
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)

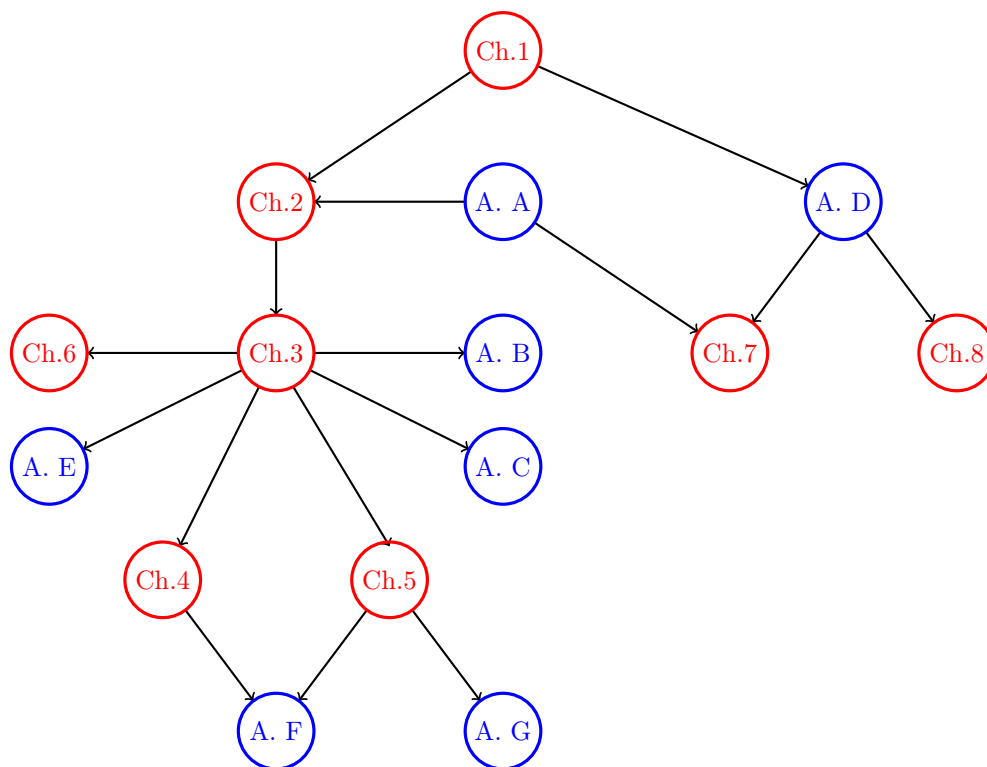


Table des matières

1	Introduction au Machine Learning	5
1.1	Les différentes approches du Machine Learning	5
1.2	Apprentissage supervisé - plus formellement	6
1.3	Comment sélectionner les modèles ?	8
1.3.1	Train, Validation, Test	8
1.3.2	Cross-validation	9
1.3.3	Occam's Razor	10
2	Régression linéaire et variantes	12
2.1	Régression linéaire classique	12
2.2	Mesurer la performance d'une régression	15
2.2.1	Erreur quadratique moyenne	15
2.2.2	Coefficient de détermination R^2	16
2.3	Régressions pénalisées	17
2.3.1	Régression Ridge	18
2.3.2	Régression LASSO	18
2.3.3	Régression ElasticNet	19
2.4	Régressions polynomiales : approximateurs universels	19
3	Régression Logistique	22
3.1	Modélisation	22
3.2	Descente de gradient	24
3.3	Mesurer la performance d'une classification	27
3.3.1	Pour un seuil fixé	27
3.3.2	Sans choix de seuil	30
4	Arbre et Random Forest	32
4.1	Arbre de décision	32
4.1.1	Meilleure partition	33
4.1.2	Critères d'arrêt	36
4.2	Méthodes ensemblistes	37
4.2.1	Bagging	37
4.2.2	Random Forest	39
5	Boosting	41
5.1	Algorithme AdaBoost	41
5.2	Gradient Boosting	43
5.2.1	Principaux algorithmes de Gradient Boosting	46
6	Support Vector Machine	50
6.1	Intuition	50
6.2	Formalisation du problème	51
6.2.1	Dans le cas séparable	51
6.2.2	Dans le cas non-séparable	53
6.2.3	Problème primal et dual	54

6.2.4	Kernel trick	58
6.3	L'algorithme en pratique	60
6.3.1	Noyaux classiques	60
6.3.2	Fine-tuning	60
7	Clustering	62
7.1	Distance : ce qui se ressemble est proche	62
7.2	Approche statistique	63
7.2.1	K-Means	63
7.2.2	Kmeans++ : un meilleur départ	65
7.3	Approche par densité	66
7.3.1	DBSCAN	68
7.3.2	OPTICS	68
7.4	Mesures de performance	71
7.4.1	Silhouette score	71
7.4.2	L'index de Calinski-Harabasz	72
8	Réduction de dimension	74
8.1	Lemme de Johnson-Lindenstrauss	74
8.2	Analyse par composantes principales	77
8.2.1	Diagonalisation d'une matrice	77
8.2.2	Application à la réduction de dimensions	80
8.3	UMAP	82
8.3.1	Formation du graphe en grande dimension	82
8.3.2	Réduction du graphe	84
8.3.3	Utilisation en pratique d'UMAP	85
A	Rappel et utilisation de la convexité pour le Machine Learning	87
A.1	Définition et propriétés	87
A.1.1	Ensemble et fonction convexe	87
A.1.2	Caractérisation du premier et deuxième ordre	88
A.2	Résultats d'optimisations	90
A.3	Vitesse de convergence pour la descente de gradient	92
A.3.1	Pour une fonction Lipschitzienne	92
A.3.2	Fonction β -smooth	93
B	Algorithme du Perceptron	95
B.1	Description	95
B.2	Preuve de convergence	97
B.3	Bonus : utilisation de l'inégalité de Cauchy-Schwarz	98
C	Algorithme Naïve Bayes	99
C.1	Probabilité conditionnelle et théorème de Bayes	99
C.2	Application en Machine Learning	101
D	Fléau de la dimension	103
D.1	Volume d'une hypersphère	103
D.2	Orthogonalité à la surface d'une hypersphère	106
D.3	Interpolation et extrapolation	107
E	Accélération de Nesterov	108
E.1	Etude analytique	108
E.2	Etude via son équation différentielle	108
F	Stacking et Blending : au-delà du Bagging et Boosting	109

G Double descent : vers le Machine Learning moderne	111
G.1 Prédiction de la théorie de Vapnik-Chervonenkis	111
G.1.1 Cas d'AdaBoost	113
G.2 Double Descente : un challenge théorique	114
G.3 Comment exploiter la double descente?	115

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 :

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

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

On définit les notations :

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

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 :

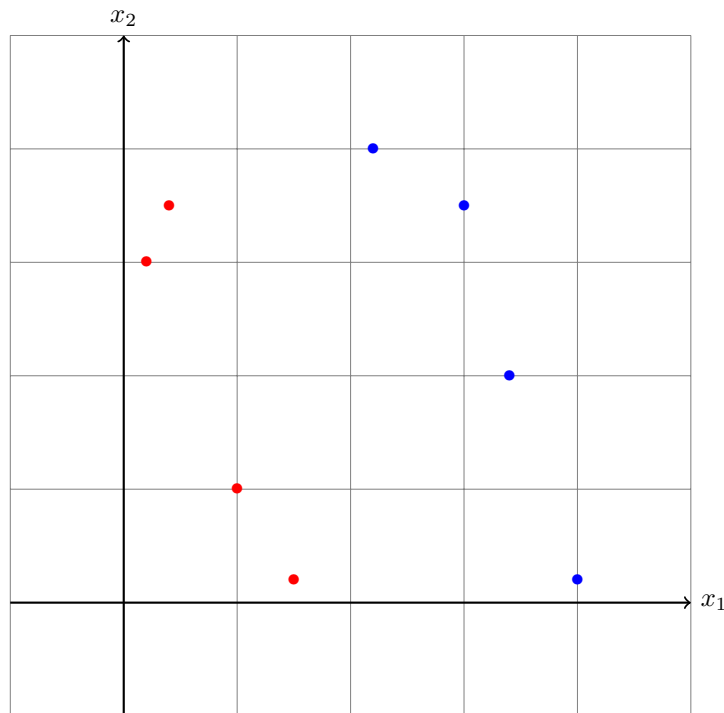


FIGURE 1.1 – Exemple d'un problème de classification avec deux classes et deux indicateurs pour identifier la classe

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).

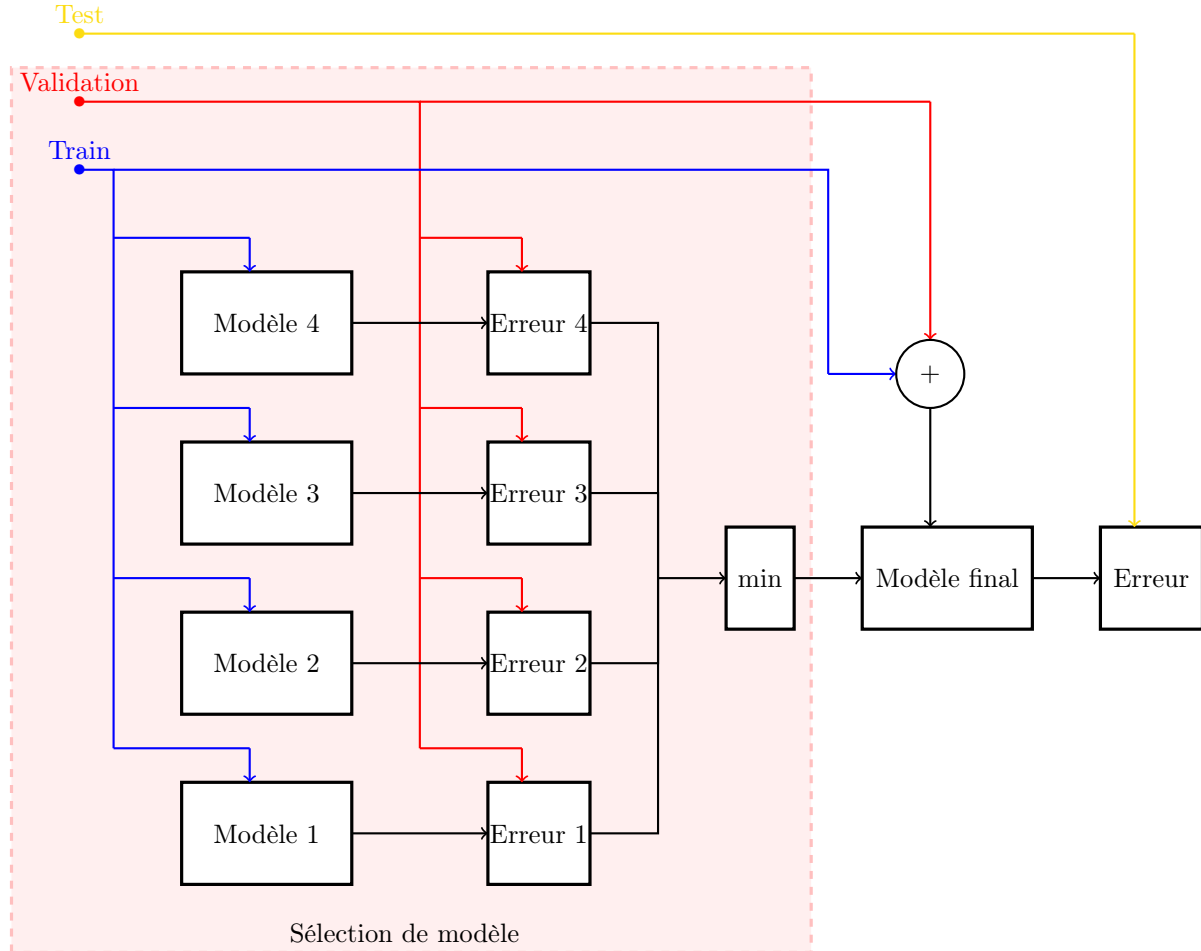


FIGURE 1.2 – Méthodologie Train - Validation - Test

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 partie égale 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

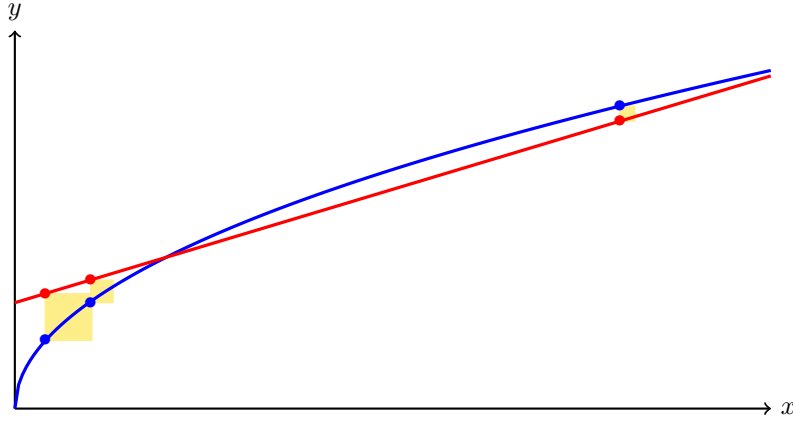


FIGURE 2.1 – Visualisation de la MSE entre la Régression linéaire et vraie courbe

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).

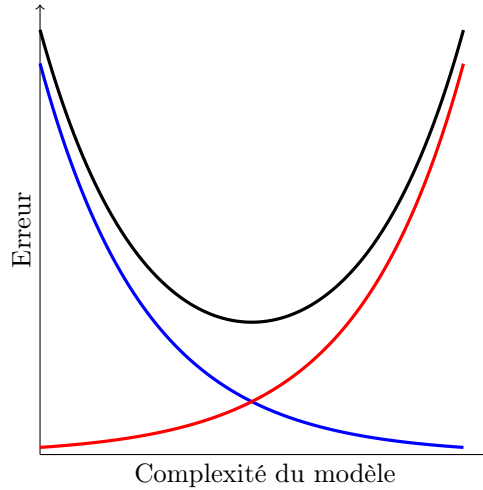


FIGURE 2.2 – Trade-off **biais**-**variance**

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

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

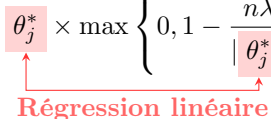
2.3.2 Régression LASSO

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 :

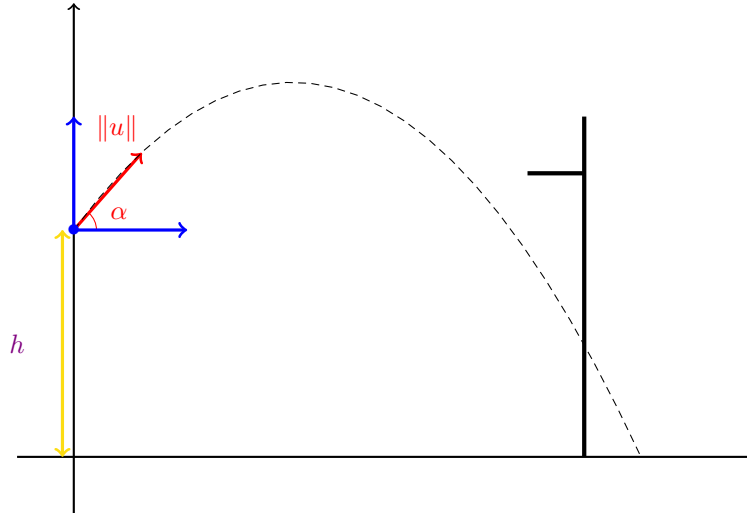


FIGURE 2.3 – Simulation d'un mini-jeu de basketball

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

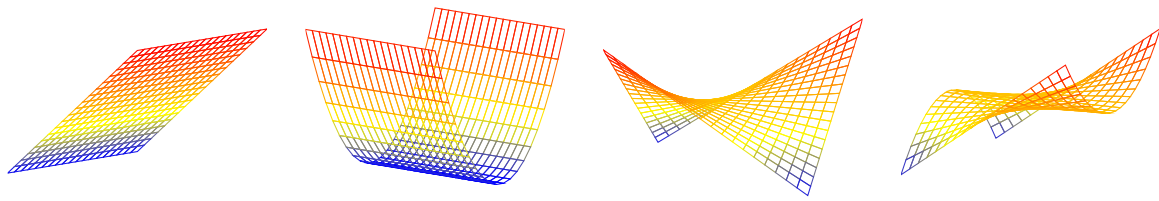
Et qui renvoie une nouvelle matrice de données avec les informations polynomiales.

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 :



(a) $f(x_1, x_2) = x_1$

(b) $f(x_1, x_2) = x_2^2$

(c) $f(x_1, x_2) = x_1 x_2$

(d) $f(x_1, x_2) = x_1 x_2^2$

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

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.

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

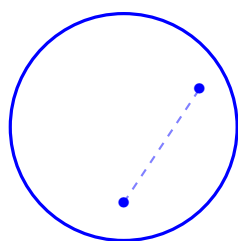
A.1.1 Ensemble et fonction convexe

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 :

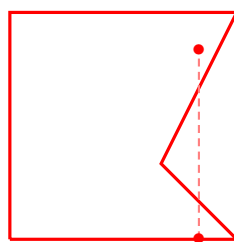
Définition 14. Soit A un sous-ensemble de \mathbb{R}^d . On dit que A est un ensemble convexe si :

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

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).

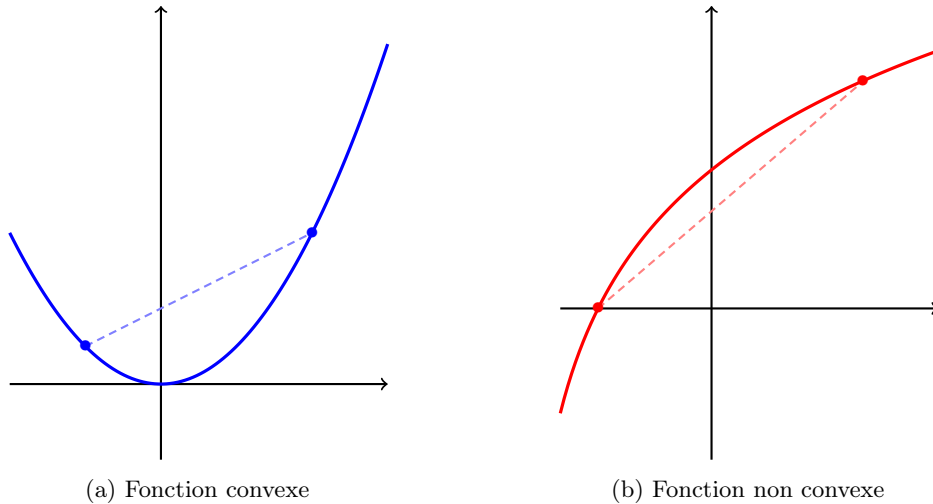


FIGURE A.2 – Exemple et contre exemple de fonction convexe

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).

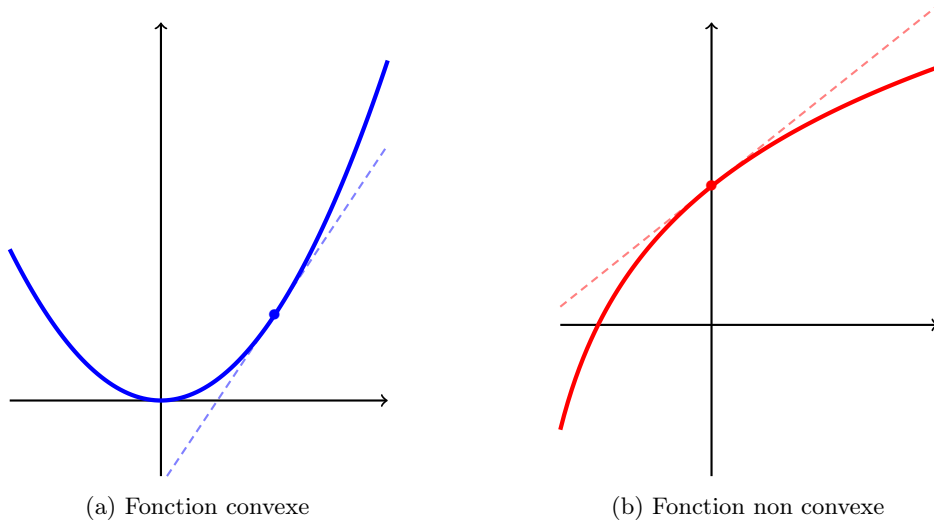


FIGURE A.3 – Illustration de la propriété de sous-estimateur pour une fonction **convexe** et contre exemple pour une fonction **non convexe**

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).

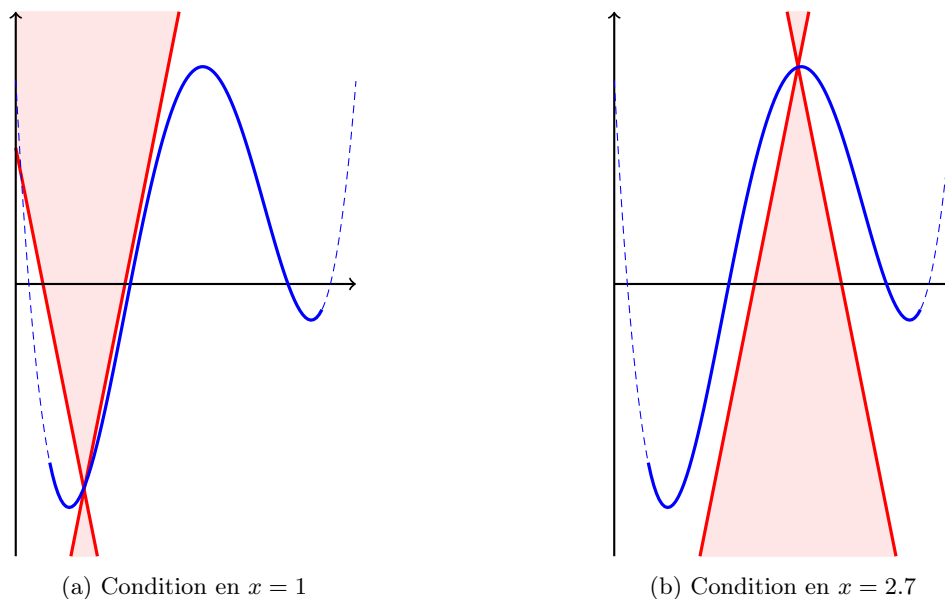


FIGURE A.4 – Illustration de la condition lipschitzienne pour une fonction

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 :

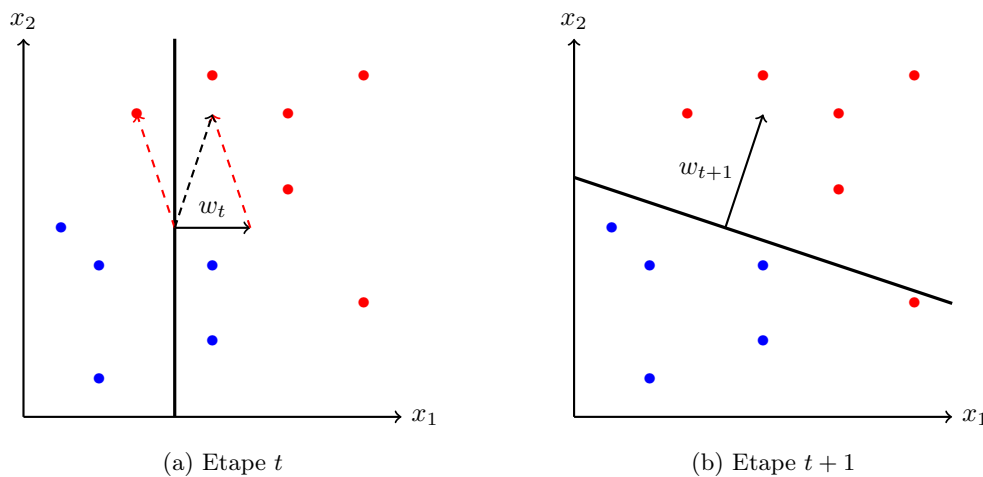


FIGURE B.1 – Intuition géométrique du perceptron (-1)

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 :

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

□

Annexe C

Algorithme Naive Bayes

Le Machine Learning est un domaine au croisement de l'analyse, l'algèbre et des statistiques comme nous avons pu le constater au travers des chapitres précédents. Nous avons exploité largement plusieurs formalismes et idées qui n'ont pas explicitement été développée pour cet usage.

D'autres domaines des mathématiques plus spécifiques encore sont exploités actuellement, on pense par exemple à la théorie des catégories pour l'algorithme UMAP. Mais nous n'avons jusque-là jamais exploité le formalisme et les idées des probabilités.

C'est un choix qui a été fait pour rendre le cours le plus compréhensible possible. La force des mathématiques est d'être capable de faire des liens entre les différents domaines qui composent cette discipline, mais ce n'est pas une chose simple et cela peut être largement perturbant pour quiconque n'y est pas initié.

Par soucis de compréhension, nous avons donc fait le choix de proposer la présentation des algorithmes sous le prisme de l'analyse et de l'algèbre. Nous proposons ici un algorithme qui exploite et nécessite une vision probabiliste.

C.1 Probabilité conditionnelle et théorème de Bayes

Considérons un lancé de dé à 6 faces équilibrés. Nous nous intéressons à deux événements :

- A : le nombre est 2
- B : le nombre est pair

Nous sommes parfaitement capables de calculer $\mathbb{P}(A) = \frac{1}{6}$ et $\mathbb{P}(B) = \frac{1}{2}$. Mais quelle est la probabilité que le nombre soit 2 **sachant** que le nombre est pair ? Autrement dit quelle est la probabilité de l'événement A sachant que l'événement B s'est réalisé ?

Elle vaut évidemment $\frac{1}{3}$ puisque parmi les 3 possibilités de nombre pair, il n'y en a qu'une qui réalise l'événement A . Mais comment formaliser cela ?

Définition 18 (Probabilité conditionnelle). *Soient A et B deux événements, et $\mathbb{P}(B) \neq 0$. La probabilité conditionnelle de A sachant B est définie par :*

$$\mathbb{P}(A | B) = \frac{\mathbb{P}(A \cap B)}{\mathbb{P}(B)}$$

La notation $A | B$ se lit " A sachant B ", et on remarque que l'on obtient bien la bonne probabilité dans l'exemple introductif.

Exercice C.1 (Propriétés). Soient A_1 , A_2 et B trois événements, et $\mathbb{P}(B) \neq 0$. Montrer que :

1. $\mathbb{P}(A_1 \cup A_2 | B) = \mathbb{P}(A_1 | B) + \mathbb{P}(A_2 | B) - \mathbb{P}(A_1 \cap A_2 | B)$
2. $\mathbb{P}(\overline{A_1} | B) = 1 - \mathbb{P}(A_1 | B)$

Solution. Soient A_1 , A_2 et B trois événements, et $\mathbb{P}(B) \neq 0$.

1. Par définition d'une probabilité conditionnelle, et par les axiomes de probabilité :

$$\begin{aligned}
 \mathbb{P}(A_1 \cup A_2 | B) &= \frac{\mathbb{P}((A_1 \cup A_2) \cap B)}{\mathbb{P}(B)} \\
 &= \frac{\mathbb{P}((A_1 \cap B) \cup (A_2 \cap B))}{\mathbb{P}(B)} \\
 &= \frac{\mathbb{P}(A_1 \cap B) + \mathbb{P}(A_2 \cap B) - \mathbb{P}(A_1 \cap A_2 \cap B)}{\mathbb{P}(B)} \\
 &= \mathbb{P}(A_1 | B) + \mathbb{P}(A_2 | B) - \mathbb{P}(A_1 \cap A_2 | B)
 \end{aligned}$$

2. Par définition d'une probabilité conditionnelle :

$$\begin{aligned}
 \mathbb{P}(\overline{A_1} | B) + \mathbb{P}(A_1 | B) &= \frac{\mathbb{P}(\overline{A_1} \cap B) + \mathbb{P}(A_1 \cap B)}{\mathbb{P}(B)} \\
 &= 1
 \end{aligned}$$

□

Exercice C.2. Soient A , B et C trois événements de probabilité non nulle. Montrer que :

1. $\mathbb{P}(A \cap B) = \mathbb{P}(A | B) \mathbb{P}(B)$
2. $\mathbb{P}(A \cap B \cap C) = \mathbb{P}(A | B \cap C) \mathbb{P}(B | C) \mathbb{P}(C)$
3. En exploitant la première égalité, montrer que $\mathbb{P}(A | B) = \frac{\mathbb{P}(A) \mathbb{P}(B | A)}{\mathbb{P}(B)}$

Solution. Soient A , B et C trois événements de probabilité non nulle.

1. Le résultat est immédiat avec la définition d'une probabilité conditionnelle.
2. Avec la définition d'une probabilité conditionnelle et la question précédente, on a :

$$\begin{aligned}
 \mathbb{P}(A \cap B \cap C) &= \mathbb{P}(A | B \cap C) \mathbb{P}(B \cap C) \\
 &= \mathbb{P}(A | B \cap C) \mathbb{P}(B | C) \mathbb{P}(C)
 \end{aligned}$$

On y voit une forme de factorisation d'une intersection via des probabilités conditionnelles.

3. Avec la définition d'une probabilité conditionnelle et la première question, on a :

$$\begin{aligned}
 \mathbb{P}(A | B) &= \frac{\mathbb{P}(A \cap B)}{\mathbb{P}(B)} \\
 &= \frac{\mathbb{P}(B \cap A)}{\mathbb{P}(B)} \\
 &= \frac{\mathbb{P}(A) \mathbb{P}(B | A)}{\mathbb{P}(B)}
 \end{aligned}$$

□

La dernière égalité est particulièrement intéressante : nous sommes capables de calculer la probabilité de A sachant B si l'on connaît la probabilité de B sachant A (et d'autres choses). Si nous pensons en terme temporel, nous venons presque d'inverser le temps !

On rappelle qu'en probabilité l'univers Ω est l'ensemble des issues possibles. On peut définir $(A_i)_{i \leq n}$ une partition de l'univers si chacun des événements A_i sont exclusifs et que l'union des A_i forme exactement Ω .

Nous avons maintenant l'ensemble des informations pour énoncer le théorème qui permet de définir l'algorithme Naive Bayes.

Théorème 7 (Bayes, 1763). Soit $(A_i)_{i \leq n}$ une partition de l'univers Ω . Soit B un événement de probabilité non nulle. On a :

$$\mathbb{P}(A_i | B) = \frac{\mathbb{P}(B | A_i) \mathbb{P}(A_i)}{\sum_{j=1}^n \mathbb{P}(B | A_j) \mathbb{P}(A_j)}$$

Démonstration. Par récurrence immédiate, nous avons montré dans un exercice que l'initialisation était vraie. □

C.2 Application en Machine Learning

Considérons un dataset de classification avec K classes :

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

Ensemble de dimension finie K

Pour simplifier les notations, nous ferons la confusion entre les variables aléatoires $(x_j)_{j \leq d}$ et les réalisations $x^{(i)} = (x_k)_{k \leq d}^{(i)}$ qui forment le dataset \mathcal{D} . De même, nous ferons la confusion entre la variable aléatoire y et les réalisations y_i qui forment le dataset \mathcal{D} .

Nous aimerions être capables d'estimer la probabilité : $\mathbb{P} \left(y = k \mid \bigcap_{j=1}^d x_j \right)$ c'est à dire la probabilité que l'observation considérée soit de la classe $k \leq K$. Nous pouvons le faire, mais il va falloir construire un tableau énorme où l'on va répertorier la probabilité pour chacune des possibilités de $\bigcap_{j=1}^d x_j$. Il va falloir également avoir une quantité astronomique d'observations pour avoir une bonne estimation de chacune des cellules. Ce n'est pas possible en pratique. Exploitions la section précédente.

Par le théorème de Bayes, on a :

$$\begin{array}{c}
\text{Antérieur} \quad \text{Vraisemblance} \\
\downarrow \quad \downarrow \\
\mathbb{P}(\{y = k\}) \times \mathbb{P}\left(\bigcap_{j=1}^d x_j \mid \{y = k\}\right) \\
\uparrow \quad \uparrow \\
\text{Postérieur} \quad \text{Evidence}
\end{array}
\quad \mathbb{P}\left(\{y = k\} \mid \bigcap_{j=1}^d x_j\right) = \frac{\mathbb{P}(\{y = k\}) \times \mathbb{P}\left(\bigcap_{j=1}^d x_j \mid \{y = k\}\right)}{\mathbb{P}\left(\bigcap_{j=1}^d x_j\right)} \quad (C.1)$$

Chacun des termes a un nom qui a du sens dans le domaine des probabilités bayésiennes. On remarque que le dénominateur est constant si l'on connaît l'ensemble des probabilités associées aux informations, ainsi seulement le numérateur nous intéresse. Nous avons accès à $\mathbb{P}(\{y = k\})$, il nous reste à bien estimer la vraisemblance.

En exploitant la généralisation d'une égalité que nous avons montré dans un précédent exercice, on peut montrer que :

$$\mathbb{P}\left(\bigcap_{j=1}^d x_j \mid \{y = k\}\right) = \prod_{j=1}^d \mathbb{P}\left(x_j \mid \{y = k\} \cap \bigcap_{i=1}^{j-1} x_i\right)$$

Nous ne semblons pas plus avancés à ce stade. Nous allons donc faire l'hypothèse **naïve** qui donne le nom de cet algorithme : les $(x_j)_{j \leq d}$ sont indépendants entre eux conditionnellement à l'événement $\{y = k\}$. Si l'on traduit formellement :

$$\forall i, j \leq d, i \neq j \Rightarrow \mathbb{P}(x_i \mid \{y = k\} \cap x_j) = \mathbb{P}(x_i \mid \{y = k\}) \quad (\text{Hypothèse naïve})$$

Ainsi, nous pouvons réécrire l'équation (C.1) comme :

$$\mathbb{P}\left(\{y = k\} \mid \bigcap_{j=1}^d x_j\right) = \frac{\mathbb{P}(\{y = k\}) \times \prod_{j=1}^d \mathbb{P}(x_j \mid \{y = k\})}{\mathbb{P}\left(\bigcap_{j=1}^d x_j\right)} \quad (C.2)$$

Il ne reste plus qu'à estimer maintenant chacune des valeurs $\mathbb{P}(x_j \mid \{y = k\})$. On le fait avec un dataset en distinguant les cas :

- S'il s'agit d'une variable continue : on applique une loi gaussienne par exemple, donc on va apprendre deux paramètres (moyenne et écart-type)
- S'il s'agit d'une variable discrète : on applique une loi binomiale par exemple

Ceci conclut la présentation succincte de l'algorithme Naïve Bayes. Comme expliqué dans l'introduction, nous aurions pu proposer d'autres visions pour présenter quelques algorithmes. Ceci est notamment vrai pour les algorithmes de régression linéaire et logistique que l'on peut unir formellement via l'exploitation de la famille exponentielle par exemple. Nous avons touché du doigt ce lien en observant par exemple que les équations de descente de gradient sont largement similaires, mais cela va bien au-delà.

Si le lecteur a une fibre probabiliste, nous l'encourageons à redécouvrir le Machine Learning par ce prisme : multiplier les angles de vue sur un même sujet permet de l'embrasser complètement.

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.