INTRODUCTION AU MACHINE LEARNING

2022-2026

Théo Lopès-Quintas

Cadre et approche du cours

Alan Turing publie *Computing Machinery and Intelligence* en 1950 [Tur50], qui deviendra un article fondamental pour l'intelligence artificielle. Une citation devenue célèbre a motivé l'écriture de ce cours :

> *Nous ne pouvons qu'avoir un aperçu du futur, mais cela suffit pour comprendre qu'il y a beaucoup à faire.*
> — Alan Turing (1950)

C'est par cette vision des années 1950 que nous nous proposons de remonter le temps et de découvrir l'ensemble des grandes briques élémentaires du Machine Learning moderne. En partant d'algorithmes difficiles à dater comme la régression linéaire ou logistique, jusqu'aux récentes avancées sur le Gradient Boosting avec CatBoost ou UMAP pour la réduction de dimensions en 2018.
Mais la remarque de Turing reste encore vraie à ce jour ! Le cours ne peut pas couvrir l'ensemble des idées développées en Machine Learning, et ne peut pas prédire l'ensemble des idées à venir.

> *En règle générale, je dirais que l'on n'apprend que dans les cours où l'on travaille sur des problèmes. Il est essentiel que les étudiants tentent de résoudre des problèmes. [...] Se borner à écouter ne sert pas à grand chose.*
> — Werner Heisenberg (1963)

L'objectif premier de ce cours est de former des esprits à naviguer avec les nouvelles idées qui se développeront tout au long de leur vie. La meilleure manière de le faire est de suivre le conseil d'Heisenberg : **essayer**.
C'est pourquoi nous proposons de nombreux exercices et de nombreuses visualisations pour manipuler et créer une intuition visuelle des choses que l'on traite. De même, nous adoptons un ton différent des cours classiques qui s'apparentent plus à une discussion orale afin d'imiter les discussions internes lors d'une recherche.

> *La logique ne fait que sanctionner les conquêtes de l'intuition.*
> — Jacques Hadamard (1972)

Nous ne pouvons donc pas uniquement nous reposer sur un langage moins soutenu et des exemples visuels pour être capable de devenir des artisans du Machine Learning. C'est pourquoi nous ne cacherons pas les difficultés mathématiques abordables et expliquerons autant que nécessaire chacune des formules et les finesses qu'elles contiennent. Apprendre à lire une équation en profondeur renseigne bien plus qu'un long texte.

Ces trois citations ont guidé la construction et la rédaction de ce cours. En résumé, nous souhaitons :

- Apporter une connaissance fine des principaux algorithmes de Machine Learning en expliquant les raisons de leurs développements
- Apprendre à lire une équation mathématique qui formule des problèmes issus de notre intuition
- Délivrer les clés de lecture pour s'émerveiller dans un domaine en plein développement dans les années à venir

Ces trois lignes directrices guident les chapitres présentés, et le dernier point est notamment renforcé par les annexes. Elles ne sont pas obligatoires pour l'examen, mais fortement conseillées pour avoir une vue un peu plus complète du domaine, ainsi qu'un aperçu plus récent des développements en cours.

Table des matières

Chapitre 1

Introduction au Machine Learning

Les termes d'intelligence artificielle (IA) et Machine Learning (ML) sont fréquemment confondu et leur hiérarchie n'est pas toujours clair. Un **algorithme** est une séquence d'instructions logique ordonnée pour répondre explicitement à un problème. Par exemple, une recette de cuisine est un algorithme, mais tous les algorithmes ne sont pas des recettes de cuisine. Un algorithme d'intelligence d'artificielle est un algorithme, mais il n'est pas explicitement construit pour répondre à un problème : il va s'adapter. S'il s'appuie sur des données, alors on parle d'algorithme de Machine Learning [1].
Le terme d'intelligence artificielle vient de la conférence de Dartmouth en 1957 où l'objectif était de copier le fonctionnement des neurones. Mais les concepts d'intelligence artificielle était déjà proposé par Alan Turing, et la méthode des moindres carrés de Legendre (la fameuse tendance linéaire dans Excel) date de bien avant 1957. Depuis, le domaine s'est structuré autour d'une philosophie d'ouverture. Ainsi, nous avons des datasets commun, des algorithmes identiques et des compétitions commune pour pouvoir progresser ensemble.

Nous proposons dans ce chapitre d'introduire les différentes approches du Machine Learning et les grands principes. Pour le rendre aussi général que possible, nous ne discuterons pas d'algorithmes en particulier, mais supposerons que nous en avons un. La description de ces objets sera le coeur des prochains chapitre.

1.1 Les différentes approches du Machine Learning

Quand on parle de Machine Learning, on parle d'un grand ensemble contenant plusieurs approches différentes. Leur point commun est que la donnée est la source de l'apprentissage de paramètres optimaux selon une procédure donnée. Pour saisir les différences entre ces approches, regardons ce dont chacune a besoin pour être suivie.

- **Apprentissage supervisé** : je dispose d'une base de données qui contient une colonne que je souhaite prédire
- **Apprentissage non-supervisé** : je dispose seulement d'une base de données composée d'indicateurs

Ces deux approches représentent l'écrasante majorité des utilisations en entreprise. Se développe également une troisième approche : l'apprentissage par renforcement, qui nécessiterai un cours dédié [2].
Au sein de ces deux grandes approches se trouvent des sous catégories :

- **Apprentissage supervisé** : je dispose d'une base de données qui contient une colonne que je souhaite prédire qui est ...
  - **Régression** : ... une valeur continue

1. Et si la classe d'algorithme est un réseau de neurone, alors on parle de Deep Learning. Ce n'est pas au programme du cours.
2. Elle est au coeur de l'alignement des modèles de langage avec la préférence humaine par exemple.

  - **Classification** : ... une classe, une valeur discrète

- **Apprentissage non-supervisé** : je dispose seulement d'une base de données composée d'indicateurs...
  - **Clustering** : ... et je veux rassembler des observations qui se ressemblent
  - **Réduction de dimension** : ... et je veux réduire la dimension de l'espace engendré par la base de données en perdant le moins d'information possible

Pour saisir les utilisations possibles de ces approches, prenons l'exercice suivant :

Exercice 1.1. *Nous travaillons dans une concession automobile et nous avons à notre disposition une base de données avec l'ensemble des caractéristiques de chaque voiture, chaque ligne de cette base de données étant un modèle de voiture que l'on vend.*
Donner pour chaque demande le type d'approche que l'on peut suivre.

1. *Prédire le type de voiture*
2. *Visualiser en deux dimensions la base de données*
3. *Prédire le prix d'une voiture*
4. *Recommander des voitures à un client se rapprochant de sa voiture de rêve*

On peut également ajouter comme consigne supplémentaire d'imaginer la demande initiale du manager qui a amené le data-scientist à reformuler la demande en ces phrases simples (sauf pour la question 4).

Solution.
1. **Prédire le type de voiture** : apprentissage supervisé - classification. On cherche ici à prédire une classe (un modèle de voiture) en fonction du reste des caractéristiques. La demande initiale du manager pourrait être : quels sont les éléments différentiants qui permettent de dire qu'une voiture est plutôt d'un certain type que d'un autre ? En apprenant à un algorithme à différencier les modèles, on peut étudier les caractéristiques qu'il a utilisées en priorité pour prendre une décision [3].
2. **Visualiser en deux dimensions la base de données** : apprentissage non supervisé - réduction de dimension. On souhaite visualiser une base de données potentiellement en très grande dimension (disons 30 caractéristiques) en seulement 2, mais de manière la plus fidèle possible.
3. **Prédire le prix d'une voiture** : apprentissage supervisé - régression. On prédit cette fois une valeur continue, donc il ne s'agit pas d'une classe. La demande initiale du manager pourrait être : quelles sont les caractéristiques qui font augmenter le prix d'une voiture ? La démarche est proche de l'exemple 1.
4. **Recommander des voitures à un client se rapprochant de sa voiture de rêve** : apprentissage non-supervisé - clustering. En regroupant les voitures qui se ressemblent, nous sommes capables de proposer au client des voitures *proches* de la voiture qu'il recherche.

□

Ces deux grandes approches ne sont bien évidemment pas les seules, et comportent des branches très spécifiques et très développées. Dans un souci de simplicité, nous ne présentons que ces deux approches, mais nous donnerons les clés pour naviguer dans la théorie et la pratique plus profondes de ces branches.

3. À condition que l'algorithme soit performant.

1.2 Apprentissage supervisé - plus formellement

Pour chaque problème de Machine Learning supervisé on dispose d'un dataset $\mathcal{D}$ formé de la manière suivante :

$$\mathcal{D} = \left\{ (x^{(i)}, y_i) \mid \forall i \leqslant \underbrace{n}_{\text{Nombre d'observations}},\ x^{(i)} \in \mathbb{R}^{\overbrace{d}^{\text{Nombre d'informations}}}, y_i \in \mathcal{Y} \right\} \tag{1.1}$$

Avec $\mathcal{Y} \subseteq \mathbb{R}$ s'il s'agit d'une régression et un ensemble fini s'il s'agit d'une classification. Un dataset $\mathcal{D}$ est donc l'ensemble des paires $(x^{(i)}, y_i)$ où $x^{(i)}$ est un vecteur de d informations et $y_i \in \mathcal{Y}$ est un nombre ou une classe d'intérêt associée à l'observation i. On définit les notations :

- X : la matrice des informations définies par : $X = \begin{pmatrix} x_1^{(1)} & x_2^{(1)} & x_3^{(1)} & \cdots & x_d^{(1)} \\ x_1^{(2)} & x_2^{(2)} & x_3^{(2)} & \cdots & x_d^{(2)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x_1^{(n)} & x_2^{(n)} & x_3^{(n)} & \cdots & x_d^{(n)} \end{pmatrix}$
- y : le vecteur réponse composé des $(y_i)_{i \leqslant n} \in \mathcal{Y}^n$

L'hypothèse du data-scientist est qu'il existe une certaine fonction f, inconnue a priori, qui fait le lien entre les observations $x \in \mathbb{R}^d$ et la réponse y. Bien sûr, **aucun modèle n'est parfait**, donc il y a un terme d'erreur incompressible [4] qui est dû à des informations que l'on a pas à disposition par exemple. Formellement, on peut résumer cela à :

$$\exists f : \mathbb{R}^d \to \mathcal{Y},\ \forall i \leqslant n,\ y_i = f\left(x^{(i)}\right) + \varepsilon_i \ \text{ avec } \varepsilon_i \text{ le bruit}$$

De manière évidente, on peut trouver une fonction qui permettrait d'avoir pour un dataset donné $y_i = f(x_i)$ pour toutes les observations, mais ça ne serait vrai que pour le dataset $\mathcal{D}$ que l'on observe. On souhaite que ce soit le cas pour tous les datasets que l'on puisse observer sur cette tâche : on parle de généralisation.

On va donc chercher à approcher f par des formes de fonctions particulières via des procédures particulières que l'on verra en détail plus tard. Chaque forme de fonction est paramétrée par un vecteur $\theta \in \mathbb{R}^{d'}$ (où parfois $d = d'$). Finalement, on cherche la meilleure forme de fonction paramétrée f_θ de la meilleure manière. Mais comment définir *la meilleure* ? On considère deux fonctions :

- La **fonction de perte** $\mathcal{L} : \mathbb{R}^{d'} \times \mathbb{R}^d \times \mathbb{R}$
- La **fonction de coût** $\mathcal{C} : \mathbb{R}^{d'} \times \mathcal{M}_{n,d} \times \mathbb{R}^n$

Avec les ensembles de définitions des fonctions, on saisit que la fonction de perte est associée à un point de la matrice X. Alors que la fonction de coût s'applique à l'ensemble de la matrice X. En pratique, le data-scientist choisit sa fonction de perte, et définit quasiment toujours la fonction de coût comme la somme pour chaque observation de la fonction de perte.
Nous sommes donc en train de décrire un problème d'optimisation, pour lequel on cherche à trouver la meilleure forme de fonction à travers le meilleur paramétrage de son vecteur de paramètre θ en minimisant la fonction de perte associée $\mathcal{C}$. Il est à noter que minimiser la valeur de $\mathcal{C}$ ne veut pas dire qu'on minimise la valeur de $\mathcal{L}$ pour chacun des points séparément : on trouve un juste équilibre global qui nous permet d'atteindre le minimum.
Pour essayer de comprendre ce passage, faisons un exercice :

4. Voir l'équation (2.3).

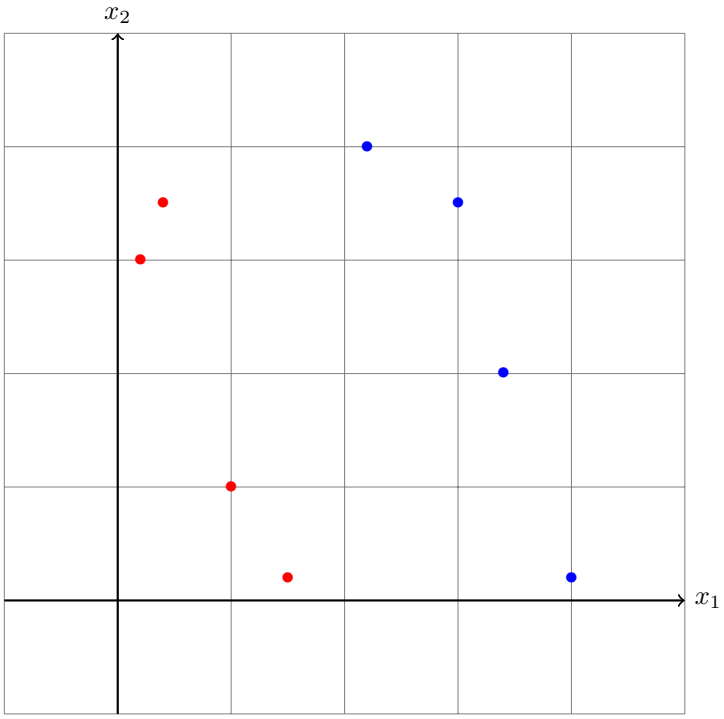


FIGURE 1.1 – Exemple d'un problème de classification avec deux classes et deux indicateurs pour identifier la classe

Exercice 1.2. *À l'aide des données représentée dans la figure (1.1) trouver la meilleure fonction f_θ qui renvoie 0 pour les ronds bleus et 1 pour les ronds rouges parmi les propositions suivantes :*

1. $f_\theta(x_1, x_2) = \mathbb{1}_{x_1 \leqslant \theta}$
2. $f_\theta(x_1, x_2) = \mathbb{1}_{x_2 \leqslant \theta}$

Avec $\mathcal{L}(\theta; x, y) = \mathbb{1}_{y \neq f_\theta(x_1, x_2)}$ *donc* $\mathcal{C}(\theta; X, y) = \sum_{i=1}^{n} \mathcal{L}(\theta, x, y)$.

Solution. Remarquons qu'ici, bien que l'on ait un problème avec $d = 2$ informations, on ne paramètre f_θ qu'avec $d' = 1$ information. Visuellement, on voit que la première proposition est la bonne puisqu'elle permet de classifier parfaitement les points avec $\theta = 2$, la fonction de perte est nulle !
Il y a bien sur d'autres manières de définir la fonction f_θ et même d'autres valeurs de θ sont possibles pour l'exemple que l'on donne. □

La fonction de perte que l'on a proposé ici n'est pas de la *meilleure* forme. En effet, on préférerait avoir à disposition une fonction plus régulière et surtout **convexe**. Ce n'est pas toujours possible, mais on essaye de se mettre dans cette configuration le plus souvent possible : cela nous assure que les procédures d'optimisation que l'on étudiera plus tard vont converger vers la bonne solution. Et on demande une forme de continuité pour tirer parti d'un algorithme fondamental que l'on présentera plus tard. Pour plus de précisions sur la convexité, on peut se référer à la section (A).

1.3 Comment sélectionner les modèles ?

Nous avons présenté jusqu'à maintenant le cadre formel qui nous permettra par la suite de définir tout un ensemble d'algorithmes qui vont *apprendre* à partir d'informations à réaliser une tâche donnée. Nous verrons également comment mesurer les performances de chaque algorithme pour chacune des tâches. On

suppose pour cette partie que l'on dispose d'un ensemble de modèles (algorithme entraîné) et que l'on a sélectionné une métrique de performance. Ce que l'on veut, c'est faire une **sélection de modèles**. Mais quelle stratégie adopter ?

Un hackathon en Machine Learning est une compétition entre data-scientists (ou étudiants) dont le but est de trouver la meilleure manière de répondre à une tâche donnée. Par exemple : étant donné un dataset de prix de l'action Tesla, définir le modèle qui aura la meilleure performance sur un jeu de données non connu à l'avance. Autrement dit, on donne un dataset $\mathcal{D}$ définie comme à l'équation (1.1) avec les réponses pour s'entraîner, et on donne un dataset sans les réponses. L'équipe est censée proposer un modèle, prédire les valeurs associées au dataset sans réponse et les soumettre pour mesurer la performance. Dans notre exemple, l'équipe qui aura prédit les prix les plus proches des prix réels remportera la compétition !

Une équipe dans un hackathon dispose donc d'un jeu d'entraînement et d'un jeu de test. Plus généralement, on peut toujours se placer dans ce cadre-là. L'équipe va essayer plusieurs modèles différents, et cherche à savoir comment sélectionner le meilleur modèle. Ils ne vont pas soumettre plusieurs prédictions, ils ne doivent en soumettre qu'une seule.

1.3.1 Train, Validation, Test

Une des manières les plus classiques pour faire de la sélection de modèles est de séparer en trois parties le dataset $\mathcal{D}$:

1. **Train** : pour apprendre les paramètres optimaux de l'algorithme
2. **Validation** : pour mesurer les performances de l'algorithme sur des données non vues à l'entraînement
3. **Test** : pour soumettre la prédiction

On comprend donc que l'équipe doit scinder son dataset avec les réponses en deux parties : *train* et *validation*. Cette étape est réalisée aléatoirement dans la majorité des cas, mais il faut faire attention à ce que cela n'introduise pas de biais [5].

On peut visualiser la démarche générale avec le schéma (1.2).

Un des inconvénients de cette méthode est qu'elle ne donne que des valeurs de performances et non des intervalles de confiance.

1.3.2 Validation croisée

La validation croisée (ou *cross validation* en anglais) revient à découper à nouveau le dataset d'entraînement pour apprendre plusieurs fois les meilleurs paramètres. C'est de ces multiples apprentissages que nous allons être capables de mesurer précisément l'erreur *normale* que l'on peut attendre d'un modèle. Il en existe plusieurs variantes, commençons par la *K-Fold Cross Validation*.

On découpe le dataset de *train* en K partie égale et on va apprendre K fois les meilleurs paramètres donc mesurer K fois la performance de l'algorithme. La structure d'entraînement est la suivante :

A chaque pli, on extrait du dataset d'entraînement le dataset de test et on apprend sur le reste, puis on mesure les performances sur le dataset de test. Avec cette stratégie, nous sommes capables d'avoir des intervalles de confiance et une mesure plus précise de la performance réelle que l'on peut attendre d'un modèle choisi.

Une seconde manière de faire une *Cross-Validation* est le *Leave-One-Out Cross-Validation* où l'on prend $K = n$ avec n le nombre d'observations. L'intérêt est d'avoir une mesure encore plus précise de la qualité de prédiction du modèle que l'on considère. L'un des inconvénients majeur est que cela peut devenir très long et très coûteux en opération de calcul puisqu'il faut entraîner n fois l'algorithme sur *presque* l'ensemble du dataset. En revanche, cette méthode peut être intéressante dans le cas d'un petit

5. Dans le cadre de prédiction de prix par exemple, il ne faut s'entraîner que sur des données plus anciennes que les données avec lesquelles on mesure notre performance.

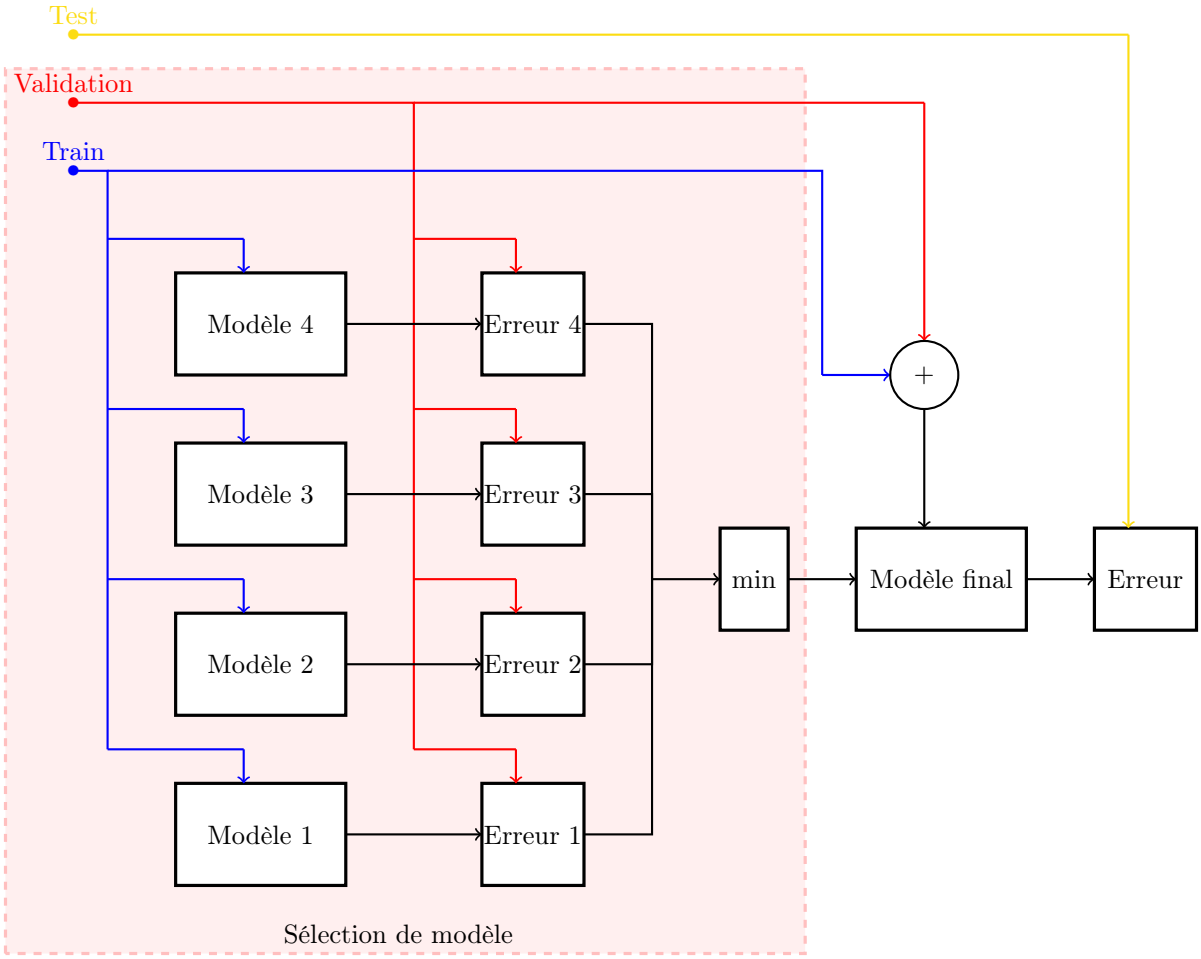


FIGURE 1.2 – Méthodologie Train - Validation - Test

FIGURE 1.3 – Cross-validation avec $K = 4$ plis

dataset.

On peut se demander s'il existe un nombre K optimal pour estimer sans biais l'erreur produite. [BG03] montre qu'un nombre K optimal, pour toutes les distributions et, qui permet d'estimer la variance sans

être biaisé n'existe pas.

> *The main theorem shows that there exists no universal (valid under all distributions) unbiased estimator of the variance of K-fold cross-validation.*
> — Yoshua Bengio, Yves Grandvalet (2003)

Ainsi, il n'y a pas de règles pré-définies qui dictent la valeur de K, c'est au jugement du data-scientist de trancher. Le choix est en pratique surtout dicté par la volumétrie de données à disposition : plus la taille est grande, plus on peut se permettre d'avoir $K = 10$ par exemple.

1.3.3 Occam's Razor

Le rasoir d'Occam (ou Ockham) est au départ un principe philosophique qui a été interprété dans le domaine du Machine Learning. Initialement, le rasoir d'Occam peut se formuler comme *"Les multiples ne doivent pas être utilisés sans nécessité"* ou encore *"Les hypothèses suffisantes les plus simples doivent être préférées"* : c'est un principe de simplicité. Dans notre domaine, [Dom99] note qu'il y a généralement deux interprétations différentes :

> *First razor : Given two models with the same generalization error, the simpler one should be preferred because simplicity is desirable in itself. [...]*
> *Second razor : Given two models with the same training-set error, the simpler one should be preferred because it is likely to have lower generalization error.*
> — Pedro Domingos (1999)

Il dédie cet article et un autre à montrer que la seconde interprétation est fausse, et que la première n'est pas si évidente. Il n'y a aucun consensus fort sur la question. Cependant, **nous recommandons fortement de simplifier au maximum les modèles** par expérience. Cela permet d'avoir des itérations d'évolution du modèle plus rapides, être dépendant de moins de variations dans les données et d'avoir une explicabilité plus rapide. Cet avis est largement discutable, et nous renvoyons à la propre expérience du lecteur pour approfondir la question.

Nous avons présenté dans ce bref chapitre les approches que nous traiterons dans ce cours, et comment nous pourrons comparer et choisir les modèles que nous entraînerons. Notons que la préparation et le traitement de la clé de voûte du Machine Learning, la donnée, n'est pas couverte dans ce document. Il est essentiellement forgé par l'expérience et l'échange avec des pairs. Nous avons également mis à disposition un ensemble de notebook traitant de techniques particulières.

Chapitre 2

Régression linéaire et variantes

La régression linéaire est l'algorithme le plus utilisé au monde de par sa simplicité et ses propriétés mathématiques. Nous présenterons son fonctionnement en détail, ainsi que ses variantes. Nous discuterons également de la manière de mesurer les performances d'un algorithme général répondant à un problème de régression.

La régression linéaire répond à un problème de prédiction d'une valeur continue. On peut imaginer par exemple :

- Prédire le prix d'une action
- Prédire la température qu'il fera dans deux heures
- Prédire l'espérance de vie d'une personne
- Prédire la consommation électrique de la population dans trente minutes
- Prédire la taille adulte d'un enfant

Avec son nom, on comprend que la régression linéaire modélise la fonction f_θ du problème (1.2) comme une combinaison linéaire des indicateurs présents dans la base de données.

Plus formellement, on dispose de n vecteurs qui sont des observations avec d indicateurs et on dispose d'un vecteur avec n coordonnées qui représente les valeurs que l'on cherche à prédire. On peut représenter cela sous la forme condensée suivante :

$$\mathcal{D} = \left\{ (x^{(i)}, y_i) \mid \forall i \leqslant n,\ x^{(i)} \in \mathbb{R}^d, y_i \in \mathbb{R} \right\}$$

2.1 Régression linéaire classique

Le problème de la régression consiste à avoir une prédiction $\hat{y}_i = f_\theta\left(x^{(i)}\right)$ la plus proche de y possible. On se propose donc de résoudre le problème :

$$\theta^* = \underset{\theta \in \mathbb{R}^d}{\arg\min} \sum_{i=1}^{n} \left(y_i - f_\theta\left(x^{(i)}\right) \right)^2 \tag{2.1}$$

Valeur prédite

Vraie valeur

Traduisons le problème. On cherche le vecteur de paramètres θ qui va minimiser la somme des écarts quadratiques entre la valeur que l'on souhaite et la valeur de notre modèle. Ce n'est pas forcément plus clair dit comme cela, alors essayons de le visualiser avec la figure (2.1).

Dans ce graphique, les points bleus représentent notre y et la ligne rouge représente les prédictions pour chaque x possible dans cet intervalle. On a tracé l'écart entre la ligne rouge et le point bleu et on en

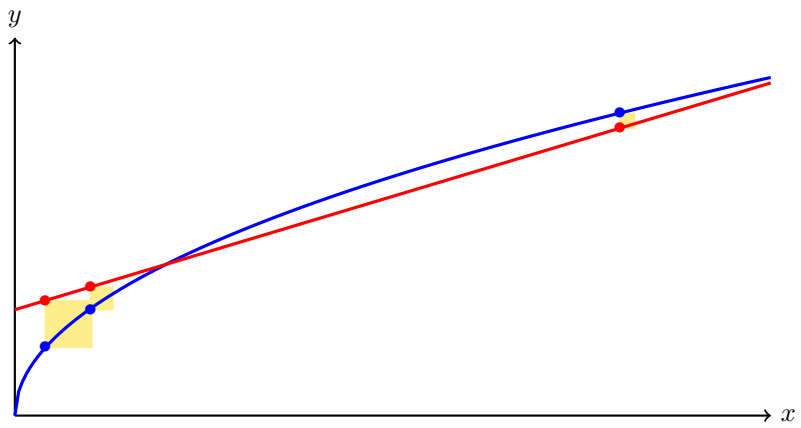


FIGURE 2.1 – Visualisation de la MSE entre la Régression linéaire et vraie courbe

créé un carré. Plus le carré est grand, plus notre erreur est grande.
C'est exactement ce que représente le problème que l'on cherche à résoudre : on veut trouver les paramètres θ qui nous permettent de limiter la surface des carrés d'erreurs.

Pour revenir à la modélisation : on suppose que la relation entre y et x est linéaire. On modélise donc cela par :

$$\hat{y} = \theta_0 + \sum_{j=1}^{d} \theta_j \times x_j \tag{2.2}$$

On peut réécrire notre problème (2.1) comme :

$$\theta^* = \underset{\theta \in \mathbb{R}^{d+1}}{\arg\min} \sum_{i=1}^{n} \left[y_i - \left(\theta_0 + \sum_{j=1}^{d} \theta_j \times x_j^{(i)} \right) \right]^2$$

Pour mieux comprendre et manipuler ces équations, on se propose de résoudre l'exercice suivant.

Exercice 2.1 (Régression linéaire avec une seule information)**.** *On suppose que l'on dispose d'un dataset* $\mathcal{D} = \left\{ (x^{(i)}, y_i) \mid \forall i \leqslant n : x^{(i)} \in \mathbb{R}, y_i \in \mathbb{R} \right\}$*. On a donc une seule information pour prédire la valeur* y*.*

1. *Écrire le problème (2.1) dans le cadre de l'exercice.*
2. *Donner le meilleur vecteur de paramètre* θ*.*

   On note $\overline{x} = \dfrac{1}{n} \sum_{i=1}^{n} x^{(i)}$*. On rappelle avec cette convention que pour* $u, v \in \mathbb{R}^n$ *:*

$$\begin{aligned} \mathrm{Cov}(u, v) &= \overline{uv} - \overline{u} \times \overline{v} \\ \mathbb{V}[u] &= \overline{u^2} - \overline{u}^2 \end{aligned}$$

3. *Montrer que* θ_0^* *et* θ_1^* *les deux paramètres optimaux peuvent s'écrire :*

$$\begin{aligned} \theta_0^* &= \overline{y} + \theta_1^* \times \overline{x} \\ \theta_1^* &= \frac{\mathrm{Cov}(x, y)}{\mathbb{V}[x]} \end{aligned}$$

En finance, le coefficient θ_1 calculé ici est une formule connue, c'est le β d'un stock ! Lorsque l'on considère un stock, on le compare au marché, et on cherche à trouver une relation entre le return d'un stock et celui du marché :

$$r_{stock} = \alpha + \beta r_{market}$$

C'est une régression linéaire. On a donc une forme fermée simple pour le cas où l'on a une seule information pour prédire y. On aimerait en être capable pour n'importe quel nombre d'informations. Pour cela, nous allons reformuler le problème (2.1) de manière matricielle.

On note Y le vecteur de longueur n qui est formé de l'ensemble des $(y_i)_{i\leqslant n}$ du dataset $\mathcal{D}$. On note X la matrice formée par l'ensemble des informations que l'on a. Chaque ligne correspond à une observation :

$$X = \begin{pmatrix} x_1^{(1)} & x_2^{(1)} & x_3^{(1)} & \cdots & x_d^{(1)} \\ x_1^{(2)} & x_2^{(2)} & x_3^{(2)} & \cdots & x_d^{(2)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x_1^{(n)} & x_2^{(n)} & x_3^{(n)} & \cdots & x_d^{(n)} \end{pmatrix}$$

De manière classique, la première colonne est remplacée par un vecteur de valeur 1 : c'est pour modéliser ensuite le paramètre θ_0. Ainsi dans la suite, par rapport à la première écriture on considère toujours que l'on a d informations mais intercept compris cette fois. La modélisation s'écrit maintenant comme :

$$\begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix} = \begin{pmatrix} x_1^{(1)} & x_2^{(1)} & x_3^{(1)} & \cdots & x_d^{(1)} \\ x_1^{(2)} & x_2^{(2)} & x_3^{(2)} & \cdots & x_d^{(2)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x_1^{(n)} & x_2^{(n)} & x_3^{(n)} & \cdots & x_d^{(n)} \end{pmatrix} \times \begin{pmatrix} \theta_1 \\ \theta_2 \\ \vdots \\ \theta_d \end{pmatrix} + \begin{pmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \vdots \\ \varepsilon_n \end{pmatrix}$$

$$\iff$$

$$Y = X\theta + \varepsilon \text{ , avec } \varepsilon \text{ un vecteur de bruit.}$$

On rappelle que la norme d'un vecteur $u \in \mathbb{R}^d$ se définit comme : $\|u\| = \sqrt{\sum_{i=1}^{d} u_i^2}$. Ainsi, le problème que l'on cherche à résoudre est :

$$\theta^* = \underset{\theta \in \mathbb{R}^d}{\arg\min} \|Y - X\theta\|^2$$

Avec des mathématiques qui exploitent la notion de projection orthogonale et des résultats d'algèbre linéaire, on établit le résultat suivant.

Proposition 1. *Si la matrice X est de rang plein, alors :*

$$\theta^* = ({}^tXX)^{-1}\,{}^tXY$$

La notion de rang plein en algèbre linéaire a un sens très précis, et nous allons le simplifier ici. Dire que la matrice X est de rang plein signifie que le nombre de lignes est supérieur au nombre de colonnes, et qu'il n'y a aucune colonne qui est formée comme combinaison linéaire des autres. Avant de commenter le résultat, on peut vérifier la cohérence de la formule avec un exercice.

Exercice 2.2. *Vérifier à l'aide des dimensions que la formule est cohérente.*

Solution. Dans un premier temps : $X \in \mathcal{M}(n,d) \implies ({}^tXX) \in \mathcal{M}(d,d) \implies ({}^tXX)^{-1} \in \mathcal{M}(d,d)$. Pour la culture, le fait que la matrice $({}^tXX)$ soit inversible vient du fait que la matrice X est de rang plein. Dans un deuxième temps : $X \in \mathcal{M}(n,d) \wedge Y \in \mathbb{R}^n \implies {}^tXY \in \mathbb{R}^d$. Finalement, $({}^tXX)^{-1} \in \mathcal{M}(d,d) \wedge {}^tXY \in \mathbb{R}^d \implies ({}^tXX)^{-1}\,{}^tXY \in \mathbb{R}^d$ ce qui est cohérent car on souhaite obtenir un vecteur représentant les d paramètres de la régression linéaire. □

Ce résultat revient à dire que pour n'importe quel problème de régression linéaire bien posé, il existe une formule exacte pour calculer les paramètres optimaux. Il n'y a aucune hypothèse supplémentaire que d'avoir plus d'observations que d'indicateurs et que les indicateurs ne soient pas la somme les uns des autres.

2.2 Mesurer la performance d'une régression

Maintenant que l'on sait comment exploiter au mieux une régression linéaire, nous devons être capable de mesurer autrement que visuellement la qualité de prédiction de notre algorithme. On adresse ici un problème plus large que celui de la mesure de performance de la régression linéaire : la mesure de performance d'une régression en général.

2.2.1 Erreur quadratique moyenne

On suppose dans toute la section que l'on dispose d'un dataset $\mathcal{D}$ défini comme dans (1.1). La première idée que l'on peut avoir est de s'inspirer du problème (2.1) et définir la *Mean Squared Error* (MSE) :

$$\mathrm{MSE}(y,\hat{y}) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

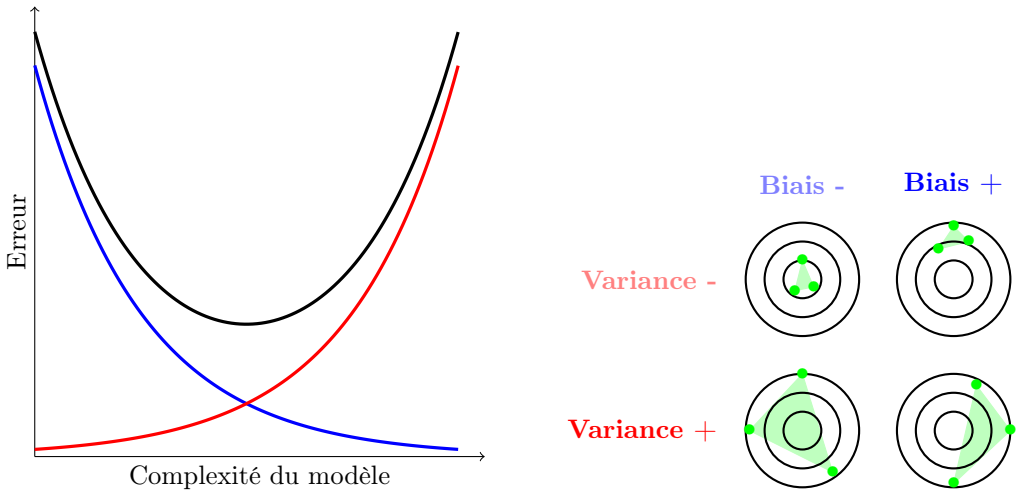
L'intérêt de la MSE est qu'elle est convexe [1] donc pratique pour les problèmes d'optimisation. Mais elle est peu interprétable telle qu'elle. Elle a toutefois un intérêt pédagogique : on peut facilement introduire le problème biais-variance du data-scientist.

Prenons un cadre général, on cherche une fonction $\hat{f}(x)$ qui dépend donc du dataset $\mathcal{D}$ que l'on utilise pour apprendre la fonction f. On note :

- $\mathrm{Bias}\left[\hat{f}(x)\right] = \mathbb{E}\left[\hat{f}(x)\right] - f(x)$: l'écart moyen entre la valeur prédite et la vraie valeur
- $\mathbb{V}\left[\hat{f}(x)\right] = \mathbb{E}\left[\left(\mathbb{E}\left[\hat{f}(x)\right] - \hat{f}(x)\right)\right)^2\right]$: la dispersion moyenne des valeurs prédites autour de la moyenne

Le biais mesure notre proximité à la fonction réelle, donc à quel point on *l'apprend*. Naturellement, plus on a un modèle complexe plus le biais est faible. Inversement, la variance qui mesure à quel point le modèle s'éloigne fréquemment de sa valeur moyenne, va avoir tendance à augmenter avec la complexité du modèle. On peut comprendre le biais comme la mesure des efforts de simplifications des hypothèses du modèle.

1. Voir annexe (A).



(a) Simplifié (b) Visualisation

FIGURE 2.2 – Illustration du trade-off biais-variance

On retrouve ici l'intuition que l'on a développée avec les exemples vus précédemment. On peut formaliser cette intuition avec l'équation (2.3).

$$\text{MSE}(y, \hat{f}(x)) = \left(\text{Bias}\left[\hat{f}(x)\right]\right)^2 + \mathbb{V}\left[\hat{f}(x)\right] + \underbrace{\sigma^2}_{\text{Erreur incompressible}} \tag{2.3}$$

La décomposition de la MSE avec le biais et la variance explicite, au passage, une erreur minimale incompressible pour le test. En effet, en créant un modèle qui interpole l'ensemble des points, alors on obtient une valeur de MSE nulle, mais il aura perdu la capacité de généralisation. C'est ce qui est exprimé en terme statistique dans l'équation précédente.

Nous devons donc trouver une alternative plus interprétable que la MSE mais avec la même idée sous-jacente. Et si on prenait simplement la racine carrée ?

$$\text{RMSE}(y, \hat{y}) = \sqrt{\sum_{i=1}^{n} \frac{1}{n} (y_i - \hat{y}_i)^2}$$

Exercice 2.3 (Ordre de grandeur)**.** *Montrer que :*

$$\text{RMSE}(y, \overline{y}) = \overline{y^2} - \overline{y}^2$$

En déduire une interprétation de la RMSE et un critère de performance d'une régression.

Le principal intérêt de la RMSE est qu'elle mesure l'écart-type de l'erreur que l'on commet avec notre prédiction, donc notre algorithme. Si on l'analyse en terme d'unité, alors on est dans les mêmes unités de mesure que le vecteur y.

2.2.2 Coefficient de détermination R^2

Une autre manière de mesurer la performance est de regarder ce que l'on *explique*. On définit :

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \overline{y})^2}$$

Prédiction du modèle

Prédiction moyenne

Il s'agit du coefficient de détermination. Il permet de mesurer notre capacité à bien expliquer les données, du moins mieux que ce qu'arrive à le faire un modèle *bête* qui va prédire systématiquement la valeur moyenne. La vision est donc bien complémentaire à celle de la RMSE.

Exercice 2.4. *On suppose que l'on dispose des vecteurs y et $\hat{y}$.*

1. *Comment interpréter la valeur 1 pour le R^2 ? Et la valeur 0 ?*
2. *Le R^2 peut-il être négatif ?*

Solution. 1. Si $R^2 = 1 \iff \sum_{i=1}^{n}(y_i - \hat{y}_i)^2 = 0 \iff \forall i \leqslant n,\ \hat{y}_i = y_i$ donc qu'on classifie parfaitement.

Si $R^2 = 0 \iff \sum_{i=1}^{n}(y_i - \hat{y}_i)^2 = \sum_{i=1}^{n}(y_i - \overline{y})^2$ donc la qualité de la prédiction est la même que la prédiction moyenne. A noter qu'à la différence du point précédent, on ne peut pas dire que $\forall i \leqslant n,\ \hat{y}_i = \overline{y}$.

2. Oui, il suffit que la prédiction de l'algorithme soit moins bonne que celle de la prédiction moyenne. Ce n'est pas censé arriver, mais c'est théoriquement possible, et on peut construire facilement des exemples.

□

2.3 Régressions pénalisées

Une régression linéaire n'est possible que lorsque la matrice X est de rang plein. Or il est fréquent en médecine par exemple que le nombre d'informations mesurées soit supérieur au nombre d'observations ($d >> n$). Avec l'essor technologique qui permet le Big Data, il est fréquent de rencontrer des problèmes de régression avec beaucoup plus d'indicateurs que d'observations. Quelques cas typiques :

- **La génétique** : le génome humain est d'une incroyable complexité, et il n'y a pas énormément d'observations.
- **La finance de marché** : on peut vouloir prédire un prix d'une option en considérant l'ensemble des instruments financiers du marché, et leurs évolutions.
- **L'image** : apprendre à un algorithme à estimer la taille d'un objet nécessite énormément de photos différentes car chacune des photos contient un très grand nombre de pixels. Le phénomène est encore pire pour la vidéo !

Pour répondre à cette problématique, on modifie le problème (2.1) en ajoutant des contraintes sur le vecteur de paramètre θ :

$$\theta^* = \arg\min_{\theta \in \mathbb{R}^d} \sum_{i=1}^{n} \left(y_i - f_\theta\left(x^{(i)}\right)\right)^2 + \mathcal{P}_\lambda(\theta) \tag{2.4}$$

Apprentissage

Pénalisation

L'idée est de contrôler la complexité du modèle via une pénalisation $\mathcal{P}$ qui dépend du paramètre λ. C'est un vecteur de nombres qui dépend du nombre de conditions que l'on impose. L'objectif est donc d'avoir la possibilité de limiter le sur-apprentissage en simplifiant le modèle.

2.3.1 Régression Ridge

La régression Ridge [Tib96, Tib11] est définie par le problème où l'on choisit un paramètre $\lambda \in [0, +\infty[$ et une pénalité :

$$\theta^*_{\text{Ridge}} = \underset{\theta \in \mathbb{R}^d}{\arg\min} \|Y - X\theta\|^2 + \lambda \|\theta\|^2 \tag{Ridge}$$

Cette pénalité est appelée la pénalité L_2 en référence à la norme euclidienne utilisée. On comprend que λ a pour rôle de contrôler à quel point on souhaite avoir un modèle avec des coefficients proches de zéros.

Exercice 2.5. *Quelle est la solution du problème (Ridge) pour $\lambda = 0$? Même question pour $\lambda \to +\infty$.*

Solution. Pour $\lambda = 0$ le problème est identique à celui d'une régression linéaire sans pénalisation. Inversement, plus λ tend vers $+\infty$, plus les coefficients tendent vers 0, en valant exactement 0 si λ pouvait prendre la valeur $+\infty$. □

De la même manière que pour la régression linéaire, la régression Ridge dispose d'une solution avec une forme fermée :

$$\theta^*_{\text{Ridge}} = ({}^tXX + n\lambda \mathbb{I}_d)^{-1}\, {}^tXY$$

On peut remarquer que la solution est très proche de celle de la régression linéaire sans pénalisation. Rappelons que pour la régression linéaire, nous avions besoin que la matrice X soit de rang plein, ici ce n'est pas le cas ! La pénalisation L_2 nous permet d'avoir systématiquement un inverse quand $\lambda > 0$. Pour le prouver, il faut considérer les notions fondamentales d'algèbre qui impliquent les valeurs propres et les vecteurs propres. Nous ne les aborderons pas ici, mais dans le chapitre dédié à la réduction de dimension (chapitre 7).

Exercice 2.6. *Vérifier à l'aide des dimensions que la formule pour la régression Ridge est cohérente.*

Solution. Même démarche que précédemment en utilisant que $n\lambda \mathbb{I}_d \in \mathcal{M}_{d,d}$ par définition. □

2.3.2 Régression LASSO

La régression LASSO (*Least Absolute Selection and Shrinkage Operator*) [Tib96, Tib11] est définie par le problème :

$$\theta^*_{\text{LASSO}} = \underset{\theta \in \mathbb{R}^d}{\arg\min} \|Y - X\theta\|^2 + \lambda \|\theta\|_1 \tag{LASSO}$$

Avec $\|\theta\|_1 = \sum_{j=1}^{d} |\theta_j|$. Ici la pénalité est appelée la pénalité L_1. Cette fois, on ne peut pas obtenir de formule fermée pour tous les cas de ce problème. Si on impose comme condition que ${}^tXX = \mathbb{I}_d$, alors on peut obtenir une forme fermée pour ce problème :

$$\forall j \leqslant d, \quad \theta^*_{\text{LASSO},j} = \theta^*_j \times \max\left\{0, 1 - \frac{n\lambda}{|\theta^*_j|}\right\}$$

Régression linéaire

Avec ce problème-là, on comprend que les coefficients d'apprentissage vont converger plus vite vers 0 que pour la régression Ridge. En pratique, il est extrêmement rare d'obtenir la condition ${}^tXX = \mathbb{I}_d$, mais la conclusion que les coefficients appris convergent plus rapidement vers 0 reste vrai.

Quel est l'intérêt d'ajouter cette pénalité en terme de biais et variance ?

Exercice 2.7 (Biais/Variance pour Ridge et LASSO). *Pour la régression Ridge, puis la régression LASSO, comment évolue le biais quand λ augmente ? Même question pour la variance.*

Solution.
- Régression Ridge : quand λ est faible, on ne fait pas tendre fortement les coefficients vers zéro, donc on n'introduit pas beaucoup de biais mais on laisse la variance forte. Inversement, quand λ est fort, les coefficients tendent plus vite vers zéro donc un biais est fort mais une variance réduite.
- Régression LASSO : même explication que pour Ridge, avec la différence que la vitesse de convergence vers 0 des coefficients est plus rapide ici.

□

On comprend donc que l'intérêt de ces deux méthodes est de permettre au data scientist d'avoir la main sur une manière de limiter la variance du modèle.

2.3.3 Régression ElasticNet

ElasticNet [ZH05] correspond à une combinaison entre Ridge et LASSO :

$$\theta^*_{\text{ElasticNet}} = \underset{\theta\in\mathbb{R}^d}{\arg\min} \|Y - X\theta\|^2 + \lambda\|\theta\|_1 + \mu\|\theta\|^2 \qquad \text{(ElasticNet)}$$

La manière de sélectionner les meilleurs paramètres pour l'ElasticNet, Ridge ou LASSO est une question en soi ! On peut tout à fait traiter le problème avec une validation croisée par exemple.

Un des inconvénients des régressions pénalisées par une norme L_1 est la lenteur de calcul puisqu'il n'existe pas de forme fermée comme pour la régression linéaire classique ou la régression Ridge.

2.4 Régressions polynomiales : approximateurs universels

Le grand défaut de la régression linéaire, pénalisée ou non, est qu'elle suppose que la relation entre la variable que l'on veut prédire et les variables explicatives, est linéaire. Or ce n'est pas toujours le cas ! Prenons un exemple :

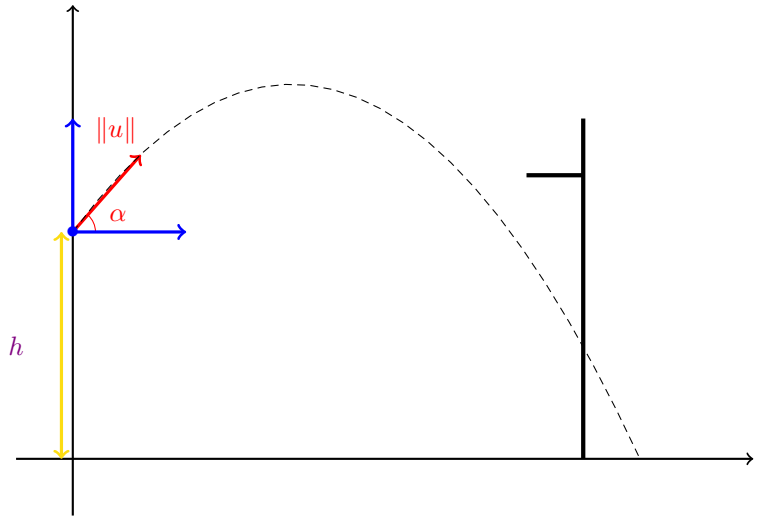


FIGURE 2.3 – Simulation d'un mini-jeu de basketball

Nous connaissons la trajectoire de la balle, le paramètre h et l'angle α, mais nous ne connaissons pas la vitesse de lancement $\|u\|$. Avec la physique Newtonienne, nous sommes capable d'établir que :

$$y = -\frac{g}{2\|u\|^2 \cos^2(\alpha)} x^2 + \|u\| \tan(\alpha) x + h$$

On voit que la relation entre y et x n'est clairement pas linéaire. Donc on ne pourra pas prédire parfaitement la position de la balle pour chaque x. Donc on ne sera pas capable d'apprendre les meilleurs paramètres pour finalement trouver la vitesse de lancement $\|u\|$. Mais si au lieu de donner seulement l'information de x, on donne également l'information de x^2 alors on pourra répondre au problème : c'est une régression polynomiale.
En effet, nous aurons 3 coefficients alors que l'on dispose d'une seule information (x) et de l'intercept, et chacun correspondra exactement à l'équation physique du mouvement.

$$y = \theta_0 + \theta_1 x + \theta_2 x^2$$

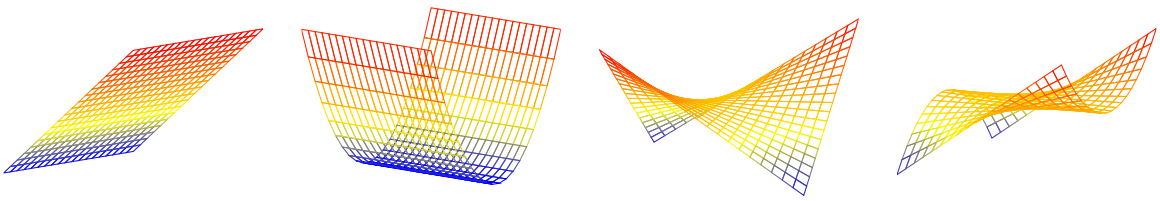
Une régression polynomiale est une régression linéaire où l'on va considérer les informations initiales, les informations initiales mise à plusieurs puissances mais également les interactions entre les informations.

Exercice 2.8. *Étant donné une matrice de données X, écrire une fonction* polynomial_features *qui prend en paramètres :*

- *degree : la puissance maximale que l'on autorise*
- *combinaison : les interactions entre features*

Et qui renvoie une nouvelle matrice de données avec les informations polynomiales.

Voyons des exemples de surfaces que l'on se permet d'apprendre avec une telle méthode :



(a) $f(x_1, x_2) = x_1$ (b) $f(x_1, x_2) = x_2^2$ (c) $f(x_1, x_2) = x_1 x_2$ (d) $f(x_1, x_2) = x_1 x_2^2$

FIGURE 2.4 – Exemple de surfaces pour deux features avec des polynômes de degré au plus 3

Comme on a décidé d'exploiter les informations polynomiales avec interactions avec degré 3, on voit que le degré maximal est 3 dans (2.4d) x_1 est de degré 1 et x_2 de degré 2. On apprend des formes de fonctions bien plus complexes. Et dans ce cas, on écrit donc le problème comme :

$$y = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_1 x_2 + \theta_5 x_2^2 + \theta_6 x_1^3 + \theta_7 x_1^2 x_2 + \theta_8 x_1 x_2^2 + \theta_9 x_2^3$$

En faisant ça, on exploite l'idée d'approximation polynomiale du théorème de Weierstrass en 1885 :

Théorème 1 (Weierstrass Approximation Theorem)**.** *Soit f une fonction continue sur un intervalle fermé à valeur réelle. Alors il existe une suite de polynômes qui converge uniformément vers $f(x)$ pour $x \in [a, b]$.*

C'est une idée mathématique fondamentale qui stipule qu'un objet *compliqué* peut être décrit comme une combinaison d'objets plus simples. On appelle cela *universal approximation* et les objets simples

des *universal approximators*. Ainsi, plus formellement, on peut avec assez d'approximateurs universels approcher aussi finement que l'on souhaite toute fonction continue par morceaux comme combinaison linéaire des objets simples.

On comprend qu'avec une régression polynomiale, on se rapproche de l'approximation universelle, et on appelle cette famille la famille des noyaux $\left(f_i(x) = x^i\right)_{i\in\mathbb{N}}$. Il peut donc être tentant de donner toujours beaucoup plus de puissance à son modèle avec des features polynomiales.
On ne recommande pas en pratique d'aller au-delà du degré trois (sauf dans certain cas précis) pour les raisons suivantes :

- **La variance augmente** : on peut se retrouver dans un cas où la variance sera beaucoup plus forte
- **La généralisation baisse** : il est fortement probable d'entrer dans un régime d'interpolation des données d'apprentissage et donc de ne pas réussir à généraliser correctement
- **La dimension augmente** : en augmentant le nombre d'informations, on incrémente le nombre de dimensions dans lesquelles on travaille. Cela impacte également les temps de calculs

Le dernier point est plus longuement discuté dans l'annexe (D) qui traite du fléau de la dimension. En résumé : plus la dimension d'un espace augmente, plus les données se concentrent et la notion de distance perd exponentiellement vite son sens. Ce que cela veut dire en pratique est que les algorithmes auront du mal à apprendre les données en y donnant du sens.

Ceci conclut la présentation d'un des algorithmes les plus utilisés dans le monde entier. Nous sommes à présent capables de traiter avec une puissance raisonnable n'importe quel problème de régression.

Chapitre 3

Régression Logistique

A la fin du XVII^{e} siècle, les nombres complexes sont connus et exploités. Leonhard Euler donne une définition de la fonction exponentielle complexe et, connaissant la complémentarité entre la fonction exponentielle et logarithme, les mathématiciens cherchent à définir une fonction logarithme complexe. Au delà de cette complétude, on cherche à définir un logarithme sur l'ensemble des réels, et pas uniquement une partie. En 1702, Jean Bernoulli en exploitant le calcul intégral obtient une première expression d'un logarithme complexe. Ce résultat soulève de nombreuses contradictions analytiques.

> *Quoique la doctrine des logarithmes soit si solidement établie, que les vérités qu'elle renferme semblent aussi rigoureusement démontrées que celles de la Géométrie ; les Mathématiciens sont pourtant encore fort partagés sur la nature des logarithmes négatifs et imaginaires.*
> — Leonhard Euler (1749)

Les débats et recherches continueront jusqu'à la publication de 1749 d'Euler en exhibant la première fonction multiforme pour définir le logarithme complexe.

> *Il y a toujours une infinité de logarithmes qui conviennent également à chaque nombre proposé. Ou, si y marque le logarithme du nombre x, je dis que y renferme une infinité de valeurs différentes.*
> — Leonhard Euler (1749)

A travers cette fabuleuse recherche de complétude (habituelle en mathématiques) nous avons découvert de nouvelles approches et objets qui posent encore question.

Nous nous attelons ici à obtenir une version analogue la régression linéaire qui nous permette de répondre à une tâche de classification. De même, nous serons amenés à définir des notions et approches fertiles que nous exploiterons dans plusieurs chapitres. Toujours à l'image de l'exponentielle et du logarithme, la régression linéaire et la régression logistique sont les bases fondamentales du Machine Learning.

3.1 Modélisation

Considérons un dataset de classification binaire :

$$\mathcal{D} = \left\{ (x^{(i)}, y_i) \mid \forall i \leqslant n,\ x^{(i)} \in \mathbb{R}^d, y_i \in \{0, 1\} \right\}$$

L'ensemble des valeurs pour les $(y_i)_{i \leqslant n}$ est discret, contrairement à la régression linéaire dans laquelle les valeurs étaient continues. Cette différence fait que l'on ne peut pas considérer les mêmes méthodes : rien ne garantit qu'avec une régression linéaire on puisse correctement apprendre les données de ce dataset. En effet, rien ne permet de borner la prédiction entre 0 et 1, on doit donc choisir une autre modélisation. On va plutôt vouloir modéliser le problème pour prédire la probabilité d'appartenance à la classe 1 (par exemple). Pour le moment, on ne sait rien faire d'autre qu'une régression linéaire. Il faudrait donc que l'on puisse transformer l'estimation de la probabilité $\mathbb{P}(x)$ que l'observation x soit de la classe 1, à un

problème que l'on peut résoudre avec une régression linéaire.

Une manière de le faire est de considérer la cote, comme utilisé dans les paris sportifs par exemple, et modéliser cela comme une régression linéaire :

$$\ln\left(\frac{\mathbb{P}(x)}{1-\mathbb{P}(x)}\right) = \sum_{i=1}^{d} \theta_i x^{(i)} \tag{3.1}$$

On modélise donc le logarithme de la cote comme une fonction linéaire des informations dont on dispose. Ainsi, on a :

$$\mathbb{P}(x) = \frac{1}{1+\exp\left\{-\sum_{i=1}^{d} \theta_i x^{(i)}\right\}} \tag{3.2}$$

On reconnaît la fonction sigmoid : $\sigma(x) = \dfrac{1}{1+e^{-x}}$ qui a cette forme caractéristique de S comme sur la figure (3.1).

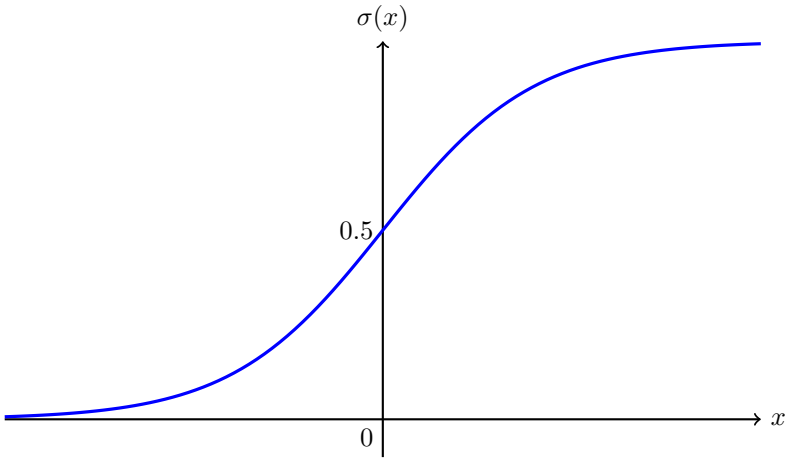


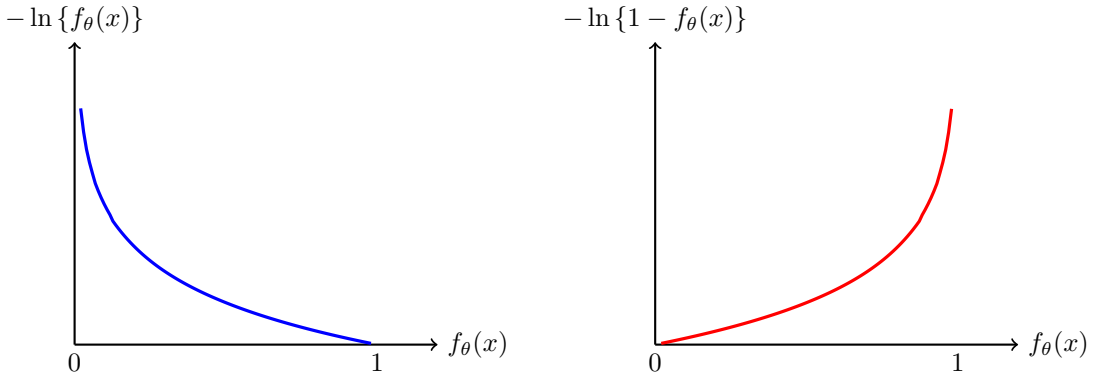
FIGURE 3.1 – Fonction sigmoid σ ou fonction logistique

On comprend avec l'équation (3.1) l'interprétation que l'on peut faire des valeurs des coefficients. Si l'on augmente de 1 la valeur de l'information x_j, on augmente le logarithme de la côte par θ_j. C'est équivalent à dire qu'on augmente la côte par e^{θ_j}. Ainsi, avoir un coefficient θ_j positif augmente la probabilité de prédire 1 quand la variable j augmente, lorsque que les autres variables sont identiques. Inversement pour un coefficient négatif. Attention, la probabilité n'augmente pas linéairement avec θ_j, comme en témoigne l'équation (3.2).

On souhaite maintenant apprendre les meilleurs coefficients. Nous devons donc chercher un problème sous la forme d'une fonction de perte à minimiser, avec comme condition de trouver une fonction de perte qui soit convexe. Nous proposons la suivante :

$$\mathcal{L}(\theta; x, y) = -\Big[\underbrace{y \ln\{f_\theta(x)\}}_{\text{Observation positive}} + \underbrace{(1-y)\ln\{1-f_\theta(x)\}}_{\text{Observation négative}}\Big]$$

Décortiquons cette fonction de perte. Il y a deux parties : une qui traite les observations positives (*i.e* quand $y = 1$) et une seconde qui traite les observations négatives (*i.e* quand $y = 0$). Quand $y = 1$, on souhaite que l'estimateur de la probabilité que $y = 1$ soit le plus proche possible de 1, donc on veut modifier la valeur de θ si ce n'est pas le cas et la conserver si c'est le cas. Les deux fonctions $x \mapsto -\ln\{x\}$ et $x \mapsto -\ln\{1-x\}$ le font parfaitement :



De plus, on peut montrer que ces deux fonctions sont convexes. Ainsi, nous savons qu'il existe une unique solution au problème suivant :

$$\theta^* = \underset{\theta \in \mathbb{R}^d}{\arg\min} \sum_{i=1}^{n} \mathcal{L}\left(\theta; x^{(i)}, y_i\right)$$

Cependant, nous ne pouvons pas résoudre *à la main* ce problème. Il nous faut une autre approche.

3.2 Descente de gradient

La descente de gradient est une méthode d'optimisation numérique qui s'applique dans de très nombreux domaines. Pour appliquer cette méthode, il faut un problème qui puisse s'écrire sous la forme suivante, avec une fonction f différentiable :

$$x^* = \underset{x \in \mathbb{R}^d}{\arg\min}\, f(x)$$

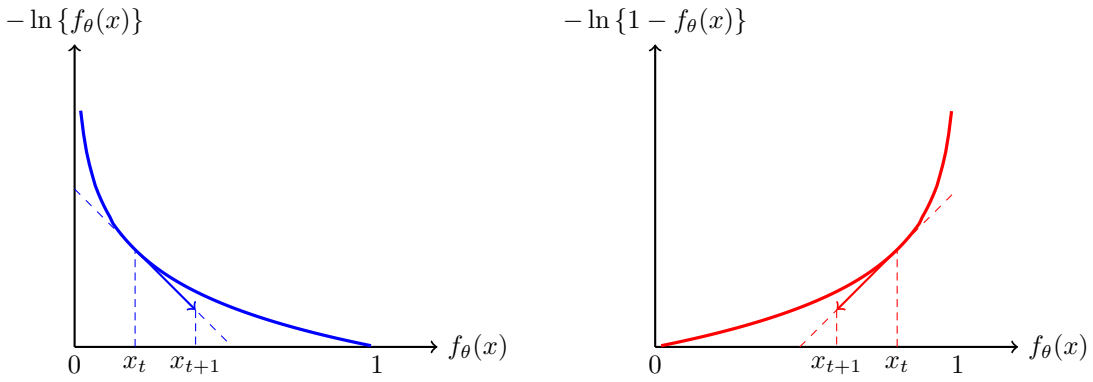
Cela est identique aux différents problèmes que nous avons rencontrés jusqu'ici ! La méthode de descente de gradient est une méthode itérative qui va approcher la solution en appliquant la suite :

$$x_{t+1} = x_t - \eta \nabla f(x_t) \quad , \eta > 0 \tag{3.3}$$

A chaque itération, nous allons nous déplacer dans la direction opposée à la valeur du gradient pour tendre vers la limite de cette suite : le minimum de la fonction d'intérêt.

À chaque itérations, on *descend* le gradient et l'on s'approche ainsi du minimum de la fonction. Si l'on choisit x_{t+1} comme le point où la tangente croise $y = 0$, alors on obtient l'algorithme de Newton-Raphson qui est également un algorithme d'optimisation. Il est différent de la descente de gradient, car la descente de gradient ne cherche pas à aller *jusqu'à* $y = 0$: il va dans cette direction mais parcourt $\eta \nabla f(x_t)$ dans cette direction.

Exercice 3.1. *Écrire en Python l'algorithme de descente de gradient. La fonction prendra en paramètre la fonction* `f`*, sa dérivée* `df` *et l'ensemble des paramètres que vous jugerez utiles.*



En deux dimensions, on ne traite plus d'une courbe dont on cherche le minimum comme nous le voyions dans les premiers exemples mais d'une surface. Dans la figure (3.2) on voit comment la descente se comporte en plaçant un point à chaque itération.

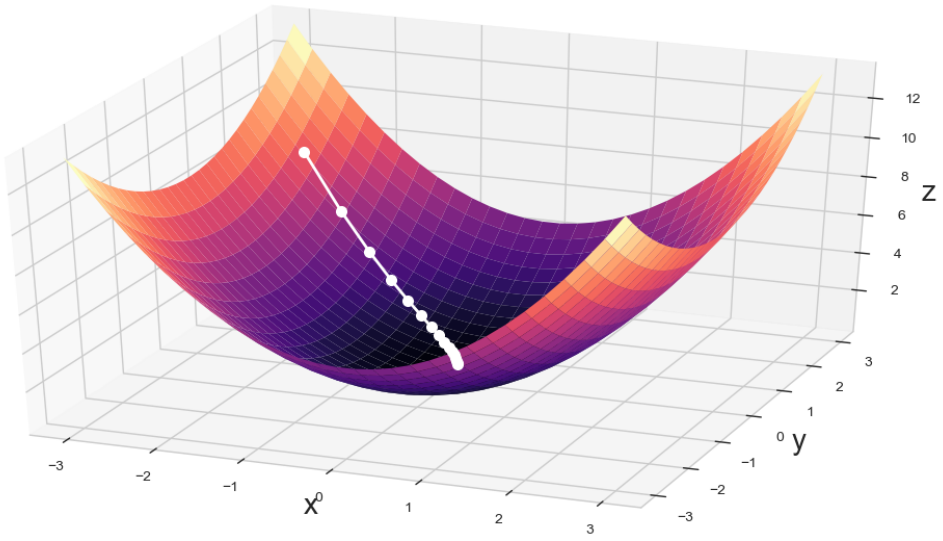


FIGURE 3.2 – Exemple d'une descente de gradient pour la fonction $f(x,y) = x^2 + \frac{1}{2}y^2$ avec $\eta = 0.1$

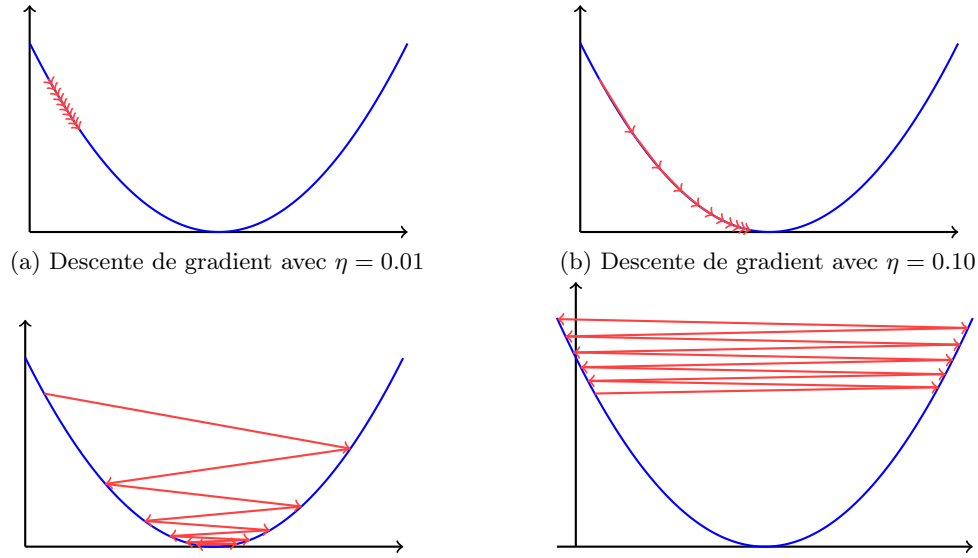
Le paramètre η est crucial dans la descente de gradient, c'est le *learning rate*. Son rôle est de contrôler l'amplitude des pas de descente. Toujours dans la figure en deux dimensions, on remarque que les points sont de plus en plus rapprochés. La descente de gradient est de moins en moins brusque, et on modifie que très peu le paramètre dans les dernières itérations. Voyons sur l'exemple (**??**) en une seule dimension comment se comporte la descente de gradient en fonction de la valeur de η.

Le learning rate contrôle la vitesse de convergence vers le minimum [1]. Il doit être bien choisi sinon on s'expose à deux problèmes :

- On descend trop doucement : le learning rate est trop faible. C'est le cas de la première descente de gradient (3.3a).
- On ne descend pas et on diverge : le learning rate est trop fort. C'est le cas de la dernière descente de gradient (3.3d).

La deuxième descente de gradient (3.3b) est parfaite : il y a autant d'itérations que pour les autres essais, mais elle converge rapidement vers le minimum que l'on cherche sans pour autant être instable

1. S'il existe !



(a) Descente de gradient avec $\eta = 0.01$

(b) Descente de gradient avec $\eta = 0.10$

(c) Descente de gradient avec $\eta = 0.90$

(d) Descente de gradient avec $\eta = 1.01$

FIGURE 3.3 – Impact du learning rate η sur la descente de gradient

comme (3.3c).
Il n'existe pas de règle universelle pour trouver le learning rate optimal, il s'obtient par de multiples essais empiriques. Il n'est pas non plus nécessaire qu'il soit constant : on peut définir des suites de learning rate par exemple. On peut par exemple vouloir avoir un learning rate fort pour les premières itérations, puis le réduire progressivement avec le nombre d'itérations.

Exercice 3.2. *A l'aide de l'équation (3.3), montrer que la descente de gradient pour le problème :*

$$x^* = \arg\min_{x \in \mathbb{R}} (x-1)^2$$

Peut s'écrire sous la forme :

$$x_{t+1} = x_t - 2\eta(x_t - 1)$$

Suite à cet exercice trivial pour manipuler les équations, appliquons cette idée à notre problème :

$$\begin{aligned} \theta^* &= \arg\min_{\theta \in \mathbb{R}^d} \sum_{i=1}^{n} \mathcal{L}(\theta; x^{(i)}, y_i) \\ \mathcal{L}(\theta; x, y) &= -\left[y \ln\left\{f_\theta(x)\right\} + (1-y) \ln\left\{1 - f_\theta(x)\right\}\right] \end{aligned}$$

Pour le faire, rien de mieux qu'un exercice guidé.

Exercice 3.3. *On rappelle que* $f_\theta(x) = \dfrac{1}{1+e^{-<\theta,x>}}$. *Montrer que :*

1. $f_\theta(x) = \dfrac{e^{<\theta,x>}}{1+e^{<\theta,x>}}$
2. $f_\theta(-x) = 1 - f_\theta(x)$
3. $\dfrac{\partial \ln}{\partial \theta_j}\left(f_\theta(x)\right) = x_j\left(1 - f_\theta(x)\right)$
4. $\dfrac{\partial \ln}{\partial \theta_j}\left(1 - f_\theta(x)\right) = -x_j f_\theta(x)$
5. $\dfrac{\partial \mathcal{L}}{\partial \theta_j}\left(\theta; x^{(i)}, y_i\right) = x_j^{(i)}\left(f_\theta(x^{(i)}) - y_i\right)$
6. *Conclure que la descente de gradient pour le problème avec la fonction de coût est :*

$$\theta_j^{t+1} = \theta_j^t - \eta \sum_{i=1}^{n} x_j^{(i)}\left(f_\theta(x^{(i)}) - y_i\right)$$

Si l'on veut aller encore plus loin, on peut essayer de résoudre le même exercice mais en traitant le problème de la régression linéaire cette fois. On en déduit exactement la même équation d'actualisation des paramètres pour la régression linéaire !

3.3 Mesurer la performance d'une classification

Nous sommes maintenant capables de traiter un problème de classification. Nous devons être capables de mesurer la performance d'un algorithme. Le but est le même que celui de la régression, avoir des métriques différentes pour avoir des éclairages complémentaires sur la qualité de la modélisation. Il comporte néanmoins une différence majeure. En classification on traite de classe, et donc un modèle va prédire un score de probabilité d'appartenance à une classe. Ainsi, en disant qu'à partir d'un certain seuil on appartient à une classe ou une autre, on aura des performances différentes d'un autre seuil. Commençons par voir comment mesurer la performance d'un classifier pour un seuil donné.

3.3.1 Pour un seuil fixé

On suppose que l'on dispose d'un dataset $\mathcal{D}$ et d'un classifier entraîné. On suppose de plus que nous sommes dans le cadre d'un algorithme qui prédit si la valeur d'un indice aujourd'hui est plus faible que celle de demain. Autrement dit, on veut pouvoir prédire la hausse de la valeur d'un indice boursier. Avec ce dataset et le classifier entraîné, on prédit, et pour un seuil fixé, nous obtenons $\hat{y}$ notre vecteur de classe prédite. Avec y qui comporte les vraies classes, nous sommes capables de construire la **matrice de confusion** :

		Prédit	
		Classe 0 (baisse)	Classe 1 (hausse)
Réel	Classe 0	TN	FP
	Classe 1	FN	TP

Avec :

- **TP** : Vrai positif - une hausse identifiée comme une hausse
- **FN** : Faux négatif - une hausse identifiée comme une baisse
- **FP** : Faux positif - une baisse identifiée comme une hausse

- **TN** : Vrai négatif - une baisse identifiée comme une baisse

On souhaite être le plus précis possible dans tous les cas de figure et donc avoir les plus grandes valeurs possibles dans les vrais positifs et vrais négatifs. On ne peut, cependant, éviter les faux positifs et les faux négatifs.

Accuracy

Une première manière de mesurer la performance est de considérer l'**accuracy** définie comme :

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

L'accuracy mesure de manière frontale la proportion d'observations bien classées. Elle est très facilement interprétable mais se comporte mal pour des datasets déséquilibrés : des datasets où la classe d'intérêt est sous-représentée ou sur-représentée. Dans notre cas, il est probable qu'il y ait autant de hausse que de baisse. Mais si l'on se place dans le cadre de la prédiction de hausse exceptionnelle, naturellement on aura beaucoup moins d'observations dans la classe d'intérêt.

Exercice 3.4. *On souhaite prédire une hausse exceptionnelle, et dans le dataset que l'on a à disposition, il y a 1% de classe 1 (hausse exceptionnelle). Construire un algorithme qui permet d'atteindre 99% d'accuracy.*

Solution. Un simple algorithme qui prédit systématiquement 0 répond au problème. □

Avec cet exercice on comprend la limitation claire et le piège dans lequel il ne faut pas tomber avec l'accuracy. Il nous faut donc d'autres mesures de performance.

Precision, Recall et F1-score

La précision et le recall sont deux mesures différentes mais toujours présentées ensembles car complémentaires. Elles sont définies par :

$$\begin{aligned} \text{Précision} &= \frac{TP}{TP + FP} \\ \text{Recall} &= \frac{TP}{TP + FN} \end{aligned}$$

La précision mesure la proportion de bonne prédiction quand la prédiction est positive. Dans notre exemple, une pertinence de 70% indique que sur 100 hausses prédites, 70 évolutions ont effectivement été des hausses.
Le recall mesure la proportion de prédictions positives qui ont été prédites comme telle. Dans notre exemple, un recall de 60% indique que sur 100 hausses constatées, nous avions prédit 60 d'entre elles.

La précision (ou pertinence) et le recall (ou couverture) adressent deux visions complémentaires qu'il faut prioriser selon le problème que l'on traite. Et il faut souvent choisir l'une des deux à maximiser, tout en gardant une performance décente pour l'autre. Quand on ne sait vraiment pas laquelle prioriser, on peut considérer le F1-score :

$$\text{F1-score} = \frac{2}{\frac{1}{\text{Précision}} + \frac{1}{\text{Recall}}}$$

Ce qui correspond à la moyenne harmonique entre la précision et le recall. Maximiser le F1-score revient à maximiser le compromis entre précision et recall, mais pas au sens d'une moyenne arithmétique classique. La moyenne harmonique est toujours inférieure à la valeur d'une moyenne arithmétique [2], donc le F1-score est assez conservateur sur la performance.

Exercice 3.5. *Vous avez trop de mails, et vous demandez à votre data scientist de concevoir un algorithme qui va prioriser les mails en essayant de prédire les mails qui sont les plus importants. Vous lui donnez un dataset d'entraînement et un dataset de test. Dans le dataset de test, il y a 1000 mails dont 200 sont importants. Il vous présente un premier modèle qui pour un certain seuil (A) présente la matrice de confusion suivante :*

		Prédit	
		Classe 0	*Classe 1*
Réel	*Classe 0*	700	100
	Classe 1	50	150

Pour un autre seuil (B), il présente cette matrice de confusion :

		Prédit	
		Classe 0	*Classe 1*
Réel	*Classe 0*	760	40
	Classe 1	80	120

1. *Calculer l'accuracy, la précision, le recall et le F1-score de chacun des seuils.*
2. *Conclure sur le seuil que vous souhaitez conserver.*

Solution. On a le tableau suivant :

Seuil	Accuracy	Précision	Recall	F1-score
A	85%	60%	75%	67%
B	88%	75%	60%	67%

TABLE 3.1 – Performance de l'algorithme pour deux seuils différents

Les deux seuils ont le même F1-score, mais pas la même accuracy. C'est expliqué par l'*échange* de valeur entre la précision et le recall sur les deux seuils. C'est un cas qui illustre à la fois l'inefficacité de l'accuracy dans un cadre déséquilibré, et aussi l'importance de regarder plusieurs métriques. Si l'on se concentre uniquement sur le F1-score, nous voyons deux seuils à la performance identique alors que ce n'est pas le cas.
Ici, il faut que l'on décide si l'on préfère lire le plus de mails importants (seuil B) quitte à lire également des mails peu pertinents ou au contraire ne lire que des mails importants quitte à ne pas tous les lire. □

Les choix de mesure de performance **doivent** être traités avec le métier. Je conseille de traiter le sujet au tout début d'un projet, car la décision qui en résultera orientera le reste du développement. Un datascientist peut, dans sa phase de test, utiliser d'autres métriques de performance, mais ce seront les métriques de performance validées avec le métier qui feront foi.

3.3.2 Sans choix de seuil

À ce stade, nous sommes capables de dire qu'un seuil est meilleur qu'un autre en terme de performance selon une ou plusieurs métriques. De même, nous sommes capables de comparer des modèles entre eux

2. Ce résultat est prouvé dans la section (B.4)

pour des seuils fixés. Mais ne pourrions-nous pas avoir une idée de la performance *globale* du classifier ? On cherche un moyen de s'affranchir du choix de seuil pour mesurer la performance d'un classifier.

Courbe ROC

L'aire sous la courbe ROC est la métrique la plus connue qui répond à cette problématique. On définit la courbe ROC comme l'ensemble des points $(\mathrm{TPR}(t), \mathrm{FPR}(t))$ où t représente un seuil possible d'un score. Bien qu'en réalité cette courbe soit composée de points discrets, on interpole linéairement pour former une courbe (cela se justifie mathématiquement).

Exercice 3.6 (Baseline aléatoire)**.** *Supposons que l'on construise un classifier qui prédit aléatoirement. A quoi ressemble sa courbe ROC ?*

Solution. On prédira 1 quand le score associé à une observation est supérieur à un seuil donné.

Si le seuil est 0, alors on prédit toujours 1, donc $\mathrm{TPR}(0) = 1$ et $\mathrm{FPR}(0) = 1$. Inversement, si le seuil est 1, alors on prédit toujours 0, donc $\mathrm{TPR}(1) = 0$ et $\mathrm{FPR}(1) = 0$. On sait donc que la courbe passera forcément par les points $(0, 0)$ et $(1, 1)$, peu importe le modèle concerné.

Considérons maintenant un modèle qui prédit aléatoirement chacune des observations. Donc si le seuil est p, alors $\mathrm{TPR}(p) = p$ et $\mathrm{FPR}(p) = p$. Donc la courbe ROC d'un modèle aléatoire est la droite $y = x$. □

La courbe ROC a donc l'appréciable propriété d'avoir une baseline visuelle claire. L'aire sous cette courbe vaut 0.5, et c'est cela qui nous intéresse. Plus une courbe ROC est performante, plus elle va tendre vers la courbe brisée $(0, 0) \to (0, 1) \to (1, 1)$, et l'aire de cette courbe vaut 1 ! L'interprétation est la suivante : le meilleur modèle est celui qui obtient la plus grande aire sous la courbe. Inversement, un très mauvais modèle obtient une aire sous sa courbe inférieure à 0.5.

Courbe precision-recall

Dans la même veine que la courbe ROC, on peut décider de regarder la courbe precision-recall, et on recommande [FK15] qui peut à lui seul alimenter toute une séance. Avant de ne parler que de la courbe precision-recall, il est à noter qu'un lien est établi entre cette courbe et la courbe ROC et on invite à étudier [DG06].
La courbe precision-recall est définie, comme son nom l'indique, comme l'ensemble des points (pertinence, recall) pour chacun des seuils possibles.

Exercice 3.7 (Baseline aléatoire)**.** *Supposons que l'on construise un classifier qui prédit aléatoirement. A quoi ressemble sa courbe precision-recall ?*

Solution. Si on note p la proportion de la classe d'intérêt, alors un modèle aléatoire aura une précision de p. Ainsi, sa courbe est une ligne horizontale avec une precision fixée. □

Avec cette méthode, il n'existe pas de baseline aléatoire universelle. De plus [FK15] montre que l'analyse est bien plus fine dans ce cas, et on peut rapidement se tromper dans l'exploitation de ces courbes.

> *Precision-Recall analysis abounds in applications of binary classification where true negatives do not add value and hence should not affect assessment of the classifier ?s performance. Perhaps inspired by the many advantages of receiver operating characteristic (ROC) curves and the area under such curves for accuracy- based performance assessment, many researchers have taken to report Precision-Recall (PR) curves and associated areas as performance metric. We demonstrate in this paper that this practice is fraught with difficulties, mainly because of incoherent scale assumptions.*
>
> — Peter Flach, Meelis Kull (2006)

Dans cet article, les auteurs proposent une manière de modifier l'espace de projection pour obtenir les propriétés théoriques *agréables* que possède la courbe ROC. On conseille de le lire en plusieurs temps : les mathématiques sont abordables, mais nécessitent du temps pour visualiser et saisir les nuances de l'analyse. C'est l'une des raisons qui explique pourquoi la courbe ROC est beaucoup plus utilisée en pratique.

Chapitre 4

Arbre et Random Forest

La structure d'un enseignement en mathématiques n'a pas évolué depuis les éléments d'Euclide. Nous nous appuyons sur un ensemble de règles que l'on suppose vrai, les axiomes, souvent très simple. Puis en les combinant astucieusement nous pouvons obtenir des résultats plus conséquents et avec un ensemble de résultats nous pouvons avoir une connaissance plus approfondie d'un sujet. Avec ces lemmes et propositions nous arrivons parfois à trouver des théorèmes qui font souvent le lien entre des idées fortes de plusieurs domaines. Finalement, l'ensemble de ces grands résultats nous permet d'obtenir une vue d'ensemble des mathématiques chaque jour un peu plus complet, et chaque jour nous mesurons notre ignorance sur certains sujets.

À la liste des blocs fondamentaux du Machine Learning s'ajoute l'algorithme que nous allons présenter dans ce chapitre. En combinant des informations entres elles, les arbres de décisions vont, à l'image des lemmes et des propositions, être capables de devenir des *théorèmes* qui donnent une vue globale sur le dataset que l'on apprend. En les cumulant nous sommes capables d'avoir une vue encore plus complète et fine du dataset que l'on traite.

Si l'on poursuit la réflexion, on se dit que l'on pourrait donc en cumulant les arbres être capable de tout apprendre et tout comprendre ! Cependant cela contredit ce que nous avions annoncé dans l'introduction : aucun modèle n'est parfait. De même qu'en mathématiques, Kurt Gödel est un logicien qui a réussit à montrer que pour un système d'axiomes donné il est impossible de tout démontrer. Ainsi, à une question mathématique la réponse peut être vrai, faux ou indécidable !

> *Kurt Gödel's achievement in modern logic is singular and monumental - indeed it is more than a monument, it is a landmark which will remain visible far in space and time. The subject of logic has certainly completely changed its nature and possibilities with Gödel's achievement.*
>
> — John Von Neumann (1931)

Dans une moindre mesure, l'algorithme Random Forest que nous présenterons naturellement après les arbres est également une étape majeure du développement du Machine Learning, et reste un des algorithmes les plus performants à ce jour.

4.1 Arbre de décision

Les arbres sont pour la première fois présentés dans [BFOS84], et des améliorations de ces arbres sont présentés : [Qui86] et [Qui96]. Essayons de formaliser l'idée finalement assez intuitive !

On considère un problème de classification avec un dataset $\mathcal{D} = \left\{(x^{(i)}, y_i) \mid \forall i \leqslant n,\ x^{(i)} \in \mathbb{R}^d, y_i \in \{0,1\}\right\}$. On veut construire un estimateur qui est capable de détecter des tendances non linéaires sans avoir à les prévoir dans le feature engineering.
L'idée est de partitionner l'espace engendré par $\mathcal{D}$, dont voici la procédure à chaque étape :

1. Pour chaque information $j \leqslant d$, on cherche la meilleure séparation de l'espace sous la forme $x_j \leqslant \alpha$ avec α la valeur de la coupure (à trouver)
2. On sélectionne la meilleure coupure parmi les d meilleures coupures identifiées à l'étape précédente.
3. On applique les deux étapes précédentes aux deux espaces définis par la coupure : l'espace qui vérifie la condition, et celui qui ne la vérifie pas

Ce fonctionnement récursif, explicable sous forme de règles, nous permet d'obtenir la fonction de classification estimée :

Probabilité de la classe d'intérêt dans la partition P

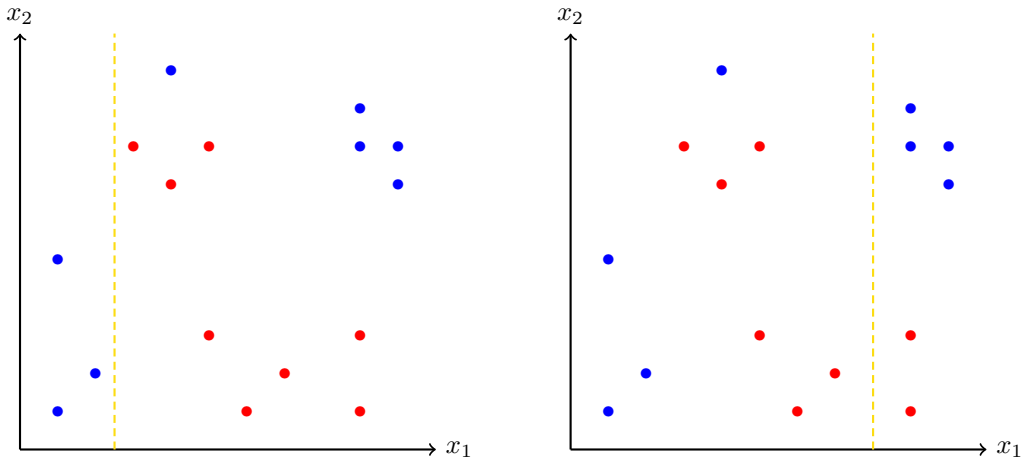
$$f_\theta(x) = \sum_{P \in \theta} \mu_P \mathbb{1}_{\{x \in P\}}$$

Partition de l'espace

On comprend à présent le nom de l'algorithme : arbre de décision. Une fois que l'on a une observation à prédire, il faut descendre un ensemble de conditions qui vont nous amener dans une partition apprise qui nous donne la probabilité de faire partie de la classe d'intérêt.

4.1.1 Meilleure partition

Le challenge est donc de réussir à partitionner l'espace des données $\mathcal{D}$. La difficulté réside dans le choix de la *meilleure coupure*. Nous avons déjà vu cette idée dans l'exercice (1.1), et nous avions soulevé la question suivante : comment trouver la meilleure séparation ? Ce problème est illustré dans la figure (4.1).



(a) Séparation 1 (b) Séparation 2

FIGURE 4.1 – Sélection de la meilleure séparation de l'espace pour une information donnée

On voit ici les deux possibilités. Vaut-il mieux écarter directement 3 observations du même groupe du reste, ou augmenter les proportions des classes différentes dans des espaces différents pour faciliter le travail des prochaines coupures ?

Pour être capable de choisir entre la séparation 1 et la séparation 2, nous devons être capables de définir ce que l'on appelle intuitivement *la meilleure coupure*.

Définition 1 (Mesure d'hétérogénéité). *Soit $\Phi : [0,1] \mapsto \mathbb{R}_+$ une fonction continue. On dit que Φ est une mesure d'hétérogénéité si et seulement si :*

- *Désordre : Φ est maximal en $x = \frac{1}{2}$*
- *Ordre : Φ est nulle en $x = 0$ et $x = 1$*
- *Monotonie : Φ est croissante sur $\left[0, \frac{1}{2}\right]$ puis décroissante sur $\left[\frac{1}{2}, 1\right]$*

On comprend avec la définition que l'on travaille avec la proportion d'observations de la classe d'intérêt. Ainsi, on demande à cette fonction d'être maximale en $x = \frac{1}{2}$ pour modéliser le désordre maximal dans une partition qui a autant d'observations de la classe 0 et de la classe 1.
De même, l'ordre est maximal quand la proportion est égale à 0 ou égale à 1. La propriété de monotonie nous assure que l'on ne peut pas obtenir d'autres minimums (même locaux) autres que 0 et 1.

La mesure la plus classique est l'indice de Gini :

$$\Phi(p) = 2p(1-p) \qquad \text{(Indice de Gini)}$$

Une autre mesure très classique, issue de la thermodynamique et de la théorie de l'information, est l'entropie :

$$\Phi(p) = -p\log_2(p) - (1-p)\log_2(1-p) \qquad \text{(Entropie)}$$

Exercice 4.1. *Vérifier que l'indice de Gini et l'entropie sont bien deux mesures d'hétérogénéité.*

On dispose maintenant de deux manières de mesurer l'impureté d'une partition. Il reste à trouver la meilleure division δ de l'espace t.
On note $\Delta\Phi(\delta, t)$ la variation d'impureté réalisée par la division δ de l'espace t. Cette coupure divise l'espace t en un espace t_g qui vérifie la condition de δ, et t_d qui ne la vérifie pas. On peut donc définir la variation d'impureté par :

Proportion de la classe d'intérêt qui vérifie la condition

$$\Delta\Phi(\delta, t) = \Phi(t) - \Big(p_g \, \Phi(t_g) + p_d \, \Phi(t_d) \Big)$$

Proportion de la classe d'intérêt qui ne vérifie pas la condition

On voit bien ici l'idée de vouloir réduire le plus possible le *désordre* dans l'espace t en cherchant à le subdiviser en deux parties plus homogènes. On peut maintenant définir la meilleure division comme :

$$\delta^*(t) = \arg\max_{\delta \in \Xi} \Delta\Phi(\delta, t) \quad \text{avec } \Xi \text{ l'ensemble des coupures possibles}$$

Exercice 4.2. *Comparer les séparations de l'exemple (4.1) et statuer sur la meilleure des deux pour chaque mesure d'hétérogénéité.*

On applique itérativement cette procédure pour obtenir un partitionnement qui ressemble à la figure (4.2).
Maintenant que l'on a le partitionnement, on peut facilement résumer l'algorithme sous forme d'arbre également.

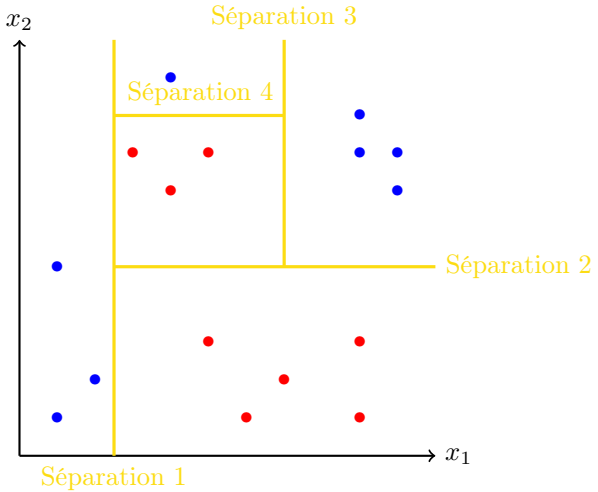


FIGURE 4.2 – Exemple de partitionnement de l'espace

Exercice 4.3. *Écrire le partitionnement de la figure (4.2) sous forme d'un arbre de décision.*

Solution. On peut l'écrire sous la forme présentée en figure (4.3).

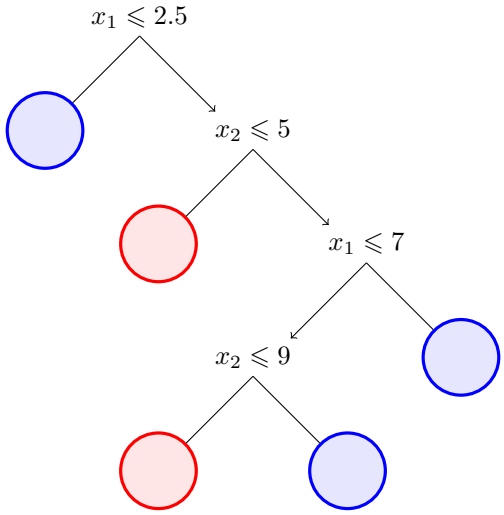


FIGURE 4.3 – Arbre de décision lié au problème de classification (4.1)

□

Les différentes *versions* de l'algorithme des arbres de décisions que l'on a citées en introduction (CART, ID3, C4.5) sont des versions différentes de l'algorithme présenté ici (CART). L'existence de ces différentes versions sont la démonstration que les arbres de décisions sont intéressants au-delà du Machine Learning, en informatique en général, par exemple que construire un arbre de décision binaire optimal est un problème NP-Complet [LR76].

4.1.2 Critères d'arrêt

Nous sommes donc capables de créer un arbre, mais nous devons aussi donner des critères d'arrêt. Nous pouvons contrôler la génération de l'arbre avec les mesures suivantes :

- Demander un nombre minimal d'observations dans un noeud pour le couper, et si ce nombre n'est pas atteint, on obtient une feuille.
- Ne pas couper un noeud s'il n'y a aucun gain supérieur à un seuil défini par l'utilisateur, on obtient une feuille
- Limiter la profondeur de l'arbre en ne dépassant pas un nombre maximal de coupures de noeud

L'ensemble de ces critères a pour but de limiter le sur-apprentissage des données par l'algorithme, et d'essayer de maximiser sa capacité a généraliser son apprentissage.

Nous savons donc maintenant comment se construit et se régularise un arbre de classification. Comment cela fonctionne-t-il pour faire de la régression ? Les arbres sont capables de faire les deux tâches en remplaçant les fonctions d'hétérogénéité par la MSE par exemple. Ainsi, un arbre cherche à minimiser la MSE à chaque coupure, et pour donner la prédiction finale, on peut imaginer prendre la moyenne des valeurs y présentent dans la feuille (ou la médiane).

Le principal avantage des arbres de décision est **l'explicabilité** : il est aisé de comprendre pourquoi une décision a été prise par un arbre. Les arbres sont aussi facilement utilisables pour des problèmes avec plusieurs classes, et en utilisant des données à la fois numériques et catégorielles. Cela en fait un algorithme très complet et souple qui justifie son utilisation dans de nombreux cas d'usage.

Malheureusement, les arbres sont également largement sujets à un sur-apprentissage et à des problèmes de variances. Un petit changement dans les données d'entraînement peut donner lieu à des arbres très différents avec des performances très différentes. Les prédictions d'un arbre sont continues par morceaux, et donc ne sont pas régulières du tout. Ainsi, il est très difficile pour un arbre d'extrapoler, c'est à dire se prononcer sur une observation qui se place sur un domaine qu'il n'a jamais vu.

Exercice 4.4 (Difficulté à extrapoler)**.** *Exhiber un exemple de problème de régression (à construire) où une régression linéaire réussit à extrapoler, mais pas un arbre de décision.*

On peut résoudre cet exercice de multiples manières, mais nous n'en proposerons qu'une.

Solution. Prenons un exemple le plus simple possible : une seule information $x \in [0, 10]$ qui donne la variable d'intérêt y :

$$y = x + \frac{1}{2}\cos(x) + \varepsilon \text{ avec } \varepsilon \sim \mathcal{N}(0, 0.1)$$

On génère un dataset avec des exemples pour x entre 0 et 10. Les deux algorithmes apprennent facilement les bons paramètres sur ce dataset simple, mais l'arbre ne prédira pas correctement pour $x = 12$, alors que la régression linéaire oui. □

Devant cet exemple jouet, on peut imaginer une situation plus proche de la réalité :

Exercice 4.5 (Faire communiquer deux algorithmes)**.** *On souhaite prédire des prix de certaines crypto-monnaies qui sont connues pour être particulièrement volatiles. L'enjeu d'estimer au mieux le prix est donc fort : une bonne prédiction peut donner lieu à un grand gain, et une mauvaise prédiction une grande perte. On sollicite notre équipe de data-scientists, et ils nous présentent deux algorithmes :*

- *Régression Linéaire : marche plutôt bien, et reste raisonnablement correcte pour les grandes variations de prix*
- *Arbre de régression : marche beaucoup mieux quand les prix sont dans les moyennes, mais est très mauvais dès qu'on sort des prix moyens*

Expliquer succinctement pourquoi les comportements relatifs étaient prévisibles, et proposer des solutions pour utiliser les deux algorithmes ensembles et faire mieux que les deux séparément.

Solution. L'arbre possède de meilleures performances que la régression linéaire sur des prix *classique* car il a beaucoup plus de puissance pour apprendre qu'une régression linéaire classique. Donc tant que les données de tests ressemblent à celles d'entraînement, l'arbre aura plutôt tendance à être meilleur que la régression.
Quand les variations de prix seront inédites, alors on aura le même phénomène que celui qu'on a observé dans l'exercice précédent : la régression linéaire pourra prendre des valeurs qu'elle n'a jamais prise, mais pas l'arbre.

Finalement, il faudrait être capable de combiner les deux approches pour être plus performant globalement. Il existe plusieurs moyens, faire la moyenne des prédictions en est une par exemple. On peut la pondérer par la performance des algorithmes sur un jeu de données différent de celui de l'entraînement pour avoir une performance accrue éventuellement. □

4.2 Méthodes ensemblistes

On souhaiterait être capable de réduire la variance d'un modèle sans pour autant accepter d'augmenter le biais. En effet, on se souvient de l'équation (2.3) :

$$\mathrm{MSE}(y, \hat{f}(x)) = \left(\mathrm{Bias}\left[\hat{f}(x)\right]\right)^2 + \mathbb{V}\left[\hat{f}(x)\right] + \sigma^2$$

Avec σ^2 une erreur incompressible. On comprend également avec cette équation que l'on ne sera pas capable de réduire la variance d'un algorithme sans augmenter le biais, notre objectif semble impossible. Il l'est, mais pas si on considère *plusieurs* algorithmes : c'est l'idée des méthodes ensemblistes.

4.2.1 Bagging

Supposons que l'on traite un problème de régression, que l'on dispose de m régresseurs $(f_k)_{k\leqslant m}$ chacun entraîné sur m échantillons issus de la distribution engendrée par le dataset. Par souci de lisibilité, on ne note pas la dépendance de f en son vecteur de paramètres θ. On construit un régresseur fort à partir de ces modèles :

$$F(x) = \frac{1}{m}\sum_{k=1}^{m} f_k(x)$$

Pour saisir l'intérêt de la proposition, résolvons l'exercice suivant.

Exercice 4.6. *Montrer que :*

1. $\mathbb{E}[F(x)] = \mathbb{E}[f_k(x)]$ *pour n'importe quel* $k \leqslant m$ *puisque* $(f_k(x))_{k \leqslant m}$ *suivent la même loi.*
2. $\mathbb{V}[F(x)] = \frac{1}{m}\mathbb{V}[f_k(x)]$ *pour n'importe quel* $k \leqslant m$.
3. *Conclure sur l'intérêt de la méthode proposée.*

Solution. On suppose que les variables $(f_k(x))_{k \leqslant m}$ sont indépendantes et identiquement distribuées. Cela vient de l'entraînement sur des datasets indépendants et identiquement distribués.

1. Par linéarité de l'espérance, et parce que les $(f_k(x))_{k \leqslant m}$ suivent la même loi, on déduit :

$$\mathbb{E}[F(x)] = \mathbb{E}\left[\frac{1}{m}\sum_{k=1}^{m} f_k(x)\right] = \frac{1}{m}\sum_{k=1}^{m}\mathbb{E}[f_k(x)] = \mathbb{E}[f_k(x)]$$

2. Avec les propriétés classiques de la variance, et parce que les $(f_k(x))_{k \leqslant m}$ sont indépendants et identiquement distribués :

$$\mathbb{V}[F(x)] = \mathbb{V}\left[\frac{1}{m}\sum_{k=1}^{m} f_k(x)\right] = \frac{1}{m^2}\sum_{k=1}^{m}\mathbb{V}[f_k(x)] = \frac{1}{m}\mathbb{V}[f_k(x)]$$

3. On conserve le même biais que les algorithmes composant le régresseur fort, mais on réduit la variance de l'estimateur proportionnellement au nombre de régresseurs que l'on forme.

□

On comprend donc que l'idée ensembliste est de tirer profit du nombre d'estimateurs *faibles* pour former un estimateur *fort* avec une variance réduite par rapport à chacun de ses composants. Si l'on pousse l'idée encore plus loin, faisons une infinité d'estimateurs ! Ainsi, nous aurons une variance qui tend vers 0. Les résultats théoriques de l'exercice reposent sur l'idée que l'on est capable d'entraîner m régresseurs avec m datasets indépendants et identiquement distribués. Dans la pratique, ce n'est jamais vraiment le cas, donc tendre vers une variance nulle n'est pas possible. Comment faire pour s'en rapprocher le plus possible ?

La solution donne son nom à la section : nous allons réaliser du **bootstrap aggregation** également appelé **bagging**. L'idée est d'utiliser la distribution empirique que l'on observe avec le dataset $\mathcal{D}$ comme approximation de la vraie distribution sous-jacente que l'on suppose : nous générons donc m datasets en échantillonnant, avec remplacement, le dataset $\mathcal{D}$. Le reste de la méthode est décrit au-dessus. L'idée est que plus le dataset $\mathcal{D}$ est grand (donc $n \to +\infty$) alors la distribution empirique converge vers la distribution réelle. Ainsi, nous avons un boostrap très efficace pour exploiter au mieux l'idée d'ensemble.

En pratique, les datasets formés par le boostrap ne sont pas parfaitement indépendants, donc nous n'atteindrons jamais la réduction de variance théorique. Mais on peut montrer que si les estimateurs *faibles* ont une variance σ^2 et que les datasets entre eux sont corrélés avec une valeur ρ, alors :

$$\mathbb{V}\left[\frac{1}{m}\sum_{k=1}^{m} f_k(x)\right] = \frac{1}{m}(1-\rho)\sigma^2 + \rho\sigma^2 \tag{4.1}$$

Exercice 4.7. *Vérifier que la formule est cohérente avec le cas où les datasets sont indépendants. Que se passe-t-il quand les datasets sont parfaitement corrélés ?*

Solution. Quand $\rho = 0$ il n'y a aucune corrélation et on retrouve bien la prédiction théorique précédente. A l'inverse, quand $\rho = 1$ il n'y a aucun gain à utiliser le bagging : on va en réalité apprendre m fois le même modèle. □

Nous avons présenté jusqu'ici le bagging dans le cadre d'une régression, mais à nouveau cela peut parfaitement s'appliquer dans le cadre d'une classification. L'agrégation des différents estimateurs *faibles* est très similaire qu'il s'agisse de probabilité estimée ou de classe directement.

La méthode du bagging [Bre96a] nous permet donc de tirer parti de l'ensemble des briques élémentaires du Machine Learning que l'on a apprises jusqu'à maintenant pour construire des estimateurs plus performants.

4.2.2 Random Forest

Pour pouvoir exploiter au maximum l'idée du bagging, il est nécessaire d'avoir une corrélation entre les estimateurs la plus faible possible, comme nous l'avons vu dans l'équation (4.1). Il faut donc intégrer un peu d'aléatoire, notamment dans les arbres de décisions, qui seront les estimateurs *faibles* de l'ensemble. Pour chaque coupure lors de l'entraînement de l'arbre, au lieu de chercher la meilleure séparation pour toutes les d informations possibles, on ne cherche la meilleure coupure que pour un sous-ensemble aléatoire de ces informations. On ajoute donc beaucoup d'aléatoire à la méthode, ce qui nous garantit un peu plus de faible corrélation entre les estimateurs.

Avant de détailler les grands paramètres de l'algorithme Random Forest [Bre01], étudions la proposition de l'algorithme Extra Trees [GEW06].
La procédure est identique à l'algorithme de Random Forest, mais diffère dans la sélection de la coupure. Au lieu de chercher la coupure optimale pour chaque sous-ensemble d'informations sélectionnées, il va chercher la meilleure coupure *aléatoire* parmi l'ensemble des sous-ensemble d'informations sélectionné. L'aléatoire est ajouté même au niveau de la sélection de la valeur, ce n'est plus du tout optimal. On réduit donc très fortement la variance puisqu'il est beaucoup plus probable que chaque estimateur soit très faiblement corrélé au reste. Cependant, on perd nettement en qualité pour chaque estimateur *faible*.

Que ce soit pour l'algorithme Random Forest ou Extra Trees, les paramètres disponibles dans Scikit-learn pour exploiter ces algorithmes sont très similaires, et on peut décrire les principaux :

- Paramétrer les arbres :
  - `criterion` : pour définir la métrique à utiliser pour faire une coupure
  - `max_depth` : limiter la profondeur maximale d'un arbre
  - `min_samples_leaf` : nombre minimal d'observations dans une feuille
  - `max_features` : nombre d'informations à considérer pour chaque coupure
- Paramétrer la forêt :
  - `n_estimators` : nombre d'arbres à construire dans la forêt
  - `boostrap` : si vrai, le boostrap est à utiliser. Sinon la totalité du dataset est utilisée pour construire chaque arbre
  - `max_samples` : taille de chaque nouveau dataset créé quand on utilise le boostrap

Avec la théorie que l'on a vue jusqu'à présent, on sait comment réagir pour chaque possibilité lors de l'entraînement d'un arbre. Voyons plusieurs cas d'usage.

Exercice 4.8. *On travaille avec un data-scientist. Pour chaque description, proposer des axes de recherche pour améliorer son utilisation de l'algorithme Random Forest.*

1. *On obtient des performances faibles, on dirait que l'on sous-apprend les données.*
2. *On obtient de très bonnes performances sur le dataset de train, mais très mauvaises sur le jeu de test.*
3. *On obtient des performances correctes, mais très variables.*

Solution.

1. Si l'on sous-apprend les données, alors c'est que le modèle n'est pas assez complexe pour apprécier l'ensemble des particularités des données. Une manière d'avancer dans ce cas est d'augmenter la profondeur des arbres, baisser le nombre d'observations dans une feuille ou bien d'augmenter le nombre d'informations considérées pour chaque coupure. On peut également essayer d'augmenter le nombre d'arbres dans l'ensemble. Bien sûr, il ne faut pas tester tous ces axes en même temps.
2. Nous sommes dans un cas de sur-apprentissage potentiel [1]. Il faut faire l'inverse de ce que l'on a préconisé à la question précédente.
3. Cette fois nous avons trouvé le bon trade-off entre sous-apprentissage et sur-apprentissage. Mais la variance de la Random Forest est encore trop forte. Une piste est de limiter le nombre d'informations à considérer pour chaque coupure, faire du boostrap (si ce n'est pas le cas) avec un peu moins de données pour chaque arbre. Si ce n'est pas suffisant, on peut envisager d'exploiter l'algorithme Extra Trees plutôt que Random Forest.

□

On peut noter qu'il existe d'autres manières de définir un critère de coupure (par exemple De Mantaras [DM91]) mais il n'y a pas d'implémentation dans scikit-learn. Il faut donc le faire nous-même.

Nous pouvons conclure ce chapitre en remarquant que les arbres sont des *universal approximators* car avec un ensemble d'arbres (une Random Forest par exemple) on peut effectivement apprendre n'importe quelle fonction continue par morceau.

1. Ou alors le dataset d'entraînement est très différent du dataset de test, ce qui peut être le cas dans le cadre de données qui changent dans le temps. Par exemple, le volume de paris sur un site de paris sportif entre un jour avec ou sans Ligue des champions.

Chapitre 5

Boosting

Jusqu'ici nous avons appris à entraîner et mesurer la performance d'un algorithme de Machine Learning à partir d'un dataset pour des problèmes de classification et de régressions. Pour obtenir la meilleure performance, nous organisons des compétitions de modèle après avoir réaliser un travail conséquent sur l'extraction d'informations pertinente.
Ces étapes, caricaturale, sont l'essentiel du Machine Learning, mais nous n'avons pas encore rencontré la superstar des compétitions de Machine Learning : la méthode du Boosting et un cas particulier, le Gradient Boosting.

On commencera par étudier l'algorithme AdaBoost qui illustre parfaitement les grandes idées du Boosting. Nous traiterons ensuite d'une partie du Boosting pour finalement introduire les trois algorithmes les plus utilisés dans les années 2020 qui exploitent le Gradient Boosting. Ce chapitre a pour but d'introduire les notions essentielles à la compréhension du Boosting, comprendre les différences entre les différents algorithmes et se préparer à comprendre les potentiels algorithmes qui seront présentés dans le futur.

> *Boosting is an approach to machine learning based on the idea of creating a highly accurate prediction rule by combining many relatively weak and inaccurate rules.*
> — Robert Schapire (2013)

5.1 Algorithme AdaBoost

L'algorithme AdaBoost [1] [FS97] repose sur l'idée de noter les weak learners formés et les observations. En notant les observations, on attire l'attention des weak learners sur celles étant les plus difficiles à bien prédire. En notant les weak learners, on donne plus de poids aux weak learners les plus performants dans la prédiction finale.
Les weak learners d'AdaBoost sont appelés des **souches** : ce sont des arbres de décision de profondeur 1 ou 2. Ces modèles étant très simplistes, c'est le travail successif des différents weak learners et de l'association de toutes ces souches qui donne la force d'AdaBoost.

Plus formellement, on note :

- T le nombre d'itérations [2] que l'on réalisera
- $w_t^{(i)}$ la note comprise entre 0 et 1 à l'itération $t \leqslant T$ pour l'observation $i \leqslant n$
- h_θ un weak learner d'AdaBoost paramétré par le vecteur d'information θ

La décision finale est prise par l'ensemble des T weak learners pondérés par leur score propre α_t que l'on définira plus tard. Ainsi, on peut résumer la fonction de prise de décision d'AdaBoost par :

1. Récompensé par le prix Gödel 5 ans après sa publication, le prix le plus prestigieux en informatique après le prix Turing.
2. On parle également d'époques.

Nombre d'époques

$$f_\theta(x) = \sum_{t=1}^{T} \alpha_t \, h_{\theta_t}(x) \qquad \text{(AdaBoost)}$$

Note du weak learner de l'époque t

Il suffira de prendre l'indicatrice que ce nombre est positif pour prédire la classe 1 par exemple. Il nous reste à comprendre comment apprendre pour chaque époque t la note α_t et quel est le problème que le weak learner h_{θ_t} va résoudre.

Comme annoncé, une des idées principales d'AdaBoost est de noter les observations du dataset. Autrement dit, on note la paire $(x^{(i)}, y_i)$ pour $i \leqslant n$ avec le poids $w_\cdot^{(i)}$ en reprenant les notations précédemment introduites. Le problème que l'on se posera à chaque époque $t \leqslant T$ est :

$$\theta_t = \arg\min_{\theta \in \Theta} \frac{\sum_{i=1}^{n} w_t^{(i)} \mathbb{1}_{\{y_i \neq h_\theta(x^{(i)})\}}}{\sum_{i=1}^{n} w_t^{(i)}}$$

On pondère les erreurs par les notes qui sont attribuées à chacune des observations, c'est ainsi que l'on arrive à forcer l'attention sur certaines observations. Les notes sont initialisées uniformément. Une fois le meilleur paramétrage du weak learner trouvé, on calcule l'erreur associée et donc sa note :

$$\begin{aligned} \varepsilon_t &= \frac{\sum_{i=1}^{n} w_t^{(i)} \mathbb{1}_{\{y_i \neq h_\theta(x^{(i)})\}}}{\sum_{i=1}^{n} w_t^{(i)}} \\ \alpha_t &= \frac{1}{2} \ln\left(\frac{1-\varepsilon_t}{\varepsilon_t}\right) \end{aligned}$$

La valeur de α_t va influer sur la mise à jour des notes $w_{t+1}^{\cdot}$. Mais avant de voir comment, décortiquons un peu plus la manière de noter le weak learner à l'époque t.

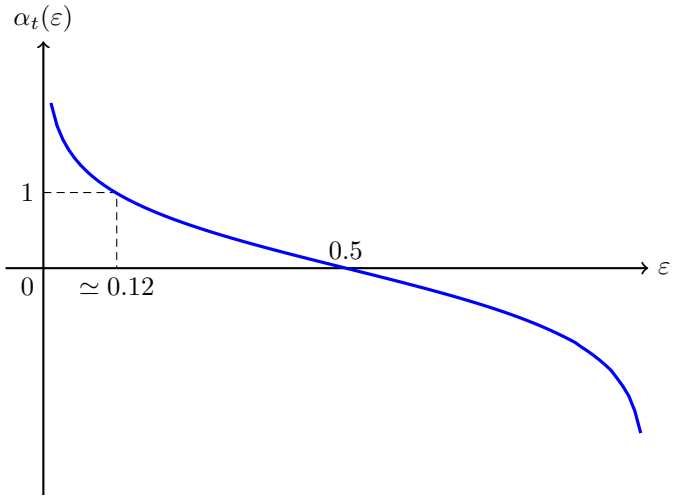


FIGURE 5.1 – Graphe de la fonction : $x \mapsto \frac{1}{2} \ln\left(\frac{1-x}{x}\right)$

Exercice 5.1 (Étude de α_t)**.** *Soit $t \leqslant T$ une époque, on s'appuiera sur la figure (5.1).*

1. *Montrer que $\varepsilon_t \in [0,1]$*
2. *Commenter la forme de fonction quand ε est au voisinage de 0.5. Même question pour 0 et pour 1.*

Solution. 1. Clairement $w_t^{(i)} \mathbb{1}_{\{y_i \neq h_\theta(x^{(i)})\}} \leqslant w_t^{(i)}$ pour tout $i \leqslant n$, d'où le résultat.

2. Lorsque ε est proche de 0.5 alors α_t est proche de 0. Autrement dit, quand le weak learner a des performances proches de l'aléatoire alors sa note est proche de zéro. C'est cohérent puisqu'on ne peut pas lui faire suffisamment confiance dans ses prédictions.
Quand ε est proche de 0 c'est l'inverse : le weak learner a réussi a bien discriminer les classes, on le note donc très fort pour suivre au maximum ses prédictions. A l'opposé, quand ε est proche de 1, le weak learner se trompe quasiment systématiquement, donc on a intérêt à suivre l'opposé de ce qu'il préconise.

□

Ainsi, en ayant détaillé le comportement de la note de l'algorithme, les notes des observations peuvent se mettre à jour de la manière suivante :

$$\forall i \leqslant n, \quad w_{t+1}^{(i)} = w_t^{(i)} \exp\left(-\alpha_t h_{\theta_t}\left(x^{(i)}\right)(2y_i - 1)\right)$$

On peut résumer ce que l'on vient de voir :

- Initialisation : Nombre d'époques T et initialiser les notes des observations
- Pour chaque époque :
  1. Trouver le meilleur paramétrage pour une souche dans un problème prenant en compte la difficulté de classification de chaque observation
  2. Calculer la note du weak learner appris
  3. Mettre à jour les notes des observations

Après cette présentation d'AdaBoost, il reste encore des choses à dire et on observe parfois des comportements étranges qui sont encore des problèmes non résolus à ce jour. Pour lire une introduction à ces challenge, voir l'annexe G.

AdaBoost est un bon exemple pour comprendre la différence entre la méthode du Boosting et la méthode du Bagging. Ici, chaque weak learner s'appuie sur le travail des précédents weak learners. Alors que dans le Bagging, chaque weak learner travaille de manière isolée, sans savoir ce que les autres *font*.

5.2 Gradient Boosting

Pour présenter le Gradient Boosting, considérons le dataset :

$$\mathcal{D} = \left\{(x^{(i)}, y_i) \mid \forall i \leqslant n,\ x^{(i)} \in \mathbb{R}^d, y_i \in \mathcal{Y}\right\}$$

Pour simplifier la lecture de cette partie, on adapte les notations utilisées jusqu'ici :

- On ne rend plus explicite la dépendance en θ de la fonction f
- On définit la fonction de perte $\mathcal{L} : \mathcal{Y} \times \mathcal{Y}$. Par exemple $\mathcal{L}(y, f(x)) = (y - f(x))^2$.

On reprend la notation h pour désigner un weak learner, et on note M le nombre de weak learners utilisés. On notera h_m pour désigner le m-ième weak learner. Pour apprendre le dataset $\mathcal{D}$, on se propose un premier modèle f_0 défini comme :

$$f_0 = \arg\min_{\gamma \in \mathbb{R}} \sum_{i=1}^{n} \mathcal{L}(y_i, \gamma)$$

On cherche la meilleure constante pour prédire l'ensemble du dataset.

Exercice 5.2 (Cas de la MSE)**.** *On considère un problème de régression. Résoudre le problème :*

$$\gamma^* = \arg\min_{\gamma \in \mathbb{R}} \sum_{i=1}^{n} (y_i - \gamma)^2$$

Plus généralement, quelles sont les propriétés analytiques que l'on souhaite avoir pour la fonction de perte $\mathcal{L}$?

Solution. En dérivant par rapport à γ, on obtient que $\gamma^* = \overline{y}$. □

Prédire un dataset par une constante n'est pas très performant. Donc on cherche à l'améliorer itérativement. Ainsi, on cherche à améliorer f_{m-1} à l'étape m de sorte que :

$$\begin{array}{rcl} \forall i \leqslant n,\ f_m\left(x^{(i)}\right) & = & f_{m-1}\left(x^{(i)}\right) + h_m\left(x^{(i)}\right) = y_i \\ & \iff & \\ \forall i \leqslant n,\ h_m\left(x^{(i)}\right) & = & y_i - f_{m-1}\left(x^{(i)}\right) \end{array}$$

On cherche à améliorer les modèles successivement en essayant d'apprendre les **résidus** du modèle précédent. À l'inverse du Bagging où chaque weak learners est indépendant, ici les weak learners apprennent itérativement et sont donc fortement liés.

Exercice 5.3 (Descente de gradient et résidus)**.** *Soit la fonction de perte $\mathcal{L}(y, f(x)) = (y - f(x))^2$ et la fonction de coût $\mathcal{C}(y, f(x)) = \sum_{i=1}^{n} \mathcal{L}\left(y_i, f\left(x^{(i)}\right)\right)$. Montrer que :*

$$-\frac{\partial \mathcal{C}}{\partial f_{m-1}\left(x^{(i)}\right)}\left(y_i, f_{m-1}\left(x^{(i)}\right)\right) = \frac{2}{n} h_m\left(x^{(i)}\right)$$

Le résultat de cet exercice se généralise, il y a un lien entre l'opposé du gradient de la fonction de coût et les résidus. Ainsi, si l'on compile les différentes équations que l'on a écrites jusqu'à présent on a :

$$\begin{array}{rcl} f_m(x) & = & f_{m-1}(x) - \gamma \sum_{i=1}^{n} \frac{\partial \mathcal{C}}{\partial f_{m-1}\left(x^{(i)}\right)}\left(y_i, f_{m-1}\left(x^{(i)}\right)\right) \\ & = & f_{m-1}(x) + \gamma' h_m(x) \end{array}$$

On reconnait clairement l'équation d'une descente de gradient, d'où le nom de Gradient Boosting. On peut optimiser la valeur de γ pour qu'elle prenne la valeur qui minimise la fonction de perte :

$$\gamma_m = \underset{\gamma \in \mathbb{R}}{\arg\min} \sum_{i=1}^{n} \mathcal{L}\left(y_i, f_{m-1}\left(x^{(i)}\right) + \gamma h_m\left(x^{(i)}\right)\right)$$

Nous avons réussi à trouver comment minimiser à chaque itération la fonction de coût. Mais avec une telle complexité, on s'expose à un overfitting fort. Ainsi, on exploite à nouveau la théorie de la descente de gradient pour définir un learning rate $\eta \in]0, 1]$. Finalement, on peut résumer le Gradient Boosting à :

$$f_0(x) = \underset{\gamma \in \mathbb{R}}{\arg\min} \sum_{i=1}^{n} \mathcal{L}(y_i, \gamma) \qquad \text{(Initialisation)}$$

$$\forall m \leqslant 1,\ f_m(x) = f_{m-1}(x) + \eta \gamma_m h_m(x) \qquad \text{(Itération)}$$

$$F(x) = \sum_{m=0}^{M} \gamma_m h_m(x) \text{ avec } \gamma_0 = 1 \qquad \text{(Strong learner)}$$

On peut résumer le principe du Gradient Boosting avec la figure (5.2).

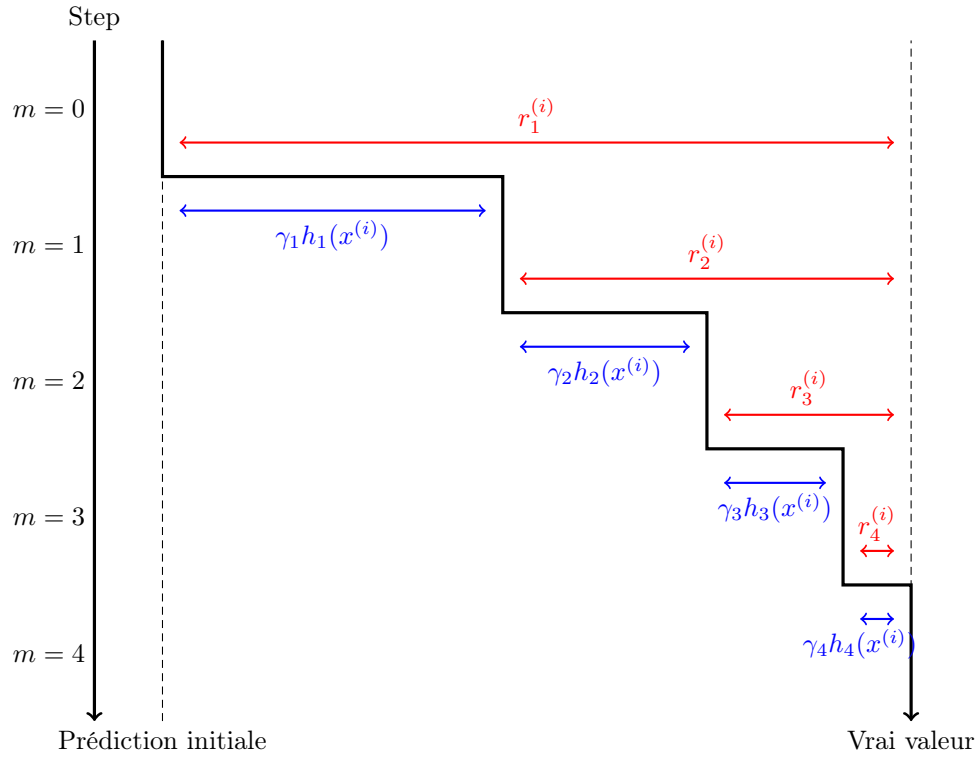


FIGURE 5.2 – Principe du Gradient Boosting pour une observation

On a une réduction des résidus à chaque étape grâce à une correction adaptative. Ici on a prit $\eta = 1$. Avoir un learning rate fort peut mener à beaucoup de sur-apprentissage mais permet de réduire le temps de calcul via le nombre d'itérations.
Empiriquement, on observe qu'avoir un learning rate faible nous permet d'avoir une bonne généralisation. Mais pour s'assurer d'avoir peu d'overfitting, nous avons plusieurs méthodes :

- Contrôler la complexité des weak learners

- Pénaliser les valeurs γ_m
- Limiter le nombre d'itérations
- Utiliser un early stopping : arrêter l'apprentissage quand il n'y a pas plus d'apprentissage

Nous avons présenté de manière générale le Gradient Boosting, mais en pratique les weak learners utilisés sont le plus souvent des arbres. C'est notamment le cas des algorithmes plus spécifiques que nous allons présenter ensuite.

5.2.1 Principaux algorithmes de Gradient Boosting

XGBoost [CG16] s'appuie sur des weak learners qui sont des arbres et apporte plusieurs améliorations majeures dans ce domaine. Si l'on reprend les explications de l'algorithme de Gradient Boosting classique, à chaque étape nous choisissons un arbre h_m qui répond au problème :

$$h_m^* = \underset{h_m \text{ possible}}{\arg\min} \sum_{i=1}^{n} \mathcal{L}\left(y_i, f_{m-1}\left(x^{(i)}\right) + h_m\left(x^{(i)}\right)\right) \tag{5.1}$$

Cela veut dire que nous allons, pour construire chaque arbre, calculer de nombreuses fois la valeur de la fonction de coût. En pratique, voici le problème qui est résolu :

$$h_m^* = \underset{h_m \text{ possible}}{\arg\min} \sum_{i=1}^{n} \left(y_i - \left(f_{m-1}\left(x^{(i)}\right) + h_m\left(x^{(i)}\right)\right)\right)^2 \tag{5.2}$$

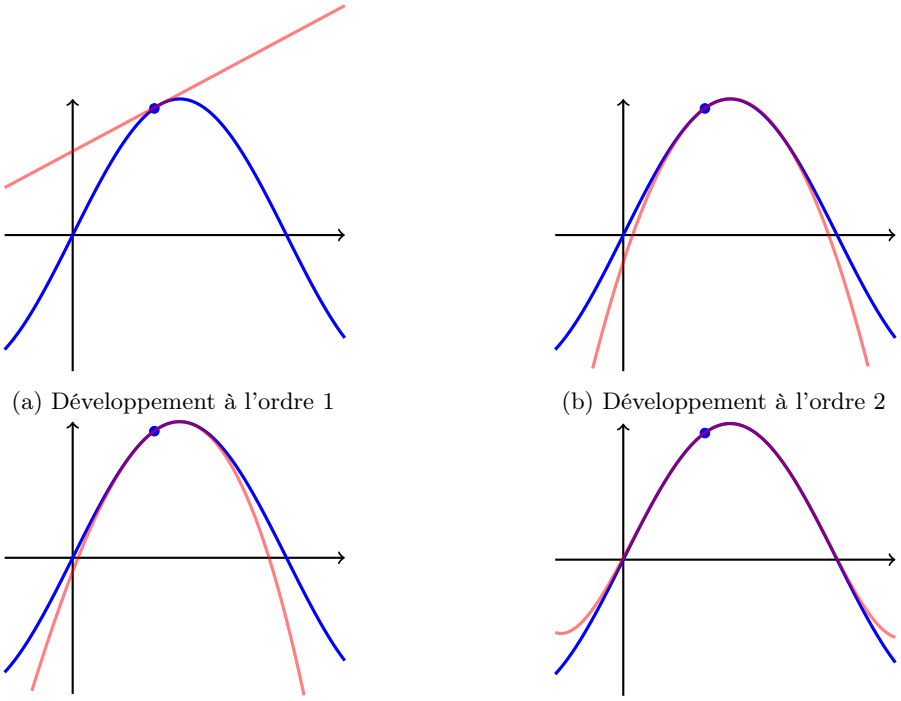
Le problème (5.2) est une approximation du vrai problème (5.1), donc nous n'optimisons pas vraiment ce que l'on souhaite. Mais cela reste une bonne approximation et cela fonctionne tout de même ! L'apport d'XGBoost est d'être plus précis dans l'approximation qui est réalisée, en résolvant un autre problème. Avant de voir laquelle, rappelons un résultat d'analyse.

Théorème 2 (Taylor). *Soit $I \subset \mathbb{R}$ et $a \in I$. Soit $f : I \mapsto \mathbb{R}$ une fonction n-fois dérivable en a. Alors :*

$$f(x) = \sum_{k=0}^{n} \frac{f^{(k)}(a)}{k!}(x-a)^k + R_n(x)$$

Avec le reste $R_n(x)$ négligeable devant $(x-a)^n$ au voisinage de a.

Ce théorème permet d'approcher une fonction au voisinage d'un point avec un polynôme. C'est particulièrement intéressant si nous devons travailler localement avec des fonctions compliquées ou lourde à évaluer. Cette manière d'exprimer une fonction est appelée un développement de Taylor à l'ordre n. On peut visualiser ce résultat avec la figure (5.3).



(a) Développement à l'ordre 1

(b) Développement à l'ordre 2

(c) Développement à l'ordre 3

(d) Développement à l'ordre 4

FIGURE 5.3 – Développement de Taylor pour $f(x) = 2\sin(x)$ au point $x = 1.3$

On voit clairement qu'avec l'ordre d'approximation qui augmente, le voisinage est de plus en plus important. Cela s'explique avec une autre notion d'analyse (les séries) et rappelle également le tradeoff biais-variance ! Avec cette idée que l'on peut approcher une fonction complexe avec un polynôme d'ordre arbitraire localement, nous pouvons proposer une meilleure approximation avec l'exercice (5.4).

Exercice 5.4 (Nouvelle fonction de coût)**.** *Nous reprenons l'ensemble des notations définies jusqu'à présent.*

1. *Soit $f : \mathcal{I} \mapsto \mathbb{R}$ une fonction n fois dérivable, $a \in I$ et $h \in \mathbb{R}$ tel que $a + h \in I$. Justifier :*

$$f(a+h) = \sum_{k=0}^{n} \frac{f^{(k)}(a)}{k!} h^k + R_n(h)$$

   Avec $R_n(x)$ une fonction négligeable devant h^n au voisinage de 0.

2. *A l'aide de l'expression précédente, proposer une approximation à l'ordre 2 de l'expression :*

$$\phi\left(f_{m-1}\left(x^{(i)}\right) + h_m\left(x^{(i)}\right)\right) = \sum_{i=1}^{n} \mathcal{L}\left(y_i, f_{m-1}\left(x^{(i)}\right) + h_m\left(x^{(i)}\right)\right)$$

   Où $\mathcal{L}$ est une fonction dérivable deux fois sur $\mathbb{R}$.

3. *Nous obtenons une approximation du problème du choix du meilleur weak learner h_m. Identifier les termes constants et commenter sur la vitesse de calcul par rapport à la méthode classique.*

Solution. 1. En exploitant le théorème (2) de Taylor avec le changement de variable de x en $a+h$, on obtient le résultat.

2. Avec la question précédente, et la fonction proposée, on obtient :

$$h_m^* = \underset{h_m \text{ possible}}{\arg\min} \sum_{i=1}^{n} \mathcal{L}\left(y_i, \hat{y}_i^{(m-1)}\right) + \nabla\mathcal{L}\left(y_i, \hat{y}_i^{(m-1)}\right) h_m\left(x^{(i)}\right) + \frac{1}{2}\nabla^2\mathcal{L}\left(y_i, \hat{y}_i^{(m-1)}\right) h_m\left(x^{(i)}\right) \tag{5.3}$$

3. En reprenant l'expression précédente et en écrivant en rouge les termes constants pour chacune des itérations, on a :

$$h_m^* = \underset{h_m \text{ possible}}{\arg\min} \sum_{i=1}^{n} \mathcal{L}\left(y_i, \hat{y}_i^{(m-1)}\right) + \nabla\mathcal{L}\left(y_i, \hat{y}_i^{(m-1)}\right) h_m\left(x^{(i)}\right) + \frac{1}{2}\nabla^2\mathcal{L}\left(y_i, \hat{y}_i^{(m-1)}\right) h_m\left(x^{(i)}\right)$$

Ainsi, par rapport à la manière naïve du Gradient Boosting classique, nous avons pour chacune des itérations l'essentiel des termes à évaluer qui deviennent des constantes. Nous allons donc évaluer bien moins souvent certaines fonctions lourdes et donc gagner du temps de calcul.

□

Avec cet exercice nous voyons que la fonction qui sera optimisée pour construire les arbres est encore une approximation (5.3) : mais meilleure que celle de l'implémentation naïve. Ces optimisations permettent à XGBoost d'être un algorithme très performant et rapide d'exécution.

Dans le même esprit, [KMF+17] introduit un des algorithmes concurrents à XGBoost : LightGBM. Il présente deux nouveautés pour permettre un apprentissage beaucoup plus rapide que XGBoost en plus d'une méthode différente d'apprentissage. Au lieu de construire les arbres par niveau, l'arbre est développé feuille par feuille en fonction du gain le plus important. Même si l'on peut obtenir le même arbre à la fin, la vitesse de calcul n'est pas la même.

> *Gradient Boosting Decision Tree (GBDT) is a popular machine learning algorithm, and has quite a few effective implementations such as XGBoost and pGBRT. Although many engineering optimizations have been adopted in these implementations, the efficiency and scalability are still unsatisfactory when the feature dimension is high and data size is large. A major reason is that for each feature, they need to scan all the data instances to estimate the information gain of all possible split points, which is very time consuming. To tackle this problem, we propose two novel techniques : Gradient-based One-Side Sampling (GOSS) and Exclusive Feature Bundling (EFB)*
>
> — Guolin Ke et al. (2017)

La première astuce qui permet à LightGBM d'être plus rapide est le nombre de prises en compte des gradients d'erreurs. En effet, si un gradient est de valeur faible, c'est que l'apprentissage est bon, donc qu'il faut se concentrer sur les gradients les plus élevés. C'est le principe de *Gradient Based One Side Sampling* (GOSS). L'idée est de conserver les $a\%$ gradient les plus élevés et un sous ensemble de $b\%$ des gradients plus faibles restants. Ainsi, le calcul du gain qui consiste à faire une coupure à un certain noeud pour une certaine information est beaucoup plus rapide.

La seconde astuce est de réduire la dimension des données d'entrée en fusionnant des informations qui sont mutuellement exclusives par exemple. C'est l'*Exclusive Feature Bundling Technique* on réduit à nouveau le nombre de tests de coupure à faire. Les hyperparamètres de LightGBM sont dans le même ton que ceux de XGBoost et on renvoie le lecteur vers les documentations respectives des algorithmes pour être plus exhaustif.

Finalement, CatBoost [DEG18] est le dernier grand algorithme de Boosting. A nouveau des améliorations ont été développées pour réduire le temps de calcul. L'une d'elles est le *Minimal Variance Sampling*

(MVS) qui réduit le nombre nécessaire pour sélectionner la meilleure coupure. On y gagne en temps mais également en performance. On peut trouver les détails techniques dans [IG19].

Pour chacun des algorithmes, les principaux hyperparamètres sont :

- Pour paramétrer les arbres :
  - criterion : pour définir la métrique à utiliser pour faire une coupure
  - max_depth : limiter la profondeur maximale d'un arbre
  - min_samples_leaf : nombre minimal d'observations dans une feuille
  - max_features : nombre d'informations à considérer pour chaque coupure
- Pour paramétrer le boosting :
  - n_estimators : nombre d'arbres à construire dans la forêt
  - learning_rate : pas de descente pour réduire le poids des arbres successifs
  - subsample : fraction des données à utiliser pour apprendre chaque weak learner. Si inférieur à 1, alors on obtient une descente de Gradient Stochastique
  - init : premier modèle qui sera amélioré. Si non renseigné, un modèle très simple sera utilisé
- Pour arrêter plus tôt le boosting :
  - validation_fraction : proportion des données d'entraînement à conserver pour tester l'early-stopping
  - n_iter_no_change : nombre minimal d'itérations sans améliorations avant d'arrêter l'apprentissage
  - tol : valeur minimale de modification de la loss qui déclenche l'arrêt prématuré

Cette liste n'est pas exhaustive et ne tient pas compte des particularités de chacun des algorithmes, on renvoie donc le lecteur aux documentations officielles. Chaque hyperparamètre permet de jouer sur la puissance du modèle et permet également de prévenir l'overfitting. C'est l'intégration de la régularisation, en plus des nombreuses astuces d'optimisation en temps qui font de cet algorithme l'un des plus utilisés au monde.

Les méthodes ensemblistes et de boosting tirent leurs forces d'un grand nombre de weak learners. Ce qui fait une distinction fondamentale entre ces deux méthodes et l'indépendance ou non des weak learners. Les méthodes ensemblistes comme les Random Forest apprennent en parallèle chacun des arbres, là où une méthode de boosting apprend chacun des weak learners de manière séquentielle : en fonction de la performance du précédent weak learner.

Chapitre 6

Clustering

Il existe de nombreuses études et tests psychologique qui visent à assigner une étiquette à chaque être humain. Cambridge Analytica est une société anglaise créé en 2013 qui a exploité cette idée pour identifier des groupes de personnes qui seraient susceptibles de voter pour un candidat plutôt qu'un autre. Après des essais de cette méthode dans des élections en Inde ou en Afrique du Sud (soupçonné), c'est à la campagne présidentielle des États-Unis que sera utilisé ses avancés analytiques en faveur du président élu Donald Trump. Le scandale vient de l'exploitation d'une fuite de données de Facebook vers Cambridge Analytica pour réaliser cette manipulation ciblée.

Ici n'avons pas accès à un dataset qui nous permet de savoir avec certitude si un candidat va voter pour Donald Trump ou non. En revanche, nous sommes capables d'identifier des personnes qui vont le faire très certainement et d'autres non. Puis nous sommes capables à l'aide d'étude psycho-sociale et à l'aide des informations rendues publique sur Facebook par les utilisateurs de les regrouper en typologie d'électeur. Ainsi, nous pouvons identifier des personnes à l'intérieur de groupe votant massivement pour Donald Trump et donc les cibler pour s'en assurer.

La partie de regroupement des individus se distingue nettement de l'approche supervisée que nous avons étudiée jusqu'à présent : nous n'avons pas accès une labellisation parfaite ! Il s'agit de l'apprentissage non-supervisé où l'on cherche à regrouper les vecteurs $x_i \in \mathbb{R}^d$ pour $i \leqslant n$ qui se *ressemblent*. Nous verrons dans un premier temps ce que l'on entend par *se ressembler* puis deux approches différentes pour répondre à la problématique de l'apprentissage non supervisé.

6.1 Distance : ce qui se ressemble est proche

Intuitivement, nous disons que deux objets qui se ressemblent doivent être proches selon plusieurs critères. Evidemment, on se ressemble parfaitement à soi-même, et si deux objets se ressemblent il ne doit pas y en avoir un qui ressemble plus à l'autre. C'est exactement le début d'une distance en mathématiques. Plus formellement :

Définition 2. *Une métrique pour un ensemble M est une fonction $d : M \times M \to \mathbb{R}_+$ telle que pour tout $x, y, z \in M$:*

1. ***Indiscernabilité*** *:* $d(x, y) = 0 \iff x = y$
2. ***Symétrie*** *:* $d(x, y) = d(y, x)$
3. ***Sous-additivité*** *:* $d(x, z) \leqslant d(x, y) + d(y, z)$

Les deux premières conditions ont été souhaitées juste avant, mais pas la dernière. Nous en avons besoin, et cette condition s'appelle également l'inégalité triangulaire. Pour la comprendre, considérons trois endroits. Une distance ne sera cohérente que si pour aller d'un endroit à un autre la mesure est plus

faible que de faire un détour par le dernier endroit !

Les distances les plus classiques sont de la famille [1] $\mathcal{L}_p$ et sont de la forme $d(x,y) = \left(\sum_{i=1} \|x_i - y_i\|^p\right)^{\frac{1}{p}}$. La distance Manhattan en est un cas particulier avec $p = 1$ et la distance la plus classique est la distance euclidienne avec $p = 2$.

On doit garder en tête que toutes les métriques de cette famille de la forme souffrent grandement du fléau de la dimension [2]. Le meilleur conseil que l'on puisse donner est donc de rester autant que possible avec relativement peu d'indicateurs par rapport au nombre d'observations que l'on a à disposition.

6.2 Approche statistique

Une première manière de traiter le problème est d'exploiter le domaine des statistiques. L'algorithme des K-Means qui est de loin le plus utilisé en pratique en est un représentant.

6.2.1 K-Means

L'algorithme K-Means vise à partitionner l'espace des features en K clusters où chaque observation appartient au cluster avec la distance la moyenne la plus faible.

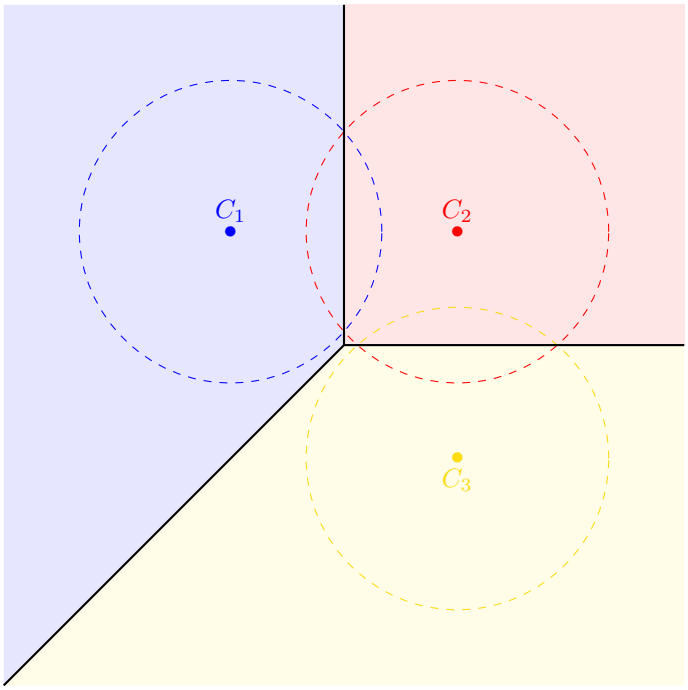


FIGURE 6.1 – Exemple d'un clustering avec K-Means pour $K = 3$ clusters

Dans la figure (6.1) il faut bien comprendre que la partition de l'espace est représentée par les trois espaces colorés. Les cercles ne représentent pas les clusters, mais donnent une idée de la concentration des données autour du centre. L'algorithme K-Means ne renvoie pas des cercles, mais des cellules de Voronoi

1. Cette notation vient de la théorie de la mesure, une magnifique théorie qui généralise la notion d'intégrale à d'autres espaces et qui, au prix d'une introduction très théorique, permet de démontrer des résultats remarquables dans de nombreux domaines des mathématiques plus simplement.
2. Voir l'annexe (D).

(les espaces colorés) par construction.

Pour exploiter l'algorithme K-Means, il faut spécifier une distance $d : \mathbb{R}^d \to \mathbb{R}_+$ et un nombre de clusters K que l'on souhaite obtenir. On définit les notations :

- C_k le k-ième cluster de coordonnées $\mu_k \in \mathbb{R}^d$
- $\mu \in \mathcal{M}_{d,K}$ la matrice engendrée par les $(\mu_k)_{k \leqslant K}$
- $z_i^k = \mathbb{1}_{\{x_i \in C_k\}}$ et $z \in \mathcal{M}_{n,K}$ la matrice engendrée par les $(z_i^k)_{i \leqslant n}^{k \leqslant K}$

Avec ces notations, on définit la distortion comme :

Nombre d'observations — Nombre de clusters

$$J(\mu, z) = \sum_{i=1}^{n} \sum_{k=1}^{K} z_i^k \|x_i - \mu_k\|^2 \qquad \text{(Distortion)}$$

On cherche les meilleurs $(\mu_k)_{k \leqslant K}$ qui permettent de minimiser J :

$$\mu = \underset{\mu \in \mathcal{M}_{d,K}}{\arg\min}\, J(\mu, z)$$

On décrit la procédure d'apprentissage de l'algorithme K-Means dans l'algorithme (1).

Algorithm 1: Procédure d'entraînement de l'algorithme K-Means

for $k \in \{1, \dots, K\}$ **do**
  μ_k = random value
while μ *does not converge* **do**
  for $i \in \{1, \dots, n\}$ **do**
    for $k \in \{1, \dots, K\}$ **do**
      $z_i^k = \mathbb{1}_{d(x_i, \mu_k) = \arg\min_{s \in \{1, \dots, K\}} d(x_i, \mu_s)}$
  for $k \in \{1, \dots, K\}$ **do**
    $$\mu_k = \frac{\sum_{i=1}^{n} x_i z_i^k}{\sum_{i=1}^{n} z_i^k}$$

Se posent alors plusieurs questions :

1. Est-ce que l'algorithme converge assurément ? Comment est-on sûr que l'on minimise bien J ?
2. Comment choisir efficacement le nombre K ?

On pourra répondre à la seconde question plus tard. Pour répondre à la première question on peut résoudre l'exercice :

Exercice 6.1 (Convergence de l'algorithme K-Means)**.** *On reprend les notations précédentes.*

1. *Montrer que :*
$$\frac{\partial J}{\partial \mu_k}(\mu, z) = -2 \sum_{i=1}^{n} z_i^k (x_i - \mu_k)$$
2. *Conclure sur la convergence de J.*

Solution. Pour résoudre la première question, il suffit de dériver J par rapport à μ_k. Et de cette équation on déduit :

$$\mu_k = \frac{\sum_{i=1}^{n} x_i z_i^k}{\sum_{i=1}^{n} z_i^k}$$

Qui est exactement l'actualisation de la valeur μ_k présentée dans la procédure d'apprentissage. On comprend donc qu'on minimise bien J. Pas au sens d'une descente de gradient puisqu'ici on ne descend pas d'une partie du gradient mais on descend pour chaque coordonnées jusqu'à l'intersection avec l'axe des abscisses.
L'optimisation est donc locale, on ne peut pas garantir qu'un minimum sera un minimum global. □

Se pose maintenant la question de l'initialisation, qui va influer sur la vitesse de convergence de l'algorithme et la convergence. Au départ nous utilisions plusieurs fois l'algorithme avec des vecteurs de départs aléatoires et on conserve la partition qui minimise le plus la distortion.

6.2.2 Kmeans++ : un meilleur départ

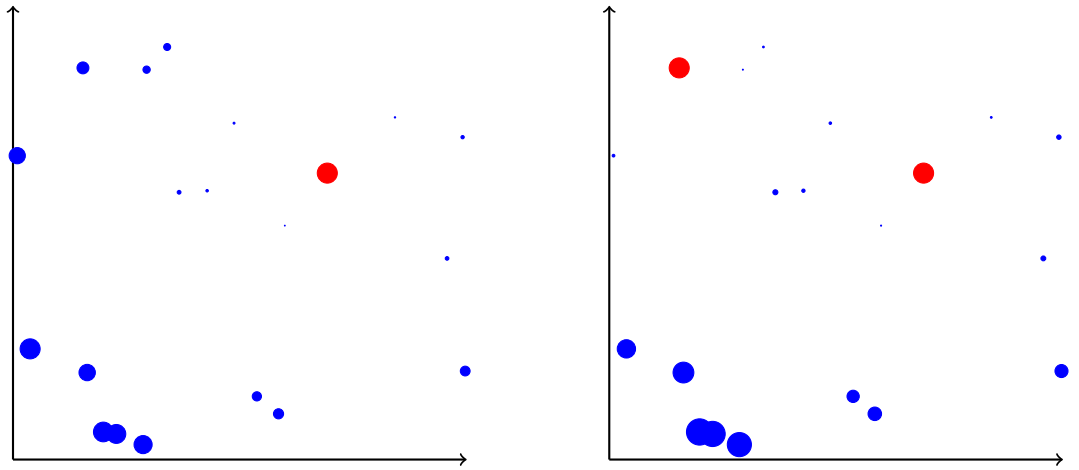
Suivre cette méthode, nous expose à des problèmes théoriques de convergence, qu'on rencontre en pratique. C'est pour cela qu'en 2006 David Arthur et Sergei Vassilvitskii propose une nouvelle manière de choisir les centres initiaux dans le rapport technique [AV06].

L'idée est de construire et étendre les centres de proche en proche. La procédure est présentée avec l'algorithme (2).

Algorithm 2: Procédure d'apprentissage de l'algorithme K-Means++

Sélectionner un centre parmi les points à clusteriser, de manière uniforme
for $k \in \{2, \dots, K\}$ **do**
  for $x \in \{x_1, \dots, x_n\}$ *tel que* x *n'est pas un centre de cluster* **do**
    $D(x) =$ distance entre x et le cluster le plus proche
  Choisir un point x aléatoirement suivant une distribution proportionnelle à $D(x)^2$ pour être un cluster
Réaliser un K-Means classique avec les clusters choisis

On peut visualiser ce qui est fait lorsqu'un centre est choisi par l'algorithme avec la figure (6.2).



(a) Sélection du deuxième centre

(b) Sélection du troisième centre

FIGURE 6.2 – Distribution proportionnelle à la distance au carré des centres pour plusieurs itérations

On voit clairement avec la figure (6.2) les différences de distributions et l'intuition derrière la méthode. Dans la première figure, le deuxième centre est à choisir, et on voit que les points proches du centre sont très peu considérés donc augmentent la *couverture* de l'espace. Cela assure que l'initialisation des centres ne les placera pas tous proches les uns des autres.

6.3 Approche par densité

L'approche par densité nécessite également d'avoir une métrique pour mesurer la distance entre deux points d'un dataset. Ce qui la distingue de l'approche statistique que nous avons présentée est la nécessité de deux paramètres supplémentaires :

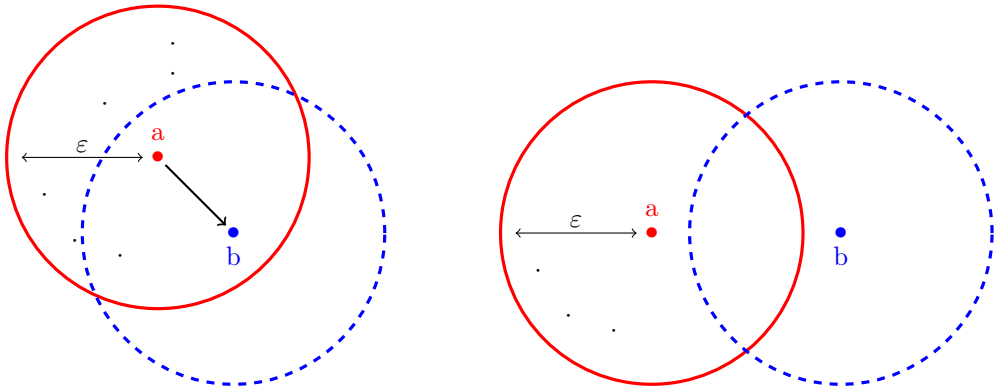
- ε : un seuil, qui sera utilisé pour décider de la proximité entre deux objets
- $MinPts$: un nombre, le minimum d'objets dans le voisinage d'un point pour que ce point soit considéré comme un point central

On notera le voisinage d'un objet $x \in M$ par $N_\varepsilon(x) = \{y \in M | d(x,y) \leqslant \varepsilon\}$ et $\#N_\varepsilon(x)$ le nombre de voisin, où $\#$ correspond au cardinal de l'ensemble considéré.

Définition 3. *Un objet b est directement atteignable depuis un objet a dans un ensemble d'objets D si :*

1. $b \in N_\varepsilon(a)$
2. $\#N_\varepsilon(a) \geqslant MinPts$

Pour comprendre visuellement cette définition :



(a) b est directement atteignable depuis a (b) b n'est pas directement atteignable depuis a

FIGURE 6.3 – Objet directement atteignable

Dans le cas présenté dans la figure (6.3b), b n'est pas directement atteignable depuis a parce que les deux conditions de la définition ne sont pas respectées. b n'est ni dans le voisinage de a et a n'est pas un point central.
On imagine facilement comment appliquer cette définition plusieurs fois pour traduire le fait que deux points sont connectés *par densité*. C'est l'objet de la définition :

Définition 4. *Un objet b est atteignable par densité depuis un objet a dans un ensemble d'objets D s'il existe une chaîne d'objets $o_0, \ldots, o_{n-1}$ telle que $o_0 = a$ et $o_{n-1} = b$ et que pour tout $i \leqslant n-1, o_i \in D$ et o_{i+1} est directement atteignable depuis o_i*

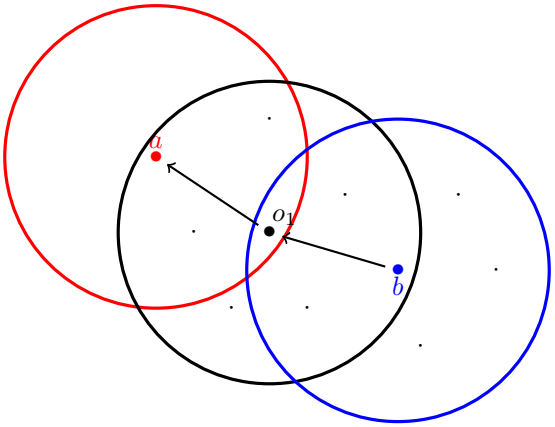


FIGURE 6.4 – Exemple de point atteignable par densité

Avec la figure (6.4) on voit que b est atteignable par densité depuis a, mais l'inverse n'est pas vrai. On peut donc définir ce que cela veut dire que deux objets soient mutuellement atteignables par densité :

Définition 5. *Un objet a est connecté par densité à un objet b dans un ensemble D s'il existe un objet $o \in D$ tel que a et b soit tous les deux atteignables par densité depuis o.*

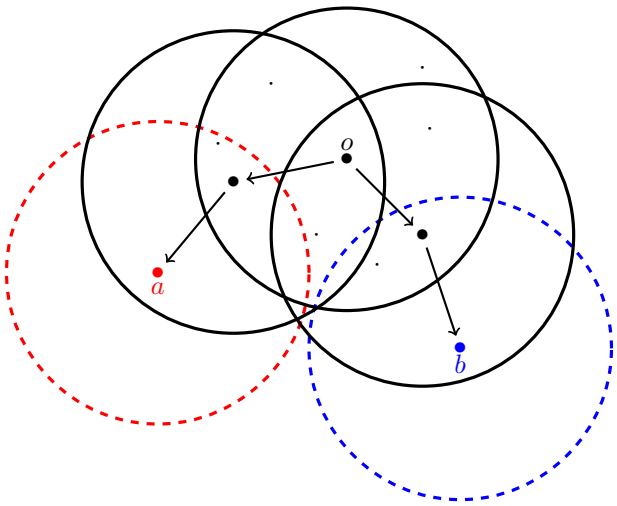


FIGURE 6.5 – Exemple d'objet connecté par densité

Dans la figure (6.5), a et b sont connectés par densité. Nous avons maintenant tous les outils pour parler de **cluster**.

Définition 6. *Soit D un ensemble d'objets. Un cluster C est un sous-ensemble non vide de D qui vérifie :*

1. ***Maximalité** : pour tout $a, b \in D$, si $b \in C$ et que b is atteignable par densité depuis a, alors $a \in C$*
2. ***Connectivité** : pour tout $a, b \in C, b$ est connecté par densité avec a*

*Tous les objets de D qui ne sont contenus dans aucun cluster sont regroupés dans un cluster qu'on appelle le **bruit**.*

Avec cette définition d'un cluster, on comprend que chaque objet présent dans un cluster est soit un point central soit un point frontière (*i.e.* pas un point central, mais un point atteignable par densité depuis un point central). Nous avons maintenant les outils nécessaires pour introduire l'algorithme DBSCAN.

6.3.1 DBSCAN

DBSCAN a été présenté dans l'article [EKS+96] en 1996, et dans ce papier se trouve l'ensemble des définitions que nous avons présentées. L'algorithme décrit peut s'écrire sous la forme :

Algorithm 3: DBSCAN

```
cluster = 0;
for point ∈ database do
    if label(point) ≠ undefined then continue ;
    neighbors = range_query(database, metric, point, eps);
    if |neighbors| < min_pts then
        label(point) = -1;                    // Cluster -1 is the cluster of noise objects
        continue ;
    cluster = cluster + 1 ;
    label(point) = cluster ;
    neighbors = neighbors - point ;
    for neighbor in neighbors do
        if label(neighbor) = -1 then label(neighbor) = cluster;
        if label(neighbor) ≠ undefined then continue ;
        label(neighbor) = cluster;
        local_neighborhood = range_query(database, metric, neighbor, eps);
        if |local_neighborhood| ⩾ min_pts then neighbors = neighbors ∪ local_neighborhood;
```

Avec la fonction :

Algorithm 4: Neighborhood query

```
Function range_query(database, metric, point, eps) :
    neighborhood = ∅;
    for observation ∈ database do
        if metric(point, observation) ⩽ eps then neighborhood = neighborhood ∪ observation;
return neighborhood
```

On comprend à quel point le choix de la métrique est fondamental dans l'algorithme DBSCAN : il définit la géométrie du problème. On comprend également le fonctionnement de DBSCAN : il va chercher un point central et définir son voisinage. Puis il va inspecter son voisinage pour étendre avec des points centraux ce cluster, et il le fera tant qu'il ne peut plus ajouter de points à ce cluster. L'algorithme répétera la procédure jusqu'à ce qu'il ne puisse plus créer de cluster, et les points restants seront labellisés comme du bruit.

On sent également une limite. Puisque DBSCAN ne travaille qu'avec un unique seuil ε, on peut tout à fait avoir des clusters à l'intérieur d'un cluster, donc un changement de densité. L'algorithme ne saura pas reconnaître ce changement de densité. C'est pour cela qu'OPTICS a été présenté.

6.3.2 OPTICS

Voyons un exemple de la remarque précédente avec la figure 6.6.

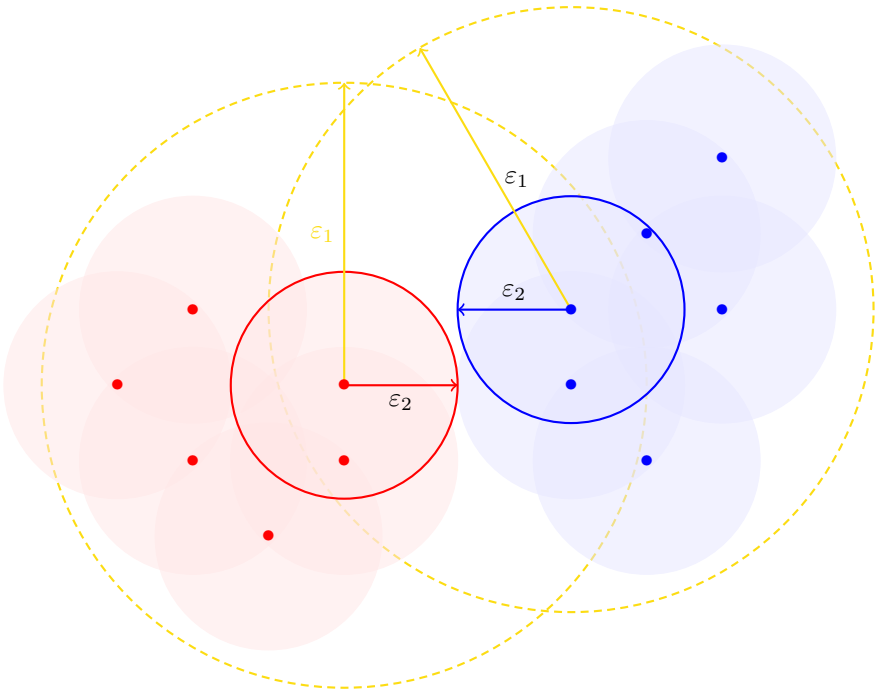


FIGURE 6.6 – Deux clustering différents pour deux valeurs de ε

Si nous avions choisi ε_1 jaune alors nous aurions eu un unique cluster jaune ici. Or, en prenant ε_2 noir, nous obtenons 2 clusters différents : un rouge et un bleu. Bien que nous voyions ici clairement le problème, il n'est peut être pas évident de choisir ε_2 comme valeur quand on regarde la totalité de la base, nous n'avons montré ici qu'un cluster qui pose problème.
Finalement, il serait agréable d'avoir des valeurs de ε différentes pour détecter ces densités particulières à l'intérieur d'un cluster. C'est tout l'enjeu de l'algorithme OPTICS présenté en 1999 (soit 3 ans après DBSCAN) par Mihael Ankerst, Markus Breunig, Hans-Peter Kreigel et Jörg Sander dans l'article *OPTICS : Ordering points to identify the clustering structure* [ABKS99].

Comme le fait comprendre le titre de l'article, l'idée est d'ordonner la totalité des points de la base. Pour le faire, on introduit deux nouvelles notions :

Définition 7 (Distance au centre)**.** *Soit a un objet dans un ensemble D et soit $\varepsilon > 0$. On définit $MinPts - distance : D \mapsto \mathbb{R}^+$ une fonction qui donne la distance entre a et son MinPts-voisin. On définit alors la distance au centre de a comme égale à $MinPts - distance$ si le voisinage de a contient plus d'éléments que MinPts, elle n'est pas définie sinon.*

La distance au centre d'un objet a est donc simplement la plus petite distance ε' entre a et un objet de son ε-voisinage de sorte que a resterait un point central pour son ε'-voisinage.

Définition 8 (Distance d'atteinte)**.** *Soit a et b deux objets dans un ensemble D. Alors, la distance d'atteinte de a par rapport à b est définie comme le maximum entre la distance au centre de b et la distance entre a et b, à condition que le ε-voisinage de b contienne plus que MinPts éléments.*

Ainsi, la distance d'atteinte d'un objet a par rapport à un autre objet b est la plus petite distance telle que p est atteignable par densité depuis b si b est un point central.

Avec la figure (6.7) on visualise les deux notions que l'on vient de présenter. Avec ces deux nombres, pour chaque vecteur nous sommes capable de les ordonner par la distance au centre. Puisque la distance d'atteinte dépend d'un point central depuis lequel on calcule, il a été décidé de ne stocker que la plus

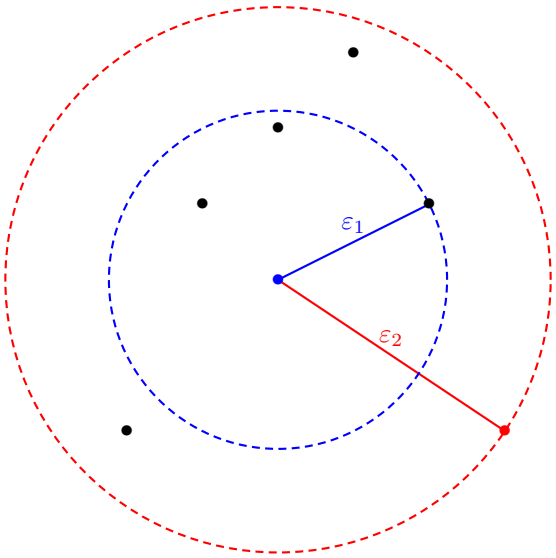


FIGURE 6.7 – Distance au centre et Distance d'atteinte pour l'objet rouge

petite des distances d'atteinte pour chaque point.

Ainsi, nous sommes capables de générer, pour une base de données, un graphique qui représente pour chaque point sa distance d'atteinte.

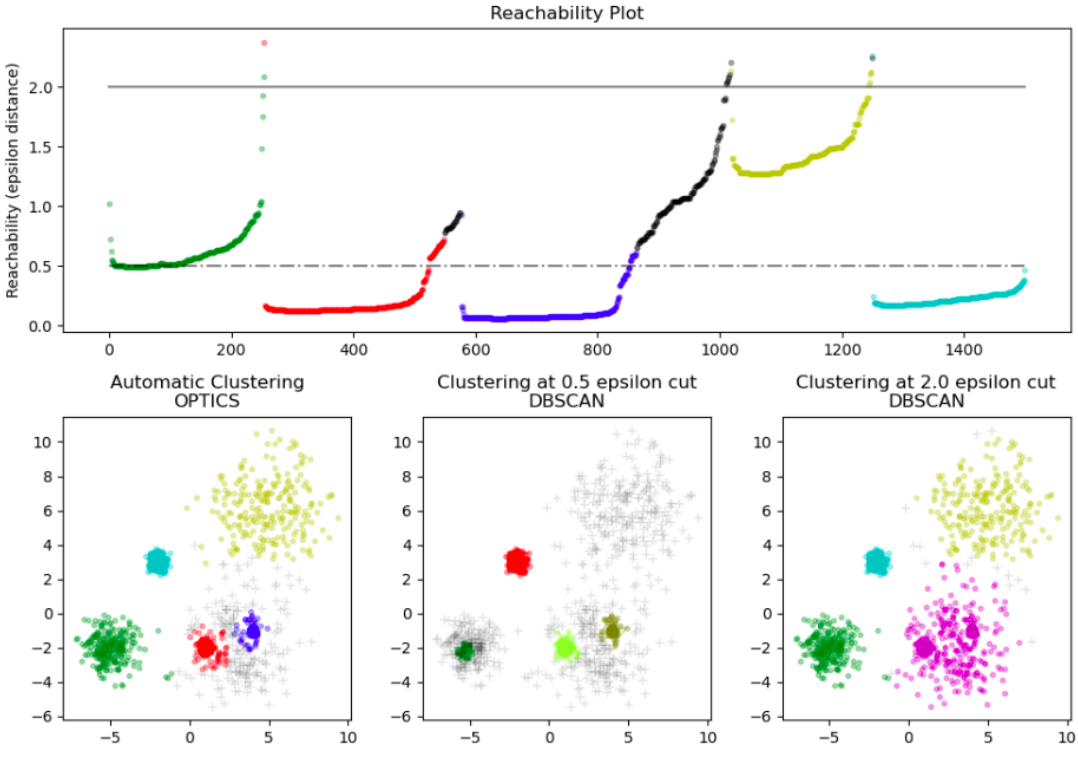


FIGURE 6.8 – Illustration d'OPTICS par scikit-learn

On comprend avec la figure (6.8) l'exploitation pratique que l'on peut faire du graphique *Reachability Plot* en proposant plusieurs seuils. A noter qu'il est nécessaire par la suite d'exploiter un algorithme DBSCAN pour extraire les clusters depuis ce graphique.

Nous venons de décrire deux algorithmes (liés) de clustering en prenant une approche par densité. Avec la présentation de l'algorithme K-Means, ceci conclut la présentation des grands algorithmes de clustering, mais ce n'est pas une présentation exhaustive.

Il nous reste maintenant à comprendre comment nous pouvons, à l'image de l'apprentissage supervisé, mesurer les performances d'un algorithme de clustering.

6.4 Mesures de performance

À la différence de l'apprentissage supervisé, nous n'avons pas accès à un vecteur nous donnant la réponse attendue. Ainsi, il est difficile de mesurer la qualité d'un clustering. Néanmoins, il existe quelques métriques qui sont largement utilisées. Elles s'appliquent pour les deux approches, statistique et à densité, et nous verrons les limites de ces métriques.

6.4.1 Silhouette score

Le silhouette score est présenté en 1987 [Rou87], et nécessite pour chaque observation x_i le calcul de deux nombres :

- a_i : la distance moyenne entre x_i et les autres points du cluster
- b_i : la distance moyenne entre x_i et les autres points du cluster le plus proche

On calcule, à l'aide des deux précédentes valeurs un score s_i :

$$s_i = \frac{b_i - a_i}{\max(a_i, b_i)}$$

Finalement, le silhouette score est définie comme :

$$S = \frac{1}{n} \sum_{i=1}^{n} s_i$$

Pour bien saisir les définitions :

Exercice 6.2 (Silhouette score)**.** *On s'intéresse à l'interprétation du silhouette score S comme défini précédemment.*

1. *Montrer que $\forall i \leqslant n,\ s_i \in [-1, 1]$*
2. *En déduire que $S \in [-1, 1]$*
3. *Quelle information nous est donnée quand S est proche de 1 ? Même question pour 0 et -1.*

Solution. 1. Soit $i \leqslant n$, par définition a_i et b_i sont des moyennes de distances, donc positives ou nulles. Donc on a clairement que $b_i - a_i \in [-\max\{a_i, b_i\}, \max\{a_i, b_i\}]$ d'où le résultat.

2. Direct avec la question précédente.

3. Si S est proche de 1, c'est que l'on a beaucoup de s_i qui sont proches de 1. Ainsi, b_i est suffisamment *grand* pour prendre le pas sur a_i et donc faire tendre s_i vers 1. Autrement dit, les clusters sont bien définis et suffisamment distants les uns des autres.
   Il suffit de faire le raisonnement inverse pour l'interprétation du score proche de -1. On en conclut que les clusters sont mal définis donc que le clustering n'a pas réussi à capturer la structure sous-jascente du dataset.
   Si S est proche de zéro, les coefficients s_i sont soit tous proches de 0 soit ils s'annulent mutuellement. Dans les deux cas, cela veut dire qu'il y a des clusters mal définis.

□

Grâce à sa simplicité d'explication, qu'il soit borné et l'idée générale du silhouette score fait qu'il s'agit de la métrique la plus utilisée pour mesurer la qualité d'un clustering. Et pourtant, ce score est connu pour favoriser les clusters ayant des formes convexes. Ainsi, le score sera plus élevé pour un scoring type K-Means qu'un clustering par densité. Prenons un exemple classique et connu :

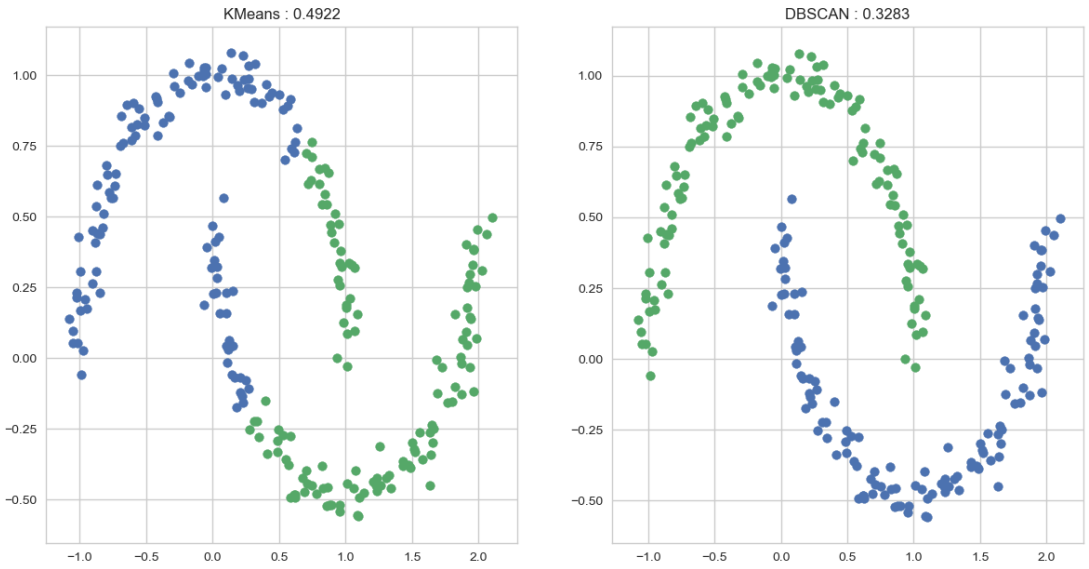


FIGURE 6.9 – Comparaison de la valeur du silhouette score entre deux méthodes de clustering

L'algorithme DBSCAN a clairement mieux capturé la forme globale des deux groupes, et pourtant l'algorithme K-Means obtient un meilleur score. Puisque les cellules de Voronoi formées par K-Means sont convexes par définition, le silhouette score est plus élevé bien que les clusters ne soient pas *corrects*.

6.4.2 L'index de Calinksi-Harabasz

Également connu sous le nom de *Variance Ratio Criterion*, ce score cherche aussi les clusters les mieux définis. Mais cette fois, il utilise un point de vue plus statistique en regardant les choses avec et sans le cluster.
Pour définir l'index de Calinksi-Harabasz, on suppose que l'on dispose un ensemble de donnés $\mathcal{D}$ de taille n partitionné en k clusters. On note $\mathcal{C}(q)$ l'ensemble des points à l'intérieur du cluster q, C_q le centre du cluster q and n_q le nombre de points du cluster q.

On appelle W_k la matrice de dispersion intra-cluster et B_k la matrice de dispersion inter-cluster, définies par :

$$\begin{aligned} W_k &= \sum_{q=1}^{k} \sum_{x \in \mathcal{C}(q)} (x - C_q)(x - C_q)^T \\ B_k &= \sum_{q=1}^{k} n_q (C_q - C_{\mathcal{D}})(C_q - C_{\mathcal{D}})^T \end{aligned}$$

Finalement on définit le score comme :

$$S = \frac{\operatorname{tr}(B_k)}{\operatorname{tr}(W_k)} \frac{n_{\mathcal{D}} - k}{k - 1}$$

Le score est plus rapide à calculer que le silhouette score, mais il n'est pas borné. Un *bon* clustering aura un grand score. En effet, on veut avoir une petite distance intra-cluster mais une grande distance

inter-cluster. Donc un score faible veut dire que l'on a pas réussi à distinguer proprement les différents clusters.
Mais est-ce que ce score préfère-t-il les formes convexes comme le fait le silhouette score ?

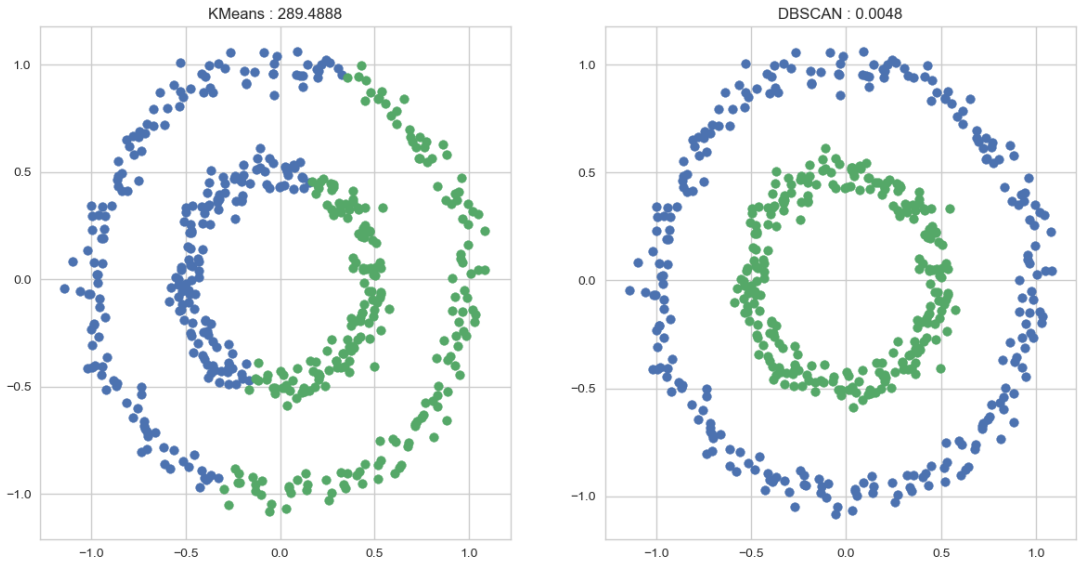


FIGURE 6.10 – Comparaison de l'index Calinski-Harabasz pour deux algorithmes de clustering

La différence est énorme. Pour le silhouette score, on a 0.3 pour le KMeans et 0.1 pour DBSCAN. À nouveau, le clustering reste faiblement noté bien qu'il y ait des clusters parfaits. Évidemment, nous ne serons pas toujours dans un cadre en deux ou trois dimensions où l'on peut valider visuellement un clustering.
Il existe bien sûr d'autres métriques pour mesurer la performance d'un clustering, mais nous n'avons pas trouvé de méthodes qui nous paraissaient universellement performantes. Mais est-on donc fatalement amené à ne jamais savoir si l'on a réussi un clustering ?

Pas vraiment. Cela nécessite d'avoir la bonne compréhension de la géométrie du problème qui dépend de chaque domaine/cas d'usage et également de la possibilité de traiter avec un expert humain. Plus que jamais, dans le cadre d'un clustering il est nécessaire de travailler en collaboration avec un expert métier qui saura aiguiller les recherches, analyser les clusters et apporter les modifications essentielles pour amener le clustering vers sa meilleure version.

Chapitre 7

Réduction de dimension

Calculer une matrice de distance entre chacun des points est un classique de la programmation : trouver le plus court chemin, trouver des observations proches pour une distance, clustering en général... Cette opération essentielle est extrêmement coûteuse en temps de calcul et en mémoire. Pour n observations, nous allons devoir calculer $n(n-1)$ distances, et ça sera encore pire si la distance est en très grande dimension. De plus, plus nous travaillons en grande dimension moins la distance a de *sens* : c'est le fléau de la dimension [1].
Il est donc souhaitable d'être capables de réduire la dimension de l'espace dans lequel on travaille, sans pour autant *perdre* trop d'informations : c'est l'objet de ce chapitre.

Dans le cadre supervisé, nous considérions un dataset composé d'un vecteur et la réponse attendue, mais dans le cadre non supervisé nous n'avons que des observations et aucune réponse. On peut donc modéliser les données que l'on a disposition comme une matrice (en reprenant les notations du cours) :

$$X = \begin{pmatrix} x_1^{(1)} & x_2^{(1)} & x_3^{(1)} & \cdots & x_d^{(1)} \\ x_1^{(2)} & x_2^{(2)} & x_3^{(2)} & \cdots & x_d^{(2)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x_1^{(n)} & x_2^{(n)} & x_3^{(n)} & \cdots & x_d^{(n)} \end{pmatrix}$$

7.1 Lemme de Johnson-Lindenstrauss

On souhaite observer les mêmes informations dans un espace de départ de grande dimension dans un espace de dimension inférieur. Une manière de le faire est de trouver une manière de plonger les observations dans un espace de plus petite dimension en conservant les distances entre les points.
Plus formellement, nous cherchons une fonction $f : \mathbb{R}^d \to \mathbb{R}^k$ avec $k << d$ telle que pour $\varepsilon > 0$ et $\forall (u, v) \in \mathcal{D}^2$, nous ayons la propriété :

$$(1-\varepsilon)\ \|u-v\|_2^2 \leqslant \|f(u)-f(v)\|_2^2 \leqslant (1+\varepsilon)\ \|u-v\|_2^2 \tag{7.1}$$

Distance dans l'espace de départ

Distorsion

Cela ressemble à la définition d'une fonction lipschitzienne mais sans coefficient unique de lipschitz. Il sera remplacé par à une distorsion d'ordre au plus $1+\varepsilon$. Cette similitude est expliquée par le titre de l'article *Extension of Lipschitz mapping into a Hilbert space* publié en 1984 par William Johnson et

1. Exploré dans l'annexe D

Joram Lindenstrauss [WBJ84]. Être capable de prouver qu'une telle fonction existe, et dire comment nous pouvons la construire est crucial. De plus, nous sentons que nous allons avoir une dépendance entre la dimension de l'espace d'arrivée k et la distorsion maximale que l'on s'autorise ε. Ce papier répond à toutes ces questions via le lemme suivant.

Lemme 1 (Johnson-Lindenstrauss). *Soit $\varepsilon > 0$. Si $k > \dfrac{24}{3\varepsilon^2 - 2\varepsilon^3} \ln(n)$, alors il existe une fonction $f : \mathbb{R}^d \to \mathbb{R}^k$ qui vérifie l'équation (7.1) pour tout $(u, v) \in \mathcal{D}^2$.*

Commençons par observer que la dimension de l'espace d'arrivé ne dépend pas de la dimension de l'espace de départ ! Pour le comprendre, si $n > d$ alors nous pouvons plutôt considérer l'espace engendré par les n observations plutôt que d. Nous sommes bien dans un des pires scénarios, mais la borne sur k tient toujours. Remarquons, de plus, que le résultat est vrai peu importe les caractéristiques du dataset $\mathcal{D}$, ce qui rend le lemme encore plus général.

Avec la table (7.1), nous pouvons voir les dimensions minimales de l'espace de réduction en fonction de n et ε.

	distorsion ε					
n	0.1	0.2	0.3	0.4	0.5	0.8
500	5326	1434	690	423	298	166
1000	5920	1594	767	470	331	185
1500	6268	1687	812	498	351	195
2000	6515	1754	844	518	364	203
2500	6706	1805	869	533	375	209
3000	6862	1847	889	545	384	214
3500	6994	1883	906	556	391	218
4000	7109	1914	921	565	398	222
4500	7210	1941	934	573	403	225
5000	7300	1965	946	580	408	228
5500	7382	1987	956	587	413	230
6000	7456	2007	966	593	417	233
6500	7525	2026	975	598	421	235
7000	7588	2043	983	603	424	237
7500	7647	2059	991	608	428	238
8000	7703	2073	998	612	431	240
8500	7755	2087	1005	616	434	242
9000	7804	2101	1011	620	437	243
9500	7850	2113	1017	624	439	245
10000	7894	2125	1023	627	442	246

TABLE 7.1 – Dimension minimale de l'espace de réduction pour plusieurs n et ε

Nous sentons la progression logarithmique en nombre d'observations : seulement 48% d'augmentation de l'espace de réduction pour ε entre $n = 500$ et $n = 10000$ qui représente une augmentation de 1900% en n.
On observe également le comportement prévisible de l'évolution en ε qui n'est pas linéaire. Etudions un peu plus la valeur de k avec un exercice.

Exercice 7.1 (Etude de k). *On considère la fonction* $f(n, \varepsilon) = \dfrac{24 \ln(n)}{3\varepsilon^2 - 2\varepsilon^3}$ *pour* $n \in \mathbb{N}^*$ *et* $\varepsilon \in]0, 1]$.

1. *Que signifie la valeur de* $\varepsilon = 1$ *pour notre problème ?*
2. *Montrer que la fonction est strictement décroissante en* ε *pour* n *fixé.*
3. *On souhaite réduire un dataset avec* n *observations et* d *colonnes d'un facteur* c. *Autrement dit, on souhaite obtenir une réduction dans un espace de dimension* $cd \in \mathbb{N}$. *Quelle condition doit vérifier* c *pour que cela soit possible ?*

Solution.

1. Cela veut dire qu'on relâche une des deux bornes de l'inégalité (7.1), mais qu'on contraint toujours *à la hausse* les variations de distances.
2. En dérivant simplement f par rapport à ε, on obtient le résultat souhaité. Ce qui est cohérent : plus on demande une distorsion faible, plus la dimension de l'espace de réduction sera élevée pour conserver au mieux les distances.
3. On souhaite donc ici que $f(n, \varepsilon) = cd$. Autrement dit :

$$\varepsilon^2(3 - 2\varepsilon) = \frac{24 \ln(n)}{cd}$$

On peut montrer que la fonction $g(x) = x^2(3 - 2x)$ est croissante pour $x \in [0, 1]$ et varie également de 0 à 1. Ainsi :

$$\frac{24 \ln(n)}{cd} < 1 \iff c > \frac{24 \ln(n)}{d}$$

D'où la condition sur le facteur de réduction c.

□

Remarquons que le lemme n'est pas toujours avantageux : pour $n = 500$ et $\varepsilon = 0.1$, on ne peut pas projeter le dataset dans un espace de dimension plus petit. Pour le faire il faut accepter une distorsion d'au moins 0.4.

Avec uniquement le lemme, nous savons qu'une telle fonction existe et quelle est la réduction de dimension que l'on peut obtenir. Mais nous ne savons pas comment construire une telle fonction. On obtient la réponse en démontrant le lemme puisque nous pouvons le faire via une démonstration par construction. Nous ne le ferons pas ici [2], mais il existe de très nombreuses manières de construire une telle fonction et qu'il s'agit d'un domaine de recherche en informatique. Ce résultat est largement utilisé pour la construction d'algorithmes qui exploitent de très larges bases de données ; essentiellement pour accélérer les temps de calcul des distances entre chacune des observations de la base.

Un point commun de chacune des preuves du lemme est que la fonction f est construite en faisant apparaître des projections **aléatoires**. Autrement dit, en projetant aléatoirement dans un espace de dimension k nous pouvons réduire la dimension de l'espace de départ : tout dépend de cet *aléatoire*.
Dans la preuve que nous avons faite, nous avons utilisé une matrice dense de variables aléatoires gaussienne : c'est coûteux à la fois en taille et en temps de génération. [Ach03] montre que l'on peut avoir le même résultat avec une matrice avec $\dfrac{2}{3}$ des entrées vides et uniquement remplie avec des valeurs $+1$ et -1. C'est bien moins volumineux pour le stockage et rapide à générer.
La meilleure borne a été proposée par [DMK14] et on peut montrer que la borne ne peut pas être améliorée. En pratique, il y a encore beaucoup de travail sur ce lemme puisque nous n'avons considéré que la distance euclidienne. Si nous travaillons avec la distance manhattan, alors ce résultat ne tient plus. De manière générale, pour les normes $\mathcal{L}_p$, le résultat n'est pas vrai. Même si la norme euclidienne est de loin la plus utilisée, la norme $\mathcal{L}_1$ a plus de sens en grande dimension comme nous l'avons vu avec la

2. Il suffit de m'envoyer un message pour obtenir la preuve rédigée pour une distribution gaussienne.

régression LASSO.

Pour notre problématique de Machine Learning, ce lemme est utile puisqu'il peut permettre de stocker plus efficacement des données vidéo par exemple, mais il y a plusieurs inconvénients :

- La dimension de l'espace de réduction reste élevée pour une visualisation en très petite dimension
- A ce jour, les projections sont faites aléatoirement, donc sans procédure *intelligente*
- Il n'y a pas de maîtrise du data-scientist autre que par le paramètre ε des projections

Ces trois points sont pourtant des *desideratas*. Il nous faut donc développer d'autres manières de réduire la dimension.

7.2 Analyse par composantes principales

Dans l'analyse par composantes principales il est question d'identifier les *directions* qui expliquent le mieux les variations des données. Si l'on suppose que les données se distribuent uniformément au sein d'une ellipse, alors nous identifions clairement les axes qui pourraient définir une ellipse.
L'objectif est donc d'identifier ces axes qui nous semblent évidents en petite dimension parce que nous sommes capables de les voir, mais cette fois en grande dimension. Ces axes pourrons définir un nouveau repère pour exprimer plus efficacement les informations que nous avons.

Pour le faire, voici les questions auxquelles nous devons répondre :

1. Comment trouver les directions qui *décrivent* le mieux les variations ?
2. Comment s'assurer que les directions formeront bien une nouvelle base ?
3. Comment prioriser les directions ?

On remarque que les questions ne sont pas toutes bien posées, et nous devons nous laisser guider pour les affiner. Les réponses se trouvent dans des notions fondamentales d'algèbre.

7.2.1 Diagonalisation d'une matrice

Soit la matrice A définie comme :

$$A = \frac{1}{2}\begin{pmatrix} 3 & 1 \\ 1 & 3 \end{pmatrix}$$

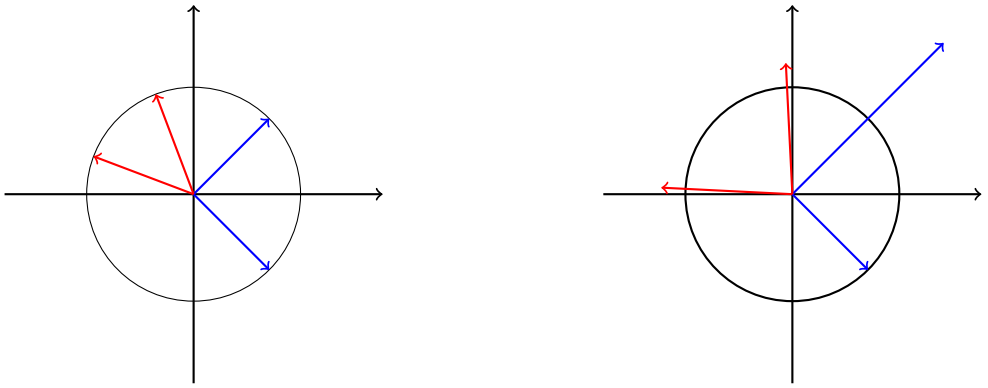
On sait que si l'on prend un vecteur $u \in \mathbb{R}^2$, on peut multiplier la matrice A par u : on dit en algèbre que A agit sur u. Visualisons ce que cela veut dire géométriquement avec la figure (7.1).
On remarque que les deux vecteurs rouges ont changé de direction et de taille, là où les vecteurs bleus ont conservé leurs directions (mais pas nécessairement leurs tailles). Il est étonnant d'avoir une invariance de direction pour seulement ces deux directions-là (on peut le vérifier par le calcul). Ces deux vecteurs bleus semblent être des vecteurs particuliers pour la matrice A. Généralisons : on considère une matrice carrée A de taille n dans l'ensemble de la section.

Définition 9 (Valeur propre et vecteur propre)**.** *Un vecteur $v \in \mathbb{R}^n$ est appelé un vecteur propre de A si et seulement si :*

$$\exists \lambda \in \mathbb{R},\ Av = \lambda v$$

*λ est une **valeur propre** de A associée au **vecteur propre** v.*

Dans notre exemple, le vecteur $(1, 1)$ est un vecteur propre de A, et sa valeur propre λ est 2. Pour le vecteur $(1, -1)$ sa valeur propre est 1, d'où l'invariance d'échelle. Remarquons que si v ne doit pas être



(a) Vecteurs initiaux (b) Vecteurs après application de la matrice

FIGURE 7.1 – Effet de la matrice sur deux vecteurs

nul pour être un vecteur propre, $\lambda = 0$ est cependant autorisé.

Exercice 7.2 (Non unicité d'un vecteur propre)**.** *Soit λ une valeur propre de A associée à un vecteur propre v. Montrer qu'il existe une infinité de vecteurs propres liés à la valeur propre λ.*

Solution. Par définition, on a que $Av = \lambda v$, donc si on multiplie par une constante $c \in \mathbb{R}^*$, l'équation devient $cAv = c\lambda v \iff A(cv) = \lambda(cv)$ d'où cv est un vecteur propre de A. □

Suite à cette remarque, on comprend que nous n'obtiendrons l'unicité pour un vecteur propre qu'en imposant des règles. Les plus classiques demandent à ce que le vecteur soit de norme 1, mais ça ne garantit par l'unicité encore. Comment trouver les valeurs propres d'une matrice ?

Théorème 3 (Caractérisation des valeurs propres)**.** *$\lambda \in \mathbb{R}$ est une valeur propre de A si et seulement si :*

$$\det(\lambda I_n - A) = 0$$

Exercice 7.3. *En exploitant la définition d'une valeur propre et d'un vecteur propre, démontrer le théorème (3).*

Solution. Soit $\lambda \in \mathbb{R}$ une valeur propre de A. Par définition, il existe $v \in \mathbb{R}^n$ un vecteur non nul tel que :

$$Av = \lambda v \iff \lambda v - Av = 0 \iff (\lambda I_n - A)v = 0$$

Or v est un vecteur non nul par définition, donc $\det(\lambda I_n - A) = 0$.

Inversement, si $\det(\lambda I_n - A) = 0$ alors il existe $v \in \mathbb{R}$ non nul tel que :

$$(\lambda I_n - A)v = 0 \iff Av = \lambda v$$

Donc λ est une valeur propre de A. □

On appelle l'équation définie dans le théorème (3) **l'équation caractéristique** de A. Le polynôme en λ issu de cette équation est de degré au plus n donc nous sommes certains d'avoir n racines, potentiellement complexes et potentiellement répétées. Ici, nous avons bien nos deux vecteurs bleus qui sont les seuls vecteurs propres de la matrice A de l'exemple.

Maintenant que nous sommes capables de trouver l'ensemble des valeurs propres d'une matrice, en résolvant l'équation $Av = \lambda v$ en v et en sélectionnant les vecteurs propres de norme 1, nous sommes également capables de calculer l'ensemble des vecteurs propres.
Un résultat [3] nous informe que cet ensemble de vecteurs propres forme une **base** : les vecteurs sont de norme 1 et orthogonaux entre eux. Ainsi, on définit :

- $\Phi = \begin{bmatrix} v_1 & v_2 & \cdots & v_n \end{bmatrix}$
- $\Lambda = \begin{pmatrix} \lambda_1 & 0 & 0 & 0 \\ 0 & \lambda_2 & 0 & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & \lambda_n \end{pmatrix}$

Et on peut donc écrire l'équation :

$$A\Phi = \Phi\Lambda \iff A = \Phi\Lambda\Phi^{-1}$$

On vient de **diagonaliser** la matrice A à l'aide de ses valeurs propres et vecteurs propres.

Exercice 7.4 (Puissance d'une matrice)**.** *Soit A une matrice diagonalisable de taille d. Calculer A^n pour $n \in \mathbb{N}$.*

Solution. Puisque A est diagonalisable, il existe Φ et Λ défini comme précédemment telle que $A = \Phi\Lambda\Phi^{-1}$. Par récurrence rapide, on peut montrer que :

$$A^n = \Phi\Lambda^n\Phi^{-1}$$

Avec :

$$\Lambda^n = \begin{pmatrix} \lambda_1^n & 0 & 0 & 0 \\ 0 & \lambda_2^n & 0 & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & \lambda_d^n \end{pmatrix}$$

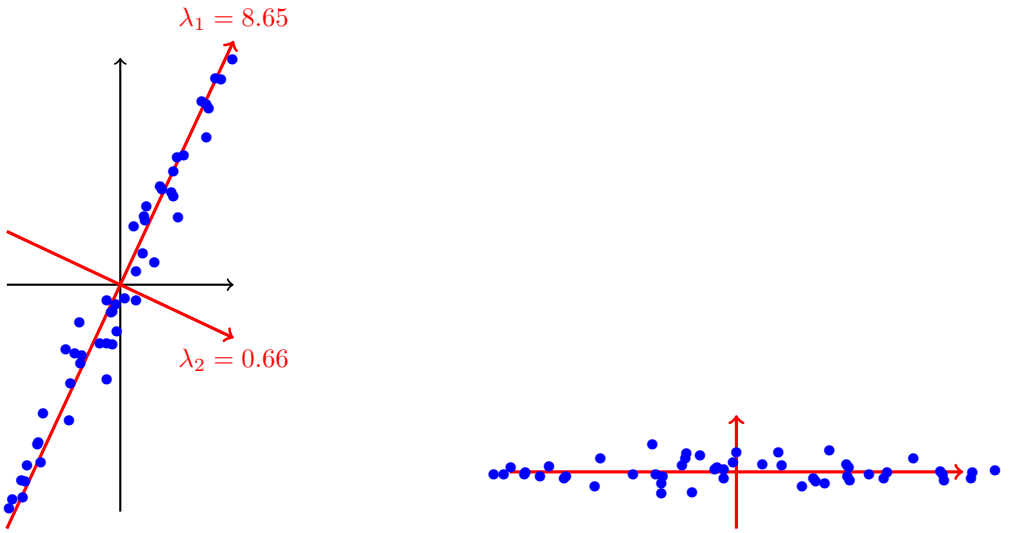
□

C'est un résultat qui permet d'accélérer les calculs de puissance de matrices puisque nous n'avons besoin de calculer que deux multiplications de matrice, au lieu de n. Mais on pourrait se dire que calculer l'inverse de la matrice Φ est coûteux en temps. En pratique, comme nous avons une base, $\Phi^{-1} = \Phi^t$.

Voyons sur un exemple comment l'analyse par composantes principales nous permet de mieux visualiser une matrice. Attention : pour obtenir cette figure, nous n'avons pas encore tout les bagages théoriques nécessaires, ce sera fait juste après.

Dans la figure (7.2), on présente la projection de l'espace de départ (engendré par X) et l'espace défini par les vecteurs propres d'une certaine matrice. C'est exactement ce que l'on voulait : le grand axe de variation a été compris, et les deux vecteurs propres forment une base orthonormée pour projeter X. Ainsi, on peut se suffire de la composante liée à λ_1 pour bien approcher des variations de la matrice : on a bien réduit la dimension.

3. Que nous ne démontrerons pas



(a) Dans l'espace de départ

(b) Dans l'espace engendré par les vecteurs propres

FIGURE 7.2 – Projection d'une matrice X dans l'espace engendré par ses vecteurs propres

7.2.2 Application à la réduction de dimensions

C'est cette dernière remarque qui motive notre utilisation de ces résultats d'algèbre. Nous pouvons projeter X dans un sous-ensemble de ses vecteurs propres et conserver *a priori* le plus de variations expliquées. Se posent alors deux questions :

1. Est-on certain que l'on pourra toujours diagonaliser X ?
2. Comment prioriser les composantes à conserver ?

La première question est répondue par la négative : il n'y a aucune garantie mathématique qu'une matrice quelconques puisse être diagonalisable. X n'est même pas une matrice carrée en règle générale ! Puisqu'on cherche à identifier les axes où la variance est la plus forte, décomposons la matrice de variance-covariance de X ! On la note Σ et dans notre cas [4], elle se définit comme :

$$\Sigma = \frac{XX^t}{n-1}$$

Chaque coefficient $(\sigma_{ij})_{1 \leqslant i,j \leqslant d}$ de Σ représente la covariance entre l'information i et l'information j. Ainsi, quand $i = j$, cela revient à avoir la variance de l'information i, d'où le nom de matrice de variance-covariance. Cela fait déjà plus de sens philosophiquement, et on a cette fois la garantie de toujours pouvoir la diagonaliser puisque Σ est une matrice symétrique semi-définie positive.

Il reste à savoir comment prioriser les composantes à conserver. On remarque dans la figure (7.2) que les valeurs propres associées à chacun des axes sont très différentes et on dirait que la valeur est corrélée à *l'importance* de la direction.
Voyons avec un exercice comment nous pouvons tirer parti de cette information.

4. Avec une matrice X centrée.

Exercice 7.5 (Trace de matrice semblable et variance expliquée)**.** *Soit $A, B \in \mathcal{M}_{n,n}$. On dit que A et B sont deux matrices semblables s'il existe $P \in \mathcal{M}_{n,n}$ telle que :*

$$A = PBP^{-1}$$

On note de manière classique $A = (a_{ij})_{1 \leqslant i,j \leqslant n}$ et on définit l'opérateur trace :

$$\mathrm{Tr}\,(A) = \sum_{i=1}^{n} a_{ii}$$

L'opérateur trace possède la propriété suivante :

$$\mathrm{Tr}\,(AB) = \mathrm{Tr}\,(BA)$$

1. *Traduire avec des mots ce que représente l'opérateur trace.*
2. *Montrer que les traces de deux matrices semblables sont égales.*
3. *Si l'on diagonalise une matrice de variance covariance, à quoi correspond la somme des valeurs propres ?*

Solution. 1. L'opérateur trace d'une matrice renvoie la somme de ses éléments diagonaux.

2. En reprenant les notations de l'exercice, on a :

$$\begin{aligned} \mathrm{Tr}\,(A) &= \mathrm{Tr}\left(PBP^{-1}\right) \\ &= \mathrm{Tr}\left(BP^{-1}P\right) \\ \mathrm{Tr}\,(A) &= \mathrm{Tr}\,(B) \end{aligned}$$

3. Si on diagonalise la matrice de variance-covariance Σ, alors :

$$\exists \Phi, \Lambda \in \mathcal{M}_{d,d},\ \Sigma = \Phi \Lambda \Phi^{-1}$$

Ce sont des matrices semblables donc :

$$\mathrm{Tr}\,(\Sigma) = \mathrm{Tr}\,(\Lambda) \Longleftrightarrow \sum_{i=1} \sigma_{ii} = \sum_{i=1}^{d} \lambda_i$$

Autrement dit, la somme des variances de chacune des informations contenues dans la matrice X est égale à la somme des valeurs propre de la matrice de variance-covariance.

□

Avec cet exercice on comprend comment nous allons prioriser les axes à conserver : plus une valeur propre est grande, plus sa composante associée est importante. De plus, nous sommes capables de définir la proportion de variance expliquée par chacune des composantes.
Par exemple, dans la figure (7.2), la première composante explique 93% de la variance. D'où notre intuition de dire qu'il suffisait de conserver uniquement la première composante pour mieux visualiser les données.

Par rapport au lemme de Johnson-Lindenstrauss, cette fois nous sommes capables d'expliquer chacun des axes créés avec une procédure intelligente. En revanche, nous faisons l'hypothèse que les variations peuvent être expliquées par une combinaison linéaire des informations présentes, ce qui n'est pas forcément le cas.
Finalement, l'analyse par composantes principales cherche à résumer la *forme* de la matrice avec une vision macro via une explication de la variance. Que faire si nous souhaitons réduire la dimension en conservant une vision plus micro cette fois ?

7.3 UMAP

On souhaiterait être capable de choisir la dimension de réduction, et projeter de manière intelligente les données dans cet espace en mettant l'accent sur des structures *locales* des données. L'algorithme UMAP [MHM18] répond à ce souhait. La théorie des catégories et des notions de la géométrie différentielles sont exploitées pour donner un cadre mathématique de très grande qualité à cet algorithme. Nous ne recommandons donc pas de lire en détail la partie théorique de l'article, et nous n'allons que donner les intuitions et problèmes qui sont discutés dans cette partie-là.

L'idée de l'algorithme est de fonctionner en deux temps :

1. Apprendre la structure des données (locale) dans l'espace de départ
2. Projeter la structure des données dans l'espace d'arrivé

Avant de traiter en détail ces deux parties, laissons-nous guider par les hyper-paramètres de cet algorithme :

- `k` : nombre de voisins à considérer dans l'espace de départ pour apprendre la structure des données
- `d` : la dimension de l'espace de réduction
- `min_dist` : la séparation souhaitée entre deux points proches dans l'espace de réduction
- `n_epochs` : le nombre d'itérations d'optimisation pour la projection dans l'espace de réduction

Avec cela, nous comprenons que choisissons vraiment la dimension de l'espace d'arrivée ! C'est un point fort par rapport à Johnson-Lindenstrauss où nous avons une borne et des conditions supplémentaires, et par rapport à l'analyse par composante principale où l'on sélectionne les composantes en essayant d'avoir une explication de la variance suffisamment forte.
Egalement, il semblerait que le placement des points dans l'espace de réduction soit itératif, donc il y aurait un problème d'optimisation à résoudre. Il nous reste à comprendre le fonctionnement des deux étapes de l'algorithme pour comprendre l'utilité des deux hyper-paramètres restants.

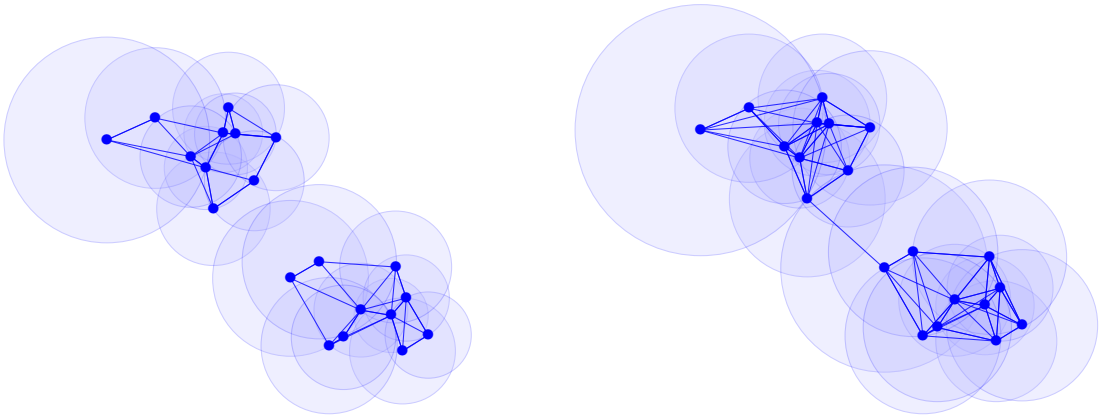
7.3.1 Formation du graphe en grande dimension

Pour un point donné, on cherche à trouver son voisinage pour estimer sa densité. Cependant, puisque nous sommes potentiellement en grande dimension, nous aurons des problèmes liés au fléau de la dimension [5]. Pour s'en sortir on décide de munir chaque point d'un espace avec sa propre métrique. On s'assure de revenir dans un espace de dimension bien plus petit et également d'avoir une notion de distance cohérente [6]. Ainsi, nous pouvons prendre des boules de rayon 1 pour chaque point avec sa propre distance et en réalité s'assurer que chaque point possède bien k voisins à l'intérieur de sa boule.

En faisant cela, on résout des problèmes mais on en crée un de taille : nous n'avons plus de cohérence globale. Ainsi, la partie la plus mathématique de cet article adresse cette problématique et définit un nouveau type d'objet qui permet de définir une structure cohérente à plus haut niveau, en contrepartie de la perte de certitude sur l'appartenance ou non de point à un voisinage donné. C'est le choix de travailler avec un nombre de voisins variable qui nous permet de simplifier l'écriture de cette partie plutôt que d'avoir des rayons variables.

Toutes ces intuitions mathématiques se réalisent concrètement sous la forme d'un graphe $\overline{G}$ dirigé où chaque arête possède un poids. Un noeud du graphe correspond ici à un point, et une arête à un chemin entre deux points. Il nous reste à trouver comment définir le poids de chaque arête pour être capable de définir complètement le graphe orienté. Plus formellement, pour chaque point x_i, on commence par

5. Voir l'annexe D.
6. Puisqu'on peut normaliser la distance par rapport au voisin le plus éloigné.



(a) Pour $k = 3$ voisins

(b) Pour $k = 5$ voisins

FIGURE 7.3 – Graphes appris pour deux valeurs de k, le nombre de voisins considérés pour construire l'espace métrique

trouver ses k voisins les plus proches selon la distance d que l'on aura sélectionné. On note cet ensemble $\mathcal{N}(x_i) = \{x_i^{(1)}, \dots x_i^{(k)}\}$. On peut donc définir :

$$\rho_i = \min \left\{ d\left(x_i, x_i^{(j)}\right) | 1 \leqslant j \leqslant k,\ d\left(x_i, x_i^{(j)}\right) > 0 \right\}$$

Cette définition de ρ_i dérive de contraintes théorique : il s'agit de s'assurer que x_i sera connecté à au moins un autre point du dataset que l'on considère avec un poids d'arête qui vaut 1.
Puis on définit un coefficient de normalisation σ_i qui est solution de l'équation :

$$\sum_{j=1}^{k} \exp\left(\frac{-\max\left\{0, d\left(x_i, x_i^{(j)}\right)\right) - \rho_i\right\}}{\sigma_i} \right) = \frac{\ln(k)}{\ln(2)}$$

Ce coefficient permet de normaliser les distances locale pour chaque point x_i. Tout cela nous permet de définir le poids d'une arête comme :

$$w\left(\left(x_i, x_i^{(j)}\right)\right) = \exp\left(\frac{-\max\left\{0, d\left(x_i, x_i^{(j)}\right) - \rho_i\right\}}{\sigma_i} \right)$$

Pour comprendre chacune de ces notions, résolvons l'exercice suivant.

Exercice 7.6 (Valeur de w)**.** *On reprend l'ensemble des notations jusqu'à présent.*

1. *Que cela signifie-t-il quand* $\max\left\{0, d\left(x_i, x_i^{(j)}\right) - \rho_i\right\} > 0$ *?*
2. *Quelle est la plus grande valeur que peut prendre* $w\left(\left(x_i, x_i^{(j)}\right)\right)$ *?*
3. *Est-ce que w est symétrique ?*

Solution.

1. Si $\max\left\{0, d\left(x_i, x_i^{(j)}\right) - \rho_i\right\} > 0$, alors c'est que $d\left(x_i, x_i^{(j)}\right) > \rho_i$. Par définition de ρ_i cela veut dire que j n'est pas le voisin le plus proche, et on peut interpréter la quantité $d\left(x_i, x_i^{(j)}\right) - \rho_i$ comme la distance qui empêche le voisin j d'être le voisin le plus proche du vecteur i.
2. La valeur la plus grande qui peut être atteinte est 1, et c'est le cas quand le vecteur j est le voisin le plus proche du vecteur i.
3. Non, la valeur de ρ_i et σ_i sont dépendantes de l'espace métrique du point i que l'on considère. Ainsi, pour deux vecteur i_1 et i_2, il faut d'abord que tous les deux soient voisins l'un de l'autre (ce qui n'est pas garanti). Rien que cette condition nous permet d'affirmer que la relation n'est pas symétrique.

□

Nous avons maintenant réussi à construire un graphe orienté à partir des données d'entrée, et l'on note A la matrice adjacente au graphe $\overline{G}$. On peut interpréter chaque coefficient A_{ij} de A comme la probabilité que l'arête de x_i vers x_j existe. Cette asymétrie vient des espaces métriques différents entre les points et également de la valeur de σ_i pour chaque point, comme vu dans l'exercice précédent.
Si l'on considère maintenant la matrice B définie par :

$$B = A + A^t - A \circ A^t$$

Avec $\circ$ la notation du produit Hadamard défini par $(U \circ V)_{ij} = U_{ij}V_{ij}$: il s'agit du produit terme à terme de deux matrices de même taille. Cette matrice B peut sembler sortir du chapeau, mais elle obéit à une norme classique en logique floue. Ce niveau de mathématique dépasse à nouveau largement ce cours. Voyons en quoi cette matrice est utile.

Exercice 7.7 (Symétrie)**.** *Soient $U, V \in \mathcal{M}_{n,m}$. On rappelle que :*

$$\begin{aligned}(U+V)^t &= U^t + V^t \\ (U \circ V)^t &= U^t \circ V^t\end{aligned}$$

Montrer que la matrice B est symétrique.

Solution. On a :

$$\begin{aligned}B^t &= (A + A^t - A \circ A^t)^t \\ &= A^t + A - A^t \circ A \\ &= A + A^t - A \circ A^t\end{aligned}$$

Ainsi, $B = B^t$ donc B est symétrique. □

Alors on peut interpréter chaque coefficient B_{ij} comme la probabilité que au moins une des deux arêtes existe (x_i vers x_j ou x_j vers x_i). En effet, pour rassembler l'ensemble des métriques en un espace global, en logique floue nous travaillons avec l'union de chacune de ces métriques, d'où l'interprétation. Finalement on définit le graphe G avec la matrice adjacente donnée par B. Nous avons donc construit un graphe qu'il nous reste maintenant à réduire dans l'espace de plus petite dimension d.

7.3.2 Réduction du graphe

UMAP utilise un algorithme *force-based layout* ou algorithme de dessin basé sur des forces en français. Ce fonctionnement exploite des forces attractives et répulsives sur les arcs et les noeuds. L'idée est de resserrer les liens et d'éloigner les observations.

Nous ne détaillerons pas plus le fonctionnement de cette réduction car elle est essentiellement technique et on peut se suffire de l'intuition. Il est à noter cependant qu'ici le problème que l'on cherche à résoudre

n'est pas convexe. Ainsi, plusieurs essais pour une même base de données peuvent conduire à des résultats différents.
Focalisons-nous maintenant sur un aspect plus pratique de UMAP.

7.3.3 Utilisation en pratique d'UMAP

On rappelle que l'on a 4 principaux hyper-paramètres :

- n : nombre de voisins à considérer dans l'espace de départ pour apprendre la structure des données
- d : la dimension de l'ensemble de réduction
- min_dist : la séparation souhaitée entre deux points proches dans l'espace de réduction
- n_epochs : le nombre d'itérations d'optimisation pour la projection dans l'espace de réduction

A cette liste, non exhaustive, des paramètres de UMAP nous ajoutons le paramètre set_op_mix_ratio. C'est un nombre entre 0 et 1 qui donne l'interpolation entre une intersection et une union dans la logique floue lors de la construction de l'espace métrique pour un vecteur donné. Concrètement, cela signifie que l'on peut jouer sur l'importance ou non que chaque point **possède** un voisinage.

Exercice 7.8 (Mise en situation)**.** *Nous travaillons avec un data-scientist qui ne connait pas bien UMAP et sollicite vos conseils.*

1. *J'ai l'impression que je ne capture pas assez bien les structures très locales de mon dataset. Que faire ?*
2. *J'aimerais être capable d'exclure des valeurs atypiques pour mieux les analyser ensuite. Y a-t-il un moyen de le faire avec UMAP ?*
3. *Je souhaiterais prédire le prix d'une voiture avec UMAP. Comment procéder ?*
4. *J'aimerai avoir une approche globale comme l'analyse par composante principale, mais je préfère utiliser UMAP. Comment faire ?*
5. *Quand je réalise mon graphique, j'ai besoin que les groupes soit bien distincts. Comment m'en assurer ?*
6. *Bien que j'ai suivi vos conseils, les groupes ne sont toujours pas bien distinct. Y a-t-il une autre piste d'amélioration ?*

Solution.

1. C'est que probablement le nombre de voisins considérés n est trop grand. Une piste d'amélioration est de réduire ce nombre-là.
2. Une possibilité est de chercher à avoir un nombre de voisins pas trop faible mais d'exploiter le paramètre set_op_mix_ratio pour isoler l'observation dans l'espace réduit.
3. Ce n'est pas possible : UMAP n'est pas un algorithme de prédiction, c'est un algorithme de réduction de dimensions non-supervisé. En réalité, il est capable de faire une réduction de dimension *supervisé* en prenant en compte une cible, mais ça ne permet pas de faire de la prédiction.
4. Une manière de le faire est d'augmenter le nombre de voisins pour moins donner de poids aux structures locale. Cela peut en revanche ralentir les calculs.
5. On peut essayer avec des valeurs de min_dist plutôt faible, dans l'idée de rapprocher ce qui se ressemble le plus.
6. Si cela ce suffit pas, alors on peut aussi permettre à l'algorithme d'optimisation de l'affichage en petite dimension d'avoir plus d'itérations en augmentant la valeur du nombre d'époque de descente de gradient.

□

Chapitre 8

Modèles de langage

Dans son ouvrage *L'obsolescence de l'homme*, Gunther Anders explore le concept de *honte prométhéenne* : cette sensation de décalage entre les capacités humaines et les prouesses technologiques. Il décrit ce sentiment comme une forme de honte ressentie par les humains face à l'efficacité et la perfection des machines qu'ils ont eux-mêmes créées. Cette honte prométhéenne naît de la prise de conscience que les machines peuvent accomplir des tâches avec une précision et une rapidité que les humains ne peuvent égaler.

Un exemple concret de cette honte prométhéenne peut être observé dans le domaine de l'intelligence artificielle (IA). Les modèles de langage, tels que ceux utilisés dans les assistants virtuels ou les systèmes de traduction automatique, sont capables de traiter et de générer du texte avec une efficacité et une rapidité impressionnantes. Face à ces performances, les humains peuvent ressentir une forme de honte ou d'inadéquation, se demandant comment ils peuvent rivaliser avec des machines qui semblent comprendre et manipuler le langage mieux qu'eux.

Cependant, cette sensation de décalage peut être atténuée par une meilleure compréhension de la technologie. L'objectif de ce cours est de démystifier les modèles de langage en expliquant leur fonctionnement. En comprenant comment ces modèles sont conçus, entraînés et utilisés, nous pouvons non seulement apprécier leur puissance et limite, mais aussi mieux les intégrer dans notre quotidien. En réduisant le mystère qui entoure ces technologies, nous pouvons également réduire le sentiment de honte prométhéenne et mieux coexister avec les outils que nous avons créés.

Précisons que les modèles utilisés pour obtenir les modèles de langage dont nous allons parler sont des réseaux de neurones avec une architecture transformers. Les réseaux de neurones ne sont pas au programme du cours parce qu'ils nécessiterait un cours dédié, idem pour l'architecture transformers. Nous traiterons dans cette séance uniquement de notions abordables avec nos connaissances, mais qui informe tout autant sur le fonctionnement d'un modèle de langages et de l'écosystème.

8.1 Collecter et transformer du texte

Avec l'ouverture de ChatGPT au public fin novembre 2022, OpenAI s'est placé comme leader dans le domaine des LLM. Depuis, OpenAI noue des partenariats. D'abord avec The Associated Press et Axel Springer, puis en mars 2024 avec le journal français Le Monde et le groupe espagnol Prisa Media. L'objectif affiché est de changer la relation que l'on a avec les informations, et renforcer la *confiance* dans ChatGPT avec des liens vers les articles des journaux concernés. Les partenaires d'OpenAI gagnent en visibilité et financièrement. En retour, OpenAI accède de manière exclusive aux contenus des journaux : cela formera des datasets de très grande qualité. Et si la bataille n'était plus seulement théorique et technologique, mais également dans la donnée ?

8.1.1 Comment constituer un corpus ?

La première source de texte pour composer un corpus d'apprentissage vient d'internet. CommonCrawl est une organisation à but non-lucratif qui crée des *images* d'internet à chaque fois qu'un *crawl* est lancé. Parmi l'ensemble des données récoltées, on y trouve des pages web, du texte et du code par exemple. Depuis 2008 la base de données est augmentée presque tous les mois. Ce dataset est une pièce majeure dans la conception du dataset d'entraînement des LLM, mais il ne peut pas être exploité tel quel : aucun traitement de la donnée n'est réalisé.

Dans l'optique de nettoyer les *dumps* de CommonCrawl est introduit par Google le dataset C4 [RSR+19], pour *Colossal Cleaned Common Crawl Corpus*, pour entraîner le modèle T5. Plus tard les datasets RefinedWeb [PMH+23] et FineWeb [PKvWW24] sont construits de la même manière pour respectivement entraîner Falcon et LLaMa 3. Décrivons un peu plus en détail ces les différents filtre appliqués sur les pages internet :

- **URL** : suppression de pages selon une liste d'URL (∼4.6M de sites) et selon la présence de certains mots dans les URL
- **Langue** : ne sont conservées que les pages dont la langue identifiée est l'anglais (le modèle fastText [JGBM16] est utilisé avec un score de 0.65)
- **Qualité et répétition** : ne sont conservées que les pages qui vérifient les conditions proposées pour construire MassiveText qui a entraîné Gopher [RBC+21] :
  - La page contient entre 50 et 100 000 mots, et la longueur moyenne des mots est comprise entre 3 et 10
  - La page a un ratio entre symboles et mots inférieur à 0.1 pour les symboles dièse ou points de suspension
  - La page a moins de 90% de ses lignes qui commencent par des puces et qui ont moins de 30% des lignes qui se terminent par des points de suspension
  - 80% des mots doivent contenir au moins une lettre
  - Au moins deux mots parmi la liste : *the, be, to, of, and, that ; have, with* doivent être présents dans le document
  - La page ne dépasse aucun des seuils de répétitions décrits dans l'article [RBC+21] section A.1.1. En un mot, plusieurs approches à plusieurs niveaux sont utilisées pour identifier les n-grams qui se répètent dans une phrase, paragraphe...

Notons que les conditions imposées dans MassiveText sont des heuristiques qui fonctionnent pour la langue anglaise : il faut probablement adapter cela pour chaque langue. Toutes ces étapes combinées suppriment presque 70% des données présentes dans CommonCrawl initialement, ce qui correspond à peu près à 36T tokens. La prochaine étape est la déduplication des données.

Rappelons que les LLM ne sont entraînés, en général, que sur une seule époque. Les réseaux de neurones dans d'autres domaines, comme la vision, peuvent être entraînés sur les mêmes images plusieurs centaines de fois. C'est donc surprenant de vouloir supprimer les doublons des pages internet. [LIN+21] identifie quatre avantages d'entraîner un LLM sur un dataset dédoublonné :

1. Réduction du risque de mémorisation de certaines séquences présentes trop souvent (démontré dans [CIJ+22])
2. Réduction de l'overlap entre train et test. L'article exhibe une séquence de 61 mots qui est répétée 61 036 fois dans C4 [1]
3. Gain en rapidité d'entraînement, donc évitement de coût d'entraînement

1. "by combining fantastic ideas, interesting arrangements, and following the current trends in the field of that make you more inspired and give artistic touches. We'd be honored if you can apply some or all of these designs in your wedding. Believe me, brilliant ideas would be perfect if they can be applied in real life and make the people around you amazed !"

4. Gain en performance jusqu'à 10% grâce à du texte de meilleure qualité

Mais comment le faire ? De deux manières : on cherche des doublons exacts ou des doublons semblables. Pour la première méthode, c'est assez simple à mettre en oeuvre, c'est moins clair pour la seconde.

Considérons deux ensembles A et B. On définit l'indice de Jaccard comme :

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} \qquad \text{(Indice de Jaccard)}$$

Plus la valeur est proche de 1, plus A et B sont proche. Cependant, calculer ce coefficient avec l'ensemble de nos données est beaucoup trop coûteux. [Bro97] introduit la méthode MinHash qui permet d'approcher cet indice à l'aide de fonction de hachage. En appliquant la fonction de hachage à A et B, la probabilité que la valeur minimale du hash de A et la valeur minimale du hash de B soit égale est exactement $J(A,B)$! On a un estimateur non biaisé, construit à partir d'une fonction de hachage arbitraire. Cependant, cet estimateur a une variance très forte, c'est pourquoi il faut répéter cela plusieurs fois. Prenons par exemple [2] la table suivante :

Texte 1	Texte 2	Texte 3	Texte 4
1	0	1	0
1	0	0	1
0	1	0	1
0	1	0	1
0	1	0	1
1	0	1	0
1	0	1	0

TABLE 8.1 – Vecteurs représentant des textes

On considère les permutations :

$$P1 = [1,3,7,6,2,5,4], \quad P2 = [4,2,1,3,6,7,5], \quad P3 = [1,4,7,6,1,2,5]$$

On obtient alors :

T1	T2	T3	T4
1	0	1	0
0	1	0	1
1	0	1	0
1	0	1	0
1	0	0	1
0	1	0	1
0	1	0	1

T1	T2	T3	T4
0	1	0	1
1	0	0	1
1	0	1	0
0	1	0	1
1	0	1	0
1	0	1	0
0	1	0	1

T1	T2	T3	T4
1	0	1	0
0	1	0	1
1	0	1	0
1	0	1	0
1	0	1	0
1	0	0	1
0	1	0	1

On calcule la signature en prenant le premier indice où la valeur est 1 :

S1	S2	S3	S4
1	2	1	2
2	1	4	1
2	1	2	1

Avec cette table, on estime par exemple la similarité de T1 et T3 à 0.66 alors que la véritable similarité est 0.75 : c'est une bonne approximation. Plus concrètement, FineWeb collecte l'ensemble des 5-grams de

2. Largement inspiré du schéma de Park ChangUk.

chaque document et calcule 112 fonctions de hash réparties dans 14 blocs de 8 hashs chacun. Ainsi, si deux documents ont pour similarité s, la probabilité qu'il soit effectivement identifié comme similaire est :

$$\mathbb{P}\left(\text{documents similaire identifié}\right) = (1 - (1 - s^8)^{14})$$

Si on reprend notre cas, on a $s = 0.75$ d'où la probabilité que la similarité soit identifiée est 0.77 ! RefineWeb utilise 9000 fonctions de hachage divisées en 450 blocs de 20 hachages chacun. Cela nécessite évidemment plus de puissance de calcul, mais cela permet aussi une meilleure approximation. En pratique, les deux courbes théoriques se croisent aux alentours du seuil 0.75 : c'est celui qui est choisi pour décider que deux documents sont des doublons. Donc dans notre exemple, nous venons d'identifier un doublons en T1 et T3.

Nous savons maintenant identifier les doublons, mais doit-on le faire pour chaque dump individuellement ou collectivement ? D'après ce que nous avons expliqué plus tôt, moins on a de doublons, plus la performance est forte. FineWeb montre que c'est l'inverse qui se produit : ce changement permet d'avoir des performances similaires à RefinedWeb pour l'instant. Pour avoir de encore meilleures performances, FineWeb modifie et ajoute d'autres filtres de qualité comme, par exemple, supprimer des documents où la fraction des lignes terminant par une ponctuation est inférieure à 0.12% ou des documents où la fraction des lignes de moins de 30 caractères est supérieure à 0.67%.

Tout ce travail (dont nous n'avons pas présenté l'exhaustivité) permet d'obtenir des datasets de plus en plus conséquents, mais de qualité. FineWeb est composé de 15T de tokens rendus publics, alors que RefineWeb n'est pas rendu complètement public. Un autre dataset de l'ordre de grandeur de FineWeb, RedPajama v2 [Com23] composé de 30.4T de tokens (le double !) donne de moins bonnes performances pour les LLM parce que la qualité des filtres est moins bonne. La série de modèle Qwen 2.5[Tea24] suit la même direction : plus de tokens de plus haute qualité. Un autre exemple de l'importance de la qualité du dataset est l'entraînement de phi-1 de Microsoft [GZA+23] destiné à écrire du code de qualité. Il a été entraîné avec très peu de tokens (voir table 8.2) parce que son corpus est composé d'exemples de très bonne qualité de code. [3] On notera que la version phi-3 est elle entraîné sur beaucoup plus de tokens que la version précédente, mais encore de taille moindre par rapport à ses contemporains.

Modèle	Année	Tokens d'entraînement
Chinchilla	Mars 2022	1.3T
PaLM	Avril 2022	768B
Phi-1	Novembre 2022	6B
LLaMa	Février 2023	1.4T
PaLM 2	Mai 2023	3.6T
Falcon	Juin 2023	5T
LLaMa 2	Juillet 2023	2T
Gemma	Février 2024	6T
Phi-3	Avril 2024	3.3T
LLaMa 3	Avril 2024	15T
Qwen 2.5	Septembre 2024	18T

TABLE 8.2 – Nombre de token d'entraînement pour quelques LLM

Nous avons montré combien le travail de preprocessing pour les données issue de CommonCrawl est important pour obtenir LLM performant. Mais il existe en pratique de nombreux datasets au-delà des données issues du Common Crawl. Tous sont construit avec une attention à la qualité et peuvent être également filtré pour obtenir une qualité encore supérieure. Avec la table (8.3) on note que la part dans les modèles fondamentaux les plus performant la part d'internet reste forte.

Il est donc possible de construire son propre corpus en utilisant des datasets spécialisés tels que GitHub ou BigQuery pour le code, ProofPile-2 pour les écrits mathématiques ou PubMedCentral pour le domaine

3. On invite le lecteur à lire ce fabuleux article, et les suivants.

Modèle	Page Web	Code	Encyclopédique	Livres	Académique	Réseaux sociaux	Langue
LLaMa	82	4.5	4.5	4.5	2.5	2.0	0
GPT-3	60	22	3	15	0	0	0
PaLM	27	50	4	13	0	5	1
Gopher	58	3	2	27	0	0	10
Chinchilla	55	4	1	30	0	0	10
Falcon	100	0	0	0	0	0	0
phi-1	0	100	0	0	0	0	0
LaMDA	12.5	50	12.5	0	0	12.5	12.5

TABLE 8.3 – Composition du corpus d'entraînement (en %). Source : [LCL+24]

médical. Ces datasets peuvent également être utilisés pour spécialiser un modèle fondamental.

Cependant, l'ensemble des datasets précédents concerne presque exclusivement du texte en anglais. Par conséquent, entraîner un modèle de langage performant dans plusieurs langues peut s'avérer difficile en raison de la prédominance du contenu anglais. Pour y remédier, il existe le dataset mC4 qui est une extension du dataset C4 dans 101 langues, issue de la version multilingue de Common Crawl. Les mêmes types de conditions de filtrage ont été réalisés par l'équipe de Google pour obtenir un corpus propre et cohérent. La deuxième version de Fineweb contient elle aussi un fort travail dans le même sens mais spécifique à chaque langue [4].

Langue	Tokens dans mC4		Locuteurs natifs	
	Nombre (en milliards)	Proportion (%)	Nombre (en millions)	Proportion (%)
Anglais	2000	45.5	379	5.1
Russe	250	5.7	154	2.1
Allemand	200	4.5	95	1.3
Français	170	3.9	76.8	1.0
Espagnol	160	3.6	460	6.2
Chinois simplifié	150	3.4	1300	17.6
Portugais	120	2.7	234	3.2
Italien	100	2.3	59.8	0.8
Polonais	90	2.0	46.6	0.6
Japonais	80	1.8	128	1.7
Néerlandais	70	1.6	23	0.3
Turc	60	1.4	71.7	0.9
Coréen	50	1.1	77.2	1.0
Arabe	50	1.1	315	4.2
Vietnamien	40	0.9	77.2	1.0
Persan	40	0.9	65	0.9
Indonésien	40	0.9	43.3	0.6
Hindi	30	0.7	615	8.3
Tchèque	30	0.7	10.7	0.1
Ukrainien	30	0.7	34.9	0.5

TABLE 8.4 – 20 langues les plus représentées dans mC4, sources Ethnologue et [RSR+19]

Avec le tableau (8.4), on note que les vingt langues les plus représentées dans mC4 correspondent à plus de 85% des tokens du dataset complet. Si l'on compare à la proportion de locuteurs natifs, alors on remarque de fortes disparités dans la représentativité.

4. Le dataset a été publié début décembre, et à la date où nous écrivons ces lignes le détails des travaux n'a pas encore été rendu publique

La majorité des LLM disponibles sont construits par des entreprises américaines et le nombre de personnes pouvant converser en anglais est largement supérieur à 5% de la population mondiale. Cependant, cela peut être un élément différenciant : c'est en quoi Mistral se démarque avec des modèles de taille raisonnable mais plus performants dans les langues européennes. L'entreprise n'a pas révélé le détail de la composition de son dataset d'entraînement ni sa construction, mais affirme en interview avoir largement investi dans le travail de nettoyage et de diversification du dataset.

Par ailleurs, les datasets présenté dans cette section sont largement issue de CommonCrawl, alors qu'il existe de nombreuse sources de données. Il est cependant difficile de les mettre en commun, notamment à cause de la duplication. Voir plusieurs fois la même phrase pour un LLM pendant l'entraînement peut faire baisser la performance de généralisation. Ainsi, la recherche dans ce sens continue avec TxT360 [LT24] par exemple qui produit également un dataset de 15T tokens, mais provenant de plus de sources différentes et dédupliqué globalement [5].

Nous avons mis en lumière dans cette section combien la qualité du dataset et la diversité des tokens permettent d'obtenir des corpus d'entraînement performants pour entraîner des LLM contenant nombre des avancées que nous avons présentées dans les précédentes séances. Il reste à construire ces fameux tokens !

8.1.2 Comment rendre intelligible du texte pour un modèle de langage ?

Le modèle ne peut pas travailler avec du texte de manière brute : il faut trouver la meilleure manière de représenter des phrases sous forme de nombres. Pour le faire, il existe plusieurs approches différentes mais toutes ont comme point commun de chercher à représenter des *mots* par des nombres.

Naïvement, on peut se proposer de séparer les phrases par des espaces et on obtiendra l'ensemble des mots. En associant à chaque mot un identifiant (un nombre) alors on vient bien de passer du texte à une représentation compréhensible par un modèle. Cependant, dès que l'on va considérer un modèle avec plusieurs langues voire des langages informatiques, on aura un dictionnaire qui peut être très grand. Cela nécessitera d'avoir un modèle conséquent et complexifiera l'entraînement. Pour réduire un peu la dimension du vocabulaire, on peut extraire la ponctuation, bien qu'elle soit parfois *collée* à un mot :

mots, → mots ,

Cela suppose un nombre *fini* de mots. Dans des langues agglutinantes comme le turc ou le japonais, voire l'allemand dans une moindre mesure, on ne peut pas obtenir ce nombre fini. On ne capturera pas non plus la spécificité de ces langues. Si s'attaquer à des mots entiers n'est pas la bonne approche, on peut suivre l'opposé : associer à chaque **caractère** un identifiant. Cette fois, on aura un dictionnaire beaucoup plus petit mais aussi mais la puissance de représentation de chaque caractère est limitée.

La bonne solution doit se trouver entre les deux. Partons du principe que des *mots* fréquents devraient apparaître dans le dictionnaire, mais que des *mots* peu fréquents devraient être décomposés en des sous-mots plus utiles. Par exemple :

antisymétrique → anti symétrique

Dans cet exemple, le mot *antisymétrique* est construit à partir de deux **tokens** : anti et symétrique. On appellera donc **token** une suite de caractères, une définition plus large que celle de mots. De cette manière, on peut atteindre un vocabulaire de taille raisonnable tout en ayant un nombre de tokens utiles élevé. Par exemple, BERT [DCLT18] utilise ce genre d'approche pour tokeniser une phrase avec un vocabulaire de 30000 tokens. On obtient :

Un e phrase en f ran çais

5. À la date où nous écrivons ces lignes, fin octobre 2024, aucun modèle de langage à notre connaissance n'a été entraîné avec ce dataset.

Notons que les espaces ne sont pas compris dans ce tokenizer. Chaque token est associé à un nombre, et le modèle verra donc cette phrase comme le vecteur suivant (dans le cas de BERT [6]) :

[12118 , 1162 , 7224 , 4035 , 175 , 4047 , 26293]

On vient de réduire une phrase de 22 caractères en 7 tokens ! Si l'on tokenize cette fois pour travailler avec un modèle Mistral :

Une	phrase	en	français
16803	15572	1249	15067

Ou pour un modèle Gemma :

Une	phrase	en	français
19750	20911	659	24913

Contrairement à BERT, les espaces sont parfois intégrés aux tokens, et on observe que les tokens sont aussi plus long pour les tokenizers plus récents. Il nous reste à comprendre comment obtenir cela.

Byte-Pair Encoding et Word Piece

[SHB15] adapte l'algorithme Byte-Pair Encoding (BPE) initialement construit pour la compression [Gag94]. Cet algorithme a besoin du résultat d'un tokenizer : un texte tokenisé et le vocabulaire associé, mais également d'une taille de vocabulaire souhaitée. Puis, BPE identifie les tokens qui apparaissent le plus souvent ensemble pour former un nouveau token du vocabulaire, composé de deux tokens déjà présents. Cette étape est répétée jusqu'à ce qu'on obtienne un vocabulaire d'une taille souhaitée.

Exercice 8.1. *On considère le vocabulaire suivant $V = [a, c, e, h, i, n, p, s, t]$ et les mots suivant avec leurs fréquences :*

("chat", 5), ("chats", 3), ("chien", 2), ("patte", 5)

1. *Après avoir écrit les mots avec le vocabulaire de base, quelle est la paire la plus fréquente ?*
2. *Réécrire les mots avec un nouveau symbole, associé à la paire la plus fréquente.*
3. *Recommencer les deux premières étapes.*
4. *Quelle était la longueur du texte avant ? Et maintenant ?*

Solution.

1. La paire "at" est la plus fréquente avec 13 occurrences.
2. On note "#" le nouveau symbole pour remplacer "at". On a :
   - ("chat", 5) → ("ch#", 5)
   - ("chats", 3) → ("ch#s", 3)
   - ("chien", 2) → ("chien", 2)
   - ("patte", 5) → ("p#te", 5)
3. La paire "ch" est la plus fréquente avec 10 occurrences, on a maintenant avec "&" comme symbole pour "ch" : ("&#", 5), ("&#s", 3), ("&ien", 2), ("p#te", 5)

6. Voir le space Tokenizer Playground sur Hugging Face

4. Nous avions un texte de longueur de 70 caractères, et maintenant 47 en augmentant la taille du vocabulaire de 2 tokens : $V = [c, e, h, i, l, m, o, t, \# = at, \& = ch]$

□

GPT-2 [RWC+19] utilise l'ensemble des 256 caractères unicode comme vocabulaire de base, puis a fait 50000 combinaisons pour atteindre un vocabulaire de taille 50257 en ajoutant un token spécial pour la fin de phrase. Ce fonctionnement est le plus fréquent : on tokenize en utilisant les caractères utf-8, puis on construit des tokens avec plus de sens.
Nous avions utilisé l'exemple de BERT, mais ce modèle n'utilise pas l'algorithme BPE pour la tokenization, il utilise WordPiece introduit initialement pour un problème de traduction Japonais-Coréen [SN12] puis décrit plus en détail dans [WSC+16]. L'algorithme est très similaire à BPE et se différencie par le choix de la paire à former. Au lieu de prendre la paire la plus fréquente, WordPiece essaie de maximiser la vraisemblance des données d'entraînement une fois que la paire est ajoutée au vocabulaire.

Concrètement, en reprenant l'exercice précédent, la paire la plus fréquente reste "at", mais comme les caractères "a" et "t" sont présent souvent, on obtient un score de :

$$S(\text{"at"}) = \frac{\text{Fréquence de "at"}}{\text{Fréquence de "a"} \times \text{Fréquence de "t"}} = \frac{13}{13 \times 18} \simeq 0.05$$

Mais si l'on regarde, la paire "ch" est la plus vraisemblable :

$$S(\text{"ch"}) = \frac{10}{10 \times 10} = 0.1$$

C'est cette paire qui sera sélectionné pour continuer la tokenization. La philosophie de WordPiece n'est pas de tokenizer *le plus possible* le texte (comme BPE), mais plutôt d'évaluer ce que l'on perd en construisant le nouveau caractère et en s'assurant que ça vaille le coup.

SentencePiece et Unigram

Si BPE et WordPiece utilisent une taille de vocabulaire faible puis l'augmentent, Unigram [Kud18] fait l'inverse. À partir d'un vocabulaire contenant l'ensemble des caractères du texte et par exemple des sous-chaînes de caractères les plus communes, l'objectif d'Unigram est de supprimer des tokens pour réduire le vocabulaire à la taille souhaitée. Cette réduction est faite de sorte à minimiser l'augmentation d'une loss calculée sur les données d'entraînement. Autrement dit, on supprime les chaînes de caractères qui modifient le moins le dataset.

Puisque Unigram n'est pas construit par une succession de règles de rapprochement, il est possible qu'il y ait plusieurs manières de tokeniser un mot. Pour résoudre ce problème, pendant la phase de construction du vocabulaire, la probabilité de chaque token est conservée. Ainsi, la tokenisation la plus probable est choisie. On peut donc avoir des mots similaires tokenisés de manières différentes selon le contexte.

Unigram n'est jamais utilisé seul, mais conjointement avec SentencePiece [KR18]. L'algorithme a été proposé pour résoudre un problème dont les précédents souffrent : les mots ne sont pas forcément séparés par des espaces dans toutes les langues. Ainsi, SentencePiece inclut les espaces dans le jeu de caractères à utiliser. Puis on utilise Unigram ou BPE pour pouvoir construire un vocabulaire de la taille que l'on souhaite. En pratique, les autres algorithmes se sont adaptés en considérant un espace comme un caractère du vocabulaire de base.

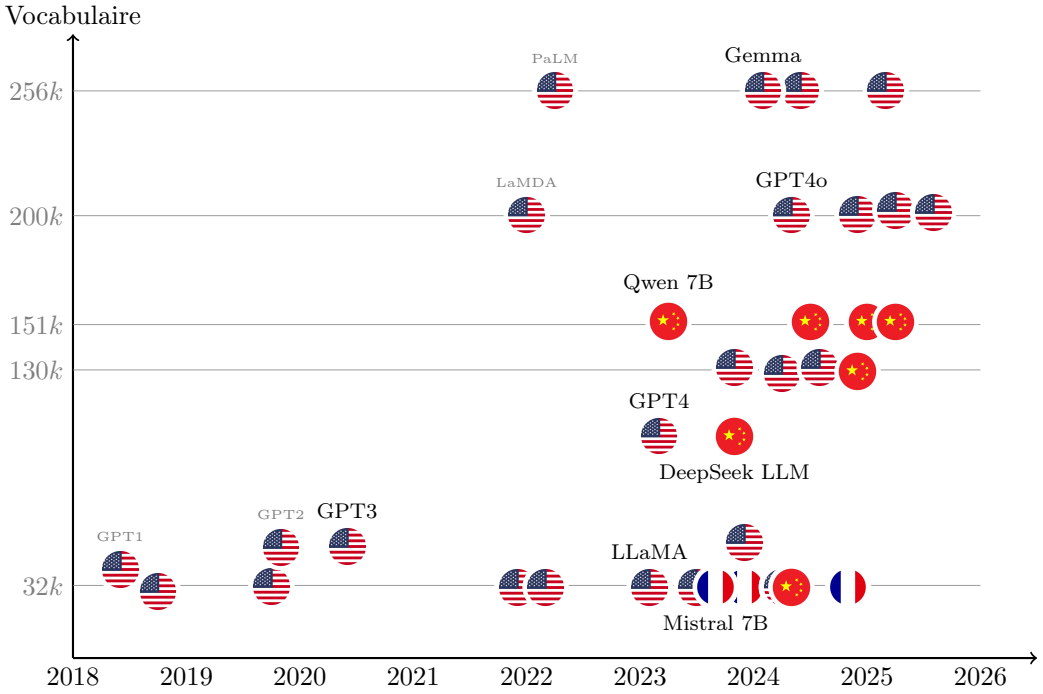


FIGURE 8.1 – Taille de vocabulaire des principaux modèles

L'algorithme SentencePiece est aujourd'hui le plus utilisé, en combinaison avec BPE. Concernant la taille du vocabulaire, s'il semblait y avoir un consensus aux alentours de 32k tokens jusqu'en 2023, les *nouveaux* modèles de langage explore des valeurs plus élevées. Cela s'explique par deux principales raisons :

1. **Tokens spécifiques** : pour mieux capturer à la fois le langage et le code, avoir plus de tokens permet de compresser plus efficacement le texte d'apprentissage.

2. **Langues différentes** : avec de plus en plus de modèles polyglottes, conserver une efficacité de représentation par le tokenizer passe par une augmentation de la taille.

[?] s'intéresse particulièrement à cette dernière et montre que pour l'algorithme BPE une taille de vocabulaire de 33k est optimale pour du texte exclusivement en anglais. Si nous avons plusieurs langue alors une taille de vocabulaire jusqu'à trois fois plus grande est plus adapté : si on ne change pas cette taille de vocabulaire, alors il y aura une augmentation de 68% des coûts d'entraînement à cause de ce problème, sans compter les coûts lors de l'inférence.

Cependant, on observe plutôt des valeurs autours de 151k voire +200k tokens, ce qui est largement plus que la préconisation de l'article. Cet écart peut être expliqué par les langues retenue par l'article : anglais, allemand, français, italien et espagnol. Les langues asiatiques nécessite des ajustements par rapport aux langues européennes par exemple.

Puisque les tokenizers doivent être *ouvert* pour pouvoir estimer les coûts d'appels aux modèles, nous pourrions deviner de quoi est composé le dataset d'entraînement. [?] réalise ce travail. L'approche est d'abord validée avec les modèles dont les données d'entraînement sont connues comme les modèles LLaMa. Puis la méthodologie est appliqué à des modèles privatifs comme GPT4o ou Claude. On note alors par exemple que GPT4o s'est entraîné avec plus de donnée de langue autre que l'anglais que GPT3.5 et que LLaMa 3 est le modèle dont le mix de texte dans une langue différente que l'anglais est le plus fort [7].

7. Plus précisément, les langues considérées sont essentiellement latine ou cyrillique.

Également, Claude semble s'est entraîné en grande partie sur du code. Il semblerait donc que pour une performance optimal en code des modèles lui soit dédié, peut-être en partie pour l'importance de la tokenization. Ainsi, la famille Qwen 2.5 contient plusieurs modèles de taille différentes dédiées au code [?], nous avons également vu Codestral de Mistral qui utilise par ailleurs l'architecture Mamba et non transformer, et bien sur phi-1 [GZA+23] de Microsoft.

Finalement, même si les modèles sont fermé, nous pouvons avoir une idée de l'orientation qui a été souhaité par ses concepteurs : c'est un premier pas dans l'explicabilité des modèles.

8.2 Alimenter un modèle avec des tokens

Maintenant que nous avons un ensemble de token, nous devons le *transmettre* au modèle. Comme précisé dans l'introduction, nous n'allons pas présenter comment est défini un modèle à l'architecture transformers. Nous nous intéressons plutôt à des idées qui peuvent être réutilisée en Machine Learning, dans la continuité de la première partie : autour des tokens.

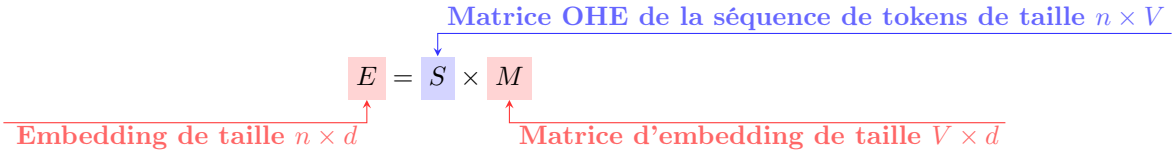
8.2.1 Embedding : du nombre vers le vecteur

Un embedding est une manière de représenter un mot par un vecteur dense de taille fixe. Nous avons déjà vu cette idée avec des vecteurs *sparse* : le one-hot-encoding (OHE). Pour rappel, le OHE transforme une colonne constituée de p modalités en p colonnes où chaque colonne représente une modalité et vaut 0 ou 1 si la modalité que la colonne représente est présente dans la colonne initiale. Par exemple avec trois modalités pour quatre observations :

$$\begin{pmatrix} \text{reine} \\ \text{roi} \\ \text{chien} \\ \text{reine} \end{pmatrix} \rightarrow \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix}$$

Un des défauts du OHE est que la taille du vecteur peut être très grande et souvent vide, rendant difficile pour un modèle de capturer les relations sémantiques entre les mots. Par exemple, si on a des observations qui représente des êtres vivants et que l'on a les modalités : reine, roi et chien, on aimerait que roi et reine soient *proche*. Or avec un one-hot-encoding les trois seront à même distance !

Il faudrait pouvoir avoir des vecteurs représentatif plus intelligent. Un mot dans différents contexte peut prendre plusieurs signification. Ainsi, représenter un mot par un unique nombre comme nous le faisons actuellement ne permet pas à un modèle de pouvoir capturer toutes ces nuances. Nous aimerions donc représenter les tokens en vecteurs dense[8]. Pour une séquence de longueur $n \in \mathbb{N}^*$ tokenizé avec un vocabulaire de taille V, que l'on veut représenter par un embedding de taille $d \in \mathbb{N}^*$, on a :



Les valeurs de la matrice M sont ceux que le modèle cherche à apprendre pour représenter au mieux les mots. Si la représentation est performante, alors que le modèle de langage a toute les chances de devenir plus performant sur les tâches qu'on lui confiera. Pour initialiser les valeurs de la matrice M, on génère les valeurs aléatoirement selon une distribution normale ou uniforme. De cette manière, le modèle est capable de s'adapter aux données spécifique du problème et cette solution est très simple à mettre en place. Cependant, la convergence peut prendre plus de temps puisqu'il y a tout à apprendre. Pour éviter cela, on peut utiliser des embeddings publics déjà entraîné pour accélérer la phase d'entraînement de notre modèle. Cependant, il est probable que l'embedding ne soit pas optimal pour notre cas d'usage. Il y aura forcément une phase d'ajustement du modèle pour être le plus performant possible.

8. Avec des valeurs non nulle dans la majorité des coordonnées.

8.2.2 Et la position ?

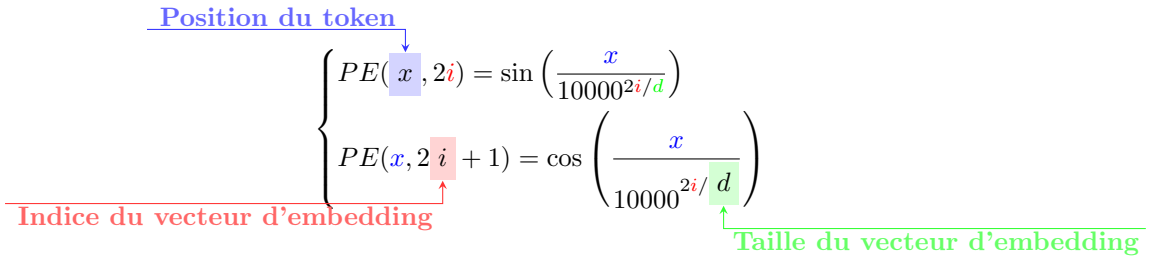
Il reste encore des difficultés pour pouvoir comprendre le langage. La position des mots dans une phrase compte ! Et pourtant, le mécanisme au coeur de l'architecture Transformers n'est pas capable de comprendre la position *nativement*. Ainsi, dans la phrase : *Un étudiant explique à un autre étudiant*, le mot *étudiant* fait référence à deux personnes distincte. Nous le savons parce que la position compte ! Mais ce que reçoit un modèle est sa version tokenizé pour LLaMa 3 :

Un	ét	udiant	expl	ique	à	un	autre	ét	udiant
1844	14240	41939	3327	2428	3869	653	47838	14240	41939

Puis le modèle va transformer ce vecteur avec l'embedding qu'il aura appris pendant l'entraînement. Pourtant, la représentation de *étudiant* sera la même, avec la difficulté supplémentaire qu'il s'agit de deux tokens différent !

On aimerait pouvoir transmettre l'information de la position au modèle. La première idée serait de simplement ajouter à chaque coordonnée du vecteur représentant le token dans l'embedding l'indice de sa position. Ce n'est pas au programme du cours, mais cette solution entraîne de très forte instabilité dans l'entraînement du modèle et les valeurs de l'embedding étant centrée autour de zéro avec une petite variance, l'ajout de la position bruite complètement l'information contenu dans l'embedding. Il nous faut trouver une manière qui ne modifie pas trop cette connaissance.

On pourrait utiliser la représentation en binaire de la position et faire cycler les bits pour obtenir la taille du vecteur initial. Puisque cela revient à ajouter des 1 aux coordonnées parfois, cela va entraîner des vecteurs ayant des *changements de directions* très brusque : c'est clairement contre indiqué pour de l'optimisation.
Cependant, si à la place on considérait des sinus et des cosinus, on obtiendrait quelque chose de cyclique à nouveau et surtout quelque chose de lisse ! C'est la solution proposée initialement dans l'article et se résume avec ces équations :



La valeur 10000 a été obtenue expérimentalement et ne semble pas avoir de sens plus profond [9]. La valeur d est également appelé la dimension du modèle : c'est à partir de celle-ci que le reste des dimensions de l'architecture est calibré.
Prenons par exemple $d = 4$. Alors la taille du vecteur d'embedding positionnel que l'on va créer sera de longueur 4 et on a :

- Token à la position 0 :

$$\begin{aligned} v &= \left(\sin\left(\frac{0}{10000^{2\times 0/4}}\right), \cos\left(\frac{0}{10000^{2\times 1/4}}\right), \sin\left(\frac{0}{10000^{2\times 2/4}}\right), \cos\left(\frac{0}{10000^{2\times 3/4}}\right)\right) \\ &= (0, 1, 0, 1) \end{aligned}$$

- Token à la position 1 :

$$\begin{aligned} v &= \left(\sin\left(\frac{1}{10000^{2\times 0/4}}\right), \cos\left(\frac{1}{10000^{2\times 1/4}}\right), \sin\left(\frac{1}{10000^{2\times 2/4}}\right), \cos\left(\frac{1}{10000^{2\times 3/4}}\right)\right) \\ &= (0.84, 0.99, 0.01, 0.99) \end{aligned}$$

9. À notre connaissance.

- Token à la position 2 :

$$\begin{aligned} v &= \left(\sin\left(\frac{2}{10000^{2\times 0/4}}\right), \cos\left(\frac{2}{10000^{2\times 1/4}}\right), \sin\left(\frac{2}{10000^{2\times 2/4}}\right), \cos\left(\frac{2}{10000^{2\times 3/4}}\right)\right) \\ &= (0.90, 0.98, 0.02, 0.99) \end{aligned}$$

Cette définition a également une jolie propriété de rotation. Cette idée de rotation est au coeur de la manière moderne d'encoder la position dans les derniers modèle. Nous ne la couvrons pas ici, la manière que nous venons de présenter donne déjà l'ensemble des clés pour comprendre.
Le vecteur d'encoding positionnel que l'on vient de définir agit comme une boussole pour le modèle pour être capable de comprendre la position des mots. Chacune des valeurs peut être représenté comme des marques sur une scène lors d'un spectacle, permettant aux intervenants de se placer.
Par ailleurs, on sait que les modèles de langage ne sont pas capable de traiter des longueurs de texte arbitraire, : ils ont une limite. Si le texte d'entrée dépasse cette limite, alors le texte est découpé en plusieurs partie capable d'être traitée par le modèle. Une part important d'information risque donc d'être perdu, mais l'encoding positionnel que l'on vient de décrire lui permet de ne pas être complètement perdu.

Jusqu'ici, nous avons compris comment obtenir des tokens et les transmettre au modèle en lui indiquant de manière claire où sont situé les tokens les uns par rapport aux autres. Il n'est pas dans le cadre du cours de comprendre ce qui se passe après pendant l'entraînement. Si nous considérons à présent que l'on possède un modèle entraîné, il nous reste à comprendre comment on génère du texte !
Les modèles de langage prédisent pour chaque séquence de token la probabilité pour chaque token d'être le suivant. Pour écrire du texte, il suffirait donc de simplement sélectionner le plus probable. Cependant, *générer* du texte n'est pas si simple. En suivant cette méthode naïve, on arrive à des textes dégénérés et de faible qualité.

Considérons un LLM qui, pour une séquence de token, prédit le prochain. Alors, il produit un vecteur de taille $d \in \mathbb{N}$ avec d le nombre de token dans le vocabulaire du modèle. Si l'on essaye de dépasser la méthode naïve qui consiste à sélectionner le token avec la probabilité la plus forte, on peut proposer de tirer aléatoirement le prochain token en suivant la distribution induite par ces probabilités. On génère effectivement un texte plus riche que la méthode naïve, mais on ne règle pas le problème de dégénération. Aussi, même un token très peu probable peut être sélectionné, ce qui fait gagner en richesse linguistique mais perdre en sens.

8.2.3 Ajuster les probabilités avec la température

Nous avons besoin d'avoir un vecteur de probabilité qui soit généré. Un réseau de neurones ne le fait pas naturellement mais en ajoutant la fonction d'activation softmax on peut alors former un vecteur de probabilité :

$$p_i = \frac{\exp\left(\frac{-\varepsilon_i}{\tau}\right)}{\sum_{j=1}^{d} \exp\left(\frac{-\varepsilon_j}{\tau}\right)}$$

i-ème valeur initiale du vecteur

Température

Le paramètre τ est appelé la température par inspiration du domaine de la thermodynamique en physique. Étudions plus en détail les possibilités de cette fonction d'activation.

Exercice 8.2 (Température). *On considère $\varepsilon_1, \varepsilon_2 \in \mathbb{R}$ tels que $\varepsilon_1 < \varepsilon_2$. On définit $0 < \tau_1 < \tau_2$ deux températures.*

1. *Pour une température $\tau > 0$ fixée, on note p_1 et p_2 les valeurs associées à la transformation softmax de ε_1 et ε_2. A-t-on que $p_1 < p_2$?*
2. *Comment varie la valeur de p quand τ varie ?*
3. *Calculer $\lim_{\tau \to +\infty} p_i$.*

Solution. On reprend les notations de l'exercice et les notations précédentes.

1. Par définition de p_1 et p_2, on a que $p_1 > p_2$, donc la fonction ne conserve pas l'ordre selon ε.
2. On sait que l'ordre n'est pas conservé, mais on ne sait pas dans quel sens *varie* la valeur p_i en fonction de τ. Calculons pour $i \leqslant d$ avec $d \in \mathbb{R}$ la taille du vecteur $(\varepsilon)_{i \leqslant d}$.

$$\frac{\partial p_i}{\partial \tau} = \frac{e^{-\frac{\varepsilon_i}{\tau}} \left(\sum_{j=1}^{d} \varepsilon_j e^{-\frac{\varepsilon_j}{\tau}} - \varepsilon_i \right)}{\left(\tau \sum_{j=1}^{d} \varepsilon_j e^{-\frac{\varepsilon_j}{\tau}} \right)^2}$$

   On a donc que :

$$\frac{\partial p_i}{\partial \tau} < 0 \iff \varepsilon_i > \sum_{j=1}^{d} \varepsilon_j e^{-\frac{\varepsilon_j}{\tau}}$$

   Autrement dit, la valeur de la probabilité estimée va diminuer quand elle est supérieure à cette quantité. Donc quand la valeur prédite est grande, elle va décroitre quand τ augmente et réciproquement quand τ diminue.

3. On a clairement que $\lim_{\tau \to +\infty} p_i = \frac{1}{d}$ autrement dit on atteint une distribution uniforme. C'est cohérent avec le résultat précédent !

□

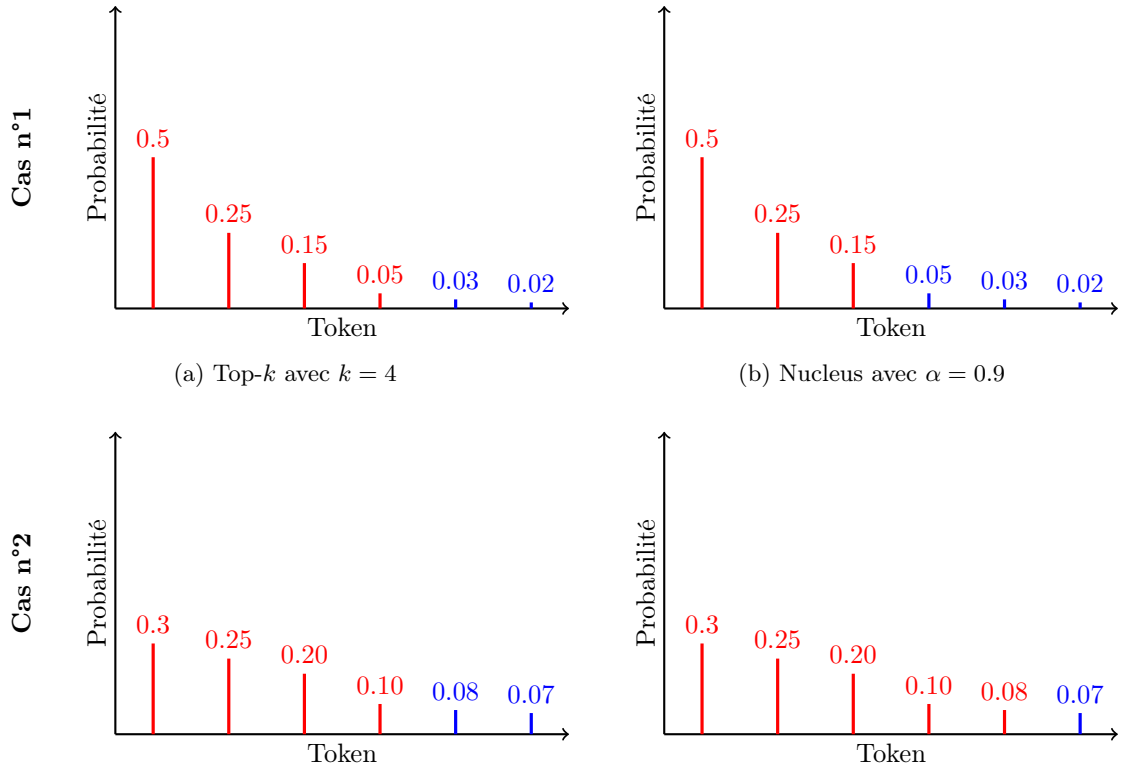
Avec l'exercice précédent, on comprend mieux le nom de *température* : quand la température est élevée, les probabilités ont tendance à s'uniformiser, ce qui correspond à un état physique excité où l'issue est beaucoup plus aléatoire. Réciproquement, quand la température est proche de zéro, le comportement est beaucoup plus prévisible avec moins de prise de risque pour le LLM. Mais cette technique ne suffit pas pour éviter des textes dégénérés.

8.2.4 Top-k et *nucleus sampling*

[FLD18] propose une méthode qui permet de limiter la dégénération de texte tout en augmentant sa richesse linguistique. Il s'agit de la méthode *top-k sampling*.
Plutôt que de considérer l'ensemble des tokens pour tirer selon la distribution induite, on ne considère cette fois que les $k \in \mathbb{N}$ tokens les plus probables. On ajoute donc un hyper-paramètre à régler finement pour éviter le mieux possible une dégénération. Cette méthode est largement utilisée de nos jours pour l'ensemble des LLM et donc des chatbots.

The curious case of neural texte de*generation* [HBD+19] propose une autre manière de résoudre le même problème initial. Il repose sur l'observation que dans certains contextes la suite d'une séquence de tokens est *évidente* et qu'il n'y a pas besoin de considérer beaucoup de tokens, et que dans d'autres

contextes la suite est moins directe, d'où la nécessité d'avoir un grand nombre de tokens parmi lesquels choisir. Cette souplesse n'est pas permise par le top-k sampling, mais par le *nucleus sampling*.
Au lieu de choisir les k tokens les plus probables pour continuer, on sélectionne l'ensemble des tokens qui font que l'on a α% de la distribution. On gagne la souplesse du nombre de tokens à considérer mais on ne règle pas le problème d'un hyper-paramètre à régler. On peut observer la différence de comportement entre les deux sampling avec la figure (8.2).



(a) Top-k avec $k = 4$

(b) Nucleus avec $\alpha = 0.9$

(c) Top-k avec $k = 4$

(d) Nucleus avec $\alpha = 0.9$

FIGURE 8.2 – Exemple pour deux sélections de tokens de deux stratégies de sampling

Combiner une valeur bien choisie pour τ permet de mieux calibrer la valeur α du nucleus sampling. En revanche, cela n'a pas d'impact pour le top-k sampling comme vu dans l'exercice précédent.
Ces différentes valeurs sont donc clés pour pouvoir générer un texte aligné avec ce que l'on souhaite : un texte créatif ou informatif et rigoureux. Les paramètres peuvent être modifiés dans la plupart des ChatBots lors de l'instruction, et il convient de le préciser quand on rend compte d'expérimentations dans un article. On peut prendre par exemple l'article *LIMA : Less is more for alignment* [ZLX+23] qui précise systématiquement ces méthodes et les paramètres utilisés pour produire le prochain mot.

8.3 Quelques tendances

La volonté de pouvoir converser avec une machine n'est pas nouvelle : dès 1954 avec l'expérience Georgetown-IBM on montre sur des tâches de traduction du russe vers l'anglais que la machine peut traiter du langage. De nombreuses autres techniques ont tenté de traiter ce sujet, mais c'est à partir de 2017 avec l'article *Attention is all you need* [VSP+17] que l'on a la première brique de la révolution des années 2020. Il est difficile de documenter l'ensemble des évolutions et des idées qui sont apparu pendant cette période. Nous nous proposons dans cette section de discuter de quelques éléments et tendances que l'on observe.

8.3.1 Les *Scaling laws*

L'objectif principal en Machine Learning est d'obtenir un modèle performant capable de généraliser à des données non vues pendant l'entraînement. [Vap99] montre pour la première fois que la capacité d'un modèle à généraliser à partir de données d'entraînement dépend de sa complexité et de la quantité de données disponibles [10]. La principale difficulté pour appliquer concrètement les résultats obtenus sont dans le calcul de la *complexité*.

[KMH+20] propose les premières *Scaling Laws* pour les réseaux de neurones avec une architecture transformers. Elles étendent l'idée de Vapnik en fournissant des relations quantitatives précises entre la taille du modèle, la quantité de données et la performance. Il existe plusieurs équations dans ces lois, nous n'en considérerons ici que deux :

Nombre de paramètres

$$\mathcal{L}(N) = \left(\frac{N_c}{N}\right)^{\alpha_N} \qquad \text{avec } \alpha_N \sim 0.076, N_c \sim 8.8 \times 10^{13}$$

$$\mathcal{L}(D) = \left(\frac{D_c}{D}\right)^{\alpha_D} \qquad \text{avec } \alpha_D \sim 0.095, D_c \sim 5.4 \times 10^{13}$$

Taille du dataset

Exercice 8.3 (Lien entre N et D)**.** *On considère les deux lois précédentes.*

1. *Si l'on doit doubler l'un des deux paramètres, quel gain en performance peut-on espérer ?*
2. *Si l'on modifie N alors on modifie la valeur de la fonction de perte. Cependant, il faut également modifier D : déduire la valeur de D en fonction de N.*
3. *Dans l'article, la relation suivante est proposée :*

$$\mathcal{L}(N, D) = \left[\left(\frac{N_c}{N}\right)^{\frac{\alpha_N}{\alpha_D}} + \frac{D_c}{D}\right]^{\alpha_D}$$

Est-ce cohérent avec les deux premières lois ?

Solution. 1. Si l'on double le nombre de paramètre, alors on réduit d'un facteur $2^{-\alpha_N}$ la fonction de perte soit 0.95, autrement dit une baisse de 5%. Pour la taille de dataset on réduit d'un facteur $2^{-\alpha_D}$ soit 0.93 donc une baisse de 7%.

2. On a que $\mathcal{L}(N) = \mathcal{L}(D) \iff D = D_c \left(\frac{N}{N_c}\right)^{\frac{\alpha_N}{\alpha_D}}$. En faisant des approximations on obtient : $D \sim 377 \times N^{0.8}$.

3. Quand N tends vers l'infini on retrouve la loi pour D et inversement, ainsi cela semble cohérent.

□

10. Pour plus de détails, voir la section (G.1)

Ces deux lois concernent le nombre de paramètres, hors embedding, et la taille du dataset en token [11]. Le troisième paramètre identifié pour obtenir des performances est le budget de calcul C qui a lui aussi sa loi. En pratique, cela correspond à la puissance machine que l'on a disposition et pour combien de temps. Ce budget se compte en FLOPS et dépend de N et D :

$$FLOPS(N, D) = k\,ND$$

Constante dépendante du matériel

On prend généralement $k = 6$. Ainsi, pour un budget de calcul fixé C, nous sommes capables d'identifier les paramètres (N, D) optimaux, et [KMH+20] conclu que N est de l'ordre de $C^{0.73}$ et D est de l'ordre de $C^{0.27}$. Autrement dit, avoir un plus grand modèle me permet un gain plus grand d'optimisation de la fonction de perte. Partant de cette conclusion, l'ensemble de l'écosystème a commencé à construire des modèles de plus en plus grand et ils sont effectivement plus performant que les précédents. Cependant, une étude de Deepmind propose d'autres valeurs pour les scaling laws, et les *prouve* avec son modèle Chinchilla [HBM+22]. Selon Deepmind on a cette fois N est de l'ordre de $C^{0.5}$ et D est de l'ordre de $C^{0.5}$. Autrement dit, il faut entraîner un modèle sur plus de tokens !
Pour comprendre l'écart entre les deux prédictions, [PS24] pointe deux principales raisons :

1. Le nombre de paramètre n'est pas calculé de la même manière : les lois de Kaplan exclus les paramètres d'embedding alors que Chinchilla les inclus

2. La taille des modèles utilisé par Kaplan pour apprendre les bons paramètres de lois étaient de trop petites taille et ont donc sous-estimé l'impact du dataset

Regardons concrètement comment ces deux lois ont impactés la conceptions de modèles de langage avec la figure 8.3.

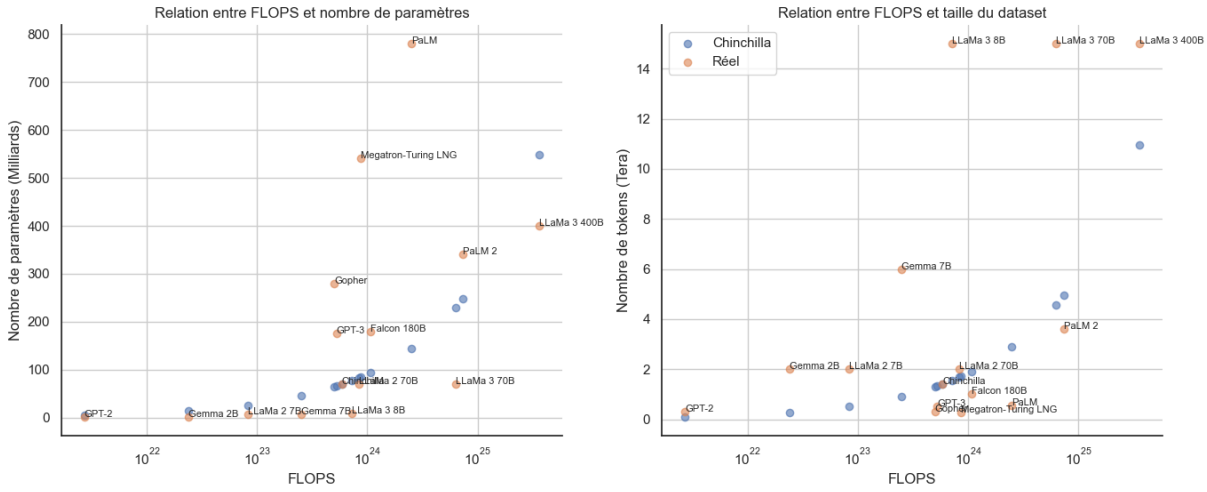


FIGURE 8.3 – Comparaison entre les lois de Chinchilla et les modèles disponibles

Premièrement, on observe bien que les prédictions de Chinchilla sont des lois de puissance entre le nombre de paramètre / la taille du dataset et le budget de calcul. Deuxièmement, on observe qu'à part Chinchilla, aucun modèle ne suit vraiment les prédictions. Troisièmement, les modèles les plus *anciens* tendent à avoir plus de paramètre que ce que prévoit Chinchilla par rapport aux modèles les plus récents. En contrepartie, ces mêmes modèles utilisent des datasets moins conséquent.

Le dernier point est une conséquence direct du temps : avant Chinchilla l'emphase était mise sur la taille des modèles avant la quantité et la qualité du dataset. Les lois de Kaplan supportant cela, les travaux ont été mené dans ce sens. Ce n'est que les années qui suivent Chinchilla que l'on a commencé à porter plus d'attention à l'autre aspect. Cependant, les modèles plus récent semblent être sur-entraîné par

11. Dépendant donc de la méthode de tokenization et de la taille du vocabulaire.

rapport à ce que prévoit Chinchilla : c'est exactement la remarque que Meta souligne dans ses publications de ses modèles LLaMa. Ce n'est pas que les lois de Chinchilla sont fausse, elles ne visent pas la même chose que Meta ! Chinchilla cherche à trouver le point optimal entre coût de calcul, taille du dataset et nombre de paramètre. Meta cherche plutôt à avoir un modèle performant : ce n'est pas le même objectif.

8.3.2 La *quantization*

Nous avons jusqu'ici évoqué de très grands modèles de langage qu'on appelle *Large Langage Model* (LLM). Ils sont cependant réservé à une élite qui peut payer les ressources de calculs pour les exploiter. On observe alors en 2023/2024 plutôt la conception de *petits* modèles entraîné sur de large corpus de texte qualitatif. Au delà de la performance métrique vis à vis du coût de calcul, ces *petits* modèles tiennent dans la poche, plus précisément dans un smartphone ! Pour que cela puisse fonctionner, il faut que le modèle soit de taille modeste pour à la fois ne pas pénaliser le stockage et avoir des réponses rapides.
Cependant, ces modèles étant plus petit ils sont moins performant, ce qui ne règle pas le problème. Une deuxième tendance, qui accompagne la précédente, est celle de faire *maigrir* les modèles. Un gros modèle pèse lourd en stockage parce qu'il faut conserver des milliards de nombre flottant avec une grande précision, autrement dit sur un grand nombre de bits. Que se passerait-il si on stockait ces mêmes nombres mais avec un nombre de bits plus restreint ? Cela s'appelle la quantization, et cela permet d'avoir des modèles moins lourd, sans pour autant être des modèles avec moins de paramètre. Le prix a payer semble être d'avoir un modèle moins performant. Et pourtant !

[KAS+24] montre qu'en réalité un modèle quantizé jusqu'à 4 bit obtient des performances similaires au modèle avec la précision maximale. Une quantization plus prononcé dégrade très fortement en revanche la qualité du modèle. Pour se donner un ordre d'idée du gain en terme de taille :

Modèle	Initial	Q8	Q6	Q4
LLaMa 3.2 3B	6.43	3.42	2.74	2.11
LLaMa 3.2 1B	2.48	1.32	1.09	0.87
Mistral 7B v0.3	14.50	7.70	5.95	4.37

TABLE 8.5 – Taille du fichier (GB) contenant le modèle selon le niveau de quantization

Cela permet à un utilisateur d'exploiter des modèles de hautes qualité avec son installation personnelle pour pouvoir développer ses propres cas d'usage. Par exemple un service de discussion avec les fichiers présents dans son ordinateur, un système de résumé personnalisé...

Mais la quantization ne concerne pas forcément que la modification d'un modèle après avoir été entraîné. On peut également se demander s'il est possible de s'entrainer en plus petite précision : cela réduirait le coût d'entraînement des modèles. Nous ne rentrerons pas plus dans le détail de la quantization pendant l'entraînement, concentrons-nous sur une équation proposée dans [KAS+24].

En regardant la figure 8.3, les modèles les plus récent semble privilégier la quantité de donnée au nombre de paramètre pour un budget de calcul fixé [12]. L'article propose l'équation suivante pour modéliser l'impact sur la performance du modèle d'une quantization de précision P notée δ :

$$\delta(N, D, P) = C \left(\frac{D^{\gamma_D}}{N^{\gamma_N}} \right) e^{-\frac{P}{\gamma_P}}$$

Où C, γ_D, γ_N et γ_{post} sont des constantes positives. On comprend que la perte en performance d'un modèle de langage quantizé est grandement liée à un ratio entre le nombre de token d'entraînement D et le nombre de paramètre du modèle N. Ainsi, quand on s'entraîne avec beaucoup de donnée et peu de paramètres, il y a un nombre de token au delà duquel ajouter des tokens a en réalité un impact

12. On dit que les modèles sont sur-entraîné.

négatif sur la performance du modèle quantizé. L'intuition derrière cela est que les poids vont contenir beaucoup d'informations et donc la performance du modèle va souffrir d'une baisse de précision lors de la quantization.

Exercice 8.4 (Taille critique)**.** *On considère un modèle entraîné en pleine précision, puis il est quantizé. La loi d'échelle s'écrit comme :*

$$\mathcal{L}(N, D, P) = AN^{-\alpha} + BD^{-\beta} + \delta_{PTQ}(N, D, P)$$

Trouver la taille D_{crit} telle que ajouter des tokens lors de l'entraînement va réduire la performance du modèle quantizé.

Solution. On cherche à résoudre :

$$\frac{\partial \mathcal{L}}{\partial D}(N, D, P) > 0 \tag{8.1}$$

Autrement dit, on veut connaître la taille à partir de laquelle la valeur de la fonction de perte va croître, selon la taille du dataset D. Pour commencer, on a :

$$\begin{aligned}
\frac{\partial \mathcal{L}}{\partial D}(N, D, P) &= -\beta B D^{-\beta-1} + \frac{\partial \delta_{PTQ}}{\partial D}(N, D, P) \\
&= -\beta B D^{-\beta-1} + \gamma_D C N^{-\gamma_N} e^{-\frac{P}{\gamma_P}} D^{\gamma_D - 1}
\end{aligned}$$

Dans (8.1), on obtient :

$$\begin{aligned}
\frac{\partial \mathcal{L}}{\partial D}(N, D, P) > 0 &\iff \gamma_D C N^{-\gamma_N} e^{-\frac{P}{\gamma_P}} D^{\gamma_D - 1} > \beta B D^{-\beta-1} \\
&\iff D^{\gamma_D + \beta} > \frac{\beta B N^{\gamma_N}}{C \gamma_D} e^{-\frac{P}{\gamma_P}} \\
&\iff D > \left(\frac{\beta B N^{\gamma_N}}{C \gamma_D} e^{-\frac{P}{\gamma_P}} \right)^{\frac{1}{\gamma_D + \beta}}
\end{aligned}$$

□

Rétrospectivement, sur cette séance, bien que nous touchions au coeur de l'intelligence artificielle générative pour le langage, nous n'avons jamais autant discuté de sujets similaires à ceux de l'IA prédictive *classique* : la bonne collecte des données, sa mise en haute qualité, la bonne exploitation des résultats du modèle et l'accessibilité au plus grand nombre de ces outils. Si c'est vrai pour ces thèmes, c'est également le cas pour les biais inhérents à ces modèles. Cette fois, ils sont exacerbés : la puissance de nuisance est plus forte et pernicieuse, et les biais sont encore plus difficiles à détecter dans cet immense collection de données, nécessaire à la bonne performance métrique des modèles.

Ces effets mériteraient d'avoir une séance voire un module entier dédié. Nous nous contenterons ici de recommander l'article *On the dangers of stochastic parrots : can Language Models be too big ?* [**?**]. Porté par la question *How big is too big ?*, l'article liste l'ensemble des risques associés à ces modèles de langage. Parmi ceux que nous avons traités, nous pouvons citer les problèmes d'inclusion des langues autres que l'anglais et l'ensemble de la culture sous-jacente, la source principale (internet) largement filtrée qui représente à nouveau une sous-population du monde. Également l'impact écologique de ces modèles : entraînement, expériences et inférences à grande échelle est discuté et d'autres initiatives proposent de mesurer le coût environnemental associé à ces modèles. Les impacts des Language Models (LMs) dans le futur sont anticipés : phénomène de boucle où des LMs génèrent le texte sur lesquels d'autres s'entraîneront, propageant ainsi les biais des premiers. Évidemment les usages malveillants malgré un travail sur la sécurité et la toxicité toujours plus important.

Finalement, la fameuse citation d'Alan Turing est plus que jamais d'actualité.

Nous ne pouvons qu'avoir un aperçu du futur, mais cela suffit pour comprendre qu'il y a beaucoup à faire.

— Alan Turing (1950)

Appendices

Annexe A

Rappel et utilisation de la convexité pour le Machine Learning

L'enjeu de cette annexe est de fournir des explications plus mathématiques autour de la convexité afin d'appréhender au mieux de futurs challenges où le cadre du cours sera dépassé et qu'il faudra *tout* refaire. L'annexe sert aussi de support au reste des chapitres présentés en cours pour mieux comprendre les différentes remarques autour des fonctions de pertes et problèmes d'optimisation entre autres. Nous commencerons par une introduction classique à la notion de convexité, puis nous traiterons de différents résultats d'optimisation. Pour finir, nous étudierons l'intérêt de travailler avec des fonctions de pertes convexes pour la descente de gradient.

A.1 Définition et propriétés

A.1.1 Ensemble et fonction convexe

Dans l'ensemble de la section on travaillera toujours en dimension $d \in \mathbb{N}^*$. Commençons par définir ce que l'on appelle un ensemble convexe :

Définition 10. *Soit A un sous-ensemble de $\mathbb{R}^d$. On dit que A est un ensemble convexe si :*

$$\forall a, b \in A, \forall t \in [0,1],\ ta + (1-t)b \in A$$

On appelle donc un ensemble convexe un ensemble dans lequel on peut relier deux points linéairement avec uniquement des points de l'ensemble. On peut illustrer cette définition avec la figure (A.1).

(a) Ensemble convexe (b) Ensemble non convexe

FIGURE A.1 – Exemple et contre exemple d'ensemble convexe

L'ensemble bleu répond parfaitement à l'exigence de la définition : chaque point de l'ensemble est joignable linéairement avec uniquement des points de l'ensemble. L'ensemble rouge n'est pas convexe

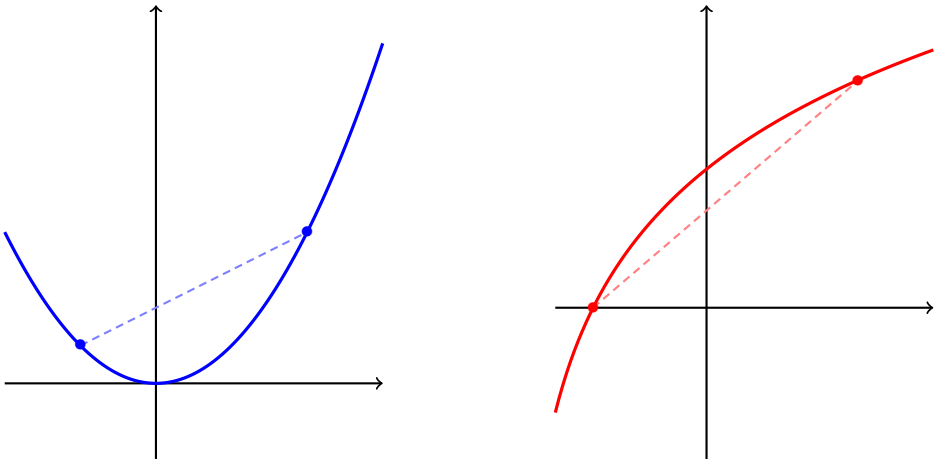
parce qu'entre a et b le *chemin* entre les deux points n'est pas entièrement contenu dans l'ensemble. Avec cette notion, nous sommes capable de définir une fonction convexe :

Définition 11. *Soit A un sous-ensemble convexe de $\mathbb{R}^d$. Une fonction $f : A \mapsto \mathbb{R}$ est convexe si et seulement si :*

$$\forall a, b \in A, \forall t \in [0,1],\ f\left(ta + (1-t)b\right) \leqslant tf(a) + (1-t)f(b)$$

On dira que f est strictement convexe si l'inégalité est stricte.

La définition de fonction convexe ressemble à la définition d'un ensemble convexe. Mais la différence réside dans l'utilisation d'une inégalité ici, et que l'on traite avec les images des points de l'ensemble convexe A. A nouveau, on peut visualiser cette définition avec la figure (A.2).



(a) Fonction convexe (b) Fonction non convexe

FIGURE A.2 – Exemple et contre exemple de fonction convexe

Exercice A.1. *Donner des exemples de fonctions qui répondent aux critères suivants.*

1. *Une fonction strictement convexe.*
2. *Une fonction convexe qui n'est pas strictement convexe.*
3. *Une fonction convexe qui n'est pas strictement convexe ni affine.*

Solution. On propose ici une possibilité, mais il y en a bien sur beaucoup plus.

1. La fonction $x \mapsto x^4$ est strictement convexe.
2. N'importe quelle fonction affine est convexe mais pas strictement convexe.
3. La fonction $x \mapsto \max\{0, x\}$ est convexe mais pas strictement convexe ni affine.

□

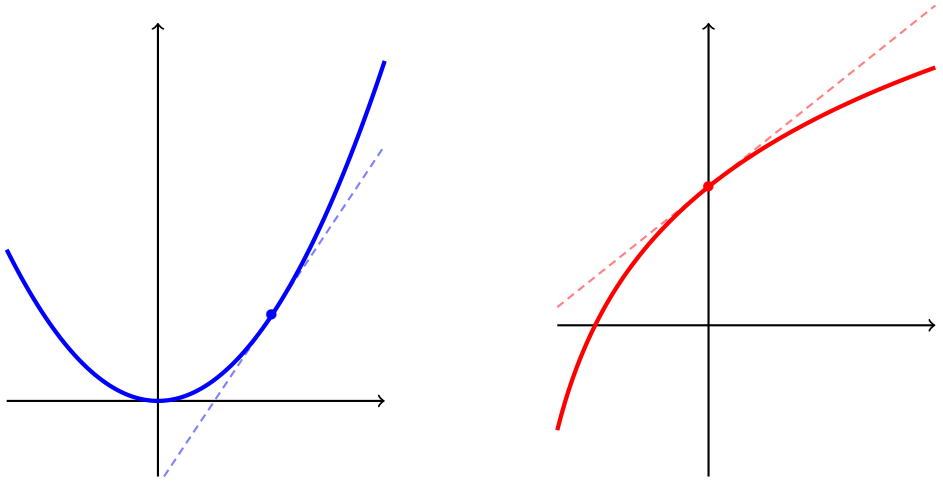
A.1.2 Caractérisation du premier et deuxième ordre

Il peut parfois être difficile de prouver qu'une fonction est convexe avec la définition que l'on vient de donner. Il nous faudrait une caractérisation plus simple d'utilisation :

Théorème 4 (Caractérisation des fonctions convexes)**.** *Soit A un sous-ensemble convexe de $\mathbb{R}^d$ et soit $f : A \mapsto \mathbb{R}$ deux fois différentiable. Alors les propriétés suivantes sont équivalentes :*

1. *f est convexe*
2. $\forall x, y \in A,\ f(y) \geqslant f(x) + \nabla f(x)(y - x)$
3. $\forall x \in A,\ \nabla^2 f(x) \succeq 0$

On sait déjà ce que la première propriété veut dire, il nous reste à comprendre les deux suivantes. Dans la deuxième condition, on reconnait un développement de Taylor à l'ordre 1, ce qui nous dit que chaque tangente de la fonction f est un sous-estimateur global. On peut le visualiser avec deux exemples dans la figure (A.3).



(a) Fonction convexe (b) Fonction non convexe

FIGURE A.3 – Illustration de la propriété de sous-estimateur pour une fonction convexe et contre exemple pour une fonction non convexe

La troisième propriété veut dire qu'il n'y a pas de courbure négative dans la courbe de la fonction f. Autrement dit, que la dérivée de la fonction f est croissante. En dimension une, cela veut dire que la dérivée seconde est toujours positive ou nulle. Il s'agit maintenant de prouver le théorème (4)

Démonstration. Commençons par montrer que si f est convexe, alors $\forall x, y \in A,\ f(y) \geqslant f(x) + \nabla f(x)(y - x)$.

Soit $x, y \in A$, par définition on a que :

$$\begin{aligned} \forall t \in [0,1],\ f(tx + (1-t)y) &\leqslant tf(x) + (1-t)f(y) \\ \forall t \in [0,1],\ f(x + t(y-x)) &\leqslant f(x) + t(f(x) - f(y)) \\ \forall t \in [0,1],\ f(y) - f(x) &\geqslant \frac{f(x + t(y-x)) - f(x)}{t} \end{aligned}$$

Ainsi, en prenant la limite pour $t \downarrow 0$, on a que :

$$\forall x, y \in A,\ f(y) \geqslant f(x) + \nabla f(x)(y - x)$$

D'où le résultat souhaité. Montrons maintenant que si on a le résultat précédent, alors f est convexe.

Soit $x, y \in A$ et on définit $z = tx + (1-t)y$. Par la propriété que l'on suppose, on a :

$$\begin{aligned} f(x) &\geqslant f(z) + \nabla f(z)(x - z) \\ f(y) &\geqslant f(z) + \nabla f(z)(y - z) \end{aligned}$$

En multipliant la première équation par $t \in [0, 1]$ et la seconde équation par $(1-t)$, on obtient :

$$\begin{aligned} tf(x) + (1-t)f(y) &\geqslant f(z) + \nabla f(z)(tx + (1-t)y - z) \\ &= f(z) \\ f(tx + (1-t)y) &\leqslant tf(x) + (1-t)f(y) \end{aligned}$$

On a donc montré que les deux premières propositions sont équivalentes. Montrons maintenant que les deux dernières propriétés sont équivalentes en dimension 1 pour simplifier les calculs.

Supposons que $\forall x \in A,\ f''(x) \geqslant 0$, alors par le théorème de la valeur moyenne de Taylor on a que :

$$\begin{aligned} \exists z \in [x, y], \qquad f(y) &= f(x) + f'(x)(y - x) + \frac{1}{2} f''(z)(y - x)^2 \\ \Rightarrow \quad f(y) &\geqslant f(x) + f'(x)(y - x) \end{aligned}$$

Finalement, supposons que $\forall x, y \in A,\ f(y) \geqslant f(x) + \nabla f(x)(y - x)$. Soit $x, y \in A$ tels que $y > x$. On a que :

$$\begin{aligned} f(y) &\geqslant f(x) + f'(x)(y - x) \\ f(x) &\geqslant f(y) + f'(y)(x - y) \end{aligned}$$

On en déduit donc que :

$$f'(x)(y - x) \leqslant f(y) - f(x) \leqslant f'(y)(y - x)$$

Donc en divisant par $(y - x)^2$ puis en prenant la limite pour $y \to x$, on obtient bien que la propriété souhaité. □

Ce résultat conclut notre section de présentation des notions de convexité. Intéressons-nous maintenant à l'utilisation de cette notion pour l'optimisation.

A.2 Résultats d'optimisations

On considère un problème d'optimisation sans contraintes avec une fonction $f : \mathbb{R}^d \mapsto \mathbb{R}$ différentiable :

$$x^* = \arg\min_{x \in \mathbb{R}^d} f(x) \tag{A.1}$$

Notons qu'ici nous n'avons pas encore spécifié que f est convexe. Dans le cadre général, on sait qu'une condition nécessaire pour que x soit une solution de ce problème est que $\nabla f(x) = 0$. Mais ce n'est pas une condition suffisante ! De plus, si cette condition nécessaire est vraie, et qu'il s'agit d'un minimum, alors on ne peut pas dire plus que "x est un minimum local" avec ces informations.

Exercice A.2. *Donner un exemple de fonction qui répond à chaque critère :*

1. *Une fonction où il existe $x \in \mathbb{R}$ tel que $f'(x) = 0$ et que x est un minimum local.*
2. *Une fonction où il existe $x \in \mathbb{R}$ tel que $f'(x) = 0$ mais que x n'est ni un minimum local ni un maximum local.*
3. *Une fonction où il existe une infinité de $x \in \mathbb{R}$ tel que $f'(x) = 0$ qui sont tous des minimum locaux.*

Solution. On propose ici une possibilité, mais il y en a bien sûr beaucoup plus.

1. La fonction $x \mapsto x^3 - x$ possède un minimum local (et un maximum local).
2. La fonction $x \mapsto x^3$ en $x = 0$.
3. La fonction $x \mapsto \cos(x)$ contient une infinité de point x tel que $f'(x) = 0$ (tous espacés de π) mais seulement *la moitié* sont des minimums.

□

On voit donc que nous n'avons pas de critères simples et clairs dans le cas général sur l'existence et l'unicité d'un minimum global. Voyons ce qu'il en est quand la fonction f est convexe.

Proposition 2. *Soit un problème d'optimisation sans contraintes comme présenté dans (A.1) avec f une fonction convexe et différentiable. Alors, chaque point x qui vérifie $\nabla f(x) = 0$ est un minimum global.*

Pour une fonction convexe différentiable, la condition $\nabla f(x) = 0$ est une condition nécessaire et suffisante pour caractériser un minimum global.

Exercice A.3. *Prouver la proposition (2) à l'aide du théorème (4).*

Solution. Comme $f : A \mapsto \mathbb{R}$ est convexe et différentable, d'après le théorème (4) on a :

$$\forall x, y \in A,\ f(y) \geqslant f(x) + \nabla f(x)(y - x)$$

Donc en particulier pour $\overline{x}$ défini comme $\nabla f(\overline{x}) = 0$:

$$\begin{aligned} \forall y \in A,\ f(y) &\geqslant f(\overline{x}) + \nabla f(\overline{x})(y - \overline{x}) \\ \forall y \in A,\ f(y) &\geqslant f(\overline{x}) \end{aligned}$$

Qui est bien la définition d'un minimum global d'une fonction. □

Mais nous n'avons toujours pas l'unicité d'un minimum à ce stade. Pour l'obtenir, nous avons besoin d'avoir la stricte convexité.

Proposition 3. *Soit un problème d'optimisation sans contraintes comme présenté dans (A.1) avec f une fonction strictement convexe et différentiable. Si $\nabla f(\overline{x}) = 0$, alors $\overline{x}$ est l'unique minimum global de f.*

Nous obtenons cette fois l'unicité à l'aide de la stricte convexité. Voyons comment.

Exercice A.4. *Prouver la proposition (3) en raisonnant par l'absurde. On suppose donc qu'il existe deux minimaux globaux et on aboutit à une absurdité en exploitant la stricte convexité.*

Solution. On suppose qu'il existe $a, b \in A$ qui minimisent f tel quel que $a \neq b$. Prenons $z = \dfrac{a+b}{2} \in A$ comme combinaison convexe de deux points du domaine $\left(\text{avec } t = \dfrac{1}{2}\right)$. Alors :

$$f(z) < \frac{1}{2}f(a) + \frac{1}{2}f(b) = f(a) = f(b)$$

On vient de trouver un nouveau nombre z qui minimise encore mieux la fonction f que les deux meilleurs minimiseurs : absurde, d'où l'unicité. □

Avec ces deux derniers résultats, nous comprenons pourquoi il est important de travailler avec des fonctions de perte convexes : elles nous garantissent qu'en suivant une descente de gradient, nous atteindrons bien un minimum global. Voyons à présent à quelle vitesse.

A.3 Vitesse de convergence pour la descente de gradient

Dans cette section nous allons donner des preuves de vitesse de convergence pour deux hypothèses sur la fonction f.

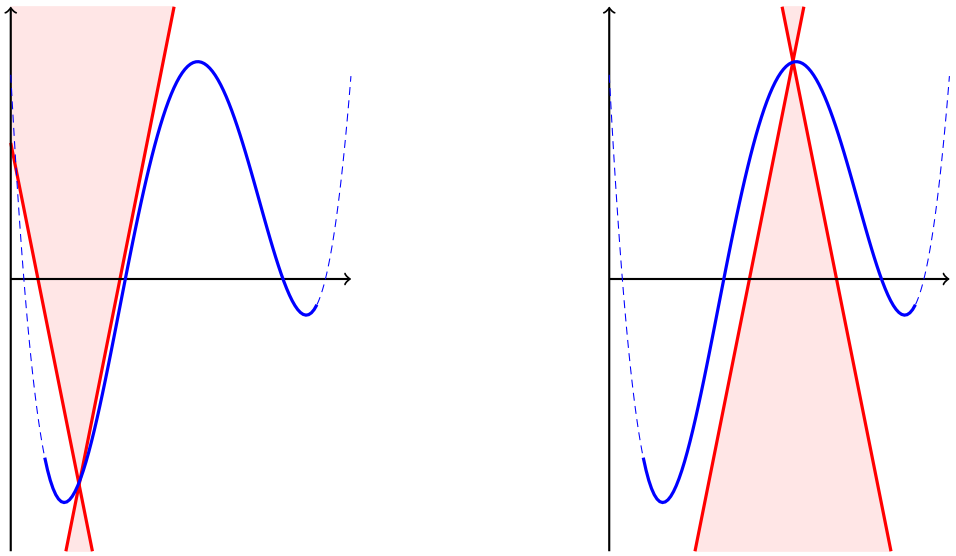
A.3.1 Pour une fonction Lipschitzienne

La plus petite supposition qui nous permet de garantir la convergence vers le minimum global est que la fonction soit Lipschitzienne.

Définition 12 (Fonction L-Lipschitzienne)**.** *Soit $f : \mathcal{D} \to \mathbb{R}$ une fonction convexe. On dit que f est L-Lipschitzienne si il existe un nombre L tel que :*

$$|f(y) - f(x)| \leqslant L\|y - x\|$$

Pour comprendre visuellement cette définition, on peut s'appuyer sur la figure (A.4).



(a) Condition en $x = 1$ (b) Condition en $x = 2.7$

FIGURE A.4 – Illustration de la condition lipschitzienne pour une fonction

Dire qu'une fonction est L-lipschitzienne revient à dire que ses variations seront toujours hors des cônes rouge, autrement dit on contraint la progression de la fonction. Ici, on remarque que la fonction n'est pas lipschitzienne pour ce L-là sur $[0, 5]$ parce que la fonction passe dans les cônes rouge. En revanche,

elle l'est pour l'intervalle $[0.5, 4.5]$. Remarquons également que nous avons une fonction qui peut être lipschitzienne sans être convexe. Supposer les deux, permet d'avoir le résultat suivant.

Proposition 4. *Soit $x^* \in \mathbb{R}^d$ le minimum de la fonction $f : \mathbb{R}^d \mapsto \mathbb{R}$ convexe et L-Lipschitzienne. Soit $\varepsilon > 0$ l'erreur que l'on accorde pour arrêter la descente de gradient. On choisit $\eta = \frac{\varepsilon}{L^2}$. Alors en $T = \frac{L^2\|x^* - x_0\|^2}{\varepsilon^2}$ itérations, l'algorithme converge vers x^* avec une erreur de ε.*

Nous avons donc la garantie que nous sommes capables de converger vers le minimum de la fonction f avec un nombre d'itérations que l'on maitrise. Si l'on note $\widetilde{x} \in \mathbb{R}^d$ le point que l'on obtient à la suite de la descente de gradient, on a $f(\widetilde{x}) \leqslant f(x^*) + \varepsilon$.

Pour faire la preuve de ce résultat, nous avons besoin d'un résultat intermédiaire.

Lemme 2. *Pour $(x_t)_{t\in\mathbb{N}}$ la suite définie pour une descente de gradient qui cherche à minimiser une fonction f convexe et L-Lipschitzienne, on a :*

$$\|x_{t+1} - x^*\|^2 \leqslant \|x_t - x^*\|^2 - 2\eta\left(f(x_t) - f(x^*)\right) + \eta^2 L^2$$

Exercice A.5. *Prouver le lemme (2).*

Solution.

$$\begin{aligned}
\|x_{t+1} - x^*\|^2 &= \|x_t - \eta\nabla f(x_t) - x^*\|^2 \\
&= \|x_t - x^*\|^2 - 2\eta\left(\nabla f(x_t)\right)^t (x_t - x^*) + \eta^2\|\nabla(f(x_t))\|^2 \\
&\leqslant \|x_t - x^*\|^2 - 2\eta\left(f(x_t) - f(x^*)\right) + \eta^2 L^2
\end{aligned}$$

En exploitant le fait que f soit convexe et L-Lipschizienne pour la dernière inégalité. □

Nous avons donc maintenant tous les outils pour démontrer le résultat annoncé dans la proposition (4).

Démonstration. Soit $\Phi(t) = \|x_t - x^*\|^2$. Alors avec $\eta = \frac{\varepsilon}{L^2}$ et le précédent lemme on a :

$$\Phi(t) - \Phi(t+1) > 2\eta\varepsilon - \eta^2 L^2 = \frac{\varepsilon^2}{L^2}$$

Puisque par définition, $\Phi(0) = \|x^* - x_0\|$ et $\Phi(t) \geqslant 0$, la précédente inéquation induit que :

$$\begin{aligned}
\Phi(t) \leqslant \Phi(t-1) - \frac{\varepsilon^2}{L^2} &\Leftrightarrow \Phi(t) \leqslant \Phi(0) - t\frac{\varepsilon^2}{L^2} \\
&\Leftrightarrow 0 \leqslant \Phi(0) - t\frac{\varepsilon^2}{L^2} \\
&\Leftrightarrow t \leqslant \Phi(0)\frac{L^2}{\varepsilon^2}
\end{aligned}$$

□

Nous avons tout de même une convergence assez lente puis qu'elle est de l'ordre de $\frac{1}{\varepsilon^2}$, donc pour une erreur de 0.1, il faut 100 itérations, et c'est une erreur très large dans beaucoup d'applications.

A.3.2 Fonction β-smooth

Une manière d'accélérer cette convergence est de faire plus d'hypothèse sur la fonction en demandant qu'elle soit β-smooth.

Définition 13 (Fonction β-smooth)**.** *Soit $f : \mathcal{D} \mapsto \mathbb{R}$ une fonction convexe différentiable. On dit que f est β-smooth si :*

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

Avec l'exercice suivant, on peut découvrir une interprétation de cette nouvelle notion.

Exercice A.6 (Caractérisation d'une fonction β-smooth)**.** *Montrer que f est une fonction β-smooth si, et seulement si, le gradient de f est β-Lipschitz.*

Avec cette condition supplémentaire, nous sommes en capacité d'avoir une convergence plus rapide.

Proposition 5. *Soit $x^* \in \mathbb{R}^d$ le minimum de la fonction $f : \mathbb{R}^d \mapsto \mathbb{R}$ convexe et β-smooth. Soit $\varepsilon > 0$ l'erreur que l'on accorde pour arrêter la descente de gradient. On choisit $\eta = \frac{1}{\beta}$.*
Alors en $T = \frac{2\beta\|x^* - x_0\|^2}{\varepsilon}$ itérations, l'algorithme converge vers x^* avec une erreur de ε.

La vitesse de convergence annoncée est beaucoup plus grande ! Dans le cas précédent, pour $\varepsilon = 0.1$ nous avions besoin de 100 itérations (en supposant que le numérateur soit 1) alors qu'ici 10 itérations suffisent.

Nous ne prouverons pas ici le résultat et renvoyons vers le très complet article de Sébastien Bubeck *Convex optimization : Algorithms and complexity* publié en 2015, pour avoir le détail de la preuve. Dans ce même article, nous pouvons trouver d'autres hypothèses, comme par exemple que f soit fortement convexe, pour accélérer encore la convergence.

Connaître ces propriétés permet parfois de choisir une fonction plutôt qu'une autre quand elles remplissent le même rôle dans l'entraînement d'un modèle. Avoir une vitesse de convergence plus rapide parce que nous avons choisi une fonction à optimiser la plus régulière possible permet de faire gagner parfois des jours voire semaines de calculs. C'est ce qui explique le très grand nombre de différentes méthodes de descente de gradient qui ont été développées.

Annexe B

Algorithme du Perceptron

Pour atteindre une compréhension pleine de l'atome, il a fallu que de nombreux modèles imparfaits soient proposés. L'un des modèles les plus connus est celui de Niels Bohr [Boh13]. Il est célèbre pour sa tentative d'explication du mouvement des électrons autour du noyau avec les prémices de la théorie quantique. Ce modèle est aujourd'hui complètement dépassé mais reste enseigné parce qu'il représente un moment d'histoire décisif. Nous avons pu nous inspirer de ses idées et ses imperfections pour avancer dans la bonne direction.

Cette annexe présente un monument d'histoire de l'intelligence artificielle au sens où nous l'entendons aujourd'hui, qui est largement dépassé mais dont les défauts et réussites ont inspiré l'ensemble des développements qui ont été présentés dans ce cours.

B.1 Description

[MP69] décrit l'algorithme du perceptron. Il s'agit d'un algorithme de classification qui va séparer linéairement deux groupes. Naturellement, on peut étendre cet algorithme à une classification multi-classe.

On considère le problème de classification avec $\mathcal{Y} = \{-1, 1\}$ et on définit la fonction signe sgn par :

$$\text{sgn}(x) = \begin{cases} +1 & \text{si } x > 0 \\ -1 & \text{sinon} \end{cases}$$

Pour $u, v \in \mathbb{R}^d$, on note le produit scalaire $\langle u, v \rangle = \sum_{i=1}^{d} u_i v_i$. De manière classique, dans l'algorithme du perceptron on utilise plutôt la notation w que la notation θ pour le paramètre de la forme de fonction que l'on cherche.
On rappelle que l'ensemble des $x \in \mathbb{R}^d$ tel que $\langle w, x \rangle = 0$ forme un hyperplan de vecteur normal w.

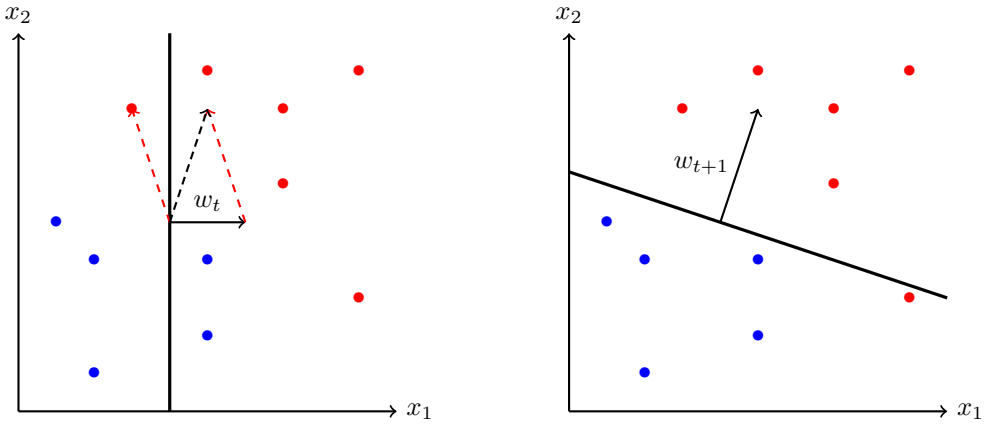
Le problème d'apprentissage du papier originel de 1969 est, avec les conventions que l'on a choisi :

$$\begin{aligned} f_w(x) &= \text{sgn}(\langle w, x \rangle) \\ w^* &= \arg\min_{w \in \mathbb{R}^d} \sum_{i=1}^{n} \max\{0, -y_i \langle w, x_i \rangle\} \end{aligned}$$

Pour des notations plus compactes, de la même manière que pour la régression linéaire, on considère que le paramètre w_1 est associé à une information qui vaut systématiquement 1 (cela permet de représenter le biais). Ce problème est résolu en regardant les observations une par une et en appliquant la règle suivante :

$$w_{t+1} = w_t + yx \mathbb{1}_{y\langle w_t, x \rangle \leqslant 0}$$

On dit donc que l'on modifie le vecteur paramètre uniquement lorsqu'il y a une erreur, et on le corrige proportionnellement à l'observation x. Visuellement, on peut comprendre le fonctionnement du perceptron à travers une étape :



(a) Etape t (b) Etape $t+1$

FIGURE B.1 – Intuition géométrique du perceptron (-1)

On voit qu'à l'étape t un point rouge est classifié comme négatif, alors qu'il devrait être positif (du côté pointé par w_t). En appliquant la règle de mise à jour des poids, on obtient w_{t+1} à l'étape suivante. Pour encore mieux saisir le perceptron, rien de mieux qu'un exercice :

Exercice B.1 (Programmation d'un perceptron)**.** *On suppose que l'on dispose d'un dataset $\mathcal{D}$.*

1. *Ecrire une fonction* update_weight *qui prend en paramètre une observation x et sa classe y et met à jour la valeur du vecteur de poids w.*
2. *Ecrire une fonction* perceptron *qui prend en paramètre une matrice d'observation X, son vecteur de classe associé y et un paramètre* epoch *qui représente le nombre de fois où chaque d'observation sera vue par l'algorithme.*
3. *Ajouter un mécanisme de vérification de convergence pour arrêter l'apprentissage plus tôt, et donc éviter de faire 50 époques quand 10 suffisent.*

L'un des grands intérêts du perceptron est qu'il s'agissait d'un des premiers algorithmes *apprenant* qui avait une preuve de convergence.

Théorème 5 (Convergence du perceptron)**.** *On suppose que les données issues d'un dataset $\mathcal{D}$ soient linéairement séparables et de plus que :*

$$\exists \gamma > 0, \forall i \leqslant n,\ y_i \langle w^*, x_i \rangle \leqslant \gamma$$

On note $R = \max_{1 \leqslant i \leqslant n} \|x_i\|$*. Alors la k-ième erreur de classification du perceptron aura lieu avant :*

$$k \leqslant \left(\frac{R}{\gamma}\right)^2 \|w^*\|^2$$

Ce que la condition du paramètre γ veut dire est que pour le vecteur de paramètres w optimal, les données sont parfaitement séparées et avec une marge d'au moins γ. Il nous reste à démontrer ce résultat historique.

B.2 Preuve de convergence

Pour le prouver, nous allons commencer par rappeler la fameuse inégalité de Cauchy-Schwartz :

Proposition 6 (Cauchy-Schwarz). *Soient $u, v \in \mathbb{R}^n$ et $\|\cdot\| : \mathbb{R}^n \to \mathbb{R}^+$ une norme pour $\mathbb{R}^n$. Alors :*

$$\langle u, v\rangle \leqslant \|u\|\|v\|$$

Démonstration. On considère le polynôme $P : \mathbb{R} \to \mathbb{R}$ défini par

$$\begin{aligned} P(\lambda) &= \|u + \lambda v\|^2 \\ &= \|u\|^2 + 2\lambda\langle u, v\rangle + \lambda^2\|v\|^2 \end{aligned}$$

Par définition, on sait que $\forall\lambda \in \mathbb{R}, P(\lambda) \geqslant 0$. Autrement dit, son discriminant Δ est négatif ou nul. Alors on a que :

$$4\langle u, v\rangle - 4\|u\|^2\|u\|^2 \leqslant 0 \iff \langle u, v\rangle \leqslant \|u\|\|v\|$$

□

On remarque au passage que le cas d'égalité apparaît quand $\langle u, v\rangle = \|u\|\|v\|$ autrement dit quand ils sont colinéaires de même sens.

Nous avons maintenant l'ensemble des outils pour aboutir à la démonstration du théorème (5).

Démonstration. Puisque le vecteur des paramètres w n'est mis à jour que lors d'une erreur de prédiction, on note w_k le vecteur lorsque la k-ième erreur est commise. Alors :

$$w_k = w_{k-1} + y_i x_i$$

On commence par remarquer que :

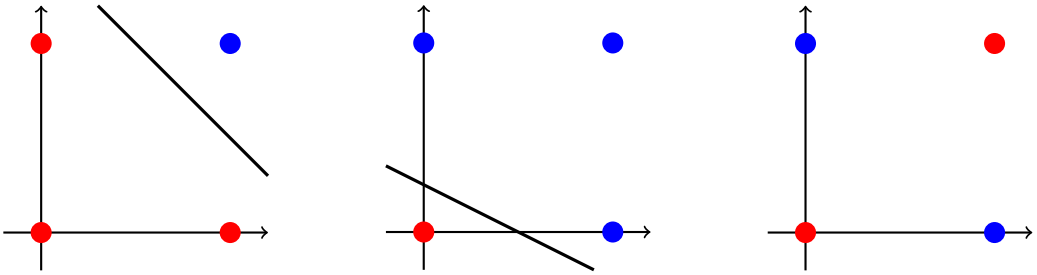
$$\begin{aligned} \langle w_k, w^*\rangle &= \langle w_{k-1} + y_i x_i, w^*\rangle \\ &= \langle w_{k-1}, w^*\rangle + y_i\langle x_i, w^*\rangle \\ &\geqslant \langle w_{k-1}, w^*\rangle + \gamma \end{aligned}$$

Donc par une récurrence immédiate, on a :

$$\langle w_k, w^*\rangle \geqslant k\gamma$$

Puisqu'on veut borner le nombre d'erreurs , il faut borner l'apparition de k, et donc majorer le terme $\langle w_k, w^*\rangle$. On peut regarder l'inégalité de Cauchy-Schwarz quand on cherche à majorer de telles quantités. Pour le faire, on doit avoir une idée de la norme de w_k :

$$\begin{aligned} \|w_k\|^2 &= \|w_{k-1}\|^2 + 2y_i\langle w_k, x_i\rangle + y_i^2\|x_i\|^2 \\ &\leqslant \|w_{k-1}\|^2 - 2\gamma + R^2 \\ &\leqslant kR^2 \end{aligned}$$



(a) Opérateur AND (b) Opérateur OR (c) Opérateur XOR

FIGURE B.2 – Représentation de quelques opérateurs logique

Ainsi, en combinant les résultats que l'on a obtenus, on arrive à :

$$\begin{aligned} k^2\gamma^2 &\leqslant \langle w_k, w^* \rangle^2 \\ &\leqslant kR^2\|w^*\|^2 \\ k &\leqslant \left(\frac{R}{\gamma}\right)^2 \|w^*\|^2 \end{aligned}$$

□

B.3 Problème XOR

Dans les premiers travaux autours du perceptron, nous nous sommes questionné sur la possibilité de reproduire les fonctions logiques NOT, AND, OR avec un réseau de neurones. Pour l'opérateur de négation, $w = -1$ et $b = 0.5$ suffisent pour renvoyer les valeurs attendues.

Dans le cas de XOR, on ne peut pas trouver un couple (w, b) qui permettent à un perceptron de séparer les classes correctement. Pourtant, nous sommes capables de représenter les opérateurs logiques *de base*.

Exercice B.2 (Réécriture de XOR)**.** *Montrer que pour deux booléens A et B, on a :*

$$XOR(A, B) = AND(NOT(AND(A, B)), OR(A, B))$$

Expliquer comment représenter l'opérateur XOR.

Solution. Si l'on dresse la table de vérité, on a :

A	B	NOT(AND(A, B))	OR(A, B)	AND(NOT(AND(A, B)), OR(A, B))	XOR(A, B)
1	0	1	1	1	1
1	1	0	1	0	0
0	0	1	0	0	0
0	1	1	1	1	1

Il faudrait donc avoir plusieurs perceptrons à la suite pour pouvoir représenter l'opérateur XOR : un perceptron ne suffit plus, il faut les agencer en réseaux ! □

Ce problème a été mal interprété au moment de la publication du livre [MP69] et a donné lieu, malgré lui, au premier hiver des réseaux de neurones.

B.4 Bonus : utilisation de l'inégalité de Cauchy-Schwarz

Dans la section (3.3.1) est présentée la notion de F_1-score, qui correspond à la moyenne harmonique entre la précision et le recall. Il est également affirmé qu'une moyenne harmonique est plus conservative car toujours inférieure ou égale à la moyenne arithmétique. Il reste à le prouver.

Exercice B.3 (Moyenne harmonique)**.** *Soit $(x_k)_{k\leqslant n}$ des réels strictement positifs. On définit :*

- *Moyenne arithmétique :* $A_n = \dfrac{1}{n}\displaystyle\sum_{k=1}^{n} x_k$
- *Moyenne harmonique :* $H_n = \dfrac{n}{\displaystyle\sum_{k=1}^{n}\frac{1}{x_k}} \iff \dfrac{1}{H_n} = \dfrac{1}{n}\displaystyle\sum_{k=1}^{n}\frac{1}{x_k}$

Montrer que :

$$H_n \leqslant A_n$$

Solution. Puisque $(x_k)_{k\leqslant n}$, on sait qu'il existe une unique suite $(y_k)_{k\leqslant n}$ telle que $\forall k \leqslant n, x_k = y_k^2$. Ainsi, on a :

$$\begin{aligned}
\frac{A_n}{H_n} &= \left(\frac{1}{n}\sum_{k=1}^{n} y_k^2\right)\left(\frac{1}{n}\sum_{k=1}^{n}\frac{1}{y_k^2}\right) \\
&\leqslant \frac{1}{n^2}\sum_{k=1}^{n}\frac{y_k^2}{y_k^2} \quad \text{avec l'inégalité de Cauchy-Schwartz} \\
&\leqslant 1 \\
H_n &\leqslant A_n
\end{aligned}$$

□

Annexe C

Algorithme Naive Bayes

Le Machine Learning est un domaine au croisement de l'analyse, l'algèbre et des statistiques comme nous avons pu le constater au travers des chapitres précédents. Nous avons exploité largement plusieurs formalismes et idées qui n'ont pas explicitement été développée pour cet usage.

D'autres domaines des mathématiques plus spécifiques encore sont exploités actuellement, on pense par exemple à la théorie des catégories pour l'algorithme UMAP. Mais nous n'avons jusque-là jamais exploité le formalisme et les idées des probabilités.

C'est un choix qui a été fait pour rendre le cours le plus compréhensible possible. La force des mathématiques est d'être capable de faire des liens entre les différents domaines qui composent cette discipline, mais ce n'est pas une chose simple et cela peut être largement perturbant pour quiconque n'y est pas initié. Par soucis de compréhension, nous avons donc fait le choix de proposer la présentation des algorithmes sous le prisme de l'analyse et de l'algèbre. Nous proposons ici un algorithme qui exploite et nécessite une vision probabiliste.

C.1 Probabilité conditionnelle et théorème de Bayes

Considérons un lancé de dé à 6 faces équilibrés. Nous nous intéressons à deux événements :

- A : le nombre est 2
- B : le nombre est pair

Nous sommes parfaitement capables de calculer $\mathbb{P}(A) = \frac{1}{6}$ et $\mathbb{P}(B) = \frac{1}{2}$. Mais quelle est la probabilité que le nombre soit 2 **sachant** que le nombre est pair ? Autrement dit quelle est la probabilité de l'événement A sachant que l'événement B s'est réalisé ?
Elle vaut évidemment $\frac{1}{3}$ puisque parmi les 3 possibilités de nombre pair, il n'y en a qu'une qui réalise l'événement A. Mais comment formaliser cela ?

Définition 14 (Probabilité conditionnelle)**.** *Soient A et B deux événements, et $\mathbb{P}(B) \neq 0$. La probabilité conditionnelle de A sachant B est définie par :*

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

La notation $A \mid B$ se lit *"A sachant B"*, et on remarque que l'on obtient bien la bonne probabilité dans l'exemple introductif.

Exercice C.1 (Propriétés). *Soient A_1, A_2 et B trois événements, et $\mathbb{P}(B) \neq 0$. Montrer que :*

1. $\mathbb{P}(A_1 \cup A_2 \mid B) = \mathbb{P}(A_1 \mid B) + \mathbb{P}(A_2 \mid B) - \mathbb{P}(A_1 \cap A_2 \mid B)$
2. $\mathbb{P}\left(\overline{A_1} \mid B\right) = 1 - \mathbb{P}(A_1 \mid B)$

Solution. Soient A_1, A_2 et B trois événements, et $\mathbb{P}(B) \neq 0$.

1. Par définition d'une probabilité conditionnelle, et par les axiomes de probabilité :

$$\begin{aligned}
\mathbb{P}(A_1 \cup A_2 \mid B) &= \frac{\mathbb{P}((A_1 \cup A_2) \cap B)}{\mathbb{P}(B)} \\
&= \frac{\mathbb{P}((A_1 \cap B) \cup (A_2 \cap B))}{\mathbb{P}(B)} \\
&= \frac{\mathbb{P}(A_1 \cap B) + \mathbb{P}(A_2 \cap B) - \mathbb{P}(A_1 \cap A_2 \cap B)}{\mathbb{P}(B)} \\
&= \mathbb{P}(A_1 \mid B) + \mathbb{P}(A_2 \mid B) - \mathbb{P}(A_1 \cap A_2 \mid B)
\end{aligned}$$

2. Par définition d'une probabilité conditionnelle :

$$\begin{aligned}
\mathbb{P}\left(\overline{A_1} \mid B\right) + \mathbb{P}(A_1 \mid B) &= \frac{\mathbb{P}\left(\overline{A_1} \cap B\right) + \mathbb{P}(A_1 \cap B)}{\mathbb{P}(B)} \\
&= 1
\end{aligned}$$

□

Exercice C.2. *Soient A, B et C trois événements de probabilité non nulle. Montrer que :*

1. $\mathbb{P}(A \cap B) = \mathbb{P}(A \mid B)\,\mathbb{P}(B)$
2. $\mathbb{P}(A \cap B \cap C) = \mathbb{P}(A \mid B \cap C)\,\mathbb{P}(B \mid C)\,\mathbb{P}(C)$
3. *En exploitant la première égalité, montrer que* $\mathbb{P}(A \mid B) = \dfrac{\mathbb{P}(A)\,\mathbb{P}(B \mid A)}{\mathbb{P}(B)}$

Solution. Soient A, B et C trois événements de probabilité non nulle.

1. Le résultat est immédiat avec la définition d'une probabilité conditionnelle.
2. Avec la définition d'une probabilité conditionnelle et la question précédente, on a :

$$\begin{aligned}
\mathbb{P}(A \cap B \cap C) &= \mathbb{P}(A \mid B \cap C)\,\mathbb{P}(B \cap C) \\
&= \mathbb{P}(A \mid B \cap C)\,\mathbb{P}(B \mid C)\,\mathbb{P}(C)
\end{aligned}$$

   On y voit une forme de factorisation d'une intersection via des probabilités conditionnelles.

3. Avec la définition d'une probabilité conditionnelle et la première question, on a :

$$\begin{aligned}
\mathbb{P}(A \mid B) &= \frac{\mathbb{P}(A \cap B)}{\mathbb{P}(B)} \\
&= \frac{\mathbb{P}(B \cap A)}{\mathbb{P}(B)} \\
&= \frac{\mathbb{P}(A)\,\mathbb{P}(B \mid A)}{\mathbb{P}(B)}
\end{aligned}$$

□

La dernière égalité est particulièrement intéressante : nous sommes capables de calculer la probabilité de A sachant B si l'on connait la probabilité de B sachant A (et d'autres choses). Si nous pensons en terme temporel, nous venons presque d'inverser le temps !

On rappelle qu'en probabilité l'univers Ω est l'ensemble des issues possibles. On peut définir $(A_i)_i \leqslant n$ une partition de l'univers si chacun des événements A_i sont exclusifs et que l'union des A_i forme exactement Ω. Nous avons maintenant l'ensemble des informations pour énoncer le théorème qui permet de définir l'algorithme Naive Bayes.

Théorème 6 (Bayes, 1763)**.** *Soit $(A_i)_i \leqslant n$ une partition de l'univers Ω. Soit B un événement de probabilité non nulle. On a :*

$$\mathbb{P}(A_i \mid B) = \frac{\mathbb{P}(B \mid A_i)\,\mathbb{P}(A_i)}{\sum_{j=1}^{n} \mathbb{P}(B \mid A_j)\,\mathbb{P}(A_j)}$$

Démonstration. Par récurrence immédiate, nous avons montré dans un exercice que l'initialisation était vraie. □

C.2 Application en Machine Learning

Considérons un dataset de classification avec K classes :

$$\mathcal{D} = \left\{ (x^{(i)}, y_i) \;\mid\; \forall i \leqslant n,\ x^{(i)} \in \mathbb{R}^d, y_i \in \mathcal{Y} \right\}$$

$\mathcal{Y}$: Ensemble de dimension finie K

Pour simplifier les notations, nous ferons la confusion entre les variables aléatoires $(x_j)_{j \leqslant d}$ et les réalisations $x^{(i)} = (x_k)_{k \leqslant d}^{(i)}$ qui forment le dataset $\mathcal{D}$. De même, nous ferons la confusion entre la variable aléatoire y et les réalisations y_i qui forment le dataset $\mathcal{D}$.

Nous aimerions être capables d'estimer la probabilité : $\mathbb{P}\left(y = k \mid \bigcap_{j=1}^{d} x_j\right)$ c'est à dire la probabilité que l'observation considérée soit de la classe $k \leqslant K$. Nous pouvons le faire, mais il va falloir construire un tableau énorme où l'on va répertorier la probabilité pour chacune des possibilités de $\bigcap_{j=1}^{d} x_j$. Il va falloir également avoir une quantité astronomique d'observations pour avoir une bonne estimation de chacune des cellules. Ce n'est pas possible en pratique. Exploitons la section précédente.

Par le théorème de Bayes, on a :

$$\mathbb{P}\left(\{y = k\} \mid \bigcap_{j=1}^{d} x_j\right) = \frac{\mathbb{P}(\{y = k\}) \times \mathbb{P}\left(\bigcap_{j=1}^{d} x_j \mid \{y = k\}\right)}{\mathbb{P}\left(\bigcap_{j=1}^{d} x_j\right)} \tag{C.1}$$

Chacun des termes a un nom qui a du sens dans le domaine des probabilités bayésiennes. On remarque que le dénominateur est constant si l'on connait l'ensemble des probabilités associées aux informations, ainsi seulement le numérateur nous intéresse. Nous avons accès à $\mathbb{P}(\{y = k\})$, il nous reste à bien estimer la vraisemblance.

En exploitant la généralisation d'une égalité que nous avons montré dans un précédent exercice, on peut montrer que :

$$\mathbb{P}\left(\bigcap_{j=1}^{d} x_j \mid \{y = k\}\right) = \prod_{j=1}^{d} \mathbb{P}\left(x_j \mid \{y = k\} \cap \bigcap_{i=1}^{j-1} x_i\right)$$

Nous ne semblons pas plus avancés à ce stade. Nous allons donc faire l'hypothèse **naïve** qui donne le nom de cet algorithme : les $(x_j)_{j \leqslant d}$ sont indépendants entre eux conditionnellement à l'événement $\{y = k\}$. Si l'on traduit formellement :

$$\forall i, j \leqslant d,\ i \neq j \Rightarrow \mathbb{P}(x_i \mid \{y = k\} \cap x_j) = \mathbb{P}(x_i \mid \{y = k\}) \qquad \text{(Hypothèse naïve)}$$

Ainsi, nous pouvons réécrire l'équation (C.1) comme :

$$\mathbb{P}\left(\{y = k\} \mid \bigcap_{j=1}^{d} x_j\right) = \frac{\mathbb{P}(\{y = k\}) \times \prod_{j=1}^{d} \mathbb{P}(x_j \mid \{y = k\})}{\mathbb{P}\left(\bigcap_{j=1}^{d} x_j\right)} \tag{C.2}$$

Il ne reste plus qu'à estimer maintenant chacune des valeurs $\mathbb{P}(x_j \mid \{y = k\})$. On le fait avec un dataset en distinguant les cas :

- S'il s'agit d'une variable continue : on applique une loi gaussienne par exemple, donc on va apprendre deux paramètres (moyenne et écart-type)
- S'il s'agit d'une variable discrète : on applique une loi binomiale par exemple

Ceci conclut la présentation succincte de l'algorithme Naïve Bayes. Comme expliqué dans l'introduction, nous aurions pu proposer d'autres visions pour présenter quelques algorithmes. Ceci est notamment vrai pour les algorithmes de régression linéaire et logistique que l'on peut unir formellement via l'exploitation de la famille exponentielle par exemple. Nous avons touché du doigt ce lien en observant par exemple que les équations de descente de gradient sont largement similaires, mais cela va bien au-delà.
Si le lecteur a une fibre probabiliste, nous l'encourageons à redécouvrir le Machine Learning par ce prisme : multiplier les angles de vue sur un même sujet permet de l'embrasser complètement.

Vraisemblance

Antérieur

Postérieur

Evidence

Annexe D

Fléau de la dimension

Georg Cantor est un mathématicien allemand du 19e siècle qui est particulièrement connu pour son travail sur la théorie des ensembles et plus spécifiquement sur ses résultats concernant l'infini. L'ensemble des nombres entiers est infini par construction, mais l'ensemble des nombres entiers relatifs également. Et intuitivement, nous nous disons que ces infinis ne sont pas vraiment les mêmes, puisqu'il semblerait que $\mathbb{Z}$ soit plus grand que $\mathbb{N}$! Georg Cantor montre que ces deux infinis sont en fait les mêmes : il y a autant de nombres dans $\mathbb{Z}$ que dans $\mathbb{N}$. Plus fort encore, il démontre la puissance de l'infini qui nous donne un résultat encore plus contre intuitif : il y a autant de nombres dans l'intervalle $[0, 1]$ que dans $\mathbb{R}$ tout entier ! Alors qu'il venait de le démontrer, il a envoyé une lettre à son ami mathématicien Dedekind :

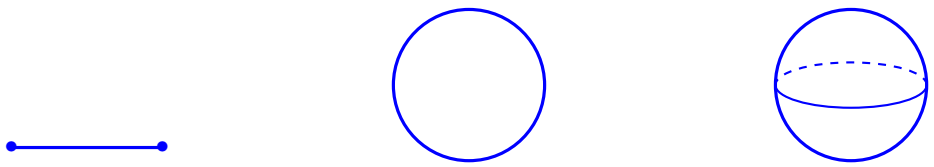
> *Tant que vous ne m'aurez pas approuvé, je ne puis que dire : je le vois mais je ne le crois pas.*
> — Georg Cantor (1877)

Il a prouvé quelque chose que l'ensemble de la communauté pensait intuitivement fausse, et lui-même n'y croyait pas. Nous sommes toujours mis en difficulté quand il s'agit de traiter avec l'infini, ou des grandes quantités. Cette remarque nous amène donc à nous questionner sur l'impact d'un grand nombre d'informations quand nous entraînons un modèle de Machine Learning.

Le **fléau de la dimension** est une notion connue en statistiques et en Machine Learning. Ce terme rassemble tout un ensemble de phénomènes qui se produit en très grande dimension, mais pas dans une dimension plus petite. Nous proposons dans cette annexe d'illustrer quelques-uns des phénomènes étranges de la grande dimension et ses impacts en Machine Learning.

D.1 Volume d'une hypersphère

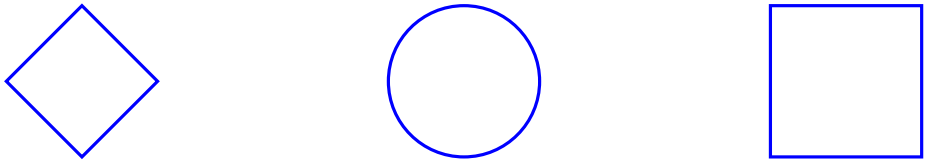
Pour essayer de sentir les problèmes de la très grande dimension, on s'intéresse au volume d'une hypersphère.



(a) En dimension 1, volume = 2 (b) En dimension 2, volume = π (c) En dimension 3, volume = $\frac{4}{3}\pi$

FIGURE D.1 – Représentation et volume d'une hypersphère de rayon 1 dans 3 espaces de dimensions différentes

Avec la figure (D.1) nous avons l'intuition que le volume augmente avec la dimension. Donc pour une hypersphère de très grande dimension, on devrait avoir un très grand volume.
Nous avons tracé et mesuré le volume pour la distance euclidienne classique, mais nous pouvons aussi utiliser d'autres distances (autre norme) comme montré dans la figure (D.2).



(a) Avec la norme 1 (b) Avec la norme 2 (c) Avec la norme infinie

FIGURE D.2 – Représentation d'une hypersphère de rayon 1 en dimension 2 pour 3 normes différentes

Formalisons le problème et généralisons-le pour calculer le volume d'une hypersphère en n'importe quelle dimension et pour n'importe quelle norme. Soit $n \in \mathbb{N}^*$ la dimension de l'espace, on appelle *boule* ou hypersphère l'objet défini par :

$$\begin{aligned} B_n^p(R) &= \{(x_1, \dots, x_n) \in \mathbb{R}^n, \sum_{i=1}^n x_i^p \leqslant R^p\} \\ &= \{u \in \mathbb{R}^n, \|u\|_p^p \leqslant R^p\} \end{aligned}$$

Avec $\|u\|_p$ la norme p définie comme $\|u\|_p^p = \sum_{i=1}^n x_i^p$. $B_n^p(R)$ est la boule de dimension n avec une p-norme de rayon R. On définit $V_n^p(R)$ le volume de la boule $B_n^p(R)$ *i.e.* la mesure de $B_n^p(R)$ pour la mesure de Lesbegue dans $\mathbb{R}^n$. Formellement :

$$V_n^p(R) = \int_{B_n^p(R)} \bigotimes_{i=1}^n \mathrm{d}x_i$$

Proposition 7 (Volume d'une hypersphere)**.** *Avec les notations précédentes, on a :*

$$\forall R > 0, \forall n \geqslant 2, \forall p \geqslant 1, \quad V_n^p(R) = \frac{\left(2R\Gamma\left(\frac{1}{p}+1\right)\right)^n}{\Gamma\left(\frac{n}{p}+1\right)}$$

Et son équivalent quand n tend vers l'infini :

$$V_n^p(R) \sim \sqrt{\frac{p}{2\pi n}} \left[2R\Gamma\left(\frac{1}{p}+1\right)\left(\frac{pe}{n}\right)^{\frac{1}{p}}\right]^n$$

Avec la fonction Γ définie comme :

$$\Gamma(x) = \int_0^{+\infty} e^{-t} t^{x-1} \mathrm{d}t$$

La preuve de ce résultat est assez technique, nous ne la présenterons pas ici [1]. Ce que ce résultat exhibe, c'est que le volume d'une hypersphère en grande dimension tend exponentiellement vite vers 0, c'est complètement contre intuitif ! Visualisons les courbes de cette fonction avec la figure (**??**).

On retrouve bien le comportement en hausse que nous avions observé, mais on comprend que le comportement ultime est que le volume tende vers 0 très rapidement. Avant de discuter de ce que ce résultat implique, regardons un autre résultat contre intuitif.

1. Si elle est nécessaire, il suffit de m'envoyer un message pour obtenir la démonstration complète.

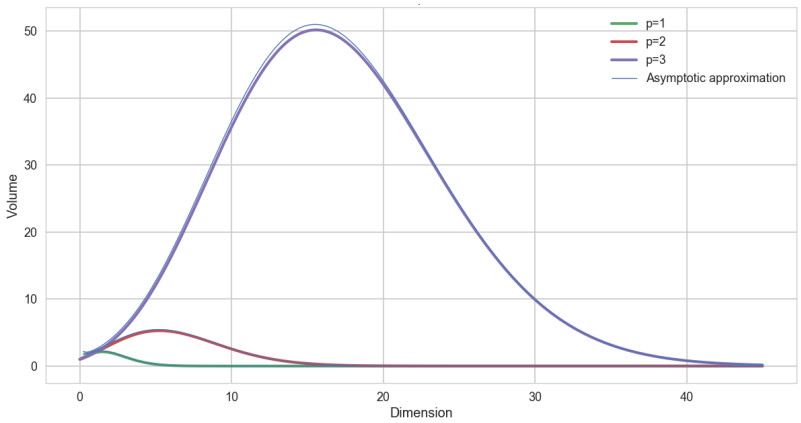


FIGURE D.3 – Volume de la boule unité en fonction de la dimension de son espace pour trois p-normes

Exercice D.1 (Concentration dans l'hypersphère)**.** *Soit $\varepsilon > 0$. On considère une hypersphère de rayon R. Montrer que :*

$$\frac{V_n^p(R-\varepsilon)}{V_n^p(R)} = \left(1 - \frac{\varepsilon}{R}\right)^n$$

C'est encore plus étrange : les points semblent se concentrer proche des frontières de l'hypersphère, donc en ayant un *centre* vide. Cela veut dire que plus la dimension augmente, plus le volume tend vers 0 et que dans le même temps les données se rapprochent des frontières.
Donc si l'on distribue des points uniformément dans une sphère, la distribution des distances entre les points ne sera pas informative du tout. Ces intuitions sont confirmées par la figure (D.4).

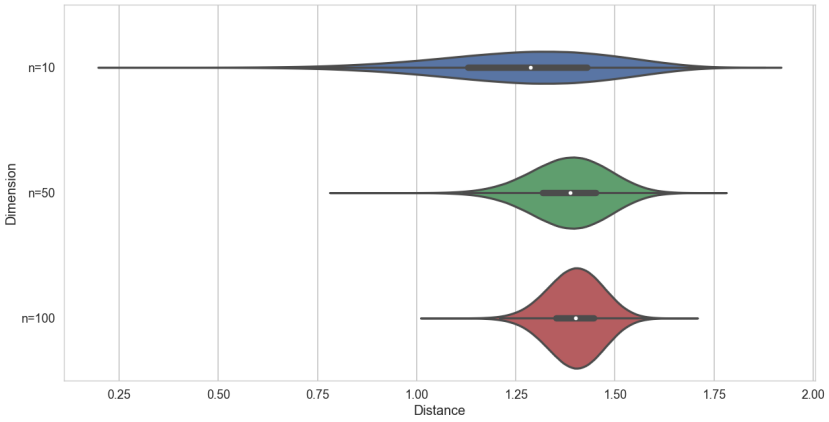


FIGURE D.4 – Distribution des distances entre chaque point en fonction de la dimension de l'espace

Ainsi, plus la dimension est grande, moins la notion de distance a du sens. Au-delà de l'aspect combinatoire et de stockage des données, avoir un modèle qui a moins d'indicateurs pour s'entraîner aura plus de chance d'être performant et *utile*. Par exemple, l'ensemble des méthodes de clustering par exemple seront largement impactées par une très grande dimension. Finalement, on peut remettre en cause l'exactitude d'une idée répandue : "*Avec plus d'informations les modèles sont meilleurs*". Ce qui est plus exact est qu'avec les informations utiles, les modèles sont meilleurs. C'est tout l'enjeu de la phase exploratoire et d'augmentation des données pour répondre à un problème de Machine Learning.

D.2 Orthogonalité à la surface d'une hypersphère

Nous avons donc montré que n'importe quelle distance issue d'une norme $\mathcal{L}_p$ était soumise au fléau de la dimension. En NLP *classique*, il est fréquent d'utiliser la distance cosinus, où le cadre est très souvent en très grande dimension. Les résultats semblent montrer que le fléau de la dimension n'affecte pas la distance cosinus. Vérifions.

On peut définir le produit scalaire entre $x, y \in \mathbb{R}^n$ comme :

$$< x, y >= \|x\|_2 \|y\|_2 \cos(\theta)$$

Avec θ l'angle entre le représentant de x et le représentant de y à l'origine comme défini dans la figure (D.5).

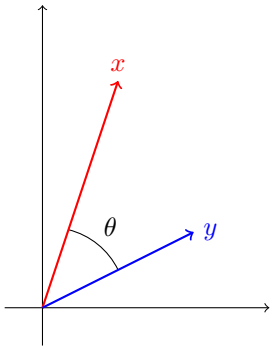


FIGURE D.5 – Définition de la métrique cosinus

Nous pouvons donc naturellement définir la métrique cosinus comme :

$$cosine(x, y) = \frac{< x, y >}{\|x\|_2 \|y\|_2}$$

On comprend que la distance cosinus sera bornée dans $[-1, 1]$ contrairement aux restes des distances qui ne sont pas bornées. Par définition, la métrique cosinus est liée à la distance euclidienne, et on remarque que si l'on prend x et y deux vecteurs unitaires, on obtient :

$$\begin{aligned} \|x - y\|_2^2 &= \|x\|_2^2 + \|y\|_2^2 - 2 < x, y > \\ &= \|x\|_2^2 + \|y\|_2^2 - 2cosine(x, y)\|x\|\|y\| \\ &= 2\left[1 - cosine(x, y)\right] \end{aligned}$$

Il serait donc très surprenant qu'une distance avec un lien aussi fort avec la distance euclidienne, ne souffre pas du fléau de la dimension. Nous avons un résultat intéressant :

Lemme 3. *Soit x, y deux vecteurs choisis indépendamment à la surface d'une hypersphère. Alors, avec une probabilité supérieure à $1 - \frac{1}{n}$:*

$$|cosine(x, y)| = O\left(\sqrt{\frac{\log n}{n}}\right)$$

Autrement dit, en prenant deux vecteurs aléatoirement à la surface d'une boule en dimension n, on a avec une très grande probabilité que ces deux vecteurs sont orthogonaux. Cela rend inutilisable la métrique cosinus en très grande dimension.

Démonstration. Soit $a \in \mathbb{R}^n$ tel que $\|a\|_2 = 1$. Soit $x \in S_n^2$ avec $S_n^p = \{x \in \mathbb{R}^n \mid \|x\|_p = 1\}$ la surface de l'hyperboule unitaire. Soit $x \in \mathbb{R}^n$ un vecteur construit avec chacune de ses coordonnées sélectionnées aléatoirement entre -1 et 1. Puis on normalise le vecteur x.

Soit $X =< a, x >$, alors il est simple de montrer que $\mathbb{E}[X] = 0$ et $\mathbb{E}\left[X^2\right] = \frac{1}{n}$. Ainsi, en exploitant l'inégalité de Chernoff on a :

$$\mathbb{P}(|X| \geqslant t) \leqslant 2e^{-\frac{nt^2}{4}}$$

Donc pour $\varepsilon = 2e^{-\frac{nt^2}{4}} \iff t = \sqrt{\frac{-4\log\left(\frac{\varepsilon}{2}\right)}{n}}$ et si l'on choisit $\varepsilon = \frac{1}{n}$, on obtient :

$$\mathbb{P}\left(|cosine(x,y)| \geqslant \sqrt{\frac{4\log(2n)}{n}}\right) \leqslant \frac{1}{n}$$

□

Cette preuve complète la présentation du second comportement étrange que l'on mentionne concernant les hyperboules et hypersphères.

Un deuxième problème majeur en grande dimension est que le nombre de données à obtenir pour être capable d'avoir des garanties statistiques sur la qualités de l'apprentissage est colossal, c'est exponentiel. Ainsi, on peut se poser la question de la capacité des algorithmes à *apprendre* en grande dimension.

D.3 Interpolation et extrapolation

L'ensemble du Machine Learning tel qu'on l'a présenté correspond à de l'interpolation et à essayer de faire en sorte que cette interpolation puisse être capable d'extrapoler correctement. Randall Balestriero, Jerome Pesenti et Yann Le Cun ont publié en 2021 l'article *Learning in High Dimension always amount to extrapolation* [BPL21] dont voici le résumé :

> *The notion of interpolation and extrapolation is fundamental in various fields from deep learning to function approximation. Interpolation occurs for a sample x whenever this sample falls inside or on the boundary of the given dataset ?s convex hull. Extrapolation occurs when x falls outside of that convex hull. One fundamental (mis)conception is that state-of-the-art algorithms work so well because of their ability to correctly interpolate training data. A second (mis)conception is that interpolation happens throughout tasks and datasets, in fact, many intuitions and theories rely on that assumption. We empirically and theoretically argue against those two points and demonstrate that on any high-dimensional (>100) dataset, interpolation almost surely never happens. Those results challenge the validity of our current interpolation/extrapolation definition as an indicator of generalization performances.*
>
> — Randall Balestriero, Jerome Pesenti et Yann Le Cun (2021)

L'objet de l'article est de montrer que l'on comprend et définit mal les notions d'interpolation et d'extrapolation en Machine Learning. Cela a des impacts théoriques et donc pratiques sur notre conception et les garanties mathématiques que l'on peut avoir sur les comportements des algorithmes présentés en très grande dimension. Nous invitons à lire en détail cet article pour en apprendre plus sur le sujet en lui-même, mais également pour voir qu'un domaine qui semble plutôt bien établi et en constante expansion se pose encore des questions sur ses fondements.

En résumé, le fléau de la dimension met en lumière les limites de notre intuition humaine et nous amène à nous questionner encore aujourd'hui sur les fondements communément acceptés. Être capable de répondre à ces questions nous permettrait d'être plus précis et plus complet sur notre approche de l'apprentissage en grande dimension.
De manière plus pragmatique, un data scientist doit être au courant que ces questions existent et que le fléau de la dimension va impacter son travail. D'où les techniques de réduction de dimension qui aident à résoudre le problème, mais ne le résolvent clairement pas par construction.

Annexe E

Support Vector Machine

Note : ce chapitre était enseigné de 2022 à 2024, il est conservé pour complétude.

Nous travaillons quasiment toujours avec les nombres réels, mais nous devons parfois exploiter les propriétés du monde complexe pour atteindre une vérité dans le monde réel. Citons par exemple les démonstrations des identités trigonométriques qui sont largement simplifiées avec l'utilisation de l'exponentielle complexe ou bien le théorème des résidus qui nous permet de calculer des intégrales trop compliquées pour être traitées simplement par l'analyse réelle.

> *The shortest path between two truths in the real domain passes through the complex domain.*
> — Jacques Hadamard (1991)

L'idée sous-jacente est d'exploiter les qualités d'abstractions des mathématiques pour réécrire le problème dans un nouveau paradigme et transférer les connaissances du domaine pour proposer une solution. Cette idée de réécriture de problème est au coeur de l'algorithme que nous allons présenter ici. Développé au sein du laboratoire Bell par Vladimir Vapnik et ses collègues Bernhard Boser, Isabelle Guyon et Corinna Cortes pour ne citer qu'eux [1], ces travaux s'appuient largement sur les développements en analyse sur l'optimisation plus spécifiquement.

E.1 Intuition

On considère un problème de classification, donc on a accès à un dataset défini comme :

$$\mathcal{D} = \left\{(x_i, y_i) \mid \forall i \leqslant n,\ x_i \in \mathbb{R}^d, y_i \in \{-1, 1\}\right\}$$

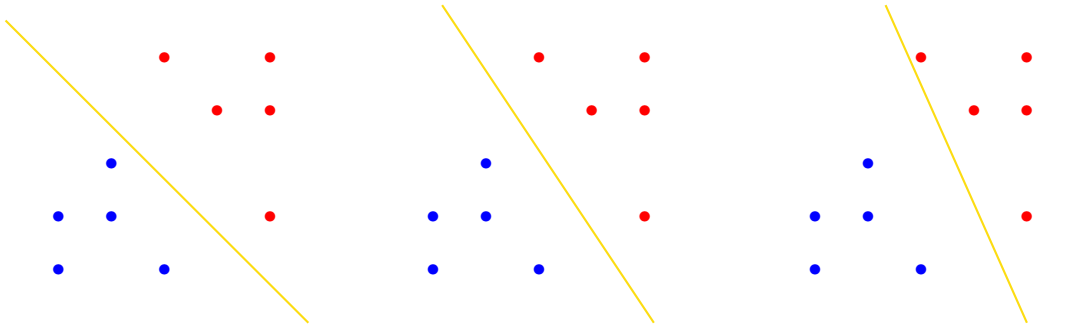
Noter qu'ici on demande à ce que $\mathcal{Y} = \{-1, 1\}$ et non $\{0, 1\}$ comme avant. En supposant que les données soient linéairement séparables, nous aimerions être capables de trouver un hyperplan qui sépare parfaitement les données. On sait qu'un hyperplan s'écrit sous la forme :

$$\langle \underbrace{w}_{\text{Direction } w \in \mathbb{R}^d}, x\rangle + \underbrace{b}_{\text{Offset } b \in \mathbb{R}} = 0 \tag{E.1}$$

Avec la figure (E.1) nous pouvons visualiser trois situations où les données sont parfaitement séparées. Mais quelle est la meilleure ?

On a intuitivement envie de dire que l'hyperplan (E.1b) est le meilleur car il coupe *loin* des données. C'est ce que l'on va appeler la *marge*. On veut trouver l'hyperplan qui va séparer les données parfaitement et avec la plus grande marge. En reprenant notre exemple, cela donne la figure (E.2).

1. Notons que Bernhard Boser est professeur à l'université de Berkeley, Isabelle Guyon est professeur titulaire à la Chaire Big Data de Paris Saclay, Corinna Cortes est à la tête de la recherche chez Google. Dans ce même laboratoire à la même époque il y avait également Yann Le Cun, à la tête de la recherche chez Meta.



(a) $w = \begin{pmatrix} -0.9 \\ -0.4 \end{pmatrix}$ et $b = 0.50$ (b) $w = \begin{pmatrix} -0.86 \\ -0.57 \end{pmatrix}$ et $b = 0.43$ (c) $w = \begin{pmatrix} -0.7 \\ -0.7 \end{pmatrix}$ et $b = 0.50$

FIGURE E.1 – Exemple de trois hyperplans possibles pour séparer parfaitement les données

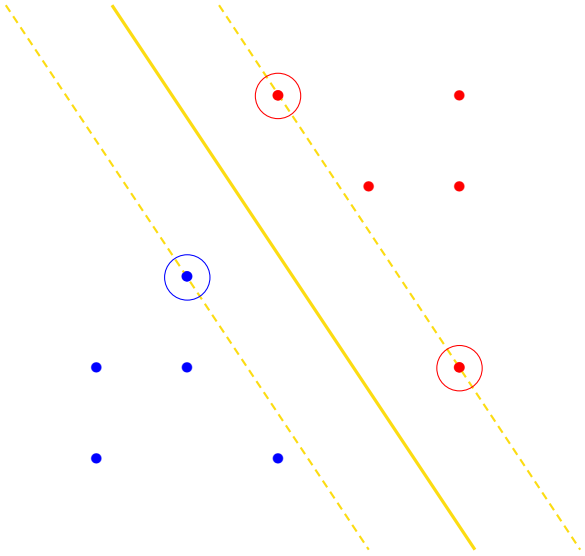


FIGURE E.2 – Classification avec l'algorithme Support Vector Machine

On comprend avec la figure (E.2) le nom de l'algorithme : il va chercher à construire la marge la plus grande possible en s'appuyant sur des vecteurs présents dans les données. On appellera ces vecteurs des vecteurs supports.
Maintenant que l'on sait ce qu'on cherche, écrivons-le.

E.2 Formalisation du problème

Nous allons commencer par nous intéresser au cas où les données sont séparables, donc il existe un hyperplan qui puisse séparer parfaitement les deux classes que l'on considère.
Puis nous généraliserons le problème au cas où les données ne sont pas linéairement séparables en exploitant les résultats et remarques faites dans le cas séparable.

E.2.1 Dans le cas séparable

Dire que l'on sépare parfaitement les données revient à dire que :

$$\begin{cases} \langle w, x_i \rangle + b \geqslant 0 & \text{pour } y_i = +1 \\ \langle w, x_i \rangle + b < 0 & \text{pour } y_i = -1 \end{cases} \iff y_i(\langle w, x_i \rangle + b) > 0$$

Il nous reste à définir *la plus grande marge*. On peut montrer que pour n'importe quel point x, la distance entre x et l'hyperplan défini dans l'équation (E.1) est $\frac{|\langle w, x\rangle + b|}{\|w\|}$. Mais essayer de trouver la marge la plus large possible en sachant que sa valeur dépend des points est difficile.

Exercice E.1 (Marge de valeur 1)**.** *En remarquant que :*

$$\forall \lambda > 0,\ \langle (\lambda w), x\rangle + (\lambda b) > 0 \iff \langle w, x\rangle + b > 0$$

Montrer que l'on peut définir la largeur totale de la marge comme $\gamma = \frac{2}{\|w\|_2}$.

Solution. Puisque l'équation nous montre que la décision que l'on prendra est indépendante de changements d'échelles, on peut choisir (w, b) tel que $|\langle w, x\rangle + b| = 1$. Et donc des deux côtés de l'hyperplan on a bien la marge γ attendue. □

De ce résultat, on comprend également que l'on peut ré-écrire *séparer parfaitement les données* comme :

$$\begin{cases} \langle w, x_i\rangle + b \geqslant 1 & \text{pour } y_i = +1 \\ \langle w, x_i\rangle + b < -1 & \text{pour } y_i = -1 \end{cases} \iff y_i(\langle w, x_i\rangle + b) - 1 > 0$$

On peut donc écrire le problème initial que l'on veut résoudre comme :

$$\begin{aligned} (w, b)^* = \underset{(w,b)\in\mathbb{R}^d\times\mathbb{R}}{\arg\max} \quad & \frac{2}{\|w\|_2} \\ \text{tel que} \quad & \forall i \leqslant n,\ y_i(\langle w, x_i\rangle + b) - 1 \geqslant 0 \end{aligned}$$

Mais nous préférerions l'écrire comme un problème avec un arg min comme nous l'avons fait jusqu'à maintenant.

Exercice E.2 (Transformation du problème)**.** *Montrer que :*

$$\underset{w\in\mathbb{R}^d}{\arg\max}\ \frac{2}{\|w\|_2} = \underset{w\in\mathbb{R}^d}{\arg\min}\ \frac{1}{2}\|w\|_2^2$$

Solution. Par définition d'une norme, $\|w\|_2 \geqslant 0$ et dans notre cas on ne peut obtenir un vecteur de norme nulle. Donc $\|w\|_2 > 0$. Or la fonction $x \mapsto \frac{1}{x}$ est décroissante sur $\mathbb{R}_+^*$. De plus, minimiser x revient à minimiser x^2 pour $x \geqslant 0$. Donc on obtient bien l'égalité souhaitée. □

Finalement, le problème que l'on cherche à résoudre est :

$$\begin{aligned} (w, b)^* = \underset{(w,b)\in\mathbb{R}^d\times\mathbb{R}}{\arg\min} \quad & \frac{1}{2}\|w\|_2^2 \\ \text{tel que} \quad & \forall i \leqslant n,\ y_i(\langle w, x_i\rangle + b) - 1 \geqslant 0 \end{aligned}$$

Et ce problème est déjà bien plus simple numériquement à résoudre que la précédente formulation. Nous avons fait tous ces efforts pour réécrire des problèmes pour seulement le cas où les données sont linéairement séparables. Remarquons que le problème est convexe, mais sous contrainte. Nous pouvons également définir une descente de gradient en projetant dans des espaces particuliers pour inclure l'impact des contraintes, mais c'est déjà plus difficile.
En regardant la figure (E.3) on se dit que même si les données sont linéairement séparables, il serait peut être mieux d'accepter d'avoir une erreur pour avoir une marge plus large.

Ceci motive encore plus le traitement du cas non-séparable des données.

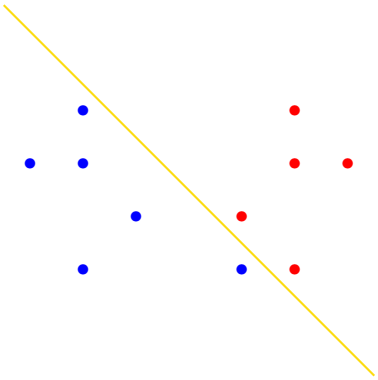


FIGURE E.3 – Cas séparable où il serait préférable d'avoir une erreur

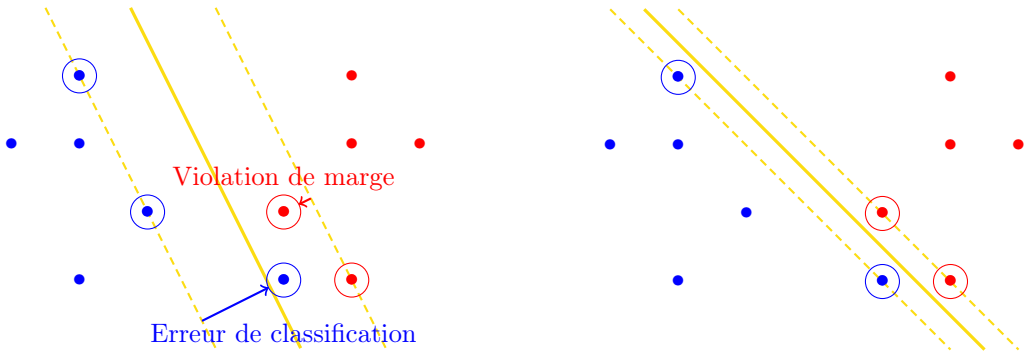
E.2.2 Dans le cas non-séparable

Nous cherchons maintenant une marge *soft* qui autorise certains points à être à l'intérieur de la marge. Ce que nous interdisions auparavant. Nous proposons alors le problème suivant :

Scalaire pour moduler la pénalisation

$$
\begin{aligned}
(w,b)^* = \underset{(w,b)\in\mathbb{R}^d\times\mathbb{R}}{\arg\min} \quad & \frac{1}{2}\|w\|_2 + \nu \sum_{i=1}^{n} \varepsilon_i \\
\text{tel que} \quad & \forall i \leqslant n,\ y_i(\langle w, x_i\rangle + b) \geqslant 1 - \varepsilon_i \\
& \forall i \leqslant n,\ \varepsilon_i \geqslant 0
\end{aligned}
$$

C'est une pénalisation ! Ici elle est convexe et c'est une pénalisation L_1 comme la régression LASSO, donc nous aurons plutôt des petites erreurs. Voyons comment cela se représente visuellement avec la figure (E.4).



(a) Avec $\nu = 1$ (b) Avec $\nu = 10$

FIGURE E.4 – Différence d'apprentissage pour deux valeurs de pénalisations différentes

On voit comment la pénalité fonctionne. Nous avons décidé de fixer la valeur $\nu = 1$ pour la figure (E.4a) et il y a 5 vecteurs supports ici, 3 sur les frontières et deux à l'intérieur de la marge. Il y a même une erreur de classification ici. Le modèle a pris les libertés qui lui semblaient nécessaires pour obtenir une marge la plus large possible.
Dans la figure (E.4b), pour la valeur $\nu = 10$, il y a 4 vecteurs support et aucune erreur de classification ou violation de marge. La pénalité était trop forte pour que le modèle prenne la liberté de faire des erreurs. Mais nous obtenons clairement une marge beaucoup plus petite que dans l'autre figure.

Exercice E.3 (Autre pénalisation). *Un datascientist qui travaille avec vous vous propose une autre manière de pénaliser les erreurs du modèle : la norme 0. Autrement dit compter le nombre d'erreur que se permet l'algorithme. Il prétend que l'on peut toujours résoudre le problème aussi facilement. Que lui répondez-vous ?*

Solution. Sa solution est probablement meilleure que de prendre la pénalisation L_1, mais elle n'est plus convexe. Cela veut dire qu'on ne peut plus compter sur les méthodes qui supposent la convexité pour résoudre un problème. Finalement, la pénalisation L_1 est la relaxation convexe de la pénalisation L_0. □

A ce stade, nous avons toujours un problème avec une fonction convexe à minimiser, mais sous contraintes. Les mathématiciens ont développé la branche de l'optimisation et nous allons exploiter plusieurs résultats fondamentaux de la discipline (sans les démontrer).

E.2.3 Problème primal et dual

Nous avons un problème d'optimisation sous contraintes où chacune des contraintes est une inégalité. Nous pouvons alors exploiter une notion d'optimisation qui s'appelle le **lagrangien**. Commençons par remarquer que plus généralement, on peut écrire notre problème d'optimisation de la manière suivante :

$$x^* = \arg\min_{x \in \Omega} \overbrace{f(x)}^{f : \Omega \to \mathbb{R}} \quad \text{telque} \quad \forall i \leqslant \underbrace{m}_{\text{Nombre de contraintes d'inégalités } g}, \; g_i(x) \leqslant 0$$

Définition 15 (Lagrangien). *Pour un problème d'optimisation sous contraintes d'inégalités comme défini précédemment, on définit le lagrangien du problème comme l'application* $\mathcal{L} : \mathbb{R}^d \times \mathbb{R}_+^m \to \mathbb{R}$ *:*

$$\mathcal{L}(x, \lambda) = f(x) + \sum_{i=1}^{m} \lambda_i g_i(x)$$

On appelle les valeurs du vecteur $\lambda \in \mathbb{R}_+^d$ les multiplicateurs de Lagrange. Chacun est associé à une contrainte d'inégalité précise. Pour saisir l'intérêt de considérer le lagrangien ici, nous devons rentrer encore plus dans les mathématiques d'optimisation liées à la théorie du point selle et de la dualité.
Pour le faire, prenons une application :

$$L : \overbrace{U}^{U \subset \mathbb{R}^d} \times \underbrace{P}_{P \subset \mathbb{R}_+^m} \to \mathbb{R}$$

Nous nous plaçons dans un cadre général, mais nous comprenons déjà avec les dimensions que nous allons pouvoir appliquer ces résultats à notre cas d'étude.

Définition 16 (Point selle). *On dit que le couple* $(u^*, p^*) \in U \times P$ *est un point selle de* L *si, et seulement si,*

$$\forall (u, p) \in U \times P, \; L(u^*, p) \leqslant L(u^*, p^*) \leqslant L(u, p^*)$$

On peut visualiser un point selle d'une fonction avec la figure (E.5).
Il s'agit d'un point très particulier où selon le chemin d'une coordonnée nous sommes au maximum, et selon l'autre nous somme au minimum. Autrement dit, u^* est un minimum de la fonction $u \mapsto L(u, p^*)$ et p^* est un maximum de la fonction $p \mapsto L(u^*, p)$. Il est matérialisé par un point rouge dans la figure

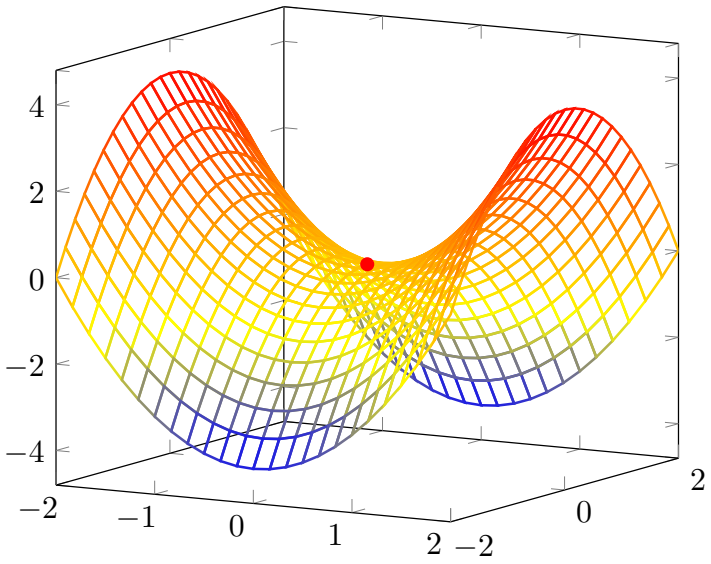


FIGURE E.5 – Point selle de la fonction $(x, y) \mapsto x^2 - y^2$

précédente.

Le titre de la section est problème primal et dual, mais nous n'avons introduit aucun des deux à ce stade.

Définition 17 (Problème primal et dual)**.** *Avec les notations précédentes, on définit les fonctions :*

$$\mathcal{I}(u) = \sup_{p \in P} L(u, p) \quad \text{et} \quad \mathcal{G}(p) = \inf_{u \in U} L(u, p)$$

On appelle problème ***primal*** *le problème de minimisation :*

$$\inf_{u \in U} \mathcal{I}(u) = \inf_{u \in U} \sup_{p \in P} L(u, p)$$

On appelle problème ***dual*** *le problème de maximisation :*

$$\sup_{p \in P} \mathcal{G}(p) = \sup_{p \in P} \inf_{u \in U} L(u, p)$$

On a défini deux problèmes qui se ressemblent beaucoup, mais qui sont différents. Le résultat suivant nous montre que ces deux problèmes sont intimement liés au point selle (u^*, p^*).

Théorème 7 (Dualité)**.** *Le point $(u^*, p^*) \in U \times P$ est un point selle de L si, et seulement si,*

$$L(u^*, p^*) = \mathcal{I}(u^*) = \mathcal{G}(p^*)$$

Autrement dit, un point selle résout à la fois le problème primal et dual ! Nous ne prouverons pas ce théorème mais chercherons à l'appliquer directement à notre problème d'optimisation.

Proposition 8. *Avec les notations précédentes, si (w^*, λ^*) est un point selle de $\mathcal{L}$ le lagrangien alors w^* est un minimum global de f sur Ω.*
De plus, si f et les contraintes $(g_i)_{i\leqslant m}$ sont de classe $\mathcal{C}^1$ et convexe, alors :

$$\exists \lambda^* \in \mathbb{R}_+^m, \begin{cases} \nabla f(w^*) + \sum_{i=1}^{m} \lambda_i^* \nabla g_i(w^*) & = 0 \\ \forall i \leqslant m, \ \lambda_i^* \geqslant 0 \\ \forall i \leqslant m, \ \lambda_i^* g_i(w^*) = 0 \\ \forall i \leqslant m, \ g_i(w^*) \leqslant 0 \end{cases}$$

Avec des conditions de convexité, nous avons obtenu beaucoup d'égalités et d'inégalités qui nous seront utiles pour la résolution du problème. Nous avons suffisamment de matériel pour trouver les x et λ.

A nouveau, nous ne démontrerons pas ce résultat, mais appliquons-le à notre étude. L'intérêt des deux résultats est que nous avons une stratégie de résolution en deux étapes :

1. Trouver, pour λ fixé le meilleur x en fonction de λ
2. Trouver le meilleur λ et en déduire l'expression finale de x

Pour écrire le lagrangien, on décide de noter avec deux symboles les multiplicateurs de lagrange pour distinguer les deux types d'inégalités. Pour rappel, on veut résoudre :

$$\begin{aligned} (w,b)^* = \underset{(w,b)\in\mathbb{R}^d\times\mathbb{R}}{\arg\min} \quad & \frac{1}{2}\|w\|_2 + \nu \sum_{i=1}^{n} \varepsilon_i \\ \text{tel que} \quad & \forall i \leqslant n, \ y_i(\langle w, x_i\rangle + b) \geqslant 1 - \varepsilon_i \\ & \forall i \leqslant n, \ \varepsilon_i \geqslant 0 \end{aligned}$$

Nous sommes bien dans le cadre du théorème avec des contraintes d'inégalité qui sont linéaires donc convexes. Le lagrangien est explicité dans l'équation (E.2).

$$\mathcal{L}\left((w,b,\varepsilon_i),(\mu_i,\delta_i)\right) = \frac{1}{2}\|w\|^2 + \nu\sum_{i=1}^{n}\varepsilon_i - \left(\sum_{i=1}^{n}\mu_i\left[y_i(\langle w,x_i\rangle + b) - (1-\varepsilon)\right] + \sum_{i=1}^{n}\delta_i\varepsilon_i\right) \tag{E.2}$$

En nous appuyant sur les deux résultats précédents et la stratégie, nous avons les outils pour résoudre le problème dual. Puisque l'ensemble des conditions de la proposition (8) sont remplies, nous avons que :

$$\begin{cases} \frac{\partial \mathcal{L}}{\partial w}((w,b,\varepsilon_i),(\mu_i,\delta_i)) & = 0 \\ \frac{\partial \mathcal{L}}{\partial b}((w,b,\varepsilon_i),(\mu_i,\delta_i)) & = 0 \\ \forall i \leqslant n, \ \frac{\partial \mathcal{L}}{\partial \varepsilon_i}((w,b,\varepsilon_i),(\mu_i,\delta_i)) & = 0 \\ \forall i \leqslant n, \ \mu_i & \geqslant 0 \\ \forall i \leqslant n, \ \delta_i & \geqslant 0 \\ \forall i \leqslant n, \ \mu_i\left[y_i(\langle w,x_i\rangle + b) - (1-\varepsilon_i)\right] & = 0 \\ \forall i \leqslant n, \ \delta_i\varepsilon_i & = 0 \end{cases}$$

Résoudre un tel système quand on n'y est pas habitué peut être difficile. Suivons donc une résolution guidée.

Exercice E.4 (Résolution). *On considère le système précédent.*

1. *En exploitant la première équation, obtenez une expression de w.*
2. *En exploitant la deuxième équation, obtenez une condition sur les $(\mu_i)_{i\leqslant n}$ et les $(y_i)_{i\leqslant n}$.*
3. *A l'aide de l'équation (3) et de l'équation (5), montrer que $\forall i \leqslant n,\ 0 \leqslant \mu_i \leqslant \nu$*
4. *Pour $i \leqslant n$ fixé, que peut-on déduire sur le vecteur x_i quand $\mu_i = 0$? On s'aidera de l'équation (6).*
5. *Pour $i \leqslant n$ fixé, que peut-on déduire sur le vecteur x_i quand $\mu_i > 0$? On s'aidera de l'équation (6).*
6. *Pour $i \leqslant n$ fixé, que peut-on déduire sur le vecteur x_i quand $\mu_i = \nu$? On s'aidera de l'équation (3) et de l'équation (7).*

Solution. 1. En dérivant le lagrangien par rapport à w, on obtient :

$$w = \sum_{i=1}^{n} \mu_i y_i x_i$$

Les valeurs de $(\mu_i)_{i\leqslant n}$ semblent centrales pour construire l'hyperplan puisqu'elles sont dans la constitution de sa direction.

2. En dérivant le lagrangien par rapport à b, on obtient :

$$\sum_{i=1}^{n} \mu_i y_i = 0$$

A nouveau, les valeurs des $(\mu_i)_{i\leqslant n}$ apparaissent, il faut que l'on comprenne plus en profondeur le comportement.

3. Soit $i \leqslant n$ fixé. En dérivant le lagrangien par rapport à ε_i, on a :

$$\nu - \mu_i - \delta_i = 0$$

En combinant avec le fait que $\delta_i \geqslant 0$ (équation 4) et $\mu_i \geqslant 0$ (équation 5), on peut écrire :

$$0 \leqslant \mu_i \leqslant \nu$$

Nous sommes donc capables de borner les valeurs des $(\mu_i)_{i\leqslant n}$.

4. Soit $i \leqslant n$ fixé. Avec l'équation (6), et la supposition que $\mu_i = 0$, cela veut dire que :

$$y_i(\langle w, x_i\rangle + b) > 1 - \varepsilon_i$$

Autrement dit le vecteur x_i est bien classifié et il n'est pas un vecteur support, donc en dehors de la marge.

5. Soit $i \leqslant n$ fixé. Avec l'équation (6), et la supposition que $\mu_i > 0$, cela veut dire que :

$$y_i(\langle w, x_i\rangle + b) = 1 - \varepsilon_i$$

Autrement dit que le vecteur x_i est un vecteur support et qu'il est a une erreur de marge ε_i.

6. Soit $i \leqslant n$ fixé. On suppose que $\mu_i = \nu$. De l'équation (3) nous avons montré dans une question précédente que μ_i, ν et δ_i sont liés, et que dans le cas où $\mu_i = \nu$ on a $\delta_i = 0$. En injectant cette information dans l'équation (7), on en déduit que $\varepsilon_i > 0$ donc que le vecteur x_i est une erreur de classification de marge.

□

De cet exercice nous comprenons que la valeur des $(\mu_i)_{i\leqslant n}$ est centrale et permet d'avoir une classification de chacun des points :

- Si $\mu_i = 0$, alors x_i est bien classifié et n'est pas dans la marge.
- Si $0 < \mu_i < \nu$, alors x_i est un vecteur support sur la marge.
- Si $\mu_i = \nu$ alors x_i est une erreur de classification avec une erreur de marge.

Avec l'écriture de w que nous avons exhibée, et la remarque sur la valeur de $\mu_i = 0$, on comprend pourquoi l'algorithme s'appelle *Support Vector Machine*. N'interviennent dans le calcul de w uniquement les points mal classés où à la frontière. Ils ne serviront donc pas pour le second problème (problème dual) que l'on doit résoudre.

$$\begin{aligned}(\mu_i^*)_{i\leqslant n} = \underset{(\mu_i)_{i\leqslant n}\in\mathbb{R}_+^n}{\arg\max} \quad & \sum_{i=1}^n \mu_i - \frac{1}{2}\sum_{i=1}^n\sum_{j=1}^n \mu_i\mu_j(y_iy_j\langle x_i, x_j\rangle) \\ \text{tel que} \quad & \forall i \leqslant n,\ 0 \leqslant \mu_i \leqslant \nu \\ & \sum_{i=1}^n \mu_i y_i = 0\end{aligned}$$

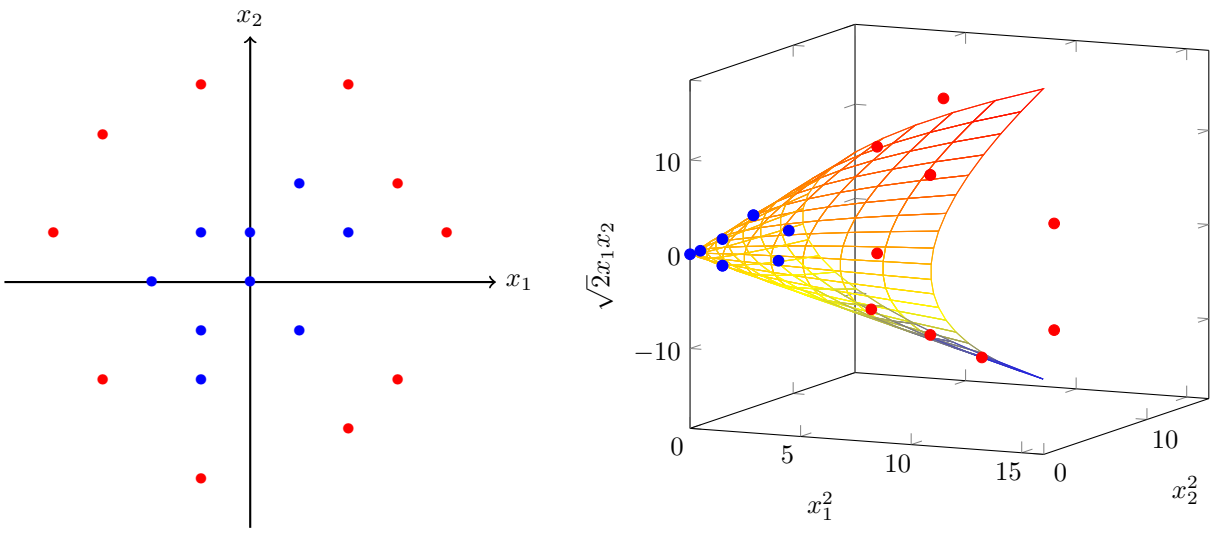
On peut encore simplifier l'écriture du problème avec des notations matricielles. On note 1_d le vecteur contenant d coefficients tous égaux à 1. On peut réécrire le problème :

$$\begin{aligned}\mu_{i\leqslant n}^* = \underset{\mu\in\mathbb{R}_+^n}{\arg\max} \quad & \langle \mu, 1\rangle - \frac{1}{2}\mu^t M\mu \\ \text{tel que} \quad & \forall i \leqslant n,\ 0 \leqslant \mu_i \leqslant \nu \\ & \sum_{i=1}^n \mu_i y_i = 0\end{aligned}$$

Où l'on a défini la matrice de Gram $M \in \mathcal{M}_{n,n}$ qui a pour coefficient $M_{ij} = y_iy_j\langle x_i, x_j\rangle$. Le grand intérêt d'avoir réécrit le problème de la sorte est que la fonction que l'on cherche à maximiser s'écrit avec le produit scalaire des paires $\langle x_i, x_j\rangle$, ce qui veut dire que l'on peut utiliser le **kernel trick**.

E.2.4 Kernel trick

Nous venons de résoudre le problème que nous nous sommes posés : trouver l'hyperplan qui sépare le mieux les données avec la marge la plus grande possible. Mais nous n'avons en pratique réglé que le cas où les données seront séparées par un hyperplan. Peut-être que la meilleure forme n'est pas linéaire. La figure (E.6) nous montre un exemple.



(a) Dans l'espace initial

(b) Dans l'espace de dimension supérieure

FIGURE E.6 – Application d'un noyau pour classifier deux groupes

Comme on le voit dans la figure (E.6a) les données disponibles en deux dimensions sont bien séparables, mais pas linéairement. En revanche elles le sont en trois dimensions dans la figure (E.6b). L'idée est donc de déplacer les données d'un espace de petite dimension à un espace de plus grande dimension, et on peut même le faire non linéairement. On exploitera donc notre travail de séparation par un hyperplan dans l'espace de plus grande dimension.

Un noyau [2] K est obtenu à partir d'une application qui transfère une observation x_i vers un espace de plus grande dimension.

Espace de départ de dimension d

$$\phi : \mathbb{R}^d \to \mathbb{R}^{d'}$$

Espace d'arrivé de dimension $d' > d$

Et nous avons utilisé dans notre exemple l'application : $\phi(x_1, x_2) = (x_1^2, x_2^2, \sqrt{2}x_1x_2)$. On voit bien qu'on passe de $\mathbb{R}^2$ à $\mathbb{R}^3$.

Définition 18 (Noyau)**.** *Avec les notations précédentes, on appelle K le noyau associé à ϕ, l'application définie par :*

$$K(x, y) = \langle \phi(x), \phi(y) \rangle$$

Avant de voir des exemples de noyaux, nous comprenons que l'on peut remplacer les $\langle x_i, x_j \rangle$ par $K(x_i, x_j)$, et donc exploiter des formes de noyaux complètement différents. Autrement dit, nous sommes capables de séparer des données même de manière non linéaire, sans faire aucun effort théorique par rapport à la théorie pour séparer linéairement des données.
Il reste une dernière difficulté : si nous utilisons un Kernel, nous calculons $\phi(w)$ et non directement w. Comment faire, alors qu'on ne connaît quasiment jamais l'expression exacte de ϕ mais seulement de K, pour prédire pour une nouvelle valeur ?

Puisque nous avons vu que l'on peut remplacer l'ensemble des produits scalaires par un noyau K que l'on sélectionne, et que seuls quelques vecteurs supports sont nécessaires pour définir l'hyperplan, alors on peut écrire le classifier comme :

2. *Kernel* en allemand.

$$f(x) = \langle \phi(w), \phi(x) \rangle + b = \sum_{i=1}^{n} \mu_i y_i K(x_i, x_j)$$

Et il suffit de prendre le signe de f pour avoir la classe prédite ! Mais, nous n'avons jamais explicité comment calculer b.

Exercice E.5 (Valeur de b)**.** *En reprenant le système du problème primal, et en s'aidant particulièrement de l'équation (6), expliquer comment calculer b.*

Solution. La valeur de b peut être calculée à partir de n'importe quel vecteur support classé comme $+1$ par exemple. On a que $\langle \phi(w), \phi(x) \rangle + b = 1 \Leftrightarrow b = 1 - K(w, x)$. □

E.3 L'algorithme en pratique

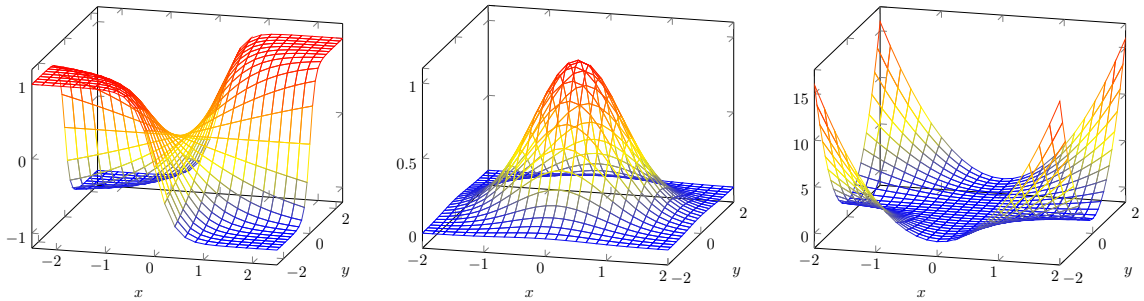
Maintenant que nous avons décrit l'algorithme avec les détails mathématiques nécessaires (en omettant les preuves tout de même), il faut que nous sachions quels sont les outils avec lesquels nous allons pouvoir travailler. Dans un premier temps, nous devons comprendre quel noyau classique nous pouvons utiliser, puis quels sont les autres paramètres que l'on peut faire varier pour obtenir les meilleures performances possibles.

E.3.1 Noyaux classiques

On peut définir des noyaux nous-mêmes, mais les noyaux les plus classiques sont les suivants :

- Noyau linéaire : $K(x_1, x_2) = \langle x_1, x_2 \rangle$
- Noyau polynomial : $K(x_1, x_2) = (\gamma \langle x_1, x_2 \rangle + r)^d$
- Noyau Radial Basis Function : $K(x_1, x_2) = \exp\left\{ -\dfrac{\|x_1 - x_2\|^2}{\sigma^2} \right\} = \exp\left\{ -\gamma \|x_1 - x_2\|^2 \right\}$
- Noyau sigmoid : $K(x_1, x_2) = \tanh\left(\gamma \langle x_1, x_2 \rangle + r \right)$

Nous avons donc fait la théorie avec le noyau linéaire, et on remarque que le noyau *Radial Basis Function* ressemble beaucoup à la gaussienne et dans ce cas on appelle le paramètre $\frac{1}{\gamma}$ la *bande*, ce qui se justifie visuellement. Les différents noyaux sont présentés dans la figure (E.7).



(a) Noyau sigmoid (b) Noyau RBF (c) Noyau polynomial

FIGURE E.7 – Représentation des différents noyaux classiques (pour $\gamma = 1$, $r = 0$ et $d = 2$)

Remarquons que nous ne sommes pas forcément capables d'expliciter pour chacun des exemples la fonction ϕ initiale, nous avons juste besoin de savoir que K est bien un noyau [3].

3. Pour ça on utilise le théorème de Mercer, mais nous ne le présenterons pas ici.

E.3.2 Fine-tuning

- Paramétrer le noyau :
  - kernel : pour définir le noyau avec lequel on veut travailler
  - degree : degré du noyau polynômial si kernel = 'poly', ignorer sinon
  - gamma : coefficient du noyau pour les noyaux 'poly', 'rbf' ou 'sigmoid'.
  - coef0 : terme indépendant (r) dans le noyau polynomial ou sigmoid, ignorer sinon.
- Paramétrer l'algorithme :
  - C : la pénalité ν
  - max_iter : le nombre maximum d'itérations du solveur numérique pour résoudre le problème
  - tol : tolérance pour le critère d'arrêt du solveur

Exercice E.6. *Vous travaillez avec une datascientist qui vient d'apprendre comment fonctionne un SVM, mais elle a encore besoin d'être guidée. Elle a pour mission d'identifier des entreprises qui vont faire faillite dans 6 mois. Elle dispose d'un dataset adapté.*
Son intuition est qu'elle peut extraire de l'information utile pour aider la prise de décision d'investir où non dans une entreprise avec les vecteurs supports. (qui sont ici des entreprises). Mais elle ne sait pas comment s'y prendre. Voici les questions qu'elle vous pose.

1. *Dois-je travailler uniquement avec le noyau linéaire pour avoir des vecteurs de support ?*
2. *J'aimerais être la plus précise possible dans ma classification. Comment dois-je régler les hyperparamètres ?*
   Après une première version, elle se rend compte que les frontières de décisions sont trop compliquées pour être traduites simplement.
3. *J'ai trouvé que le noyau RBF était le plus performant, et de loin. Comment puis-je le conserver mais obtenir des frontières de décisions peut-être un peu plus simples ? Tant pis s'il y a des erreurs.*

Solution.
1. Tous les noyaux utiliseront des vecteurs supports, donc on peut exploiter l'ensemble des noyaux à disposition, voire en définir un spécifique.
2. Le paramètre central pour cela sera C avec une valeur élevée pour pénaliser autant que possible les erreurs de classification. Mais parfois, le modèle n'aura pas la puissance nécessaire pour éviter toutes les erreurs. Donc il faut exploiter les paramètres des noyaux : γ notamment.
3. Inversement ici, on veut une frontière de décision plus simple donc on peut baisser la valeur de C et travailler autrement les paramètres des noyaux. Par définition du problème, on peut garantir que la valeur des erreurs commises est faite pour être minimale via la régularisation L_1.

□

L'exemple est inspiré de l'article *Learning machine supporting bankruptcy prediction* publié par Wolfgang Karl Hardle, Linda Hoffmann et Rouslan Moro en 2011 [HHM11]. L'algorithme y est présenté de la même manière qu'ici mais avec une introduction plus complète à la théorie de Vapnik développée en 1999 dans le livre *The nature of statistical learning theory* [Vap99]. Nous avons choisi de ne pas faire cette introduction parce qu'elle est très théorique, technique et nécessite beaucoup de temps, même si elle est extrêmement intéressante.

Nous avons donc présenté et détaillé le fonctionnement d'un nouvel algorithme, hautement modifiable et qui pourtant permet d'avoir des garanties mathématiques très fortes. Le temps d'entraînement sur de très grandes données fait que l'algorithme est moins utilisé que les précédents, mais peut tout à fait entrer en compétition avec ces derniers.

Annexe F

Stacking et Blending : au-delà du Bagging et Boosting

Nous avons déjà traité de la méthode du bagging et du boosting en détail. Nous allons présenter ici deux méthodes ensemblistes supplémentaires, proches, pour tirer parti des nombreux algorithmes de Machine Learning à notre disposition.
[Wol92] introduit la notion de stacking mais telle que nous la connaissons aujourd'hui, et telle que nous allons la présenter, est décrite dans [Bre96b]. Mais ce n'est pas avant [VdLPH07] que les fondations mathématiques que l'on va expliciter ici ne sont développées.

Dans un cadre d'apprentissage supervisé, nous supposons que nous disposons d'un dataset $\mathcal{D}$ qui permet de répondre à un problème de classification ou de régression. A l'aide des algorithmes que nous avons présenté jusqu'ici, nous sommes capables de construire m modèles. L'idée commune du stacking et du blending est de construire un **méta-modèle** qui est un algorithme, potentiellement le même qu'un des m modèles, qui apprend des prédictions des m-modèles.
Pour illustrer l'idée dans un autre contexte. Nous souhaitons prédire le cheval qui remportera une course. Pour prendre notre décision, nous demandons à trois experts de nous donner leurs prédictions. Après avoir longuement travaillé avec eux, nous avons appris à prendre de meilleures décisions de pari en s'appuyant sur leurs expertises respectives. C'est la même chose pour le stacking et le blending : un méta-modèle s'appuie sur d'autres modèles.

Ce qui va différencier les deux approches est la manière d'entraîner ce méta-modèle. Commençons par le blending : le dataset $\mathcal{D}$ est séparé en deux parties (pas forcément égales). La première partie sert à entraîner les premiers algorithmes et ils prédisent la valeur souhaitée sur la seconde partie du dataset. C'est sur ces prédictions que le méta-modèle apprend à prédire la vraie valeur.

Un des avantages de cette méthode est qu'elle est très simple à comprendre et mettre en place. Le désavantage est que les premiers algorithmes n'apprennent pas sur l'ensemble de la base de données disponible.
[Bre96b] propose d'exploiter la totalité du dataset $\mathcal{D}$ à la fois pour l'entraînement des premiers algorithmes mais également pour le méta-modèle. Pour éviter les problèmes évidents de fuite de données, cela est fait via une validation croisée présentée dans la section (**??**).

Ces deux méthodes peuvent améliorer les performances de prédiction par rapport à chacun des algorithmes qui les constituent. En revanche, on ajoute une couche de complexité non négligeable qui nuira à la simplicité d'interprétation ainsi qu'à une certaine robustesse dans le temps. Les erreurs commises par un algorithme suite à une modification de la distribution du dataset de départ peuvent être démultipliées par le méta-modèle dans un second temps qui dégrade considérablement la performance de l'algorithme.

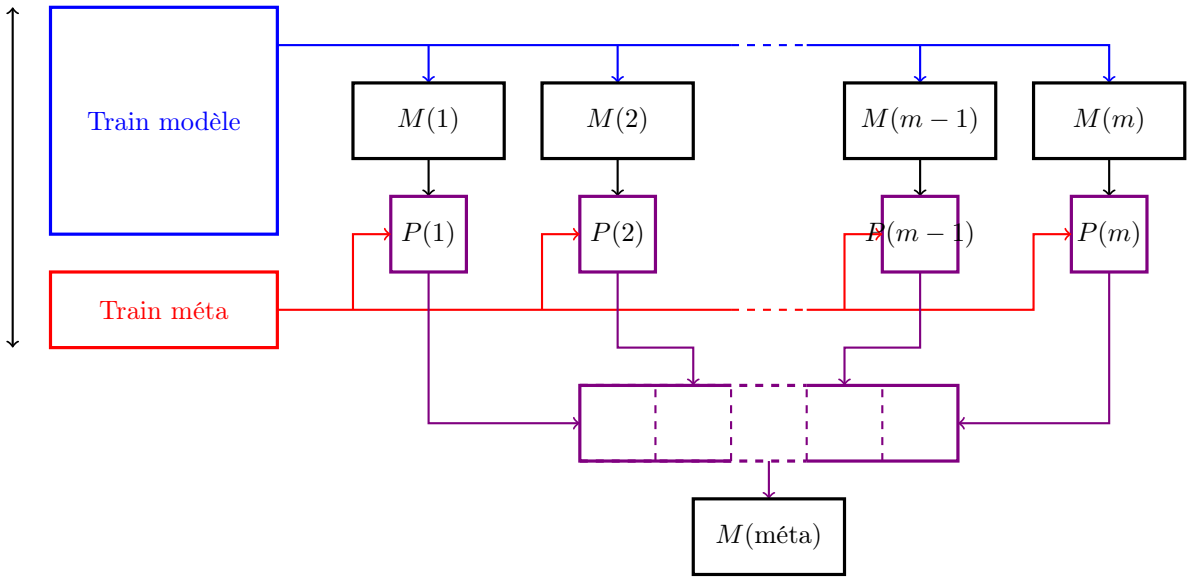


FIGURE F.1 – Illustration du blending

Annexe G

Double descente : vers le Machine Learning moderne

En 1975 est publié le premier *Jargon File* : un glossaire de l'argot des programmeurs. Ce projet se décrit comme le *"Hacker's Dictionary"*. A l'intérieur de ce glossaire se trouve la définition du terme *grok* :

> *When you claim to "grok" some knowledge or technique, you are asserting that you have not merely learned it in a detached instrumental way but that it has become part of you, part of your identity. For example, to say that you "know" Lisp is simply to assert that you can code in it if necessary - but to say you "grok" Lisp is to claim that you have deeply entered the world-view and spirit of the language, with the implication that it has transformed your view of programming.*
>
> — The Jargon File (1975)

C'est également un des termes qui a été proposé pour nommer le phénomène que nous allons présenter rapidement dans cette annexe. Dans la même lignée que l'introduction du chapitre sur la régression logistique avec le logarithme complexe ou la conclusion de l'annexe présentant le fléau de la dimension, cette annexe ne vise qu'à proposer une vision instantanée d'une partie de la recherche contemporaine. Ces recherches, notamment sur ce phénomène, challengent nos compréhensions des sujets et nécessitent des développements inédits pour mieux appréhender les arcanes du Machine Learning

G.1 Prédiction de la théorie de Vapnik-Chervonenkis

Comme expliqué dans le chapitre (E) sur les SVM, nous avons décidé de ne pas présenter la théorie de Vapnik-Chervonenkis pour éviter une théorie et une technique trop complexe pour les contraintes du cours, bien qu'elle soit passionnante. Pour tout de même présenter les grandes lignes en reprenant [Sch13] :

> *Intuitively, for a learned classifier to be effective and accurate in its predictions, it should meet three conditions :*
>
> 1. *It should have been trained on "enough" training examples*
> 2. *It should provide a good fit to those training examples (usually meaning that it should have low training error)*
> 3. *It should be simple*
>
> *This last condition, our expectation that simpler rules are better, is often referred to as Occam's razor.*
> *In formalizing these conditions, Vapnik and Chervonenkis established a foundation for understanding the fundamental nature of learning, laying the groundwork for the design of effective and principled learning algorithms.*
>
> — Robert E. Shapire (1991)

Voyons comment ces concepts se traduisent théoriquement. L'objectif que l'on poursuit, présenté dans le chapitre d'introduction, est d'obtenir un modèle qui permet une généralisation forte. C'est-à-dire que le modèle soit performant sur l'ensemble des paires $(x, y) \in \mathbb{R}^d \times \mathcal{Y}$, en reprenant les notations du cours, pour une tâche donnée.

Nous noterons $R(f)$ la valeur de cette erreur de généralisation sur l'ensemble des possibilités. Bien sûr, nous ne sommes pas capables de calculer sa valeur. Nous devons donc suivre une approche : estimer sa valeur à l'aide d'un dataset d'entraînement. C'est celle que nous avons suivi jusqu'à présent. L'hypothèse de Vapnik est qu'avec la loi forte des grands nombres, plus nous augmenterons le nombre d'observations dans le dataset d'entraînement, plus notre estimation $\hat{R}(f)$ de $R(f)$ sera précise.

Partant de ces deux définitions et de cette hypothèse, un des résultats fort de la théorie VC est une borne supérieure (G.1) sur la capacité de généralisation d'un algorithme de Machine Learning. Avec une probabilité d'au moins $1 - \eta$, on a :

Complexité du modèle

$$R(f) \leqslant \hat{R}(f) + \sqrt{\frac{h\left[\ln\left(\frac{2n}{h}\right) + 1\right] - \ln\left(\frac{\eta}{4}\right)}{n}} \tag{G.1}$$

Observations test

Il est important de préciser que la valeur [1] de h doit être inférieure à $n+1$. Regardons cette équation à la lumière des trois conditions présentées plus tôt. Avoir entraîné sur suffisamment d'observations permet également d'avoir une erreur sur le dataset d'entraînement faible. Ainsi, on minimise la première partie de l'inégalité. La deuxième partie est minimisée quand le paramètre h qui correspond à la complexité du modèle est faible. Nous voyons comment ces trois principes sont représentés dans cette borne théorique.

Exercice G.1. *Prouver que la seconde partie de la borne est bien minimisée quand la complexité du modèle est faible.*

Solution. Étudions la fonction f définie sur $[1, n+1]$ par :

$$f(x) = \frac{1}{n}\left(x\left[\ln\left(\frac{2n}{x}\right) + 1\right] - \ln\left(\frac{\eta}{4}\right)\right)$$

On montre simplement que :

$$f'(x) = \frac{1}{n}\ln\left(\frac{2n}{x}\right)$$

Puisque $x \leqslant n+1$, on a que $f'(x) \geqslant 0$ sur l'ensemble de définition. On comprend également que f va croitre rapidement au début puis plus lentement vers la fin de l'intervalle. □

La courbe prédite par cette borne et le principe du tradeoff biais-variance est représenté par la figure (G.1).

1. Il s'agit là de la complexité au sens de Vapnik-Chervonenkis qui est la partie théorique difficile.

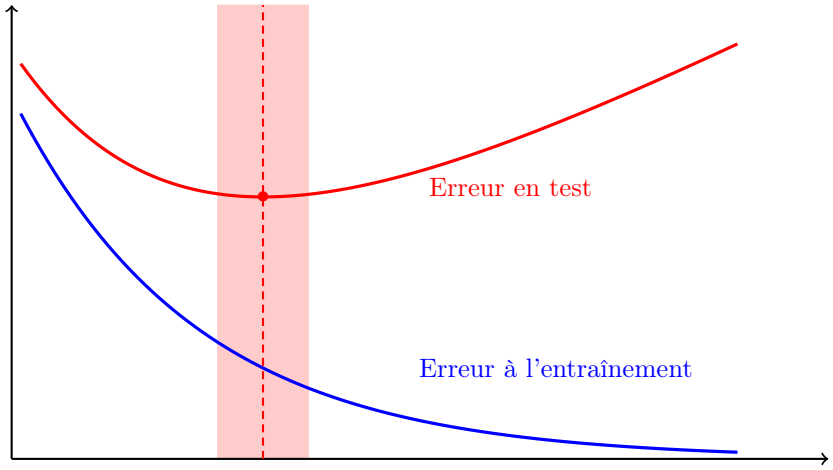


FIGURE G.1 – Prédiction théorique de la théorie VC

C'est ce que nous observons en pratique ! La zone en rouge représente la zone que l'on cherche à atteindre pour obtenir la meilleure généralisation possible. Si l'on donne trop de puissance au modèle, on diminue l'erreur sur le train voire on atteint le régime d'interpolation au détriment de la performance sur le test. *Et pourtant...*

G.1.1 Cas d'AdaBoost

Expliciter la dépendance de h en fonction des paramètres de l'algorithme que l'on considère est un problème complexe, mais nous pouvons l'appliquer pour l'algorithme AdaBoost. Décrit dans la section (5.1), il fonctionne par itérations d'apprentissage T et cherche à obtenir de meilleures performances à chaque tour. Nous sommes capables d'obtenir la borne suivante pour l'algorithme AdaBoost :

$$R(f) \leqslant \hat{R}(f) + \mathcal{O}\left(\sqrt{\frac{T\ln(|H|) + T\ln\left(\frac{n}{T}\right) - \ln(\eta)}{n}}\right) \tag{G.2}$$

Complexité des autres hypothèses

Signifie : de l'ordre de

Nous pouvons réaliser le même exercice qu'auparavant et nous aurons le même résultat théorique : AdaBoost va sur-apprendre les données. *Et pourtant* dans la figure (G.2) on observe :

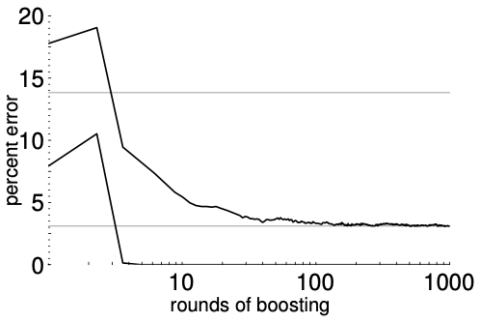


FIGURE G.2 – Erreur de classification d'AdaBoost en fonction du nombre d'époques (R. Schapire [Sch13])

Dans un premier temps nous retrouvons la prédiction théorique, mais soudainement la généralisation s'améliore ! Deux phénomènes contre-intuitifs se réalisent :

1. Une amélioration de la généralisation a lieu alors que nous avons largement dépassé le point d'interpolation
2. Une amélioration de la généralisation a lieu alors que nous avons depuis un grand nombre d'époques une erreur d'entraînement extrêmement faible voire nulle

> *Breakthroughs in machine learning are rapidly changing science and society, yet our fundamental understanding of this technology has lagged far behind. Indeed, one of the central tenets of the field, the bias-variance trade-off, appears to be at odds with the observed behavior of methods used in the modern machine learning practice.*
> — Mikhail Belkin et al. (2019)

La précédente section serait-elle fausse ? En pratique elle est correcte sur son domaine de prédiction : la borne n'existe pas quand on dépasse le point d'interpolation. Ainsi, nous n'avons plus de garantie théorique passé ce point, et nous avons *parfois* ce genre de phénomène. Il n'est pas observé pour tous les algorithmes, ni à chaque datasets testés. Ce phénomène est nommé **Grokking** ou encore plus largement **Double descent**.

G.2 Double Descente : un challenge théorique

Mikhail Belkin, Daniel Hsu, Siyuan Ma et Soumik Mandal publient *Reconciling modern machine-learning practice and the classical bias-variance trade-off* [BHMM19] en 2019 où est présenté le concept de double descente, d'où est tiré la précédente citation. L'article a pour but de proposer une explication qui dépasse nos analyses statistiques classiques pour entrer dans l'ère moderne des modèles sur-paramétrés. On y trouve une visualisation graphique de la double descente :

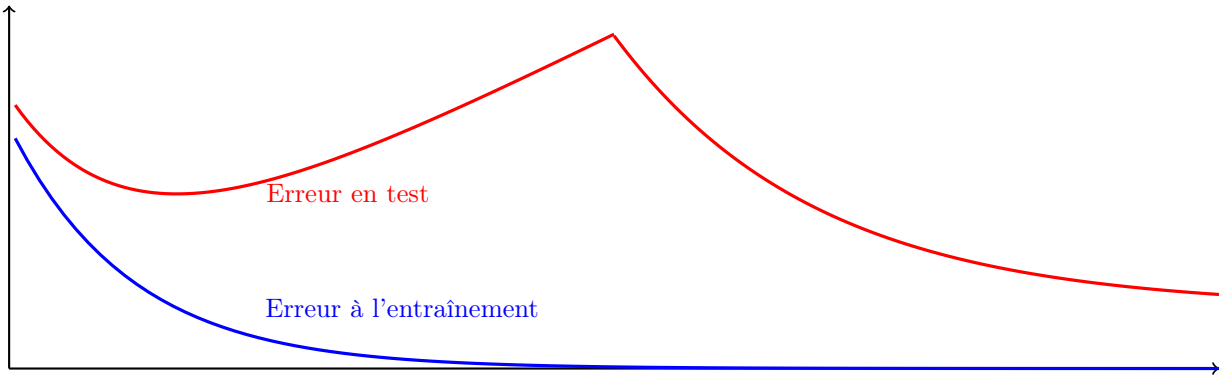


FIGURE G.3 – Courbe d'erreur empirique

Cette courbe de double descente était déjà présente dans l'article *Explaining the success of AdaBoost and Random Forests as interpolating classifiers* [WOBM17] de Abraham J. Wyner, Matthew Oslan, Justin Bleich et David Mease en 2017. Si nous savions déjà qu'AdaBoost pouvait présenter cette courbe spécifique, c'est nouveau pour la Random Forest.

Dans tous les cas, il est nécessaire d'atteindre ce régime d'interpolation et d'aller bien au-delà en terme de capacité du modèle. Cette question fait partie intégrante de la thèse de Preetum Nakkiran *Towards an Empirical Theory of Deep Learning* [Nak21] (2021). Guidée par la question suivante :

> *How does what we* **do** *affect what we* **get** *?*
> — Preetum Nakkiran (2021)

Ses travaux montrent que : *The relationship between what we do and what we get is not always a continuous and well-behaved one* [Nak21] notamment avec l'article dédié à la double descente *Deep double descent : where bigger models and more data hurt* co-écrit avec Gal Kaplun, Yamini Bansal, Tristan Yang,

Boaz Barak et Ilya Sutskever en 2021. Il y est notamment montré combien les données en entrée peuvent impacter la performance d'apprentissage et donc également l'impact de la double descente. L'article est surtout écrit pour un algorithme qui n'est pas traité en cours (réseau de neurones) mais ses résultats restent vrais pour l'ensemble des algorithmes qui exhibe ce phénomène.
Une autre question traitée dans l'article est la définition de complexité. Nous avons commencé l'annexe en présentant la complexité au sens de la dimension VC, mais ce n'est pas la seule manière de faire. Nous pouvons emprunter à l'informatique théorique la complexité au sens de Rademacher par exemple. Les auteurs proposent une nouvelle manière de définir la complexité qui permet d'expliquer la double descente. Nous ne la présenterons pas ici, mais nous la mentionnons pour faire un lien avec la tentative de définition des concepts d'interpolation et d'extrapolation relatés dans l'annexe sur le fléau de la dimension (D).
Nous sommes dans une période qui commence à exploiter la force d'expérimentation qui nous a guidé jusqu'ici pour mieux comprendre ce que nous observons.

G.3 Comment exploiter la double descente ?

Concrètement à ce jour, nous ne savons pas parfaitement si nous allons pouvoir tirer profit d'une double descente avant de la tester. Ainsi, puisque c'est une opération qui peut coûter cher en temps de calcul et donc financièrement également, nous n'avons pas toujours la possibilité de tester. De plus, le modèle généré est par construction un modèle qui sera suffisamment sur-paramétré pour exhiber cette double descente donc ne sera pas exploitable par tous, comme les LLM (Large Language Models) en traitement du langage [2].

En 2020, après les travaux de Belkin qui introduisent formellement la double descente, est publié l'article *Do we need zero training loss after achieving zero training error ?* écrit par Takashi Ishida, Ikko Yamane, Tomoya Sakai, Gang Niu et Masashi Sugiyama [IYS+20]. Puisque nous allons systématiquement atteindre le régime d'interpolation, nous n'allons plus faire d'erreurs sur le dataset de train et par conséquent avoir une fonction de perte égale à 0. Le titre de l'article se demande s'il est nécessaire d'avoir une fonction de perte égale à zéro, puisqu'il est obligatoire de ne pas faire d'erreur. En modifiant la fonction de perte, nous pouvons tout à fait tendre vers le régime d'interpolation et le dépasser tout en ayant une fonction de perte non nulle.

Exercice G.2 (Flooding loss)**.** *Nous notons $\mathcal{L} : \mathbb{R}^d \mapsto \mathbb{R}_+$ une fonction de perte et notons $\tilde{\mathcal{L}}_b$ la flooding loss associée à $\mathcal{L}$ définie par :*

$$\tilde{\mathcal{L}}_b(\theta) = \mid \mathcal{L}(\theta) - b \mid + b \qquad \text{(Flooding loss)}$$

Rappelons qu'une fonction de perte est exploitée pour trouver le meilleur paramètre pour un problème de machine learning supervisé, donc répond au problème :

$$\theta^* = \arg\min_{\theta \in \mathbb{R}^d} \mathcal{L}(\theta)$$

1. *Calculer le gradient de la fonction $\tilde{\mathcal{L}}_b$.*
2. *Puisque nous approchons θ^* par une descente de gradient, analyser la direction des gradients de $\tilde{\mathcal{L}}_b$ par rapport à ceux de $\mathcal{L}$.*
3. *Pour mieux visualiser, nous prenons le cas où $\mathcal{L}(\theta) = (\theta - 1)^2$, donc $d = 1$ et clairement $\theta^* = 1$. Dessiner la fonction $\tilde{\mathcal{L}}_b$.*
4. *Va-t-on obtenir exactement le même θ^* si l'on utilise $\mathcal{L}$ ou si l'on utilise $\tilde{\mathcal{L}}_b$?*

Solution. Nous reprenons les notations de l'exercice ainsi que celles du cours.

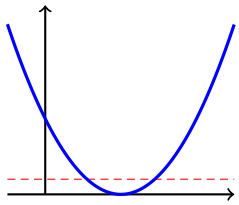
2. Ils s'appuient sur des structures d'algorithmes qui atteignent les centaines de milliards de paramètres.

1. Par les règles habituelles de dérivations, on a :

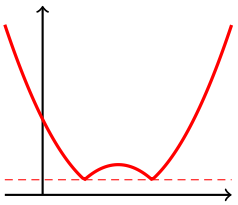
$$\nabla\tilde{\mathcal{L}}_b(\theta) = \nabla\mathcal{L}(\theta) \times \begin{cases} 1 & \text{si } \mathcal{L}(\theta) \geqslant b \\ -1 & \text{si } \mathcal{L}(\theta) < b \end{cases}$$

2. Avec la question précédente, on comprend que tant que la valeur de la fonction de perte est supérieure à la valeur seuil b, les deux fonctions de perte $\mathcal{L}$ et $\tilde{\mathcal{L}}_b$ ont les mêmes gradients dans le même sens. En revanche, dès que la valeur de la fonction de perte $\mathcal{L}$ est inférieure à ce même seuil b, les gradients sont opposés. Nous comprenons donc que la fonction de perte $\tilde{\mathcal{L}}_b$ va osciller autour de la valeur b à partir du moment où elle atteindra cette valeur.

3. Pour $\mathcal{L}(\theta) = (\theta - 1)^2$, on a :



(a) Fonction de perte initiale $\mathcal{L}$



(b) Fonction de perte $\tilde{\mathcal{L}}_b$ pour $b = 0.2$

FIGURE G.4 – Illustration de la *flooding loss* pour $\mathcal{L}(\theta) = (\theta - 1)^2$

4. Avec la visualisation précédente, nous voyons bien que nous n'aurons pas exactement la même valeur pour θ^*, nous aurons un point moins optimal.

□

Il est montré dans l'article que l'on peut obtenir la courbe de double descente beaucoup plus tôt que ce que nous avons pu observer jusqu'ici. Ainsi, nous pouvons avoir des modèles moins complexes plus rapidement entraînés.

Ce papier étant encore récent à l'heure où ce cours est écrit, et le phénomène étant encore mal compris par la communauté, nous avons décidé de présenter tout de même ces résultats pour aiguiller les étudiants sur une piste de recherche prometteuse. Egalement pour montrer l'importance de toujours se poser la question :

Et si tout était faux ?

Bibliographie

[ABKS99] Mihael Ankerst, Markus M Breunig, Hans-Peter Kriegel, and Jörg Sander. Optics : Ordering points to identify the clustering structure. *ACM Sigmod record*, 1999.

[Ach03] Dimitris Achlioptas. Database-friendly random projections : Johnson-lindenstrauss with binary coins. *Journal of Computer and System Sciences*, 2003.

[AV06] David Arthur and Sergei Vassilvitskii. k-means++ : The advantages of careful seeding. Technical report, Standford, 2006.

[BFOS84] Leo Breiman, Jerome H Friedman, Richard A Olshen, and Charles J Stone. Classification and regression trees. *Machine learning*, 1984.

[BG03] Yoshua Bengio and Yves Grandvalet. No unbiased estimator of the variance of k-fold cross-validation. *Advances in Neural Information Processing Systems*, 2003.

[BHMM19] Mikhail Belkin, Daniel Hsu, Siyuan Ma, and Soumik Mandal. Reconciling modern machine-learning practice and the classical bias-variance trade-off. *Proceedings of the National Academy of Sciences*, 2019.

[Boh13] Niels Bohr. On the constitution of atoms and molecules. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 1913.

[BPL21] Randall Balestriero, Jerome Pesenti, and Yann LeCun. Learning in high dimension always amounts to extrapolation. *arXiv preprint arXiv :2110.09485*, 2021.

[Bre96a] Leo Breiman. Bagging predictors. *Machine learning*, 1996.

[Bre96b] Leo Breiman. Stacked regressions. *Machine learning*, 1996.

[Bre01] Leo Breiman. Random forests. *Machine learning*, 2001.

[Bro97] Andrei Z Broder. On the resemblance and containment of documents. In *Proceedings. Compression and Complexity of SEQUENCES 1997 (Cat. No. 97TB100171)*, 1997.

[CG16] Tianqi Chen and Carlos Guestrin. Xgboost : A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, 2016.

[CIJ+22] Nicholas Carlini, Daphne Ippolito, Matthew Jagielski, Katherine Lee, Florian Tramer, and Chiyuan Zhang. Quantifying memorization across neural language models. *arXiv preprint arXiv :2202.07646*, February 2022.

[Com23] Together Computer. Redpajama : an open dataset for training large language models. 2023.

[DCLT18] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert : Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv :1810.04805*, October 2018.

[DEG18] Anna Veronika Dorogush, Vasily Ershov, and Andrey Gulin. Catboost : gradient boosting with categorical features support. *arXiv preprint arXiv :1810.11363*, 2018.

[DG06] Jesse Davis and Mark Goadrich. The relationship between precision-recall and roc curves. In *Proceedings of the 23rd international conference on Machine learning*, 2006.

[DM91] R López De Mántaras. A distance-based attribute selection measure for decision tree induction. *Machine learning*, 1991.

[DMK14] Jelani Nelson Daniel M. Kane. Sparser johnson-lindenstrauss transforms. Arxiv, February 2014.

[Dom99] Pedro Domingos. The role of occam's razor in knowledge discovery. *Data mining and knowledge discovery*, 1999.

[EKS+96] Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In *kdd*, 1996.

[FK15] Peter Flach and Meelis Kull. Precision-recall-gain curves : Pr analysis done right. *Advances in neural information processing systems*, 2015.

[FLD18] Angela Fan, Mike Lewis, and Yann Dauphin. Hierarchical neural story generation. *arXiv preprint arXiv :1805.04833*, May 2018.

[FS97] Yoav Freund and Robert E Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of computer and system sciences*, 1997.

[Gag94] Philip Gage. A new algorithm for data compression. *The C Users Journal*, 1994.

[GEW06] Pierre Geurts, Damien Ernst, and Louis Wehenkel. Extremely randomized trees. *Machine learning*, 2006.

[GZA+23] Suriya Gunasekar, Yi Zhang, Jyoti Aneja, Caio César Teodoro Mendes, Allie Del Giorno, Sivakanth Gopi, Mojan Javaheripi, Piero Kauffmann, Gustavo de Rosa, Olli Saarikivi, et al. Textbooks are all you need. *arXiv preprint arXiv :2306.11644*, June 2023.

[HBD+19] Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes, and Yejin Choi. The curious case of neural text degeneration. *arXiv preprint arXiv :1904.09751*, April 2019.

[HBM+22] Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, et al. Training compute-optimal large language models. *arXiv preprint arXiv :2203.15556*, Mars 2022.

[HHM11] Wolfgang Karl Härdle, Linda Hoffmann, and Rouslan Moro. Learning machines supporting bankruptcy prediction. In *Statistical Tools for Finance and Insurance*. Springer, 2011.

[IG19] Bulat Ibragimov and Gleb Gusev. Minimal variance sampling in stochastic gradient boosting. *Advances in Neural Information Processing Systems*, 2019.

[IYS+20] Takashi Ishida, Ikko Yamane, Tomoya Sakai, Gang Niu, and Masashi Sugiyama. Do we need zero training loss after achieving zero training error ? *arXiv preprint arXiv :2002.08709*, 2020.

[JGBM16] Armand Joulin, Edouard Grave, Piotr Bojanowski, and Tomas Mikolov. Bag of tricks for efficient text classification. *arXiv preprint arXiv :1607.01759*, July 2016.

[KAS+24] Tanishq Kumar, Zachary Ankner, Benjamin F Spector, Blake Bordelon, Niklas Muennighoff, Mansheej Paul, Cengiz Pehlevan, Christopher Ré, and Aditi Raghunathan. Scaling laws for precision. *arXiv preprint arXiv :2411.04330*, Novembre 2024.

[KMF+17] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. Lightgbm : A highly efficient gradient boosting decision tree. *Advances in neural information processing systems*, 2017.

[KMH+20] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv :2001.08361*, January 2020.

[KR18] Taku Kudo and John Richardson. Sentencepiece : A simple and language independent subword tokenizer and detokenizer for neural text processing. *arXiv preprint arXiv :1808.06226*, August 2018.

[Kud18] Taku Kudo. Subword regularization : Improving neural network translation models with multiple subword candidates. *arXiv preprint arXiv :1804.10959*, April 2018.

[LCL+24] Yang Liu, Jiahuan Cao, Chongyu Liu, Kai Ding, and Lianwen Jin. Datasets for large language models : A comprehensive survey. *arXiv preprint arXiv :2402.18041*, February 2024.

[LIN+21] Katherine Lee, Daphne Ippolito, Andrew Nystrom, Chiyuan Zhang, Douglas Eck, Chris Callison-Burch, and Nicholas Carlini. Deduplicating training data makes language models better. *arXiv preprint arXiv :2107.06499*, July 2021.

[LR76] Hyafil Laurent and Ronald L Rivest. Constructing optimal binary decision trees is np-complete. *Information processing letters*, 1976.

[LT24] Omkar Pangarkar Xuezhi Liang Zhen Wang Li An Bhaskar Rao Linghao Jin Huijuan Wang Zhoujun Cheng Suqi Sun Cun Mu Victor Miller Xuezhe Ma Yue Peng Zhengzhong Liu Eric P. Xing Liping Tang, Nikhil Ranjan. Txt360 : A top-quality llm pre-training dataset requires the perfect blend. Hugging Face Space, October 2024.

[MHM18] Leland McInnes, John Healy, and James Melville. Umap : Uniform manifold approximation and projection for dimension reduction. *arXiv preprint arXiv :1802.03426*, 2018.

[MP69] Marvin Minsky and Seymour Papert. Perceptrons. *MIT press*, 1969.

[Nak21] Preetum Nakkiran. *Towards an Empirical Theory of Deep Learning*. PhD thesis, Harvard University, 2021.

[PKvWW24] Guilherme Penedo, Hynek Kydlíček, Leandro von Werra, and Thomas Wolf. Fineweb. 2024.

[PMH+23] Guilherme Penedo, Quentin Malartic, Daniel Hesslow, Ruxandra Cojocaru, Alessandro Cappelli, Hamza Alobeidli, Baptiste Pannier, Ebtesam Almazrouei, and Julien Launay. The refinedweb dataset for falcon llm : outperforming curated corpora with web data, and web data only. *arXiv preprint arXiv :2306.01116*, June 2023.

[PS24] Tim Pearce and Jinyeop Song. Reconciling kaplan and chinchilla scaling laws. *arXiv preprint arXiv :2406.12907*, September 2024.

[Qui86] J. Ross Quinlan. Induction of decision trees. *Machine learning*, 1986.

[Qui96] J. Ross Quinlan. Improved use of continuous attributes in c4.5. *Journal of artificial intelligence research*, 1996.

[RBC+21] Jack W Rae, Sebastian Borgeaud, Trevor Cai, Katie Millican, Jordan Hoffmann, Francis Song, John Aslanides, Sarah Henderson, Roman Ring, Susannah Young, et al. Scaling language models : Methods, analysis & insights from training gopher. *arXiv preprint arXiv :2112.11446*, December 2021.

[Rou87] Peter J Rousseeuw. Silhouettes : a graphical aid to the interpretation and validation of cluster analysis. *Journal of computational and applied mathematics*, 20 :53–65, 1987.

[RSR+19] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of machine learning research*, October 2019.

[RWC+19] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, February 2019.

[Sch13] Robert E Schapire. Explaining adaboost. In *Empirical inference*. Springer, 2013.

[SHB15] Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units. *arXiv preprint arXiv :1508.07909*, August 2015.

[SN12] Mike Schuster and Kaisuke Nakajima. Japanese and korean voice search. In *2012 IEEE international conference on acoustics, speech and signal processing (ICASSP)*, 2012.

[Tea24] Qwen Team. Qwen2.5 : A party of foundation models. https ://qwenlm.github.io/blog/qwen2.5/, September 2024.

[Tib96] Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society : Series B (Methodological)*, Tibshirani, Robert, 1996.

[Tib11] Robert Tibshirani. Regression shrinkage and selection via the lasso : a retrospective. *Journal of the Royal Statistical Society : Series B (Methodological)*, 2011.

[Tur50] Alan M Turing. Computing machinery and intelligence. *Mind*, 1950.

[Vap99] Vladimir Vapnik. *The nature of statistical learning*. Springer science & business media, 1999.

[VdLPH07] Mark J Van der Laan, Eric C Polley, and Alan E Hubbard. Super learner. *Statistical applications in genetics and molecular biology*, 2007.

[VSP+17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in Neural Information Processing Systems*, 2017.

[WBJ84] Joram Lindenstrauss William B. Johnson. Extension of lipschitz mapping into a hilbert space. *Contemporary Mathematics*, 1984.

[WOBM17] Abraham J Wyner, Matthew Olson, Justin Bleich, and David Mease. Explaining the success of adaboost and random forests as interpolating classifiers. *The Journal of Machine Learning Research*, 2017.

[Wol92] David H Wolpert. Stacked generalization. *Neural networks*, 1992.

[WSC+16] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. Google's neural machine translation system : Bridging the gap between human and machine translation. *arXiv preprint arXiv :1609.08144*, September 2016.

[ZH05] Hui Zou and Trevor Hastie. Regularization and variable selection via the elastic net. *Journal of the royal statistical society : series B (statistical methodology)*, 2005.

[ZLX+23] Chunting Zhou, Pengfei Liu, Puxin Xu, Srini Iyer, Jiao Sun, Yuning Mao, Xuezhe Ma, Avia Efrat, Ping Yu, Lili Yu, et al. Lima : Less is more for alignment. *arXiv preprint arXiv :2305.11206*, May 2023.