09- Introduction à l'intelligence artificielle et à l'apprentissage machine

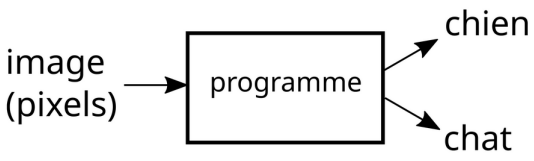
1 Introduction

L'intelligence artificielle (IA) vise à créer des systèmes informatiques qui peuvent réaliser des tâches qui nécessitent normalement l'intelligence humaine, comme la reconnaissance de la parole, la reconnaissance d'images, l'établissement d'un diagnostic, la prise de décision, la traduction automatique...

Il est aussi possible de générer des textes, des images, de la musique, etc.
On parle alors d'**IA générative**, comme *ChatGPT* (Open AI), ou *Gemini* (Google), ou le français *Le Chat* (Mistral AI).

Qu'est-ce que l'apprentissage machine (machine learning en anglais) ?

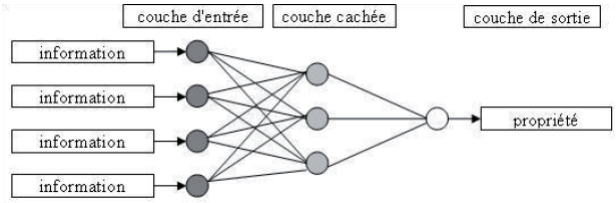
C'est une branche de l'IA. Par exemple on souhaite faire apprendre à la machine à reconnaître des images de chat et de chien. On va alors lui proposer une base d'apprentissage (des images de chat et de chien), et on implémentera une méthode qui permettra ensuite à la machine, sur une image nouvelle, à reconnaître s'il s'agit d'un chat ou d'un chien.



La technique de l'apprentissage machine (ou apprentissage automatique) , est connue depuis les années 1950, mais elle a eu un essor considérable ces dernières années grâce à la quantité de données disponibles pour réaliser l'apprentissage, et grâce à la puissance de calculs des ordinateurs.

Les méthodes d'apprentissage sont très nombreuses ; conformément au programme, nous détaillerons ***l'algorithme des k plus proches voisins***, et ***l'algorithme des k moyennes***.

Mais citons également la méthode d'apprentissage consistant à utiliser des **réseaux de neurones** (ainsi nommés par analogie avec le cerveau humain), à deux ou d'avantages de couches : ce sont en fait des programmes qui, à l'aide de paramètres passés en entrée, fournissent des réponses au problème posé . Lors de la phase d'apprentissage, ces programmes s'ajusteront, pour être optimaux lorsqu'ils seront utilisés sur de nouvelles données.



Un réseau de neurones à 3 couches

2 Un exemple d'apprentissage supervisé : l'algorithme des k plus proches voisins (ou kNN pour *k Nearest Neighbors*)

2.1 Présentation

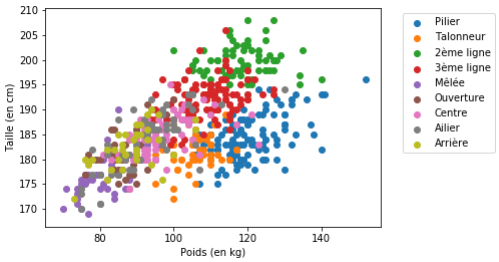
L'algorithme kNN est une méthode d'apprentissage supervisé utilisée pour la classification.

Cet apprentissage est dit **supervisé** car lors de la phase d'apprentissage, les données proposées à la machine pour son entrainement auront déjà été classifiées (par un humain). Par exemple, on proposera à la machine un millier d'images de chat ou de chien, toutes étiquetées "chat" ou "chien". Il va alors falloir créer un programme qui fera le lien entre les caractéristiques de ces différentes photos, et le fait qu'il s'agisse d'un chien ou d'un chat.

Nous allons travailler sur l'exemple simple suivant : nous disposons de **la liste des 595 joueurs de Top14** (le championnat de France de rugby) de la saison 2019/2020 , qui sera notre base d'apprentissage, avec notamment pour chaque joueur **son poste** (Ailier, Talonneur, 2ème ligne, etc.) et **deux caractéristiques** : sa taille et son poids.

Cet exemple est très simple puisqu'ici nous allons classer les individus en utilisant deux caractéristiques seulement (taille et poids), ce qui nous permet de faire un dessin en deux dimensions. Si le nombre de caractéristiques retenues est p, il faudra donc travailler en dimension p...

Chaque joueur pourra être vu comme un point dans $\mathbb{R}^2$.
La figure ci-contre montre la répartition de ces points par poste pour la base d'apprentissage.



On remarque que les différentes classes ont naturellement tendance à créer des amas dans le plan des observations. L'algorithme kNN va justement exploiter cette propriété.

2.2 Principe de fonctionnement

L'algorithme kNN est *un algorithme de prédiction* : il s'agit d'associer à une nouvelle donnée (la taille et le poids d'un nouvel individu) une classe d'appartenance (on **prédit** la classe d'un individu).

Cet algorithme fonctionne en comparant la distance entre la nouvelle donnée en entrée et les données d'entraînement. On fixe au départ un entier naturel k non nul, puis pour la nouvelle donnée d'entrée, l'algorithme identifie les k données d'entraînement les plus proches (au sens de la distance euclidienne).
Ces k voisins sont appelés les k plus proches voisins (kNN).
La classe estimée de la nouvelle donnée d'entrée sera alors la classe majoritaire parmi ces k voisins.

2.3 Choix de la valeur de k

Le choix de la valeur de k peut affecter les performances de l'algorithme. Si k est trop petit, l'algorithme peut être sensible aux points aberrants (valeurs extrêmes), ce qui peut entraîner une mauvaise classification. Si k est trop grand, cela peut entraîner une baisse de la précision de l'algorithme, car les données voisines peuvent alors ne pas être représentatives de la nouvelle donnée d'entrée.

Diverses considérations théoriques et expérimentales mènent à choisir, si n est le nombre d'individus de la base d'apprentissage et C le nombre de classes, $\boldsymbol{k \approx \sqrt{\frac{n}{C}}}$.

2.4 Principales étapes de l'algorithme

Concrètement, soit $(X_i, c_i)_{1\leq i\leq n}$ est une base d'apprentissage où :

- n est le nombre d'individus de la base d'apprentissage
- pour tout i entre 1 et n, X_i est un vecteur de $\mathbb{R}^p$ correspondant aux p caractéristiques de l'individu $n°i$
- pour tout i entre 1 et n, c_i désigne la classe associée de l'individu $n°i$

Dans l'exemple des postes au rugby, $n = 595$ et $p = 2$.

Pour trouver la classe d'une nouvelle donnée X, l'algorithme se déroule en 3 étapes :

1. Calculer la distance $d_i = \|X - X_i\|$ de X à chaque observation X_i $(1 \leq i \leq n)$
2. Trier par ordre croissant ces distances et retenir l'ensemble I des k premiers indices.
3. Déterminer la classe majoritaire parmi les $(c_i)_{i\in I}$.

2.5 Comment vérifier l'efficacité de la prévision

Un algorithme d'apprentissage ne produit pas en général des résultats parfaits, et il est donc important de pouvoir l'évaluer. Pour cela, on utilise généralement *une base de test*, distincte de la base d'apprentissage, sur laquelle les vraies classes sont également connues.

On peut alors évaluer les performances de l'algorithme sur cette base de test en calculant *la matrice de confusion associée*.

Définition 1 - *Matrice de confusion*
La matrice de confusion d'un résultat de classification sur une base de test est une matrice carrée d'ordre C (le nombre de classes, en supposant ces classes répertoriées de 0 à $C-1$) dont le coefficient (i, j) est le nombre d'observations pour laquelle la classe vraie est i et la classe estimée est j.

Lorsque la matrice de confusion est diagonale, cela signifie que toutes les observations de la base de test ont été correctement classifiées. En pratique, cette situation se produit rarement, et on évalue l'efficacité de l'algorithme en examinant la dispersion des valeurs en dehors de la diagonale.

Exemple (épreuve CCINP PSI 2019) : On dispose d'une base de patients, classés en 3 catégories selon l'état de leur colonne vertébrale (classe 0 : "normal", classe 1 : "hernie discale", classe 2 : "spondylolisthésis"), chaque patient étant caractérisé par une série de 6 mesures médicales (angle d'orientation du bassin, angle d'incidence du bassin, etc.).

On programme la méthode kNN à partir de cette base d'apprentissage, et on l'applique alors, avec $k = 8$, sur une base de test constitué de 100 patients dont l'état est connu.

On obtient alors la matrice de confusion : $\begin{pmatrix} 23 & 4 & 7 \\ 7 & 11 & 1 \\ 5 & 2 & 40 \end{pmatrix}$.

Sur la première ligne de la matrice, on constate qu'il y a eu 23 éléments du groupe 0 classés groupe 0 (donc bien classés), 4 éléments du groupe 0 classés groupe 1 , et 7 éléments du groupe 0 classés groupe 2. Au total sur les 34 éléments de groupe 0, seuls 23 ont été correctement reconnus.

Plus globalement, on peut dire que sur la base de test, le taux de réussite de la méthode est $\dfrac{23+11+40}{100}$, soit 74%.

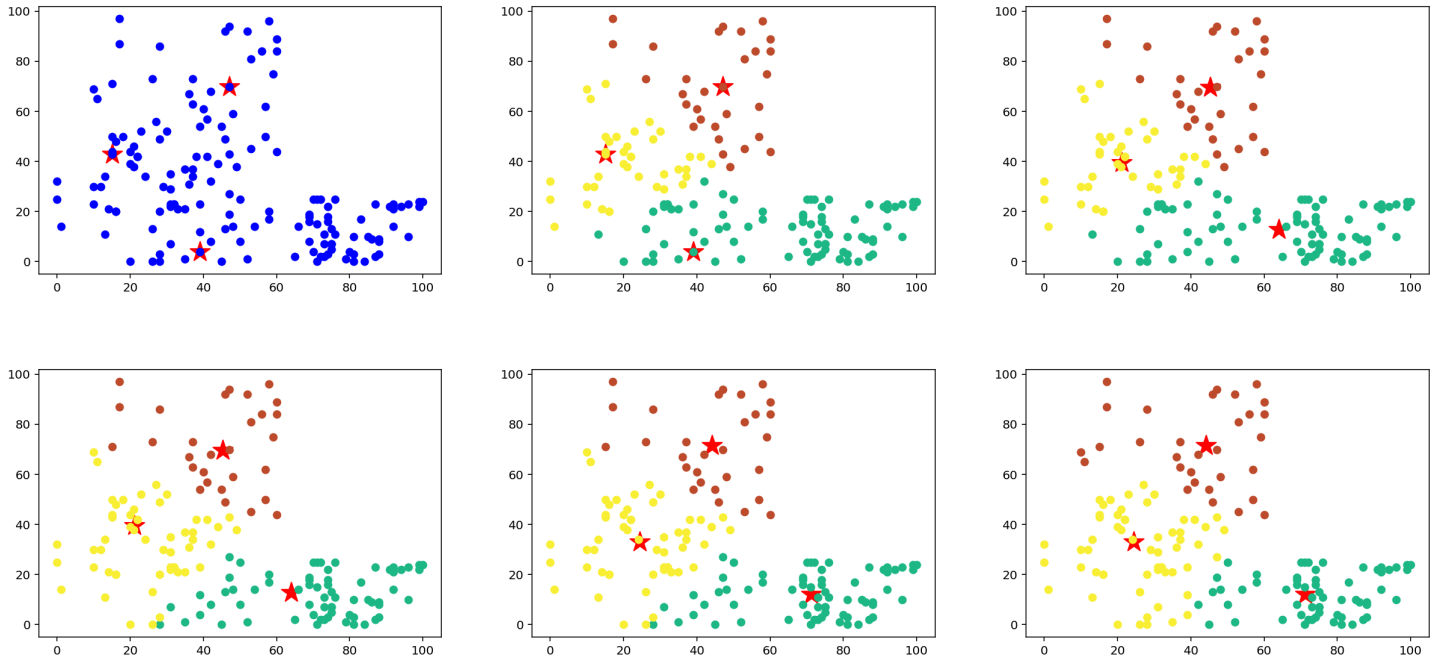
3 Un exemple d'apprentissage non supervisé : l'algorithme des k-moyennes

Grand concept du machine learning, il permet à un ordinateur d'apprendre lui-même sur les données que l'humain lui fournit. Concrètement, les algorithmes non supervisés repèrent des similarités dans les données pour pouvoir ensuite les structurer. Par exemple ils peuvent étudier les similarités entre individus, ce qui rend possible leur division en différents groupes. Ce partitionnement des individus est appelé *clustering*.

La méthode la plus classique est la méthode des k-moyennes (*k-means* en anglais). Elle ne nécessite qu'un seul choix de départ : k, le nombre de classes voulues. Les étapes sont alors les suivantes :

1. On initialise l'algorithme avec k points au hasard parmi les n individus. Ces k points représentent alors les k centres de classes.
2. On répète alors la suite des deux opérations suivants :
   (a) Pour chaque individu, on trouve de quel centre de classe il est le plus proche. Chaque individu fera alors partie d'une classe, et l'ensemble des individus sera partitionné en k classes.
   (b) On calcule alors l'isobarycentre des points de chacune des classes (on a donc k barycentres, aussi appelées k moyennes). Ces points deviennent nos nouveaux centres de classe., et on recommence à partir de l'étape 2, en déterminant, pour chaque point, quel est le centre de classe qui lui est le plus proche. Certains points vont alors changer de classe.
3. On réitère la suite des deux opérations de l'étape 2., jusqu'à ce que nos centres de classe ne bougent plus (en pratique, pour ne pas que l'algorithme tourne trop longtemps, on s'arrête quand toutes les distances entre les anciens centres et les nouveaux centres soient inférieures à un certain ε donné)

Illustration avec $k = 3$ (on souhaite faire trois classes d'individus) :



Exemples d'applications de l'algorithme des k-moyennes *(paragraphe réalisé par Le chat de Mistral)*

L'algorithme des k-moyennes (k-means) est largement utilisé dans divers domaines pour le regroupement de données en clusters. Voici une liste d'exemples où cet algorithme est couramment appliqué :

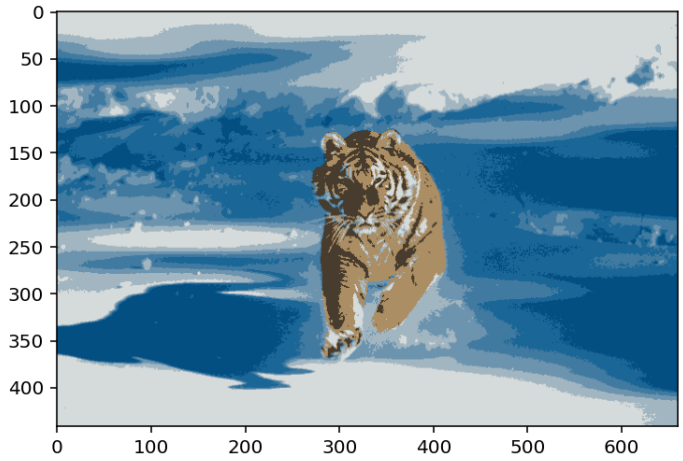
- **Segmentation de marché** : Les entreprises utilisent le k-means pour segmenter leurs clients en groupes distincts en fonction de leurs comportements d'achat, permettant ainsi des stratégies de marketing ciblées.
- **Analyse de données biologiques** : En bioinformatique, le k-means est appliqué pour regrouper des gènes ou des protéines ayant des expressions similaires, aidant à identifier des motifs biologiques significatifs.
- **Analyse de données de santé** : En médecine, le k-means est utilisé pour regrouper les patients en fonction de leurs symptômes ou de leurs réponses aux traitements, facilitant ainsi les diagnostics et les plans de traitement personnalisés.
- **Analyse de texte** : En traitement du langage naturel, le k-means peut être utilisé pour regrouper des documents ou des articles en fonction de leur contenu, aidant à la classification et à l'organisation des informations.
- **Analyse d'images** : En traitement d'images, le k-means est utilisé pour la compression d'images et la segmentation d'images en régions homogènes, facilitant ainsi l'analyse et la reconnaissance d'objets.

Illustration de la compression d'image avec le k-means :

On part d'une image en couleurs RGB, donc chaque pixel est codé sur 3 octets (256^3 possibilités de couleur). On décide alors que seulement k couleurs seront utilisés. Il faut donc partitionner les pixels en k-groupes, et affecter à chacun de ces groupes la couleur de leur centre de classe.

Si on décide par exemple d'avoir 8 couleurs, seulement 3 bits seront nécessaires pour stocker la couleur de chaque pixel, d'où une image 8 fois plus petite à stocker que l'originale.

Voici un exemple avant et après compression (seulement 8 couleurs sont utilisées sur l'image de droite) :



4 Exemple de programmation de l'algorithme KNN : A quelle poste joueriez-vous au rugby ?

Ici, la base d'apprentissage est sous forme d'un fichier texte, nommé *"JoueursTop14.txt"* de 595 lignes, où chaque ligne est composée de 6 champs, séparés par des points virgules :

- le premier champ est le club du joueur
- le deuxième champ est constitué du prénom et du nom du joueur
- le troisième champ est la date de naissance du joueur
- le quatrième champ est le poste du joueur
- le cinquième champ est le poids en kg du joueur
- le sixième champ est la taille en cm du joueur

1. Mise en forme des données de la base d'apprentissage
   (a) Ecrire une suite d'instructions créant la variable globale **liste_joueurs**, qui est une liste de dictionnaires contenant les données 'Poste', 'Poids' et 'Taille' du fichier. Cette liste devra donc comporter 595 dictionnaires, et le début de cette liste pourra être :

   $$\texttt{liste\_joueurs=[\{'Poste':'Pilier','Poids':83,'Taille':122\},\{'Poste': } \cdots \texttt{\}} \cdots \texttt{]}$$

   (b) Ecrire une fonction **postes** sans paramètres, qui renvoie un dictionnaire dont les clés sont les postes possibles et les valeurs 0 (plus tard cela correspondra au nombre de postes proches de notre donnée testée). Vérifier que votre dictionnaire est bien de longueur 9 (les 9 postes possibles au rugby)

   L'appel de cette fonction avec la liste des joueurs pourra retourner :

   $$\texttt{\{'Pilier':0,'Ailier':0, } \cdots \texttt{\}}$$

2. Introduction d'un nouveau point
   (a) Ecrire une fonction **liste_joueurs_distances** qui a pour paramètres un poids et une taille, et qui renvoie une liste de 295 dictionnaires ayant deux clés : le poste d'un joueur et la distance de ce joueur par rapport aux 'poids' et 'taille' saisis.

   L'appel de cette fonction avec la liste des joueurs et votre poids et votre taille pourra retourner :

   $$\texttt{[\{'Poste': 'Pilier', 'distance': 48.093658625644196\},\{'Poste': } \cdots \texttt{\}} \cdots \texttt{]}$$

   (liste de 595 dictionnaires)

   (b) On souhaite à présent trier la liste retournée par la fonction précédente, selon les distances croissantes contenues dans chacun des dictionnaires. Pour cela on va utiliser la fonction préprogrammée **sorted** de Python.
      i. Tester dans le shell la fonction **sorted** sur une liste de nombres de votre choix.

   Ici on ne va pas avoir directement une liste de nombres, il va donc falloir préciser en fonction de quelle donnée on veut faire le tri. Ceci est possible en précisant un argument **key** dans la fonction **sorted** de la manière suivante :

   ```
   1 | liste_distances_triee =sorted( liste_distances ,key=lambda d:d['distance'])
   ```

      ii. Ecrire une fonction **KNN**, de paramètres k le nombre de voisins, un poids, une taille, et qui renvoie le dictionnaire des postes mis à jour de la façon suivante : parmi les k joueurs dont la distance avec la position testée est la plus petite, on connaîtra la répartition entre les différents postes

   (c) Ecrire une fonction **prediction** , de paramètres k le nombre de voisins, un poids, une taille, et qui renvoie une prédiction sur le 'Poste' le plus adapté au poids et la taille de l'individu sur lequel on fait la prédiction.