UiO : University of Oslo

Applications of Machine Learning in Experimental Particle Physics

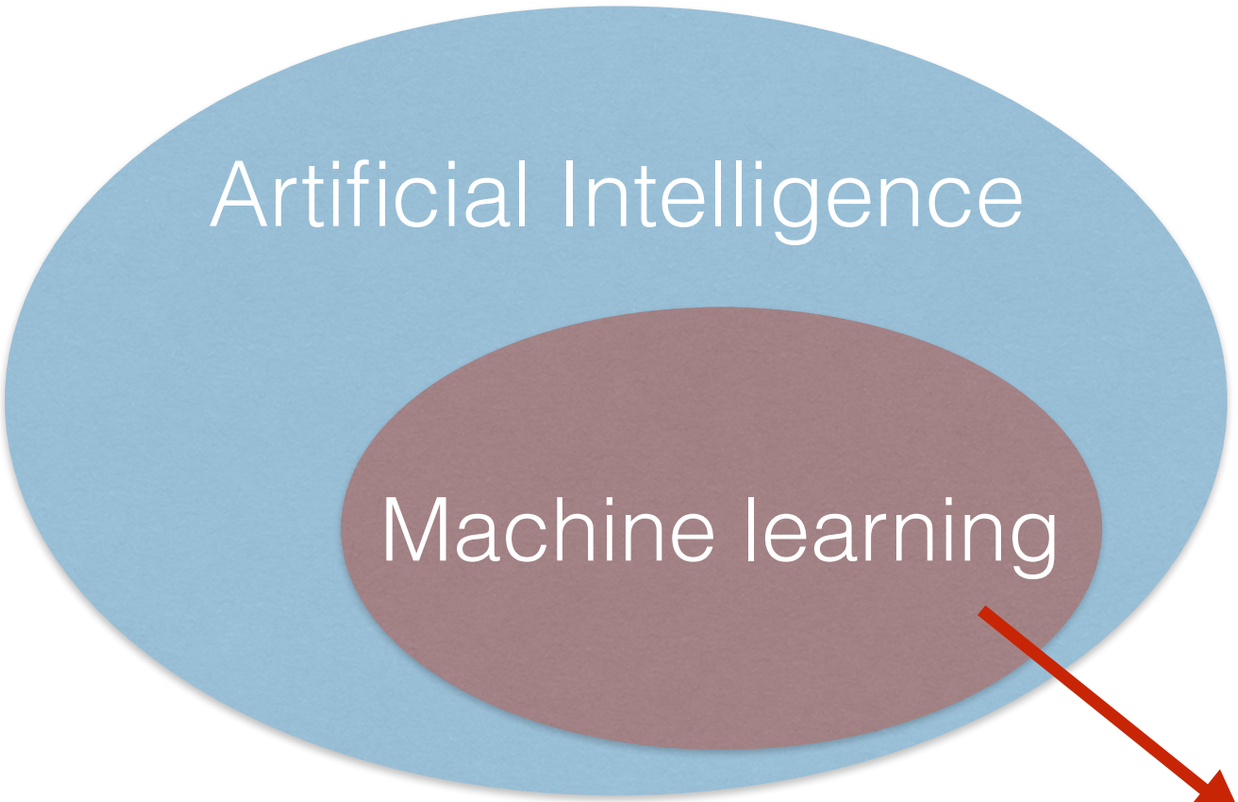
James Catmore
University of Oslo

Overview

- Review of basic machine learning concepts
- Boosted Decision Trees in detail
  - and why I think they are *awesome*
- Neural networks
- Future opportunities and challenges

What is "Machine Learning"?

- We used to call it "multivariate analysis"
- The media often call it "artificial intelligence" or A.I.



uses statistical inference to extract generalities from "training" data

→ **"learns" from the training data**

→ when exposed to new data, ***demonstrates behaviours that have not been explicitly programmed***

Supervised learning

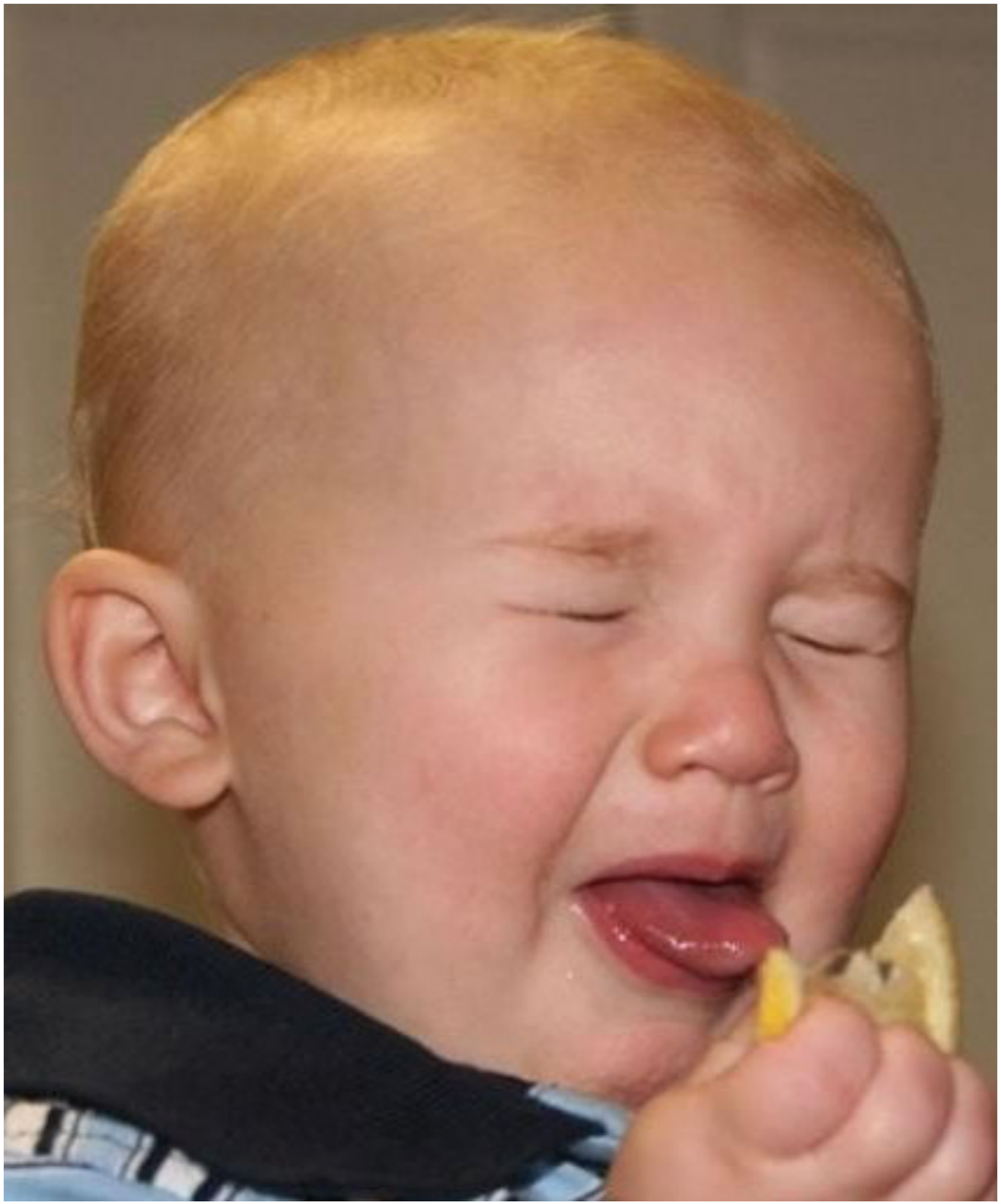
Most physics analyses

Unsupervised learning

Cluster-finding (calorimeter and inner tracker)

Reinforcement learning

No current use in physics (to my knowledge)



Types of ML algorithm

Objective function

- All machine learning algorithms are attempting to minimise an *objective function:*

$$J(\Theta) = L(\Theta) + \Omega(\Theta)$$

Loss function: how far away from the target?

Regularisation: how complex is the model?

$$L(\Theta) = L(\theta_0, \theta_1, \ldots, \theta_n) = \frac{1}{2m} \sum_{i=1}^{m} \left(h(x^{(i)}) - y^{(i)} \right)^2$$

Model parameters: these are what we change during training

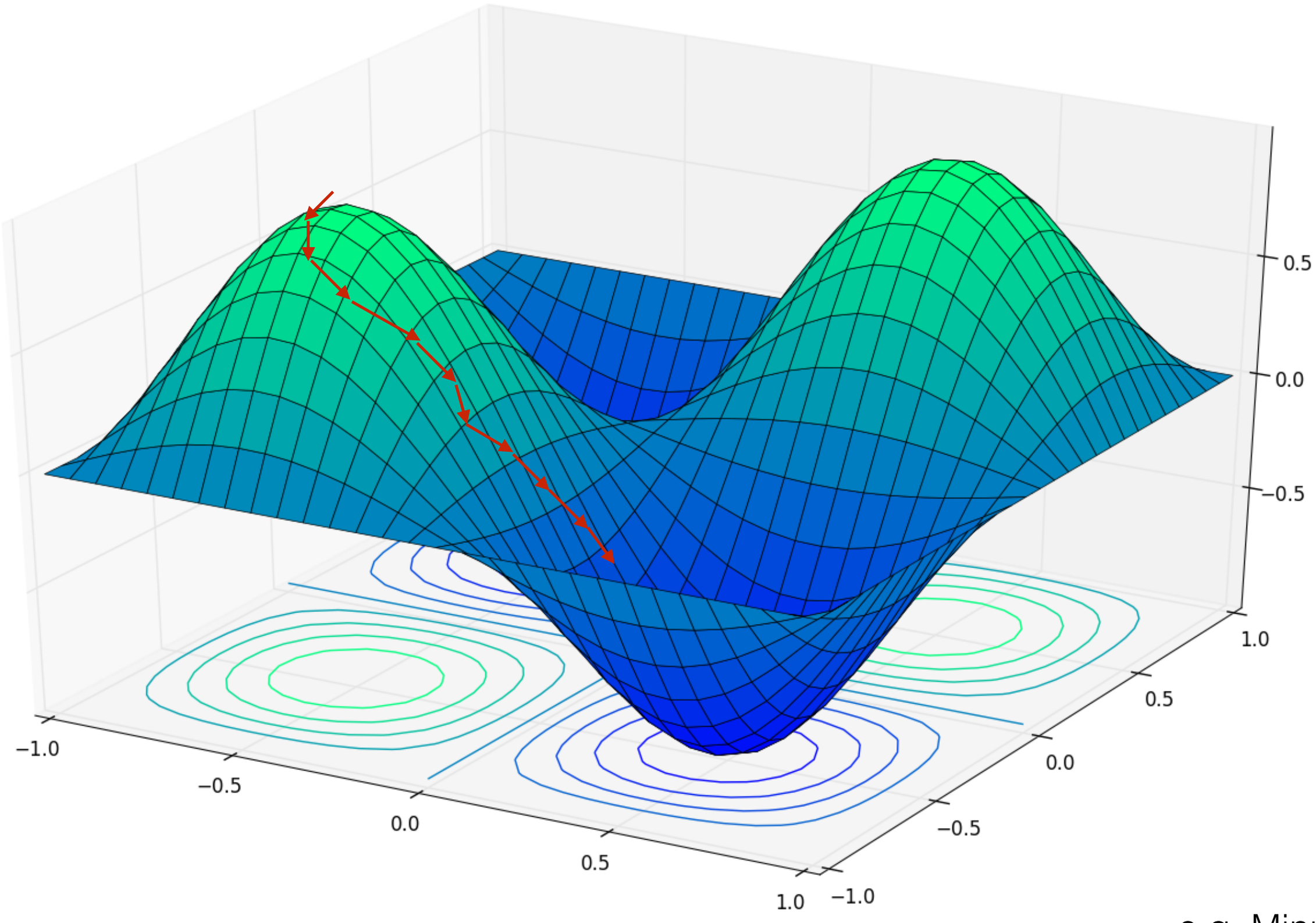
Mean squared error

Model

Training data

Target

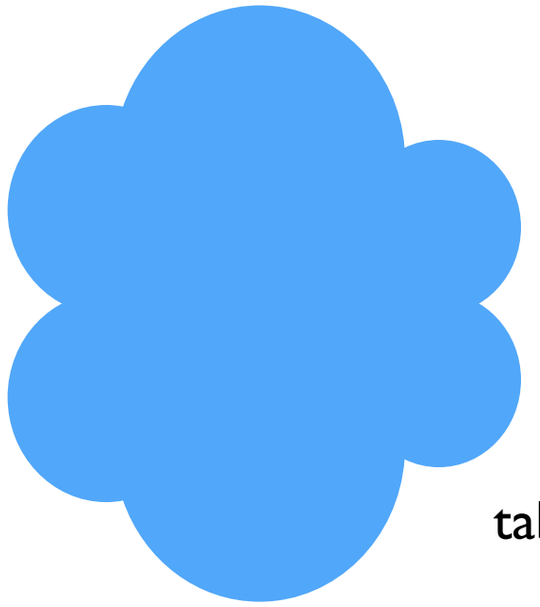
Method of Steepest Descent (a.k.a. Gradient Descent)



e.g. Minuit…

How supervised machine learning works

- Assume there is some data generating process that we wish to understand and/or whose behaviour we wish to be able to predict



PREPROCESSING

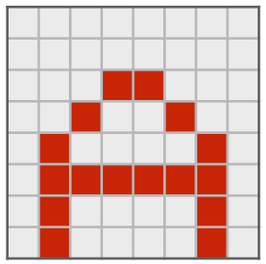
also… data preparation…
tabulation… feature extraction…

	var 1	var 2	…	var M
1	#	#	…	#
2	#	#	…	#
3	#	#	…	#
4	#	#	…	#
5	#	#	…	#
6	#	#	…	#
7	#	#	…	#
…	…	…	…	…
N	#	#	…	#

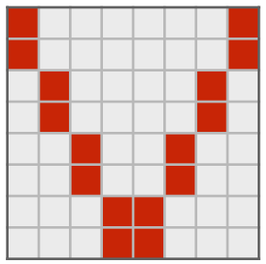
- Assume also that we are able to somehow **label** some of the data being produced…

Labelling examples

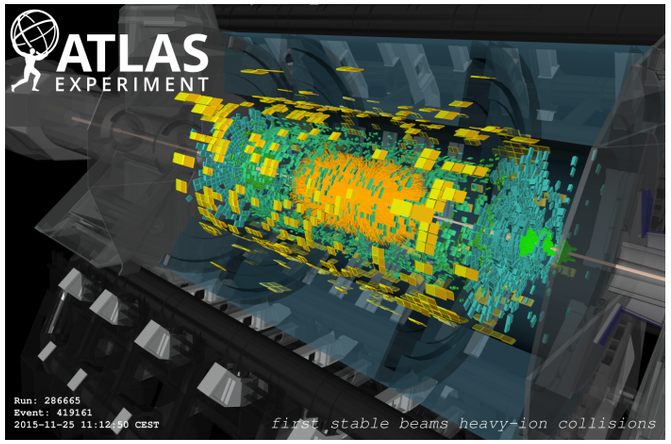
CLASSIFICATION



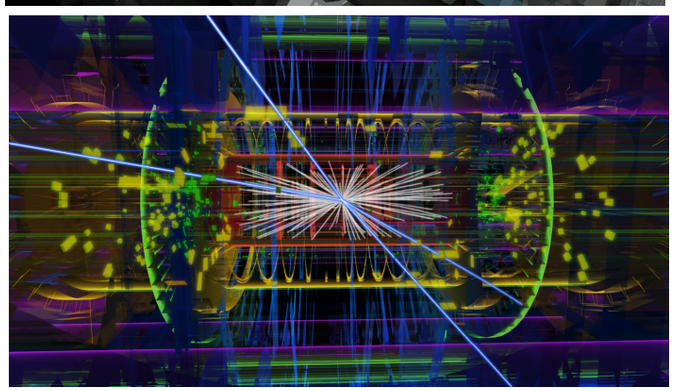
“A”



“V”



“Signal”

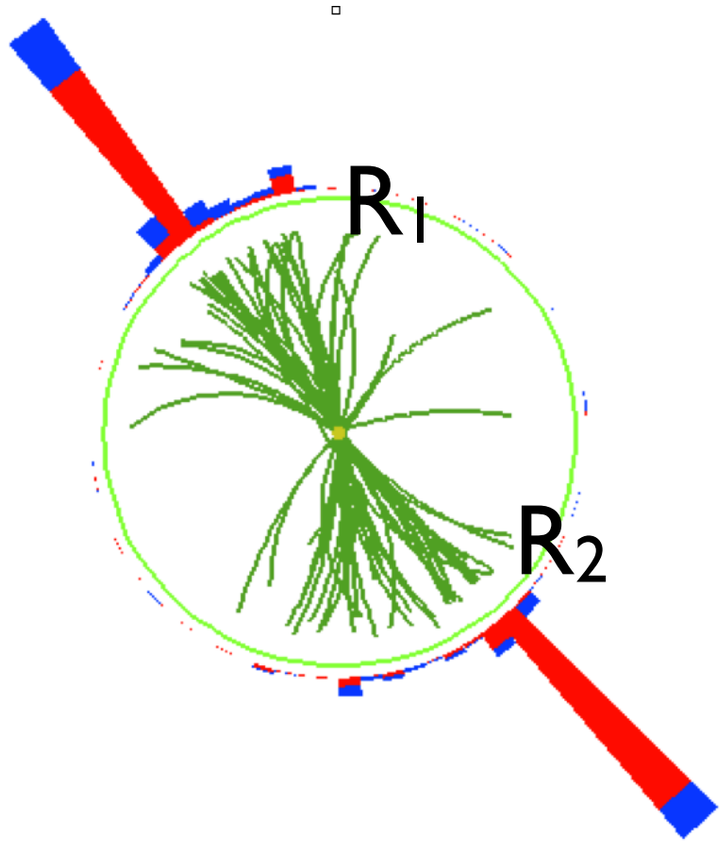


“Background”

REGRESSION

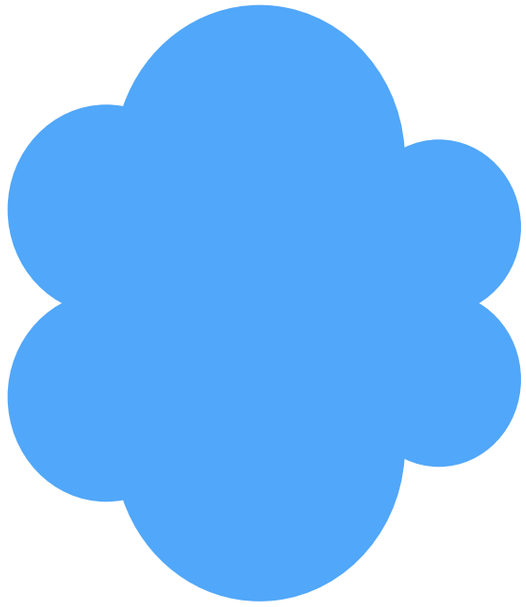
NOK 8 600 000

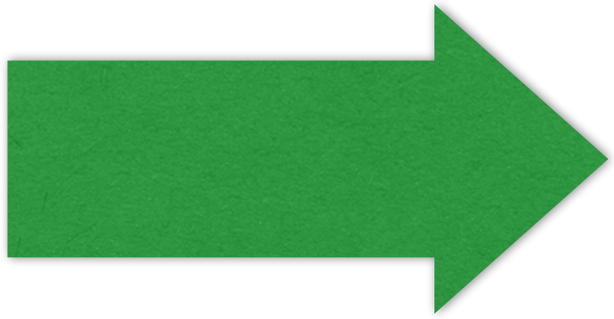
NOK 15 200 000



How supervised machine learning works

- Suppose that we collect and (somehow) label a batch of data from the generating process





	var 1	var 2	…	var M	**LABEL**
1	#	#	…	#	#
2	#	#	…	#	#
3	#	#	…	#	#
4	#	#	…	#	#
5	#	#	…	#	#
6	#	#	…	#	#
7	#	#	…	#	#
…	…	…	…	…	…
N	#	#	…	#	#

- Let us split this sample into three pieces

Training sample

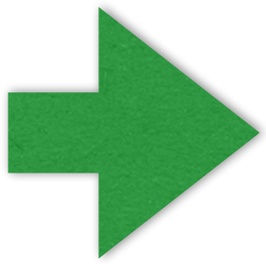
Validation sample

Evaluation sample

How supervised machine learning works

TRAINING

var 1	var 2	…	var M	**LABEL**
#	#	…	#	#
#	#	…	#	#
#	#	…	#	#
#	#	…	#	#
#	#	…	#	#
#	#	…	#	#
#	#	…	#	#
#	#	…	#	#
…	…	…	…	…
#	#	…	#	#



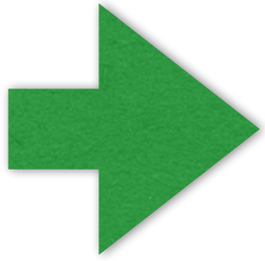
ALGORITHM

Hyperparameters

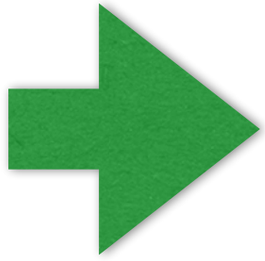
How supervised machine learning works

VALIDATION

var 1	var 2	…	var M
#	#	…	#
#	#	…	#
#	#	…	#
#	#	…	#
#	#	…	#
#	#	…	#
#	#	…	#
…	…	…	…
#	#	…	#



TRAINED ALGORITHM



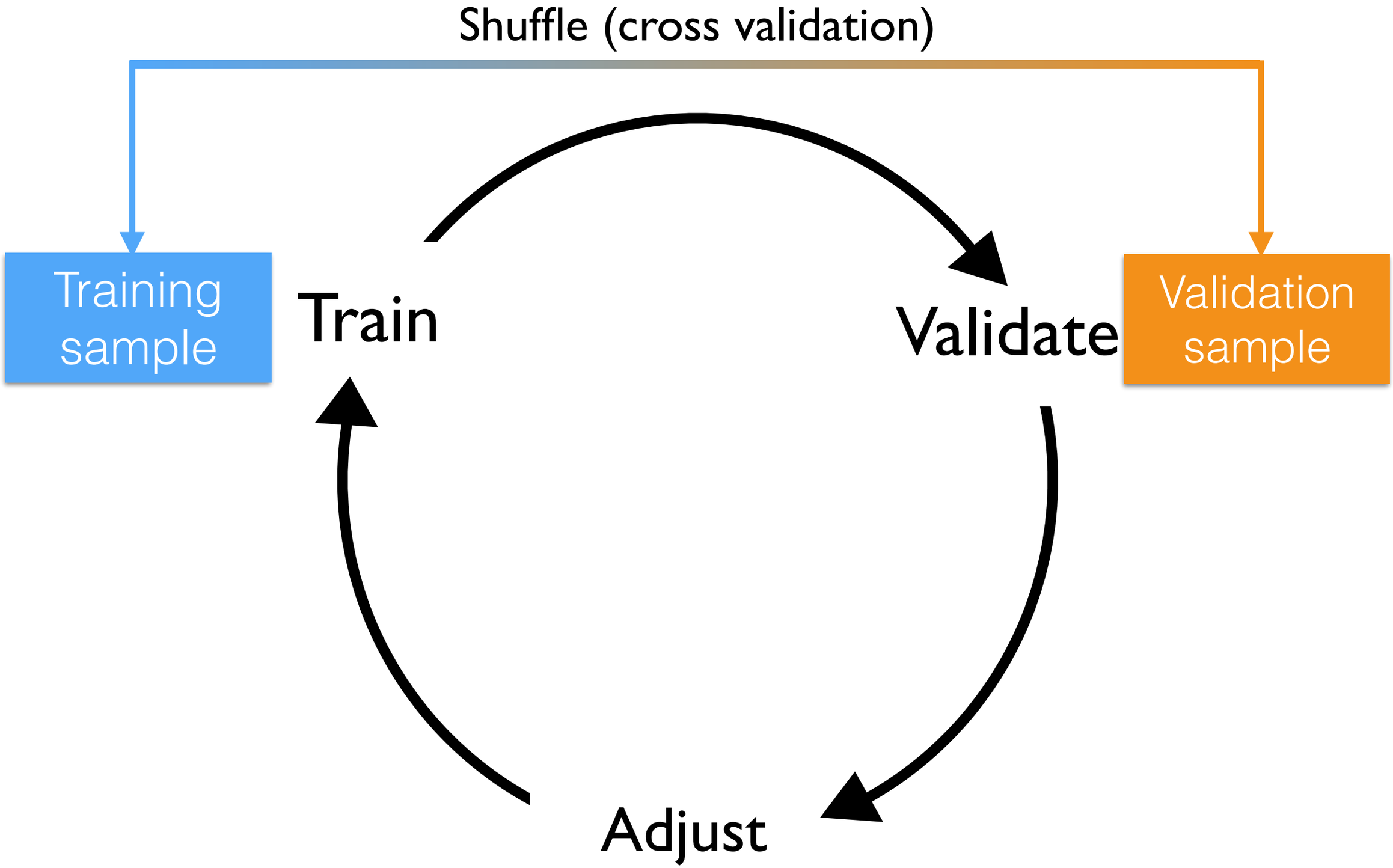
PREDICTION
#
#
#
#
#
#
#
…
#

Comparison between the prediction and labels allows a determination of the algorithm performance

LABEL
#
#
#
#
#
#
#
#
…
#

Hyperparameters can be adjusted and the training repeated

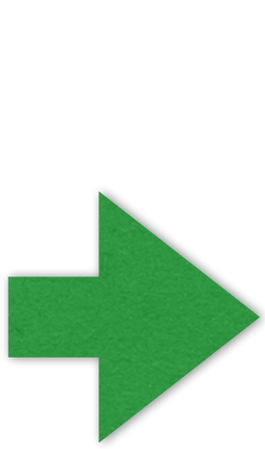
HYPERPARAMETER TUNING



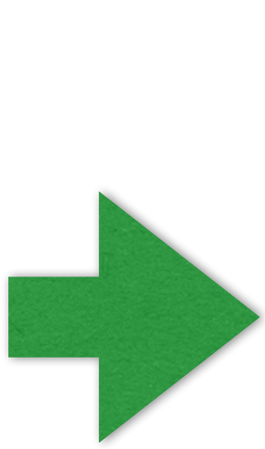
How supervised machine learning works

PERFORMANCE EVALUATION

var 1	var 2	…	var M
#	#	…	#
#	#	…	#
#	#	…	#
#	#	…	#
#	#	…	#
#	#	…	#
#	#	…	#
…	…	…	…
#	#	…	#



TRAINED ALGORITHM

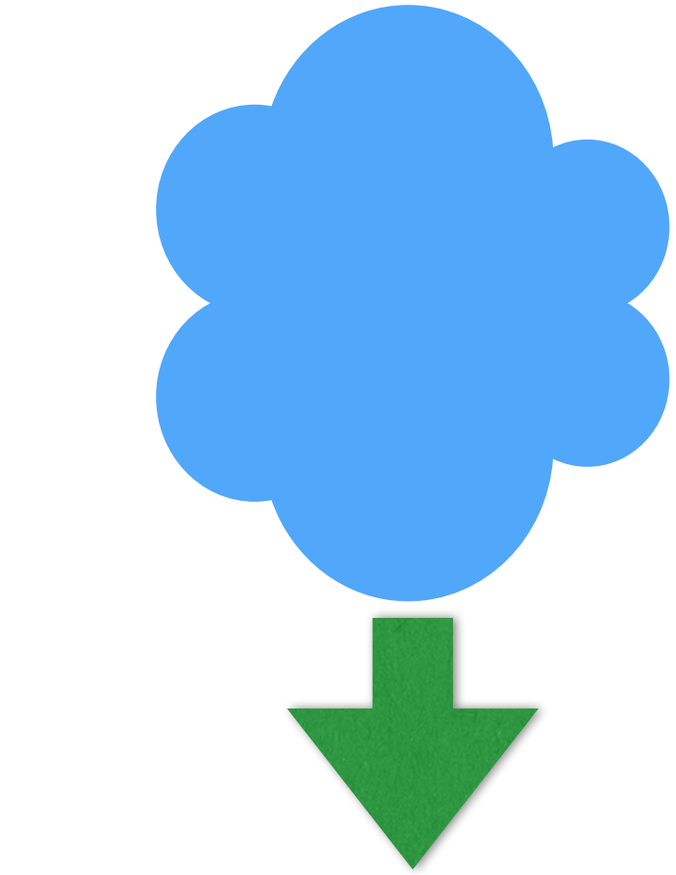


PREDICTION
#
#
#
#
#
#
#
…
#

LABEL
#
#
#
#
#
#
#
…
#

No further adjustments permitted without fresh training data.
The performance as evaluated in this step is assumed
to be the working performance of the algorithm

DEPLOYMENT



	var 1	var 2	…	var M
1	#	#	…	#
2	#	#	…	#
3	#	#	…	#
4	#	#	…	#
5	#	#	…	#
6	#	#	…	#
7	#	#	…	#
…	…	…	…	…
X	#	#	…	#

New unlabelled data

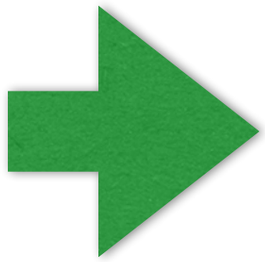
TRAINED ALGORITHM

The predictions are now used for practical purposes

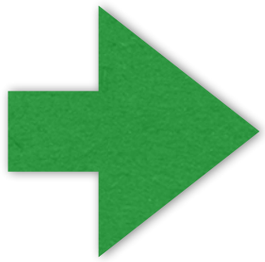
PREDICTION
#
#
#
#
#
#
#
#
…
#

Projection to a single variable

	var 1	var 2	…	var M
1	#	#	…	#
2	#	#	…	#
3	#	#	…	#
4	#	#	…	#
5	#	#	…	#
6	#	#	…	#
7	#	#	…	#
…	…	…	…	…
X	#	#	…	#



TRAINED ALGORITHM



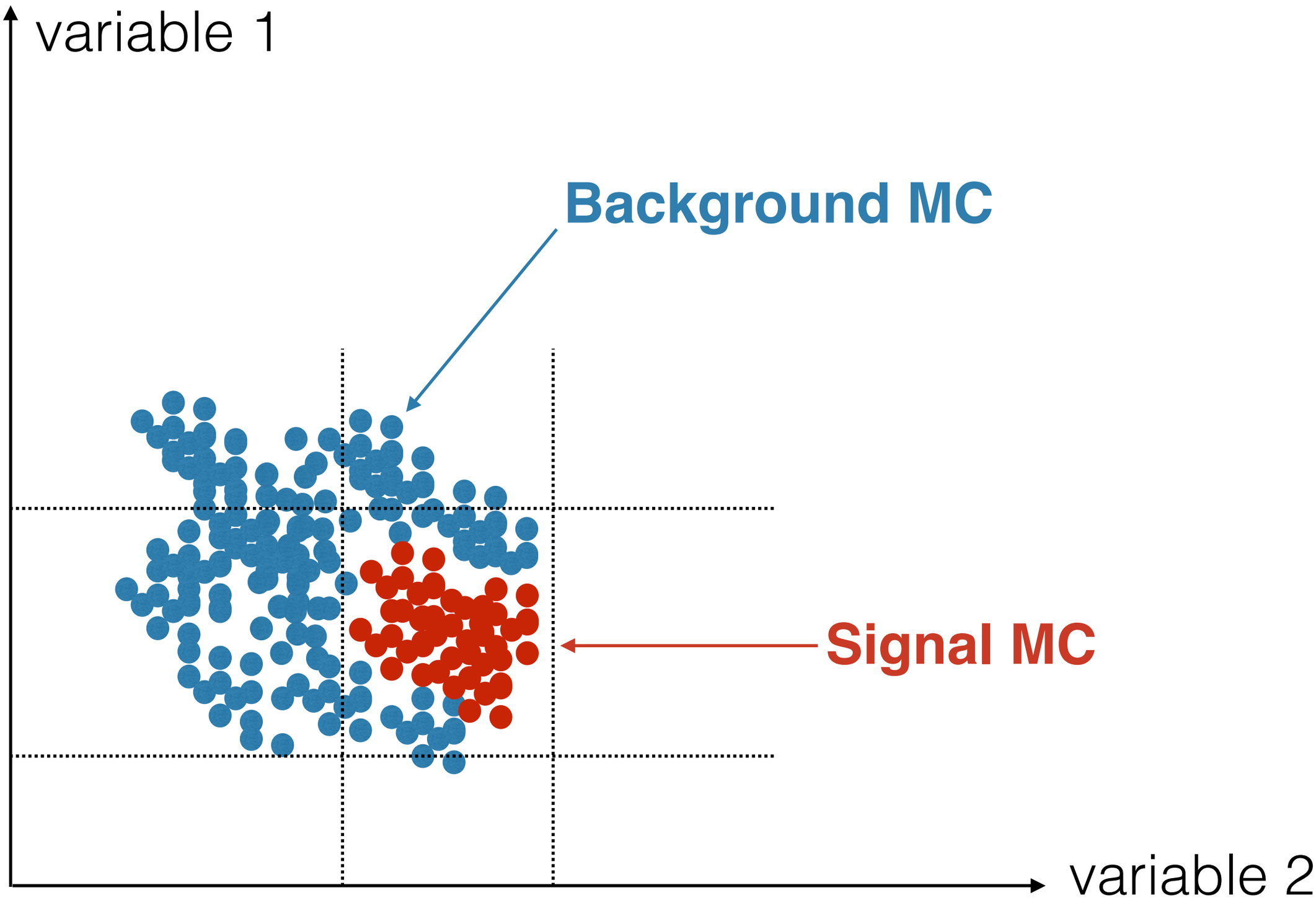
PREDICTION
#
#
#
#
#
#
#
…
#

M variables ⟶ 1 variable

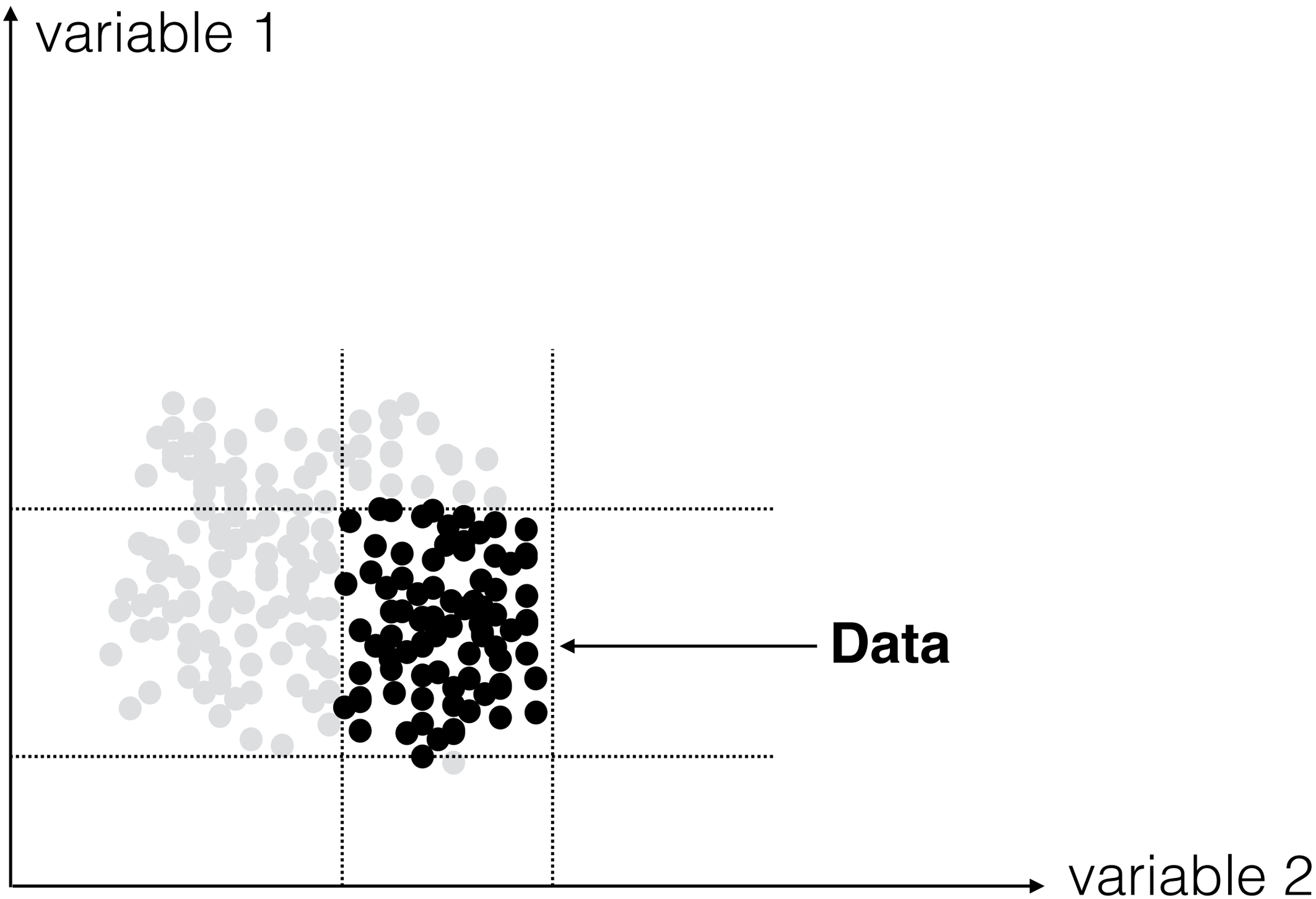
$$f(x_1, x_2, \ldots, x_M) = y$$

The algorithm learns how to project M variables into a single variable

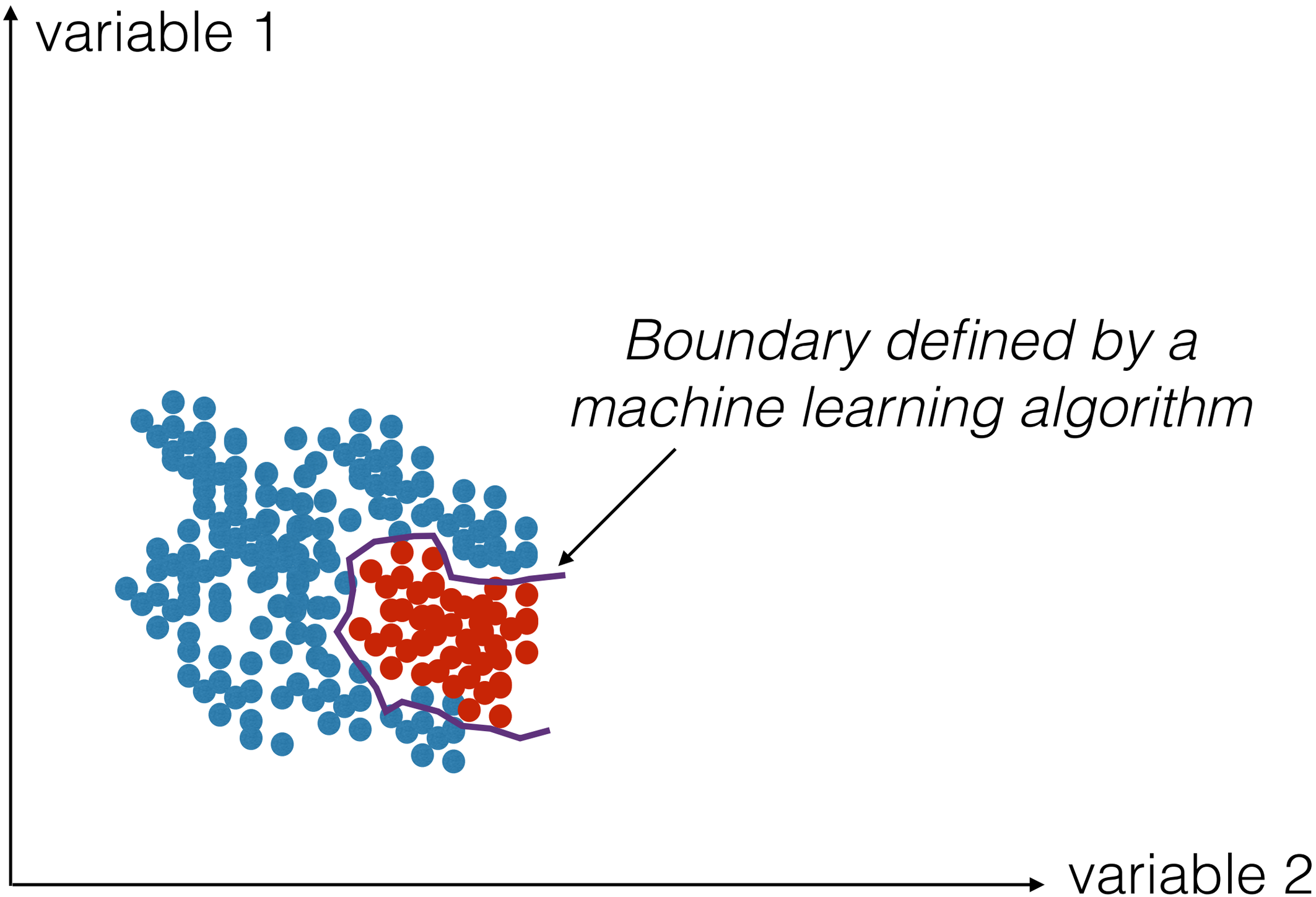
Physics analysis: “rectangular cuts”



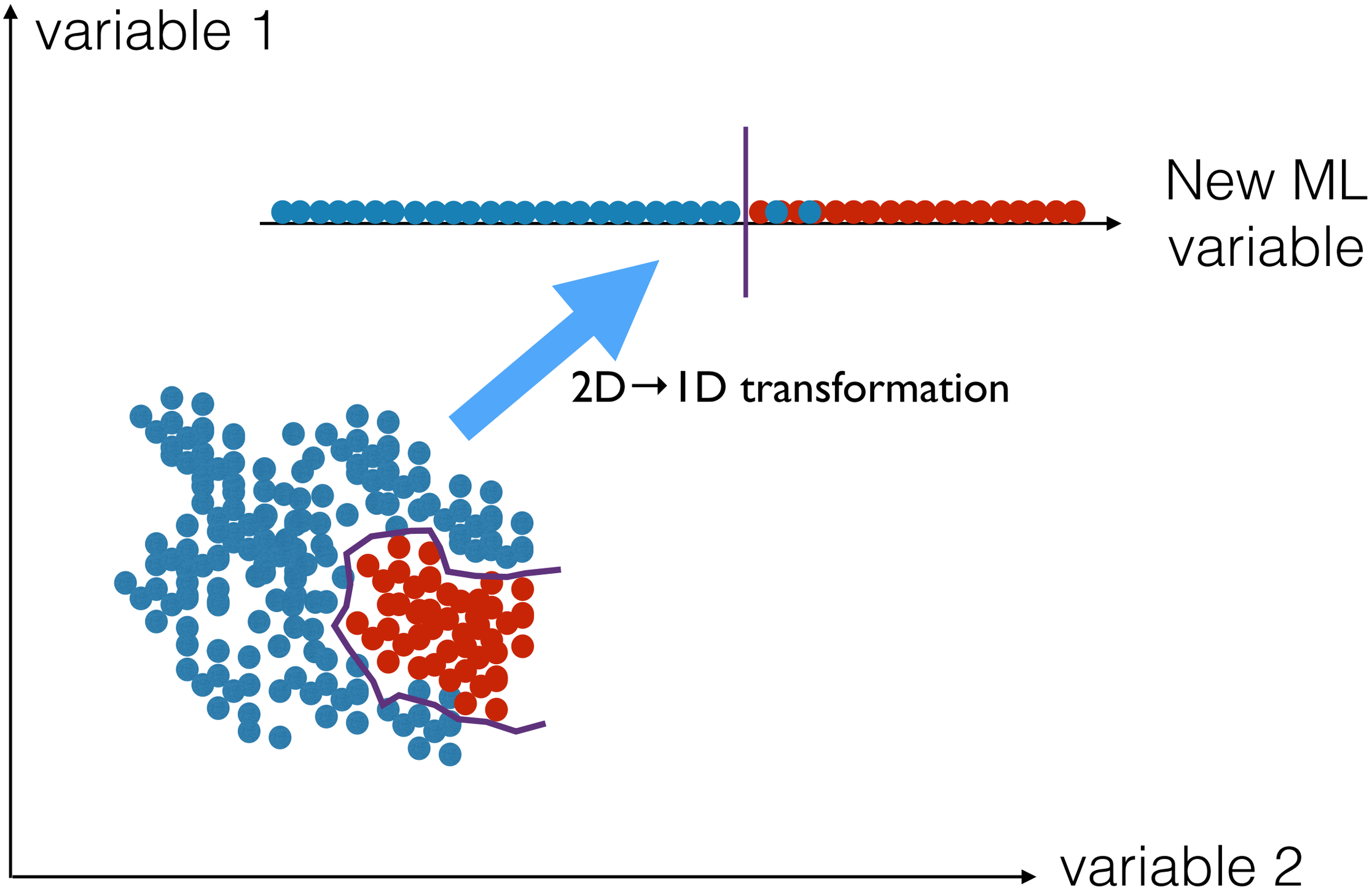
Physics analysis: “rectangular cuts”



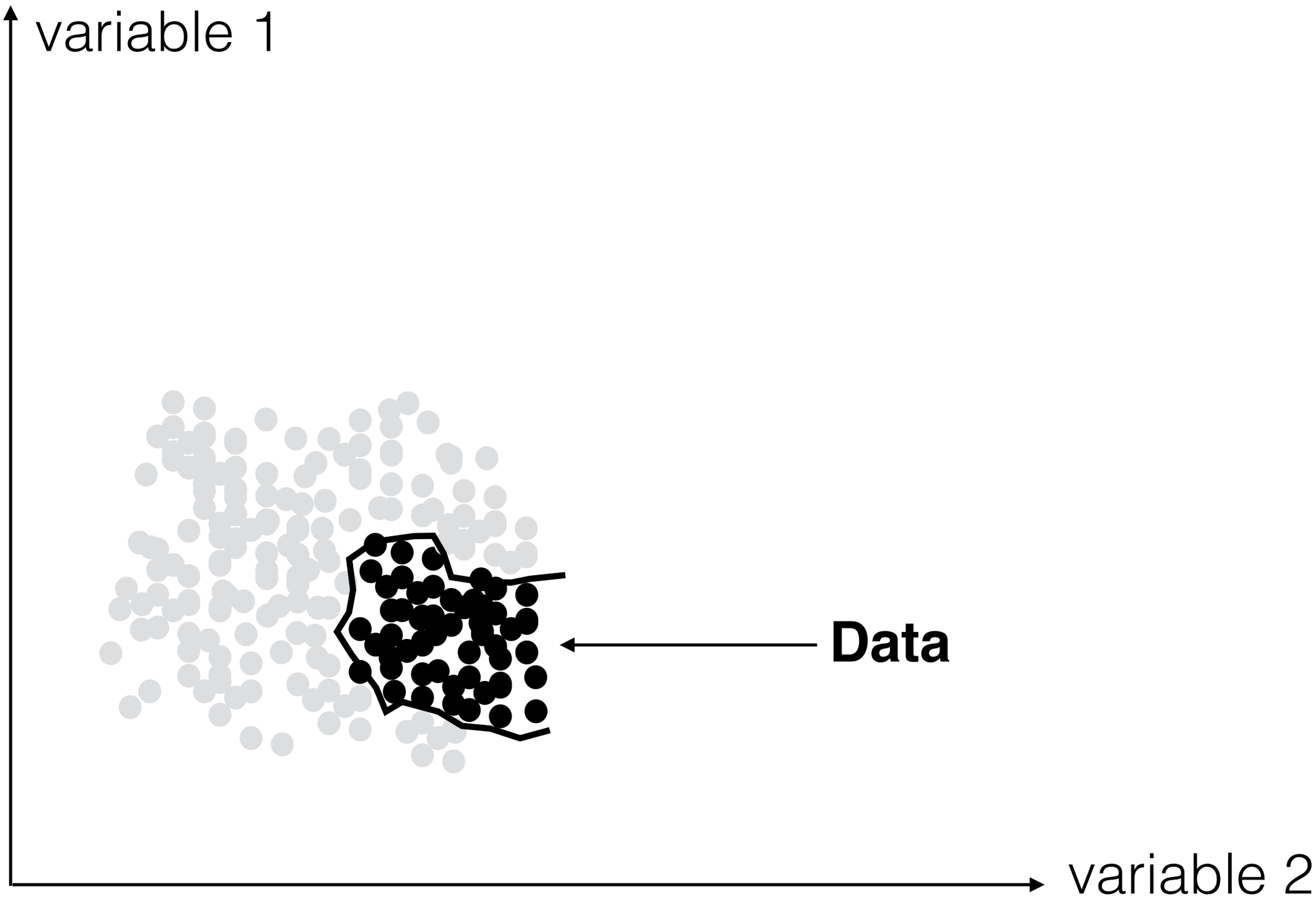
Physics analysis: multi-variate (ML)



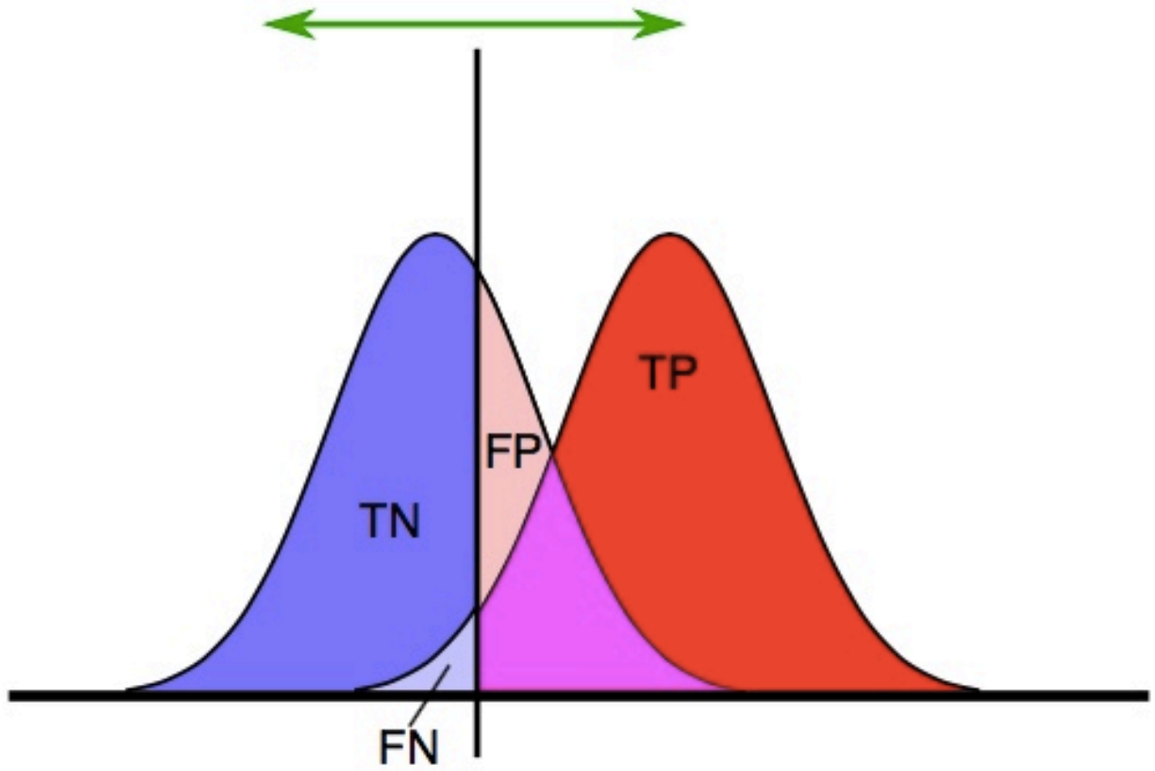
Physics analysis: multi-variate (ML)



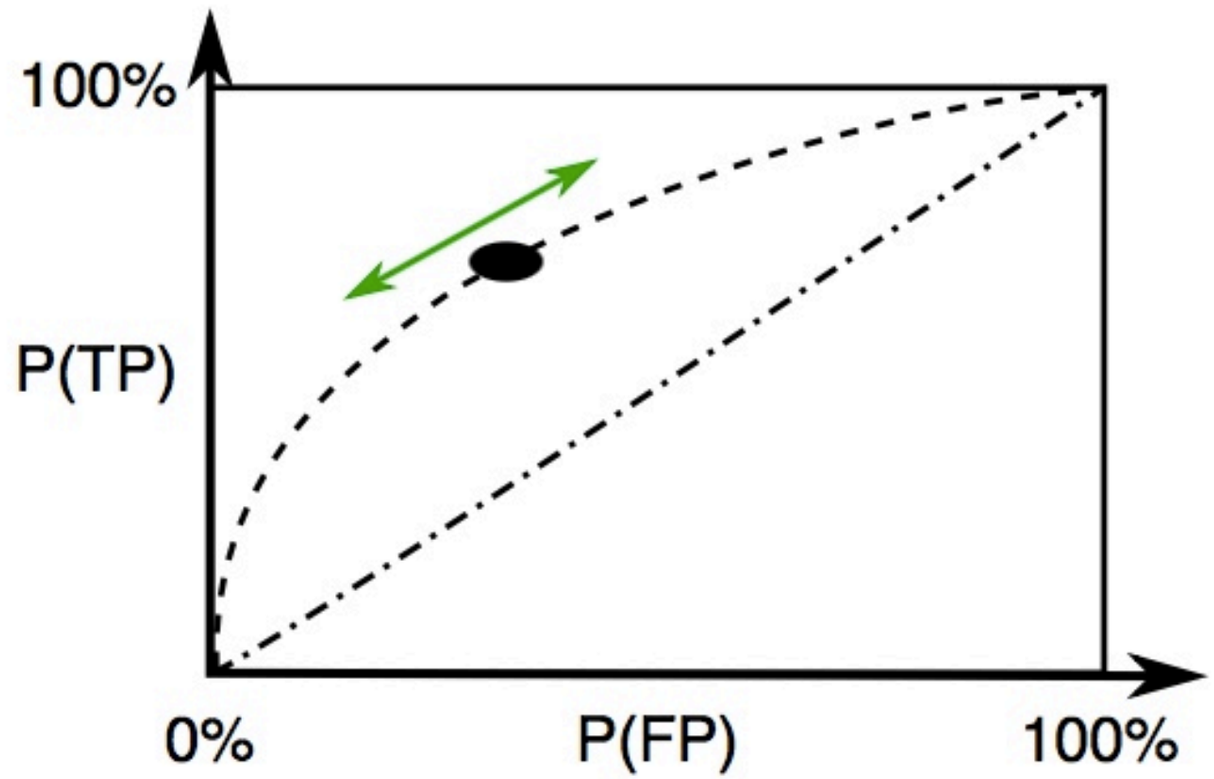
Physics analysis: multi-variate (ML)

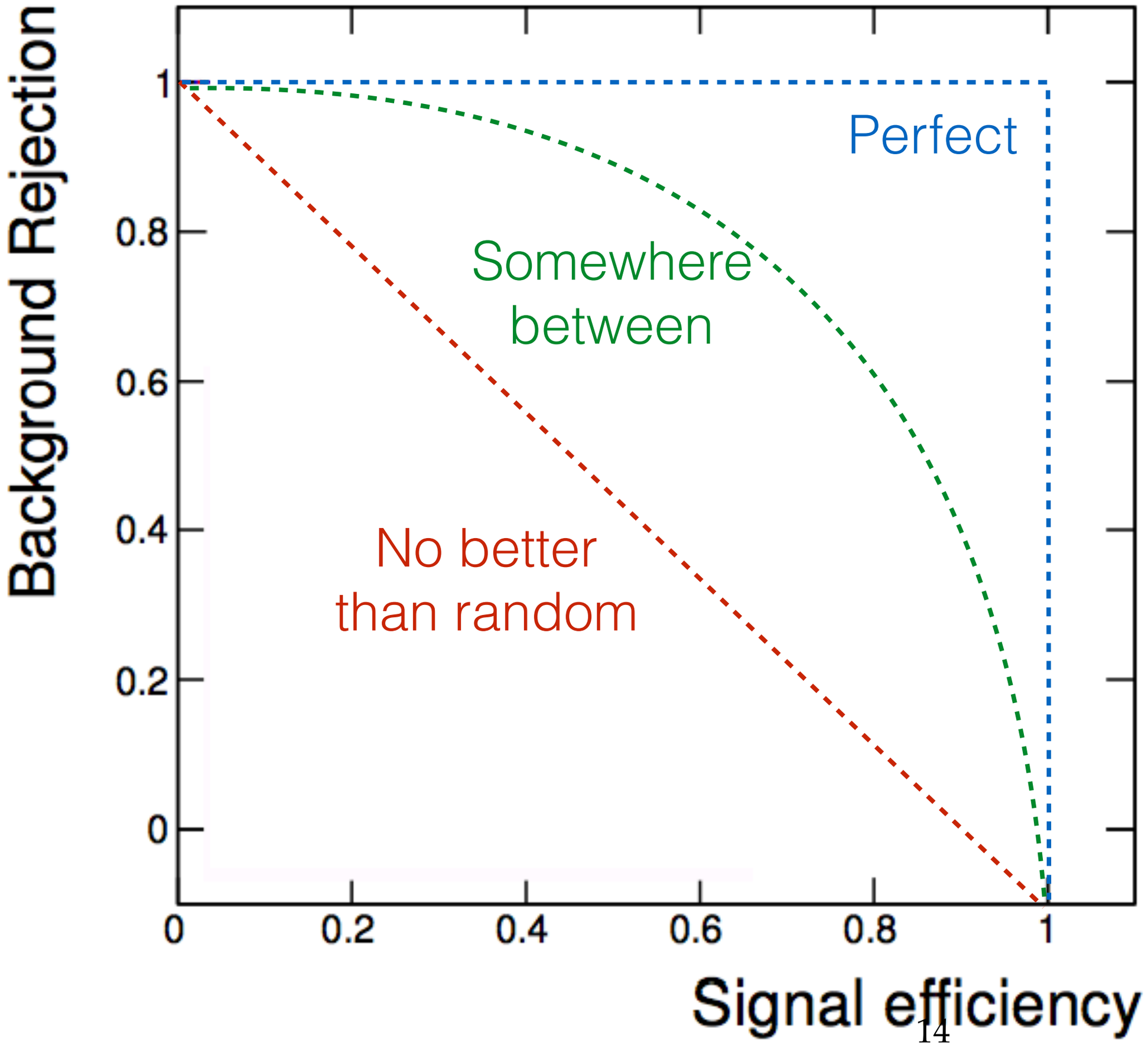


Performance evaluation: Receiver-Operator Curve

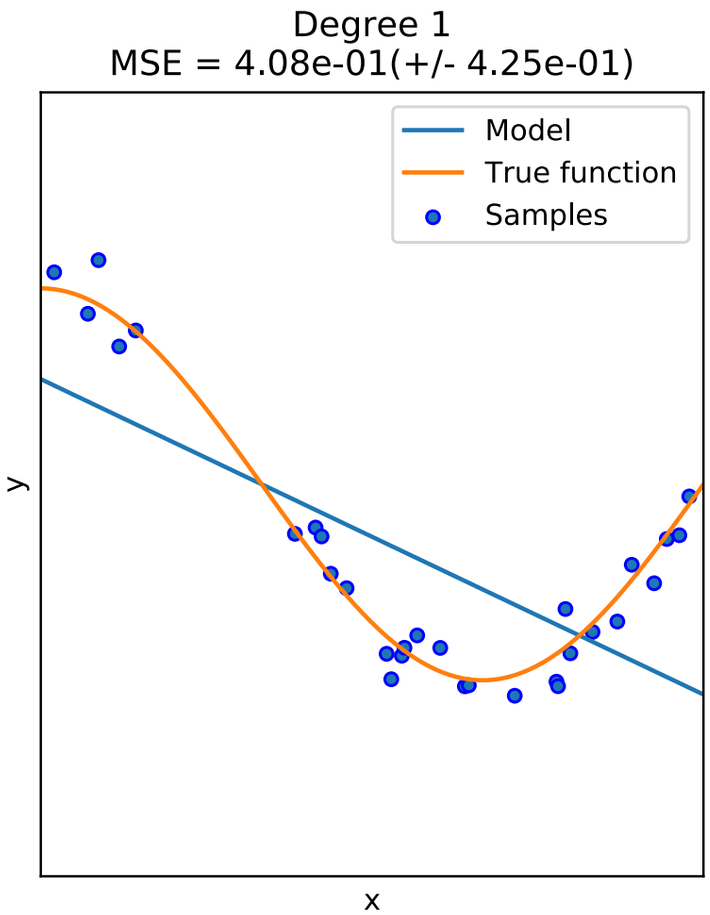


TP	FP
FN	TN

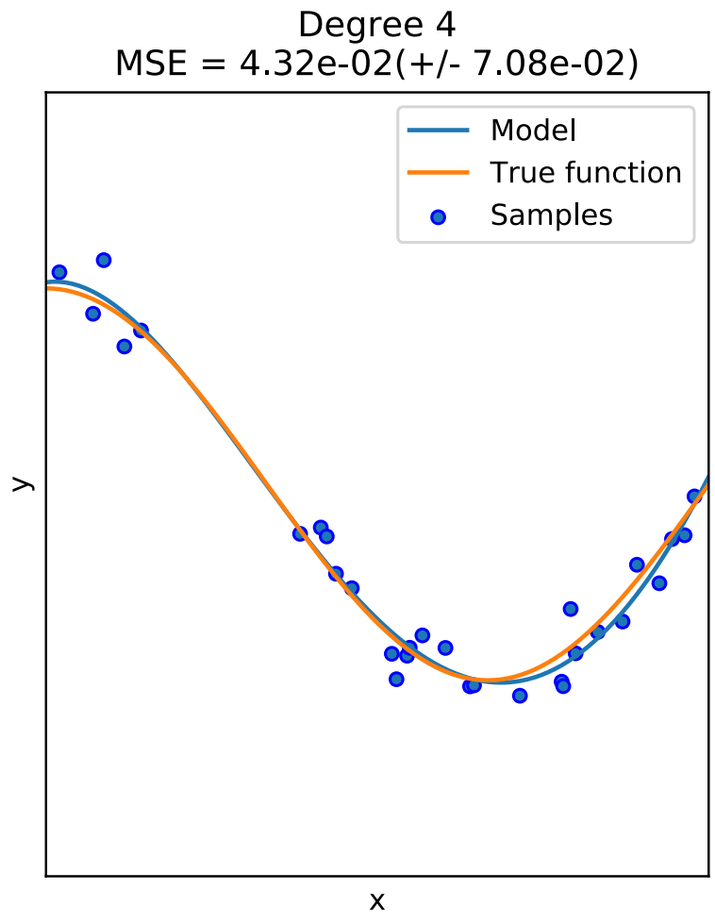



Background Rejection
1
0.8
0.6
0.4
0.2
0
0
0.2
0.4
0.6
0.8
1
Signal efficiency
Perfect
Somewhere
between
No better
than random

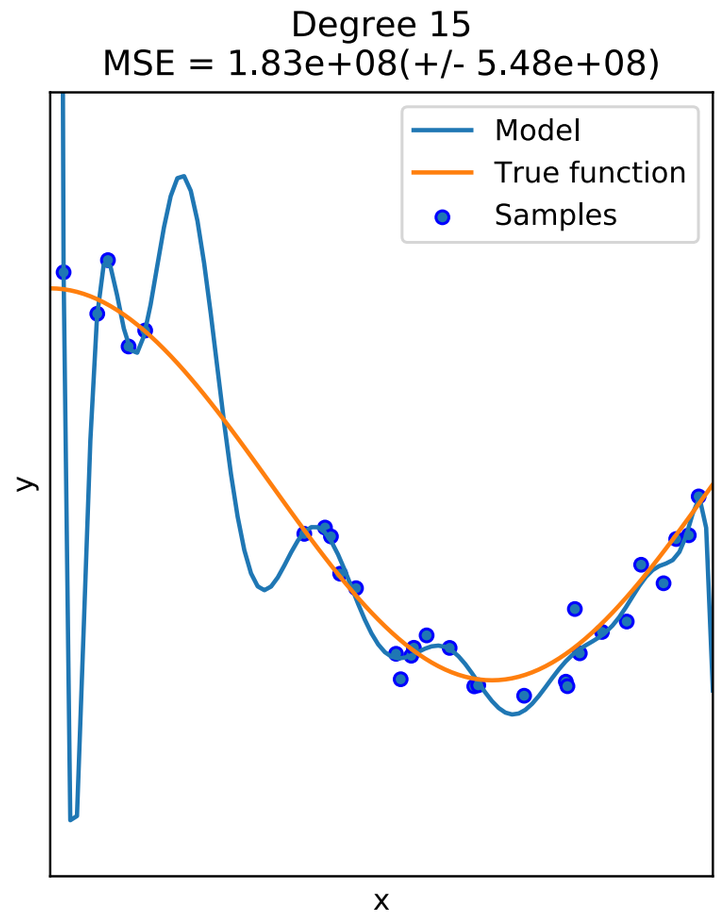
Overfitting and bias vs variance



Underfitted:
model is too simple
(high bias, low variance)



Appropriate

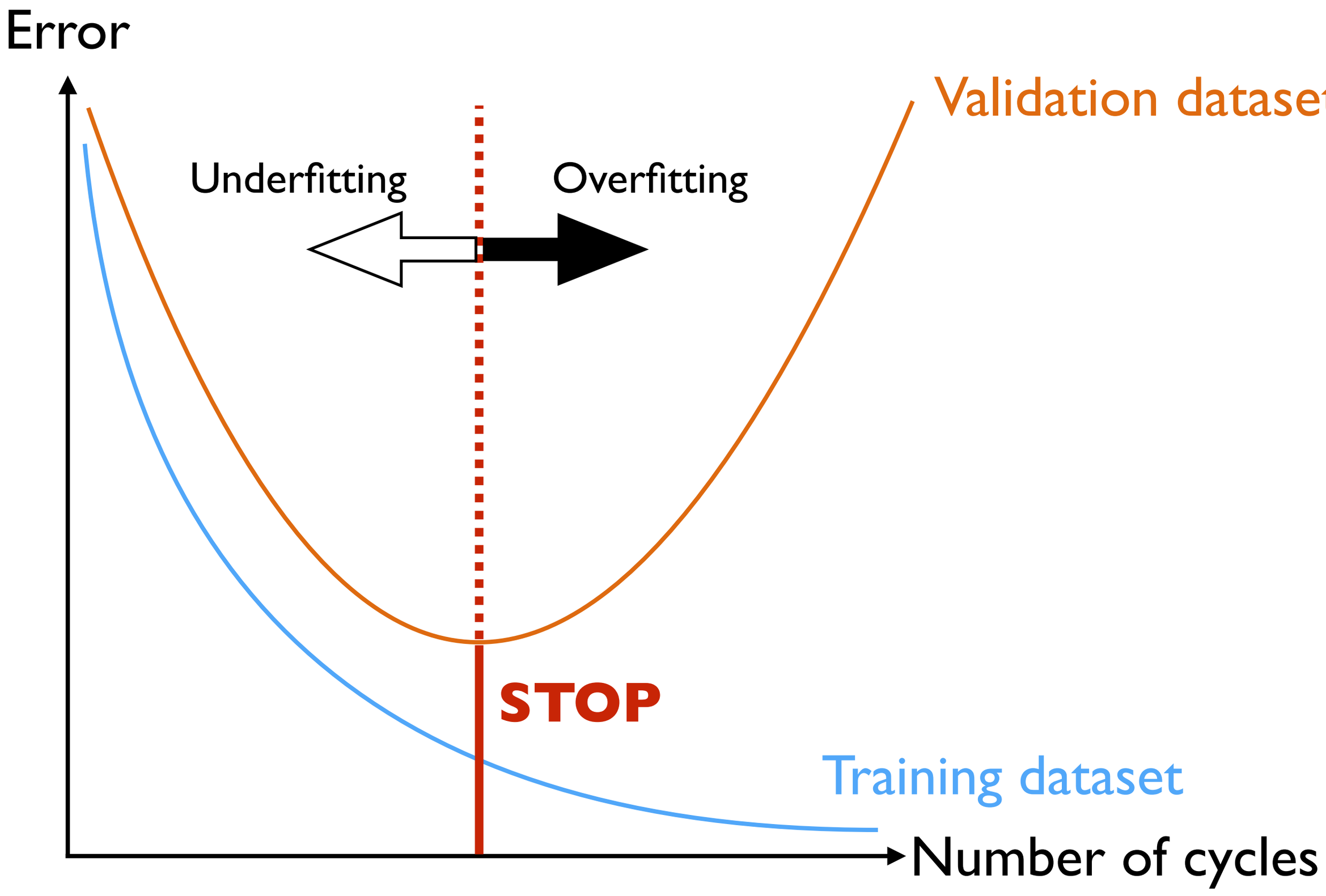


Overfitted:
model is too complex
(low bias, high variance)

$$J(\Theta) = L(\Theta) + \Omega(\Theta)$$

Regularisation is the means of avoiding overtraining: imposes a penalty for complexity

Overfitting and bias vs variance



Exemplary achievements of multivariate techniques in HEP

Top quark mass measurement @ Tevatron	Shallow NNs, BDTs
Single top quark discovery @ Tevatron	Shallow NNs, BDTs
Higgs discovery ($H \to \gamma\gamma$) @ CMS	BDT
Observation of $H \to bb$ @ ATLAS, CMS	BDT
Observation of $B_s \to \mu\mu$ @ ATLAS, CMS, LHCb	BDT
Observation of associated Higgs and top quark pair production (“ttH”) @ ATLAS, CMS	BDT (XGBoost @ ATLAS)
Jet flavour tagging	Boosted decision trees (BDT), shallow neural networks (NN), recurrent NN

Software commonly used in HEP

Package	Description	Data structure	ROOT integration?	Download from…
TMVA	ML framework + algorithms	ROOT TTrees	Yes	Ships with ROOT (root.cern)
SciKitLearn	ML framework + algorithms	NumPy arrays / Pandas dataframe	Via pyROOT	http://scikit-learn.org or via Anaconda
Keras	Wrapper for NNs - TensorFlow and Theano	NumPy arrays & Pandas dataframe	Via pyROOT	https://keras.io or via Anaconda

- The *frameworks* offer tools for making performance plots, organising data, cross validation etc.
  - They have built-in algs but these may not be the best for a given task - be prepared to plug in external applications from outside our community
- Divide between the ROOT and Python ecosystems becomes less important as Python-driven ROOT becomes more advanced
  - Uproot: convert ROOT files to Pandas dataframes without a ROOT installation
- If you are using the Python ecosystem, use Anaconda rather than installing each package independently
- Data structures are a fascinating topic by themselves - in particular our use of "ragged arrays" is bizarrely rare in the wider world. See (e.g.) this talk at CHEP2018

Inter-experiment Machine Learning group (IML)

- 3rd annual meeting: agenda
- Wide variety of topics reflecting the growing interest in machine learning in our field
- A few particular highlights (my choice)
  - Conceptual overview of ML in HEP (S. Gleyzer)
  - Statistical and information theory foundation of deep learning (N. Tishby)
  - Future directions for HEP (K. Cranmer)
  - Handling uncertainties with adversarial training (P. Galler)
  - Decoding physics information from deep NNs (T. Cheng)
  - Tracking machine learning challenge (D. Rousseau)
  - Containers and machine learning (M. Feickert)

Open data portal

- For those of us in an experiment there is no shortage of data and simulation to play with
- But getting it into a reasonable shape can be a lot of work
- Those who are not members of a big experiment may struggle to get enough data to try out their new ideas

- Solution: Open Data Portal - http://opendata.cern.ch
  - More than 1 PB of data and simulation for the general public to explore
  - Ideal for educational purposes as well!
- Examples in this talk prepared with the HiggsML challenge dataset:
  - http://opendata.cern.ch/record/328
  - Jupyter notebooks (prepared jointly with Eirik Gramstad for training new students in Oslo) - specifically BDT tutorial

Boosted Decision Trees

Decision trees

Consider some labelled multi-variate data in two categories, signal and background

To build a decision tree:

	var 1	var 2	…	var M	**LABEL**
1	#	#	…	#	S
2	#	#	…	#	S
3	#	#	…	#	B
4	#	#	…	#	B
5	#	#	…	#	B
6	#	#	…	#	S
7	#	#	…	#	B
…	…	…	…	…	…
N	#	#	…	#	B

I Cut the data using the variable that best separates the signal from background

	var 17	**LABEL**
1	#	S
2	#	S
3	#	B
4	#	B
5	#	B
6	#	S
7	#	B
…	…	…
N	#	B

var17 > X

S	S	S	S	S	B
S	S	S	S	S	B
S	S	S	S	B	B
S	S	S	S	B	B
S	S	S	S	B	B

var17 < X

B	B	B	B	B	S
B	B	B	B	B	S
B	B	B	B	B	S
B	B	B	B	S	S
B	B	B	B	S	S

Decision trees

2 Repeat with the two separated portions of data

S	S	S	S	S	B
S	S	S	S	S	B
S	S	S	S	B	B
S	S	S	S	B	B
S	S	S	S	B	B

var10 > Y

S	S	S	S
S	S	S	S
S	S	S	S
S	S	S	S
S	S	S	B

var10 < Y

S	B
B	B
B	B
B	B
B	B

B	B	B	B	B	S
B	B	B	B	B	S
B	B	B	B	B	S
B	B	B	B	S	S
B	B	B	B	S	S

var3 > Z

S	B
S	S
S	S
S	S
S	S

var3 < Z

B	B	B	S
B	B	B	S
B	B	B	B
B	B	B	B
B	B	B	B

3 Continue until the separation between S and B reaches the required level

Decision trees

4 Use the trained tree (= cuts set) to separate unlabelled data

	var 1	var 2	…	var M
1	#	#	…	#
2	#	#	…	#
3	#	#	…	#
4	#	#	…	#
…	…	…	…	…
N	#	#	…	#

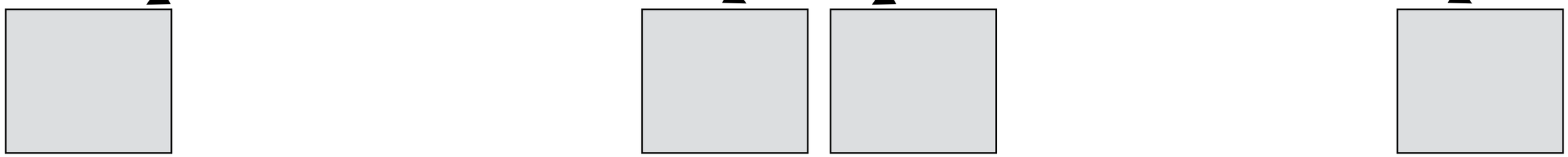
var17 > X

	var 1	var 2	…	var M
…	#	#	…	#
…	#	#	…	#
…	#	#	…	#

var17 < X

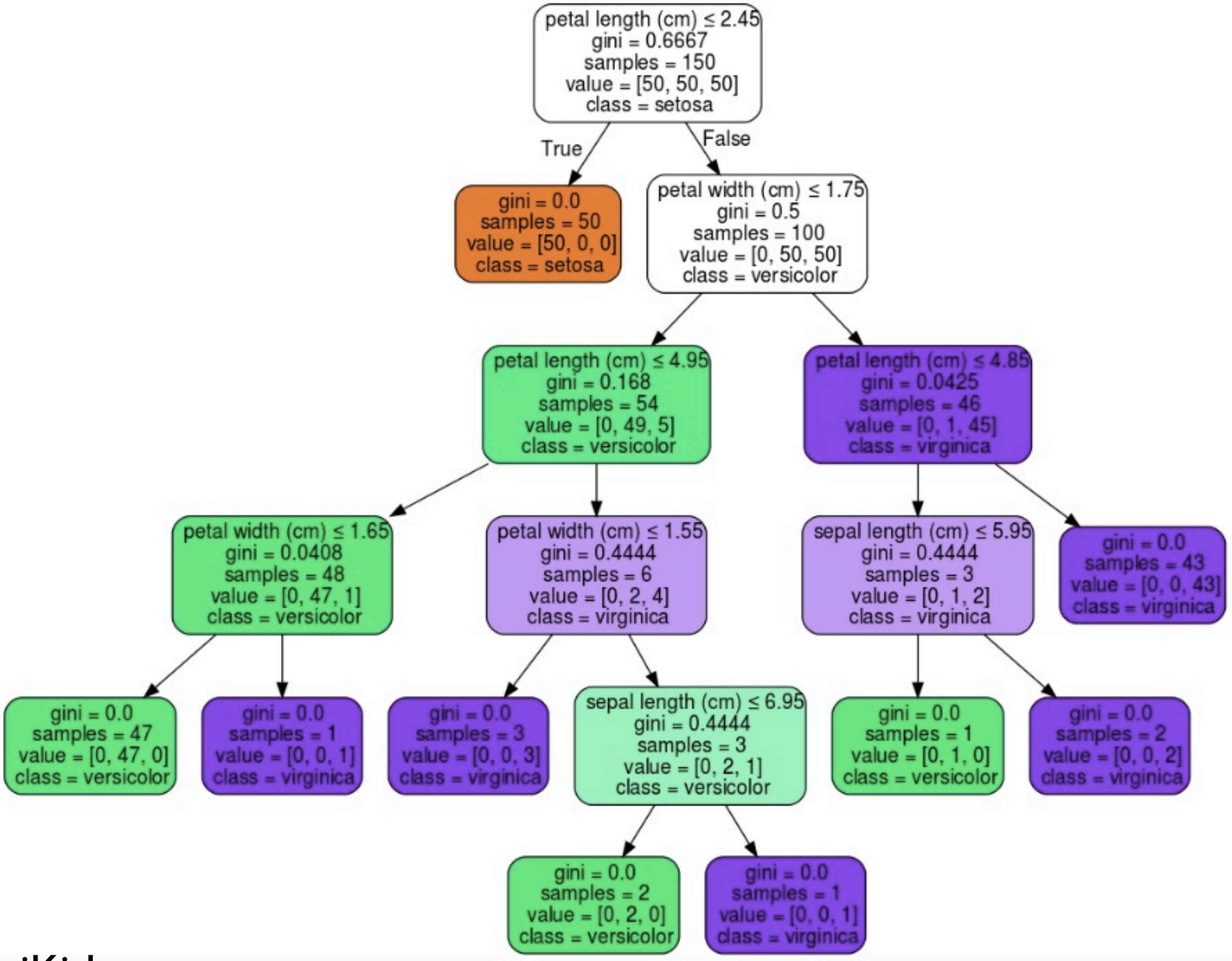
	var 1	var 2	…	var M
…	#	#	…	#
…	#	#	…	#
…	#	#	…	#

var10 > Y **var10 < Y** **var3 > Z** **var3 < Z**



Decision trees

- Advantages
  - Variable scaling/normalisation not required
  - Very straightforward to use (and explain)
  - Can directly visualise how the training has proceeded ("transparent box")
  - Able to handle numerical and categorical variables, multi-classes and run classification/regression without much change to the algorithm
- Disadvantages
  - Susceptible to learn from random fluctuations (over-training)
  - Unstable and fragile: small variations in the training data can lead to completely different trees being formed
  - Unbalanced classes may lead to skewed trees



Boosting

- **Boosting** is based on the concept that an *ensemble of weak learners* can be statistically combined to produce much stronger predictions
  - Weak = only slightly better than 50:50 guess
  - Not restricted to decision trees
- Enables us to preserve many of the advantages of decision trees whilst avoiding some of their pitfalls
- Many different ways of boosting - we'll look at two that are heavily used in HEP
  - Adaptive boosting - AdaBoost
  - Gradient boosting - XGBoost

Boosting

- In all cases we seek an ensemble of trees *T* that minimises the value of some objective function on the training data, with each new tree trying to correct the mistakes of its predecessors

Ensemble

$$T(\mathbf{x}) = \sum_{i=1}^{M} \alpha_i h_i(\mathbf{x})$$

Sum over trees

Trees

Weights

First tree

$$T_0(\mathbf{x}) = 0$$

$$T_m(\mathbf{x}) = T_{m-1}(\mathbf{x}) + \arg\min_{h_m \in H} \sum_{i=1}^{n} J\left(y_i, T_{m-1}(x_i) + h_m(x_i)\right)$$

Sum over events

Updated ensemble

Current tree

Add the tree h_m from the total space of trees *H* that minimises the objective *J*

The means by which the new tree is added distinguishes the different boosting methods

Adaptive boosting (AdaBoost)

Basic idea: when building the ensemble of decision trees, previously misclassified events are **upweighted** for the next tree

$$\mathbf{x} = \{w_1\mathbf{x}_1, \ldots, w_N\mathbf{x}_N\}; \mathbf{y} = \{y_1, \ldots, y_N\} \qquad y \in \{-1, 1\}$$

Weighted data — Target

$$\{w_{1,1} \ldots, w_{N,1}\} = \{\frac{1}{N}, \ldots, \frac{1}{N}\}$$ Initial event weights

AdaBoost

Let us build an ensemble T of *M* decision trees:

For *m* in 1,…, *M*:

$h_m(\mathbf{x})$ *m*th tree built as per recipe on previous slides

$$\epsilon_m = \left(\sum_{i=0}^{n} w_i^{(m)} : h_m(x_i) \neq y_i \right) / \sum_{i=0}^{n} w_i^{(m)}$$

Error: sum of weights for misclassified events / total

Append the *m*th tree to the ensemble:

$$T(\mathbf{x}) = T(\mathbf{x}) + \alpha_m h_m(\mathbf{x}) \text{ where } \alpha_m = \frac{1}{2} \ln \left(\frac{1 - \epsilon_m}{\epsilon_m} \right)$$

Update the weights:

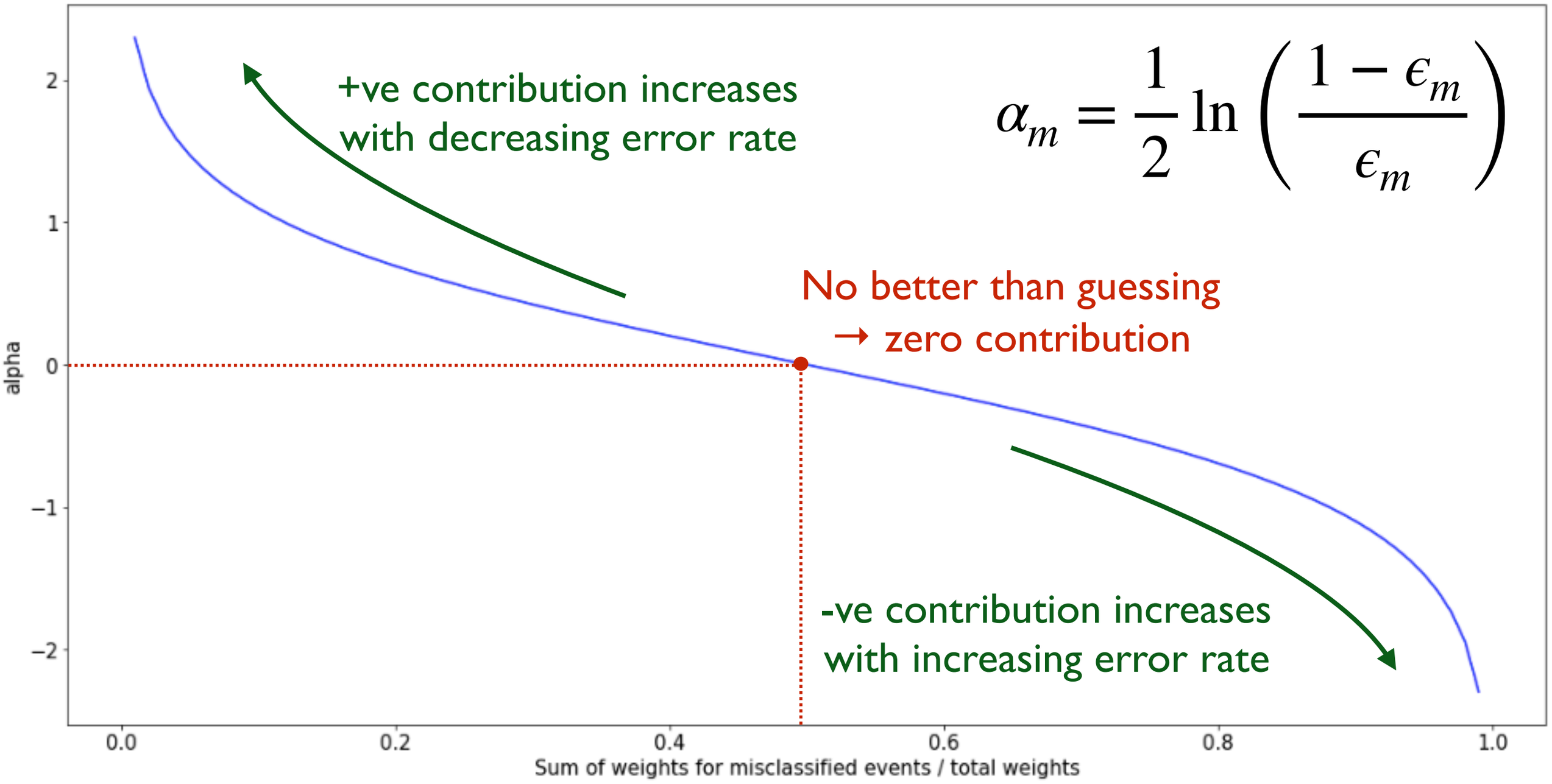
$$w_{i,m+1} = w_{i,m} e^{-y_i \alpha_m h_m(x_i)}$$

Observe that wrongly classified events are **upweighted** and correctly classified events are **downweighted**

T is used to evaluate new data

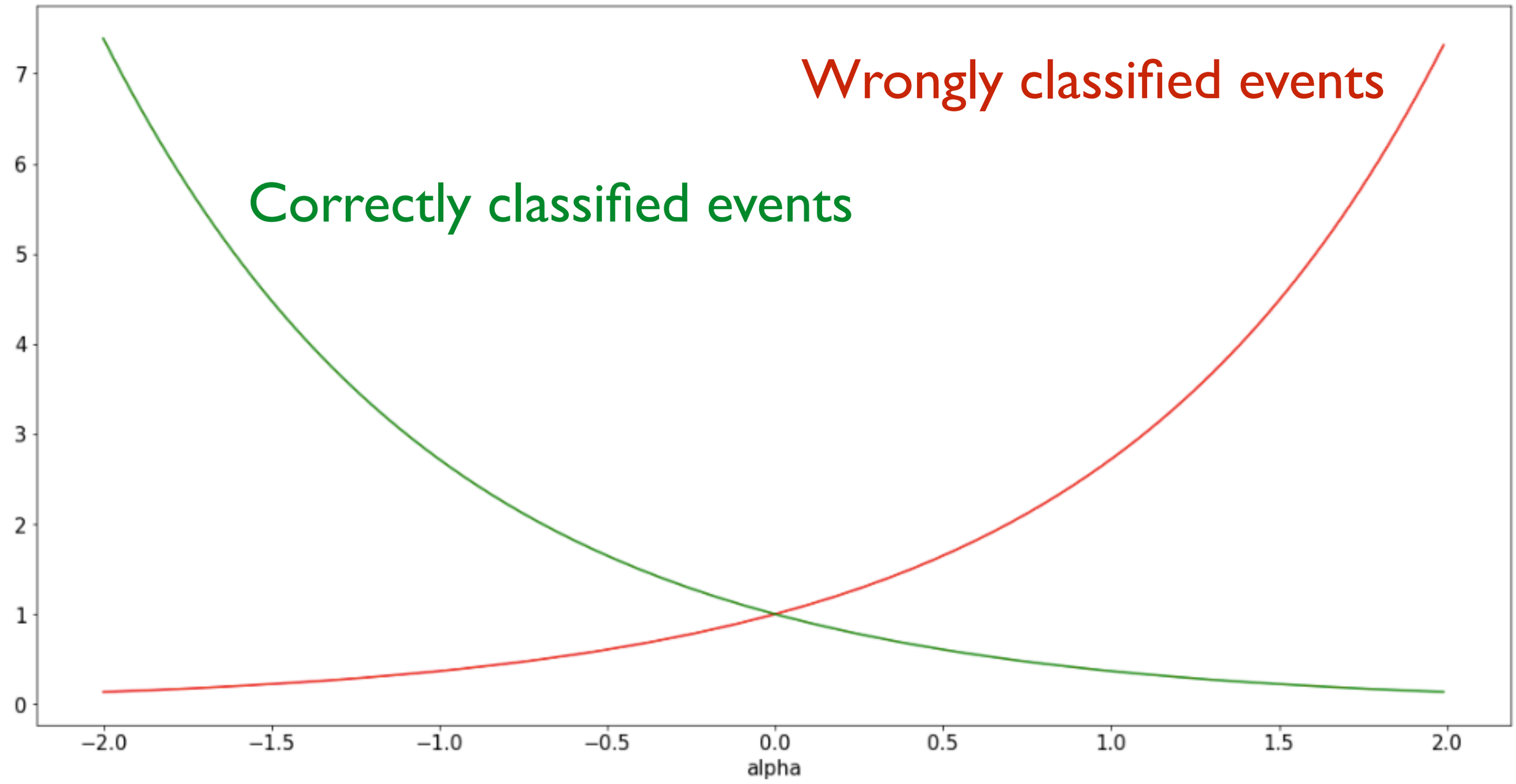
α: contribution per tree

$$\alpha_m = \frac{1}{2} \ln \left(\frac{1 - \epsilon_m}{\epsilon_m} \right)$$



Weight update

$$w_{i,m+1} = w_{i,m} e^{-y_i \alpha_m h_m(x_i)}$$



Tree *m* mostly wrong

Tree *m* useless

Tree *m* mostly right

Gradient boosting: basic idea

- Once a BDT has reached perfection:

$$T_{m+1}(\mathbf{x}) = T_m(\mathbf{x}) + h(\mathbf{x}) = \mathbf{y}$$

such that $h(\mathbf{x}) = \mathbf{y} - T_m(\mathbf{x})$

- These *residuals* are the negative gradients w.r.t. T(x) of the objective function → fit each new *h* to the current residuals
- Equivalent to gradient descent on the loss function

Gradient boosting: algorithm

For m in 1,…, M:

$$r_{i,m} = -\left[\frac{\partial J(y_i, T_{m-1}(\mathbf{x}_i))}{\partial T_{m-1}(\mathbf{x}_i)}\right] \text{ for } i = 0,\ldots,n$$

Gradients

Train h_m using $r_{i,m}$ as the event weights

Find:

$$\gamma_m = \arg\min_{\gamma} \sum_{i=1}^{n} J\left(y_i, T_{m-1}(\mathbf{x}_i) + \gamma h_m(\mathbf{x}_i)\right)$$

Update T:

$$T_m(\mathbf{x}) = T_{m-1}(\mathbf{x}) + \gamma_m h_m(\mathbf{x})$$

Trained BDT: T_M

Typical BDT hyperparameters

- Total number of decision trees in the ensemble
  - Usually in the low 100s
- Maximum depth of the individual decision trees
  - Usually around 3-6, can even be 1 (e.g. one partition per cycle)
- Learning rate (if less than 1, shrinks the contribution of each new classifier)
  - Means of *regularisation* - avoiding over-fitting
- Minimum number of events in a leaf (e.g. threshold at which a split is made)
  - Often no minimum, e.g. it is 1
- Criterion by which nodes are split
- Objective function
- Evaluation method

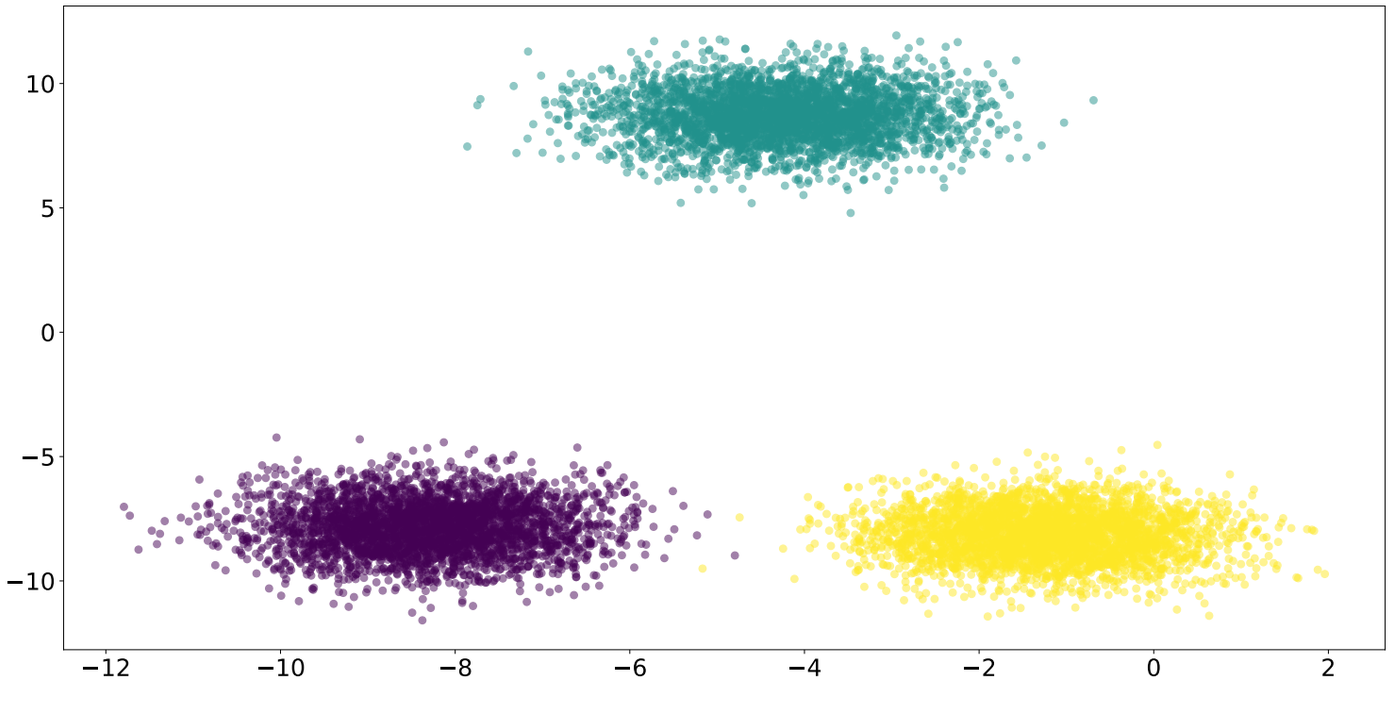
Interesting side-note: not so common to hear discussion of these hyperparameters in physics analyses - does everyone just use the defaults?

XGBoost

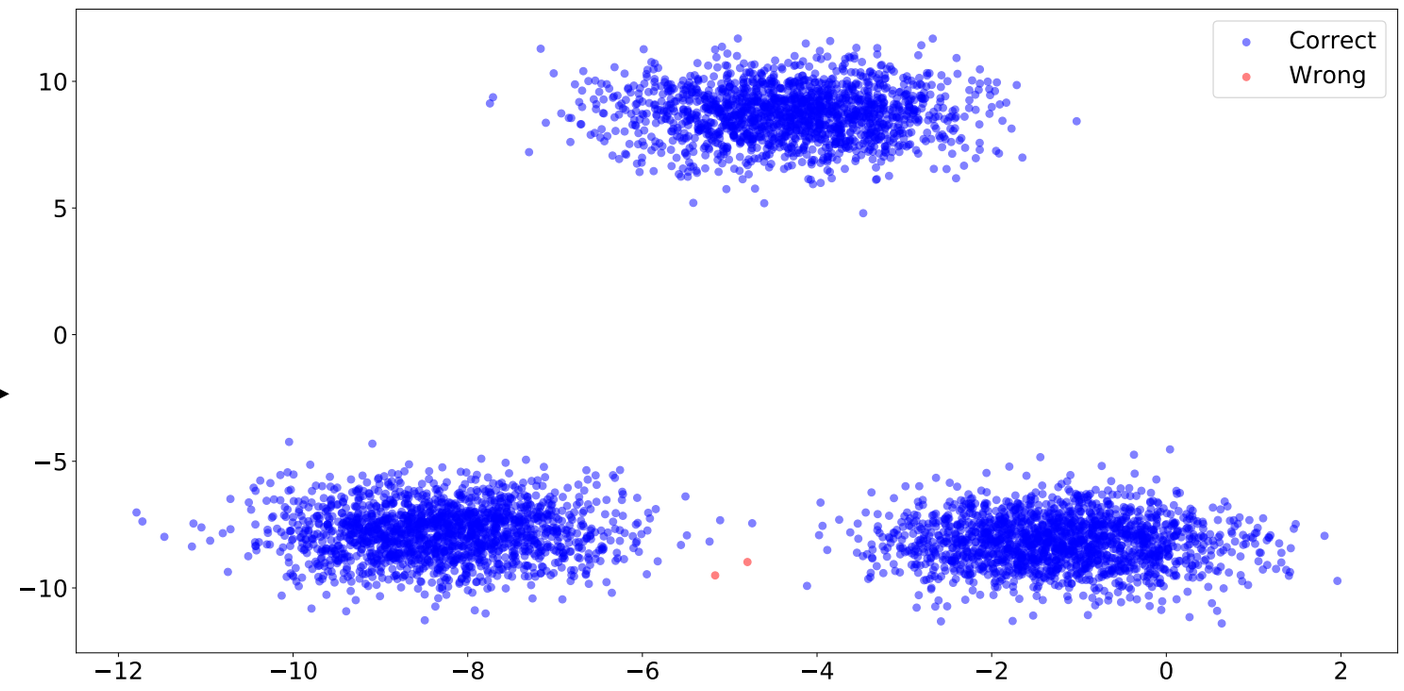
- E**x**treme **G**radient **Boost**ing - documentation
- Implementation of gradient boosted decision trees
- Well-known for winning the HiggsML challenge
- Versions exist for Python (incl. Anaconda), R, C++ and TMVA (via the R interface)

- Key features
  - Heavily optimised, leading to fast performance
  - Built-in support for multi-threaded training and distributed training
  - Automated handling of missing variables (very useful, e.g. jet p_T when there are no jets…)
  - Feature importance analysis and tree visualisation

Linear discriminant: linearly separable clusters

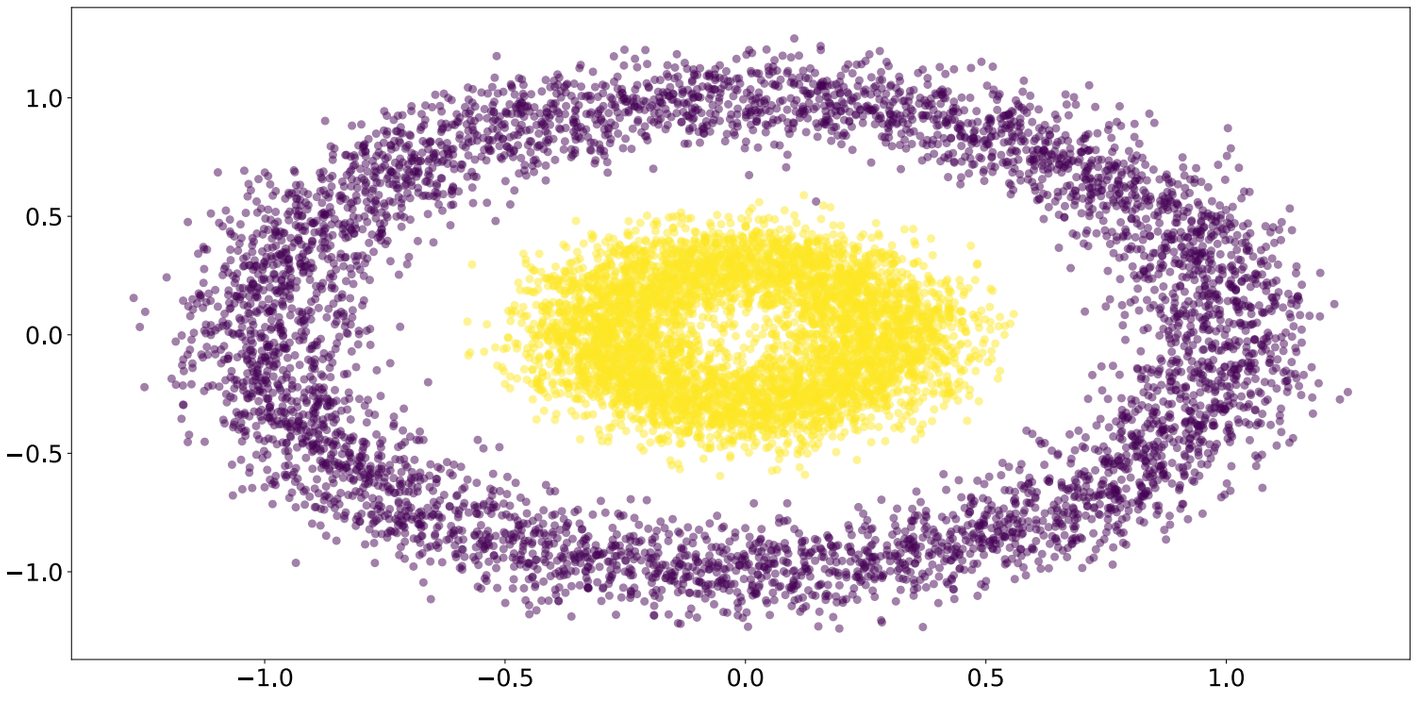


Linear discriminant (Fisher)

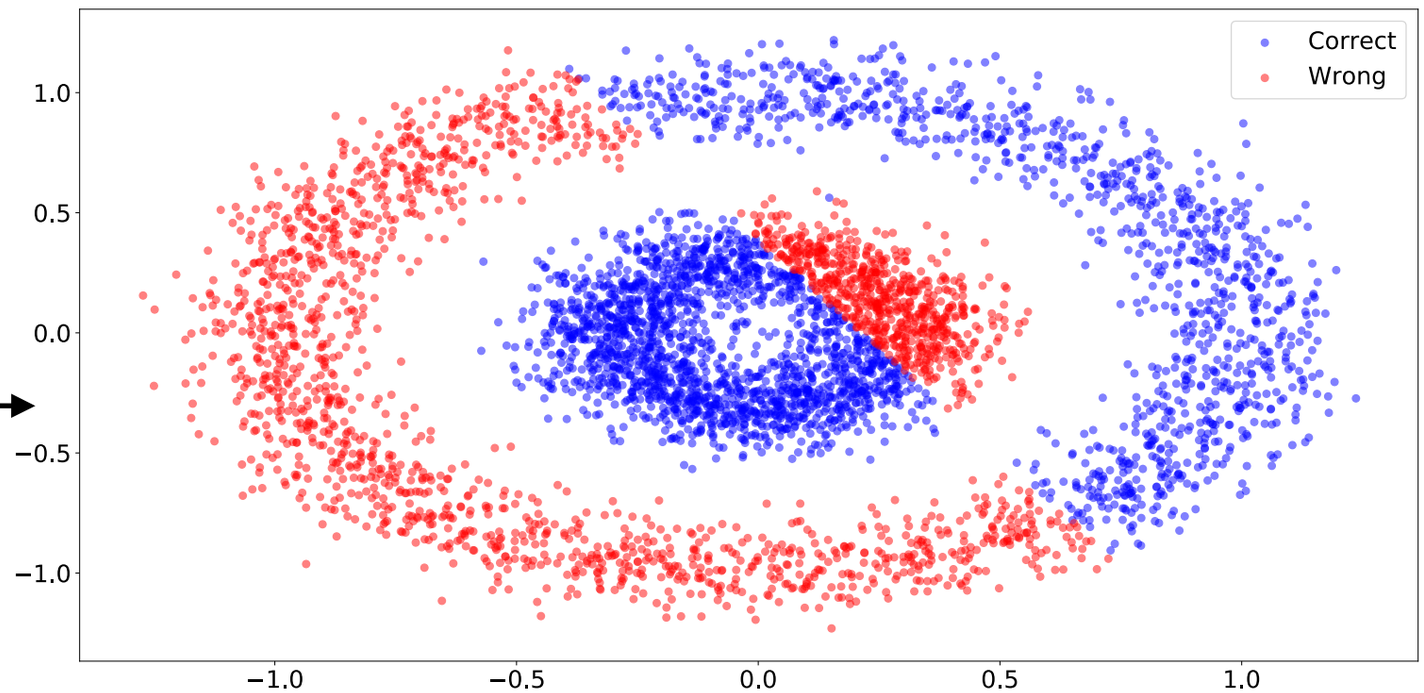


No need for anything fancy

Linear discriminant: concentric circles



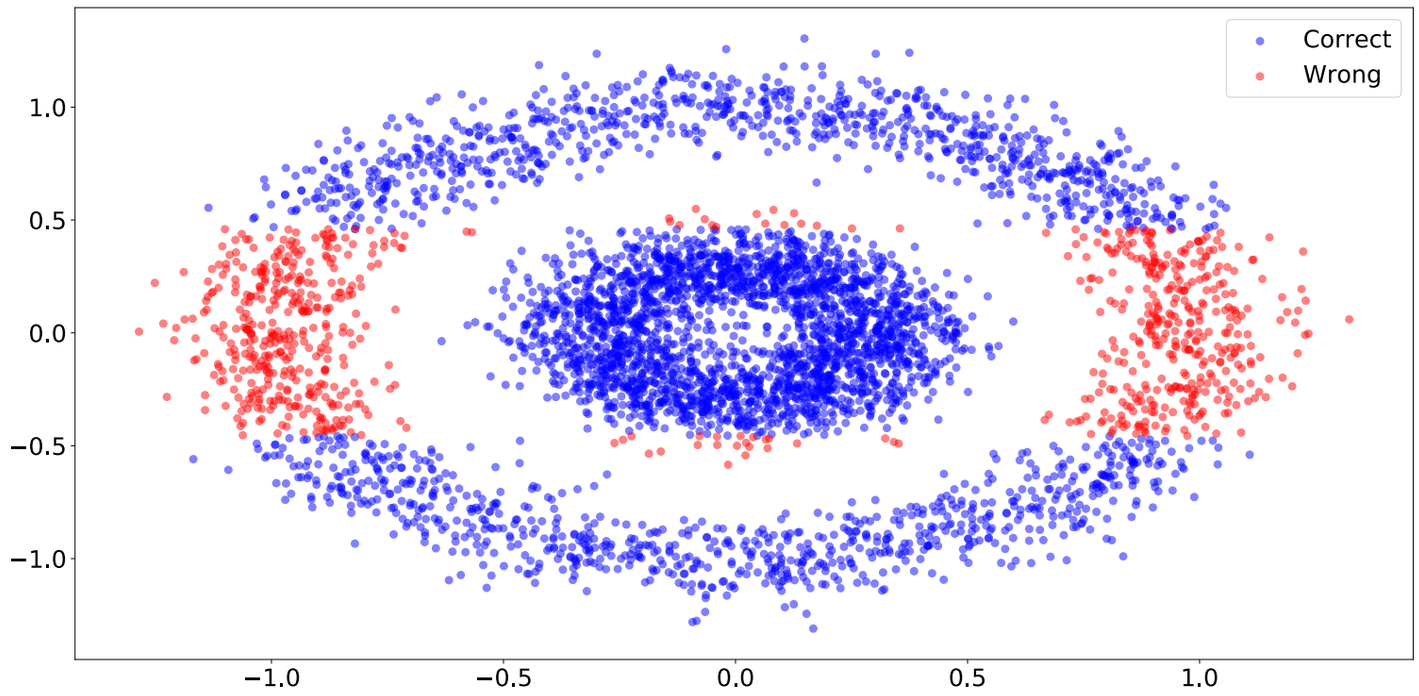
Linear discriminant (Fisher)



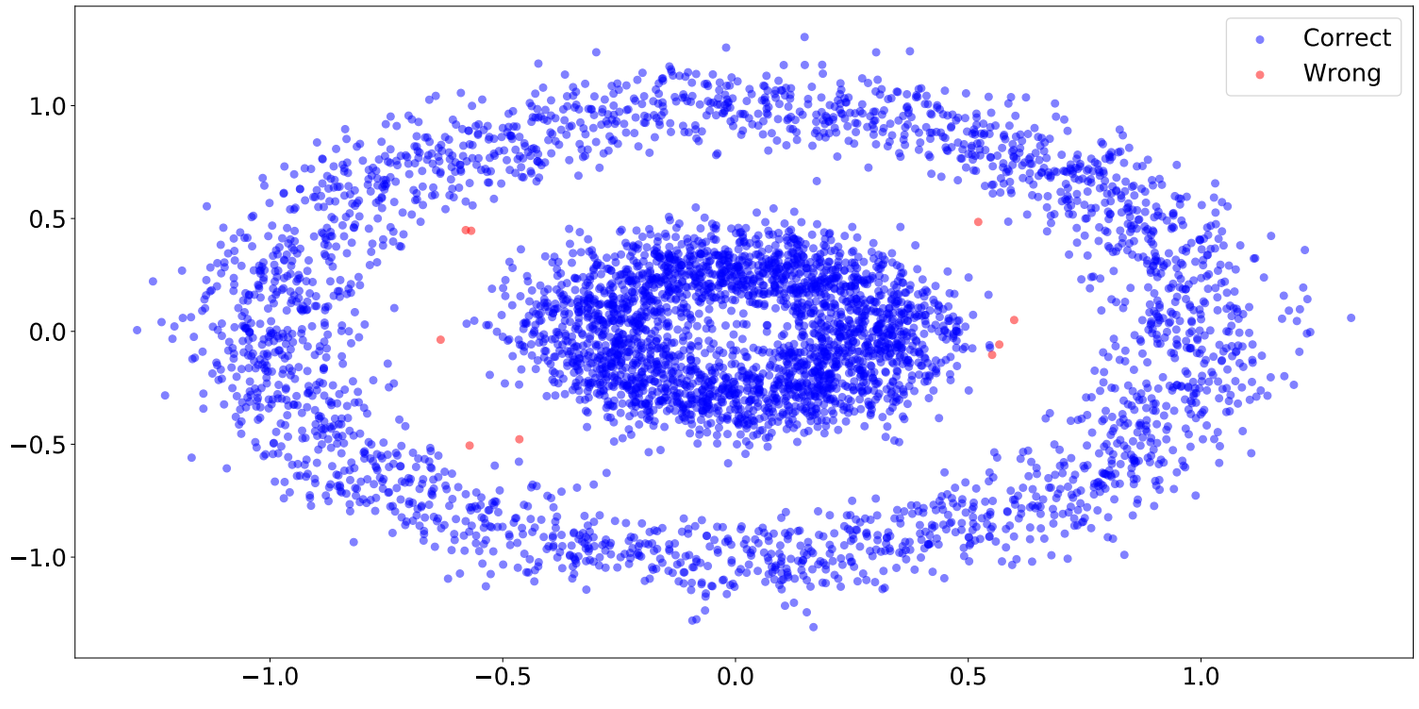
L.D. doesn't work at all as expected

BDT on concentric circles

1 tree, depth=2



50 trees, depth=2



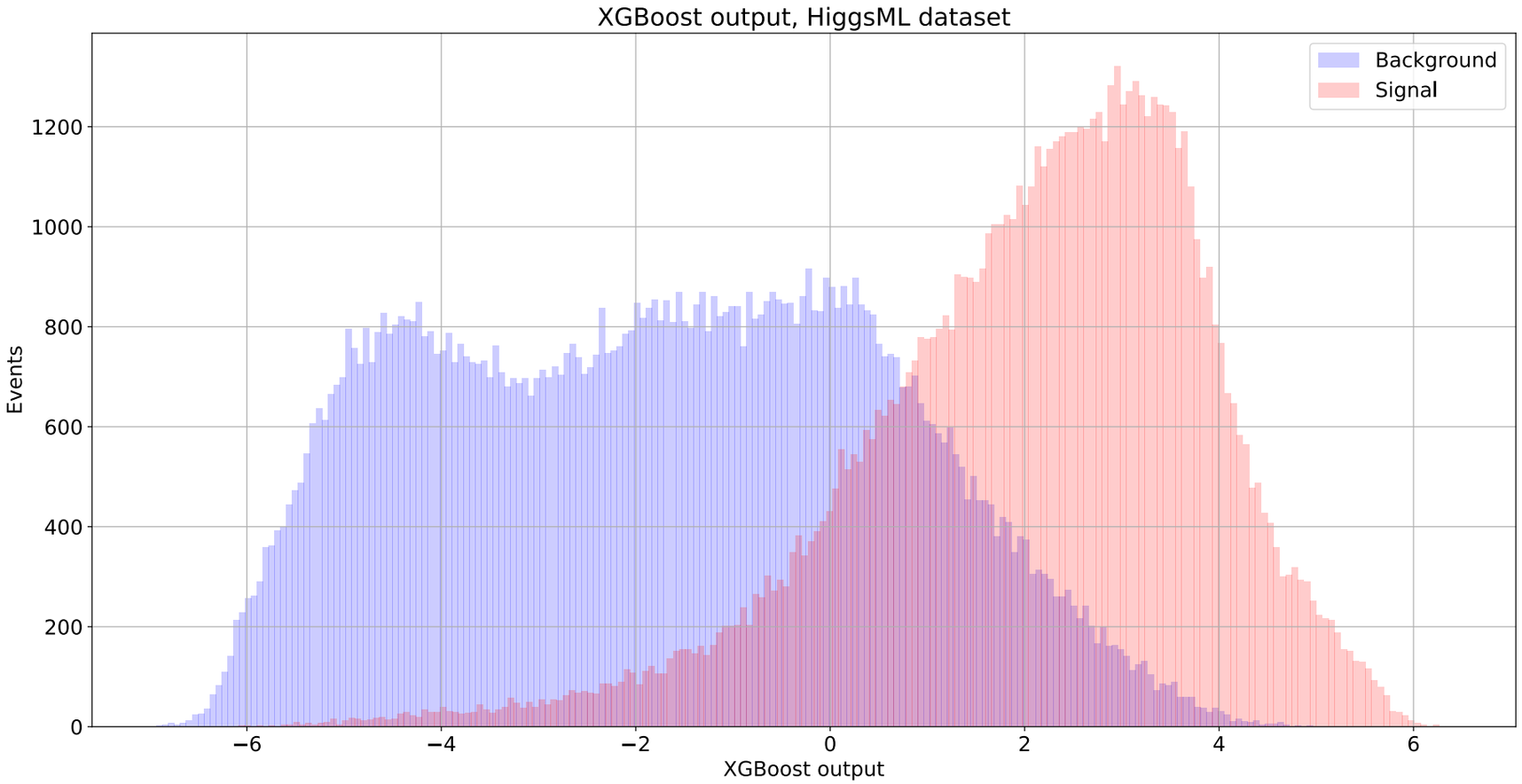
(Note: single tree with depth 6 can achieve the same without boosting)

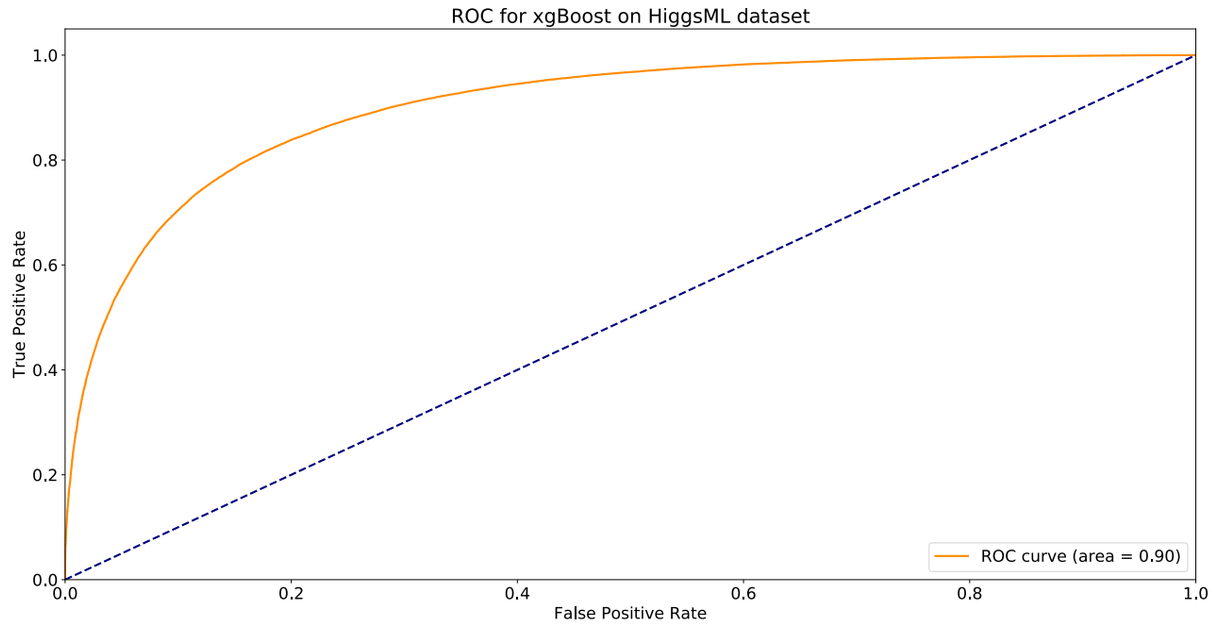
XGBoost demo: HiggsML

- http://opendata.cern.ch/record/328
- Signal: H→ττ; background: Z→ττ, ttbar, W→leptons
- 818238 events available in total
- Out of the box test (no hyperparameter tuning)
- Download dataset (CSV) from the OpenData portal
- Use all available variables
- In the Jupyter notebook advertised earlier
  - Use Pandas to digest the CSV
  - Split into NumPy arrays: 100K training events and 100K evaluation events (each with half signal, half data)
  - Train XGBoost, with the following hyperparameters:
    - 120 trees, max depth = 6, learning rate = 0.3

DER_mass_MMC: The estimated mass m_H of the Higgs boson candidate, obtained through a probabilistic phase space integration.
DER_mass_transverse_met_lep: The transverse mass between the missing transverse energy and the lepton.
DER_mass_vis: The invariant mass of the hadronic tau and the lepton.
DER_pt_h: The modulus of the vector sum of the transverse momentum of the hadronic tau, the lepton and the missing transverse energy vector.
DER_deltaeta_jet_jet: The absolute value of the pseudorapidity separation between the two jets (undefined if PRI_jet_num ≤ 1).
DER_mass_jet_jet: The invariant mass of the two jets (undefined if PRI_jet_num ≤ 1).
DER_prodeta_jet_jet: The product of the pseudorapidities of the two jets (undefined if PRI_jet_num ≤ 1).
DER_deltar_tau_lep: The R separation between the hadronic tau and the lepton.
DER_pt_tot: The modulus of the vector sum of the missing transverse momenta and the transverse momenta of the hadronic tau, the lepton, the leading jet (if PRI_jet_num $\geq$) and the subleading jet (if PRI jet num = 2) (but not of any additional jets).
DER_sum_pt: The sum of the moduli of the transverse momenta of the hadronic tau, the lepton, the leading jet (if PRI jet num ≥ 1) and the subleading jet (if PRI jet num = 2) and the other jets (if PRI jet num = 3).
DER_pt_ratio_lep_tau: The ratio of the transverse momenta of the lepton and the hadronic tau.
DER_met_phi_centrality: The centrality of the azimuthal angle of the missing transverse energy vector w.r.t. the hadronic tau and the lepton.
DER_lep_eta_centrality: The centrality of the pseudorapidity of the lepton w.r.t. the two jets (undefined if PRI_jet_num ≤ 1).
PRI_tau_pt: The transverse momentum $\sqrt{p_x^2 + p_y^2}$ of the hadronic tau.
PRI_tau_eta: The pseudorapidity η of the hadronic tau.
PRI_tau_phi: The azimuth angle ϕ of the hadronic tau.
PRI_lep_pt: The transverse momentum $\sqrt{p_x^2 + p_y^2}$ of the lepton (electron or muon).
PRI_lep_eta: The pseudorapidity η of the lepton.
PRI_lep_phi: The azimuth angle ϕ of the lepton.
PRI_met: The missing transverse energy $\overrightarrow{E}_T^{miss}$
PRI_met_phi: The azimuth angle ϕ of the mssing transverse energy
PRI_met_sumet: The total transverse energy in the detector.
PRI_jet_num: The number of jets (integer with value of 0, 1, 2 or 3; possible larger values have been capped at 3).
PRI_jet_leading_pt: The transverse momentum $\sqrt{p_x^2 + p_y^2}$ of the leading jet, that is the jet with largest transverse momentum (undefined if PRI_jet_num = 0).
PRI_jet_leading_eta: The pseudorapidity η of the leading jet (undefined if PRI jet num = 0).
PRI_jet_leading_phi: The azimuth angle ϕ of the leading jet (undefined if PRI jet num = 0).
PRI_jet_subleading_pt: The transverse momentum $\sqrt{p_x^2 + p_y^2}$ of the leading jet, that is, the jet with second largest transverse momentum (undefined if PRI_jet_num ≤ 1).
PRI_jet_subleading_eta: The pseudorapidity η of the subleading jet (undefined if PRI_jet_num ≤ 1).
PRI_jet_subleading_phi: The azimuth angle ϕ of the subleading jet (undefined if PRI_jet_num ≤ 1).
PRI_jet_all_pt: The scalar sum of the transverse momentum of all the jets of the events.
Weight: The event weight w_i
Label: The event label (string) $y_i \in \{s, b\}$ (s for signal, b for background).

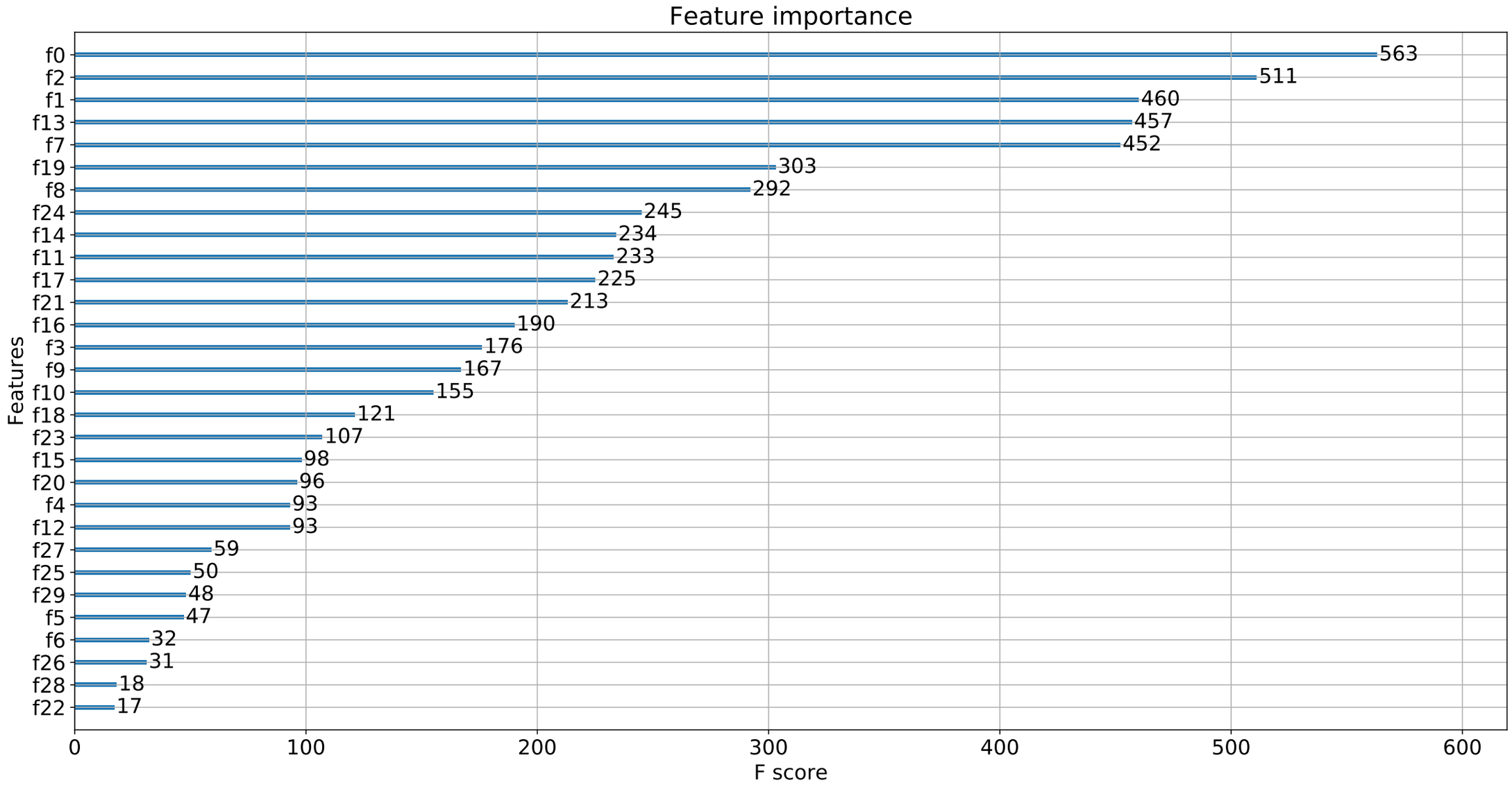
XGBoost demo: HiggsML





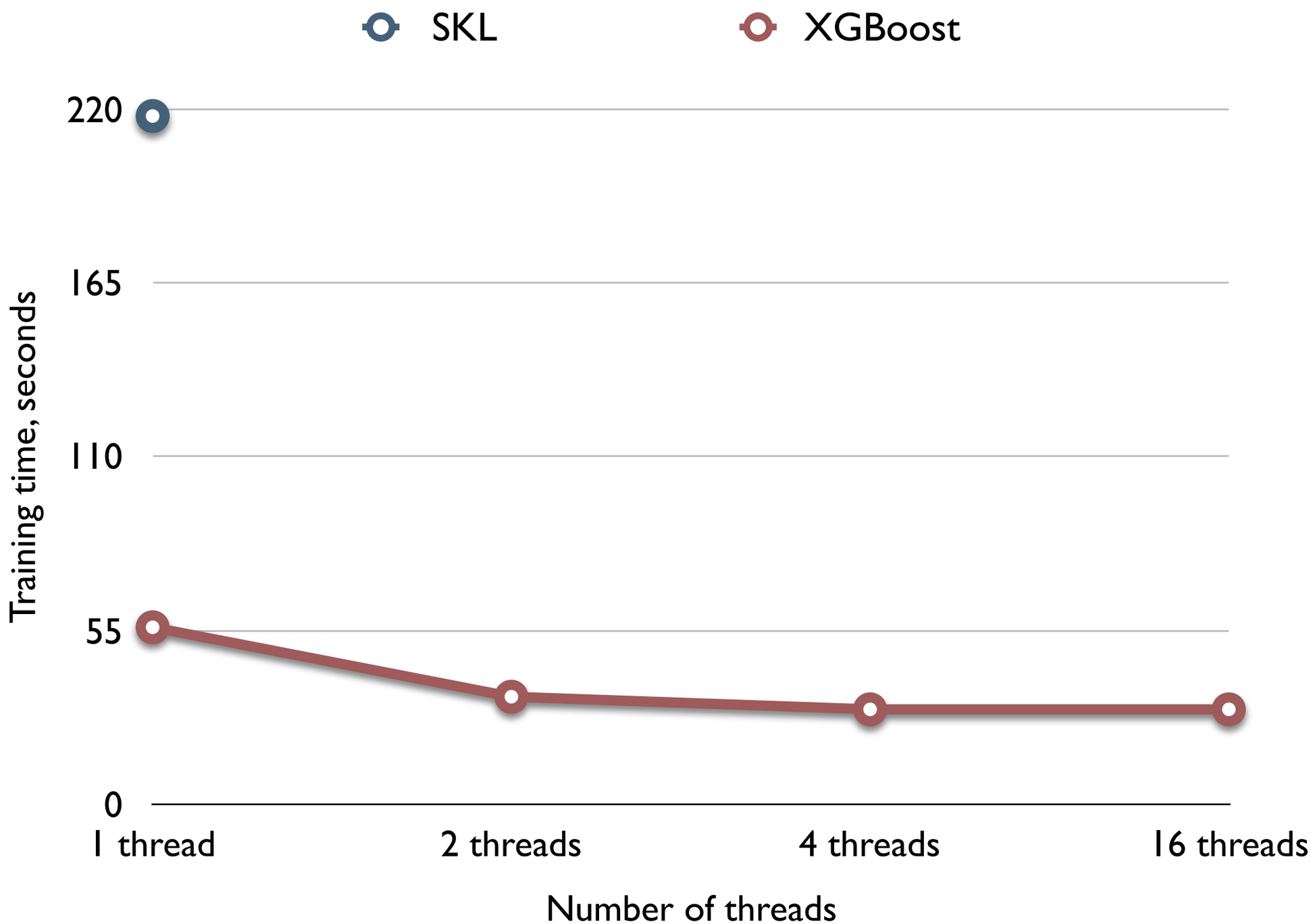
Variable importance

- The more often a variable is used to split the data, the more important (discriminating) it is
- Enables creation of plots like this:



Note that BDTs aren't perturbed by under-utilised variables - they just don't use them to cut the data

Performance - training time (on my 4 core laptop)



Remarks on BDTs

- Most analyses use the TMVA BDT, but should also try XGBoost
  - SKL BDTs seem to be less optimised than either
- Do we need to spend more time optimising the hyperparameters?
- In particular
  - those related to the internals of the trees rather than the boosting, e.g. leaf splitting criteria and means of evaluation
  - objective function
- Not clear that there is any benefit to using *shallow* neural networks over BDTs
- BDTs are highly performant (especially XGBoost), intuitive, robust (especially against missing variables), require modest computing power, and are well suited to physics analysis where the data is labelled

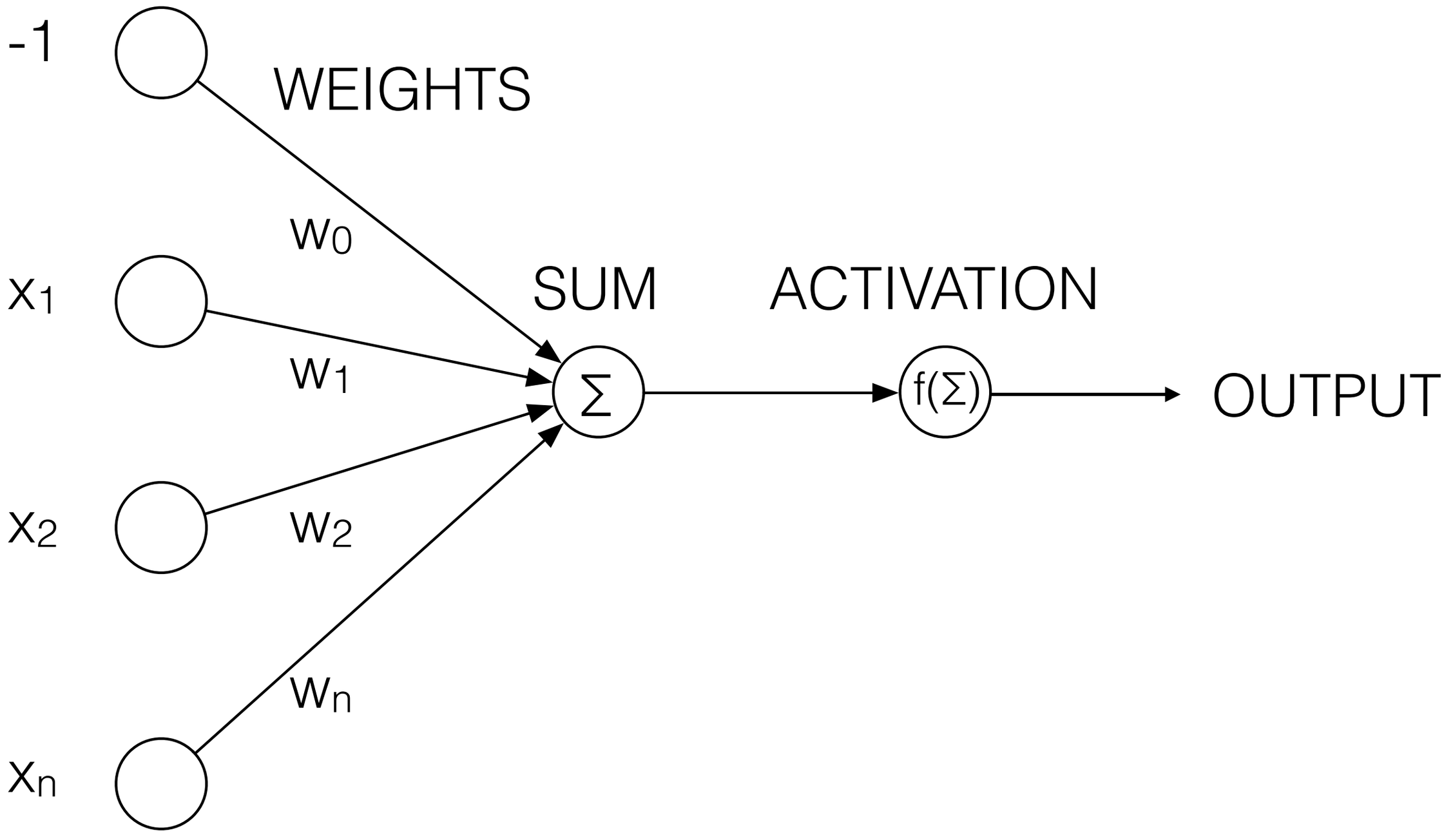
Neural networks

Artificial neural networks

- First conceived of in the 1940s-1950s, inspired by research into biological neural processes
- Key developments:
  - Perceptron (1958)
  - Back-propagation (1975)
- Initial development was slow due to limited computing power and insufficient training data; many machine learning researchers lost interest and focused on other techniques such as support vector machines and linear methods
- In the past 15 years several factors combined to completely change the situation
  - Growth of the internet and smart phones = massively more training data and network capacity
  - Huge increase in computing power and memory
  - New ideas from the academic community, spread via open source software
- This led to the astonishing capabilities of **deep learning** that we see today

The perceptron model

LINEAR MODEL



The perceptron model: how it learns

$\mathbf{x} = \{\mathbf{x}_1, \ldots, \mathbf{x}_M\}; \mathbf{y} = \{y_1, \ldots, y_M\}$ Training data

M events with N variables

At each step t: evaluate for each event *j:*

$$y_j(t) = f\left[w_0(t) + w_1(t)x_{j,1} + w_2(t)x_{j,2} + \ldots + w_N(t)x_{j,N}\right]$$

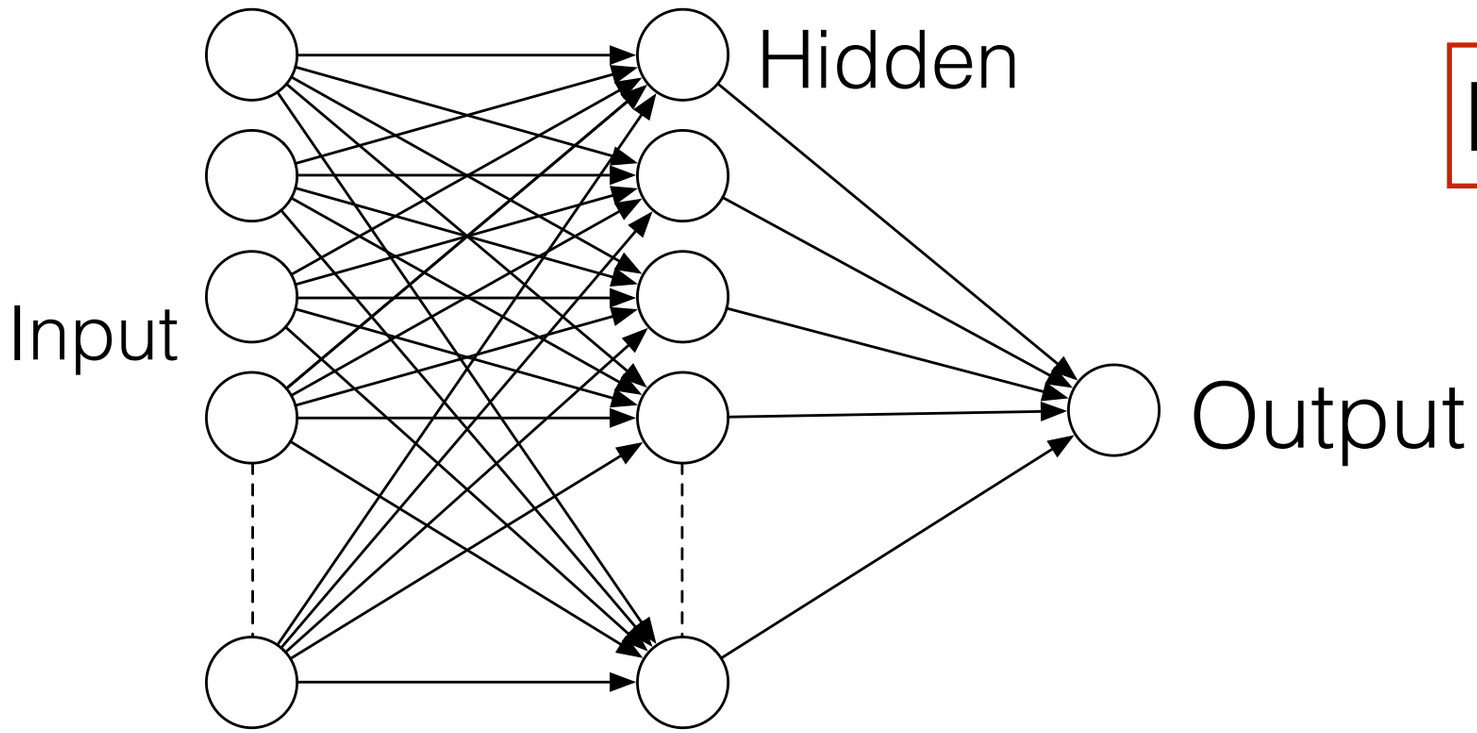
For each variable *i* from 1 to N, update the weights:

$$w_i(t+1) = w_i(t) + r \cdot \left(y_j - y_j(t)\right) x_{j,i}$$

Learning rate

Continue until $\frac{1}{M}\sum_{j=0}^{M} |\, y_j - y_j(t) \,|$ reaches some appropriate level, or the number of steps t exceeds some value

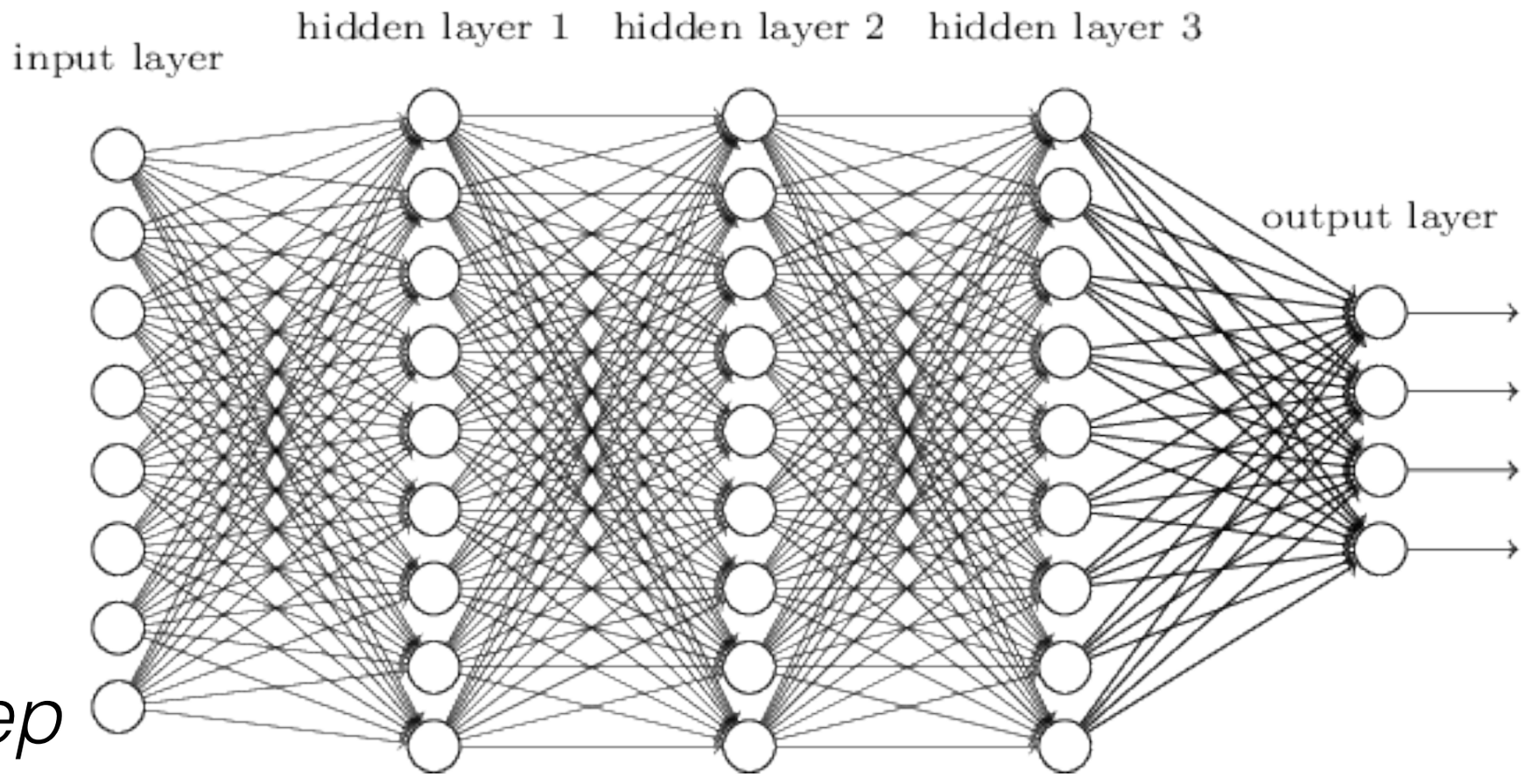
Multi-layer perceptrons



NON-LINEAR MODEL

Shallow

Deep



Back propagation

- The perceptron learning mechanism doesn't extend to multiple layers
- Instead *back propagation* is used to update the weights (which may number millions in a very deep network)
- Basic idea:
  - Pass each event through the network (feed forward) arriving at a result
  - Compare with the target using some loss function, arriving at some error
  - Propagate this error backwards through the network, node by node and layer by layer, until each neuron has its own contribution to the overall error
  - Use these errors to calculate the partial derivatives of the loss function w.r.t. the weight at each neuron (*gradients*), by applying the chain rule for derivatives
  - Use gradient descent to minimise the loss, which it does by updating the weights
  - New examples are then passed through the trained network (fixed weights)

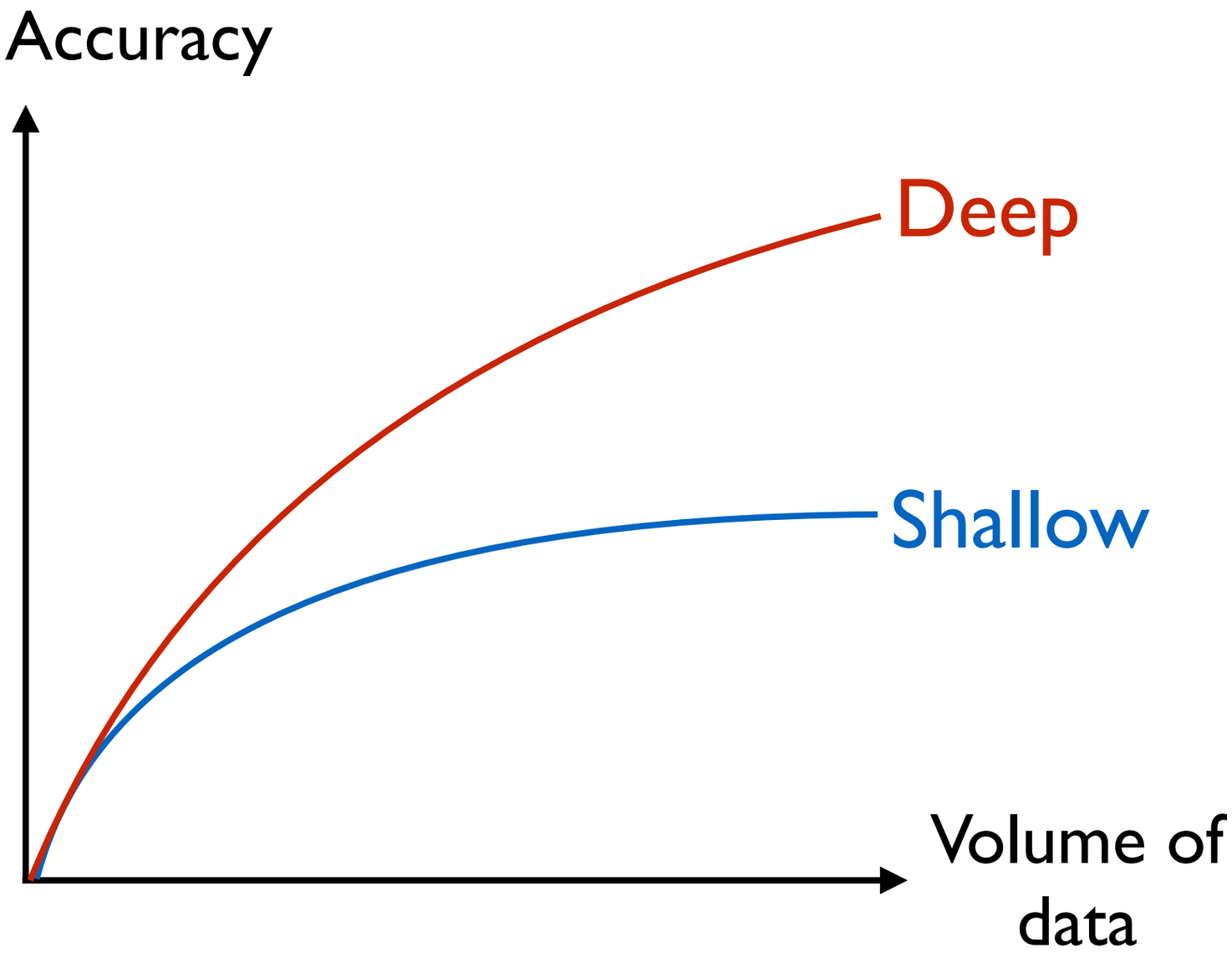
Choreography of back-propagation

- An *epoch* is a single pass through all of the training data
  - Training over 100 epochs means the networks sees the training data, in full, 100 times
- In *stochastic training* each new event leads to a weight update → noisier so less likely to fall into local minima during minimization
- In *batch training* weights are only updated after a large number of events have been fed forwards, with the errors accumulating → faster
- *Mini-batch training* is a compromise between the two, with small batches being selected at random from the full sample

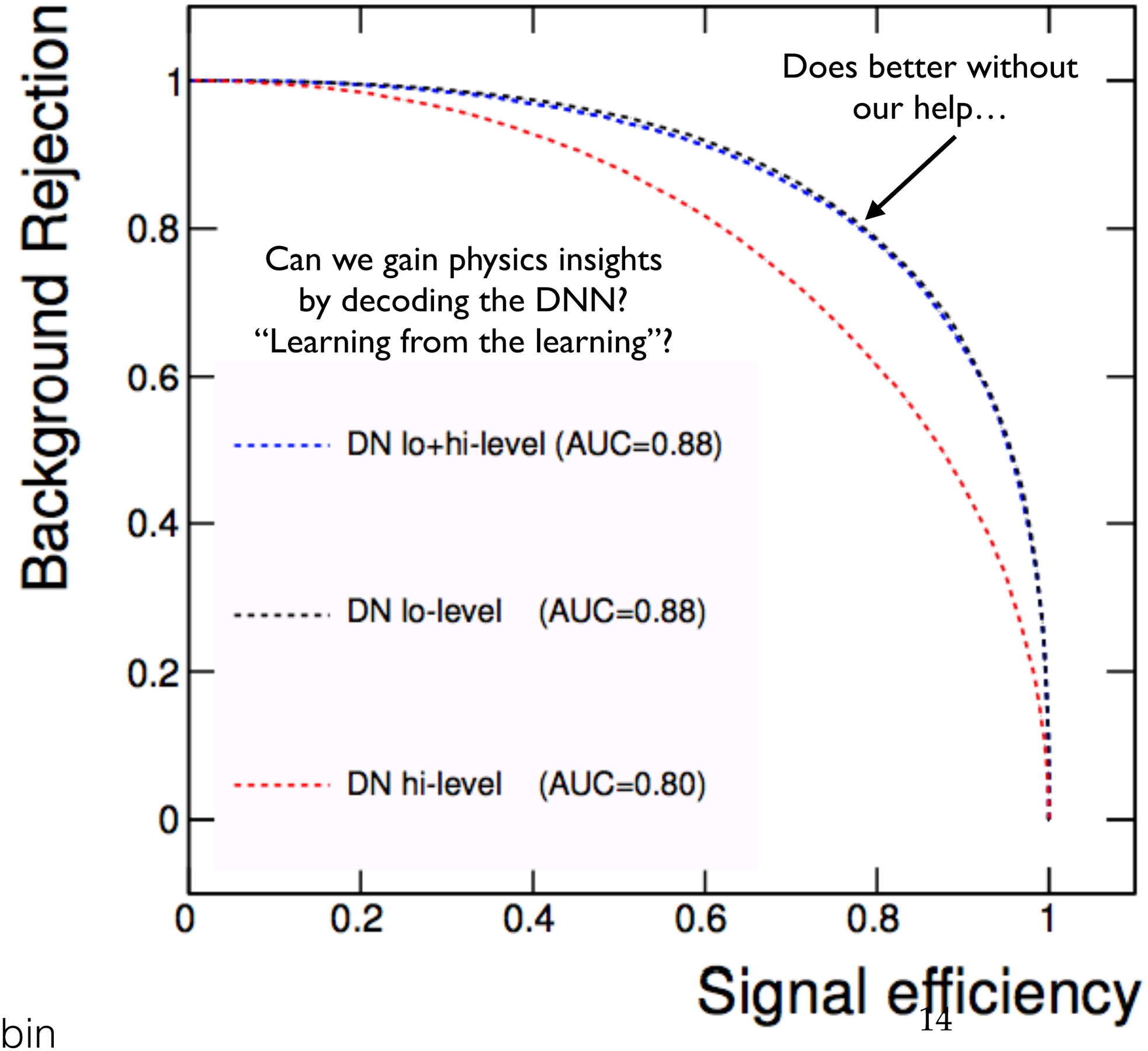
Deep learning

- Deep neural networks have a much larger parameter space and, given enough training data, can model more complex behaviour than a shallow network or a BDT

- Deep neural networks are inherently suited to vectorisation and co-processors → allow use of GPUs or dedicated hardware

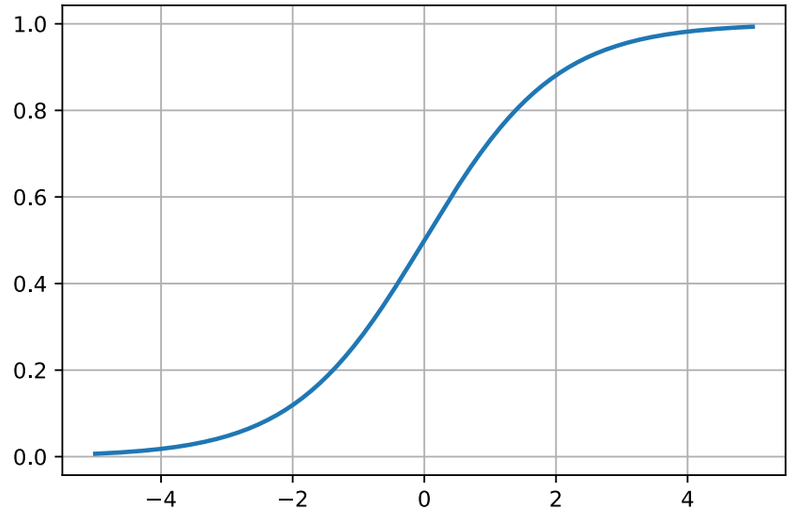


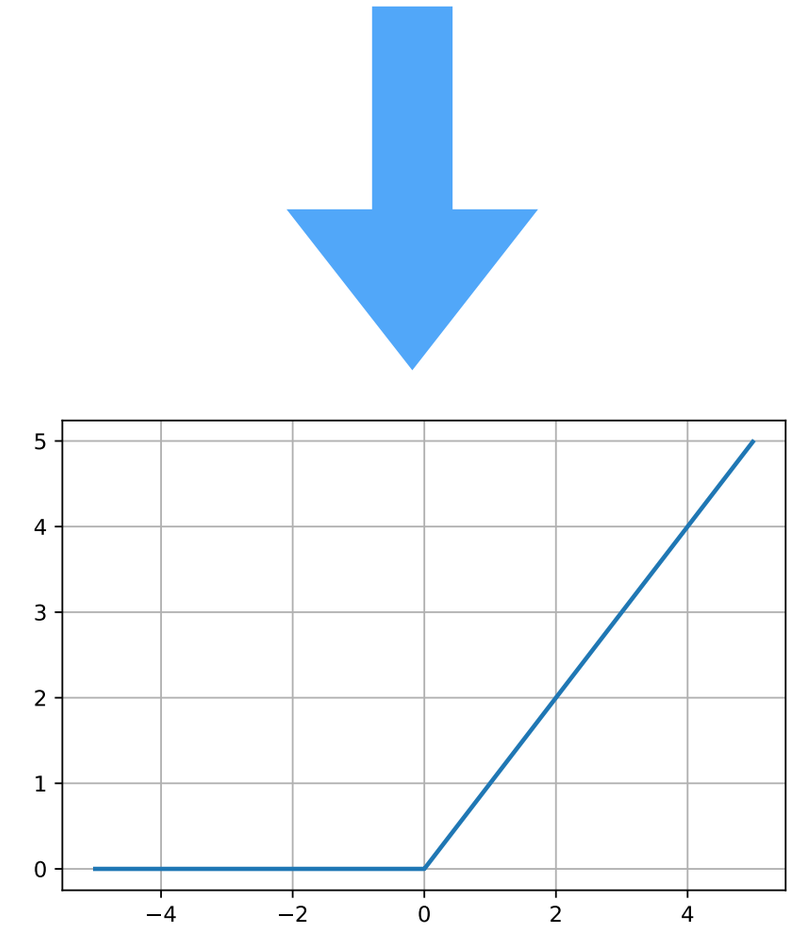
Deep learning allows us to extract the maximum possible value from large datasets


Background Rejection
Signal efficiency
0
0.2
0.4
0.6
0.8
1
Does better without
our help…
Can we gain physics insights
by decoding the DNN?
"Learning from the learning"?
DN lo+hi-level (AUC=0.88)
DN lo-level (AUC=0.88)
DN hi-level (AUC=0.80)

Two particular deep learning challenges

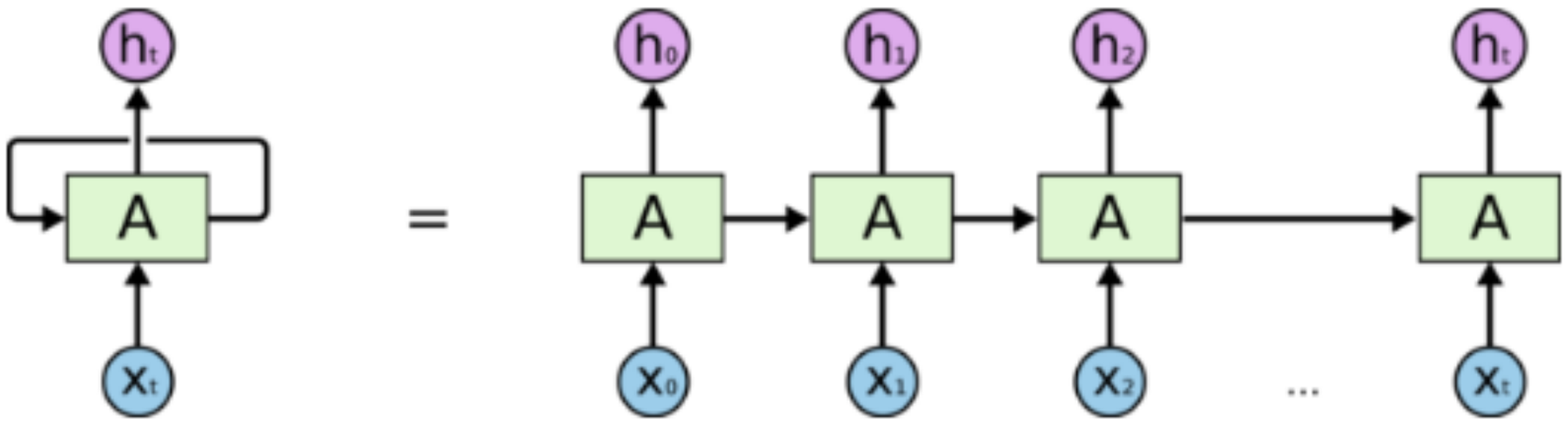
- Vanishing gradient problem
  - With so many nodes the gradients can tend to zero leading to a breakdown in the training
  - Partly solved by use of rectifier activation functions, e.g. ReLU, rather than sigmoid or tanh (etc)
  - Use of sign of the gradients only, etc
- Overtraining
  - Very easy for deep NNs to train on random fluctuations due to their high capacity
  - Solving this has required many innovations in regularisation for neural networks
    - Dropout, weight decay, early stopping, data augmentation, momentum…





Recurrent neural networks

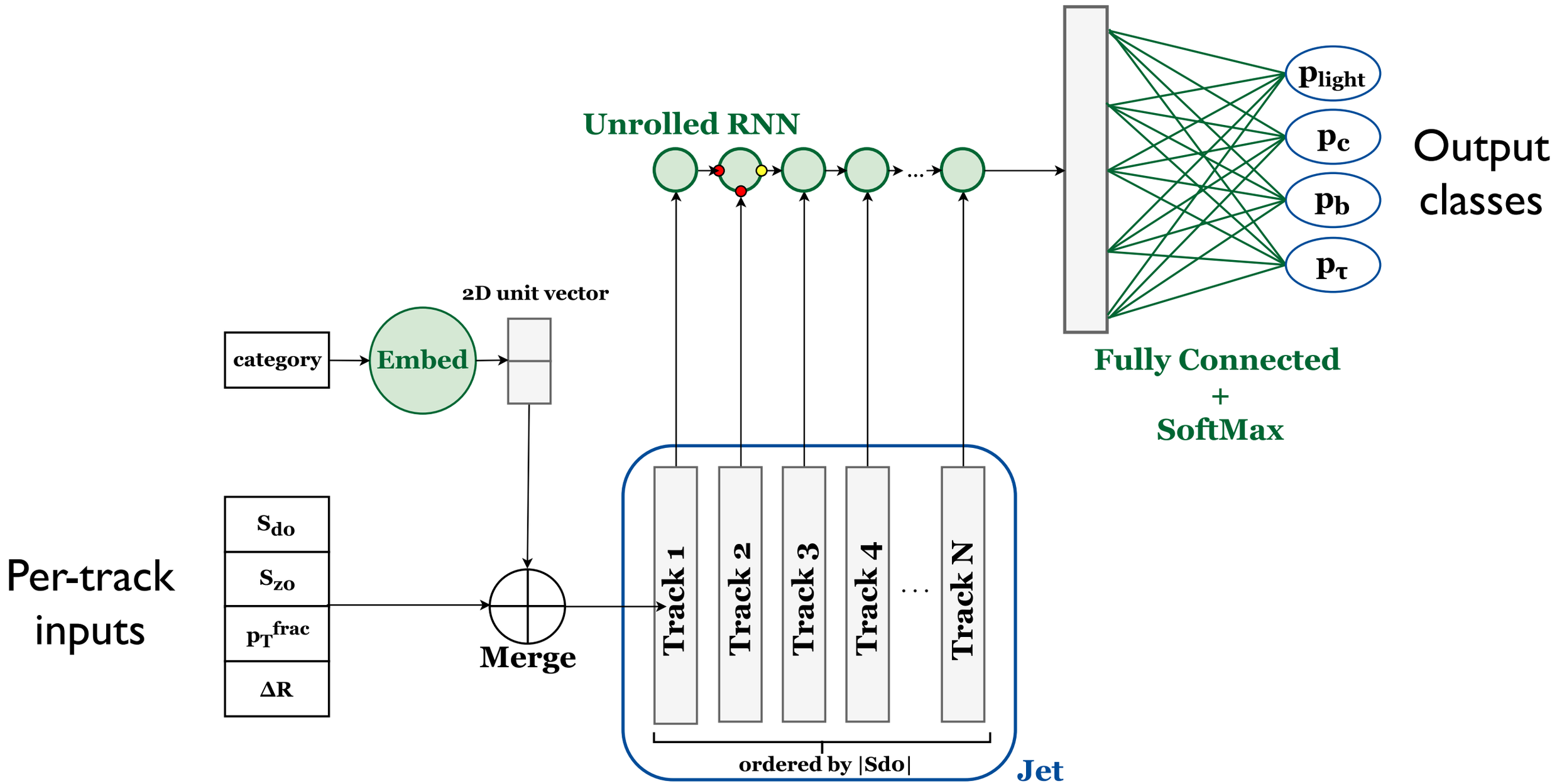
- Different architecture with loops in the structure
  - The loops allow information to persist
  - Each additional input adds a new variant of the network to a chain (directed graph)
  - This allows processing of time varying sequences of information: breaks the mould that there must be a fixed size vector for all inputs and outputs - obvious use cases include
- Natural language processing, machine translation, speech and music recognition
- In our field - **flavour tagging** with the time varying sequence being the tracks in a jet



An unrolled recurrent neural network.

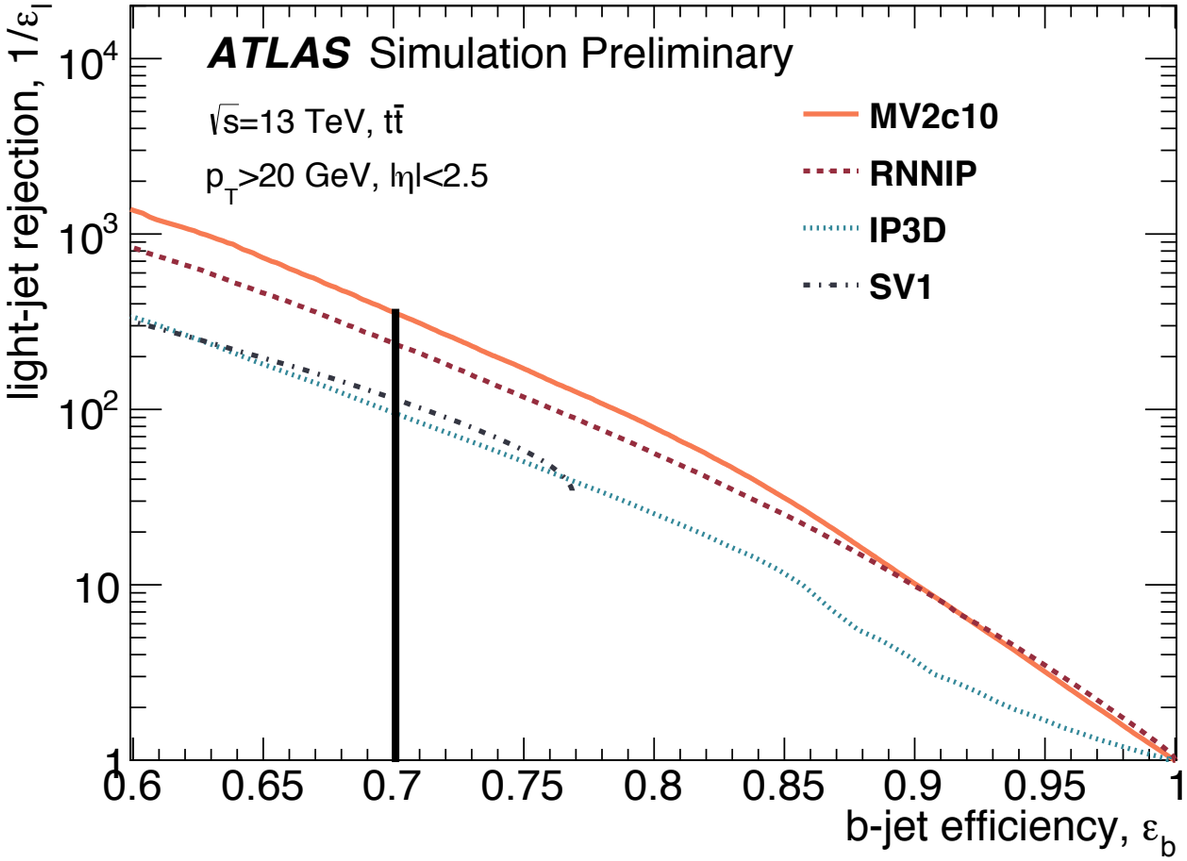
Flavour tagging with recurrent neural networks

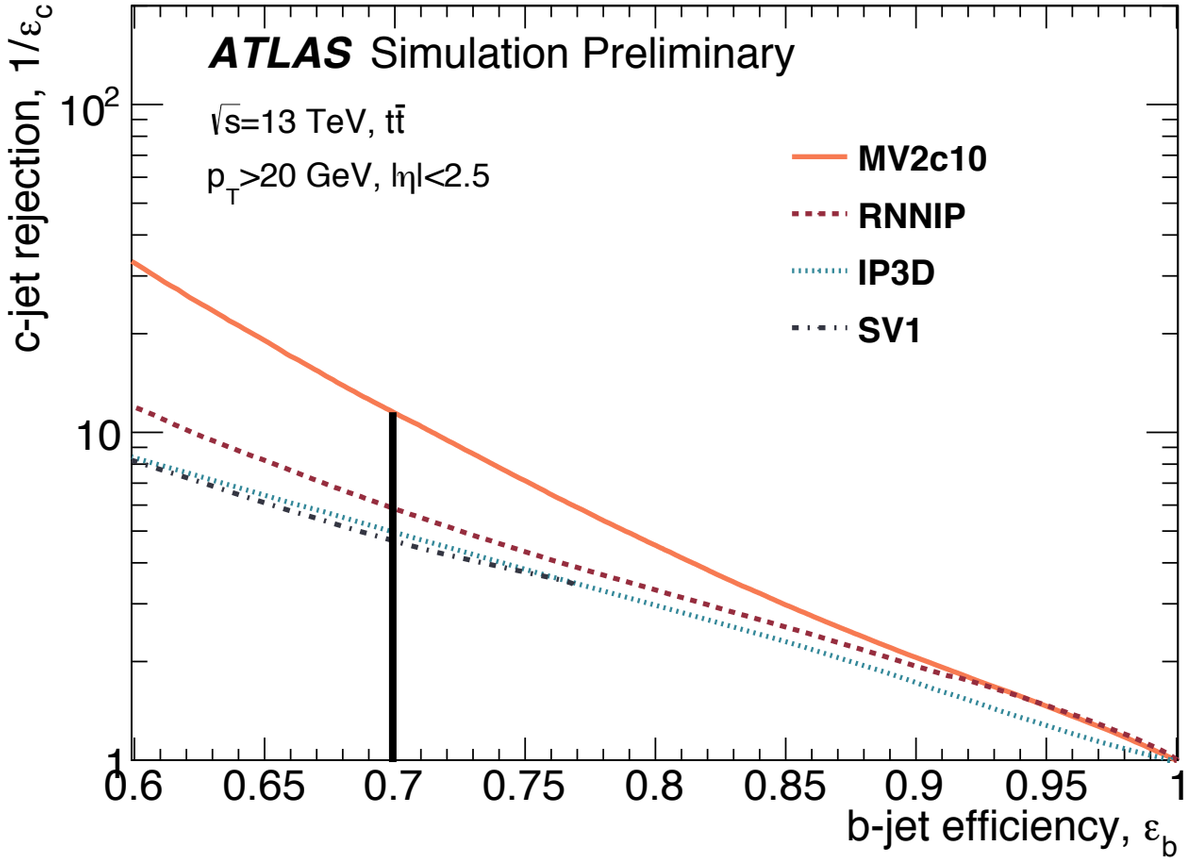
- First steps: ATL-PHYS-PUB-2017-003
  - https://cds.cern.ch/record/2255226



Flavour tagging with recurrent neural networks

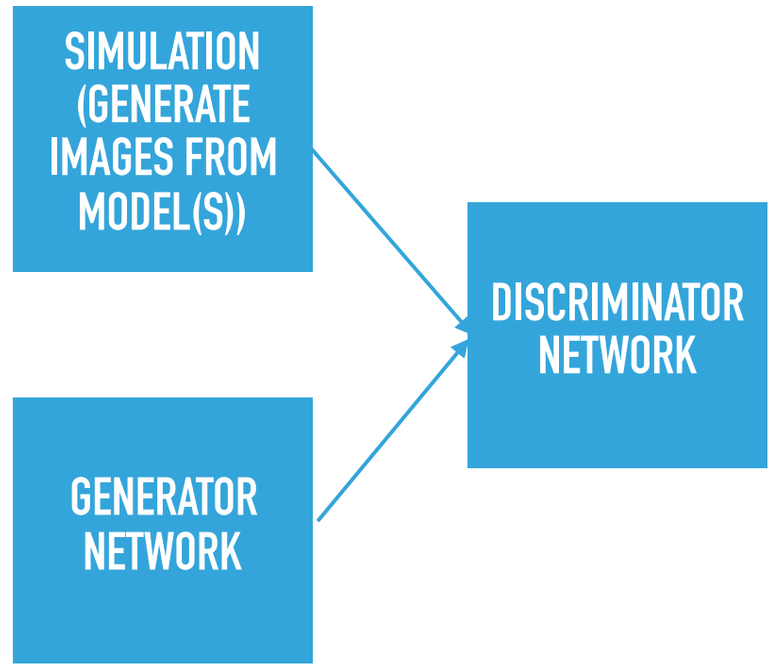
- Promising results
  - RNN outperforms traditional taggers
  - Orange line is for a vertexing-based tagger which is unable to tag the ~20% of events that do not have a vertex





Adversarial training

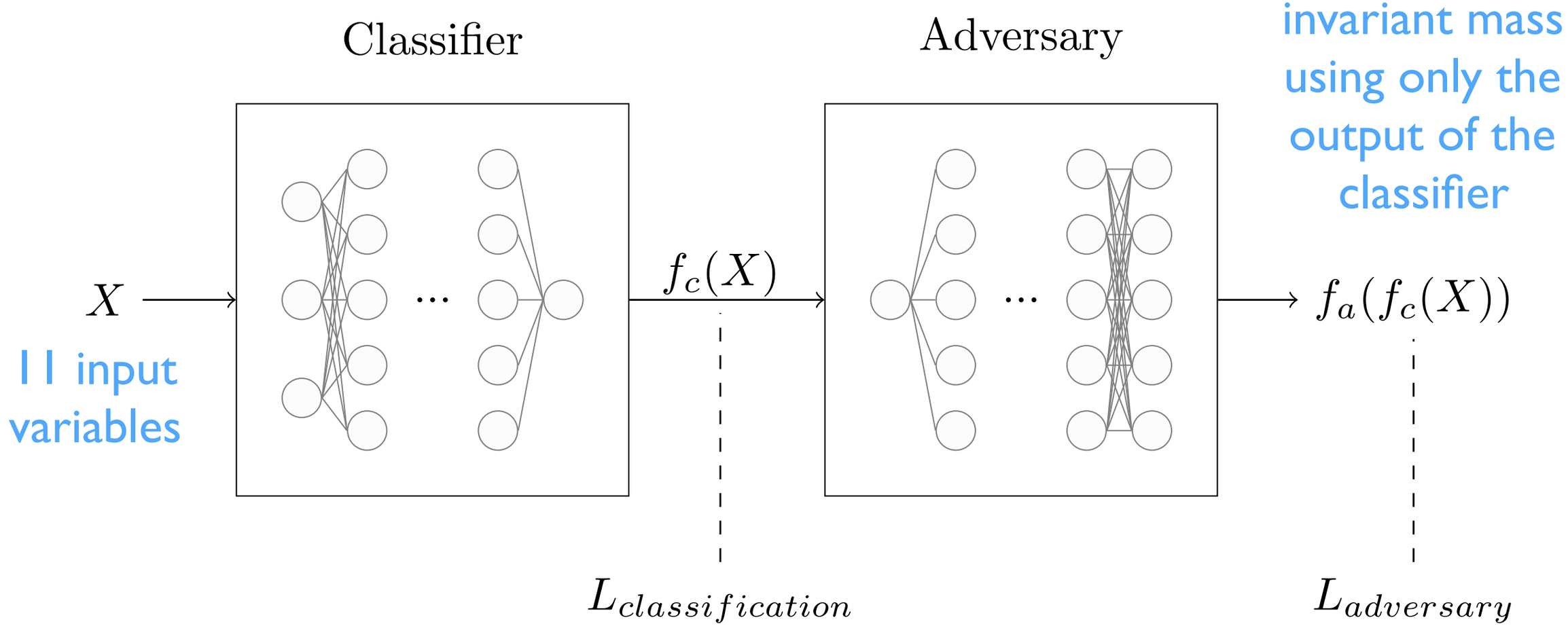
- Idea: two networks working in opposition to one another, with one trying to identify the mistakes of the other
  - optimal solution reached when the opponents are in equilibrium
- Usually discussed as a means of doing ultra-fast simulation (generative-adversarial networks)
  - Generative network trying to generate new events from some template dataset, with the opponent trying to distinguish the new events from those in the template datasets
- But also shown to have potential as a means of reducing the impact of systematics on physics results by decorrelating nuisance parameters



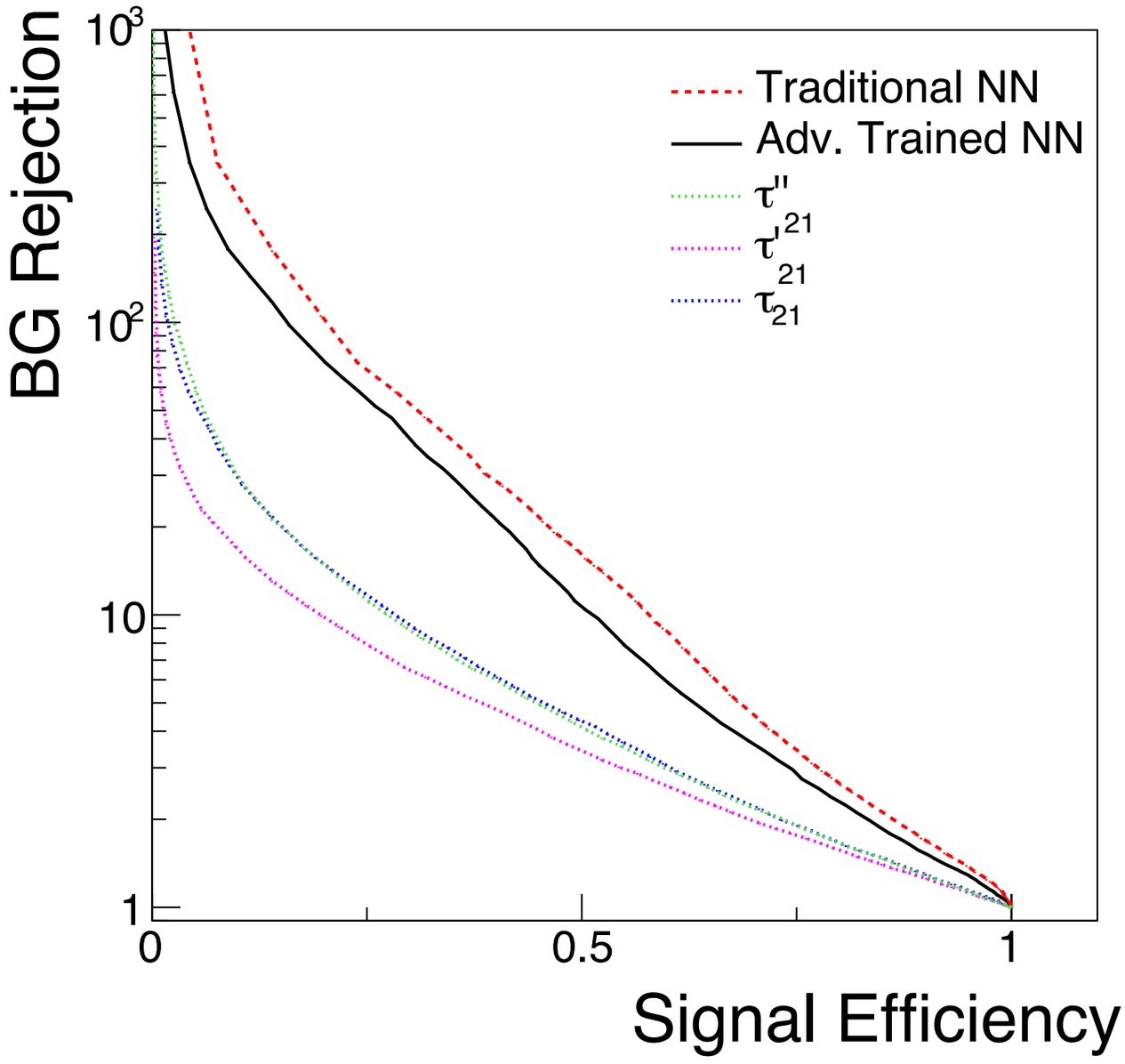
Example of decorrelation using adversarial training

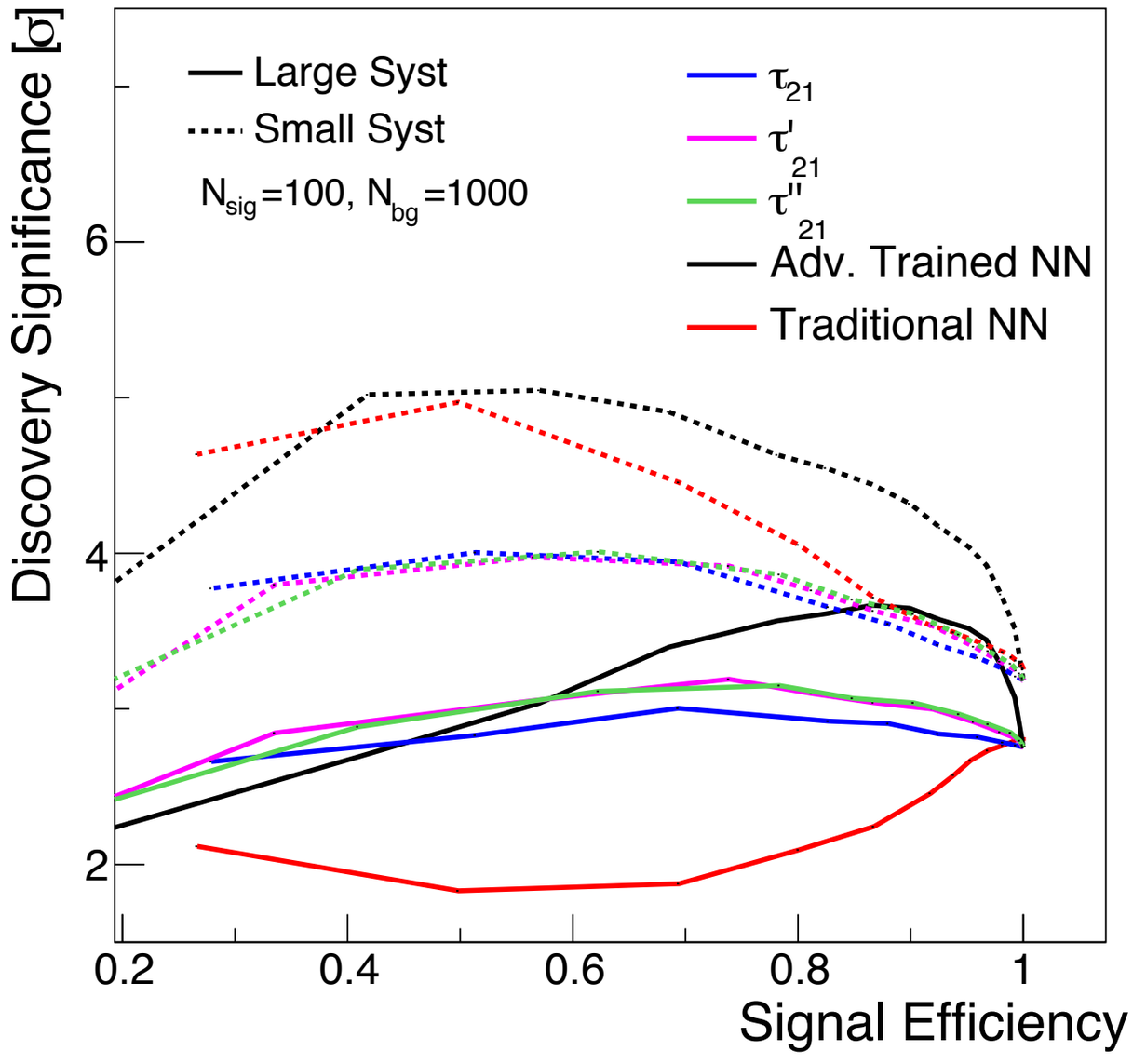
- arXiv:1703.03507v1 (Whiteson, Goul, Søgaard)
- Aims to build a neural network jet substructure tagger for discriminating boosted decay signals while remaining largely uncorrelated with the jet mass

Tries to build invariant mass using only the output of the classifier



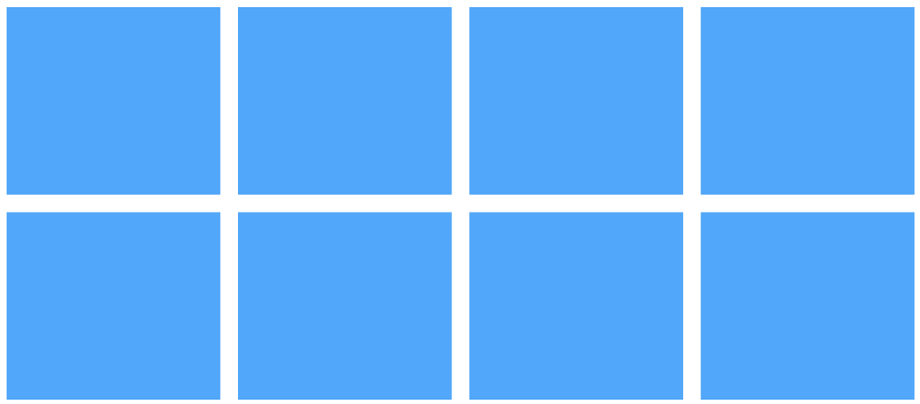
Example of decorrelation using adversarial training



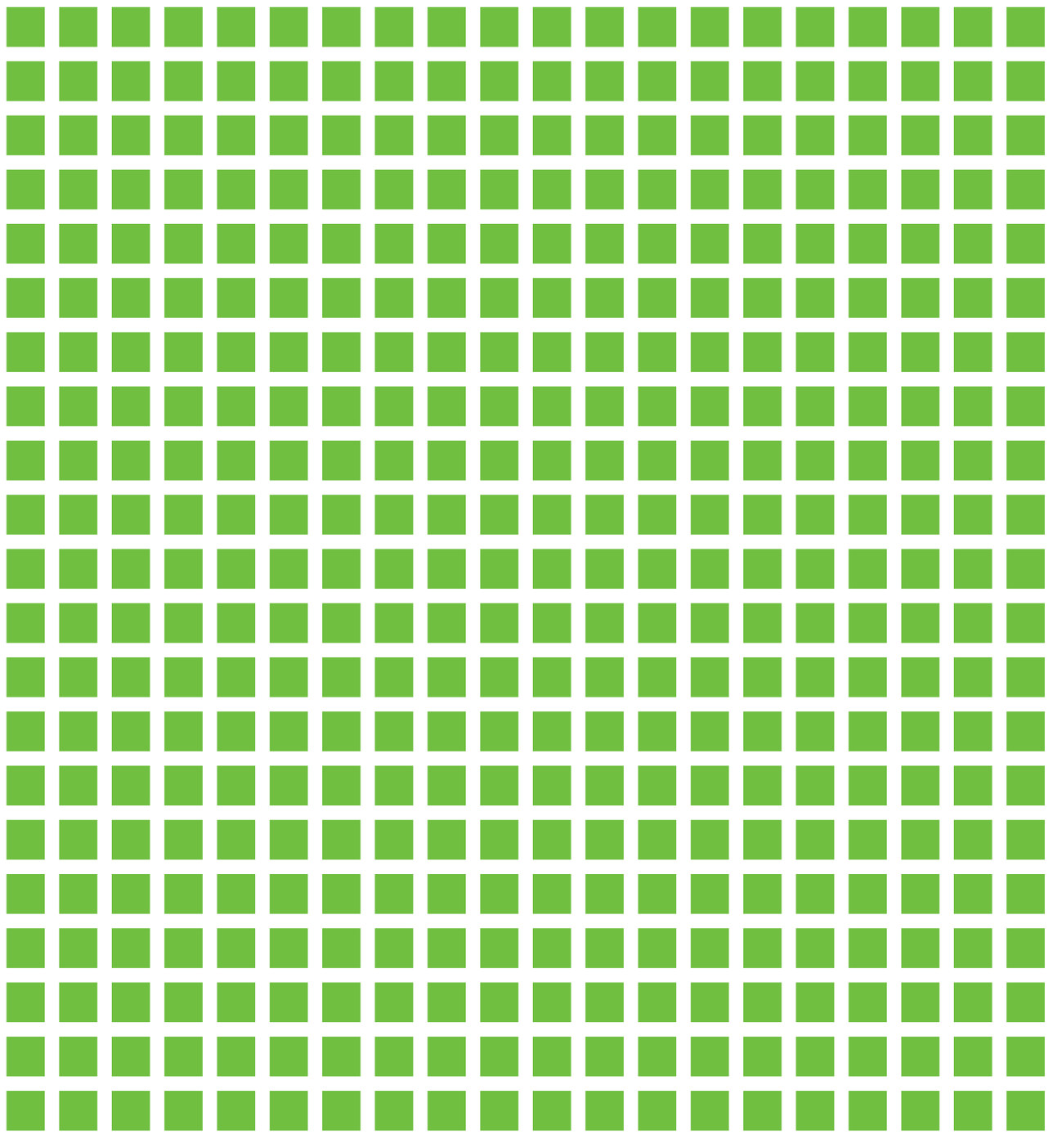


Deep neural networks and GPUs

CPU:
general purpose, higher precision;
faster clock speed → power hungry → few



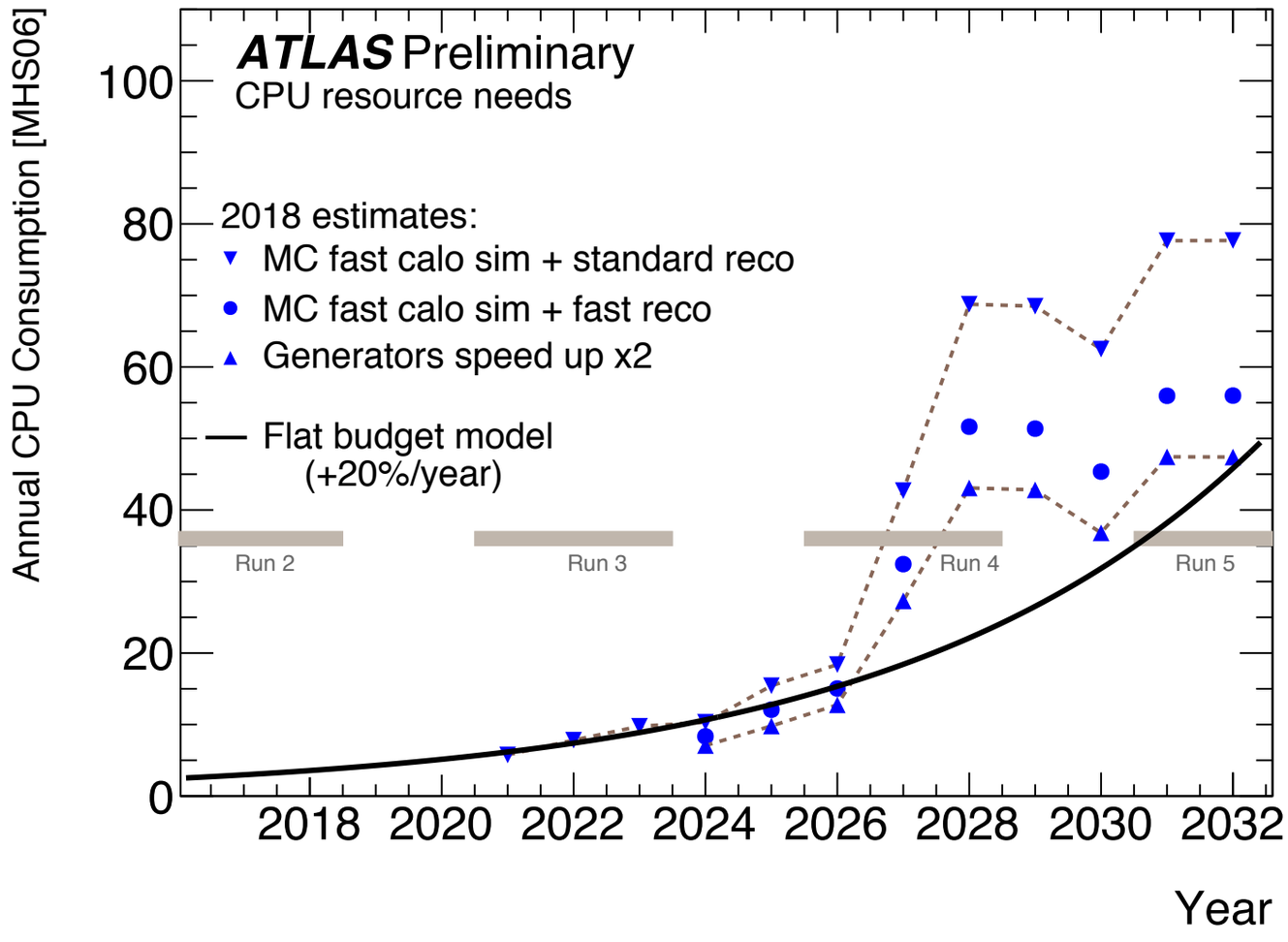
GPU:
optimised for graphics manipulation, lower precision;
slower clock speed → power efficient → many



- Underlying silicon transistor technology is the same
- GPUs are optimised for graphics manipulation
  - simple low precision calculations carried out repeatedly over a very large data space (e.g. the pixels on a screen) → massive parallelism
  - they do *not* work well with branchy, interdependent code requiring high precision (e.g. HEP software)
  - perfectly suited to back propagation and minimisation for deep learning training
- GPUs are becoming a more important component of high performance computers used in research, especially in the US

Deep neural networks and GPUs

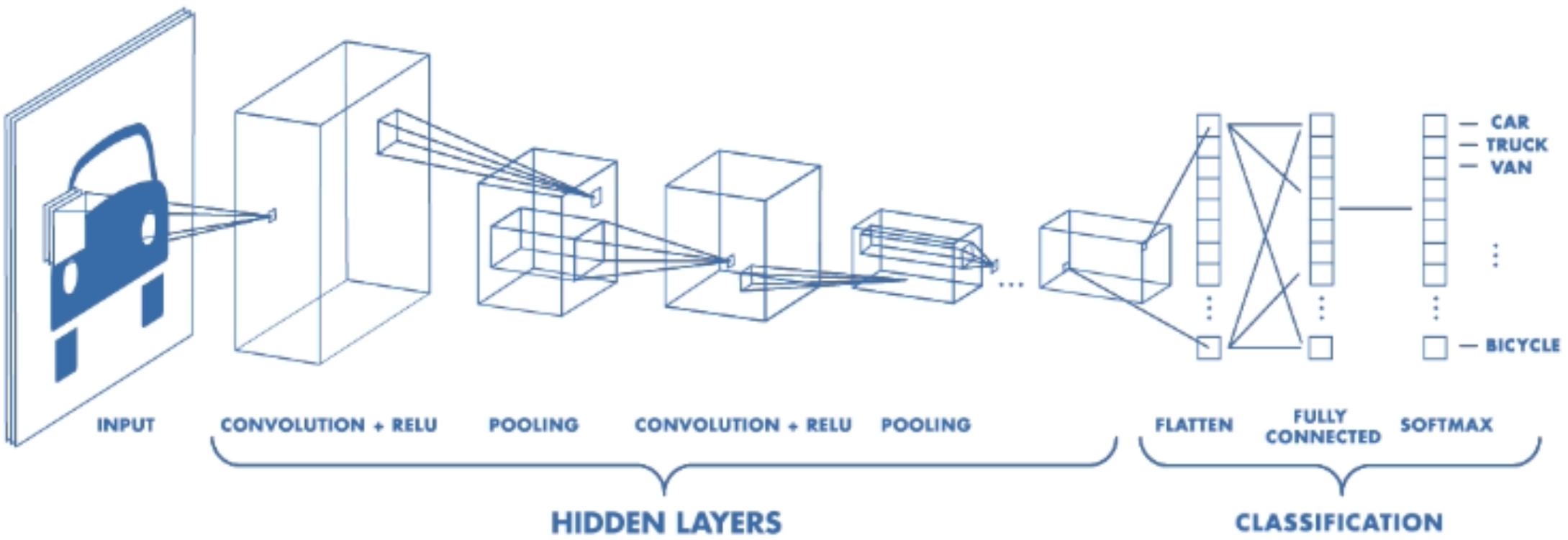
- Graph-computation tools such as Tensorflow are GPU-ready
- Is deep learning the key to unlocking the power of accelerators for HEP?
  - Detector simulation with generative-adversarial techniques and auto-encoders
  - Pattern recognition for tracking
  - Image techniques for jet reconstruction?
- How much of our data processing can we sub-contract to deep learning?
  - Note that Tensorflow can also be used for non-ML computations: possibility for event generation?
- This will be the main question we have to address before HL-LHC…



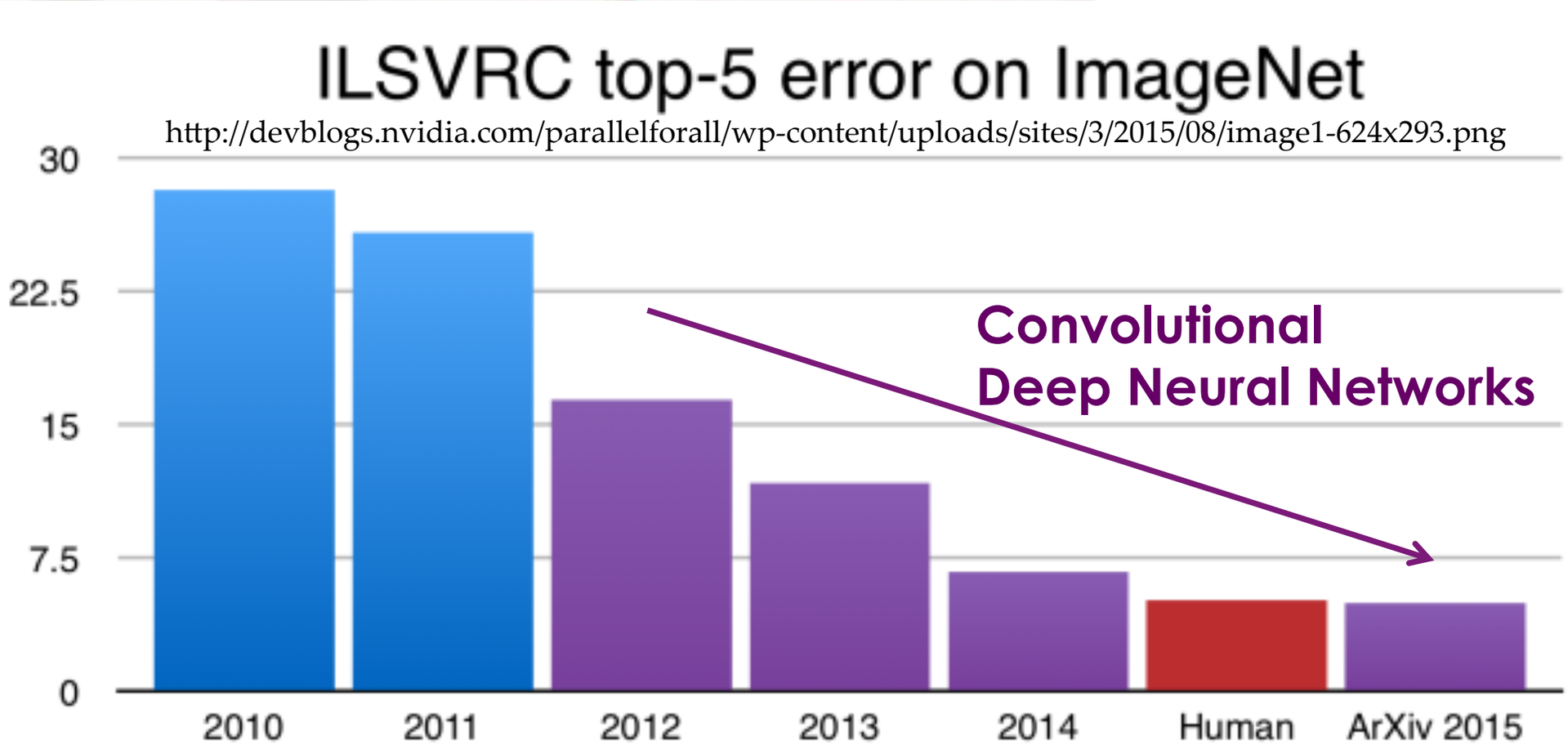
Backup slides

Convolutional neural networks

- Special architecture which explicitly assumes the inputs are images
  - Massively reduces the number of neurons needed to do image recognition
- Main difference: different layers have different behaviours
  - Each layer is "3D", that is as well as the pixel position there is also a depth (colour or level of grey)
  - Convolution filter slides over small areas of the image to build a "feature map"



A. Karpathy
L. Fei-Fei



Open questions

- Issues surrounding training:
  - When is it appropriate to use different training schemes, e.g. n-fold cross validation versus a simple three-way split, etc
  - What is the correct way to deal with very rare backgrounds in the training, where a cross-section corrected weighting will give a handful of events only (e.g. triboson and Higgs backgrounds)
- Dealing with unbalanced samples - comparisons of different ML algorithms are often shown based on equally sized "signal" and "background" samples, but in real life one usually has much more of the latter than the former. How should this be dealt with in the training? Do we need to reconsider the statistics we request for signal processes (see the next point)
- Strategy for MC production requests when use of machine learning is planned
- Hyperparameter tuning - what is recommended here? Grid searches, random searches etc?
- Variable selection - what is the best way of choosing which variables to keep and which to drop? One-at-a-time? BDT/NN variable importance measures? Studying the variation of the output w.r.t. each variable? Etc. What is the role of "really understanding/reproducing" variables in order that they play a decisive/important role?
- Dealing with variables that are undefined for some events - quite often one will encounter "-999" etc in an n-tuple for cases where the variables isn't defined, e.g. "pT of the 5th jet" etc. What should one do with these cases? Some machine learning literature advocates replacing them with random noise or the mean of the other variables in the column, but this doesn't seem appropriate for our field. One could transform them to categorical variables that are always defined?
- Memory issues - I guess this differs from case to case, but quite often one has a very large dataset that it isn't practical to load into memory at once. Some algorithms are more amenable to training in batches than others (easy for neural networks but less obvious for ensemble methods like BDTs).