

Lectures on Machine Learning



Tommaso Dorigo, INFN-Padova
Braga, March 25-27, 2019

Two words about your lecturer

I work for INFN – Istituto Nazionale di Fisica Nucleare, in Padova

Am a member of the CMS collaboration at CERN (2001-)

- formerly (1992-2012) also a member of CDF @ Tevatron

Long-time interest in statistics; member of CMS Statistics Committee, 2009-, and chair, 2012-2015

Developed several MVA algorithms for data analysis in HEP (Hyperball, MuSclFit, Inverse bagging, Hemisphere mixing, INFERNO)

Editor of **Reviews in Physics**, an Elsevier journal

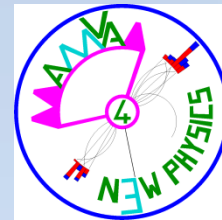
Scientific coordinator of **AMVA4NewPhysics**, ITN network on MVA for HEP

Scientific coordinator of accelerator-based research for INFN-Padova

Blogging about physics since 2005 - www.science20.com, formerly at <http://dorigo.wordpress.com> and <http://qd.typepad.com/6/>

In 2016 I wrote a book on searches for signals in CDF →

When I don't do physics or statistics I play chess / piano



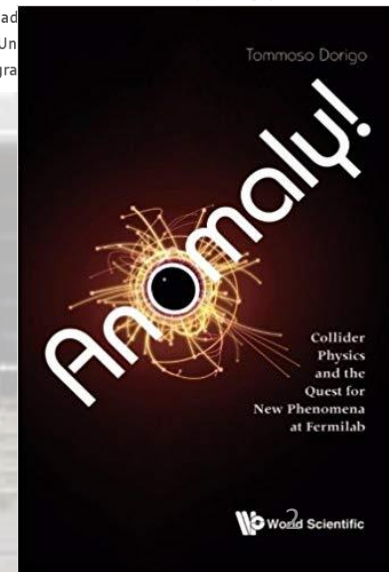
Toward A Solution Of The DAMA-Libra Dark Matter Puzzle

By Tommaso Dorigo | March 12th 2019 02:27 AM | 3923 reads | [Print](#) | [E-mail](#)

[RSS](#) [Share / Save](#) [Facebook](#) [Twitter](#) [Like](#)



What is dark matter (DM)? This is one of the most pressing questions in fundamental science nowadays. Dark matter exists in the Universe and appears to only interact gravitationally.



Notes on my lectures

- **My slides are full of text!**

- this makes it harder to focus on what I actually say during the lecture...

- On the other hand, they offer clarity for offline consumption / consultation

Try to pay attention to me rather than on the projected text. You will probably get the gist of it sooner.

- **I have way too much material**

the subject is vast! We will have to skip some things / go fast here and there

- **I am here at your service**

- Some of the things we will go over are highly non-intuitive, so I expect you to be confused if you have no previous understanding of a topic.

- hence please ask questions if you do not understand something. You need not raise your hand.

- sometimes I may omit answering straight away / the question may be better covered in later material

- **I will be at your service also after the school**

- please email me at tommaso.dorigo@gmail.com if you have nasty questions on the topics covered during the lectures, and I will do my best to answer them.

Suggested reading

These lectures are based on a number of different sources, as well as on some personal alchemy

I tried to provide references but sometimes I failed [apologies...]

A formal treatment of most of the covered material, and more in-depth than what I can go here, is offered in a couple of excellent textbooks:

Springer Series in Statistics

Trevor Hastie
Robert Tibshirani
Jerome Friedman

The Elements of Statistical Learning

Data Mining, Inference, and Prediction

Second Edition

Springer

Hastie, Tibshirani, Friedman:
The elements of statistical learning

→ AVAILABLE ONLINE FOR FREE!

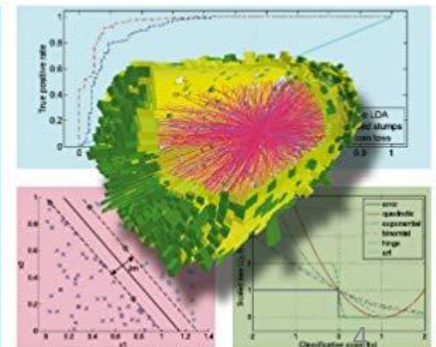
Narsky, Porter: Statistical Analysis
techniques in Particle Physics,
Wiley

Ilya Narsky, Frank C. Porter

WILEY-VCH

Statistical Analysis Techniques in Particle Physics

Fits, Density Estimation and Supervised Learning



Other credits

Besides the textbooks quoted above, the production of these lectures also benefited from perusing additional assorted material:

- **Trevor Hastie**'s lectures on Statistical Learning, Padova 2017
- Some material from **Michael Kagan**'s lectures at TASI 2017
- One or two ideas from **Ilya Narsky**'s lectures on ML with MatLab, 2018
- A couple of graphs from **Lorenzo Moneta**'s lecture at last year's school
- Some slides from interesting presentations at ACAT 2019

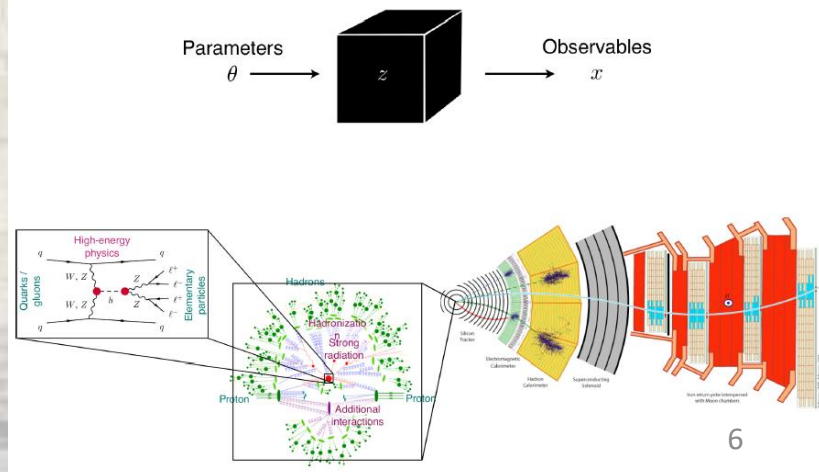
Plus random plots / stuff I collected around

Contents - 1

Lecture 1: An introduction to Machine Learning

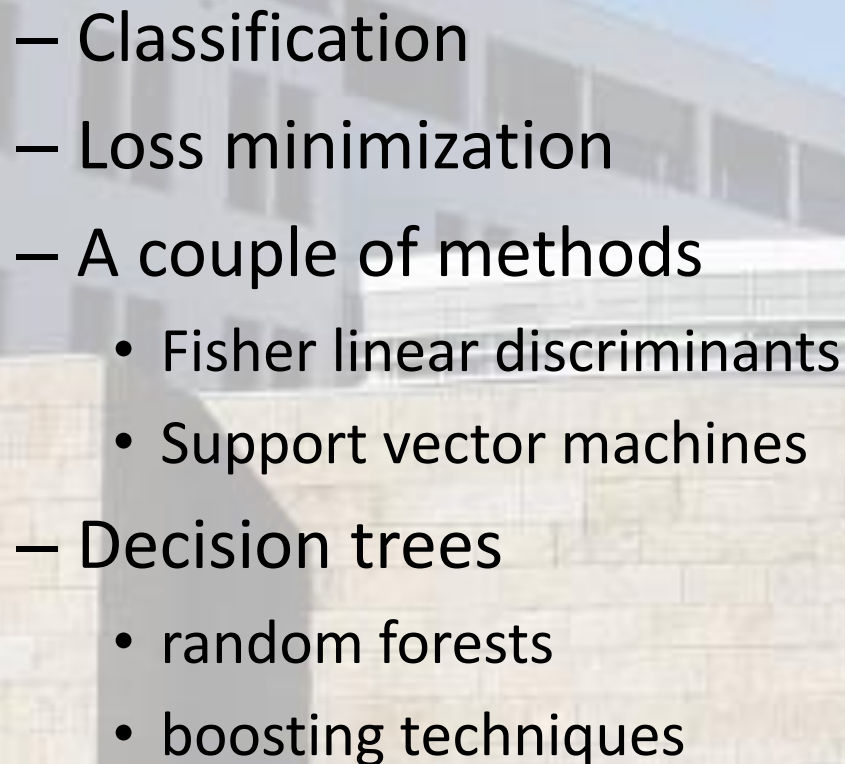
- Introduction
 - Map of ML problems
 - Supervised and unsupervised learning
- Density estimation
 - kNN
 - Divergence measures
- Resampling techniques
- The data
- The model

Particle physics



Contents - 2

Lecture 2: Classification and decision trees

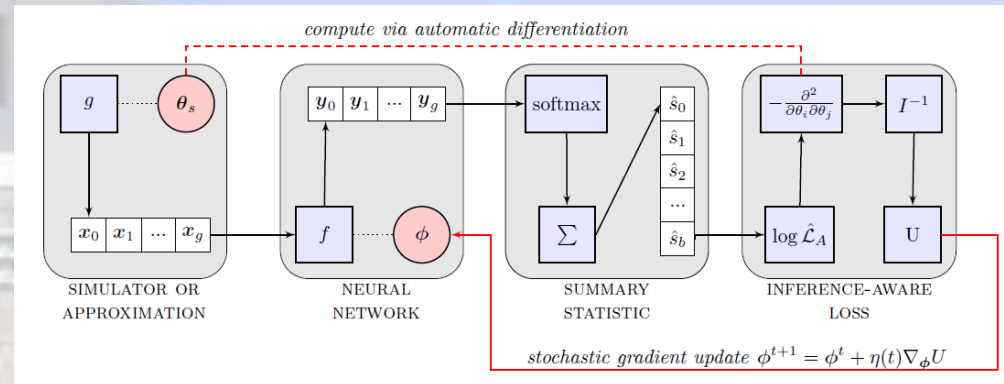
- 
- Classification
 - Loss minimization
 - A couple of methods
 - Fisher linear discriminants
 - Support vector machines
 - Decision trees
 - random forests
 - boosting techniques



Contents - 3

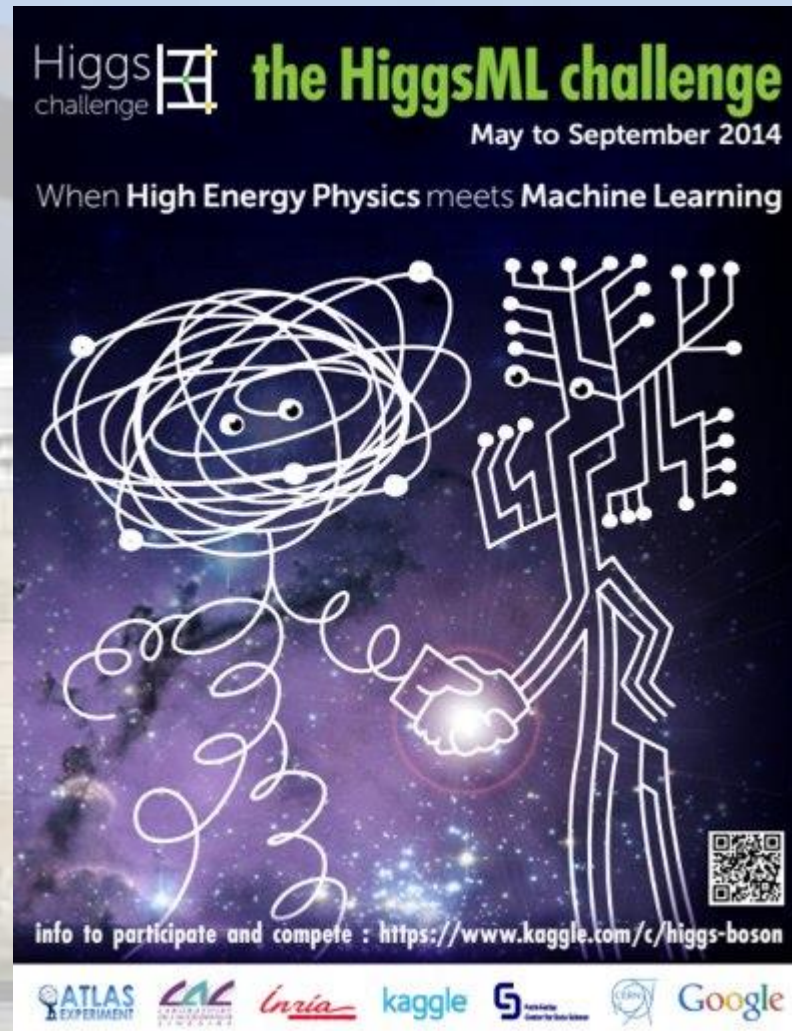
Lecture 3: NNs and HEP applications

- Neural networks
- Playing with NNs
- Advanced techniques
 - INFERNO
 - DNNs
 - Convolutional NNs
 - Genetic algorithms
- Practical tips
- Machine Learning in HEP
 - Generalities
 - kNN for $H \rightarrow b\bar{b}$
 - Clustering for HH theory space benchmarking
 - Higgs Kaggle challenge
- Conclusions

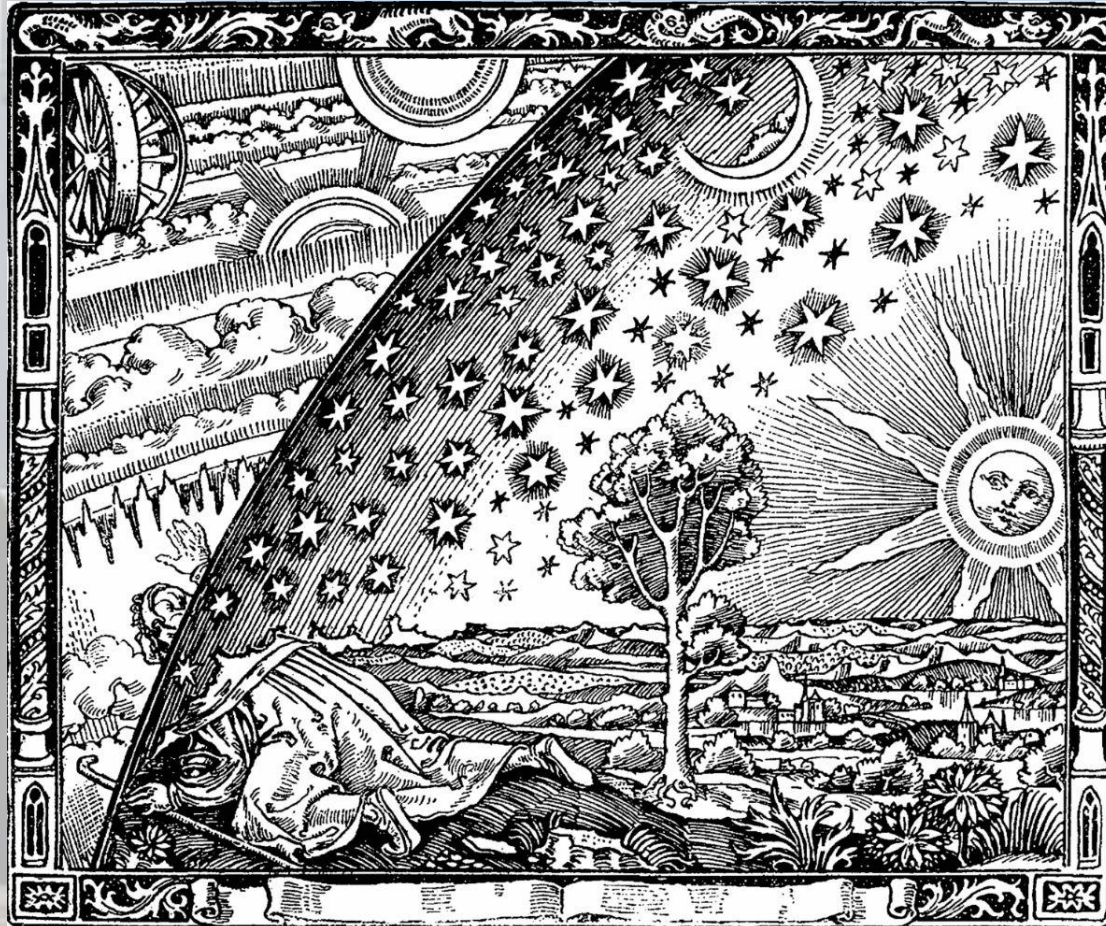


Lecture 1

An introduction to Machine Learning



INTRODUCTION



Machine Learning is all around us



Areas of development for ML tools

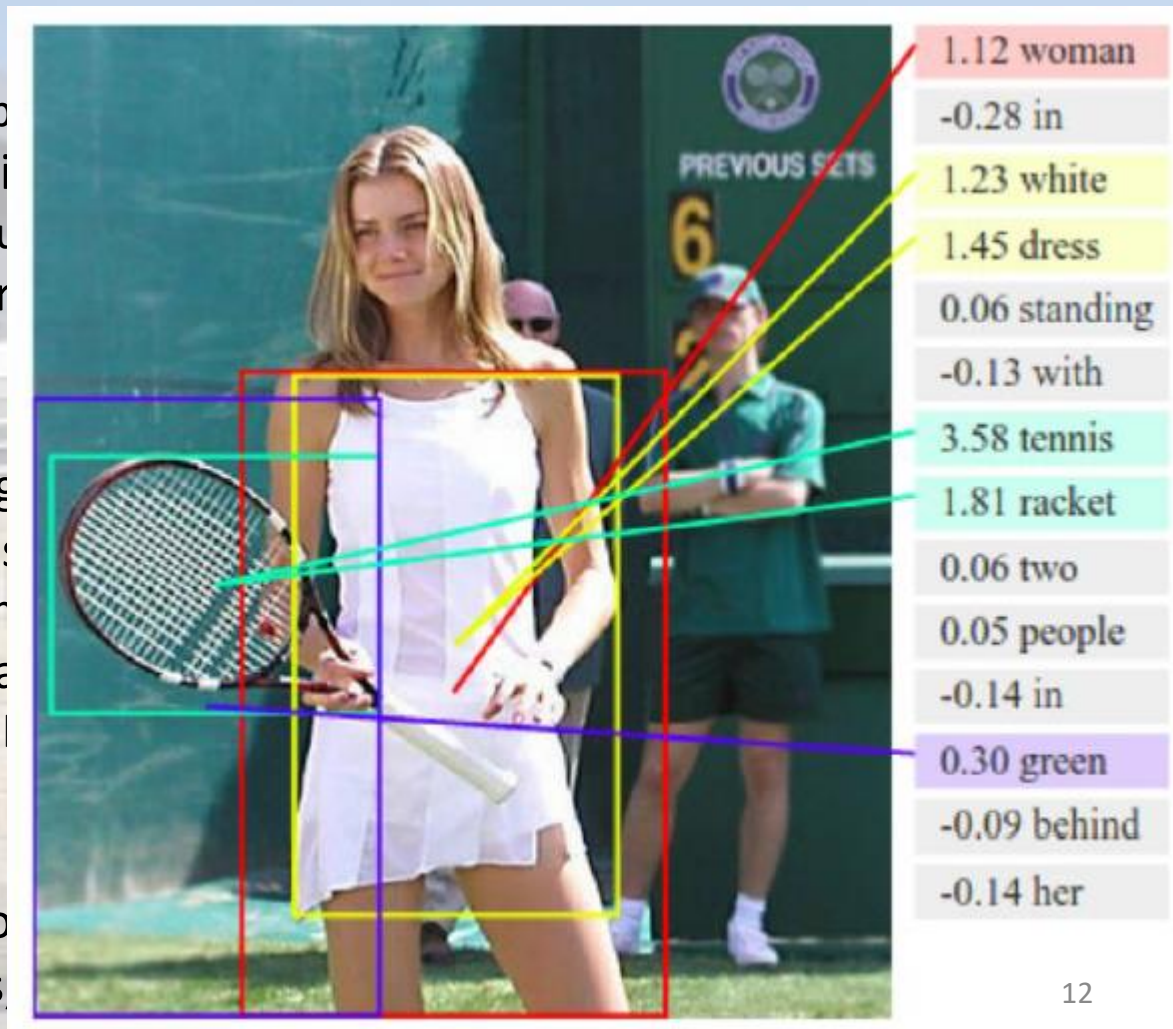
More market-driven:

- Speech and handwriting
 - often the previous step
- Search engines, advertising
 - ways to guess what you want
- Stock market analysis, price prediction

More research-oriented:

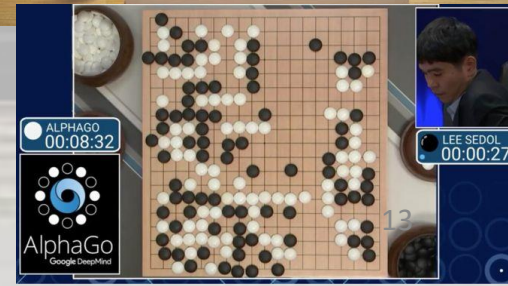
- Master closed systems (e.g. Siri)
- Natural language processing
 - allows computers to understand human language
- Self-driving cars, object recognition
 - methods to give spatial awareness
- AGI

→ Distinction more and more important
→ For fundamental physics



The path to Artificial Intelligence

- Alan Turing: machine shuffling 0's and 1's can simulate any mathematical deduction or computation
 - the Chinese room
- Dartmouth conference, 1956 → many astonishing developments, checkers, automated theorem solvers
 - and funding from US defense budget
- In the seventies, research effort dampened
- Eighties: Expert systems
- Nineties: the second AI winter
- Solutions to closed system problems; deep blue → alpha zero
- Development of Neural Networks
- Toward AGI



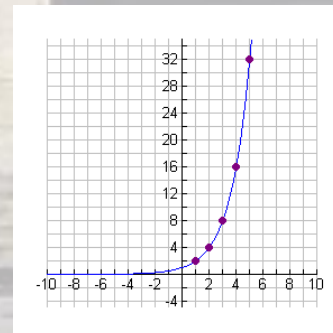
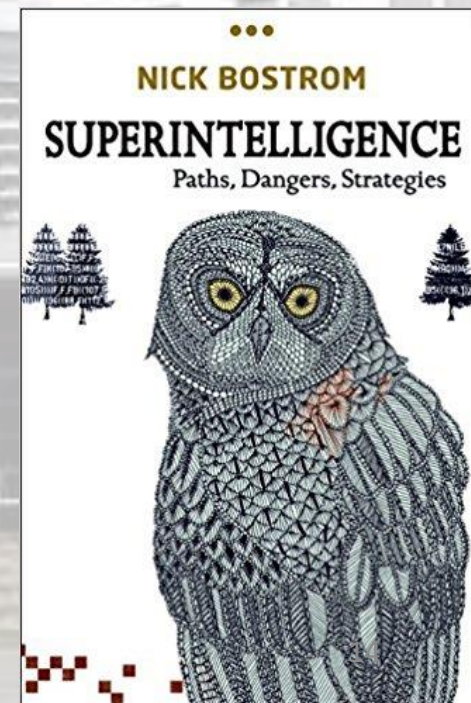
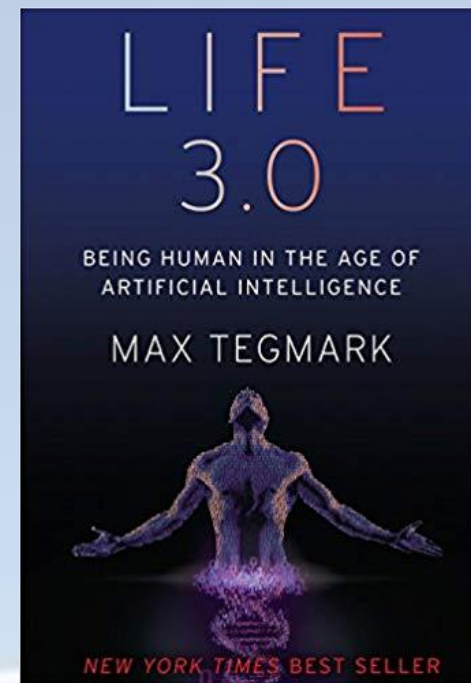
Is AI desirable?

Humanity is going to face in the next few decades the biggest "growing pain" of its history: **how to deal with the explosive force of artificial intelligence**

Once we create a superintelligence, there is no turning back – in terms of safeguards we have to "get it right the first time", or we risk to become extinct in a very short time

But we cannot stop AI research, no more than we could stop nuclear weapons proliferation

Interesting times ahead!



Can we define Machine Learning?

No self-respected lecture on Machine Learning avoids defining ML at the very beginning, and who am I to blow against the wind?

There are various options on how to define ML. Things I have heard around:

- “[Machine Learning is the] field of study that gives computers the ability to learn without being explicitly programmed.” - Arthur Samuel (1959)
 - ok, but learning what?
- Wikipedia: "A scientific discipline concerned with the design and development of algorithms that allow computers to evolve behaviors based on empirical data"
 - correct but slightly vague
- Mitchell (1997) provides a succinct definition: “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 ”
- The fitting of data with complex functions
 - in the case of ML, we learn the parameters AND the function
- Mathematical models learnt from data that characterize the patterns, regularities, and relationships amongst variables in the system
 - more lengthy description of "fitting data"

Can we agree on what "Learning" is?

Maybe this really is the relevant question. Not idle to answer it, as by clarifying what our goal is we take a step in the right direction

Some **definitions of Learning**:

- the process of acquiring new, or modifying existing, knowledge, behaviors, skills, values, or preferences
- the acquisition of knowledge or skills through study, experience, or being taught.
- becoming aware of (something) by information or from observation.
- Also, still valid is Mitchell's definition in previous slide

In general: **learning involves adapting our response to stimuli via a continuous inference.** The inference is done by comparing new data to data already processed. How is that done? **By the use of analogy.**

What about analogies?

Analogies: arguably the building blocks of our learning process. **We learn new features in a unknown object by analogy to known features in similar known objects**

This is true for language, tools, complex systems –
EVERYTHING WE INTERACT WITH

... But before we even start to use analogy for inference or guesswork, we have to **classify** unknown elements in their **equivalence class!**

(Also, one has to define a metric along which analogous objects align)

In a sense, **Classification** is even more "fundamental" in our learning process than is **Analogy**.

Or maybe we should say that classification is the key ingredient in building analogies.



L'Analogie
Cœur de la pensée

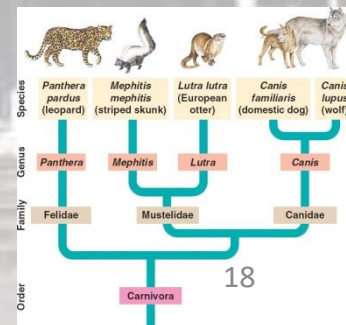
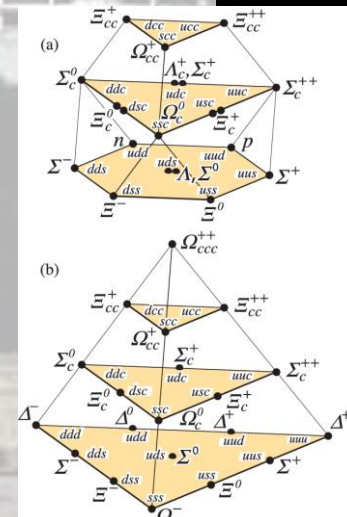
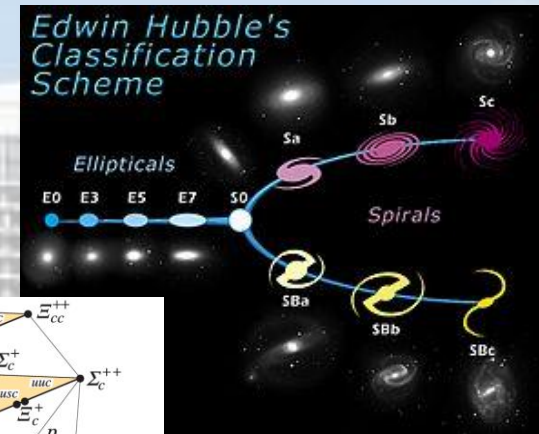
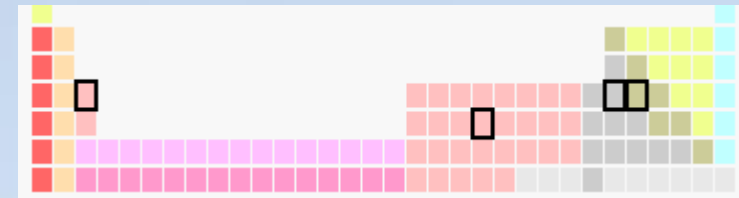
**Douglas
Hofstadter
Emmanuel
Sander**



Classification and the scientific method

Countless examples may be made of how breakthroughs in the understanding of the world were brought by classifying observations

- Mendeleev's table, 1869 (150 years ago!)
- Galaxies: Edwin Hubble, 1926
- Evolution of species: Darwin, 1859 (classification of organisms was instrumental in allowing a model of the evolution, although he used genealogy criteria rather than similarity)
- Hadrons: Gell-Mann and Zweig, 1964



Classification is at the heart of Decision Making

Finding similarities and distinctions between elements, creating classes, looks like an abstract task

Yet it is **at the heart of our decision making process**

→ in a game, **choosing the next move** entails a classification of the possible actions as good or bad ones, as a preprocessing step before going "deep" in the analysis

→ an **expert system** driving your vehicle must constantly decide how to react to external stimuli: it does so by classifying them (potentially dangerous → requires choice of proper reaction; irrelevant → can be ignored)

Decisions are **informed by previous experience**, and by information processing

That is how we "comprehend" the world and **assign it to some description**

Ultimately, an important component of intelligence is the continuous processing of our sensorial inputs, classifying them to react one way or the other to them

Classes of Statistical Learning algorithms

Supervised:

if we know the probability density of S and B , or if at least we can estimate it
→ E.g. we use "labeled" training events ("Signal" or "Background")
to estimate $p(x|S)$, $p(x|B)$ or their ratio

Semi-supervised:

it has been shown that even knowing the label for part of the data is sufficient to construct a classifier

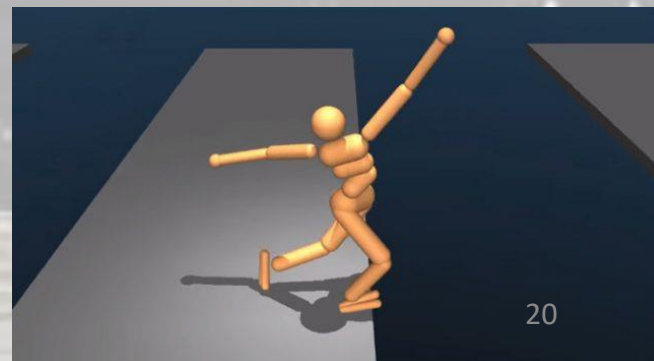
Un-supervised:

if we lack an a-priori notion of the structure of the data, and we let an algorithm discover it without e.g. labeling classes → **cluster analysis, anomaly detection, unconditional density estimation.**

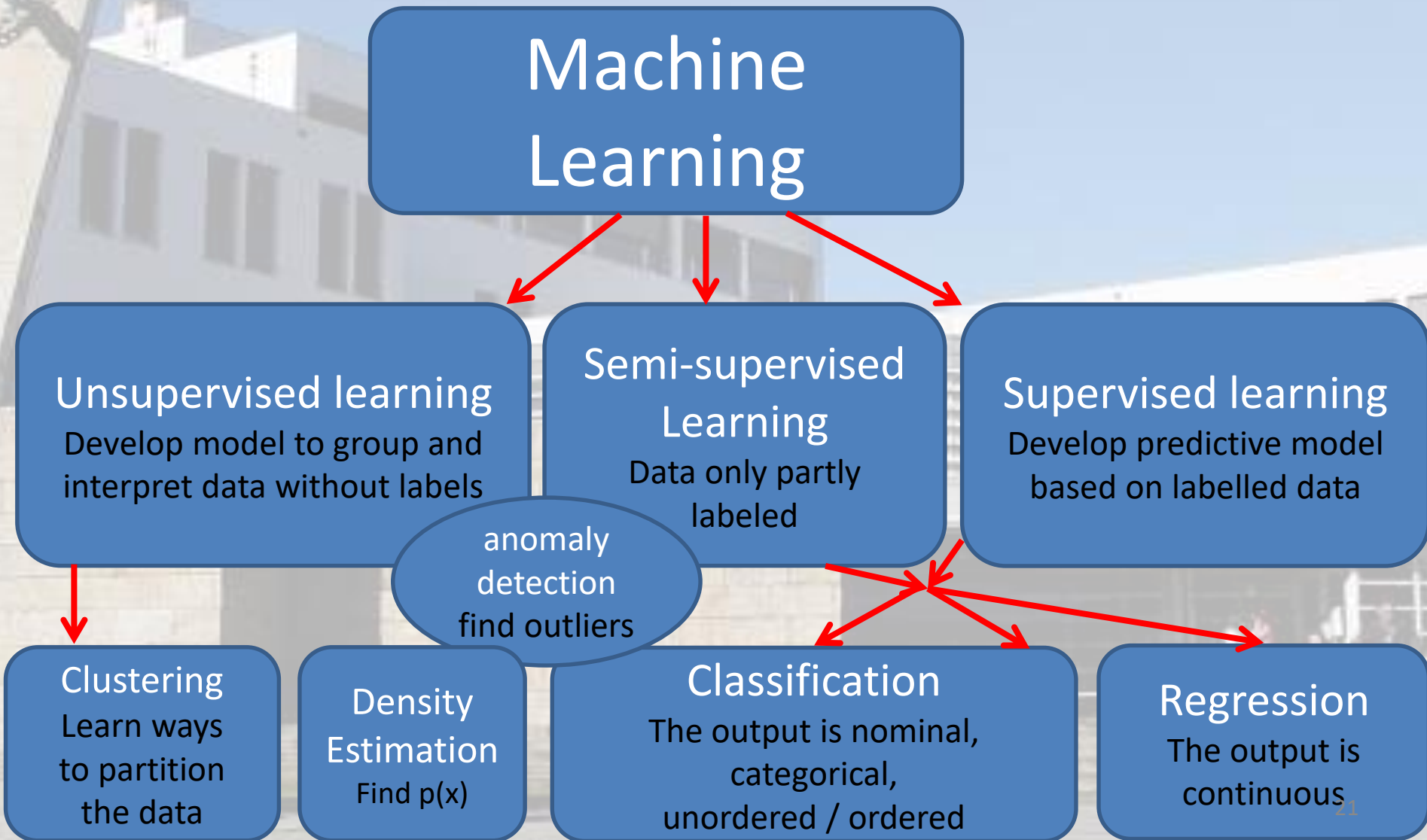
We may also single out:

Reinforcement learning:

the algorithm learns from the success or failure of its own actions
→ E.g. a robot reaches its goal or fails



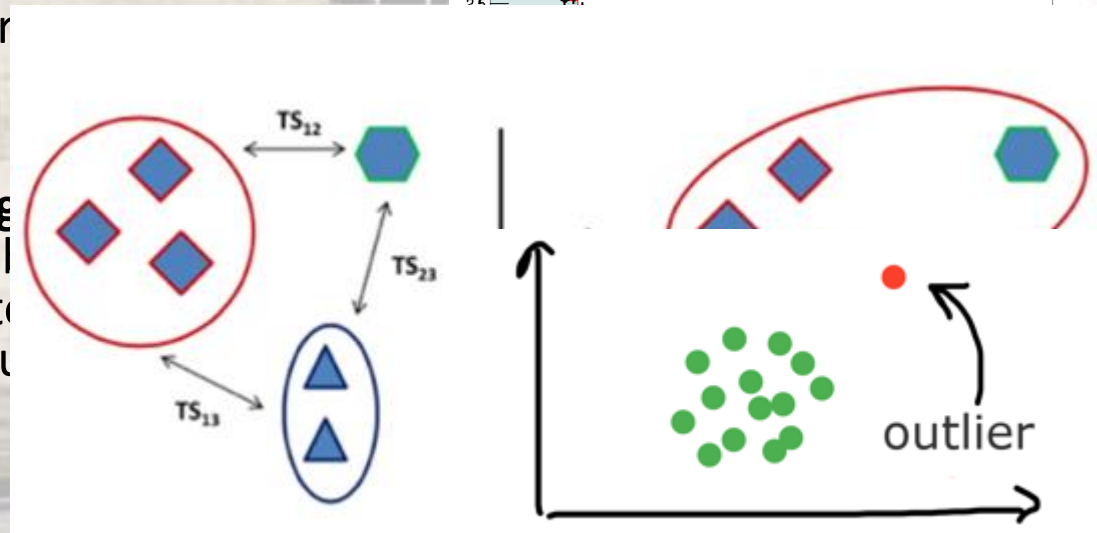
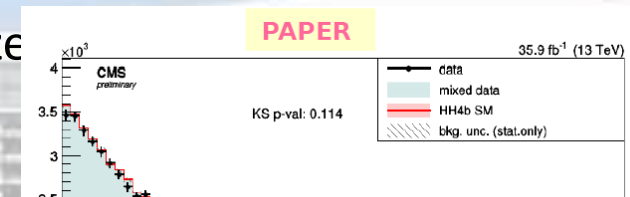
A map to clarify the players role



A more complete list of ML tasks /1

Classification and Regression are important tasks belonging to the "supervised" realm. But there are many other tasks:

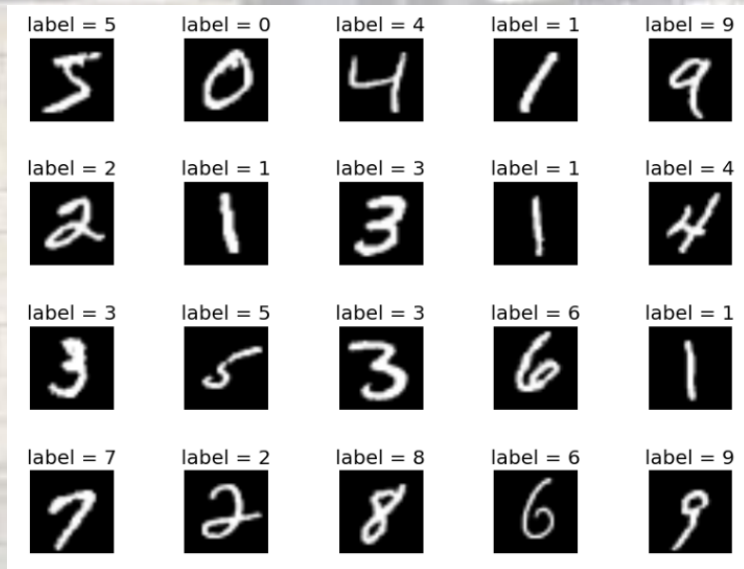
- **Density estimation:** this is usually an ingredient of classification, but can be a task of its own.
- **Clustering:** find structures in the data, organize them. This can be a useful input to other tasks
- **Anomaly detection** (e.g. fraud in purchasing habits; or new particles in physics)
- **Classification with missing data:** this is more than simple classification. It involves finding a mapping into the categorical space of missing components (but how to handle it)



A more complete list of ML tasks /2

- **Structured output:**

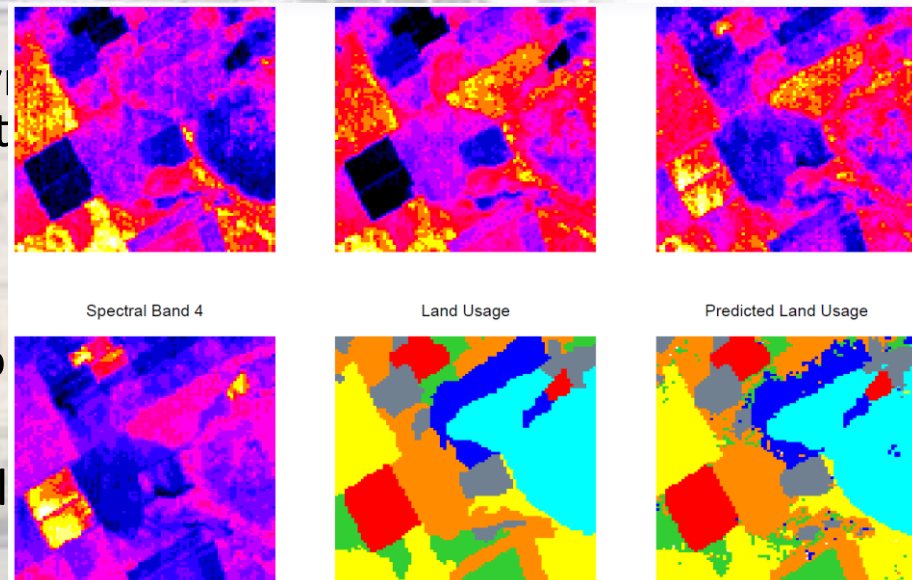
- **Transcription:** e.g. transform unstructured representation of data into discrete textual form; e.g. images of handwriting or numbers (Google Street view does it with DNN for street addresses), or speech recognition from audio stream
- **Machine translation**
- **parsing sentences** into grammatical structures
- **image segmentation**, e.g. aerial pictures → road positions or land usage
- **image captioning**



speech synthesis input

this can be used for custom

example



The supervised learning problem

- **Starting point:**

- A vector of n predictor measurements X (a.k.a. inputs, regressors, covariates, features, independent variables).
- One has training data $\{(x,y)\}$: events (or examples, instances, observations...)
- The outcome measurement Y (a.k.a. dependent variable, or response, or target)
 - In classification problems Y can take a discrete, unordered set of values (signal/background, index, type of class)
 - In regression, Y has a continuous value

- **Objective:**

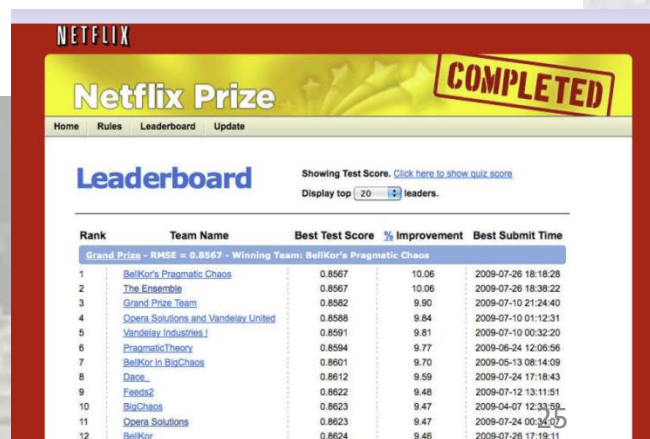
- Using the data at hand, we want to predict y^* given x^* , when (x^*, y^*) does not necessarily belong to the training set.
- The prediction should be **accurate**: $|f(x^*) - y^*|$ must be small according to some useful metric (se later)

- We would like to also:

- understand what feature of X affects the outcome and how
- assess the quality of our prediction

Example: the Netflix challenge

- competition started in October 2006. Training data is ratings for 18,000 movies by 400,000 Netflix customers, each rating between 1 and 5.
- training data is very sparse— about 98% missing.
- objective is to predict the rating for a set of 1 million customer-movie pairs that are missing in the training data.
- Netflix's original algorithm achieved a root MSE of 0.953. The first team to achieve a 10% improvement wins one million dollars.



The screenshot shows the Netflix Prize Leaderboard page. At the top, there's a yellow banner with 'Netflix Prize' and a 'COMPLETED' stamp. Below the banner, there's a navigation bar with 'Home', 'Rules', 'Leaderboard', and 'Update'. The main heading is 'Leaderboard'. To the right, it says 'Showing Test Score. [Click here to show quiz score](#)' and 'Display top 20 leaders.' Below this is a table with 5 columns: Rank, Team Name, Best Test Score, % Improvement, and Best Submit Time. The table lists 12 teams, with the top team being 'BellKor's Pragmatic Chaos' with a Best Test Score of 0.8567 and a % Improvement of 10.06.

Rank	Team Name	Best Test Score	% Improvement	Best Submit Time
Grand Prize - RMSE = 0.8567 - Winning Team: BellKor's Pragmatic Chaos				
1	BellKor's Pragmatic Chaos	0.8567	10.06	2009-07-26 18:18:28
2	The Ensemble	0.8567	10.06	2009-07-26 18:38:22
3	Grand Prize Team	0.8582	9.90	2009-07-10 21:24:40
4	Coopers Solutions and Vandelay United	0.8588	9.84	2009-07-10 01:12:31
5	Vandelay Industries I	0.8591	9.81	2009-07-10 00:32:20
6	PragmaticTheory	0.8594	9.77	2009-06-24 12:06:56
7	BellKor in BioChaos	0.8601	9.70	2009-05-13 08:14:09
8	Dave	0.8612	9.59	2009-07-24 17:18:43
9	Feedz	0.8622	9.48	2009-07-12 13:11:51
10	BioChaos	0.8623	9.47	2009-04-07 12:34:59
11	Coopers Solutions	0.8623	9.47	2009-07-24 00:34:40
12	BellKor	0.8624	9.46	2009-07-26 17:19:11

The unsupervised learning problem

- **Starting point:**
 - A vector of n predictor measurements X (a.k.a. inputs, regressors, covariates, features, independent variables).
 - One has training data $\{x\}$: events (or examples, instances, observations...)
 - There is no outcome variable Y
- **Objective** is much fuzzier:
 - Using the data at hand, find groups of events that behave similarly, find features that behave similarly, find linear combinations of features exhibiting largest variation
- Hard to find a metric to see how well you are doing
- Result can be **useful as a pre-processing step** for supervised learning

Ideal predictions, a' la Bayes

For a regression problem, the best prediction we can make for Y based on the input $X=x$ is given by the function

$$f(x) = \text{Ave}(Y|X=x)$$

This is the conditional expectation: you just derive the Y average for all examples having the relevant x .

- It is the best predictor if we want to minimize the average squared error,

$$\text{Ave}(Y-f(X))^2.$$

- But it is NOT the best predictor if you use other metrics. E.g., if you wish to minimize $\text{Ave}|Y-f(X)|$ you should rather pick... Who can guess it?

$$f(x) = \text{Median}(Y|X=x)$$

If we instead are after a qualitative output Y in $\{1...M\}$ (a discrete one, as in multi-class classification tasks) what we can do is to compute

$$P(Y=m|X=x)$$

for each m : conditional probability of class m at position $X=x$; then we take as the class prediction

$$C(x) = \text{Arg max}_j \{P(Y=j|X=x)\}.$$

The above is the majority vote classifier.

Problem solved? Let us try and see how to implement these ideas.

Implementation

To predict Y at $X=x^*$, collect all pairs (x^*, y) in your training data, then

- For regression, get

$$f(x^*) = \text{Ave } (y | X=x^*)$$

- For classification, get

$$c(x^*) = \text{Arg max}_j \{P(Y=j | X=x^*)\}$$

Alas, this would be good, but...

We usually have sparse training data, obtained by forward simulation. Our simulator gives $p(x|y)$ but the process is stochastic. That means we cannot invert the simulator, extracting $p(y|x)$!

In most cases we have NO observations with $X=x^*$.

Who you're gonna call ?

Density estimation methods!

DENSITY ESTIMATION



Density Estimation

Given a sample of data X , one wishes to determine their prior PDF $p(X)$.

One can solve this with parametric or non-parametric approaches.

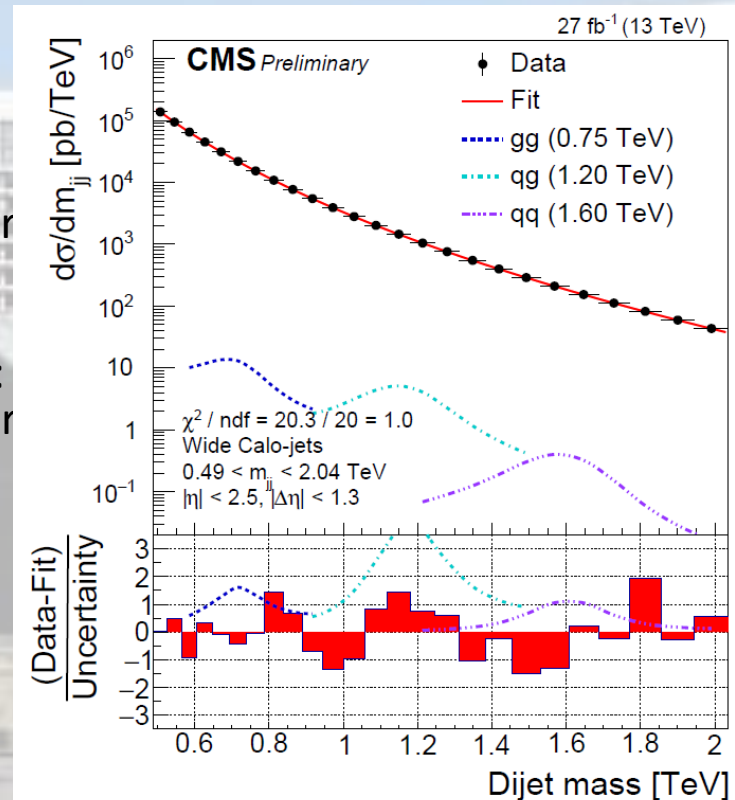
Parametric: find a model within a class, which fits the observed density

→ an assumption is necessary as the starting point

Non-parametric: sample-based estimators.

The most common is the histogram. NP density estimator have a number of attractive properties for experimental sciences:

- are easy to use for two things dear to HEP/astro-HEP: efficiency estimates (e.g. from Bernoulli trials) and for background subtraction
- lend themselves to be good inputs to unfolding methods
- are an excellent visualization tool, both in 1 and 2D

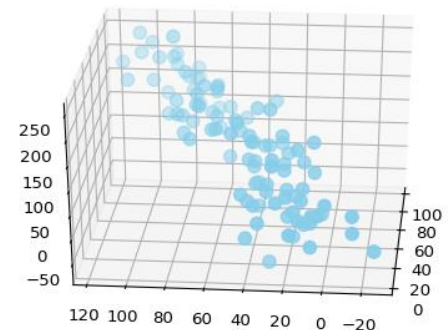
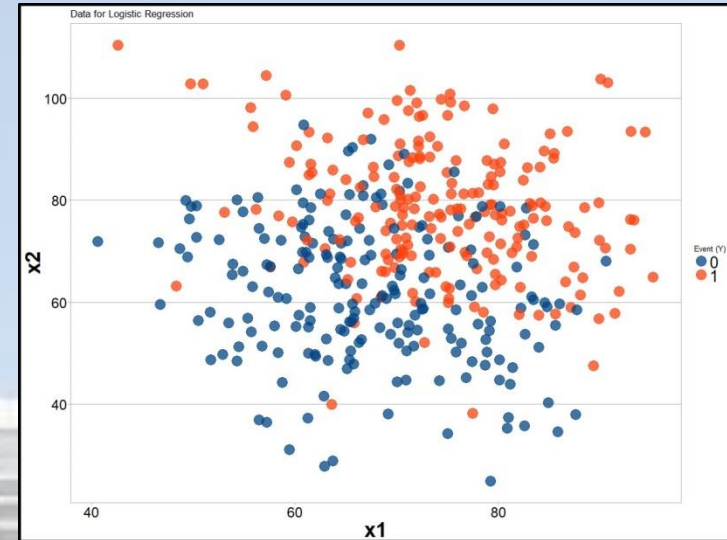


The empirical density estimate

With sparse data, the most obvious estimate of the density from which i.i.d. data $\{x_i\}$ are drawn is called "empirical probability density function" (EPDF), which can be obtained by placing a Dirac delta function at each observation x_i :

$$\hat{f}(x) = \frac{1}{N} \sum_{i=1}^N \delta(x - x_i)$$

Of course, the EPDF is rarely useful in practice as a computation tool. However, it is a common way of visualizing 2D or 3D data (scatterplots)



Histograms

Histograms are a versatile, intuitive, ubiquitous way to get a quick density estimate from data points:

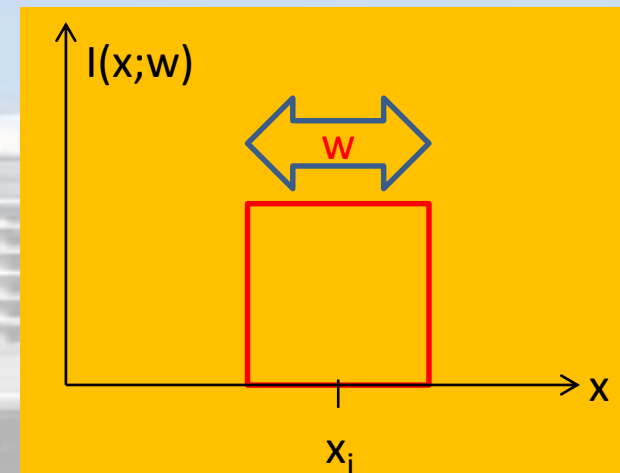
$$h(x) = \text{Sum}_{i=1 \dots N} I(x - x_i; w)$$

where w is the width of the bin, and I is a uniform interval function (indicator function),

$$I(x; w) = \begin{cases} 1 & \text{for } x \text{ in } [-w/2, w/2], \\ 0 & \text{otherwise} \end{cases}$$

The density estimate provided by $h(x)$ is then

$$f(x) = h(x)/(Nw)$$



Yet **histograms have drawbacks**:

- they are *discontinuous*
- they *lose information* on the true location of each data points through the use of a "regularization", the bin width w
- *not unique*: there is a 2D infinity of histogram-based PDF estimates possible for each dataset, depending on binning and offset.

Kernel density estimation

A useful generalization of the histogram is obtained by substituting the indicator function with a suitable "kernel":

$$\hat{f}(x) = \frac{1}{N} \sum_{i=1}^N k(x - x_i; w)$$

The kernel function $k(x, w)$ is normalized to unity. It is typically of the form

$$k(x - x_i; w) = \frac{1}{w} K\left(\frac{x - x_i}{w}\right)$$

The advantage of using a kernel instead than a delta is evident: **we obtain a continuous, smooth function**. This comes, of course, at the cost of a modeling assumption.

A common kernel is the **Gaussian distribution**; the results however depend more on the smoothing parameter w than on the choice of the specific form of k .

The "kernelization" of the data can be operated in multiple dimensions too. The kernels operating in each component are identical but may have different smoothing parameters:

$$\hat{f}(x, y) = \frac{1}{N w_x w_y} \sum_{i=1}^N K\left(\frac{x - x_i}{w_x}\right) K\left(\frac{y - y_i}{w_y}\right)$$

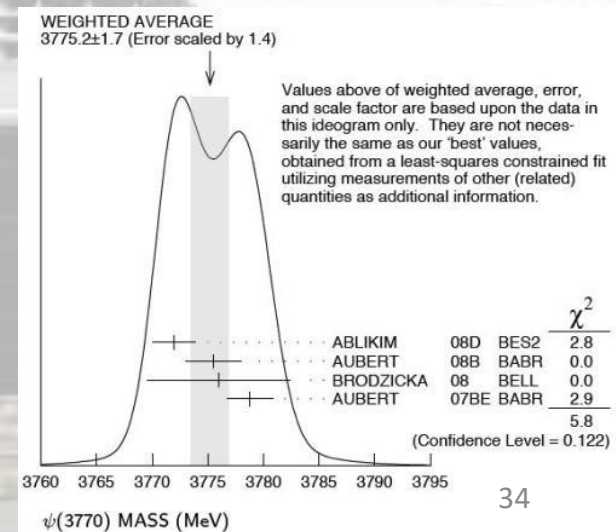
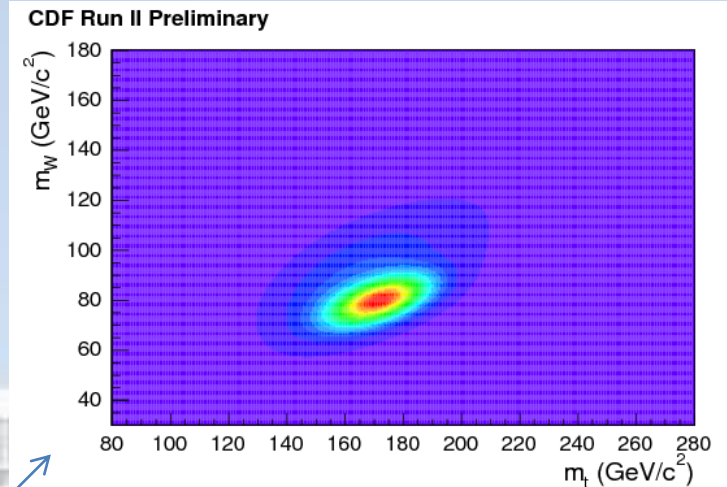
A different extension of the KDE idea is to adapt the smoothness parameter w to reflect the precision of the local estimate of the data density → adaptive kernel estimation

Ideograms

A further extension of KDE is the **use of different kernels for different data points** → **ideograms**. This may be very useful if the data coordinate x comes endowed with information on its measurement precision σ . In that case we may **substitute the point with a Gaussian kernel, whose smoothing parameter w is equal to the precision σ** . This improves the overall density estimate

The ideogram method is sometimes used in HEP, e.g. it has been employed in top quark mass measurements.

The PDG uses it to report the PDF of a unknown parameter when the individual estimates carry different uncertainty AND there is tension between the estimates (such that the weighted average is not necessarily the whole story)

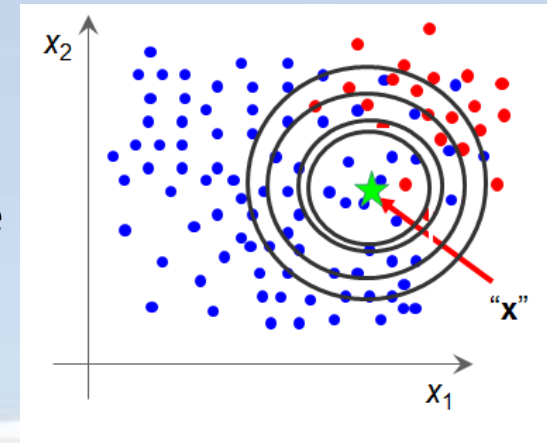


Going multi-D: K-Nearest Neighbours

The kNN algorithm tries to determine the local density of multi-dimensional data by finding how many data points are contained in the **surroundings** of every point in feature space

One usually "weighs" data points with a suitable function of the distance from the test point

Problem: **how to define the distance in an abstract space?**



Also, your features might include real numbers, categories, days of the week, etcetera...
In general, it is useful to remove the dimensional nature of the features

General recipe: **standardization**

- for continuous variable x : find variance σ^2 , obtain standardized variable $y^2 = x^2 / \sigma^2$
- for categorical variable c :
 - if target has large variance along c , can keep it as is (defines hyperplane, will only use data with $c^* = c$ to estimate local density)
 - otherwise may still try to standardize

More on kNN

Once the data is properly standardized one can construct an Euclidean distance:

$$D(y, y') = \sum_{i=1}^D (y - y')^2$$

kNN density estimates can be endowed with several parameters to improve their performance

Most obvious is k: how many events in the ball?

Rule of thumb: estimating a mean → if target varies a lot, can use small k; if variation small, k needs to allow precise estimate

Common to have k=20-50, but it of course depends on dimensionality D and size N of training data

kNN, continued

Also crucial to assess:

- **relative importance of variables:** assign larger weight to more meaningful components in the feature space (ones along which target has largest variance)
- **local gradients-aware:** one may try and adapt the shape of the "hyperellipsoid" to reflect how much target $y(x)$ is variable in each direction, at test point x^*

In general, kNN estimates suffer from a couple of shortcomings:

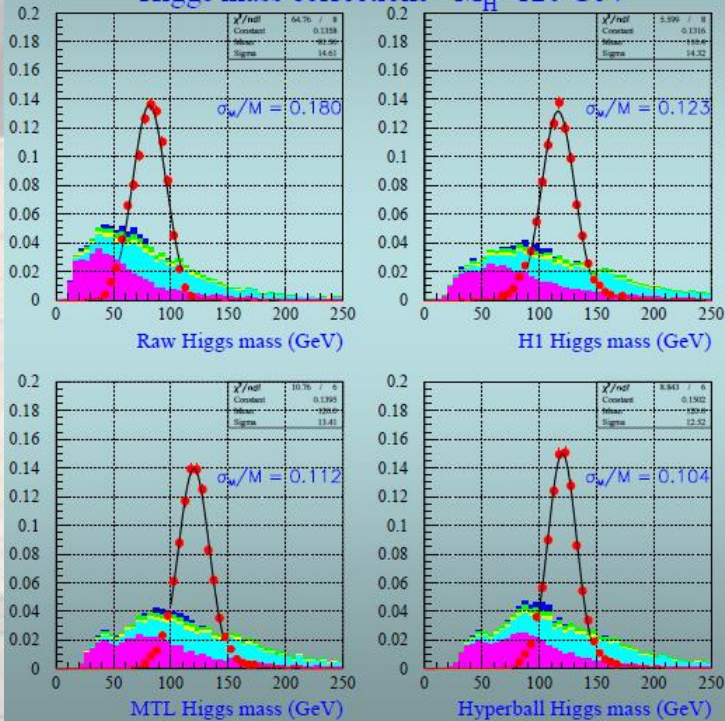
- The evaluation of local density requires to use all data for each point of feature space → **CPU expensive** (but there are shortcuts)
- The **curse of dimensionality:** for $D > 8-9$, they become insensitive to local density (see next slide)

One example

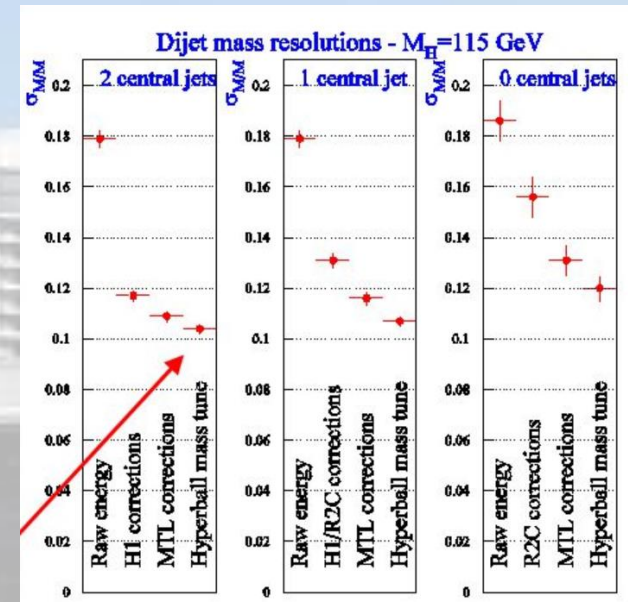
In 2003 the Tevatron experiments assessed their chances to observe the Higgs with more data / improved detectors

One of the crucial issues: M_{bb} regression

Higgs mass corrections - $M_H = 120$ GeV



Customized kNN algorithm with balls shaped by local gradient ("Hyperball algorithm") brought down relative mass resolution from 17% to 10% \rightarrow large increase in observability



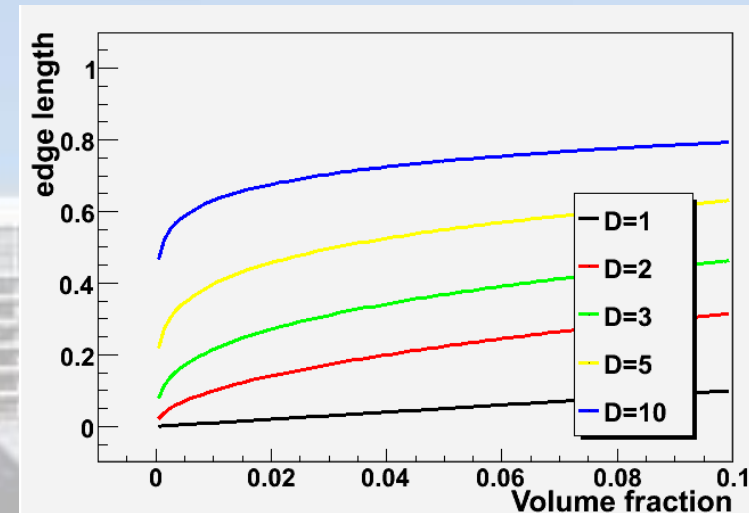
The Curse of Dimensionality

The estimate of density in a D-dimensional space is usually impossible, due to the lack of sufficient labeled data for the calculation to be meaningful. An adequate amount of data must grow exponentially with D for a good representation

For high problem dimensionality, the k "closest" events are in no way close to the point where an estimate of the density is sought:

$$\text{edge length} = (\text{fraction of volume})^{1/D}$$

This makes kNN impractical for $D > 8-10$ dimensions



In 10 dimensions, if a hypersphere captures 1% of the feature space, it has a radius of 63% in each variable span

HEP analyses often have >10 important variables so the kNN has limited use as a generative algorithm for S/B discrimination

But, one can apply dimensional reduction techniques to still use it, or resample subspaces³⁹

What If We Ignore Correlations?

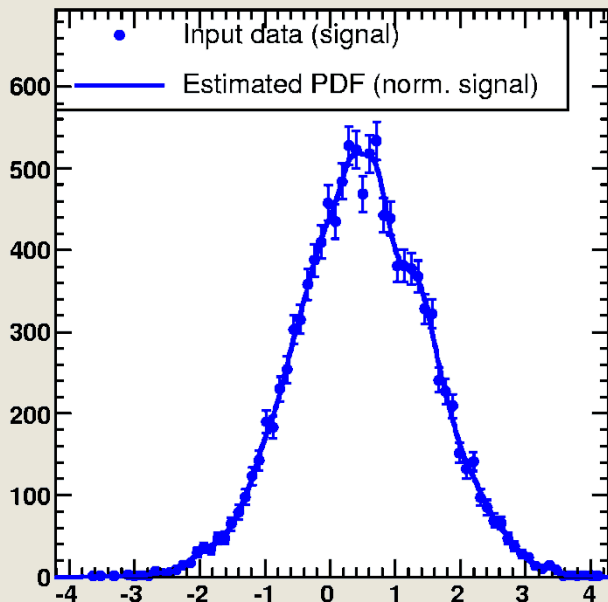
All PDF-estimation methods similarly fail when D is large.

A "**Naïve Bayesian**" approach consists in ignoring altogether the correlations between the variables of the feature space

That is, one only looks at the "marginals": $P(\mathbf{x}) \cong \prod_{i=0}^D P_i(\mathbf{x})$

P is a product of "marginal PDFs" P_i

One usually models the P_i with a smoothing of histograms



The $P(x)$ thus obtained can be used to construct a discriminant (a likelihood ratio), or more simply, to assign the class label to the highest $p(x)$ given x

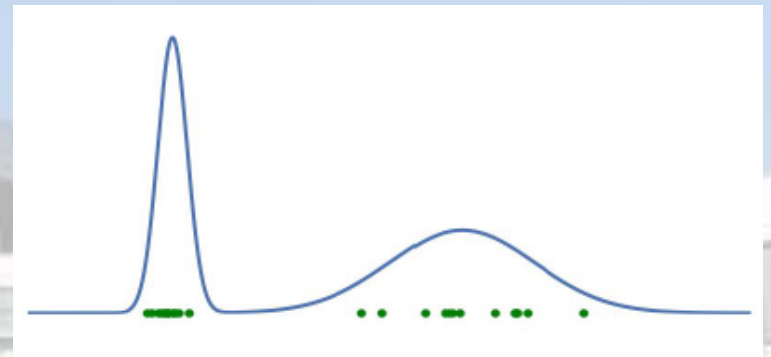
The method works if correlations are unimportant.
In HEP, however, this is not usually the case!

A small diversion: parametric density estimators

Given an empirical density estimate $p(x)$ obtained by N i.i.d. data points $\{x_i\}$, we might want/need to formulate our problem as that of **finding parameters of a function $q(\theta)$ such that it gets "closer" to $p(x)$.**

The focus is what "closer" means!

The density estimation problem can be summarized formally as follows:



What we need is to construct a probability **divergence $D(Q; P)$** between two smooth probability densities $Q: \mathbb{R}^d \rightarrow \mathbb{R}^+$ and $P: \mathbb{R}^d \rightarrow \mathbb{R}^+$, where Q is defined by a parametric generative model with parameters θ , and P is a target density.

If H is a space of densities ($H = \{p: \mathbb{R}^d \rightarrow \mathbb{R}^+, \|p\| = 1\}$), we can define a probability divergence $D(P; Q)$ as a map $D: H^2 \rightarrow \mathbb{R}^+$ that enjoys some properties (next slide)

The goal is to have a consistent interpretation of $D(Q; P)$ as well as use it to modify the parameters so that Q approaches P .

Divergence measures

In general, one could argue that all we need is to find some D which guarantees

- $D(Q,P) = 0 \iff Q=P$
- $D(Q,P) \geq 0$

There are additional properties we might want D to have; e.g., be a **metric**. A metric also has the properties that

- $D(Q,P) = D(P,Q)$
- $D(A,C) \leq D(A,B) + D(B,C)$ (triangle inequality)

The improved interpretability of distance between densities that is offered by these added properties is an important property.

But we might rather prefer our divergence to have other properties. In particular, we wish to make our D a **measure of relative information**.

This is what makes the **Kullback-Leibler divergence** KL so useful.

The Kullback-Leibler divergence

The KL divergence has its origins in **information theory**, where one wishes to quantify the amount of information contained in data.

One defines Entropy H for a probability distribution as

$$H = - \sum_{i=1}^N p(x_i) \cdot \log p(x_i)$$

In base 2, H is the **minimum number of bits that encode the available information of p** . If we know this, we have a way to quantify the information in our data.

Given that, we can extract the information loss we incur in, if we substitute the data p with a model q of it. This is given by the KL divergence, defined as (for discrete datasets)

$$D_{KL}(p||q) = \sum_{i=1}^N p(x_i) \cdot (\log p(x_i) - \log q(x_i))$$

KL / continued

What the KL divergence provides is the **expectation of the log difference between the probability of data in the original distribution with the approximating distribution**. So we could write it as

$$D_{KL}(p||q) = E[\log p(x) - \log q(x)]$$

The value of D_{KL} gives us a measure of the number of bits we lose if we approximate p with q . As such, it does not possess one of the properties of a true metric, as

$$D_{KL}(p;q) \neq D_{KL}(q;p).$$

But its importance cannot be underestimated in ML, as it provides exactly what we need in our attempts to estimate densities by successive approximations.

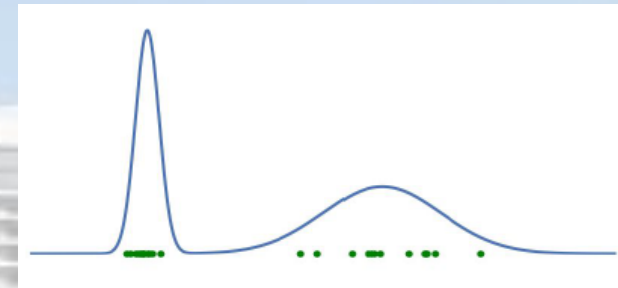
→ often used in the definition of loss functions

Another way to see D_{KL} : expected number of bits that we have to transmit to a receiver in order to communicate the density Q given that the receiver already knows the density P

Example: comparison of divergences

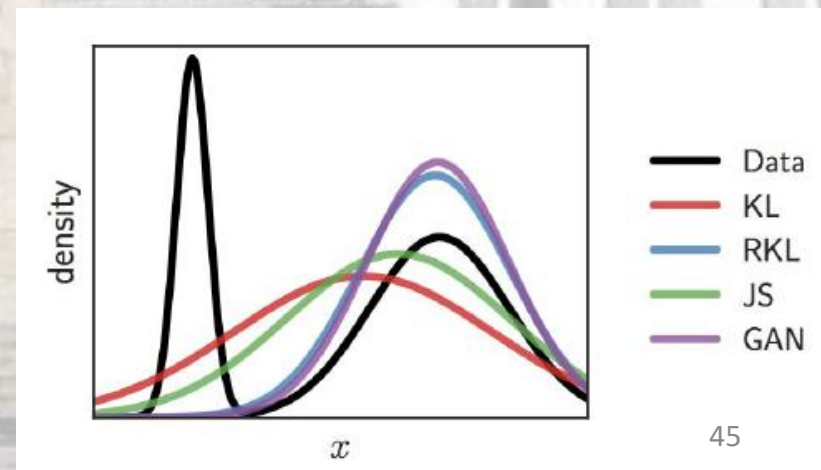
Below are some formulations of different choices for probability divergence measures; p is the data and q is the model

Name	$D_f(p(x) q(x))$
"forward" Kullback-Leibler	$\int p(x) \log \frac{p(x)}{q(x)} dx$
"reverse" Kullback-Leibler	$\int q(x) \log \frac{q(x)}{p(x)} dx$
Jensen-Shannon	$\int \left\{ \frac{1}{2} p(x) \log \frac{2p(x)}{p(x)+q(x)} + q(x) \log \frac{2q(x)}{p(x)+q(x)} \right\} dx$
GAN	$\int \left\{ p(x) \log \frac{2p(x)}{p(x)+q(x)} + q(x) \log \frac{2q(x)}{p(x)+q(x)} \right\} dx$



The minimization of different divergence measures for a Gaussian model of the data provides quite different answers →

This shows that it is important to know what we are using in our minimization problem!



RESAMPLING TECHNIQUES



Resampling Techniques

Resampling techniques are pivotal for a number of ML tasks:

- hypothesis testing (construction of discriminative methods)
- estimation of bias and variance (optimization of predictors)
- cross-validation (estimate accuracy of predictors)

Resampling allows you to avoid modeling assumptions, as you construct a non-parametric model from the data themselves. The benefits can be huge (as measured e.g. by performance of boosting methods)

Here we only briefly flash the generalities of three basic ingredients:

- permutation tests
- bootstrap
- jackknife
- ~~cross-validation techniques~~ ← will treat later

The theoretical basis for resampling methods lies with their (usually very good) asymptotic properties of consistency and convergence (in probability)

Permutation Sampling

Permutation sampling is mainly used in hypothesis tests; usually the question is: are datasets **A** $\{x_1 \dots x_{N_A}\}$ and **B** $\{x_1 \dots x_{N_B}\}$ sampled from the same parent PDF?

The way to answer is to form a sensitive statistic T (say: the mean of the observations) and **compare quantitatively** the difference between T_A and T_B

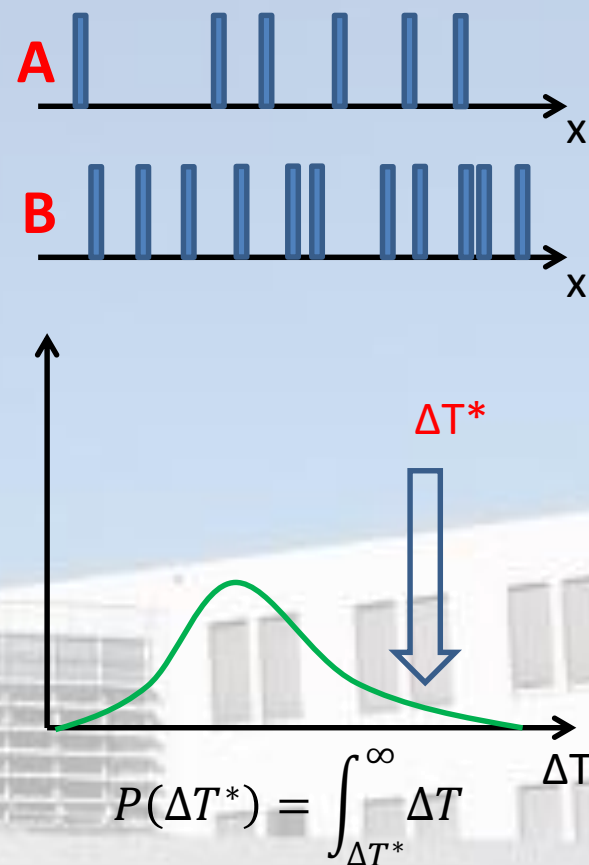
$$\Delta T^* = T_A - T_B$$

with what one would expect if the PDF were the same.

The comparison requires to **know how T distributes under the null hypothesis** (green curve).

Here permutation comes in: we assume $A=B$, merge $A+B=C$, and sample the N_A+N_B observations creating all possible pairs of splits A', B' **still of size $\{N_A, N_B\}$** , computing distances $\Delta T = T_{A'} - T_{B'}$.

This provides **all the information in the data** about the distribution of ΔT . **The permutation test makes no model assumptions, so it is called an exact test.**



The Bootstrap

The Bootstrap (B.Efron, 1979) is called this way because it allows to "*pull oneself up from one's own bootstraps*".



Motivating problem: get the variance of an estimator $\hat{\theta}(x)$ of a parameter θ , for a sample of i.i.d. observations $\{x_1 \dots x_N\}$.

We may **generate M replicas of the dataset X** by repeatedly picking N observations at random from X , with replacement.

Then we estimate θ in each replica, and proceed to obtain a sample mean and variance with the $\hat{\theta}_i(x)$,

$$\bar{\theta} = \frac{1}{M} \sum_{i=1}^M \theta_i$$

$$s^2_{\theta} = \frac{1}{M-1} \sum_{i=1}^M (\hat{\theta}_i - \bar{\theta})^2$$

You get an estimate of variance without assumptions of the distribution

Bootstrap-cum-density estimation

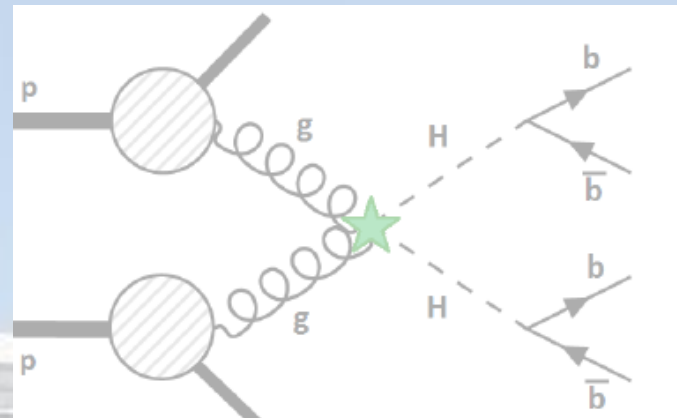
example: $hh \rightarrow bbbb$ CMS 2017

In a recent CMS search of $hh \rightarrow bbbb$ production (CMS-PAS-HIG-17-017), the background was dominated by QCD multijet production – notoriously a difficult process to model with MC simulations

In order to train a classifier, you need a good multi-D model of the features...

A synthetic background sample was constructed by **mixing hemispheres**

The idea is that QCD gives two hard quarks or gluons that scatter off one another, and the complexity arises independently, as a combination of FSR, pileup, MI
 \rightarrow can try to create models of single "halves" of the events



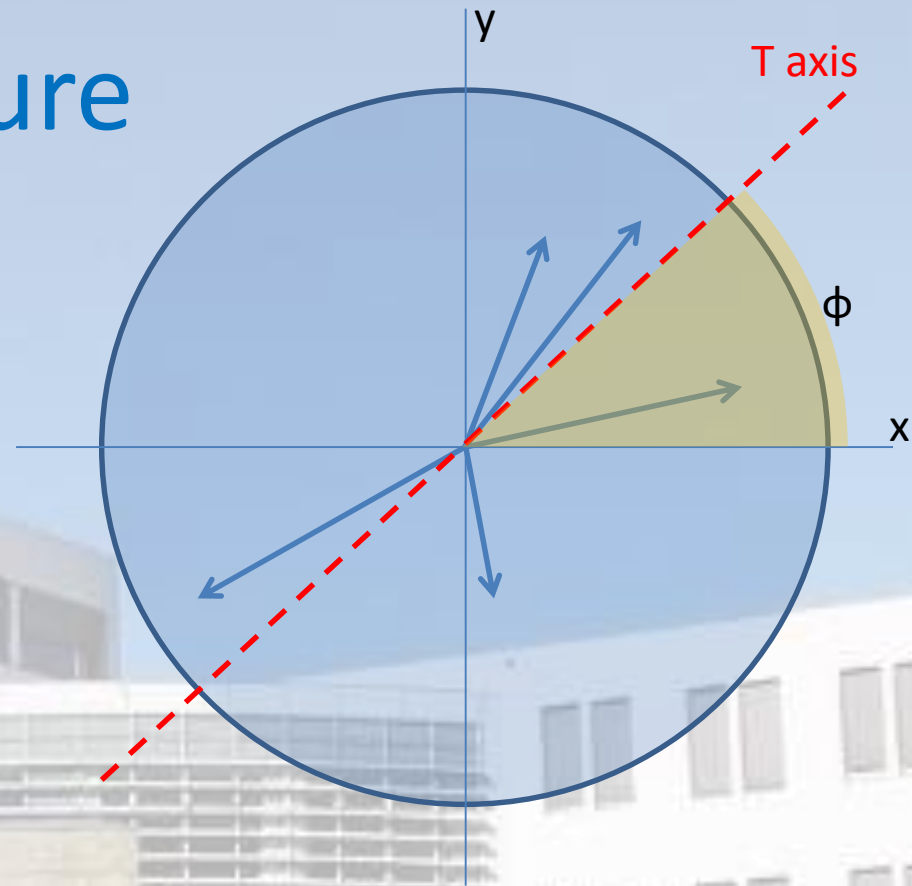
The mixing procedure

1) **For each event** in the original sample:

- Find transverse thrust axis
i.e., determine angle ϕ such that

$$T = \sum p_T^{jet} \cos(\varphi_T - \varphi_{jet})$$

is maximized

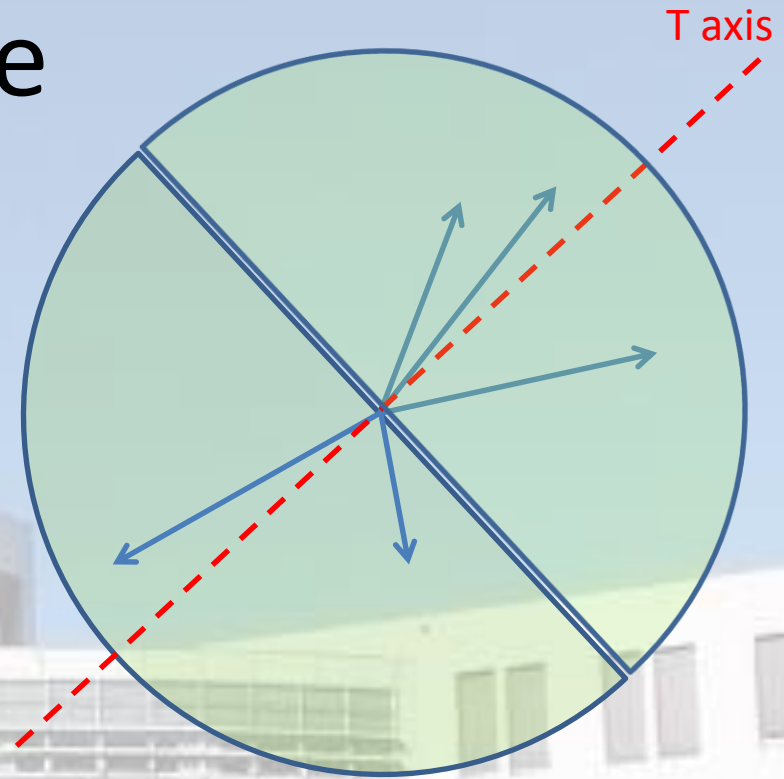


The mixing procedure

1) **For each event** in the original sample:

- Find transverse thrust axis
- **Divide event in two halves using plane orthogonal to it**

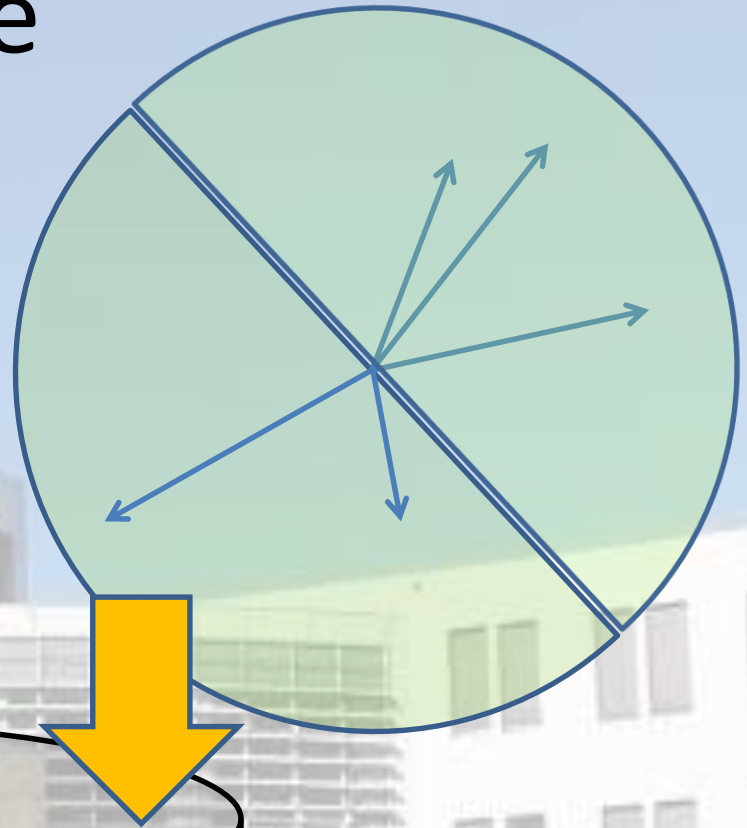
This defines *two* jet collections for each event (hemispheres)



The mixing procedure

1) **For each event** in the original sample:

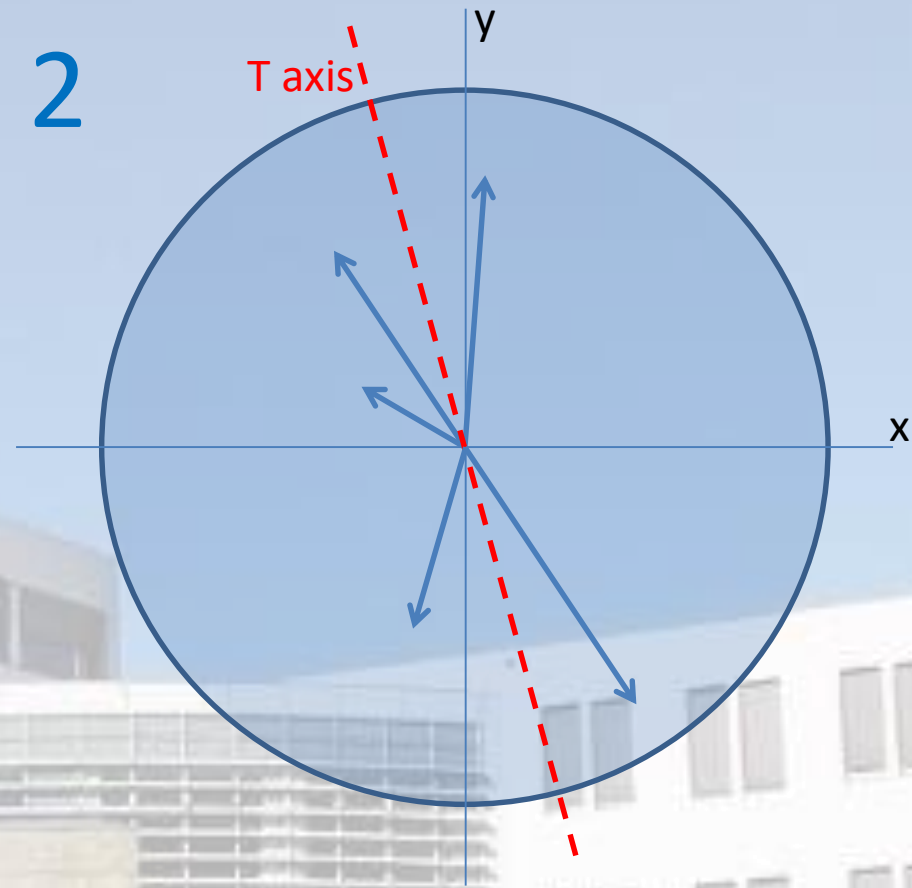
- Find transverse thrust axis
- Divide event in two halves using plane orthogonal to it
- **Store resulting hemispheres in the "hemisphere library"**



Mixing procedure - 2

2) Take again original sample: for each event

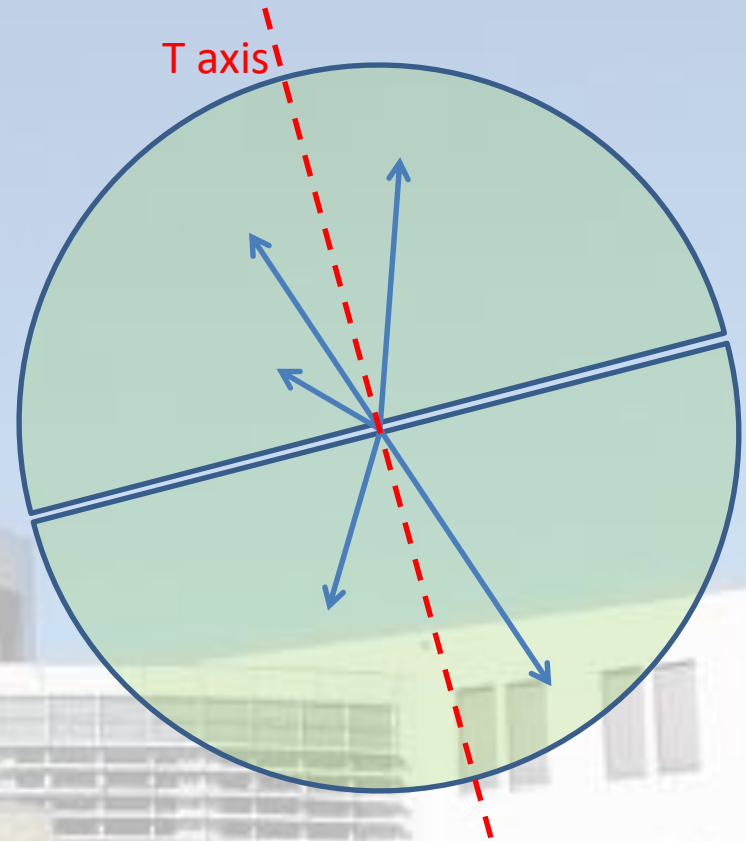
- Find transverse thrust axis,



Mixing procedure - 2

2) Take again original sample: for each event

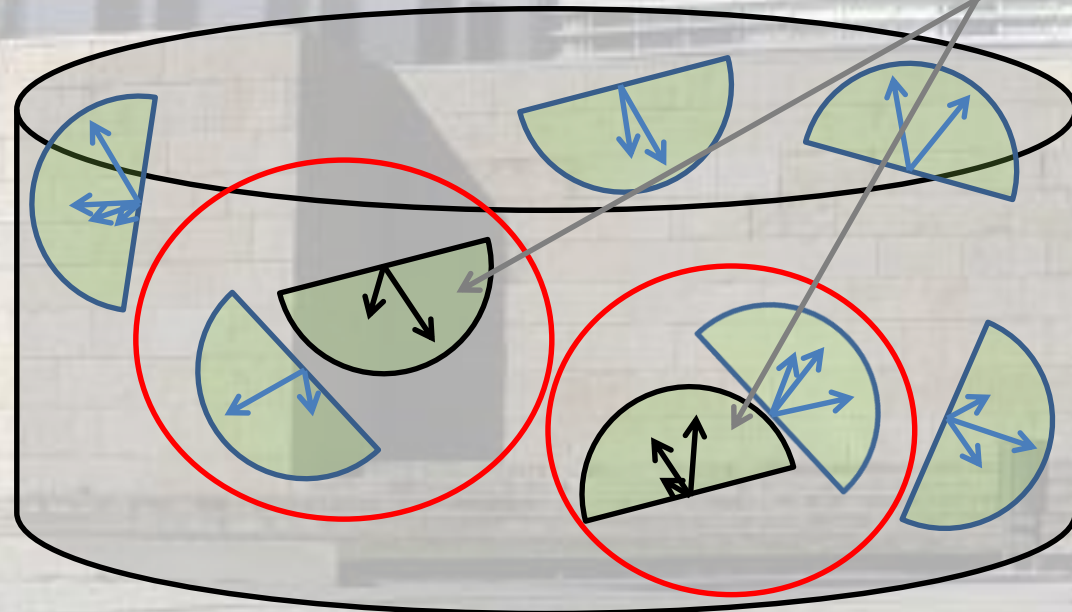
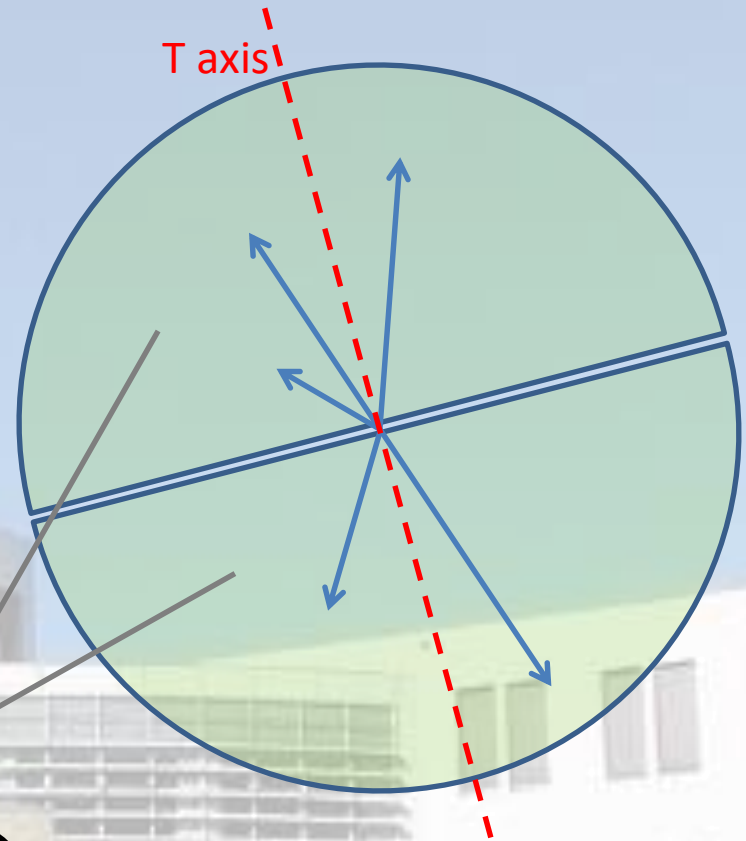
- Find transverse thrust axis, identify the two hemispheres making it up



Mixing procedure - 2

2) Take again original sample: for each event

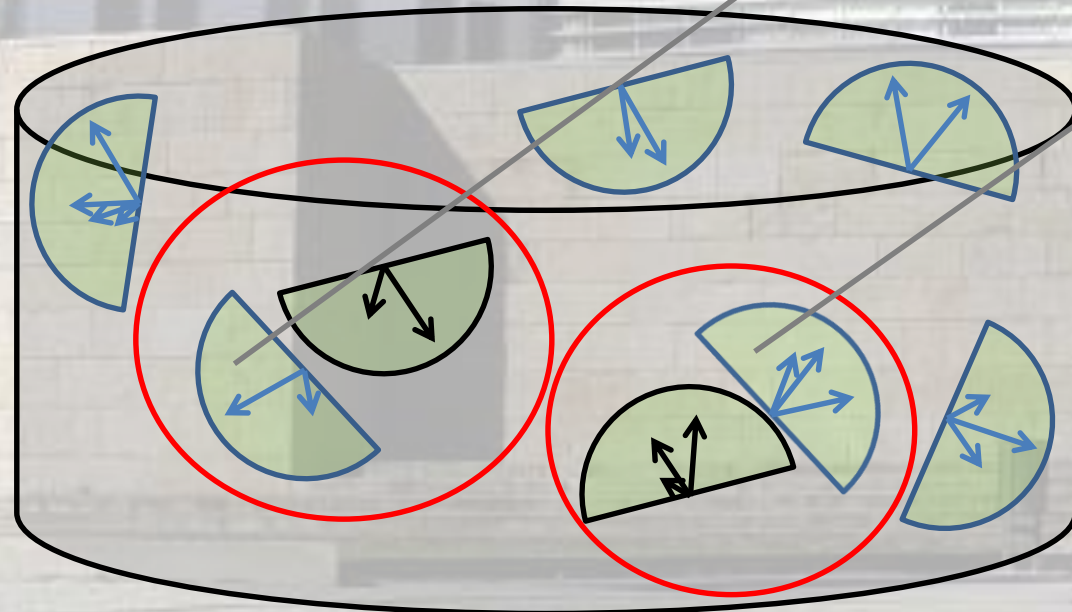
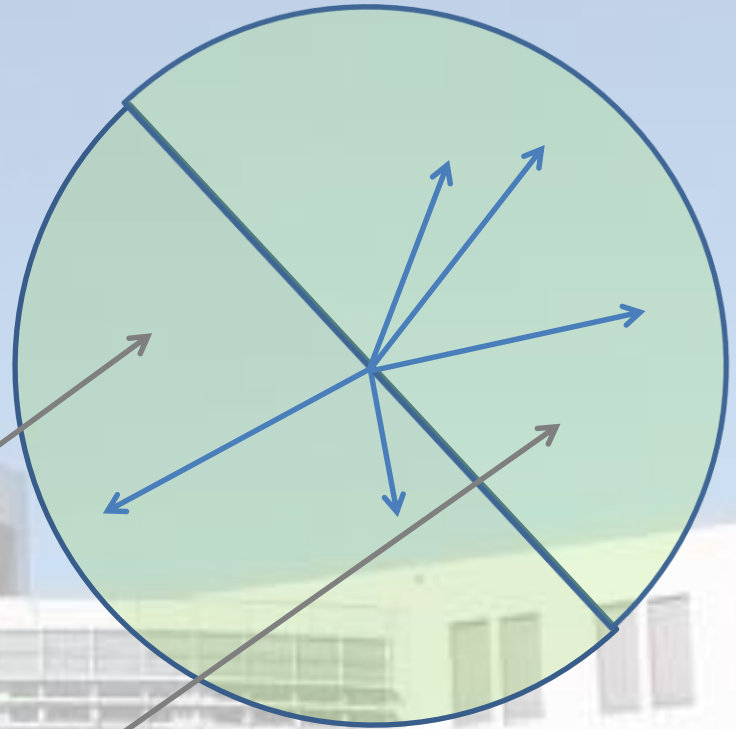
- Find transverse thrust axis, identify the two hemispheres making it up
- Look in hemisphere library for two SIMILAR hemispheres



Mixing procedure - 2

2) Take again original sample: for each event

- Find transverse thrust axis, identify the two hemispheres making it up
- Look in hemisphere library for two SIMILAR hemispheres
- **Construct an artificial event with them**

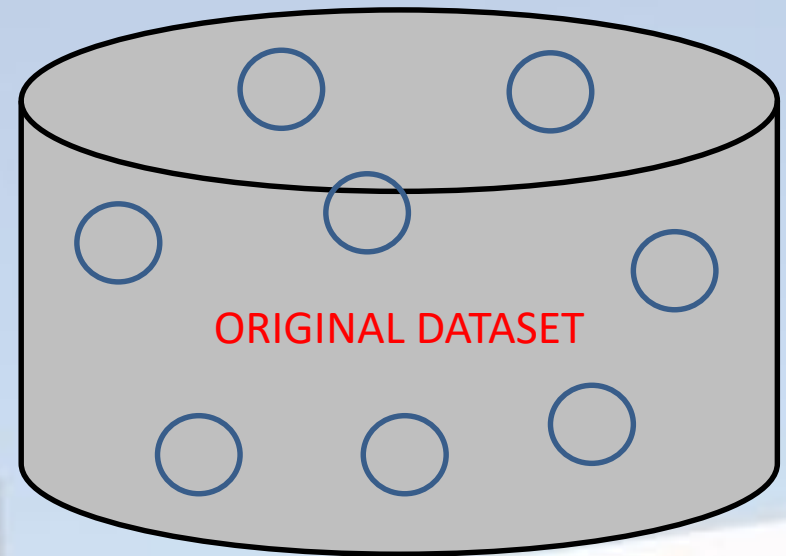


Mixing procedure - 2

2) Take again original sample: for each event

- Find transverse thrust axis, identify the two hemispheres making it up
- Look in hemisphere library for two SIMILAR hemispheres
- Construct an artificial event with them

The procedure creates an artificial dataset which can be used for modeling purposes

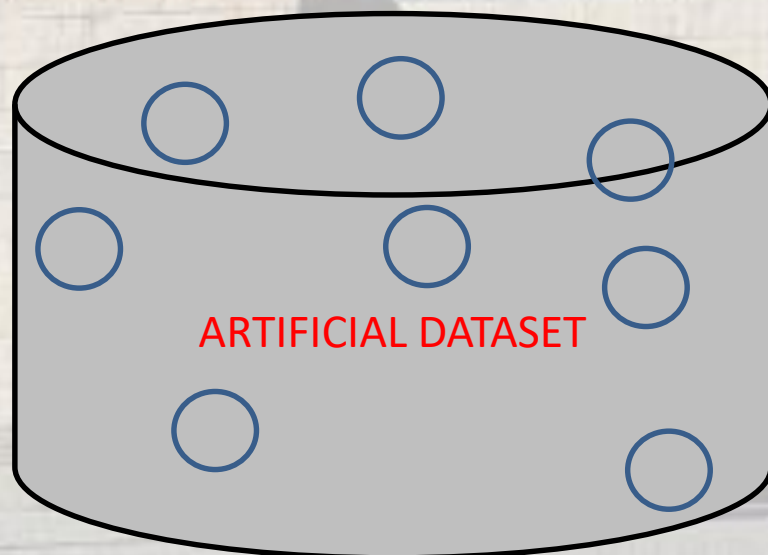


Hemisphere similarity criteria :

- Number of jets (req. equal)
- Number of b-tags (req. equal)
- Thrust
- Thrust minor
- Hemisphere mass
- Sum of jets p_z components

The 4 continuous variables are used to define a **kNN distance** which yields the similarity measure:

$$D(1p)^2 = \frac{(T(h_1) - T(h_p))^2}{V_T} + \frac{(M(h_1) - M(h_p))^2}{V_M} + \frac{(|P_z(h_1)| - |P_z(h_p)|)^2}{V_{P_z}} + \frac{(T_a(h_1) - T_a(h_p))^2}{V_{T_a}}$$



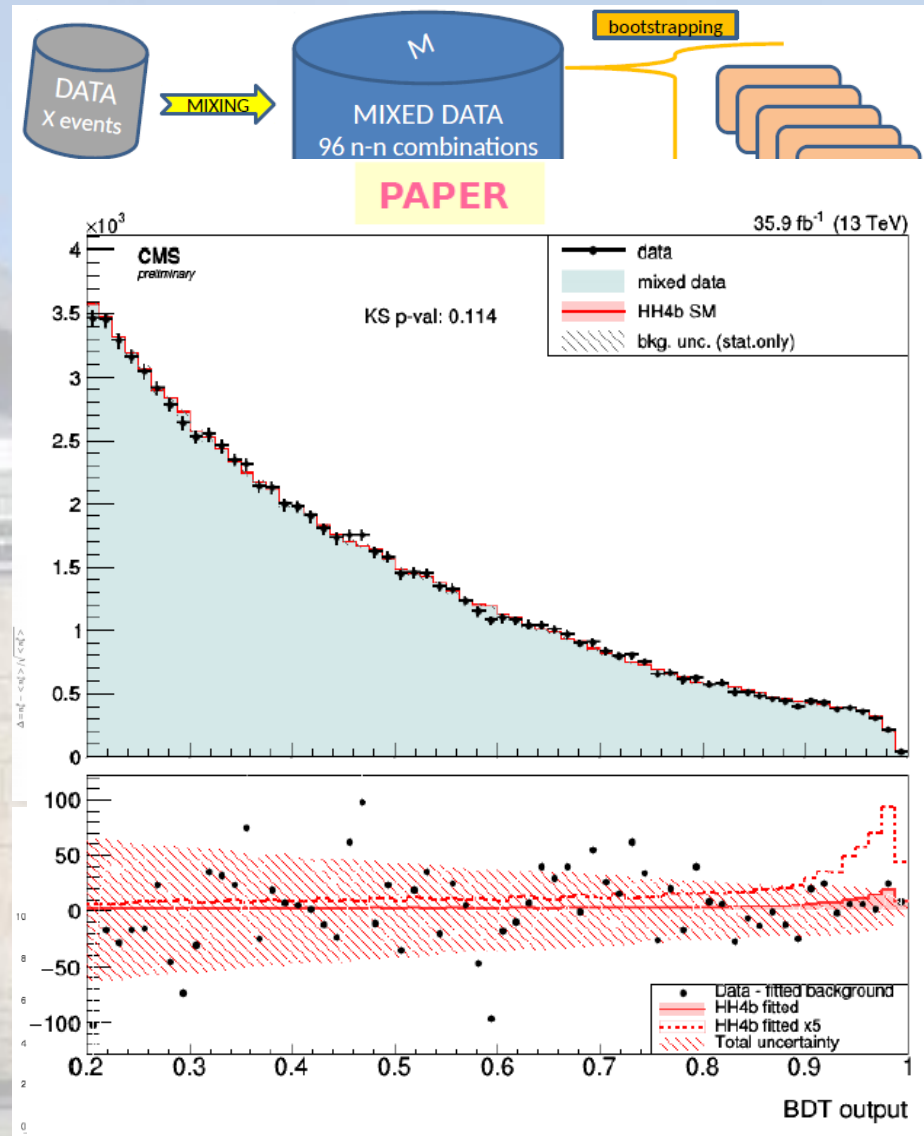
Modeling the target function

A residual bias in the modeling of the BDT distribution was suspected by studying the modeling agreement in control samples

It was then estimated by bootstrap resampling of the same data sample

This allowed to extract a bias function, which improved the model accuracy significantly

A test in a separate control sample confirmed the improvement



Jackknife resampling

The jackknife is called after the large pocket knife one brings around in an excursion – it is indeed a versatile statistics tool.



Again the most common use is determining the bias and variance of an estimator. One works out n different estimates of the estimator by leaving out in turn each one of the n i.i.d. data points $\{x_1 \dots x_n\}$.

$$\bar{x}_i = \frac{1}{n-1} \sum_{j=1, j \neq i}^n x_j, \quad i = 1, \dots, n$$

One may then get e.g. an estimate of the variance of the mean:

$$\text{Var}(\bar{x}) = \frac{n-1}{n} \sum_{i=1}^n (\bar{x}_i - \bar{x})^2$$

Given an estimator θ , we **may also compute its bias in a similar way**: if the jackknife average is

$$\hat{\theta}_{(.)} = \frac{1}{n} \sum_{i=1}^n \hat{\theta}_{(i)} \quad \text{where } \hat{\theta}_{(i)} \text{ is the estimate obtained by leaving out the } i\text{-th value,}$$

then the bias on the estimator is

$$\widehat{\text{Bias}}_{(\theta)} = (n-1)(\hat{\theta}_{(.)} - \hat{\theta})$$

What is better between bootstrap and jackknife? it depends on the problem! For smallish N jackknife is faster; however the bootstrap uses more information with non-linear estimators.

THE DATA



The data

A set of multi-dimensional data is made of N individual **events** (AKA cases or examples), made up each of D variables (or **features**, or attributes, or predictors)

We can think of our data as a table, where each column is an observed feature, each line a different event: we can organize them in a **$N \times D$ matrix**

In physics and astrophysics, typically N is large and D is small: these data are called "**tall**"

In other disciplines one instead frequently encounters "**wide**" data, with few examples and many features: e.g. in DNA testing one may have thousands of gene sequences

The distinction is useful as different ML algorithms apply more successfully to the analysis of data depending on their shape; wide data is often problematic

- BDTs can handle wide data just as well as tall data
- resampling techniques may help with wide data
- but kNN and other simple methods do not work well with wide data
- also, linear discriminant analysis encounters difficulties with wide data as inversion matrix gets singular for $N < D$

Data preprocessing

An important part of data analysis concerns its preprocessing – a sometimes annoying chore, which forces you to fiddle with non-high-tech tools

You have to preprocess your data if

- some of the features are **missing** in a few of the events
- there are **outliers** that spoil the accuracy of your model
- some of the features are **categorical**

A preprocessing is not mandatory:

- you can **remove incomplete events** (but see below)
- you may decide to **ignore the effect of outliers**
- you may split the data in subsets with homogeneous categories and proceed with each, or **use methods that are robust** to their existence

In general, the proper handling of missing data, outliers, and categorical features can significantly improve your model

A common mistake to be avoided: ignore preprocessing, rush to build the most performant super-duper classifier or regressor. Later, find out that the performance loss you took by ignoring the former step is not made up by the care you put on the latter

Categories of missing values

There are three categories of data with missing values. Understanding to which case your data belongs is important

- **MCAR:** missing completely at random
 - e.g. you are combining energy and timing observations of GRBs from two telescopes T1 and T2, and T2 is inactive on Sundays → there is no correlation between the lack of T2-related features of GRB candidate events and the GRB-related parameters of interest
- **MAR:** missing at random
 - e.g. T2 is overall less sensitive to GRBs in a specific area of the sky, so we can predict with a probabilistic model the lack of T2 data from T1 readings

In general, for MCAR and MAR the mechanism that produces missing data can be ignorable, if it does not interact with the mechanism by which data are generated

- **MNR:** missing not at random
 - e.g. T2 is less sensitive in a certain energy range, where it may fail to read a signal above instrumental noise: the lack of data depends on the properties (detectable energy) of the missing data themselves
 - the mechanism producing incomplete data affects the observed population!

Should I worry about missing values?

You should apply hypothesis testing to determine what is the situation of your missing values

- may compare the marginal distribution of each non-missing feature in data that has/lacks values of each one of features at a time
 - but beware of multiple testing: you will likely find that there is a significant difference in marginals for one feature, if you had many to test → apply usual Bonferroni-type corrections to handle this
- or may do some global MVA test (e.g. assuming multivariate normality) between complete and incomplete data in the considered feature

In general, **insight in the way the feature went missing is very important to understand whether you are in MCAR/MAR or MNAR**. You can often ignore the issue of missing data in the former case, you must address it in the latter.

When it matters and why

Narsky 2014 describes the formal argument defining why we can ignore the presence of missing values in training data, in the case of MCAR or MAR.

The probability to observe data X with missing values described by an indicator function I can be written $p(x, i | \theta, \phi)$. x is D -dimensional and i_n is zero if the n -th component x_n is missing; θ are the parameters of interest and ϕ track the mechanism whereby values are missing

We can factorize p if the θ are distinct from the ϕ :

$$p(x|i) = p(x|\theta)p(i|x, \phi)$$

An observation x belonging to X can have missing or non missing data, $x \in \{X_{obs}, X_{mis}\}$; the probability of observing it can then be written

$$p(x_{obs}, i; \theta, \phi) = \int p(x_{obs}, x_{mis}; \theta) p(i|x_{obs}, x_{mis}; \phi) dx_{mis}$$

MCAR and MAR probability densities

(From the previous slide):

$$p(x_{obs}, i; \theta, \varphi) = \int p(x_{obs}, x_{mis}; \theta) p(i|x_{obs}, x_{mis}; \varphi) dx_{mis}$$

For MCAR data the mechanism of missing data is independent on observed and missed values, in which case $p(i|x_{obs}, x_{mis}; \varphi) = p(i, \varphi)$
so the integral simplifies to $p(x_{obs}, i; \theta, \varphi) = p(x_{obs}, \theta) p(i; \varphi)$

For MAR data the missing data mechanism is explained by the observed data,

$$p(i|x_{obs}, x_{mis}; \varphi) = p(i|x_{obs}, \varphi)$$

so we find

$$p(x_{obs}, i; \theta, \varphi) = p(x_{obs}, \theta) p(i|x_{obs}, \varphi)$$

→ In both MCAR and MAR, we see that the **observed data carries sufficient information on the parameter of interest.**

This is not the case of MNAR, when the pattern of missing data may affect the estimates one produces with the unprocessed data.

Listwise deletion and imputation

If your data are MCAR or MAR, you may (or should, depending on what you will do next) manipulate them

Listwise deletion is the removal of events containing missing features.

Excluding MNR, this **introduces no bias, but reduces power**

Imputation is the procedure of inserting values for the missing features. This avoids the defaulting result of elimination of an event from the data set (commonly applied by statistical packages).

The value is usually an **estimate** of the missing entry **dependent** (by regression, e.g. using a fit) **or not** (e.g. in mean substitution, by averaging the available entries) **on the other features of the same event**. Each choice has drawbacks.

Another common practice is to use the so-called **hot deck imputation**: one sorts the data using some of the features, then replaces the missing entry with that of the event that precedes it. A better idea (similar to regression) may be to use some form of kNN, averaging elements close to the one with the missing entry

More preprocessing: Centering, scaling, reflections

Some ML algorithms benefit from preprocessing of the features by **standardization procedures**, operated with univariate transforms

Centering:

Centering consists in subtracting means off the marginals.

If $X = \{x_1, \dots, x_N\}$ are the relevant coordinates of your N data examples, centering produces

$$X' = \{x_1 - \mu^*, \dots, x_N - \mu^*\},$$

where μ^* is the observed mean.

Note that since $\mu^* \neq \mu$, $E[X] \neq 0$ in general.

Scaling and reflections

Scaling: Multiplying each feature by a positive constant

Reflection: multiplication by negative constant

The main application of scaling is to force all features to have the same variance (usually chosen to be 1.0 \rightarrow standardization).

By scaling one can "remove" the dimensional character of different features, to facilitate the interpretation of the resulting Euclidean distance

When should you use these preprocessing steps?

- Centering is **useless for decision trees, random forests, BDTs**
- Centering can instead **improve training stability for neural networks** (when applied with scaling)
- **Scaling is useful in kNN applications**, which are instead insensitive to centering or reflection. The same is true for distance-based methods

Data with unbalanced classes

In classification applications, it is usually the case that the amount of training data for each class differs. Most algorithms confronted with unbalanced training samples will learn more about one class than the other → smaller classification error for the oversampled class

To handle this, one can **use Bayes theorem**, obtaining from the learned $p(x|c_i)$ a posterior probability $p(c_i|x)$ by accounting for the class population in the training:

$$p(c_1|x) = \frac{p(x|c_1)p(c_1)}{p(x|c_1)p(c_1)+p(x|c_2)p(c_2)}$$

This procedure is also known as "weighting" the training data.

If one wants to mix data in equal proportions, one may **undersample the majority class** or **oversample the minority class**. The former reduces CPU but also information; the latter is effective IF one does it by synthesizing new observations. This can be done e.g. using local density estimates (e.g. kNN)

THE MODEL



The mathematical model

Machine learning relies on building a model of your data: a mathematical characterization of the studied system, in probabilistic terms

To build a model, you need

- to understand the structure of your data
- to clarify the problem you want to solve: e.g. regression, classification, clustering, ...

Based on the above inputs, you may choose the most appropriate ML method

E.g.

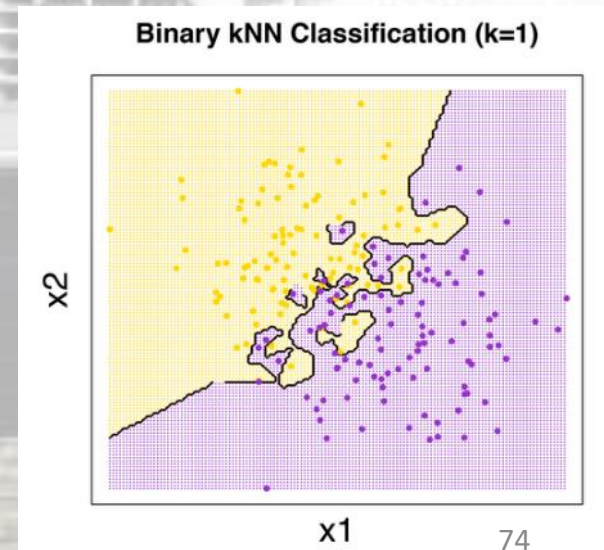
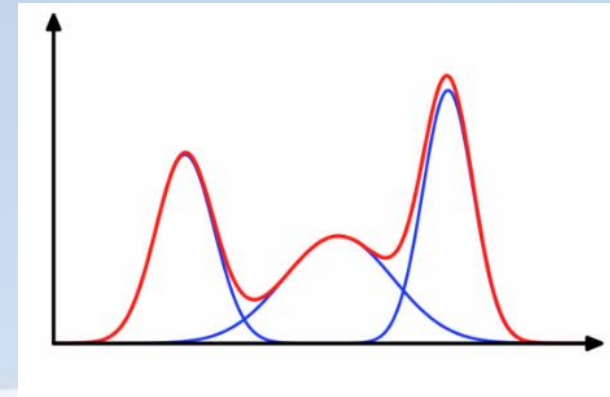
- for a low-D regression task, you might want to specify a family of parametric functions, and proceed to find the best choice of parameters
- for a complex classification task, you might focus on designing a proper architecture for a DNN

The method will **learn the model parameters** from available training data

The learned model will allow to **make predictions or inference** on previously unseen data

Parametric or non-parametric?

- **Parametric models** are fully defined by a function, with a fixed set of parameters
 $f(x;\theta) = \dots$
 - They can be a good choice when you want to retain insight in what is learned
 - They involve an **assumption** on the behaviour of the data \rightarrow bias
- **Non-parametric models** do not have a fixed set of parameters, and they may become arbitrarily more complex as you let them learn more information from training data
 - Assumption free (almost)
 - Higher variance, less bias



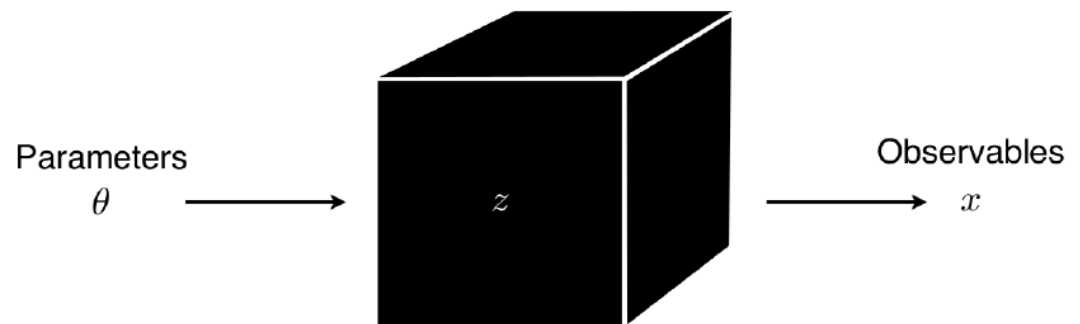
Going forward or backward

In most applications of interest to fundamental science we observe natural phenomena and try to decrypt them by constructing a model, then doing inference on its parameters

We are helped by simulations that, based on the model, allow to artificially generate observations based on chosen parameter values.

Problem: the generative process is usually affected by stochasticity \rightarrow cannot be reversed trivially, as same θ lead to different x at random

We have no analytic likelihood!
The inference process becomes intractable, forcing us to use work-arounds.



Prediction (simulation):

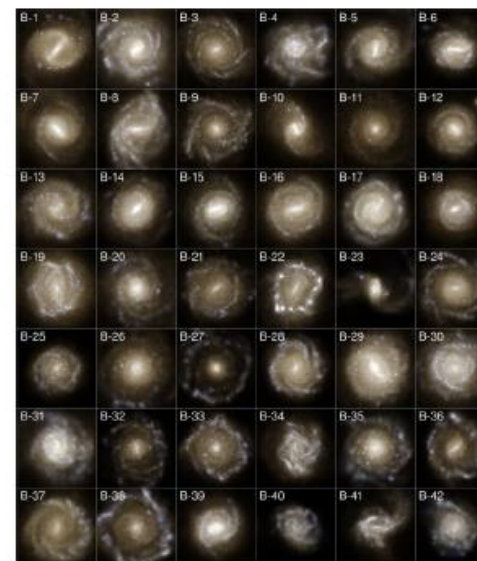
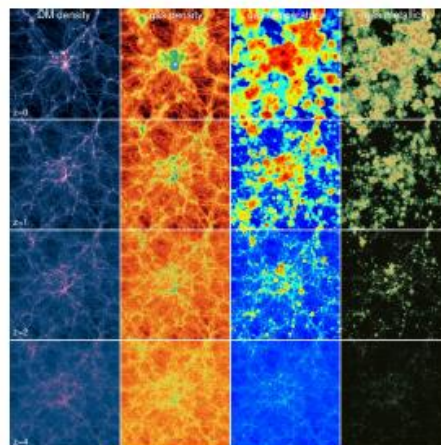
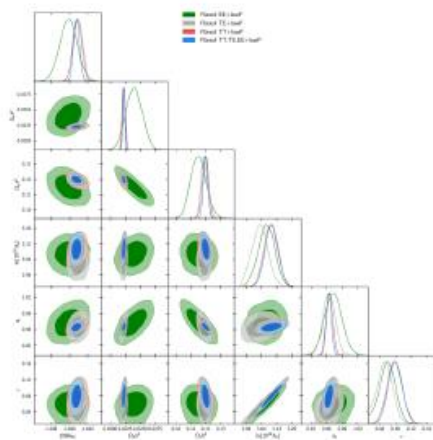
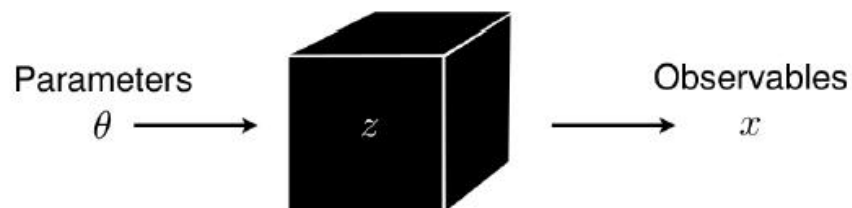
- Well-understood mechanistic model
- Simulator can generate samples

Inference:

- Likelihood function $p(x|\theta)$ is intractable
- Goal: estimator $\hat{p}(x|\theta)$

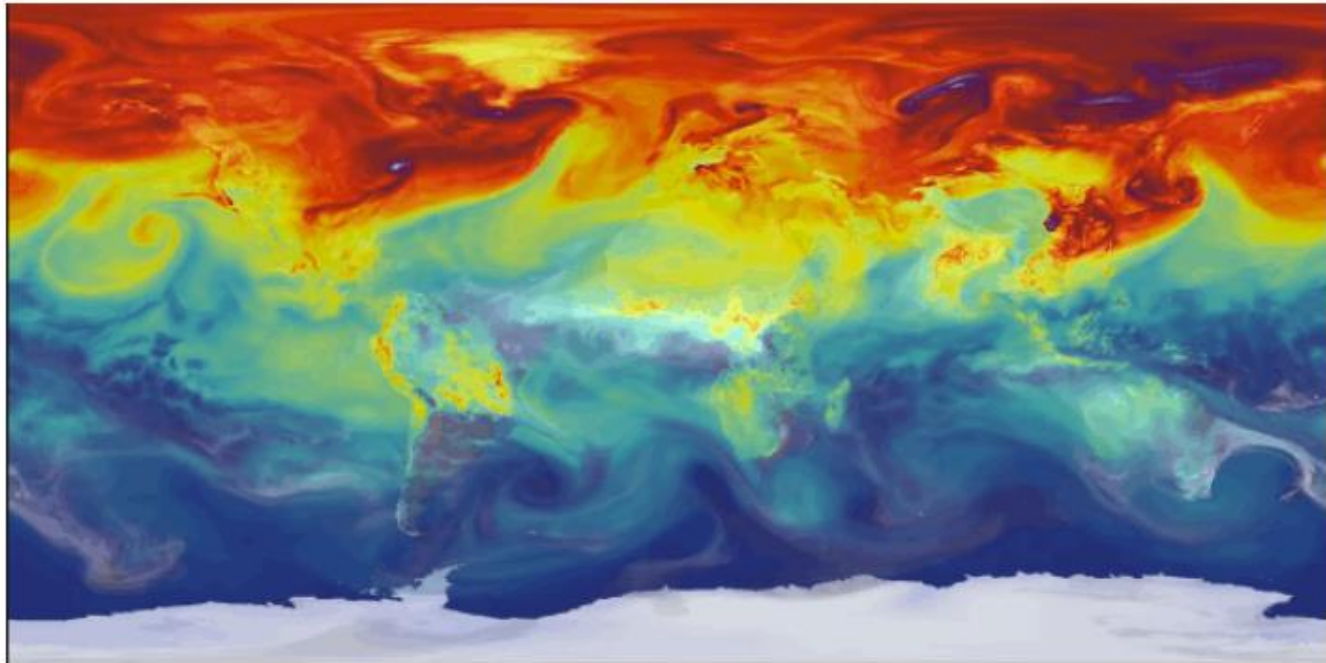
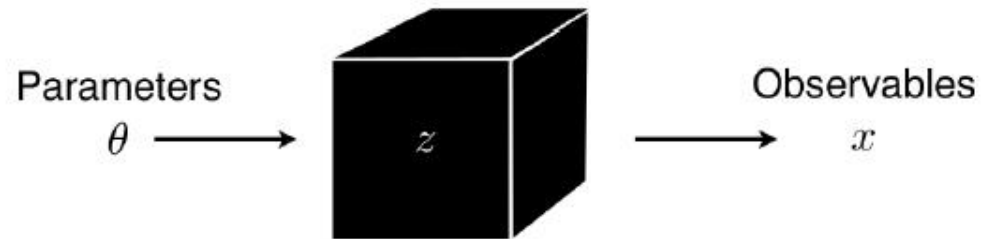
Cases when we can only go forward / 1

Cosmological N-body simulations



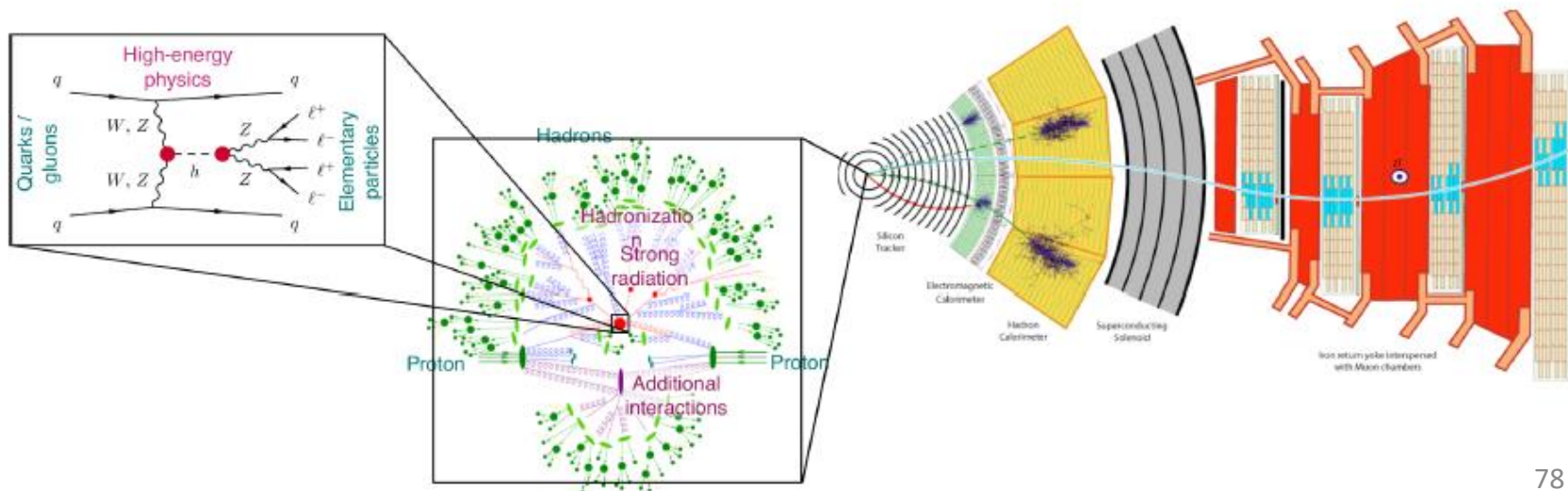
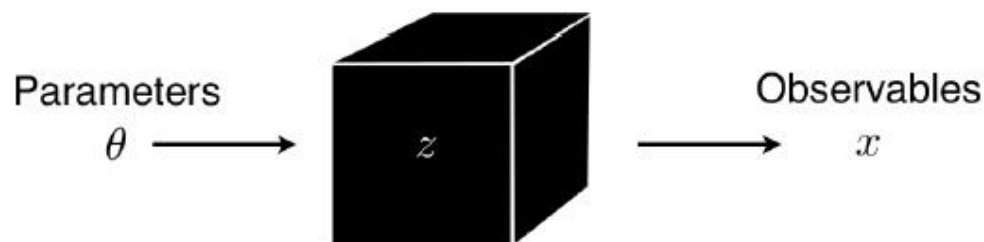
Cases when we can only go forward / 2

Climatology



Cases when we can only go forward / 3

Particle physics



How to deal with this?

We resort to the construction of **proper summary statistics**, which have a much lower dimensionality than the observed data.

This in general throws away information, unless T is sufficient

Statistical sufficiency: the definition stems from the factorization theorem of Fisher-Neyman.

T is sufficient for X if

$$f(X|\theta) = h(X) g(T(X)|\theta)$$

in other words, *all* the information on the unknown parameters θ provided by data X is accessible through the function T

The problem is that **finding sufficient statistics is very hard**, when at all possible. Machine learning methods are however capable of extracting summaries from the data that offer useful surrogates

Generative and Discriminative models

In the light of the ill-defined nature of the PDFs we deal with in our physics problems, and bearing in mind the Neyman-Pearson lemma, the goal then becomes: **estimate** $p(x|S)$ and $p(x|B)$, and then **construct their ratio $r(x)$**

Many MVA algorithms do precisely that: they approximate multivariate densities. Among them are kernel density estimators, nearest neighbors...

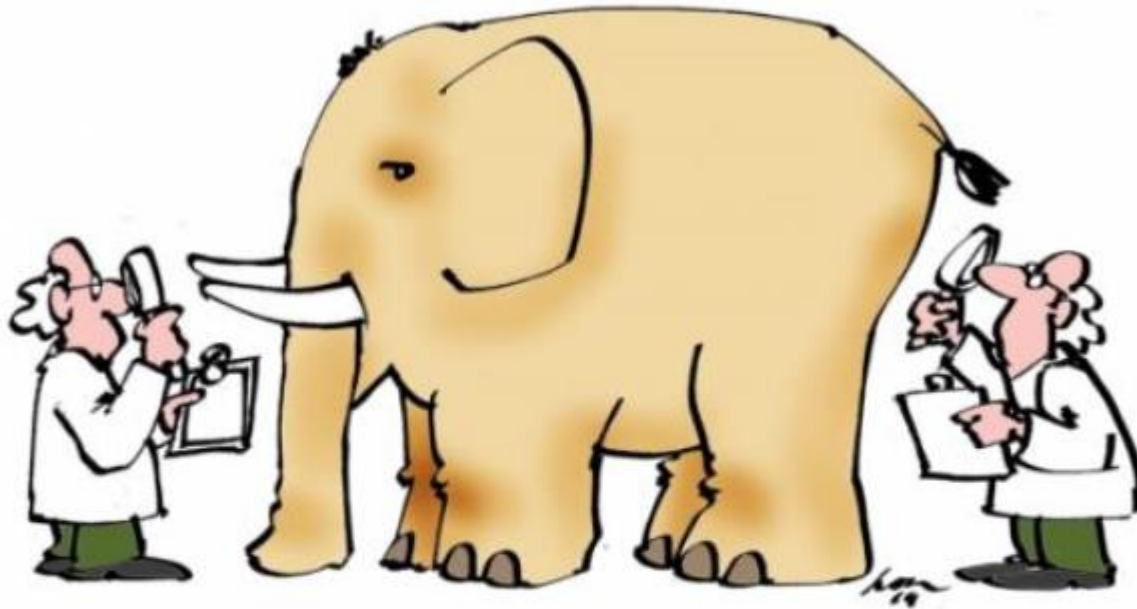
→ **generative algorithms**

or one may **approximate** the “likelihood ratio”, or a monotonous function of it, directly: finding hyperplanes in the observation space where $r(x)$ is constant allows then to separate S from B pseudo-optimally

There is a bunch of ways to learn a monotonous function of the LR: linear discriminators, BDTs, neural networks. These are globally called

→ **discriminative algorithms**

LECTURE 1 CONCLUSIONS



"Statistics: The only science that enables different experts using the same figures to draw different conclusions."

Evan Esar

Conclusions for lecture 1

- Machine learning has a large overlap with statistical learning, which has been around for much longer.
 - Emphasis is on **large-scale applications**, and on **prediction accuracy** (as opposed to emphasis on models and their uncertainty)
- Density estimation is an important ingredient of many ML methods
 - especially when they require pdfs as inputs
- Data preprocessing may be an essential step that pays dividends in performance later on
- In fundamental science we often deal with the lack of analytic likelihoods
 - ML methods can provide **effective approximations to summary statistics** to carry out the inference work