Computational foundations of Data Sciences

M2 Maths-Info (S1)

Théo LACOMBE
theo.lacombe@univ-eiffel.fr
4B182

Last compilation : December 12, 2024

Organization of the course:

- 8×3h (including break).
- Expected: 6 lectures (including exercises), 2 lab session.

Grading:

- Project (/8).
- Exam (/12).
- **Bonus:** +0.1 pts for each typo reported (send me an email). Max +1 pts.

Material: On `elearning` (will be used as the main communication channel).

Lab sessions: With `Python` via `notebook Jupyter`. You can bring your own laptop.

Disclaimer: Some illustrations are taken from a course I am teaching in French and thus may have a French caption/legend... I will try to improve on this overtime; this does not count as a typo.
Also, I will not print the slides (and discourage you from doing so): 200+ pages with many typos...

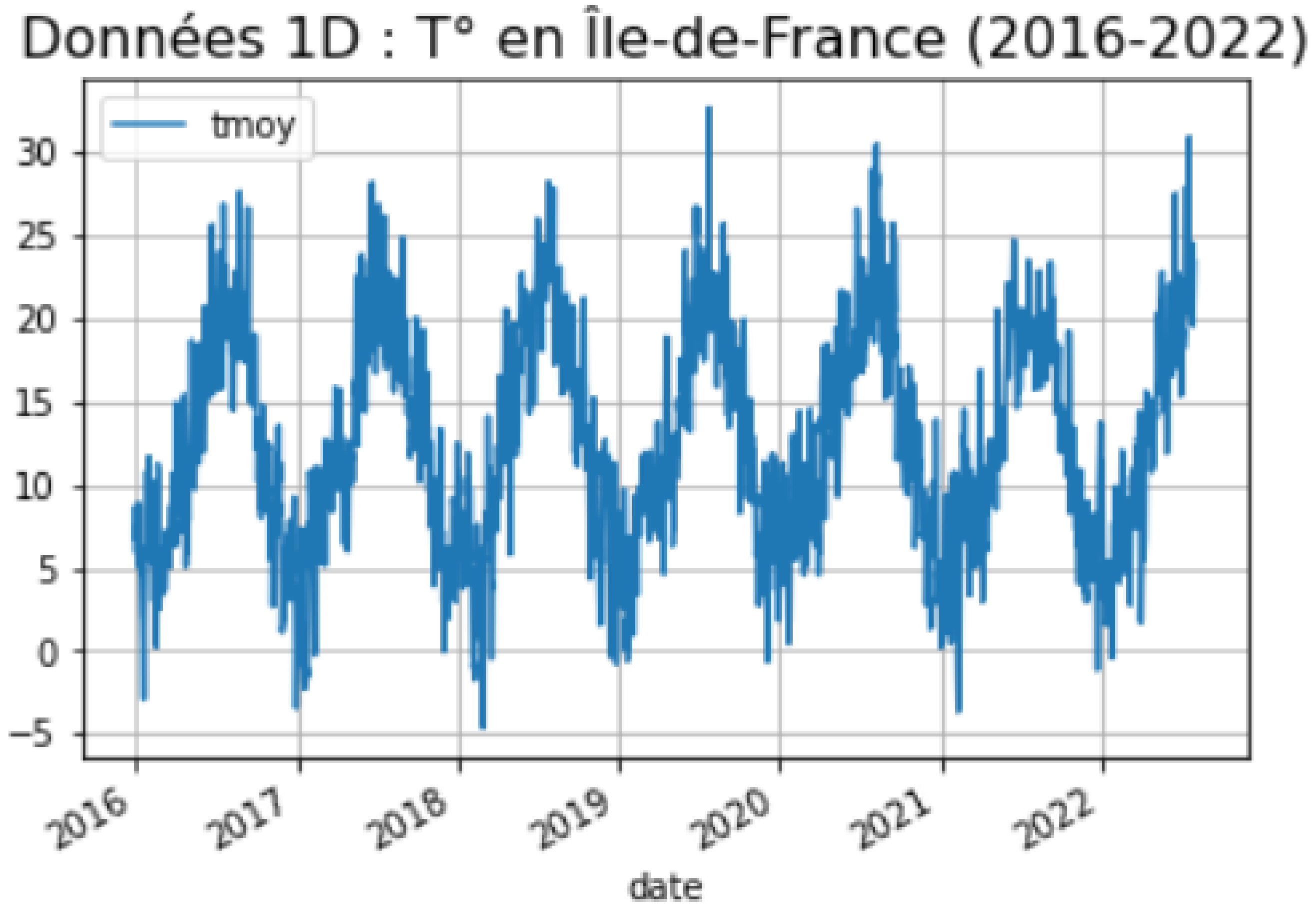
Outline:

- **Chapter 0: Generalities.**
- **Chapter 1: Some practical tools.**
- **Chapter 2: Supervised learning (1).**
- **Chapter 3: An optimization detour.**
- **Chapter 4: Supervised-learning (2) : classification.**
- **Chapter 5: Unsupervised-learning.**
- **Chapter 6: Kernel methods.**

Chapter 0: Generalities and some terminology

Introduction : A data (datum?) is a piece of information recorded by a biological or artificial system. Data can appear through different forms:

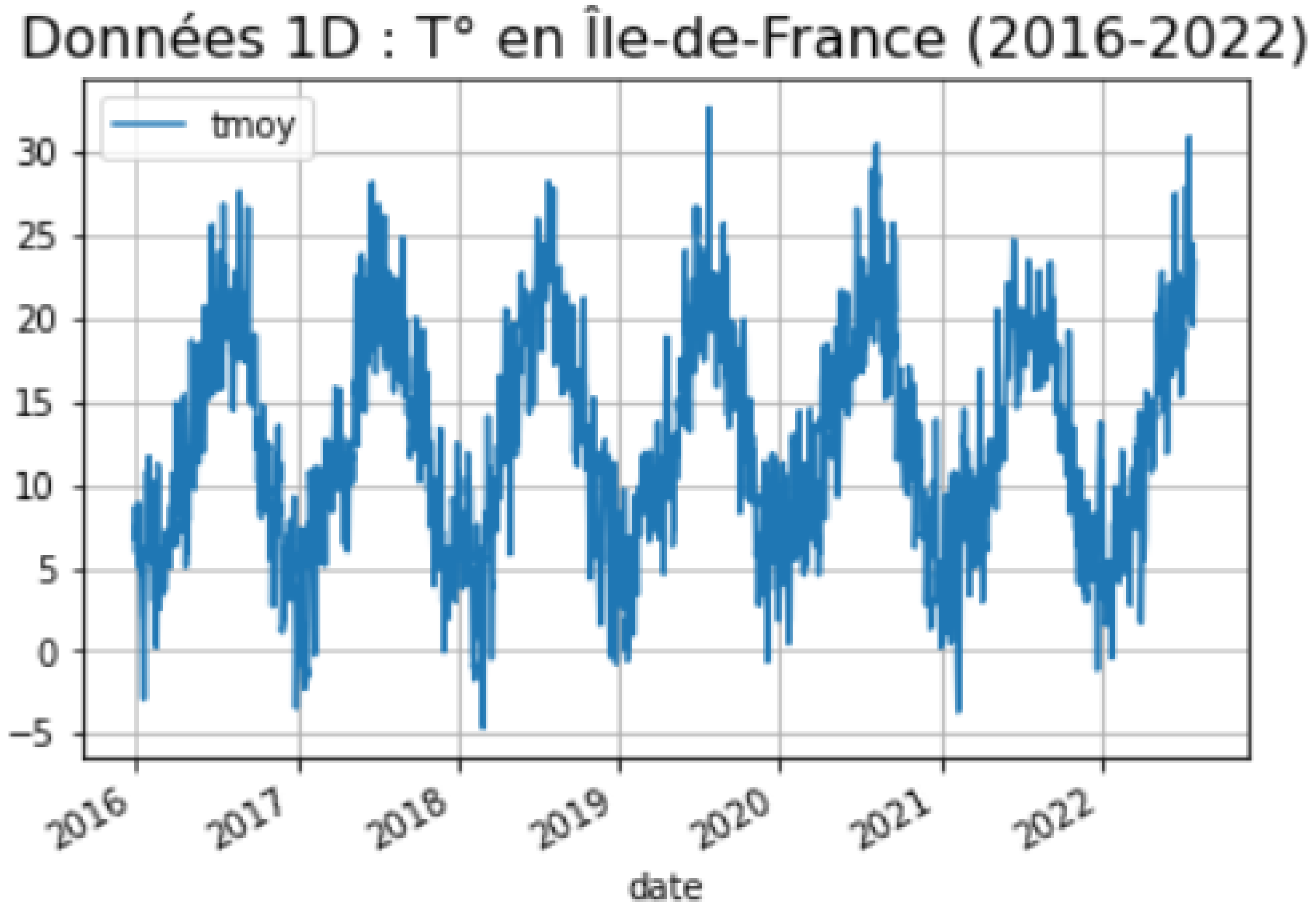
- A single number : heat (e.g. $T = 38°C$), height of someone (e.g. $h = 178$cm), binary variable (e.g. 1 if someone has a driver license, 0 otherwise), etc.



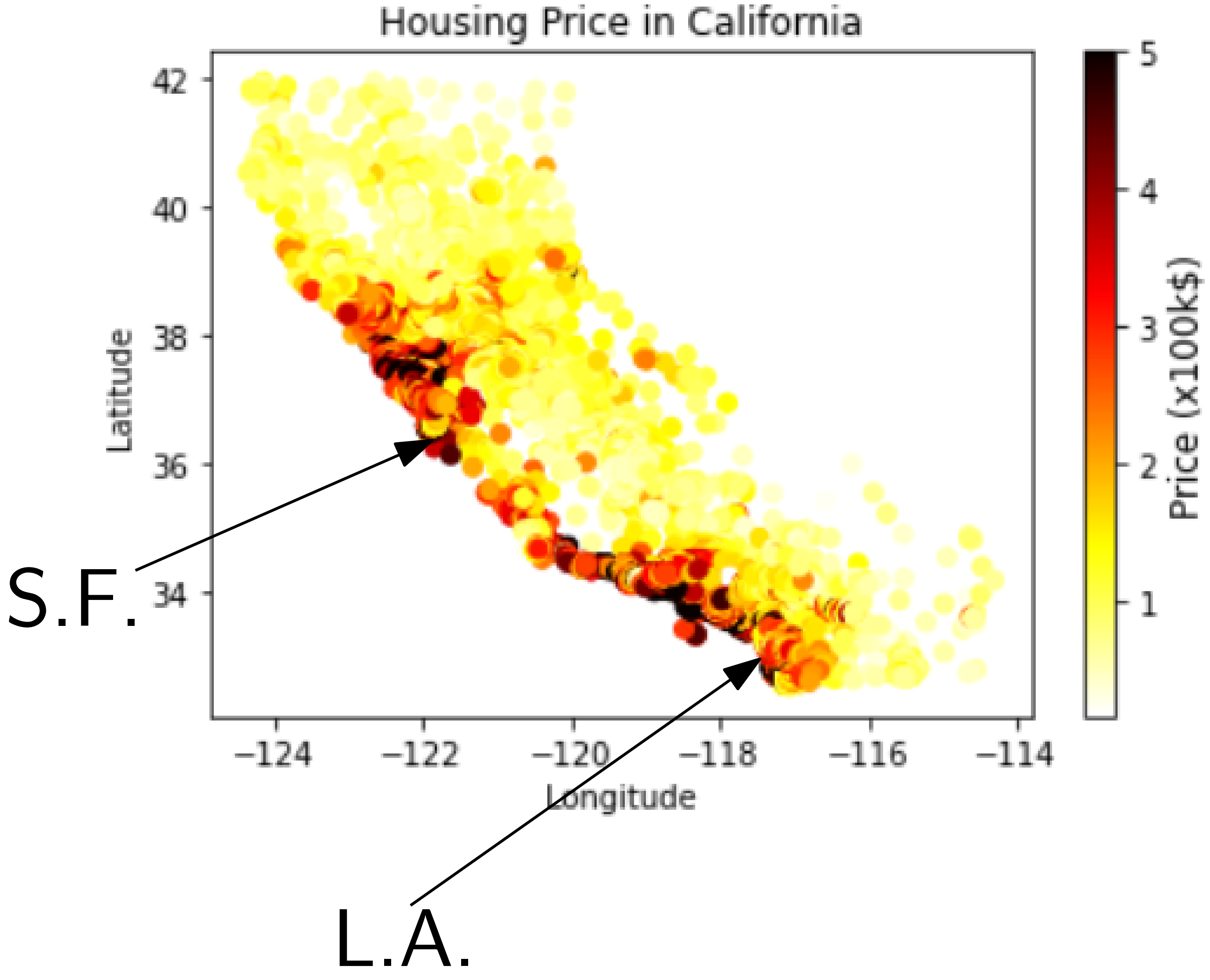
Chapter 0: Generalities and some terminology

Introduction : A data (datum?) is a piece of information recorded by a biological or artificial system. Data can appear through different forms:

- A single number : heat (e.g. $T = 38°C$), height of someone (e.g. $h = 178$cm), binary variable (e.g. 1 if someone has a driver license, 0 otherwise), etc.
- A collection of coordinates: GPS location (x, y) of some place, description (taille, poids, âge) of an individual, etc.



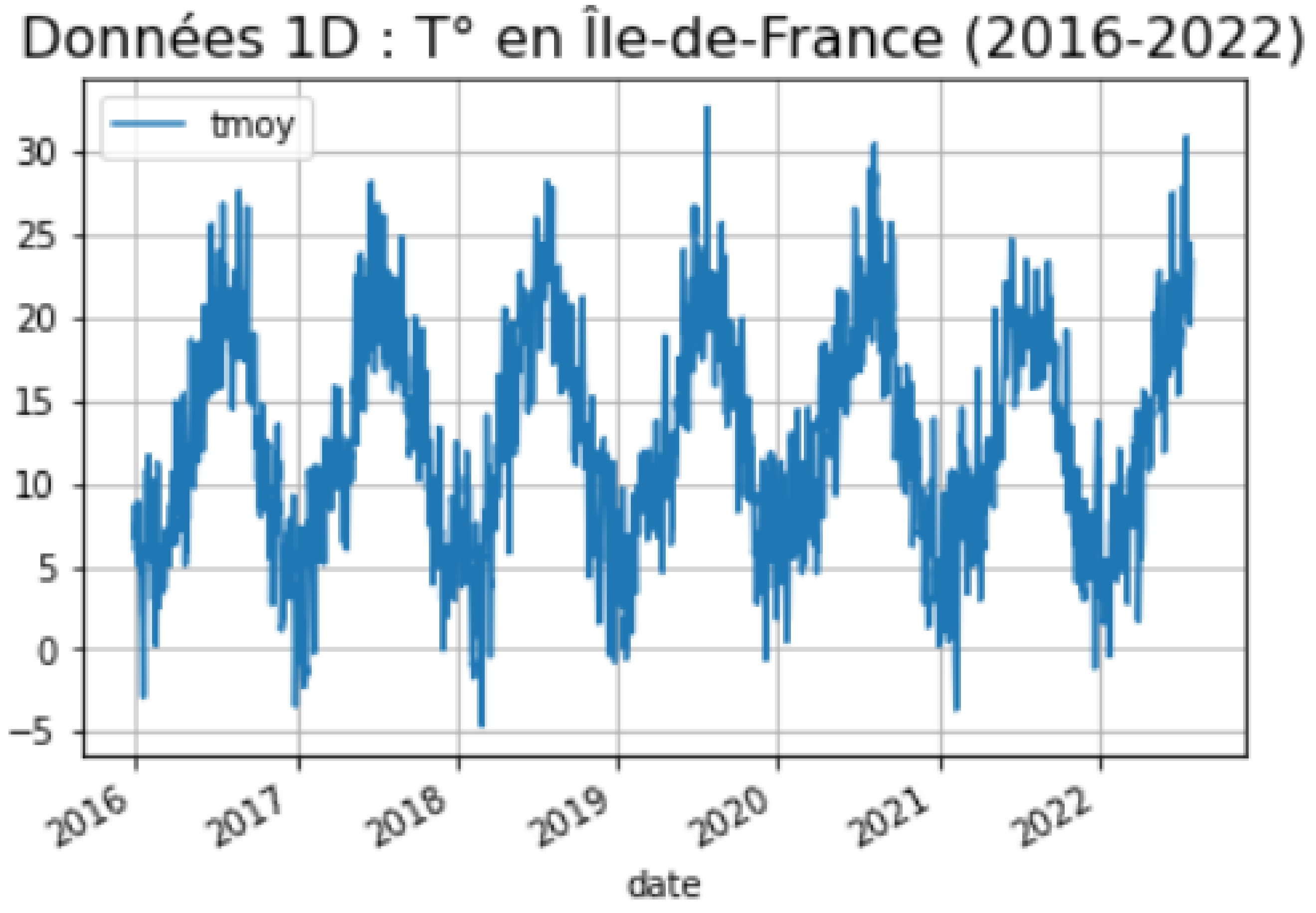
California Housing dataset



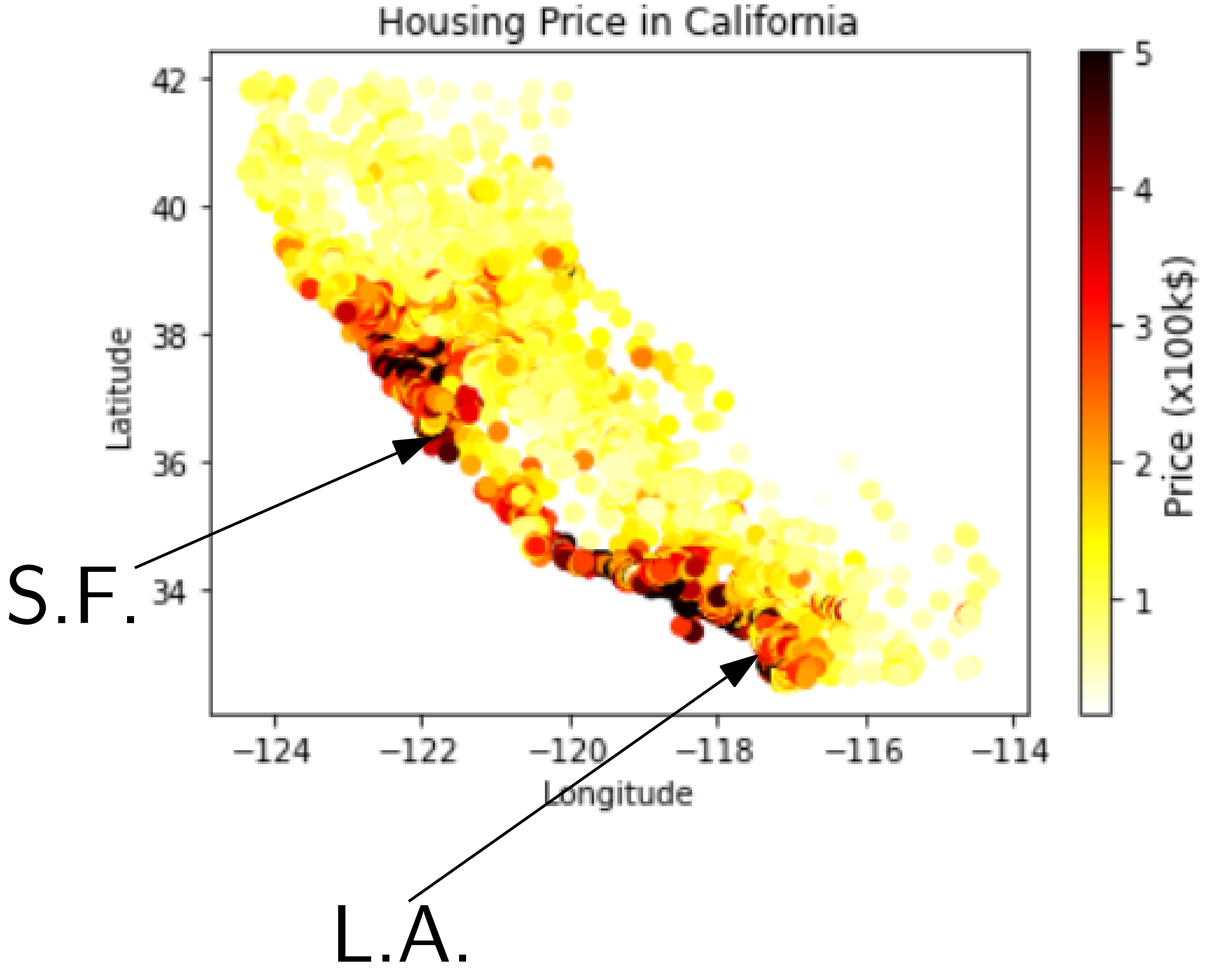
Chapter 0: Generalities and some terminology

Introduction : A data (datum?) is a piece of information recorded by a biological or artificial system. Data can appear through different forms:

- A single number : heat (e.g. $T = 38°C$), height of someone (e.g. $h = 178$cm), binary variable (e.g. 1 if someone has a driver license, 0 otherwise), etc.
- A collection of coordinates: GPS location (x, y) of some place, description (taille, poids, âge) of an individual, etc.
- But also much more complicated structures: words/text, graphs, etc.



California Housing dataset



Text: *Amazon Review Dataset*


```
{"overall": 5.0,
"reviewText": "Great product and price!"
}
```

```
{"overall": 1.0,
"reviewText": "I wore these shoe one time,
               the left shoe/sole started
               squeaking and won't stop.
               I want to return for a refund."
}
```

Chapter 0: Generalities and some terminology

Goal: Data Sciences aim at extracting information from data, in order to

- Understand some phenomena, such as:
  - Data visualization (curves, histograms, etc.) for interpretability.
  - Discovering relations between variables x and y : e.g. height $\leftrightarrow$ weight, age $\leftrightarrow$ efficiency/dangerosity of a medical treatment, etc.
  - Detect clusters: groups of data that share common properties.
- Make predictions on new data, such as:
  - Guess a value: e.g. the price of a flat given the price of other flats.
  - Take decisions: e.g. decide if an autonomous car should stop (1) or not (0).
- And many other things (data generation, reinforcement learning, etc.).

Chapter 0: Generalities and some terminology

Goal: Data Sciences aim at extracting information from data, in order to

- Understand some phenomena, such as:
  - Data visualization (curves, histograms, etc.) for interpretability.
  - Discovering relations between variables x and y : e.g. height $\leftrightarrow$ weight, age $\leftrightarrow$ efficiency/dangerosity of a medical treatment, etc.
  - Detect clusters: groups of data that share common properties.
- Make predictions on new data, such as:
  - Guess a value: e.g. the price of a flat given the price of other flats.
  - Take decisions: e.g. decide if an autonomous car should stop (1) or not (0).
- And many other things (data generation, reinforcement learning, etc.).

In practice, data sciences involve

- maths (mostly statistics, linear algebra, and optimization theory).
- Algorithms (from maths to code).
- Computer science (with dedicated software—see Chapter 1).

Chapter 0: Generalities and some terminology

Definition:

We say that we work with vectorized data if all the data belong to a common space $\mathbb{R}^d$—d being the dimension of the data. The coordinate of $x \in \mathbb{R}^d$, denoted by $x[i]$ (for $i \in \{1, \ldots, d\}$) are typically called the features of x.

Chapter 0: Generalities and some terminology

Definition:

We say that we work with vectorized data if all the data belong to a common space $\mathbb{R}^d$—d being the dimension of the data. The coordinate of $x \in \mathbb{R}^d$, denoted by $x[i]$ (for $i \in \{1, \dots, d\}$) are typically called the features of x.

Remark: With few exceptions, in this course, we will only consider vectorized data.

Why? Because the Euclidean space $\mathbb{R}^d$ comes with a linear structure: given $x_1, x_2 \in \mathbb{R}^d$, you can compute straightforwardly important quantities such as:

- $\frac{1}{2}(x_1 + x_2)$ (middle point / average),
- $x_2 - x_1$ (difference), (then norms, etc.),
- more generally, Linear algebra (apply matrix A to transform your data in a simple way, etc.).

Things get (much) harder if you do not have access to such tools (how would you compute the difference between two graphs? The average of several words?).

Definition:

A dataset is a collection of n data ("observations") $x_1, \dots, x_n \in \mathbb{R}^d$.

Remark: We will denote by $x_j[i]$ the ith feature of the jth observation in our dataset.

A dataset made of n observations in dimension d can equivalently be represented by a $n \times d$ matrix

$$X = \begin{pmatrix} x_1[1] & \dots & x_1[d] \\ \vdots & & \vdots \\ x_n[1] & \dots & x_n[d] \end{pmatrix}.$$

Number of observations n

Number of features d

Definition:

A dataset is a collection of n data ("observations") $x_1, \ldots, x_n \in \mathbb{R}^d$.

Remark: We will denote by $x_j[i]$ the ith feature of the jth observation in our dataset.

A dataset made of n observations in dimension d can equivalently be represented by a $n \times d$ matrix

$$X = \begin{pmatrix} x_1[1] & \ldots & x_1[d] \\ \vdots & & \vdots \\ x_n[1] & \ldots & x_n[d] \end{pmatrix}.$$

Number of observations n

Number of features d

Remark: Nowadays, we are often confronted to huge datasets ($n \simeq 10^9$) in high dimension ($d \simeq 10^6$). This is what we call big data. Leveraging such datasets in practice requires specific methods (parallel computing, etc.). This will not be covered by this course.

Categorical data:

Definition:

A data is said to be categorical if (some of) its *features* take values in a **finite** set.

Example : a color set {red, blue, green, orange, white, black}, a city name, a university track...

Categorical data:

Definition:

A data is said to be categorical if (some of) its *features* take values in a **finite** set.

Example : a color set {red, blue, green, orange, white, black}, a city name, a university track...

Question : How to incorporate categorical data in some numerical analysis?
→ In practice, most machine learning models require the data to be purely vectorized. How to turn our categorical data into vectors in a meaningful way?

Categorical data:

Definition:

A data is said to be categorical if (some of) its *features* take values in a **finite** set.

Example : a color set {red, blue, green, orange, white, black}, a city name, a university track...

Question : How to incorporate categorical data in some numerical analysis?
→ In practice, most machine learning models require the data to be purely vectorized. How to turn our categorical data into vectors in a meaningful way?

First idea: Assign an arbitrary number to all possible values. For instance, red → 1, blue → 2, etc.

Categorical data:

Definition:

A data is said to be categorical if (some of) its *features* take values in a **finite** set.

Example : a color set {red, blue, green, orange, white, black}, a city name, a university track...

Question : How to incorporate categorical data in some numerical analysis?
→ In practice, most machine learning models require the data to be purely vectorized. How to turn our categorical data into vectors in a meaningful way?

First idea: Assign an arbitrary number to all possible values. For instance, red → 1, blue → 2, etc.

Issue: This introduce some *implicit geometry* in your data that may fickle your models!
→ There is (*a priori*) no reason to consider that "red $\leqslant$ blue $\leqslant$ green", or that "orange $= 4\times$ red", etc.

Chapter 0: Generalities and some terminology

Categorical data:

Definition:

A data is said to be categorical if (some of) its *features* take values in a **finite** set.

Example : a color set {red, blue, green, orange, white, black}, a city name, a university track...

Question : How to incorporate categorical data in some numerical analysis?
$\rightarrow$ In practice, most machine learning models require the data to be purely vectorized. How to turn our categorical data into vectors in a meaningful way?

Second idea: Rely on *one-hot encoding*: if you have K possible values for the *feature* of interest (e.g. K colors), you can represent the k-th category by the vector

$$(0, \dots, 0, 1, 0, \dots, 0) \in \mathbb{R}^K$$

k-th coordinate of the vector

For instance, if you have three colors red, blue, green:
red $\leftrightarrow (1, 0, 0)$
blue $\leftrightarrow (0, 1, 0)$
green $\leftrightarrow (0, 0, 1)$

Categorical data:

Definition:

A data is said to be categorical if (some of) its *features* take values in a **finite** set.

Example : a color set {red, blue, green, orange, white, black}, a city name, a university track...

Question : How to incorporate categorical data in some numerical analysis?
→ In practice, most machine learning models require the data to be purely vectorized. How to turn our categorical data into vectors in a meaningful way?

Second idea: Rely on *one-hot encoding*: if you have K possible values for the *feature* of interest (e.g. K colors), you can represent the k-th category by the vector

$$(0, \dots, 0, 1, 0, \dots, 0) \in \mathbb{R}^K$$

↑ k-th coordinate of the vector

Warning: The larger K, the larger the dimension of your one-hot-encoding (which can be an issue in some situations).

Categorical data:

Definition:

A data is said to be categorical if (some of) its *features* take values in a **finite** set.

Example : a color set {red, blue, green, orange, white, black}, a city name, a university track...

Question : How to incorporate categorical data in some numerical analysis?
→ In practice, most machine learning models require the data to be purely vectorized. How to turn our categorical data into vectors in a meaningful way?

Second idea: Rely on *one-hot encoding*: if you have K possible values for the *feature* of interest (e.g. K colors), you can represent the k-th category by the vector

$$(0, \ldots, 0, 1, 0, \ldots, 0) \in \mathbb{R}^K$$

↑ k-th coordinate of the vector

Remark: It may happen that we do not know all possible categories in advance. In that case, it can be convenient to create the category "other".

Chapter 0: Generalities and some terminology

Descriptive statistics:

In practice, do **not** rush by applying sophisticated machine learning models immediately; it can be very useful to perform some descriptive statistics on our dataset.

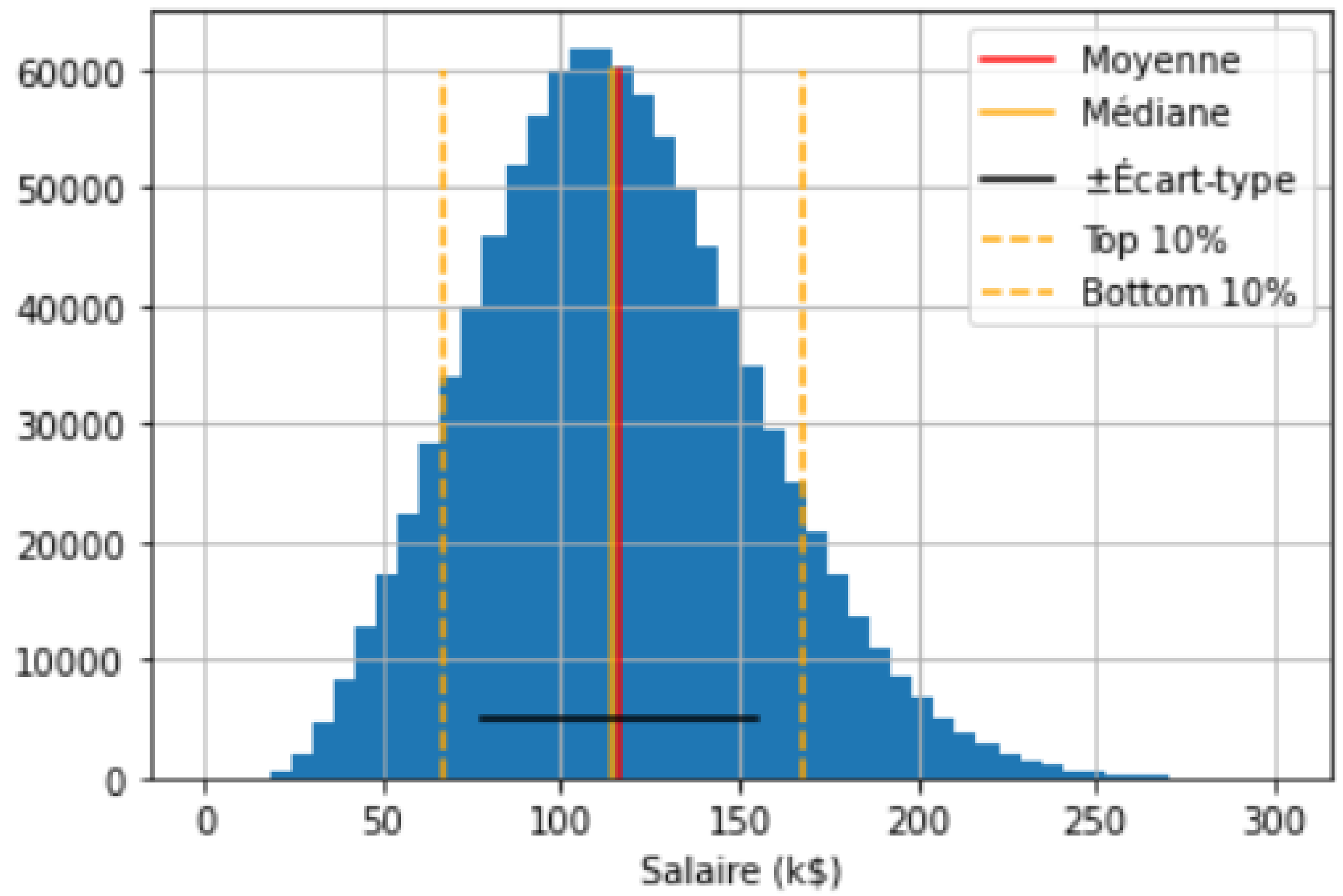
It is about looking for some standard quantities such as the mean, the variance / standard-deviation, the correlation between features, the quantiles or conditional laws.

Do not neglect this preliminary phase. It often allows you to "understand" your dataset, the kind of issue you may face when doing further analysis, etc.

Descriptive statistics:

In practice, do **not** rush by applying sophisticated machine learning models immediately; it can be very useful to perform some descriptive statistics on our dataset.

It is about looking for some standard quantities such as the mean, the variance / standard-deviation, the correlation between features, the quantiles or conditional laws.



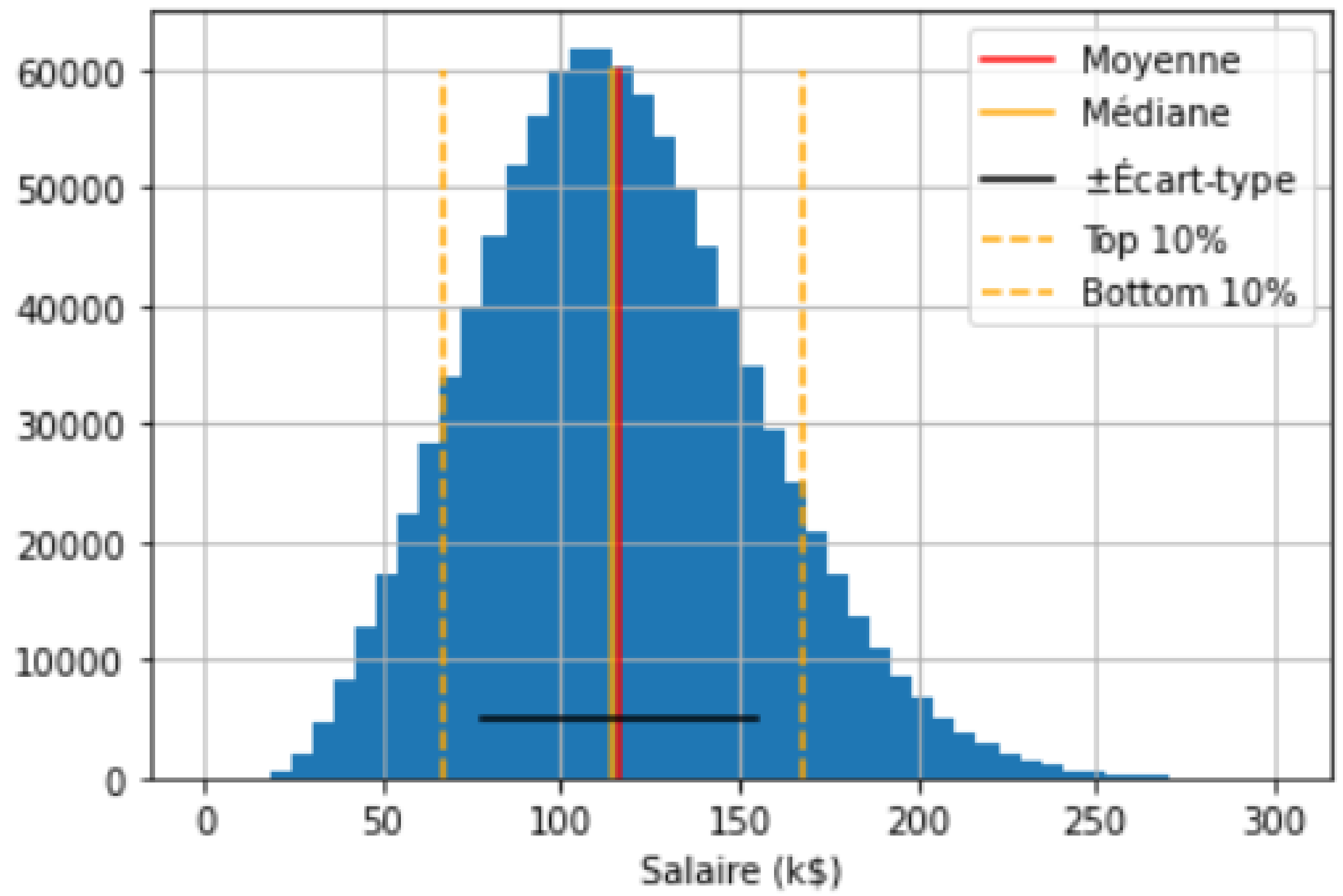
Do not neglect this preliminary phase. It often allows you to "understand" your dataset, the kind of issue you may face when doing further analysis, etc.

Descriptive statistics:

In practice, do **not** rush by applying sophisticated machine learning models immediately; it can be very useful to perform some descriptive statistics on our dataset.

It is about looking for some standard quantities such as the mean, the variance / standard-deviation, the correlation between features, the quantiles or conditional laws.

	yearsExperience	milesFromMetropolis	salary
yearsExperience	1.000000	0.000673	0.375013
milesFromMetropolis	0.000673	1.000000	-0.297666
salary	0.375013	-0.297666	1.000000



Do not neglect this preliminary phase. It often allows you to "understand" your dataset, the kind of issue you may face when doing further analysis, etc.

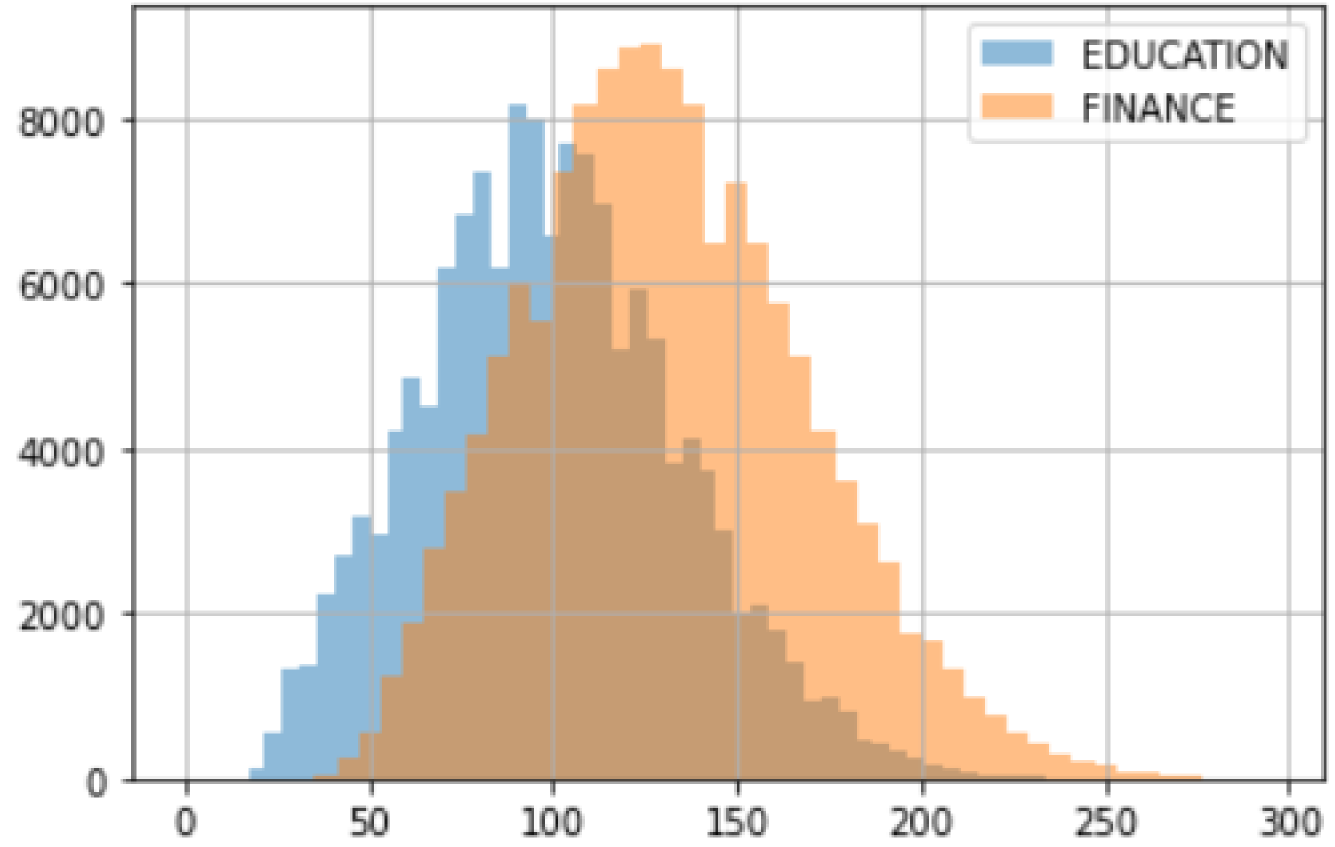
Chapter 0: Generalities and some terminology

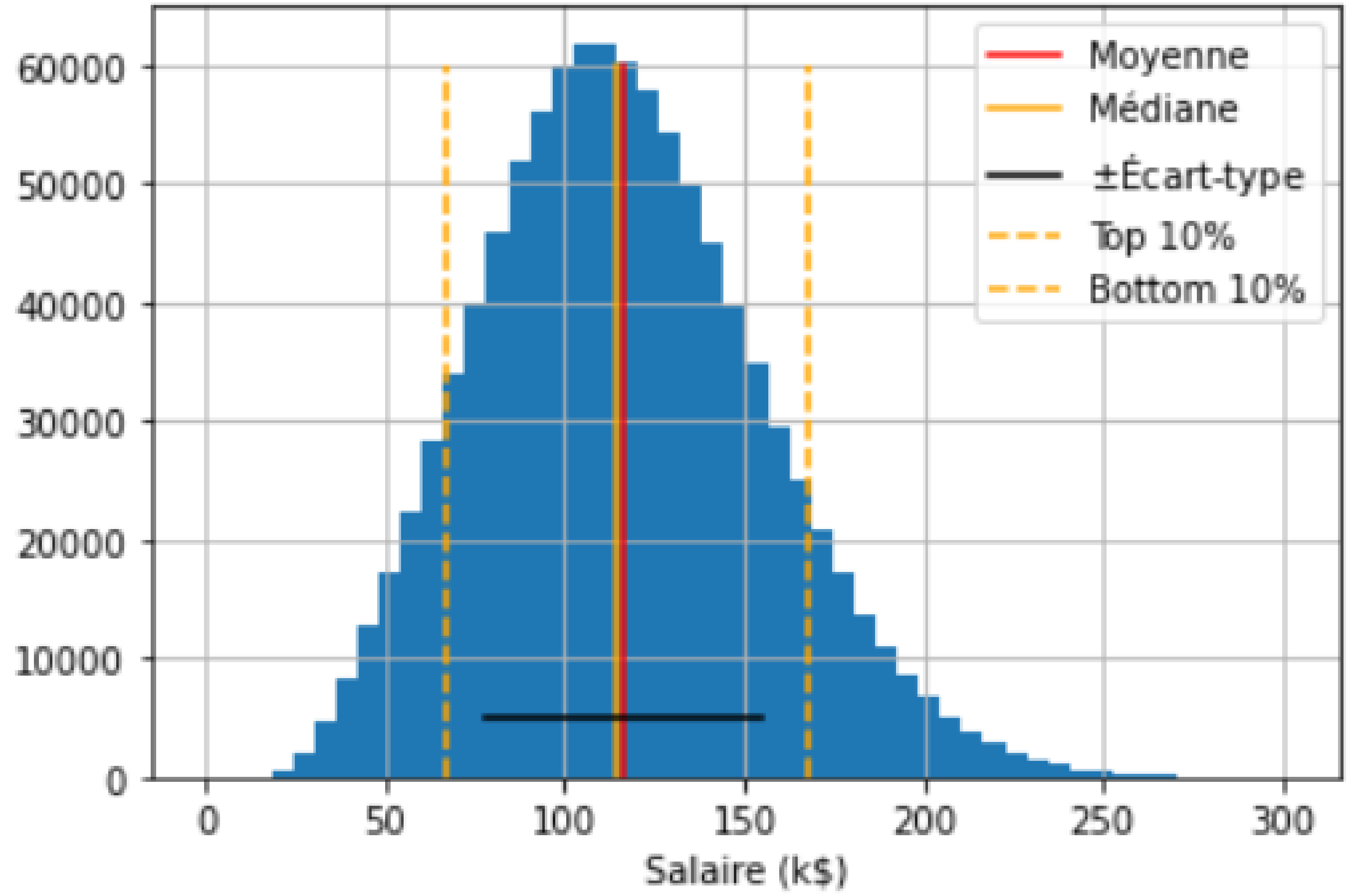
Descriptive statistics:

In practice, do **not** rush by applying sophisticated machine learning models immediately; it can be very useful to perform some descriptive statistics on our dataset.

It is about looking for some standard quantities such as the mean, the variance / standard-deviation, the correlation between features, the quantiles or conditional laws.

	yearsExperience	milesFromMetropolis	salary
yearsExperience	1.000000	0.000673	0.375013
milesFromMetropolis	0.000673	1.000000	-0.297666
salary	0.375013	-0.297666	1.000000





Do not neglect this preliminary phase. It often allows you to "understand" your dataset, the kind of issue you may face when doing further analysis, etc.

Descriptive statistics:
In practice, do **not** rush by applying sophisticated machine learning models immediately; it can be very useful to perform some descriptive statistics on our dataset.
It is about looking for some standard quantities such as the mean, the variance / standard-deviation, the correlation between features, the quantiles or conditional laws.

	yearsExperience	milesFromMetropolis	salary
yearsExperience	1.000000	0.000673	0.375013
milesFromMetropolis	0.000673	1.000000	-0.297666
salary	0.375013	-0.297666	1.000000

A word about correlation: The correlation between two variables/features X and Y indicate if *knowing* Y *gives some information on* X ; we denote by $X|Y$ the relation "X given Y".
• Correlation close to 1 $\Rightarrow$ "when Y increases, X tends to increase as well" (e.g. X=weight, Y=height),
• close to -1 $\Rightarrow$ "Y increases $\leftrightarrow$ X decreases" (e.g. risk of heart attack | sport practice),
• close to 0 : "no clear relation" (e.g. height | hour at which you were born).

Do not neglect this preliminary phase. It often allows you to "understand" your dataset, the kind of issue you may face when doing further analysis, etc.

Chapter 0: Generalities and some terminology

Descriptive statistics:
In practice, do **not** rush by applying sophisticated machine learning models immediately; it can be very useful to perform some descriptive statistics on our dataset.
It is about looking for some standard quantities such as the mean, the variance / standard-deviation, the correlation between features, the quantiles or conditional laws.

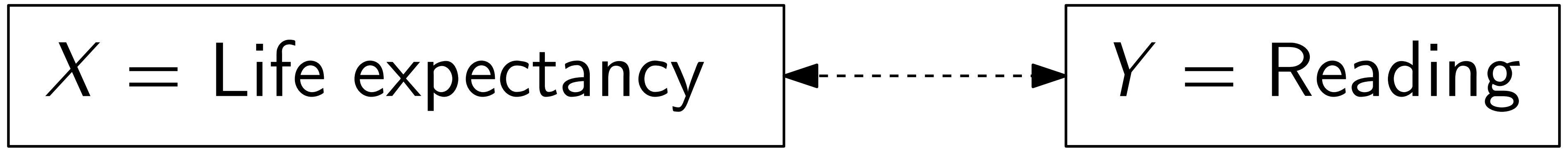
Warning, Do not confuse correlation and causality!

Example: X = life expectancy, Y = weekly reading time.
Observation: These variables are correlated (the more you read, the more you live). But can you faithfully conclude that reading does increase *directly* the life expectancy (everything else remaining unchanged)?

Une étude prouve que lire des livres prolonge la vie

Par Alice Develey
Publié le 09/08/2016 à 12:08, mis à jour le 10/08/2016 à 10:29

D'après une récente étude menée par l'Université de Yale, lire plus de 3h30 par semaine aiderait à prolonger l'espérance de vie de plus de 20% sur douze ans.
Source : Le Figaro.

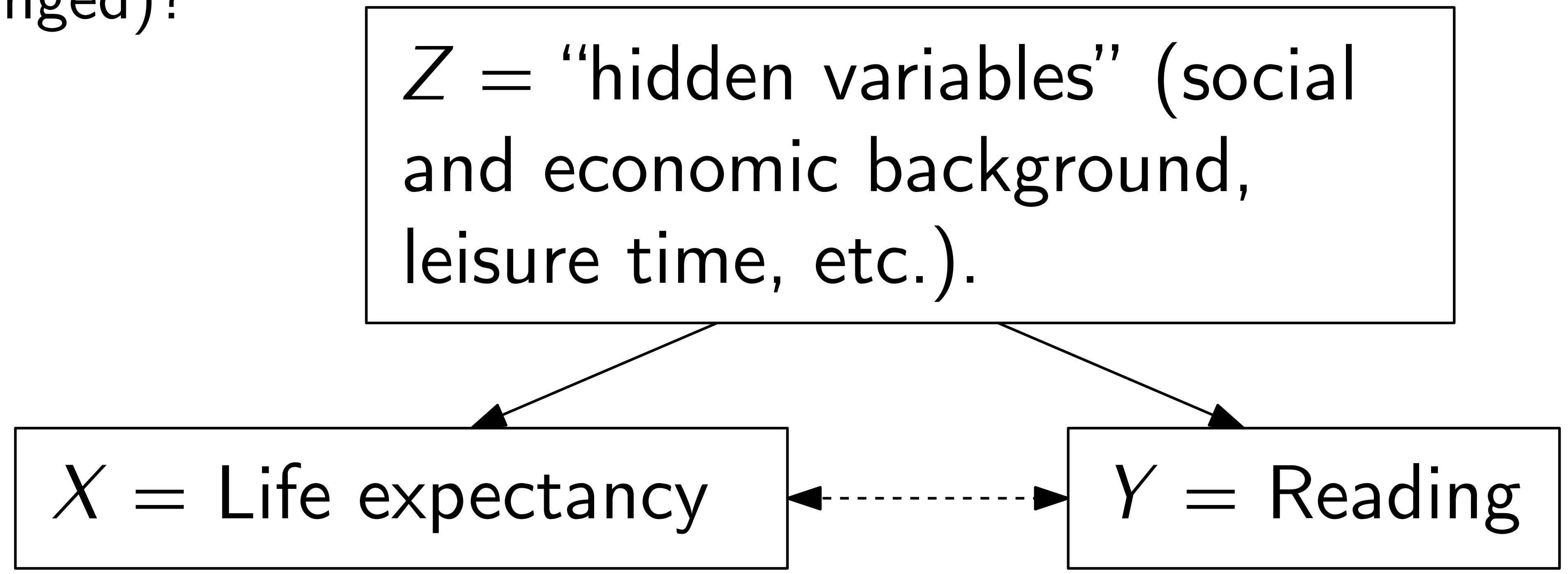


Do not neglect this preliminary phase. It often allows you to "understand" your dataset, the kind of issue you may face when doing further analysis, etc.

Descriptive statistics:
In practice, do **not** rush by applying sophisticated machine learning models immediately; it can be very useful to perform some descriptive statistics on our dataset.
It is about looking for some standard quantities such as the mean, the variance / standard-deviation, the correlation between features, the quantiles or conditional laws.

Warning, Do not confuse correlation and causality!

Example: X = life expectancy, Y = weekly reading time.
Observation: These variables are correlated (the more you read, the more you live). But can you faithfully conclude that reading does increase *directly* the life expectancy (everything else remaining unchanged)?




Une étude prouve que lire des livres prolonge la vie

Par Alice Develey
Publié le 09/08/2016 à 12:08, mis à jour le 10/08/2016 à 10:29

D'après une récente étude menée par l'Université de Yale, lire plus de 3h30 par semaine aiderait à prolonger l'espérance de vie de plus de 20% sur douze ans.

Source : Le Figaro.

Do not neglect this preliminary phase. It often allows you to "understand" your dataset, the kind of issue you may face when doing further analysis, etc.

Learning: Aside from data visualization, most tasks in data sciences are machine learning (ML) tasks.
There are two main categories of ML tasks: supervised learning and unsupervised learning.

Chapter 0: Generalities and some terminology

Learning: Aside from data visualization, most tasks in data sciences are machine learning (ML) tasks.
There are two main categories of ML tasks: supervised learning and unsupervised learning.

Supervised learning: For each observation $x_j \in \mathcal{X} = \mathbb{R}^d$, we are also given a corresponding label $y_j \in \mathcal{Y}$.
The goal is to design a model $F : \mathcal{X} \to \mathcal{Y}$ such that $F(x_j) \simeq y_j$ **on average**. Formally, we search F that would **minimize**

$$\frac{1}{n} \sum_{j=1}^{n} \ell(F(x_j), y_j), \tag{1}$$

where ℓ is a loss function that measures the discrepancy between a prediction $F(x_j)$ and the expected label y_j.

Chapter 0: Generalities and some terminology

Learning: Aside from data visualization, most tasks in data sciences are machine learning (ML) tasks.
There are two main categories of ML tasks: supervised learning and unsupervised learning.

Supervised learning: For each observation $x_j \in \mathcal{X} = \mathbb{R}^d$, we are also given a corresponding label $y_j \in \mathcal{Y}$.
The goal is to design a model $F\colon \mathcal{X} \to \mathcal{Y}$ such that $F(x_j) \simeq y_j$ **on average**. Formally, we search F that would **minimize**

$$\frac{1}{n}\sum_{j=1}^{n} \ell(F(x_j), y_j), \tag{1}$$

where ℓ is a loss function that measures the discrepancy between a prediction $F(x_j)$ and the expected label y_j.

Example 1: Predict the weight of someone (label) given their height (observation).
One possible model is to take the height $x \in \mathcal{X} = \mathbb{R}$ and to multiply it by a parameter $\theta \in \mathbb{R}$. Hopefully, one has $\theta \cdot x \simeq y$, the corresponding weight. We will often chose the loss function to be $\ell(F(x), y) = ||F(x) - y||^2$. In that case, our goal is therefore to find θ that minimizes

$$\theta \mapsto L(\theta) = \frac{1}{n}\sum_{j=1}^{n} ||\theta \cdot x_j - y_j||^2.$$

L is called the objective function, and (because $\ell = ||\cdot - \cdot||^2$) is called here the mean squared error (MSE).

Learning: Aside from data visualization, most tasks in data sciences are machine learning (ML) tasks.
There are two main categories of ML tasks: supervised learning and unsupervised learning.

Supervised learning: For each observation $x_j \in \mathcal{X} = \mathbb{R}^d$, we are also given a corresponding label $y_j \in \mathcal{Y}$.
The goal is to design a model $F : \mathcal{X} \to \mathcal{Y}$ such that $F(x_j) \simeq y_j$ **on average**. Formally, we search F that would **minimize**

$$\frac{1}{n} \sum_{j=1}^{n} \ell(F(x_j), y_j), \tag{1}$$

where ℓ is a loss function that measures the discrepancy between a prediction $F(x_j)$ and the expected label y_j.

Example 2: Predict, given the desciption of an email x (sender, date, content) if it is a spam ($y = 1$) or not ($y = 0$).
We can evaluate a model F by counting the number of errors it makes, that is when $F(x_j) \neq y_j$, yielding

$$\frac{1}{n} \sum_{j=1}^{n} 1_{F(x_j) \neq y_j}.$$

An example of (naive) model would be to say $F(x) = 1$ in the email x includes “Congratulations, you won!” and 0 otherwise.

Learning: Aside from data visualization, most tasks in data sciences are machine learning (ML) tasks.
There are two main categories of ML tasks: supervised learning and unsupervised learning.

Unsupervised learning: When you do not have labels. In that case, the objective function **depends only on the observations**.

Chapter 0: Generalities and some terminology

Learning: Aside from data visualization, most tasks in data sciences are machine learning (ML) tasks.
There are two main categories of ML tasks: supervised learning and unsupervised learning.

Unsupervised learning: When you do not have labels. In that case, the objective function **depends only on the observations**.

Example 1: We are given observations $x_1, \dots, x_n \in \mathbb{R}^d$ and we seek for a **representative** $\hat{x}$ that would be close, on average and for the squared Euclidean loss, from the $(x_j)_{j=1}^n$. It should thus minimize the objective function

$$x \mapsto \frac{1}{n} \sum_{j=1}^{n} ||x_j - x||^2, \quad \text{for } x \in \mathbb{R}^d.$$

Exercise: Determine the expression of the optimal $\hat{x}$. What if we had chosen $\ell = ||\cdot - \cdot||$? (no square) And $\ell = ||\cdot - \cdot||^p$ for $p > 1$ (but $p \neq 2$)?

Learning: Aside from data visualization, most tasks in data sciences are machine learning (ML) tasks.
There are two main categories of ML tasks: supervised learning and unsupervised learning.

Unsupervised learning: When you do not have labels. In that case, the objective function **depends only on the observations**.

Example 2: dimensionality reduction. Assume that we are given a dataset $X \in \mathbb{R}^{n \times D}$ (n observations in dimension D) with D large. For various reasons (visualization, computational efficiency...), one may want to "approximate" X by a lower dimensional object $\hat{X} \in \mathbb{R}^{n \times d}$, with $d \ll D$.

Learning: Aside from data visualization, most tasks in data sciences are machine learning (ML) tasks.
There are two main categories of ML tasks: supervised learning and unsupervised learning.

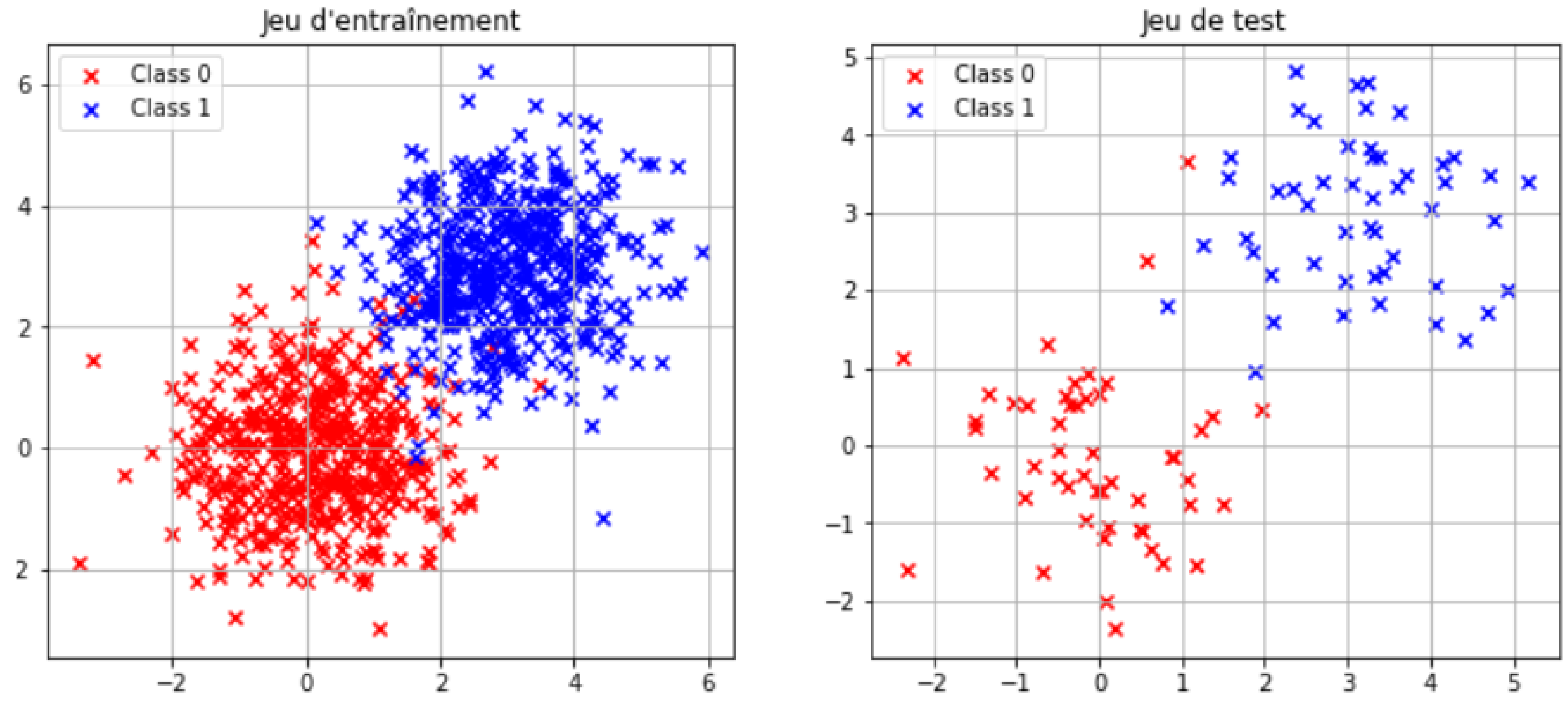
Supervised	Unsupervised
⊕ We have a ground truth and a clear objective. We can precisely measure "how good" is our model and estimate the probability that its predictions are correct/close to the true value. ⊖ We need actual labels to train and test our model. Producing labels is demanding, and concerning in itself in some cases. → Most common situation in practice.	⊕ No need for labels. Recording data is sufficient. ⊖ Hard to evaluate a given model, say that a model is better than another one. We do not know what the best possible model is; how good or bad we currently are. → Mostly used in exploratory phases, or as a preprocessing.

Chapter 0: Generalities and some terminology

Learning: Aside from data visualization, most tasks in data sciences are machine learning (ML) tasks.
There are two main categories of ML tasks: supervised learning and unsupervised learning.

The machine learning routine: Roughly speaking, addressing a machine learning task goes in the following way:

1. Collect observations (and labels). Split them into two groups: the training set and the test set (or validation set).
2. Fix an objective (here, classify blue points vs. red points).

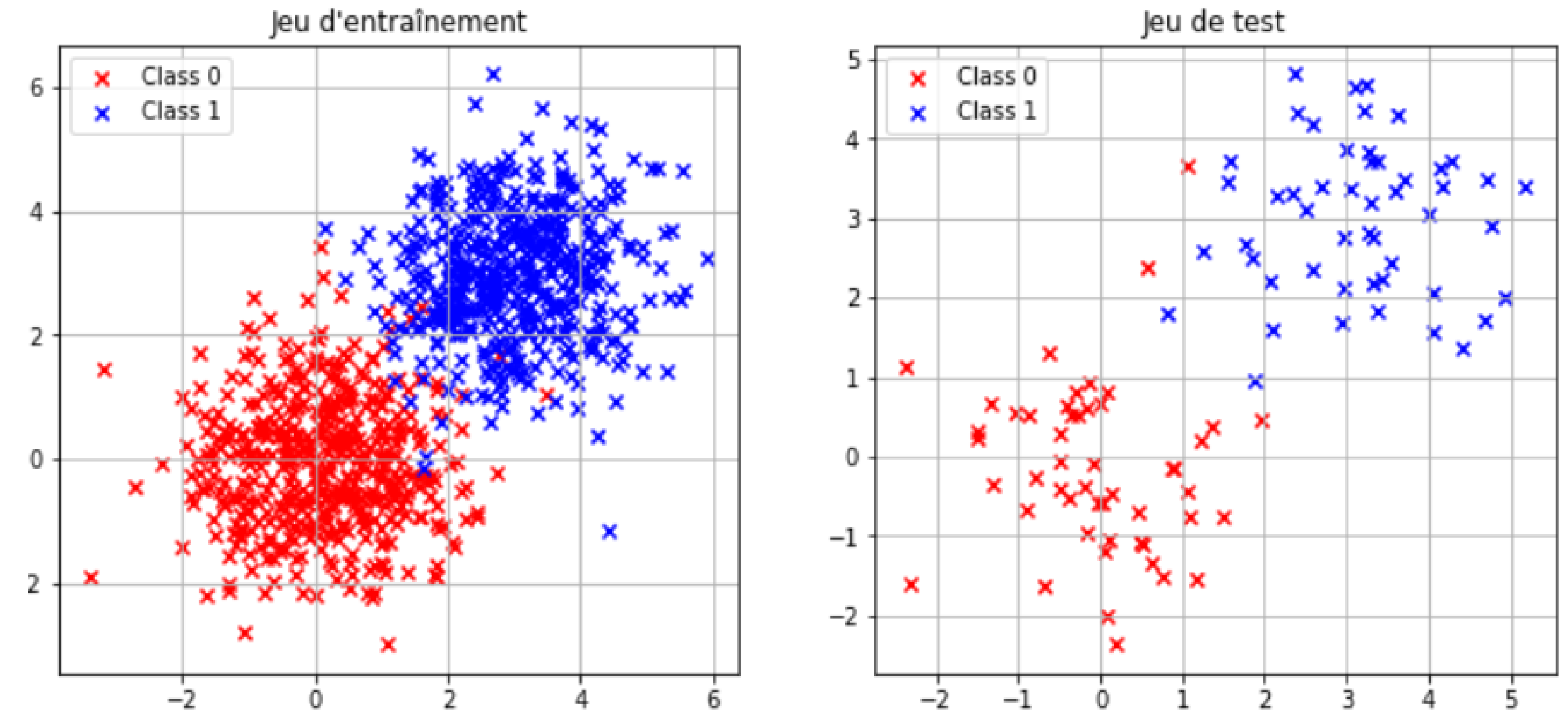


Chapter 0: Generalities and some terminology

Learning: Aside from data visualization, most tasks in data sciences are machine learning (ML) tasks.
There are two main categories of ML tasks: supervised learning and unsupervised learning.

The machine learning routine: Roughly speaking, addressing a machine learning task goes in the following way:

1. Collect observations (and labels). Split them into two groups: the training set and the test set (or validation set).
2. Fix an objective (here, classify blue points vs. red points).
3. Chose a class of models (here, a logistic regression, see later).



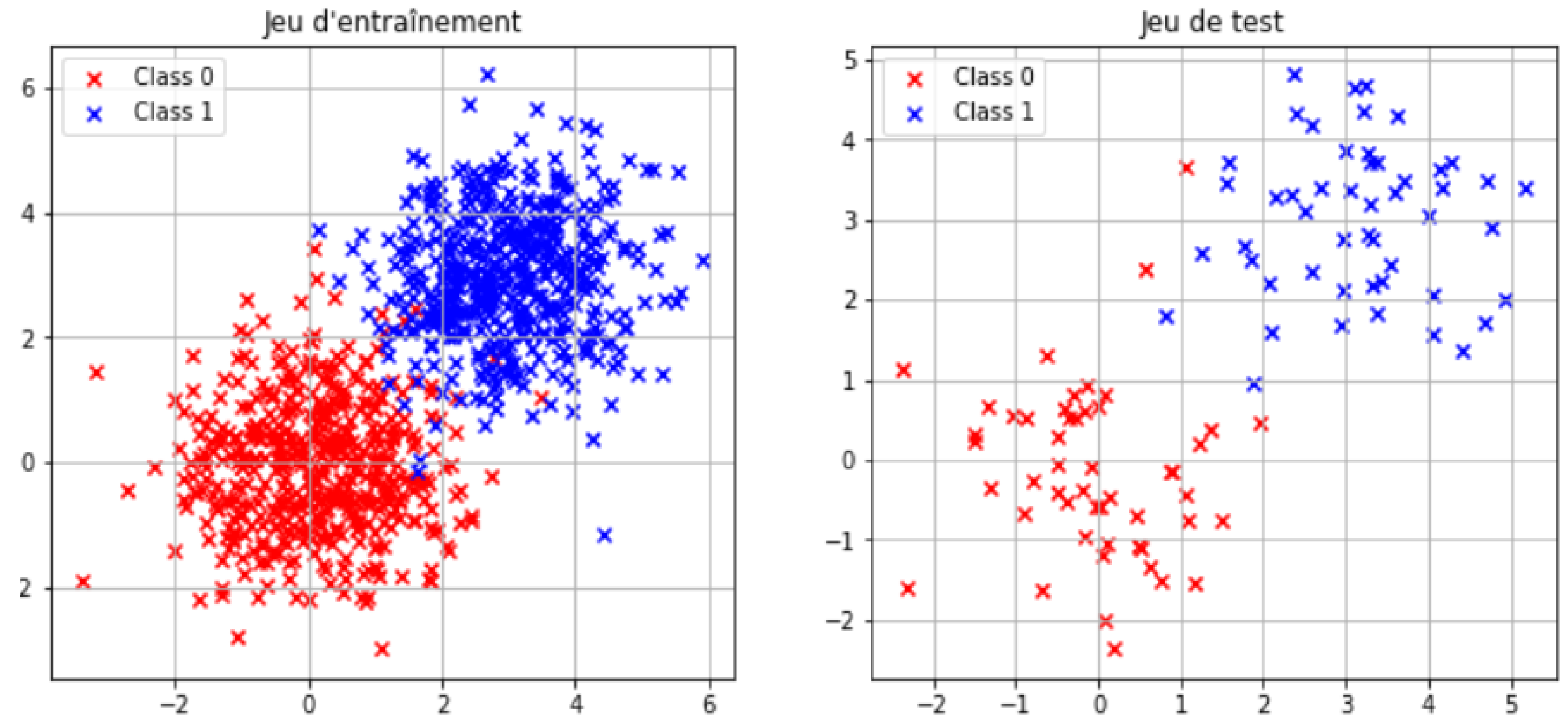

```
model = LogisticRegression()
```

Chapter 0: Generalities and some terminology

Learning: Aside from data visualization, most tasks in data sciences are machine learning (ML) tasks.
There are two main categories of ML tasks: supervised learning and unsupervised learning.

The machine learning routine: Roughly speaking, addressing a machine learning task goes in the following way:

1. Collect observations (and labels). Split them into two groups: the training set and the test set (or validation set).
2. Fix an objective (here, classify blue points vs. red points).
3. Chose a class of models (here, a logistic regression, see later).
4. Train your model so that it adapts to the specificity of this dataset.




```
model = LogisticRegression()
model.fit(x_train, y_train)
s = model.score(x_train,y_train)
print("Le score (proportion de prédictions correctes) du modèle sur le jeu d'entraînement est de %.2f%%" %(100*s))
```

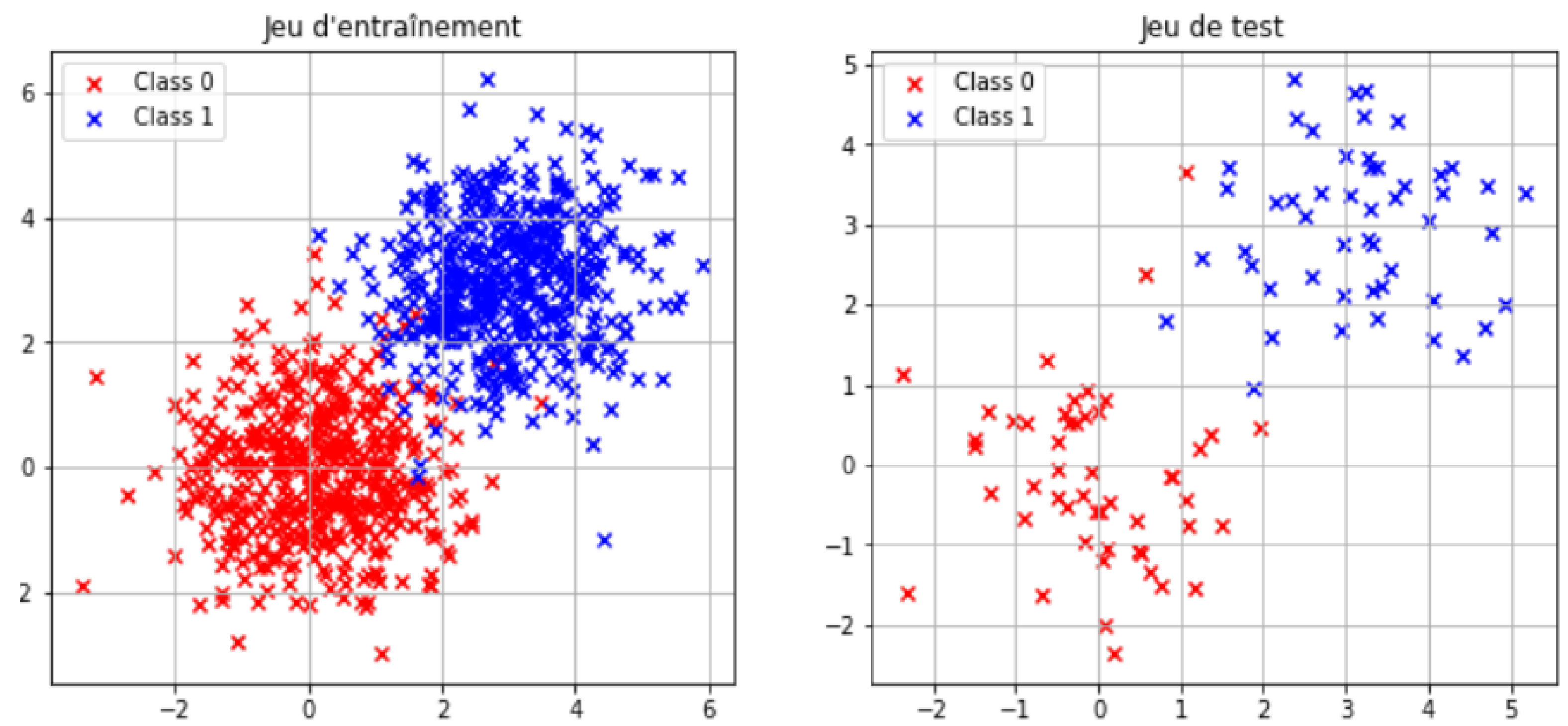
```
Le score (proportion de prédictions correctes) du modèle sur le jeu d'entraînement est de 98.30%
```

Chapter 0: Generalities and some terminology

Learning: Aside from data visualization, most tasks in data sciences are machine learning (ML) tasks.
There are two main categories of ML tasks: supervised learning and unsupervised learning.

The machine learning routine: Roughly speaking, addressing a machine learning task goes in the following way:

1. Collect observations (and labels). Split them into two groups: the training set and the test set (or validation set).
2. Fix an objective (here, classify blue points vs. red points).
3. Chose a class of models (here, a logistic regression, see later).
4. Train your model so that it adapts to the specificity of this dataset.
5. Test your model on the test set.




```python
model = LogisticRegression()
model.fit(x_train, y_train)
s = model.score(x_train,y_train)
print("Le score (proportion de prédictions correctes) du modèle sur le jeu d'entraînement est de %.2f%%" %(100*s))
s_test = model.score(x_test, y_test)
print("Le score (proportion de prédictions correctes) du modèle sur le jeu de test est de %.2f%%" %(100*s_test))
```

```
Le score (proportion de prédictions correctes) du modèle sur le jeu d'entraînement est de 98.30%
Le score (proportion de prédictions correctes) du modèle sur le jeu de test est de 97.00%
```

- On the difference between "statistical learning" and "machine learning".

What you do with T. Bonis in "mathematical foundation for Data Sciences"

What we'll do in this class.

→ Work in an abstract / hypothetical setting where observations $X \in \mathcal{P}(\mathcal{X})$ (and possible labels $Y \in \mathcal{P}(\mathcal{Y})$) are random variables (following a **joint law**), consider class of models $\mathcal{F}$, and try to minimize quantities like

$$\min_{f \in \mathcal{F}} \mathbb{E}[\ell(f(X), Y)].$$

For instance, if $\ell(x, y) := |x - y|^2$, and $\mathcal{F}$ is the set of any (measurable) functions from $\mathcal{X}$ to $\mathcal{Y}$, you'll learn that the optimal f^* is given by $f^*(x) = \mathbb{E}[Y|X = x]$.

Chapter 0: Generalities and some terminology

- On the difference between "statistical learning" and "machine learning".

What you do with T. Bonis in "mathematical foundation for Data Sciences"

→ Work in an abstract / hypothetical setting where observations $X \in \mathcal{P}(\mathcal{X})$ (and possible labels $Y \in \mathcal{P}(\mathcal{Y})$) are random variables (following a **joint law**), consider class of models $\mathcal{F}$, and try to minimize quantities like

$$\min_{f \in \mathcal{F}} \mathbb{E}[\ell(f(X), Y)].$$

For instance, if $\ell(x, y) := |x - y|^2$, and $\mathcal{F}$ is the set of any (measurable) functions from $\mathcal{X}$ to $\mathcal{Y}$, you'll learn that the optimal f^* is given by $f^*(x) = \mathbb{E}[Y|X = x]$.

What we'll do in this class.

→ We have a **realization** of observations $x_1, \dots, x_n$ (and possible labels $y_1, \dots, y_n$), and try to find a model $f \in \mathcal{F}$ that is good on these observations. In statistical terms, we will compute an estimator $\hat{f}_n$ based on the $(x_i)_i$ and $(y_i)_i$, and provided n is large and assuming that $x_i, y_i \sim^{iid} X, Y$, we may expect (hope...) that $\hat{f}_n \simeq f^*$.
Machine learning can be thought as "empirical statistical learning".

- On the difference between "statistical learning" and "machine learning".

What you do with T. Bonis in "mathematical foundation for Data Sciences"

→ Work in an abstract / hypothetical setting where observations $X \in \mathcal{P}(\mathcal{X})$ (and possible labels $Y \in \mathcal{P}(\mathcal{Y})$) are random variables (following a **joint law**), consider class of models $\mathcal{F}$, and try to minimize quantities like

$$\min_{f \in \mathcal{F}} \mathbb{E}[\ell(f(X), Y)].$$

For instance, if $\ell(x, y) := |x - y|^2$, and $\mathcal{F}$ is the set of any (measurable) functions from $\mathcal{X}$ to $\mathcal{Y}$, you'll learn that the optimal f^* is given by $f^*(x) = \mathbb{E}[Y|X = x]$.

What we'll do in this class.

→ We have a **realization** of observations $x_1, \dots, x_n$ (and possible labels $y_1, \dots, y_n$), and try to find a model $f \in \mathcal{F}$ that is good on these observations.
In statistical terms, we will compute an estimator $\hat{f}_n$ based on the $(x_i)_i$ and $(y_i)_i$, and provided n is large and assuming that $x_i, y_i \sim^{iid} X, Y$, we may expect (hope...) that $\hat{f}_n \simeq f^*$.
Machine learning can be thought as "empirical statistical learning".

There will be redundancy (with different perspectives) between the two courses, but also differences in the type of problems we consider. For instance, an important question in ML is to learn $\hat{f}_n$, which is an optimization problem.
→ How do we compute $\hat{f}_n$ based on the obs/labels? What guarantees do we have?

Chapter 1: Some tools for data science

We present some standard tools used routinely in data science, all developed in `Python`.

Why `Python`? It is **the** reference programming language for the **exploratory** part of data science.
Advantages : Very easy to get started (script), many free and open-source libraries, nice development interface using `Jupyter-notebook`, environment management using `pip` and `conda`, etc.
Drawbacks : Possibly slow, non-typed ($\Rightarrow$ easy to get bugs). One may prefer more robust programming languages (`Java`,`C++`,`Scala`...) when deployed in production.

Chapter 1: Some tools for data science

We present some standard tools used routinely in data science, all developed in `Python`.

Why `Python`? It is **the** reference programming language for the **exploratory** part of data science.
Advantages : Very easy to get started (script), many free and open-source libraries, nice development interface using `Jupyter-notebook`, environment management using `pip` and `conda`, etc.
Drawbacks : Possibly slow, non-typed ($\Rightarrow$ easy to get bugs). One may prefer more robust programming languages (`Java`,`C++`,`Scala`...) when deployed in production.

Conda: We suggest that you use anaconda to manage your environment: where you define all the libraries you use, the corresponding versions, etc. You can find on elearning a file entitled `datascience.yml` that describes the `conda` environment used in this class. You can replicate it on your laptop using

```
$ conda env create -f datascience.yml
```

in a terminal.
You can also work with other tools (e.g. `pip`).

Chapter 1: Some tools for data science

We present some standard tools used routinely in data science, all developed in `Python`.

Why `Python`? It is **the** reference programming language for the **exploratory** part of data science.
Advantages : Very easy to get started (script), many free and open-source libraries, nice development interface using `Jupyter-notebook`, environment management using `pip` and `conda`, etc.
Drawbacks : Possibly slow, non-typed ($\Rightarrow$ easy to get bugs). One may prefer more robust programming languages (`Java`,`C++`,`Scala`...) when deployed in production.

Conda: We suggest that you use anaconda to manage your environment: where you define all the libraries you use, the corresponding versions, etc. You can find on elearning a file entitled `datascience.yml` that describes the `conda` environment used in this class. You can replicate it on your laptop using

```
$ conda env create -f datascience.yml
```

in a terminal.
You can also work with other tools (e.g. `pip`).

Jupyter Notebook/Lab: We will work with notebook Jupyter. It provides a very convenient interface to code in Python in a "dynamic" way.

Chapter 1: Some tools for data science

Here is a short presentation of the libraries we will use in this course.

- `NumPy` (`import numpy as np`):

This is the reference library for numerics. It enables efficient manipulation of `array` (vectors, matrices...).

Chapter 1: Some tools for data science

Here is a short presentation of the libraries we will use in this course.

- NumPy (`import numpy as np`):

This is the reference library for numerics. It enables efficient manipulation of `array` (vectors, matrices...).

- `matplotlib (import matplotlib.pyplot as plt)`

Standard libraries for plot in Python. Perfect interface with NumPy.

Chapter 1: Some tools for data science

Here is a short presentation of the libraries we will use in this course.

- `NumPy` (`import numpy as np`):
This is the reference library for numerics. It enables efficient manipulation of `array` (vectors, matrices...).

- `matplotlib (import matplotlib.pyplot as plt)`
Standard libraries for plot in Python. Perfect interface with NumPy.

- `SciPy`:
Extension of NumPy with more advanced scientific calculus (matrix reduction, Fourier transform, graphs manipulation, etc.); perfect interface with NumPy.

Chapter 1: Some tools for data science

Here is a short presentation of the libraries we will use in this course.

- `pandas`:

Library to manage and visualize datasets, encoded as `dataframe`.

Chapter 1: Some tools for data science

Here is a short presentation of the libraries we will use in this course.

- `pandas`:
Library to manage and visualize datasets, encoded as `dataframe`.

- `scikit-learn` :
The reference library for every “basic” machine learning model.

Chapter 1: Some tools for data science

Here is a short presentation of the libraries we will use in this course.

- `pandas`:
Library to manage and visualize datasets, encoded as `dataframe`.

- `scikit-learn` :
The reference library for every "basic" machine learning model.

- `jax` :
Developed by Google Brain. A library dedicated to optimization, and leveraging nicely automatic differentiation. Pretty well designed for mathematicians!

Chapter 1: Some tools for data science

Here is a short presentation of the libraries we will use in this course.

- `pandas`:
Library to manage and visualize datasets, encoded as `dataframe`.

- `scikit-learn` :
The reference library for every "basic" machine learning model.

- `jax` :
Developed by Google Brain. A library dedicated to optimization, and leveraging nicely automatic differentiation. Pretty well designed for mathematicians!

- `tensorflow` et `pytorch`:
Respectively developed by Google Brain (Alphabet) and Meta (previously Facebook), these two libraries are dedicated to neural networks. They will not be used in this introductory course, but will be used in the optional course dedicated to Deep Learning in the next semester.

Chapter 2: Supervised Learning (1) - Regression

This chapter is dedicated to a large class of supervised learning problems: regression problems.

Chapter 2: Supervised Learning (1) - Regression

This chapter is dedicated to a large class of supervised learning problems: regression problems.

Reminder : Supervised learning problems are described through observations $x_1, \dots, x_n \in \mathbb{R}^d$, and labels $y_1, \dots, y_n \in \mathcal{Y}$. Formally, we assume that they are i.i.d. data following a joint law Γ. Our goal is to design a model $F : \mathbb{R}^d \to \mathcal{Y}$ such that $\mathbb{E}_{(x,y)\sim\Gamma}[\ell(F(x), y)]$ is small for the chosen loss function ℓ.
In practice, Γ is unknown, so we replace the above expectation by its empirical counterpart using our training data, that is $\frac{1}{n}\sum_{i=1}^n \ell(F(x_i), y_i)$.

Chapter 2: Supervised Learning (1) - Regression

This chapter is dedicated to a large class of supervised learning problems: regression problems.

Reminder : Supervised learning problems are described through observations $x_1, \dots, x_n \in \mathbb{R}^d$, and labels $y_1, \dots, y_n \in \mathcal{Y}$. Formally, we assume that they are i.i.d. data following a joint law Γ. Our goal is to design a model $F : \mathbb{R}^d \to \mathcal{Y}$ such that $\mathbb{E}_{(x,y)\sim\Gamma}[\ell(F(x), y)]$ is small for the chosen loss function ℓ.
In practice, Γ is unknown, so we replace the above expectation by its empirical counterpart using our training data, that is $\frac{1}{n}\sum_{i=1}^{n} \ell(F(x_i), y_i)$.

Definition:

When the labels are quantitative variables, that is $\mathcal{Y} = \mathbb{R}^k$, $k \geqslant 1$, we say that we are addressing a regression task. In that case, a typical choice of loss function is $\ell(F(x), y) = ||F(x) - y||^2$, inducing the mean squared error.

As detailed in Chapter 4, the other fundamental scenario is when the labels are categorical variables (i.e. $\mathcal{Y}$ is finite; e.g. color, type of animal, etc.), in which case we say that we are addressing a classification problem.

Chapter 2: Supervised Learning (1) - Regression

This chapter is dedicated to a large class of supervised learning problems: regression problems.

Reminder : Supervised learning problems are described through observations $x_1, \dots, x_n \in \mathbb{R}^d$, and labels $y_1, \dots, y_n \in \mathcal{Y}$. Formally, we assume that they are i.i.d. data following a joint law Γ. Our goal is to design a model $F : \mathbb{R}^d \to \mathcal{Y}$ such that $\mathbb{E}_{(x,y)\sim\Gamma}[\ell(F(x), y)]$ is small for the chosen loss function ℓ.
In practice, Γ is unknown, so we replace the above expectation by its empirical counterpart using our training data, that is $\frac{1}{n}\sum_{i=1}^n \ell(F(x_i), y_i)$.

Definition:

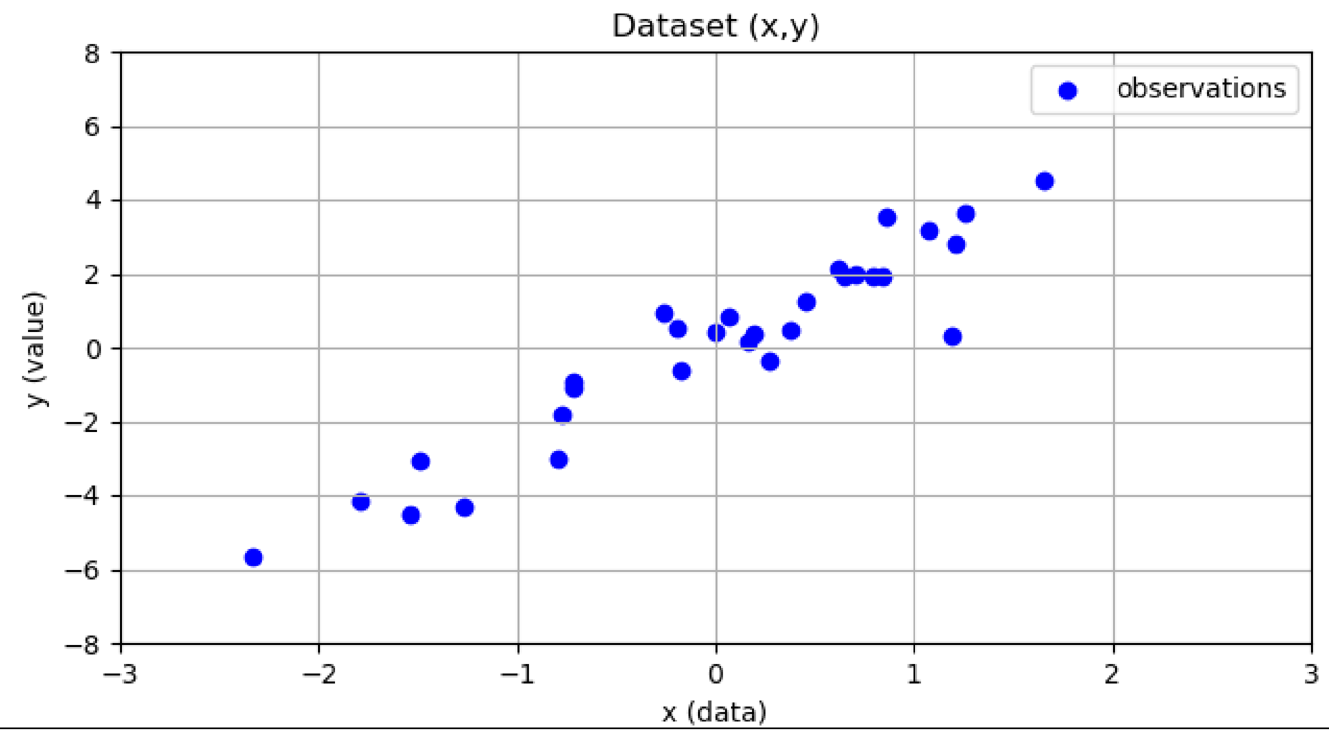
When the labels are quantitative variables, that is $\mathcal{Y} = \mathbb{R}^k$, $k \geqslant 1$, we say that we are addressing a regression task. In that case, a typical choice of loss function is $\ell(F(x), y) = ||F(x) - y||^2$, inducing the mean squared error.

As detailed in Chapter 4, the other fundamental scenario is when the labels are categorical variables (i.e. $\mathcal{Y}$ is finite; e.g. color, type of animal, etc.), in which case we say that we are addressing a classification problem.

Examples: Predict the age of someone $y \in \mathbb{R}$, a GPS position $(y_1, y_2) \in \mathbb{R}^2$, the price $y \in \mathbb{R}$ of an appartment... → Regression.
Predict if a drug is dangerous ($y = 0$) or not ($y = 1$), if a picture represent a `cat`, a `dog` or `else`... → Classification.

1. Linear regression.

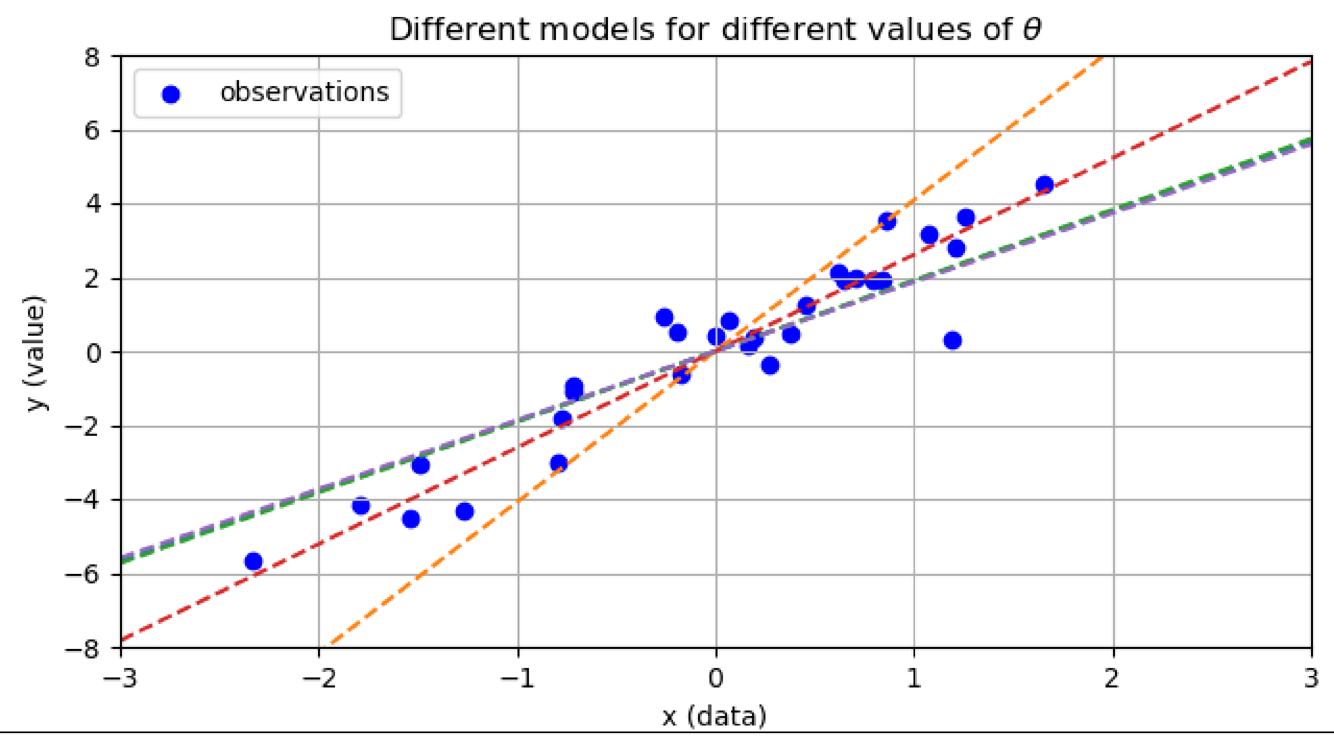
This is the simplest regression model one could consider. Consider observations $x_1, \dots, x_n \in \mathbb{R}$ (1D) with corresponding labels $y_1, \dots, y_n \in \mathbb{R}$ (1D as well). For instance: height and weight of someone.



1. Linear regression.

This is the simplest regression model one could consider. Consider observations $x_1, \ldots, x_n \in \mathbb{R}$ (1D) with corresponding labels $y_1, \ldots, y_n \in \mathbb{R}$ (1D as well). For instance: height and weight of someone.
The simplest model consists of assuming that y is mostly **proportional** to x, that is there exists $\theta \in \mathbb{R}$ such that $y \simeq \theta \cdot x = F_\theta(x)$. We say that F_θ is a parametric model.
The goal is to find **the best θ possible** with respect to the MSE. We thus want to minimize the objective function

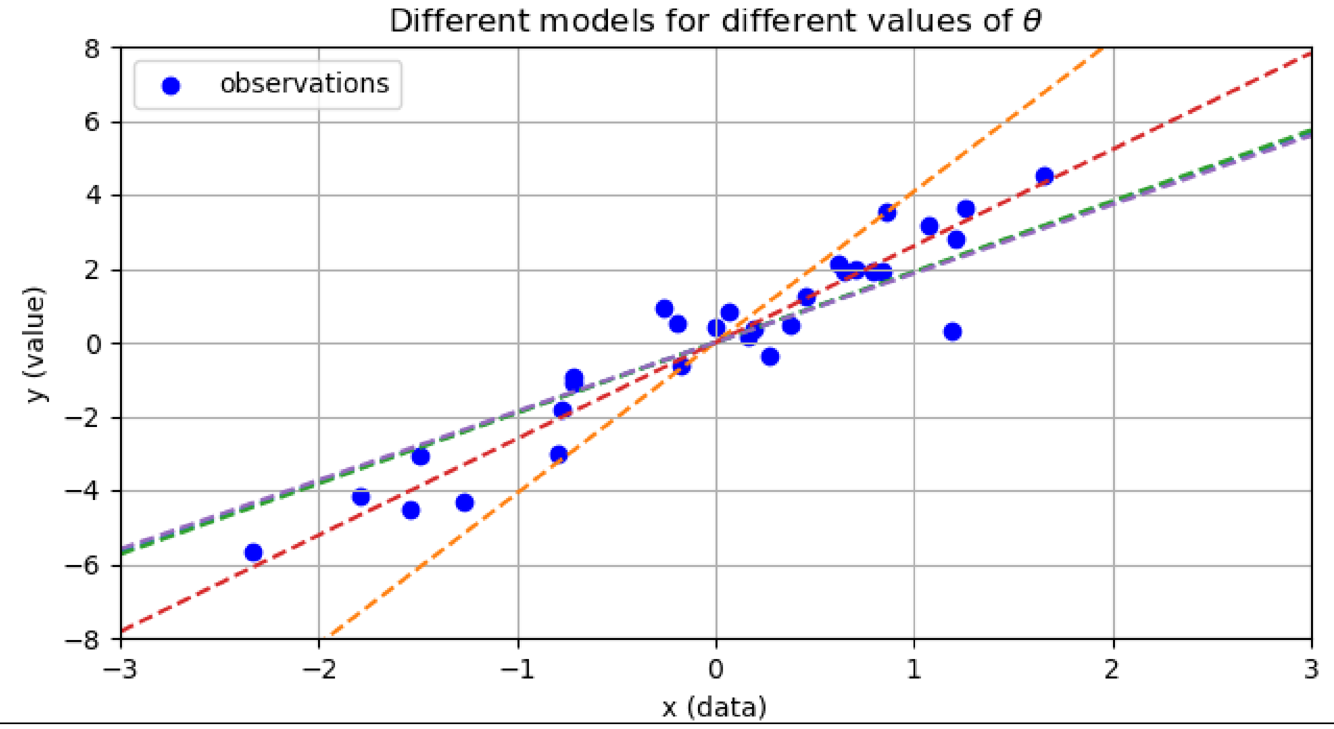
$$L : \theta \mapsto \frac{1}{n} \sum_{i=1}^{n} (\theta x_i - y_i)^2. \tag{2}$$



1. Linear regression.

This is the simplest regression model one could consider. Consider observations $x_1, \ldots, x_n \in \mathbb{R}$ (1D) with corresponding labels $y_1, \ldots, y_n \in \mathbb{R}$ (1D as well). For instance: height and weight of someone.
The simplest model consists of assuming that y is mostly **proportional** to x, that is there exists $\theta \in \mathbb{R}$ such that $y \simeq \theta \cdot x = F_\theta(x)$. We say that F_θ is a parametric model.
The goal is to find **the best θ possible** with respect to the MSE. We thus want to minimize the objective function

$$L : \theta \mapsto \frac{1}{n} \sum_{i=1}^{n} (\theta x_i - y_i)^2. \qquad (2)$$

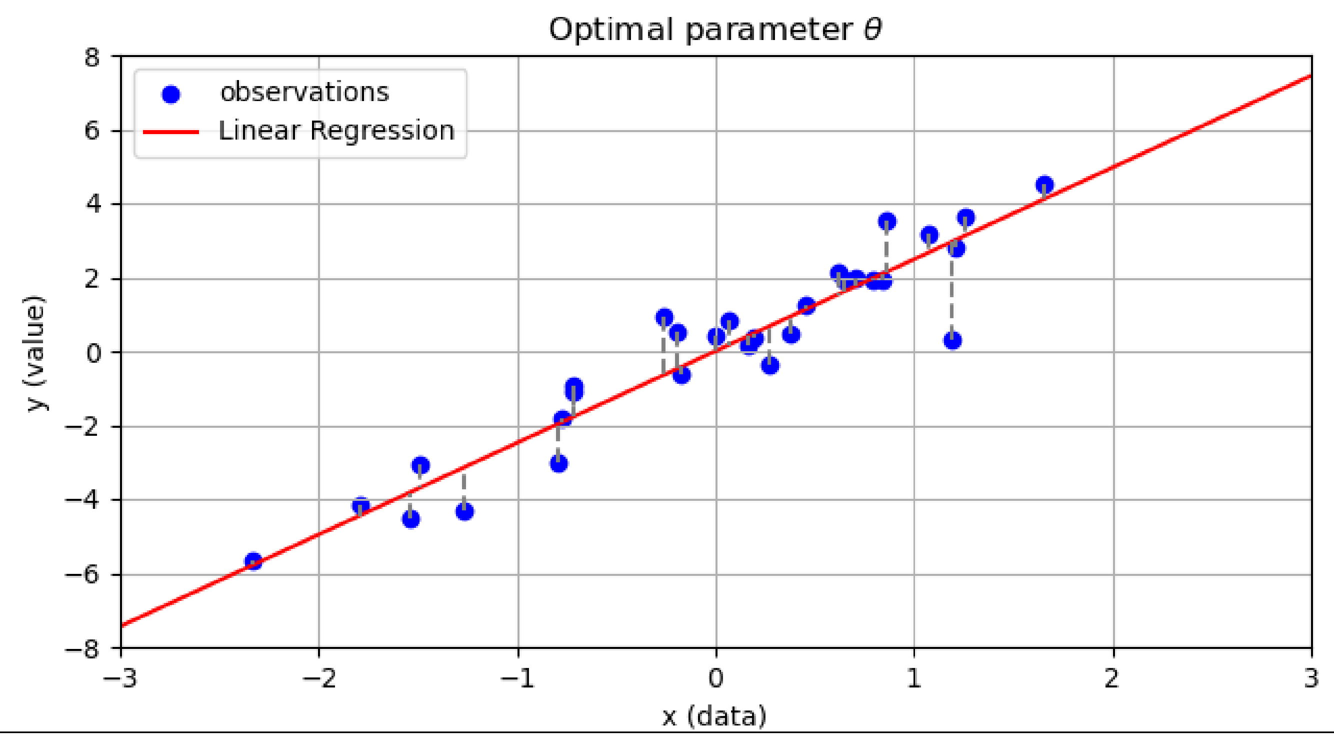


Exercise: Find the expression of the optimal θ.

1. Linear regression.

This is the simplest regression model one could consider. Consider observations $x_1, \dots, x_n \in \mathbb{R}$ (1D) with corresponding labels $y_1, \dots, y_n \in \mathbb{R}$ (1D as well). For instance: height and weight of someone.
The simplest model consists of assuming that y is mostly **proportional** to x, that is there exists $\theta \in \mathbb{R}$ such that $y \simeq \theta \cdot x = F_\theta(x)$. We say that F_θ is a parametric model.
The goal is to find **the best θ possible** with respect to the MSE. We thus want to minimize the objective function

$$L : \theta \mapsto \frac{1}{n} \sum_{i=1}^{n} (\theta x_i - y_i)^2. \quad (2)$$

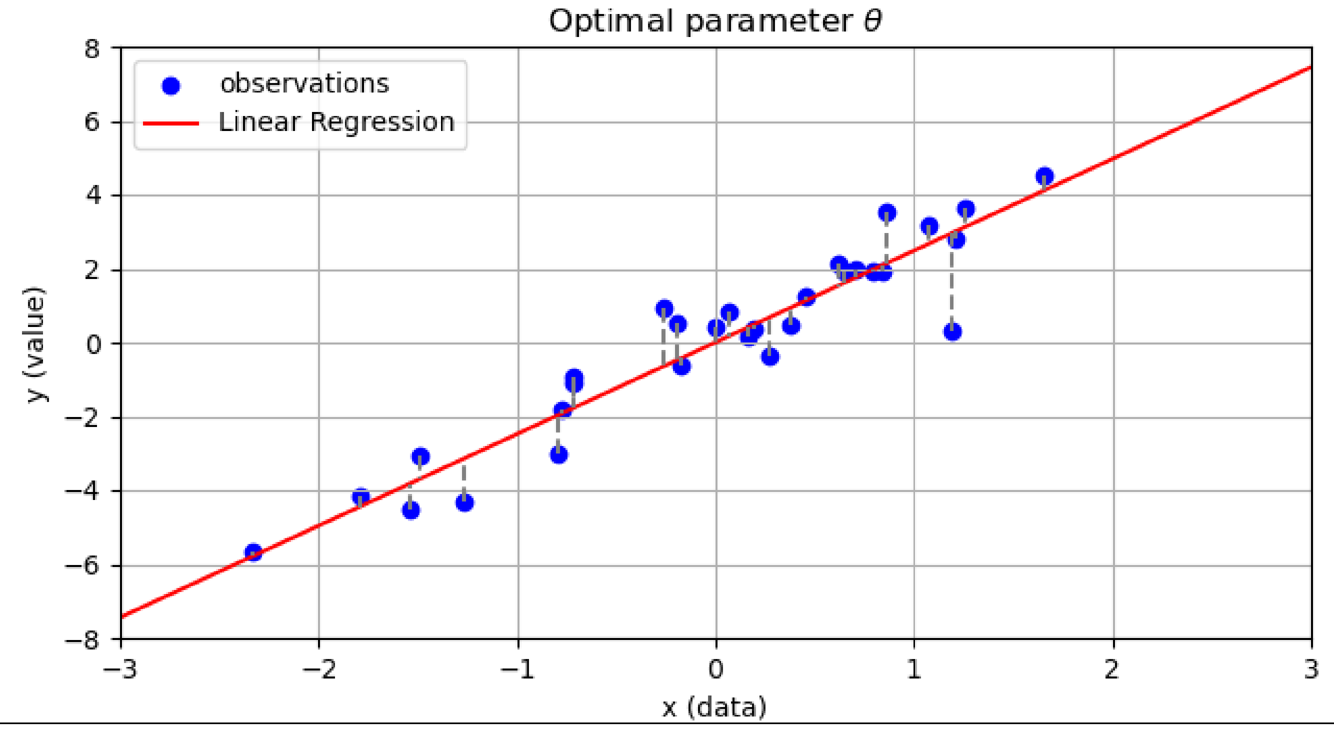


Exercise: Find the expression of the optimal θ.

1. Linear regression.

This is the simplest regression model one could consider. Consider observations $x_1, \ldots, x_n \in \mathbb{R}$ (1D) with corresponding labels $y_1, \ldots, y_n \in \mathbb{R}$ (1D as well). For instance: height and weight of someone.
The simplest model consists of assuming that y is mostly **proportional** to x, that is there exists $\theta \in \mathbb{R}$ such that $y \simeq \theta \cdot x = F_\theta(x)$. We say that F_θ is a parametric model.
The goal is to find **the best** θ **possible** with respect to the MSE. We thus want to minimize the objective function

$$L : \theta \mapsto \frac{1}{n} \sum_{i=1}^{n} (\theta x_i - y_i)^2. \tag{2}$$



Exercise: Find the expression of the optimal θ.

In Short:

Observations and labels are **fixed**, and learning is about optimizing the parameters of the model in order to minimize the training loss.

1. Linear regression.

The previous example is about observations and labels in 1D. We can generalize to more complex data (in higher dimension) in the following way:

Definition:

Let $x_1, \dots, x_n \in \mathbb{R}^d$ and $y_1, \dots, y_n \in \mathbb{R}^k$ be a dataset of observations and labels.
A linear regression is a model parametrized by a matrix $A \in \mathbb{R}^{k \times d}$ and a vector (called bias or intercept) $b \in \mathbb{R}^k$ of the form

$$F_{A,b}(x) = A \cdot x + b. \tag{3}$$

Training a linear regression amounts to minimizing the following objective function:

$$(A, b) \mapsto \sum_{i=1}^{n} \|Ax_i + b - y_i\|^2 \tag{4}$$

1. Linear regression.

The previous example is about observations and labels in 1D. We can generalize to more complex data (in higher dimension) in the following way:

Definition:

Let $x_1, \dots, x_n \in \mathbb{R}^d$ and $y_1, \dots, y_n \in \mathbb{R}^k$ be a dataset of observations and labels.
A linear regression is a model parametrized by a matrix $A \in \mathbb{R}^{k \times d}$ and a vector (called bias or intercept) $b \in \mathbb{R}^k$ of the form

$$F_{A,b}(x) = A \cdot x + b. \tag{3}$$

Training a linear regression amounts to minimizing the following objective function:

$$(A, b) \mapsto \sum_{i=1}^{n} \|Ax_i + b - y_i\|^2 \tag{4}$$

In Short:

We are looking for a linear combinaison of the features that allows us to retrieve the labels. For instance (random values for the sake of illustration), a linear regression may explain that the weight of somebody can be approximated by $2.4 \times$ height $+ 0.5 \times$ age $- 0.2 \times$ h sport / week $+ 1.2$. Here, $A = (2.4, 0.5, -0.2) \in \mathbb{R}^{3\times 1}$ and $b = 1.2 \in \mathbb{R}$ (because our labels are in dimension 1).

1. Linear regression.

The previous example is about observations and labels in 1D. We can generalize to more complex data (in higher dimension) in the following way:

Definition:

Let $x_1, \dots, x_n \in \mathbb{R}^d$ and $y_1, \dots, y_n \in \mathbb{R}^k$ be a dataset of observations and labels.
A linear regression is a model parametrized by a matrix $A \in \mathbb{R}^{k \times d}$ and a vector (called bias or intercept) $b \in \mathbb{R}^k$ of the form

$$F_{A,b}(x) = A \cdot x + b. \tag{3}$$

Training a linear regression amounts to minimizing the following objective function:

$$(A, b) \mapsto \sum_{i=1}^{n} \|Ax_i + b - y_i\|^2 \tag{4}$$

Remark: Observe that $Ax + b = (A, b) \cdot \begin{pmatrix} x \\ 1 \end{pmatrix}$. Therefore, the bias term can be encompassed in the matrix A by "augmenting" the training observations (adding a 1 as last coordinate).
→ In a nutshell, the bias can be ignored in theoretical analysis (and is often automatically added in implementation).

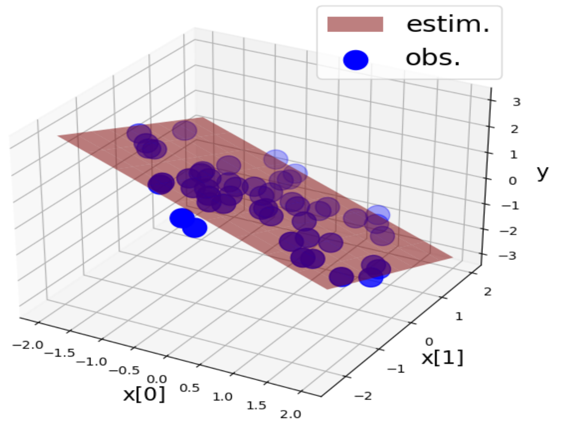
1. Linear regression.

Theorem:

Given a dataset $X = \begin{pmatrix} x_1[1] & \dots & x_1[d] & 1 \\ & \vdots & & \\ x_n[1] & \dots & x_n[d] & 1 \end{pmatrix} \in \mathbb{R}^{n\times(d+1)}$ and $y = \begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix} \in \mathbb{R}^{n\times k}$. Assume that $X^T X$ is non-singular (invertible). Let $M = \begin{pmatrix} A \\ b \end{pmatrix} \in \mathbb{R}^{(d+1)\times k}$.

The optimal parameter M^* for the linear regression of X, Y—that is the minimizer of $L: M \mapsto \|XM - y\|_2^2$, where $\|U\|_2^2 = \mathrm{Tr}(UU^T)$ denotes the (squared) Froebenius norm of a matrix—is given by

$$M^* = (X^T X)^{-1} X^T y.$$



1. Linear regression.

Theorem:

Given a dataset $X = \begin{pmatrix} x_1[1] & \dots & x_1[d] & 1 \\ & \vdots & & \\ x_n[1] & \dots & x_n[d] & 1 \end{pmatrix} \in \mathbb{R}^{n \times (d+1)}$ and $y = \begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix} \in \mathbb{R}^{n \times k}$. Assume that $X^T X$ is non-singular (invertible). Let $M = \begin{pmatrix} A \\ b \end{pmatrix} \in \mathbb{R}^{(d+1) \times k}$.

The optimal parameter M^* for the linear regression of X, Y—that is the minimizer of $L \colon M \mapsto \|XM - y\|_2^2$, where $\|U\|_2^2 = \mathrm{Tr}(UU^T)$ denotes the (squared) Froebenius norm of a matrix—is given by

$$M^* = (X^T X)^{-1} X^T y.$$

Exercise: Prove this theorem.

Interpret the assumption "$X^T X$ is invertible" in three different ways:

- In algebraic terms (what can you say about the equation satisfied by M^*?),
- In analytic terms (what can you say about the loss function L?),
- In geometric terms ("what's the shape of X?").

1. Linear regression.

Theorem:

Given a dataset $X = \begin{pmatrix} x_1[1] & \dots & x_1[d] & 1 \\ & \vdots & & \\ x_n[1] & \dots & x_n[d] & 1 \end{pmatrix} \in \mathbb{R}^{n \times (d+1)}$ and $y = \begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix} \in \mathbb{R}^{n \times k}$. Assume that $X^T X$ is non-singular (invertible). Let $M = \begin{pmatrix} A \\ b \end{pmatrix} \in \mathbb{R}^{(d+1) \times k}$.

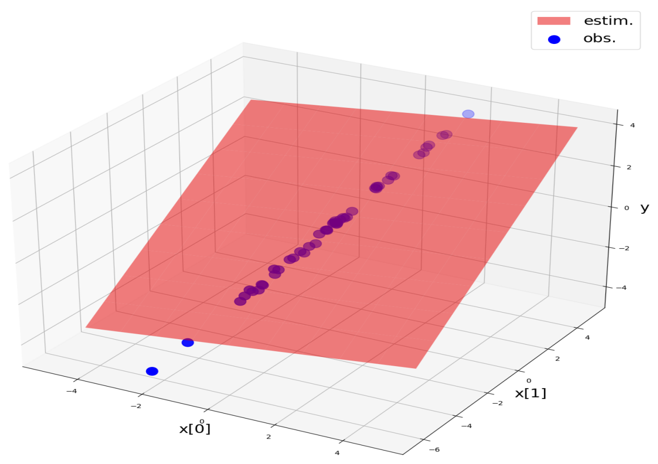
The optimal parameter M^* for the linear regression of X, Y—that is the minimizer of $L \colon M \mapsto \|XM - y\|_2^2$, where $\|U\|_2^2 = \mathrm{Tr}(UU^T)$ denotes the (squared) Froebenius norm of a matrix—is given by

$$M^* = (X^T X)^{-1} X^T y.$$

Exercise: Prove this theorem.

Interpret the assumption "$X^T X$ is invertible" in three different ways:

- In algebraic terms (what can you say about the equation satisfied by M^*?),
- In analytic terms (what can you say about the loss function L?),
- In geometric terms ("what's the shape of X?").



1. Linear regression.

Theorem:

Given a dataset $X = \begin{pmatrix} x_1[1] & \dots & x_1[d] & 1 \\ & \vdots & & \\ x_n[1] & \dots & x_n[d] & 1 \end{pmatrix} \in \mathbb{R}^{n\times(d+1)}$ and $y = \begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix} \in \mathbb{R}^{n\times k}$. Assume that $X^T X$ is non-singular (invertible). Let $M = \begin{pmatrix} A \\ b \end{pmatrix} \in \mathbb{R}^{(d+1)\times k}$.

The optimal parameter M^* for the linear regression of X, Y—that is the minimizer of $L\colon M \mapsto \|XM - y\|_2^2$, where $\|U\|_2^2 = \mathrm{Tr}(UU^T)$ denotes the (squared) Froebenius norm of a matrix—is given by

$$M^* = (X^T X)^{-1} X^T y.$$

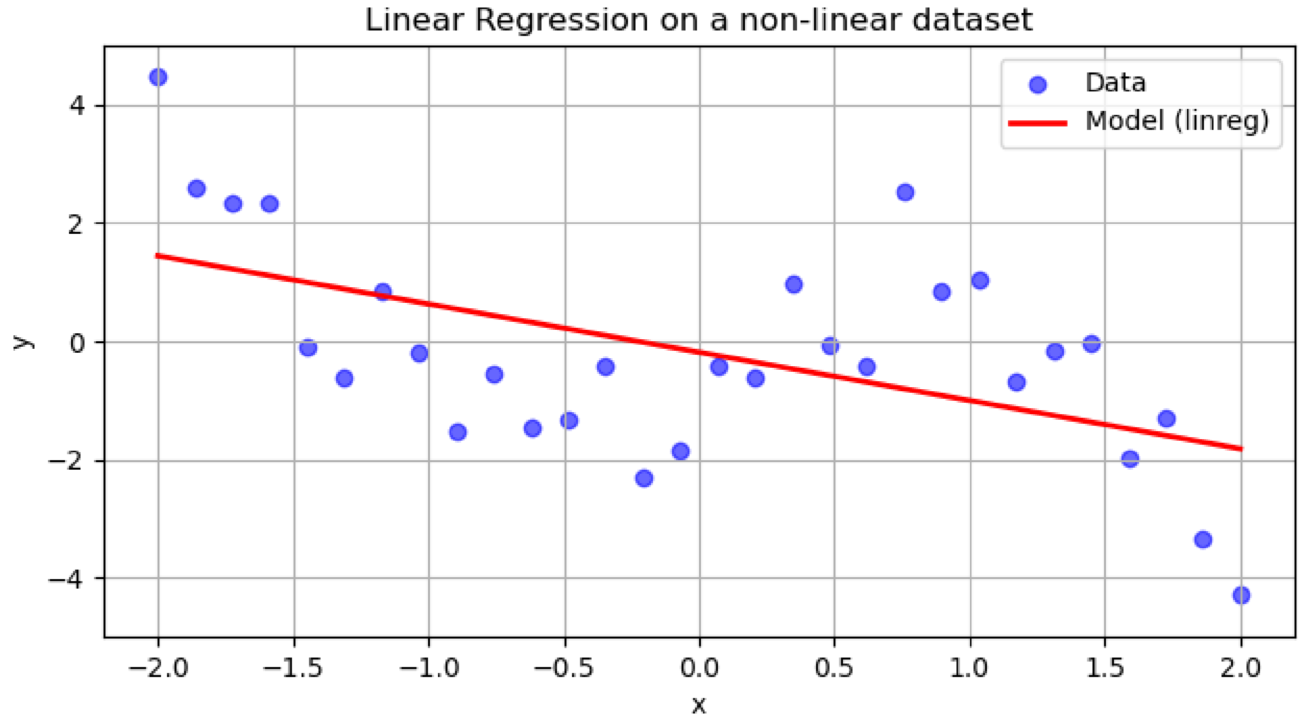
In Short:

We have access to a **closed form** for the optimal parameter of a linear regression. We will see that this is not always the case when dealing with more complicated models. It is thus **easy** to numerically solve this problem.

In practice: You can use the class `LinearRegression()` of the module `sklearn.linear_model`.

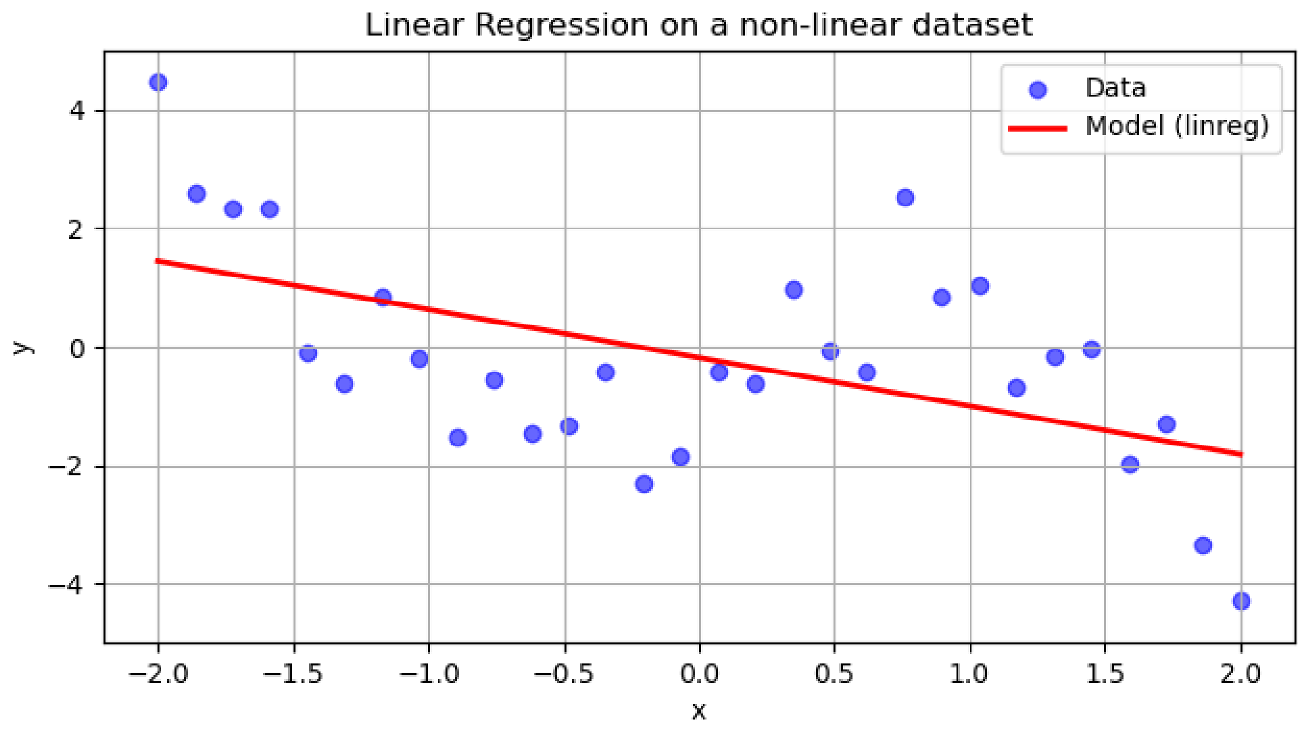
2. Polynomial regression

Of course, there is no reason to always assume that our data follow a linear relation $y \simeq Ax$.



2. Polynomial regression

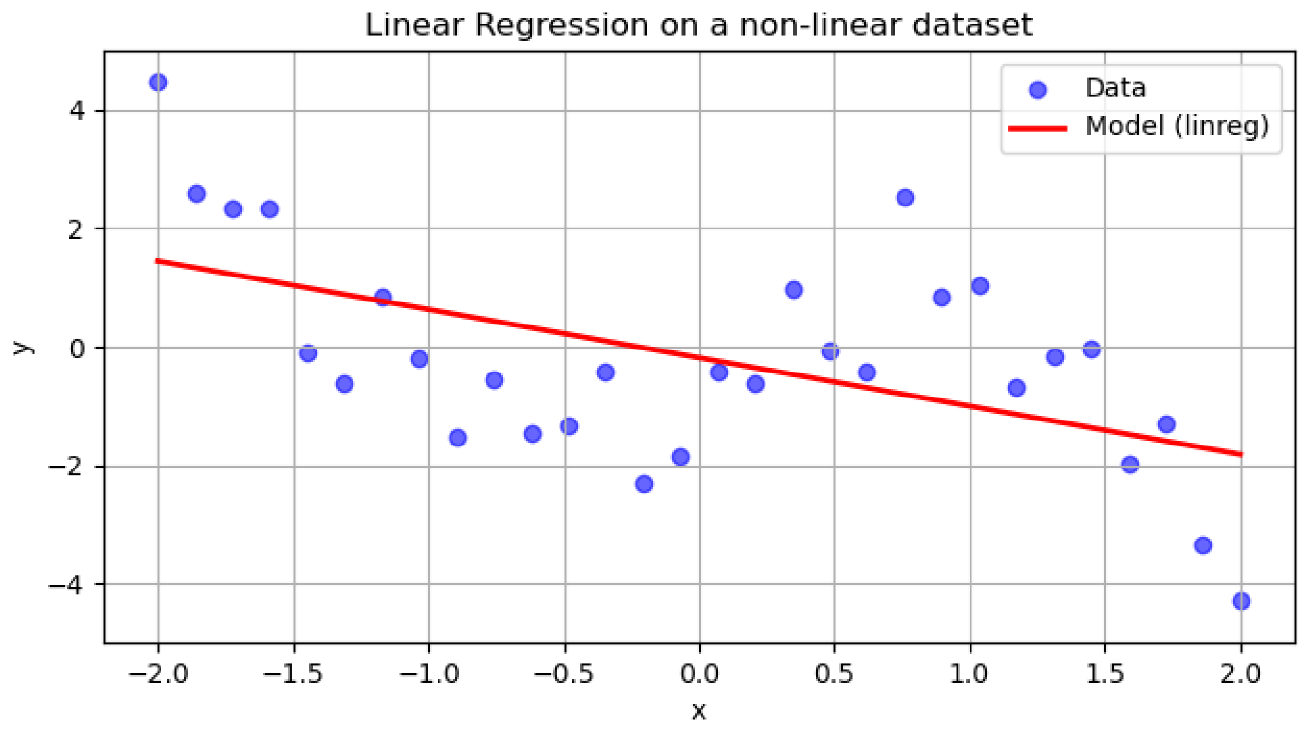
Of course, there is no reason to always assume that our data follow a linear relation $y \simeq Ax$.



It is natural to generalize the linear regression to have more expressive models (able to learn more subtle relations).

2. Polynomial regression

Of course, there is no reason to always assume that our data follow a linear relation $y \simeq Ax$.



It is natural to generalize the linear regression to have more expressive models (able to learn more subtle relations).

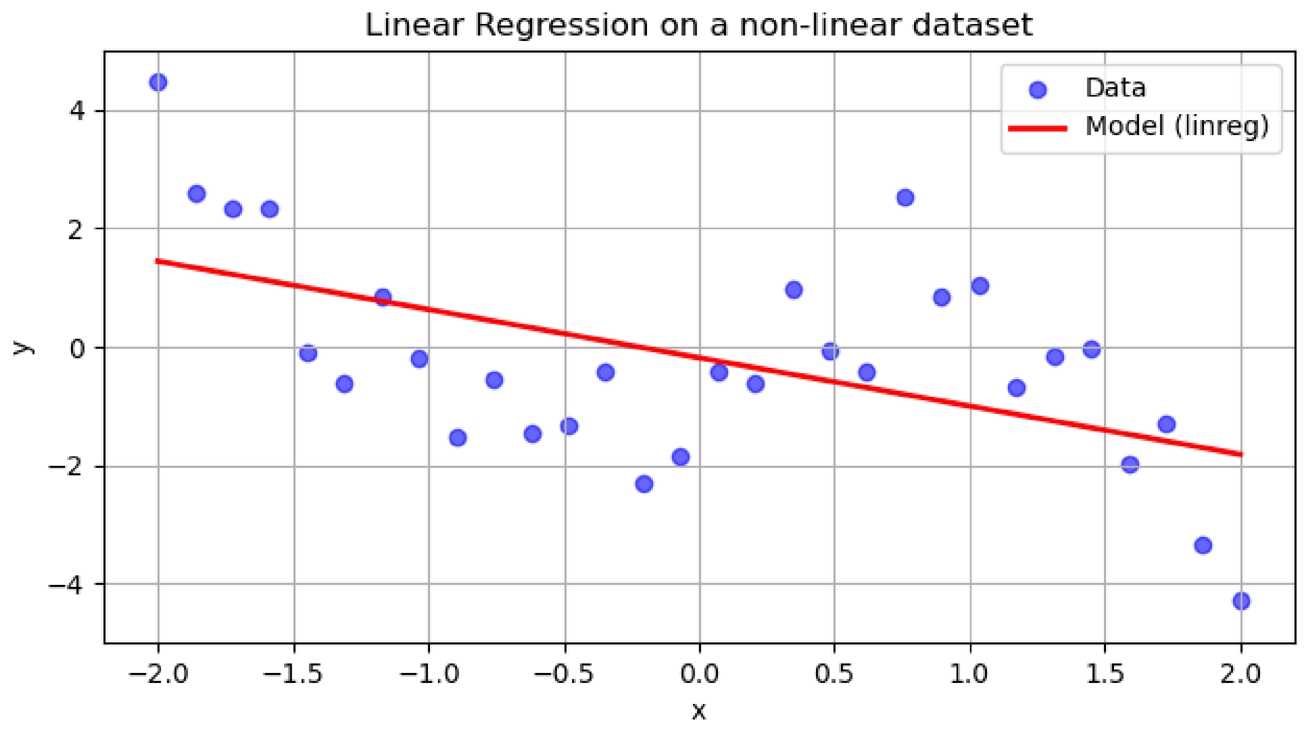
Definition:

Assume that $\mathcal{X} = \mathbb{R}$ and $\mathcal{Y} = \mathbb{R}$ (observations and labels in dimension 1). A polynomial regression of degree p consists of training a model F depending on $p+1$ parameters $\theta = (\theta_0, \dots, \theta_p) \in \mathbb{R}^{p+1}$ of the form

$$F_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \cdots + \theta_p x^p. \tag{5}$$

2. Polynomial regression

Of course, there is no reason to always assume that our data follow a linear relation $y \simeq Ax$.



It is natural to generalize the linear regression to have more expressive models (able to learn more subtle relations).

Definition:

Assume that $\mathcal{X} = \mathbb{R}$ and $\mathcal{Y} = \mathbb{R}$ (observations and labels in dimension 1). A polynomial regression of degree p consists of training a model F depending on $p+1$ parameters $\theta = (\theta_0, \dots, \theta_p) \in \mathbb{R}^{p+1}$ of the form

$$F_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \cdots + \theta_p x^p. \quad (5)$$

But... if we let $x' = (1, x, \dots, x^p) \in \mathbb{R}^{p+1}$, the problem boils down to a linear regression of dimension $d = p+1$ for the observations, and $k = 1$ for the labels! We can thus find the optimal θ using the previous theorem on this "augmented" dataset.

2. Polynomial regression

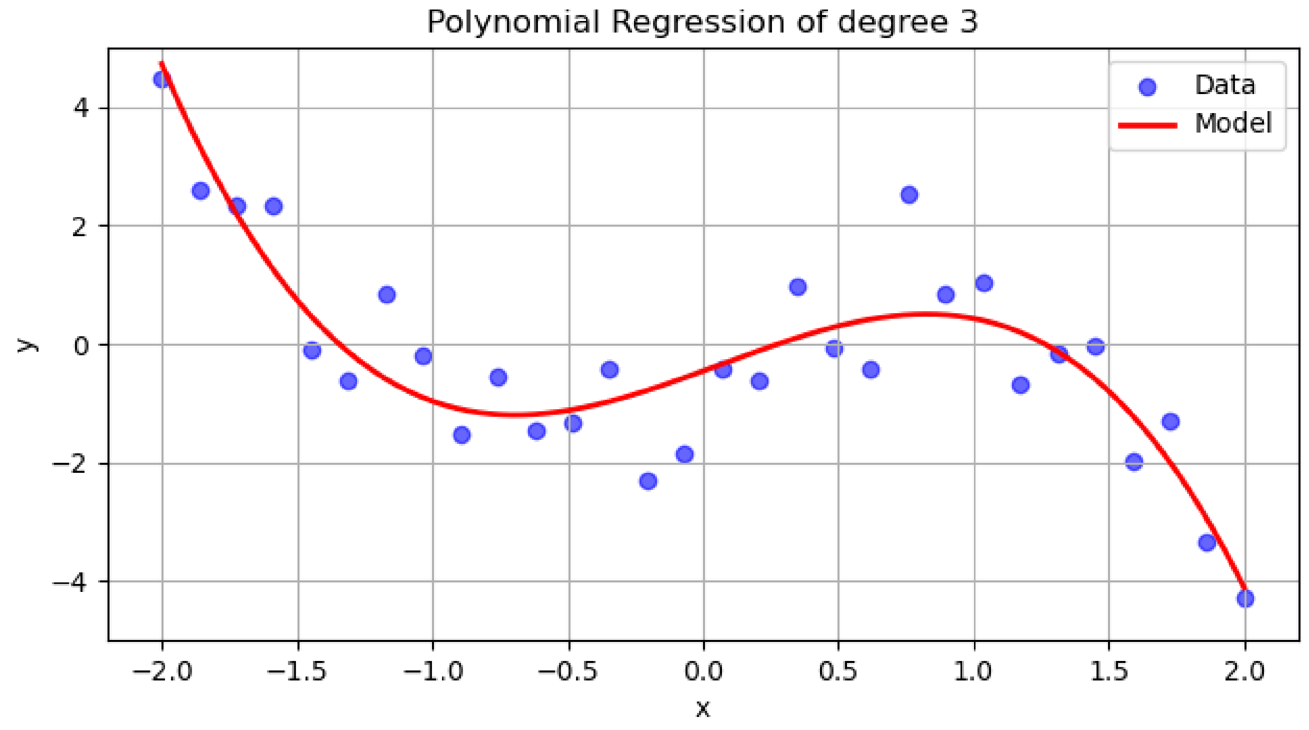
In practice: We build the “augmented data” $x' = (1, x, x^2, \dots, x^p)$ using the class `PolynomialFeatures()` of `sklearn.preprocessing`, then simply run a `LinearRegression()`.

2. Polynomial regression

In practice: We build the "augmented data" $x' = (1, x, x^2, \ldots, x^p)$ using the class `PolynomialFeatures()` of `sklearn.preprocessing`, then simply run a `LinearRegression()`.

```
degree = 3
# Create a pipeline for polynomial regression
model = make_pipeline(PolynomialFeatures(degree), LinearRegression())
model.fit(x, y)
# Predict over a fine grid
x_fit = np.linspace(-2, 2, 500).reshape(-1, 1)
y_fit = model.predict(x_fit)
```

Easy!



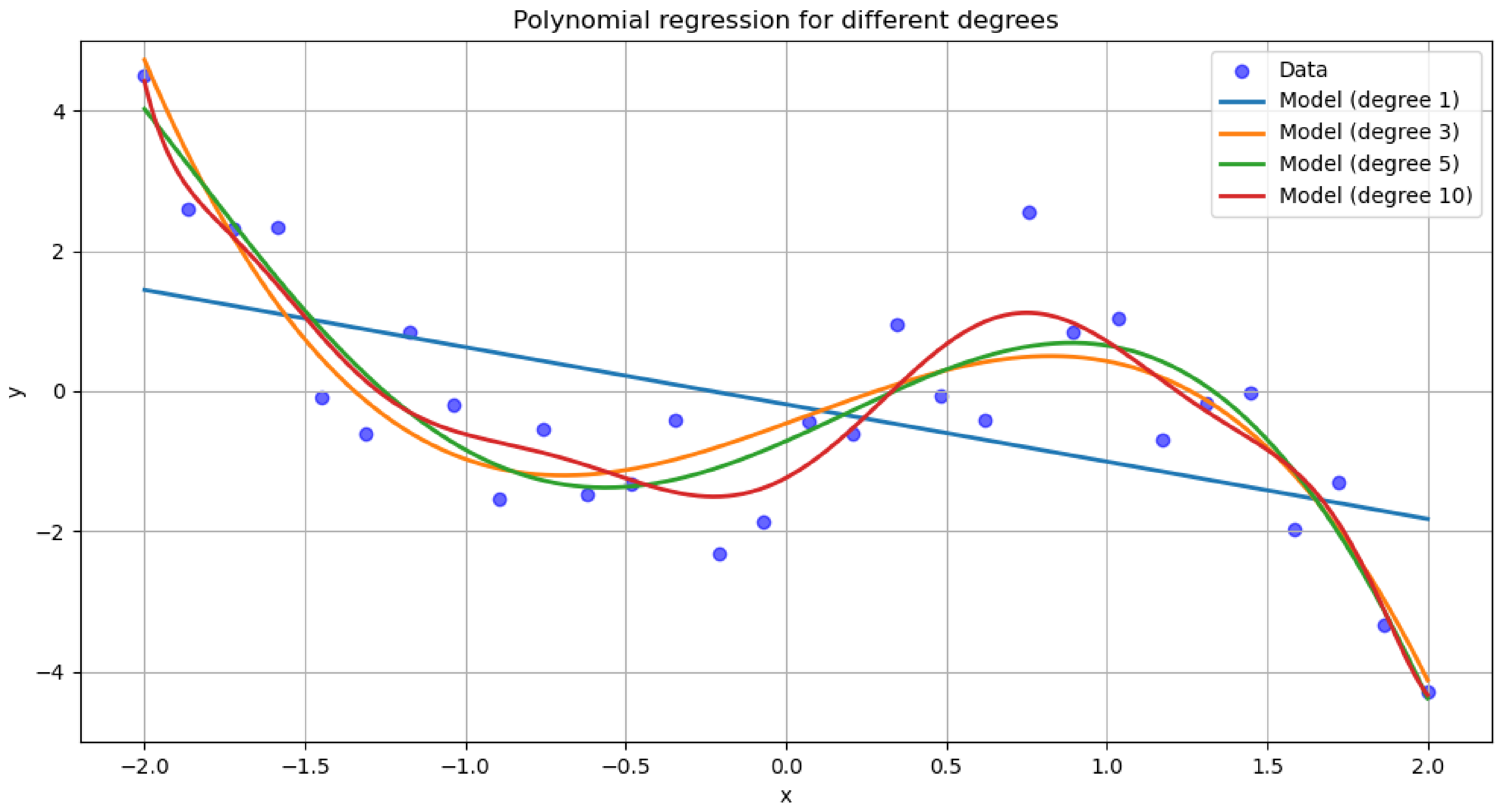
2. Polynomial regression

Last step: chose the degree p...

Note: p is **not** optimized during the training. This is a parameter that is chosen by the user (you) **from the start**. Such parameters (not optimized) are called hyperparameters.

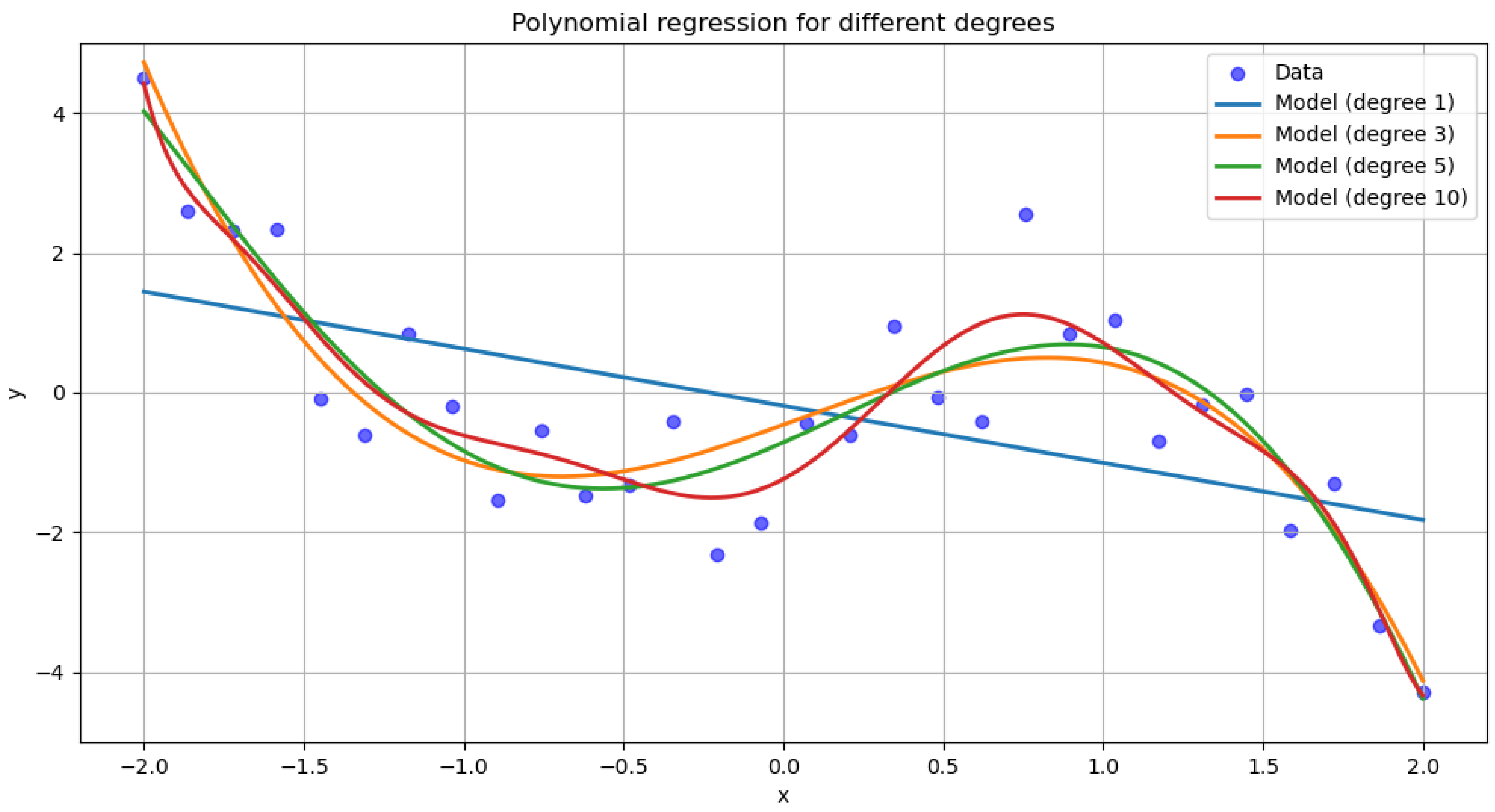
2. Polynomial regression

Last step: chose the degree p...



2. Polynomial regression

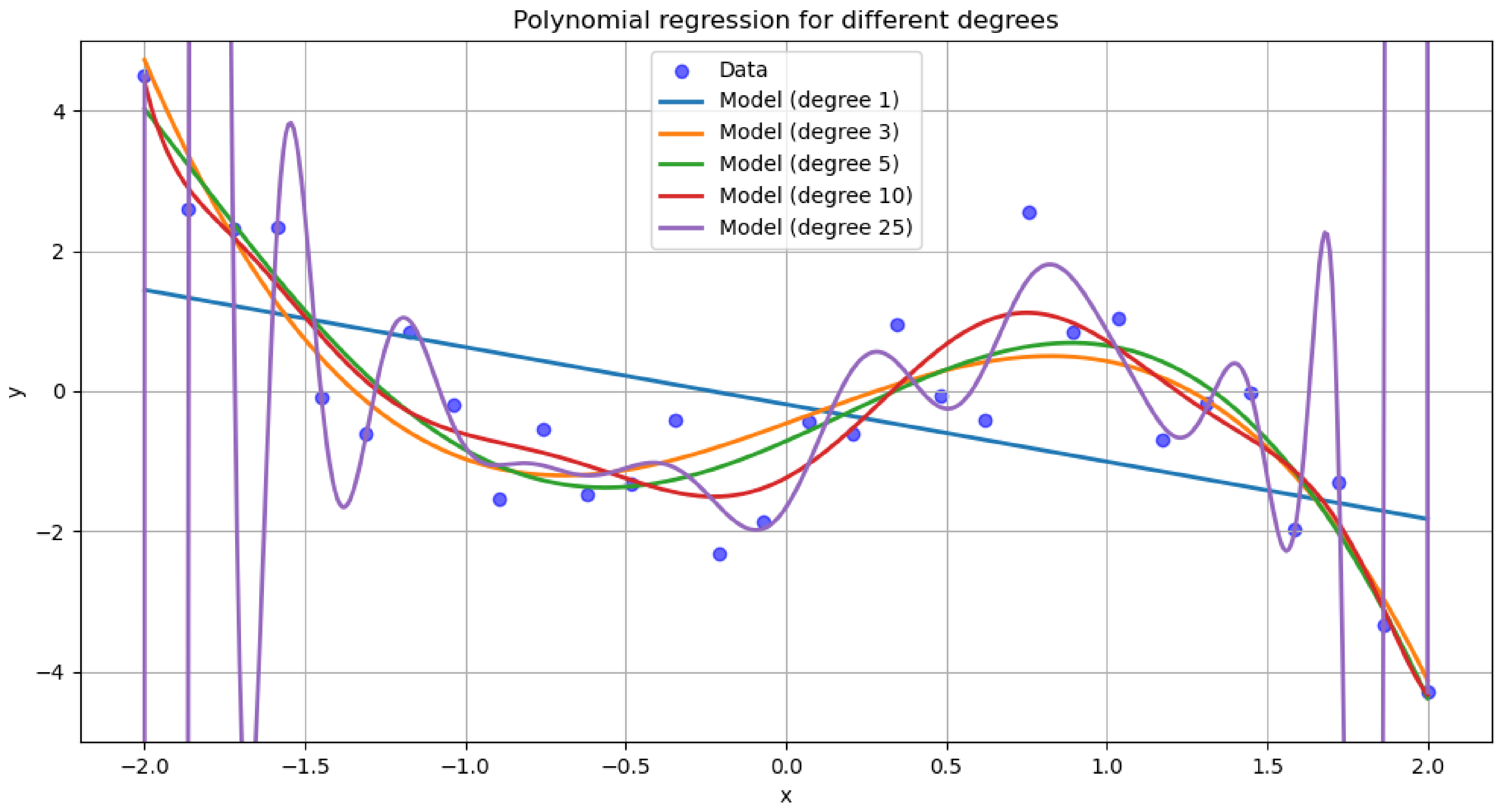
Last step: chose the degree p...



Exercise: Prove that increasing the maximal degree d always decreases the objective loss after training.

2. Polynomial regression

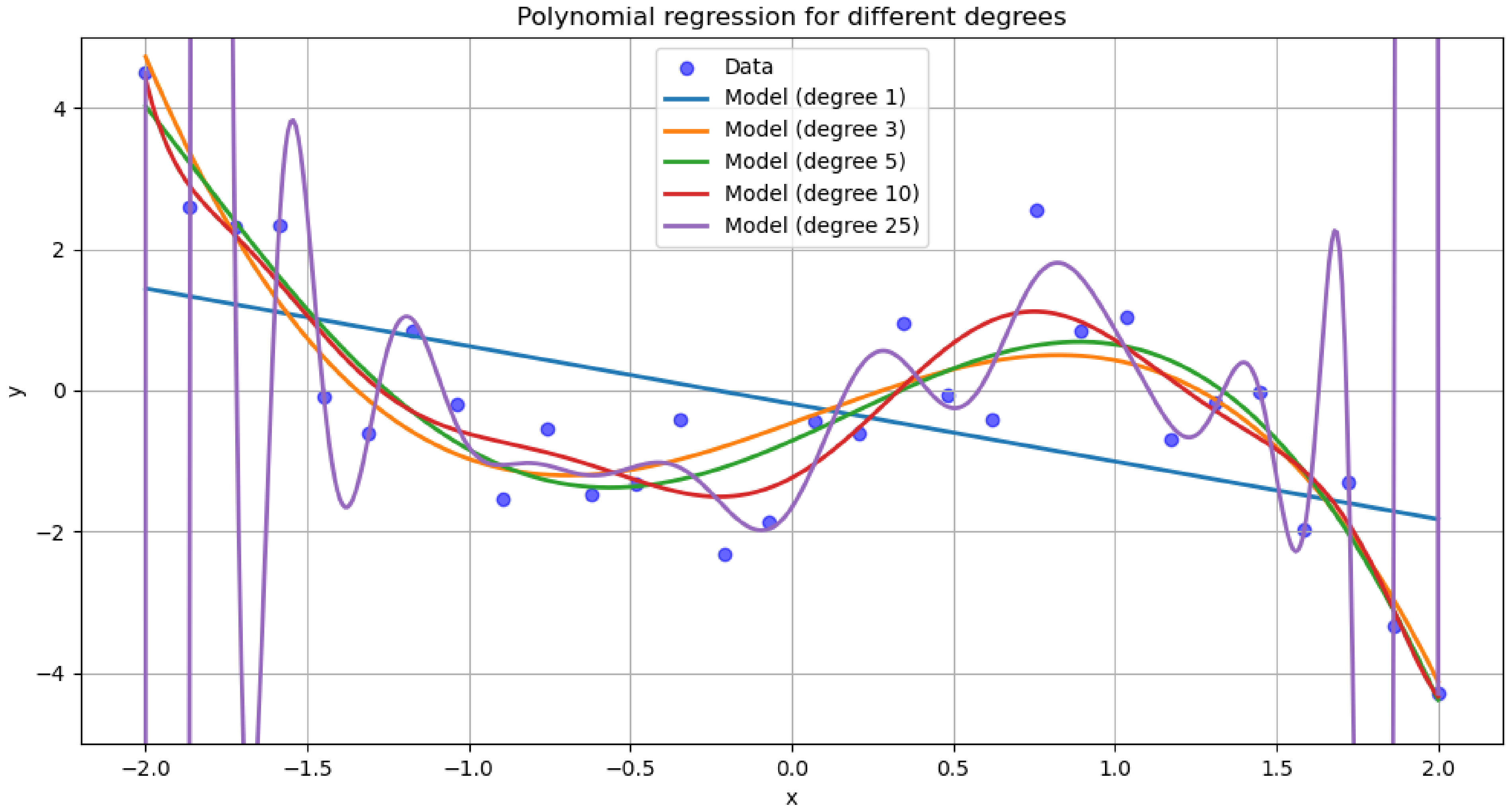
Last step: chose the degree p...



Exercise: Prove that increasing the maximal degree d always decreases the objective loss after training.

2. Polynomial regression

Last step: chose the degree p...

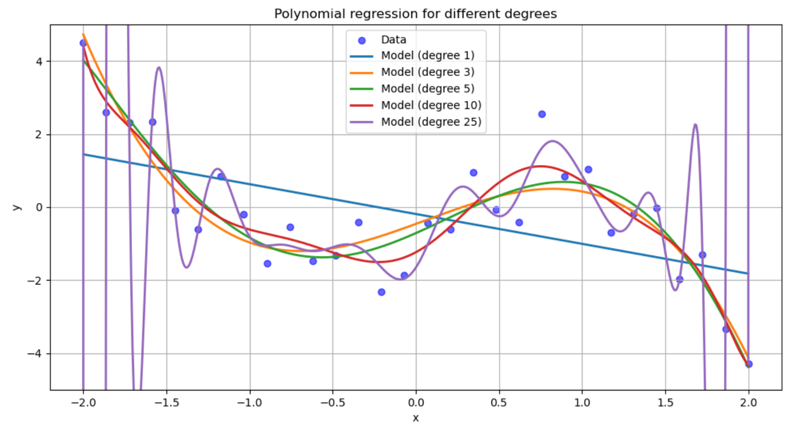


Exercise: Prove that increasing the maximal degree d always decreases the objective loss after training.

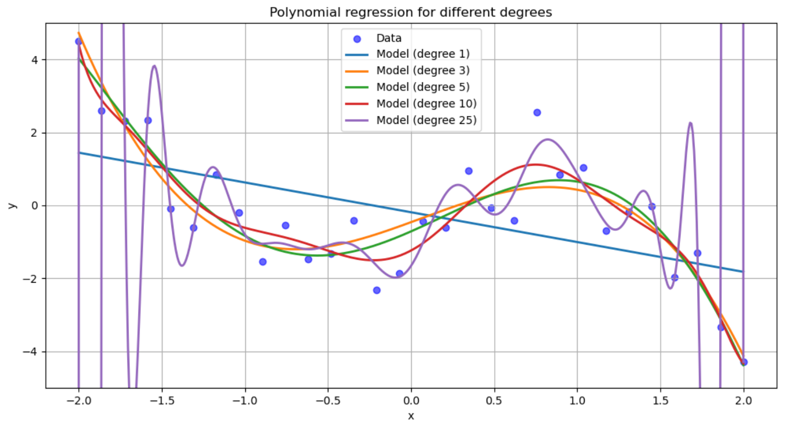
Core idea: Increasing the complexity ($\sim$ number of parameters) of a model will always make it **more expressive** : the loss will always get smaller on the **training data** if we minimize over a larger class of models.
Here, the set of polynomials of degree $\leqslant 15$ is larger and can better adapt to the training data that polynomials of degree $\leqslant 2$. It does not mean that this model is better/more useful in practical application, even though the loss is smaller! How to handle that?

3. Training, Testing and Overfitting...

Core idea: What is the real issue with the (optimal) polynomial of degree $p = 25$? Why is it not useful in practice?



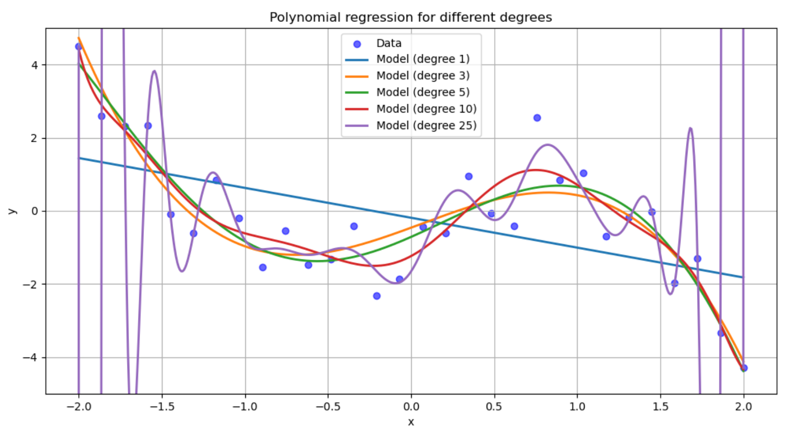
3. Training, Testing and Overfitting...



Core idea: What is the real issue with the (optimal) polynomial of degree $p = 25$? Why is it not useful in practice?

Because it cannot generalize.

3. Training, Testing and Overfitting...



Core idea: What is the real issue with the (optimal) polynomial of degree $p = 25$? Why is it not useful in practice?

Because it cannot generalize.

Definition:

We say that a model can generalize if it can produce **valid predictions** on **new** data that are following the same law Γ as the training observations.
In practice, we split **randomly** our observations in **two groups** :

- The training set: the one that will be used to optimize the parameters of our models by minimizing the *training loss* L_{train}.
- The test set (or validation set) on which we simply evaluate the performance of the model (test/validation loss).

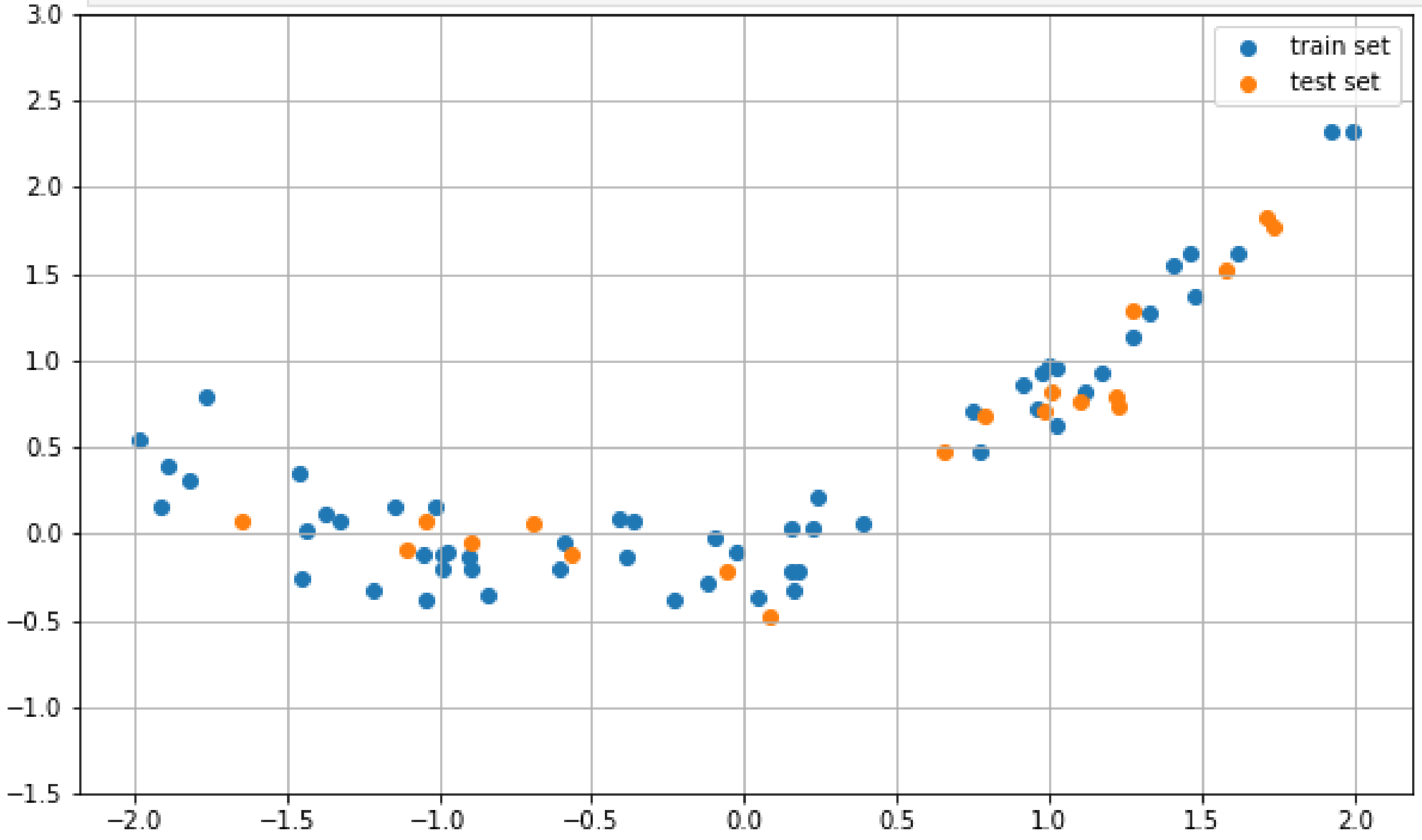
Whenever the training loss is small but the test loss is high, we say that our model is overfitting.

3. Training, Testing and Overfitting...

In practice : You can use the method `train_test_split` of the module `sklearn.model_selection`. A common practice is to put 75% of the data in the train set and 25% in the test set.

```
from sklearn.model_selection import train_test_split
```

```
# Sépare par défaut avec 75% de train et 25% de test.
x_train, x_test, y_train, y_test = train_test_split(x, y)
```

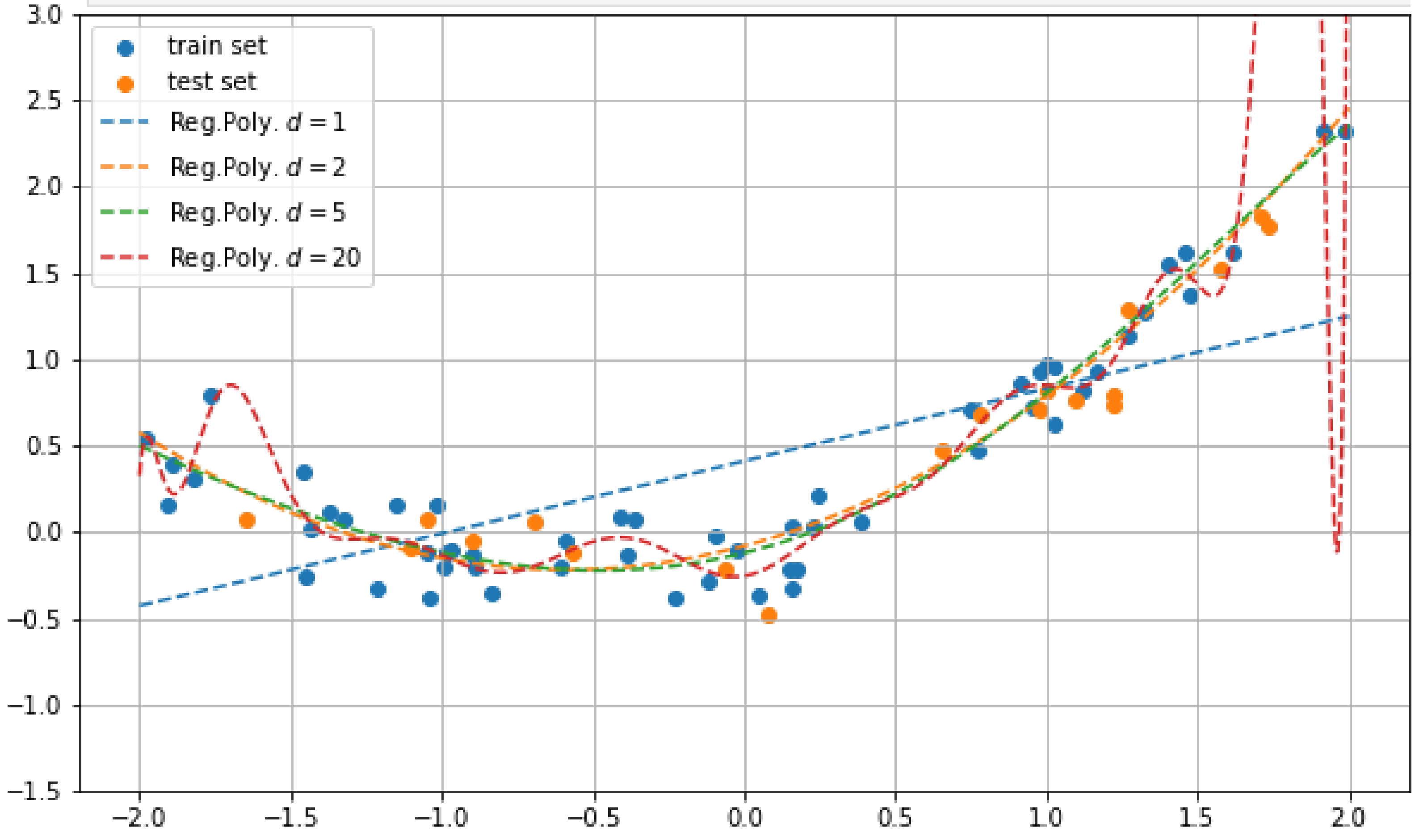


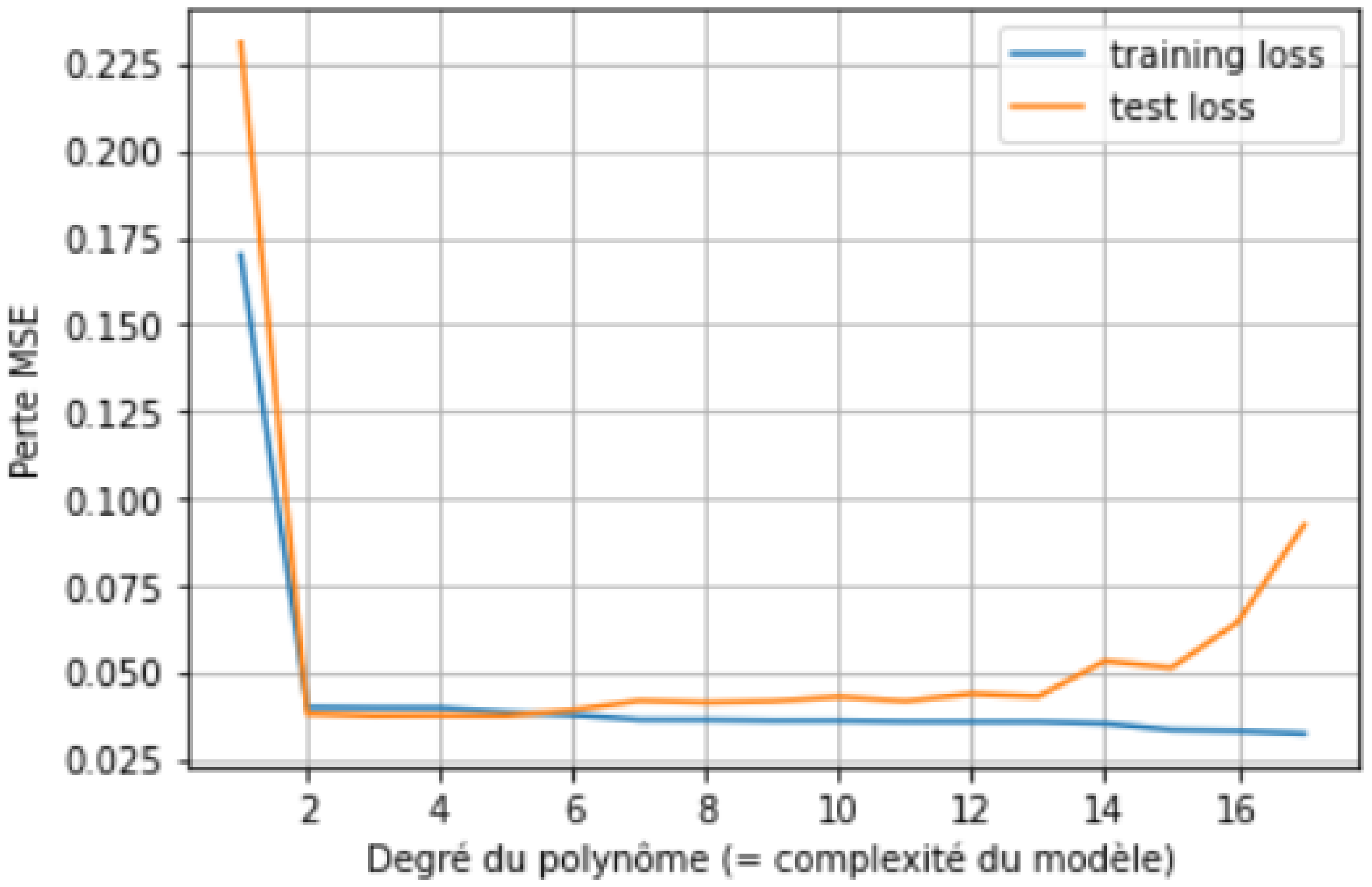
3. Training, Testing and Overfitting...

In practice : You can use the method `train_test_split` of the module `sklearn.model_selection`. A common practice is to put 75% of the data in the train set and 25% in the test set.

```
from sklearn.model_selection import train_test_split
```

```
# Sépare par défaut avec 75% de train et 25% de test.
x_train, x_test, y_train, y_test = train_test_split(x, y)
```



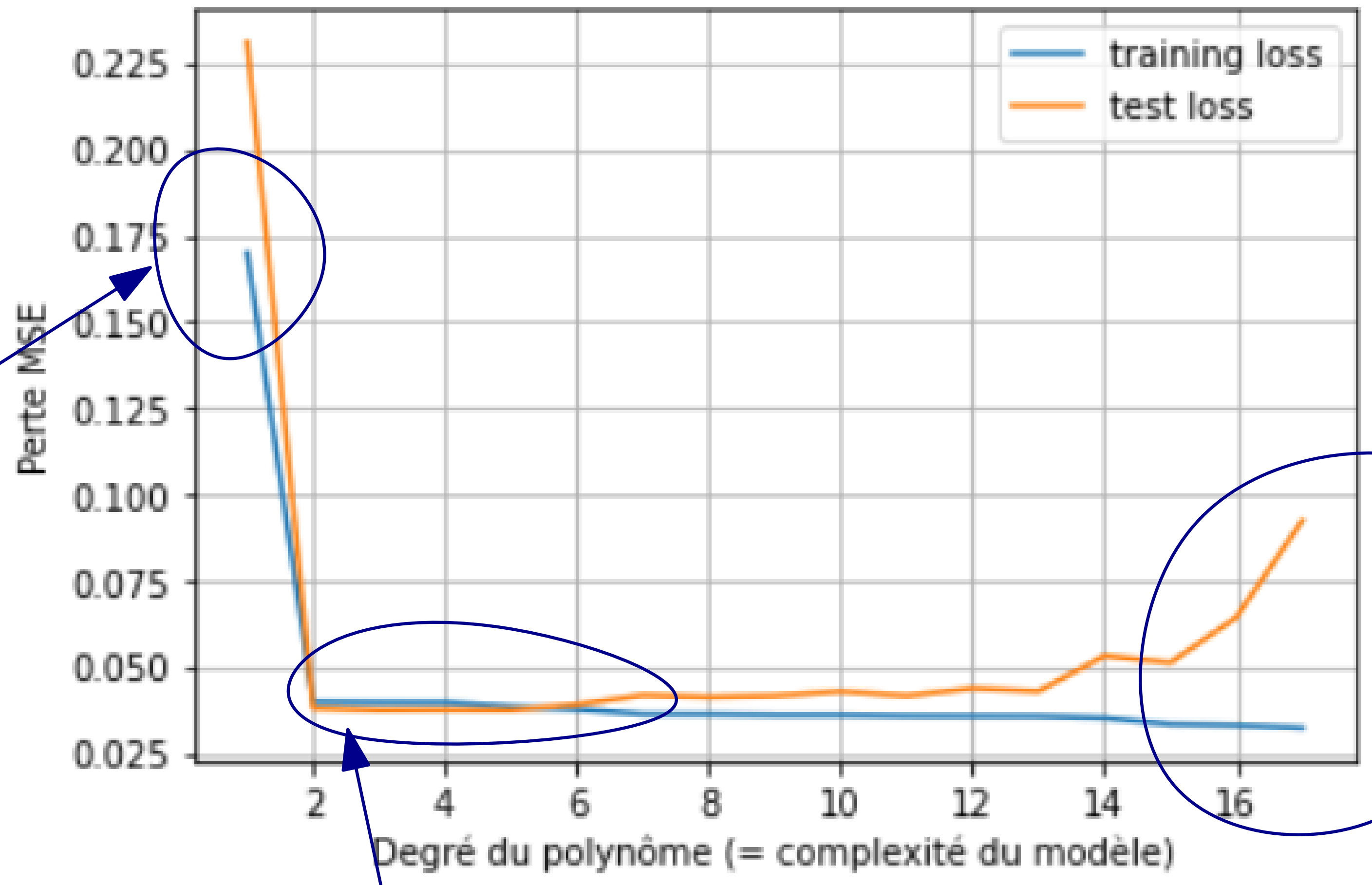


3. Training, Testing and Overfitting...

In practice : You can use the method `train_test_split` of the module `sklearn.model_selection`. A common practice is to put 75% of the data in the train set and 25% in the test set.

```
from sklearn.model_selection import train_test_split
```

```
# Sépare par défaut avec 75% de train et 25% de test.
x_train, x_test, y_train, y_test = train_test_split(x, y)
```



The test loss is much larger than the training loss: this is overfitting due to the model being too complex.

High training loss: the model is too simple to properly learn the relationship between observations and labels.

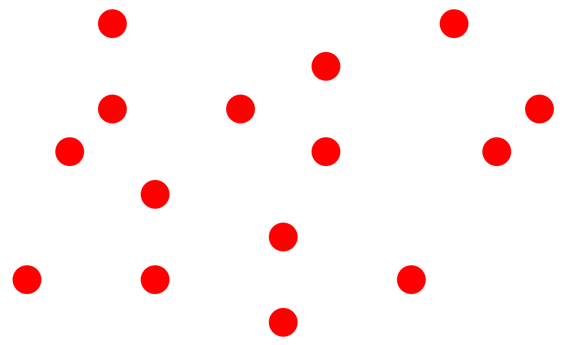
The training and test losses are low and of similar order of magnitude: this is promising!

3. Training, Testing and Overfitting...

In Short:

Once you have trained your model and provided that it achieves a reasonnably low training loss, you **must** test it by looking at its performances on observations that were not seen during the training phase (but distributed similarly to the training set).

Step 0: Collect observations and labels $(x_i, y_i)_i$



Data (obs+labels)

3. Training, Testing and Overfitting...

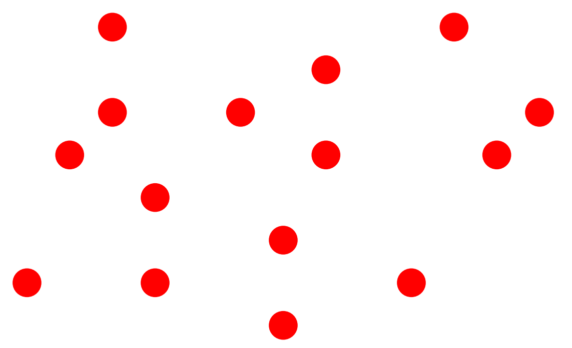
In Short:

Once you have trained your model and provided that it achieves a reasonnably low training loss, you **must** test it by looking at its performances on observations that were not seen during the training phase (but distributed similarly to the training set).

Step 0: Collect observations and labels $(x_i, y_i)_i$

↓

Step 1: Chose a class of models (e.g. linear regressions)



Data (obs+labels)

$$(F_\theta)_\theta$$

(Class of) model

3. Training, Testing and Overfitting...

In Short:

Once you have trained your model and provided that it achieves a reasonnably low training loss, you **must** test it by looking at its performances on observations that were not seen during the training phase (but distributed similarly to the training set).

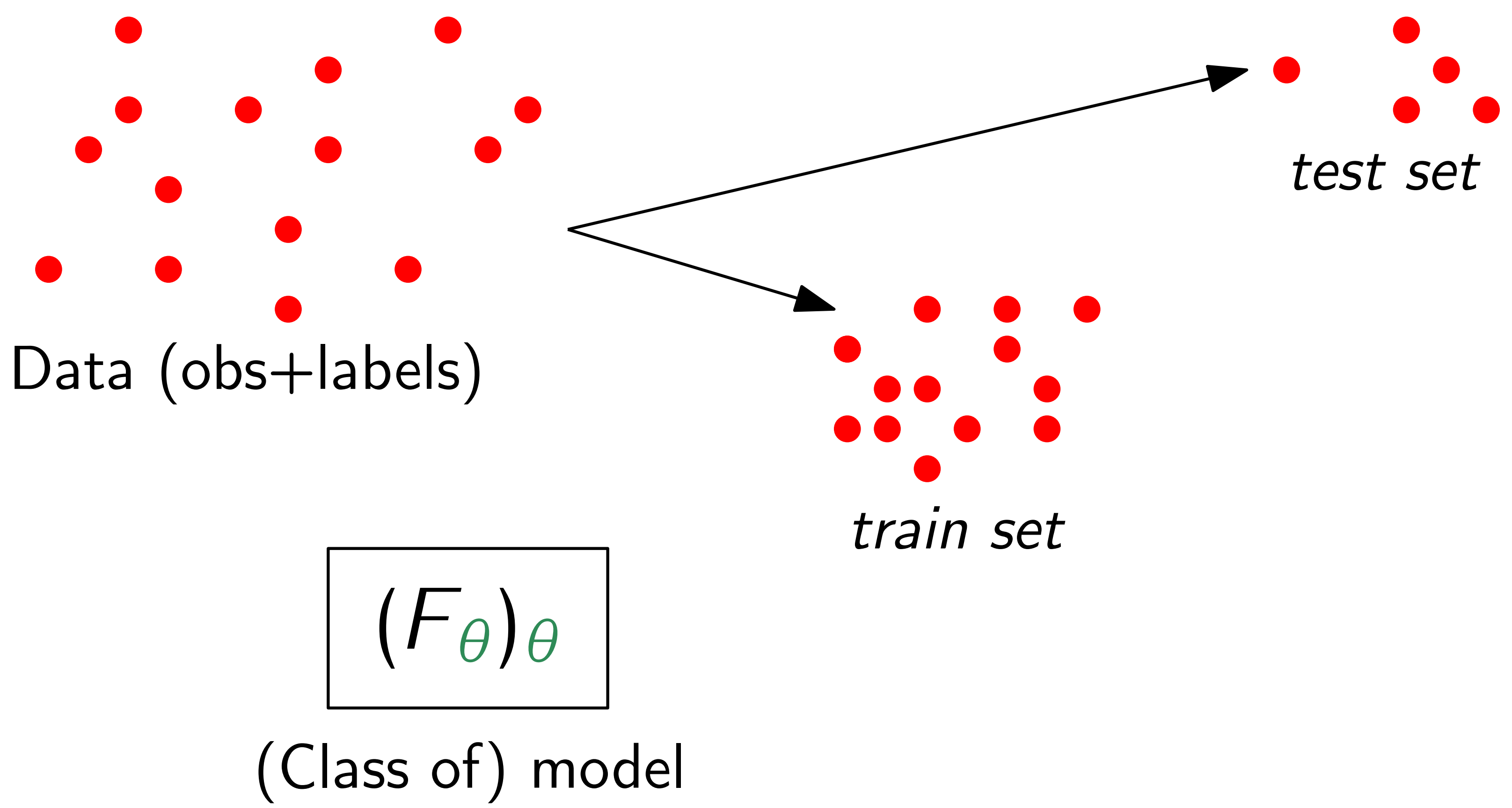
Step 0: Collect observations and labels $(x_i, y_i)_i$

↓

Step 1: Chose a class of models (e.g. linear regressions)

↓

Step 2: Split the *train set* and the *test set*.



3. Training, Testing and Overfitting...

In Short:

Once you have trained your model and provided that it achieves a reasonnably low training loss, you **must** test it by looking at its performances on observations that were not seen during the training phase (but distributed similarly to the training set).

Step 0: Collect observations and labels $(x_i, y_i)_i$

↓

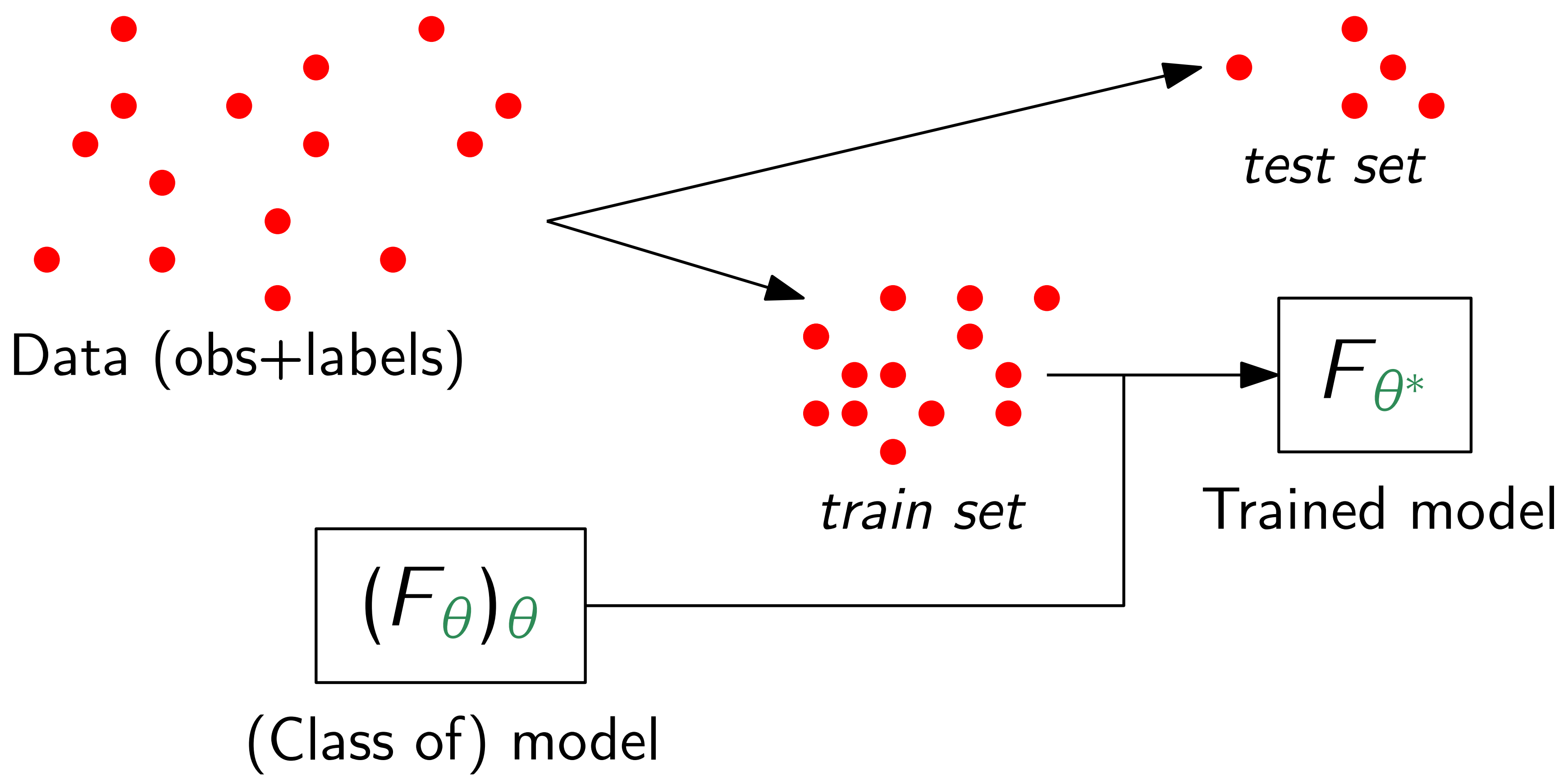
Step 1: Chose a class of models (e.g. linear regressions)

↓

Step 2: Split the *train set* and the *test set*.

↓

Step 3: Train (optimize the parameters) your model on the train set.



3. Training, Testing and Overfitting...

In Short:

Once you have trained your model and provided that it achieves a reasonnably low training loss, you **must** test it by looking at its performances on observations that were not seen during the training phase (but distributed similarly to the training set).

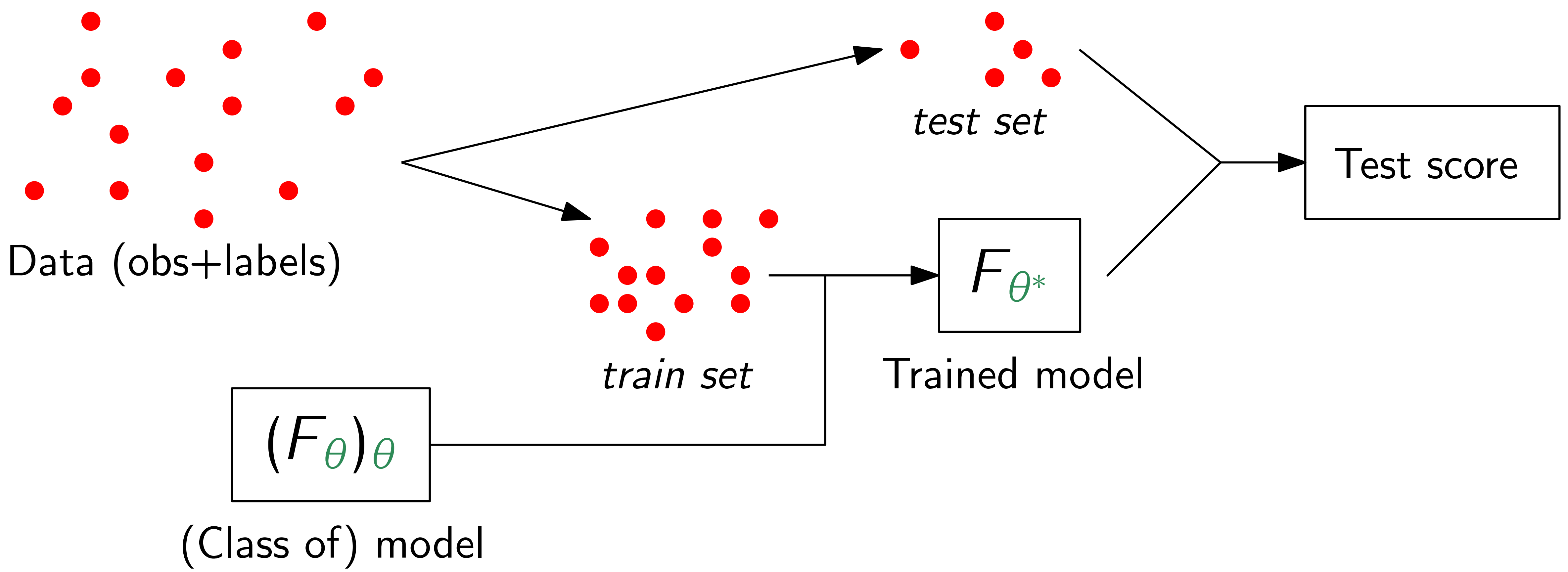
Step 0: Collect observations and labels $(x_i, y_i)_i$

Step 1: Chose a class of models (e.g. linear regressions)

Step 2: Split the *train set* and the *test set*.

Step 3: Train (optimize the parameters) your model on the train set.

Step 4: We evaluate our model on the test set



3. Training, Testing and Overfitting...

In Short:

Once you have trained your model and provided that it achieves a reasonnably low training loss, you **must** test it by looking at its performances on observations that were not seen during the training phase (but distributed similarly to the training set).

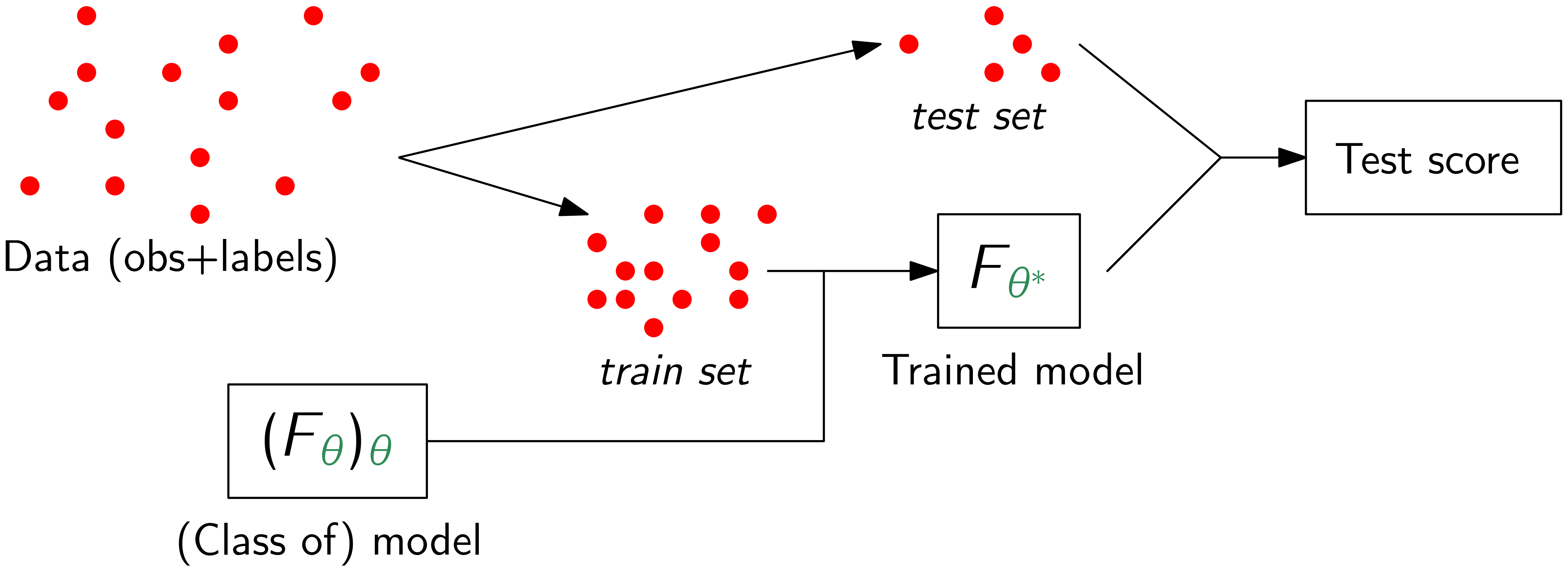
Step 0: Collect observations and labels $(x_i, y_i)_i$

Step 1: Chose a class of models (e.g. linear regressions)

Step 2: Split the *train set* and the *test set*.

Step 3: Train (optimize the parameters) your model on the train set.

Step 4: We evaluate our model on the test set



Step 5: And if we want to be sure that our performances are not biased by a "lucky" split, we repeat Steps 2—4 several times. Then, we report the average validation loss and its **standard deviation** on the different trials.

4. Mitigating overfitting: regularization.

4. Mitigating overfitting: regularization.

Intuitive idea: Allow for complex models (large space of parameters) but penalize the use of large parameters (which typically induce irregularity in your model).

4. Mitigating overfitting: regularization.

Intuitive idea: Allow for complex models (large space of parameters) but penalize the use of large parameters (which typically induce irregularity in your model).

Definition:

Let $X \in \mathbb{R}^{n \times d}$ be a set of n observations in dimension d, and $Y \in \mathbb{R}^{n \times k}$ be a corresponding set of labels.
The p-regularized (or penalized) Linear Regression (for $p \geqslant 1$) with parameter $M^* \in \mathbb{R}^{d \times k}$ is defined as $x \mapsto xM^*$ where M^* is the minimizer of

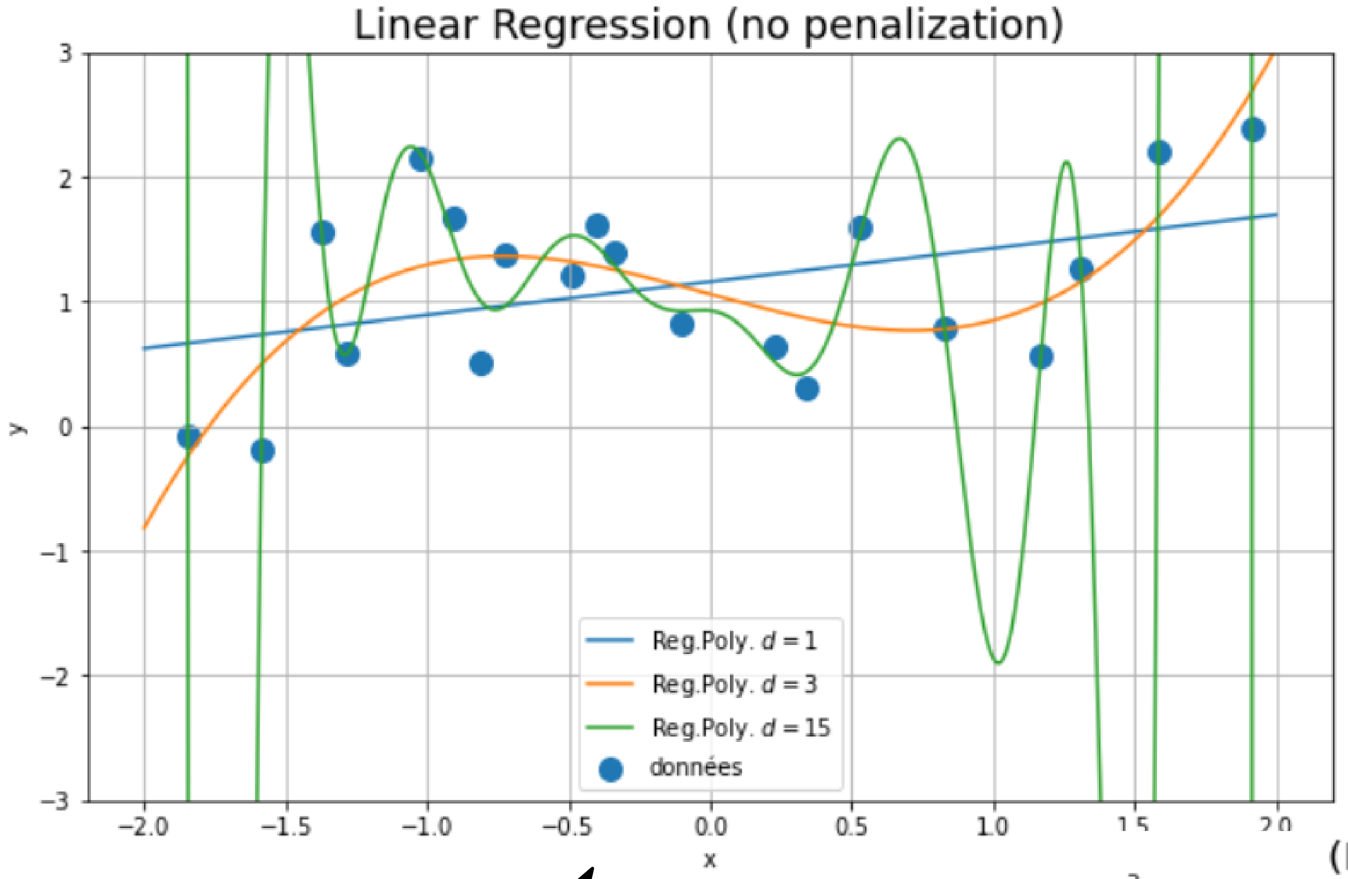
$$M \mapsto \|XM - Y\|_2^2 + \lambda \|M\|_p^p,$$

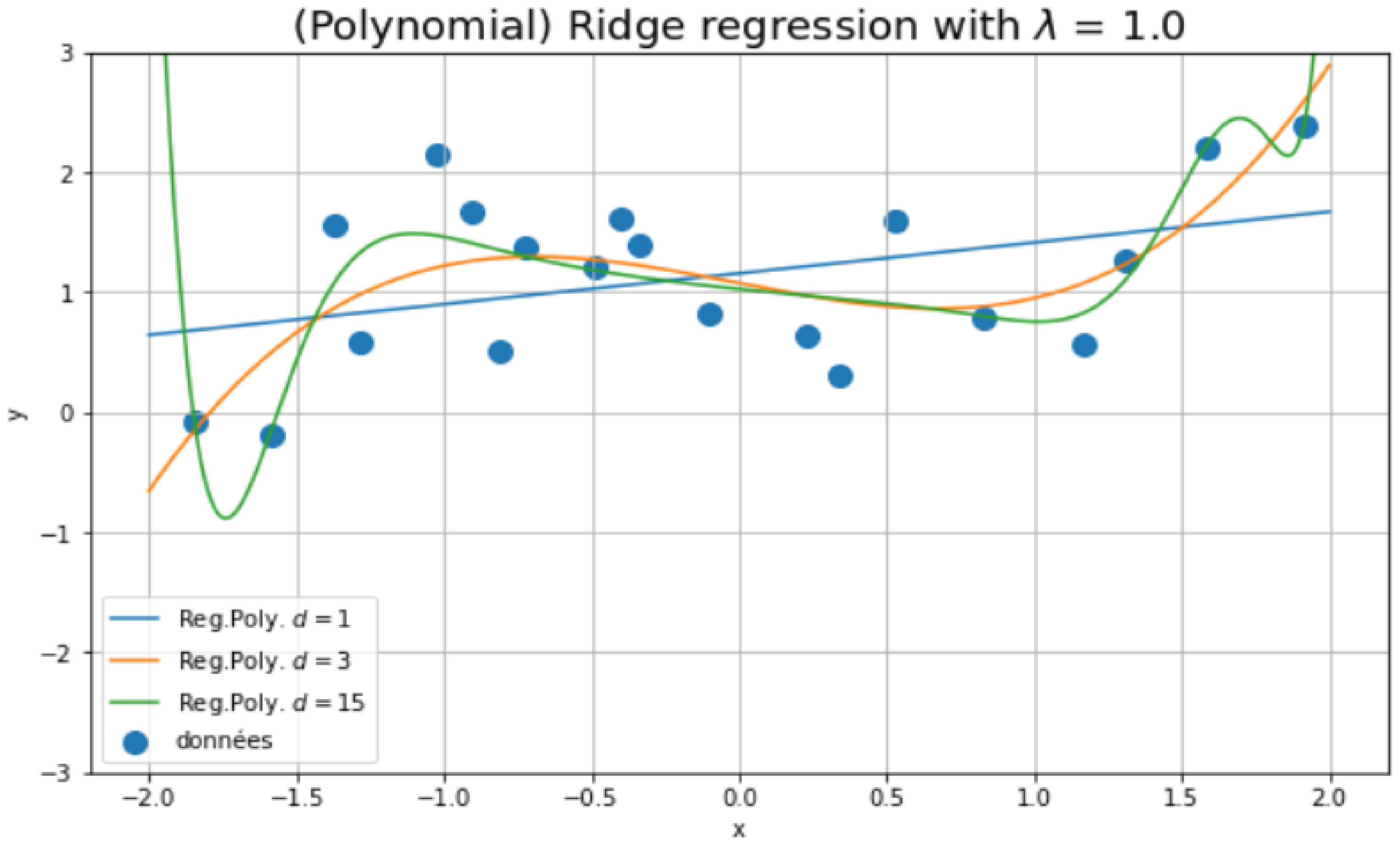
where $\lambda > 0$ is an hyper-parameter.
When $p = 1$, this model is referred to as the Lasso regression, when $p = 2$ it is referred to as the Ridge regression.

4. Mitigating overfitting: regularization.

Intuitive idea: Allow for complex models (large space of parameters) but penalize the use of large parameters (which typically induce irregularity in your model).

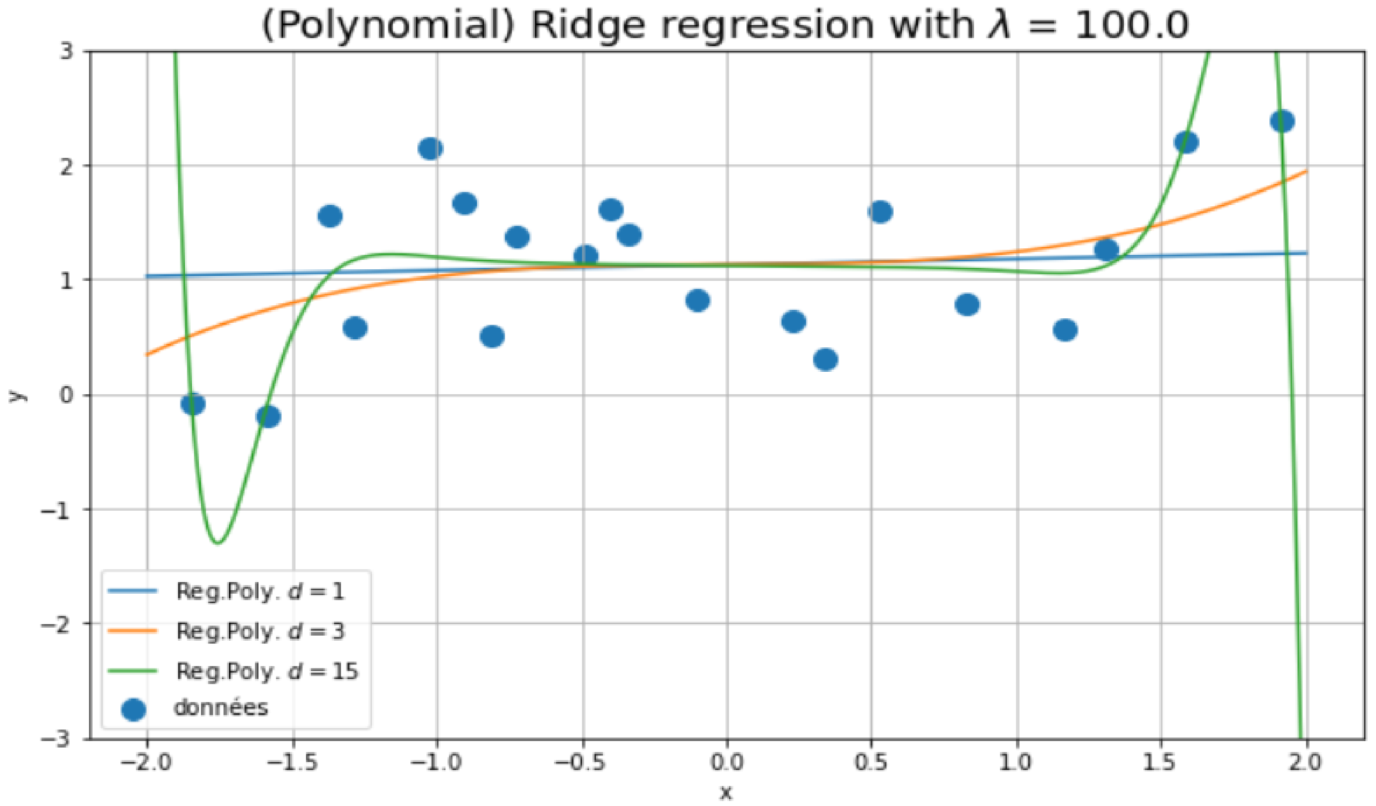
Example: In the context of polynomial regression ($k = 1$), $M = (\theta_0, \dots, \theta_d)$ which are the coefficients of the polynom we learn. Penalizing large norm for M means favoring small coefficients, that is small variations $\Rightarrow$ more regularity.





Don't use too complex models without regularization

Don't over-regularize



4. Mitigating overfitting: regularization.

Intuitive idea: Allow for complex models (large space of parameters) but penalize the use of large parameters (which typically induce irregularity in your model).

Definition:

Let $X \in \mathbb{R}^{n\times d}$ be a set of n observations in dimension d, and $Y \in \mathbb{R}^{n\times k}$ be a corresponding set of labels.
The p-regularized (or penalized) Linear Regression (for $p \geqslant 1$) with parameter $M^* \in \mathbb{R}^{d\times k}$ is defined as $x \mapsto xM^*$ where M^* is the minimizer of

$$M \mapsto \|XM - Y\|_2^2 + \lambda \|M\|_p^p,$$

where $\lambda > 0$ is an hyper-parameter.
When $p = 1$, this model is referred to as the Lasso regression, when $p = 2$ it is referred to as the Ridge regression.

Proposition:

Assuming that the matrix $X^TX + \lambda n \mathrm{Id}_d$ is non-singular, the optimal M^* for the **Ridge** regression is given by

$$M^* = (X^TX + \lambda n \mathrm{Id}_d)^{-1} X^T Y.$$

5. About other regression models.

Of course, linear models are just (very important) examples among the broad variety of machine learning models dedicated to regression tasks. For different models proposed by `scikit-learn`, we can mention:

- The k-nearest-neighbors model (see lab): to a new observation x we assign the value $F(x) = (y_{i_1} + y_{i_2} + \cdots + y_{i_k})/k$ where $x_{i_1}, \ldots, x_{i_k}$ are the k observations in the training set that are the closest to x. Observe that :
  - The parameter k is chosen once for all (hyper-parameter).
  - This model is (trainable) parameter-less, it does not need to be trained!
- Decision trees : they "cut" the space using a series of thresholds (that are learned during training).

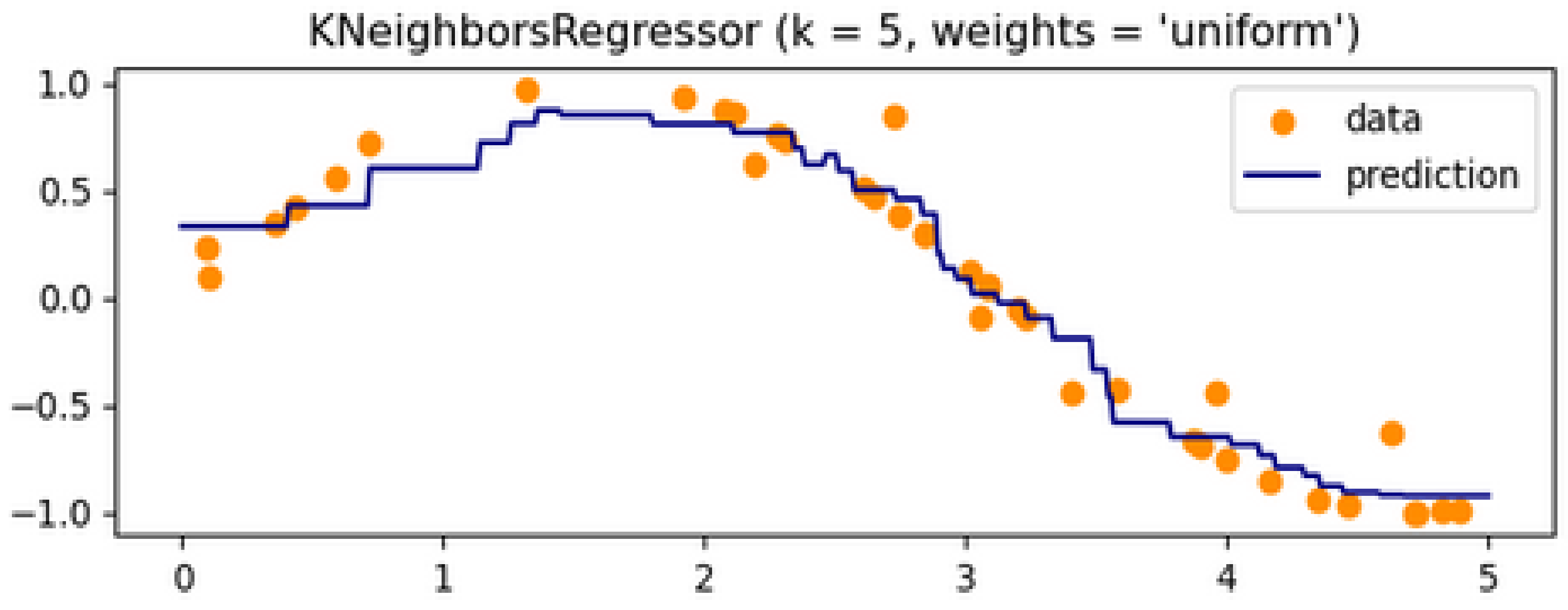


Illustration of a k-NN regression (sklearn)

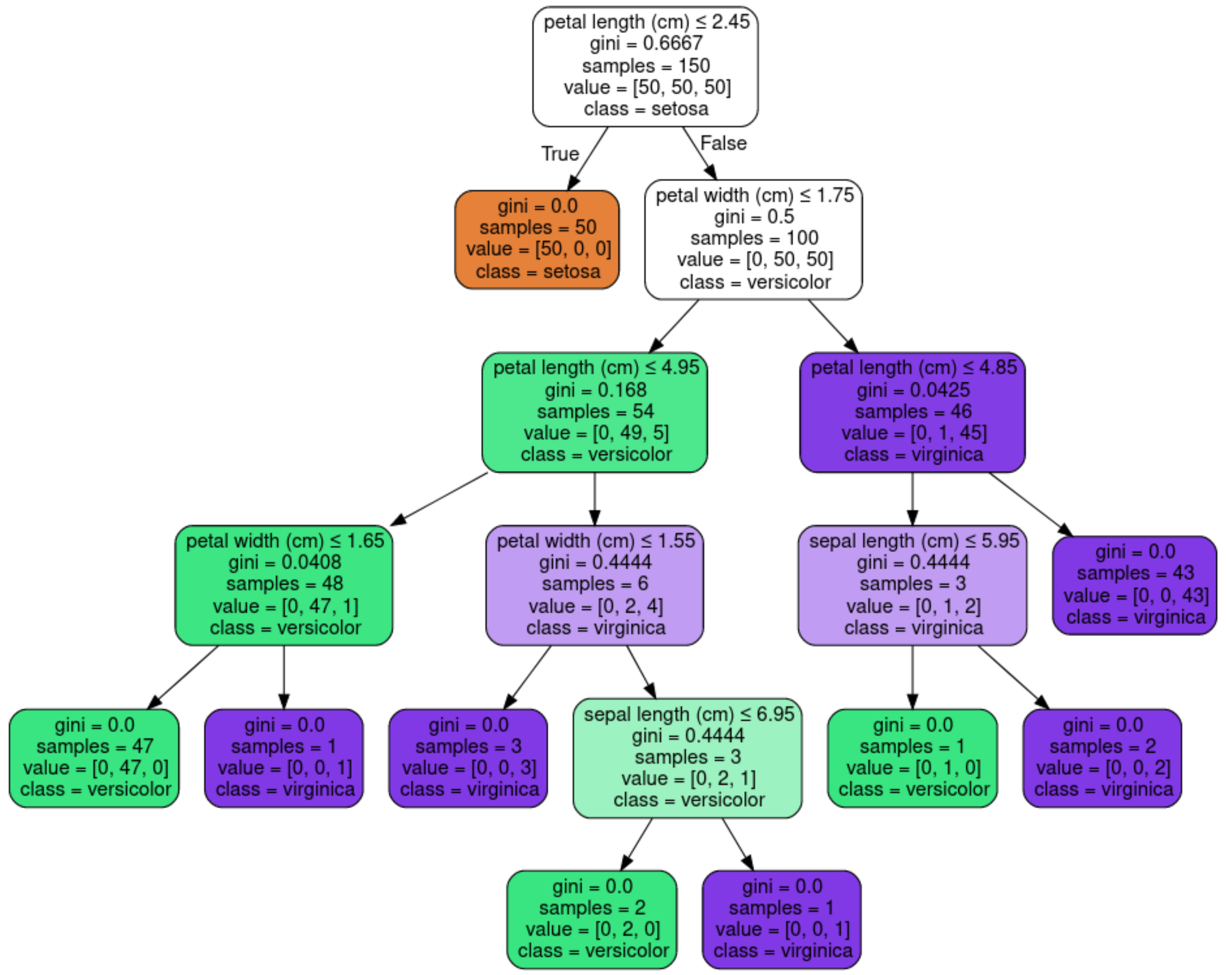


Illustration of a decision tree (sklearn)

6. About the optimization of the parameters...

As explained before, the parameters θ of a regression model must be **optimized** to be adapted to training data (the "learning" phase)—the goal being to **minimize** an objective function that assess if our model is able to relate the observations x_i to their corresponding labels y_i on the training set.
When our model F is a linear (or polynomial) model trained to minimize the MSE, we have access to an **explicit** formula for the optimal parameter θ based on the training data. But this is not always the case.

Question : Given training observations $(x_i)_i$ and labels $(y_i)_i$, a parametric model $F_\theta : x \mapsto F_\theta(x)$ and a loss function ℓ, how do we minimize the objective function

$$L : \theta \mapsto \sum_{i=1}^{n} \ell(F_\theta(x_i), y_i) \quad ?$$

Summary

In Short:

1. A regression model is a model that aims at predicting a variable y that is continuous (typically, a real number) given an observation x.
2. The simplest regression model is the linear regression: $F_\theta(x) = A \cdot x + b$, where $\theta = (A, b)$ represent the parameters of the model. We say that this is a parametric model.
3. We try to optimize θ to minimize the Mean Squared Error (MSE) on the training data.
4. A strength of linear regression: we have access to a closed form for the optimal parameter θ^* (the one that minimizes the MSE on the training data).
5. We can consider polynomial regressions, which are more expressive, and which actually boil down to linear regression on augmented observations.
6. Warning! A more expressive model will always be better **on the training set**. What really matters are its performances **on the validation set**.

Chapter 3: An optimization detour

Training a machine learning model F_θ boils down to optimizing its parameters in order to **minimize** an objective function $\theta \mapsto L(\theta)$. In this chapter, we will discuss the main algorithms (and its variations) used in ML: the gradient descent.

Chapter 3: An optimization detour

1. The gradient descent algorithm

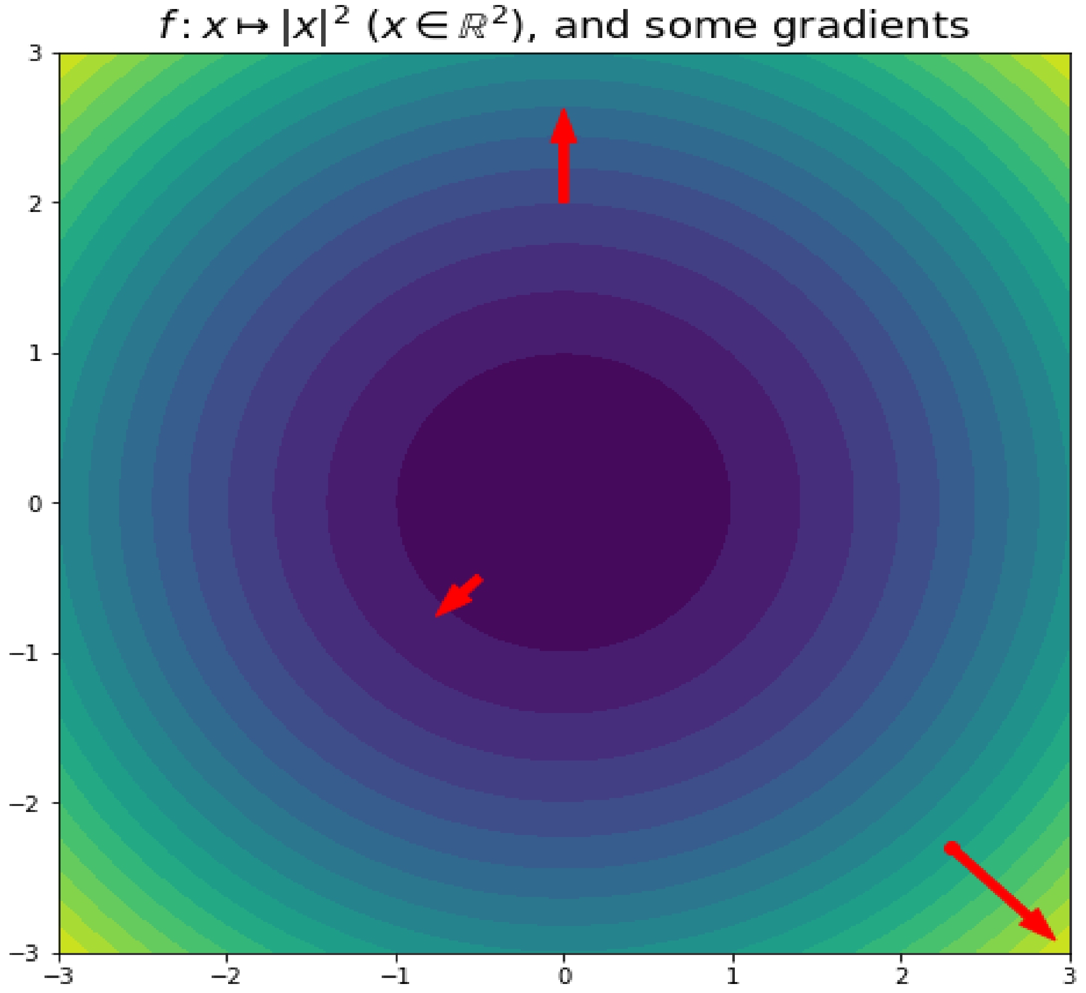
Definition:

Let $L : \mathbb{R}^d \to \mathbb{R}$. We say that L admits a gradient at $\theta \in \mathbb{R}^d$ if there exists a vector $\nabla L(\theta) \in \mathbb{R}^d$ (the gradient) such that

$$L(\theta + \mathrm{d}\theta) = L(\theta) + \langle \nabla L(\theta), \mathrm{d}\theta \rangle + o(\mathrm{d}\theta),$$

for all variation $\mathrm{d}\theta \in \mathbb{R}^d$. If $\nabla L(\theta) = 0$, we say that θ is a critical point of L.

$\to$ It describes the first-order variation of L: when we move from θ in any direction $\mathrm{d}\theta$, locally, the variation of L is $\langle \nabla L(\theta), \mathrm{d}\theta \rangle$. In particular, if we go in the direction $\mathrm{d}\theta = \nabla L(\theta)$, we maximize (locally) the variation of L. We say that the gradient is the steepest ascent direction. Conversely, $-\nabla L(\theta)$ is the steepest descent direction.



1. The gradient descent algorithm

Definition:

Let $L : \mathbb{R}^d \to \mathbb{R}$. We say that L admits a gradient at $\theta \in \mathbb{R}^d$ if there exists a vector $\nabla L(\theta) \in \mathbb{R}^d$ (the gradient) such that

$$L(\theta + \mathrm{d}\theta) = L(\theta) + \langle \nabla L(\theta), \mathrm{d}\theta \rangle + o(\mathrm{d}\theta),$$

for all variation $\mathrm{d}\theta \in \mathbb{R}^d$. If $\nabla L(\theta) = 0$, we say that θ is a critical point of L.

→ It describes the first-order variation of L: when we move from θ in any direction $\mathrm{d}\theta$, locally, the variation of L is $\langle \nabla L(\theta), \mathrm{d}\theta \rangle$. In particular, if we go in the direction $\mathrm{d}\theta = \nabla L(\theta)$, we maximize (locally) the variation of L. We say that the gradient is the steepest ascent direction. Conversely, $-\nabla L(\theta)$ is the steepest descent direction.

Proposition:

Assume that $L : \mathbb{R}^d \to \mathbb{R}$ is smooth (i.e. admits gradient everywhere), and that $\nabla L(\theta) \neq 0$ at some $\theta \in \mathbb{R}^d$.
Then, for λ small enough, $L(\theta - \lambda \nabla L(\theta)) < L(\theta)$.

1. The gradient descent algorithm

Definition:

Let $L : \mathbb{R}^d \to \mathbb{R}$. We say that L admits a gradient at $\theta \in \mathbb{R}^d$ if there exists a vector $\nabla L(\theta) \in \mathbb{R}^d$ (the gradient) such that

$$L(\theta + \mathrm{d}\theta) = L(\theta) + \langle \nabla L(\theta), \mathrm{d}\theta \rangle + o(\mathrm{d}\theta),$$

for all variation $\mathrm{d}\theta \in \mathbb{R}^d$. If $\nabla L(\theta) = 0$, we say that θ is a critical point of L.

→ It describes the first-order variation of L: when we move from θ in any direction $\mathrm{d}\theta$, locally, the variation of L is $\langle \nabla L(\theta), \mathrm{d}\theta \rangle$. In particular, if we go in the direction $\mathrm{d}\theta = \nabla L(\theta)$, we maximize (locally) the variation of L. We say that the gradient is the steepest ascent direction. Conversely, $-\nabla L(\theta)$ is the steepest descent direction.

Proposition:

If θ is a (local) minimum of L, that is there exists an open neighborhood U of θ such that $L(\theta) \leqslant L(\theta')$ for any $\theta' \in U$, then $\nabla L(\theta) = 0$.

- This also holds for (local) maxima. Points θ which are neither local maximum nor minimum are called saddle points.
- To characterize minimizers, we can use the criterion $\nabla^2 L(\theta) \succeq 0$ (SDP, i.e. eigenvalues $\geqslant 0$).

1. The gradient descent algorithm

Definition:

Given L smooth, $\lambda > 0$, an initial θ_0, define the sequence

$$\theta_{t+1} = \theta_t - \lambda \nabla L(\theta_t).$$

This sequence is called a gradient descent (GD) for L with initialization θ_0 with learning rate (or step-size) λ.

Algorithm:

The simplest GD algorithm applied to $L : \mathbb{R}^d \to \mathbb{R}$ (smooth) consists thus of:

- Choose $\lambda > 0$, $\theta_0 \in \mathbb{R}^d$,
- Fix a number of iterations T,
- Build the sequence $\theta_1, \dots, \theta_T$.
- Return θ_T.

Intuition: Hopefully, (i) we produce a **converging** sequence $(\theta_t)_t$ (as $T \to \infty$), (ii) the sequence $(L(\theta_t))_t$ is decreasing, (iii) it converges toward a minimum of L.

1. The gradient descent algorithm

Definition:

Given L smooth, $\lambda > 0$, an initial θ_0, define the sequence

$$\theta_{t+1} = \theta_t - \lambda \nabla L(\theta_t).$$

This sequence is called a gradient descent (GD) for L with initialization θ_0 with learning rate (or step-size) λ.

Algorithm:

The simplest GD algorithm applied to $L : \mathbb{R}^d \to \mathbb{R}$ (smooth) consists thus of:

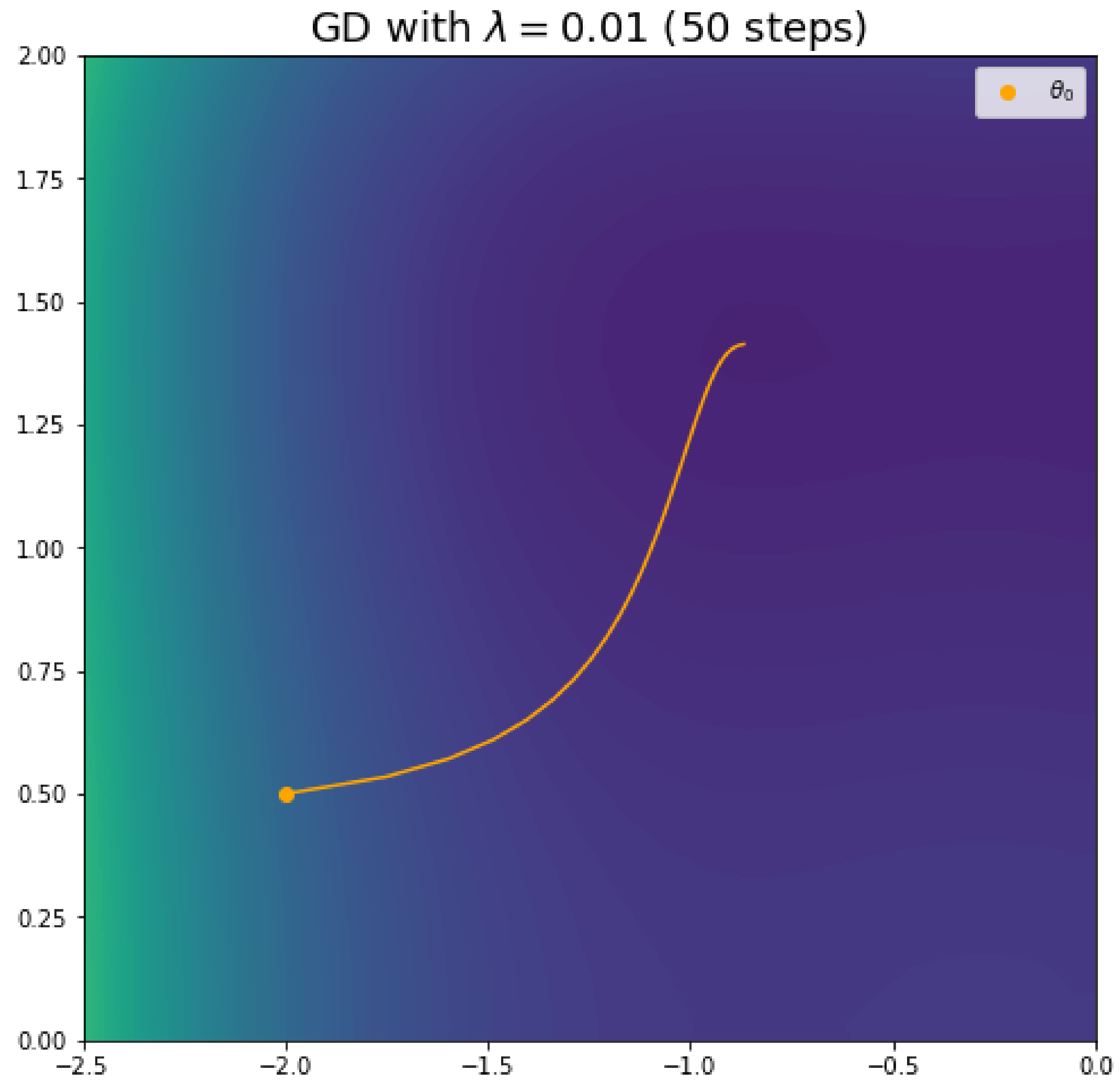
- Choose $\lambda > 0$, $\theta_0 \in \mathbb{R}^d$,
- Fix a number of iterations T,
- Build the sequence $\theta_1, \ldots, \theta_T$.
- Return θ_T.

Intuition: Hopefully, (i) we produce a **converging** sequence $(\theta_t)_t$ (as $T \to \infty$), (ii) the sequence $(L(\theta_t))_t$ is decreasing, (iii) it converges toward a minimum of L.

Note: Many variations, including step-dependent parameters λ_t (typically $\lambda_t \to 0$), stopping criterion, etc.

1. The gradient descent algorithm

We pick in the following $L : \mathbb{R}^2 \to \mathbb{R}, (x, y) \mapsto x^4 + y^4 - 2x^2 - 4y^2 + x$ (but it does not matter much).



1. The gradient descent algorithm

Interpretation: The iteration $\theta_{t+1} = \theta_t - \lambda \nabla L(\theta_t)$ can be understood as an explicit Euler discretization of the ordinary differential equation

$$\frac{\mathrm{d}\theta}{\mathrm{d}t} = -\nabla L(\theta(t))$$

with step size λ. The solution $t \mapsto \theta(t)$ of this ODE is called a gradient flow. In some sense, it can be proved under mild assumptions that the sequence $(\theta_t)_t$ converges toward the curve $t \mapsto \theta(t)$ when $\lambda \to 0$ and $T \to \infty$ (note that we overloaded notation here).

1. The gradient descent algorithm

Interpretation: The iteration $\theta_{t+1} = \theta_t - \lambda \nabla L(\theta_t)$ can be understood as an explicit Euler discretization of the ordinary differential equation

$$\frac{\mathrm{d}\theta}{\mathrm{d}t} = -\nabla L(\theta(t))$$

with step size λ. The solution $t \mapsto \theta(t)$ of this ODE is called a gradient flow. In some sense, it can be proved under mild assumptions that the sequence $(\theta_t)_t$ converges toward the curve $t \mapsto \theta(t)$ when $\lambda \to 0$ and $T \to \infty$ (note that we overloaded notation here).

- Some limitations of the Gradient Descent:

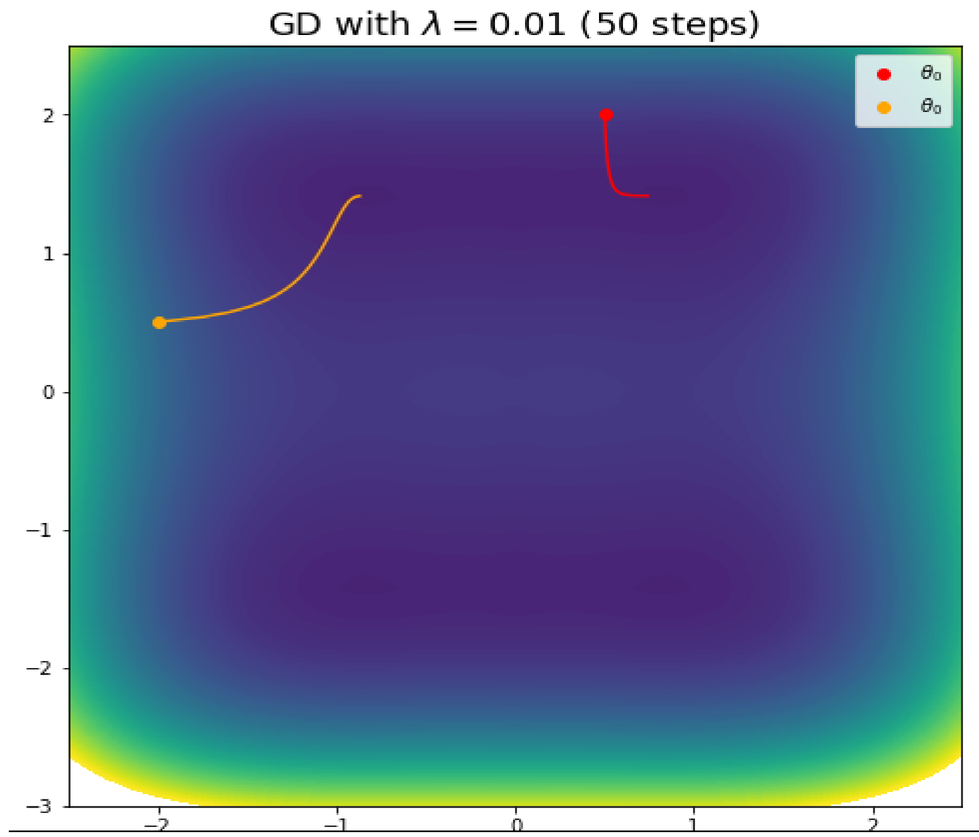
1. The gradient descent algorithm

Interpretation: The iteration $\theta_{t+1} = \theta_t - \lambda \nabla L(\theta_t)$ can be understood as an explicit Euler discretization of the ordinary differential equation

$$\frac{\mathrm{d}\theta}{\mathrm{d}t} = -\nabla L(\theta(t))$$

with step size λ. The solution $t \mapsto \theta(t)$ of this ODE is called a gradient flow. In some sense, it can be proved under mild assumptions that the sequence $(\theta_t)_t$ converges toward the curve $t \mapsto \theta(t)$ when $\lambda \to 0$ and $T \to \infty$ (note that we overloaded notation here).

- Some limitations of the Gradient Descent:



$\to$ Dependence on the initialization (not too bad, but keep it in mind if you pick random θ_0)

Chapter 3: An optimization detour

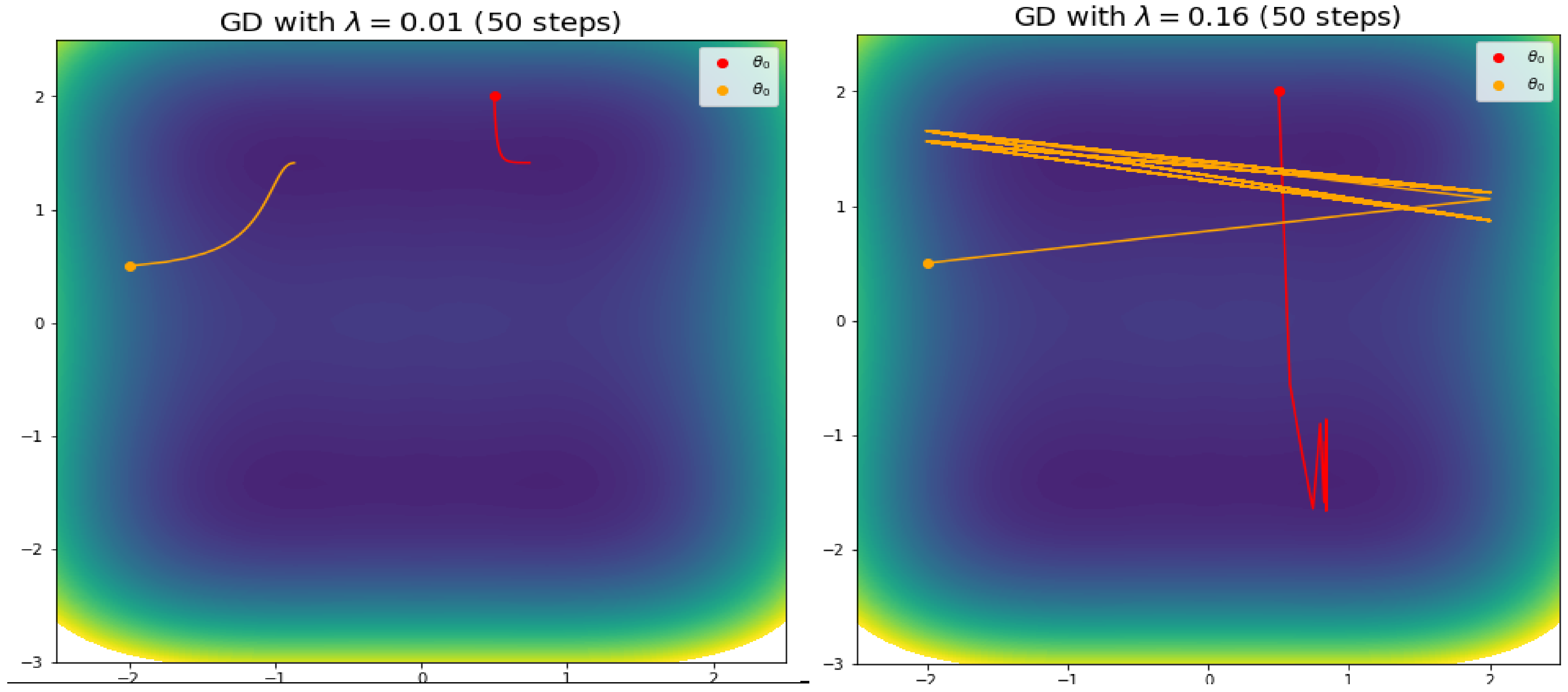
1. The gradient descent algorithm

Interpretation: The iteration $\theta_{t+1} = \theta_t - \lambda \nabla L(\theta_t)$ can be understood as an explicit Euler discretization of the ordinary differential equation

$$\frac{\mathrm{d}\theta}{\mathrm{d}t} = -\nabla L(\theta(t))$$

with step size λ. The solution $t \mapsto \theta(t)$ of this ODE is called a gradient flow. In some sense, it can be proved under mild assumptions that the sequence $(\theta_t)_t$ converges toward the curve $t \mapsto \theta(t)$ when $\lambda \to 0$ and $T \to \infty$ (note that we overloaded notation here).

- Some limitations of the Gradient Descent:



→ Dependence on the initialization (not too bad, but keep it in mind if you pick random θ_0)

→ if λ is too large, may not converge.

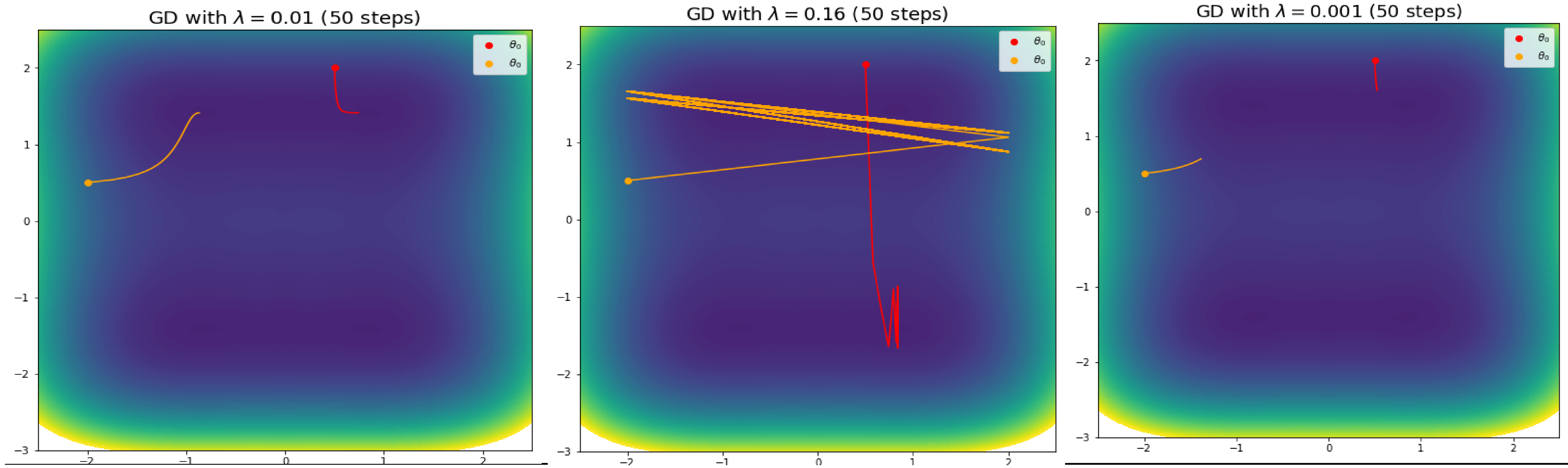
1. The gradient descent algorithm

Interpretation: The iteration $\theta_{t+1} = \theta_t - \lambda \nabla L(\theta_t)$ can be understood as an explicit Euler discretization of the ordinary differential equation

$$\frac{\mathrm{d}\theta}{\mathrm{d}t} = -\nabla L(\theta(t))$$

with step size λ. The solution $t \mapsto \theta(t)$ of this ODE is called a gradient flow. In some sense, it can be proved under mild assumptions that the sequence $(\theta_t)_t$ converges toward the curve $t \mapsto \theta(t)$ when $\lambda \to 0$ and $T \to \infty$ (note that we overloaded notation here).

- Some limitations of the Gradient Descent:



→ Dependence on the initialization (not too bad, but keep it in mind if you pick random θ_0)

→ if λ is too large, may not converge.

→ if λ is too small, takes a long time to converge.

2. Convex functions.

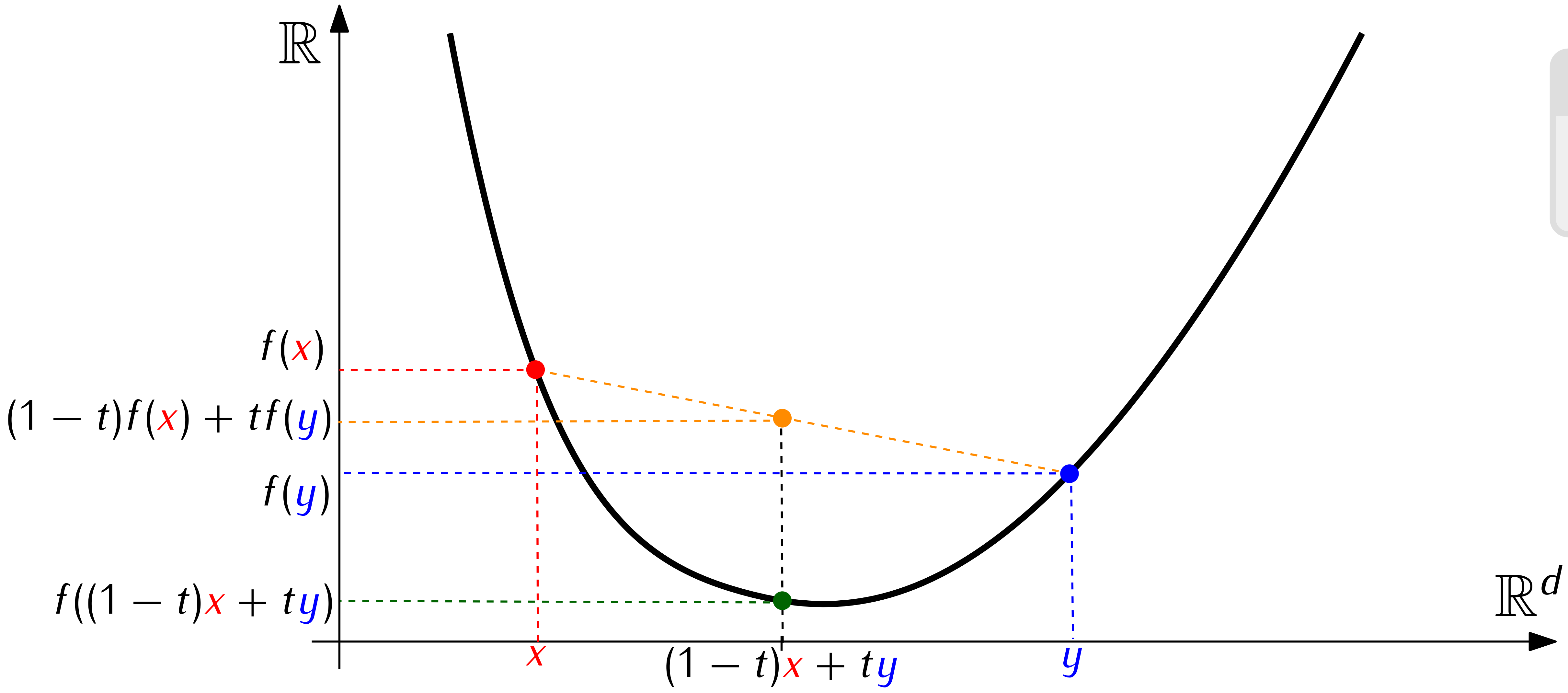
2. Convex functions.

Definition:

A function $f : \mathbb{R}^d \to \mathbb{R} \cup \{+\infty\}$ is said to be convex if for any $x, y \in \mathbb{R}^d$ and any $t \in [0, 1]$,

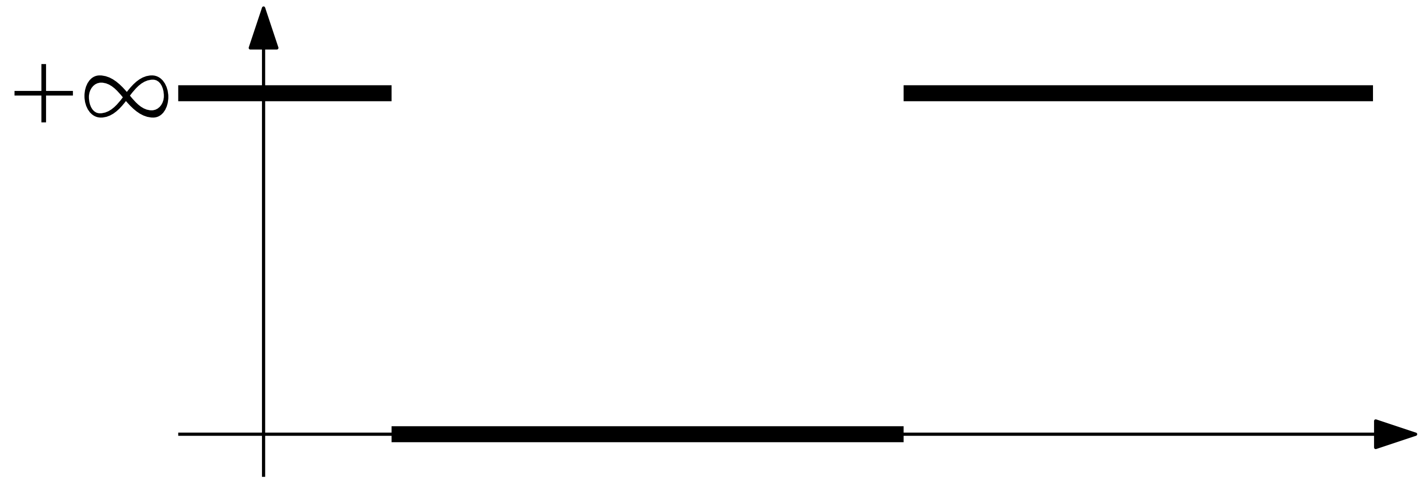
$$f((1-t)x + ty) \leqslant (1-t)f(x) + tf(y). \tag{6}$$

We define the domain of f as $D(f) = \{x,\ f(x) < +\infty\}$, which is assumed to be a convex subset of $\mathbb{R}^d$.



In Short:

The **curve** is always below chord joining any x, y.



(Surprisingly?) this function is convex

Chapter 3: An optimization detour

2. Convex functions.

Proposition:

- If f is convex, then it is continuous (on the interior of its domain). It may not be differentiable, but it is differentiable Lebesgue-a.e.

If we assume that it is differentiable, its gradient has to be monotone, that is:

- for all $x, y \in D(f)$,

$$\langle \nabla f(x) - \nabla f(y), x - y \rangle \geqslant 0. \tag{7}$$

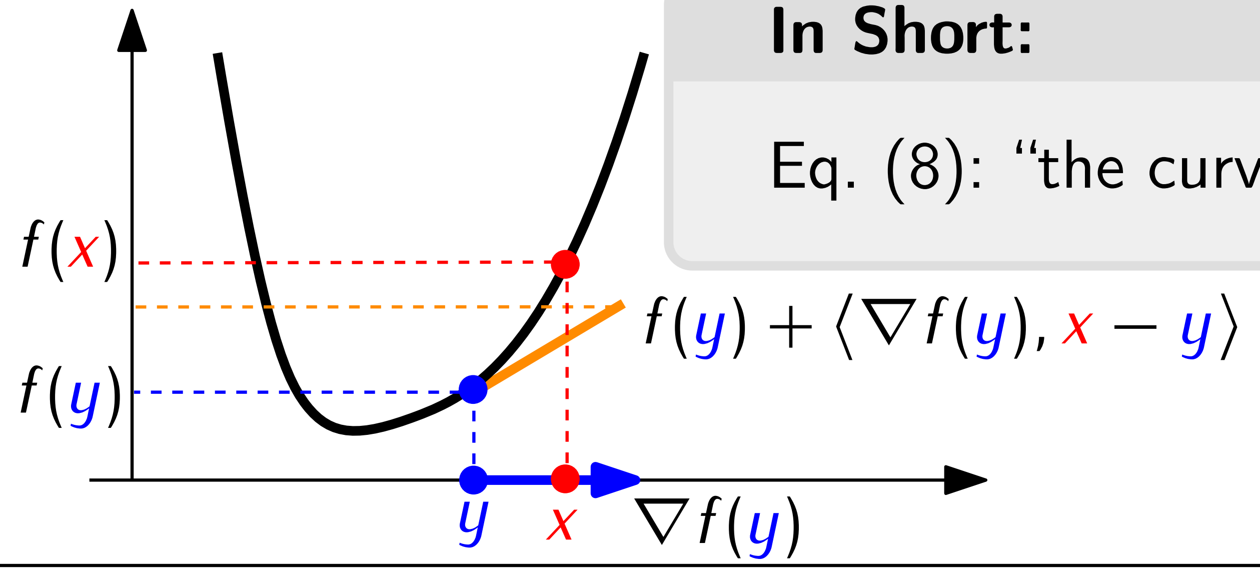
Furthermore,

$$f(x) \geqslant f(y) + \langle \nabla f(y), x - y \rangle . \tag{8}$$

- Therefore, if $\nabla f(x) = 0$, then x is a (global) minimizer of f (in particular, no local minimizer).

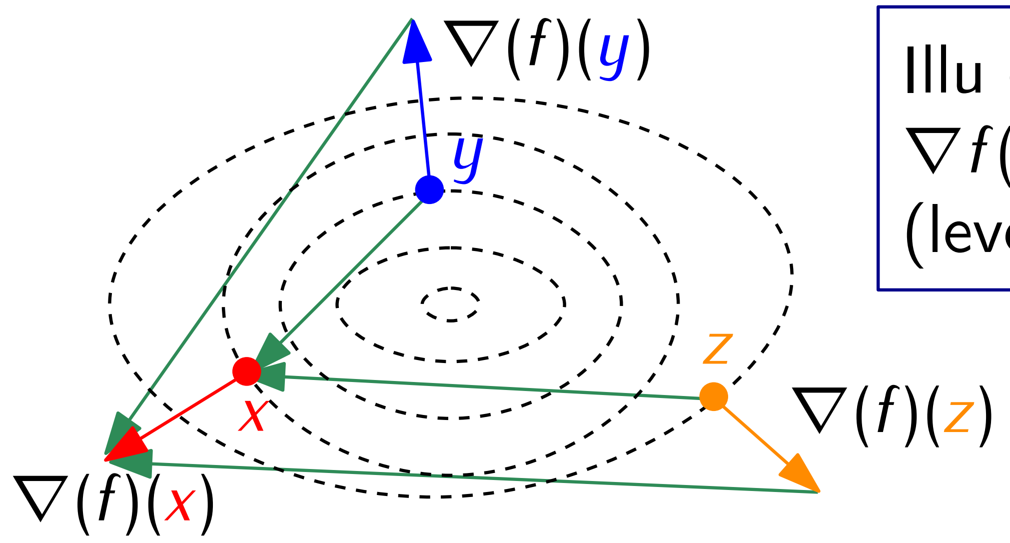
If f has a second derivative, we have

- the Hessian matrix $\nabla^2 f(x) = \left(\frac{\partial^2 f}{\partial^2 x_i x_j}(x) \right)$ shall be positive semi-definite for every $x \in D(f)$, that is $\forall x \in D(f)$, $\forall u \in \mathbb{R}^d$, $u^T \nabla^2 f(x) u \geqslant 0$, denoted by $\nabla^2 f(x) \succeq 0$ which is equivalent to say that the eigenvalues of $\nabla^2 f(x)$ are non-negative.



In Short:

Eq. (8): "the curve is above any tangent plane".



Illu of Eq. (7). Note: $\nabla f(x) \perp \{f(x') = f(x)\}$ (levelset).

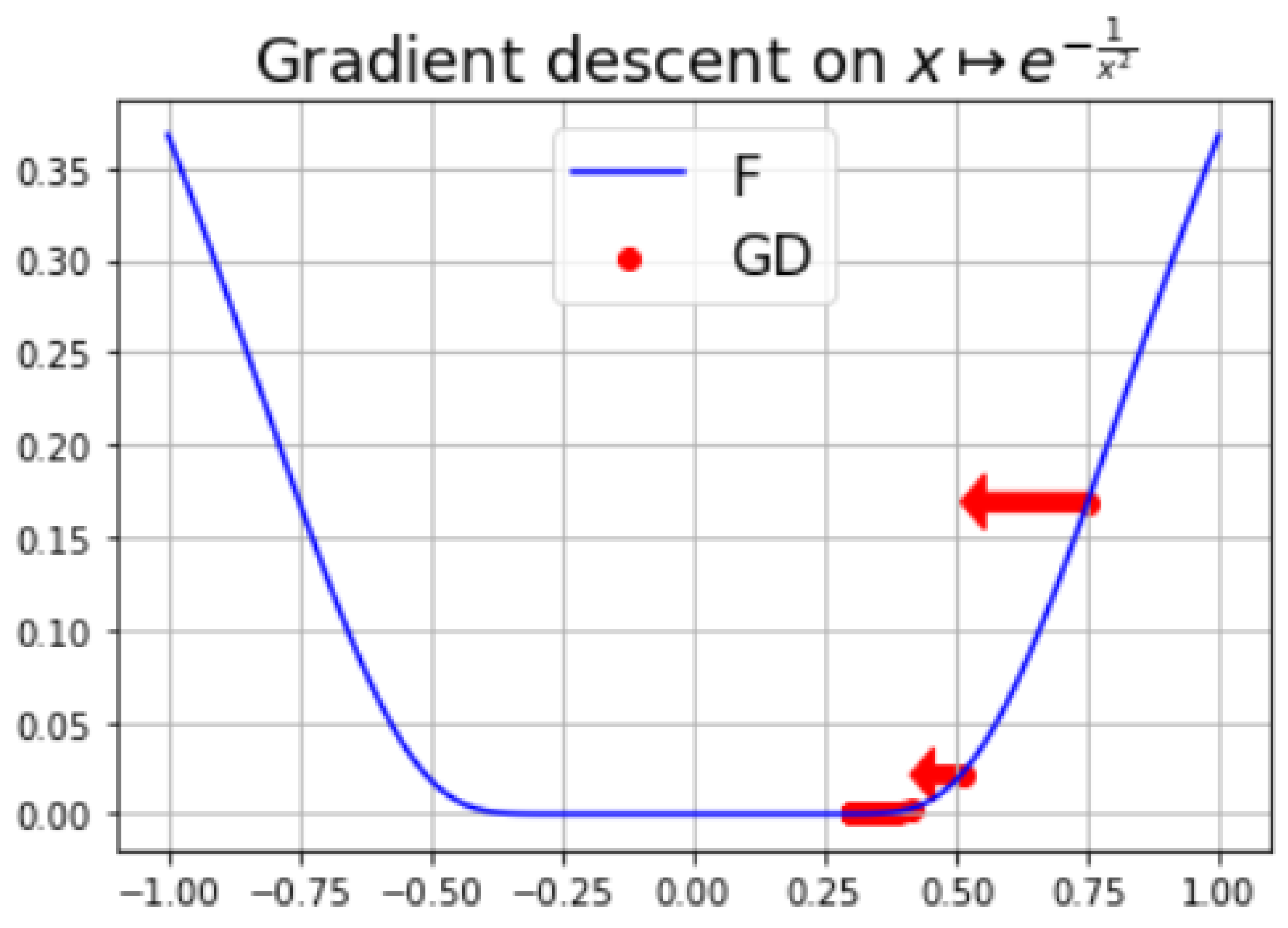
2. Convex functions.

Intuition: The gradient descent algorithm should work (very) well on convex functions... under suitable assumptions!

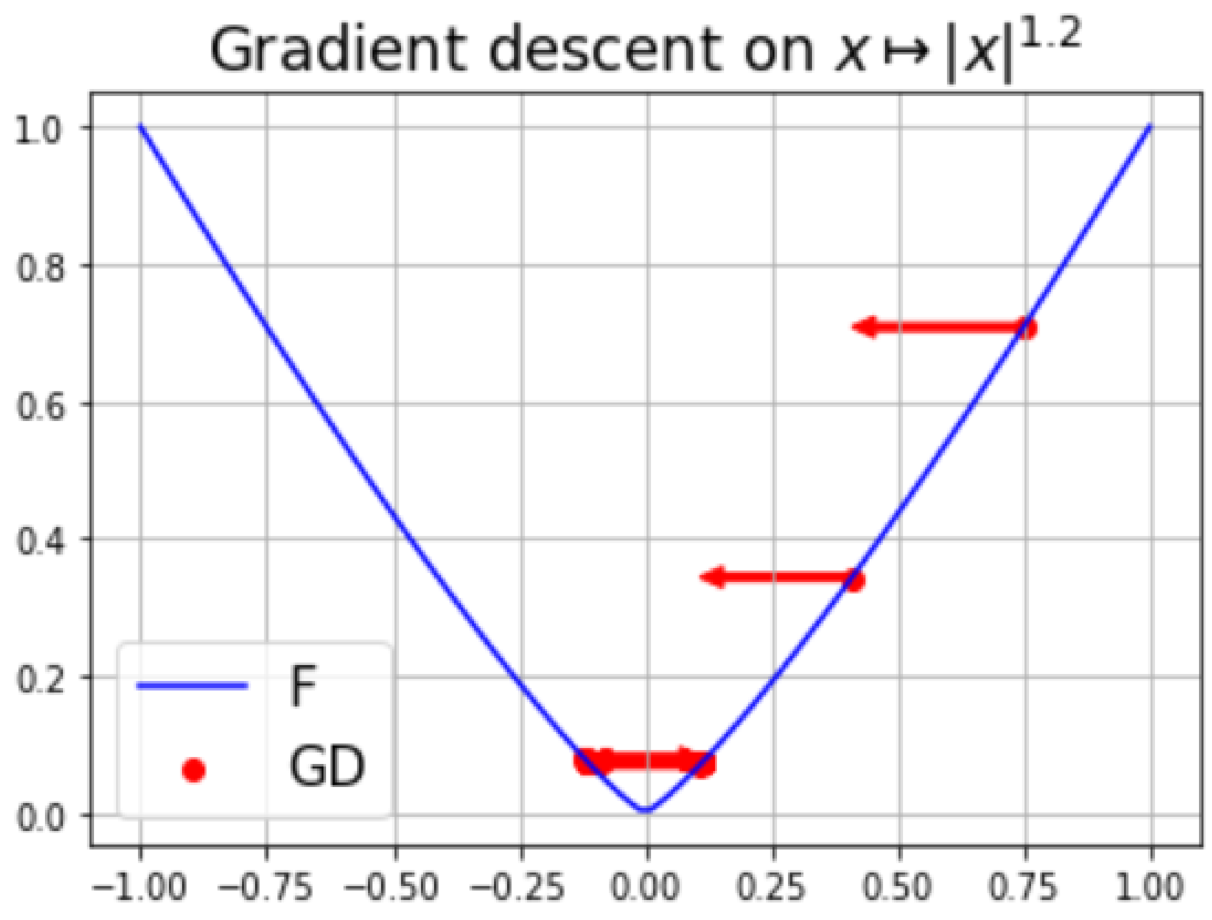
2. Convex functions.

Intuition: The gradient descent algorithm should work (very) well on convex functions... under suitable assumptions!

- f should be "sufficiently curved"...
- ... but not too much.



Not sufficiently curved ⇒ super slow convergence toward the global minimizer of f.



Albeit being convex and $\mathcal{C}^1$, the function is very curvy around its minimizer $x^* = 0$ ⇒ the GD bounces ⇒ super slow convergence (if any).

2. Convex functions.

Definition:

If f is convex, and $\alpha > 0$, we say that f is α-strongly convex if for all $x, y \in D(f)$, $t \in [0,1]$,

$$f(tx + (1-t)y) \leqslant tf(x) + (1-t)f(y) - \frac{\alpha}{2} t(1-t)\|x-y\|^2.$$

Proposition:

If f is convex and twice differentiable, it is α-strongly convex iff one of the following hold:

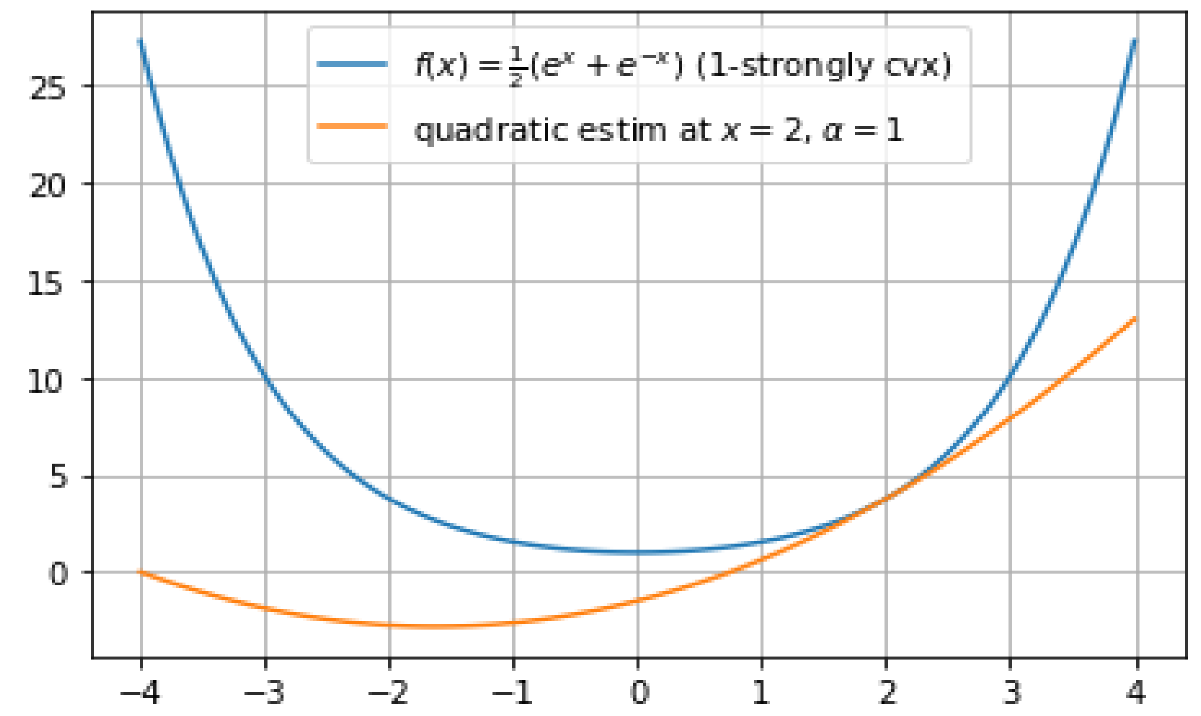
- for all $x, y \in D(f)$,

$$f(y) \geqslant f(x) + \langle \nabla f(x), y - x \rangle + \frac{\alpha}{2}\|x-y\|^2,$$

- The function $x \mapsto f(x) - \frac{\alpha}{2}\|x\|^2$ is convex,
- We have $\nabla^2 f(x) \succeq \alpha I$ for every $x \in D(f)$.

In Short:

Everywhere, f is above its tangent + a parabola of curvature α (second derivative) $\Rightarrow$ its curvature is larger than α everywhere.



2. Convex functions.

Definition:

If f is convex, and $\alpha > 0$, we say that f is α-strongly convex if for all $x, y \in D(f)$, $t \in [0,1]$,

$$f(tx + (1-t)y) \leqslant tf(x) + (1-t)f(y) - \frac{\alpha}{2}t(1-t)\|x-y\|^2.$$

Proposition:

If f is convex and twice differentiable, it is α-strongly convex iff one of the following hold:

- for all $x, y \in D(f)$,

$$f(y) \geqslant f(x) + \langle \nabla f(x), y - x\rangle + \frac{\alpha}{2}\|x-y\|^2,$$

- The function $x \mapsto f(x) - \frac{\alpha}{2}\|x\|^2$ is convex,
- We have $\nabla^2 f(x) \succeq \alpha I$ for every $x \in D(f)$.

In Short:

Everywhere, f is above its tangent + a parabola of curvature α (second derivative) $\Rightarrow$ its curvature is larger than α everywhere.

Proposition:

If f is α-scvx, it has a unique minimizer x^*.

Exercise: Prove this.

2. Convex functions.

Definition:

If f is convex, and $\alpha > 0$, we say that f is α-strongly convex if for all $x, y \in D(f)$, $t \in [0, 1]$,

$$f(tx + (1-t)y) \leqslant tf(x) + (1-t)f(y) - \frac{\alpha}{2} t(1-t)||x-y||^2.$$

Proposition:

If f is α-strongly convex, it satisfies the so-called Polyak-Lojasiewicz condition (PL), that is for all $x \in D(f)$,

$$0 \leqslant f(x) - f(x^*) \leqslant \frac{1}{2\alpha} ||\nabla f(x)||^2.$$

In Short:

Gradients get large when far from the global minimizer, and conversely, $||\nabla f(x_t)|| \to 0 \Rightarrow f(x_t) \to f(x^*)$.

2. Convex functions.

Definition:

We say that f is β-smooth if it is differentiable and is gradient is β-Lipschitz:

$$\forall x, y \in D(f), \|\nabla f(x) - \nabla f(y)\| \leqslant \beta \|x - y\|.$$

2. Convex functions.

Definition:

We say that f is β-smooth if it is differentiable and is gradient is β-Lipschitz:

$$\forall x, y \in D(f), \|\nabla f(x) - \nabla f(y)\| \leqslant \beta\|x - y\|.$$

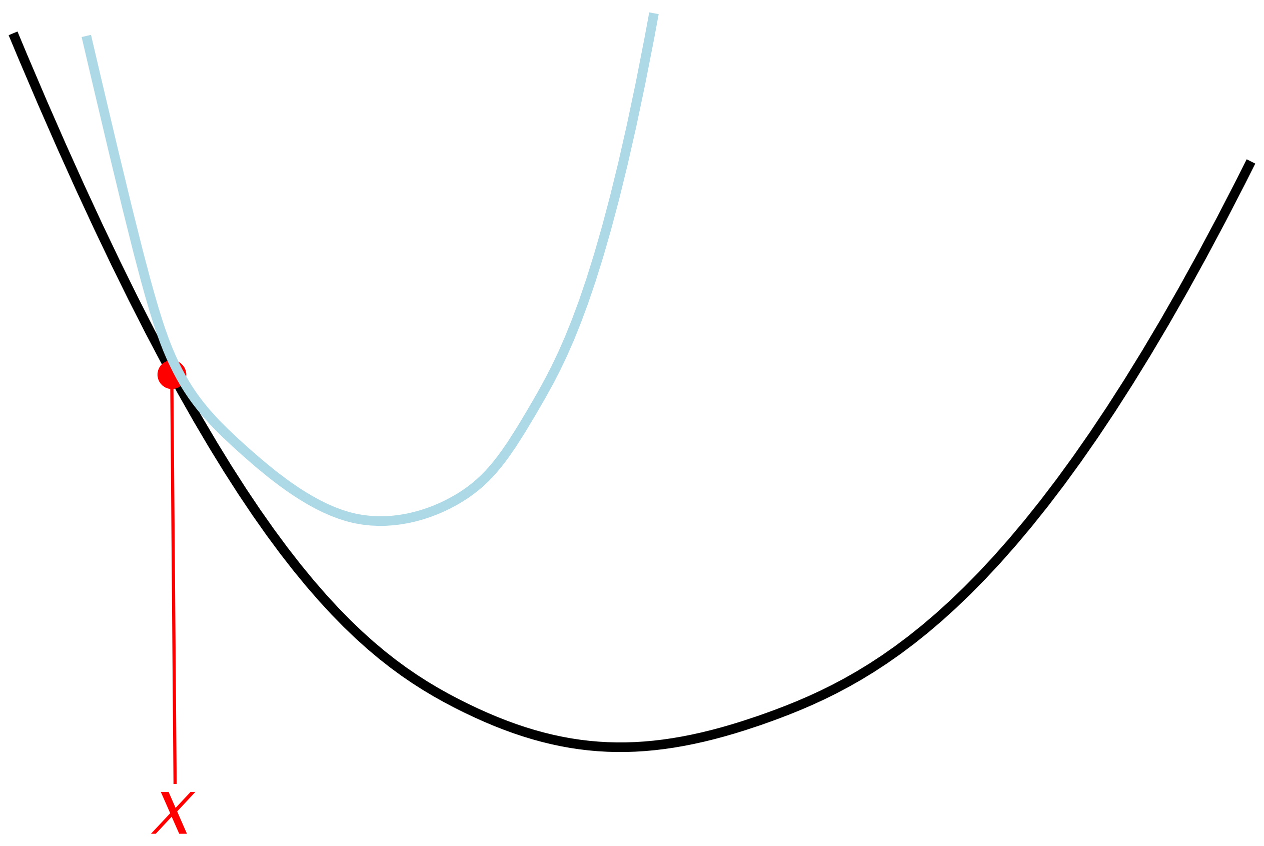
Proposition:

If f is convex and twice differentiable, it is β-smooth iff

- for all $x, y \in D(f)$,

$$f(y) \leqslant f(x) + \langle \nabla f(x), y - x \rangle + \frac{\beta}{2}\|y - x\|^2.$$

- $\nabla^2 f(x) \preceq \beta I$ for all $x \in D(f)$.



In Short:

The graph of f is **upper**-bounded by its tangent + a parabola of curvature β.

2. Convex functions.

Definition:

We say that f is β-smooth if it is differentiable and is gradient is β-Lipschitz:

$$\forall x, y \in D(f), \|\nabla f(x) - \nabla f(y)\| \leqslant \beta \|x - y\|.$$

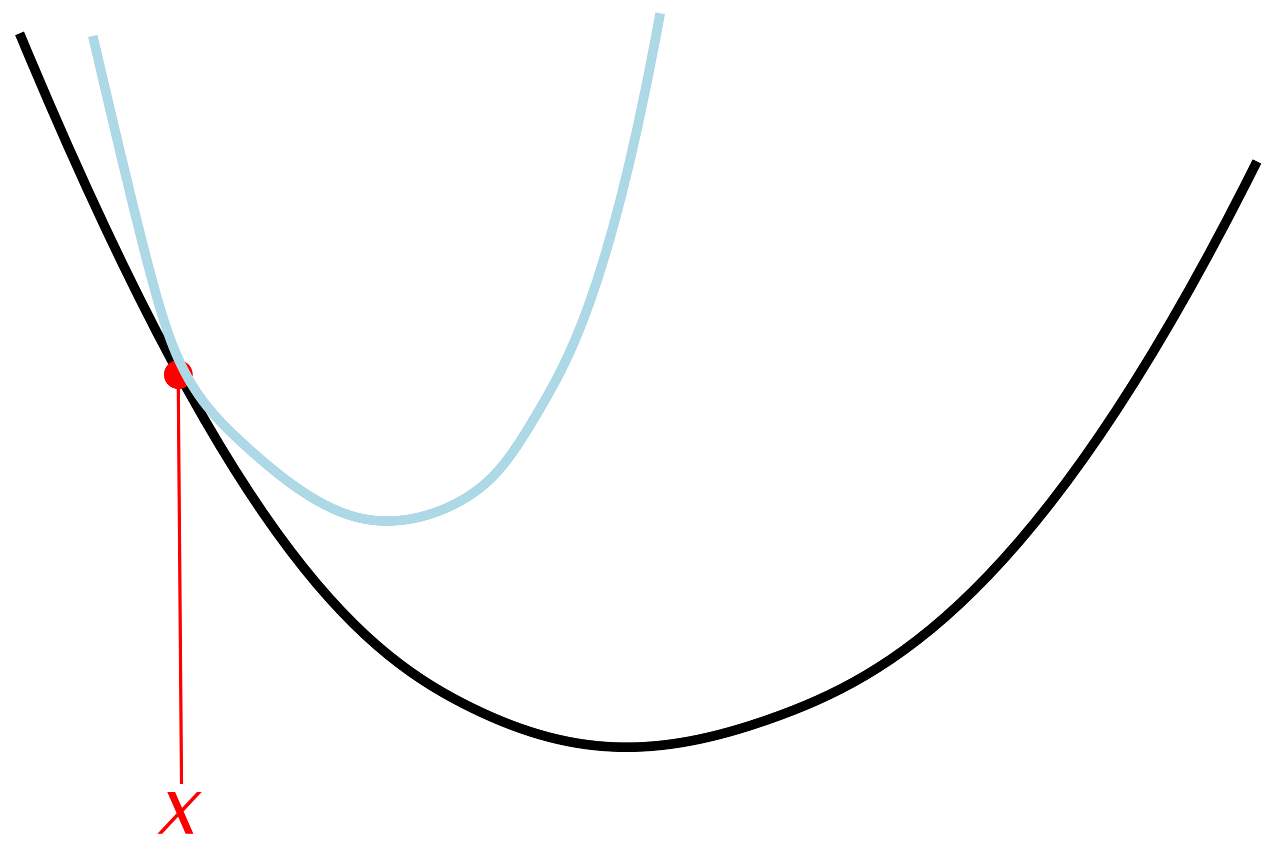
Proposition:

If f is convex and twice differentiable, it is β-smooth iff

- for all $x, y \in D(f)$,

$$f(y) \leqslant f(x) + \langle \nabla f(x), y - x \rangle + \frac{\beta}{2}\|y - x\|^2.$$

- $\nabla^2 f(x) \preceq \beta I$ for all $x \in D(f)$.



Question: What happen if we try to minimize this upper-bound (the right-hand-side term) in terms of y?

2. Convex functions.

The nicest convex functions are those being α-strongly convex and β-smooth.

Exercise: Consider $f : \mathbb{R}^2 \to \mathbb{R}$ defined by $f(x_1, x_2) \mapsto \frac{\alpha}{2} x_1^2 + \frac{\beta}{2} x_2^2$, with $\beta > \alpha$. Prove that it is α-strongly convex and β-smooth.

Exercice: Give a simple example of convex function that is
not β-smooth. Not α-strongly convex.

3. Gradient Descent for convex functions

3. Gradient Descent for convex functions

Proposition:

Let f be α-cvx and β-smooth. Let x^* be its single minimizer. Pick $\lambda_t = \lambda \leqslant \frac{1}{\beta}$ (constant). Then the GD for f with initialization x_0 and step-size λ satisfies, for all $T \in \mathbb{N}$,

$$f(x_T) - f(x^*) \leqslant (1 - \alpha\lambda)^T (f(x_0) - f(x_*)) \leqslant e^{-\alpha\lambda T}(f(x_0) - f(x^*)). \tag{9}$$

If f is only convex (not strongly), we get the slower rate (with $\lambda = \frac{1}{\beta}$), assuming that f has a global minimizer x^*,

$$f(x_T) - f(x^*) \leqslant \frac{\beta \|x_0 - x^*\|^2}{2T}.$$

3. Gradient Descent for convex functions

Proposition:

Let f be α-cvx and β-smooth. Let x^* be its single minimizer. Pick $\lambda_t = \lambda \leqslant \frac{1}{\beta}$ (constant). Then the GD for f with initialization x_0 and step-size λ satisfies, for all $T \in \mathbb{N}$,

$$f(x_T) - f(x^*) \leqslant (1 - \alpha\lambda)^T (f(x_0) - f(x_*)) \leqslant e^{-\alpha\lambda T}(f(x_0) - f(x^*)). \tag{9}$$

If f is only convex (not strongly), we get the slower rate (with $\lambda = \frac{1}{\beta}$), assuming that f has a global minimizer x^*,

$$f(x_T) - f(x^*) \leqslant \frac{\beta \|x_0 - x^*\|^2}{2T}.$$

Remark: If we pick $\lambda = \frac{1}{\beta}$ in (9), the convergence rate is exactly α/β. The ratio $\kappa = \beta/\alpha \geqslant 1$ is called the condition number of f, it is an upperbound between the largest eigenvalue of $\nabla^2 f(x)$ (at any x) and the lowest one. In particular, to get x_T such that $f(x_T) \leqslant f(x^*) + \epsilon$, we should take $T = \log(\epsilon^{-1})\kappa$.

Chapter 3: An optimization detour

3. Gradient Descent for convex functions

Proposition:

Let f be α-cvx and β-smooth. Let x^* be its single minimizer. Pick $\lambda_t = \lambda \leqslant \frac{1}{\beta}$ (constant). Then the GD for f with initialization x_0 and step-size λ satisfies, for all $T \in \mathbb{N}$,

$$f(x_T) - f(x^*) \leqslant (1 - \alpha\lambda)^T (f(x_0) - f(x_*)) \leqslant e^{-\alpha\lambda T}(f(x_0) - f(x^*)). \tag{9}$$

If f is only convex (not strongly), we get the slower rate (with $\lambda = \frac{1}{\beta}$), assuming that f has a global minimizer x^*,

$$f(x_T) - f(x^*) \leqslant \frac{\beta \|x_0 - x^*\|^2}{2T}.$$

Remark: If we pick $\lambda = \frac{1}{\beta}$ in (9), the convergence rate is exactly α/β. The ratio $\kappa = \beta/\alpha \geqslant 1$ is called the condition number of f, it is an upperbound between the largest eigenvalue of $\nabla^2 f(x)$ (at any x) and the lowest one. In particular, to get x_T such that $f(x_T) \leqslant f(x^*) + \epsilon$, we should take $T = \log(\epsilon^{-1})\kappa$.

Remark: In most applications, we do not know α, β, so the take home message is "you want to pick λ as large as possible, but if it's too large ($> \beta^{-1}$), convergence may fail".

4. Stochastic Gradient Descent.

4. Stochastic Gradient Descent.

Intuition: Training a parametric model $F(\theta, \cdot)$ on a dataset $(x_i, y_i)_{i=1}^n$ boils down to minimize a function of the form

$$L(\theta) = \frac{1}{n} \sum_{i=1}^{n} \underbrace{\ell(F(\theta, x_i), y_i)}_{f_i(\theta)}.$$

It's gradient is given by

$$\nabla L(\theta) = \frac{1}{n} \sum_{i=1}^{n} \nabla f_i(\theta).$$

Question: Do we **really** need to take into account **all** the observations $(x_i, y_i)_{i=1}^n$ at **each gradient step**, especially when n is large? (Computational efficiency.)

4. Stochastic Gradient Descent.

Intuition: Training a parametric model $F(\theta, \cdot)$ on a dataset $(x_i, y_i)_{i=1}^n$ boils down to minimize a function of the form

$$L(\theta) = \frac{1}{n} \sum_{i=1}^n \underbrace{\ell(F(\theta, x_i), y_i)}_{f_i(\theta)}.$$

It's gradient is given by

$$\nabla L(\theta) = \frac{1}{n} \sum_{i=1}^n \nabla f_i(\theta).$$

Question: Do we **really** need to take into account **all** the observations $(x_i, y_i)_{i=1}^n$ at **each gradient step**, especially when n is large? (Computational efficiency.)

Key idea: Take $j \sim \mathrm{Unif}(\{1, \dots, n\})$, and observe that

$$\mathbb{E}[\nabla f_j(\theta)] = \underbrace{\frac{1}{n}}_{\mathbb{P}(i=j)} \sum_{i=1}^n \nabla f_i(\theta) = \nabla L(\theta).$$

4. Stochastic Gradient Descent.

Algorithm:

Input: Observations–labels $(x_i, y_i), i = 1, \dots, n$. Class of model $\{F(\theta, \cdot), \theta \in \Theta\}$. Loss ℓ. Initial $\theta_0 \in \mathbb{R}^d$. Number of step $T \in \mathbb{N}$.
For $t = 1, \dots, T$,

- Pick $j \sim \mathrm{Unif}(\{1, \dots, n\})$,
- Compute $f_j(\theta_t) := \ell(F(\theta_t, x_j), y_j)$.
- Update $\theta_{t+1} = \theta_t - \lambda \nabla f_j(\theta_t)$.

Return θ_T.

Key idea: Take $j \sim \mathrm{Unif}(\{1, \dots, n\})$, and observe that

$$\mathbb{E}[\nabla f_j(\theta)] = \underbrace{\frac{1}{n}}_{\mathbb{P}(i=j)} \sum_{i=1}^{n} \nabla f_i(\theta) = \nabla L(\theta).$$

4. Stochastic Gradient Descent.

Algorithm:

Input: Observations–labels $(x_i, y_i), i = 1, \dots, n$. Class of model $\{F(\theta, \cdot), \theta \in \Theta\}$. Loss ℓ. Initial $\theta_0 \in \mathbb{R}^d$. Number of step $T \in \mathbb{N}$.
For $t = 1, \dots, T$,

- Pick $j \sim \mathrm{Unif}(\{1, \dots, n\})$,
- Compute $f_j(\theta_t) := \ell(F(\theta_t, x_j), y_j)$.
- Update $\theta_{t+1} = \theta_t - \lambda \nabla f_j(\theta_t)$.

Return θ_T.

→ This is exactly the same as a standard GD, but where we replace $\nabla L(\theta)$ by the (random) $\nabla f_j(\theta)$.

Remark: In practice, it is common to (i) randomly shuffle the dataset, (ii) split it into batches (usually of size 16 or 32), (iii) go through batches in order, compute the average gradient on the batch and update, (iv) repeat.

For hardware reasons

Once we've parse the whole dataset one time by iterating on the $n/32$ batches, we say that we ran one epoch. In many libraries (e.g. `tensorflow`), we set the number of epoch, not of iterations!

```
model.fit(train_images, train_labels, epochs=2)

Epoch 1/2
1563/1563 [==============================] - 61s 39ms/step - loss: 1.5311 - accuracy: 0.4642
Epoch 2/2
1563/1563 [==============================] - 59s 38ms/step - loss: 1.2577 - accuracy: 0.5618
```

The number of batches ($n/32$)

4. Stochastic Gradient Descent.

Algorithm:

Input: Observations–labels $(x_i, y_i), i = 1, \dots, n$. Class of model $\{F(\theta, \cdot), \theta \in \Theta\}$. Loss ℓ. Initial $\theta_0 \in \mathbb{R}^d$. Number of step $T \in \mathbb{N}$.
For $t = 1, \dots, T$,

- Pick $j \sim \mathrm{Unif}(\{1, \dots, n\})$,
- Compute $f_j(\theta_t) := \ell(F(\theta_t, x_j), y_j)$.
- Update $\theta_{t+1} = \theta_t - \lambda \nabla f_j(\theta_t)$.

Return θ_T.

Question: Why is doing an SGD a good idea?

4. Stochastic Gradient Descent.

Unbiased estimate: Recall: our goal is not exactly to minimize the empirical risk/train error $L(\theta) = \frac{1}{n}\sum_{i=1}^{n} f_i(\theta)$ ($\Rightarrow$ overfitting) but actually to minimize $\mathcal{L}(\theta) = \mathbb{E}[f_{x,y\sim\Gamma}(\theta)]$ (theoretical risk), where we assume that $(x, y) \sim \Gamma$.
Now, $\nabla f_{(x,y)}(\theta)$ is an unbiased estimate of $\nabla\mathcal{L}(\theta)$. It however has some variance (say, in dimension 1):

$$\mathrm{Var}[\nabla f_i(\theta)] = \mathbb{E}[|\nabla f_i(\theta)|^2] - \underbrace{\mathbb{E}[\nabla f_i(\theta)]^2}_{=|\nabla\mathcal{L}(\theta)|^2}.$$

In particular, at the optimum θ^* of $\mathcal{L}$, $\mathbb{E}[\nabla f_i(\theta^*)] = \nabla\mathcal{L}(\theta^*) = 0$, but the variance is

$$\mathbb{E}[|\nabla f_i(\theta^*)|^2] > 0,$$

unless we have perfect interpolation (i.e. $f_i(\theta^*) = 0$ that is $F(\theta^*, x_i) = y_i$ for all i, which is unlikely).
$\rightarrow$ around θ^*, the norm of the gradient won't go to 0, and thus the SGD won't converge.
In constrat, observe that

$$\mathbb{E}[\langle\nabla\mathcal{L}(\theta), \nabla f_i(\theta)\rangle] = \|\nabla\mathcal{L}(\theta)\|^2,$$

so "far from the optimum" (when $\|\nabla\mathcal{L}(\theta)\|^2$ is large), the scalar product is likely to be $\geqslant 0$ (assuming some regularity/concentration) $\rightarrow$ likely to draw a descent direction.

4. Stochastic Gradient Descent.

Proposition:

Assume that L is α-strongly convex, and $\mathbb{E}[f_i(\theta)^2] \leqslant \beta^2$ for some β. Consider the SGD algorithm with fixed step size $\lambda \leqslant \frac{1}{\alpha}$, with θ_t the t-th step. We have

$$\mathbb{E}[||\theta_T - \theta^*||_2^2] \leqslant (1 - \alpha\lambda)^T ||\theta_0 - \theta^*||_2^2 + \frac{\lambda}{\alpha}\beta^2. \tag{10}$$

4. Stochastic Gradient Descent.

Proposition:

Assume that L is α-strongly convex, and $\mathbb{E}[f_i(\theta)^2] \leqslant \beta^2$ for some β. Consider the SGD algorithm with fixed step size $\lambda \leqslant \frac{1}{\alpha}$, with θ_t the t-th step. We have

$$\mathbb{E}[||\theta_T - \theta^*||_2^2] \leqslant (1 - \alpha\lambda)^T ||\theta_0 - \theta^*||_2^2 + \frac{\lambda}{\alpha}\beta^2. \tag{10}$$

Interpretation: The first term will go faster to 0 if $\lambda \to \alpha^{-1}$. On the other hand, the second term (that does not go to 0) invites us to take $\lambda \to 0$ (but in that regime, the first term may fail to converge).
Conclusion: take decreasing steps. How much ?

4. Stochastic Gradient Descent.

Proposition:

Assume that L is α-strongly convex, and $\mathbb{E}[f_i(\theta)^2] \leqslant \beta^2$ for some β. Consider the SGD algorithm with fixed step size $\lambda \leqslant \frac{1}{\alpha}$, with θ_t the t-th step. We have

$$\mathbb{E}[||\theta_T - \theta^*||_2^2] \leqslant (1 - \alpha\lambda)^T ||\theta_0 - \theta^*||_2^2 + \frac{\lambda}{\alpha}\beta^2. \quad (10)$$

Proposition:

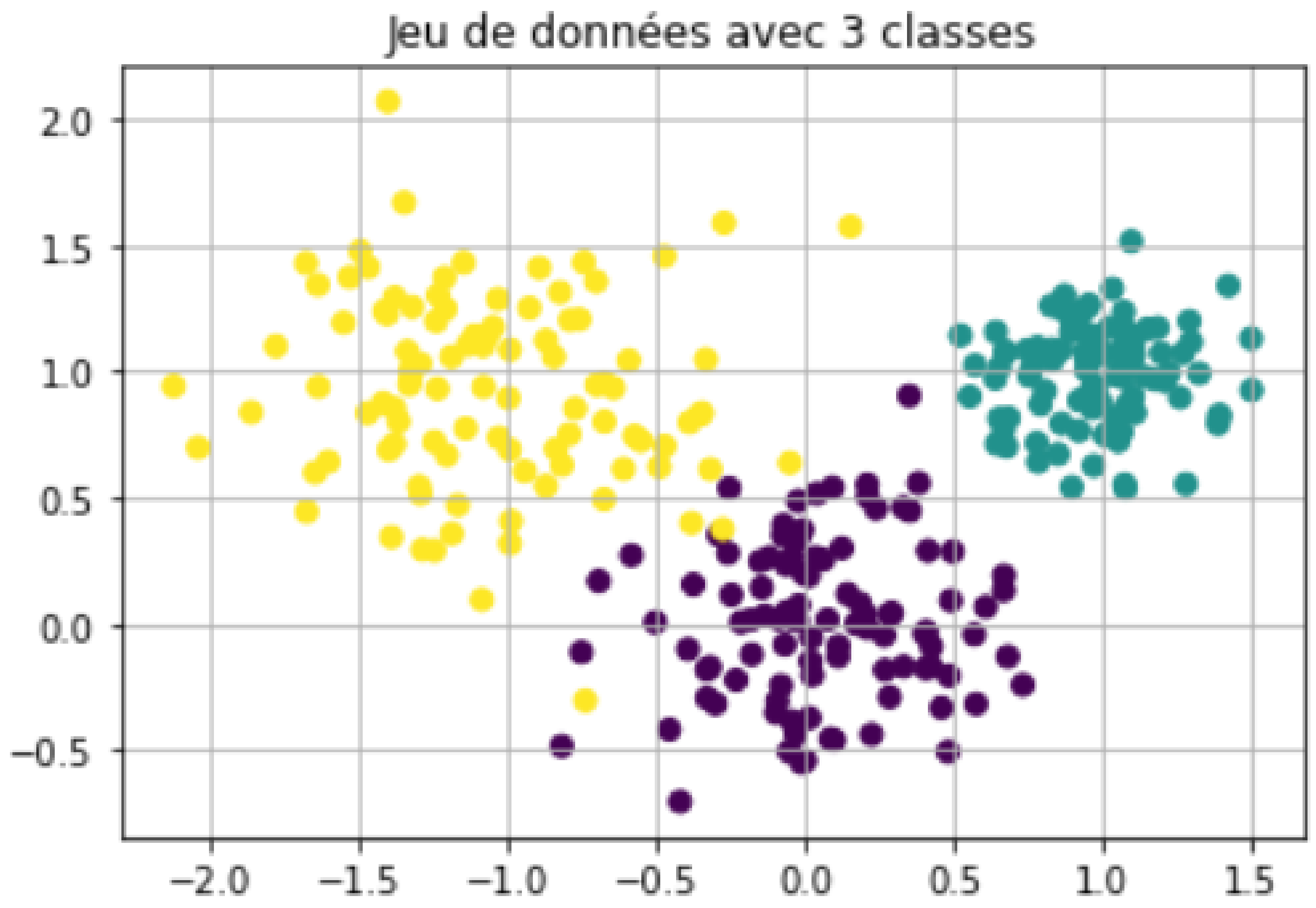
Same assumption, but now λ_t is such that $\sum_t \lambda_t = +\infty$, but $\sum_t {\lambda_t}^2 < \infty$. Then,

$$\mathbb{E}[||\theta_T - \theta^*||_2^2] \leqslant \frac{1}{\alpha \sum_{t=1}^{T} \lambda_t} \left(||\theta_0 - \theta^*||_2^2 + \frac{\sum_t {\lambda_t}^2}{\alpha}\beta^2 \right). \quad (11)$$

Chapter 4: Supervised learning (2)–Classification

This chapter is dedicated to classification tasks. Recall that it means that the labels $(y_i)_i$ take values in a **finite** set, called classes.

Example: The observations $(x_i)_i$ are images (pictures), the labels $(y_i)_i$ describe what is represented on the picture (`"car"`, `"cat"`, etc.).
A general way of thinking is the following: we try to separate points of different colors.

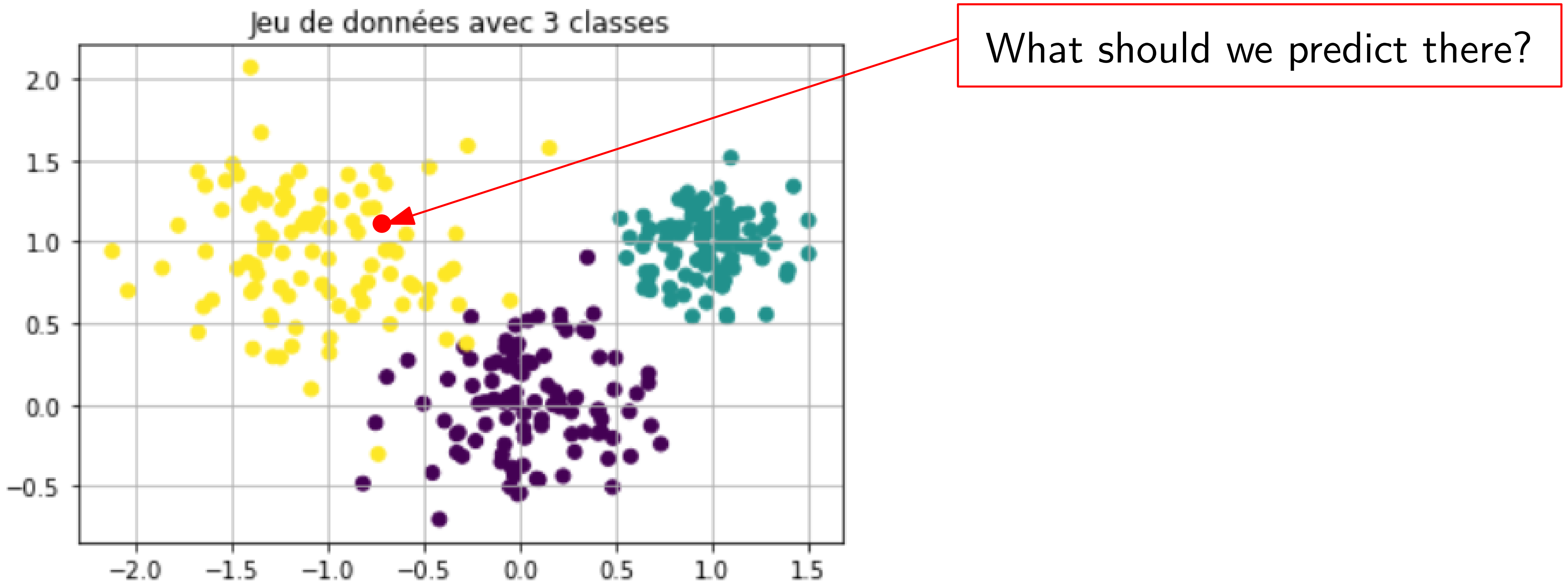


Chapter 4: Supervised learning (2)–Classification

This chapter is dedicated to classification tasks. Recall that it means that the labels $(y_i)_i$ take values in a **finite** set, called classes.

Example: The observations $(x_i)_i$ are images (pictures), the labels $(y_i)_i$ describe what is represented on the picture (`"car"`, `"cat"`, etc.).
A general way of thinking is the following: we try to separate points of different colors.

Remark: As for regression tasks, what matters is the performances of the model on **new** observations.

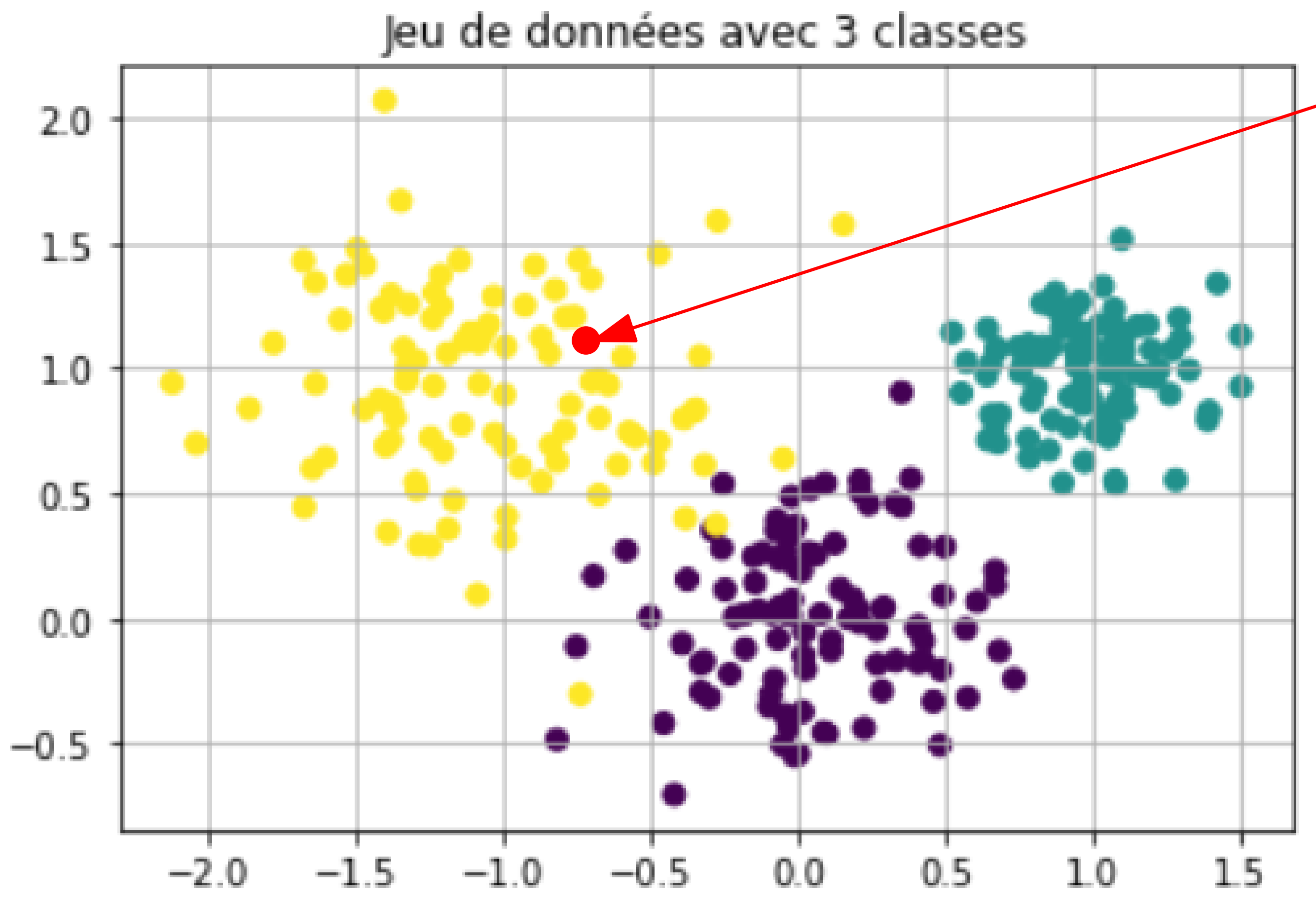


Chapter 4: Supervised learning (2)–Classification

This chapter is dedicated to classification tasks. Recall that it means that the labels $(y_i)_i$ take values in a **finite** set, called classes.

Example: The observations $(x_i)_i$ are images (pictures), the labels $(y_i)_i$ describe what is represented on the picture (`"car"`, `"cat"`, etc.).
A general way of thinking is the following: we try to separate points of different colors.

Remark: As for regression tasks, what matters is the performances of the model on **new** observations.



What should we predict there?

Question: How do we measure the performances of a classification model?

1. Accuracy

Consider observations $(x_i)_i \in \mathbb{R}^d$ and labels $(y_i)_i \in \{1, \dots, K\}$, where K is the number of classes.
The goal of a model $F : \mathbb{R}^d \to \{1, \dots, K\}$ is to satisfy, as often as possible, $F(x) = y$ for each pair of observation-label (x, y).

1. Accuracy

Consider observations $(x_i)_i \in \mathbb{R}^d$ and labels $(y_i)_i \in \{1, \dots, K\}$, where K is the number of classes.
The goal of a model $F : \mathbb{R}^d \to \{1, \dots, K\}$ is to satisfy, as often as possible, $F(x) = y$ for each pair of observation-label (x, y).

Definition:

The *accuracy* of a model F on a training set $(x_1, y_1), \dots, (x_n, y_n)$ is given by

$$\mathrm{acc}(F) = \frac{1}{n} \sum_{i=1}^{n} 1_{F(x_i)=y_i}, \tag{12}$$

where $1_{F(x_i)=y_i}$ equals 1 if $F(x_i) = y_i$ and 0 otherwise.

1. Accuracy

Consider observations $(x_i)_i \in \mathbb{R}^d$ and labels $(y_i)_i \in \{1, \dots, K\}$, where K is the number of classes.
The goal of a model $F : \mathbb{R}^d \to \{1, \dots, K\}$ is to satisfy, as often as possible, $F(x) = y$ for each pair of observation-label (x, y).

Definition:

The *accuracy* of a model F on a training set $(x_1, y_1), \dots, (x_n, y_n)$ is given by

$$\mathrm{acc}(F) = \frac{1}{n} \sum_{i=1}^{n} 1_{F(x_i)=y_i}, \tag{12}$$

where $1_{F(x_i)=y_i}$ equals 1 if $F(x_i) = y_i$ and 0 otherwise.

In Short:

We are simply counting the average proportion of "good answers" given by our model.

Remark: $\mathrm{acc}(F) \in [0, 1]$, often expressed as a **percentage**. It can be interpreted as a **probability** that the predictions made by our model are correct.

1. Accuracy

Consider observations $(x_i)_i \in \mathbb{R}^d$ and labels $(y_i)_i \in \{1, \dots, K\}$, where K is the number of classes.
The goal of a model $F : \mathbb{R}^d \to \{1, \dots, K\}$ is to satisfy, as often as possible, $F(x) = y$ for each pair of observation-label (x, y).

Definition:

The *accuracy* of a model F on a training set $(x_1, y_1), \dots, (x_n, y_n)$ is given by

$$\mathrm{acc}(F) = \frac{1}{n} \sum_{i=1}^{n} 1_{F(x_i)=y_i}, \tag{12}$$

where $1_{F(x_i)=y_i}$ equals 1 if $F(x_i) = y_i$ and 0 otherwise.

Remark: In some contexts, some errors are "worse" than others (e.g. medical prediction, autonomous car). We can account for these by slightly modifying the definition of the accuracy.

1. Accuracy

Consider observations $(x_i)_i \in \mathbb{R}^d$ and labels $(y_i)_i \in \{1, \dots, K\}$, where K is the number of classes.
The goal of a model $F : \mathbb{R}^d \to \{1, \dots, K\}$ is to satisfy, as often as possible, $F(x) = y$ for each pair of observation-label (x, y).

Definition:

The *accuracy* of a model F on a training set $(x_1, y_1), \dots, (x_n, y_n)$ is given by

$$\mathrm{acc}(F) = \frac{1}{n} \sum_{i=1}^{n} 1_{F(x_i)=y_i}, \tag{12}$$

where $1_{F(x_i)=y_i}$ equals 1 if $F(x_i) = y_i$ and 0 otherwise.

Remark: In some contexts, some errors are "worse" than others (e.g. medical prediction, autonomous car). We can account for these by slightly modifying the definition of the accuracy.

Remark: A classification model is usually called a classifier.

2. An example: the linear classifier

Just as for regression tasks, we will often consider parametric models, that is of the form F_θ for some parameter θ.

Example: Observations $x \in \mathbb{R}^d$, and labels in two classes : $y \in \{0, 1\}$. One can consider a simple adaptation of the linear regression: for $\theta \in \mathbb{R}^{d+1}$, let

$$A_\theta(x) = \theta_0 + \theta_1 x[1] + \cdots + \theta_d x[d]$$

then

$$F_\theta(x) = \begin{cases} 1 \text{ if } A_\theta(x) \geqslant 0 \\ 0 \text{ if } A_\theta(x) < 0 \end{cases} .$$

2. An example: the linear classifier

Just as for regression tasks, we will often consider parametric models, that is of the form F_θ for some parameter θ.

Example: Observations $x \in \mathbb{R}^d$, and labels in two classes : $y \in \{0, 1\}$. One can consider a simple adaptation of the linear regression: for $\theta \in \mathbb{R}^{d+1}$, let

$$A_\theta(x) = \theta_0 + \theta_1 x[1] + \cdots + \theta_d x[d]$$

then

$$F_\theta(x) = \begin{cases} 1 \text{ if } A_\theta(x) \geqslant 0 \\ 0 \text{ if } A_\theta(x) < 0 \end{cases} .$$

We see that the behavior of our model F_θ changes when $A_\theta(x) = 0$.
The set of x solving this equation defines the decision boundary of our model (the region where the model "hesitates" between 0 and 1).

2. An example: the linear classifier

Just as for regression tasks, we will often consider parametric models, that is of the form F_θ for some parameter θ.

Example: Observations $x \in \mathbb{R}^d$, and labels in two classes : $y \in \{0, 1\}$. One can consider a simple adaptation of the linear regression: for $\theta \in \mathbb{R}^{d+1}$, let

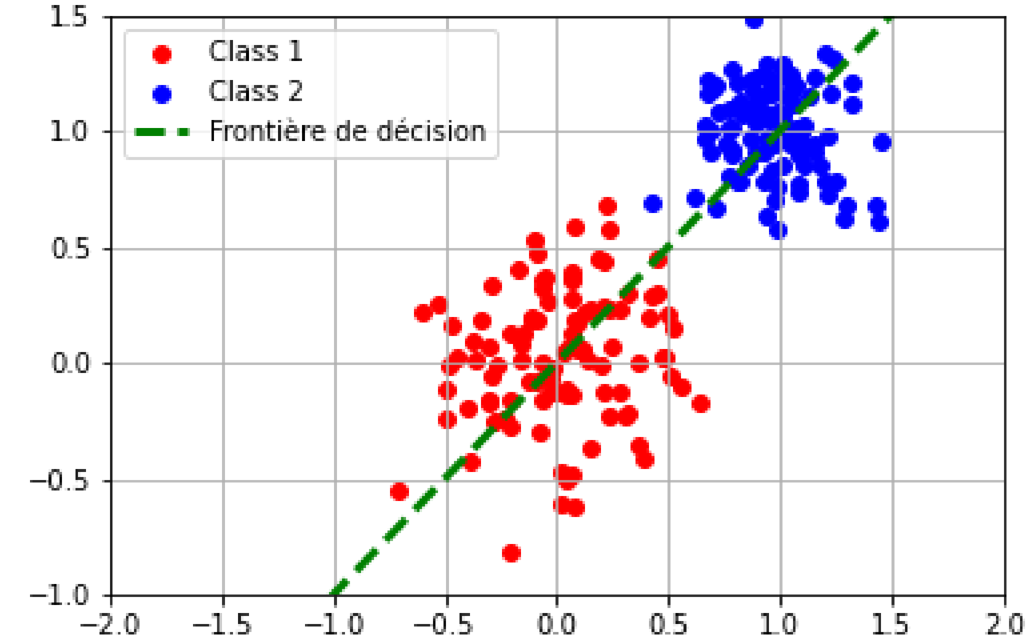
$$A_\theta(x) = \theta_0 + \theta_1 x[1] + \cdots + \theta_d x[d]$$

then

$$F_\theta(x) = \begin{cases} 1 \text{ if } A_\theta(x) \geqslant 0 \\ 0 \text{ if } A_\theta(x) < 0 \end{cases}.$$

We see that the behavior of our model F_θ changes when $A_\theta(x) = 0$. The set of x solving this equation defines the decision boundary of our model (the region where the model "hesitates" between 0 and 1).

Observation: The equation $A_\theta(x) = 0$ defines an hyperplane of $\mathbb{R}^d$.



Decision boundary obtained for a given θ.

2. An example: the linear classifier

Just as for regression tasks, we will often consider parametric models, that is of the form F_θ for some parameter θ.

Example: Observations $x \in \mathbb{R}^d$, and labels in two classes : $y \in \{0, 1\}$. One can consider a simple adaptation of the linear regression: for $\theta \in \mathbb{R}^{d+1}$, let

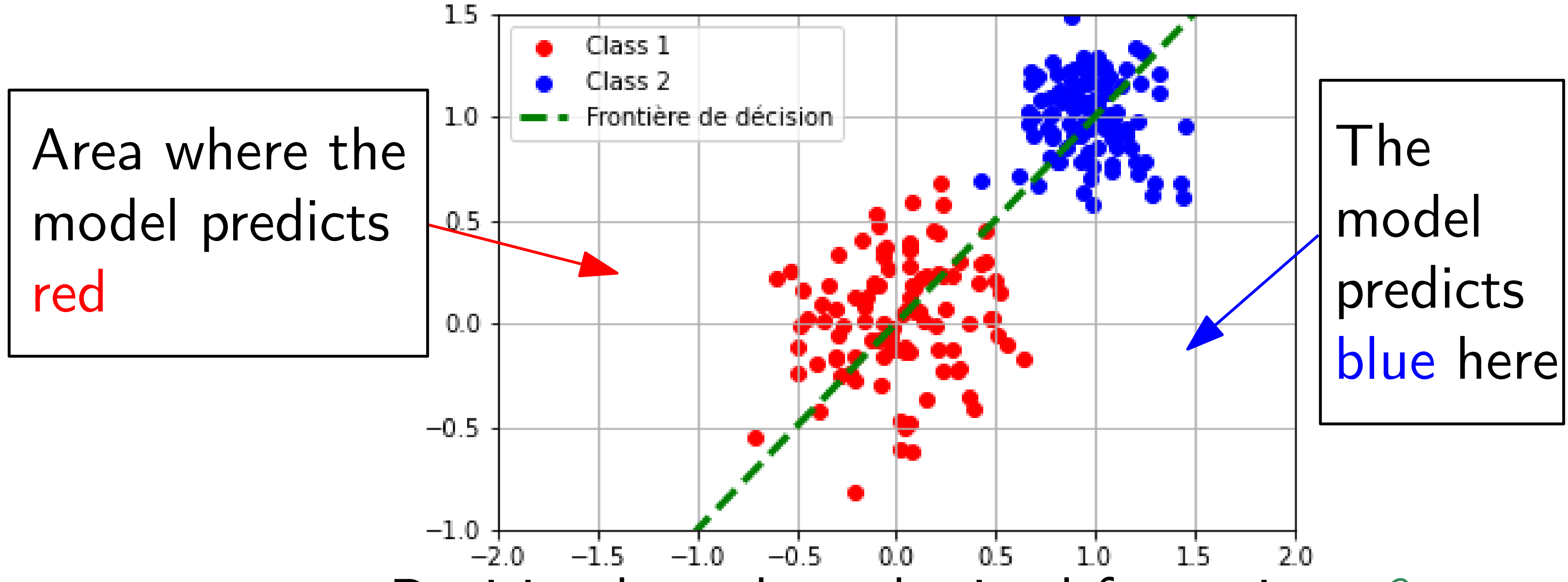
$$A_\theta(x) = \theta_0 + \theta_1 x[1] + \cdots + \theta_d x[d]$$

then

$$F_\theta(x) = \begin{cases} 1 \text{ if } A_\theta(x) \geqslant 0 \\ 0 \text{ if } A_\theta(x) < 0 \end{cases} .$$

We see that the behavior of our model F_θ changes when $A_\theta(x) = 0$. The set of x solving this equation defines the decision boundary of our model (the region where the model "hesitates" between 0 and 1).

Observation: The equation $A_\theta(x) = 0$ defines an hyperplane of $\mathbb{R}^d$.



Decision boundary obtained for a given θ.

2. An example: the linear classifier

Just as for regression tasks, we will often consider parametric models, that is of the form F_θ for some parameter θ.

Example: Observations $x \in \mathbb{R}^d$, and labels in two classes : $y \in \{0, 1\}$. One can consider a simple adaptation of the linear regression: for $\theta \in \mathbb{R}^{d+1}$, let

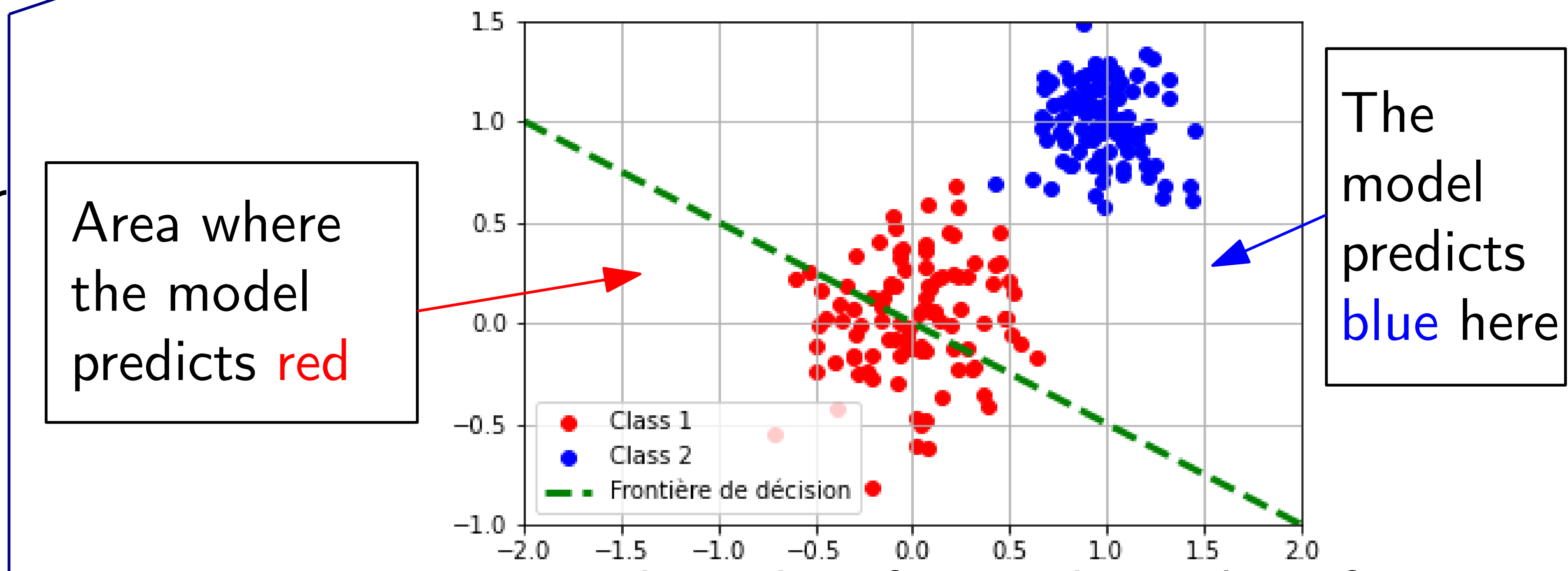
$$A_\theta(x) = \theta_0 + \theta_1 x[1] + \cdots + \theta_d x[d]$$

then

$$F_\theta(x) = \begin{cases} 1 \text{ if } A_\theta(x) \geqslant 0 \\ 0 \text{ if } A_\theta(x) < 0 \end{cases} .$$

We see that the behavior of our model F_θ changes when $A_\theta(x) = 0$. The set of x solving this equation defines the decision boundary of our model (the region where the model "hesitates" between 0 and 1).

Observation: The equation $A_\theta(x) = 0$ defines an hyperplane of $\mathbb{R}^d$.



Decision boundary for another value of θ.

2. An example: the linear classifier

Just as for regression tasks, we will often consider parametric models, that is of the form F_θ for some parameter θ.

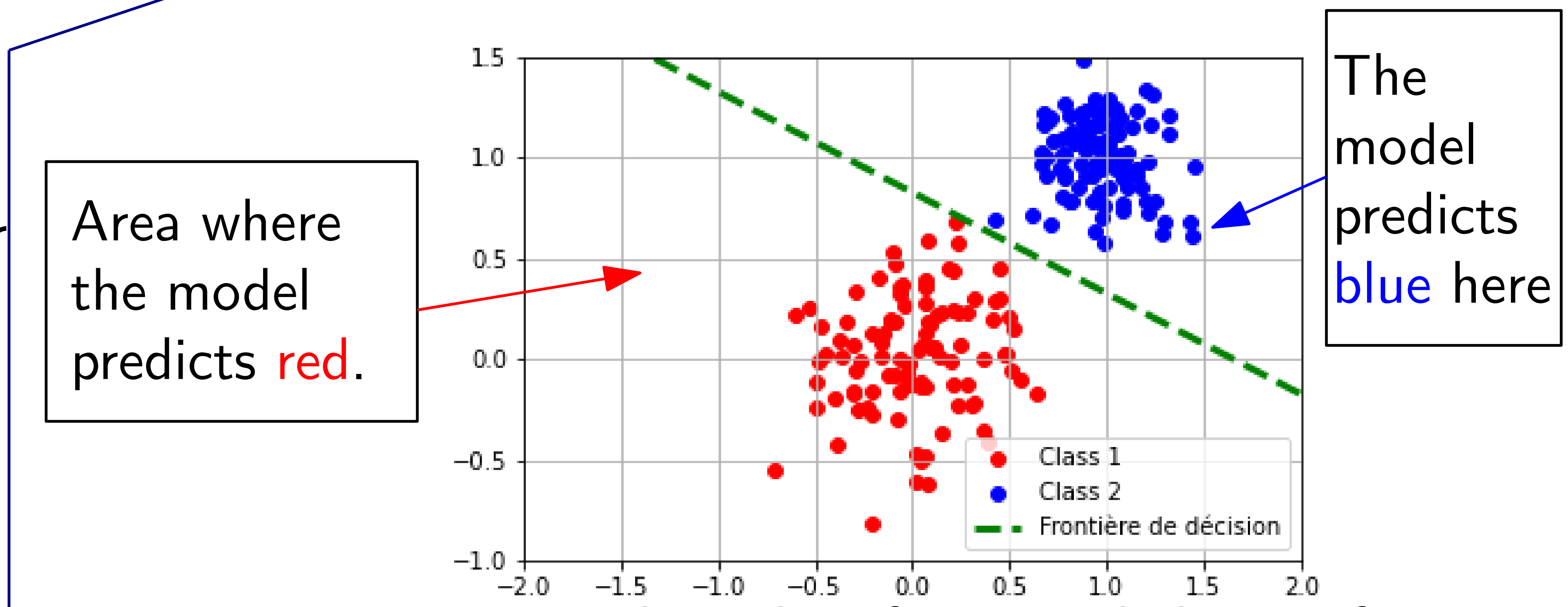
Example: Observations $x \in \mathbb{R}^d$, and labels in two classes : $y \in \{0, 1\}$. One can consider a simple adaptation of the linear regression: for $\theta \in \mathbb{R}^{d+1}$, let

$$A_\theta(x) = \theta_0 + \theta_1 x[1] + \cdots + \theta_d x[d]$$

then

$$F_\theta(x) = \begin{cases} 1 \text{ if } A_\theta(x) \geqslant 0 \\ 0 \text{ if } A_\theta(x) < 0 \end{cases}.$$

We see that the behavior of our model F_θ changes when $A_\theta(x) = 0$. The set of x solving this equation defines the decision boundary of our model (the region where the model "hesitates" between 0 and 1).
Observation: The equation $A_\theta(x) = 0$ defines an hyperplane of $\mathbb{R}^d$.



Decision boundary for a good choice of θ (accuracy $\simeq 100\%$).

2. An example: the linear classifier

Just as for regression tasks, we will often consider parametric models, that is of the form F_θ for some parameter θ.

Example: Observations $x \in \mathbb{R}^d$, and labels in two classes : $y \in \{0, 1\}$. One can consider a simple adaptation of the linear regression: for $\theta \in \mathbb{R}^{d+1}$, let

$$A_\theta(x) = \theta_0 + \theta_1 x[1] + \cdots + \theta_d x[d]$$

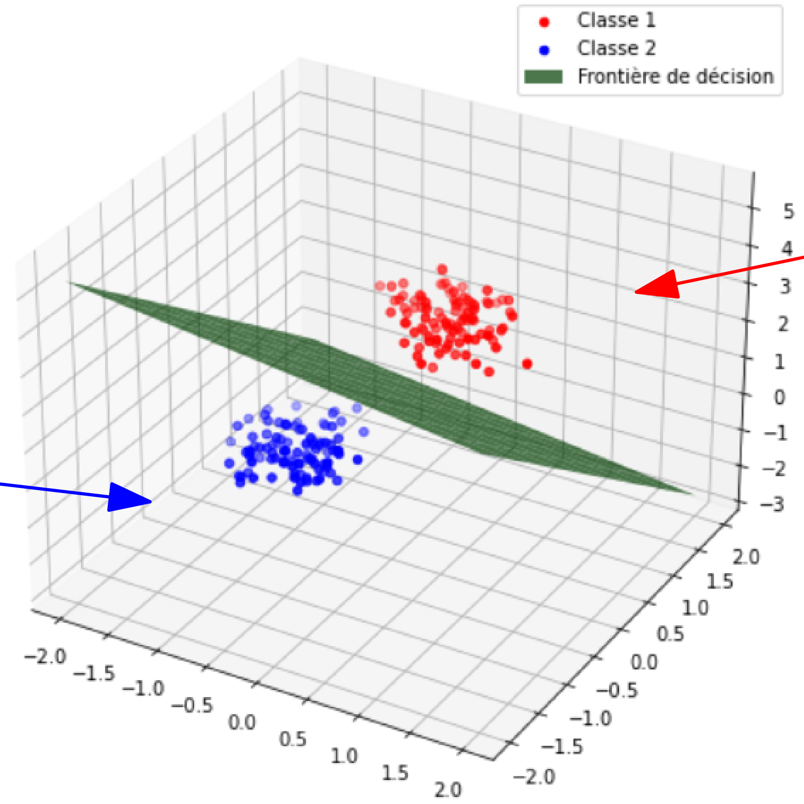
then

$$F_\theta(x) = \begin{cases} 1 \text{ if } A_\theta(x) \geqslant 0 \\ 0 \text{ if } A_\theta(x) < 0 \end{cases} .$$

We see that the behavior of our model F_θ changes when $A_\theta(x) = 0$. The set of x solving this equation defines the decision boundary of our model (the region where the model "hesitates" between 0 and 1).
Observation: The equation $A_\theta(x) = 0$ defines an hyperplane of $\mathbb{R}^d$.

Remark: in dimension 2, hyperplanes are straight lines. In dimension 3, they become 2D-planes. In higher dimension, we obtain a flat hypersurface that split the space in two areas.

The model predicts blue here



Area where the model predicts red

2. An example: the linear classifier

Just as for regression tasks, we will often consider parametric models, that is of the form F_θ for some parameter θ.

Example: Observations $x \in \mathbb{R}^d$, and labels in two classes : $y \in \{0,1\}$. One can consider a simple adaptation of the linear regression: for $\theta \in \mathbb{R}^{d+1}$, let

$$A_\theta(x) = \theta_0 + \theta_1 x[1] + \cdots + \theta_d x[d]$$

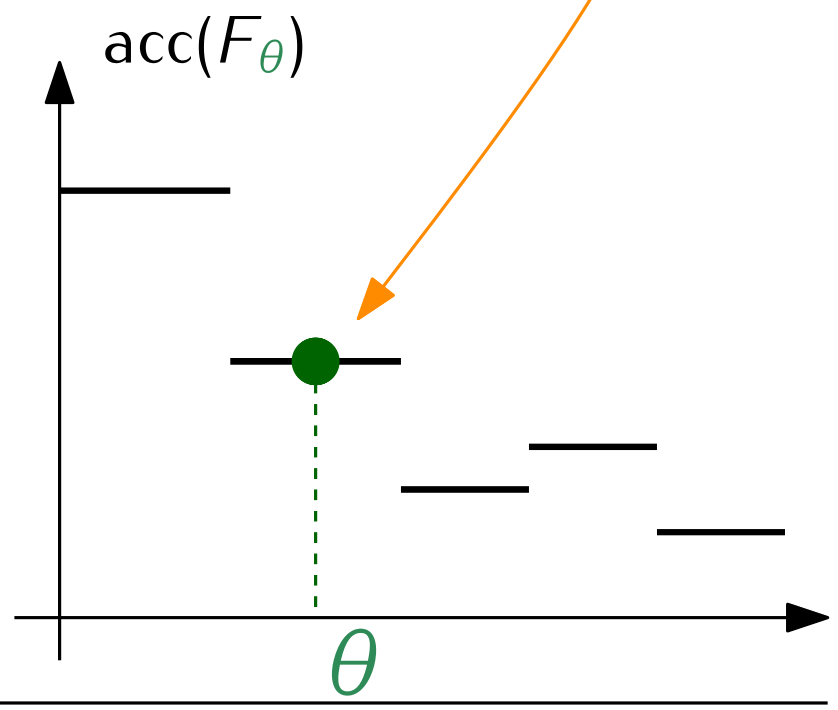
then

$$F_\theta(x) = \begin{cases} 1 \text{ if } A_\theta(x) \geqslant 0 \\ 0 \text{ if } A_\theta(x) < 0 \end{cases}.$$

No way to locally increase or reduce the objective value $\mathrm{acc}(F_\theta)$ $\Rightarrow \nabla_\theta \mathrm{acc}(F_\theta) = 0$.

Training: As for the regression problem, the goal is to optimize the parameter θ so that the model gets the best possible score (on the training set **and** on the test set). But...

1. We do not have access to a close form for the optimal θ^*.
2. We may consider using a gradient descent, but the map $\theta \mapsto \mathrm{acc}(F_\theta)$ is locally constant, thus its gradient (whenever defined) is 0, and GD won't work.



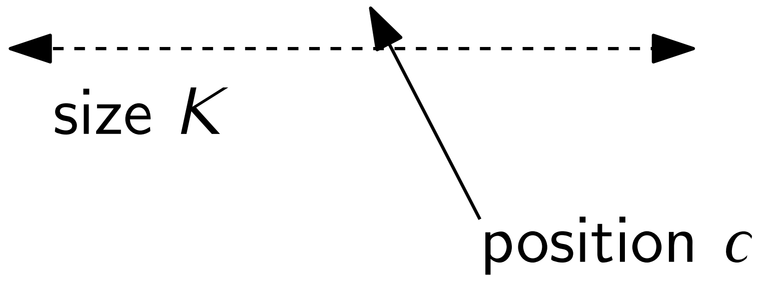
3. Training a classification model: the cross-entropy loss

In order to optimize the parameter θ of a classifier $F_\theta : \mathbb{R}^d \to Y = \{1, \dots K\}$ where K represents the number of classes, one must **modify the objective function**. The one we present below, called the cross-entropy loss (or log-loss, logistic loss...), is widely used in practice (see later for the interpretation).

3. Training a classification model: the cross-entropy loss

In order to optimize the parameter θ of a classifier $F_\theta : \mathbb{R}^d \to Y = \{1, \ldots K\}$ where K represents the number of classes, one must **modify the objective function**. The one we present below, called the cross-entropy loss (or log-loss, logistic loss...), is widely used in practice (see later for the interpretation).

Key idea: We will simply retrieve a regression task. We build a loss L such that minimizing $L \simeq$ maximizing $\mathrm{acc}(F_\theta)$.
For this, we first transform the labels: if $y = c$, with $c \in \{1, \ldots, K\}$, we build $y' = (0, \ldots, 0, 1, 0, \ldots, 0) \in \mathbb{R}^K$.

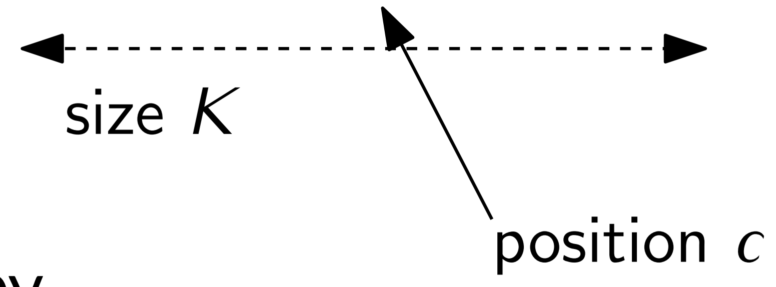


This re-definition of the labels is called *one-hot-encoding*.

3. Training a classification model: the cross-entropy loss

In order to optimize the parameter θ of a classifier $F_\theta : \mathbb{R}^d \to Y = \{1, \dots K\}$ where K represents the number of classes, one must **modify the objective function**. The one we present below, called the cross-entropy loss (or log-loss, logistic loss...), is widely used in practice (see later for the interpretation).

Key idea: We will simply retrieve a regression task. We build a loss L such that minimizing $L \simeq$ maximizing $\mathrm{acc}(F_\theta)$.
For this, we first transform the labels: if $y = c$, with $c \in \{1, \dots, K\}$, we build $y' = (0, \dots, 0, 1, 0, \dots, 0) \in \mathbb{R}^K$.



This re-definition of the labels is called *one-hot-encoding*.

Example: If we have three classes `cat`, `dog` and `pizza`, we represent the label `cat` by $(1, 0, 0)$, `dog` by $(0, 1, 0)$, and `pizza` by $(0, 0, 1)$.

3. Training a classification model: the cross-entropy loss

In order to optimize the parameter θ of a classifier $F_\theta : \mathbb{R}^d \to Y = \{1, \dots K\}$ where K represents the number of classes, one must **modify the objective function**. The one we present below, called the cross-entropy loss (or log-loss, logistic loss...), is widely used in practice (see later for the interpretation).

Key idea: We will simply retrieve a regression task. We build a loss L such that minimizing $L \simeq$ maximizing $\mathrm{acc}(F_\theta)$.
For this, we first transform the labels: if $y = c$, with $c \in \{1, \dots, K\}$, we build $y' = (0, \dots, 0, 1, 0, \dots, 0) \in \mathbb{R}^K$.

size K

position c

This re-definition of the labels is called *one-hot-encoding*.

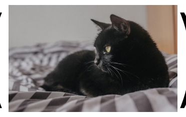
Example: If we have three classes `cat`, `dog` and `pizza`, we represent the label `cat` by $(1, 0, 0)$, `dog` by $(0, 1, 0)$, and `pizza` by $(0, 0, 1)$.

Intuition: We switch from a model that should make prediction in a **finite** set $\{1, \dots, K\}$ to a model whose outputs belong to $\mathbb{R}^K$. This is now a regression task. For instance, we want that

F_θ () $\simeq (1, 0, 0)$

3. Training a classification model: the cross-entropy loss

Probabilist viewpoint: The representation $y = (1, 0, 0)$ can be interpreted as "This object is 100% a cat". Therefore, **if** we can force our model to produce prediction on the probability simplex $\Sigma_K = \{(p_1, \dots, p_K) \in [0,1]^K, \sum_{i=1}^K p_i = 1\}$, we can interpret a given prediction $(p_1, \dots, p_K) = F_\theta(x)$ as a "probability" (or "likelyhood") that x belong to each of the classes.

For instance, if $F_\theta($$) = (0.98, 0.01, 0.01)$, it suggests that the model is 98% conviced that the input is a `cat`. If it is $(0.51, 0.49, 0)$, the model is closely hesitating between `cat` and `dog`.

To produce outputs in the probability simplex, we will use the *softmax* function.

Intuition: We switch from a model that should make prediction in a **finite** set $\{1, \dots, K\}$ to a model whose outputs belong to $\mathbb{R}^K$. This is now a regression task. For instance, we want that

$F_\theta\Bigg($  $\Bigg) \simeq (1, 0, 0)$

3. Training a classification model: the cross-entropy loss

Probabilist viewpoint: The representation $y = (1, 0, 0)$ can be interpreted as "This object is 100% a cat". Therefore, **if** we can force our model to produce prediction on the probability simplex $\Sigma_K = \{(p_1, \dots, p_K) \in [0, 1]^K, \sum_{i=1}^K p_i = 1\}$, we can interpret a given prediction $(p_1, \dots, p_K) = F_\theta(x)$ as a "probability" (or "likelyhood") that x belong to each of the classes.

For instance, if $F_\theta($$) = (0.98, 0.01, 0.01)$, it suggests that the model is 98% conviced that the input is a cat. If it is $(0.51, 0.49, 0)$, the model is closely hesitating between cat and dog.

To produce outputs in the probability simplex, we will use the *softmax* function.

Definition:

The *softmax* is defined as :

$$\begin{aligned} \mathrm{smax} : \quad \mathbb{R}^K &\to \Sigma_K \\ (f_1, \dots, f_K) &\mapsto \left(\frac{e^{f_1}}{\sum_{j=1}^K e^{f_j}}, \dots, \frac{e^{f_K}}{\sum_{j=1}^K e^{f_j}} \right). \end{aligned} \tag{13}$$

3. Training a classification model: the cross-entropy loss

Last step: Applying the softmax to the predictions $F_\theta(x)$ provides elements in Σ_K, which should be compared to the actual (one-hot-encoded) labels of the form $(0, \dots, 0, 1, 0, \dots, 0)$.
One may consider using the MSE, but as explained later, it is much better to use the cross-entropy loss.

Definition:

Formally, the cross-entropy loss is defined as:

$$L(\theta) = -\frac{1}{N} \sum_{i=1}^{N} y_i \cdot \log[\mathrm{smax}(F_\theta(x_i))] \tag{14}$$

where

- θ represents the parameters of the model,
- The log is applied term-wise.

The predictions of the model (before applying the softmax) are called logits.

3. Training a classification model: the cross-entropy loss

• Understanding the CE loss as a Maximum Likelihood Estimation. Consider the classification problem with K classes with data (x, y). Our goal is to estimate the **probability distribution** $\{\mathbb{P}(y = c|x), c = 1, \dots, K\} \in \Sigma_K \subset \mathbb{R}^K$. For clarity, let $\mathbf{y}$ denote the one-hot-encoding of y.

Now, we seek for a model F such that $\mathbb{P}(y = c|x) = \mathrm{smax}(F(x))[c]$. This can be compactly summarized as:

$$\mathbb{P}(y|x) = \mathbf{y} \cdot \mathrm{smax}(F(x)).$$

Now, the likelihood of observing an i.i.d. sample $(x_i, y_i)_i$ is given by

$$\mathbb{P}((x_i, y_i)_{i=1}^N) = \prod_{i=1}^N \mathbb{P}(x_i, y_i) = \prod_{i=1}^N \mathbb{P}(y_i|x_i)\mathbb{P}(x_i).$$

Indep. of F.

Maximizing the likelihood boils down to find F that minimizes the quantity

$$-\sum_{i=1}^N \mathbf{y}_i \cdot \log[\mathrm{smax}(F(x_i))].$$

Note that we used that $\log(\mathbf{y}_i \cdot \mathrm{smax}(F(x_i))) = \mathbf{y}_i \cdot \log[\mathrm{smax}(F(x_i))]$ (termwise log).

3. Training a classification model: the cross-entropy loss

In Short:

To train a classifier:

1. Change the representation of the labels: $y = k$ becomes $\mathbf{y} = (0, \dots, 0, \underbrace{1}_{\text{position } c}, 0, \dots, 0)$: this is the one-hot encoding.
2. Turn the output of your models (the logits) to probability distribution $(p_1, \dots, p_K)$, using the softmax.
3. Use as objective function the *cross-entropy loss*—akin to a maximum likelihood estimator—and minimize it on the training set.
4. Assess the "practical" performance of your model by evaluating its accuracy on the training **and** test sets. Even though we did not exactly optimized the accuracy (but a "smoothed" version of it), empirically having a low cross-entropy yields a high accuracy.

3. Training a classification model: the cross-entropy loss

In Short:

To train a classifier:

1. Change the representation of the labels: $y = k$ becomes $\mathbf{y} = (0, \dots, 0, \underbrace{1}_{\text{position } c}, 0, \dots, 0)$: this is the one-hot encoding.
2. Turn the output of your models (the logits) to probability distribution $(p_1, \dots, p_K)$, using the softmax.
3. Use as objective function the *cross-entropy loss*—akin to a maximum likelihood estimator—and minimize it on the training set.
4. Assess the "practical" performance of your model by evaluating its accuracy on the training **and** test sets. Even though we did not exactly optimized the accuracy (but a "smoothed" version of it), empirically having a low cross-entropy yields a high accuracy.

In practice: No worries, all these steps can be performed by using the methods provided by standard libraries. Nonetheless, for this, you need to know that they exist, their names, etc.

4. Application: the logistic regression

This is probably the most famous classification model!

4. Application: the logistic regression

This is probably the most famous classification model!

For the sake of simplicity, consider first a binary classification problem $Y = \{0, 1\}$, with observations $x \in \mathbb{R}^d$. Let $w \in \mathbb{R}^{d+1}$, and let

$$A_w(x) = w \cdot \mathbf{x}, \qquad \text{which can also be written } \langle w, \mathbf{x} \rangle \text{ or } w^T \mathbf{x},$$

where $\mathbf{x} = (1, x) \in \mathbb{R}^{d+1}$ (recall, the "augmented" observation).

4. Application: the logistic regression

This is probably the most famous classification model!

For the sake of simplicity, consider first a binary classification problem $Y = \{0, 1\}$, with observations $x \in \mathbb{R}^d$. Let $w \in \mathbb{R}^{d+1}$, and let

$$A_w(x) = w \cdot \mathbf{x}, \qquad \text{which can also be written } \langle w, \mathbf{x} \rangle \text{ or } w^T \mathbf{x},$$

where $\mathbf{x} = (1, x) \in \mathbb{R}^{d+1}$ (recall, the "augmented" observation).

Definition:

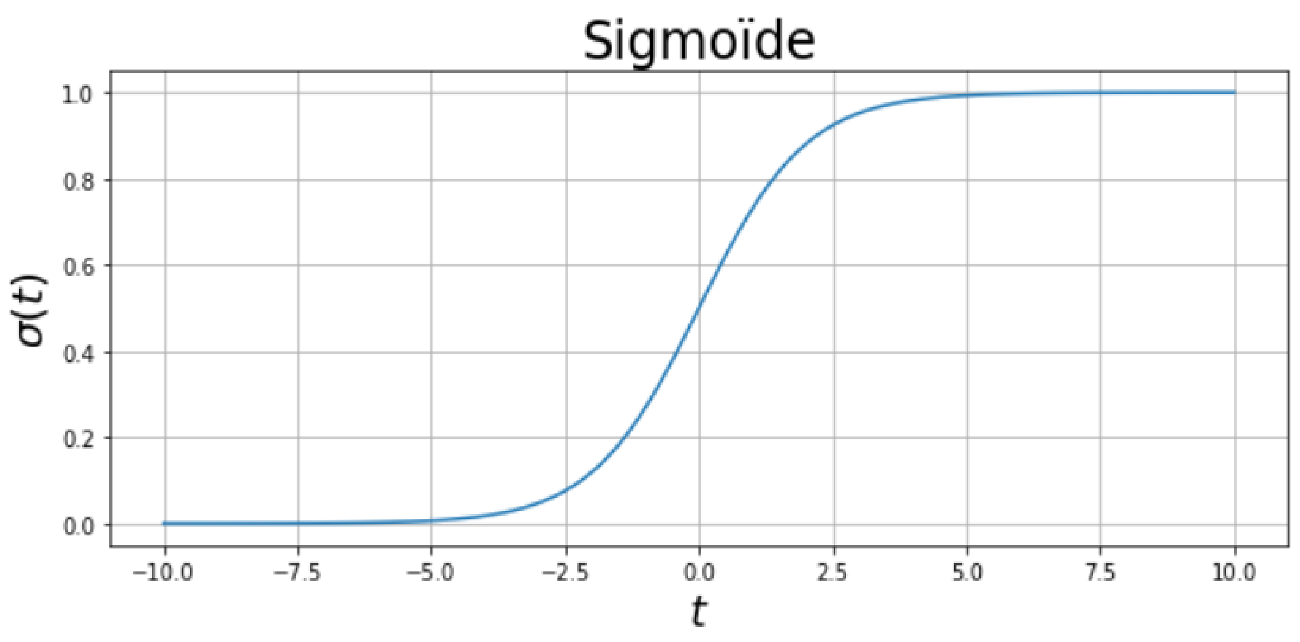
We define the sigmoid (or logistic) function:

$$\forall t \in \mathbb{R}, \quad \sigma(t) = \frac{1}{1 + e^{-t}} \in]0, 1[. \tag{15}$$

The Logistic Regression with parameter w is defined as

$$\mathrm{LogReg}_w(x) = \sigma(w \cdot \mathbf{x}). \tag{16}$$

If $\mathrm{LogReg}_w(x) \geqslant \frac{1}{2}$, we eventually predict 1, otherwise 0.



4. Application: the logistic regression

It is trained by minimizing the loss

$$L(w) = -\frac{1}{N}\left(\sum_{i:y_i=1} \log(\sigma(w \cdot \mathbf{x}_i)) + \sum_{i:y_i=0} \log(1 - \sigma(w \cdot \mathbf{x}_i))\right) \quad (17)$$

Exercise : Check that this is equivalent to training a linear model with the cross-entropy.

Definition:

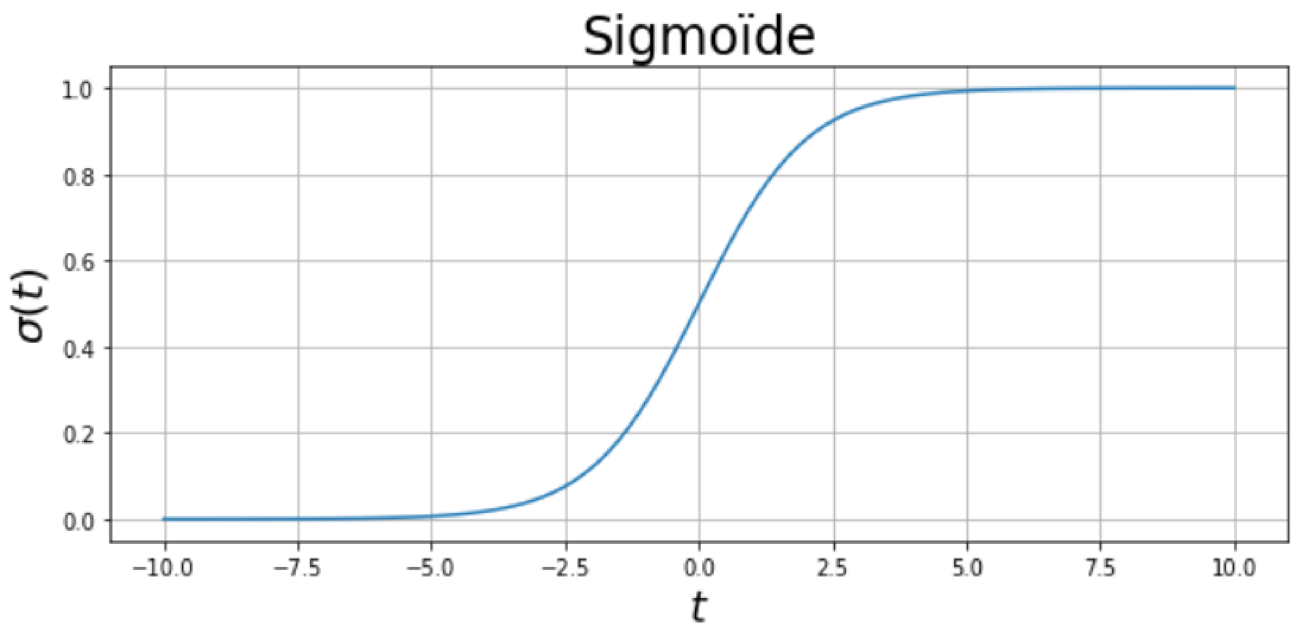
We define the sigmoid (or logistic) function:

$$\forall t \in \mathbb{R}, \quad \sigma(t) = \frac{1}{1+e^{-t}} \in]0,1[. \quad (15)$$

The Logistic Regression with parameter w is defined as

$$\mathrm{LogReg}_w(x) = \sigma(w \cdot \mathbf{x}). \quad (16)$$

If $\mathrm{LogReg}_w(x) \geqslant \frac{1}{2}$, we eventually predict 1, otherwise 0.



4. Application: the logistic regression

It is trained by minimizing the loss

$$L(w) = -\frac{1}{N}\left(\sum_{i:y_i=1} \log(\sigma(w \cdot \mathbf{x}_i)) + \sum_{i:y_i=0} \log(1-\sigma(w \cdot \mathbf{x}_i))\right) \tag{17}$$

Optimization: There is no closed form for the optimal w^* that would minimize this loss on a training set.
It is typically a situation where one relies on Gradient Descent (or more sophisticated methods, like second order solver, called Newton method, etc.).

4. Application: the logistic regression

It is trained by minimizing the loss

$$L(w) = -\frac{1}{N}\left(\sum_{i:y_i=1} \log(\sigma(w \cdot \mathbf{x}_i)) + \sum_{i:y_i=0} \log(1-\sigma(w \cdot \mathbf{x}_i))\right) \quad (17)$$

Optimization: There is no closed form for the optimal w^* that would minimize this loss on a training set.
It is typically a situation where one relies on Gradient Descent (or more sophisticated methods, like second order solver, called Newton method, etc.).

In practice: `scikit-learn.linear_model.LogisticRegression` allows you to set up a logistic regression easily.
More generally, the cross-entropy loss (and its gradient!) is provided by most of machine learning libraries: `sklearn.metrics.log_loss`, `tensorflow.keras.losses.BinaryCrossentropy`, `torch.nn.CrossEntropyLoss`, etc.

```
model = LogisticRegression()
model.fit(X_train,y_train)
```

```
score_train = model.score(X_train, y_train)
print("Score de notre model sur le jeu d'entraînement: %.2f %%" %(100*score_train))
```

```
Score de notre model sur le jeu d'entraînement: 99.00 %
```

```
score_test = model.score(X_test, y_test)
print("Score de notre model sur le jeu de validation: %.2f %%" %(100*score_test))
```

```
Score de notre model sur le jeu de validation: 99.00 %
```

4. Application: the logistic regression

The model can be adapted to more than two classes, in which case it is often called multinomial logistic regression. Formally, our model is simply given by

$$F_w(x) = w \cdot \mathbf{x},$$

for which the cross-entropy loss reads

$$L : w \mapsto -\frac{1}{N}\sum_{i=1}^{N} \mathbf{y}_i \cdot \log[\mathrm{smax}(w \cdot \mathbf{x}_i)].$$

4. Application: the logistic regression

The model can be adapted to more than two classes, in which case it is often called multinomial logistic regression. Formally, our model is simply given by

$$F_w(x) = w \cdot \mathbf{x},$$

for which the cross-entropy loss reads

$$L : w \mapsto -\frac{1}{N} \sum_{i=1}^{N} \mathbf{y}_i \cdot \log[\mathrm{smax}(w \cdot \mathbf{x}_i)].$$

Proposition:

This function is convex.

Exercise: Prove it.

5. Limits of linear models

5. Limits of linear models

Definition:

A (binary) classifier is said to be linear if its decision boundary is an hyperplane, that is characterized by an equation of the form $Ax + b = 0$.

Example: The prediction of a logistic regression changes whenever $\sigma(A_\theta(x)) = \frac{1}{2}$, where we recall that $A_\theta(x) = \theta_0 + \theta_1 x[1] + \cdots + \theta_d x[d]$ and $\sigma(t) = \frac{1}{1+e^{-t}}$. Observe that $\sigma(t) = \frac{1}{2} \Leftrightarrow t = 0$, so that $A_\theta(x) = 0$, which is an hyperplane.

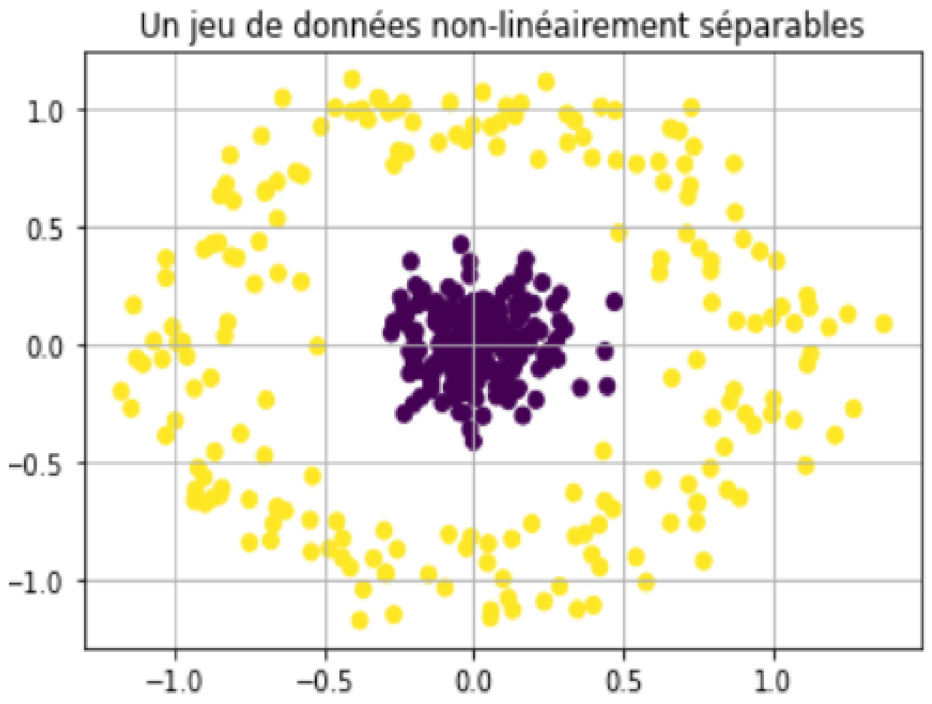
5. Limits of linear models

Definition:

A (binary) classifier is said to be linear if its decision boundary is an hyperplane, that is characterized by an equation of the form $Ax + b = 0$.

Example: The prediction of a logistic regression changes whenever $\sigma(A_\theta(x)) = \frac{1}{2}$, where we recall that $A_\theta(x) = \theta_0 + \theta_1 x[1] + \cdots + \theta_d x[d]$ and $\sigma(t) = \frac{1}{1+e^{-t}}$. Observe that $\sigma(t) = \frac{1}{2} \Leftrightarrow t = 0$, so that $A_\theta(x) = 0$, which is an hyperplane.

Issue: Hyperplanes split the Euclidean space $\mathbb{R}^d$ in **two halves** with a **flat** boundary. They can only achieve good performances on data that are (mostly) linearly separable.



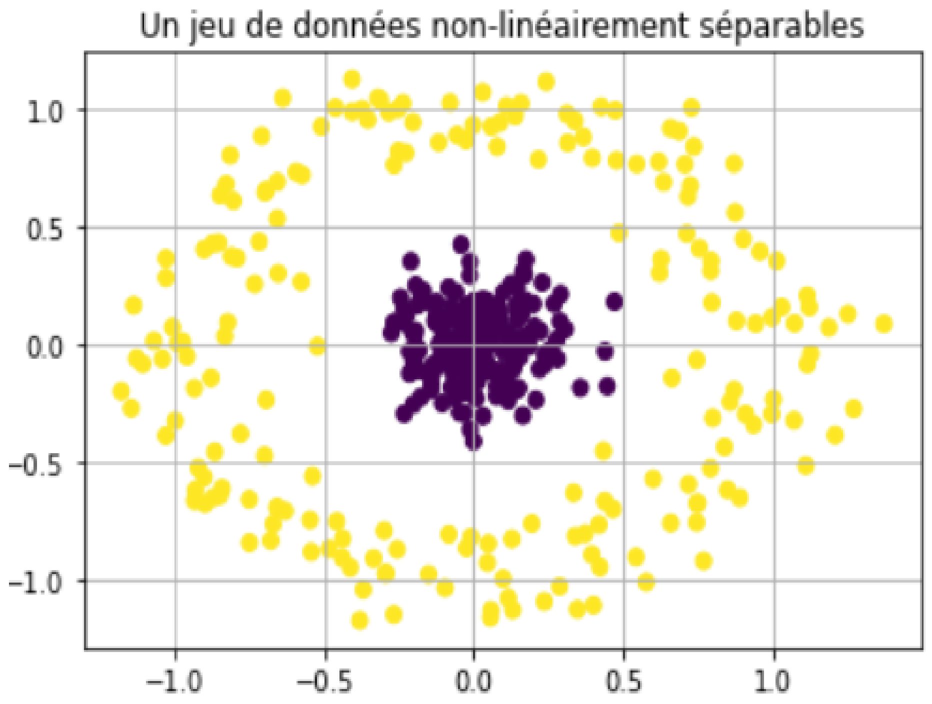
5. Limits of linear models

Definition:

A (binary) classifier is said to be linear if its decision boundary is an hyperplane, that is characterized by an equation of the form $Ax + b = 0$.

Example: The prediction of a logistic regression changes whenever $\sigma(A_\theta(x)) = \frac{1}{2}$, where we recall that $A_\theta(x) = \theta_0 + \theta_1 x[1] + \cdots + \theta_d x[d]$ and $\sigma(t) = \frac{1}{1+e^{-t}}$. Observe that $\sigma(t) = \frac{1}{2} \Leftrightarrow t = 0$, so that $A_\theta(x) = 0$, which is an hyperplane.

Issue: Hyperplanes split the Euclidean space $\mathbb{R}^d$ in **two halves** with a **flat** boundary. They can only achieve good performances on data that are (mostly) linearly separable.




```
model = LogisticRegression()
model.fit(X,y)
print("Score du modèle sur le jeu d'entraînement : %.2f %%" %(100*model.score(X,y)))
```

```
Score du modèle sur le jeu d'entraînement : 50.50 %
```

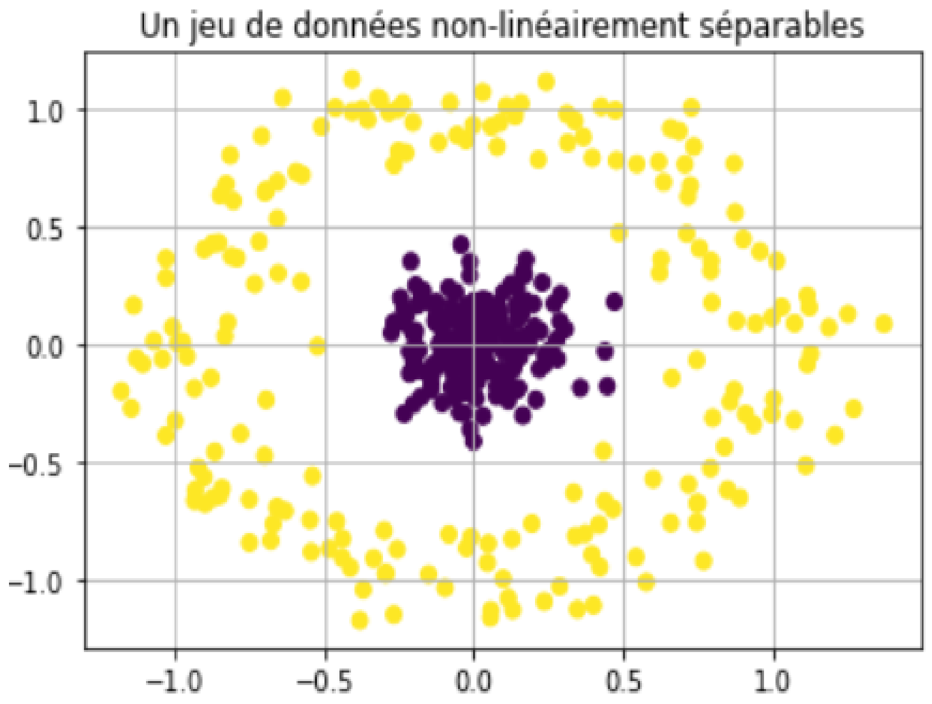
5. Limits of linear models

Definition:

A (binary) classifier is said to be linear if its decision boundary is an hyperplane, that is characterized by an equation of the form $Ax + b = 0$.

Example: The prediction of a logistic regression changes whenever $\sigma(A_\theta(x)) = \frac{1}{2}$, where we recall that $A_\theta(x) = \theta_0 + \theta_1 x[1] + \cdots + \theta_d x[d]$ and $\sigma(t) = \frac{1}{1+e^{-t}}$. Observe that $\sigma(t) = \frac{1}{2} \Leftrightarrow t = 0$, so that $A_\theta(x) = 0$, which is an hyperplane.

Issue: Hyperplanes split the Euclidean space $\mathbb{R}^d$ in **two halves** with a **flat** boundary. They can only achieve good performances on data that are (mostly) linearly separable.



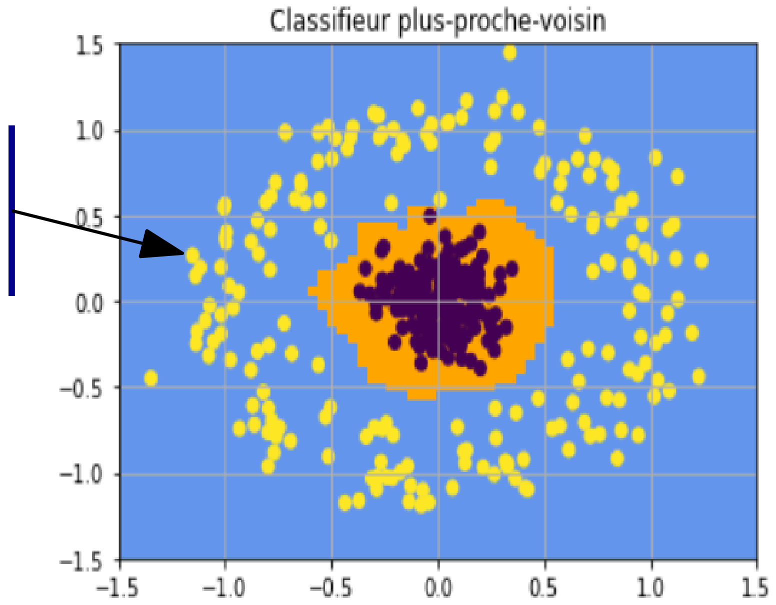

```
model = LogisticRegression()
model.fit(X,y)
print("Score du modèle sur le jeu d'entraînement : %.2f %%" %(100*model.score(X,y)))
```

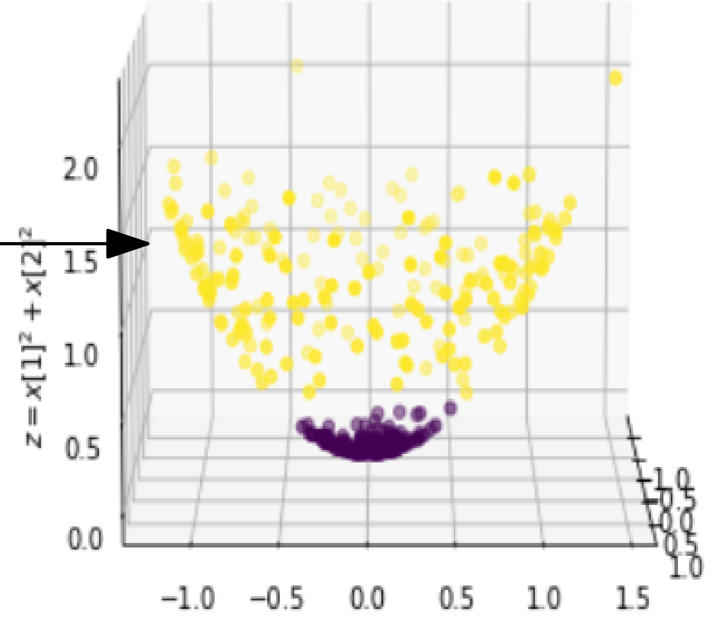
```
Score du modèle sur le jeu d'entraînement : 50.50 %
```

Remark: 50.5% is a disastrous accuracy for a binary classifier (between two balanced classes, that is with the same number of observations in each class). This is (almost) the score that would reach a trivial classifier predicting always 1 (or always 0).

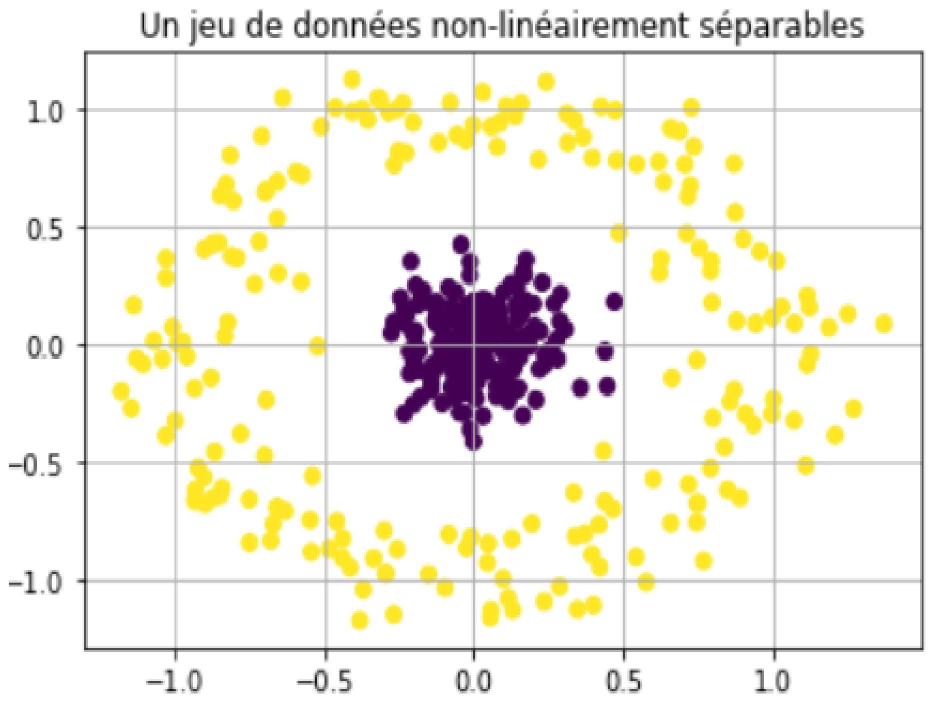
5. Limits of linear models

- Some possible workarounds:
  - Transform our data by **augmenting the dimension** in order to make them linearly separable.
  - Use non-linear model, as the nearest-neighbor method, or the celebrated neural networks (see the course of the second semester !).





Issue: Hyperplanes split the Euclidean space $\mathbb{R}^d$ in **two halves** with a **flat** boundary. They can only achieve good performances on data that are (mostly) linearly separable.




```
model = LogisticRegression()
model.fit(X,y)
print("Score du modèle sur le jeu d'entraînement : %.2f %%" %(100*model.score(X,y)))
```

```
Score du modèle sur le jeu d'entraînement : 50.50 %
```

Remark: 50.5% is a disastrous accuracy for a binary classifier (between two balanced classes, that is with the same number of observations in each class). This is (almost) the score that would reach a trivial classifier predicting always 1 (or always 0).

Chapter 5: Unsupervised learning

This chapter is dedicated to two elementary methods in unsupervised learning: the k-means problem and the Principal Component Analysis (PCA). Few words are also given about autoencoders.

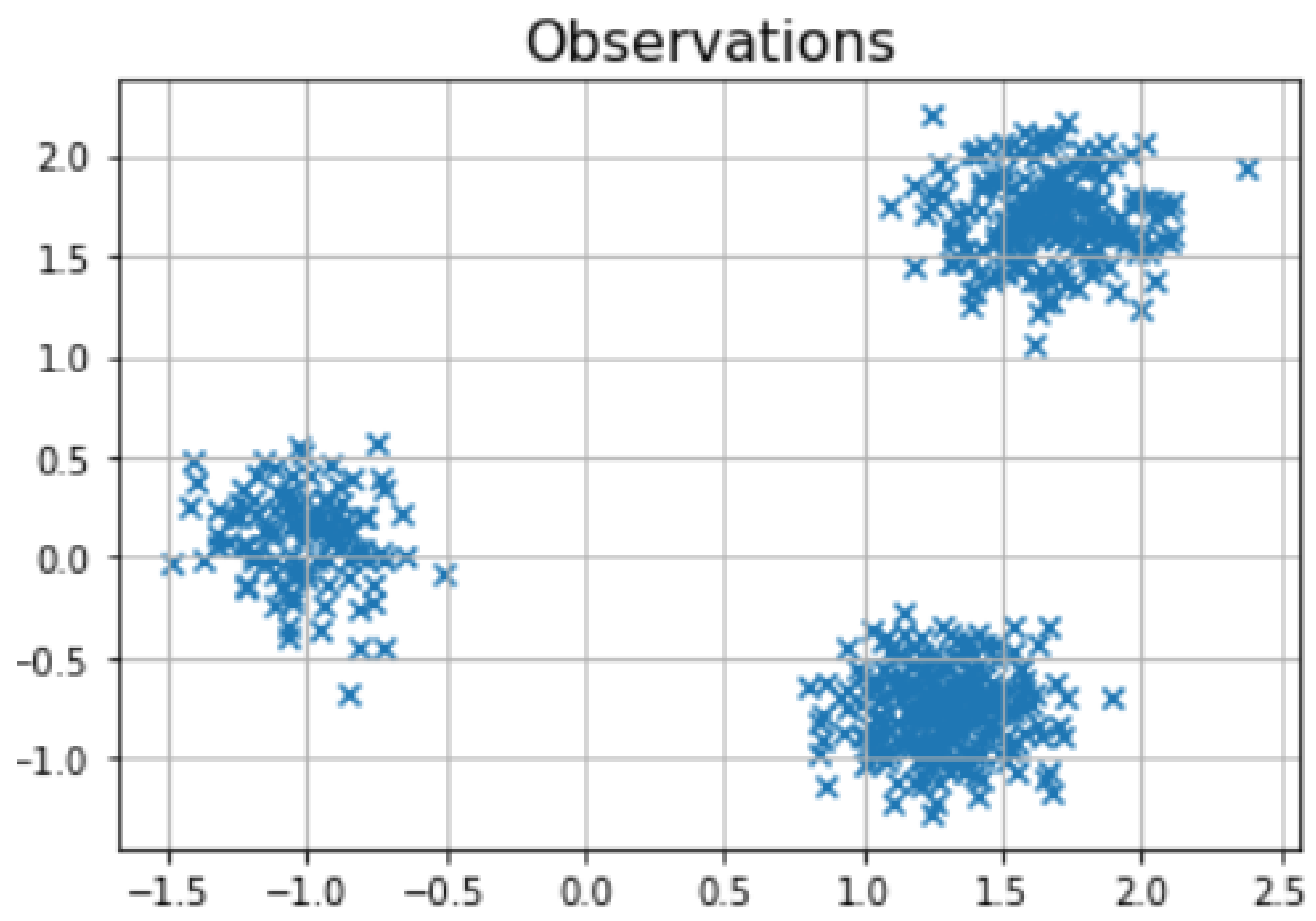
Recall: Unsupervised learning problems are problems where no labels are accessible.

Chapter 5: Unsupervised learning

1.1. The k-means problem.

Idea: Consider a set of observations $\{x_1, \dots, x_n\}$ in $\mathbb{R}^d$, and an integer $k \in \mathbb{N}$.
The main goal of a clustering algorithm is to gather the observations in k groups (clusters) such that:

- Observations belonging to a same cluster should be close to each other,
- Observations belonging to different clusters should be far from each other.



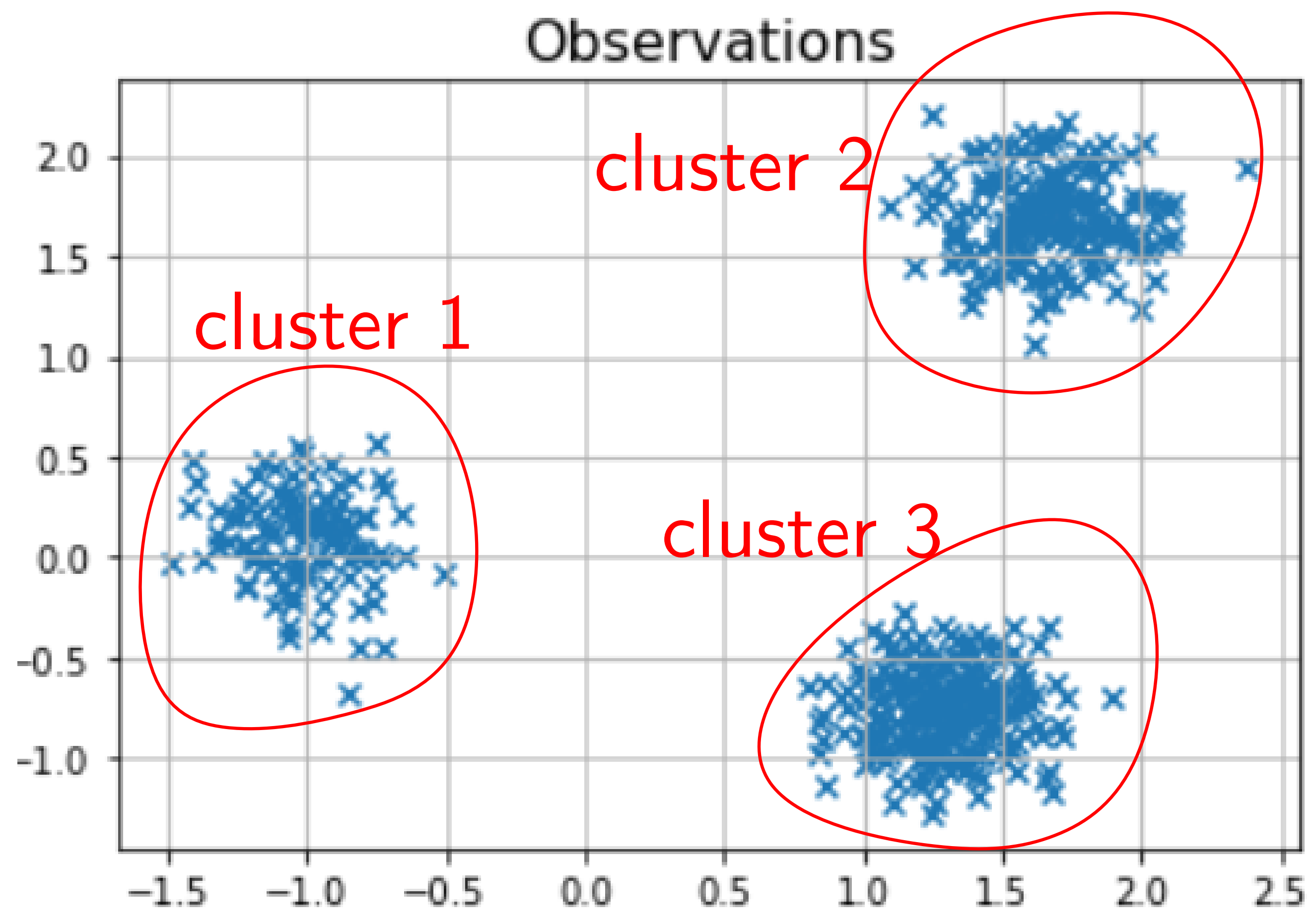
Chapter 5: Unsupervised learning

1.1. The k-means problem.

Idea: Consider a set of observations $\{x_1, \dots, x_n\}$ in $\mathbb{R}^d$, and an integer $k \in \mathbb{N}$.

The main goal of a clustering algorithm is to gather the observations in k groups (clusters) such that:

- Observations belonging to a same cluster should be close to each other,
- Observations belonging to different clusters should be far from each other.

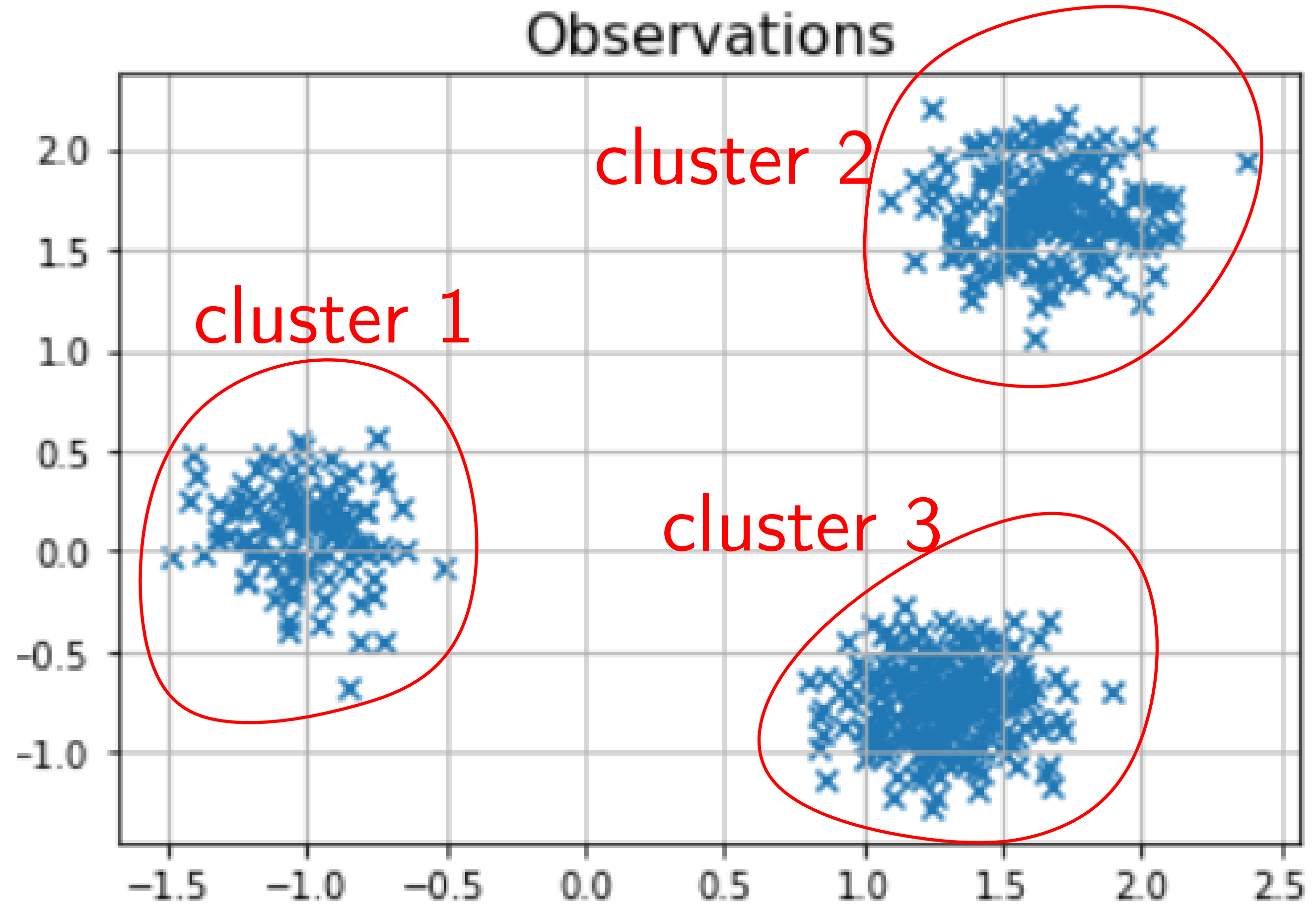


Chapter 5: Unsupervised learning

1.1. The k-means problem.

Idea: Consider a set of observations $\{x_1, \dots, x_n\}$ in $\mathbb{R}^d$, and an integer $k \in \mathbb{N}$.
The main goal of a clustering algorithm is to gather the observations in k groups (clusters) such that:

- Observations belonging to a same cluster should be close to each other,
- Observations belonging to different clusters should be far from each other.



Definition:

The "k-means problem" consists of performing clustering in the following way:
We want to find k-points $c_1, \dots, c_k \in \mathbb{R}^d$ (called centroids) in order to minimize the objective function

$$L(c_1, \dots, c_k) := \sum_{i=1}^{n} \min_{j=1,\dots,k} \|x_i - c_j\|^2. \quad (18)$$

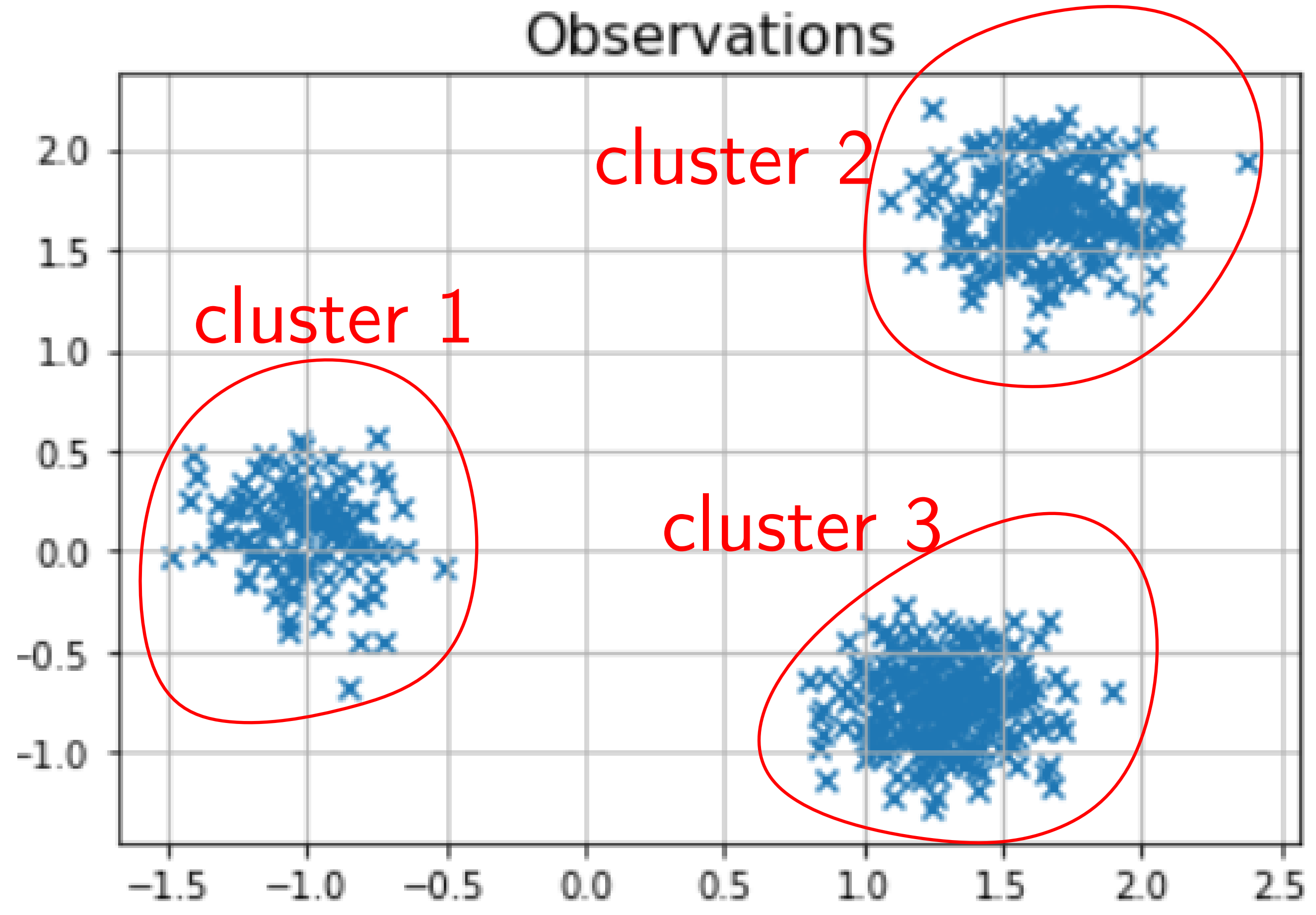
The clusters $\mathcal{C}_1, \dots, \mathcal{C}_k$ are then given, for $j \in \{1, \dots, k\}$ by

$$\mathcal{C}_j := \{i,\ \|x_i - c_j\| \leqslant \|x_i - c_{j'}\|,\ \forall j' \neq j\}. \quad (19)$$

1.1. The k-means problem.

Idea: Consider a set of observations $\{x_1, \ldots, x_n\}$ in $\mathbb{R}^d$, and an integer $k \in \mathbb{N}$.
The main goal of a clustering algorithm is to gather the observations in k groups (clusters) such that:

- Observations belonging to a same cluster should be close to each other,
- Observations belonging to different clusters should be far from each other.



Definition:

The "k-means problem" consists of performing clustering in the following way:
We want to find k-points $c_1, \ldots, c_k \in \mathbb{R}^d$ (called centroids) in order to minimize the objective function

$$L(c_1, \ldots, c_k) := \sum_{i=1}^{n} \min_{j=1,\ldots,k} \|x_i - c_j\|^2. \tag{18}$$

Distance between the observation x_i and its **closest** centroid c_j.

The clusters $\mathcal{C}_1, \ldots, \mathcal{C}_k$ are then given, for $j \in \{1, \ldots, k\}$ by

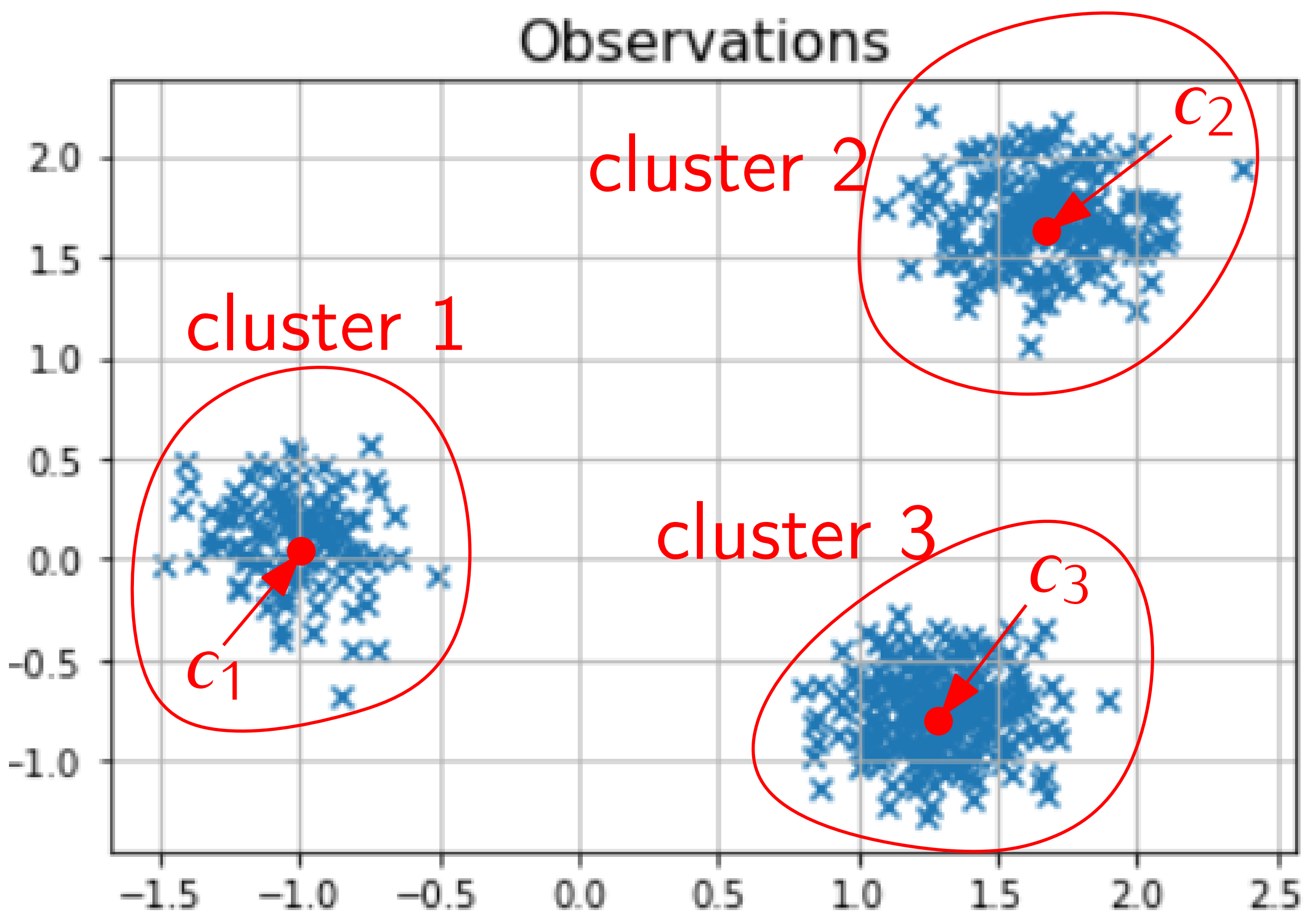
$$\mathcal{C}_j := \{i,\ \|x_i - c_j\| \leqslant \|x_i - c_{j'}\|,\ \forall j' \neq j\}. \tag{19}$$

c_j is the closest centroid with respect to x_i

1.1. The k-means problem.

Idea: Consider a set of observations $\{x_1, \ldots, x_n\}$ in $\mathbb{R}^d$, and an integer $k \in \mathbb{N}$.
The main goal of a clustering algorithm is to gather the observations in k groups (clusters) such that:

- Observations belonging to a same cluster should be close to each other,
- Observations belonging to different clusters should be far from each other.



Definition:

The "k-means problem" consists of performing clustering in the following way:
We want to find k-points $c_1, \ldots, c_k \in \mathbb{R}^d$ (called centroids) in order to minimize the objective function

$$L(c_1, \ldots, c_k) := \sum_{i=1}^{n} \min_{j=1,\ldots,k} \|x_i - c_j\|^2. \tag{18}$$

Distance between the observation x_i and its **closest** centroid c_j.

The clusters $\mathcal{C}_1, \ldots, \mathcal{C}_k$ are then given, for $j \in \{1, \ldots, k\}$ by

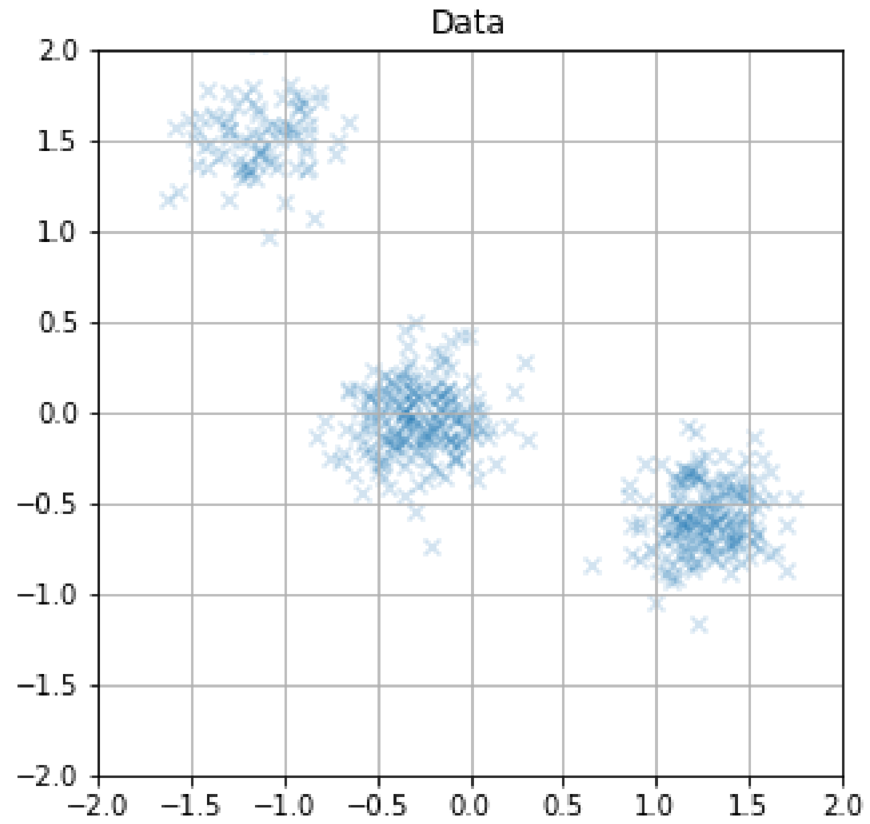
$$\mathcal{C}_j := \{i,\ \|x_i - c_j\| \leqslant \|x_i - c_{j'}\|,\ \forall j' \neq j\}. \tag{19}$$

c_j is the closest centroid with respect to x_i

1.2. Solving the k-means problem: Lloyd algorithm (1957).

1.2. Solving the k-means problem: Lloyd algorithm (1957).

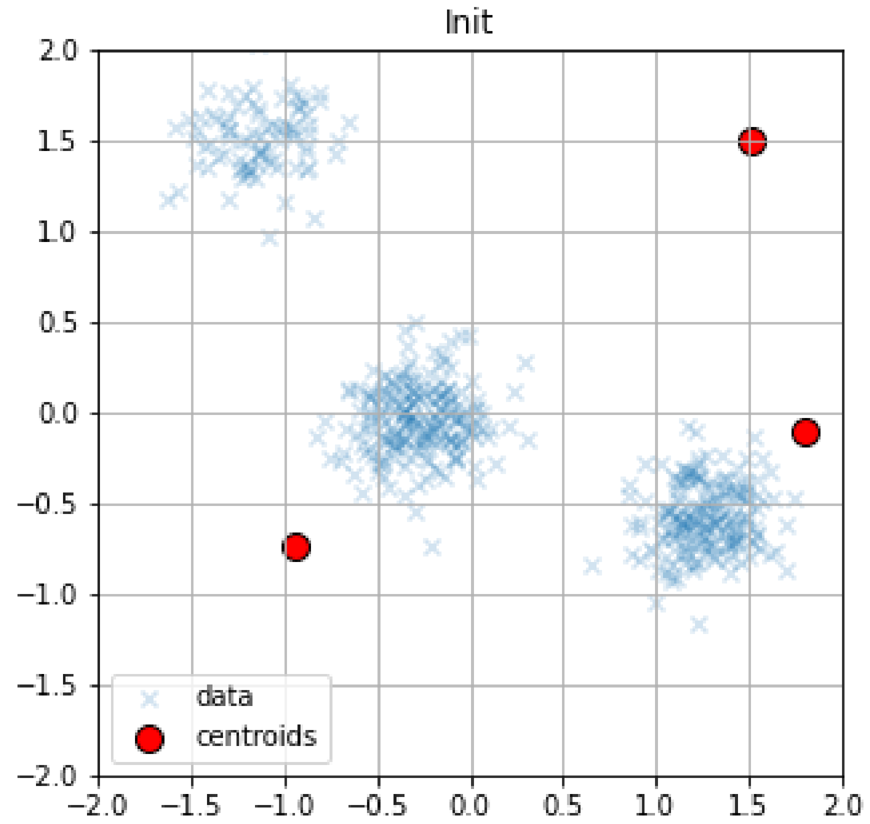
Input: Data $x_1, \ldots, x_n$. Integer k (number of clusters desired).



1.2. Solving the k-means problem: Lloyd algorithm (1957).

Input: Data $x_1, \ldots, x_n$. Integer k (number of clusters desired).

Step 1 - Initialisation: Pick initial positions for the centroids (e.g. randomly) : $c_1, \ldots, c_k$.

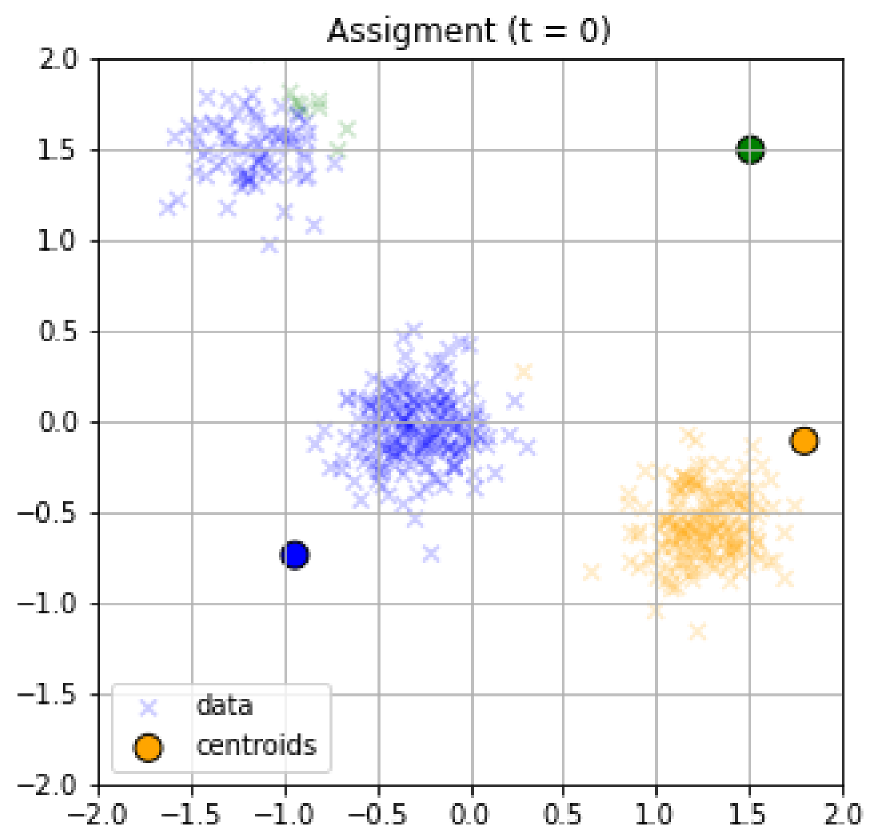


1.2. Solving the k-means problem: Lloyd algorithm (1957).

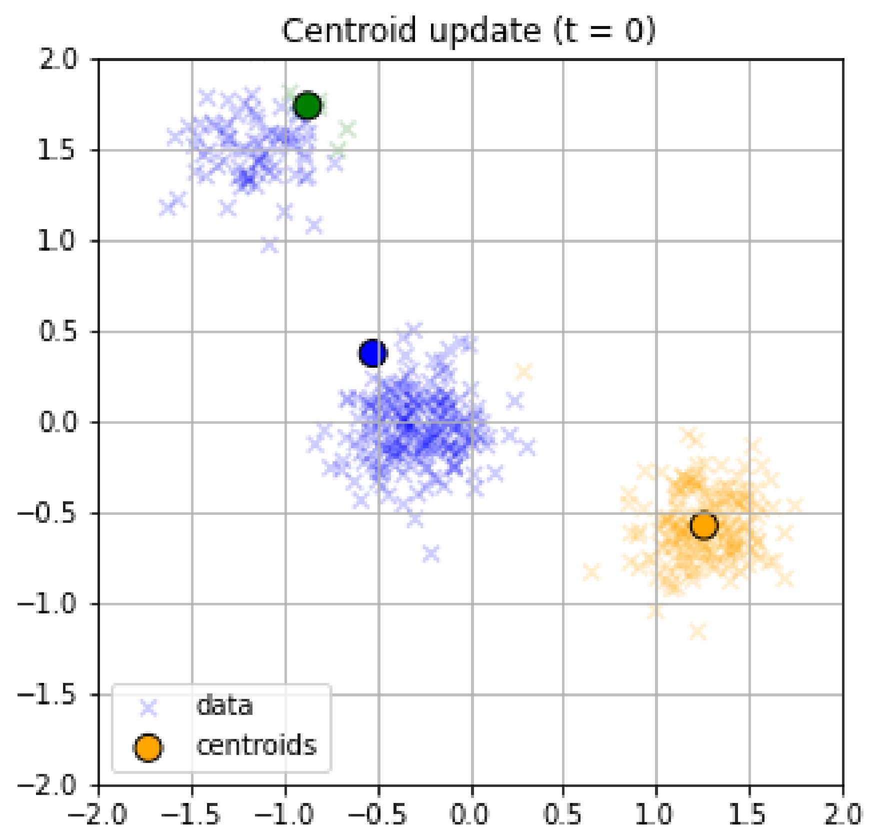
Input: Data $x_1, \ldots, x_n$. Integer k (number of clusters desired).

Step 1 - Initialisation: Pick initial positions for the centroids (e.g. randomly) : $c_1, \ldots, c_k$.

Step 2 - Assignment: For $i \in \{1, \ldots, n\}$ determine the closest centroids c_j to x_i.
Denote it by $x_i \to c_j$.
Store in $\mathcal{C}_j$ all the x_i such that $x_i \to c_j$.



1.2. Solving the k-means problem: Lloyd algorithm (1957).



Input: Data $x_1, \dots, x_n$. Integer k (number of clusters desired).

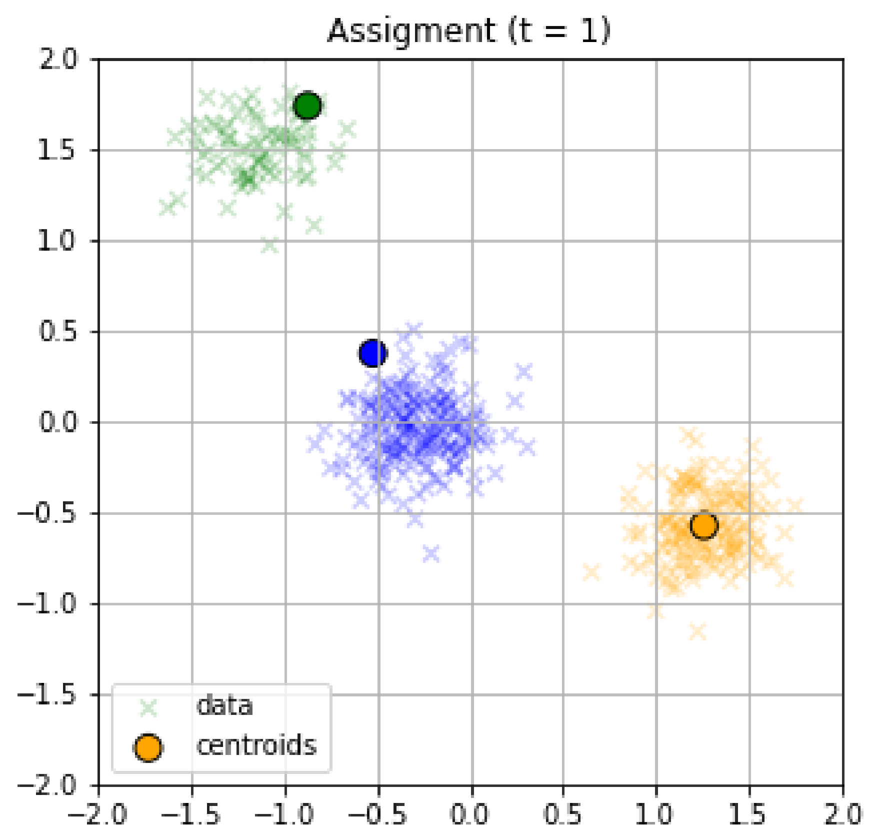
Step 1 - Initialisation: Pick initial positions for the centroids (e.g. randomly) : $c_1, \dots, c_k$.

Step 2 - Assignment: For $i \in \{1, \dots, n\}$ determine the closest centroids c_j to x_i.
Denote it by $x_i \to c_j$.
Store in $\mathcal{C}_j$ all the x_i such that $x_i \to c_j$.

Step 3 - Update centroids: For $j = 1, \dots, k$, do :

$$c_j \leftarrow \frac{1}{\#\mathcal{C}_j} \sum_{i \in \mathcal{C}_j} x_i.$$

1.2. Solving the k-means problem: Lloyd algorithm (1957).



Input: Data $x_1, \dots, x_n$. Integer k (number of clusters desired).

Step 1 - Initialisation: Pick initial positions for the centroids (e.g. randomly) : $c_1, \dots, c_k$.

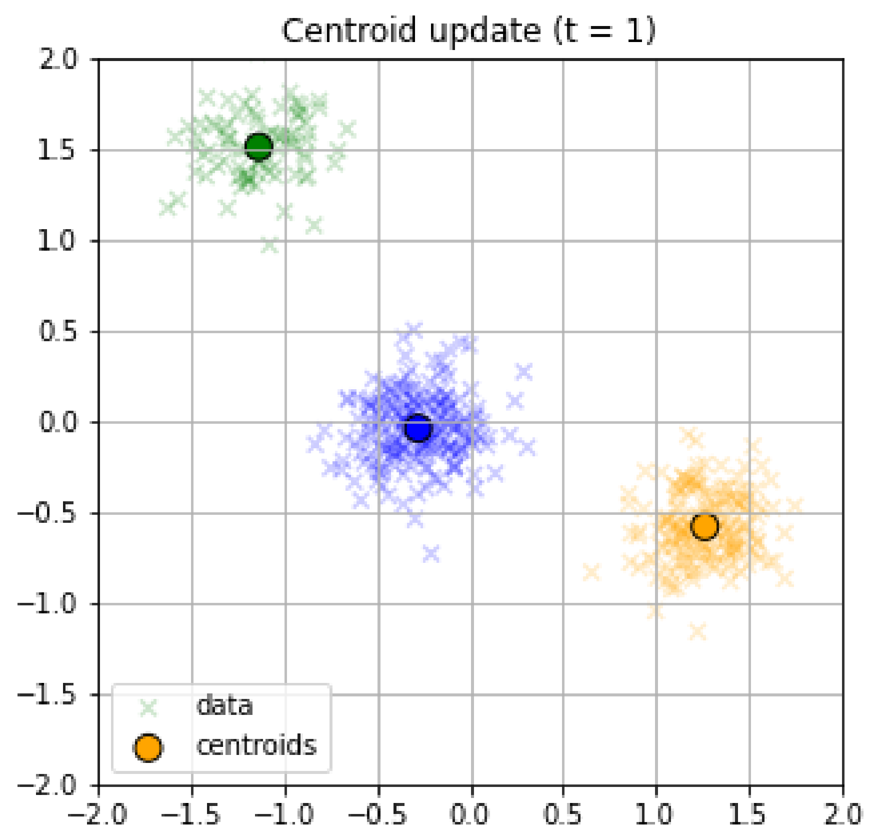
Step 2 - Assignment: For $i \in \{1, \dots, n\}$ determine the closest centroids c_j to x_i.
Denote it by $x_i \to c_j$.
Store in $\mathcal{C}_j$ all the x_i such that $x_i \to c_j$.

Step 3 - Update centroids: For $j = 1, \dots, k$, do :

$$c_j \leftarrow \frac{1}{\#\mathcal{C}_j} \sum_{i \in \mathcal{C}_j} x_i.$$

Step 4 - Iteration: Repeat steps 2 and 3 until convergence, that is when the centroids are no longer moving.

1.2. Solving the k-means problem: Lloyd algorithm (1957).



Input: Data $x_1, \dots, x_n$. Integer k (number of clusters desired).

Step 1 - Initialisation: Pick initial positions for the centroids (e.g. randomly) : $c_1, \dots, c_k$.

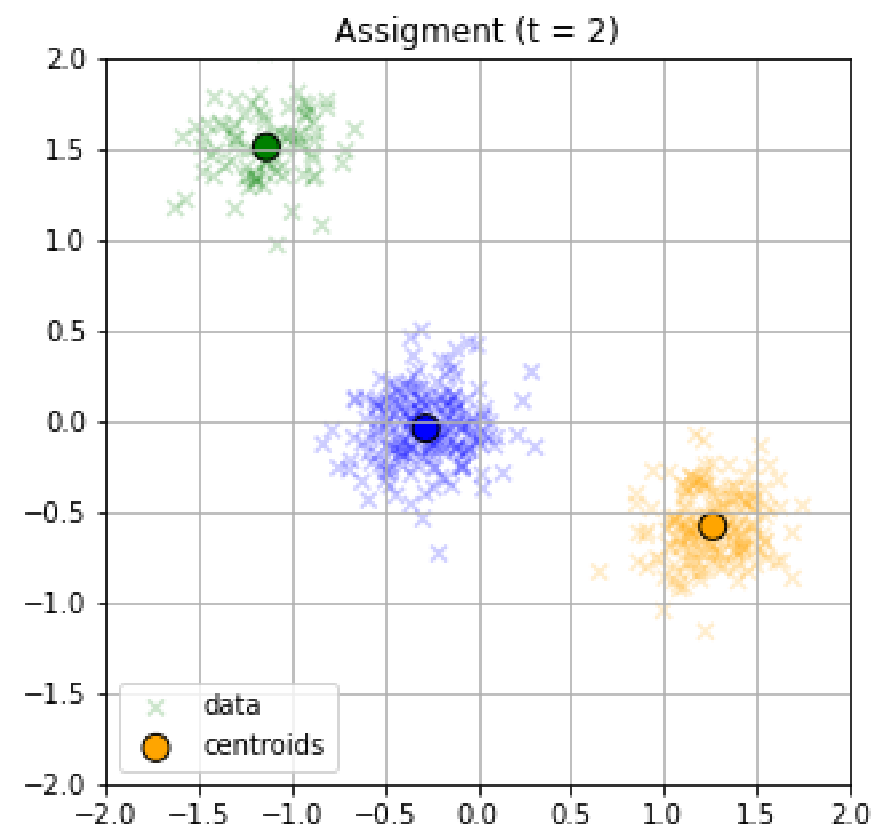
Step 2 - Assignment: For $i \in \{1, \dots, n\}$ determine the closest centroids c_j to x_i.
Denote it by $x_i \to c_j$.
Store in $\mathcal{C}_j$ all the x_i such that $x_i \to c_j$.

Step 3 - Update centroids: For $j = 1, \dots, k$, do :

$$c_j \leftarrow \frac{1}{\#\mathcal{C}_j} \sum_{i \in \mathcal{C}_j} x_i.$$

Step 4 - Iteration: Repeat steps 2 and 3 until convergence, that is when the centroids are no longer moving.

1.2. Solving the k-means problem: Lloyd algorithm (1957).



Input: Data $x_1, \ldots, x_n$. Integer k (number of clusters desired).

Step 1 - Initialisation: Pick initial positions for the centroids (e.g. randomly) : $c_1, \ldots, c_k$.

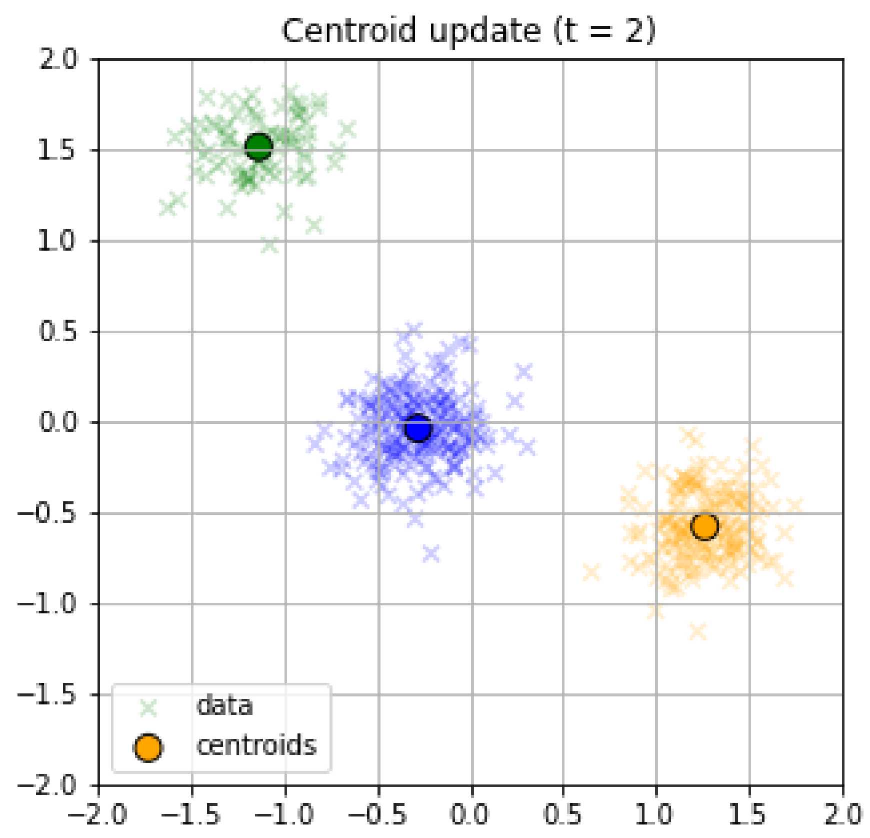
Step 2 - Assignment: For $i \in \{1, \ldots, n\}$ determine the closest centroids c_j to x_i.
Denote it by $x_i \to c_j$.
Store in $\mathcal{C}_j$ all the x_i such that $x_i \to c_j$.

Step 3 - Update centroids: For $j = 1, \ldots, k$, do :

$$c_j \leftarrow \frac{1}{\#\mathcal{C}_j} \sum_{i \in \mathcal{C}_j} x_i.$$

Step 4 - Iteration: Repeat steps 2 and 3 until convergence, that is when the centroids are no longer moving.

1.2. Solving the k-means problem: Lloyd algorithm (1957).



Input: Data $x_1, \ldots, x_n$. Integer k (number of clusters desired).

Step 1 - Initialisation: Pick initial positions for the centroids (e.g. randomly) : $c_1, \ldots, c_k$.

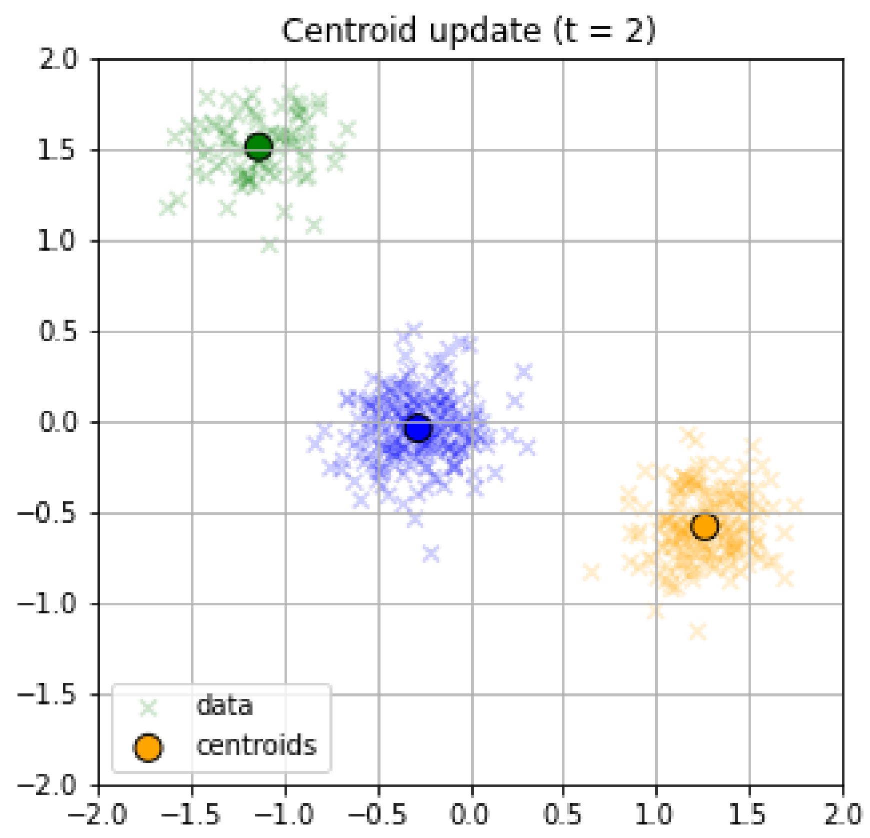
Step 2 - Assignment: For $i \in \{1, \ldots, n\}$ determine the closest centroids c_j to x_i.
Denote it by $x_i \to c_j$.
Store in $\mathcal{C}_j$ all the x_i such that $x_i \to c_j$.

Step 3 - Update centroids: For $j = 1, \ldots, k$, do :

$$c_j \leftarrow \frac{1}{\#\mathcal{C}_j} \sum_{i \in \mathcal{C}_j} x_i.$$

Step 4 - Iteration: Repeat steps 2 and 3 until convergence, that is when the centroids are no longer moving.

1.2. Solving the k-means problem: Lloyd algorithm (1957).



Input: Data $x_1, \dots, x_n$. Integer k (number of clusters desired).

Step 1 - Initialisation: Pick initial positions for the centroids (e.g. randomly) : $c_1, \dots, c_k$.

Step 2 - Assignment: For $i \in \{1, \dots, n\}$ determine the closest centroids c_j to x_i.
Denote it by $x_i \to c_j$.
Store in $\mathcal{C}_j$ all the x_i such that $x_i \to c_j$.

Step 3 - Update centroids: For $j = 1, \dots, k$, do :

$$c_j \leftarrow \frac{1}{\#\mathcal{C}_j} \sum_{i \in \mathcal{C}_j} x_i.$$

Step 4 - Iteration: Repeat steps 2 and 3 until convergence, that is when the centroids are no longer moving.

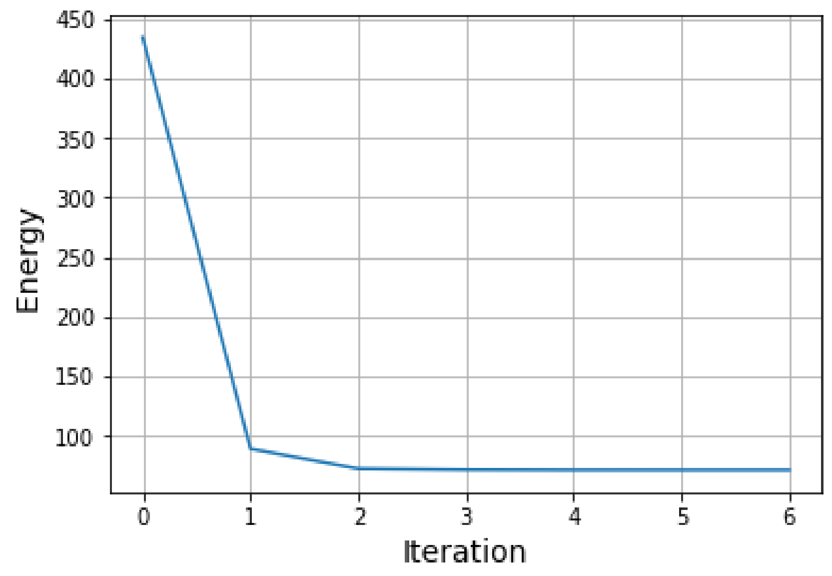
Step 5 - Output: Return the clusters $\mathcal{C}_1, \dots, \mathcal{C}_k$ and the centroids $c_1, \dots, c_k$.

1.2. Solving the k-means problem: Lloyd algorithm (1957).

Proposition:

At each iteration of the Lloyd algorithm, the objective value $L(c_1, \dots, c_k)$ decreases.
Given that $L \geqslant 0$, the objective value **converges**.
In addition, assuming the x_i are in a *generic* configuration, the centroids $(c_j)_{j=1,\dots,k}$ (and thus the clusters $(\mathcal{C}_j)_{j=1,\dots,k}$) converge as well. Therefore, the Lloyd algorithm **converges** toward a **configuration** that is a **local minimizer of the "energy"** of the system.

Exercise: Prove it.

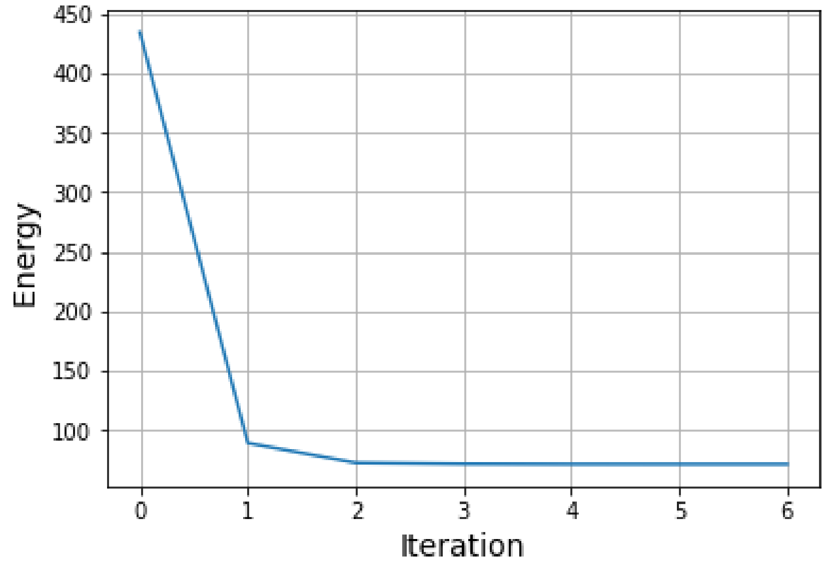


1.2. Solving the k-means problem: Lloyd algorithm (1957).

Proposition:

At each iteration of the Lloyd algorithm, the objective value $L(c_1, \dots, c_k)$ decreases.
Given that $L \geqslant 0$, the objective value **converges**.
In addition, assuming the x_i are in a *generic* configuration, the centroids $(c_j)_{j=1,\dots,k}$ (and thus the clusters $(\mathcal{C}_j)_{j=1,\dots,k}$) converge as well. Therefore, the Lloyd algorithm **converges** toward a **configuration** that is a **local minimizer of the "energy"** of the system.

Exercise: Prove it.



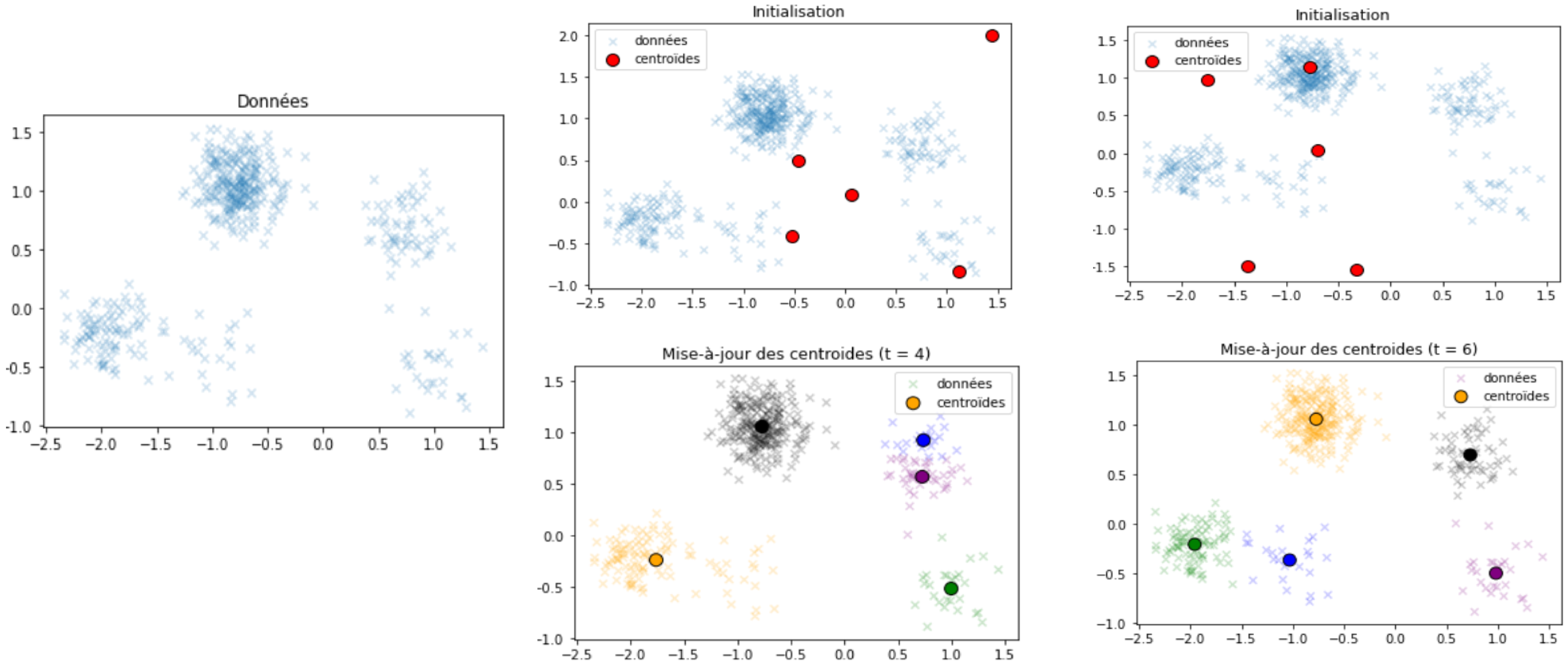
Warning: We just have **local convergence**, that is slightly perturbing the centroids $(c_j)_{j=1}^k$ cannot decrease the objective L.
But there could exist different configuration that would be significantly better.
This is an consequence of the objective function being non-convex.
There is no efficient algorithm (that is, with polynomial complexity) that would be guaranted to converge toward a global minimizer of the k-means problem. The problem is said to be NP-hard.

1.3 - Limitations of k-means / Lloyd.

1.3 - Limitations of k-means / Lloyd.

- The output depends on the initialization. As the objective function in k-means is non-convex, the result we get (and its quality, the running time, etc.) can depend on the initialization (often random) of the algorithm.

→ Trick: try several initialization and start from the best one.



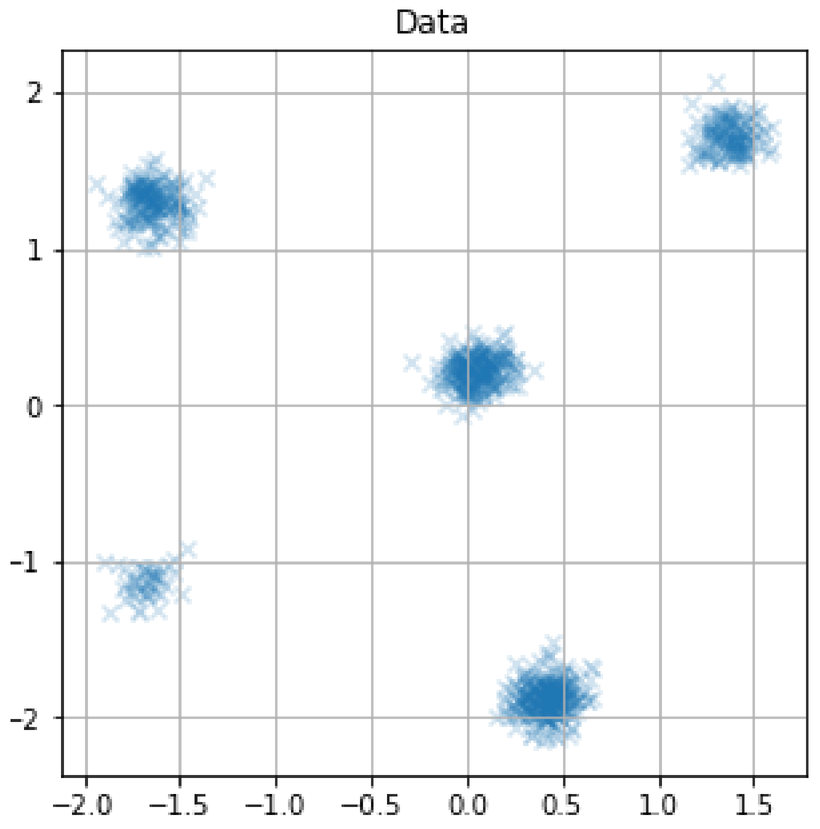
L-final : 0.27

L-final : 0.24

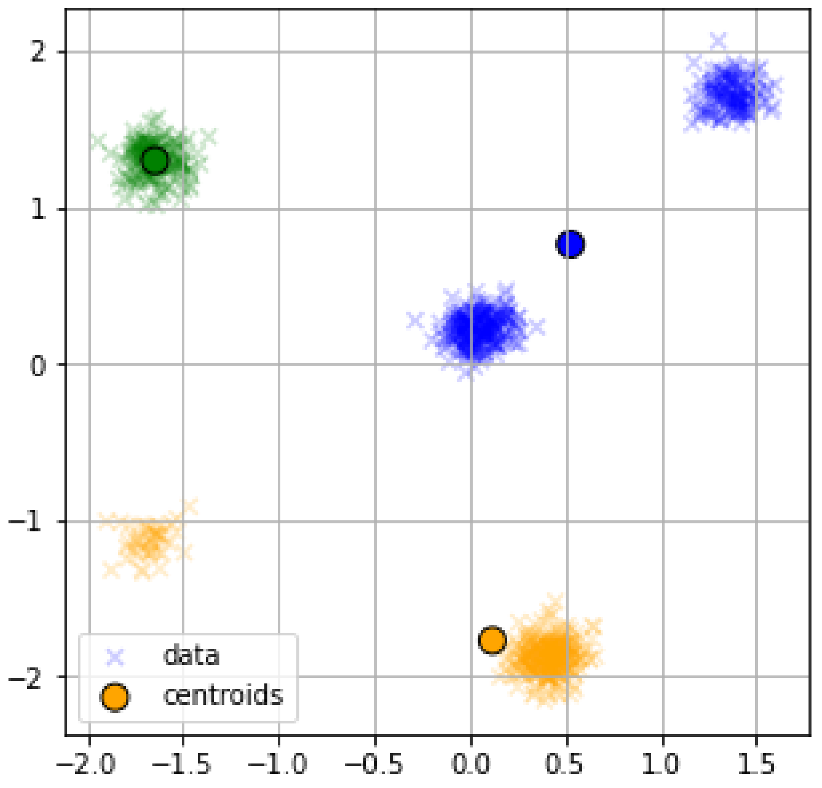
1.3 - Limitations of k-means / Lloyd.

• Picking the number of centroid k. The k-means problem requires to “guess” the correct number of centroids to use. Using too many/few of them yields unsatisfactory results.

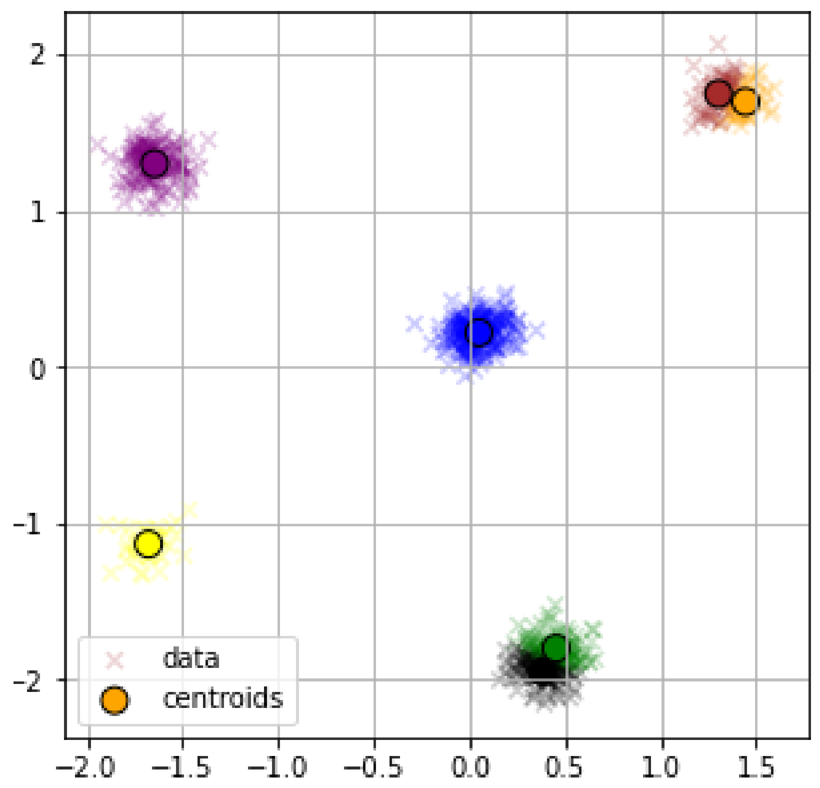
→ “Elbow rule” : try different value for k. When the limit energy stagnates, we may have reach a relevant value for k.



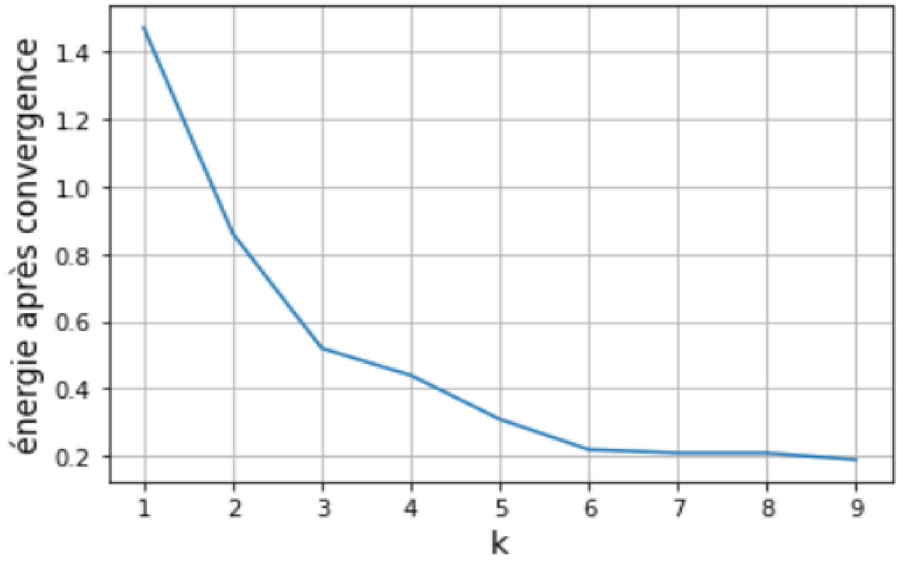
Output of Lloyd’s alg. $k = 3$



Output of Lloyd’s alg. $k = 7$



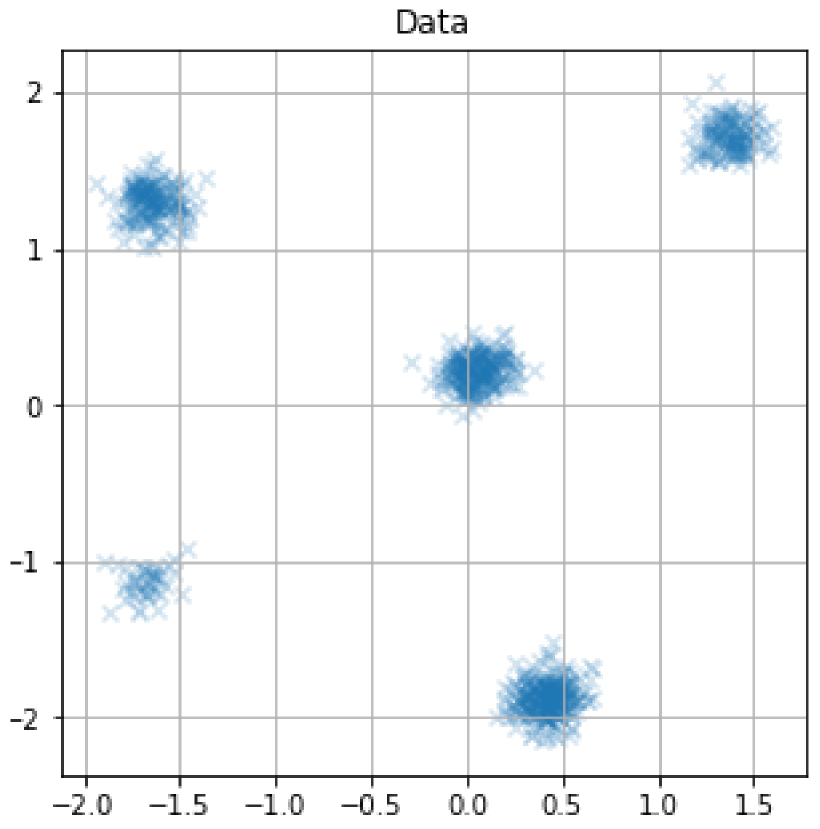
Evolution of final L wrt k.



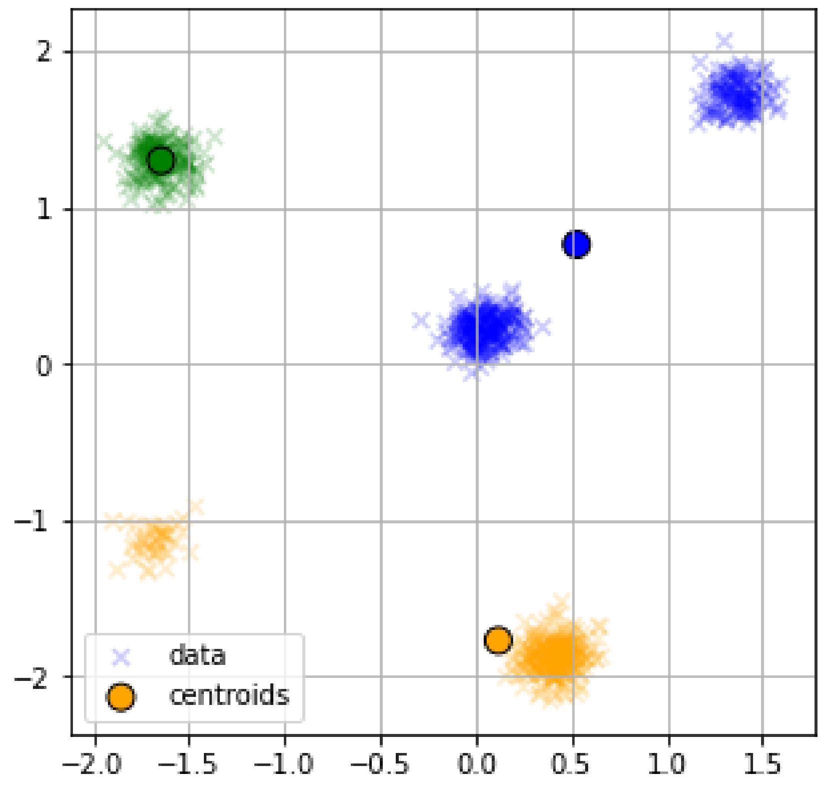
1.3 - Limitations of k-means / Lloyd.

• Picking the number of centroid k. The k-means problem requires to "guess" the correct number of centroids to use. Using too many/few of them yields unsatisfactory results.

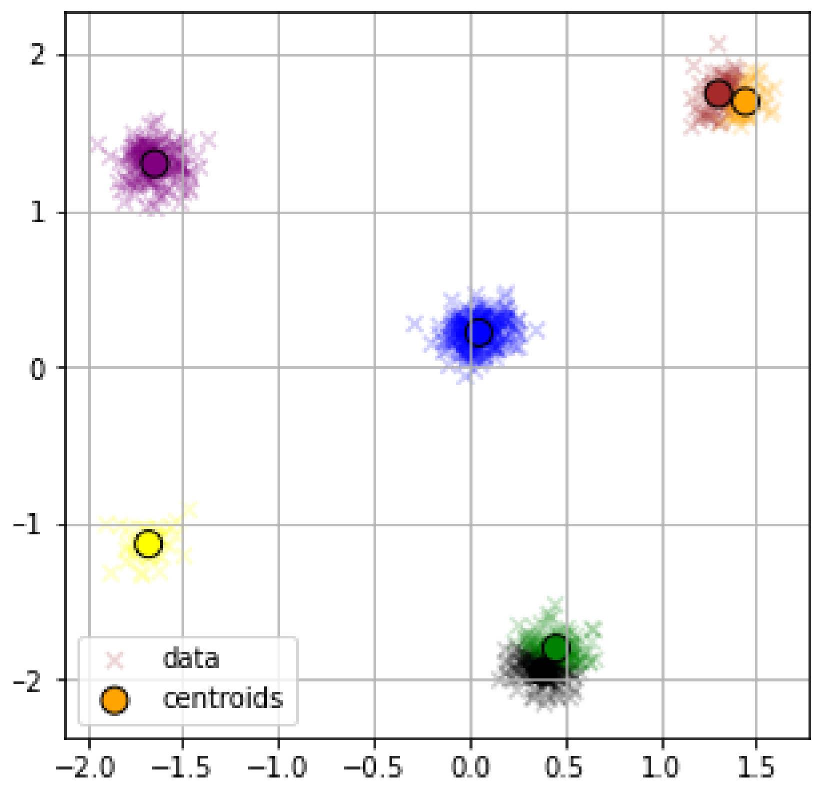
→ "Elbow rule" : try different value for k. When the limit energy stagnates, we may have reach a relevant value for k.



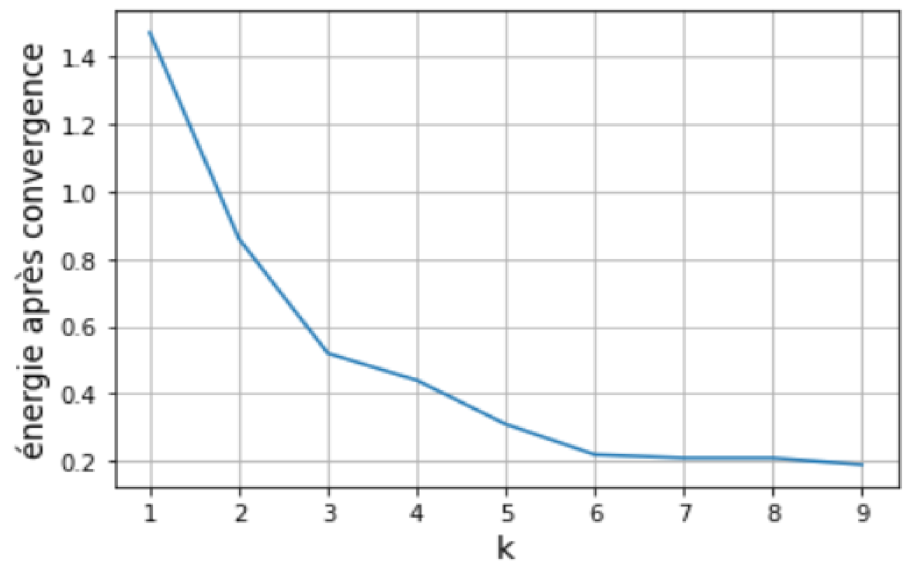
Output of Lloyd's alg. $k = 3$



Output of Lloyd's alg. $k = 7$

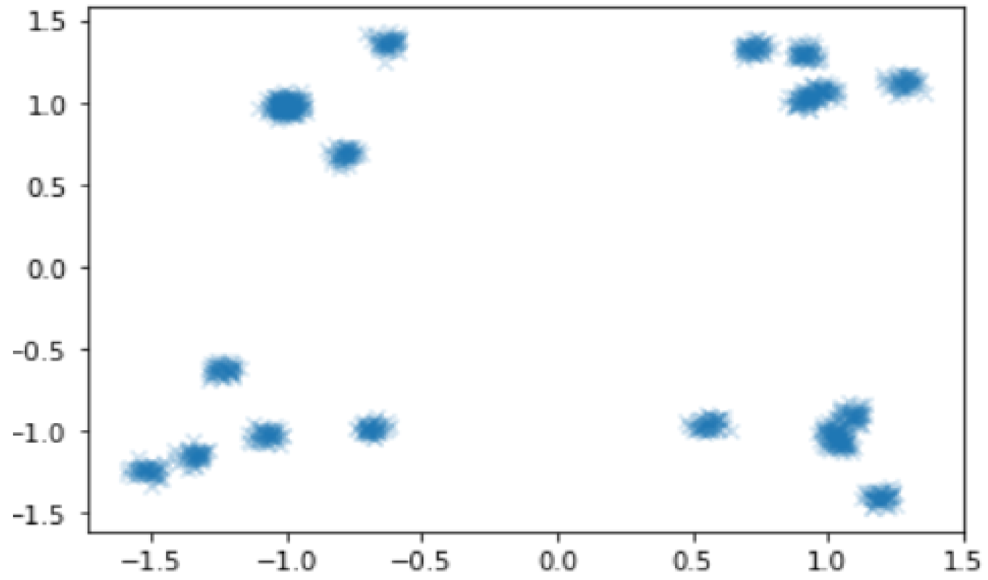


Evolution of final L wrt k.



Warning: Sometimes, even the question of the "number of clusters" is meaningless...

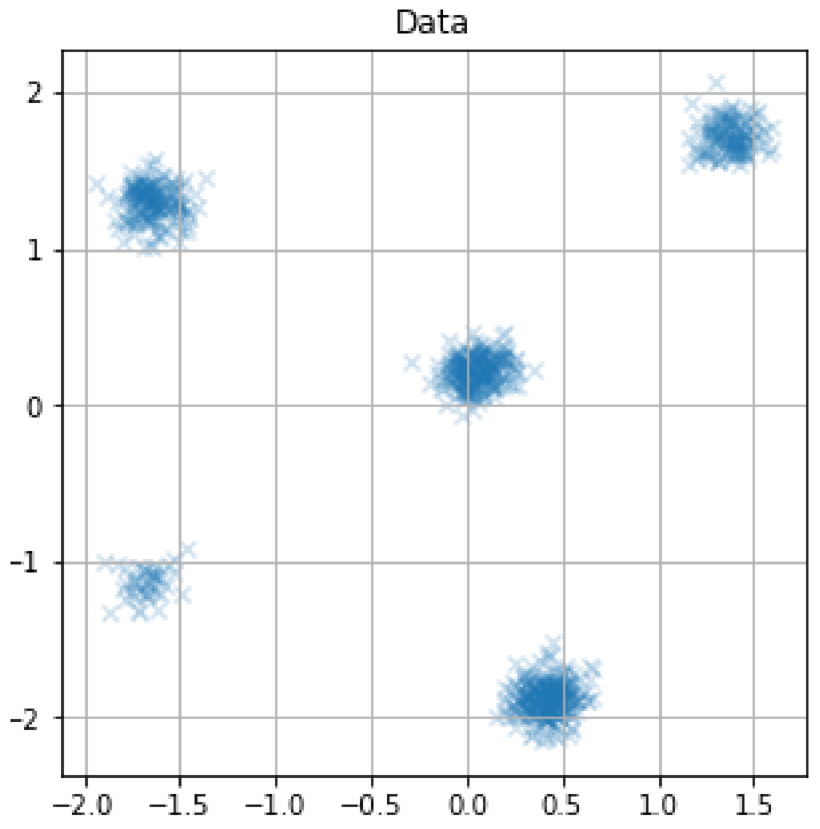
→ multi-scale approaches.



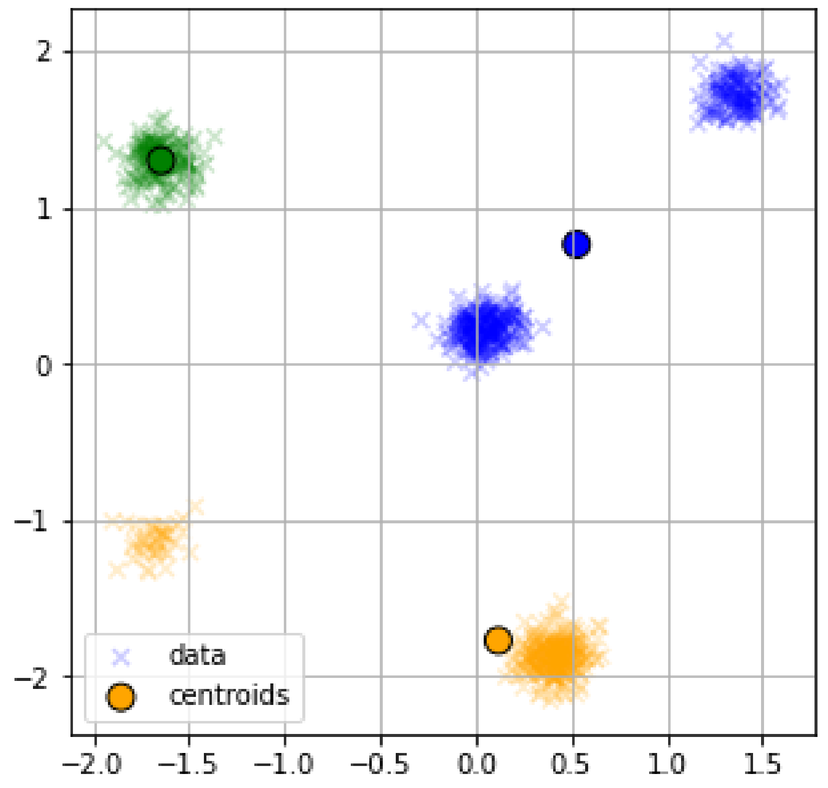
1.3 - Limitations of k-means / Lloyd.

• Picking the number of centroid k. The k-means problem requires to "guess" the correct number of centroids to use. Using too many/few of them yields unsatisfactory results.

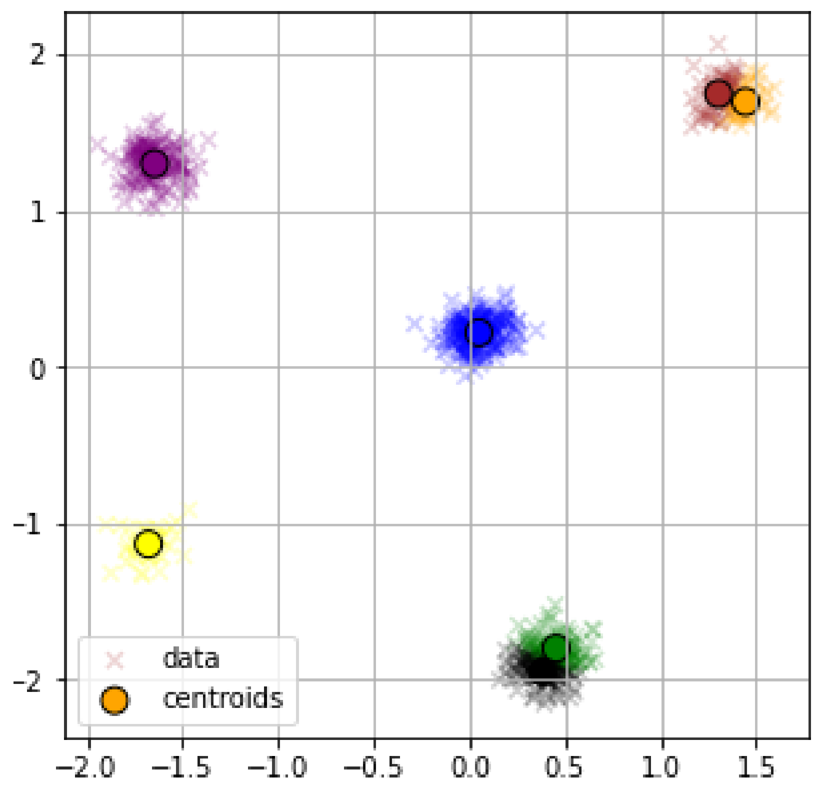
→ "Elbow rule" : try different value for k. When the limit energy stagnates, we may have reach a relevant value for k.



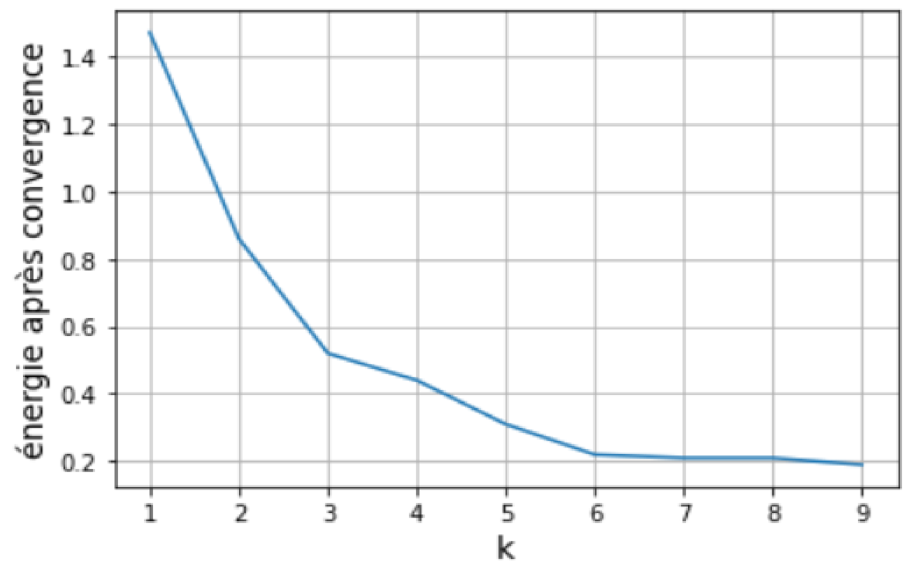
Output of Lloyd's alg. $k = 3$



Output of Lloyd's alg. $k = 7$



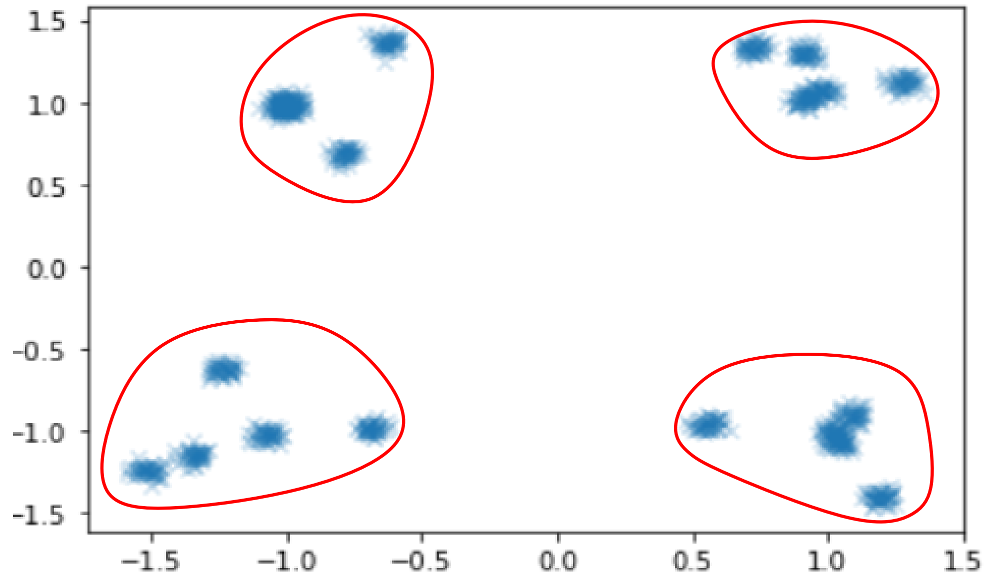
Evolution of final L wrt k.



Warning: Sometimes, even the question of the "number of clusters" is meaningless...

→ multi-scale approaches.

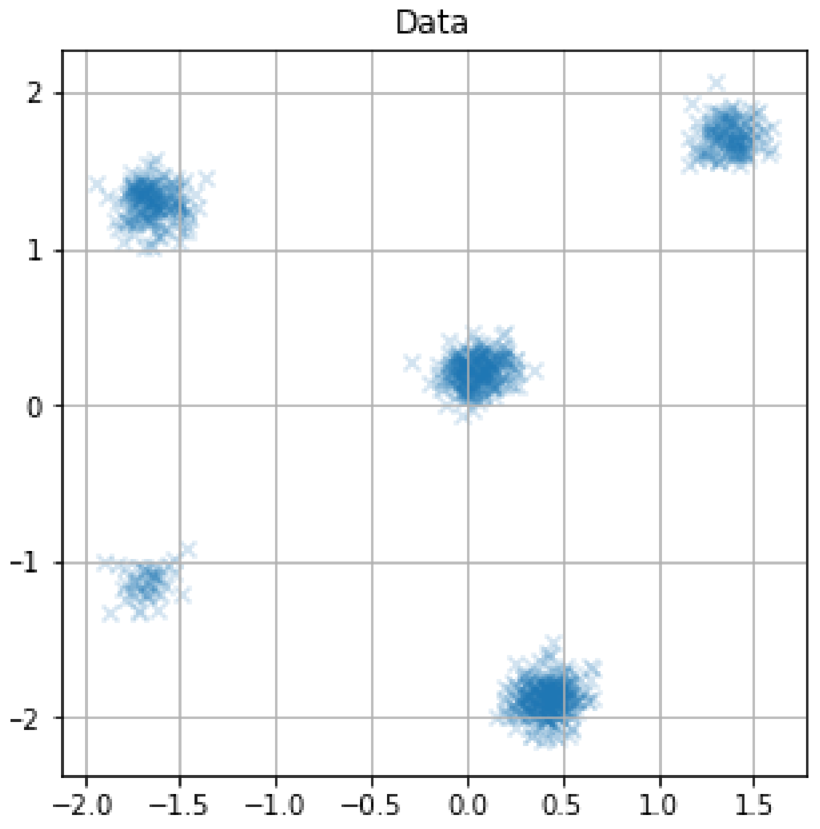
4 clusters ?



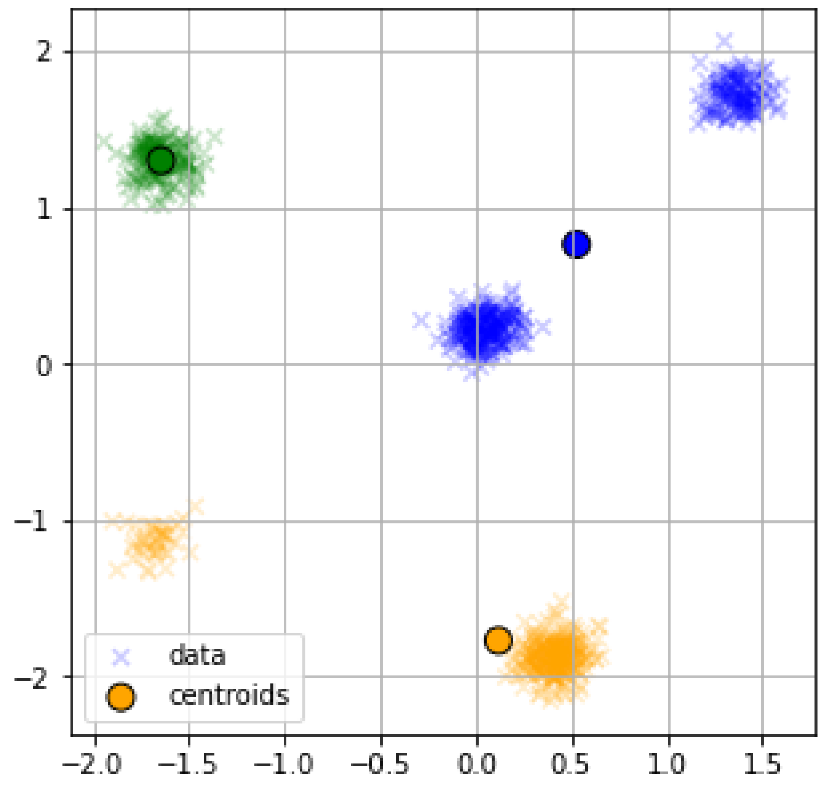
1.3 - Limitations of k-means / Lloyd.

• Picking the number of centroid k. The k-means problem requires to "guess" the correct number of centroids to use. Using too many/few of them yields unsatisfactory results.

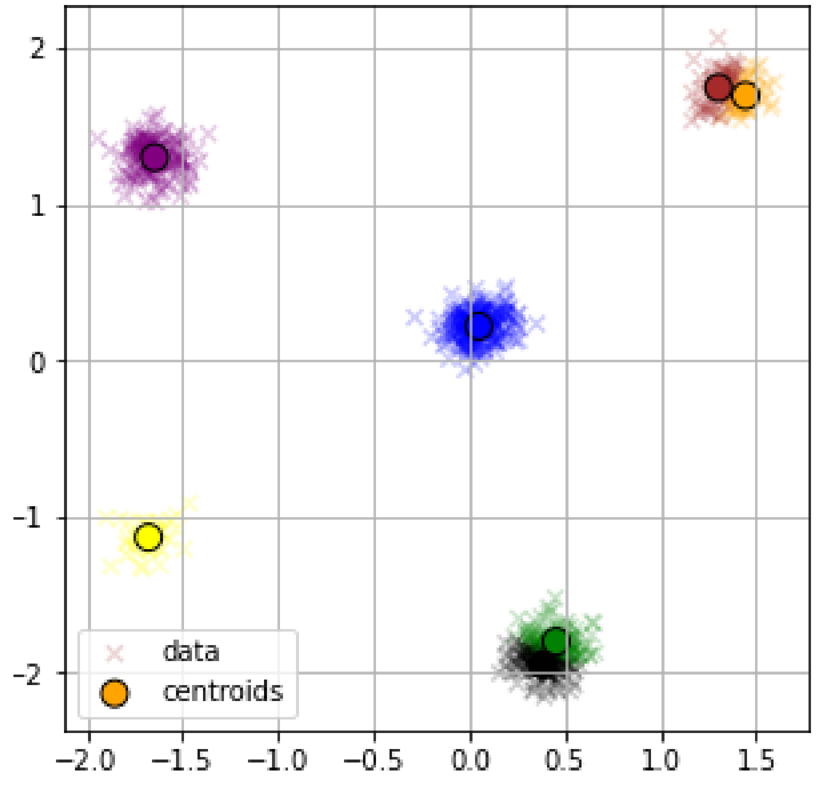
→ "Elbow rule" : try different value for k. When the limit energy stagnates, we may have reach a relevant value for k.



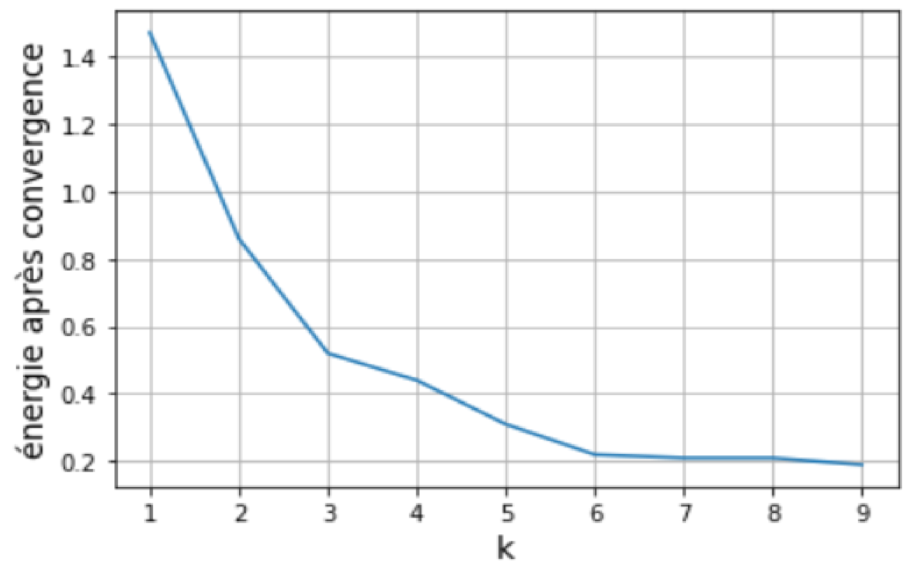
Output of Lloyd's alg. $k = 3$



Output of Lloyd's alg. $k = 7$



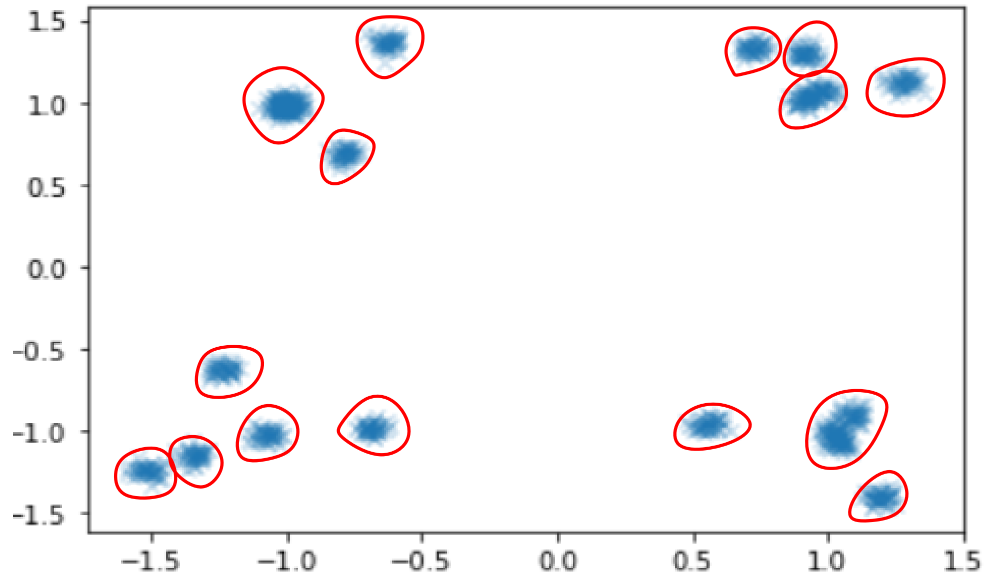
Evolution of final L wrt k.



Warning: Sometimes, even the question of the "number of clusters" is meaningless...

→ multi-scale approaches.

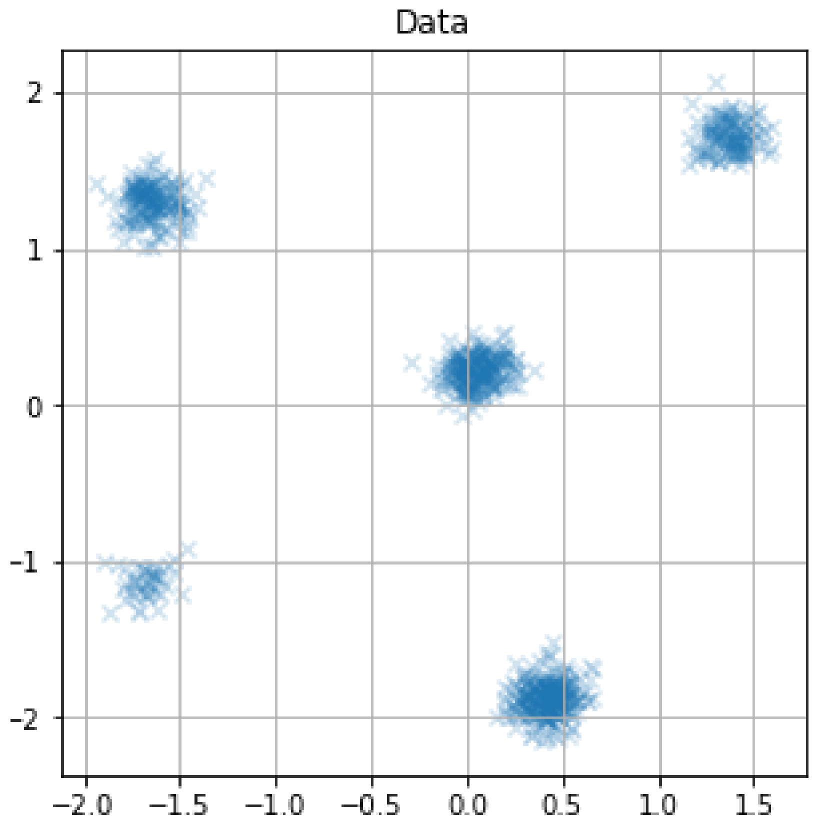
15 clusters ?



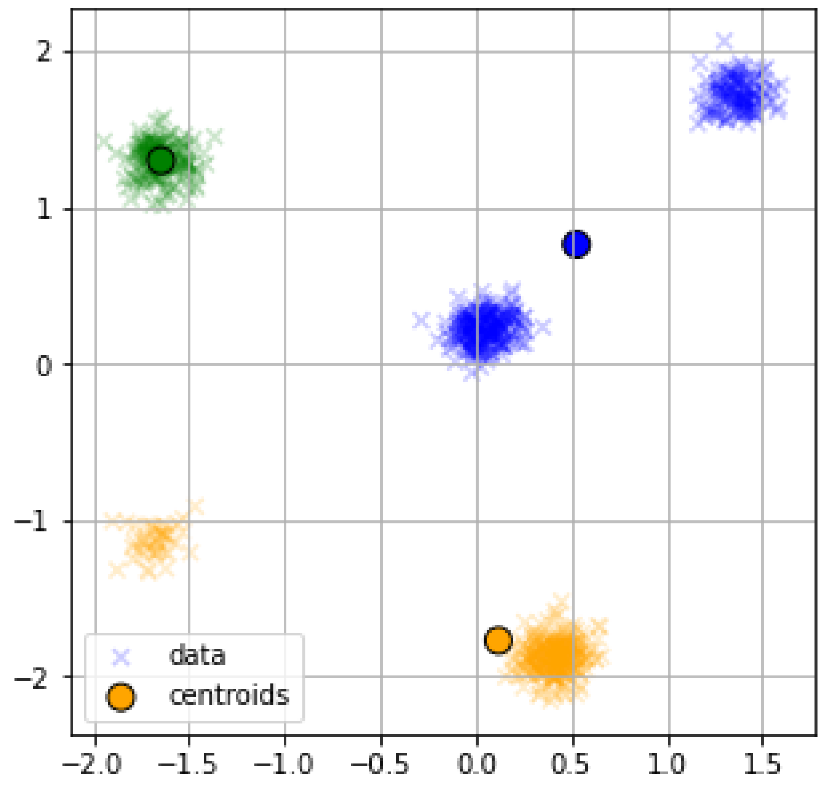
1.3 - Limitations of k-means / Lloyd.

• Picking the number of centroid k. The k-means problem requires to "guess" the correct number of centroids to use. Using too many/few of them yields unsatisfactory results.

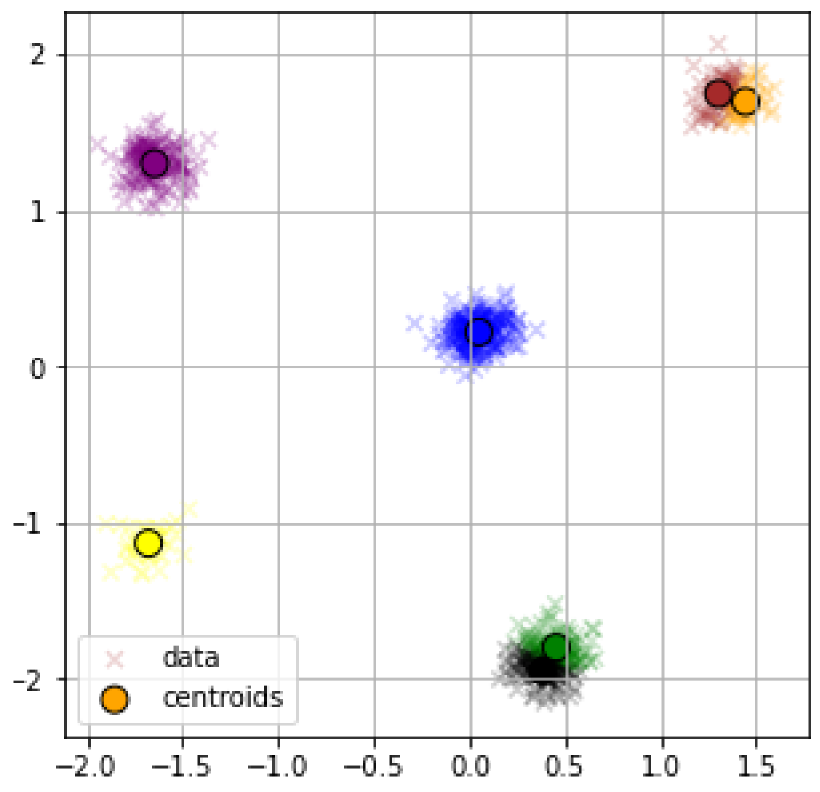
→ "Elbow rule" : try different value for k. When the limit energy stagnates, we may have reach a relevant value for k.



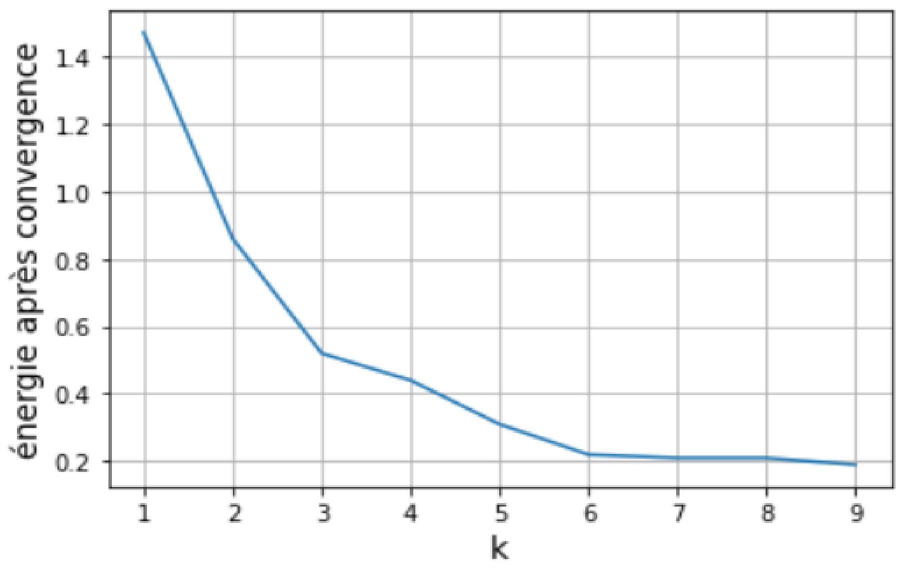
Output of Lloyd's alg. $k = 3$



Output of Lloyd's alg. $k = 7$



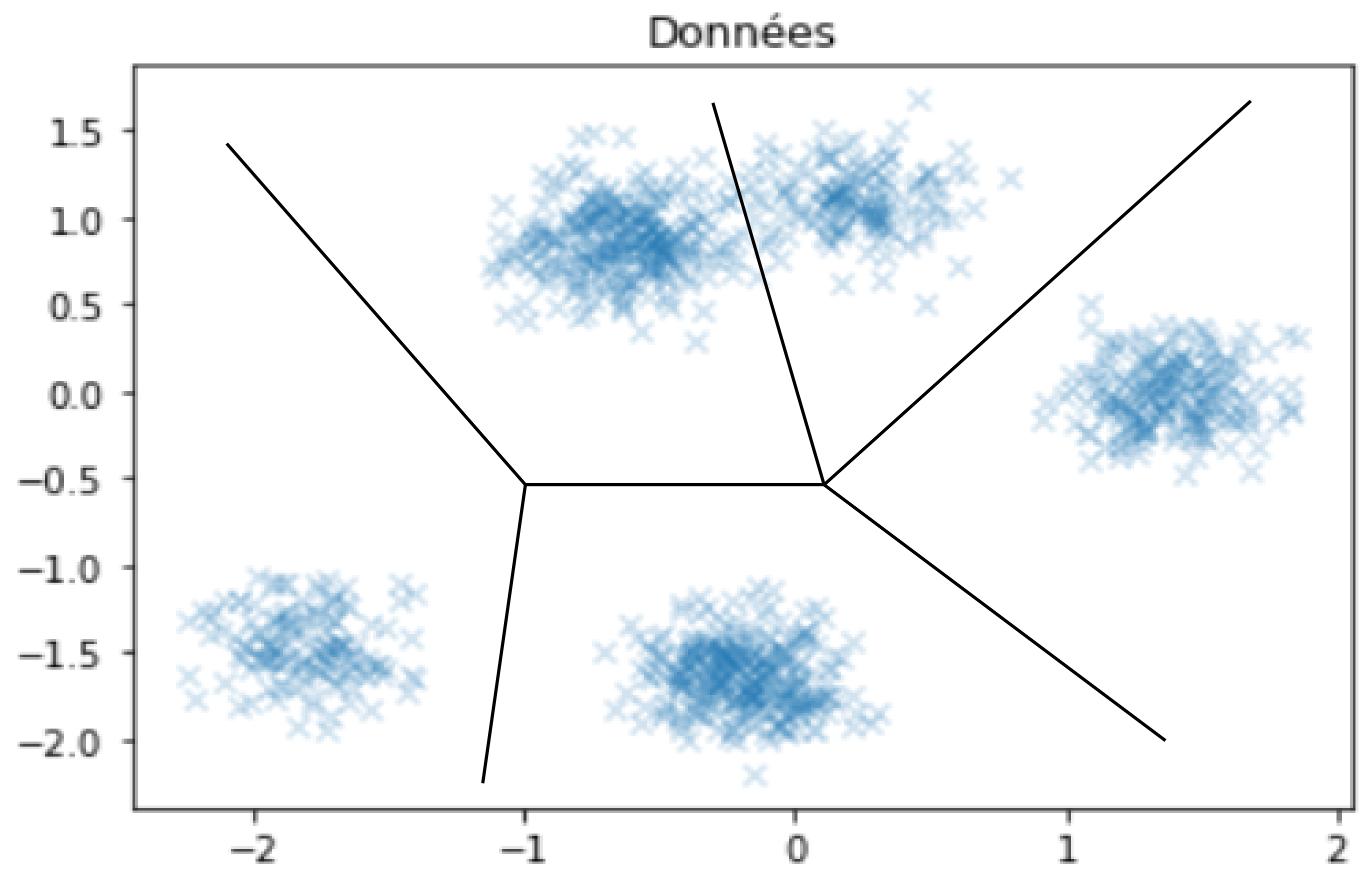
Evolution of final L wrt k.



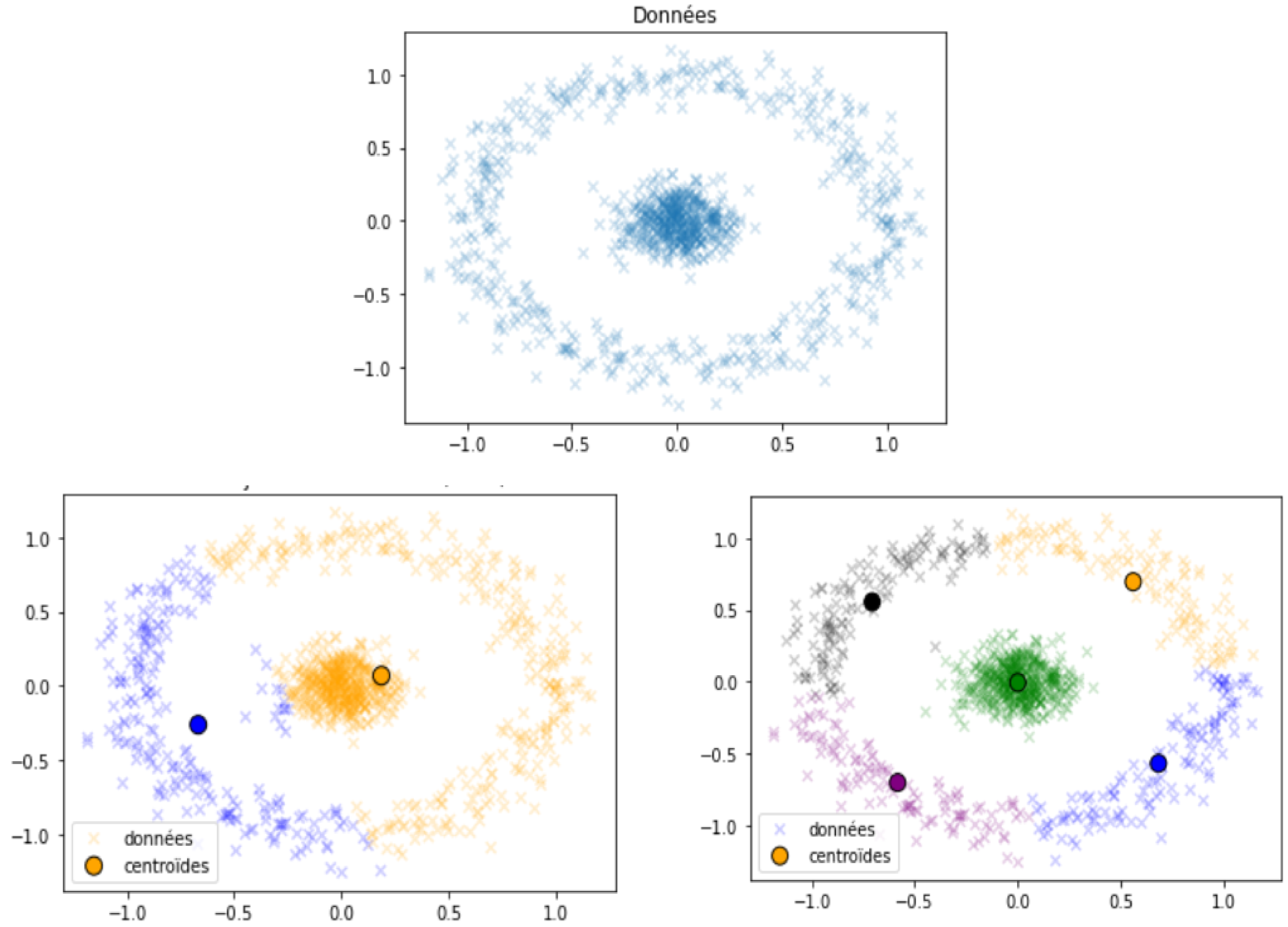
Warning: Here, everything may look easy because we can visualize our data since they are in dimension 2. But in higher dimension, one must be able to understand and interpret the results without being able to visualize the data.

1.3 - Limitations of k-means / Lloyd.

• Linear boundaries: k-means is a linear clustering model, which means (similarly to linear classification models) that it can only perform well on clusters that can be separated by an hyperplane.



Linear separation between the clusters



Dataset without natural linear separatation between the clusters

2. Principal Component Analysis (PCA).

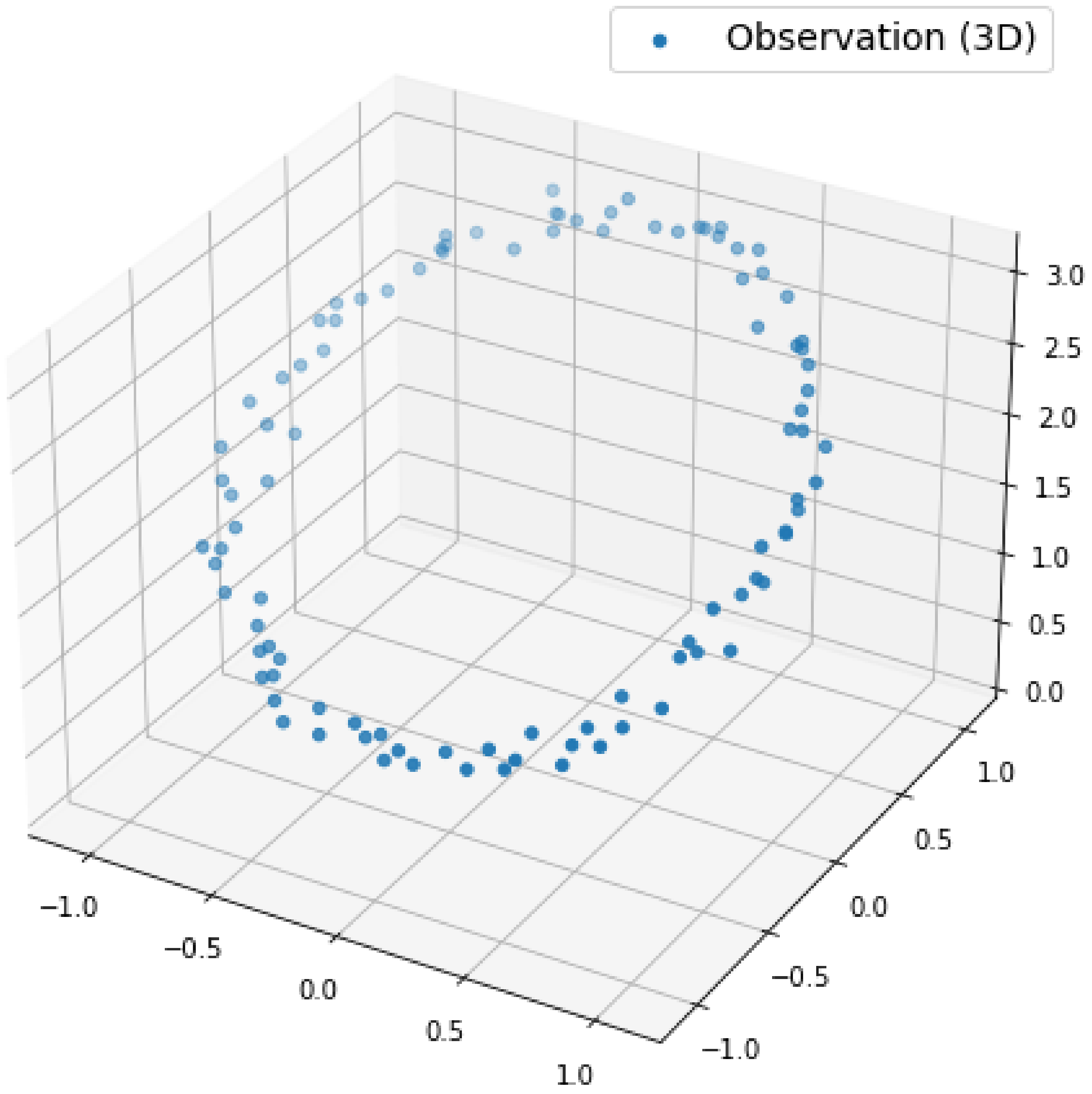
2. Principal Component Analysis (PCA).

This is a method to perform dimensionality reduction. Assume that we are given observations $x_1, \dots, x_n \in \mathbb{R}^D$ with D large. We want to build a point cloud $\hat{x_1}, \dots, \hat{x_n}$ in $\mathbb{R}^d$ with $d \ll D$ that "looks like" the initial observations.

2. Principal Component Analysis (PCA).

This is a method to perform dimensionality reduction. Assume that we are given observations $x_1, \dots, x_n \in \mathbb{R}^D$ with D large. We want to build a point cloud $\hat{x_1}, \dots, \hat{x_n}$ in $\mathbb{R}^d$ with $d \ll D$ that "looks like" the initial observations.

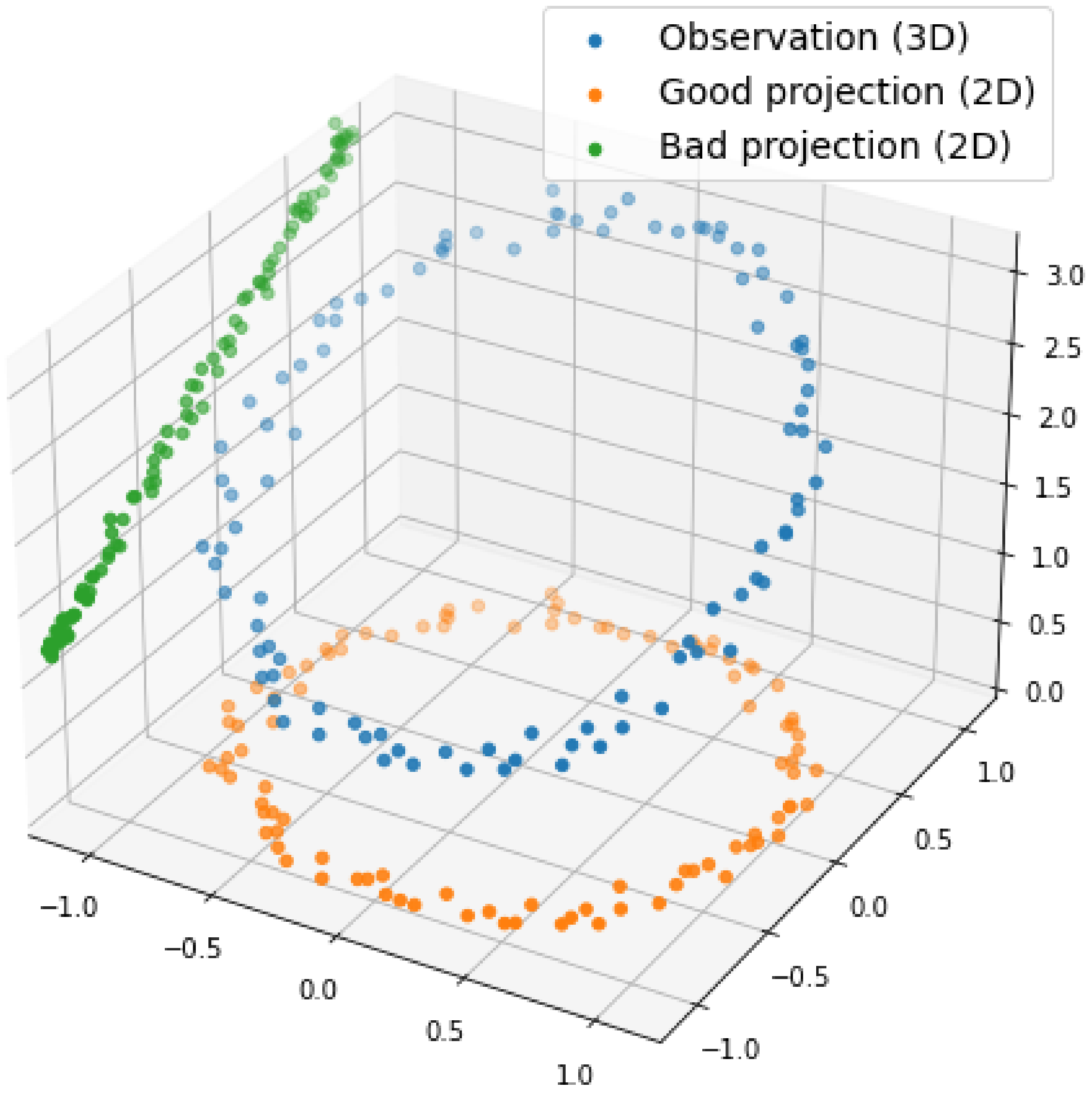
As we are reducing the dimension, we are "compressing" the data. We will necessarily **lose information**, the goal is to lose it **as few as possible**. The information is measured in terms of variance, that should be maximized.



2. Principal Component Analysis (PCA).

This is a method to perform dimensionality reduction. Assume that we are given observations $x_1, \ldots, x_n \in \mathbb{R}^D$ with D large. We want to build a point cloud $\hat{x_1}, \ldots, \hat{x_n}$ in $\mathbb{R}^d$ with $d \ll D$ that "looks like" the initial observations.

As we are reducing the dimension, we are "compressing" the data. We will necessarily **lose information**, the goal is to lose it **as few as possible**. The information is measured in terms of variance, that should be maximized.

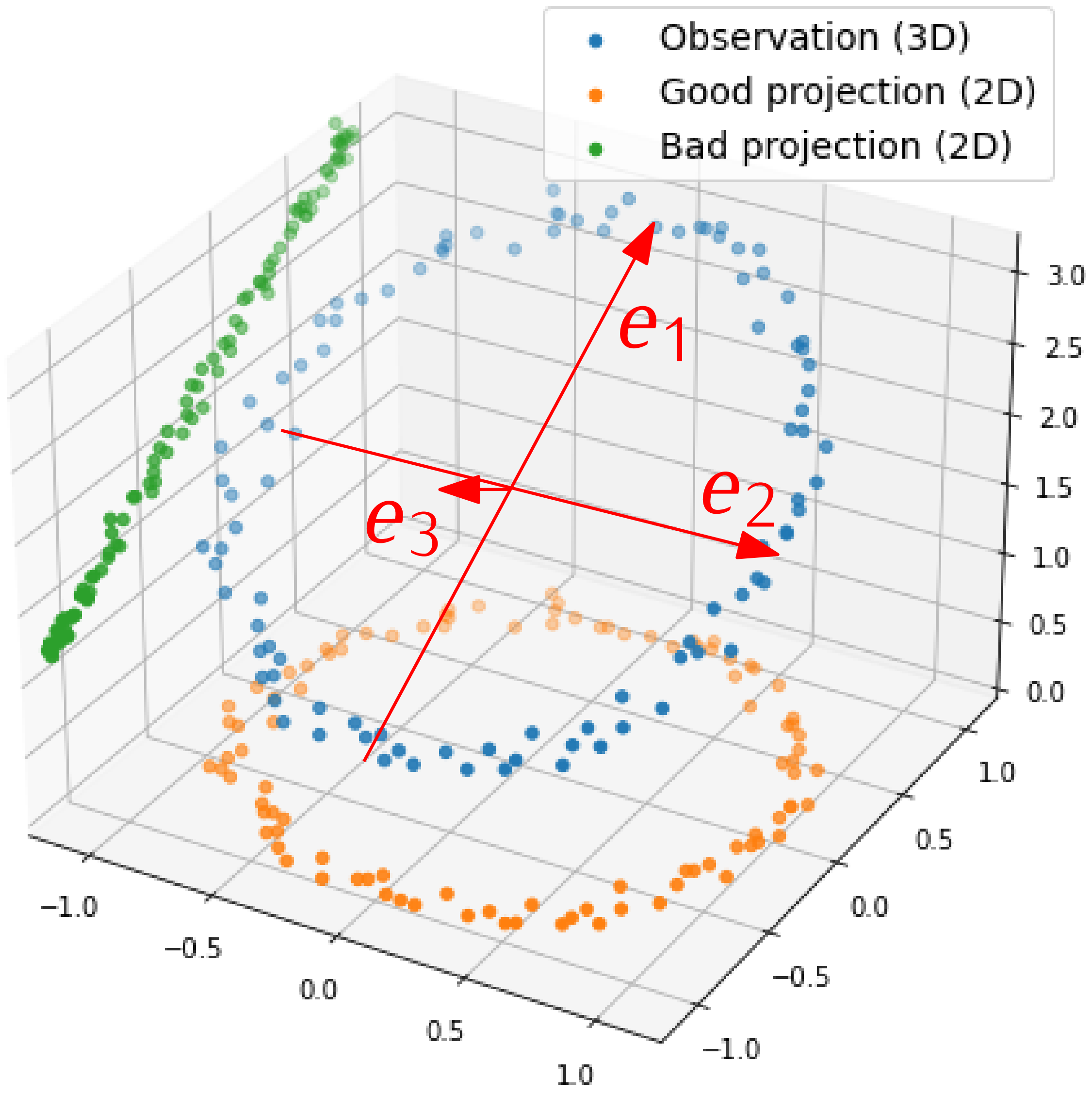


Chapter 5: Unsupervised learning

2. Principal Component Analysis (PCA).

This is a method to perform dimensionality reduction. Assume that we are given observations $x_1, \ldots, x_n \in \mathbb{R}^D$ with D large. We want to build a point cloud $\hat{x_1}, \ldots, \hat{x_n}$ in $\mathbb{R}^d$ with $d \ll D$ that "looks like" the initial observations.

As we are reducing the dimension, we are "compressing" the data. We will necessarily **lose information**, the goal is to lose it **as few as possible**. The information is measured in terms of variance, that should be maximized.



Main idea: We will identify directions ("principal components") $e_1, e_2, \ldots, e_D$ (a basis of $\mathbb{R}^D$) such that:

- e_1 is the direction in which the variance of your set of observations is maximal, e_2 is the second highest, and so on.
- For each direction, we have access to a value $\lambda_1 \geqslant \lambda_2 \geqslant \ldots \lambda_D$ which roughly indicates how large the variance is in that direction.
- We pick the d first directions and we **project** our observation on $(e_1, \ldots, e_d)$.

2. Principal Component Analysis (PCA).

- A formal analysis of the PCA.

Let $X \in \mathbb{R}^{n\times D}$ denote a dataset of n observations in dimension D. Assume that X is centered, that is $1_n X = 0$, where $1_n = (1, \dots, 1) \in \mathbb{R}^n$. This can be obtained by translating the dataset by $-\frac{1}{n}\sum_{j=1}^n x_j$.

2. Principal Component Analysis (PCA).

- A formal analysis of the PCA.

Let $X \in \mathbb{R}^{n \times D}$ denote a dataset of n observations in dimension D. Assume that X is centered, that is $1_n X = 0$, where $1_n = (1, \ldots, 1) \in \mathbb{R}^n$. This can be obtained by translating the dataset by $-\frac{1}{n}\sum_{j=1}^{n} x_j$.

Definition:

The covariance of X is the matrix $C = X^T X \in \mathbb{R}^{D \times D}$.

Interpretation: This $D \times D$ matrix indicates the similarity between the *features* (the D coordinates of the points in X).

2. Principal Component Analysis (PCA).

- A formal analysis of the PCA.

Let $X \in \mathbb{R}^{n \times D}$ denote a dataset of n observations in dimension D. Assume that X is centered, that is $1_n X = 0$, where $1_n = (1, \dots, 1) \in \mathbb{R}^n$. This can be obtained by translating the dataset by $-\frac{1}{n}\sum_{j=1}^n x_j$.

Definition:

The covariance of X is the matrix $C = X^T X \in \mathbb{R}^{D \times D}$.

Interpretation: This $D \times D$ matrix indicates the similarity between the *features* (the D coordinates of the points in X).

Question / Exercise: What is the direction u (i.e. a unit vector in $\mathbb{R}^D$) that would maximize the variance of the **projection** of X along u, that is : which $u \in S^D$ maximizes $u \mapsto (Xu)^T X u$?

2. Principal Component Analysis (PCA).

• A formal analysis of the PCA.

Let $X \in \mathbb{R}^{n\times D}$ denote a dataset of n observations in dimension D. Assume that X is centered, that is $1_n X = 0$, where $1_n = (1, \dots, 1) \in \mathbb{R}^n$. This can be obtained by translating the dataset by $-\frac{1}{n}\sum_{j=1}^n x_j$.

Definition:

The covariance of X is the matrix $C = X^T X \in \mathbb{R}^{D\times D}$.

Interpretation: This $D \times D$ matrix indicates the similarity between the *features* (the D coordinates of the points in X).

Observation: It is symmetric (and real-valued), hence diagonalisable in an orthonormal basis. Let $\lambda_1 \geqslant \lambda_2 \geqslant \dots \geqslant \lambda_D \geqslant 0$ denote its eigenvalues (in decreasing order), and let Q be the transition matrix, that is $C = Q^T \Delta Q$ with $\Delta = \mathrm{diag}(\lambda_1, \dots, \lambda_D)$.

Because C is positive semi-definite

2. Principal Component Analysis (PCA).

• A formal analysis of the PCA.

Let $X \in \mathbb{R}^{n \times D}$ denote a dataset of n observations in dimension D. Assume that X is centered, that is $1_n X = 0$, where $1_n = (1, \dots, 1) \in \mathbb{R}^n$. This can be obtained by translating the dataset by $-\frac{1}{n}\sum_{j=1}^n x_j$.

Definition:

The covariance of X is the matrix $C = X^T X \in \mathbb{R}^{D \times D}$.

Interpretation: This $D \times D$ matrix indicates the similarity between the *features* (the D coordinates of the points in X).

Observation: It is symmetric (and real-valued), hence diagonalisable in an orthonormal basis. Let $\lambda_1 \geqslant \lambda_2 \geqslant \dots \geqslant \lambda_D \geqslant 0$ denote its eigenvalues (in decreasing order), and let Q be the transition matrix, that is $C = Q^T \Delta Q$ with $\Delta = \mathrm{diag}(\lambda_1, \dots, \lambda_D)$.

Because C is positive semi-definite

• Now, let $u \in \mathbb{R}^D$ be a unit vector and Xu be the projection of X in that direction. Asking that Xu has the largest possible variance means that

$$(Qu)^T \Delta (Qu)$$

should be maximized, which tells us that $Qu = (1, 0, \dots, 0)^T \in \mathbb{R}^D$, that is u is the first column of Q^T, i.e. the (unit) eigenvector e_1 associated to λ_1. Now, the second best direction (orthogonal to e_1) is the second column of Q, etc.

2. Principal Component Analysis (PCA).

This is a method to perform dimensionality reduction. Assume that we are given observations $x_1, \ldots, x_n \in \mathbb{R}^D$ with D large. We want to build a point cloud $\hat{x_1}, \ldots, \hat{x_n}$ in $\mathbb{R}^d$ with $d \ll D$ that "looks like" the initial observations.

In practice: We use the class `PCA()` of `sklearn.decomposition`.
We can specify (among other things):

- `n_components`, the number of dimension ("components") that we want to keep. If set to `None`, all components are kept (and we can select them afterwards),

then we retrieve (after running the method `.fit(X)`) the methods

- `.transform(X)` that applies the dimensionality reduction (projection) to the observations `X`.
- `.components_` that returns the (eigen)vectors that indicate the principal components.
- `.explained_variance_ratio_`, that indicate the contribution (in percentage) of each direction in terms of variance. For instance, the output `[0.52, 0.45, 0.03]` is interpreted as "the first component accounts for 52% of the variance of my observations, the second 45%, and the third one 3%".

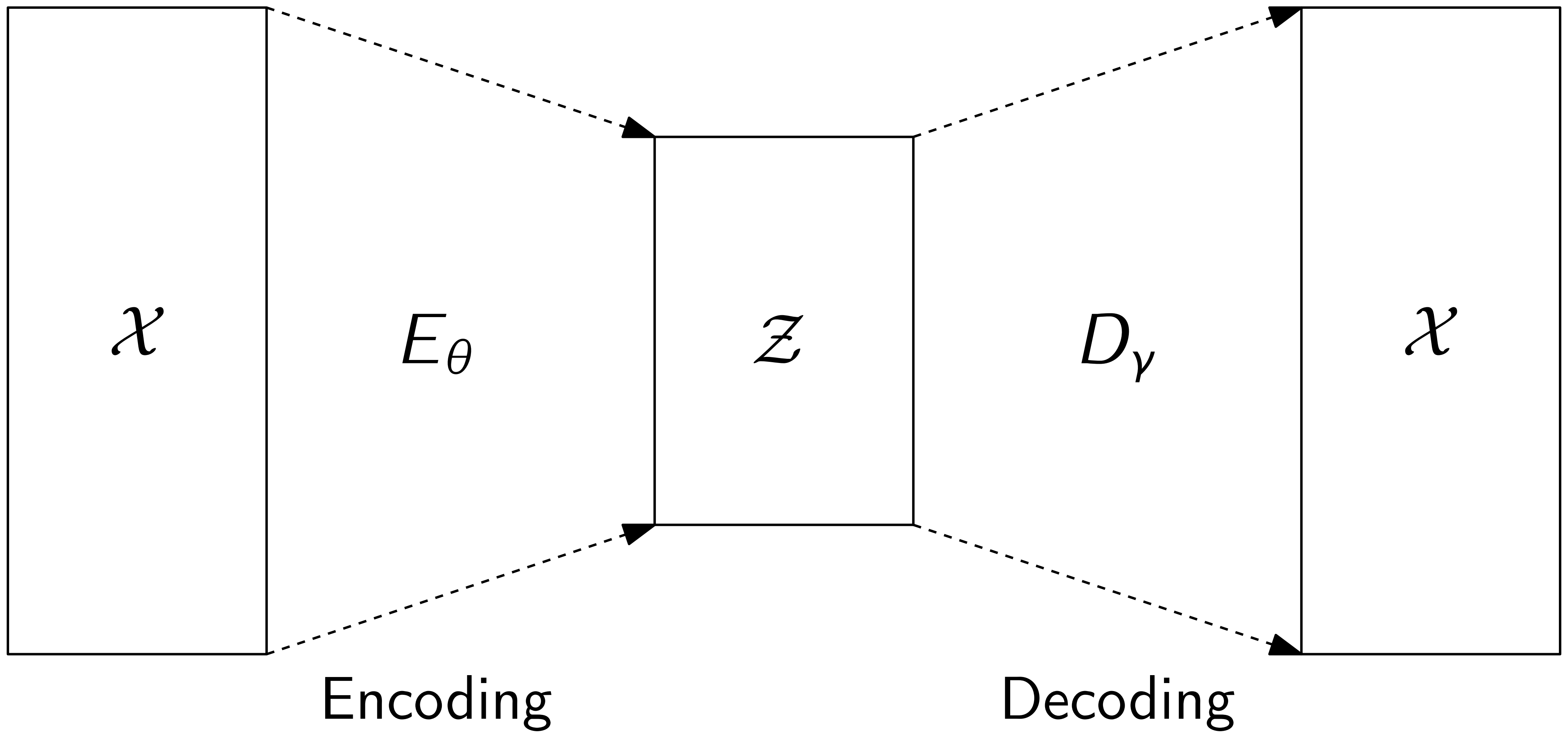
3. Autoencoders

3. Autoencoders

Let $\mathcal{X}$ and $\mathcal{Z}$ be two spaces. An autoencoder (AE) is a couple of two parametrized models $E_\theta : \mathcal{X} \to \mathcal{Z}$ and $D_\gamma : \mathcal{Z} \to \mathcal{X}$ trained such that (essentially)

$$x \simeq D_\gamma(E_\theta(x)).$$

E_θ is said to be the encoder and D_γ is the decoder. The set $\mathcal{Z}$ is called the latent space.



3. Autoencoders

Let $\mathcal{X}$ and $\mathcal{Z}$ be two spaces. An autoencoder (AE) is a couple of two parametrized models $E_\theta : \mathcal{X} \to \mathcal{Z}$ and $D_\gamma : \mathcal{Z} \to \mathcal{X}$ trained such that (essentially)

$$x \simeq D_\gamma(E_\theta(x)).$$

E_θ is said to be the encoder and D_γ is the decoder. The set $\mathcal{Z}$ is called the latent space.

Typically,

- $\mathcal{X} = \mathbb{R}^D$ and $\mathcal{Z} = \mathbb{R}^d$ with $d \ll D$, hence AE can be seen as dimensionality reduction techniques.
- E_θ and D_γ are neural networks, i.e. sequences of linear transformations followed by non-linear activations: $x = x_0 \to \sigma_1(W_1 x_0 + b_1) = x_1 \to \sigma_2(W_2 x_1 + b_2) = x_2 \to \cdots \to x_L$.

3. Autoencoders

Let $\mathcal{X}$ and $\mathcal{Z}$ be two spaces. An autoencoder (AE) is a couple of two parametrized models $E_\theta : \mathcal{X} \to \mathcal{Z}$ and $D_\gamma : \mathcal{Z} \to \mathcal{X}$ trained such that (essentially)

$$x \simeq D_\gamma(E_\theta(x)).$$

E_θ is said to be the encoder and D_γ is the decoder. The set $\mathcal{Z}$ is called the latent space.

Typically,

- $\mathcal{X} = \mathbb{R}^D$ and $\mathcal{Z} = \mathbb{R}^d$ with $d \ll D$, hence AE can be seen as dimensionality reduction techniques.
- E_θ and D_γ are neural networks, i.e. sequences of linear transformations followed by non-linear activations: $x = x_0 \to \sigma_1(W_1 x_0 + b_1) = x_1 \to \sigma_2(W_2 x_1 + b_2) = x_2 \to \cdots \to x_L$.

Some applications:

- Dimensionality reduction and compression,
- Noise reduction: intuitively, if the reconstruction is not perfect, it's likely (hopefully) that the salient features have been reproduced and the noise removed,
- Anomaly detection: AE are expected to have worse reconstruction performanced on anomalies,
- Data generation: sampling a new z in $\mathcal{Z}$ and then "decoding" it using D_γ should (hopefully) provide a new "likely" observation!

3. Autoencoders

Let $\mathcal{X}$ and $\mathcal{Z}$ be two spaces. An autoencoder (AE) is a couple of two parametrized models $E_\theta : \mathcal{X} \to \mathcal{Z}$ and $D_\gamma : \mathcal{Z} \to \mathcal{X}$ trained such that (essentially)

$$x \simeq D_\gamma(E_\theta(x)).$$

E_θ is said to be the encoder and D_γ is the decoder. The set $\mathcal{Z}$ is called the latent space.

Under-determination: Observe that if we compose E_θ and D_γ by any diffeomorphism $\varphi : \mathcal{Z} \to \mathcal{Z}$ (i.e. we consider $(\varphi \circ D_\theta, E_\gamma \circ \varphi^{-1})$) the performance is unchanged. Therefore, the problem is heavily under-determined.

It is thus natural to consider regularized version of AE, either by

- Restricting the class of models (i.e. very shallow networks (L small)),
- Adding regularization in the reconstruction to favor smooth encoder/decoder,
- Add some penalty term on the encoding, e.g. reproduce geometric or topological properties of the input training set $x_1, \ldots, x_n$ (see "Topological Auto Encoders" for instance).

Chapter 6: Kernel methods

This chapter is dedicated to a class of methods that enables the use of basics (typically linear) models for data that may not be linearly separable... or that may not even be living in a Euclidean space!

Chapter 6: Kernel methods

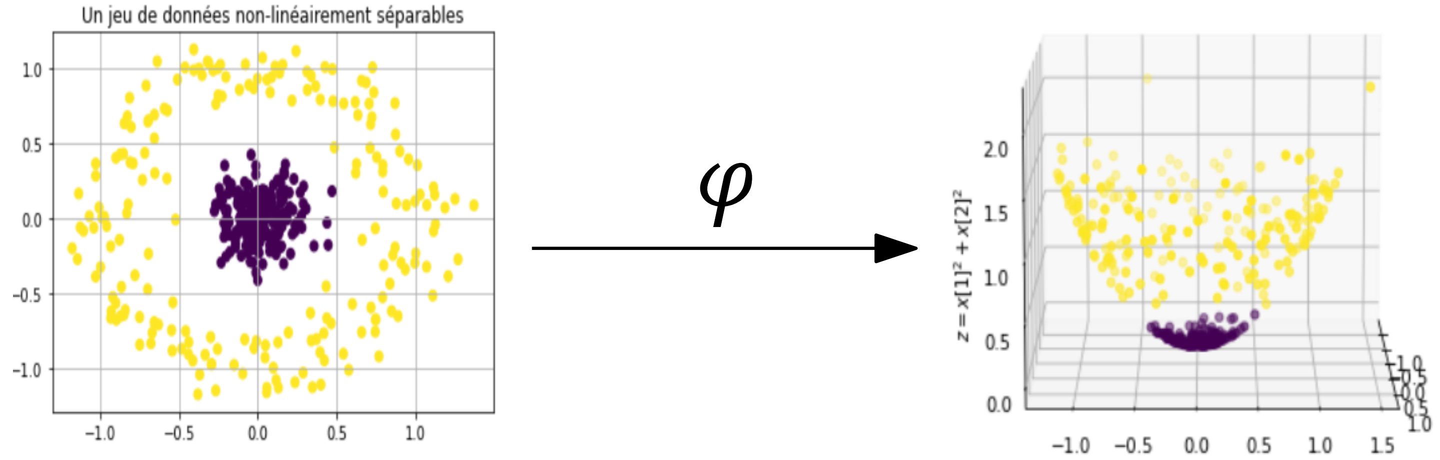
This chapter is dedicated to a class of methods that enables the use of basics (typically linear) models for data that may not be linearly separable... or that may not even be living in a Euclidean space!

1. Introduction and main idea: Consider data living in a possibly abstract set $\mathcal{X}$ (e.g. text, graphs...) and assume that you can find a feature map $\varphi : \mathcal{X} \to \mathcal{H}$ where $\mathcal{H}$ is a Hilbert space, that is a vector space equipped with an inner-product $\langle \cdot, \cdot \rangle_{\mathcal{H}}$ which is complete for the norm $x \mapsto \langle x, x \rangle_{\mathcal{H}}^{\frac{1}{2}}$.
As $\mathcal{H}$ has a linear structure, we can run our favorite algorithm (k-means, classification...) using the "representations / embeddings / featurizations / vectorizations" $\varphi(x)$. If φ is well-chosen for our problem, we may achieve good performances even with simple models.

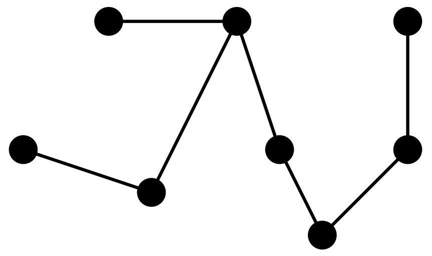
This chapter is dedicated to a class of methods that enables the use of basics (typically linear) models for data that may not be linearly separable... or that may not even be living in a Euclidean space!

1. Introduction and main idea: Consider data living in a possibly abstract set $\mathcal{X}$ (e.g. text, graphs...) and assume that you can find a feature map $\varphi : \mathcal{X} \to \mathcal{H}$ where $\mathcal{H}$ is a Hilbert space, that is a vector space equipped with an inner-product $\langle \cdot, \cdot \rangle_{\mathcal{H}}$ which is complete for the norm $x \mapsto \langle x, x \rangle_{\mathcal{H}}^{\frac{1}{2}}$.
As $\mathcal{H}$ has a linear structure, we can run our favorite algorithm (k-means, classification...) using the "representations / embeddings / featurizations / vectorizations" $\varphi(x)$. If φ is well-chosen for our problem, we may achieve good performances even with simple models.

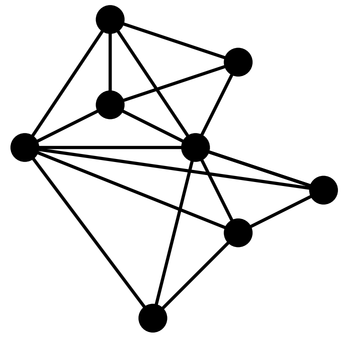
Example 1: Take $\varphi : \mathbb{R}^2 \to \mathbb{R}^3$, defined as $\varphi(a, b) = (a, b, a^2 + b^2)$.



Example 2: Say our data are graphs: $x = (V, E)$, where V is a set of vertices and $E \subset V \times V$ is the set of edges. We may define $\varphi : (V, E) \mapsto (\#V, \#E, \frac{\#E}{\#V}) \in \mathbb{R}^3$. This may be a good *featurisation* of our data (e.g. if we need to discriminate between densely/sparsely connected large/small graphs).



$\xrightarrow{\varphi}$ $(8, 7, 7/8)$



$\xrightarrow{\varphi}$ $(8, 17, 17/8)$

Chapter 6: Kernel methods

This chapter is dedicated to a class of methods that enables the use of basics (typically linear) models for data that may not be linearly separable... or that may not even be living in a Euclidean space!

1. Introduction and main idea: Consider data living in a possibly abstract set $\mathcal{X}$ (e.g. text, graphs...) and assume that you can find a feature map $\varphi : \mathcal{X} \to \mathcal{H}$ where $\mathcal{H}$ is a Hilbert space, that is a vector space equipped with an inner-product $\langle \cdot, \cdot \rangle_{\mathcal{H}}$ which is complete for the norm $x \mapsto \langle x, x \rangle_{\mathcal{H}}^{\frac{1}{2}}$.
As $\mathcal{H}$ has a linear structure, we can run our favorite algorithm (k-means, classification...) using the "representations / embeddings / featurizations / vectorizations" $\varphi(x)$. If φ is well-chosen for our problem, we may achieve good performances even with simple models.

Exercise:

1. Show that performing a polynomial regression of degree d on a set $(x_i, y_i)_{i=1}^n$ (with observations and labels in $\mathbb{R}$) can be understood as performing a linear regression for a suited feature map φ. What is the embedding dimension (dimension of $\mathcal{H}$)? What is the complexity to solve this problem (using the closed form formula , see Chapter 2)?
2. Show that the parameter θ of this linear regression can be assumed to be of the form $\theta = \sum_{i=1}^n b_i \varphi(x_i)$, where $b_i \in \mathbb{R}$ for $i = 1, \dots, n$.
3. Deduce that the optimal θ^* only depends on the Gram matrix $G = (\langle \varphi(x_i), \varphi(x_j) \rangle)_{ij}$ and the vector of labels $Y = (y_1, \dots, y_n)$.
4. Does the observations made in Questions 2 and 3 depend on the choice of φ? What is the computational complexity of this approach? Does it depend on the embedding dimension?
5. What can you conclude from this?

Chapter 6: Kernel methods

This chapter is dedicated to a class of methods that enables the use of basics (typically linear) models for data that may not be linearly separable... or that may not even be living in a Euclidean space!

1. Introduction and main idea: Consider data living in a possibly abstract set $\mathcal{X}$ (e.g. text, graphs...) and assume that you can find a feature map $\varphi : \mathcal{X} \to \mathcal{H}$ where $\mathcal{H}$ is a Hilbert space, that is a vector space equipped with an inner-product $\langle \cdot, \cdot \rangle_{\mathcal{H}}$ which is complete for the norm $x \mapsto \langle x, x \rangle_{\mathcal{H}}^{\frac{1}{2}}$.
As $\mathcal{H}$ has a linear structure, we can run our favorite algorithm (k-means, classification...) using the "representations / embeddings / featurizations / vectorizations" $\varphi(x)$. If φ is well-chosen for our problem, we may achieve good performances even with simple models.

The **crucial observation** is that training many linear models (including Linear Regression from the previous example) can be done by **manipulating only the inner-products** $\langle \varphi(x_i), \varphi(x_j) \rangle_{\mathcal{H}}$. This is called the kernel trick. It means that we do not have to explicitly compute the embeddings $\varphi(x)$, as long as we are capable of computing the inner-products

$$K(x, x') := \langle \varphi(x), \varphi(x') \rangle_{\mathcal{H}} .$$

Chapter 6: Kernel methods

This chapter is dedicated to a class of methods that enables the use of basics (typically linear) models for data that may not be linearly separable... or that may not even be living in a Euclidean space!

1. Introduction and main idea: Consider data living in a possibly abstract set $\mathcal{X}$ (e.g. text, graphs...) and assume that you can find a feature map $\varphi : \mathcal{X} \to \mathcal{H}$ where $\mathcal{H}$ is a Hilbert space, that is a vector space equipped with an inner-product $\langle \cdot, \cdot \rangle_{\mathcal{H}}$ which is complete for the norm $x \mapsto \langle x, x \rangle_{\mathcal{H}}^{\frac{1}{2}}$.
As $\mathcal{H}$ has a linear structure, we can run our favorite algorithm (k-means, classification...) using the "representations / embeddings / featurizations / vectorizations" $\varphi(x)$. If φ is well-chosen for our problem, we may achieve good performances even with simple models.

The **crucial observation** is that training many linear models (including Linear Regression from the previous example) can be done by **manipulating only the inner-products** $\langle \varphi(x_i), \varphi(x_j) \rangle_{\mathcal{H}}$. This is called the kernel trick. It means that we do not have to explicitly compute the embeddings $\varphi(x)$, as long as we are capable of computing the inner-products

$$K(x, x') := \langle \varphi(x), \varphi(x') \rangle_{\mathcal{H}} .$$

??? How could we compute $\langle \varphi(x), \varphi(x') \rangle_{\mathcal{H}}$ without computing the vectorizations $\varphi(x), \varphi(x')$?

Chapter 6: Kernel methods

This chapter is dedicated to a class of methods that enables the use of basics (typically linear) models for data that may not be linearly separable... or that may not even be living in a Euclidean space!

1. Introduction and main idea: Consider data living in a possibly abstract set $\mathcal{X}$ (e.g. text, graphs...) and assume that you can find a feature map $\varphi : \mathcal{X} \to \mathcal{H}$ where $\mathcal{H}$ is a Hilbert space, that is a vector space equipped with an inner-product $\langle \cdot, \cdot \rangle_{\mathcal{H}}$ which is complete for the norm $x \mapsto \langle x, x \rangle_{\mathcal{H}}^{\frac{1}{2}}$.
As $\mathcal{H}$ has a linear structure, we can run our favorite algorithm (k-means, classification...) using the "representations / embeddings / featurizations / vectorizations" $\varphi(x)$. If φ is well-chosen for our problem, we may achieve good performances even with simple models.

The **crucial observation** is that training many linear models (including Linear Regression from the previous example) can be done by **manipulating only the inner-products** $\langle \varphi(x_i), \varphi(x_j) \rangle_{\mathcal{H}}$. This is called the kernel trick. It means that we do not have to explicitly compute the embeddings $\varphi(x)$, as long as we are capable of computing the inner-products

$$K(x, x') := \langle \varphi(x), \varphi(x') \rangle_{\mathcal{H}} .$$

??? How could we compute $\langle \varphi(x), \varphi(x') \rangle_{\mathcal{H}}$ without computing the vectorizations $\varphi(x), \varphi(x')$?

Bold (but brilliant) idea: Define $K : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ first, and hope that if K satisfies some good properties, then there may exist a Hilbert space $\mathcal{H}$ and a feature map $\varphi : \mathcal{X} \to \mathcal{H}$ such that $K(x, x') = \langle \varphi(x), \varphi(x') \rangle_{\mathcal{H}}$.
If this holds, we can directly compute the Gram matrix G from the $(K(x_i, x_j))_{ij}$ without never explicitly computing the $\varphi(x)$!

2. Reproducing Kernel Hilbert Spaces (RKHS)

Consider a set $\mathcal{X}$ and a map $K : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$. Let us try to find some necessary conditions on K to have

$$K(x, x') = \langle \varphi(x), \varphi(x') \rangle_{\mathcal{H}}$$

for some Hilbert space $\mathcal{H}$ and some $\varphi : \mathcal{X} \to \mathcal{H}$.

2. Reproducing Kernel Hilbert Spaces (RKHS)

Consider a set $\mathcal{X}$ and a map $K : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$. Let us try to find some necessary conditions on K to have

$$K(x, x') = \langle \varphi(x), \varphi(x') \rangle_{\mathcal{H}}$$

for some Hilbert space $\mathcal{H}$ and some $\varphi : \mathcal{X} \to \mathcal{H}$.

- First, K should be symmetric.

2. Reproducing Kernel Hilbert Spaces (RKHS)

Consider a set $\mathcal{X}$ and a map $K : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$. Let us try to find some necessary conditions on K to have

$$K(x, x') = \langle \varphi(x), \varphi(x') \rangle_{\mathcal{H}}$$

for some Hilbert space $\mathcal{H}$ and some $\varphi : \mathcal{X} \to \mathcal{H}$.

- First, K should be symmetric.
- Second, observe that for any $n \in \mathbb{N}$, $\lambda_1, \dots, \lambda_n \in \mathbb{R}$, $x_1, \dots, x_n \in \mathcal{X}$,

$$0 \leqslant \|\sum_{i=1}^{n} \lambda_i \varphi(x_i)\|_{\mathcal{H}}^2 = \sum_{1 \leqslant i,j \leqslant n} \lambda_i \lambda_j \langle \varphi(x_i), \varphi(x_j) \rangle_{\mathcal{H}} = \sum_{1 \leqslant i,j \leqslant n} \lambda_i \lambda_j K(x_i, x_j). \tag{20}$$

2. Reproducing Kernel Hilbert Spaces (RKHS)

Consider a set $\mathcal{X}$ and a map $K : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$. Let us try to find some necessary conditions on K to have

$$K(x, x') = \langle \varphi(x), \varphi(x') \rangle_{\mathcal{H}}$$

for some Hilbert space $\mathcal{H}$ and some $\varphi : \mathcal{X} \to \mathcal{H}$.

- First, K should be symmetric.
- Second, observe that for any $n \in \mathbb{N}$, $\lambda_1, \dots, \lambda_n \in \mathbb{R}$, $x_1, \dots, x_n \in \mathcal{X}$,

$$0 \leqslant \|\sum_{i=1}^{n} \lambda_i \varphi(x_i)\|_{\mathcal{H}}^2 = \sum_{1 \leqslant i,j \leqslant n} \lambda_i \lambda_j \langle \varphi(x_i), \varphi(x_j) \rangle_{\mathcal{H}} = \sum_{1 \leqslant i,j \leqslant n} \lambda_i \lambda_j K(x_i, x_j). \tag{20}$$

Definition:

A map $K : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ satisfying these two assumptions is said to be a positive semidefinite (PSD) kernel.

2. Reproducing Kernel Hilbert Spaces (RKHS)

Theorem:

Let K be a PSD kernel on a set $\mathcal{X}$. Then there exists a Hilbert space $\mathcal{H}$ and a map $\varphi : \mathcal{X} \to \mathcal{H}$ such that

$$K(x, x') = \langle \varphi(x), \varphi(x') \rangle_{\mathcal{H}}$$

for all x, x' in $\mathcal{X}$.

2. Reproducing Kernel Hilbert Spaces (RKHS)

Theorem:

Let K be a PSD kernel on a set $\mathcal{X}$. Then there exists a Hilbert space $\mathcal{H}$ and a map $\varphi : \mathcal{X} \to \mathcal{H}$ such that

$$K(x, x') = \langle \varphi(x), \varphi(x') \rangle_{\mathcal{H}}$$

for all x, x' in $\mathcal{X}$.

Proof: Define $\varphi : \mathcal{X} \to \mathbb{R}^{\mathcal{X}}$ as $\varphi(x) = K(x, \cdot)$. Let $\mathcal{H}_0$ be the vector space of all finite sums $\sum_{i=1}^n \lambda_i \varphi(x_i)$, for $n \in \mathbb{N}, \lambda_i \in \mathbb{R}, x_i \in \mathcal{X}$. Now, for $f = \sum_{i=1}^n \lambda_i \varphi(x_i)$ and $g = \sum_{j=1}^m \mu_j \varphi(x'_j)$ in $\mathcal{H}_0$, define

$$\langle f, g \rangle_{\mathcal{H}_0} := \sum_{i=1}^n \sum_{j=1}^m \lambda_i \mu_j \varphi(x_i) \varphi(x'_j),$$

and check that it properly defines an inner product on $\mathcal{H}_0$. Eventually, consider the completion $\mathcal{H}$ of $\mathcal{H}_0$, that is a Hilbert space by definition, and observe that $\langle \varphi(x), \varphi(x') \rangle_{\mathcal{H}} = K(x, x')$ by construction.

2. Reproducing Kernel Hilbert Spaces (RKHS)

Theorem:

Let K be a PSD kernel on a set $\mathcal{X}$. Then there exists a Hilbert space $\mathcal{H}$ and a map $\varphi : \mathcal{X} \to \mathcal{H}$ such that

$$K(x, x') = \langle \varphi(x), \varphi(x') \rangle_{\mathcal{H}}$$

for all x, x' in $\mathcal{X}$.

Remark (some terminology): Since we defined $\varphi(x) = K(x, \cdot)$, observe that for any $f = \sum_{i=1}^n \lambda_i \varphi(x_i)$ in $\mathcal{H}_0$ (and by limit in $\mathcal{H}$), one has

$$f(x) = \sum_{i=1}^n \lambda_i \varphi(x_i)(x) = \sum_{i=1}^n \lambda_i K(x_i, x) = \sum_{i=1}^n \lambda_i \langle \varphi(x_i), \varphi(x) \rangle_{\mathcal{H}_0} = \langle f, \varphi(x) \rangle_{\mathcal{H}_0} = \langle f, K(x, \cdot) \rangle_{\mathcal{H}_0}$$

so in a nutshell we can evaluate f at x by computing the inner-product of f with $K(x, \cdot)$, so we can "reproduce" f from the kernel K, hence we say that $\mathcal{H}$ is a Reproducing Kernel Hilbert Space (associated to the reproducing kernel K).

3. Some properties and examples.

Proposition:

Let K_1, K_2 be two PSD kernels on a set $\mathcal{X}$. Then,

1. $K_1 + K_2$ is a PSD kernel,
2. $K_1 \cdot K_2$ is a PSD kernel,
3. If $\mathcal{X} \subset \mathbb{R}^d$ and $K(x, x') = h(x - x')$ for some h, then K is a kernel if the Fourier transform of h

$$\hat{h}(\omega) := \int e^{-2i\pi\langle \omega, x\rangle} h(x) \mathrm{d}x$$

is non-negative for every $\omega \in \mathbb{R}^d$.

Proof: Exercise.

3. Some properties and examples.

Proposition:

1. If $\varphi : \mathcal{X} \to \mathcal{H}$ for some Hilbert space $\mathcal{H}$, then $K(x, x') = \langle \varphi(x), \varphi(x') \rangle_{\mathcal{H}}$ is a Kernel.
2. If K is a kernel, K^n is a kernel for $n \in \mathbb{N}$. In particular, $(x, x') \mapsto \langle x, x' \rangle_{\mathcal{H}}^n$ defines a kernel on $\mathcal{H}$ (you can take $\mathcal{H} = \mathbb{R}^d$).
3. For $\sigma > 0$, the function $K_\sigma : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ given by

$$K(x, x') = \exp\left(-\frac{\|x - x'\|^2}{2\sigma^2}\right)$$

defines the so-called Gaussian kernel (also called RBF).

Proof: 1. and 2. are clear (2. follows by induction from the previous proposition). For 3., exercise!

Chapter 6: Kernel methods

3. Some properties and examples.

Some intuition: The Gaussian Kernel $(x,x') \mapsto \exp\left(-\frac{\|x-x'\|_2^2}{2\sigma^2}\right)$ is widely used as it naturally catches some geometric information of your (Euclidean) data :

- x close to x' $\Rightarrow$ $\|x-x'\|$ small $\Rightarrow$ $K(x,x') \simeq 1$ $\rightarrow$ high similarity,
- x far from x' $\Rightarrow$ $\|x-x'\|$ large $\Rightarrow$ $K(x,x') = \langle \varphi(x), \varphi(x') \rangle_{\mathcal{H}} \simeq 0$ $\rightarrow$ the embeddings are (almost) orthogonal in the Hilbert space.

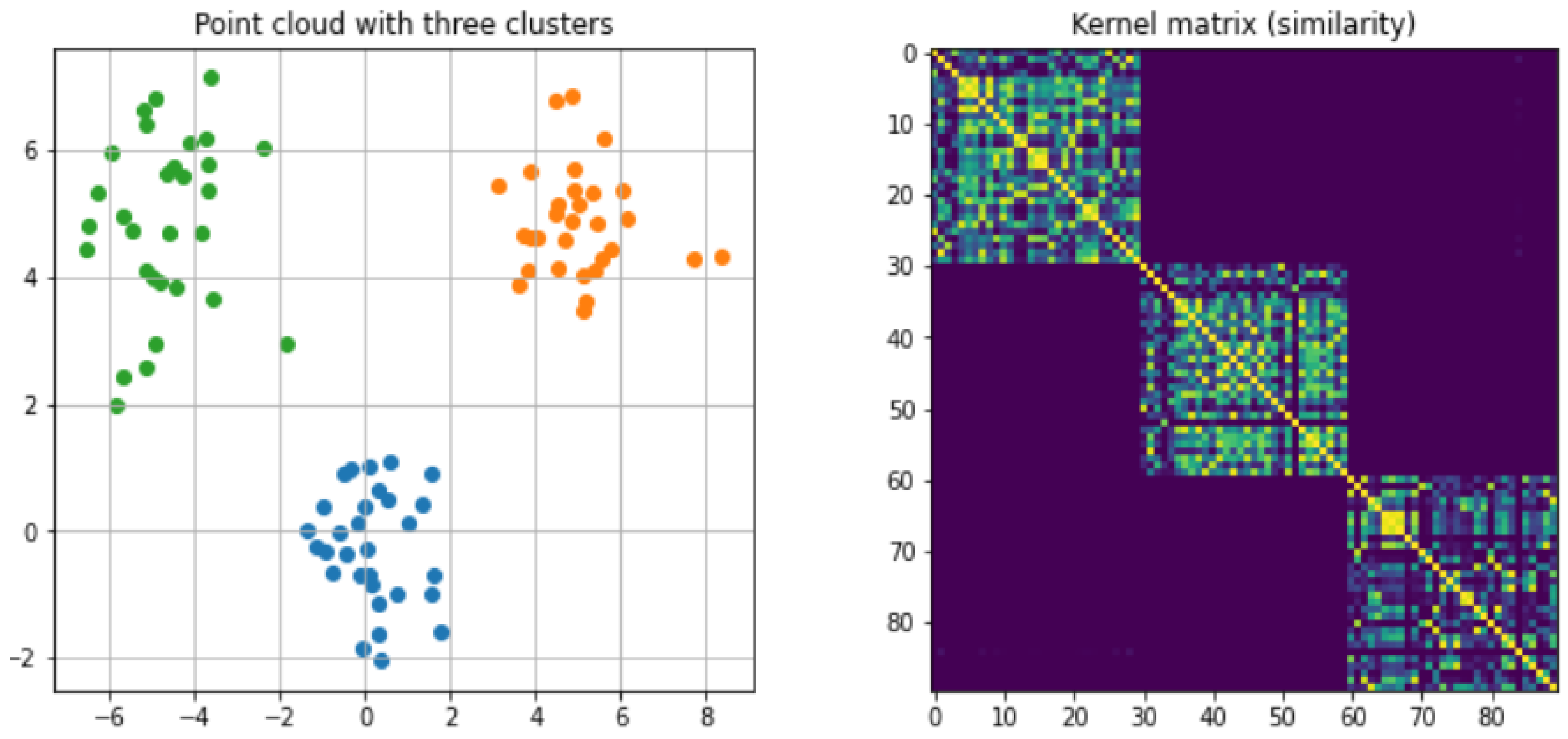


Illustration on linearly separable clusters

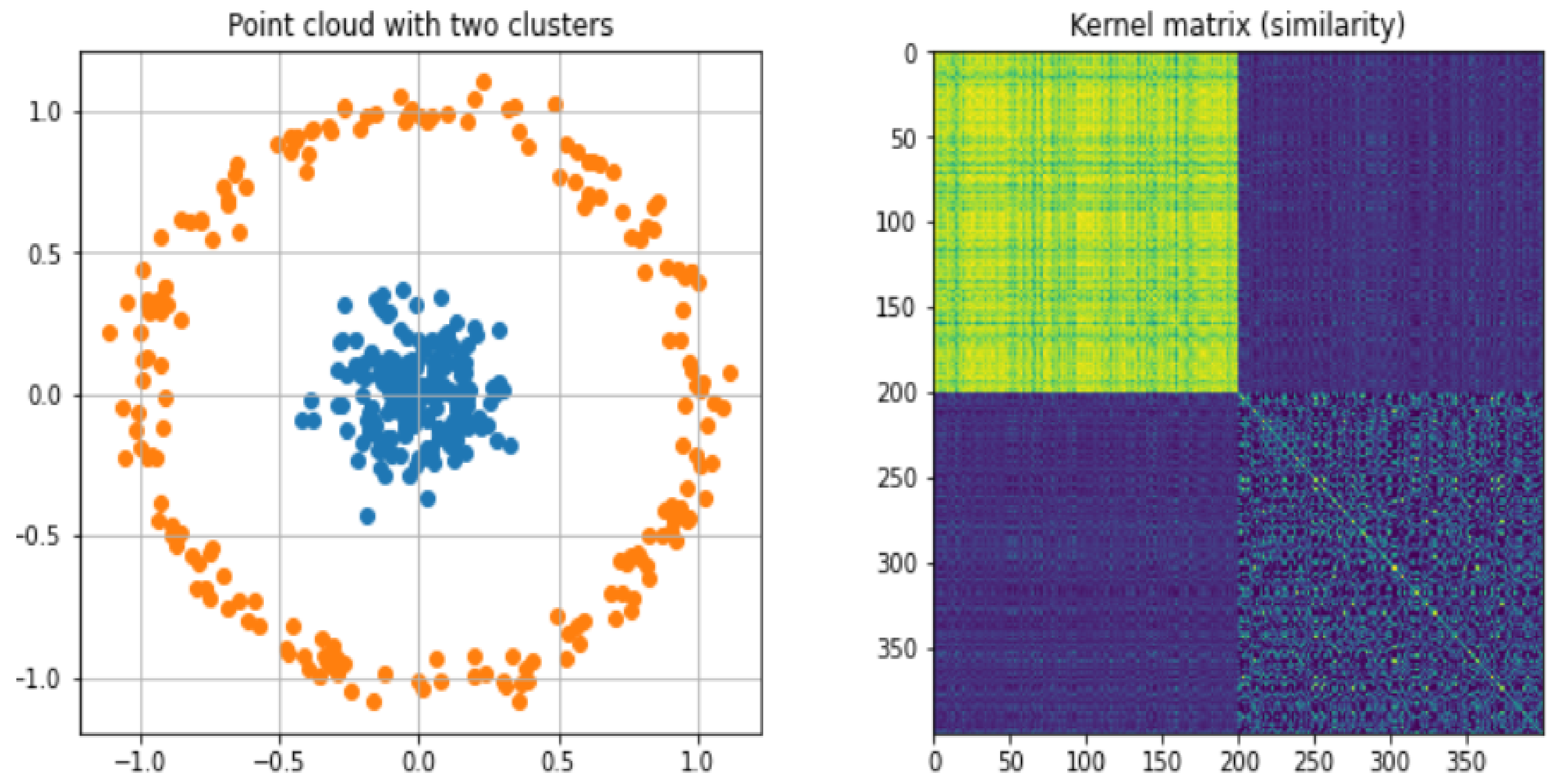


Illustration on non-linearly separable clusters

4. Examples of Kernel trick.

4. Examples of Kernel trick.

• 4.1. Kernel PCA.

Let $\mathcal{X}$ be a set, that we do not assume to be Euclidean (e.g. words, graphs...). Let $X = (x_1, \dots, x_n) \subset \mathcal{X}$ be a set of observations, and assume that we are given a kernel K on $\mathcal{X}$, and $\varphi : \mathcal{X} \to \mathcal{H}$ be the corresponding feature map (with RKHS $\mathcal{H}$).
As $\mathcal{X}$ has no structure, we cannot apply PCA on X directly. However, we can consider the embedded point cloud $Z = (\varphi(x_1), \dots, \varphi(x_n)) \subset \mathcal{H}$.
Recall that PCA in $\mathbb{R}^d$ required to compute the $d \times d$ **covariance** matrix $C = X^T X$. Here, as $\mathcal{H}$ may be infinite dimensional, this does not make sense, and we rather consider the Gram matrix $ZZ^T \in \mathbb{R}^{n\times n}$ whose coordinates are by definition $K(x_i, x_j)$, that can also be diagonalized, etc. This only requires to know K, not φ.

Application: Take a batch of 785 words with two main groups: words refering to countries (e.g. France, Italy, India, etc.) and words refering to feelings (e.g. sadness, joy, etc.). Build a Kernel based on the W2V-embedding [Mikolov et al., 2013], and apply Kernel-PCA with dimension 2.

Credit: Vincent Divol

4. Examples of Kernel trick.

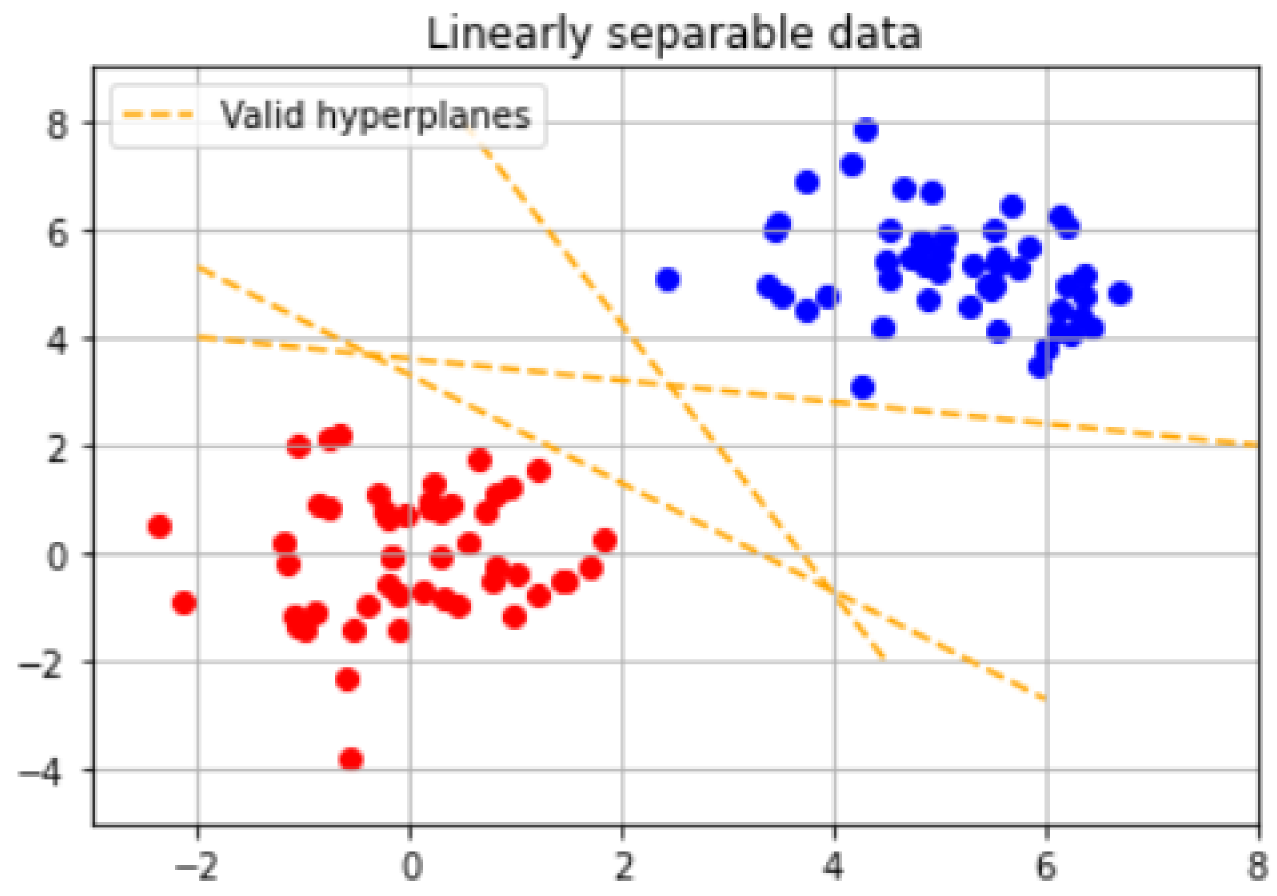
• 4.2. Kernel SVM.

The Support Vector Machine (SVM) is a very popular model to design some sort of "optimal **linear** classifier" for binary classification. As we'll see, though being linear in its seminal formulation, it can be "kernelized" and thus used to separate non-linear data.

4. Examples of Kernel trick.

• 4.2. Kernel SVM.

We consider a binary classification problem, with $\mathcal{Y} = \{-1, +1\}$ (for convenience).
Assume first that the observations are in $\mathcal{X} = \mathbb{R}^d$, and that they are linearly separable.
The performance (accuracy) of a (binary) classifier is entirely determined by its decision boundary. Many classifiers could be optimal for our problem...

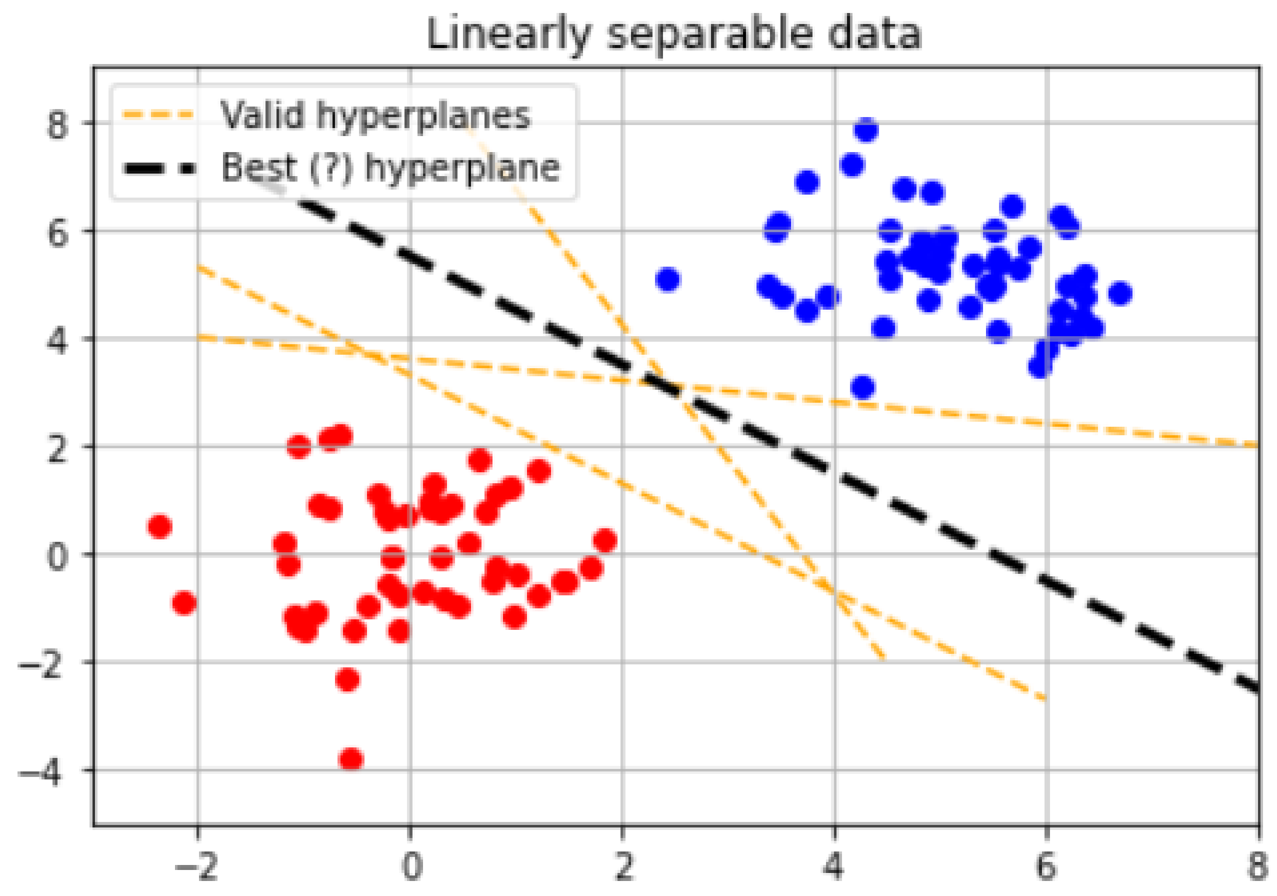


4. Examples of Kernel trick.

• 4.2. Kernel SVM.

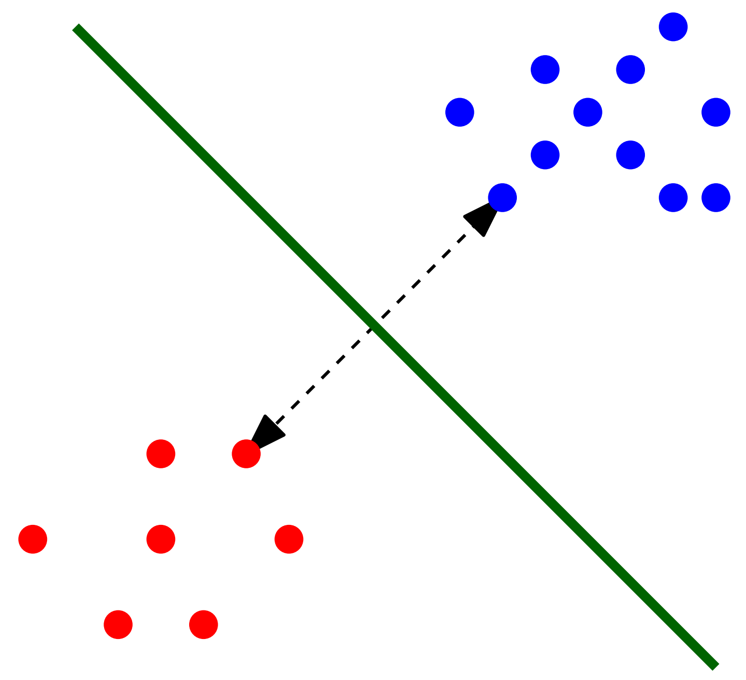
We consider a binary classification problem, with $\mathcal{Y} = \{-1, +1\}$ (for convenience).
Assume first that the observations are in $\mathcal{X} = \mathbb{R}^d$, and that they are linearly separable.
The performance (accuracy) of a (binary) classifier is entirely determined by its decision boundary. Many classifiers could be optimal for our problem...

... But some classifiers are more optimal than others!



4. Examples of Kernel trick.

• 4.2. Kernel SVM.

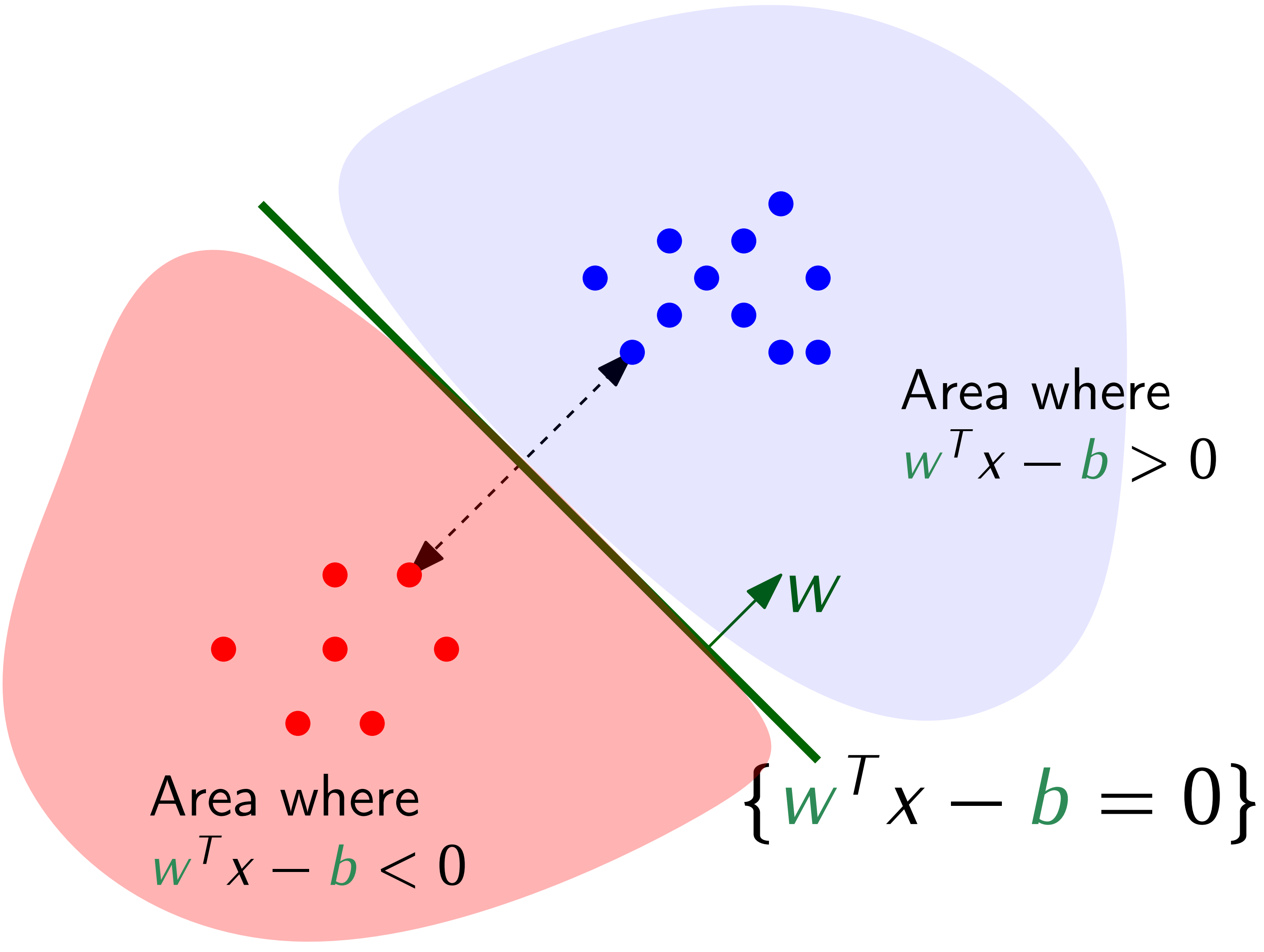


Idea: The (linear) SVM model encourages the decision boundary to **maximize** a margin condition: being as far as possible from the observations ($\Rightarrow$ more robust, better generalization, etc.).

4. Examples of Kernel trick.

• 4.2. Kernel SVM.

Idea: The (linear) SVM model encourages the decision boundary to **maximize** a margin condition: being as far as possible from the observations ($\Rightarrow$ more robust, better generalization, etc.).



Formally: An affine hyperplane $H_{w,b}$ of $\mathbb{R}^d$ is described the equation

$$w^T x - b = 0,$$

where $w \in \mathbb{R}^d$ is a normal vector of the hyperplane and $b \in \mathbb{R}$. Saying that $H_{w,b}$ perfectly separates the data $(x_i, y_i)_{i=1}^n$ means that for all $i = 1, \dots, n$

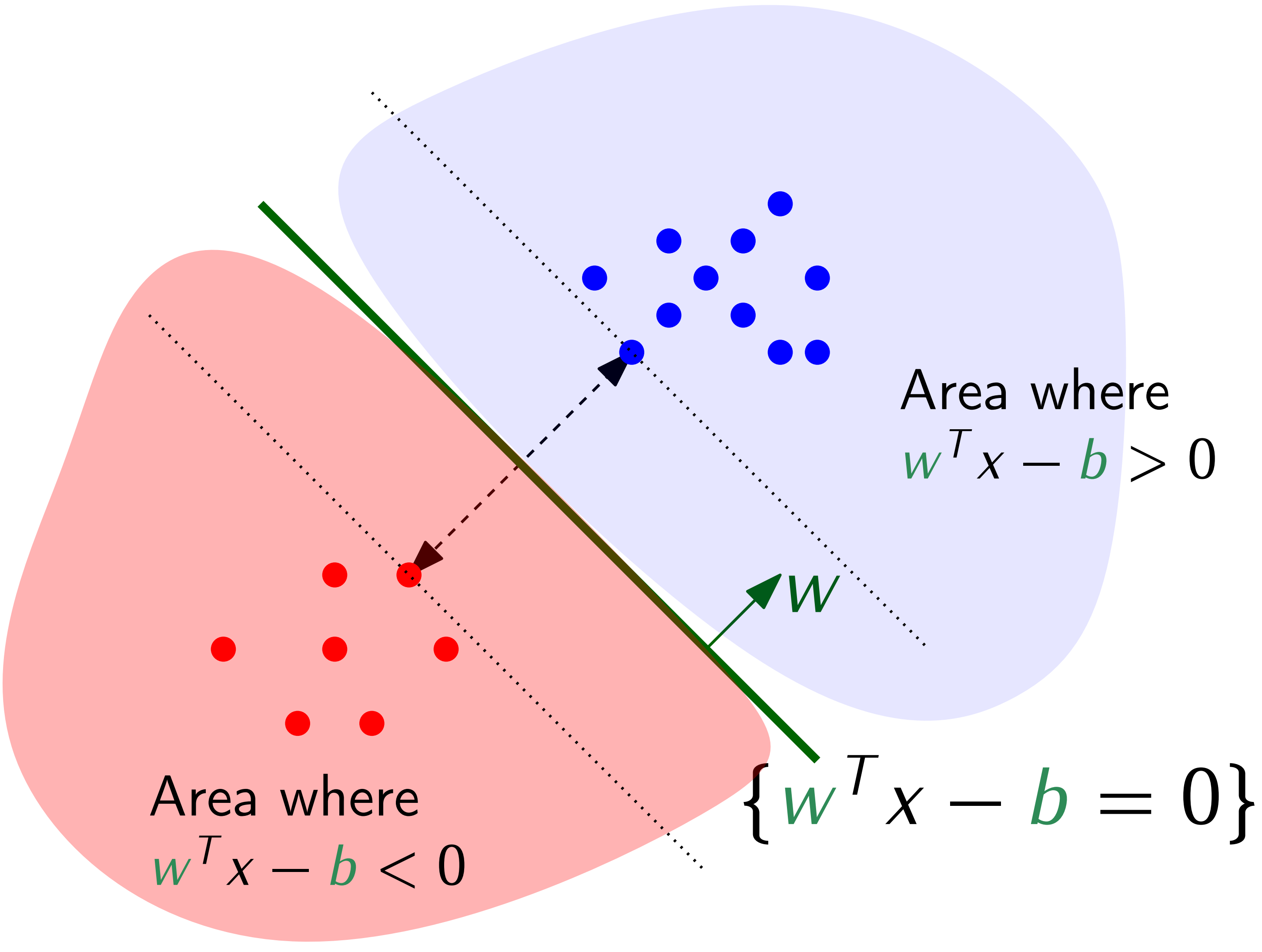
$$w^T x_i - b > 0 \text{ if } y_i = 1, \qquad w^T x_i - b < 0 \text{ if } y_i = -1$$

or, in compact form and using that we can rescale w, b,

$$\forall i = 1, \dots, n, \quad y_i(w^T x_i - b) \geqslant 1 \tag{21}$$

4. Examples of Kernel trick.

• 4.2. Kernel SVM.



Idea: The (linear) SVM model encourages the decision boundary to **maximize** a margin condition: being as far as possible from the observations (⇒ more robust, better generalization, etc.).

Formally: An affine hyperplane $H_{w,b}$ of $\mathbb{R}^d$ is described the equation

$$w^T x - b = 0,$$

where $w \in \mathbb{R}^d$ is a normal vector of the hyperplane and $b \in \mathbb{R}$. Saying that $H_{w,b}$ perfectly separates the data $(x_i, y_i)_{i=1}^n$ means that for all $i = 1, \dots, n$

$$w^T x_i - b > 0 \text{ if } y_i = 1, \qquad w^T x_i - b < 0 \text{ if } y_i = -1$$

or, in compact form and using that we can rescale w, b,

$$\forall i = 1, \dots, n, \quad y_i(w^T x_i - b) \geqslant 1 \tag{21}$$

Eventually, the margin of a valid $H_{w,b}$ is given by the distance between the two limit hyperplanes $\{w^T x - b = \pm 1\}$ and the distance between these two hyperplanes is $\frac{2}{\|w\|}$ (homework), so maximizing the margin means minimizing $\|w\|$.

4. Examples of Kernel trick.

- 4.2. Kernel SVM.

Definition:

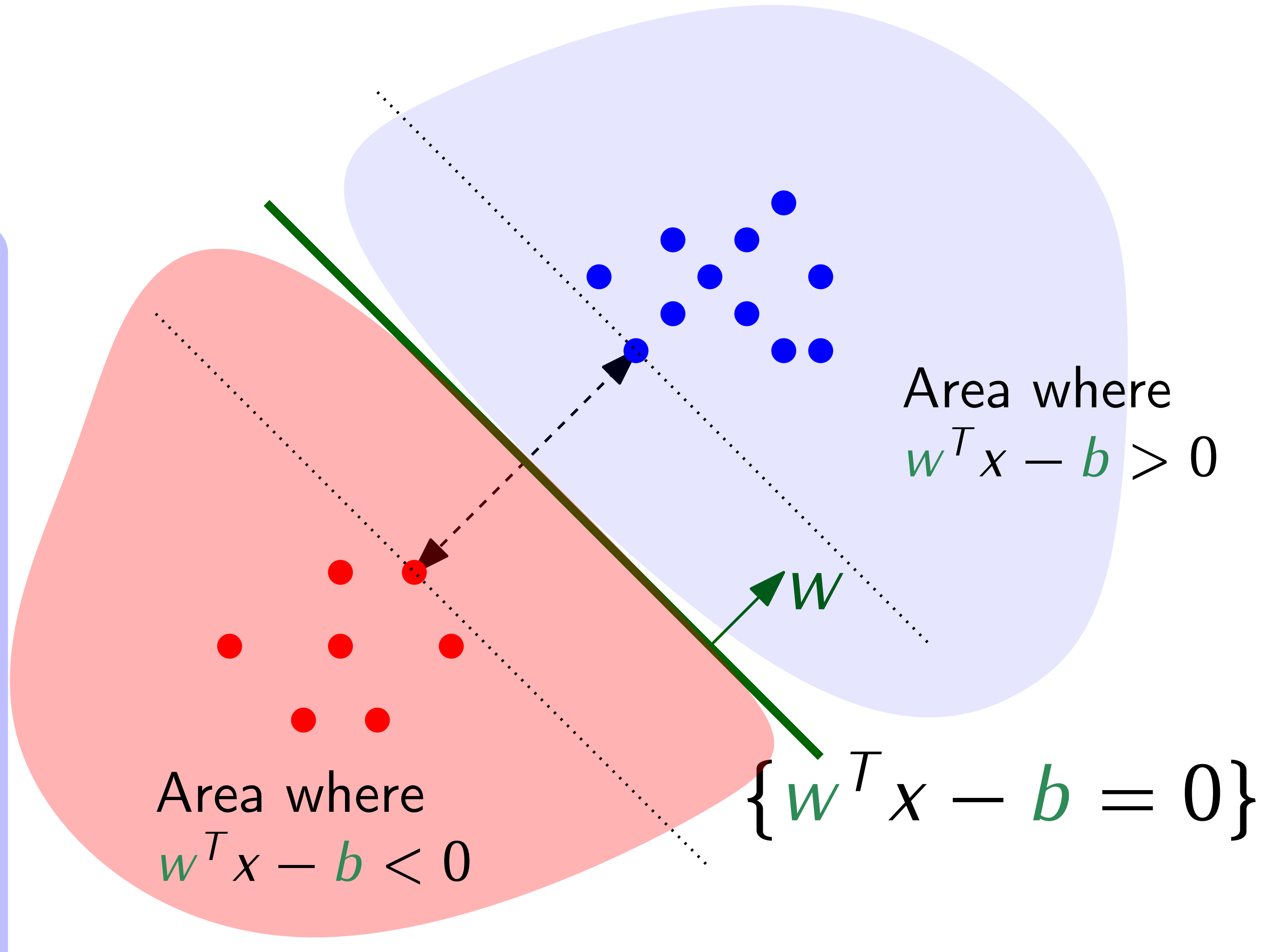
The hard-margin linear SVM model is the (binary) classifier defined by

$$x \mapsto \mathrm{sign}(w^T x - b),$$

where $w \in \mathbb{R}^d$ and $b \in \mathbb{R}$ are solutions of the constrained optimization problem

$$\min_{w,b} ||w||^2,$$

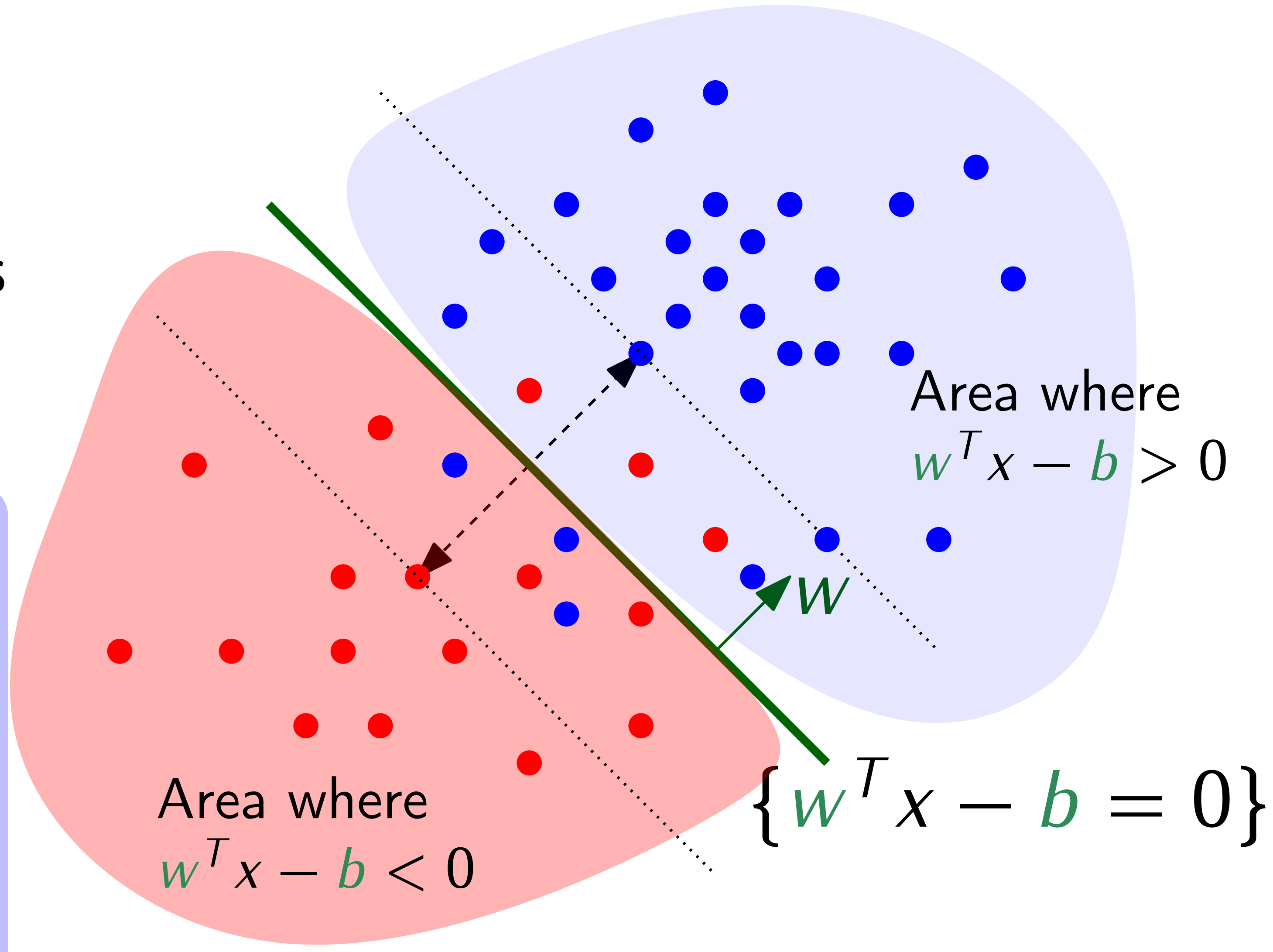
$$\text{subject to } \forall i = 1, \dots, n,\ y_i(w^T x_i - b) \geqslant 1.$$



4. Examples of Kernel trick.

• 4.2. Kernel SVM.

Remark: If the observations are not linearly separable, the set of valid hyperplanes for the hard-margin SVM is empty (the problem is infeasible). Therefore, it is convenient to consider a softened version of SVM in practice.



Definition:

The soft-margin linear SVM model is the (binary) classifier defined by

$$x \mapsto \mathrm{sign}(w^T x - b),$$

where $w \in \mathbb{R}^d$ and $b \in \mathbb{R}$ are solutions of the unconstrained optimization problem

$$\min_{w,b} \left\{ \|w\|^2 + \lambda \frac{1}{n} \sum_{i=1}^{n} \psi(1 - y_i(w^T x_i - b)) \right\},$$

where $\lambda > 0$ is an hyper-parameter and $\psi : \mathbb{R} \to \mathbb{R}$ is a divergence (we pay we when violate the constraint); for instance one can use $\psi(t) = \max(0, t)$— the so-called Hinge loss.

4. Examples of Kernel trick.

• 4.2. Kernel SVM.

Kernel trick: Eventually, assume now that our observations $(x_i)_{i=1}^n$ belong to a set $\mathcal{X}$ equipped with a (PSD) kernel $K : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$. We know that there exist a Hilbert space $\mathcal{H}$ and a map $\varphi : \mathcal{X} \to \mathcal{H}$ such that $K(x, x') = \langle \varphi(x), \varphi(x') \rangle_{\mathcal{H}}$. Let us show that we can solve the SVM problem in the embedding space $\mathcal{H}$, namely

$$\min_{w,b} \left\{ \|w\|^2 + \lambda \frac{1}{n} \sum_{i=1}^n \psi(1 - y_i(\langle w, \underline{\varphi(x_i)} \rangle_{\mathcal{H}} - b)) \right\}, \quad \text{with } w \in \mathcal{H}, b \in \mathbb{R}, \tag{22}$$

by only manipulating K (i.e. we do not need to know φ nor $\mathcal{H}$). For this, we rely on the following proposition...

4. Examples of Kernel trick.

• 4.2. Kernel SVM.

Kernel trick: Eventually, assume now that our observations $(x_i)_{i=1}^n$ belong to a set $\mathcal{X}$ equipped with a (PSD) kernel $K : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$. We know that there exist a Hilbert space $\mathcal{H}$ and a map $\varphi : \mathcal{X} \to \mathcal{H}$ such that $K(x, x') = \langle \varphi(x), \varphi(x') \rangle_{\mathcal{H}}$. Let us show that we can solve the SVM problem in the embedding space $\mathcal{H}$, namely

$$\min_{w,b} \left\{ \|w\|^2 + \lambda \frac{1}{n} \sum_{i=1}^n \psi(1 - y_i(\langle w, \varphi(x_i) \rangle_{\mathcal{H}} - b)) \right\}, \quad \text{with } w \in \mathcal{H}, b \in \mathbb{R}, \tag{22}$$

by only manipulating K (i.e. we do not need to know φ nor $\mathcal{H}$). For this, we rely on the following proposition...

Proposition:

When solving (22), one can restrict to $w = \sum_{i=1}^n a_i \varphi(x_i)$.

Exercise: Prove this proposition.

4. Examples of Kernel trick.

• 4.2. Kernel SVM.

Kernel trick: Eventually, assume now that our observations $(x_i)_{i=1}^n$ belong to a set $\mathcal{X}$ equipped with a (PSD) kernel $K : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$. We know that there exist a Hilbert space $\mathcal{H}$ and a map $\varphi : \mathcal{X} \to \mathcal{H}$ such that $K(x, x') = \langle \varphi(x), \varphi(x') \rangle_{\mathcal{H}}$. Let us show that we can solve the SVM problem in the embedding space $\mathcal{H}$, namely

$$\min_{w,b} \left\{ \|w\|^2 + \lambda \frac{1}{n} \sum_{i=1}^n \psi(1 - y_i(\langle w, \varphi(x_i) \rangle_{\mathcal{H}} - b)) \right\}, \quad \text{with } w \in \mathcal{H}, b \in \mathbb{R}, \tag{22}$$

by only manipulating K (i.e. we do not need to know φ nor $\mathcal{H}$). For this, we rely on the following proposition...

Proposition:

When solving (22), one can restrict to $w = \sum_{i=1}^n a_i \varphi(x_i)$.

Corollary: We can solve the SVM problem with the embedded observations $\varphi(x_1), \ldots, \varphi(x_n)$ by solving (e.g. with Gradient Descent)

$$\min_{a \in \mathbb{R}^n, b \in \mathbb{R}} \left\{ \sum_{1 \leqslant i,j \leqslant n} a_i a_j K(x_i, x_j) + \frac{\lambda}{n} \sum_{1 \leqslant i \leqslant n} \psi \left(1 - y_i \left(\sum_{j=1}^n a_j K(x_i, x_j) - b \right) \right) \right\}.$$

5. Some limitations of Kernel methods.

- Choosing/Defining a good kernel is often hard.
- Losing interpretability: "what happen in the RKHS stays in the RKHS". For instance, you may compute an average $\mu := \frac{1}{n}\sum_{i=1}^{n} \varphi(x_i) \in \mathcal{H}$, but there is no reason to expect that there exists some x such that $\mu = \varphi(x)$.
- You typically need to compute and store the Gram matrix, which is of size $n \times n$, yielding a complexity of $\mathcal{O}(n^2)$. If you need to invert or diagonalize it, the complexity becomes $\mathcal{O}(n^3)$, which tends to be prohibitive for large n (say $n \geqslant 10^4$).