

Machine Learning: short introduction (Trees and ANN)

Sergey Korpachev

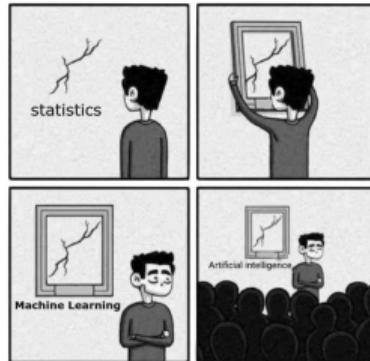
Outline

- Machine learning (ML)
- Data
- Features in ML
- ML pipeline
- Linear model
- Decision tree
- Neural network

Machine learning

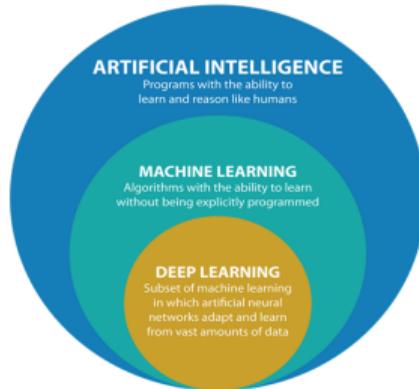
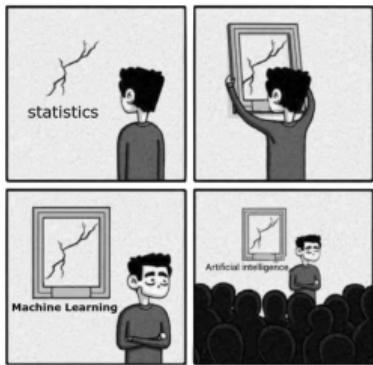
Machine learning

—○ what is it?



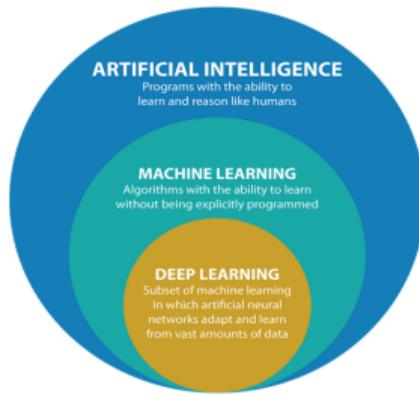
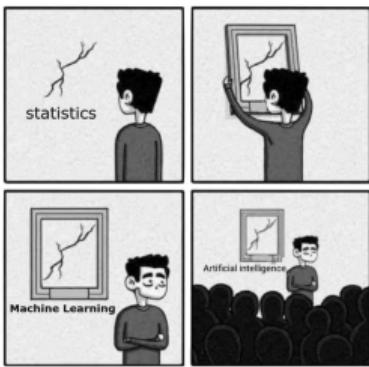
Machine learning

—○ what is it?



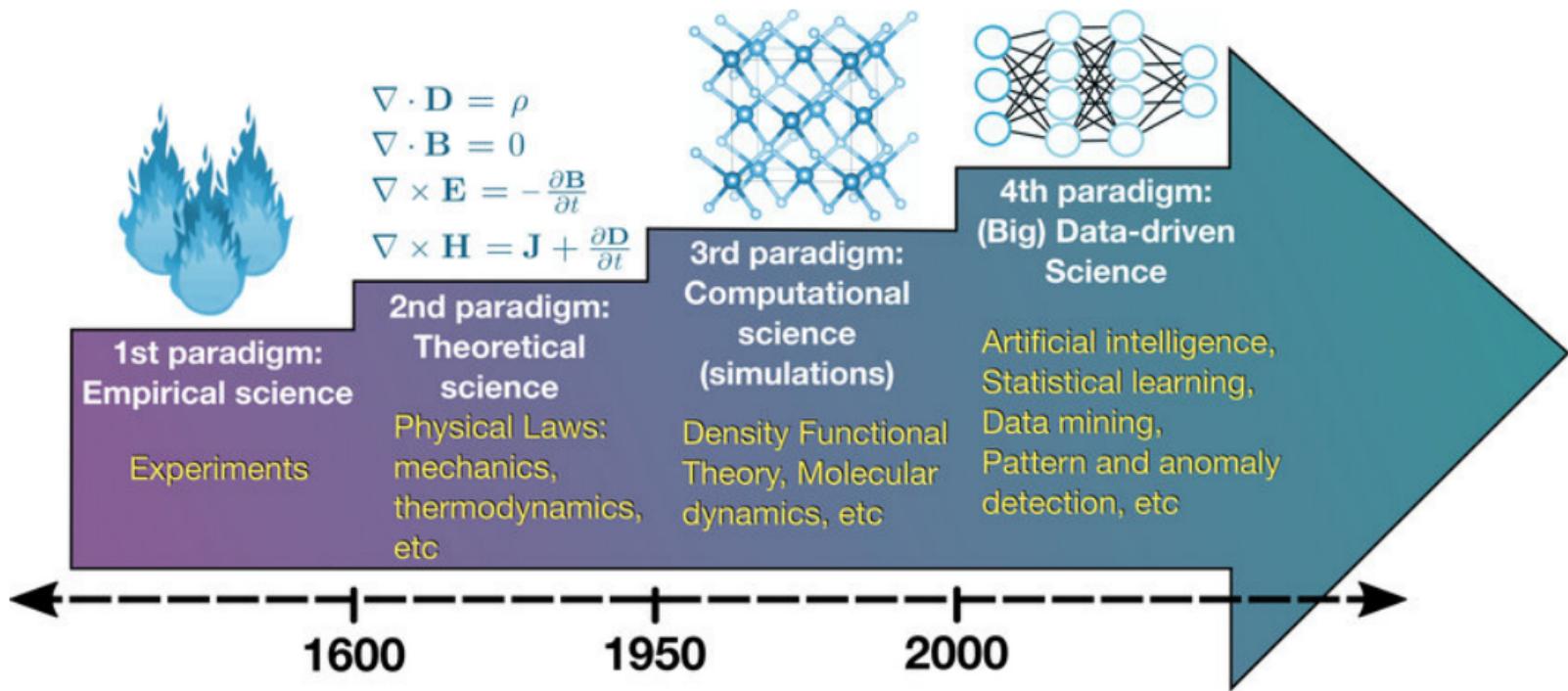
Machine learning

— o what is it?



Machine learning

—○ The Fourth Paradigm (Tony Hey, ...)



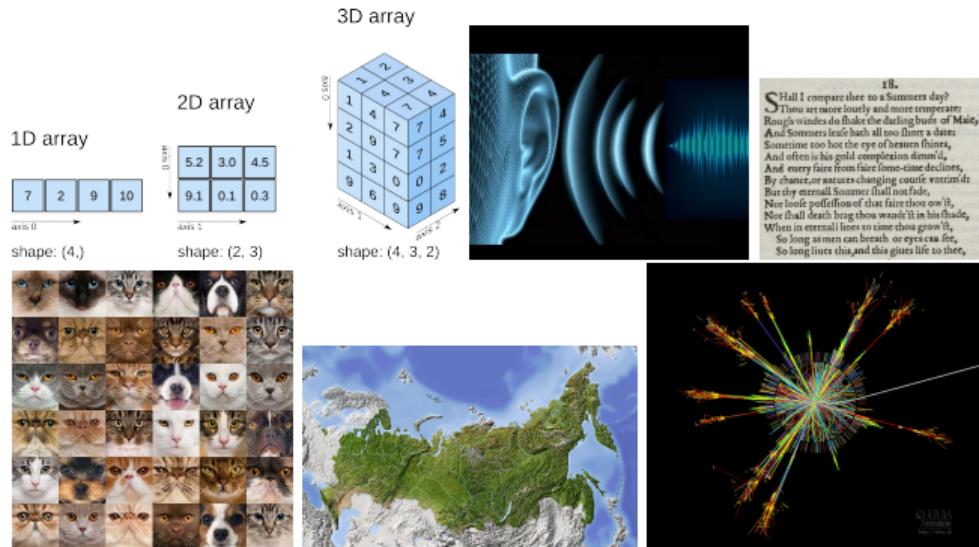
Data

Data

—○ what is data?

Anything can be data:

- ① Numbers
- ② Text
- ③ Images
- ④ Sound
- ⑤ Geomap
- ⑥ Particle collisions
- ⑦ Knowledge
- ⑧ You name it



Features in ML

Features in ML

—○ not HEP

Supervised learning

Classification

- cat, dog or muffin
- relevant or spam
- disease or not
- good or bad
- ...

Regression

- rent price
- temperature
- annual profit
- driving time
- ...

Don't forget about **unsupervised learning, reinforcement learning, semi-supervised learning** and so on.

Supervised learning

Classification

- b , c, uds jet
- π , K, μ particle
- $t\bar{t}$ or QCD event
- select or reject trigger candidate
- ...

Regression

- energy resolution
- pile-up mitigation
- ...

Don't forget about **unsupervised learning, reinforcement learning, semi-supervised learning** and so on.

Features in ML

—○ ML challenges: slide 1 (A. Ustyuzhanin)

- Precise and fast particle tracking (single tracks, shower, jets)
- Particle identification
- Fast and accurate online data processing and filtering
- Anomaly detection (data quality monitoring, infrastructure monitoring)
- Detector design optimization (bayesian optimization, surrogate modelling)
- Data analysis (signal from background separation, ...)
- Simulation (speed-up simulation using generative models, simulator parameters optimization - tuning)
- ...

Features in ML

 ——→ ML challenges: slide 2 (A. Ustyuzhanin)

Tracking system features

- Particle momentum
- Particle charge
- Track parameters
- Quality of track fit
- Number of track hits
- ...

RICH features

- Angle θ
- Quality of angle reconstruction
- Reconstructed particle type
- Reconstructed particle energy
- Light intensity
- ...

Calorimeter features

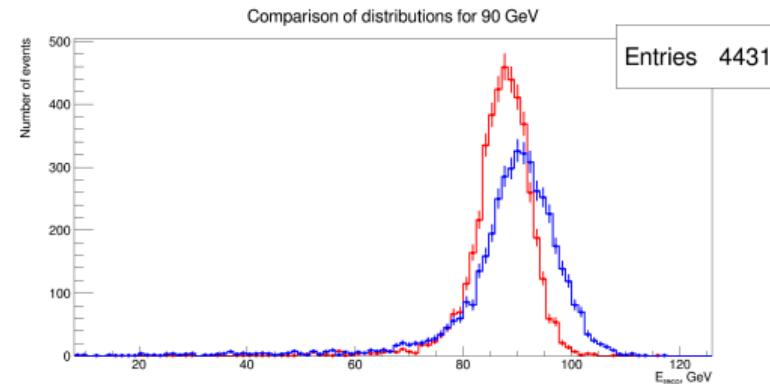
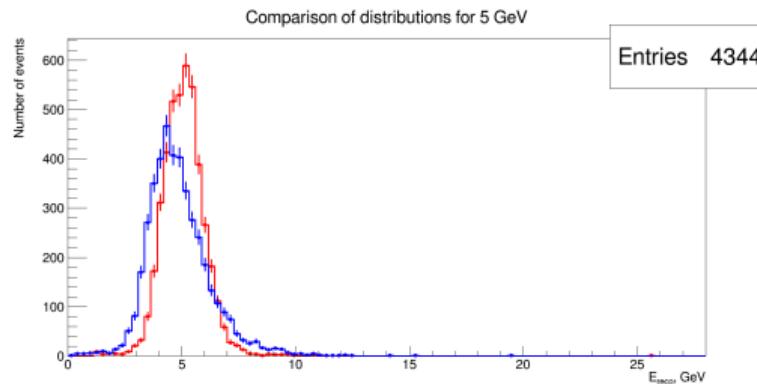
- Measured particle energy
- Shower parameters: length, width, ...
- Number of clusters in each layer Intensity of the clusters
- Intensity of the clusters
- Distance from track of the original particle
- ...

Muon detector features

- Muon track parameters
- Quality of track fit
- Number of active layers
- Distance between the track and the active layers
- Length of shower
- ...

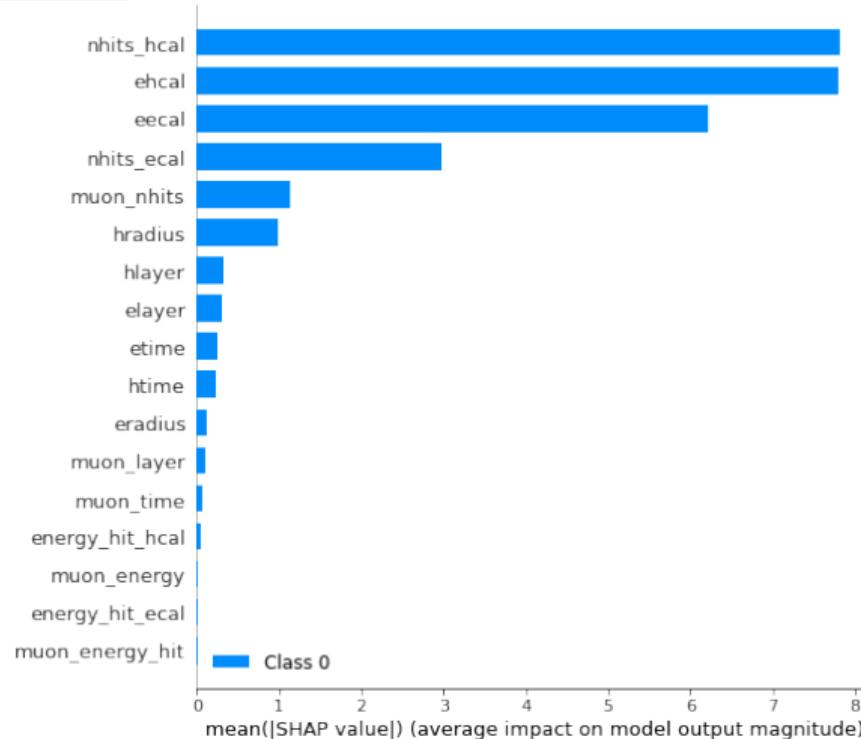
Features in ML

—○ Example of my regression problem



Features in ML

—○ SHAP value

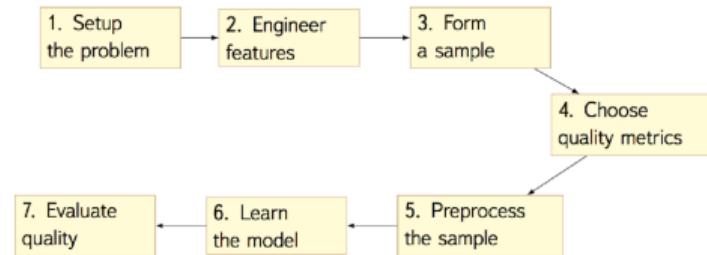


ML pipeline

ML pipeline

— o what is ML pipeline?

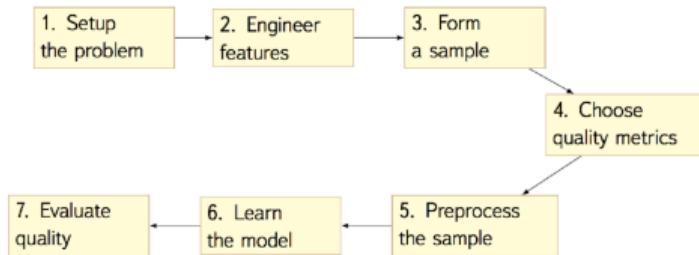
Machine Learning Pipeline



ML pipeline

— o what is ML pipeline?

Machine Learning Pipeline

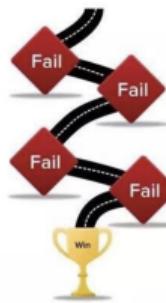


Machine Learning Pipeline

What Most People Think



What Successful People Know

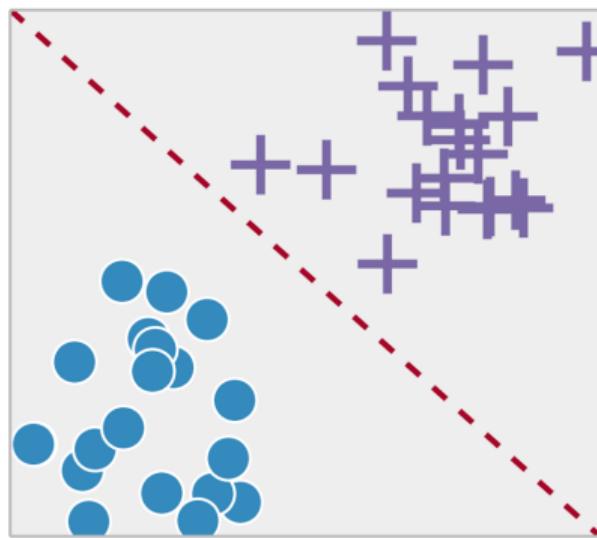


Linear model

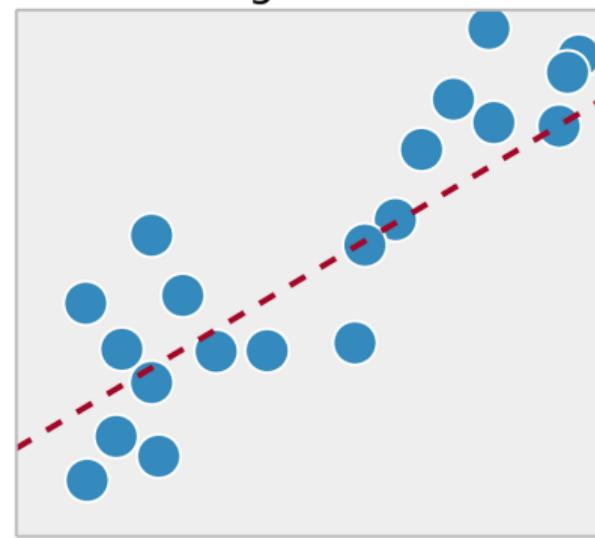
Linear model

—○ classification and regression

Classification



Regression



Linear model

—○ regression

- $Y = \mathbb{R}$
- N objects with K real features: $D = \mathbb{R}^K$
- $a(\mathbf{x}, \boldsymbol{\theta}) = \theta_0 + \sum_{j=1}^K f_j(\mathbf{x}) \cdot \theta_j$
- Extend and reassign:
 $[1, f_1(\mathbf{x}), \dots, f_K(\mathbf{x})] \equiv \mathbf{x}$
 $[\theta_0, \dots, \theta_K] \equiv \boldsymbol{\theta}$
- Then $a(\mathbf{x}, \boldsymbol{\theta}) = \langle \mathbf{x}, \boldsymbol{\theta} \rangle$
- Minimization problem:
 - $\mathcal{L}(\boldsymbol{\theta}) = (\langle \mathbf{x}, \boldsymbol{\theta} \rangle - y)^2$
 - $Q(\boldsymbol{\theta}) = \frac{1}{N} \sum_{i=1}^N (\langle \mathbf{x}_i, \boldsymbol{\theta} \rangle - y_i)^2$
- Then we need to minimize $Q(\boldsymbol{\theta})$ by varying $\boldsymbol{\theta}$:
 - $\hat{\boldsymbol{\theta}} = \arg \min_{\boldsymbol{\theta} \in \Theta} Q(\boldsymbol{\theta})$

Linear model

—○ classification

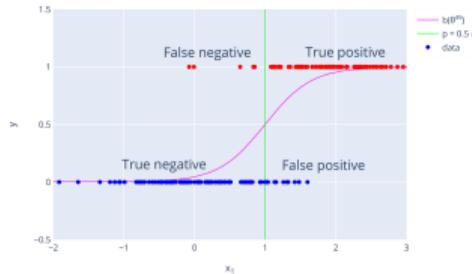
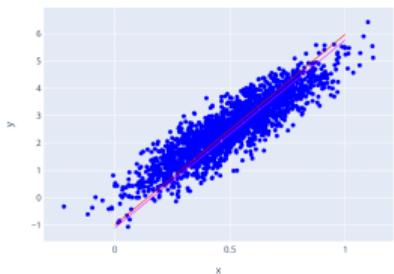
- $Y = \{-1, +1\}$
- N objects with K real features: $D = \mathbb{R}^K$
- $a(\mathbf{x}, \boldsymbol{\theta}) = \theta_0 + \sum_{j=1}^K f_j(\mathbf{x}) \cdot \theta_j$
- Extend and reassign:
 $[1, f_1(\mathbf{x}), \dots, f_K(\mathbf{x})] \equiv [1, x_1, \dots, x_K] \equiv \mathbf{x}$
 $[\theta_0, \dots, \theta_K] \equiv \boldsymbol{\theta}$
- Then $a(\mathbf{x}, \boldsymbol{\theta}) = \langle \mathbf{x}, \boldsymbol{\theta} \rangle$

- Minimization problem:
 - $\mathcal{L}(\boldsymbol{\theta}) = [\text{sign}\langle \mathbf{x}, \boldsymbol{\theta} \rangle \neq y]$
 - $Q(\boldsymbol{\theta}) = \frac{1}{N} \sum_{i=1}^N [\text{sign}\langle \mathbf{x}_i, \boldsymbol{\theta} \rangle \neq y_i]$
- Then we need to minimize $Q(\boldsymbol{\theta})$ by varying $\boldsymbol{\theta}$:
 - $\hat{\boldsymbol{\theta}} = \arg \min_{\boldsymbol{\theta} \in \Theta} Q(\boldsymbol{\theta})$

Decision tree

Decision tree

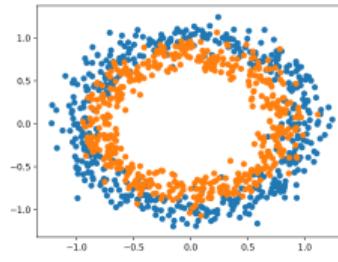
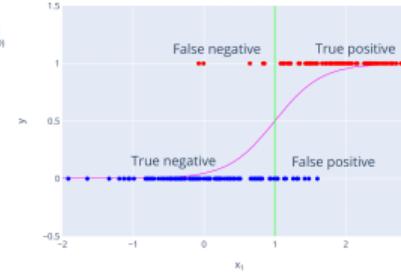
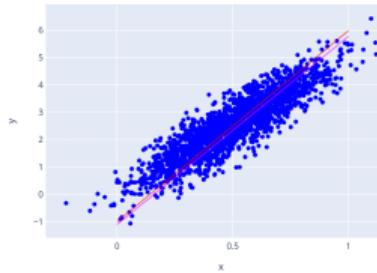
—○ motivation



- In the previous part we explored linear models:
 - nicely describe linear correlations
 - fast and easy to train

Decision tree

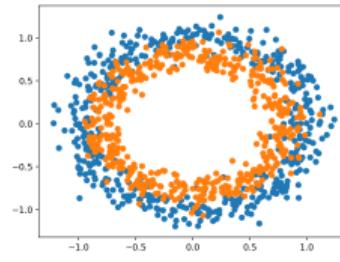
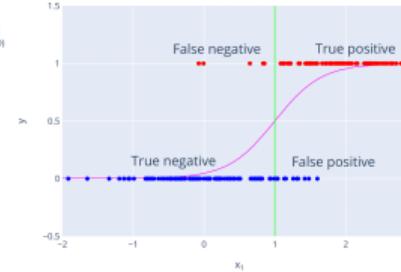
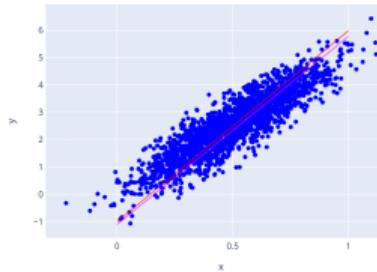
—○ motivation



- In the previous part we explored linear models:
 - nicely describe linear correlations
 - fast and easy to train
- However, they are poor in modelling non-linearities
- Additional heuristics might be applied (e.g. polynomial features) but are ad-hoc

Decision tree

—○ motivation



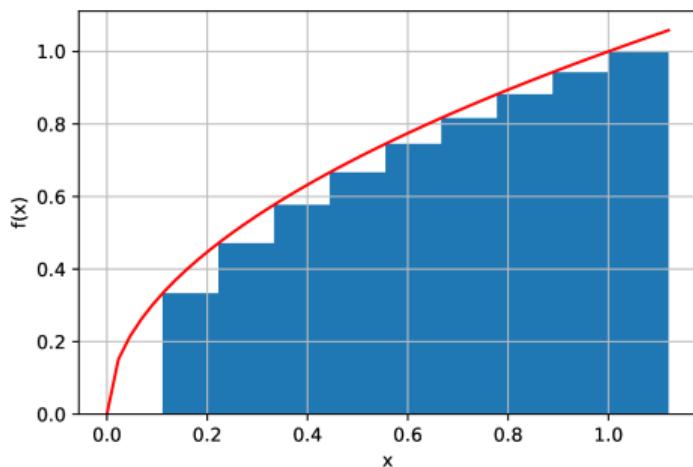
- In the previous part we explored linear models:
 - nicely describe linear correlations
 - fast and easy to train
- However, they are poor in modelling non-linearities
- Additional heuristics might be applied (e.g. polynomial features) but are ad-hoc

Is there a more general way to describe non-linearities?

Decision tree

—○ motivation

- What is the simplest way to approximate a function?
- Using **piecewise linear function**

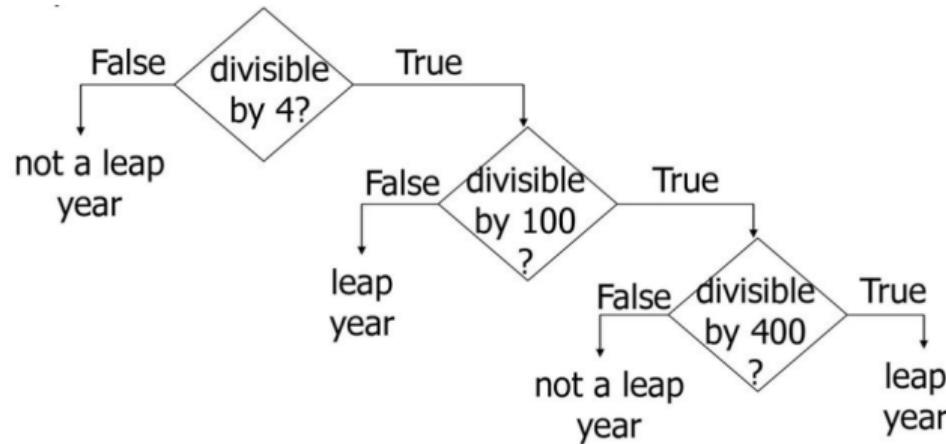


$$f(x) = \begin{cases} 0, & x < 0.12 \\ 0.35, & 0.12 \leq x < 0.22 \\ 0.47, & 0.22 \leq x < 0.36 \\ \dots \end{cases}$$

Decision tree

— o intuition

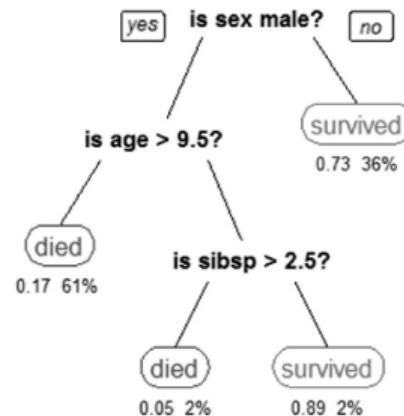
- What is the simplest way to categorize things?
- Ask yes/no questions



Decision tree

— o intuition

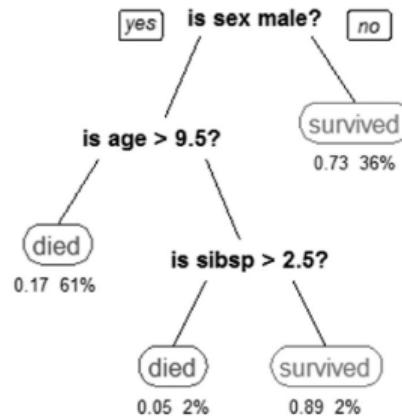
- What is the simplest way to categorize things?
- Ask yes/no questions



Decision tree

— o intuition

- What is the simplest way to categorize things?
- Ask yes/no questions



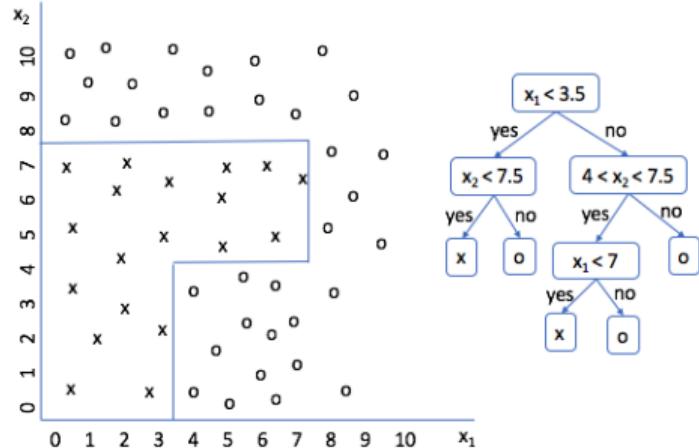
Can we mathematically formulate this?

Decision tree

—○ algorithm

Algorithm 1: Decision tree

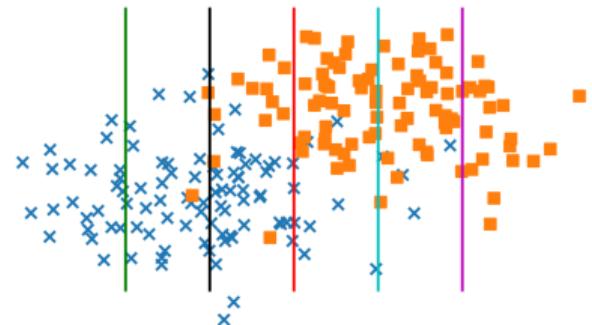
- 1: Initialize: hyperparameters, leaf set = $\{X_{\text{train}}\}$
- 2: **while** not stopping criteria **do**
- 3: leaf to split = Choose(leaf set)
- 4: left leaf, right leaf = Split(leaf to split)
- 5: Add left leaf, right leaf to leaf set
- 6: Remove leaf to split from leaf set
- 7: **end while**



Decision tree

—○ split criteria

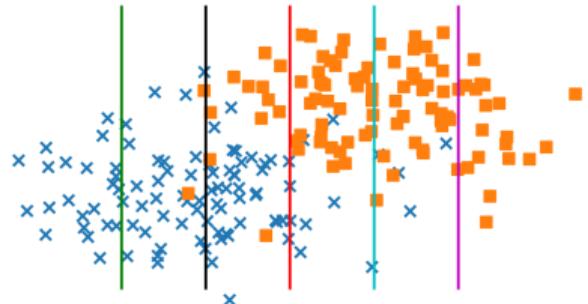
- Suppose there is a classification problem
- So we want to separate one class from the other by constructing **decision boundary**
- And using decision tree algorithm described earlier
- To do that we need to start splitting on features
- Which **split** is the best?



Decision tree

—○ split criteria

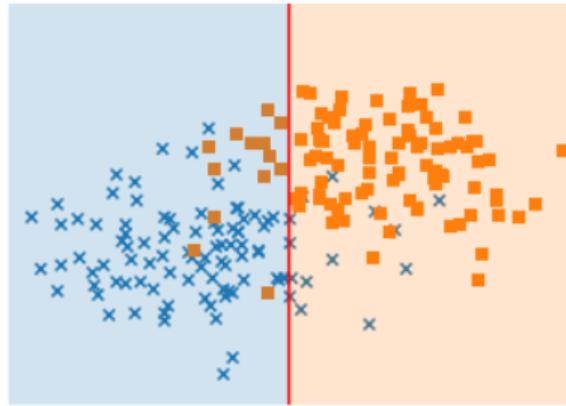
- Suppose there is a classification problem
- So we want to separate one class from the other by constructing **decision boundary**
- And using decision tree algorithm described earlier
- To do that we need to start splitting on features
- Which **split** is the best?
- We need some criteria



Decision tree



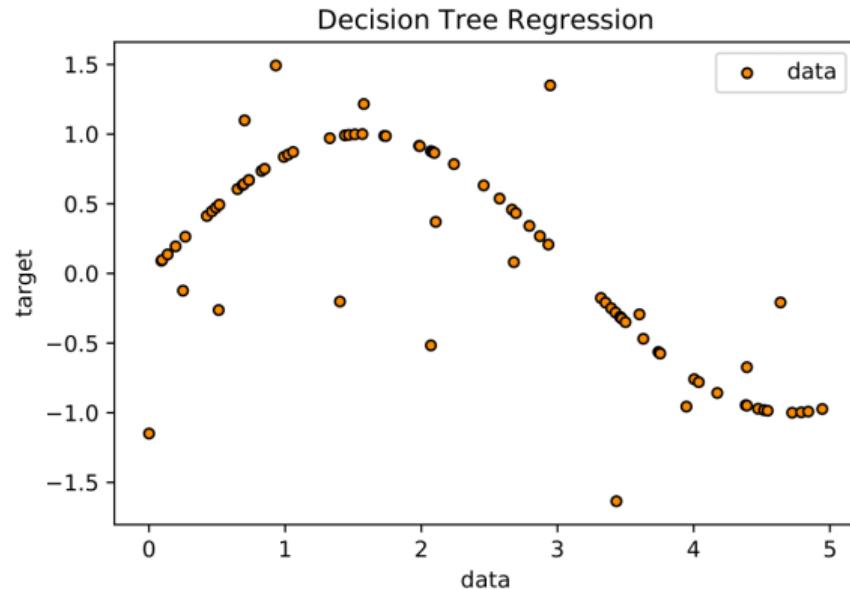
split criteria



The “best” split is a central one because it introduces **purity** in the best way. And we have some functions to measure the purity!

Decision tree

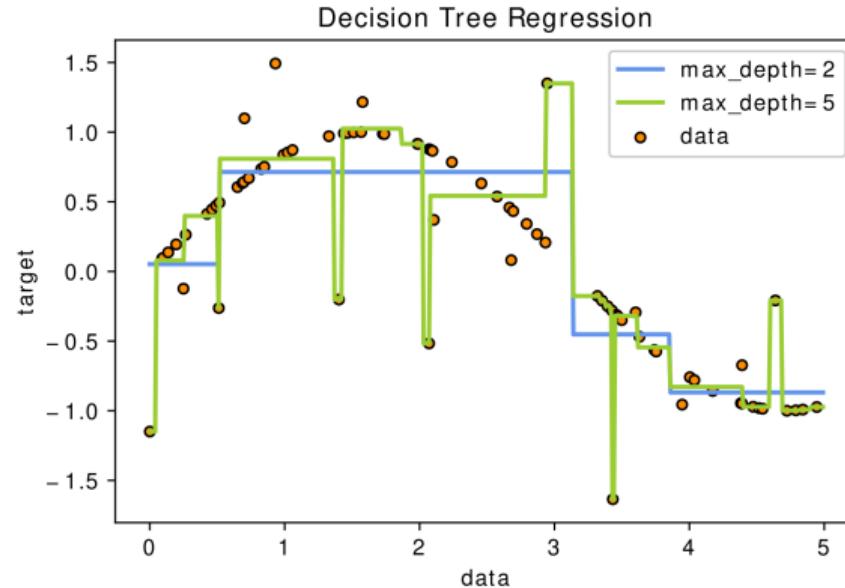
—○ regression



Can we use decision tree approach for regression problem?

Decision tree

—○ regression



Can we use decision tree approach for regression problem? Yes

Decision tree

—○ growing

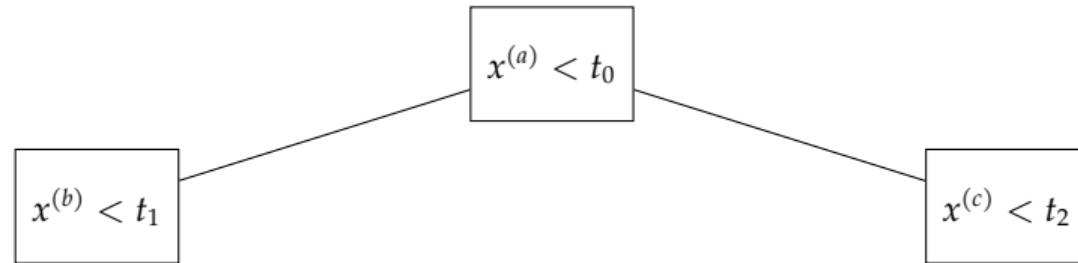
What about tree construction?

$$x^{(a)} < t_0$$

Decision tree

—○ growing

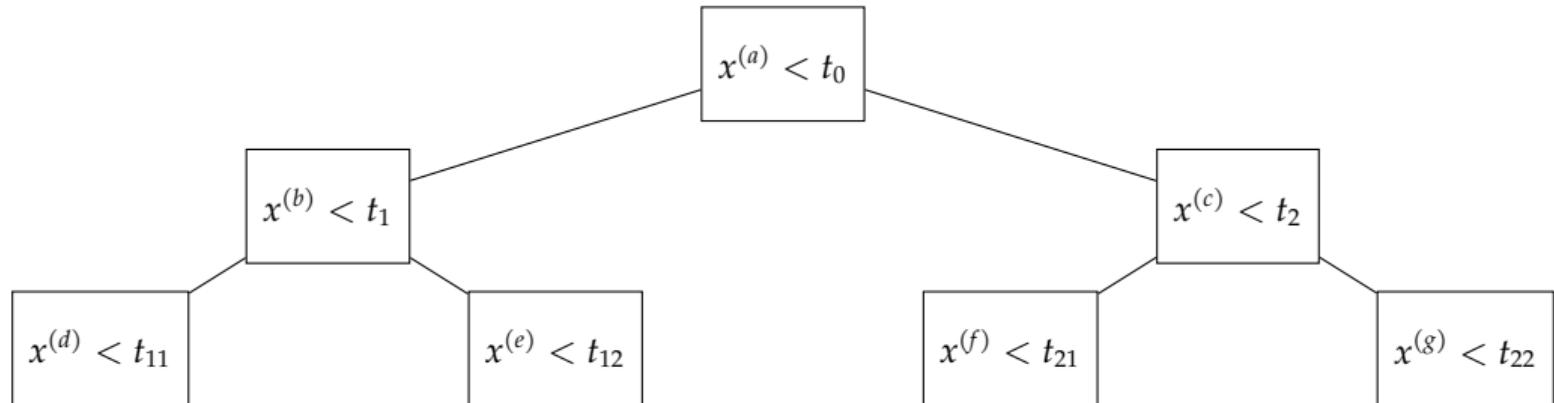
What about tree construction? Do it recursively by greedy algorithm



Decision tree

—○ growing

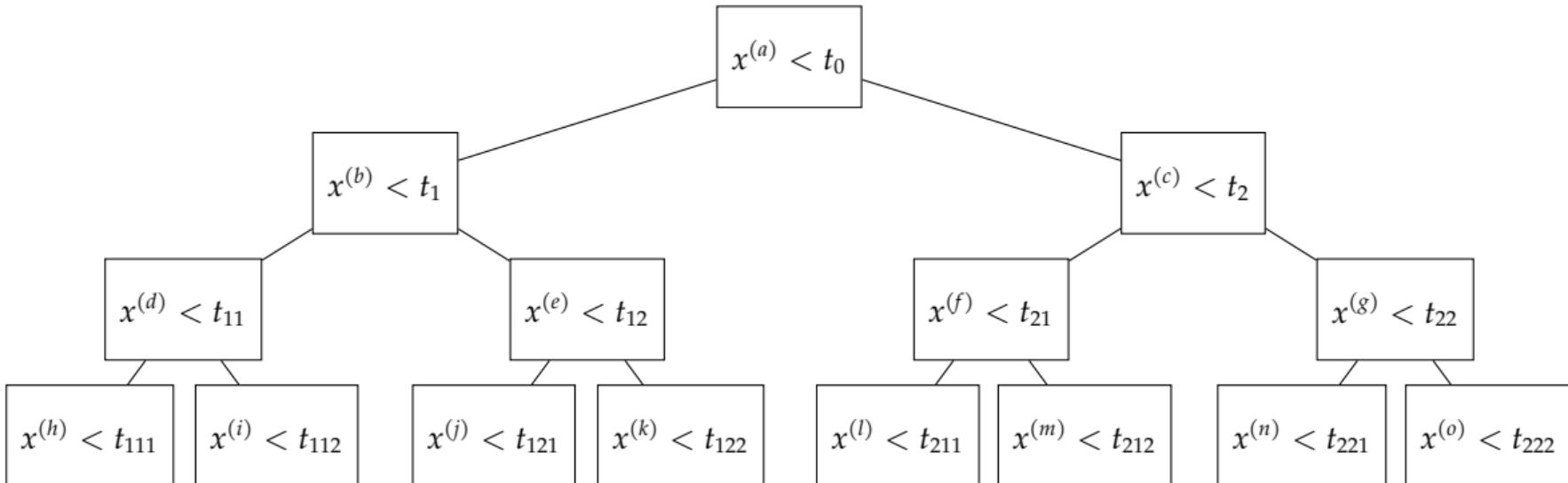
What about tree construction? Do it recursively by **greedy** algorithm



Decision tree

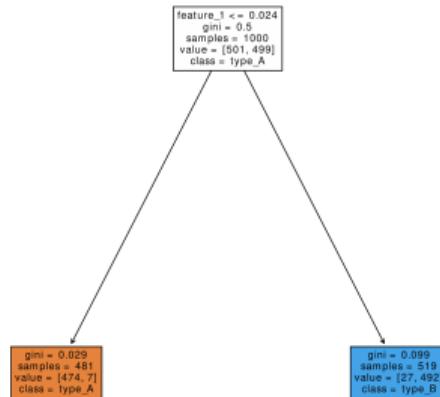
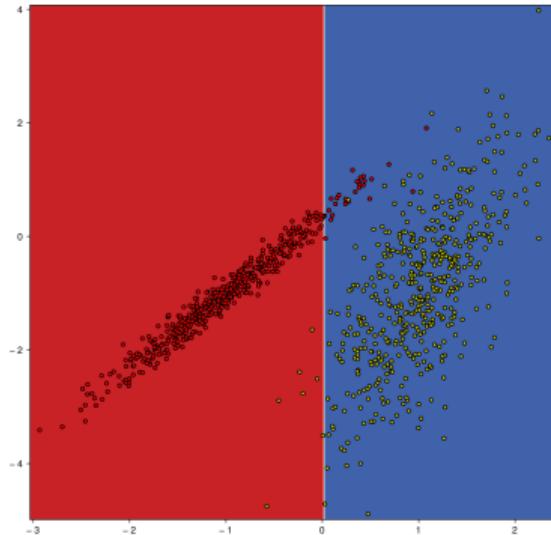
—○ growing

What about tree construction? Do it recursively by greedy algorithm



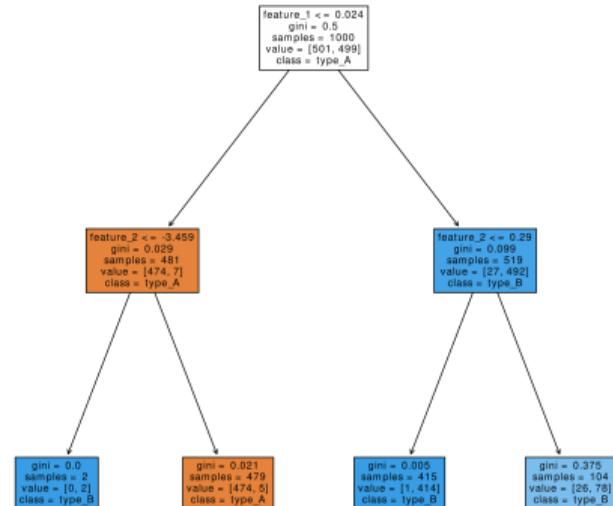
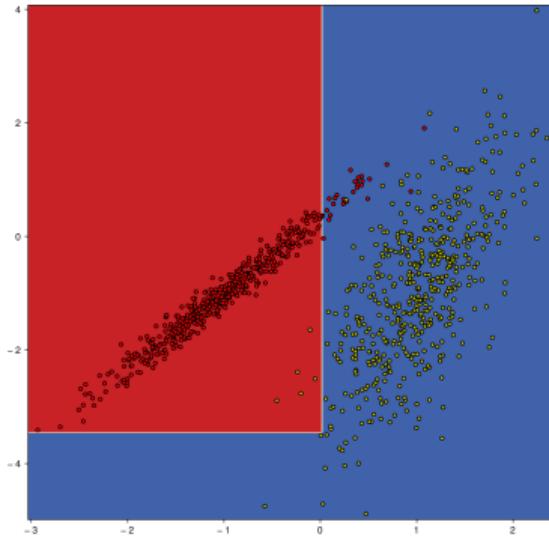
Decision tree growing

check out also [these](#) visuals



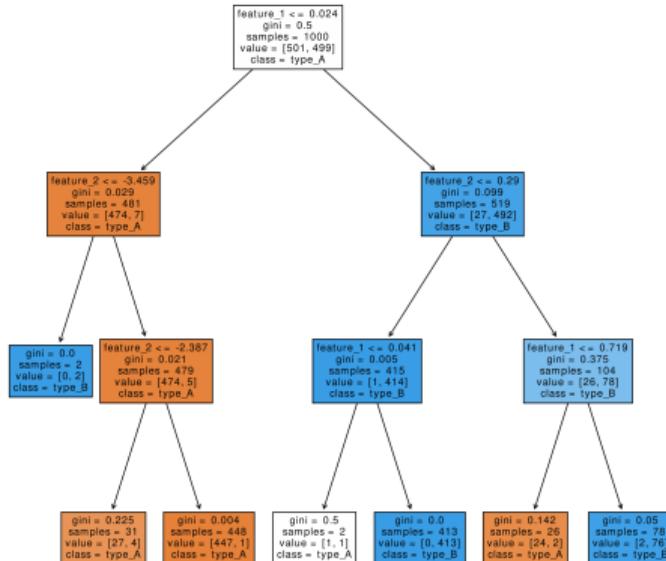
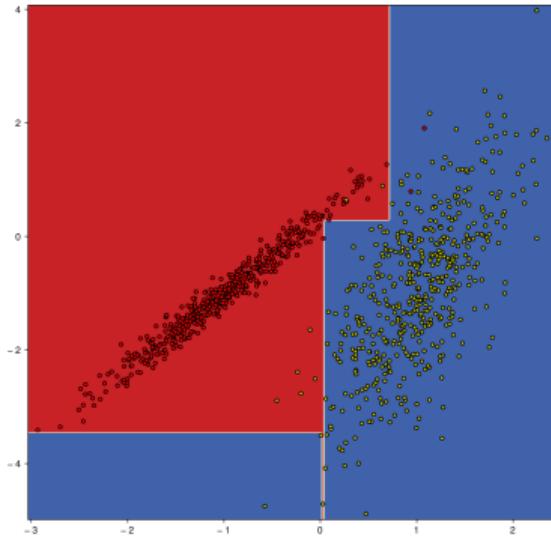
Decision tree growing

check out also these visuals



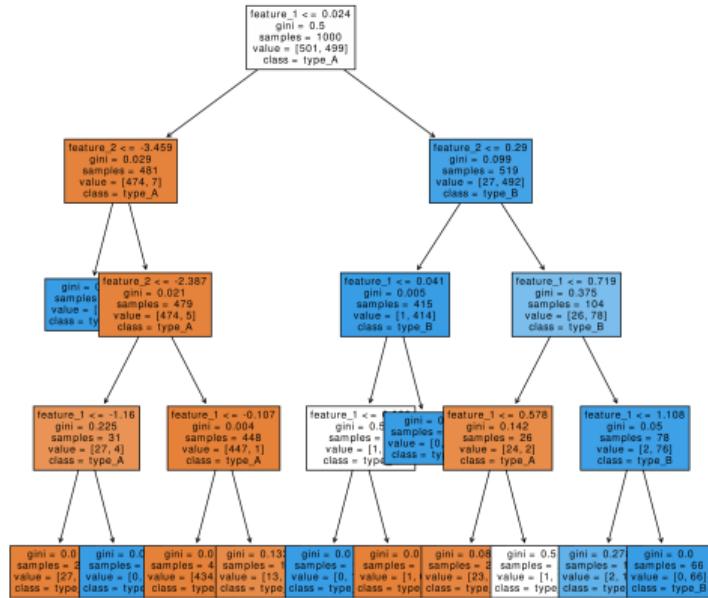
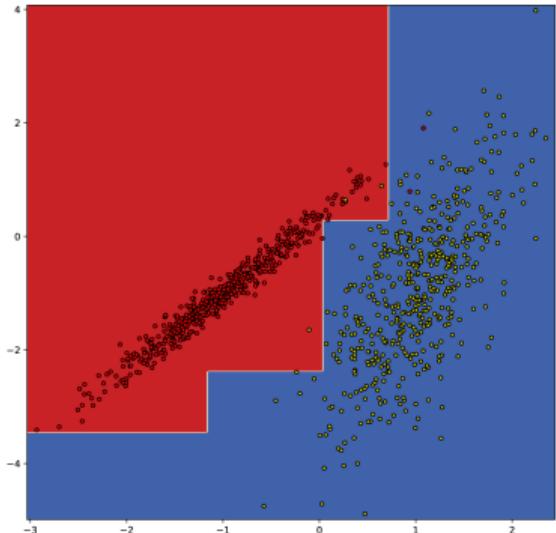
Decision tree \longrightarrow growing

check out also [these](#) visuals



Decision tree growing

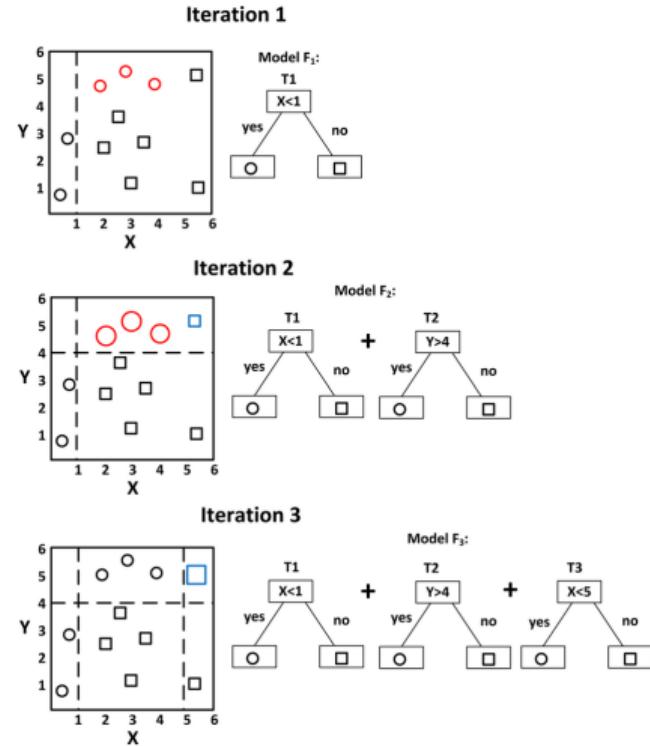
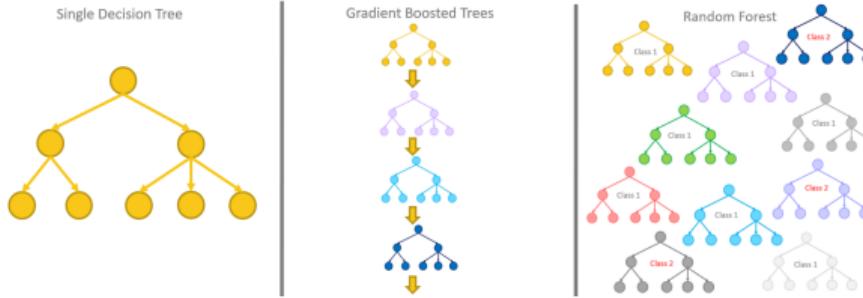
check out also these visuals



Random forest and gradient boosting

Random forest and gradient boosting

—○ what's next after decision trees?

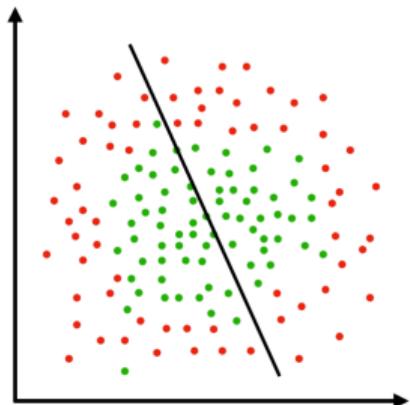


Modelling nonlinearities

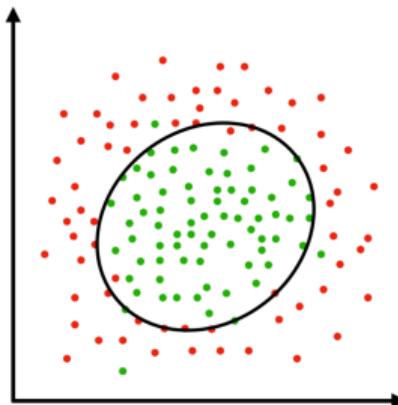
Modelling nonlinearities

linear models

What we have



What we want

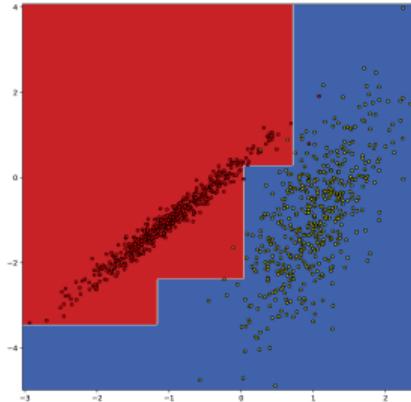
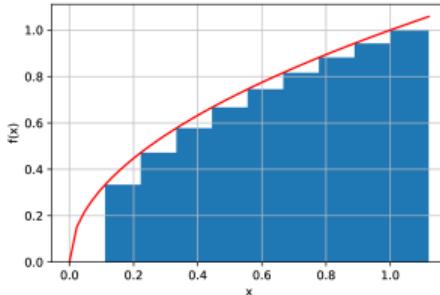


Linear models can't simply describe complex nonlinear data

Modelling nonlinearities

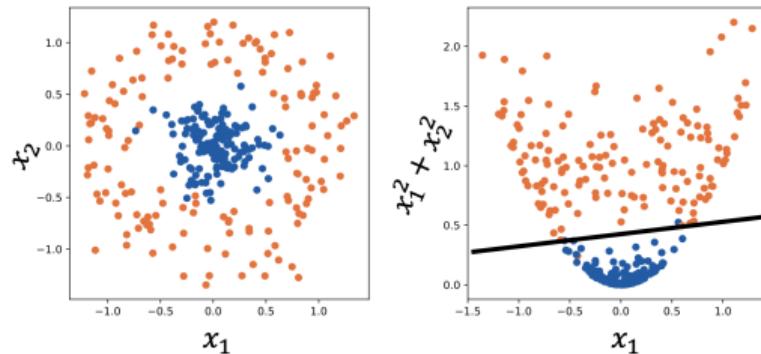
—○ trees

- (Ensembles of) Trees were designed to approximate nonlinearities and are **pretty good** in it + they are **fast and interpretable**
- But they are just “brute-force” algorithms – **don’t infer symmetries** in data by design
- **Ad-hoc, cut-based and piecewise approximations** of data at hand + not differentiable and smooth



Modelling nonlinearities

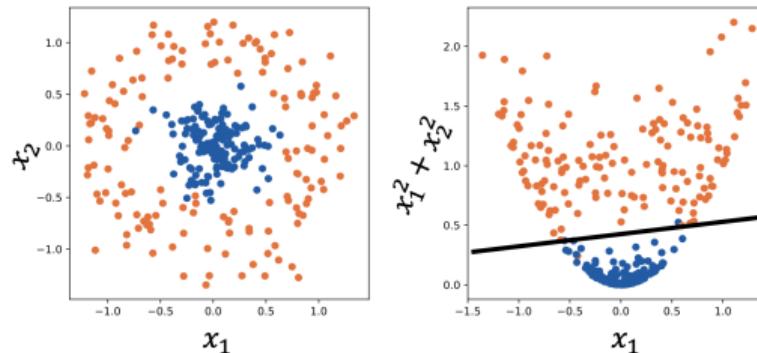
—○ feature engineering



- But sometimes we know *a priori* that there are transformations simplifying the problem → even linear model can do the job
- However, this **feature engineering** is non-trivial, requires domain knowledge and is time-consuming

Modelling nonlinearities

—○ feature engineering



- But sometimes we know *a priori* that there are transformations simplifying the problem → even linear model can do the job
- However, this **feature engineering** is non-trivial, requires domain knowledge and is time-consuming

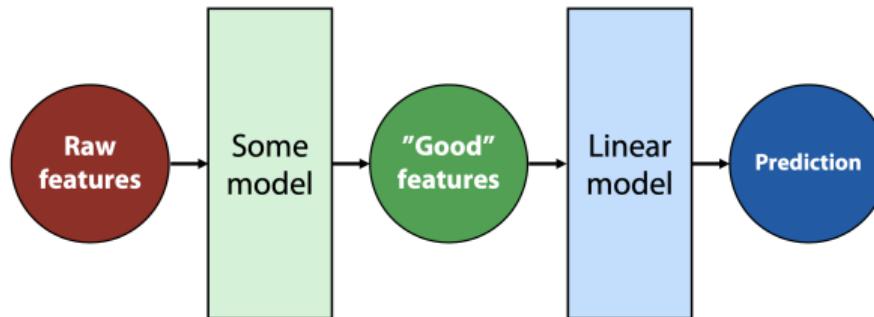
What if we design a model which could **automatically** feature-engineer itself?

Neural Network

Neural Network

—○ automating FE

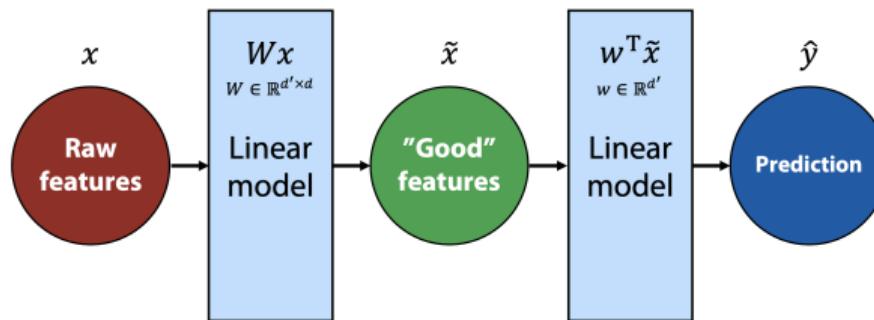
NN illustrations from
ML in HEP 2020



- Let's use a **simple linear model** to solve our supervised problem
- Add a block to a linear model which will automatically **generate new features** for it
- Two blocks would work together as a **single model** ⇒ their parameters are updated simultaneously
- And **automatically**, by e.g. gradient descent (given their differentiability)

Neural Network

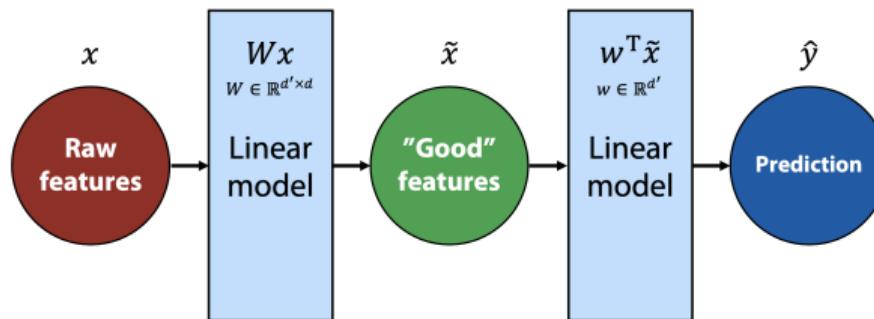
—○ automating FE



- Would a linear model work as a feature generating model?

Neural Network

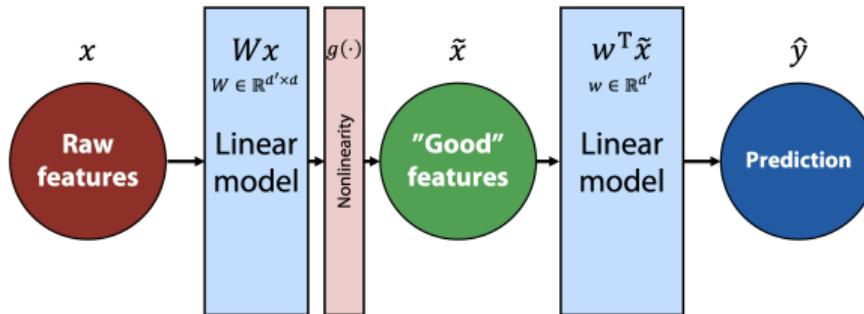
—○ automating FE



- Would a linear model work as a feature generating model? **No**
- $\hat{y} = w^T \tilde{x} = w^T (Wx) = (w^T W)x = w'^T x \Rightarrow$ it is still a linear model
- Input feature space has not changed, only the model weights

Neural Network

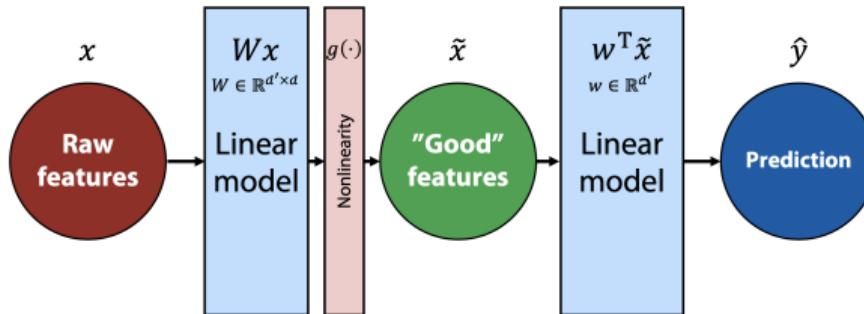
—○ automating FE



- Let's then introduce **nonlinearity** to our model
→ $\hat{y} = w^T \tilde{x} = w^T g(Wx)$,
where $g(\cdot)$ – some nonlinear scalar function (applied elementwise)

Neural Network

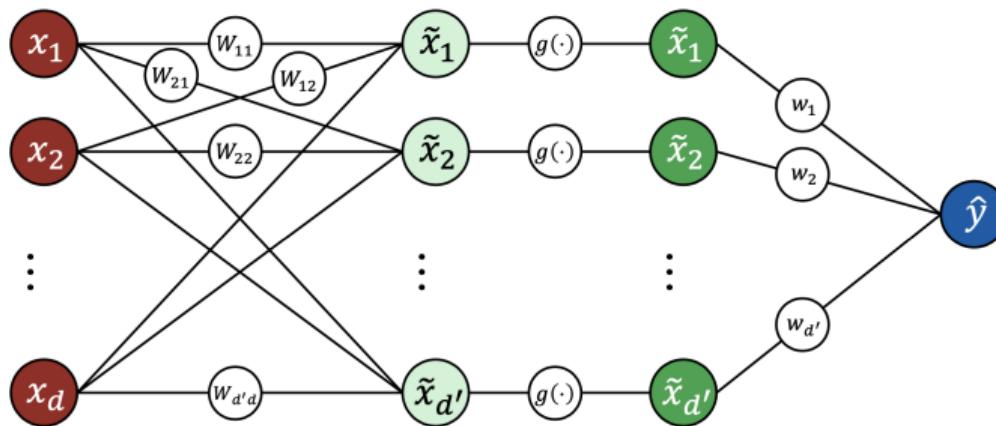
—○ automating FE



- Let's then introduce **nonlinearity** to our model
 - $\hat{y} = w^T \tilde{x} = w^T g(Wx)$,
where $g(\cdot)$ – some nonlinear scalar function (applied elementwise)
 - This is the simplest example of a **neural network**

Neural Network

—○ architecture



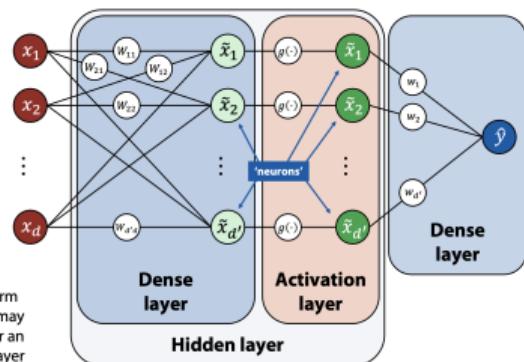
Essentially, NN is just a **composite function** that maps a set of X to a set of Y

$$\hat{y} = w^T \tilde{x} = w^T g(Wx)$$

Neural Network

—○ terminology

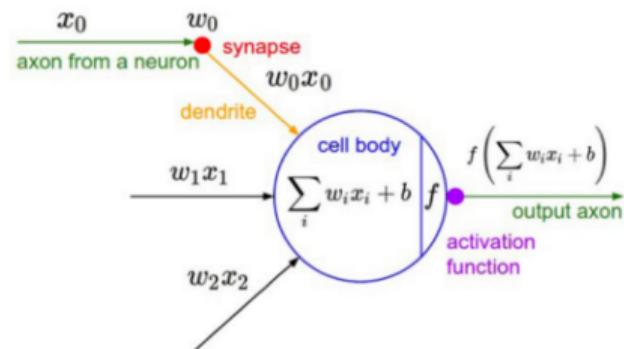
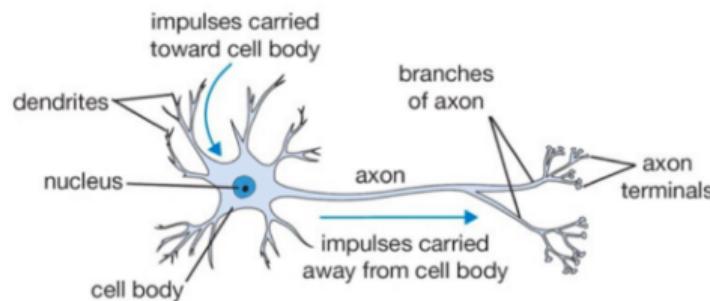
Feed-forward network:



Note: the term "activation" may also stand for an output of a layer

- Brown nodes x_1, x_2, \dots, x_d – features from an **input layer**
- Green nodes $\tilde{x}_1, \tilde{x}_2, \dots, \tilde{x}_{d'}$ – **neurons** from a **hidden layer**
- Blue node \hat{y} – neuron from an **output layer**
- Straight lines (edges) between neurons – **weights** w_{ij}
- $g(\cdot)$ – nonlinear **activation** function, e.g. sigmoid $\sigma(\cdot)$
- **Important:** each neuron has additional **bias** b associated to it and added to other inputs (not illustrated)

Neural Network —○ human brain



Training

Training —— o how to train?

- Since NN fundamentally is a parametrized differentiable model we can optimise the loss function and use **gradient descent** to train it:

$$\omega_{k+1} \leftarrow \omega_k - \eta \cdot \nabla Q(\omega_k)$$

- But **gradients** are hard to derive analytically – writing down all the derivatives is tough and tedious (especially for large NN)
- Note that NN is just a **composite model** \Rightarrow can use **chain rule** for differentiating it

Training —— o chain rule

- Computing derivative of a "base" function is simple \Rightarrow decompose composite function into a set of base ones and differentiate them one by one
- Let's recall the rule:

$$\frac{\partial f(t(x))}{\partial x} = \frac{\partial f(t)}{\partial t} \cdot \frac{\partial t(x)}{\partial x}$$

Exercise: can you compute derivative of sigmoid function by decomposing it into base functions?

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

Training —— o chain rule

- Computing derivative of a "base" function is simple \Rightarrow decompose composite function into a set of base ones and differentiate them one by one
- Let's recall the rule:

$$\frac{\partial f(t(x))}{\partial x} = \frac{\partial f(t)}{\partial t} \cdot \frac{\partial t(x)}{\partial x}$$

Exercise: can you compute derivative of sigmoid function by decomposing it into base functions?

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

$$\sigma(x) = f(g(h(p(x)))), \quad f(z) = \frac{1}{z}, \quad g(z) = 1 + z, \quad h(z) = e^z, \quad p(z) = -z$$

Training —— o chain rule

- Computing derivative of a "base" function is simple \Rightarrow decompose composite function into a set of base ones and differentiate them one by one
- Let's recall the rule:

$$\frac{\partial f(t(x))}{\partial x} = \frac{\partial f(t)}{\partial t} \cdot \frac{\partial t(x)}{\partial x}$$

Exercise: can you compute derivative of sigmoid function by decomposing it into base functions?

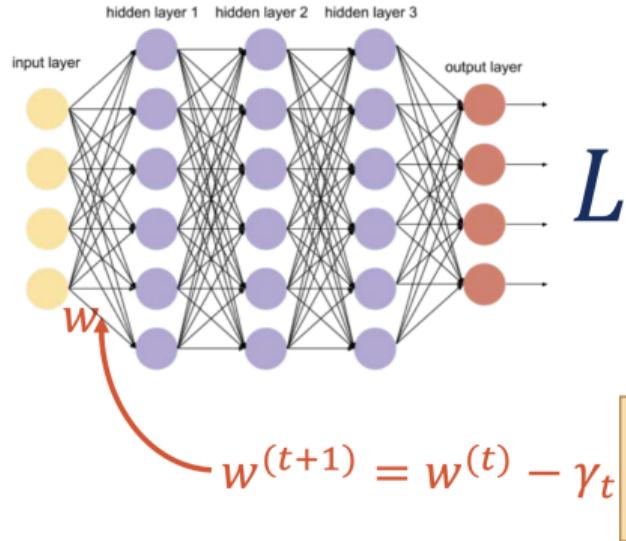
$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

$$\sigma(x) = f(g(h(p(x)))), \quad f(z) = \frac{1}{z}, \quad g(z) = 1 + z, \quad h(z) = e^z, \quad p(z) = -z$$

$$\sigma'(x) = \frac{\partial f}{\partial g} \cdot \frac{\partial g}{\partial h} \cdot \frac{\partial h}{\partial p} \cdot \frac{\partial p}{\partial x} = -\frac{1}{g^2} \cdot 1 \cdot e^p \cdot (-1) = \dots = \sigma(x) \cdot (1 - \sigma(x))$$

Training — o backpropagation

backprop illustrations from [DMIA](#)

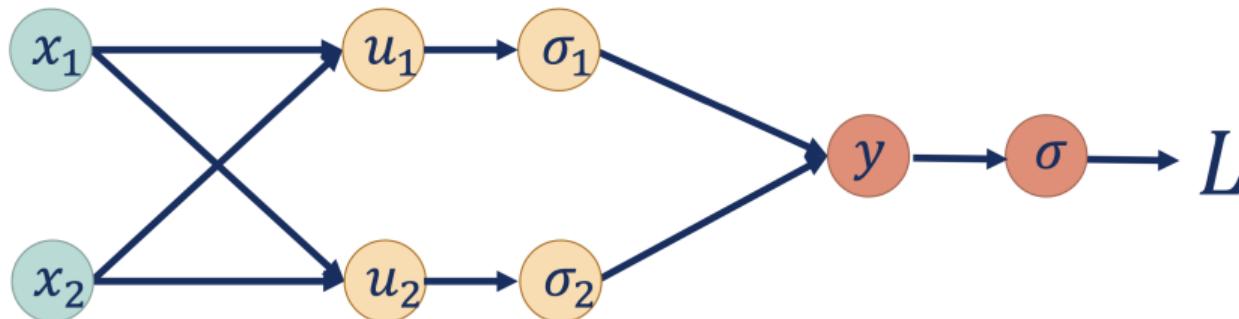


- So how to find partial derivatives of loss function with respect to some weight?

Training

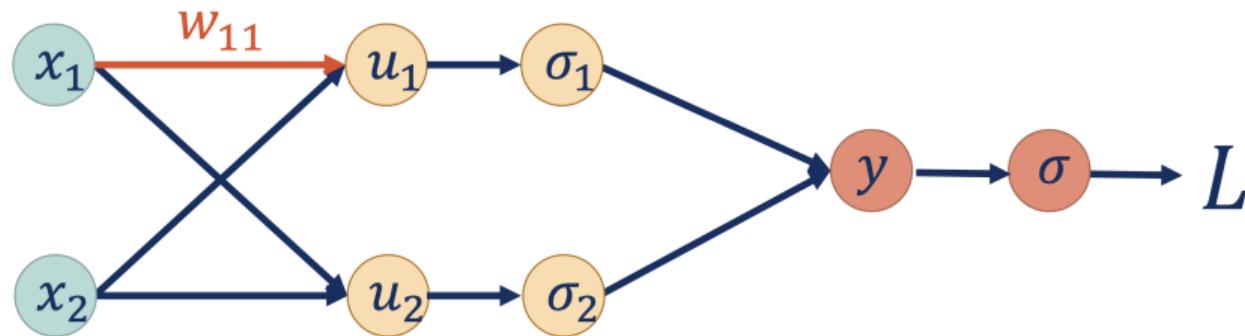
—○ backpropagation

backprop illustrations from [DMIA](#)



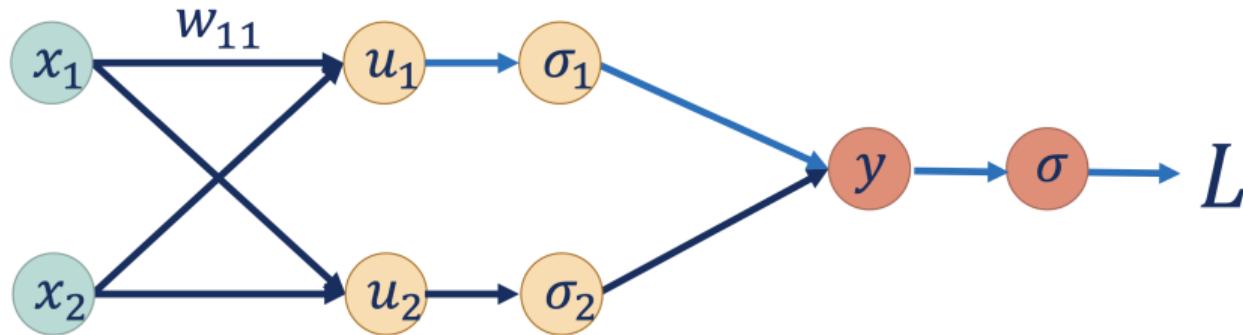
- Let's consider a simplified neural network
- Represent it in the form of a **computational graph**

Training —○ backpropagation



$$w_{11}^{(t+1)} = w_{11}^{(t)} - \gamma_t \frac{\partial L}{\partial w_{11}}$$

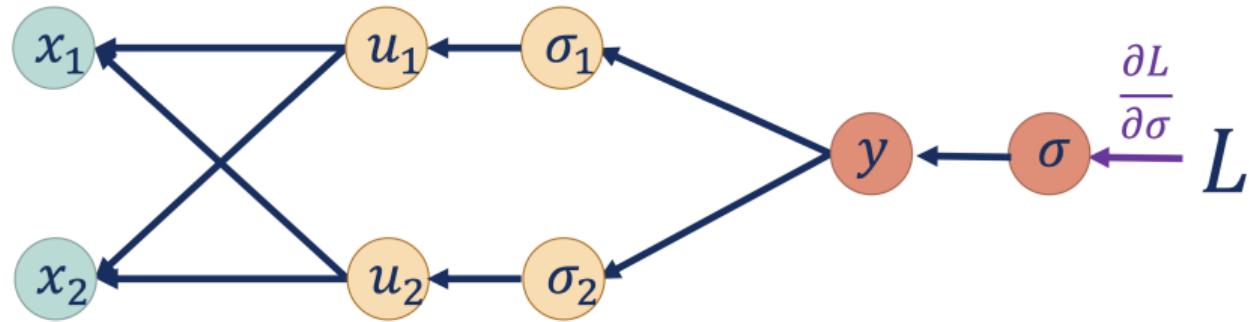
Training —○ backpropagation



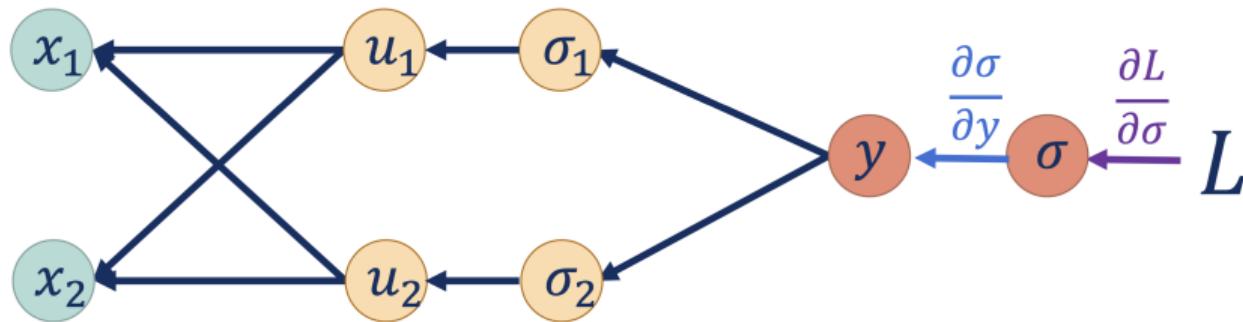
$$w_{11}^{(t+1)} = w_{11}^{(t)} - \gamma_t \frac{\partial L}{\partial w_{11}}$$

$$\frac{\partial L}{\partial w_{11}} = \frac{\partial L}{\partial u_1} \frac{\partial u_1}{\partial w_{11}} = \frac{\partial L}{\partial u_1} x_1$$

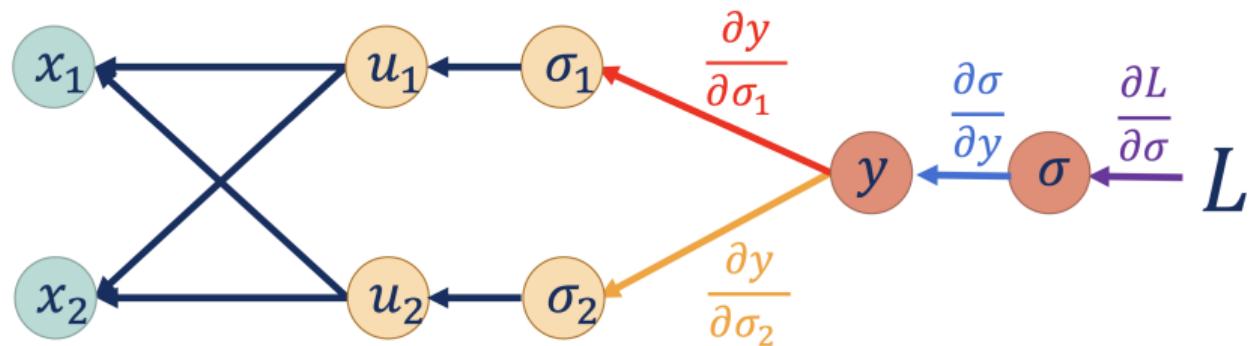
Training —○ backpropagation



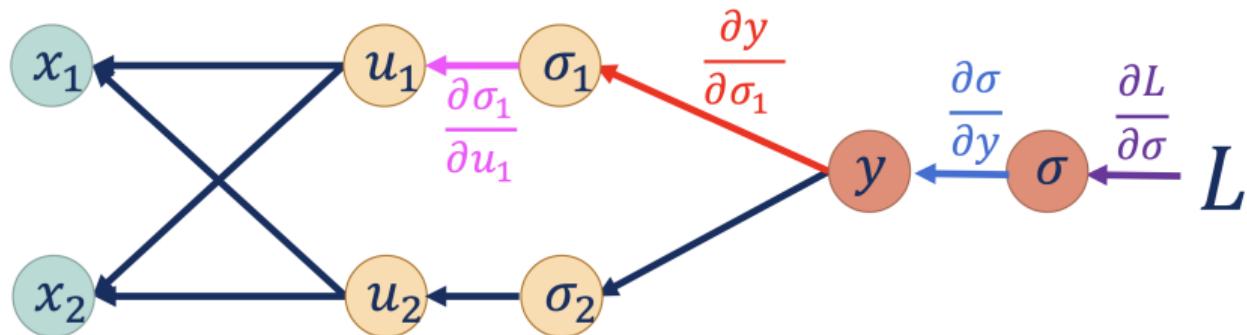
Training —○ backpropagation



Training —○ backpropagation



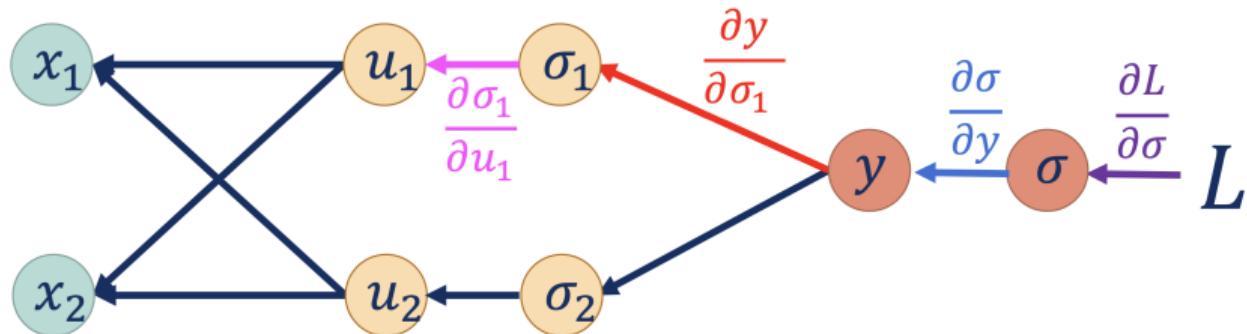
Training —○ backpropagation



$$\frac{\partial L}{\partial u_1} = \frac{\partial L}{\partial \sigma} \frac{\partial \sigma}{\partial y} \frac{\partial y}{\partial \sigma_1} \frac{\partial \sigma_1}{\partial u_1}$$

Training

—○ backpropagation



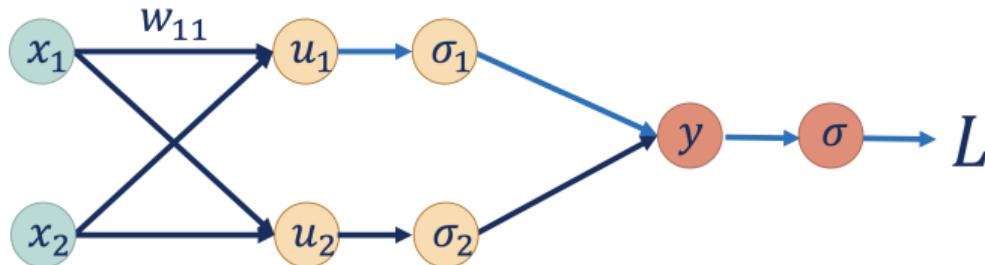
$$\frac{\partial L}{\partial u_1} = \frac{\partial L}{\partial \sigma} \frac{\partial \sigma}{\partial y} \frac{\partial y}{\partial \sigma_1} \frac{\partial \sigma_1}{\partial u_1}$$

- this procedure is called **backpropagation**
- its idea is to **collect derivatives** at each step in the computational graph w/o recalculating them every single time for every weight

Training

—○ wrap it up

train NN in your browser!

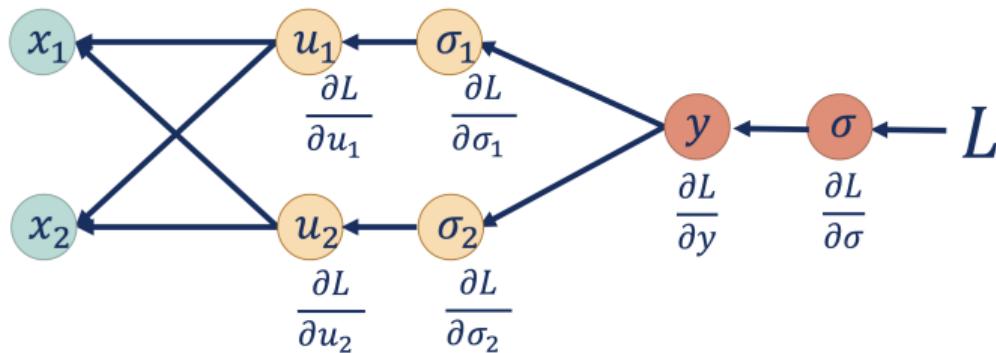


- 1 make **forward pass** through NN to calculate the output of each neuron and value of loss function

Training

—○ wrap it up

[train NN in your browser!](#)

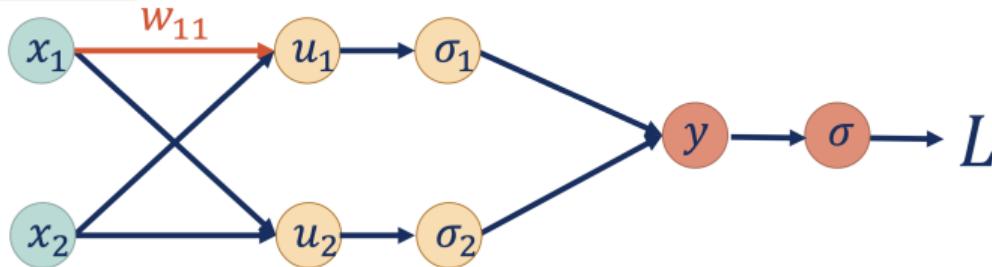


- 1 make **forward pass** through NN to calculate the output of each neuron and value of loss function
- 2 with **backward pass** go back through NN to calculate gradients for all weights

Training

—○ wrap it up

train NN in your browser!



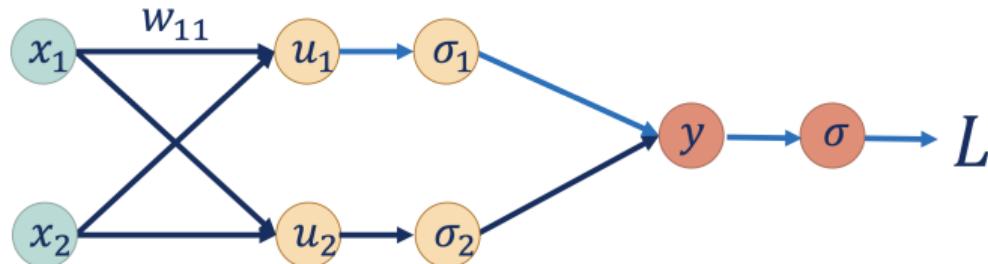
$$w_{11}^{(t+1)} = w_{11}^{(t)} - \gamma_t \frac{\partial L}{\partial w_{11}}$$

- 1 make **forward pass** through NN to calculate the output of each neuron and value of loss function
- 2 with **backward pass** go back through NN to calculate gradients for all weights
- 3 update weights with their gradients

Training

—○ wrap it up

train NN in your browser!



- 1 make **forward pass** through NN to calculate the output of each neuron and value of loss function
- 2 with **backward pass** go back through NN to calculate gradients for all weights
- 3 update weights with their gradients
- 4 repeat until convergence

*one iteration (**epoch**) = forward and backward pass

Summary

Summary

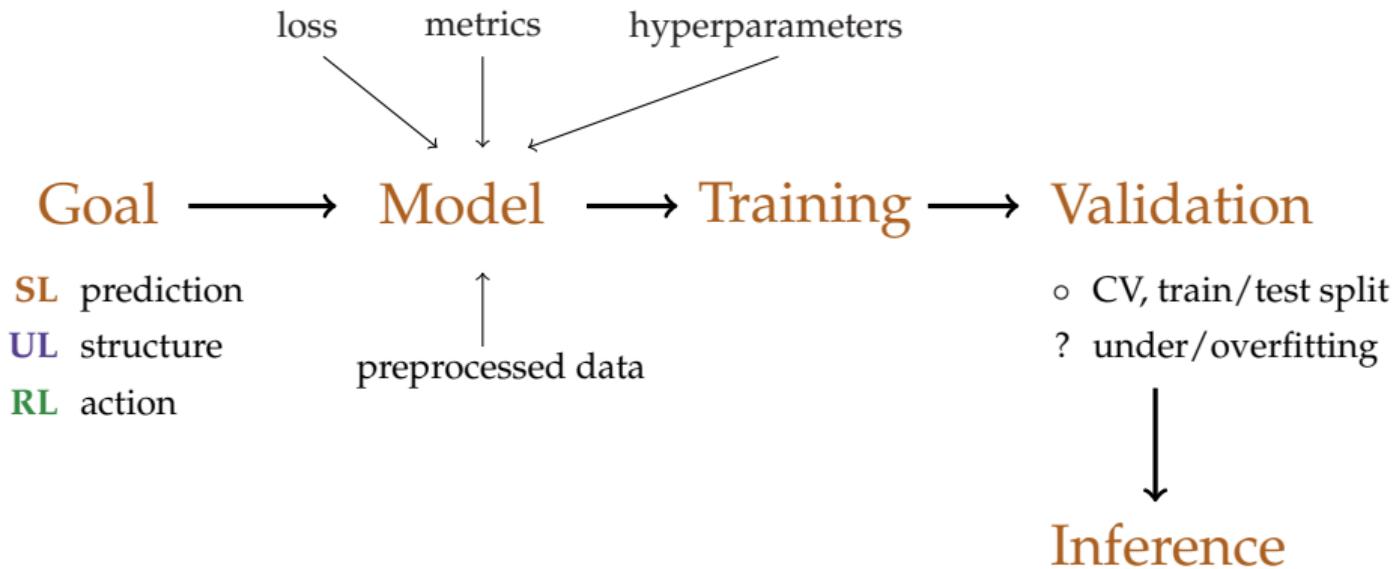
- Science of getting computers to act without being explicitly programmed [Coursera]
- Scientific study of algorithms and statistical models that computer systems use to effectively perform a specific task without using explicit instructions, relying on patterns and inference instead [Wikipedia]
- Science searching for hidden relations in data [A. Volokhova]

In our course, you can learn more about machine learning, related machine learning software and all the necessary materials: Dépôt.

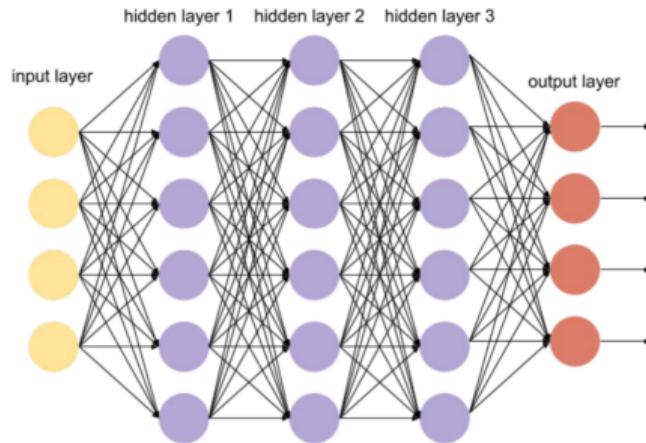
Backup slides

Backup slides

—○ what is ML pipeline?



Backup slides —○ NN formula



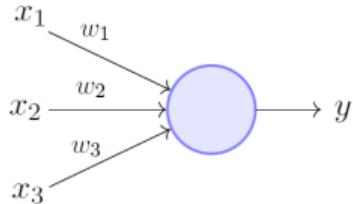
$$h_t = f(W h_{t-1} + b)$$

$$a(x) = f_3(W_3 f_2(W_2 f_1(W_1 f_0(W_0 x + b_0) + b_1) + b_2) + b_3)$$



Backup slides

—○ calculating weights: slide 1



Perceptron Model (Minsky-Papert in 1969)

Source

$$Q(y_{model}, y_{target}) = Q(y(x, w, b), y_{target})$$

$$out = y = \sigma(sum) = \sigma\left(\sum_{i=1}^3 (x_i \cdot w_i) + b\right)$$

$$\frac{\partial Q}{\partial w_i} = \frac{\partial Q}{\partial y} \cdot \frac{\partial y}{\partial \sigma} \cdot \frac{\partial \sigma}{\partial sum} \cdot \frac{\partial sum}{\partial w_i} = \frac{\partial Q}{\partial y} \cdot \textcolor{red}{1} \cdot \frac{\partial \sigma}{\partial sum} \cdot \frac{\partial sum}{\partial w_i}$$

$$\frac{\partial \sigma}{\partial sum} = \sigma(sum) \cdot (1 - \sigma(sum))$$

Backup slides ——○ calculating weights: slide 2

$$Q(y(x, w, b), y_{target})$$

$$Q = (y_{target} - y(x_1, w_1, b))^2$$

$$out = y = (x \cdot w) + b$$

$$\frac{\partial Q}{\partial w_i} = \frac{\partial Q}{\partial y} \cdot \frac{\partial y}{\partial w_i}$$

Input: $x = 1$, $w = 0.1$ and $b = 1$

$y_{target} = 2$ and learning rate: $\eta = 0.1$

Backup slides —○ calculating weights: slide 2

$$\begin{aligned} Q(y(x, w, b), y_{target}) \\ Q = (y_{target} - y(x_1, w_1, b))^2 \\ out = y = (x \cdot w) + b \\ \frac{\partial Q}{\partial w_i} = \frac{\partial Q}{\partial y} \cdot \frac{\partial y}{\partial w_i} \end{aligned}$$

Input: $x = 1$, $w = 0.1$ and $b = 1$

$y_{target} = 2$ and learning rate: $\eta = 0.1$

How to find w_{new} and b_{new} ?

$$\begin{aligned} \frac{\partial Q}{\partial y} &= -2 \cdot (y_{target} - y) \\ \frac{\partial Q}{\partial w} &= \frac{\partial Q}{\partial y} \cdot \frac{\partial y}{\partial w} = -2 \cdot x \cdot (y_{target} - y) \\ \frac{\partial Q}{\partial b} &= \frac{\partial Q}{\partial y} \cdot \frac{\partial y}{\partial b} = -2 \cdot (y_{target} - y) \\ [w_{new}, b_{new}] \Rightarrow \theta_{i+1} &= \theta_i - \eta \cdot \frac{\partial Q(\theta_i)}{\partial \theta_i} \end{aligned}$$

Backup slides

—○ calculating weights: slide 2

$$Q(y(x, w, b), y_{target})$$

$$Q = (y_{target} - y(x_1, w_1, b))^2$$

$$out = y = (x \cdot w) + b$$

$$\frac{\partial Q}{\partial w_i} = \frac{\partial Q}{\partial y} \cdot \frac{\partial y}{\partial w_i}$$

Input: $x = 1$, $w = 0.1$ and $b = 1$

$y_{target} = 2$ and learning rate: $\eta = 0.1$

How to find w_{new} and b_{new} ?

$$\frac{\partial Q}{\partial y} = -2 \cdot (y_{target} - y)$$

$$\frac{\partial Q}{\partial w} = \frac{\partial Q}{\partial y} \cdot \frac{\partial y}{\partial w} = -2 \cdot x \cdot (y_{target} - y)$$

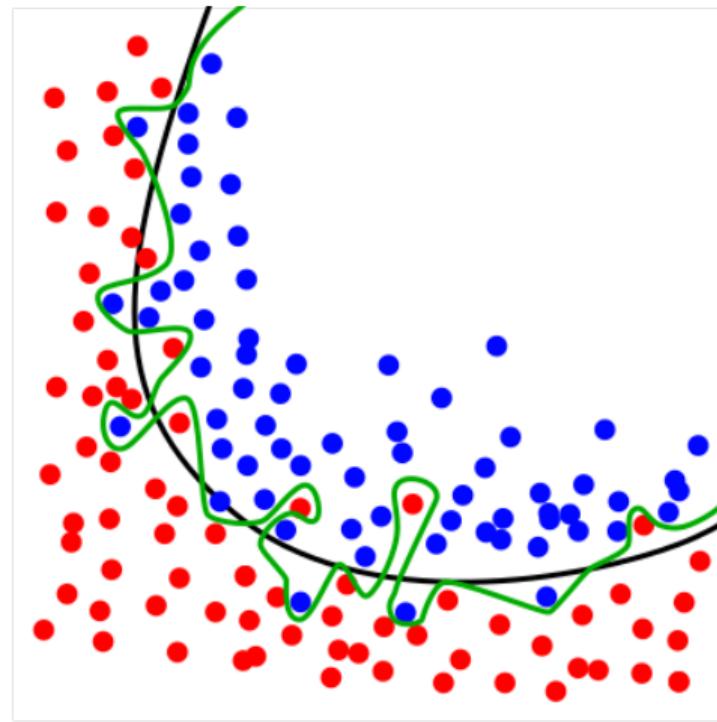
$$\frac{\partial Q}{\partial b} = \frac{\partial Q}{\partial y} \cdot \frac{\partial y}{\partial b} = -2 \cdot (y_{target} - y)$$

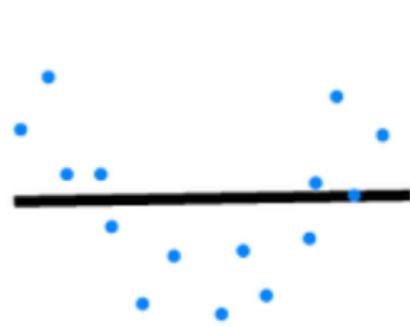
$$[w_{new}, b_{new}] \Rightarrow \theta_{i+1} = \theta_i - \eta \cdot \frac{\partial Q(\theta_i)}{\partial \theta_i}$$

y	Q	$\frac{\partial Q}{\partial y}$	$\frac{\partial Q}{\partial w}$	$\frac{\partial Q}{\partial b}$	w_{new}	b_{new}
1.1	0.81	-1.8	-1.8	-1.8	0.28	1.18

Backup slides

—○ overtraining: slide 1

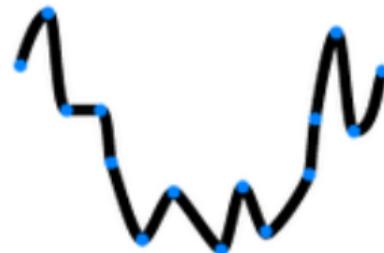




Underfitting



Desired

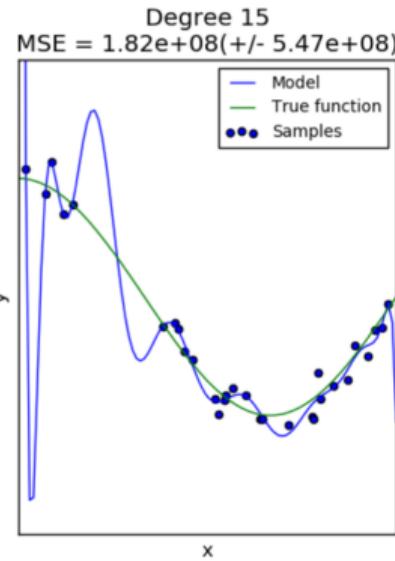
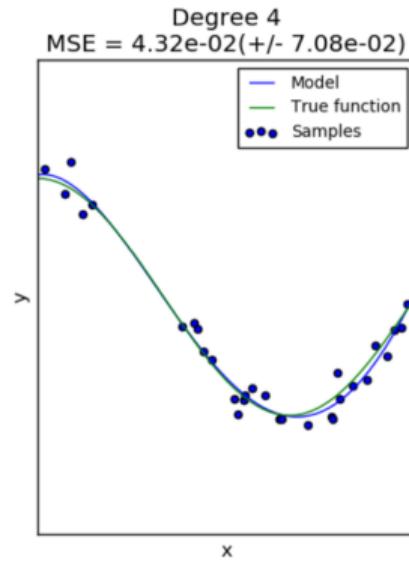
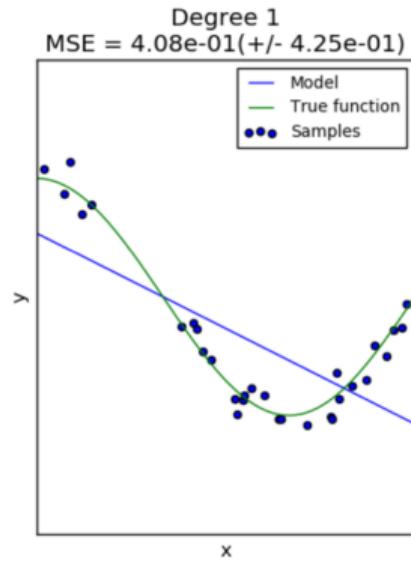


Overfitting

Backup slides

—○ overtraining: slide 3

Overfitting



История про танки



Классификатор: есть танки на снимке или нет

Backup slides

—○ Math and ML

THE MATHS
BEHIND DEEP LEARNING

import keras



Backup slides

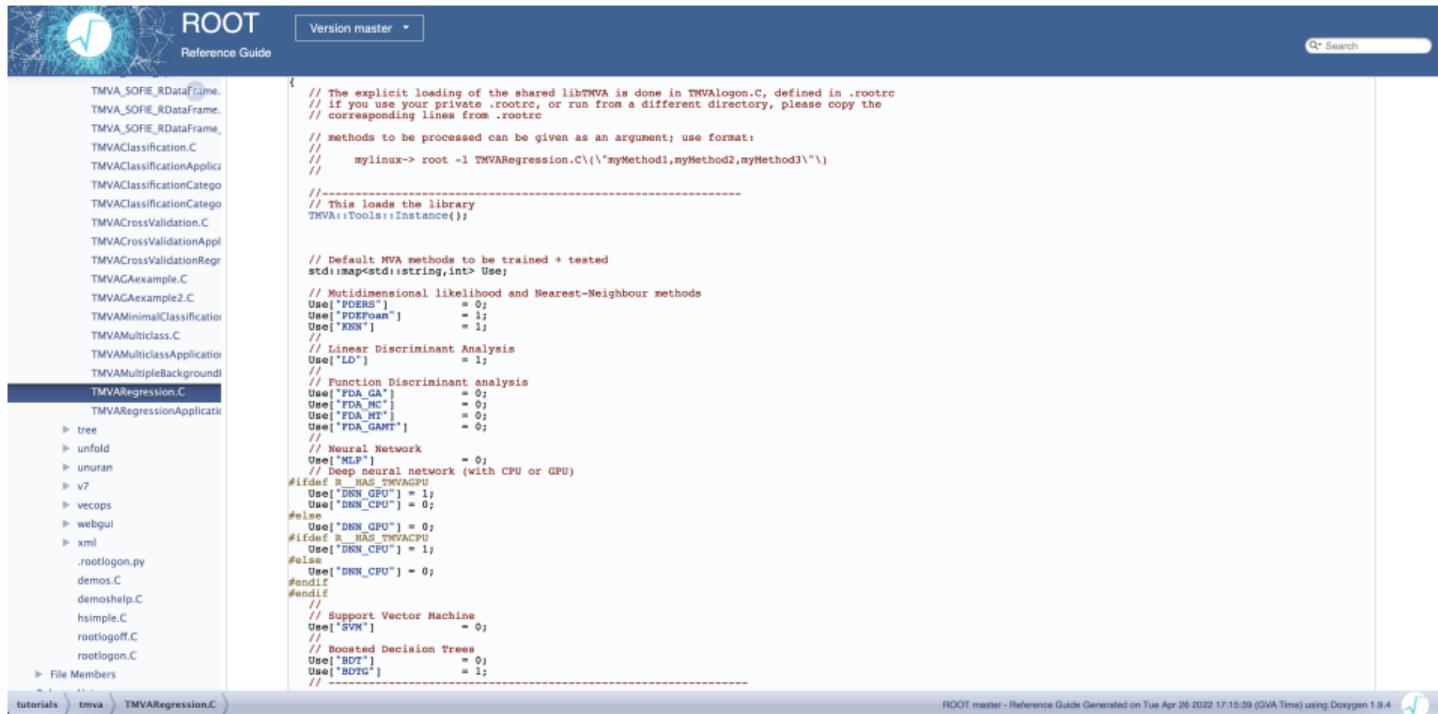
—○ The Toolkit for Multivariate Analysis (TMVA)

TMVA is the ROOT library that provides the interfaces and implementations of the above mentioned machine learning techniques. The package includes:

- Neural networks
- Deep networks
- Boosted/Bagged decision trees
- Multidimensional k-nearest neighbour classifier
- Support Vector Machine (SVM)
- ...

<https://root.cern/manual/tmva/>

Backup slides ——○ TMVA 1

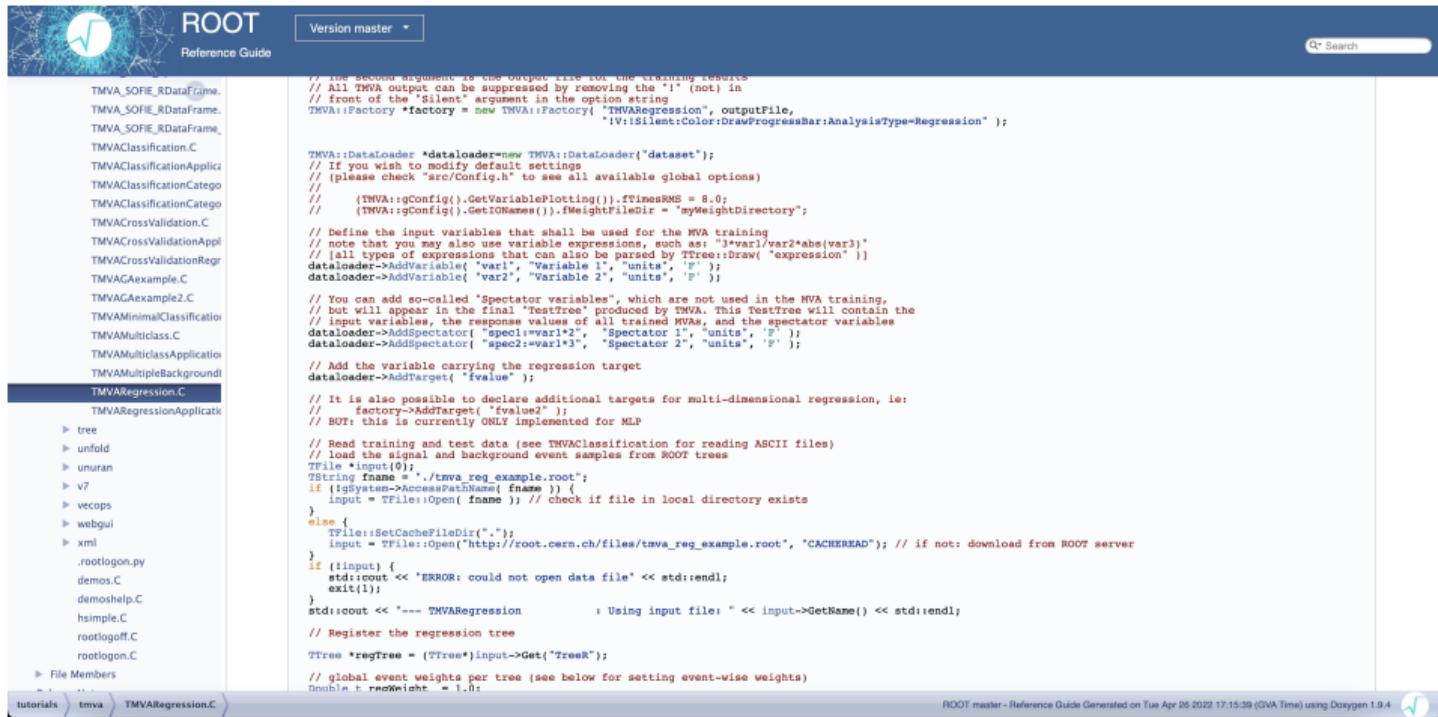


The screenshot shows the ROOT Reference Guide interface with the title "TMVARegression.C". The page content is a C++ code snippet for the TMVARegression class. The code includes comments explaining the explicit loading of shared libraries and the methods to be processed. It defines several member variables and their initial values for different MVA methods like PDEPOM, KNN, LDA, FDA, and RNN. The code also handles neural network configurations based on environment variables and flags. At the bottom, it includes support vector machine and boosted decision tree options.

```
{  
    // The explicit loading of the shared libTMVA is done in TMVAlogon.C, defined in .rootrc  
    // if you use your private .rootrc, or run from a different directory, please copy the  
    // corresponding lines from .rootrc  
  
    // methods to be processed can be given as an argument; use format:  
    // mylinux-> root -l TMVARegression.C('myMethod1,myMethod2,myMethod3')  
  
    //-----  
    // This loads the library  
    TMVA::Tools::Instance();  
  
    // Default MVA methods to be trained + tested  
    std::map<std::string,int> Use;  
  
    // Multidimensional likelihood and Nearest-Neighbour methods  
    Use["PDEPOM"] = 0;  
    Use["PDEPOM"] = 0;  
    Use["KNN"] = 1;  
    //  
    // Linear Discriminant Analysis  
    Use["LDA"] = 1;  
    //  
    // Function Discriminant analysis  
    Use["FDA_GA"] = 0;  
    Use["FDA_MC"] = 0;  
    Use["FDA_MH"] = 0;  
    Use["FDA_GAHT"] = 0;  
    //  
    // Neural Network  
    Use["MLP"] = 0;  
    // Deep neural network (with CPU or GPU)  
    #ifdef __HAS_TMVACPU  
    Use["RNN_GPU"] = 1;  
    Use["RNN_CPU"] = 0;  
    #else  
    Use["RNN_CPU"] = 0;  
    #endif __HAS_TMVACPU  
    Use["RNN_CPU"] = 1;  
    #else  
    Use["RNN_CPU"] = 0;  
    #endif  
    #endif  
    //  
    // Support Vector Machine  
    Use["SVM"] = 0;  
    //  
    // Boosted Decision Trees  
    Use["BDT"] = 0;  
    Use["BDTGC"] = 1;  
    //-----
```

ROOT master - Reference Guide Generated on Tue Apr 26 2022 17:15:39 (GVA Time) using Doxygen 1.9.4

Backup slides —○ TMVA 2



The screenshot shows a web-based documentation interface for the ROOT framework. At the top, there's a logo and the word "ROOT". Below it, a search bar and a "Reference Guide" link. The main content area displays the source code for `TMVARegression.C`. The code is a C++ script for training a TMVA regression model. It includes comments explaining the steps: reading training and test data from ASCII files, loading signal and background event samples, creating a `TTree`, and registering the regression tree. It also handles global event weights per tree. The code is annotated with various `//` and `/* */` comments throughout.

```
// The second argument is the output file for the training results
// All TMVA output can be suppressed by removing the '!' (not) in
// front of the "Silent" argument in the option string
TMAVIFactory *factory = new TMAVIFactory("TMVARegression", outputFile,
                                         "!V!Silent!Color!DrawProgressBar:AnalysisType=Regression");

TMVA::DataLoader *dataLoader=new TMVA::DataLoader("dataset");
// If you wish to modify default settings
// (please check "src/Config.h" to see all available global options)
// (TMVA::gConfig().GetVariablePlotting(), fTMVAeSMS = 8.0;
// (TMVA::gConfig().GetIOName()), fWeightfileDir = "myWeightDirectory";

// Define the input variables that shall be used for the MVA training
// note that you may also use variable expressions, such as: "1+var1/var2*abs(var3)"
// [all types of expressions that can also be parsed by TTree::Draw("expression")]
dataLoader->AddVariable("var1", "Variable 1", "units", "[F]");
dataLoader->AddVariable("var2", "Variable 2", "units", "[F]");

// You can add so-called 'spectator variables', which are not used in the MVA training,
// but will appear in the final 'TestTree' produced by TMVA. This TestTree will contain the
// input variables, the response values of all trained MVAs, and the spectator variables
dataLoader->AddSpectator("spec1==var1^2", "Spectator 1", "units", "[F]");
dataLoader->AddSpectator("spec2==var1^3", "Spectator 2", "units", "[F]");

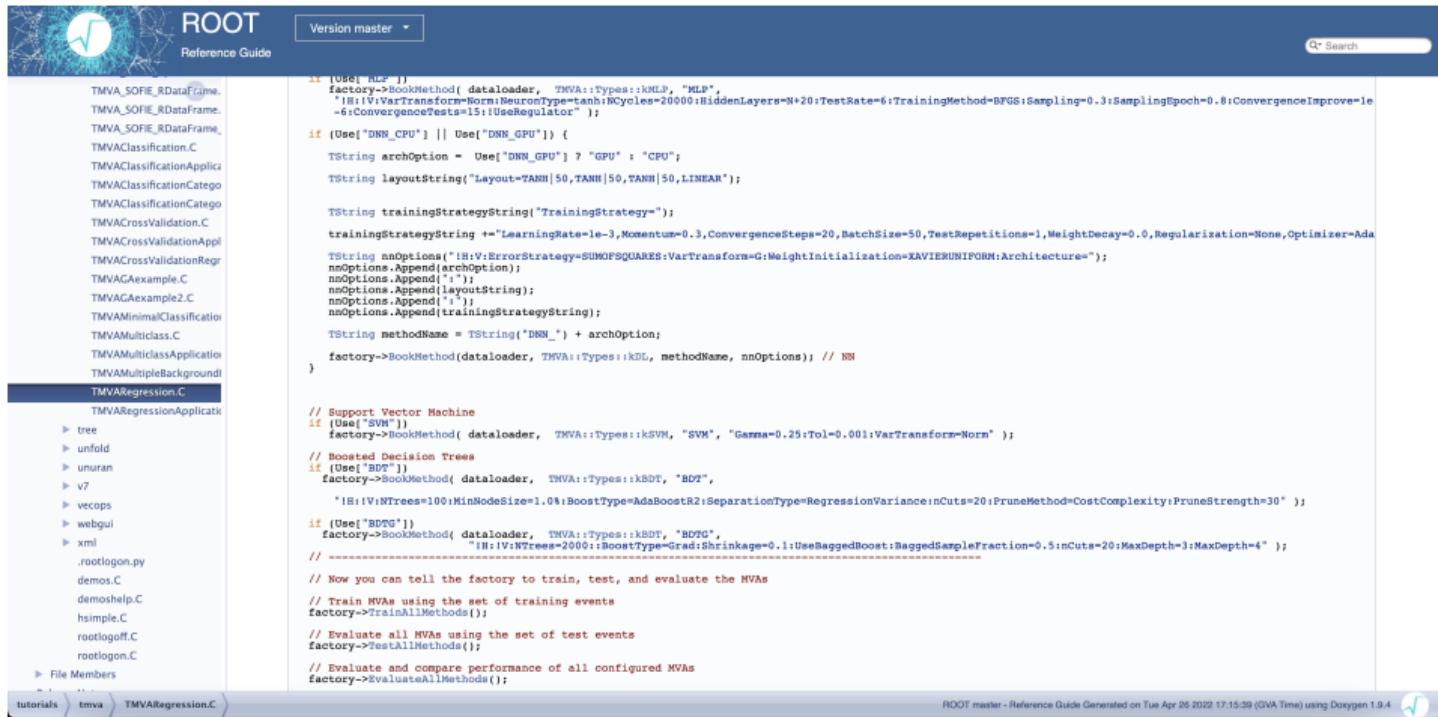
// Add the variable carrying the regression target
dataLoader->AddTarget("fvalue");

// It is also possible to declare additional targets for multi-dimensional regression, ie:
// factory->AddTarget("fvalue2");
// BUT: this is currently ONLY implemented for MLP

// Read training and test data (see TMVAClassification for reading ASCII files)
// load the signal and background event samples from ROOT trees
TFile *input(0);
 TString fname = ".tmva_reg_example.root";
 if (!gSystem->AccessPathName( fname )) {
   input = TFile::Open( fname ); // check if file in local directory exists
 }
 else {
   TFile::SetCacheFileDir(".");
   input = TFile::Open("http://root.cern.ch/files/tmva_reg_example.root", "CACHEREAD"); // if not: download from ROOT server
 }
 if (!input) {
   std::cout << "ERROR: could not open data file" << std::endl;
   exit(1);
 }
 std::cout << "---- TMVARegression           : Using input file: " << input->GetName() << std::endl;

// Register the regression tree
TTree *regTree = (TTree*)input->Get("TreeR");
// global event weights per tree (see below for setting event-wise weights)
Double_t recWeight = 1.0;
```

Backup slides —○ TMVA 3



The screenshot shows the ROOT Reference Guide interface with the title "TMVARegression.C". The left sidebar lists various TMVA components and files, including TMVA_SOFIE_RDataFrame, TMVA_SOFIE_RDataFrame, TMVA_SOFIE_RDataFrame, TMVAClassification.C, TMVAClassificationApplica, TMVAClassificationCatego, TMVAClassificationCatego, TMVACrossValidation.C, TMVACrossValidationAppl, TMVACrossValidationRegr, TMVAGAexample.C, TMVAGAexample2.C, TMVAMinimalClassification, TMVAMulticlass.C, TMVAMulticlassApplicatio, TMVAMultipleBackground, TMVARegression.C, TMVARegressionApplicat, tree, unfold, unuran, v7, vecops, webgui, xml, .rootlogon.py, demos.C, demoshelp.C, hsimple.C, rootlogoff.C, rootlogon.C, File Members, ..., tutorials, tmva, TMVARegression.C.

The main content area displays the TMVARegression.C code:

```
1 // [USE_RLF]
2 factory->BookMethod( dataloader, TMVA::Types::kMLP, "MLP",
3   "IH:IV:VarTransform=Norm:NeuronType=tanh:NCycles=2000:HiddenLayers=N+20:TestRate=6:TrainingMethod=BFGS:Sampling=0.3:SamplingEpoch=0.8:ConvergenceImprove=le
4   -6:ConvergenceTests=5:UseRegulator" );
5
6 if (Use["DNN_CPU"] || Use["DNN_GPU"]) {
7   TString archOption = Use["DNN_GPU"] ? "GPU" : "CPU";
8   TString layoutString("Layout=TANN|50,TANN|50,TANN|50,LINEAR");
9
10  TString trainingStrategyString("TrainingStrategy");
11  trainingStrategyString += "LearningRate=1e-3,Momentum=0.3,ConvergenceSteps=20,BatchSize=50,TestRepetitions=1,WeightDecay=0.0,Regularization=None,Optimizer=Ada
12  nmOptions.Append("IH:V:ErrorStrategy=SUMOFSQUARES:VarTransform=G:WeightInitialization=XAVIERUNIFORM:Architecture=");
13  nmOptions.Append(archOption);
14  nmOptions.Append("I+");
15  nmOptions.Append("V+");
16  nmOptions.Append("G+");
17  nmOptions.Append("A+");
18  nmOptions.Append("R+");
19  nmOptions.Append("T+");
20  nmOptions.Append("L+");
21  nmOptions.Append(trainingStrategyString);
22
23  TString methodName = TString("DNN_") + archOption;
24
25  factory->BookMethod(dataloader, TMVA::Types::kDL, methodName, nmOptions); // NN
}
26
27 // Support Vector Machine
28 if (Use["SVM"])
29   factory->BookMethod( dataloader, TMVA::Types::kSVM, "SVM", "Gamma=0.25:Tol=0.001:VarTransform=Norm" );
30
31 // Boosted Decision Trees
32 if (Use["BDT"])
33   factory->BookMethod( dataloader, TMVA::Types::kBDT, "BDT",
34     "IH:IV:NTrees=100:MinNodeSize=1.0:BoostType=AdaBoostR2:SeparationType=RegressionVariance:nCuts=20:PruneMethod=CostComplexity:PruneStrength=30" );
35
36 if (Use["BDTG"])
37   factory->BookMethod( dataloader, TMVA::Types::kBDT, "BDTG",
38     "IH:IV:NTrees=2000:BoostType=GradShrinkage=0.1:UseBaggedBoost:BaggedSampleFraction=0.5:nCuts=20:MaxDepth=3:MaxDepth=4" );
39
40 // Now you can tell the factory to train, test, and evaluate the MVAs
41
42 // Train MVAs using the set of training events
43 factory->TrainAllMethods();
44
45 // Evaluate all MVAs using the set of test events
46 factory->TestAllMethods();
47
48 // Evaluate and compare performance of all configured MVAs
49 factory->EvaluateAllMethods();
```

At the bottom right, there are navigation icons for back, forward, search, and other document functions.

Backup slides ——○ XGBoost

```
import xgboost as xgb
xgb_model = xgb.XGBClassifier(learning_rate=0.001,
                               max_depth = 1,
                               n_estimators = 100,
                               scale_pos_weight=5)
xgb_model.fit(x_train, y_train)

print("Accuracy on training set: {:.3f}".format(xgb_model.score(x_train, y_train)))
print("Accuracy on validation set: {:.3f}".format(xgb_model.score(x_val, y_val)))

Accuracy on training set: 0.415
Accuracy on validation set: 0.414
```

Predictions

```
In [14]: xgb_predict=xgb_model.predict(x_val)
xgb_predict

Out[14]: array([1, 1, 1, ..., 1, 1, 1])
```

Backup slides ——o Keras 1

```
In [3]: # Function to create model, required for KerasClassifier
def create_model():
    # default values
    activation='relu' # or linear
    dropout_rate=0.0 # or 0.2
    init_mode='uniform'
    weight_constraint=0 # or 4
    optimizer='adam' # or SGD
    lr = 0.01
    momemntum=0
    # create model
    model = Sequential()
    model.add(Dense(8,
                   input_dim=input_dim, kernel_initializer=init_mode,
                   activation=activation,
                   kernel_constraint=maxnorm(weight_constraint)))
    model.add(Dropout(dropout_rate))
    model.add(Dense(1, kernel_initializer=init_mode, activation='sigmoid'))
    # Compile model
    model.compile(loss='binary_crossentropy',
                  optimizer=optimizer,
                  metrics=['accuracy'])
    return model
```

Backup slides ——o Keras 2

```
from keras.models import Sequential
from keras.layers.core import Dense
from keras.optimizers import SGD

np.random.seed(1)

model = Sequential()
model.add(Dense(input_dim=X_train.shape[1],
                output_dim=50,
                init='uniform',
                activation='tanh'))

model.add(Dense(input_dim=50,
                output_dim=50,
                init='uniform',
                activation='tanh'))

model.add(Dense(input_dim=50,
                output_dim=y_train_OneHotEncoding.shape[1],
                init='uniform',
                activation='softmax'))

sgd = SGD(lr=0.001, decay=1e-7, momentum=.9)
model.compile(loss='categorical_crossentropy', optimizer=sgd)

model.fit(X_train, y_train_OneHotEncoding, nb_epoch=50,
           batch_size=300, verbose=1,
           validation_split=0.1, show_accuracy=True)

/usr/local/lib/python2.7/dist-packages/keras/models.py:610: UserWarning
  warning: The "show_accuracy" argument is deprecated, instead you shoul
d pass the "accuracy" metric to the model at compile time:
`model.compile(optimizer, loss, metrics=["accuracy"])'
  warnings.warn('The "show_accuracy" argument is deprecated, '
```

Backup slides ——o Keras 3

```
11 import pandas as pd
12 import numpy as np
13 from sklearn import preprocessing
14 from sklearn.model_selection import train_test_split
15 import keras
16 from keras.models import Sequential
17 from keras.layers import Dense
18 from keras.wrappers.scikit_learn import KerasRegressor
19
20 data = pd.read_csv("USA_Housing_prel.csv")
21 min_max = preprocessing.MinMaxScaler()
22
23 X = data.iloc[:,0:5]
24 Y = data.iloc[:,5]
25
26 X_train,Y_train,X_test,Y_test = train_test_split(X,Y,test_size=0.25,random_state=4)
27
28 def baseline_model():
29     model = Sequential()
30     model.add(Dense(5,input_shape=(5,),kernel_initializer='normal',activation='relu'))
31     model.add(Dense(1,kernel_initializer='normal'))
32     model.compile(loss='mean_squared_error', optimizer='adam')
33     return model
34
35 estimator = KerasRegressor(build_fn=baseline_model,epochs=5,batch_size=32)
36
37 estimator.fit(X_train,Y_train)
```