Introduction to Machine Learning

Computing Methods for Experimental Physics and Data Analysis

Andrea.Rizzi@unipi.it

Topics

1. Introduction to machine learning
   a. Basic concepts: loss, overfit, underfit
   b. Examples of linear regression, boosted decision trees
   c. Exercise with colab, numpy, scikit
2. Deep Neural Networks
   a. Basic FeedForward networks and backpropagation
   b. Importance of depth, gradient descent, optimizers
   c. Introduction to tools and first exercises
3. Convolutional and Recurrent networks
   a. Reduction of complexity with invariance: RNN and CNN
   b. CNN exercise
4. Autoencoders and Generative Adversarial Networks
   a. GAN exercises
5. Graph Neural Network

Quick Poll

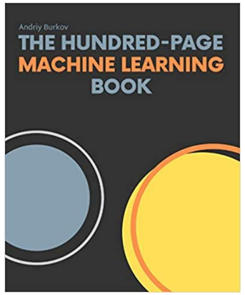
- How many of you…
  - ...know what an artificial neural network is?
  - ...used a multivariate / machine learning / neural network / BDT analysis technique?
  - ...know what a linear regression is? and what PCA is?

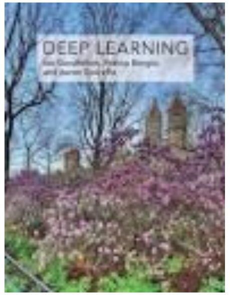
Introduction on these ML lectures

Compared to past 2 years we increased the hours spent on ML, still we can just give an introduction and to try to explain some concepts of what machine learning and deep learning techniques are (trying not to go too formal, but definitely beyond the "blackbox")

This is also an active research field, new ideas emerge every year, if interested you should learn to stay tuned (read, study, update your knowledges)

Suggested books (both having free online versions)

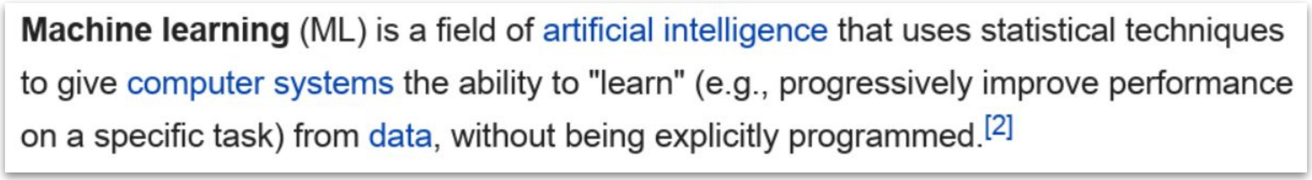




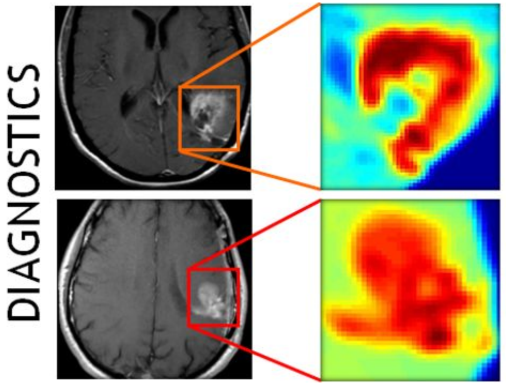
- **"The 100 pages ML book",** Andriy Burkov
  - Can be read almost from A to Z
  - **http://themlbook.com/wiki/doku.php**
- **"The Deep Learning Book"**, Goodfellow, Bengio, Courville (MIT Press)
  - More like a reference book, read individual chapters
  - https://www.deeplearningbook.org/

Machine learning

A possible definition (from wikipedia):



Machine learning (ML) is a field of artificial intelligence that uses statistical techniques to give computer systems the ability to "learn" (e.g., progressively improve performance on a specific task) from data, without being explicitly programmed.[2]



Replace "programmers" with computer programs

- Learn from examples ("training" phase)

Multiple applications, for example:

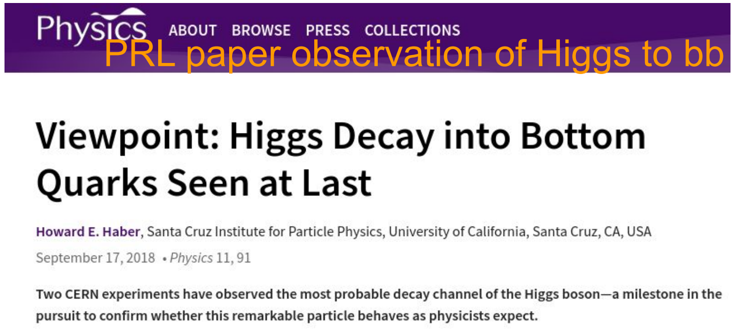
- Image and speech processing
- Agents able to play chess, go or drive cars
- Detect anomalies (e.g. in credit card usage)
- Applications for research in various scientific fields
  - E.g. physics!

In experimental and applied physics

examples are everywhere..

- Particle identification and kinematic measurement
- Signal to background discrimination (BDT and DNN are very popular in HEP experiments)
- Computer assisted processing of medical exams (ECG, CT, etc...)
- Processing of astrophysics data

… and your ideas! This is a growing field ...


Physics ABOUT BROWSE PRESS COLLECTIONS

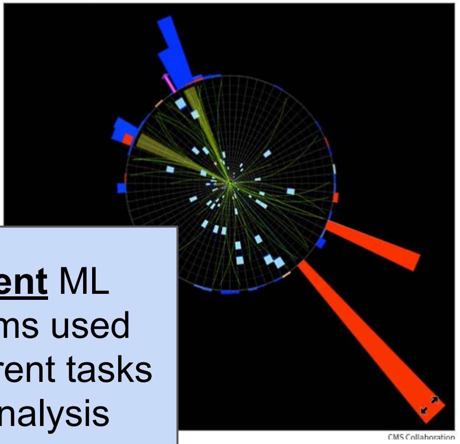
PRL paper observation of Higgs to bb

Viewpoint: Higgs Decay into Bottom Quarks Seen at Last

Howard E. Haber, Santa Cruz Institute for Particle Physics, University of California, Santa Cruz, CA, USA

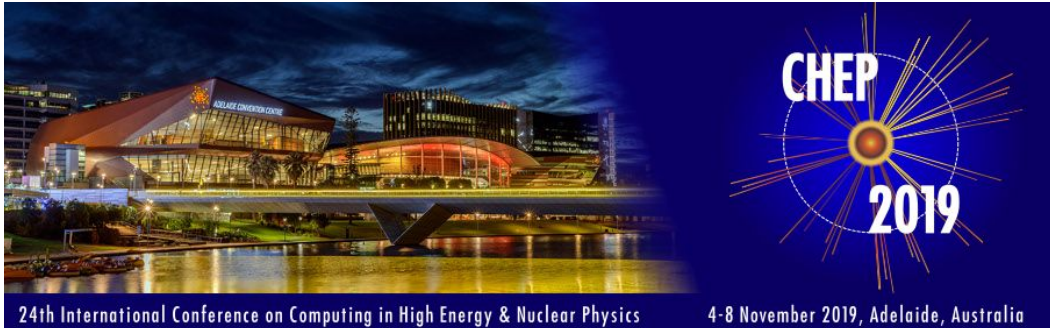
September 17, 2018 • *Physics* 11, 91

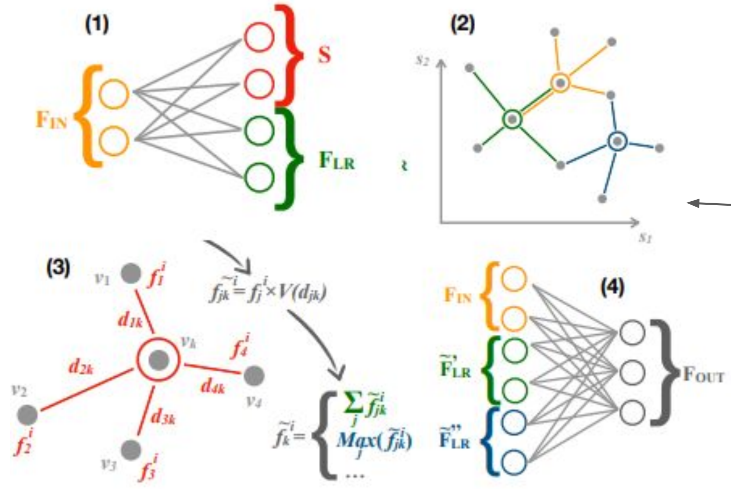
Two CERN experiments have observed the most probable decay channel of the Higgs boson—a milestone in the pursuit to confirm whether this remarkable particle behaves as physicists expect.



<u>4 different</u> ML algorithms used for different tasks in this analysis

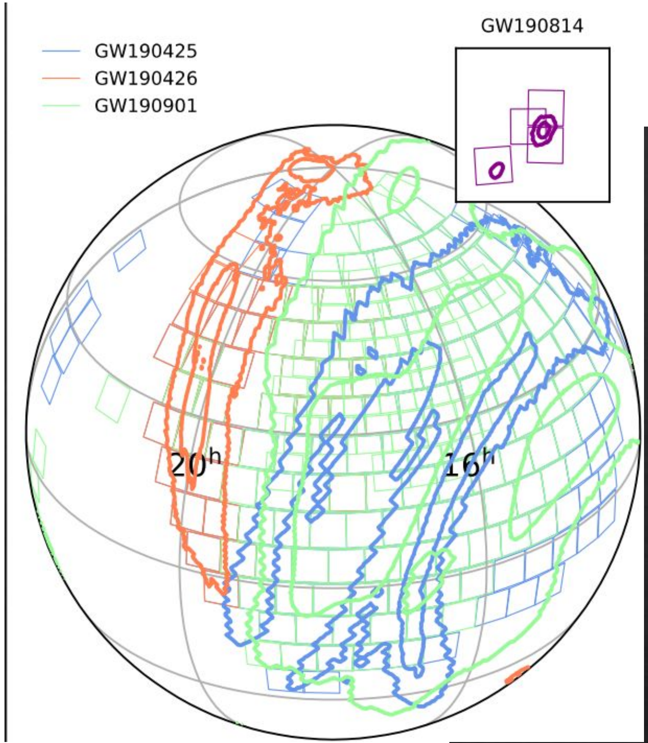
Computing in High Energy Physics Conference

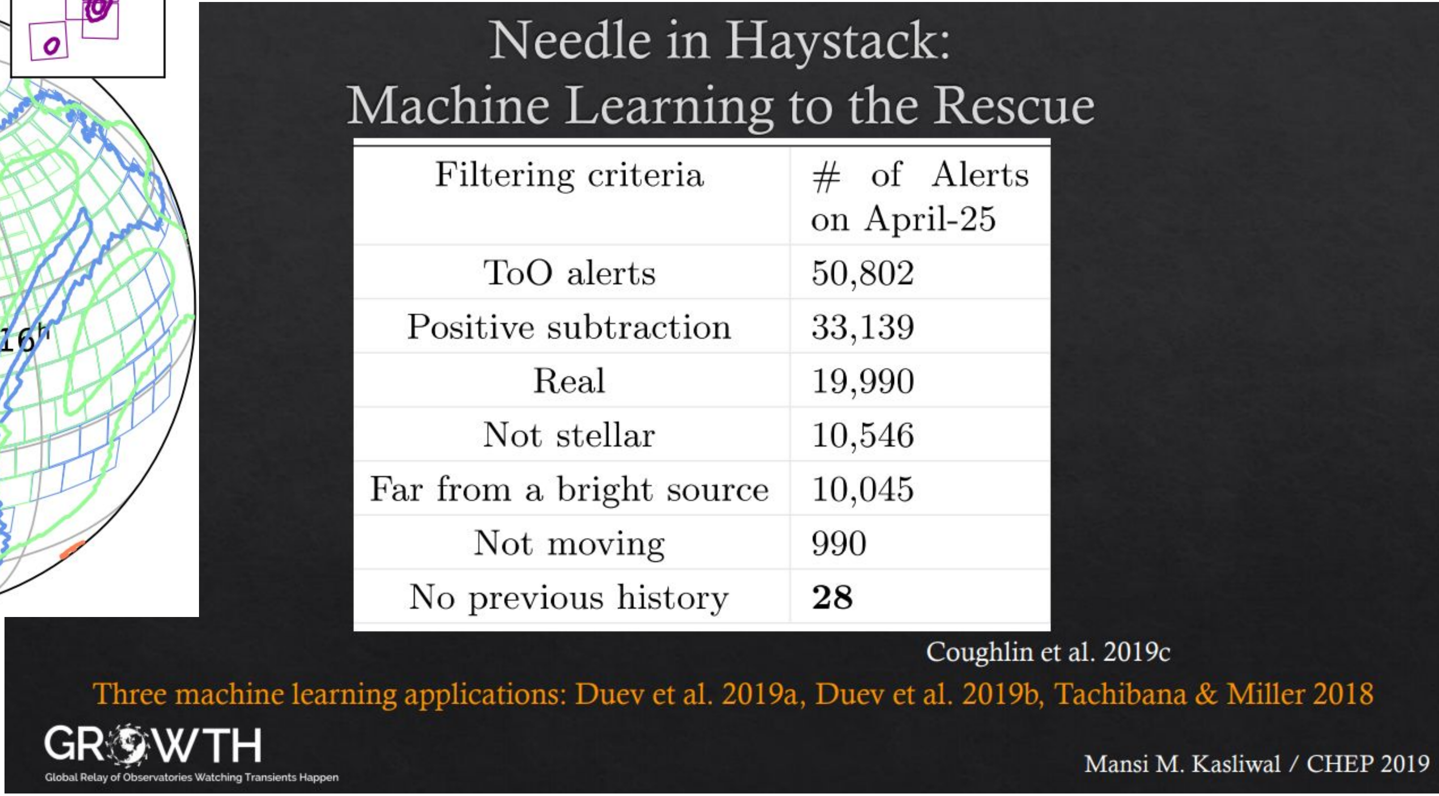

CHEP
2019
24th International Conference on Computing in High Energy & Nuclear Physics
4-8 November 2019, Adelaide, Australia



- Machine learning for QCD theory and data analysis
- BESIII drift chamber tracking with machine learning
- FPGA-accelerated machine learning inference as a service for particle physics computing
- Constraining effective field theories with machine learning
- Fast simulation methods in ATLAS: from classical to generative models
- Using ML to Speed Up New and Upgrade Detector Studies
- The Tracking Machine Learning Challenge
- *Particle Reconstruction with Graph Networks for irregular detector geometries*
- ...42 contribution with “Machine Learning” in the title/abstract

Real time alerts and automatic telescope pointing





Needle in Haystack: Machine Learning to the Rescue

Filtering criteria	# of Alerts on April-25
ToO alerts	50,802
Positive subtraction	33,139
Real	19,990
Not stellar	10,546
Far from a bright source	10,045
Not moving	990
No previous history	**28**

Coughlin et al. 2019c

Three machine learning applications: Duev et al. 2019a, Duev et al. 2019b, Tachibana & Miller 2018

GROWTH
Global Relay of Observatories Watching Transients Happen

Mansi M. Kasliwal / CHEP 2019

Machine Learning basics (or the “dictionary” for next lectures)

Types of typical ML problems

- **Classification:** which category a given input belongs to.
- **Regression:** value of a real variable given the input.
- **Clustering:** group similar samples
- **Anomaly detection:** identify inputs that are different from others
- **Generation/synthesis of samples:** produce new samples, similar to the original data, starting from noise/random numbers
- **Denoising:** remove noise from an input dataset
- **Transcriptions:** describe in some language the input data
- **Translations:** translate between languages
- **Encoding and decoding:** transform input data to a different representation
- ...many more...

Function approximation

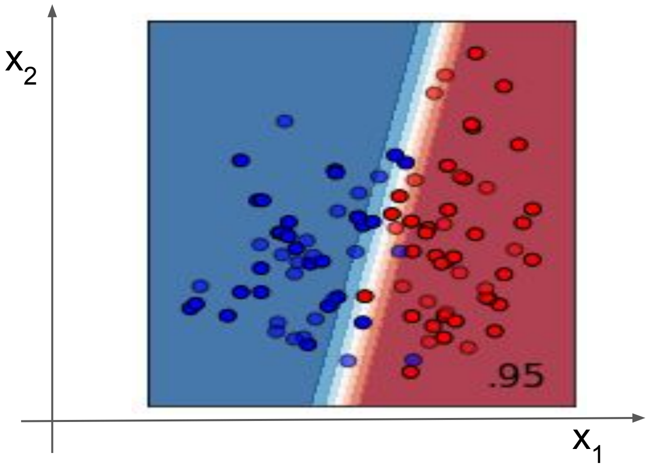
- The goal of a ML algorithm is to approximate an unknown function (often related to some Probability Density Function of the data) given some example data
- The function is often $f(\mathbf{x}): R^n \to R^m$ (often $m=1$)
  - In **classification** we try to approximate the probability for each example, with inputs represented as a vector $\mathbf{x}$ to belong to a given category (y) (e.g. the probability to be a LHC Higgs signal event vs a Standard Model background one)
  - In **regression** we approximate the function that given the inputs ($\mathbf{x}$) returns the value of the variable to predict (y) (e.g. given the data read from some particle detectors, estimate the particle energy)

We usually call "y" the **output or target**

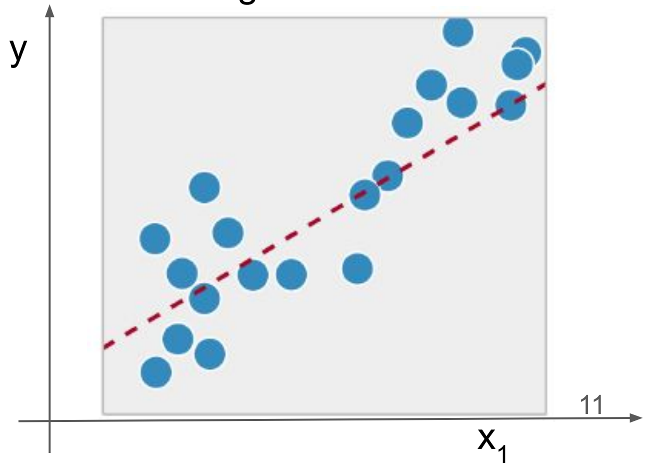
$$y=f(\mathbf{x})$$

We usually call "x" the **inputs or features**

classification



regression



Model and Hyper-parameters

- A model for the functions that can be used to approximate the "*f(**x**)*" must be specified. The model can be something simple (e.g. sum of polynomials up to degree **N**) or more complex (e.g. all the functions that could be coded in **M** lines of C++)
- Different ML techniques are based on different "models"
    - Each technique ("class of model") further allow to specify the exact model
    - The parameters describing the exact model are called "**hyper-parameters**" (e.g. the degree **N** of the polynomial, or the maximum number of C++ line **M** can be considered hyper parameters)
- We will see example of techniques with different models and complexity:
    - Linear regression
    - Decision trees
    - Principal Component Analysis
    - Nearest Neighbour
    - **Artificial Neural Networks**

Parameters

- A specific model typically have parameters (e.g. the coefficients of the polynomials or the characters of the 10 lines of C++).
- Parameters are learned in the “training phase”.
- Different models or similar model with different hyper-parameters settings have different *n.d.o.f.* in the parameters phase space

y(x) = ax + bx^2 + cx^3 + d   (*a,b,c,d* are the parameters)

Objective function

- A goal for what is “a good approximation” have to be defined
- This is called objective function (or **loss function** or error function …)
- Is a function that returns higher(or lower) value depending how good or bad the approximation is
  - *Loss* functions have to be *minimized*
- Examples of loss functions
  - Classification problems: binary cross entropy
  - Regression problems: Mean Square Error (i.e. the chi2 with sigma=1, I hope you are not surprised by this choice!)

The process is not very different from a typical phys-lab1 chi2 fit… but the number of parameters can be several orders of magnitude larger (10^3 to 10^6)

Objective function: **binary cross entropy**

- In classification problems the function to approximate is typically R^n -> [0,1]
  - Where, for example, 0 means background and 1 means signal
- The binary cross entropy is defined as follows ($\hat{y}_i$ is the output of the classifier)

$$D_{BCE}(y,\hat{y}) = -\frac{1}{N}\sum_{i=1}^{N}\left[y_i log(\hat{y}_i) + (1-y_i) log(1-\hat{y}_i)\right]$$

- The above function has large value when an example with y=1 is classifed as a ~ $\hat{y}_i$ and no loss when ~ $\hat{y}_i$
  - Viceversa if y=0 …
- Minimizing the binary cross-entropy we maximize the likelihood in a process with 0/1 outcome (where the output of the tagger is interpreted as a probability)

$$L = \prod_i p_i^{y_i}(1-p_i)^{1-y_i}$$

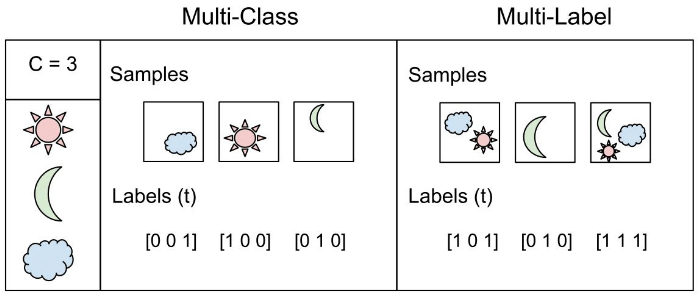
$$-log(L) = -log(\prod_i p_i^{y_i}(1-p_i)^{1-y_i}) = -\sum_i [y_i log(p_i) + (1-y_i) log(1-p_i)]$$

Learning / Training

- For a given model, and given set of hyper-parameters, how do we infer the parameters that minimize the objective function?
- The idea of ML is to get the parameters from “data” in a so called “training” step
- Each ML technique has a different approach to training
- Different types of training
    - **Supervised**: i.e. for each example we know the correct answer
    - **Unsupervised**: we do not know “what is what”, we ask the ML algorithm to learn the probability density function of the examples in the **features** (i.e. the inputs!) space
    - **Reinforcement learning**: have agents playing a punishment/reward game

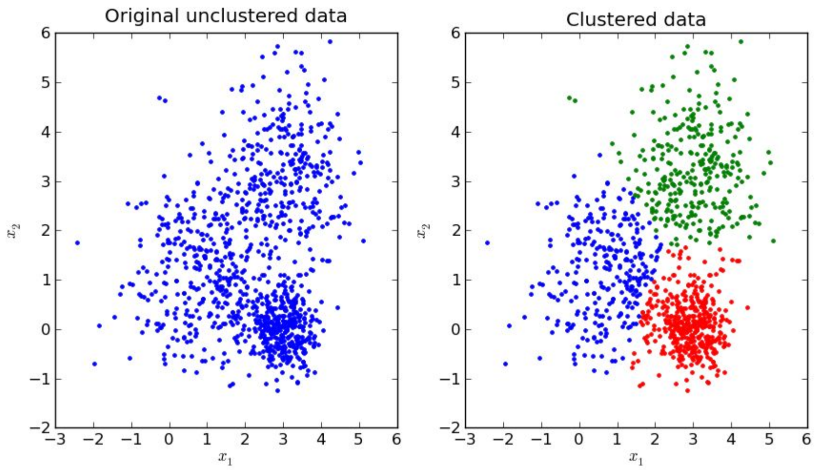
Supervised learning

- We want to teach something we (the supervisors) already know (at least on the training samples)
- For each example we need to have the “right answer” / “truth” , for example:
  - Labels telling if a given example **signal** or **background**
  - Labels classifying the content of an image (multiple labels are possible)
  - Correct values of some quantity, e.g. generated energy of a particle in a detector simulation
  - More complex multi-label or multi-class classification
- Sample can be labelled in various ways:
  - Humans labelling existing data
  - Data being “generated” from known functions (e.g. simulations)
- Learn the probability of the label y, given the input **x**, i.e. $P(y\,|\,\boldsymbol{x})$



Unsupervised learning

- Often we do not have labels (or we have labels only for few data points)
- Unsupervised learning techniques allow to train networks that can perform similar tasks as the supervised ones, e.g.
  - Classification of “common” patterns (clustering)
  - Dimensionality reduction, compression
  - Prediction of missing inputs
  - Anomaly detection



- In practice learn the Probability Density Function of the data, independently of any “label” variable, i.e. P($\boldsymbol{x}$)

Supervised vs unsupervised

Supervised and unsupervised are not as different as one would imagine, in fact

- P($\boldsymbol{x}$) can be seen as *n* supervised problems, one for each feature

$$p(\mathbf{x}) = \prod_{i=1}^{n} p(\mathrm{x}_i \mid \mathrm{x}_1, \ldots, \mathrm{x}_{i-1})$$

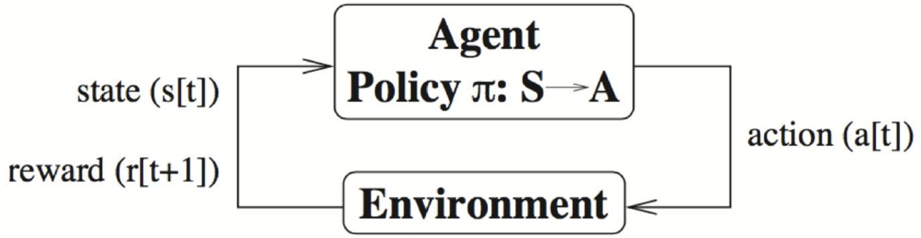
- P(y | $\boldsymbol{x}$) can also be computed, if we treat y as an “$\boldsymbol{x}$” in unsupervised learning deriving hence $p(\mathbf{x}, y)$, as

$$p(y \mid \mathbf{x}) = \frac{p(\mathbf{x}, y)}{\sum_{y'} p(\mathbf{x}, y')}$$

Reinforcement learning

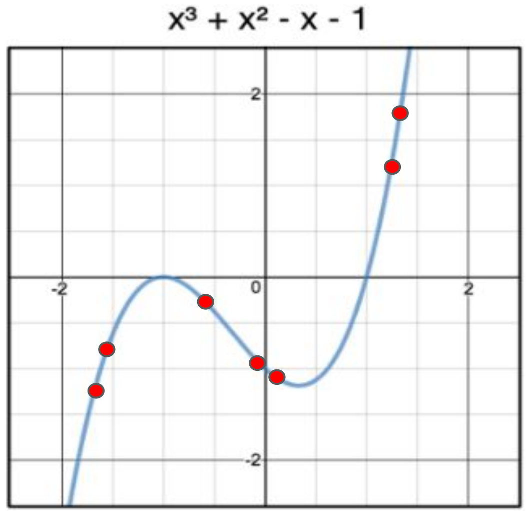
Applies to “agents” acting in an “environment” that updates their state

- It is similar to supervised learning as a “reward” has to be calculated
- The *supervisor* anyhow doesn’t necessarily know what is the best action to perform in a given state to interact with the environment, it just computes the final reward
- Learn to make best decision in a given situation
    - The right move in chess or go match
    - Drive a car in the traffic
    - Etc..



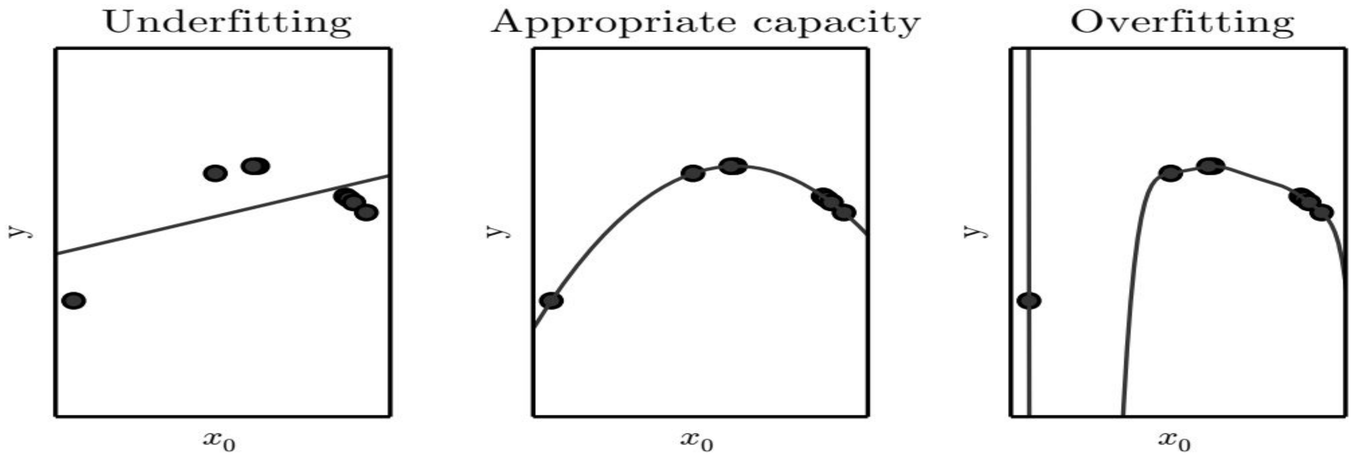
Capacity and representational power

- Different models (i.e. techniques/hyper-parameters values) allow to represent different type of functions
- Models with more free parameters typically can approximate a larger number of functions (or can better approximate a given function) => **higher capacity**
- Remember: we do not know the actual function to approximate, we just want to **learn from examples**
- With limited samples we have a tradeoff to handle:
  - accuracy in representation **vs** generalization of the results



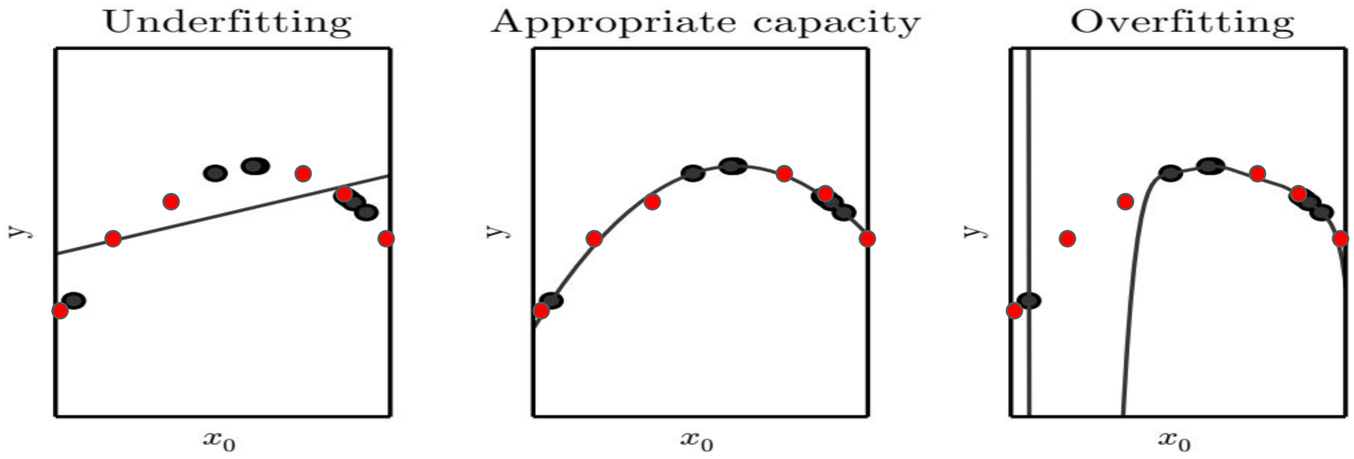
Capacity and representational power

- Underfitting: the sample is badly represented
- Overfitting / Appropriate capacity are less obvious to define
    - Lack of “generalization” -> overfitting



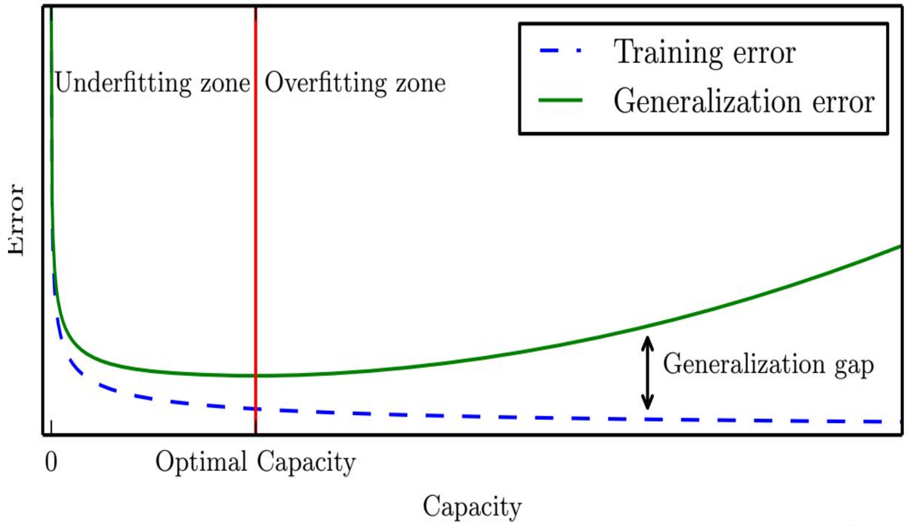
Capacity and representational power

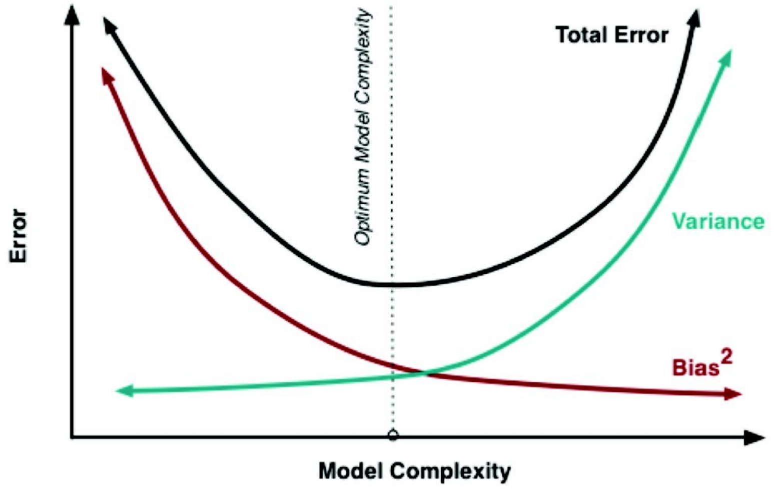
- Underfitting: the sample is badly represented
- Overfitting / Appropriate capacity are less obvious to define
    - Lack of “generalization” -> overfitting
    - Typical method is to check on independent sample
        - Or just split your sample in two and use only half for training



Generalization

- We can compare the accuracy between the “training” sample and the “generalization/validation” sample





- Bias/variance trade-off
  - y: function (with random noise)
  - h(x): approximated function

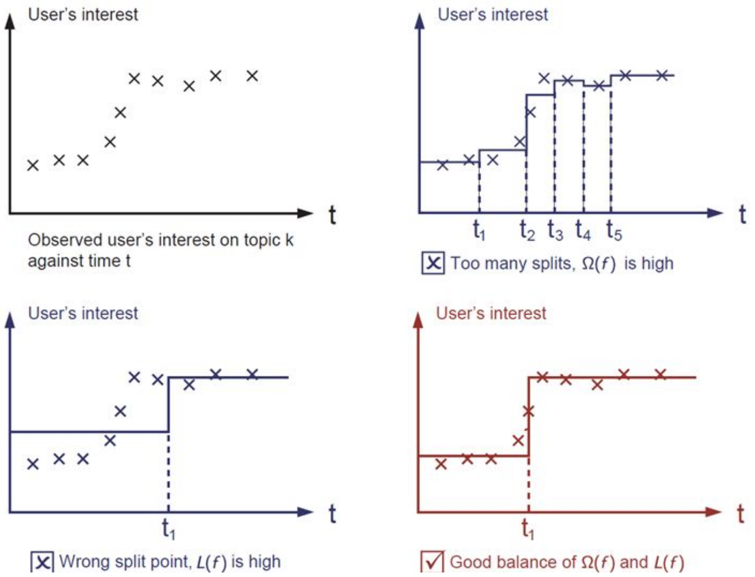
$$E[(y-h(x))^2] = \underbrace{E[(y-\bar{y})^2]}_{\text{Noise}} + \underbrace{(\bar{y}-\bar{h}(x))^2}_{\text{Bias Squared}} + \underbrace{E[(h(x)-\bar{h}(x))^2]}_{\text{Variance}}$$

Regularization

In order to control the “generalization gap”

- the objective function can be modified adding a regularization term
  - Introduce a “cost” in increasing the capacity of the model or in accessing some parts of the model-parameters space
- the examples in training dataset can be increased with augmentation techniques
  - Adding stochastic noise to existing examples
  - Transforming the existing examples with transformation that are known to be invariant for the solution we look for

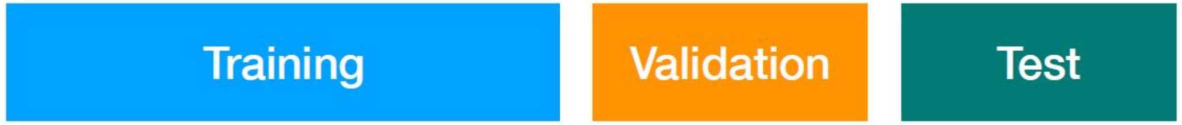
$$\text{obj}(\theta) = L(\theta) + \Omega(\theta)$$



https://xgboost.readthedocs.io/en/latest/tutorials/model.html

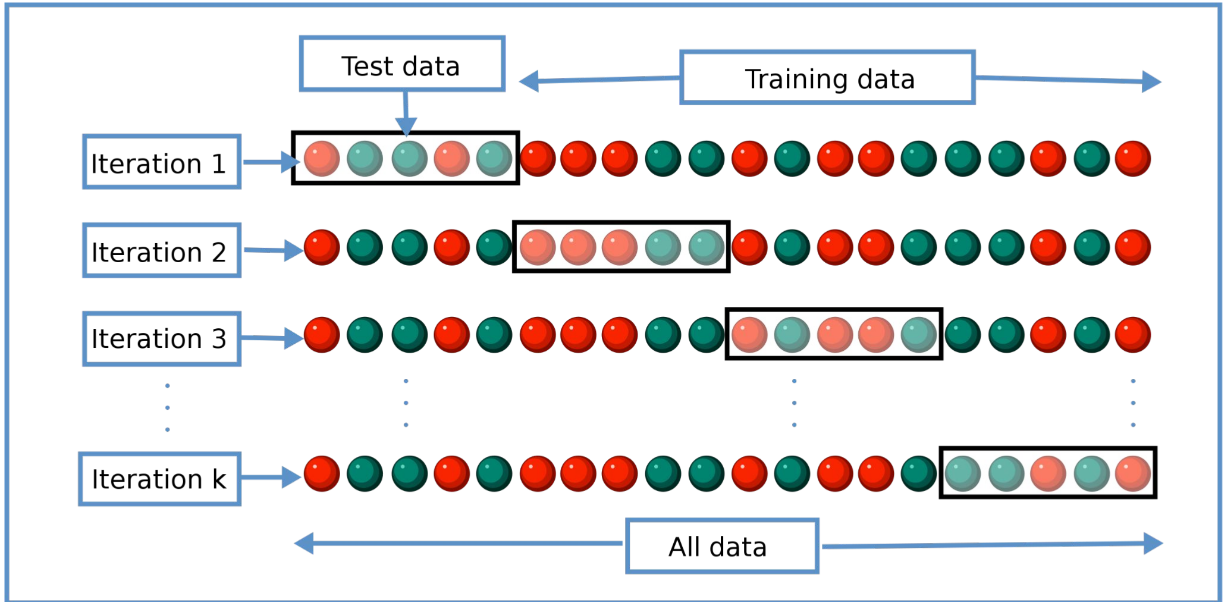
Hyperparameters(model) optimization

- It is normal to have to test a few, if not several, configurations in the model hyper-parameter space
  - Scans of hyper-parameters are often performed
  - Different techniques used
- Effectively a “second” minimization is done
  - First minimization is on the parameter => minimize on the “training dataset”
  - Second minimization is on the hyper-parameters => minimize on the “validation dataset”
- A third dataset (“test dataset”) is then also needed
  - To assess the performance of the algorithm in an unbiased way
  - To make an unbiased prediction of the algorithm output
- Original dataset is typically split in uneven parts to be used as *training, validation* and *test*



K-folding cross validation

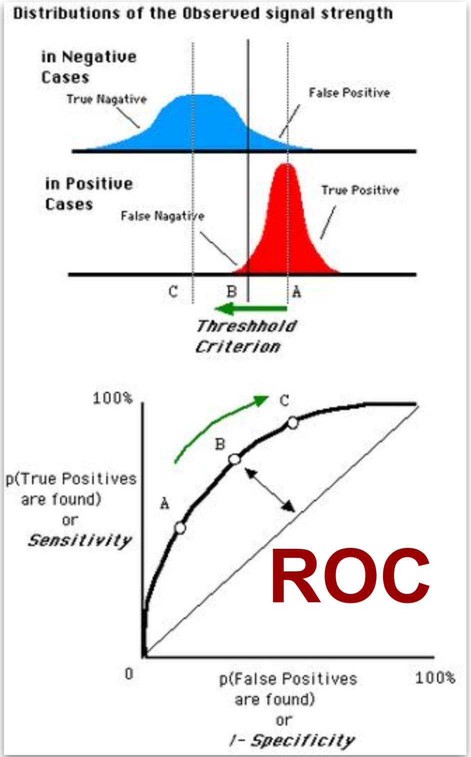
- If the sample is statistically limited, splitting in 3 chunks means loosing examples
- With K-folding, “K” independent trainings are performed, each using a different chunk of data for “training” and for “testing” (and another one for validation if a hyper parameter scan is performed)



Inference

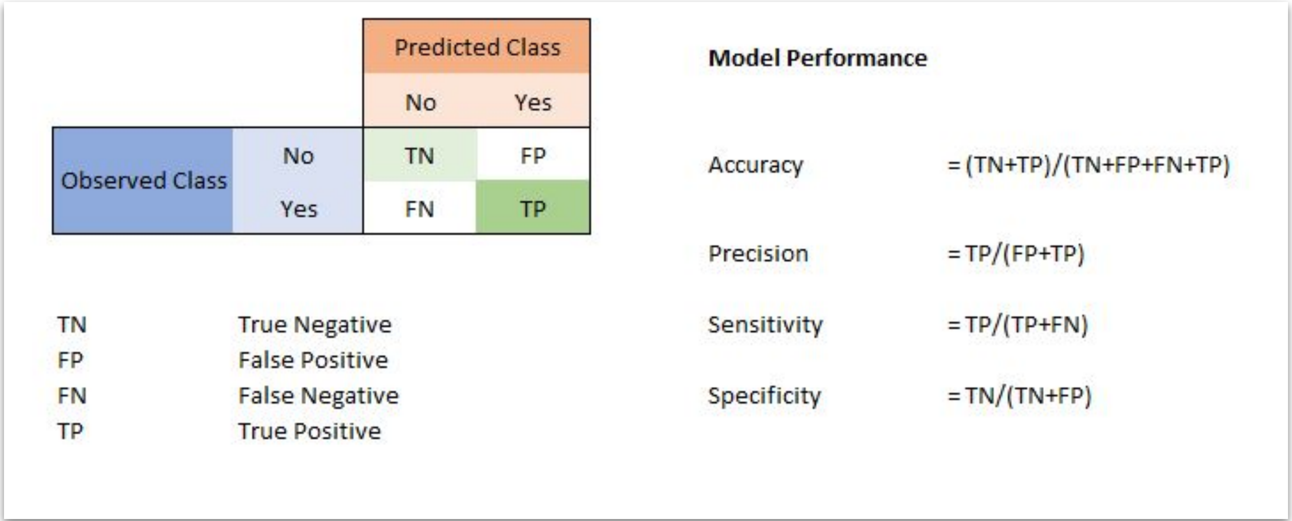
- A ML model that has been trained can than be used to act on some new data (or on the test dataset if a prediction has to be made)
- The evaluation of the algorithm output on the “unseen” data is called *inference*
- From a computing point of view *inference* is usually faster than *training*

Accuracy, Precision, Sensitivity, Specificity



Distributions of the Observed signal strength
in Negative Cases
True Nagative
False Positive
in Positive Cases
False Nagative
True Positive
C
B
A
Threshhold Criterion
100%
p(True Positives are found) or Sensitivity
0
p(False Positives are found) or 1- Specificity
100%

ROC



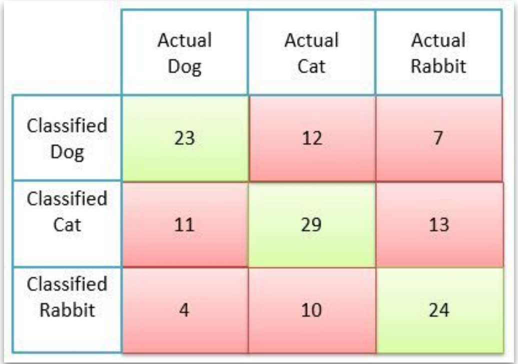
		Predicted Class	
		No	Yes
Observed Class	No	TN	FP
	Yes	FN	TP

TN	True Negative
FP	False Positive
FN	False Negative
TP	True Positive

Model Performance

Accuracy	= (TN+TP)/(TN+FP+FN+TP)
Precision	= TP/(FP+TP)
Sensitivity	= TP/(TP+FN)
Specificity	= TN/(TN+FP)

Confusion Matrix



	Actual Dog	Actual Cat	Actual Rabbit
Classified Dog	23	12	7
Classified Cat	11	29	13
Classified Rabbit	4	10	24

Examples of ML techniques

Linear regression

SUPERVISED

- Solve a regression problem, i.e. perdict the value of y when $\boldsymbol{x}$ is given
    - Approximate an unknown "y=f(**x**)" given a some examples of (y,**x**)
- **Model**: $y=w_i x_i$, i.e. the function is a linear combination of the input paramters
- **Parameters**: w_i
- Let's suppose we have m examples in the form of pairs $(\boldsymbol{x},y)_j$
- The **objective function** can be the *mean squared error, MSE=* $|y_j - w_i x_{ij}|^2/m$
- Learning: find the w_i that minimize the MSE on the given dataset
    - Linear regression have an analytical solution (i.e. a minimum for the MSE) that can be computed by requiring the gradient of the MSE to be zero (if you want to see the math https://en.wikipedia.org/wiki/Linear_regression#Least-squares_estimation_and_related_techniques)
- We could increase the **capacity** of the model using polynomials instead of linear functions
    - The number of parameters would increase as we now would have the second order coefficients too

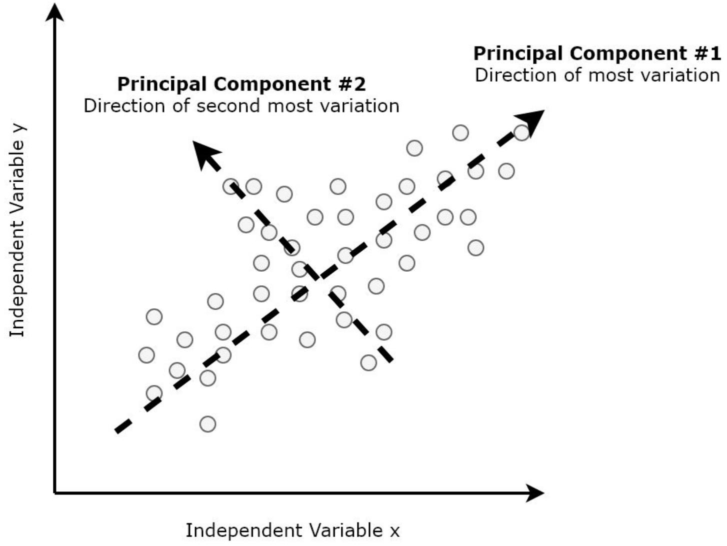
Principal Component Analysis (aka PCA)

UNSUPERVISED

- Orthogonal transformation of the input phase space such that
  - The first transformed coordinate has maximum variance
  - The 2nd transformed coordinated has 2nd max variance
  - …etc...
- Can be computed as the eigenvalue decomposition of the covariance matrix

$$\sigma^2_{ij} = \frac{1}{n}\sum_{h=1}^{n}(x_{hi} - \mu_i)(x_{hj} - \mu_j)$$

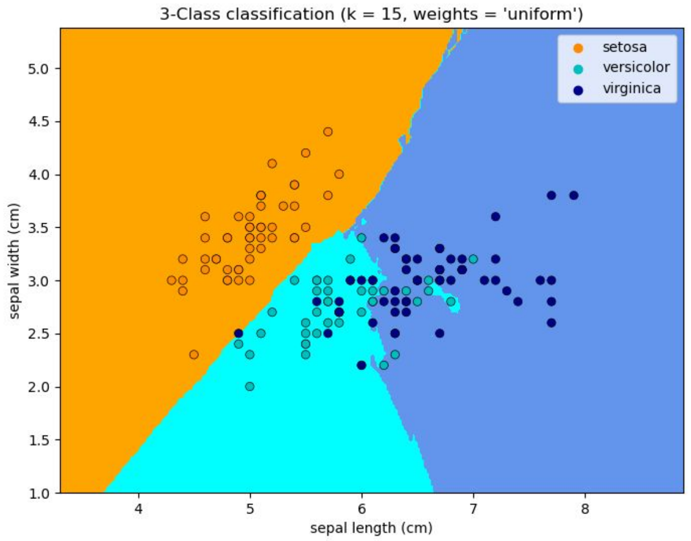
- Useful to transform the data in a normalized form (scaling by the variance of each component)
- Reduce dimensionality (by taking only first N components)

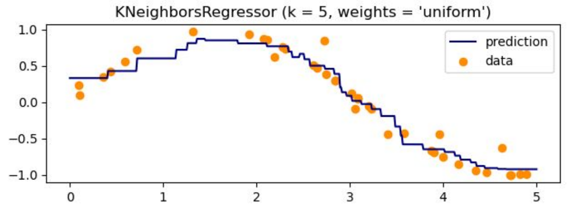


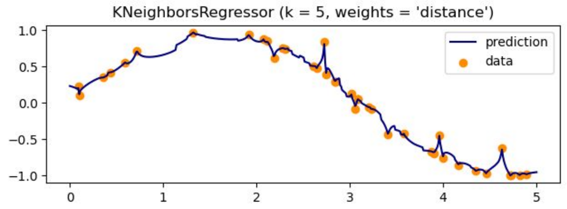
More complex dimensionality reduction **Manifold Learning:** https://github.com/jakevdp/PythonDataScienceHandbook/blob/master/notebooks/05.10-Manifold-Learning.ipynb

Nearest neighbors

- A very powerful way to do classification or regression is to look at points in the training datasets that are close to sample to evaluate
- Multiple neighbors can be used for a more stable evaluation
- On large dataset it could be a problem to keep all training points for the evaluation phase



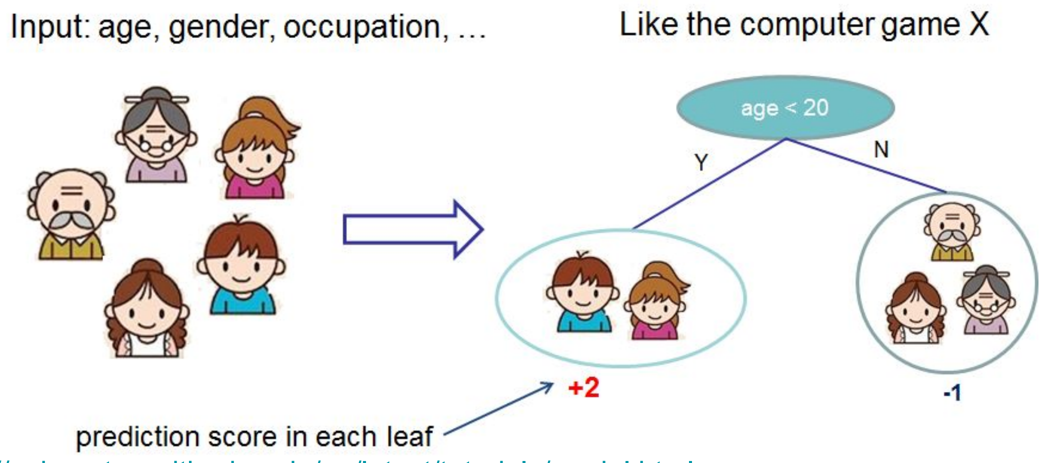


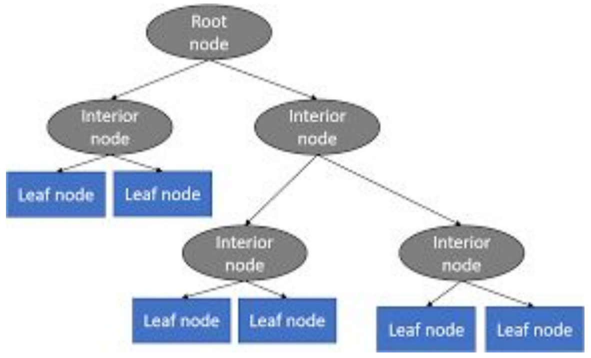


Figures from https://scikit-learn.org/stable/modules/neighbors.html

Decision trees

- The functions used in the “model” are decision trees, each node has a pass/fail condition on some input variable
- Classification and regression trees (CART)
  - Examples are categorized based on individual “cuts” on a single input feature
  - A score is given in each leaf
- Trees can have different depths (depth is an hyper-parameter)

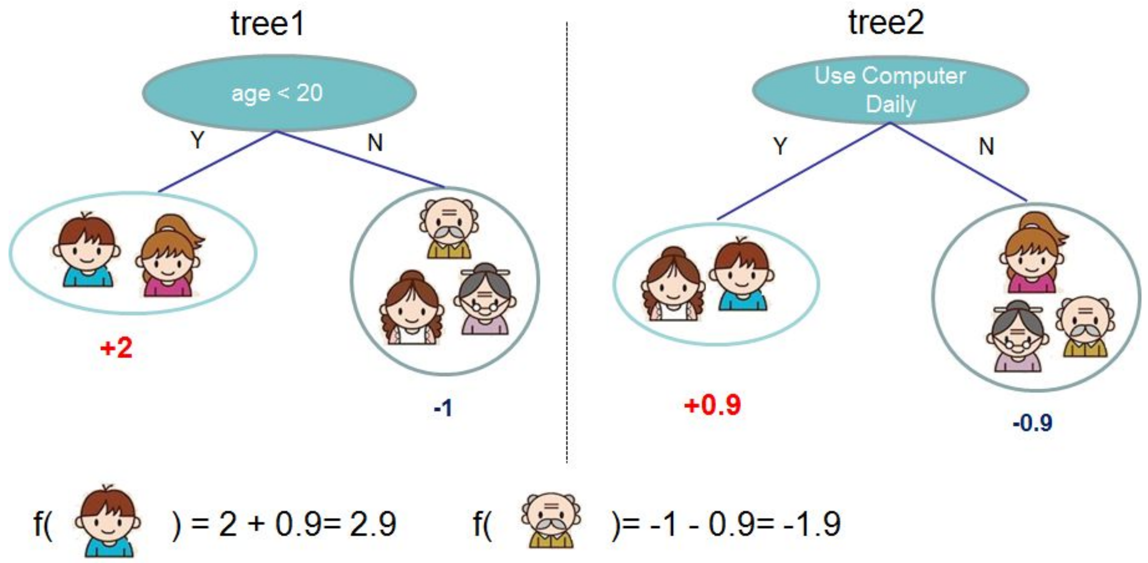

Input: age, gender, occupation, ...
Like the computer game X
age < 20
Y
N
+2
-1
prediction score in each leaf



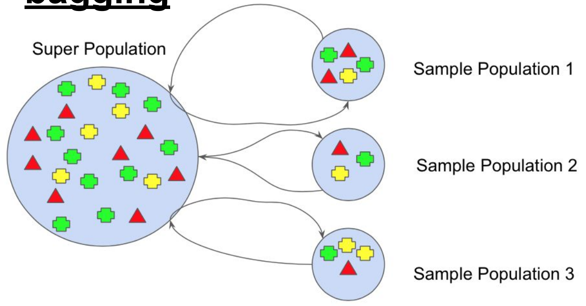
https://xgboost.readthedocs.io/en/latest/tutorials/model.html

Ensembles of trees

- A single tree is typically not a very performant
- Combine multiple trees (#trees is an hyperpar)
  - Random forest (bagging)
  - Gradient boosting
  - Adaptive boosting

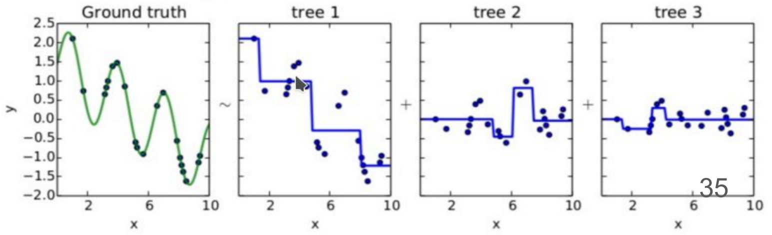


bagging



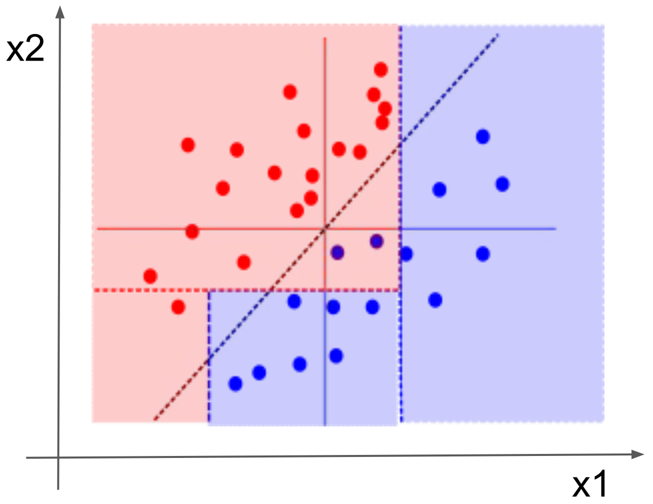
Gradient boosting

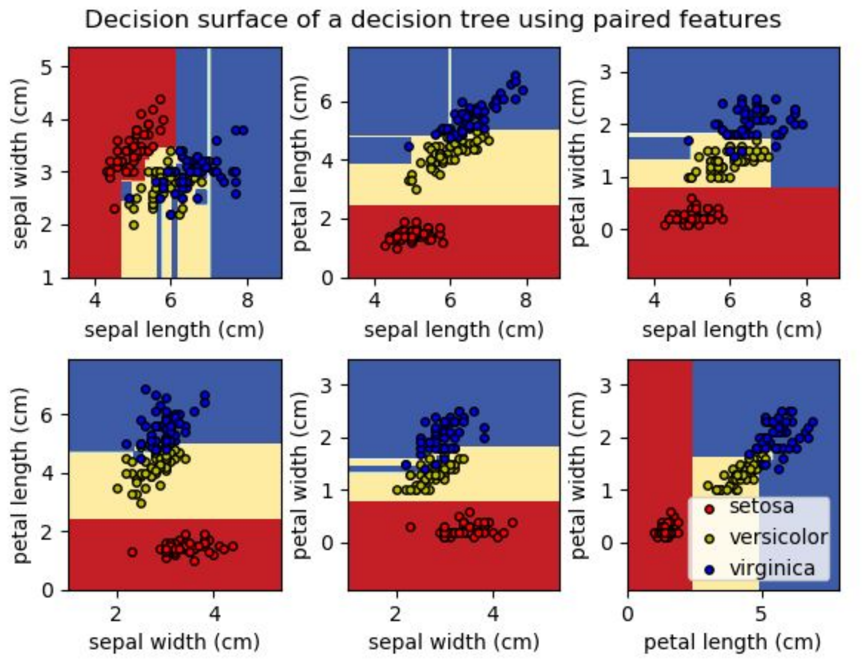
$$\hat{y}_i^{(0)} = 0$$
$$\hat{y}_i^{(1)} = f_1(x_i) = \hat{y}_i^{(0)} + f_1(x_i)$$
$$\hat{y}_i^{(2)} = f_1(x_i) + f_2(x_i) = \hat{y}_i^{(1)} + f_2(x_i)$$
$$\dots$$
$$\hat{y}_i^{(t)} = \sum_{k=1}^{t} f_k(x_i) = \hat{y}_i^{(t-1)} + f_t(x_i)$$



Limitations of decision trees

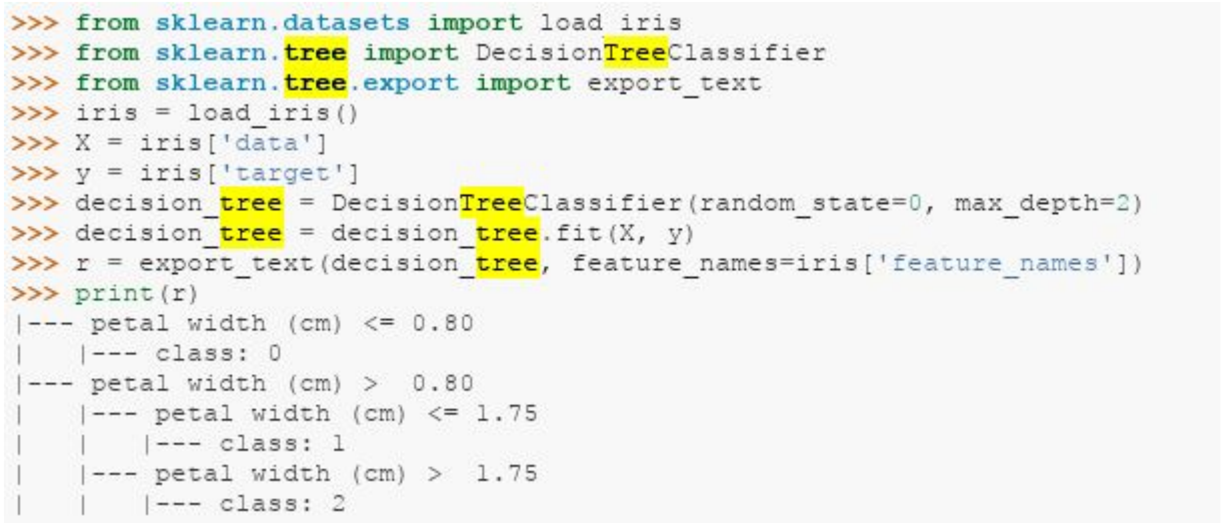
- Cuts are axis aligned
- Classification of ***x1 > x2*** is a hard problem for a decision tree





Decision trees tools

- Very powerful tools (e.g. the “workhorse” for classification/regression at the LHC)
- Various tools exists for using decision trees in python
    - In python environment: XGBoost, sklearn.tree
    - In ROOT libraries the “TMVA” package supports boosted decision trees (BDT)

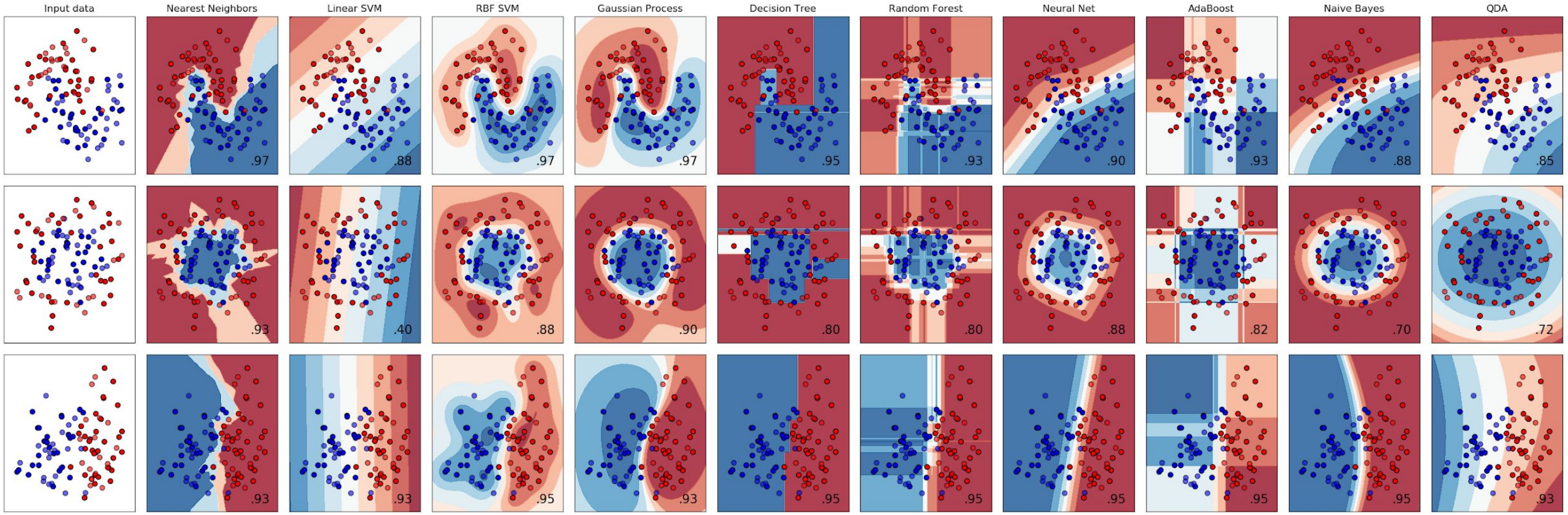


```
>>> from sklearn.datasets import load_iris
>>> from sklearn.tree import DecisionTreeClassifier
>>> from sklearn.tree.export import export_text
>>> iris = load_iris()
>>> X = iris['data']
>>> y = iris['target']
>>> decision_tree = DecisionTreeClassifier(random_state=0, max_depth=2)
>>> decision_tree = decision_tree.fit(X, y)
>>> r = export_text(decision_tree, feature_names=iris['feature_names'])
>>> print(r)
|--- petal width (cm) <= 0.80
|   |--- class: 0
|--- petal width (cm) >  0.80
|   |--- petal width (cm) <= 1.75
|   |   |--- class: 1
|   |--- petal width (cm) >  1.75
|   |   |--- class: 2
```

Many more ML techniques!

Scikit-learn library offers many ML techniques implementation in python

https://scikit-learn.org/stable/auto_examples/classification/plot_classifier_comparison.html#sphx-glr-auto-examples-classification-plot-classifier-comparison-py

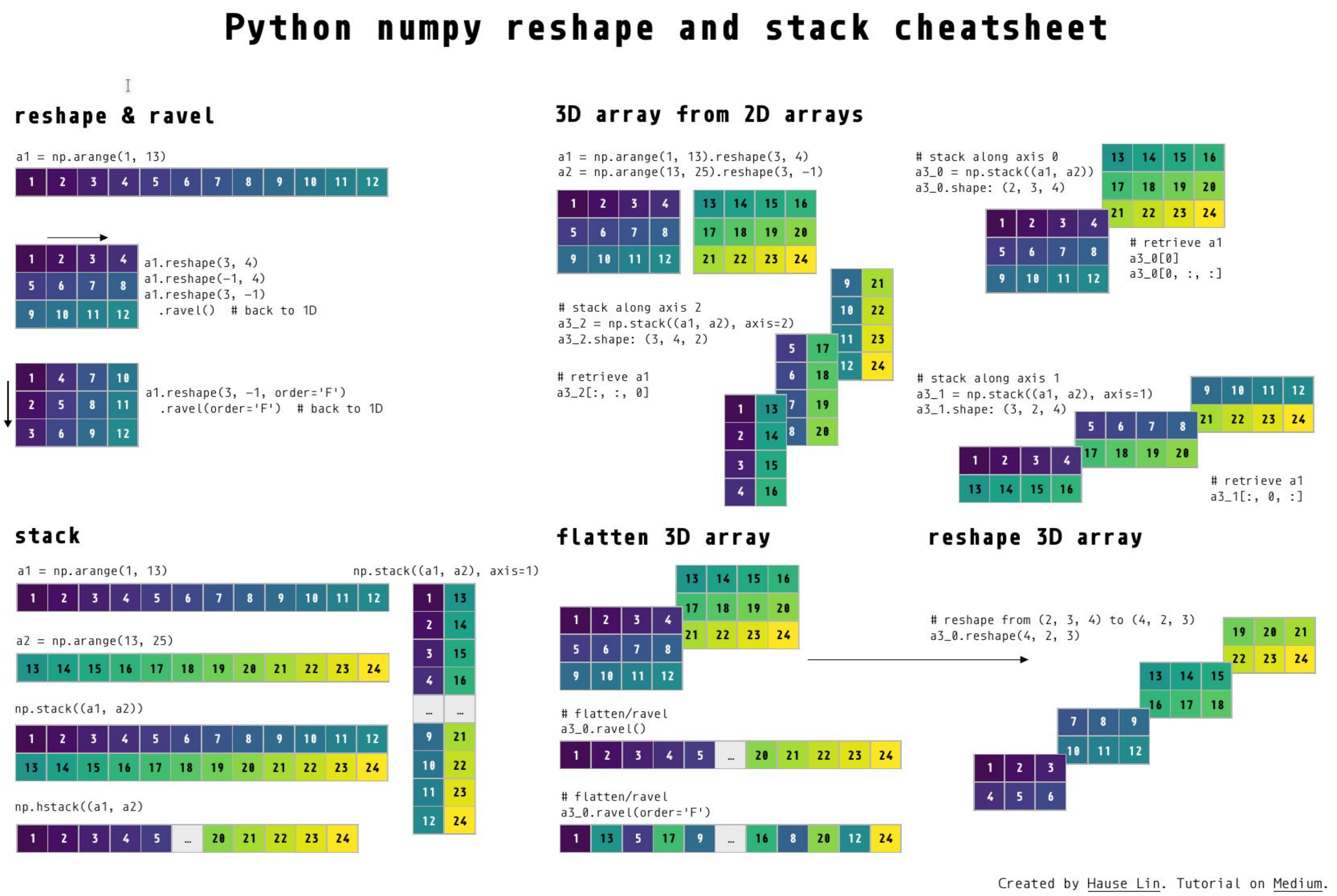


Today hands on session

In the next weeks we will use "colab" from google to run py notebooks

First exercise is taken from *Python Data Science Handbook by Jake VanderPlas* with some minor edits *(the content is available on GitHub. The text is released un the CC-BY-NC-ND license, and code is released under the MIT license. If you find this content useful, please consider supporting the work by buying the book!)*

Click here and "make a copy" to be able to edit:
https://colab.research.google.com/drive/1Sqn5fuiB5-2EP6UKUmwqjQd_b3uUNu2r?usp=sharing



Python numpy reshape and stack cheatsheet

reshape & ravel

```
a1 = np.arange(1, 13)
```

```
a1.reshape(3, 4)
a1.reshape(-1, 4)
a1.reshape(3, -1)
  .ravel()  # back to 1D
```

```
a1.reshape(3, -1, order='F')
  .ravel(order='F')  # back to 1D
```

3D array from 2D arrays

```
a1 = np.arange(1, 13).reshape(3, 4)
a2 = np.arange(13, 25).reshape(3, -1)
```

```
# stack along axis 0
a3_0 = np.stack((a1, a2))
a3_0.shape: (2, 3, 4)
```

```
# retrieve a1
a3_0[0]
a3_0[0, :, :]
```

```
# stack along axis 2
a3_2 = np.stack((a1, a2), axis=2)
a3_2.shape: (3, 4, 2)
```

```
# retrieve a1
a3_2[:, :, 0]
```

```
# stack along axis 1
a3_1 = np.stack((a1, a2), axis=1)
a3_1.shape: (3, 2, 4)
```

```
# retrieve a1
a3_1[:, 0, :]
```

stack

```
a1 = np.arange(1, 13)
```

```
a2 = np.arange(13, 25)
```

```
np.stack((a1, a2))
```

```
np.hstack((a1, a2)
```

```
np.stack((a1, a2), axis=1)
```

flatten 3D array

```
# flatten/ravel
a3_0.ravel()
```

```
# flatten/ravel
a3_0.ravel(order='F')
```

reshape 3D array

```
# reshape from (2, 3, 4) to (4, 2, 3)
a3_0.reshape(4, 2, 3)
```

Created by Hause Lin. Tutorial on Medium.

https://towardsdatascience.com/reshaping-numpy-arrays-in-python-a-step-by-step-pictorial-tutorial-aed5f471cf0b