

Initiation au Machine Learning

10 novembre 2021

Table des matières

1	Les grands principes	5
1.1	Introduction	5
1.1.1	Quelques bribes d'histoire	5
1.1.2	Définition de l'Intelligence Artificielle	7
1.1.3	Les moteurs de règles	8
1.2	Le Machine Learning	9
1.2.1	Première approche	9
1.2.2	Les définitions du Machine Learning	9
1.2.3	L'apprentissage supervisé, qu'est-ce-que c'est ?	10
1.2.4	Illustrons cela sur un exemple	10
1.2.5	L'apprentissage non supervisé, c'est vraiment différent ?	12
1.2.6	L'apprentissage par renforcement, si, si, vous connaissez...!	12
1.2.7	Un beau dessin...	13
1.3	Les données	13
1.4	Quelques fondamentaux avant de plonger dans la programmation !	15
1.4.1	Petit rappel : Apprentissage Supervisé et problème de Régression	15
1.4.2	Voici la recette à suivre pour réaliser votre premier modèle de Machine Learning	15
1.5	Python, le langage du dataScientist	19
1.5.1	Pandas	20
1.5.2	Numpy	20
1.5.3	Matplotlib et Seaborn	21
1.5.4	Scikit-Learn	21
1.5.5	TensorFlow	21
1.5.6	Jupyter et les notebooks	21
1.5.7	1er notebook, utilisation des bibliothèques de base pour la data science	21
1.5.8	2ème notebook, utilisation de la bibliothèque Scikit-Learn	22
2	Machine Learning avec Scikit-Learn	25
2.1	Test et validation	25
2.2	Généralisation, surapprentissage, sous-apprentissage	28
2.2.1	Le biais et la variance en statistiques	28
2.2.2	La généralisation	29
2.2.3	Surapprentissage et compromis biais-variance	29
2.2.4	Les problèmes mal posés	31
2.2.5	En résumé	31
2.3	Les métriques	32
2.3.1	Métriques pour la régression	32
2.3.2	Les métriques pour la classification	33
2.4	Quelques précisions sur les métriques de classification	34
2.4.1	Compromis Précision-Rappel	34
2.4.2	Courbe ROC-AUC	36
2.5	Les algorithmes les plus fréquemment utilisés pour l'apprentissage supervisé	37
2.6	Les algorithmes les plus fréquemment utilisés pour l'apprentissage non supervisé	37

Chapitre 1

Les grands principes

1.1 Introduction

1.1.1 Quelques bribes d'histoire

L'intelligence artificielle est, depuis toujours, une des préoccupations principales de l'homme et rejoint l'histoire de la pensée. L'intelligence artificielle a évolué en dents de scie lors des cinquante dernières années. Elle a connu des périodes fastes entre 1957 et 1973, des moments de disette jusqu'au début des années 1980 et de désillusion durant la décennie 1990, le matériel ne suivant pas les besoins de calcul, pour, enfin, finir en apothéose dans les années 2010. Mais cette histoire a démarré il y a bien longtemps.

L'année 1950 est un tournant décisif dans l'histoire de l'intelligence artificielle, avec la publication de l'article « Computing Machinery and Intelligence » d'Alan Turing¹, la figure tutélaire de l'informatique moderne, vainqueur du code allemand des machines Enigma². Dans son article, Turing pose les fondements de l'IA moderne et décrit le test de Turing qui met au défi des humains de reconnaître si leur interlocuteur est un humain ou une machine.



Fig. 1.2 : Alan Turing

Fig. 1.1 : La machine Enigma

¹https://fr.wikipedia.org/wiki/Alan_Turing

²[https://fr.wikipedia.org/wiki/Enigma_\(machine\)](https://fr.wikipedia.org/wiki/Enigma_(machine))

John McCarthy³ et Marvin Minsky⁴ posent ensuite les bases de la « machine pensante » ainsi que l'expression « intelligence artificielle », qui apparaît en 1956 au cours de la fameuse conférence de Dartmouth⁵, dans le New Hampshire, aux États-Unis.

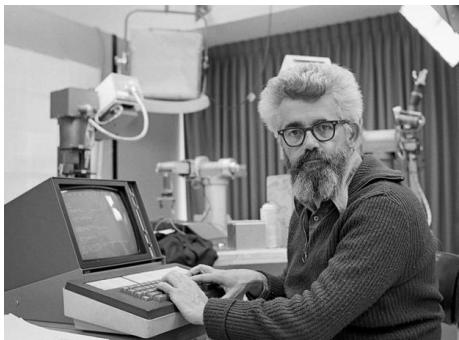


Fig. 1.3 : John McCarthy

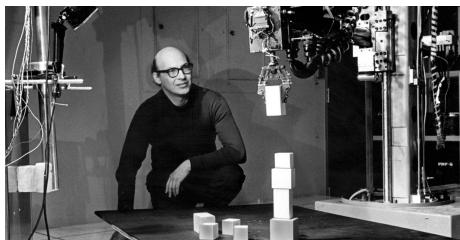


Fig. 1.4 : Marvin Minsky

Dès le début, deux courants de pensée vont s'affronter et rythmer les différentes recherches de l'IA : le cognitivisme et le connexionnisme. Deux notions qu'il faut intégrer. La première, le cognitivisme, privilégie une vision et un traitement symboliques de la pensée. Cette vision symbolique de l'intelligence et ses outils d'évaluation d'expression symbolique appelés « moteurs d'inférences » donnèrent naissance dans les années 1960-1970 aux systèmes experts que nous utilisons encore aujourd'hui dans l'industrie et dont nous connaissons les limites. Le connexionnisme, pour sa part, est créé par deux neurologues, Warren McCulloch⁶ et Walter Pitts,⁷ qui proposent, dès 1943, dans leur publication « A Logical Calculus of Ideas Immanent in Nervous Activity » de reproduire dans une machine le fonctionnement interne du cerveau humain et inventent le neurone formel, le premier modèle mathématique du neurone. Marvin Minsky généralisera le concept de neurone formel au réseau de neurones dans une thèse à Princeton intitulée *Neural Nets and the Brain Model Problem* (1954).

Les fondations de l'IA étaient nées. Il faudra attendre la fin des années 1990 pour que ces théories s'affinent, et que la puissance de calcul des ordinateurs permette leur utilisation et de nouveaux développements. C'est à cette même époque qu'un chercheur français, Yann Le Cun⁸, va donner aux réseaux de neurones leurs premières lettres de noblesse en réalisant un lecteur de code postal manuscrit que l'US Post va utiliser à grande échelle. Non content de ce premier développement, il inventera dans la foulée les réseaux de neurones profonds, le Deep Learning, en étudiant la physiologie de l'œil humain.



Fig. 1.5 : Yann Le Cun

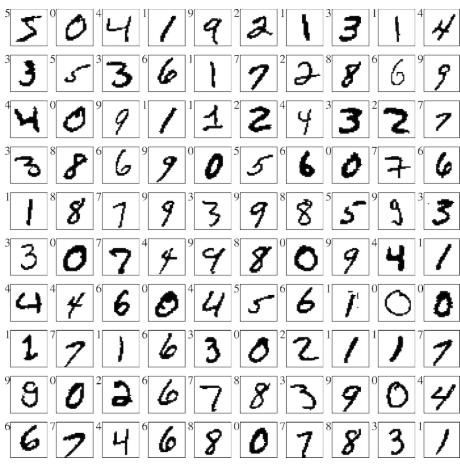


Fig. 1.6 : Les 100 premiers chiffres du MNIST

Mais c'est en 2012 que la mèche s'allume définitivement. Au cours de la célèbre compétition annuelle ImageNet (Large

³https://fr.wikipedia.org/wiki/John_McCarthy

⁴https://fr.wikipedia.org/wiki/Marvin_Minsky

⁵https://fr.wikipedia.org/wiki/Conférence_de_Dartmouth

⁶https://fr.wikipedia.org/wiki/Warren_McCulloch

⁷https://fr.wikipedia.org/wiki/Walter_Pitts

⁸https://fr.wikipedia.org/wiki/Yann_Le_Cun

ScaleVisual Recognition Challenge 2012), un chercheur canadien, Geoffrey Hinton⁹, et son équipe utilisent le Deep Learning et explosent littéralement les scores de reconnaissance d'objets dans une image en passant d'un coup de 25 à 16 % d'erreur! Le succès est retentissant et, en moins de deux ans, les laboratoires du monde entier et des milliers de chercheurs vont adopter cette technologie et arriver à des scores de seulement quelques pour-cents d'erreur. L'IA que nous connaissons maintenant était née, mais encore fallait-il qu'elle sorte des laboratoires.



Fig. 1.7 : Geoffrey Hinton



Fig. 1.8 : Yoshua Bengio, Geoffrey Hinton et Yann Le Cun, lauréats du Prix Turing en 2019

C'est en 2016 que la société DeepMind, rachetée par Google et issue des laboratoires de l'université de Cambridge en Angleterre, crée le programme AlphaGo¹⁰ et bat le champion du monde du jeu de Go, Lee Sedol. Le monde entier découvre alors, grâce aussi à la puissance du marketing de Google, les progrès de l'intelligence artificielle fondée sur le Machine Learning et le Deep Learning, héritiers des premiers réseaux de neurones. C'est ce dernier événement qui va nous faire entrer de plain-pied dans cette nouvelle ère technologique où l'IA devient une réalité, quitte les laboratoires pour entrer dans des biens de la grande consommation comme le smartphone et, enfin, grâce à la création de nombreuses librairies open source facilitant le développement d'applications, arriver dans les entreprises.

L'histoire de l'intelligence artificielle nous raconte donc la capacité qu'a eue l'homme à réaliser l'analyse physiologique du fonctionnement du cerveau, l'analyse de son fonctionnement symbolique et les différentes modélisations mathématiques qui en ont découlé¹¹.

1.1.2 Définition de l'Intelligence Artificielle

L'intelligence artificielle¹² rassemble toutes les techniques permettant à des ordinateurs de simuler et de reproduire l'intelligence humaine. La notion d'intelligence artificielle fait son apparition dans le langage courant ces dernières années, mais on peut considérer qu'elle existe depuis que l'ordinateur fait tourner des algorithmes qui ne sont que des reproductions du raisonnement humain.

À l'image de l'humain, le cerveau et son intelligence sont là pour interagir avec leur environnement et c'est cette même interaction forte qui permettra au cerveau de se développer,¹³ L'intelligence est donc un tout indissociable que les théoriciens de l'intelligence artificielle ont décomposé, pour une meilleure compréhension, en différentes fonctions permettant de simuler l'ensemble des fonctions cognitives :

1. Les capacités de perception ou comment capter les flux d'information : l'ouïe, la vue...
2. La mémoire, l'apprentissage et la représentation de la connaissance.
3. Le calcul sur les représentations. La pensée.
4. Les capacités de communication expressives.

⁹https://fr.wikipedia.org/wiki/Geoffrey_Hinton

¹⁰<https://fr.wikipedia.org/wiki/AlphaGo>

¹¹https://fr.wikipedia.org/wiki/Prix_Turing

¹²https://fr.wikipedia.org/wiki/Intelligence_artificielle

¹³comme l'expliquent merveilleusement bien les neurologues Danièle Tritsch et Jean Mariani dans leur excellent livre «Ça va pas la tête! Cerveau, immortalité et intelligence artificielle».

5. Les capacités exécutives.

Ces 5 fonctions cognitives décrivent le large spectre d'utilisation de l'intelligence artificielle et permettent de mieux comprendre pourquoi on a l'impression d'en trouver « un peu partout ».

L'IA actuelle utilise, pour réaliser ces 5 fonctions cognitives, principalement 3 technologies que sont les moteurs de règles (SI... ALORS...), le Machine Learning et le Deep Learning, chacun étant un sous-ensemble de l'intelligence artificielle comme le montre le schéma suivant :

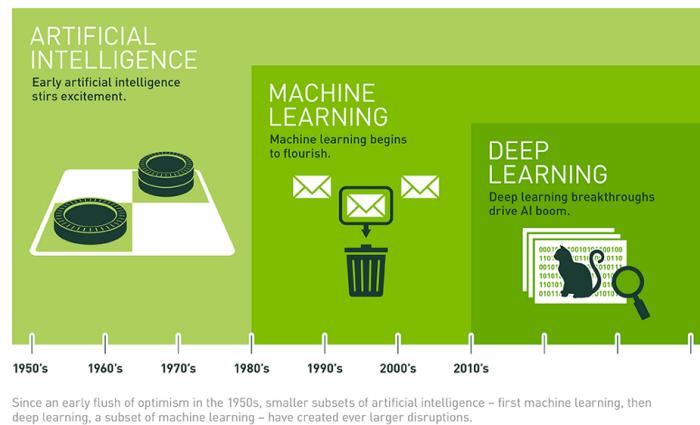


Fig. 1.9 : Les poupées russes

1.1.3 Les moteurs de règles

Les bonnes vieilles règles du type « Si... Alors... Sinon... » ont encore de beaux jours devant elles. Après avoir permis de créer les premiers systèmes experts, elles sont maintenant employées dans des environnements beaucoup plus simples pour décrire des raisonnements, digitaliser de l'expertise ou réaliser des workflows comme dans le RPA (Robotic Process Automation) dans des langages presque naturels que l'on peut retrouver ensuite sous forme d'arbre de décision. Ces moteurs de règles ont beaucoup d'avantages car ils permettent de séparer les traitements et les données, de centraliser et de rendre accessible la gestion de la connaissance et, surtout, de définir des règles compréhensibles et rédigables par des humains.

Cependant, le moteur de règles est limité par le savoir décrit par les règles. En effet, un moteur de règles n'est pas capable de générer ses propres règles, de modifier ses règles en fonction des modifications de son environnement... Un moteur de règles n'apprend pas, son savoir est statique. Comme la programmation, on le dit déterministe car son comportement est défini par la personne qui le programme, qui fait les règles. Nous verrons dans le chapitre suivant qu'il en est tout autrement avec le Machine Learning qui, comme son nom l'indique, va apprendre à partir d'exemples et construire un savoir non plus déterministe mais statistique.

C'est le fameux changement de paradigme.



Fig. 1.10 : Le nouveau paradigme

1.2 Le Machine Learning

1.2.1 Première approche

Nous l'avons vu dans sa définition, l'IA cherche à recréer des raisonnements humains. Il en est un qu'elle réussit plutôt pas mal : la généralisation d'un apprentissage.

Pourquoi pense-t-on d'abord à une Ferrari quand on voit une voiture de sport rouge ? Parce que la plupart des Ferrari que nous avons vues étaient rouges et que nous n'avons jamais vu de Lamborghini rouges, ou que très rarement pour tromper l'ennemi. En pensant immédiatement qu'une voiture de sport rouge est une Ferrari, nous généralisons ce que nous avons vu ou appris par notre expérience. Nous savons que nous avons une forte probabilité de ne pas nous tromper. C'est ce que l'on appelle un raisonnement par analyse statistique. Notre cerveau a créé son propre modèle statistique, une fonction de reconnaissance de voiture de sport !

De la même manière, après avoir programmé nos ordinateurs de façon déterministe au travers de longues listes d'instructions, nous allons maintenant avoir la possibilité de montrer à un ordinateur des exemples et lui demander d'en tirer une compréhension statistique à travers la création d'une fonction qui lui permettra de généraliser de la même manière que l'humain. C'est ce que nous appelons le machine learning.

Le machine learning est une technique qui rend possible la généralisation d'un raisonnement à partir d'exemples sans qu'il soit nécessaire de s'appuyer sur une équation pré-déterminée en tant que modèle. Les algorithmes de machine learning font appel à des méthodes de calcul qui leur permettent de créer leur propre fonction, de s'adapter et devenir plus performants à mesure que le nombre d'échantillons disponibles pour l'apprentissage augmente.

Prenons l'exemple d'un agent immobilier d'un arrondissement de Paris. De par son expérience sur le terrain, il a gardé à l'esprit les prix des dernières transactions en fonction de leurs spécificités (surface, orientation, étage, ancien, neuf...), si bien qu'il est capable de donner immédiatement un prix à quelqu'un qui lui demanderait. Il a donc construit intuitivement un modèle à partir d'un historique et d'exemples qu'il a tirés de sa pratique. Au cours de l'apprentissage de son métier, le cerveau de notre agent immobilier a créé, au fur et à mesure, une fonction qui, si on lui donne assez d'informations, est capable de donner le prix d'un bien immobilier avec une très faible marge d'erreur. Créer la fonction, c'est être capable de généraliser et de donner un prix, un résultat pour n'importe quelle donnée d'entrée.

C'est ce même apprentissage que les algorithmes de machine learning permettent de simuler. En parcourant les exemples du jeu d'apprentissage, le prix des transactions immobilières en fonction de la surface, de l'orientation, de l'étage... sur les trois dernières années (surface, orientation -> prix), l'algorithme va calculer lui aussi une fonction de manière à ce que si on lui donne les paramètres d'entrée (surface, orientation...), il puisse calculer le prix avec la plus faible marge d'erreur.

Une fois l'apprentissage réalisé nous obtenons un modèle, une fonction, capable de prédire le prix d'un bien immobilier à partir des paramètres en notre possession. En ayant créé ce modèle, nous avons mathématiquement simulé la démarche intellectuelle de notre agent immobilier, et donc réussi à approcher un raisonnement humain, ce qui est bien la définition de l'intelligence artificielle. Il ne reste plus qu'à l'intégrer dans l'environnement cible. Si c'est un site Web classique, le modèle donnera automatiquement le prix en fonction des données entrées par l'utilisateur. Si le site Web est un peu plus évolué, on pourra lui adjoindre un chatbot qui se chargera de l'interactivité avec l'utilisateur. L'apprentissage de ce modèle sera ensuite mis à jour périodiquement en intégrant les prix des dernières transactions pour qu'il intègre l'évolution de son environnement. À travers cet exemple, nous avons automatisé la fonction de calcul du prix d'un bien immobilier d'un agent immobilier et lui avons donné des fonctions d'interactivité, elles aussi issues de l'IA.

Ce premier exemple nous fait entrer de plain-pied dans l'intelligence artificielle et nous fait toucher du doigt pratiquement tous les concepts.

Notons que 90 % de l'intelligence artificielle de ces dernières années est fondée sur cette technique d'apprentissage. Connaissant les réponses du passé, on utilise ces exemples pour en déduire une fonction de prédiction, le modèle, qui va nous permettre de généraliser notre connaissance pour une utilisation future.

1.2.2 Les définitions du Machine Learning

Définition de Arthur Samuel en 1959

Le Machine Learning est la science qui permet à un ordinateur d'apprendre sans avoir été explicitement programmé pour cela.

Définition de Tom Mitchell en 1998

Une machine apprend quand sa performance à faire une certaine tâche s'améliore avec de nouvelles tâches

Définition intuitive

Le Machine Learning ou l'apprentissage automatique¹⁴ consiste à laisser l'ordinateur apprendre quel calcul effectuer, plutôt que de lui donner à faire ce calcul explicitement (c'est à dire le programmer) pour résoudre un problème donné.

Pour donner à une machine (un ordinateur, un robot,...) la capacité d'apprendre, par analogie avec les humains, on utilise des méthodes d'apprentissage.

Parmi ces méthodes on distingue trois grandes classes¹⁵ :

- (i) l'apprentissage supervisé (supervised Machine Learning)
- (ii) l'apprentissage non supervisé (unsupervised ML)
- (iii) l'apprentissage par renforcement (reinforcement learning)

1.2.3 L'apprentissage supervisé, qu'est-ce-que c'est ?

On parle d'apprentissage supervisé lorsqu'on fournit beaucoup d'exemples à une machine qu'elle doit étudier pour établir une relation entre ces exemples.

Il y a 4 notions fondamentales¹⁶ pour l'apprentissage supervisé :

- le DATASET
- le MODÈLE et ses PARAMÈTRES
- la FONCTION de COÛT
- l'ALGORITHME d'APPRENTISSAGE

Le MODÈLE est la tâche à accomplir; par exemple, reconnaître un animal sur une photo, prédire le prix d'une maison... C'est une représentation (simplifiée !) de la réalité. Cela peut donc prendre plusieurs formes, une courbe, un dessin, une équation...

Un modèle de Machine Learning est construit à partir de données. Ces données forment le DATASET. Dans les faits, c'est nous qui le choisissons.

Pour choisir le meilleur modèle, il faut donc mesurer la performance d'un modèle donné, c'est que l'on appelle la FONCTION de COÛT que l'on cherche à minimiser.

Le modèle prédit des valeurs. Pour le calcul de la fonction de coût, on mesure les erreurs commises entre ces prédictions et les vraies valeurs issues du dataset. Ce sont les METRIQUES qui permettent de mesurer cette différence. Il en existe de nombreuses.

L'ALGORITHME d'APPRENTISSAGE va permettre le calcul du minimum de la fonction de coût grâce à une partie des données du dataset qu'on appelle le jeu d'entraînement. On appelle cette étape, l'apprentissage. Il existe beaucoup d'algorithmes différents pour cet apprentissage. Le but de cette initiation est d'en apprendre quelques uns !

1.2.4 Illustrons cela sur un exemple

Voici un dataset de prix d'appartement. En général d'ailleurs, les dataset sont des tableaux Excel, tout simplement :

¹⁴https://fr.wikipedia.org/wiki/Apprentissage_automatique

¹⁵ Il en existe d'autres, mais on ne considère dans ce cours que ces trois principalement cf url note précédente

¹⁶ Retenez bien ces 4 notions, elles vont nous guider tout le long de ce cours !

id	nature	commune	parcelle	type	surface	piece	prix	latitude	longitude
1	2017-1	Vente	01053	01053000BK0039	Appartement	37	2	27000	46.20616
2	2017-5	Vente	01344	01344000AK0042	Appartement	22	1	258000	46.19745
3	2017-5	Vente	01344	01344000AK0042	Appartement	120	5	258000	46.19745
4	2017-5	Vente	01344	01344000AK0042	Appartement	22	1	258000	46.19745
5	2017-7	Vente	01053	01053000BM0425	Appartement	69	3	162000	46.21193
6	2017-10	Vente	01053	01053000AD0198	Appartement	38	1	177000	46.20526
7	2017-10	Vente	01053	01053000AD0198	Appartement	38	1	177000	46.20526
8	2017-28	Vente	01053	01053000AB0455	Appartement	88	3	380000	46.21051
9	2017-28	Vente	01053	01053000AB0455	Appartement	132	4	380000	46.21051
10	2017-28	Vente	01053	01053000AB0455	Appartement	51	2	380000	46.21051

Fig. 1.11 : Un exemple de dataset

Observons les colonnes : la colonne "prix" est notre cible (target). Les autres colonnes sont les caractéristiques (features) de l'appartement.

Le modèle c'est donc la prédiction du prix d'un appartement ayant des caractéristiques précises. On peut développer plusieurs types de modèles. Attention, dans ce cas précis, une représentation graphique est impossible car il y a trop de caractéristiques.

Le modèle le plus simple est un modèle linéaire, c'est à dire un modèle où si on note y la cible et x_1, x_2, x_3, \dots les caractéristiques on essaye d'écrire :

$$y = a_1x_1 + a_2x_2 + a_3x_3 + \dots + b$$

Mais on peut très bien imaginer prendre un modèle non linéaire, par exemple :

$$y = a_1x_1 + a_{12}x_1x_2 + a_{22}x_2^2 + \dots$$

On le voit de nombreux modèles sont possibles. Quand on peut les représenter graphiquement, parce que le nombre de caractéristiques est 1 ou 2, voire 3, alors c'est encore plus facile de choisir son modèle. Mais la plupart du temps, on ne peut pas.

Les a_i, a_{ij}, \dots sont les paramètres du modèle.

Une fois le modèle choisi, par exemple ici un modèle linéaire, faisons une présentation graphique en 2D. La fonction coût mesure la somme des erreurs commise comme l'illustre le schéma ci-dessous

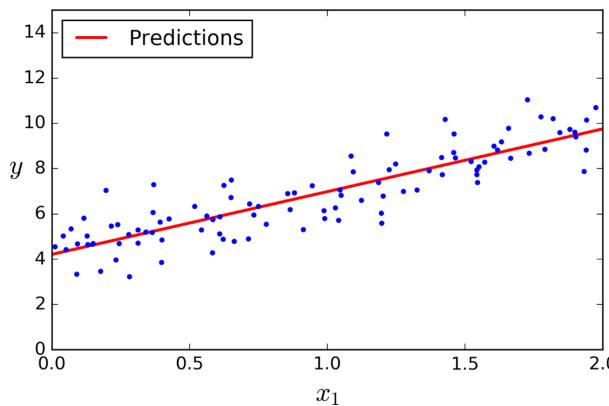


Fig. 1.12 : Régression linéaire

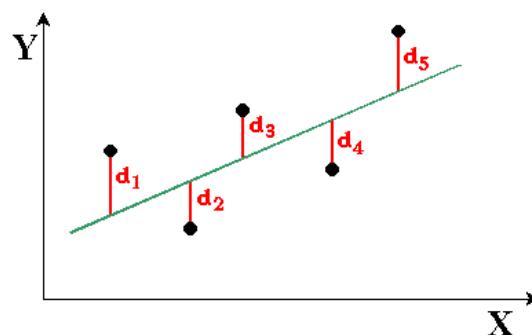


Fig. 1.13 : Les erreurs

Un "bon" modèle est un modèle qui commet de petites erreurs et donc dont la fonction coût est petite.

APPRENDRE (notion d'apprentissage donc), c'est minimiser cette fonction coût, une fois le modèle choisi; ainsi, l'apprentissage supervisé c'est trouver les paramètres du modèle qui minimisent la fonction de coût.

Il existe deux grands types de problèmes d'apprentissage supervisé. :

- LA RÉGRESSION : on cherche à prédire la valeur d'une variable continue pouvant donc potentiellement prendre une infinité de valeurs, par exemple on cherche à prédire le prix d'un loyer.
- LA CLASSIFICATION : on cherche à prédire une valeur discrète ne pouvant donc prendre qu'un nombre fini de valeurs, par exemple, on cherche à classer un objet dans une catégorie.

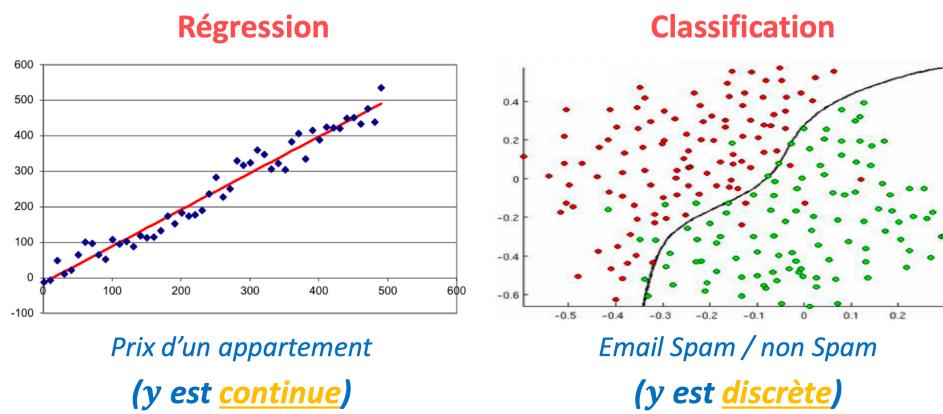


Fig. 1.14 : Régression versus classification

1.2.5 L'apprentissage non supervisé, c'est vraiment différent ?

L'apprentissage non supervisé c'est apprendre sans exemple de ce qu'il faut apprendre ! Dans l'apprentissage non supervisé, on dispose bien d'un dataset X mais sans cible y et la machine apprend donc à reconnaître des structures dans les données X . On peut aussi, c'est ce que fait par exemple l'algorithme de Netflix, regrouper des données dans des CLUSTERS, c'est d'ailleurs ce que l'on appelle le CLUSTERING. L'apprentissage non supervisé permet également la détection d'anomalies, par exemple les fraudes à la carte bancaire.

1.2.6 L'apprentissage par renforcement, si, si, vous connaissez... !

Vous avez déjà vu le film "Un jour sans fin" ou plus récemment "Edge of Tomorrow" ? Et bien ces films sont l'illustration parfaite de ce qu'est l'apprentissage par renforcement. Cette méthode s'inspire de la façon dont on éduque les animaux domestiques en leur offrant une récompense s'ils font la bonne action commandée et une punition, on parle d'ailleurs plutôt de pénalité, dans le cas contraire, sans y être obligé.

1.2.7 Un beau dessin...

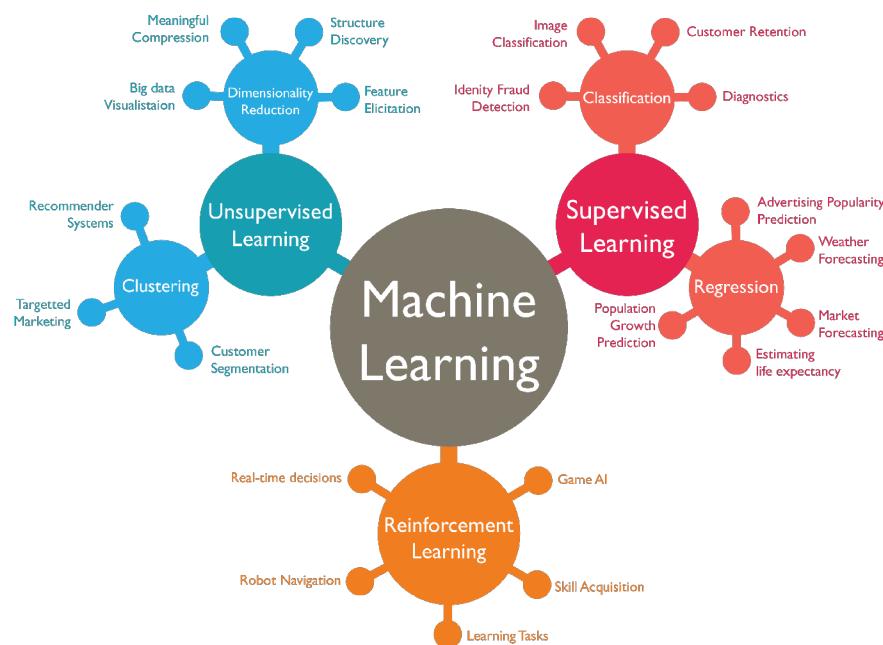


Fig. 1.15 : Les trois grands types d'apprentissage machine

1.3 Les données

Vous l'avez compris, sans données, pas d'apprentissage machine. On dit depuis quelques temps que les données, les data, sont le nouveau pétrole ou le nouvel or noir du XXI^e siècle. L'analogie a du sens, même si les dégâts qu'ont causé les énergies carbonées ne soient (pas encore) à la hauteur de ceux produits par les données.

Les GAFAM¹⁷ (Google, Amazon, Facebook, Apple, Microsoft) et leur pendants chinois les BATX¹⁸ (Baidu, Alibaba, Tencent, Xiaomi) l'ont parfaitement compris depuis quelques années. Les quantités de données, produites par vous, nous... sont accaparées par ces sociétés pour en faire du business. Mais ces sociétés ne cherchent plus seulement à capter toutes nos données, mais à orienter, modifier et conditionner tous nos comportements : notre vie social, nos émotions, nos pensées les plus intimes¹⁹, ... jusqu'à notre bulletin de vote. En un mot, décider à notre place à des fins strictement lucratives.

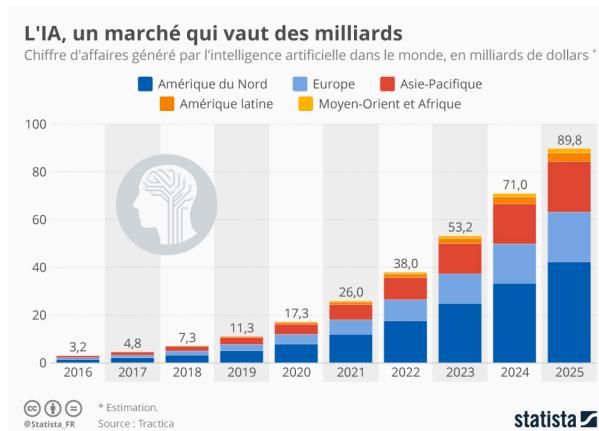


Fig. 1.16 : Un marché très très juteux

¹⁷<https://fr.wikipedia.org/wiki/GAFAM>

¹⁸<https://fr.wikipedia.org/wiki/BATX>

¹⁹ ce que Shoshana Zuboff appelle "le surplus comportemental de chaque individu" https://fr.wikipedia.org/wiki/Shoshana_Zuboff

Ces quantités de données phénoménales sont appelées **BIG DATA**²⁰ et on parle alors souvent de la règle des 3V (voire 5V, voire 9V !) pour décrire les qualités qui leur sont nécessaires pour servir les projets d'intelligence artificielle.



Fig. 1.17 : En une minute !

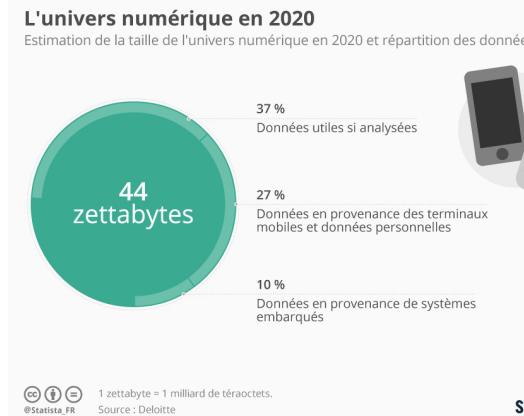


Fig. 1.18 : 44 zettabit de données produites en 2020, c'est à dire 44 milliards de téra-octets !

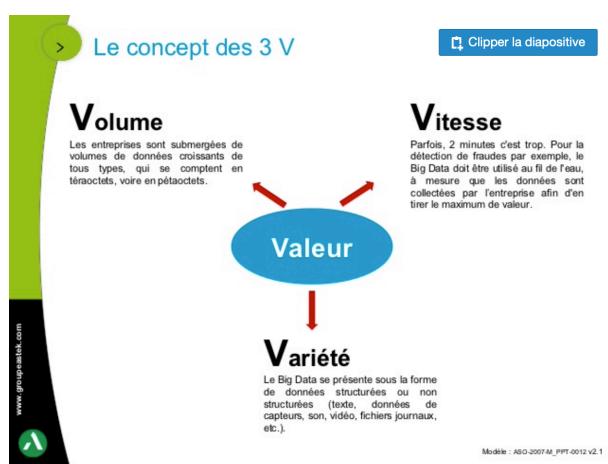


Fig. 1.19 : Les 3V du Big Data

Heureusement, il n'y a pas que les GAFAM et BATX qui collectent nos données. Pour contrer cette main mise de sociétés privées, les états ont inventés l'**OPEN DATA**.²¹ Le terme d'*open data* est apparu pour la première fois en 1995, dans un document d'une agence scientifique américaine. Il y est question de l'ouverture des données géophysiques et environnementales. « Notre atmosphère, les océans et la biosphère forment un ensemble intégré qui transcende les frontières », écrivent les auteurs de ce rapport. Ils promeuvent un échange complet et ouvert des données scientifiques entre les différents pays, condition indispensable de l'analyse et de la compréhension de ces phénomènes globaux.

L'idée de **BIEN COMMUN** appliquée aux connaissances a déjà été théorisé, et ceci bien avant l'invention d'Internet. Robert King Merton²² est l'un des pères de la sociologie des sciences. La théorie qui porte son nom montre le bénéfice de l'ouverture des données scientifiques. Il importe, explique Merton dès 1942, que les résultats des travaux soient accessibles

²⁰https://fr.wikipedia.org/wiki/Big_data

²¹https://fr.wikipedia.org/wiki/Données_ouvertes

²²https://fr.wikipedia.org/wiki/Robert_King_Merton

à tous librement. Chaque chercheur doit contribuer au « pot commun » et renoncer aux droits de propriété intellectuelle pour permettre l'avancée de la connaissance.

Les technologies de l'information ont par ailleurs donné un nouveau souffle à la philosophie des biens communs. La lauréate du "prix Nobel" d'économie 2009, Elinor Orstrom, a démontré par ses travaux la particularité des biens communs informationnels. Ceux-ci s'apparentent à des biens publics, car leur usage par l'un ne remet pas en cause l'usage par l'autre. Mais ce sont des biens publics d'un genre inédit : non seulement leur utilisation n'appauvrit pas le stock commun, mais au contraire il l'enrichit.

Bien avant de constituer un objet technique ou politique, le mouvement open data prend ainsi racine dans les milieux scientifiques. Ce sont les chercheurs qui ont perçu les premiers le bénéfice de l'ouverture et du partage des données.

Mais c'est la rencontre de cette idée scientifique avec les idéaux du logiciel libre et de l'open source qui façonne l'open data tel qu'il se met en place aujourd'hui.



Fig. 1.20 : L'open data

1.4 Quelques fondamentaux *avant de plonger dans la programmation !*

1.4.1 Petit rappel : Apprentissage Supervisé et problème de Régression

Si vous cherchez à prédire le cours de la bourse, le prix d'un appartement, ou bien l'évolution de la température sur Terre, alors vous cherchez en fait à résoudre un problème de régression. Si vous disposez d'un dataset (x, y) alors vous pouvez utiliser l'apprentissage supervisé pour développer un modèle de régression.

Regardons comment un modèle de Machine Learning se construit.

1.4.2 Voici la recette à suivre pour réaliser votre premier modèle de Machine Learning

Récolter vos données

Imaginez que plusieurs agences immobilières vous aient fourni des données sur des appartements à vendre, notamment le prix de l'appartement (y) et la surface habitable (x). En Machine Learning, on dit que vous disposez de m exemples d'appartements. On désigne par :

$x^{(i)}$ la surface habitable de l'exemple (appartement) i

$y^{(i)}$ le prix de l'exemple (appartement) i

En visualisant (ici c'est possible donc profitons-en !) votre dataset, vous obtenez le nuage de points suivant :

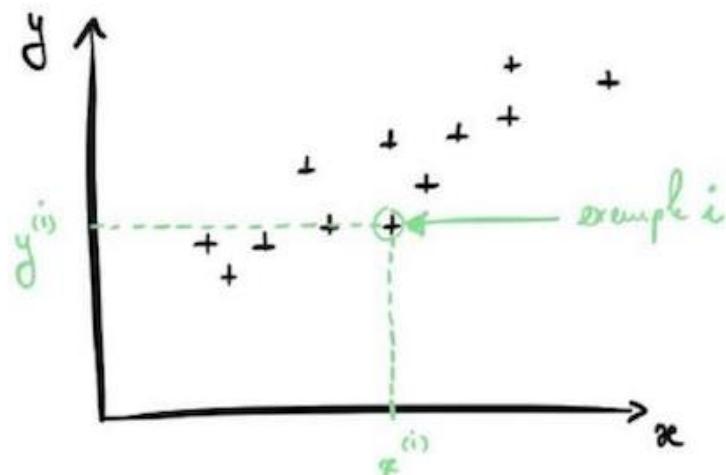


Fig. 1.21 : loyers en fonction de la surface

Créer un modèle linéaire

A partir de ces données, on développe un modèle linéaire $f(x) = ax + b$ où a et b sont les paramètres du modèle. Un bon modèle donne de petites erreurs entre ses prédictions $f(x)$ et les exemples (y) du dataset. Nous ne connaissons pas les valeurs des paramètres a et b , ce sera le rôle de la machine de les trouver, de sorte à tracer un modèle qui s'insère bien dans notre nuage de point comme ci-dessous :

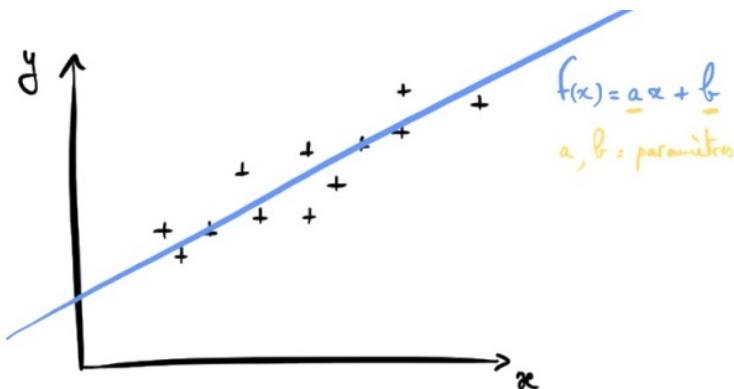


Fig. 1.22 : Modèle

Définir La Fonction Coût

Pour la régression linéaire, on utilise la norme euclidienne pour mesurer les erreurs entre $f(x)$ et (y) . Concrètement, voici la formule pour exprimer l'erreur i entre le prix $y^{(i)}$ et la prédition faite en utilisant la surface $x^{(i)}$: $\text{erreur}^{(i)} = ((x^{(i)}) - y^{(i)})^2$

Par exemple, imaginez que le 10ième exemple de votre dataset soit un appartement de $x^{(10)} = 80^2$ dont le prix s'élève à $y^{(10)} = 100000$ € et que votre modèle prédise un prix de $f(x^{(10)}) = 100,002$ €.

L'erreur pour cette exemple est donc : $\text{erreur}^{(10)} = 4$

Chaque prédition s'accompagne d'une erreur, on a donc m erreurs. On définit la FONCTION COÛT $J(a, b)$ comme étant la moyenne de toutes les erreurs :

$$J(a, b) = \frac{1}{2m} \sum_{i=1}^m ((x^{(i)}) - y^{(i)})^2$$

On appelle cette erreur Mean Squared Error (MSE) c'est à dire l'erreur quadratique moyenne. C'est une des métriques que l'on utilisera souvent.

Trouver les paramètres qui minimisent la fonction de coût

La prochaine étape est l'étape la plus excitante, il s'agit de laisser la machine apprendre quels sont les paramètres qui minimisent la Fonction de Coût, c'est-à-dire les paramètres qui nous donnent le meilleur modèle.

Pour trouver le minimum, on utilise un algorithme d'optimisation qui s'appelle Gradient Descent (la descente de gradient).

Pour bien comprendre la descente de gradient, imaginez-vous perdu en montagne. Votre but est de rejoindre le refuge qui se trouve au point le plus bas de la vallée. Vous n'avez pas pris de carte avec vous donc vous ne connaissez pas les coordonnées de ce refuge, vous devez le trouver tout seul.

Pour vous en sortir, voici une stratégie à adopter :

1. Depuis votre position actuelle, vous partez en direction de là où la pente descend le plus fort.
2. Vous avancez une certaine distance en suivant cette direction coûte que coûte (même si ça implique de remonter une pente)
3. Une fois cette distance parcourue, vous répétez les 2 premières opérations en boucle, jusqu'à atteindre le point le plus bas de la vallée.



Fig. 1.23 : Illustration de la descente de gradient

Les étapes 1, 2 et 3 forment ce qu'on appelle l'algorithme de Gradient Descent. Cet algorithme vous permet de trouver le minimum de la Fonction Coût $J(a, b)$ (le point le plus bas de la montagne) en partant de coordonnées a et b aléatoires (votre position initiale dans la montagne) :

1. Calculer la pente de la Fonction Coût, c'est-à-dire la dérivée de $J(a, b)$.
2. Évoluer d'une certaine distance α dans la direction de la pente la plus forte. Cela a pour résultat de modifier les paramètres a et b .
3. Recommencer les étapes 1 et 2 jusqu'à atteindre le minimum de $J(a, b)$.

Pour illustrer l'algorithme, voyez le dessin ci-dessous, où on montre la recherche du paramètre a idéal (la même chose s'applique au paramètre b)

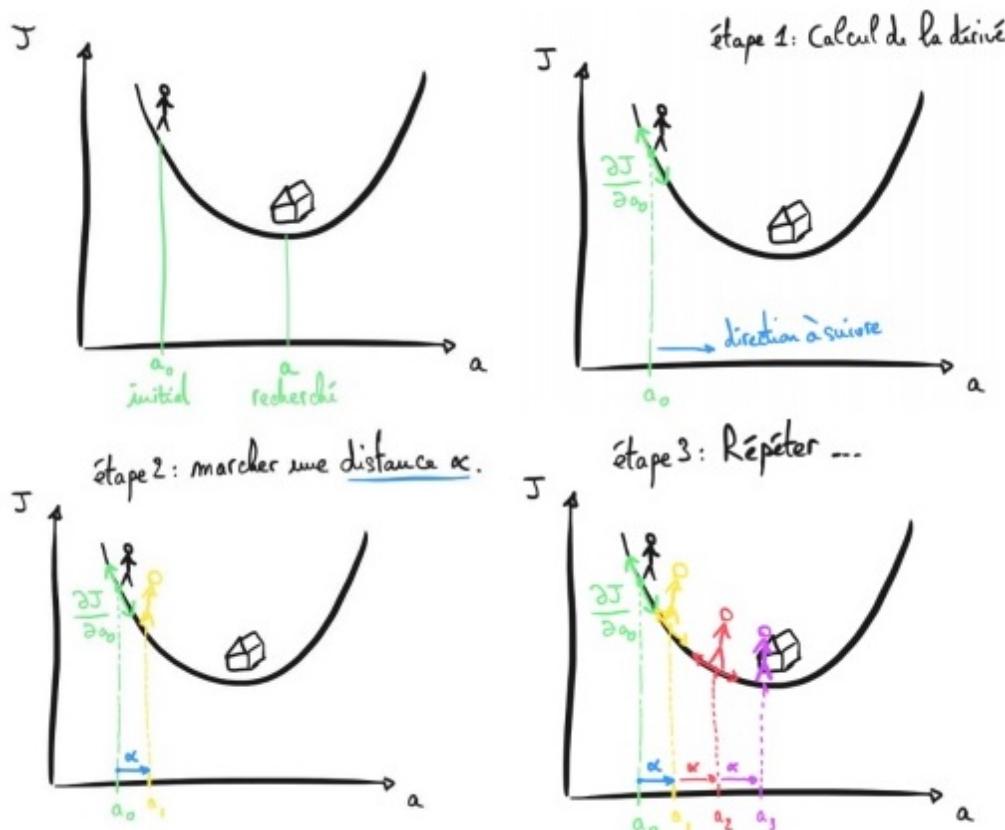


Fig. 1.24 : Illustration de la descente de gradient

C'est cet algorithme qui est utilisé systématiquement dans toutes les librairies de programmation pour les calculs des modèles de machine learning ; il s'agit d'un processus itératif où α est la vitesse d'apprentissage (learning rate) de la boucle d'itérations suivantes :

$$\begin{cases} a = a - \alpha \frac{\partial J(a, b)}{\partial a} \\ b = b - \alpha \frac{\partial J(a, b)}{\partial b} \end{cases}$$

Cet α est ce que l'on appelle un hyper-paramètre. A la différence des paramètres a et b , qui sont appris par la machine, c'est à vous de le régler pour faire en sorte que l'apprentissage aboutisse à la découverte du minimum de la fonction de coût. Et c'est une vraie difficulté. Car si la vitesse est trop lente, c'est à dire α petit, alors la distance parcourue à chaque étape du processus itératif est très petite et on converge très très doucement vers le minimum. En revanche, si la vitesse est trop grande, c'est à dire α grand, alors la distance parcourue est trop grande et on peut rebondir de paroi en paroi et ne jamais converger vers ce minimum recherché.

A vous donc de trouver le bon compromis... !

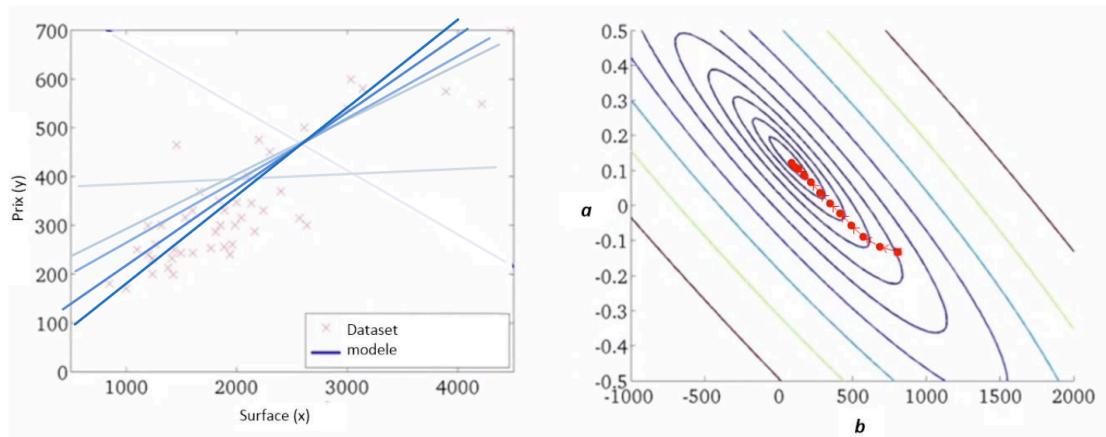


Fig. 1.25 : Quand le learning rate est bon...

Le calcul des dérivées partielles pour la descente de gradient s'effectue en utilisant la notation matricielle qui nous suivra dorénavant tout le long de ce cours. Car, vous l'avez compris, nous parlons de dataset composés de m exemples et avec n caractéristiques.

Dans la pratique, on exprime donc notre dataset et nos paramètres sous forme matricielle, ce qui simplifie beaucoup les calculs. On crée ainsi un vecteur $\theta = \begin{pmatrix} a \\ b \end{pmatrix} \in \mathbb{R}^{n+1}$ qui contient tous les paramètres pour notre modèle, un vecteur $y \in \mathbb{R}^{m \times 1}$ et une matrice $X \in \mathbb{R}^{m \times n}$ qui inclut toutes les features n . Dans la régression linéaire, $n = 1$

Résumé des étapes pour développer un programme de Régression Linéaire qui nous servira comme modèle pour le reste !

La recette de la régression linéaire :

1. Récolter des données $(X, y) \in \mathbb{R}^{m \times 1}$
2. Donner à la machine un modèle linéaire du type $F(X) = X \cdot \theta$ où $\theta = \begin{pmatrix} a \\ b \end{pmatrix}$
3. Créer la Fonction de Coût : $J(\theta) = \frac{1}{2m} \sum (F(X) - y)^2$
4. Calculer le gradient et utiliser l'algorithme de Gradient Descent en répétant en boucle jusqu'à la convergence :

$$\theta = \theta - \alpha \times \frac{\partial J(\theta)}{\partial \theta}$$

où le gradient est :

$$\frac{\partial J(\theta)}{\partial \theta} = \frac{1}{m} X^T \cdot (F(X) - y)$$

Cette dernière formule est tout simplement obtenue grâce à la formule de la dérivée d'une composée, à savoir $(f \circ g)' = (f' \circ g) \times g'$.

Passons maintenant à la programmation...

1.5 Python, le langage du dataScientist

Python²³ est le langage de programmation le plus utilisé dans le domaine du Machine Learning, du Big Data et de la Data Science.

²³<https://www.python.org/>

Créé en 1991, le langage de programmation Python apparu à l'époque comme une façon d'automatiser les éléments les plus ennuyeux de l'écriture de scripts ou de réaliser rapidement des prototypes d'applications.

Depuis quelques années, toutefois, ce langage de programmation s'est hissé parmi les plus utilisés dans le domaine du développement de logiciels, de gestion d'infrastructure et d'analyse de données. Il s'agit d'un élément moteur de l'explosion du Big Data.

Python est un langage de programmation open source créé par le programmeur Guido van Rossum en 1991. Il tire son nom de l'émission Monty Python's Flying Circus.

Il s'agit d'un langage de programmation interprété, qui ne nécessite donc pas d'être compilé pour fonctionner. Un programme "interpréteur" permet d'exécuter le code Python sur n'importe quel ordinateur. Ceci permet de voir rapidement les résultats d'un changement dans le code. En revanche, ceci rend ce langage plus lent qu'un langage compilé comme le C. En tant que langage de programmation de haut niveau, Python permet aux programmeurs de se focaliser sur ce qu'ils font plutôt que sur la façon dont ils le font. Ainsi, écrire des programmes prend moins de temps que dans un autre langage.

Le langage Python doit sa popularité à plusieurs avantages qui profitent aussi bien aux débutants qu'aux experts. Tout d'abord, il est facile à apprendre et à utiliser. Ses caractéristiques sont peu nombreuses, ce qui permet de créer des programmes rapidement et avec peu d'efforts. De plus, sa syntaxe est conçue pour être lisible et directe.

Un autre avantage du Python est sa popularité. Ce langage fonctionne sur tous les principaux systèmes d'exploitation et plateformes informatiques. De plus, même s'il ne s'agit clairement pas du langage le plus rapide, il compense sa lenteur par sa versatilité.

Le langage Python doit sa popularité à plusieurs avantages qui profitent aussi bien aux débutants qu'aux experts. Tout d'abord, il est facile à apprendre et à utiliser. Ses caractéristiques sont peu nombreuses, ce qui permet de créer des programmes rapidement. De plus, sa syntaxe est conçue pour être lisible et directe.

Un autre avantage du Python est sa popularité. Ce langage fonctionne sur tous les principaux systèmes d'exploitation et plateformes informatiques. De plus, même s'il ne s'agit clairement pas du langage le plus rapide, il compense sa lenteur par sa versatilité.

Python est le langage le plus utilisé pour la Data Science. Pour cause, ce langage est simple, lisible, propre, flexible et compatible avec de nombreuses plateformes. Ses nombreuses bibliothèques, telles que TensorFlow, Pandas et Numpy permettent d'effectuer une large variété de tâches. Ainsi, selon un sondage mené en 2020 par O'Reilly, 74

Il permet le prototypage rapide, et le code peut être exécuté n'importe où : Windows, macOS, UNIX, Linux... sa flexibilité permet de prendre en charge le développement de modèles de Machine Learning, le forage de données, la classification et bien d'autres tâches plus rapidement que les autres langages.

Des bibliothèques comme Scrapy et BeautifulSoup permettent d'extraire des données depuis internet, tandis que Seaborn et Matplotlib aident à la Data Visualization. De leur côté, Tensorflow, Keras et Theano permettent le développement de modèles de Deep Learning, et Scikit-Learn aide au développement d'algorithmes de Machine Learning.

Si le Python s'est érigé comme le meilleur langage de programmation pour le Big Data, c'est grâce à ses différents packages et bibliothèques de science des données. Voici les plus populaires.

1.5.1 Pandas

Pandas²⁴ est l'une des bibliothèques de science des données les plus populaires. Elle a été développée par des Data Scientists habitués au langage R et au Python, et est aujourd'hui utilisée par un grand nombre de scientifiques et d'analystes.

Elle offre de nombreuses fonctionnalités natives très utiles. Il est notamment possible de lire des données en provenance de nombreuses sources, de créer de larges dataframes à partir de ces sources, et d'effectuer des analyses agrégées basées sur les questions auxquelles on souhaite obtenir des réponses.

Des fonctionnalités de visualisation permettent également de générer des graphiques à partir des résultats des analyses, ou de les exporter au format Excel. On peut aussi s'en servir pour la manipulation de tableaux numériques et de séries temporelles.

1.5.2 Numpy

NumPy²⁵ est un package utilisé pour les calculs scientifiques en Python. Il est idéal pour les opérations liées à l'algèbre linéaire, aux transformations de Fourier, ou au crunching de nombres aléatoires.

Il peut être utilisé en guise de container multi-dimensionnel de données génériques. De plus, il s'intègre facilement avec de nombreuses bases de données différentes.

²⁴<https://pandas.pydata.org/>

²⁵<https://numpy.org/>

1.5.3 Matplotlib et Seaborn

Matplotlib²⁶ est une bibliothèque destinée à tracer et visualiser des données sous formes de graphiques. Elle peut être combinée avec les bibliothèques python de calcul scientifique NumPy et SciPy.

Seaborn²⁷ est également une bibliothèque destinée à tracer et à visualiser des données sous forme graphique. Néanmoins, elle est plus élégante pour des représentations "simples" de dataset mais moins complète que Matplotlib.

1.5.4 Scikit-Learn

Scikit-Learn²⁸ est LA bibliothèque pour les algorithmes Machine Learning, de classification, de régression ou de clustering.

Cette bibliothèque de Machine Learning pour Python se révèle complémentaire pour les autres bibliothèques telles que NumPy et SciPy. Elle est développée par l'INRIA²⁹.

1.5.5 TensorFlow

Développé par Google Brain, TensorFlow³⁰ est une bibliothèque de Deep Learning. Ses graphiques de data flow et son architecture flexible permettent d'effectuer des opérations et des calculs de données à l'aide d'une API unique sur de multiples CPU ou GPU depuis un PC, un serveur ou même un appareil mobile.

1.5.6 Jupyter et les notebooks

Les notebooks Jupyter³¹ sont des cahiers électroniques qui, dans le même document, peuvent rassembler du texte, des images, des formules mathématiques et du code informatique exécutable. Ils sont manipulables interactivement dans un navigateur web.

Initialement développés pour les langages de programmation Julia, Python et R (d'où le nom Jupyter), les notebooks Jupyter supportent près de 40 langages différents.

La cellule est l'élément de base d'un notebook Jupyter. Elle peut contenir du texte formaté au format Markdown ou du code informatique qui pourra être exécuté.

Pour un mini tutoriel sur les Jupyter Notebook, voir : https://python.sdv.univ-paris-diderot.fr/18_jupyter/

1.5.7 1er notebook, utilisation des bibliothèques de base pour la data science

Téléchargez ce notebook que vous trouverez sur mon GitHub³² : https://github.com/jmbernabotto/INSA2021/blob/main/numpy_et_pandas_et_matplotlib.ipynb

²⁶<https://matplotlib.org/>

²⁷<https://seaborn.pydata.org>

²⁸<https://scikit-learn.org/stable/>

²⁹<https://inria.fr/fr>

³⁰<https://www.tensorflow.org/?hl=fr>

³¹<https://jupyter.org/>

³²GitHub permet aux développeurs de stocker et de partager, publiquement ou non, le code qu'ils créent. La plate-forme accueille ainsi plus de 80 millions de projets, qu'il s'agisse de logiciels, de sites Web, d'applications pour mobile ou tous autres types de programme informatique — et ce quel que soit le langage de programmation utilisé. Le site est aussi un espace collaboratif. Chaque utilisateur peut contribuer aux projets mis en ligne publiquement sur GitHub, en proposant des modifications. Le succès de GitHub repose notamment sur la façon dont le site a facilité ce processus. Pour que les utilisateurs ne se dérangent pas mutuellement en modifiant un programme en même temps, ils téléchargent chacun de son côté le code sur son ordinateur, effectuent les modifications, qui sont ensuite publiées sur GitHub après validation. Le site se base pour cela sur Git, un outil développé en 2005 par Linus Torvalds, le célèbre créateur de Linux — GitHub aura eu pour talent de rendre ce système plus simple d'utilisation et compréhensible par le plus grand nombre. Chaque modification du code est ainsi stockée sur GitHub, et il est possible de suivre pas à pas chaque étape de développement d'un programme. Des espaces de discussion permettent à tous les développeurs d'échanger sur chaque programme et contribution.

1.5.8 2ème notebook, utilisation de la bibliothèque Scikit-Learn

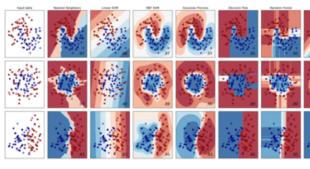
scikit-learn
Machine Learning in Python

Getting Started Release Highlights for 0.24 GitHub

• Simple and efficient tools for predictive data analysis
 • Accessible to everybody, and reusable in various contexts
 • Built on NumPy, SciPy, and matplotlib
 • Open source, commercially usable - BSD license

Classification
Identifying which category an object belongs to.

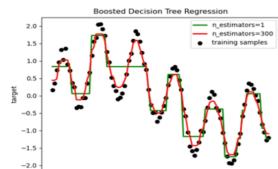
Applications: Spam detection, image recognition.
Algorithms: SVM, nearest neighbors, random forest, and more...



Examples

Regression
Predicting a continuous-valued attribute associated with an object.

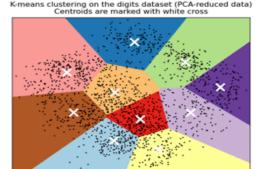
Applications: Drug response, Stock prices.
Algorithms: SVR, nearest neighbors, random forest, and more...



Examples

Clustering
Automatic grouping of similar objects into sets.

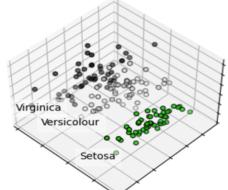
Applications: Customer segmentation, Grouping experiment outcomes
Algorithms: k-Means, spectral clustering, mean-shift, and more...



Examples

Dimensionality reduction
Reducing the number of random variables to consider.

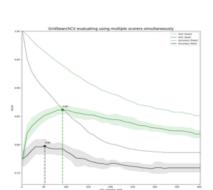
Applications: Visualization, Increased efficiency
Algorithms: k-Means, feature selection, non-negative matrix factorization, and more...



Examples

Model selection
Comparing, validating and choosing parameters and models.

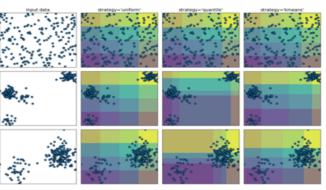
Applications: Improved accuracy via parameter tuning
Algorithms: grid search, cross validation, metrics, and more...



Examples

Preprocessing
Feature extraction and normalization.

Applications: Transforming input data such as text for use with machine learning algorithms.
Algorithms: preprocessing, feature extraction, and more...



Examples

Fig. 1.26 : Le portail de Scikit-Learn

La bibliothèque Scikit-Learn utilise la programmation de type objet pour l'implémentation de tous les algorithmes de Machine Learning qu'elle intègre. La structure d'utilisation de ces objets est toujours la même :

`modele=Classe(hyper-paramètres)` où `modele` est l'**objet de la classe** `Classe` du modèle choisi parmi tous les modèles disponibles ; on l'appelle **estimateur**.

3 méthodes sont présentes dans toutes les classes. "fit", "score" et "predict"

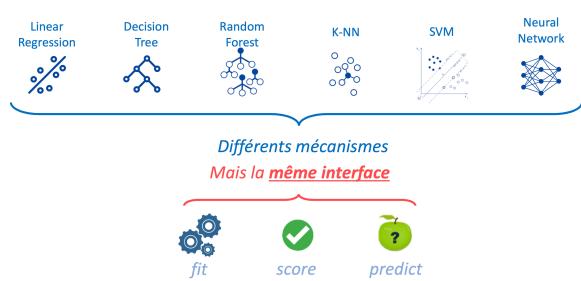


Fig. 1.27 : Interface de Scikit-Learn

1. Sélectionner un **estimateur** et préciser ses **hyperparamètres** :

$model = \underbrace{\text{LinearRegression}(\dots)}_{\substack{\text{objet} \\ \text{Constructeur}}}$ $\underbrace{\dots}_{\text{Hyperparamètres}}$

2. Entrainer le modèle sur les données X, y (divisées en 2 tableaux **Numpy**)

$model.fit(X, y)$

3. Évaluer le modèle

$model.score(X, y)$

4. Utiliser le modèle

$model.predict(X)$

Téléchargez ce notebook que vous trouverez sur mon GitHub : https://github.com/jmbernabotto/INSA2021/blob/main/scikit_learn_1.ipynb

Chapitre 2

Machine Learning avec Scikit-Learn

2.1 Test et validation

La seule façon de savoir comment un modèle va fonctionner sur de nouveaux cas, c'est de l'essayer effectivement sur ceux-ci. Une des façons de procéder consiste à mettre notre modèle en production et à surveiller ses résultats. C'est efficace, mais si votre modèle est très mauvais, vos utilisateurs se plaindront. Ce n'est donc pas la meilleure idée.

Une solution préférable consiste à partager vos données en deux ensembles : le **jeu d'entraînement** et le **jeu de test**. Comme ces noms le laissent entendre, vous entraînez votre modèle sur le jeu d'entraînement, et vous le testez sur le jeu de test. Le taux d'erreur sur les nouveaux exemples est appelé erreur de généralisation (ou erreur hors échantillon), et en évaluant votre modèle sur le jeu de test, vous obtenez une estimation de cette erreur : celle-ci vous indique comment votre modèle se comportera sur des cas qu'il n'a jamais rencontrés auparavant. Si l'erreur d'apprentissage est faible (c'est-à-dire si votre modèle fait peu d'erreurs sur le jeu d'entraînement) mais si l'erreur de généralisation est élevée, cela signifie que votre modèle surajuste les données d'entraînement.

Théorème 2.1.1 (Théorème du « No Free Lunch¹ »). *Un modèle est une version simplifiée des observations. Les simplifications ont pour but de faire disparaître les détails superflus qui ne se généraliseront vraisemblablement pas aux nouvelles observations. Cependant, pour décider quelles données supprimer ou sauvegarder, vous devez faire des hypothèses. En choisissant un modèle linéaire, vous faites par exemple l'hypothèse que les données sont fondamentalement linéaires et que la distance entre les observations et la ligne droite ne correspond qu'à un bruit qui peut sans danger être ignoré.*

Théorème 2.1.2. *En Machine Learning on ne valide jamais un modèle pour la production en le testant sur des données qu'il a déjà vues*

On utilise communément 80 % des données pour l'entraînement, et on en met 20 % de côté pour les tests.

Imaginons que nous ayons un jeu de données d'images de chat et d'autres animaux. Le modèle est de bien classer les images dans les catégories "CHAT" et "AUTRES".

¹Dans un célèbre article publié en 1996 David Wolpert a démontré que si vous ne faites absolument aucune hypothèse sur les données, alors il n'y a aucune raison de préférer un modèle à un autre. C'est le théorème du No Free Lunch (que l'on peut traduire par « pas de repas gratuit », ce qui signifie qu'on n'a rien sans rien). Pour certains ensembles de données, le meilleur modèle est un modèle linéaire, tandis que pour d'autres, c'est un réseau neuronal. Il n'y a pas de modèle qui soit a priori fonctionner mieux : la seule façon de le savoir avec certitude consiste à les évaluer tous (d'où le nom du théorème). Vu que ce n'est pas possible, vous devrez en pratique faire quelques hypothèses raisonnables à propos de vos données et évaluer seulement quelques modèles appropriés. Ainsi, pour de simples tâches vous pourrez évaluer des modèles linéaires avec différents niveaux de régularisation, alors que pour un problème complexe vous pourrez évaluer différents réseaux neuronaux.

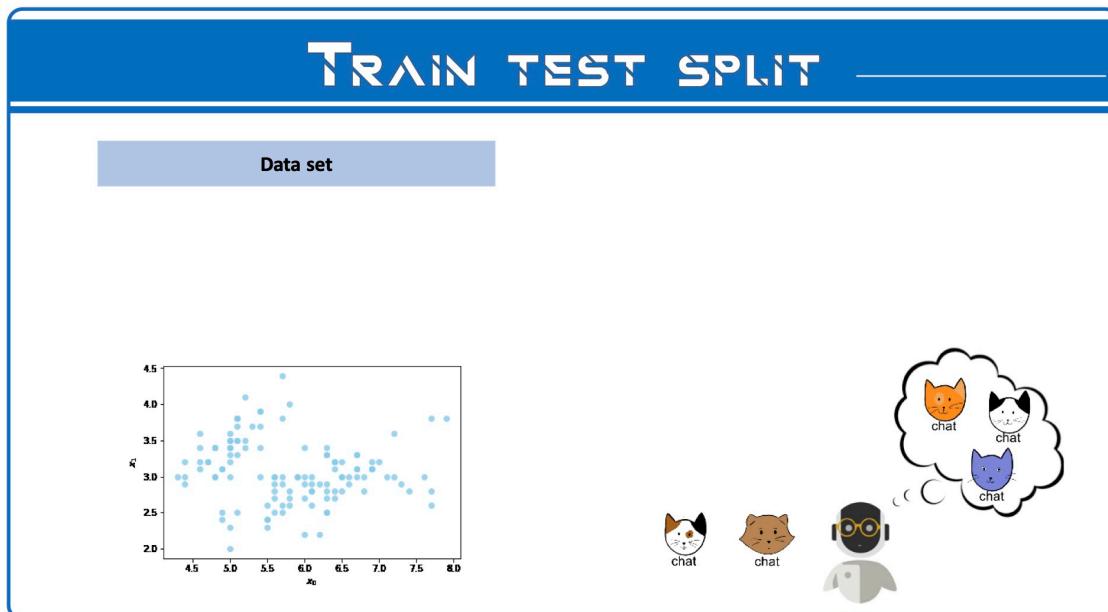
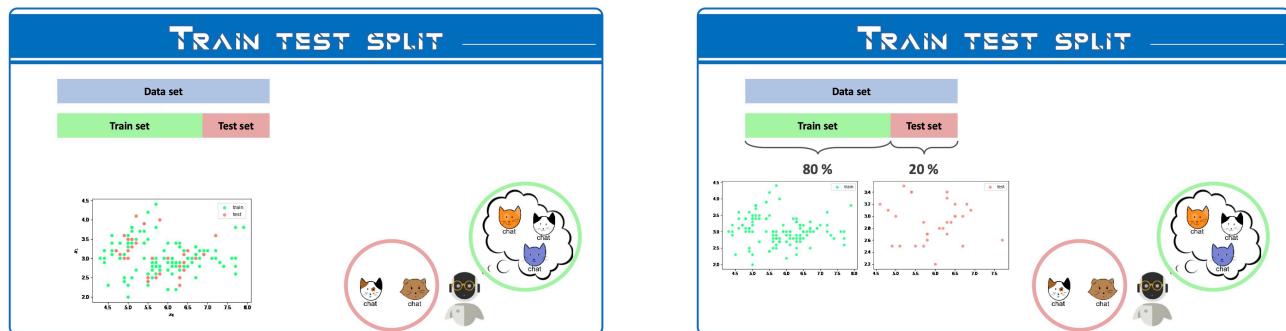
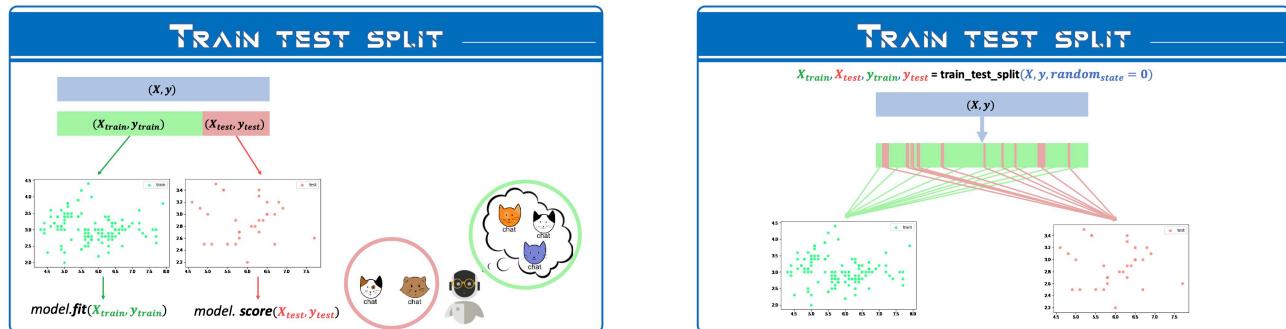


Fig. 2.1 : Le jeu de données

On sépare donc le jeu de données



L'évaluation d'un modèle est donc très simple : il suffit d'utiliser un jeu de test (avec Scikit-Learn par exemple !)

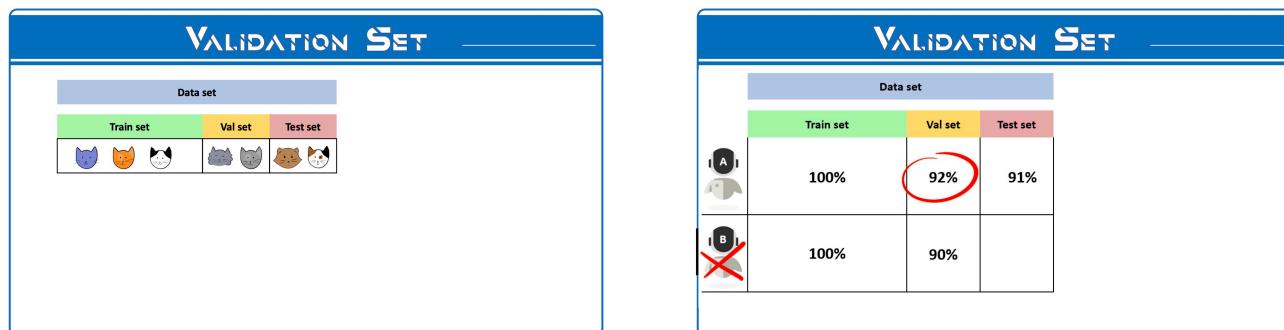


Supposons maintenant que vous hésitez entre deux modèles (par exemple un modèle linéaire et un modèle polynomial) : comment décider ? Une solution consiste à entraîner les deux et à comparer la façon dont ils se généralisent à l'aide du jeu de test.

Vous placez alors ce modèle en production, mais malheureusement il ne se comporte pas aussi bien qu'attendu et produit 15 % d'erreurs. Que s'est-il passé ?

Le problème, c'est que vous avez mesuré l'erreur de généralisation à de multiples reprises sur le jeu de test, et vous avez adapté le modèle et ses hyperparamètres pour obtenir le meilleur modèle pour ces données. Ceci signifie qu'il est peu probable que le modèle obtienne d'aussi bons résultats sur de nouvelles données.

Une solution courante à ce problème est d'avoir un second jeu en réserve appelé **jeu de validation**. Vous entraînez plusieurs modèles avec différents hyperparamètres sur le jeu d'entraînement, vous sélectionnez le modèle et les hyperparamètres qui donnent les meilleurs résultats sur le jeu de validation puis, quand vous êtes satisfait de votre modèle, vous effectuez un test final unique sur le jeu de test pour obtenir une estimation de l'erreur de généralisation.



Pour éviter de gaspiller trop de données d'entraînement dans les jeux de validation, on utilise couramment une technique de validation croisée : le jeu d'entraînement est partagé en sous-ensembles complémentaires, chaque modèle est entraîné sur une combinaison différente de ces sous-ensembles puis validé sur les sous-ensembles restants. Une fois le type de modèle et les hyperparamètres sélectionnés, un modèle final utilisant ces hyperparamètres est entraîné sur le jeu d'entraînement complet et l'erreur de généralisation est mesurée sur le jeu de test.

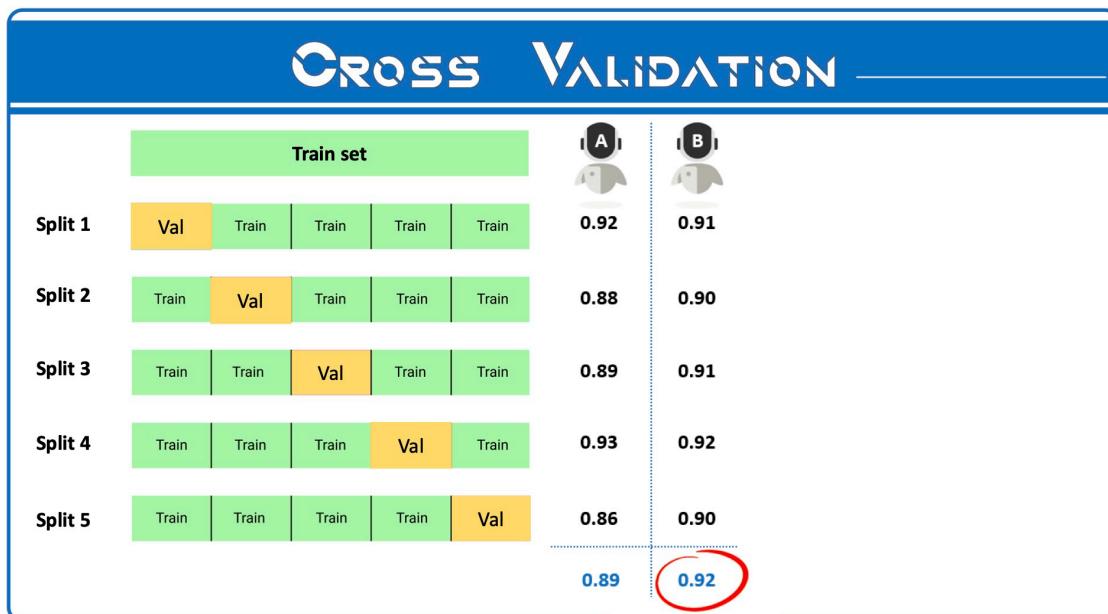


Fig. 2.2 : La validation croisée

Il existe plusieurs stratégies pour la Cross-Validation.

- **KFold**. On découpe des sections égales les unes après les autres. Inconvénient : pose problème avec les classes déséquilibrées
- **Leave one Out** On prend toutes les données sauf une et on évalue sur cette donnée, et on recommence autant de fois qu'il y a de données. Inconvénient : très gourmand en ressource

- **Shuffle Split** On prend le jeu d'entraînement entier et on distribue aléatoirement une portion de données pour créer un nouveau jeu d'entraînement et le reste pour le jeu de validation. Et on recommence un certain nombre de fois. Inconvénient : si le découpage pour le jeu de validation est petit, on risque d'avoir 1 split sans données d'une des classes.
- **Stratified KFold** Choix par défaut dans Scikit-Learn. On crée des splits avec une petite portion de chacune des classes.
- **Group KFold** A choisir quand les données dépendent les unes des autres.

2.2 Généralisation, surapprentissage, sous-apprentissage

2.2.1 Le biais et la variance en statistiques

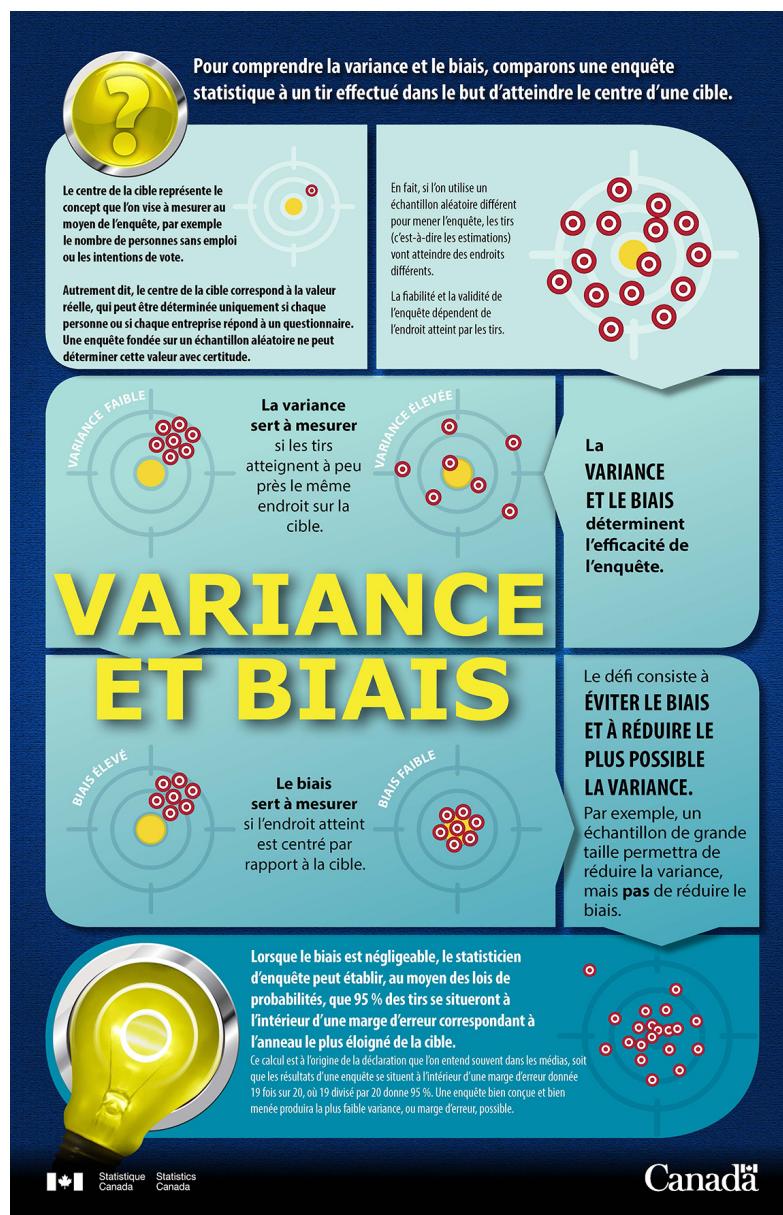


Fig. 2.3 : Biais-Variance

2.2.2 La généralisation

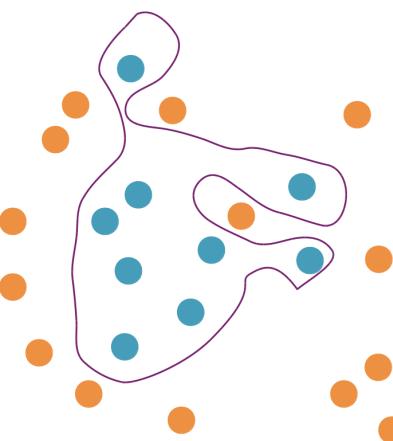
Un bon modèle de machine learning, c'est un modèle qui généralise.

La généralisation, c'est la capacité d'un modèle à faire des prédictions non seulement sur les données que vous avez utilisées pour le construire, mais surtout sur de nouvelles données : c'est bien pour ça que l'on parle d'apprentissage !

Pour vous donner un exemple un peu extrême, imaginez que vous vouliez catégoriser des images : certaines représentent des chats, d'autres non. Pour construire votre modèle, vous disposez d'un jeu de données de 500 images de chats, et 500 images qui ne sont pas des chats. Vous pouvez construire un algorithme très simple : pour classer une image, regarder pixel par pixel si elle est exactement identique à une de celles dans vos données. Si c'est le cas, retournez l'étiquette (chat ou pas-chat) associée. Dans le cas contraire, répondez au hasard. Cet algorithme marche très bien sur le jeu de données initial, mais n'a rien appris du tout : il ne sait pas faire de prédictions ! Évaluer un modèle sur le jeu de données sur lequel on l'a construit ne nous permet donc pas du tout de savoir comment il se comportera sur de nouvelles données, celles sur lesquelles il est vraiment intéressant de faire de la prédiction.

2.2.3 Surapprentissage et compromis biais-variance

Sans tomber dans le cas extrême décrit plus haut, on peut facilement se retrouver sans le vouloir avec un modèle qui colle trop aux données sur lesquelles on apprend (aussi appelées "jeu d'entraînement"), et qui soit trop sensible à leurs moindres variations pour bien les représenter. Un tel modèle aura de très bonnes performances sur le jeu d'entraînement mais sera mauvais sur de nouvelles données. On parle alors de sur-apprentissage (overfitting en anglais).



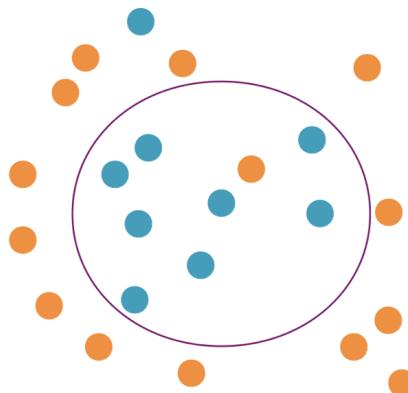
Sur cet exemple, le modèle (la ligne violette) qui sépare les points bleus des points oranges colle bien aux données, il ne fait aucune erreur. Mais est-ce qu'il modélise bien la réalité ?

Un modèle qui sur-apprend est un modèle qui est trop complexe par rapport à la réalité qu'il essaie de représenter. Nous avons tendance à préférer des modèles simples ; c'est le principe du rasoir d'Ockham², selon lequel les hypothèses suffisantes les plus simples sont les plus vraisemblables.

Par ailleurs, coller de trop près aux données est une mauvaise idée car elles sont inévitablement bruitées :

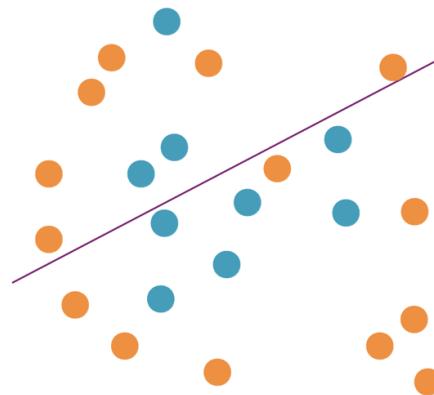
- Par des erreurs de mesure (les appareils que nous utilisons pour mesurer les variables qui représentent nos données peuvent faire des erreurs techniques) ;
- Par des erreurs d'étiquetage (l'erreur est humaine, et il se peut que certaines des étiquettes ne soient pas les bonnes) ;
- Parce que nous n'avons pas mesuré les variables les plus pertinentes, soit parce qu'on ne les connaît pas, soit parce qu'elles sont très compliquées à mesurer.

²https://fr.wikipedia.org/wiki/Guillaume_d%27Ockham



Ce modèle fait des erreurs sur le jeu d'apprentissage, mais il va probablement mieux généraliser

Il faut néanmoins aussi éviter les modèles trop simples, qui ne parviendront pas à bien représenter le phénomène qui nous intéresse, et qui ne feront pas de bonnes prédictions. On parle dans ce cas de "sous-apprentissage".



Ce modèle, trop simple, représente trop mal les données pour prédire

Le concept de compromis biais-variance nous permet de bien résumer la situation : un modèle simple (variance faible) risque le sous-apprentissage (biais élevé y compris sur les données d'entraînement). Un modèle complexe (variance élevée) risque le sur-apprentissage (biais faible sur les données d'entraînement mais élevé sur de nouvelles données).

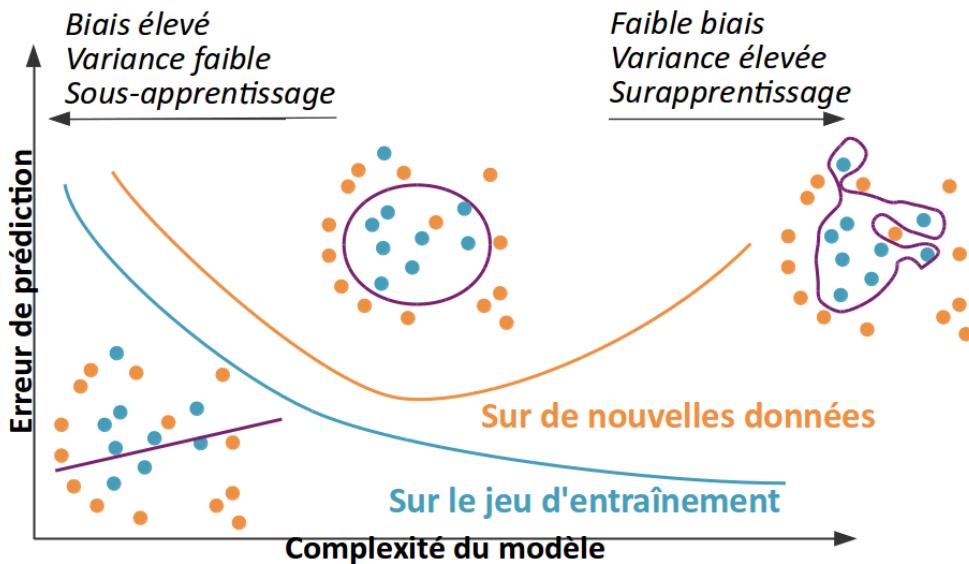


Fig. 2.4 : Compromis Biais-Variance : on souhaite trouver un modèle intermédiaire, vers le creux de la courbe orange, là où le biais de prédiction est le plus faible et la généralisation la meilleure.

La complexité d'un modèle peut être mesurée de différentes façons. L'une d'entre elles est le nombre de paramètres utilisés par ce modèle : plus il y a de paramètres, plus le modèle est complexe.

2.2.4 Les problèmes mal posés

Pourquoi ne peut-on pas choisir *a priori* la complexité de notre modèle ? En fait, toutes nos difficultés viennent du fait que les problèmes d'apprentissage sont des problèmes mal posés (ill-posed en anglais). Nous n'arrivons pas à les formuler de sorte à avoir une solution unique. Les données que nous observons ne sont pas suffisantes pour modéliser correctement le phénomène que nous cherchons à comprendre pour répondre à notre problématique, et nous devons rajouter des hypothèses (par exemple, que la frontière séparant les points bleus des points orange est un cercle) pour finalement arriver à un bon modèle. Ces hypothèses forment ce que l'on appelle le biais d'induction (inductive bias en anglais).

Attention ! Vous pouvez être amenés à faire des compromis sur la complexité d'un modèle pour respecter des contraintes sur le temps de calcul (que ce soit pour l'entraînement ou pour la prédiction) ou les ressources en mémoire. Votre application nécessite-t-elle d'utiliser un modèle que vous devrez faire tourner pendant une heure pour produire une réponse, si un autre modèle peut vous donner une réponse de qualité légèrement inférieure en quelques secondes ?

2.2.5 En résumé

- En apprentissage supervisé, le but est de produire des modèles qui généralisent, c'est-à-dire qui sont capables de faire de bonnes prédictions sur de nouvelles données
- De bonnes performances sur le jeu d'entraînement ne garantissent pas que le modèle sera capable de généraliser !
- On cherche à développer un modèle qui soit suffisamment complexe pour bien capturer la nature des données (et éviter ainsi le sous-apprentissage), mais suffisamment simple pour éviter le sur-apprentissage.

- Attention aux contraintes de temps de calcul et aux ressources en mémoire !
- En cas de surapprentissage on peut :
 - Essayer un modèle plus simple !
 - Réduire la dimensionnalité du jeu de données (cf chapitre sur PCA)
 - Ajouter, si c'est possible bien-sûr, des données au jeu d'entraînement
 - Régulariser. La régularisation d'un modèle permet de limiter sa variance sans sacrifier son biais. Il existe trois types de régularisation couramment employées.
 - * La régularisation $L1$, dite Lasso où l'on cherche

$$\min \left| C|w| + \frac{1}{N} \sum_{i=1}^N (f_{w,b}(x_i) - y_i)^2 \right|$$

Elle produit un modèle creux c'est à dire un modèle où la plupart des paramètres sont nuls si l'hyper-paramètre C est suffisamment grand. Cela revient à faire de la sélection de variables en décidant de l'annexe et du superflu !

- * La régularisation $L2$, dite Ridge où l'on cherche

$$\min \left| C\|w\|^2 + \frac{1}{N} \sum_{i=1}^N (f_{w,b}(x_i) - y_i)^2 \right|$$

Si l'objectif est de maximiser la performance du modèle, alors la régression Ridge donne de meilleurs résultats.

- * La régularisation Elastic Net qui est la somme de la régularisation Lasso et de la régularisation Ridge avec deux hyper-paramètres à régler C_1 et C_2 dans

$$\min \left(\left| C_1|w| + \frac{1}{N} \sum_{i=1}^N (f_{w,b}(x_i) - y_i)^2 \right| + \left| C_2\|w\|^2 + \frac{1}{N} \sum_{i=1}^N (f_{w,b}(x_i) - y_i)^2 \right| \right)$$

2.3 Les métriques

Les métriques sont utilisées pour évaluer les performances du modèle choisi initialement, une fois la phase d'entraînement terminée. Dans le cas de l'apprentissage supervisé, on va donc mesurer, d'où le terme de métriques, la différence entre les prédictions faites par le modèle, qu'on note en général \hat{y} , et les vraies valeurs du jeu d'entraînement, qu'on note y .

Il ne faut pas les confondre avec la fonction de coût qui elle mesure, pendant l'entraînement, l'erreur commise entre les prédictions et les vraies valeurs.

2.3.1 Métriques pour la régression

Observons cette visualisation sur le github : [metrique de regression.pdf](#)

Il en existe 4 essentielles :

- La MSE : Mean Squared Error : $\frac{1}{m} \sum (\hat{y} - y)^2$; pénalise les grandes erreurs car l'importance des erreurs est exponentielle à son amplitude.
- La MAE : Mean Absolute Error : $\frac{1}{m} \sum |\hat{y} - y|$; l'importance de l'erreur est proportionnelle à son amplitude.
- Le coefficient de corrélation : $R^2 = 1 - \frac{\sum (\hat{y} - y)^2}{\sum (\bar{y} - y)^2}$; c'est celui par défaut dans Scikit-Learn avec l'instruction `modele.score(X, y)` avec les estimateurs de régression. Ce coefficient évalue la performance du modèle par rapport au niveau de variations présent dans les données. C'est celui que l'on connaît bien et qui est utilisé en statistiques pour la régression linéaire par exemple. Plus R^2 est proche de 1, mieux c'est. Un coefficient de détermination de 0,74 signifie que le modèle choisi décrit 74 % des variations des données.
- La Median Absolute Error : `median|\hat{y} - y|`. Utilisation plus rare.

2.3.2 Les métriques pour la classification

Elles sont beaucoup plus nombreuses. Matrice de confusion³, accuracy-score⁴ (justesse, exactitude), recall-score (rapport)⁵, precision-score (précision), f1-score, roc-curve⁶, roc-auc-score...

Fig. 2.5 : Matrice de confusion

Voyons les dans un notebook Jupyter en Python. `matrice_de_confusion_ROC_AUC.ipynb`

Mais alors, puisqu'elles sont si nombreuses, laquelle utiliser ? Et bien justement, puisqu'elles sont nombreuses, autant les utiliser toutes ! Évidemment, selon le contexte, vous verrez que certaines sont plus utiles que d'autres. mais n'oubliez pas les autres pour autant !

Une astuce de programmation dans Scikit-Learn permet de les avoir toutes :

```
import sklearn.metrics
sorted(sklearn.metrics.SCORERS.keys())
dont voici la sortie :
```

```
'accuracy',
'adjusted_mutual_info_score',
'adjusted_rand_score',
'average_precision',
'balanced_accuracy',
'completeness_score',
'explained_variance',
'f1',
'f1_macro',
'f1_micro',
'f1_samples',
'f1_weighted',
'fowlkes_mallows_score',
'homogeneity_score',
'jaccard',
```

³https://fr.wikipedia.org/wiki/Matrice_de_confusion

⁴https://en.wikipedia.org/wiki/Accuracy_and_precision

⁵https://fr.wikipedia.org/wiki/Pr%C3%A9cision_et_rappel

⁶https://fr.wikipedia.org/wiki/Courbe_ROC

```
'jaccard_macro',
'jaccard_micro',
'jaccard_samples',
'jaccard_weighted',
'max_error',
'mutual_info_score',
'neg_brier_score',
'neg_log_loss',
'neg_mean_absolute_error',
'neg_mean_gamma_deviance',
'neg_mean_poisson_deviance',
'neg_mean_squared_error',
'neg_mean_squared_log_error',
'neg_median_absolute_error',
'neg_root_mean_squared_error',
'normalized_mutual_info_score',
'precision',
'precision_macro',
'precision_micro',
'precision_samples',
'precision_weighted',
'r2',
'recall',
'recall_macro',
'recall_micro',
'recall_samples',
'recall_weighted',
'roc_auc',
'roc_auc_ovr',
'roc_auc_ovr_weighted',
'roc_auc_ovr_weighted',
'v_measure_score'
```

2.4 Quelques précisions sur les métriques de classification

2.4.1 Compromis Précision-Rappel

Revenons sur la classification qui est donc un des modèles de Machine Learning Supervisé et en particulier sur la matrice de confusion. Reprenons pour ce propos, le jeu de données MNIST. Pour simplifier cette discussion, on va considérer une reconnaissance sur le chiffre 5 uniquement. Il s'agit donc d'une classification binaire où l'on a un 5 ou pas un 5.

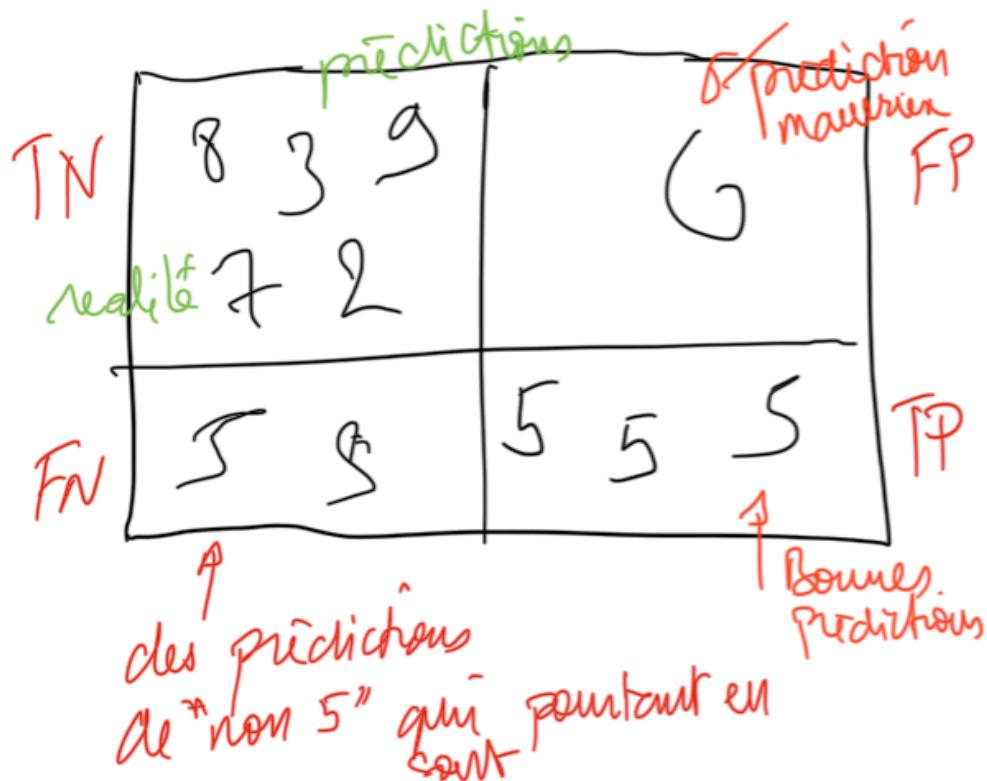


Fig. 2.6 : Une matrice de confusion

Si on les compte, voici les résultats numériques :

	prédictions pas 5	prédictions 5
réel pas 5	53272	1307
réel 5	1077	4344

Calculons l'exactitude des prédictions :

$$\text{Exactitude} = \text{Accuracy} = \frac{53272 + 4344}{60000} = 96\%$$

Calculons la précision, c'est à dire l'exactitude des prédictions positives, les 5.

$$\text{Precision} = \frac{TP}{TP + FP} = \frac{4344}{1307 + 4344} = 77\%$$

Calculons le rappel, c'est à dire le taux de positifs ayant été correctement détectés, prédits :

$$\text{Rappel} = \frac{TP}{TP + FN} = \frac{4344}{4344 + 1077} = 80\%$$

Calculons maintenant le F1-Score qui favorise les classificateurs ayant une précision et un rappel équivalents.

$$\text{F1-score} = \frac{2}{\frac{1}{\text{precision}} + \frac{1}{\text{rappel}}} = 78\%$$

Mais ce n'est pas forcément ce que l'on souhaite. Par exemple, imaginons que nous souhaitions un classificateur pour détecter les vidéos sans danger pour les enfants de moins de 12 ans. On préférera un faible rappel, donc on rejettéra des "bonnes" vidéos (au sens de positives) mais on souhaite conserver uniquement les vidéos vraiment sans danger, c'est à dire que l'on souhaite une grande précision.

En revanche, si on veut détecter des fraudes, il faut mieux un grand rappel (oui, on aura alors beaucoup de faux positifs, c'est à dire des transactions qui ne sont pas des fraudes mais détectées comme telles) mais on peut se contenter d'une précision faible afin de bien capturer toutes les fraudes.

Malheureusement, vous l'avez compris, accroître la précision diminue le rappel et inversement. C'est ce que l'on appelle le compromis précision/rappel.

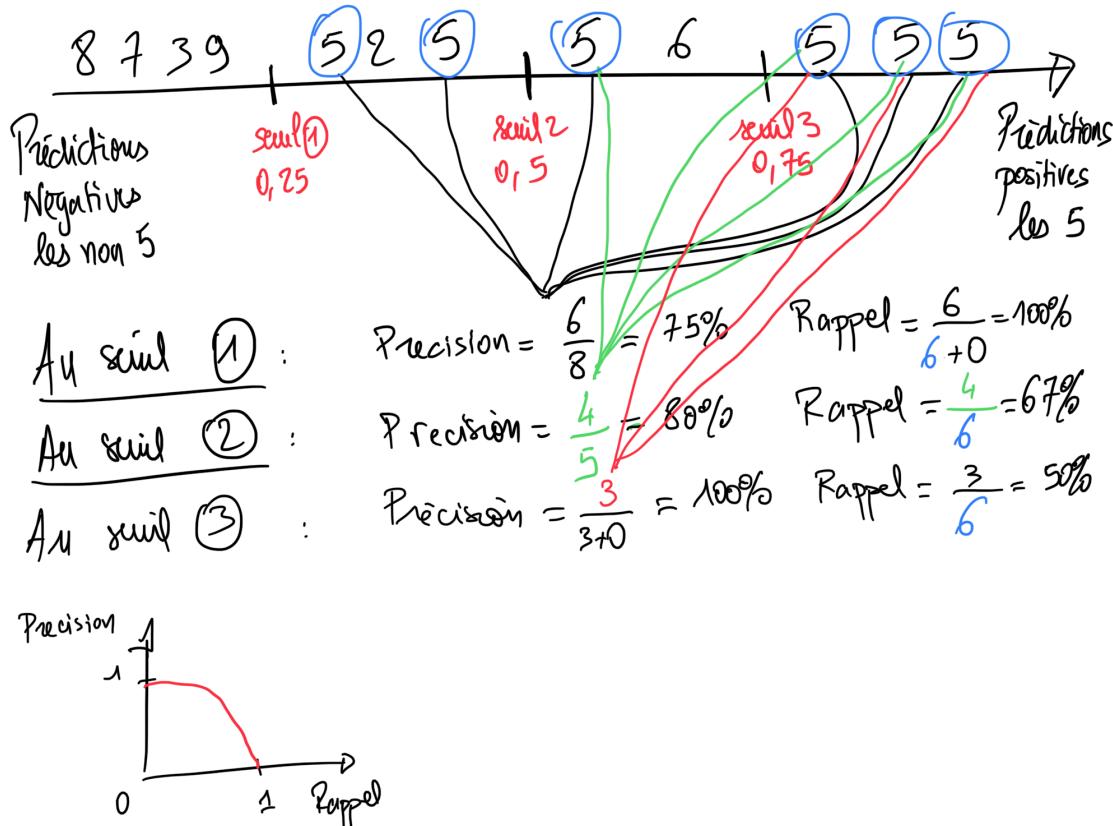


Fig. 2.7 : Compromis précision/rappel

2.4.2 Courbe ROC-AUC

La fonction d'efficacité du récepteur, plus fréquemment désignée sous le terme « courbe ROC » (de l'anglais receiver operating characteristic, pour « caractéristique de fonctionnement du récepteur ») dite aussi caractéristique de performance (d'un test) ou courbe sensibilité/spécificité, est une mesure de la performance d'un classificateur binaire, c'est-à-dire d'un système qui a pour objectif de catégoriser des éléments en deux groupes distincts sur la base d'une ou plusieurs des caractéristiques de chacun de ces éléments. Graphiquement, on représente souvent la mesure ROC sous la forme d'une courbe qui donne le taux de vrais positifs (fraction des positifs qui sont effectivement détectés) en fonction du taux de faux positifs (fraction des négatifs qui sont incorrectement détectés). On calcule ces deux taux avec des seuils qui passent progressivement de 0 à 1.

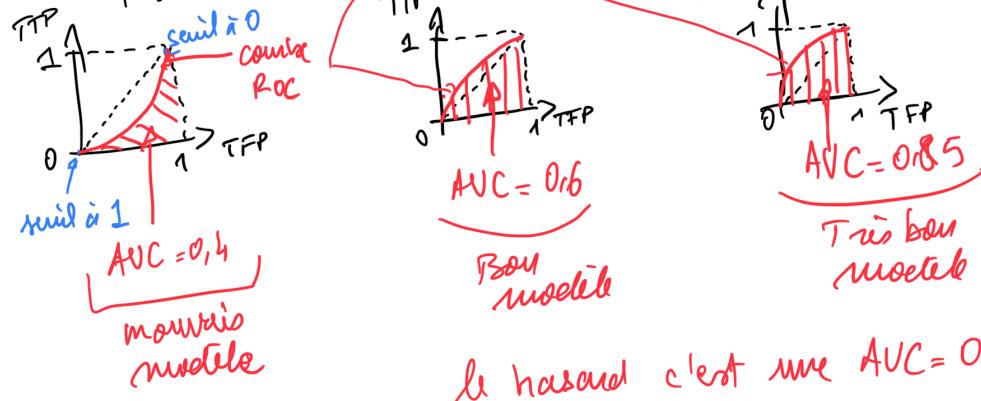
taux de vrais positifs

$$TTP = \frac{TP}{TP+FN} = \text{Rappel}$$

taux de faux positifs

$$TFP = \frac{FP}{FP+TN}$$

On calcule ces deux taux avec des seuils qui on fait passer de 0 à 1. On a alors plusieurs cas de figures



Le hasard c'est une $AUC = 0,5$

Fig. 2.8 : Courbe ROC-AUC

2.5 Les algorithmes les plus fréquemment utilisés pour l'apprentissage supervisé

Les différents types de classification

- Régression linéaire
- Régression logistique
- Arbres de décisions
- SVM
- KNN
- La méthode GridSearch
- Le préprocessing
- Le préprocessing avancé

2.6 Les algorithmes les plus fréquemment utilisés pour l'apprentissage non supervisé

- PCA
- KMean
- Métriques du KMean
- Isolation Forest