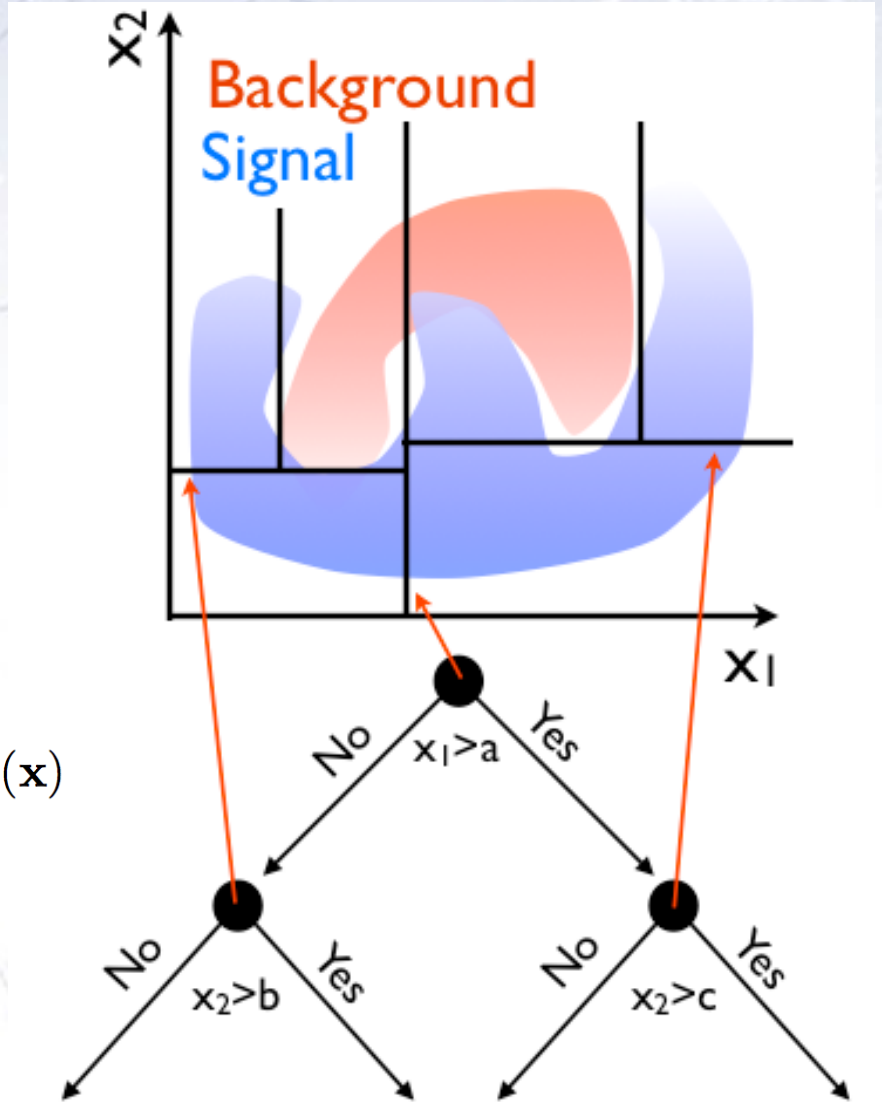
First classifier

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

Parameters in event N

Individual tree

Boosting is from 1997 (AdaBoost).



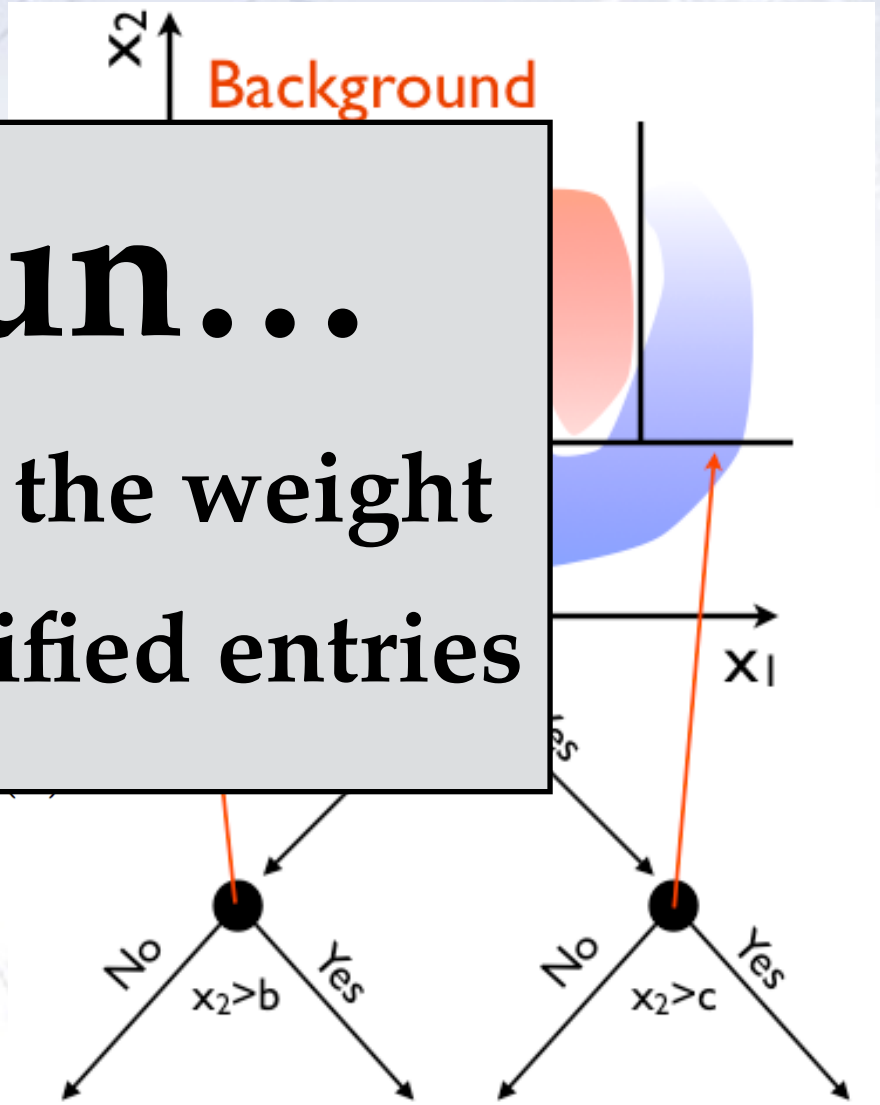
Boosting… using many trees!

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

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

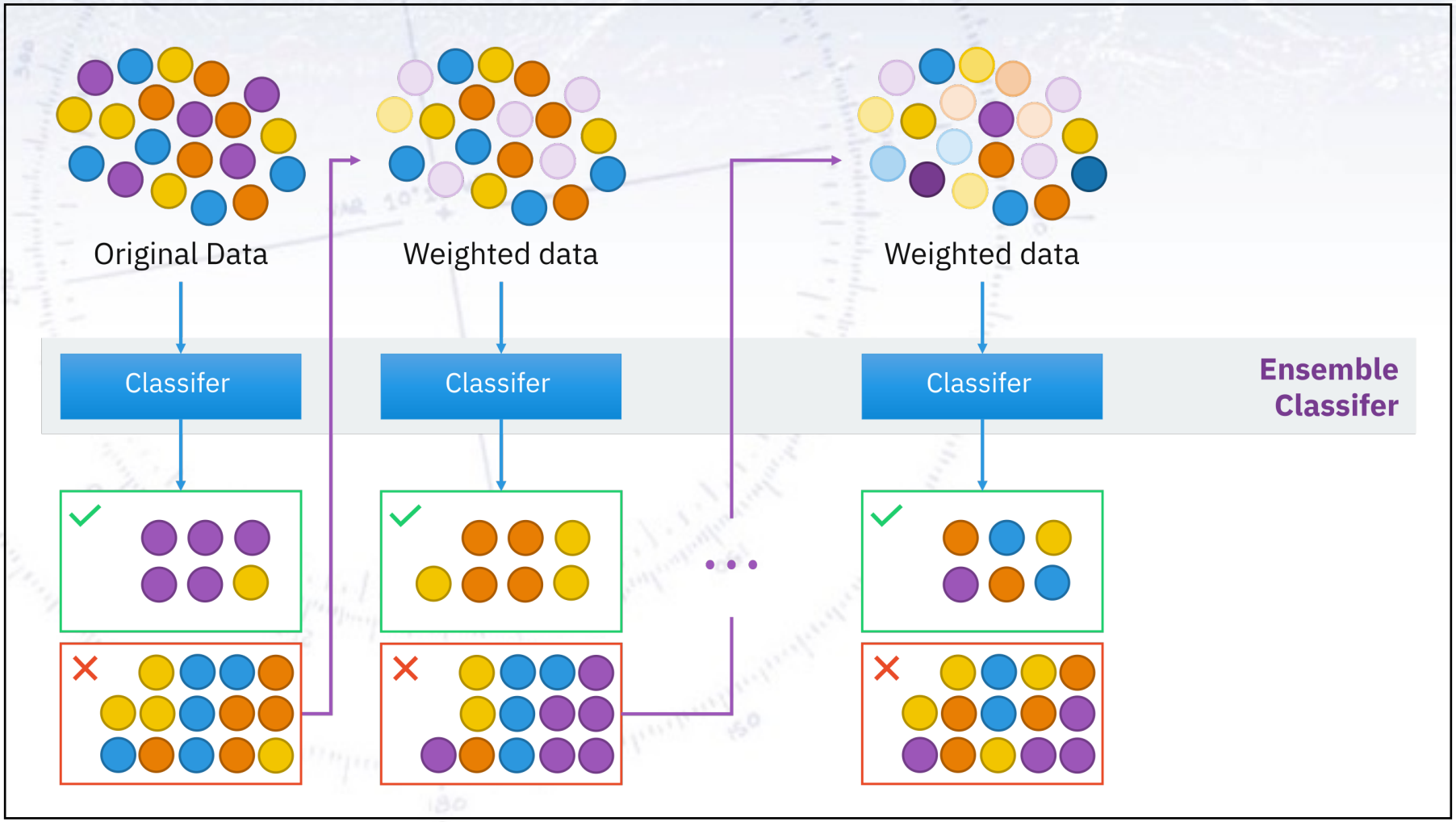
Parameters in event N

Individual tree

Boosting is from 1997 (AdaBoost).

Boosting illustrated

Boosting provides a reweighing scheme giving harder cases higher weights.
At the end of training, the trees are collected into an "ensemble classifier".



Where to split?

How does the algorithm decide which variable to split on and where to split?

There are several ways in which this can be done, and there is a difference between how to do it for classification and regression. But in general, one would like to **make the split, which maximises the improvement gained by doing so.**

In **classification**, one often uses the average binary cross entropy (aka. "log-loss"):

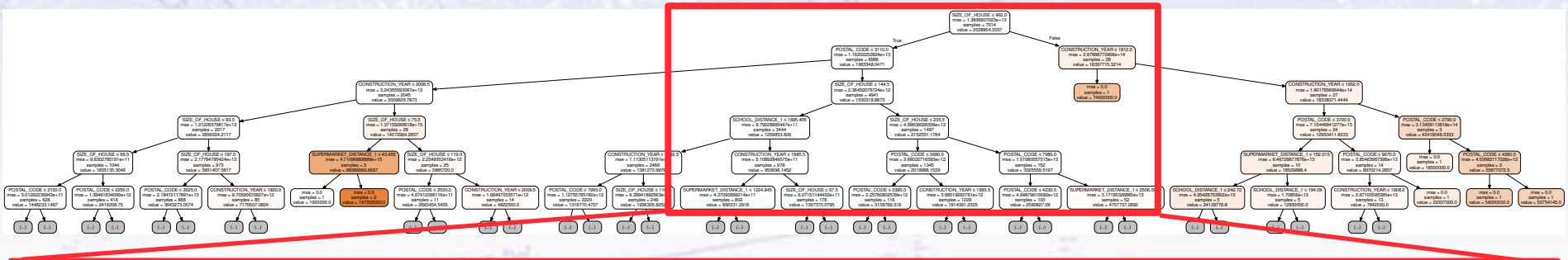
$$\mathcal{L} = -\frac{1}{N}\sum_{n=1}^{N}\left[y_n \log \hat{y}_n + (1-y_n)\log(1-\hat{y}_n)\right]$$

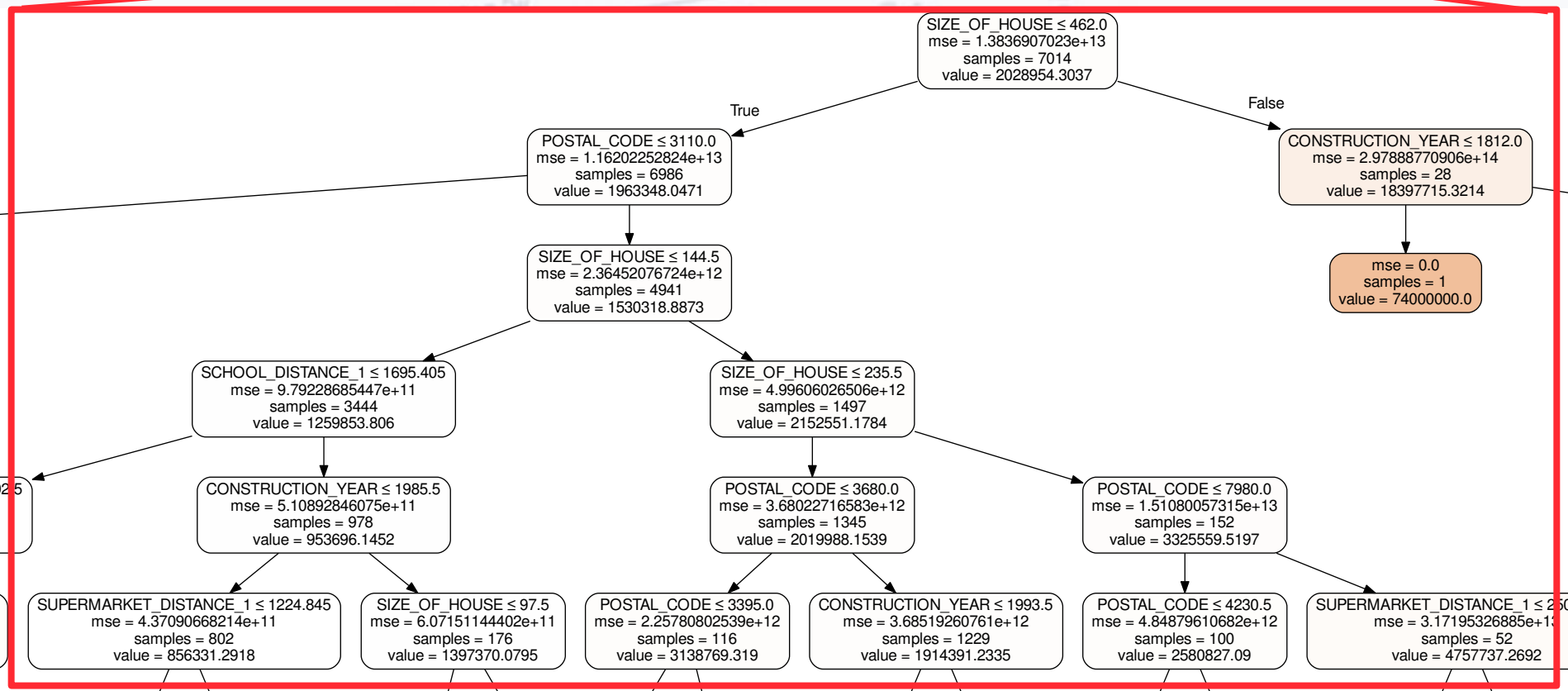
Here, y_n is the truth, while $\hat{y}_n$ is the estimate (in [0,1]).

Other alternatives include using Gini coefficients, Variance reduction, and even ChiSquare. However, in classification the above is somewhat "standard".

Housing Prices decision tree

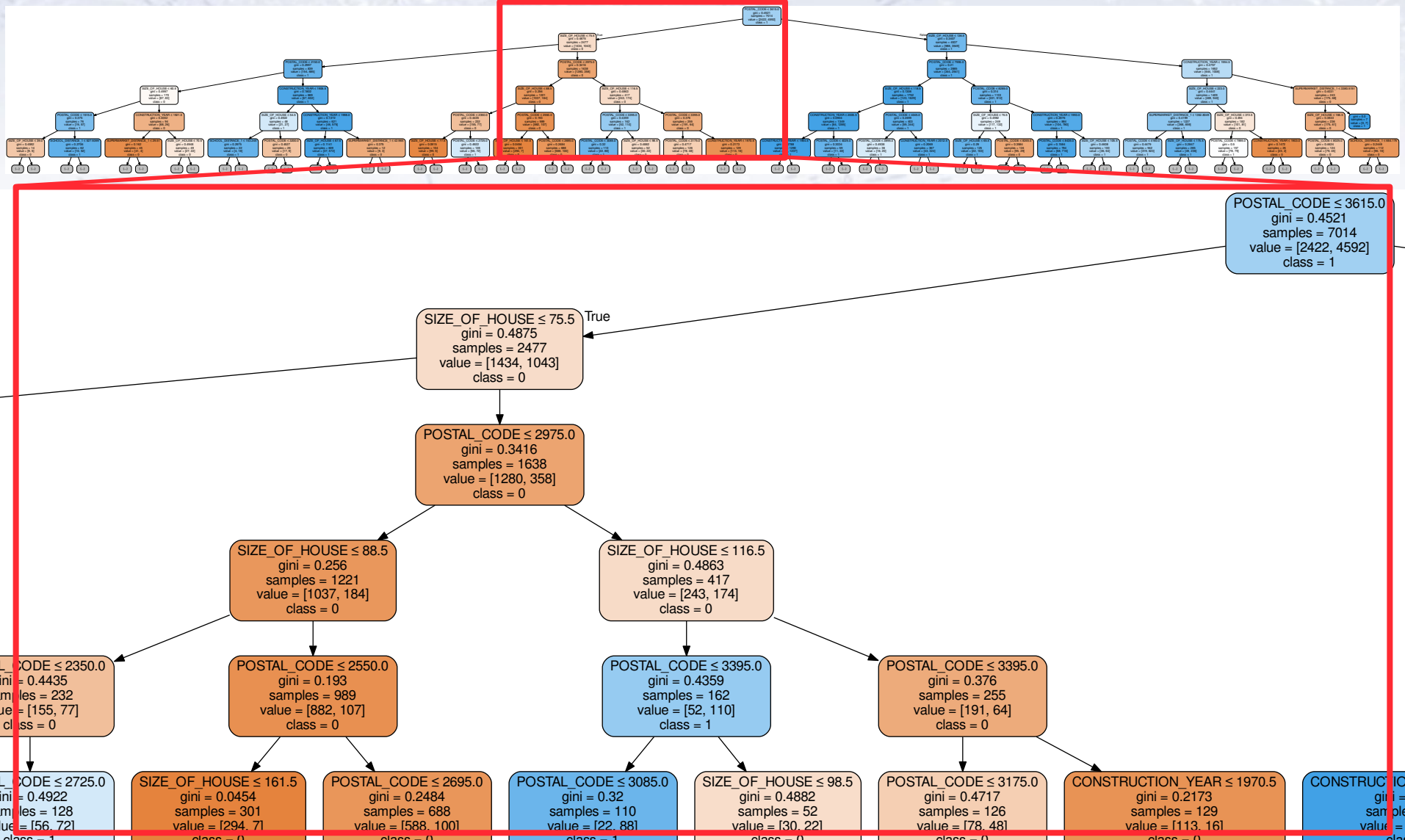
Decision tree for estimating the price in the housing prices data set:





Housing Type decision tree

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



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)

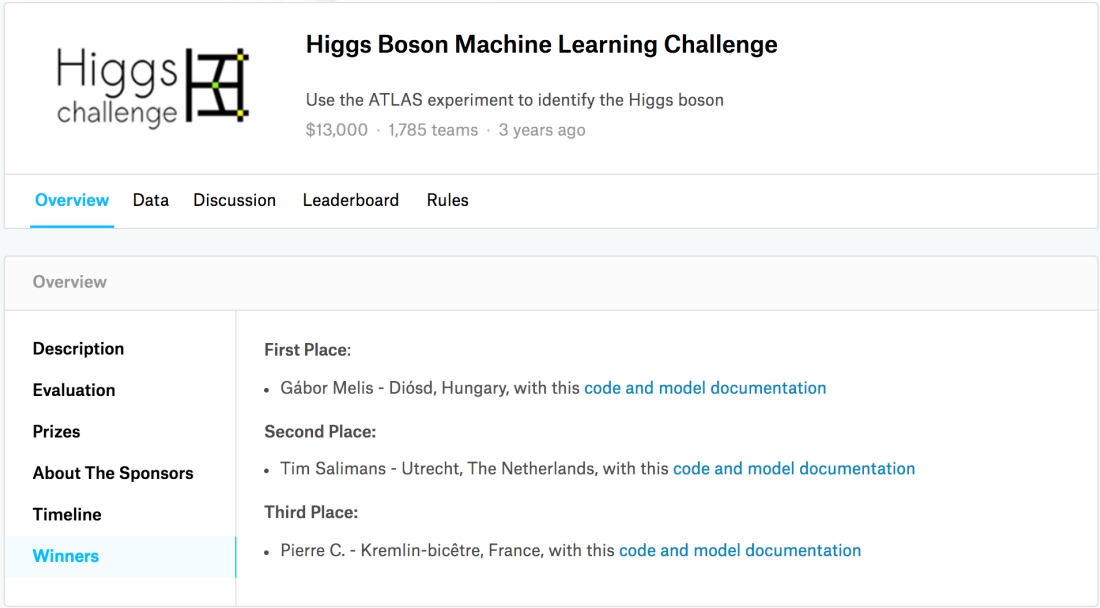
XGBoost history

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.



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:

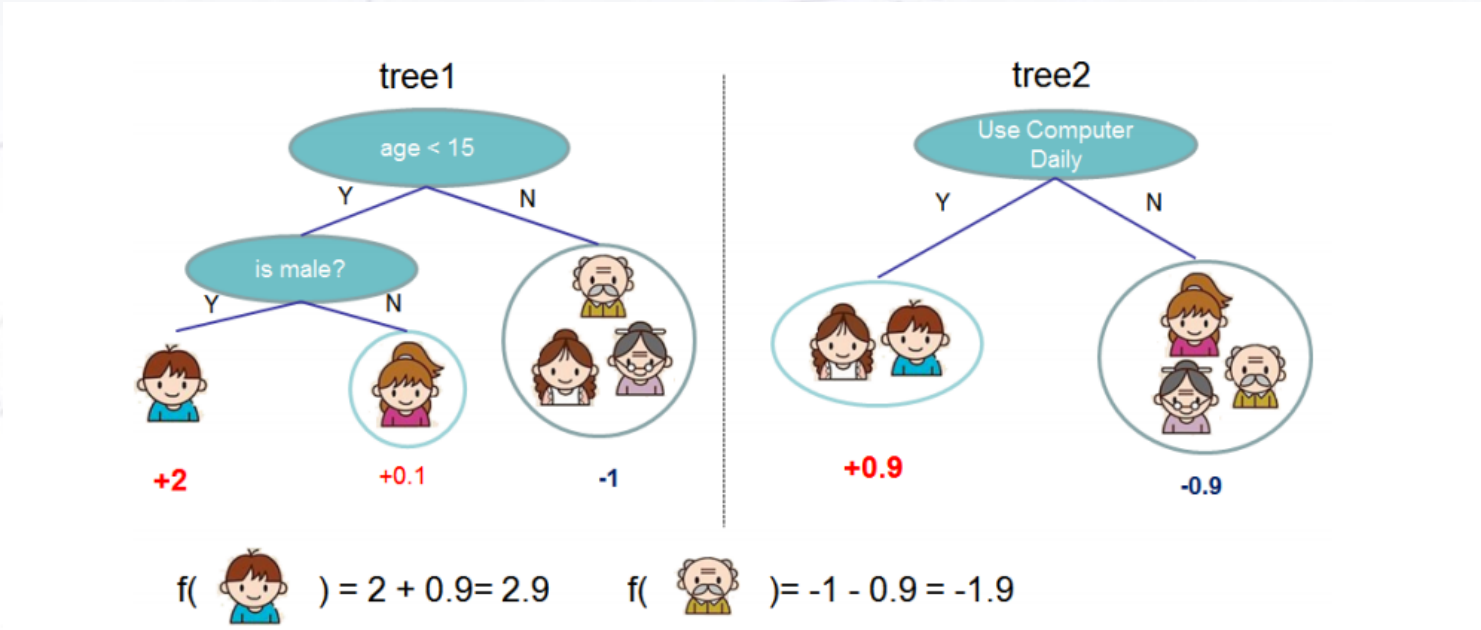


Figure 1: Tree Ensemble Model. The final prediction for a given example is the sum of predictions from each tree.

XGBoost

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 — — — — — — — — —

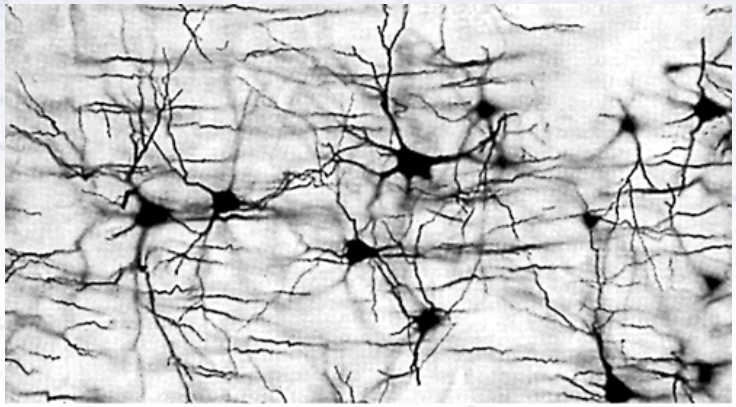
Meanwhile, LightGBM has seen the light of day, and it is even faster…

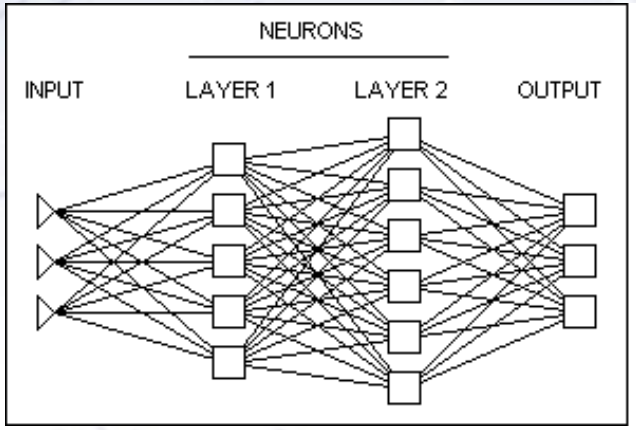
Which algorithm takes the crown: Light GBM vs XGBOOST?

Very good blog with introduction to tree based learning

Neural Network models

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]

A "Linear Network"

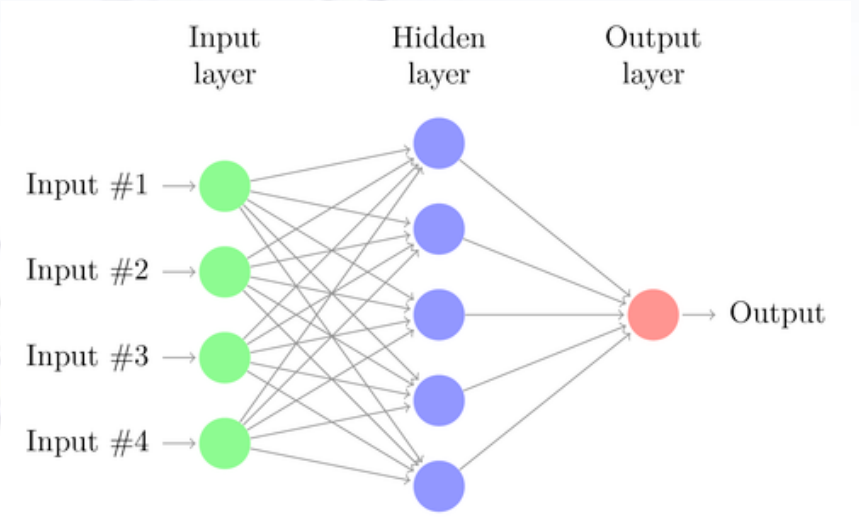
Imagine that we consider a "Linear Network", and use the (simplest) architecture: A single layer (linear) perceptron:

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

As can be see, this is simply **a linear regression in multiple dimensions** or the (linear) Fisher Discriminant.

Well, then we could consider putting in a hidden (linear) layer:

$$tt(x) = t(a_0 + \sum a_i x_i)$$

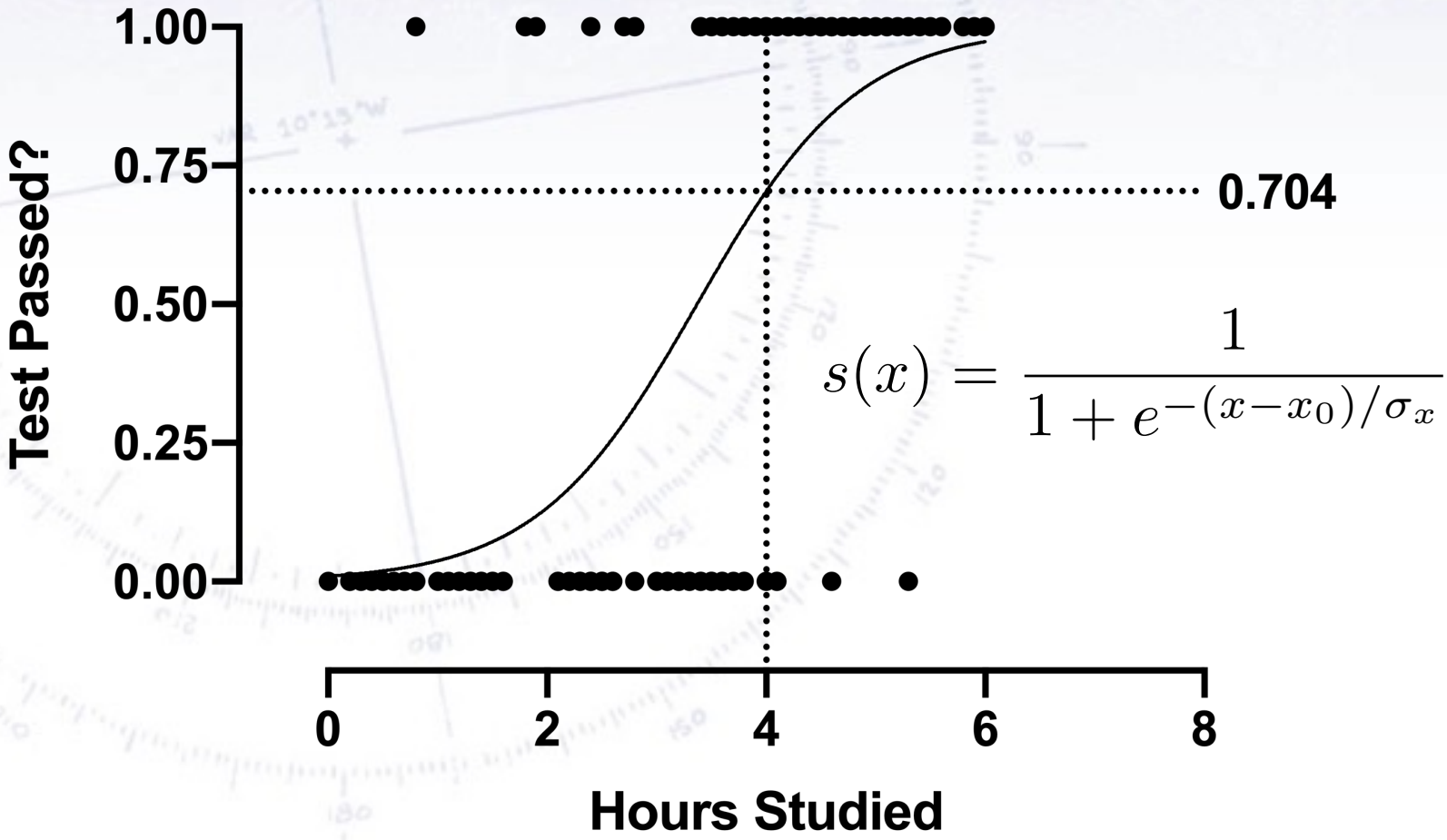


However, this doesn't help anything
as combination of linear functions remain linear. It boils down to the Fisher again!

What we need is something non-linear in the function…

Logistic Regression

Though the word "regression" suggests otherwise, this is in fact a way of doing classification, as the "regression" is usually for a score (s) in the interval [0,1].

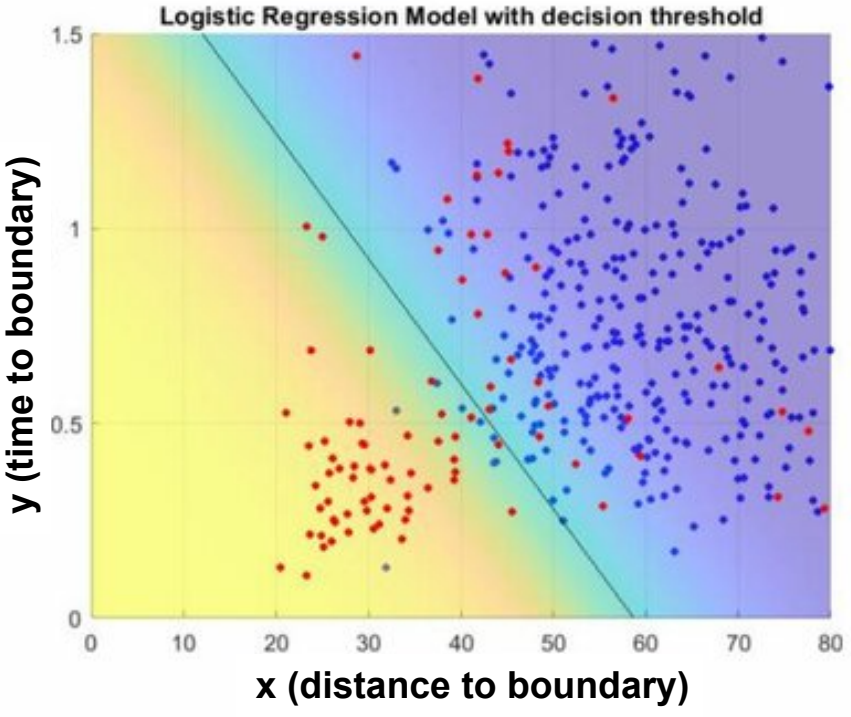


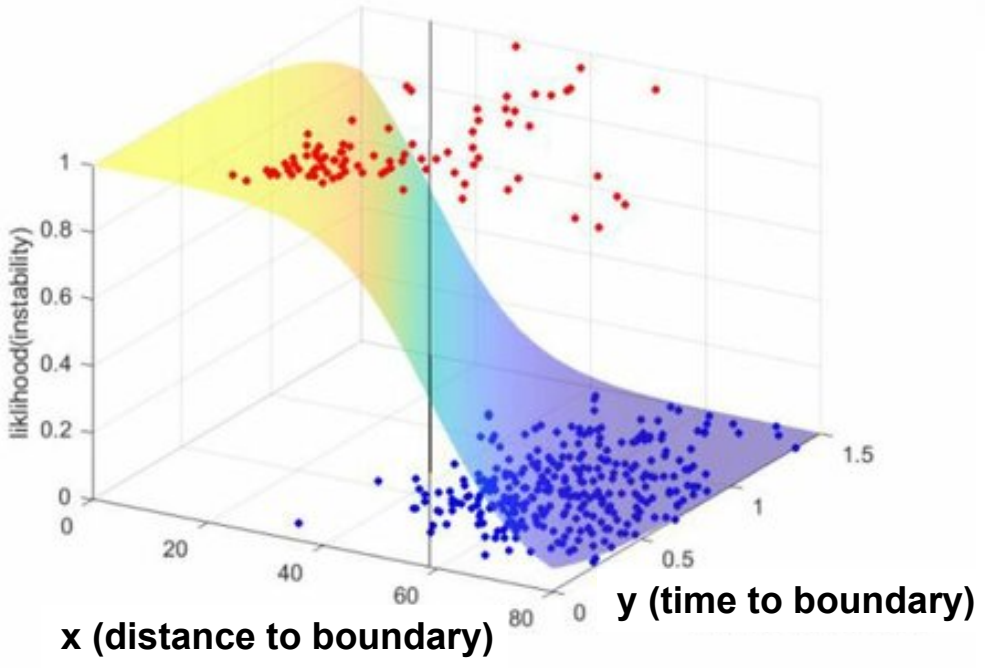
Logistic Regression

Though the word "regression" suggests otherwise, this is in fact a way of doing classification, as the "regression" is usually for a score (s) in the interval [0,1].

The model expands naturally with more parameters:

$$s(x) = \frac{1}{1 + e^{-(x-x_0)/\sigma_x - (y-y_0)/\sigma_y}}$$





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:

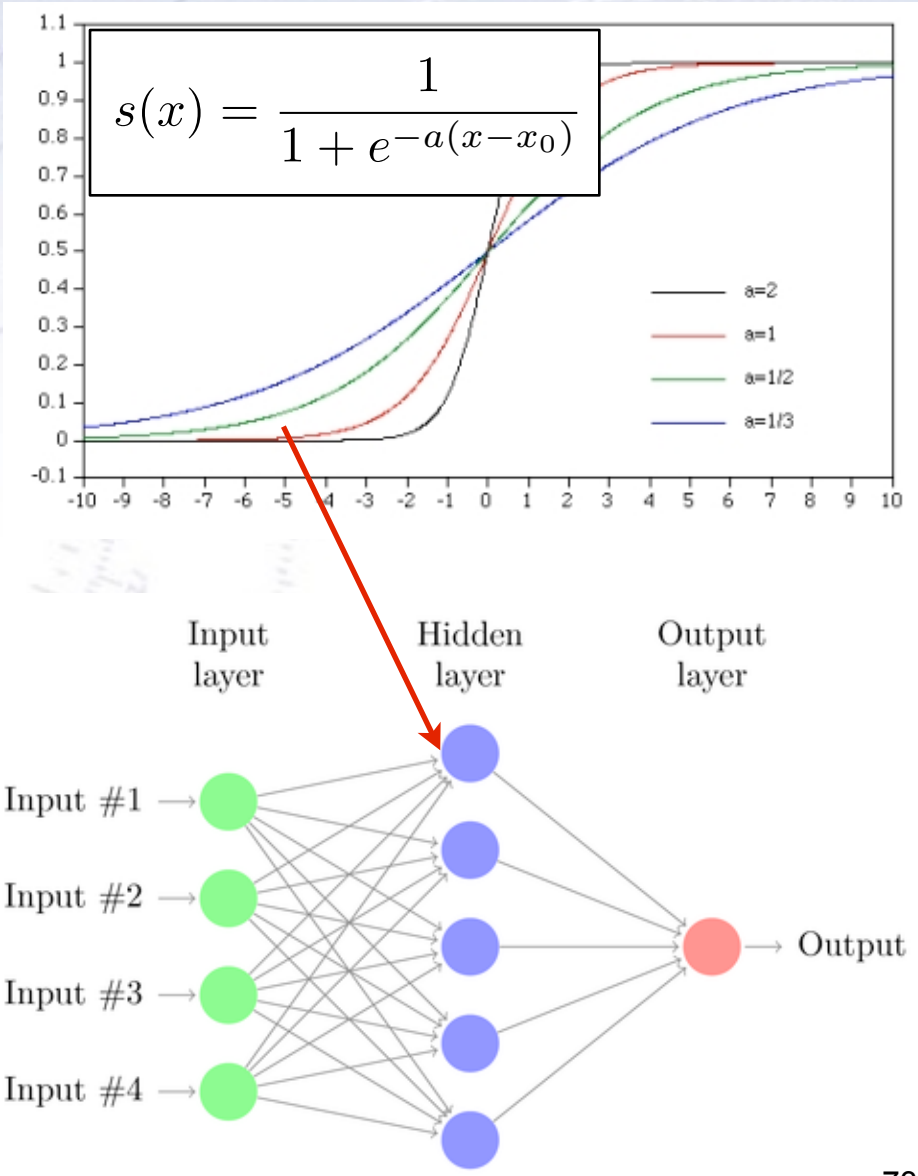
$$t(x) = s\left(a_0 + \sum a_i x_i\right)$$

This can be generalised to a multilayer perceptron (shown right, 1 hidden layer):

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

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

Activation function can be any "sigmoidal" function.

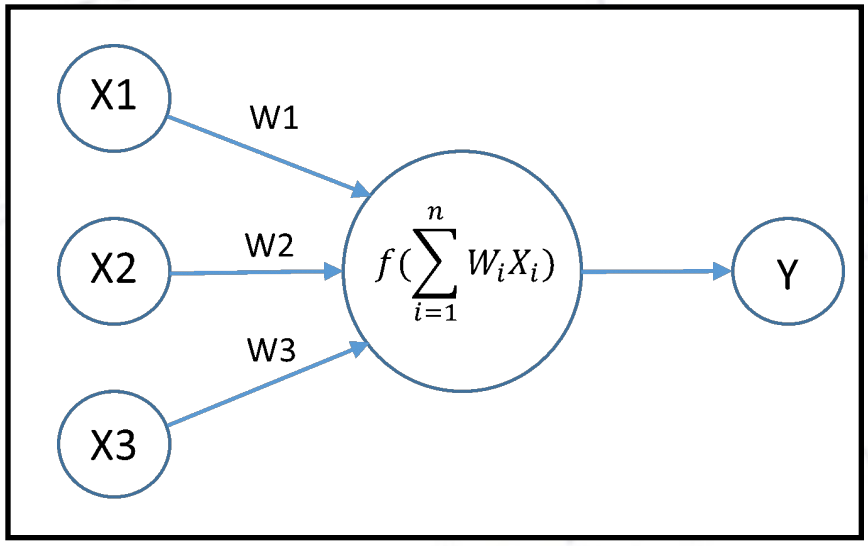


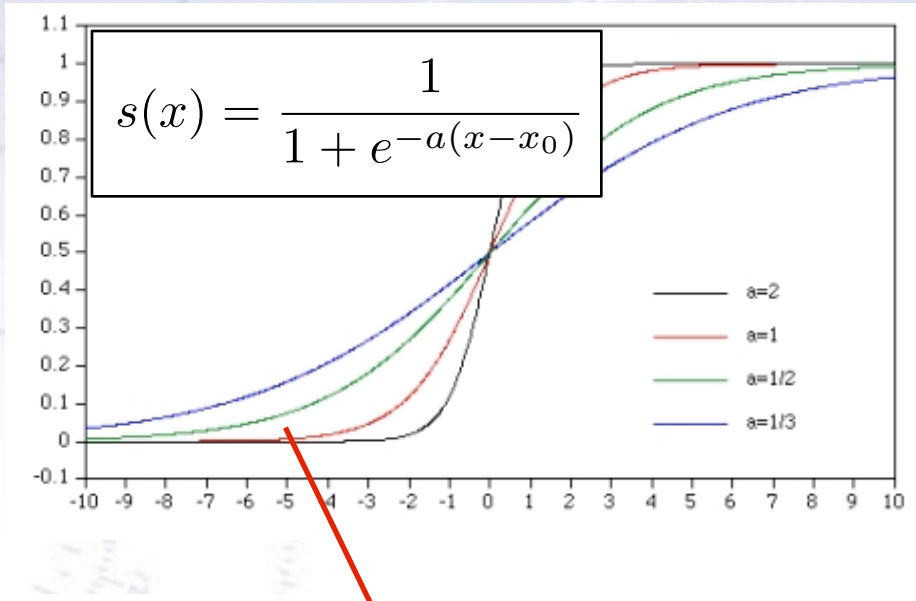
Neural Networks

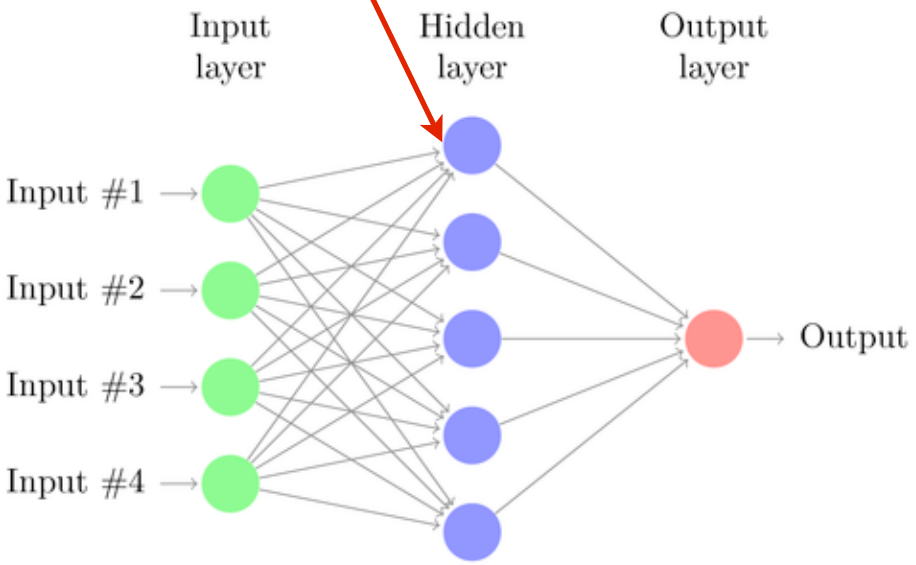
Neural Networks combine the input variables using a "activation" function s(x) to assign, if the variable indicates signal or background.

The simplest is a single layer perceptron:

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







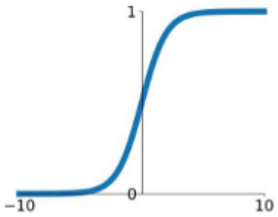
Activation Functions

There are many different activation functions, some of which are shown below. They have different properties, and can be considered a HyperParameter.

Activation Functions

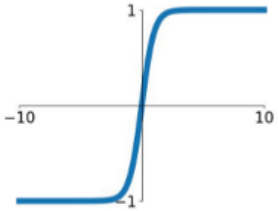
Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



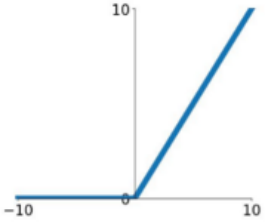
tanh

$$\tanh(x)$$



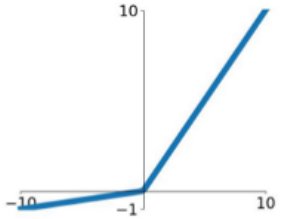
ReLU

$$\max(0, x)$$



Leaky ReLU

$$\max(0.1x, x)$$

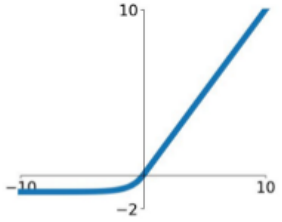


Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

ELU

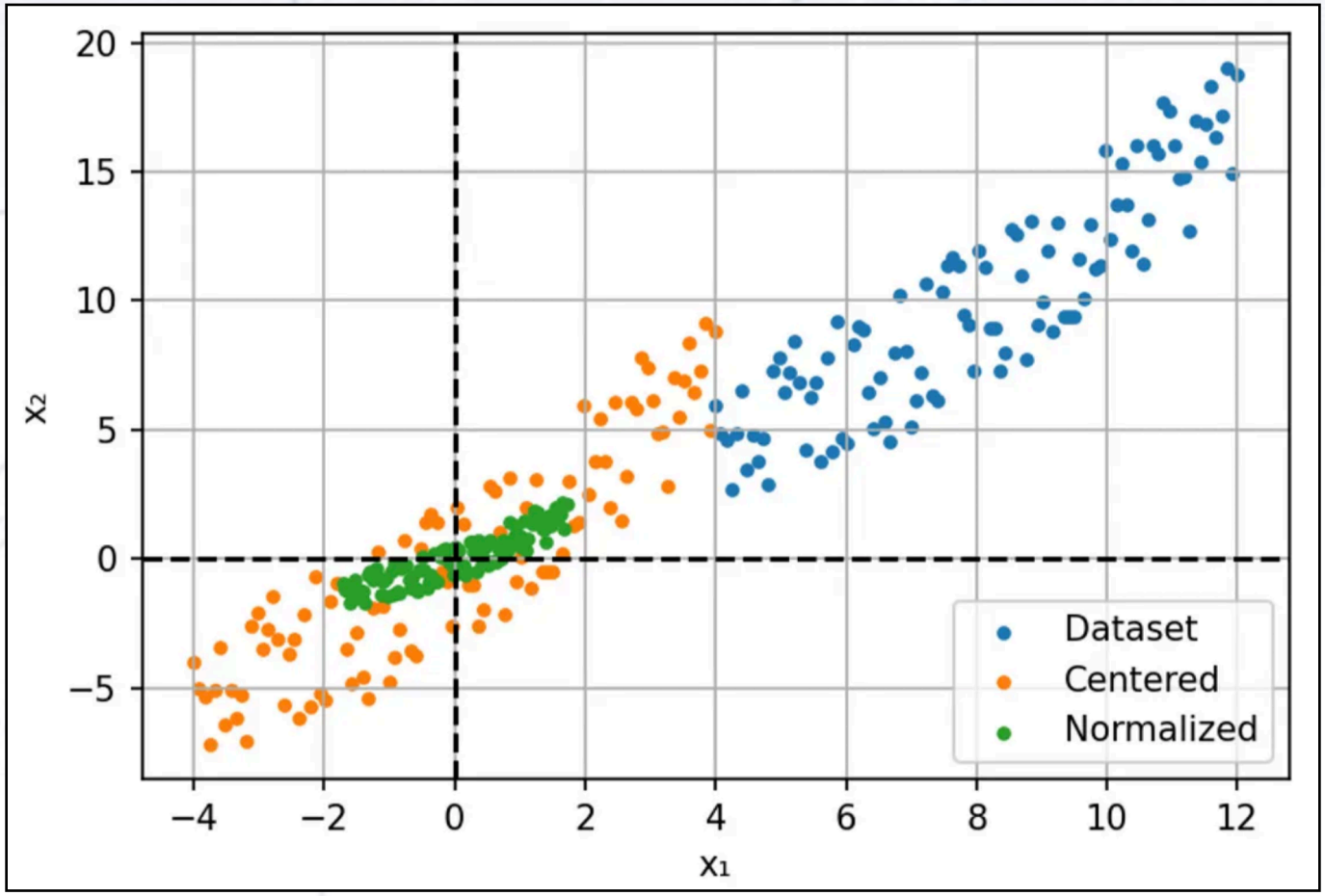
$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



For a more complete list, check: https://en.wikipedia.org/wiki/Activation_function

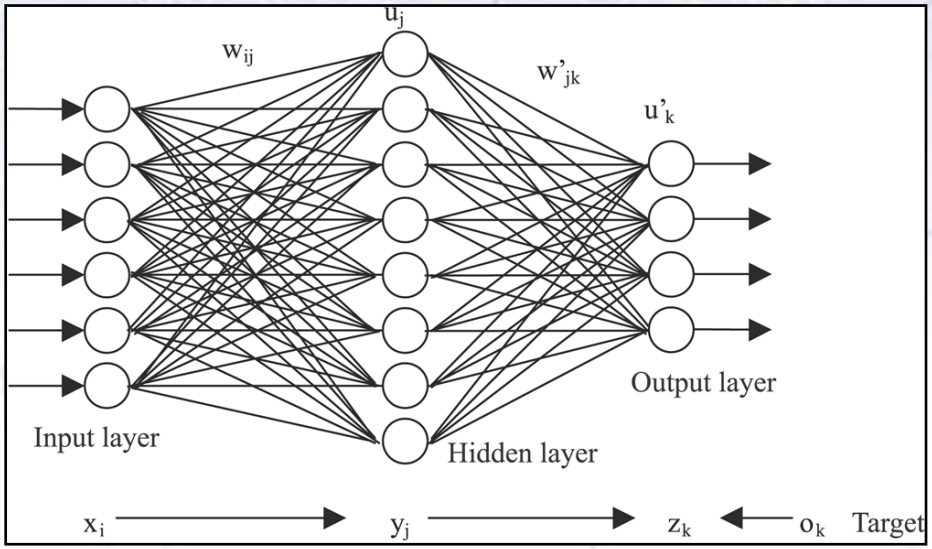
Normalising Inputs

While tree based learning is invariant to (transformations of) distributions, Neural Networks are not. To avoid hard optimisation, vanishing/exploding gradients, and differential learning rates, one should normalise the input:



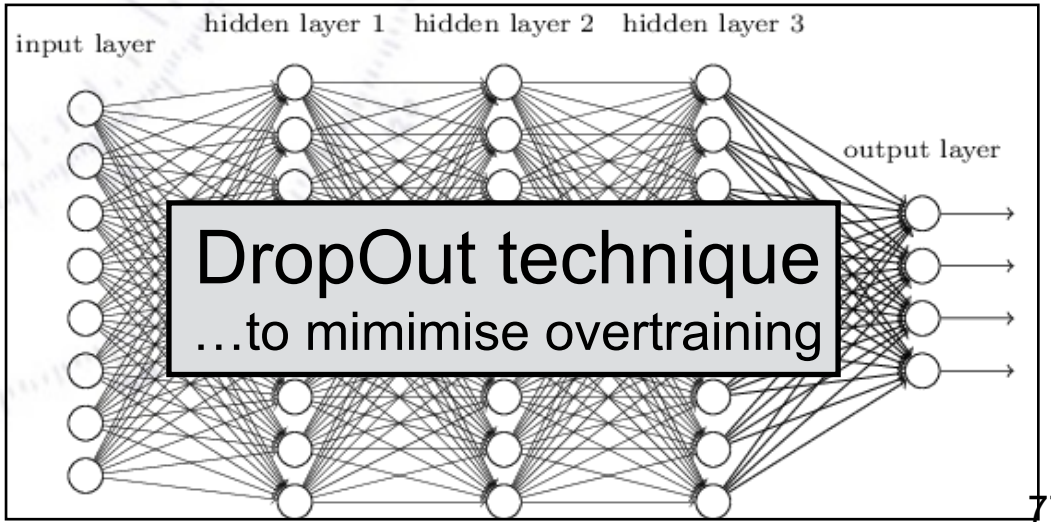
Deep Neural Networks

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.

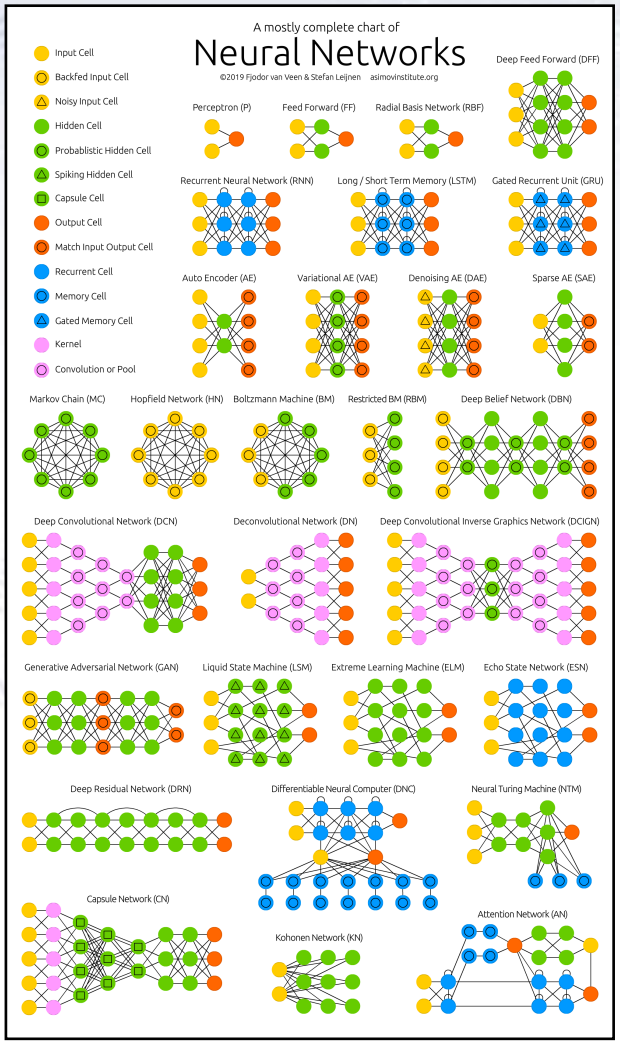
Currently, DNNs can have up to millions of neurons and connections, which compares to about the **brain of a worm**.



The role of NNs

The reason why NNs play such a central role is that they are versatile:

- Recurrent NNs (for time series)
- Convolutional NNs (for images)
- Adversarial NNs (for simulation)
- Graph NNs (for geometric data)
- etc.



Unlike trees, NNs typically make the "foundation" of all the more advanced ML paradigms. However, they are harder to optimise!
This is why trees a great for simpler tasks (i.e. data that typically fits into an excel sheet [2110.01889]), while NNs are typically used for the more advanced.

Have this in mind, when you attack problems with ML - and like any other project or analysis, it is typically good to get a "rough result" fast, and then to refine it from there.

Preprocessing Data

When data is imperfect

So far, we have looked at “perfect” data, i.e. data without any flaws in it. However, real world datasets are hardly ever “perfect”, but contains flaws that makes preprocessing imperative.

Effects may be (non-exhaustive list):

- NaN-values and "Non-values" (i.e. -9999)
- Wild outliers (i.e. values far outside the typical range)
- Shifts in distributions (i.e. part of data having a different mean/width/etc.)
- Mixture of types (i.e. numerical and text, from something humans filled out)

It is also important to consider, if entries are missing…

1. **Randomly** (in which case there should be no bias from omitting) or
2. **Following some pattern** (in which case there could be problems!).

In case of NaN values, we might simply decide to drop the variable column or entry row, requiring that all variables/entries have reasonable values.

Alternatively, we might insert “imputed” values instead, saving statistics.

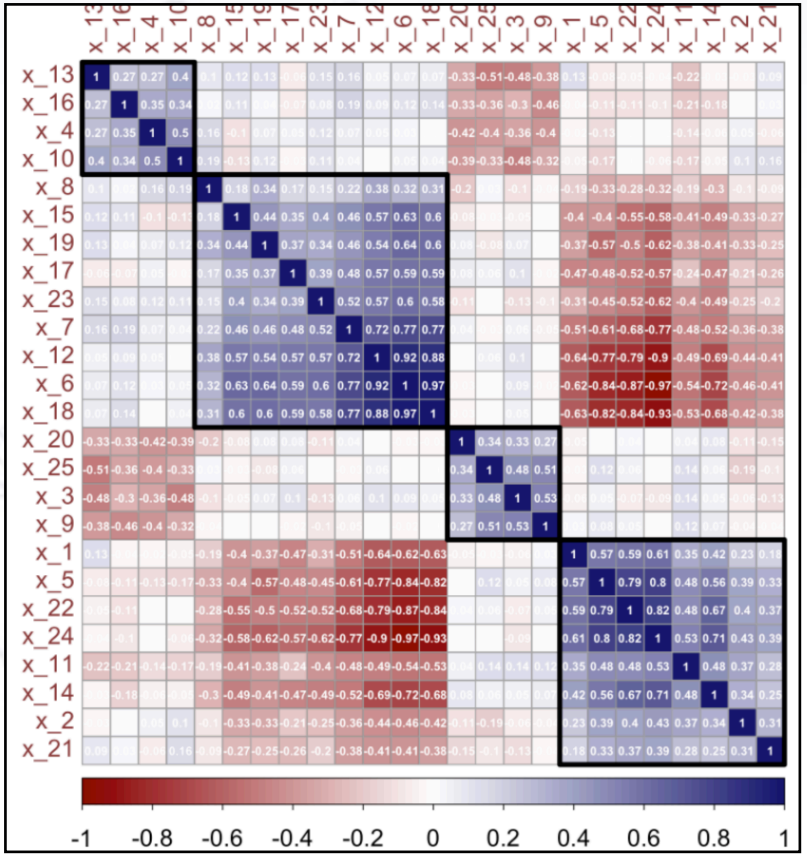
NaN-values tend to correlate

It is often seen, that several variables have the same source, and thus their NaN occurrence might be correlated with each other.

This can be tested by substituting 0's for numerical values and 1's for NaN values. By considering the correlation matrix of these substitute 0/1 values, one gets a pretty clear picture.

Typically, some entries are 100% correlated, as the source of these variables is shared.

Based on this information, one can better decide how to deal with these variables.



Conclusions

No matter what you plan to do with data, my first advice is always:

Print & Plot

This is your first assurance, that you even remotely know what the data contains, and your first guard against nasty surprises.

Also, working with others (from know-nothings to domain experts) you will be required to show the input, and assuring that it is valid and makes sense.

Remember to do so in all your ML work…

Dividing Data

How to “use” your data?

If you train you algorithm on all data, you will not recognise overtrain, nor what the expected performance on new data will be. Thus we divide the data into:

Train Dataset

- Set of data used for **learning** (by the model), that is, to fit the parameters to the machine learning model using **stochastic gradient descent**.

Valid Dataset

- Set of data used to provide an **unbiased evaluation of intermediate models** fitted on the training dataset while tuning model parameters and hyperparameters, and also for selecting input features.

Test Dataset

- Set of data used to provide an **unbiased evaluation of a final model** fitted on the training dataset.

How to do the division?

You can of course do this yourself with your own code, but there are specially prepared functions for the task:

Scikit-Learn method:

```
from sklearn.model_selection import train_test_split
X_train, X_rem, y_train, y_rem = train_test_split(X, y, train_size=0.8)
X_valid, X_test, y_valid, y_test = train_test_split(X_rem, y_rem, test_size=0.5)
```

Fast_ML method:

```
from fast_ml.model_development import train_valid_test_split
X_train, y_train, X_valid, y_valid, X_test, y_test =
train_valid_test_split(df, target = '?', train_size=0.8, valid_size=0.1, test_size=0.1)
```

There are a few important things to remember:

- Always do the data cleaning, selecting, weighting, etc. **before** splitting!
- If there is "more than enough" data, then use **less than the total**.
- If there is "a little too little" data, then use **cross validation (next)**.

k-fold 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.

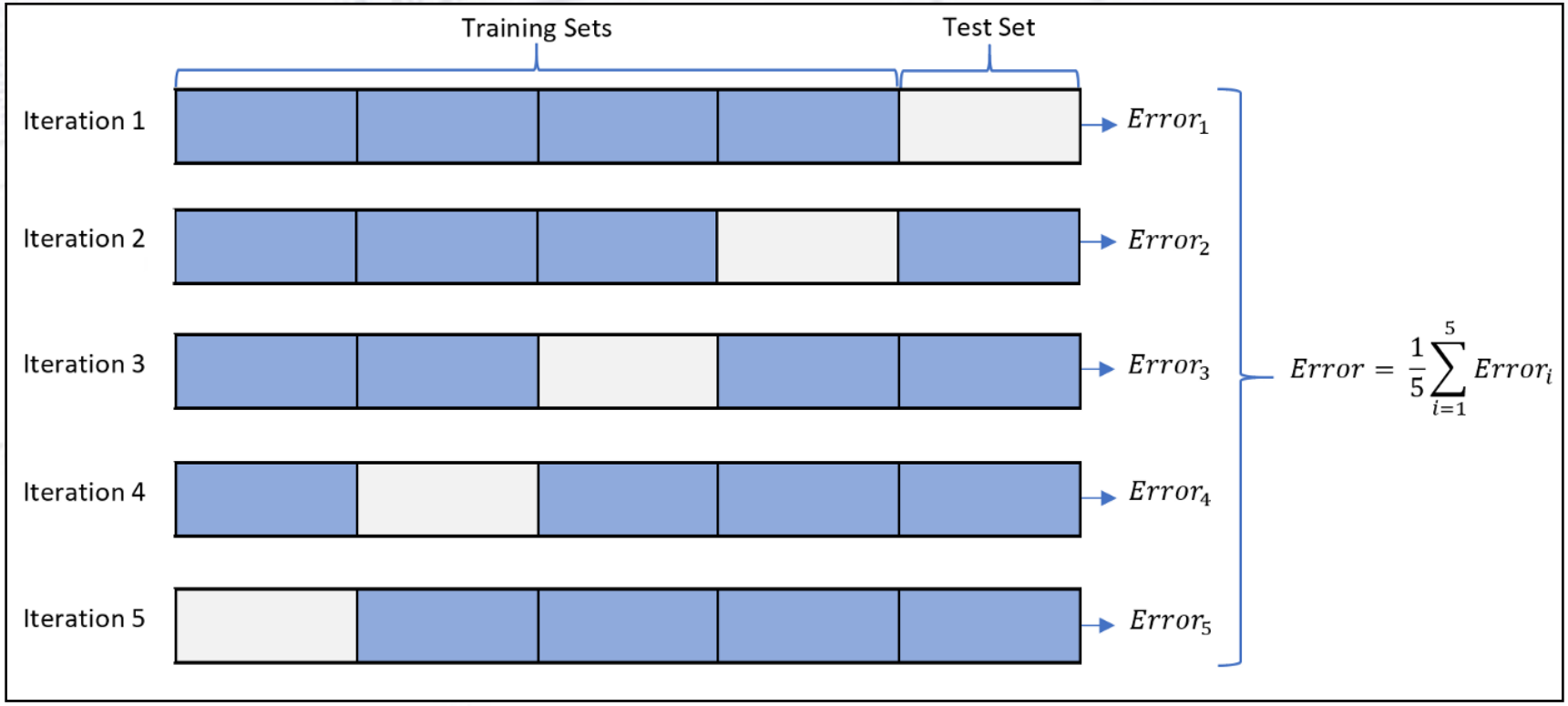
This is then repeated for each fold… CPU-intensive, but easily parallelisable and smart especially for small data samples.

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

Getting an uncertainty estimate

The k-fold cross validation (CV) does not only allow you to train on almost all (1 - (1/k)) and test on all the data, but also has a two additional advantages:

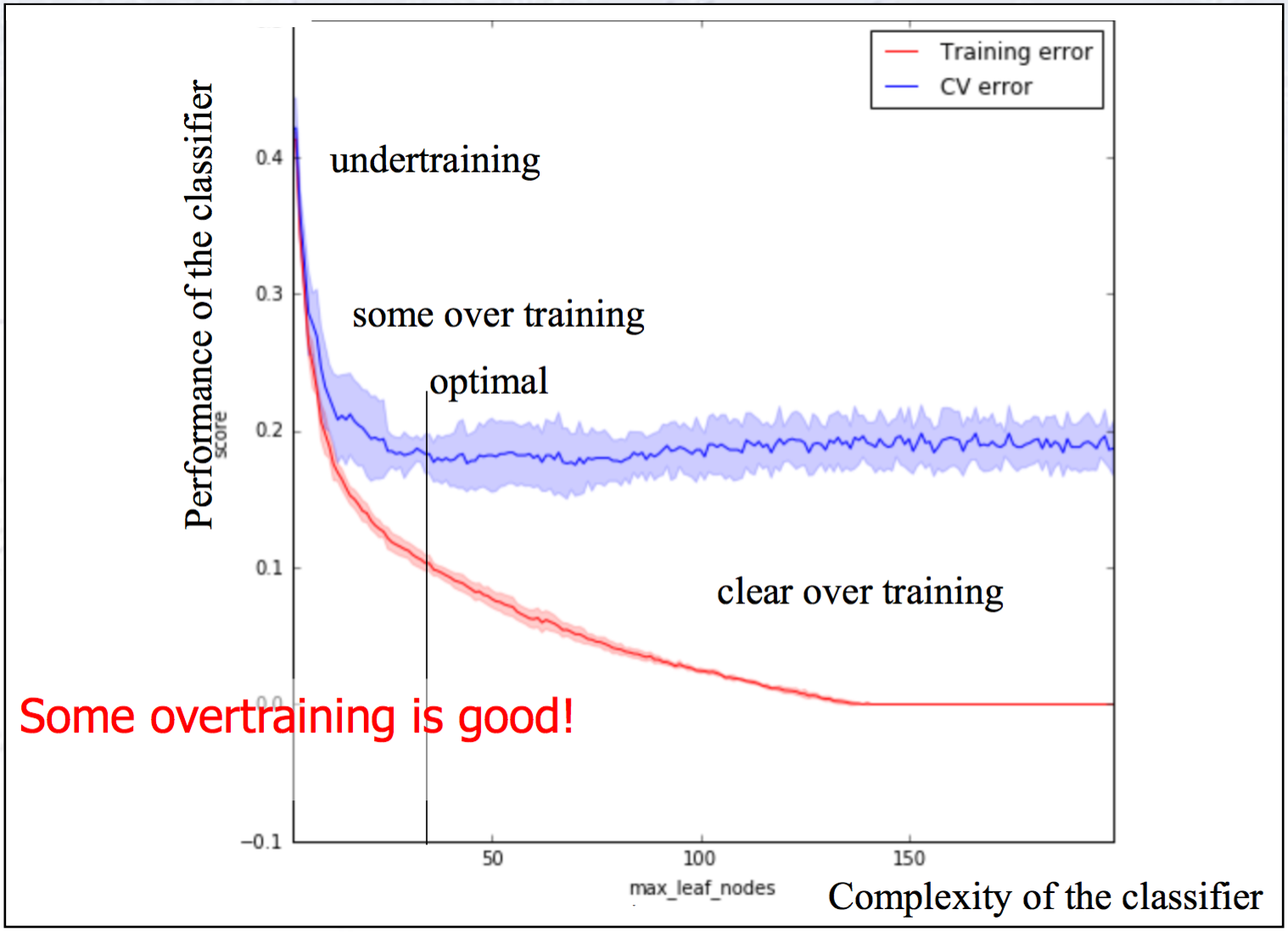
- If you consider the performance ("Error" below) on each fold, then you can also calculate the uncertainty on the performance.
- Since you can test on all data, the uncertainty on the loss estimate goes down.



Train, Validation & Test

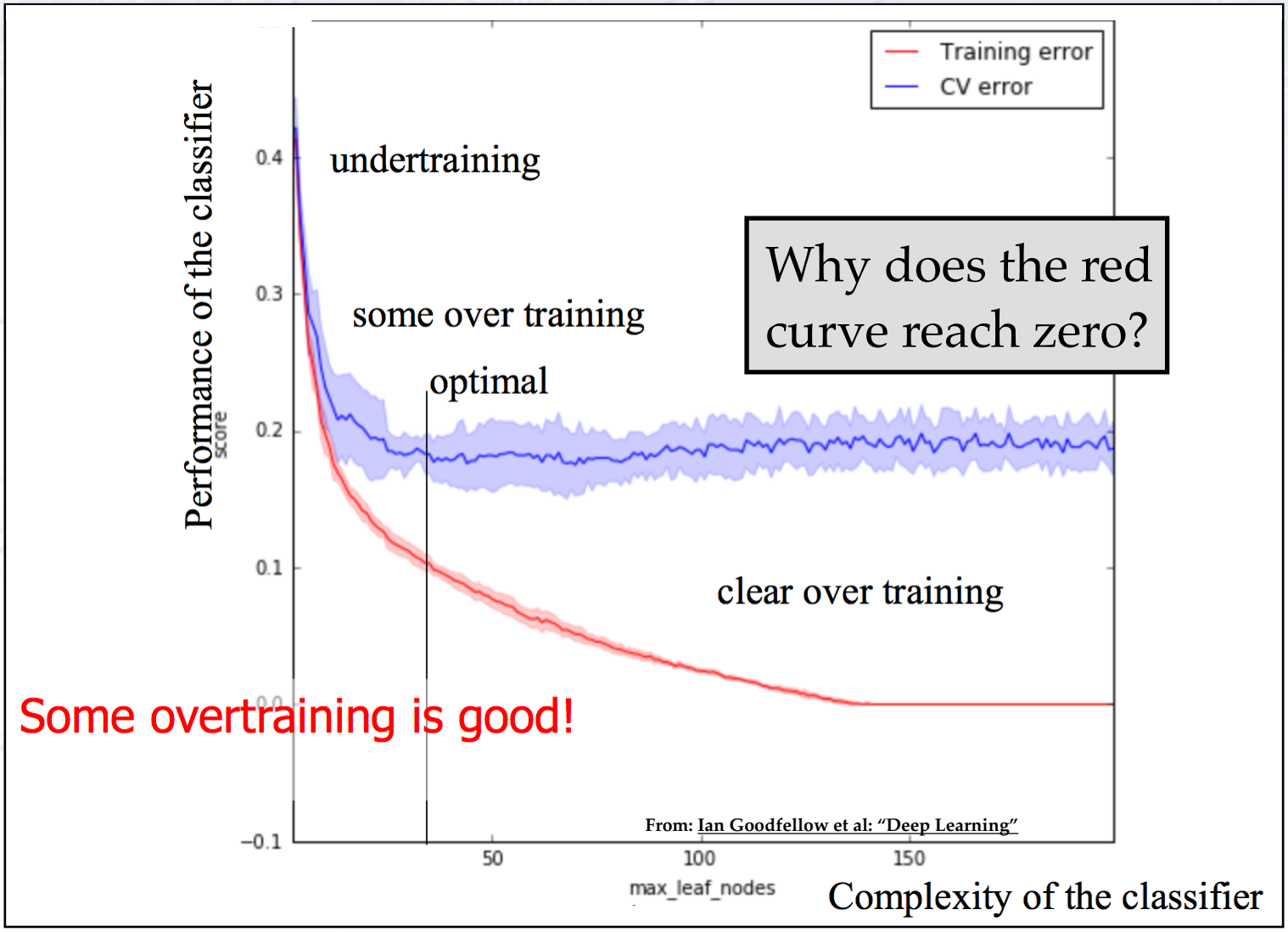
Real overtraining

The "real" limit of overtraining, is when the (Cross) Validation (CV) error starts to grow!



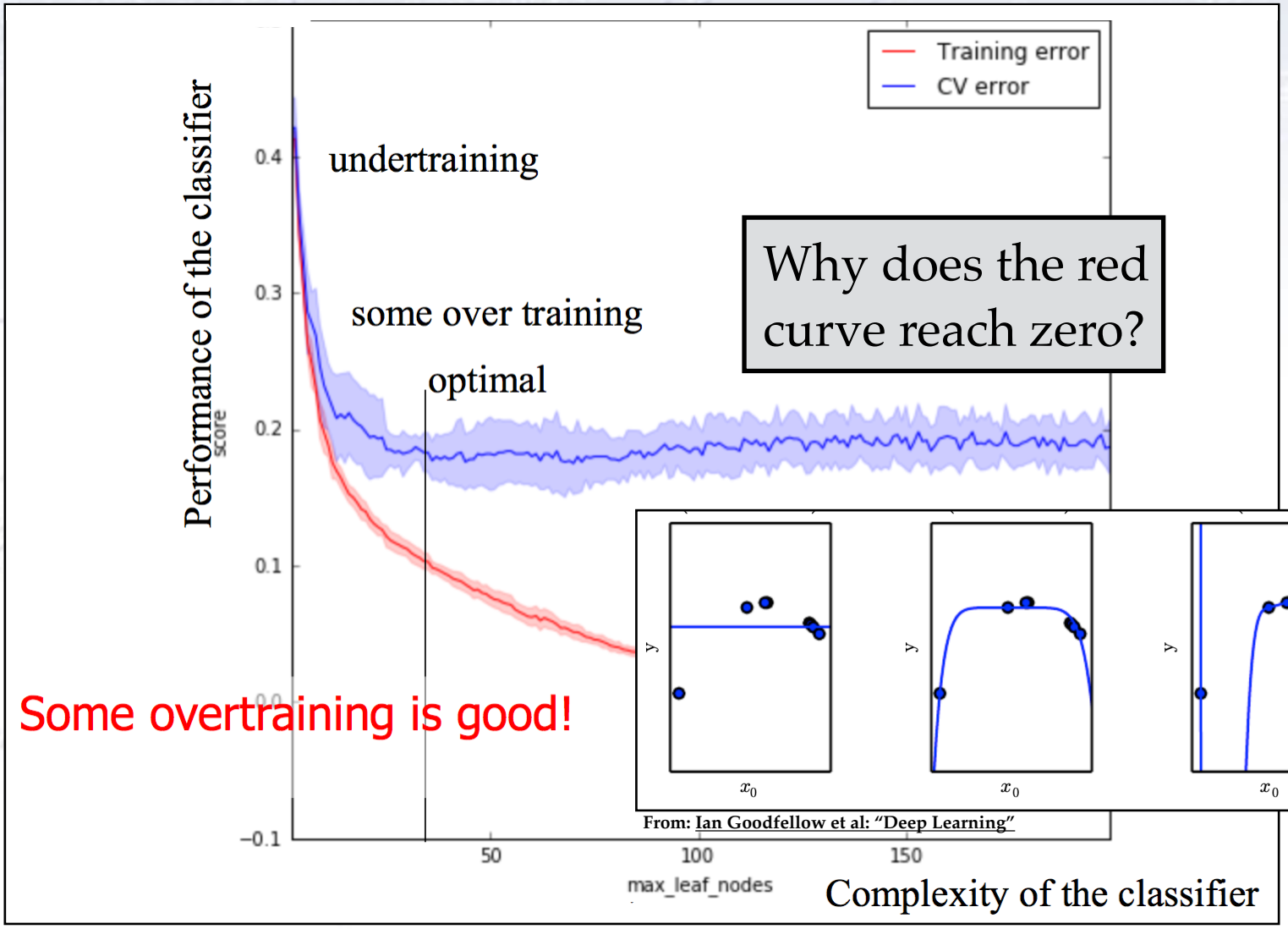
Real overtraining

The "real" limit of overtraining, is when the (Cross) Validation (CV) error starts to grow!



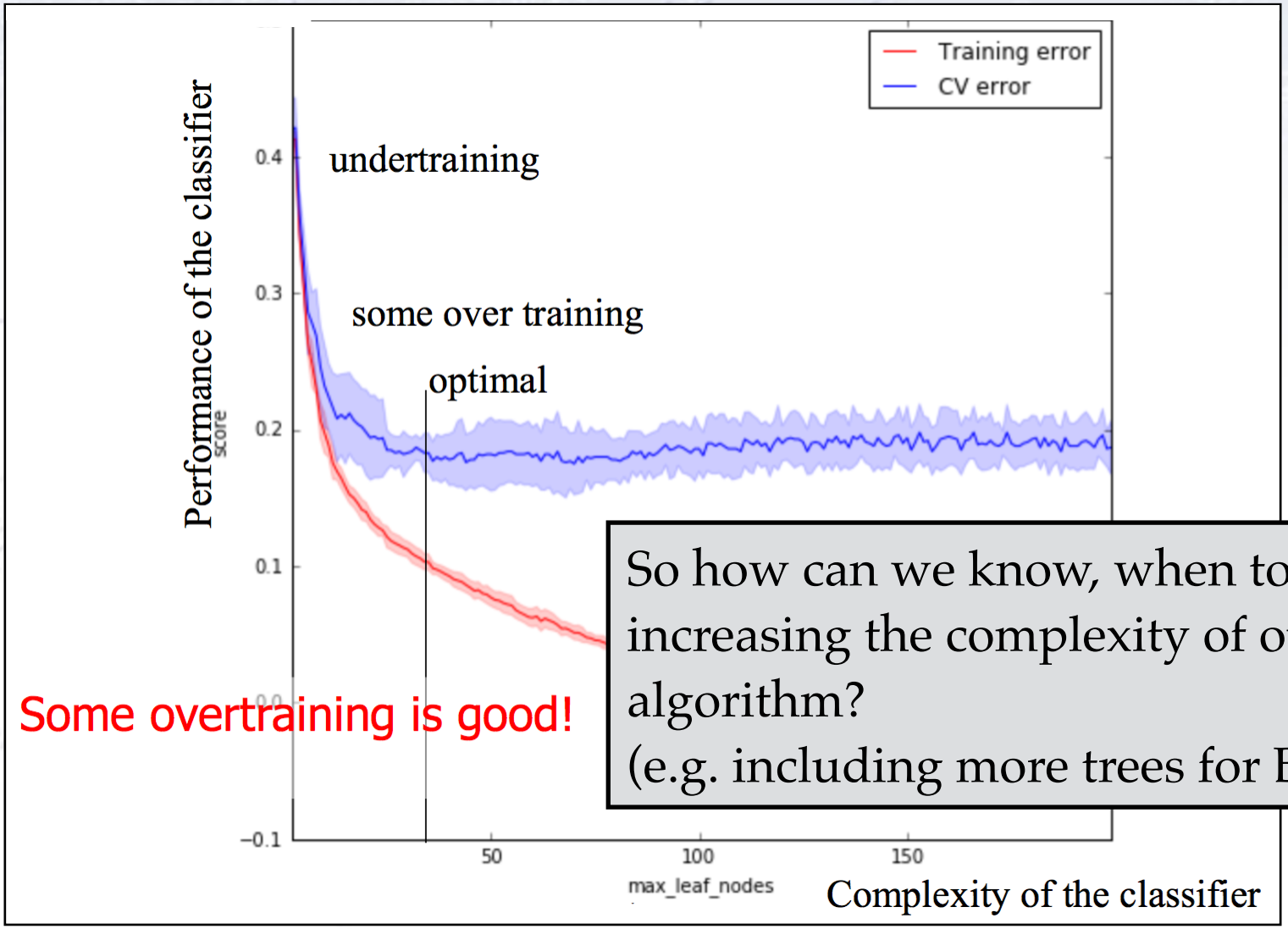
Real overtraining

The "real" limit of overtraining, is when the (Cross) Validation (CV) error starts to grow!



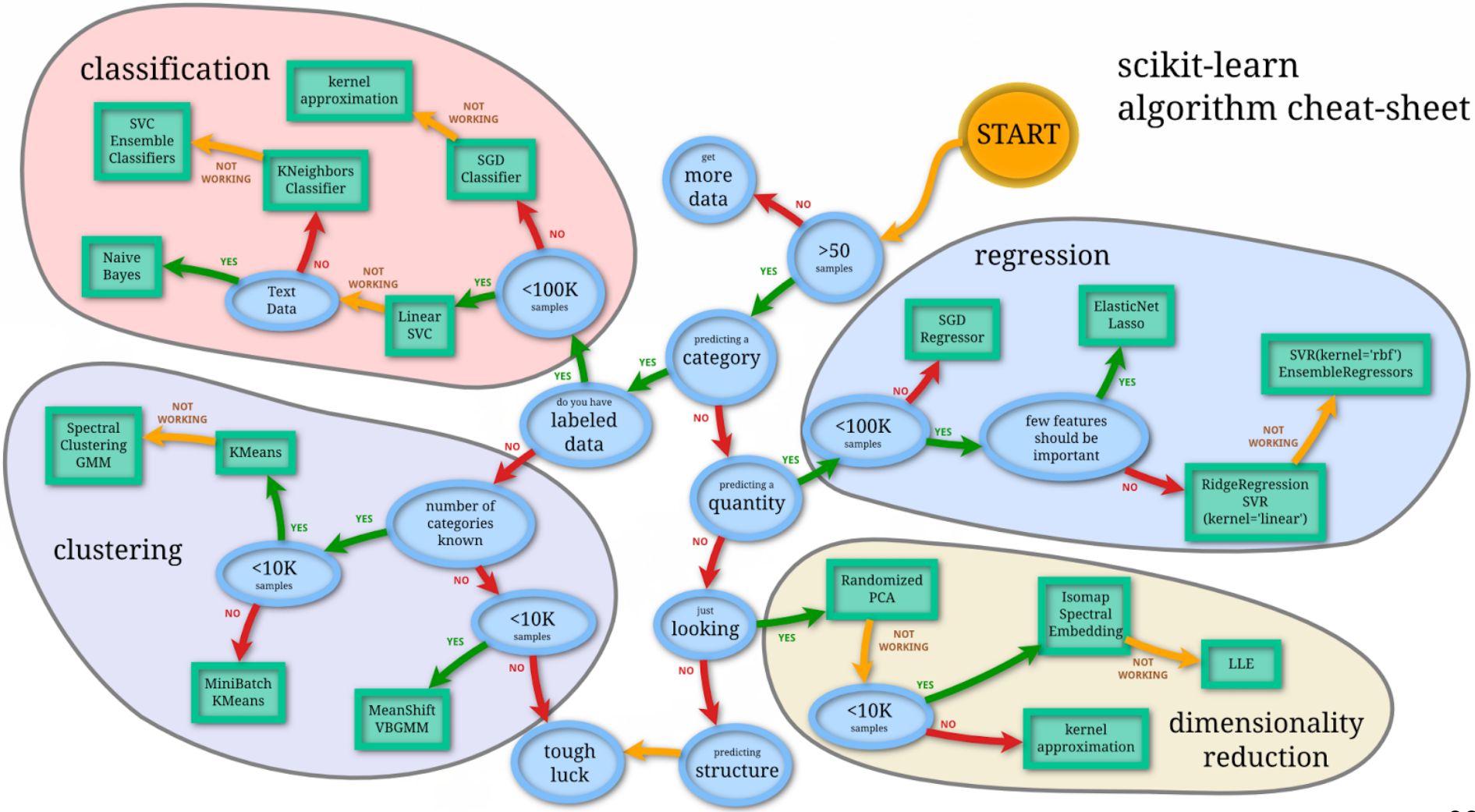
Real overtraining

The "real" limit of overtraining, is when the (Cross) Validation (CV) error starts to grow!



Which method to use?

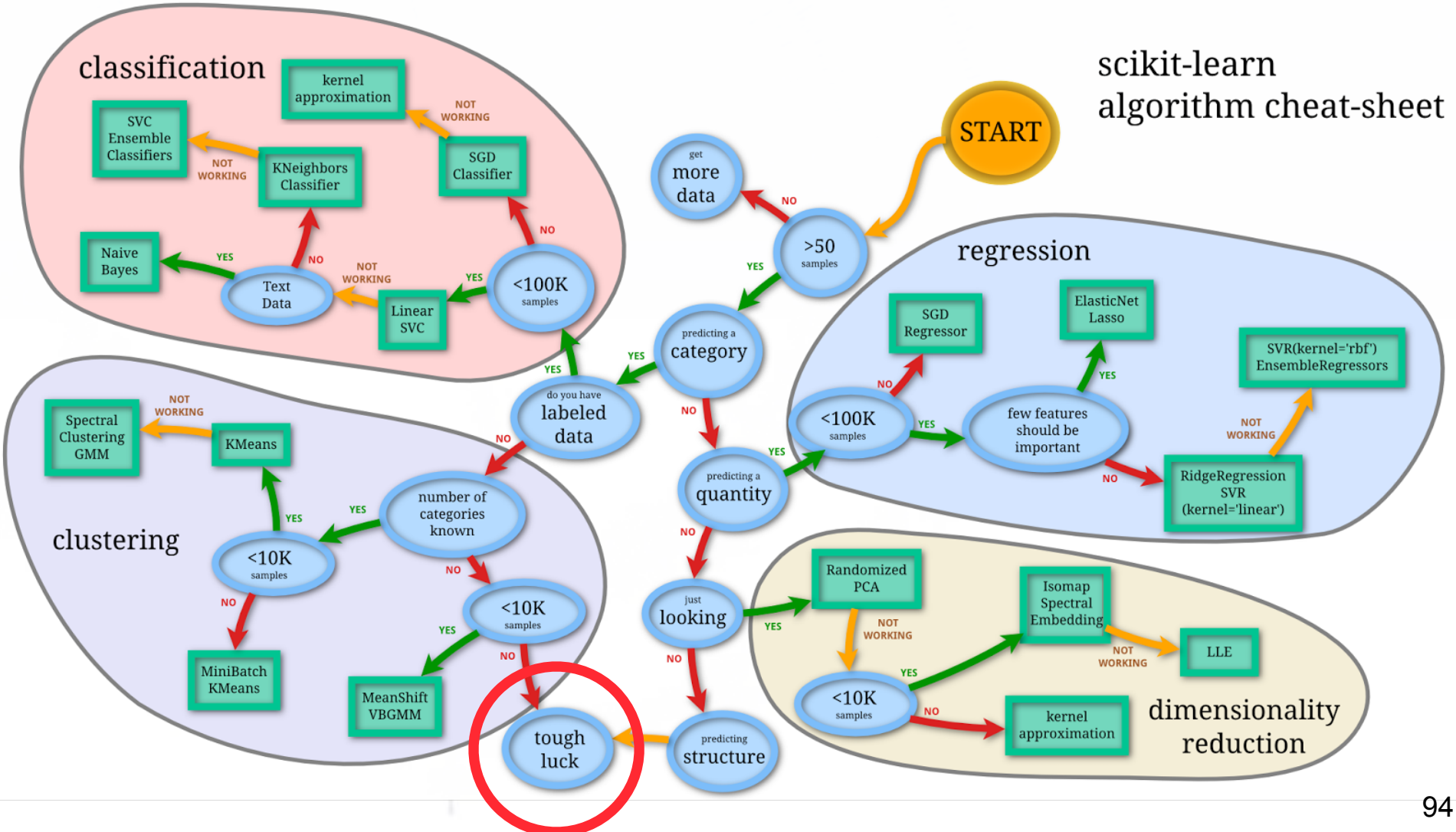
There is no good/simple answer to this, though people have tried, e.g.:



Note: Old Scikit-Learn overview!

Which method to use?

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

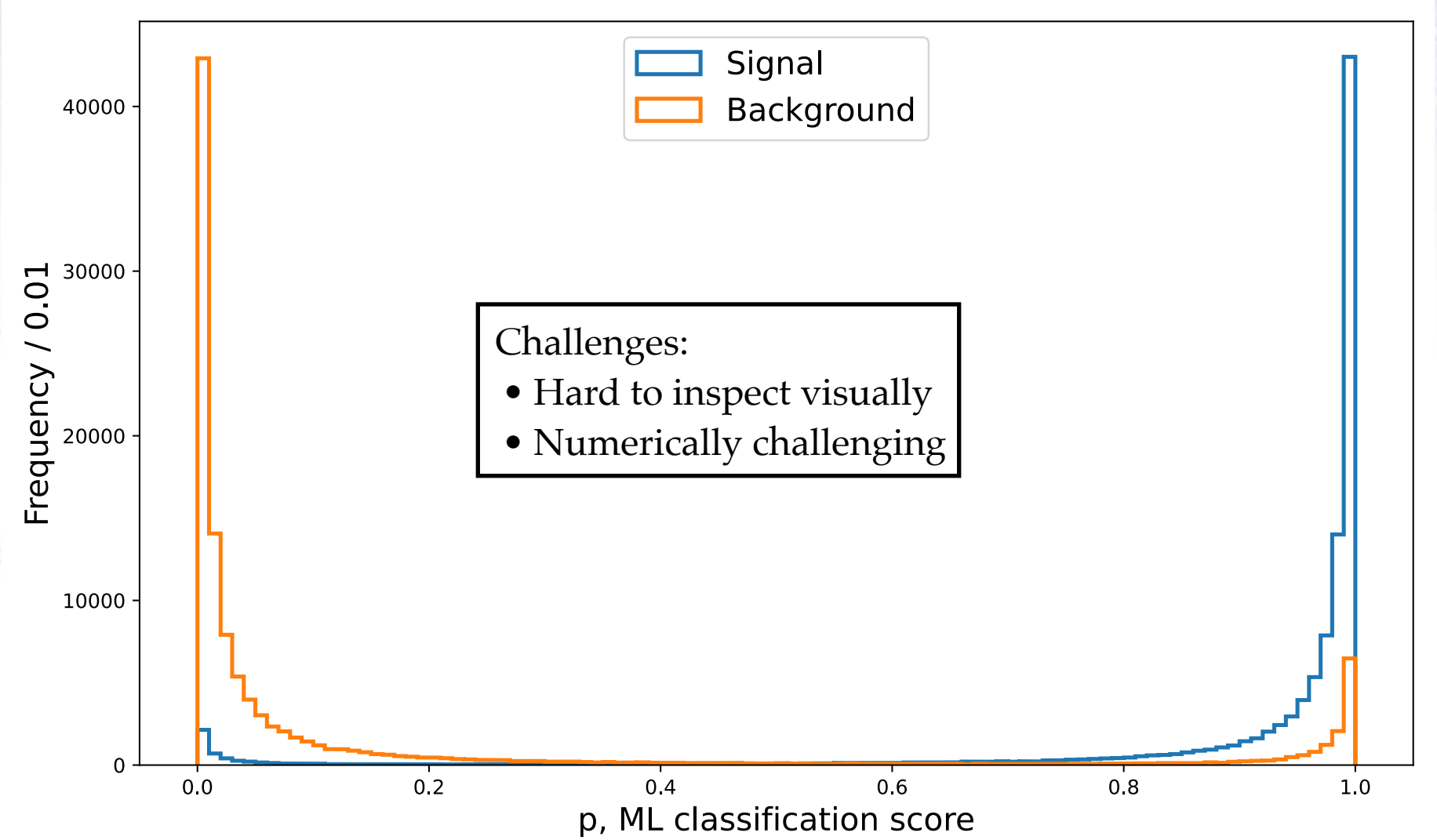


Bonus Slides

The ML output

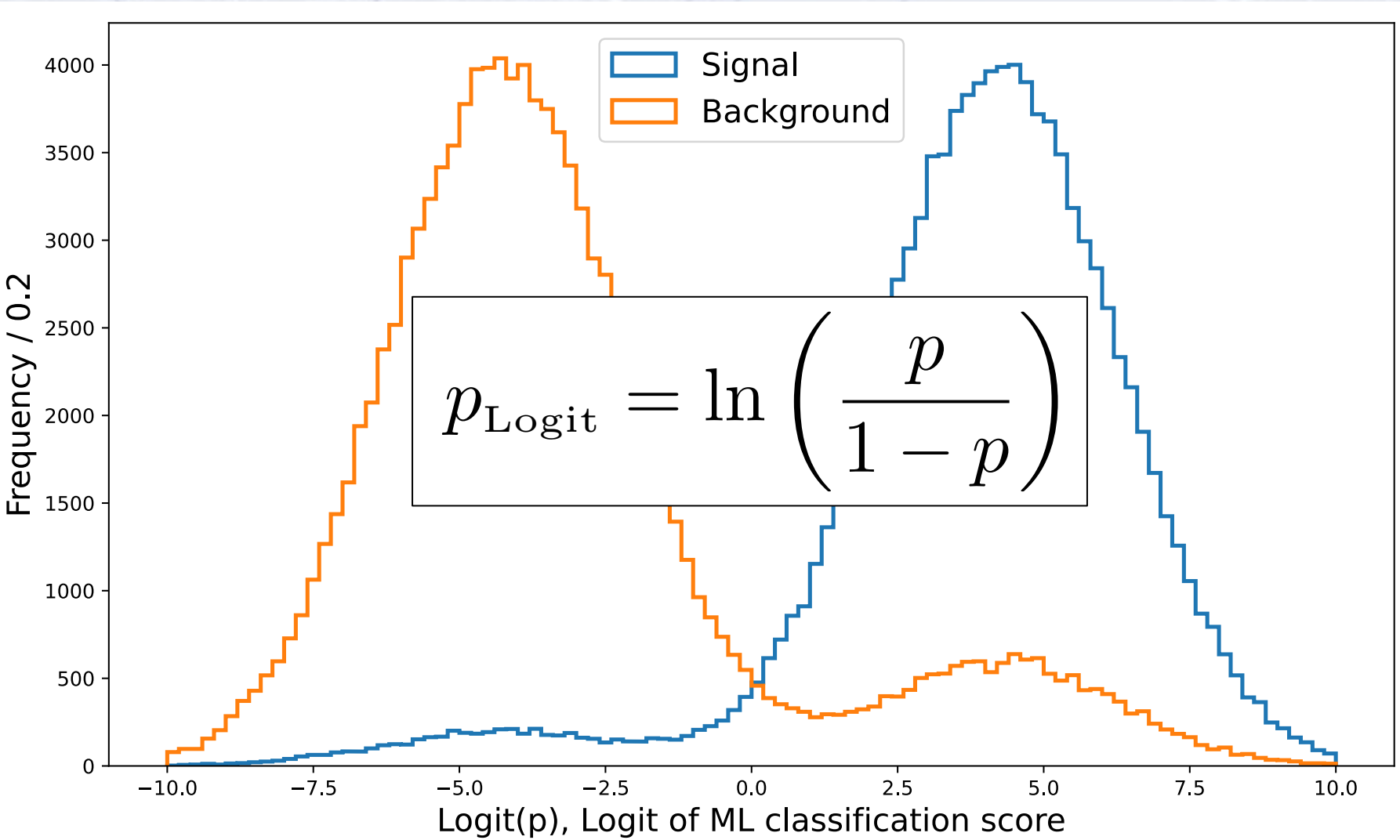
Typical ML Distribution

An ML score distribution from binary classification typically looks as follows:



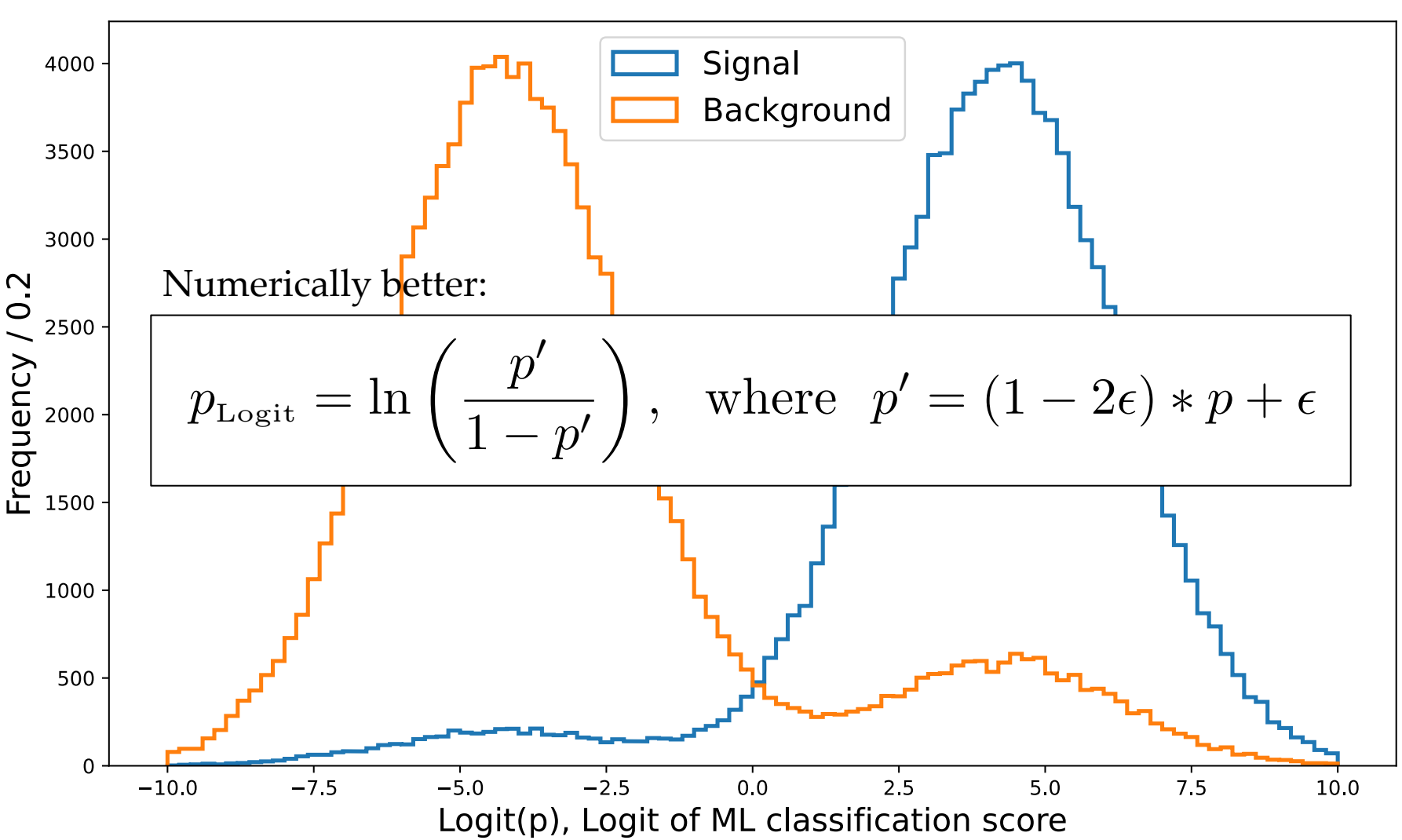
Logit transformation

Once logit transformed, it takes on a "nicer" (and numerically friendly) shape:



Logit transformation

Once logit transformed, it takes on a "nicer" (and numerically friendly) shape:



When to apply ML?

When to use ML?

Using ML in an analysis is usually a (favorable) trade-off between:

- **Higher statistics → Lower statistical error**
  (better efficiency, sharper peaks… *unless the cases are simple!*)
- **Larger data-MC differences → Higher systematic errors**
  (more inputs, non-linearities… *unless there are good control channels!*)

So consider the table of uncertainties from a previous analysis (or estimate these with a colleague), and **ask yourself which of the two are dominant?**

Jet multiplicity	Measured cross section $\pm$ (stat.) $\pm$ (syst.) $\pm$ (lumi.) [pb]			
	$Z \to \ell\ell$			
≥ 0 jets	$740 \pm$	$1 \pm$	$23 \pm$	16
≥ 1 jets	$116.0 \pm$	$0.3 \pm$	$9.7 \pm$	2.5
≥ 2 jets	$27.0 \pm$	$0.1 \pm$	$2.8 \pm$	0.6
≥ 3 jets	$6.20 \pm$	$0.04 \pm$	$0.82 \pm$	0.14
≥ 4 jets	$1.48 \pm$	$0.02 \pm$	$0.23 \pm$	0.04
≥ 5 jets	$0.36 \pm$	$0.01 \pm$	$0.07 \pm$	0.01
≥ 6 jets	$0.079 \pm$	$0.004 \pm$	$0.018 \pm$	0.002
≥ 7 jets	$0.0178 \pm$	$0.0019 \pm$	$0.0049 \pm$	0.0005

Search for Higgs boson decays to a Z boson and a photon in proton-proton collisions at $\sqrt{s} = 13$ TeV

CMS Collaboration

Summary

A search is performed for a standard model (SM) Higgs boson decaying into a lepton pair (e^+e^- or $\mu^+\mu^-$) and a photon with $m_{\ell^+\ell^-} > 50$ GeV. The analysis is performed using a sample of proton-proton (pp) collision data at $\sqrt{s} = 13$ TeV, corresponding to an integrated luminosity of 138 fb^{-1}. The main contribution to this final state is from Higgs boson decays to a Z boson and a photon ($H \to Z\gamma \to \ell^+\ell^-\gamma$). The best fit value of the signal strength $\hat{\mu}$ for $m_H = 125.38$ GeV is $\hat{\mu} = 2.4^{+0.8}_{-0.9}$ (stat) $^{+0.3}_{-0.2}$ (syst) $= 2.4 \pm 0.9$. This measurement corresponds to $\sigma(pp \to H)\mathcal{B}(H \to Z\gamma) = 0.21 \pm 0.08$ pb. The measured value is 1.6 standard deviations higher than the SM prediction. The observed (expected) local significance is 2.7 (1.2) standard deviations, where the expected significance is determined for the SM hypothesis. The observed (expected) upper limit at 95% confidence level on μ is 4.1 (1.8). In addition, a combined fit with the $H \to \gamma\gamma$ analysis of the same data set [18] is performed to measure the ratio $\mathcal{B}(H \to Z\gamma)/\mathcal{B}(H \to \gamma\gamma) = 1.5^{+0.7}_{-0.6}$, which is consistent with the ratio of 0.69 ± 0.04 predicted by the SM at the 1.5 standard deviation level.

With this in mind, consider if it is worthwhile to apply Machine Learning.

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.

Links

Links to many ML resources:

https://www.nbi.dk/~petersen/Teaching/ML2025/MLlinks.html

Links to great online NN toy:

https://playground.tensorflow.org/

Link to super online CNN toy:

https://adamharley.com/nn_vis/cnn/2d.html