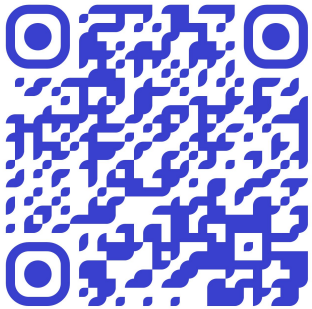
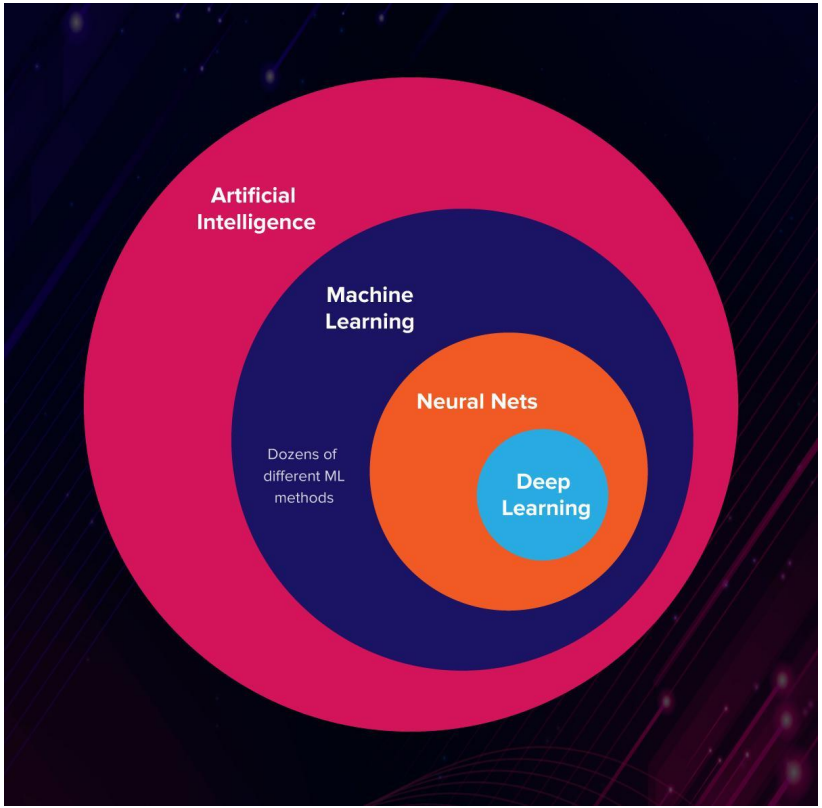
Intro to ML

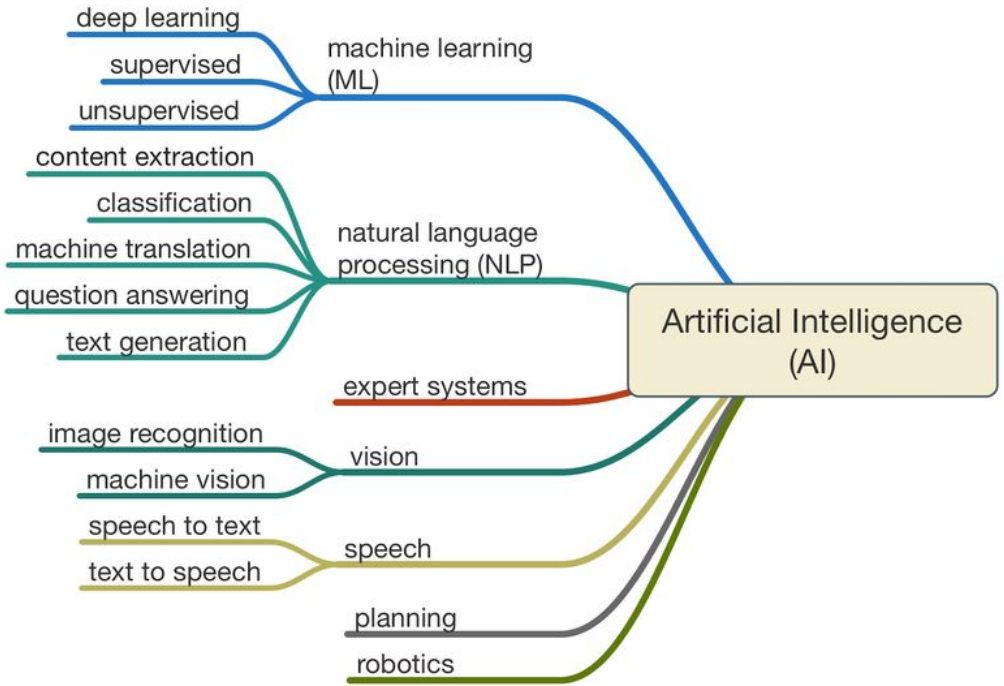


Serafina Di Gioia (postdoc @ ICTP)
10/09/2025

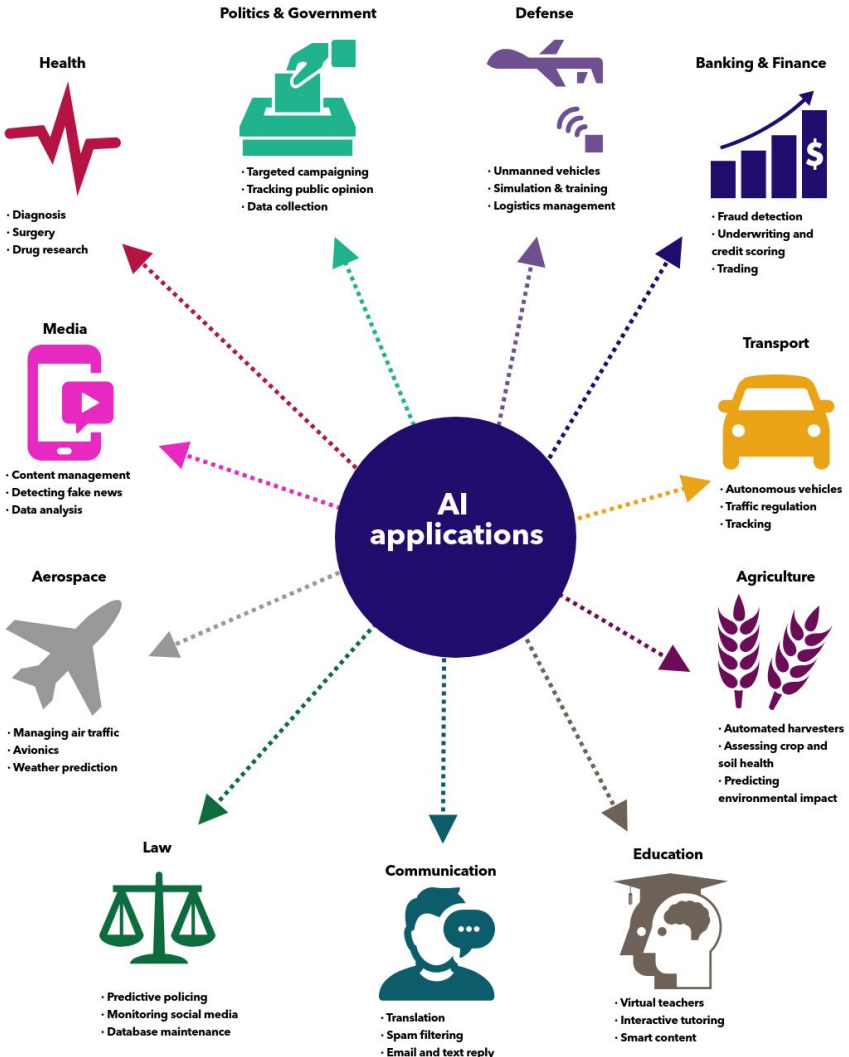
What is AI?

AI is a field of research aimed at developing intelligent entities or systems capable of replicating human-like cognition and behavior





What kind of task are solved by AI systems today?

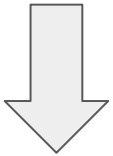


What is ML?

in 1960

use and development of computer systems that are able to learn and adapt without following explicit instructions

shift in perspective



in 2023

subfield of AI that uses algorithms trained on data to produce adaptable models that can perform a variety of complex tasks (e.g. classification, regression and forecasting) without the need to code specific instructions for different tasks

Books on ML

Bishop, Pattern recognition and ML

Elements of Statistical Learning, Springer, 2009, by Hastie et al.

Hands-On Machine Learning with Scikit-Learn and TensorFlow,
by Aurelien Geron

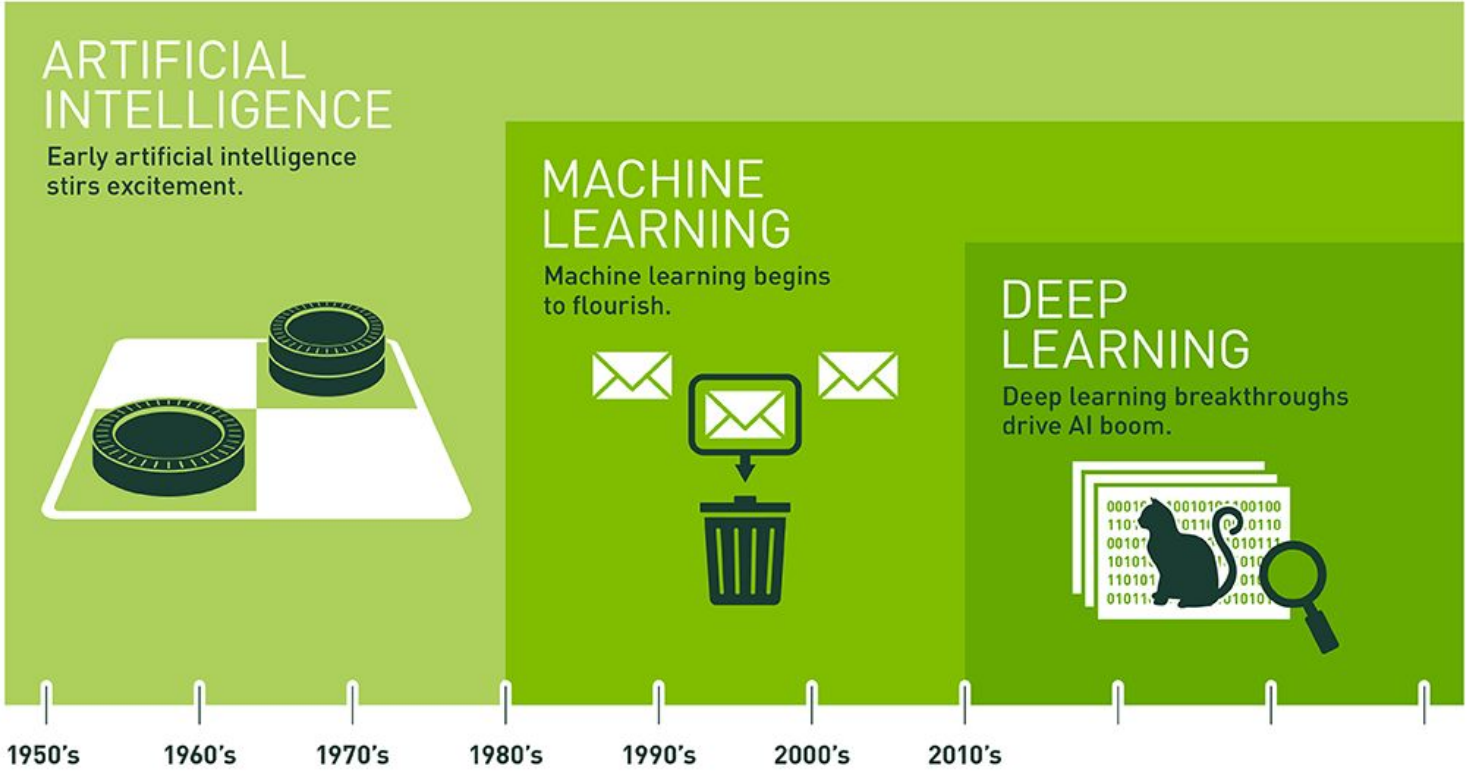
Deep Learning,
by Ian Goodfellow and Yoshua Bengio and Aaron Courville

(free online http://www.deeplearningbook.org/)

Why ML for weather/climate?

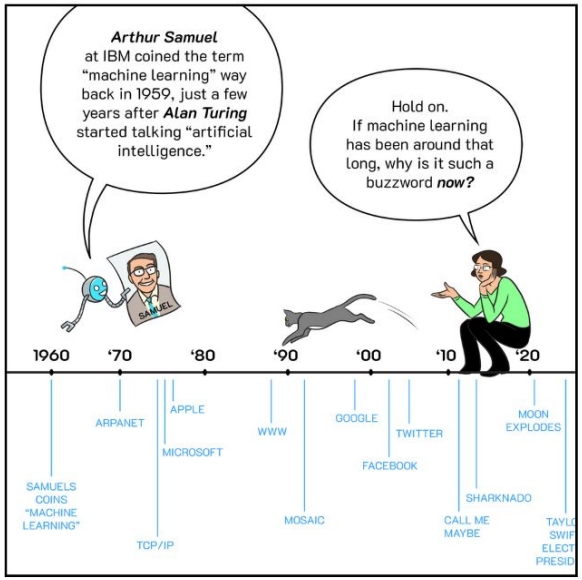
- Forecasting
- Downscaling
- Anomaly detection
- Data assimilation
- Hybrid climate modeling (equation discovery, faster simulations)
- Optimizing the parametrization of climate simulations

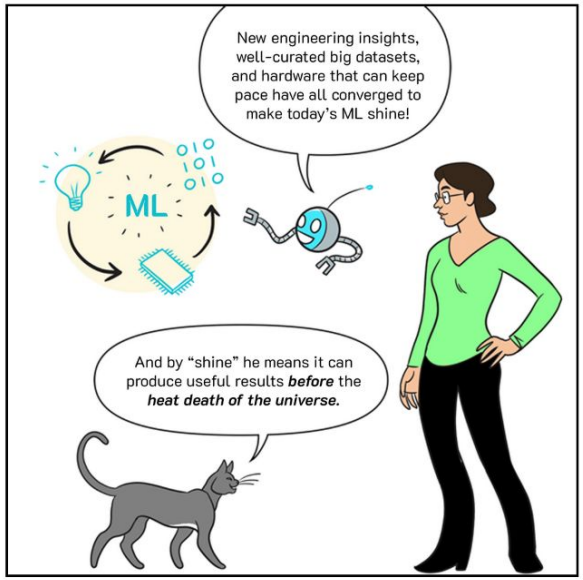
From LISP to the DL revolution…

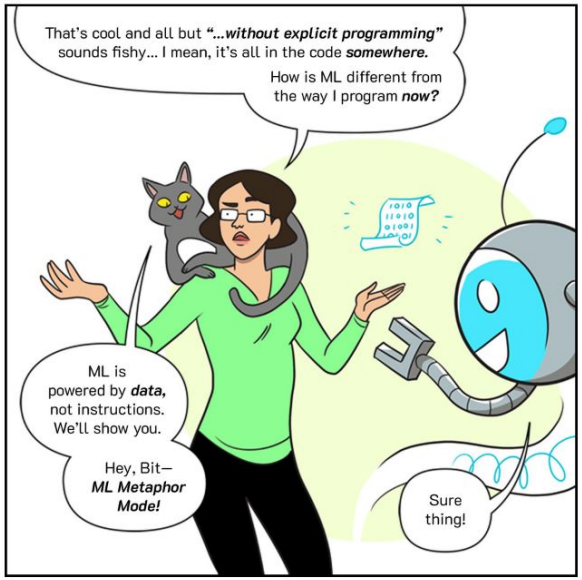


Since an early flush of optimism in the 1950s, smaller subsets of artificial intelligence – first machine learning, then deep learning, a subset of machine learning – have created ever larger disruptions.

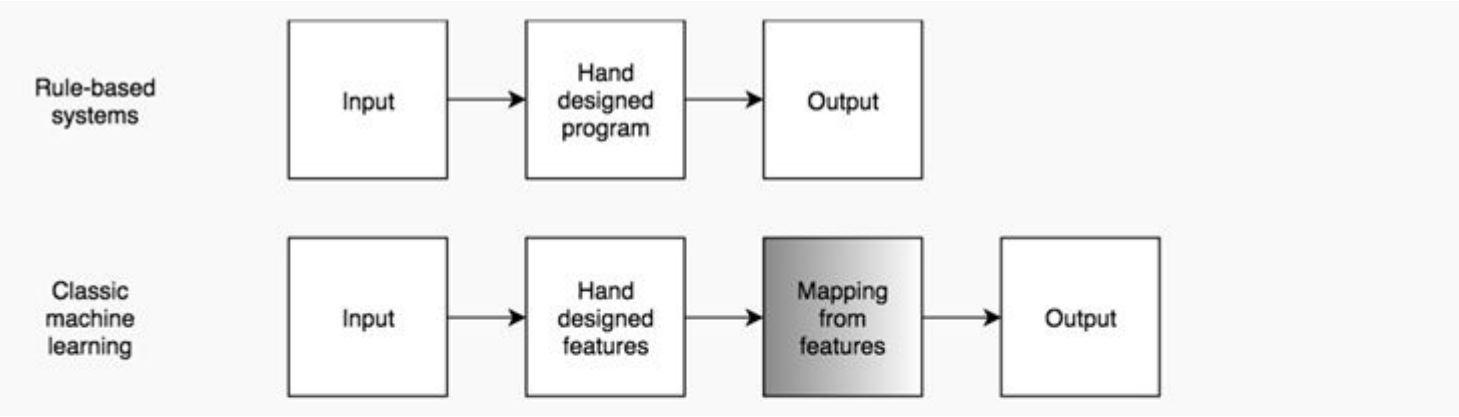
History of ML



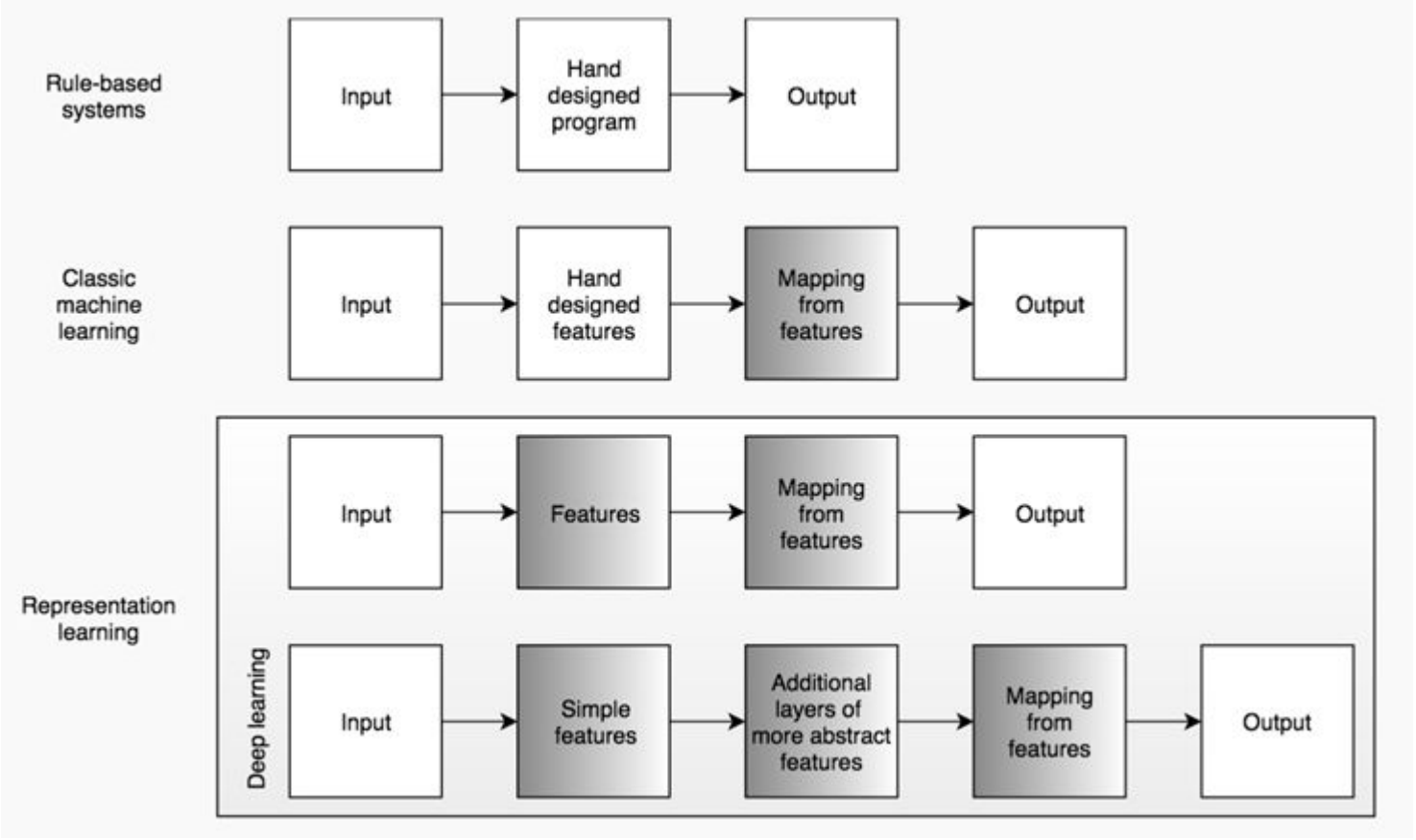




ML vs classical explicit programming



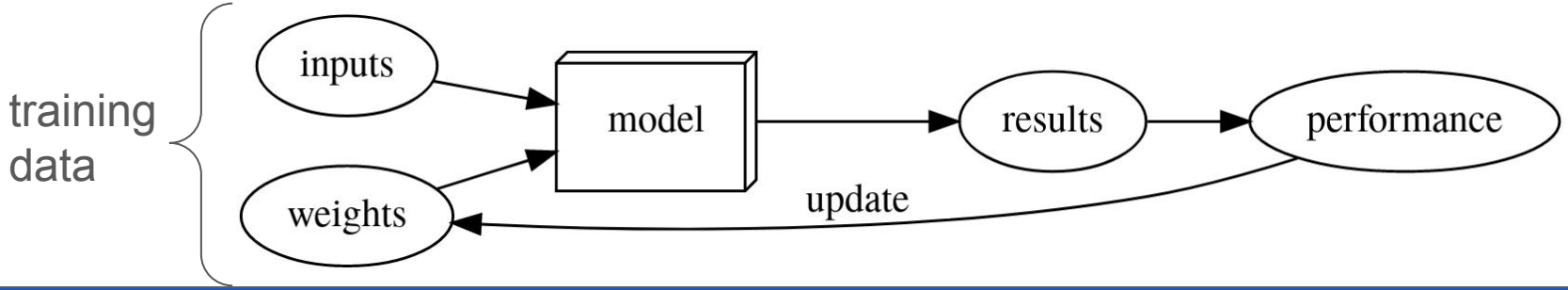
DL vs ML vs classical explicit programming



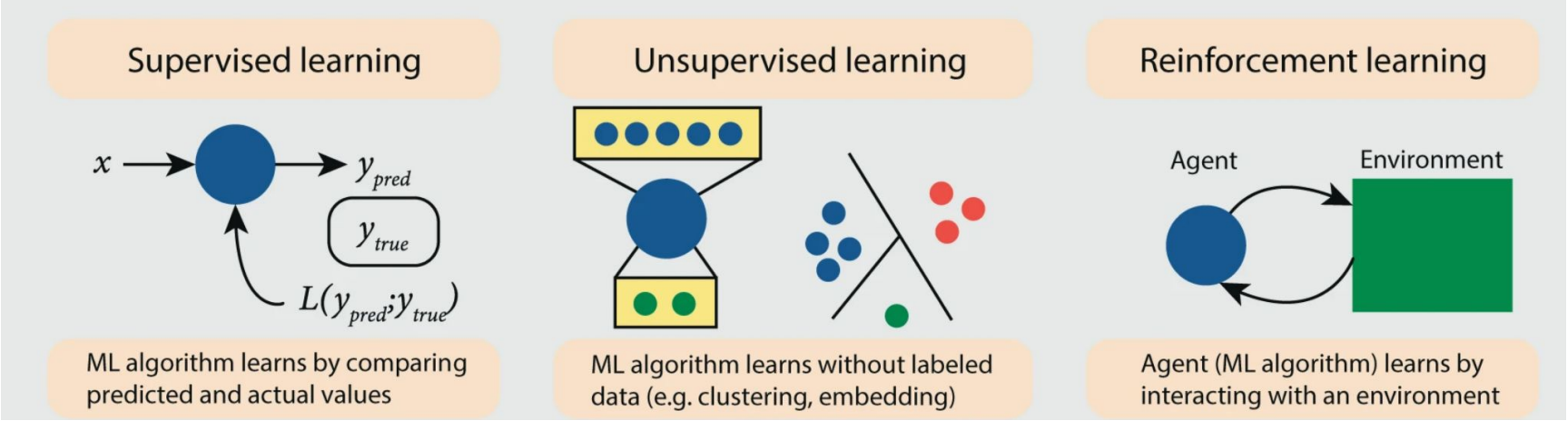
Building blocks of ML algorithms

ML algorithms have three main components

1. **decision process**: based on some input data, which can be labeled or unlabeled, your algorithm will produce an estimate about a pattern in the data. This estimate can be used to solve a regression or classification task
2. **error function**: it evaluates the prediction of the model. If there are known examples, an error function can make a comparison to assess the accuracy of the model.
3. **Model Optimization Process (training)**: If the model can fit better to the data points in the training set, then weights are adjusted to reduce the discrepancy between the known example and the model estimate. The algorithm will repeat this “evaluate and optimize” process, updating weights autonomously until a threshold of accuracy has been met.



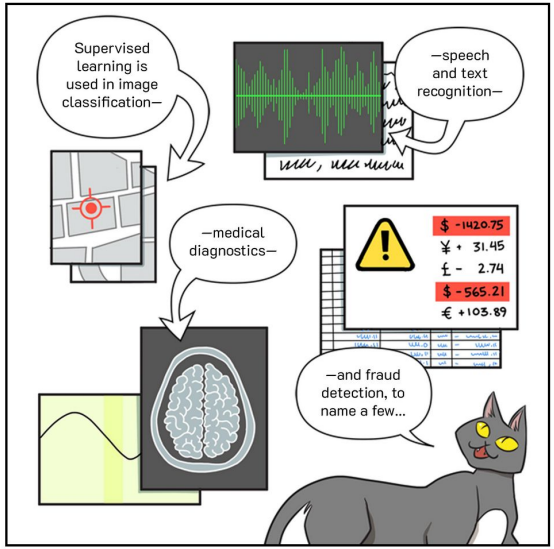
ML learning frameworks



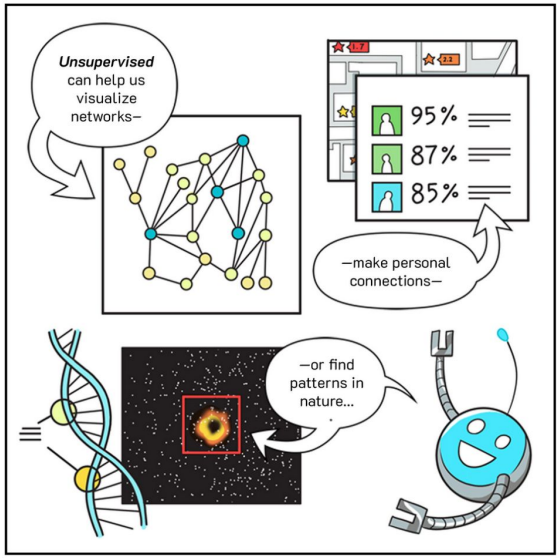
Learning type	Model building	Examples
Supervised	Algorithms or models learn from labeled data (task-driven approach)	Classification, regression
Unsupervised	Algorithms or models learn from unlabeled data (Data-Driven Approach)	Clustering, associations, dimensionality reduction
Semi-supervised	Models are built using combined data (labeled + unlabeled)	Classification, clustering
Reinforcement	Models are based on reward or penalty (environment-driven approach)	Classification, control

Applications of different learning methods

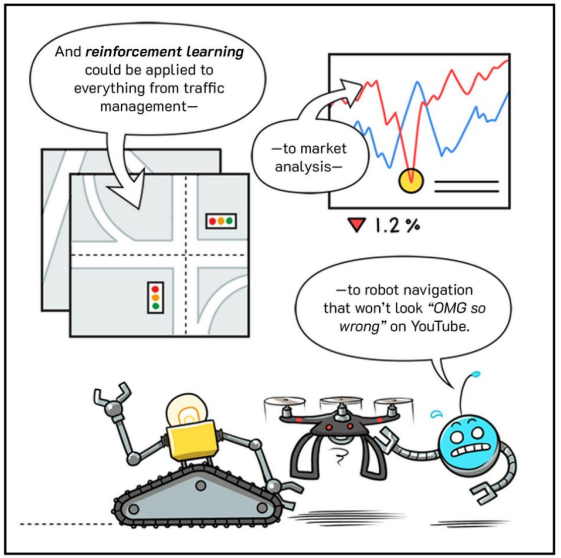
Supervised learning



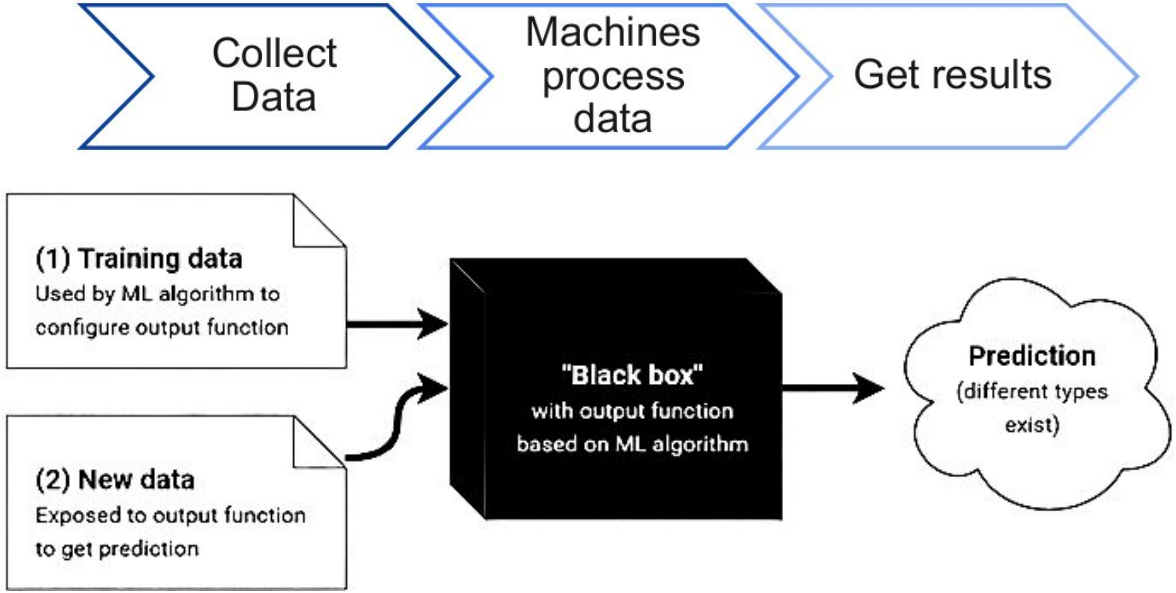
Unsupervised learning



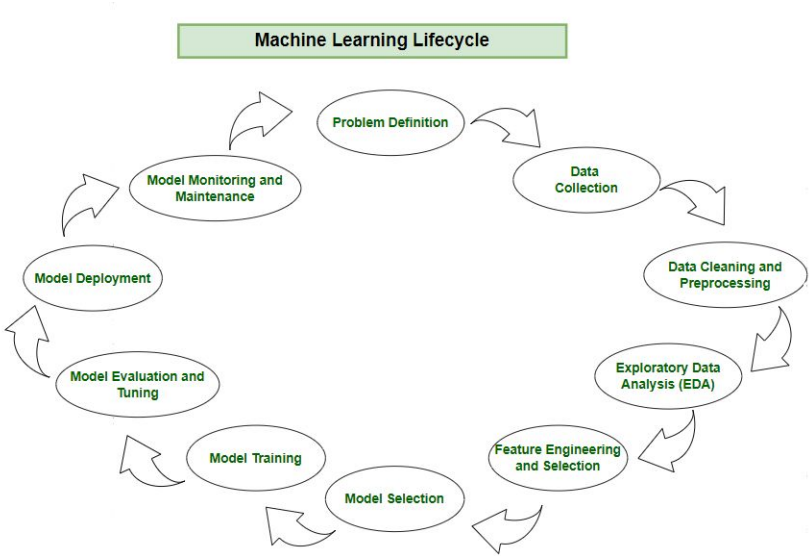
Reinforcement Learning



What non-data scientists think of ML:



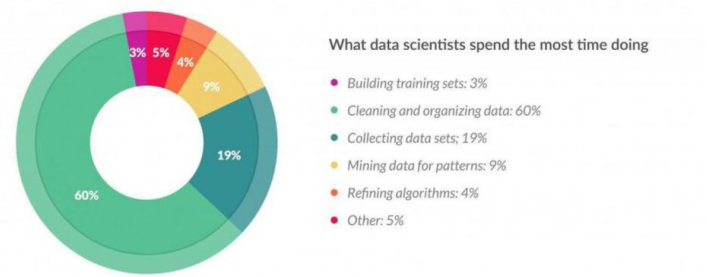
What ML is in reality?



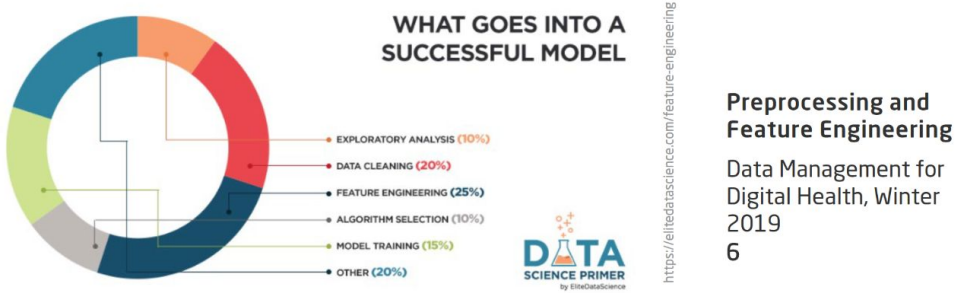
1. Problem definition
2. Data collection and exploration
3. Data preparation
4. Modeling (model selection & training)
5. Model evaluation
6. Model deployment and maintenance

The importance of Exploratory data analysis…

From a survey compiled by ML scientists some years ago:



Source: https://www.forbes.com/sites/gilpress/2016/03/23/data-preparation-most-time-consuming-least-enjoyable-data-science-task-survey-says/



Preprocessing and Feature Engineering

Data Management for Digital Health, Winter 2019

6

What are data?

for a science communicator

Factual information (such as measurements or statistics) used as a basis for reasoning, discussion, or calculation

for a ML engineer

for a scientist

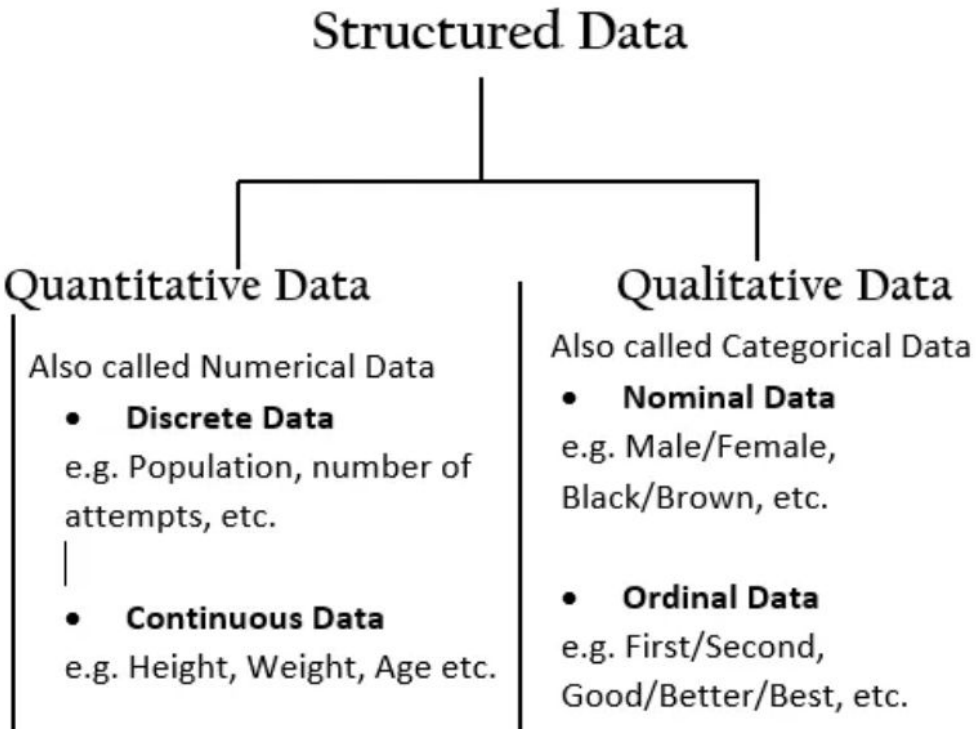
What are data?

for a science communicator	Factual information (such as measurements or statistics) used as a basis for reasoning, discussion, or calculation
for a ML engineer	Information in digital form that can be transmitted or processed
for a scientist	

What are data?

for a science communicator	Factual information (such as measurements or statistics) used as a basis for reasoning, discussion, or calculation
for a ML engineer	Information in digital form that can be transmitted or processed
for a scientist	Information output by a sensing device or organ that includes both useful and irrelevant or redundant information

Data type

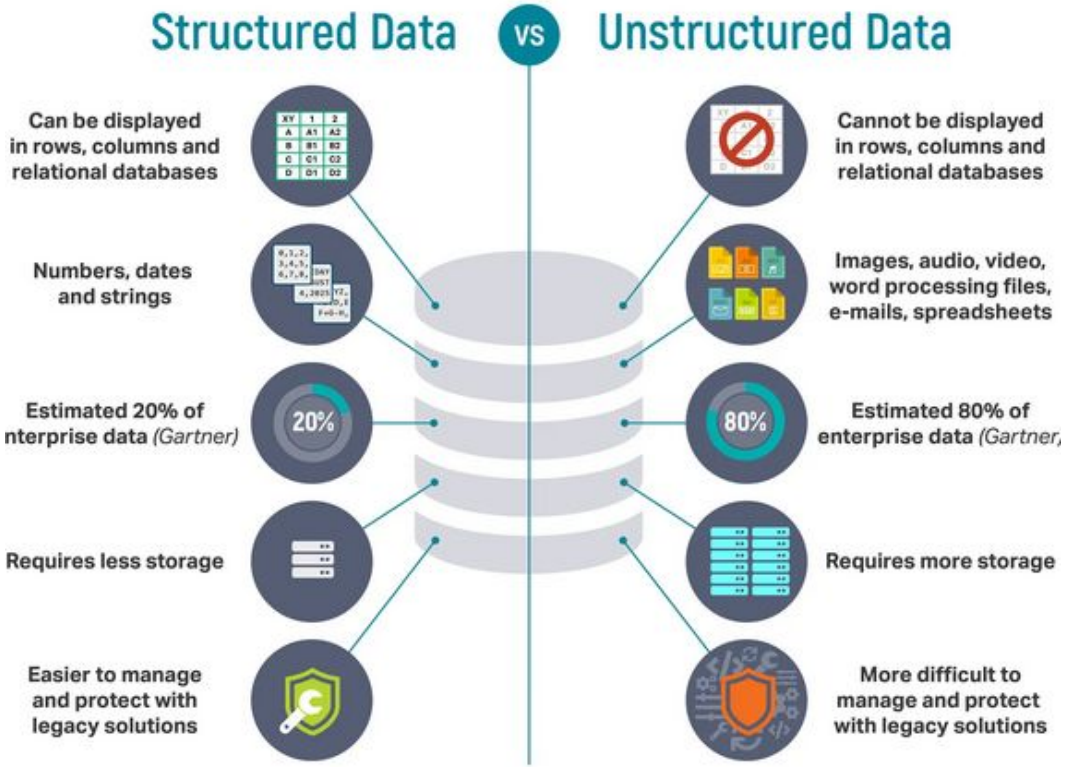


Unstructured Data

Data that doesn't follow a pattern or sequence is called an unstructured data.
e.g. Audio, Video, Image.

Properties of structured vs Unstructured data

also called tabular data



Seaborn

How to install it:

```
pip install seaborn[stats]
```

```
conda install seaborn -c conda-forge
```

Documentation at:

https://seaborn.pydata.org/api.html

Pandas

How to install it:

```
pip install pandas
```

```
conda install -c conda-forge pandas
```

Documentation at:

https://pandas.pydata.org/docs/getting_started/index.html#getting-started

scikit-learn

How to install it:

```
pip install scikit-learn
```

```
conda install -c scikit-learn-intelex
```

Documentation at:

https://scikit-learn.org/stable/user_guide.html

Data containers in Pandas

Series

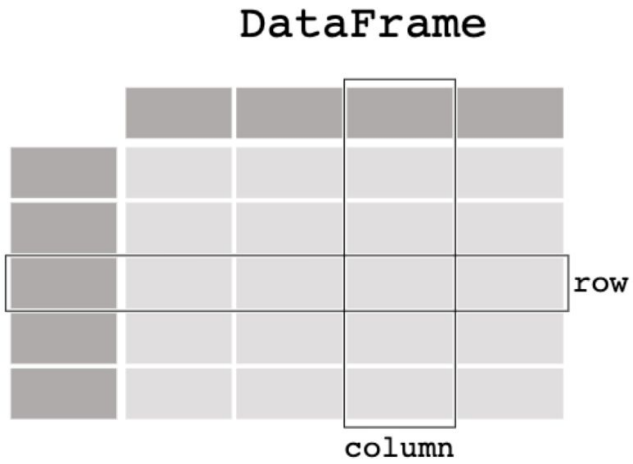
A Series is one-dimensional array-like object containing an array of data (of any NumPy data type) and an associated array of data-labels, called its index.

Dataframe

Tabular, spreadsheet-like data structure containing an ordered collection of columns of potentially different value types (numeric, string, etc.)

it can be regarded as a dict of Series

What a Dataframe looks like…

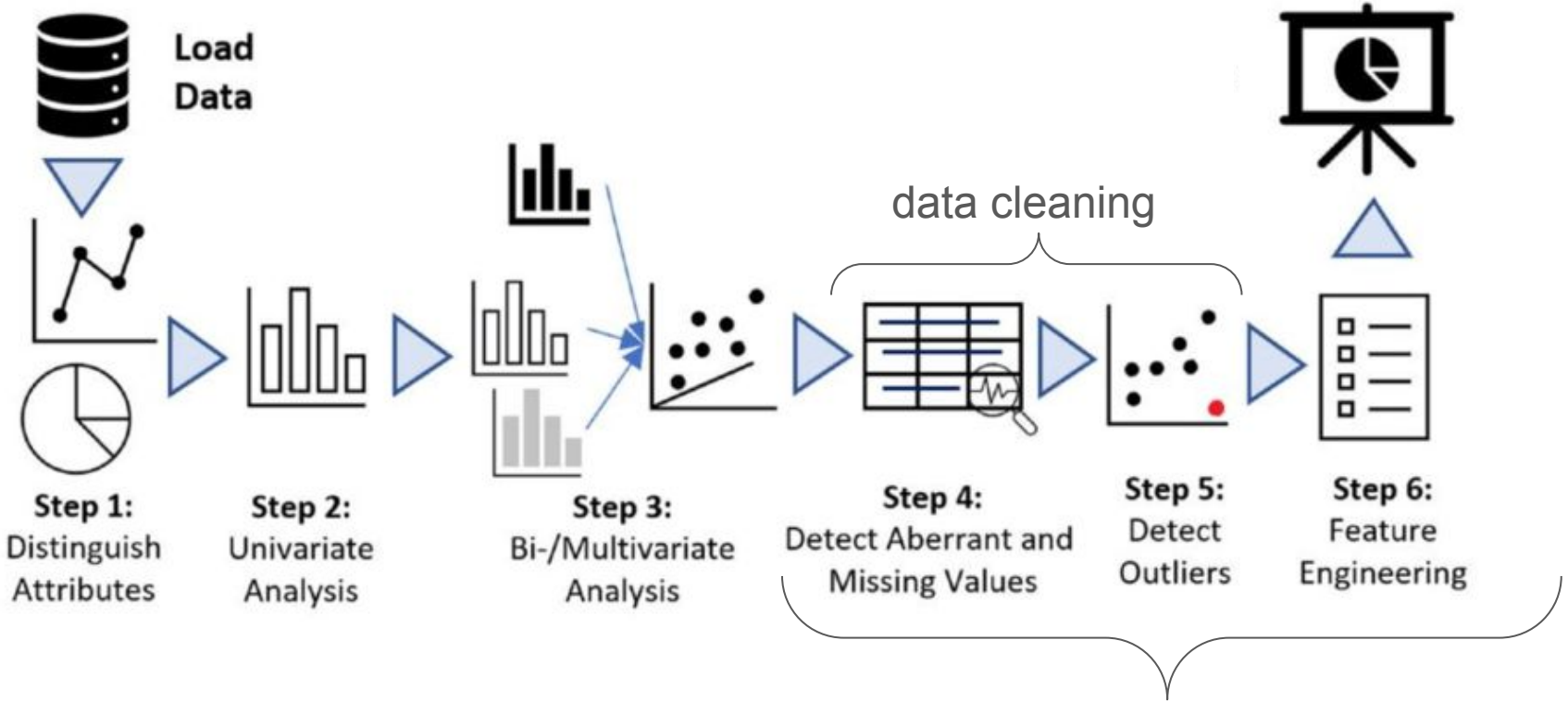


Visit https://pandas.pydata.org/pandas-docs/stable/getting_started/intro_tutorials/01_table_oriented.html for a more in-depth walkthrough

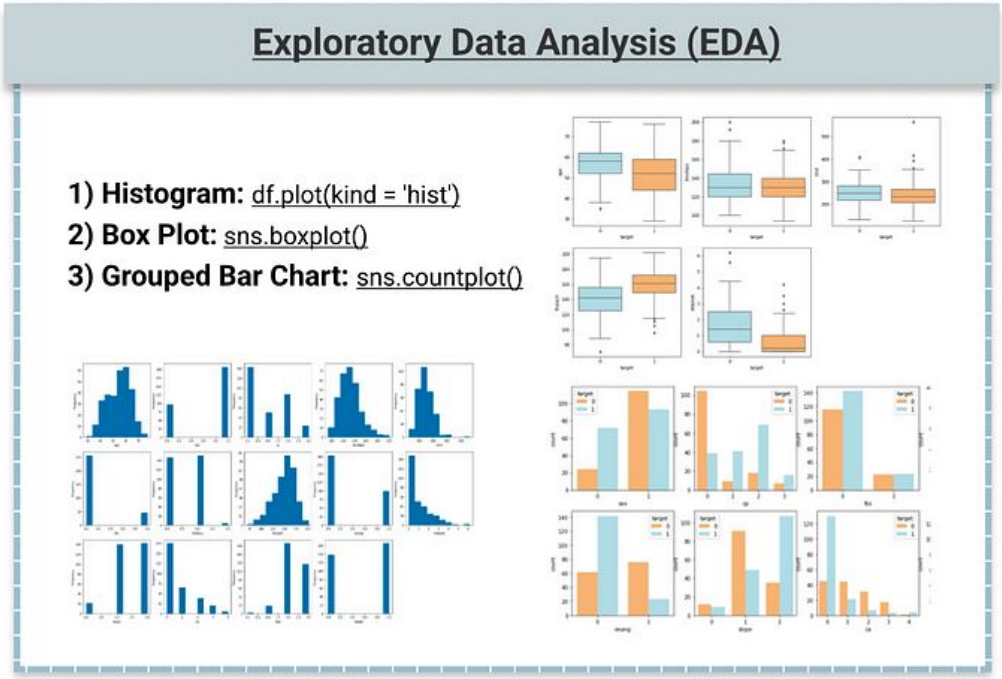
High-level view of dataframes in Pandas

- head() – first N observations
- tail() – last N observations
- describe() – statistics of the quantitative data
- dtypes – the data types of the columns
- columns – names of the columns
- shape – the # of (rows, columns)

Exploratory Data Analysis (EDA)



Step 0-1: Distinguish attributes and visualizing the data



Step 2-3 : Statistical analysis

Univariate analysis:

- ❏ histograms/kDE
- ❏ summary statistics

to answer:

Questions

- What is the central tendency of the data (mean, median, mode)?
- What is the spread of the data (range, interquartile range, standard deviation)?
- What is the shape of the distribution (symmetrical, skewed)?

Multivariate analysis:

- ❏ pair correlation
- ❏ multivariate joint distribution properties

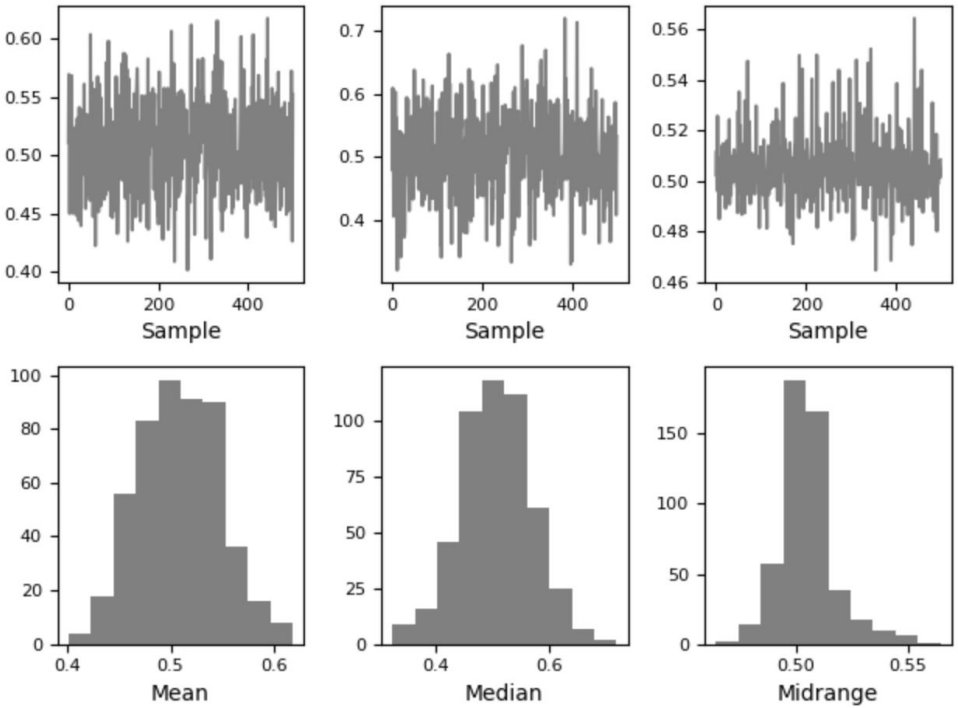
How can we visualize uncertainty on summary statistics (e.g. mean)?

Using bootstrap…

```
In [115]: from pandas.plotting import bootstrap_plot

In [116]: data = pd.Series(np.random.rand(1000))

In [117]: bootstrap_plot(data, size=50, samples=500, color="grey");
```



What is a Bootstrap sampling?

Initial full dataset Z

Bootstrap samples $Z_i, i = 1, \ldots, m$ of size $B = 5$

Bootstrap samples are expected to **approximately** be **representative and independent samples** of the true data distribution (almost $i.i.d.$)

- For n big enough, sampling from Z is like sampling from the true data distribution
- If $n >> B$ the samples should not be much correlated

We are thus assuming to fit and then average m independent models…

Step 4-5 : Data wrangling

It includes the following tasks:

- ❏ reformatting/cleaning data (outliers removal)
- ❏ data quality assessment
- ❏ data integration (imputation of missing values)
- ❏ data preprocessing

Libraries to implement data wrangling:

- Custom code (e.g. Pandas in Python, cudf, dplyr in R)

EDA is an iterative process

First you explore the data graphically

1) Construct graphics and summary statistics (e.g. mean) to explore data properties

2) Inspect “answer” and assess new questions

3) Repeat…

keeping an eye on how to transform data appropriately (e.g., normalize, log)

Finally, you perform DATA PREPROCESSING (also called feature engineering)

Feature preprocessing for continuous variables

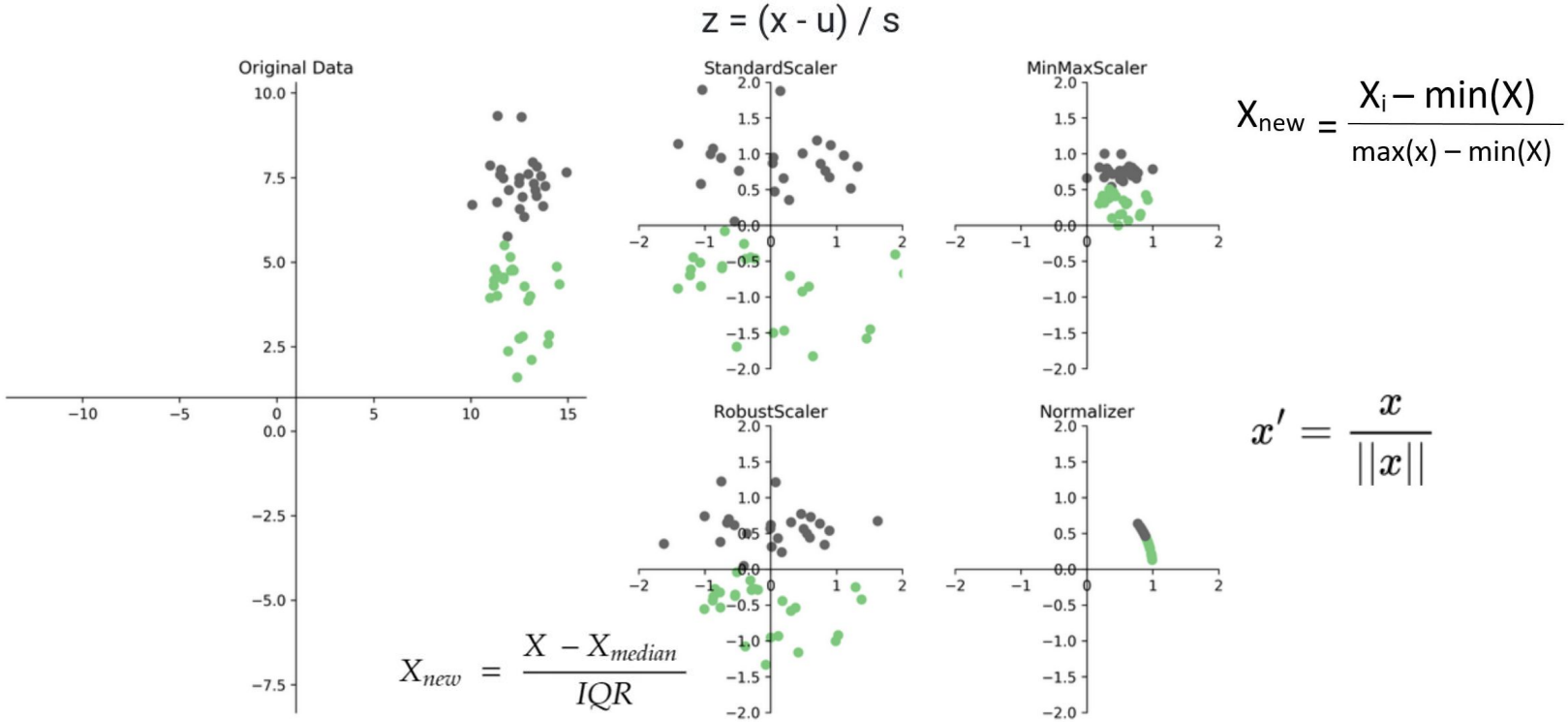
Why?

Feature scaling is essential for machine learning algorithms that calculate **distances between data**. If not scaled, the feature with a higher value range starts dominating when calculating distances

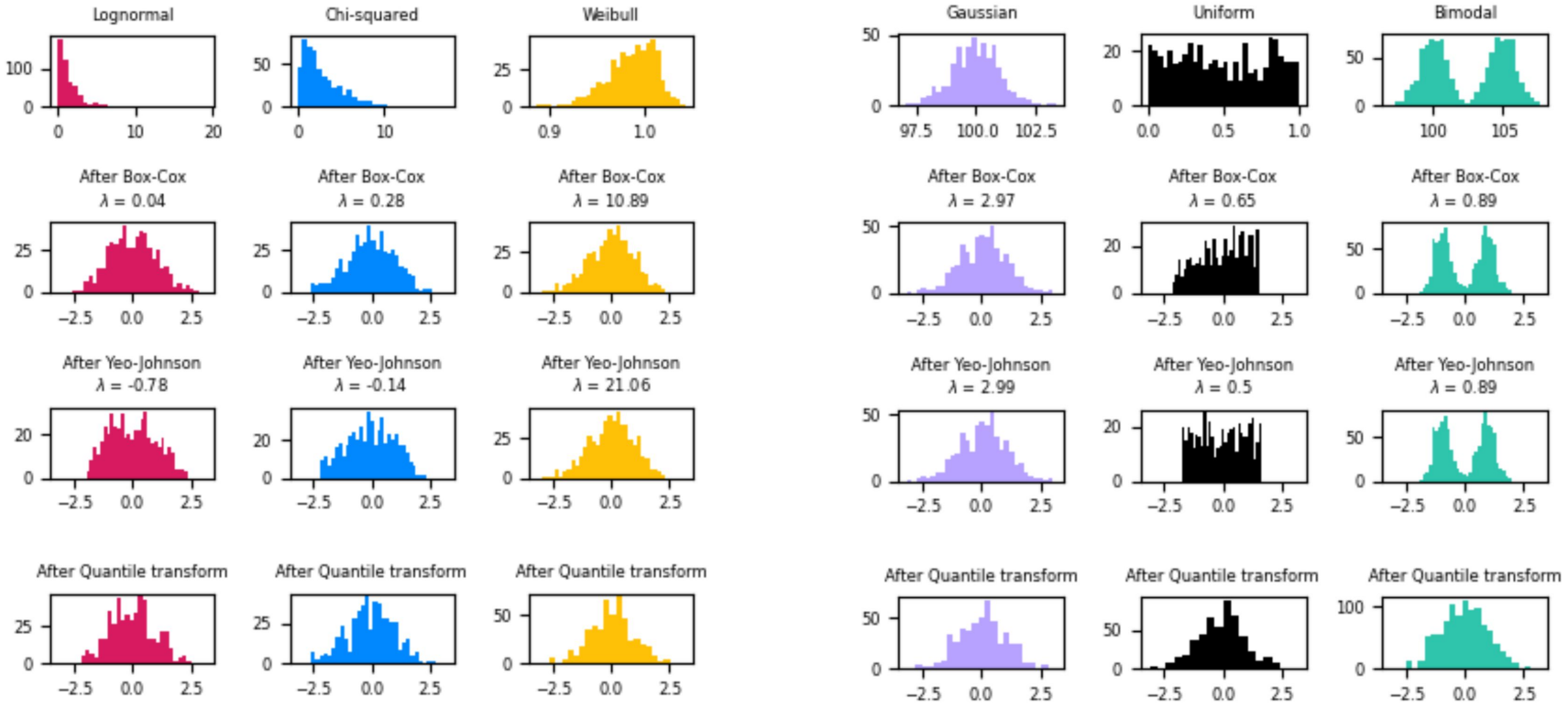
Functions in Scikit-learn

1. Min Max Scaler
2. Standard Scaler
3. Max Abs Scaler
4. Robust Scaler
5. Quantile Transformer Scaler
6. Power Transformer Scaler
7. Unit Vector Scaler

Visualizing the rescaling



Visualizing different non-linear transformation in sklearn



Feature preprocessing for categorical variables

The two most common techniques

Label encoding

- ✓ Label encoding returns different values for different classes, bringing in a natural ordering
- ✓ *Recommended when* it's reasonable to assume some sort of (equally spaced) ordering in your categorical feature

country	country
France	1
Italy	0
Spain	2

teaching_level	teaching_level
Beginner	0
Medium	1
Advanced	2

One Hot encoding

- ✓ One hot encoding returns as many dummy variables (0/1 columns) as the classes of the categorical feature
- ✓ *Recommended when* no ordering can be assumed in your categorical feature

country	country_France	country_Italy	country_Spain
France	1	0	0
Italy	0	1	0
Spain	0	0	1

EDA with Pandas + seaborn

Data I/O

pd.read_csv()

Summary statistics

df.info()

df.describe()

Missing values

.isnull(), .isna(), .notnull()

Outlier detection

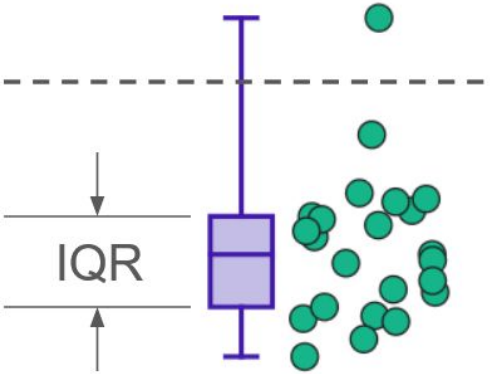
Plots

- **Bar charts** - compare different categories.
- **Line charts -** show trends over time or across different categories.
- **Pie charts -** show proportions or percentages of different categories.
- **Histograms -** show the distribution of a single variable.
- **Heatmaps -** show the correlation between different variables.
- **Scatter plots -** show the relationship between two continuous variables.
- **Box plots -** show the distribution of a variable and identify outliers.
- **Violin plots -** check normality assumption

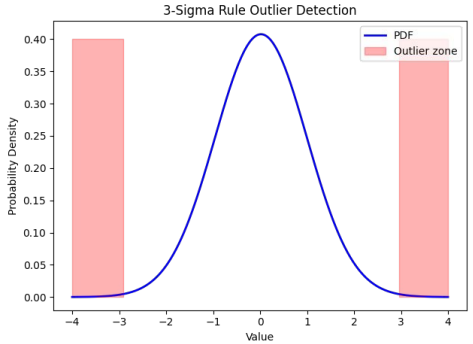
Outlier detection

there are 2 common methods:

- 1.5 IQR rule
- 3 sigma rule



IQR= Q3 - Q1

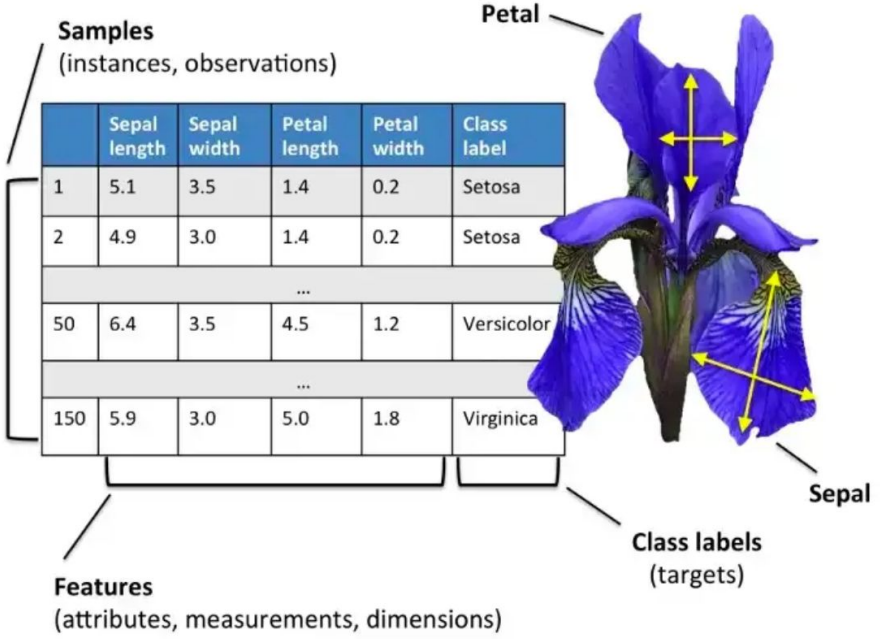


Iris dataset

small classical ML dataset

first work:
Fisher, 1936

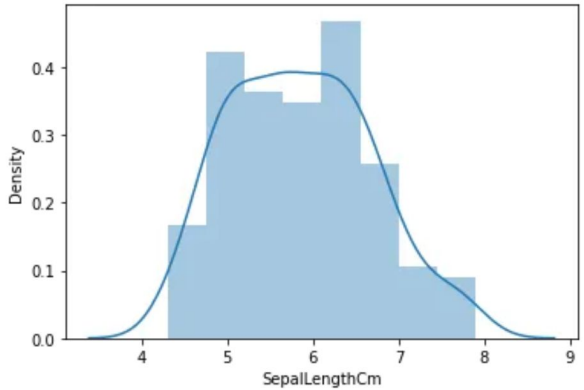
UCI repo:
https://archive.ics.uci.edu/dataset/53/iris



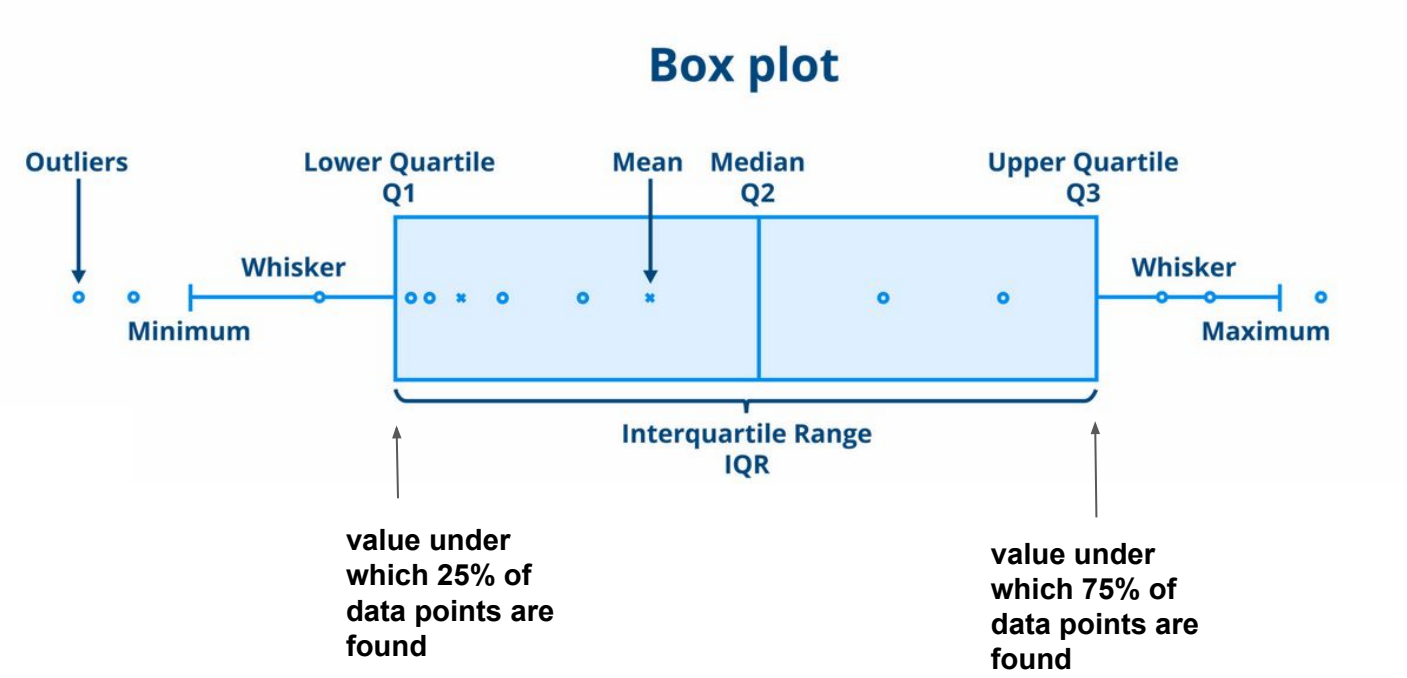
	Sepal length	Sepal width	Petal length	Petal width	Class label
1	5.1	3.5	1.4	0.2	Setosa
2	4.9	3.0	1.4	0.2	Setosa
...					
50	6.4	3.5	4.5	1.2	Versicolor
...					
150	5.9	3.0	5.0	1.8	Virginica

Univariate analysis on Iris data - Distplot

- What is the central tendency of the data (mean, median, mode)?
- What is the spread of the data (range, interquartile range, standard deviation)?
- What is the shape of the distribution (symmetrical, skewed)?

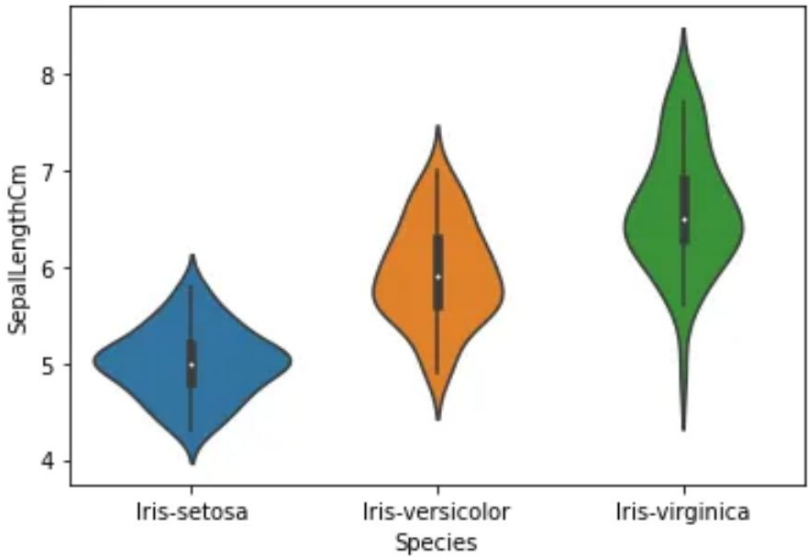


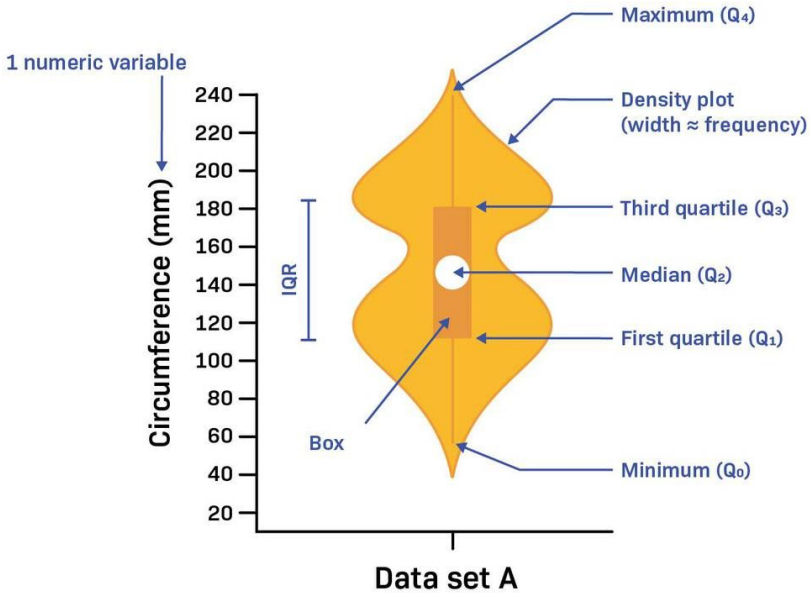
Boxplots


Box plot
Outliers
Lower Quartile
Q1
Mean
Median
Q2
Upper Quartile
Q3
Whisker
Whisker
Minimum
Maximum
Interquartile Range
IQR
value under which 25% of data points are found
value under which 75% of data points are found

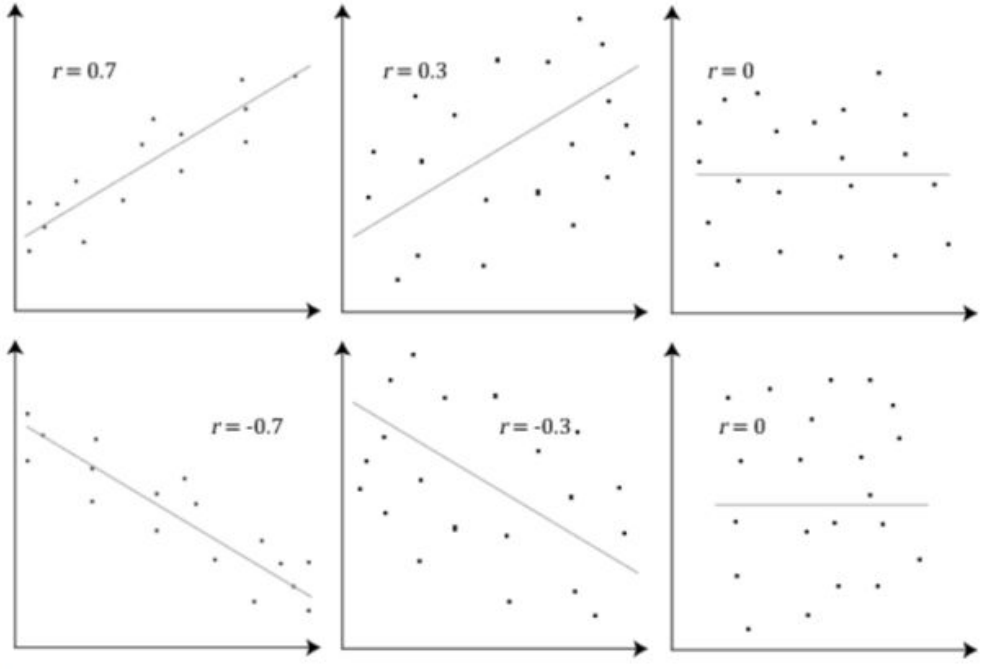
Violin plot

```
# Plotting the violinplot
sns.violinplot(x='Species', y='SepalLengthCm', data=data)
plt.show()
```





Pearson Correlation



Pearson correlation coefficient

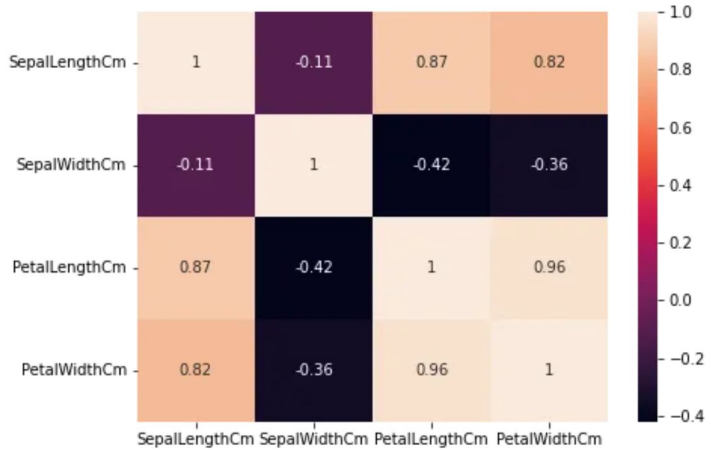
$$r = \frac{1}{n-1} \sum_{i=1}^{n} \left(\frac{x_i - \bar{x}}{s_x} \right) \left(\frac{y_i - \bar{y}}{s_y} \right)$$

Z-score

$$-1 < r(x, y) < 1$$

Plots for Feature association analysis: heatmap

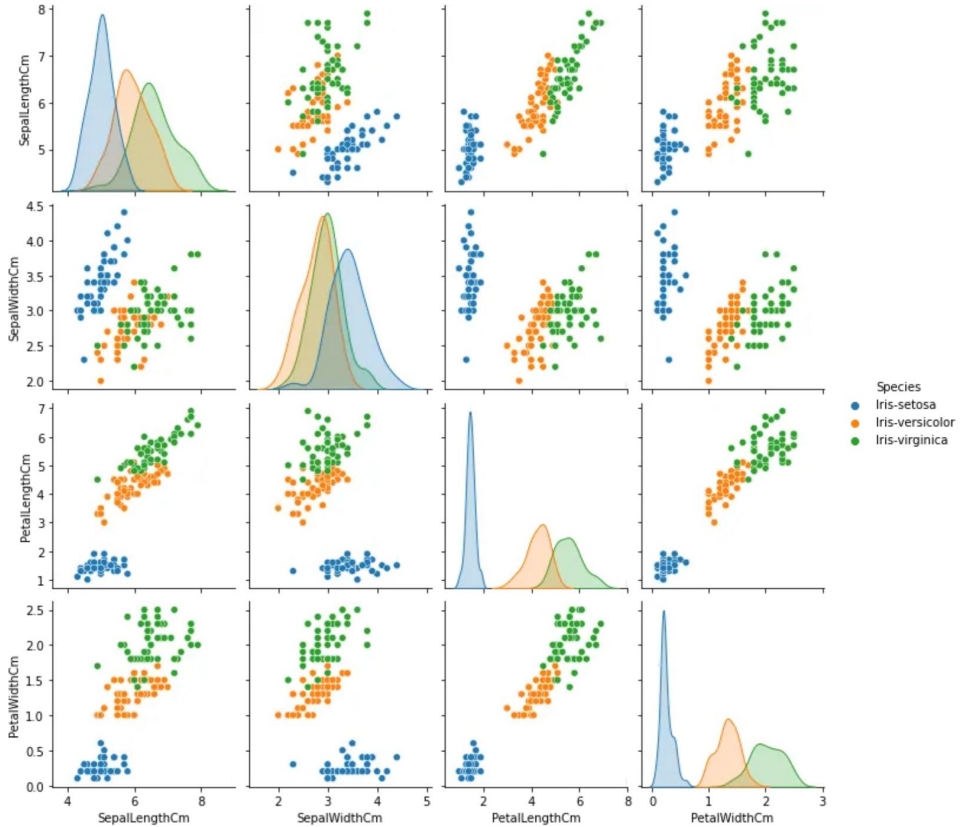
```
plt.figure(figsize=(7,5))
# Plotting the heatmap
sns.heatmap(data.corr(), annot=True)
plt.show()
```



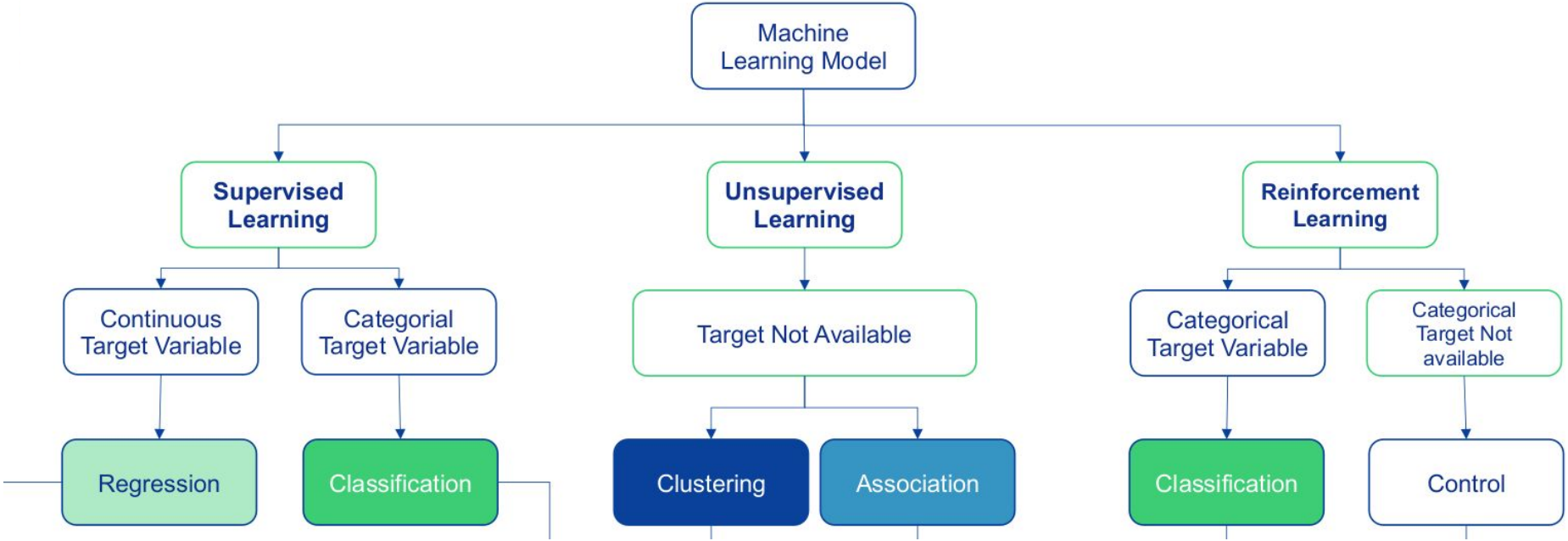
Multivariate analysis -> Pair plot (with Seaborn)

it shows joint and marginal distributions for all pairwise relationships and for each variable, respectively

```
sns.pairplot(data,hue="Species")
plt.show()
```



Traditional ML scenarios



Supervised learning framework

- Training : $\{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \ldots, (\boldsymbol{x}_n, y_n)\}$
- $\boldsymbol{x}_i$: input vector

$$\boldsymbol{x}_i = \begin{bmatrix} x_{i,1} \\ x_{i,2} \\ \vdots \\ x_{i,n} \end{bmatrix}, \quad x_{i,j} \in \mathbb{R}$$

- y : response variable
  - $y \in \{-1, 1\}$: binary classification
  - $y \in \mathbb{R}$: regression
  - what we want to be able to predict, having observed some new $\boldsymbol{x}$.

The supervised learning problem

Problem: Given a set of examples (training set)

$$\mathcal{T} = \left\{ (x^{(i)}, y^{(i)}), i = 1, \ldots, n \right\} \qquad x^{(i)} \in \mathbb{R}^p$$

we wish to estimate a function

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

such that $\hat{y}$ (predictor) is, in some sense, close to y.

Theoretical VS Empirical Risk

The theoretical risk of a candidate model f is the expected loss

$$\mathcal{R}(f) := \mathbb{E}_{xy}[L(y, f(\mathbf{x}))] = \int L(y, f(\mathbf{x}))\, \mathrm{d}\mathbb{P}_{xy}$$

- Average error we incur when we use f on data from $\mathbb{P}_{xy}$
- Goal in ML: Find a hypothesis $f \in \mathcal{H}$ that **minimizes** this

To estimate the empirical risk we
Just sum up all losses over training data

$$\mathcal{R}_{\text{emp}}(f) = \sum_{i=1}^{n} L\left(y^{(i)}, f\left(\mathbf{x}^{(i)}\right)\right)$$

Supervised learning as optimization problem

ERM optimization problem:

$$\hat{\boldsymbol{\theta}} = \arg\min_{\boldsymbol{\theta} \in \Theta} \mathcal{R}_{\text{emp}}(\boldsymbol{\theta})$$

For **(global) minimum** $\hat{\boldsymbol{\theta}}$:

$$\forall \boldsymbol{\theta} \in \Theta : \quad \mathcal{R}_{\text{emp}}(\hat{\boldsymbol{\theta}}) \leq \mathcal{R}_{\text{emp}}(\boldsymbol{\theta})$$

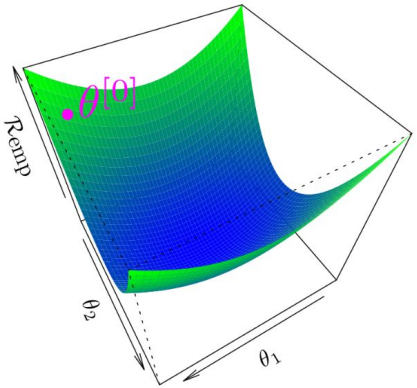
Does not imply that $\hat{\boldsymbol{\theta}}$ is unique.

- Best numerical optimizer depends on problem structure
- Continuous params? Uni-modal $\mathcal{R}_{\text{emp}}(\boldsymbol{\theta})$?
- Numerical optimization not our focus here, now

Stochastic Gradient Descent

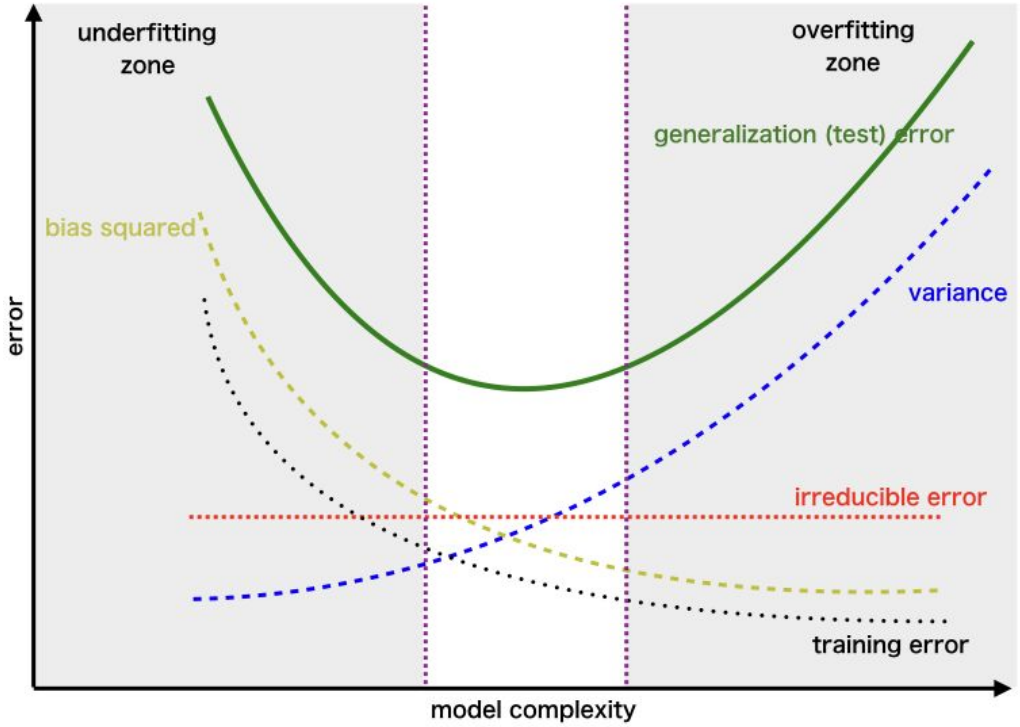
The simple idea of GD is to iteratively go from the current candidate $\boldsymbol{\theta}^{[t]}$ in the direction of the negative gradient, i.e., the direction of the steepest descent, with learning rate α to the next $\boldsymbol{\theta}^{[t+1]}$:

$$\boldsymbol{\theta}^{[t+1]} = \boldsymbol{\theta}^{[t]} - \alpha \frac{\partial}{\partial \boldsymbol{\theta}} \mathcal{R}_{\text{emp}}(\boldsymbol{\theta}^{[t]}).$$



We choose a random start $\boldsymbol{\theta}^{[0]}$ with risk $\mathcal{R}_{\text{emp}}(\boldsymbol{\theta}^{[0]}) = 76.25$.

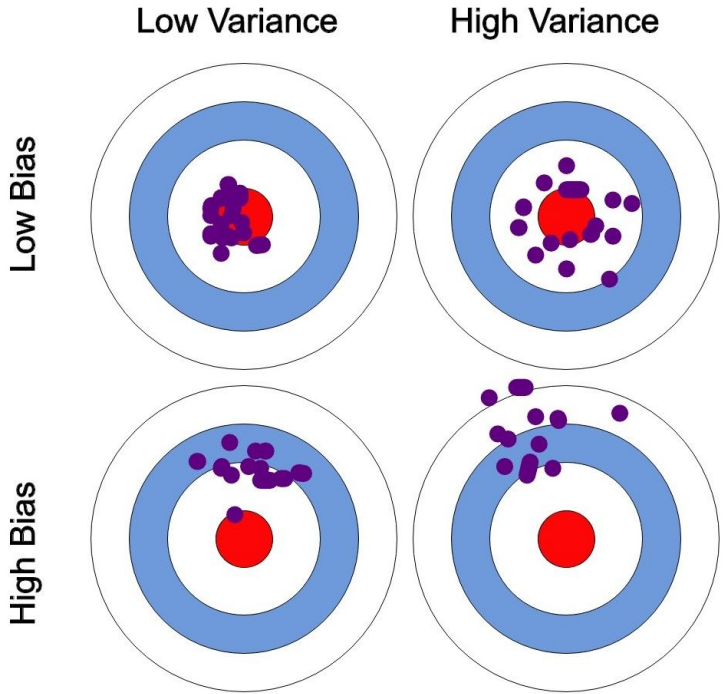
What is the best model complexity?



Bias and variance

Bias: a systematic shift of the entire distribution of points

Variance: the spread in data from with respect to the mean position.



How do we prevent overfitting without reducing the training sample size significantly?

TRAINING/DEVELOPMENT/TEST SET

- Highly efficient and easy to implement from a computational point of view

- The dataset has to be inherently homogeneous, otherwise with a single ‘unlucky’ split you may end up with a not representative training set

CROSS-VALIDATION

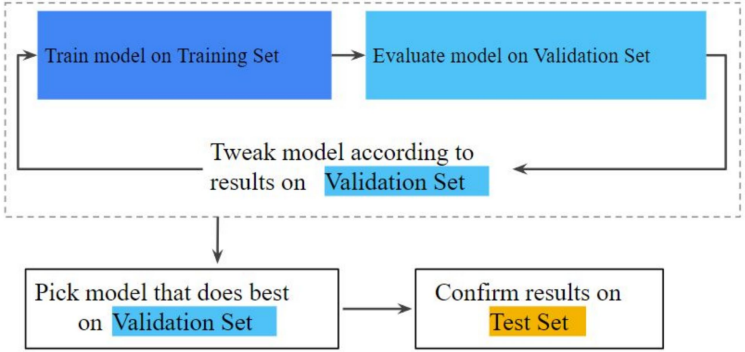
- Efficient sample re-use (all data are used for training, hence all possible sub-populations are covered)
- Adaptable to different sample sizes

- Can become computationally expensive when too many folds need to be evaluated

Training/Validation/Test split (to build robust models)

To deploy a ML algorithm parametrized by a set of hyper-parameters we do:

1. Select HP values
2. Split input data set into training/validation/test set
3. Train on the training set
4. Evaluate performances on the validation set
5. Go back to 1 (select new HP values)
6. Select HP leading to the smallest validation error
7. Compute error estimation on the test set



Cross-validation

- **Exhaustive cross-validation** methods learn and test on all possible ways to divide the original sample into a training and a validation set

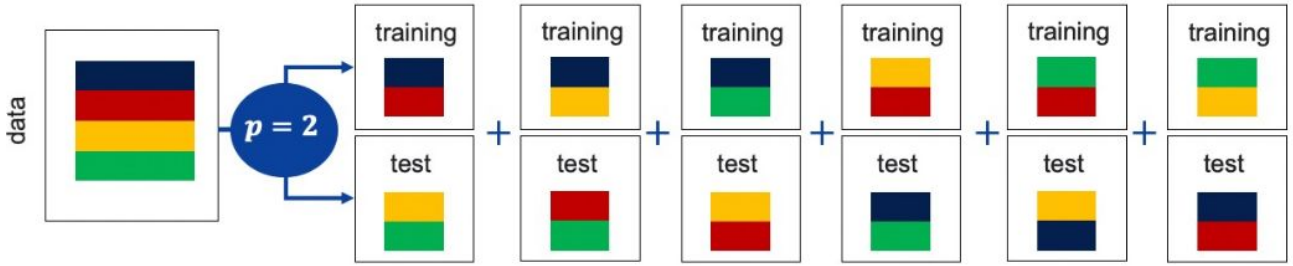
LEAVE-*p*-OUT CROSS-VALIDATION

- Given a dataset containing n events
- Use p events out of the n for the validation and $(n-p)$ for the training
- Repeat the training and test for all possible combinations of the p events
- Average the results of each train-test to obtain the overall results

How many combinations can we create if n=100 and p=30?

The number of possible combination is determined by the Binomial coefficient $C_p^n = \binom{n}{p}$. Therefore: $C_{30}^{100} = \binom{100}{30} \cong 3 * 10^{25}$

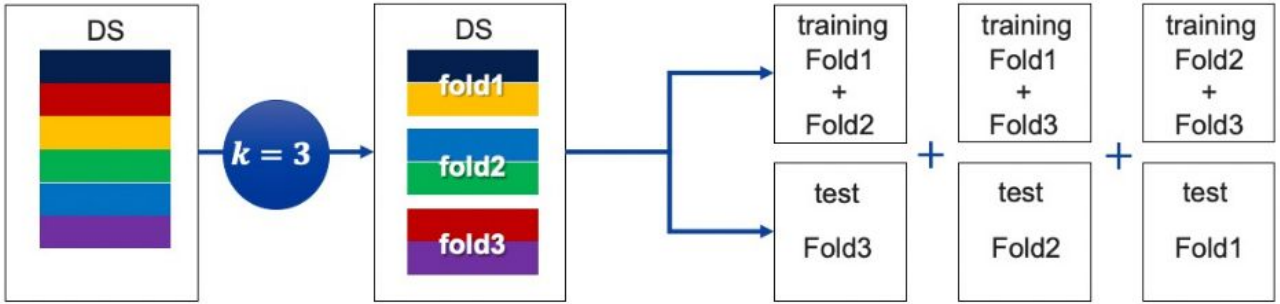
For $p > 1$ and for even moderately large n, it may quickly become computationally infeasible



Non-exhaustive cross-validation do not compute all ways of splitting the original sample. Those methods are approximations of leave-*p*-out cross-validation.

K-FOLD CROSS-VALIDATION

- Split the dataset into k random subsample
- Use $(k-1)$ samples for the training and the remaining one for the test
- Repeat the process k times and average the results
- If $k = n$ we come back to the leave-p-out strategy with $p = 1$
- A standard value of k is 10



Other causes of overfitting: data leakage

Preprocessing Leakage: If data preprocessing (like normalisation, feature selection) uses the entire dataset rather than just the training set, information from the test set can leak into the model.

Model Selection Leakage: When model selection or hyperparameter tuning is done using the test set, it leads to an overfitting on the test data, providing a misleadingly high performance estimate (more on that later).

Temporal Data Leakage: In time-series data, using future data in the training process leads to leakage as the model gets access to information it would not have in a real-world scenario.

Regression models

- **KNN regression**
- **Linear regression**
- **Regularized/penalized linear regression (Lasso, Ridge)**
- **Bayesian linear regression**

The first two regression can be implemented with OLS method or with SGD.

KNN regression

Given a dataset $D = \{(x^{(1)}, y^{(1)}), \ldots, (x^{(N)}, y^{(N)})\}$ for every new x do:

1. find the k-number obs in D most similar to X (these k obs. points are called k-nearest neighbours of x)
2. average the output of the k-nearest neighbors of x

$$\hat{y} = \frac{1}{K} \sum_{k=1}^{K} y^{(n_k)}$$

Pros -Cons of KNN

K-NN regression has both strengths and weaknesses. Some are listed here:

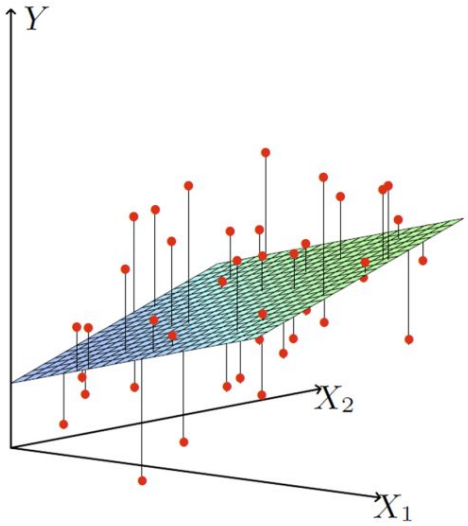
Strengths: K-nearest neighbors regression

1. is a simple, intuitive algorithm,
2. requires few assumptions about what the data must look like, and
3. works well with non-linear relationships (i.e., if the relationship is not a straight line).

Weaknesses: K-nearest neighbors regression

1. becomes very slow as the training data gets larger,
2. may not perform well with a large number of predictors, and
3. may not predict well beyond the range of values input in your training data.

Linear regression



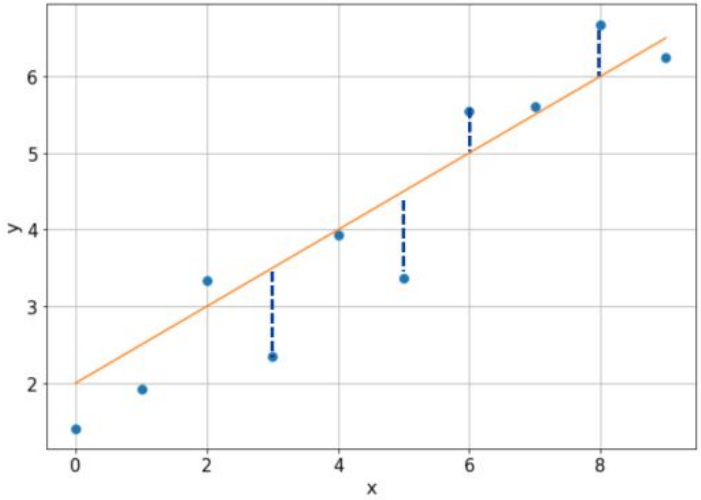
Linear least squares fitting with $X \in \mathbb{R}^2$. *We seek the linear function of* X *that minimizes the sum of squared residuals from* Y.

From: Hastie, Tibshirani, Friedman, "The Elements of Statistical Learning", Springer,

Linear regression

Model definition: $y = \beta_0 + \beta_1 x + \varepsilon$, where:

- β_0 is the intercept of the model,
- β_1 represents the slope



- The best model is the one that minimises the *sum of squared (SS) residuals* over the n observations:

$$\hat{\beta} = \underset{\beta}{\operatorname{argmin}} \sum_{i=1}^{n} [y_i - f(\beta, x_i)]^2$$

COST FUNCTION

- The algorithm used to find the optimal values $\hat{\beta}_0$ and $\hat{\beta}_1$ is called **OLS** (Ordinary Least Squares)

$$\hat{y} = \hat{\beta}_0 + \hat{\beta}_1 x$$

Ordinary Least Squares

$$\hat{y} = w^T \mathbf{x} + b = \sum_{i=1}^{p} w_i x_i + b$$

$$\min_{w \in \mathbb{R}^p, b \in \mathbb{R}} \sum_{i=1}^{n} ||w^T \mathbf{x}_i + b - y_i||^2$$

Unique solution if $\mathbf{X} = (\mathbf{x}_1, \ldots \mathbf{x}_n)^T$ has full column rank.

Ridge Regression

$$\min_{w \in \mathbb{R}^p, b \in \mathbb{R}} \sum_{i=1}^{n} (w^T \mathbf{x}_i + b - y_i)^2 + \alpha ||w||^2$$

L2 regularization

Always has a unique solution.

Tuning parameter alpha.

Lasso Regression

$$\min_{w \in \mathbb{R}^p, b \in \mathbb{R}} \sum_{i=1}^{n} ||w^T \mathbf{x}_i + b - y_i||^2 + \underbrace{\alpha ||w||_1}_{\text{L1 regularization}}$$

- Shrinks w towards zero like Ridge
- Sets some w exactly to zero - automatic feature selection!

Linear regression from a probabilistic perspective

If we assume $Y \in \mathbb{R}$ and $\mathbf{X} \in \mathbb{R}^p$

$$f_{Y|\mathbf{X}}(y|\mathbf{x}) = \mathcal{N}(y|\mathbf{w}^T\mathbf{x} + w_0, \sigma^2),$$

Parameters of $f_{Y|\mathbf{X}}$ are unknown; instead, i.i.d. training data:

$$\mathcal{D} = \big((\mathbf{x}_1, y_1), ..., (\mathbf{x}_n, y_n)\big)$$

the likelihood is :

$$\mathrm{L} = \prod_{i=1} \mathcal{N}(y_i; \mathbf{w}^T\mathbf{x}_i + w_0, \sigma^2)$$

$$\log(\mathrm{L}) = K - \frac{1}{2\sigma^2} \sum_{i=1}^{n} (y_i - \mathbf{w}^T\mathbf{x}_i - w_0)^2$$

Bayesian Learning for supervised ML

Step 1: *Given* n *data,* $D = x_{1:n} = \{x_1, x_2, \ldots, x_n\}$*, write down the expression for the* ***likelihood****:*

$$p(D \mid \theta)$$

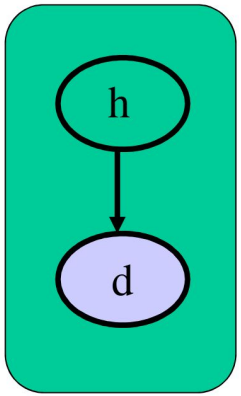
Step 2: *Specify a* ***prior****:* $p(\theta)$

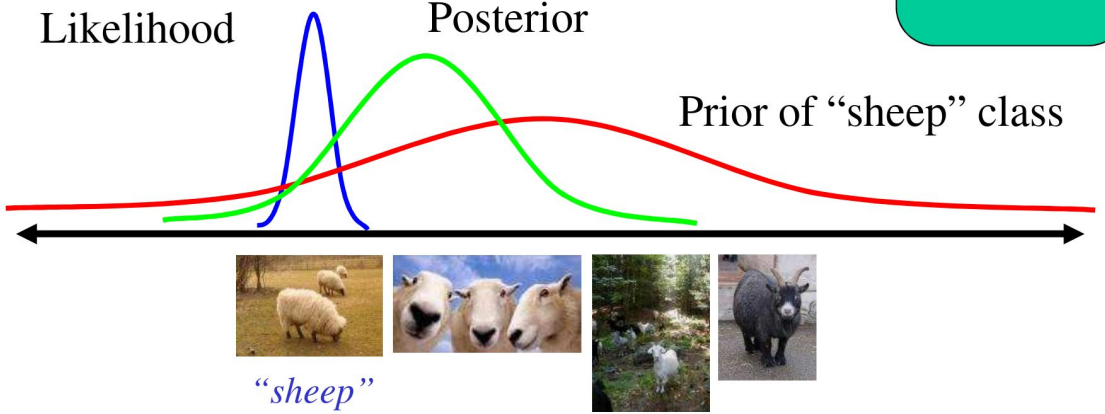
Step 3: *Compute the* ***posterior****:*

$$p(\theta \mid D) = \frac{p(D \mid \theta)\, p(\theta)}{p(D)}$$

Bayesian Learning for supervised ML

$$p(h \mid d) = \frac{p(d \mid h)\, p(h)}{\sum_{h' \in H} p(d \mid h')\, p(h')}$$



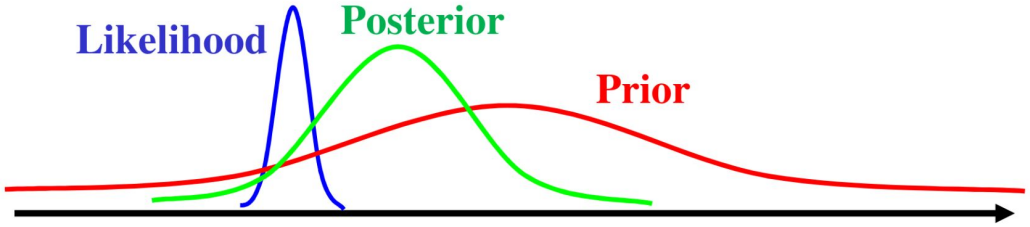


Bayesian Linear regression

The likelihood is a Gaussian, $\mathcal{N}(\mathbf{y}|\mathbf{X}\boldsymbol{\theta}, \sigma^2\mathbf{I}_n)$. The conjugate prior is also a Gaussian, which we will denote by $p(\boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{\theta}|\boldsymbol{\theta}_0, \mathbf{V}_0)$.

Using Bayes rule for Gaussians, the posterior is given by

$$
\begin{aligned}
p(\boldsymbol{\theta}|\mathbf{X}, \mathbf{y}, \sigma^2) &\propto \mathcal{N}(\boldsymbol{\theta}|\boldsymbol{\theta}_0, \mathbf{V}_0)\mathcal{N}(\mathbf{y}|\mathbf{X}\boldsymbol{\theta}, \sigma^2\mathbf{I}_n) = \mathcal{N}(\boldsymbol{\theta}|\boldsymbol{\theta}_n, \mathbf{V}_n) \\
\boldsymbol{\theta}_n &= \mathbf{V}_n\mathbf{V}_0^{-1}\boldsymbol{\theta}_0 + \frac{1}{\sigma^2}\mathbf{V}_n\mathbf{X}^T\mathbf{y} \\
\mathbf{V}_n^{-1} &= \mathbf{V}_0^{-1} + \frac{1}{\sigma^2}\mathbf{X}^T\mathbf{X}
\end{aligned}
$$



Special case:

Consider the special case where $\boldsymbol{\theta}_0 = \mathbf{0}$ and $\mathbf{V}_0 = \tau_0^2 \mathbf{I}_d$, the posterior mean reduces to

$$\begin{aligned} \boldsymbol{\theta}_n &= \frac{1}{\sigma^2}\mathbf{V}_N \mathbf{X}^T \mathbf{y} = \frac{1}{\sigma^2}\left(\frac{1}{\tau_0^2}\mathbf{I}_d + \frac{1}{\sigma^2}\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T\mathbf{y} \\ &= \left(\lambda \mathbf{I}_d + \mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T\mathbf{y} \end{aligned}$$

where we have defined $\lambda := \frac{\sigma^2}{\tau_0^2}$. We have therefore recovered **ridge regression** again!

Extending regression to non-normal /non-linear problems

GLM

$$g(\mathbb{E}[Y]) = \beta_0 + \beta_1 X_1 + \beta_2 X_2$$

GAM

Further relax GLMs by modeling each predictor with a **smooth function**:

$$g(\mathbb{E}[Y]) = \beta_0 + f_1(X_1) + f_2(X_2)$$

Common smooth functions: splines, loess…

Advantages: captures non-linear effects, retains interpretability

Gaussian Process regression

There are two possible definitions:

1. *A Gaussian process is a collection of random variables, any finite number of which have consistent Gaussian distributions.*
2. *A Gaussian process is a probability distribution over possible functions that fit a set of points.*

$$f(\mathbf{x}) \sim GP(m(\mathbf{x}), \kappa(\mathbf{x}, \mathbf{x}'))$$

$$m(\mathbf{x}) = \mathbb{E}[f(\mathbf{x})]$$

$$\kappa(\mathbf{x}, \mathbf{x}') = \mathbb{E}\left[(f(\mathbf{x}) - m(\mathbf{x}))(f(\mathbf{x}') - m(\mathbf{x}'))^T\right]$$

$$k(x, x') = \exp(-\tfrac{1}{2}(x - x')^2)$$

Important hyper-pars of GP regressor

Non-parametric (∞-parametric)

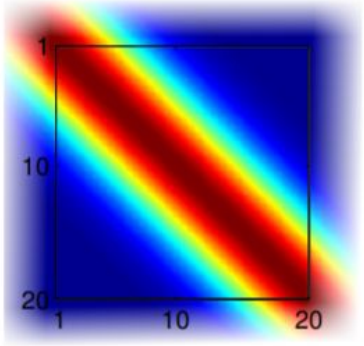
$$p(y|\theta) = \mathcal{N}(0, \Sigma)$$

$$\Sigma(x_1, x_2) = K(x_1, x_2) + I\sigma_y^2$$

$$K(x_1, x_2) = \sigma^2 e^{-\frac{1}{2l^2}(x_1 - x_2)^2}$$

vertical-scale horizontal-scale

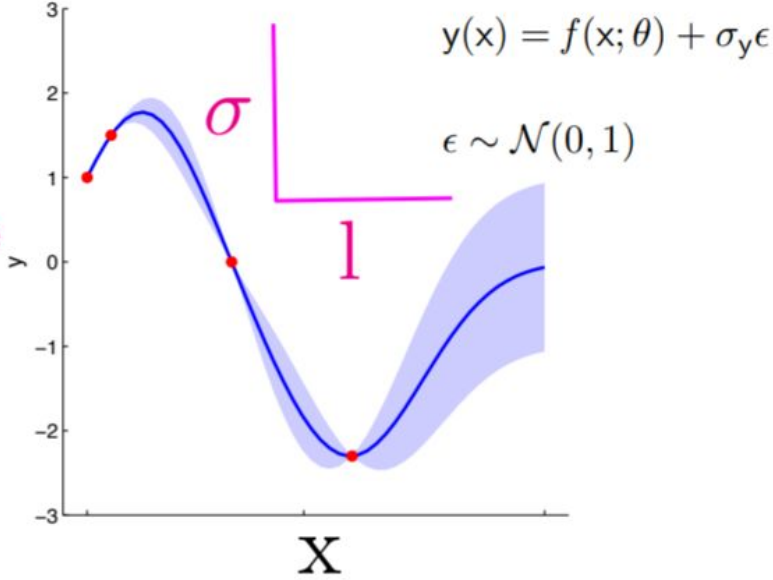
$\Sigma =$



Parametric model

$$y(x) = f(x; \theta) + \sigma_y \epsilon$$

$$\epsilon \sim \mathcal{N}(0, 1)$$



GP algorithm

input: X (inputs), $\mathbf{y}$ (targets), k (covariance function), σ_n^2 (noise level), $\mathbf{x}_*$ (test input)

2: $L := \text{cholesky}(K + \sigma_n^2 I)$

$\boldsymbol{\alpha} := L^\top \backslash (L \backslash \mathbf{y})$

4: $\bar{f}_* := \mathbf{k}_*^\top \boldsymbol{\alpha}$ } predictive mean eq. (2.25)

$\mathbf{v} := L \backslash \mathbf{k}_*$

6: $\mathbb{V}[f_*] := k(\mathbf{x}_*, \mathbf{x}_*) - \mathbf{v}^\top \mathbf{v}$ } predictive variance eq. (2.26)

$\log p(\mathbf{y}|X) := -\frac{1}{2}\mathbf{y}^\top \boldsymbol{\alpha} - \sum_i \log L_{ii} - \frac{n}{2} \log 2\pi$ eq. (2.30)

8: **return**: $\bar{f}_*$ (mean), $\mathbb{V}[f_*]$ (variance), $\log p(\mathbf{y}|X)$ (log marginal likelihood)

Estimating model performance

Mean squared error	$\mathrm{MSE} = \frac{1}{n}\sum_{t=1}^{n} e_t^2$
Root mean squared error	$\mathrm{RMSE} = \sqrt{\frac{1}{n}\sum_{t=1}^{n} e_t^2}$
Mean absolute error	$\mathrm{MAE} = \frac{1}{n}\sum_{t=1}^{n} \lvert e_t \rvert$
Mean absolute percentage error	$\mathrm{MAPE} = \frac{100\%}{n}\sum_{t=1}^{n} \left\lvert \frac{e_t}{y_t} \right\rvert$

Pros/cons of MSE vs RMSE vs MAE

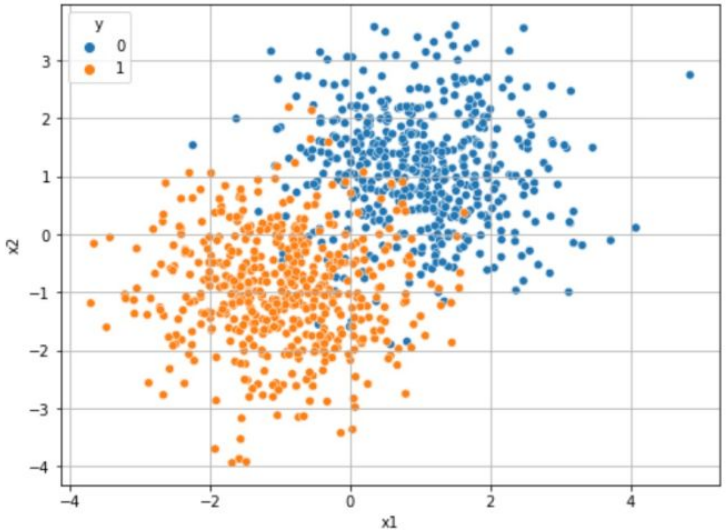
- $MSE = \frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2$ — It assigns more weight to bigger errors, useful when interested in **detecting outliers**
- $RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2}$ — It maintains the same properties as the MSE, but it is in the same scale as y
- $MAE = \frac{1}{n}\sum_{i=1}^{n}|\hat{y}_i - y_i|$ — It is simply the average difference observed in the predicted and actual values across the whole test set, useful if not particularly interested in outliers

Classification models

- They all share the same schema, by having:
  - a response variable (**target**) of categorical nature,
  - a number of explanatory variables (**features**) which are assumed to be helpful in classifying each observation

target

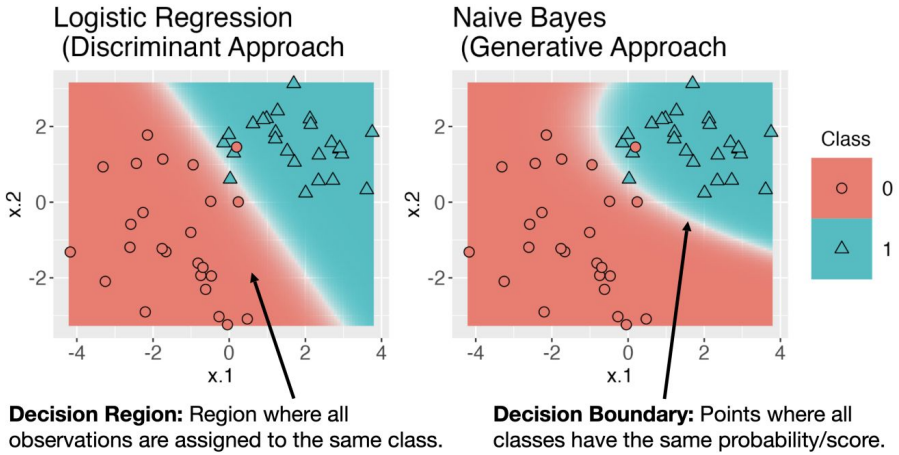
$$y = \begin{cases} 0, & \textit{negative class} \\ 1, & \textit{positive class} \end{cases}$$



Approaches to classification

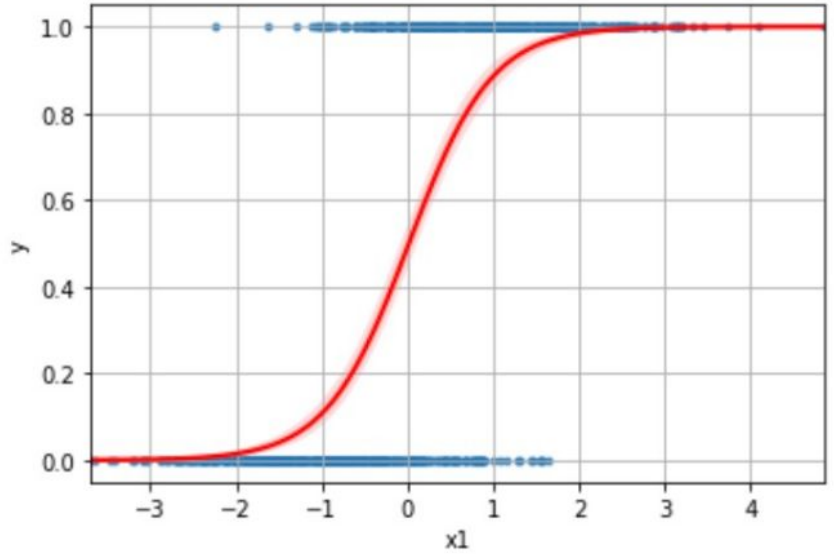
Two fundamental approaches exist to construct a classifier:

- **Discriminant approach** asks "What is the best prediction for the class given these data?" (uses loss functions and empirical risk minimization)
- **Generative approach** asks "Which class tends to have data like these?" (models the feature distributions in each class separately)



Logistic regression for binary classification

The logistic regression aims at modelling a function with this shape (also called **logistic or sigmoid function**):

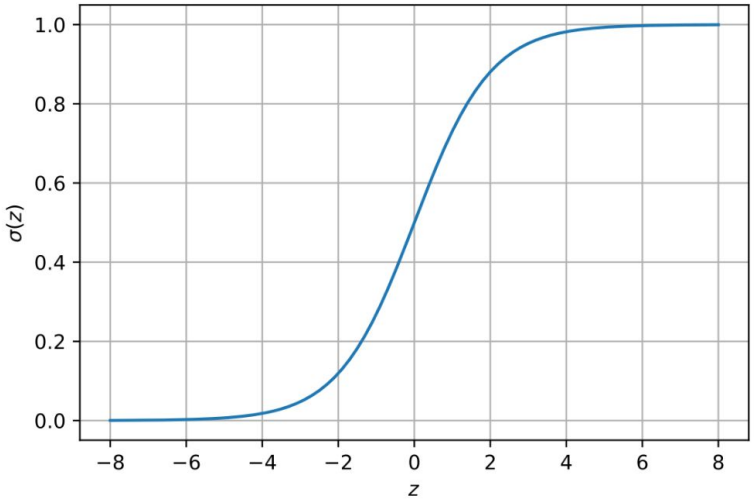


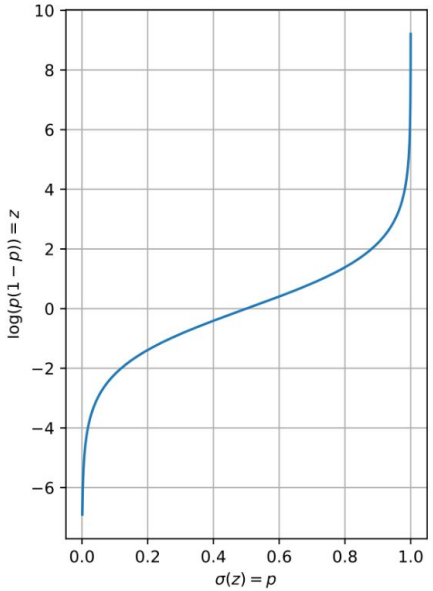
- It is bounded in $[0, 1]$ as probabilities also are
- It is S-shaped, being is a cumulative distribution along the variable $\int_{x0}^{x1} pdf(x)\,dx$

$$p(x;\beta) = \frac{e^{\beta_0+\vec{\beta}\cdot\vec{x}}}{1+e^{\beta_0+\vec{\beta}\cdot\vec{x}}}$$

The logit function

$$z = \left(\sum_{i=1}^{n} w_i x_i\right) + b$$



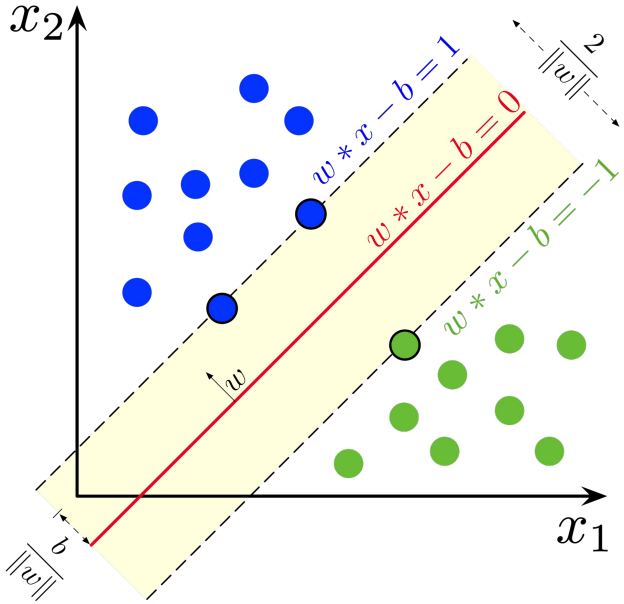


$$\sigma(z) = \frac{1}{1 + \exp(-z)}$$

$$logit(\sigma(z)) = \log(\sigma(z)/(1 - \sigma(z))) = z$$

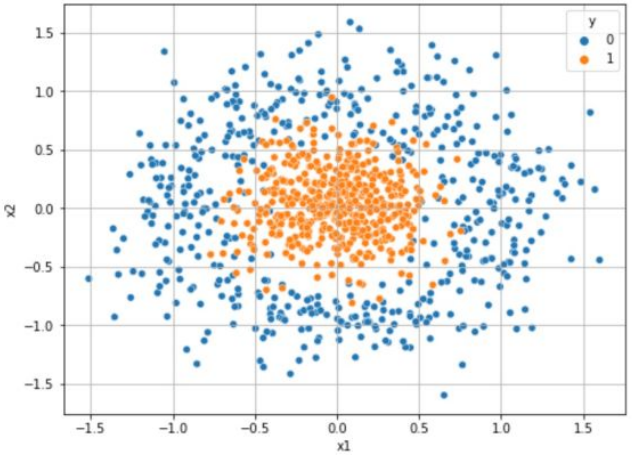
Support vector classifier

Basic idea: maximize the distance between the support vector points and the decision boundary between classes

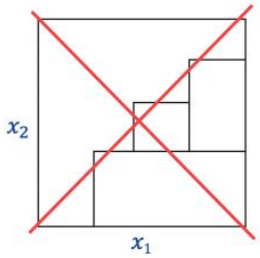


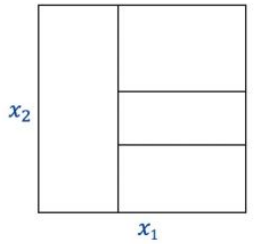
Decision trees

TASK: fit a 2-depth tree on the following data



- The algorithm aims at partitioning the two-dimensional features space by **recursive binary splitting (sub-optimal solution)**





- At each node, all the available **features are evaluated**
- A given **objective function** is then optimized to find the most informative feature and point for the split

How do decision trees actually works?

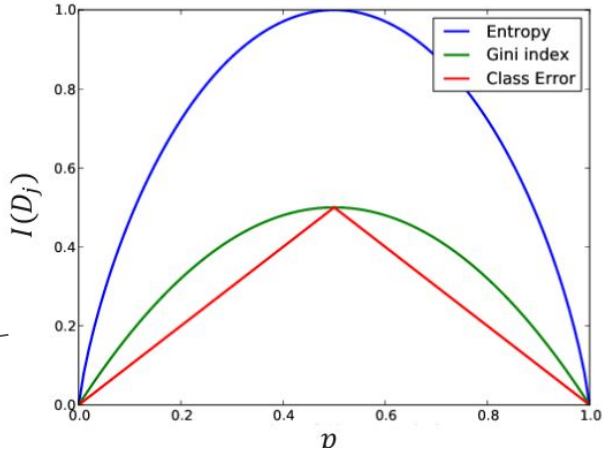
Splitting algorithm

1. Calculate the $I(D)$ of the parent node
2. Calculate the weighted average $I(D)$ of child nodes
3. Subtract the two quantities (estimate IG)
4. Select the split with the highest information gain

$$IG(D_p) = I(D_p) - \sum_{j=1}^{m} \frac{n_j}{n_p} I(D_j)$$

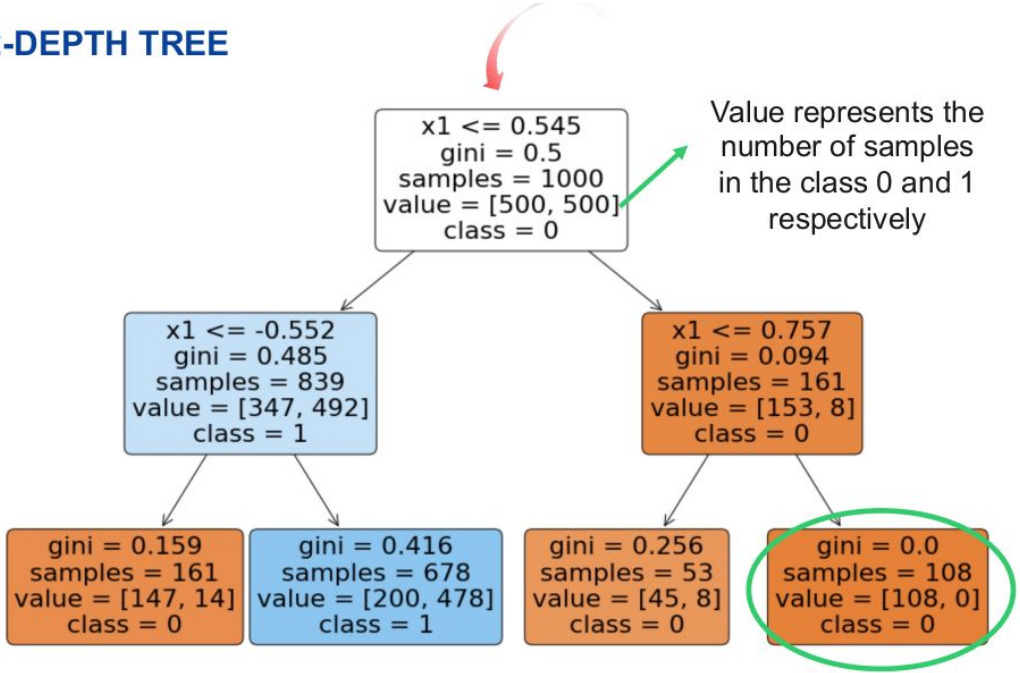
Gini index: $I(D_j) = \sum_{k=0,1} p_{jk}(1 - p_{jk})$

Entropy: $I(D_j) = -\sum_{k=0,1} p_{jk} \log_2(p_{jk})$

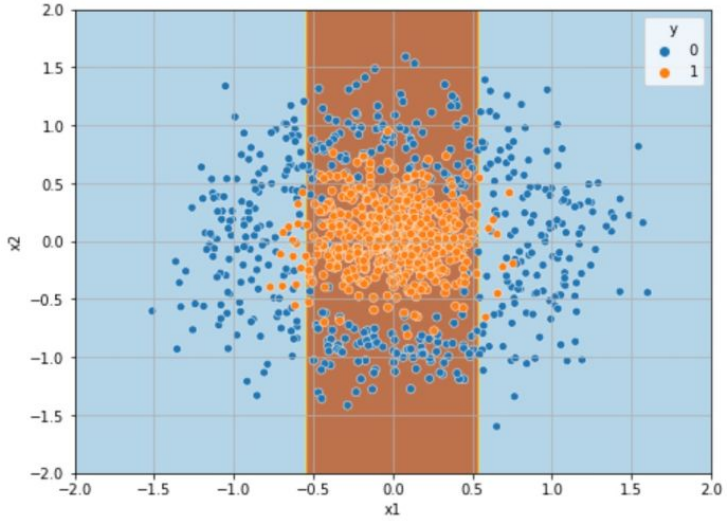


What is the best depth for our task?

2-DEPTH TREE



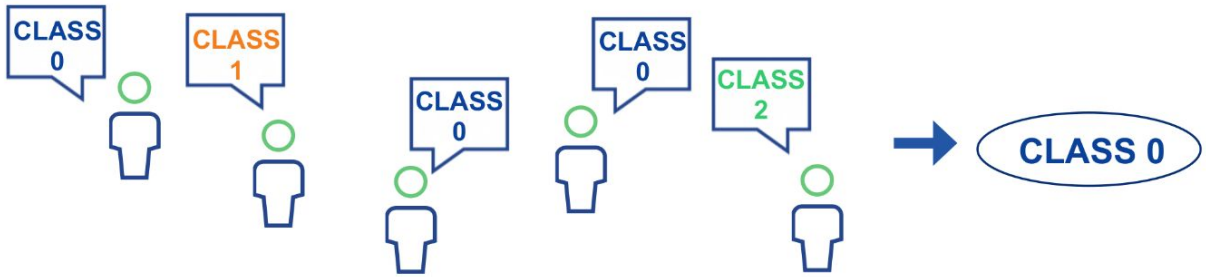
Decision boundaries



The power of collective knowledge

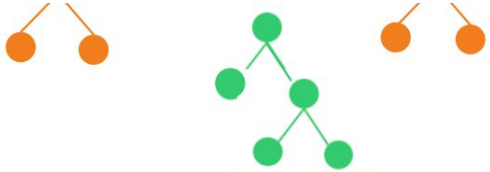
Ensemble models can be used both in regression and classification tasks

- **REGRESSION**: the final predictions can be the average or weighted average of the single models' predictions
- **CLASSIFICATION**: as before, the average or weighted average may be taken to compute the scores. The final classification is based on the **majority vote** among the single classifiers



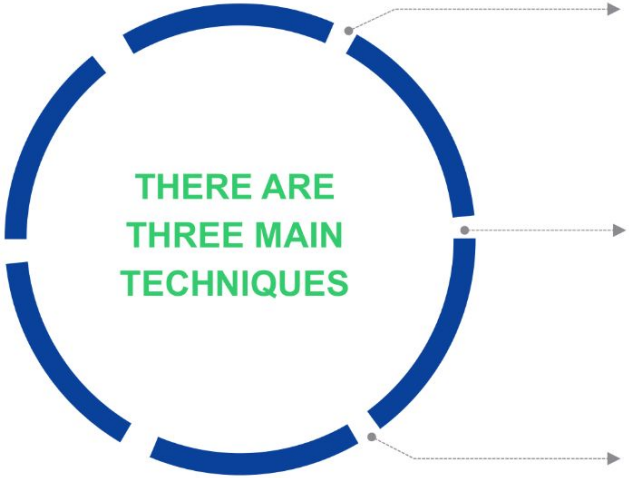
***Wisdom of Crowds* (Surowiecki, 2004)**
The collective knowledge of a diverse and independent body of people typically exceeds the knowledge of any single individual, and can be harnessed by voting

Defining the base classifier



- We call **base or weak classifier** the model used as **building block of the ensamble model**
- The base classifier is trained on **subsamples** of the training data, so that:
  - it should learn the details of the specific subsample
  - does not need to be much granular (e.g. a very deep decision tree) to learn details because the subsample has a relatively small size, hence less information to draw from (→ ***weak***)
- As it is trained multiple times, the **computational cost** of training the model is a relevant aspect to consider
- A very widespread choice is the **decision tree**
  - It is a highly unstable model (**high variance**: any little perturbation of the training data may result in significantly different tree structures)
  - Notoriously very fast to run

Ensemble models



BAGGING → parallel training with different training sets

BOOSTING → sequential training, iteratively re-weighting training examples so current classifier focuses on hard examples

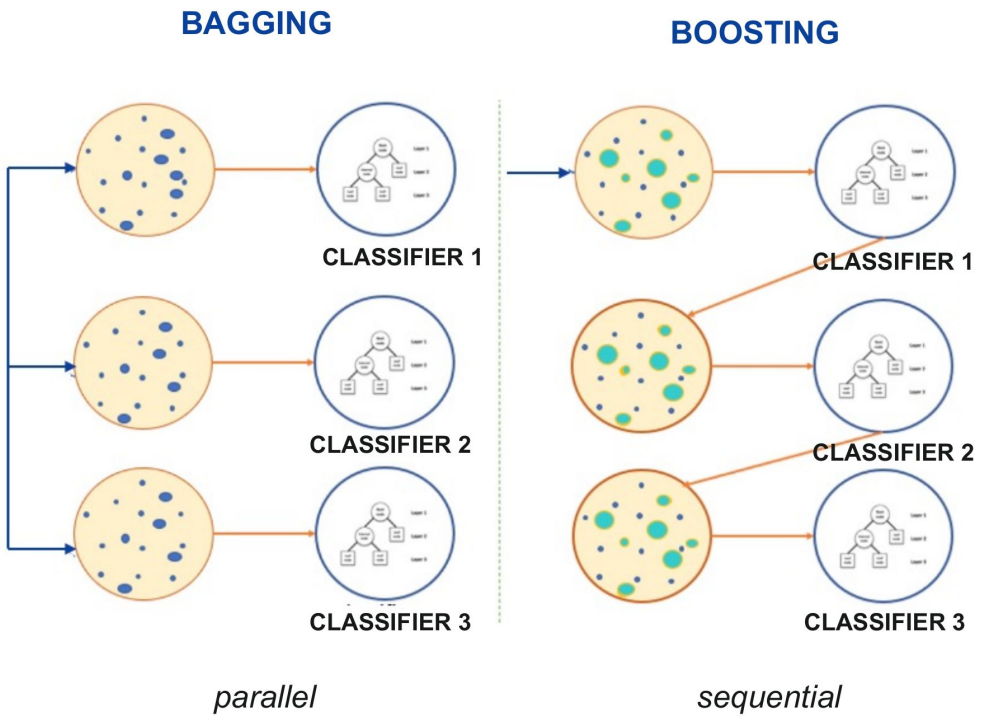
RANDOM FOREST → parallel training with different training sets and feature sets

Boosting

What differs is the **strategy** used to generate the subsets:

- *bagging* uses bootstrap sampling,
- *boosting,* at each step, increases the probability of picking events that were misclassified at the previous step

In the boosting model, each *weak classifier* is mostly focused on **learning how to correct the errors** done by the previous *weak classifier*



Bagging

weak learner

DATASET

Let $Z = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}$ be the complete training set

SAMPLING

Create m subsamples Z_i of size B out of Z using **bootstrapping** techniques (random sampling with replacement)

BASE LEARNER

Let $C_i(x)$ be the classifier trained on the subsample Z_i (**base classifier**)

COMBINED CLASSIFIER

Finally define the **ensemble classifier** as the average prediction over the m base learners:

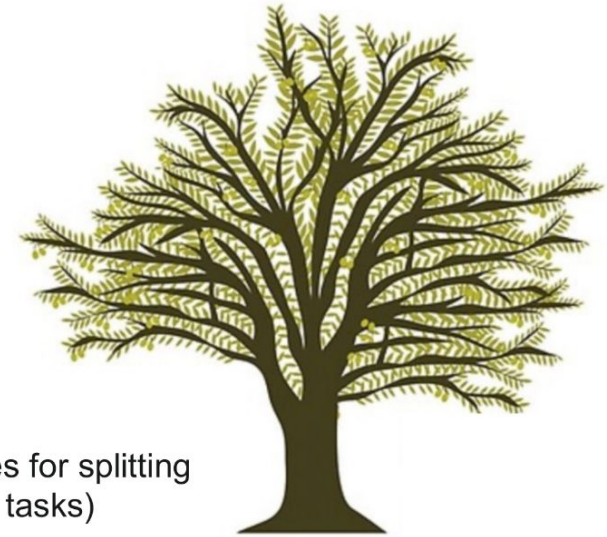
$$C_{bag}(x) = \frac{1}{m}\sum_{i=1}^{m} C_i(x)$$

Random Forest

- **Random forest** is a substantail modification of *bagging* that builds a large collection of ***de-correlated trees*** and then averages them (it is therefore specifically developed for trees as base learners)

The idea is to **improve the variance reduction** of *bagging* **by reducing the correlation of the trees**

To do so, in the tree-growing process ***features are randomly selected***



- Before each split, $m \leq p$ of the input features are selected at random as candidates for splitting (a common choice is $m = \sqrt{p}$ for classification tasks and $m = \lfloor p/3 \rfloor$ for regression tasks)

What kind of problems we solve with ensemble models?

Averaging weak learners outputs do not change the expected value (bias of the model) but **reduces its variance**

(just like averaging $i.i.d.$ random variables preserve expected value but reduces variance)

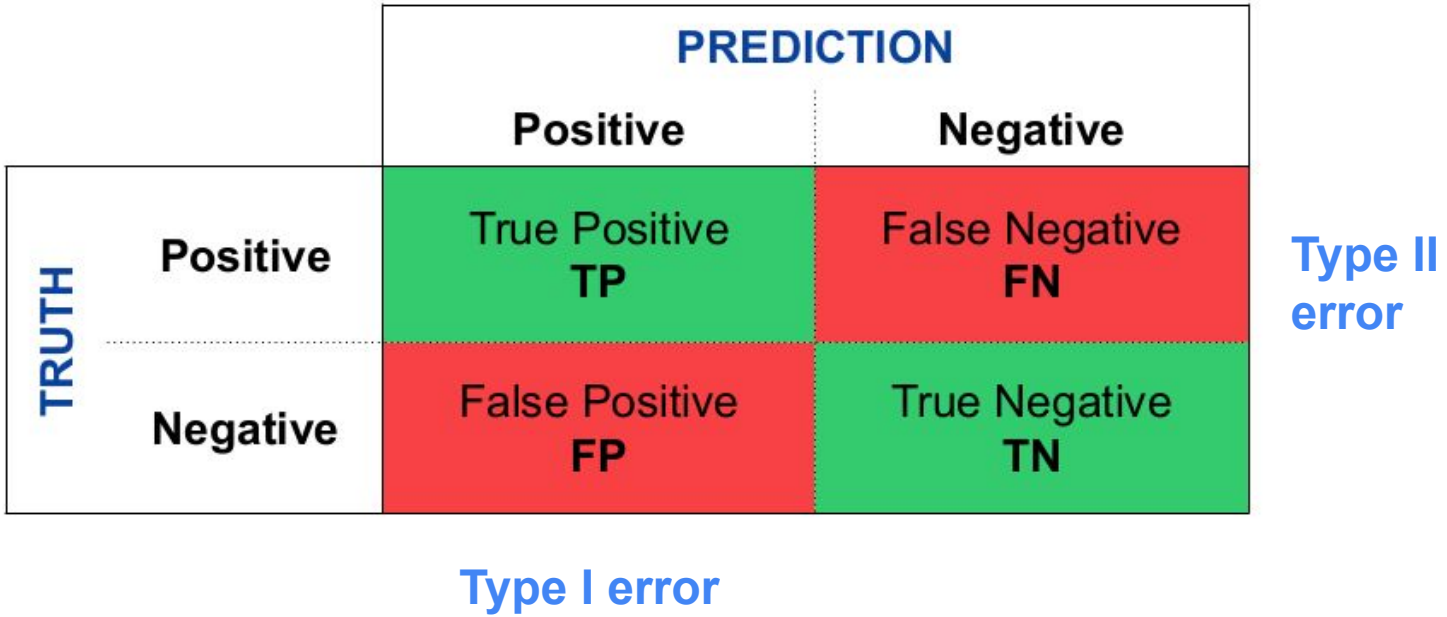
A variance reduction **decreases the prediction error** of the final model (bias-variance tradeoff)

$$Err(x) = Bias^2 + Variance + Irreducible\ Error$$

On which models bagging will perform especially well?

On low bias, high variance models (e.g. trees)

Confusion matrix



Evaluation metrics for classifiers

F1 score

$$2\frac{precision \cdot recall}{precision + recall} = \frac{2 \cdot TP}{2 \cdot TP + FP + FN}$$

$$\texttt{accuracy}(y, \hat{y}) = \frac{1}{n_{\text{samples}}} \sum_{i=0}^{n_{\text{samples}}-1} 1(\hat{y}_i = y_i)$$

FPR (False Positive Rate)

- *What is the proportion of misclassified instances out of all the true negatives?*
- $\frac{FP}{FP+TN}$ (should be as low as possible)

RECALL or TPR (True Positive Rate)

- *What proportion of actual positives is identified correctly?*
- $\frac{TP}{TP+FN}$ (should be high as possible)

PRECISION

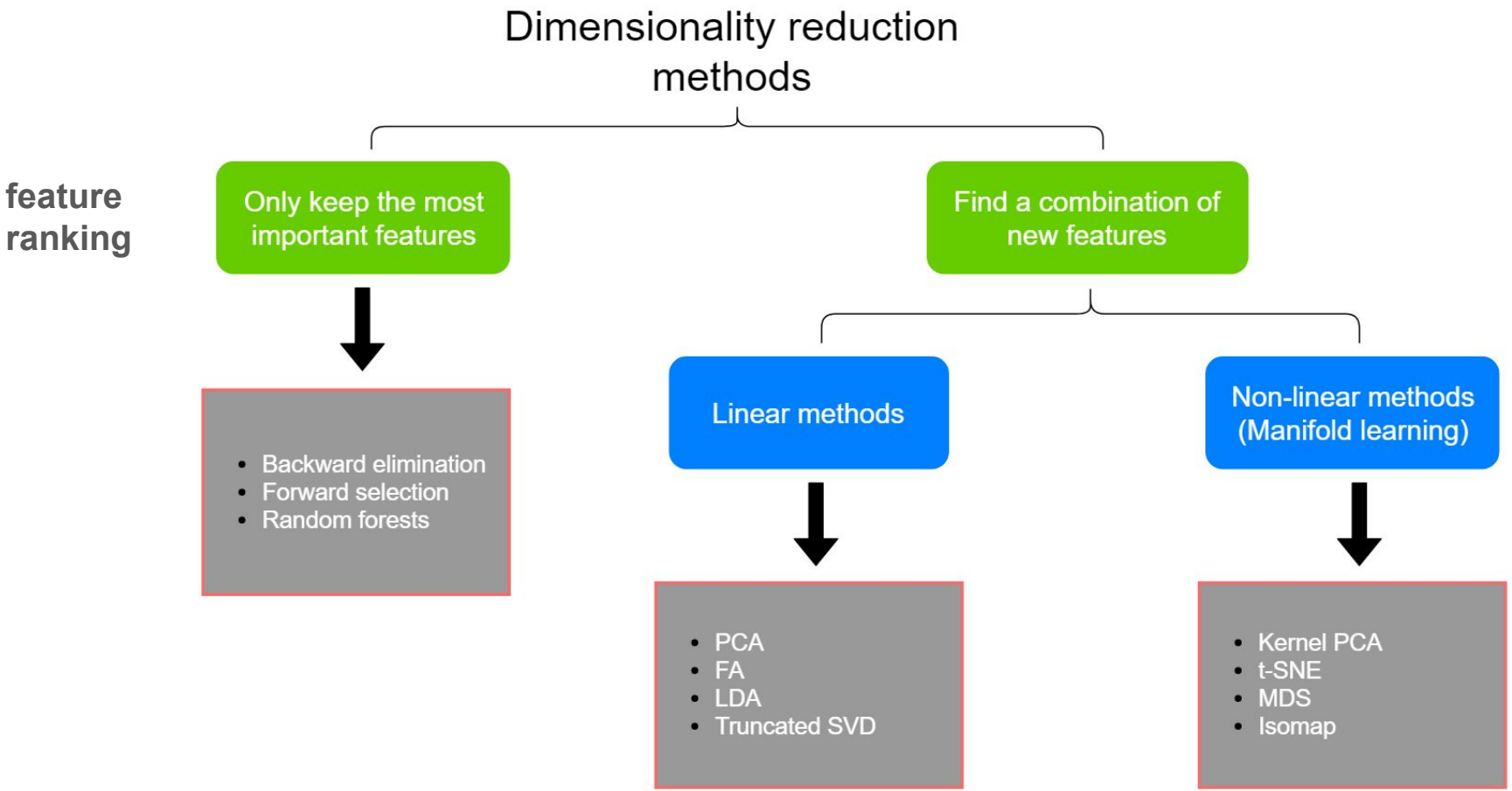
- *What proportion of positive predictions is actually correct?*
- $\frac{TP}{TP+FP}$ (should be high as possible)

How these metrics are called in scikit-learn?

Scoring string name	Function	Comment
Classification		
‘accuracy’	`metrics.accuracy_score`	
‘balanced_accuracy’	`metrics.balanced_accuracy_score`	
‘top_k_accuracy’	`metrics.top_k_accuracy_score`	
‘average_precision’	`metrics.average_precision_score`	
‘neg_brier_score’	`metrics.brier_score_loss`	
‘f1’	`metrics.f1_score`	for binary targets

Unsupervised Learning algorithm

- Manifold Learning (Dim. Red. & ID est.):
  - PCA
  - K-PCA
  - ISOMAP
  - t-SNE
  - Autoencoders
- Density Estimation:
  - Histograms
  - Kernel Density Estimation
  - k-Nearest Neighbor
  - Generative Adversarial NN
- Clustering:
  - k-means/c-means, kernel k-means, spectral clustering…
  - Hierarchical clustering
  - Density Based clustering
  - Self Organizig Maps


Dimensionality reduction methods
feature ranking
Only keep the most important features
Find a combination of new features
Backward elimination
Forward selection
Random forests
Linear methods
Non-linear methods (Manifold learning)
PCA
FA
LDA
Truncated SVD
Kernel PCA
t-SNE
MDS
Isomap

PCA - scheme 1

Given m observations of n-dim column vectors , to perform PCA we follow the 6 steps below:

1. Compute the mean vector $\vec{\mu} = \frac{1}{m}\sum_{j=1}^{m}\vec{x}^{(j)}$
2. Compute the covariance matrix (C = X^T X, S= C - C_mean)
3. Compute the sorted eigenvalue/eigenvector pairs (Diagonalization)
4. Choose the number of dimensions (k) in which project the data
5. Project the input data onto the selected k eigenvectors

PCA - scheme 2

Scheme 2 : PCA by SVD

Step 1. Center the data matrix,

$$\bar{X} = X - \mathbf{1}_n \bar{x}^\top.$$

Step 2. Apply singular value decomposition to $\bar{X}$,

$$\bar{X} = U \Sigma V^\top.$$

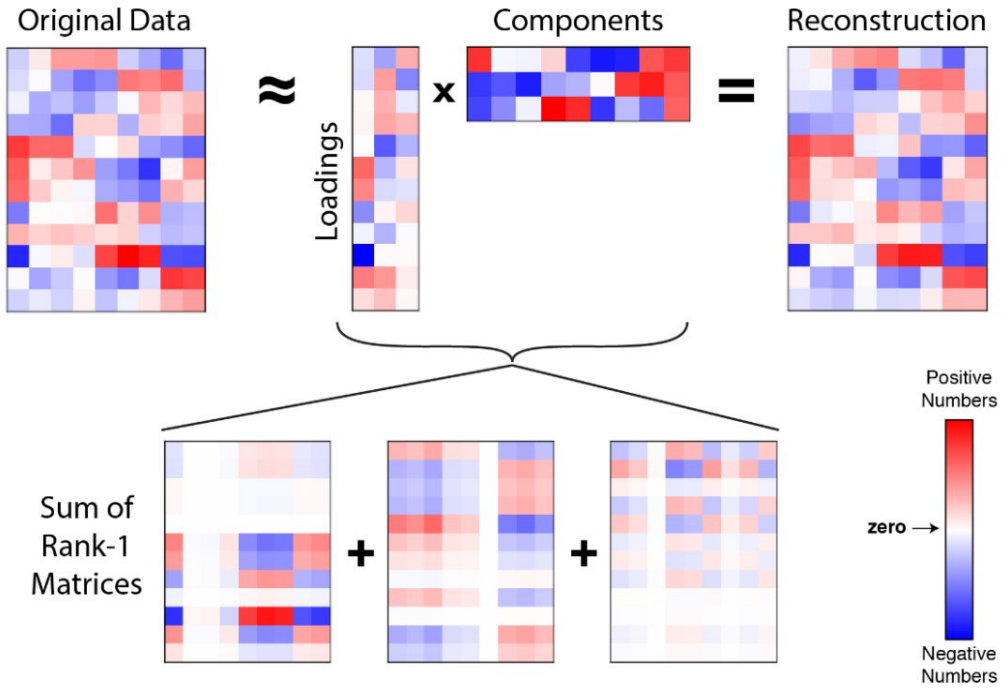
Note that left and right singular vectors u_i, v_j are ordered according to the descending order of singular values $\sigma_1 \geq \ldots \geq \sigma_p$, respectively.

Step 3. Assemble the projection matrix $V_{1:k}$ by taking the first k right singular vectors,

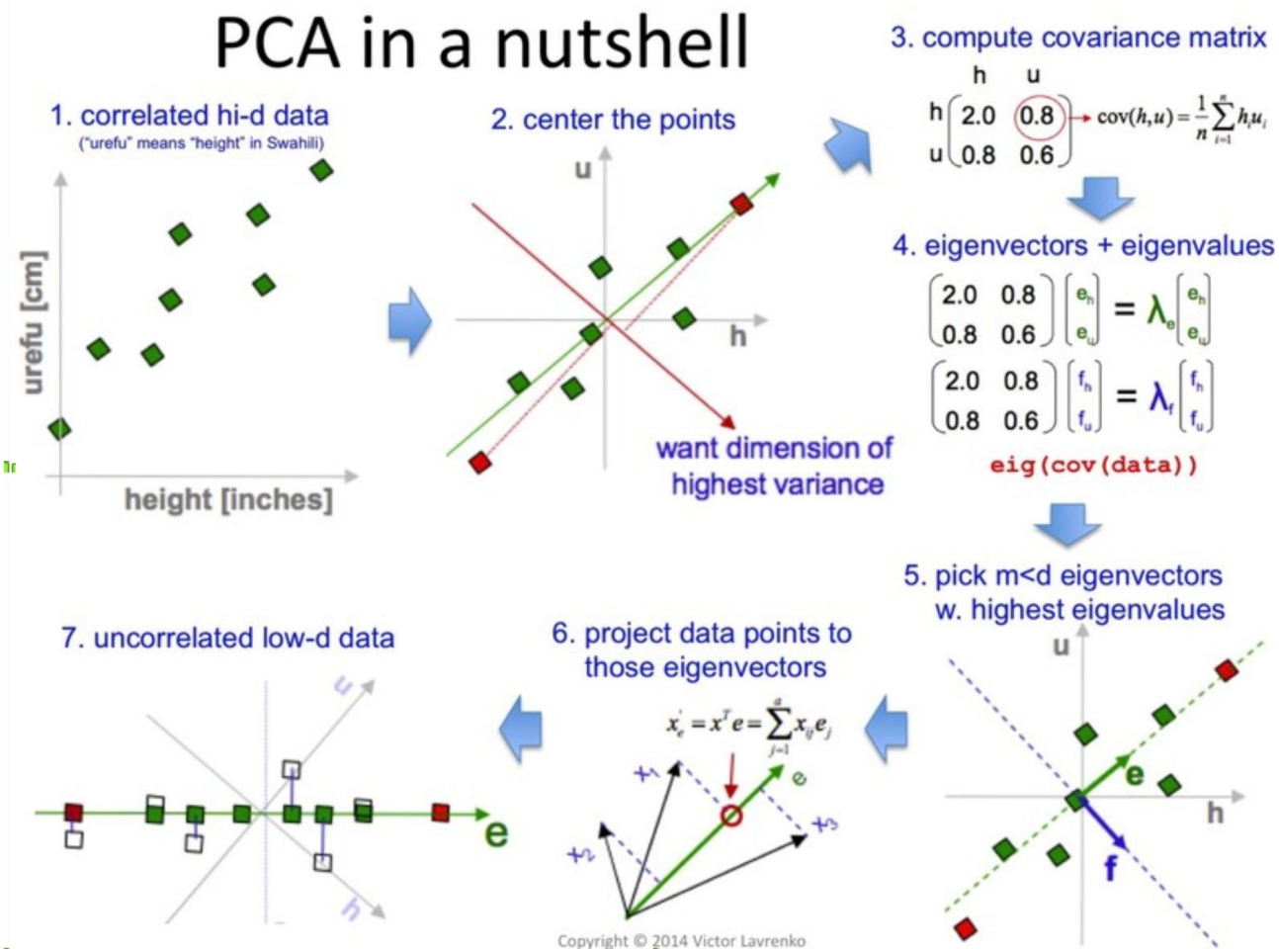
$$V_{1:k} = [v_1, \ldots, v_k] \in \mathbb{R}^{p \times k}.$$

Step 4. Compute the low-dimensional embedding $Y = X V_{1:k} \in \mathbb{R}^{n \times k}$.

$$\mathbf{X} \approx \sum_{k=1}^{r} \mathbf{w}_k \mathbf{c}_k^T \quad \text{or} \quad \mathbf{X} \approx WC^T$$

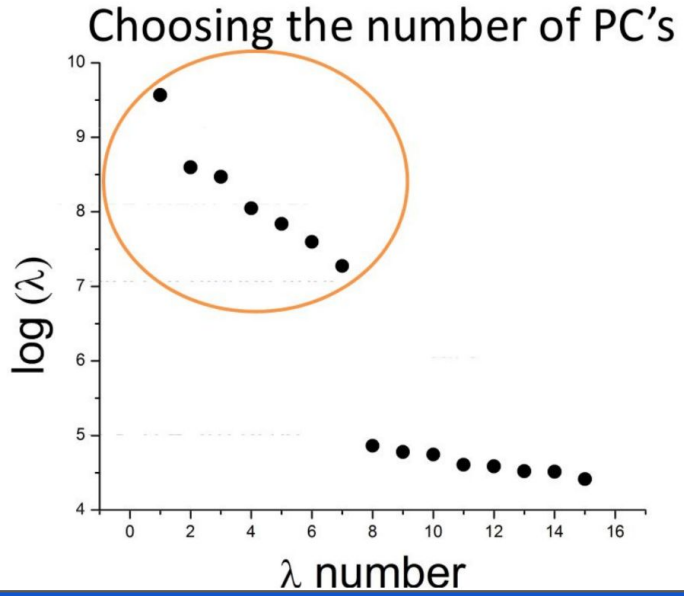


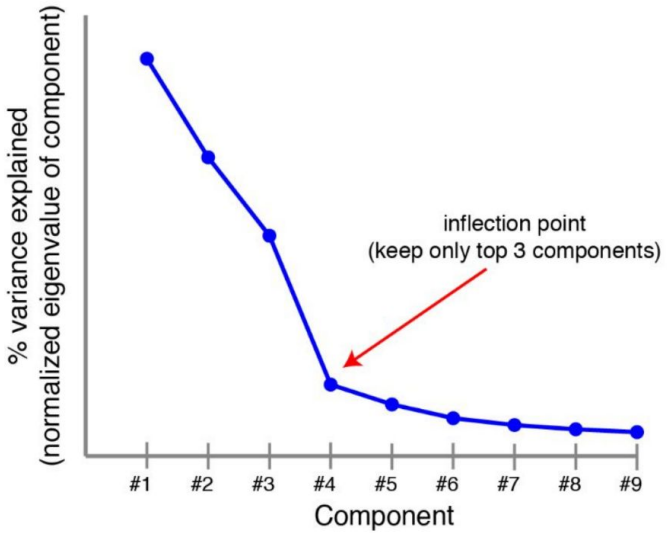
Example reconstruction of data with 3 principal components. A data matrix (*left*) is approximated by the product of a $n \times r$ matrix and a $r \times p$ matrix (i.e. WC^T). This product is at most a rank-r matrix (in this example, $r = 3$). Each paired column of W and row of C^T form an outer product, so the full reconstruction can also be thought of as a sum of r rank-one matrices.


PCA in a nutshell
1. correlated hi-d data
("urefu" means "height" in Swahili)
urefu [cm]
height [inches]
2. center the points
u
h
want dimension of highest variance
3. compute covariance matrix
h u
h 2.0 0.8
u 0.8 0.6
cov(h,u) = 1/n Σ h_i u_i
4. eigenvectors + eigenvalues
2.0 0.8 / 0.8 0.6 (e_h, e_u) = λ_e (e_h, e_u)
2.0 0.8 / 0.8 0.6 (f_h, f_u) = λ_f (f_h, f_u)
eig(cov(data))
5. pick m<d eigenvectors
w. highest eigenvalues
u
h
e
f
6. project data points to
those eigenvectors
x'_e = x^T e = Σ x_ij e_j
7. uncorrelated low-d data
e
Copyright © 2014 Victor Lavrenko

How do I choose the number of PCA components?

The scree plot (Cattel method)





Clustering

Clustering tries to separate data “naturally”, in such a way that ***similar*** elements lay in the same cluster while ***dissimilar*** elements belong to a different one

Similarity

pairwise function of the features. Its definition depends on the nature of input data.
It can be seen as a distance (but it does not always corresponds to metric distances!)

Clustering ≠ classification

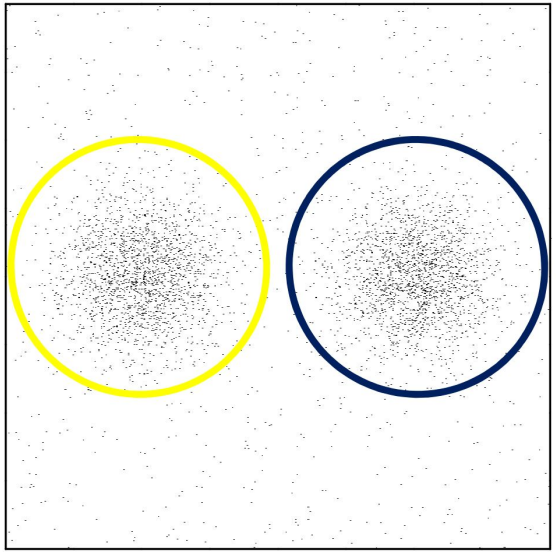
it generates groups without labels

Considerations on clustering

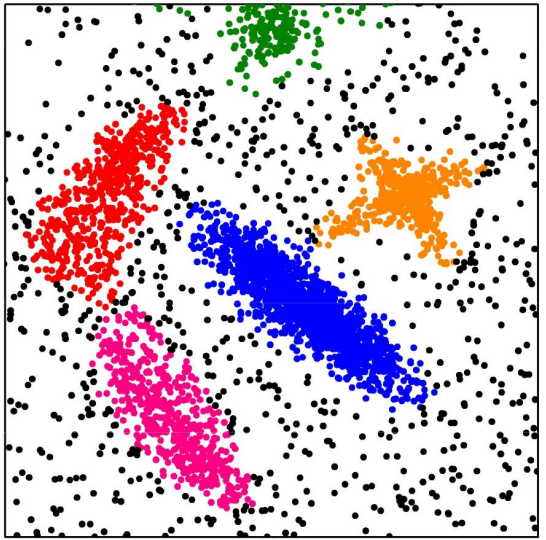
The result of the clustering process depends on:

- your cluster definition
- the metric adopted to measure distances
- the feature you choose to consider to cluster data

What is a cluster?

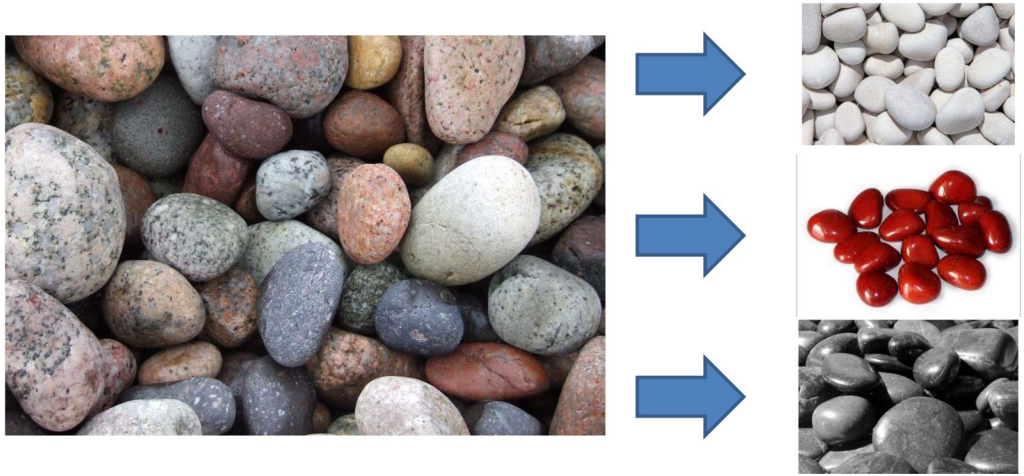


Other cluster examples…



Clustering stone collections

Cluster by light wavelength



Are they nice or ugly?
Color classification?

Most common flat clustering algorithm:

K-means: A flat clustering algorithm

MacQueen, J. (1967, June). Some methods for classification and analysis of multivariate observations. In Proceedings of the fifth Berkeley symposium on mathematical statistics and probability (Vol. 1, No. 14, pp. 281-297).

K-means algorithm

Input: cluster size *k*, instances $\{\overrightarrow{x_i}\}_{i=1}^{n}$
Output: cluster membership assignments $\{z_i\}_{i=1}^{n}$

1. Initialize *k* cluster centroids $\{\overrightarrow{c_l}\}_{l=1}^{k}$ (randomly from the data set).
2. Repeat until no instance changes its cluster membership:
   - Decide the cluster membership of instances by assigning them to the nearest cluster centroid

     $$z_i = argmin_l(d(\overrightarrow{c_l}, \overrightarrow{x_i}))$$ *Minimize intra distance*
   - Update the *k* cluster centroids based on the assigned cluster membership

     $$\overrightarrow{c_l} = \frac{\sum_{i=1}^{n} \delta_{z_{il}} \overrightarrow{x_i}}{\sum_{i=1}^{n} \delta_{z_{il}}}$$ *Maximize inter distance*

Loss function/Objective function

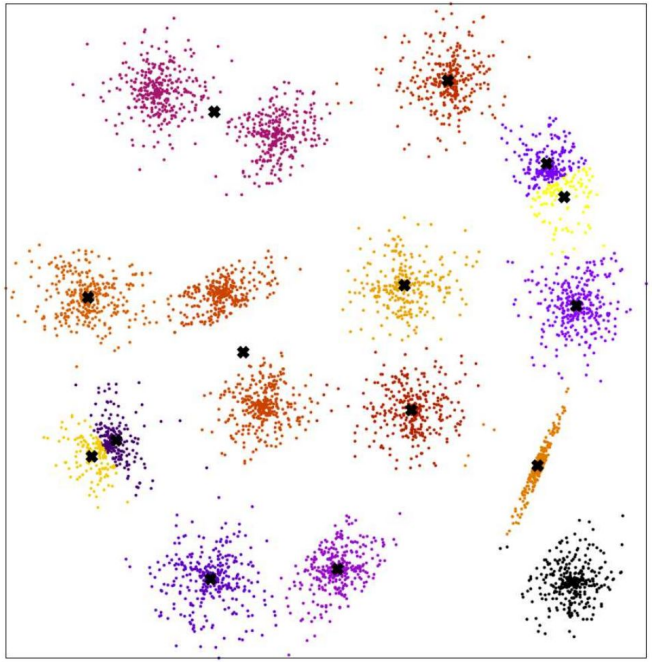
- Loss function:

$$O(z) = \sum_{l=1}^{k} \sum_{i=1}^{n} \delta(z_i, l) \quad \|\overrightarrow{x_i} - \overrightarrow{c_l}\|^2$$

- z is an array with n components that reflects the assignation in clusters.
- $\overrightarrow{c_l}$ is the vector with the coordinates of the l-th cluster centroid. $\overrightarrow{c_l} = \frac{\sum_{i=1}^{n} \delta_{z_{il}} \overrightarrow{x_i}}{\sum_{i=1}^{n} \delta_{z_{il}}}$

K-means steps

- Randomly pick *k* centers.
- Assign each point to its nearest center.
- Recompute centers.
- Iterate…



Outliers and novelty detections

(2 faces of anomaly detection)

Many applications require being able to decide whether a new observation belongs to the same distribution as existing observations (it is an *inlier*), or should be considered as different (it is an *outlier*). Often, this ability is used to clean real data sets. Two important distinctions must be made:

outlier detection:	The training data contains outliers which are defined as observations that are far from the others. Outlier detection estimators thus try to fit the regions where the training data is the most concentrated, ignoring the deviant observations.
novelty detection:	The training data is not polluted by outliers and we are interested in detecting whether a **new** observation is an outlier. In this context an outlier is also called a novelty.

other methods for outliers detection in scikit-learn