INTRODUCTION AU MACHINE LEARNING

Sorelle Perelle NGUIAKAM

21 juillet 2025

Table des matières

INTRODUCTION

Le Machine Learning (Apprentissage Automatique) est une discipline qui permet aux ordinateurs d'apprendre sans être explicitement programmés. Contrairement à la programmation traditionnelle où l'on fournit un calcul à l'ordinateur, le Machine Learning est utilisé pour résoudre des problèmes où le calcul précis est inconnu, comme la reconnaissance faciale, la prédiction boursière, ou la composition musicale. Son inventeur, Arthur Samuel, l'a défini en 1959 comme "la science qui permet aux ordinateurs d'apprendre sans être explicitement programmés". Aujourd'hui, le Machine Learning est omniprésent dans notre quotidien, intervenant par exemple lorsque nous utilisons Google, Netflix, YouTube ou Amazon. Il permet de diagnostiquer des cancers, de bloquer des spams et des virus, et est essentiel au développement des voitures autonomes. Apprendre le Machine Learning, souvent confondu à tort avec l'Intelligence Artificielle par le grand public, est une nouvelle compétence professionnelle qui offre des opportunités professionnelles extraordinaires et développe la capacité à résoudre des problèmes, quel que soit votre métier. Il consiste à laisser la machine apprendre quel calcul effectuer à partir d'expériences, s'inspirant de la manière dont les êtres humains apprennent.

1 LES FONDATIONS DU MACHINE LEARNING

Cette première partie, intitulée "**Les fondations du Machine Learning**" , est cruciale pour comprendre les bases de cette discipline et préparer le terrain pour les concepts plus avancés qui suivront. Elle a pour objectif de démystifier le Machine Learning en expliquant pourquoi il est utilisé et comment la machine apprend.
Pour permettre aux ordinateurs d'apprendre à partir d'expériences, cette section introduit trois méthodes d'apprentissage principales :

1.1 L'apprentissage supervisé

L'apprentissage supervisé (Supervised Learning) est la méthode la plus courante. Elle consiste à fournir à la machine de nombreux exemples (appelés un Dataset) sous forme de paires (x, y), où 'x' sont les caractéristiques (**features**) et 'y' est la cible (**target**) que l'on souhaite prédire. C'est comparable à apprendre une langue avec un livre de traduction (exemples français-chinois). Les deux applications les plus courantes de l'apprentissage supervisé sont les problèmes de **Régression** et de **Classification**.

Les problèmes de Régression visent à prédire la valeur d'une variable continue (qui peut prendre une infinité de valeurs), comme le prix d'un appartement ou la consommation d'essence.

Les problèmes de Classification cherchent à classer un objet dans différentes classes (une variable discrète, avec un nombre fini de valeurs), par exemple identifier si un e-mail est un spam ou si une tumeur est maligne.

Quatre notions clés sont essentielles pour l'apprentissage supervisé :
• **Le Dataset :** Le tableau de données contenant les questions (x) et les réponses (y) que la machine doit apprendre.
• **Le Modèle et ses paramètres :** Une fonction mathématique (linéaire ou non-linéaire) créée à partir du Dataset, dont les coefficients sont les paramètres.
• **La Fonction Coût :** Mesure l'ensemble des erreurs entre les prédictions du modèle et le Dataset. L'objectif est d'avoir une petite Fonction Coût pour un bon modèle.
• **L'Algorithme d'apprentissage :** La méthode utilisée pour que la machine trouve les paramètres du modèle qui minimisent la Fonction Coût. L'algorithme de Gradient Descent est un exemple courant.

1.2 L'apprentissage non supervisé

Dans **l'apprentissage non supervisé** (Unsupervised Learning), la machine apprend à reconnaître des structures dans un Dataset qui ne contient pas de valeurs cibles (un Dataset non étiqueté, ou "unlabelled"). Cela s'apparente à apprendre le chinois seul en Chine, sans livre de traduction. Il est utilisé pour regrouper des données similaires (clustering), détecter des anomalies ou réduire la dimension des données.

1.3 L'apprentissage par renforcement

L'apprentissage par renforcement (Reinforcement Learning) est méthode inspirée de l'éducation des animaux de compagnie, où une "friandise" est donnée pour une bonne action. Elle est plus avancée mathématiquement et est notamment populaire en robotique.

1.4 Les principales bibliothèques

Pour la mise en œuvre pratique de ces fondations du Machine Learning, Python est l'outil de prédilection. L'installation d'Anaconda est recommandée, car elle regroupe tous les outils et bibliothèques nécessaires, comme Jupyter Notebook pour créer et partager du code Python. Les principales bibliothèques à maîtriser pour le Machine Learning sont :

• **Numpy :** Essentielle pour créer et manipuler des matrices avec efficacité, ce qui est fondamental pour le calcul matriciel en Machine Learning.

• **Matplotlib :** Permet de visualiser les Datasets, les fonctions et les résultats sous forme de graphes, courbes et nuages de points.

• **Sklearn (Scikit-learn) :** Cette bibliothèque contient l'ensemble des fonctions de l'état de l'art du Machine Learning, incluant les algorithmes les plus importants et diverses fonctions de pré-traitement des données. Grâce à elle, il n'est souvent pas nécessaire d'écrire les équations mathématiques complexes, car elles sont déjà implémentées.

• **Pandas :** Une excellente bibliothèque pour importer des tableaux de données (comme des fichiers Excel au format CSV) dans Python, réaliser des statistiques et charger le Dataset dans Sklearn.

En somme, cette partie pose les bases essentielles pour quiconque souhaite comprendre le fonctionnement conceptuel et les outils principaux du Machine Learning, en soulignant le rôle des données et des méthodes d'apprentissage pour résoudre des problèmes complexes

2 LA REGRESSION LINEAIRE

Cette partie constitue une étape cruciale pour mettre en pratique les concepts appris précédemment, en se concentrant sur les notions de **Dataset**, de **Modèle**, de **Fonction Coût** et de **Gradient Descent** à travers l'exemple concret de la régression linéaire.
La régression linéaire est une application de l'apprentissage supervisé, qui nécessite un Dataset contenant des exemples (x, y) pour que la machine puisse apprendre la relation entre x (les caractéristiques) et $\mathbf{y}$ (la valeur à prédire).

Le processus de développement d'un modèle de régression linéaire suit une recette précise :

2.1 Récolte de données (Dataset)

Tout projet de Machine Learning commence avec un Dataset, qui est un tableau de données contenant des exemples. Dans le cadre de l'apprentissage supervisé, ce dataset est "étiqueté", ce qui signifie qu'il contient des paires de "questions" et de "réponses" (ou exemples), notées (**x, y**)
• La variable $\boldsymbol{x}$ est appelée une feature (un facteur qui influence y). Un dataset typique comporte souvent plusieurs features ***(x1, x2, ...)***, qui sont regroupées dans une matrice X.
• La variale $\boldsymbol{y}$ est appelé target(cible). C'est la valeur a predire.
Ces exemples sont visualisés comme un nuage de points. Le Machine Learning permet d'analyser ces données bien au-delà de ce que 99.9% des gens font avec Excel.

2.2 Créer un modèle linéaire

À partir de ces données, on développe un modèle linéaire, une fonction mathématique simple comme $\boldsymbol{f(x) = ax + b}$. Les coefficients $\boldsymbol{a}$ et $\boldsymbol{b}$ sont les paramètres du modèle que la machine doit apprendre. Le but est de trouver les valeurs de a et b qui permettent de tracer une droite qui s'insère au mieux dans le nuage de points. Pour un modèle linéaire avec plusieurs features, cela se généralise à une forme matricielle $\mathbf{F(X) = X.\Theta}$, où $\mathbf{\Theta}$ est un vecteur contenant tous les paramètres.

2.3 Définir La Fonction Coût

Un bon modèle est celui qui minimise les **erreurs** entre ses prédictions $\boldsymbol{f(x)}$ et les vraies valeurs $\boldsymbol{y}$ du Dataset. Pour la régression linéaire, la fonction coût est définie comme la moyenne des erreurs quadratiques **(Mean Squared Error - MSE)**. Elle mesure l'ensemble de ces erreurs. La formule concrète de l'erreur pour un exemple i est
$\mathbf{erreur(i) = (f(x(i)) - y(i))^2}$. La Fonction Coût $\boldsymbol{J(a,b)}$ est alors :

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

où m est le nombre d'exemples. L'objectif central est de trouver les paramètres du modèle qui minimisent cette Fonction Coût, afin d'obtenir un modèle avec de **petites erreurs**.

2.4 L'Algorithme d'Apprentissage : Minimiser la Fonction Coût

Pour trouver le minimum de la fonction coût, on utilise un algorithme d'optimisation appelé **Gradient Descent** (la descente de gradient).

Analogie de la montagne : Imaginez-vous perdu en montagne et cherchant le refuge au point le plus bas. Vous descendez toujours dans la direction de la pente la plus forte, en avançant d'une certaine distance à chaque pas.

Application au Machine Learning : L'algorithme de Gradient Descent calcule la pente de la Fonction Coût (c'est-à-dire sa dérivée) et met à jour les paramètres (a et b) en se déplaçant d'une certaine distance dans la direction de la pente la plus forte (qui mène au minimum).

Le Learning Rate α est un hyper-paramètre crucial : s'il est trop lent, l'entraînement est long ; s'il est trop grand, le modèle peut ne jamais converger.

La mise à jour des paramètres se répète en **boucle (itérations)** jusqu'à atteindre le minimum de la fonction coût.

Formules de mise à jour des paramètres (Gradient Descent) : À chaque itération, les paramètres a et b sont mis à jour selon les formules suivantes, où α représente le Learning Rate :

$$\begin{aligned} \mathbf{a} &= \mathbf{a} - \alpha\left(\frac{\partial \mathbf{J}(\mathbf{a},\mathbf{b})}{\partial \mathbf{a}}\right) \\ \mathbf{b} &= \mathbf{b} - \alpha\left(\frac{\partial \mathbf{J}(\mathbf{a},\mathbf{b})}{\partial \mathbf{b}}\right) \end{aligned} \tag{2}$$

Calcul des dérivées partielles (Gradient) : Les dérivées partielles de la Fonction Coût par rapport à a et b (qui représentent la pente) sont nécessaires pour le Gradient Descent. Pour la régression linéaire, elles sont définies comme suit :

$$\begin{aligned} \frac{\partial \mathbf{J}(\mathbf{a},\mathbf{b})}{\partial \mathbf{a}} &= \frac{\mathbf{1}}{\mathbf{m}} \sum_{\mathbf{i=1}}^{\mathbf{m}} (\mathbf{ax} + \mathbf{b} - \mathbf{y})\mathbf{x} \\ \frac{\partial \mathbf{J}(\mathbf{a},\mathbf{b})}{\partial \mathbf{b}} &= \frac{\mathbf{1}}{\mathbf{m}} \sum_{\mathbf{i=1}}^{\mathbf{m}} (\mathbf{ax} + \mathbf{b} - \mathbf{y}) \end{aligned} \tag{3}$$

Utilisation des matrices (pour des calculs simplifiés) : Dans la pratique, pour simplifier les calculs, le Dataset et les paramètres sont souvent exprimés sous forme matricielle. Le vecteur des paramètres est noté $\mathbf{\Theta}$ (où $\mathbf{\Theta} = (\mathbf{a},\mathbf{b})^{\mathbf{T}}$. La formule du modèle devient $\mathbf{F}(\mathbf{\Theta}) = \mathbf{X}.\mathbf{\Theta}$. Les formules du Gradient et du gradient descent peuvent alors être écrites de manière plus compacte :

$$\text{Gradient : } \frac{\partial \boldsymbol{J}(\boldsymbol{\Theta})}{\partial \boldsymbol{\Theta}} = \frac{\mathbf{1}}{\boldsymbol{m}} \boldsymbol{X}^{\boldsymbol{T}} (\boldsymbol{F}(\boldsymbol{X}) - \boldsymbol{Y}) \tag{4}$$

$$\text{Gradient Descent : } \boldsymbol{\Theta} = \boldsymbol{\Theta} - \boldsymbol{\alpha} \left(\frac{\partial \boldsymbol{J}(\boldsymbol{\Theta})}{\partial \boldsymbol{\Theta}} \right) \tag{5}$$

Une fois cet algorithme programmé, vous pourrez observer votre première intelligence artificielle apprendre à prédire, par exemple, le prix d'un appartement. Cette approche est la base pour des algorithmes plus complexes, y compris ceux qui reconnaissent des visages ou prédisent le cours de la bourse.

Au-delà de la Linéarité : La Régression Polynomiale La beauté de la régression linéaire est sa capacité à être étendue à des problèmes non-linéaires. Si une relation entre les features et la target n'est pas une ligne droite (par exemple, une courbe parabolique), on peut utiliser la Régression Polynomiale. Cela implique d'ajouter des "features polynomiales" au dataset (par exemple, $\mathbf{x^2}, \mathbf{x^3}$, etc.). La machine peut alors développer un modèle plus complexe qui épouse l'allure des données, sans modifier l'algorithme d'entraînement sous-jacent, grâce au calcul matriciel et aux fonctions comme PolynomialFeatures de **Sklearn**. C'est là que le Machine Learning surpasse Excel dans sa capacité à traiter des milliers de variables.

3 REGRESSION LOGISTIQUE ET LES PROBLEMES DE CLASSIFICATION

Après avoir abordé la régression linéaire qui prédit des valeurs continues (comme le prix d'un appartement), il est temps d'explorer les problèmes de Classification. Ces problèmes consistent à classer un objet dans différentes classes, c'est-à-dire à prédire la valeur d'une variable discrète qui ne prend qu'un nombre fini de valeurs. Un exemple typique est de prédire si un email est un spam (y = 1) ou non (y = 0). La Régression Logistique est un modèle particulièrement adapté aux problèmes de classification binaire, où la variable cible (y) ne peut prendre que deux valeurs (par exemple, 0 ou 1). C'est une application de l'Apprentissage Supervisé. Voici les éléments clés pour comprendre la Régression Logistique et la Classification :

3.1 Le Modèle de Régression Logistique

Pour un problème de classification binaire, un modèle linéaire simple comme $\mathbf{F} = \mathbf{X}.\boldsymbol{\Theta}$ (utilisé en régression linéaire) n'est pas approprié car il peut produire des valeurs en dehors de l'intervalle. L'objectif est de prédire une probabilité d'appartenance à une classe. Pour cela, la Régression Logistique utilise une fonction spéciale appelée la Fonction Sigmoïde (ou Fonction Logistique), notée σ. Cette fonction a la particularité de toujours retourner une valeur comprise entre 0 et 1, ce qui en fait un excellent choix pour représenter des probabilités. **La Formule du Modèle de Régression Logistique :** Le modèle de Régression Logistique applique la fonction sigmoïde au produit matriciel des features et des paramètres ($\mathbf{X}.\Theta$) :

$$\boldsymbol{\sigma}(\mathbf{X}.\boldsymbol{\Theta}) = \frac{\mathbf{1}}{\mathbf{1} + \mathbf{e}^{-\mathbf{X}.\theta}}$$

Où :

— $\mathbf{X}$ est la matrice des features (les facteurs d'entrée).

— $\boldsymbol{\Theta}$ est le vecteur des paramètres du modèle (les coefficients que le modèle va apprendre).

— $\boldsymbol{e}$ est la base du logarithme naturel.

À partir de la valeur de $\sigma(\mathbf{X}.\theta)$, une frontière de décision est définie, généralement à un seuil de 0.5. Si la probabilité prédite est ≥ 0.5, l'objet est classé dans la classe 1 ; sinon, il est classé dans la classe 0.
Exemple : Pour classer un email en spam ou non, si $\sigma(\mathbf{X}.\theta)$ prédit 0.8, l'email est classé comme spam (y = 1) car $\mathbf{0.8} > \mathbf{0.5}$. Si elle prédit 0.2, il est classé comme non-spam (y = 0).

3.2 La Fonction Coût pour la Régression Logistique

La Fonction Coût mesure les erreurs du modèle. Contrairement à la régression linéaire qui utilise l'Erreur Quadratique Moyenne (MSE), l'utilisation de la MSE pour la régression logistique ne donnerait pas une courbe convexe. Une fonction coût non-convexe poserait problème pour l'algorithme de Gradient Descent, car il risquerait de se bloquer dans un minimum local sans trouver le minimum global. Pour cette raison, une nouvelle Fonction Coût est développée spécifiquement pour la régression logistique, souvent appelée **Cross-Entropy Loss** ou **Log Loss**. Cette fonction pénalise fortement le modèle lorsque ses prédictions sont loin des valeurs réelles.
La Formule de la Fonction Coût (Cross-Entropy Loss) : La Fonction Coût complète pour la Régression Logistique est exprimée ainsi :

$$\boldsymbol{J(\Theta) = -\frac{1}{m}\sum_{i=1}^{m}[y^{(i)}\log(\sigma(\mathrm{X}^{(i)}.\Theta)) + (1-y^{(i)})\log(1-\sigma(\mathrm{X}^{(i)}.\Theta))]}$$

— m est le nombre d'exemples dans le Dataset.

— $y^{(i)}$ est la valeur réelle (0 ou 1) pour l'exemple i.

— $\boldsymbol{\sigma(\mathrm{X}^{(i)}.\Theta)}$ est la prédiction du modèle pour l'exemple i.

Cette formule assure que la fonction coût est convexe, permettant au Gradient Descent de trouver le minimum global.

3.3 L'Algorithme de Minimisation : Gradient Descent

L'objectif central est de trouver les paramètres $\boldsymbol{\Theta}$ qui minimisent la Fonction Coût. L'algorithme d'apprentissage le plus courant pour y parvenir est le Gradient Descent. Le principe reste le même que pour la régression linéaire : on ajuste les paramètres itérativement en se déplaçant dans la direction de la pente la plus forte de la Fonction Coût (son gradient).
Le Gradient de la Fonction Coût pour la Régression Logistique : Étonnamment, bien que la Fonction Coût soit différente, la formule du gradient de la Fonction Coût par rapport aux paramètres $\boldsymbol{\Theta}$ est la même que pour la régression linéaire :

$$\boldsymbol{\frac{\partial \mathrm{J}(\Theta)}{\partial \Theta} = \frac{1}{m}\mathrm{X}^{T}.(\sigma(\mathrm{X}.\Theta) - \mathrm{y})}$$

La Règle de Mise à Jour des Paramètres (Gradient Descent) : À chaque itération, les paramètres sont mis à jour selon la formule :

$$\boldsymbol{\Theta = \Theta - \alpha \times \frac{\partial \mathrm{J}(\Theta)}{\partial \Theta}}$$

Où : $\boldsymbol{\alpha}$ est le **Learning Rate** (vitesse d'apprentissage), un hyper-paramètre qui contrôle la taille du pas à chaque itération. Un bon choix de α est crucial pour que le modèle converge. En pratique, des bibliothèques comme Sklearn simplifient énormément cette

implémentation. La fonction SGDClassifier de Sklearn, par exemple, gère le calcul de la Fonction Coût (loss='log'), des gradients et de l'algorithme de minimisation pour vous.

3.4 Un Autre Algorithme de Classification : K-Nearest Neighbour (K-NN)

En plus de la Régression Logistique, il existe d'autres algorithmes de classification. L'un des plus simples à comprendre est le K-Nearest Neighbour (K-NN), ou "le voisin le plus proche".

Principe de K-NN : Pour classer un nouvel exemple, l'algorithme trouve les K exemples les plus proches dans le Dataset d'entraînement, puis attribue à ce nouvel exemple la classe majoritaire parmi ces K voisins.
Exemple : Si vous avez un point inconnu et que ses 3 voisins les plus proches sont 2 points de la classe 'rouge' et 1 point de la classe 'bleue', le point inconnu sera classé 'rouge'.

Distance Métrique : La "proximité" est mesurée à l'aide d'une fonction de distance, la plus courante étant la distance Euclidienne (la ligne droite entre deux points). D'autres comme la distance de Manhattan ou cosinus peuvent être utilisées.

Le rôle de K : Le paramètre K (le nombre de voisins à considérer) est crucial, choisir un K trop petit peut rendre le modèle sensible au bruit et conduire à l'Overfitting (le modèle est trop spécialisé aux données d'entraînement). Un K plus grand permet au modèle de mieux généraliser. **Implémentation :** Sklearn propose la classe KNeighborsClassifier pour implémenter facilement cet algorithme.
Exemple concret : Le K-NN est utilisé pour la vision par ordinateur, comme la reconnaissance de chiffres écrits à la main. Le modèle analyse les features (par exemple, la valeur de chaque pixel d'une image) et trouve les chiffres les plus similaires dans son Dataset pour faire une prédiction. Un modèle K-NN peut atteindre une précision de 99% pour cette tâche.

4 Sklearn : Le cœur du Machine Learning en Python

Sklearn est une librairie Python essentielle qui contient "toutes les fonctions de l'état de l'art du Machine Learning". Elle simplifie grandement l'implémentation des algorithmes en encapsulant les calculs complexes de fonctions coût et de gradients.

4.1 SGDRegressor (Régression Linéaire)

SGDRegressor est une classe de Sklearn utilisée pour résoudre les problèmes de Régression Linéaire et Polynômiale. En régression, l'objectif est de prédire la valeur d'une variable continue, comme le prix d'un appartement.

•**Rôle et Fonctionnement :**

1. Le SGDRegressor implémente l'algorithme de Gradient Descent Stochastique (Stochastic Gradient Descent Regressor), qui est une variante du Gradient Descent. L'objectif principal du Gradient Descent est de trouver les paramètres (θ) du modèle qui minimisent la Fonction Coût.
2. Pour la régression linéaire, la Fonction Coût est l'Erreur Quadratique Moyenne (MSE), qui est la moyenne des erreurs au carré entre les prédictions du modèle et les valeurs réelles.
3. Le SGDRegressor gère automatiquement le calcul de cette Fonction Coût, de ses gradients (dérivées partielles de la Fonction Coût par rapport aux paramètres), et l'algorithme de minimisation lui-même. Ainsi, vous n'avez pas besoin d'écrire ces équations mathématiques complexes.
4. Équations implementees par SGDRegressor :

   Modèle Linéaire : $\boldsymbol{F(X) = X \cdot \theta}$

   Fonction Coût (MSE) : $\boldsymbol{J(\Theta) = \frac{1}{2m} \sum (F(X) - y)^2}$

   Gradient : $\boldsymbol{\frac{\partial J(\Theta)}{\partial \Theta} = \frac{1}{m} X^T (F(X) - Y)}$

   Mise à jour des paramètres (Gradient Descent) : $\boldsymbol{\Theta = \Theta - \alpha \times \frac{\partial J(\Theta)}{\partial \Theta}}$

•**Utilisation avec Sklearn :**

1. Définition du modèle : **model = SGDRegressor(**max_iter**=1000, eta0=0.001)**.
2. Entraînement du modèle : **model.fit(x, y)**. Cette étape est l'apprentissage où le modèle ajuste ses paramètres.
3. Évaluation du modèle : **model.score(x, y)**. Cette fonction calcule le coefficient de détermination (R^2), une métrique de performance.
4. Prédictions : **model.predict(x)**.

•**Exemple :** Prédire le prix d'un appartement selon sa surface habitable. Initialement, un modèle peut être mauvais si entraîné avec de mauvais hyper-paramètres (ex : max_iter=**100**, eta0=0.0001), mais en les ajustant (max_iter=**1000**, eta0=0.001), la performance peut s'améliorer significativement (ex : $R^2 = 94\%$).

4.2 KNeighborsClassifier (K-Nearest Neighbour - K-NN)

KNeighborsClassifier est utilisé pour les problèmes de Classification. La classification vise à classer un objet dans différentes classes, c'est-à-dire à prédire la valeur d'une variable discrète (qui ne prend qu'un nombre fini de valeurs), comme déterminer si un email est un spam ou non.

•Rôle et Fonctionnement (Algorithme K-NN) :

1. K-NN est "probablement l'algorithme le plus simple à comprendre de tous" pour la classification.
2. Principe : Pour classer un nouvel exemple, l'algorithme trouve les K exemples les plus proches dans le Dataset d'entraînement, puis il attribue à ce nouvel exemple la classe majoritaire parmi ces K voisins. **Exemple :** Si vous voulez savoir si votre ami a chaud ou froid à 15°C, et que ses réactions passées indiquent "froid" à 10°C et "chaud" à 30°C, K-NN classerait "froid" car 15°C est plus proche de 10°C.
3. Distance Métrique : La "proximité" est mesurée à l'aide d'une fonction de distance. La plus courante est la distance Euclidienne (la ligne droite entre deux points), mais d'autres comme la distance de Manhattan ou cosinus peuvent être utilisées. Sklearn implémente ces équations pour vous.
4. Le rôle de K : Le paramètre K (le nombre de voisins à considérer) est crucial. Un K trop petit peut rendre le modèle sensible au bruit et entraîner l'Overfitting (le modèle est trop spécialisé aux données d'entraînement et perd sa capacité à généraliser). Un K plus grand permet au modèle de mieux généraliser et est moins sensible aux "impuretés et cas particuliers".
5. K-NN peut gérer des problèmes de classification avec plusieurs classes.

•Utilisation avec Sklearn :

1. Définition du modèle : model = KNeighborsClassifier(). (Le paramètre $n\_neighbors$ peut être ajusté pour K).
2. Entraînement du modèle : **model.fit(X, y)**.
3. Évaluation du modèle : **model.score(X, y)**.
4. Prédictions : **model.predict(test)**.

•Exemple : La reconnaissance de chiffres écrits à la main. Un Dataset d'images de chiffres (0 à 9) est utilisé, où chaque image est représentée par 64 features (la valeur de chaque pixel). Un modèle K-NN entraîné sur ce Dataset peut atteindre une précision de 99% pour identifier le bon chiffre.

4.3 MLPClassifier (Multi-Layer Perceptron - Réseaux de Neurones)

MLPClassifier est la classe Sklearn pour implémenter des Réseaux de Neurones de base, spécifiquement un Multi-Layer Perceptron (MLP). Les Réseaux de Neurones sont des modèles "bien plus complexes" que les précédents, pouvant contenir des millions de paramètres. C'est le fondement du Deep Learning.

•Rôle et Fonctionnement :

1. Les Réseaux de Neurones sont structurés en couches : une couche d'entrée (input layer), une ou plusieurs couches cachées (hidden layers), et une couche de sortie (output layer).
2. Chaque "rond" dans le réseau est un neurone, qui représente une fonction d'activation. Pour un réseau de neurones basique, la Fonction Logistique (Sigmoïde) est souvent utilisée comme fonction d'activation.
3. Le rôle du neurone (Perceptron) : Il calcule d'abord une somme linéaire des entrées pondérées par des paramètres ($\mathbf{z} = \sum \mathbf{x}\theta$), puis applique la fonction d'activation à cette somme pour produire une sortie ($\sigma(\mathbf{z})$).
4. Deep Learning : Plus il y a de couches cachées, plus le réseau est considéré comme "profond" (deep), et plus le modèle est riche et capable de résoudre des problèmes complexes.
5. Processus d'apprentissage :
   — Forward Propagation : Les données d'entrée passent à travers le réseau, de la couche d'entrée à la couche de sortie, pour obtenir une prédiction.
   — Fonction Coût : Une Fonction Coût (similaire à la Cross-Entropy Loss de la régression logistique) mesure l'erreur de la prédiction.
   — Back Propagation : La "contribution de chaque neurone dans l'erreur finale" est calculée en remontant l'erreur de la sortie vers l'entrée. Ce calcul des gradients est mathématiquement complexe mais est géré par la librairie.
   — Gradient Descent : Les paramètres du modèle sont mis à jour itérativement pour minimiser la Fonction Coût, en utilisant les gradients calculés par la rétropropagation.
6. Défis et solutions : Les grands Datasets et les millions de paramètres nécessitent des techniques spécifiques pour l'entraînement : Mini-Batch Gradient Descent (fragmenter les données), Batch Normalization (mettre les variables à la même échelle), et Distributed Deep Learning (utiliser le Cloud).

•Utilisation avec Sklearn :

1. Définition du modèle : model = **MLPClassifier(**$hidden\_layer\_sizes = (10, 10, 10), max\_iter$ = 1000).$hidden\_layer\_sizes$ permet de spécifier le nombre de neurones par couche cachée (ici, 3 couches de 10 neurones).
2. Entraînement du modèle : model.fit(X, y).
3. Évaluation du modèle : model.score(X, y).

•Exemple : Identifier des espèces de fleurs Iris en se basant sur des features comme la longueur et la largeur des pétales et des sépales. Un MLPClassifier peut atteindre un score de 98% pour cette tâche.

En résumé, Sklearn est un outil puissant qui permet de mettre en œuvre des algorithmes sophistiqués de Machine Learning en quelques lignes de code. Que ce soit pour la régression avec SGDRegressor, la classification par similarité avec KNeighborsClassifier, ou la classification complexe avec les réseaux de neurones via MLPClassifier, le processus général reste le même : importer le Dataset, choisir le modèle approprié, l'entraîner (.fit()), et l'évaluer (.score())

5 Réseaux de Neurones

Les Réseaux de Neurones (ou Neural Networks en anglais) représentent une catégorie de modèles de Machine Learning beaucoup plus complexes que ceux que nous avons pu voir précédemment. Ils sont la pierre angulaire du Deep Learning, un domaine qui a révolutionné la reconnaissance vocale, la vision par ordinateur, l'analyse de sentiments, le traitement du langage naturel et même la création artistique.

5.1 Definition

Un Réseau de Neurones est une fonction mathématique complexe avec potentiellement des millions de coefficients (appelés paramètres). Contrairement à un modèle de régression linéaire qui n'a que deux coefficients (a et b), un réseau de neurones peut en avoir une quantité énorme, lui permettant de s'attaquer à des tâches bien plus avancées.

5.2 Les Défis du Deep Learning

La puissance des Réseaux de Neurones vient avec des exigences importantes :
• **Grands Datasets :** Ils nécessitent généralement des millions de données pour être entraînés efficacement.
• **Temps d'apprentissage :** L'entraînement peut prendre beaucoup de temps, parfois plusieurs jours.
• **Puissance de calcul :** Ils demandent une grande puissance de calcul. Pour surmonter ces défis, des techniques avancées comme le Mini-Batch Gradient Descent, le Batch Normalization et le Distributed Deep Learning ont été développées.
Pour surmonter ces obstacles, des techniques avancées ont été développées :

- **Mini-Batch Gradient Descent :** Cette méthode fragmente le Dataset en petits lots pour simplifier le calcul du gradient à chaque itération, ce qui accélère le processus d'apprentissage.
- **Batch Normalization :** Elle consiste à mettre toutes les variables d'entrée et de sortie internes du Réseau de Neurones à la même échelle, ce qui aide à éviter des calculs de gradients extrêmes et stabilise l'apprentissage.
- **Distributed Deep Learning :** Cette approche utilise le Cloud pour diviser la charge de travail et la répartir sur plusieurs machines, augmentant ainsi la puissance de calcul disponible

5.3 Anatomie d'un Réseau de Neurones

Un Réseau de Neurones est structuré en plusieurs couches (layers) :
• **Une couche d'entrées** (input layer) : où les données initiales (features $\boldsymbol{x}$) sont introduites.
• **Une ou plusieurs couches cachées** (hidden layers) : des couches intermédiaires entre l'entrée et la sortie.

• **Une couche de sorties** (output layer) : qui produit le résultat final du modèle. Chaque petit cercle dans ces couches est appelé un neurone. Un neurone est fondamentalement une fonction d'activation.
• **Le Perceptron** (le neurone de base) : Le réseau de neurones le plus simple est le perceptron, qui est identique à la Régression Logistique que nous avons étudiée. Un neurone effectue deux opérations principales :

1. Il calcule une somme linéaire des entrées multipliées par leurs poids (paramètres $\boldsymbol{\theta}$). C'est un calcul linéaire. Si $\boldsymbol{X}$ est la matrice des entrées et $\boldsymbol{\theta}$ le vecteur des paramètres, cette somme peut être représentée comme : $\boldsymbol{z} = \sum \boldsymbol{x\theta}$ ou de manière plus générale sous forme matricielle $\boldsymbol{X}.\boldsymbol{\theta}$.
2. Il passe cette somme ($\boldsymbol{z}$ ou $\boldsymbol{X}.\boldsymbol{\theta}$) à travers une fonction d'activation non-linéaire. La fonction logistique (ou sigmoïde) est un exemple courant pour les réseaux de neurones basiques. L'équation du modèle de Régression Logistique (qui est un perceptron) est :
$$\boldsymbol{\sigma}(\boldsymbol{X}.\boldsymbol{\theta}) = \frac{\mathbf{1}}{\mathbf{1} + \mathbf{e}^{-\boldsymbol{X}.\boldsymbol{\theta}}}$$
D'autres fonctions d'activation comme la tangente hyperbolique (tanh) ou la ReLU (Rectified Linear Unit) sont également utilisées pour accélérer l'apprentissage.

• **Réseaux à plusieurs neurones** (Deep Learning) : En connectant plusieurs de ces perceptrons en couches, on construit un Réseau de Neurones. Plus il y a de couches cachées, plus le réseau est considéré comme "**profond**", d'où le terme Deep Learning. Les sorties d'une couche de neurones deviennent les entrées de la couche suivante. Par exemple, pour un réseau simple à 3 neurones, la sortie finale peut être $\boldsymbol{a_3 = \sigma(\theta_1 a_1 + \theta_2 a_2)}$, où a_1 et a_2 sont les sorties de la couche précédente, et θ_1, θ_2 sont les paramètres associés aux connexions.

5.4 L'Analogie avec le Cerveau Humain

Historiquement, les Réseaux de Neurones ont été comparés au cerveau humain pour illustrer leur potentiel. L'idée était que, comme les neurones biologiques, les fonctions d'activation produisent un signal si les entrées dépassent un certain seuil. Cependant, il est important de noter que cette analogie est aujourd'hui considérée comme simpliste et que les Réseaux de Neurones ne fonctionnent pas réellement comme le cerveau humain ; ils sont avant tout d'énormes compositions de fonctions mathématiques. L'expression "Réseau de Neurones" a persisté pour des raisons de marketing et de facilité de compréhension.

5.5 L'entraînement d'un Réseau de Neurones

L'entraînement suit les quatre étapes fondamentales de l'apprentissage supervisé :

1. **Le Dataset** : Un ensemble de données étiquetées $(\boldsymbol{X}, \boldsymbol{y})$ est fourni au réseau.
2. **Le Modèle (Forward Propagation) :** Le calcul des prédictions à partir des entrées en traversant le réseau de la couche d'entrée vers la couche de sortie est appelé Forward Propagation. Il s'agit d'emboîter les fonctions d'activation de chaque couche.

3. **La Fonction Coût et son Gradient :** Une fonction coût mesure les erreurs entre les prédictions du modèle et les vraies valeurs du dataset. Pour les perceptrons (neurones de base) et donc la régression logistique, la Fonction Coût complète est :

$$J(\boldsymbol{\theta}) = -\frac{\mathbf{1}}{\boldsymbol{m}} \sum \boldsymbol{y} \times \mathbf{log}(\boldsymbol{\sigma}(\boldsymbol{X}.\boldsymbol{\theta})) + (\mathbf{1} - \boldsymbol{y}) \times \mathbf{log}(\mathbf{1} - \boldsymbol{\sigma}(\boldsymbol{X}.\boldsymbol{\theta}))$$

   Pour calculer la contribution de chaque neurone à l'erreur finale (le gradient de la Fonction Coût), une technique mathématiquement complexe appelée **Back Propagation** est utilisée. Pour la régression logistique (et donc un neurone), le gradient est :

$$\frac{\boldsymbol{\partial J(\theta)}}{\boldsymbol{\partial \theta}} = \frac{\mathbf{1}}{\boldsymbol{m}} \boldsymbol{X^T}.(\boldsymbol{\sigma}(\boldsymbol{X}.\boldsymbol{\theta}) - \boldsymbol{y})$$

4. **L'Algorithme de Minimisation :** Le Gradient Descent est l'algorithme principal utilisé pour ajuster les paramètres du modèle afin de minimiser la Fonction Coût. Son principe reste le même que celui vu précédemment. L'ajustement des paramètres $\boldsymbol{\theta}$ à chaque itération est donné par : $\boldsymbol{\theta} = \boldsymbol{\theta} - \boldsymbol{\alpha} \times \frac{\partial J(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}}$ où $\boldsymbol{\alpha}$ est le Learning Rate (vitesse d'apprentissage).

5.6 Programmation d'un Réseau de Neurones

Bien que les frameworks comme Tensorflow soient souvent utilisés pour les Réseaux de Neurones complexes, la bibliothèque Sklearn permet de programmer des réseaux de neurones basiques. Par exemple, pour identifier des espèces de fleurs d'Iris en utilisant 4 caractéristiques (longueur/largeur du pétale, longueur/largeur du sépale), on peut utiliser le MLPClassifier (Multi-Layer Perceptron Classifier) de Sklearn :
• **Importation :** from sklearn.$neural\_network$ import MLPClassifier
• **Création du modèle :** model = $MLPClassifier(hidden\_layer\_sizes$=(10, 10, 10), $max\_iter = 1000$) pour un réseau avec trois couches cachées, chacune ayant 10 neurones.
• **Entraînement :** model.fit(X, y).
• **Évaluation :** model.score(X, y) (un score de 984% peut être obtenu pour l'exemple des Iris, montrant la haute précision du modèle).

En résumé, les Réseaux de Neurones et le Deep Learning sont des outils puissants pour résoudre des problèmes complexes de régression et de classification, en s'appuyant sur les principes fondamentaux du Machine Learning mais avec une architecture et des capacités de modélisation accrues.

6 Gérer un projet de Machine Learning

Mener à bien un projet de Machine Learning (ML) ne se limite pas à connaître les algorithmes ; il est crucial de savoir comment les utiliser efficacement. Les sources soulignent que l'erreur majeure des novices est de négliger l'importance de la préparation des données et de la détection de problèmes comme l'Over fitting.

6.1 Préparation des Données (Data Pre-processing)

L'étape la plus importante et la plus chronophage (environ 80% du temps d'un Data Scientist) est la préparation des données. La performance d'un programme de Machine Learning dépend avant tout de la quantité et de la qualité des données du Dataset. Actions clés pour préparer votre Dataset :

- **Supprimer les anomalies et erreurs** : Il est fréquent qu'un Dataset contienne des données incorrectes qu'il faut supprimer pour ne pas biaiser l'apprentissage de la machine.
- **Normaliser les données** : Mettre les données sur une même échelle pour accélérer et améliorer l'efficacité de l'apprentissage.
- **Gérer les valeurs manquantes** : Assigner une valeur par défaut aux données manquantes.
- **Convertir les features catégorielles** : Transformer les données non numériques (ex : "homme/femme") en valeurs numériques (ex : homme=0, femme=1).
- **Nettoyer les features redondantes** : Supprimer les caractéristiques (features) fortement corrélées pour faciliter l'apprentissage.
- **Créer de nouvelles features (Feature Engineering)** : Dériver de nouvelles caractéristiques à partir des existantes pour enrichir le Dataset.

Exemple : Si vous avez la "longueur du jardin" (x_1) et la "largeur du jardin" (x_2), vous pouvez créer une nouvelle feature "surface du jardin" ($x_3 = x_1 \times x_2$). Outils comme sklearn et pandas sont essentiels pour ces étapes. Il est crucial que les données utilisées pour l'entraînement proviennent de la même distribution que celles sur lesquelles le modèle sera appliqué. Par exemple, entraîner un modèle sur des photos haute définition de poires et l'utiliser ensuite sur des photos floues ou déformées réduira sa précision. L'expertise du Data Scientist dans le domaine d'application est également primordiale pour interpréter correctement les données et éviter les biais.

6.2 L'Over fitting et l'Under fitting

Un modèle de Machine Learning doit être capable de bien généraliser, c'est-à-dire de faire de bonnes prédictions sur de nouvelles données qu'il n'a jamais vues.

• **Over fitting (Grande Variance) :** Ce phénomène se produit lorsque le modèle s'est trop spécialisé sur les données d'entraînement. Il a appris le "bruit" des données d'entrée plutôt que la relation sous-jacente.

Conséquence : Le modèle donne de très petites erreurs (voire zéro) sur le Dataset d'entraînement, mais de grandes erreurs sur de nouvelles données. Il est souvent causé

par un modèle trop complexe ou un Dataset insuffisant.
Exemple : Un modèle qui relie parfaitement tous les points d'un nuage de données, même les valeurs aberrantes.

• **Under fitting (Grand Biais) :** À l'inverse, l'Under fitting survient lorsque le modèle est trop simple et ne parvient pas à capturer la complexité des données.
Conséquence : Le modèle donne de grandes erreurs tant sur le Dataset d'entraînement que sur les nouvelles données.
L'objectif est de trouver le juste milieu entre un biais élevé et une variance élevée

6.3 La Régularisation

La régularisation est une méthode essentielle pour limiter la variance d'un modèle (et donc l'Over fitting) sans pour autant augmenter son biais de manière significative.

• **Pénalisation de la Fonction Coût :** Une technique courante consiste à ajouter un terme de pénalité aux paramètres du modèle dans la Fonction Coût. Pour la régression linéaire, cela donne la formule de la Régularisation Ridge (ou L2) :

$$\boldsymbol{J}(\boldsymbol{\theta}) = \frac{1}{2m} \sum (\boldsymbol{F}(\boldsymbol{X}) - \boldsymbol{Y})^2 + \boldsymbol{\lambda} \sum \boldsymbol{\theta}^2$$

où $\boldsymbol{\lambda}$ est le facteur de régularisation. Un $\boldsymbol{\lambda}$ trop grand peut mener à l'Under fitting, tandis qu'un $\boldsymbol{\lambda}$ trop faible peut conduire à l'Over fitting.
Exemple : Les sources montrent qu'un modèle de régression linéaire avec Ridge(alpha=0.1) (où alpha est $\boldsymbol{\lambda}$) peut avoir un coefficient R^2 légèrement plus faible sur le Dataset d'entraînement, mais offre une bien meilleure généralisation sur de nouvelles données comparé à un modèle sans régularisation qui souffre d'Over fitting.

• **Pour K-Nearest Neighbour (K-NN) :** Augmenter la valeur de K (le nombre de voisins) permet au modèle de moins tenir compte des anomalies et d'améliorer sa généralisation.

• **Pour les Réseaux de Neurones :** La technique de Dropout désactive aléatoirement certains neurones à chaque cycle d'entraînement, rendant le réseau moins sensible aux détails et améliorant la généralisation.

6.4 Le Train set et le Test set

Pour diagnostiquer correctement la performance d'un modèle et éviter l'Over fitting, il est impératif de le tester sur des données qu'il n'a pas utilisées pendant son entraînement.
• **Division du Dataset :** Le Dataset est aléatoirement divisé en deux parties :
- Train set (80%) : Utilisé pour entraîner le modèle et trouver les meilleurs paramètres.
- Test set (20%) : Utilisé pour évaluer la vraie performance du modèle sur des données nouvelles et non vues.

• **Outil :** La fonction *train_test_split* de sklearn est utilisée pour cette division. • **Diagnostic du modèle :** En analysant la Fonction Coût ($\boldsymbol{J}(\boldsymbol{\theta})$) sur les deux ensembles :

— Si les erreurs sont grandes sur le Train set et le Test set, le modèle a un grand biais (Under fitting).
Solutions : Créer un modèle plus complexe, ajouter plus de features, entraîner plus longtemps, diminuer le Learning Rate, collecter plus de features importantes.
— Si les erreurs sont faibles sur le Train set mais grandes sur le Test set, le modèle a une grande variance (Over fitting).
Solutions : Utiliser la régularisation, utiliser un modèle plus simple (moins de paramètres/features), collecter plus de données.

Le développement d'un modèle de Machine Learning est un cycle itératif : on part d'une idée, on code, on évalue, on diagnostique le biais ou la variance, puis on ajuste la stratégie pour corriger les problèmes rencontrés.

CONCLUSION

Ce rapport a présenté les fondements du Machine Learning, mettant en lumière la manière dont une machine peut apprendre à partir d'expériences sans nécessiter de programmation explicite. À travers l'exploration de concepts clés tels que l'apprentissage supervisé illustré par la Régression Linéaire et la Régression Logistique pour la classification et l'apprentissage non-supervisé via le K-Mean Clustering, nous avons acquis une compréhension essentielle des mécanismes qui sous-tendent ces technologies.

Au cœur de chaque modèle, le rôle crucial du Dataset, du Modèle, de la Fonction Coût et de l'Algorithme d'apprentissage, comme le Gradient Descent, a été mis en évidence. L'utilisation d'outils tels que Python et la bibliothèque Sklearn facilite considérablement l'implémentation de ces concepts, rendant accessible la création de modèles complexes.

Cependant, la réussite d'un projet de Machine Learning repose principalement sur la qualité et la préparation des données. Le processus de nettoyage, normalisation, et enrichissement du Dataset est une étape qui demande un investissement de temps significatif pour un Data Scientist.

De plus, nous avons appris à surmonter des défis majeurs tels que l'Overfitting et l'Underfitting. Les techniques de Régularisation, incluant l'ajout de termes de pénalité et l'ajustement du paramètre K pour K-NN, sont essentielles pour améliorer la capacité de généralisation des modèles sur de nouvelles données. La division du Dataset en Train set et Test set est impérative pour évaluer la performance réelle des modèles et diagnostiquer efficacement ces problèmes.

En résumé, le développement d'un modèle de Machine Learning est un cycle itératif qui nécessite une compréhension approfondie des données ainsi qu'une capacité à diagnostiquer et corriger les erreurs. Fort de ces connaissances, nous sommes désormais prêt à aborder des problèmes concrets et à transformer des données en solutions intelligentes.