

# (Short introduction to) machine learning (3/3)

---

Diego Tonelli (INFN Trieste)  
[diego.tonelli@cern.ch](mailto:diego.tonelli@cern.ch)

*CERN-Fermilab HCP Summer School*  
*CERN Aug 30, 2017*

# What?

---

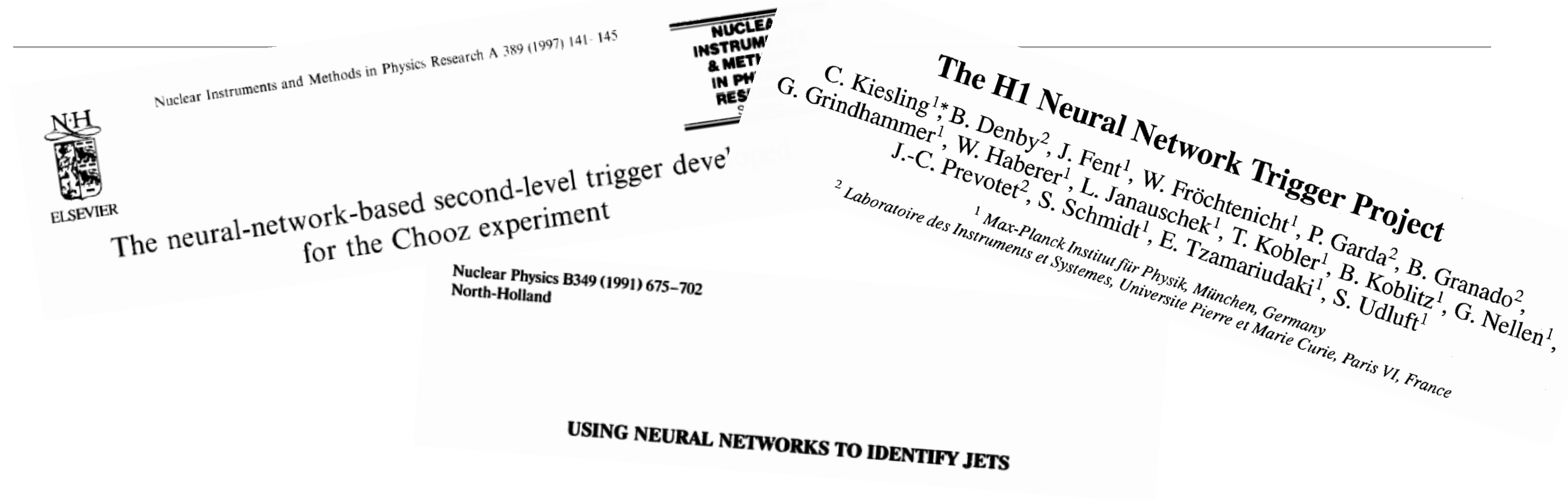
Giving computers the ability to learn without explicitly programming them

Use statistics, mathematics, and computer science to determine mathematical models, learned from data, that capture the patterns and relationships between the features of the data.

Formulated around the 1950ies.

Currently rapidly evolving, driven by many relevant applications in language processing, speech and handwriting recognition, vision, computer vision, fraud detection, financial markets analysis, search engines, spam/virus detection, medical diagnosis, robotics, automation, advertising, data science.

# In HEP



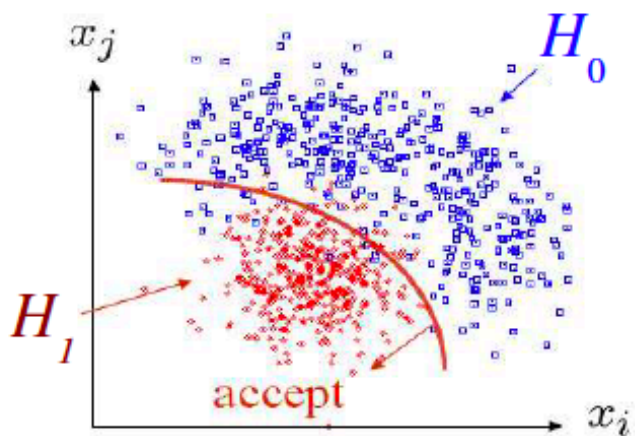
First, pioneering applications appeared in the 90ies

Became more popular in the 2000's (LEP/Tevatron) until today's boom: classify "signals" from "backgrounds" both online and offline, improve reconstruction of heavy particles from incomplete decay products, etc..

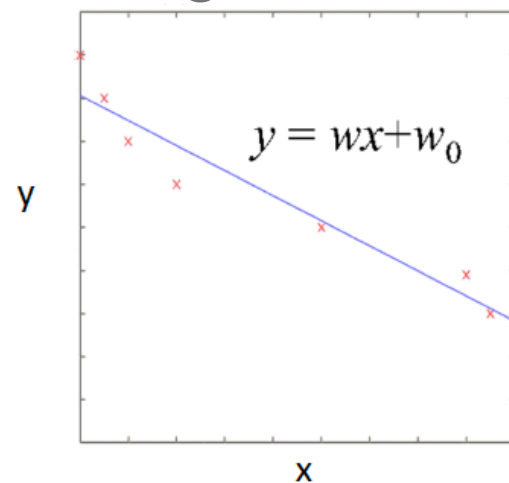
# The model

Central assumption: observed data are generated from the probabilistic distribution  $p(x|m)$ , the “model”, which is a mathematical description of the system of interest. The model depends on the data and on what we want to accomplish, e.g.:

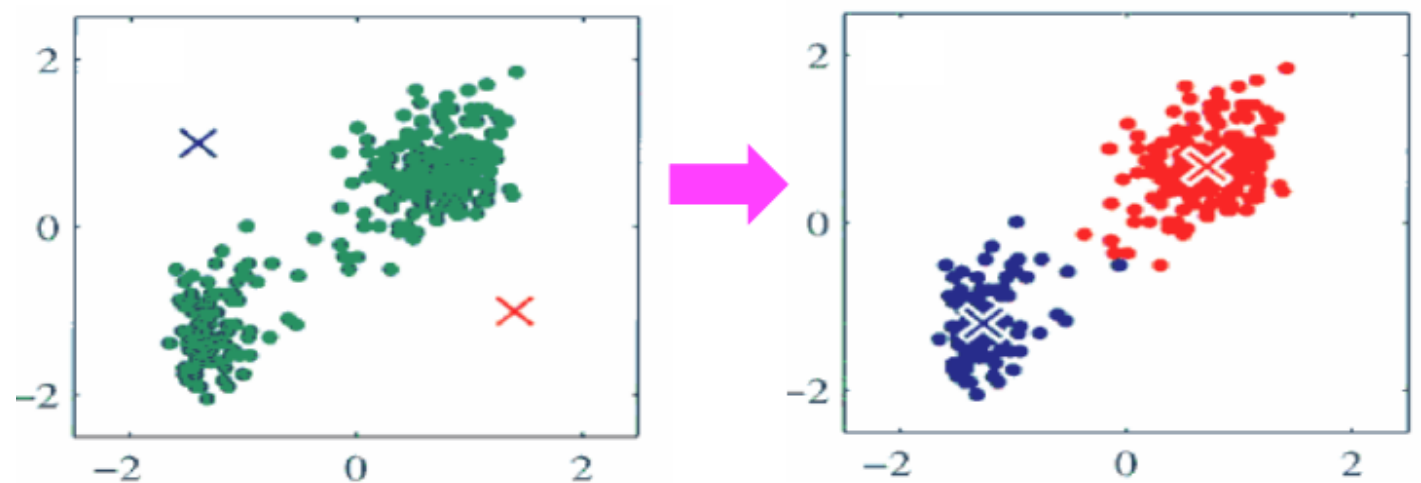
Classification



Regression



Clustering



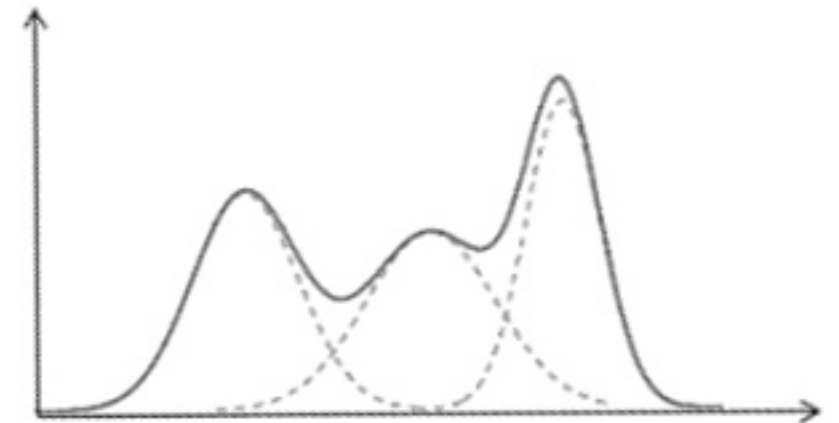
An approximation of the model is **learned** by using the information associated with input data. It is then used to identify the relevant properties of the system of interest and **predict** new data points.



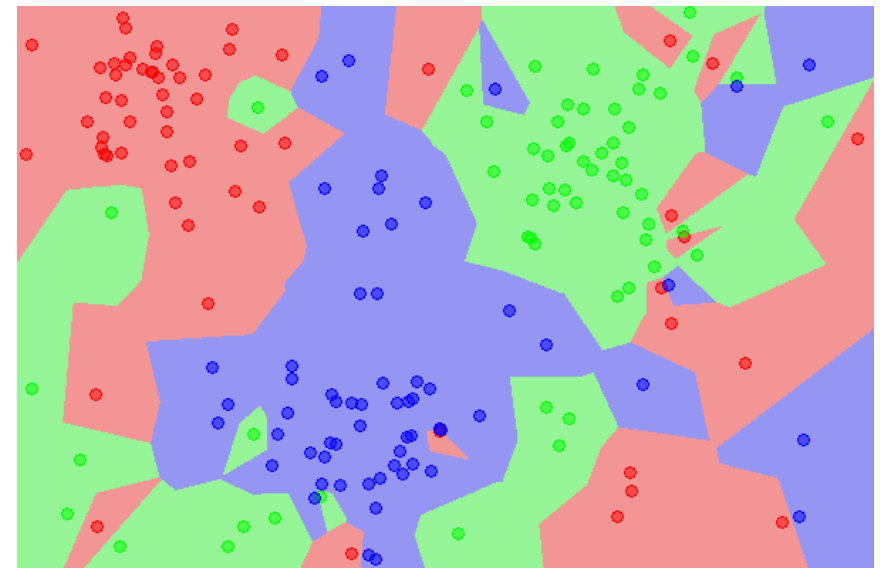
# Parametric vs non-parametric

---

Parametric models — fully specified by a number of parameters that does not grow with the size of the data set used to extract them. E.g, Gaussian mixture models



Non-parametric models — may grow in complexity with more data. E.g., a model that predicts the location of a data point in the feature space using the nearest known set of data points



# Supervised learning

---

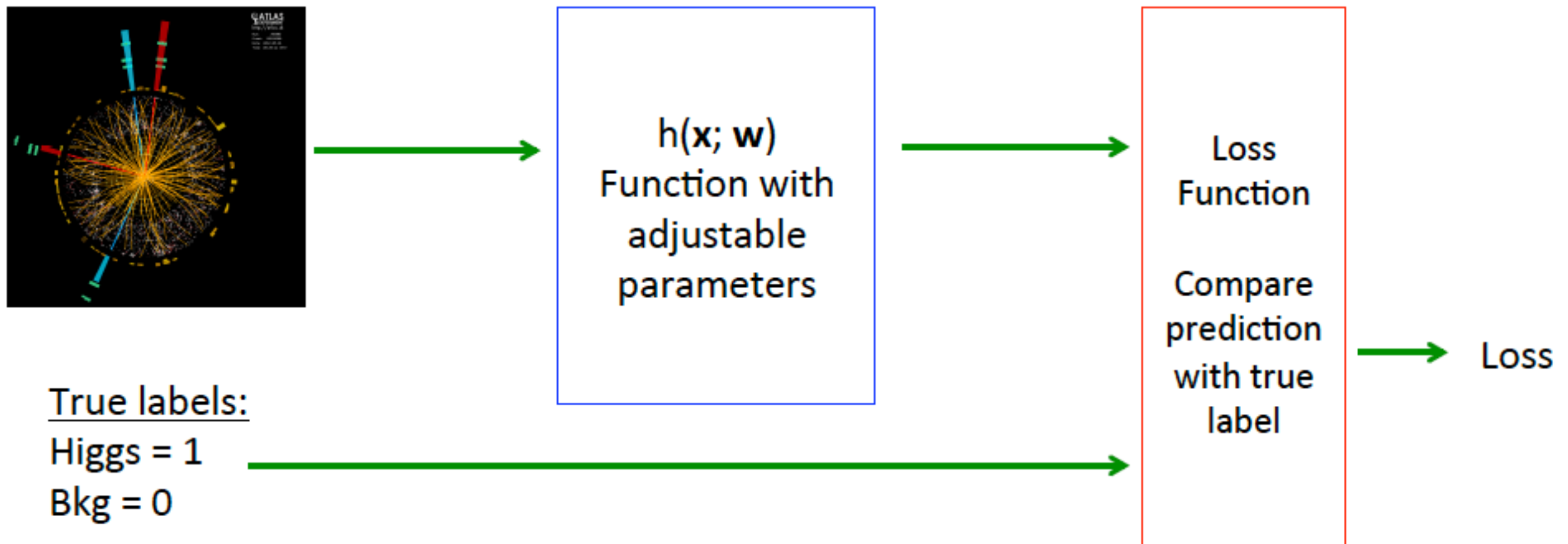
Define a model  $h(\vec{x}|\vec{w})$  flexible enough to be able to adapt to the problem at hand (but not more flexible than that)

Feed a set of “training” data  $\vec{x}_t$  to the model so that it can “learn” (adjust its parameters) for modeling any new data optimally [for the task]: give it  $N$  example events, each associated with feature variables  $\vec{x}$  and the label (or target)  $y$ . This is the value of the quantity I want the model to predict — can be a class label (signal, background or pion, kaon) or a real number (electron energy..).

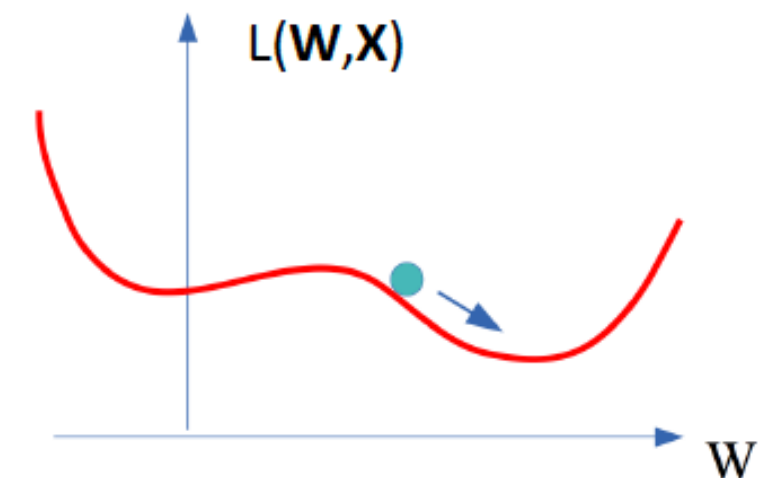
During learning iterate over the training data by adjusting the model-parameters  $w$  until a “distance” figure of merit that quantifies the difference of the model from the truth reaches a sufficiently low value. Define  $h(\vec{x}) = y$ .

Test performance on an independent labeled sample

# Supervised learning



- Design function with adjustable parameters
- Design a Loss function
- Find best parameters which minimize loss
  - Use a labeled *training-set* to compute loss
  - Adjust parameters to reduce loss function
  - Repeat until parameters stabilize
- Estimate final performance on *test-set*



[Kagan, Le Cun]

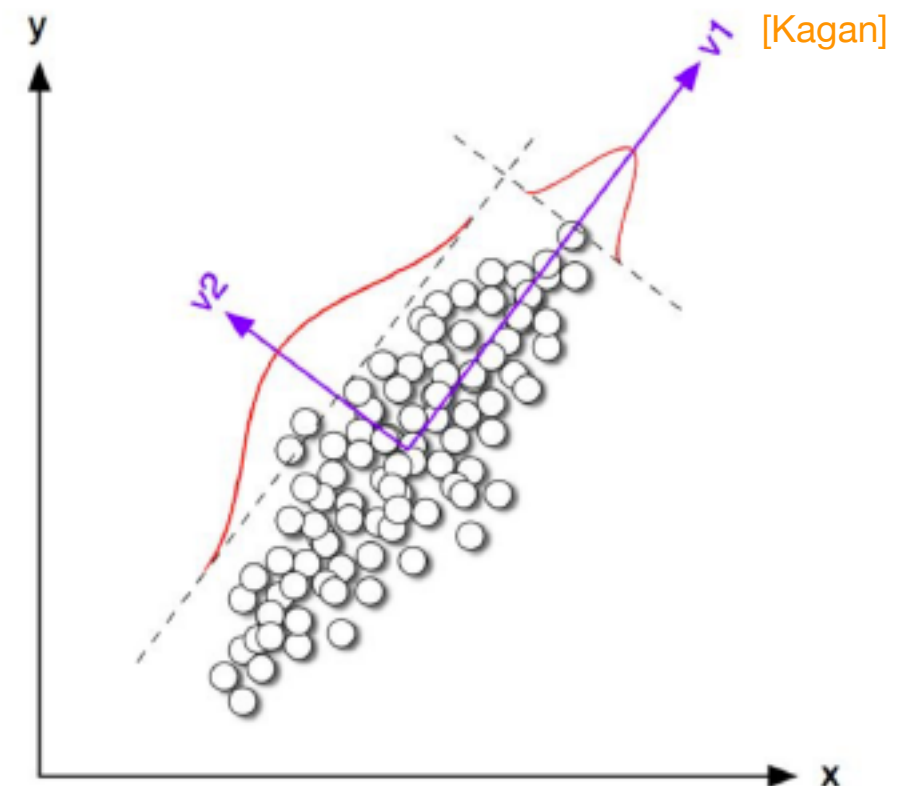
# Unsupervised learning

---

As before, but labels are not known. The task is to find structure/pattern in the data.

Clustering: partition data into subsets according to similarities in the feature space

Dimensionality reduction: find a lower-dimensional (simpler) representation of the data



# What follows

---

In HEP, ML approaches have been mostly applied to classification: signal/background, kaon/pion, photon/electron; quark-jet/gluon-jet, b-jet/ light-jet. Typically supervised due to availability of simulated and control samples

With the LHC, applications to a broader set of tasks are becoming popular (e.g., reweight multidimensional distributions to match to each other)

Our general discussion will be mostly restricted to supervised binary classification

# The classification task

In a sample of physics data, observe candidate “signal” events, contaminated by “background” events. Each is associated to a set  $\vec{x}$  of **variables (or features or predictors)** e.g.,

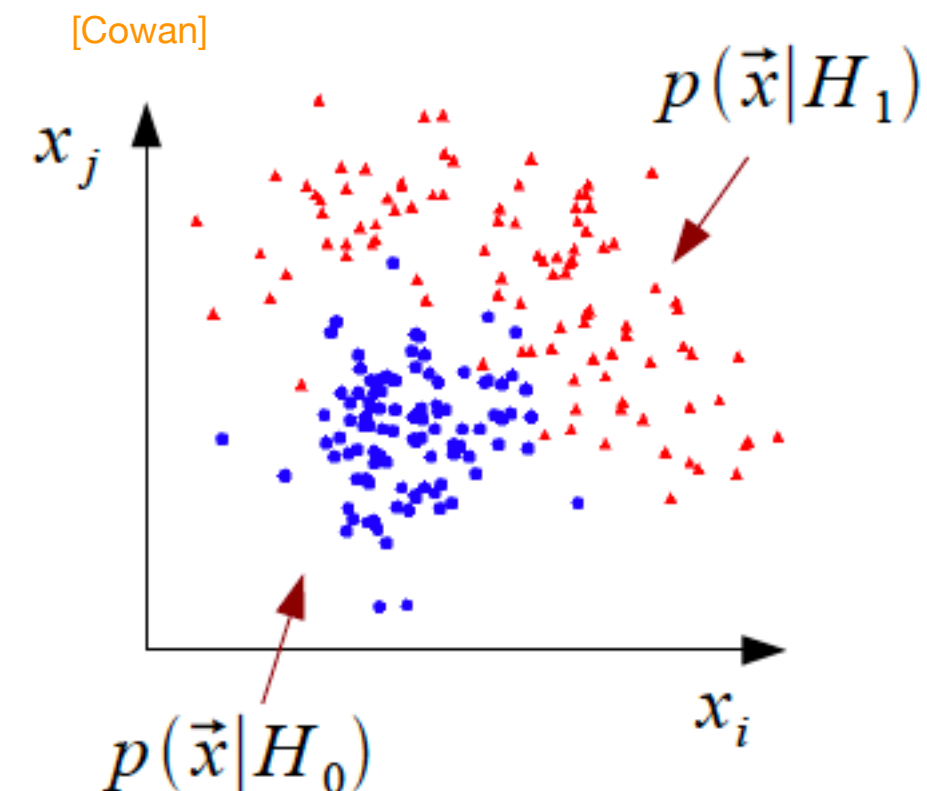
$x_1$  = transverse momentum

$x_2$  = displacement from collision point

$x_3 = \dots$

$x_n = \dots$

$\vec{x}$  is distributed according to an n-dimensional joint probability density  $p(\vec{x}|\mathbf{m})$ , which differs for signal ( $H_1$ ) and background candidates ( $H_0$ ).

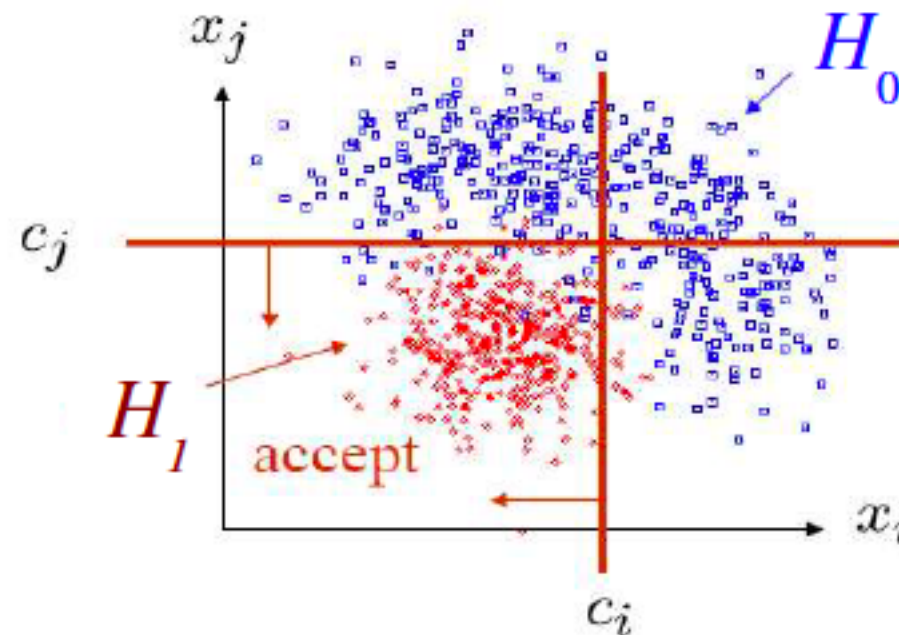


The goal is to classify the events within the signal or background categories.

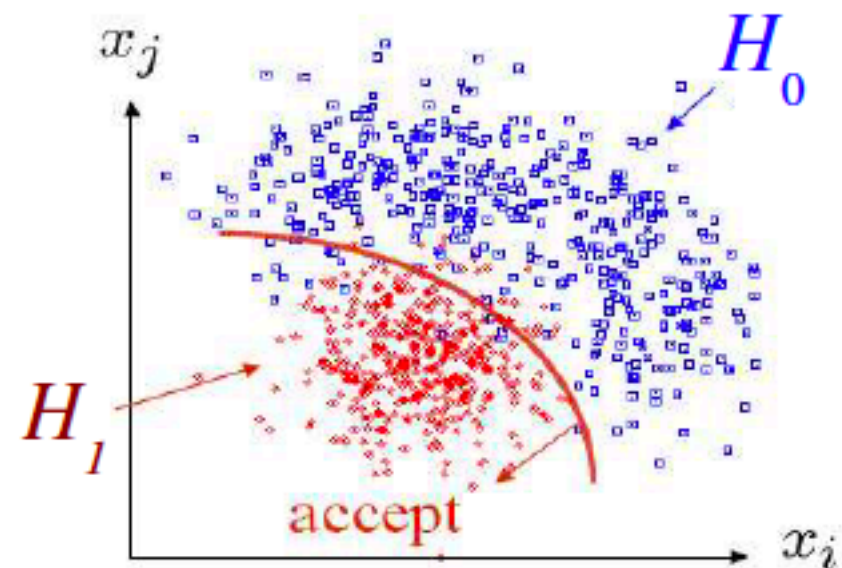
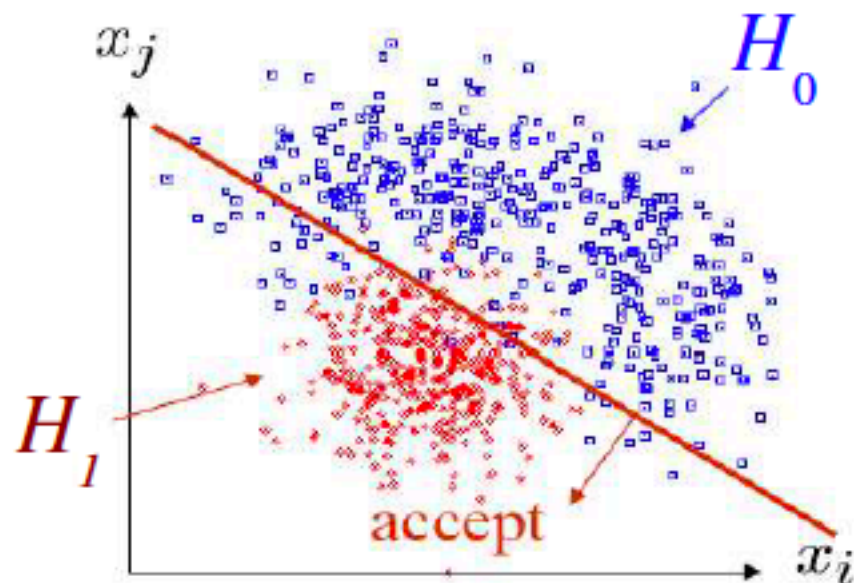
# Decision boundaries

Can do it with cuts

[Cowan]



Or identify some sort of decision boundary



[Cowan]

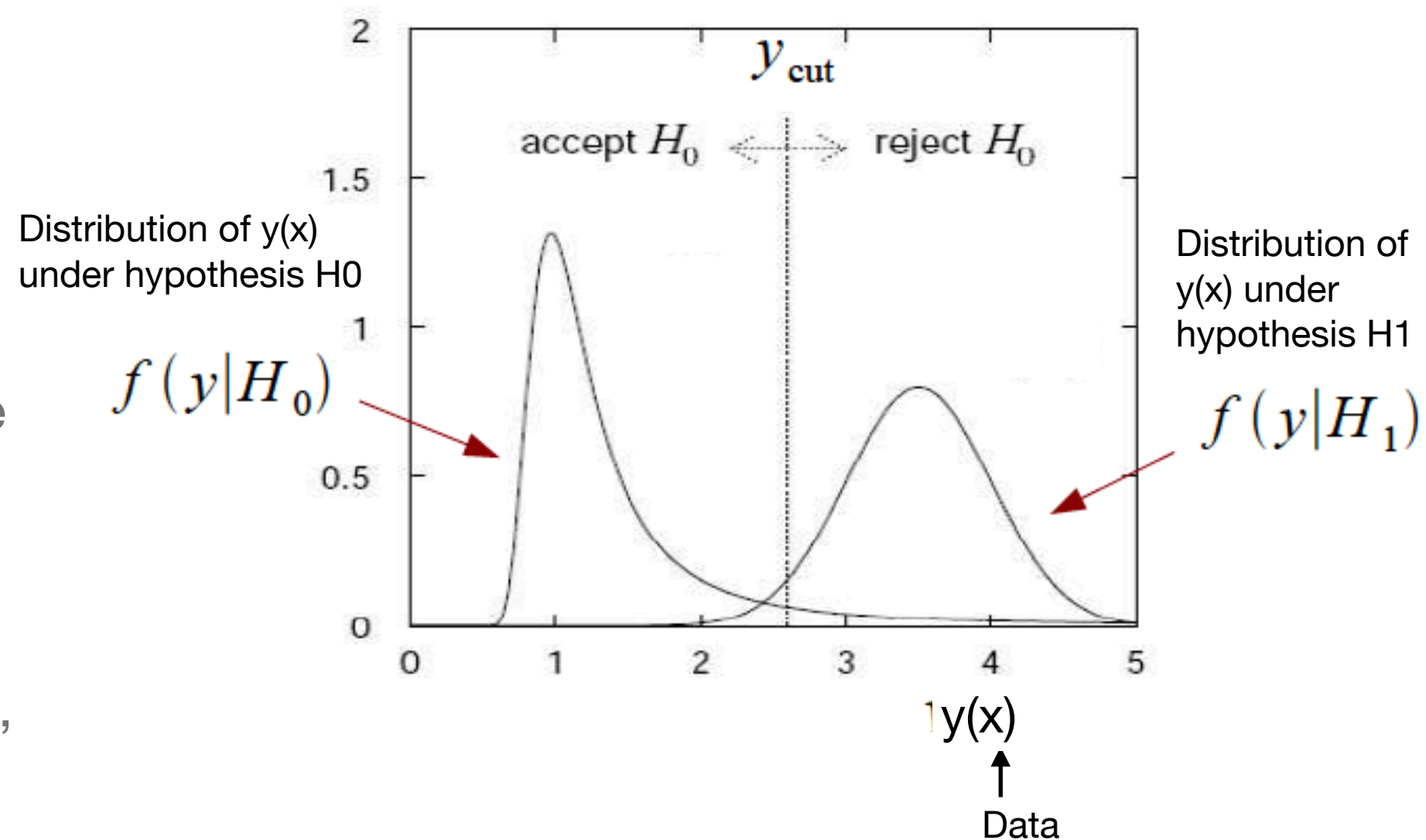


# Decision boundaries

Decision boundary — a function of the data that allows separation between classes. Surface in the  $n$ -dimensional space of the features.

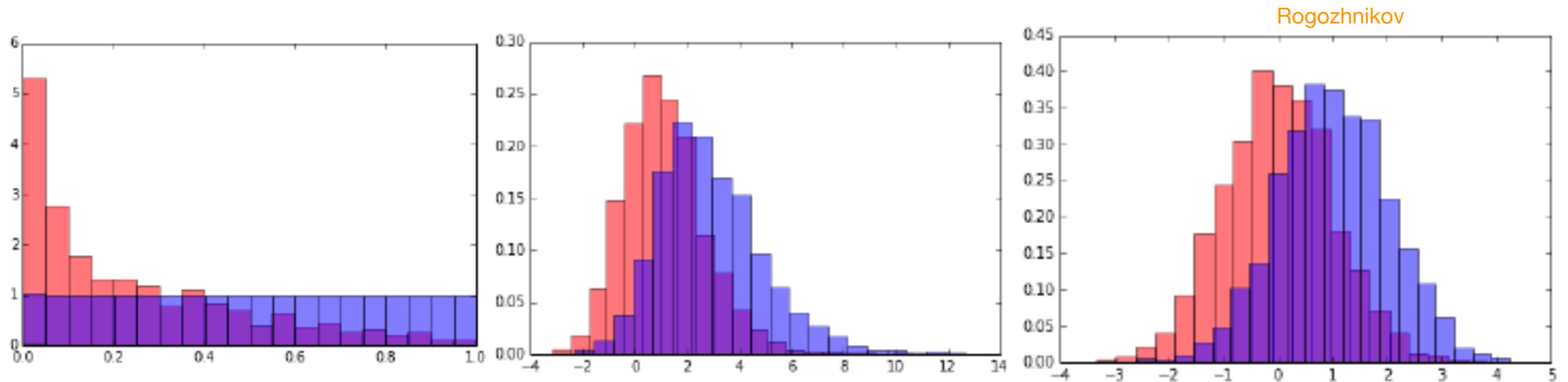
Can make  $y(\vec{x})$  as a scalar number and determine it in a way that its distributions for the signal and background samples are maximally separated.

With such a dimensionality reduction, a “cut” on  $y(\vec{x})$  offers a decision boundary





# Binary classification performance



Three classifiers separate “red” from “blue” classes of events.

Which one does it better?

# ROC

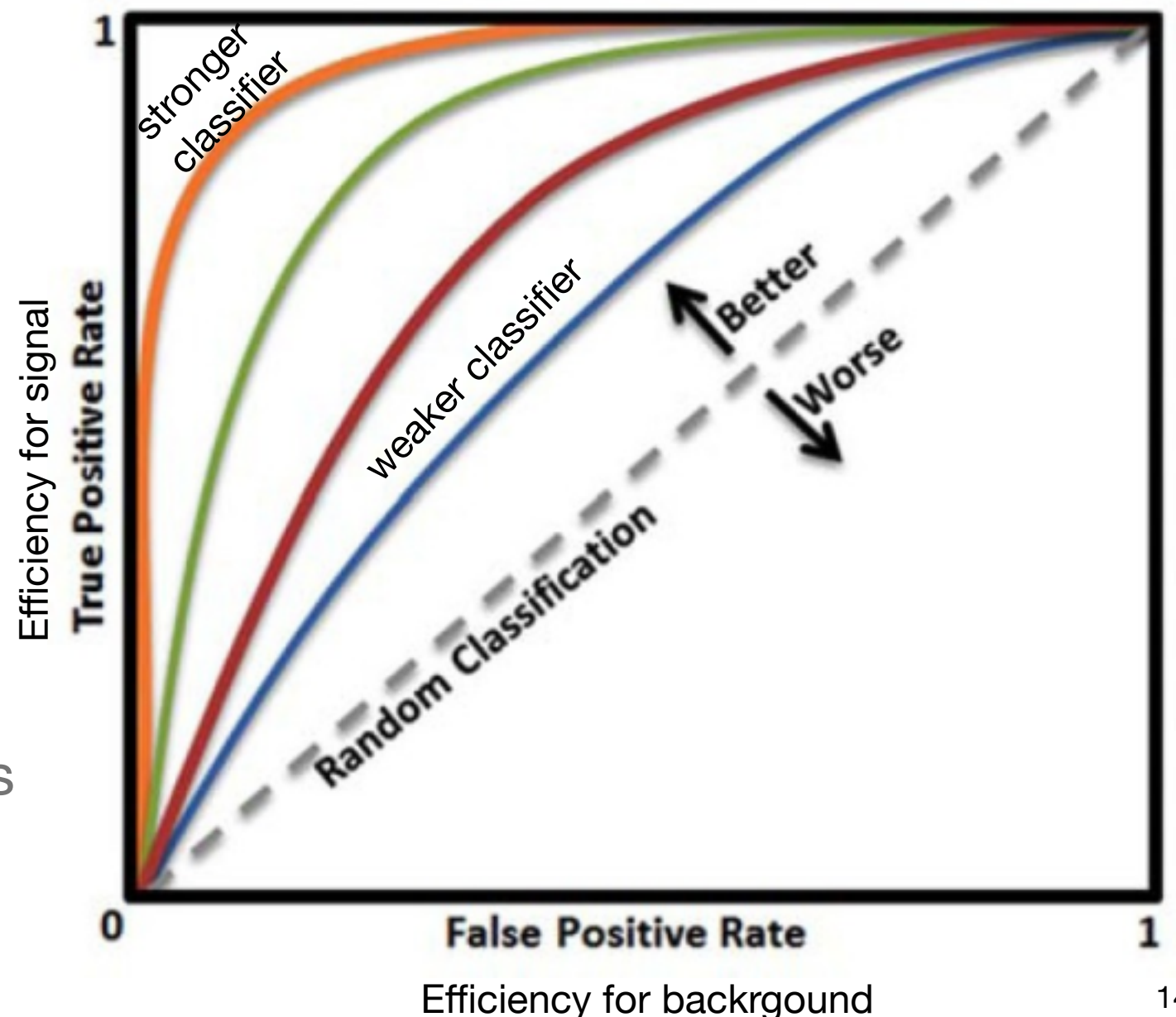
None: all have the same **receiver operating characteristic** (ROC) curve: signal classification efficiency vs background misclassification efficiency.

Standard measure of performance for binary classifiers.

Each point in the curve corresponds to a threshold in the classifier output.

Get as much top right as possible.

Is there any optimal variable that, given the information in data, allows separating two classes of events with minimum false positive rate at given true positive rate?

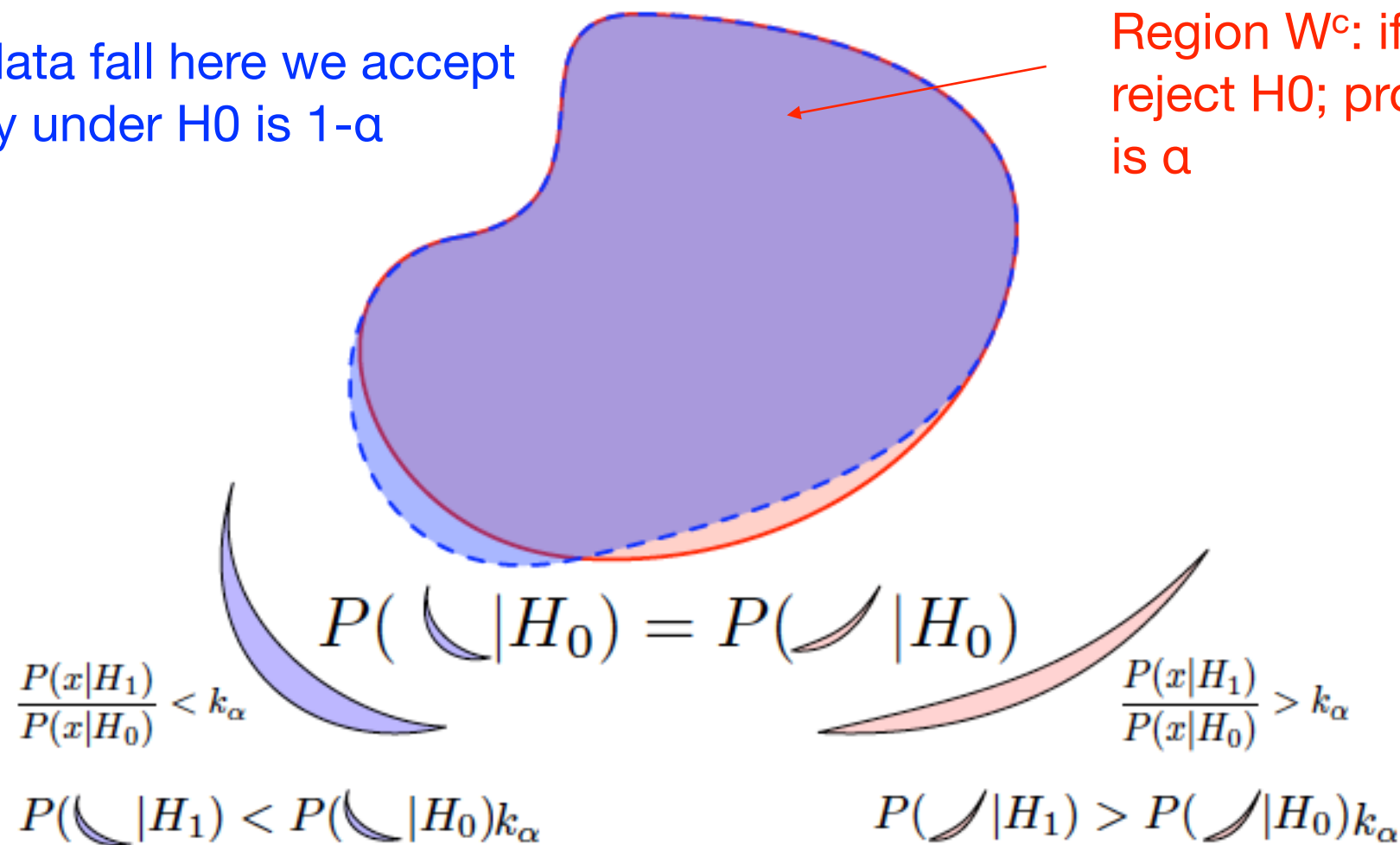


# Yes

---

Region W: if data fall here we accept  $H_0$ ; probability under  $H_0$  is  $1-\alpha$

Region  $W^c$ : if data fall there we reject  $H_0$ ; probability under  $H_0$  is  $\alpha$



$$P(W|H_1) < P(W^c|H_1)$$

The new region region has less power.

# Neyman-Pearson Lemma — remember?

---

The optimal variable exist and it is the likelihood ratio

For any false positive rate (i.e., misclassification of true background events), the region  $W$  of acceptance of  $H_0$ , which minimizes the probability to accept  $H_0$  when  $H_1$  (or, to classify as background a true signal event) is true, is a contour (a cut, in 1D) of the likelihood ratio.

$$\frac{p(x|H_1)}{p(x|H_0)} > k_\alpha$$

Therefore the optimal decision boundary is (where  $x$  can be multidimensional)

$$u(\vec{x}) = \frac{p(\vec{x}|H_1)}{p(\vec{x}|H_0)} = \frac{p(\vec{x}|s)}{p(\vec{x}|b)}$$

(or any monotonic function of it)

# Problem

---

Rarely the densities  $p(\vec{x}|H_1)$  and  $p(\vec{x}|H_0)$  that are needed to evaluate the likelihood ratio for each event are known.

Most of the supervised machine-learning classification task boils down to use the data to find the best approximation of the likelihood ratio

# Guessing the density

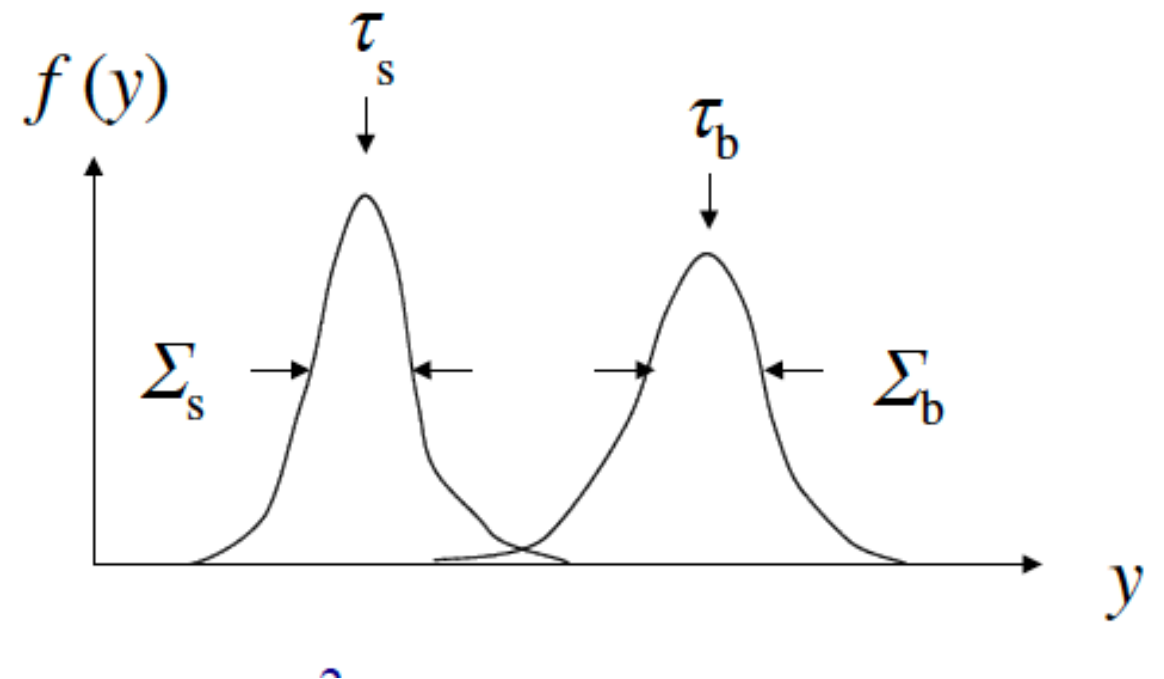
---

Simple guess: assume  $y$  to be a linear function of the features

$$y(\vec{x}) = \sum_{i=1}^n w_i x_i$$

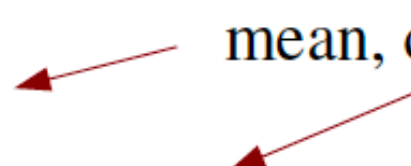
and find the coefficients  $w_i$  that maximize the separation between the distributions of  $y(x)$  on signal and background events:

$$J(\vec{w}) = \frac{(\tau_s - \tau_b)^2}{\Sigma_s^2 + \Sigma_b^2}$$



# Fisher's discriminant

---

We have  $(\mu_k)_i = \int x_i p(\vec{x}|H_k) d\vec{x}$   mean, covariance of  $\mathbf{x}$  [Cowan]

$$(V_k)_{ij} = \int (x - \mu_k)_i (x - \mu_k)_j p(\vec{x}|H_k) d\vec{x}$$

where  $k = 0, 1$  (hypothesis)

and  $i, j = 1, \dots, n$  (component of  $\mathbf{x}$ )

$$J(\vec{w}) = \frac{(\tau_s - \tau_b)^2}{\Sigma_s^2 + \Sigma_b^2}$$

For the mean and variance of  $y(\vec{x})$  we find

$$\tau_k = \int y(\vec{x}) p(\vec{x}|H_k) d\vec{x} = \vec{w}^T \vec{\mu}_k$$

$$\Sigma_k^2 = \int (y(\vec{x}) - \tau_k)^2 p(\vec{x}|H_k) d\vec{x} = \vec{w}^T V_k \vec{w}$$

# Fisher's discriminant

---

The numerator of  $J(\mathbf{w})$  is

[Cowan]

$$(\tau_0 - \tau_1)^2 = \sum_{i,j=1}^n w_i w_j (\mu_0 - \mu_1)_i (\mu_0 - \mu_1)_j$$

$$= \sum_{i,j=1}^n w_i w_j B_{ij} = \vec{w}^T B \vec{w}$$

← 'between' classes

and the denominator is

$$\Sigma_0^2 + \Sigma_1^2 = \sum_{i,j=1}^n w_i w_j (V_0 + V_1)_{ij} = \vec{w}^T W \vec{w}$$

← 'within' classes

→ maximize

$$J(\vec{w}) = \frac{\vec{w}^T B \vec{w}}{\vec{w}^T W \vec{w}} = \frac{\text{separation between classes}}{\text{separation within classes}}$$



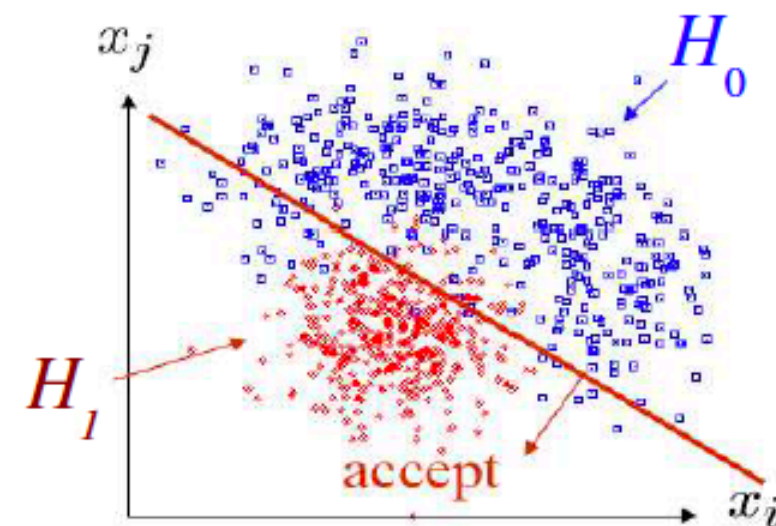
# Fisher's discriminant

Setting  $\frac{\partial J}{\partial w_i} = 0$  gives Fisher's linear discriminant function:

[Cowan]

$$y(\vec{x}) = \vec{w}^T \vec{x} \quad \text{with } \vec{w} \propto W^{-1}(\vec{\mu}_0 - \vec{\mu}_1)$$

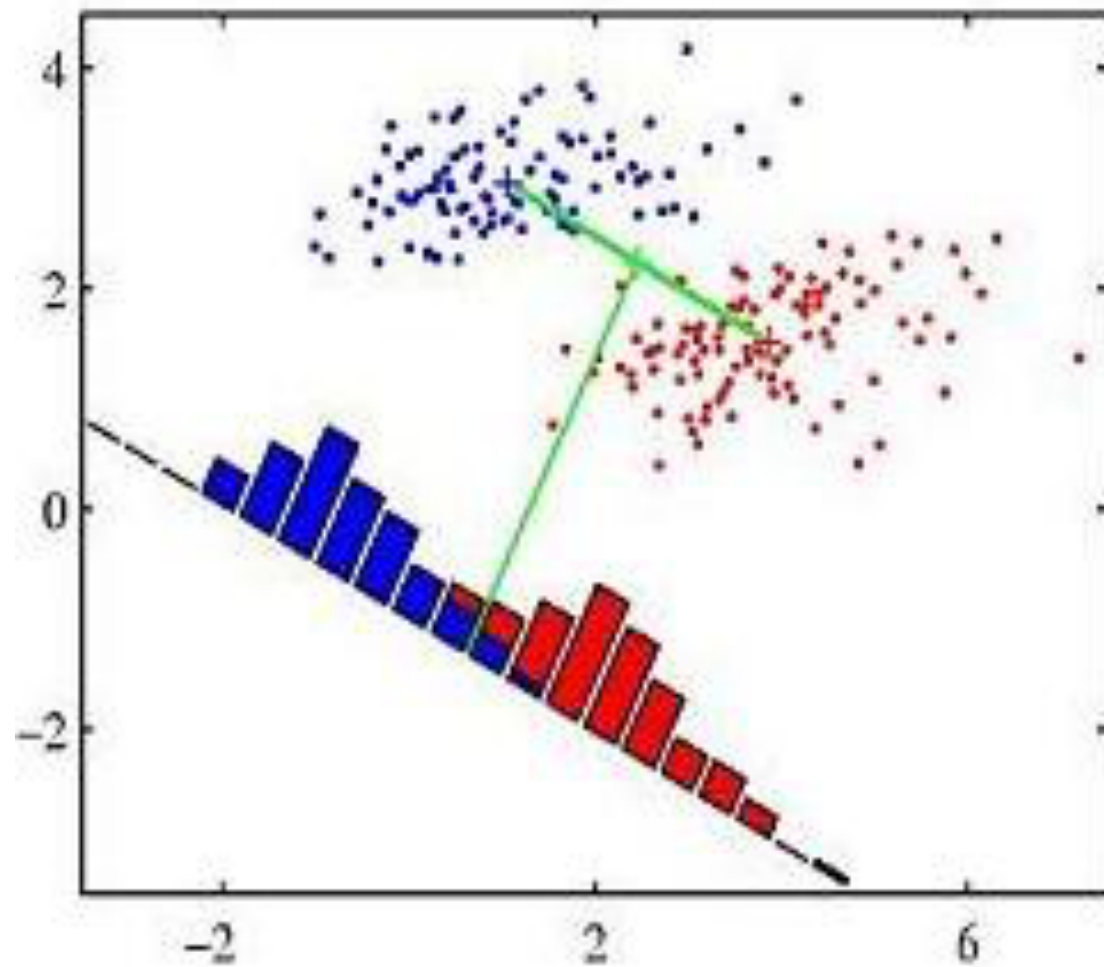
The resulting weights define the linear decision boundary such that the projection of the points along the tangent of the boundary produces maximally separated distributions.



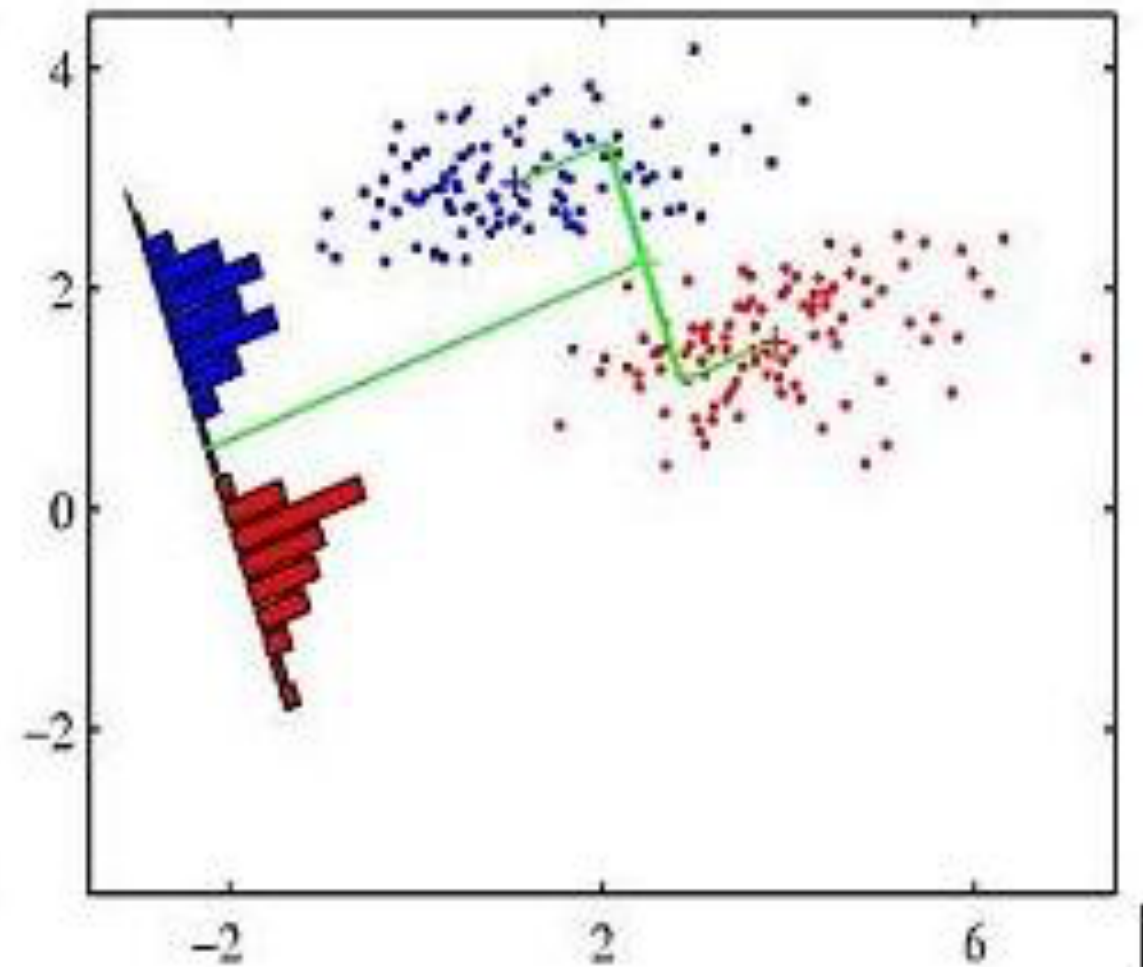
# Fisher's discriminant

---

Suboptimal separation



Fisher discriminant



# Fisher's discriminant

---

For Gaussian data with equal covariance the Fisher discriminant offers the optimal decision boundary.

Suppose  $f(\mathbf{x}|H_k)$  is a multivariate Gaussian with mean values

[Cowan]

$$E_0[\vec{x}] = \vec{\mu}_0 \text{ for } H_0 \quad E_1[\vec{x}] = \vec{\mu}_1 \text{ for } H_1$$

and covariance matrices  $V_0 = V_1 = V$  for both. We can write the Fisher's discriminant function (with an offset) is

$$y(\vec{x}) = w_0 + (\vec{\mu}_0 - \vec{\mu}_1)^T V^{-1} \vec{x}$$

The likelihood ratio is thus

$$\frac{p(\vec{x}|H_0)}{p(\vec{x}|H_1)} = \exp\left[-\frac{1}{2}(\vec{x} - \vec{\mu}_0)^T V^{-1}(\vec{x} - \vec{\mu}_0) + \frac{1}{2}(\vec{x} - \vec{\mu}_1)^T V^{-1}(\vec{x} - \vec{\mu}_1)\right] = e^y$$

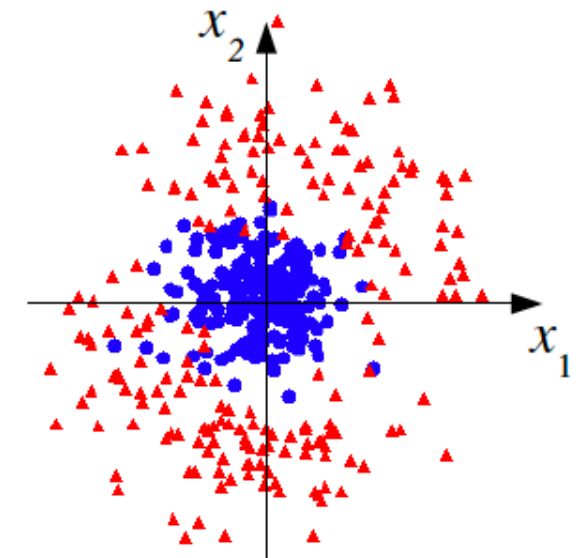
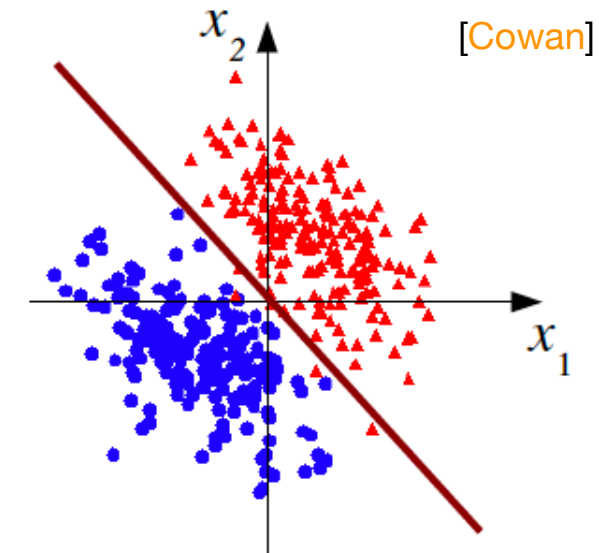
The Fisher's discriminant is a monotonic function of the likelihood ratio and is therefore optimal (for Gaussian data with equal covariance)

# Limitations of linear boundaries

---

A linear decision boundary is optimal when the classes of events to be separated are distributed as multivariate Gaussians with same covariance and differing mean

When data are non-Gaussian, linear decision boundaries can fail.



# Limitations of linear boundaries

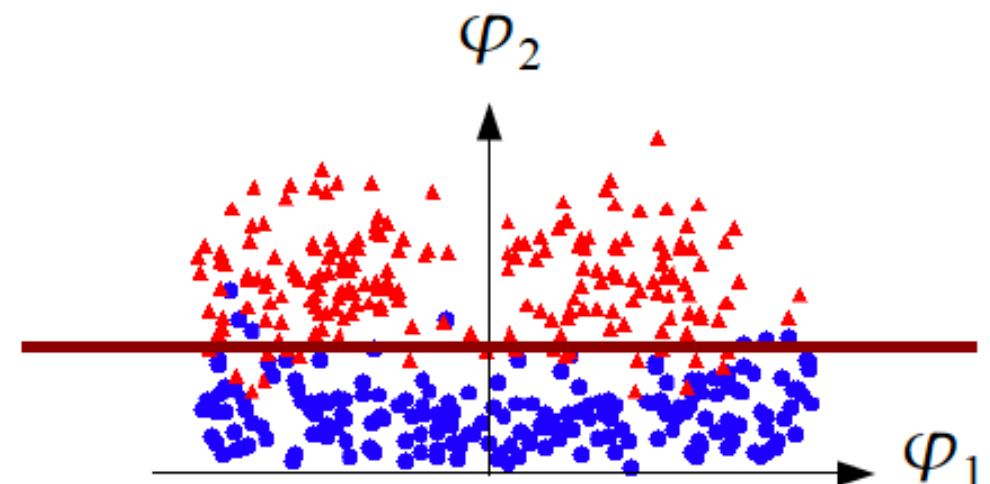
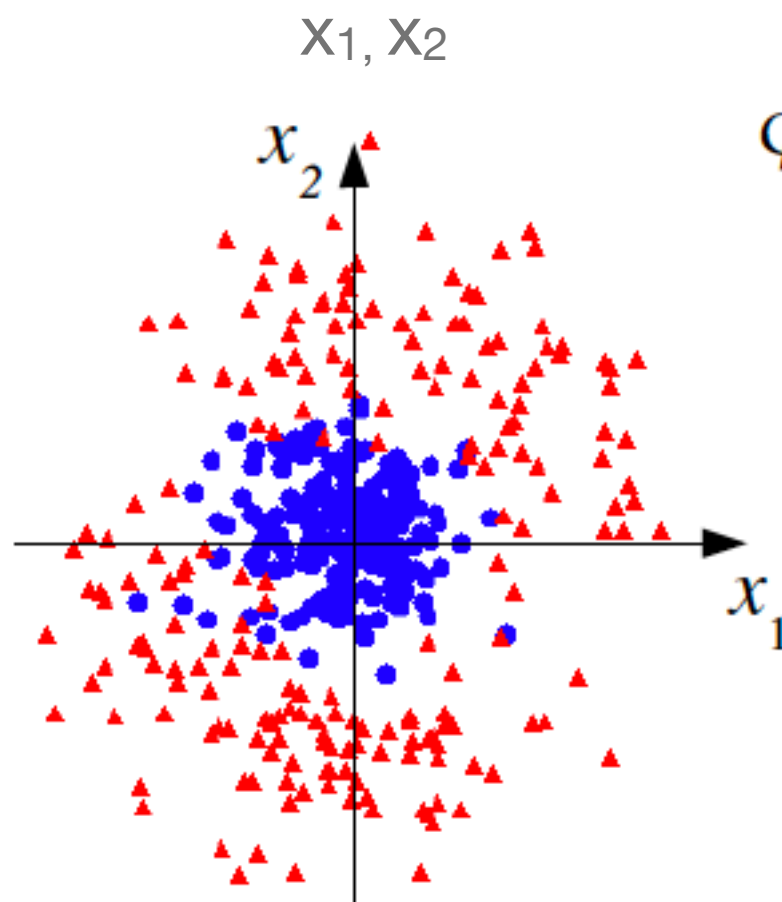
Occasionally in simple problems, a nonlinear transformation that maps the feature space into variables that are more likely to be linearly separable is evident

$$x_1, \dots, x_n \implies \phi_1(\vec{x}) \dots \phi_n(\vec{x})$$

[Cowan]

$$\phi_1(x_1, x_2) = \tan^{-1}(x_2/x_1)$$

$$\phi_2(x_1, x_2) = (x_1^2 + x_2^2)^{1/2}$$



In general, the functions of the feature space  $\vec{\phi}(\vec{x}, \vec{w})$  depend also on free parameters  $\vec{w}$ .

# Nonlinear discriminants

---

In general, the set of basis functions in the feature space that allows/optimizes the classification is not evident.

A number of approaches offer algorithms to identify and parametrize such basis functions to offer effective classification.

Among the most commonly used nonlinear discriminants in HEP are artificial *neural networks* (some similarities with neuronal functionality)

Used in HEP since the early 80's — quite some time after the initial works by McCulloch and Pitts (1943) and Rosenblatt (1962).

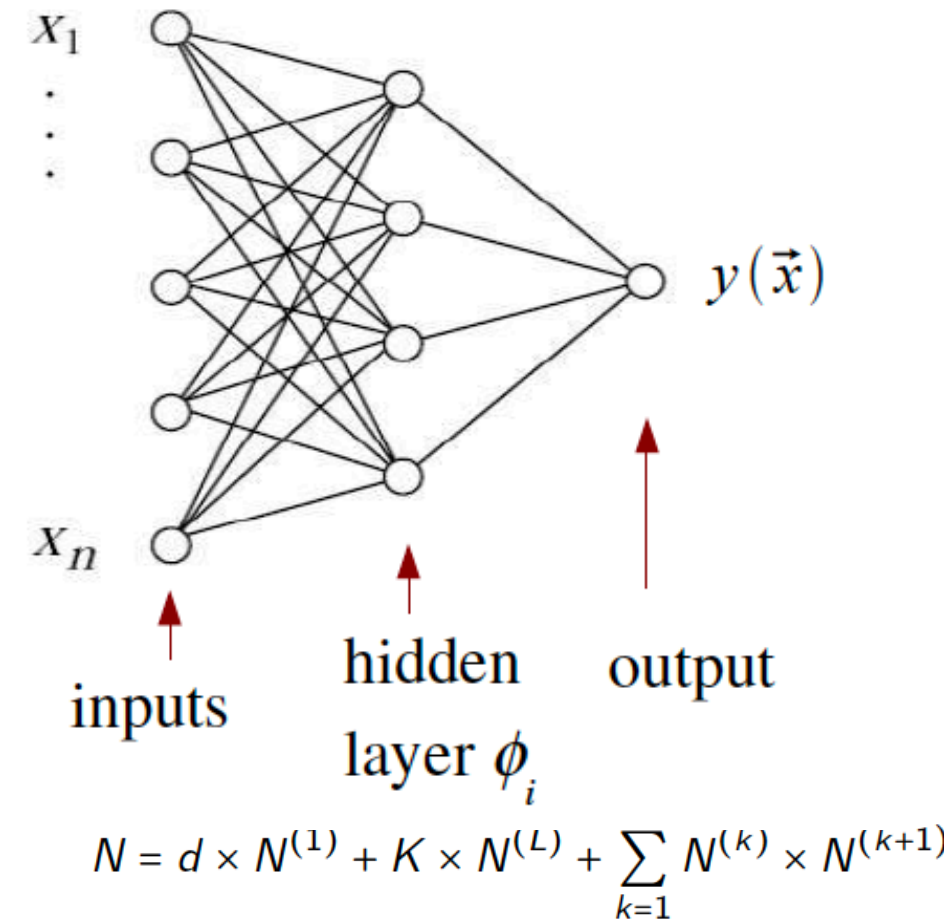
# Artificial Neural Networks

Define a number of **input “nodes”** (driven by the dimensionality of feature space  $\vec{x}$ ) and an **output  $y(\vec{x})$** . i.e., a scalar variable where a single cut defines a decision boundary.

Choose a number (1 to few) of **intermediate “hidden” layers**. In each, choose a number of nodes. More layers/nodes imply more model parameters ( $N$ ).

Each node connects to the downstream nodes. The intensities of the connections are tunable weights  $w$

Choose a monotonic nonlinear function that expresses the “excitation” of each node in response to input from the upstream nodes (e.g.,  **$h(s) = 1 / (1 + e^{-s})$** )



- $L$  is the number of **hidden layers**
- $N^{(k)}$  is the number of nodes for layer  $k$
- $d$  is the input vector size,  $K$  is the output size



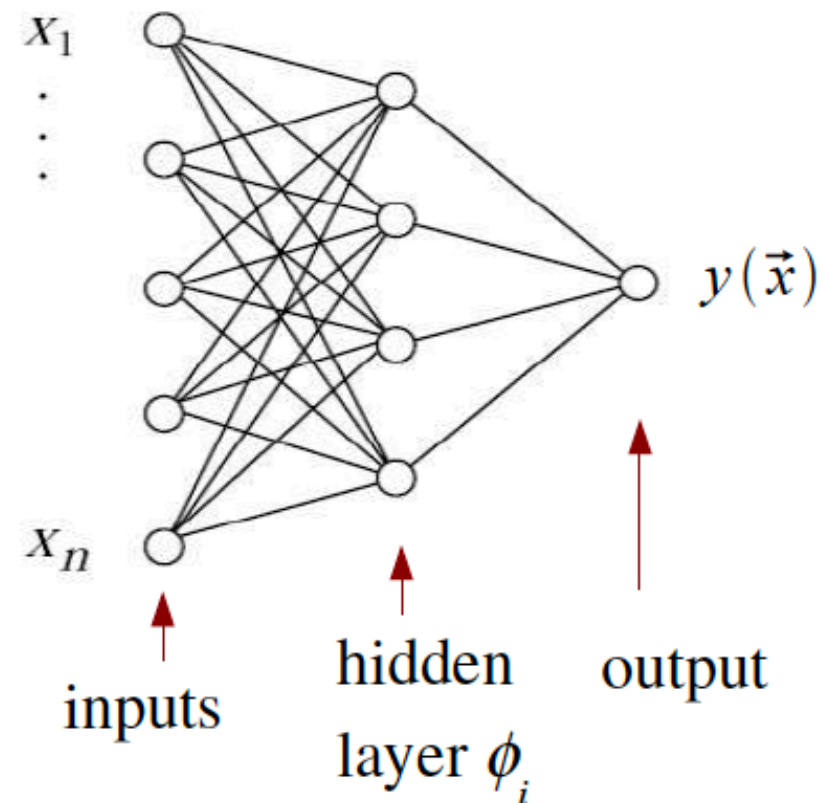
# Artificial Neural Networks

---

Superscript for weights indicates layer number

$$\varphi_i(\vec{x}) = h\left(w_{i0}^{(1)} + \sum_{j=1}^n w_{ij}^{(1)} x_j\right)$$

$$y(\vec{x}) = h\left(w_{10}^{(2)} + \sum_{j=1}^n w_{1j}^{(2)} \varphi_j(\vec{x})\right)$$



When the ANN receives some input data, in each node, the weighted inputs incoming from the preceding nodes are fed to the activation function, which outputs to the resulting activation intensity to the following nodes.

The classification performance depends on the value of the weights. These are optimized during the training phase.



# Training

---

Before the ANN can classify unknown events, the ANN is fed events of known classification (simulation, typically) so that it can “learn”. Each event “a” comes with its set of features  $\vec{x}_a = (x_1, \dots, x_n)$  and its true class  $t_a = 0$  or  $1$ .

The set of optimal weights  $w$  is obtained by minimizing an error function that quantifies how much the classification achieved by the ANN departs from the true classification (known for the training sample). Error-function example

$$E(\mathbf{w}) = \frac{1}{2} \sum_{a=1}^N |y(\vec{x}_a, \mathbf{w}) - t_a|^2:$$

Minimizing  $E(w)$  over the space of weights to obtain their optimal values

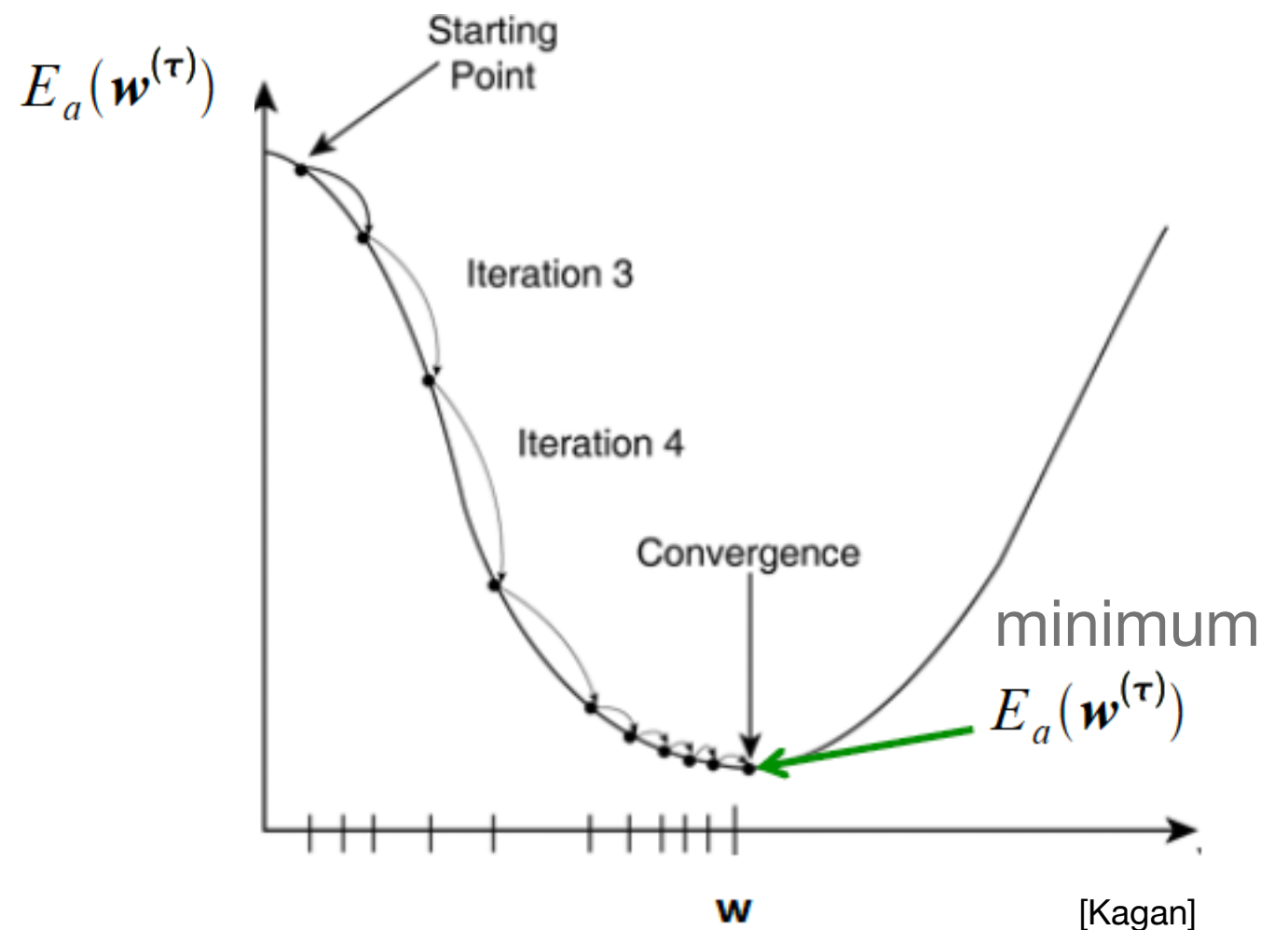
# Gradient descent

The error function is minimized numerically, e.g., by using the gradient descent method: start from an initial guess (or random choice) and make a step in the direction of maximum decrease.

[Cowan]

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta \nabla E_a(\mathbf{w}^{(\tau)})$$

Update  $w$  for each training event  $a$ .



Error backpropagation: determines the derivatives needed to calculate the gradient directions at each node using a recursive rule.

# Overtraining

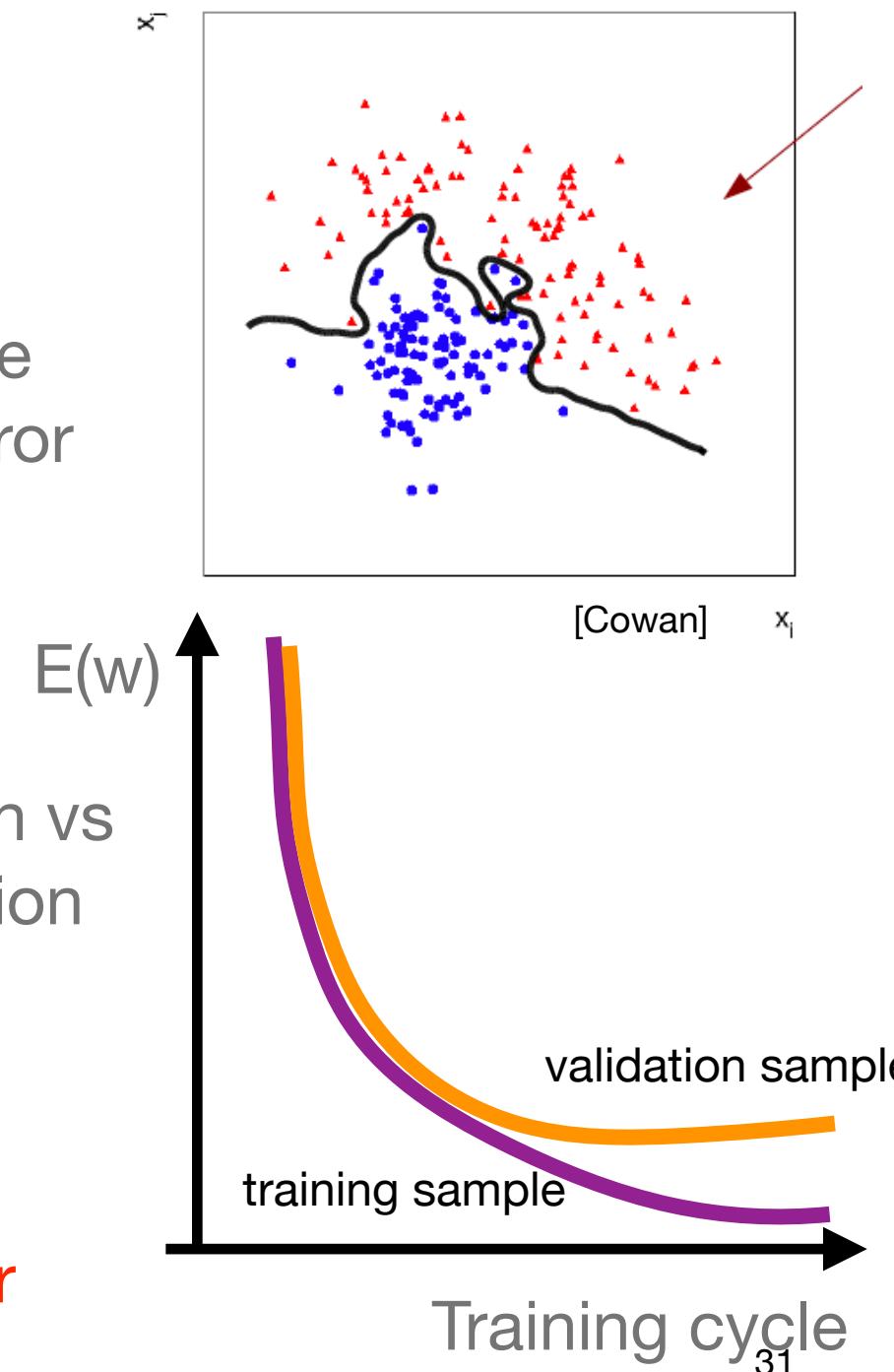
The number of inputs and inner nodes should be optimized for the problem .

Too many nodes (i.e., free parameters) yield outputs conforming too closely the training data.

**Overtraining:** decision boundary follows the details of the statistical fluctuations yielding an unrealistically small error rate on the training data.

Evaluate classification performance on a independent validation sample. Different behavior of the error function vs training cycle between the training sample and a validation sample indicates the onset of overtraining.

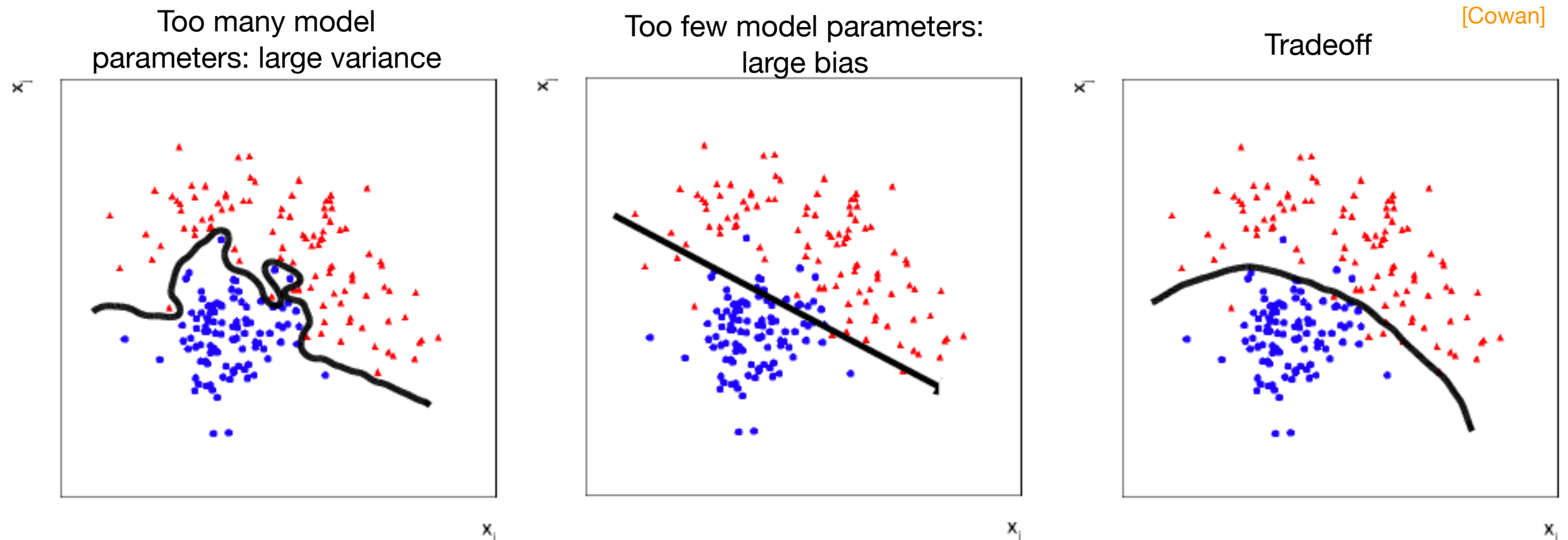
After the ANN architecture is optimized, the expected performance should be evaluated in a test sample (other than the training and validation samples).



# Bias-variance tradeoff

At given training sample size, the higher the ANN complexity (more hidden layers, more nodes) the larger the statistical errors of the ANN parameters, since information from the same data is used to determine a larger number of parameters. This yields overtraining and high variance.

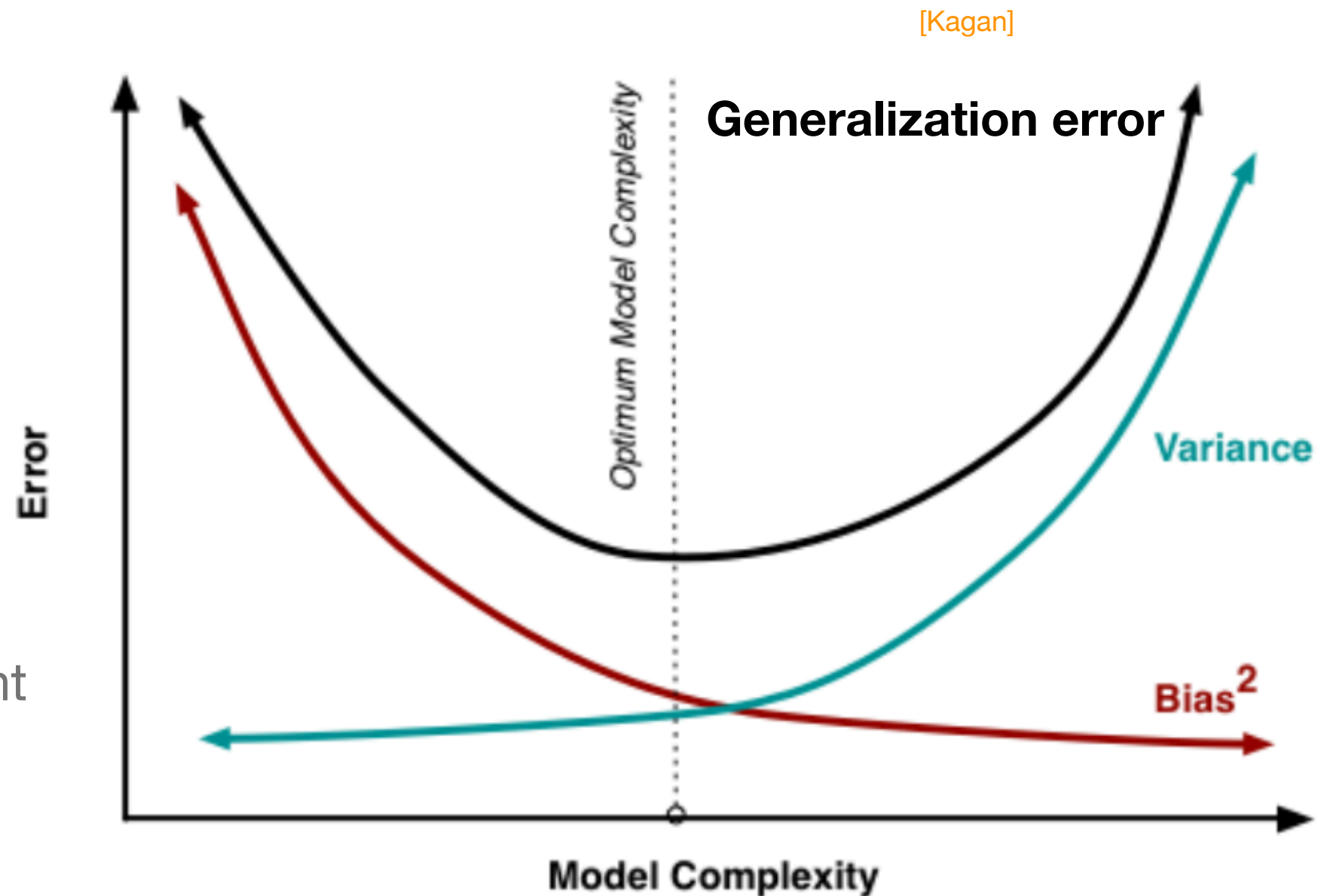
However, with the too few nodes, the ANN struggles to exploit the nonlinearities, yielding bias.



# Bias-variance tradeoff

Evaluate the generalization error of your procedure by splitting data set in three samples:

- 1 **training sample**: fit values of model parameters
2. **validation sample**: check performance on independent data and optimize it by tuning # of parameters
3. **Test sample** final evaluation of performance, with all parameters fixed



# Regularization

---

One approach for avoiding overfitting/overtraining is to incorporate in the cost function a penalty for large weights.

$$\tilde{E}(\mathbf{w}) = E(\mathbf{w}) + \frac{\lambda}{2} \mathbf{w}^T \mathbf{w}$$

Increasing the regularization parameter  $\lambda$  drives the weights to zero, thus effectively reducing the number of nodes unless they are consistently supported by the training data

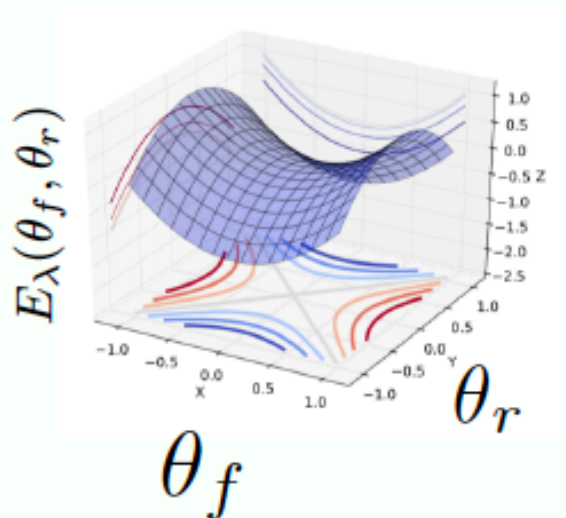
# Advanced usages — adversarial networks

Typically classifier  $\mathbf{f}(\mathbf{x})$  trained to minimize loss  $\mathbf{L}_f$ .

- want classifier output to be insensitive to systematics (nuisance parameter  $\mathbf{v}$ )
- introduce an **adversary**  $\mathbf{r}$  that tries to predict  $\mathbf{v}$  based on  $\mathbf{f}$ .
- setup as a minimax game:

$$\hat{\theta}_f, \hat{\theta}_r = \arg \min_{\theta_f} \max_{\theta_r} E(\theta_f, \theta_r).$$

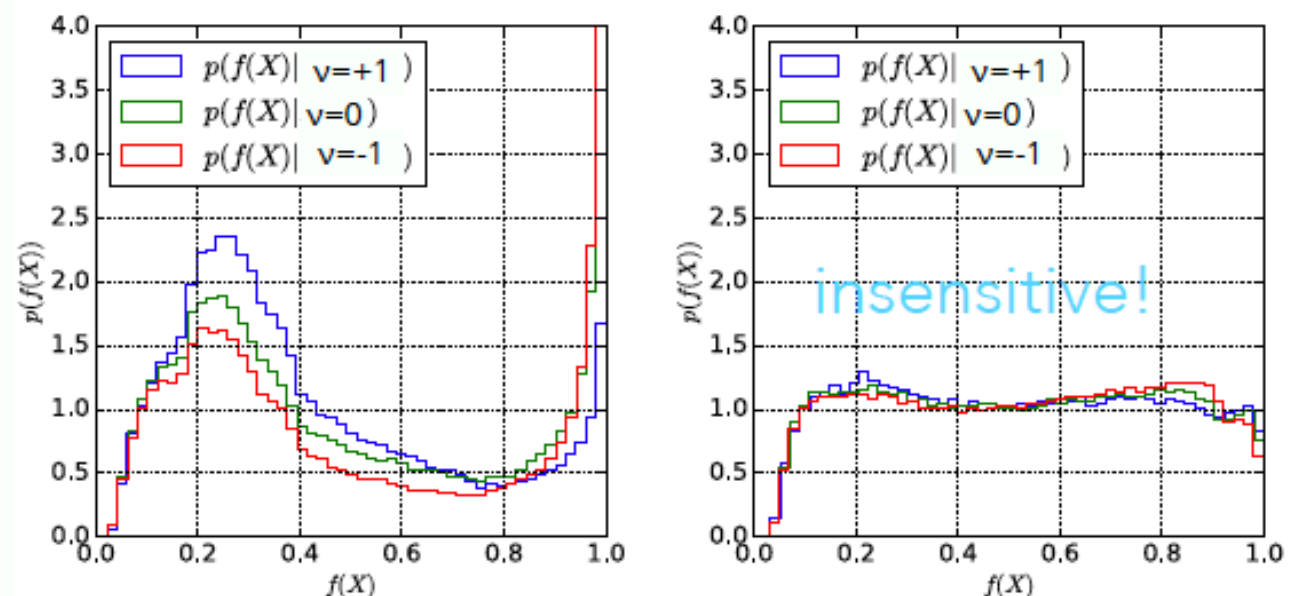
$$E_\lambda(\theta_f, \theta_r) = \mathcal{L}_f(\theta_f) - \lambda \mathcal{L}_r(\theta_f, \theta_r)$$



Make the classifier output less sensitive to systematic effects or less biasing toward variables that may need to be fit downstream in the analysis.

Similar techniques been applied to decision trees (more later)

Discriminating function for various values of  $\mathbf{v}$



# Universal approximation theorem

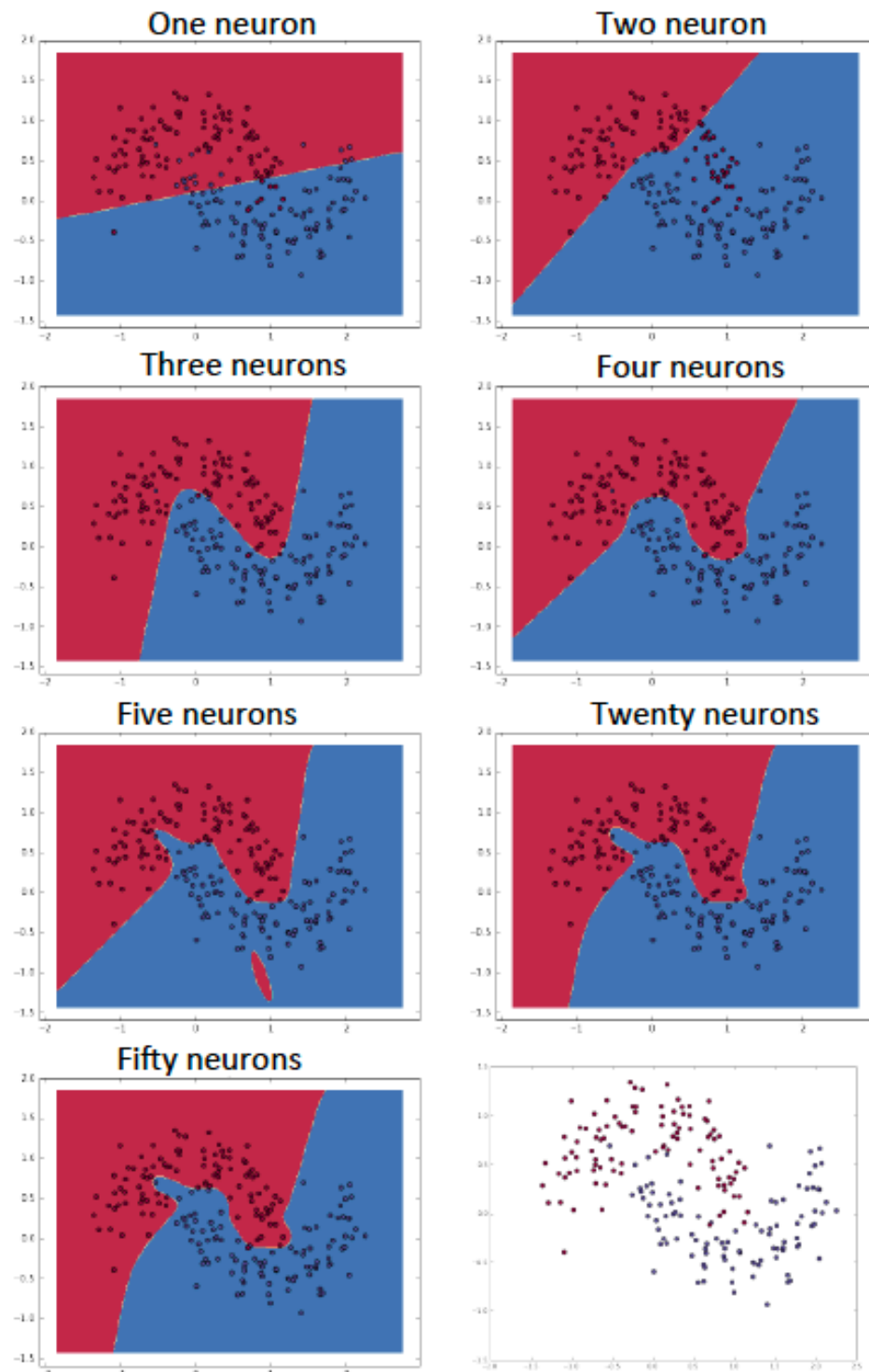
---

A feed-forward multilayer perceptron with a single hidden layer and a finite number of nodes activated through any continuous nonpolynomial function can approximate arbitrarily well any continuous function

(Says nothing on how many nodes we need or how many data...)

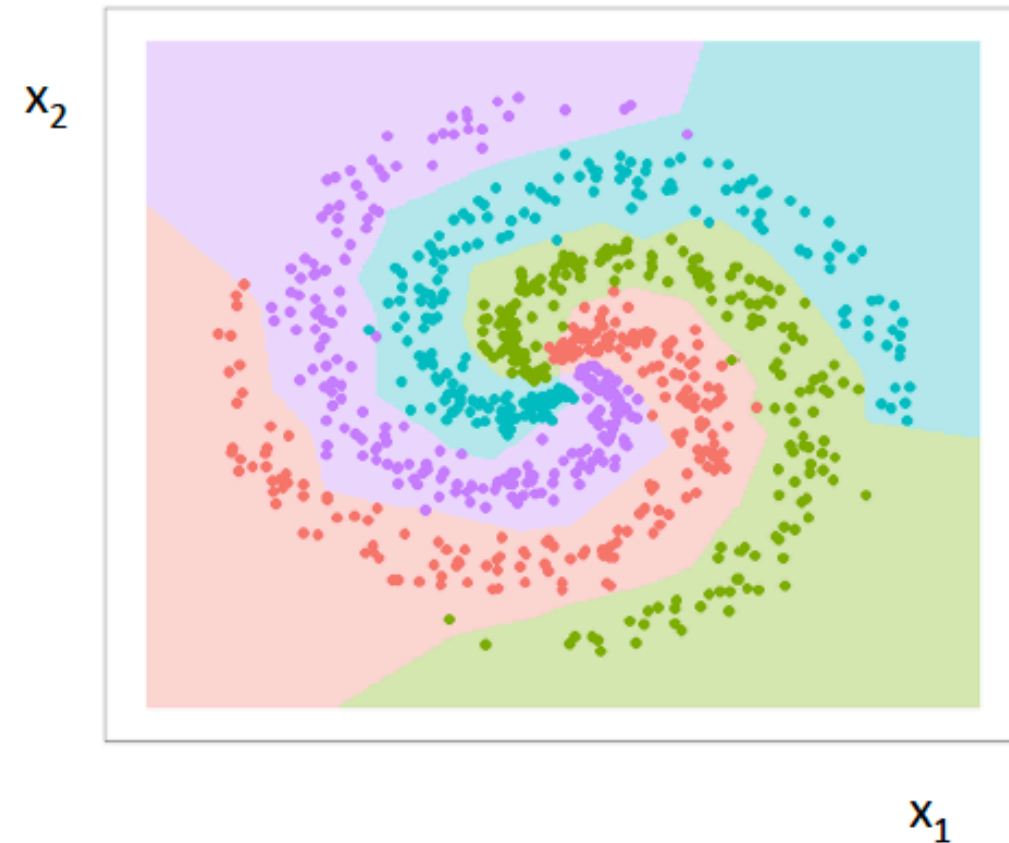


# Universal approximation theorem



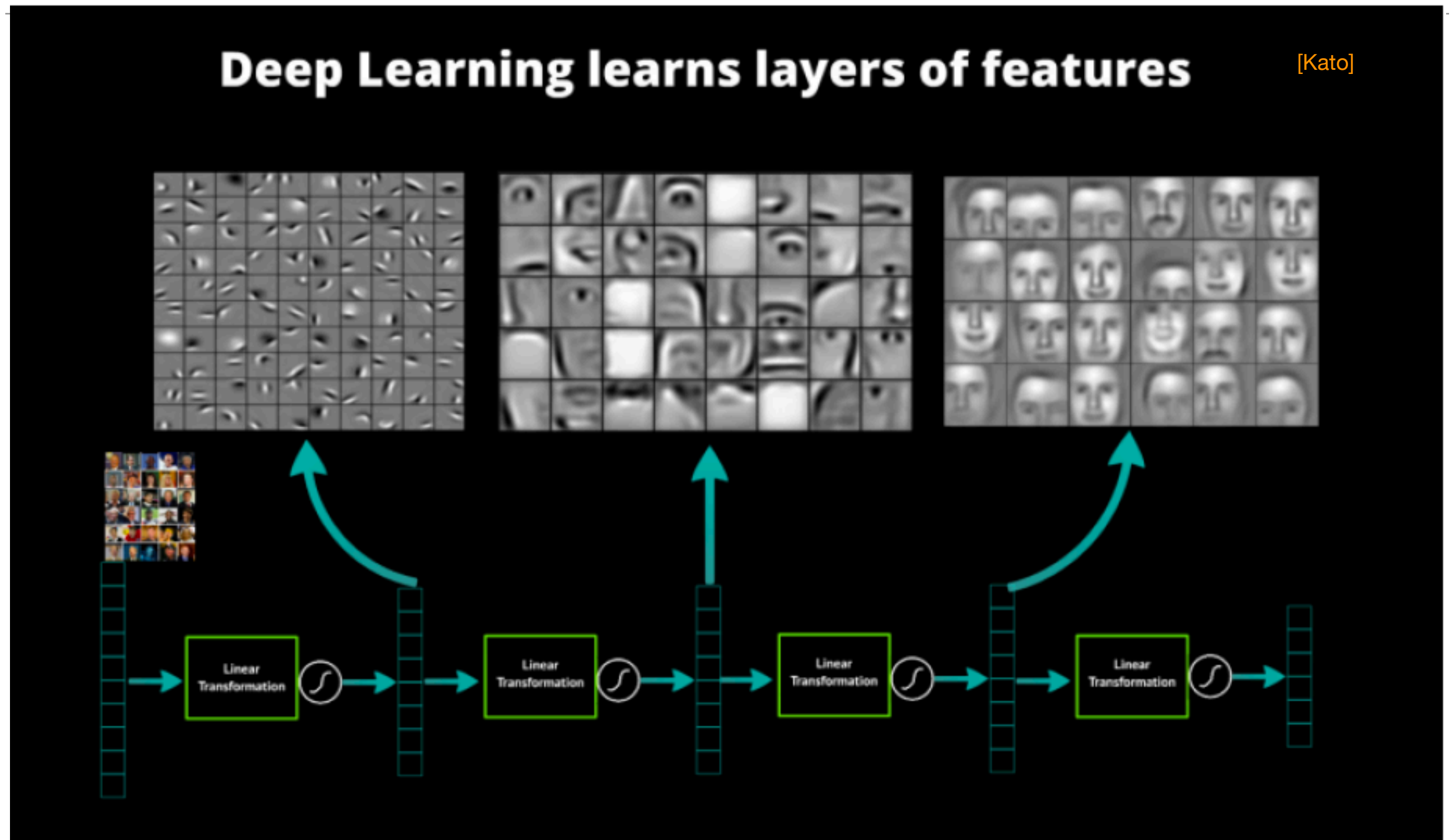
4-class classification  
2-hidden layer NN  
ReLU activations  
L2 norm regularization

[Kagan]



2-class classification  
1-hidden layer NN  
L2 norm regularization

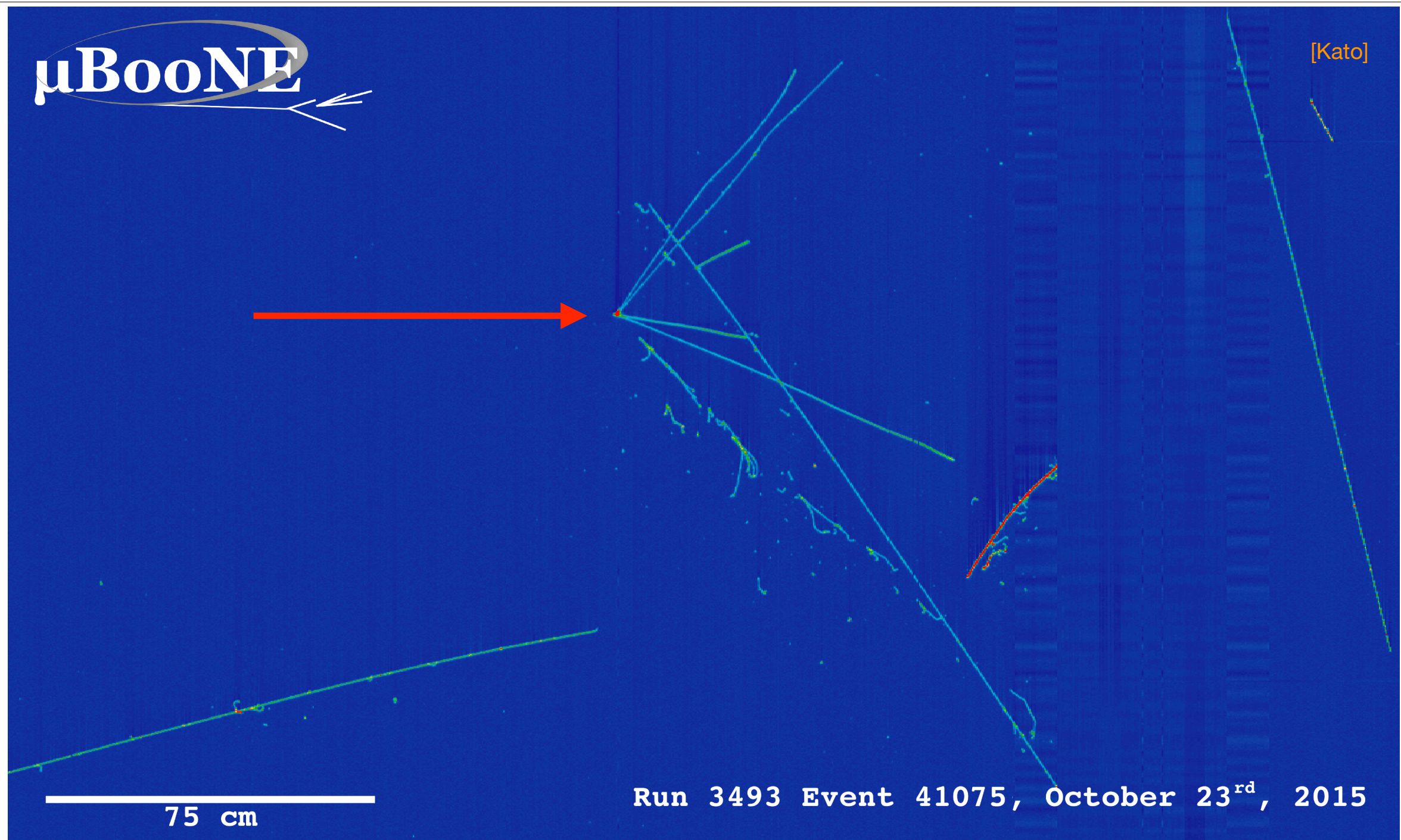
# Advanced usages — convolutional networks



Promising for the analysis of LAr time-projection chamber “event pictures” from current and future neutrino experiments



# Advanced usages — convolutional networks



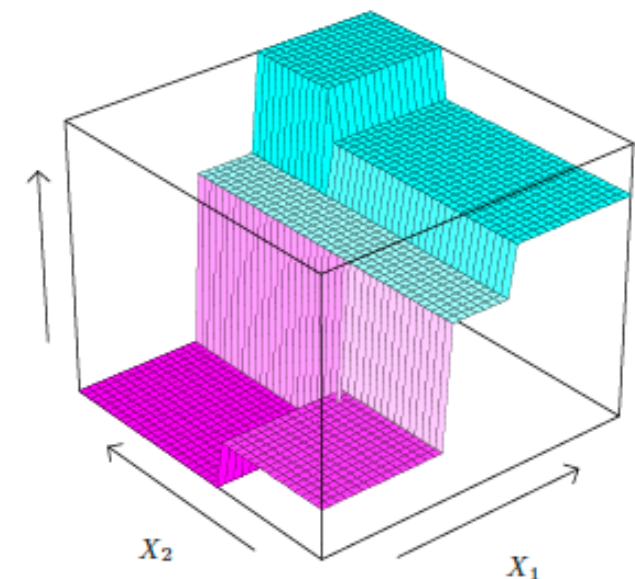
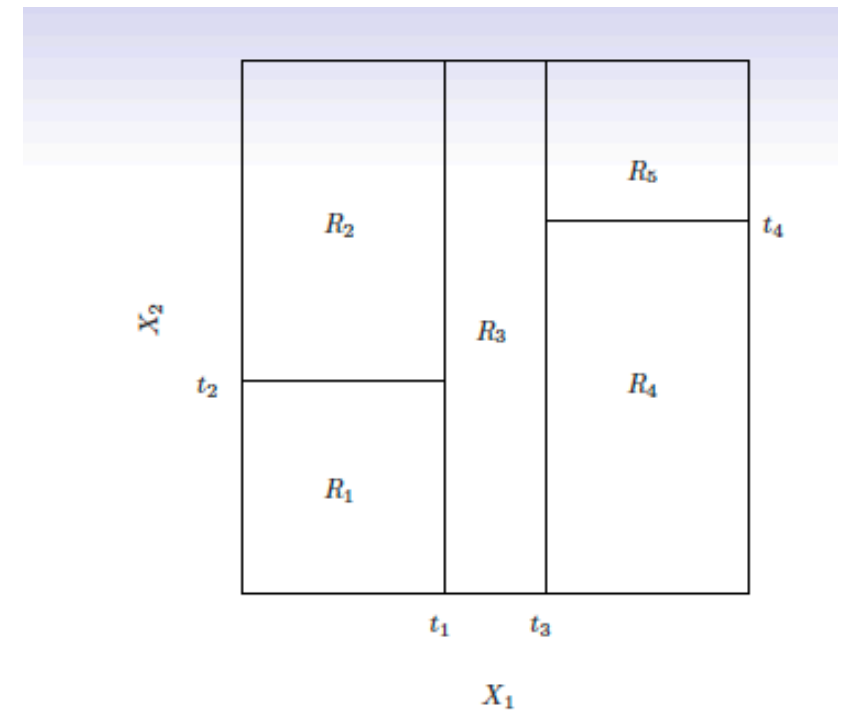
Promising for the analysis of LAr time-projection chamber “event pictures” produced in current and future neutrino experiments

# Decision trees

Algorithms that subdivide the space of the input variables (features) into a number of simple nonoverlapping regions (n-dimensional rectangles).

Each region will be labeled as “signal region” or “background region” according to the predominant category of training events that populate it after training.

Such labels are used to classify the test events.



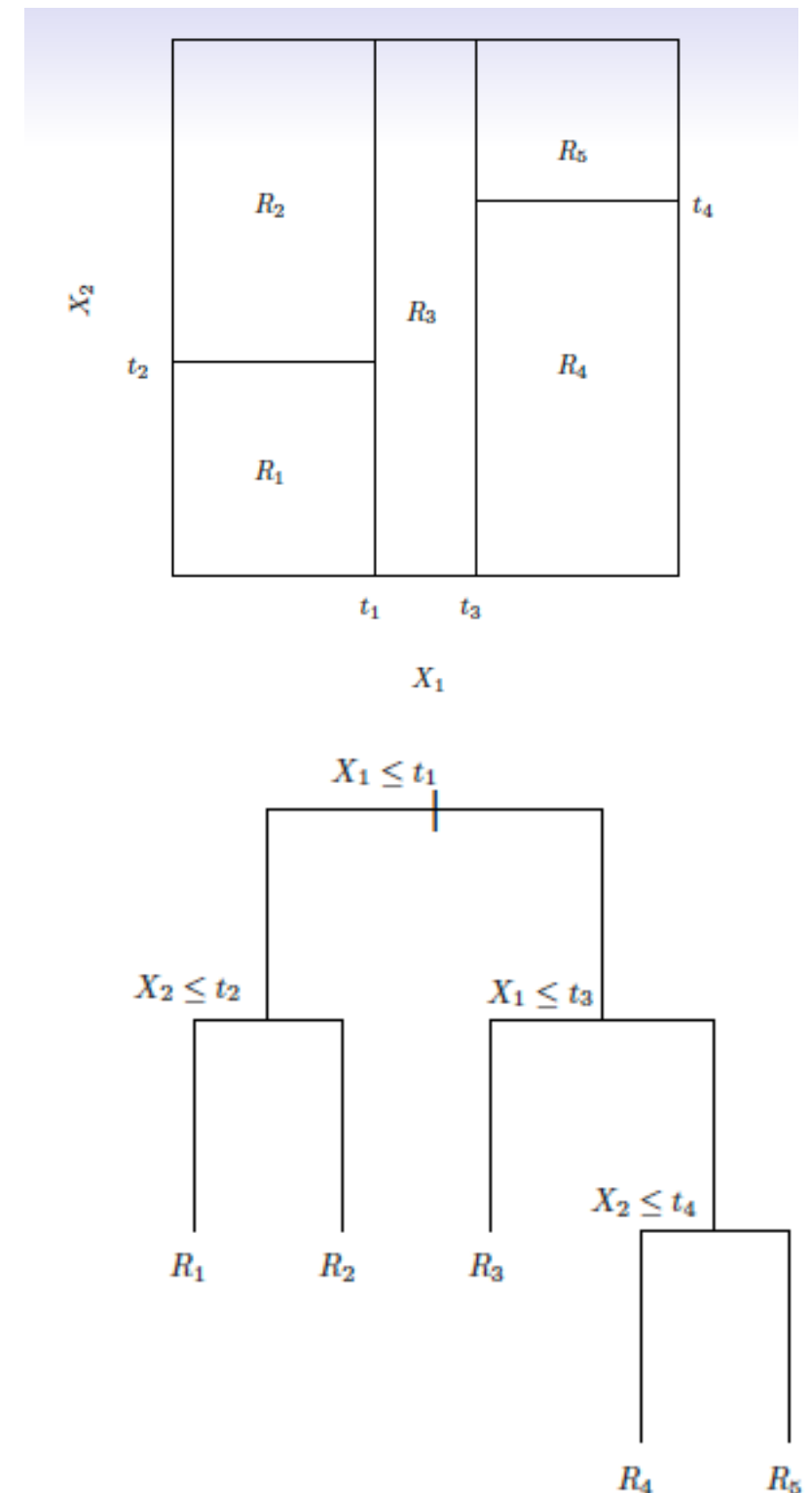


# Decision trees

Subdivision of feature-space corresponds to a set of splitting rules.

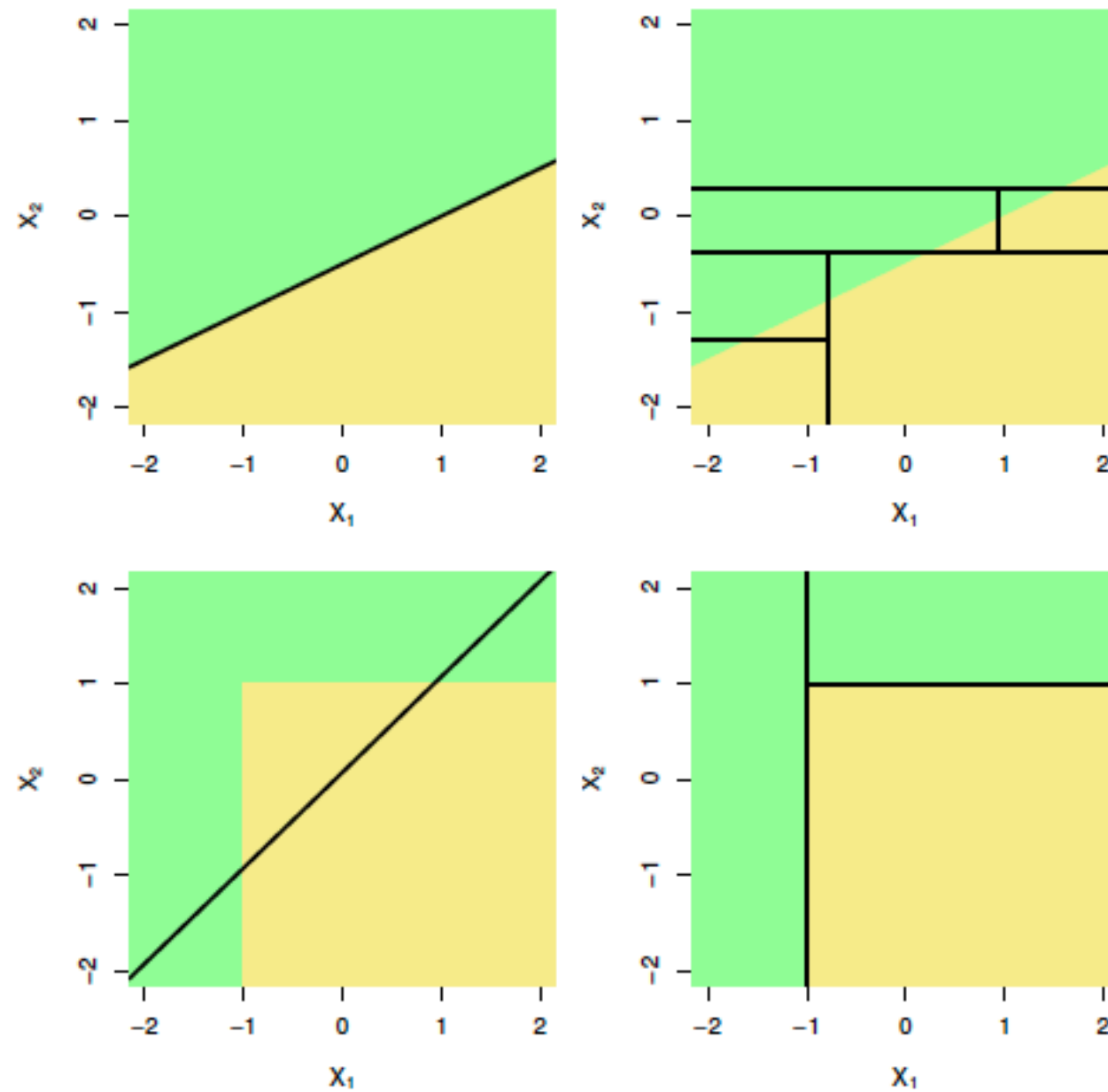
Represent through an (inverted) tree-like structure made of a cascade of decision nodes, each associated to an input variable

Each node tests a single feature of the event at a time (e.g., cuts on a single variable) and routes the event on one of its downstream branches. partitioning the sample into subsamples of increasing purity until the final classification is reached (events accumulated in the terminal nodes (leafs)).



# Trees vs linear models

---



# Building the tree —how the subregions are chosen

---

Recursive binary splitting.

At each building step, choose the input variable and the cut threshold on it that minimizes a cost function.

Misclassification rate, the fraction of events of the training sample in that region that do not belong to the most common class (not really used)

$$E = 1 - \max_k(\hat{p}_{mk}).$$

Here  $\hat{p}_{mk}$  represents the proportion of training observations in the  $m$ th region that are from the  $k$ th class.

Typically use cross entropy

$$D = - \sum_{k=1}^K \hat{p}_{mk} \log \hat{p}_{mk}. \quad \text{or functions of it (Gini)}$$

# Building the tree —how the subregions are chosen

---

Repeat the process of choosing the variable and cut value that minimizes the cost but do it restricting to one of the two regions previously identified.

Without a stopping criterion, the training sample could end up being exactly classified, leading to strong overtraining. Typically impose stopping criteria like minimum number of entries in a node or achieved purity.

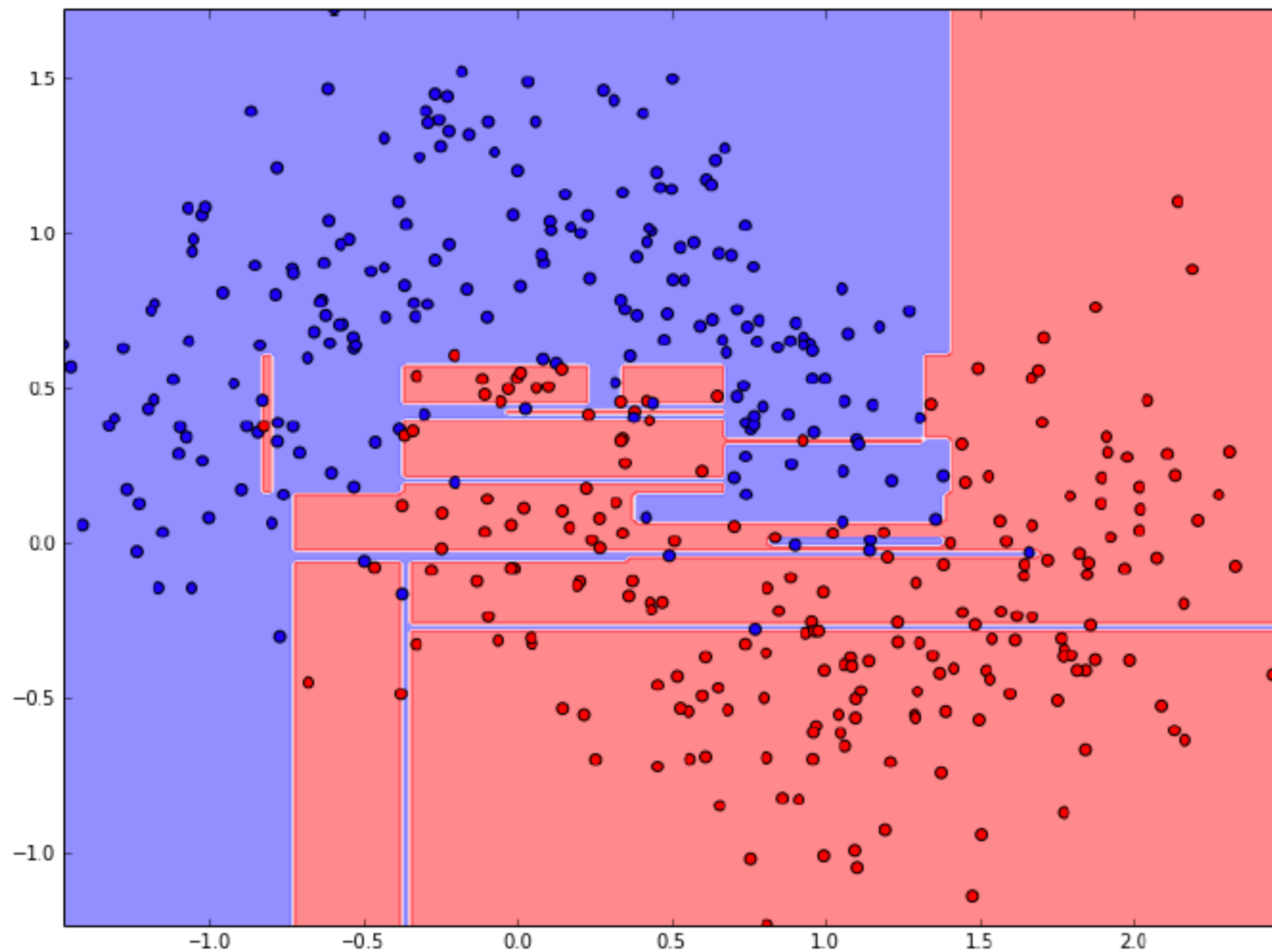
Still decision trees can grow very large and pruning is applied. For instance terminal leaves are recombined if their purity is compatible within statistical uncertainties.



# Instabilities

Tree keeps splitting until each event is correctly classified

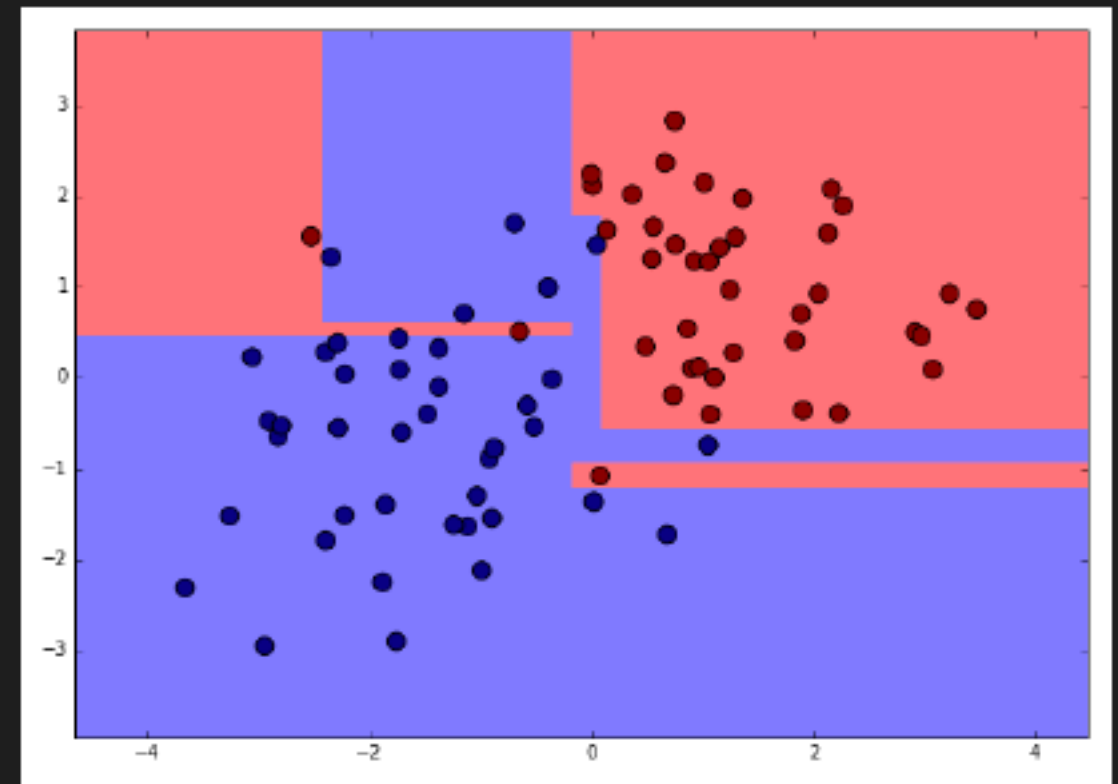
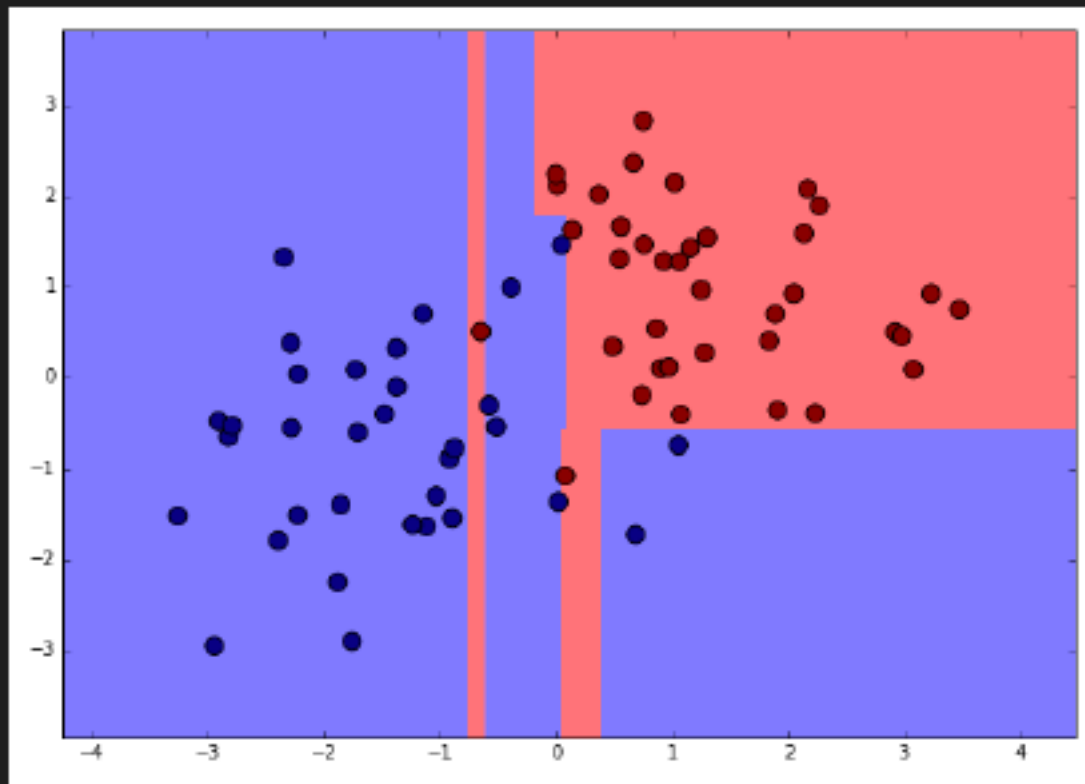
[Rogozhnikov]



# Instabilities

[Rogozhnikov]

Little variation in training dataset produce different classification rule.



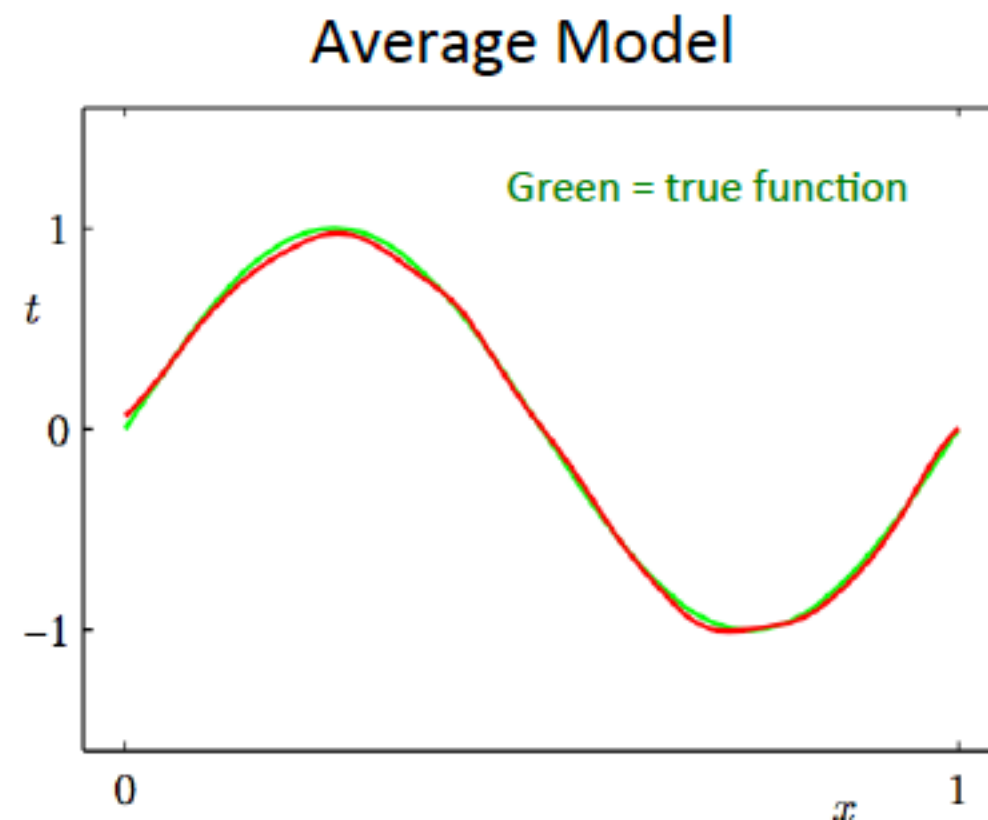
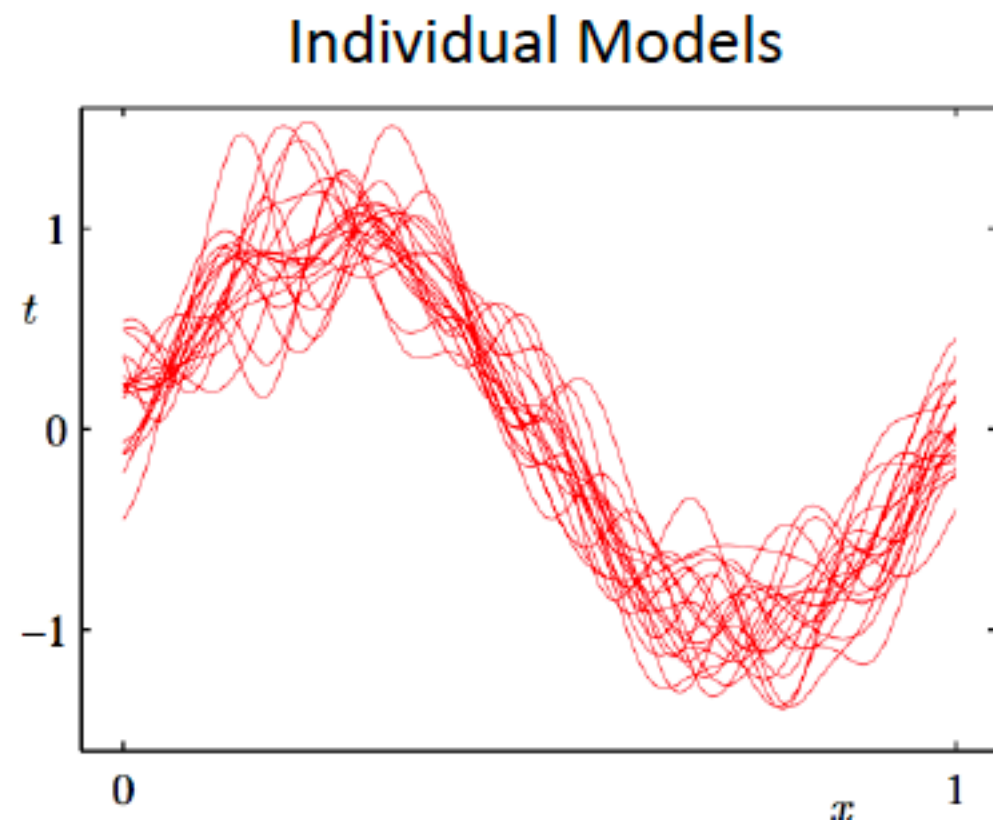
# Improving trees

---

Typically classification performance of decision trees is not competitive with that from other machine-learning approaches.

By aggregating multiple decision trees, the classification performances is improved.

Combination of several weak learners with high variance goes a long way



# Improving trees - bagging/boosting

---

Generate  $N$  nearly independent training samples by resampling a single training sample. Build a tree based on each of the resampled training samples. For each test point in feature space, look at the classification of the  $N$  trees, and **define the classification output of the tree as the most common output** among the  $N$  trees (bagging)

Train  $N$  trees in sequence, giving in each more weight to the training examples misclassified in the previous tree. Take the weighted vote of the outputs to classify the examples. (boosting)

(not specific of decision trees, can be applied to other machine-learning methods).

# Improving trees - random forests

---

Tweak the bagging algorithm decorrelates better the trees obtained by the resampled training data.

When considering a split in the building step, only a random subset of the whole set of features is available for choose the splitting-variable.

This “forces” the trees to develop differently thus reducing correlations among them and therefore the variance of the bagged trees

E.g, if one feature is much more discriminating than all others. A split based such feature would likely to be at the top of most the trees derived from the resampled training data. Such trees will therefore be similar and yield strongly cotrrelated outputs strongly. Since correlations do not reduce by averaging, the advantages of bagging will be lost.

# Practical advice

---

# No free lunch theorem

**ARTICLE** \_\_\_\_\_ **Communicated by Steven Nowlan**

## The Lack of A Priori Distinctions Between Learning Algorithms

David H. Wolpert

*The Santa Fe Institute, 1399 Hyde Park Rd.,  
Santa Fe, NM, 87501, USA*

Many ML methods and tools out there to improve our reach in HEP-specific problems: linear, nearest neighbor, ANN, DeepNN, DT ensembles, support vector machines...

With no prior knowledge, general statements on performance are hard.  
Performance very much dependent on the details of the problem at hand.

The only shortcut to your trial and error is if your problem mirrors a similar problem somebody else has already explored.

# Empirical heuristics

[Kagan]

- Test 179 classifiers (no deep neural networks) on 121 datasets  
<http://jmlr.csail.mit.edu/papers/volume15/delgado14a/delgado14a.pdf>
  - *The classifiers most likely to be the bests are the random forest (RF) versions, the best of which (...) achieves 94.1% of the maximum accuracy overcoming 90% in the 84.3% of the data sets*

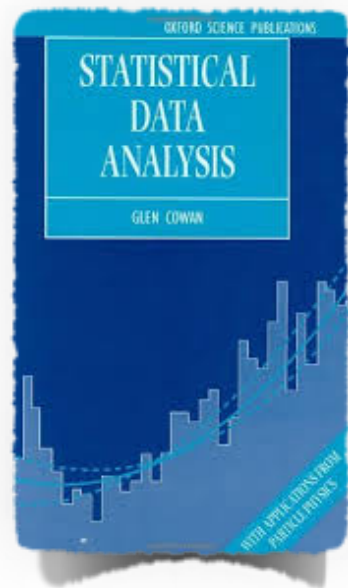
## From Kaggle

- For Structured data: “High level” features that have meaning
  - Winning algorithms have been lots of **feature engineering + random forests**, or more recently **XGBoost** (also a decision tree based algorithm)
- Unstructured data: “Low level” features, no individual meaning
  - Winning algorithms have been deep learning based, **Convolutional NN** for image classification, and **Recurrent NN** for text and speech

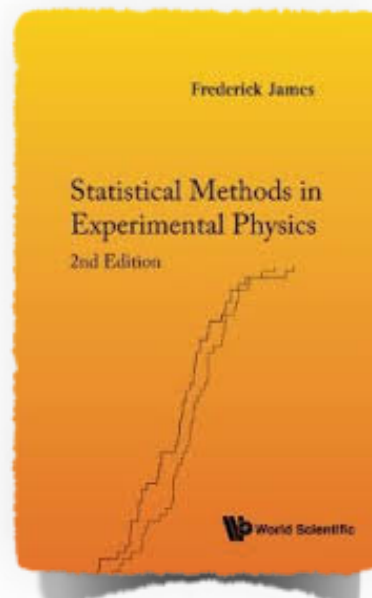


Thanks for your attention  
and your questions

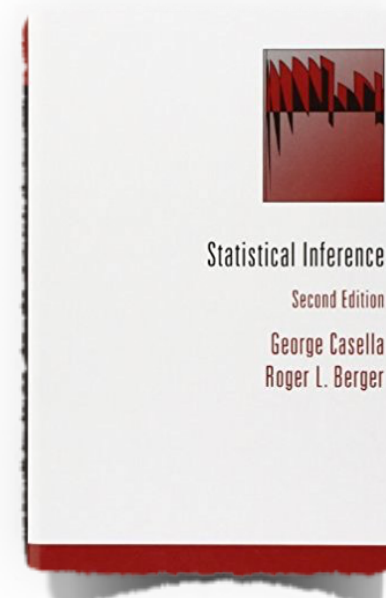
# Further readings - books



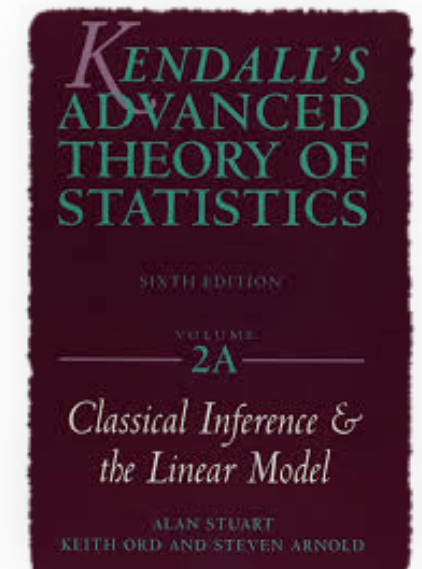
G. Cowan, "Statistical data analysis"



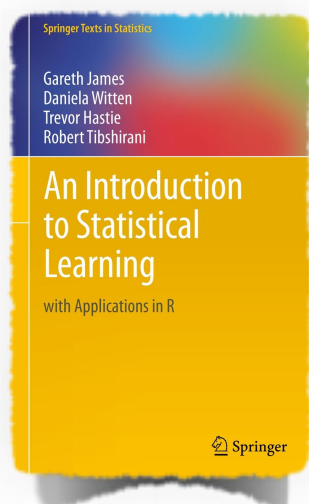
F. James, "Statistical Methods in Experimental Physics, data analysis"



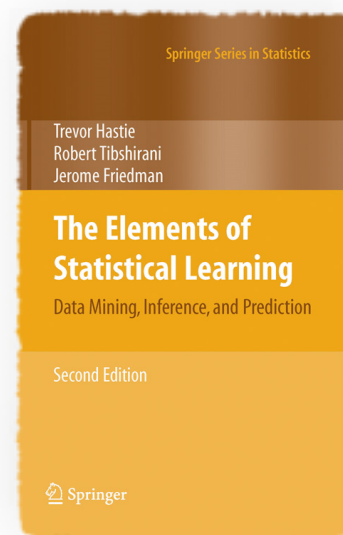
G. Casella, R. Berger, "Statistical Inference"



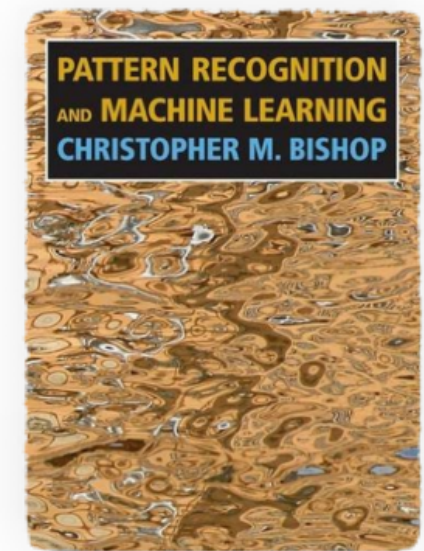
A. Stuart, et al "Kendall's Advanced Theory of Statistics Vol 2A"



T. Hastie et al., "An Introduction to Statistical learning"



T. Hastie et al., "The elements of statistical learning"



C. Bishop "Pattern recognition and machine learning"

# Further readings — slides/docs

---

- Statistics@ <http://hcpss.web.cern.ch/hcpss/> (Excellent lectures by K. Cranmer, G. Cowan, B. Cousins et al.)
- Lectures from Glen Cowan's page <https://www.pp.rhul.ac.uk/~cowan/>
- Terascale Stat School (especially 2015 F. James' lectures) <https://indico.desy.de/conferenceDisplay.py?confId=11244>
- T. Junk's lectures from [www-cdf.fnal.gov/~trj/](http://www-cdf.fnal.gov/~trj/)
- L. Lyons lectures: <https://indico.cern.ch/event/431038/>
- Notes from CDF's Statistics Committee public page <https://www-cdf.fnal.gov/physics/statistics/>
- B. Cousins' stuff: try to find his (CMS restricted) "Statistics in Theory - prelude to Statistics in Practice" lectures. Look at his statistics papers on inspire and the references he recommends.
- Proceedings/docs from the PHYSTAT conferences and workshops, linked from [phystat.org](http://phystat.org)
- IML material (<https://indico.cern.ch/category/8009/> and recent HEP-relevant resources linked from <https://github.com/iml-wg/HEP-ML-Resources#lectures>.