Algorithms for Data Analysis

Julien Tissier

January 18, 2018

Overview

1. Introduction to Data Analysis
   - Definition
   - Tools and libraries
   - Numpy
   - Pandas
   - Matplotlib
2. Machine Learning
   - Definition
   - Supervised vs. Unsupervised
   - Training, test & validation sets
   - Overfitting/Underfitting
3. Supervised Learning
   - k-Nearest Neighbors
   - Linear Models
   - Decision Trees
   - Support Vector Machine
   - Neural networks
4. Unsupervised Learning
   - Principal Component Analysis
   - Clustering

What is data analysis?

Data

Pieces of information (measurement, values, facts...) that can be :

- structured (matrices, tabular data, RDBMS, time series...)
- unstructured (news articles, webpages, images/video...)

Data Analysis

Process of preparing, transforming and using models to find more information from data, as well as visualizing results.

How to analyze data?

There are usually two steps in data analysis. The first one is to find and **develop models** that can extract useful information from data (with languages like R or MATLAB). The second one is to **develop programs** that can be used in production systems (with languages like Java or C++).

With a growing popularity among scientists as well as the development of efficient libraries (numpy, pandas), **Python** became a great tool for data analysis. Python has many advantages :

- great for string/data processing
- can be used for both prototyping/production
- has a lot of existing libraries
- can easily integrate C/C++/FORTRAN legacy code
- easy to read/develop

Libraries

This course will be based on Python 3.5 (or above) and the following libraries :

- IPython (6.2+) : enhanced Python shell
- numpy (1.13+) : fast/efficient arrays and operations
- pandas (0.20+) : data structures (Series/DataFrame)
- matplotlib (2.1.0+) : plots and 2D visualization
- scipy (0.19+) : scientific algorithms
- scikit-learn (0.19+) : machine learning algorithms

Jupyter Notebook will also be used to give you samples of code, as they provide a more interactive way to learn and discover how these libraries work.

Numpy

Numpy (**Num**erical **Py**thon) is a high performance scientific computing library that can be used for matrices computations, Fourier transforms, linear algebra, statistical computations...

The main type of data in Numpy is the **ndarray** :

- n-dimensional array
- fixed size
- homogeneous datatypes
- similar to C arrays (continuous block of memory)

Why is Numpy efficient?

As a high level language, Python is slow to do any heavy computations, especially if very large arrays are involved. Numpy solves this problem thanks to the ndarray datatypes :

- efficient memory management (continuous block)
- use C loops instead of Python loops for computations on array
- vectorized operations (computations are done block by block, not element by element)
- rely on low-level routines for some operations (BLAS/LAPACK)

Numpy

Example

```
import numpy as np

a = np.array([1, 2, 3, 4])
b = np.array([6, 7, 8, 9])

c = a * b

d = np.array([[1, 2], [3, 4]])
```

Pandas

Pandas is a high-performance Python library used to work with data and analyze them. It contains many pre-implemented methods to read and parse data, as well as common statistical computations (mean, variance, correlation...).

Pandas has two main datatypes:

- Series : one-dimensional container. Indexes can be integers (like an array) or other objects (string, date...)
- DataFrame : tabular data, like a spreadsheet. It contains multiple rows and multiple columns. It can be seen as a collection of Series.

Pandas

Example (Series)

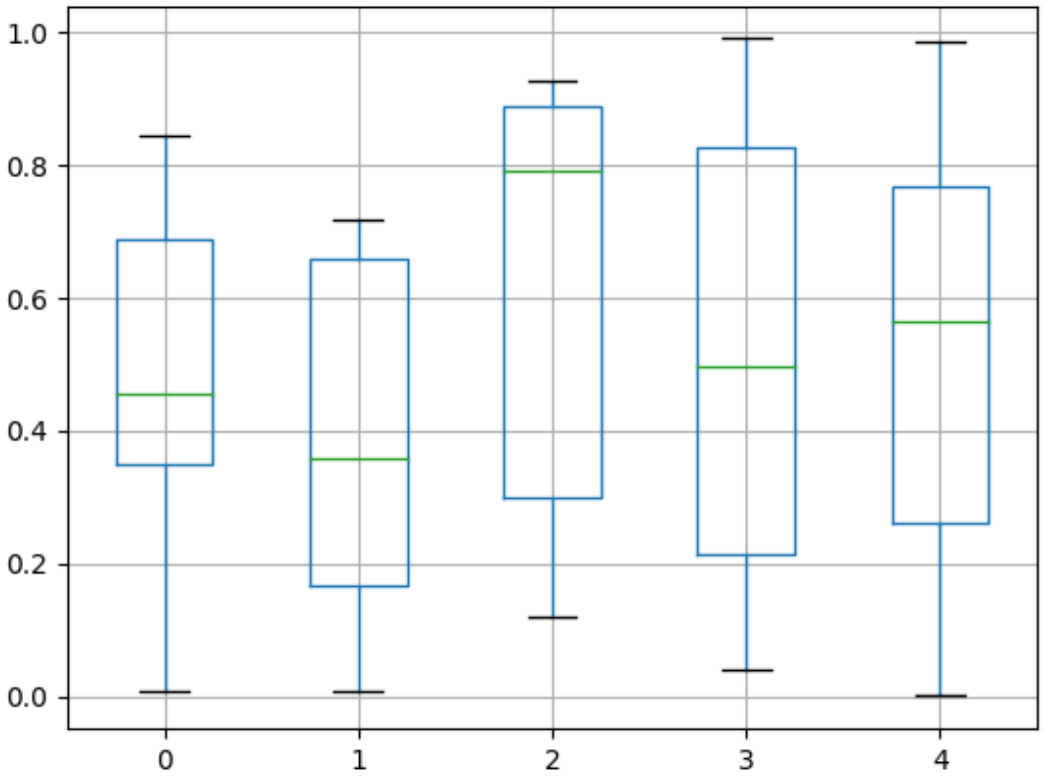
```
from pandas import Series

s1 = Series([4, 7, 9, -1])
s2 = Series([12, 3.2, "John"],
            index=["mark", "gpa", "name"])
```

Example (DataFrame)

```
from pandas import DataFrame
df = DataFrame({"key1": [1, 2, 3],
                "key2": [4, 5, 6]})
```

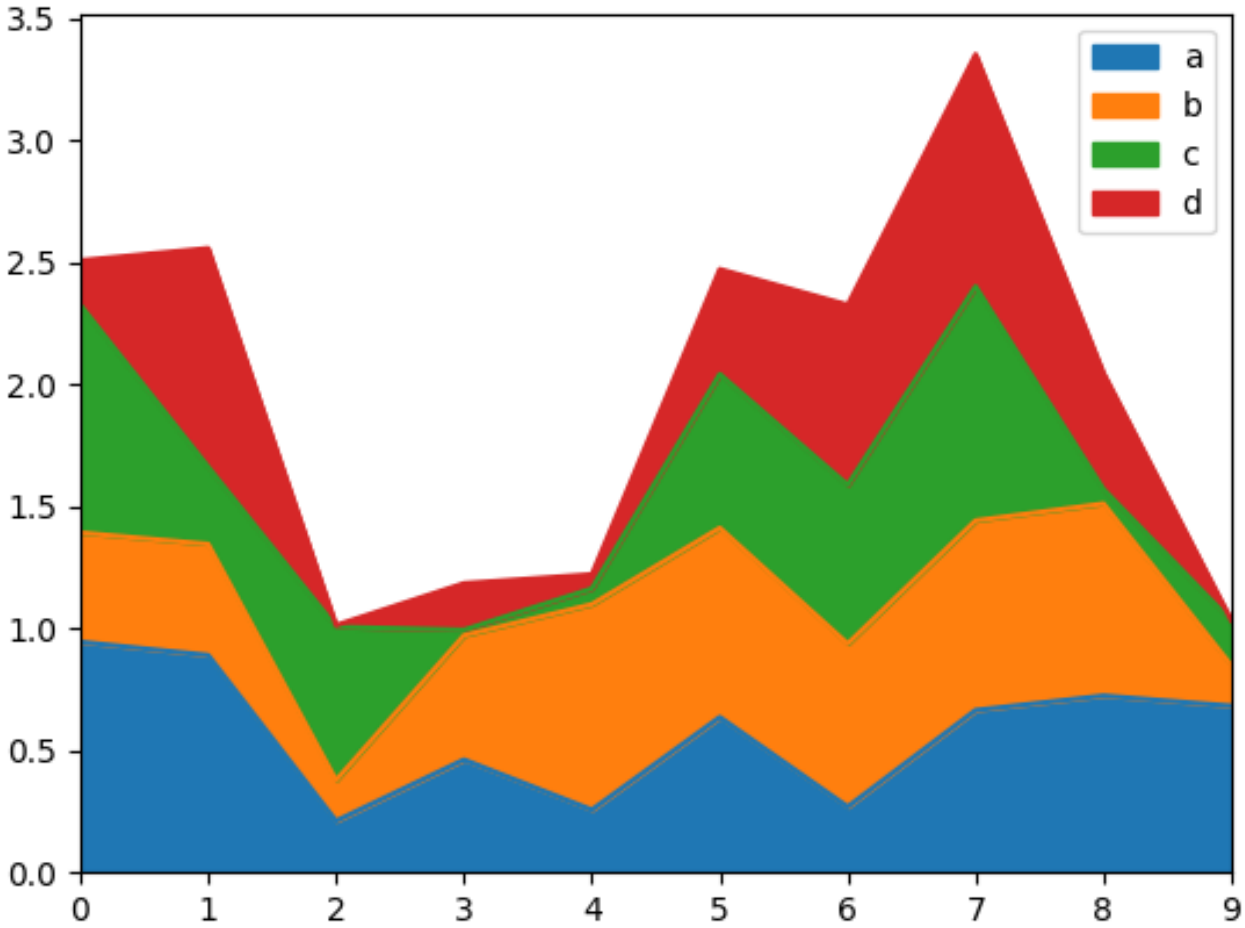
Matplotlib

Matplotlib is a plotting library used to visualize data and create graphics. Pandas directly uses matplotlib for representation.



Matplotlib

Matplotlib is a plotting library used to visualize data and create graphics. Pandas directly uses matplotlib for representation.



What is Machine Learning?

Machine Learning (ML)

ML is when a computational system can **automatically learn** and extract **knowledge from data** without being explicitly programmed. ML is at the intersection of statistics, artificial intelligence and computer science.
ML is now everywhere:

- automatic friend tagging in your Facebook photos
- music recommendation for Spotify/YouTube
- self-driven cars
- medical diagnosis in a hospital
- and many more...

ML can be **supervised** or **unsupervised**.

What is Machine Learning?

Brief history

At the beginning, many decision-making applications (like spam detection) were built with a lot of manually programmed if/else conditions. These methods have some drawbacks :

- it requires the **knowledge of an expert** to design the rules
- it is **very specific** to a task

With ML, you only need to feed a model with your data. The algorithm will find the rules by itself.

Supervised Learning

In supervised learning, you give the algorithm two kinds of data :

- **inputs** (an image, information about a person...)
- **outputs** (name of the object, the salary...)

The algorithm learns to find a way to **produce the output given the input**. The algorithm is then capable of producing the output of an input it has never seen before.

Supervised learning can be used to :

- recognize handwritten digits (input=image, output=digit)
- identify spams (input=mail, output=yes/no)
- predict a price (input=house infomation, output=price)

Supervised Learning

The inputs are in general real value vectors (sometimes it can be binary vectors). Each dimension of this vector is called a **feature**.

There are two main problems that can be solved with supervised learning :

- **classification** : the output is the class the item belongs to. It can be a binary classification problem (yes/no) or a multi-class classification problem (class 1, class 2, class 3...)
- **regression** : the output is a real value number

Unsupervised Learning

In unsupervised learning, there are no known output (or label), so you only give the algorithm the inputs. For example, if you want to regroup similar clients according to their customer habits, you do not know in advance what are the groups.

The main problems that can be solved with unsupervised learning are :

- **clustering**
- **dimensionality reduction**

The main difficulty of unsupervised learning algorithms is to **evaluate them**. Since we do not have any labels, there are not any "true" valid answer. Most of the time, we need to manually evaluate if the output of the algoritm is relevant. Sometimes, unsupervised learning algorithms are used as a **pre-processing** step before supervised learning algorithms and can lead lead to a more accurate model or a faster training.

How to evaluate a ML algorithm?

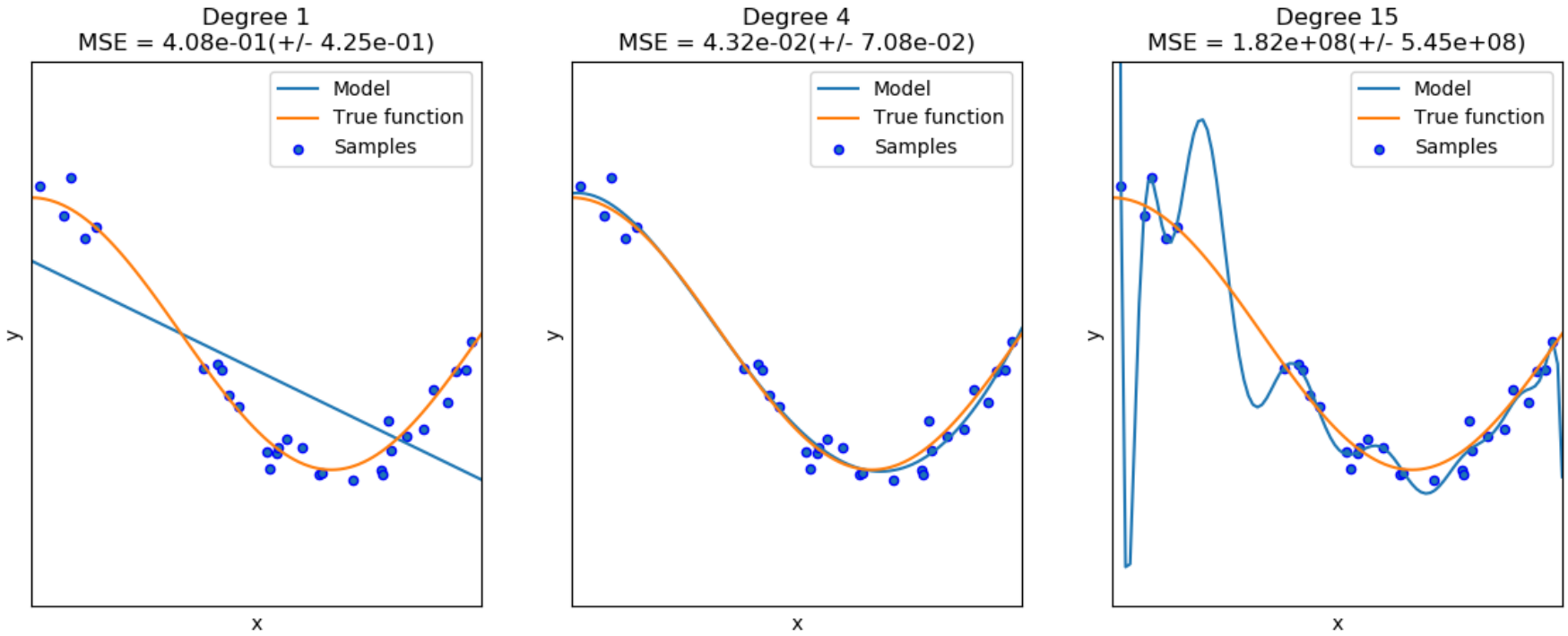
After training a ML algorithm with data, we need to evaluate its performance. One way would be to feed the algorithm with the same data, compute the output and compare it with the original output.

This method is bad because it does not tell us the capacity of the algorithm **to generalize** (is the algorithm able to perform well on unseen data ?). One way to solve this problem is to separate the data into 3 sets :

- training set : used to train the algorithm
- validation set : used to adjust the algorithm
- test set : used to measure the performance of the algorithm

Problems of ML

- **underfitting** : when the model is too simple. The training score and the test score are both bad (left)
- **overfitting** : when the model is too complex (right). The training score is good but the test score is bad. The model is not able to generalize well.

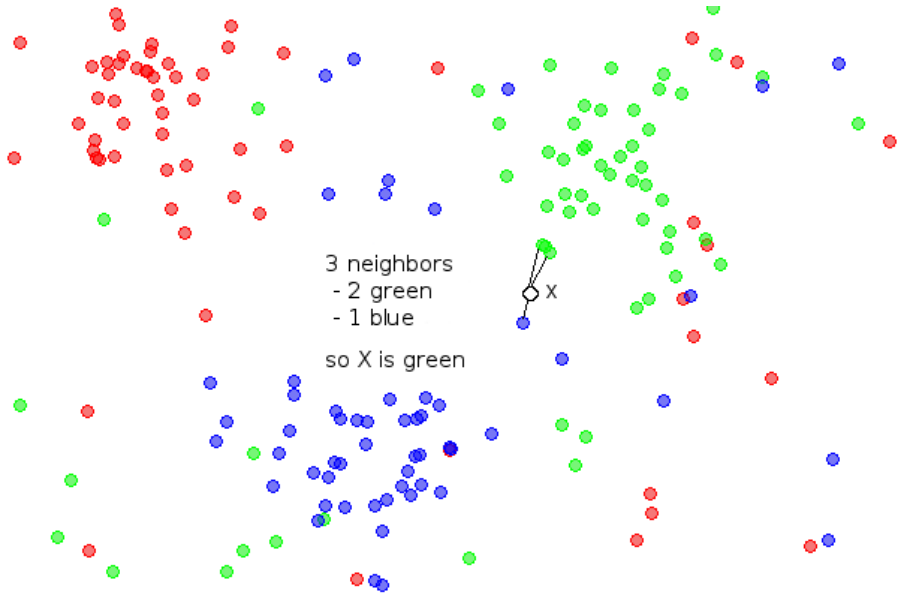


k-Nearest Neighbors - Classification

Inputs : set of training points $\mathcal{S}$, point X to classify
Procedure :

- find the k closest points (the neighbors) to X from $\mathcal{S}$
- find the most frequent class c among the neighbors
- assign c to X

The k closest points are defined as the points with minimal distance (usually the Euclidean distance) to X. If there is a tie in finding c, we can increase/decrease k until the tie is broken.



k-Nearest Neighbors - Regression

Inputs : set of training points $\mathcal{S}$, point X to predict value
Procedure :

- find the k closest points (the neighbors) to X from $\mathcal{S}$
- compute the average value v of all neighbors
- assign v to X

If k is set to 1, the value assigned to X is simply the value of its nearest neighbor.

k-Nearest Neighbors

The k-nearest neighbors is one of the simplest model in ML. It usually has two main parameters : **the number of neighbors** used and **the distance** used to find the closest points.

Pros

- model is easy to understand
- does not require extended tuning to achieve good performance
- works well as a baseline before training more complex models

Cons

- slow when many training points or many features
- not very good on sparse data

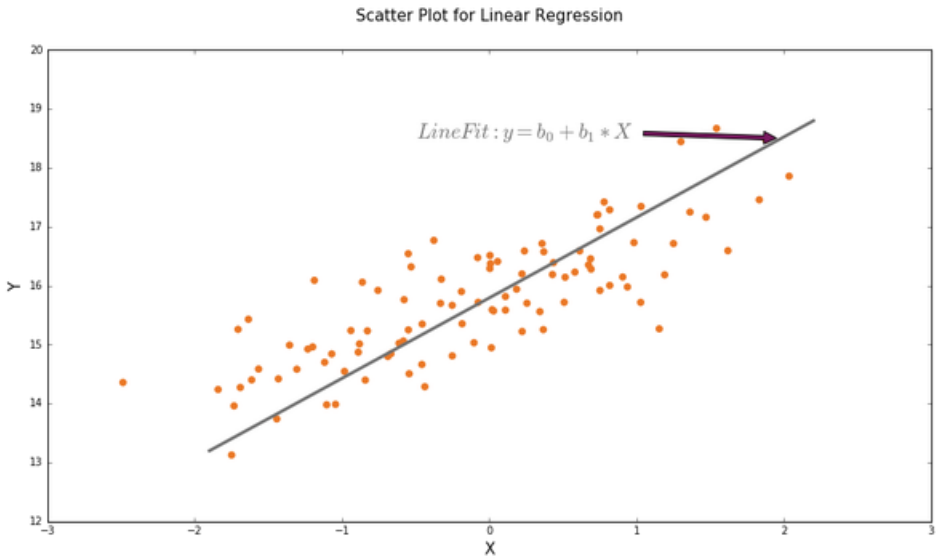
Linear Models - Regression

Linear models are usually used to make values prediction. The output $\hat{y}$ is a **linear function** of each features x_i of a given input vector X (hence the name of **linear** models).

$$\hat{y} = \sum_{i=1}^{k} w_i * x_i + b$$

The model learns the coefficients w_i and b. The model is trained to minimize the **Mean Squared Error (MSE)** on all training points X in $\mathcal{S}$ (where y is the correct value of X).

$$MSE = \frac{1}{\#\mathcal{S}} \sum_{X \in \mathcal{S}} (\hat{y} - y)^2$$



Linear Models - Regression

- Training

  Inputs : set of training points $\mathcal{S}$

  Procedure :
  - compute the MSE
  - compute the partial derivatives $\frac{\partial MSE}{\partial w_i}$ and $\frac{\partial MSE}{\partial b}$
  - solve the linear system given by the equations $\frac{\partial MSE}{\partial w_i} = 0$ and $\frac{\partial MSE}{\partial b} = 0$
  - the solution is unique and gives you the w_i and b

  Output : learned w_i and b

- Predicting

  Inputs : learned w_i and b, point X to predict value

  Procedure : compute $\hat{y}$ with the formula and w_i, b, x_i

  Output : predicted value $\hat{y}$

Linear Models - Classification

Linear models can also be used for either binary or multi-class classification problems. For binary classification (1 or 0 labels), the model evaluates the output with :

$$\hat{y} = \begin{cases} 1 & \text{if } w \cdot x + b \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

We can rewrite the condition to remove the bias b with $\sum_{i=0}^{k} w_i * x_i \geq 0$ and set x_0 to 1 for all vectors X. The objective of the model is to find a good w to predict the correct label (such that $\hat{y} = y$). Different methods exist to find w :

- perceptron
- logistic regression
- support vector machines (SVM)

Linear Models - Perceptron

We can define a loss ℓ (called the zero-one loss) for each point (x, y) from our training set as follows :

$$\ell(x, y, w) = 1_{[(w \cdot x)y \leq 0]}$$

The perceptron algorithm learns the w weights as follows :

- initialize w to 0
- **for** step in [1..n]
  - **for** x in training dataset

    if $(w \cdot x) \times y \leq 0$

    $\mathrm{w} = \mathrm{w} + \lambda\ \mathrm{y} \cdot \mathrm{x}$

The perceptron algorithm only updates the weights w when it encounters a misclassified example. If the dataset is **linearly separable**, the perceptron algorithm is guaranteed to find an optimal solution.

Linear Models - Logistic Regression

Instead of directly finding the output $\hat{y}$ with the sign function, we can compute the **probability of belonging to class 1** defined as follows :

$$P(y = 1|x) = \sigma(w \cdot x) = \frac{1}{1 + e^{-w \cdot x}} \in [0, 1]$$

The idea is to find w such that this probability is high (near 1) for points labeled with 1, and low (near 0) for points labeled with 0. We can define another loss function (called negative log-likelihood) :

$$\ell(x, y, w) = -y \times \log(\sigma(w \cdot x)) - (1 - y) \times \log(1 - \sigma(w \cdot x))$$

The model is trained to minimize the loss for all training examples. Since this loss function is convex, we can use different algorithms like gradient descent, L-BFGS...

Linear Models

Linear models have two main parameters : **the weight of regularization** and **the type of regularization** used.

Pros

- models are fast to train and predict
- can be used with large datasets (scalability)
- works well on sparse data

Cons

- learned coefficients are not very interpretable
- not very good at detecting correlated features

Decision Trees

Decision trees are mostly used for classification problems. They learn a **sequence of conditions** (if/else questions) to find the class of a vector x. Let's suppose we have the following dataset and we want to predict if the game can be played (9 yes / 5 no) :

	Outlook	Humidity	Wind	Play
1	Sunny	High	Weak	No
2	Sunny	High	Strong	No
3	Overcast	High	Weak	Yes
4	Rain	High	Weak	Yes
5	Rain	Normal	Weak	Yes
6	Rain	Normal	Strong	No
7	Overcast	Normal	Strong	Yes
8	Sunny	High	Weak	No
9	Sunny	Normal	Weak	Yes
10	Rain	Normal	Weak	Yes
11	Sunny	Normal	Strong	Yes
12	Overcast	High	Strong	Yes
13	Overcast	Normal	Weak	Yes
14	Rain	High	Strong	No

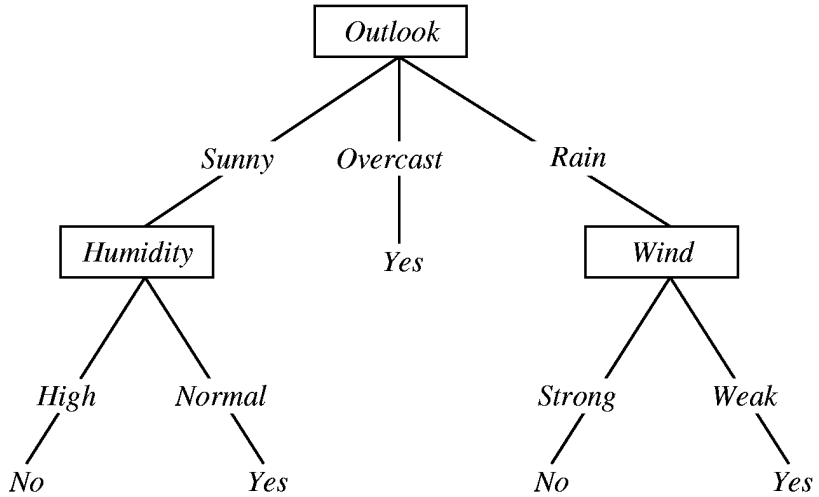
Decision Trees

Many algorithms exist to create a decision tree. One of them is ID3. At each step, it computes the entropy of each unused attributes with :

$$Entropy(S) = \sum_{x \in X} -p(x) \log_2 p(x)$$

where $p(x)$ is the ration between the number of elements in x and in S. The attribute that has the lower entropy is chosen as a node in the tree.

	Outlook	Humidity	Wind	Play
1	Sunny	High	Weak	No
2	Sunny	High	Strong	No
3	Overcast	High	Weak	Yes
4	Rain	High	Weak	Yes
5	Rain	Normal	Weak	Yes
6	Rain	Normal	Strong	No
7	Overcast	Normal	Strong	Yes
8	Sunny	High	Weak	No
9	Sunny	Normal	Weak	Yes
10	Rain	Normal	Weak	Yes
11	Sunny	Normal	Strong	Yes
12	Overcast	High	Strong	Yes
13	Overcast	Normal	Weak	Yes
14	Rain	High	Strong	No



Decision Trees

Decision trees have many parameters to control their size : maximum depth, number of leaf, acceptable ratio in a node... They provide a strong understandable tool for people not familiar with the ML field.

Pros

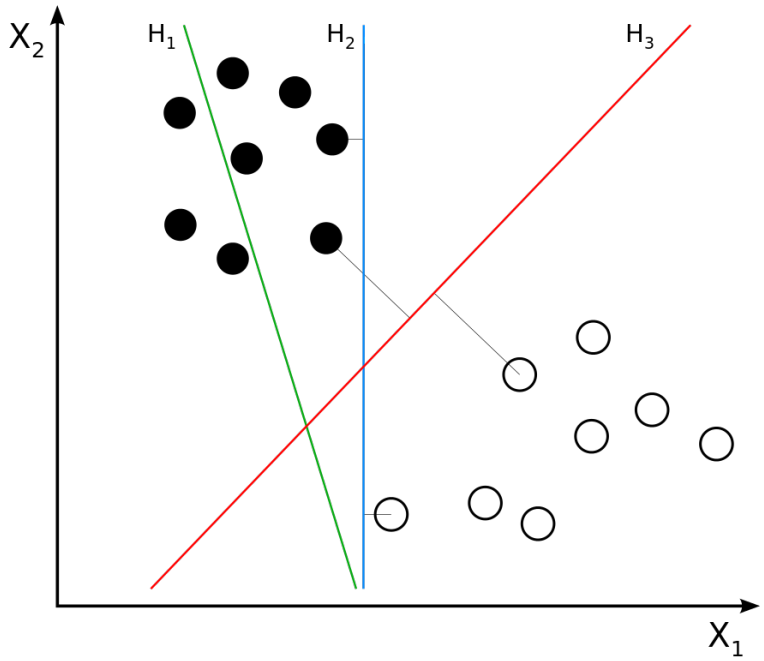
- can visually be understood
- can be used with large datasets
- no preprocessing needed (normalization of features)

Cons

- tend to easily overfit
- poor generalization performance

Support Vector Machine

Support Vector Machine (SVM) is an algorithm used to solve **classification problems**. The main goal is to find the **optimal hyperplane**.

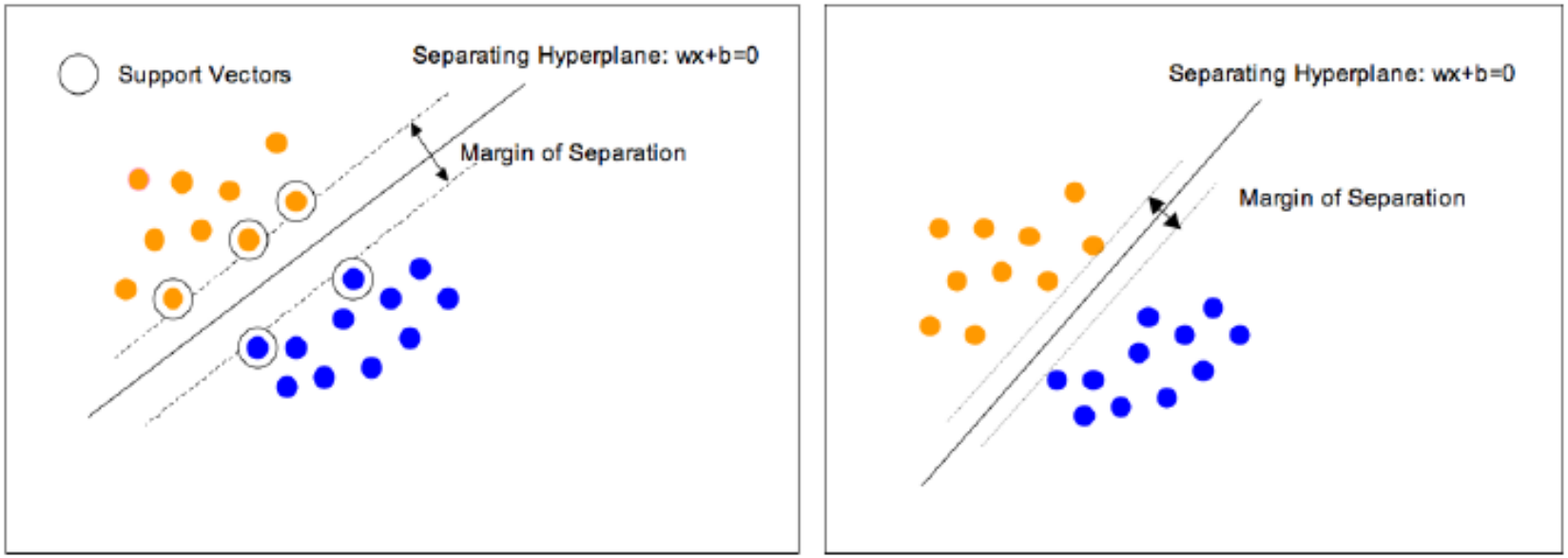


- H_1 is not correct
- H_2 is correct, but not the best
- H_3 is correct and optimal

What is the optimal hyperplane?

If the training data is linearly separable, an infinite number of correct hyperplanes can be found. So we need to discriminate them to choose the optimal one. We introduce the notion of **margin** as the distance between the hyperplane and the closest points from training set.

The **optimal hyperplane** is the one with the **largest margin**. The data points on the margin are called **support vectors**.



Finding the optimal hyperplane

The equation of an hyperplane is:

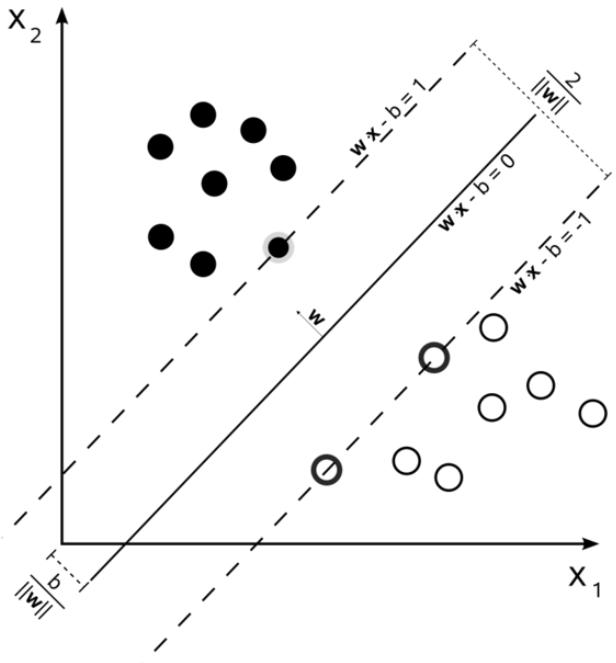
$$\vec{w} \cdot \vec{x} - b = 0 \qquad (1)$$

The equations of the hyperplanes representing the margins are:

$$\begin{cases} \vec{w} \cdot \vec{x} - b = +1 \\ \vec{w} \cdot \vec{x} - b = -1 \end{cases} \qquad (2)$$

The distance between the margins and the optimal hyperplane is:

$$\frac{1}{||w||^2} \qquad (3)$$



Finding the optimal hyperplane

The main goal of SVM is to **maximize the margin**, this means to **minimize the value of** $||w||^2$.
We also want to correctly classify the points w.r.t. the margin (i.e. no points are between the two hyperplanes defining the margin). This means:

$$\begin{cases} \vec{\boldsymbol{w}} \cdot \vec{\boldsymbol{x}} - b \geq +1 \text{ for labels} = +1 \\ \vec{\boldsymbol{w}} \cdot \vec{\boldsymbol{x}} - b \leq -1 \text{ for labels} = \text{-1} \end{cases} \tag{4}$$

This can be rewritten as:

$$y \times (\vec{\boldsymbol{w}} \cdot \vec{\boldsymbol{x}} - b) \geq +1 \tag{5}$$

So the SVM solves (with the **Lagrangian multiplier** method):

$$\min_{w,b} ||w||^2 \quad s.t. \quad y_i \times (\vec{\boldsymbol{w}} \cdot \vec{\boldsymbol{x_i}} - b) \geq +1 \quad \forall\ i \tag{6}$$

Soft margin

The assumption that our dataset is linearly separable is strong and not true for most of the cases. We can create a new model that will find the optimal hyperplane, but do not try to correctly classify every training point. We introduce the **hinge loss** as:

$$\max(0, 1 - y_i \times (\vec{\boldsymbol{w}} \cdot \vec{\boldsymbol{x_i}} - b)) \tag{7}$$

The problem to optimize becomes:

$$\min_{w,b} \left(||w||^2 + \lambda \sum_{i=1}^{n} \max(0, 1 - y_i \times (\vec{\boldsymbol{w}} \cdot \vec{\boldsymbol{x_i}} - b)) \right) \tag{8}$$

Support Vector Machine

SVM finds the optimal hyperplane by maximizing the margin. With hard margin, no tuning is required but it does not work if the data is not linearly separable. With soft margin, there is always a correct solution and the model is more robust to outliers.

Pros

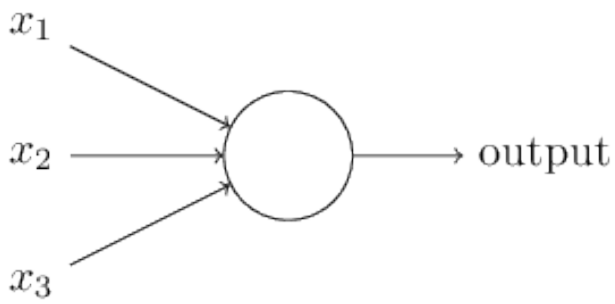
- work well on high or low dimension data
- allow very complex boundaries (with kernel trick)

Cons

- do not scale well
- require preprocessing and tuning
- hard to inspect and understand

Neural Networks

A neural network is a model used for either **classification** or **regression** problems. The main unit of a neural network is called a **neuron** :

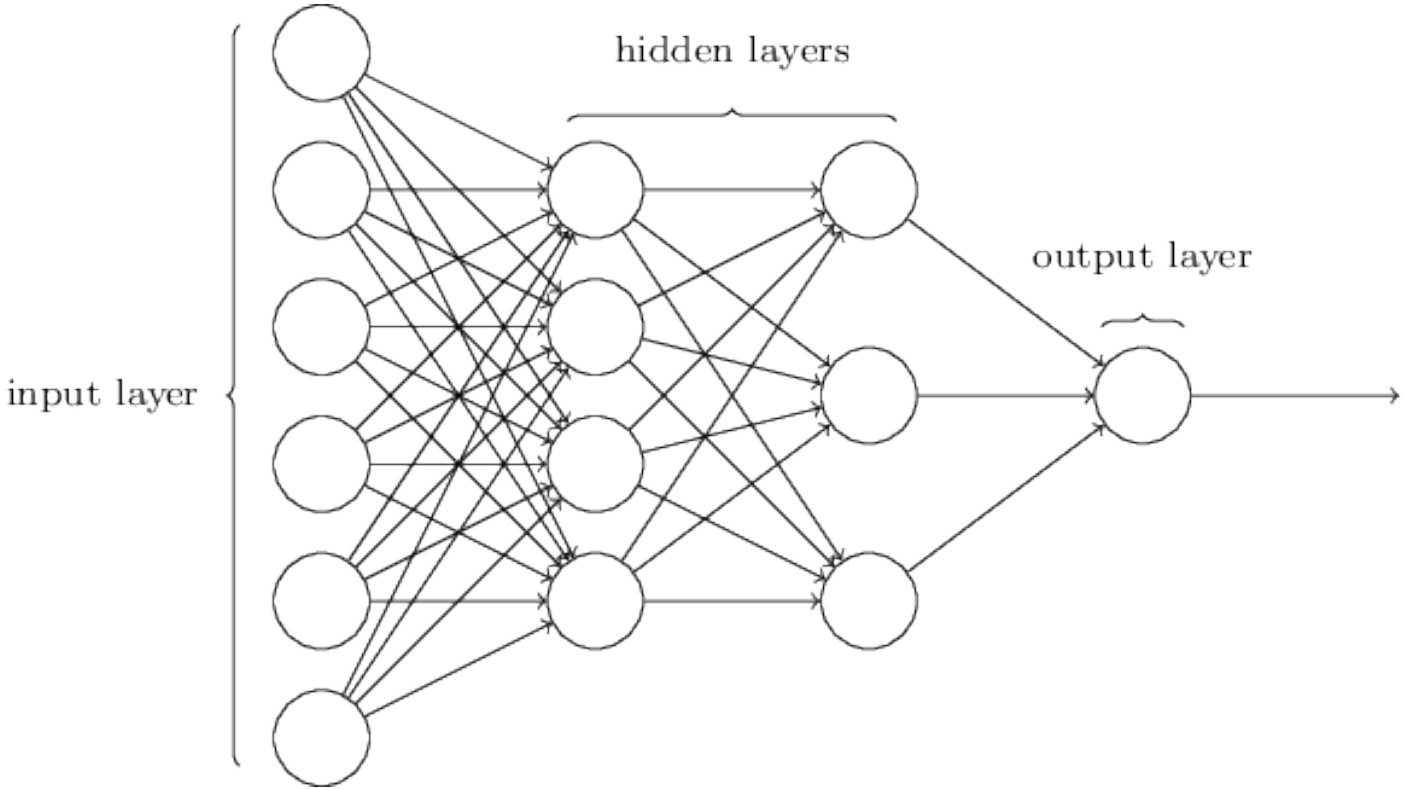


Each neuron has a weight vector w and a bias b. Its output is:

$$y = f(\boldsymbol{w} \cdot \boldsymbol{x} + b) \tag{9}$$

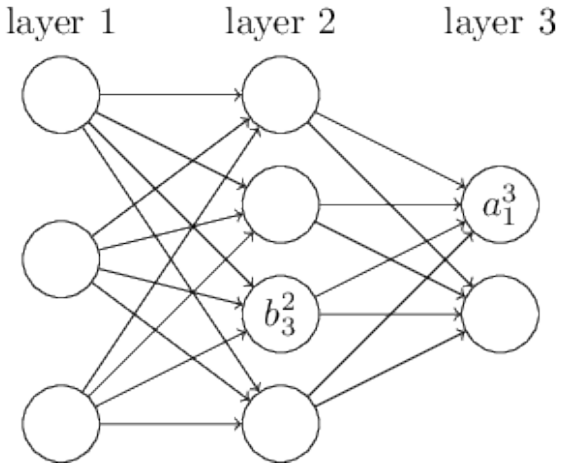
Neural Networks

When we combine several neurons, we obtain a neural network :



Each neuron from one layer is connect to each neurons from the following layer. We call this the **fully connected** architecture.

Training neural networks



The goal is to be able to predict the output given the input. We can compute the cost for one training example as the difference between the predicted output and its label :

$$C_i = \frac{1}{2}||\boldsymbol{y_i} - a^l(\boldsymbol{x_i})||^2 \tag{10}$$

Training neural networks

We want to minimize the cost C_i. We can use the **gradient descent** to do so. We compute the partial derivative of C_i w.r.t. to the weights and biases of the last layer.

$$\frac{\partial C_i}{\partial w_{jk}^l} \quad and \quad \frac{\partial C_i}{\partial b_j^l} \tag{11}$$

And then update the weights and biases with :

$$w_{jk}^l = w_{jk}^l - \lambda \times \frac{\partial C_i}{\partial w_{jk}^l} \tag{12}$$

$$b_j^l = b_j^l - \lambda \times \frac{\partial C_i}{\partial b_j^l} \tag{13}$$

Training neural networks

We repeat the same process on the layer before the last one, and again on the layer before and again until the input layer. Since we are moving backward, we are calling this the **backpropagation**.

We repeat this process for each training examples. Once we have seen all training examples, we call this an **epoch**. We repeat the backprogation algorithm until a certain number of epoch is done, or until the network has a good accuracy (or a cost below a threshold).

Neural Networks

Neural network is a really big subject and we only cover a fraction of it in this course. Many more implementations exist (convolutional neural network, deep network, recurrent neural network...). Neural networks have been proved to be able to compute any function (so in theory, it can always predict the right output given the input).

Pros

- can capture information in large datasets
- can build complex models

Cons

- slow to train (if deep)
- require preprocessing (homogeneous data)
- hard to tune

Representing data

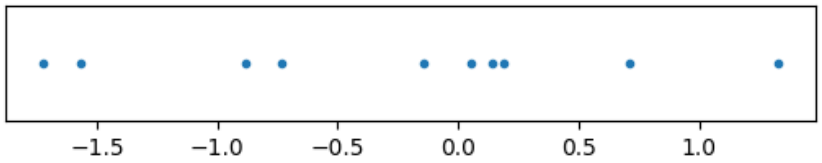


Figure: 1 dimension

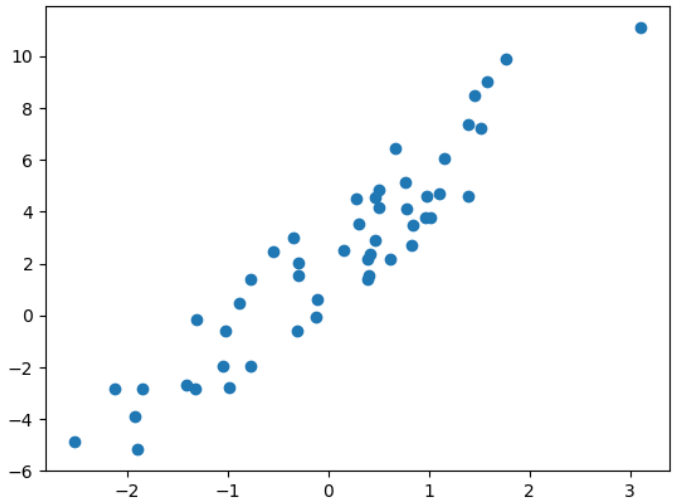


Figure: 2 dimensions

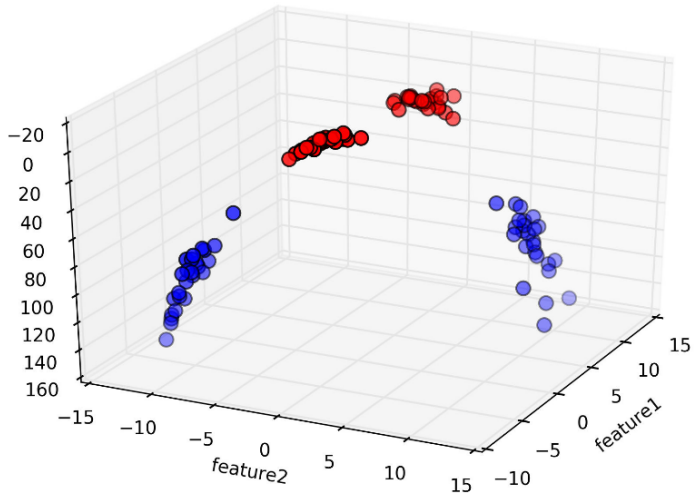


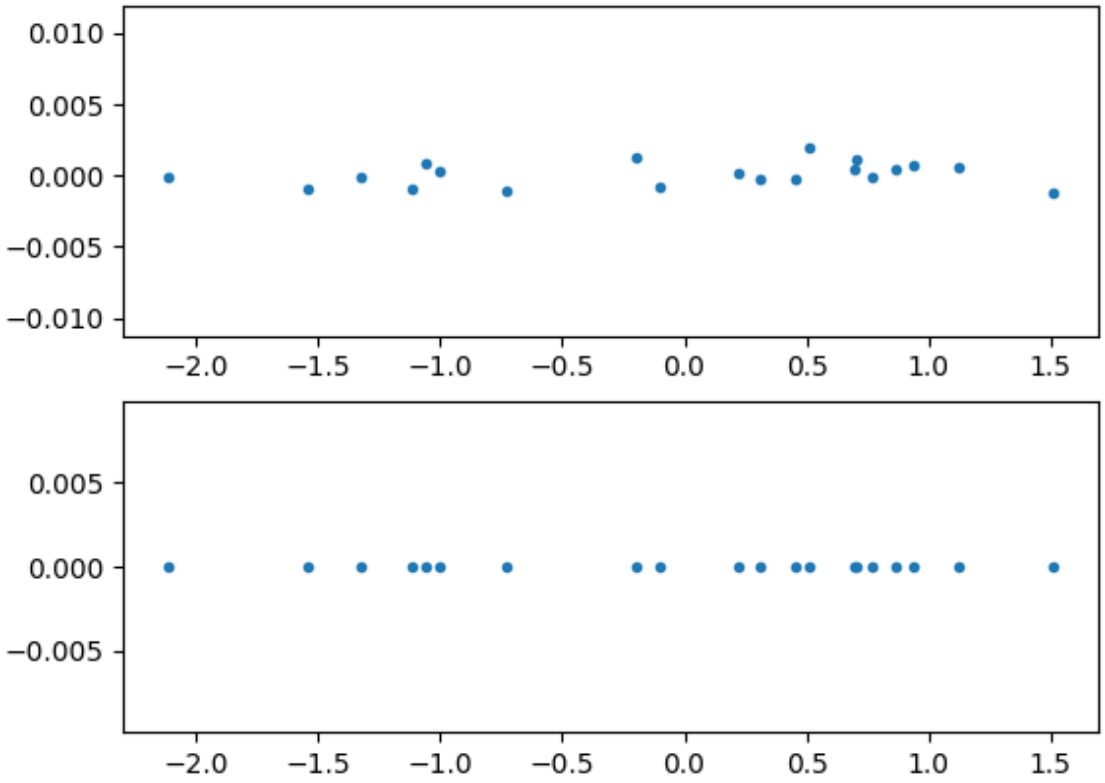
Figure: 3 dimensions

4 dimensions ?

n dimensions ?

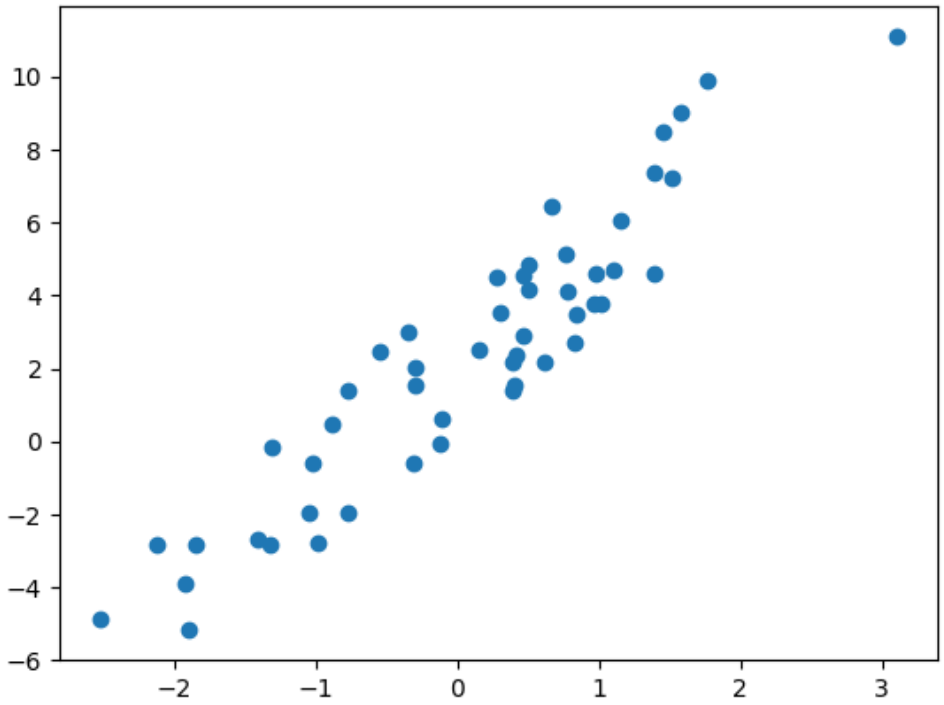
Reducing the dimension

To visualize data, we need to project it into a 1, 2 or 3 dimensional space (because we can not draw a representation if there are more than 3 dimensions). So we **need to reduce the number of dimensions**. One way is to remove unimportant dimensions.



Finding the important dimensions

We consider a feature to be important if **it has a lot of variations**. Indeed, if all the examples share almost the same value for a certain features, we can not extract useful information from this feature. So the most important dimensions are the ones that **maximize the variance**.



Principal Component Analysis

PCA finds the directions that have the most variations. If the vectors have n dimension, PCA will find n directions. The directions found are ranked according to their variance, so first direction has the most variations, second one has the second most variations...

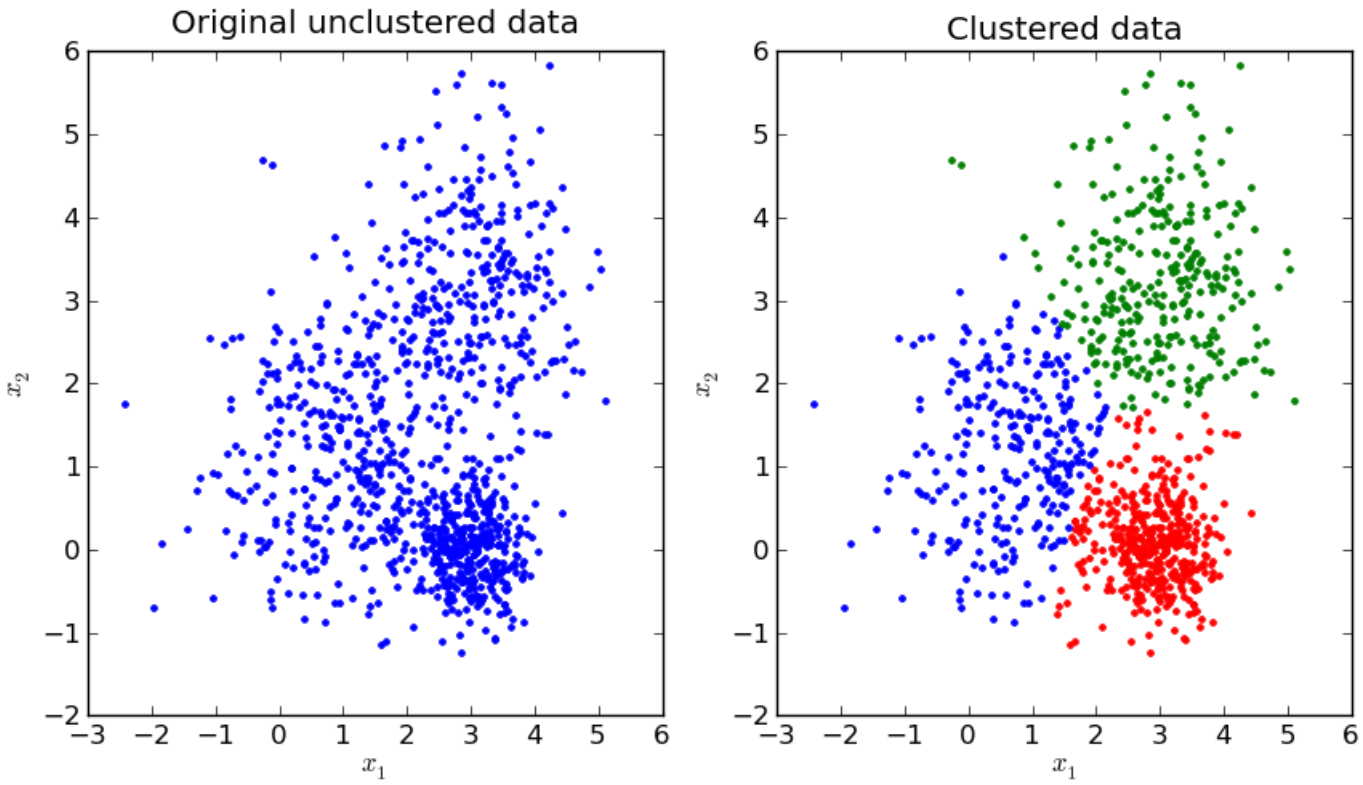
These directions are called **principal components (PC)** : PC1 has the most variations, PC2 the second most variations... All principal components are **orthogonal** to each other. So PCA is like "finding a new set of axis" to represent the data. Then, by only selecting the 2 or 3 first PC, we can draw our dataset into a space where we can visualize our data.

We can compute the PC with :

- eigenvalue decomposition of the covariance matrix
- singular value decomposition

Clustering

Clustering is a method to divide a dataset into a certain number of **clusters**. A cluster is a group of points that have minimal distance between them. Below is an example with 3 clusters.



K-means clustering algorithm

K-means is one fast algorithm that can be used to separate a dataset into k clusters with complexity $\mathcal{O}(k * m * d)$.

Inputs : set of n data points $\mathcal{X}$, number of clusters k
Procedure :

- select k points at random. They form the **centroids** $c_i \in C$
- **while** centroids do not converge
  - **for** x_i in $\mathcal{X}$
    - find nearest centroid c_j
    - assign x_i to cluster C_j
  - **for** each cluster C_j
    - define new centroid c_j as $\frac{1}{n_j} \sum_{x_i \in C_j} x_i$