Apprentissage-Automatisé

Release 1.0.0-beta0

IVIA-AF Team

Dec 10, 2023

Contents

1 | Introduction Générale à l'Apprentissage Automatique

Nous parlerons de:

- Apprentissage Supervisé
- Apprentissage Non-Supervisé
- Les méthodes à noyaux (Kernel methods)
- Apprentissage par Renforcement

Motivations et les applications pour chaque type d'apprentissage.

1.1 C'est quoi Apprentissage Automatique?

Fig. [1.1].

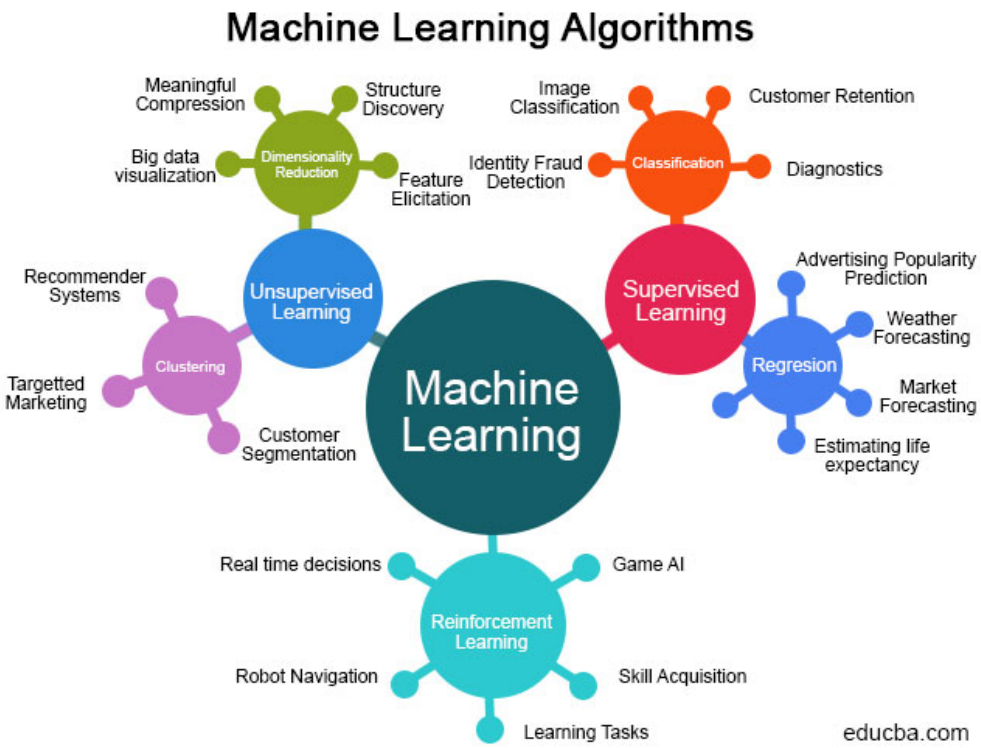


Fig. 1.1.1: Les types d'I.A

1.2 Convention Mathématiques pour le document

- Les matrices seront notées en lettre **majuscule** et seront mises en **gras**. Par exemple,

  $\mathbf{X}$

- Les vecteurs seront notés en lettre **miniscule** et mises en **gras**. Par exemple,

  $\mathbf{x}$.

- Lécriture mathématique de probabilités, espérance seront respectivement:

  $\mathbb{P}$, $\mathbb{E}$

- Il sera aussi important de ponctuer les équations.
- Numéroter les équations principales.
- Tous les ensemble seront notés en utilisant

  $\mathbb{R}$

  par example.

- les expressions mathématiques qui sont écrites à travers les textes seront écrites dans

  Prob

- Si c'est un symbole qui est un vecteur, on écrit (par exemple, si c'est alpha)

  $\boldsymbol{\alpha}$

  par exemple.

2 | Pré-requis

Python est le langage de programmation préféré des Data Scientistes. Ils ont besoin d'un langage facile à utiliser, avec une disponibilité décente des bibliothèques et une grande communauté. Les projets ayant des communautés inactives sont généralement moins susceptibles de mettre à jour leurs plates-formes. Mais alors, pourquoi Python est populaire en Data Science ?

Python est connu depuis longtemps comme un langage de programmation simple à maîtriser, du point de vue de la syntaxe. Python possède également une communauté active et un vaste choix de bibliothèques et de ressources. Comme résultat, vous disposez d'une plate-forme de programmation qui est logique d'utiliser avec les technologies émergentes telles que l'apprentissage automatique (Machine Learning) et la Data Science.

2.1 Langage Python et ses Librairies

Python est un langage de programmation puissant et facile à apprendre. Il dispose de structures de données de haut niveau et permet une approche simple mais efficace de la programmation orientée objet. Parce que sa syntaxe est élégante, que son typage est dynamique et qu'il est interprété, Python est un langage idéal pour l'écriture de scripts quand on fait de l'apprentissage automatique et le développement rapide d'applications dans de nombreux domaines et sur la plupart des plate-formes.

2.1.1 Installation de Python et Anaconda

L'installation de Python peut-être un vrai challenge. Déjà il faut se décider entre les versions 2.X et 3.X du langage, par la suite, choisir les librairies nécessaires (ainsi que les versions compatibles) pour faire de l'apprentissage automatique (Machine Learning); sans oublier les subtilités liées aux différents Systèmes d'exploitation (Windows, Linux, Mac…) qui peuvent rendre l'installation encore plus *"douloureuse"*.

Dans cette partie nous allons installer pas à pas un environnement de développement Python en utilisant Anaconda[^1]. A l'issue de cette partie, nous aurons un environnement de développement fonctionnel avec les librairies (packages) nécessaires pour faire de l'apprentissage automatique (Machine Learning).

Qu'est ce que Anaconda ?

L'installation d'un environnement Python complet peut-être assez complexe. Déjà, il faut télécharger Python et l'installer, puis télécharger une à une les librairies (packages) dont on a besoin. Parfois, le nombre de ces librairies peut-être grand. Par ailleurs, il faut s'assurer de la compatibilité entre les versions des différents packages qu'on a à télécharger. *Bref, ce n'est pas amusant*!

Alors Anaconda est une distribution Python. À son installation, Anaconda installera Python ainsi qu'une multitude de packages dont vous pouvez consulter la liste[1]. Cela nous évite de nous ruer dans les problèmes

[1] https://docs.anaconda.com/anaconda/packages/pkg-docs/#Python-3-7

d'incompatibilités entre les différents packages. Finalement, Anaconda propose un outil de gestion de packages appelé Conda. Ce dernier permettra de mettre à jour et installer facilement les librairies dont on aura besoin pour nos développements.

Téléchargement et Installation de Anaconda

Note: Les instructions qui suivent ont été testées sur Linux/Debian. Le même processus d'installation pourra s'appliquer pour les autres systèmes d'exploitation.

Pour installer Anaconda sur votre ordinateur, vous allez vous rendre sur le site officiel[2] depuis lequel l'on va télécharger directement la dernière version d'Anaconda. Prenez la version du binaire qu'il vous faut :

- Choisissez le système d'exploitation cible (Linux, Windows, Mac, etc…)
- Sélectionnez la version 3.X (à l'heure de l'écriture de ce document, c'est la version 3.8 qui est proposée, surtout pensez à toujours installer la version la plus récente de Python), compatible (64 bits ou 32 bits) avec l'architecture de votre ordinateur.

Après le téléchargement, si vous êtes sur Windows, alors rien de bien compliqué double cliquez sur le fichier exécutable et suivez les instructions classique d'installation d'un logiciel sur Windows.

Si par contre vous êtes sur Linux, alors suivez les instructions qui suivent:

- Ouvrez votre terminal et rassurez vous que votre chemin accès est celui dans lequel se trouve votre fichier d'installation.
- Exécutez la commande: $ bash Anaconda3-2020.02-Linux-x86_64.sh, rassurez vous du nom du fichier d'installation, il peut changer selon la version que vous choisissez.

Après que l'installation se soit déroulée normalement, éditez le fichier caché **.bashrc** pour ajouter le chemin d'accès à Anaconda. Pour cela exécutez les commandes suivantes:

- $ cd ~
- $ gedit .bashrc
- Ajoutez cette commande à la dernière ligne du fichier que vous venez d'ouvrir
- export PATH= ~/anaconda3/bin:$PATH

Maintenant que c'est fait, enregistrez le fichier et fermez-le. Puis exécutez les commandes suivantes

- $ conda init
- $ Python

Pour ce qui est de l'installation sur Mac, veuillez suivre la procédure d'installation dans la documentation d'Anaconda[3].

Il existe une distribution appelée Miniconda[4] qui est un programme d'installation minimal gratuit pour conda. Il s'agit d'une petite version bootstrap d'Anaconda qui inclut uniquement conda, Python, les packages dont ils dépendent, et un petit nombre d'autres packages utiles.

Terminons cette partie en nous familiarisant avec quelques notions de la programmation Python.

Première utilisation de Anaconda

La distribution Anaconda propose deux moyens d'accéder à ses fonctions: soit de manière graphique avec Anaconda-Navigator, soit en ligne de commande (depuis Anaconda Prompt sur Windows, ou un terminal pour

[2] http://docs.anaconda.com/anaconda/navigator/
[3] https://docs.anaconda.com/anaconda/install/mac-os/
[4] https://docs.conda.io/en/latest/miniconda.html

Linux ou MacOS). Sous Windows ou MacOs, démarrez Anaconda-Navigator dans le menu des programmes. Sous Linux, dans un terminal, tapez la commande : $ anaconda-navigator (cette commande est aussi disponible dans le prompt de Windows). Anaconda-Navigator propose différents services (déjà installés, ou à installer). Son onglet Home permet de lancer le service désiré. Les principaux services à utiliser pour développer des programmes Python sont :

- Spyder
- IDE Python
- Jupyter notebook et jupyter lab : permet de panacher des cellules de commandes Python (code) et des cellules de texte (Markdown).

Pour la prise en main de Python nous allons utiliser jupyter lab.

2.1.2 Prise en main de Python

Nous avons préparé un notebook qui nous permettra d'aller de zèro à demi Héros en Python. Le notebook se trouve ici[5].

Python pour les debutants : Notions de bases du python

Les nombres avec Python!

Dans cette partie, nous allons tout apprendre comment utiliser les nombres sur Python.

Nous allons couvrir les points suivants :

1. Les types de nombres sur Python
2. Arithmetique de base sur les nombres
3. Différences entre Python 2 et 3
4. Assignation d'objets dans Python

Les types de nombres

Python a plusieurs "types" de nombres. Nous nous occuperons principalement des entiers (integers) et des nombres à virgule flottante (float).

Le type entier (integer) représente des nombres entiers, positifs ou negatifs. Par exemple: 7 et -1 sont des integers.

Le type flottant (float) de Python est facile à identifier parce que les nombres sont représentés avec la partie décimal (Attention, ici les nombre decimaux sont representer avec un point et non la virgule comme habituellement en langue française), ou alors avec le signe exponentiels (reprensenter par **e**) pour représenter les puissances de 10. Par exemple, 2.0 et -2.1 sont des nombres de type flottant. 2e3 (2 fois 10 à la puissance 3) est aussi un nombre de type flottant en Python.

Voici un tableau des deux types que nous manipulerons le plus dans ce cours:

[5] https://colab.research.google.com/drive/1zILtNrCmPDFyQQ1Ev1H4jeHx7FuyEZ27?usp=sharing

Table 2.1.1: Les types de nomber en Python.

Exemples	Type
1, 4, -2, 100	Integers
1.4, 0.3, -0.5, 2e2, 3e2	Floating-point numbers (float)

Voyons maintenant, quelques examples d'arithmétique simples que l'on peux appliquer sur ses elements.

Arithmétique

```
# Addition de 1 et 3 = 4
1+3
```

```
4
```

```
# Soustraction de 1 dans 3 = 2
3-1
```

```
2
```

```
# Multiplication de 2 par 2 = 4
2*2
```

```
4
```

```
# Division de 5 par 2 = 2.5 (remaquez qu'on a la un flottant)
5/2
```

```
2.5
```

```
# Puissances. 2 a la  puissances 3 = 8
2**3
```

```
8
```

```
#la racine carree peut se calculer  de la même manière = 2.0
4**0.5
```

```
2.0
```

Ordre de priorite

```
# Voyons quel ordre de priorite python fais sur les operteurs
2 + 10 * 10 + 3  # avec ceci nous obtenons un resultats errone = 105
```

```
105
```

```
# Avec des parenthèse nous pouvons garder le contrôle de ces priorités
(2+10) * (10+3)
```

```
156
```

Les variables

Maintenant que nous savons comment utiliser python en mode calculette, nous pouvons créer des variables et leur assigner des valeurs comme des nombres. Notons que l'utilisation des variables sur python est un tout petit peu different que ce qui ce passe dans d'autre langage de programmation. Contrairement a d'autre language, python utilise une technique de typage faible. Les variables ne sont pas declarees a l'avance avant l'untilisation.

Il suffit d'un simple signe égal = pour assigner un nom à une variable. Voyons quelques exemples pour détailler la façon de faire.

```
# Créons un objet nommé "x" et "y" et nous leur assignons la valeur 5 et .2␣
↪respectivement
x = 5 # ici la variable x est de type integer
y = .2 # ici la variable y est de type flottant (notons que 0.2 peut aussi s
↪'ecrire .2 tout simplement)
```

Maintenant, nous pouvons appeler x or y dans un script Python qui les traitera comme le nombre 5 ou .2 respectivement

```
# additionner des objets
x+y
```

```
5.2
```

```
# Ré-assignation
x = 10
```

```
# Vérification
x
```

```
10
```

```
# Nous utilisons x pour assigner x de nouveau
x = x + x
```

```
# Vérification
x
```

```
20
```

```
# Incrementation par 1 (on peux aussi incrementer avec n'importe quel nombre)
x = x+1
```

```
x
```

```
21
```

```
#Autre syntaxe pour incrementer une variable
x += 1
```

```
x
```

```
22
```

```
#Decrementation par 1
x = x -1
```

```
x
```

```
21
```

```
#Autre syntaxe pour decrementer une variable
x -= 1
```

```
x
```

```
20
```

```
#Nous pouvons aussi multiplier l'ancienne valeur de x par un quelconque␣
↪nombre.
x *=2
```

```
x
```

```
40
```

```
#Nous pouvons aussi diviser l'ancienne valeur de x par un quelconque nombre.
x /=2
```

```
x
```

```
20.0
```

NB: Toutes les operations que nous venons de voir s'applique aussi sur les flottants

Tout de meme, voici quelques règles à respecter pour créer un nom de variable :

1. Les noms ne doivent pas commencer par un nombre.
2. Pas d'espace dans les noms, utilisez _ à la place
3. Interdit d'utiliser les symboles suivants : '",<>/?|()!@#$%^&*~-+
4. C'est une bonne pratique (PEP8) de mettre les noms en minuscules

Les noms de variables permettent de conserver et manipuler plusieurs valeurs de façon simple avec Python.

```
# Utilisez toujours des noms explicites pour mieux suivre ce que fait votre␣
↪code !
prix_vente = 280

prix_unitaire = 150

benefice = prix_vente - prix_unitaire
```

```
# Visualiser le benefice !
benefice
```

```
130
```

2.2 Les Bases Mathématiques pour l'Apprentissage Automatique

Dans cette section, nous allons présenter les notions mathématiques essentielles à l'apprentissage automatique (machine learning). Nous n'aborderons pas les théories complexes des mathématiques afin de permettre aux débutants (en mathématiques) ou mêmes les personnes hors du domaine mais intéressées à l'apprentissage automatique de pouvoir en profiter.

2.2.1 Algèbre linéaire et Analyse

Définition d'espaces vectoriels. Un espace vectoriel est un triplet $(V, +, *)$ formé d'un ensemble V muni de deux lois,

$$\begin{aligned} + : \quad & V \times V \longrightarrow V \\ & (u, v) \mapsto u + v \end{aligned}$$

et (2.2.1)

$$\begin{aligned} * : \quad & \mathbb{K} \times V \longrightarrow V, \text{ avec } \mathbb{K} \text{ un corps commutatif} \\ & (\lambda, v) \mapsto \lambda * v = \lambda v \end{aligned}$$

qui vérifient:

1. associativité de $+ : \forall\, u, v, w \in V, \quad (u + v) + w = u + (v + w)$
2. commutativité de $+ : \forall\, u, v \in V, \quad u + v = v + u$
3. existence d'élément neutre pour $+ : \exists\, e \in V : \forall\, u \in V, \quad u + e = e + u = u$
4. existence d'élément opposé pour $+ : \forall\, u \in V, \exists\, v \in V : u + v = v + u = 0$. On note $v = -u$ et v est appelé l'opposé de u
5. existence de l'unité pour $* : \exists\, e \in \mathbb{K}$ tel que $\forall\, u \in V, \quad e * u = u$
6. associativité de $* : \forall\, (\lambda_1, \lambda_2, u) \in \mathbb{K} \times \mathbb{K} \times V, \quad (\lambda_1 \lambda_2) * u = \lambda_1 * (\lambda_2 * u)$
7. somme de vecteurs (distributivité de * sur +) : $\forall\, (\lambda, u, v) \in \mathbb{K} \times V \times V, \quad \lambda * (u + v) = \lambda * u + \lambda * v$
8. $: \forall\, (\lambda_1, \lambda_2, u) \in \mathbb{K} \times \mathbb{K} \times V, \quad (\lambda_1 + \lambda_2) * u = \lambda_1 * u + \lambda_2 * u.$

Remarque 1: Les éléments de V sont appelés des **vecteurs**, ceux de $\mathbb{K}$ sont appelés des **scalaires** et l'élément neutre pour $+$ est appelé **vecteur nul**. Finalement, V est appelé $\mathbb{K}$-espace vectoriel ou espace vectoriel sur $\mathbb{K}$.
Base d'un espace vectoriel. Soit V un $\mathbb{K}$-espace vectoriel. Une famille de vecteurs $\mathcal{B} = \{b_1, b_2, \ldots, b_n\}$ est appelée base de V si les deux propriétés suivantes sont satisfaites:

- $\forall\, u \in V, \exists\, c_1, \ldots, c_n \in \mathbb{K}$ tels que $u = \sum_{i=1}^{n} c_i b_i$ (On dit que $\mathcal{B}$ est **une famille génératrice** de V).
- $\forall\, \lambda_1, \ldots, \lambda_n \in \mathbb{K}, \quad \sum_{i=1}^{n} \lambda_i b_i = 0 \Longrightarrow \lambda_i = 0 \quad \forall\, i.$ (On dit que les éléments de $\mathcal{B}$ sont *linéairement indépendants*).

Lorsque $u = \sum_{i=1}^{n} c_i b_i$, on dit que $c_1, \ldots, c_n$ sont les coordonnées de u dans la base $\mathcal{B}$. Si de plus aucune confusion n'est à craindre, on peut écrire:

$$\mathbf{u} = \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_n \end{bmatrix}. \tag{2.2.2}$$

Définition. Le nombre d'éléments dans une base d'un espace vectoriel est appelé **dimension** de l'espace vectoriel.

NB: Un espace vectoriel ne peut être vide (il contient toujours le vecteur nul). L'**espace vectoriel nul** $\{0\}$ n'a pas de base et est **de dimension nulle**. Tout **espace vectoriel non nul** de dimension finie admet une infinité de bases mais sa **dimension est unique**.

Exemples d'espaces vectoriels: Pour tous $n, m \geq 1$, l'ensemble des matrices $\mathcal{M}_{nm}$ à coefficients réels et l'ensemble $\mathbb{R}^n$ sont des $\mathbb{R}$-espace vectoriels. En effet, il est très facile de vérifier que nos exemples satisfont les huit propriétés énoncées plus haut. Dans le cas particulier $V = \mathbb{R}^n$, toute famille d'exactement n vecteurs linéairement indépendants en est une base. En revanche, toute famille de moins de n vecteurs ou qui contient plus que n vecteurs ne peut être une base de $\mathbb{R}^n$.

Matrices: Soit $\mathbb{K}$ un corps commutatif. Une matrice en mathématiques à valeurs dans $\mathbb{K}$ est un tableau de nombres, où chaque nombre est un élément de $\mathbb{K}$. Chaque ligne d'une telle matrice est un vecteur (élément d'un $\mathbb{K}$-espace vectoriel). Une matrice est de la forme:

$$\mathbf{M} = \begin{bmatrix} a_{11} & a_{12} \dots a_{1n} \\ a_{21} & a_{22} \dots a_{2n} \\ \vdots & \\ a_{m1} & a_{m2} \dots a_{mn} \end{bmatrix}. \tag{2.2.3}$$

On note aussi $\mathbf{M} = \big(a_{ij}\big)_{1 \leq i \leq m, 1 \leq j \leq n}$.

La matrice ci-dessus est carrée si $m = n$. Dans ce cas, la suite $[a_{11}, a_{22}, \dots, a_{mm}]$ est appelée **diagonale** de M. Si tous les coefficients hors de la diagonale sont zéro, on dit que la matrice est diagonale. Une matrice avec tous ses coefficients nuls est dite matrice **nulle**.

Produit de matrices. Soient $\mathbf{A} = \big(a_{ij}\big)_{1 \leq i \leq m, 1 \leq j \leq n}, \mathbf{B} = \big(b_{ij}\big)_{1 \leq i \leq n, 1 \leq j \leq q}$ deux matrices.
On définit le produit de **A** par **B** et on note $\mathbf{A} \times \mathbf{B}$ ou simplement **AB**, la matrice M définie par:

$$\mathbf{M}_{ij} = \sum_{\ell=1}^{n} a_{i\ell} b_{\ell j}, \text{ pour tout } i \text{ et } j. \tag{2.2.4}$$

Important.

- Le produit **AB** est possible si et seulement si le nombre de colonnes de **A** est égal au nombre de lignes de **B**.
- Dans ce cas, **AB** a le même nombre de lignes que **A** et le même nombre de colonnes que **B**.
- Un autre point important à noter est que le produit matriciel n'est pas commutatif (**AB** n'est pas toujours égal à **BA**).

Exemple. Soient les matrices **A** et **B** définies par:

$$\mathbf{A} = \begin{bmatrix} 2 & -3 & 0 \\ 5 & 11 & 5 \\ 1 & 2 & 3 \end{bmatrix}, \quad \mathbf{B} = \begin{bmatrix} 1 & 3 \\ -5 & 1 \\ 1 & 2 \end{bmatrix}, \mathbf{A} + \mathbf{B} = \begin{bmatrix} 1 & 3 \\ -5 & 1 \\ 1 & 2 \end{bmatrix} \tag{2.2.5}$$

Le nombre de colonnes de la matrice **A** est égal au nombre de lignes de la matrice **B**.

$$\mathbf{AB} = \begin{bmatrix} 2\times 1+(-3)\times(-5)+0\times 1 & 2\times 3+(-3)\times 1+0\times 2 \\ 5\times 1+11\times(-5)+5\times 1 & 5\times 3+11\times 1+5\times 2 \\ 1\times 1+2\times(-5)+3\times 1 & 1\times 3+2\times 1+3\times 2 \end{bmatrix} = \begin{bmatrix} 17 & 3 \\ -45 & 33 \\ -6 & 11 \end{bmatrix}. \tag{2.2.6}$$

Le produit **BA** n'est cependant pas possible.

Somme de matrices et multiplication d'une matrice par un scalaire.

La somme de matrices et multiplication d'une matrice par un scalaire se font coefficients par coefficients.

Avec les matrice $\mathbf{A}, \mathbf{B}$ de l'exemple précédent, et $\mathbf{C} = \begin{bmatrix} -2 & -7 & 3 \\ 5 & 10 & 5 \\ 12 & 9 & 3 \end{bmatrix}$, on a:

$$\mathbf{A}+\mathbf{C} = \begin{bmatrix} 2+(-2) & -3+(-7) & 0+3 \\ 5+5 & 11+10 & 5+5 \\ 1+12 & 2+9 & 3+3 \end{bmatrix} = \begin{bmatrix} 0 & -10 & 3 \\ 10 & 21 & 10 \\ 13 & 11 & 6 \end{bmatrix}, \text{ et pour tout } \lambda\in\mathbb{R}, \quad \lambda\mathbf{B} = \begin{bmatrix} \lambda & 3\lambda \\ -5\lambda & \lambda \\ \lambda & 2\lambda \end{bmatrix}. \tag{2.2.7}$$

NB: La somme de matrice n'est définie que pour des matrices de même taille.

Déterminant d'une matrice.

Soit $\mathbf{A} = (a_{ij})_{1\leq i\leq n, 1\leq j\leq n}$ une matrice carrée d'ordre n. Soit $\mathbf{A}_{i,j}$ la sous-matrice de **A** obtenue en supprimant la ligne i et la colonne j de **A**. On appelle **déterminant** (au développement suivant la ligne i) de **A** et on note $\det(\mathbf{A})$, le nombre

$$\det(\mathbf{A}) = \sum_{j=1}^{n} a_{ij}(-1)^{i+j}\det(\mathbf{A}_{i,j}), \tag{2.2.8}$$

avec le déterminant d'une matrice carrée de taille 2×2 donné par:

$$\det\left(\begin{bmatrix} a & b \\ c & d \end{bmatrix}\right) = ad-bc. \tag{2.2.9}$$

NB: Le développement suivant toutes les lignes donne le même résultat.

Le déterminant d'une matrice a une deuxième formulation dite de Leibniz[6] que nous n'introduisons pas dans ce document.

Inverse d'une matrice. Soit **A** une matrice carrée d'ordre n. **A** est **inversible** s'il existe une autre matrice notée $\mathbf{A}^{-1}$ telle que $\mathbf{A}\mathbf{A}^{-1} = \mathbf{A}^{-1}\mathbf{A} = \mathbf{I}_n$, où $\mathbf{I}_n$ est la matrice identité de taille $n\times n$.

Les matrices, leurs inverses et les opérations sur les matrices sont d'une importance capitale dans l'apprentissage automatique.

Vecteurs propres, valeurs propres d'une matrice.

Soient E un espace vectoriel et **A** une matrice. Un vecteur $\mathbf{v}\in E$ est dit **vecteur propre** de **A** si $\mathbf{v}\neq 0$ et il existe un scalaire λ tel que $\mathbf{A}\mathbf{v} = \lambda\mathbf{v}$. Dans ce cas, on dit que λ est la **valeur propre** associée au vecteur propre **v**.

[6] https://fr.wikipedia.org/wiki/Formule_de_Leibniz#D

Applications linéaires et changement de base d'espaces vectoriels.
Soient $(E, \mathcal{B}),\ (F, \mathcal{G})$ deux $\mathbb{K}$-espace vectoriels, chacun muni d'une base et $f : E \to F$ une application.
On dit que f est **linéaire** si les propriétés suivantes sont satisfaites:

1. Pour tous $\mathbf{u}, \mathbf{v} \in E$, $f(\mathbf{u} + \mathbf{v}) = f(\mathbf{u}) + f(\mathbf{v})$.
2. Pour tout $(\lambda, \mathbf{u}) \in \mathbb{K} \times E$, $f(\lambda \mathbf{u}) = \lambda f(\mathbf{u})$.

On suppose que $\mathcal{B} = \{e_1, e_2, \ldots, e_n\}$ et $\mathcal{G} = \{e'_1, e'_2, \ldots, e'_m\}$.
De manière équivalente, f est linéaire s'il existe une matrice $\mathbf{A}$ telle que pour tout $\mathbf{x} \in E, \quad f(x) = \mathbf{A}\mathbf{x}$.
Dans ce cas, la matrice $\mathbf{A}$ que l'on note $Mat_{\mathcal{B},\mathcal{G}}(f)$ est appelée matrice (représentative) de l'application linéaire f dans le couple de coordonnées $(\mathcal{B}, \mathcal{G})$.
La matrice $\mathbf{A}$ est unique et de taille $m \times n$ (notez la permutation *dimension de l'espace d'arrivée puis dimension de l'espace de départ dans la taille de la matrice*). De plus, la colonne j de la matrice $\mathbf{A}$ est constituée des coordonnées de $f(e_j)$ dans la base $\mathcal{G}$ de F. Lorsque $E = F$, l'application linéaire f est appelée **endomorphisme** de E et on écrit simplement $Mat_{\mathcal{B}}(f)$ au lieu de $Mat_{\mathcal{B},\mathcal{G}}(f)$.

Définition. Soient E un espace vectoriel de dimension finie et, $\mathcal{B}$ et $\mathcal{C}$, deux bases de E. On appelle **matrice de passage** de la base $\mathcal{B}$ à la base $\mathcal{C}$ la matrice de l'application identité

$$\begin{aligned} id_E : \quad & (E, \mathcal{C}) \to (E, \mathcal{B}) : \\ & x \mapsto x \quad . \end{aligned} \tag{2.2.10}$$

Cette matrice est notée $P_{\mathcal{B}}^{\mathcal{C}}$ et on a $P_{\mathcal{B}}^{\mathcal{C}} := Mat_{\mathcal{C},\mathcal{B}}(id_E)$.

Note: Si $\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$ est un vecteur de E exprimé dans la base $\mathcal{B}$, alors l'expression de $\mathbf{x}$ dans la base $\mathcal{C}$ est donnée par $\begin{bmatrix} x'_1 \\ x'_2 \\ \vdots \\ x'_n \end{bmatrix} = (P_{\mathcal{B}}^{\mathcal{C}})^{-1}\mathbf{x} = P_{\mathcal{C}}^{\mathcal{B}}\mathbf{x}$.

Exemple. Si $E = \mathbb{R}^3$ avec ses deux bases

$$\mathcal{B} = \left(\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \right) \text{ et } \mathcal{C} = \left(\begin{bmatrix} -1 \\ 2 \\ 3 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \\ 5 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \right), \tag{2.2.11}$$

on a $P_{\mathcal{B}}^{\mathcal{C}} = \begin{bmatrix} -1 & 0 & 0 \\ 2 & 1 & 0 \\ 3 & 5 & 1 \end{bmatrix}$ (c'est-à-dire qu'on exprime les vecteurs de $\mathcal{C}$ dans $\mathcal{B}$ pour former $P_{\mathcal{B}}^{\mathcal{C}}$).

Formule du changement de base pour une application linéaire.
Soient E une application linéaire et, $\mathcal{B}$ et $\mathcal{C}$, deux bases de E. Alors

$$Mat_{\mathcal{C}}(f) = P_{\mathcal{C}}^{\mathcal{B}} Mat_{\mathcal{B}}(f) P_{\mathcal{B}}^{\mathcal{C}}, \tag{2.2.12}$$

ou encore

$$Mat_{\mathcal{C}}(f) = (P_{\mathcal{B}}^{\mathcal{C}})^{-1} Mat_{\mathcal{B}}(f) P_{\mathcal{B}}^{\mathcal{C}}. \tag{2.2.13}$$

Diagonalisation et décomposition en valeurs singulières.
Diagonalisation. Soit **A** une matrice carrée à coefficients dans $\mathbb{K} = \mathbb{R}$ ou $\mathbb{C}$. On dit que **A** est **diagonalisable** s'il existe une matrice inversible **P** et une matrice diagonale **D** telles que $\mathbf{A} = \mathbf{PDP}^{-1}$. On dit aussi que **A** est similaire à **D**.
Important. Soient E un espace vectoriel de dimension finie et f un endomorphisme de E de matrice représentative (dans une base $\mathcal{B}$ de E) diagonalisable $\mathbf{A} = \mathbf{PDP}^{-1}$. On rappelle que les colonnes de **P** sont les vecteurs propres de **A**. Alors ces colonnes (dans leur ordre) constituent une base de E, et dans cette base, la matrice **A** est représentée par la matrice diagonale **D**. En d'autres termes, si $\mathcal{C}$ est la base des vecteurs propres de **A**, alors $Mat_{\mathcal{C}}(f) = \mathbf{D}$. Enfin, la matrice **D** est constituée des valeurs propres de **A** et le processus de calcul de **P** et **D** est appelé **diagonalisation**.
Décomposition en valeurs singulières.
Soit **M** une matrice de taille $m \times n$ et à coefficients dans $\mathbb{K} = \mathbb{R}$ ou $\mathbb{C}$. Alors **M** admet une factorisation de la forme $\mathbf{M} = \mathbf{U}$?$\mathbf{V}^*$, où $$

- **U** est une matrice unitaire (sur $\mathbb{K}$) de taille $m \times m$.
- $\mathbf{V}^*$ est l'adjoint (conjugué de la transposée) de **V**, matrice unitaire (sur $\mathbb{K}$) de taille $n \times n$
- ? est une matrice de taille $m \times n$ dont les coefficients diagonaux sont les valeurs singulières de **M**, i.e, les racines carrées des valeurs propres de $\mathbf{M}^*\mathbf{M}$ et tous les autres coefficients sont nuls.

Cette factorisation est appelée **la décomposition en valeurs singulières** de **M**. **Important.** Si la matrice **M** est de rang r, alors

- les r premières colonnes de **U** sont les vecteurs singuliers à gauche de **M**
- les r premières colonnes de **V** sont les vecteurs singuliers à droite de **M**
- les r premiers coefficients strictement positifs de la diagonale de ? sont les valeurs singulières de **M** et tous les autres coefficients sont nuls.

Produit scalaire et normes vectorielles. Soit V un espace vectoriel sur $\mathbb{R}$.
On appelle produit scalaire sur V toute application

$$\begin{aligned} \langle .,. \rangle : \quad & V \times V \to \mathbb{R} \\ & (\mathbf{u}, \mathbf{v}) \mapsto \langle \mathbf{u}, \mathbf{v} \rangle, \end{aligned} \tag{2.2.14}$$

telle que, $\forall(\lambda_1, \lambda_2, \mathbf{u}, \mathbf{v}, \mathbf{w}) \in \mathbb{R} \times \mathbb{R} \times V \times V \times V$,

- $\langle \mathbf{u}, \mathbf{v} \rangle = \langle \mathbf{v}, \mathbf{u} \rangle$(symétrie)
- 1. $\langle \lambda_1 \mathbf{u} + \lambda_2 \mathbf{v}, \mathbf{w} \rangle = \lambda_1 \langle \mathbf{u}, \mathbf{w} \rangle + \lambda_2 \langle \mathbf{v}, \mathbf{w} \rangle$ (linéarité à gauche)
  2. $\langle \mathbf{u}, \lambda_1 \mathbf{v} + \lambda_2 \mathbf{w} \rangle = \lambda_1 \langle \mathbf{u}, \mathbf{v} \rangle + \lambda_2 \langle \mathbf{u}, \mathbf{w} \rangle$ (linéarité à droite)
- $\langle \mathbf{u}, \mathbf{u} \rangle \geq 0$ (positive)
- $\langle \mathbf{u}, \mathbf{u} \rangle = 0 \implies \mathbf{u} = 0$ (définie)

$$\begin{aligned} \|.\| : \quad & V \to \mathbb{R}_+ \\ & \mathbf{v} \mapsto \|\mathbf{v}\| \end{aligned} \tag{2.2.15}$$

$V \; \forall \; (\lambda, \mathbf{u}, \mathbf{v}) \in \mathbb{R} \times V \times V$

- $\|\lambda \mathbf{u}\| = |\lambda| \times \|\mathbf{u}\|$
- $\|\mathbf{u} + \mathbf{v}\| \leq \|\mathbf{u}\| + \|\mathbf{u}\| + \|v\|$(inégalité triangulaire)

Remarque 2 Si $\langle .,. \rangle$ est un produit scalaire sur V, alors $\langle .,. \rangle$ induit une norme sur V. En effet,

$$\begin{aligned} \|.\|_{\langle .,. \rangle} : \quad & V \to \mathbb{R}_+ \\ & \mathbf{u} \mapsto \|\mathbf{u}\| = \sqrt{\langle \mathbf{u}, \mathbf{u} \rangle} \end{aligned} \tag{2.2.16}$$

Exemples de normes et produits scalaires.
Prenons $V = \mathbb{R}^n$.
• Les applications

$$\begin{aligned} \rho : \quad & V \times V \to \mathbb{R} \\ & (\mathbf{u}, \mathbf{v}) \mapsto \sum_{i=1}^{n} u_i v_i, \end{aligned} \tag{2.2.17}$$

et

$$\begin{aligned} \mu : \quad & V \to \mathbb{R}_+ \\ & \mathbf{u} \mapsto \sqrt{\sum_{i=1}^{n} u_i^2}, \end{aligned} \tag{2.2.18}$$

sont respectivement un produit scalaire et une norme sur V. Il faut remarquer que $\forall\ \mathbf{u} \in V, \quad \mu(\mathbf{u}) = \sqrt{\rho(\mathbf{u}, \mathbf{u})}$.

• Pour tout $p \in \mathbb{N}^*$, l'application

$$\begin{aligned} \mu_p : \quad & V \to \mathbb{R}_+ \\ & \mathbf{u} \mapsto \left(\sum_{i=1}^{n} |u_i|^p \right)^{\frac{1}{p}}, \end{aligned} \tag{2.2.19}$$

est une norme sur V appelée norme p.

Dans le cas $p = 2$, on retrouve la norme μ ci-dessus appelée norme euclidienne.

Remarque 3. Un espace vectoriel muni d'une norme est appelé **espace vectoriel normé**.

Notion de distance.

Soit E un ensemble non vide. Toute application $d : E \times E \to \mathbb{R}_+$ qui satisfait pour tout $x, y, z \in E$:

$$\begin{aligned} &\bullet \quad d(x, y) = d(y, x) \text{ (symétrie)} \\ &\bullet \quad d(x, y) = 0 \Longrightarrow x = y \text{ (séparation)} \\ &\bullet \quad d(x, y) \leq d(x, z) + d(z, y) \text{ (inégalité triangulaire)} \end{aligned} \tag{2.2.20}$$

Exemples de distances.

•

$$\begin{aligned} d : \quad & \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}_+ \\ & (\mathbf{u}, \mathbf{v}) \mapsto \left(\sum_{i=1}^{n} |u_i - v_i| \right). \end{aligned}$$

•

$$\begin{aligned} d : \quad & \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}_+ \\ & (\mathbf{u}, \mathbf{v}) \mapsto \left(\sum_{i=1}^{n} |u_i - v_i|^2 \right)^{\frac{1}{2}}. \end{aligned}$$

• C'est la généralisation de la distance euclidienne et de la distance de Manhattan

$$\begin{aligned} d_{Minkowski} : \quad & \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}_+ \\ & (\mathbf{u}, \mathbf{v}) \mapsto \left(\sum_{i=1}^{n} |u_i - v_i|^p \right)^{\frac{1}{p}}, p \geq 1. \end{aligned}$$

Espaces métriques.

Définition. Un **espace métrique** est un ensemble E muni d'une distance d; on écrit (E, d).

Remarque 4. Tout espace vectoriel normé est un espace métrique.

Suites dans un espace métrique.

Soit (E, d) un espace métrique. On appelle **suite** (d'éléments de E) et on note $(u_n)_{n\in I}$ ou $(u)_n$ une application:

$$\begin{aligned} u: \quad & I \to E \\ & n \mapsto u(n) := u_n \end{aligned} \tag{2.2.21}$$

où I est une partie infinie de $\mathbb{N}$. On dit que la suite $(u)_n$ converge vers $u^* \in E$ si pour tout $\epsilon > 0$ il existe $N \in \mathbb{N}$ tels que:

$$\forall\, n \in \mathbb{N}, \quad n > N \Longrightarrow d(u_n, u^*) < \epsilon \tag{2.2.22}$$

En d'autres termes, la suite $(u)_n$ converge vers $u^* \in E$ si pour tout $\epsilon > 0$, il existe un entier $N \in \mathbb{N}$ tel que pour tout $n > N$, u_n est contenu dans la boule $\mathcal{B}_\epsilon$ centrée en u^* et de rayon ϵ.

NB: La suite $(u)_n$ à valeurs dans E peut converger dans un ensemble autre que E.

Définition. La suite $(u)_n$ d'éléments de E est dite de Cauchy si pour tout $\epsilon > 0$, il existe $N \in \mathbb{N}$ tel que:

$$\forall\, n > m \in \mathbb{N}, \quad m > N \Longrightarrow d(u_n, u_m) < \epsilon. \tag{2.2.23}$$

Autrement dit, tous les termes u_n, u_m d'une suite de Cauchy se rapprochent de plus en plus lorsque n et m sont suffisamment grands.

Espaces métriques complets.

Définition. Un espace métrique (E, d) est dit **complet** si toute suite de Cauchy de E converge dans E.

Un espace métrique complet est appelé **espace de Banach**.

2.2.2 Calcul du gradient (dérivation).

Fonction réelle.

Définition.

Soit $f : J \to \mathbb{R}$ une fonction, avec J un intervalle ouvert de $\mathbb{R}$.

On dit que f est **dérivable** en $a \in J$ si la limite:

$$\lim_{h\to 0} \frac{f(a+h) - f(a)}{h} \text{ est finie.} \tag{2.2.24}$$

Si f est dérivable en a, la dérivée de f en a est notée $f'(a)$. La fonction dérivée de f est notée f' ou $\frac{\mathrm{d}f}{\mathrm{d}x}$ ou $\mathrm{d}f$.

Exemple de dérivées.

- **Fonctions polynomiales.**
  La dérivée de la fonction $f(x) = a_n x^n + a_{n-1}x^{n-1} + \cdots + a_1 x + a_0$, avec les a_i des constantes, est $f'(x) = n a_n x^{n-1} + (n-1)a_{n-1}x^{n-2} + \cdots + a_1$.

- **Fonction exponentielle de base :math:`e`**.
  La dérivée de la fonction $f(x) = \exp(x)$ est la fonction f elle-même, i.e, $\frac{\mathrm{d}\exp}{\mathrm{d}x}(x) = \exp(x)$.
- **Fonctions trigonométriques.**
  $\frac{\mathrm{d}\cos}{\mathrm{d}x}(x) = -\sin x$ et $\frac{\mathrm{d}\sin}{dx}(x) = \cos x$.
- **Fonction logarithme népérien**.
  $\frac{\mathrm{d}\ln}{\mathrm{d}x}(x) = \frac{1}{x}$.

Propriétés.
Soient $J \subseteq \mathbb{R}$ un intervalle ouvert, $u,\ v\ : J \to \mathbb{R}$ deux fonctions et $\lambda \in \mathbb{R}$. Alors on a les propriétés suivantes de la dérivée:

1. $(u+v)' = u' + v'$
2. $(uv)' = uv' + u'v$
3. $(\lambda u)' = \lambda u'$

Ces propriétés s'étendent aux fonctions vectorielles en dimension supérieure.
Fonctions vectorielles.
Soit $f : \mathcal{O} \to \mathbb{R}^p$ une fonction, avec $\mathcal{O}$ une partie ouverte de $\mathbb{R}^n, n, p \geq 1$.
On dit que f est **différentiable** (au sens de Fréchet) en $\mathbf{a} \in \mathcal{O}$, s'il existe une application linéaire continue $L : \mathbb{R}^n \to \mathbb{R}^p$ telle que pour tout $h \in \mathbb{R}^n$, on a

$$\lim_{h \to 0} \frac{f(\mathbf{a}+h) - f(\mathbf{a}) - L(h)}{\|h\|} = 0. \tag{2.2.25}$$

Si f est différentiable en tout point de $\mathcal{O}$, on dit que f est différentiable sur $\mathcal{O}$.

La différentielle de f est notée Df.
Dérivées partielles.
Soient $\mathbf{a} = \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{bmatrix} \in \mathcal{O} \subseteq \mathbb{R}^n$ et $f : \mathcal{O} \to \mathbb{R}^p$ une fonction.
On dit que f admet une dérivée partielle par rapport à la $j - me$ variable x_j si la limite:

$$\lim_{h \to 0} \frac{f(a_1, a_2, \ldots, a_j + h, \ldots, a_n) - f(\mathbf{a})}{h} \text{ est finie.} \tag{2.2.26}$$

La dérivée partielle par rapport à la variable x_j de f en $\mathbf{a}$ est notée $\frac{\partial f}{\partial x_j}(\mathbf{a})$.

Note. Si f est différentiable, alors f admet des dérivées partielles par rapport à toutes les variables.

Gradient et Matrice Jacobienne. Soit $f : \mathcal{O} \subseteq \mathbb{R}^n \to \mathbb{R}^p$ une fonction différentiable.
On suppose que les fonctions composantes de f sont $f_1, f_2, \ldots, f_p$.
Alors la matrice des dérivées partielles

$$\begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \cdots & \frac{\partial f_2}{\partial x_n} \\ \vdots & \vdots & & \vdots \\ \frac{\partial f_p}{\partial x_1} & \frac{\partial f_p}{\partial x_2} & \cdots & \frac{\partial f_p}{\partial x_n} \end{bmatrix} \tag{2.2.27}$$

est appelée la **matrice jacobienne** de f, notée $\mathbf{J}_f$ ou $\mathbf{J}(f)$.

Dans le cas $p = 1$, le vecteur $\begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \\ \vdots \\ \frac{\partial f}{\partial x_n} \end{bmatrix}$ est appelé **gradient** de f et noté ∇f ou $\mathbf{grad}(f)$.

Exemples du calcul de dérivées et de gradients sur :math:`mathbb{R}^n`.

- $f(\mathbf{x}) = \langle \mathbf{x}, \mathbf{x} \rangle = \mathbf{x}^T\mathbf{x}$. Le gradient de f est $\nabla f(\mathbf{x}) = 2\mathbf{x}$
- $f(\mathbf{x}) = \mathbf{A}\mathbf{x} + \mathbf{b}$, avec $\mathbf{A}$ une matrice et $\mathbf{b}$ un vecteur. On a $Df(\mathbf{x}) = \mathbf{A}$.

Dérivées de fonctions composées.

Il existe souvent des fonctions dont le gradient ne peut facilement être calculé en utilisant les formules précédentes. Pour trouver le gradient d'une telle fonction, on va réécrire la fonction comme étant une composition de fonctions dont le gradient est facile à calculer en utilisant les techniques que nous allons introduire. Dans cette partie nous allons présenter trois formules de dérivation de fonctions composées.

Composition de fonctions à une seule variable.
Soit $f, g, h : \mathbb{R} \to \mathbb{R}$, trois fonctions réelles telles que $f(x) = g(h(x))$.

$$\frac{\mathrm{d}f}{\mathrm{d}x} = \frac{\mathrm{d}g}{\mathrm{d}h}\frac{\mathrm{d}h}{\mathrm{d}x} \tag{2.2.28}$$

Formule de dérivée totale.

Soit $f : \mathbb{R}^{n+1} \to \mathbb{R}$ telle que $f = f(x, u_1(x), \ldots, u_n(x))$ avec $u_i : \mathbb{R} \to \mathbb{R}$ alors

$$\frac{\mathrm{d}f\,(x, u_1, \ldots, u_n)}{\mathrm{d}x} = \frac{\partial f}{\partial x} + \frac{\partial f}{\partial u_1}\frac{\mathrm{d}u_1}{\mathrm{d}x} + \frac{\partial f}{\partial u_2}\frac{\mathrm{d}u_2}{\mathrm{d}x} + \ldots + \frac{\partial f}{\partial u_n}\frac{\mathrm{d}u_n}{\mathrm{d}x} = \frac{\partial f}{\partial x} + \sum_{i=1}^{n}\frac{\partial f}{\partial u_i}\frac{\mathrm{d}u_i}{\mathrm{d}x}. \tag{2.2.29}$$

Formule générale de dérivées de fonctions composées.
Soit

$$\begin{array}{ll} f : \mathbb{R}^k \to \mathbb{R}^m & g : \mathbb{R}^n \to \mathbb{R}^k \\ \mathbf{x} \mapsto f(\mathbf{x}) & \mathbf{x} \mapsto g(\mathbf{x}) \end{array} \tag{2.2.30}$$

où $\mathbf{x} = (x_1, \ldots, x_n)$, $f(\mathbf{x}) = (f_1(\mathbf{x}), \ldots, f_m(\mathbf{x}))$ et $g(\mathbf{x}) = (g_1(\mathbf{x}), \ldots, g_k(\mathbf{x}))$.

Le gradient de $f(g(\mathbf{x}))$ est défini comme suit:

$$\frac{\partial}{\partial \mathbf{x}} \mathbf{f}(\mathbf{g}(\mathbf{x})) = \begin{bmatrix} \frac{\partial f_1}{\partial g_1} & \frac{\partial f_1}{\partial g_2} & \cdots & \frac{\partial f_1}{\partial g_k} \\ \frac{\partial f_2}{\partial g_1} & \frac{\partial f_2}{\partial g_2} & \cdots & \frac{\partial f_2}{\partial g_k} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial g_1} & \frac{\partial f_m}{\partial g_2} & \cdots & \frac{\partial f_m}{\partial g_k} \end{bmatrix} \begin{bmatrix} \frac{\partial g_1}{\partial x_1} & \frac{\partial g_1}{\partial x_2} & \cdots & \frac{\partial g_1}{\partial x_n} \\ \frac{\partial g_2}{\partial x_1} & \frac{\partial g_2}{\partial x_2} & \cdots & \frac{\partial g_2}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial g_k}{\partial x_1} & \frac{\partial g_k}{\partial x_2} & \cdots & \frac{\partial g_k}{\partial x_n} \end{bmatrix} \tag{2.2.31}$$

2.2.3 Probabilités

La théorie des probabilités constitue un outil fondamental dans l'apprentissage automatique. Les probabilités vont nous servir à modéliser une expérience aléatoire, c'est-à-dire un phénomène dont on ne peut pas prédire l'issue avec certitude, et pour lequel on décide que le dénouement sera le fait du hasard.

Définition.
Une probabilité est une application sur $\mathcal{P}(\Omega)$, l'ensemble des parties de Ω telle que:

- $0 \leq \mathbb{P}(A) \leq 1$, pour tout événement $A \subseteq \Omega$;
- $\mathbb{P}(A) = \sum_{\{\omega\} \in A} \mathbb{P}(\omega)$, pour tout événement A;
- $\mathbb{P}(\Omega) = \sum_{A_i} \mathbb{P}(A_i) = 1$, avec les $A_i \subseteq \Omega$ une partition de Ω.

Proposition. Soient A et B deux événements,

1. Si A et B sont incompatibles, $\mathbb{P}(A \cup B) = \mathbb{P}(A) + \mathbb{P}(B)$.
2. $\mathbb{P}\left(A^c\right) = 1 - \mathbb{P}(A)$, avec A^c le complémentaire de A.
3. $\mathbb{P}(\emptyset) = 0$.
4. $\mathbb{P}(A \cup B) = \mathbb{P}(A) + \mathbb{P}(B) - \mathbb{P}(A \cap B)$.

Preuve voir [@epardoux]
Ci-dessous une définition plus générale de probabilité, valable pour des espaces des événements possibles non dénombrables.
Définition. Soit A une expérience alátoire et Ω l'espace des événements possibles associés. Une probabilité sur Ω est une application définie sur l'ensemble des événements, qui vérifie:

::: {.center}

- **Axiome 1:** $0 \leq \mathbb{P}(A) \leq 1$, pour tout événement A;

- **Axiome 2:** Pour toute suite d'événements $(A_i)_{i\in\mathbf{N}}$, deux à deux incompatibles,

$$\mathbb{P}\left(\bigcup_{i\in\mathbf{N}} A_i\right) = \sum_{i\in\mathbf{N}} \mathbb{P}\left(A_i\right); \tag{2.2.32}$$

- **Axiome 3:** $\mathbb{P}(\Omega) = 1$. ::

NB : Les événements $(A_i)_{i\in\mathbf{N}}$ sont deux à deux incompatibles, si pour tous $i \neq j$, $A_i \cap A_j = \emptyset$.

Indépendance et conditionnement.

Motivation.

Quelle est la probablité d'avoir un cancer du poumon?

Information supplémentaire: vous fumez une vingtaine de cigarettes par jour. Cette information va changer la probabilité. L'outil qui permet cette mise à jour est la probabilité conditionnelle.

Définition.

Étant donnés deux événements A et B, avec $\mathbb{P}(A) > 0$, on appelle probabilité de B conditionnellement à A, ou sachant A, la probabilité notée $\mathbb{P}(B \mid A)$ définie par:

$$\mathbb{P}(B \mid A) = \frac{\mathbb{P}(A \cap B)}{\mathbb{P}(A)}. \tag{2.2.33}$$

L'équation *[prob_condit]* (page **??**) peut aussi s'écrire comme $\mathbb{P}(A \cap B) = \mathbb{P}(B \mid A)\mathbb{P}(A)$.

De plus, la probabilité conditionnelle sachant A, notée $\mathbb{P}(. \mid A)$ est une nouvelle probabilité et possède toutes les propriétés d'une probabilité.

Proposition. Formule des probabilités totales généralisée

Soit $(A_i)_{i\in I}$ (I un ensemble fini d'indices) une partition de Ω telle que $0 < \mathbb{P}(A_i) \leq 1 \quad \forall\, i \in I$. Pour tout événement B, on a

$$\mathbb{P}(B) = \sum_{i\in I} \mathbb{P}\left(B|A_i\right) \mathbb{P}\left(A_i\right). \tag{2.2.34}$$

La formule des probabilités totales permet de servir les étapes de l'expérience aléatoire dans l'ordre chronologique. **Proposition.** Formule de Bayes généralisée

Soit $(A_i)_{i\in I}$ une partition de Ω tel que $0 \leq \mathbb{P}(A_i) \leq 1, \forall\, i \in I$. Soit un événement B, tel que $\mathbb{P}(B) > 0$. Alors pour tout $i \in I$,

$$\mathbb{P}(A_i|B) = \frac{\mathbb{P}(B|A_i)\,\mathbb{P}(A_i)}{\sum_{i\in I}\mathbb{P}(B|A_i)\,\mathbb{P}(A_i)}. \tag{2.2.35}$$

Définition.
Deux événements A et B sont dits **indépendants** si

$$\mathbb{P}(A \cap B) = \mathbb{P}(A)\mathbb{P}(B). \tag{2.2.36}$$

S'ils sont de probabilité non nulle, alors

$$\mathbb{P}(B|A) = \mathbb{P}(B) \Leftrightarrow \mathbb{P}(A|B) = \mathbb{P}(A) \Leftrightarrow \mathbb{P}(A \cap B) = \mathbb{P}(A)\mathbb{P}(B). \tag{2.2.37}$$

Variables aléatoires.

Définition.
Une variable aléatoire (v.a) X est une fonction définie sur l'espace fondamental Ω, qui associe une valeur numérique à chaque résultat de l'expérience aléatoire étudiée. Ainsi, à chaque événement élémentaire ω, on associe un nombre $X(\omega)$.

Une variable qui ne prend qu'un nombre dénombrable de valeurs est dite **discrète** (par exemple le résultat d'une lancée d'une pièce de monnaie, ...), sinon, elle est dite **continue** (par exemple le prix d'un produit sur le marché au fil du temps, distance de freinage d'une voiture roulant à 100 km/h).

Variable aléatoire discrète

Définition.
L'espérance mathématique ou moyenne d'une v.a discrète X est le réel

$$\mathbb{E}[X] = \sum_{k=0}^{\infty} k\mathbb{P}[X = k]. \tag{2.2.38}$$

Pour toute fonction g,

$$\mathbb{E}[g(X)] = \sum_{k=0}^{\infty} g(k)\mathbb{P}[X = k]. \tag{2.2.39}$$

Définition.
La variance d'une v.a discrète X est le réel positif

$$Var[X] = \mathbb{E}\left[(X - \mathbb{E}[X])^2\right] = \sum_{k=0}^{\infty} (k - \mathbb{E}[X])^2 \, \mathbb{P}[X = k] = \mathbb{E}[X^2] - \mathbb{E}[X]^2 \tag{2.2.40}$$

et l'écart-type de X est la racine carrée de sa variance. $$

Exemple: Loi de Bernoulli
La loi de Bernoulli est fondamentale pour la modélisation des problèmes de classification binaire en apprentissage automatique. On étudie que les expériences aléatoires qui n'ont que deux issues possibles (succès ou échec). Une expérience aléatoire de ce type est appelée une épreuve de Bernoulli. Elle se conclut par un succès si l'évènement auquel on s'intéresse est réalisé ou un échec sinon. On associe à cette épreuve une variable aléatoire Y qui prend la valeur 1 si l'évènement est réalisé et la valeur 0 sinon. Cette v.a. ne prend donc que deux valeurs (0 et 1) et sa loi est donnée par :

$$\mathbb{P}[Y = 1] = p, \quad \mathbb{P}[Y = 0] = q = 1 - p. \qquad \text{Avec } p \in [0, 1]. \tag{2.2.41}$$

On dit alors que Y suit une loi de Bernoulli de paramètre p, notée $\mathcal{B}(p)$. La v.a. Y a pour espérance p et pour variance $p(1 - p)$. En effet,

$$\mathbb{E}[Y] = 0 \times (1 - p) + 1 \times p = p \tag{2.2.42}$$

et

$$\text{Var}(Y) = \mathbb{E}\left[Y^2\right] - \mathbb{E}[Y]^2 = \mathbb{E}[Y] - \mathbb{E}[Y]^2 = p(1 - p). \tag{2.2.43}$$

Schéma de Bernoulli :

- Chaque épreuve a deux issues : succès $[S]$ ou échec $[E]$.
- Pour chaque épreuve, la probabilité d'un succès est la même, notons $\mathbb{P}(S) = p$ et $\mathbb{P}(E) = q = 1 - p$.
- Les n épreuves sont **indépendantes** : la probabilité d'un succès ne varie pas, elle ne dépend pas des informations sur les résultats des autres épreuves.

Variable aléatoire continue

Contrairement aux v.a. discrètes, les v.a. continues sont utilisées pour mesurer des grandeurs "continues" (comme distance, masse, pression…). Une variable aléatoire continue est souvent définie par sa densité de probabilité ou simplement densité. Une densité f décrit la loi d'une v.a X en ce sens:

$$\forall\, a, b \in \mathbb{R}, \quad \mathbb{P}[a \leq X \leq b] = \int_a^b f(x)dx \tag{2.2.44}$$

et

$$\forall\, x \in \mathbb{R}, \quad F(x) = \mathbb{P}[X \leq x] = \int_{-\infty}^{x} f(t)dt \tag{2.2.45}$$

. On en déduit qu'une densité doit vérifier

$$\forall\, x \in \mathbb{R}, \quad f(x) \geq 0 \text{ et } \int_{\mathbb{R}} f(x)dx = 1 \tag{2.2.46}$$

.

Définition.
On appelle densité de probabilité toute fonction réelle positive, d'intégrale 1.

Définition.
L'espérance mathématiques de la v.a X est définie par

$$\mathbb{E}[X] = \int_{\mathbb{R}} x f(x) dx. \tag{2.2.47}$$

Exemple. La loi normale
C'est la loi de probabilité la plus importante. Son rôle est central dans de nombreux modèles probabilistes et en statistique. Elle possède des propriétés intéressantes qui la rendent agréable à utiliser. La densité d'une variable aléatoire suivant la loi normale de moyenne μ et d'écart-type σ ($\mathcal{N}(\mu, \sigma^2)$) est définie par

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right), \quad \forall\, x \in \mathbb{R}. \tag{2.2.48}$$

Quand $\mu = 0$ et $\sigma = 1$, on parle de loi normale centrée et réduite.

Loi des grands nombres

Considérons une suite $(X_n)_{n\geq 1}$ de v.a. indépendantes et de même loi. Supposons que ces v.a. ont une espérance, m et une variance, σ^2.

Théorème.

$$\mathbb{E}\left[\sum_{i=1}^{n} X_i\right] = nm \tag{2.2.49}$$

$$\text{Var}\left[\sum_{i=1}^{n} X_i\right] = n\sigma^2 \tag{2.2.50}$$

Définition.
La moyenne empirique des v.a. $X_1, \ldots, X_n$ est la v.a.

$$\bar{X_n} = \frac{X_1 + \cdots + X_n}{n}. \tag{2.2.51}$$

On sait d'ores et déjà que la moyenne empirique a pour espérance m et pour variance $\frac{\sigma^2}{n}$. Ainsi, plus n est grand, moins cette v.a. varie. A la limite, quand n tend vers l'infini, elle se concentre sur son espérance, m. C'est la loi des grands nombres.

Théorème. Convergence en Probabilité
Quand n est grand, $\bar{X_n}$ est proche de m avec une forte probabilité. Autrement dit,
$\forall\, \varepsilon \geq 0, \quad \lim\_n \to \infty \mathbb{P}(|\bar{X_n} - m| > \varepsilon) = 0.$

Théorème central limite Le Théorème central limite est très important en apprentissage automatique. Il est souvent utilisé pour la transformation des données surtout au traitement de données aberrantes.

Théorème.
Pour tous réels $a < b$, quand n tend vers $+\infty$,

$$\mathbb{P}\left(a \leq \frac{\bar{X}_n - m}{\sigma/\sqrt{n}} \leq b\right) \longrightarrow \int_a^b \frac{1}{\sqrt{2\pi}} e^{-x^2/2} \mathrm{d}x. \tag{2.2.52}$$

On dit que $\dfrac{\bar{X}_n - m}{\sigma/\sqrt{n}}$ converge en loi vers la loi normale $\mathcal{N}(0,1)$.

Intervalles de confiance

Soit X un caractère (ou variable) étudié sur une population, de moyenne m et de variance σ^2. On cherche ici à donner une estimation de la moyenne m de ce caractère, calculée à partir de valeurs observées sur un échantillon $(X_1, ..., X_n)$. La fonction de l'échantillon qui estimera un paramètre est appelée estimateur, son écart-type est appelé erreur standard et est noté SE. L'estimateur de la moyenne m est la moyenne empirique:

$$\frac{1}{n}\sum_{i=1}^{n} X_i \tag{2.2.53}$$

D'après les propriétés de la loi normale, avec un erreur $\alpha = 5\%$ quand n est grand on sait que

$$\mathbb{P}\left[m - 2\sigma/\sqrt{n} \leq \bar{X_n} \leq m + 2\sigma/\sqrt{n}\right] = 1 - \alpha = 0.954 \tag{2.2.54}$$

ou, de manière équivalente,

$$\mathbb{P}\left[\bar{X_n} - 2\sigma/\sqrt{n} \leq m \leq \bar{X_n} + 2\sigma/\sqrt{n}\right] = 1 - \alpha = 0.954 \tag{2.2.55}$$

Ce qui peut se traduire ainsi: quand on estime m par $\bar{X_n}$, l'erreur faite est inférieure à $2\sigma/\sqrt{n}$, pour $95,4\%$ des échantillons. Ou avec une probabilité de $95,4\%$, la moyenne inconnue m est dans l'intervalle $\left[\bar{X_n} - 2\sigma/\sqrt{n},\ \bar{X_n} + 2\sigma/\sqrt{n}\right]$.

Voir [@estatML] pour plus d'explication.

Définition.
On peut associer à chaque incertitude α, un intervalle appelé intervalle de confiance de niveau de confiance $1-\alpha$, qui contient la vraie moyenne m avec une probabilité égale à $1-\alpha$.

Définition.
Soit Z une v.a.. Le fractile supérieur d'ordre α de la loi de Z est le réel z qui vérifie

$$\mathbb{P}\left[Z \geq z\right] = \alpha \tag{2.2.56}$$

Le fractile inférieur d'ordre α de la loi Z est le réel z qui vérifie

$$\mathbb{P}\left[Z \leq z\right] = \alpha. \tag{2.2.57}$$

Quand l'écart-type théorique de la loi du caractère X étudié n'est pas connu, on l'éstime par l'écart-type empirique s_{n-1}. Comme on dispose d'un grand échantillon, l'erreur commise est petite. L'intervalle de confiance, de niveau de confiance $1-\alpha$ devient :

$$\left[\bar{x}_n - z_{\alpha/2}\frac{s_{n-1}}{\sqrt{n}},\ \bar{x}_n + z_{\alpha/2}\frac{s_{n-1}}{\sqrt{n}}\right] \tag{2.2.58}$$

où

$$s_{n-1}^2 = \frac{1}{n-1}\sum_{i=1}^{n}\left(x_i - \bar{x}\right)^2. \tag{2.2.59}$$

2.2.4 Estimations paramétriques

Soit $(\Omega, \mathcal{A}, \mathbf{P})$ un espace probabilisé et **x** une $v.a.$ de $(\Omega, \mathcal{A})$ dans $(E, \mathcal{E})$. La donnée d'un modèle statistique c'est la donnée d'une famille de probabilités sur $(E, \mathcal{E})$, $\{\mathbb{P}_\theta, \theta \in \Theta\}$. Le modèle étant donné, on suppose alors que la loi de **x** appartient au modèle $\{\mathbf{P}_\theta, \theta \in \Theta\}$. Par exemple dans le modèle de Bernoulli, $\mathbf{x} = (x_1, \ldots, x_n)$ où les x_i sont $i.i.d.$ (indépendantes et identiquement distribuées) de loi de Bernoulli de paramètre $\theta \in]0,1]$. $E = \{0,1\}^n$, $\mathcal{E} = \mathcal{P}(E)$, $\Theta =]0,1]$ et $P_\theta = \left((1-\theta)\delta_0 + \theta\delta_1\right)^{\otimes n}$.
Définition.
On dit que le modèle $\{\mathbb{P}_\theta, \theta \in \Theta\}$ est identifiable si l'application

$$\begin{array}{l}\Theta \to \{P_\theta, \theta \in \Theta\} \\ \theta \mapsto P_\theta\end{array} \tag{2.2.60}$$

est injective.

Définition.
Soit $g:\ \Theta \to \mathbb{R}^k$. On appelle estimateur de $g(\theta)$ au vu de l'observation x, toute application $T : \Omega \to \mathbb{R}^k$ de la forme $T = h(x)$ où $h : E \mapsto \mathbb{R}^k$ mesurable. Un estimateur ne doit pas dépendre de la quantité $g(\theta)$ que l'on cherche à estimer. On introduit les propriètés suivantes d'un estimateur.
Définition.

T est un estimateur sans biais de $g(\theta)$ si pour tout $\theta \in \Theta, \quad \mathbb{E}_\theta[T] = g(\theta)$.

Dans le cas contraire, on dit que l'estimateur T est biaisé et on appelle biais la quantité $\mathbb{E}_\theta[T] - g(\theta)$.

Généralement **x** est un vecteur $(x_1, \ldots, x_n)$ d'observations (n étant le nombre d'entre elles). Un exemple important est le cas où $x_1, \ldots, x_n$ forme un n-échantillon c'est à dire lorsque que $x_1, \ldots, x_n$ sont i.i.d. On peut alors regarder des propriétés asymptotiques de l'estimateur, c'est-à-dire en faisant tendre le nombre d'observations n vers $+\infty$. Dans ce cas, il est naturel de noter $T = T_n$ comme dépendant de n. On a alors la définition suivante :

Définition.

T_n est un estimateur consistant de $g(\theta)$ si pour tout $\theta \in \Theta$, T_n converge en probabilité vers $g(\theta)$ sous P_θ lorsque $n \to \infty$.

On définit le risque quadratique de l'estimateur dans le cas où $g(\theta) \in \mathbb{R}$.

Définition.

Soit T_n un estimateur de $g(\theta)$. Le risque quadratique de T_n est défini par

$$R(T_n, g(\theta)) = \mathbb{E}_\theta[(T_n - g(\theta))^2]. \tag{2.2.61}$$

Estimation par la méthode des moments

Considérons un échantillon $\mathbf{x} = (x_1, \ldots, x_n)$. Soit $f = (f_1, \ldots, f_k)$ une application de $\mathcal{X}$ dans $\mathbb{R}^k$ tel que le modèle $\{\mathbb{P}_\theta, \theta \in \Theta\}$ est identifiable si l'application Φ

$$\begin{aligned} \Phi : \ & \Theta \to \mathbb{R}^k \\ & \theta \mapsto \Phi(\theta) = \mathbb{E}_\theta[f(x)] \end{aligned} \tag{2.2.62}$$

est injective. On définit l'estimateur $\hat{\theta}_n$ comme la solution dans Θ (quand elle existe) de l'équation

$$\mathbb{E}_\theta[f(\mathbf{x})] \approx \frac{1}{n} \sum_{i=1}^{n} f(x_i). \tag{2.2.63}$$

Souvent, lorsque $\mathcal{X} \subset \mathbb{R}$, on prend $f_i(x) = x^i$ et Φ correspond donc au ième moment de la variable de X_i sous $\mathbb{P}_\theta$. Ce choix justifie le nom donné à la méthode. Voici quelques exemples d'estimateurs bâtis sur cette méthode.

Exemple. Loi uniforme

Ici $k = 1$, Q_θ est la loi uniforme sur $[0, \theta]$ avec $\theta > 0$. On a pour tout θ, $\mathbb{E}_\theta[X_1] = \dfrac{\theta}{2}$, on peut donc prendre par exemple $\Phi(\theta) = \dfrac{\theta}{2}$ et $f(x) = x$. L'estimateur obtenu par la méthode des moments est alors $\hat{\theta}_n = 2\bar{X}_n$. Cet estimateur est sans biais et constant.

Exemple. Loi normale

Ici $k = 2$, on prend $Q_\theta = \mathcal{N}(m, \sigma^2)$ avec $\theta = (m, \sigma^2) \in \mathbb{R} \times \mathbb{R}_+^*$. Pour tout θ, $\mathbb{E}_\theta[X_1] = m$ et $\mathbb{E}_\theta[X_1^2] = m^2 + \sigma^2$, on peut donc prendre par exemple, $f_1(x) = x$ et $f_2(x) = x^2$ ce qui donne $\Phi(m,\ \sigma^2) = (m, m^2 + \sigma^2)$. L'estimateur obtenu par la méthode des moments vérifie

$$\hat{m}_n = \bar{X}_n \text{ et } \hat{m}_n^2 + \hat{\sigma}_n^2 = \frac{1}{n}\sum_{i=1}^{n} X_i^2, \tag{2.2.64}$$

c'est-à-dire

$$\hat{\theta}_n = \left(\bar{X}_n,\ \frac{1}{n}\sum_{i=1}^{n}\left(X_i - \bar{X}_n\right)^2\right). \tag{2.2.65}$$

L'estimateur est consistant mais l'estimateur de la variance est biaisé.

Estimation par maximum de vraisemblance

Soit $\{E,\ \mathcal{E},\ \{P_\theta,\ \theta \in \Theta\}\}$ un modèle statistique, où $\Theta \subset \mathbb{R}^k$. On suppose qu'il existe une mesure σ-finie μ qui domine le modèle, c'est à dire que $\forall\ \theta \in \Theta$, P_θ admet une densité par rapport à μ.

Définition.

Soit **x** une observation. On appelle vraisemblance de **x** l'application

$$\begin{array}{l} \Theta \to \mathbb{R}_+ \\ \theta \mapsto \mathbb{P}(\theta,\ \mathbf{x}) \end{array} \tag{2.2.66}$$

On appelle estimateur du maximum de vraisemblance de θ, tout élément $\hat{\theta}$ de Θ maximisant la vraisemblance, c'est à dire vérifiant

$$\hat{\theta} = \arg\max_{\theta \in \Theta} \mathbf{P}(\theta,\ \mathbf{x}) \tag{2.2.67}$$

Considérons le cas typique où $\mathbf{x} = (x_1, \ldots, x_n)$, les x_i formant un n-échantillon de loi Q_{θ_0} où Q_{θ_0} est une loi sur $\mathcal{X}$ de paramètre inconnu $\theta_0 \in \Theta \subset \mathbb{R}^k$. On suppose en outre que pour tout $\theta \in \Theta, Q_\theta$ est absolument continue par rapport à une mesure ν sur $\mathcal{X}$. Dans ce cas, en notant

$$q(\theta, x) = \frac{dQ_\theta}{d\nu}(x) \tag{2.2.68}$$

et en prenant $\mu = \nu^{\otimes n}$ on a la vraisemblance qui s'écrit sous la forme

$$\mathbb{P}(\theta, \mathbf{x}) = \prod_{i=1}^{n} q\left(\theta, x_i\right) \tag{2.2.69}$$

et donc

$$\hat{\theta}_n = \arg\max_{\theta \in \Theta} \frac{1}{n}\sum_{i=1}^{n} \log\left[q\left(\theta, x_i\right)\right] \tag{2.2.70}$$

avec la convention $\log(0) = -\infty$.

Exemple. Modèle de Bernoulli

Soit $Q_{\theta_0} = \mathcal{B}(\theta)$ avec $\theta \in [0,1] = \Theta$. Pour tout $\theta \in]0,1[$ et $x \in \{0,1\}$

$$q(\theta, x) = \theta^x (1-\theta)^{1-x} = (1-\theta) \exp\left[x \log\left(\frac{\theta}{1-\theta}\right)\right] \tag{2.2.71}$$

et donc l'estimateur du maximum de vraisemblance doit maximiser dans l'intervalle [0,1].

$$\log\left(\theta^{S_n}(1-\theta)^{n-S_n}\right) = S_n \log\left(\frac{\theta}{1-\theta}\right) + n \log(1-\theta) \tag{2.2.72}$$

avec $S_n = \sum_i x_i$ ce qui conduit à $\hat{\theta}_n = \bar{\mathbf{x}}$ en résolvant l'équation $\nabla \log(q(\theta, x)) = 0$.

3 | Apprentissage Supervisé

Dans ce chapitre, nous allons explorer l'apprentissage supervisé. Ce type d'apprentissage, aussi connu sous le nom d'apprentissage avec tutelle (maître), permet de déterminer la relation qui existe entre une variable explicative **X** et une variable à expliquer (étiquette) **y**. En d'autres termes, l'apprentissage supervisé est le processus permettant à un modèle d'apprendre, en lui fournissant des données d'entrée ainsi que des données de sortie. Cette paire d'entrée/sortie est généralement appelée «données étiquetées». Dans un cadre illustratif, pensez à un enseignant qui, connaissant la bonne réponse à une question, évaluera un élève en fonction de l'exactitude de sa réponse à cette question. Pour plus de clarification, comparons l'approche de l'apprentissage automatique à la programmation traditionnelle.

- Dans la programmation traditionnelle, comme illustré dans la figure *1.1* (page **??**), nous avons une fonction f connue qui reçoit la donnée en entrée **x** et renvoie la réponse correspondante **y** en sortie. Par exemple, nous pouvons penser à écrire une fonction f qui calcule le carré d'un nombre; si nous donnons en entrée le nombre 2, notre programme va nous renvoyer la valeur $f(2) = 2^2 = 4 = y$.

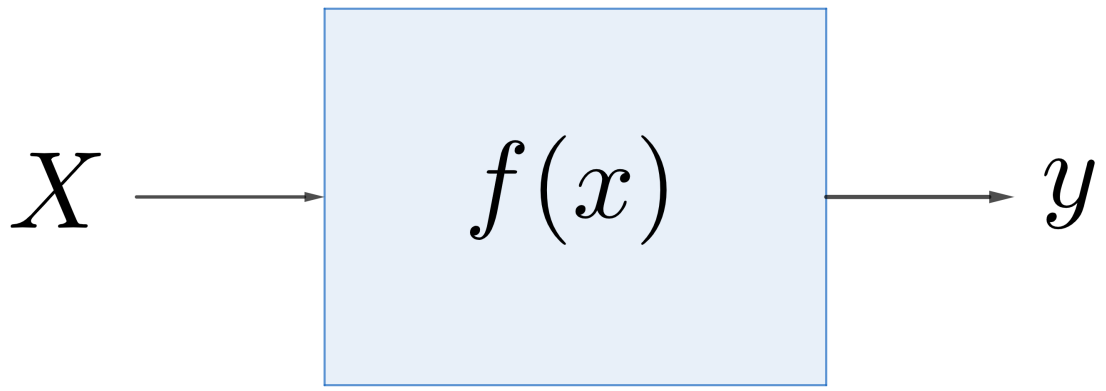


Fig. 3.1: L'approche traditionnelle

- L'approche de la programmation utilisée dans l'apprentissage automatique est tout à fait différente de la précédente. Dans cette dernière, nous ne connaissons pas la fonction f et nous voulons donc l'approximer

par une fonction $\hat{f}$ en utilisant les données à notre disposition. Cette approche est donc divisée en deux phases. La première est la phase où nous entraînons notre fonction $\hat{f}$ (figure *1.2* (page **??**)). Si nous revenons à notre exemple précédent, cette étape pourra consister à présenter à la fonction $\hat{f}$, plusieurs couples de nombres et leurs carrés $\{(2,4),(3,9),(4,16),\cdots\}$. L'objectif ici est de trouver un moyen d'estimer la fonction "carrée" en observant uniquement les données à notre disposition.

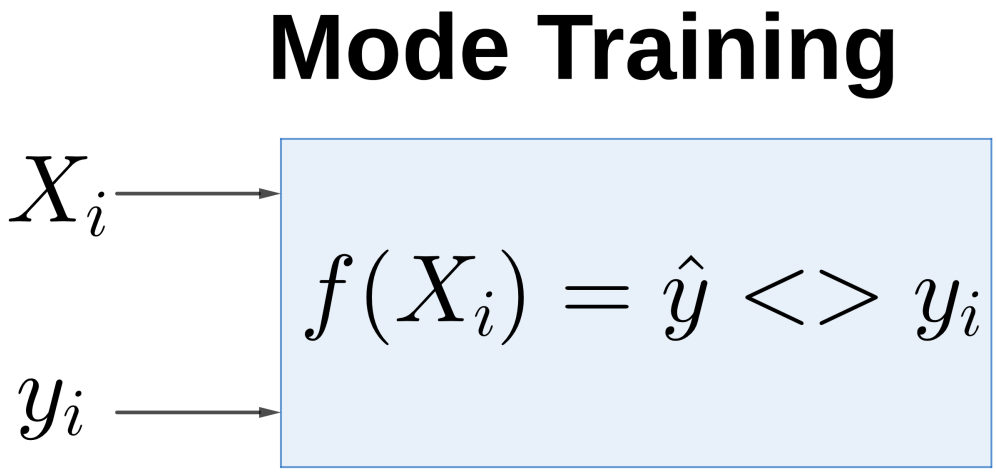


Fig. 3.2: L'approche apprentissage automatique

La dernière étape consiste à fournir un nouveau nombre à notre fonction $f^{\star}$, obtenue après l'étape 1, afin qu'elle prédise (*approximativement*) le carré de ce nombre.

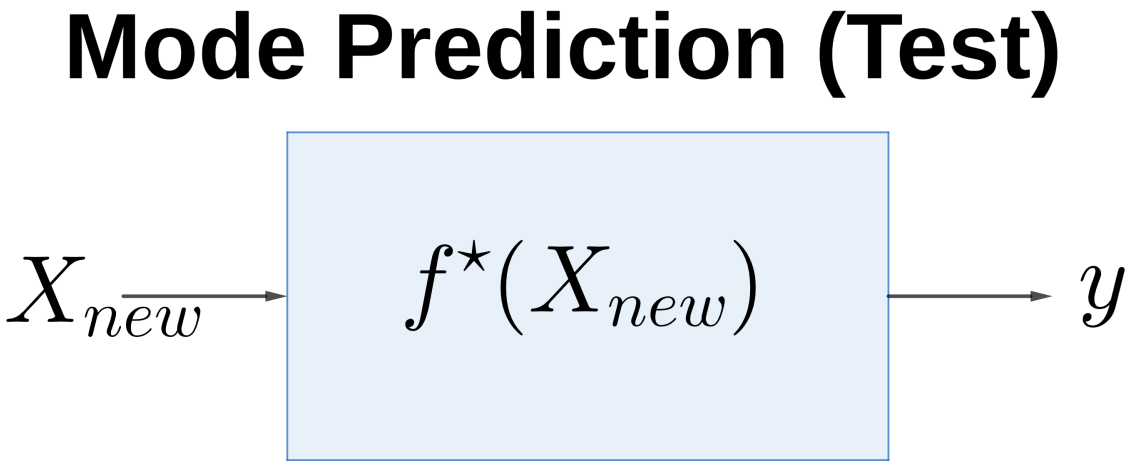


Fig. 3.3: L'approche apprentissage automatique

Dans la suite du cours, nous reviendrons beaucoup plus en détails sur les étapes ci-dessus présentées.

L'apprentissage supervisé est souvent utilisé pour deux types de problèmes: les problèmes de régression et les problèmes de classification.

3.1 Problèmes de Régression

Dans l'apprentissage supervisé, on parle de problèmes de régression lorsque la variable à expliquer **y** est continue. Par exemple lorsqu'on veut **prédire le prix** d'une bouteille de vin sur la base de **certaines variables** (le pays de fabrication, qualité, le taux d'alcool, etc.). Il s'agit bel et bien d'un problème de régression.

3.1.1 La Régression Linéaire

La régression linéaire est un problème de régression pour lequel le modèle ou la fonction dépend linéairement de ses paramètres [@reg_lin]. Les différents types de régression linéaire que nous connaissons sont la régression linéaire affine, la régression linéaire polynomiale et la régression linéaire à fonctions de base radiales. Dans ce document, nous allons nous focaliser sur deux types fondamentaux de régression linéaire: la régression linéaire affine et la régression linéaire polynomiale.

La régression linéaire affine

Une régression linéaire de paramètre $\boldsymbol{\theta}$ est dite affine si pour tout $\mathbf{x} \in \mathbb{R}^d$.

$$f_{\boldsymbol{\theta}}(\mathbf{x}) = \boldsymbol{\theta}_0 + \boldsymbol{\theta}_1^T \mathbf{x} = \begin{bmatrix} \boldsymbol{\theta}_0 & \boldsymbol{\theta}_1^T \end{bmatrix} \begin{bmatrix} 1 \\ \mathbf{x} \end{bmatrix} \tag{3.1.1}$$

avec $\boldsymbol{\theta}_0 \in \mathbb{R}$ et $\boldsymbol{\theta}_1 \in \mathbb{R}^d$. Le terme $[1, \mathbf{x}]$ est appelé attribut du modèle et il sera noté par $\phi(\mathbf{x})$.

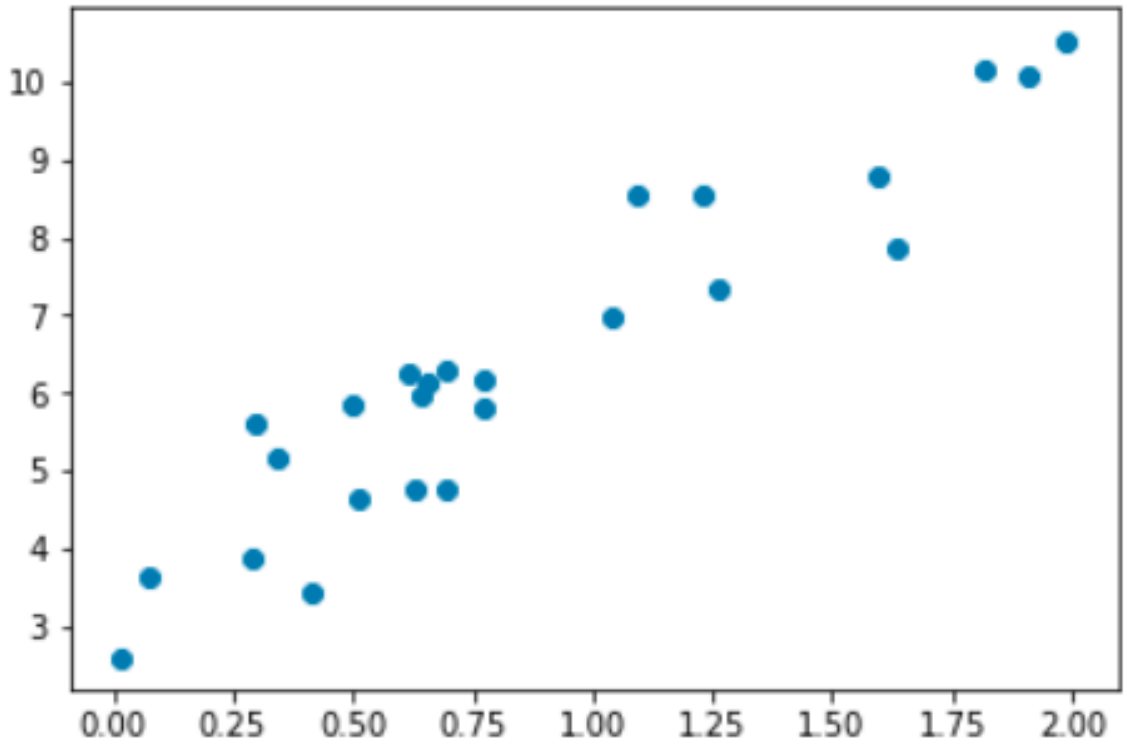


Fig. 3.1.1: Représentation graphique d'un exemple de données d'entraînement

Les jeux de données représentés dans la figure *1.4* (page **??**) forment un ensemble d'entraînement où la régression linéaire affine sera la plus appropriée.
Pour déterminer les meilleurs paramètres de la régression linéaire affine deux différentes méthodes sont utilisées à savoir: la méthode explicite et la méthode approximative.

- **La méthode explicite**

  Dans le cas de la régression linéaire affine, la méthode explicite peut-être utilisée par le biais de l'estimation du maximum de vraisemblance qui interpelle la notion de probabilité conditionnelle.

  Pour être plus concret, nous allons considérer l'expression suivante:
  $y_i = f_{\boldsymbol{\theta}}(\mathbf{x}_i) + \varepsilon \quad$ avec $\varepsilon \sim N(0, \sigma^2)$.

  Dans cette expression, nous supposons que f est la fonction que nous allons estimer à partir de son paramètre $\boldsymbol{\theta}$ et qui nous permettra de faire nos prédictions pour chaque élément donné à partir du domaine d'entraînement. Nous noterons par $\hat{f}$ comme étant la fonction estimée de f.

  Pour une suite de points $(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), ..., (\mathbf{x}_n, y_n)$ représentant le domaine d'entraînement nous supposons que les y_i suivent chacun une loi normale et qu'ils sont aussi indépendants et identiquement distribués (i.i.d).

  Alors, nous avons $\mathbf{x} = \{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n\} \in \mathbb{R}^{n \times d}$ et $\mathbf{y} = \{y_1, y_2, ..., y_n\} \in \mathbb{R}^n$.

Déterminons le paramètre $\boldsymbol{\theta}^*$ qui maximise la vraisemblance.

$$\begin{aligned}\mathbb{P}(y_1, y_2, .., y_n|\mathbf{x}_1, \mathbf{x}_2, \cdots \mathbf{x}_n; \boldsymbol{\theta}) &= \mathbb{P}(\mathbf{y}|\mathbf{x}; \boldsymbol{\theta}) \\ &= \prod_i^n \mathbb{P}(y_i|\mathbf{x}_i, \boldsymbol{\theta}) \text{ avec } y_i \sim N(\boldsymbol{\theta}^T\mathbf{x}_i; \sigma^2)\end{aligned} \tag{3.1.2}$$

Dans ce cas nous avons:

$$\mathbb{P}(y_i|\mathbf{x}_i; \boldsymbol{\theta}) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(yi - \boldsymbol{\theta}^T\mathbf{x_i})^2}{2\sigma^2}\right). \tag{3.1.3}$$

Nous savons que, la fonction logarithme est une fonction strictement croissante, ce qui implique que le paramètre $\boldsymbol{\theta}^*$ qui maximise la vraisemblance maximise aussi le logarithme-vraisemblance. Ainsi, en appliquant le logarithme de la vraisemblance, nous avons:

$$\log \mathbb{P}(\mathbf{y}|\mathbf{x}; \boldsymbol{\theta}) = \sum_{i=1}^{n} \log \mathbb{P}(y_i|\mathbf{x}_i; \boldsymbol{\theta}). \tag{3.1.4}$$

Pour chaque $i \in \{1, 2, ..., n\}$, $\log \mathbb{P}(y_i|\mathbf{x}_i; \boldsymbol{\theta}) = \log\left(\frac{1}{\boldsymbol{\sigma}\sqrt{2\pi}}\right) - \frac{\left(y_i - \boldsymbol{\theta}^T\mathbf{x}_i\right)^2}{2\sigma^2}$

$$\begin{aligned}\implies \log \mathbb{P}(\mathbf{y}|\mathbf{x}; \boldsymbol{\theta}) &= -\frac{1}{2\sigma^2}\sum_{i=1}^{n}(y_i - \boldsymbol{\theta}^T\mathbf{x}_i)^2 + c^{ste} \\ &= -\frac{1}{2\sigma^2}(\mathbf{y} - \boldsymbol{\theta}\mathbf{x})^T(\mathbf{y} - \boldsymbol{\theta}\mathbf{x}) + c^{ste}\end{aligned} \tag{3.1.5}$$

Ainsi, la dérivée partielle du logarithme de la vraisemblance par rapport à $\boldsymbol{\theta}$ est donnée par:

$$\begin{aligned}\frac{\partial \log \mathbb{P}(\mathbf{y}|\mathbf{x}, \boldsymbol{\theta})}{\partial \boldsymbol{\theta}} &= \frac{\partial}{\partial \boldsymbol{\theta}}\left(-\frac{1}{2\sigma^2}(\mathbf{y} - \mathbf{x}\boldsymbol{\theta})^T(\mathbf{y} - \mathbf{x}\boldsymbol{\theta}) + c^{ste}\right) \\ &= \frac{1}{\sigma^2}\mathbf{x}^T(\mathbf{x}\boldsymbol{\theta} - \mathbf{y}).\end{aligned} \tag{3.1.6}$$

Alors, résoudre l'équation:

$$\frac{\partial \log \mathbb{P}(\mathbf{y}|\mathbf{x}, \boldsymbol{\theta})}{\partial \boldsymbol{\theta}} = 0 \tag{3.1.7}$$

nous permettra de trouver la valeur de $\boldsymbol{\theta}^*$.

$$\begin{aligned}\frac{\partial \log \mathbb{P}(\mathbf{y}|\mathbf{x}, \boldsymbol{\theta})}{\partial \boldsymbol{\theta}} &= 0, \\ \mathbf{x}^T(\mathbf{x}\boldsymbol{\theta} - \mathbf{y}) &= 0, \\ \mathbf{x}^T\mathbf{x}\boldsymbol{\theta} &= \mathbf{x}^T\mathbf{y}.\end{aligned} \tag{3.1.8}$$

En supposant que la matrice $\mathbf{x}^T\mathbf{x}$ est inversible nous avons :

$$\boldsymbol{\theta}^* = (\mathbf{x}^T\mathbf{x})^{-1}\mathbf{x}^T\mathbf{y}. \tag{3.1.9}$$

Alors, vu que nous avons déterminé le paramètre $\boldsymbol{\theta}^*$, la fonction $\hat{\boldsymbol{f}}$ associée au paramètre $\boldsymbol{\theta}^*$, souvent appelée “hypothèse” ou “modèle” s’écrit comme

$$\hat{\boldsymbol{f}}(\mathbf{x}) = \boldsymbol{\theta}^* \mathbf{x}. \tag{3.1.10}$$

[droite]

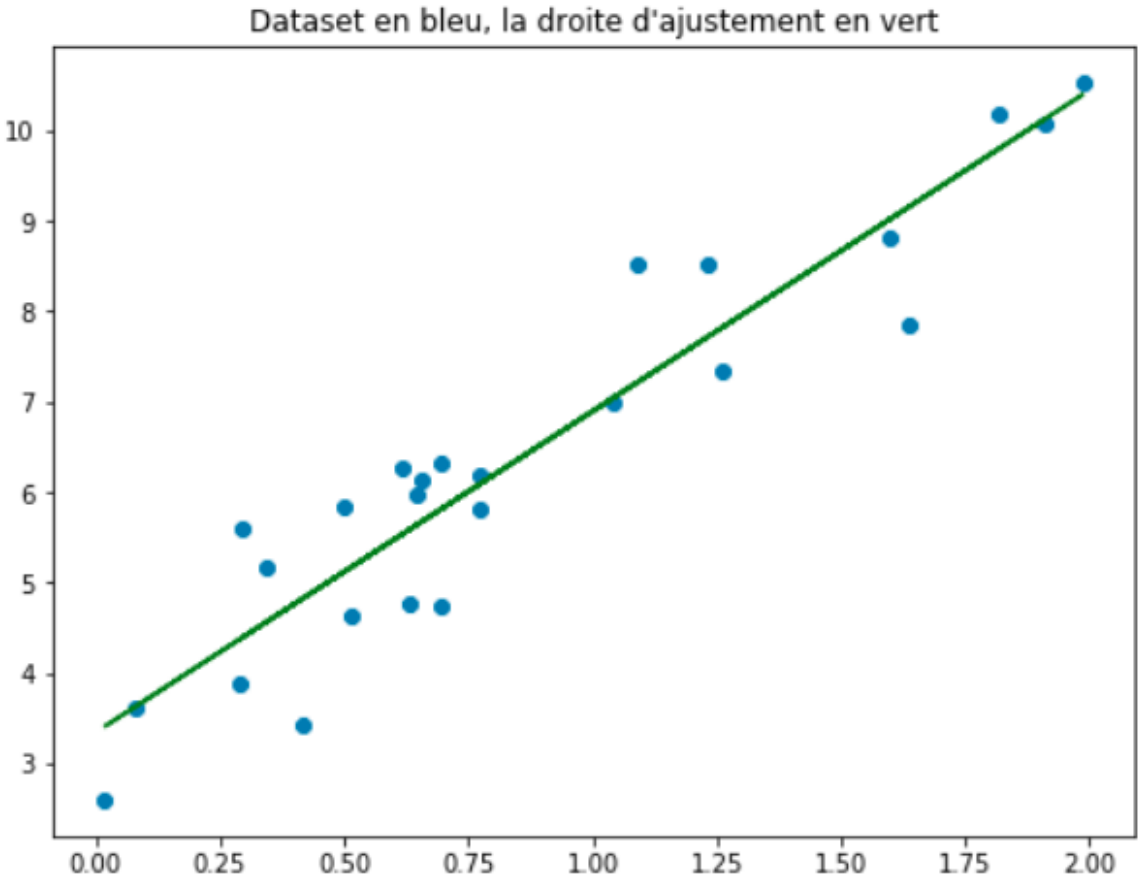


Fig. 3.1.2: Représentation graphique de la fonction $\hat{f}$ définie dans l’ensemble X à valeur dans $\mathbb{R}$.

La méthode explicite nous permet d’obtenir la solution exacte de l’équation *[equat]* (page **??**). Tout de même, trouver cette solution exacte est souvent très compliquée dans le cas où l’étude se fait sur un grand ensemble de jeux de données (la complexité pour trouver l’inverse dans l’équation *[star]* (page **??**) est $\mathcal{O}(n^3)$). Pour cela, dans ce qui suit, nous allons présenter des méthodes alternatives qui nous permettront de donner une valeur approchée à la solution exacte.

- **Méthodes approximatives**

Dans cette partie, nous allons utiliser une méthode itérative pour estimer la valeur des paramètres de l’équation suivante:

$$\mathbf{y} = \boldsymbol{\theta}\mathbf{x} \tag{3.1.11}$$

,

où $\boldsymbol{\theta} \in \mathbb{R}^{d+1}$ est le vecteur de paramètres à estimer; $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n\} \in \mathbb{R}^{n\times(d+1)}$ et $\mathbf{y} = \{y_1, y_2, ..., y_n\} \in \mathbb{R}^n$ les données. $$

La fonction de perte

La fonction de perte mesure la différence entre la valeur observée et la valeur estimée. En apprentissage automatique, l'objectif est d'optimiser la fonction de perte. Il existe différentes fonctions de perte selon le critère (ou métrique permettant d'évaluer la performance du modèle) adopté(e). Dans cette partie, nous allons utiliser l'erreur quadratique moyenne (appelé Mean Square Error (MSE) en anglais) pour définir notre fonction de perte.

L'erreur quadratique moyenne entre le $\mathbf{y}$ observé et le $\hat{\curvearrowright}$ prédit est donnée par:

$$\text{MSE}(\mathbf{y}, \hat{\boldsymbol{y}}) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2, \tag{3.1.12}$$

où n est la dimension des vecteurs $\mathbf{y}$ et $\hat{\boldsymbol{y}}$.

Dans le cas de la régression linéaire, cette fonction peut être réécrite comme étant une fonction E de $\boldsymbol{\theta}$.

$$E\left(\boldsymbol{\theta}\right) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \boldsymbol{\theta}^T\mathbf{x}_i)^2. \tag{3.1.13}$$

Par conséquent, le paramètre $\boldsymbol{\theta}$ qui correspond à la meilleure ligne d'ajustement sera tout simplement la valeur qui minimise la fonction de perte E. Pour cela, nous allons introduire une méthode la plus souvent utilisée pour minimiser une fonction (éventuellement convexe) dans l'apprentissage automatique à savoir la descente de gradient.

[f_convexe]

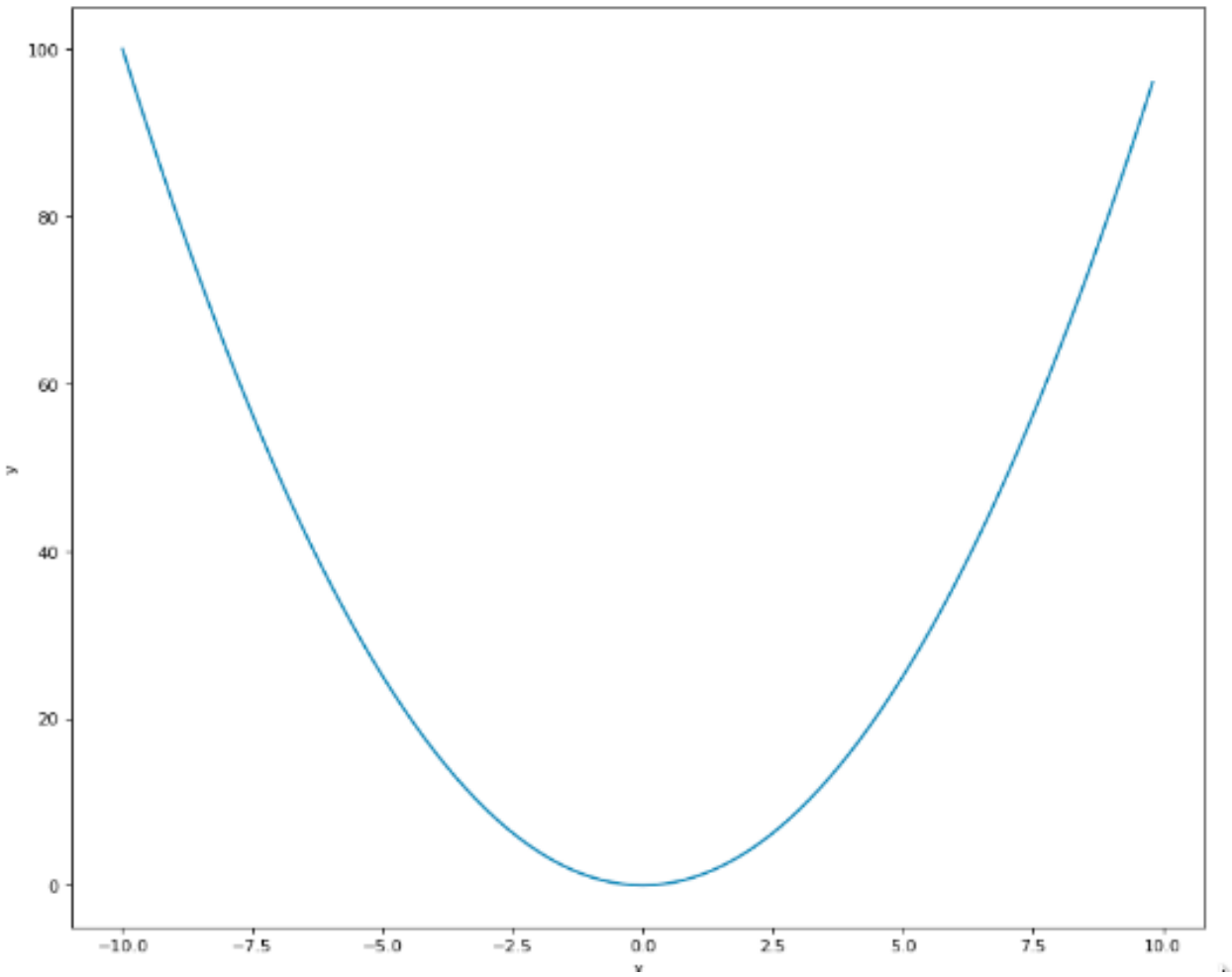


Fig. 3.1.3: Représentation graphique d'une fonction convexe

Descente de gradient {GD}

La descente de gradient est une procédure itérative d'optimisation dans laquelle, à chaque étape, on améliore la solution en essayant de minimiser la fonction de perte considérée [@desc_grad]. Elle est appliquée lorsque l'on cherche le minimum d'une fonction dont on connaît l'expression analytique, qui est dérivable, mais dont le calcul direct du minimum est difficile.

Pour entamer cette procédure, nous allons commencer par initialiser le paramètre $\boldsymbol{\theta}$. Ensuite, nous calculons la dérivée partielle de la fonction E par rapport au paramètre $\boldsymbol{\theta}$ donnée par:

$$\frac{\partial E}{\partial \boldsymbol{\theta}} = -\frac{2}{n}\sum_{i=1}^{n} \mathbf{x}_i(y_i - \boldsymbol{\theta}^T \mathbf{x}_i). \tag{3.1.14}$$

Pour trouver les meilleurs paramètres, nous allons répéter le processus ci-dessous jusqu'à ce que la fonction de perte soit très proche ou égale à 0.

$$\boldsymbol{\theta} = \boldsymbol{\theta} - \gamma \cdot \frac{\partial E}{\partial \boldsymbol{\theta}}, \tag{3.1.15}$$

La valeur de $\boldsymbol{\theta}$ trouvée après convergence est la valeur optimale que nous noterons par $\boldsymbol{\theta}^*$.

Alors, concernant l'exemple de la figure *1.4* (page **??**), notre hypothèse ou modèle sera représenté par une droite d'ajustement de la même forme que celle en couleur verte sur la figure *1.5* (page **??**). Cette droite est d'équation:

$\mathbf{y} = \boldsymbol{\theta}^* \mathbf{x}$.

Implementation

class **LinearRégression**(): def **__init__** (self): pass def **fonction_perte**(self, y_vrai, y_prédit): définit une fonction de perte et retourne sa valeur def **algorithme**(self, **x**, **y**, taux_apprentissage, nombre_itération): initialiser les paramètres $\boldsymbol{\theta}_0$ et $\boldsymbol{\theta}_1$

for i in range(nombre_itération): **prédiction**(x), calcule la perte au moyen de **fonction_perte**, mise à jour des paramètres $\boldsymbol{\theta}_0$ et $\boldsymbol{\theta}_1$, return $\boldsymbol{\theta}_0, \boldsymbol{\theta}_1$, def **prédiction**(self, **x**): y_prédit $= \boldsymbol{\theta}_0^T \mathbf{x} + \boldsymbol{\theta}_1$, return y_prédit

Un exemple d'implementation de régression linéaire est disponible ici[7]

La régression linéaire polynomiale

La régression linéaire de paramètre $\boldsymbol{\theta}$ est dite polynomiale si pour tout $\mathbf{x} \in \mathbb{R}^d$,

$$\begin{aligned} \mathrm{f}_{\boldsymbol{\theta}}(\mathbf{x}) &= \boldsymbol{\theta}_0 + \boldsymbol{\theta}_1 \mathbf{x}^1 + ... + \boldsymbol{\theta}_m \mathbf{x}^m \\ &= [\boldsymbol{\theta}_0, \boldsymbol{\theta}_1, ..., \boldsymbol{\theta}_m] \begin{bmatrix} 1 \\ \mathbf{x}^1 \\ \vdots \\ \mathbf{x}^m \end{bmatrix} \\ &= \sum_{i=0}^{m} \boldsymbol{\theta}_i \mathbf{x}^i, \end{aligned} \tag{3.1.16}$$

avec comme attribut le vecteur $\phi(\mathbf{x}) = \left[1, \mathbf{x}^1, ..., \mathbf{x}^m\right]^T$. Ainsi, deux méthodes existent pour déterminer le meilleur paramètre $\boldsymbol{\theta}^*$.

1. **Estimation par la méthode du maximum de vraisemblance (appelée MLE: Maximum Likelihood Estimation)**: Suivant de manière analogique de la détermination du paramètre $\boldsymbol{\theta}^*$ sur la partie précédente, la meilleure valeur du paramètre $\boldsymbol{\theta}^*$ est déterminée par $\boldsymbol{\theta}^* = (\mathbf{x}^T \mathbf{x})^{-1} \mathbf{x}^T \mathbf{y}$.
2. **Estimation par la méthode d'un posteriori maximal (appelée MAP: Maximum A Posteriori)**: La méthode consiste à trouver la valeur $\boldsymbol{\theta}^*_{\text{MAP}}$ qui maximise le produit entre la vraisemblance et la distribution à priori des paramètres $\boldsymbol{\theta}$ comme l'indique l'équation *[naivebayes]* (page **??**). Cette méthode d'estimation apparaît généralement dans un cadre bayésien. Tout comme la méthode du maximum de vraisemblance, elle peut être utilisée afin d'estimer un certain nombre de paramètres inconnus, comme les paramètres d'une densité de probabilité, reliés à un échantillon donné. La seule différence avec la méthode de maximum de vraisemblance est sa possibilité de prendre en compte un à priori non uniforme sur les paramètres à estimer. Ainsi, nous pouvons dire que l'estimateur au maximum de vraisemblance est l'estimateur MAP pour une distribution à priori uniforme. Par le théorème de Bayes, nous pouvons obtenir le postérieur comme un produit de vraisemblance avec :

$$\begin{aligned} \mathbb{P}\left(\boldsymbol{\theta} | \mathbf{y}; \mathbf{x}\right) &= \frac{\mathbb{P}\left(\mathbf{y}; \mathbf{x} | \boldsymbol{\theta}\right) \mathbb{P}\left(\boldsymbol{\theta}\right)}{\mathbb{P}\left(\mathbf{y}; \mathbf{x}\right)} \\ &\propto \mathbb{P}\left(\mathbf{y}; \mathbf{x} | \boldsymbol{\theta}\right) \mathbb{P}\left(\boldsymbol{\theta}\right). \end{aligned}$$

[7] https://colab.research.google.com/drive/1Ad94wJI2hch6BxpRV9P-SZcc3UfI2zwY#scrollTo=BPyCNZ9Z6tMg

Avec $\mathbf{Y}|\boldsymbol{\theta} \sim \mathcal{N}(\boldsymbol{\theta}^T\mathbf{x}, \sigma^2)$ et $\boldsymbol{\theta} \sim \mathcal{N}(\mathbf{0}, \lambda^2\mathbf{I})$ où $\mathbf{I}$ représente la matrice identité dont la dimension est la longueur du vecteur $\boldsymbol{\theta}$. Ainsi, nous pouvons écrire la vraisemblance comme:

$$\mathbb{P}(\boldsymbol{\theta}|\mathbf{y};\mathbf{x}) = \frac{1}{\sigma\sqrt{2\pi}}\exp\left(-\frac{(y-\boldsymbol{\theta}^T\mathbf{x})^2}{2\sigma^2}\right)\frac{1}{\lambda\sqrt{2\pi}}\exp\left(-\frac{\boldsymbol{\theta}^2}{2\lambda^2}\right) \tag{3.1.17}$$

En utilisant la fonction logarithme, nous avons

$$\begin{aligned}\log\mathbb{P}(\boldsymbol{\theta}|\mathbf{x},\mathbf{y}) &= \log\left(\frac{1}{\sigma\sqrt{2\pi}}\exp\left(-\frac{(\mathbf{y}-\boldsymbol{\theta}^T\mathbf{x})^2}{2\sigma^2}\right)\frac{1}{\lambda\sqrt{2\pi}}\exp\left(-\frac{\boldsymbol{\theta}^2}{2\lambda^2}\right)\right)\\ &= -\frac{1}{2\sigma^2}(\mathbf{y}-\boldsymbol{\theta}^T\mathbf{x})^2 - \frac{1}{2\lambda^2}\boldsymbol{\theta}^2 + c^{te}\\ &= -\frac{1}{2\sigma^2}||\mathbf{y}-\boldsymbol{\theta}\mathbf{x}||^2 - \frac{1}{2\lambda^2}||\boldsymbol{\theta}||^2 + c^{te}.\end{aligned} \tag{3.1.18}$$

Et le paramètre à estimer $\boldsymbol{\theta}^*$ correspond au $\boldsymbol{\theta}$ qui annule la dérivée partielle de $\log\mathbb{P}(\mathbf{y}|\mathbf{x},\boldsymbol{\theta})$ par rapport à $\boldsymbol{\theta}$.

$$\frac{\partial\log\mathbb{P}(\mathbf{y}|\mathbf{x},\boldsymbol{\theta})}{\partial\boldsymbol{\theta}} = 0 \Longleftrightarrow \frac{\partial\log}{\partial\boldsymbol{\theta}}\left(-\frac{1}{2\sigma^2}||\mathbf{y}-\boldsymbol{\theta}\mathbf{x}||^2 - \frac{1}{2\lambda^2}||\boldsymbol{\theta}||^2 + c^{te}\right) = 0.$$

Ceci revient à déterminer le $\boldsymbol{\theta}$ qui annule l'expression

$$\frac{1}{\sigma^2}\mathbf{x}^T(\mathbf{y}-\boldsymbol{\theta}\mathbf{x}) - \frac{1}{\lambda^2}\boldsymbol{\theta}. \tag{3.1.19}$$

Alors,

$$\begin{aligned}&-\frac{1}{\sigma^2}\mathbf{x}^T(\mathbf{y}-\boldsymbol{\theta}\mathbf{x}) - \frac{1}{\lambda^2}\boldsymbol{\theta} = 0 \Longleftrightarrow -\frac{1}{\sigma^2}\mathbf{x}^T\mathbf{y} + \frac{1}{\sigma^2}\boldsymbol{\theta}\mathbf{x}^T\mathbf{X} - \frac{1}{\lambda^2}\boldsymbol{\theta} = 0\\ &(\frac{1}{\sigma^2}\mathbf{x}^T\mathbf{x} - \frac{1}{\lambda^2})\boldsymbol{\theta} = \frac{1}{\sigma^2}\mathbf{x}^T\mathbf{y} \Longleftrightarrow \boldsymbol{\theta}^* = \left(\mathbf{x}^T\mathbf{x} - \frac{\sigma^2}{\lambda^2}\mathbf{I}\right)^{-1}\mathbf{x}^T\mathbf{y}.\end{aligned} \tag{3.1.20}$$

Cas Pratique

3.2 Les Problèmes de Classification

A la différence avec le problème de régression, la classification est un autre type de problème d'apprentissage supervisé où la variable à prédire est discrète (ou qualitative ou catégorique). Cette variable discrète peut être binaire (deux classes) ou multiple (multi-classe). Par exemple lorsqu'on veut **catégoriser** si un e-mail reçu est un 'spam' ou "non-spam" il s'agit bel et bien d'un problème de classification.

3.2.1 L'algorithme des K plus proches voisins (K-NN)

L'algorithme des K plus proches voisins aussi appelé K-Nearest Neighbors (K-NN) en anglais est une méthode d'apprentissage supervisé utilisée pour la classification aussi bien que la régression () [@goodfellow2016deep]. Il est compté parmi les plus simples algorithmes d'apprentissage automatique supervisé, facile à mettre en oeuvre et à comprendre.

Toutefois dans l'industrie, il est plus utilisé pour les problèmes de classification. Son fonctionnement se base sur le principe suivant: *dis moi qui sont tes voisins, je te dirais qui tu es …*

L'objectif de cet algorithme est de déterminer la classe d'une nouvelle observation x en fonction de la classe majoritaire parmi ses K plus proches voisins. Donc l'algorithme est basé sur la mesure de similarité des voisins proches pour classifier une nouvelle observation x.

La méthode des K plus proches voisins, où K représente le nombre de voisins proches est une méthode non-paramétrique. Cela signifie que l'algorithme permet de faire une classification sans faire d'hypothèse sur la fonction $y = f(\mathbf{x}_1, \mathbf{x}_2, \ldots \mathbf{x}_n)$ qui relie la variable dépendante **y** aux variables indépendantes $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n$.

Soit $\mathcal{D}$ l'ensemble des données ou l'échantillon d'apprentissage, défini par:

$$\mathcal{D} = \{(\mathbf{x}_i, y_i), i = 1, \ldots, n\}, \tag{3.2.1}$$

où $y_i \in \{1, \ldots, c\}$ dénote la classe de la donnée i et $\mathbf{x}_i = (\mathbf{x}_{i1}, \ldots, \mathbf{x}_{im})$ est le vecteur représentant les variables (attributs) prédictrices de la donnée i.

Supposons un nouveau point **p** pour lequel nous voulons prédire la classe dans la quelle il doit appartenir comme indiqué dans la figure *1.7* (page **??**).

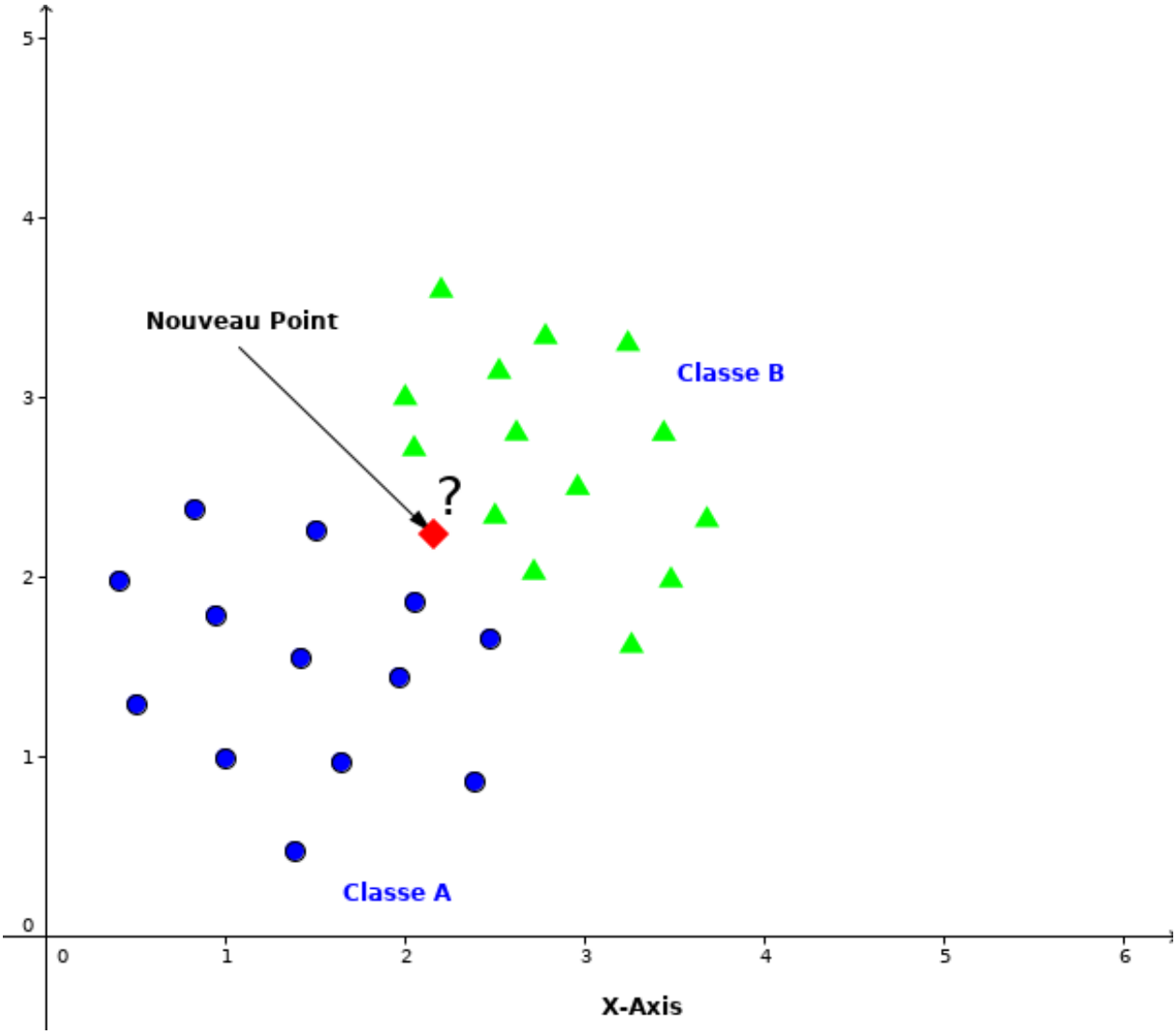


Fig. 3.2.1: Classification d'un nouveau point entre deux classes

La première chose à faire est de calculer la distance entre le point **p** avec tous les autres points. Ensuite trouver les K points les plus proches de **p**. C'est-à-dire les K points dont la distance avec **p** est minimale. Les $K-$points

plus proches de **p** dans l'échantillon d'apprentissage sont obtenus par:

$$K - \underset{\mathbf{x}_i}{\text{argmin}} \{d(\mathbf{p}, \mathbf{x}_i), i = 1, \ldots, n\}. \tag{3.2.2}$$

Pour tout $i \in \{1, \ldots, n\}, d_{p,i} := \{d(p, \mathbf{x}_i),\ i = 1, \ldots, n\}$ où d est une fonction de distance. Et en suite la classe prédite de **p** notée $\hat{\mathbf{y}}$ est la classe majoritairement représentée par les k voisins.

Les points similaires ou les points les plus proches sont sélectionnés en utilisant une fonction de distance telle que la distance euclidienne *[Euclidienne]* (page **??**), la distance de Manhattan *[Manhattan]* (page **??**) et la distance de Minkowski *[Minkowski]* (page **??**). On choisit la fonction de distance en fonction des types de données manipulées, par exemple dans le cas où les données sont quantitatives et du même type, c'est la distance euclidienne qui est utilisée.

Les points les plus proches de P sont trouvés en utilisant une fonction de distance telle que la distance Euclidienne, la distance de Minkowski et la distance de Manhattan.

Algorithme

Soient $\mathcal{D}$ un échantillon d'apprentissage des observations $\mathbf{x}_i$ relatives à une classe y_i $\mathcal{D} = \{(\mathbf{x}_i, y_i), i = 1, \ldots, n\}$ et **p** une nouvelle observation dont la classe $\hat{c}$ doit être prédite. **Les étapes de l'algorithme:** Ainsi, l'algorithme se présente comme suit:

1. Choisir le paramètre K, le nombre de voisins les plus proches;
2. Calculer la distance de la nouvelle observation **p** avec tous les autres échantillons selon une fonction de distance choisie $d(p, \mathbf{x}_i)$;
3. Sélectionner les K plus proches voisins de **p**;
4. Former la collection K_c des étiquettes des K plus proches voisins de **p**;
5. Et la classe de **p**, $\hat{c}$ est choisie d'après la majorité des K_c plus proches voisins, c'est-à-dire

$$\hat{c} = \text{Mode}(K_c) \tag{3.2.3}$$

.

6. Répéter l'étape 2 à 5 pour chaque nouveau point à classifier.

L'algorithme *[algorithme_knn]* (page **??**) nous présente le pseudo-code de la méthode de plus proches voisins.

Un ensemble de données $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}, i = 1, \ldots, n$ Choisir une fonction de distance d Choisir un nombre $K \in \mathbb{N}^*$ Pour une nouvelle observation **p** dont on veut prédire la classe $\hat{c}$: Calculer la distance $d(\mathbf{p}, \mathbf{x}_i)$ Retenir les K observations proches de **p**: $K - \underset{\mathbf{x}_i}{\arg\min} \{d(\mathbf{p}, \mathbf{x}_i), i = 1, \ldots, n\}$ Prendre les valeurs y_k des K observations retenues: Si on effectue une régression: $\hat{c} = \frac{1}{K} \sum_{k=1}^{K} y_k$ (la moyenne ou la médiane des y_k retenues) Si on effectue une classification : Calculer le mode des y_k retenues Retourner $\hat{c}$, la valeur qui a été prédite par K-NN pour l'observation **p**.

Comment choisir la valeur de K ?

- En générale, le choix de la valeur de $K \in \mathbb{N}^*$ dépend du jeu de données. Pour la classification binaire (en deux classes) par exemple il est préférable de choisir la valeur K impaire pour éviter les votes égalitaires. Historiquement, la valeur optimale de K pour la plupart de données est choisie entre 3 et 10 [@shalev2014understanding].
- Une optimale valeur de K peut être sélectionnée par diverses techniques heuristiques dont la **validation-croisée** (que nous allons expliquer ci-dessous).
- Notons que, si l'algorithme est utilisé pour la régression, c'est la moyenne (ou la médiane) des variables **y** des K plus proches observations qui sera utilisée pour la prédiction. Et dans le cas de la classification, c'est le mode des variables **y** des K plus proches observations qui servira pour la prédiction.

Validation-croisée (Cross-Validation)

La Validation-croisée (*Cross-validation*) est une méthode très populaire utilisée pour estimer la performance d'un algorithme. C'est une méthode statistique souvent utilisée dans des procédures d'estimation et aussi pour la sélection de modèles [@chen2019mehryar].

Son principe est le suivant :

- séparer les données en données en deux échantillon (apprentissage et validation);
- construire l'estimateur sur l'échantillon d'apprentissage et utiliser l'échantillon de validation pour évaluer l'erreur de prédiction;
- Répéter plusieurs fois le processus et enfin faire une moyenne des erreurs de prédiction obtenues.

C'est une technique très utilisée pour le choix de meilleurs paramètres et hyperparamètres d'un modèle, par exemple pour le choix de la meilleure valeur de K dans l'algorithme de plus proches voisins (K-NN).

On distingue les variantes suivantes de la technique de validation-croisée:

- K-fold cross-validation : Partitionnement des données en K sous-ensembles. Chaque sous-ensemble sert à tour de rôle d'échantillon de validation et le reste de sous-ensembles d'échantillon d'apprentissage. En pratique la valeur de K varie entre 5 et 10.
- Leave-one-out cross-validation: qui signifie de laisser à tour de rôle une observation comme échantillon de validation et le reste des données comme échantillon d'apprentissage. C'est un n-fold validation-croisée avec n, le nombre total d'observations.
- Leave-q-out qui signifie de laisser à tour de rôle q observations comme échantillon de validation et le reste des données comme échantillon d'apprentissage. C'est une $\lceil \frac{n}{q} \rceil$-fold validation-croisée.

D'une manière générale, la procédure de partitionnement des données se présente souvent comme suit:

$$\underbrace{Apprentissage}_{70\%} + \underbrace{Validation}_{10\%} + \underbrace{Test}_{20\%} \tag{3.2.4}$$

- Les données d'apprentissage permettent de trouver un estimateur.
- Les données de validation nous permettent de trouver les meilleurs paramètres du modèle.
- Les données test permettent de calculer l'erreur de prédiction finale. Notons que cette méthode de validation-croisée est utilisée pour tous types d'algorithme d'apprentissage.

Les avantages de l'algorithme de K-NN

- Il est simple, facile à interpréter.
- Il n'existe pas de phase d'apprentissage proprement dite comme c'est le cas pour les autres algorithmes, c'est pour cela qu'on le classifie dans le *Lazy Learning*.
- Offre des performances très intéressantes lorsque le volume de données d'apprentissage est trop large.
- Le temps d'exécution est minimum par rapport à d'autres algorithmes de classification.
- Il peut être utilisé pour classification et régression.
- Il ne fait pas d'hypothèse (linéaire, affines,..) sur les données.

Les limitations de l'algorithme de K-NN

- L'algorithme a plus besoin de mémoire car l'ensemble des données doivent être garder dans cette dernière pour pouvoir effectuer la prédiction d'une nouvelle observation à chaque fois.
- Sensibles aux attributs non pertinents et non corrélés.
- L'étape de la prédiction peur être lente dû au calcul de distance de chaque nouvelle observation avec les jeux de données en entier à chaque prédiction.
- Le choix de la fonction de distance ainsi que le nombre de voisins K peut ne pas être évident. C'est ainsi qu'il faut essayer plusieurs combinaisons(en utilisant la méthode de validation-croisée) pour avoir un bon résultat.

Exemple pratique

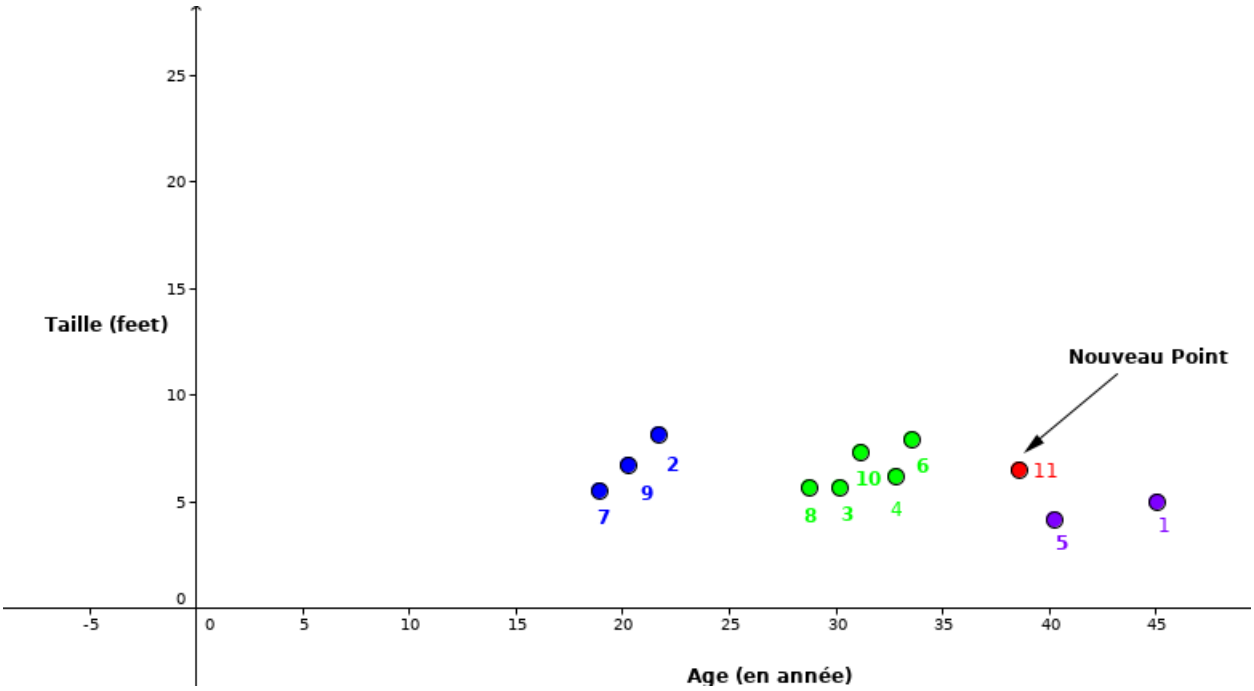
Ci-dessous, nous allons prendre un exemple simple pour comprendre l'intuition derrière l'algorithme de K-NN. Considérons le tableau de données ci-dessous qui contient la taille (feet), l'âge (année) et le poids(en kilogramme) de 10 personnes où ID représente l'identifiant de chaque personne dans le tableau. Comme vous le remarquez le poids du ID 11 est manquant. Nous allons appliquer l'algorithme de K-NN pour prédire le poids de la personne ID 11.

Table 3.2.1: Données pour l'illustration

ID	Taille	Âge	Poids
1	5	45	77
2	5.11	26	47
3	5.6	30	55
4	5.9	34	59
5	4.8	40	72
6	5.8	36	60
7	5.3	19	40
8	5.8	28	60
9	5.5	23	45
10	5.6	32	58
11	5.5	38	?

[tab:rrr]

Nous pouvons représenter graphiquement les données du tableau *1.1* (page **??**) en se basant sur la taille et l'âge.



Comme nous le remarquons, le point rouge (ID 11) est notre nouvelle observation dont nous voulons prédire la classe dans laquelle il appartient.

Étape 1: Commençons par choisir le nombre des voisins les plus proche. Pour notre cas prenons $K = 5$.

Étape 2: Calculer la distance entre le nouveau point (rouge) avec tous les autres points

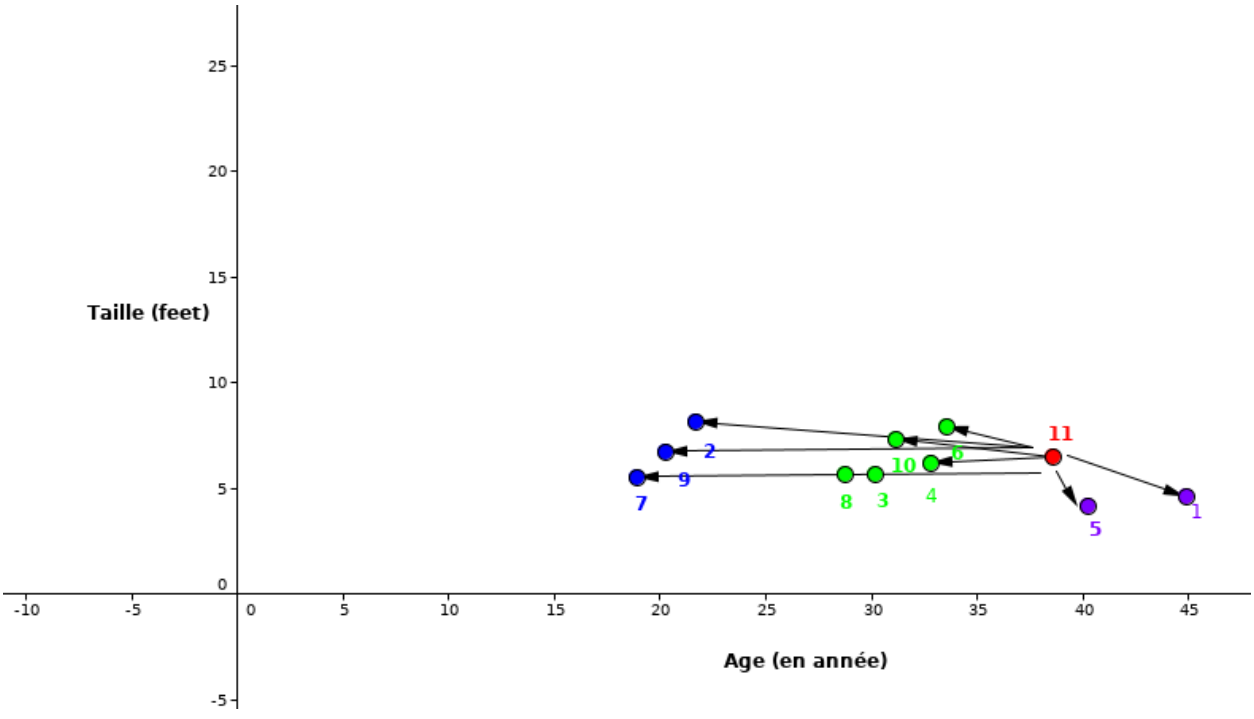


Fig. 3.2.2: La distance euclidienne entre nouvelle observation et tous les autres points

Étape 3: Sélectionner les 5 plus proches voisins.

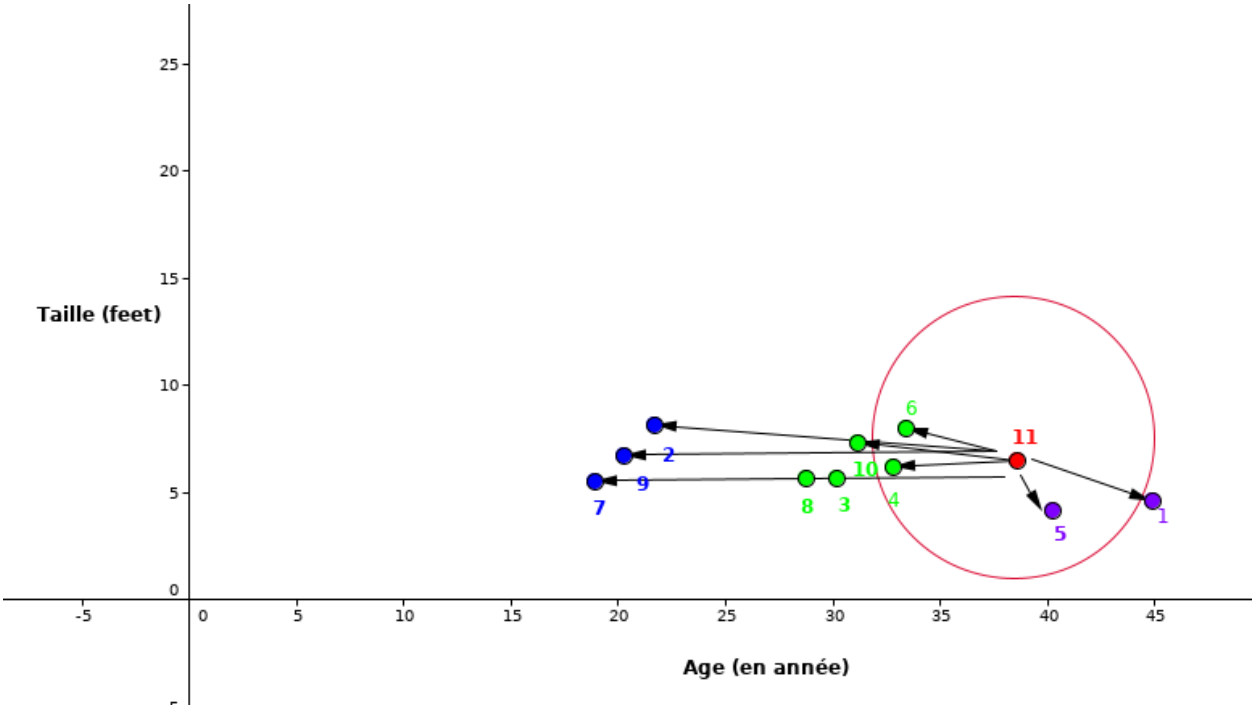


Fig. 3.2.3: Sélection de 5 plus proches voisins

Étape 4: Pour la valeur de $K = 5$, les points les plus proches sont $1,\ 4,\ 5,\ 6$ et 10.

Étape 5: Ainsi, comme la classe des points $4,\ 6$ et 10 est majoritaire donc le point 11 se classifie dans cette classe. Et comme c'est un cas de la régression, la prédiction pour ID11 est la moyenne de ces 5 voisins les plus proches c'est-à-dire $(77 + 59 + 72 + 60 + 58)/5 = 65.2\ Kg$. Ainsi le poids prédit pour ID11 est $65.2\ Kg$.

3.2.2 L'algorithme du Perceptron

L'algorithme de perceptron est un algorithme d'apprentissage supervisé utilisé pour la classification binaire (c'est-à-dire séparant deux classes). C'est l'un des tout premier algorithme d'apprentissage supervisé et de réseau de neurones artificiels le plus simple. Le terme vient de l'unité de base dans un neurone[8] qui s'appelle *perceptron*.

C'est un type de classification linéaire, c'est-à-dire que les données d'apprentissage sont séparées par une droite classées dans des catégories correspondantes de telle sorte que si la classification à deux catégories est appliquée, toutes les données sont rangées dans ces deux catégories. Dans ce cas on cherche à trouver un hyperplan qui sépare correctement les données en deux catégories. Comme nous le voyons dans la figure *1.12* (page **??**) ci-dessous.

|L'algorithme du perceptron| |L'algorithme du perceptron|

L'idée générale de l'algorithme du perceptron est d'initialiser le vecteur de poids réels $\mathbf{w} \in \mathbb{R}^d$ au vecteur nul ou à une variable aléatoire, itérer un nombre de fois jusqu'à la convergence sur les données d'apprentissage.

Soient $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$, un ensemble de données où $\mathbf{x}_i \in \mathbb{R}^d$ est le vecteur d'entrées de dimension d et l'étiquette $y_i \in \{-1, 1\}$; $\mathbf{w} \in \mathbb{R}^d$ un vecteur poids de dimension d.

[8] https://fr.wikipedia.org/wiki/R%C3%A9seau_de_neurones_artificiels#Perceptron_2

L'objectif est de trouver un hyperplan $\mathbf{w}.\mathbf{x} + b = 0$ qui sépare les deux classes. Le terme b est l'intercepte appelé aussi "*bias term*" et $\mathbf{w} \cdot \mathbf{x}$ est le produit scalaire défini par $\langle \mathbf{w}, \mathbf{x} \rangle := \sum_{s=1}^{d} w_s \mathbf{x}_s$.

C'est-à-dire apprendre le vecteur **w** tel que:

$$\mathbf{y} = \text{signe}\,(\mathbf{w} \cdot \mathbf{x} + b) \tag{3.2.5}$$

$$\begin{aligned} &\text{Si } \mathbf{w} \cdot \mathbf{x} + b > 0 \text{ alors } \operatorname{sgn}(\mathbf{w} \cdot \mathbf{x} + b) = +1, \text{ pour tout } \mathbf{x} \text{ appartenant à la classe positive.} \\ &\text{Si } \mathbf{w} \cdot \mathbf{x} + b \leq 0 \text{ alors } \operatorname{sgn}(\mathbf{w} \cdot \mathbf{x} + b) = -1, \text{ pour tout } \mathbf{x} \text{ appartenant à la classe négative.} \end{aligned} \tag{3.2.6}$$

Algorithme

Soient les données d'apprentissage $\mathcal{D} = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}$ où $\mathbf{x}_i \in \mathbb{R}^d$, $y_i \in \{-1, 1\}$ et T le nombre d'itérations. L'algorithme se présente comme suit:

Initialiser le vecteur $\mathbf{w} \leftarrow 0$ **Pour** itération de 1 à T: ** Pour** chaque exemple $(\mathbf{x}_i, y_i) \in \mathcal{D}$: Calculer la prédiction $\hat{y_i} = \text{signe}(\mathbf{w}.\mathbf{x}_i + b)$ ** Si** $\hat{y_i} \neq y_i$ **alors** Ajuster **w** : par $\mathbf{w_{t+1}} \leftarrow \mathbf{w_t} + \mathbf{x}_i$ si y_i est positive $\mathbf{w_{t+1}} \leftarrow \mathbf{w_t} - \mathbf{x}_i$ si y_i est négative **Fin si** **Fin pour Fin pour**

L'avantage de l'algorithme de perceptron est sa simplicité et son efficacité de séparer linéairement les données d'apprentissage. Néanmoins, tous les hyperplans qui séparent les données sont équivalents.

L'algorithme ne peut séparer et converger vers la solution uniquement que si les données d'apprentissage sont linéairement séparables. Aussi, il n'est pas efficace quand il y a beaucoup d'attributs.

3.2.3 La Régression Logistique

La régression logistique est une méthode de classification simple mais puissante, pour les données binaires, et elle peut facilement être étendue à plusieurs classes. Considérons d'abord le modèle de régression le plus simple, correspondant à celui que nous avons vu précédemment, pour effectuer la classification. Nous le trouverons bientôt totalement insuffisant pour ce que nous voulons réaliser et il sera instructif de voir exactement pourquoi. Le modèle de la régression linéaire comme défini dans l'équation *[reg_lin]* (page **??**), peut se réécrire comme:

$$h_{\boldsymbol{\theta}}(\mathbf{X}) = \mathbf{X}\boldsymbol{\theta}, \tag{3.2.7}$$

où **X** est une matrice de taille $n \times d$ et $\boldsymbol{\theta} \in \mathbb{R}^d$.

Comme dans de nombreux problèmes d'estimation de paramètres, $\boldsymbol{\theta}$ est trouvé en minimisant certaines fonctions de pertes qui capturent à quel point notre prédiction est proche de la valeur réelle. Quand nous faisons des hypothèses sur la distribution des données, la fonction de perte est souvent en termes de vraisemblance des données. Cela signifie que le $\boldsymbol{\theta}$ optimal est celui pour lequel les données observées ont la probabilité la plus élevée. Par exemple, la régression linéaire suppose généralement que la variable dépendante est normalement distribuée autour de la moyenne $h_{\boldsymbol{\theta}}(\mathbf{x})$. Il peut être montré que la solution du maximum de vraisemblance est le $\boldsymbol{\theta}$ qui minimise la somme des erreurs quadratiques (la différence entre les valeurs prédites et les valeurs correctes). Si nous essayons d'utiliser la régression linéaire pour un problème de classification (prenons l'exemple d'une classification binaire) une méthode simple serait de grouper les données de telle sorte que:

$$y = \begin{cases} 1, & \text{si } \boldsymbol{\theta}^T \mathbf{x} > 0 \quad (\mathbf{x} \in \mathbb{R}^d) \\ 0, & \text{sinon} \end{cases} \tag{3.2.8}$$

Ceci est illustré par la figure *1.13* (page **??**).

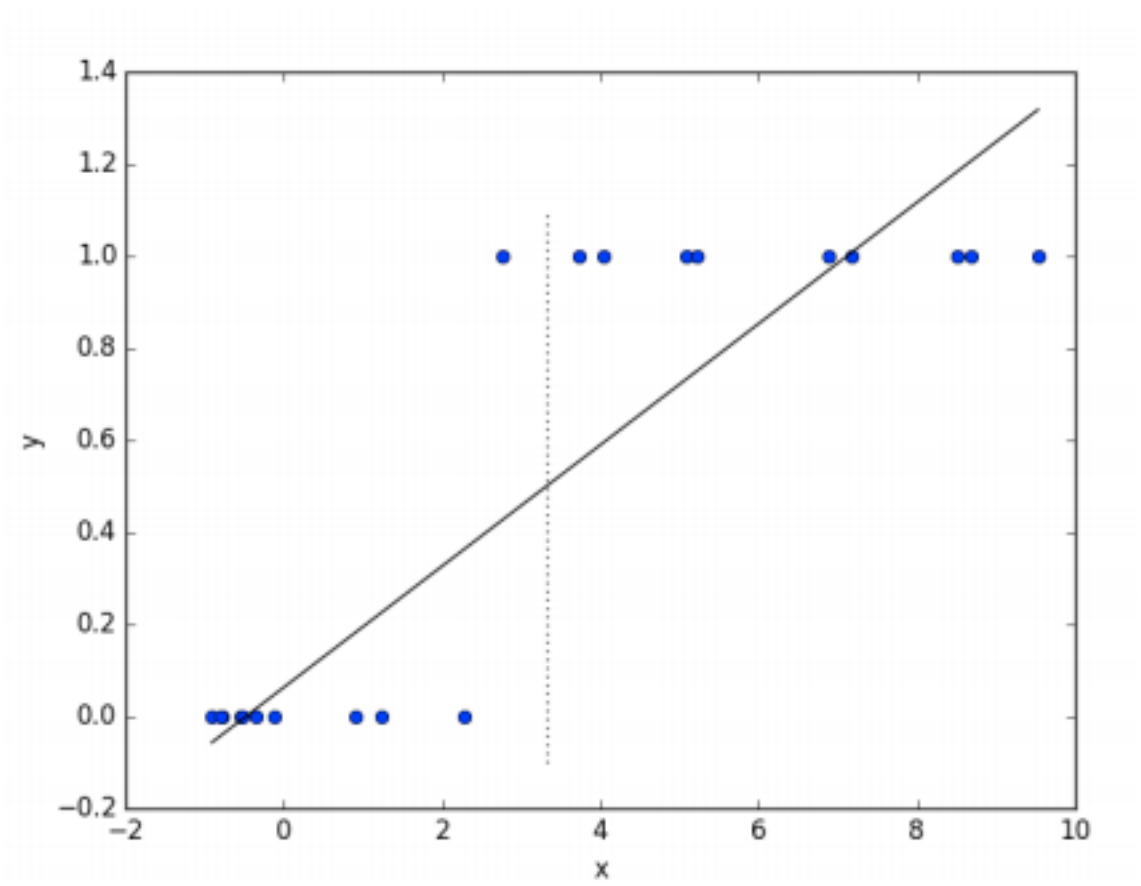


Fig. 3.2.4: Régression linéaire dans le cas d'une classification binaire

La régression logistique binaire

La régression logistique ordinaire ou régression logistique binaire vise à expliquer une variable d'intérêt binaire (c'est-à-dire de type « oui / non » ou « vrai / faux »). Les variables explicatives qui seront introduites dans le modèle peuvent être quantitatives (l'âge, la taille, etc) ou qualitatives (le genre par exemple).

Exemple

Dans cet exemple, la variable explicative x est une matrice de vecteurs colonnes $\mathbf{x}_1, \mathbf{x}_2$ où $\mathbf{x}_1$ est le vecteur 'apprendre' qui représente le nombre d'heures d'étude de l'étudiant, et $\mathbf{x}_2$ est le nombre d'heures pendant lesquelles l'étudiant dort. L'objectif est de prédire si l'étudiant va réussir à l'examen ou non, respectivement représentée par les classes 1 et 0.

Apprendre	Dormir	Réussir
4.85	9.63	1
8.62	3.23	0
5.43	8.23	1
9.21	6.34	0

Comme dans le cas de la régression linéaire, on suppose que les données suivent une fonction linéaire de la forme:

$$\mathbf{y} = \boldsymbol{\theta}^T \mathbf{x}. \tag{3.2.9}$$

y représente la variable à expliquer, **x** est la variable explicative et $\boldsymbol{\theta}$ un paramètre. Comme nous l'avons vu, la variable **y** est une variable continue. Pour utiliser cette technique dans le cas d'une variable discrète (ici binaire), supposons que p est la probabilité qu'un événement se réalise; alors $1-p$ est la probabilité de l'évènement contraire. La variable aléatoire **y** qui prend les valeurs 'oui' ou 'non' (1,0 en langage machine) suit la loi de Bernoulli. On définit ce qu'on appelle la transformation logit donnée par l'équation suivante:

$$\log\left(\frac{p}{1-p}\right) = \boldsymbol{\theta}^T\mathbf{x}. \tag{3.2.10}$$

Cette transformation donne la relation entre la probabilité qu'un évenment se réalise et la combinaison linéaire des variables. Le rapport $\frac{p}{1-p}$ est appelé le Rapport de Côte (RC). En appliquant l'exponentielle sur cette relation on obtient:

$$\frac{p}{1-p} = e^{\boldsymbol{\theta}^T\mathbf{x}}. \tag{3.2.11}$$

Ce qui implique:

$$\begin{aligned} p &= \frac{e^{\boldsymbol{\theta}^T\mathbf{x}}}{1+e^{\boldsymbol{\theta}^T\mathbf{x}}} \\ &= \frac{1}{1+e^{-\boldsymbol{\theta}^T\mathbf{x}}} \end{aligned}$$

Avec $z = \boldsymbol{\theta}^T\mathbf{x}$, la fonction σ définie par $\sigma(z) = \dfrac{1}{1+e^{-z}}$ est appelée la fonction Sigmoïde ou logistique. Dans les lignes qui suivent, nous allons donner certaines de ses propriétés.

3.2.4 La fonction Sigmoïde et ses propriétés

Elle est définie par:

$$\sigma(z) = \frac{1}{1+e^{-z}}, \tag{3.2.12}$$

La représentation graphique de la fonction Sigmoïde est donnée par la figure *1.14* (page **??**).

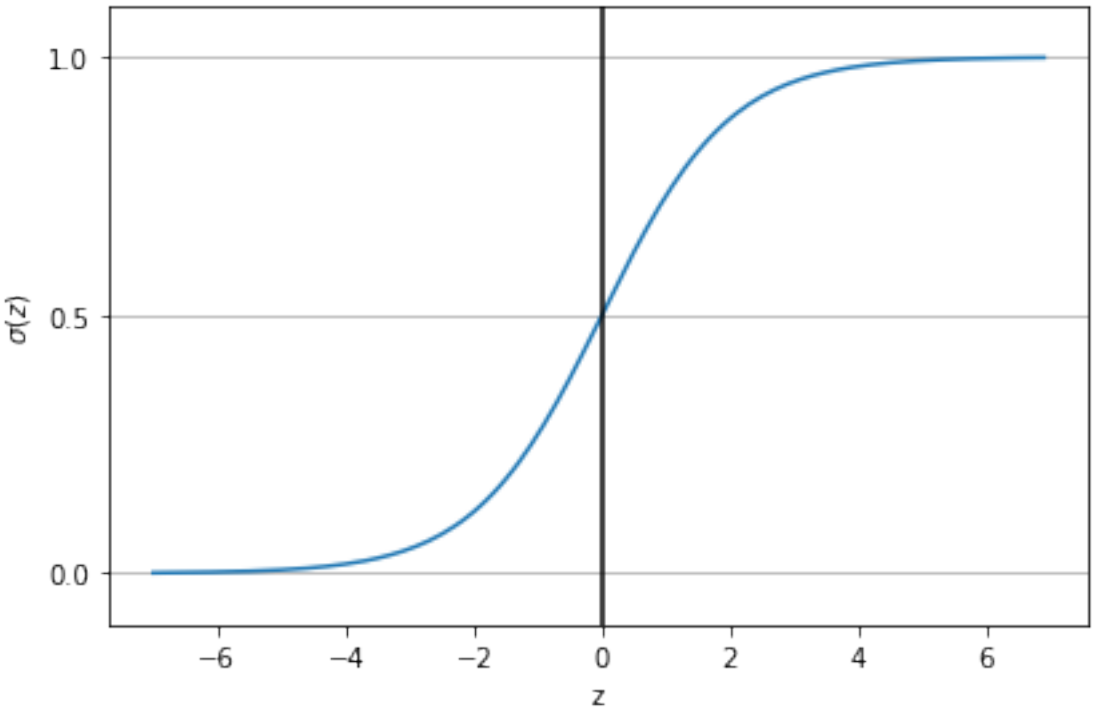


Fig. 3.2.5: La fonction Sigmoïde

Une qualité importante de cette fonction est qu'elle transforme tous les nombres réels sur la plage $[0, 1]$. En régression logistique, cette transformation $\sigma(z)$ nous permet d'avoir une vue probabiliste qui est d'une importance cruciale pour la classification. Avec cette fonction, les nombres positifs deviennent des probabilités élevées; les nombres négatifs deviennent de faible probabilité.

Comme vous l'avez sûrement remarqué, l'algorithme de régression linéaire repose sur l'obtention d'un paramètre $\boldsymbol{\theta}^*$ dit optimal. Dans la section suivante nous allons discuter sur le choix et l'obtention de la valeur $\boldsymbol{\theta}^*$.

Estimation du maximum de vraisemblance

Pour choisir la valeur du paramètre $\boldsymbol{\theta}$, on utilise la méthode du maximum de vraisemblance. La variable à expliquer $\mathbf{y}$ est une variable binaire, i.e $\mathbf{y} \in \{0, 1\}$ et la fonction Sigmoïde nous permet de projeter les résultats dans l'intervalle $[0, 1]$. Plus précisément, pour un z considéré, on choisit $\sigma(z) \in [0, 1]$ comme étant un paramètre d'une loi de Bernoulli et ainsi, on a pour tout $i = 1, \ldots, n$:

$$\begin{aligned} \mathbb{P}(Y = y_i) &= (\sigma(z))^{y_i} (1 - \sigma(z))^{1-y_i} \\ &= \left(\sigma(\boldsymbol{\theta}^T \mathbf{x}_i)\right)^{y_i} \left(1 - \sigma(\boldsymbol{\theta}^T \mathbf{x}_i)\right)^{1-y_i} \end{aligned} \tag{3.2.13}$$

Par suite, on peut exprimer la vraisemblance:

$$\ell(\boldsymbol{\theta}) = \prod_{i=1}^{n} \mathbb{P}\left(Y = y_i\right) \tag{3.2.14}$$

En appliquant la fonction logarithme, on obtient le logarithme-vraisemblance donnée par l'expression suivante:

$$\log(\ell(\boldsymbol{\theta})) \equiv L(\boldsymbol{\theta}) = \sum_{i=1}^{n} y_i \log \sigma\left(\boldsymbol{\theta}^T \mathbf{x}_i\right) + (1 - y_i) \log\left(1 - \sigma(\boldsymbol{\theta}^T \mathbf{x}_i)\right) \tag{3.2.15}$$

Notre objective est de trouver le paramètre $\boldsymbol{\theta}$ qui maximise la vraisemblance:

$$\max_{\boldsymbol{\theta}} L(\boldsymbol{\theta}) = \max_{\boldsymbol{\theta}} \sum_{i=1}^{n} y_i \log \sigma\left(\boldsymbol{\theta}^T \mathbf{x}_i\right) + (1 - y_i) \log\left(1 - \sigma(\boldsymbol{\theta}^T \mathbf{x}_i)\right) \tag{3.2.16}$$

En générale, pour trouver l'estimation du maximum de vraisemblance, nous dérivons d'abord la log-vraisemblance par rapport à $\boldsymbol{\theta}$. Pour commencer, prenons la dérivée par rapport à une composante de $\boldsymbol{\theta}$, disons θ^j

$$\begin{aligned} \frac{\partial}{\partial \theta^j} L(\boldsymbol{\theta}) &= \left[y_i \frac{1}{\sigma(\boldsymbol{\theta}^T \mathbf{x}_i)} - (1 - y_i) \frac{1}{1 - \sigma(\boldsymbol{\theta}^T \mathbf{x}_i)} \right] \sigma(\boldsymbol{\theta}^T \mathbf{x}_i)(1 - \sigma(\boldsymbol{\theta}^T \mathbf{x}_i))(\frac{\partial}{\partial \theta^j} \boldsymbol{\theta}^T \mathbf{x}_i) \\ &= \left[y_i (1 - \sigma(\boldsymbol{\theta}^T \mathbf{x}_i)) - (1 - y_i) \sigma(\boldsymbol{\theta}^T \mathbf{x}_i) \right] \mathbf{x}_i^j \\ &= \left(y_i - \sigma(\boldsymbol{\theta}^T \mathbf{x}_i) \right) \mathbf{x}_i^j \end{aligned} \tag{3.2.17}$$

Une méthode classique de trouver le $\boldsymbol{\theta}$ optimal est de poser la dérivée $\frac{\partial}{\partial \theta^j} L(\boldsymbol{\theta}) = 0$ pour tout j et trouver la valeur exacte de $\boldsymbol{\theta}$ qui maximise la vraisemblance. Cependant, la solution exacte n'est pas toujours facile à calculer à cause du fait que c'est une équation transcendantale (il n'y a pas de solution analytique). Par conséquent, la technique souvent utilisée pour résoudre ce problème est la méthode de la descente de gradient *1.1.1.2* (page **??**).

Ainsi en utilisant la technique de la descente de gradient, le paramètre θ^j sera mis à jour par la formule suivante, pour chaque itération:

$$\theta^j = \theta^j + \gamma . \frac{\partial}{\partial \theta^j} L(\boldsymbol{\theta}). \tag{3.2.18}$$

Le paramètre γ est appelé taux d'apprentissage. Le paramètre $\boldsymbol{\theta}^*$ obtenu à la sortie de cet algorithme maximise la log-vraisemblance.

Ainsi nous considérons un seuil 0.5 et regroupons les données comme suit:

$$\hat{y} = \begin{cases} 1, & \text{si } \sigma(\boldsymbol{\theta}^{*T} \mathbf{x}) > 0.5 \\ 0, & \text{sinon} \end{cases} \tag{3.2.19}$$

Cas pratique

3.2.5 Régression logistique multinomiale

La régression logistique multinomiale est une forme généralisée de la régression logistique binaire utilisée pour estimer la probabilité quand le nombre de classe C est supérieur à deux. Prenons l'exemple de reconnaissance de chiffres à partir de leur image. Cette tâche consiste à identifier une image comme l'un des éléments de l'ensemble $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$. Dans ce cas, la fonction [^1]Sigmoïde utilisée dans le cas binaire est remplacée par la [^2]*fonction Softmax*.

3.2.6 La fonction Softmax et ses propriétés

La régression logistique multinomiale utilise une *fonction Softmax* pour modéliser la relation entre les variables explicatives et les probabilités de chaque classe. Elle prédit la classe qui a la probabilité la plus élevée parmi toutes les classes.

La fonction Softmax est définie par:

$$\text{Softmax}(z)_i = \frac{e^{z_i}}{\sum_{j=1}^{C} e^{z_j}}, \quad i = 1, \cdots, C.u \tag{3.2.20}$$

où C est le nombre de classes.

Cette fonction Softmax prend comme entrée $\mathbf{z}$ qui est un vecteur de dimension n et produit $\hat{\mathbf{y}}$ un vecteur de même dimension de valeurs réelles entre 0 et 1.

Toutes les valeurs z_i sont les éléments du vecteur d'entrée et peuvent prendre n'importe quelle valeur réelle. Le dénominateur de la formule *[soft]* (page **??**) est le terme de normalisation qui garantit que toutes les valeurs de sortie de la fonction totaliseront 1, constituant ainsi une distribution de probabilité valide.

On peut écrire la probabilité de la classe c pour $c = 1, \cdots, C$ sachant $\mathbf{x}$ comme:

$$\begin{bmatrix} \mathbb{P}(y=1|\mathbf{x}) \\ \vdots \\ \mathbb{P}(y=C|\mathbf{x}) \end{bmatrix} = \begin{bmatrix} \text{Softmax}(\mathbf{x})_1 \\ \vdots \\ \text{Softmax}(\mathbf{x})_C \end{bmatrix} = \frac{1}{\sum_{j=1}^{C} e^{\boldsymbol{\theta}^T \mathbf{x}_j}} \begin{bmatrix} e^{\boldsymbol{\theta}^T \mathbf{x}_1} \\ \vdots \\ e^{\boldsymbol{\theta}^T \mathbf{x}_C} \end{bmatrix} \tag{3.2.21}$$

3.2.7 Estimation du maximum de vraisemblance

De la même façon que dans le cas de la régression binaire, nous suivrons la même procédure pour déterminer le $\boldsymbol{\theta}$ qui maximise la vraisemblance.

$$\ell(\boldsymbol{\theta}) = \prod_{i=1}^{n} \mathbb{P}(y_i|\mathbf{x}; \boldsymbol{\theta}). \tag{3.2.22}$$

L'équation *[lv]* (page **??**) suppose que les instances de données ont été générées indépendamment. Ainsi, en appliquant le logarithme sur *[lv]* (page **??**), nous obtenons:

$$\begin{aligned} L(\boldsymbol{\theta}) &= \sum_{j=1}^{n} \sum_{i=1}^{C} \log \left[\left(\frac{e^{\boldsymbol{\theta}_i^T \mathbf{x}_j}}{\sum_{k=1}^{C} e^{\boldsymbol{\theta}_k^T \mathbf{x}_j}} \right)^{y_j} \right] \\ &= \sum_{j=1}^{n} \sum_{i=1}^{C} y_j \log \left(\frac{e^{\boldsymbol{\theta}_i^T \mathbf{x}_j}}{\sum_{k=1}^{C} e^{\boldsymbol{\theta}_i^T \mathbf{x}_j}} \right). \end{aligned} \tag{3.2.23}$$

Donc maximiser la log-vraisemblance équivaut à minimiser l'opposé de la log-vraisemblance donnée par l'expression suivante :

$$L(\boldsymbol{\theta}) = -\sum_{j=1}^{n} \sum_{i=1}^{C} y_j \log \left(\frac{e^{\boldsymbol{\theta}_i^T \mathbf{x}_j}}{\sum_{k=1}^{C} e^{\boldsymbol{\theta}_k^T \mathbf{x}_j}} \right) = -\sum_{j=1}^{n} \sum_{i=1}^{C} y_j \left[\log \left(e^{\boldsymbol{\theta}_i^T \mathbf{x}_j} \right) - \log \left(\sum_{k=1}^{C} e^{\boldsymbol{\theta}_k^T \mathbf{x}_j} \right) \right]. \tag{3.2.24}$$

Cet opposé de la log-vraisemblance est aussi connu sous le nom de **l'entropie de Shannon (cross-entropy)** qui est la fonction de perte dans ce cas. Pour trouver le $\boldsymbol{\theta}$ qui minimise cette fonction de perte, on suit la même procédure que dans le cas de la régression logistique, c'est-à-dire trouver la dérivée de la fonction de perte et appliquer l'algorithme de la descente de gradient.

En Calculant la dérivée de la fonction de perte par rapport à $\boldsymbol{\theta}_j$ on a:

$$\frac{\partial}{\partial \boldsymbol{\theta}_j} L(\boldsymbol{\theta}) = -\sum_{i=1}^{n} y_i \mathbf{x}_i + \sum_{i=1}^{n} \frac{1}{\sum_{k=1}^{C} e^{\boldsymbol{\theta}_k^T \mathbf{x}_i}} e^{\boldsymbol{\theta}_j^T \mathbf{x}_i} \mathbf{x}_i \tag{3.2.25}$$

Maintenant on utilise l'algorithme de la descente de gradient durant le processus d'entraînement pour obtenir le $\boldsymbol{\theta}$ optimal. À chaque itération, on met à jour chacun des paramètres $\boldsymbol{\theta}_j$ par la formule suivante:

$$\boldsymbol{\theta}_j = \boldsymbol{\theta}_j - \gamma \frac{\partial}{\partial \boldsymbol{\theta}_j} L(\boldsymbol{\theta}). \tag{3.2.26}$$

A la sortie de cette algorithme nous obtenons le paramètre optimal $\boldsymbol{\theta}^*$. Ainsi, le vecteur $\hat{\mathbf{y}}$ prédit est donné par:

$$\hat{\mathbf{y}} = \text{argmax} \;\; \text{softmax}(\boldsymbol{\theta}^\star \mathbf{x}_{test}). \tag{3.2.27}$$

3.2.8 Naïve Bayes

Dans cette sous section nous allons introduire un autre algorithme souvent utilisé dans le cadre des problèmes de classification. Cet algorithme est connu sous le nom de **Naïve Bayes**, naïve parce qu'il fait une simple hypothèse sur les données, celle qui suppose qu'elles sont indépendantes les une des autres, même si ceci n'est souvent pas vrai en pratique.

Les systèmes modernes populaires comme la classification des émails reçus comme **spam** ou **non-spam** sont souvent implémentés avec naïves Bayes et quelques fois leur performance est difficile à surpasser par des algorithmes sophistiqués.

Naïve Bayes fait partie des algorithmes de type **génératif**, qui sont différents des algorithmes vus jusque-là qui sont de type **discriminatif**.

L'une des grandes différences entre les algorithmes de type génératif et ceux dits discriminatifs réside dans le fait que les premiers font une hypothèse sur les données $\mathbb{P}(\mathbf{x}|y)$ tandis que les derniers font une hypothèse sur les étiquettes (classes) $\mathbb{P}(y|\mathbf{x})$.

Pour ceux qui sont familiers avec les notions mathématiques, peut être que vous vous êtes posé la question si cet algorithme a un lien avec la règle de Bayes (Naïve Bayes) ? OUI! vous avez raison car cette formule nous donne un lien entre les algorithmes de type discriminatif et ceux de type génératif.

Rappelons la formule *[naivebayes]* (page **??**)

$$\mathbb{P}(\mathbf{x}|y) = \frac{\mathbb{P}(y|\mathbf{x})\mathbb{P}(\mathbf{x})}{\mathbb{P}(y)}. \tag{3.2.28}$$

Considérons le cas de la classification des emails (spam ou no-spam). Nous pouvons ré-écrire la formule précédente comme :

$$\mathbb{P}(\text{document}|\text{classe}) = \frac{\mathbb{P}(\text{classe}|\text{document})\mathbb{P}(\text{document})}{\mathbb{P}(\text{classe})}. \tag{3.2.29}$$

Avec **document** qui représente l'ensemble des émails, et **classe** leur catégorie.

Pour mieux illustrer la notion introduite dans le paragraphe précédent, réfléchissons sur un exemple pratique. Disons que vous êtes responsable d'un champ de feuille de manioc et vous voulez avoir un système qui vous alarme dès qu'une chèvre entre dans le champ. Nous supposerons (non réaliste) que dans cette contrée nous ne pouvons avoir que deux espèces d'animaux, chèvre et chien, donc nous avons deux catégories c_1="chèvre" et c_2 = "chien".

Comme vous l'avez sûrement appris dans ce cours, les algorithmes que nous utilisons en apprentissage automatique ne prennent en entrée que les données de type numérique. Alors nos données deviennent :

1. **x** qui représente certaines caractéristiques des deux animaux, par exemple : 'couleur', 'cris', 'vitesse', …, toutes ces caractéristiques ont une représentation numérique (cris : bêler=0, aboyer=1, …).

2. y qui prend la valeur “0” pour une chèvre et “1” pour un chien.

Dans ce contexte, un algorithme de type discriminatif va chercher à apprendre une application (mapping) de l'espace des **x** dans l'espace des étiquettes $\{0, 1\}$; en d'autres termes, ce type d'algorithmes va chercher à trouver une ligne (ou hyperplan) qui va séparer les chiens des chèvres étant données les caractéristiques (**x**) observées, tandis que celui de type génératif va se focaliser à la modélisation des caractéristiques qui distinguent un chien d'une chèvre.

fonction $Entrainer\_Nave\_Bayes(D,\ C)$: **Pour chaque** classe dans C: N_{doc} = nombre de documents dans D N_c = nombre de documents de classe c dans D log Prob$[c]$ = log $\frac{N_c}{N_{doc}}$ V = vocabulaire de D classedoc$[c]$ = **ajouter**(d) pour tout $d \in D$ avec pour classe c **Pour chaque** mot w de classe $c \in C$ dans V: **compter**(w, c) = nombre d'occurrence du mot w dans classedoc$[c]$ log Prob_wc$[w, c] \leftarrow$ log $\frac{\text{compter}(w,c)}{\sum_{w' \in V}(\text{compter}(w',c))}$ **retourner** log Prob, log Prob_wc, V

[train_naive]

fonction $Tester\_naive\_bayes(testdoc,\ logProb,\ logProb\_wc,\ C,\ V)$: **Pour chaque** classe $c \in C$: somme[c] = log Prob$[c]$ **Pour chaque** position i dans $testdoc$: mot = testdoc[i] **Si** mot $\in V$: somme[c] = somme[c] + log Prob_wc$[mot, c]$

retourner $\arg\max_c$ somme[c]

[test_naive]

Ainsi, parlons de comment nous pouvons construire un modèle de classification en utilisant Naïve Bayes.

Entraînement du Naïve Bayes (Exemple Pratique)

Considérons le cas de l'analyse de sentiments, sur base des commentaires de gens après avoir suivi un film. Pour raison de simplicité nous considérerons un exemple de quelques phrases et leurs classes correspondantes.

Mode	Classe	Documents
Train	$-$	tout simplement ennuyeux
	$-$	tout à fait prévisible et manque d'énergie
	$-$	pas de surprises et très peu de rires
	$+$	très intéressant
	$+$	le film le plus amusant de l'année
Test	?	prévisible sans amusement

(3.2.30)

[fig:my_label]

Nous commencerons par calculer le **à priori** pour les deux classes comme élaboré dans l'algorithme *[train_naive]* (page **??**).

$$\mathbb{P}(+) = \log \frac{N_+}{N_{doc}} = \log \frac{2}{5} = -0.916290731874155 \tag{3.2.31}$$

$$\mathbb{P}(-) = \log \frac{N_-}{N_{doc}} = \log \frac{3}{5} = -0.5108256237659907 \tag{3.2.32}$$

Comme vous l'avez remarqué, les formules que nous utilisons pour notre algorithme sont dans l'échelle logarithmique, ceci, pour raison d'éviter ce qui est connu en anglais comme underflow (quand les valeurs sont très proches de zéro au point d'être vue par l'ordinateur comme zéro), overflow (quand les valeurs sont très grandes) mais aussi pour augmenter la vitesse de calcul.

La prochaine étape consiste à définir notre vocabulaire V. Le V est un ensemble qui contient les mots uniques de nos données.

V = {‘tout’, ‘simplement’, ‘ennuyeux’, ‘à’, ‘fait’, ‘prévisible’, ‘et’, ‘manque’, ‘d’, ‘énergie’, ‘pas’, ‘de’, ‘surprises’, ‘très’, ‘peu’, ‘rires’, ‘intéressant’, ‘le’, ‘film’, ‘plus’, ‘amusant’, ‘l’, ‘année’}.

Nous avons au total 23 mots uniques. Nous allons par la suite, calculer les valeurs de $classedoc$ pour les deux classes:

- classedoc$[+]$ = [“très intéressant”, “le film le plus amusant de l’année”]
- classedoc$[-]$ = [“tout simplement ennuyeux”, “tout à fait prévisible et manque d’énergie”, “pas de surprises et très peu de rires”].

Calculons ensuite le logarithme de la probabilité (d’apparition) de chaque mot dans notre vocabulaire étant donné une classe particulière. En pratique, pour le calcul de log Prob_wc, il arrive que nous rencontrons de nouveaux mots qui n’étaient pas présents dans l’étape d’entraînement, ceci conduit au fait que log Prob_wc $=$ $\log(0)$ qui n’est pas défini. Alors pour contourner ce problème, plusieurs alternatives existent dans la littérature; dans le contexte de notre exemple, nous allons utiliser la technique appelée “add-one (Laplace) smothing” qui va transformer la formule de log Prob $_wc$ fournie dans l’algorithme *[train_naive]* (page **??**):

$$\begin{aligned} \log \text{Prob}\_wc &= \log \frac{\text{compter}(w,c)+1}{\sum_{w' \in V} (\text{compter}\,(w',c)+1)} \\ &= \log \frac{\text{compter}(w,c)+1}{\sum_{w' \in V} (\text{compter}\,(w',c)) + |V|} \end{aligned} \tag{3.2.33}$$

En réalité comme vous l’avez peut être remarqué, l’algorithme de Naïve Bayes est simplement un comptage systématique des mots.

Notre tâche ici est de classifier la phrase “*prévisible sans amusement*” comme soit positive (+) ou négative (-).

```
Sentiment positive (+)                                                  ␣
↪ Sentiment négative (-)
```

4 | Bibliography

The breakthrough of deep learning origins from (Krizhevsky *et al.*, 2017) for computer vision, there is a rich of following up works, such as (He *et al.*, 2016). NLP is catching up as well, the recent work (Devlin *et al.*, 2018) shows significant improvements.

Two keys together (Devlin *et al.*, 2018, He *et al.*, 2016). Single author , two authors Newell and Rosenbloom (1980)

Bibliography

Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2018). Bert: pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 770–778).

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2017). Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, *60*(6), 84–90.

Newell, A., & Rosenbloom, P. S. (1980). *Mechanisms of skill acquisition and the law of practice.* CARNEGIE-MELLON UNIV PITTSBURGH PA DEPT OF COMPUTER SCIENCE.