Machine Learning Elements

An (In)formal Introduction

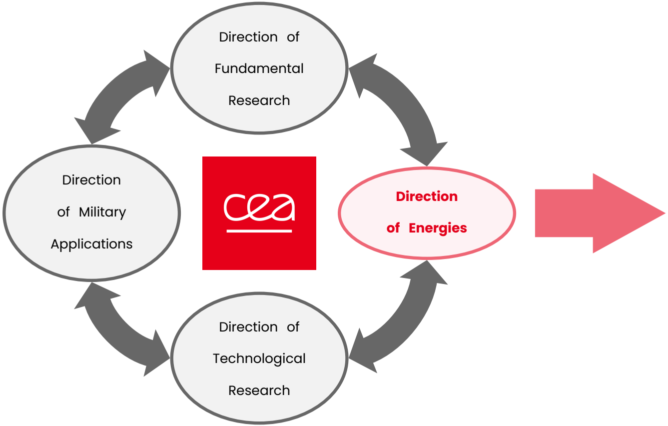
Riccardo Finotello

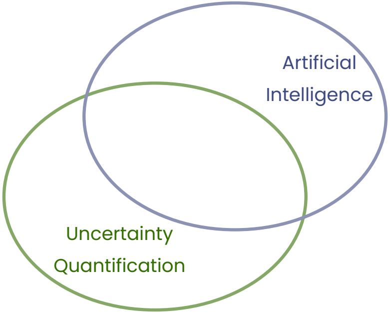
Artificial Intelligence and Modern Physics
A two-way connection

Monopoli (Bari, IT) | 29 September – 5 October 2024

Foreword

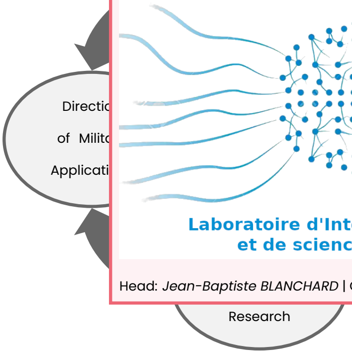
The LIAD Task Force

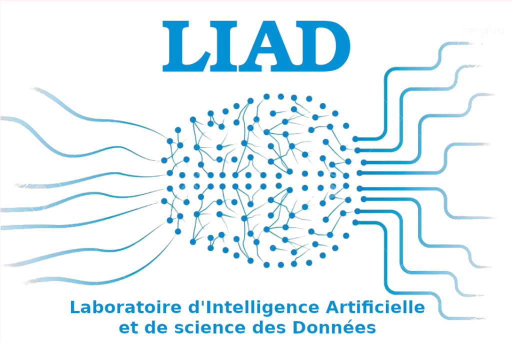




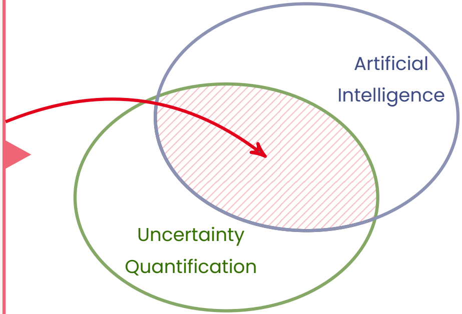
Foreword

The LIAD Task Force





Head: *Jean-Baptiste BLANCHARD* | Core business: URANIE



Foreword

For the Hands-On Sessions

You can follow the instructions at

https://github.com/thesfinox/aiphy-intro-ml-homework

Tutorials are presented as *Jupyter notebooks*. You will have several options to run them:

- run online using *Binder* in extreme cases,
- install Docker and use the dedicated image*,
- create and activate a local Python environment.

Disclaimer

You are warmly invited to download the presentation : some details might be deliberately hidden in small print 🤭.

* This is the preferred method, especially if you are using *Windows* OS. Should you encouter any issues, do not hesitate to ask for help!

Table of contents

1. Some History and Philosophy to Start

What is ML? Some philosophical considerations...

Table of contents

"Programming computers to learn from experience should eventually eliminate the need for much of this detailed programming effort"

Arthur Samuel, IBM (1959)

What is ML?

The historical concept

Birth of the term

A. L. Samuel

Some Studies in Machine Learning Using the Game of Checkers

Abstract: Two machine-learning procedures have been investigated in some detail using the game of checkers. Enough work has been done to verify the fact that a computer can be programmed so that it will learn to play a better game of checkers than can be played by the person who wrote the program. Furthermore, it can learn to do this in a remarkably short period of time (8 or 10 hours of machine-playing time) when given only the rules of the game, a sense of direction, and a redundant and incomplete list of parameters which are thought to have something to do with the game, but whose correct signs and relative weights are unknown and unspecified. The principles of machine learning verified by these experiments are, of course, applicable to many other situations.

Arthur Samuel, IBM Journal (1959)

What is ML?

The historical concept

Birth of the term

A. L. Samuel

Some Studies in Machine Learning Using the Game of Checkers

Abstract: Two machine-learning procedures have been investigated in some detail using the game of checkers. Enough work has been done to verify the fact that a computer can be programmed so that it will learn to play a better game of checkers than can be played by the person who wrote the program. Furthermore, it can learn to do this in a remarkably short period of time (8 or 10 hours of machine-playing time) when given only the rules of the game, a sense of direction, and a redundant and incomplete list of parameters which are thought to have something to do with the game, but whose correct signs and relative weights are unknown and unspecified. The principles of machine learning verified by these experiments are, of course, applicable to many other situations.

Arthur Samuel, IBM Journal (1959)

1. DATA
2. LEARN
3. PREDICT

What is ML?

The historical concept

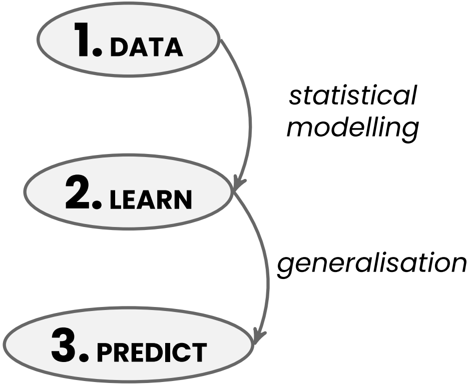
Birth of the term

A. L. Samuel

Some Studies in Machine Learning Using the Game of Checkers

Abstract: Two machine-learning procedures have been investigated in some detail using the game of checkers. Enough work has been done to verify the fact that a computer can be programmed so that it will learn to play a better game of checkers than can be played by the person who wrote the program. Furthermore, it can learn to do this in a remarkably short period of time (8 or 10 hours of machine-playing time) when given only the rules of the game, a sense of direction, and a redundant and incomplete list of parameters which are thought to have something to do with the game, but whose correct signs and relative weights are unknown and unspecified. The principles of machine learning verified by these experiments are, of course, applicable to many other situations.

Arthur Samuel, IBM Journal (1959)

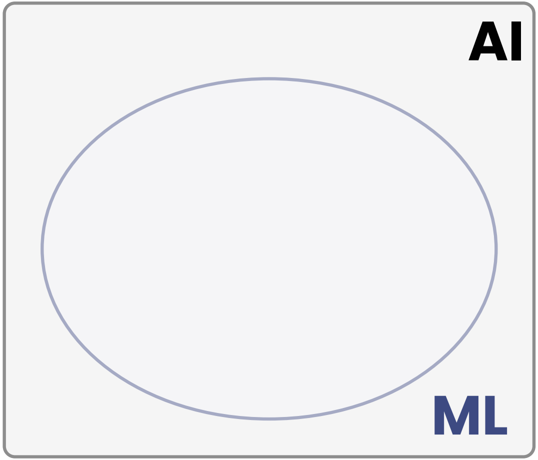


What is ML?
Intuition and nested definitions

- **Artificial Intelligence**
  - “human behaviour” emulation
  - pattern recognition
  - learning processes
  - decision making

What is ML?

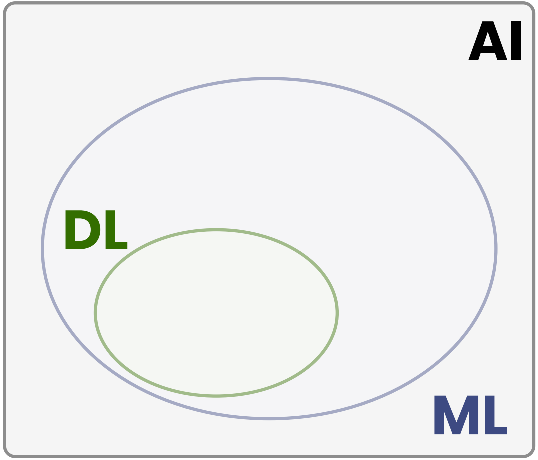
Intuition and nested definitions



- **Artificial Intelligence**
  - "human behaviour" emulation
  - pattern recognition
  - learning processes
  - decision making
- **Machine Learning**
  - data exploitation
  - statistical modelling
  - generalisation on new data

What is ML?

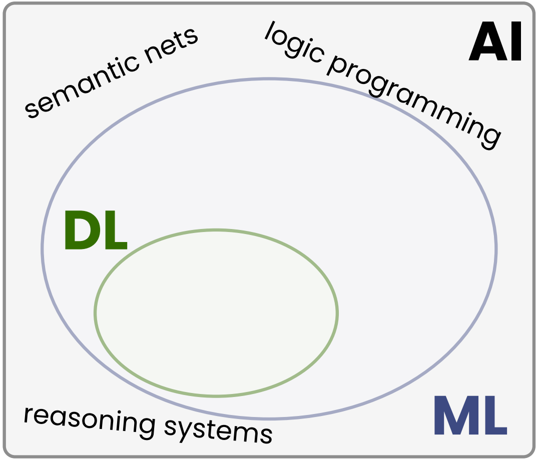
Intuition and nested definitions



- **Artificial Intelligence**
  - “human behaviour” emulation
  - pattern recognition
  - learning processes
  - decision making
- **Machine Learning**
  - data exploitation
  - statistical modelling
  - generalisation on new data
- **Deep Learning**
  - complex pattern processing
  - highly non linear problems
  - “mimic” human brains

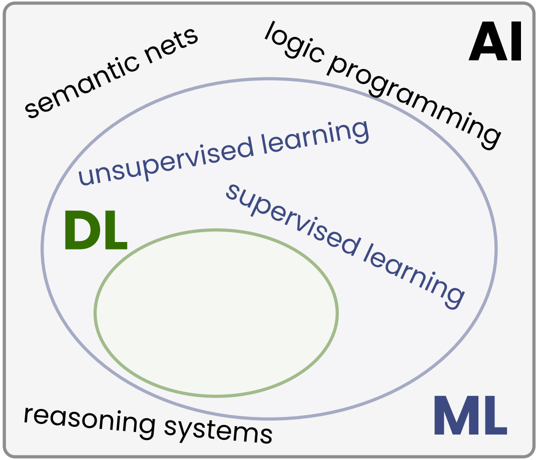
What is ML?

Intuition and nested definitions



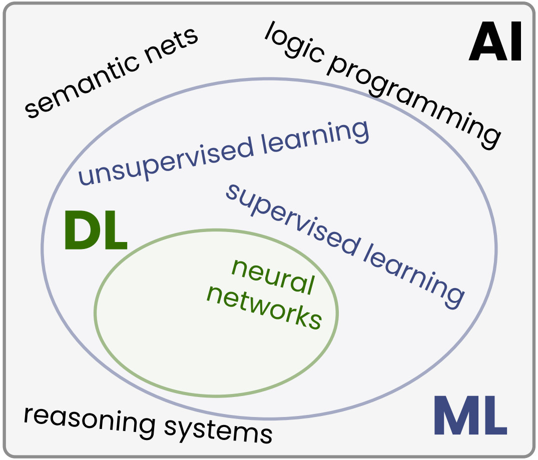
- **Artificial Intelligence**
  - "human behaviour" emulation
  - pattern recognition
  - learning processes
  - decision making
- **Machine Learning**
  - data exploitation
  - statistical modelling
  - generalisation on new data
- **Deep Learning**
  - complex pattern processing
  - highly non linear problems
  - "mimic" human brains

What is ML?
Intuition and nested definitions



- **Artificial Intelligence**
  - “human behaviour” emulation
  - pattern recognition
  - learning processes
  - decision making
- **Machine Learning**
  - data exploitation
  - statistical modelling
  - generalisation on new data
- **Deep Learning**
  - complex pattern processing
  - highly non linear problems
  - “mimic” human brains

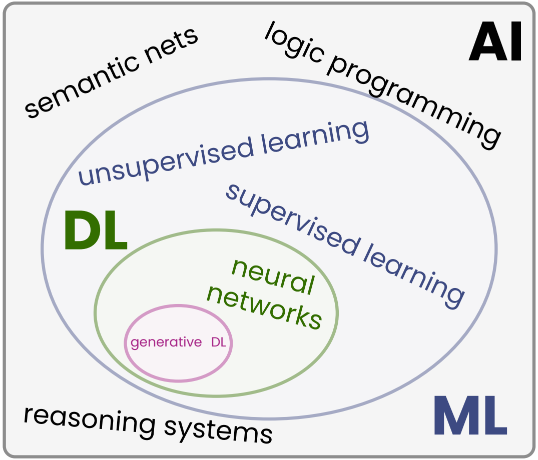
What is ML?
Intuition and nested definitions



- **Artificial Intelligence**
  - “human behaviour” emulation
  - pattern recognition
  - learning processes
  - decision making
- **Machine Learning**
  - data exploitation
  - statistical modelling
  - generalisation on new data
- **Deep Learning**
  - complex pattern processing
  - highly non linear problems
  - “mimic” human brains

What is ML?

Intuition and nested definitions

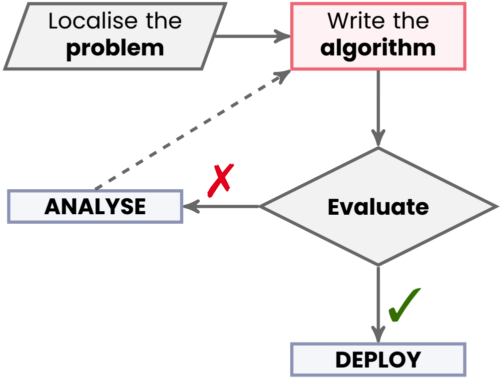


- **Artificial Intelligence**
  - “human behaviour” emulation
  - pattern recognition
  - learning processes
  - decision making
- **Machine Learning**
  - data exploitation
  - statistical modelling
  - generalisation on new data
- **Deep Learning**
  - complex pattern processing
  - highly non linear problems
  - “mimic” human brains

What is ML?
Intuitive behaviour

Write a **spam** filter:

- you know what characterises an email as spam
- you write a set of rules to flag emails as spam
- you evaluate your algorithm and decide whether to deploy it or reassess

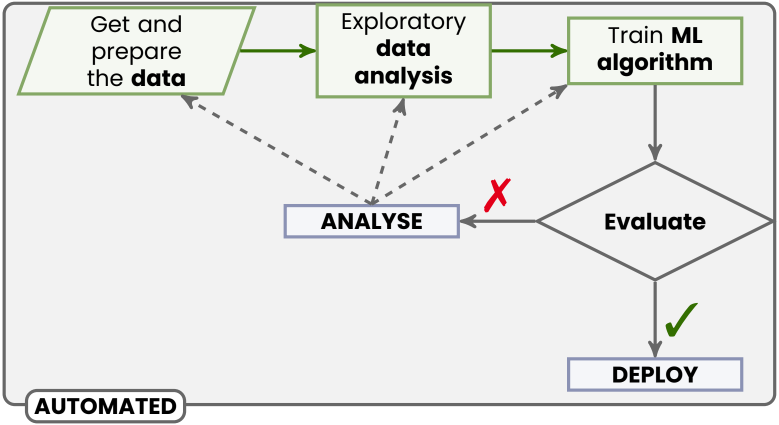


What is ML?

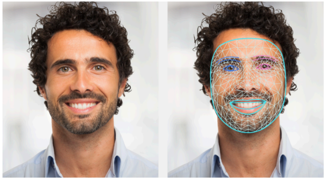
Intuitive behaviour

Write a **spam** filter:

- you prepare a data tidying **pipeline**
- you write a ML training **pipeline** to flag emails as spam
- you write an evaluation **pipeline** and decide whether to deploy the model or reassess

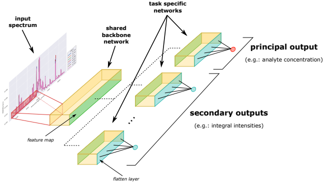


Why ML?
Real world scenarios

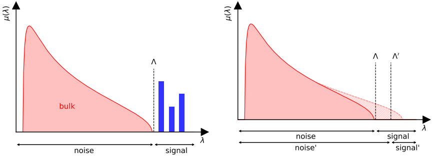


Mediapipe face landmarker by G

Detectron2 panoptic segmentation / 3D pose estimation model by f



Trustworthy *laser-induced breakdown spectroscopy* (arXiv:2210.03762)



Signal detection via *functional renormalization group* (arXiv:2310.07499)

2. The ML Mindset

Data preparation

Table of contents

“ Paradoxically, data is the most under-valued and de-glamorised aspect of AI ”

Sambasivan *et al.* CHI'21

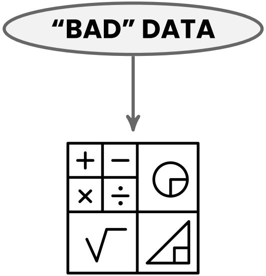
The ML Mindset
Worst case scenario

"BAD" DATA

- **"Bad" input data**
  - *insufficient data* not enough data to learn anything useful
  - *untidy data* bad missing data fillers, wrong categorical encoding, non representative samples, etc…
  - *data leakage* the model "sees" the generalisation data
  - *unbalanced dataset* the model develops a "bias"

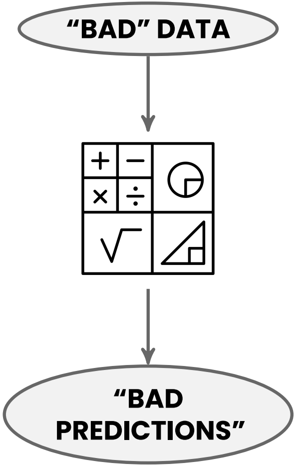
The ML Mindset

Worst case scenario



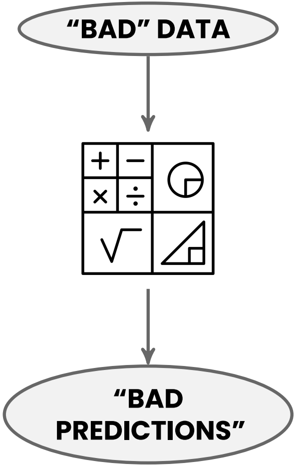
- **"Bad" input data**
  - *insufficient data* not enough data to learn anything useful
  - *untidy data* bad missing data fillers, wrong categorical encoding, non representative samples, etc…
  - *data leakage* the model "sees" the generalisation data
  - *unbalanced dataset* the model develops a "bias"

The ML Mindset
Worst case scenario



- **“Bad” input data**
  - *insufficient data* not enough data to learn anything useful
  - *untidy data* bad missing data fillers, wrong categorical encoding, non representative samples, etc…
  - *data leakage* the model “sees” the generalisation data
  - *unbalanced dataset* the model develops a “bias”
- **“Bad” predictions**
  - *biased model* cannot model input data $\Rightarrow$ bad generalisation performance
  - *overfitting* model is too “adapted” $\Rightarrow$ bad generalisation performance

The ML Mindset
Worst case scenario

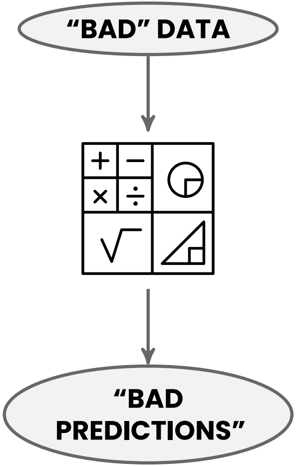


- **"Bad" input data**
  - *insufficient data* not enough data to learn anything useful
  - *untidy data* bad missing data fillers, wrong categorical encoding, non representative samples, etc...
  - *data leakage* the model "sees" the generalisation data
  - *unbalanced dataset* the model develops a "bias"
- **"Bad" predictions**
  - *biased model* cannot model input data ⇒ bad generalisation performance
  - *overfitting* model is too "adapted" ⇒ bad generalisation performance

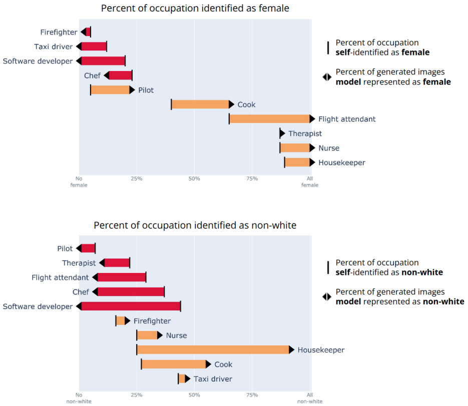
Garbage IN ⇒ Garbage OUT

Racial, sexist, and religious biases in trained models are **always** the result of human errors (even accidental)!

The ML Mindset
Worst case scenario



Simple examples of biases in AI



Bianchi *et al.* FAccT'23

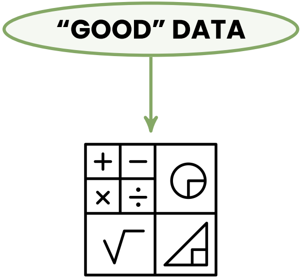
The ML Mindset

ML pipelines and working operations

"GOOD" DATA

The ML Mindset

ML pipelines and working operations



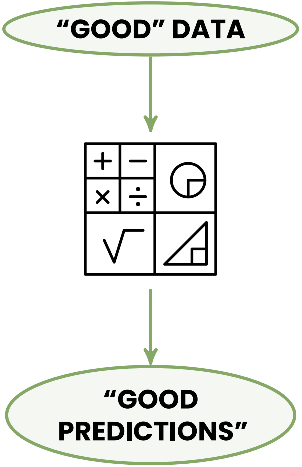
ML pipeline

Introduce a set of "checklists" to…

- …ensure high **data quality** (and tidiness);
- …streamline **analysis** and **model building**;
- …simplify the **learning** process and its **evaluation**;
- …grant **reproducibility** and **experimentation**

The ML Mindset

ML pipelines and working operations



ML pipeline

Introduce a set of “checklists” to…

- …ensure high **data quality** (and tidiness);
- …streamline **analysis** and **model building**;
- …simplify the **learning** process and its **evaluation**;
- …grant **reproducibility** and **experimentation**

=

“ML PIPELINE”

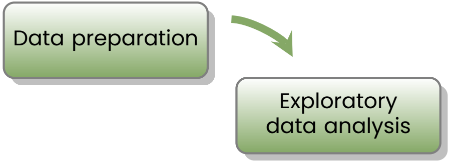
The ML Mindset

A clockwork pipeline

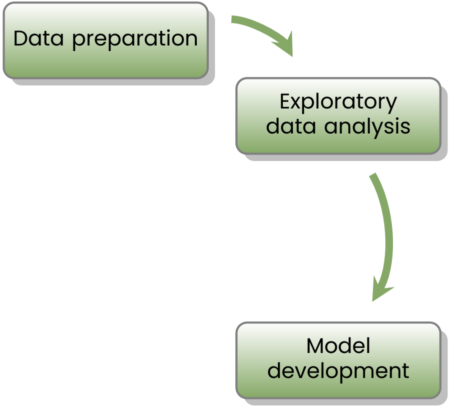
Data preparation

The ML Mindset

A clockwork pipeline

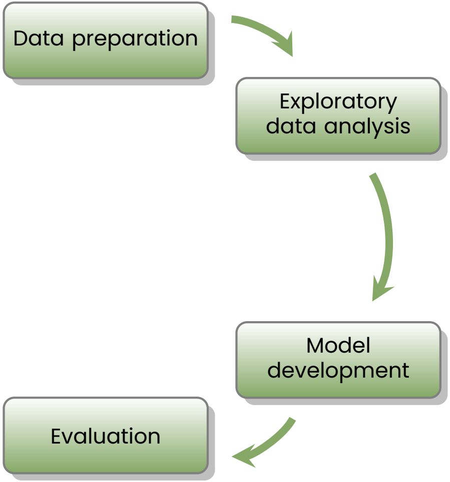


The ML Mindset
A clockwork pipeline

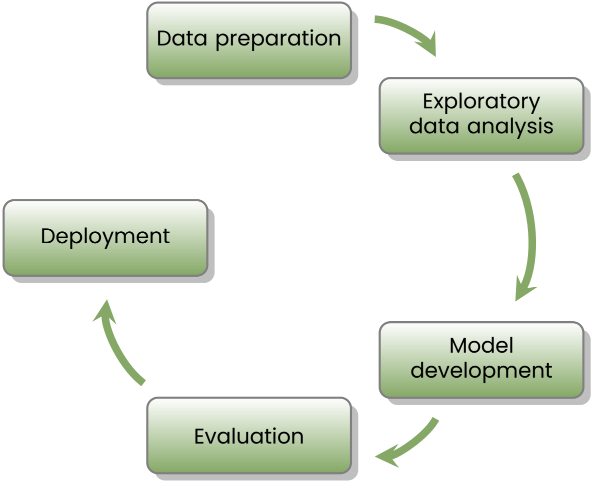


The ML Mindset

A clockwork pipeline

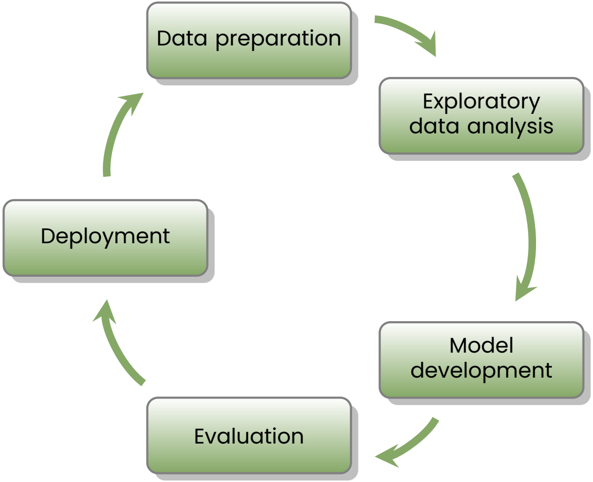


The ML Mindset
A clockwork pipeline



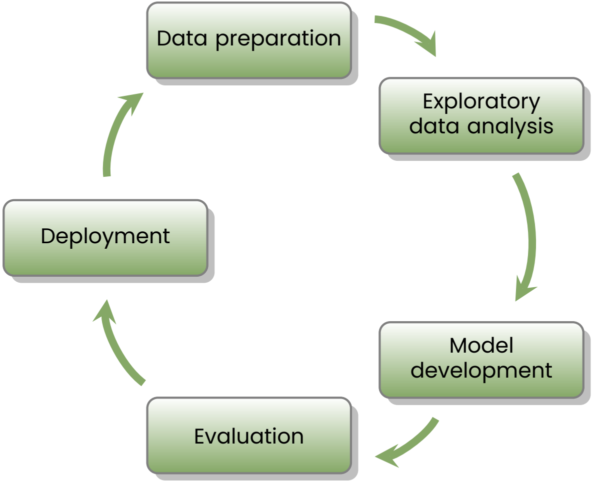
The ML Mindset

A clockwork pipeline



The ML Mindset

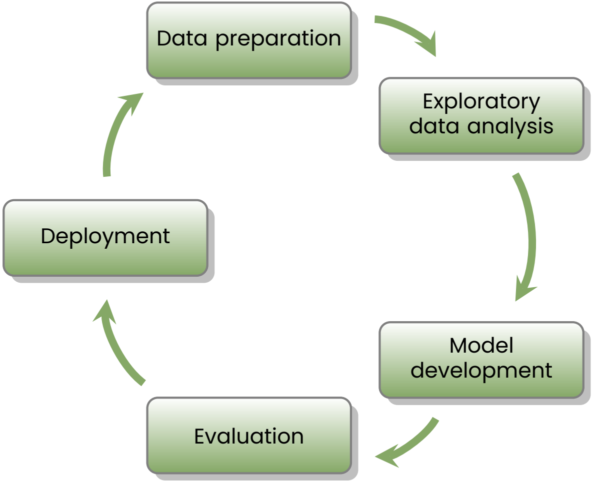
A clockwork pipeline



(xkcd.com)

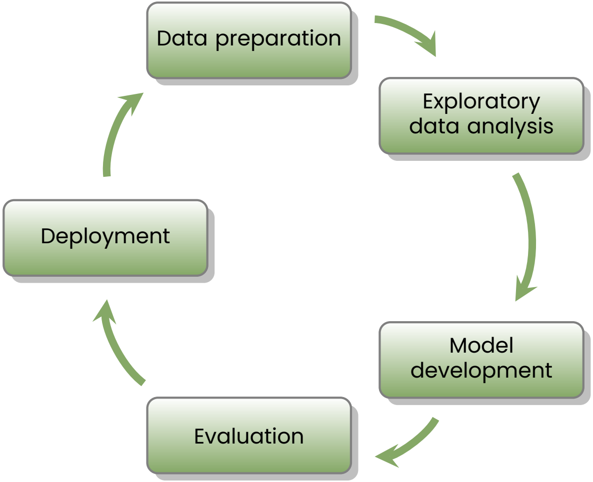
The ML Mindset

A clockwork pipeline



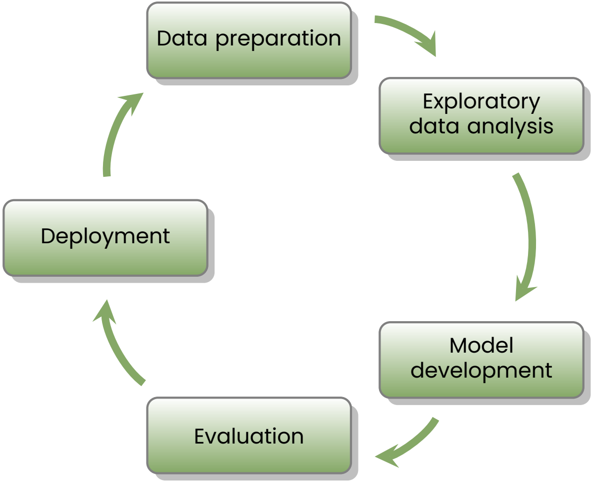
- **modular** break down complex problems into small bricks

The ML Mindset
A clockwork pipeline



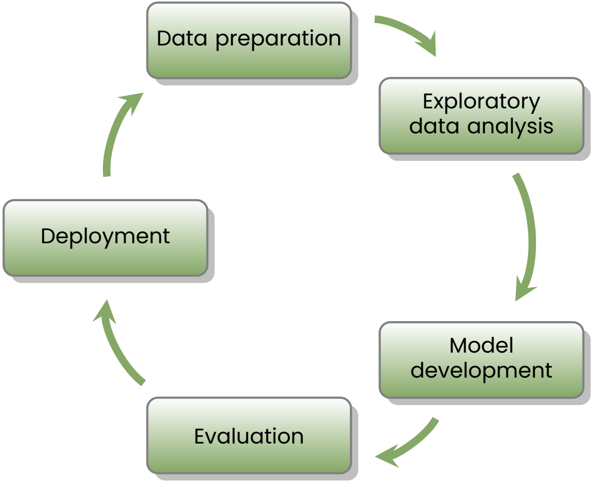
- **modular** break down complex problems into small bricks
- **reproducible** trace back analysis to well-defined breakpoints

The ML Mindset
A clockwork pipeline



- **modular** break down complex problems into small bricks
- **reproducible** trace back analysis to well-defined breakpoints
- **experimental** trial-and-error is permitted and easier to implement

The ML Mindset
A clockwork pipeline



- **modular** break down complex problems into small bricks
- **reproducible** trace back analysis to well-defined breakpoints
- **experimental** trial-and-error is permitted and easier to implement
- **collaborative** agreeing on sensible choices enables peaceful and fruitful collaborations

The *Scikit-learn* guideline

"Improving the documentation is no less important than improving the library itself."

2. The ML Mindset

Validation and test sets

Table of contents

Dealing with Data

A generic beginning of a project

Let $\mathcal{D}_N = \{(\vec{x}_i, \vec{y}_i) \mid \mathbb{K}^p \ni \vec{x}_i \sim \mathcal{P}(X),\ \mathbb{K}^q \ni \vec{y}_i \sim \mathcal{P}(Y) \quad \forall i = 1, 2, \dots, N\}$:

$$\boxed{\mathcal{D}_N}$$

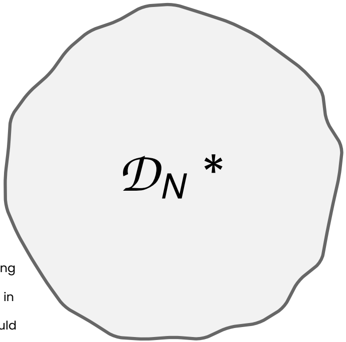
Some definitions to start:

- $\vec{x}$ are called **features** / exogenous variables / regressors / predictors / explanatory variables
- $\vec{y}$ are called **labels** / endogenous variables / regressands / targets / explained variables

Dealing with Data

A generic beginning of a project

Let $\mathcal{D}_N = \{(\vec{x}_i, \vec{y}_i) \mid \mathbb{K}^p \ni \vec{x}_i \sim \mathcal{P}(X),\ \mathbb{K}^q \ni \vec{y}_i \sim \mathcal{P}(Y) \quad \forall i = 1, 2, \dots, N\}$:



* Imagine if an ordering "by label" y_i was left in the dataset: what would happen in the following?

- **shuffle** the **dataset** to avoid **biases**: any possible ordering of the data should **never interfere**

Dealing with Data

A generic beginning of a project

Let $\mathcal{D}_N = \{(\vec{x}_i, \vec{y}_i) \mid \mathbb{K}^p \ni \vec{x}_i \sim \mathcal{P}(X),\ \mathbb{K}^q \ni \vec{y}_i \sim \mathcal{P}(Y) \quad \forall i = 1, 2, \dots, N\}$:

$$\boxed{\mathcal{D}_N}$$

- **shuffle** the **dataset** to avoid **biases**: any possible ordering of the data should **never interfere**

Dealing with Data

A generic beginning of a project

Let $\mathcal{D}_N = \{(\vec{x}_i, \vec{y}_i) \mid \mathbb{K}^p \ni \vec{x}_i \sim \mathcal{P}(X),\ \mathbb{K}^q \ni \vec{y}_i \sim \mathcal{P}(Y) \quad \forall i = 1, 2, \dots, N\}$:

$\mathcal{D}_N^{(\mathrm{dev})}$ $\mathcal{D}_N^{(\mathrm{test})}$

The number of samples to leave in the splits is highly dependent on the size of the dataset, type of task, computing resources, regularity of the data, etc…Traditionally, smaller datasets may require $\sim 80\,\%$ of training data. However, **Big Data** may take up to $99\,\%$ of training data, as the test set will remain statistically relevant.

(e.g. see Andrew Ng (2019))

- **shuffle** the **dataset** to avoid **biases**: any possible ordering of the data should **never interfere**
- prepare a **test set** and "hide" it until your final evaluation

Data leakage

The **test set** should be **randomly** and **independently** chosen to represent "real-world" data. It must **never** come into contact with **training** procedures.

Dealing with Data

A generic beginning of a project

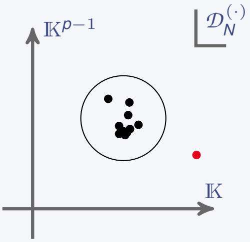
Let $\mathcal{D}_N = \{(\vec{x}_i, \vec{y}_i) \mid \mathbb{K}^p \ni \vec{x}_i \sim \mathcal{P}(X),\ \mathbb{K}^q \ni \vec{y}_i \sim \mathcal{P}(Y) \quad \forall i = 1, 2, \dots, N\}$:

$\mathcal{D}_N^{(\text{dev})}$ | $\mathcal{D}_N^{(\text{test})}$

The number of samples to leave in the splits is highly dependent on the size of the dataset, type of task, computing resources, regularity of the data, etc...Traditionally, smaller datasets may require $\sim 80\,\%$ of training data. However, **Big Data** may take up to $99\,\%$ of training data, as the test set will remain statistically relevant.

(e.g. see Andrew Ng (2019))

Food for thought... Outliers?



1. you have some **pictures**, and outliers are overexposed samples you would get rid of anyway
2. you analyse **financial** data where stock returns are capped to a given value
3. you are building a **cybersecurity** defence and outliers are attacked data
4. you have **scientific** data, and you are trying to derive an analytical formula using insights from ML

Dealing with Data

A generic beginning of a project

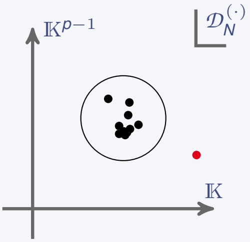
Let $\mathcal{D}_N = \{(\vec{x}_i, \vec{y}_i) \mid \mathbb{K}^p \ni \vec{x}_i \sim \mathcal{P}(X),\ \mathbb{K}^q \ni \vec{y}_i \sim \mathcal{P}(Y) \quad \forall i = 1, 2, \dots, N\}$:

$\mathcal{D}_N^{(\mathrm{dev})}$ $\mathcal{D}_N^{(\mathrm{test})}$

The number of samples to leave in the splits is highly dependent on the size of the dataset, type of task, computing resources, regularity of the data, etc…Traditionally, smaller datasets may require $\sim 80\,\%$ of training data. However, **Big Data** may take up to $99\,\%$ of training data, as the test set will remain statistically relevant.

(e.g. see Andrew Ng (2019))

Food for thought… Outliers?



1. you have some **pictures**, and outliers are overexposed samples you would get rid of anyway
2. you analyse **financial** data where stock returns are capped to a given value
3. you are building a **cybersecurity** defence and outliers are attacked data
4. you have **scientific** data, and you are trying to derive an analytical formula using insights from ML

1. remove them everywhere (they do not represent real-world data) 2. leave them everywhere / move them to $\mathcal{D}_N^{(\mathrm{test})}$ and cap the value 3. move them to $\mathcal{D}_N^{(\mathrm{test})}$ (grasp a model of easy cases to predict complex ones) 4. move them to $\mathcal{D}_N^{(\mathrm{test})}$ (the model should be able to predict them anyway)

Dealing with Data

The need of self-evaluation

$$\mathcal{D}_N^{(\mathrm{dev})}$$

Let us suppose:

- good **exploratory data analysis**
- model **dev. and training** on $\mathcal{D}_N^{(dev)}$
- **no bias** / **sensible** choices
- **good** performance
- in general... **nothing strange**

Dealing with Data
The need of self-evaluation

$\mathcal{D}_N^{(\mathrm{dev})}$

$\mathcal{D}_N^{(\mathrm{test})}$

Let us suppose:

- good **exploratory data analysis**
- model **dev. and training** on $\mathcal{D}_N^{(dev)}$
- **no bias** / **sensible** choices
- **good** performance
- in general... **nothing strange**

You get:

- **Bad generalisation** performance
- **Biased/unbalanced** results

Dealing with Data
The need of self-evaluation

$\mathcal{D}_N^{(\mathrm{dev})}$

$\mathcal{D}_N^{(\mathrm{test})}$

Let us suppose:

- good **exploratory data analysis**
- model **dev. and training** on $\mathcal{D}_N^{(dev)}$
- **no bias** / **sensible** choices
- **good** performance
- in general... **nothing strange**

You get:

- **Bad generalisation** performance
- **Biased/unbalanced** results

Most probably, the model **overfits** (see later) $\mathcal{D}_N^{(\mathrm{dev})}$ and is unable to **statistically** represent new data!

Dealing with Data
The need of self-evaluation

$\mathcal{D}_N^{(\mathrm{dev})}$

$\mathcal{D}_N^{(\mathrm{test})}$

Let us suppose:

- good **exploratory data analysis**
- model **dev. and training** on $\mathcal{D}_N^{(dev)}$
- **no bias** / **sensible** choices
- **good** performance
- in general... **nothing strange**

You get:

- **Bad generalisation** performance
- **Biased/unbalanced** results

Most probably, the model **overfits** (see later) $\mathcal{D}_N^{(\mathrm{dev})}$ and is unable to **statistically** represent new data! In other words, we need to **evaluate** the model before deploying it, or it will be, in general, a catastrophe!

Validation

The importance of choosing a validation set

Holdout validation

$$\mathcal{D}_N^{(\text{dev})}$$

Validation

The importance of choosing a validation set

Holdout validation

$\boxed{\quad\mathcal{D}_N^{(\text{train})}\quad}\ \boxed{\mathcal{D}_N^{(\text{val})}}$

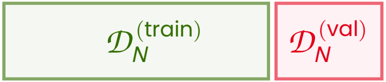
- build $\mathcal{D}_N^{(\text{val})} \subset \mathcal{D}_N^{(\text{dev})}$ and $\mathcal{D}_N^{(\text{train})} = \mathcal{D}_N^{(\text{dev})} \setminus \mathcal{D}_N^{(\text{val})}$ **once**
- computing time-**friendly**
- good for (very) **large datasets**
- **easy** to implement, **easy** to use usually,

no boilerplate code in Pytorch Lightning, Keras, Hugging Face, etc...

Validation

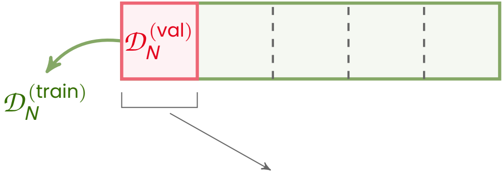
The importance of choosing a validation set

Holdout validation



- build $\mathcal{D}_N^{(\text{val})} \subset \mathcal{D}_N^{(\text{dev})}$ and $\mathcal{D}_N^{(\text{train})} = \mathcal{D}_N^{(\text{dev})} \setminus \mathcal{D}_N^{(\text{val})}$ **once**
- computing time-**friendly**
- good for (very) **large datasets**
- **easy** to implement, **easy** to use usually, no boilerplate code in Pytorch Lightning, Keras, Hugging Face, etc...

Cross validation (K-fold)



K "folds" (chunks) of data

- more **robust** estimator
- first **insight** into **uncertainties**
- **time consuming** for **large** datasets
- might need some **coding** yes, `scikit-learn` has a good implementation! Do not worry!

Validation

Validation error and how to use it

Let $\mathrm{dist}(y, \widehat{y})$ be a metric **distance** between **ground truth** y and its **prediction** $\widehat{y}$ (e.g. mean squared error, cross entropy, etc.) [N.B.: what said for a scalar y can be said for $\vec{y}$] and compute *error* $\mathcal{E}$ on $\mathcal{D}^{(\mathrm{val})}$.

Validation

Validation error and how to use it

Let $\mathrm{dist}(y, \widehat{y})$ be a metric **distance** between **ground truth** y and its **prediction** $\widehat{y}$ (e.g. mean squared error, cross entropy, etc.) [N.B.: what said for a scalar y can be said for $\vec{y}$] and compute *error* $\mathcal{E}$ on $\mathcal{D}^{(\mathrm{val})}$.

Holdout validation

$$\begin{aligned}\mathcal{E}_{\mathcal{D}_N^{(\mathrm{val})}}(y, \widehat{y}) &= \mathbb{E}_{\mathcal{D}_N^{(\mathrm{val})}}\left[\mathrm{dist}(y, \widehat{y})\right] \\ &= \frac{1}{m}\sum_{p=0}^{m-1} \mathrm{dist}(y_p, \widehat{y}_p).\end{aligned}$$

Validation

Validation error and how to use it

Let $\mathrm{dist}(y, \widehat{y})$ be a metric **distance** between **ground truth** y and its **prediction** $\widehat{y}$ (e.g. mean squared error, cross entropy, etc.) [N.B.: what said for a scalar y can be said for $\vec{y}$] and compute *error* $\mathcal{E}$ on $\mathcal{D}^{(\mathrm{val})}$.

Holdout validation

$$\mathcal{E}_{\mathcal{D}_N^{(\mathrm{val})}}(y, \widehat{y}) = \mathbb{E}_{\mathcal{D}_N^{(\mathrm{val})}}\left[\mathrm{dist}(y, \widehat{y})\right] = \frac{1}{m} \sum_{p=0}^{m-1} \mathrm{dist}(y_p, \widehat{y}_p).$$

Cross validation (K-fold)

$$\mathcal{E}^{(K)}_{\mathcal{D}_N^{(\mathrm{val})}}(y, \widehat{y}) = \mathbb{E}_{\text{K-folds}}\left[\mathbb{E}_{\mathcal{D}_N^{(\mathrm{val})}}\left[\mathrm{dist}(y, \widehat{y})\right]\right] = \frac{1}{K} \sum_{i=0}^{K-1} \frac{1}{m_i} \sum_{p=0}^{m_i-1} \mathrm{dist}(y_p^{[i]}, \widehat{y}_p^{[i]}).$$

Notation: let $n = \left|\mathcal{D}_N^{(\mathrm{dev})}\right|$, then $\lfloor \frac{n}{K} \rfloor \leq m_i \leq \lfloor \frac{n}{K} \rfloor + 1$ is the size of the i-th validation fold $\Rightarrow n - \frac{n}{K} = \frac{K-1}{K} n$ is the size of the remaining set.

The particular case $K = n$ is called "leave-one-out" validation.

Validation

Validation error and how to use it

Let $\mathrm{dist}(y, \hat{y})$ be a metric **distance** between **ground truth** y and its **prediction** $\hat{y}$ (e.g. mean squared error, cross entropy, etc.) [N.B.: what said for a scalar y can be said for $\vec{y}$] and compute *error* $\mathcal{E}$ on $\mathcal{D}^{(\mathrm{val})}$.

Holdout validation

$$\mathcal{E}_{\mathcal{D}_N^{(\mathrm{val})}}(y, \hat{y}) = \mathbb{E}_{\mathcal{D}_N^{(\mathrm{val})}}\left[\mathrm{dist}(y, \hat{y})\right] = \frac{1}{m}\sum_{p=0}^{m-1} \mathrm{dist}(y_p, \hat{y}_p).$$

Cross validation (K-fold)

$$\mathcal{E}^{(K)}_{\mathcal{D}_N^{(\mathrm{val})}}(y, \hat{y}) = \mathbb{E}_{\text{K-folds}}\left[\mathbb{E}_{\mathcal{D}_N^{(\mathrm{val})}}\left[\mathrm{dist}(y, \hat{y})\right]\right] = \frac{1}{K}\sum_{i=0}^{K-1}\frac{1}{m_i}\sum_{p=0}^{m_i-1} \mathrm{dist}(y_p^{[i]}, \hat{y}_p^{[i]}).$$

$\mathcal{E}_{\mathcal{D}_N^{(\mathrm{val})}}(y, \hat{y})$ can be used as **estimate** of the **prediction error**

Notation: let $n = \left|\mathcal{D}_N^{(\mathrm{dev})}\right|$, then $\left\lfloor \frac{n}{K} \right\rfloor \le m_i \le \left\lfloor \frac{n}{K} \right\rfloor + 1$ is the size of the i-th validation fold $\Rightarrow n - \frac{n}{K} = \frac{K-1}{K} n$ is the size of the remaining set.

The particular case $K = n$ is called “leave-one-out” validation.

Validation

Validation error and how to use it

Let $\mathrm{dist}(y, \hat{y})$ be a metric **distance** between **ground truth** y and its **prediction** $\hat{y}$ (e.g. mean squared error, cross entropy, etc.) [N.B.: what said for a scalar y can be said for $\vec{y}$] and compute *error* $\mathcal{E}$ on $\mathcal{D}^{(\mathrm{val})}$.

Holdout validation

$$\mathcal{E}_{\mathcal{D}_N^{(\mathrm{val})}}(y, \hat{y}) = \mathbb{E}_{\mathcal{D}_N^{(\mathrm{val})}}\left[\mathrm{dist}(y, \hat{y})\right] = \frac{1}{m} \sum_{p=0}^{m-1} \mathrm{dist}(y_p, \hat{y}_p).$$

$\mathcal{E}_{\mathcal{D}_N^{(\mathrm{val})}}(y, \hat{y})$ can be used as **estimate** of the **prediction error**

Cross validation (K-fold)

$$\mathcal{E}^{(K)}_{\mathcal{D}_N^{(\mathrm{val})}}(y, \hat{y}) = \mathbb{E}_{\text{K-folds}}\left[\mathbb{E}_{\mathcal{D}_N^{(\mathrm{val})}}\left[\mathrm{dist}(y, \hat{y})\right]\right] = \frac{1}{K} \sum_{i=0}^{K-1} \frac{1}{m_i} \sum_{p=0}^{m_i-1} \mathrm{dist}(y_p^{[i]}, \hat{y}_p^{[i]}).$$

More complicated than that...The choice depends on K. Moreover, the whole $\mathcal{D}_N^{(\mathrm{dev})}$ is used!

Notation: let $n = \left|\mathcal{D}_N^{(\mathrm{dev})}\right|$, then $\left\lfloor \frac{n}{K} \right\rfloor \leq m_i \leq \left\lfloor \frac{n}{K} \right\rfloor + 1$ is the size of the i-th validation fold $\Rightarrow n - \frac{n}{K} = \frac{K-1}{K} n$ is the size of the remaining set.

The particular case $K = n$ is called "leave-one-out" validation.

Validation

Validation error and how to use it

Let $\mathrm{dist}(y, \widehat{y})$ be a metric **distance** between **ground truth** y and its **prediction** $\widehat{y}$ (e.g. mean squared error, cross entropy, etc.) [N.B.: what said for a scalar y can be said for $\vec{y}$] and compute *error* $\mathcal{E}$ on $\mathcal{D}^{(\mathrm{val})}$.

Holdout validation

$$\mathcal{E}_{\mathcal{D}_N^{(\mathrm{val})}}(y, \widehat{y}) = \mathbb{E}_{\mathcal{D}_N^{(\mathrm{val})}}\left[\mathrm{dist}(y, \widehat{y})\right] = \frac{1}{m} \sum_{p=0}^{m-1} \mathrm{dist}(y_p, \widehat{y}_p).$$

$\mathcal{E}_{\mathcal{D}_N^{(\mathrm{val})}}(y, \widehat{y})$ can be used as **estimate** of the **prediction error**

Cross validation (K-fold)

$$\mathcal{E}^{(K)}_{\mathcal{D}_N^{(\mathrm{val})}}(y, \widehat{y}) = \mathbb{E}_{\text{K-folds}}\left[\mathbb{E}_{\mathcal{D}_N^{(\mathrm{val})}}\left[\mathrm{dist}(y, \widehat{y})\right]\right] = \frac{1}{K} \sum_{i=0}^{K-1} \frac{1}{m_i} \sum_{p=0}^{m_i-1} \mathrm{dist}(y_p^{[i]}, \widehat{y}_p^{[i]}).$$

More complicated than that…The choice depends on K. Moreover, the whole $\mathcal{D}_N^{(\mathrm{dev})}$ is used!

We will come back to this later…

Notation: let $n = \left|\mathcal{D}_N^{(\mathrm{dev})}\right|$, then $\left\lfloor \frac{n}{K} \right\rfloor \leq m_i \leq \left\lfloor \frac{n}{K} \right\rfloor + 1$ is the size of the i-th validation fold $\Rightarrow n - \frac{n}{K} = \frac{K-1}{K} n$ is the size of the remaining set.

The particular case $K = n$ is called “leave-one-out” validation.

Validation

Parameters vs hyperparameters

Let $\mathrm{M} = \left\{ f^{(n)} \;\middle|\; n = 1,\, 2,\, \ldots \right\}$ set of *models* (e.g. linear model, support vector machine, decision tree, neural network, etc.)

Validation

Parameters vs hyperparameters

Let M $= \left\{ f^{(n)} \;\middle|\; n = 1,\, 2,\, \dots \right\}$ set of *models* (e.g. linear model, support vector machine, decision tree, neural network, etc.)

Each $f^{(n)}$ has two sets of dependencies $\Rightarrow f^{(n)} = f^{(n)}(\Theta\, ;\, \Omega)$:

- Θ is the set of **parameters** (i.e. the *weights* of the model $\to y = \vec{\beta} \cdot \vec{x}$)
- Ω is the set of **hyperparameters** (i.e. *constraints* of the model $\to y = \vec{\beta} \cdot \vec{x} + \lambda\, \vec{\beta} \cdot \vec{\beta}$)

Validation

Parameters vs hyperparameters

Let $\mathrm{M} = \{ f^{(n)} \mid n = 1, 2, \dots \}$ set of *models* (e.g. linear model, support vector machine, decision tree, neural network, etc.)

Each $f^{(n)}$ has two sets of dependencies $\Rightarrow f^{(n)} = f^{(n)}(\Theta\,;\Omega)$:

- Θ is the set of **parameters** (i.e. the *weights* of the model $\to y = \vec{\beta} \cdot \vec{x}$)
- Ω is the set of **hyperparameters** (i.e. *constraints* of the model $\to y = \vec{\beta} \cdot \vec{x} + \lambda\, \vec{\beta} \cdot \vec{\beta}$)

$\mathcal{D}_N^{(\text{train})}$ | $\mathcal{D}_N^{(\text{val})}$

you *train* the parameters Θ of a model $f^{(n)}$

you *validate* the hyperparameters Ω of model $f^{(n)}$ vs $f^{(m)}$ $(n \neq m)$

Validation

Parameters vs hyperparameters

Let $M = \{ f^{(n)} \mid n = 1, 2, \dots \}$ set of *models* (e.g. linear model, support vector machine, decision tree, neural network, etc.)

Each $f^{(n)}$ has two sets of dependencies $\Rightarrow f^{(n)} = f^{(n)}(\Theta\,;\Omega)$:

- Θ is the set of **parameters** (i.e. the *weights* of the model $\rightarrow y = \vec{\beta} \cdot \vec{x}$)
- Ω is the set of **hyperparameters** (i.e. *constraints* of the model $\rightarrow y = \vec{\beta} \cdot \vec{x} + \lambda\, \vec{\beta} \cdot \vec{\beta}$)

$\mathcal{D}_N^{(\text{train})}$	$\mathcal{D}_N^{(\text{val})}$
you *train* the parameters Θ of a model $f^{(n)}$	you *validate* the hyperparameters Ω of model $f^{(n)}$ vs $f^{(m)}$ $(n \neq m)$
Loss functions	Metrics
mean squared error, mean absolute error, cross entropy, KL divergence, style transfer loss, etc.	mean squared error, accuracy, precision, recall, F-score, Rand index, etc.

Validation

Parameters vs hyperparameters

Let M $= \{ f^{(n)} \mid n = 1, 2, \dots \}$ set of *models* (e.g. linear model, support vector machine, decision tree, neural network, etc.)

Each $f^{(n)}$ has two sets of dependencies $\Rightarrow f^{(n)} = f^{(n)}(\Theta\,;\Omega)$:

- Θ is the set of **parameters** (i.e. the *weights* of the model $\rightarrow y = \vec{\beta} \cdot \vec{x}$)
- Ω is the set of **hyperparameters** (i.e. *constraints* of the model $\rightarrow y = \vec{\beta} \cdot \vec{x} + \lambda\, \vec{\beta} \cdot \vec{\beta}$)

$\mathcal{D}_N^{(\text{train})}$ | $\mathcal{D}_N^{(\text{val})}$

Training

you *train* the parameters Θ of a model $f^{(n)}$

Loss functions

mean squared error, mean absolute error, cross entropy, KL divergence, style transfer loss, etc.

you *validate* the hyperparameters Ω of model $f^{(n)}$ vs $f^{(m)}$ $(n \neq m)$

Metrics

mean squared error, accuracy, precision, recall, F-score, Rand index, etc.

Model selection (hyperparameter tuning)

Validation

Parameters vs hyperparameters

Let M $= \{ f^{(n)} \mid n = 1, 2, \ldots \}$ set of *models* (e.g. linear model, support vector machine, decision tree, neural network, etc.)

Each $f^{(n)}$ has two sets of dependencies $\Rightarrow f^{(n)} = f^{(n)}(\Theta\,;\Omega)$:

- Θ is the set of **parameters** (i.e. the *weights* of the model $\to y = \vec{\beta} \cdot \vec{x}$)
- Ω is the set of **hyperparameters** (i.e. *constraints* of the model $\to y = \vec{\beta} \cdot \vec{x} + \lambda\, \vec{\beta} \cdot \vec{\beta}$)

$\mathcal{D}_N^{(\text{train})}$ | $\mathcal{D}_N^{(\text{val})}$

Training

you *train* the parameters Θ of a model $f^{(n)}$

Loss functions

mean squared error, mean absolute error, cross entropy, KL divergence, style transfer loss, etc.

see later...

you *validate* the hyperparameters Ω of model $f^{(n)}$ vs $f^{(m)}$ $(n \neq m)$

Metrics

mean squared error, accuracy, precision, recall, F-score, Rand index, etc.

Model selection

(hyperparameter tuning)

Validation metrics

Evaluating a model

The pletora of evaluation functions at our disposal strongly depends on the *task*!

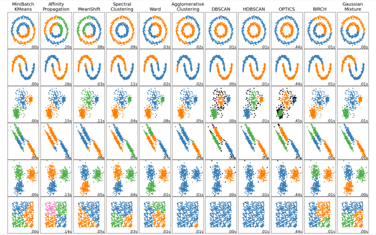
Validation metrics

Evaluating a model

The pletora of evaluation functions at our disposal strongly depends on the *task*!

Clustering task – Purity

Let $\mathrm{K} = \{k_1, k_2, \dots, k_P\}$ the set of clusters, and $\mathrm{C} = \{c_1, c_2, \dots, c_L\}$ the set of classes of N points:



$$\mathrm{P}(\mathrm{K}, \mathrm{C}) = \frac{1}{N} \sum_{p=1}^{P} \max_{\ell=1,\dots,L} |k_p \cap c_\ell|$$

Purity is the normalised *mode* of the clusters. What happens if $K = N$?

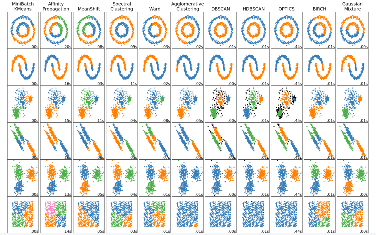
Validation metrics

Evaluating a model

The pletora of evaluation functions at our disposal strongly depends on the *task*!

Clustering task – Normalized Mutual Information

Let $\mathrm{K} = \{k_1, k_2, \dots, k_P\}$ the set of clusters, and $\mathrm{C} = \{c_1, c_2, \dots, c_L\}$ the set of classes of N points:



$$\mathrm{NMI}(\mathrm{K}, \mathrm{C}) = 2 \frac{\mathrm{I}(\mathrm{K}, \mathrm{C})}{\mathrm{H}(\mathrm{K}) + \mathrm{H}(\mathrm{C})}$$

where $\mathrm{H}(\cdot) = -\mathbb{E}_{\mathcal{P}(\cdot)}\left[\ln \mathcal{P}(\cdot)\right]$ and

$$\mathrm{I}(\mathrm{K}, \mathrm{C}) = \mathbb{E}_{\mathcal{P}(\mathrm{K} \cap \mathrm{C})}\left[\ln \frac{\mathcal{P}(\mathrm{K} \cap \mathrm{C})}{\mathcal{P}(\mathrm{K})\mathcal{P}(\mathrm{C})}\right]$$

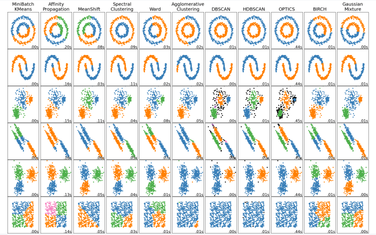
Validation metrics

Evaluating a model

The pletora of evaluation functions at our disposal strongly depends on the *task*!

Clustering task – Rand Index

Let K $= \{k_1, k_2, \dots, k_P\}$ the set of clusters, and C $= \{c_1, c_2, \dots, c_L\}$ the set of classes of N points:



Consider the $N(N-1)/2$ couples:

- TP $\to$ similar objects in the same clusters
- TN $\to$ different objects in different clusters
- FP $\to$ *different* objects in the same clusters
- FN $\to$ *similar* objects in different clusters

$$\mathrm{RI}(\mathrm{K}, \mathrm{C}) = \frac{TP + TN}{TP + TN + FP + FN}$$

Some of you might recognise the classification "accuracy" in this definition: even though the idea is not far, this is different!

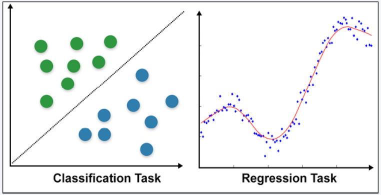
Validation metrics

Evaluating a model

The pletora of evaluation functions at our disposal strongly depends on the *task*!

Regression task – p-norm

Let $y_i \in \mathbb{R}$ be the *ground truth*, and $\widehat{y}_i \in \mathbb{R}$ the *prediction* of the i-th sample $(i = 1, 2, \dots, N)$.



$$||y - \widehat{y}||_p = \left(\sum_{i=1}^{N} (y_i - \widehat{y}_i)^p \right)^{\frac{1}{p}}$$

Specific cases:

- $p = 0 \rightarrow$ no. of non-zero elements
- $p = \infty \rightarrow$ 📖 *can you compute it?*

Validation metrics

Evaluating a model

The pletora of evaluation functions at our disposal strongly depends on the *task*!

Classification task – Accuracy, precision, recall and confusion matrix

Let $C_i \in \{0, 1\}$ be the *ground truth*, and $\widehat{C}_i \in \{0, 1\}$ the *prediction* of the i-th sample $(i = 1, 2, \dots, N)$.*

ground truth \ prediction	$\widehat{C} = 1$	$\widehat{C} = 0$
$C = 1$	TP	FN
$C = 0$	FP	TN

confusion matrix

Consider the possibilities:

- TP $\to C = \widehat{C} = 1$
- TN $\to C = \widehat{C} = 0$
- FP $\to C = 0$ and $\widehat{C} = 1$ (type I)
- FN $\to C = 1$ and $\widehat{C} = 0$ (type II)

*Class assignements are based on the *probability* of belonging to a class, that is $\widehat{C} = 1 \Leftrightarrow \mathcal{P}(\widehat{Y} = 1) > \eta$, where η is an arbitrary threshold.

Validation metrics

Evaluating a model

The pletora of evaluation functions at our disposal strongly depends on the *task*!

Classification task – Accuracy, precision, recall and confusion matrix

Let $C_i \in \{0, 1\}$ be the *ground truth*, and $\widehat{C}_i \in \{0, 1\}$ the *prediction* of the i-th sample $(i = 1, 2, \dots, N)$.*

ground truth \ prediction	$\widehat{C} = 1$	$\widehat{C} = 0$
$C = 1$	TP	FN
$C = 0$	FP	TN

confusion matrix

$$\text{accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

$$\text{precision} = \frac{TP}{TP + FP}$$

$$\text{recall} = \frac{TP}{TP + FN}$$

*Class assignements are based on the *probability* of belonging to a class, that is $\widehat{C} = 1 \Leftrightarrow \mathcal{P}(\widehat{Y} = 1) > \eta$, where η is an arbitrary threshold.

Validation metrics

Evaluating a model

The pletora of evaluation functions at our disposal strongly depends on the *task*!

Classification task – Accuracy, precision, recall and confusion matrix

Let $C_i \in \{0, 1\}$ be the *ground truth*, and $\widehat{C}_i \in \{0, 1\}$ the *prediction* of the i–th sample ($i = 1, 2, \dots, N$).*

ground truth \ prediction	$\widehat{C} = 1$	$\widehat{C} = 0$
$C = 1$	TP	FN
$C = 0$	FP	TN

confusion matrix

$$\mathrm{F}_\beta = (1 + \beta^2) \frac{\text{precision} \cdot \text{recall}}{\beta^2 \cdot \text{precision} + \text{recall}}$$

Imagine you are testing the presence of an infection in the population: would you prefer a highly *precise* test or go for higher *recall*? Why?

*Class assignements are based on the *probability* of belonging to a class, that is $\widehat{C} = 1 \Leftrightarrow \mathcal{P}(\widehat{Y} = 1) > \eta$, where η is an arbitrary threshold.

Validation metrics

Evaluating a model

The pletora of evaluation functions at our disposal strongly depends on the *task*!

Classification task – Accuracy, precision, recall and confusion matrix

Let $C_i \in \{0, 1\}$ be the *ground truth*, and $\widehat{C}_i \in \{0, 1\}$ the *prediction* of the i-th sample ($i = 1, 2, \dots, N$).*

ground truth \ prediction	$\widehat{C} = 1$	$\widehat{C} = 0$
$C = 1$	TP	FN
$C = 0$	FP	TN

confusion matrix

$$\text{sensitivity} = \frac{TP}{TP + FN} \quad (= \text{recall})$$

$$\text{specificity} = \frac{TN}{TN + FP}$$

*Class assignements are based on the *probability* of belonging to a class, that is $\widehat{C} = 1 \Leftrightarrow \mathcal{P}(\widehat{Y} = 1) > \eta$, where η is an arbitrary threshold.

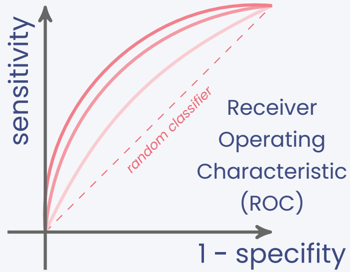
Validation metrics

Evaluating a model

The pletora of evaluation functions at our disposal strongly depends on the *task*!

Classification task – Accuracy, precision, recall and confusion matrix

Let $C_i \in \{0, 1\}$ be the *ground truth*, and $\widehat{C}_i \in \{0, 1\}$ the *prediction* of the i-th sample $(i = 1, 2, \ldots, N)$.*



$$\text{sensitivity} = \frac{TP}{TP+FN} \text{ (= recall)}$$

$$\text{specificity} = \frac{TN}{TN+FP}$$

All metrics depend on the **decision threshold** $M = M(\eta)$. What if we use it as a parameter? We can use the **Area Under the Curve** (AUC) to evaluate the classifier!

*Class assignements are based on the *probability* of belonging to a class, that is $\widehat{C} = 1 \Leftrightarrow \mathcal{P}(\widehat{Y} = 1) > \eta$, where η is an arbitrary threshold.

Validation metrics

Evaluating a model

The pletora of evaluation functions at our disposal strongly depends on the *task*!

Binary classification

The output of the model for the i–th sample $\hat{y}_i \in \mathbb{R}^K$. We can use a **sigmoid** normalisation

$$\hat{y}'_i = \frac{1}{1 + e^{-\hat{y}_i}} \in [0, 1]$$

to **interpret** the result as a probability of belonging to the positive class. In other words, the class assignment is:

$$\hat{C} = 1 \quad \Leftrightarrow \quad \mathrm{P}\,(Y_i = 1) = \hat{y}'_i > \eta,$$

where η is an arbitrary threshold (e.g. $\eta = 0.5$).

Validation metrics

Evaluating a model

The pletora of evaluation functions at our disposal strongly depends on the *task*!

Multiclass classification

The output of the model for the i-th sample $\hat{y}_i \in \mathbb{R}^K$. We can use a **softmax** normalisation

$$\mathrm{P}\left(Y_i = k\right) = \hat{y}_i'^{\,(k)} = \frac{e^{\hat{y}_i^{(k)}}}{\sum_{\ell=1}^{K} e^{\hat{y}_i^{(\ell)}}} \quad \text{s.t.} \quad \sum_{k=1}^{K} \hat{y}_i'^{\,(k)} = 1$$

to **interpret** the result as a probability of belonging to the k-th class. In other words, the class assignment is:

$$\widehat{C}_i = \underset{k=1,\dots,K}{\arg\max}\, \hat{y}_i'^{\,(k)} = \underset{k=1,\dots,K}{\arg\max}\, \hat{y}_i^{(k)}$$

HOMEWORK: prove that **softmax** for binary classification is a **sigmoid**.

Validation metrics

Evaluating a model

The pletora of evaluation functions at our disposal strongly depends on the *task*!

Target encoding

Compare **softmax**-activated layer classes $\Rightarrow$ use **one-hot encoding**:

$$\mathrm{P}\left(Y_i = k\right) = 1 \quad \Rightarrow \quad \vec{y}_i = \left(0, \dots, 0, \quad \underbrace{1,}_{k\text{-th position}} \quad 0, \dots, 0, \right) .$$

```python
import numpy as np


def one_hot_encoding(y: np.ndarray, n_classes: int) -> np.ndarray:
    n_samples = y.shape[0]  # no. of samples in y
    one_hot = np.zeros((n_samples, n_classes))  # no. of classes
    one_hot[np.arange(n_samples), y] = 1  # set 1 depending on ground truth
    return one_hot
```

This enables comparisons $\vec{y}$ vs $\vec{\hat{y}}$ after **softmax** by comparing “bits” of information contained in the vectors.

2. The ML Mindset

The variance vs bias trade-off

Table of contents

The ML Mindset
Variance and bias

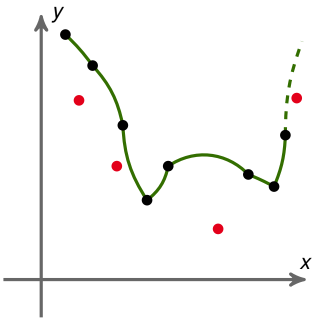
Objective

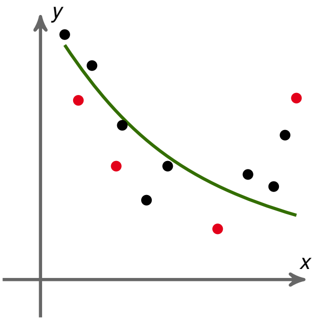
Build a model which **learns** to **predict** (*generalise*) meaningful data

The ML Mindset
Variance and bias

Objective

Build a model which **learns** to **predict** (*generalise*) meaningful data



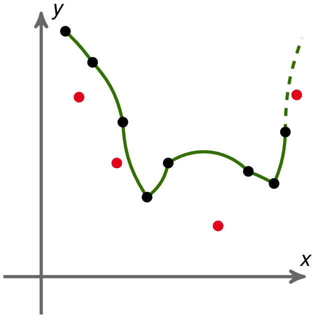


The ML Mindset
Variance and bias

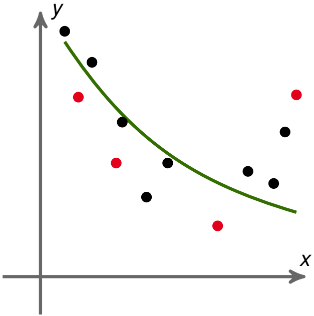
Objective

Build a model which **learns** to **predict** (*generalise*) meaningful data

Less assumptions more parameters



More assumptions less parameters

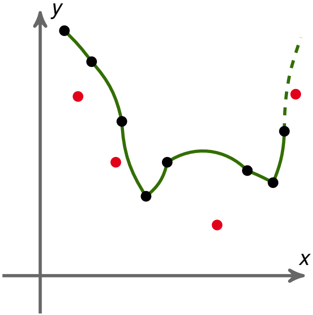


The ML Mindset
Variance and bias

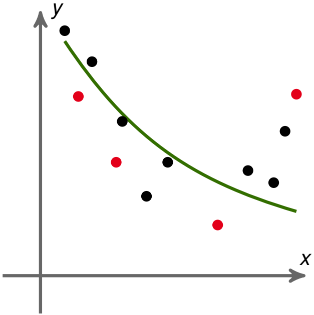
Objective

Build a model which **learns** to **predict** (*generalise*) meaningful data

OVERFITTING MODEL



UNDERFITTING MODEL

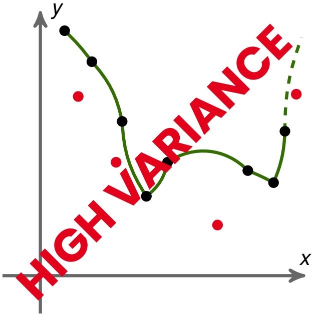


The ML Mindset
Variance and bias

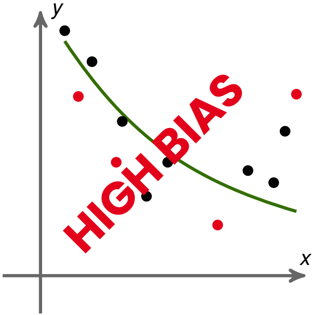
Objective

Build a model which **learns** to **predict** (*generalise*) meaningful data

OVERFITTING MODEL



UNDERFITTING MODEL



The Variance vs Bias Trade-off

Variance and bias in prediction

Without loss of generality, define the **true value**: **N.B.**: $\mathrm{Var}(X) = \mathbb{E}[X^2] - \mathbb{E}[X]^2$

$$y = f(\vec{x}) + \varepsilon \in \mathbb{R}, \quad \vec{x} \in \mathbb{R}^p, \quad \mathbb{E}_{(\vec{x},y)}\left[\varepsilon\right] = 0, \quad \mathrm{Var}_{(\vec{x},y)}(\varepsilon) = \mathbb{E}_{(\vec{x},y)}\left[\varepsilon^2\right] = \sigma^2.$$

The Variance vs Bias Trade-off

Variance and bias in prediction

Without loss of generality, define the **true value**: **N.B.**: $\mathrm{Var}(X) = \mathbb{E}[X^2] - \mathbb{E}[X]^2$

$$y = f(\vec{x}) + \varepsilon \in \mathbb{R}, \quad \vec{x} \in \mathbb{R}^p, \quad \mathbb{E}_{(\vec{x},y)}\left[\varepsilon\right] = 0, \quad \mathrm{Var}_{(\vec{x},y)}(\varepsilon) = \mathbb{E}_{(\vec{x},y)}\left[\varepsilon^2\right] = \sigma^2.$$

Build the **model** to predict a **label** (*"supervised"* see later)

$$\widehat{f}_{\mathcal{D}_N} : \mathbb{R}^p \to \mathbb{R}$$

using data in $\mathcal{D}_N = \left\{ (\vec{x}^{(i)}, y^{(i)}) \;\middle|\; \vec{x}^{(i)} \in \mathbb{R}^p,\ y^{(i)} \in \mathbb{R} \quad \forall i = 1, 2, \dots, N \right\}$.

The Variance vs Bias Trade-off

Variance and bias in prediction

Without loss of generality, define the **true value**: **N.B.**: $\mathrm{Var}(X) = \mathbb{E}[X^2] - \mathbb{E}[X]^2$

$$y = f(\vec{x}) + \varepsilon \in \mathbb{R}, \quad \vec{x} \in \mathbb{R}^p, \quad \mathbb{E}_{(\vec{x},y)}\left[\varepsilon\right] = 0, \quad \mathrm{Var}_{(\vec{x},y)}(\varepsilon) = \mathbb{E}_{(\vec{x},y)}\left[\varepsilon^2\right] = \sigma^2.$$

Build the **model** to predict a **label** (*"supervised"* see later)

$$\widehat{f}_{\mathcal{D}_N} : \mathbb{R}^p \to \mathbb{R}$$

using data in $\mathcal{D}_N = \left\{ (\vec{x}^{(i)}, y^{(i)}) \mid \vec{x}^{(i)} \in \mathbb{R}^p,\ y^{(i)} \in \mathbb{R} \quad \forall i = 1, 2, \dots, N \right\}$.

Generalisation error

Let $(\vec{x}', y')$ be an **unseen** pair, and compute the squared error from a *trained* model $\widehat{f}_{\mathcal{D}_N}$:

$$\mathbb{E}_{(\vec{x},y),\ \mathcal{D}_N}\left[\left(y' - \widehat{f}_{\mathcal{D}_N}(\vec{x}') \right)^2 \right]$$

We consider the *mean squared error* for simplicity, but the same holds for other kinds of generalisation error.

The Variance vs Bias Trade-off
Variance and bias in prediction

$$\mathbb{E}_{(\vec{x},y),\,\mathcal{D}_N}\left[\left(y' - \widehat{f}_{\mathcal{D}_N}(\vec{x}')\right)^2\right]$$

The Variance vs Bias Trade-off

Variance and bias in prediction

$$\mathbb{E}_{(\vec{x},y),\,\mathcal{D}_N}\left[\left(y' - \widehat{f}_{\mathcal{D}_N}(\vec{x}')\right)^2\right] = \mathbb{E}_{(\vec{x},y),\,\mathcal{D}_N}\left[\left(\varepsilon + f(\vec{x}') - \widehat{f}_{\mathcal{D}_N}(\vec{x}')\right)^2\right]$$

The Variance vs Bias Trade-off

Variance and bias in prediction

$$
\begin{aligned}
\mathbb{E}_{(\vec{x},y),\,\mathcal{D}_N}\left[\left(y' - \widehat{f}_{\mathcal{D}_N}(\vec{x}')\right)^2\right] &= \mathbb{E}_{(\vec{x},y),\,\mathcal{D}_N}\left[\left(\varepsilon + f(\vec{x}') - \widehat{f}_{\mathcal{D}_N}(\vec{x}')\right)^2\right] \\
&= \mathbb{E}_{(\vec{x},y)}\left[\varepsilon^2\right] + \mathbb{E}_{\mathcal{D}_N}\left[\left(f(\vec{x}') - \widehat{f}_{\mathcal{D}_N}(\vec{x}')\right)^2\right] + \mathbb{E}_{(\vec{x},y),\,\mathcal{D}_N}\left[2\,\varepsilon\left(f(\vec{x}') - \widehat{f}_{\mathcal{D}_N}(\vec{x}')\right)\right]
\end{aligned}
$$

The Variance vs Bias Trade-off

Variance and bias in prediction

$$
\begin{aligned}
\mathbb{E}_{(\vec{x},y),\,\mathcal{D}_N}\left[\left(y' - \hat{f}_{\mathcal{D}_N}(\vec{x}')\right)^2\right] &= \mathbb{E}_{(\vec{x},y),\,\mathcal{D}_N}\left[\left(\varepsilon + f(\vec{x}') - \hat{f}_{\mathcal{D}_N}(\vec{x}')\right)^2\right] \\
&= \mathbb{E}_{(\vec{x},y)}\left[\varepsilon^2\right] + \mathbb{E}_{\mathcal{D}_N}\left[\left(f(\vec{x}') - \hat{f}_{\mathcal{D}_N}(\vec{x}')\right)^2\right] + \mathbb{E}_{(\vec{x},y),\,\mathcal{D}_N}\left[2\,\varepsilon\left(f(\vec{x}') - \hat{f}_{\mathcal{D}_N}(\vec{x}')\right)\right] \\
&= \sigma^2 + \mathbb{E}_{\mathcal{D}_N}\left[\left(f(\vec{x}') - \hat{f}_{\mathcal{D}_N}(\vec{x}')\right)^2\right] + 2\,\mathbb{E}_{(\vec{x},y)}\left[\varepsilon\right]\mathbb{E}_{\mathcal{D}_N}\left[f(\vec{x}') - \hat{f}_{\mathcal{D}_N}(\vec{x}')\right]
\end{aligned}
$$

The Variance vs Bias Trade-off

Variance and bias in prediction

$$
\begin{aligned}
\mathbb{E}_{(\vec{x},y),\,\mathcal{D}_N}\left[\left(y' - \widehat{f}_{\mathcal{D}_N}(\vec{x}')\right)^2\right] &= \mathbb{E}_{(\vec{x},y),\,\mathcal{D}_N}\left[\left(\varepsilon + f(\vec{x}') - \widehat{f}_{\mathcal{D}_N}(\vec{x}')\right)^2\right] \\
&= \mathbb{E}_{(\vec{x},y)}\left[\varepsilon^2\right] + \mathbb{E}_{\mathcal{D}_N}\left[\left(f(\vec{x}') - \widehat{f}_{\mathcal{D}_N}(\vec{x}')\right)^2\right] + \mathbb{E}_{(\vec{x},y),\,\mathcal{D}_N}\left[2\,\varepsilon\left(f(\vec{x}') - \widehat{f}_{\mathcal{D}_N}(\vec{x}')\right)\right] \\
&= \sigma^2 + \mathbb{E}_{\mathcal{D}_N}\left[\left(f(\vec{x}') - \widehat{f}_{\mathcal{D}_N}(\vec{x}')\right)^2\right] + 2\,\cancelto{0}{\mathbb{E}_{(\vec{x},y)}\left[\varepsilon\right]}\,\mathbb{E}_{\mathcal{D}_N}\left[f(\vec{x}') - \widehat{f}_{\mathcal{D}_N}(\vec{x}')\right]
\end{aligned}
$$

The Variance vs Bias Trade-off

Variance and bias in prediction

$$
\begin{aligned}
\mathbb{E}_{(\vec{x},y),\,\mathcal{D}_N}\left[\left(y' - \widehat{f}_{\mathcal{D}_N}(\vec{x}')\right)^2\right] &= \mathbb{E}_{(\vec{x},y),\,\mathcal{D}_N}\left[\left(\varepsilon + f(\vec{x}') - \widehat{f}_{\mathcal{D}_N}(\vec{x}')\right)^2\right] \\
&= \mathbb{E}_{(\vec{x},y)}\left[\varepsilon^2\right] + \mathbb{E}_{\mathcal{D}_N}\left[\left(f(\vec{x}') - \widehat{f}_{\mathcal{D}_N}(\vec{x}')\right)^2\right] + \mathbb{E}_{(\vec{x},y),\,\mathcal{D}_N}\left[2\,\varepsilon\left(f(\vec{x}') - \widehat{f}_{\mathcal{D}_N}(\vec{x}')\right)\right] \\
&= \sigma^2 + \mathbb{E}_{\mathcal{D}_N}\left[\left(f(\vec{x}') - \widehat{f}_{\mathcal{D}_N}(\vec{x}')\right)^2\right] + 2\,\cancelto{0}{\mathbb{E}_{(\vec{x},y)}\left[\varepsilon\right]}\,\mathbb{E}_{\mathcal{D}_N}\left[f(\vec{x}') - \widehat{f}_{\mathcal{D}_N}(\vec{x}')\right] \\
&= \sigma^2 + \mathbb{E}_{\mathcal{D}_N}\left[f(\vec{x}') - \widehat{f}_{\mathcal{D}_N}(\vec{x}')\right]^2 + \mathrm{Var}_{\mathcal{D}_N}\left(f(\vec{x}') - \widehat{f}_{\mathcal{D}_N}(\vec{x}')\right)
\end{aligned}
$$

The Variance vs Bias Trade-off

Variance and bias in prediction

$$
\begin{aligned}
\mathbb{E}_{(\vec{x},y),\,\mathcal{D}_N}\left[\left(y' - \widehat{f}_{\mathcal{D}_N}(\vec{x}')\right)^2\right]
&= \mathbb{E}_{(\vec{x},y),\,\mathcal{D}_N}\left[\left(\varepsilon + f(\vec{x}') - \widehat{f}_{\mathcal{D}_N}(\vec{x}')\right)^2\right] \\
&= \mathbb{E}_{(\vec{x},y)}\left[\varepsilon^2\right] + \mathbb{E}_{\mathcal{D}_N}\left[\left(f(\vec{x}') - \widehat{f}_{\mathcal{D}_N}(\vec{x}')\right)^2\right] + \mathbb{E}_{(\vec{x},y),\,\mathcal{D}_N}\left[2\,\varepsilon\left(f(\vec{x}') - \widehat{f}_{\mathcal{D}_N}(\vec{x}')\right)\right] \\
&= \sigma^2 + \mathbb{E}_{\mathcal{D}_N}\left[\left(f(\vec{x}') - \widehat{f}_{\mathcal{D}_N}(\vec{x}')\right)^2\right] + 2\,\overbrace{\cancel{\mathbb{E}_{(\vec{x},y)}\left[\varepsilon\right]}}^{0}\,\mathbb{E}_{\mathcal{D}_N}\left[f(\vec{x}') - \widehat{f}_{\mathcal{D}_N}(\vec{x}')\right] \\
&= \sigma^2 + \mathbb{E}_{\mathcal{D}_N}\left[f(\vec{x}') - \widehat{f}_{\mathcal{D}_N}(\vec{x}')\right]^2 + \mathrm{Var}_{\mathcal{D}_N}\left(f(\vec{x}') - \widehat{f}_{\mathcal{D}_N}(\vec{x}')\right) \\
&= \quad \sigma^2 \quad + \quad \mathbb{E}_{\mathcal{D}_N}\left[f(\vec{x}') - \widehat{f}_{\mathcal{D}_N}(\vec{x}')\right]^2 \quad + \quad \mathrm{Var}_{\mathcal{D}_N}\left(\widehat{f}_{\mathcal{D}_N}(\vec{x}')\right)
\end{aligned}
$$

The Variance vs Bias Trade-off

Variance and bias in prediction

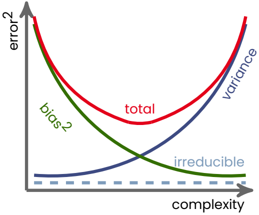
$$
\begin{aligned}
\mathbb{E}_{(\vec{x},y),\,\mathcal{D}_N}\left[\left(y' - \widehat{f}_{\mathcal{D}_N}(\vec{x}')\right)^2\right] &= \mathbb{E}_{(\vec{x},y),\,\mathcal{D}_N}\left[\left(\varepsilon + f(\vec{x}') - \widehat{f}_{\mathcal{D}_N}(\vec{x}')\right)^2\right] \\
&= \mathbb{E}_{(\vec{x},y)}\left[\varepsilon^2\right] + \mathbb{E}_{\mathcal{D}_N}\left[\left(f(\vec{x}') - \widehat{f}_{\mathcal{D}_N}(\vec{x}')\right)^2\right] + \mathbb{E}_{(\vec{x},y),\,\mathcal{D}_N}\left[2\,\varepsilon\left(f(\vec{x}') - \widehat{f}_{\mathcal{D}_N}(\vec{x}')\right)\right] \\
&= \sigma^2 + \mathbb{E}_{\mathcal{D}_N}\left[\left(f(\vec{x}') - \widehat{f}_{\mathcal{D}_N}(\vec{x}')\right)^2\right] + 2\,\cancelto{0}{\mathbb{E}_{(\vec{x},y)}\left[\varepsilon\right]}\,\mathbb{E}_{\mathcal{D}_N}\left[f(\vec{x}') - \widehat{f}_{\mathcal{D}_N}(\vec{x}')\right] \\
&= \sigma^2 + \mathbb{E}_{\mathcal{D}_N}\left[f(\vec{x}') - \widehat{f}_{\mathcal{D}_N}(\vec{x}')\right]^2 + \mathrm{Var}_{\mathcal{D}_N}\left(f(\vec{x}') - \widehat{f}_{\mathcal{D}_N}(\vec{x}')\right) \\
&= \underbrace{\sigma^2}_{\text{irreducible error}} + \underbrace{\mathbb{E}_{\mathcal{D}_N}\left[f(\vec{x}') - \widehat{f}_{\mathcal{D}_N}(\vec{x}')\right]^2}_{\textbf{bias}^2} + \underbrace{\mathrm{Var}_{\mathcal{D}_N}\left(\widehat{f}_{\mathcal{D}_N}(\vec{x}')\right)}_{\textbf{variance}}
\end{aligned}
$$

The Variance vs Bias Trade-off
Variance and bias in prediction

$$
\begin{aligned}
\mathbb{E}_{(\vec{x},y),\,\mathcal{D}_N}\left[\left(y' - \widehat{f}_{\mathcal{D}_N}(\vec{x}\,')\right)^2\right] &= \mathbb{E}_{(\vec{x},y),\,\mathcal{D}_N}\left[\left(\varepsilon + f(\vec{x}\,') - \widehat{f}_{\mathcal{D}_N}(\vec{x}\,')\right)^2\right] \\
&= \mathbb{E}_{(\vec{x},y)}\left[\varepsilon^2\right] + \mathbb{E}_{\mathcal{D}_N}\left[\left(f(\vec{x}\,') - \widehat{f}_{\mathcal{D}_N}(\vec{x}\,')\right)^2\right] + \mathbb{E}_{(\vec{x},y),\,\mathcal{D}_N}\left[2\,\varepsilon\left(f(\vec{x}\,') - \widehat{f}_{\mathcal{D}_N}(\vec{x}\,')\right)\right] \\
&= \sigma^2 + \mathbb{E}_{\mathcal{D}_N}\left[\left(f(\vec{x}\,') - \widehat{f}_{\mathcal{D}_N}(\vec{x}\,')\right)^2\right] + 2\,\cancelto{0}{\mathbb{E}_{(\vec{x},y)}\left[\varepsilon\right]}\,\mathbb{E}_{\mathcal{D}_N}\left[f(\vec{x}\,') - \widehat{f}_{\mathcal{D}_N}(\vec{x}\,')\right] \\
&= \sigma^2 + \mathbb{E}_{\mathcal{D}_N}\left[f(\vec{x}\,') - \widehat{f}_{\mathcal{D}_N}(\vec{x}\,')\right]^2 + \mathrm{Var}_{\mathcal{D}_N}\left(f(\vec{x}\,') - \widehat{f}_{\mathcal{D}_N}(\vec{x}\,')\right) \\
&= \underbrace{\underbrace{\sigma^2}_{\text{irreducible error}}}_{\text{aleatoric uncertainty}} + \underbrace{\underbrace{\mathbb{E}_{\mathcal{D}_N}\left[f(\vec{x}\,') - \widehat{f}_{\mathcal{D}_N}(\vec{x}\,')\right]^2}_{\textbf{bias}^2} + \underbrace{\mathrm{Var}_{\mathcal{D}_N}\left(\widehat{f}_{\mathcal{D}_N}(\vec{x}\,')\right)}_{\textbf{variance}}}_{\text{epistemic uncertainty}}
\end{aligned}
$$

High bias or **high variance** produce high/bad **generalisation errors**!
The choice of a good **validation strategy** becomes **fundamental**!

The Variance vs Bias Trade-off
Variance and bias in prediction



$$
\begin{aligned}
\mathbb{E}_{(\vec{x},y),\,\mathcal{D}_N}\left[\left(y' - \widehat{f}_{\mathcal{D}_N}(\vec{x}')\right)^2\right]
&= \mathbb{E}_{(\vec{x},y),\,\mathcal{D}_N}\left[\left(\varepsilon + f(\vec{x}') - \widehat{f}_{\mathcal{D}_N}(\vec{x}')\right)^2\right] \\
&= \mathbb{E}_{(\vec{x},y)}\left[\varepsilon^2\right] + \mathbb{E}_{\mathcal{D}_N}\left[\left(f(\vec{x}') - \widehat{f}_{\mathcal{D}_N}(\vec{x}')\right)^2\right] + \mathbb{E}_{(\vec{x},y),\,\mathcal{D}_N}\left[2\,\varepsilon\left(f(\vec{x}') - \widehat{f}_{\mathcal{D}_N}(\vec{x}')\right)\right] \\
&= \sigma^2 + \mathbb{E}_{\mathcal{D}_N}\left[\left(f(\vec{x}') - \widehat{f}_{\mathcal{D}_N}(\vec{x}')\right)^2\right] + 2\,\cancelto{0}{\mathbb{E}_{(\vec{x},y)}\left[\varepsilon\right]}\,\mathbb{E}_{\mathcal{D}_N}\left[f(\vec{x}') - \widehat{f}_{\mathcal{D}_N}(\vec{x}')\right] \\
&= \sigma^2 + \mathbb{E}_{\mathcal{D}_N}\left[f(\vec{x}') - \widehat{f}_{\mathcal{D}_N}(\vec{x}')\right]^2 + \mathrm{Var}_{\mathcal{D}_N}\left(f(\vec{x}') - \widehat{f}_{\mathcal{D}_N}(\vec{x}')\right) \\
&= \underbrace{\underbrace{\sigma^2}_{\text{irreducible error}}}_{\text{aleatoric uncertainty}} + \underbrace{\underbrace{\mathbb{E}_{\mathcal{D}_N}\left[f(\vec{x}') - \widehat{f}_{\mathcal{D}_N}(\vec{x}')\right]^2}_{\textbf{bias}^2} + \underbrace{\mathrm{Var}_{\mathcal{D}_N}\left(\widehat{f}_{\mathcal{D}_N}(\vec{x}')\right)}_{\textbf{variance}}}_{\text{epistemic uncertainty}}
\end{aligned}
$$

High bias or **high variance** produce high/bad **generalisation errors**!
The choice of a good **validation strategy** becomes **fundamental**!

Back to Cross Validation

A variance and bias perspective

What K should I choose? What happens to the **estimate** of the **prediction error**

$$\mathcal{E}^{(K)}_{\mathcal{D}_N^{(\mathrm{val})}}(y, \widehat{y}) = \mathbb{E}_{\text{K-folds}}\left[\mathbb{E}_{\mathcal{D}_N^{(\mathrm{val})}}\left[\mathrm{dist}(y, \widehat{y})\right]\right]$$

in **cross validation**?

Back to Cross Validation

A variance and bias perspective

What K should I choose? What happens to the **estimate** of the **prediction error** in **cross validation**?

Consider its *stability analysis*: (i.e. the study of its covariance, as it is an average of i.i.d. variables $\Rightarrow$ *central limit theorem*. See Bengio and Grandvalet, 2004)

$$\mathrm{Var}^{(K)}_{\mathcal{D}_N^{(\mathrm{val})}}\left(\mathrm{dist}(y, \widehat{y})\right) = \frac{1}{K^2} \sum_{i,j=0}^{K-1} \frac{1}{m_i\, m_j} \sum_{p,q=0}^{m_{i,j}-1} \mathrm{Cov}\left(\mathrm{dist}(y_p^{(i)}, \widehat{y}_p^{(i)}),\, \mathrm{dist}(y_q^{(j)}, \widehat{y}_q^{(j)})\right)$$

Back to Cross Validation

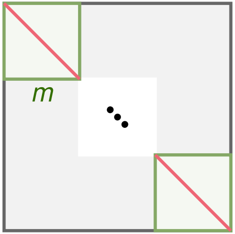
A variance and bias perspective

What K should I choose? What happens to the **estimate** of the **prediction error** in **cross validation**?

Consider its *stability analysis*: (i.e. the study of its covariance, as it is an average of i.i.d. variables $\Rightarrow$ *central limit theorem*. See Bengio and Grandvalet, 2004)

$$\mathrm{Var}^{(K)}_{\mathcal{D}_N^{(\mathrm{val})}}\left(\mathrm{dist}(y, \hat{y})\right) = \frac{1}{K^2} \sum_{i,j=0}^{K-1} \frac{1}{m_i\, m_j} \sum_{p,q=0}^{m_{i,j}-1} \mathrm{Cov}\left(\mathrm{dist}(y_p^{(i)}, \hat{y}_p^{(i)}),\, \mathrm{dist}(y_q^{(j)}, \hat{y}_q^{(j)})\right)$$

$$= \sum_{\textit{everything}}$$



$$= \frac{1}{n^2}\sigma^2 + \frac{m-1}{n}\omega + \frac{n-m}{n}\gamma$$

Lemma: $\nexists$ unbiased estimator of $\mathrm{Var}(\mathrm{dist}(y, \hat{y}))$

Back to Cross Validation

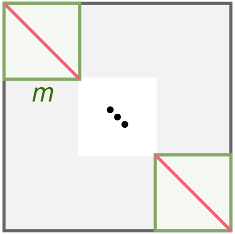
A variance and bias perspective

What K should I choose? What happens to the **estimate** of the **prediction error** in **cross validation**?

Consider its *stability analysis*: (i.e. the study of its covariance, as it is an average of i.i.d. variables $\Rightarrow$ *central limit theorem*. See Bengio and Grandvalet, 2004)

$$\mathrm{Var}^{(K)}_{\mathcal{D}_N^{(\mathrm{val})}}\left(\mathrm{dist}(y, \widehat{y})\right) = \frac{1}{K^2} \sum_{i,j=0}^{K-1} \frac{1}{m_i\, m_j} \sum_{p,q=0}^{m_{i,j}-1} \mathrm{Cov}\left(\mathrm{dist}(y_p^{(i)}, \widehat{y}_p^{(i)}),\ \mathrm{dist}(y_q^{(j)}, \widehat{y}_q^{(j)})\right)$$

$$= \sum_{\textit{everything}}$$



ω and γ depend on correlations!

$$= \frac{1}{n^2}\sigma^2 + \frac{m-1}{n}\omega + \frac{n-m}{n}\gamma$$

Lemma: $\nexists$ unbiased estimator of $\mathrm{Var}(\mathrm{dist}(y, \widehat{y}))$

2. The ML Mindset

Learning objectives and loss functions

Table of contents

Learning Objectives

Do machines learn?

Consider all elements:

- a "machine" needs **good data** as input even though nobody wants to tidy data for life...
- we need **structured** procedures to avoid mistakes
- we must use **good practices** (data split, validation, etc.)
- we have to deal with **bias** and **variance**
- a "machine" needs an **architecture** and an **objective** to train *what everyone wants!*

Learning Objectives

Do machines learn?

Consider all elements:

- a “machine” needs **good data** as input even though nobody wants to tidy data for life…
- we need **structured** procedures to avoid mistakes
- we must use **good practices** (data split, validation, etc.)
- we have to deal with **bias** and **variance**
- a “machine” needs an **architecture** and an **objective** to train *what everyone wants!*

How does a “machine” learn?

Learning Objectives

Do machines learn?

Consider all elements:

- a “machine” needs **good data** as input even though nobody wants to tidy data for life...
- we need **structured** procedures to avoid mistakes
- we must use **good practices** (data split, validation, etc.)
- we have to deal with **bias** and **variance**
- a “machine” needs an **architecture** and an **objective** to train *what everyone wants!*

How does a “machine” learn?

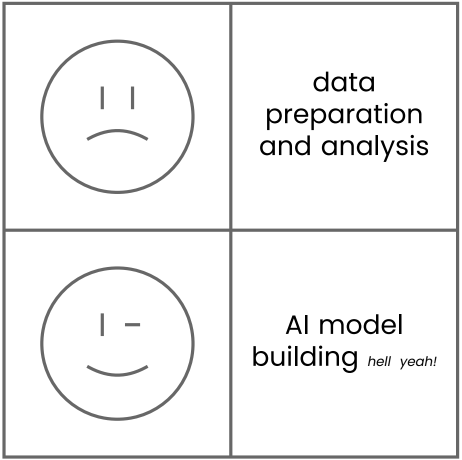
	data preparation and analysis
	AI model building *hell yeah!*

Learning Objectives
Do machines learn?

Consider all elements:

- a "machine" needs **good data** as input even though nobody wants to tidy data for life…
- we need **structured** procedures to avoid mistakes
- we must use **good practices** (data split, validation, etc.)
- we have to deal with **bias** and **variance**
- a "machine" needs an **architecture** and an **objective** to train *what everyone wants!*

How does a "machine" learn?



Learning Objectives

Do machines learn?

Consider all elements:

- a “machine” needs **good data** as input even though nobody wants to tidy data for life...
- we need **structured** procedures to avoid mistakes
- we must use **good practices** (data split, validation, etc.)
- we have to deal with **bias** and **variance**
- a “machine” needs an **architecture** and an **objective** to train *what everyone wants!*

How does a “machine” learn?

	data preparation and analysis
	AI model building *hell yeah!*

Learning Objectives

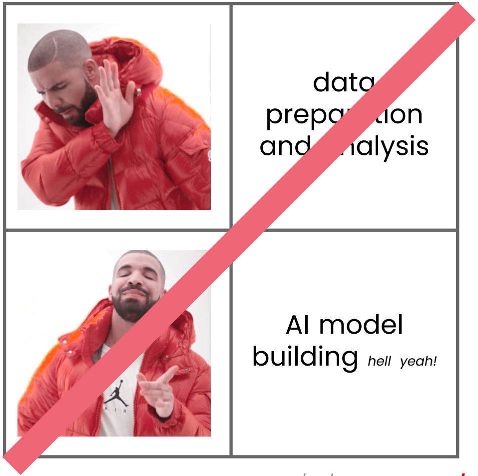
Do machines learn?

Consider all elements:

- a “machine” needs **good data** as input even though nobody wants to tidy data for life…
- we need **structured** procedures to avoid mistakes
- we must use **good practices** (data split, validation, etc.)
- we have to deal with **bias** and **variance**
- a “machine” needs an **architecture** and an **objective** to train *what everyone wants!*

How does a “machine” learn?

please, do not follow the advice, this is just a meme…



Loss Functions
Prediction estimation

Machines can learn in different ways:

$$\mathrm{dist}(y, \widehat{y}) \xrightarrow{\textit{better...}} (\mathcal{L},\ \mathscr{K},\ g) \quad \textbf{“loss (function)”}\ \text{(sometimes } \textit{Lagrangian}\text{)},$$

where $\mathscr{K} \sim \mathbb{C}^n$ (at least locally) with a **metric** tensor g:

$$\begin{aligned} \mathcal{L} : \mathscr{K} &\longrightarrow \mathbb{R} \\ Z &\longmapsto \mathcal{L}(Z) \overset{def}{=} \text{“distance from target”} \end{aligned}$$

Loss Functions

Prediction estimation

Machines can learn in different ways:

$$\mathrm{dist}(y, \widehat{y}) \xrightarrow{\text{better...}} (\mathscr{L},\ \mathscr{K},\ g) \quad \textbf{“loss (function)”} \text{ (sometimes } \textit{Lagrangian}\text{)},$$

where $\mathscr{K} \sim \mathbb{C}^n$ (at least locally) with a **metric** tensor g:

$$\begin{aligned} \mathscr{L} : \mathscr{K} &\longrightarrow \mathbb{R} \\ Z &\longmapsto \mathscr{L}(Z) \overset{def}{=} \text{“distance from target”} \end{aligned}$$

Notice $Z = Z(y, \widehat{y})$ and $\widehat{y} = f_{\{\Theta;\, \Omega\}}(x)$. The *training* problem (i.e. finding the *best* Θ^*) becomes:

$$\Theta^* = \arg\min_{\Theta} \mathscr{L}(Z) = \arg\min_{\Theta} \mathscr{L}\left(y, \widehat{y}(\Theta, \Omega)\right)$$

Should you see a correlation between $\mathscr{L}$ and the logarithm of a **likelihood** function, you would be basically right...

Loss Functions
Prediction estimation

Machines can learn in different ways:

$$\mathrm{dist}(y, \widehat{y}) \quad \xrightarrow{\textit{better...}} \quad (\mathcal{L}, \mathscr{K}, g)$$ **"loss (function)"** (sometimes *Lagrangian*),

where $\mathscr{K} \sim \mathbb{C}^n$ (at least locally) with a **metric** tensor g:

$$\begin{aligned} \mathcal{L} : \mathscr{K} &\longrightarrow \mathbb{R} \\ Z &\longmapsto \mathcal{L}(Z) \overset{\textit{def}}{=} \text{"distance from target"} \end{aligned}$$

Least squares

Let $Z = Y - \widehat{Y}$:

$$\mathcal{L}(Z) = \mathbb{E}_{\mathcal{P}(Y)}\left[\left(Y - \widehat{Y}\right)^2\right]$$

Cross entropy

Let $Z = \left(Y, \widehat{Y}\right)$:

$$\mathcal{L}(Z) = -\mathbb{E}_{\mathcal{P}(Y)}\left[\ln \widehat{Y}\right]$$

K-means clustering

Let $Z = X - \widehat{M}_c$:

$$\mathcal{L}(Z) = \mathbb{E}_{\mathcal{P}(X,C)}\left[\left(X - \widehat{M}_c\right)^2\right]$$

Loss Functions

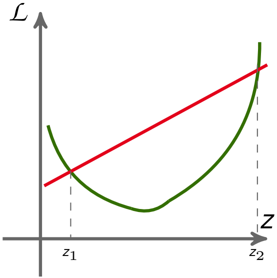
Some properties

What makes a **loss function** “good”? These properties are not always verified unfortunately…

Loss Functions
Some properties

What makes a **loss function** “good”? These properties are not always verified unfortunately...

Convexity



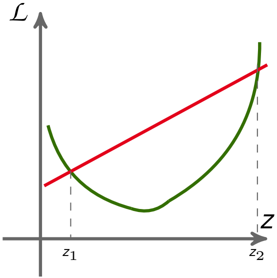
$$\mathcal{L}(t\, z_1 + (1-t)\, z_2) \leq t\, \mathcal{L}(z_1) + (1-t)\, \mathcal{L}(z_2)$$
$$\forall t \in [0,\, 1]$$

Loss Functions

Some properties

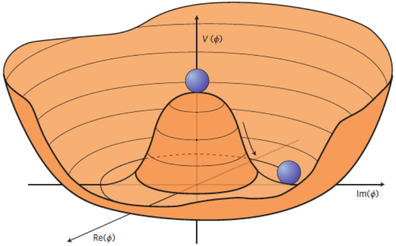
What makes a **loss function** “good”? These properties are not always verified unfortunately...

Convexity



$$\mathcal{L}(t\, z_1 + (1 - t)\, z_2) \leq t\, \mathcal{L}(z_1) + (1 - t)\, \mathcal{L}(z_2)$$
$$\forall t \in [0,\ 1]$$

Differentiability

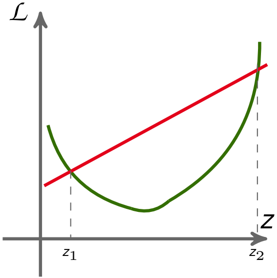


$$\forall c \in \mathscr{K} \setminus D_0\ \exists \lim_{x \to c} \frac{f(x) - f(c)}{x - c} = f'(x)\big|_{x=c}$$

Loss Functions
Some properties

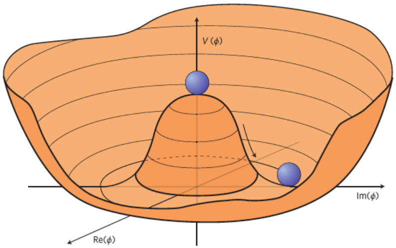
What makes a **loss function** "good"? These properties are not always verified unfortunately...

Convexity



$$\mathcal{L}(z_2) - \mathcal{L}(z_1) \geq \left.\frac{\mathrm{d}\mathcal{L}(z)}{\mathrm{d}z}\right|_{z=z_1} (z_2 - z_1)$$

Differentiability



$$\forall c \in \mathscr{K} \setminus D_0 \; \exists \lim_{x \to c} \frac{f(x) - f(c)}{x - c} = f'(x)|_{x=c}$$

Loss Functions

An example of convex loss

Let $\mathcal{P}$ and $\mathcal{Q}$ be two probability distributions of $X \in \mathfrak{X}$, and consider the Kullback-Leibler divergence:

$$\mathrm{D_{KL}}\left(\mathcal{P} \mid\mid \mathcal{Q}\right) = \mathbb{E}_{X \sim \mathcal{P}}\left[\ln \frac{\mathcal{P}(X)}{\mathcal{Q}(X)}\right] = \sum_{x \in \mathfrak{X}} \mathcal{P}(x) \ln \frac{\mathcal{P}(x)}{\mathcal{Q}(x)}.$$

Loss Functions

An example of convex loss

Let $\mathcal{P}$ and $\mathcal{Q}$ be two probability distributions of $X \in \mathfrak{X}$, and consider the Kullback-Leibler divergence:

$$\mathrm{D}_{\mathrm{KL}}\left(\mathcal{P} \parallel \mathcal{Q}\right) = \mathbb{E}_{X \sim \mathcal{P}}\left[\ln \frac{\mathcal{P}(X)}{\mathcal{Q}(X)}\right] = \sum_{x \in \mathfrak{X}} \mathcal{P}(x) \ln \frac{\mathcal{P}(x)}{\mathcal{Q}(x)}.$$

Theorem

$\mathrm{D}_{\mathrm{KL}}\left(\mathcal{P} \parallel \mathcal{Q}\right)$ *is a* ***convex*** *function in the pair* $(\mathcal{P}, \mathcal{Q})$ *over* $\mathfrak{X}$.

Loss Functions

An example of convex loss

Let $\mathcal{P}$ and $\mathcal{Q}$ be two probability distributions of $X \in \mathfrak{X}$, and consider the Kullback-Leibler divergence:

$$\mathrm{D}_{\mathrm{KL}}\left(\mathcal{P} \,||\, \mathcal{Q}\right) = \mathbb{E}_{X \sim \mathcal{P}}\left[\ln \frac{\mathcal{P}(X)}{\mathcal{Q}(X)}\right] = \sum_{x \in \mathfrak{X}} \mathcal{P}(x) \ln \frac{\mathcal{P}(x)}{\mathcal{Q}(x)}.$$

Theorem

$\mathrm{D}_{KL}\left(\mathcal{P} \,||\, \mathcal{Q}\right)$ *is a **convex** function in the pair* $(\mathcal{P}, \mathcal{Q})$ *over* $\mathfrak{X}$.

Proof

$$\mathrm{D}_{\mathrm{KL}}\left(t\,\mathcal{P}_1(X) + (1-t)\,\mathcal{P}_2(X) \,||\, t\,\mathcal{Q}_1(X) + (1-t)\,\mathcal{Q}_2(X)\right) =$$
$$= \sum_{x \in \mathfrak{X}} \left(\left(t\,\mathcal{P}_1(X) + (1-t)\,\mathcal{P}_2(X)\right) \ln \frac{t\,\mathcal{P}_1(X) + (1-t)\,\mathcal{P}_2(X)}{t\,\mathcal{Q}_1(X) + (1-t)\,\mathcal{Q}_2(X)} \right)$$

Loss Functions

An example of convex loss

Let $\mathcal{P}$ and $\mathcal{Q}$ be two probability distributions of $X \in \mathfrak{X}$, and consider the Kullback-Leibler divergence:

$$\mathrm{D_{KL}}\left(\mathcal{P} \;||\; \mathcal{Q}\right) = \mathbb{E}_{X \sim \mathcal{P}}\left[\ln \frac{\mathcal{P}(X)}{\mathcal{Q}(X)}\right] = \sum_{x \in \mathfrak{X}} \mathcal{P}(x) \ln \frac{\mathcal{P}(x)}{\mathcal{Q}(x)}.$$

Theorem

*$\mathrm{D_{KL}}\left(\mathcal{P} \;||\; \mathcal{Q}\right)$ is a **convex** function in the pair $(\mathcal{P}, \mathcal{Q})$ over $\mathfrak{X}$.*

Proof

$$\mathrm{D_{KL}}\left(t\,\mathcal{P}_1(X) + (1-t)\,\mathcal{P}_2(X) \;||\; t\,\mathcal{Q}_1(X) + (1-t)\,\mathcal{Q}_2(X)\right) \leq$$

"log sum inequality" $$\leq \sum_{x \in \mathfrak{X}} \left(t\,\mathcal{P}_1(X) \ln \frac{\cancel{t}\,\mathcal{P}_1(X)}{\cancel{t}\,\mathcal{Q}_1(X)} + (1-t)\,\mathcal{P}_2(X) \ln \frac{\cancel{(1-t)}\,\mathcal{P}_2(X)}{\cancel{(1-t)}\,\mathcal{Q}_2(X)} \right)$$

Loss Functions

An example of convex loss

Let $\mathcal{P}$ and $\mathcal{Q}$ be two probability distributions of $X \in \mathfrak{X}$, and consider the Kullback-Leibler divergence:

$$\mathrm{D}_{\mathrm{KL}}\left(\mathcal{P} \mid\mid \mathcal{Q}\right) = \mathbb{E}_{X\sim\mathcal{P}}\left[\ln \frac{\mathcal{P}(X)}{\mathcal{Q}(X)}\right] = \sum_{x\in\mathfrak{X}} \mathcal{P}(x) \ln \frac{\mathcal{P}(x)}{\mathcal{Q}(x)}.$$

Theorem

*$\mathrm{D}_{KL}\left(\mathcal{P} \mid\mid \mathcal{Q}\right)$ is a **convex** function in the pair $(\mathcal{P}, \mathcal{Q})$ over $\mathfrak{X}$.*

Proof

$$\begin{aligned}
&\mathrm{D}_{\mathrm{KL}}\left(t\,\mathcal{P}_1(X) + (1-t)\,\mathcal{P}_2(X) \mid\mid t\,\mathcal{Q}_1(X) + (1-t)\,\mathcal{Q}_2(X)\right) \leq \\
&\leq t\,\mathrm{D}_{\mathrm{KL}}\left(\mathcal{P}_1 \mid\mid \mathcal{Q}_1\right) + (1-t)\,\mathrm{D}_{\mathrm{KL}}\left(\mathcal{P}_2 \mid\mid \mathcal{Q}_2\right)
\end{aligned}$$

□

Loss Functions

An example of convex loss

Let $\mathcal{P}$ and $\mathcal{Q}$ be two probability distributions of $X \in \mathfrak{X}$, and consider the Kullback-Leibler divergence:

$$\mathrm{D}_{\mathrm{KL}}\left(\mathcal{P} \,||\, \mathcal{Q}\right) = \mathbb{E}_{X\sim\mathcal{P}}\left[\ln \frac{\mathcal{P}(X)}{\mathcal{Q}(X)}\right] = \sum_{x\in\mathfrak{X}} \mathcal{P}(x) \ln \frac{\mathcal{P}(x)}{\mathcal{Q}(x)}.$$

Theorem

$\mathrm{D}_{KL}\left(\mathcal{P} \,||\, \mathcal{Q}\right)$ *is a **convex** function in the pair* $\left(\mathcal{P}, \mathcal{Q}\right)$ *over* $\mathfrak{X}$*.*

HOMEWORK

1. prove that $f(x) = -\ln(x)$ is **convex** (or that $f(x) = \ln(x)$ is **concave**)
2. prove the "log sum inequality" (it follows from Jensen's inequality and 1.) used in the proof – have fun or look it up!
3. prove that the *cross entropy* $\mathrm{H}(\mathcal{P}, \mathcal{Q}) = -\mathbb{E}_{\mathcal{P}}\left[\ln \mathcal{Q}\right]$ is **convex** in $\mathcal{Q}$ over $\mathfrak{X}$
4. prove that any *local minimum* of a convex function is also a *global minimum* (suppose there are more, and find a contradiction...)

Loss Functions

Differentiability and gradient descent

Remember the *minimisation* problem:

$$\Theta^* = \arg\min_{\Theta} \mathcal{L}(Z) = \arg\min_{\Theta} \mathcal{L}\left(y, \widehat{y}(\Theta, \Omega)\right)$$

What if $\mathcal{L}$ is too complicated for an analytical solution?

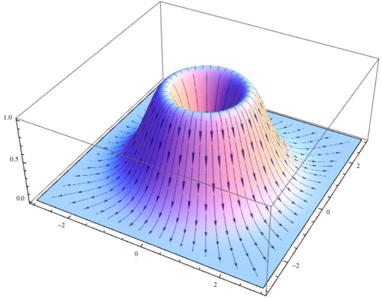
Loss Functions

Differentiability and gradient descent

Remember the *minimisation* problem:

$$\Theta^* = \arg\min_{\Theta} \mathcal{L}(Z) = \arg\min_{\Theta} \mathcal{L}\left(y, \widehat{y}(\Theta, \Omega)\right)$$

What if $\mathcal{L}$ is too complicated for an analytical solution?



Let $\vec{x}$, $\vec{v} \in \mathbb{R}^n$ and $f = f(\vec{x}) \in \mathscr{C}^2(\mathbb{R})$:

$$\vec{\nabla}_{\vec{v}} f(\vec{x}) = \vec{\nabla} f(\vec{x}) \cdot \vec{v}$$

which is maximal when $\vec{v}$ is in the **same direction** of $\vec{\nabla} f(\vec{x})$

Steepest ascent (theorem?)

The **gradient** is the direction of **steepest ascent** of the (hyper)surface.

Loss Functions

Differentiability and gradient descent

Remember the *minimisation* problem:

$$\Theta^* = \arg\min_{\Theta} \mathcal{L}(Z) = \arg\min_{\Theta} \mathcal{L}\left(y, \widehat{y}(\Theta, \Omega)\right)$$

What if $\mathcal{L}$ is too complicated for an analytical solution?

We can control the **descent** along the surface by iterating:

$$\vec{x}^{(t+1)} = \vec{x}^{(t)} - \vec{\alpha} \odot \vec{\nabla} f\left(\vec{x}^{(t)}\right)$$

where $\vec{\alpha} \in \mathbb{R}^n$ is the **learning rate** (N.B. $\alpha \in \Omega$ is a **hyperparameter** of the model often just a scalar...).

Loss Functions

Differentiability and gradient descent

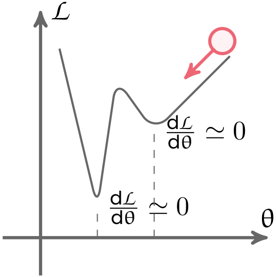
Remember the *minimisation* problem:

$$\Theta^* = \arg\min_{\Theta} \mathcal{L}(Z) = \arg\min_{\Theta} \mathcal{L}\left(y, \widehat{y}(\Theta, \Omega)\right)$$

What if $\mathcal{L}$ is too complicated for an analytical solution?

Gradient descent

Require: $\alpha \in \mathbb{R}^+$, $\theta^{(0)}$, $\mathcal{L}$, $T \in \mathbb{N} \setminus \{0\}$
for $0 \leq t < T$ **do**
$\vec{\mathcal{G}}^{(t)} \leftarrow \vec{\nabla}\mathcal{L}(y, \widehat{y}(\vec{\theta}^{(t)})) = \vec{\nabla}\mathcal{L}(\vec{\theta}^{(t)})$
$\vec{\theta}^{(t+1)} \leftarrow \vec{\theta}^{(t)} - \alpha\,\vec{\mathcal{G}}^{(t)}$ ▷ *steepest descent*
return $\vec{\theta}^{(T)}$



Loss Functions

Differentiability and gradient descent

Remember the *minimisation* problem:

$$\Theta^* = \arg\min_{\Theta} \mathcal{L}(Z) = \arg\min_{\Theta} \mathcal{L}\left(y, \widehat{y}(\Theta, \Omega)\right)$$

What if $\mathcal{L}$ is too complicated for an analytical solution?

Gradient descent

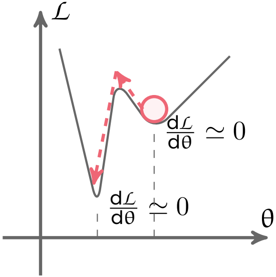
Require: $\alpha \in \mathbb{R}^+, \theta^{(0)}, \mathcal{L}, T \in \mathbb{N} \setminus \{0\}$

for $0 \le t < T$ **do**

$\quad \vec{\mathcal{G}}^{(t)} \leftarrow \vec{\nabla}\mathcal{L}(y, \widehat{y}(\vec{\theta}^{(t)})) = \vec{\nabla}\mathcal{L}(\vec{\theta}^{(t)})$

$\quad \vec{\theta}^{(t+1)} \leftarrow \vec{\theta}^{(t)} - \alpha\,\vec{\mathcal{G}}^{(t)}$ $\quad \triangleright$ *steepest descent*

return $\vec{\theta}^{(T)}$



Loss Functions

Differentiability and gradient descent

Remember the *minimisation* problem:

$$\Theta^* = \arg\min_{\Theta} \mathcal{L}(Z) = \arg\min_{\Theta} \mathcal{L}\left(y, \widehat{y}(\Theta, \Omega)\right)$$

What if $\mathcal{L}$ is too complicated for an analytical solution?

Gradient descent

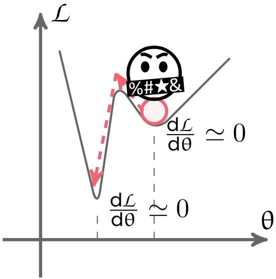
Require: $\alpha \in \mathbb{R}^+$, $\theta^{(0)}$, $\mathcal{L}$, $T \in \mathbb{N} \setminus \{0\}$

for $0 \leq t < T$ **do**

$\quad \vec{\mathcal{G}}^{(t)} \leftarrow \vec{\nabla}\mathcal{L}(y, \widehat{y}(\vec{\theta}^{(t)})) = \vec{\nabla}\mathcal{L}(\vec{\theta}^{(t)})$

$\quad \vec{\theta}^{(t+1)} \leftarrow \vec{\theta}^{(t)} - \alpha\,\vec{\mathcal{G}}^{(t)}$ ▷ *steepest descent*

return $\vec{\theta}^{(T)}$



Loss Functions

Differentiability and gradient descent

Remember the *minimisation* problem:

$$\Theta^* = \arg\min_{\Theta} \mathcal{L}(Z) = \arg\min_{\Theta} \mathcal{L}\left(y, \widehat{y}(\Theta, \Omega)\right)$$

What if $\mathcal{L}$ is too complicated for an analytical solution?

Gradient descent

Require: $\alpha \in \mathbb{R}^+, \theta^{(0)}, \mathcal{L}, T \in \mathbb{N} \setminus \{0\}$

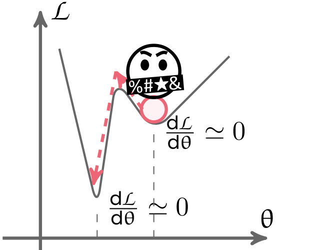
for $0 \leq t < T$ **do**

$\quad \vec{G}^{(t)} \leftarrow \vec{\nabla}\mathcal{L}(y, \hat{y}(\vec{\theta}^{(t)})) = \vec{\nabla}\mathcal{L}(\vec{\theta}^{(t)})$

$\quad \vec{\theta}^{(t+1)} \leftarrow \vec{\theta}^{(t)} - \alpha\,\vec{G}^{(t)}$ $\qquad \triangleright$ *steepest descent*

return $\vec{\theta}^{(T)}$

Does it converge?



We will see better solutions later on...

Gradient Descent

On the convergence of gradient descent

Definition | Lipschitz smoothness

Let $f \in \mathscr{C}^1(\mathbb{R}^n)$ a scalar function, and $L > 0$. We call f *L-smooth* if it is L-Lipschitz:

$$\forall \vec{x}, \vec{y} \in \mathbb{R}^n \quad \left|\left|\vec{\nabla} f(\vec{x}) - \vec{\nabla} f(\vec{y})\right|\right|_2 \leq L \, ||\vec{x} - \vec{y}||_2$$

Gradient Descent

On the convergence of gradient descent

Theorem | Smooth convex functions

Let f be a convex L-smooth scalar function over $\mathbb{R}^n$, and let $\alpha = L^{-1}$ the learning rate, then $\forall t \in [1, T]$:

$$f(\vec{x}^{(t)}) - f(\vec{x}^*) \leq \frac{2L}{T-1} \left|\left| x^{(0)} - x^* \right|\right|_2 .$$

(see Gower (Télécom Paris))

Gradient Descent

On the convergence of gradient descent

Theorem | Smooth convex functions

Let f be a convex L-smooth scalar function over $\mathbb{R}^n$, and let $\alpha = L^{-1}$ the learning rate, then $\forall t \in [1, T]$:

$$f(\vec{x}^{(t)}) - f(\vec{x}^*) \leq \frac{2L}{T-1} \left|\left| x^{(0)} - x^* \right|\right|_2 .$$

(see Gower (Télécom Paris))

Consider $\vec{x}^{(t+1)} = \vec{x}^{(t)} - \frac{1}{L} \vec{\nabla} f\left(\vec{x}^{(t)}\right)$:

$$\left|\left| \vec{\nabla} f\left(\vec{x}^{(t+1)}\right) - \vec{\nabla} f\left(\vec{x}^{(t)}\right) \right|\right|_2 = \left|\left| \int_{\vec{x}^{(t)}}^{\vec{x}^{(t+1)}} \mathrm{d}\vec{s}\ \nabla^2 f(\vec{s}) \right|\right|_2 \leq L \left|\left| \vec{x}^{(t+1)} - \vec{x}^{(t)} \right|\right|_2 = \frac{1}{L} \left|\left| \vec{\nabla} f\left(\vec{x}^{(t)}\right) \right|\right|_2$$

which shows (Taylor expansion + bound on Hessian) that

$$f\left(\vec{x}^{(t+1)}\right) \leq f\left(\vec{x}^{(t)}\right) - \frac{1}{L} \left|\left| \vec{\nabla} f\left(x^{(t)}\right) \right|\right|_2 + \frac{1}{2L} \left|\left| \vec{\nabla} f\left(\vec{x}^{(t)}\right) \right|\right|_2 = f\left(\vec{x}^{(t)}\right) - \frac{1}{2L} \left|\left| \vec{\nabla} f\left(x^{(t)}\right) \right|\right|_2 .$$

Moreover, by subtracting $f\left(\vec{x}^*\right)$:

$$f\left(\vec{x}^{(t+1)}\right) - f\left(\vec{x}^*\right) \leq f\left(\vec{x}^{(t)}\right) - f\left(\vec{x}^*\right) - \frac{1}{2L} \left|\left| \vec{\nabla} f\left(x^{(t)}\right) \right|\right|_2 .$$

Gradient Descent

On the convergence of gradient descent

Theorem | Smooth convex functions

Let f be a convex L-smooth scalar function over $\mathbb{R}^n$, and let $\alpha = L^{-1}$ the learning rate, then $\forall t \in [1, T]$:

$$f(\vec{x}^{(t)}) - f(\vec{x}^*) \leq \frac{2L}{T-1} \left\|x^{(0)} - x^*\right\|_2 .$$

(see Gower (Télécom Paris))

From the last equation

$$f\left(\vec{x}^{(t+1)}\right) - f\left(\vec{x}^*\right) \leq f\left(\vec{x}^{(t)}\right) - f\left(\vec{x}^*\right) - \frac{1}{2L}\left\|\vec{\nabla} f\left(x^{(t)}\right)\right\|_2 ,$$

apply the **convexity** property

$$f\left(\vec{x}^{(t)}\right) - f\left(\vec{x}^*\right) \leq \vec{\nabla} f\left(x^{(t)}\right) \cdot \left(\vec{x}^{(t)} - \vec{x}^*\right) \leq \left\|\vec{\nabla} f\left(x^{(t)}\right)\right\|_2^2 \left\|x^{(t)} - \vec{x}^*\right\|_2^2$$

and reconstruct:

$$\Delta^{(t+1)} \leq \Delta^{(t)} - \beta \left(\Delta^{(t)}\right)^2 \quad \Leftrightarrow \quad \beta \leq \beta \frac{\Delta^{(t)}}{\Delta^{(t+1)}} \leq \frac{1}{\Delta^{(t+1)}} - \frac{1}{\Delta^{(t)}},$$

where $\Delta^{(t)} = f\left(\vec{x}^{(t)}\right) - f\left(\vec{x}^*\right)$ and

$$\beta = \frac{1}{2L\left\|x^{(0)} - x^*\right\|_2^2}, \quad \text{since} \quad \left\|x^{(t)} - x^*\right\|_2^2 \leq \left\|x^{(0)} - x^*\right\|_2^2 .$$

Gradient Descent

On the convergence of gradient descent

Theorem | Smooth convex functions

Let f be a convex L-smooth scalar function over $\mathbb{R}^n$, and let $\alpha = L^{-1}$ the learning rate, then $\forall t \in [1, T]$:

$$f(\vec{x}^{(t)}) - f(\vec{x}^*) \leq \frac{2L}{T-1} \left|\left| x^{(0)} - x^* \right|\right|_2 .$$

(see Gower (Télécom Paris))

We finally conclude by summing over all $t \in [1, T]$:

$$(T-1)\,\beta \leq \frac{1}{\Delta(T)} - \frac{1}{\Delta(1)} \leq \frac{1}{\Delta(T)} \leq \frac{1}{\Delta(t)} .$$

$\square$

Loss Functions vs Metrics

Evaluating algorithms

Is there a difference between **metrics** and **loss functions**? Can I choose arbitrarily?

Depending on the task, we need a *differentiable* loss, but not necessarily a continuous evaluation!

Loss Functions vs Metrics
Evaluating algorithms

Regression task N.B. we are not discussing "good vs bad" metric/loss

Let $f_{\{\Theta,\Omega\}} : \mathbb{R}^n \to \mathbb{R}$, such that $\vec{x} \overset{f}{\mapsto} \widehat{y}$, a **regression** model

$$\begin{aligned} \text{metric} \quad &\to \quad ||y - \widehat{y}||_p \\ \text{loss} \quad &\to \quad ||y - \widehat{y}||_{p>0} \end{aligned}$$

A priori, we could use the loss as evaluation metric as well.

Depending on the task, we need a *differentiable* loss, but not necessarily a continuous evaluation!

Loss Functions vs Metrics

Evaluating algorithms

Regression task N.B. we are not discussing "good vs bad" metric/loss

Let $f_{\{\Theta,\Omega\}} : \mathbb{R}^n \to \mathbb{R}$, such that $\vec{x} \overset{f}{\mapsto} \hat{y}$, a **regression** model

$$\begin{aligned} \text{metric} \quad &\to \quad ||y - \hat{y}||_p \\ \text{loss} \quad &\to \quad ||y - \hat{y}||_{p>0} \end{aligned}$$

A priori, we could use the loss as evaluation metric as well.

Classification task N.B. we are not discussing "good vs bad" metric/loss

Let $f_{\{\Theta,\Omega\}} : \mathbb{R}^n \to \mathbb{R}$, such that $\vec{x} \overset{f}{\mapsto} \hat{y}$ (probability of being *positive* sample), a **classification** model

$$\begin{aligned} \text{metric} \quad &\to \quad \textit{accuracy} \\ \text{loss} \quad &\to \quad \mathbb{E}_C \left[\ln \hat{C} \right] \text{ (cross entropy)} \end{aligned}$$

Depending on the task, we need a *differentiable* loss, but not necessarily a continuous evaluation!

2. The ML Mindset

Regularisation

Table of contents

Regularisation Techniques
Containing the overfit

Q: Is there a way to *actively* reduce overfitting the training data?

Regularisation Techniques
Containing the overfit

Q: Is there a way to *actively* reduce overfitting the training data?

$$\text{regularised model} = \text{model} + \text{constraint on parameters}$$

Regularisation Techniques
Containing the overfit

Q: Is there a way to *actively* reduce overfitting the training data?

$$\mathcal{L}_{\mathsf{full}}(\Theta;\, \Omega) \quad = \quad \mathcal{L}(\Theta;\, \Omega) \quad +$$

Regularisation Techniques
Containing the overfit

Q: Is there a way to *actively* reduce overfitting the training data?

$$\mathcal{L}_{\text{full}}(\Theta;\ \Omega, \Lambda) \quad = \quad \mathcal{L}(\Theta;\ \Omega) \quad + \quad \mathcal{L}_{\text{reg}}(\Theta;\ \Lambda)$$

Regularisation Techniques
Containing the overfit

Q: Is there a way to *actively* reduce overfitting the training data?

$$\mathcal{L}_{\text{full}}(\Theta;\, \Omega, \Lambda) \quad = \quad \mathcal{L}(\Theta;\, \Omega) \quad + \quad \mathcal{L}_{\text{reg}}(\Theta;\, \Lambda)$$

L_p regularisation

Define the L_p norm of $\vec{x} \in \mathbb{R}^k$:

$$||\vec{x}||_p = \left(\sum_{i=1}^{p} |x_i|^p \right)^{\frac{1}{p}}, \qquad \text{special cases} \qquad \begin{cases} L_0 & : \; ||\vec{x}||_0 = \sum_{i=1}^{p} \delta_{|x_i|,\,0} \\ L_\infty & : \; ||\vec{x}||_\infty = \sup_{i \in [1,p]} |x_i| \end{cases}$$

Regularisation Techniques
Containing the overfit

Q: Is there a way to *actively* reduce overfitting the training data?

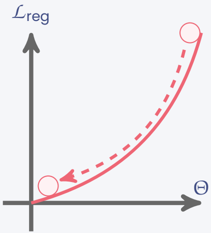
$$\mathcal{L}_{\text{full}}(\Theta;\ \Omega, \Lambda) \quad = \quad \mathcal{L}(\Theta;\ \Omega) \quad + \quad \mathcal{L}_{\text{reg}}(\Theta;\ \Lambda)$$

L_p regularisation

Then:

$$\mathcal{L}_{\text{reg}}(\Theta;\ \Lambda) = \lambda_p \, ||\Theta||_p^p \,, \quad \text{s.t.} \quad \lambda_p \in \Lambda.$$

Most common regularisation techniques are $p = 1$ (**LASSO**) and $p = 2$ (**Ridge**).



Regularisation Techniques
Containing the overfit

Q: Is there a way to *actively* reduce overfitting the training data?

$$\mathcal{L}_{\text{full}}(\Theta;\ \Omega, \Lambda) \quad = \quad \mathcal{L}(\Theta;\ \Omega) \quad + \quad \mathcal{L}_{\text{reg}}(\Theta;\ \Lambda)$$

L_1 and L_2 regularisation: probabilistic interpretation or simple trick?

Remember $\mathcal{P}(A \cap B) = \mathcal{P}(A \mid B)\mathcal{P}(B) = \mathcal{P}(B \mid A)\mathcal{P}(A)$ which implies (Bayes' theorem):

$$\underbrace{\mathcal{P}(A \mid B)}_{\text{posterior}} = \frac{\overbrace{\mathcal{P}(B \mid A)}^{\text{likelihood}}\ \overbrace{\mathcal{P}(A)}^{\text{prior}}}{\underbrace{\mathcal{P}(B)}_{\text{marginal}}}.$$

Regularisation Techniques
Containing the overfit

Q: Is there a way to *actively* reduce overfitting the training data?

$$\mathcal{L}_{\text{full}}(\Theta;\ \Omega, \Lambda) \quad = \quad \mathcal{L}(\Theta;\ \Omega) \quad + \quad \mathcal{L}_{\text{reg}}(\Theta;\ \Lambda)$$

L_1 and L_2 regularisation: probabilistic interpretation or simple trick?

$$\Theta^* = \underbrace{\arg\max_{\Theta} \left(\ln \mathcal{P}(\Theta \mid \vec{x})\right)}_{\text{MAP}} = \arg\max_{\Theta} \left(\underbrace{\ln \mathcal{P}(\vec{x} \mid \Theta)}_{\text{MLE}} + \underbrace{\ln \mathcal{P}(\Theta)}_{\text{prior}} \right)$$

where, for $\Theta = (\theta_1, \theta_2, \dots, \theta_p)$:

$$L_1: \quad \mathcal{P}(\Theta) = \text{Laplace}(\Theta \mid 0, b) = \frac{1}{2^p b^p} e^{-\frac{||\Theta||_1}{b}}, \qquad L_2: \quad \mathcal{P}(\Theta) = \mathcal{N}(\Theta \mid 0, \sigma^2) = \frac{1}{(2\pi\sigma^2)^p} e^{-\frac{||\Theta||_2^2}{2\sigma^2}}.$$

N.B.: certainly "MAP w/ prior $\Rightarrow$ penalised least squares", but "penalised least squares $\not\Rightarrow$ Gaussian/Laplace prior": it is rather a matter of efficiency and good results.

(see Gribonval (2011))

Regularisation Techniques
Containing the overfit

Q: Is there a way to *actively* reduce overfitting the training data?

$$\mathcal{L}_{\text{full}}(\Theta;\, \Omega, \Lambda) \quad = \quad \mathcal{L}(\Theta;\, \Omega) \quad + \quad \mathcal{L}_{\text{reg}}(\Theta;\, \Lambda)$$

L_1 and L_2 regularisation: probabilistic interpretation or simple trick?

$$\Theta^* = \underbrace{\arg\max_{\Theta} \left(\ln \mathcal{P}(\Theta \mid \vec{x})\right)}_{\text{MAP}} = \arg\max_{\Theta} \left(\underbrace{\ln \mathcal{P}(\vec{x} \mid \Theta)}_{\text{MLE}} + \underbrace{\ln \mathcal{P}(\Theta)}_{\text{prior}} \right)$$

where, for $\Theta = (\theta_1, \theta_2, \dots, \theta_p)$:

$$L_1: \quad \mathcal{P}(\Theta) = \text{Laplace}(\Theta \mid 0, b) = \frac{1}{2^p b^p} e^{-\frac{||\Theta||_1}{b}}, \qquad L_2: \quad \mathcal{P}(\Theta) = \mathcal{N}(\Theta \mid 0, \sigma^2) = \frac{1}{(2\pi\sigma^2)^p} e^{-\frac{||\Theta||_2^2}{2\sigma^2}}.$$

Q: $\mathcal{L}_{\text{reg}}$ adds a restrictions on the parameters of the model to contain the overfit. How to interpret this in terms of **bias** vs **variance**?

(see Gribonval (2011))

Regularisation Techniques

Containing the overfit

Q: Is there a way to *actively* reduce overfitting the training data?

$$\mathcal{L}_{\text{full}}(\Theta;\ \Omega, \Lambda) \quad = \quad \mathcal{L}(\Theta;\ \Omega) \quad + \quad \mathcal{L}_{\text{reg}}(\Theta;\ \Lambda)$$

L_1 and L_2 regularisation: probabilistic interpretation or simple trick?

$$\Theta^* = \underbrace{\arg\max_{\Theta} \left(\ln \mathcal{P}(\Theta \mid \vec{x})\right)}_{\text{MAP}} = \arg\max_{\Theta} \left(\underbrace{\ln \mathcal{P}(\vec{x} \mid \Theta)}_{\text{MLE}} + \underbrace{\ln \mathcal{P}(\Theta)}_{\text{prior}} \right)$$

where, for $\Theta = (\theta_1, \theta_2, \dots, \theta_p)$:

$$L_1: \quad \mathcal{P}(\Theta) = \text{Laplace}(\Theta \mid 0, b) = \frac{1}{2^p b^p} e^{-\frac{||\Theta||_1}{b}}, \qquad L_2: \quad \mathcal{P}(\Theta) = \mathcal{N}(\Theta \mid 0, \sigma^2) = \frac{1}{(2\pi\sigma^2)^p} e^{-\frac{||\Theta||_2^2}{2\sigma^2}}.$$

Q: $\mathcal{L}_{\text{reg}}$ adds a restrictions on the parameters of the model to contain the overfit. How to interpret this in terms of **bias** vs **variance**? Increase in bias, decrease in variance.

(see Gribonval (2011))

3. ML Algorithms

Taxonomy of algorithms

Table of contents

Types of Algorithms

A simple distinction

SUPERVISED

UNSUPERVISED

Types of Algorithms
A simple distinction

Supervised learning

- "learn" knowing the **result** (labels)
- **iterative** process with examples
- need **annotated** data

Types of Algorithms

A simple distinction

SUPERVISED

UNSUPERVISED

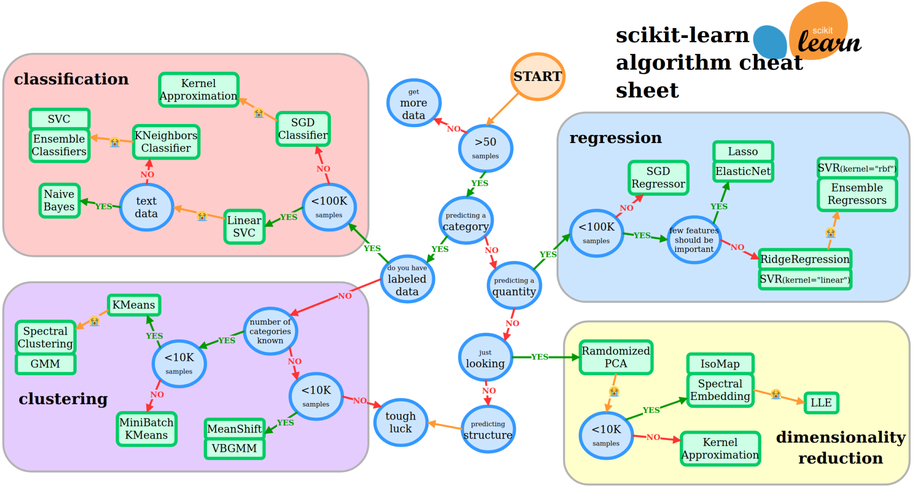
Supervised learning

- “learn” knowing the **result** (labels)
- **iterative** process with examples
- need **annotated** data

Unsupervised learning

- “learn” a **structure** in the data
- can identify usable **patterns**
- data are **not labelled**

Types of Algorithms
A more realistic scenario



Types of Algorithms
Some more details...

Supervised learning

Let $\mathcal{D} = \{(\vec{x}, y)\}$ a **labelled** dataset:

$$f_{\text{supervised}} \colon \mathbb{K}^p \to \mathbb{K}$$

- regression
- classification
- time series inference
- (LLMs, generative AI, etc. debatable)

Unsupervised learning

Let $\mathcal{D} = \{\vec{x}\}$ a set of **data points**:

$$f_{\text{unsupervised}} \colon \mathbb{K}^p \to \mathbb{K}^q$$

- principal components analysis
- clustering and manifold learning
- anomaly detection
- etc.

Types of Algorithms
A wide spectrum

Types of Algorithms

A wide spectrum

Supervised learning

Unsupervised learning

Self-supervised learning

partially labelled data, similarity analysis (e.g. SimCLR)

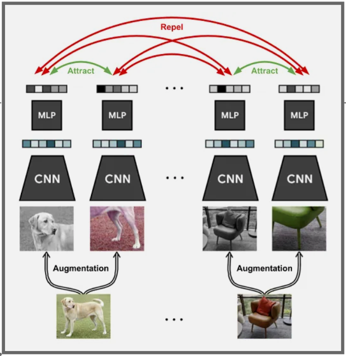
Types of Algorithms

A wide spectrum

Supervised learning

Self-supervised learning

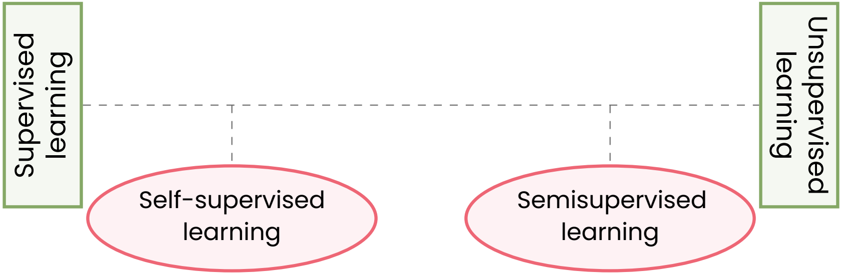
partially labelled data, similarity analysis (e.g. SimCLR)



Unsupervised learning

Types of Algorithms

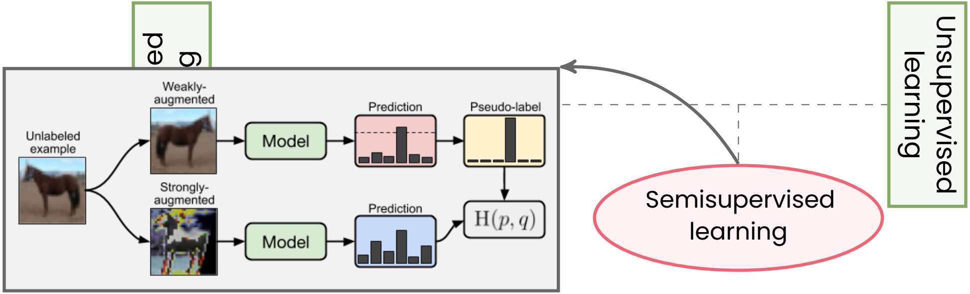
A wide spectrum



partially labelled data, similarity analysis (e.g. SimCLR)

train supervised → consistent prediction of unlabelled (e.g. FixMatch)

Types of Algorithms
A wide spectrum



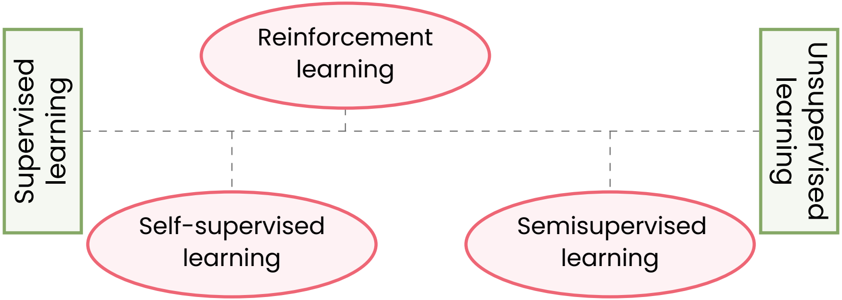
partially labelled data, similarity analysis (e.g. SimCLR)

train supervised → consistent prediction of unlabelled (e.g. FixMatch)

Types of Algorithms
A wide spectrum

prize vs penalty training

(e.g. Boston Dynamics)



partially labelled data, similarity analysis (e.g. SimCLR)

train supervised → consistent prediction of unlabelled (e.g. FixMatch)

Types of Algorithms
A wide spectrum

prize *vs* *penalty* training

(e.g. Boston Dynamics)

Reinforcement learning

Supervised learning

ed

Self-supervised learning

Semisupervised learning

partially labelled data, similarity analysis (e.g. SimCLR)

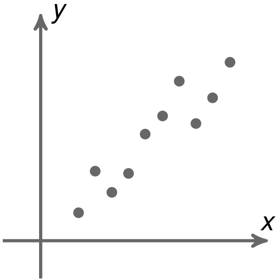
train supervised → consistent prediction of unlabelled (e.g. FixMatch)

3. ML Algorithms

Unsupervised learning

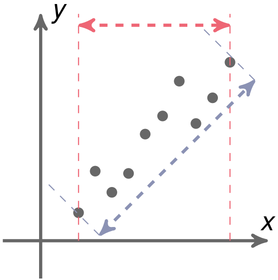
Table of contents

Principal Components Analysis
Definition



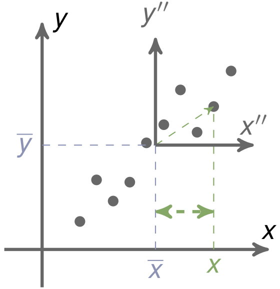
Principal Components Analysis

Definition



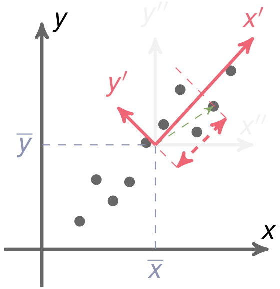
- data better from another angle ($\leftarrow\!\!-\!\!\rightarrow$ > $\leftarrow\!\!-\!\!\rightarrow$)

Principal Components Analysis
Definition



- data better from another angle (<–> > <–>)
- "distance" from centre

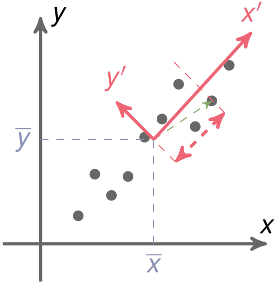
Principal Components Analysis
Definition



- data better from another angle (<–> > <–>)
- "distance" from centre
- find maximal "distance^2"

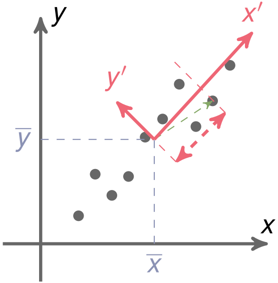
Principal Components Analysis

Definition



- data better from another angle ($\langle - \rangle$ > $\langle - \rangle$)
- "distance" from centre
- find maximal "distance^2" (i.e. maximal **variance**)

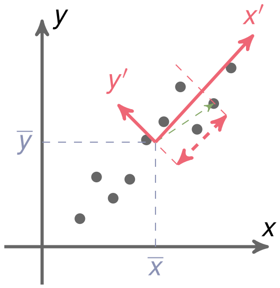
Principal Components Analysis
Definition



- data better from another angle ($\leftarrow\!-\!\rightarrow$ > $\leftarrow\!-\!\rightarrow$)
- "distance" from centre
- find maximal "distance^2" (i.e. maximal **variance**)

Let $\vec{x}_i \in \mathbb{R}^p$ for $i = 1, 2, \dots, n$ s.t. $\mathbb{E}[\vec{x}] = \overline{\vec{x}}$. Call $\vec{y}_i = \vec{x}_i - \overline{\vec{x}}$ $(i = 1, 2, \dots, n)$ the **centred** data.

Principal Components Analysis
Definition



- data better from another angle ($<\!-\!> \;>\; <\!-\!>$)
- "distance" from centre
- find maximal "distance^2" (i.e. maximal **variance**)

Let $\vec{x}_i \in \mathbb{R}^p$ for $i = 1, 2, \dots, n$ s.t. $\mathbb{E}[\vec{x}] = \overline{\vec{x}}$. Call $\vec{y}_i = \vec{x}_i - \overline{\vec{x}}$ $(i = 1, 2, \dots, n)$ the **centred** data.

Preliminaries (spectral theorem)

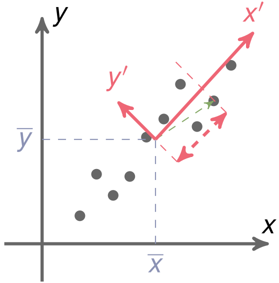
Let $M \in \mathbb{R}^{p \times p}$ s.t. $M^T = M$. Then, $\exists$ complete orthonormal basis of $\mathbb{R}^p$ $\left\{\vec{e}_1, \vec{e}_2, \dots, \vec{e}_p\right\}$ s.t.

$$M = \left(\vec{m}_i^T\right)_{i=1,2,\dots,p} = \sum_{i=1}^{p} \lambda_i \vec{e}_i \vec{e}_i^T,$$

where $\lambda_i \in \mathbb{R}^+ \quad \forall i \in [1, p]$.

Principal Components Analysis

Definition



- data better from another angle ($\langle - \rangle > \langle - \rangle$)
- "distance" from centre
- find maximal "distance²" (i.e. maximal **variance**)

Let $\vec{x}_i \in \mathbb{R}^p$ for $i = 1, 2, \dots, n$ s.t. $\mathbb{E}[\vec{x}] = \overline{\vec{x}}$. Call $\vec{y}_i = \vec{x}_i - \overline{\vec{x}}$ $(i = 1, 2, \dots, n)$ the **centred** data.

Maximal variance

Let $Y \in \mathbb{R}^{n \times p}$ matrix representation of data with **covariance** $C = n^{-1} X^T X$.

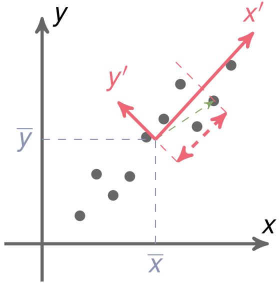
We look for a *new basis*

$$\underbrace{Y'}_{\text{scores / principal components}} = Y \underbrace{W}_{\text{loadings}}$$

which maximises the variance in each direction of the vectors $\vec{y}'_{[i]} \in \mathbb{R}^p$ $(i = 1, 2, \dots, n)$.

Principal Components Analysis

Definition



- data better from another angle ($\langle - \rangle > \langle - \rangle$)
- "distance" from centre
- find maximal "distance^2" (i.e. maximal **variance**)

Let $\vec{x}_i \in \mathbb{R}^p$ for $i = 1, 2, \dots, n$ s.t. $\mathbb{E}[\vec{x}] = \overline{\vec{x}}$. Call $\vec{y}_i = \vec{x}_i - \overline{\vec{x}}$ $(i = 1, 2, \dots, n)$ the **centred** data.

Maximal variance

We need:

$$\mathrm{Var}(y'_{[i](a)}) = \mathrm{Var}(\vec{y}_{[i]} \cdot \vec{w}_{(a)}) = \vec{w}^T_{(a)}\, C_{[i]}\, \vec{w}_{(a)}.$$

and compute

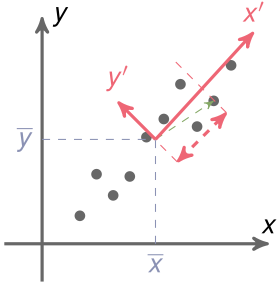
$$\arg\max_{\vec{w}_{(a)}} \vec{w}^T_{(a)}\, C_{[i]}\, \vec{w}_{(a)}$$

constrained to

$$\vec{w}_{(a)} \cdot \vec{w}_{(b)} = \delta_{ab}.$$

Principal Components Analysis

Definition



- data better from another angle ($<-> \; > \; <->$)
- "distance" from centre
- find maximal "distance^2" (i.e. maximal **variance**)

Let $\vec{x}_i \in \mathbb{R}^p$ for $i = 1, 2, \dots, n$ s.t. $\mathbb{E}[\vec{x}] = \overline{\vec{x}}$. Call $\vec{y}_i = \vec{x}_i - \overline{\vec{x}}$ $(i = 1, 2, \dots, n)$ the **centred** data.

Principal Components (theorem)

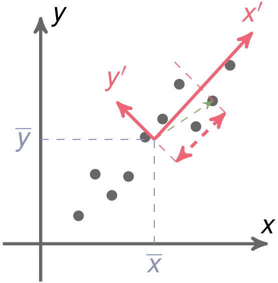
If C has distinct eigenvalues $\lambda_1, \dots, \lambda_p$, then

$\vec{w}_{(a)}$ is the eigenvector corresponding to the a-th largest eigenvector λ_a.

Moreover,

$$\mathrm{Var}(y'_{[i](a)}) = \vec{w}^T_{(a)} C_{[i]} \, \vec{w}_{(a)} = \lambda_a.$$

Principal Components Analysis
Definition



- data better from another angle ($<\!-\!> \; > \; <\!-\!>$)
- "distance" from centre
- find maximal "distance^2" (i.e. maximal **variance**)

Let $\vec{x}_i \in \mathbb{R}^p$ for $i = 1, 2, \dots, n$ s.t. $\mathbb{E}[\vec{x}] = \overline{\vec{x}}$. Call $\vec{y}_i = \vec{x}_i - \overline{\vec{x}}$ ($i = 1, 2, \dots, n$) the **centred** data.

Principal Components (theorem)

If C has distinct eigenvalues $\lambda_1, \dots, \lambda_p$, then

$\vec{w}_{(a)}$ *is the eigenvector corresponding to the a-th largest eigenvector* λ_a.

Moreover,

$$\mathrm{Var}(y'_{[i](a)}) = \vec{w}^T_{(a)} C_{[i]} \; \vec{w}_{(a)} = \lambda_a.$$

HOMEWORK

- Prove the PC theorem. Proceed iteratively from $\vec{w}_{(1)}$.

Principal Components Analysis
Dimensionality reduction | Eigenfaces

original images (Olivetti dataset)

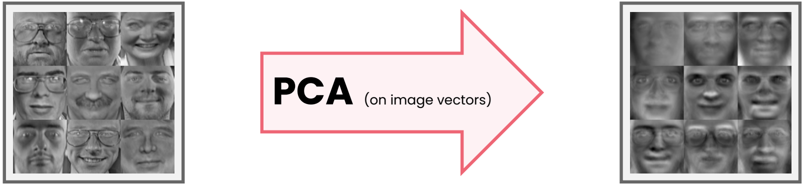
PCA (on image vectors)

eigenface basis

Principal Components Analysis

Dimensionality reduction | Eigenfaces

original images (Olivetti dataset) eigenface basis



1. let $Z \in [0, 255]^2$ be a grayscale image
2. define $\vec{y} = \mathrm{vec}(Z)$, then find *basis* of images $\{\vec{w}_{(1)}, \dots, \vec{w}_{(p)}\}$
3. write each image $\vec{y}_{([i])} = \sum_{k=1}^{p} y'_{[i](k)} \vec{w}_{(k)}$, where $y'_{[i](k)} = \vec{y}_{[i]} \cdot \vec{w}_{(k)}$
4. (optional) use $\vec{y}'_{[i]} \in \mathbb{R}^p$ to train a classifier (see Sirovich and Kirby (1987) and Turk and Petland (1991))

Principal Components Analysis

Dimensionality reduction | Eigenfaces

original images (Olivetti dataset)

PCA (on image vectors)

eigenface basis

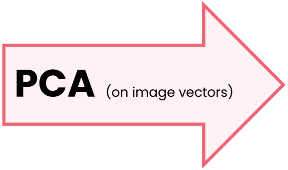
As $\lambda_{(k)}$ represents the **variance explained** by the k–th principal component:

$$\vec{y} \; W = \vec{y}'$$

Principal Components Analysis

Dimensionality reduction | Eigenfaces

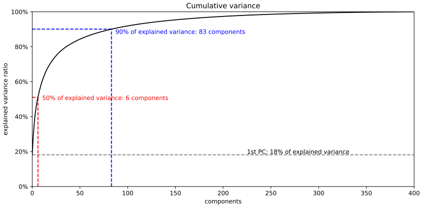
original images (Olivetti dataset)



eigenface basis

As $\lambda_{(k)}$ represents the **variance explained** by the k-th principal component:

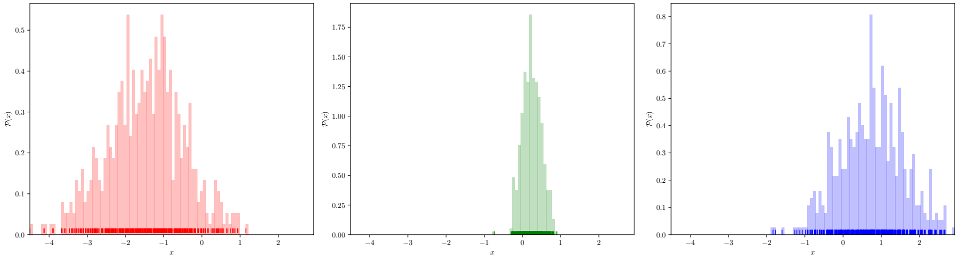
$$\vec{y}\; W_\alpha = \vec{y}'_\alpha$$



Gaussian Mixture Model

Definition

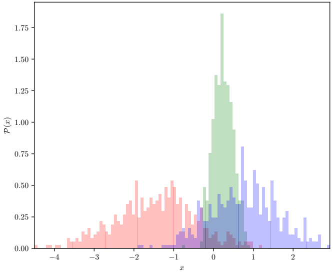
Let $\mathcal{D} = \{x_i \in \mathbb{R} \mid i = 1, 2, \dots, n\}$ s.t.



hypothesis: *data are sampled from different Gaussian distributions*
objective: can we group data according to the parameters of different Gaussian distributions?

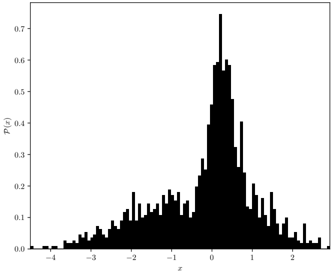
Gaussian Mixture Model

Definition



It looks easy knowing the ground truth...

Gaussian Mixture Model
Definition

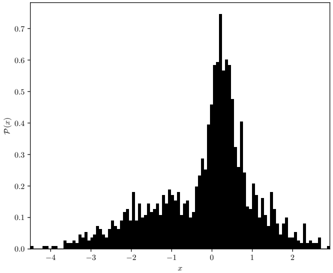


We need to build several normal distributions, then select what we need $\rightarrow$ build a *Gaussian Mixture Model*

…but it is not! (even 1D!)

Gaussian Mixture Model

Definition



...but it is not! (even 1D!)

We need to build several normal distributions, then select what we need → build a *Gaussian Mixture Model*

Reminders

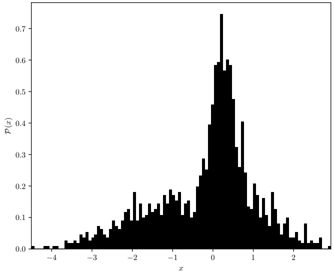
Remember:

$$\mathcal{N}(x \mid \mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}.$$

and (Bayes' theorem):

$$\mathcal{P}(A \mid B) = \frac{\mathcal{P}(B \mid A)\mathcal{P}(A)}{\mathcal{P}(B)}.$$

Gaussian Mixture Model
Definition



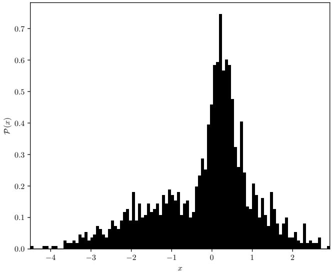
...but it is not! (even 1D!)

Gaussian Mixture Model

Consider $K > 1$ components and let $\Theta = \{(c_{(1)}, \mu_1, \sigma_1^2), \dots, (c_{(K)}, \mu_{(K)}, \sigma_{(K)}^2)\}$, where $c_{(k)}$ $(k = 1, 2, \dots, K)$ are the probabilities of picking the k-th Gaussian.

Gaussian Mixture Model

Definition



…but it is not! (even 1D!)

Gaussian Mixture Model

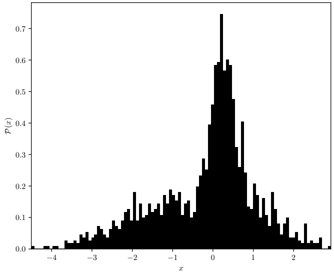
Consider $K > 1$ components and let $\Theta = \{(c_{(1)}, \mu_1, \sigma_1^2), \dots, (c_{(K)}, \mu_{(K)}, \sigma_{(K)}^2)\}$, where $c_{(k)}$ $(k = 1, 2, \dots, K)$ are the probabilities of picking the k-th Gaussian.

We observe the *marginal* (*C* is latent) likelihood:

$$\mathcal{P}(x \mid \Theta) = \sum_{k=1}^{K} \mathcal{P}(C = k \mid \Theta)\, \mathcal{P}(x \mid C = k; \Theta)$$

$$= \sum_{k=1}^{K} c_{(k)}\, \mathcal{N}(x \mid \mu_{(k)}, \sigma_{(k)}^2).$$

Gaussian Mixture Model
Definition



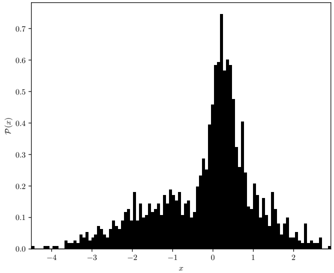
$$\mathcal{P}(x \mid \Theta) = \sum_{k=1}^{K} \mathcal{P}(C = k \mid \Theta)\, \mathcal{P}(x \mid C = k;\ \Theta)$$
$$= \sum_{k=1}^{K} c_{(k)}\ \mathcal{N}(x \mid \mu_{(k)}, \sigma^2_{(k)}).$$

Gaussian Mixture Model

Consider $K > 1$ components and let $\Theta = \{(c_{(1)}, \mu_1, \sigma_1^2), \dots, (c_{(K)}, \mu_{(K)}, \sigma^2_{(K)})\}$, where $c_{(k)}$ $(k = 1, 2, \dots, K)$ are the probabilities of picking the k-th Gaussian.

Gaussian Mixture Model

Definition



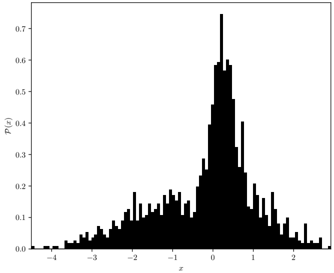
$$\mathcal{P}(x \mid \Theta) = \sum_{k=1}^{K} \mathcal{P}(C = k \mid \Theta)\, \mathcal{P}(x \mid C = k;\ \Theta)$$
$$= \sum_{k=1}^{K} c_{(k)}\ \mathcal{N}(x \mid \mu_{(k)}, \sigma^2_{(k)}).$$

Gaussian Mixture Model

How to estimate the parameters? Ideally, we would like to assign any sample $x \in \mathcal{D}$ to a generating distribution:

$$\Theta^* = \arg\max_{\Theta} \prod_{i=1}^{n} \mathcal{P}(x_i \mid \Theta)$$
$$= \arg\max_{\Theta} \sum_{i=1}^{n} \ln \mathcal{P}(x_i \mid \Theta)$$

Gaussian Mixture Model
Definition



$$\mathcal{P}(x \mid \Theta) = \sum_{k=1}^{K} \mathcal{P}(C = k \mid \Theta)\, \mathcal{P}(x \mid C = k;\ \Theta)$$
$$= \sum_{k=1}^{K} c_{(k)}\ \mathcal{N}(x \mid \mu_{(k)}, \sigma^2_{(k)}).$$

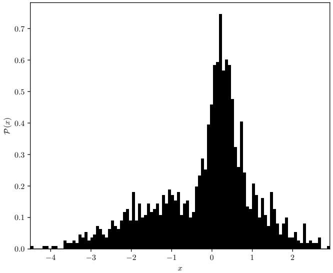
Gaussian Mixture Model

How to estimate the parameters? Ideally, we would like to assign any sample $x \in \mathcal{D}$ to a generating distribution:

$$\Theta^* = \arg\max_{\Theta} \prod_{i=1}^{n} \mathcal{P}(x_i \mid \Theta)$$
$$= \arg\max_{\Theta} \sum_{i=1}^{n} \ln \mathcal{P}(x_i \mid \Theta)$$

Use the **Expectation-Maximisation** (EM) algorithm. (EM is a technique to estimate the MLE of a latent variable model)

Gaussian Mixture Model
Definition



$$\mathcal{P}(x \mid \Theta) = \sum_{k=1}^{K} \mathcal{P}(C = k \mid \Theta)\, \mathcal{P}(x \mid C = k;\ \Theta)$$
$$= \sum_{k=1}^{K} c_{(k)}\ \mathcal{N}(x \mid \mu_{(k)}, \sigma^2_{(k)}).$$

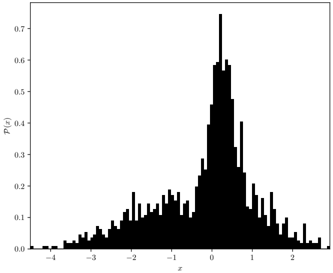
Expectation Maximisation

Consider

$$\mathcal{P}(C = k \mid x_i; \Theta) = \frac{P(x_i \mid C = k; \Theta) P(C = k)}{P(x_i \mid \Theta)}$$
$$= \frac{P(x_i \mid C = k; \Theta) P(C = k)}{\sum_{k=1}^{K} P(x_i \mid C = k; \Theta) P(C = k)}$$
$$= \frac{c_{(k)}\ \mathcal{N}(x_i \mid \mu_{(k)}, \sigma^2_{(k)})}{\sum_{k=1}^{K} c_{(k)}\ \mathcal{N}(x_i \mid \mu_{(k)}, \sigma^2_{(k)})} = \gamma_{i(k)}$$

N.B.: $\sum_{k=1}^{K} \gamma_{i(k)} = 1.$

Gaussian Mixture Model
Definition



$$\mathcal{P}(x \mid \Theta) = \sum_{k=1}^{K} \mathcal{P}(C = k \mid \Theta)\, \mathcal{P}(x \mid C = k;\ \Theta)$$
$$= \sum_{k=1}^{K} c_{(k)}\ \mathcal{N}(x \mid \mu_{(k)}, \sigma^2_{(k)}).$$

Expectation Maximisation

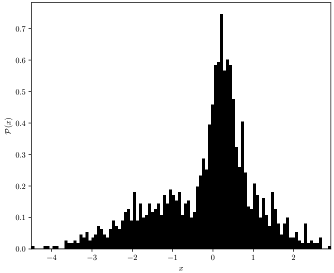
$$\mathrm{Q}(\Theta) = \mathbb{E}_{\mathcal{P}(C \mid x;\Theta)} \left[\ln \mathcal{P}(x \mid \Theta)\right]$$
$$= \sum_{i=1}^{n} \sum_{k=1}^{K} \mathcal{P}(C = k \mid x_i; \Theta) \ln \mathcal{P}(x_i \mid \Theta)$$
$$= \sum_{i=1}^{n} \sum_{k=1}^{K} \gamma_{i(k)} \ln \left(c_{(k)} \mathcal{N}(x_i \mid \mu_{(k)}, \sigma^2_{(k)}) \right)$$

Compute (until convergence to $\Theta^{\{t\}} \overset{t \gg 1}{\longrightarrow} \Theta^*$):

$$\Theta^{\{t+1\}} = \arg\max_{\Theta} \mathrm{Q}\left(\Theta^{\{t\}}\right)$$

Gaussian Mixture Model

Definition



$$\mathcal{P}(x \mid \Theta) = \sum_{k=1}^{K} \mathcal{P}(C = k \mid \Theta)\, \mathcal{P}(x \mid C = k;\ \Theta)$$
$$= \sum_{k=1}^{K} c_{(k)}\ \mathcal{N}(x \mid \mu_{(k)}, \sigma^2_{(k)}).$$

Expectation Maximisation

Compute (until convergence to $\Theta^{\{t\}} \overset{t \gg 1}{\longrightarrow} \Theta^*$):

$$\Theta^{\{t+1\}} = \arg\max_{\Theta} \mathrm{Q}\left(\Theta^{\{t\}}\right),$$

that is:

$$c^{\{t+1\}}_{(k)} = \mathbb{E}_{\mathcal{D}}\left[\gamma^{\{t\}}_{(k)}\right]$$

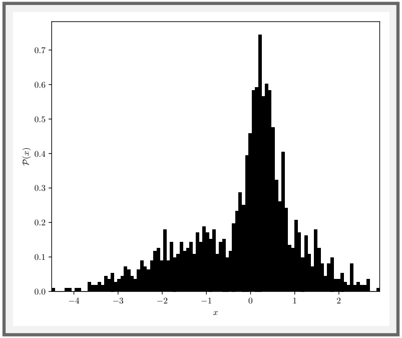
$$\mu^{\{t+1\}}_{(k)} = \frac{\mathbb{E}_{\mathcal{D}}\left[\gamma^{\{t\}}_{(k)} x\right]}{\mathbb{E}_{\mathcal{D}}\left[\gamma^{\{t\}}_{(k)}\right]}$$

$$\sigma^{\{t+1\}}_{(k)} = \frac{\mathbb{E}_{\mathcal{D}}\left[\gamma^{\{t\}}_{(k)} \left(x - \mu_{(k)}\right)^2\right]}{\mathbb{E}_{\mathcal{D}}\left[\gamma^{\{t\}}_{(k)}\right]}$$

Gaussian Mixture Model
Unsupervised learning | Clustering

Remember

$$\gamma^*_{i(k)} = \mathcal{P}(C = k \mid x_i; \Theta^*) \Rightarrow \widehat{C}_i = \arg\max_k \mathrm{softmax}\left(\gamma^*_{i(k)}\right)$$

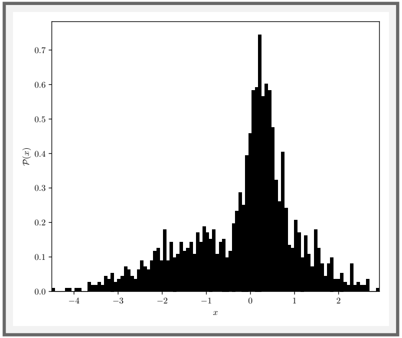


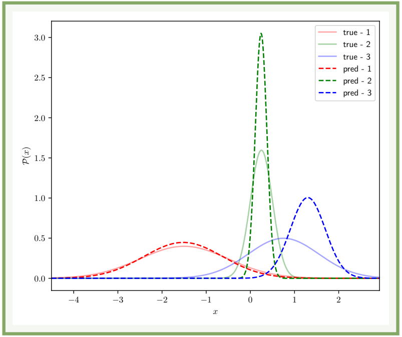
Gaussian Mixture Model

Unsupervised learning | Clustering

Remember

$$\gamma^*_{i(k)} = \mathcal{P}(C = k \mid x_i; \Theta^*) \Rightarrow \widehat{C}_i = \arg\max_k \operatorname{softmax}\left(\gamma^*_{i(k)}\right)$$





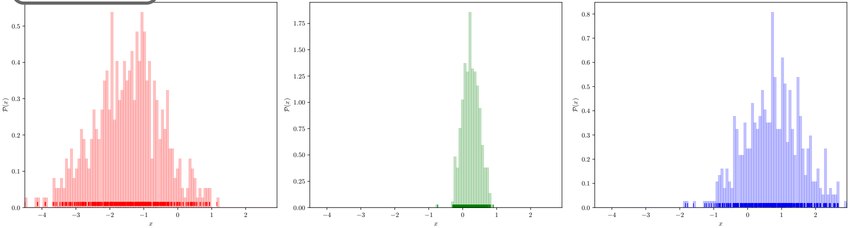
Gaussian Mixture Model

Unsupervised learning | Clustering

Remember

$$\gamma^*_{i(k)} = \mathcal{P}(C = k \mid x_i; \Theta^*) \Rightarrow \widehat{C}_i = \arg\max_k \operatorname{softmax}\left(\gamma^*_{i(k)}\right)$$

ORIGINAL

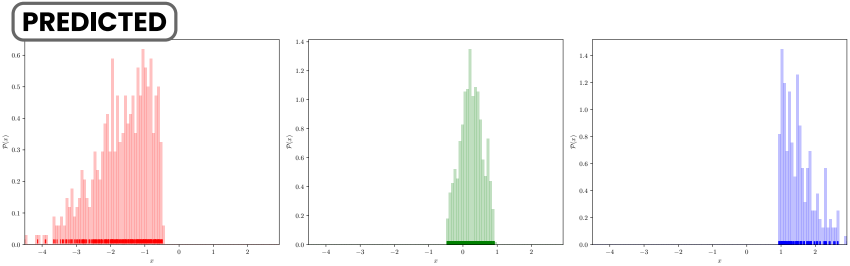


Gaussian Mixture Model

Unsupervised learning | Clustering

Remember

$$\gamma^*_{i(k)} = \mathcal{P}\left(C = k \mid x_i; \Theta^*\right) \Rightarrow \widehat{C}_i = \arg\max_k \operatorname{softmax}\left(\gamma^*_{i(k)}\right)$$

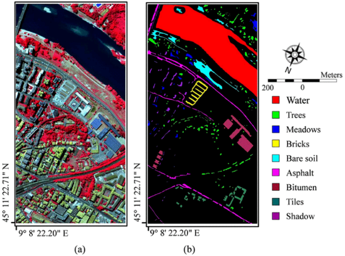


Gaussian Mixture Model

Unsupervised learning | Clustering

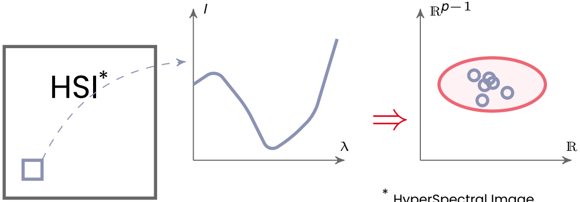
Remember

$$\gamma^*_{i(k)} = \mathcal{P}(C = k \mid x_i; \Theta^*) \Rightarrow \widehat{C}_i = \arg\max_k \operatorname{softmax}\left(\gamma^*_{i(k)}\right)$$



Zhao et al. (2016)

- can generalise to N-dimensional distributions
- used for **exploratory** data analysis...
- ...as well as unsupervised **classification**



* HyperSpectral Image

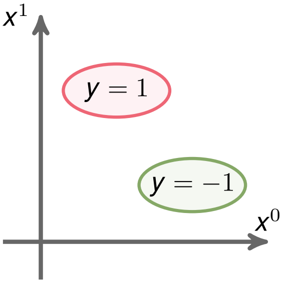
3. ML Algorithms

Supervised learning

Table of contents

Support Vector Machine

Definition | The classification case

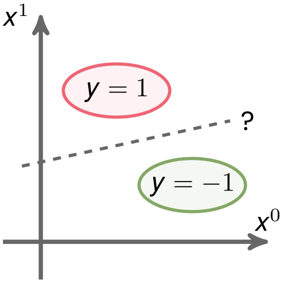


(see Cortes and Vapnik (1995))

Suppose $\mathcal{D} = \{(\vec{x}, y)\}$, s.t. $y = \pm 1$ (classification) (simple case: linearly separable)

Support Vector Machine

Definition | The classification case

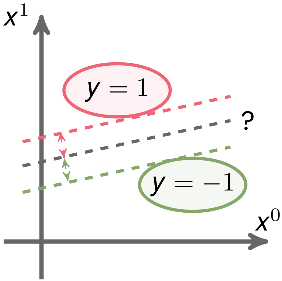


(see Cortes and Vapnik (1995))

How to choose the best **hyperplane** to separate the points?

Support Vector Machine

Definition | The classification case

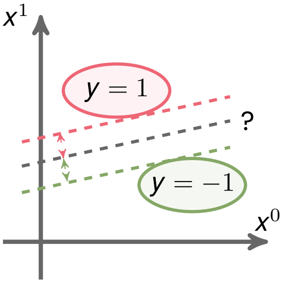


(see Cortes and Vapnik (1995))

Maximise the separation (i.e. the **margin**)!

Support Vector Machine

Definition | The classification case



(see Cortes and Vapnik (1995))

Maximise the separation (i.e. the **margin**)!

Hyperplane (strict) separation theorem

Let A, $B \subset \mathbb{R}^p$ s.t. they are **closed** and **convex**, and $A \cap B = \emptyset$. Suppose one of them is **compact**. Then $\exists \vec{w} = (\vec{a}, b) \in \mathbb{R}^p \times \mathbb{R}$ s.t.

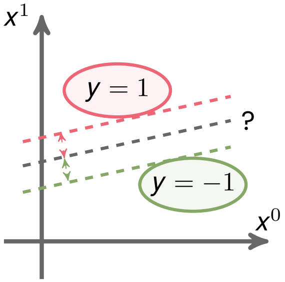
$$\vec{a} \cdot \vec{x} + b \begin{cases} > c_1 & \forall x \in A \\ < c_2 & \forall x \in B \end{cases}$$

for $c_1 > c_2$. That is,

provided a separation, there exist a hyperplane separating the two sets

Support Vector Machine

Definition | The classification case



(see Cortes and Vapnik (1995))

Maximise the separation (i.e. the **margin**)!

Hyperplane (strict) separation theorem

Let A, $B \subset \mathbb{R}^p$ s.t. they are **closed** and **convex**, and $A \cap B = \emptyset$. Suppose one of them is **compact**. Then $\exists \vec{w} = (\vec{a}, b) \in \mathbb{R}^p \times \mathbb{R}$ s.t.

$$\vec{a} \cdot \vec{x} + b \begin{cases} > c_1 & \forall x \in A \\ < c_2 & \forall x \in B \end{cases}$$

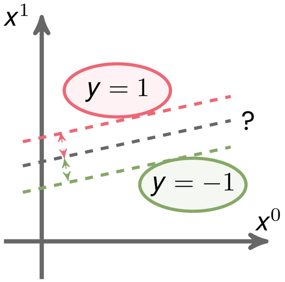
for $c_1 > c_2$. That is,

$$\exists f : \begin{array}{ccc} \mathbb{R}^p & \to & \{-1, +1\} \\ \vec{x} & \mapsto & y \end{array}$$

where y is a *boolean* variable.

Support Vector Machine

Definition | The classification case



(see Cortes and Vapnik (1995))

Support Vector Machine

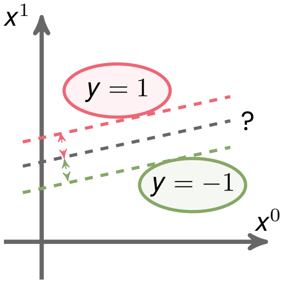
Let $\mathcal{D} = \{(\vec{x}_i, y_i) \in \mathbb{R}^p \times \{-1, 1\} \mid i = 1, 2, \dots, n\}$ **linearly separable**. Let $(\vec{a}, b) \in \mathbb{R}^p \times \mathbb{R}$ identify a hyperplane.
Then, for $i = 1, 2, \dots, n$:

$$\vec{a} \cdot \vec{x}_i + b \begin{cases} \geq +1 & \text{if } y_i = +1 \\ \leq -1 & \text{if } y_i = -1 \end{cases}$$

Maximise the separation (i.e. the **margin**)!

Support Vector Machine

Definition | The classification case



(see Cortes and Vapnik (1995))

Support Vector Machine

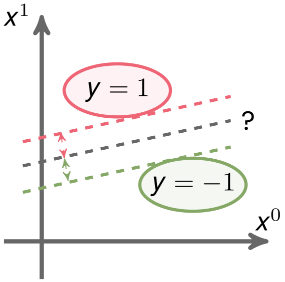
Let $\mathcal{D} = \{(\vec{x}_i, y_i) \in \mathbb{R}^p \times \{-1, 1\} \mid i = 1, 2, \dots, n\}$ **linearly separable**. Let $(\vec{a}, b) \in \mathbb{R}^p \times \mathbb{R}$ identify a hyperplane.
Then, for $i = 1, 2, \dots, n$:

$$y_i \left(\vec{a} \cdot \vec{x}_i + b \right) \geq 1.$$

Maximise the separation (i.e. the **margin**)!

Support Vector Machine

Definition | The classification case



(see Cortes and Vapnik (1995))

Maximise the separation (i.e. the **margin**)!

Support Vector Machine

Let $\mathcal{D} = \{(\vec{x}_i, y_i) \in \mathbb{R}^p \times \{-1, 1\} \mid i = 1, 2, \dots, n\}$ **linearly separable**. Let $(\vec{a}, b) \in \mathbb{R}^p \times \mathbb{R}$ identify a hyperplane.
The distance between classes

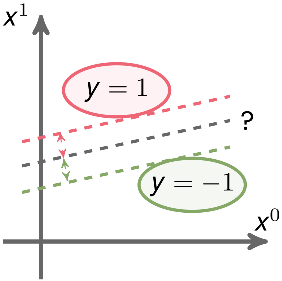
$$\rho(\vec{a}) = \min_{\vec{x}|y=+1} \frac{\vec{a}}{\left|\left|\vec{a}\right|\right|_2} \cdot \vec{x} - \max_{\vec{x}|y=-1} \frac{\vec{a}}{\left|\left|\vec{a}\right|\right|_2} \cdot \vec{x}$$

becomes maximal for the **optimal** $(\vec{a}^*, b^*)$:

$$\rho(\vec{a}^*) = \frac{2}{\left|\left|\vec{a}\right|\right|_2}.$$

Support Vector Machine

Definition | The classification case



(see Cortes and Vapnik (1995))

Maximise the separation (i.e. the **margin**)!

Support Vector Machine

$$(\vec{a}^*, b^*) = \underset{\vec{a}\in\mathbb{R}^p,\ b\in\mathbb{R}}{\arg\max}\ \rho(\vec{a}) = \underset{\vec{a}\in\mathbb{R}^p,\ b\in\mathbb{R}}{\arg\min}\ \frac{1}{2}\,||\vec{a}||_2^2$$

constrained to

$$y_i(\vec{a}\cdot\vec{x}_i + b) \geq 1\ \forall i = 1, 2, \dots, n.$$

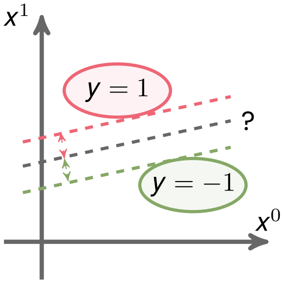
That is, find the **saddle points** of:

$$\mathrm{L}(\vec{a}, b, \vec{\mu}) = \frac{1}{2}\vec{a}\cdot\vec{a} - \sum_{i=0}^{n-1} \mu_i\left(y_i(\vec{a}\cdot\vec{x}_i + b) - 1\right)$$

s.t. $\mu_i \geq 0\ \forall i = 1, 2, \dots, n$ (Lagrange multipliers $\Rightarrow$ find the min for $\vec{a}$ and b, and the max for $\vec{\mu}$).

Support Vector Machine

Definition | The classification case



(see Cortes and Vapnik (1995))

Maximise the separation (i.e. the **margin**)!

Support Vectors

Notice that $\mu_i \geq 0 \; \forall i = 1, 2, \ldots, n$ is fundamental. From the minimisation (equation of motion), the constraint:

$$\mu_i \left(y_i(\vec{a} \cdot \vec{x}_i + b) - 1 \right) = 0$$

implies that

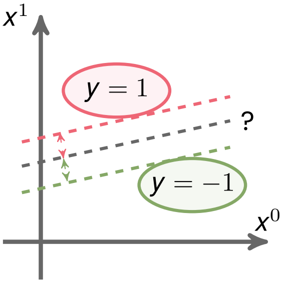
$$\mu_i > 0 \Rightarrow y_i(\vec{a} \cdot \vec{x}_i + b) = 1$$

$\forall i = 1, 2, \ldots, n$. That is,

the only contributing data points are those precisely on the margin $\Rightarrow$ ***Support Vectors***

Support Vector Machine

Definition | The classification case



(see Cortes and Vapnik (1995))

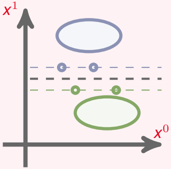
Maximise the separation (i.e. the **margin**)!

Support Vector Machine

$$(\vec{a}^*, b^*) = \arg\min_{\vec{a}\in\mathbb{R}^p,\, b\in\mathbb{R}} \arg\max_{\vec{\mu}\in\mathbb{R}^n} \mathrm{L}(\vec{a}, b, \vec{\mu})$$

can be performed to give:

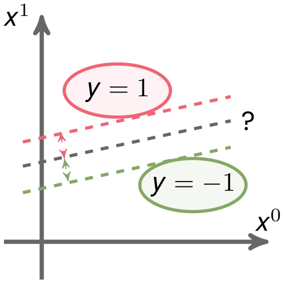
$$\vec{a}^* = \sum_{i=1}^{n} y_i\, \mu_i\, \vec{x}_i \quad \text{and} \quad b^* = -\vec{a} \cdot \vec{x}_i.$$



Hard Margin
Support Vector Machine

Support Vector Machine

Definition | The classification case



(see Cortes and Vapnik (1995))

Maximise the separation (i.e. the **margin**)!

Support Vector Machine (v2)

Suppose data is **not** perfectly separable. New **constraint**:

$$y_i \left(\vec{a} \cdot \vec{x}_i + b \right) \geq 1 - \xi_i \quad \xi_i > 0 \quad \forall i = 1, 2, \dots, n.$$

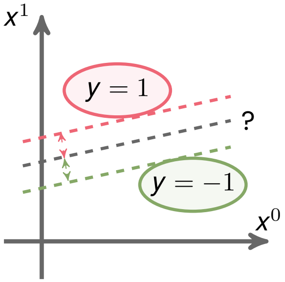
Hence $(C > 0)$:

$$\mathrm{L}(\vec{a}, b, \vec{\mu}, \vec{\xi}) = \underset{\vec{a} \in \mathbb{R}^p,\ b \in \mathbb{R},\ \vec{\xi} \in \mathbb{R}^n}{\arg\min} \frac{1}{2} \vec{a} \cdot \vec{a} + C \sum_{i=1}^{n} \xi_i$$

***Soft Margin** SVM*

Support Vector Machine

Definition | The classification case



(see Cortes and Vapnik (1995))

Maximise the separation (i.e. the **margin**)!

Kernel SVM

Suppose data not (natively) lineary separable in target space, but possibly in **feature space**:

$$\exists\, \phi \colon \mathbb{R}^p \to \mathbb{R}^P, \quad P > p$$

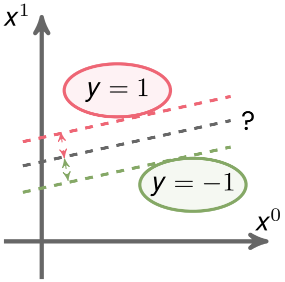
s.t. we can use SVM in P-dimensional space:

$$\vec{a}^* = \sum_{i=1}^{n} y_i \mu_i \phi(\vec{x}_i).$$

Do we need to know ϕ analytically?

Support Vector Machine

Definition | The classification case



(see Cortes and Vapnik (1995))

Maximise the separation (i.e. the **margin**)!

Kernel SVM

Let f be the SVM classifier and $\vec{x}$ a new data point:

$$f(\vec{x}) = \phi(\vec{x}) \cdot \vec{a}^* + b^*$$

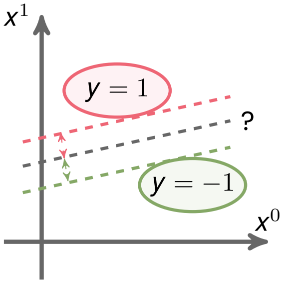
$$= \sum_{i=1}^{n} y_i \, \mu_i \, \phi(\vec{x}) \cdot \phi(\vec{x}_i) + b^* .$$

Only projections on *support vectors*:

$$K(\vec{x}, \vec{x}_i) = \phi(\vec{x}) \cdot \phi(\vec{x}_i)$$

are really necessary (not even all data points!).

Support Vector Machine
Definition | The classification case



(see Cortes and Vapnik (1995))

Maximise the separation (i.e. the **margin**)!

Kernel SVM

As long as $K \in L_2(\mathbb{R}^P)$ and symmetric:

$$K(\vec{u}, \vec{v}) = \sum_{i=1}^{\infty} \lambda_i \varphi_i(\vec{u}) \cdot \varphi_j(\vec{v}), \quad \lambda_i \geq 0.$$

Good/used choices of K:

- *radial basis func.:* $K_\sigma(\vec{u}, \vec{v}) = \exp(-\frac{||u-v||}{\sigma^2})$
- polynomial func.: $K_d(\vec{u}, \vec{v}) = (1 + \vec{u} \cdot \vec{v})^d$

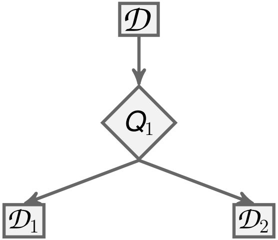
HOMEWORK

1. build a SVM for regression (change the classifier constraint to a MSE loss, see Drucker et al. (1996))
2. show that the *sigmoid* can be used as a kernel function

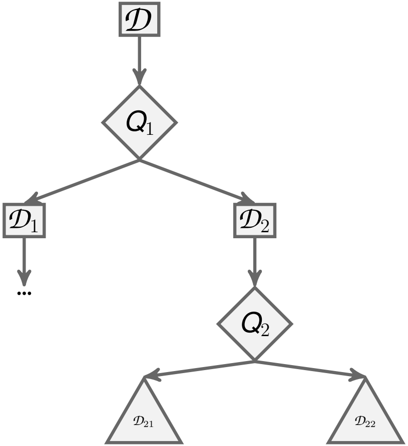
Decision Trees
Definition

$\boxed{\mathcal{D}}$

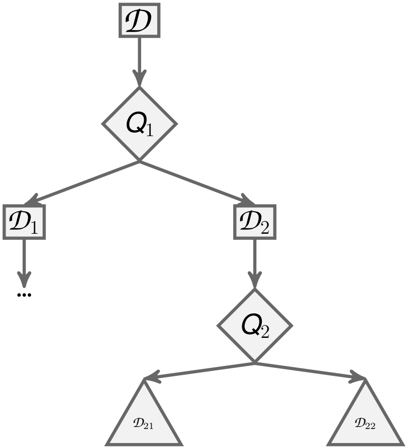
Decision Trees
Definition



Decision Trees
Definition



Decision Trees
Definition



A **decision tree** $\mathscr{T}$ is a *hierarchical* model:

- **decision nodes** where "splits" are made
- **data nodes**
  - *branches* parent partitions of data
  - *leaves* final partitions of data

That is $\mathscr{T}$ (slight abuse of notation: $\mathcal{D}$ is both the data and the domain):

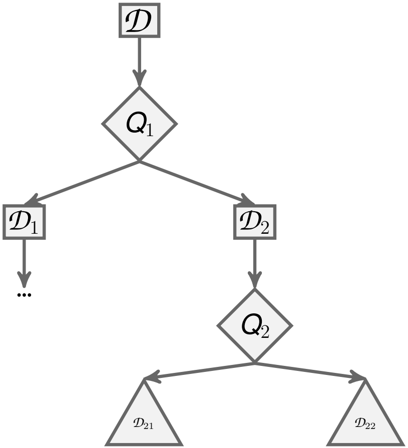
$$\mathscr{T} = \left\{ \mathcal{D}_A \subset \mathcal{D} \mid \bigcup_A \mathcal{D}_A = \mathcal{D},\ \mathcal{D}_A \cap \mathcal{D}_B = \emptyset \text{ for } A \neq B \right\}$$

where

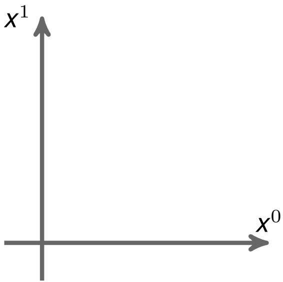
$$A = \{a_1, a_2, \dots, a_n\}$$

s.t. $|A|$ is maximal w.r.t. a *stopping criterion*.

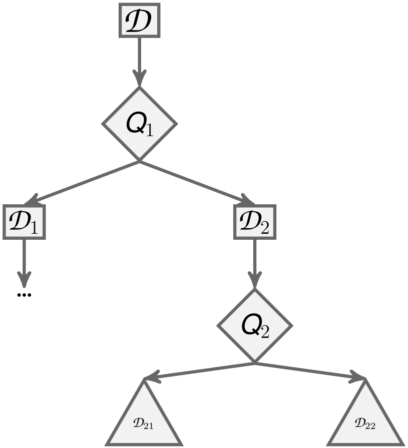
Decision Trees
Definition



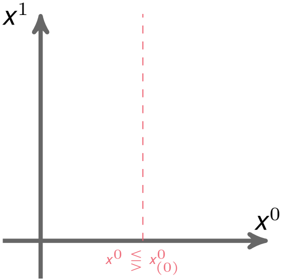
A **decision tree** $\mathscr{T}$ is *piecewise linear* as it outputs a **tesselation** of the domain space:



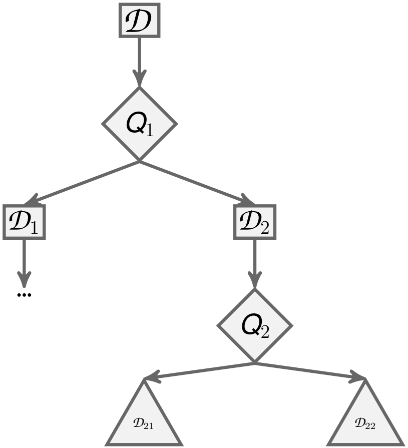
Decision Trees
Definition



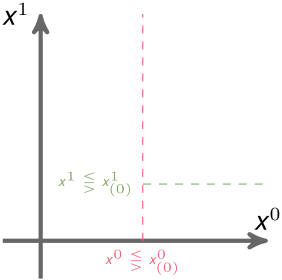
A **decision tree** $\mathscr{T}$ is *piecewise linear* as it outputs a **tesselation** of the domain space:



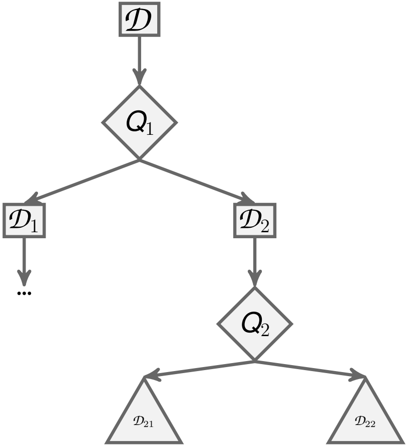
Decision Trees
Definition



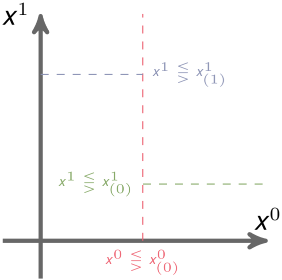
A **decision tree** $\mathscr{T}$ is *piecewise linear* as it outputs a **tesselation** of the domain space:



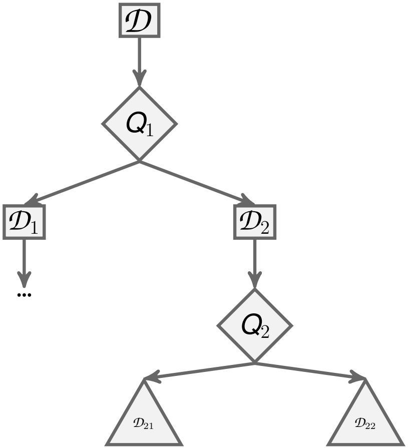
Decision Trees
Definition



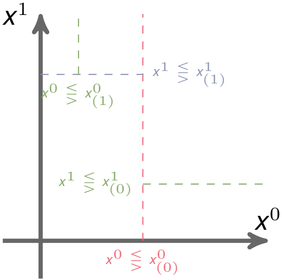
A **decision tree** $\mathscr{T}$ is *piecewise linear* as it outputs a **tesselation** of the domain space:



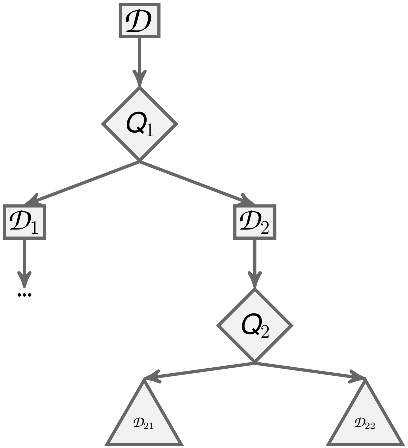
Decision Trees
Definition



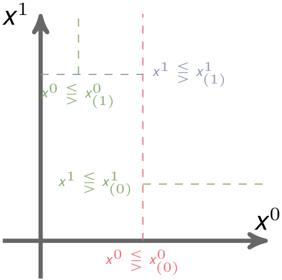
A **decision tree** $\mathscr{T}$ is *piecewise linear* as it outputs a **tesselation** of the domain space:



Decision Trees
Definition



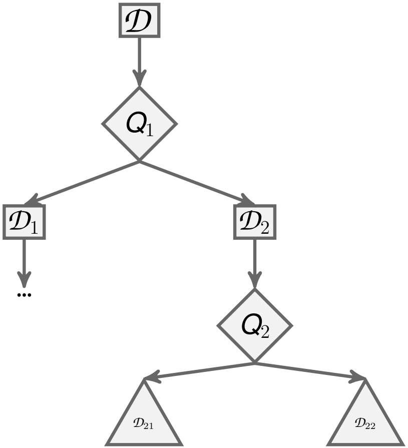
A **decision tree** $\mathscr{T}$ is *piecewise linear* as it outputs a **tesselation** of the domain space:



There exists a huge number of decision tree making algorithms (THAID, C4.5, CART, MARS, etc.) $\rightarrow$ we focus on CART and C4.5.

Decision Trees

Definition



In other words:

Require: stopping criterion $\mathscr{S}$, measure of "goodness of split" $\mathcal{G}$, $i \leftarrow 0$

while $\neg\mathscr{S}$ **do**

  loop

    select node i

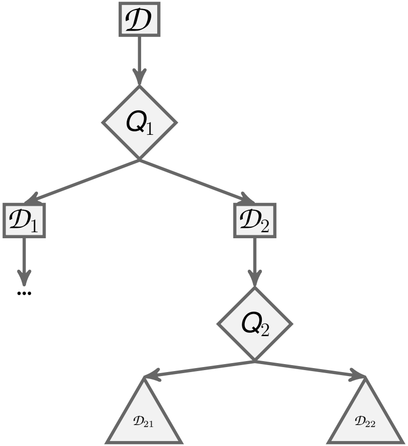
    find the best partition of $\mathcal{D}_i$ according to $\mathcal{G}$

    create child nodes i_a $(a = i1, i2, \dots)$

    loop for each child node $i \leftarrow i_a$

  return partition assignments of $\mathcal{D}$

Decision Trees
Definition



The C4.5 decision tree

Use **information gain** (mutual information) as criterion $\mathcal{G}$ (maximise):

$$\mathrm{MI}(X \in \mathcal{D}, X_i \in \mathcal{D}_A) = \mathrm{H}(X \in \mathcal{D}) - \mathrm{H}(X \in \mathcal{D} \mid X_i \in \mathcal{D}_A),$$

where

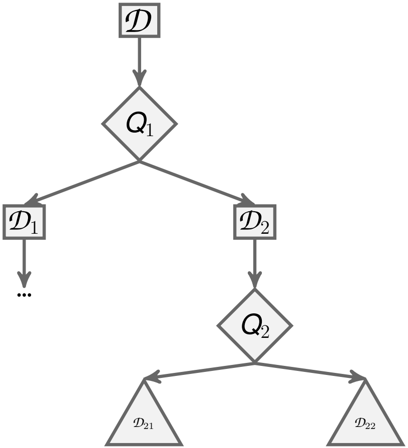
$$\mathrm{H}(X) = -\sum_{j=1}^{K} \mathcal{P}(X \in C_j) \log_2 \mathcal{P}(X \in C_j),$$

$$\mathrm{H}(X \mid X_i) = -\sum_{j=1}^{K} \mathcal{P}(X \wedge X_j) \log_2 \frac{\mathcal{P}(X \wedge X_j)}{\mathcal{P}(X \in C_j)},$$

and $\mathcal{P}(X \wedge X_j) = \mathcal{P}(X \in \mathcal{D}, X_j \in C_j)$ and $C_j \subset \mathcal{D}$.

(see Quinlan (1994))

Decision Trees
Definition



The **C4.5** decision tree

Use **information gain** (mutual information) as criterion $\mathcal{G}$ (maximise):

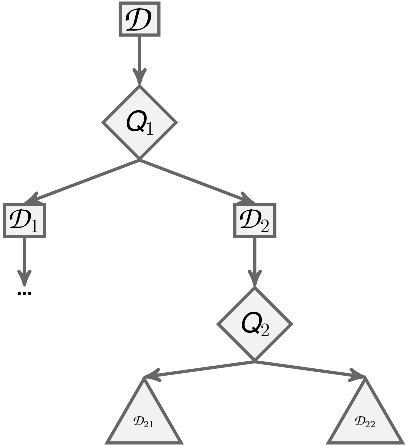
$$\mathrm{MI}(X \in \mathcal{D}, X_i \in \mathcal{D}_A) = \mathrm{H}(X \in \mathcal{D}) - \mathrm{H}(X \in \mathcal{D} \mid X_i \in \mathcal{D}_A),$$

Which implies:

- **maximise** the information acquired by the split
- **pruning** based on informative splits
  (uninformative *branches* are replaced by *leaf* nodes)
- **missing values** automatically handled
- the split can be **arbitrary** (e.g. multiclass)

(see Quinlan (1994))

Decision Trees
Definition



The **Classification And Regression Trees**

Use **Gini impurity** as $\mathcal{G}$ (minimise).
Let p_i, $i = 1, 2, \ldots, K$, be the probability of choosing an item of class C_i:

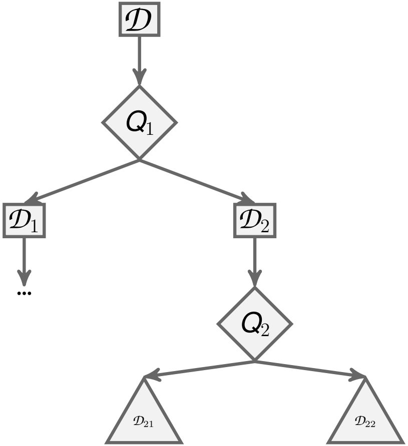
$$\mathrm{I}(X \in \mathcal{D}_A) = \sum_{i=1}^{K} p_i (1 - p_i) = 1 - \sum_{i=1}^{K} p_i^2 ,$$

that is

the probability of incorrectly classifying an item, if it were randomly labelled based on the distribution of the sample.

(see Breiman et al. (1984))

Decision Trees
Definition



The **Classification And Regression Trees**

Use **Gini impurity** as $\mathcal{G}$ (minimise).
Let p_i, $i = 1, 2, \dots, K$, be the probability of choosing an item of class C_i:

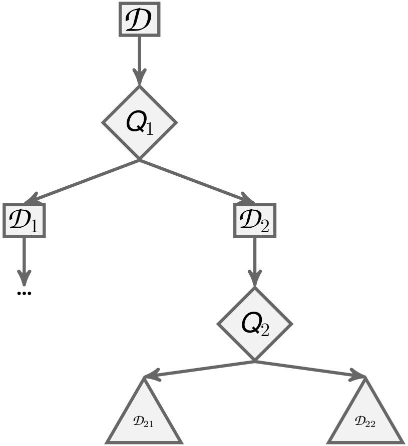
$$\mathrm{I}(X \in \mathcal{D}_A) = \sum_{i=1}^{K} p_i(1 - p_i) = 1 - \sum_{i=1}^{K} p_i^2 ,$$

that is

the **Tsallis entropy** (generalised *Boltzmann-Gibbs*) with deformation 2.

(see Breiman et al. (1984))

Decision Trees
Definition



The **Classification And Regression Trees**

Use **Gini impurity** as $\mathcal{G}$ (minimise).
Let p_i, $i = 1, 2, \dots, K$, be the probability of choosing an item of class C_i:

$$\mathrm{I}(X \in \mathcal{D}_A) = \sum_{i=1}^{K} p_i (1 - p_i) = 1 - \sum_{i=1}^{K} p_i^2 ,$$

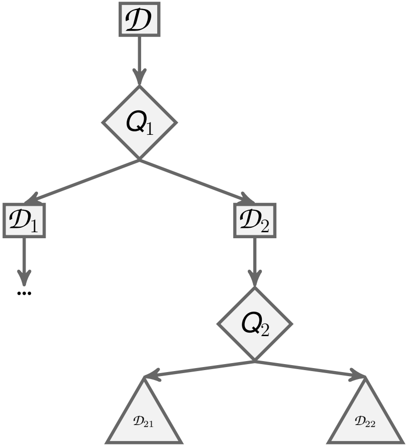
which enables:

- **binary** partitions of $\mathcal{D}_A$
- **pruning** to be enforced (e.g. cross-validation)
- **label** = **mode** of leaf node (*mean*/*median*)

(see Breiman et al. (1984))

Decision Trees

Definition

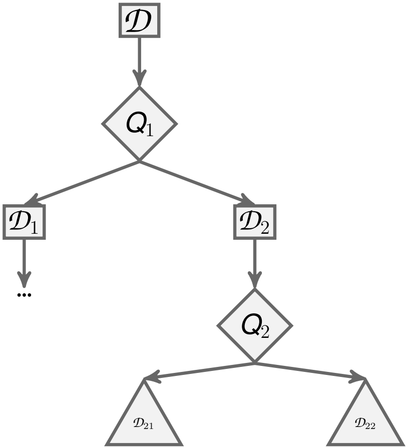


THINK

Suppose no pruning:

1. at which point does $\mathscr{T}$ *naturally* stop?
2. is $\mathscr{T}$ **high bias** or **high variance**?

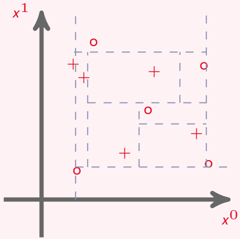
Decision Trees
Definition



THINK

Suppose no pruning:

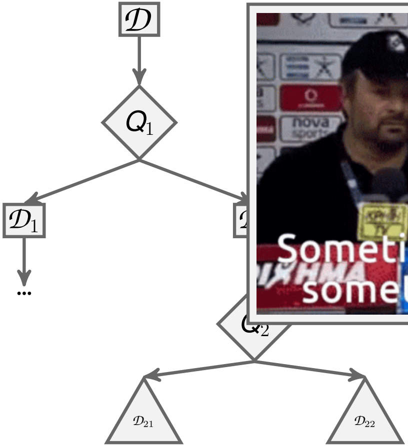
1. at which point does $\mathcal{T}$ *naturally* stop? $\rightarrow$ $|\mathcal{D}_A| = 1, \ \forall A$
2. is $\mathcal{T}$ **high bias** or **high variance**? $\rightarrow$ **VERY high variance**



Decision trees are prone to **overfitting** $\mathcal{D}^{(\text{train})}$ without appropriate strategies!

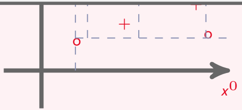
Decision Trees

Definition



$|\mathcal{D}_A| = 1, \; \forall A$

RY high variance



ecision trees are
one to **overfitting**
$\mathcal{D}^{(\text{train})}$ without
appropriate strategies!

Decision Trees
Feature ranking

Hierarchical structure enables **ranking** features $\Rightarrow$ **importance** of feature for the split.

Decision Trees

Feature ranking

Hierarchical structure enables **ranking** features $\Rightarrow$ **importance** of feature for the split.

Let

$$\mathfrak{I}_A = \frac{|\mathcal{D}_A|}{|\mathcal{D}|} \mathsf{I}\left(X \in \mathcal{D}_A\right) - \frac{|\mathcal{D}_{A1}|}{|\mathcal{D}|} \mathsf{I}\left(X \in \mathcal{D}_{A1}\right) - \frac{|\mathcal{D}_{A2}|}{|\mathcal{D}|} \mathsf{I}\left(X \in \mathcal{D}_{A2}\right)$$

the importance of node A.

Decision Trees

Feature ranking

Hierarchical structure enables **ranking** features $\Rightarrow$ **importance** of feature for the split.

Let

$$\mathfrak{I}_A = \frac{|\mathcal{D}_A|}{|\mathcal{D}|} \mathsf{I}\left(X \in \mathcal{D}_A\right) - \frac{|\mathcal{D}_{A1}|}{|\mathcal{D}|} \mathsf{I}\left(X \in \mathcal{D}_{A1}\right) - \frac{|\mathcal{D}_{A2}|}{|\mathcal{D}|} \mathsf{I}\left(X \in \mathcal{D}_{A2}\right)$$

the importance of node A.

Compute the **feature importance** of feature i:

$$\mathsf{F}_i = \frac{\sum_{a \text{ splits on } i} \mathfrak{I}_a}{\sum_a \mathfrak{I}_a}$$

Decision Trees

Feature ranking

Hierarchical structure enables **ranking** features ⇒ **importance** of feature for the split.

```python
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import precision_recall_fscore_support

# Load and split the data
iris = load_iris()
X_train, X_test, y_train, y_test = train_test_split(iris.data,
                                                    iris.target,
                                                    shuffle=True,
                                                    random_state=42)

# Train a decision tree
clf = DecisionTreeClassifier().fit(X_train, y_train)

# Predict the test set
y_test_pred = clf.predict(X_test)
prec, rec, f1, _ = precision_recall_fscore_support(y_test,
                                                   y_test_pred,
                                                   average='macro')
print(f'Precision: {prec:.0%}')
print(f'Recall: {rec:.0%}')
print(f'F1: {f1:.0%}')
```

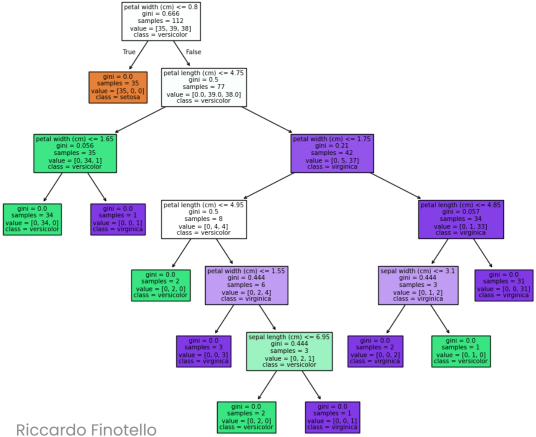
```python
from sklearn.tree import plot_tree

plot_tree(clf,
          feature_names=iris.feature_names,
          class_names=iris.target_names,
          filled=True)
```

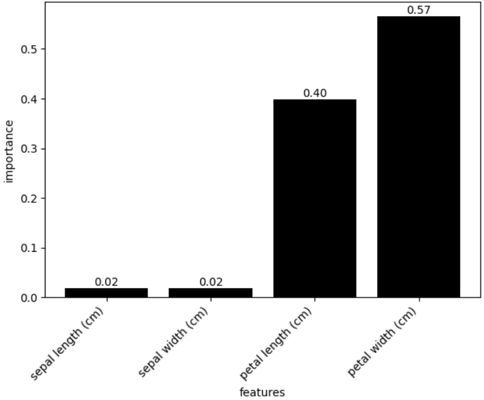
Can you produce/guess the output?

Decision Trees

Feature ranking

Hierarchical structure enables **ranking** features ⇒ **importance** of feature for the split.





3. ML Algorithms

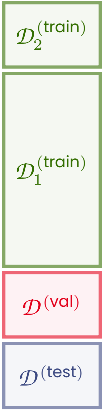
Ensemble learning

Table of contents

Ensemble Learning

Stacking / Metalearning

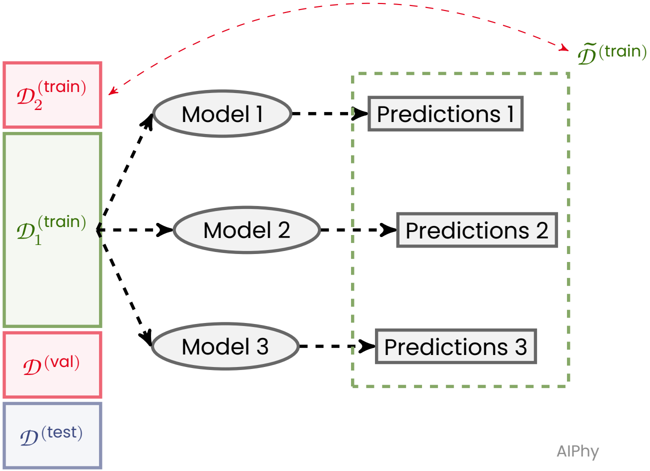
Let $\mathcal{D}_1^{(\text{train})} \cup \mathcal{D}_2^{(\text{train})} = \mathcal{D}^{(\text{train})}$ a partition of the training set:



Ensemble Learning

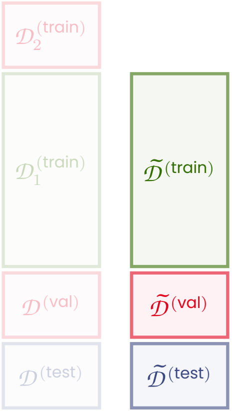
Stacking / Metalearning

Let $\mathcal{D}_1^{(\text{train})} \cup \mathcal{D}_2^{(\text{train})} = \mathcal{D}^{(\text{train})}$ a partition of the training set:



Ensemble Learning
Stacking / Metalearning

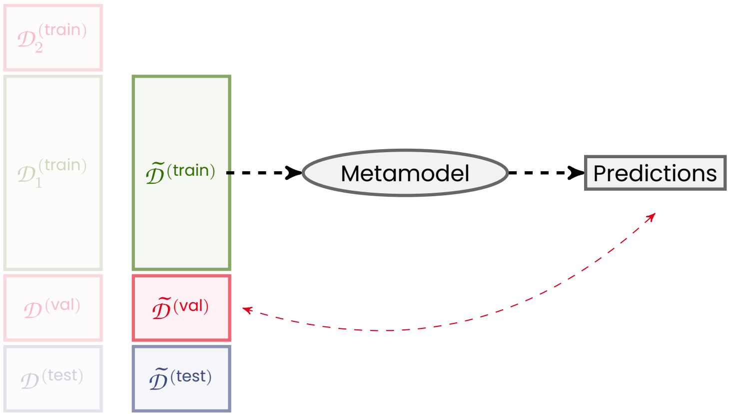
Let $\mathcal{D}_1^{(\text{train})} \cup \mathcal{D}_2^{(\text{train})} = \mathcal{D}^{(\text{train})}$ a partition of the training set:



Ensemble Learning

Stacking / Metalearning

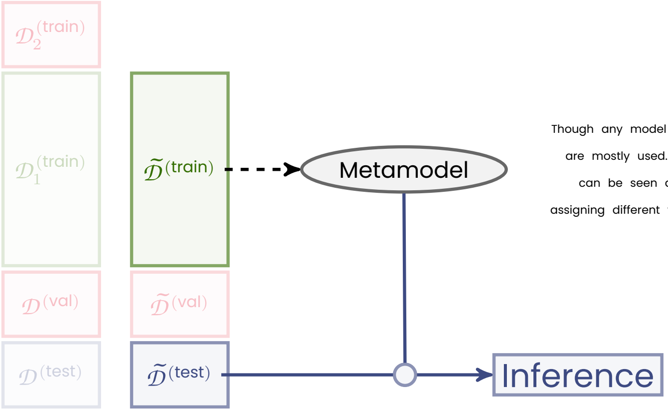
Let $\mathcal{D}_1^{(\text{train})} \cup \mathcal{D}_2^{(\text{train})} = \mathcal{D}^{(\text{train})}$ a partition of the training set:



Ensemble Learning

Stacking / Metalearning

Let $\mathcal{D}_1^{(\text{train})} \cup \mathcal{D}_2^{(\text{train})} = \mathcal{D}^{(\text{train})}$ a partition of the training set:

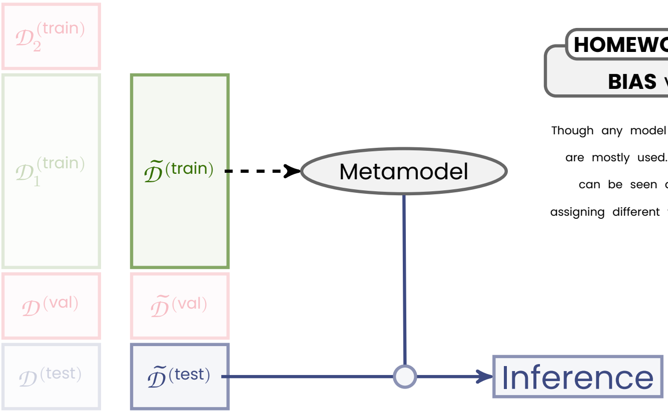


Though any model will do, *linear regression/logistic classification* are mostly used. Thanks to their simple interpretation, they can be seen as a generalisation of *majority voting*, by assigning different weights to the predictions of different models

Ensemble Learning

Stacking / Metalearning

Let $\mathcal{D}_1^{(\text{train})} \cup \mathcal{D}_2^{(\text{train})} = \mathcal{D}^{(\text{train})}$ a partition of the training set:



HOMEWORK

BIAS vs **VARIANCE**?

Though any model will do, *linear regression*/*logistic classification* are mostly used. Thanks to their simple interpretation, they can be seen as a generalisation of *majority voting*, by assigning different weights to the predictions of different models

Ensemble Learning
Bootstrap

Suppose a population with distribution $\mathcal{P}$ for which you need a **statistical estimate** with *expected value* θ, and *variance* σ^2:

1. take a sample $\mathcal{D} = \{X_1 = x_1, \dots, X_n = x_n\}$ with distribution $\widehat{\mathcal{P}}$
2. estimate $\widehat{\theta}$ using $\mathcal{D}$

Ensemble Learning
Bootstrap

Suppose a population with distribution $\mathcal{P}$ for which you need a **statistical estimate** with *expected value* θ, and *variance* σ^2:

1. take a sample $\mathcal{D} = \{X_1 = x_1, \dots, X_n = x_n\}$ with distribution $\widehat{\mathcal{P}}$
2. estimate $\widehat{\theta}$ using $\mathcal{D}$

Should you be able to repeat your estimation, you could compute

$$\mathbb{E}_{\widehat{\mathcal{P}}(X)}[\widehat{\theta}] = \theta \text{ (unbiased estimator)} \quad \mathrm{Var}_{\widehat{\mathcal{P}}(X)}[\widehat{\theta}] = \frac{n-1}{n}\sigma^2 \text{ (biased estimator)}$$

Ensemble Learning
Bootstrap

Suppose a population with distribution $\mathcal{P}$ for which you need a **statistical estimate** with *expected value* θ, and *variance* σ^2:

1. take a sample $\mathcal{D} = \{X_1 = x_1, \dots, X_n = x_n\}$ with distribution $\widehat{\mathcal{P}}$
2. estimate $\widehat{\theta}$ using $\mathcal{D}$

Should you be able to repeat your estimation, you could compute

$$\mathbb{E}_{\widehat{\mathcal{P}}(X)}[\widehat{\theta}] = \theta \text{ (unbiased estimator)} \qquad \mathrm{Var}_{\widehat{\mathcal{P}}(X)}[\widehat{\theta}] = \frac{n-1}{n}\sigma^2 \text{ (biased estimator)}$$

and show

$$\frac{\mathbb{E}_{\widehat{\mathcal{P}}(X)}[\widehat{\theta}] - \theta}{\frac{\sigma}{\sqrt{n}}} \sim \mathcal{N}(0, 1) \text{ (central limit theorem)}.$$

Ensemble Learning

Bootstrap

Suppose a population with distribution $\mathcal{P}$ for which you need a **statistical estimate** with *expected value* θ, and *variance* σ^2:

1. take a sample $\mathcal{D} = \{X_1 = x_1, \dots, X_n = x_n\}$ with distribution $\widehat{\mathcal{P}}$
2. estimate $\widehat{\theta}$ using $\mathcal{D}$

Should you be able to repeat your estimation, you could compute

$$\mathbb{E}_{\widehat{\mathcal{P}}(X)}[\widehat{\theta}] = \theta \text{ (unbiased estimator)} \qquad \mathrm{Var}_{\widehat{\mathcal{P}}(X)}[\widehat{\theta}] = \frac{n-1}{n}\sigma^2 \text{ (biased estimator)}$$

and show

$$\frac{\mathbb{E}_{\widehat{\mathcal{P}}(X)}[\widehat{\theta}] - \theta}{\frac{\sigma}{\sqrt{n}}} \sim \mathcal{N}(0,1) \text{ (central limit theorem).}$$

This might not be possible!

Ensemble Learning
Bootstrap

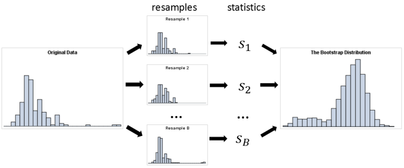
From the sample $\mathcal{D}$, you can *resample with replacement*:

$$
\text{"bootstrap"} \rightarrow
\begin{cases}
\mathcal{D}^*_{(1)} & = \{x^*_1, x^*_2, \dots, x^*_n\} \longrightarrow \theta^*_{(1)} \\
\mathcal{D}^*_{(2)} & = \{x^*_1, x^*_2, \dots, x^*_n\} \longrightarrow \theta^*_{(2)} \\
\dots & \\
\mathcal{D}^*_{(B)} & = \{x^*_1, x^*_2, \dots, x^*_n\} \longrightarrow \theta^*_{(B)}
\end{cases}
$$

Ensemble Learning
Bootstrap

From the sample $\mathcal{D}$, you can *resample with replacement*:

$$\text{"bootstrap"} \rightarrow \begin{cases} \mathcal{D}^*_{(1)} & = \{x^*_1, x^*_2, \dots, x^*_n\} \longrightarrow \theta^*_{(1)} \\ \mathcal{D}^*_{(2)} & = \{x^*_1, x^*_2, \dots, x^*_n\} \longrightarrow \theta^*_{(2)} \\ \dots & \\ \mathcal{D}^*_{(B)} & = \{x^*_1, x^*_2, \dots, x^*_n\} \longrightarrow \theta^*_{(B)} \end{cases}$$



Bootstrap

Let $\widehat{\theta}^* = \mathbb{E}_{\mathcal{P}^*(X)}[\theta^*]$, where $\mathcal{P}^*(X)$ is the *bootstrap distribution*:

$$\widehat{\theta}^* \xrightarrow{P} \widehat{\theta}.$$

Ensemble Learning
Bootstrap

(see also Chen (2019), Washington U.)

Consider the *Monte Carlo* bootstrap estimate

$$\mathrm{Var}_{\mathcal{P}^*(X)}\left(\widehat{\theta}^*\right) = \frac{1}{B-1}\sum_{i=1}^{B}\left(\widehat{\theta}^*_{(i)} - \mathbb{E}[\widehat{\theta}^*]\right) \overset{B \gg 1}{=} \mathrm{Var}_{\mathcal{P}^*(X|\mathcal{D})}\left(\widehat{\theta}^*\right)$$ (“with the sample $\mathcal{D}$ fixed”).

and prove

$$\mathrm{Var}_{\mathcal{P}^*(X|\mathcal{D})}\left(\widehat{\theta}^*\right) \xrightarrow{\mathrm{P}} \mathrm{Var}_{\widehat{\mathcal{P}}(X)}\left(\widehat{\theta}\right)$$

Ensemble Learning
Bootstrap

(see also Chen (2019), Washington U.)

Consider the *Monte Carlo* bootstrap estimate

$$\mathrm{Var}_{\mathcal{P}^*(X)}\left(\widehat{\theta}^*\right) = \frac{1}{B-1}\sum_{i=1}^{B}\left(\widehat{\theta}^*_{(i)} - \mathbb{E}[\widehat{\theta}^*]\right) \overset{B \gg 1}{=} \mathrm{Var}_{\mathcal{P}^*(X|\mathcal{D})}\left(\widehat{\theta}^*\right)$$ ("with the sample $\mathcal{D}$ fixed").

and prove

$$\mathrm{Var}_{\mathcal{P}^*(X|\mathcal{D})}\left(\widehat{\theta}^*\right) \xrightarrow{\mathrm{P}} \mathrm{Var}_{\widehat{\mathcal{P}}(X)}\left(\widehat{\theta}\right)$$

Sketch of the proof:
Let $\Delta^*(\mathcal{D}, B) = \mathrm{Var}_{\mathcal{P}^*(X|\mathcal{D})}\left(\widehat{\theta}^*\right) - \mathrm{Var}_{\widehat{\mathcal{P}}}\left(\widehat{\theta}\right)$, and suppose $\Delta^*(\mathcal{D}, B) < \frac{c}{B}$, with $c > 0$:

- (McDiarmid's inequality) $\mathrm{P}\left(|\Delta^*(\mathcal{D}, B) - \mathbb{E}[\Delta^*(\mathcal{D}, B)]| \geq \varepsilon\right) \leq 2e^{-\frac{2\varepsilon^2\sqrt{B}}{c}}$
- (Borel-Cantelli lemma) $\sum_{i=1}^{B} \mathrm{P}\left(|\Delta^*(\mathcal{D}, B) - \mathbb{E}[\Delta^*(\mathcal{D}, B)]| \geq \varepsilon\right) < \infty \Rightarrow \mathrm{P}\left(\limsup_{B\to\infty} |\Delta^*(\mathcal{D}, B) - \mathbb{E}[\Delta^*(\mathcal{D}, B)]| \geq \varepsilon\right) = 0$
- $\mathbb{E}[\Delta^*(\mathcal{D}, B)] = 0 \Rightarrow \mathrm{Var}_{\mathcal{P}^*(X|\mathcal{D})}\widehat{\theta}^* \xrightarrow{\mathrm{P}} \mathrm{Var}_{\widehat{\mathcal{P}}(X)}\widehat{\theta}$

$\square$

Ensemble Learning

Bootstrap | The bootstrap theorem

With all these elements:

$$\left.\begin{aligned} Z &= \sqrt{n}\frac{\widehat{\theta} - \theta}{\sqrt{\mathrm{Var}_{\mathcal{P}(x)}(\theta)}} \sim \mathcal{N}(0,1) \\ Z^* &= \sqrt{n}\frac{\widehat{\theta}^* - \widehat{\theta}}{\sqrt{\mathrm{Var}_{\widehat{\mathcal{P}}(x)}(\theta)}} \sim \mathcal{N}(0,1) \end{aligned}\right\} \text{estimated parameters have the same distribution!}$$

Ensemble Learning
Bootstrap | The bootstrap theorem

With all these elements:

$$\left.\begin{aligned} Z &= \sqrt{n}\frac{\widehat{\theta} - \theta}{\sqrt{\mathrm{Var}_{\mathcal{P}(\mathrm{X})}(\theta)}} \sim \mathcal{N}(0,1) \\ Z^* &= \sqrt{n}\frac{\widehat{\theta}^* - \widehat{\theta}}{\sqrt{\mathrm{Var}_{\widehat{\mathcal{P}}(\mathrm{X})}(\theta)}} \sim \mathcal{N}(0,1) \end{aligned}\right\} \text{estimated parameters have the same distribution!}$$

Slightly more formally:
Let

$$\mathrm{P}(x) = \mathrm{P}(Z \leq x) \quad \mathrm{P}^*(x) = \mathrm{P}(Z^* \leq x),$$

then (see Berry-Esseen theorem):

$$\left|\mathrm{P}(x) - \mathrm{P}^*(x)\right| \leq |\mathrm{P}(x) - \Phi(x)| + \left|\Phi(x) - \Phi^*(x)\right| + \left|\mathrm{P}^*(x) - \Phi^*(x)\right| \leq C\frac{\mu_3}{\sigma_3\sqrt{n}} + o\left(n^{-\frac{1}{2}}\right) + C^*\frac{\mu_3^*}{\sigma_3^*\sqrt{n}} \to 0$$

when $n \to \infty$. This shows $\mathrm{P} \to \mathrm{P}^*$ in distribution.

$\square$

Ensemble Learning

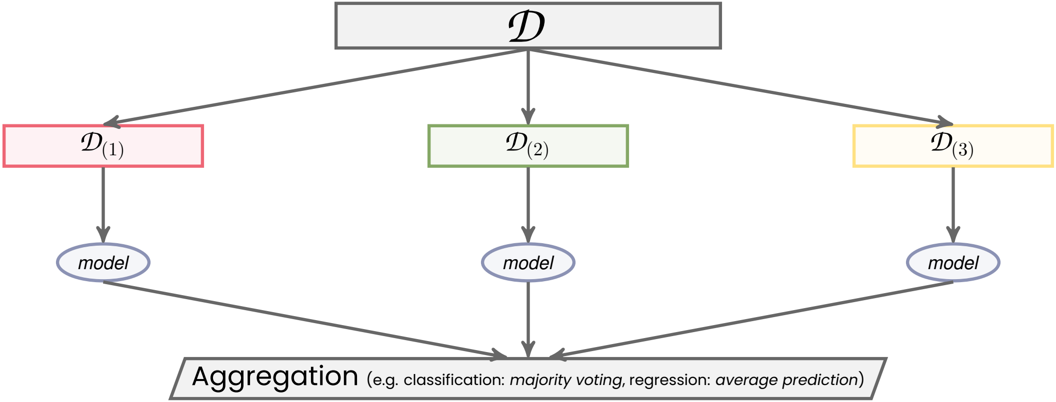
Bootstrap + Aggregating = “Bagging”

Given the previous discussion, it becomes natural to define the **bagging**:

Ensemble Learning

Bootstrap + Aggregating = "Bagging"

Given the previous discussion, it becomes natural to define the **bagging**:



more on this later...

Ensemble Learning
Boosting

Strong Learner

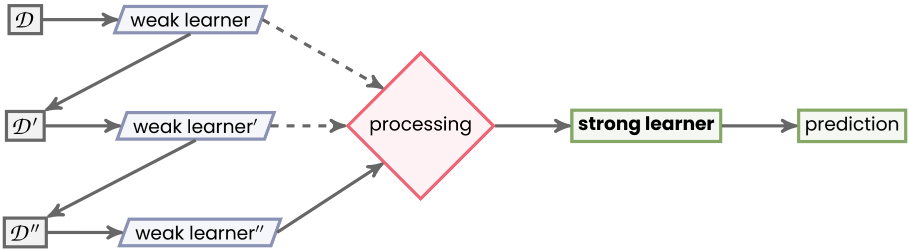
Let $\mathcal{D} = \{(\vec{x}, y)\}$ be a labelled dataset, then a model f is a **strong learner** if

$$\forall \varepsilon > 0 \quad \mathrm{P}(f(\vec{x}) \neq y) \leq \varepsilon.$$

Weak Learner

Let $\mathcal{D} = \{(\vec{x}, y)\}$ be a labelled dataset, then a model f is a **weak learner** if

$$\exists \varepsilon' > 0 \quad \mathrm{P}(f(\vec{x}) \neq y) \leq \varepsilon'.$$

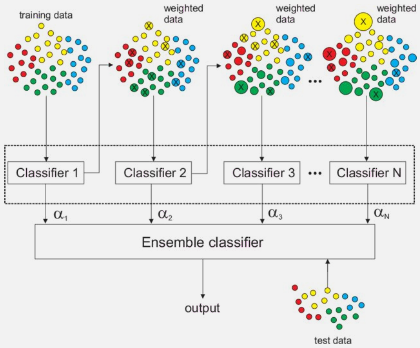


Ensemble Learning

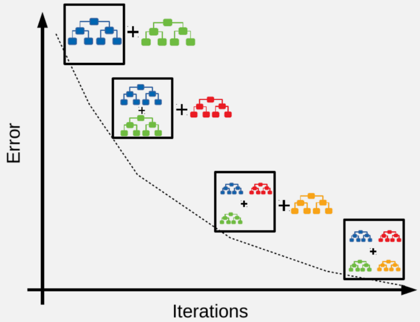
Boosting algorithms

Let $\mathcal{D} = \{(\vec{x}_i, y_i) \mid y_i \in \{-1, 1\}\ \forall i = 1, 2, \dots, N\}$, and $\mathscr{H}$ be a *weak learner* on $\mathcal{D}$:

Ada(ptive)Boost(ing)



Gradient boosting



Ensemble Learning

Boosting algorithms

Let $\mathcal{D} = \{(\vec{x}_i, y_i) \mid y_i \in \{-1, 1\}\ \forall i = 1, 2, \dots, N\}$, and $\mathscr{H}$ be a *weak learner* on $\mathcal{D}$:

Ada(ptive)Boost(ing)

Idea: weighted majority voting

Require: $M > 0$

Require: $m = 0, w_i^{(0)} \leftarrow N^{-1}, \ \forall i = 1, 2, \dots, N$

for $0 < m \leq M$ **do**

run $\mathscr{H}^{(m)}$ on $\mathcal{D}$

$\beta^{(m)} \leftarrow \sum_{i=1}^{N} w_i^{(m)} \, \theta(-y_i \mathscr{H}^{(m)}(\vec{x}_i))$ ▷ *error rate*

$\alpha^{(m)} = \frac{1}{2} \ln \frac{1-\beta^{(m)}}{\beta^{(m)}}$

$w_i^{(m+1)} \leftarrow w_i^{(m)} \operatorname{softmax}_{\alpha^{(m)}}(y_i \mathscr{H}(\vec{x}_i))$

return $\mathscr{H}(\vec{x}) = \operatorname{sign}\left(\sum_{m=1}^{M} \alpha^{(m)} \mathscr{H}^{(m)}(\vec{x})\right)$.

Gradient boosting

Idea: improving on previous attempts

Require: $M > 0, m = 0, \nu > 0, \mathscr{H}(\vec{x}) = \gamma + \sum_{i=1}^{M} \gamma^{(m)} \mathrm{h}^{(m)}(\vec{x}), \mathrm{h}^{(m)}$ *weak learner*, $\mathscr{H}^{(0)}(\vec{x}) = \arg\min_{\gamma} \mathcal{L}(y, \gamma)$

Require: $\mathcal{H}^{(m)} = \mathcal{H}^{(m-1)} + \arg\min_{\gamma^{(m)}, \mathrm{h}^{(m)}} \mathcal{L}(y, \mathcal{H}^{(m-1)}(\vec{x}) + \gamma^{(m)} \mathrm{h}^{(m)}(\vec{x}))$

for $0 < m \leq M$ **do**

$r_i^{(m)} = -\left.\frac{\delta \mathcal{L}}{\delta \mathscr{H}}\right|_{\mathscr{H} = \mathscr{H}^{(m)}}$ for $i = 1, 2, \dots, N$

train $\mathcal{H}^{(m)}$ on $\{(\vec{x}, r^{(m)})\}$

$\gamma^{(m)} \leftarrow \arg\min_{y} \mathcal{L}\left(y, \mathscr{H}^{(m-1)}(\vec{x}) + \nu\, \gamma \mathrm{h}^{(m)}(\vec{x})\right)$

$\mathscr{H}^{(m)} \leftarrow \mathscr{H}^{(m-1)} + \nu\, \gamma^{(m)} \mathrm{h}^{(m)}$ ▷ ν *learning rate*

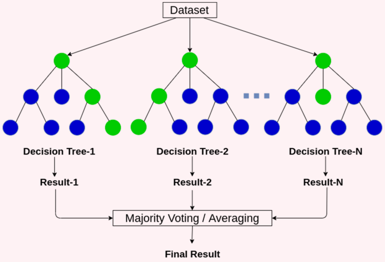
return $\mathscr{H}^{(M)}(\vec{x})$.

Remeber that θ is the Heaviside function: $\theta(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ 0 & \text{if } x < 0 \end{cases}$, and $\operatorname{softmax}_{\beta}(z) = \frac{\exp(-\beta z)}{\sum_{i=1}^{N} \exp(-\beta z_i)}$.

Ensemble Learning

Random forests and boosted decision trees

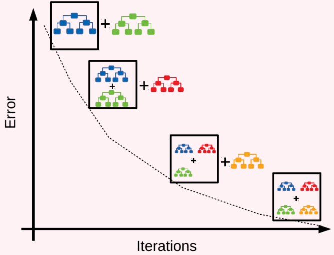
??? | Random Forest



Remember that for $Y = (y_1, \dots, y_B)$ i.i.d. (variance σ^2 and pairwise correlation ρ):

$$\rho = \frac{\mathrm{Cov}(Y)}{\sigma^2} \Leftrightarrow \mathrm{Cov}\left(\overline{Y}\right) = \rho\sigma^2 + \frac{1-\rho}{B}\sigma^2$$

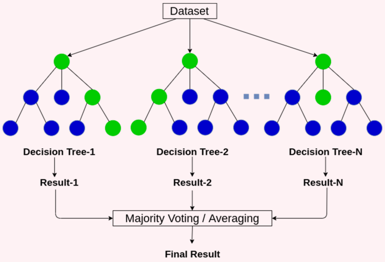
??? | Boosted Decision Trees



Ensemble Learning

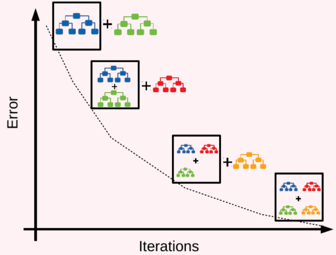
Random forests and boosted decision trees

??? | Random Forest



Variance → ?
Bias → ?

??? | Boosted Decision Trees

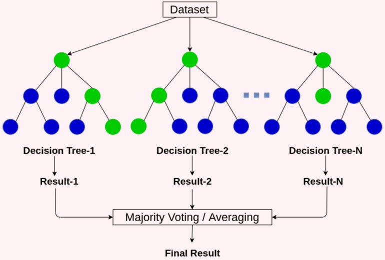


Variance → ?
Bias → ?

Ensemble Learning

Random forests and boosted decision trees

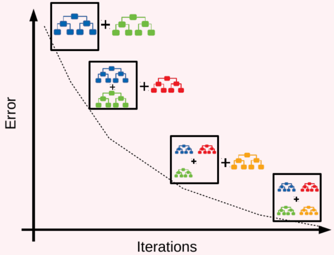
Bagging | Random Forest



Variance $\to$ reduction (Cov($\overline{Y}$) $\overset{B \gg 1}{\to}$ $\rho\sigma^2 \leq \sigma^2$)
Bias $\to$ increase (more restrictions)

Trees in **random forests** are usually **fully-grown** to start with a low bias, and to reduce bias after bagging.

Boosting | Boosted Decision Trees



Variance $\to$ increase
Bias $\to$ decrease

Trees in **gradient boosting** are usually **shallow** to start with high bias, and decrease it after boosting.

4. Neural Networks

Computational graphs

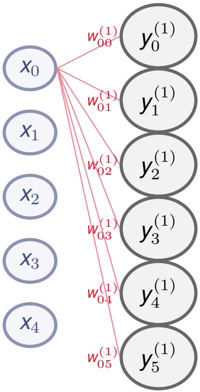
Table of contents

Computational Graphs
Preliminaries

Let $f^{(n)}$ be one of N *affine* functions:

$$f^{(n)} : \mathbb{R}^{w_{(n-1)}} \to \mathbb{R}^{w_{(n)}}, \qquad n = 1, 2, \dots, N$$

Computational Graphs
Preliminaries



Let $f^{(n)}$ be one of N *affine* functions:

$$f^{(n)} : \mathbb{R}^{w_{(n-1)}} \to \mathbb{R}^{w_{(n)}}, \qquad n = 1, 2, \dots, N$$

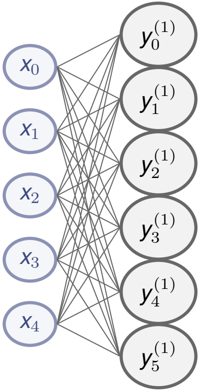
s.t.

$$\vec{y}^{(n)} = f^{(n)}\left(y^{(n-1)}\right) = W^{(n)}\vec{y}^{(n-1)} + \vec{b}^{(n)},$$

and $\vec{y}^{(0)} = \vec{x}$ (W: “weights”, b: “bias”):

$$y^{(N)} = f^{(N)} \circ f^{(N-1)} \circ \dots \circ f^{(1)}(\vec{x})$$

Computational Graphs
Preliminaries



Let $f^{(n)}$ be one of N *affine* functions:

$$f^{(n)} : \mathbb{R}^{w_{(n-1)}} \to \mathbb{R}^{w_{(n)}}, \quad n = 1, 2, \dots, N$$

s.t.

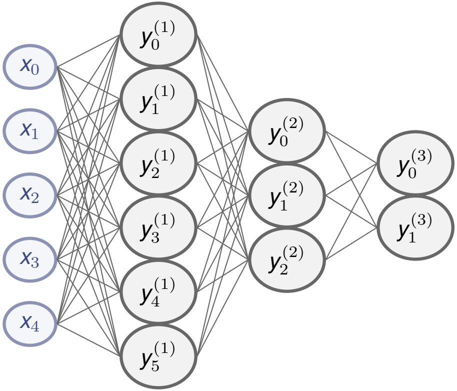
$$\vec{y}^{(n)} = f^{(n)}\left(y^{(n-1)}\right) = W^{(n)} \vec{y}^{(n-1)} + \vec{b}^{(n)},$$

and $\vec{y}^{(0)} = \vec{x}$ (W: "weights", b: "bias"):

$$y^{(N)} = f^{(N)} \circ f^{(N-1)} \circ \cdots \circ f^{(1)}(\vec{x})$$

Computational Graphs

Preliminaries



Let $f^{(n)}$ be one of N *affine* functions:

$$f^{(n)} : \mathbb{R}^{w_{(n-1)}} \to \mathbb{R}^{w_{(n)}}, \qquad n = 1, 2, \dots, N$$

s.t.

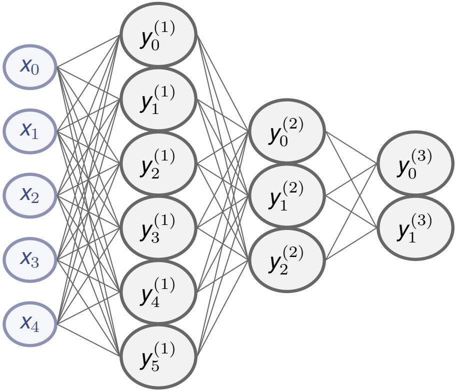
$$\vec{y}^{(n)} = f^{(n)}\left(y^{(n-1)}\right) = W^{(n)} \vec{y}^{(n-1)} + \vec{b}^{(n)},$$

and $\vec{y}^{(0)} = \vec{x}$ (W: "weights", b: "bias"):

$$y^{(N)} = f^{(N)} \circ f^{(N-1)} \circ \cdots \circ f^{(1)}(\vec{x})$$

Computational Graphs

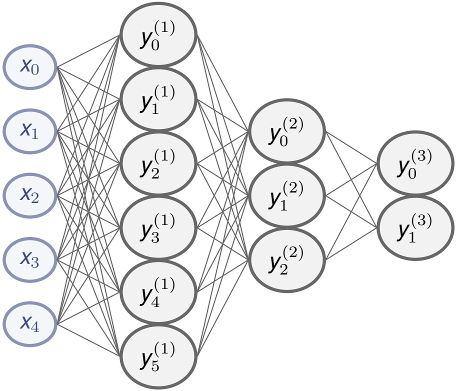
Linearity vs non linearity



Linearity of the network

$$\vec{y}^{(N)} = \underbrace{W^{(N)} W^{(N-1)} \dots W^{(1)}}_{W} \vec{x} + \underbrace{\vec{b}^{(N)} + W^{(N)} \vec{b}^{(N-1)} + \dots}_{\vec{b}}$$

Computational Graphs
Linearity vs non linearity



Linearity of the network

$$\vec{y}^{(N)} = \underbrace{W^{(N)} W^{(N-1)} \dots W^{(1)}}_{W} \vec{x} + \underbrace{\vec{b}^{(N)} + W^{(N)} \vec{b}^{(N-1)} + \dots}_{\vec{b}}$$

Activation functions

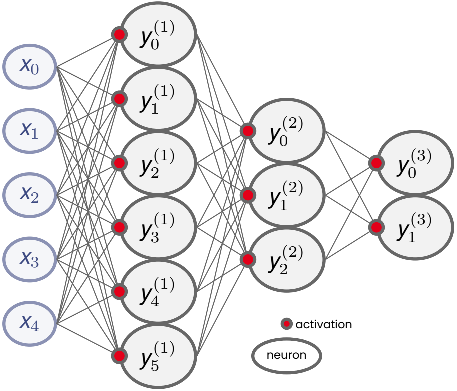
Let $a^{(n)} : \mathbb{R} \times \mathbb{R}$ **non linear**:

$$\left(g^{(n)}(\vec{x}) \right)_{ij} = a^{(n)} \left(\left(f^{(n)}(\vec{x}) \right)_{ij} \right),$$

where $i = 1, 2, \dots, w_{(n)}$ and $j = 1, 2, \dots, w_{(n-1)}$.

Computational Graphs

Neural networks



The activation on the last layer strongly depends on the task...More on this later!

Call ($\odot$ is the Hadamard product)

$$g^{(n)}(\vec{y}^{(n-1)}) = a^{(n)} \odot \left(W^{(n)} \vec{y}^{(n-1)} + \vec{b}^{(n)} \right)$$

the n-th **layer** in the graph.

Neural network

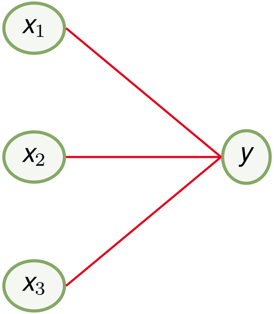
The non linear function:

$$g^{(N)} \circ g^{(N-1)} \circ \cdots \circ g^{(1)}$$

is called a (*fully connected*) **neural network** (NN) with $N - 1$ **hidden layers**.

The (Multi-Layered) Perceptron
The historical context

The structure



is called **perceptron** Rosenblatt (1958) and it represents the fundamental unit of a NN.
A *stack* of perceptrons is called **Multi-Layered Perceptron** (MLP.)

4. Neural Networks

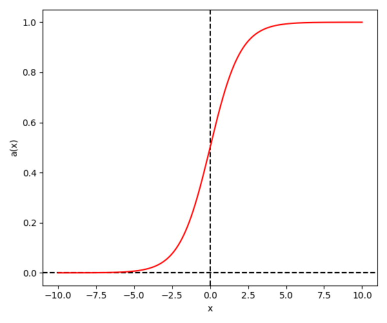
Non Linearity of Neural Networks

Table of contents

Neural Networks
Activation functions

Activation functions might depend on the task, to ease the training. For instance:

Sigmoid



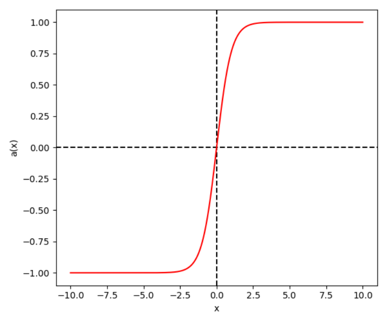
$$a(x) = \sigma(x) = \frac{1}{1 + e^{-x}}$$

- classically the first...
- gradients might saturate for $x \to \pm\infty$ see later
- good interpetation as GLM (probability)

Neural Networks
Activation functions

Activation functions might depend on the task, to ease the training. For instance:

Hyperbolic Tangent



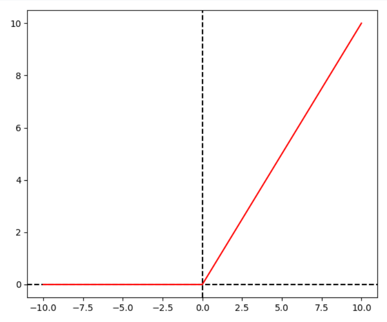
$$a(x) = \tanh(x) = \frac{e^{2x} - 1}{e^{2x} + 1}$$

- outputs naturally centred
- might saturate for $x \to \pm\infty$
- good alternative to σ
- traditionally in (old) GANs

Neural Networks
Activation functions

Activation functions might depend on the task, to ease the training. For instance:

REctified Linear Unit



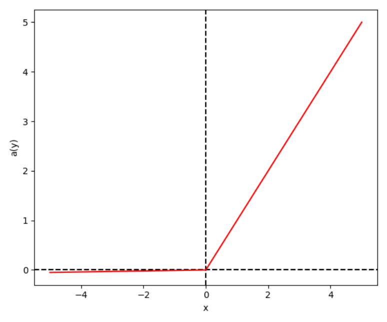
$$a(x) = \mathrm{ReLU}(x) = \max(0, x) = \begin{cases} x, & \text{if } x \geq 0 \\ 0, & \text{if } x < 0 \end{cases}$$

- **omnipresent** powerful sparsifier see Glorot et al. (2011)
- gradients might saturate for $x \to -\infty$
- forces positive outputs (!) **Q**: is it good for output layer?
- slightly non differentiable
- computationally fast

Neural Networks
Activation functions

Activation functions might depend on the task, to ease the training. For instance:

Leaky REctified Linear Unit

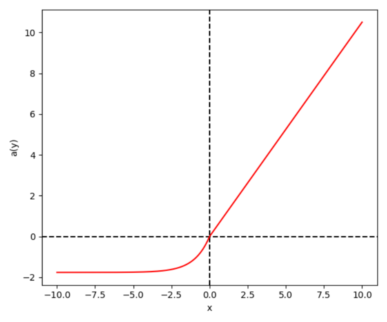


$$a(x) = \text{LeakyReLU}_{\alpha}(x) = \begin{cases} x, & \text{if } x \geq 0 \\ \alpha\, x, & \text{if } x < 0 \end{cases}$$

- new *slope* hyperparameter $\alpha \in \mathbb{R}^+$
- solves the saturation problem
- negative outputs are slightly allowed
- slightly non differentiable
- computationally fast

Activation functions might depend on the task, to ease the training. For instance:

Scaled Exponential Linear Unit



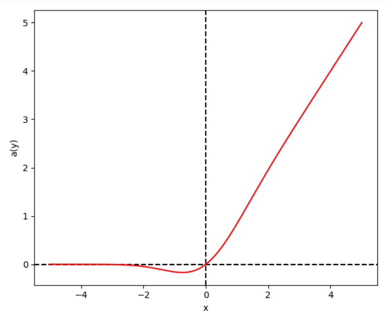
$$
\begin{aligned}
a(x) &= \mathrm{SELU}(x) \\
&= \gamma\left(\max(0, x) + \min(0, \alpha(e^x - 1))\right)
\end{aligned}
$$

- improve NN beahviour see Klambauer et al. (2017)
- solves the saturation problem
- negative outputs are not sparsified
- slightly non differentiable
- requires good initialisation see later...

Neural Networks
Activation functions

Activation functions might depend on the task, to ease the training. For instance:

Gaussian Error Linear Unit



$$a(x) = \mathrm{GELU}(x) = x\,\Phi(x)$$

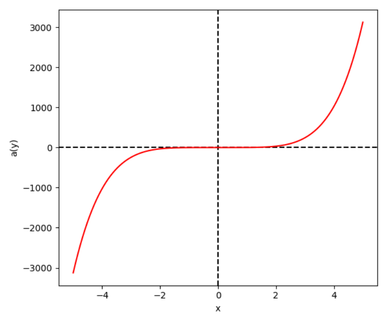
- stochastic regularisation method see Hendrycks and Gipel (2016)
- might saturate at $x \to -\infty$, but discouraged
- good normalisation of the activations

Neural Networks

Activation functions

Activation functions might depend on the task, to ease the training. For instance:

Homogeneous activation



$$a(x) = x^p, \mid \; a(\lambda x) = \lambda^p a(x)$$

- good behaviour of the network
- might help approximations
- useful in scientific/"physics informed" scenarios
- use at your own risk...

"We are all responsible users" (from the Python guide)

Neural Networks

Activation functions

Activation functions might depend on the task, to ease the training. For instance:

Last layer activations

Binary classification

$$a^{(N)}(\vec{x})_i = \sigma(x_i) = \frac{1}{1 + e^{-x_i}} \in [0, 1]$$

Multiclass classification

$$a^{(N)}(\vec{x})_i = \mathrm{softmax}(x_i) = \frac{e^{x_i}}{\sum_{i=1}^{K} e^{x_i}} \in [0, 1]^K$$

Regression

$$a^{(N)}(\vec{x})_i = \mathrm{Id}(x_i)$$

Activations are quite flexible and strongly depend on the type of task required (e.g.: if all outputs are positives, ReLU might be used for a regression task)!

4. Neural Networks

Approximation theorems

Table of contents

Neural Networks

Universal approximation theorems

Theorem (Cybenko | Approximation by sigmoid-activated NNs)

Let $\mathcal{F}$ be the set of $\mathscr{C}^1([0,1]^n)$ scalar functions, and σ be a ***sigmoid*** *function. Then*

$$\exists N > 0 \;\mid\; f(\vec{x}) = \sum_{i=1}^{N} \alpha_i \, \sigma\left(\vec{w}_i \cdot \vec{x} + b\right)$$

is dense in $\mathcal{F}$.

In simple words: using a 1-layer deep sigmoid-activated scalar NN we can approximate with arbitrary precision any $\mathscr{C}([0,1])$ scalar function. Let $g(\vec{x}) \in \mathscr{C}([0,1])$ and $f(\vec{x})$ be such NN, then

$$\forall \varepsilon > 0, \quad \sup_{\vec{x}} ||f(\vec{x}) - g(\vec{x})|| < \varepsilon.$$

Neural Networks

Universal approximation theorems

Theorem (Kolmogorov-Arnold | Approximation theorem)

Let f be a function in $\mathcal{F}$, then

$$f(\vec{x}) = \sum_{q=0}^{2n} \phi_q \left(\sum_{p=1}^{n} \varphi_{q,p}\left(x_p\right), \right)$$

where ϕ and φ are continuous scalar functions of a single variable.

In simple words: the only "needed" functions are single-variable activations and sums. Technically, if we could choose the activations of each unit (neuron), we could exactly write any multivariate function as superposition of univariate functions.

might be interesting to some of you: Liu et al. (2024)

Neural Networks

Universal approximation theorems

Theorem (Width expressivity of NNs see Lu et al. (2017)**)**

Let $g \colon \mathbb{R}^n \to \mathbb{R}$ be a Lebesgue integrable function, and $\mathfrak{F}$ be the set of ***fully connected ReLU-activated NN*** *with width $w \leq n + 4$:*

$$\exists f \in \mathfrak{F} \mid \forall \varepsilon > 0 \int_{\mathbb{R}^n} |g(\vec{x}) - f(\vec{x})| < \varepsilon$$

In other words: width-bounded NNs can be used as universal approximators on the entire domain of definition. It is also curious to see:

$$w \leq n \Rightarrow \int_{\mathbb{R}^n} |g(\vec{x}) - f(\vec{x})| \text{ diverges,}$$

hence the restrictions to $[-1, 1]^n$ (i.e. good normalisation):

$$w \leq n - 1 \Rightarrow \exists \varepsilon' > 0 \mid \int_{[-1,1]^n} |g(\vec{x}) - f(\vec{x})| \geq \varepsilon'.$$

Neural Networks

Universal approximation theorems

Theorem (Trade-off width/depth see Lu et al. (2017)**)**

Let n be the input dimensions. For any integer $k \geq n + 4$, there exists a ReLU-activated NN $f \colon \mathbb{R}^n \to \mathbb{R}$ with width $w = 2k^2$ and depth $d = 3$, such that

$$\forall b > 0, \ \forall g \colon \mathbb{R}^n \to \mathbb{R},$$

where g is a ReLU-activated NN whose parameters are bounded in $[-b, b]$, with width $w' \leq k^{\frac{3}{2}}$ and depth $d \leq k + 2$, it is true that

$$\exists \varepsilon > 0 \ \mid \ \int_{\mathbb{R}^n} (f(\vec{x}) - g(\vec{x}))^2 \geq \varepsilon.$$

In other words: any decrease in width, should be compensated by an increase in depth to keep the same expressivity of the NNs.

Neural Networks

Universal approximation theorems

Some needed questions

Q1: fully connected NNs are universal approximators! Did we answer *the ultimate question of life, the universe and everything?*

Neural Networks

Universal approximation theorems

Some needed questions

Q1: fully connected NNs are universal approximators! Did we answer *the ultimate question of life, the universe and everything?*

- the answer is “42”, not “neural networks”... 🤭
- **existence** of sth $\not\Rightarrow$ easy to find
- fully-connected NNs **low bias** $\not\Rightarrow$ not all kinds of inputs are adapted

Neural Networks

Universal approximation theorems

Some needed questions

Q1: fully connected NNs are universal approximators! Did we answer *the ultimate question of life, the universe and everything?*

- the answer is “42”, not “neural networks”... 🤭
- **existence** of sth $\nRightarrow$ easy to find
- fully-connected NNs **low bias** $\nRightarrow$ not all kinds of inputs are adapted

Q2: 🤬 shut up, I don’t care! Suppose we found a perfect NN: can we deploy it for the world to see and use?

Neural Networks

Universal approximation theorems

Some needed questions

Q1: fully connected NNs are universal approximators! Did we answer *the ultimate question of life, the universe and everything?*

- the answer is "42", not "neural networks"... 🤭
- **existence** of sth $\nRightarrow$ easy to find
- fully-connected NNs **low bias** $\nRightarrow$ not all kinds of inputs are adapted

Q2: 🤬 shut up, I don't care! Suppose we found a perfect NN: can we deploy it for the world to see and use?

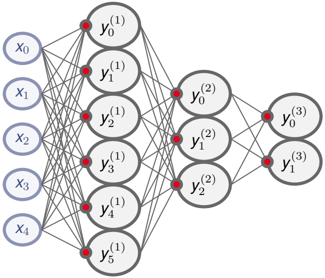
- you would lead us to another AI winter...
- NNs are trained on **samples** $\Rightarrow$ predict conditioned on that (+ some extrapolation)
- probably ok with **infinite** amount of data (**population**), but how to train?

4. Neural Networks

Neural network training

Table of contents

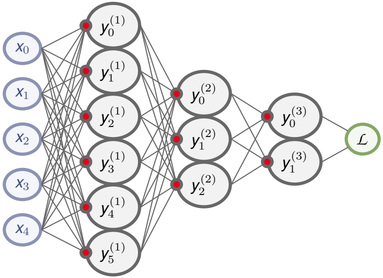
Neural Network Training
Backpropagation



see Rumelhart et al. (1986)

Neural Network Training

Backpropagation



see Rumelhart et al. (1986)

Let $\mathcal{L} : \mathbb{R}^{w^{(N)}} \to \mathbb{R}$ be a **loss function** (it depends on the task) and append it to the NN, where at the ℓ-th layer:

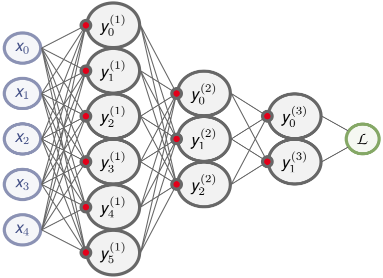
$$y_i^{(\ell+1)} = a^{(\ell)}\left(z_i^{(\ell)}\right),$$

$$z_i^{(\ell)} = \sum_{j=1}^{w^{(\ell-1)}} W_{ij}^{(\ell)} y_j^{(\ell-1)} + b_i^{(\ell)}$$

Neural Network Training

Backpropagation

see Rumelhart et al. (1986)



Let $\mathcal{L} : \mathbb{R}^{w(N)} \to \mathbb{R}$ be a **loss function** (it depends on the task) and append it to the NN, where at the ℓ-th layer:

$$y_i^{(\ell+1)} = a^{(\ell)}(z_i^{(\ell)}),$$

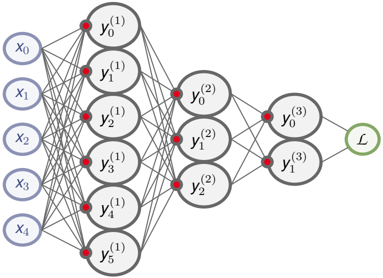
$$z_i^{(\ell)} = \sum_{j=1}^{w^{(\ell-1)}} W_{ij}^{(\ell)} y_j^{(\ell-1)} + b_i^{(\ell)}$$

We can perform **gradient descent** (GD) for each $W^{(\ell)}$, $\ell = 1, 2, \dots, N$ by computing:

$$\begin{cases} \dfrac{\partial \mathcal{L}}{\partial W_{ij}^{(\ell)}} = \displaystyle\sum_{k=1}^{w^{(\ell)}} \dfrac{\partial \mathcal{L}}{\partial z_k^{(\ell)}} \dfrac{\partial z_k^{(\ell)}}{\partial W_{ij}^{(\ell)}} = \dfrac{\partial \mathcal{L}}{\partial z_i^{(\ell)}} y_j^{(\ell-1)} \\[2ex] \dfrac{\partial \mathcal{L}}{\partial b_i^{(\ell)}} = \displaystyle\sum_{k=1}^{w^{(\ell)}} \dfrac{\partial \mathcal{L}}{\partial z_k^{(\ell)}} \dfrac{\partial z_k^{(\ell)}}{\partial b_i^{(\ell)}} = \dfrac{\partial \mathcal{L}}{\partial z_i^{(\ell)}} \end{cases}$$

Neural Network Training
Backpropagation

see Rumelhart et al. (1986)

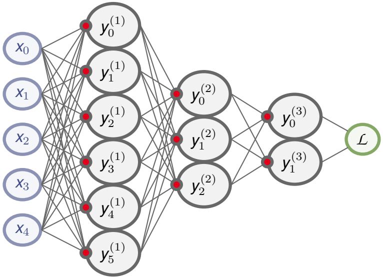


From the previous expression:

$$\delta_i^{(\ell-1)} = \frac{\partial \mathcal{L}}{\partial z_i^{(\ell-1)}}$$

Neural Network Training
Backpropagation

see Rumelhart et al. (1986)

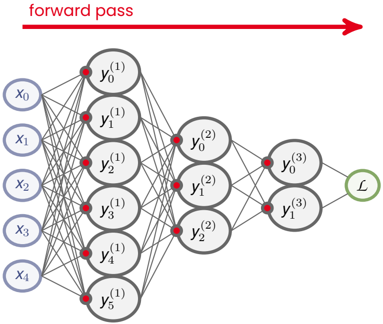


From the previous expression:

$$
\begin{aligned}
\delta_i^{(\ell-1)} &= \frac{\partial \mathcal{L}}{\partial z_i^{(\ell-1)}} \\
&= \sum_{k=1}^{w^{(\ell)}} \frac{\partial \mathcal{L}}{\partial z_k^{(\ell)}} \frac{\partial z_k^{(\ell)}}{\partial z_i^{(\ell-1)}} \\
&= \sum_{k=1}^{w^{(\ell)}} \sum_{h=1}^{w^{(\ell-1)}} \delta_k^{(\ell)} \frac{\partial z_k^{(\ell)}}{\partial a_h^{(\ell-1)}} \frac{\partial a_h^{(\ell-1)}}{\partial z_i^{(\ell-1)}} \\
&= \left(a^{(\ell-1)}\right)' \sum_{k=1}^{w^{(\ell)}} \delta_k^{(\ell)} W_{ki}^{(\ell)}
\end{aligned}
$$

Neural Network Training
Backpropagation

see Rumelhart et al. (1986)

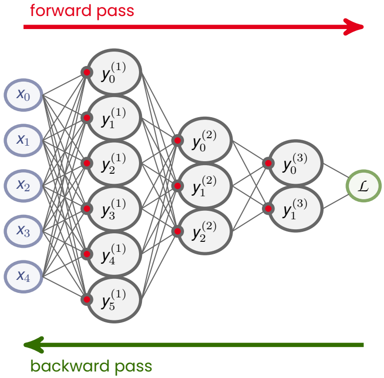


Training a NN is a **two steps** procedure:

1. during the forward pass the outputs and outputs of each layer (loss included) are computed and **stored**

Neural Network Training
Backpropagation

see Rumelhart et al. (1986)



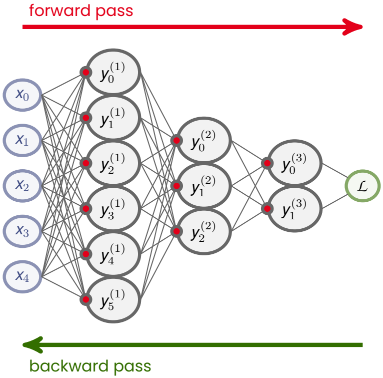
Training a NN is a **two steps** procedure:

1. during the forward pass the outputs and outputs of each layer (loss included) are computed and **stored**
2. in the backward pass the gradients of each layer are assembled **iteratively**

Neural Network Training

Backpropagation

see Rumelhart et al. (1986)



Training a NN is a **two steps** procedure:

1. during the forward pass the outputs and outputs of each layer (loss included) are computed and **stored**
2. in the backward pass the gradients of each layer are assembled **iteratively**

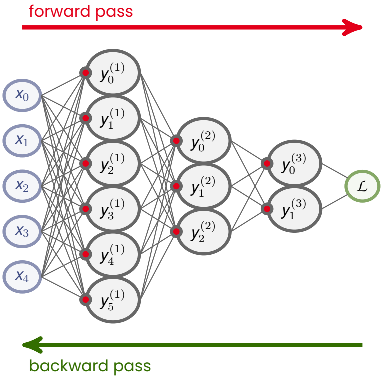
Finally, the **update** of the parameters is:

$$\begin{cases} W_{ij}^{(\ell)} & \leftarrow W_{ij}^{(\ell)} - \alpha \frac{\partial \mathcal{L}}{\partial W_{ij}^{(\ell)}} \\ b_i^{(\ell)} & \leftarrow b_i^{(\ell)} - \alpha \frac{\partial \mathcal{L}}{\partial b_i^{(\ell)}} \end{cases},$$

where α is the *learning rate* **hyperparameter**.

Neural Network Training

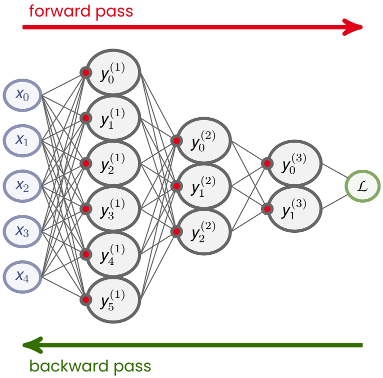
Backpropagation



Q: what are good **initialisations** of weights and biases?

- NNs **propagate** by matrix multiplication
- gradients **large** to update
- gradients **small** not to explode

Neural Network Training
Backpropagation



Q: what are good **initialisations** of weights and biases?

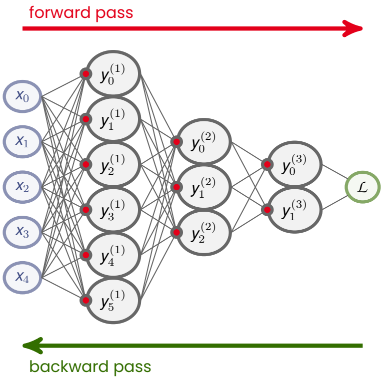
- NNs **propagate** by matrix multiplication
- gradients **large** to update
- gradients **small** not to explode

However, we know:

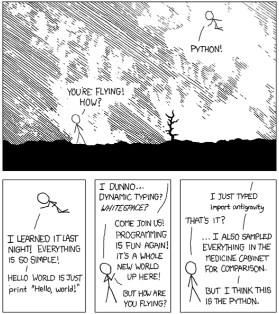
$$\begin{cases} \dfrac{\partial \mathcal{L}}{\partial W_{ij}^{(\ell)}} & = \delta_i^{(\ell)} \, y_j^{(\ell-1)} \\ \dfrac{\partial \mathcal{L}}{\partial b_i^{(\ell)}} & = \delta_i^{(\ell)} \end{cases}$$

Neural Network Training

Backpropagation



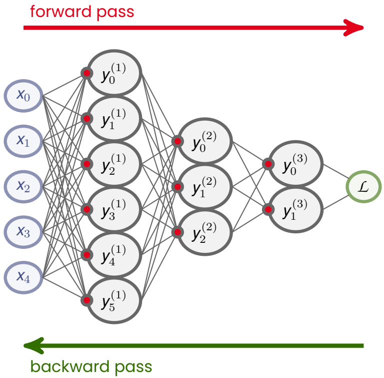
Backpropagation: usually *boilerplate code* which is already available in most frameworks (Pytorch, Lightning, Tensorflow, Keras, etc.)



(xkcd.com)

Neural Network Training

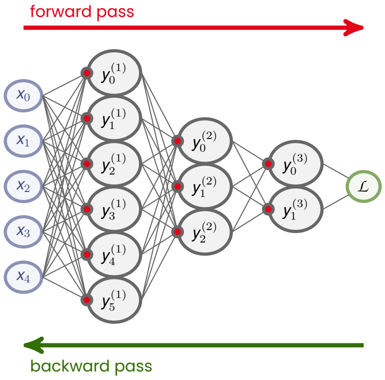
Backpropagation



THINK

What would happen if all $W_{ij}^{(\ell)}$ were to be initialised to the same constant (say 0) $\forall \ell = 1, 2, \dots, N$?

Neural Network Training
Backpropagation



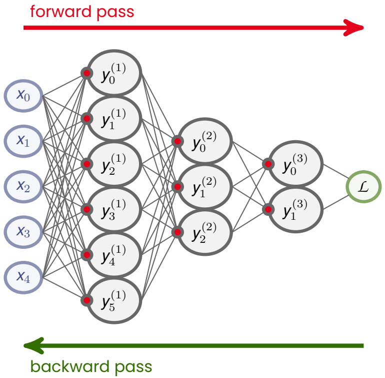
THINK

What would happen if all $W_{ij}^{(\ell)}$ were to be initialised to the same constant (say 0) $\forall \ell = 1, 2, \dots, N$?

All activations $y^{(\ell)}$ would be the **same**!

Neural Network Training

Backpropagation



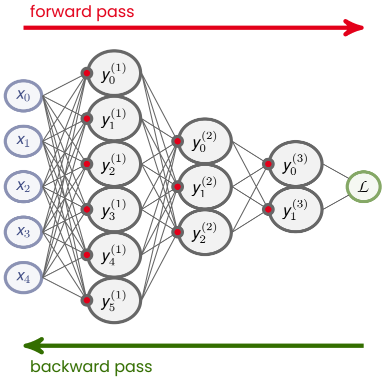
THINK

What would happen if all $W_{ij}^{(\ell)}$ were to be initialised to the same constant (say 0) $\forall \ell = 1, 2, \dots, N$?

All activations $y^{(\ell)}$ would be the **same**!

What about $\delta^{(\ell)}$?

Neural Network Training
Backpropagation



THINK

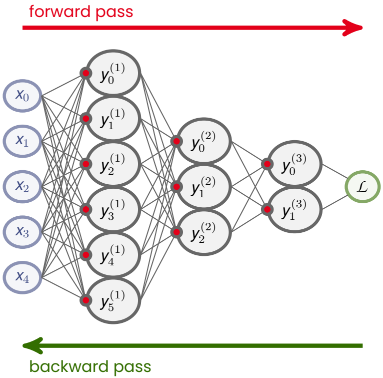
What would happen if all $W_{ij}^{(\ell)}$ were to be initialised to the same constant (say 0) $\forall \ell = 1, 2, \dots, N$?

All activations $y^{(\ell)}$ would be the **same**!

What about $\delta^{(\ell)}$?

All updates $\delta^{(\ell)}$ would be the **same**!

Neural Network Training
Backpropagation



THINK

What would happen if all $W_{ij}^{(\ell)}$ were to be initialised to the same constant (say 0) $\forall \ell = 1, 2, \dots, N$?

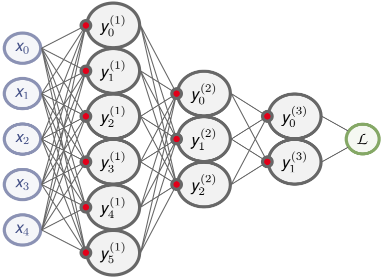
All activations $y^{(\ell)}$ would be the **same**!

What about $\delta^{(\ell)}$?

All updates $\delta^{(\ell)}$ would be the **same**!

Nothing to learn!

Neural Network Training
Weight initialisation



Initialisation

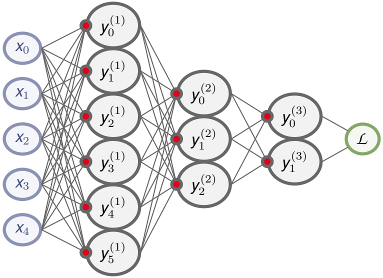
It is **fundamental** to **break the symmetry** (at least for $W^{(\ell)}$):

- initialise with **random** values $\mathcal{N}(\mu, \sigma^2)$
- avoid **large** entries
- follow good **rules of thumb**:

$$\mathbb{E}\left[y^{(\ell)}\right] = \mathbb{E}\left[y^{(\ell-1)}\right] = 0$$

$$\mathrm{Var}\left(y^{(\ell)}\right) = \mathrm{Var}\left(y^{(\ell-1)}\right)$$

Neural Network Training
Weight initialisation



Some examples

- *LeCun* initialisation (LeCun et al. (1998))

$$W_{ij}^{(\ell)} \sim \mathcal{N}\left(0, \left(w^{(\ell-1)}\right)^{-1}\right), \quad b_i^{(\ell)} = 0$$

for normally centred activations

- *Xavier/Glorot* initialisation (Glorot and Bengio (2010))

$$W_{ij}^{(\ell)} \sim \mathcal{N}\left(0, 2\left(w^{(\ell)} + w^{(\ell-1)}\right)^{-1}\right) \quad b_i^{(\ell)} = 0$$

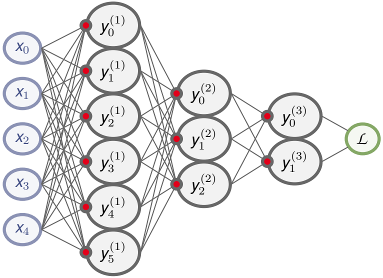
for sigmoid/tanh activations

- *(Kaiming) He* initialisation (He et al. (2015))

$$W_{ij}^{(\ell)} \sim \mathcal{N}\left(0, 2\left(w^{(\ell-1)}\right)^{-1}\right) \quad b_i^{(\ell)} = 0$$

for ReLU-family activations

Neural Network Training
Weight initialisation



Some examples

- *LeCun* initialisation (LeCun et al. (1998))

$$W_{ij}^{(\ell)} \sim \mathcal{N}\left(0, \left(w^{(\ell-1)}\right)^{-1}\right), \quad b_i^{(\ell)} = 0$$

- *Xavier/Glorot* initialisation (Glorot and Bengio (2010))

$$W_{ij}^{(\ell)} \sim \mathcal{N}\left(0, 2\left(w^{(\ell)} + w^{(\ell-1)}\right)^{-1}\right) \quad b_i^{(\ell)} = 0$$

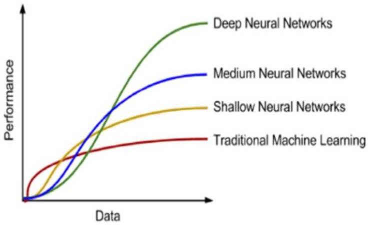
- *(Kaiming) He* initialisation (He et al. (2015))

$$W_{ij}^{(\ell)} \sim \mathcal{N}\left(0, 2\left(w^{(\ell-1)}\right)^{-1}\right) \quad b_i^{(\ell)} = 0$$

HOMEWORK

- Derive the formula of LeCun initialisation – or look it up, it is still cool!
- Derive the formula of Kaiming He initialisation (what is the difference?)

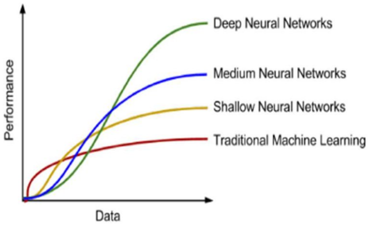
Neural Network Training
Mini-batch gradient descent



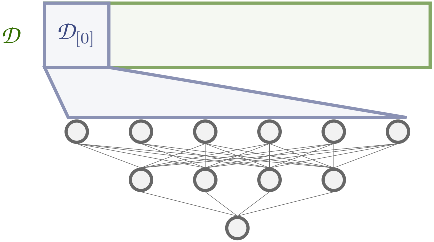
- NNs are powerful at learning/digesting huge amounts of data
- PCs might not be able to load everything all at once
- how to process lots of data?

Neural Network Training

Mini-batch gradient descent



- NNs are powerful at learning/digesting huge amounts of data
- PCs might not be able to load everything all at once
- how to process lots of data?



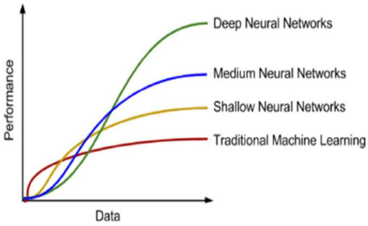
Require: dataset $\mathcal{D} = \{(\vec{x}, y)\}$

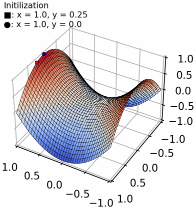
Require: $\{\mathcal{D}_{[b]}\}_{b\in[1,\mathcal{B}]}$ s.t. $\bigcup_{i=0}^{\mathcal{B}-1} \mathcal{D}_{[b]} = \mathcal{D}$

for $0 \leq b < \mathcal{B}$ **do**
  compute forward pass on $\mathcal{D}_{[b]}$
  perform *backpropagation*
  update $W^{(\ell)}$ and $b^{(\ell)}$
return trained NN

Neural Network Training

Mini-batch gradient descent





animation by Luis Medina

- NNs are powerful at learning/digesting huge amounts of data
- PCs might not be able to load everything all at once
- how to process lots of data?

Require: dataset $\mathcal{D} = \{(\vec{x}, y)\}$

Require: $\{\mathcal{D}_{[b]}\}_{b \in [1, \mathcal{B}]}$ s.t. $\bigcup_{i=0}^{\mathcal{B}-1} \mathcal{D}_{[b]} = \mathcal{D}$

for $0 \leq b < \mathcal{B}$ **do**

compute forward pass on $\mathcal{D}_{[b]}$

perform *backpropagation*

update $W^{(\ell)}$ and $b^{(\ell)}$

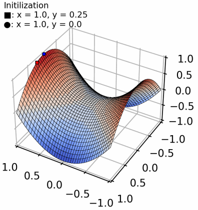
return trained NN

Neural Network Training

Mini-batch gradient descent

Define:

- **iteration**: one pass of mini-batch GD
- **epoch**: one pass over the dataset



animation by Luis Medina

Optimisation

Q: how to choose the size of the mini-batch?

Q: what happens if $\mathcal{B} = |\mathcal{D}|$?

Require: dataset $\mathcal{D} = \{(\vec{x}, y)\}$

Require: $\{\mathcal{D}_{[b]}\}_{b \in [1,\mathcal{B}]}$ s.t. $\bigcup_{i=0}^{\mathcal{B}-1} \mathcal{D}_{[b]} = \mathcal{D}$

for $0 \leq b < \mathcal{B}$ **do**

  compute forward pass on $\mathcal{D}_{[b]}$
  perform *backpropagation*
  update $W^{(\ell)}$ and $b^{(\ell)}$

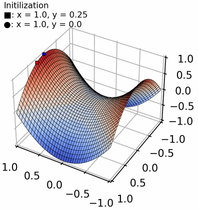
return trained NN

Neural Network Training

Mini-batch gradient descent

Define:

- **iteration**: one pass of mini-batch GD
- **epoch**: one pass over the dataset



animation by Luis Medina

Optimisation

Q: how to choose the size of the mini-batch?

Even though it has a **regularisation** effect, I would not consider it as hyperparameter: it mostly depends on memory constraints.

Q: what happens if $\mathcal{B} = |\mathcal{D}|$?

This is called “stochastic” GD. Useful for huge datasets.

Require: dataset $\mathcal{D} = \{(\vec{x}, y)\}$

Require: $\{\mathcal{D}_{[b]}\}_{b\in[1,\mathcal{B}]}$ s.t. $\bigcup\limits_{i=0}^{\mathcal{B}-1} \mathcal{D}_{[b]} = \mathcal{D}$

for $0 \leq b < \mathcal{B}$ **do**
  compute forward pass on $\mathcal{D}_{[b]}$
  perform *backpropagation*
  update $W^{(\ell)}$ and $b^{(\ell)}$
return trained NN

Neural Network Training

Advantages of mini-batch gradient descent

Honest question

Why should we use **mini-batch** gradient descent? What if my entire dataset fits into memory?

Smith et al. (2021)

Consider the gradient flow $\dot{\Theta} = f(\Theta)$ and the **discrete update**

$$\Theta_{t+1} = \Theta_t + \varepsilon f(\Theta_t) \quad \overset{\text{to be matched}}{\longleftrightarrow} \quad \Theta(t+\varepsilon) \simeq \Theta(t) + \varepsilon\, f(\Theta(t))$$

Decompose $f(\Theta) = \sum_{n=0}^{\infty} \varepsilon^n f_{(n)}(\Theta)$. Then, we have, after n iterations with step size $\varepsilon = n\,\alpha$:

$$\begin{aligned}\Theta_{t+n} &= \Theta_t + \alpha f(\Theta_t) + \alpha f(\Theta_{t+1}) + \cdots = \Theta_t + \alpha f(\Theta_t) + \alpha f(\Theta_t + \alpha f(\Theta_t)) + \ldots \\ &= \Theta_t + n\alpha f_{(0)}(\Theta_t) + n^2\alpha^2 \left(f_{(1)}(\Theta_t) + \frac{n-1}{2n} \vec{\nabla} f_{(0)}(\Theta_t) \cdot f_{(0)}(\Theta_t) \right) + \ldots\end{aligned}$$

Neural Network Training

Advantages of mini-batch gradient descent

Honest question

Why should we use **mini-batch** gradient descent? What if my entire dataset fits into memory?

Smith et al. (2021)

Consider the case of **gradient descent** (full):

$$f_{(0)}(\Theta) = -\vec{\nabla}\mathcal{L}(\Theta) \quad \text{s.t.} \quad \Theta_{t+1} = \Theta_t - \varepsilon\vec{\nabla}\mathcal{L}(\Theta)$$

Then $n \to \infty$ we need to introduce a *counterterm* if we proceed "step-by-step":

does "renormalisation" ring a bell?

$$\Theta(t+\varepsilon) = \Theta(t) - \varepsilon\mathcal{L}(\Theta) \quad \Leftrightarrow \quad f_{(1)}(\Theta) = -\frac{1}{4}\vec{\nabla}\left|\left|\vec{\nabla}\mathcal{L}(\Theta)\right|\right|_2^2 \quad \Rightarrow \quad \mathcal{L}(\Theta) \leftarrow \mathcal{L}(\Theta) + \frac{1}{4}\left|\left|\vec{\nabla}\mathcal{L}(\Theta)\right|\right|_2^2$$

Neural Network Training

Advantages of mini-batch gradient descent

Honest question

Why should we use **mini-batch** gradient descent? What if my entire dataset fits into memory?

Smith et al. (2021)

Consider now the **mini-batch** loss:

$$f_{(0)} = \widehat{\mathcal{L}}(\Theta) = \frac{1}{B} \sum_{i=0}^{B-1} \mathcal{L}^{(i)}(\Theta) = \frac{1}{B} \sum_{i=0}^{B-1} \frac{1}{|B|} \sum_{k \in B} \mathcal{L}^{k}(\Theta)$$

and compute the **discrete update** over one epoch $\left(n = |B|\right)$:

$$\begin{aligned}
\Theta_B &= \Theta_0 - \varepsilon \vec{\nabla} \mathcal{L}^{(0)}(\Theta_0) - \varepsilon \vec{\nabla} \mathcal{L}^{(1)}(\Theta_1) - \varepsilon \vec{\nabla} \mathcal{L}^{(2)}(\Theta_2) + \ldots \\
&= \Theta_0 - \varepsilon \sum_{i=0}^{B-1} \vec{\nabla} \mathcal{L}^{(i)}(\Theta_0) + \varepsilon^2 \sum_{i=0}^{B-1} \sum_{j<i} \vec{\nabla}\vec{\nabla} \mathcal{L}^{(i)}(\Theta_0) \cdot \vec{\nabla} \mathcal{L}^{(j)}(\Theta_0) + \ldots
\end{aligned}$$

Neural Network Training

Advantages of mini-batch gradient descent

Honest question

Why should we use **mini-batch** gradient descent? What if my entire dataset fits into memory?

Smith et al. (2021)

After one **epoch** is $\varepsilon \ll 1$, the mini-batch update does not introduce **noise**. However, if ε is finite, we have a $O(\varepsilon^2)$ term to keep in mind:

$$\mathbb{E}\left[\Theta_B\right] = \Theta_0 - \varepsilon\, B \vec{\nabla} \mathcal{L}(\Theta_0) + \frac{B^2 \varepsilon^2}{4} \vec{\nabla} \left(\left|\left| \vec{\nabla} \mathcal{L}\left(\Theta_0\right) \right|\right|_2^2 - \frac{1}{B^2} \sum_{i=1}^{B} \left|\left| \vec{\nabla} \mathcal{L}^{(i)}(\Theta_0) \right|\right|_2^2 . \right) + O(B^3 \varepsilon^3)$$

Neural Network Training

Advantages of mini-batch gradient descent

Honest question

Why should we use **mini-batch** gradient descent? What if my entire dataset fits into memory?

Smith et al. (2021)

The first term in the paranthesis comes from the correction to the gradient descent $f_{(1)}$, but there is **one additional term**!
Let us recover the continuous update (the gradient flow):

$$\mathbb{E}\left[\Theta_B\right] \simeq \Theta(B\varepsilon) \quad \Leftrightarrow \quad \mathcal{L}(\Theta) \leftarrow \mathcal{L}(\Theta) + \frac{1}{4}\left|\left|\vec{\nabla}\mathcal{L}(\Theta)\right|\right|_2^2 + \frac{1}{4B}\sum_{i=0}^{B-1}\left|\left|\vec{\nabla}\mathcal{L}^{(i)}(\Theta)\right|\right|_2^2$$

The last is a **regularisation term** added "automatically" by **mini-batch gradient descent**!

Neural Network Training

Optimisation

The naive GD is good but can be improved:

- weight update might get stuck
- weight update might be too slow

A simple example

Let

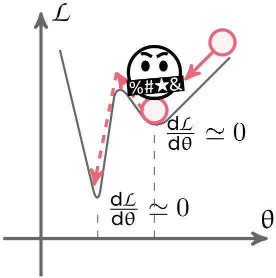
$$\mathcal{L}(\vec{\theta}) = \frac{1}{2}\left(\lambda_1 \theta_1^2 + \lambda_2 \theta_2^2\right), \quad 0 < \lambda_1 < \lambda_2,$$

s.t. $\vec{\nabla}\mathcal{L}(\vec{\theta}) = (\lambda_1\theta_1, \lambda_2\theta_2)$ to compute

$$\vec{\theta}^{(t+1)} = \vec{\theta}^{(t)} - \alpha\vec{\nabla}\mathcal{L}(\vec{\theta}), \quad \alpha > 0,$$

in order to find $\vec{\theta}^{*} = \vec{0}$.

see Waldspurger (CEREMADE)



Neural Network Training
Optimisation

The naive GD is good but can be improved:

- weight update might get stuck
- weight update might be too slow

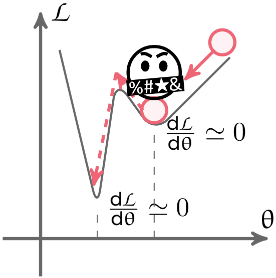
A simple example

We would like

$$\vec{\theta}^{(t+1)} = \left((1-\alpha\lambda_1)\theta_1^{(t)}, (1-\alpha\lambda_2)\theta_2^{(t)}\right)$$

s.t. $|1-\alpha\lambda_i| \ll 1$, for $i = 1, 2$.

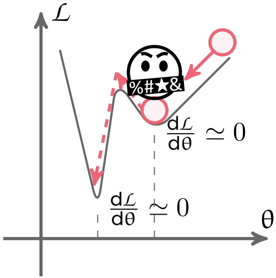
see Waldspurger (CEREMADE)



Neural Network Training
Optimisation

The naive GD is good but can be improved:

- weight update might get stuck
- weight update might be too slow



A simple example

We would like

$$\vec{\theta}^{(t+1)} = \left((1 - \alpha\lambda_1)\theta_1^{(t)}, (1 - \alpha\lambda_2)\theta_2^{(t)} \right)$$

s.t. $|1 - \alpha\lambda_i| \ll 1$, for $i = 1, 2$.

However,

$$\alpha = O(\lambda_1^{-1}) \Rightarrow 1 - \alpha\lambda_2 = 1 - \frac{\lambda_2}{\lambda_1} < 0$$

and the update of θ_2 diverges.

see Waldspurger (CEREMADE)

Neural Network Training
Optimisation

The naive GD is good but can be improved:

- weight update might get stuck
- weight update might be too slow

A simple example

We would like

$$\vec{\theta}^{(t+1)} = \left((1-\alpha\lambda_1)\theta_1^{(t)}, (1-\alpha\lambda_2)\theta_2^{(t)}\right)$$

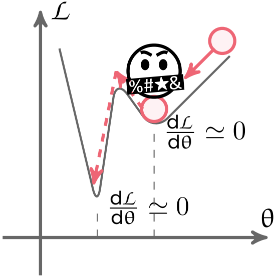
s.t. $|1-\alpha\lambda_i| \ll 1$, for $i = 1, 2$.

And

$$\alpha = \mathcal{O}(\lambda_2^{-1}) \Rightarrow 1 - \alpha\lambda_1 = 1 - \frac{\lambda_1}{\lambda_2} \ll 1$$

and the update of θ_1 is slow.

see Waldspurger (CEREMADE)



REMARKS

1. is the loss *landscape* still "nice"?
2. are all "nice" losses still **convex**?

Neural Network Training

Gradient descent and momentum

Introduce the GD algorithm with **momentum** (Ω set of weights and biases):

Neural Network Training

Gradient descent and momentum

Introduce the GD algorithm with **momentum** (Ω set of weights and biases):

The "Heavy Ball" algorithm | Polyak's Momentum (see Polyak (1964))

Require: $\alpha \in \mathbb{R}^+$, $\Omega^{(0)}$, $\mathcal{L}$, $T \in \mathbb{N} \setminus \{0\}$, $\vec{m}^{(0)} = \vec{0}$

for $0 \leq t < T$ **do**

$\vec{\mathcal{G}}^{(t)} \leftarrow \vec{\nabla}\mathcal{L}(\Omega^{(t)})$

$\vec{m}^{(t+1)} \leftarrow \gamma \vec{m}^{(t)} + (1-\gamma)\vec{\mathcal{G}}^{(t)}$ ▷ *momentum*

$\Omega^{(t+1)} \leftarrow \Omega^{(t)} - \alpha \vec{m}^{(t+1)}$ ▷ *"educated" steepest descent*

return $\Omega^{(T)}$

Neural Network Training

Gradient descent and momentum

Introduce the GD algorithm with **momentum** (Ω set of weights and biases):

The "Heavy Ball" algorithm | Polyak's Momentum (see Polyak (1964))

Require: $\alpha \in \mathbb{R}^+$, $\Omega^{(0)}$, $\mathcal{L}$, $T \in \mathbb{N} \setminus \{0\}$, $\vec{m}^{(0)} = \vec{0}$

for $0 \leq t < T$ **do**

$\vec{\mathcal{G}}^{(t)} \leftarrow \vec{\nabla}\mathcal{L}(\Omega^{(t)})$

$\vec{m}^{(t+1)} \leftarrow \gamma \vec{m}^{(t)} + (1-\gamma)\vec{\mathcal{G}}^{(t)}$ ▷ *momentum*

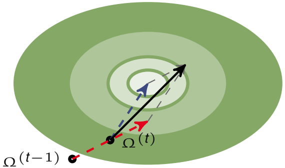
$\Omega^{(t+1)} \leftarrow \Omega^{(t)} - \alpha \vec{m}^{(t+1)}$ ▷ *"educated" steepest descent*

return $\Omega^{(T)}$

This can be equivalently expressed by:

$$\Omega^{(t+1)} = \Omega^{(t)} - \tilde{\alpha}\vec{\nabla} f\left(\Omega^{(t)}\right) + \tilde{\beta}\left(\Omega^{(t)} - \Omega^{(t-1)}\right),$$

where $\tilde{\alpha} = \alpha(1-\gamma)$ and $\tilde{\beta} = \frac{\alpha\gamma}{\alpha - 1}$.



Neural Network Training

Gradient descent and momentum

Introduce the GD algorithm with **momentum** (Ω set of weights and biases):

ADA(ptive) M(omentum estimation) (see Kingma and Ba (2014))

Require: $\alpha \in \mathbb{R}^+$, $\theta^{(0)}$, $\mathcal{L}$, $T \in \mathbb{N} \setminus \{0\}$, $\vec{m}^{(0)} = \vec{0}$, $\vec{v}^{(0)} = \vec{0}$

for $0 \leq t < T$ **do**

$\vec{\mathcal{G}}^{(t)} \leftarrow \vec{\nabla}\mathcal{L}(\Omega^{(t)})$

$\vec{m}^{(t+1)} \leftarrow \beta_1 \vec{m}^{(t)} + (1-\beta_1)\vec{\mathcal{G}}^{(t)}$ ▷ *first momentum estimate* $(\beta_1 = 0.9)$

$\vec{v}^{(t+1)} \leftarrow \beta_2 \vec{v}^{(t)} + (1-\beta_2)\left(\vec{\mathcal{G}}^{(t)}\right)^2$ ▷ *second momentum estimate* $(\beta_2 = 0.999)$

$\widehat{\vec{m}}^{(t+1)} = \frac{\vec{m}^{(t+1)}}{1-\beta_1^t}$

$\widehat{\vec{v}}^{(t+1)} = \frac{\vec{v}^{(t+1)}}{1-\beta_2^t}$

$\Omega^{(t+1)} \leftarrow \Omega^{(t)} - \alpha \frac{\widehat{\vec{m}}^{(t+1)}}{\sqrt{\widehat{\vec{v}}^{(t+1)}}+\varepsilon}$ ▷ *momenta-aware steepest descent*

return $\Omega^{(T)}$

Neural Network Regularisation
Weight decay

Idea: avoid large parameter updates when in advanced training
Response: add an (exponential) "weight decay" term in the optimisation, proportional to the magnitude of the parameters themselves:

$$\Omega^{(t+1)} = (1 - \eta)\, \Omega^{(t)} - \alpha \frac{\partial \mathcal{L}(\Omega^{(t)})}{\partial \Omega^{(t)}}, \quad \eta > 0.$$

(see also Loshchilov and Hutter (2017))

Neural Network Regularisation

Weight decay

Idea: avoid large parameter updates when in advanced training
Response: add an (exponential) "weight decay" term in the optimisation, proportional to the magnitude of the parameters themselves:

$$\Omega^{(t+1)} = (1 - \eta)\, \Omega^{(t)} - \alpha \frac{\partial \mathcal{L}(\Omega^{(t)})}{\partial \Omega^{(t)}}, \quad \eta > 0.$$

The *regularisation* helps reducing the magnitude of the parameters during training.

THINK

- Is weight decay equivalent to a L_2 regularisation term in **vanilla** GD?

$$\mathcal{L}\left(\Omega\right) \leftarrow \mathcal{L}\left(\Omega\right) + \frac{\eta}{2} ||\Omega||_2^2$$

(see also Loshchilov and Hutter (2017))

Neural Network Regularisation

Weight decay

Idea: avoid large parameter updates when in advanced training

Response: add an (exponential) "weight decay" term in the optimisation, proportional to the magnitude of the parameters themselves:

$$\Omega^{(t+1)} = (1 - \eta)\,\Omega^{(t)} - \alpha \frac{\partial \mathcal{L}(\Omega^{(t)})}{\partial \Omega^{(t)}}, \quad \eta > 0.$$

The *regularisation* helps reducing the magnitude of the parameters during training.

THINK

- Is weight decay equivalent to a L_2 regularisation term in **vanilla** GD?

$$\mathcal{L}(\Omega) \leftarrow \mathcal{L}(\Omega) + \frac{\eta}{2} \, ||\Omega||_2^2$$

Technically, iff. $\eta \leftarrow \eta\alpha^{-1}$, but this is usually ignored, so, yes!

(see also Loshchilov and Hutter (2017))

Neural Network Regularisation

Weight decay

Idea: avoid large parameter updates when in advanced training

Response: add an (exponential) "weight decay" term in the optimisation, proportional to the magnitude of the parameters themselves:

$$\Omega^{(t+1)} = (1 - \eta)\, \Omega^{(t)} - \alpha \frac{\partial \mathcal{L}(\Omega^{(t)})}{\partial \Omega^{(t)}}, \quad \eta > 0.$$

The *regularisation* helps reducing the magnitude of the parameters during training.

THINK

- Think about momentum-GD (e.g. ADAM or SGD). Is L_2 regularisation still equivalent to weight decay? Remember that the regularisation leads to

$$\vec{\mathcal{G}}^{(t)} = \vec{\nabla} \mathcal{L} \left(\Omega^{(t)} \right) + \eta\, \Omega^{(t)}$$

in the algorithm. Can you think of a straightforward modification to recover the "correct" weight decay?

(see also Loshchilov and Hutter (2017))

4. Neural Networks

Regularisation of neural networks

Table of contents

Neural Network Regularisation

Batch normalisation

Is there a way to contain the growth/variation of the layers?

Idea: eliminate/reduce the *internal covariate shift*, i.e. the **change of distribution** of the activated outputs $y^{(\ell)} = a^{(\ell)} \odot z^{(\ell)}$

see Ioffe and Szegedy (2015)

Neural Network Regularisation
Batch normalisation

Is there a way to contain the growth/variation of the layers?

Idea: eliminate/reduce the *internal covariate shift*, i.e. the **change of distribution** of the activated outputs $y^{(\ell)} = a^{(\ell)} \odot z^{(\ell)}$

Batch normalisation (training phase)

Introduce $\forall \ell = 1, 2, \dots, N$ and $\forall b = 1, 2, \dots, \mathcal{B}$:

$$\mu_{[b]}^{(\ell)} = \mathbb{E}_{\mathcal{D}_{[b]}}\left[z^{(\ell)}\right] = \frac{1}{\left|\mathcal{D}_{[b]}\right|} \sum_{i=1}^{\left|\mathcal{D}_{[b]}\right|} z_i^{(\ell)}, \qquad \left(\sigma_{[b]}^{(\ell)}\right)^2 = \mathrm{Var}_{\mathcal{D}_{[b]}}\left(z^{(\ell)}\right) = \frac{1}{\left|\mathcal{D}_{[b]}\right|} \sum_{i=1}^{\left|\mathcal{D}_{[b]}\right|} \left(z_i^{(\ell)} - \mu_{[b]}^{(\ell)}\right)^2$$

see Ioffe and Szegedy (2015)

Neural Network Regularisation

Batch normalisation

Is there a way to contain the growth/variation of the layers?

Idea: eliminate/reduce the *internal covariate shift*, i.e. the **change of distribution** of the activated outputs $y^{(\ell)} = a^{(\ell)} \odot z^{(\ell)}$

Batch normalisation (training phase)

Normalize $\forall \ell = 1, 2, \dots, N$ and $\forall b = 1, 2, \dots, \mathcal{B}$:

$$\hat{z}^{(\ell)}_{[b]} = \frac{z^{(\ell)} - \mu^{(\ell)}_{[b]}}{\sigma^{(\ell)}_{[b]} + \varepsilon}$$

see Ioffe and Szegedy (2015)

Neural Network Regularisation
Batch normalisation

Is there a way to contain the growth/variation of the layers?

Idea: eliminate/reduce the *internal covariate shift*, i.e. the **change of distribution** of the activated outputs $y^{(\ell)} = a^{(\ell)} \odot z^{(\ell)}$

Batch normalisation (training phase)

Interpolate $\forall \ell = 1, 2, \dots, N$ and $\forall b = 1, 2, \dots, \mathcal{B}$:

$$\mathrm{BN}^{(\ell)}_{[b]} \left(\widehat{z}^{(\ell)}; \gamma, \beta \right) = \gamma_{[b]}\, \widehat{z}^{(\ell)}_{[b]} + \beta_{[b]},$$

where $\gamma^{(\ell)}$ and $\beta^{(\ell)}$ are **learnable** scalars

see Ioffe and Szegedy (2015)

Neural Network Regularisation
Batch normalisation

Is there a way to contain the growth/variation of the layers?

Idea: eliminate/reduce the *internal covariate shift*, i.e. the **change of distribution** of the activated outputs $y^{(\ell)} = a^{(\ell)} \odot z^{(\ell)}$

Batch normalisation (training phase)

Replace $\forall \ell = 1, 2, \dots, N$ and $\forall b = 1, 2, \dots, \mathcal{B}$:

$$y^{(\ell)} = a^{(\ell)} \odot z^{(\ell)} \quad \rightarrow \quad \widehat{y}^{(\ell)}_{[b]} = a^{(\ell)} \odot \mathrm{BN}^{(\ell)}_{[b]} \left(\widehat{z}^{(\ell)}; \gamma, \beta \right)$$

see Ioffe and Szegedy (2015)

Neural Network Regularisation
Batch normalisation

Is there a way to contain the growth/variation of the layers?

Idea: eliminate/reduce the *internal covariate shift*, i.e. the **change of distribution** of the activated outputs $y^{(\ell)} = a^{(\ell)} \odot z^{(\ell)}$

Batch normalisation (inference phase)

Inference **must not** depend on batch size!
Compute (after training) $\forall \ell = 1, 2, \dots, N$:

$$\mu^{(\ell)} = \frac{1}{\mathcal{B}} \sum_{b=1}^{\mathcal{B}} \mu_{[b]}^{(\ell)}, \quad \left(\sigma^{(\ell)}\right)^2 = \frac{1}{\mathcal{B}} \sum_{b=1}^{\mathcal{B}} \left(\mu_{[b]}^{(\ell)} - \mu^{(\ell)}\right)^2 .$$

N.B.: the original paper uses the *unbiased* estimate of the variance.

see Ioffe and Szegedy (2015)

Neural Network Regularisation
Batch normalisation

Is there a way to contain the growth/variation of the layers?

Idea: eliminate/reduce the *internal covariate shift*, i.e. the **change of distribution** of the activated outputs $y^{(\ell)} = a^{(\ell)} \odot z^{(\ell)}$

Batch normalisation (inference phase)

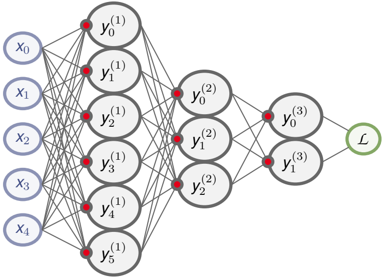
Inference **must not** depend on batch size!

Replace all $\mathrm{BN}^{(\ell)}$ operations $\forall \ell = 1, 2, \dots, N$:

$$\mathrm{BN}^{(\ell)}_{[b]}\left(\cdot; \gamma, \beta\right) \quad \to \quad \frac{\gamma}{\sigma^{(\ell)} + \varepsilon} z^{(\ell)} + \left(\beta - \frac{\gamma \mu^{(\ell)}}{\sigma^{(\ell)} + \varepsilon} \right)$$

see Ioffe and Szegedy (2015)

Neural Network Regularisation
Dropout



Some remarks:

- NNs are **high variance** models
- some paths might be strongly correlated (**co-adaptation**)

For the sake of simplicity, consider the following regression **linear model** (homogeneous) for $i = 1, 2, \dots, h$:

$$y_i = f_i(\vec{x}) = \sum_{j=1}^{k} W_{ij} x_j$$

and let $Q \in \{0, 1\}^{h \times k}$ a matrix of Bernoulli variables

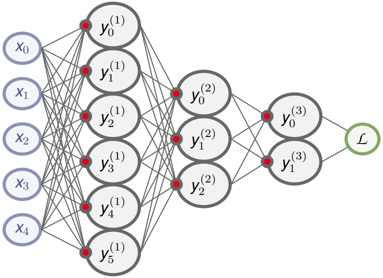
$$\mathrm{P}(Q_{ij} = 1) = p = 1 - \mathrm{P}(Q_{ij} = 0),$$

for $i = 1, 2, \dots, h$, and $j = 1, 2, \dots, k$.
In other words, $\mathbb{E}[Q_{ij}] = p$, and $\mathrm{Var}(Q_{ij}) = p(1 - p)$:

(see Wager et al. (2013))

Neural Network Regularisation

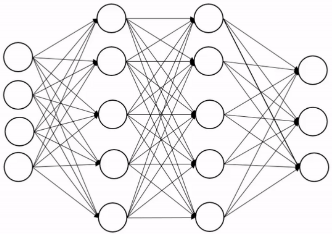
Dropout



Then, define the "dropout model":

$$\tilde{y}_i = \tilde{f}_i(\vec{x}) = \left(\left(Q \odot W\right)\vec{x}\right)_i = \sum_{j=1}^{k} Q_{ij} W_{ij} x_j$$

This is equivalent to any hidden layer in a NN:

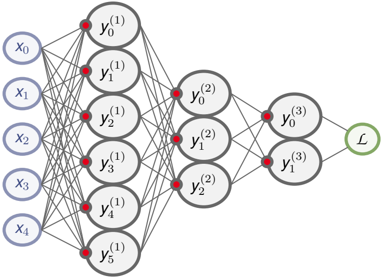


Some remarks:

- NNs are **high variance** models
- some paths might be strongly correlated (**co-adaptation**)

(see Wager et al. (2013))

Neural Network Regularisation
Dropout



We can compute the loss

$$\mathcal{L}(y, \widetilde{y}) = \frac{1}{2n} \sum_{i=1}^{n} \sum_{j=1}^{k} \left(y_i - Q_{ij} W_{ij} x_j\right)^2 ,$$

whose gradients are:

$$\frac{\partial \widetilde{\mathcal{L}}}{\partial W_{ij}} = -Q_{ij} x_j \left(y_i - \sum_{t=1}^{k} Q_{it} W_{it} x_t \right)$$

$$= -Q_{ij} y_i x_j + Q_{ij}^2 W_{ij} x_j^2 + \sum_{t=1,\, t \neq j}^{k} Q_{ij} Q_{it} W_{it} x_j x_t$$

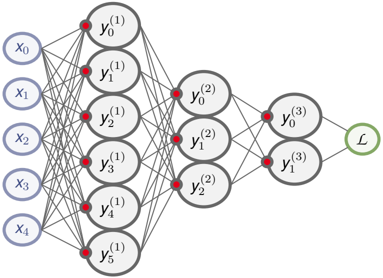
(see Wager et al. (2013))

Some remarks:

- NNs are **high variance** models
- some paths might be strongly correlated (**co-adaptation**)

Neural Network Regularisation

Dropout



Some remarks:

- NNs are **high variance** models
- some paths might be strongly correlated (**co-adaptation**)

Let us compute the average value of the gradients over the **dropout** distribution:

$$\mathbb{E}\left[\frac{\partial\widetilde{\mathcal{L}}}{\partial W_{ij}}\right] = -\mathbb{E}\left[Q_{ij}\right] y_i x_j + \mathbb{E}\left[Q_{ij}^2\right] W_{ij} x_j^2 + \sum_{t=1,\, t\neq j}^{k} \mathbb{E}\left[Q_{ij}\right]\mathbb{E}\left[Q_{it}\right] W_{it} x_j x_t.$$

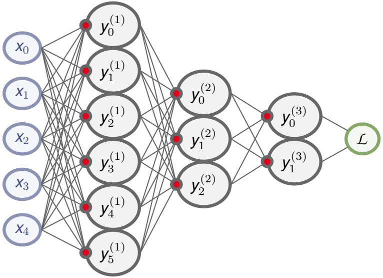
Remember that $\mathbb{E}\left[X^2\right] = \mathrm{Var}(X) + \mathbb{E}\left[X\right]^2$:

$$\mathbb{E}\left[\frac{\partial\widetilde{\mathcal{L}}}{\partial W_{ij}}\right] = -p y_i x_j + p^2 W_{ij} x_j^2 + p\,(1-p)\, W_{ij} x_j^2 + p^2 \sum_{t=1,\, t\neq j}^{k} W_{it} x_j x_t.$$

(see Wager et al. (2013))

Neural Network Regularisation

Dropout



Some remarks:

- NNs are **high variance** models
- some paths might be strongly correlated (**co-adaptation**)

Though perturbed by constants p and p^2, we can reconstruct the usual loss + a regularisation:

$$\mathbb{E}\left[\frac{\partial \widetilde{\mathcal{L}}}{\partial W_{ij}}\right] \sim \mathbb{E}\left[\frac{\partial \mathcal{L}_p}{\partial W_{ij}}\right] + p\,(1-p)\,W_{ij} x_j^2\,.$$

Dropout

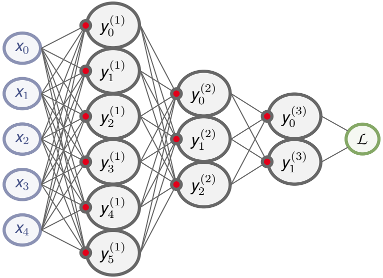
Dropout is a "feature noising" **regularisation** technique, which **can** be applied to **NN**s to prevent **overfitting** and **co-adaption**.

Dropout replaced by Id in inference (usually).

(see Wager et al. (2013))

Neural Network Regularisation

Dropout



Some remarks:

- NNs are **high variance** models
- some paths might be strongly correlated (**co-adaptation**)

Dropout

Dropout is a "feature noising" **regularisation** technique, which **can** be applied to **NNs** to prevent **overfitting** and **co-adaption**.

The elegant idea is to average the outputs of the models over a **noise** component ξ. Consider an exponential family of likelihood functions, and take the log-partition function A:

$$\mathcal{L}\left(\mathbb{E}_{\xi}\left[A\left(\Omega_{\xi}, x\right)\right]\right) = \mathcal{L}\left(A\left(\Omega, x\right)\right) + \mathrm{R}\left(\Omega\right),$$

where the regularisation

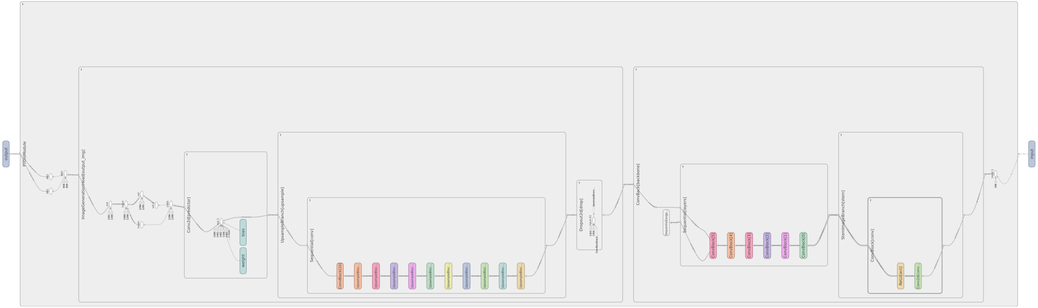
$$\mathrm{R}\left(\Omega\right) \sim \mathbb{E}_{\xi}\left[A\left(\Omega_{\xi}, x\right)\right] - A\left(\Omega, x\right) \simeq \frac{1}{2} A''\left(\Omega, x\right) \mathrm{Var}_{\xi}\left(A\left(\Omega_{\xi}, x\right)\right),$$

in the case $\mathbb{E}_{\xi}\left[\Omega_{\xi}\right] = \Omega$.

(see Wager et al. (2013))

Neural Networks

Graph visualisation



Every single object has its place in the model, which is (should be) well connected from input to output. This makes it exportable, reusable and deployable.

5. Conclusions

Concluding remarks

Table of contents

Conclusions

A quick summary

In summary, you should now have a good understanding of:

- what **ML** is, and what are its **principles**
- how to work with **ML pipelines** for high quality research
- how to perform **validation** of ML models
- how to **evaluate** the performance of ML models
- what is the **“variance vs bias”** trade-off
- what a **loss function** is, and what its purpose
- several **regularisation** techniques
- different **unsupervised and supervised techniques** (including ensembles)
- what a **NN** is and how to train one

Hoping that I did not bore anyone to death…

Bibliography I
Some personal favourites

[1] C. Albon, *Machine Learning with Python Cookbook: Practical Solutions from Preprocessing to Deep Learning*. O'Reilly Media, Inc., 2018, isbn: 978-1-4919-8938-8. [Online]. Available: `https://www.oreilly.com/library/view/machine-learning-with/9781491989371/`.

[2] CNRS, "Formation Introduction au Deep LEarning (FIDLE)," (), [Online]. Available: `https://fidle.cnrs.fr/w3/` (visited on 09/27/2024).

[3] R. A. Caruana, "Multitask Learning," *Machine Learning*, vol. 28, no. 1, pp. 41–75, 1997, issn: 1573-0565. doi: `10.1023/A:1007379606734`.

[4] R. A. Caruana, "Multitask Learning: A Knowledge-Based Source of Inductive Bias," in *Machine Learning Proceedings 1993*, San Francisco (CA): Morgan Kaufmann, 1993, pp. 41–48, isbn: 978-1-55860-307-3. doi: `10.1016/B978-1-55860-307-3.50012-5`.

[5] F. Chollet, *Deep Learning with Python*. Manning Publications Co., 2018, 384 pp., isbn: 978-1-61729-443-3. [Online]. Available: `https://www.manning.com/books/deep-learning-with-python`.

[6] R. Couillet and Z. Liao, *Random Matrix Methods for Machine Learning*. Cambridge: Cambridge University Press, 2022, isbn: 978-1-00-912323-5. doi: `10.1017/9781009128490`.

[7] M. Crawshaw, "Multi-Task Learning with Deep Neural Networks: A Survey," arXiv: `2009.09796 [cs, stat]`. (2020), [Online]. Available: `http://arxiv.org/abs/2009.09796` (visited on 08/10/2022), pre-published.

[8] A. J. Ferreira and M. A. T. Figueiredo, "Boosting Algorithms: A Review of Methods, Theory, and Applications," in *Ensemble Machine Learning: Methods and Applications*, C. Zhang and Y. Ma, Eds., New York, NY: Springer, 2012, pp. 35–85, isbn: 978-1-4419-9326-7. doi: `10.1007/978-1-4419-9326-7_2`.

[9] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. [Online]. Available: `https://www.deeplearningbook.org/`.

[10] C. Gruber, P. O. Schenk, M. Schierholz, *et al.*, "Sources of Uncertainty in Machine Learning – A Statisticians' View," arXiv: `2305.16703 [cs, stat]`. (2023), pre-published.

[11] Z. Hao, S. Liu, Y. Zhang, *et al.*, "Physics-Informed Machine Learning: A Survey on Problems, Methods and Applications," arXiv: `2211.08064 [cs, math]`. (2023), pre-published.

[12] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning* (Springer Series in Statistics). New York, NY: Springer New York, 2009, isbn: 978-0-387-84857-0 978-0-387-84858-7. doi: `10.1007/978-0-387-84858-7`.

Bibliography II

Some personal favourites

[13] MIT Open Learning Library, "Introduction to Machine Learning," (2020), [Online]. Available: `https://openlearninglibrary.mit.edu/courses/course-v1:MITx+6.036+1T2019/about`.

[14] P. Mehta, M. Bukov, C.-H. Wang, *et al.*, "A high-bias, low-variance introduction to Machine Learning for physicists," *Physics Reports*, vol. 810, pp. 1–124, 2019, issn: 03701573. doi: `10.1016/j.physrep.2019.03.001`. arXiv: `1803.08823 [physics.comp-ph]`.

[15] S. S. Skiena, *The Data Science Design Manual*. Springer, 2017, isbn: 978-3-319-55444-0. [Online]. Available: `https://link.springer.com/book/10.1007/978-3-319-55444-0`.

[16] J. Snoek, H. Larochelle, and R. P. Adams, "Practical Bayesian Optimization of Machine Learning Algorithms," in *Advances in Neural Information Processing Systems*, vol. 25, Curran Associates, Inc., 2012. [Online]. Available: `https://proceedings.neurips.cc/paper/2012/hash/05311655a15b75fab86956663e1819cd-Abstract.html` (visited on 09/27/2024).

[17] Standford Online, "CS229: Machine Learning," (2024), [Online]. Available: `https://cs229.stanford.edu/`.

[18] Standford Online, "Machine Learning Specialization," (2022), [Online]. Available: `https://online.stanford.edu/courses/soe-ymls-machine-learning-specialization`.

[19] R. Vidal, J. Bruna, R. Giryes, *et al.*, "Mathematics of Deep Learning," 2017. arXiv: `1712.04741 [cs]`. [Online]. Available: `http://arxiv.org/abs/1712.04741` (visited on 07/20/2021).

[20] A. Zheng and A. Casari, *Feature Engineering for Machine Learning: Principles and Techniques for Data Scientists*. O'Reilly Media, Inc., 2018, isbn: 978-1-4919-5324-2. [Online]. Available: `https://www.oreilly.com/library/view/feature-engineering-for/9781491953235`.

In bocca al lupo !

CEA SACLAY
91191 Gif-sur-Yvette Cedex
France
riccardo.finotello@cea.fr