
INTRODUCTION AU MACHINE LEARNING

2022-2023
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.

Un étudiant n'est pas un sac qu'on remplit mais une bougie qu'on enflamme.
— Vladimir Arnold (1972)

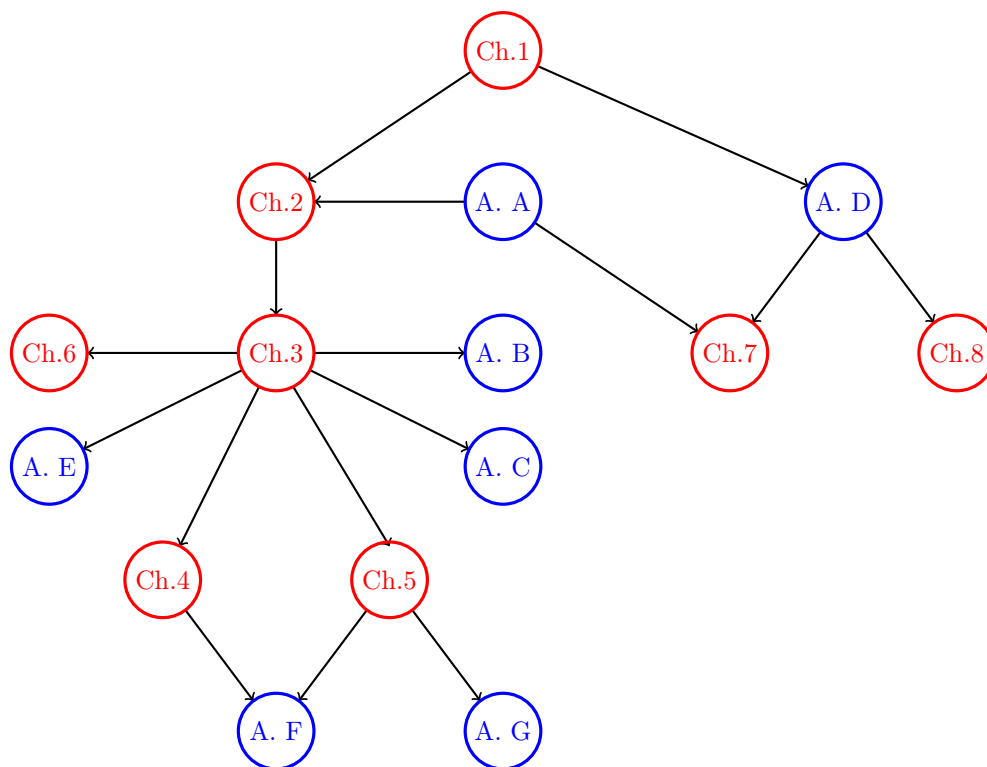


Table des matières

1	Introduction au Machine Learning	7
1.1	Les différentes approches du Machine Learning	7
1.2	Apprentissage supervisé - plus formellement	8
1.3	Comment sélectionner les modèles ?	10
1.3.1	Train, Validation, Test	10
1.3.2	Cross-validation	11
1.3.3	Occam's Razor	12
2	Régression linéaire et variantes	14
2.1	Régression linéaire classique	14
2.2	Mesurer la performance d'une régression	17
2.2.1	Erreur quadratique moyenne	17
2.2.2	Coefficient de détermination R^2	18
2.3	Régressions pénalisées	19
2.3.1	Régression Ridge	20
2.3.2	Régression LASSO	20
2.3.3	Régression ElasticNet	21
2.4	Régressions polynomiales : approximateurs universels	21
3	Régression Logistique	24
3.1	Modélisation	24
3.2	Descente de gradient	26
3.3	Mesurer la performance d'une classification	29
3.3.1	Pour un seuil fixé	29
3.3.2	Sans choix de seuil	32
4	Arbre et Random Forest	34
4.1	Arbre de décision	34
4.1.1	Meilleure partition	35
4.1.2	Critères d'arrêt	38
4.2	Méthodes ensemblistes	39
4.2.1	Bagging	39
4.2.2	Random Forest	41
5	Boosting	43
5.1	Algorithme AdaBoost	43
5.2	Gradient Boosting	45
5.2.1	Principaux algorithmes de Gradient Boosting	48
6	Support Vector Machine	52
6.1	Intuition	52
6.2	Formalisation du problème	53
6.2.1	Dans le cas séparable	53
6.2.2	Dans le cas non-séparable	55
6.2.3	Problème primal et dual	56

6.2.4	Kernel trick	60
6.3	L'algorithme en pratique	62
6.3.1	Noyaux classiques	62
6.3.2	Fine-tuning	62
7	Clustering	64
7.1	Distance : ce qui se ressemble est proche	64
7.2	Approche statistique	65
7.2.1	K-Means	65
7.2.2	Kmeans++ : un meilleur départ	67
7.3	Approche par densité	68
7.3.1	DBSCAN	70
7.3.2	OPTICS	70
7.4	Mesures de performance	73
7.4.1	Silhouette score	73
7.4.2	L'index de Calinski-Harabasz	74
8	Réduction de dimension	76
8.1	Lemme de Johnson-Lindenstrauss	76
8.2	Analyse par composantes principales	79
8.2.1	Diagonalisation d'une matrice	79
8.2.2	Application à la réduction de dimensions	82
8.3	UMAP	84
8.3.1	Formation du graphe en grande dimension	84
8.3.2	Réduction du graphe	86
8.3.3	Utilisation en pratique d'UMAP	87
A	Rappel et utilisation de la convexité pour le Machine Learning	89
A.1	Définition et propriétés	89
A.1.1	Ensemble et fonction convexe	89
A.1.2	Caractérisation du premier et deuxième ordre	90
A.2	Résultats d'optimisations	92
A.3	Vitesse de convergence pour la descente de gradient	94
A.3.1	Pour une fonction Lipschitzienne	94
A.3.2	Fonction β -smooth	95
B	Algorithme du Perceptron	97
B.1	Description	97
B.2	Preuve de convergence	99
B.3	Bonus : utilisation de l'inégalité de Cauchy-Schwarz	100
C	Algorithme Naïve Bayes	101
C.1	Probabilité conditionnelle et théorème de Bayes	101
C.2	Application en Machine Learning	103
D	Fléau de la dimension	105
D.1	Volume d'une hypersphère	105
D.2	Orthogonalité à la surface d'une hypersphère	108
D.3	Interpolation et extrapolation	109
E	Accélération de Nesterov	110
E.1	Etude analytique	110
E.2	Etude via son équation différentielle	110
F	Stacking et Blending : au-delà du Bagging et Boosting	111

G Double descent : vers le Machine Learning moderne	113
G.1 Prédiction de la théorie de Vapnik-Chervonenkis	113
G.1.1 Cas d'AdaBoost	115
G.2 Double Descente : un challenge théorique	116
G.3 Comment exploiter la double descente?	117

Chapitre 1

Introduction au Machine Learning

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 des 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, mais surtout au niveau académique, une troisième approche : l'apprentissage par renforcement, qui nécessiterait un cours dédié.

Au sein de ces deux grandes approches se trouvent des sous catégories :

- Apprentissage supervisé
 - **Régression** : prédire une valeur continue
 - **Classification** : prédire une classe, une valeur discrète
- Apprentissage non-supervisé
 - **Clustering** : rassembler des observations qui se ressemblent
 - **Réduction de dimension** : 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. Exercice

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 différenciants 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 ¹.
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.

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 :

$$\mathcal{D} = \left\{ (x^{(i)}, y_i) \mid \forall i \leq n, x^{(i)} \in \mathbb{R}^d, y_i \in \mathcal{Y} \right\} \quad (1.1)$$

Nombre d'observations Nombre d'informations

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 $(y_i)_{i \leq n} \in \mathcal{Y}^n$

L'hypothèse du data-scientist est qu'il existe une certaine fonction inconnue f qui fait le lien entre les observations $x \in \mathbb{R}^d$ et la réponse y . Bien sûr, **aucun modèle n'est parfait**, et donc il y a un terme d'erreur incompressible² qui est dû à des informations que l'on a pas à disposition par exemple. Formellement, on peut résumer cela à :

1. À condition que l'algorithme soit performant.
2. Voir l'équation (2.3).

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

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 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{L} . 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 :

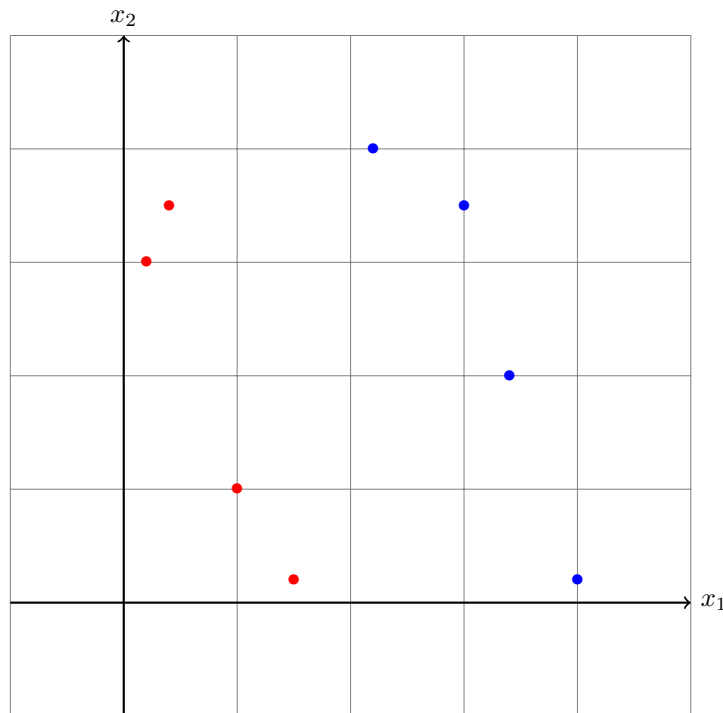


FIGURE 1.1 – Exemple d'un problème de classification avec deux classes et deux indicateurs pour identifier la classe

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

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

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 classer parfaitement les points avec $\theta = 2$, la fonction de perte est nulle !

Il y a bien sûr 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. \square

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 biais³.

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

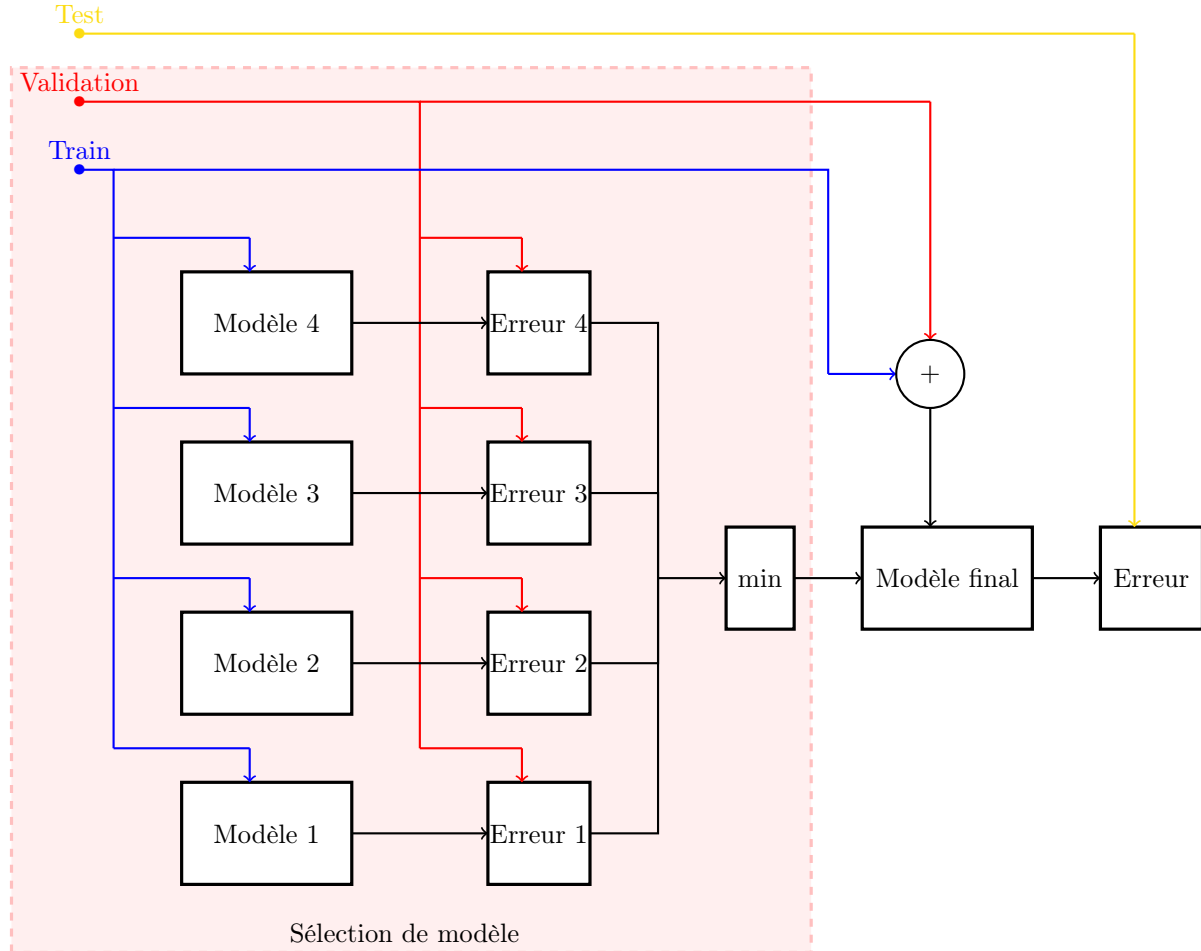


FIGURE 1.2 – Méthodologie Train - Validation - Test

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 Cross-validation

La cross validation revient à découper à nouveau le dataset de *train* pour apprendre de manière itérative 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 parties égales et on va apprendre K fois les meilleurs paramètres et donc mesurer K fois la performance de l'algorithme. La structure d'entraînement est la suivante :

A chaque pli, on extrait du dataset 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

3. 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 1.3 – Cross-validation avec $K = 4$ plis

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 dataset.

On peut se demander s'il existe un nombre K optimal pour estimer sans biais l'erreur produite. Yoshua Bengio et Yves Grandvalet montrent en 2003 dans l'article *No unbiased estimator of the variance of k-fold cross-validation* [BG03] 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, Pedro Domingos note dans son article de 1999 *The role of Occam's razor in knowledge discovery* [Dom99] qu'il y a généralement deux interprétations différentes, qu'il nomme comme deux rasoirs différents :

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 aux travaux de Domingos ainsi qu'à la propre expérience du lecteur pour approfondir la question.

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 :

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

2.1 Régression linéaire classique

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

$$\theta^* = \arg \min_{\theta \in \mathbb{R}^d} \sum_{i=1}^n \left(\underset{\text{Vraie valeur}}{y_i} - \overset{\text{Valeur prédite}}{f_\theta(x^{(i)})} \right)^2 \quad (2.1)$$

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

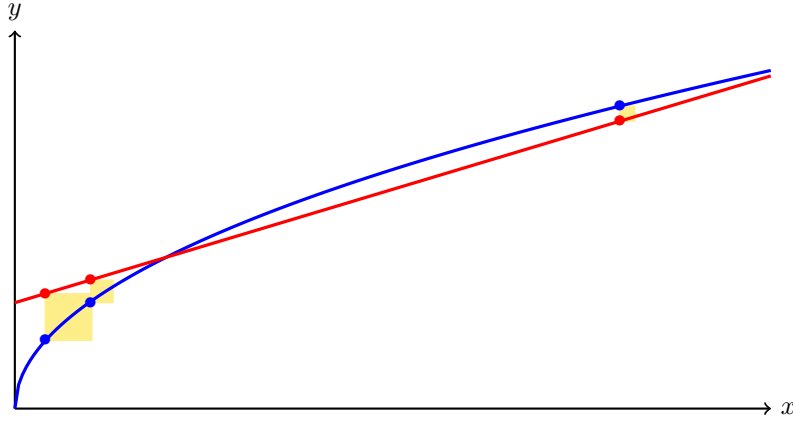


FIGURE 2.1 – Visualisation de la MSE entre la Régression linéaire et vraie courbe

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 :

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

On peut réécrire notre problème (2.1) comme :

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

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 $\mathcal{D} = \{(x^{(i)}, y_i) \mid \forall i \leq n : x^{(i)} \in \mathbb{R}, y_i \in \mathbb{R}\}$. 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 $\bar{x} = \frac{1}{n} \sum_{i=1}^n x^{(i)}$. On rappelle avec cette convention que pour $u, v \in \mathbb{R}^n$:

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

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

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

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 $(y_i)_{i \leq n}$ 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 :

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

\Longleftrightarrow

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

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 :

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

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 :*

$$\theta^* = ({}^t X X)^{-1} {}^t X Y$$

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. \square

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) :

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

L'intérêt de la MSE est qu'elle est convexe¹ 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 :

- 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
- $\mathbb{V} \left[\hat{f}(x) \right] = \mathbb{E} \left[\left(\mathbb{E} \left[\hat{f}(x) \right] - \hat{f}(x) \right)^2 \right]$: 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.

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).

1. Voir annexe (A).

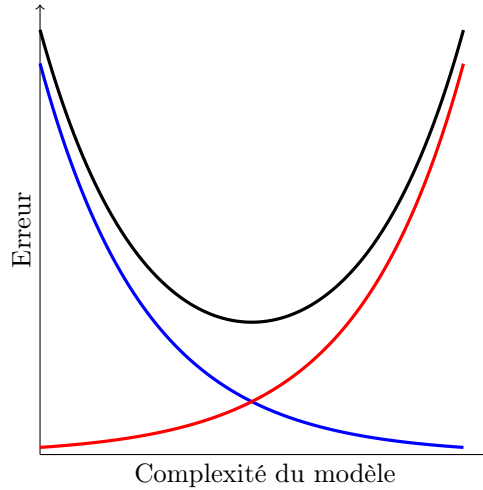


FIGURE 2.2 – Trade-off **biais**-**variance**

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

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 :*

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

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 :

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

↓ Prédiction du modèle
↑ Moyenne de la cible

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

Si $R^2 = 0 \iff \sum_{i=1}^n (y_i - \hat{y}_i)^2 = \sum_{i=1}^n (y_i - \bar{y})^2$ 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 $\forall i \leq n, \hat{y}_i = \bar{y}$.

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 ($d \gg n$). 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 :

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

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.

Les types de régression que l'on présente maintenant correspondent aux travaux de Robert Tibshirani *Regression shrinkage and selection via the LASSO* [Tib96] et également dans le papier rétrospective [Tib11].

2.3.1 Régression Ridge

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

$$\theta_{\text{Ridge}}^* = \arg \min_{\theta \in \mathbb{R}^d} \|Y - X\theta\|^2 + \lambda \|\theta\|^2 \quad (\text{Ridge})$$

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 $\lambda \rightarrow +\infty$.

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 $+\infty$. \square

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 avons 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 8).

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. \square

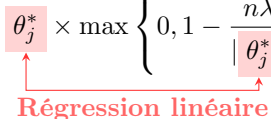
2.3.2 Régression LASSO

La régression LASSO (*Least Absolute Selection and Shrinkage Operator*) est définie par le problème :

$$\theta_{\text{LASSO}}^* = \arg \min_{\theta \in \mathbb{R}^d} \|Y - X\theta\|^2 + \lambda \|\theta\|_1 \quad (\text{LASSO})$$

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 :

$$\forall j \leq d, \quad \theta_{\text{LASSO},j}^* = \theta_j^* \times \max \left\{ 0, 1 - \frac{n\lambda}{|\theta_j^*|} \right\}$$



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 correspond à une combinaison entre Ridge et LASSO :

$$\theta_{\text{ElasticNet}}^* = \arg \min_{\theta \in \mathbb{R}^d} \|Y - X\theta\|^2 + \lambda \|\theta\|_1 + \mu \|\theta\|^2 \quad (\text{ElasticNet})$$

Cette double utilisation des régularisations est présentée par Hui Zou et Trevor Hastie *Regularization and variable selection via the Elastic net* [ZH05] en 2005. 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.

En 2013, Martin Jaggi montre que l'on peut lier la régression LASSO à un autre algorithme que l'on traitera dans le chapitre 6 dans l'article *An equivalence between the LASSO and Support Vector Machines* [Jag13].

De manière similaire en 2015, Quan Zhou and al. ont montré un lien entre l'ElasticNet et les Support Vector Machines dans l'article *A reduction of the ElasticNet to Support Vector Machines with an application to GPU computing* [ZCS⁺15]. Un des grands intérêts de ces liens est de tirer parti de la puissance de calcul qui a été développée pour le second algorithme, pour résoudre plus rapidement les régressions LASSO et ElasticNet.

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 :

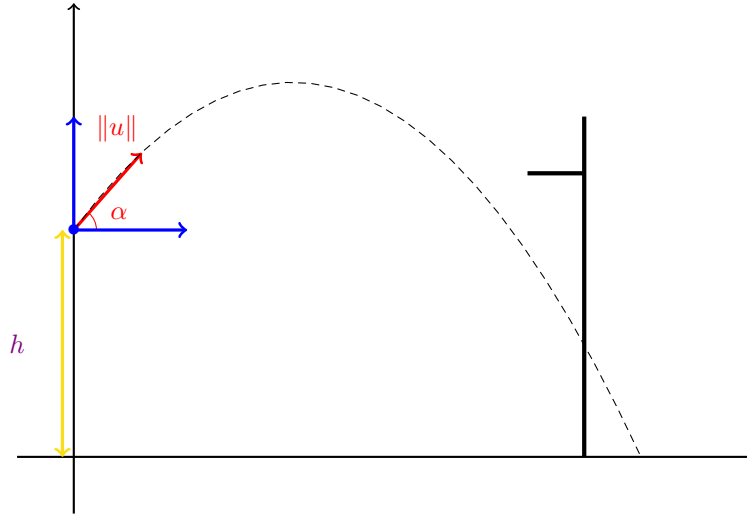


FIGURE 2.3 – Simulation d’un mini-jeu de basketball

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 :

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 :

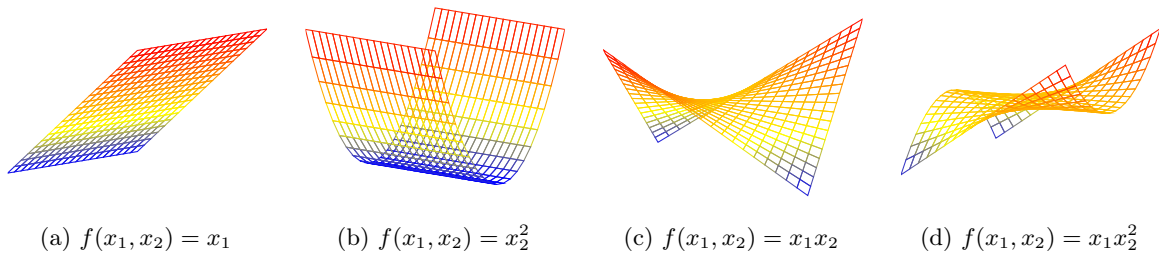


FIGURE 2.4 – Exemple de surfaces pour deux features avec des polynômes de degré au plus 3

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 $(f_i(x) = x^i)_{i \in \mathbb{N}}$. 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 :

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

L'ensemble des valeurs pour les $(y_i)_{i \leq n}$ 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 :

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

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

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

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

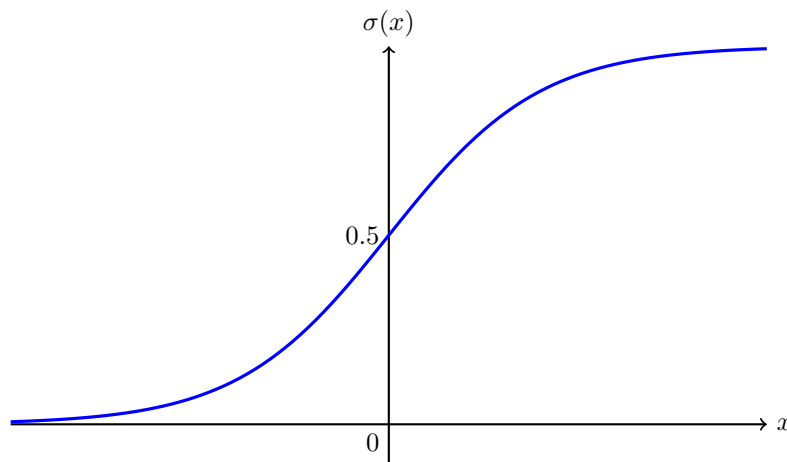


FIGURE 3.1 – Fonction sigmoid σ ou fonction logistique

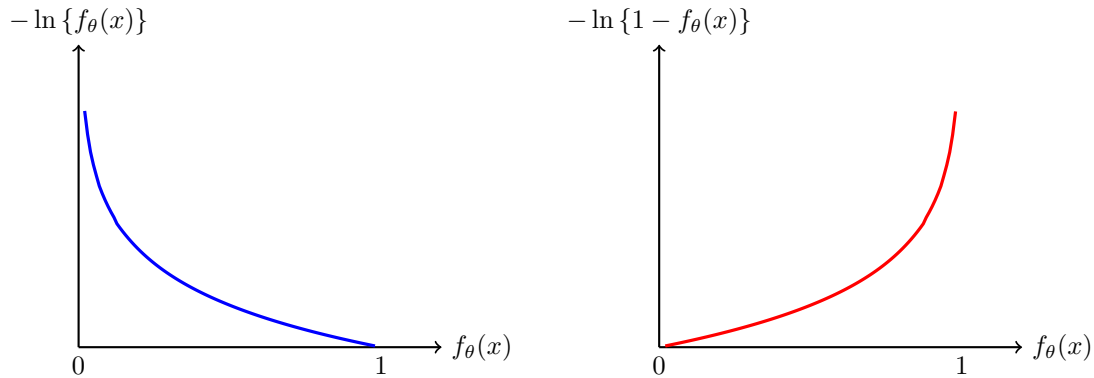
On comprend avec l'équation (3.1) l'interprétation que l'on peut faire des valeurs des coefficients. Si l'on augmente de 1 la valeur de l'information x_j , on augmente le logarithme de la cote par θ_j . C'est équivalent à dire qu'on augmente la cote par e^{θ_j} .

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 :

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

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 :

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

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 :

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

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 :

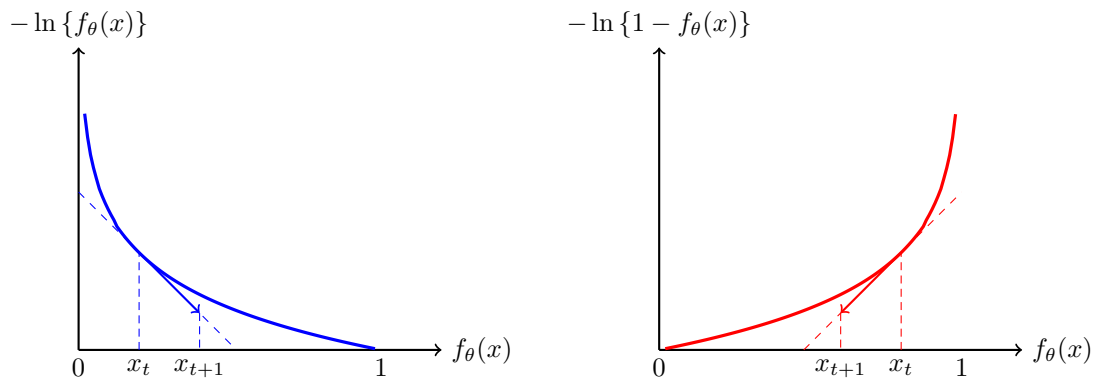
$$x_{t+1} = x_t - \eta \nabla f(x_t) \quad , \eta > 0 \quad (3.3)$$

A chaque itération, nous allons nous déplacer dans la direction opposée à la valeur du gradient.

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. On traite ici d'une surface en deux dimensions dans laquelle on



cherche les coordonnées du point minimal. Dans la figure (3.2) on voit comment la descente se comporte en plaçant un point à chaque itération.

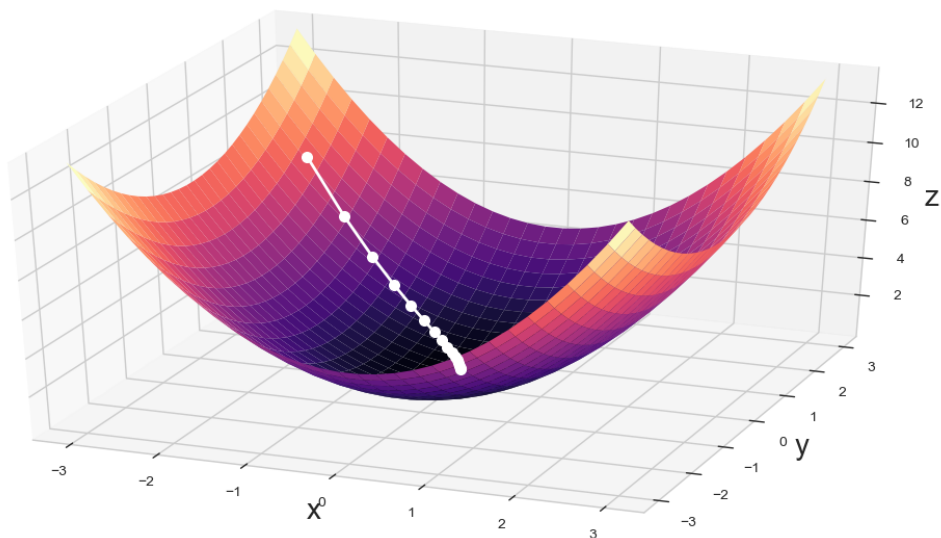


FIGURE 3.2 – Exemple d’une descente de gradient pour la fonction $f(x, y) = x^2 + \frac{1}{2}y^2$ avec $\eta = 0.1$

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 minimum¹. 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 comme (3.3c).

1. S’il existe !

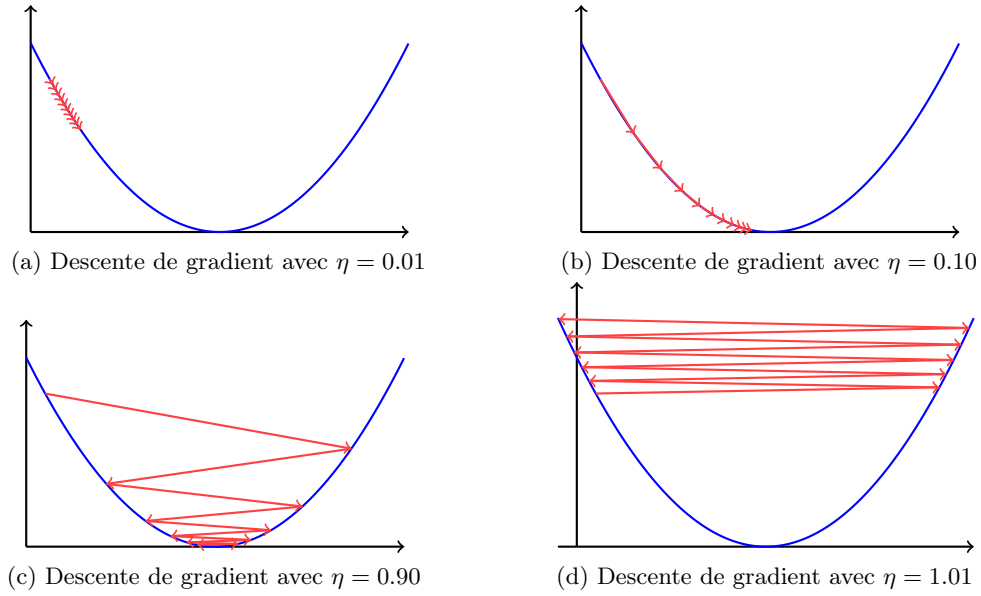


FIGURE 3.3 – Impact du learning rate η sur la descente de gradient

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 :

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

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

Exercice 3.3. On rappelle que $f_\theta(x) = \frac{1}{1 + e^{-\langle \theta, x \rangle}}$. Montrer que :

$$1. f_\theta(x) = \frac{e^{\langle \theta, x \rangle}}{1 + e^{\langle \theta, x \rangle}}$$

$$2. f_\theta(-x) = 1 - f_\theta(x)$$

$$3. \frac{\partial \ln}{\partial \theta_j} (f_\theta(x)) = x_j (1 - f_\theta(x))$$

$$4. \frac{\partial \ln}{\partial \theta_j} (1 - f_\theta(x)) = -x_j f_\theta(x)$$

$$5. \frac{\partial \mathcal{L}}{\partial \theta_j} (\theta; x^{(i)}, y_i) = x_j^{(i)} (f_\theta(x^{(i)}) - y_i)$$

6. Conclure que la descente de gradient pour le problème avec la fonction de coût est :

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

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 classifieur 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 classifieur 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 classifieur 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 :

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

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

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 avons 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 arithmétique², 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.

2. Ce résultat est prouvé dans la section (B.3)

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

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 $(\text{TPR}(t), \text{FPR}(t))$ 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 classifieur qui prédit aléatoirement. À 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 $\text{TPR}(0) = 1$ et $\text{FPR}(0) = 1$. Inversement, si le seuil est 1, alors on prédit toujours 0, donc $\text{TPR}(1) = 0$ et $\text{FPR}(1) = 0$. 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 $\text{TPR}(p) = p$ et $\text{FPR}(p) = p$. Donc la courbe ROC d'un modèle aléatoire est la droite $y = x$. \square

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 $(0, 0) \rightarrow (0, 1) \rightarrow (1, 1)$, 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 l'excellent article *Precision-recall-gain curves : PR analysis done right* de Peter Flach et Meelis Kull [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 l'article de Jesse Davis et Mark Goadrich *The relationship between Precision-Recall and ROC curves* [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 classifieur qui prédit aléatoirement. À 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 précision fixée. \square

Avec cette méthode, il n'existe pas de baseline aléatoire universelle. De plus l'article [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.

Et 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 avons annoncé dans l'introduction : aucun modèle n'est parfait. De même qu'en mathématiques, Kurt Gödel est un logicien qui a réussi 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 le papier *Classification and regression trees* [BFOS84] de Leo Breiman en 1984. Des améliorations de ces arbres sont présentés en 1986 par Quinlan *Induction of decision trees* [Qui86] et en 1996 *Improved use of continuous attributes in C4.5* [Qui96]. Essayons de formaliser l'idée finalement assez intuitive !

On considère un problème de classification avec un dataset $\mathcal{D} = \{(x^{(i)}, y_i) \mid \forall i \leq n, x^{(i)} \in \mathbb{R}^d, y_i \in \{0, 1\}\}$. 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 $j \leq d$, on cherche la meilleure séparation de l'espace sous la forme $x_j \leq \alpha$ 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 avons soulevé la question suivante : comment trouver la meilleure séparation ? Ce problème est illustré dans la figure (4.1).

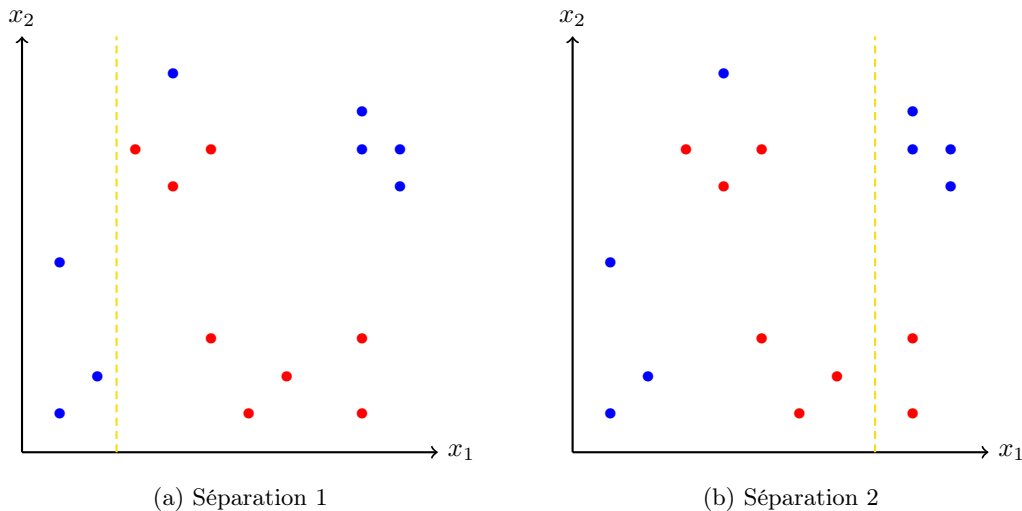


FIGURE 4.1 – Sélection de la meilleure séparation de l'espace pour une information donnée

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 :

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

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

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

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 :

$$\Delta\Phi(\delta, t) = \Phi(t) - \left(\overbrace{p_g}^{\text{Proportion de la classe d'intérêt dans } t_g} \Phi(t_g) + \underbrace{p_d}_{\text{Proportion de la classe d'intérêt dans } t_d} \Phi(t_d) \right)$$

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).

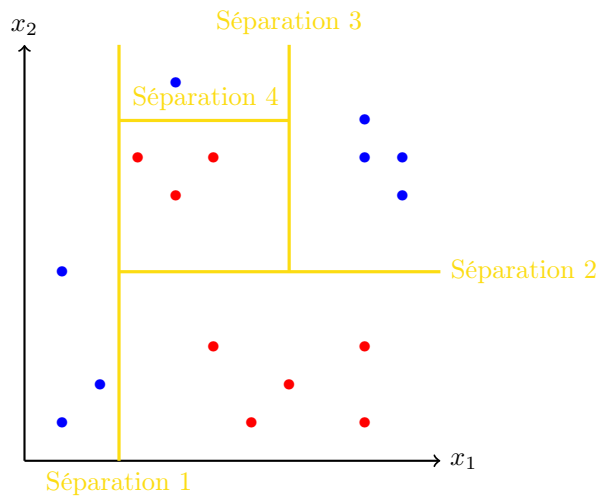


FIGURE 4.2 – Exemple de partitionnement de l'espace

Maintenant que l'on a le partitionnement, on peut facilement résumer l'algorithme sous forme d'arbre également.

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).

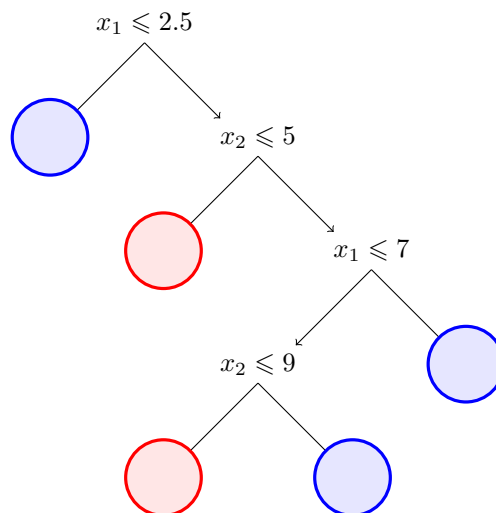


FIGURE 4.3 – Arbre de décision lié au problème de classification (4.1)

□

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. Hyafil Laurent et Ronald Rivest ont montré, par exemple, en 1976 que *Constructing optimal binary decision tree is NP-Complete* [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 à 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 présentes 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. \square

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 ensemble 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. \square

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) :

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

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

Solution. On suppose que les variables $(f_k(x))_{k \leq m}$ 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 $(f_k(x))_{k \leq m}$ 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 $(f_k(x))_{k \leq m}$ 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 $n \rightarrow +\infty$) alors la distribution empirique converge vers la distribution réelle. Ainsi, nous avons un bootstrap très efficace pour exploiter au mieux l'idée d'ensemble.

En pratique, les datasets formés par le bootstrap 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 :

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

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. À 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 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. C'est ce qu'a présenté Leo Breiman en 1996 dans l'article *Bagging predictors* [Bre96a] présentant la technique du bagging. Cinq ans plus tard il publie l'article *Random Forest* [Bre01] lié à cette technique et aux arbres plus spécifiquement.

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). Leo Breiman propose donc d'ajouter encore plus d'aléatoire dans la méthode du bagging.

Il illustre avec 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, étudions la proposition de Pierre Geurts, Damien Ernst et Louis Wehenkel en 2006 dans l'article *Extremely randomized 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-ensembles d'informations sélectionnés. 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
 - `bootstrap` : si vrai, le bootstrap 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 bootstrap

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 potentiel¹. 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 bootstrap (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

Nous sommes capables à partir d'un dataset d'entraîner plusieurs algorithmes de Machine Learning pour les deux tâches principales qui sont la régression et la classification. Nous sommes également capables de mesurer la performance d'un algorithme.

On cherche toujours à obtenir la meilleure performance, donc on organisera des compétitions de modèle, on cherchera de meilleures informations à exploiter et on fera attention au sur-apprentissage. Ceci est 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 constitue un des algorithmes de Machine Learning des plus intéressants et qui illustre parfaitement les grandes idées du Boosting. Nous traiterons ensuite d'une partie du Boosting. Finalement nous introduirons les 3 algorithmes les plus utilisés dans les années 2020 qui exploitent le Gradient Boosting.

L'intérêt de ce chapitre est 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 a été présenté par Yoav Freund et Robert Schapire en 1997 dans l'article *A decision-theoretic generalization of on-line learning and an application to boosting* [FS97]. Ce développement a été récompensé par le prix Gödel¹ 5 ans plus tard.

L'idée principale d'AdaBoost est de noter les observations et les weak learners formés. 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. Ce sont donc des modèles très simplistes, et ce sont le travail successif de notation 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'itérations² que l'on réalisera
- $w_t^{(i)}$ la note comprise entre 0 et 1 à l'itération $t \leq T$ pour l'observation $i \leq n$

1. Le plus prestigieux en informatique après le prix Turing.

2. On parle également d'époques.

- h_θ un weak learner d'AdaBoost paramétré par le vecteur d'information θ

Comme expliqué plus tôt, 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 :

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

Nombre d'époques \downarrow T
Note du weak learner de l'époque t \uparrow α_t

Il suffira de prendre l'indicatrice que ce nombre est positif pour prédire 1 par exemple. Il nous reste à comprendre comment l'on va 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 $i \leq n$ avec le poids $w^{(i)}$ en reprenant les notations précédemment introduites. C'est avec ce système de notation que l'on conduit les weak learners à porter plus d'attention sur certaines observations que d'autres. Le problème que l'on se posera à chaque époque $t \leq T$ 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. Au départ, on définit l'ensemble des notes uniformément.

Une fois le meilleur paramétrage du weak learner trouvé, on calcule l'erreur associée et donc la note :

$$\varepsilon_t = \frac{\sum_{i=1}^n w_t^{(i)} \mathbb{1}_{\{y_i \neq h_{\theta_t}(x^{(i)})\}}}{\sum_{i=1}^n w_t^{(i)}}$$

$$\alpha_t = \frac{1}{2} \ln \left(\frac{1 - \varepsilon_t}{\varepsilon_t} \right)$$

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

Exercice 5.1 (Étude de α_t). Soit $t \leq T$ 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 $w_t^{(i)} \mathbb{1}_{\{y_i \neq h_{\theta_t}(x^{(i)})\}} \leq w_t^{(i)}$ pour tout $i \leq n$, d'où le résultat.

2. On note que 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.

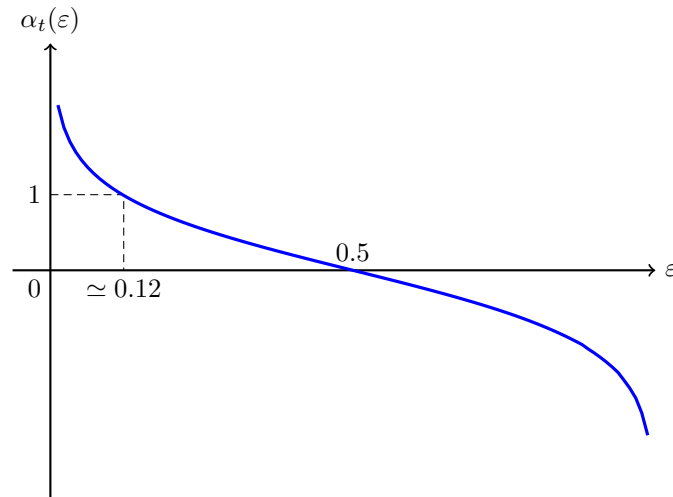


FIGURE 5.1 – Graphe de la fonction : $x \mapsto \frac{1}{2} \ln \left(\frac{1-x}{x} \right)$

Quand ε est proche de 0 c'est l'inverse : le weak learner a réussi à 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 on comprend que les notes des observations peuvent se mettre à jour de la manière suivante :

$$\forall i \leq n, w_{t+1}^{(i)} = w_t^{(i)} e^{-\alpha_t h_{\theta_t}(x^{(i)}) y_i}$$

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*.

Voyons maintenant les vrais stars des hackathons qui exploitent la descente de gradient pour maximiser les performances.

5.2 Gradient Boosting

Pour présenter le Gradient Boosting, considérons le dataset :

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

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 $\gamma^* = \bar{y}$. □

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 :

$$\begin{aligned} \forall i \leq n, f_m(x^{(i)}) &= f_{m-1}(x^{(i)}) + h_m(x^{(i)}) = y_i \\ &\iff \\ \forall i \leq n, h_m(x^{(i)}) &= y_i - f_{m-1}(x^{(i)}) \end{aligned}$$

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 $\mathcal{C}(y, f(x)) = \sum_{i=1}^n \mathcal{L}(y_i, f(x^{(i)}))$. Montrer que :*

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

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 :

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

On reconnaît 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 :

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

Nous avons donc 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 :

$$\begin{aligned}
f_0(x) &= \arg \min_{\gamma \in \mathbb{R}} \sum_{i=1}^n \mathcal{L}(y_i, \gamma) && \text{(Initialisation)} \\
\forall m \leq 1, f_m(x) &= f_{m-1}(x) + \eta \gamma_m h_m(x) && \text{(Itération)} \\
F(x) &= \sum_{m=0}^M \gamma_m h_m(x) \text{ avec } \gamma_0 = 1 && \text{(Strong learner)}
\end{aligned}$$

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

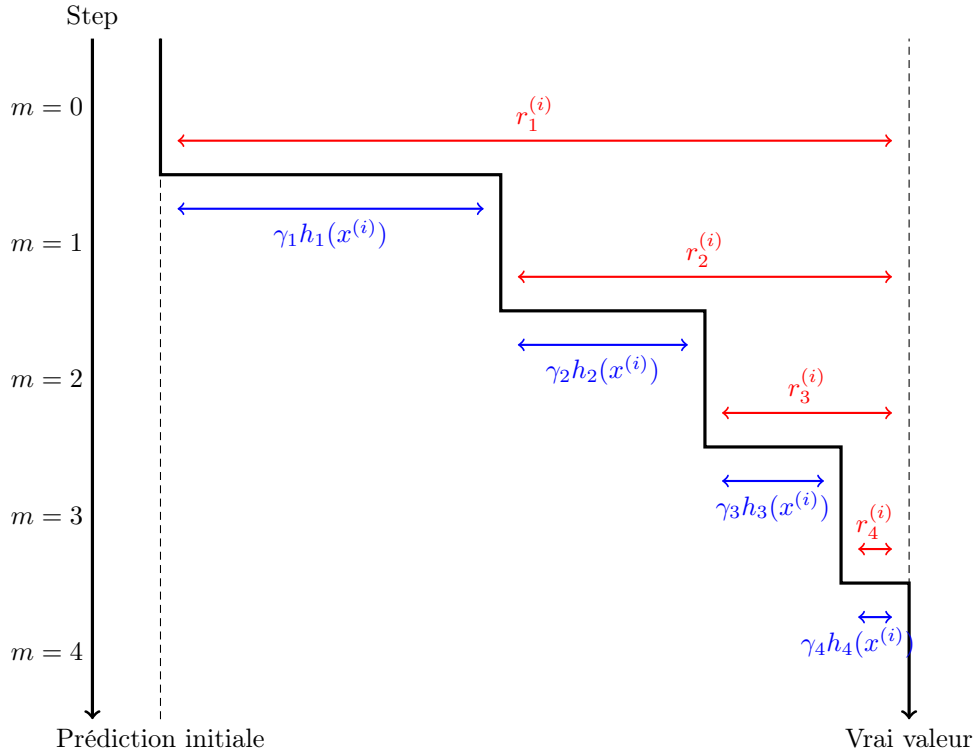


FIGURE 5.2 – Principe du Gradient Boosting pour une observation

On a une réduction des **résidus** à chaque étape grâce à une **correction** adaptative. Ici on a pris $\eta = 1$. 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 est introduit par Tianqi Chen et Carlos Guestrin en 2016 dans la publication *XGBoost : A scalable tree boosting system* [CG16]. Comme indiqué dans le titre, l'algorithme 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 :

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

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 :

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

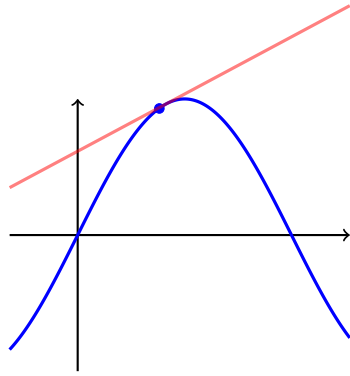
Le problème (5.2) est une approximation du vrai problème (5.1), donc nous n'optimisons pas vraiment ce que l'on souhaite. Mais cela reste une bonne approximation et cela fonctionne tout de même ! L'apport d'XGBoost est d'être plus précis dans l'approximation qui est réalisée, en résolvant un autre problème.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

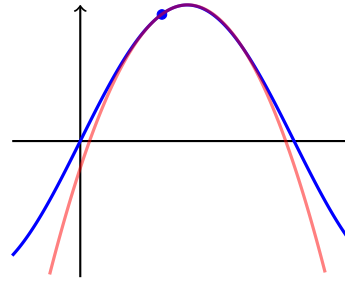
$$f(x) = \sum_{k=0}^n \frac{f^{(k)}(a)}{k!} (x-a)^k + R_n(x)$$

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

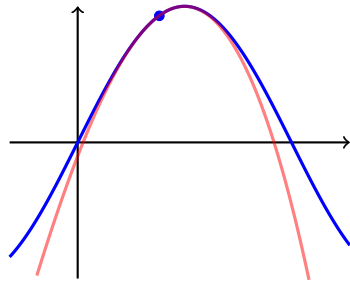
Ce théorème permet d'approcher une fonction au voisinage d'un point avec un polynôme. C'est particulièrement intéressant si nous devons travailler localement avec des fonctions compliquées ou lourde à évaluer. Cette manière d'exprimer une fonction est appelée : un développement de Taylor à l'ordre n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



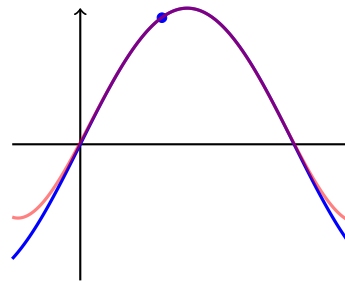
(a) Développement à l'ordre 1



(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 :

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

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

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

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. \square

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, l'article *LightGBM : A highly efficient gradient boosting decision tree* [KMF⁺17] écrit par Guolin Ke et al. introduit un des algorithmes concurrents à XGBoost. LightGBM 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 est le dernier grand algorithme de Boosting à date présenté en 2018 par Anna Veronika Dorogush, Vasily Ershov et Andrey Gulin dans *CatBoost : Gradient Boosting with categorical features support* [DEG18]. 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 l'article *Minimal Variance Sampling in stochastic Gradient Boosting* [IG19].

Pour chacun des algorithmes, les principaux hyperparamètres sont :

- 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 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.

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

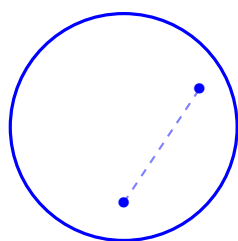
A.1.1 Ensemble et fonction convexe

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 :

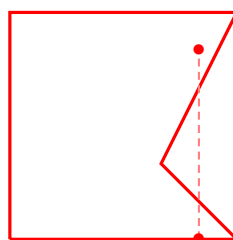
Définition 14. Soit A un sous-ensemble de \mathbb{R}^d . On dit que A est un ensemble convexe si :

$$\forall a, b \in A, \forall t \in [0, 1], ta + (1 - t)b \in A$$

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 15. Soit A un sous-ensemble convexe de \mathbb{R}^d . Une fonction $f : A \mapsto \mathbb{R}$ est convexe si et seulement si :

$$\forall a, b \in A, \forall t \in [0, 1], f(ta + (1 - t)b) \leq tf(a) + (1 - t)f(b)$$

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).

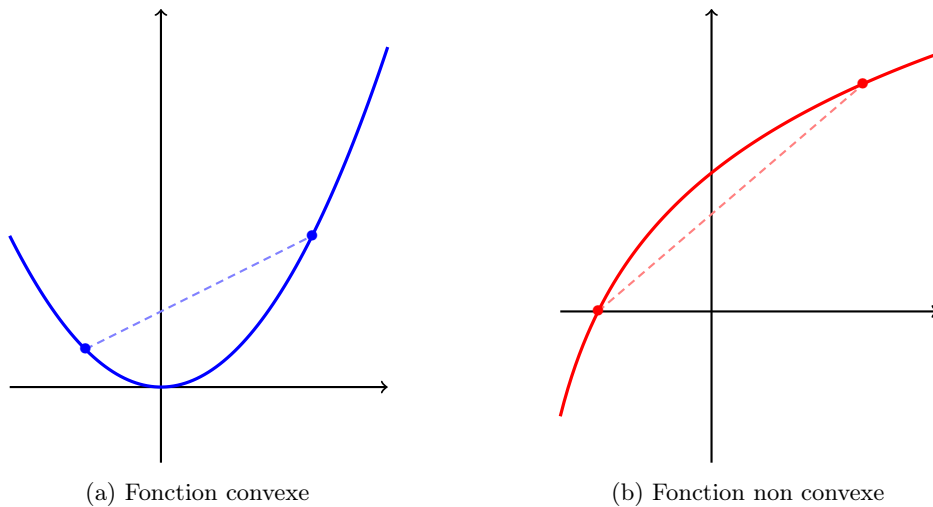


FIGURE A.2 – Exemple et contre exemple de fonction convexe

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 5 (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. $\forall x, y \in A, f(y) \geq f(x) + \nabla f(x)(y - x)$
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 reconnaît 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).

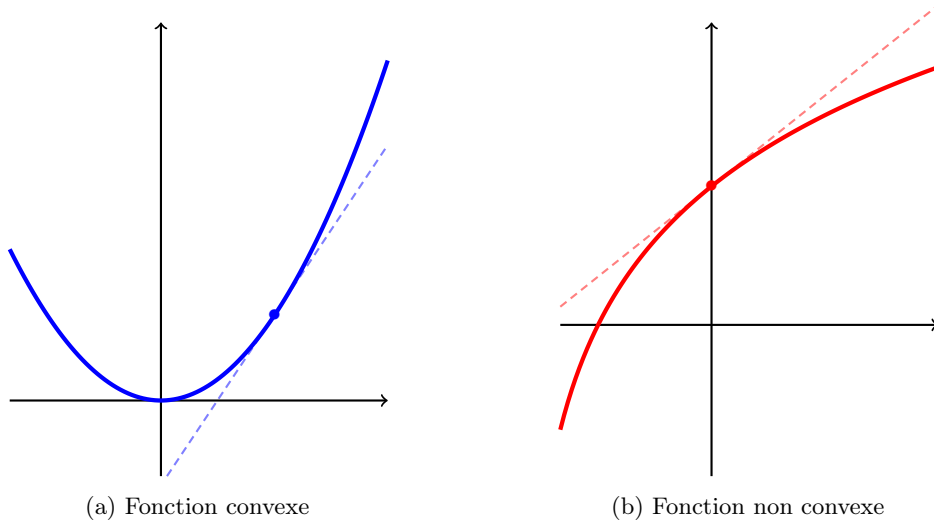


FIGURE A.3 – Illustration de la propriété de sous-estimateur pour une fonction **convexe** et contre exemple pour une fonction **non convexe**

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 (5)

Démonstration. Commençons par montrer que si f est convexe, alors $\forall x, y \in A, f(y) \geq f(x) + \nabla f(x)(y - x)$.

Soit $x, y \in A$, par définition on a que :

$$\begin{aligned} \forall t \in [0, 1], f(tx + (1 - t)y) &\leq tf(x) + (1 - t)f(y) \\ \forall t \in [0, 1], f(x + t(y - x)) &\leq f(x) + t(f(y) - f(x)) \\ \forall t \in [0, 1], f(y) - f(x) &\geq \frac{f(x + t(y - x)) - f(x)}{t} \end{aligned}$$

Ainsi, en prenant la limite pour $t \downarrow 0$, on a que :

$$\forall x, y \in A, f(y) \geq f(x) + \nabla f(x)(y - x)$$

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 :

$$\begin{aligned} f(x) &\geq f(z) + \nabla f(z)(x - z) \\ f(y) &\geq f(z) + \nabla f(z)(y - z) \end{aligned}$$

En multipliant la première équation par $t \in [0, 1]$ et la seconde équation par $(1 - t)$, on obtient :

$$\begin{aligned} tf(x) + (1 - t)f(y) &\geq f(z) + \nabla f(z)(tx + (1 - t)y - z) \\ &= f(z) \\ f(tx + (1 - t)y) &\leq tf(x) + (1 - t)f(y) \end{aligned}$$

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 $\forall x \in A, f''(x) \geq 0$, alors par le théorème de la valeur moyenne de Taylor on a que :

$$\begin{aligned} \exists z \in [x, y], \quad f(y) &= f(x) + f'(x)(y - x) + \frac{1}{2}f''(z)(y - x)^2 \\ \Rightarrow f(y) &\geq f(x) + f'(x)(y - x) \end{aligned}$$

Finalement, supposons que $\forall x, y \in A, f(y) \geq f(x) + \nabla f(x)(y - x)$. Soit $x, y \in A$ tels que $y > x$. On a que :

$$\begin{aligned} f(y) &\geq f(x) + f'(x)(y - x) \\ f(x) &\geq f(y) + f'(y)(x - y) \end{aligned}$$

On en déduit donc que :

$$f'(x)(y - x) \leq f(y) - f(x) \leq f'(y)(y - x)$$

Donc en divisant par $(y - x)^2$ puis en prenant la limite pour $y \rightarrow x$, on obtient bien que la propriété souhaitée. \square

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 3. *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 (3) à l'aide du théorème (5).*

Solution. Comme $f : A \mapsto \mathbb{R}$ est convexe et différentiable, d'après le théorème (5) on a :

$$\forall x, y \in A, f(y) \geq f(x) + \nabla f(x)(y - x)$$

Donc en particulier pour \bar{x} défini comme $\nabla f(\bar{x}) = 0$:

$$\begin{aligned} \forall y \in A, f(y) &\geq f(\bar{x}) + \nabla f(\bar{x})(y - \bar{x}) \\ \forall y \in A, f(y) &\geq f(\bar{x}) \end{aligned}$$

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 4. *Soit un problème d'optimisation sans contraintes comme présenté dans (A.1) avec f une fonction strictement convexe et différentiable. Si $\nabla f(\bar{x}) = 0$, alors \bar{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 (4) 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 $z = \frac{a+b}{2} \in A$ comme combinaison convexe de deux points du domaine (avec $t = \frac{1}{2}$). 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 16 (Fonction L -Lipschitzienne). Soit $f : \mathcal{D} \rightarrow \mathbb{R}$ une fonction convexe. On dit que f est L -Lipschitzienne si il existe un nombre L tel que :

$$|f(y) - f(x)| \leq L\|y - x\|$$

Pour comprendre visuellement cette définition, on peut s'appuyer sur la figure (A.4).

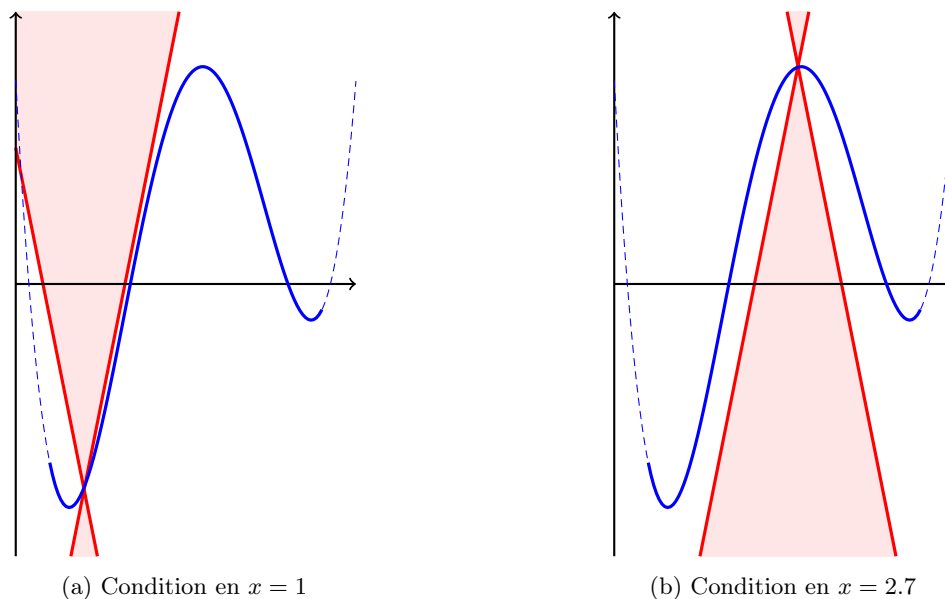


FIGURE A.4 – Illustration de la condition lipschitzienne pour une fonction

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 5. 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 maîtrise. Si l'on note $\tilde{x} \in \mathbb{R}^d$ le point que l'on obtient à la suite de la descente de gradient, on a $f(\tilde{x}) \leq f(x^*) + \varepsilon$.

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 :*

$$\|x_{t+1} - x^*\|^2 \leq \|x_t - x^*\|^2 - 2\eta(f(x_t) - f(x^*)) + \eta^2 L^2$$

Exercice A.5. *Prouver le lemme (2).*

Solution.

$$\begin{aligned} \|x_{t+1} - x^*\|^2 &= \|x_t - x^* - \eta \nabla f(x_t)\|^2 \\ &= \|x_t - x^*\|^2 - 2\eta (\nabla f(x_t))^t (x_t - x^*) + \eta^2 \|\nabla f(x_t)\|^2 \\ &\leq \|x_t - x^*\|^2 - 2\eta (f(x_t) - f(x^*)) + \eta^2 L^2 \end{aligned}$$

En exploitant le fait que f soit convexe et L -Lipschitzienne pour la dernière inégalité. \square

Nous avons donc maintenant tous les outils pour démontrer le résultat annoncé dans la proposition (5).

Démonstration. Soit $\Phi(t) = \|x_t - x^*\|^2$. Alors avec $\eta = \frac{\varepsilon}{L^2}$ et le précédent lemme on a :

$$\Phi(t) - \Phi(t+1) > 2\eta\varepsilon - \eta^2 L^2 = \frac{\varepsilon}{L^2}$$

Puisque par définition, $\Phi(0) = \|x^* - x_0\|^2$ et $\Phi(t) \geq 0$, la précédente équation ne peut pas être vérifiée pour tous $0 \leq t \leq \frac{L^2 \|x^* - x_0\|^2}{\varepsilon^2}$ d'où le résultat. \square

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 17 (Fonction β -smooth). *Soit $f : \mathcal{D} \mapsto \mathbb{R}$ une fonction convexe différentiable. On dit que f est β -smooth si :*

$$\forall x, y \in \mathcal{D}, f(y) \leq f(x) + \langle \nabla f(x), y - x \rangle + \frac{\beta}{2} \|y - x\|^2$$

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 6. 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 avons 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. C'est particulièrement vrai dans l'entraînement des réseaux de neurones, que nous ne traitons pas ici, où un très grand nombre de paramètres est à apprendre. 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 présenté en 1913 dans l'article *On the constitution of atoms and molecules*[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

Dans le papier originel de 1969 [MP69], on y 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 :

$$w_{t+1} = w_t + yx\mathbb{1}_{y\langle w_t, x \rangle \leq 0}$$

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 :

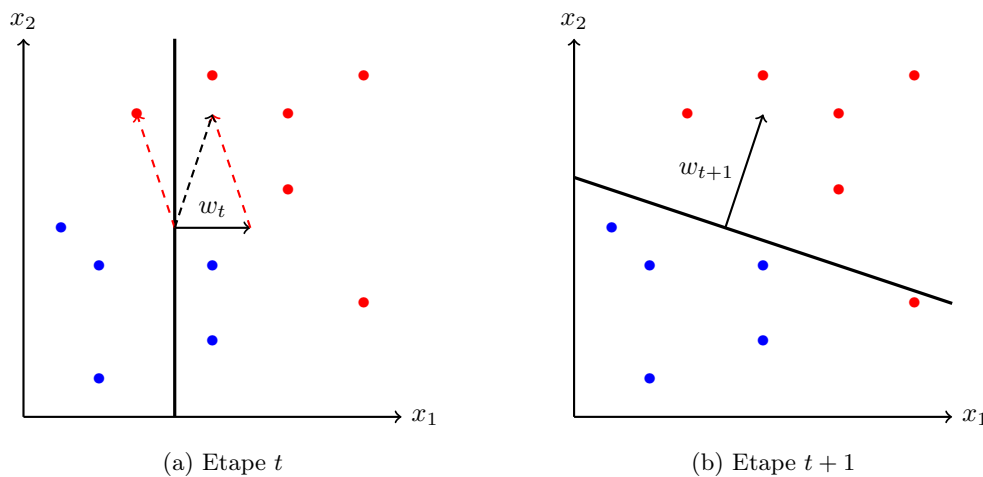


FIGURE B.1 – Intuition géométrique du perceptron (-1)

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 6 (Convergence du perceptron). *On suppose que les données issues d'un dataset \mathcal{D} soient linéairement séparables et de plus que :*

$$\exists \gamma > 0, \forall i \leq n, y_i \langle w^*, x_i \rangle \leq \gamma$$

On note $R = \max_{1 \leq i \leq n} \|x_i\|$. Alors la k -ième erreur de classification du perceptron aura lieu avant :

$$k \leq \left(\frac{R}{\gamma} \right)^2 \|w^*\|^2$$

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 7 (Cauchy-Schwarz). *Soient $u, v \in \mathbb{R}^n$ et $\|\cdot\| : \mathbb{R}^n \rightarrow \mathbb{R}^+$ une norme pour \mathbb{R}^n . Alors :*

$$\langle u, v \rangle \leq \|u\| \|v\|$$

Démonstration. On considère le polynôme $P : \mathbb{R} \rightarrow \mathbb{R}$ 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 $\forall \lambda \in \mathbb{R}, P(\lambda) \geq 0$. Autrement dit, son discriminant Δ est négatif ou nul. Alors on a que :

$$4\langle u, v \rangle - 4\|u\|^2 \|v\|^2 \leq 0 \iff \langle u, v \rangle \leq \|u\| \|v\|$$

□

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 (6).

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 :

$$\begin{aligned} \langle w_k, w^* \rangle &= \langle w_{k-1} + y_i x_i, w^* \rangle \\ &= \langle w_{k-1}, w^* \rangle + y_i \langle x_i, w^* \rangle \\ &\geq \langle w_{k-1}, w^* \rangle + \gamma \end{aligned}$$

Donc par une récurrence immédiate, on a :

$$\langle w_k, w^* \rangle \geq k\gamma$$

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 :

$$\begin{aligned}\|w_k\|^2 &= \|w_{k-1}\|^2 + 2y_i \langle w_k, x_i \rangle + y_i^2 \|x_i\|^2 \\ &\leq \|w_{k-1}\|^2 - 2\gamma + R^2 \\ &\leq kR^2\end{aligned}$$

Ainsi, en combinant les résultats que l'on a obtenus, on arrive à :

$$\begin{aligned}k^2 \gamma^2 &\leq \langle w_k, w^* \rangle^2 \\ &\leq kR^2 \|w^*\|^2 \\ k &\leq \left(\frac{R}{\gamma} \right)^2 \|w^*\|^2\end{aligned}$$

□

B.3 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.2 (Moyenne harmonique). Soit $(x_k)_{k \leq n}$ des réels strictement positifs. On définit :

- Moyenne arithmétique : $A_n = \frac{1}{n} \sum_{k=1}^n x_k$
- Moyenne harmonique : $H_n = \frac{n}{\sum_{k=1}^n \frac{1}{x_k}} \iff \frac{1}{H_n} = \frac{1}{n} \sum_{k=1}^n \frac{1}{x_k}$

Montrer que :

$$H_n \leq A_n$$

Solution. Puisque $(x_k)_{k \leq n}$, on sait qu'il existe une unique suite $(y_k)_{k \leq n}$ telle que $\forall k \leq n, x_k = y_k^2$. Ainsi, on a :

$$\begin{aligned}\frac{A_n}{H_n} &= \left(\frac{1}{n} \sum_{k=1}^n y_k^2 \right) \left(\frac{1}{n} \sum_{k=1}^n \frac{1}{y_k^2} \right) \\ &\leq \frac{1}{n^2} \sum_{k=1}^n \frac{y_k^2}{y_k^2} \quad \text{avec l'inégalité de Cauchy-Schwartz} \\ &\leq 1 \\ H_n &\leq A_n\end{aligned}$$

□

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.

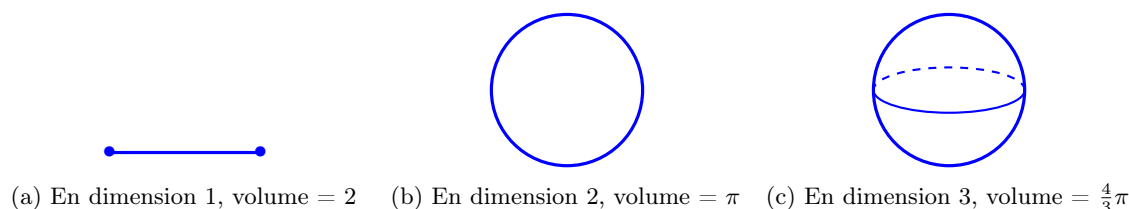


FIGURE D.1 – Représentation et volume d'une hypersphère de rayon 1 dans 3 espaces de dimensions différentes

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).

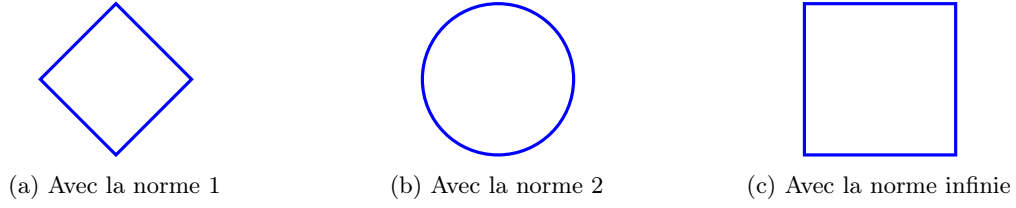


FIGURE D.2 – Représentation d'une hypersphère de rayon 1 en dimension 2 pour 3 normes différentes

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 :

$$\begin{aligned} B_n^p(R) &= \{(x_1, \dots, x_n) \in \mathbb{R}^n, \sum_{i=1}^n x_i^p \leq R^p\} \\ &= \{u \in \mathbb{R}^n, \|u\|_p^p \leq R^p\} \end{aligned}$$

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 Lebesgue dans \mathbb{R}^n . Formellement :

$$V_n^p(R) = \int_{B_n^p(R)} \bigotimes_{i=1}^n dx_i$$

Proposition 8 (Volume d'une hypersphère). *Avec les notations précédentes, on a :*

$$\forall R > 0, \forall n \geq 2, \forall p \geq 1, \quad V_n^p(R) = \frac{\left(2R\Gamma\left(\frac{1}{p} + 1\right)\right)^n}{\Gamma\left(\frac{n}{p} + 1\right)}$$

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 :

$$\Gamma(x) = \int_0^{+\infty} e^{-t} t^{x-1} dt$$

La preuve de ce résultat est assez technique, nous ne la présenterons pas ici¹. 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 avons 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.

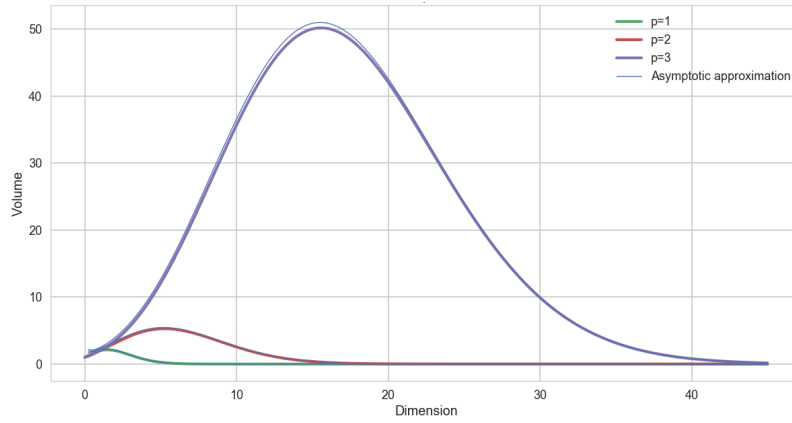


FIGURE D.3 – Volume de la boule unité en fonction de la dimension de son espace pour trois p -normes

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).

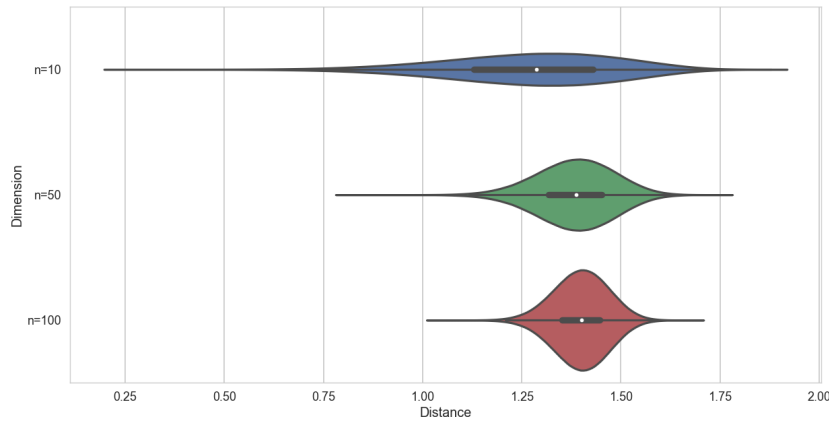


FIGURE D.4 – Distribution des distances entre chaque point en fonction de la dimension de l'espace

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 :

$$\langle x, y \rangle = \|x\|_2 \|y\|_2 \cos(\theta)$$

Avec θ l'angle entre le représentant de x et le représentant de y à l'origine comme défini dans la figure (D.5).

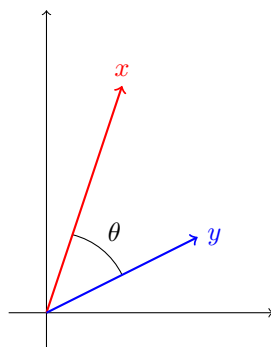


FIGURE D.5 – Définition de la métrique cosinus

Nous pouvons donc naturellement définir la métrique cosinus comme :

$$\text{cosine}(x, y) = \frac{\langle x, y \rangle}{\|x\|_2 \|y\|_2}$$

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 :

$$\begin{aligned} \|x - y\|_2^2 &= \|x\|_2^2 + \|y\|_2^2 - 2 \langle x, y \rangle \\ &= \|x\|_2^2 + \|y\|_2^2 - 2 \text{cosine}(x, y) \|x\|_2 \|y\|_2 \\ &= 2 [1 - \text{cosine}(x, y)] \end{aligned}$$

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 $1 - \frac{1}{n}$:

$$|\text{cosine}(x, y)| = O\left(\sqrt{\frac{\log n}{n}}\right)$$

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 $X = \langle a, x \rangle$, alors il est simple de montrer que $\mathbb{E}[X] = 0$ et $\mathbb{E}[X^2] = \frac{1}{n}$. Ainsi, en exploitant l'inégalité de Chernoff on a :

$$\mathbb{P}(|X| \geq t) \leq 2e^{-\frac{nt^2}{4}}$$

Donc pour $\varepsilon = 2e^{-\frac{nt^2}{4}} \iff t = \sqrt{\frac{-4 \log(\frac{\varepsilon}{2})}{n}}$ et si l'on choisit $\varepsilon = \frac{1}{n}$, on obtient :

$$\mathbb{P}\left(|\cosine(x, y)| \geq \sqrt{\frac{4 \log(2n)}{n}}\right) \leq \frac{1}{n}$$

□

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 qualité 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.

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, Stanford, 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.
- [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.
- [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.
- [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.
- [GEW06] Pierre Geurts, Damien Ernst, and Louis Wehenkel. Extremely randomized trees. *Machine learning*, 2006.
- [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.
- [Jag13] Martin Jaggi. An equivalence between the lasso and support vector machines. *Regularization, optimization, kernels, and support vector machines*, 2013.
- [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.
- [LR76] Hyafil Laurent and Ronald L Rivest. Constructing optimal binary decision trees is np-complete. *Information processing letters*, 1976.
- [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.
- [Nes83] Youri Nesterov. A method of solving a convex programming problem with convergence rate $o(1/k^2)$. *Soviet Mathematics Doklady*, 1983.
- [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.
- [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.
- [SBC14] Weijie Su, Stephen Boyd, and Emmanuel Candes. A differential equation for modeling nesterov’s accelerated gradient method : theory and insights. *Advances in neural information processing systems*, 2014.
- [Sch13] Robert E Schapire. Explaining adaboost. In *Empirical inference*. Springer, 2013.
- [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.
- [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.
- [ZCS⁺15] Quan Zhou, Wenlin Chen, Shiji Song, Jacob Gardner, Kilian Weinberger, and Yixin Chen. A reduction of the elastic net to support vector machines with an application to gpu computing. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2015.
- [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.