Machine Learning: short introduction (trees and NN)

Sergey Korpachev[1, 2] on behalf of Dépôt

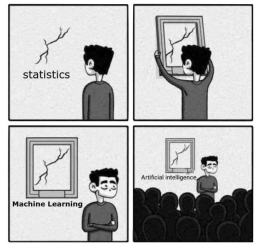
[1]Moscow Institute of Physics and Technology
[2]Lebedev Physical Institute of the Russian Academy of Sciences

Outline

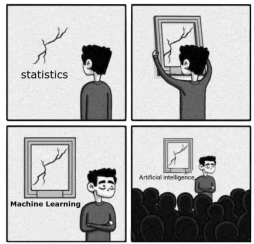
- Machine learning (ML) - what is it?
- Data
- ML pipeline
- Linear model
- Decision tree
- Neural network

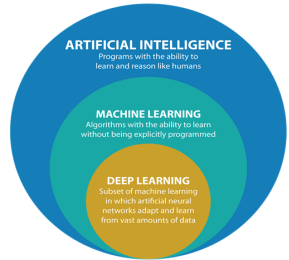
Machine learning

Machine learning — what is it?

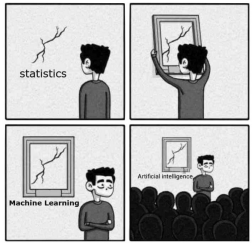


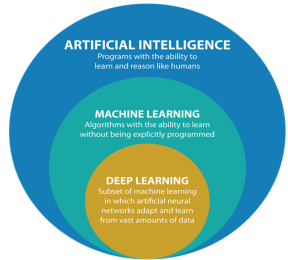
Machine learning — what is it?

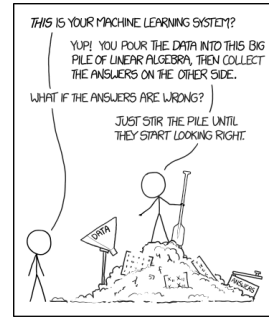




Machine learning — what is it?







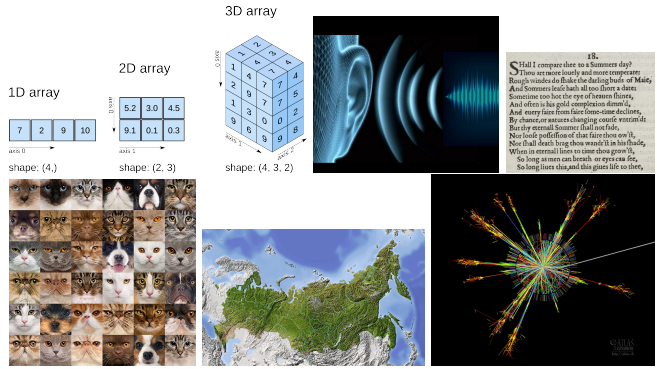
Data

Data

what is data?

Anything can be data:

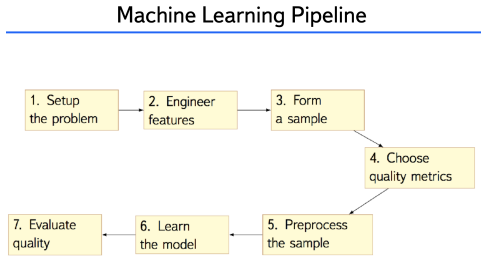
1. Numbers
2. Text
3. Images
4. Sound
5. Geomap
6. Particle collisions
7. Knowledge
8. You name it



ML pipeline

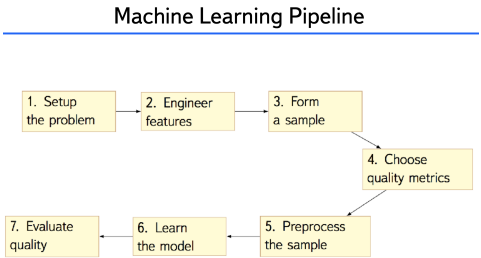
ML pipeline

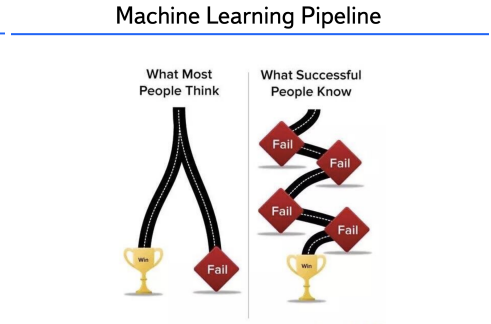
what is ML pipeline?



ML pipeline

what is ML pipeline?

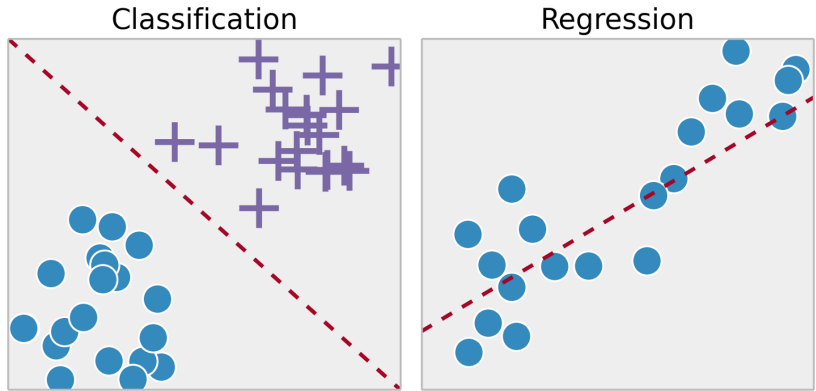




Linear model

Linear model

classification and regression



Linear model

regression

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

- Minimization problem:
  - $\mathcal{L}(\boldsymbol{\theta}) = \big(\langle \boldsymbol{x}, \boldsymbol{\theta} \rangle - y\big)^2$
  - $Q(\boldsymbol{\theta}) = \frac{1}{N} \sum_{i=1}^{N} \big(\langle \boldsymbol{x_i}, \boldsymbol{\theta} \rangle - y_i\big)^2$
- Then we need to minimize $Q(\boldsymbol{\theta})$ by varying $\boldsymbol{\theta}$:
  - $\hat{a} = \arg\min_{\boldsymbol{\theta} \in \Theta} Q(\boldsymbol{\theta})$

Linear model

classification

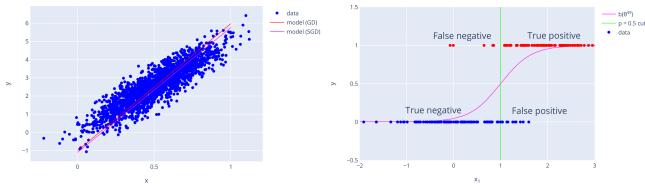
- $Y = \{-1, +1\}$
- N objects with K real features: $D = \mathbb{R}^K$
- $a(\boldsymbol{x}, \boldsymbol{\theta}) = \theta_0 + \sum_{j=1}^{K} f_j(\boldsymbol{x}) \cdot \theta_j$
- Extend and reassign:
  $[1, f_1(\boldsymbol{x}), \ldots, f_K(\boldsymbol{x})] \equiv [1, x_1, \ldots, x_K] \equiv \boldsymbol{x}$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

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

Decision tree

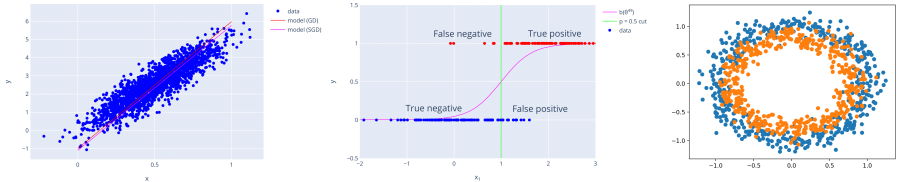
Decision tree

motivation



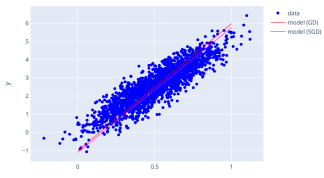
- In the previous part we explored linear models:
  - ➔ nicely describe linear correlations
  - ➔ fast and easy to train

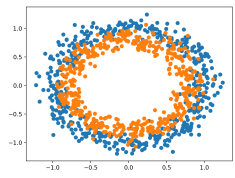
Decision tree — motivation



- 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





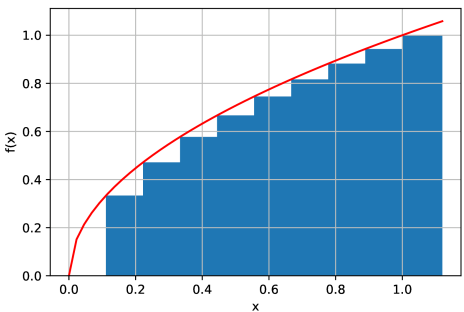
- In the previous part we explored linear models:
  - ➔ nicely describe linear correlations
  - ➔ fast and easy to train
- However, they are poor in modelling non-linearities
- Additional heuristics might be applied (e.g. polynomial features) but are ad-hoc

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

Decision tree

motivation

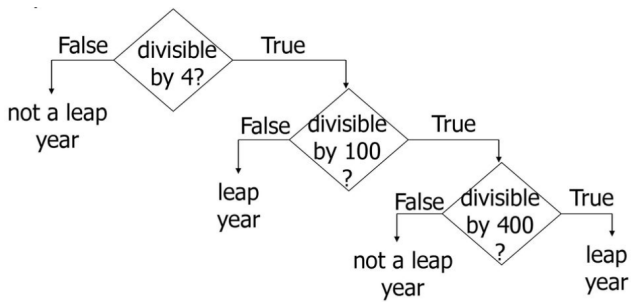
- What is the simplest way to approximate a function?
- Using **piecewise linear function**



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

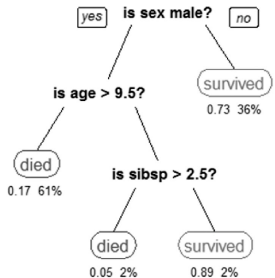
Decision tree — intuition

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



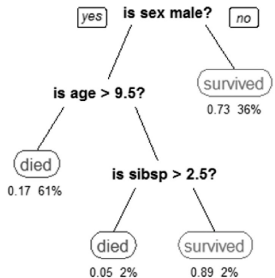
Decision tree — intuition

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



Decision tree — intuition

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



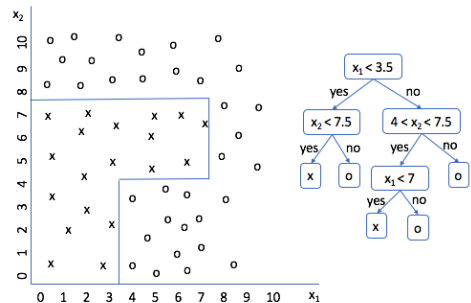
Can we mathematically formulate this?

Decision tree — algorithm

Algorithm 1: Decision tree

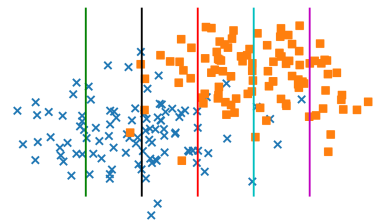
1: Initialize: hyperparameters, leaf set = $\{X_{\text{train}}\}$

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

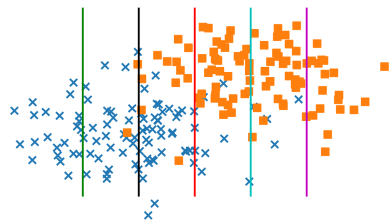
- Suppose there is a classification problem
- So we want to separate one class from the other by constructing **decision boundary**
- And using decision tree algorithm described earlier
- To do that we need to start splitting on features
- Which **split** is the best?



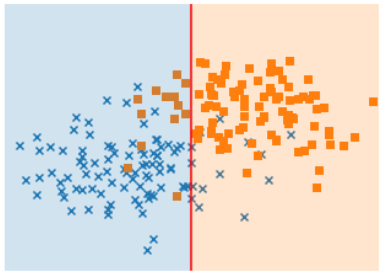
Decision tree — split criteria

- 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

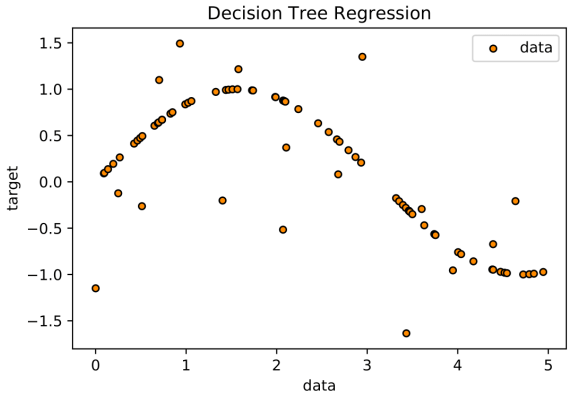


Decision tree — split criteria



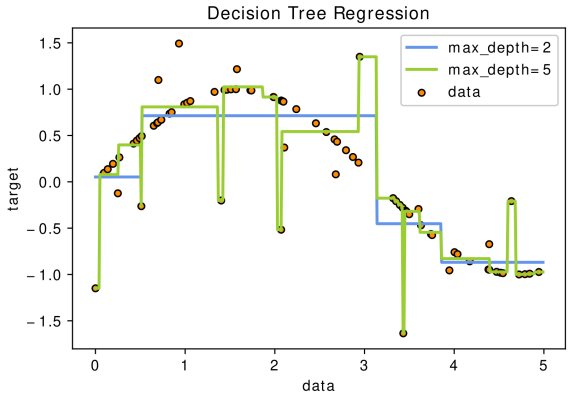
The ”best” split is a central one because it introduces **purity** in the best way. And we have some functions to measure the purity!

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

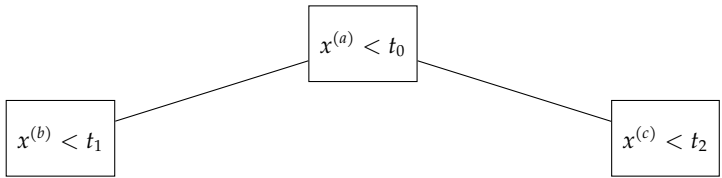
Decision tree — growing

What about tree construction?

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

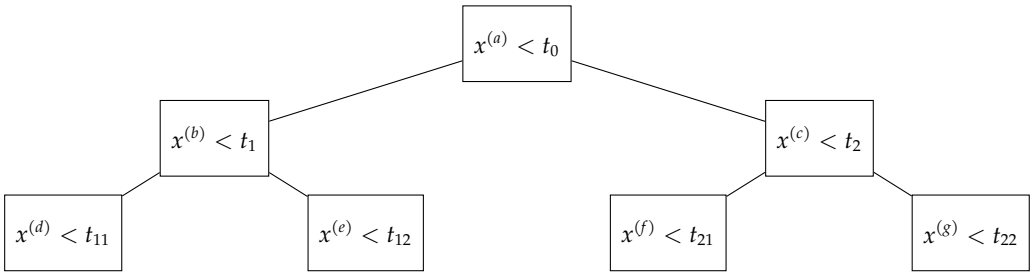
Decision tree — growing

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



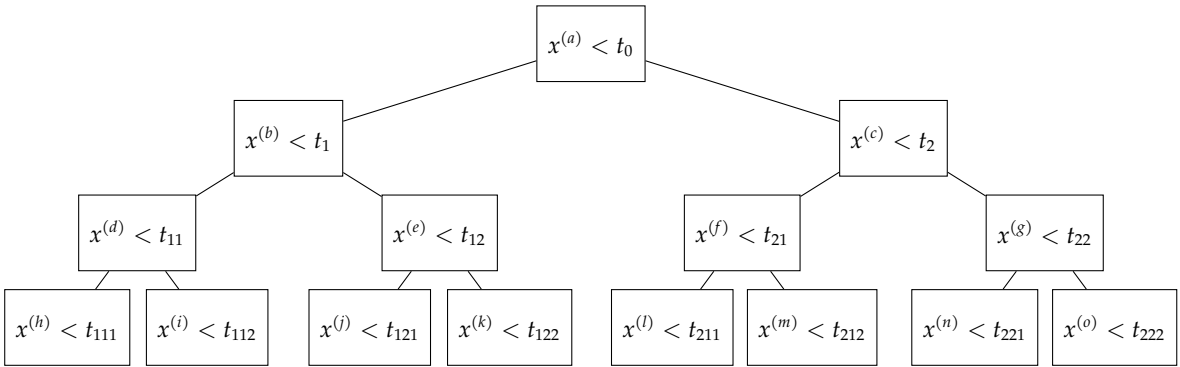
Decision tree — growing

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



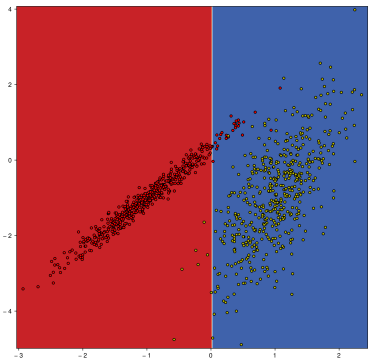
Decision tree — growing

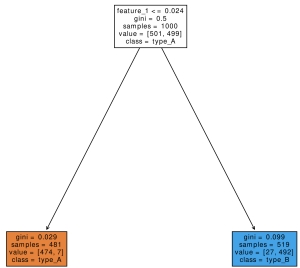
What about tree construction? Do it recursively by **greedy** algorithm



Decision tree — growing

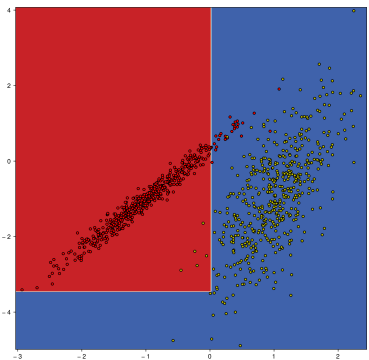
check out also these visuals

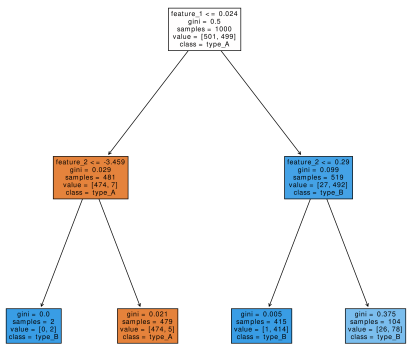




Decision tree — growing

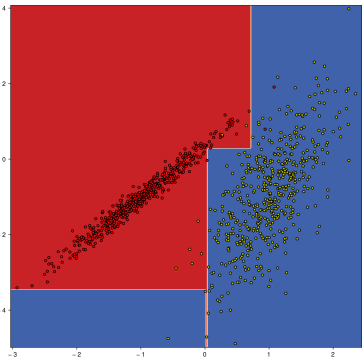
check out also these visuals

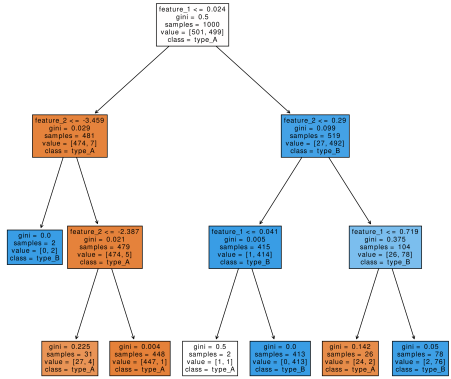




Decision tree — growing

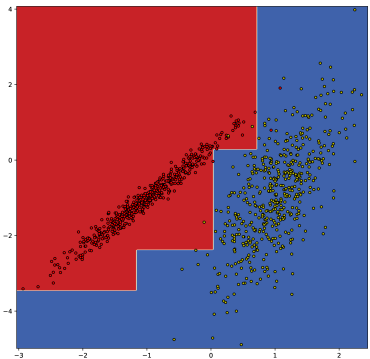
check out also these visuals

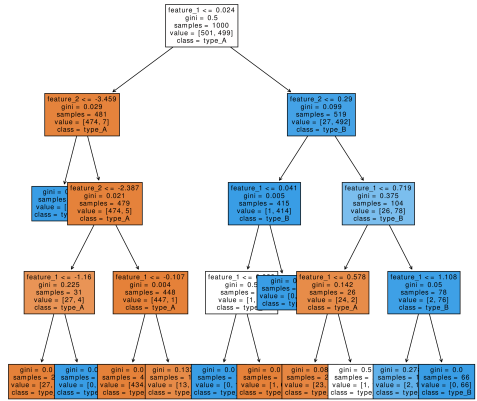




Decision tree — growing

check out also these visuals

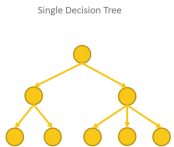


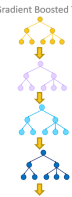


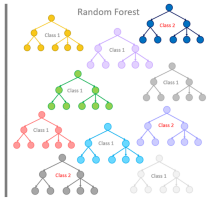
Random forest and gradient boosting

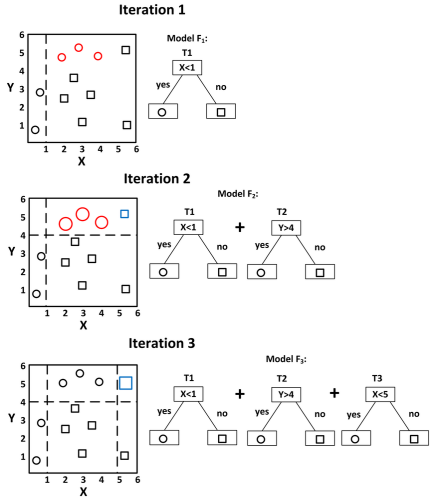
Random forest and gradient boosting

what's next after decision trees?





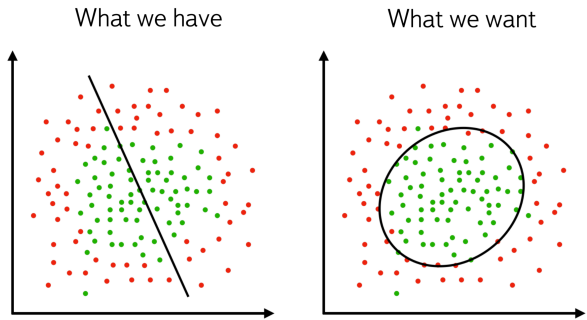




Modelling nonlinearities

Modelling nonlinearities

linear models

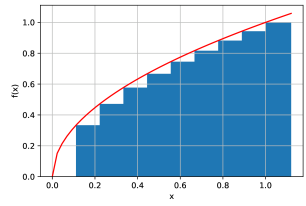


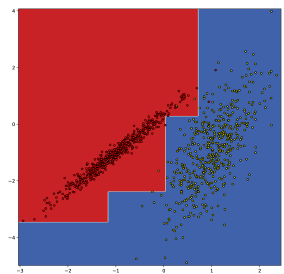
Linear models can't simply describe complex nonlinear data

Modelling nonlinearities

trees

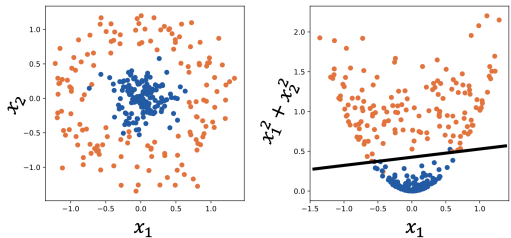
- (Ensembles of) Trees were designed to approximate nonlinearities and are **pretty good** in it + they are **fast and interpretable**
- But they are just ”brute-force” algorithms – **don’t infer symmetries** in data by design
- **Ad-hoc, cut-based and piecewise approximations** of data at hand + not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





Modelling nonlinearities

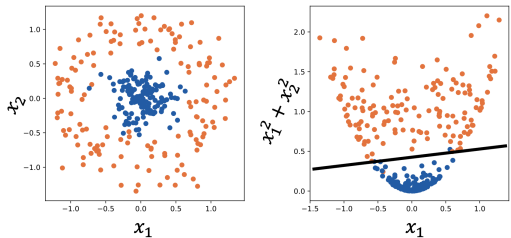
feature engineering



- But sometimes we know *a priori* that there are transformations simplifying the problem $\rightarrow$ 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



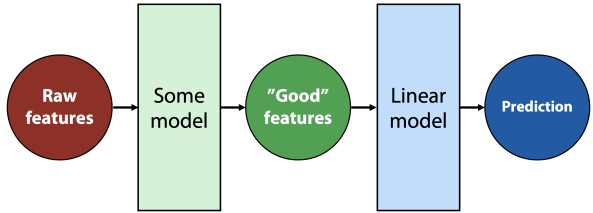
- But sometimes we know *a priori* that there are transformations simplifying the problem $\rightarrow$ even linear model can do the job
- However, this **feature engineering** is non-trivial, requires domain knowledge and is time-consuming

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

Neural Network

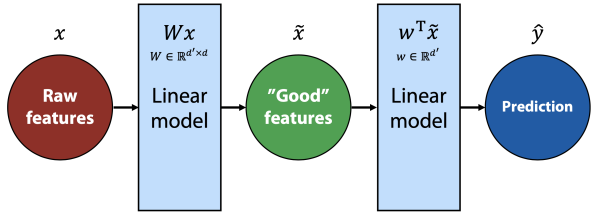
Neural Network — automating FE

NN illustrations from ML in HEP 2020



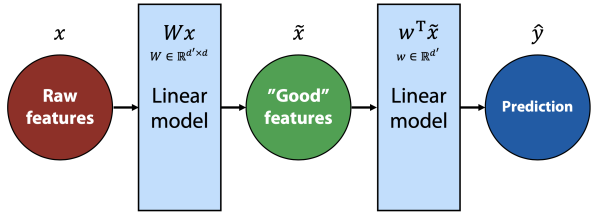
- Let's use a **simple linear model** to solve our supervised problem
- Add a block to a linear model which will automatically **generate new features** for it
- Two blocks would work together as a **single model** $\Rightarrow$ their parameters are updated simultaneously
- And **automatically**, by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

Neural Network — automating FE



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

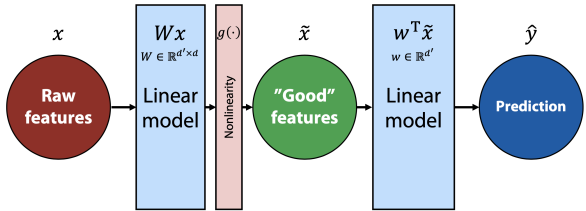
Neural Network — automating FE



- Would a linear model work as a feature generating model? No
- $\hat{y} = w^T\tilde{x} = w^T(Wx) = (w^TW)x = w'^Tx \Rightarrow$ it is still a linear model

→ Input feature space has not changed, only the model weights

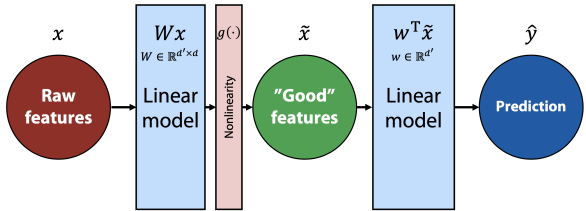
Neural Network — automating FE



- 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

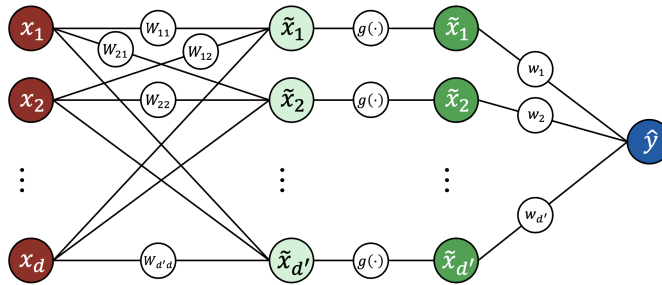


- Let's then introduce **nonlinearity** to our model

➔ $\hat{y} = w^T\tilde{x} = w^T g(Wx)$,
where $g(\cdot)$ – some nonlinear scalar function (applied elementwise)

➔ This is the simplest example of a **neural network**

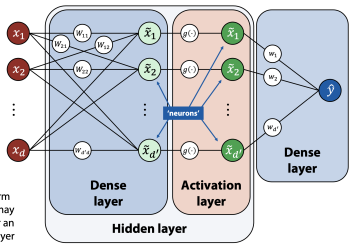
Neural Network — architecture



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

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

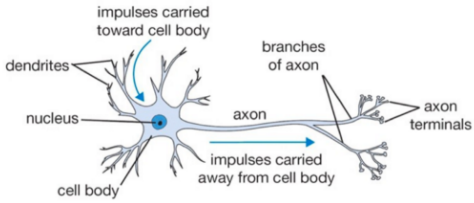
Feed-forward network:

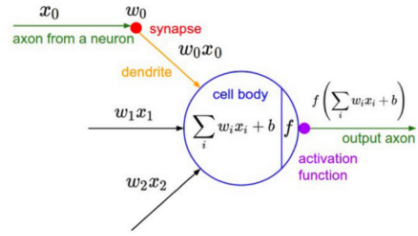


Note: the term "activation" may also stand for an output of a layer

- Brown nodes $x_1, x_2, ..., x_d$ – features from an **input layer**
- Green nodes $\tilde{x}_1, \tilde{x}_2, ..., \tilde{x}_{d'}$– **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 — how to train?

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

$$\boldsymbol{\omega}_{k+1} \leftarrow \boldsymbol{\omega}_k - \eta \cdot \nabla Q(\boldsymbol{\omega}_k)$$

- 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 — chain rule

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

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

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

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

Training — chain rule

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

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

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 — chain rule

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

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

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

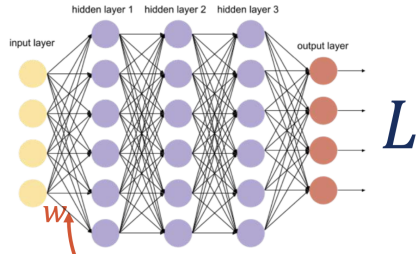
$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

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

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

Training — backpropagation

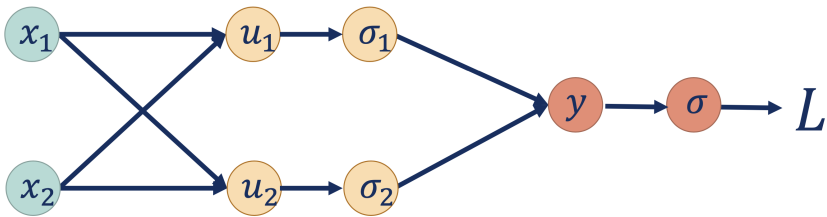
backprop illustrations from DMIA



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

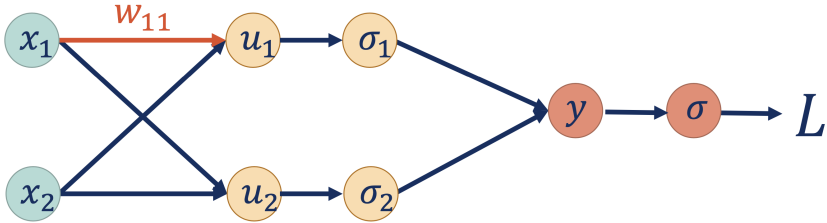
Training — backpropagation

backprop illustrations from DMIA



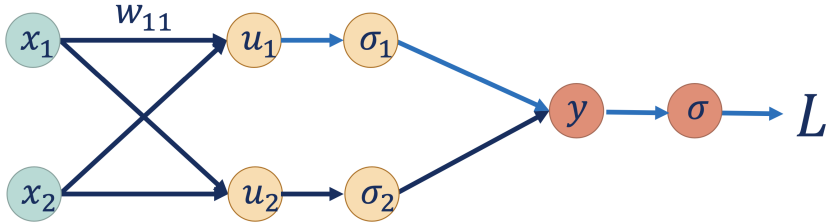
- Let's consider a simplified neural network
- Represent it in the form of a **computational graph**

Training — backpropagation



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

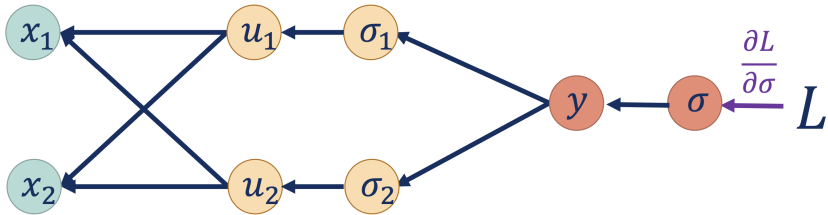
Training — backpropagation



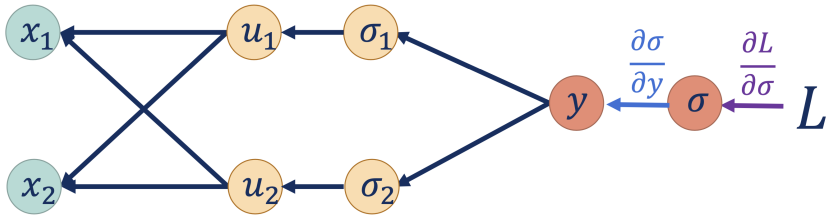
$$w_{11}^{(t+1)} = w_{11}^{(t)} - \gamma_t \frac{\partial L}{\partial w_{11}}$$

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

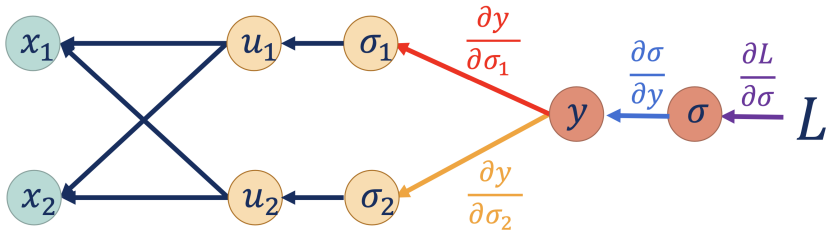
Training — backpropagation



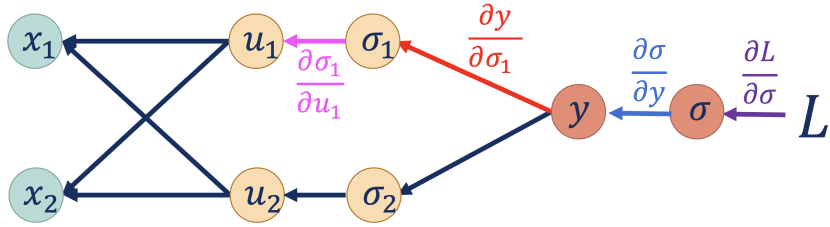
Training — backpropagation



Training — backpropagation

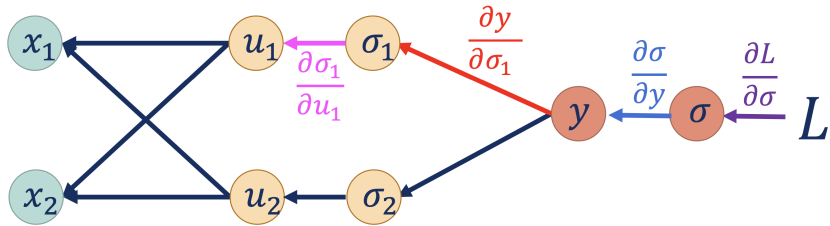


Training — backpropagation



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

Training — backpropagation



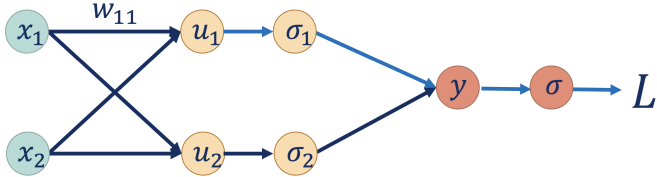
$$\frac{\partial L}{\partial u_1} = \frac{\partial L}{\partial \sigma}\frac{\partial \sigma}{\partial y}\frac{\partial y}{\partial \sigma_1}\frac{\partial \sigma_1}{\partial u_1}$$

- 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!

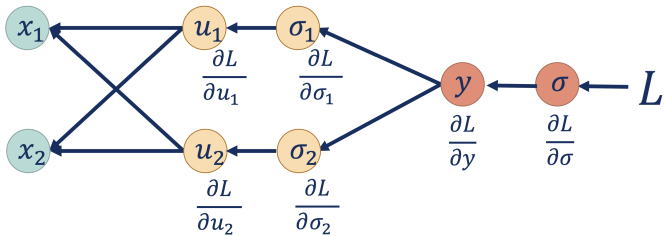


1. make **forward pass** through NN to calculate the output of each neuron and value of loss function

Training

wrap it up

train NN in your browser!

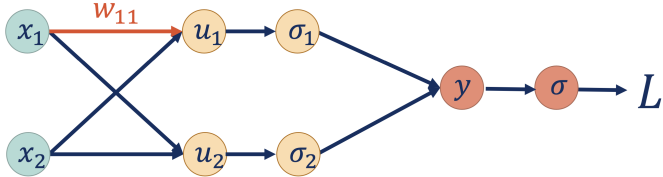


1. make **forward pass** through NN to calculate the output of each neuron and value of loss function
2. with **backward pass** go back through NN to calculate gradients for all weights

Training

wrap it up

train NN in your browser!



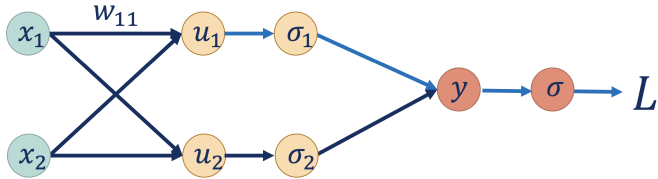
$$w_{11}^{(t+1)} = w_{11}^{(t)} - \gamma_t \frac{\partial L}{\partial w_{11}}$$

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

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.

Backup slides — features in ML: HEP

Supervised learning

Classification

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

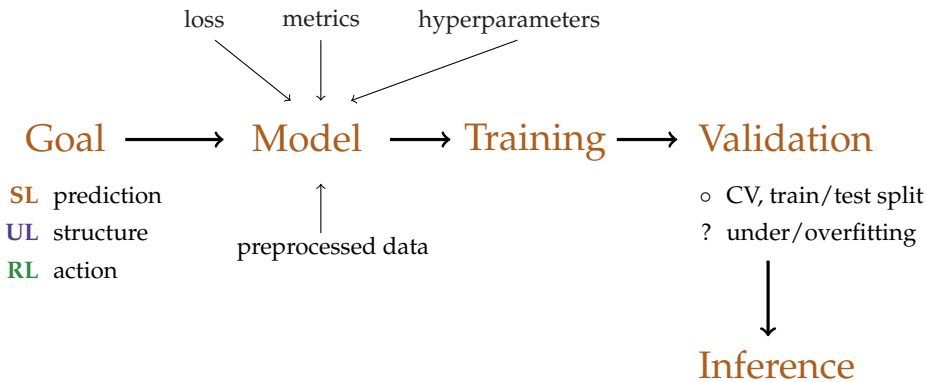
Regression

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

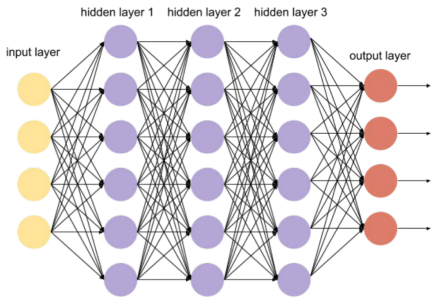
Don't forget about **unsupervised learning**, **reinforcement learning**, **semi-supervised learning** and so on.

Backup slides

what is ML pipeline?



Neural network

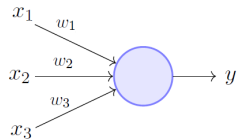


$$h_t = f(W h_{t-1} + b)$$

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

Backup slides

calculating weights: slide 1



Perceptron Model (Minsky-Papert in 1969)

Source

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

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

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

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

Backup slides

calculating weights: slide 2

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

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

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

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

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

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

Backup slides

calculating weights: slide 2

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

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}?

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

Backup slides

calculating weights: slide 2

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

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

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

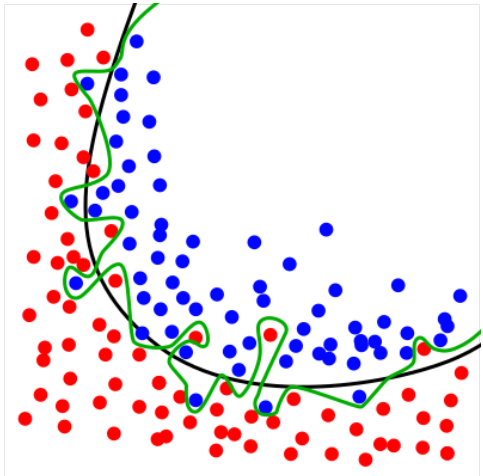
How to find w_{new} and b_{new}?

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

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

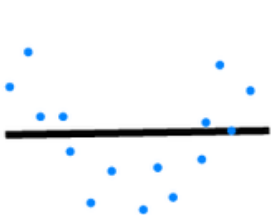
Backup slides

overtraining: slide 1



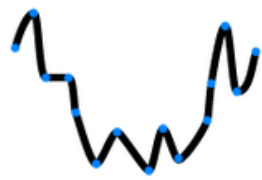
Backup slides

overtraining: slide 2



Underfitting

Desired

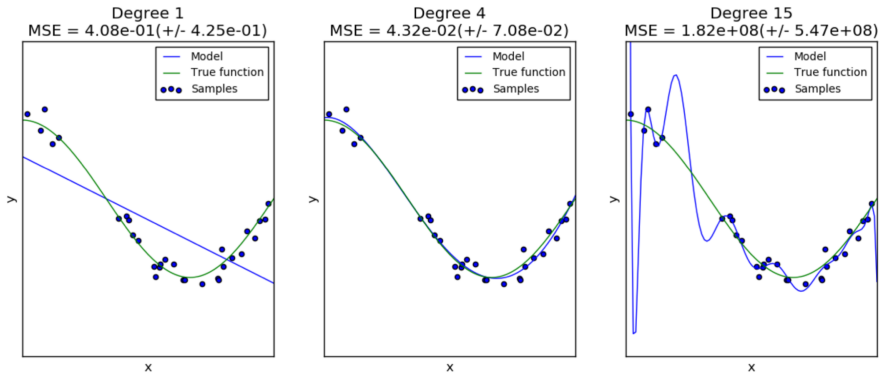


Overfitting

Backup slides

overtraining: slide 3

Overfitting



Backup slides

overtraining: slide 4

История про танки

Классификатор: есть танки на снимке или нет

Backup slides

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)
- ...

Backup slides

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
- ...

Backup slides — ML challenges: slide 3 (A. Ustyuzhanin)

Calorimeter features

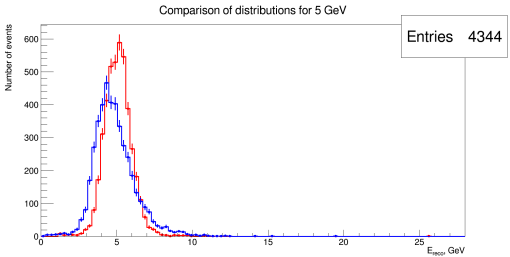
- Measured particle energy
- Shower parameters: length, width, ...
- Number of clusters in each layer Intensity of the clusters
- Intensity of the clusters
- Distance form track of the original particle
- ...

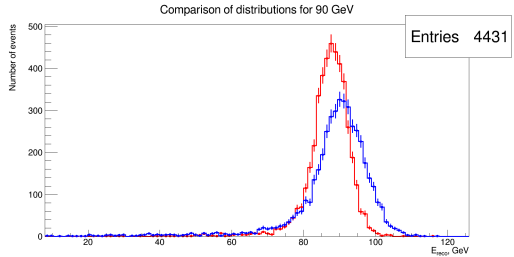
Muon detector features

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

Backup slides

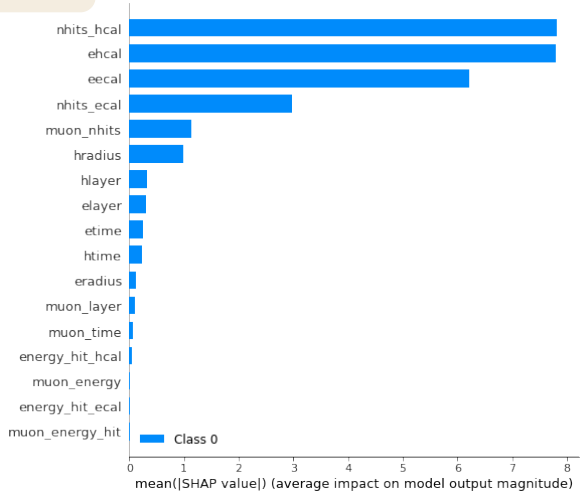
Example of my regression problem





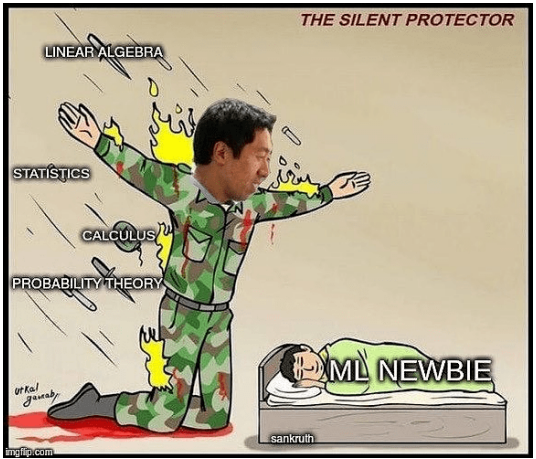
Backup slides

SHAP value



Backup slides

Math and ML



Backup slides

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

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: UserWar
ning: 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

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_pre1.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)
```