

Swiss Institute of  
Bioinformatics

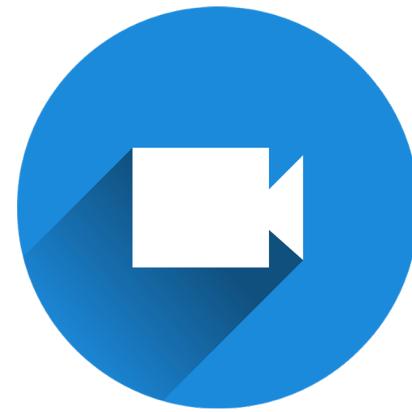
# SIB Swiss Institute of Bioinformatics

Introduction to Machine Learning with Python, 7 - 8 June 2021

*Isabelle Dupanloup and Markus Mueller*

# Course etiquette

---



Video on when  
possible

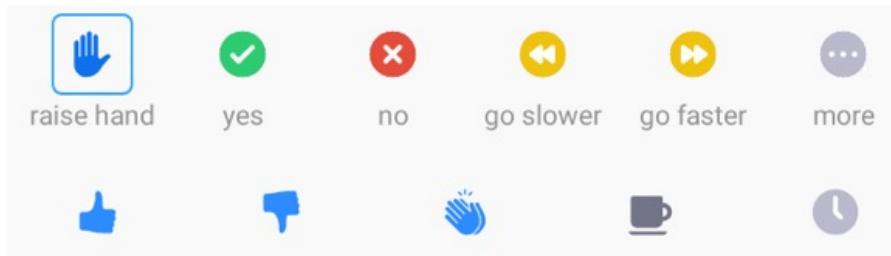


Mute when not  
speaking

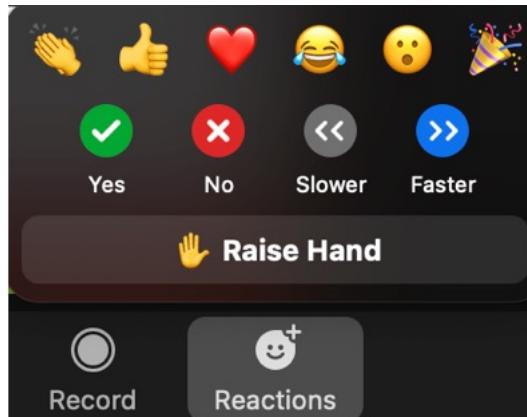
# Asking question

---

During lectures, you are encouraged to raise your hand if you have questions (if in-person), or use the “raise hand” Zoom functionality (if online). Find the buttons in the participants list (‘Participants’ button):



Alternatively, (depending on your zoom version or OS) use the ‘Reactions’ button:



# Code of conduct

SIB abides by the ELIXIR Code of Conduct. We are all thus expected to abide by the same code. In summary:

We **value** each other's perspectives providing a safe environment for people to be themselves.

We will **maintain** high ethical standards across all ELIXIR events.

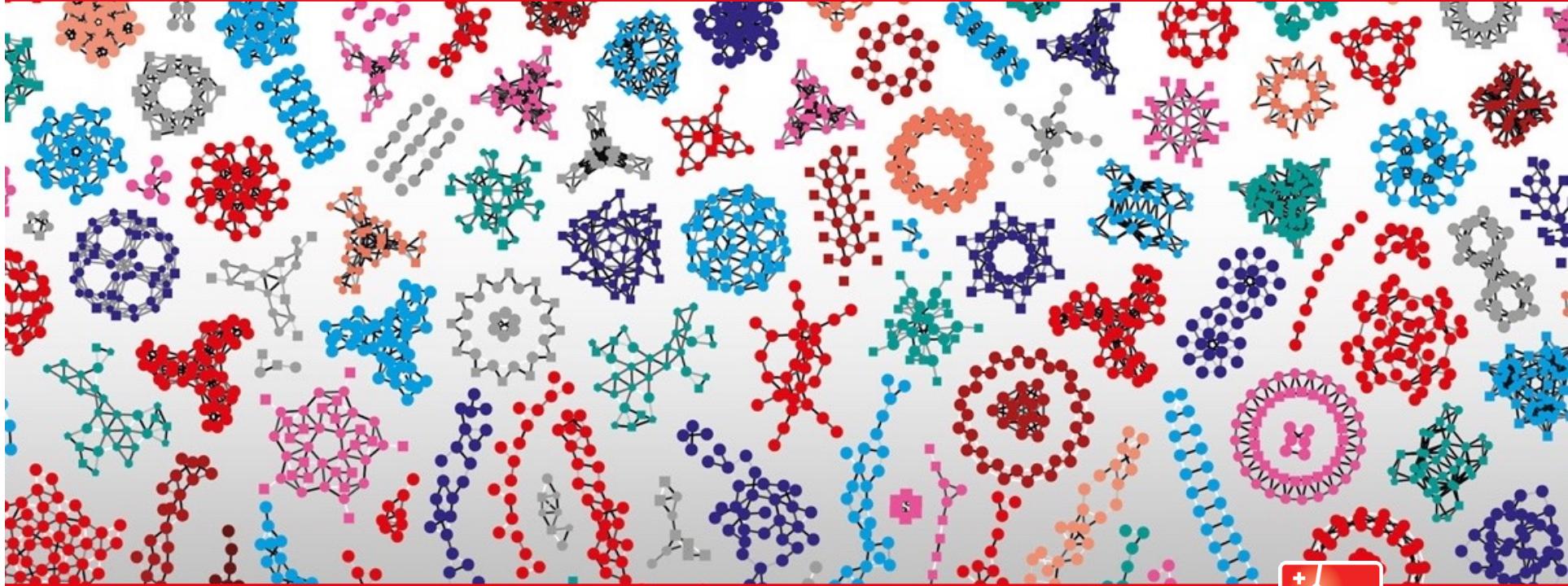
We **adopt** a zero-tolerance approach to harassment and discrimination in any form.

We will **apply** honesty and integrity in the dealing of any transgressions against the Code.

We are **committed** to making ELIXIR events a collaborative, supportive and enjoyable experience.

We will **ensure** that our environment allows everyone to feel respected and included.





Swiss Institute of  
Bioinformatics

# Thank you

Follow our activities:



# A quick round of introduction



**Isabelle Dupanloup**

- Bioinformatician and Trainer
- Bioinformatics Core Facility, SIB Swiss Institute of Bioinformatics



**Markus Mueller**

- Principal Computational Biologist
- Vital-IT, SIB Swiss Institute of Bioinformatics



**Wandrille Duchemin**

- Bioinformatics Support and Training
- SIB Swiss Institute of Bioinformatics / sciCORE UNIBAS



**Thuong Van Du Tran**

- Staff scientist
- Vital-IT, SIB Swiss Institute of Bioinformatics



**Sébastien Boyer**

- Bioinformatician and Trainer
- SIB Swiss Institute of Bioinformatics / sciCORE UNIBAS

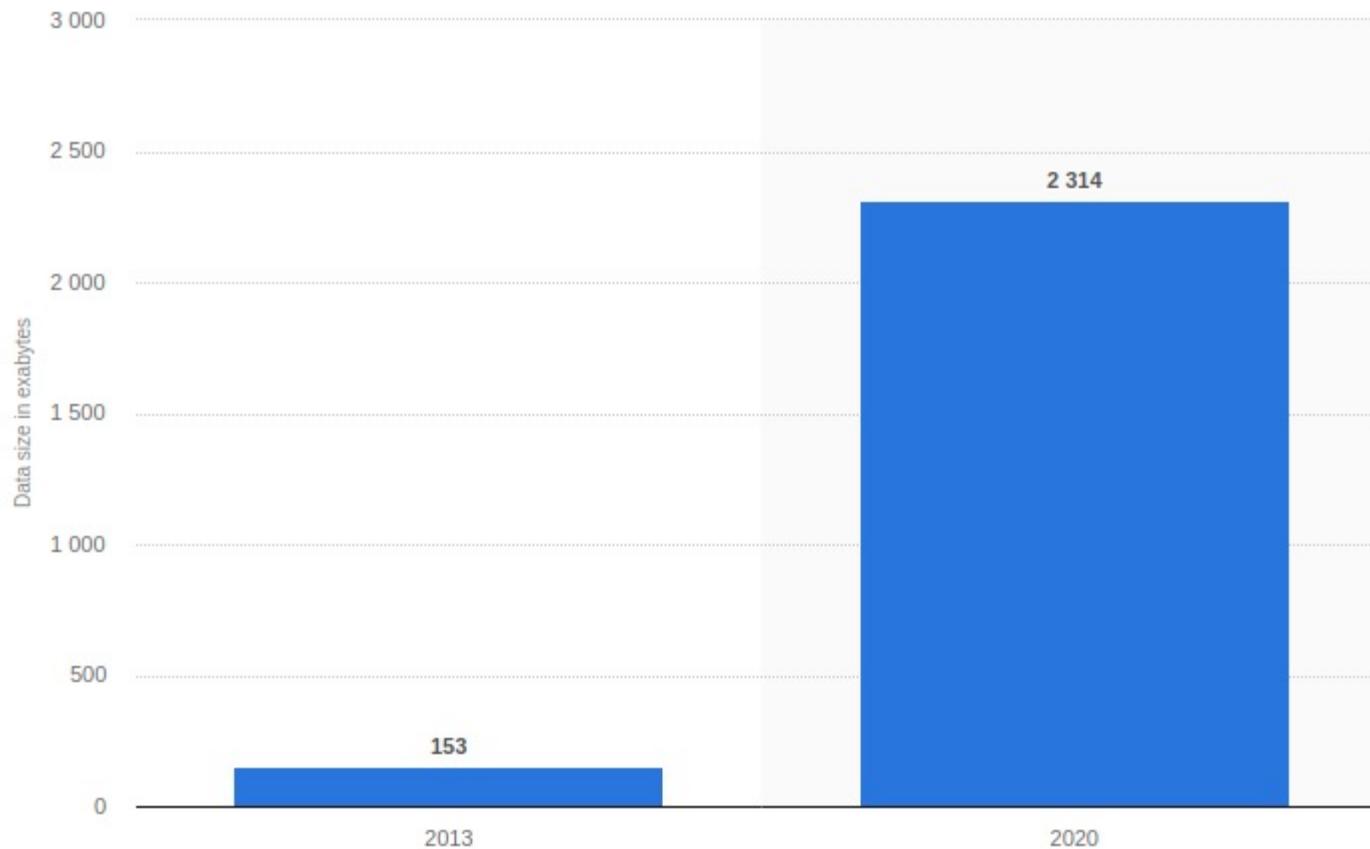
**Monique Zahn**

- Training Manager and neXtProt Quality Manager
- SIB Swiss Institute of Bioinformatics

# Course agenda

09:00 - 09:30	<b>Day 1: Course Introduction</b>	09:00 - 09:30	<b>Welcome Day 2</b>
09:30 - 10:00	<b>Introduction to Machine Learning (theory) (Isa)</b>	- Questions from Day 1	
	<b>Exploratory Analysis and Unsupervised Learning (theory)</b>	- Agenda	
10:00 - 10:30	- Data preparation for ML (Markus)	09:30 - 10:30	<b>Supervised Learning (practical) (Markus)</b>
	- Dimensionality reduction/Embedding (Markus)	10:30 - 11:00	<i>Coffee Break</i>
10:30 - 11:00	<i>Coffee Break</i>		<b>Supervised Learning (theory)</b>
11:00 - 12:00	<b>Exploratory Analysis and Unsupervised Learning (practical) (Markus)</b>	- Decision trees (Isa)	
12:00 - 12:30	<b>Exploratory Analysis and Unsupervised Learning (theory)</b>	- Random Forests (Isa)	
	- Clustering methods (Markus)	- Ada Boost (Isa)	
12:30 - 13:30	<i>Lunch break</i>	- Gradient boosting (Isa)	
13:30 - 14:00	<b>Exploratory Analysis and Unsupervised Learning (practical) (Markus)</b>	11:30 - 12:30	<b>Supervised Learning (practical) (Isa)</b>
	<b>Supervised Learning (theory)</b>	12:30 - 13:30	<i>Lunch break</i>
14:00 - 14:30	- Introduction (Isa)		<b>Supervised Learning (theory)</b>
	- Performance metrics (Isa)	13:30 - 14:30	- Deep learning (Isa)
	- Linear regression (Isa)		- Neural networks (Isa)
	- K-nearest neighbors (Markus)	14:30 - 15:00	<i>Coffee Break</i>
	- Logistic regression (Markus)	15:00 - 16:30	<b>Supervised Learning (practical) (Isa)</b>
	- Support vector machines (Markus)	16:30 - 17:00	<i>Closing questions, Discussion</i>
14:30 - 15:00	<i>Coffee Break</i>		
15:00 - 17:00	<b>Supervised Learning (practical) (Markus)</b>		

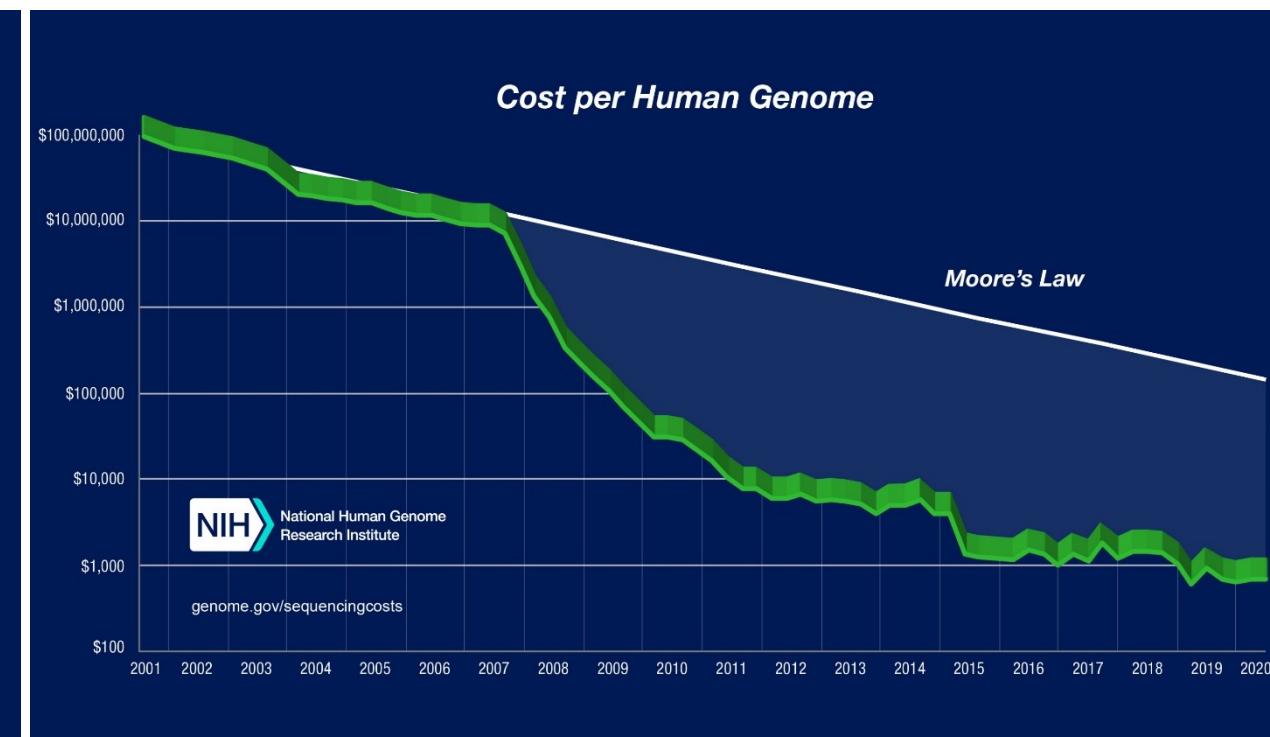
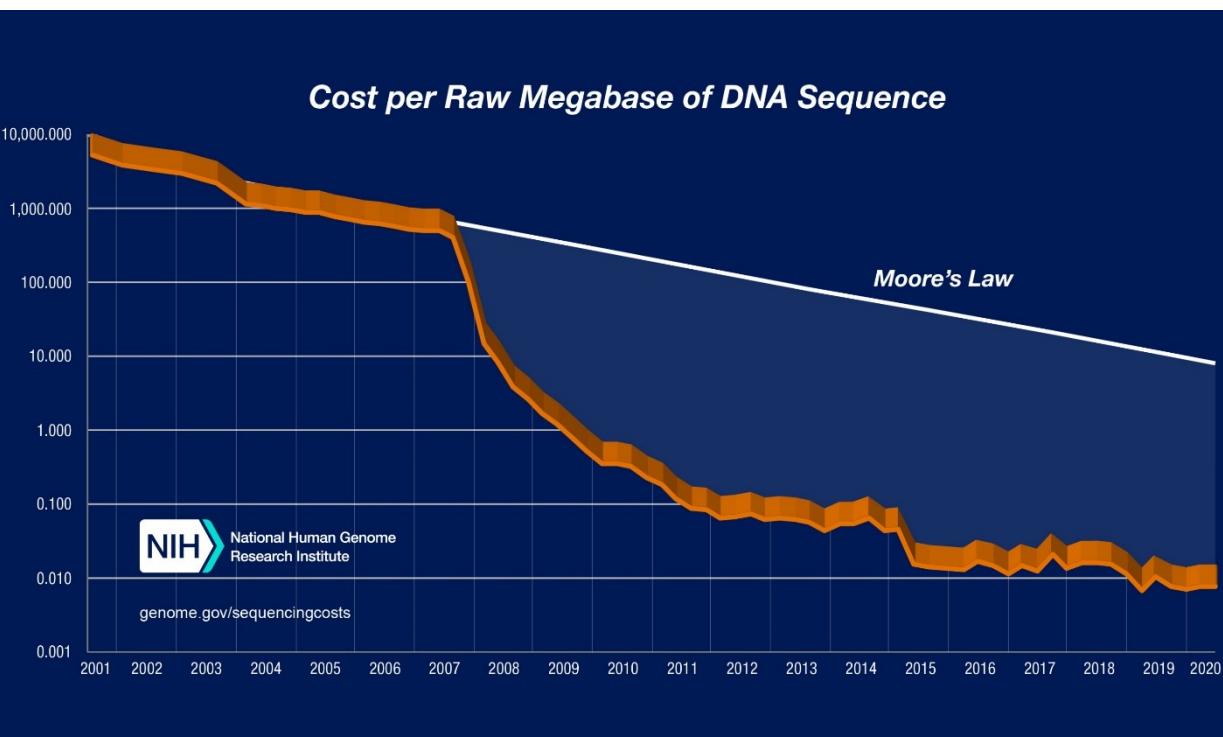
# Total amount of global healthcare data generated and projections for end 2020 (in exabytes)



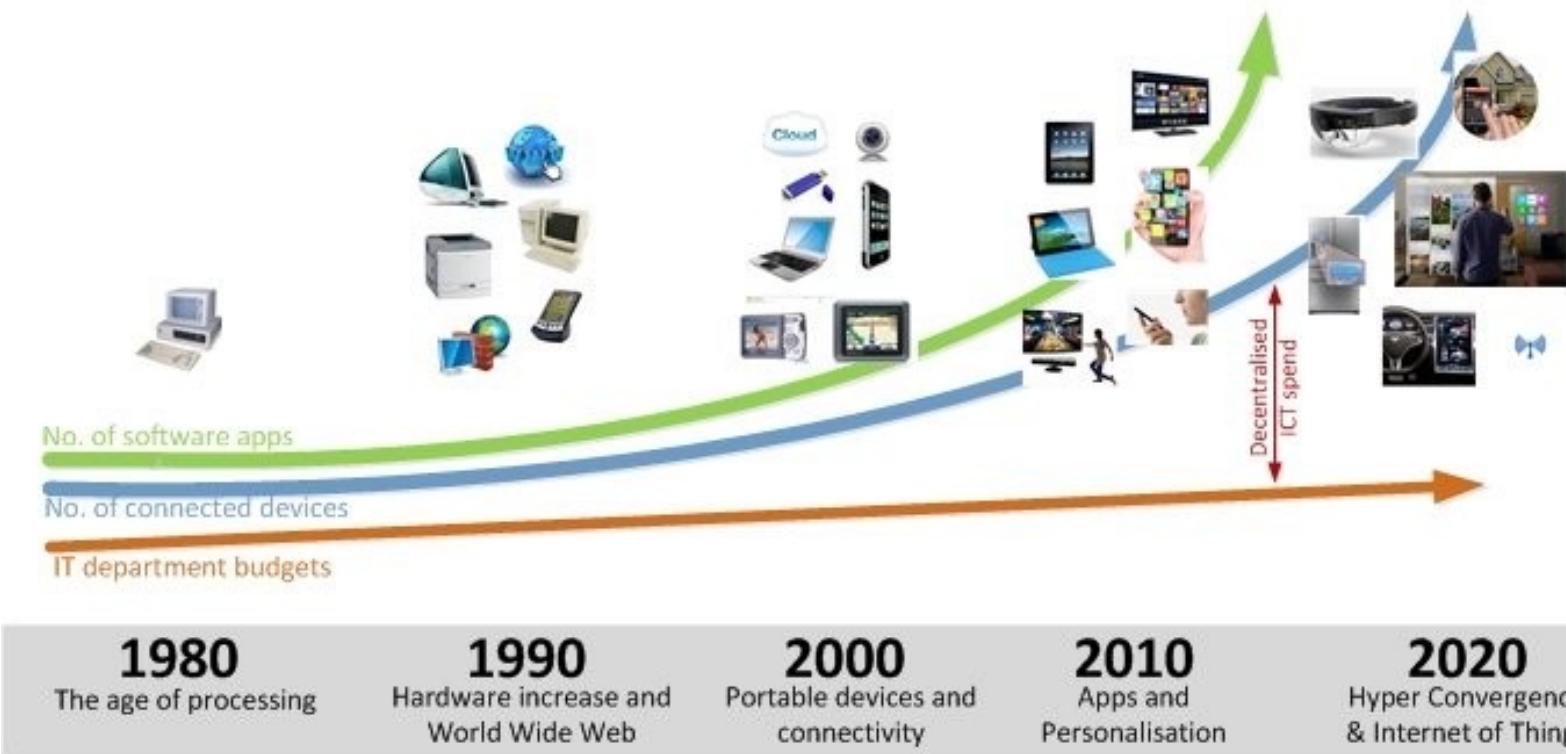
1 exabyte = 1 billion  
gigabytes

<https://www.statista.com/statistics/1037970/global-healthcare-data-volume/>

Isabelle Dupanloup, BCF



## Technology Timeline



<https://www.linkedin.com/pulse/technology-increase-vs-department-budgets-sam-errington/>

# From data to knowledge

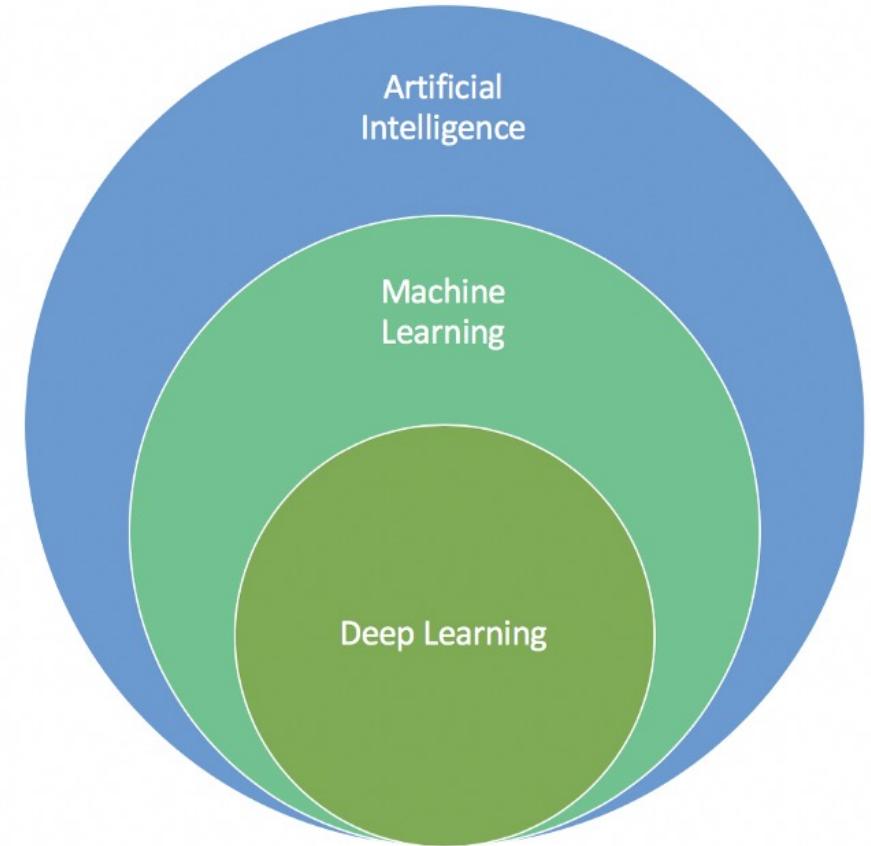


# Some definitions

- Artificial Intelligence
  - programmed by humans to perform human activities
  - computer systems (AI systems): units of “thinking machines”
    - General AI Systems can solve problems intelligently
    - Narrow AI Systems can perform specific tasks very well
- AI systems
  - Intentionality (make decisions from data, predetermined responses)
  - Intelligence (intelligent decision making)
  - Adaptability (refine their decision-making capabilities to improve the outcome)

# Some definitions

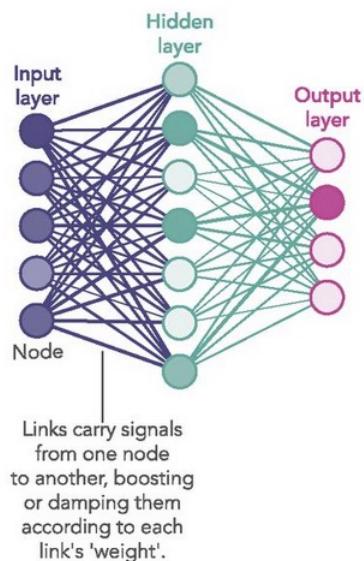
- Machine learning
  - application (or subset) of AI that allows machines to learn from data without being programmed explicitly
  - variety of algorithms to iteratively learn, describe and improve data in order to predict better outcomes
  - extensive use of statistical techniques



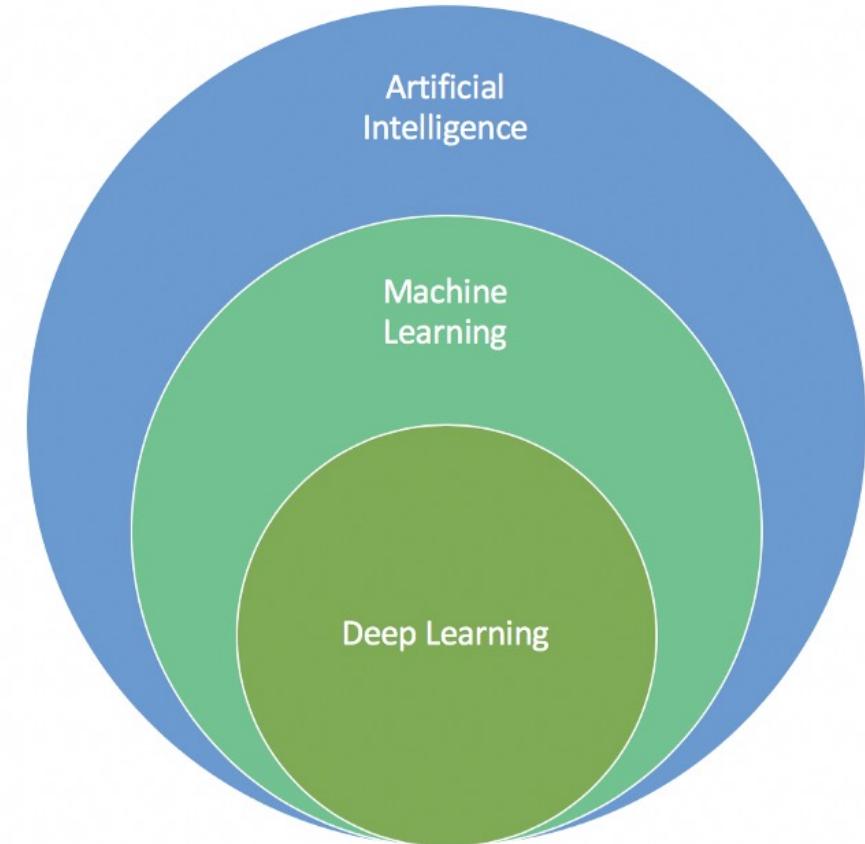
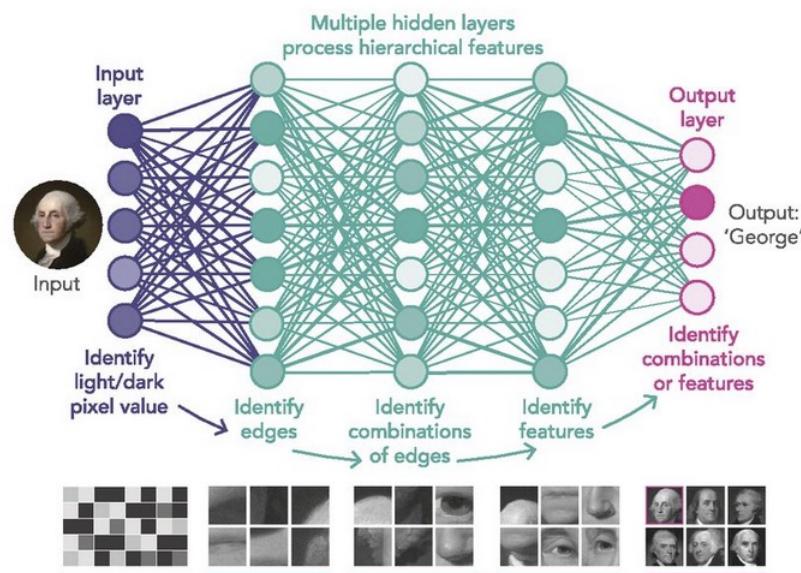
# Some definitions

- Deep Learning
  - subset of Machine Learning
  - use artificial neural networks
  - analyzes data with a logical structure similar to how a human would draw conclusions

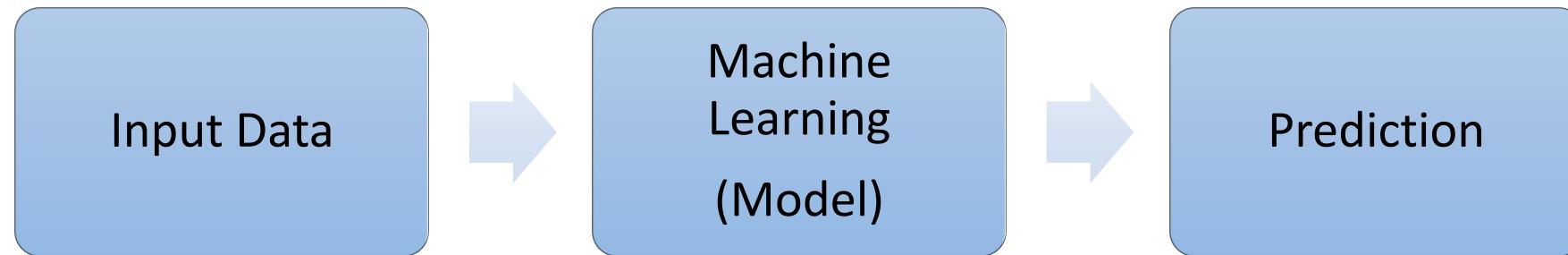
1980S-ERA NEURAL NETWORK



DEEP LEARNING NEURAL NETWORK

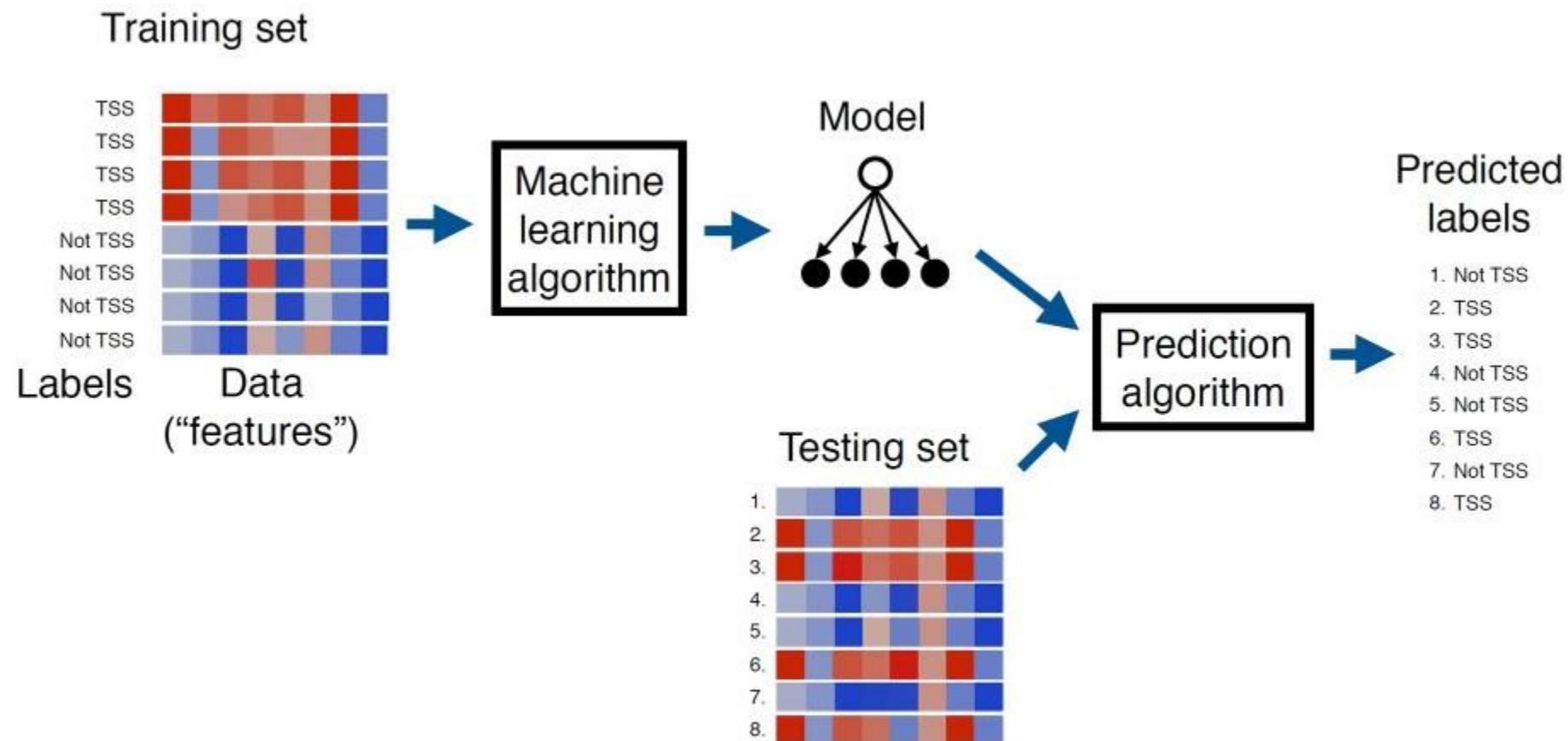


# Machine learning



- Learning begins with observations or data
- The system looks for patterns in data and makes better decisions in the future based on the examples that we provide
- The primary aim is to allow the computers learn automatically without human intervention or assistance and adjust actions accordingly

# Machine learning in omics



# Machine learning

Any machine learning problem can be represented with the following three concepts:

- We will have to learn to solve a task T.
  - For example, perform genome annotation.
- We will need some experience E to learn to perform the task. Usually, experience is represented through a dataset.
  - For the gene prediction, experience comes as a set of DNA sequences provided as input to a learning procedure, along with binary labels indicating whether each sequence is centered on a TSS or not. The learning algorithm produces a model which can then be subsequently used, in conjunction with a prediction algorithm, to assign predicted labels to unlabeled test sequences.
- We will need a measure of performance P to know how well we are solving the task and also to know whether after doing some modifications, our results are improving or getting worse.
  - The percentage of genes that our gene prediction model is correctly classifying as genes could be P for our gene prediction task.

# Machine learning in omics

## Input data

DNA sequence

DNA sequence

DNA sequence

Gene expression

Gene expression data

Histone and TF ChIP-seq data

DNA sequence + gene expression + ...

DNA sequence + histone mods + ...

DNA sequence + histone mods + ...

Sequence variants + gene expression + ...

## Task

Identify transcription start sites, splice sites, exons, etc.

Identify TF binding sites

Identify genes

Predict regulatory relationships

Identify biomarkers for a disease

Partition and label the genome with chromatin state annotation

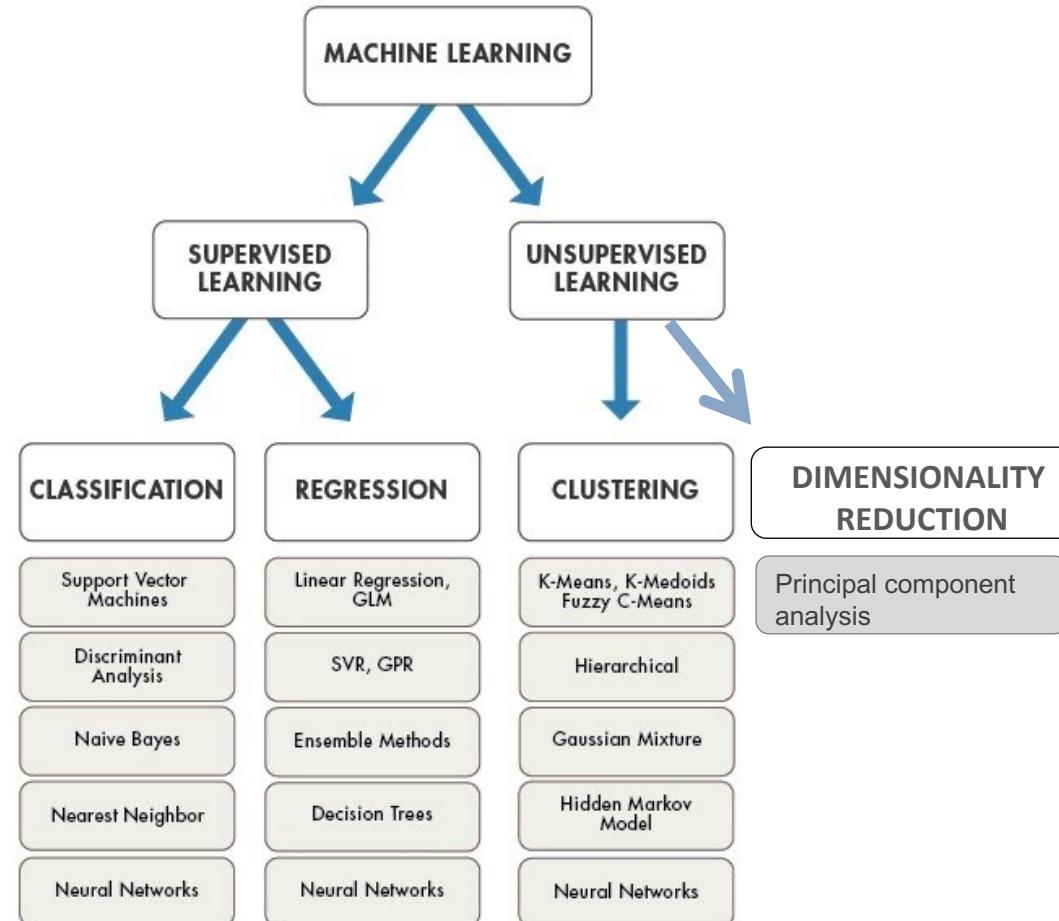
Predict gene function

Predict gene expression

Predict variant deleteriousness

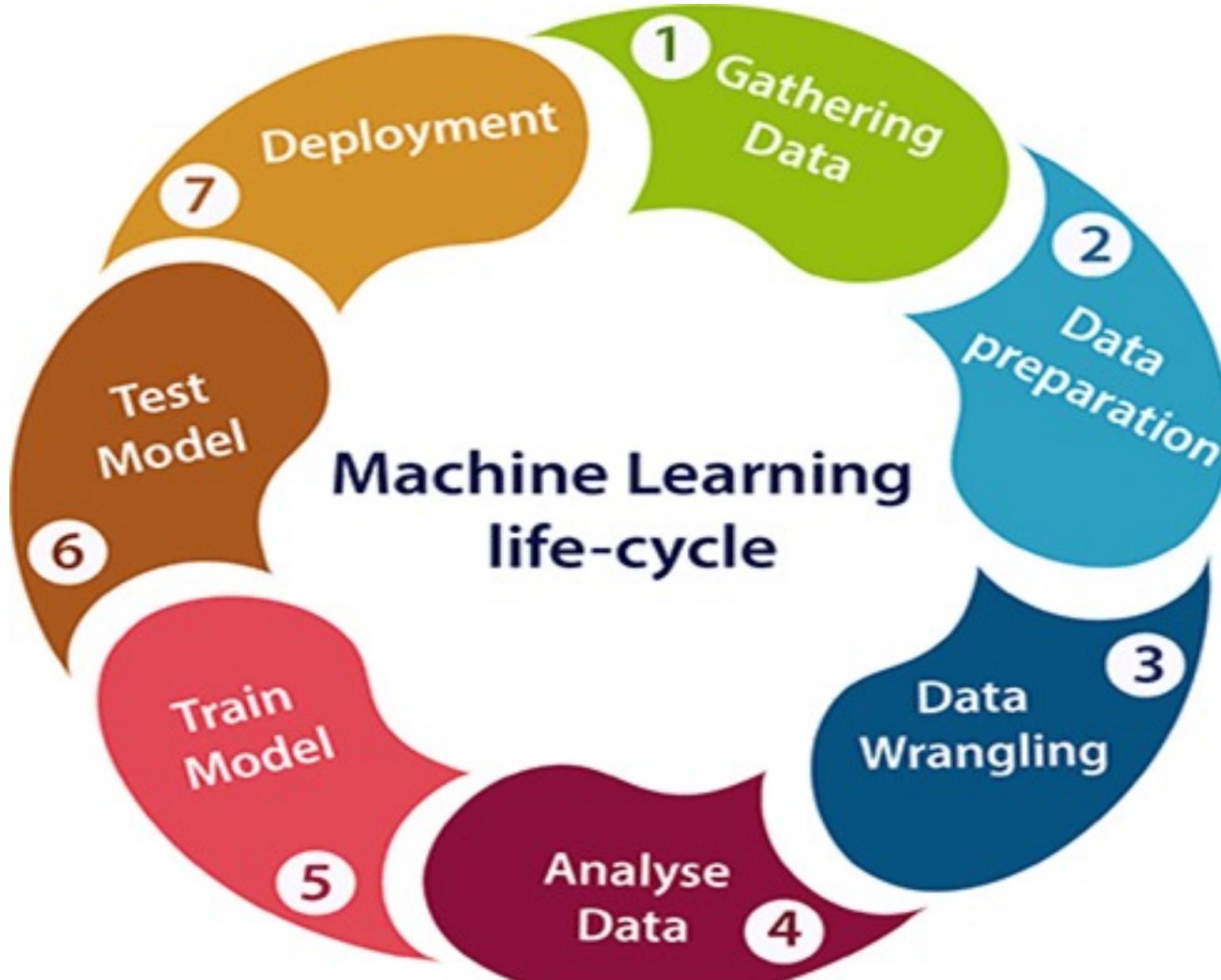
Predict disease phenotype or prognosis

# Machine learning taxonomy



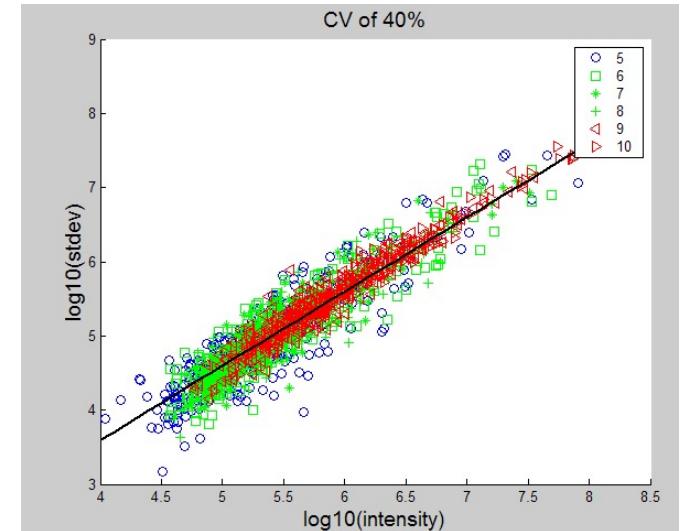
# Supervised versus unsupervised learning

- Supervised learning
  - Machine learning algorithm that is trained on labeled examples and used to predict the label of unlabeled examples.
  - Prediction limited to input labels
- Unsupervised learning
  - Machine learning algorithm that does not require labels, such as a clustering algorithm.
  - Prediction of new labels
- Semi-supervised learning
- Reinforcement Learning

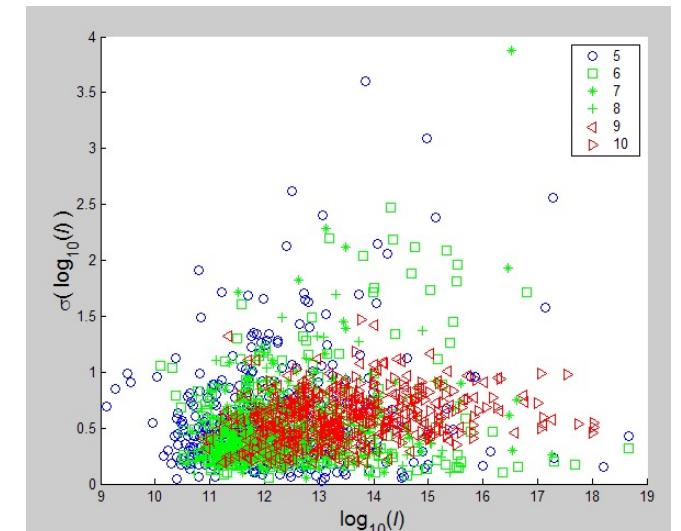


# Variance Stabilization

- In Omics data the standard deviation  $\sigma$  of a signal often grows with increasing signal value.
- This can detrimental for some ML techniques that assume that each feature value has the same error (e.g. k-means)
- If the dependency is linear a simple log-transform can stabilize the variance
- For more complex situations check [Huber et al. Bioinformatics 2002](#)
- Removing batch effects and confounding factors



$$\sigma = \alpha \cdot y \Rightarrow \log_{10}(\sigma) = \log_{10}(y) + \log_{10}(\alpha)$$



$$\sigma(\log_{10}(y)) = \text{const}$$



# Missing Values (MV)

- Omics data (especially proteomics) often have missing values, which can be caused by leaky measurement or data processing
- There are two main strategies to deal with MVs: discard or impute them
- MVs can occur at random or depend on feature values
  - For missing at random we can only replace or impute them by the best guess, the global non-missing mean ([Donders et al, J. Clin. Epi., 2006](#))
  - In the latter case, we can try to predict the MVs using the feature values by EM, KNN, SVD, or regression technique ([Troyanskaya et al, Bioinformatics, 2001](#))

# Unsupervised machine learning algorithms

- Dimensionality Reduction

Often we are working with data of high dimensionality - each observation comes with a high number of measurements - a dimension reduction procedure is usually conducted to reduce the variable space before the subsequent analysis is carried out..

- Clustering

= discover groupings inside the input data (example: cluster DNA sequences into functional groups).

- Association

= discover rules inside the input data (learn relationships between the measurements)

# Some Notations:

*M features or conditions*

*N items*

*N × M*

*Data matrix:*  $X = (x_1, x_2, \dots, x_N)^T = \begin{pmatrix} x_{11} & \cdots & x_{1M} \\ \vdots & \ddots & \vdots \\ x_{N1} & \cdots & x_{NM} \end{pmatrix}$

*Zero centering:*  $m_j = \frac{1}{N} \sum_{i=1}^N x_{ij}, x_{ij} \mapsto x_{ij} - m_j$       *Columns have 0 means*

# Principal components analysis (PCA)

- Mathematical algorithm that reduces the dimensionality of the data while retaining most of the variation in the data set
- Identifies directions, called principal components, along which the variation in the data is maximal
- By using a few components, each sample can be represented by relatively few numbers instead of by values for thousands of variables.
- Samples can then be plotted, making it possible to visually assess similarities and differences between samples and determine whether samples can be grouped.

# Principal Component Analysis (PCA)

We try to find a direction  $\mathbf{w}$  along which the data has the largest variance:

$$\mathbf{w} = \underset{\|\mathbf{w}\|=1}{\operatorname{argmax}} \sum_{i=1}^N (\mathbf{x}_i \cdot \mathbf{w})^2$$

$$\sum_{i=1}^N (\mathbf{x}_i \cdot \mathbf{w})^2 = \|\mathbf{X}\mathbf{w}\|^2 = \mathbf{w}\mathbf{X}^T\mathbf{X}\mathbf{w}$$

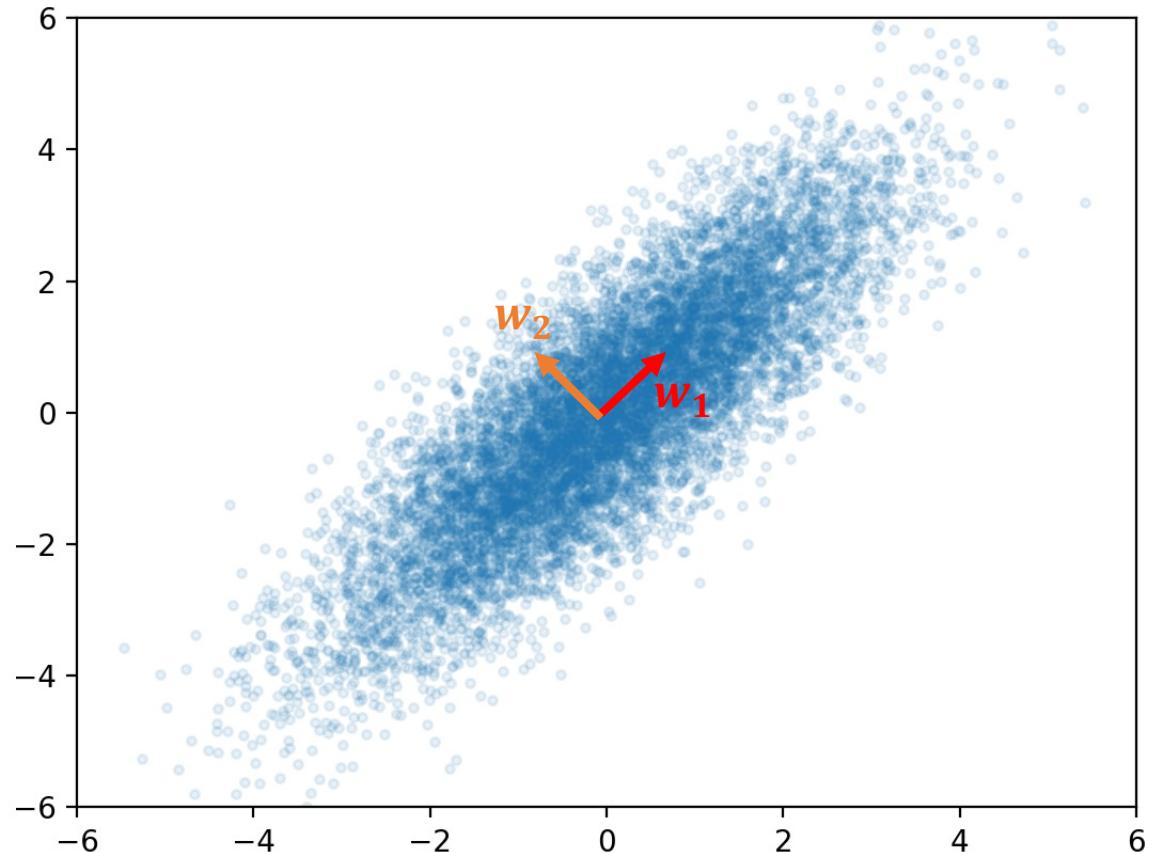
$$\frac{\partial}{\partial \mathbf{w}} (\mathbf{w}\mathbf{X}^T\mathbf{X}\mathbf{w} + \lambda \mathbf{w} \cdot \mathbf{w}) = 0 \Rightarrow \mathbf{X}^T\mathbf{X}\mathbf{w} = \lambda \mathbf{w}$$

$$\mathbf{X}^T\mathbf{X} = \mathbf{W}\mathbf{D}\mathbf{W}^T, \mathbf{W} = (\mathbf{w}_1 \cdots \mathbf{w}_M)$$

$\{\mathbf{w}_1 \cdots \mathbf{w}_M\}$  = Eigenvectors or PCs of cov. matrix  $\mathbf{X}^T\mathbf{X}$

$$\lambda_j = \sigma_j^2, \sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_M, \mathbf{D} = \begin{pmatrix} \sigma_1^2 & \cdots & 0 \\ \vdots & \sigma_j^2 & \vdots \\ 0 & \cdots & \sigma_M^2 \end{pmatrix}$$

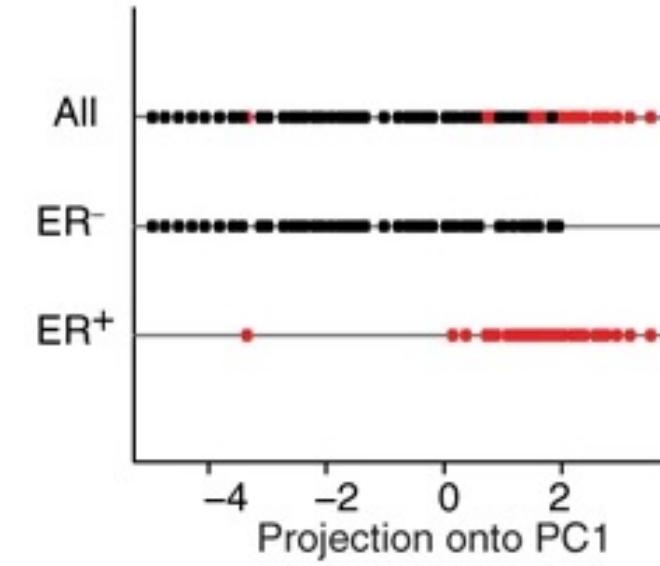
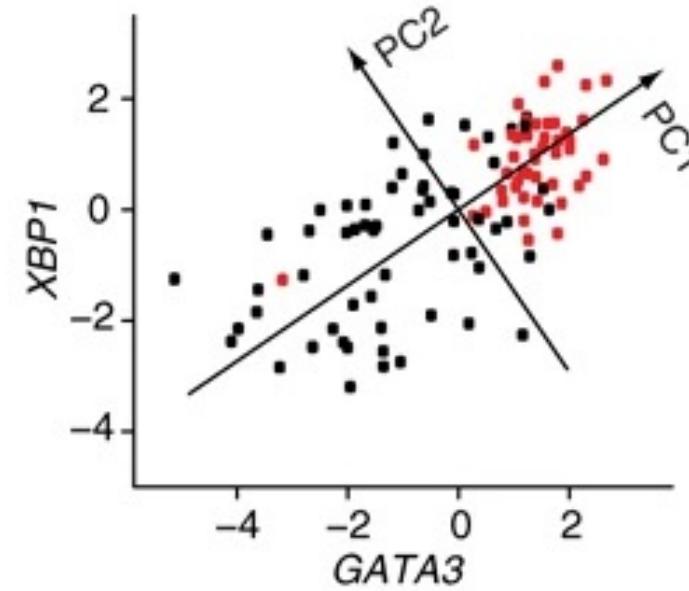
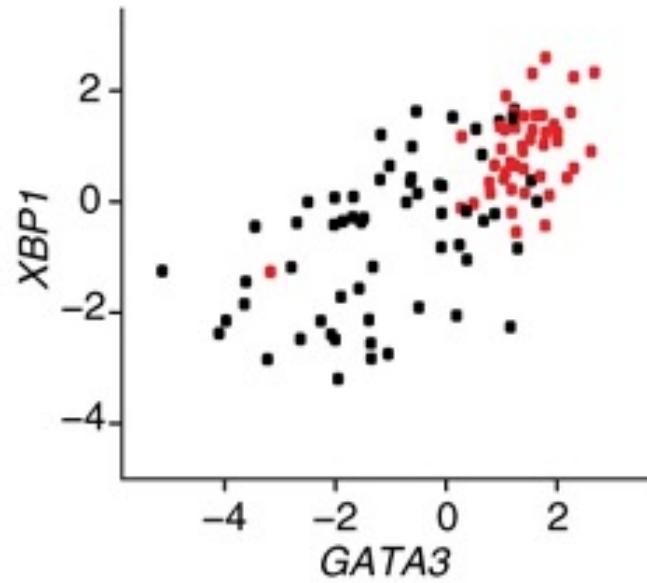
*Zero centered  $\mathbf{X}$*



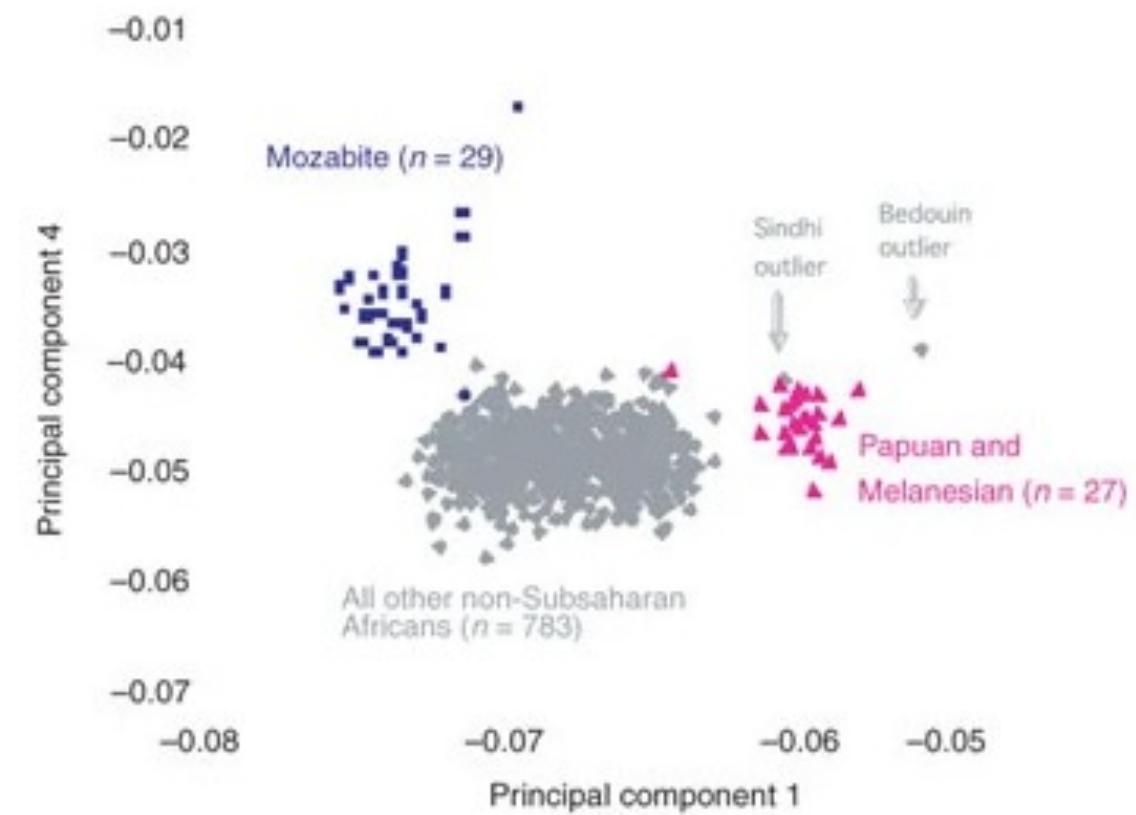
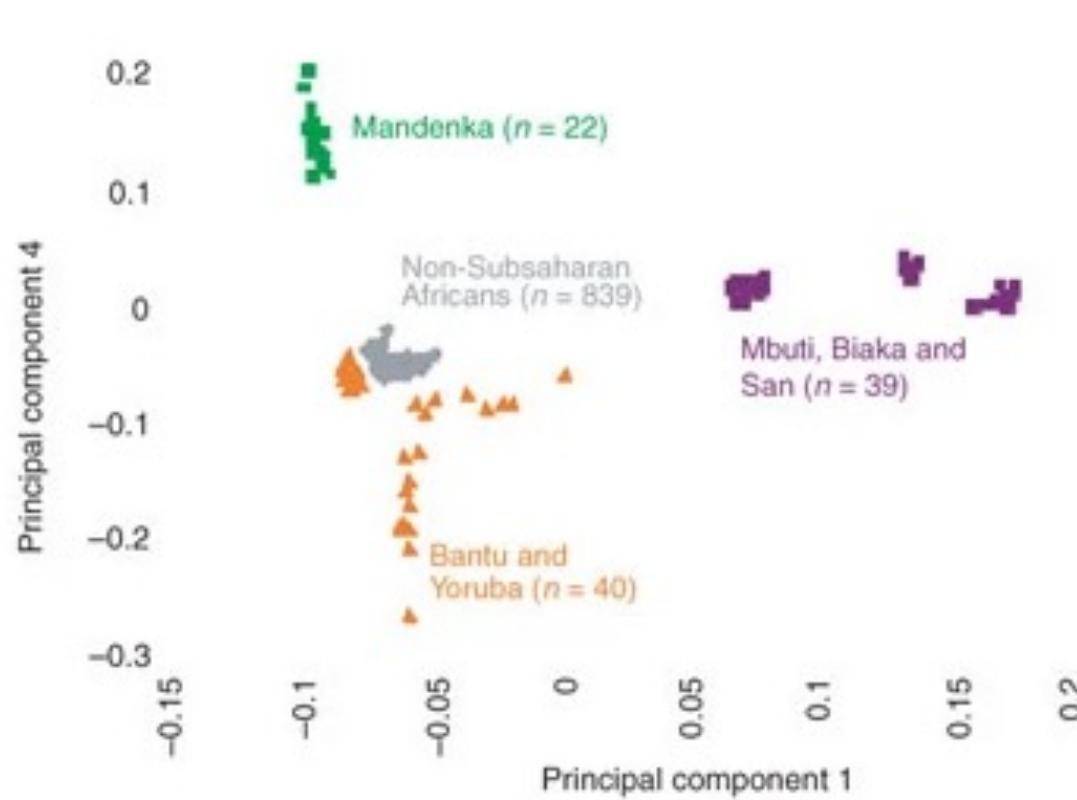
# Principal Component Analysis (PCA)

- Dimension reduction and visualization
- Decorrelate or whiten signals
- Data compression

# Principal components analysis (PCA)



# Principal components analysis (PCA)



Reich et al. (2008) Nature Genetics.

Isabelle Dupanloup, BCF

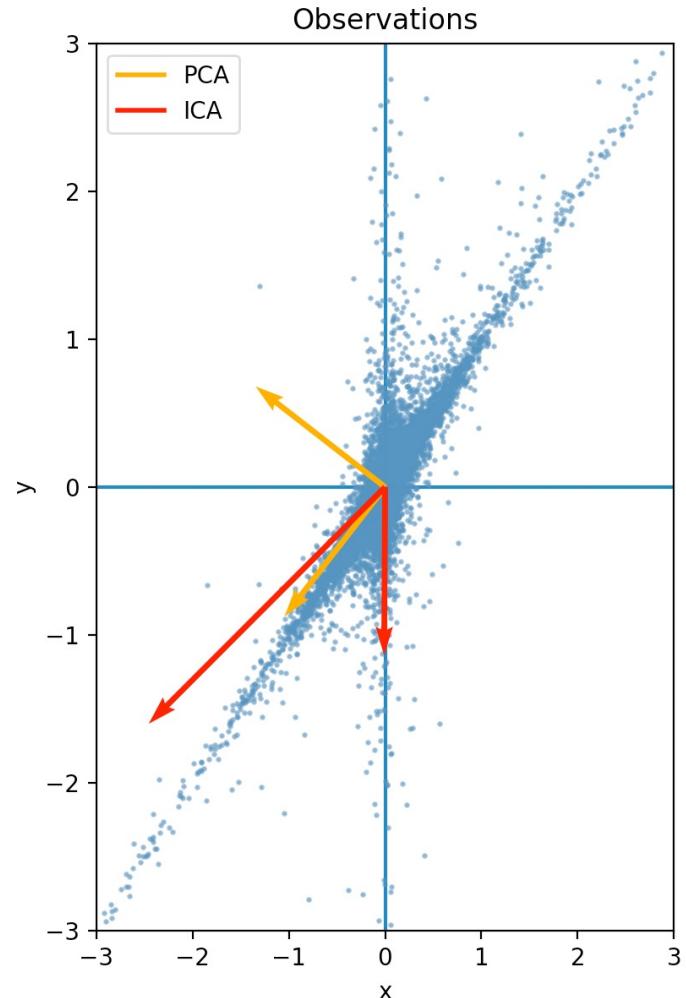
# Independent Component Analysis (ICA)

We try to find direction  $\mathbf{w}$  along which the data deviates the most from a Gaussian distribution:

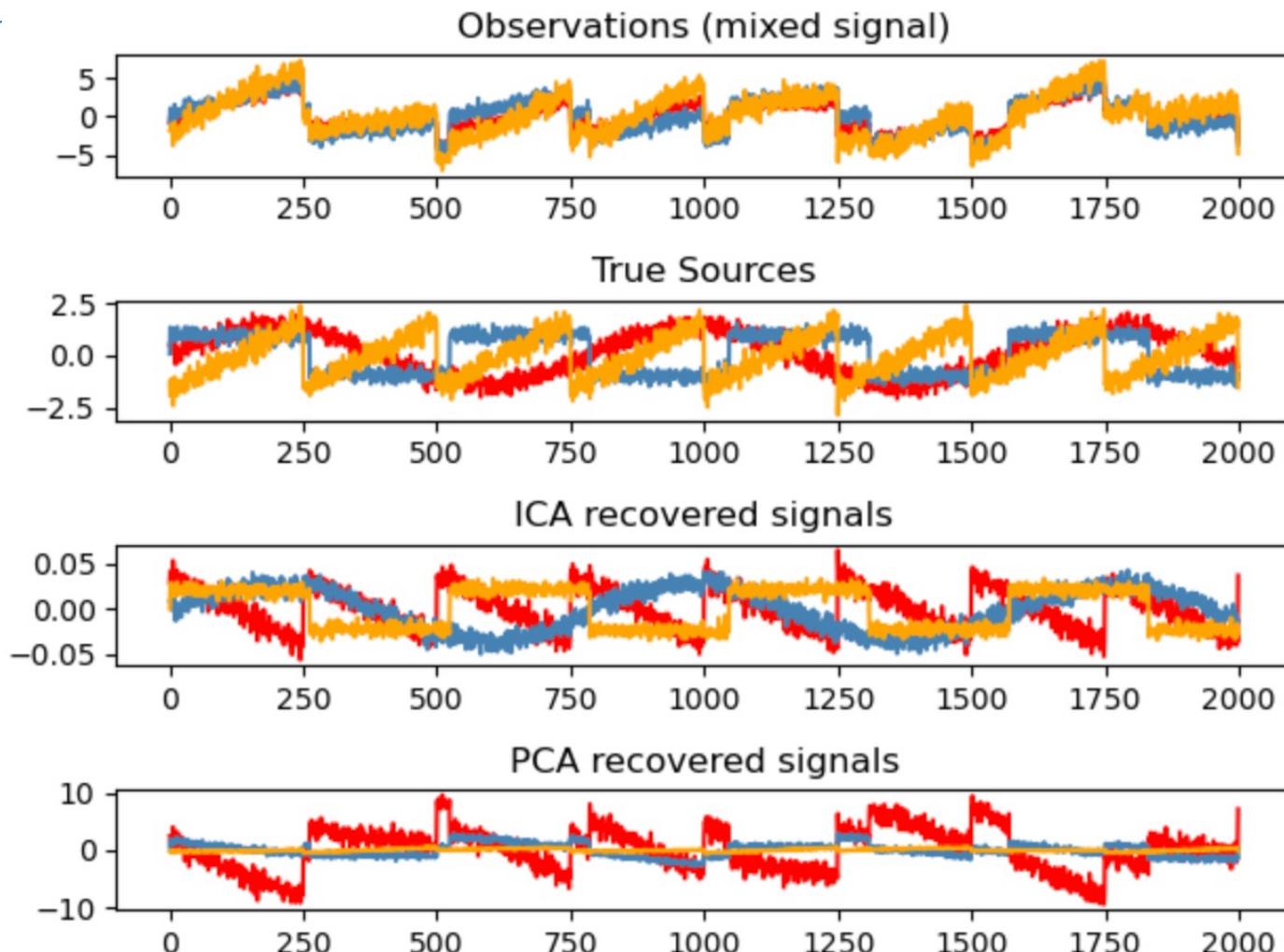
$$\mathbf{w} = \underset{\|\mathbf{w}\|=1}{\operatorname{argmax}} DG(t_1, t_2, \dots, t_N), t_i = \mathbf{x}_i \cdot \mathbf{w}$$

where  $DG(t_1, t_2, \dots, t_N)$  is a function that measures how far away  $\{t_1, t_2, \dots, t_N\}$  are from a Gaussian distribution.

Empirical functions  $DG$  were found that work well in practice and are fast to calculate ([Hyvärinen & Oja, Neural Networks, 2000](#)).



# Independent Component Analysis (ICA)



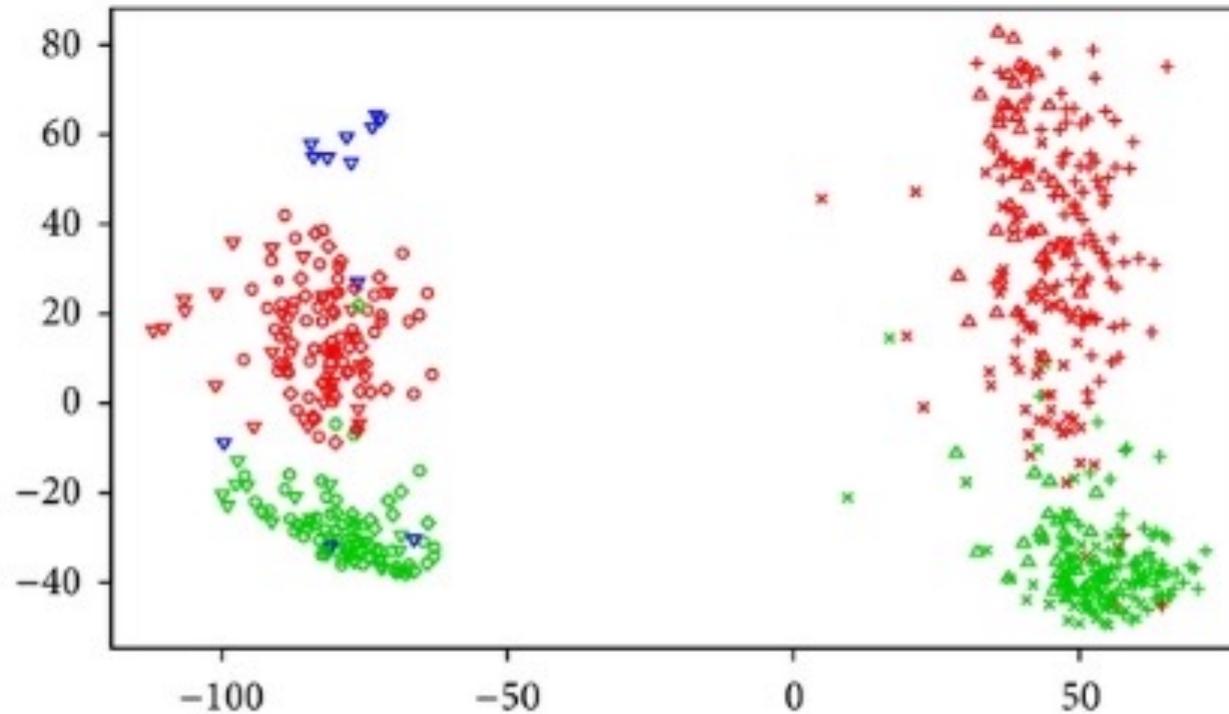
# Multidimensional scaling (MDS)

- visual representation of distances between sets of objects
- objects that are more similar (or have smaller distances) are closer together on the MDS than objects that are less similar (or have larger distances)
- MDS can also serve as a dimension reduction technique for high-dimensional data

# Multidimensional scaling (MDS)

- Step by step
  1. Assign points to arbitrary coordinates in p-dimensional space.
  2. Compute euclidean distances among all pairs of points
  3. Compare the similarity matrix with the original input matrix by evaluating the stress function. Stress is a goodness-of-fit measure, based on differences between predicted and actual distances.
  4. Adjust coordinates of each point in the direction that reduces stress.
  5. Repeat steps 2 through 4 until stress can't be minimized further.

# Multidimensional scaling (MDS)



Taminau et al. (2014) ISRN Bioinform.

- GSE10072
- ▼ GSE7670
- ♦ GSE31547
- ✖ GSE19804
- + GSE19188
- △ GSE18842
- Lung cancer
- Control
- Unknown

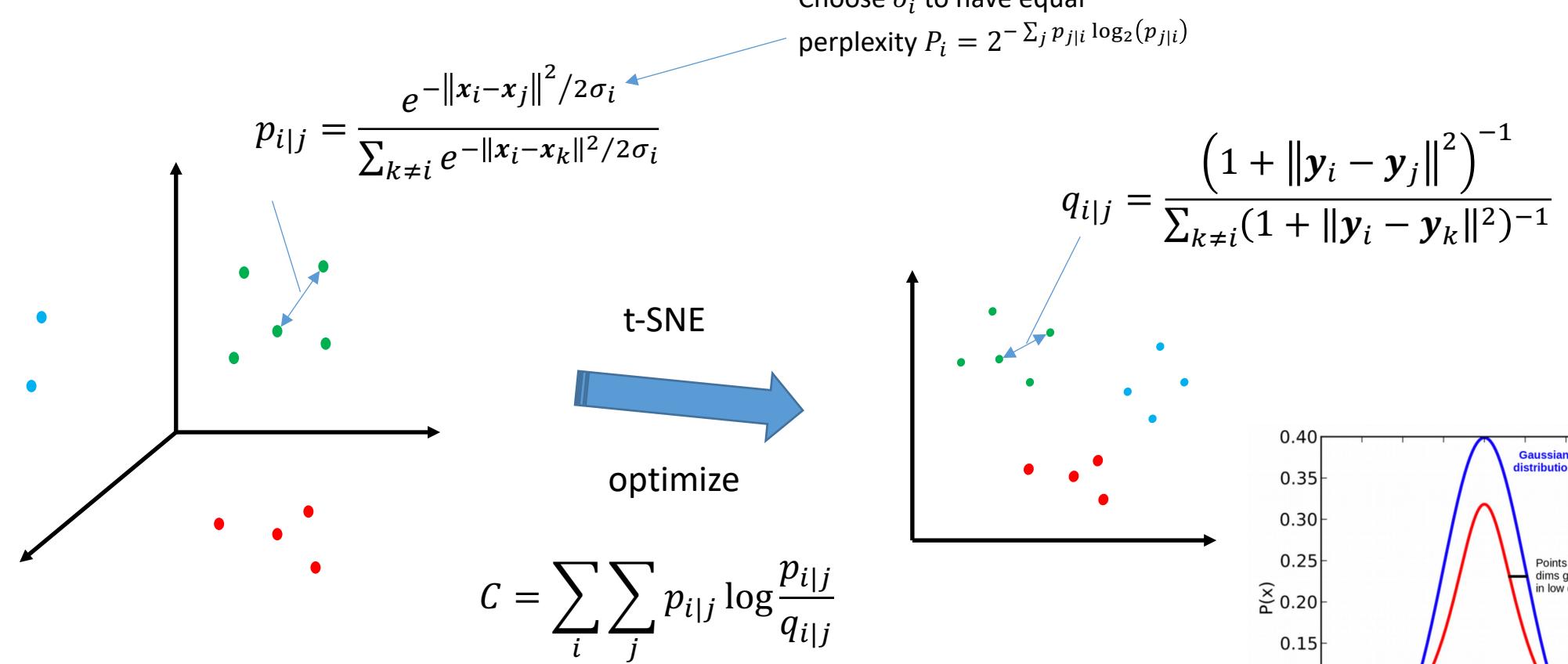
# t-distributed Stochastic Neighbor Embedding (t-SNE)

- unsupervised, non-linear technique primarily used for data exploration and visualizing high-dimensional data
- t-SNE differs from PCA by preserving only small pairwise distances or local similarities whereas PCA is concerned with preserving large pairwise distances to maximize variance
- maps a set of high-dimensional points to two dimensions, such that ideally, close neighbours remain close and distant points remain distant

# t-distributed Stochastic Neighbor Embedding (t-SNE)

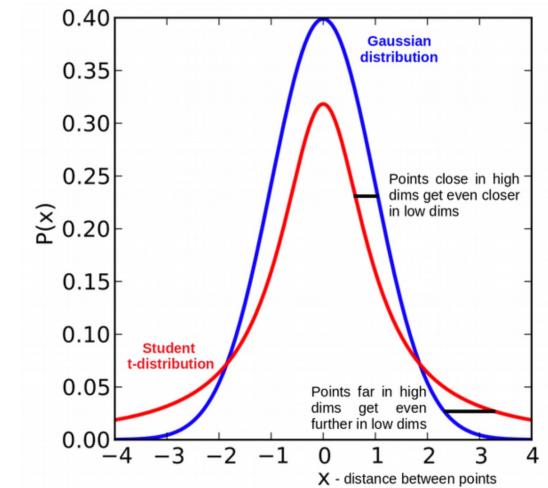
- Informally, the algorithm places all points on the 2D plane, initially at random positions, and lets them interact as if they were physical particles.
- The interaction is governed by two laws: first, all points are repelled from each other; second, each point is attracted to its nearest neighbours
- The most important parameter of t-SNE, called perplexity, determines how many of its nearest neighbours each point is attracted to.

# t-distributed Stochastic Neighborhood Embedding (t-SNE)

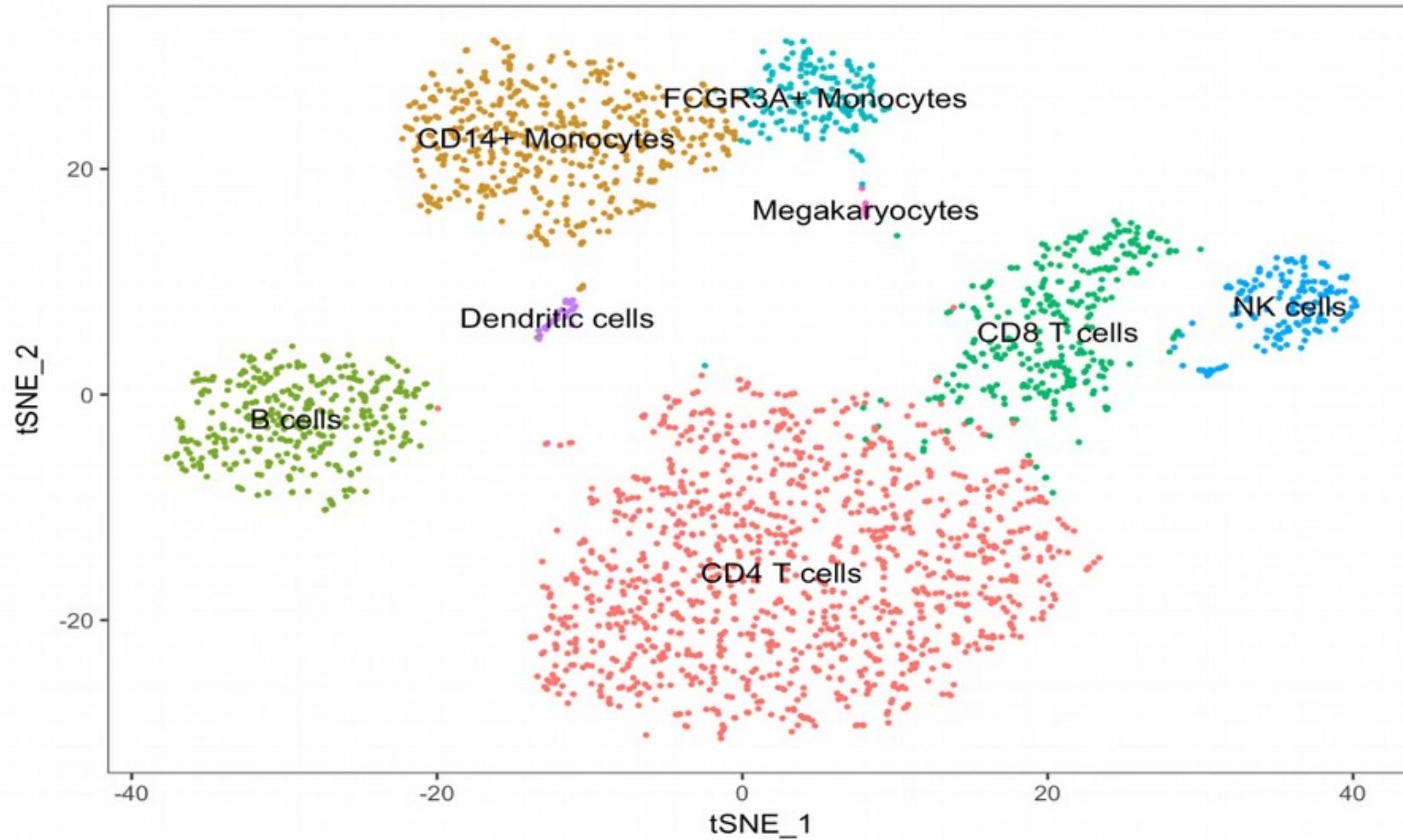


[Maaten & Hinton, JMLR, 2008](#)

Markus Müller, SIB



# t-distributed Stochastic Neighbor Embedding (t-SNE)

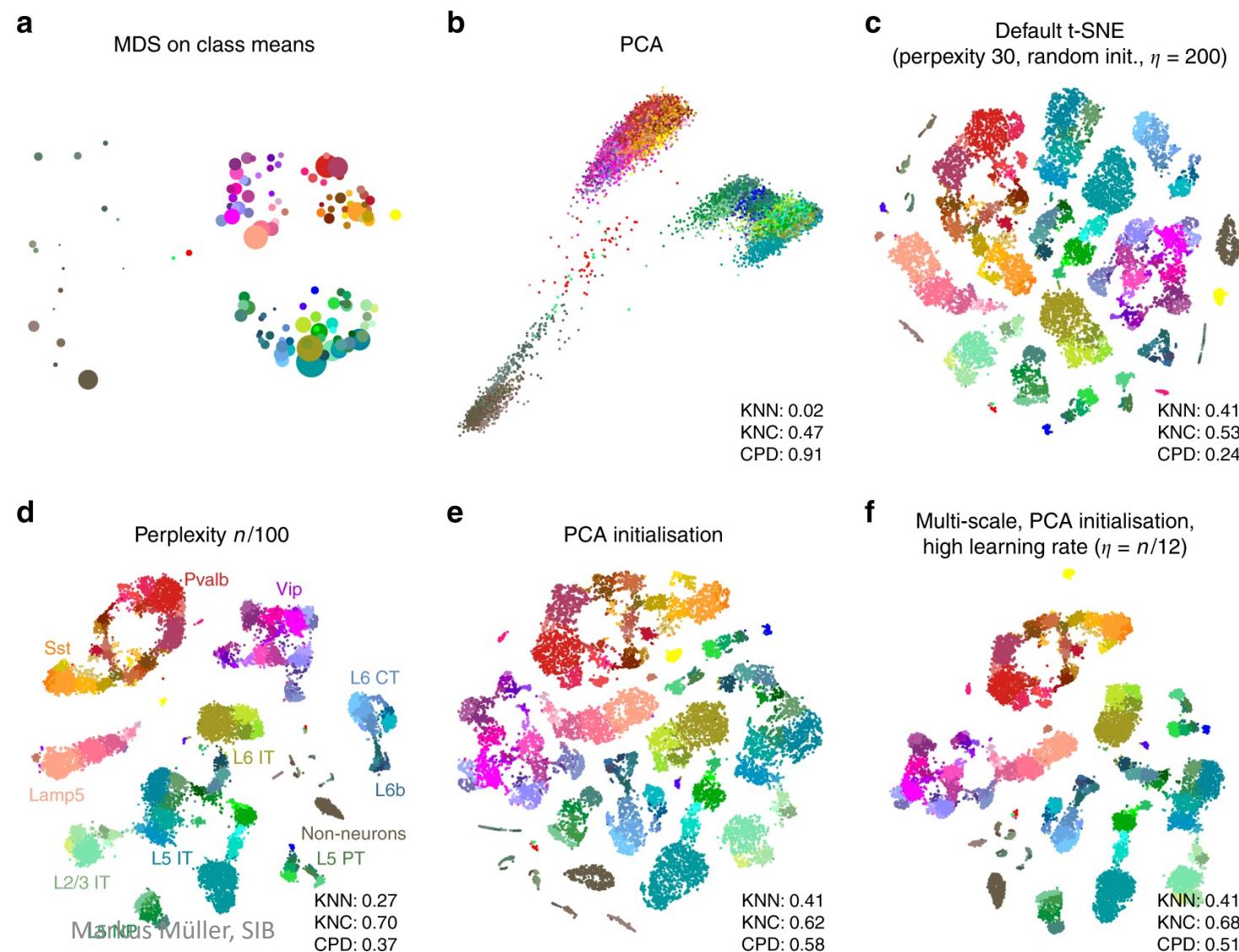


<https://towardsdatascience.com/how-to-tune-hyperparameters-of-tsne-7c0596a18868>

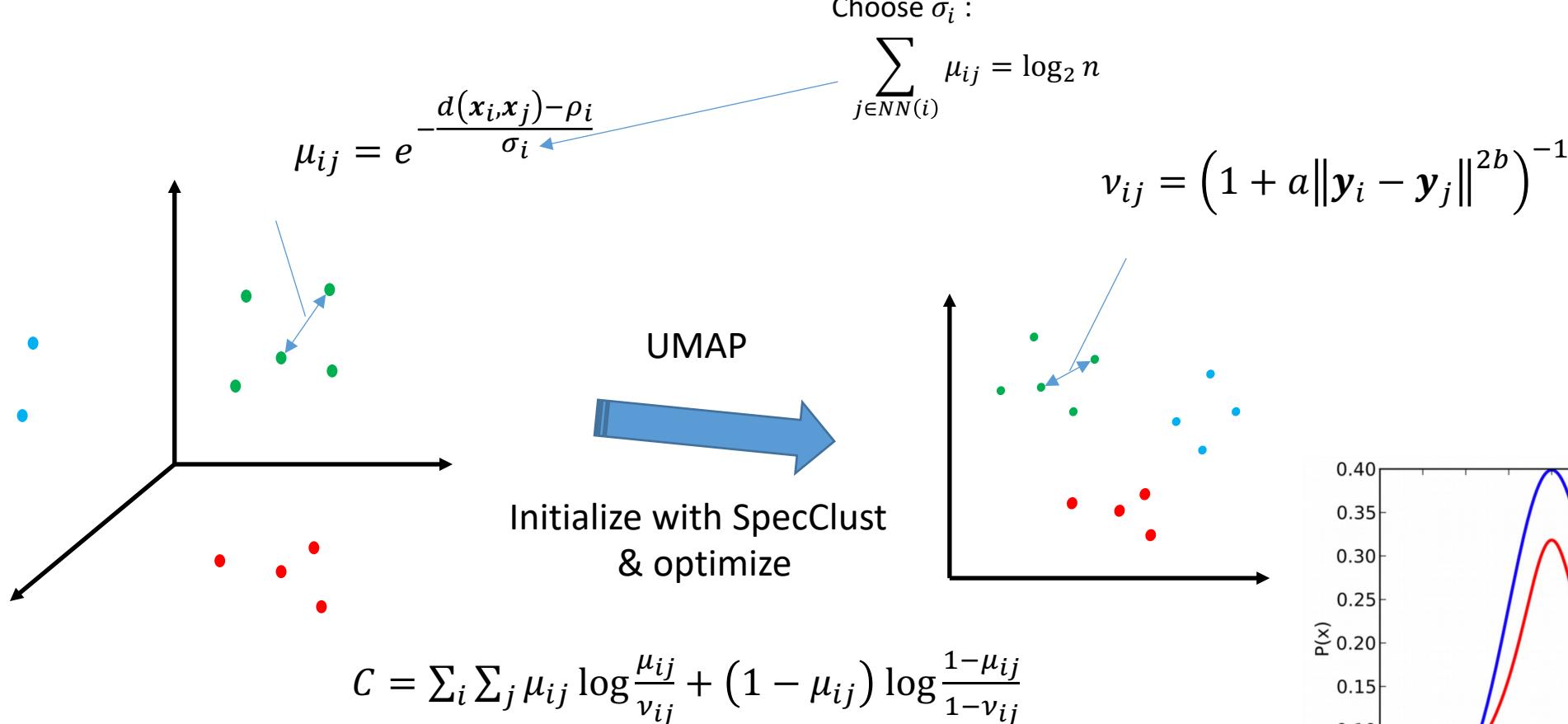
Isabelle Dupanloup, BCF

# t-distributed Stochastic Neighborhood Embedding (t-SNE)

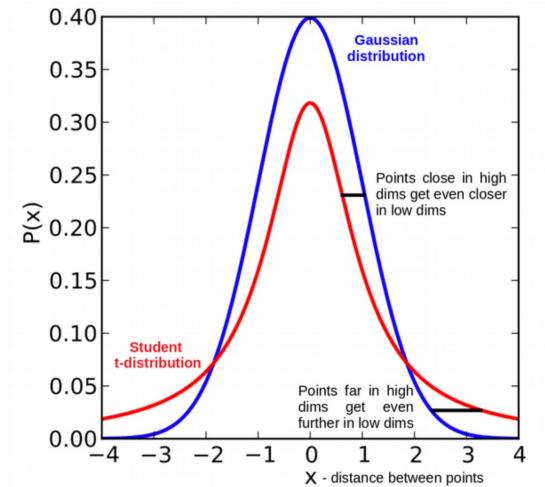
Single cell analysis  
[Kobab & Berens, Nature Comm., 2019](#)



# Uniform manifold approximation and projection (UMAP)

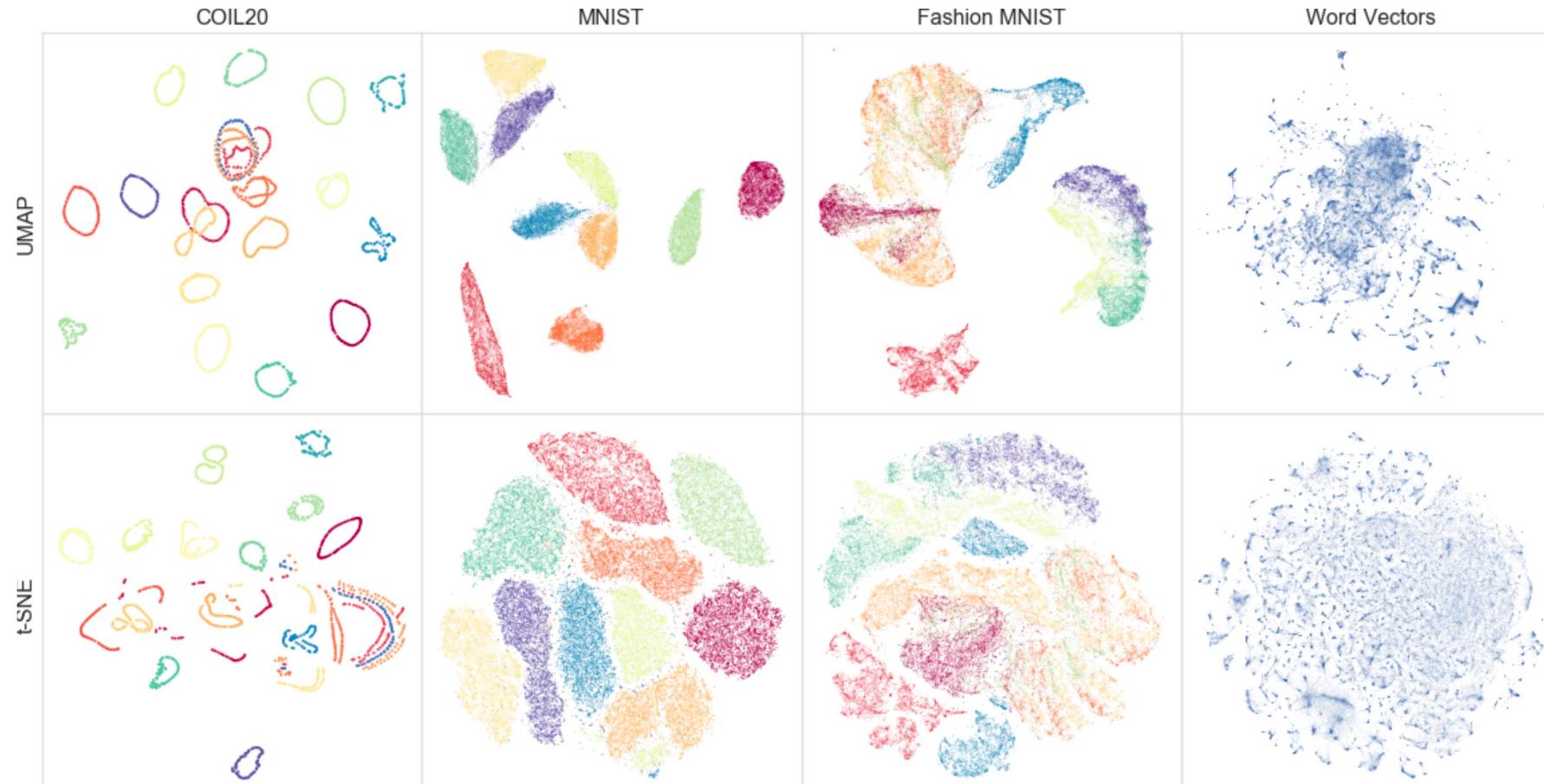


[McInnes et al., arxiv, 2018](#)



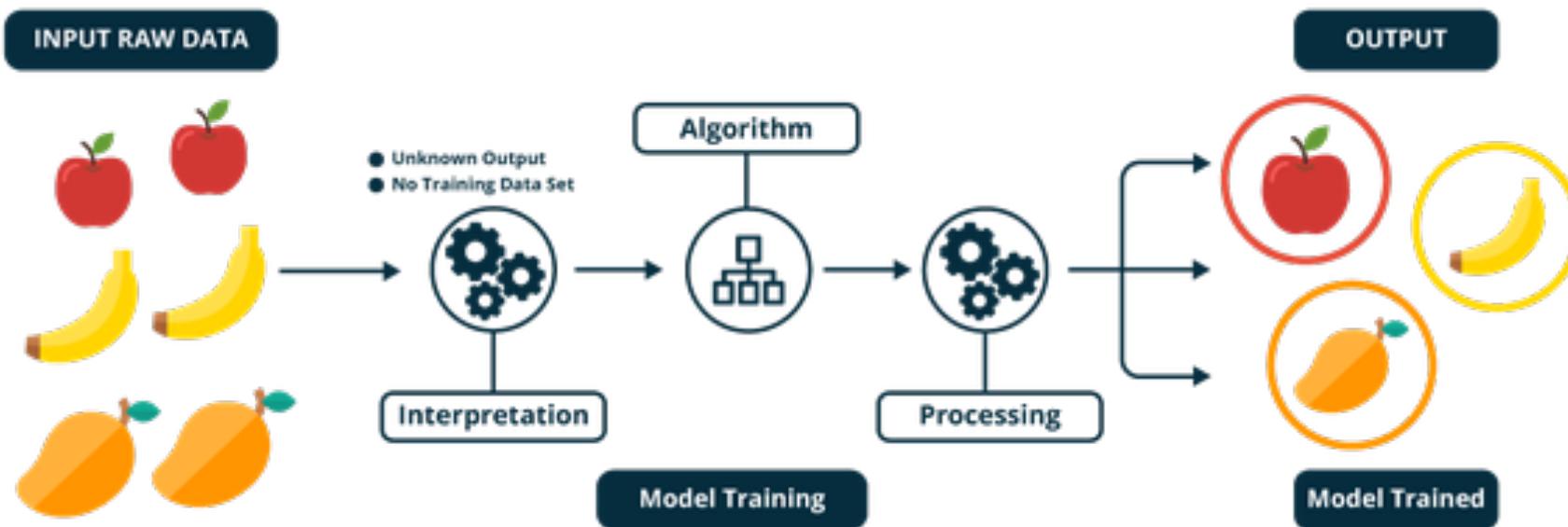
# T-SNE versus UMAP

McInnes et al., 2018



[Nikolay Oskolkov's blog](#)

# Unsupervised learning



# Unsupervised machine learning algorithms

- In contrast to supervised machine learning algorithms, they:
  - are applied when the information used to train is neither classified nor labeled.
  - can infer a function to describe a hidden structure from unlabeled data.
  - do not figure out the right output, but explore the data and can draw inferences from datasets to describe hidden structures from unlabeled data.
- The goal for unsupervised learning is to model the underlying structure or distribution in the data in order to learn more about the data.

# Clustering

- Pattern detection: discover hidden patterns in your data
- Compression: replace all cluster members by one representative
- Data augmentation: replace noisy feature vectors by the cluster consensus vector
- Dimensionality reduction: replace a feature vector by cluster similarities
- Outlier detection: detect feature vectors far away from cluster centers
- Semisupervised learning: assign cluster labels to unlabeled cluster members

# K-means Clustering

1. Choose  $K$  points from  $(\mathbf{x}_1, \mathbf{x}_2 \dots \mathbf{x}_N)$  as initial values for the cluster centers  $\boldsymbol{\mu}_k$ .  
These initial points should be spread out over the dataset.
2. Assignment step: assign each  $\mathbf{x}_i, i = 1 \dots N$ , to the cluster with the closest center:

$$c_i = \operatorname{argmin}_k \|\mathbf{x}_i - \boldsymbol{\mu}_k\|^2$$

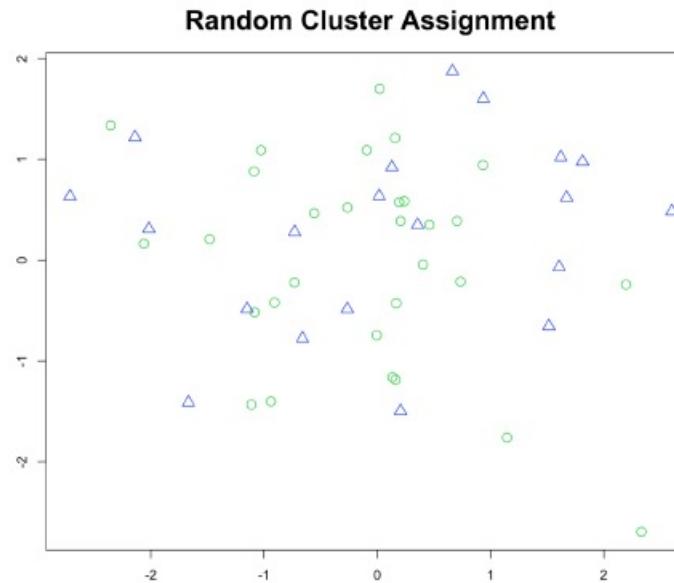
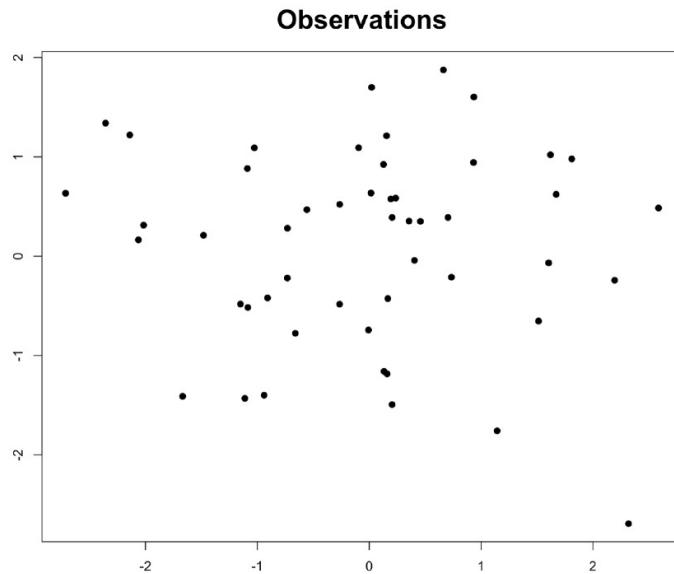
3. Update step: recalculate new  $\boldsymbol{\mu}_k$  based on cluster assignments  $C_k = \{i | c_i = k\}$  :

$$\boldsymbol{\mu}_k = \frac{1}{|C_k|} \sum_{i \in C_k} \mathbf{x}_i$$

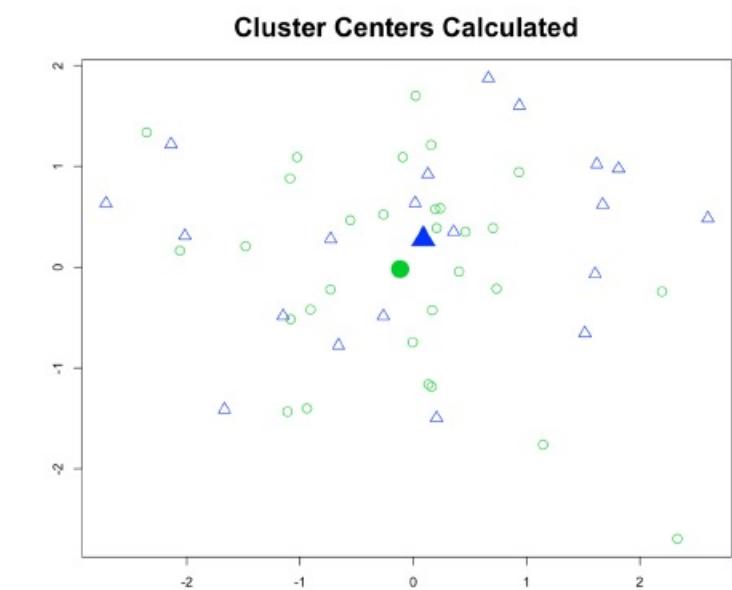
4. Iterate steps 2 & 3 until convergence

K-means at work

# K-means

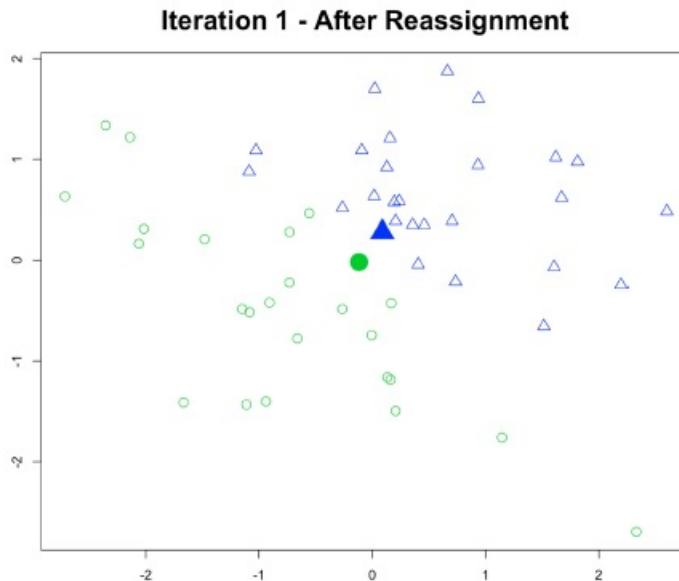


K-means initially assigns each point to one of two clusters.

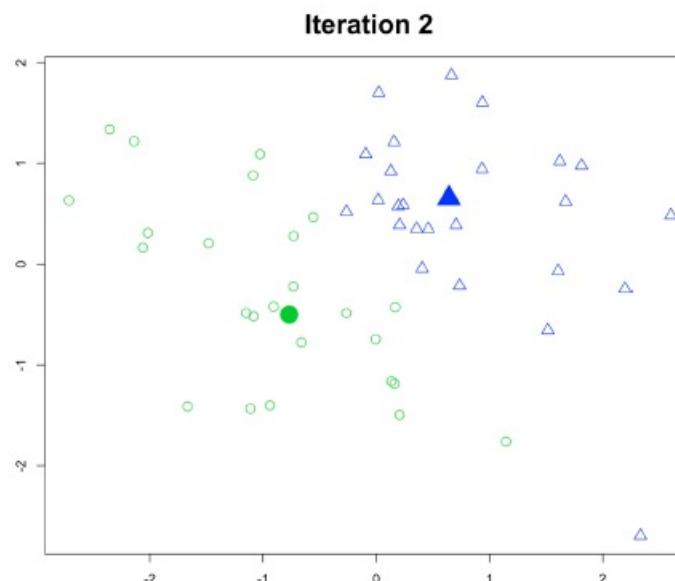


The centers of each subgroup is calculated as the average position of all the points in that group.

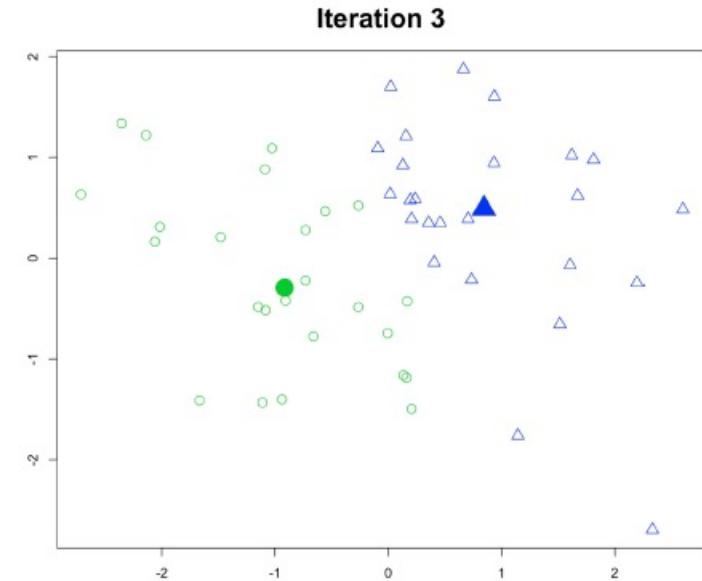
# K-means



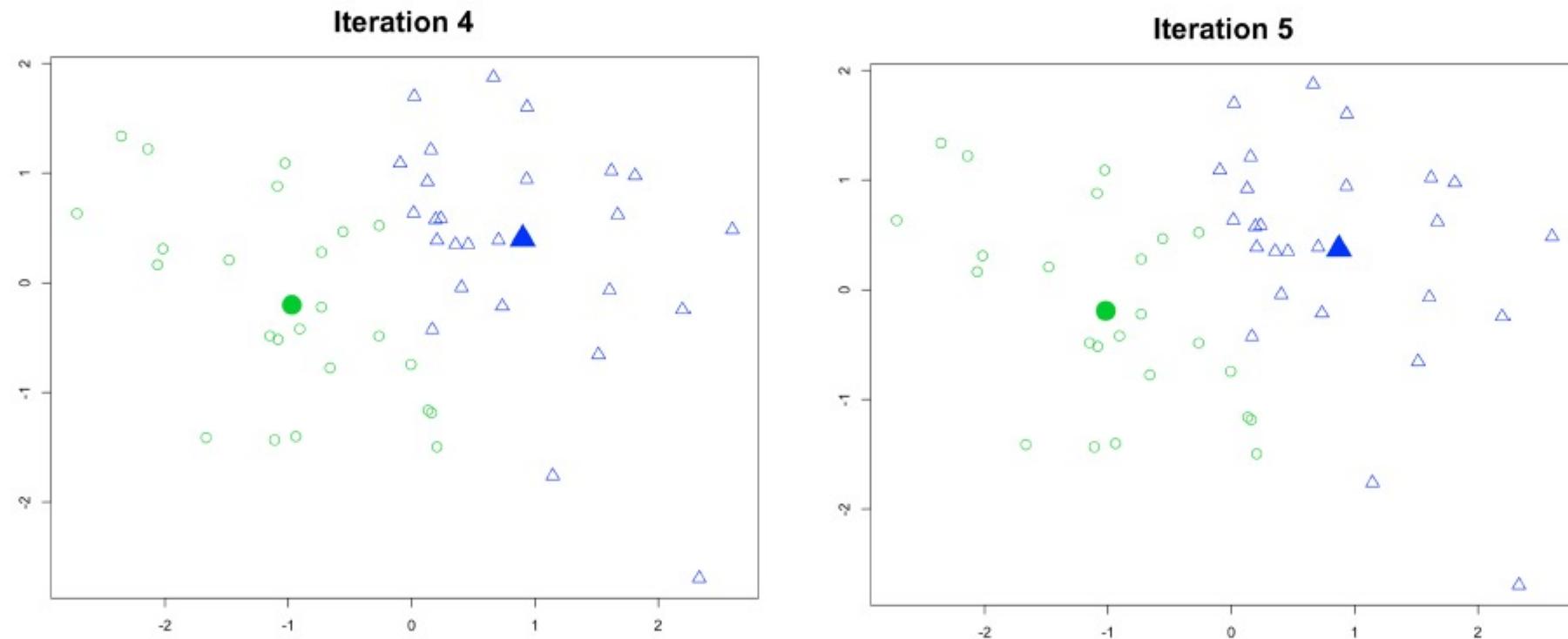
All the points closest to the solid blue triangle are assigned to the corresponding cluster while those close to the solid green circle are assigned to the other cluster.



New cluster centers are calculated and each observation reassigned to the cluster of the nearest center.



# K-means

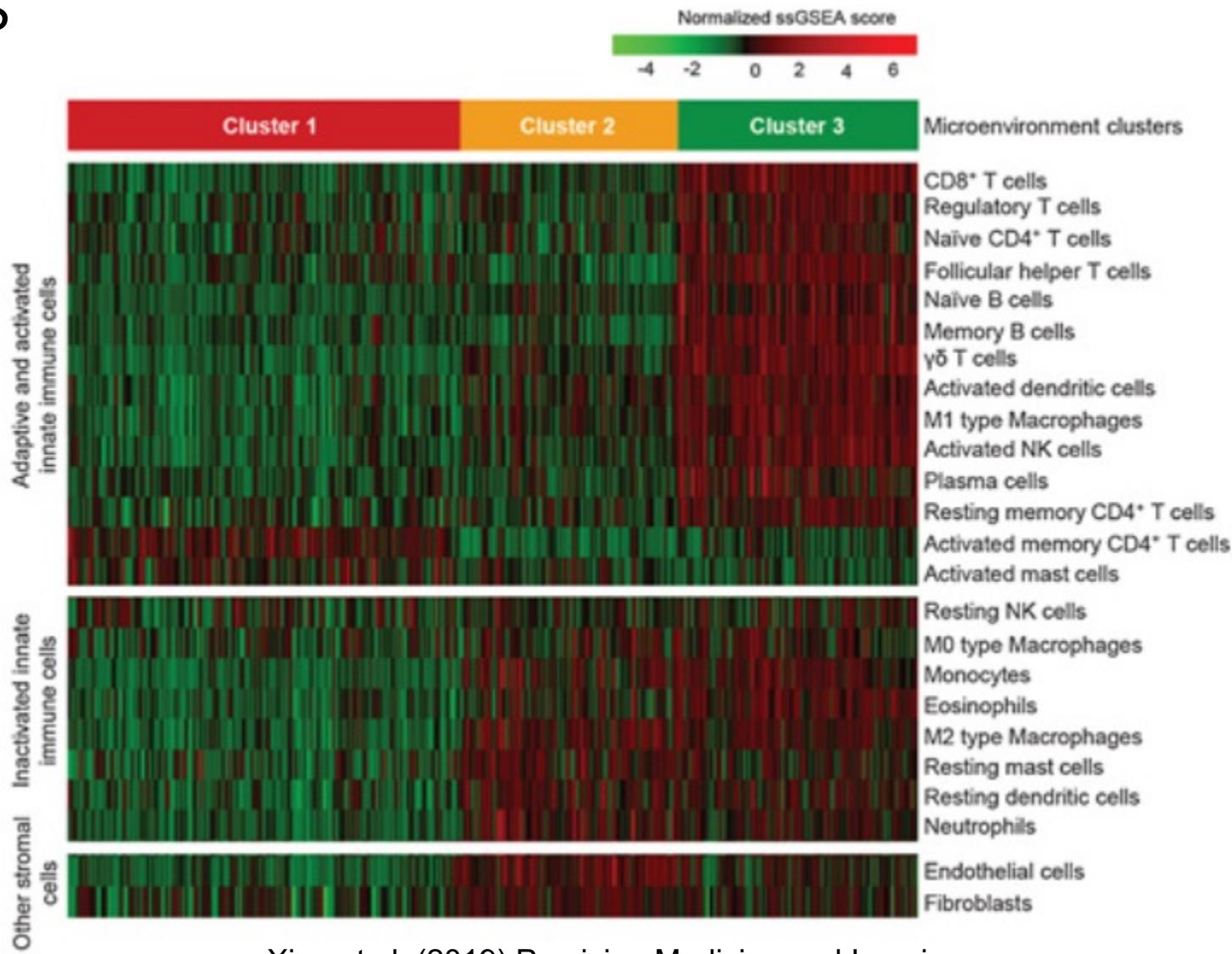


The iterative process stops after a given number of iterations or when the assignment of data points to clusters isn't changing.

# K-means

- Advantage: easy to implement and fast and efficient in terms of computational cost
- Disadvantages:
  - Choice of K
  - Initial seeds have a strong impact on final results
  - Clustering data of varying sizes and density
  - Clustering outliers

# K-means



Xiao et al. (2019) Precision Medicine and Imaging

Isabelle Dupanloup, BCF

# Gaussian Mixtures

- The data is modelled as a mixtures of  $n$  Gaussian distributions:

$$p(\mathbf{x}|\pi_k, \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k, k = 1 \dots K) = \sum_{k=1}^K \pi_k \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$$

- The parameters  $\pi_k, \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k$  are determined by the EM algorithm:

- Expectation step: calculate for each  $\mathbf{x}_i$  the probability  $z_{ik}$  to belong to cluster  $k$

$$z_{ik} = \frac{\pi_k \mathcal{N}(\mathbf{x}_i|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_{k=1}^K \pi_k \mathcal{N}(\mathbf{x}_i|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}$$

- Maximization step: recalculate new  $\pi_k, \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k$  based on  $z_{ik}$ :

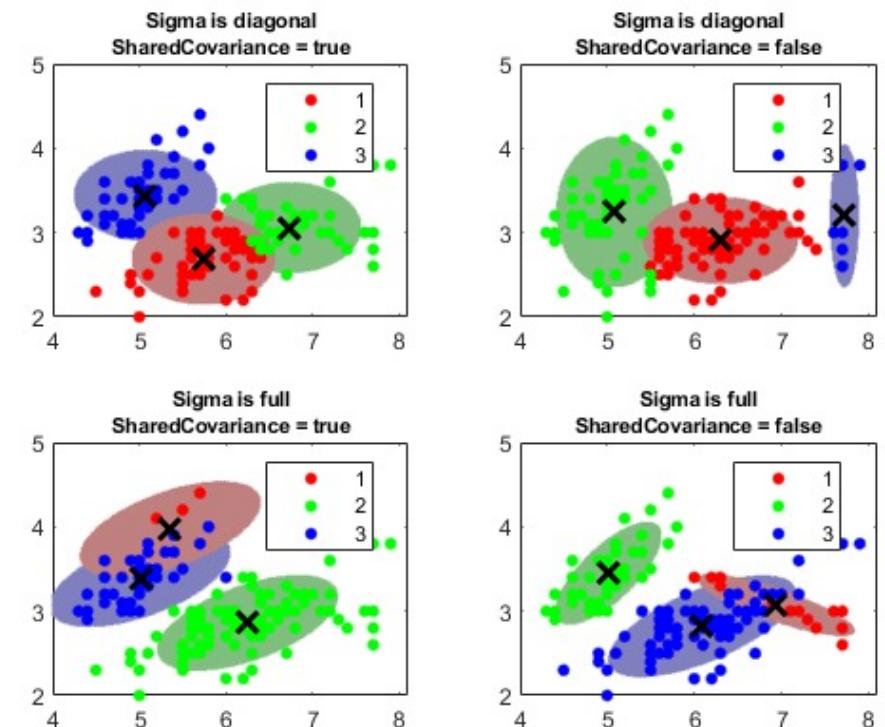
$$n_k = \sum_{i=1}^N z_{ik}, \pi_k = \frac{n_k}{N}, \boldsymbol{\mu}_k = \frac{1}{n_k} \sum_{i=1}^N z_{ik} \mathbf{x}_i, \boldsymbol{\Sigma}_k = \mathcal{M}(C_{ij}); C_{ij} = \frac{1}{n_k} \sum_{i=1}^N z_{ik} (\mathbf{x}_i - \boldsymbol{\mu}_k)^2$$

- Iterate steps 2 & 3 until convergence

- $Z_i = \underset{k}{\operatorname{argmax}} z_{ik}$ : cluster assignment for  $\mathbf{x}_i$

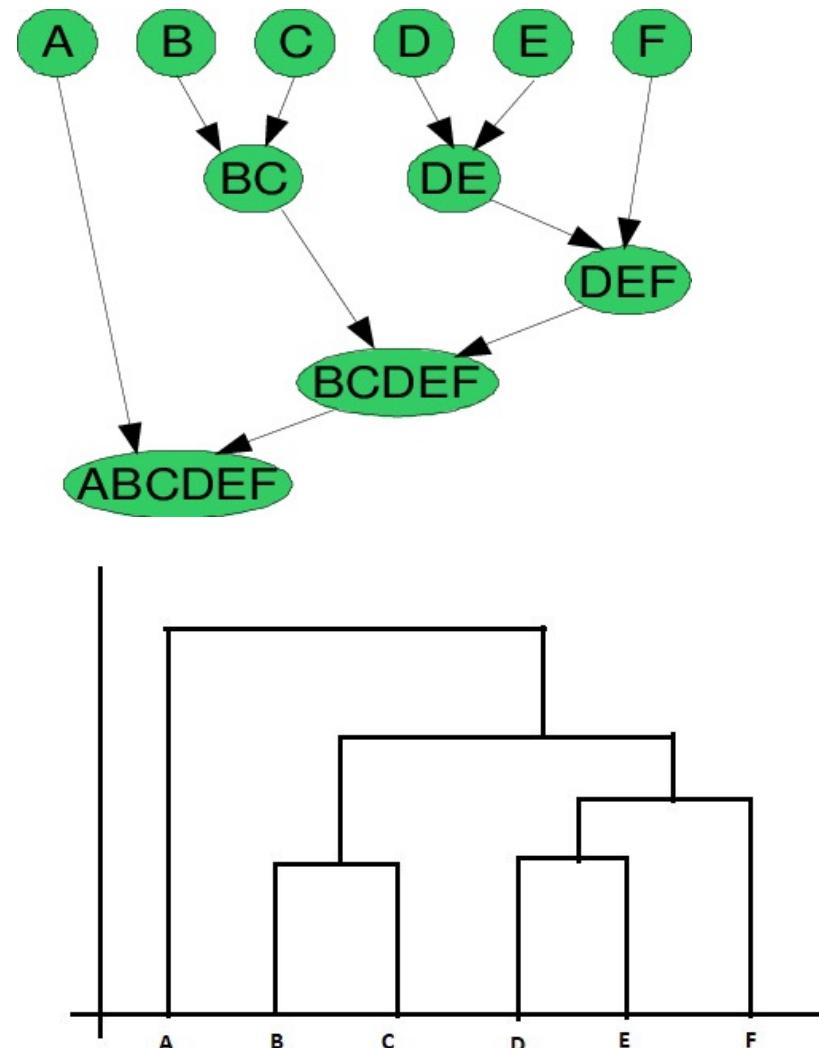
[Frailey & Raftery, The Computer Journal, 1998](#)

## Models $\mathcal{M}$

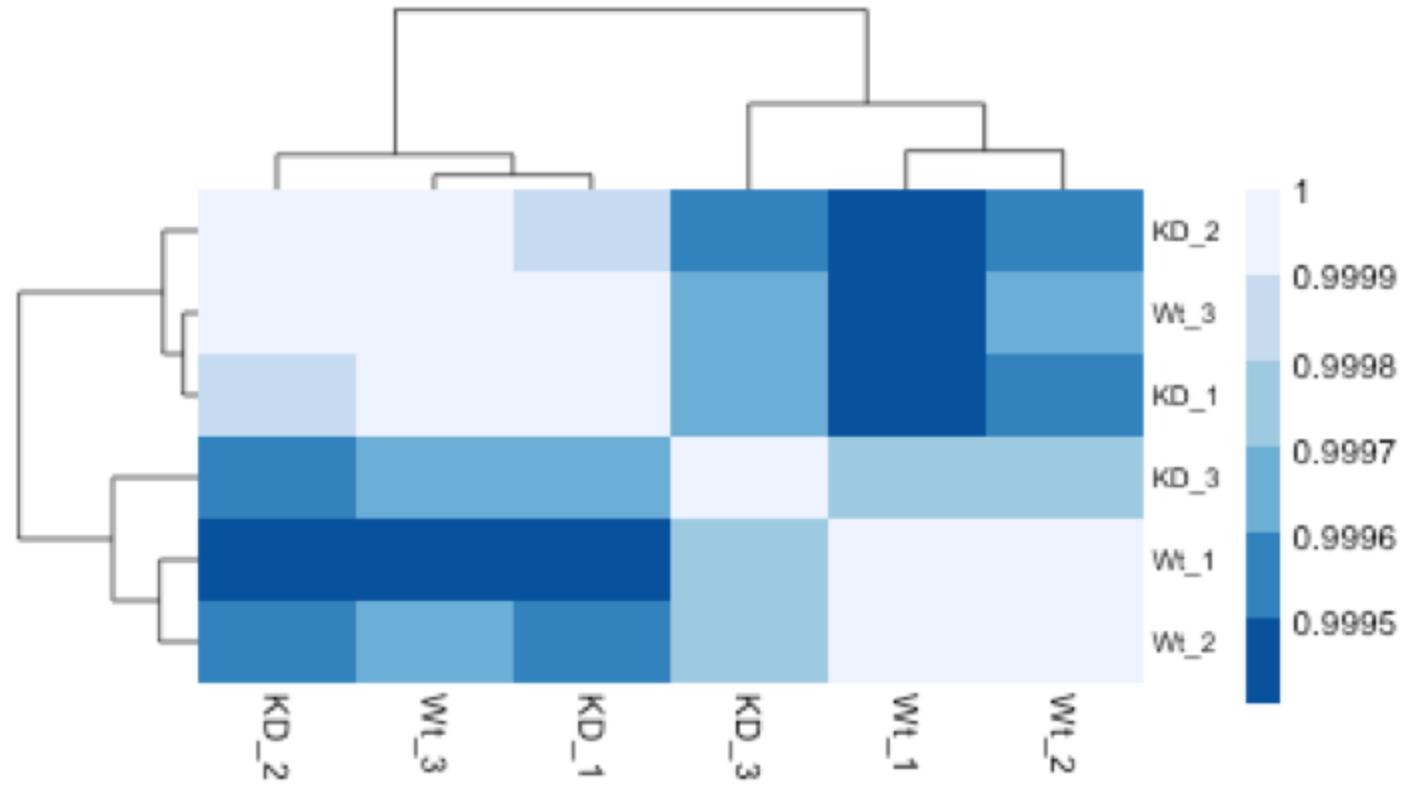


# Hierarchical clustering

- 2 types: agglomerative or divisive
- Agglomerative:
  1. Compute the proximity matrix
  2. Let each data point be a cluster
  3. Repeat: Merge the two closest clusters and update the proximity matrix
  4. Until only a single cluster remains
- Results can be visualized using a dendrogram



# Hierarchical clustering



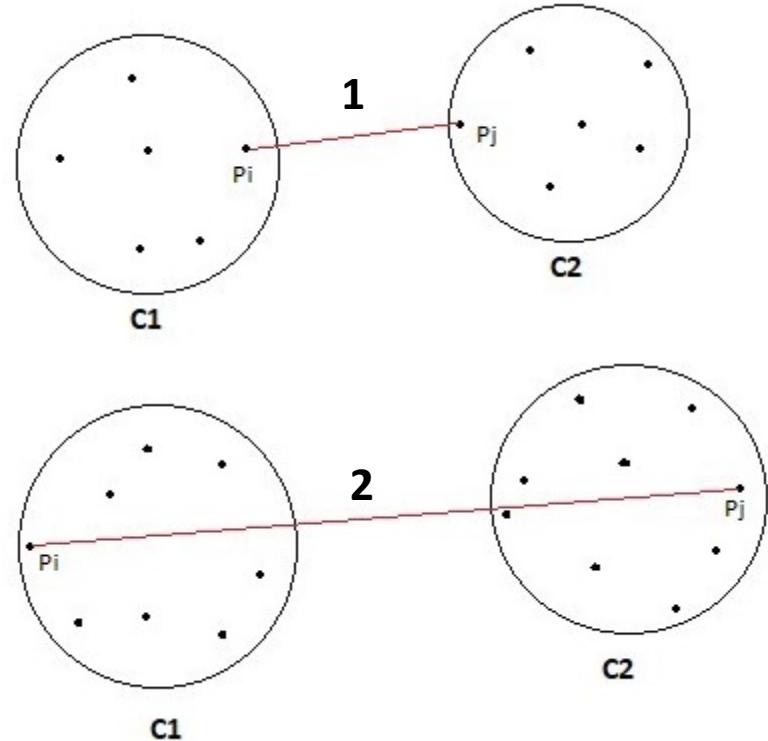
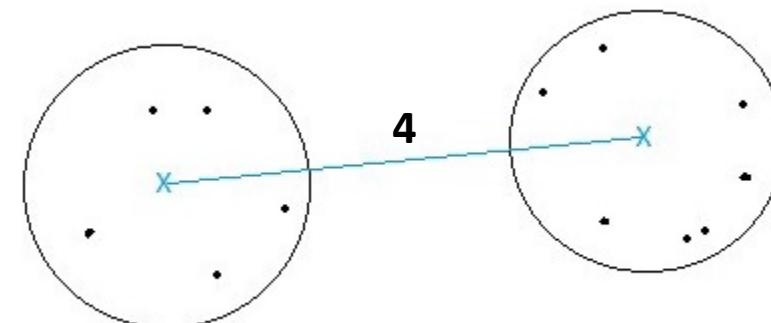
[https://hbctraining.github.io/DGE\\_workshop\\_salmon/lessons/03\\_DGE\\_QC\\_analysis.html](https://hbctraining.github.io/DGE_workshop_salmon/lessons/03_DGE_QC_analysis.html)

# Agglomerative Hierarchical Clustering

1. Initialize clusters  $C_i = \{x_i\}$ ,  $i = 1 \dots N$ ,  $id = \{1, 2, \dots, N\}$  and calculate distances  $d_{ij} = d(C_i, C_j)$
2. Merge closest clusters:  $C_i = C_i \cup C_j$ , if  $d_{ij} = \min_{(l,k)} d_{lk}$ , drop  $j$ :  $id = id \setminus j$  and recalculate  $d_{ik} = d(C_i, C_k)$ ,  $k \in id \setminus i$
3. Repeat 2 until  $|id| = K$  or  $d_{ik} \geq \varepsilon, \forall i, k \in id$

# Hierarchical clustering

- **Divisive:** start with all the data points as a single cluster and in each iteration, we separate the data points from the cluster which are not similar
- **Similarity between clusters (choice of linkage)**
  1. Min
  2. Max
  3. Group Average
  4. Distance Between Centroids
  5. Ward's Method (similar to Group Average)



c1

Isabelle Dupanloup, BCF

# Agglomerative Hierarchical Clustering

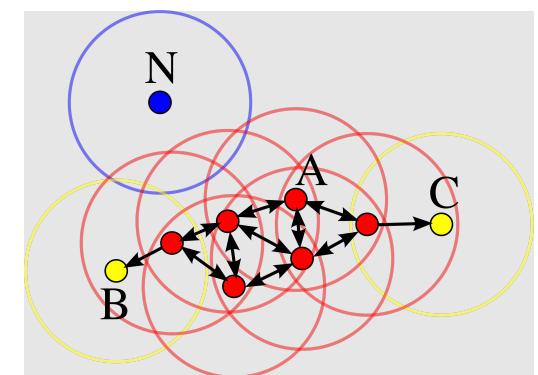
Names	Formula
Maximum or <a href="#">complete-linkage clustering</a>	$\max \{ d(a, b) : a \in A, b \in B \}.$
Minimum or <a href="#">single-linkage clustering</a>	$\min \{ d(a, b) : a \in A, b \in B \}.$
Unweighted average linkage clustering (or <a href="#">UPGMA</a> )	$\frac{1}{ A  \cdot  B } \sum_{a \in A} \sum_{b \in B} d(a, b).$
Weighted average linkage clustering (or <a href="#">WPGMA</a> )	$d(i \cup j, k) = \frac{d(i, k) + d(j, k)}{2}.$
Centroid linkage clustering, or UPGMC	$\ c_s - c_t\ $ where $c_s$ and $c_t$ are the centroids of clusters $s$ and $t$ , respectively.
Minimum energy clustering	$\frac{2}{nm} \sum_{i,j=1}^{n,m} \ a_i - b_j\ _2 - \frac{1}{n^2} \sum_{i,j=1}^n \ a_i - a_j\ _2 - \frac{1}{m^2} \sum_{i,j=1}^m \ b_i - b_j\ _2$

where  $d$  is the chosen metric. Other linkage criteria include:

- The sum of all intra-cluster variance.
- The increase in variance for the cluster being merged ([Ward's criterion](#)).<sup>[8]</sup>
- The probability that candidate clusters spawn from the same distribution function (V-linkage).
- The product of in-degree and out-degree on a k-nearest-neighbour graph (graph degree linkage).<sup>[9]</sup>
- The increment of some cluster descriptor (i.e., a quantity defined for measuring the quality of a cluster) after merging two clusters.<sup>[1]</sup>

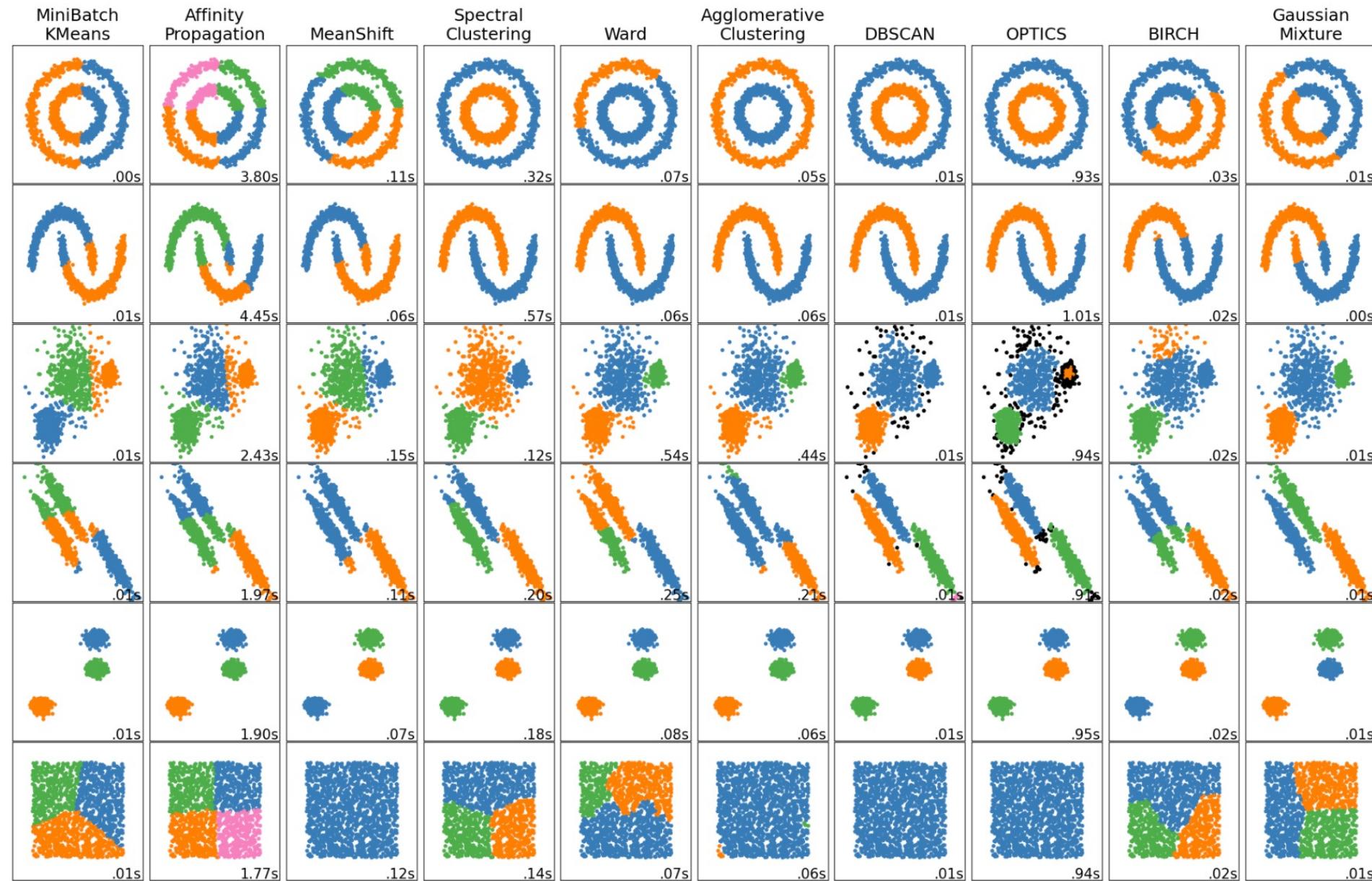
# DBSCAN

- DBSCAN uses a minimal distance  $\varepsilon$  and a minimal number of cluster seed points  $n$  as input
- It finds dense regions in the data and uses them as cluster seeds. Then it expands these clusters and adds points if they are within  $\varepsilon$  distance to a cluster. All points not within reach of a cluster are marked as outliers.
- DBSCAN can deal with clusters of arbitrary shapes.
- DBSCAN requires clusters of similar density
- Distance based clustering are difficult in high dimensions (curse of dimensionality)



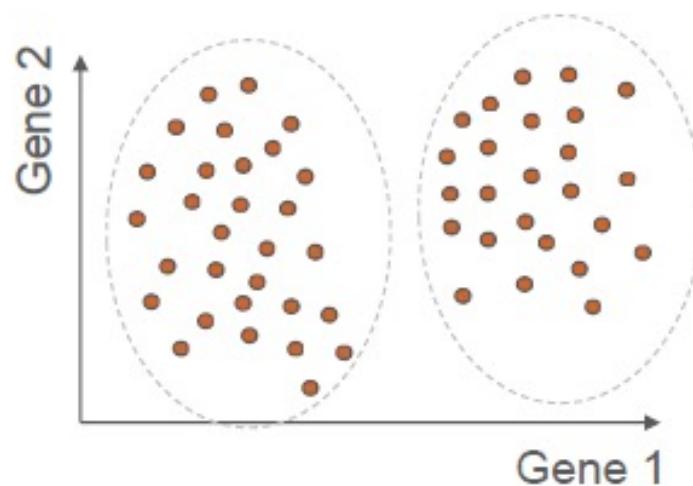
# DBSCAN

[sklearn docu](#)



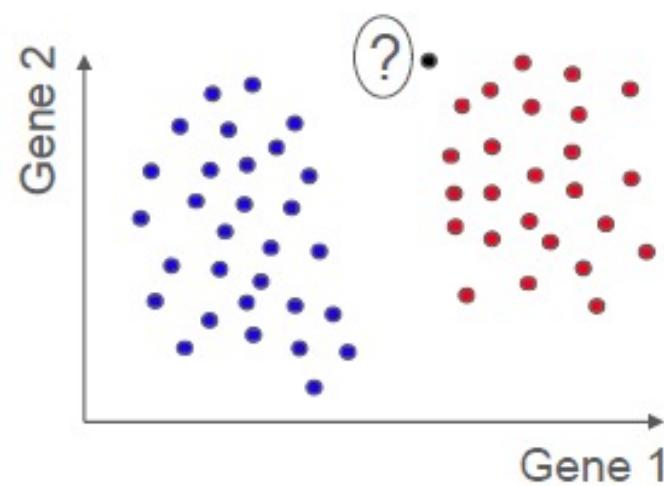
# Supervised learning

## Class discovery



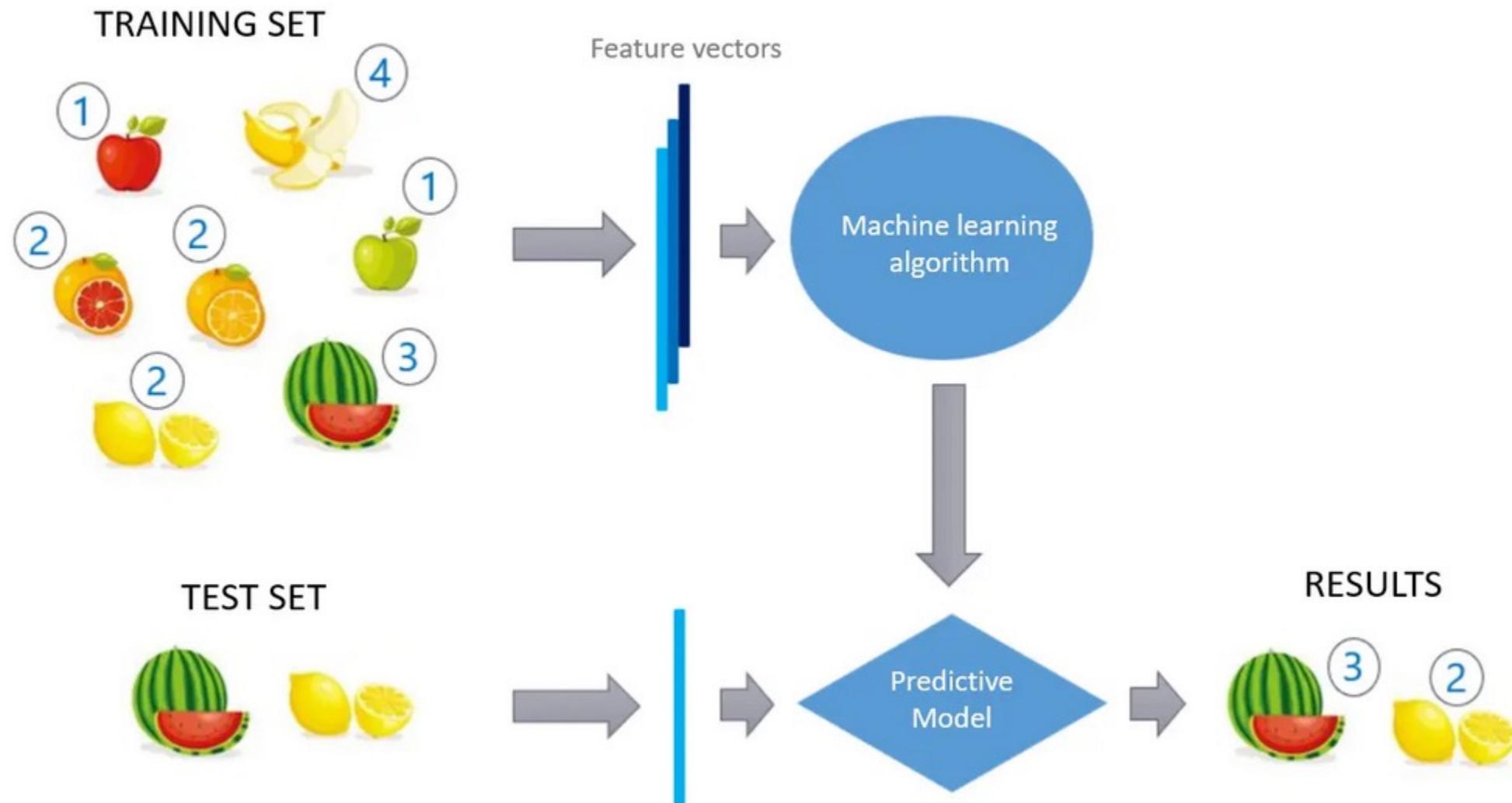
Find natural groups in the data (e.g. sets of patients with similar gene expression)

## Class prediction



Given previous measurements for which the grouping is known (**red** and **blue**), can we predict the group to which a new observation belong ?

# Supervised learning



# Supervised learning

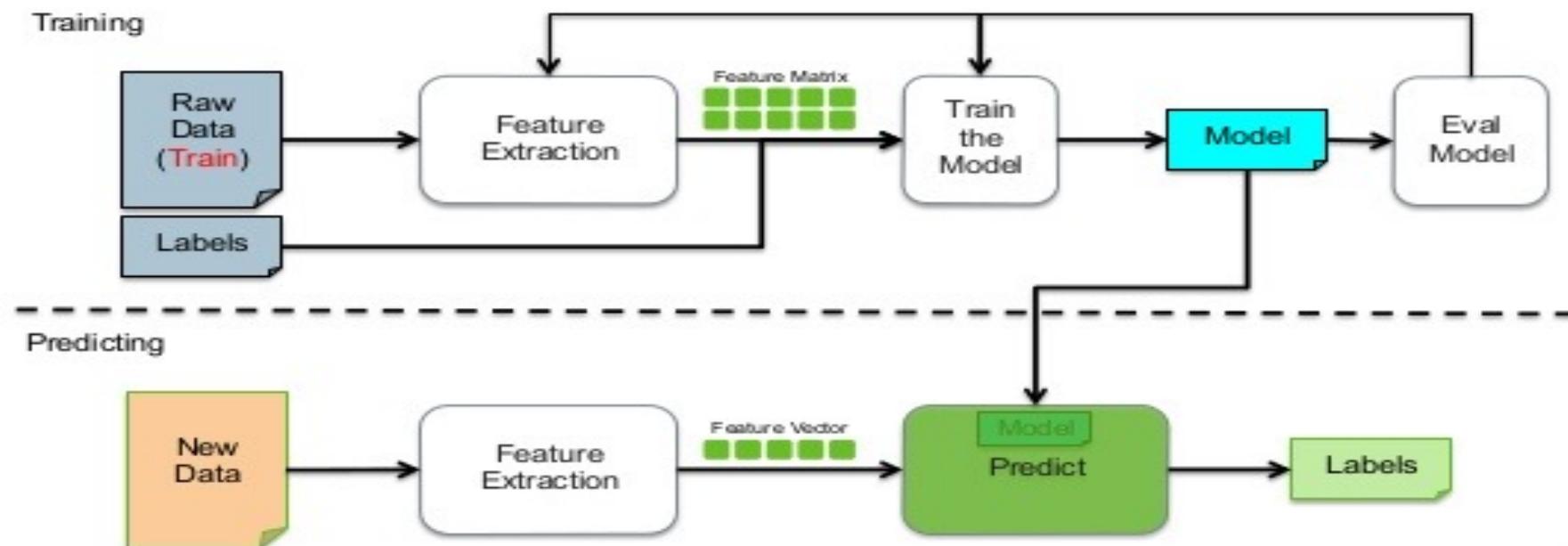
- Supervised learning is where you have input variables ( $x$ ) and an output variable ( $Y$ ) and you use an algorithm to learn the mapping function from the input to the output.

$$Y = f(X)$$

- The goal is to approximate the mapping function so well that when you have new input data ( $x$ ) that you can predict the output variables ( $Y$ ) for that data.
- It is called supervised learning because the process of an algorithm learning from the training dataset can be thought of as a teacher supervising the learning process. We know the correct answers, the algorithm iteratively makes predictions on the training data and is corrected by the teacher. Learning stops when the algorithm achieves an acceptable level of performance

# Supervised learning workflow

## Supervised Learning Workflow



# Supervised learning

- Supervised learning problems can be further grouped into regression and classification problems.
  - Classification: A classification problem is when the output variable is a category, such as “red” or “blue” or “disease” and “no disease”
  - Regression: A regression problem is when the output variable is a real value, such as “dollars” or “weight”
- Some popular examples of supervised machine learning algorithms are:
  - Linear regression for regression problems
  - Random forest for classification and regression problems
  - Support vector machines for classification problems

# Supervised learning

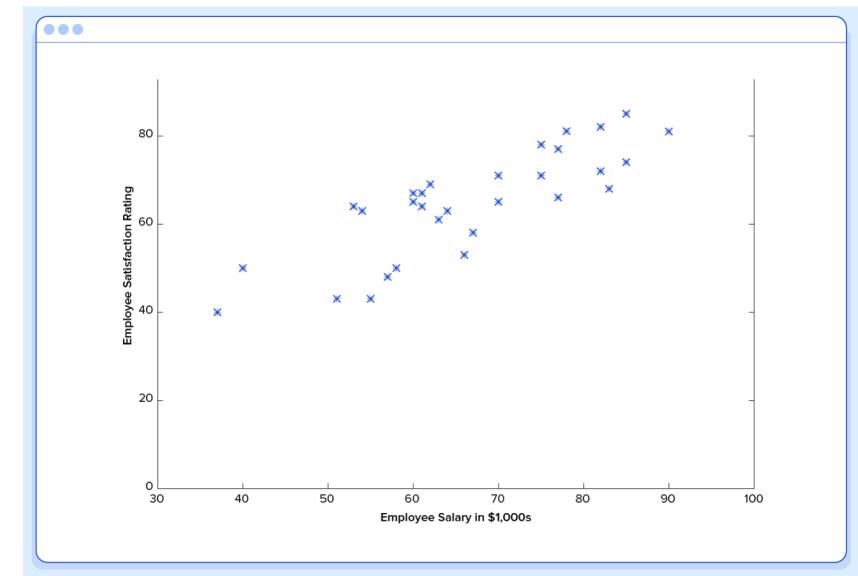
- Let's assume our simple predictor has this form:

$$h(x) = \theta_0 + \theta_1 x$$

- where  $\theta_0$  and  $\theta_1$  are constants. Our goal is to find the perfect values of  $\theta_0$  and  $\theta_1$  to make our predictor work as well as possible.
- Optimizing the predictor  $h(x)$  is done using training examples.
  - For each training example, we have an input value  $x_{\text{train}}$ , for which a corresponding output,  $y$ , is known in advance.
  - For each example, we find the difference between the known, correct value  $y$ , and our predicted value  $h(x_{\text{train}})$ .
  - With enough training examples, these differences give us a useful way to measure the “wrongness” of  $h(x)$ .
  - We can then tweak  $h(x)$  by tweaking the values of  $\theta_0$  and  $\theta_1$  to make it “less wrong”.
  - This process is repeated over and over until the system has converged on the best values for  $\theta_0$  and  $\theta_1$ .
  - In this way, the predictor becomes trained, and is ready to do some real-world predicting.

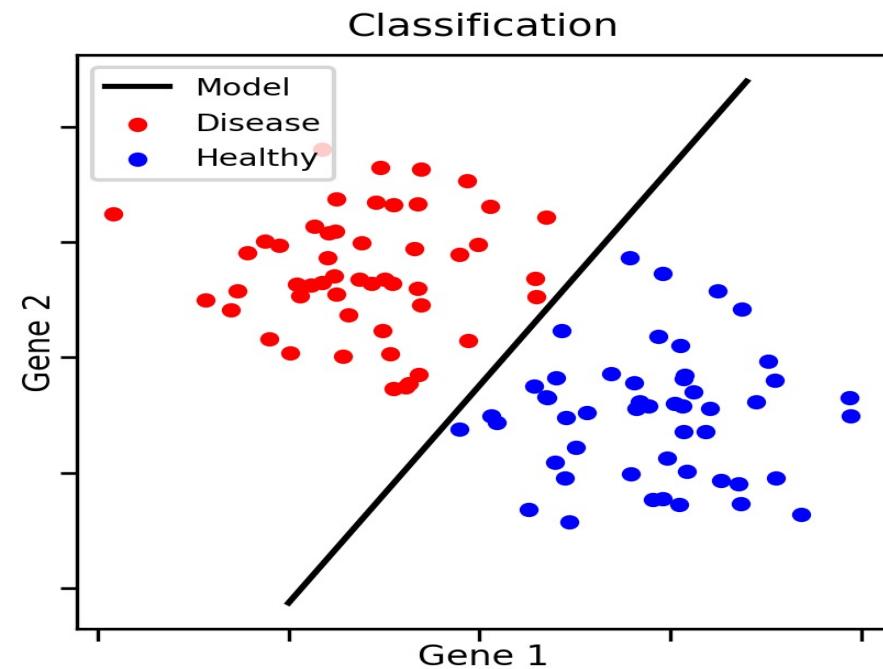
# Supervised learning

- Consider the following training data, wherein company employees have rated their satisfaction on a scale of 1 to 100:
- First, notice that the data is a little noisy.
  - That is, while we can see that there is a pattern to it (i.e. employee satisfaction tends to go up as salary goes up), it does not all fit neatly on a straight line.
  - This will always be the case with real-world data (and we absolutely want to train our machine using real-world data!).
  - So then how can we train a machine to perfectly predict an employee's level of satisfaction?
    - The answer, of course, is that **we can't**.
  - The goal of ML is never to make “perfect” guesses, because ML deals in domains where there is no such thing.
    - **The goal is to make guesses that are good enough to be useful.**

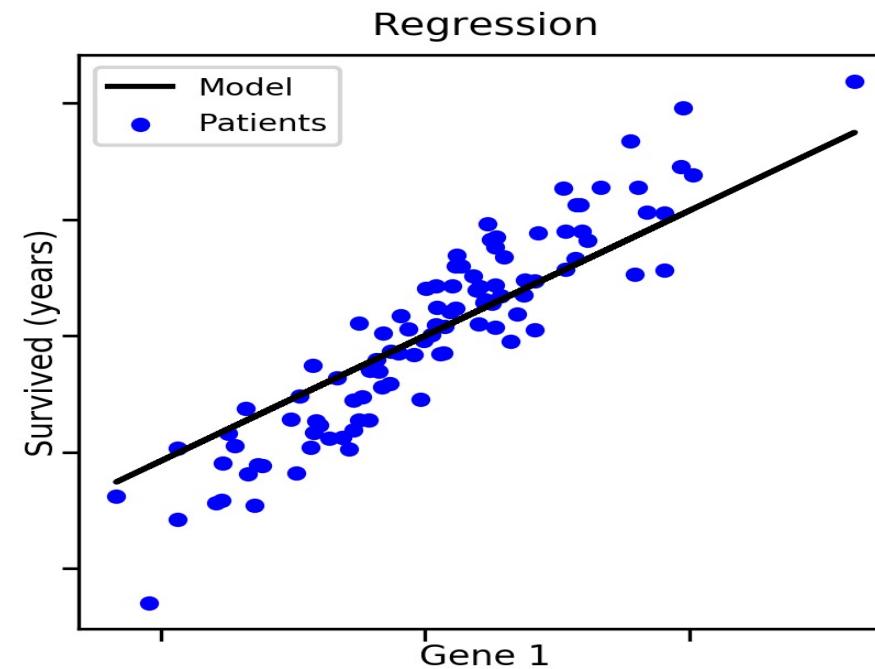


# Supervised learning

Fundamentally, classification is about predicting a label and regression is about predicting a quantity.



Classification predictive modeling is the task of approximating a mapping function ( $f$ ) from input variables ( $X$ ) to discrete output variables ( $y$ ).



Regression predictive modeling is the task of approximating a mapping function ( $f$ ) from input variables ( $X$ ) to a continuous output variable ( $y$ ).

# Supervised learning

- Supervised learning problems can be further grouped into regression and classification problems.
  - Classification: A classification problem is when the output variable is a category, such as “red” or “blue” or “disease” and “no disease”
  - Regression: A regression problem is when the output variable is a real value, such as “dollars” or “weight”
- Some popular examples of supervised machine learning algorithms are:
  - Linear regression for regression problems
  - Random forest for classification and regression problems
  - Support vector machines for classification problems

# Supervised learning datasets

- **Training dataset**

The subset of the dataset provided to the algorithm for learning is called the training set.

- **Validation dataset**

A set of examples used to tune the parameters of a classifier/regressor.

Hybrid dataset: it is training data used for testing, but neither as part of the low-level training nor as part of the final testing

- **Test dataset**

A set of examples used only to assess the performance of a fully-specified classifier/regressor

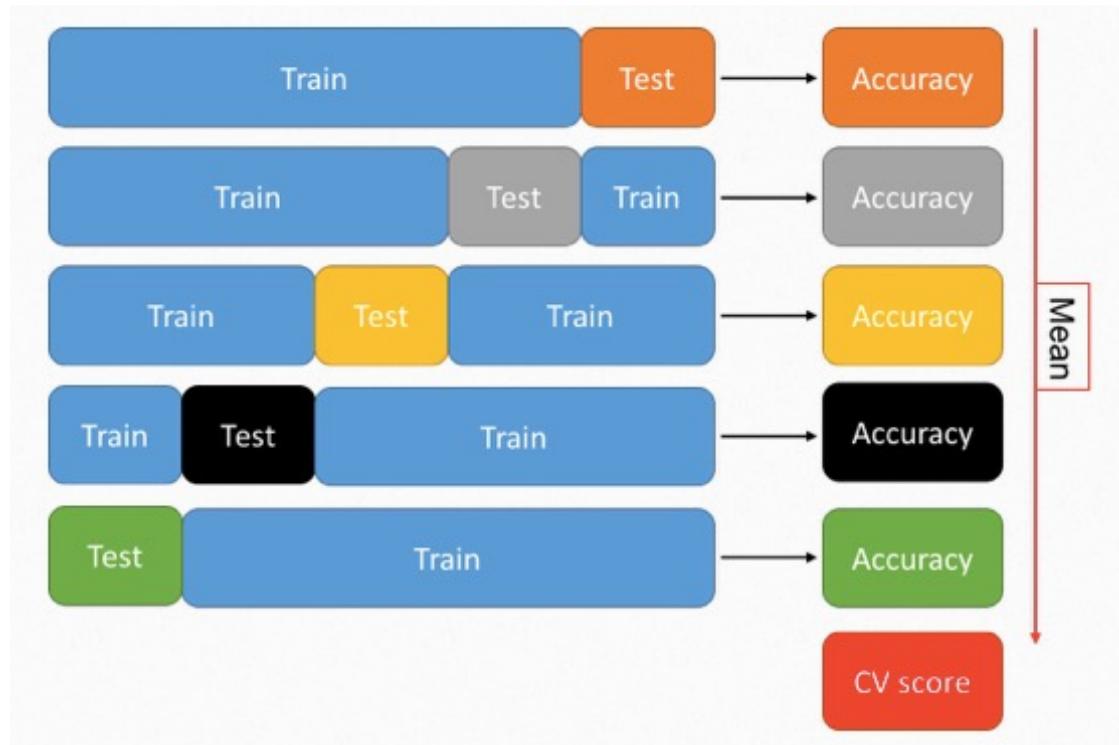
# Supervised learning datasets

- **Train-test-validation split ratio**



# Supervised learning datasets

- **K-fold cross validation**

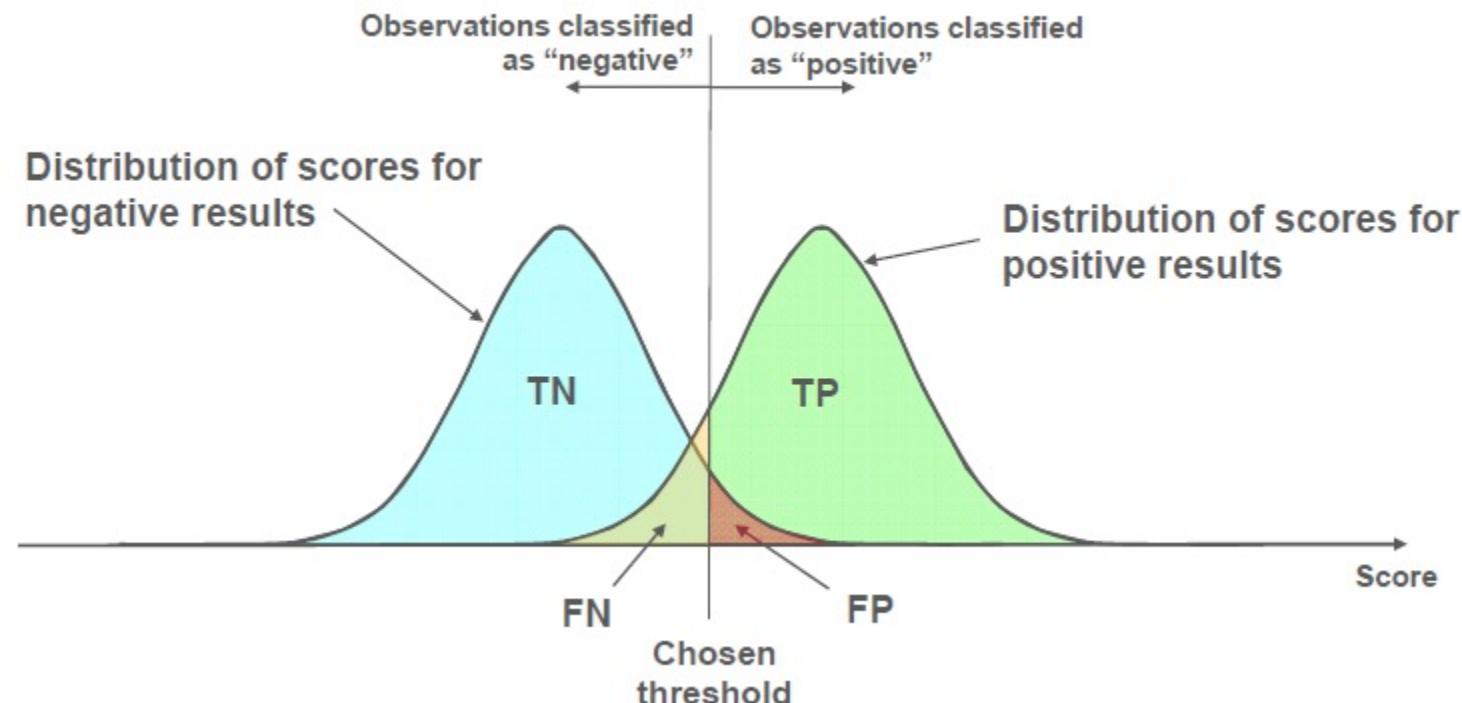


# Assessing performance

- **TP (True Positives)**: number of positive predictions which are correct
- **FP (False Positives)**: number of positive predictions which are incorrect
- **TN (True Negatives)**: number of negative predictions which are correct
- **FN (False Negatives)**: number of negative predictions which are incorrect

## confusion matrix

LABEL / PREDICTED	P	N
P	TP	FN
N	FP	TN

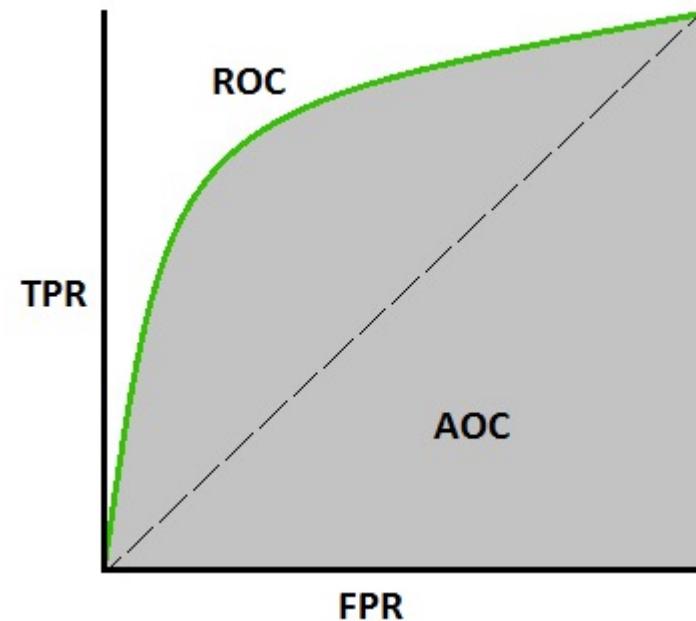


# Assessing performance

- **Precision:** fraction of the positive predictions that are actually positive:  $\frac{TP}{TP + FP}$  ← Total of positive predictions
- **Specificity:** fraction of the negative predictions that are correctly identified:  $\frac{TN}{TN + FP}$  ← Total of negative labels
- **Recall / TPR:** fraction of the positive predictions that are correctly identified:  $\frac{TP}{TP + FN}$  ← Total of positive labels  
**Sensitivity**
- **FPR:**  $1 - \text{Precision} = \frac{FP}{TN + FP}$  ← Total of negative labels
- **F-measure:** harmonic mean of precision and recall:  $2 * \frac{\text{precision} * \text{recall}}{\text{precision} + \text{recall}}$
- **Accuracy:** fraction of the total predictions that are correctly identified:  $\frac{TP + TN}{TN + TN + FP + FN}$

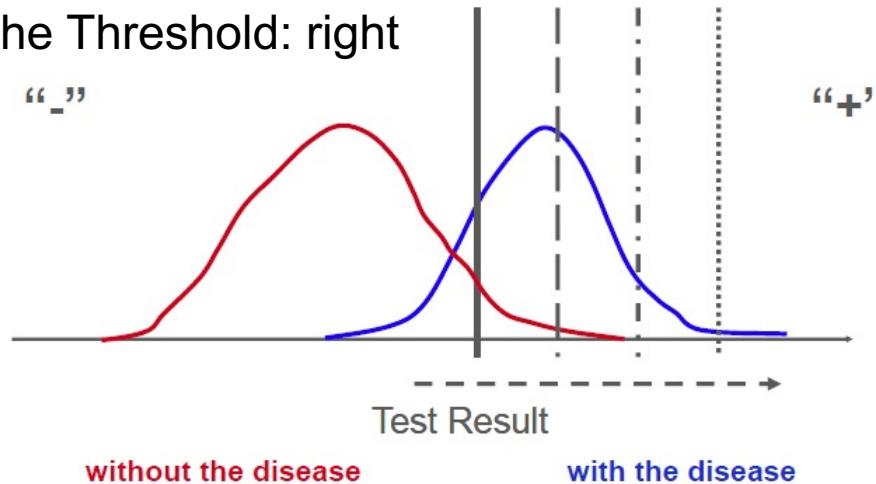
# Assessing performance: AUC - ROC curve

- **ROC = Receiver Operating Characteristic**
- **AUC = Area Under the Curve**
- AUC - ROC curve is a performance measurement for the classification problems at various threshold settings
- ROC is a probability curve and AUC represents the degree or measure of separability. It tells how much the model is capable of distinguishing between classes. Higher the AUC, the better the model is at predicting 0s as 0s and 1s as 1s. By analogy, the higher the AUC, the better the model is at distinguishing between patients with the disease and no disease.

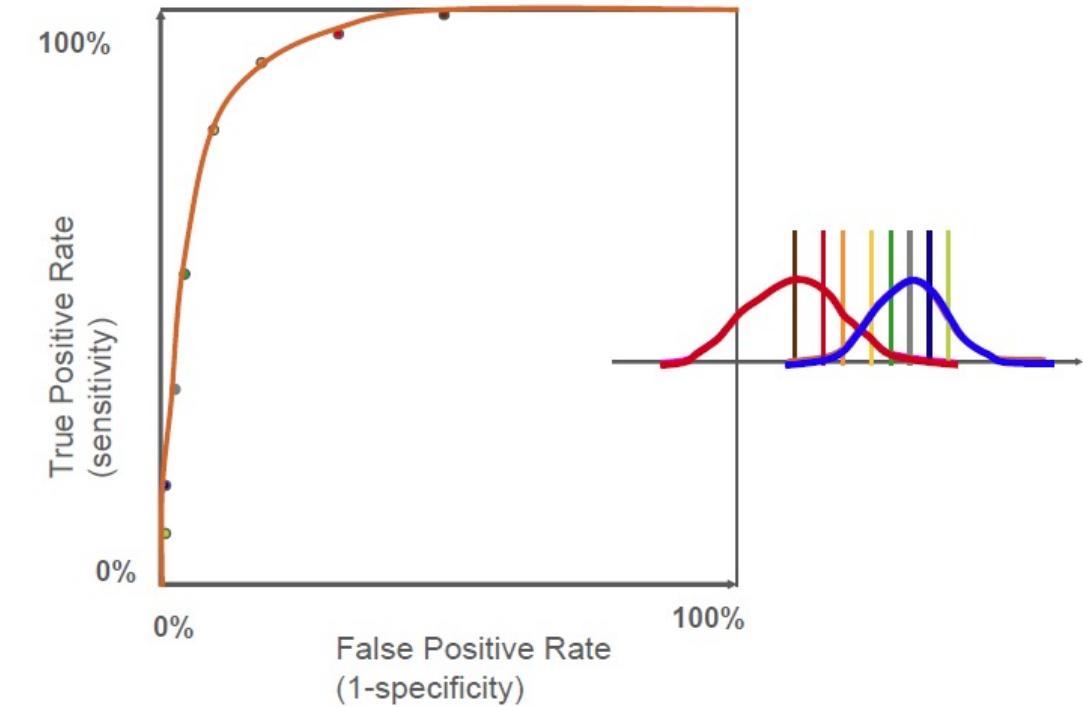
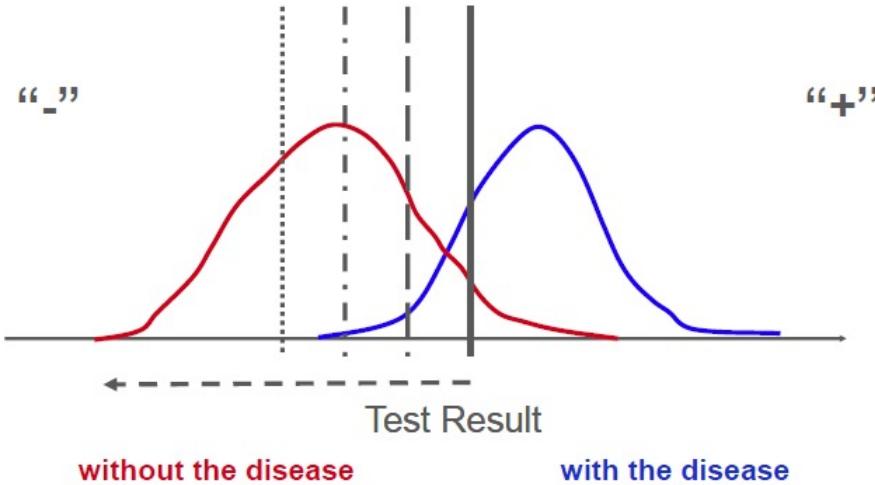


# Assessing performance: AUC - ROC curve

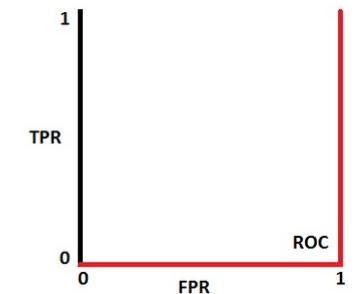
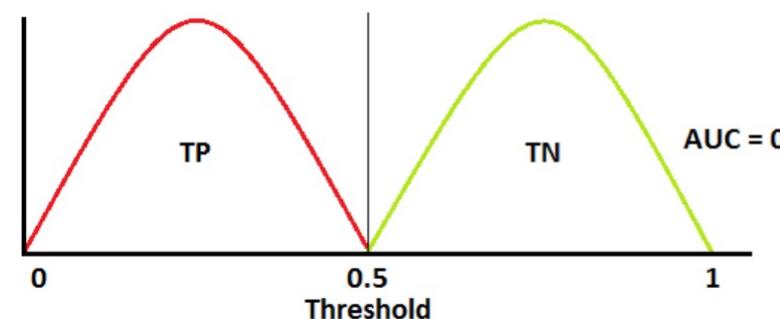
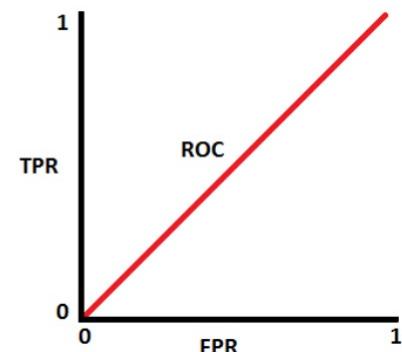
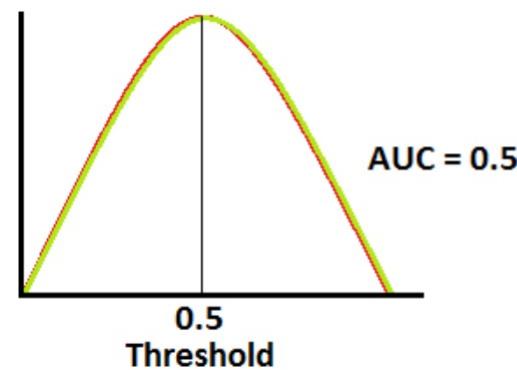
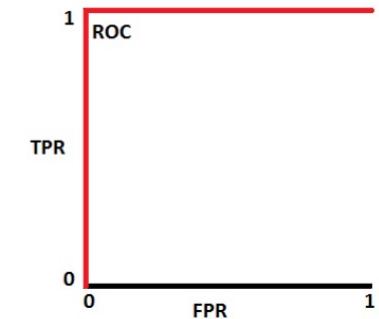
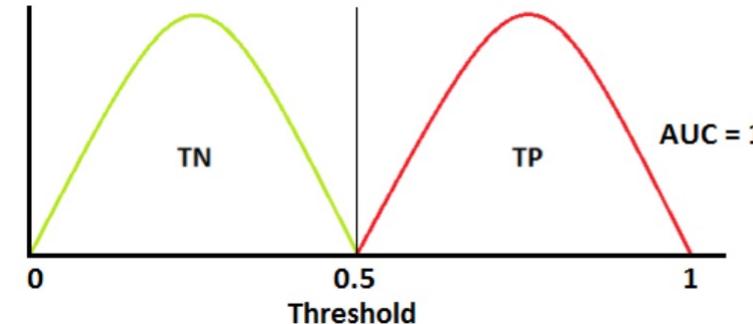
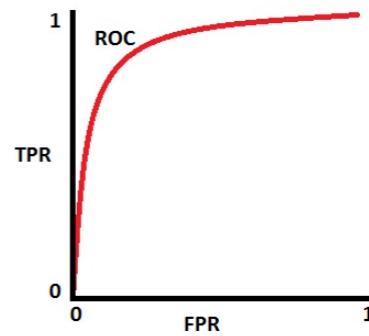
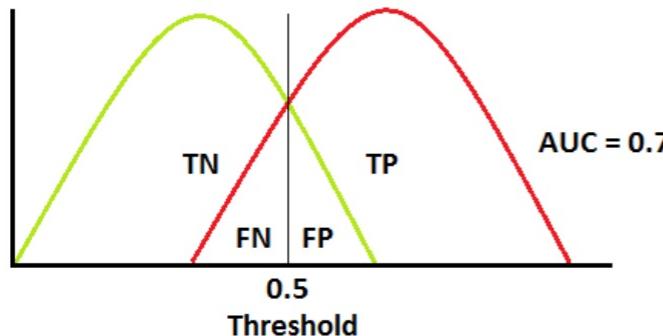
Moving the Threshold: right



Moving the Threshold: left

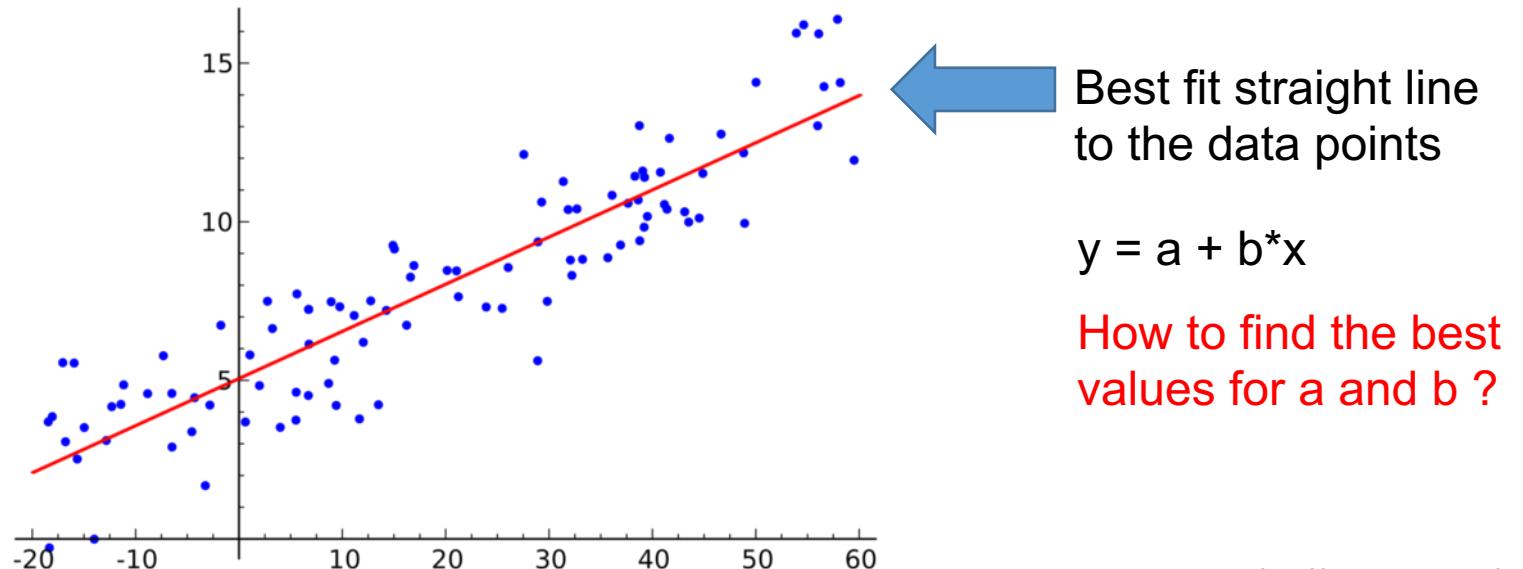


# Assessing performance: AUC - ROC curve



# Linear regression

- Regression: method of modelling a target value based on independent predictors
- Regression techniques mostly differ based on the number of independent variables and the type of relationship between the independent and dependent variables.
- Simple linear regression: type of regression analysis where the number of independent variables is one and there is a linear relationship between the independent(x) and dependent(y) variable.



# Linear regression

- Cost Function

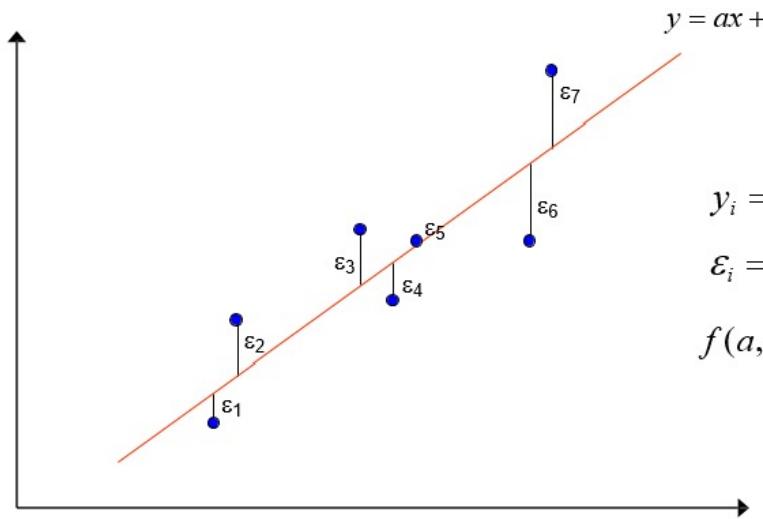
$$\underset{\text{Mean Squared Error(MSE) function}}{\minimize \frac{1}{n} \sum_{i=1}^n (pred_i - y_i)^2}$$

$$J = \frac{1}{n} \sum_{i=1}^n (pred_i - y_i)^2$$

- Model residuals (or errors): distances between the labels and the predictions using the model
- Model residuals represent the part of variability in the data the model was unable to capture

# Linear regression

Least-square fitting



Regression line  
such that:

$$\sum_i \varepsilon_i^2 = \varepsilon_1^2 + \varepsilon_2^2 + \varepsilon_3^2 + \dots$$

minimum

$$y_i = ax_i + b + \varepsilon_i$$

$$\varepsilon_i = y_i - (ax_i + b)$$

$$f(a, b) = \sum_i \varepsilon_i^2 = \sum_i [y_i - (ax_i + b)]^2$$

$$\frac{\partial f(a, b)}{\partial a} = 0$$

$$\frac{\partial f(a, b)}{\partial b} = 0$$

The least-squares procedure finds the straight line with the  
**smallest sum of squares of vertical errors.**

Minimizing  $\sum_i \varepsilon_i^2$  yields  $b_0$  and  $b_1$  estimators of  $\beta_0$  and  $\beta_1$

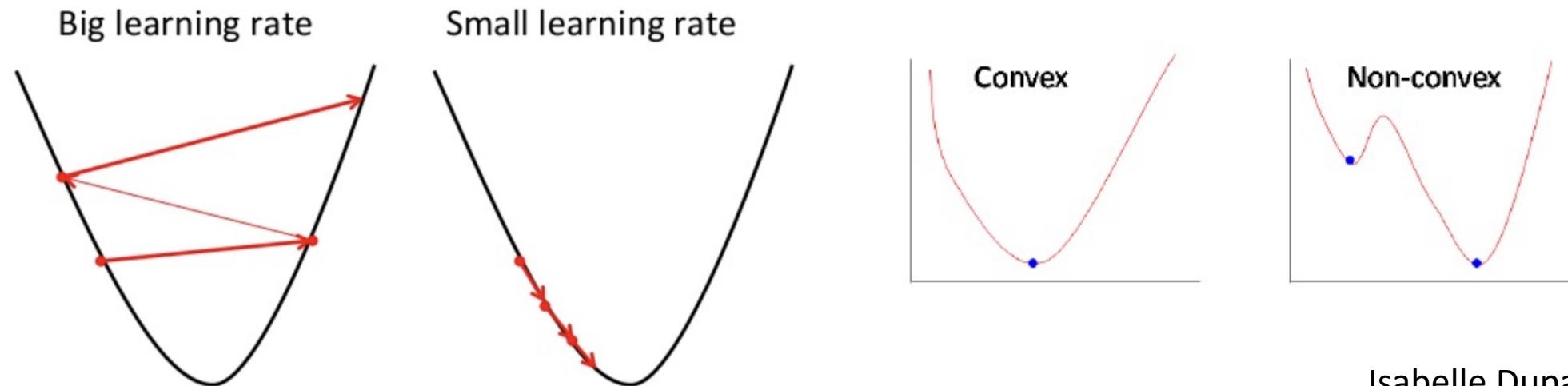
$$b_1 = \frac{\sum (X_i - \bar{X})(Y_i - \bar{Y})}{\sum (X_i - \bar{X})^2}$$

$$b_0 = \bar{Y} - b_1 \bar{X}$$

# Linear regression

- **Gradient Descent**

- Allows to optimize the values of the model coefficients by iteratively minimizing the error of the model on your training data.
- Starts with random values for each coefficient. An iterative process then goes in the direction towards minimizing the error. The process is repeated until a minimum sum squared error is achieved or no further improvement is possible.
- Uses a learning rate (alpha) parameter that determines the size of the improvement step to take on each iteration of the procedure.



# Linear regression

- **Regularization**
  - regularization methods seek to both minimize the sum of the squared error of the model on the training data (using ordinary least squares) but also to reduce the complexity of the model (like the number or absolute size of the sum of all coefficients in the model).
  - Two popular examples of regularization procedures for linear regression are:
    - Lasso Regression: where Ordinary Least Squares is modified to also minimize the absolute sum of the coefficients (called L1 regularization).
    - Ridge Regression: where Ordinary Least Squares is modified to also minimize the squared absolute sum of the coefficients (called L2 regularization).
  - These methods are effective to use when there is collinearity in your input values and ordinary least squares would overfit the training data.

# Linear regression

- **Linear Assumption.** Linear regression assumes that the relationship between your input and output is linear. It does not support anything else. This may be obvious, but it is good to remember when you have a lot of attributes. You may need to transform data to make the relationship linear (e.g. log transform for an exponential relationship).
- **Remove Noise.** Linear regression assumes that your input and output variables are not noisy. Consider using data cleaning operations that let you better expose and clarify the signal in your data. This is most important for the output variable and you want to remove outliers in the output variable ( $y$ ) if possible.
- **Remove Collinearity.** Linear regression will over-fit your data when you have highly correlated input variables. Consider calculating pairwise correlations for your input data and removing the most correlated.
- **Gaussian Distributions.** Linear regression will make more reliable predictions if your input and output variables have a Gaussian distribution. You may get some benefit using transforms (e.g. log or BoxCox) on your variables to make their distribution more Gaussian looking.
- **Rescale Inputs:** Linear regression will often make more reliable predictions if you rescale input variables using standardization or normalization.

# Feature Selection

- Many ML methods work better when the number of noisy features is small (curse of dimensionality)
- Feature selection is a way to detect uninformative features and remove them before applying the ML method.
- sklearn offers various feature selection methods in [sklearn.feature\\_selection](#)

# Some Notations:

$$\text{Data matrix: } \mathbf{X} = (\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N)^T = \begin{pmatrix} x_{11} & \cdots & x_{1M} \\ \vdots & \ddots & \vdots \\ x_{N1} & \cdots & x_{NM} \end{pmatrix}$$

*M features* *N items*

$$\text{Outcome: } \mathbf{y} = (y_1, \dots, y_N)^T$$

Outcome is a known attribute of the cases (rows). It can be a factor (e.g. healthy/diseased), a multiclass label (e.g. protein function) or a continuous variable (e.g. survival time).

# Some More Notations:

- In the classification task, we would like to predict the outcome  $y$  from the features  $X$ .
- We choose a classifier  $\mathcal{C}_\phi$  with some hyper parameters  $\phi$  (e.g  $k$  in k-NN, kernel in SVM, regularization in LR). Hyper parameters can be specified by user or learned in CV loop outside the classifier.
- Then the classifier seeks those internal parameters  $\theta$  that are best able to predict  $y$  from the features  $X$ :

$$\theta = \operatorname{argmin}_{\theta} \mathcal{L}(\mathcal{C}_\phi(X, \theta), y)$$

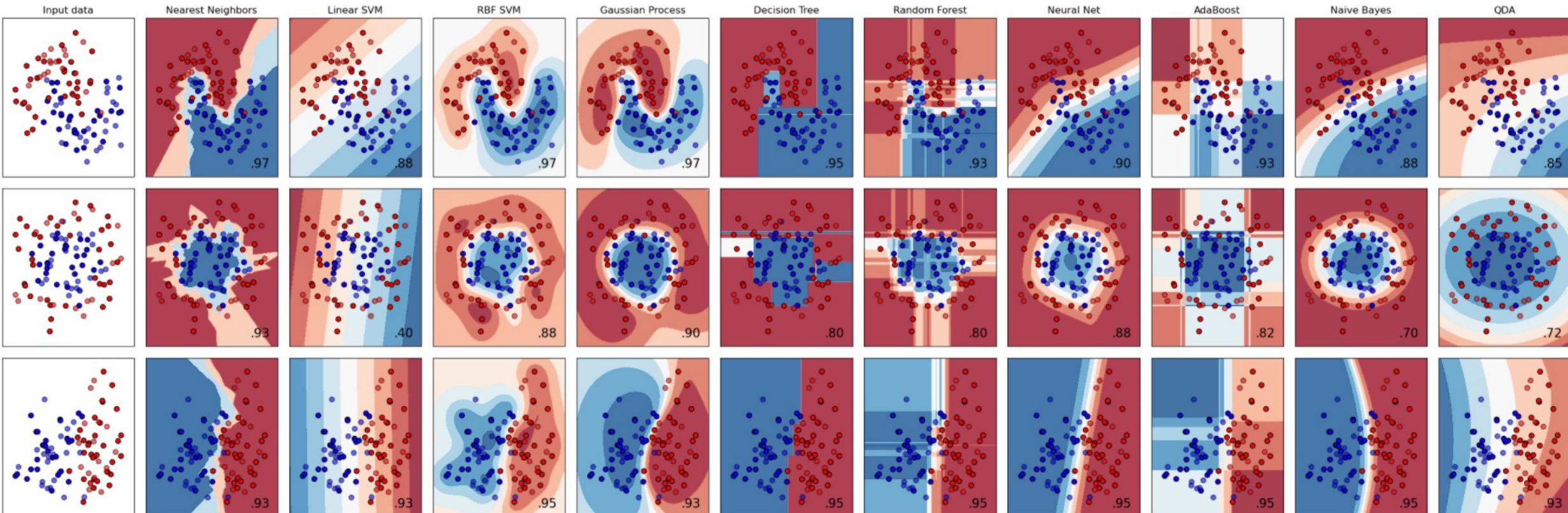
where the loss function  $\mathcal{L}$  measures the disagreement between predicted ( $\mathcal{C}_\phi(X, \theta)$ ) and known ( $y$ ) outcomes.

- The internal parameters  $\theta$  are learned by the classifier (e.g. the coefficients if  $\mathcal{C} = LogisticRegression$ , or support vector coefficients if  $\mathcal{C} = SVM$ ).

# Which Classifier Should We Choose?

- There is a mathematical theorem that says that there is no universally best classifier. It really depends on the data!
- One needs to test several classifiers for each dataset and compare the results.
- Best is to have a few favorite classifiers (e.g. kNN, LR, SVM), which you know well, and which are somehow complementary (local density based, linear, or inbetween)

# Which Classifier Should We Choose?



[sklearn classifier comparison](#)

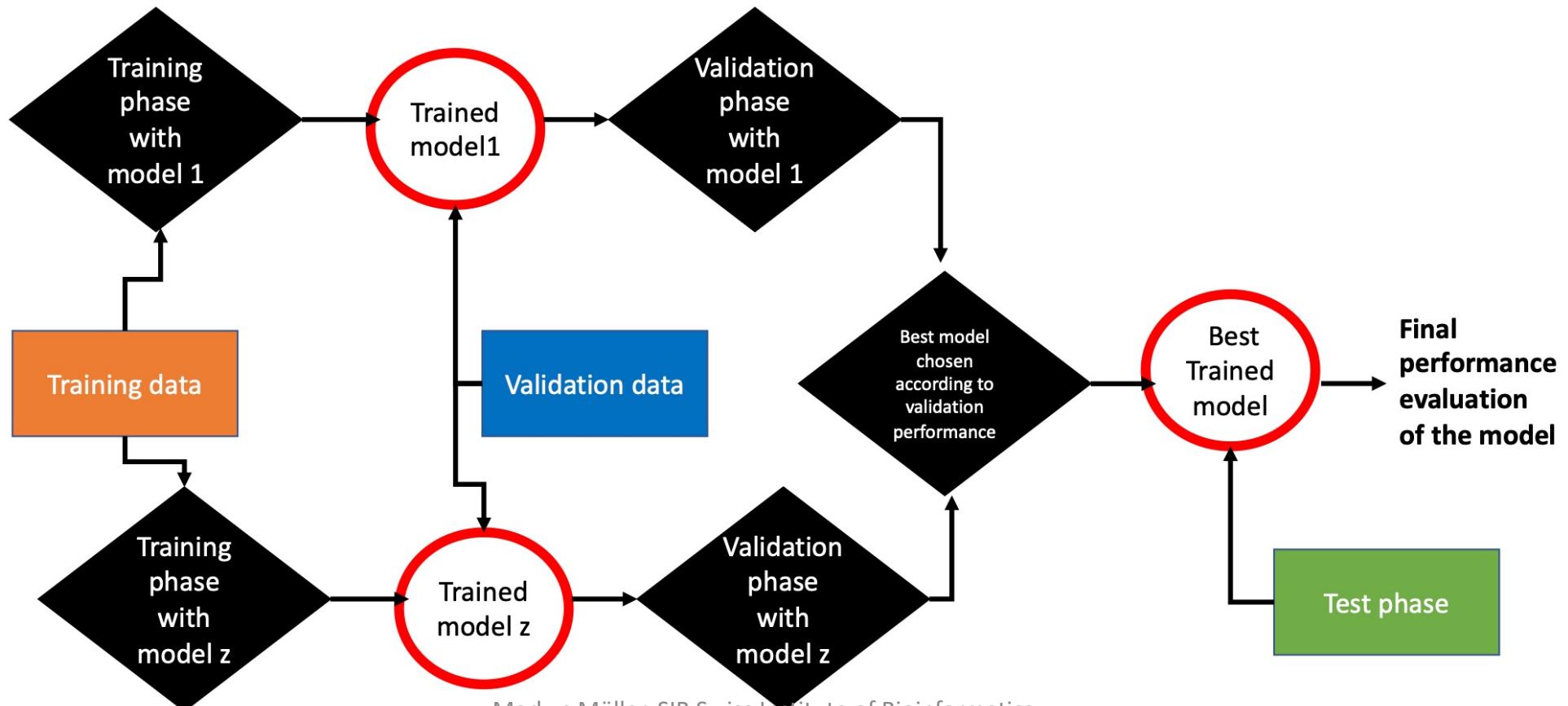
# Training, Validation and Test Sets

- Given a dataset, we can always find a classifier that perfectly predicts the outcome (for example the 1-NN classifier) for this dataset.
- However, the purpose of training a classifier is to give us accurate predictions on a new and independent test dataset. The performance on the training data may be too optimistic and cannot be reproduced on the test data. We call this “*overfitting the training data*”.
- When choosing a classifier, we will always select the one that performs best on the test or validation data.

# Training, Validation and Test Sets

sklearn [GridSearchCV](#)

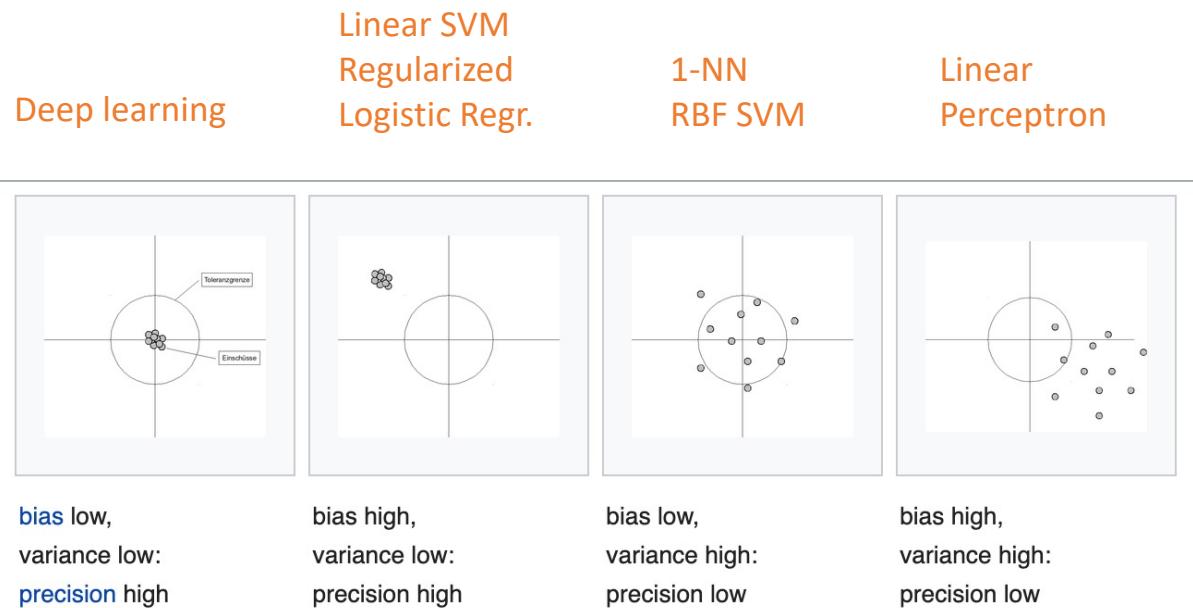
## A typical ML job



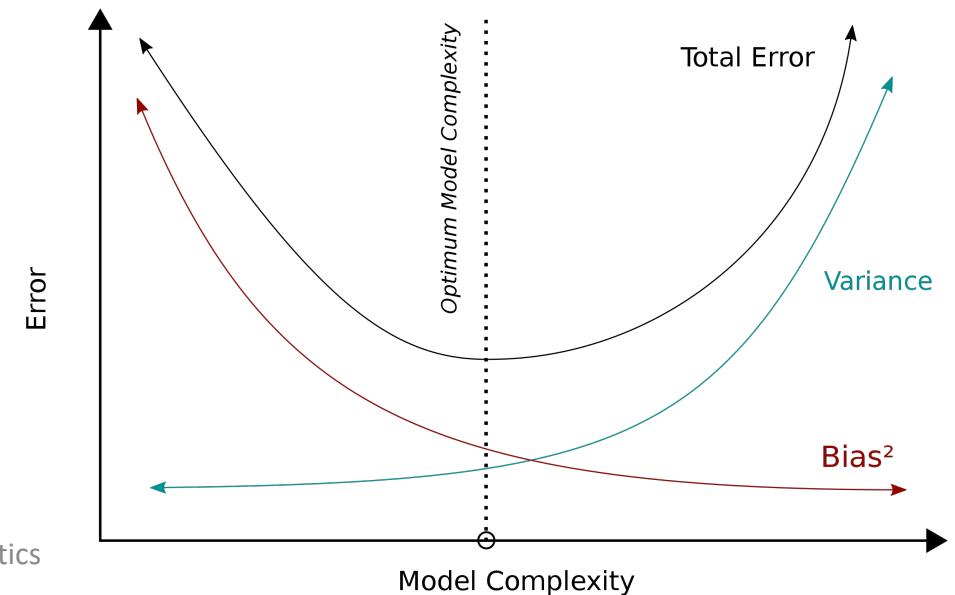
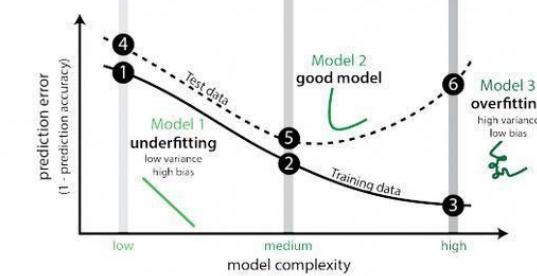
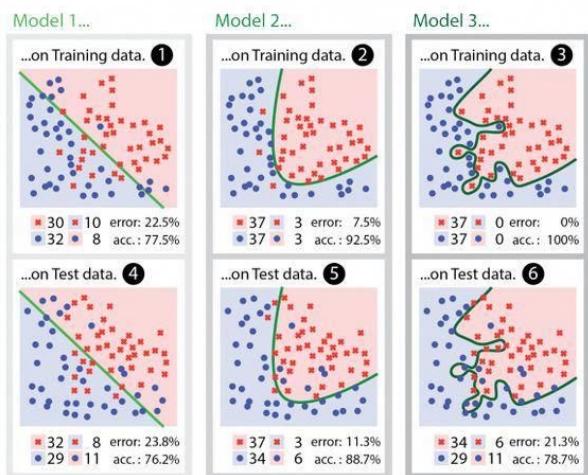
# Bias versus Variance Tradeoff

- All classifiers make errors and wrongly predict class labels for certain instances. The error can be split into two part: *bias* and *variance*.
- Variance: the classifier will depend on the subset of the data we choose for training. The more a classifier depends on the training data selection, the higher its variance.
- Bias: the variance usually decreases with the size of the training set. However, even for very large training sets, there will be an error left due to the bias of the classifier. For example, a linear classifier will never be perfectly able to separate classes with non-linear boundaries even if large datasets are available.
- Classifiers with low bias fit the training data well, but may have a large variance and low performance on the test set. Others have a larger bias, but are more stable and predictive for the test set.

# Bias versus Variance Tradeoff



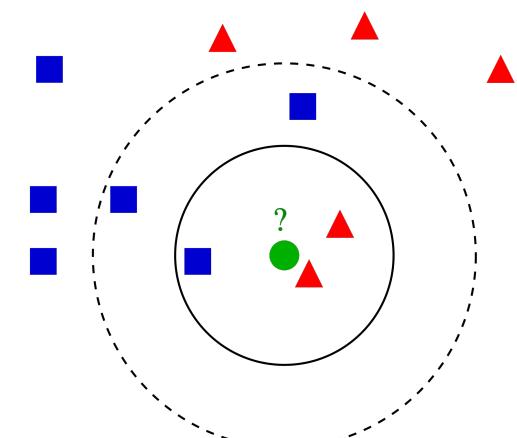
[https://en.wikipedia.org/wiki/Bias-variance\\_tradeoff](https://en.wikipedia.org/wiki/Bias-variance_tradeoff)



# K-Nearest Neighbor Classifier

- The k-Nearest neighbor (k-NN) classifier is probably the simplest non-trivial classifier. Given a training dataset  $X^{train}, y^{train}$  we predict the class of an instance  $x$  by taking the  $k$  nearest instances  $x_i^{train}, i \in kNN(x)$  according to some distance metric  $d(x, x_i^{train})$  and predict the majority class in  $y_i^{train}, i \in kNN(x)$
- Instead of simply predicting the majority class we can also weight the classes by  $1/d(x, x_i^{train})$ .
- The number of neighbors  $k$  we will determine with cross validation.

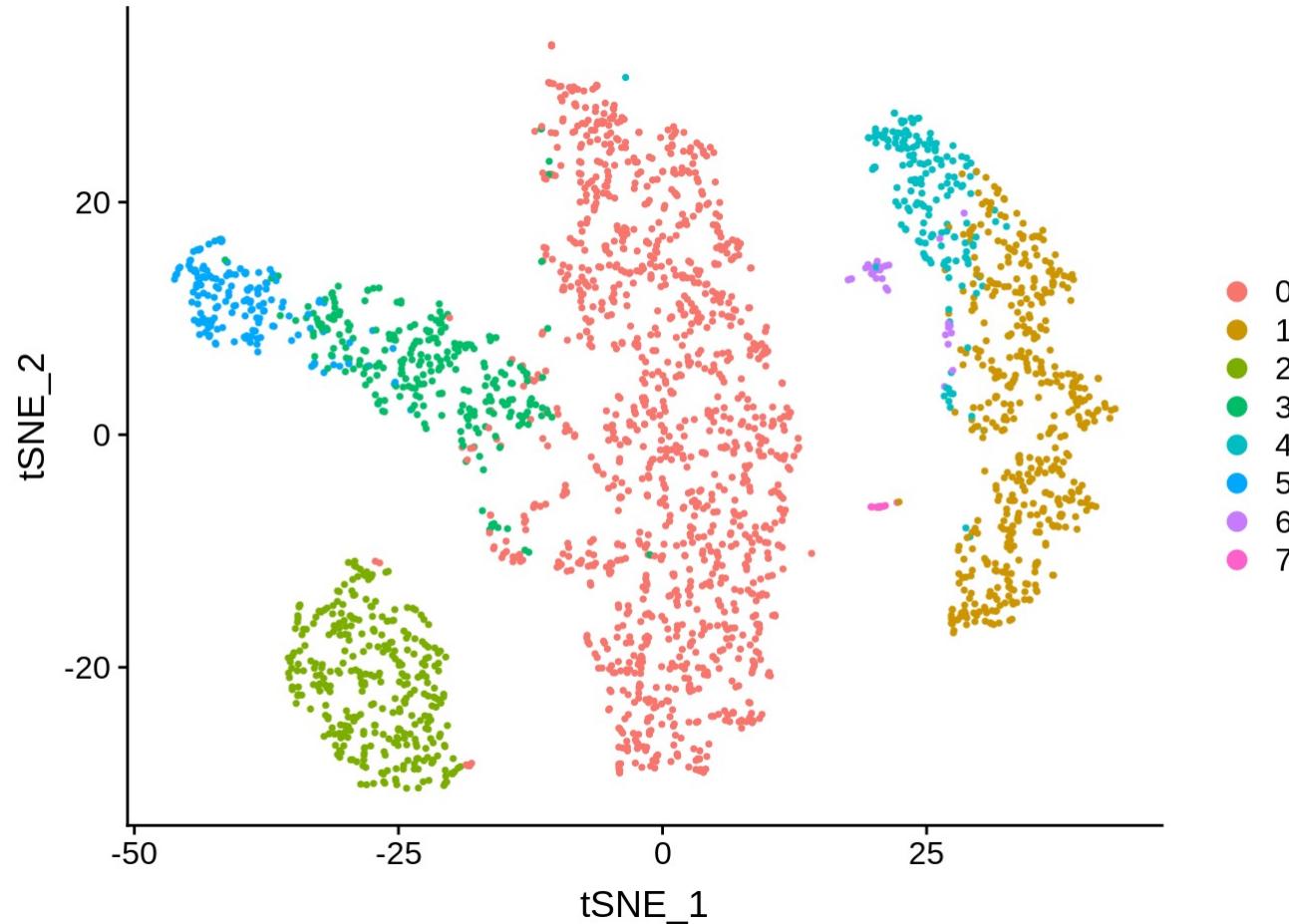
[https://en.wikipedia.org/wiki/K-nearest\\_neighbors\\_algorithm](https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm)



# k-Nearest Neighbors (KNN)

- Pros
  - Training phase much faster compared to other classification algorithms
  - No need to train a model for generalization
  - Can be useful in case of nonlinear data
  - Can be used for regression: the output value for the object is computed by the average of k closest neighbors value.
- Cons
  - Testing phase is slower and costlier in terms of time and memory
  - It requires large memory for storing the entire training dataset for prediction
  - It requires scaling of data because KNN uses the Euclidean distance between two data points to find nearest neighbors. Euclidean distance is sensitive to magnitudes. The features with high magnitudes will weight more than features with low magnitudes
  - It is not suitable for large dimensional data.

# k-Nearest Neighbors (KNN)

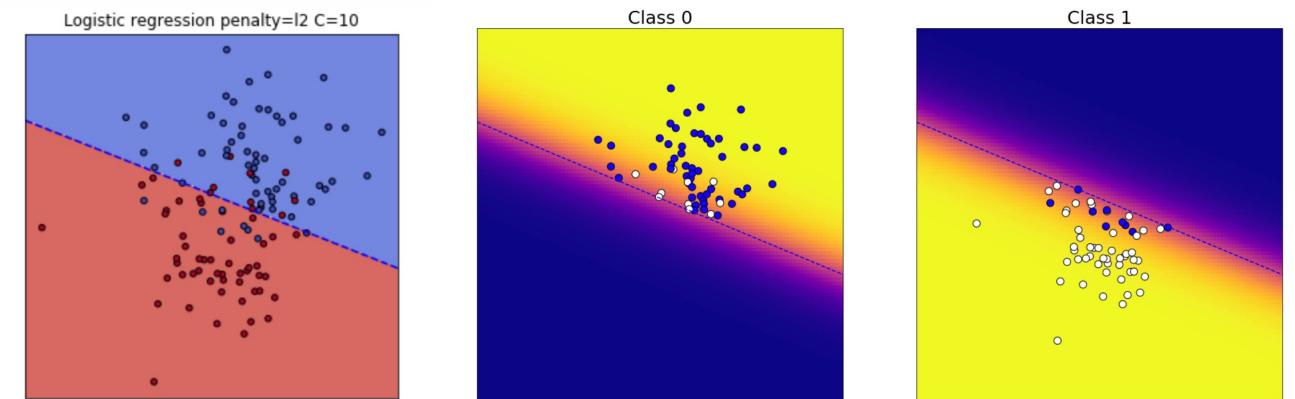
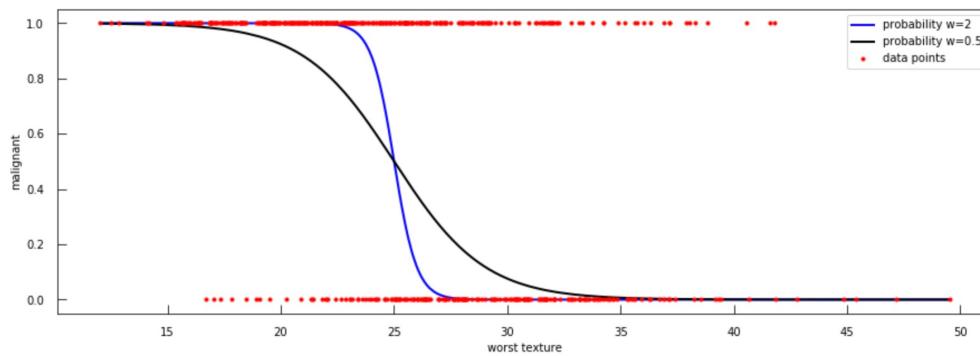


<https://scrnaseq-course.cog.sanger.ac.uk/>

Isabelle Dupanloup, BCF

# Logistic Regression

- In logistic regression we assume that our data is linearly separable by a hyperplane  $h = w_0 + \sum_{i=1}^M w_i x_i = w_0 + \mathbf{w} \cdot \mathbf{x} = 0$ .
- We assume a sigmoid shaped class probability distribution  $p$ , which only depends on the distance  $d$  to  $h$  :  $d = \frac{w_0 + \mathbf{w} \cdot \mathbf{x}}{\|\mathbf{w}\|}$



# Logistic Regression

- $p(\mathbf{x}|y=1) = \frac{1}{1+e^{-(w_0+\mathbf{w} \cdot \mathbf{x})}} \Leftrightarrow \log \frac{p}{1-p} = w_0 + \mathbf{w} \cdot \mathbf{x}$
- $p(\mathbf{x}|y=0) = 1 - p(\mathbf{x}|y=1)$
- Loss functions:

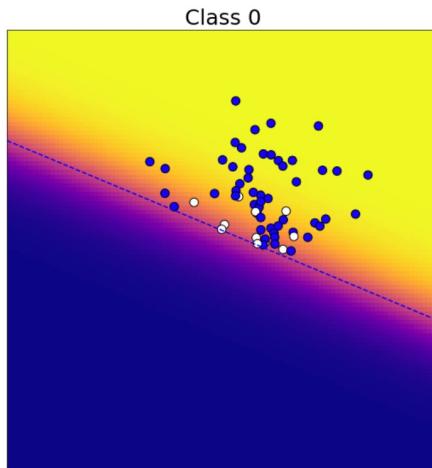
$$\mathcal{L}(w_0, \mathbf{w}, \mathbf{X}, \mathbf{y}) = -\frac{1}{N} \left\{ \sum_{i,y_i=1} \log p(\mathbf{x}_i|y=1) + \sum_{i,y_i=0} \log p(\mathbf{x}_i|y=0) \right\}$$

- Regularization: keep  $\mathbf{w}$  small and boundary smooth:

- $\mathcal{L}(w_0, \mathbf{w}, \mathbf{X}, \mathbf{y}) + \frac{1}{C} \sum_{i=1}^N |w_i|$  : lasso or 'l1'
- $\mathcal{L}(w_0, \mathbf{w}, \mathbf{X}, \mathbf{y}) + \frac{1}{C} \sum_{i=1}^N w_i^2$  : ridge or 'l2'
- $\mathcal{L}(w_0, \mathbf{w}, \mathbf{X}, \mathbf{y}) + \frac{1}{C} \sum_{i=1}^N \alpha |w_i| + (1-\alpha) w_i^2$  : elasticnet

# Logistic Regression

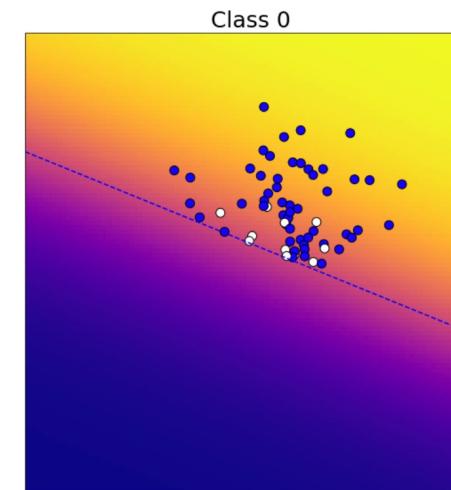
C=10



Probability

0.2 0.4 0.6 0.8

C=0.005



Probability

0.2 0.4 0.6 0.8

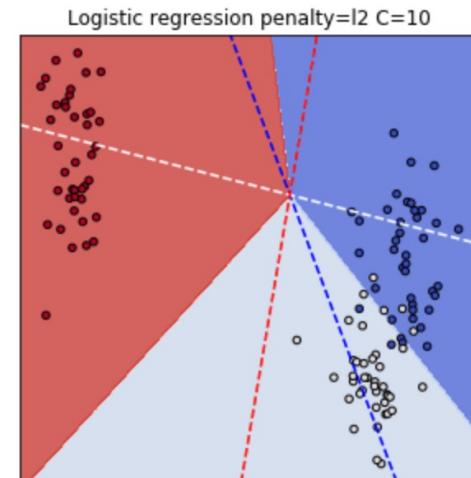
In high dimension with many noisy features the effect of regularization is more important

# Multinomial Logistic Regression ( $K$ classes)

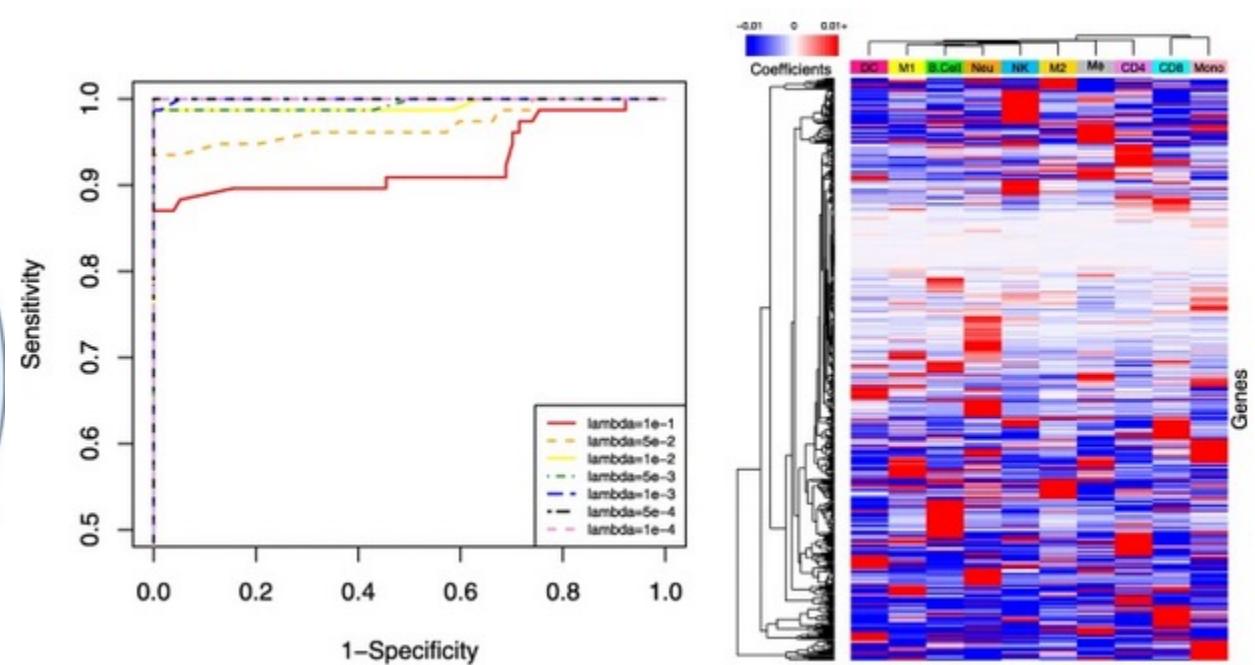
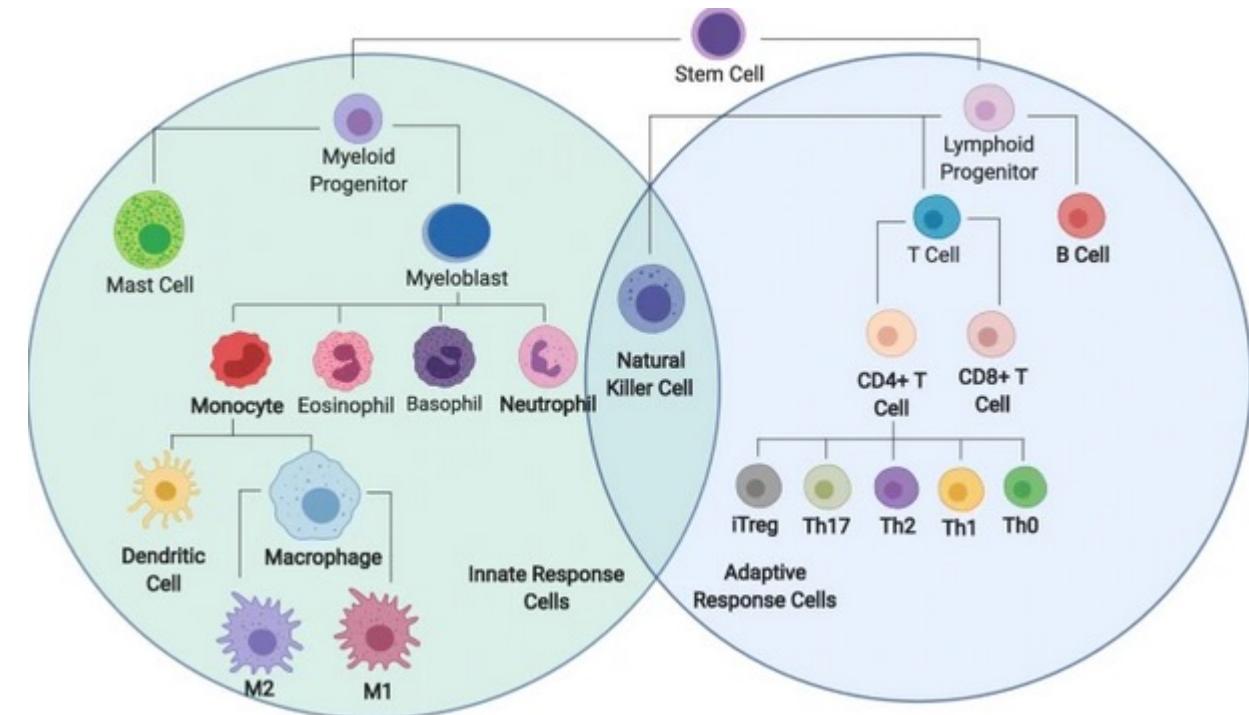
- $p(\mathbf{x}|y = k) = \frac{e^{w_{0k} + \mathbf{w}_k \cdot \mathbf{x}}}{\sum_{i=1}^K e^{w_{0i} + \mathbf{w}_i \cdot \mathbf{x}}}$
- **Class**  $y(\mathbf{x}) = \operatorname{argmax}_k p(\mathbf{x}|y = k) = \operatorname{argmax}_k (w_{0k} + \mathbf{w}_k \cdot \mathbf{x})$
- **Loss function:**

$$\mathcal{L}(w_{0k}, \mathbf{w}_k, \mathbf{X}, \mathbf{y}) = -\frac{1}{N} \sum_{i=1}^N \sum_{k=1}^K y_i^k \log p(\mathbf{x}_i | y = k);$$

$$y_i^k = \begin{cases} 1: y(\mathbf{x}_i) = k \\ 0: y(\mathbf{x}_i) \neq k \end{cases}$$



# Logistic regression



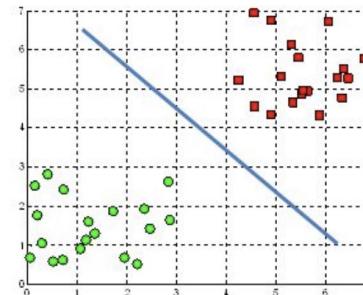
Torang et al. (2019) BMC Bioinformatics 20(1):433.

# Support Vector Machines (SVM)

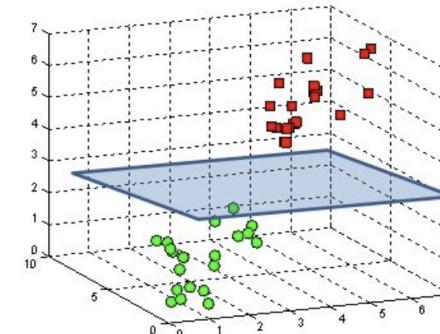
- Separating hyperplanes
- Soft margin classification
- Linear SVM
- Non-linear SVM & Kernels

# Largest Margin Separating Hyperplane

A hyperplane in  $\mathbb{R}^2$  is a line

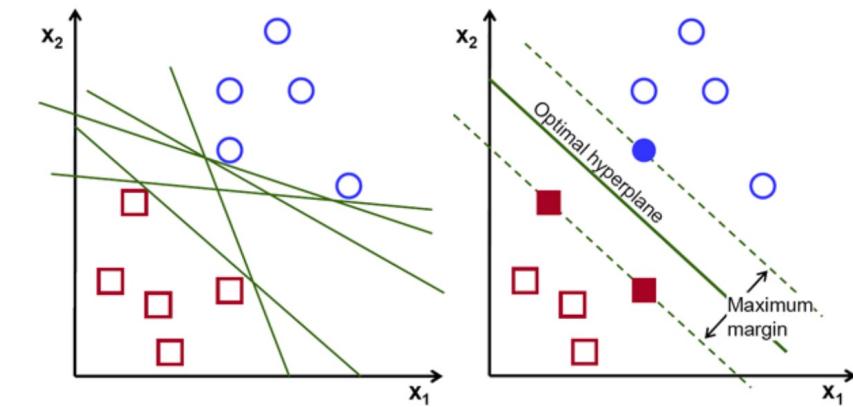


A hyperplane in  $\mathbb{R}^3$  is a plane

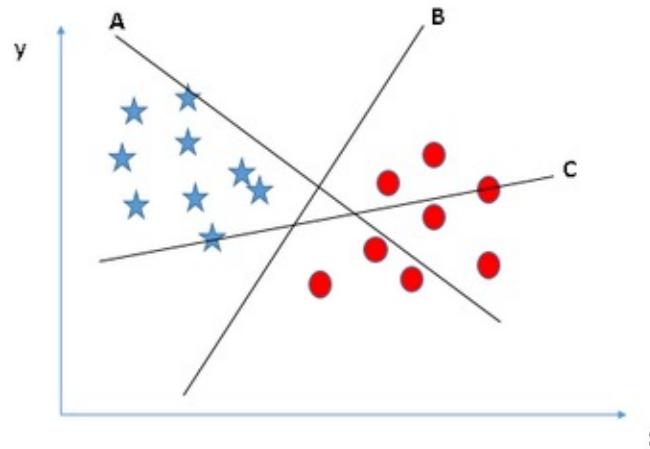


Hyperplanes are linear decision boundaries that help classify the data points.

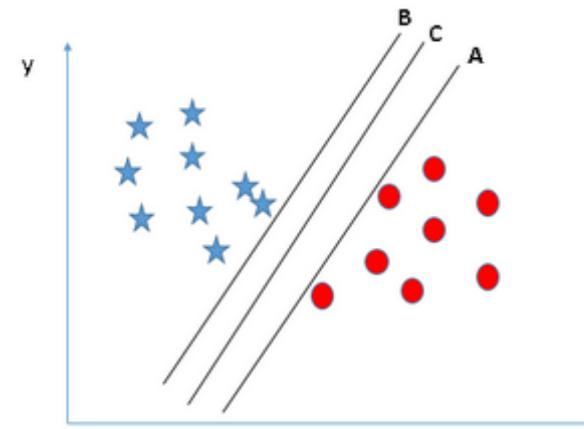
- Objective:
  - find a hyperplane in an M-dimensional space ( $M =$  number of features) that distinctly classifies the data points
  - find a plane that has the maximum margin, i.e the maximum distance between data points of both classes
- Maximizing the margin distance provides some reinforcement so that future data points can be classified with more confidence



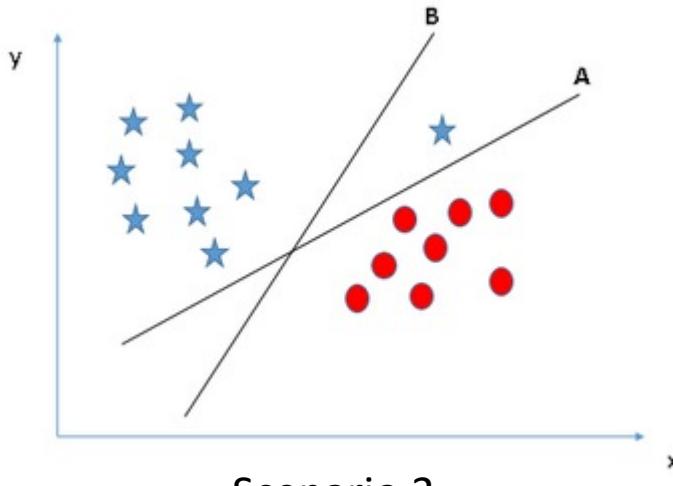
# Support vector machines (SVM)



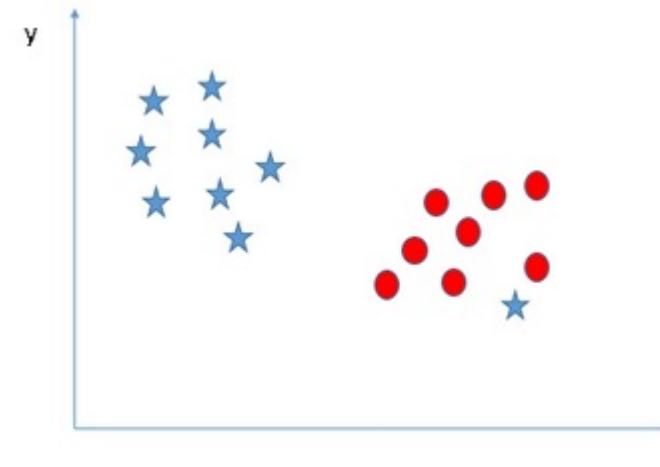
Scenario 1



Scenario 2



Scenario 3

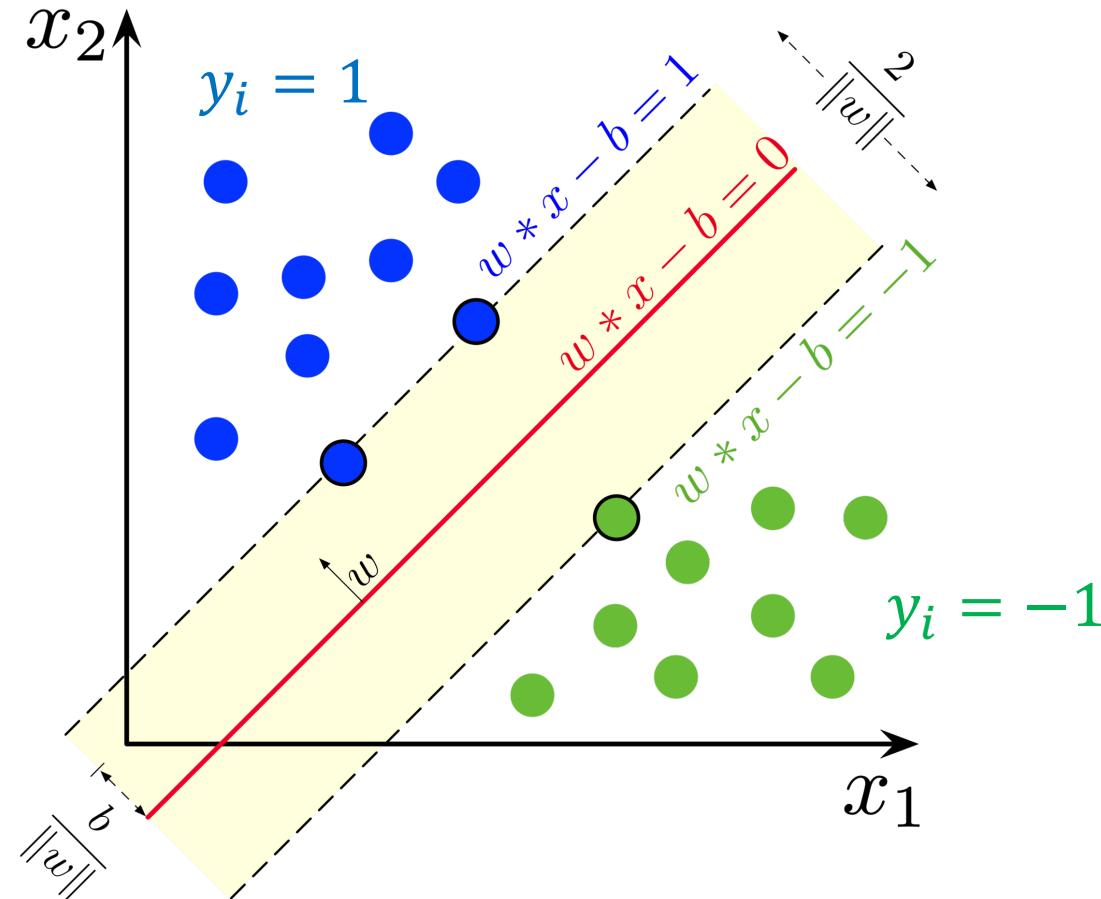


Scenario 4

# Largest Margin Separating Hyperplane

- Margin  $m = \frac{2}{\|w\|}$
- $\max_w m$   
subject to  $y_i(w \cdot x_i - b) \geq 1, i = 1..N$
- $\min \|w\|^2$   
subject to  $y_i(w \cdot x_i - b) \geq 1, i = 1..N$

[https://en.wikipedia.org/wiki/Support-vector\\_machine](https://en.wikipedia.org/wiki/Support-vector_machine)

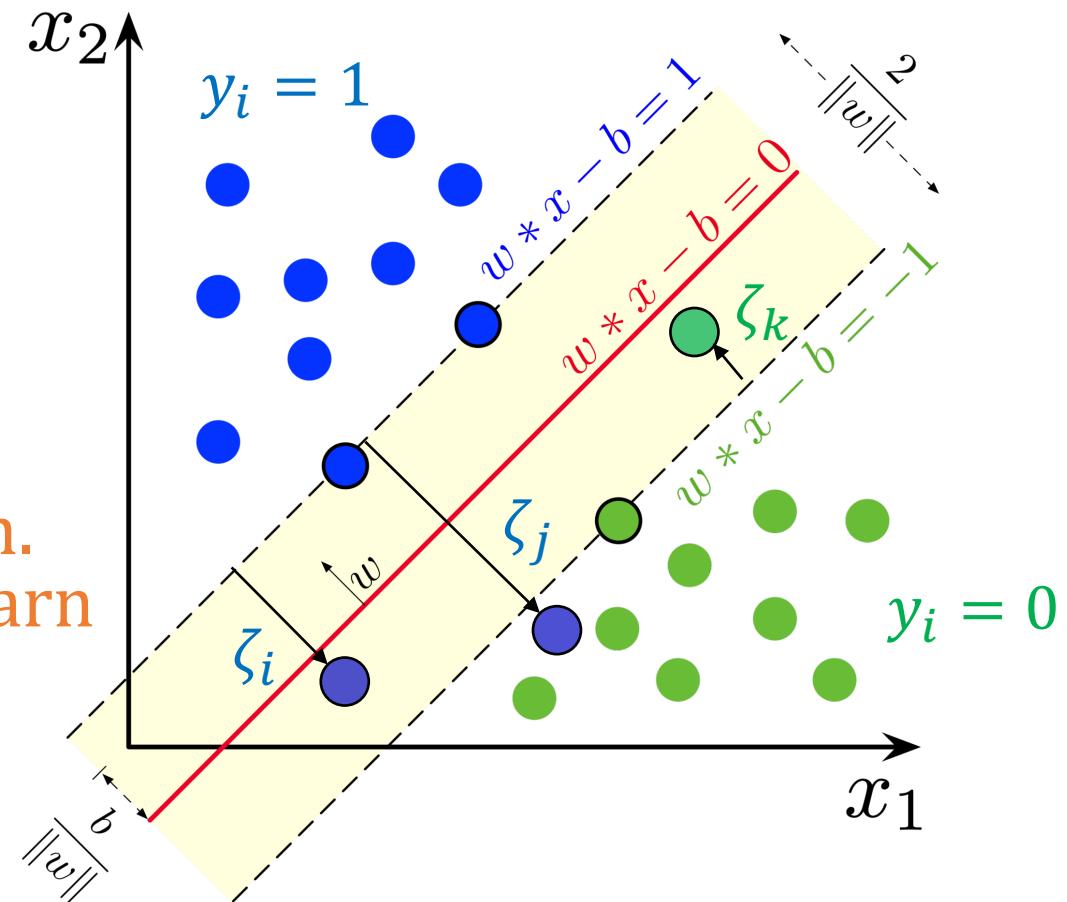


# Soft Margin Separating Hyperplane

- $\zeta_i = \max(0, 1 - y_i(\mathbf{w} \cdot \mathbf{x}_i - b))$
- $\min_{\mathbf{w}} \{\|\mathbf{w}\|^2 + C \sum_{i=1}^N \zeta_i\}$

subject to  $y_i(\mathbf{w} \cdot \mathbf{x}_i - b) \geq 1 - \zeta_i, \zeta_i \geq 0$   
 $i = 1 \dots N$

- This is a quadratic programming problem.  
 Efficient algorithms implemented in sklearn  
 LinearSVC can solve this in  $O(NM)$
- Non-linear SVM (see below) requires  
 $O(N^\alpha M)$ , where  $2 < \alpha < 3$

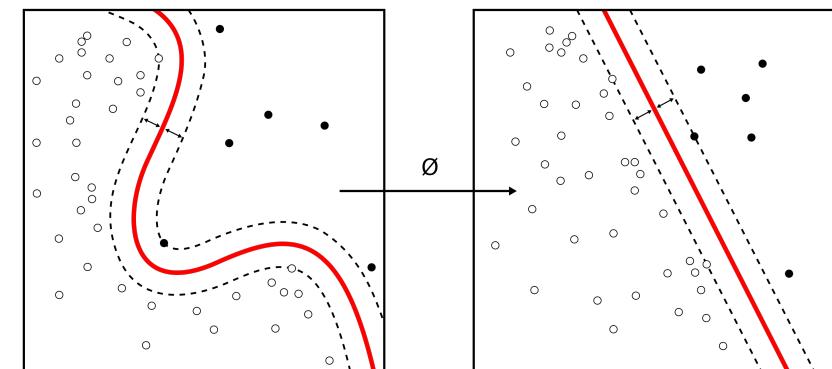


# Non-Linear Support vector machines (SVM)

- Dual form:  $\min_{\alpha} \left\{ -\frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N \alpha_i \alpha_j y_i y_j \mathbf{x}_i \cdot \mathbf{x}_j + \sum_{i=1}^N \alpha_i \right\}$   
subject to:

$$\sum_{i=1}^N y_i \alpha_i = 0 \text{ and } 0 \leq \alpha_i \leq C, i = 1 \dots N.$$

- $\mathbf{w} = \sum_{i=1}^N \alpha_i y_i \mathbf{x}_i; S = \{i | \alpha_i > 0\}; b = \frac{1}{|S|} \sum_{i \in S} (y_i - \mathbf{w} \cdot \mathbf{x}_i)$
- Prediction:  $f(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x} - b = \sum_{i=1}^N \alpha_i y_i \mathbf{x}_i \cdot \mathbf{x} - b$
- Kernel trick:  $\mathbf{x} \cdot \mathbf{y} \mapsto K(\mathbf{x}, \mathbf{y})$
- Boser, Guyon & Vapnik, Proc. 5th COLT, 1992



# Non-Linear Support vector machines (SVM)

- No need for explicit mapping:  $\mathbf{x}' \mapsto \varphi(\mathbf{x})$

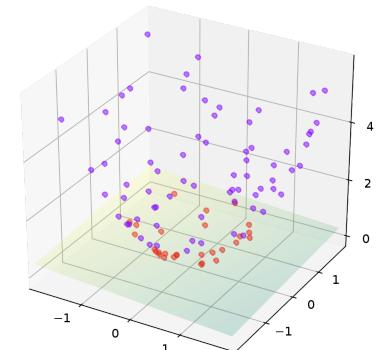
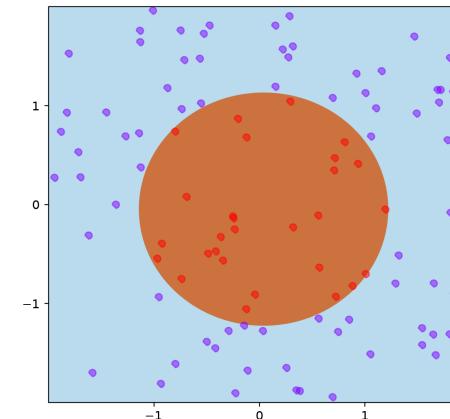
- Kernel trick:  $\mathbf{x} \cdot \mathbf{y} \mapsto K(\mathbf{x}, \mathbf{y})$

- Linear:  $K(\mathbf{x}, \mathbf{y}) = \mathbf{x} \cdot \mathbf{y}$

- Polynomial:  $K(\mathbf{x}, \mathbf{y}) = (\gamma \mathbf{x} \cdot \mathbf{y} + r)^d$

- Radial:  $K(\mathbf{x}, \mathbf{y}) = e^{-\gamma \|\mathbf{x} - \mathbf{y}\|^2}$

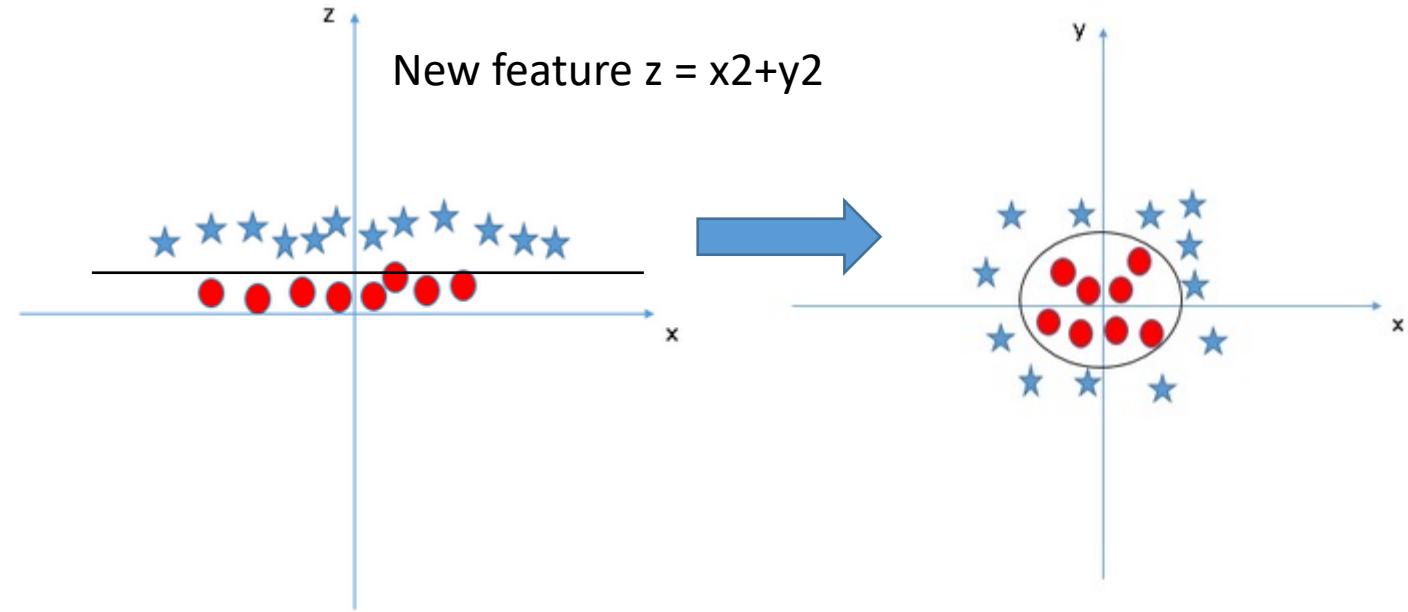
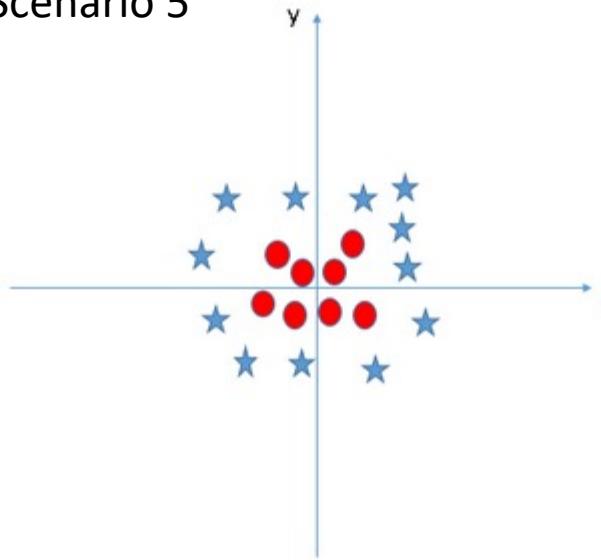
- Sigmoid:  $K(\mathbf{x}, \mathbf{y}) = \tanh(\gamma \mathbf{x} \cdot \mathbf{y} + r)$



$$\varphi((a, b)) = (a, b, a^2 + b^2)$$

# Support vector machines (SVM)

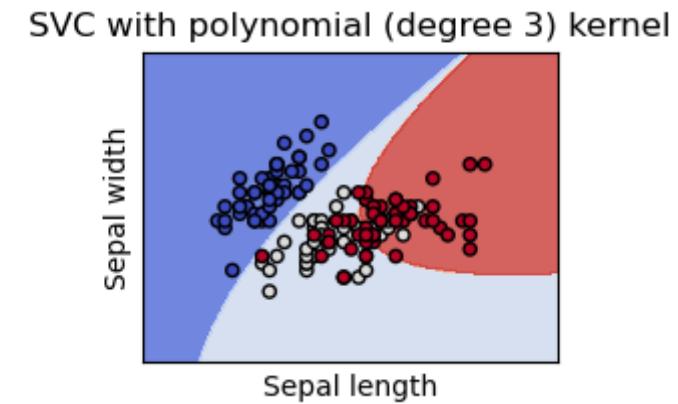
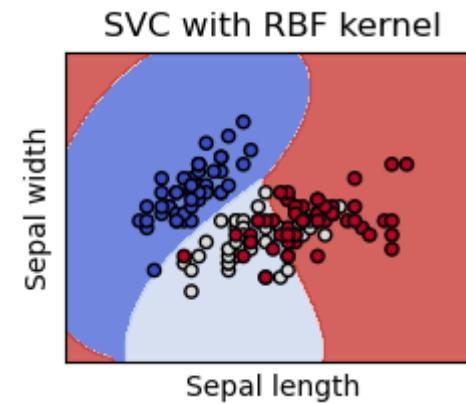
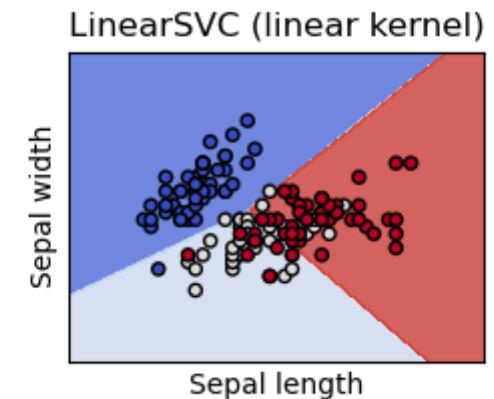
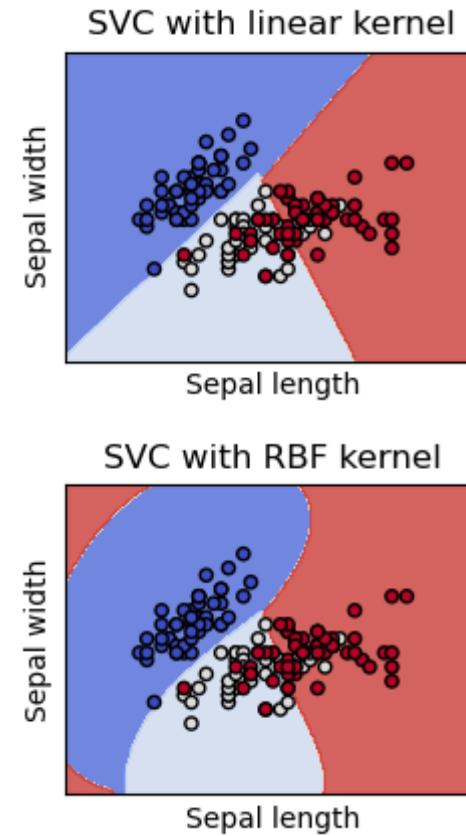
Scenario 5



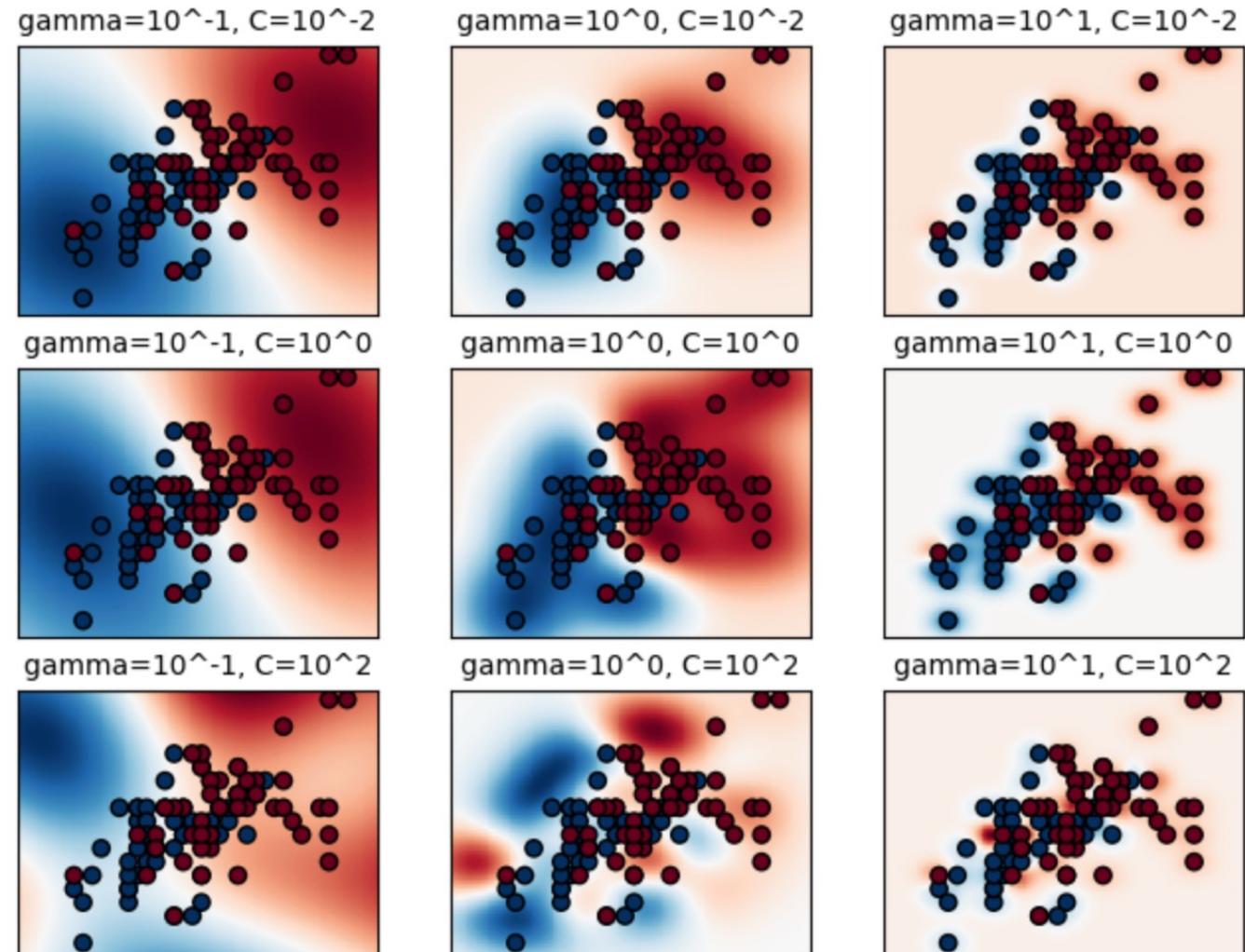
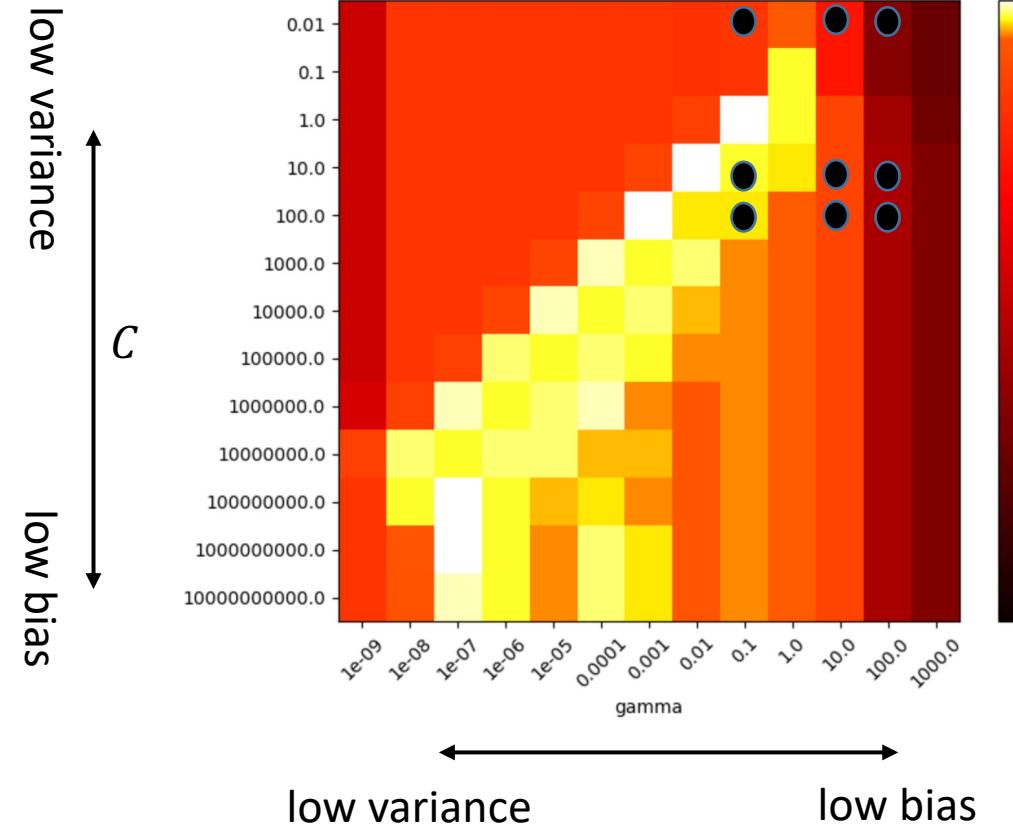
SVM kernel is a function that takes low dimensional input space and transforms it to a higher dimensional space i.e. it converts not separable problem to separable problem.

# Support vector machines (SVM)

- Hyperparameters
  - Type of kernel (linear, polynomial, RBF; etc.)
  - Parameters of the kernel
  - Type of regularization
  - $\lambda$ : regularization strength

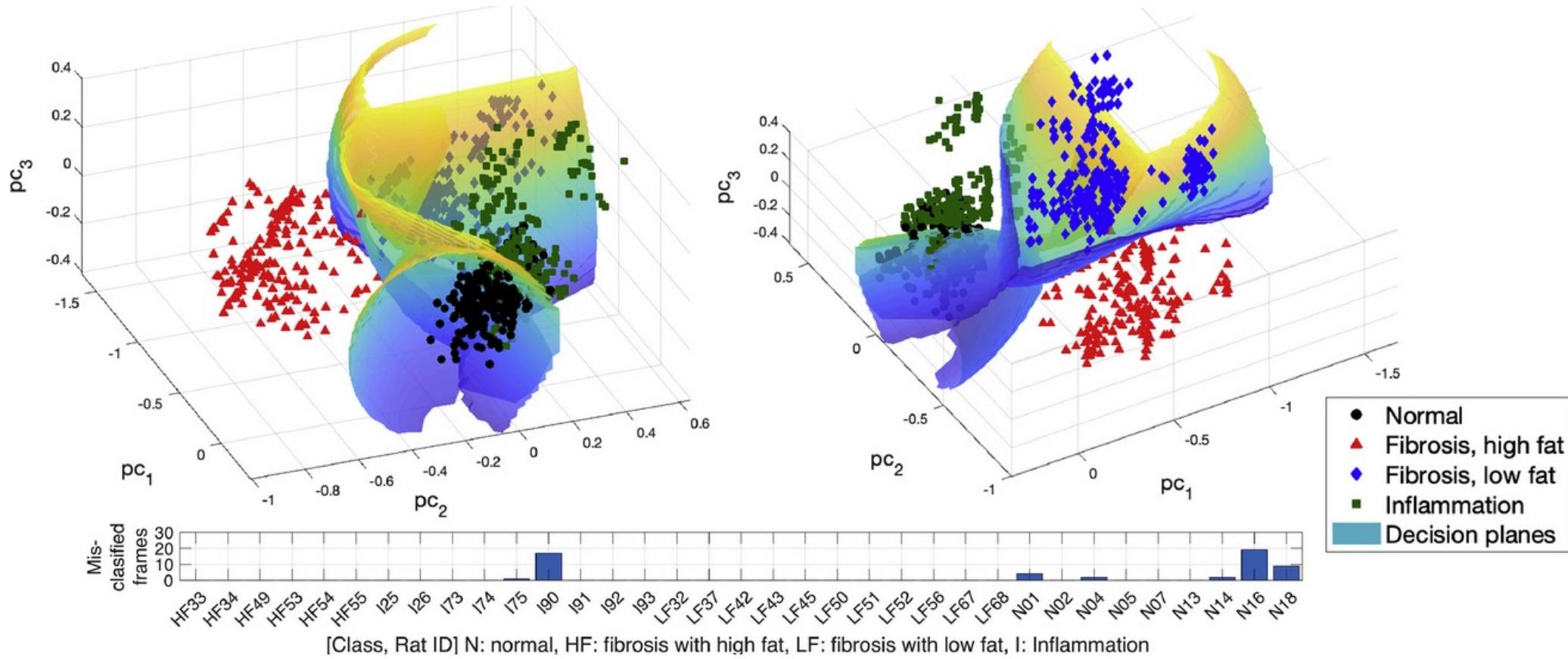


# Radial Basis Kernel



[sklearn SVC radial basis](#)

# Support vector machines (SVM)



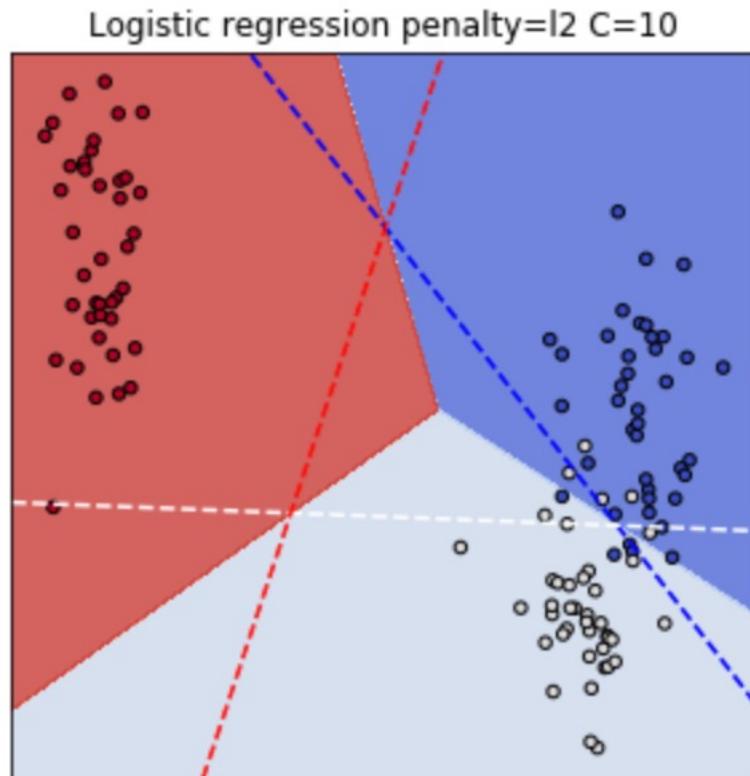
Baek et al. (2020) Ultrasound Med Biol 46(12):3379-3392.

# Multiclass Classification

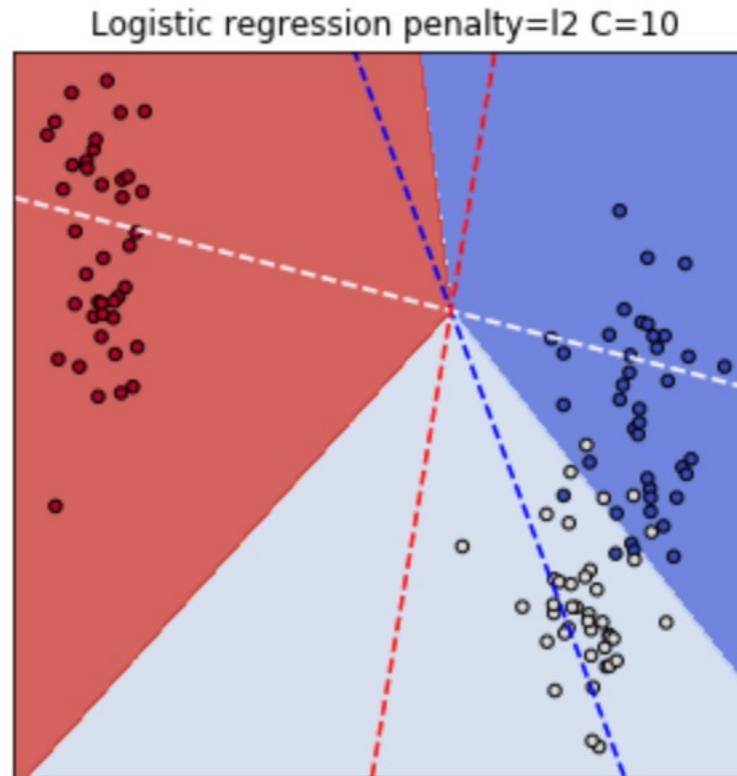
- If more than two classes there are several options to deal with them:
  - **Ovo** (one-versus-one): split the problem into  $K(K - 1)/2$  classification tasks. At prediction time we choose the class with the most votes. Implemented in SVC. See also sklearn OneVsOneClassifier.
  - **Ovr** (one-versus-rest): split the problem into  $K$  classification tasks. At prediction time we choose the class with the highest prediction score. Implemented in LogisticRegression. See also OneVsRestClassifier.
  - **Inherently multiclass**: Some classifiers have inherent support for multiclass problems (e.g. k-NN, LogisticRegression, DecisionTree, RandomForest)
- Check: <https://scikit-learn.org/stable/modules/multiclass.html>

# Multiclass Classification

ovr



multinomial

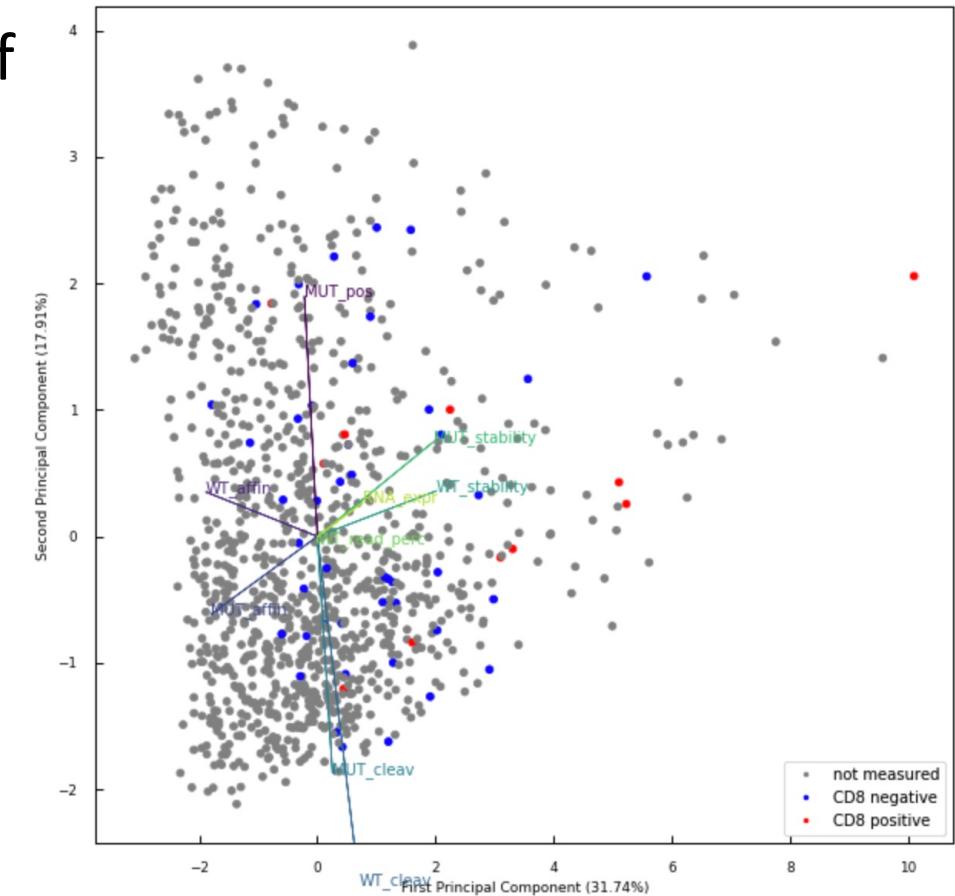


K neighbors k=5, uniform



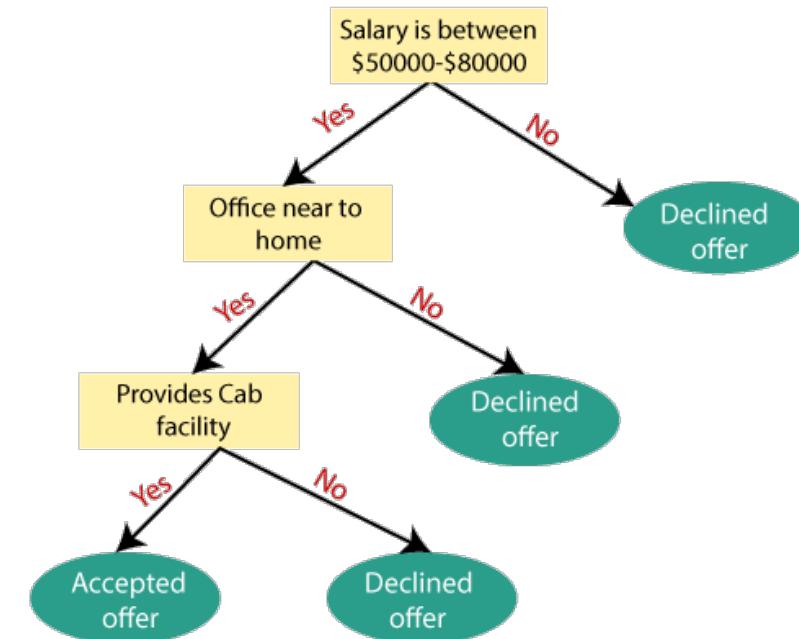
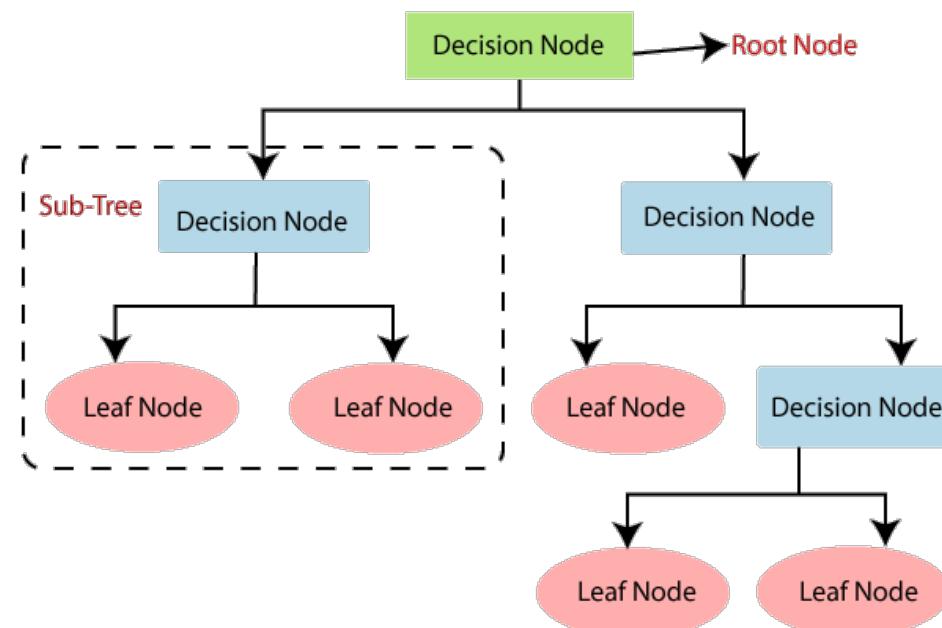
# Unbalanced Classes

- Very often in ML, there are different number of instances in each class
- This can be a problem for a ML classification method if classes are well mixed
- If ‘accuracy’ is used in a CV loop a classifier will try to predict the abundant class correctly, and ignore the rare class.
- Solution: Use ‘roc\_auc’ or work with ‘*class\_weight*’ parameter.
  - *class\_weight*=‘balanced’ :  $cw_k = \frac{N}{K \cdot \sum_{i=1}^N y_i^k}$
  - Set ‘*class\_weight*’ or learn in CV loop



# Decision trees

- Supervised learning technique that can be used for both classification and Regression problems, but mostly it is preferred for solving Classification problems
- Tree-structured classifier, where internal nodes represent the features of a dataset, branches represent the decision rules and each leaf node represents the outcome.



# Decision trees

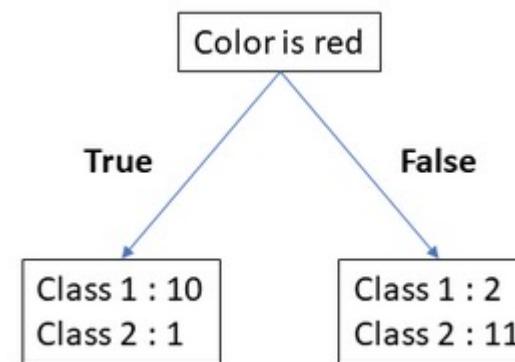
- Algorithm:
  - hierarchical sequence of questions on your features that can be answered by yes or no and that subdivides the data into subgroups on which a new question is asked, and so on and so on
  - tests all the features and chooses the most discriminative feature (in terms of the labels) to divide the data into subsets (impurity: Shannon entropy, Gini entropy)



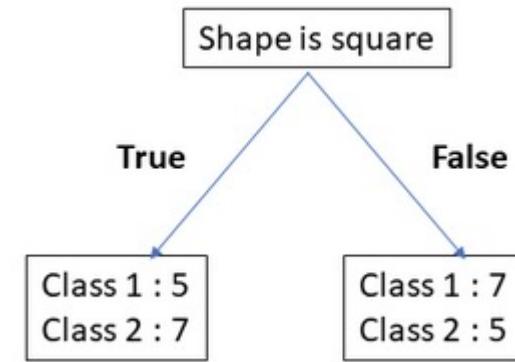
Features:

Color: blue or red

Shape: square or circle

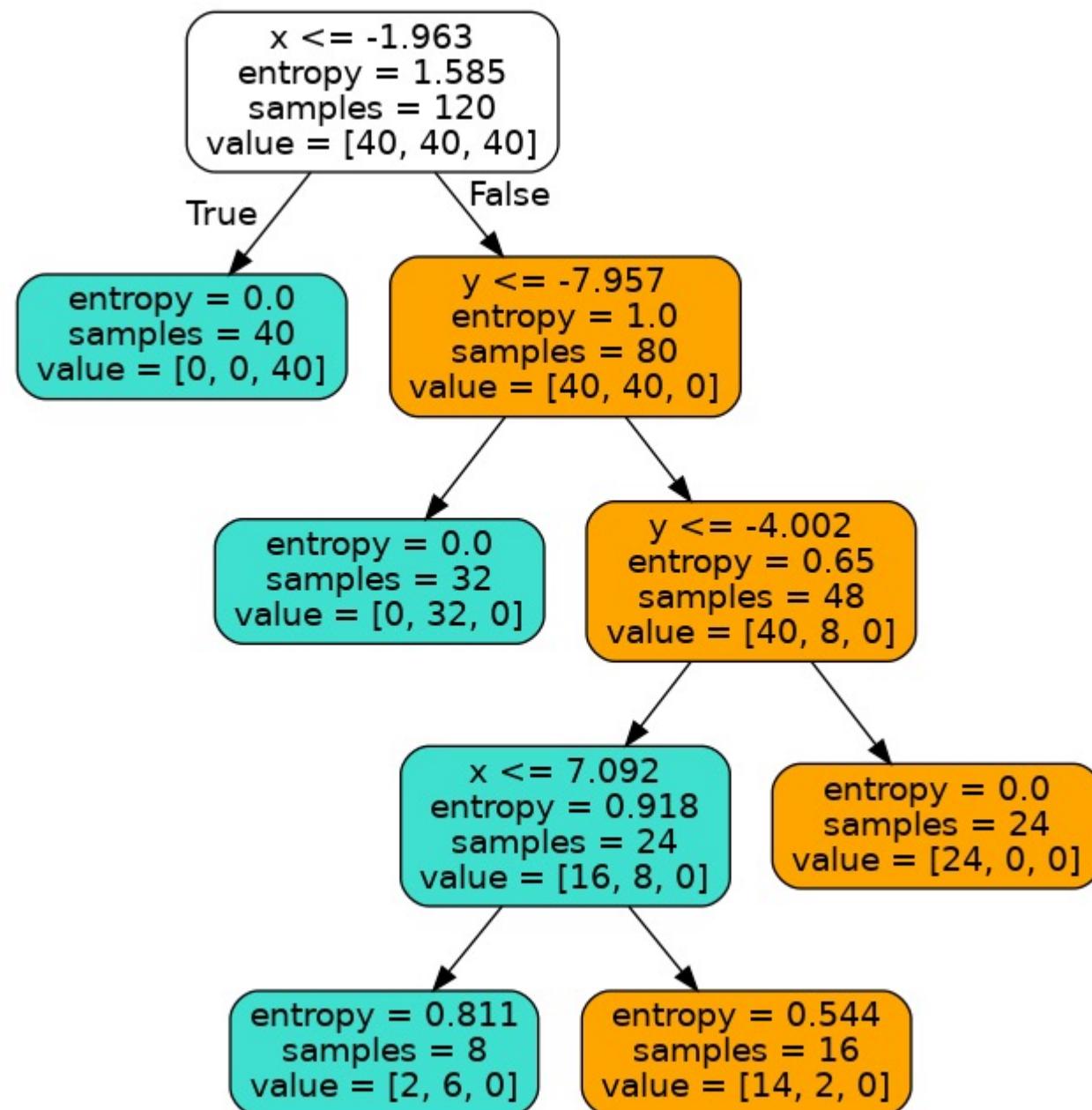


Those groups are  
highly skewed toward  
certain class



Those groups are well  
mixed

# Decision trees



# Decision trees

- Hyperparameters:
  - Criterion: function to measure the quality of a split
  - Max Tree depth: maximum depth of your model. Maximum depth refers to the length of the longest path from a root to a leaf
  - Min Splitting of nodes: minimum number of samples (i.e. training inputs) required to split an internal node
  - Min Splitting of leaves: minimum number of samples (i.e. training inputs) required to be at a leaf node
  - Maximum number of features to consider when looking for the best split: the search for a split does not stop until at least one valid partition of the node samples is found, even if it requires to effectively inspect more than max\_features features.

$$GiniIndex = 1 - \sum_j p_j^2$$

$$Gini_{min} = 1 - (1^2) = 0$$

$$Gini_{max} = 1 - (0.5^2 + 0.5^2) = 0.5$$

$$Entropy = - \sum_j p_j \cdot \log_2 p_j$$

$$Entropy_{min} = -1 \cdot \log_2(1) = 0$$

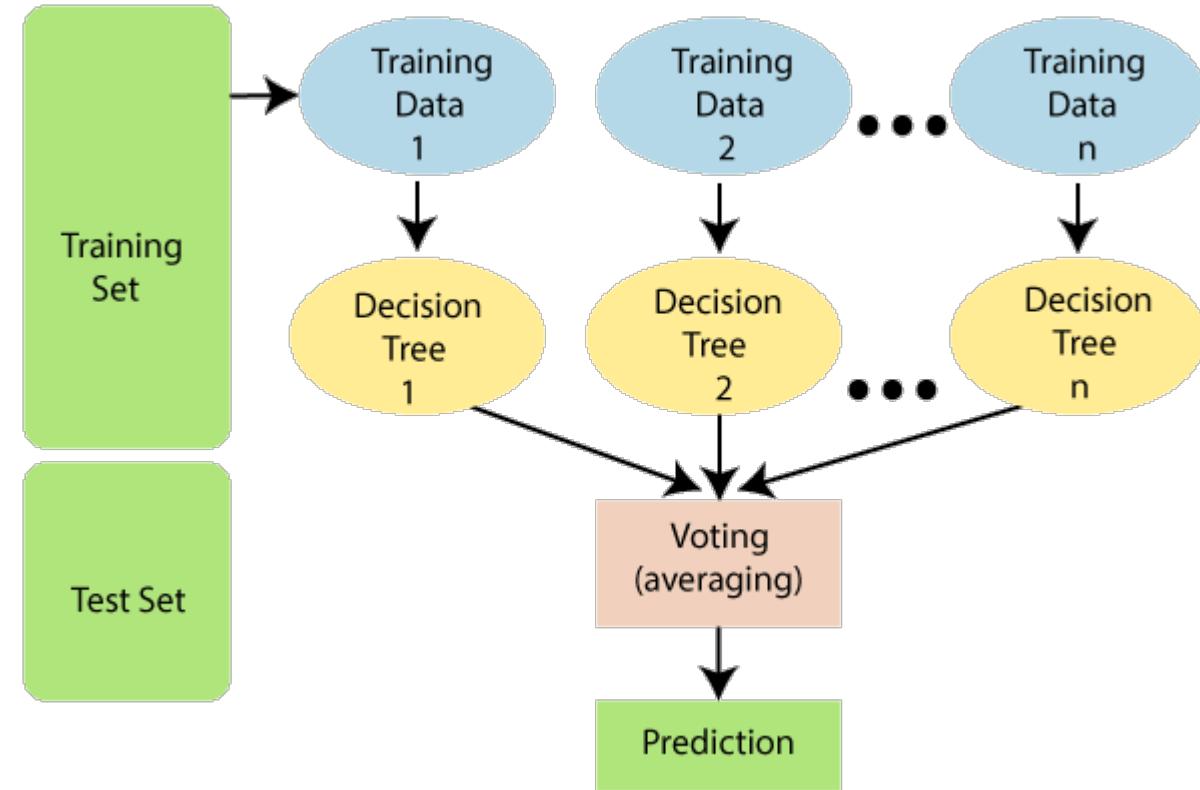
$$Entropy_{max} = -0.5 \cdot \log_2(0.5) - 0.5 \cdot \log_2(0.5) = 1$$

# Random forest

- popular machine learning algorithm which can be used for both Classification and Regression problems in ML
- based on the concept of **ensemble learning**, which is a process of *combining multiple classifiers to solve a complex problem and to improve the performance of the model.*
- Random Forest is a classifier that contains a number of decision trees on various subsets of the given dataset and takes the average to improve the predictive accuracy of that dataset.
- Instead of relying on one decision tree, the random forest takes the prediction from each tree and based on the majority votes of predictions, and it predicts the final output.
- The greater number of trees in the forest leads to higher accuracy and prevents the problem of overfitting.

# Random forest

- Bootstrap (select the features and the training samples at random, with replacement, to create the  $n$  training datasets)
- Hyperparameters:
  - Same as for the decision trees
  - Number of trees in the forest



# Random forest

› Proc Natl Acad Sci U S A. 2019 Sep 3;116(36):18119-18125. doi: 10.1073/pnas.1813645116.  
Epub 2019 Aug 16.

## Identification of the expressome by machine learning on omics data

Ryan C Sartor <sup>1</sup>, Jaclyn Noshay <sup>2</sup>, Nathan M Springer <sup>2</sup>, Steven P Briggs <sup>3</sup>

Affiliations + expand

PMID: 31420517 PMCID: PMC6731682 DOI: [10.1073/pnas.1813645116](https://doi.org/10.1073/pnas.1813645116)

[Free PMC article](#)

### Abstract

Accurate annotation of plant genomes remains complex due to the presence of many pseudogenes arising from whole-genome duplication-generated redundancy or the capture and movement of gene fragments by transposable elements. Machine learning on genome-wide epigenetic marks, informed by transcriptomic and proteomic training data, could be used to improve annotations through classification of all putative protein-coding genes as either constitutively silent or able to be expressed. Expressed genes were subclassified as able to express both mRNAs and proteins or only RNAs, and CG gene body methylation was associated only with the former subclass. More than 60,000 protein-coding genes have been annotated in the reference genome of maize inbred B73. About two-thirds of these genes are transcribed and are designated the filtered gene set (FGS). Classification of

Isabelle Dupanloup, BCF

# Adaptive Boosting (AdaBoost)

- AdaBoost algorithms can be used for both classification and regression problem
- Combine multiple “weak classifiers” into a single “strong classifier”
- Based on the concept of **ensemble learning**: power of ensembling is such that we can still build powerful ensemble models even when the individual models in the ensembles are extremely simple.
- Boosting algorithms try to build a strong learner (predictive model) from the mistakes of several weaker models.
- Boosting basically tries to reduce the bias error which arises when models are not able to identify relevant trends in the data. This happens by evaluating the difference between the predicted value and the actual value.
- You start by creating a model from the training data. Then, you create a second model from the previous one by trying to reduce the errors from the previous model. Models are added sequentially, each correcting its predecessor, until the training data is predicted perfectly or the maximum number of models have been added.

# Adaptive Boosting (AdaBoost)

- The weak learners in AdaBoost are decision trees with a single split, called **decision stumps**.
- Stumps alone are not a good way to make decisions. A full-grown tree combines the decisions from all variables to predict the target value. A stump, on the other hand, can only use one variable to make a decision
- Difference between Random Forest and AdaBoost:
  - AdaBoost uses a forest of stumps rather than trees.
  - AdaBoost stumps have different contributions to the prediction.
  - In AdaBoost, models are added sequentially, each correcting its predecessor.
- Each stump is made by taking the mistakes of the previous stumps into account by applying an importance weight on each sample. At the beginning of the algorithm, all the samples have the same weight but then after the first stump, the missclassified points will have a higher weight so that the next stumps really focuses on predicting them well.

# Adaptive Boosting (AdaBoost)

1. A weak classifier (e.g. a decision stump) is made on top of the training data based on the weighted samples. Here, the weights of each sample indicate how important it is to be correctly classified. Initially, for the first stump, we give all the samples equal weights.
2. We create a decision stump for each variable and see how well each stump classifies samples to their target classes.
3. More weight is assigned to the incorrectly classified samples so that they're classified correctly in the next decision stump. Weight is also assigned to each classifier based on the accuracy of the classifier, which means high accuracy = high weight !
4. Reiterate from Step 2 until all the data points have been correctly classified, or the maximum iteration level has been reached.

# Adaptive Boosting (AdaBoost)

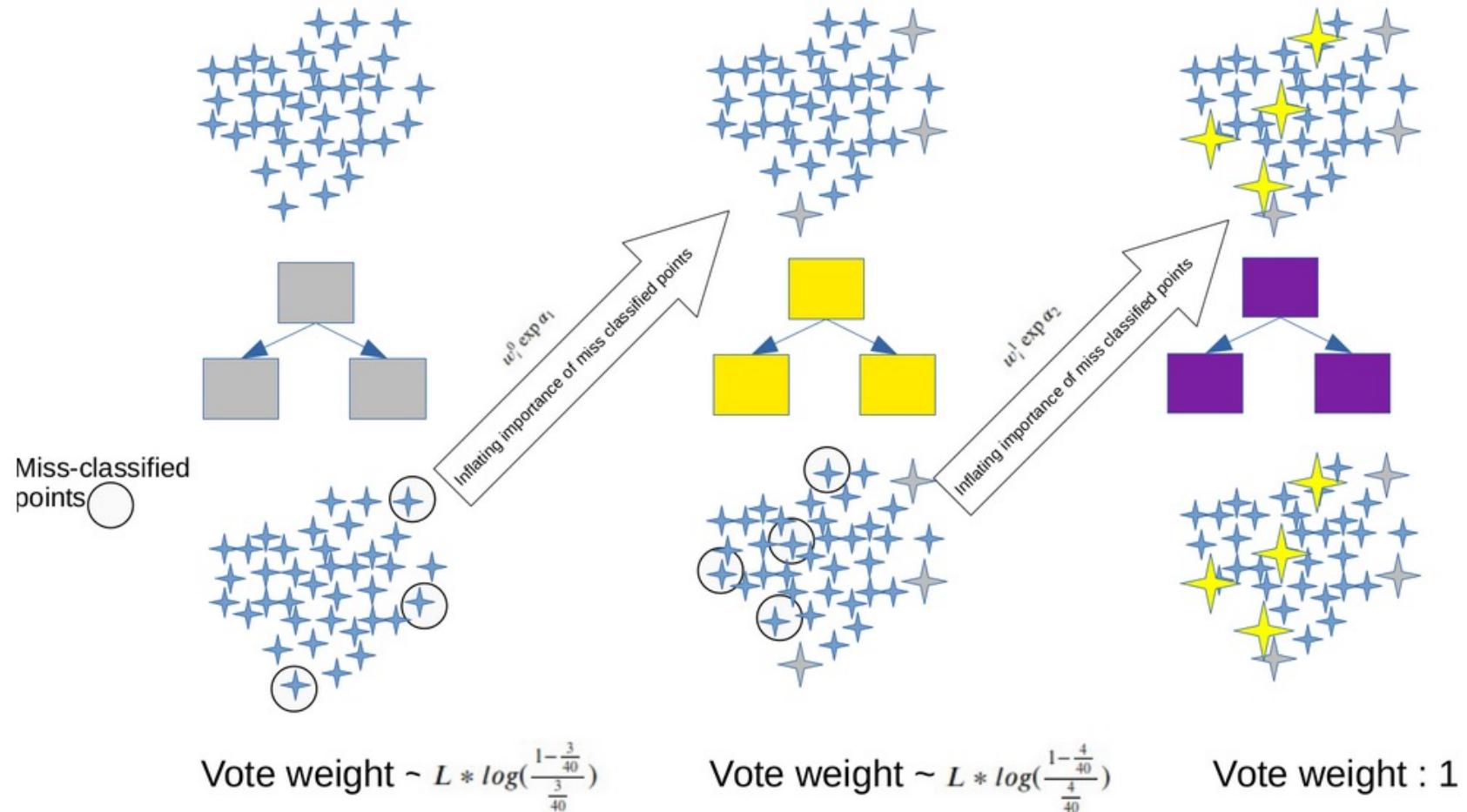
- Misclassification rate is calculated for each model:

$$\alpha = L * \ln\left(\frac{1 - error}{error}\right) \text{ with } L = \text{learning rate and } error = \frac{\sum(w(i) * \text{terror}(i))}{\sum(w(i))}$$

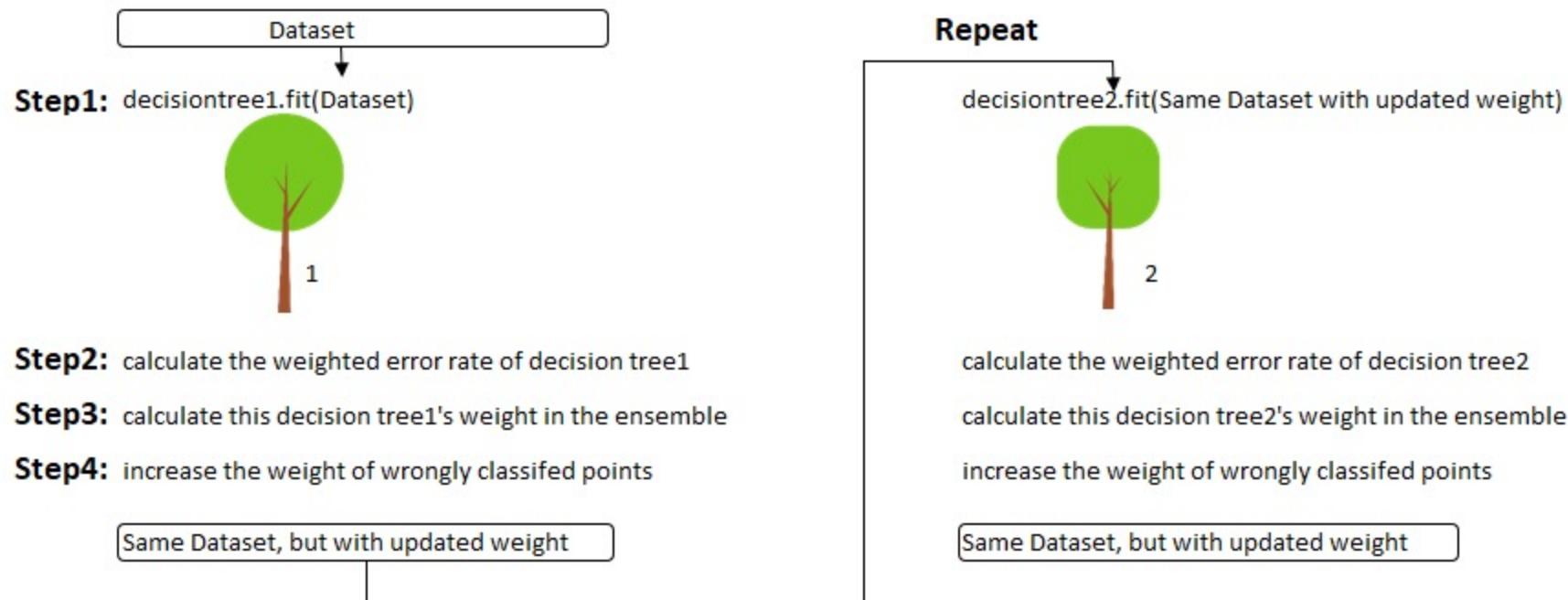
$w_i^{t+1} = w_i^t \exp \alpha_{t+1}$  if  $\text{stump}_{t+1}$  made a mistake for point i

- Predictions are made by calculating the weighted average of the weak classifiers.
- Hyperparameters:
  - Number of stumps
  - Learning rate L: weight applied to each classifier at each boosting iteration
- Need high quality data !

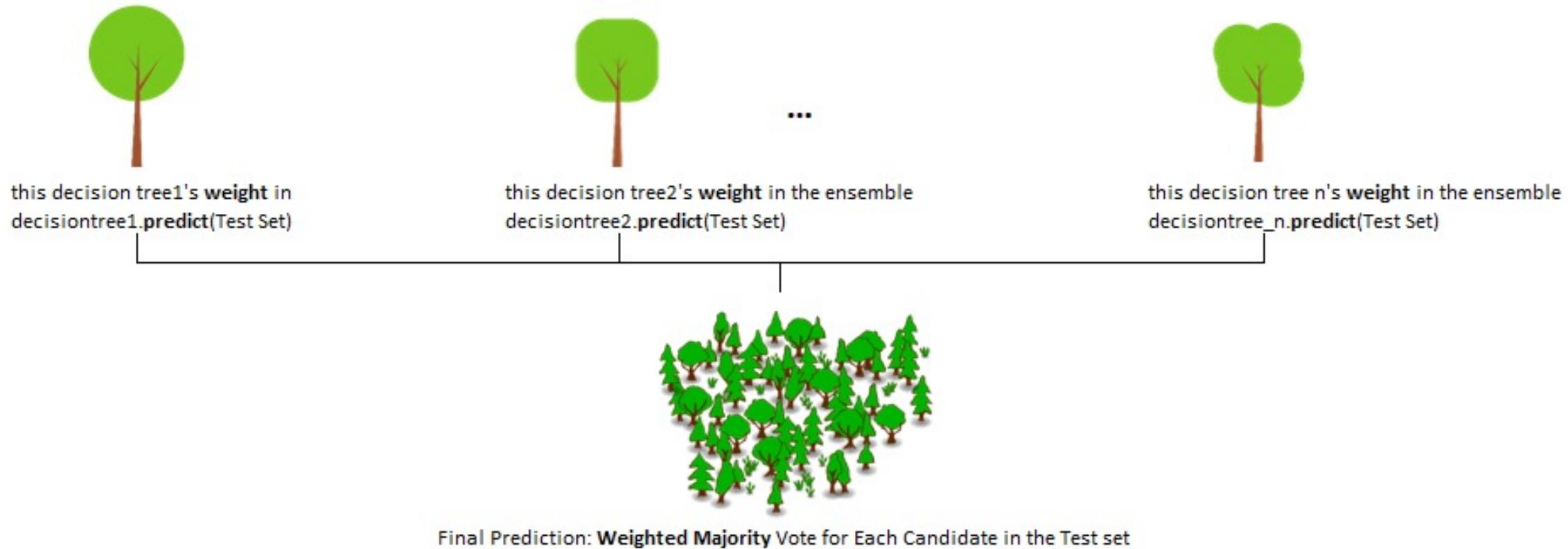
# Adaptive Boosting (AdaBoost)



# Adaptive Boosting (AdaBoost)



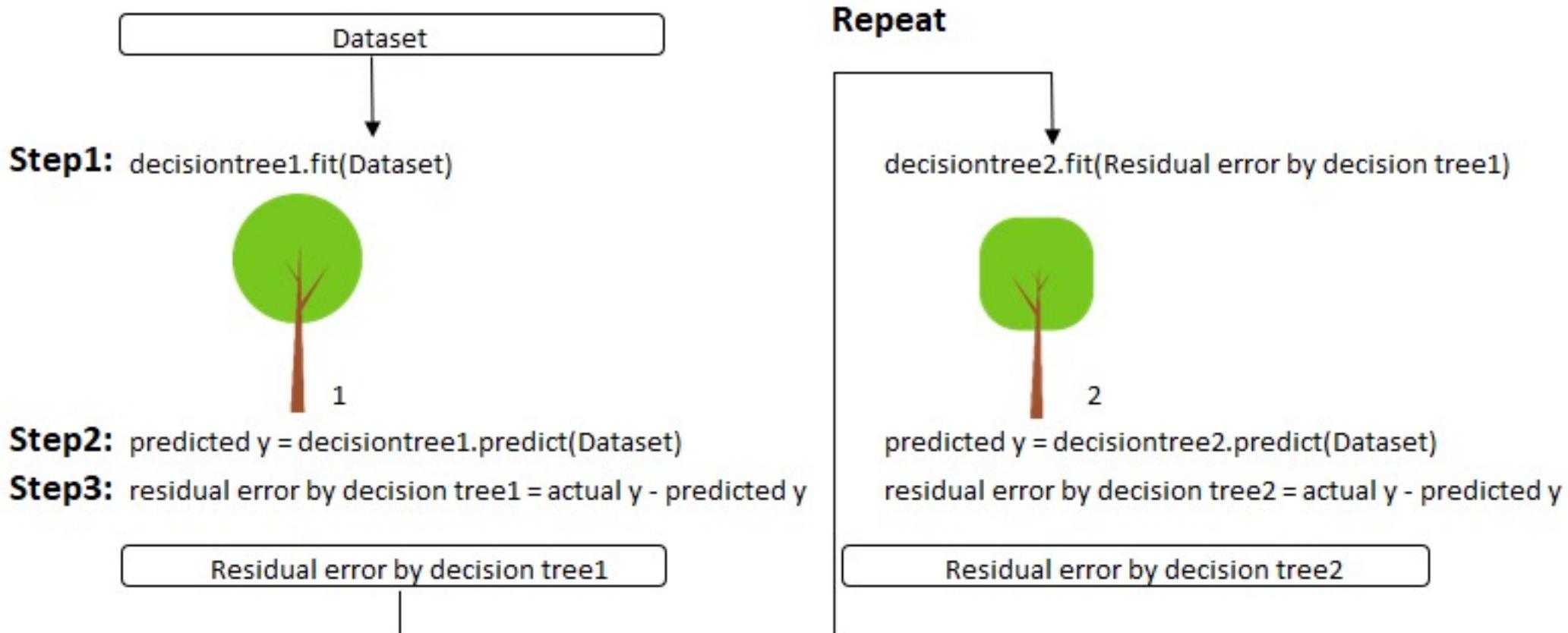
# Adaptive Boosting (AdaBoost)



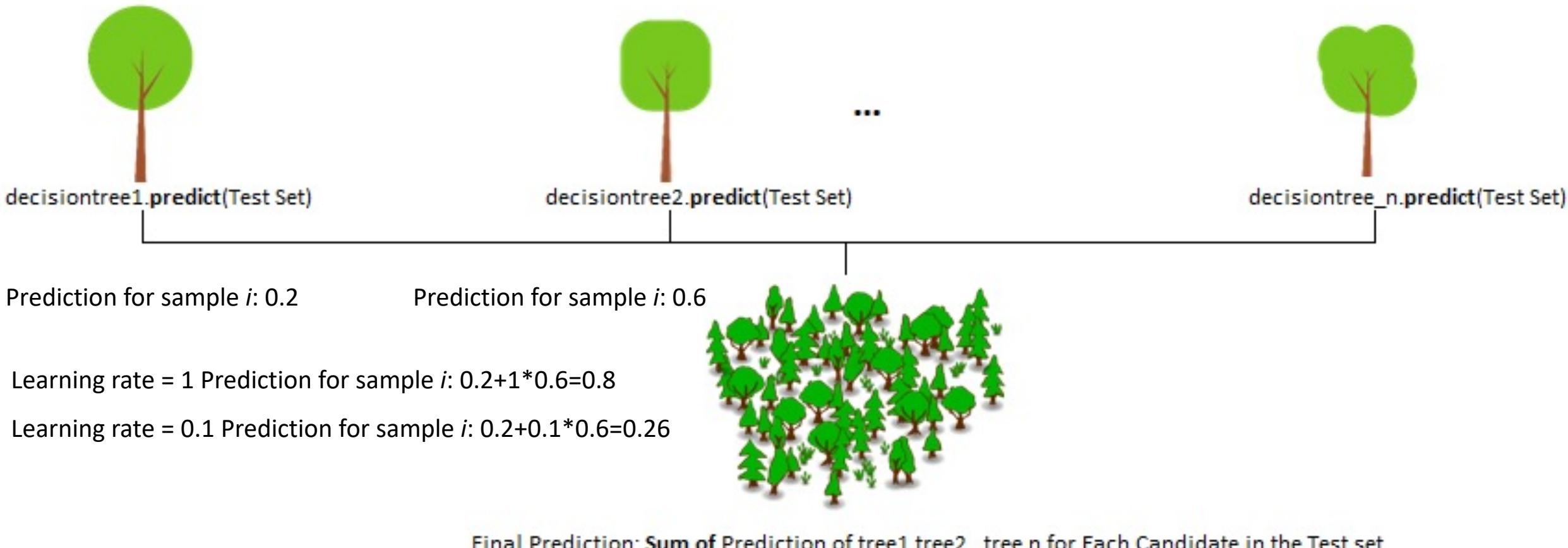
# Gradient Boosting

- Another boosting model. Remember, boosting model's key is learning from the previous mistakes.
- Can be used for both classification and regression problem
- Gradient boosting involves three elements:
  1. A loss function to be optimized.
  2. A weak learner to make predictions: decision trees (with 4-to-8 levels)
  3. An additive model to add weak learners to minimize the loss function:
    - Trees are added one at a time, and existing trees in the model are not changed.
    - A gradient descent procedure is used to minimize the loss when adding trees.

# Gradient Boosting



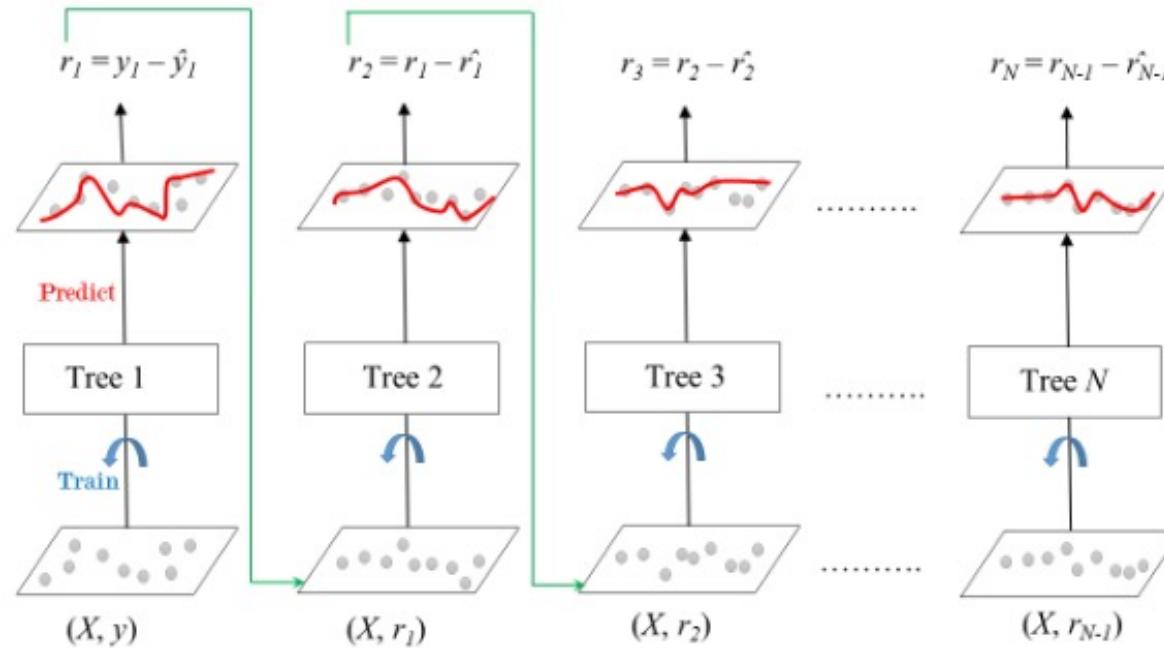
# Gradient Boosting



# Gradient Boosting

- The intuition behind gradient boosting algorithm is to repetitively leverage the patterns in residuals and strengthen a model with weak predictions and make it better
- Once we reach a stage that residuals do not have any pattern that could be modeled, we can stop modeling residuals (otherwise it might lead to overfitting). Algorithmically, we are minimizing our loss function, such that test loss reach its minima.

# Gradient Boosting



- The ensemble consists of  $N$  trees. Tree 1 is trained using the feature matrix  $X$  and the labels  $y$ . The predictions labelled  $y_1(\hat{y})$  are used to determine the training set residual errors  $r_1$ . Tree 2 is then trained using the feature matrix  $X$  and the residual errors  $r_1$  of Tree 1 as labels. The predicted results  $r_1(\hat{r})$  are then used to determine the residual  $r_2$ . The process is repeated until all the  $N$  trees forming the ensemble are trained.

# Gradient Boosting

- Gradient boosting is a greedy algorithm and can overfit a training dataset quickly.
- It can benefit from regularization methods that penalize various parts of the algorithm and generally improve the performance of the algorithm by reducing overfitting.
- Enhancements to basic gradient boosting:
  - Tree Constraints (small number of trees, short trees)
  - Shrinkage (learning rate)
  - Random sampling (bootstrap the training dataset)
- Hyperparameters:
  - Loss function: deviance (= in logistic regression) or exponential (= in AdaBoost)
  - Learning rate: shrinks the contribution of each tree
  - Number of trees, trees length, minimum number of samples required to split an internal node, minimum number of samples required to be at a leaf node
  - Number of features to consider when looking for the best split, etc.

# Naive Bayes

$$P(A|B) = \frac{P(B|A) \cdot P(A)}{P(B)}$$

- classification technique based on Bayes' Theorem with an assumption of independence among predictors (= features)
- a classification problem may have  $k$  class labels  $y_1, y_2, \dots, y_k$  and  $n$  input variables,  $X_1, X_2, \dots, X_n$ . We can calculate the conditional probability for a class label with a given instance or set of input values for each column  $x_1, x_2, \dots, x_n$  as follows:  $P(y_i | x_1, x_2, \dots, x_n)$
- The conditional probability can then be calculated for each class label in the problem and the label with the highest probability can be returned as the most likely classification.

$$P(y_i | x_1, x_2, \dots, x_n) = P(x_1, x_2, \dots, x_n | y_i) * P(y_i) / P(x_1, x_2, \dots, x_n)$$

Independence between features

$$P(x_1 | y_i) * P(x_2 | y_i) * \dots * P(x_n | y_i)$$

estimated from the data



prior  $P(y_i)$   
easy to  
estimate  
from the  
data



constant used in calculating  
the conditional probability of  
each class for a given instance



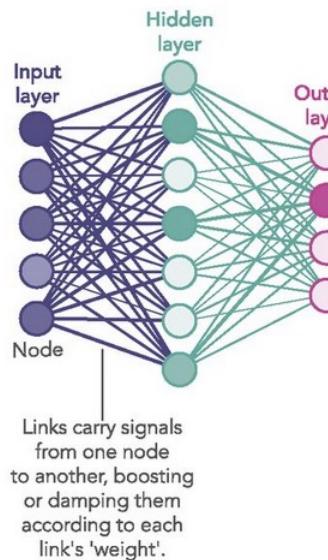
# Naive Bayes

$$P(A|B) = \frac{P(B|A) P(A)}{P(B)}$$

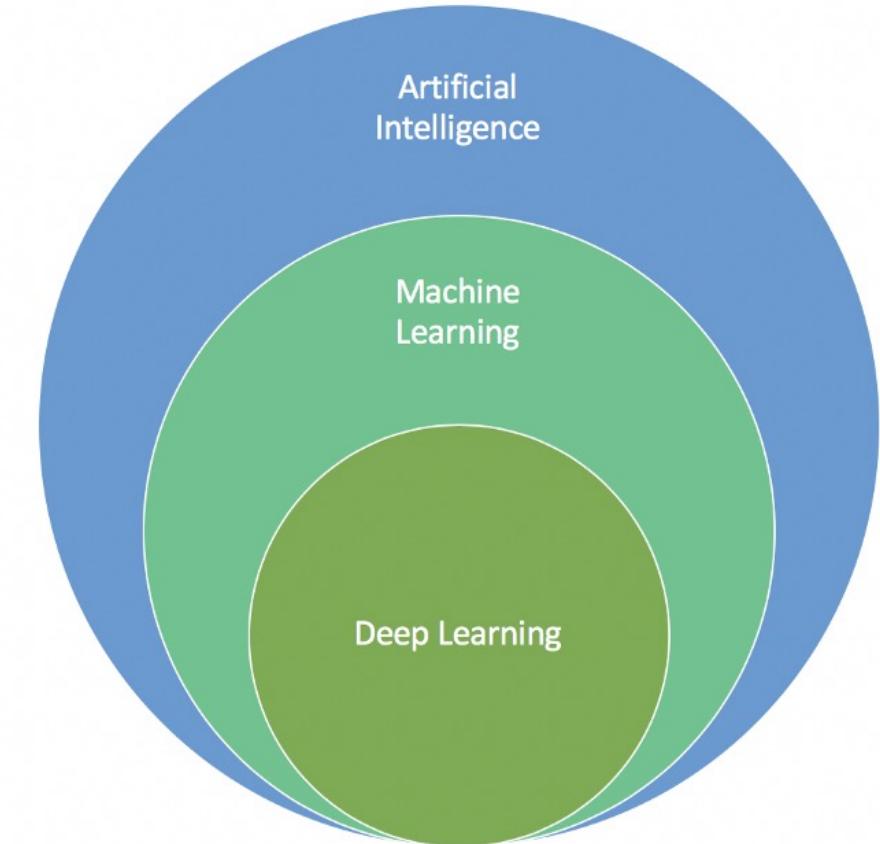
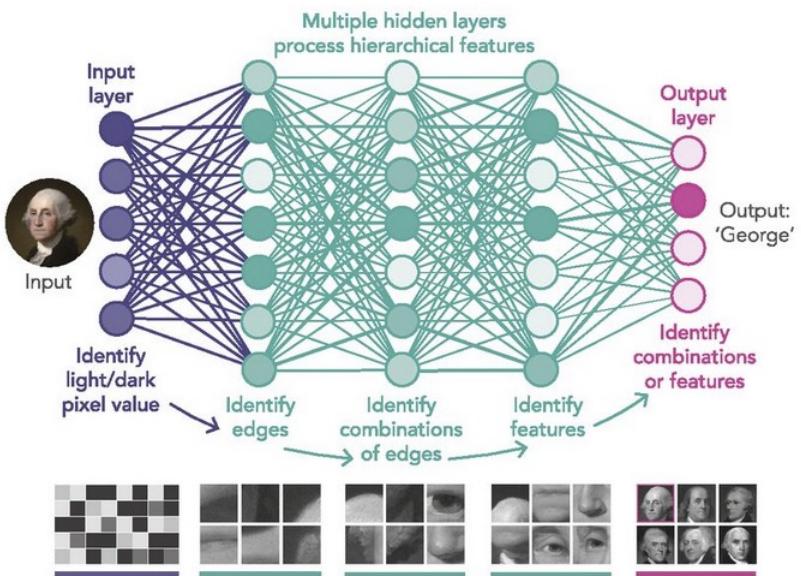
- Advantages:
  - very easy to build and particularly useful for very large data sets.
  - outperform even highly sophisticated classification methods.
  - a good choice when CPU and memory resources are a limiting factor.
  - a good method if something fast and easy that performs pretty well is needed.
- Main disadvantage: does not consider the interactions between features.
- Naive Bayes has been used in real-world applications such as:
  - mining housekeeping genes
  - genetic association studies
  - discovering Alzheimer genetic biomarkers from whole genome sequencing (WGS) data

# Deep learning

1980S-ERA NEURAL NETWORK



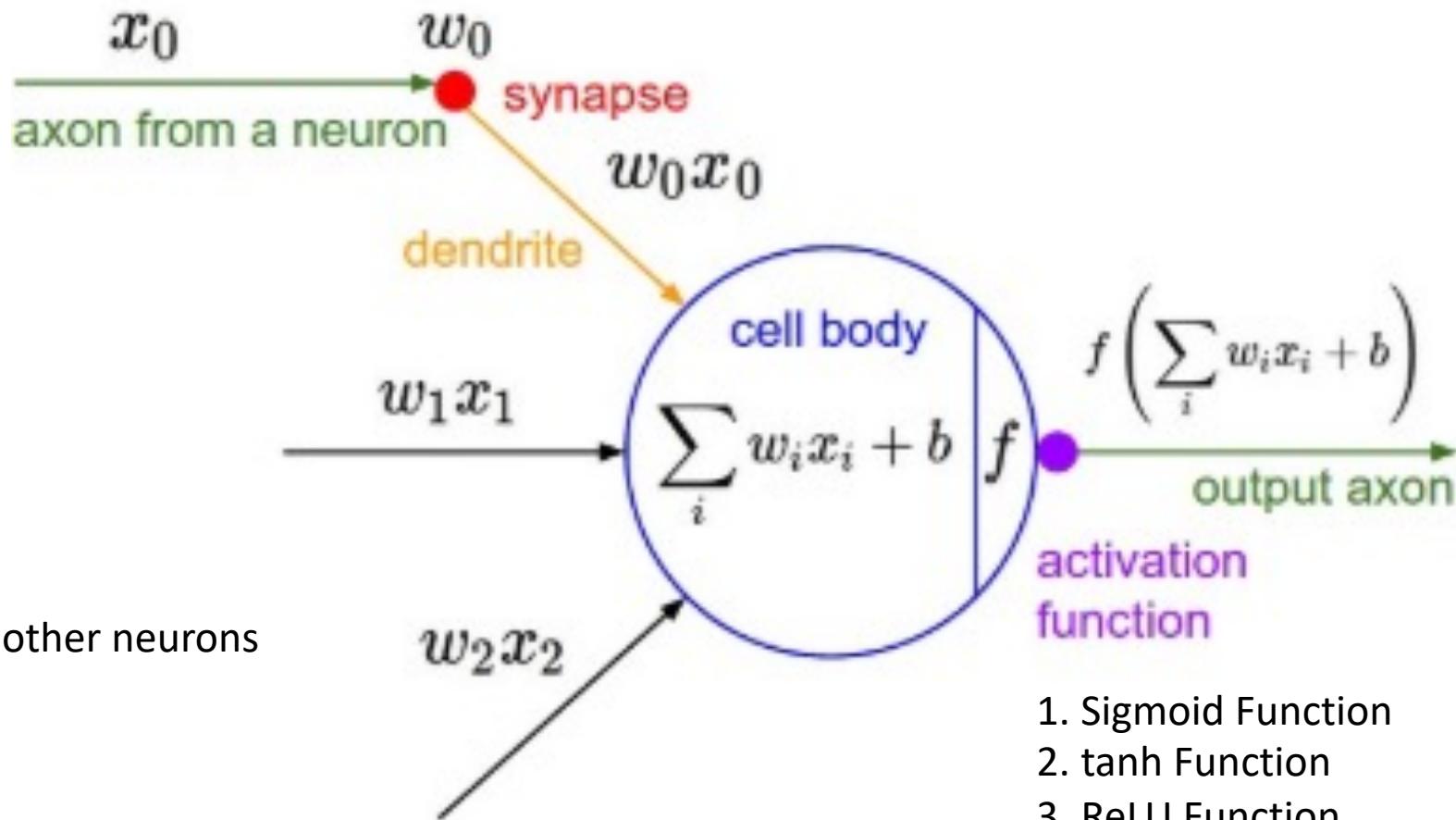
DEEP LEARNING NEURAL NETWORK



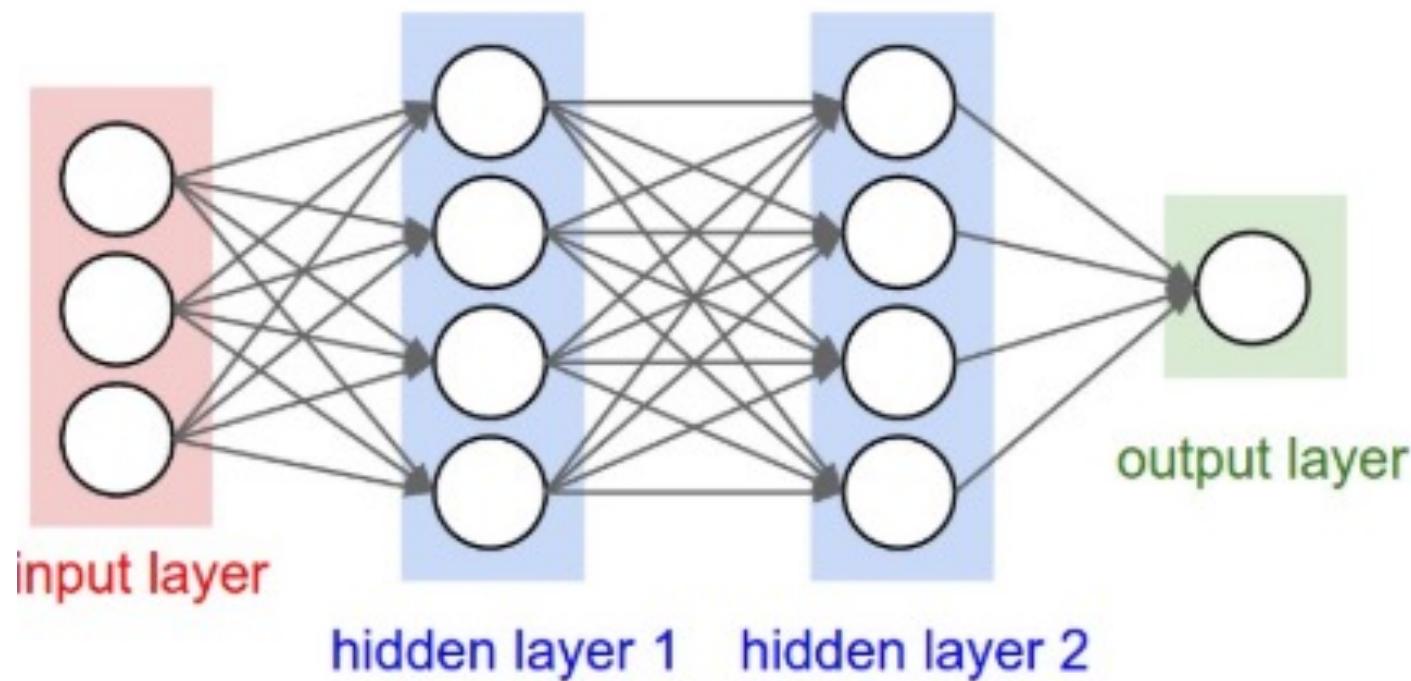
# Deep learning

- Machine Learning consists of two steps:
  1. specify a template
  2. find the best parameters for that template.
- Deep Learning is simply a subset of the architectures (or templates) that employs “neural networks” which we can specify during Step 1.
- “Neural networks” (more specifically, artificial neural networks) are loosely based on how our human brain works, and the basic unit of a neural network is a neuron.
- At the basic level, a neuron does two things:
  1. Receive input from other neurons and combine them together
  2. Perform some kind of transformation to give the neuron’s output

# Neural networks



# Neural networks



A neural network is simply a complicated ‘template’ we specify which turns out to have the flexibility to model many complicated relationships between input and output.

# Deep learning

- Model performance assessed with a loss function
- Minimizing the loss function using gradient descent (or other strategies)
- How to choose the best architecture of a neural network (hyperparameters: number of hidden layers, number of neurons in each hidden layer, type of activation functions etc.)
  1. Specify some hyper-parameters (the template)
  2. Train on the training dataset (filling in the parameters)
  3. Record the validation loss
  4. Repeat Steps 1 to 3 with a different set of hyper-parameters (many times)
  5. Pick the model with the lowest validation loss as our best model
  6. Run the chosen model on a separate test dataset to assess performance

# Deep learning

## Deep Learning in Omics Data Analysis and Precision Medicine

Jordi Martorell-Marugán <sup>1</sup>, Siham Tabik <sup>2</sup>, Yassir Benhammou <sup>2</sup>, Coral del Val <sup>2</sup>, Igor Zvir <sup>2</sup>, Francisco Herrera <sup>2</sup>, Pedro Carmona-Sáez <sup>1</sup>  
Holger Husi <sup>1</sup>, editors.

In: Computational Biology [Internet]. Brisbane (AU): Codon Publications; 2019 Nov 21. Chapter 3.

Affiliations + expand

PMID: 31815397 Bookshelf ID: NBK550335 DOI: 10.15586/computationalbiology.2019.ch3

[Free Books & Documents](#)

### Excerpt

The rise of omics techniques has resulted in an explosion of molecular data in modern biomedical research. Together with information from medical images and clinical data, the field of omics has driven the implementation of personalized medicine. Biomedical and omics datasets are complex and heterogeneous, and extracting meaningful knowledge from this vast amount of information is by far the most important challenge for bioinformatics and machine learning researchers. In this context, there is an increasing interest in the potential of deep learning (DL) methods to create predictive models and to identify complex patterns from these large datasets. This chapter provides an overview of the main applications of DL methods in biomedical research, with focus on omics data analysis and precision medicine applications. DL algorithms and the most popular architectures are introduced first. This is followed by a review of some of the main applications and problems approached by DL in omics data and medical image analysis. Finally, implementations for improving the diagnosis, treatment, and classification of complex diseases are discussed.

Isabelle Dupanloup, BCF

# Deep learning

› [Sci Rep.](#) 2021 Apr 26;11(1):8992. doi: 10.1038/s41598-021-88172-0.

## Machine learning and deep learning to predict mortality in patients with spontaneous coronary artery dissection

Chayakrit Krittawong <sup>1 2</sup>, Hafeez Ul Hassan Virk <sup>3</sup>, Anirudh Kumar <sup>4</sup>, Mehmet Aydar <sup>5</sup>,  
Zhen Wang <sup>6 7</sup>, Matthew P Stewart <sup>8 9</sup>, Jonathan L Halperin <sup>10</sup>

Affiliations + expand

PMID: 33903608 PMCID: [PMC8076284](#) DOI: 10.1038/s41598-021-88172-0

[Free PMC article](#)

### Abstract

Machine learning (ML) and deep learning (DL) can successfully predict high prevalence events in very large databases (big data), but the value of this methodology for risk prediction in smaller cohorts with uncommon diseases and infrequent events is uncertain. The clinical course of spontaneous coronary artery dissection (SCAD) is variable, and no reliable methods are available to predict mortality. Based on the hypothesis that machine learning (ML) and deep learning (DL) techniques could enhance the identification of patients at risk, we applied a deep neural network to information available in electronic health records (EHR) to predict in-hospital mortality in patients with SCAD. We extracted patient data from the EHR of an extensive urban health system and applied several ML and DL models using candidate clinical variables potentially associated with mortality. We partitioned the data into training and evaluation sets with cross-validation. We estimated model performance based on the area under the receiver-operator characteristics curve (AUC) and balanced accuracy. As sensitivity analyses, we examined

Isabelle Dupanloup, BCF

# Miscellaneous

- **NumPy** stands for Numerical Python
  - It is a Python library used for working with arrays.
  - It also has functions for working in domain of linear algebra, Fourier transform, and matrices.
  - NumPy was created in 2005 by Travis Oliphant. It is an open source project and you can use it freely.
- **Matplotlib** is a comprehensive library for creating static, animated, and interactive visualizations in Python.
- **Seaborn** is a Python data visualization library based on matplotlib. It provides a high-level interface for drawing attractive and informative statistical graphics.
- **Pandas** is a fast, powerful, flexible and easy to use open source data analysis and manipulation tool, built on top of the Python programming language.
- **Scikit-learn** (formerly scikits.learn and also known as sklearn) is a free software machine learning library for the Python programming language.[3] It features various classification, regression and clustering algorithms including support vector machines, random forests, gradient boosting, k-means and DBSCAN, and is designed to interoperate with the Python numerical and scientific libraries NumPy and SciPy.