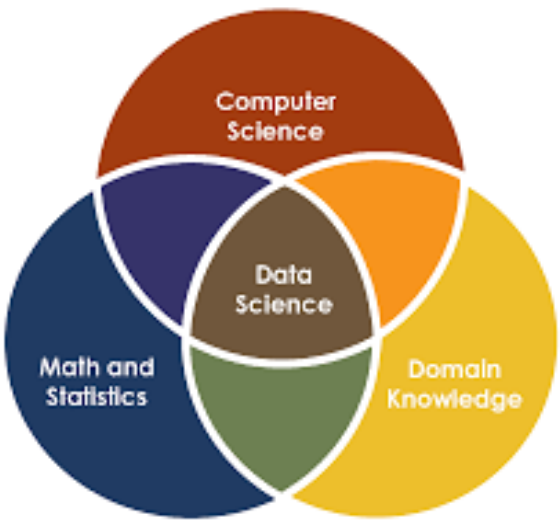
Statistical learning

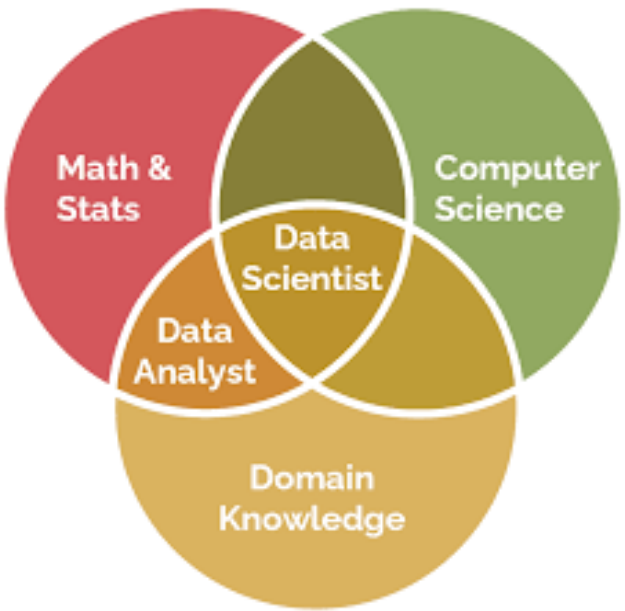
Nicolas Sutton-Charani
Euromov Digital Health in Motion
IMT Mines Alès

Course syllabus

Data science

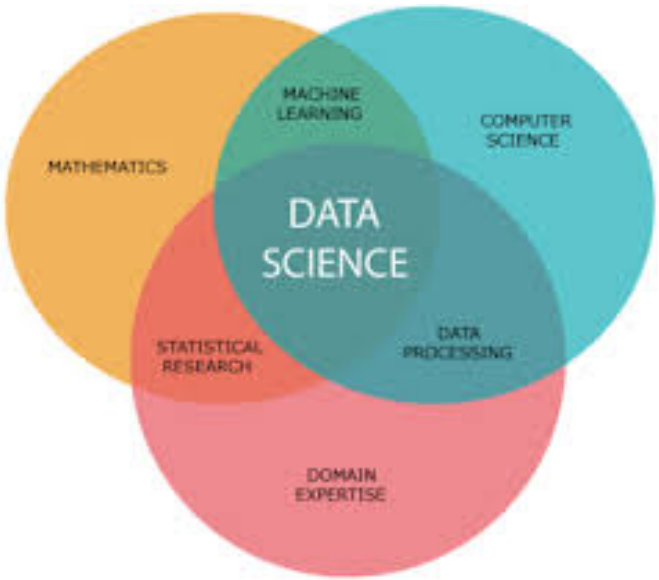
- **mathematics** : statistics, signal processing
- **computer science** : coding, logic, knowledge engineering

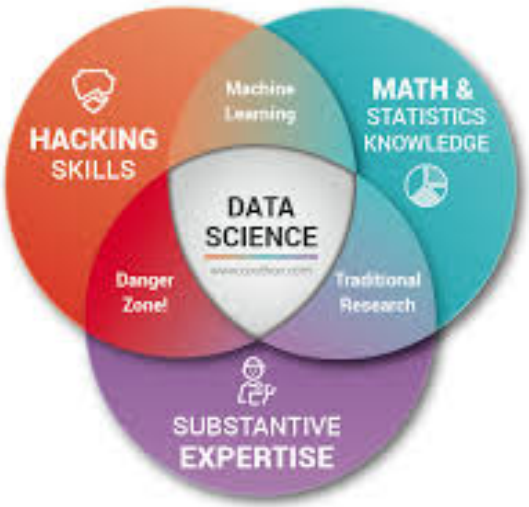




Data science

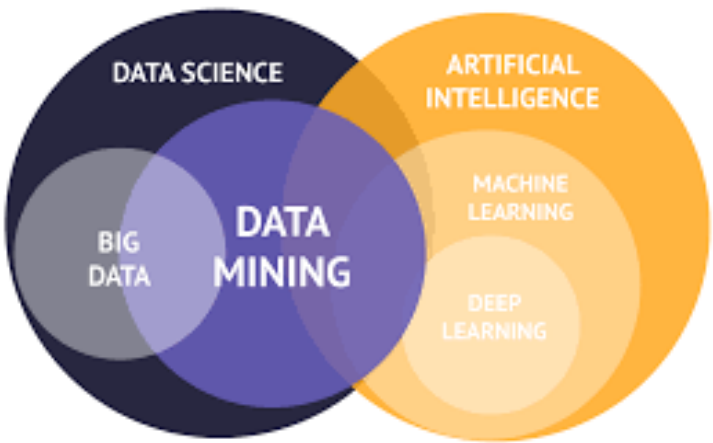
- **mathematics** : statistics, signal processing
- **computer science** : coding, logic, knowledge engineering



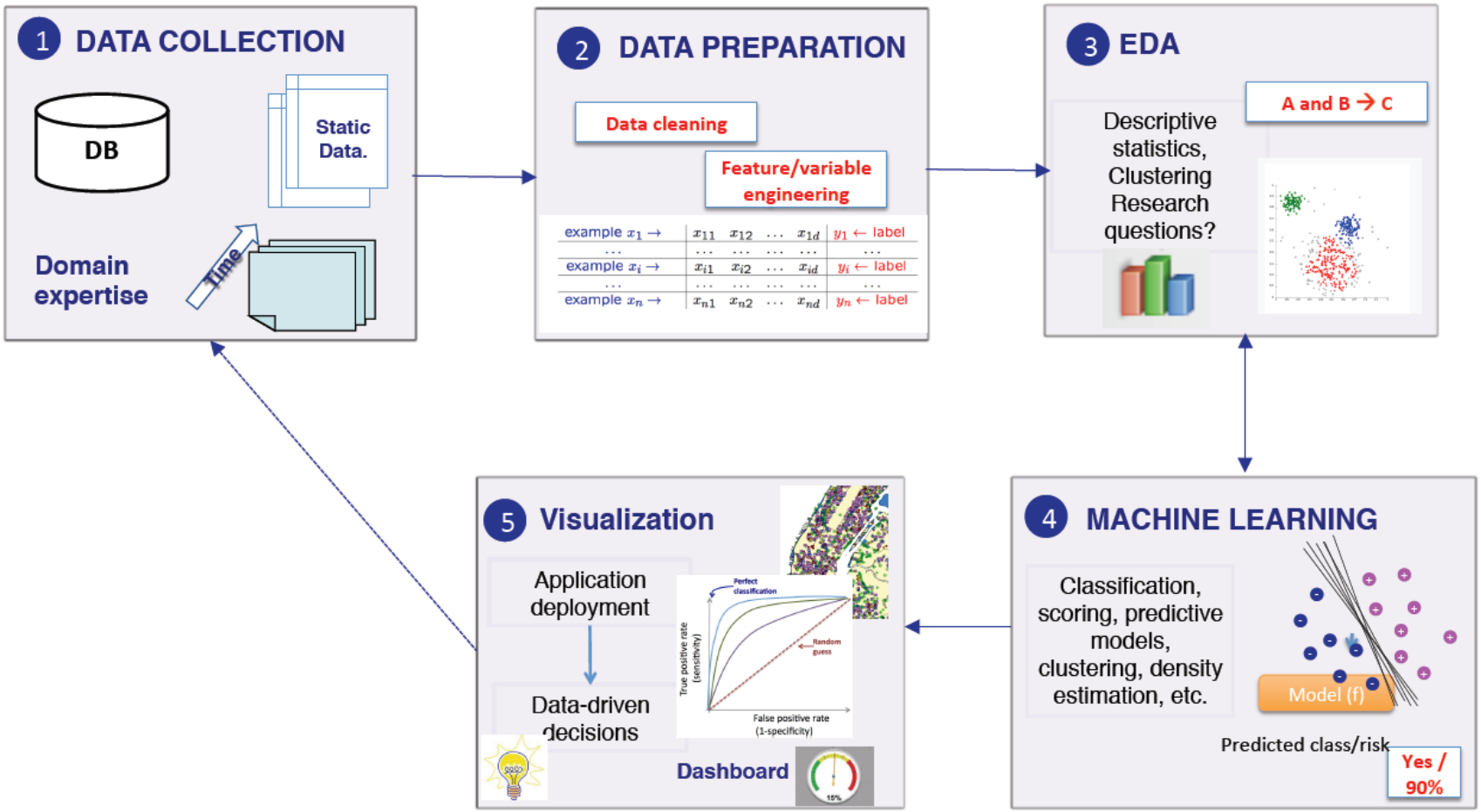


Data science

- **mathematics** : statistics, signal processing
- **computer science** : coding, logic, knowledge engineering



The Data Science process



Course syllabus

Course syllabus

Statistical learning

Definition

Learn pattern or models parameters from a set of examples.

Type of learning

- **Supervised learning** : data are labelled and the objective is to train a model able to predict labels from attributes, classification refers to categorical labels whereas regression involves numerical labels.
- **Unsupervised learning** or clustering : data are unlabelled, the objective is to find relevant clusters that fit to the examples latent structure. This allows the profiling of examples in several clusters that reflects the population main profiles.

Hypothesis

- **Supervised case** : the labels distribution depends on the attributes one.
- **Unsupervised case** : examples are structured according to an unknown pattern or distribution.

Statistical learning

Algorithms

- **clustering :** K-means, gaussian mixtures, hierarchical clustering, spectral clustering, etc.
- **classification :** K-nearest neighbours, logistic regression, Support Vector Machines (SVM), neural networks, decision trees, naive Bayes, random forest, boosting, etc.
- **regression :** K-nearest neighbors, linear regression, lasso, ridge, elasticnet, Support Vector Machines (SVM), neural networks, decision trees, random forest, boosting

Formalism

Training data

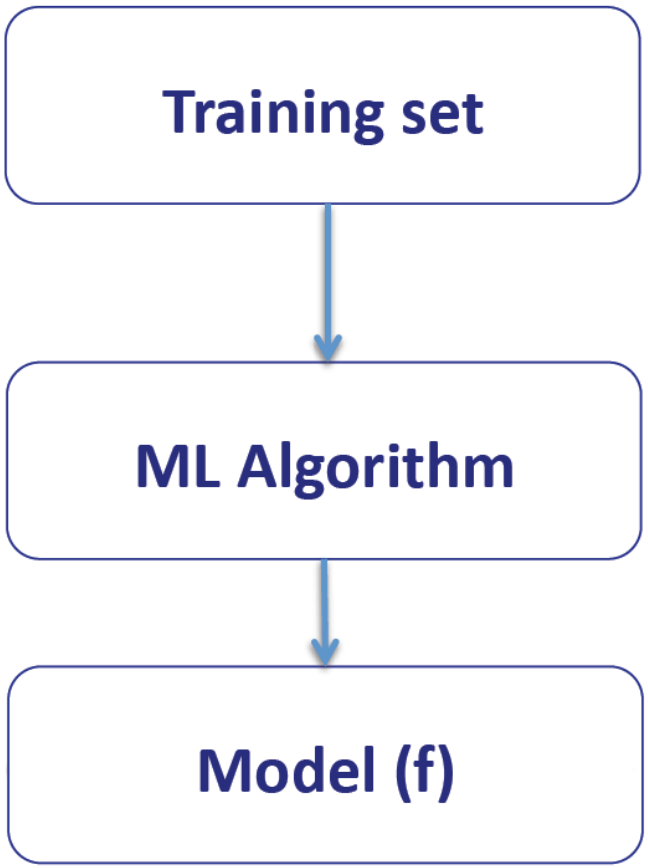
$$(x, y) = \begin{pmatrix} x_1, y_1 \\ \vdots \\ x_N, y_N \end{pmatrix} = \begin{pmatrix} x_1^1 & \dots & x_1^J & y_1 \\ \vdots & & \vdots & \vdots \\ x_n^1 & \dots & x_n^J & y_n \end{pmatrix}$$

examples (x_i, y_i) are assumed to be *i.i.d.*

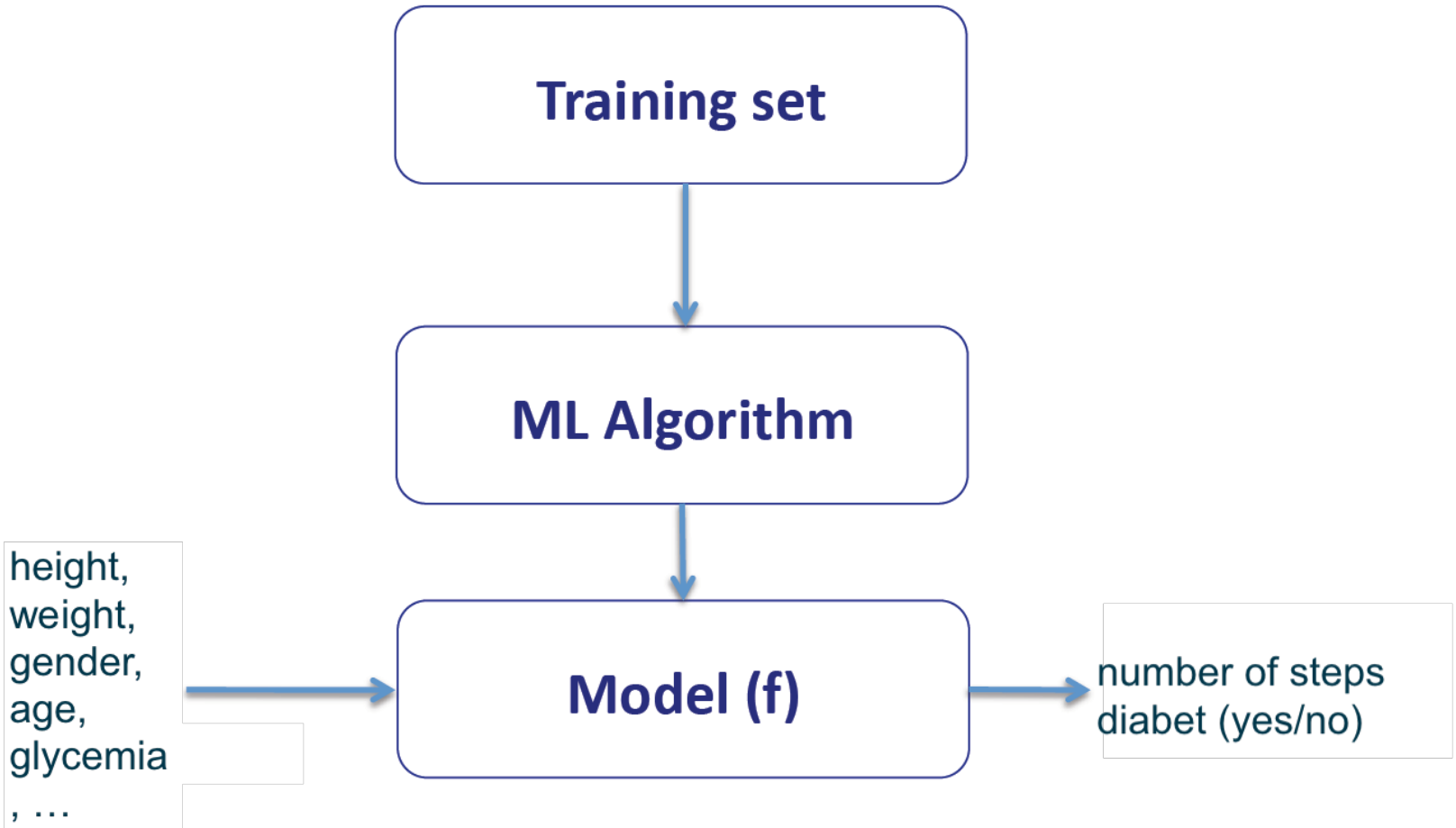
- Attributes $X = (X^1, \dots, X^J) \in \mathcal{X} = \mathcal{X}^1 \times \dots \times \mathcal{X}^J$
- Spaces $\mathcal{X}^j$ can be categorical or numerical.
- Class label $Y \in \Omega = \{\omega_1, \dots, \omega_K\}$ ($\subset \mathbb{R}$ for regression)
- A supervised model $f_\theta : \mathcal{X} \to \Omega$ aims at mapping attributes $(xi)_{i=1,\dots,n}$ to labels $(y_i)_{i=1,\dots,n}$ such that $\forall i = 1, \dots, n : f_\theta(x_i) \approx y_i$. The model's parameter $\theta \in \Theta$ is estimated by fitting it to training data (x, y).

Course syllabus

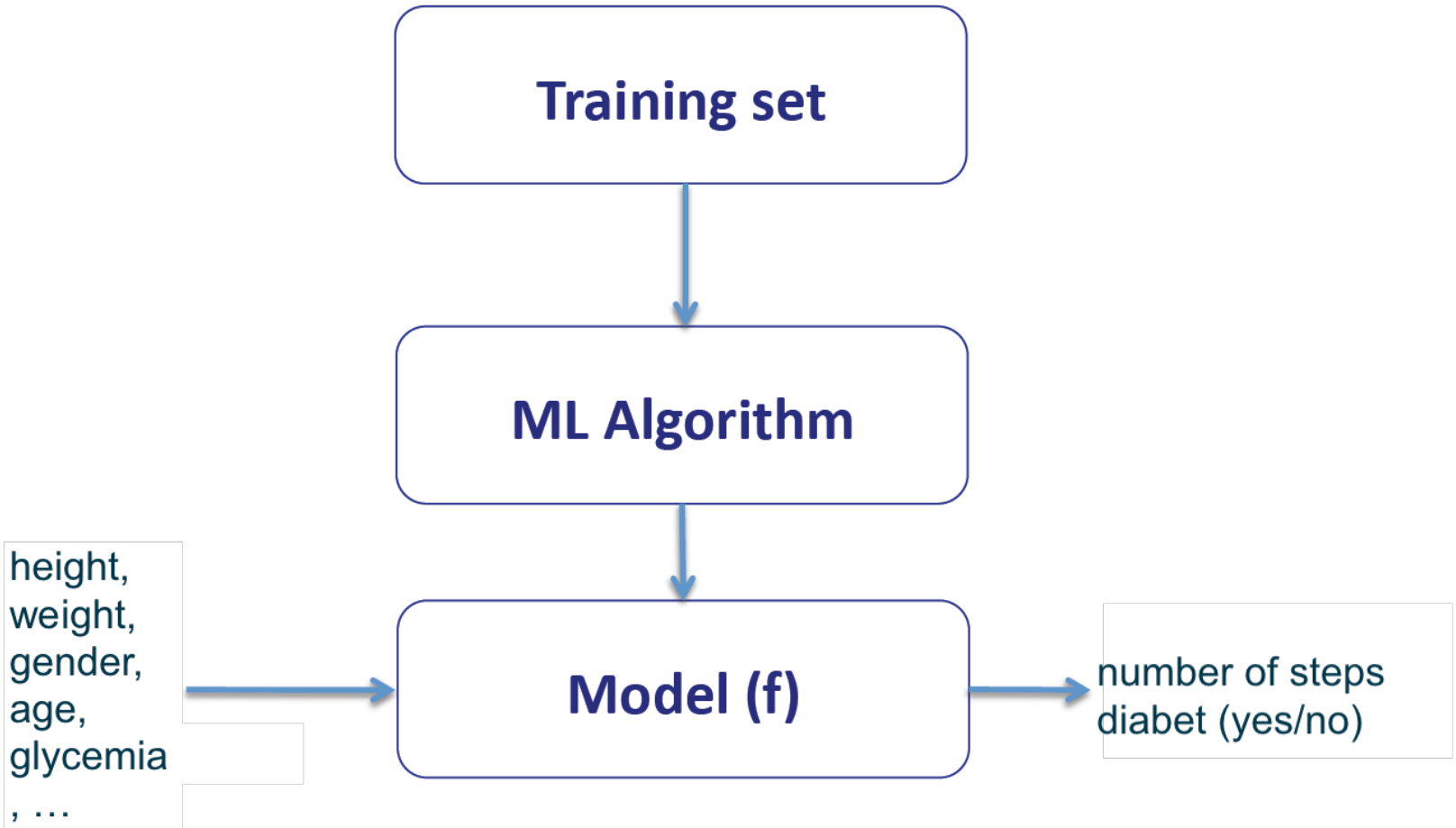
Training and Testing



Training and Testing



Training and Testing



Question : How can we be confident about f ?

Clustering

How to evaluate your model ?

- Not trivial (as compared to counting the number of errors in classification).
- **Internal evaluation :** using same data. high intra-cluster similarity and low inter-cluster similarity.
- **External evaluation :** use of ground truth of external data.

Training and Testing

- Training set is a set of examples used for learning a model (e.g., a classification model).
- Test set is used to assess the performance of the final model and provide an estimation of the test error.

Note : Never use the test set in any way to further tune the parameters or revise the model.

K-fold Cross Validation

A method for estimating test error using training data.

Data: learning algorithm A and a dataset D

Result: test performance estimator $\hat{E}$

Step 1 : Randomly partition D into K equal-size subsets $D_1, ..., D_K$

Step 2 :
for $k = 1$ *to* K **do**
    Train A on all D_l, $l \in \{1, ..., K\}$ and $l \neq k$, and get f_k;
    Apply f_k to D_k and compute $Perf(f_k, D_k)$;
end
Step 3 : Average error over all folds :

$$\hat{E} = \frac{1}{K} Perf(f_k, D_k)$$

Stratified K-fold Cross Validation

A method for estimating test error using training data.

Data: learning algorithm A and a dataset D

Result: test performance estimator $\hat{E}$

Step 1 : Randomly partition D into K equal-size subsets $D_1, ..., D_K$ with similar labels distribution

Step 2 :

for $k = 1$ *to* K **do**

  Train A on all D_l, $l \in \{1, ..., K\}$ and $l \neq k$, and get f_k;

  Apply f_k to D_k and compute $Perf(f_k, D_k)$;

end

Step 3 : Average error over all folds :

$$\hat{E} = \frac{1}{K} Perf(f_k, D_k)$$

Confusion matrix

		Actual label	
		Positive	Negative
Predicted label	Positive	**True Positive** (*TP*)	**False Positive** (*FP*)
	Negative	**False Negative** (*FN*)	**True Negative** (*TN*)

Evaluation metrics

		Actual label	
		Positive	Negative
Predicted label	Positive	**True Positive (*TP*)**	**False Positive (*FP*)**
	Negative	**False Negative (*FN*)**	**True Negative (*TN*)**

Accuracy	$\frac{TP+TN}{TP+TN+FP+FN}$	The proportion of correct predictions
Precision	$\frac{TP}{TP+FP}$	The proportion of positive predictions that were actually positive
Recall	$\frac{TP}{TP+FN}$	The proportion of positive cases that were predicted positive
Specificity	$\frac{TN}{TN+FP}$	The proportion of negative cases that were predicted negative

Evaluation metrics

Once a supervised model f is learnt $\rightarrow$ prediction $\widehat{y}_i = f(x_i)$ of y_i
EVALUATION : comparison between the y_i and $\widehat{y}_i$ terms.

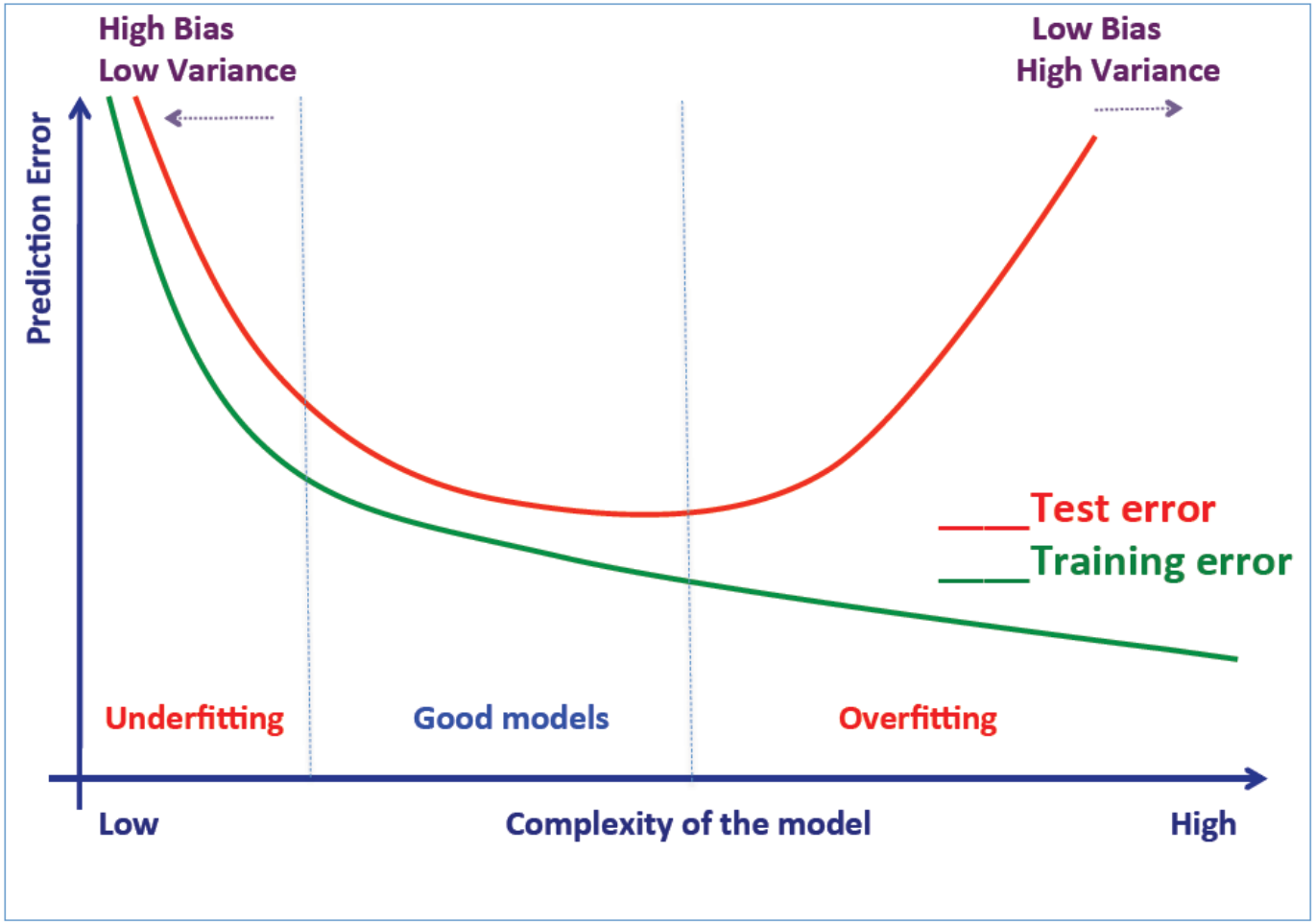
Regression

- **Root Mean Square Error :** $RMSE = \frac{1}{n}\sum_{i=1}^{n}(\widehat{y}_i - y_i)^2$
- **Mean Absolute Error :** $MAE = \frac{1}{n}\sum_{i=1}^{n}|\widehat{y}_i - y_i|$

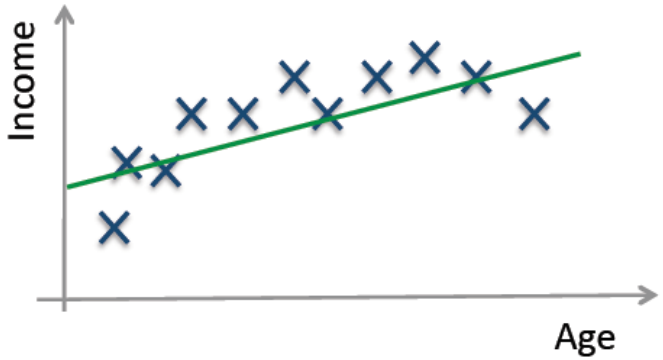
Classification

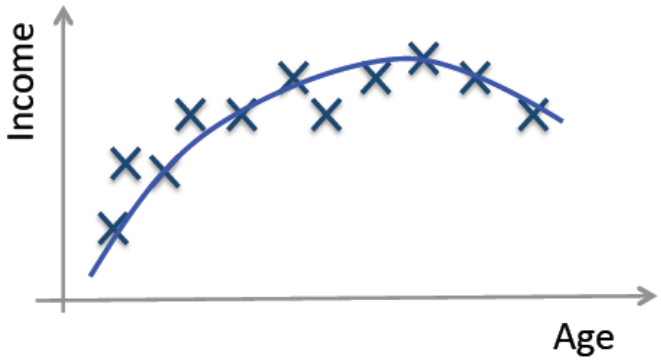
- **accuracy :** $acc = \frac{1}{n}\sum_{i=1}^{n} 1_{\{\widehat{y}_i = y_i\}}$
- **binary case :** precision, recall, F-measure, AUC, etc.

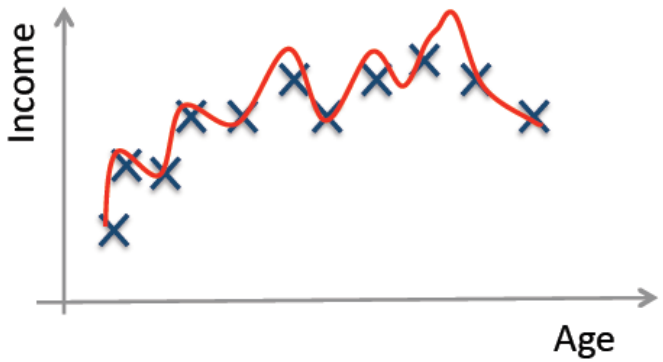
Structural Risk Minimization



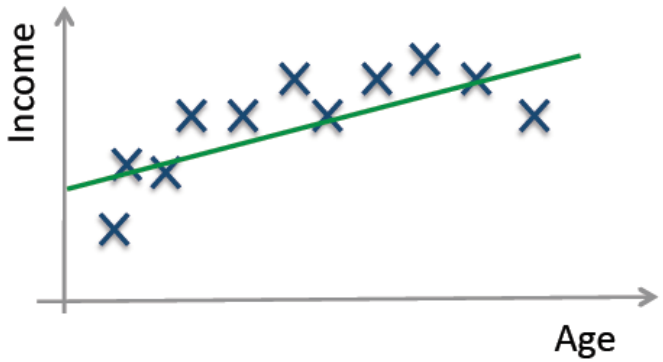
Training and Testing

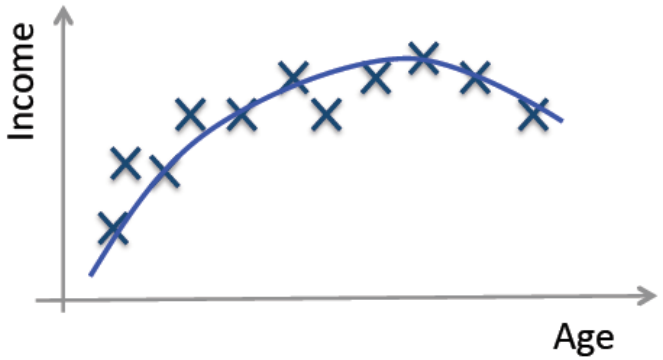




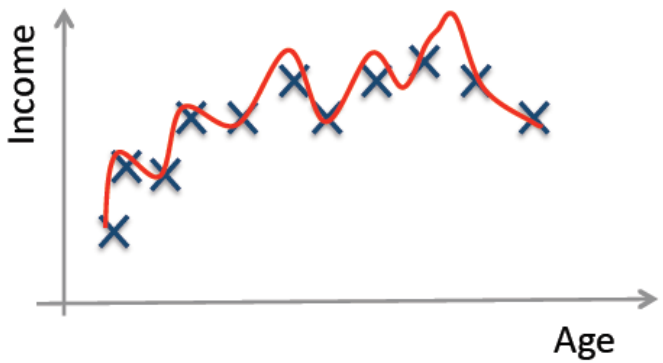


Training and Testing

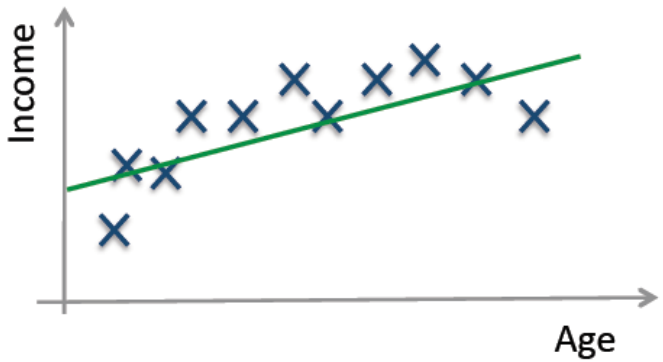


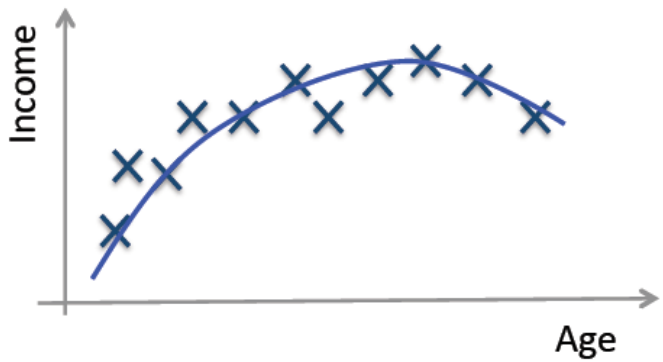


High bias (underfitting)

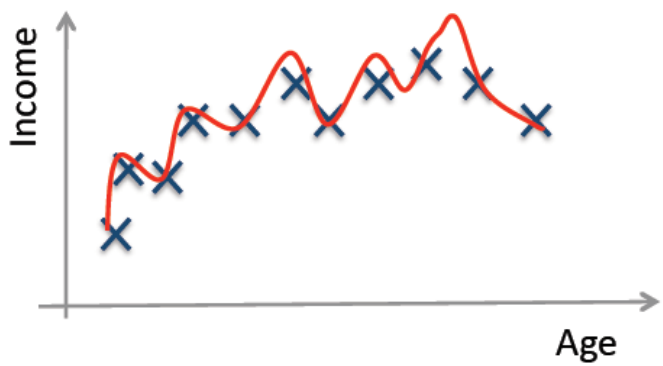


Training and Testing



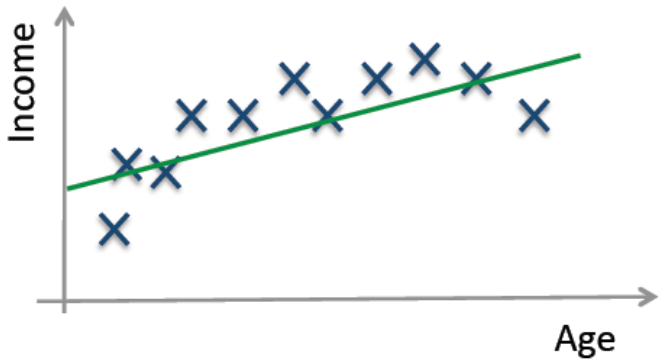


High bias (underfitting)

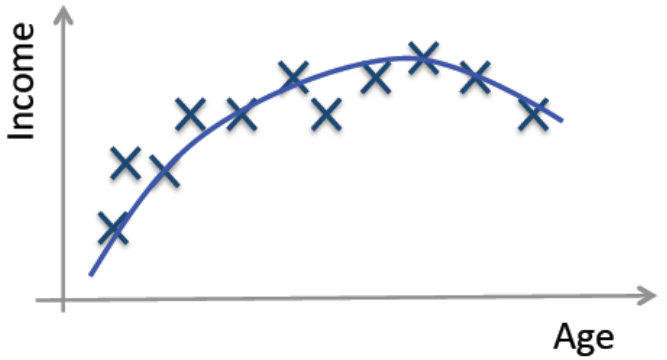


High variance (overfitting)

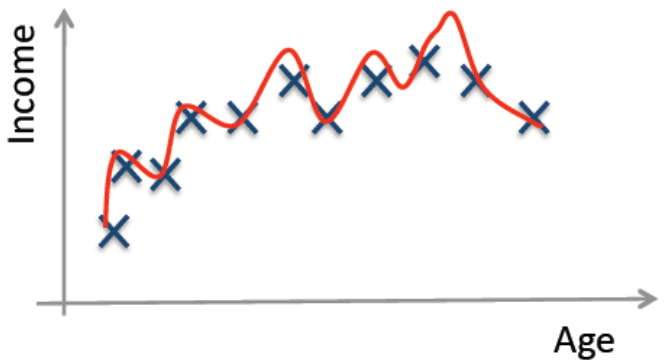
Training and Testing



High bias (underfitting)



Just right!



High variance (overfitting)

Avoid overfitting

In general, use simple models !

- **Reduce the number** of features manually or do feature selection.
- Do a **model selection**.
- Do a **cross-validation** to estimate the test error.

Course syllabus

Course syllabus

Descriptive analysis

- descriptiv statistics
  - variables
    * types
    * spaces
  - distribution
    * values (min, max, quantiles, etc)
    * plots (boxplots, pie, histograms, etc)
- correlations
  - matrix, heatmap
  - tests
  - $\rightarrow$ features selection ?
- information projection $\rightarrow$ *PCA* (or *SVD*)

Principal Component Analysis (*PCA*)

Formalism

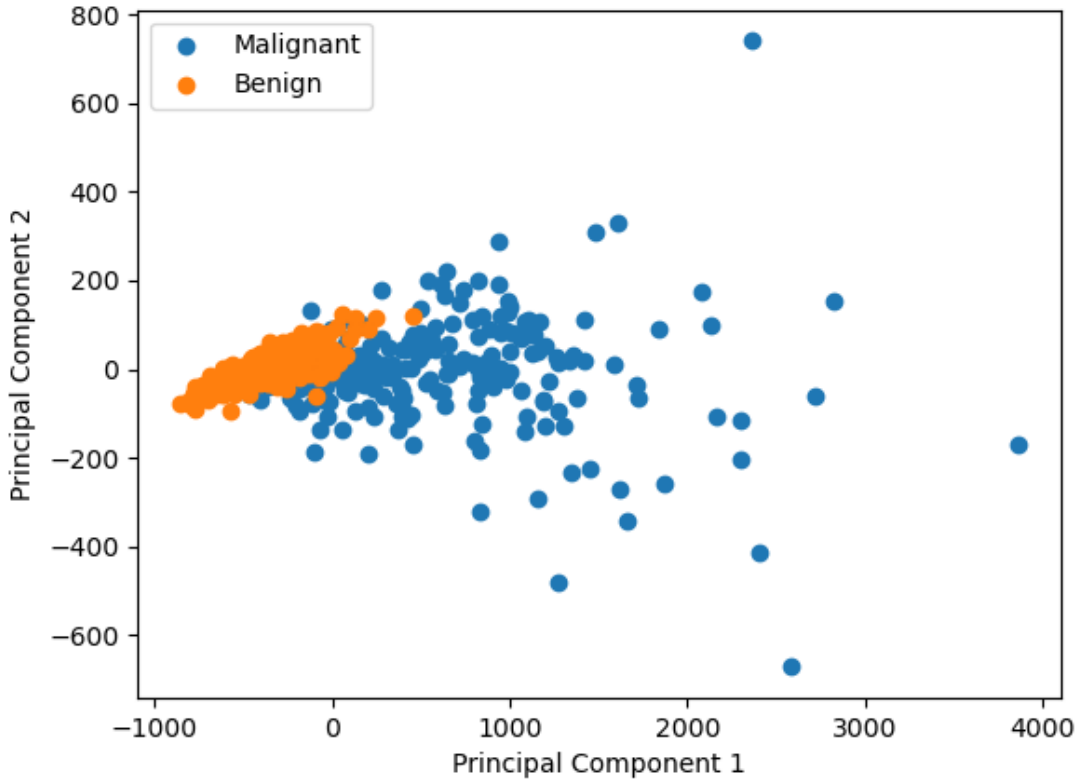
$$x = \begin{pmatrix} x_1 \\ \vdots \\ x_N \end{pmatrix} = \begin{pmatrix} x_1^1, \ldots, x_1^J \\ \vdots \\ x_N^1, \ldots, x_N^J \end{pmatrix}$$

- data are usually centered and reduced

Objectives

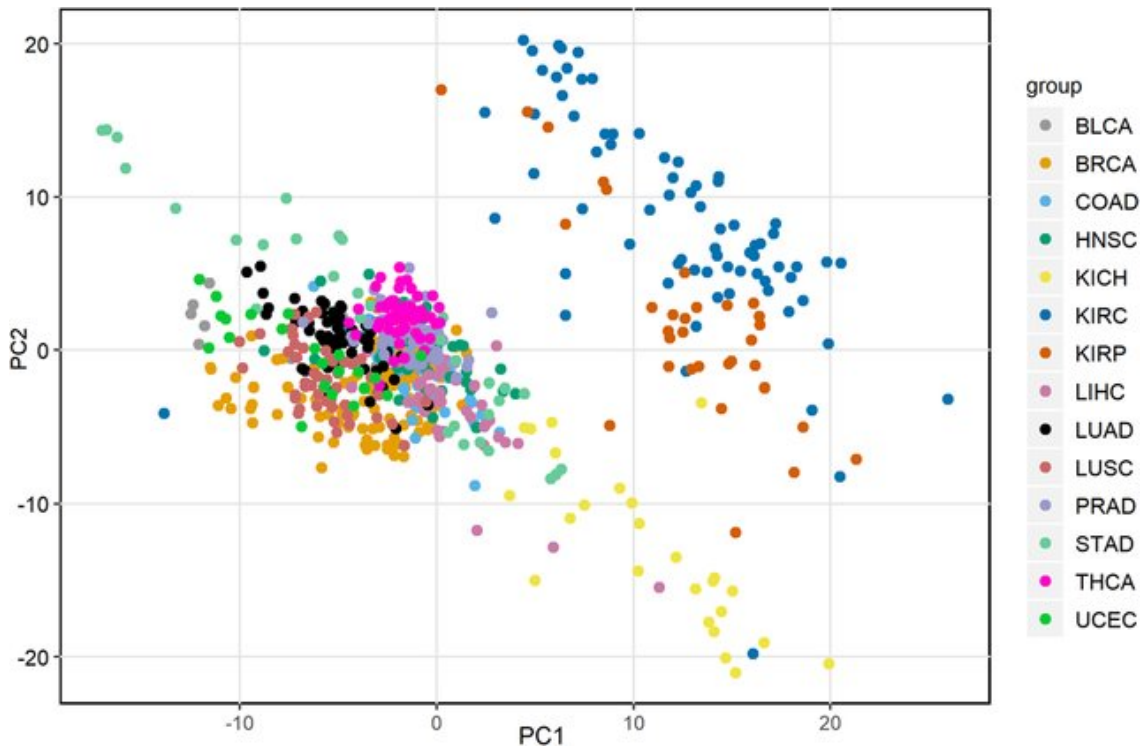
1. the "optimal" graphical representation of the observations minimizing the deformation of the point cloud in a sub-space of dimension 2 (or 3).
2. dimension reduction : $J \to q$ $(q < J)$, or the approximation of observations from J variables with q *new* variables.

Principal Component Analysis (*PCA*)



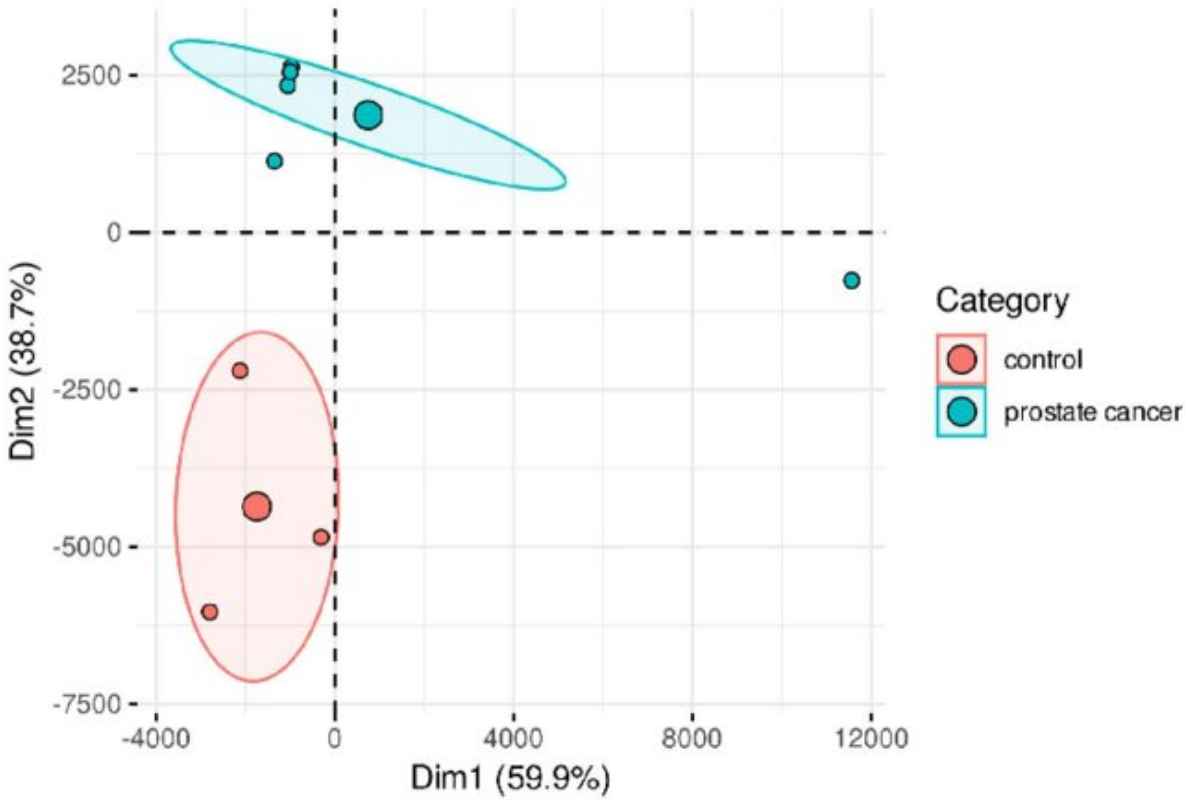
. https ://www.billconnelly.net/ ?p=697

Principal Component Analysis (*PCA*)



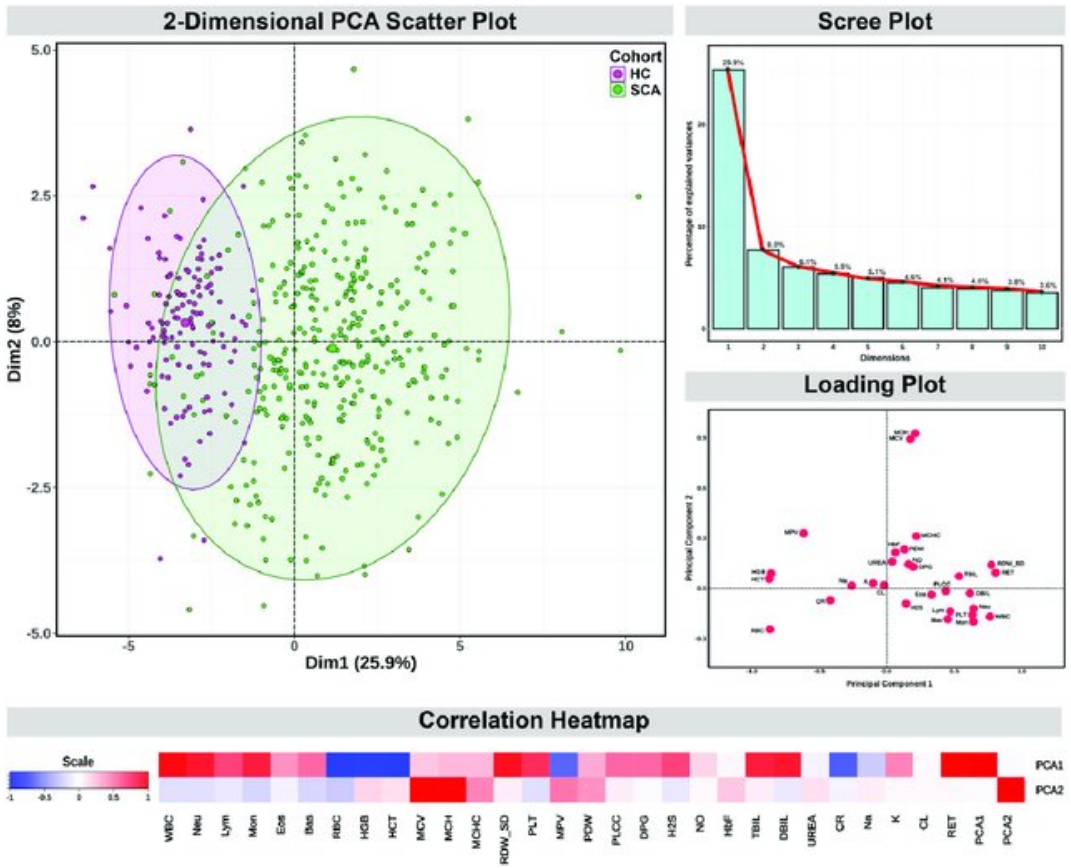
. https ://www.researchgate.net/publication/340459143_Network-based_metabolic_characterization_of_renal_cell_carcinoma/figures ?lo=1

Principal Component Analysis (*PCA*)



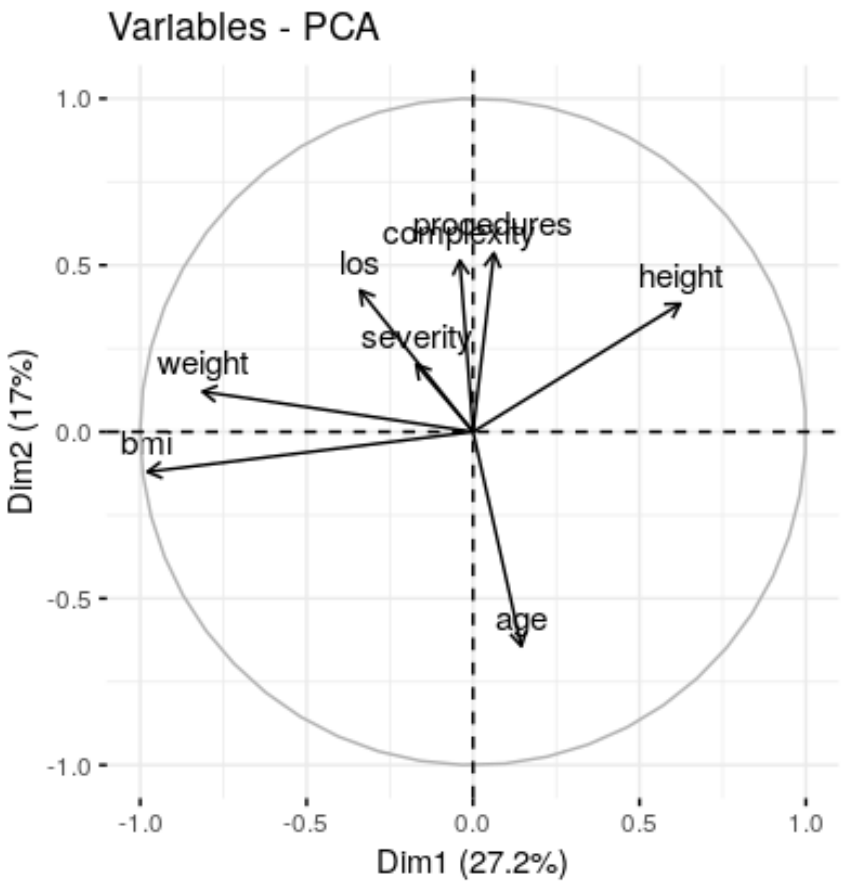
. https ://www.researchgate.net/publication/369278746_Metagenomic_insights_into_the_plasma_virome_of-_Brazilian_patients_with_prostate_cancer/figures

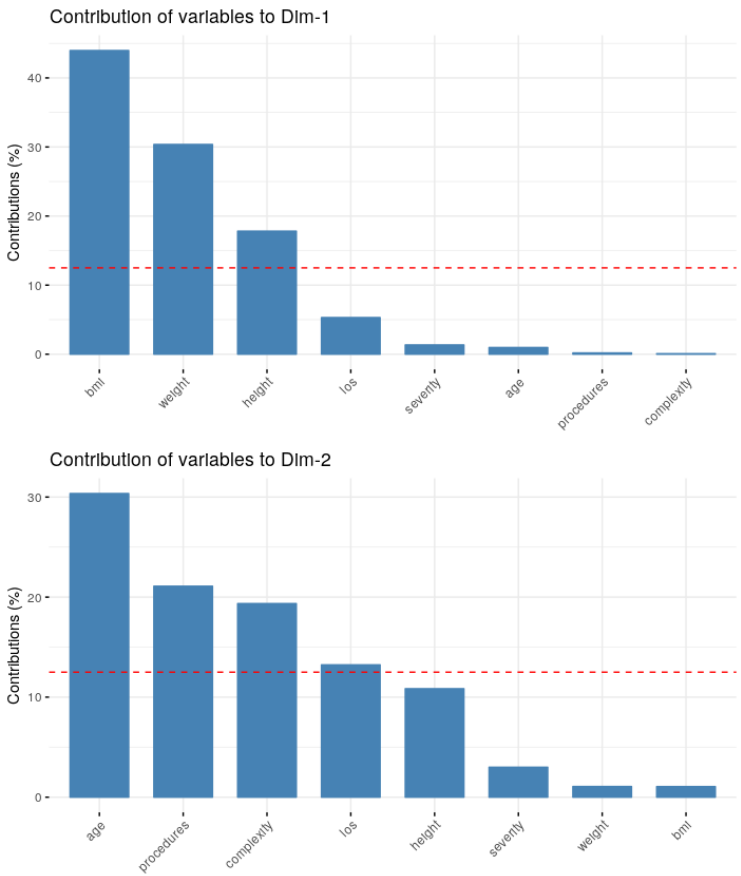
Principal Component Analysis (*PCA*)



. https://www.researchgate.net/publication/392326594_Identification_of_Hematological_Biomarkers_and_Assessment-_of_Machine_Learning_Models_for_Sickle_Cell_Anemia_Severity_Classification/figures?lo=1

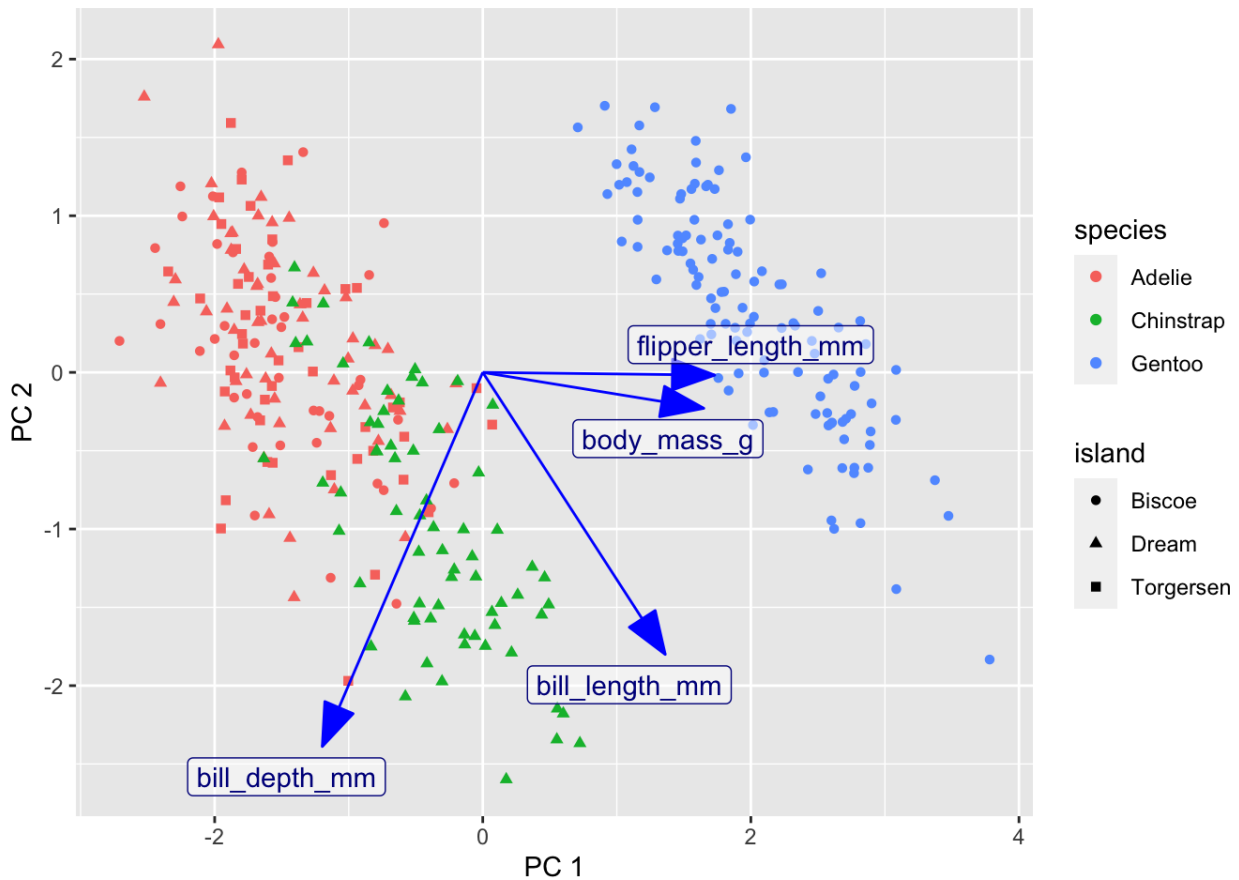
Principal Component Analysis (*PCA*)





. https ://rforhealthcare.org/principal-component-analyses/

Principal Component Analysis (*PCA*)



. https ://andrewirwin.github.io/data-visualization/pca.html

Course syllabus

Dynamical data

Time series $(x_i^j)_{i=1,\dots,N,\ t=1,\dots,T}$

- unsupervised problems : time series clustering
- supervised problems : time series classification or forecasting
- $\rightarrow$ what features to consider ?

Features extraction

- raw data ? $\rightarrow J = T \rightarrow$ deep approaches (CNN)
- framing/windowing $\rightarrow$ descritpors computation :
  - statistical descriptors (μ, σ, quantiles, kurtosis/skewness, etc.)
  - time and frequencies (Fourier) domains

Evaluation pipeline

- time series classification : features extraction $\rightarrow$ standard k-fold CV
- time series forecasting : chronological pipeline $\rightarrow$ predict iteratively the future from the past (no data mixing !)

⚠ For some application a **segmentation** step is required before features extraction $\rightarrow$ determenist or ML (deep) approaches .

Course syllabus

Images

Images $(x_i)_{i=1,\dots,N}$ with $x_i = \begin{pmatrix} x_i^{1,1} & \dots, & x_i^{1,L} \\ \vdots & & \vdots \\ x_i^{K,1} & \dots & x_i^{K,L} \end{pmatrix}$

- supervised problems : image classification
- raw data ? $\rightarrow$ deep approaches (*state of the art*)
- $\rightarrow$ what features to consider ?

Features extraction

- *bag of features* $\rightarrow$ descritpors computation :
  - statistical descriptors (mean, standard deviation, etc.)
  - texture descriptors
  - corners, edges : Histogram of oriented gradients (*HOG*)
  - color/shape-based

⚠For many application a **segmentation** step is required before features extraction $\rightarrow$ determinist or ML (deep) approaches

Course syllabus

Main algorithms

Unsupervised learning

- K-means
- hierarchical clustering

Supervised learning

- distance based : KNN
- tree based : decision tree, random forest, boosting
- Kernel based : SVM
- generalised linear model : linear regression, logistic regression, lasso, ridge, elasticnet
- neural networks : perceptron, MLP, deep networks

Course syllabus

Clustering

Training data : "examples" x.

$$x_1, \ldots, x_n \text{ with } x_i \in \Omega^1 \times \cdots \times \Omega^d$$

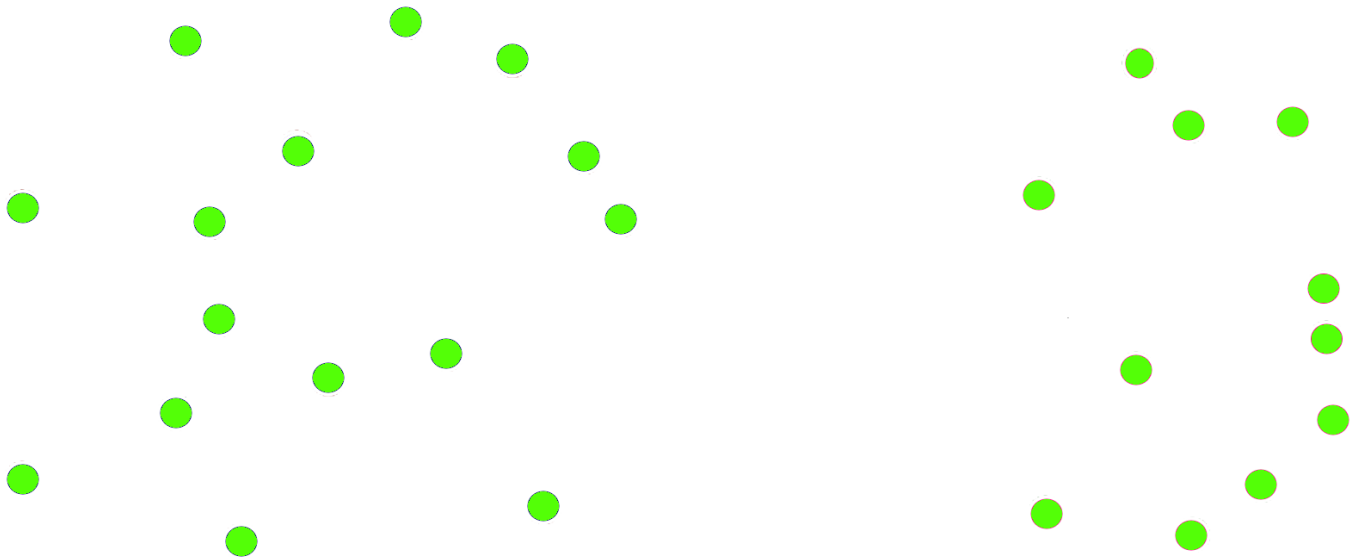
- Clustering :

$$f : \Omega^1 \times \cdots \times \Omega^d \longrightarrow \{C_1, ..., C_k\} \text{ set of clusters}$$

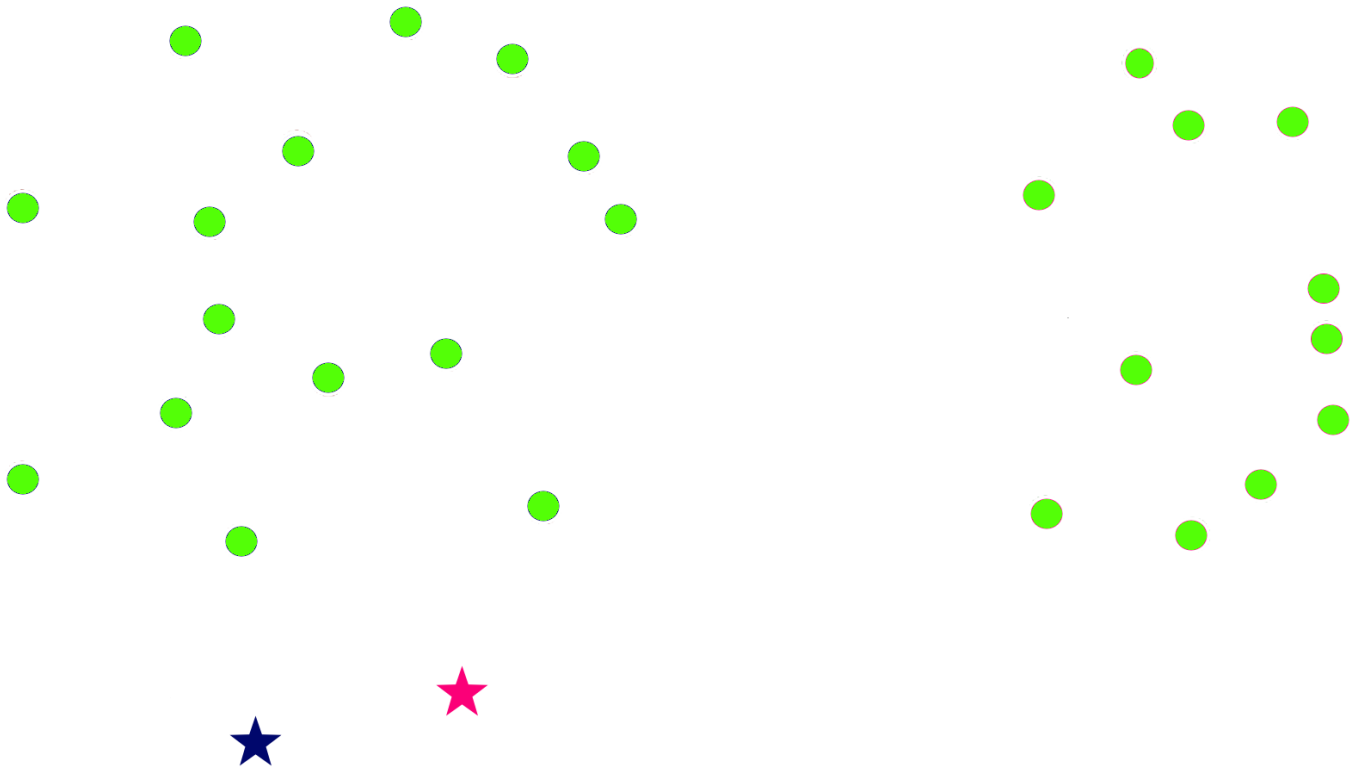
Example : Find clusters of patients/players, of therapies/exercises, ...

Course syllabus

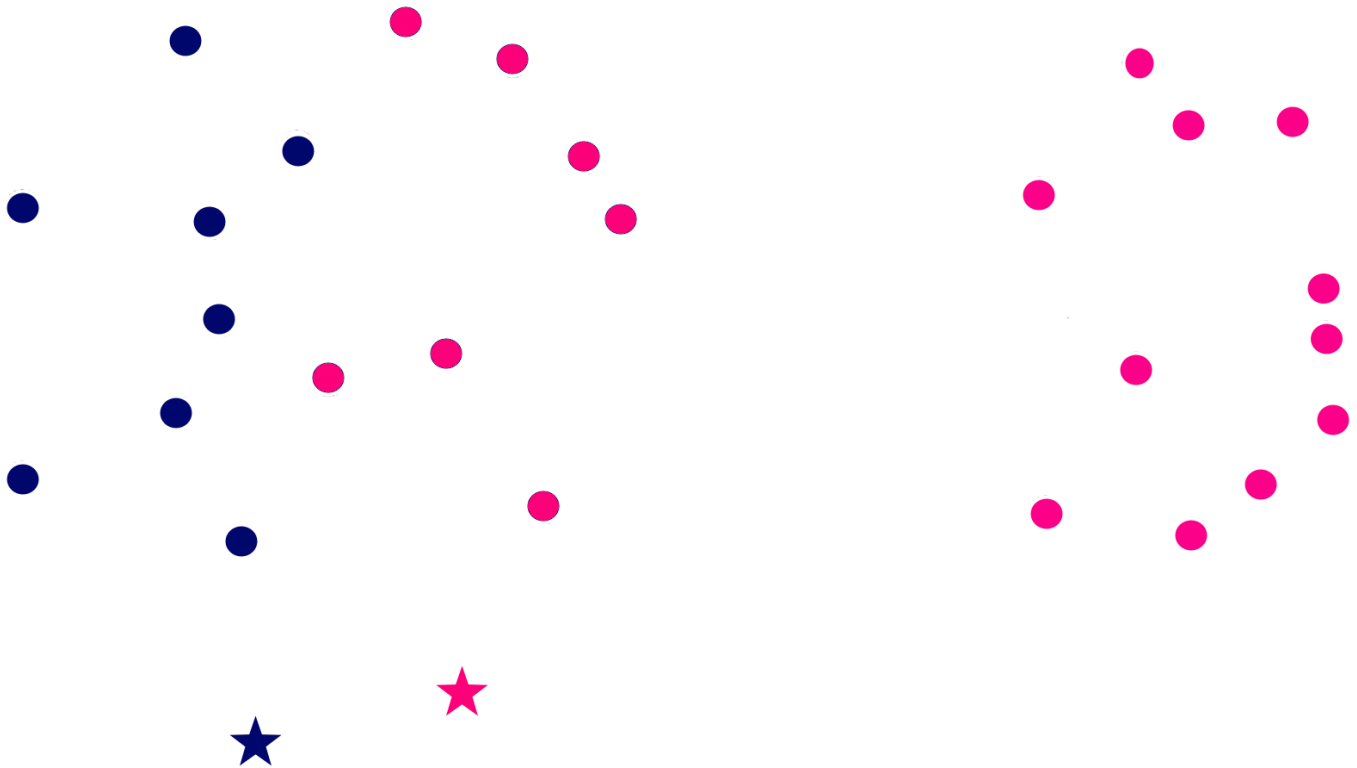
K-means



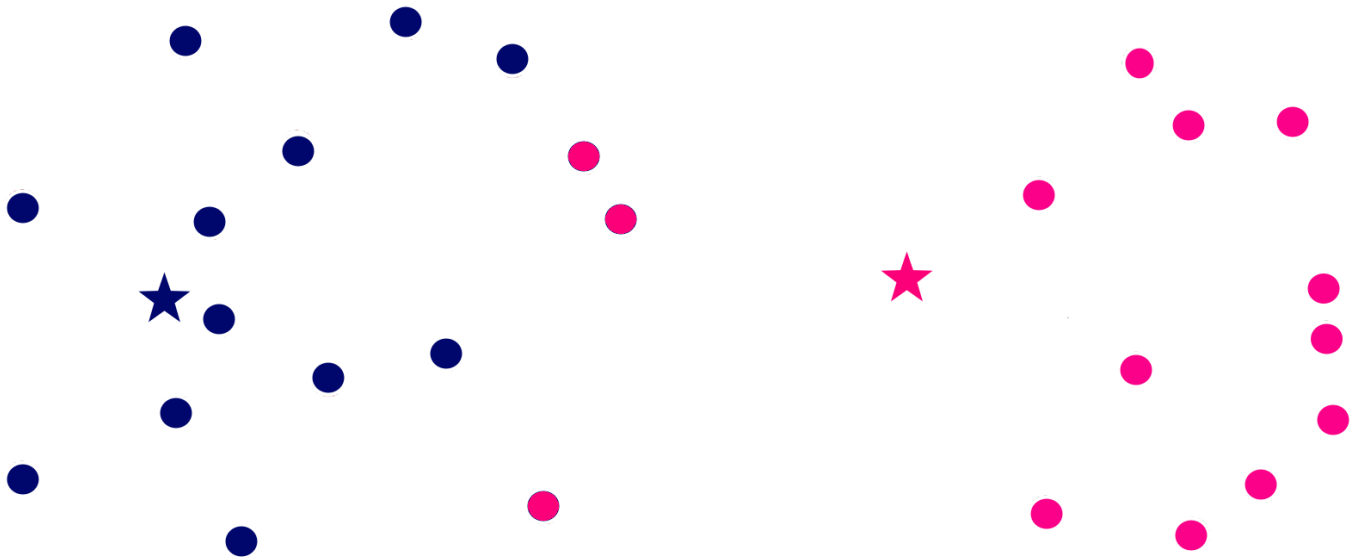
K-means



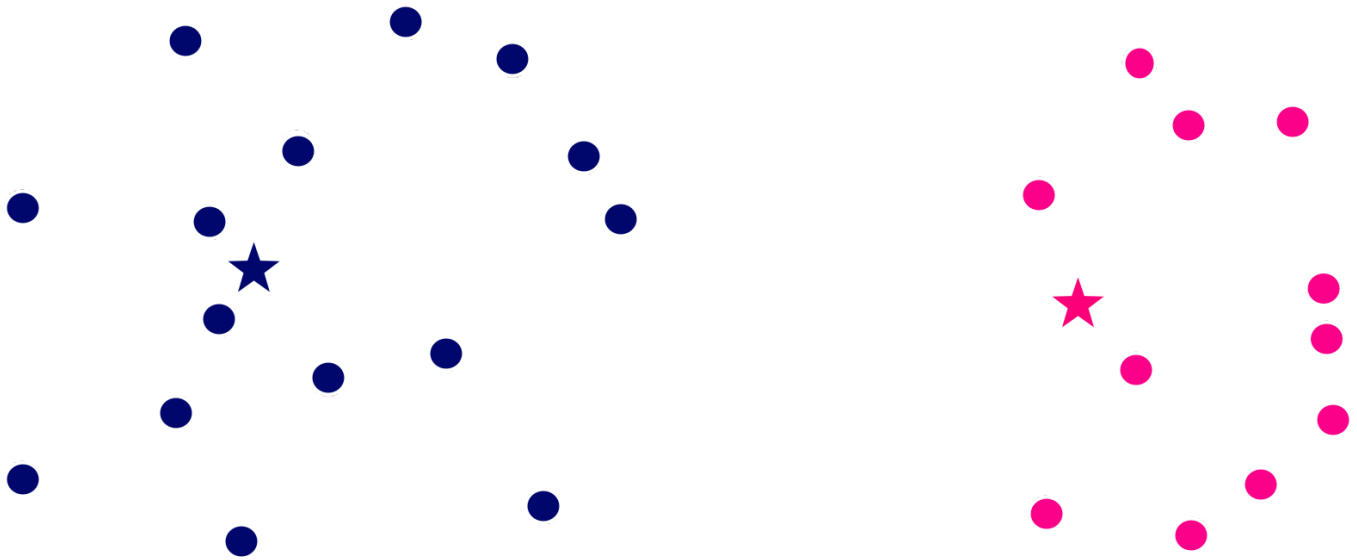
K-means



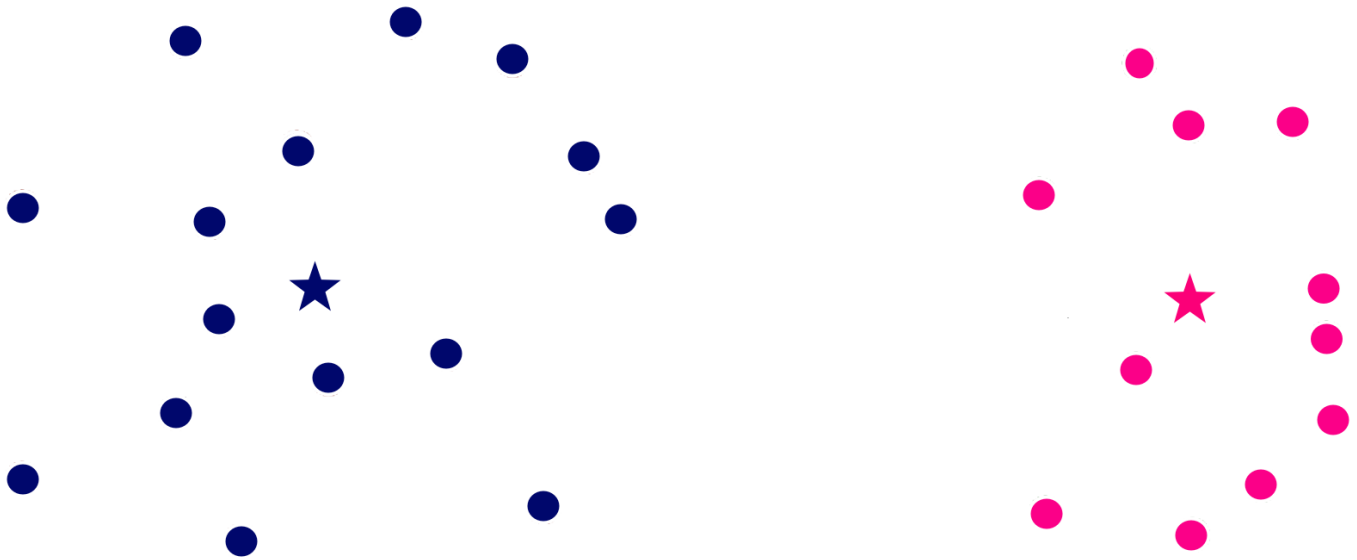
K-means



K-means



K-means



K-means

- **Goal :** Assign each example $(x_1, ..., x_n)$ to one of the K clusters $\{C_1, ..., C_K\}$.

K-means

- **Goal :** Assign each example $(x_1, ..., x_n)$ to one of the K clusters $\{C_1, ..., C_K\}$.
- Centroid μ_j is the mean of all examples in the j^{th} cluster.

K-means

- **Goal :** Assign each example $(x_1, ..., x_n)$ to one of the K clusters $\{C_1, ..., C_K\}$.
- Centroid μ_j is the mean of all examples in the j^{th} cluster.
- **Minimize :**

$$J = \sum_{j=1}^{K} \sum_{x_i \in C_j} \| x_i - \mu_j \|^2$$

Data: without labels
Result: set of clusters $\{C_1, ..., C_K\}$ and data assignement to them
Initialize randomly centers $\mu_1, ..., \mu_K$;
while *convergence** *not reached* **do**
Assign each point x_i to the cluster with the closest μ_j;
Calculate the new centers of each cluster as follows :

$$\mu_j = \frac{1}{|C_j|} \sum_{x_i \in C_j}$$

end
convergence* = no change in the clusters OR maximum number of iterations reached;

Algorithm 1: K-means

K-Means : pros and cons

- Easy to implement

  BUT...

  - Need to know K
  - Suffer from the curse of dimensionality
  - Lack of theoretical foundation

Course syllabus

Hierarchical clustering

- Hierarchical clustering is a widely used data analysis tool.
- The idea is to build a binary tree of the data that successively merges similar groups of points
- Visualizing this tree provides a useful summary of the data

Hierarchical clusering vs K-means

- Recall that k-means requires
  - A number of clusters K
  - An initial assignment of data to clusters
  - A distance measure between data d(xn, xm)
- Hierarchical clustering only requires a measure of similarity between groups of data points.

Hierarchical clustering

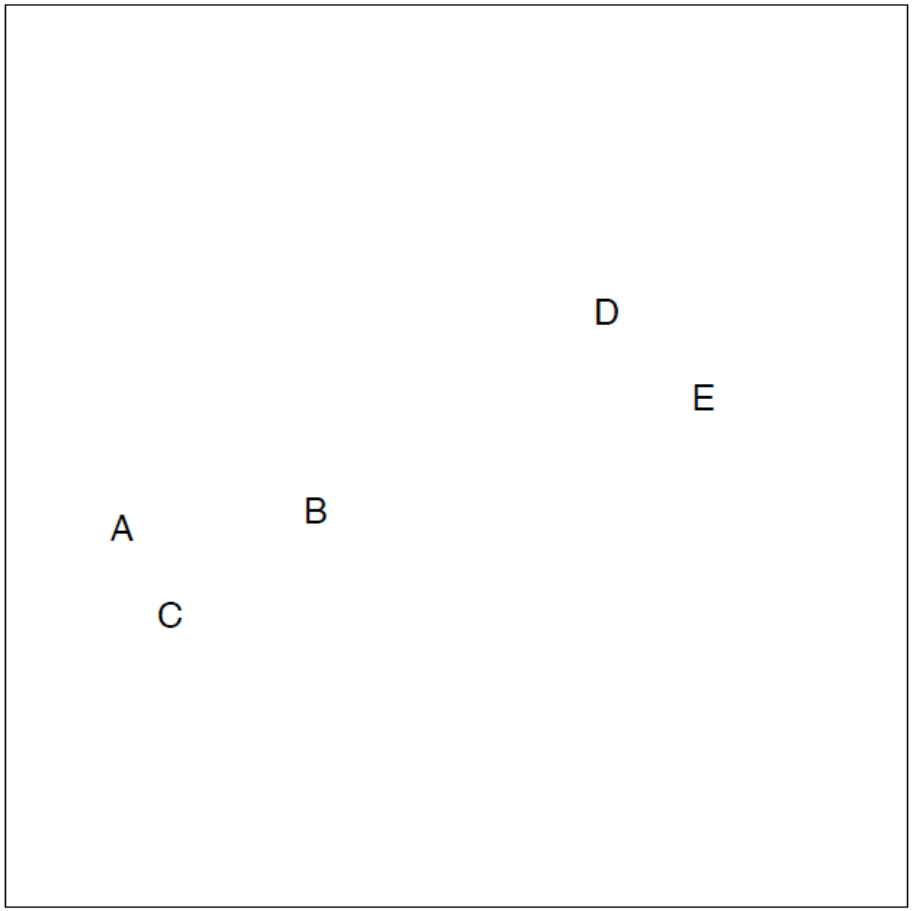
Data: without labels
Result: data tree (taxonomy)

Place each data point into its own singleton group;

while *all the data are not merged into a single cluster* **do**
    merge the two closest groups;
end

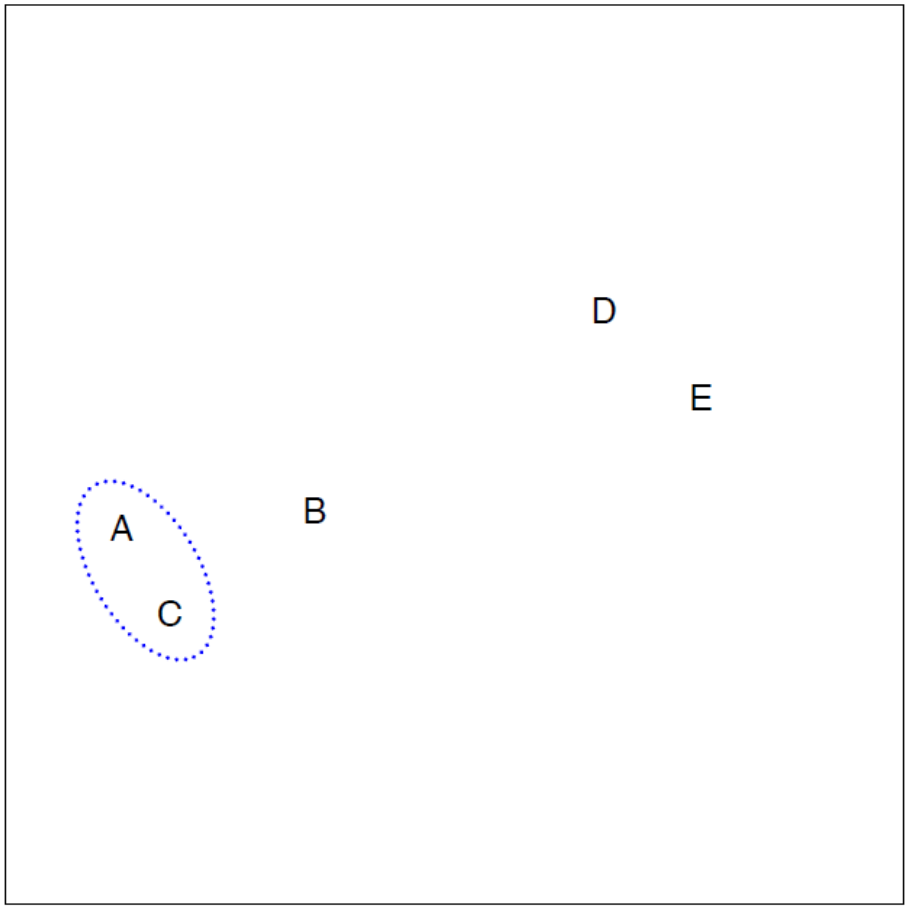
Algorithm 2: Agglomerative clustering

Hierarchical clustering



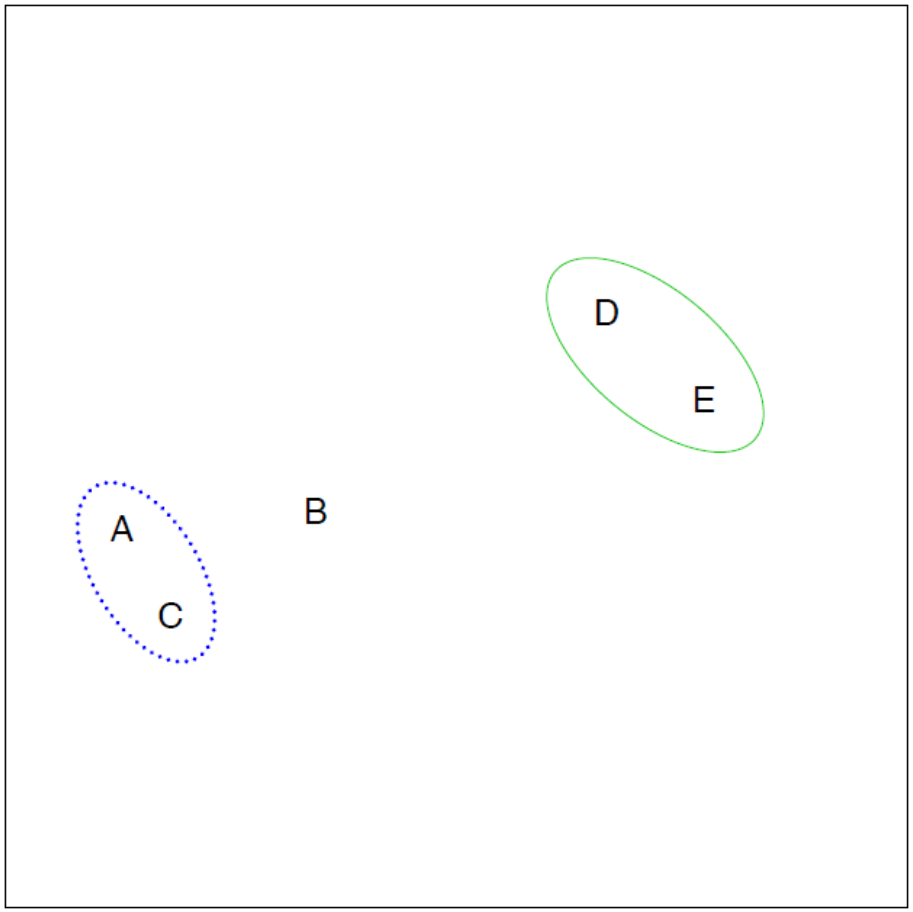
Each point starts as its own cluster.

Hierarchical clustering



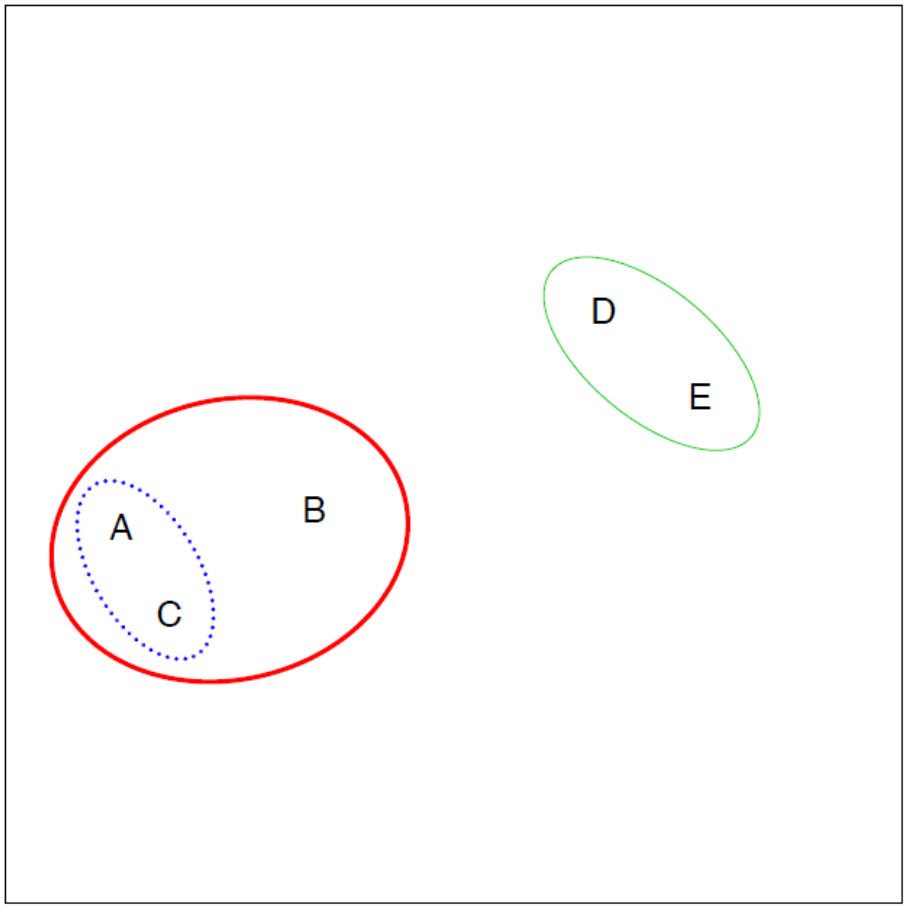
We merge the two clusters (points) that are closet to each other.

Hierarchical clustering



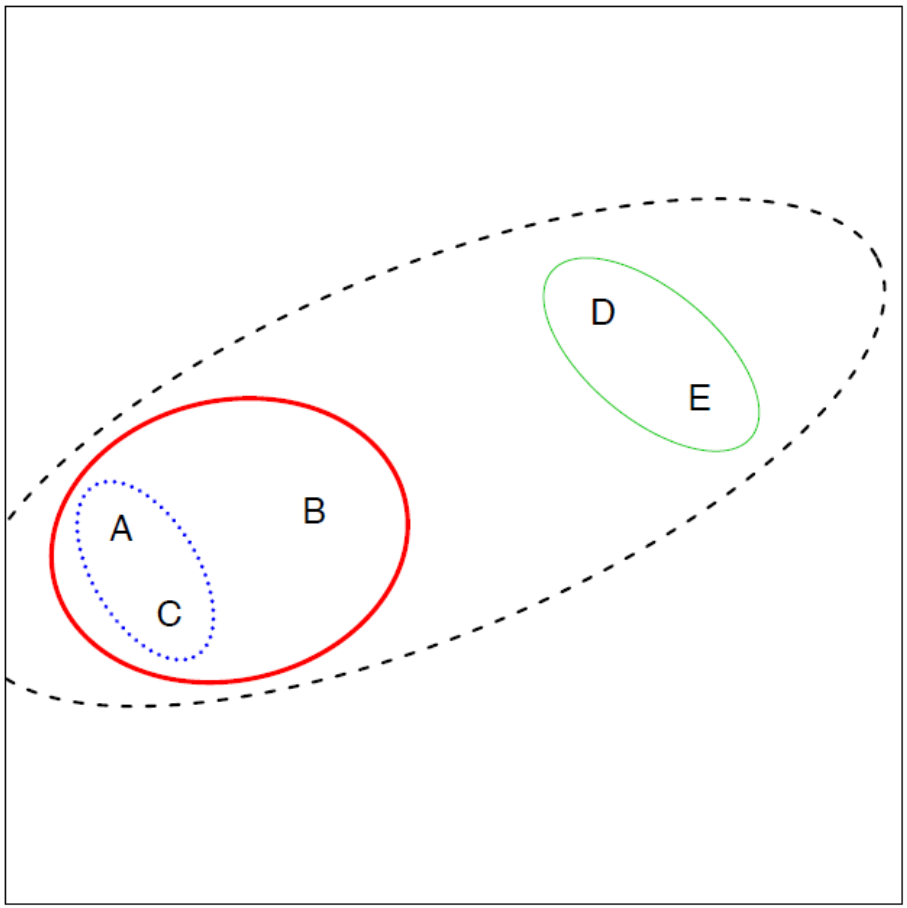
Then we merge the next two closest clusters.

Hierarchical clustering



Then the next two closest clusters...

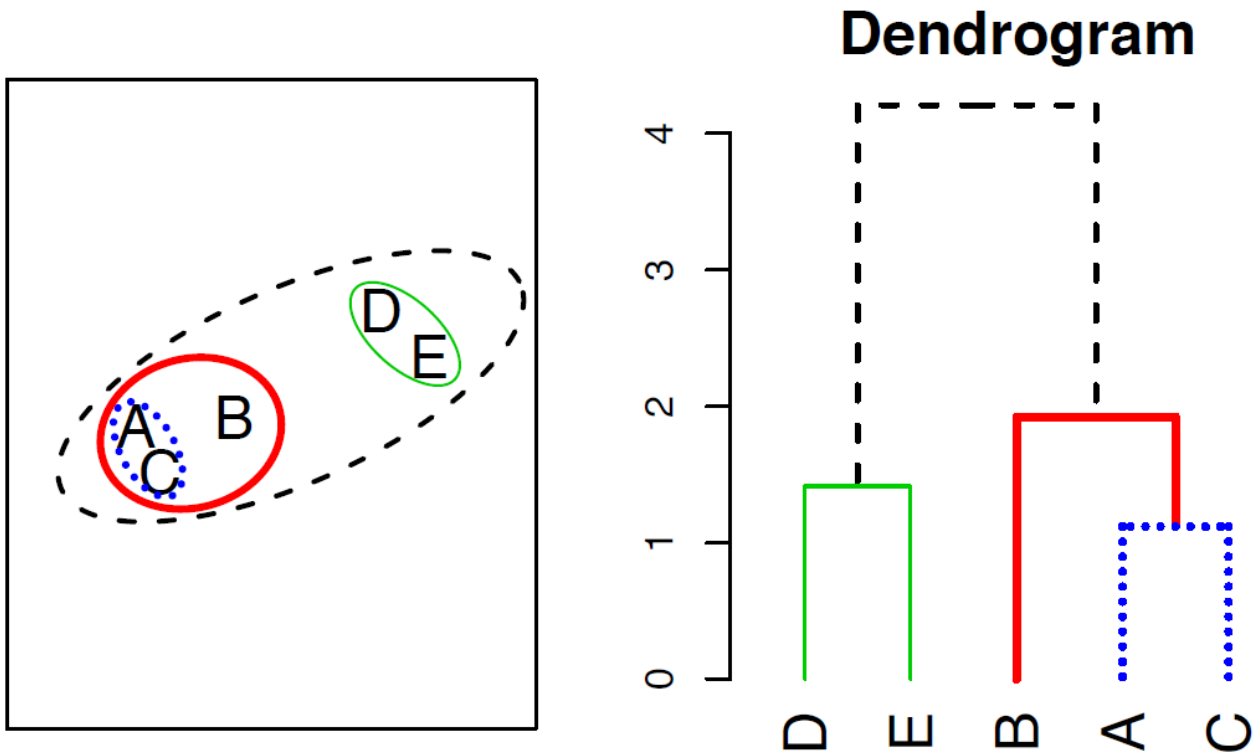
Hierarchical clustering



Until at last all of the points are all in a single cluster.

Hierarchical clustering

To visualise the results, we can look at the resulting **dendrogram**.



y-axis on dendrogram is (proportional to) the distance between the clusters that got merged at that step.

Course syllabus

Supervised learning

Given : Training data :
$(x_1, y_1), \ldots, (x_n, y_n)$, with $x_i \in \mathcal{X} = \mathcal{X}^1 \times \cdots \times \mathcal{X}^J$ and $y_i \in \Omega$ is discrete (categorical/qualitative).

Example $\Omega = \{-1, +1\}$, $\Omega = \{0, 1\}$, $\Omega = [0, 1]$.

Task : Learn a classification/regression function f_θ, i.e. estimate θ, such that :

$$\begin{array}{rccc} f_\theta : & \mathcal{X} & \to & \Omega \\ & x_i & \mapsto & f_\theta(x_i) \approx y_i \end{array}$$

Supervised learning examples

- **Classification :**

  Flue from average daily number of steps ! Who will have the flue this winter ?

# steps	flue
12000	no
8000	no
5000	yes
⋮	⋮

- **Regression :**

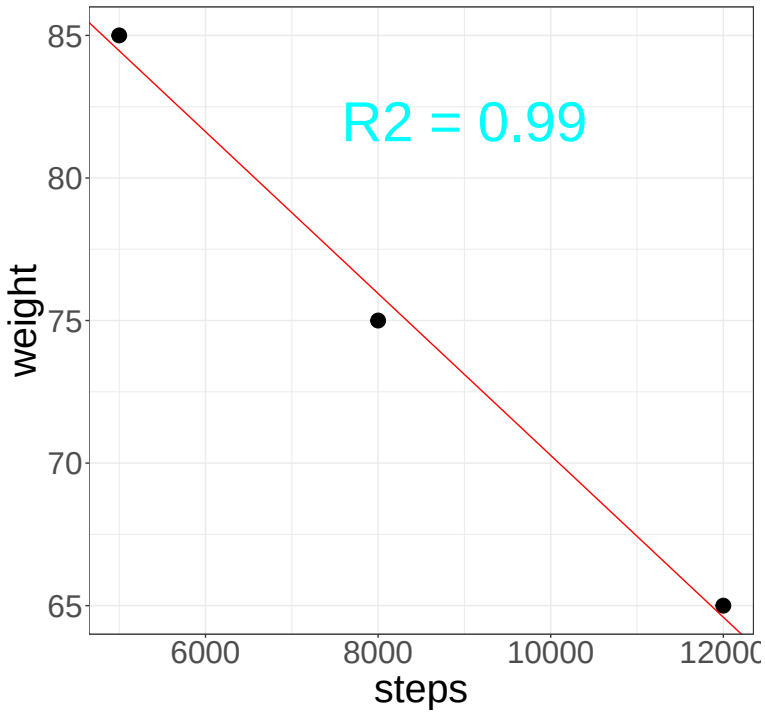
  Summer weight from average daily number of steps ! Who will be thin this summer ?

# steps	weight
12000	65
8000	70
5000	85
⋮	⋮

Course syllabus

Linear regression

We want to estimate $\theta = (a, b)$ such that weight $= a.\text{steps} + b$.



least squares estimation $\rightarrow$ here $a = -2.8$ and $b = 98.6$

$R2 = \frac{cov(X,Y)^2}{Var(X)Var(Y)} \in [0, 1]$ represents the regression quality (1 for perfection).

Prediction : what will be the weight of someone doing 10000 daily steps ?

Course syllabus

Logistic regression = a classification model

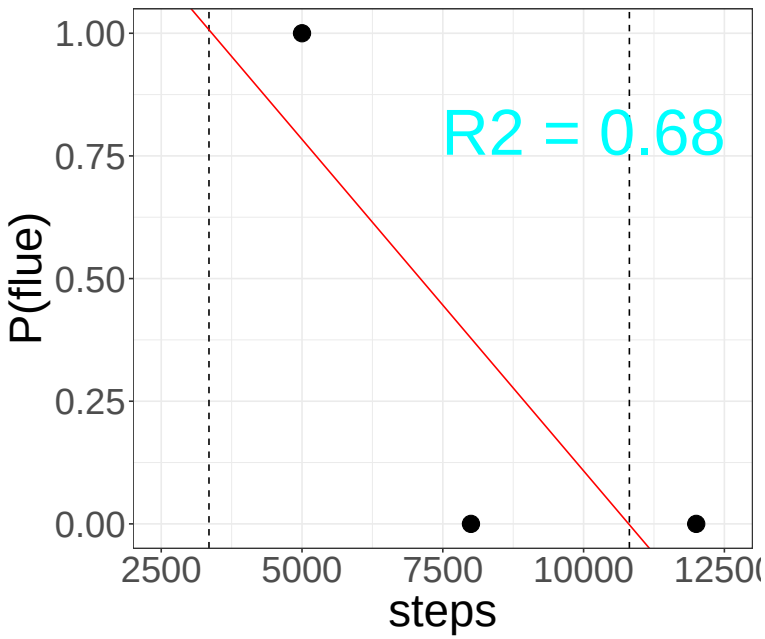
- We can't predict *flue* with any certainty. Suppose we want to predict how likely is someone to have the flue. We must compute a probability between 0 and 1 that he will have the flue.
- It makes sense and would be suitable and practical.
- In this case, the output is numerical (regression) but is bounded (classification).

$$P(y|x) = P(\text{flue} = \text{yes} \mid \#\text{steps})$$

Classification

- Can we use linear regression ?
- Yes. However...
  - Works only for Binary classification (2 classes). Won't work for Multiclass classification e.g.
    $\Omega = \{\text{green, blue, brown}\}$
    $\Omega = \{\text{stroke, heart attack, drug overdose}\}$
  - If we use linear regression, some of the predictions will be outside of [0,1].
  - Model can be poor. Example.

Logistic regression



$steps > 10813 \rightarrow P(flue) < 0\,!$

and

$steps < 3348 \rightarrow P(flue) > 1\,!$

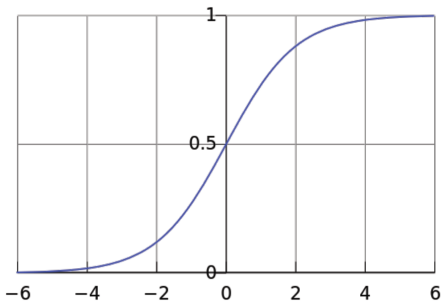
Logistic regression

$y = f(x) = \beta_0 + \beta_1 x$

flue $= \beta_0 + \beta_1 \times$ **steps**

We want $0 \leq f(x) \leq 1$; $f(x) = P(y = 1|x)$

We use the sigmoid function :



$$g(z) = \frac{e^z}{1+e^z} = \frac{1}{1+e^{-z}}$$

$$g(z) \xrightarrow[z \to -\infty]{} 0 \quad \text{and} \quad g(z) \xrightarrow[z \to +\infty]{} 1$$

Logistic Regression

$$g(\beta_0 + \beta_1 x) = \frac{e^{\beta_0+\beta_1 x}}{1 + e^{\beta_0+\beta_1 x}}$$

New $f(x) = g(\beta_0 + \beta_1 x)$

In general :

$$f(x) = g\left(\sum_{j=1}^{d} \beta_j x_j\right)$$

In other words, cast the output to bring the linear function quantity between 0 and 1.
Note : One can use other S-shaped functions.

Course syllabus

K-nearest neighbours

- Not every ML method builds a model !
- Main idea of *KNN* : Uses the similarity between examples.
- Assumption : Two similar examples should have same labels.
- Numerical features : $\mathcal{X}^1 \times \cdots \times \mathcal{X}^J \subset \mathbb{R}^J$.
- Assumes all examples (instances) are points in the J dimensional space $\mathcal{X}^1 \times \cdots \times \mathcal{X}^J$.
- *KNN* uses the standard Euclidian distance (*usually*) to define nearest neighbours.

  Given two examples x_{i_1} and x_{i_2}, $d(x_{i_1}, x_{i_2}) = \sqrt{\sum_{j=1}^{d} (x_{i_1}^j - x_{i_2}^j)^2}$

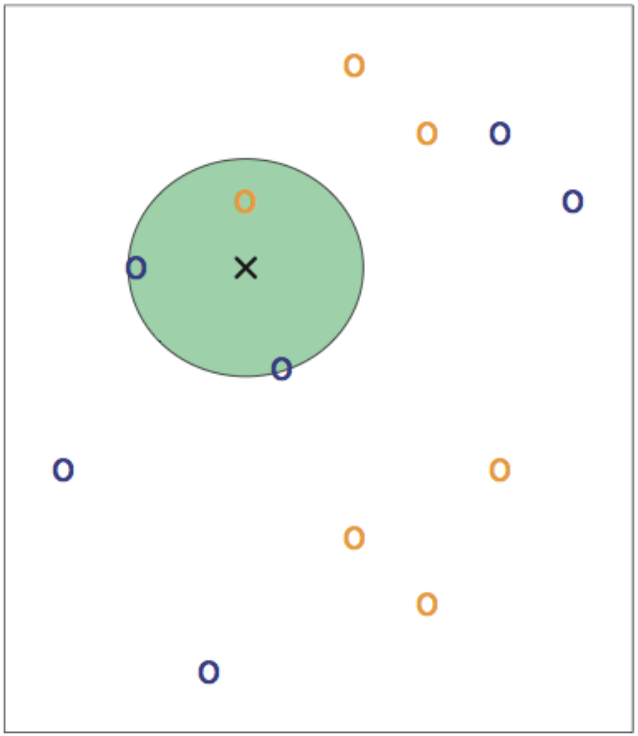
K-nearest neighbours

Training algorithm :

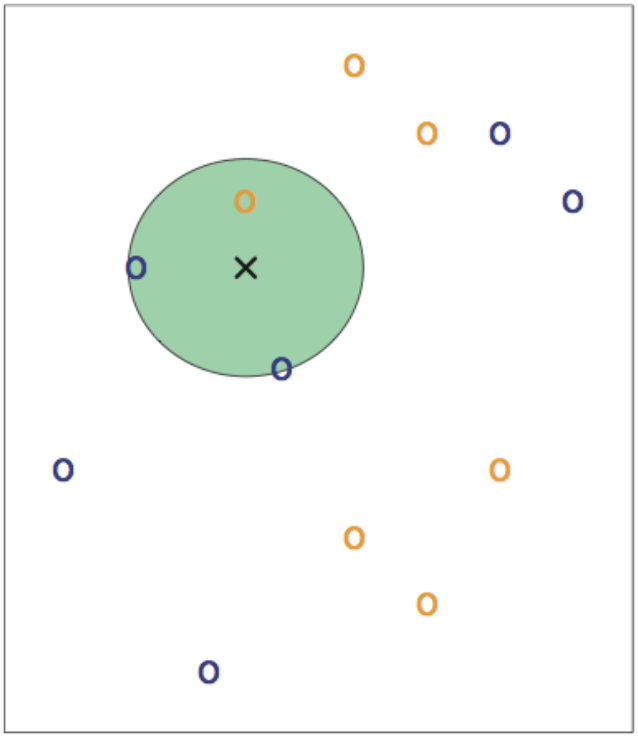
Given an example x_q to be predicted : suppose $N_K(x_q)$ is the set of the K-nearest neighbours of x_q :

- **Classification :** $\hat{y}_q = sign\left(\sum_{x_i \in N_K(x_q)} y_i\right)$
- **Regression :** $\hat{y}_q = \frac{1}{N_K(x_q)} \sum_{x_i \in N_K(x_q)} y_i$

K-nearest neighbours

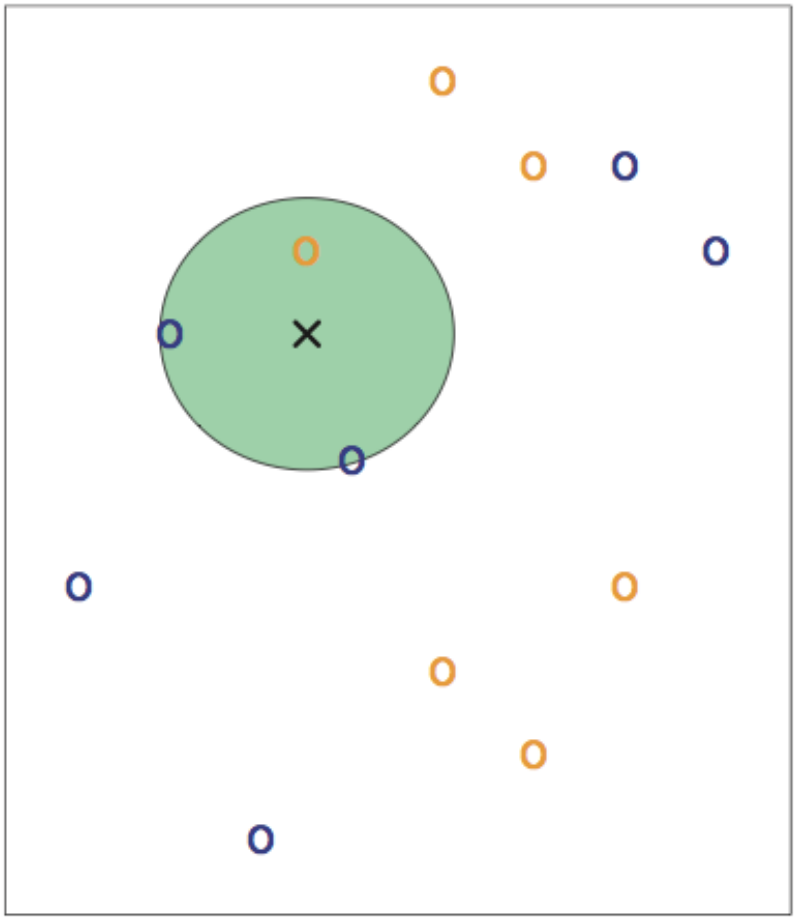


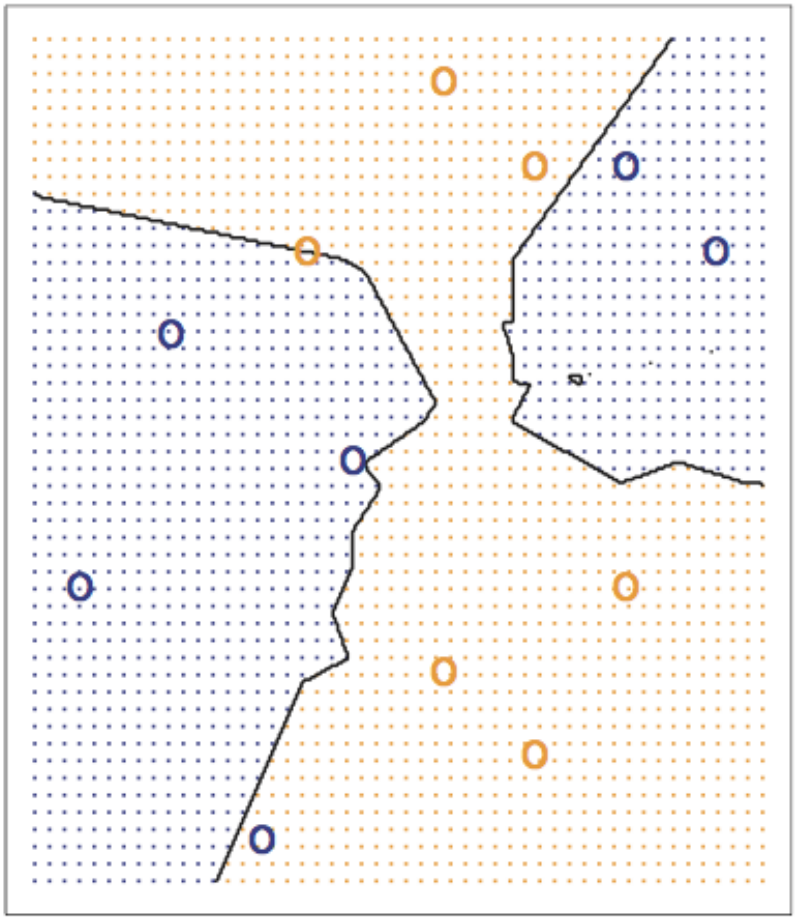
K-nearest neighbours



Question : Draw an approximate decision boundary for $K = 3$?

K-nearest neighbours





K-nearest neighbours

Question : What are the pros and cons of *KNN* ?

Pros :

+ Simple to implement.
+ Works well in practice.
+ Does not require to build a model, make assumptions, tune parameters.
+ Can be extended easily with news examples.

Cons :

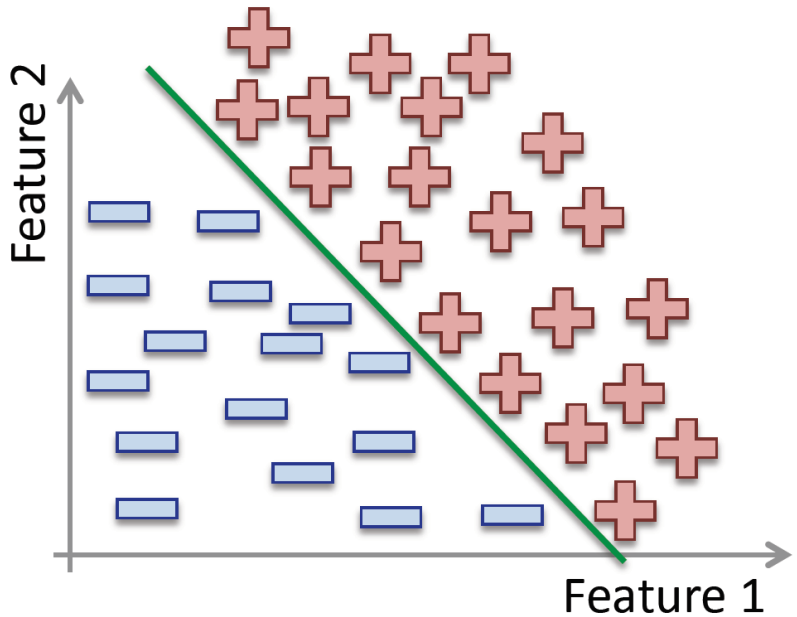
- Requires large space to store the entire training dataset.
- Slow ! Given n examples and J features. The method takes $\mathcal{O}(n \times J)$ to run.
- Suffers from the curse of dimensionality.

Course syllabus

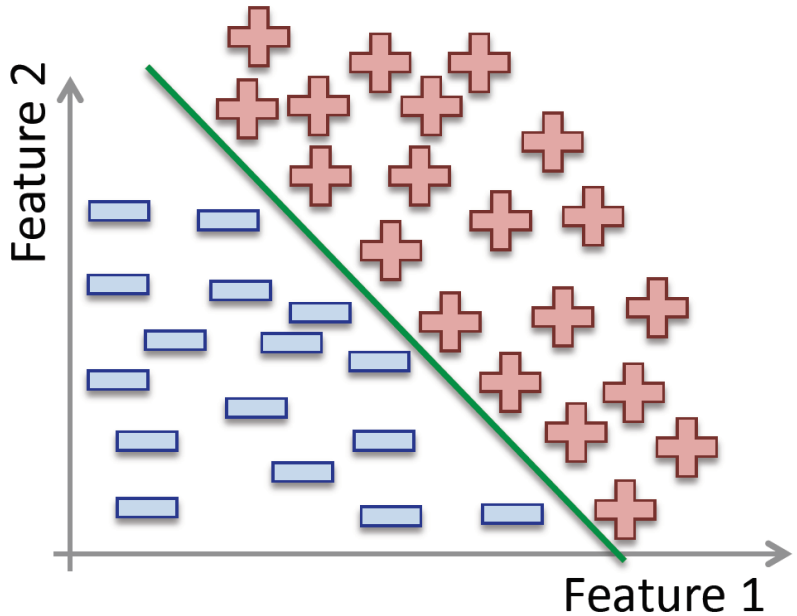
Perceptron

- Belongs to Neural Networks class of algorithms (algorithms that try to mimic how the brain functions).
- The first algorithm used was the Perceptron (Rosenblatt 1959).
- Worked extremely well to recognize :
  - handwritten characters (LeCun et al. 1989)
  - spoken words (Lang et al. 1990)
  - faces (Cottrel 1990)
- NN were popular in the 90's but then lost some of its popularity.
- Now NN back with deep learning.

Perceptron

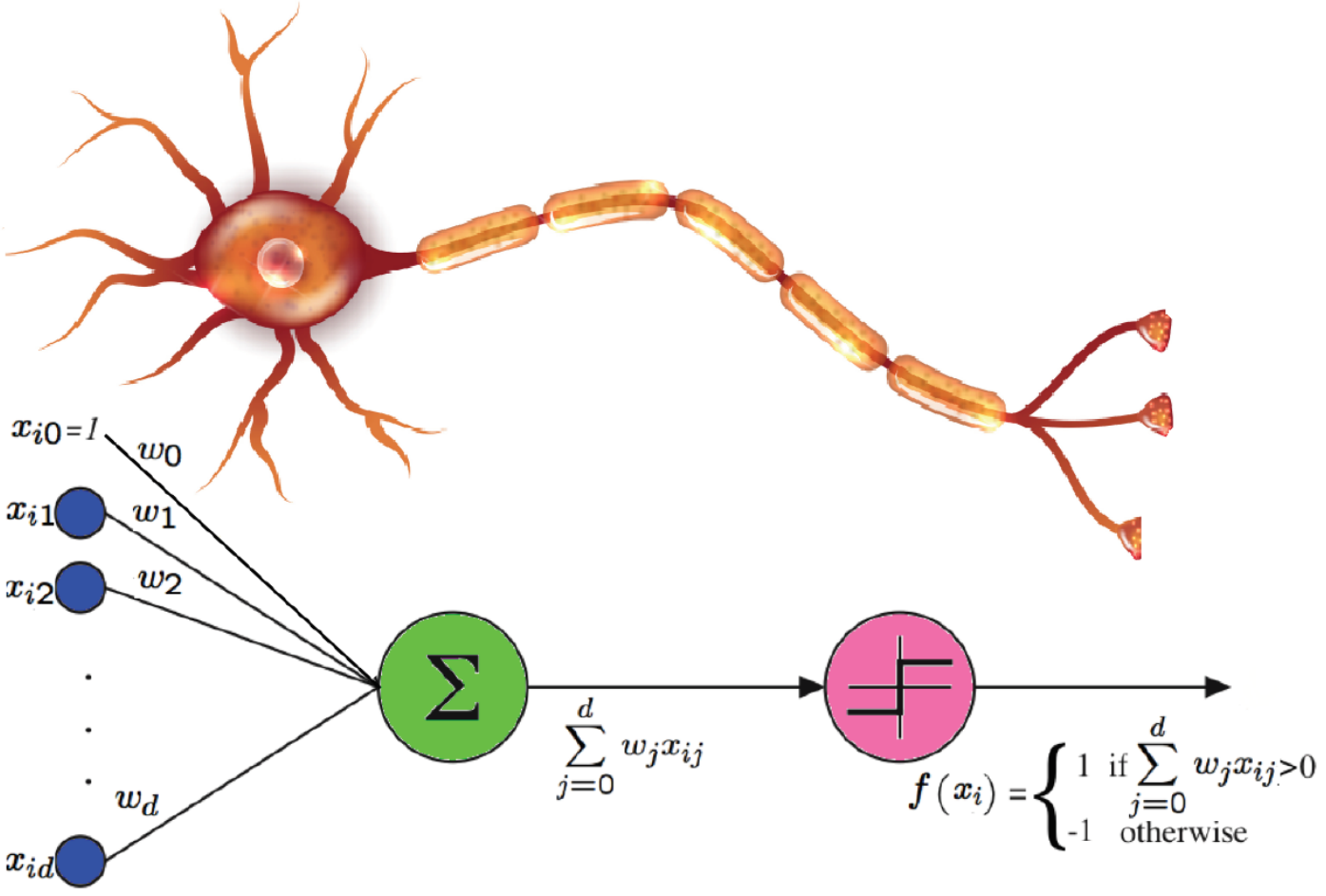


Perceptron



- Linear classification method.
- Simplest classification method.
- Simplest neural network.
- For perfectly separated data.

Perceptron



Given n examples and d features, $f(x_i) = sign\left(\sum_{j=0}^{d} w_j x_j\right)$

Perceptron

- Works perfectly if data is linearly separable. If not, it will not converge.
- Idea : Start with a random hyperplane and adjust it using your training data.
- Iterative method.

Perceptron

Data: A set of examples, $(x_1, y_1), ..., (x_n, y_n)$

Result: A perceptron defined by $(w_0, w_1, ..., w_d)$

Initialize the weights w_j to 0 $\forall j \in \{1, ..., d\}$;

while *convergence not reached* **do**
  update all w_j :
  for $j \in \{1, ..., d\}$ **do**
    for $i \in \{1, ..., n\}$ **do**
      if $y_i f(x_i) \leq 0$ *#i.e. error* **then**
        $w_j := w_j + y_i x_i$
      end
    end
  end
end

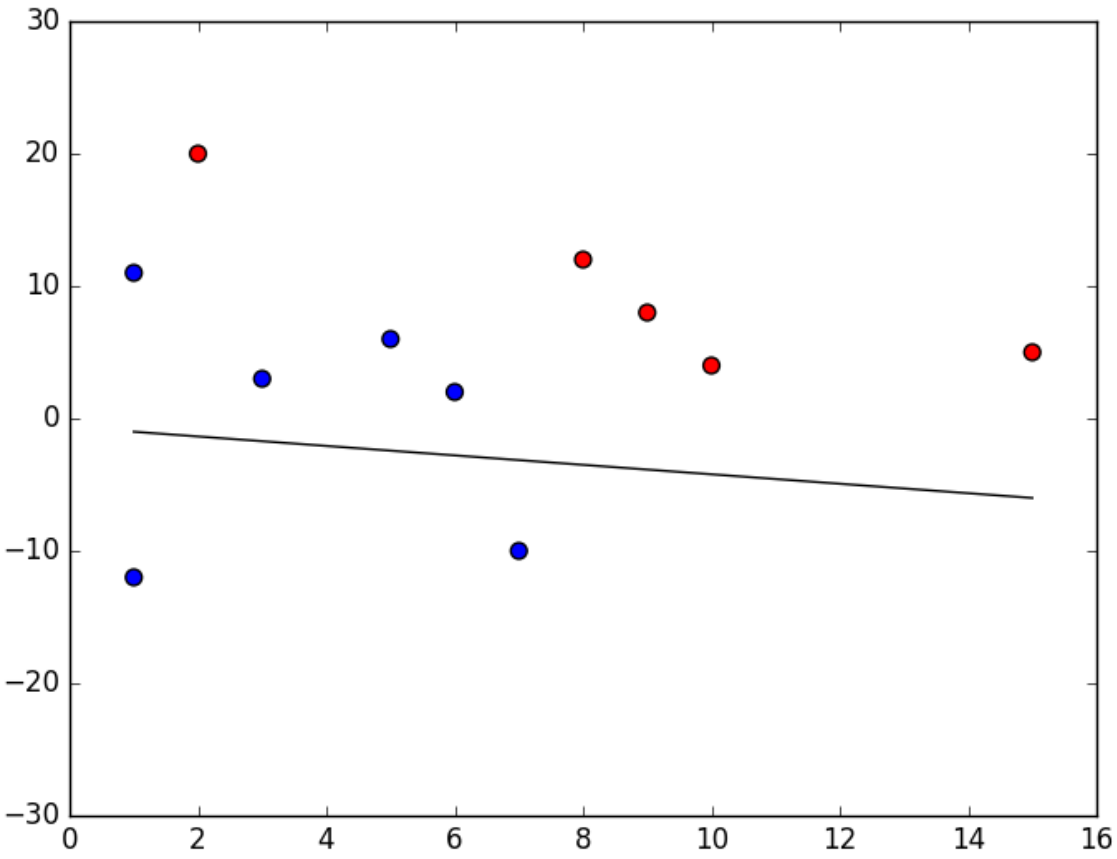
Algorithm 3: Perceptron

Perceptron

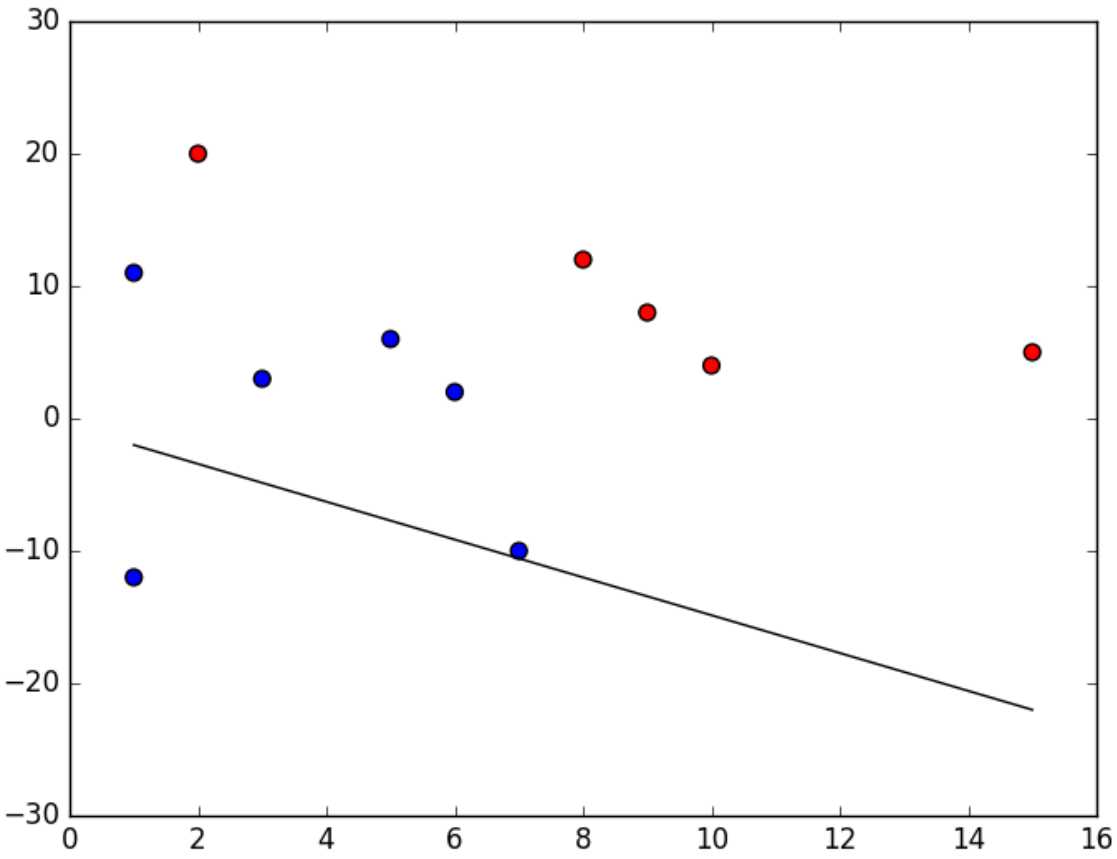
Some observations :

- The weights $w_1, ..., w_d$ determine the slope of the decision boundary.
- w_0 determines the offset of the decision boundary (can be noted b).
- weights adjustment corresponds to :
  - Mistake on positive : add x to weight vector.
  - Mistake on negative : substract x from weight vector.
  - Some other variants of the algorithm add or subtract 1.
- Convergence happens when the weights do not change anymore (difference between the last two weight vectors is $< \epsilon$).

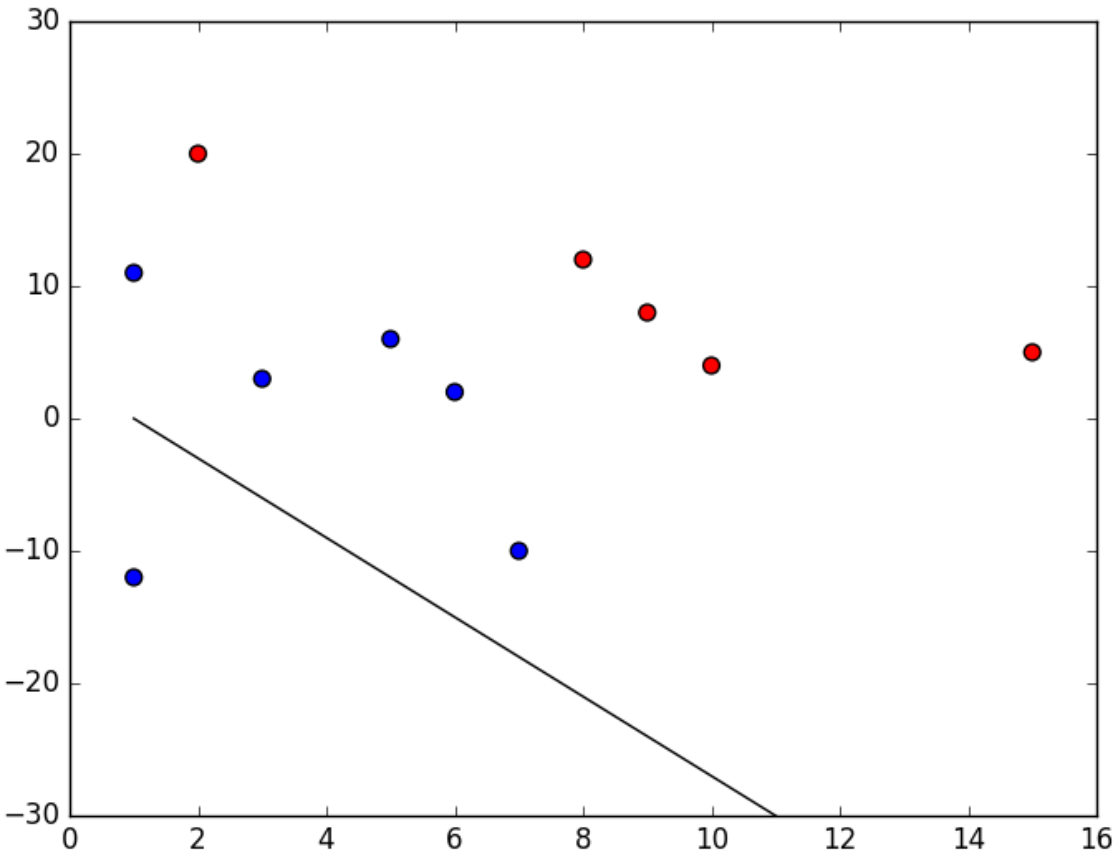
Perceptron



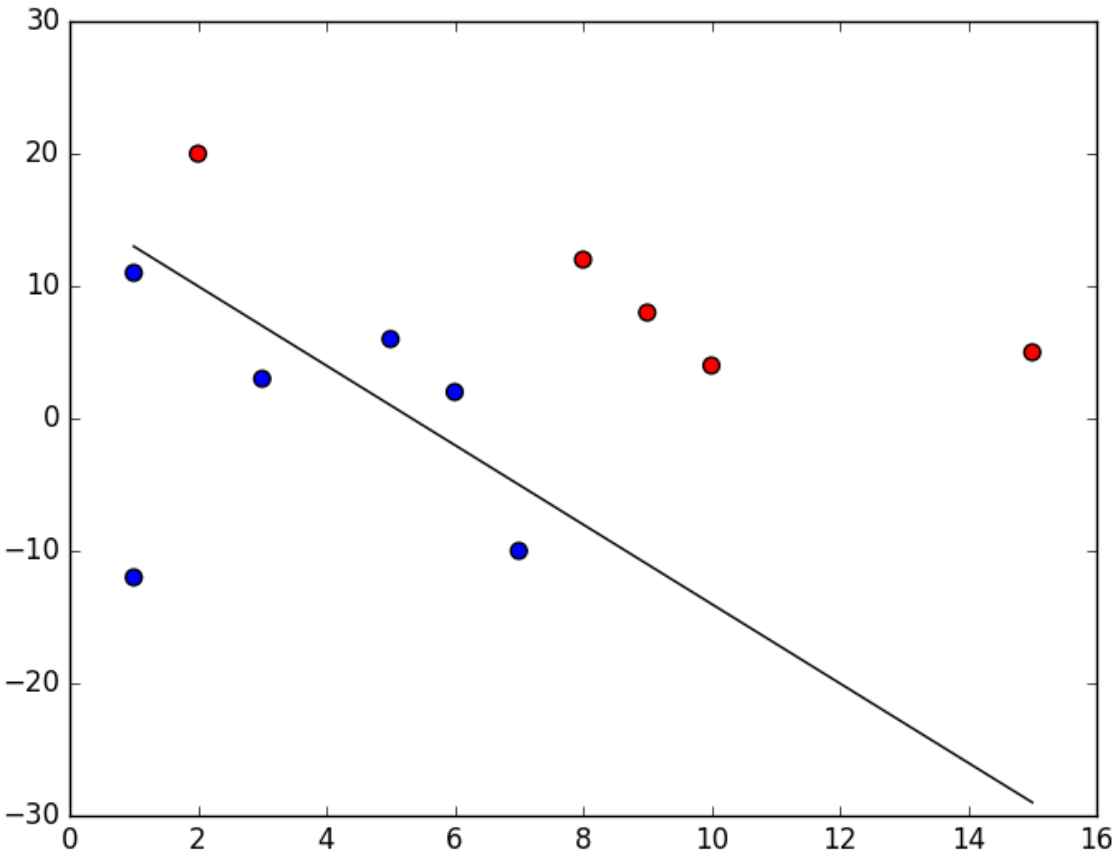
Perceptron



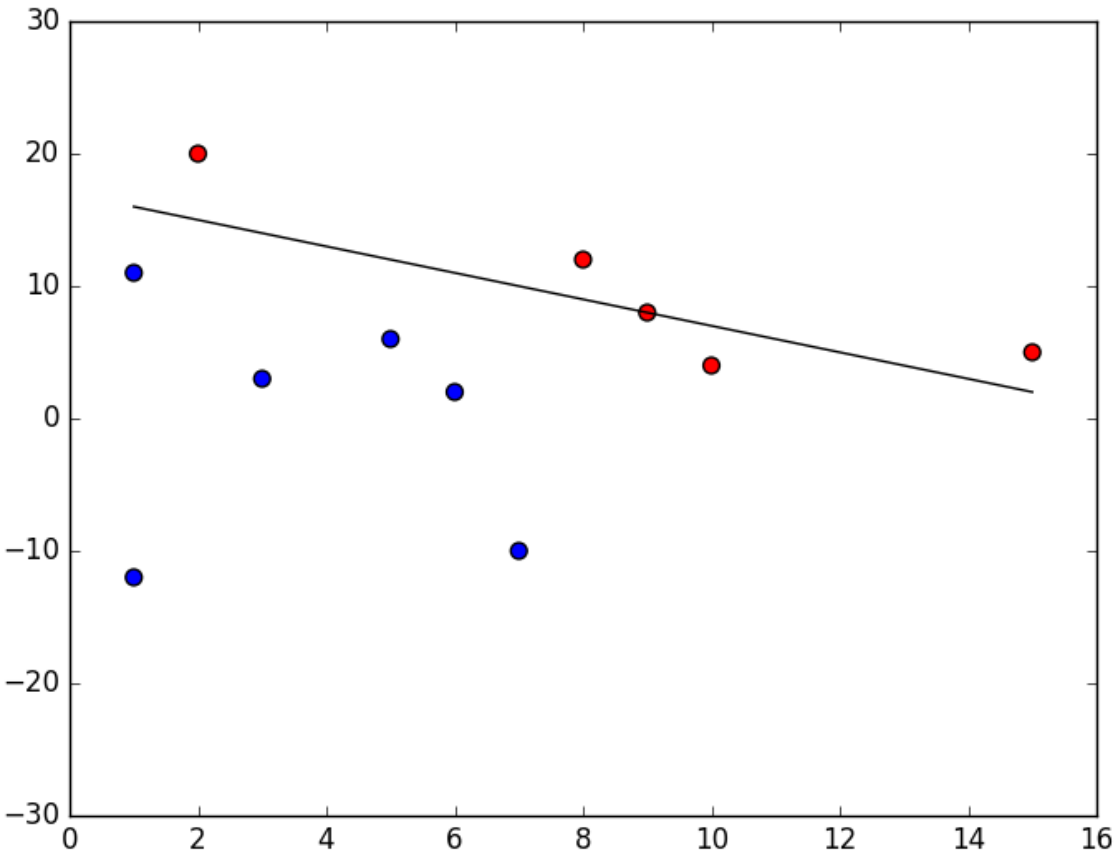
Perceptron



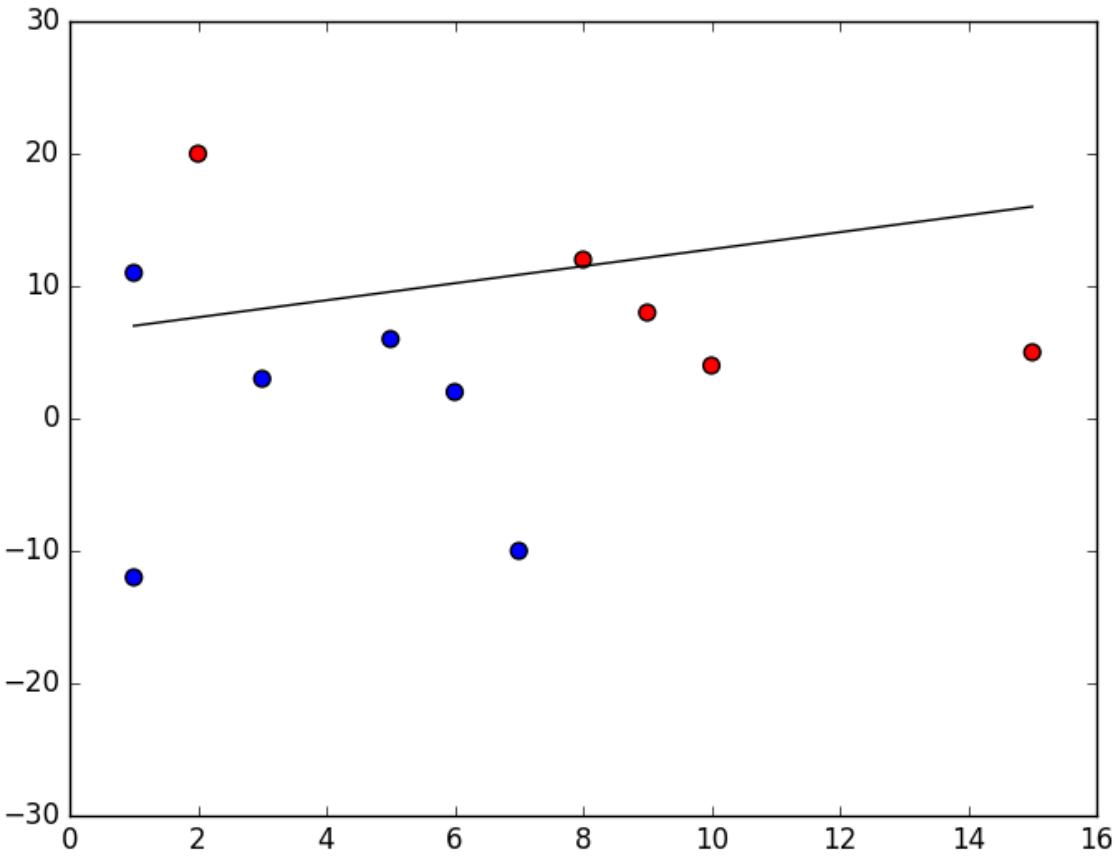
Perceptron



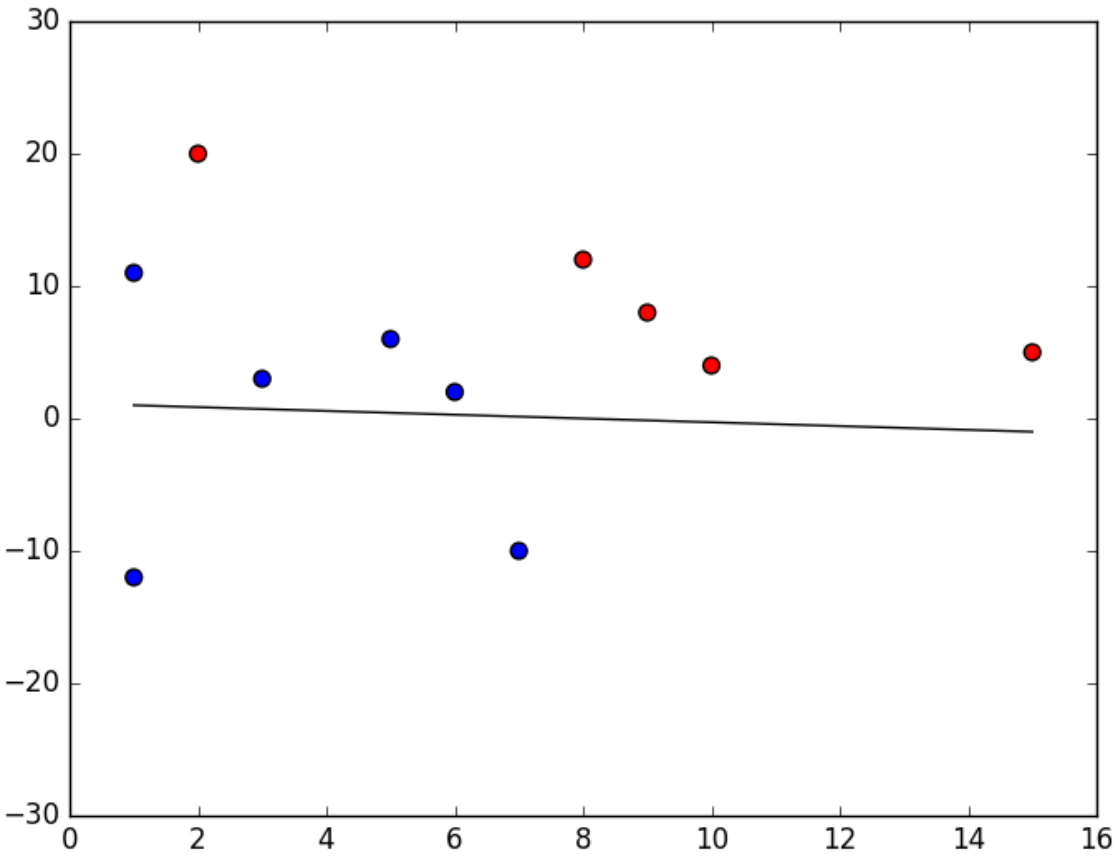
Perceptron



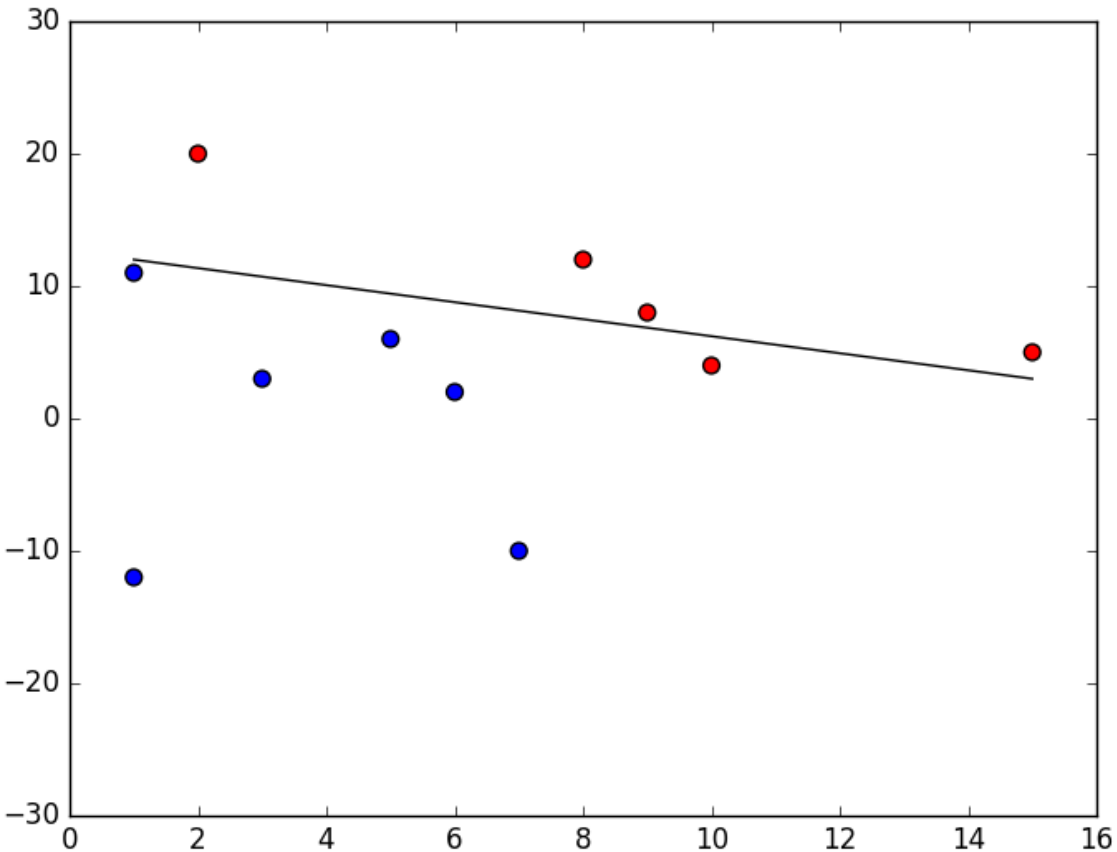
Perceptron



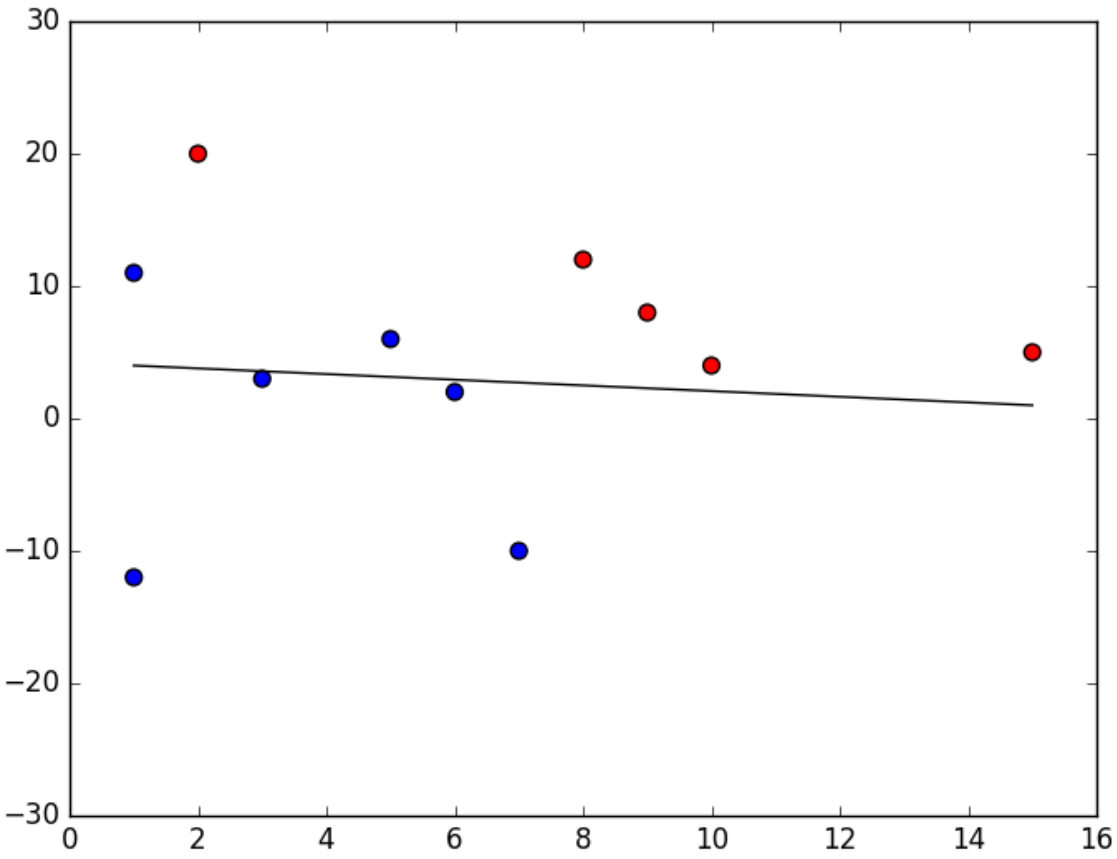
Perceptron



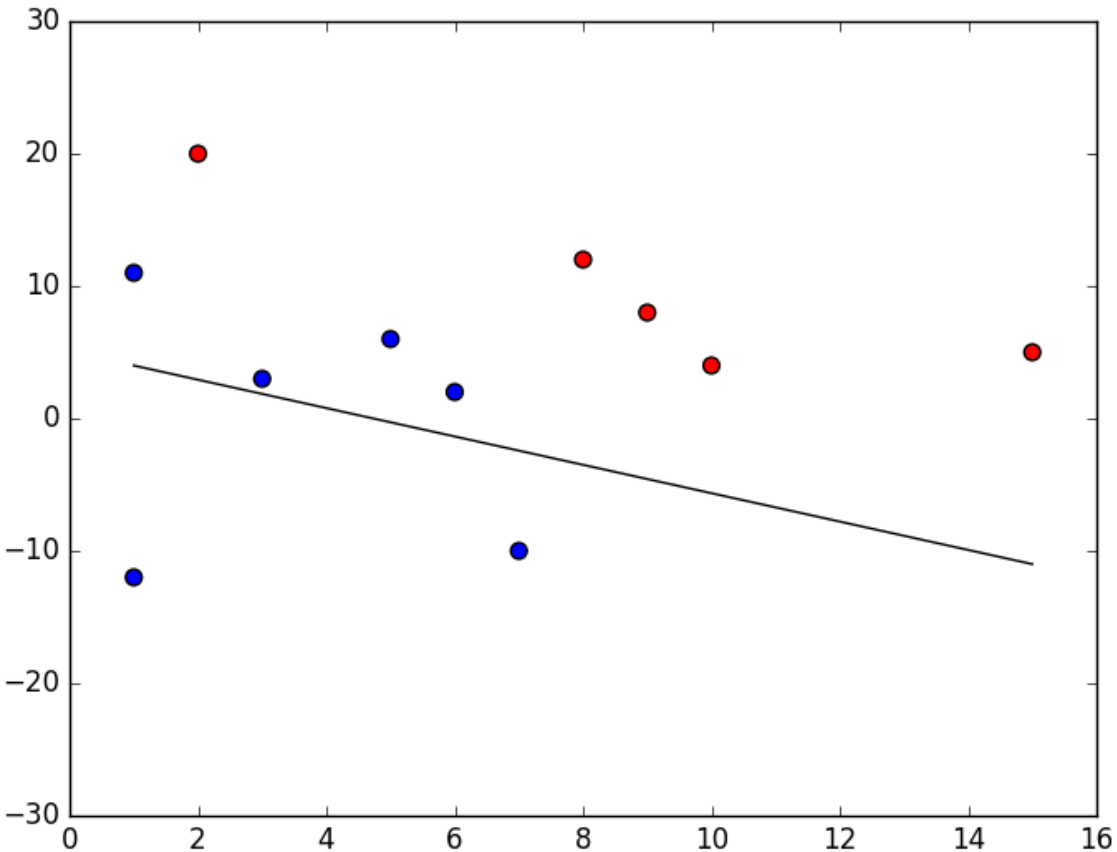
Perceptron



Perceptron

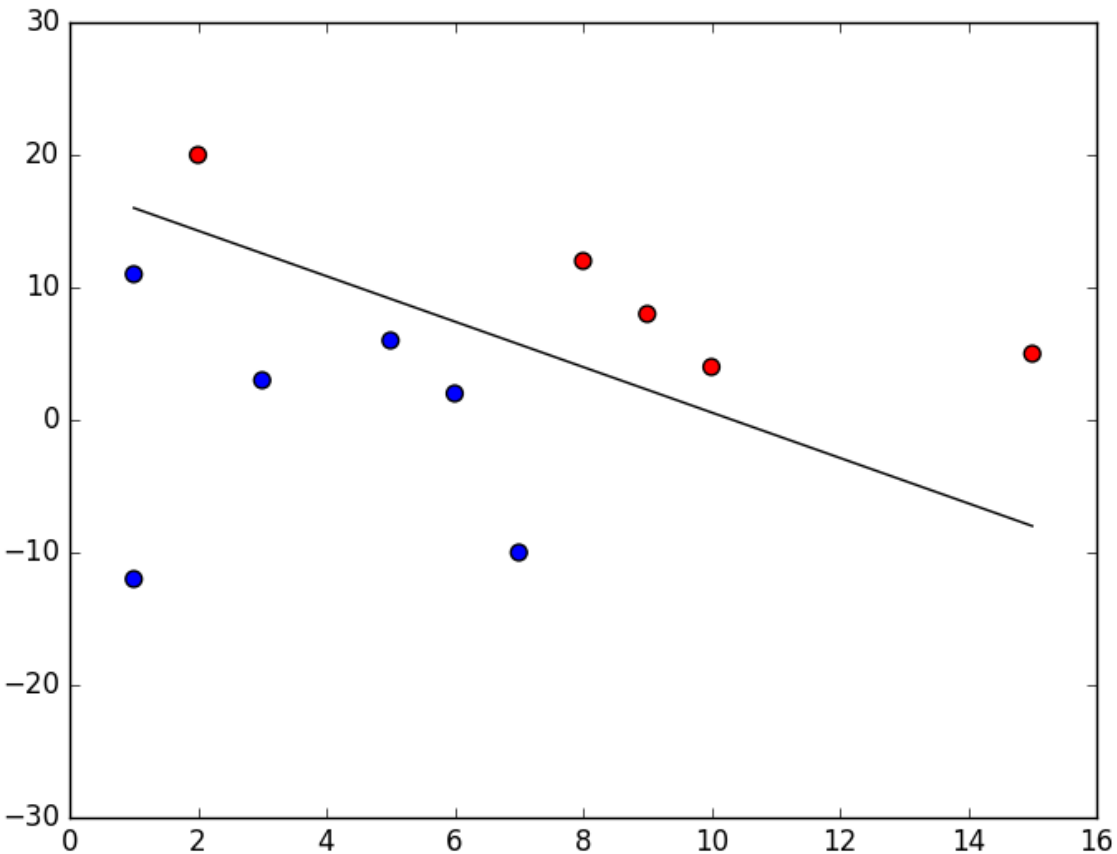


Perceptron



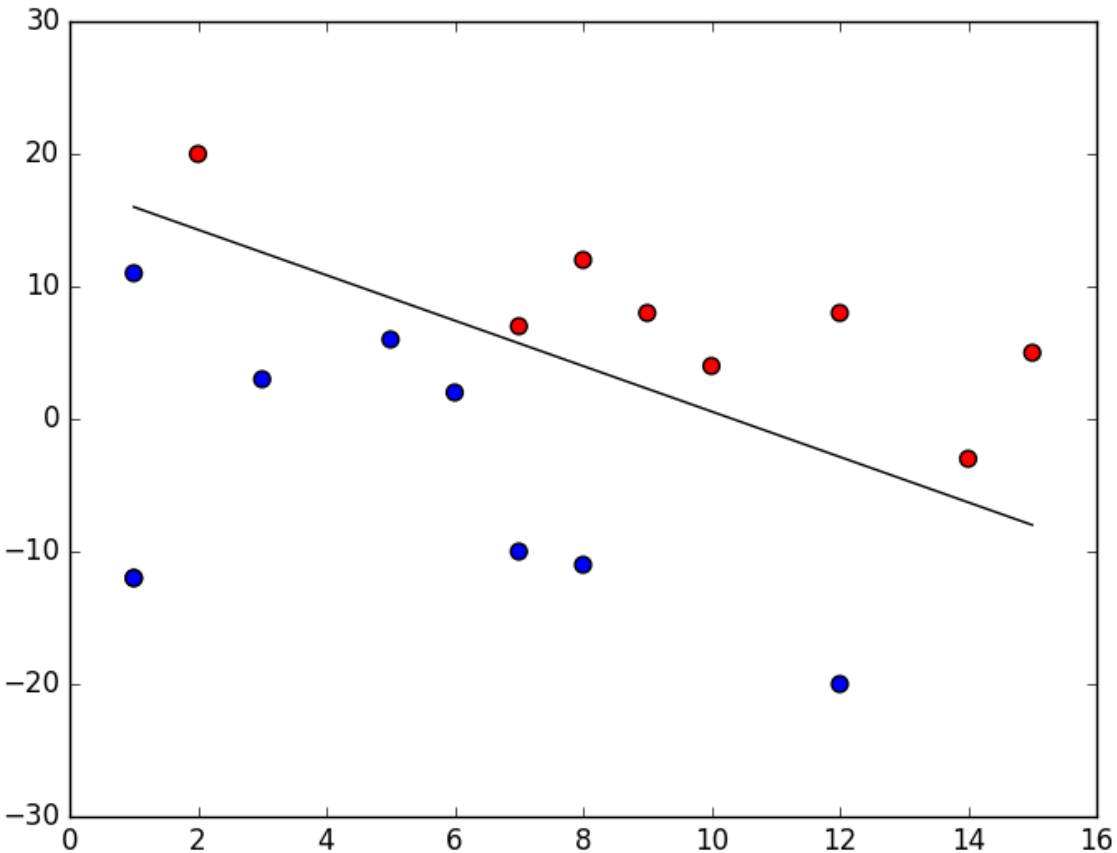
Perceptron

Finally converged !



Perceptron

With some test data :



Perceptron expressiveness

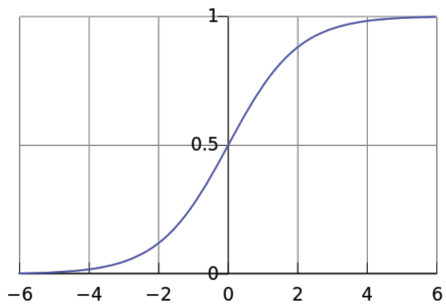
- Consider the perceptron with the *activation* function.
- Idea : Iterative method that starts with a random hyperplane and adjust it using your training data.
- It can represent Boolean functions such as AND, OR, NOT but not the XOR function.
- It produces a linear separator in the input space.

From perceptron to MLP

- The perceptron works perfectly if data are linearly separable. If not, it will not converge.
- Neural networks use the ability of the perceptrons to represent elementary functions and combine them in a network of layers of elementary questions.
- However, a cascade of linear functions is still linear,
- and we want networks that represent highly non-linear functions.

From perceptron to MLP

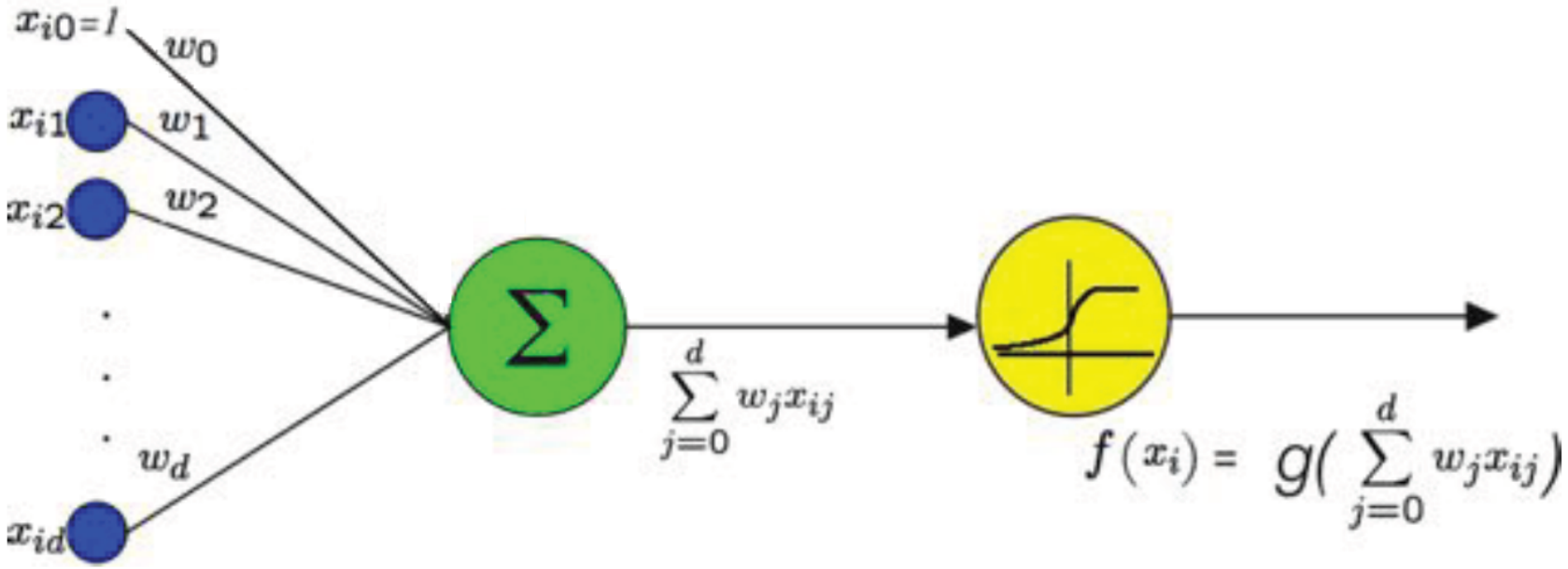
- Also, perceptron used an **activation function**, which is undifferentiable and not suitable for gradient descent (non-derivable) in case data is not linearly separable.
- We want a function whose input is a linear function of the data and whhose output is differentiable according to the data.
- One possibility is to use the sigmoid function :



$$g(z) = \frac{e^z}{1+e^z} = \frac{1}{1+e^{-z}}$$

$$g(z) \xrightarrow[z \to -\infty]{} 0 \quad \text{and} \quad g(z) \xrightarrow[z \to +\infty]{} 1$$

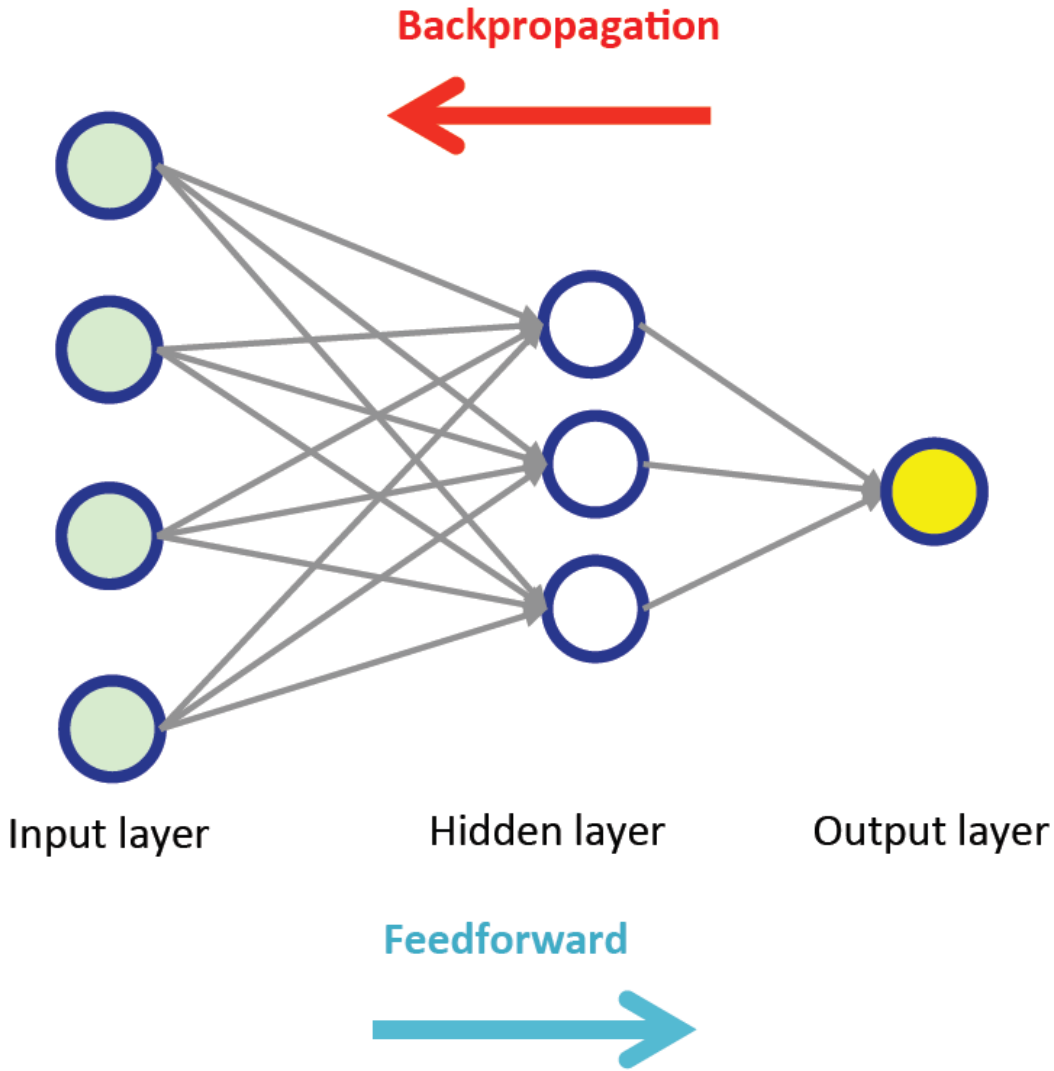
From perceptron to MLP



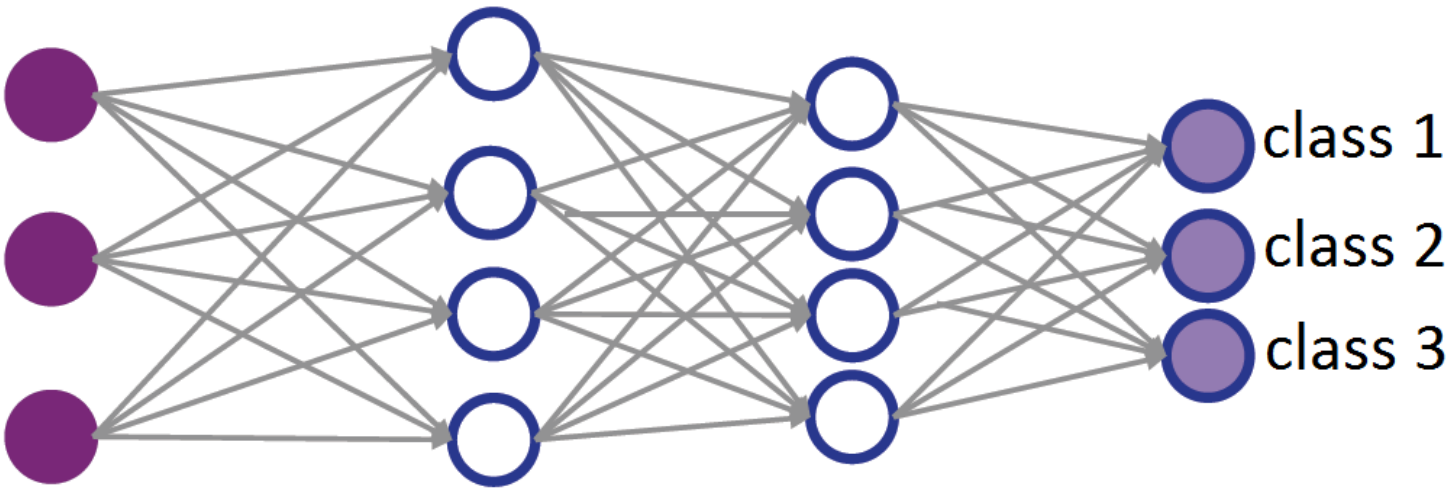
Given n examples and d features, for an example x_i (the i^{th} line in the matrix of examples) :

$$f(x_i) = \frac{1}{1 + exp\left(-\sum_{j=0}^{d} w_j x_{ij}\right)}$$

Feedforward-Backpropagation



Multi class case etc.



- Nowadays, networks with more than two layers, a.k.a. deep networks, have proven to be very effective in many domains.
- Examples of deep networks : restricted Boltzman machines, convolutional NN, auto encoders, etc.

Course syllabus

Supervised learning with R

- model fitting :

```
obj <- function_name(label ~ attribute1 + ... + attributeJ,
                     learning_data, hyperparameters)
```

or

```
obj <- function_name(label ~ ., learning_data, hyperparameters)
```

- prediction :

```
preds <- predict(obj, new_data)
```

- model evaluation :

```
acc <- mean(preds == new_data$label)
```

Models content

```
> lin_mod <- lm(Sepal.Length ~ Sepal.Width + Petal.Width, iris)
```

- simple call : *> lin_mod*

```
Call:
lm(formula = Sepal.Length ~ Sepal.Width + Petal.Width, data = iris)

Coefficients:
(Intercept)  Sepal.Width  Petal.Width
     3.4573       0.3991       0.9721
```

- *'summary'* R function : *> summary(lin_mod)*

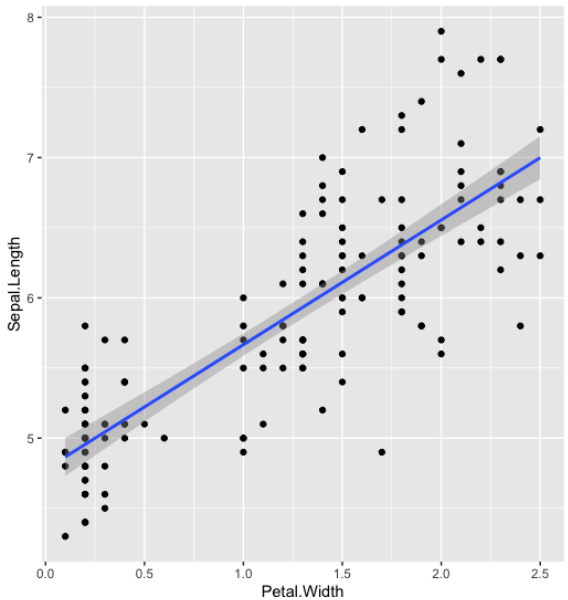
```
lm(formula = Sepal.Length ~ Sepal.Width + Petal.Width, data = iris)

Residuals:
    Min      1Q  Median      3Q     Max
-1.2076 -0.2288 -0.0450  0.2266  1.1810

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)  3.45733    0.30919   11.18  < 2e-16 ***
Sepal.Width  0.39907    0.09111    4.38 2.24e-05 ***
Petal.Width  0.97213    0.05210   18.66  < 2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.4511 on 147 degrees of freedom
Multiple R-squared:  0.7072,Adjusted R-squared:  0.7033
F-statistic: 177.6 on 2 and 147 DF,  p-value: < 2.2e-16
```

Interpretation through plots

```
> lin_mod <- lm(Sepal.Length ~ Sepal.Width + Petal.Width, iris)
```

- *simple* plots :

```
ggplot(iris, aes(x = Petal.Width,
                 y = Sepal.Length)) +
  geom_point() +
  geom_smooth(method = lm)
```



- *interpretable* plots :

```
tree <- rpart(Species ~ .,  iris)
rpart.plot(tree)
```