Intelligence Artificielle

Cours

v1.1 Ⓟ

Institution Sainte Marie – 2 Rue de l'Abbaye - 92160 Antony

Table des matières

1 Introduction à l'Intelligence Artificielle (IA) 1

2 Un algo non supervisé : k-means 9

3 Un algo supervisé : k plus proches voisins, application à la classification 14

4 Un enjeu important, la régression 20

5 Modèle du perceptron 30

6 Réseau de Neurones 33

7 Le sur-apprentissage 38

1 Introduction à l'Intelligence Artificielle (IA)

Objectif

Cette section a pour objectifs :

- Découvrir les objectifs et domaines d'application de l'IA.
- Introduire les concepts d'apprentissage supervisé et non supervisé.
- Comprendre le fonctionnement de l'algorithme des k-moyennes, k plus proches voisins et leur rôle en partitionnement,classification ou régression.

1.1 Vocabulaire de différentes approches en intelligence artificielle

L'intelligence artificielle (IA) est un domaine vaste qui englobe un ensemble de techniques automatiques ou semi-automatiques pour résoudre des problèmes (en opposition avec les techniques manuelles). Dans le langage courant, il est fréquent de confondre l'IA avec le machine learning (ML). Or, le machine learning est un sous-domaine de l'IA qui regroupe des algorithmes capables « d'apprendre » à partir de données.

CC BY NC SA

D'autres approches existent, notamment les systèmes experts, basés sur des moteurs de règles et une base de connaissances prédéfinie. Ces systèmes, bien que puissants dans certains cas, manquent de flexibilité car ils ne peuvent pas apprendre de nouvelles régles par eux-mêmes. Par exemple, un assistant virtuel (Siri) simpliste pourrait être conçu à l'aide de ce type de système, en suivant des règles conditionnelles déterminées à l'avance.

Remarque

Un notion qui ne sera pas abordé dans ce chapitre mais dont il pertinent de parler : les grands modèles de langage (LLM) (ChatGPT, LeChat etc.). Ces outils ont révolutionné le domaine de l'intelligence artificielle en permettant des interactions naturelles et contextuellement pertinentes avec les utilisateurs. Depuis leurs débuts, ces modèles ont évolué de manière significative. À leur début, ils étaient souvent critiqués pour des erreurs simples, comme l'incapacité à effectuer des calculs élémentaires ou à comprendre des nuances contextuelles. Par exemple, ChatGPT, dans ses premières versions, pouvait échouer à résoudre des opérations mathématiques basiques comme 2+2, ce qui a suscité de légers doutes quant à leur fiabilité.

Ils sont dorénavant incontournables, dans l'industrie, dans les services et dans nos applications (par exemple : Duolingo, utilise un LLM pour générer des leçons). Dans le cadre de l'étude de l'intelligence artificielle, il est essentiel de ne pas négliger l'importance des outils tels que ChatGPT, Le Chat, DeepSeek, etc. Ces outils, souvent perçus comme de simples encyclopédies interactives, possèdent en réalité des capacités bien plus étendues. Il est vrai qu'ils excellent dans la fourniture d'informations factuelles et la génération de textes cohérents, mais leur potentiel va au-delà. Par exemple, Le Chat intègre un module de calcul formel, lui permettant d'effectuer des opérations mathématiques complexes. Cette fonctionnalité illustre comment ces outils d'IA s'appuient sur des techniques plus "classiques" pour enrichir leurs capacités.

Ainsi, bien qu'ils ne "raisonnent" pas de manière humaine, ils combinent des algorithmes avancés et des bases de données étendues pour offrir leurs réponses. C'est pourquoi on se permet de nuancer leur rôle : ce ne sont pas seulement des répertoires d'informations, mais des assistants capables d'exécuter des tâches variées grâce à une intégration intelligente de diverses technologies.

1.2 Classification des algorithmes de Machine Learning

Dans le programme de CPGE, nous nous intéressons à plusieurs algorithmes de machine learning :

- **Méthode des k plus proches voisins** (ITC et SII)
- **Méthode des k-moyennes** (ITC)
- **Réseaux de neurones** (SII)

Cependant, ces trois algorithmes ne sont qu'un échantillon des nombreuses techniques existantes. On peut classer les algorithmes de machine learning en trois grandes catégories :

1.2.1 Apprentissage supervisé

L'apprentissage supervisé repose sur l'utilisation d'un ensemble de données d'entrainement étiquettées (**c'est-à-dire pour lesquelles la correspondance (entrée, sortie attendue) est connue**) contenant des exemples associés à des réponses correctes (labels). L'objectif est **d'extraire les critères significatifs** permettant de résoudre le problème pour n'importe quelle entrée.

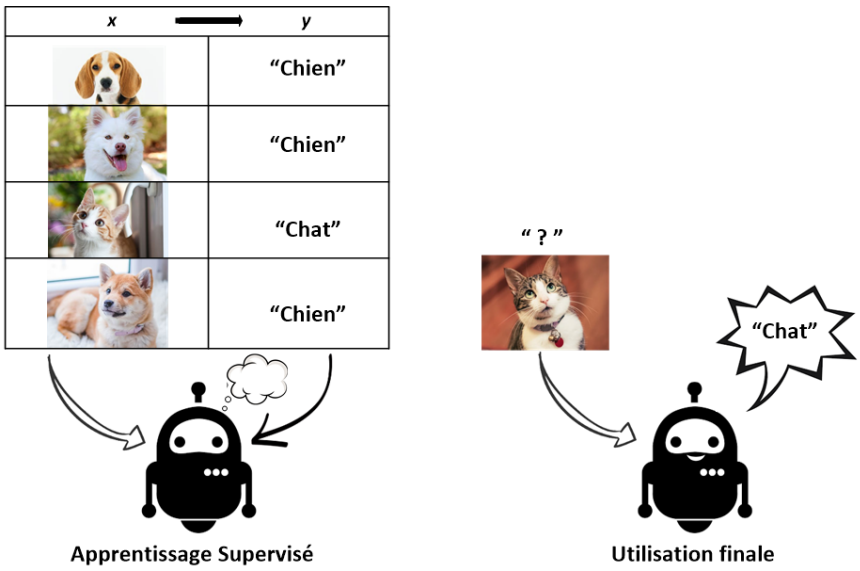


FIGURE 1 – Principe de l'apprentissage supervisé - Tiré de *https ://machinelearnia.com*

Deux types de problèmes sont principalement traités en apprentissage supervisé :

- **Régression** : Les algorithmes de régression sont utilisés pour prédire une valeur continue. Par exemple, prédire le prix d'une maison en fonction de sa taille, de son emplacement, etc. Les modèles de régression sont souvent basés sur des fonctions mathématiques continues et régulières, telles que des modèles linéaires, quadratiques, ou logarithmiques. L'objectif est de trouver une fonction qui mappe les variables d'entrée à une valeur continue de sortie.
- **Classification** : Les algorithmes de classification sont utilisés pour prédire une catégorie ou une étiquette discrète. Par exemple, déterminer si un email est un spam ou non. Les modèles de classification cherchent à diviser l'espace des entrées en régions distinctes, chacune correspondant à une classe. Même si les modèles de classification peuvent utiliser des fonctions continues pour approximer ces divisions, le résultat final est une étiquette discrète.

Il est vrai que les modèles de régression et de classification ont des objectifs différents et utilisent des approches différentes. Cependant, il existe des cas où les deux types de modèles peuvent être utilisés de manière complémentaire. Par exemple, un modèle de régression peut être utilisé pour prédire une probabilité continue, qui peut ensuite être convertie en une décision de classification en appliquant un seuil.

Remarque

Bien que les algorithmes de régression et de classification aient des objectifs distincts et utilisent des approches différentes, ils peuvent parfois se recouper dans des applications spécifiques où une prédiction continue peut être transformée en une décision discrète.

Remarque

Ici les mots continu et discret représentent une notion abstraite. Il vaudrait mieux utiliser infini vs fini (en effet sur un ordinateur rien n'est continu de par la numérisation). Et pour la classification,

l'ensemble des catégories doit être fini (dénombrable ne suffit pas).

Exemple:

Considérons un ensemble de données contenant des informations sur les étudiants d'une université donnée. Un algorithme de régression peut être utilisé pour prédire le nombre d'heures qu'un étudiant passe à étudier chaque semaine en fonction de son emploi du temps, de ses notes précédentes, et de ses activités extrascolaires. Nous utilisons la régression dans ce cas parce que le nombre d'heures d'étude est une quantité continue, avec un nombre infini de valeurs possibles.

En revanche, un algorithme de classification peut être utilisé pour prédire si un étudiant réussira ou échouera un examen en fonction des mêmes variables (emploi du temps, notes précédentes, activités extrascolaires). Dans ce cas, la classification est appropriée car le résultat est une catégorie discrète : réussite ou échec. L'algorithme analyse les données pour déterminer la probabilité que l'étudiant appartienne à l'une ou l'autre de ces catégories.

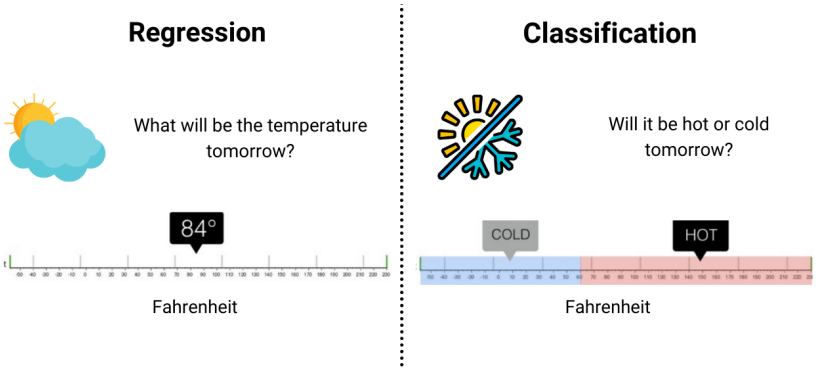


FIGURE 2 – Deux grands types de problèmes : classification ou régression. Tiré de *AI Part 3 - Regression vs Classification Models - ARNAB MUKHERJEE*

1.2.2 Apprentissage non supervisé

Contrairement à l'apprentissage supervisé, l'apprentissage non supervisé n'utilise pas de labels prédéfinis. Il vise plutôt à regrouper les données en sous-groupes semblables. Les critères de ressemblance sont très souvent implicites et non-explicables a priori (le critère ne sera simplement pas l'âge ou la taille d'une personne, mais une combinaison implicite de ces caractéristiques).

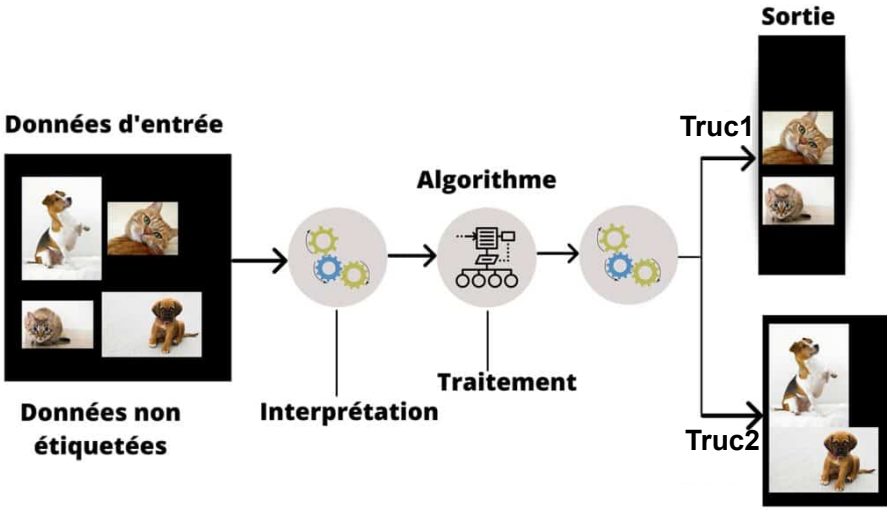


FIGURE 3 – Principe de l'apprentissage non supervisé - Tiré de *https ://www.data-transitionnumerique.com/*

Les principales méthodes de cette catégorie sont :

- **Clustering** : regroupe des données similaires sans information préalable sur les catégories (ex. segmentation de clients en marketing).
- **Réduction de dimension (Hors-programme)** : La réduction de dimension simplifie l'analyse des données en conservant les caractéristiques les plus pertinentes. Une méthode couramment utilisée est l'Analyse en Composantes Principales (ACP), qui permet de transformer un ensemble de variables corrélées en un ensemble plus petit de variables non corrélées, appelées composantes principales. Ces composantes capturent la variance maximale des données originales, facilitant ainsi la visualisation et l'interprétation des données complexes.
  L'ACP est souvent utilisée comme une étape préliminaire dans les processus d'apprentissage non supervisé, tels que le clustering. En réduisant la dimensionnalité des données, l'ACP peut aider à identifier des structures sous-jacentes et à améliorer la performance des algorithmes de clustering en éliminant le bruit et les redondances. Par exemple, dans la compression d'images, l'ACP peut être utilisée pour réduire le nombre de dimensions tout en conservant les informations essentielles, permettant ainsi une représentation plus compacte de l'image.
  Bien que l'ACP ne soit pas un algorithme d'apprentissage supervisé, elle joue un rôle important dans l'analyse exploratoire des données et la préparation des données pour des tâches d'apprentissage automatique. Elle permet de mieux comprendre la structure des données et de préparer le terrain pour des analyses plus approfondies.

Compléments

Un algorithme de clustering (de partitionnement/regroupement en bon français) a pour objectif de déterminer des groupes (on parle de clusters ou de grappes) d'observations ayant des similitudes. La notion de similarité dépend fortement d'un cas à l'autre, et généralement l'utilisateur n'a lui-même pas d'idée précise des critères selon lesquels les observations doivent être regroupées. Le clustering va donc proposer une partition des observations, en faisant en sorte que les observations d'un même groupe soient similaires, et que deux observations appartenant à deux groupes différents soient dissemblables.

Contrairement aux techniques de classification, les techniques de clustering appartiennent aux techniques d'apprentissage non supervisé : l'algorithme ne dispose pas d'exemples pour lesquels les groupes

à trouver sont connus.

Les techniques de clustering peuvent être utilisées pour découvrir des sous-groupes dans des populations, ou encore pour rattacher une nouvelle observation à un groupe d'observations aux propriétés bien connues.

Exemple:

Supposons que nous disposions d'une vaste collection d'images contenant trois fruits (i) des fraises, (ii) des poires et (iii) des pommes. **MAIS** vous ne savez par quels fruits vous avez. Les données ne sont pas étiquetées.

Dans l'ensemble de données, toutes les images sont mélangées et votre cas d'utilisation consiste à regrouper les fruits similaires, c'est-à-dire à créer N groupes avec N un paramètre que vous avez choisi. C'est exactement ce que fait un algorithme de regroupement.

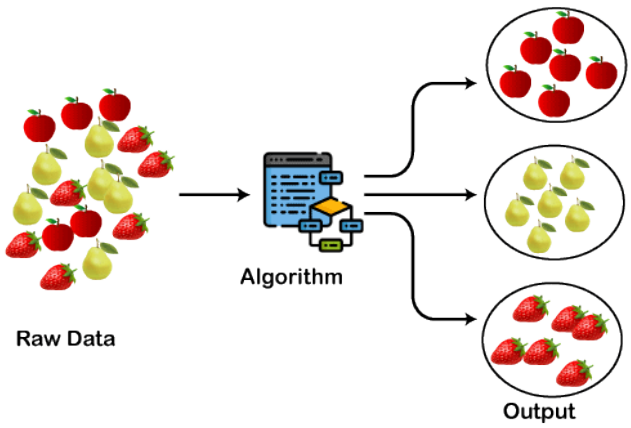


FIGURE 4 – Ici l'algorithme regroupe les données en 3 groupes distincts, on comprend vite que du choix du nombre de groupes de sortie vont dépendre les performances de l'algo. Tiré de *https ://www.datacamp.com/fr/blog/clustering-in-machine-learning*

1.2.3 Apprentissage par renforcement

Dans l'apprentissage par renforcement, un agent interagit avec un environnement et apprend par essai-erreur en maximisant une récompense. La définition de la CNIL est la suivante : « l'apprentissage par renforcement est un procédé d'apprentissage automatique consistant, pour un système autonome, à apprendre les actions à réaliser, à partir d'expériences, de façon à optimiser une récompense quantitative au cours du temps ».

Le système est plongé au sein d'un environnement, et prend ses décisions en fonction de son état courant. En retour, l'environnement procure une récompense, qui peut être positive ou négative.

Au fil des expériences, le système cherche un comportement décisionnel optimal, en ce sens qu'il maximise la somme des récompenses au cours du temps.

Exemples d'applications :

- Jeux vidéo et intelligence artificielle (ex. AlphaGo, qui a battu des champions humains au jeu de go).
- Publicité ciblée :
  Le système affiche une annonce à un utilisateur (action).
  - ◇ Si l'utilisateur clique sur l'annonce, le système reçoit une récompense positive, indiquant que l'annonce était pertinente pour cet utilisateur.
  - ◇ Si l'utilisateur ne clique pas, le système reçoit une récompense négative ou nulle, indiquant que l'annonce n'était pas pertinente.
  - ◇ Au fil du temps, le système ajuste sa stratégie pour maximiser la récompense cumulative, c'est-à-dire pour afficher des annonces qui ont une plus grande probabilité d'être cliquées par les utilisateurs.

  En utilisant l'apprentissage par renforcement, le système peut apprendre à personnaliser les annonces en fonction des préférences et des comportements des utilisateurs, améliorant ainsi l'efficacité des campagnes publicitaires.

Ce type d'apprentissage est souvent couplé à des réseaux de neurones pour améliorer l'efficacité de l'agent.

1.3 Conclusion

Le machine learning constitue aujourd'hui une part importante des techniques d'intelligence artificielle, avec comme principales approches : l'apprentissage supervisé et l'apprentissage non supervisé avec ou non présence d'apprentissage par renforcement. Ces différentes techniques permettent d'adresser une variété de problèmes, de la classification d'images à l'optimisation de décisions en environnement complexe. Voici un schéma synoptique des différents algorithmes de machine learning existants.

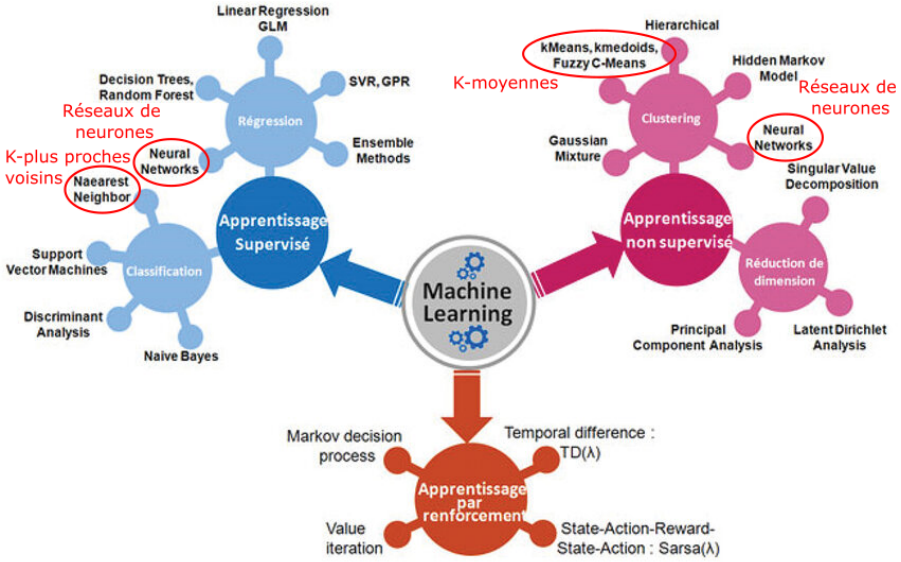


FIGURE 5 – Les différents algorithmes d'IA. Tiré de *Artificial Intelligence Methods : Toward a New Decision Making Tool- Ismail Lotfi Abdelhamid El Bouhadi*

En rouge, vous trouverez les algorithmes que nous allons voir dans ce chapitre.

2 Un algo non supervisé : k-means

2.1 Principe

L'algorithme des k-means est une méthode de classification non supervisée permettant de regrouper des données en k clusters. Il peut être vu comme un problème d'optimisation où l'objectif est de minimiser la somme des distances entre chaque point et le centroïde de son cluster, ce qui revient à minimiser le poids total de l'ensemble des clusters.

L'algorithme fonctionne de la manière suivante :

1. Initialisation : Choisir initialement k centres (centroïdes) de manière aléatoire. Ce choix initial est crucial car l'algorithme est randomisé et peut converger vers des minima locaux différents à chaque exécution, même avec le même jeu de données.
2. Affectation : Affecter chaque point du jeu de données au groupe dont le centroïde est le plus proche. La notion de "distance" est fondamentale ici, et bien que la distance euclidienne soit couramment utilisée, d'autres mesures de similarité peuvent être plus appropriées selon le contexte.
3. Mise à jour : Recalculer les centroïdes de chaque groupe en prenant la moyenne des points qui lui sont affectés.
4. Itération : Répéter les étapes d'affectation et de mise à jour jusqu'à convergence, c'est-à-dire jusqu'à ce que les centroïdes ne changent plus ou que les changements soient minimes.

Il est important de noter que le choix de la distance peut influencer fortement les résultats du clustering. L'algorithme peut être généralisé en utilisant d'autres mesures de similarité qui ne sont pas nécessairement des distances au sens mathématique strict. Cependant, l'utilisation de telles mesures peut entraîner une perte de certaines garanties de convergence théorique, bien que cela puisse être acceptable en pratique si les résultats obtenus sont pertinents pour l'application considérée.

De plus, il existe des variantes de l'algorithme des k-means où l'objectif pourrait être de maximiser la distance entre les clusters plutôt que de minimiser la distance intra-cluster. Ces variantes peuvent être utiles dans des contextes où l'on souhaite obtenir des clusters bien séparés les uns des autres.

2.1.1 Distance euclidienne

La distance euclidienne entre deux points $A(x_1, y_1)$ et $B(x_2, y_2)$ est donnée par :

$$D(A,B) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \tag{1}$$

Évidemment, cela se généralise s'il y a plus de deux caractéristiques.

2.1.2 Illustration

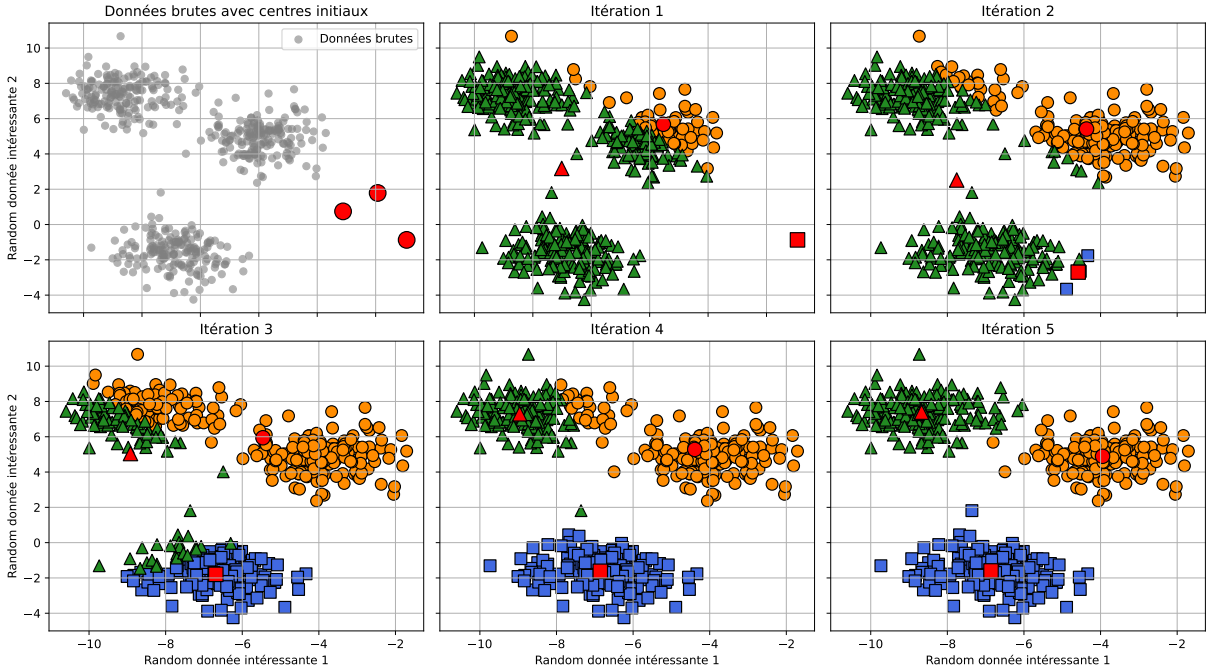


FIGURE 6 – Illustration de l'algorithme des k-means : itération par itération.

2.2 Impact des paramètres d'entrée

Le bon fonctionnement de k-means dépend de plusieurs paramètres :

- **Nombre de clusters** k : un k trop faible peut sous-estimer la structure des données, un k trop grand peut créer des clusters artificiels.
- **Initialisation des centroïdes** : une mauvaise initialisation peut affecter la convergence et mener à des résultats sous-optimaux. Une approche courante est **k-means++**, qui choisit stratégiquement les centroïdes initiaux.
- **Critère d'arrêt** : si la tolérance sur le déplacement des centroïdes est trop faible, l'algorithme peut tourner inutilement longtemps.
- **Type de distance utilisée** : la distance euclidienne est la plus courante, mais d'autres métriques (Manhattan, cosinus) peuvent être plus adaptées selon le type de données.

Remarque

En termes de complexité, l'algorithme des k-means a une complexité temporelle de $\mathcal{O}(n \cdot k \cdot i \cdot d))$, où :

- n est le nombre de points dans le jeu de données,
- k est le nombre de clusters,
- i est le nombre d'itérations nécessaires pour atteindre la convergence,
- d est la dimension des points.

2.2.1 Illustration

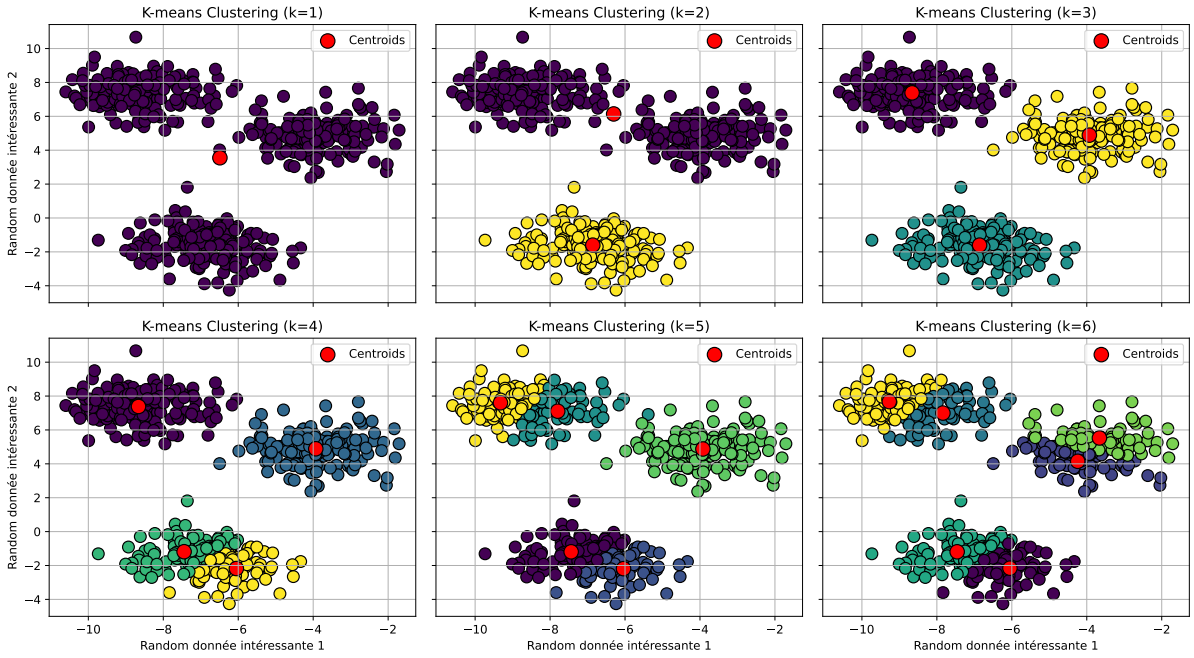


FIGURE 7 – Illustration de l'algorithme des k-means : résultats pour différentes valeurs de k.

2.3 Inertie et Méthode du coude

L'inertie d'un modèle de clustering k-means mesure la compacité des clusters en évaluant la somme des distances quadratiques entre chaque point et le centroïde de son cluster. Elle est définie par :

$$I_k = \sum_{i=1}^{k} \sum_{x \in C_i} \|x - \mu_i\|^2$$

où :

- k est le nombre de clusters,
- C_i est l'ensemble des points appartenant au cluster i,
- μ_i est le centroïde du cluster i,
- $\|x - \mu_i\|^2$ représente la distance euclidienne au carré entre un point x et son centroïde.

L'inertie diminue avec l'augmentation du nombre de clusters, mais elle ne doit pas être trop élevée pour éviter un sur-apprentissage. La « méthode du coude » consiste à représenter l'inertie en fonction de k et à identifier le point où la décroissance ralentit significativement. Ce point optimal correspond au nombre de clusters qui permet de capturer la structure des données sans sur-ajustement/ sur-apprentissage.

1. On exécute l'algorithme k-means pour différentes valeurs de k.
2. On calcule l'inertie intra-cluster, c'est-à-dire la somme des distances au carré entre chaque point et son centroïde.
3. On trace cette inertie en fonction de k et on recherche le **point d'inflexion** où la diminution devient moins significative : ce point est le **coude**, indiquant le choix optimal de k.

2.3.1 Illustration

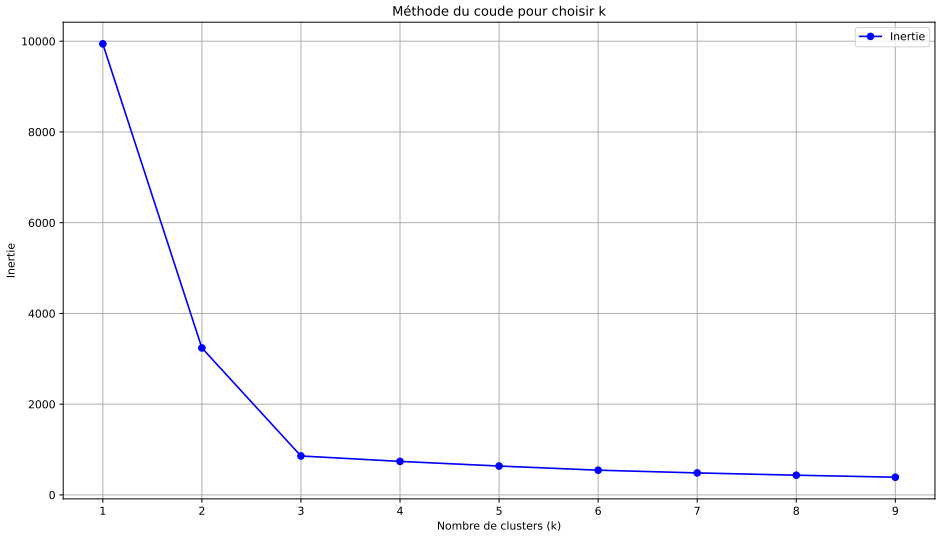


FIGURE 8 – Illustration de l'algorithme des k-means : méthode du coude. Choix du nombres de clusters k.

2.4 Exemples d'utilisation de k-means

L'algorithme des k-means est utilisé dans de nombreux domaines :

- **Segmentation d'image** : regrouper des pixels ayant des couleurs similaires pour simplifier une image.
- **Analyse de clients** : regrouper les clients en segments basés sur leurs habitudes d'achat.
- **Détection d'anomalies** : identifier des comportements inhabituels dans des ensembles de données (ex : détection de fraudes).
- **Regroupement de documents** : classer des documents selon leur contenu pour améliorer la recherche d'informations.

2.5 Et en pratique ? Implémentation des k-means

Objectif

Implémenter l'algorithme des k-means sur un jeu de données simple.

2.5.1 Étapes

1. **Initialisation** : k est notre paramètre d'entrée, on doit générer k centres aléatoires.
2. **Itérations** :
   - Calculer les distances entre les points et les centres.
   - Attribuer chaque point à un cluster.
   - Mettre à jour les centres.
3. **Convergence** : afficher les clusters finaux.

Ok sur le principe cela a l'air trivial. Une subtilité subsiste.

2.5.2 Importance de la normalisation des données

L'algorithme k-means étant basé sur la distance euclidienne, il est sensible à l'échelle des variables. Si les données comportent des caractéristiques ayant des ordres de grandeur différents, les dimensions à forte amplitude domineront le calcul des distances.

Exemple:

Considérons un ensemble de données contenant ces variables :

- La hauteur d'un individu en centimètres (ex : entre 150 et 200 cm).
- Le poids de l'individu en kilogrammes (ex : entre 50 et 120 kg).
- Les pas faits en une journée (ex : entre 1000 et 15000 pas).

Et là on note que si deux variables ne sont pas dans la même échelle, l'une pourrait avoir plus de poids dans le calcul de la distance euclidienne que l'autre. Dans ce cas, la variable « pas » aura une influence plus importante sur le calcul des distances, ce qui pourrait fausser les regroupements/agrégations/clustering k-means.

Solution : On applique une normalisation des données, par exemple via la standardisation :

$$x' = \frac{x - \mu}{\sigma} \tag{2}$$

Où μ est la moyenne et σ l'écart-type de la variable. Cette transformation permet de centrer et réduire les données pour qu'elles aient une moyenne nulle et une variance unitaire, évitant ainsi qu'une variable domine les calculs.

À retenir

L'algorithme des k-moyennes (k-*means* permet de regrouper les objets (ou sujets, ou sites, ou point, etc.) en k clusters distincts. Ce nombre k doit être spécifié, mais il existe des approches pour déterminer son nombre optimal.

Elle repose sur la **minimisation de la somme des distances euclidiennes au carré entre chaque objet (ou sujet, ou point) et le centroïde** (le point central) de son cluster.

La méthode des k-means est une approche de clustering non-hiérarchique (à l'intérieur d'un cluster les objets ne sont pas ordonnés en fonction de leurs ressemblance, cette notion n'est pas au programme).

La méthode des k-means appartient aux algorithmes **non supervisés** de machine learning. De plus, ici les objets (ou sujets, ou sites, ou points, etc.) appartiennent à un seul cluster, et les clusters ne se chevauchent pas.

3 Un algo supervisé : k plus proches voisins, application à la classification

Objectif

- Comprendre le principe de l'algorithme des k plus proches voisins (k-NN).
- Appliquer k-NN à un problème de classification.
- Étudier l'influence du paramètre k sur les performances de l'algorithme.
- Introduire la notion de matrice de confusion pour évaluer la qualité d'un modèle de classification.

3.1 Principe

L'algorithme des k plus proches voisins (k-NN, pour k-*nearest neighbors*) est un algorithme d'apprentissage supervisé qui peut être utilisé pour la classification ou la régression. Il repose sur un principe simple : pour classer un nouvel élément, on examine ses k plus proches voisins dans l'ensemble d'apprentissage et on lui attribue la classe majoritaire parmi ces voisins.

3.1.1 Illustration de la méthode des k plus proches voisins

Prenons un exemple où nous devons classer un gentil petit reptile. Nous devons déterminer s'il s'agit d'un crocodile ou d'un alligator. Chaque espèce est caractérisé par deux attributs : sa longueur (`Taille`) et la taille de sa gueule (`Gueule`). On dispose d'un ensemble d'exemples connus.

Le nouvel animal, avec des caractéristiques $(0.3, 3.5)$, doit être classé. L'algorithme k-NN consiste à observer les k voisins les plus proches de ce point et à attribuer la classe majoritaire parmi eux. Cette situation est représentée figure 9, où l'on a tracé un des paramètres en fonction de l'autre. Le point vert, le nouvel entrant doit être classé.

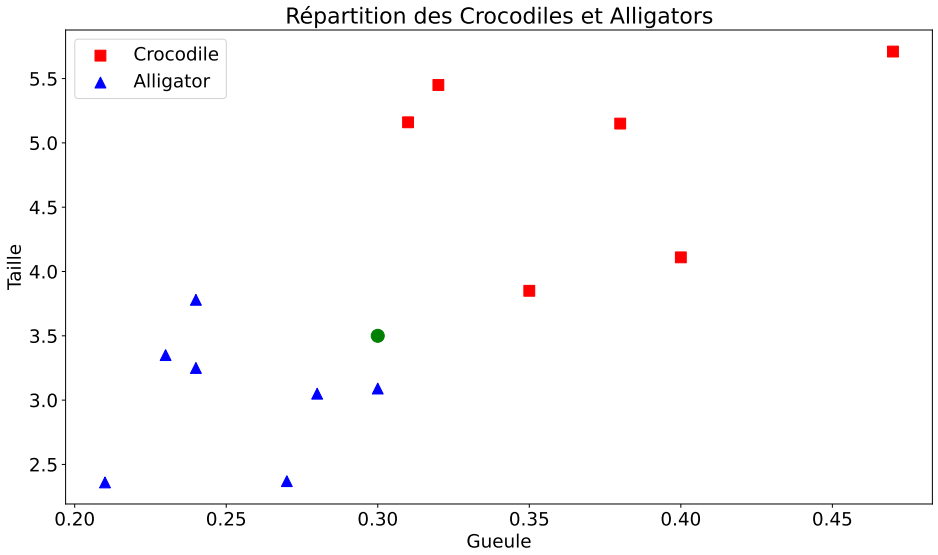


FIGURE 9 – Ici on trace une caractéristique en fonction de l'autre. Les groupes semblent assez bien séparés. Attention, cela n'est pas toujours le cas et d'autant plus si l'on prend une multitude de caractéristiques.

Il apparait évident que du choix de k dépend le résultat. Ici avec $k = 3$ l'animal test est un alligator.

3.1.2 Fonctionnement

1. Calculer la distance entre le point à classer et tous les points de l'ensemble d'entraînement (généralement avec la distance euclidienne).
2. Identifier les k plus proches voisins.
3. Attribuer la classe majoritaire parmi parmi les k plus proches voisins précédemment identifiés.

3.2 Impact des paramètres d'entrée

Le choix du paramètre k est crucial pour les performances de l'algorithme :

- Un k trop petit peut rendre l'algorithme sensible au bruit (un voisin erroné peut fausser la classification).
- Un k trop grand risque de lisser trop la classification et de réduire la capacité à identifier des structures fines dans les données.

Une méthode courante pour choisir k consiste à tester plusieurs valeurs et à mesurer les performances de l'algorithme sur un ensemble de validation.

3.3 Exemple d'application : préparation des données

Prenons ici une base de donnée connue : celle de détection de cancer du sein. On dispose ici d'une base de données contenant différentes observations de tumeurs pour lesquelles nous disposons, entre autres, des caractéristiques suivantes :

- Texture
- Périmètre
- Surface
- Et évidemment le diagnostic : bénigne ou maligne (le résultat).

Un exemple est donné dans le tableau ci-dessous :

id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean
842302	M	17.99	10.38	122.8	1001	0.1184
842517	M	20.57	17.77	132.9	1326	0.08474
84300903	M	19.69	21.25	130	1203	0.1096
84348301	M	11.42	20.38	77.58	386.1	0.1425
84358402	M	20.29	14.34	135.1	1297	0.1003
843786	M	12.45	15.7	82.57	477.1	0.1278
844359	M	18.25	19.98	119.6	1040	0.09463
84458202	M	13.71	20.83	90.2	577.9	0.1189
844981	M	13	21.82	87.5	519.8	0.1273
84501001	M	12.46	24.04	83.97	475.9	0.1186
845636	M	16.02	23.24	102.7	797.8	0.08206
84610002	M	15.78	17.89	103.6	781	0.0971

On constate à nouveau le même problème que sur l'algo des k-Means, il faut prétraiter les données et notamment les normaliser pour éviter une sur influence d'une donnée comparativement à une autre.

Nous allons donc normaliser ces données, une autre façon de faire qui marche bien lorsque tous les nombres sont positifs est la normalisation MinMax.

$$x' = \frac{x - \min(x)}{\max(x) - \min(x)}$$

Ainsi toutes les colonnes ne varient plus qu'entre 0 et 1.

3.3.1 Échantillonnages des données

Cette base de données est composée de 679 cas d'observations de tumeurs pour lesquelles nous avons 32 attributs distincts. Pour réaliser le code des k plus proches voisins, on utilise les fonctions de la bibliothèque `sklearn`.

```
1 from sklearn.model_selection import train_test_split
2
3 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=42)
```

La fonction `train_test_split` permet de séparer les données en deux groupes. Pour tester la qualité du modèle des k plus proches voisins, 80% des données sont utilisées comme données d'entraînement (test_size=0.20). Cela signifie tout simplement que ce sont les données `X_train` dont les

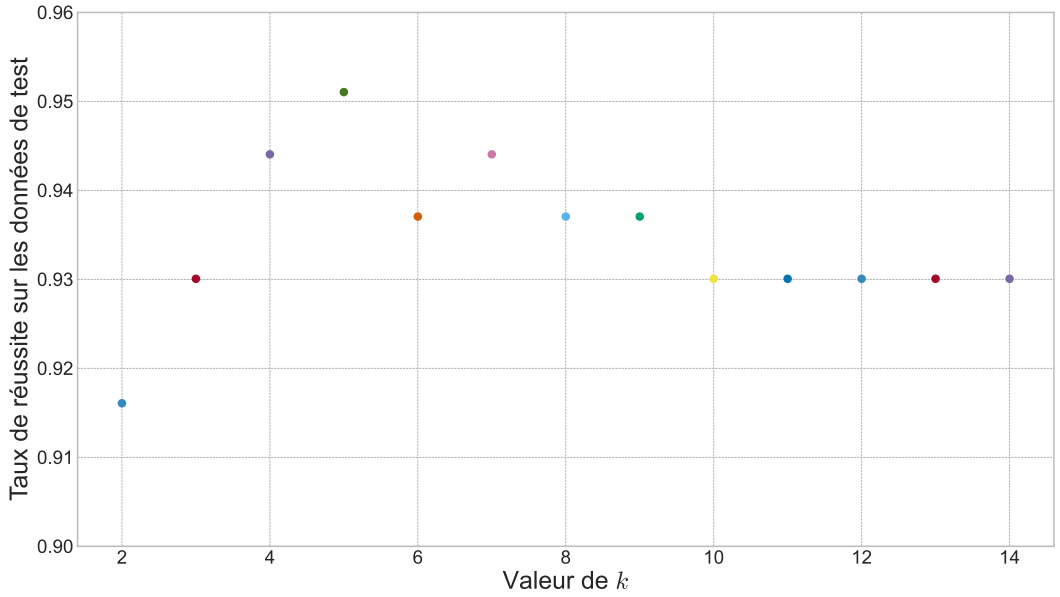
étiquettes `y_train` sont connues pour la réalisation de notre algorithme des k plus proches voisins. Les 20% des données restantes seront les données de tests `X_test`, c'est-à-dire celles dont on va chercher à déterminer les étiquettes. Or, on connait les "vraies" étiquettes de ces données `y_test` et on pourra comparer celles-ci avec les prédictions établies par l'algorithme `y_pred` et ainsi tester la qualité du modèle.

```
1 from sklearn.neighbors import KNeighborsClassifier
2 knn = KNeighborsClassifier(n_neighbors=3)
3 knn.fit(X_train, y_train)
4 y_pred = knn.predict(X_test)
5 knn.score(X_test, y_test)
```

- À la ligne 2, on crée le modèle que l'on souhaite utiliser. Ici, on va utiliser l'algorithme pour $k = 3$. Cette ligne permet d'initialiser l'algorithme.
- À la ligne 3, l'algorithme ici n'effectue aucune optimisation mais va juste sauvegarder toutes les données en mémoire. C'est sa manière d'apprendre en quelque sorte.
- À la ligne 4, on effectue les prédictions (phase d'inférence) sur les données de test.
- À la ligne 5, on évalue le score de notre modèle, on compare les étiquettes données par l'algorithme (`y_pred`) aux données de test, à celles que l'on connaissait concernant ces données (`y_test`). Cela donne un score, par exemple : $0,96$ équivalent à $96\,\%$ de réussite.

3.3.2 Recherche de la meilleure valeur de k

On peut alors tracer ce score en fonction de différentes valeur de k pour déterminer la meilleure valeur possible.



Ici, le maximum est obtenu pour $k = 5$.

3.3.3 Utilisation de la matrice de confusion

Grâce à la fonction `score`, on peut connaître la qualité de prédiction du classifieur. Cependant, il pourrait être très intéressant de savoir **où** les erreurs sont produites et **comment**. Pour cela, on va utiliser la **matrice de confusion**.

Dans le cadre de l'évaluation des performances d'une fonction de prédiction sur un problème de ***classification multiclasse*** à N étiquettes, on souhaite disposer d'une mesure de performance permettant d'inspecter en détail le comportement de la fonction de prédiction et notamment ses difficultés à prédire chacune des étiquettes. La matrice de confusion est un tableau de dimension $N \times N$ qui répertorie toutes les informations concernant les prédictions effectuées et les vraies étiquettes.

Exemple:

Imaginez une reconnaissance automatique de chiffres : vous auriez 10 étiquettes, de 0 à 9. Et donc une matrice de confusion de taille 10×10.

À chaque étiquette possible est associée un entier de 0 à 9 permettant de l'identifier. Le tableau se lit alors de la manière suivante :

- Chaque élément du tableau, placé à la ligne $i \in \{0, \dots, 9\}$ et à la colonne $j \in \{0, \dots, 9\}$ (noté M_{ij}) correspond au nombre de fois où la fonction de prédiction a produit la sortie j alors que la vraie étiquette était i.
- Ainsi les éléments sur la diagonale correspondent au nombre de prédictions justes (l'élément (i, i) correspond au nombre de fois où l'étiquette i a été correctement prédite).

On peut construire une matrice de confusion normalisée en divisant chaque élément par la somme des éléments sur la même ligne. En notant M^* la matrice de confusion normalisée, on a :

$$\forall i, j \in \{1, \dots, N\},\ M^*_{ij} = \frac{M_{ij}}{\sum_{l=1}^{N} M_{il}}$$

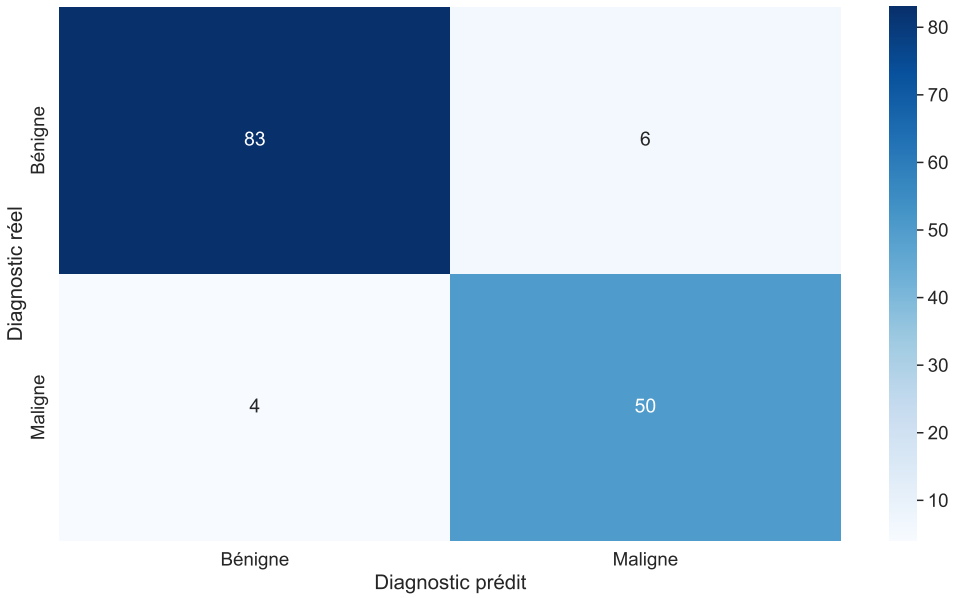
Ces deux mesures de performance portent des informations différentes :

- La matrice de confusion non normalisée permet de visualiser les plus importantes sources d'erreurs. Dans le cas (fréquent) où la répartition entre les différentes étiquettes n'est pas équilibrée, les erreurs seront majoritairement contenues dans les lignes et les colonnes correspondant à ces étiquettes.
- La matrice de confusion normalisée ne tient plus compte du déséquilibre entre étiquettes en divisant chaque ligne i par le nombre d'apparition de l'étiquette i dans la base de données (c'est à dire la somme des éléments de la ligne i de la matrice de confusion non normalisée). Celle-ci permettra d'observer les erreurs commises par la fonction de prédiction, indépendamment de la répartition des étiquettes.

3.3.4 Sensibilité et spécificité

Dans le cadre d'un test de prédiction d'une classe d'une donnée, on cherche à valider si celle-ci a été correctement faite ou non et à comprendre la répartition des erreurs. Prenons l'exemple de la dangerosité d'une tumeur présenté dans la section précédente.

Il en résulte la matrice de confusion ci-dessous :



Où les valeurs du tableaux représentent

- **VP (Vrais Positifs)** est le nombre de tumeurs bénignes correctement reconnues comme telles,
- **FP (Faux Positifs)** représente le nombre de tumeurs malignes mais qui ont été jugées bénignes,
- **FN (Faux Négatifs)** représente le nombre de tumeurs bénignes mais qui ont été jugées malignes
- **VN (Vrais Négatifs)** représente le nombre de tumeurs malignes qui ont été correctement caractérisées comme telles.

- La **sensibilité d'un critère** est la probabilité que celui-ci détecte la catégorie *positive* parmi les exemples qui le sont réellement. Il s'agit donc du rapport $\frac{VP}{VP+FN}$.
- La **spécificité d'un critère** est la probabilité que celui-ci détecte la catégorie *négative* parmi les exemples qui le sont réellement. Il s'agit donc du rapport $\frac{VN}{FP+VN}$.
- Un critère *sensible* (de sensibilité élevée) est réglé de manière à trouver tous les éléments associés à la catégorie positive, quitte à se tromper et à générer des faux positifs (FP) ce qui dégrade sa spécificité.
- Un critère *spécifique* (de spécificité élevée) est réglé de manière à trouver tous les éléments associés à la catégorie négative, quitte à générer des faux négatifs (FN) ce qui dégrade sa sensibilité.

Remarque

Avec ces notations, la justesse (*Accuracy* qui n'est autre le `score` que l'on a calculé plus haut) vaut

$$Accuracy = \frac{VP+VN}{VP+VN+FP+FN}$$

et porte donc une information différente.

À retenir

L'algorithme des k-plus proches voisins (k-*Nearest Neighbors*, k-NN) est une méthode permettant de classer des objets (ou sujets, ou sites, ou points, etc.) en fonction de leur proximité avec d'autres objets déjà étiquetés.

Il repose sur la **détermination des K voisins les plus proches d'un point à classer, selon une métrique de distance (souvent la distance euclidienne)**, puis l'attribution de la classe majoritaire parmi ces voisins.

La méthode des k-NN est un algorithme de classification **supervisé**, car il nécessite un ensemble de données d'apprentissage avec des classes connues. Elle peut être utilisée pour des problèmes de classification (attribution d'une étiquette, ce que l'on vient de voir) ou de régression (prédiction d'une valeur numérique moyenne des voisins), c'est la suite !.

Le choix du paramètre K influence directement la performance de l'algorithme : un K trop petit rend le modèle sensible au bruit, tandis qu'un K trop grand peut diluer les différences entre classes. Une bonne évaluation du modèle à l'aide de métriques comme la matrice de confusion est essentielle pour garantir de bonnes performances.

4 Un enjeu important, la régression

4.1 Intérêt de déterminer un modèle de données

4.1.1 Présentation

Lors d'une étude expérimentale, des points de mesures sont réalisés (en figure 10). Afin de traiter ces données, celles-ci sont alors souvent représentées graphiquement afin de mieux les visualiser.

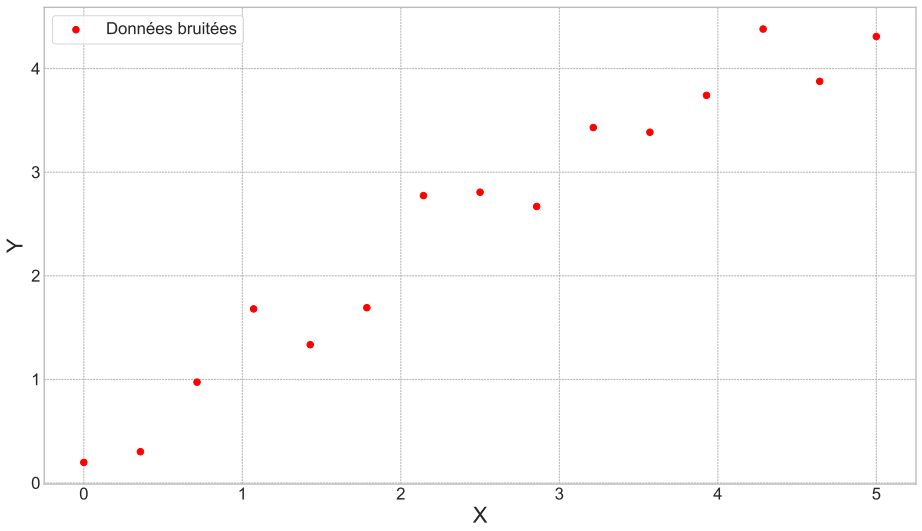


FIGURE 10 – Exemple de courbe obtenue expérimentalement.

L'ingénieur (ou plus largement le scientifique) peut alors à partir de ces données tenter de définir un modèle de comportement de ce qui a été observé. C'est-à-dire en déduire une équation liant les paramètres expérimentaux et les mesures obtenues.

Cette démarche est une démarche tout à fait classique et que vous avez déjà probablement effectuée. Le modèle que l'on vous demandait de trouver était généralement un modèle linéaire (la droite passant au mieux par le relevé de points effectués). Cette approche déductive d'un modèle à partir d'expériences est largement répandue dans le monde scientifique.

Un exemple très simple et qui est abordé au collège est d'effectuer un relevé de tension aux bornes d'une résistance ainsi que l'intensité circulant dans le circuit électrique. Ce relevé de mesures de tension et intensité permet de mettre en exergue la relation $U = R.I$.

4.1.2 Différents modèles possibles

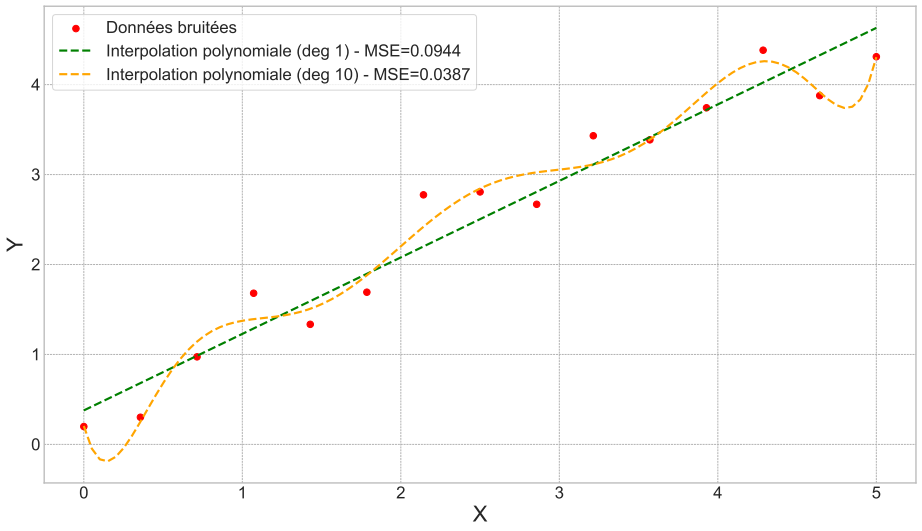
La méthode permettant de déterminer une équation à partir d'un relevé de points n'est cependant pas "magique" et nécessite en fait de faire un choix sur le type de modèle que l'on va utiliser pour déterminer la "meilleure" équation représentative des données disponibles.

Exemple:

En reprenant les données de l'exemple de la figure 10, on propose deux modèles d'interpolation/-régression de données. On notera ces deux modèles $\tilde{f}_1(x)$ et $\tilde{f}_2(x)$

- un modèle interpolant de type polynomiale : $\tilde{f}_1(x) = a \times x + b$ polynôme de degré 1
- un modèle interpolant de type polynomiale : $\tilde{f}_2(x) = \sum_{i=0}^{10} \beta_i \times x^i$ polynôme de degré 10.

Après optimisation des paramètres a, b et β_i, le résultat obtenu est :



Il apparaît évident que le polynôme de degré 10 est *plus proche* des points exéprimentaux que le polynôme de degré 1. Cependant, pour effectuer une prédiction avec une nouvelle donnée d'entrée, on risque d'obtenir un résultat complètement à coté de la plaque.

À retenir

La qualité du modèle représentatif de données obtenues par simulation numérique ou expérience dépend donc d'un choix de l'utilisateur sur le type de fonction utilisée pour la régression. Cette pratique nécessite donc un certain recul et expérience dans ce domaine pour faire des choix pertinents.

4.2 Définition de l'erreur quadratique moyenne

On dispose d'un ensemble de données $\mathcal{X} = \{\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_p\}$ avec $\mathbf{x}_i \in \mathbb{R}^n$. $\mathbf{x}_i$ peut donc être considéré comme un vecteur de dimension n. On vérifie alors :

$$\mathbf{x}_i = [\mathbf{x}_{i_1}, \mathbf{x}_{i_2}, \cdots \mathbf{x}_{i_n}]$$

Pour les données de l'ensemble $\mathcal{X}$, on dispose de leurs évaluations qui sont stockées dans l'ensemble $\mathcal{Y} = \{y_1, y_2, \cdots, y_p\}$. On vérifie $y_i \in \mathbb{R}$. (On peut généraliser à $\mathbf{y}_i \in \mathbb{R}^k$).

Dans ce cours, quel que soit le modèle utilisé pour établir une régression des données, on s'appuie sur **la minimisation de l'erreur quadratique moyenne**. On dispose d'un modèle $\tilde{f}$ dont on cherche à optimiser les paramètres `param`. (`param` est un vecteur des paramètres que l'on chercher à optimiser). On pose alors :

$$\tilde{y}_i = \tilde{f}(\mathbf{x}_i, \texttt{param})$$

L'erreur quadratique moyenne (*mean square error* en anglais) est donnée par :

$$\begin{aligned} MSE(\texttt{param}) &= \frac{1}{p}\sum_{i=1}^{p} \parallel y_i - \tilde{y}_i \parallel^2 \\ &= \frac{1}{p}\sum_{i=1}^{p} \parallel y_i - \tilde{f}(\mathbf{x}_i, \texttt{param}) \parallel^2 \end{aligned}$$

Afin de trouver le vecteur `param` le plus adapté aux données $\mathcal{X}$ et leurs évaluation $\mathcal{Y}$, il faut résoudre le problème suivant :

$$\underset{\texttt{param}}{\arg\min} \left(\frac{1}{p}\sum_{i=1}^{p} \parallel y_i - \tilde{f}(\mathbf{x}_i, \texttt{param}) \parallel^2 \right)$$

Résoudre ce problème revient à chercher la solution d'un problème de minimisation. On peut définir différents types de minimum pour une fonction f quelconque (voir figure 11)

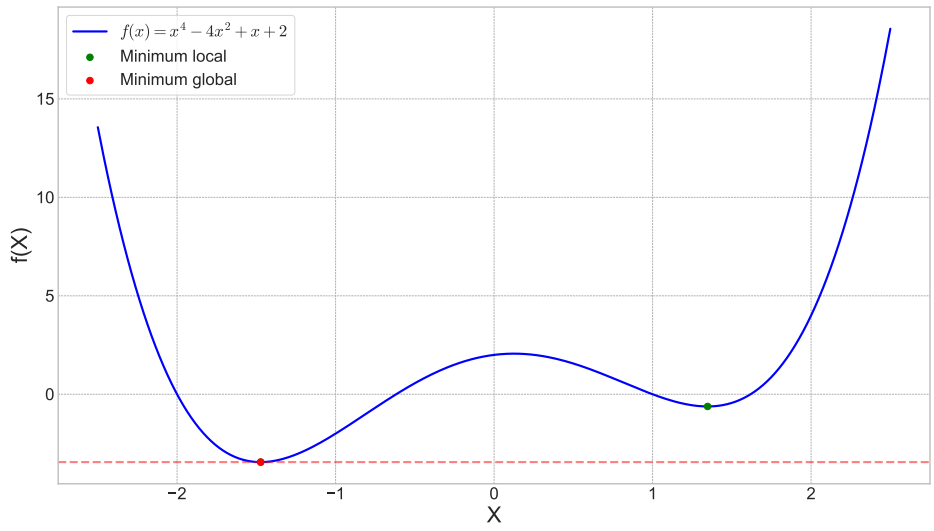


FIGURE 11 – Illustration de différents types de minima

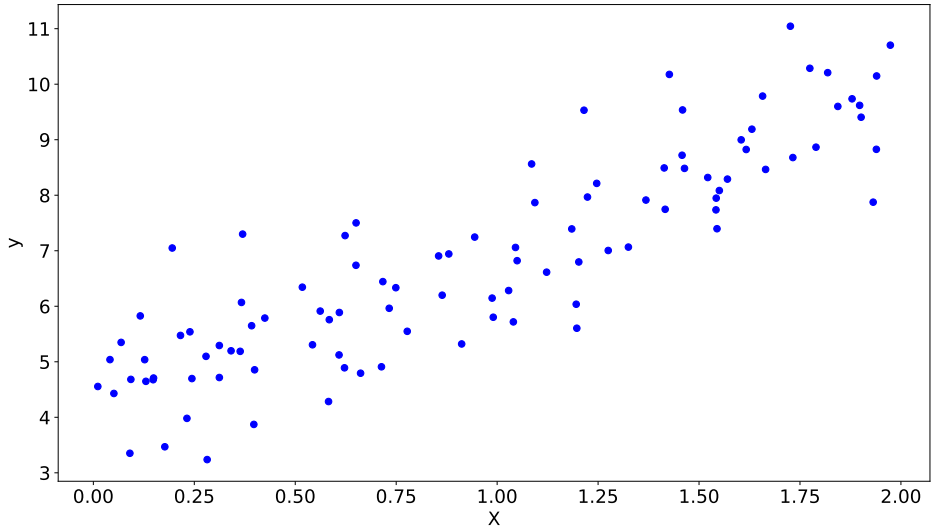
Pour résoudre, ce problème de minimisation, on peut utiliser différents algorithmes. L'un des plus simples à utiliser est l'algorithme de descente de gradient à pas constant.

4.3 Algo de descente de gradient (à pas fixe)

L'algorithme de descente de gradient à pas fixe est une méthode d'optimisation permettant de minimiser une fonction de coût en ajustant progressivement les paramètres du modèle dans la direction du gradient négatif.

Exemple:

Régression affine : Prenons un jeu de donnée quelconque, illustré par la figure suivante :



Soit une fonction de coût $J(a,b)$, typiquement l'erreur quadratique moyenne (MSE) pour un modèle affine $f(x) = ax + b$:

$$J(a,b) = \frac{1}{m}\sum_{i=1}^{m}(ax_i + b - y_i)^2$$

où (x_i, y_i) sont les points du jeu de données.

La descente de gradient met à jour a et b selon :

$$a^{(k+1)} = a^{(k)} - \alpha\frac{\partial J}{\partial a}$$

$$b^{(k+1)} = b^{(k)} - \alpha\frac{\partial J}{\partial b}$$

où α est le **pas d'apprentissage** (learning rate) et les dérivées partielles s'écrivent :

$$\frac{\partial J}{\partial a} = \frac{2}{m}\sum_{i=1}^{m} x_i(ax_i + b - y_i)$$

$$\frac{\partial J}{\partial b} = \frac{2}{m}\sum_{i=1}^{m}(ax_i + b - y_i)$$

À chaque itération, les valeurs de a et b sont ajustées pour minimiser $J(a,b)$, jusqu'à atteindre un critère d'arrêt (nombre d'itérations fixé ou convergence du gradient). Le pseudo-code se trouve ci-dessous :

Initialisation ;
Données : Une fonction $fonc$ dont on cherche un minimum. Un point $\mathbf{x}_{init} \in \mathbb{R}^n$ avec lequel on initialise l'algorithme. α est un paramètre appelé le pas. $\alpha > 0$. Un paramètre $\varepsilon > 0$ servant de critère d'arrêt ;
$\mathbf{x}_{sol} \leftarrow \mathbf{x}_{init}$;
while $|\nabla fonc(\mathbf{x}_{sol})| \geqslant \varepsilon$ **do**
  $\mathbf{x}_{sol} \leftarrow \mathbf{x}_{sol} - \alpha\nabla fonc(\mathbf{x}_{sol})$;
end
return x_{sol}

FIGURE 12 – On observe ici la valeur de l'erreur quadratique moyenne en fonction de 'évolution de paramètres a et b. Les points blancs indiquent la valeur de la MSE au fur et à mesure des itérations. Ici, on converge vers un minimum.

Remarque

Vous devriez reconnaitre la méthode de Newton utilisée en TP... Mais généralisée à une équation multiparamètres.

Ce qui itération après itération donne la figure suivante :

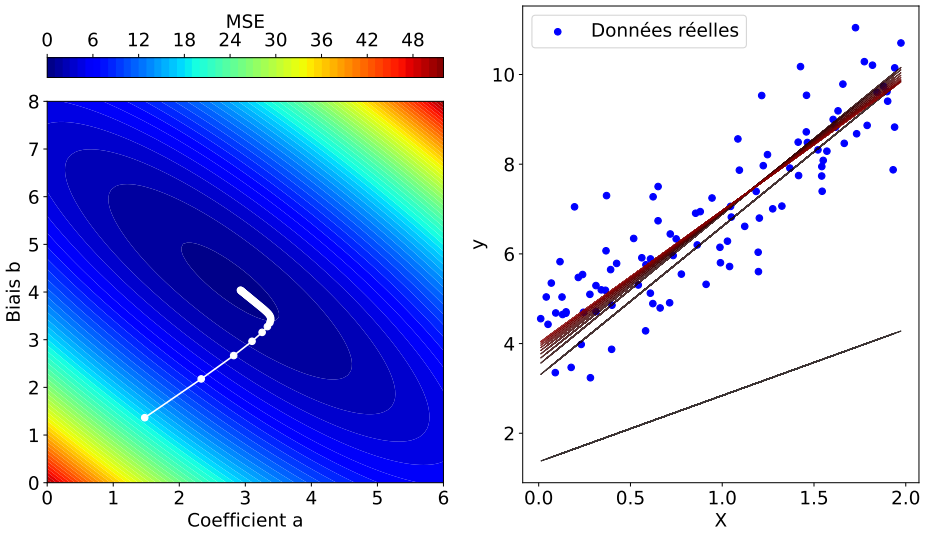


FIGURE 13 – La partie gauche de la figure représente la valeur de la fonction de coût en fonction des paramètres a et b. Les points blancs représentent l'évolution de ces paramètres au fur et à mesure des itérations de la descente de gradient. À droite, on observe le nuage de points que l'on cherche à modéliser. On observe aussi l'évolution de la modélisation affine au fil des itérations (avec les a et b correspondants).

Remarque

Il est tout à fait possible de résoudre directement ce problème en fonction du modèle $\tilde{f}$ notamment lorsque le problème est linéaire en fonction des paramètres du vecteur `param`.

4.4 Une régression non-linéaire monovariable : $\tilde{f}(t) = 1 - \exp(1 - t/\tau)$

Encore une fois, nous allons étudier un exemple pour comprendre la démarche de résolution de ce genre de problème.

Exemple:

Vous réalisez une expérience indicielle sur un système. Vous obtenez un relevé de réponse de celui-ci mais malheureusement celui-ci contient peu d'informations : 8 points (vous êtes priés de faire plus dans vos TIPE...)

Objectif

Proposer un modèle de comportement du système étudié. À la vue du relevé, le système semble pouvoir être assimilé à un système du premier ordre de fonction de transfert $H(p) = \dfrac{K}{1 + \tau.p}$.

L'entrée imposée au système est un échelon indiciel. La sortie du système a donc pour équation $s(t) = K.(1 - \mathrm{e}^{-t/\tau})$ On note très vite que le gain du système est $K = 1$. Par contre la détermination de τ parait trop hasardeuse avec les approches classiques vues en cours de Sciences de l'ingénieur.

On se propose alors de faire une régression non linéaire avec un modèle $\tilde{f}(t, \tau) = 1 - \mathrm{e}^{-t/\tau}$ dont on cherche à optimiser le paramètre τ.

4.4.1 Approche réalisée

Ici, une approche itérative est la plus indiquée pour déterminer le paramètre τ. On utilise un algorithme à descente de gradient à pas constant pour résoudre le problème.

Les instants où les mesures ont été relevées sont données dans l'ensemble $\mathcal{T} = \{0, 0.4, 0.8, 1.2, 1.6, 2, 2.4, 2.8\}$. Les valeurs de relevés effectués à ces instants sont données dans l'ensemble $\mathcal{Y} = \{0., 0.49, 0.74, 0.86, 0.93, 0.96, 0.98, 0.99\}$.

L'erreur quadratique moyenne est alors donnée par :

$$MSE(\tau) = \frac{1}{8}\sum_{i=1}^{8}\left(y_i - \left(1 - \mathrm{e}^{-t_i/\tau}\right)\right)^2$$

À titre d'illustration, voici le tracé en figure 14 de l'évolution l'erreur quadratique moyenne en fonction de τ.

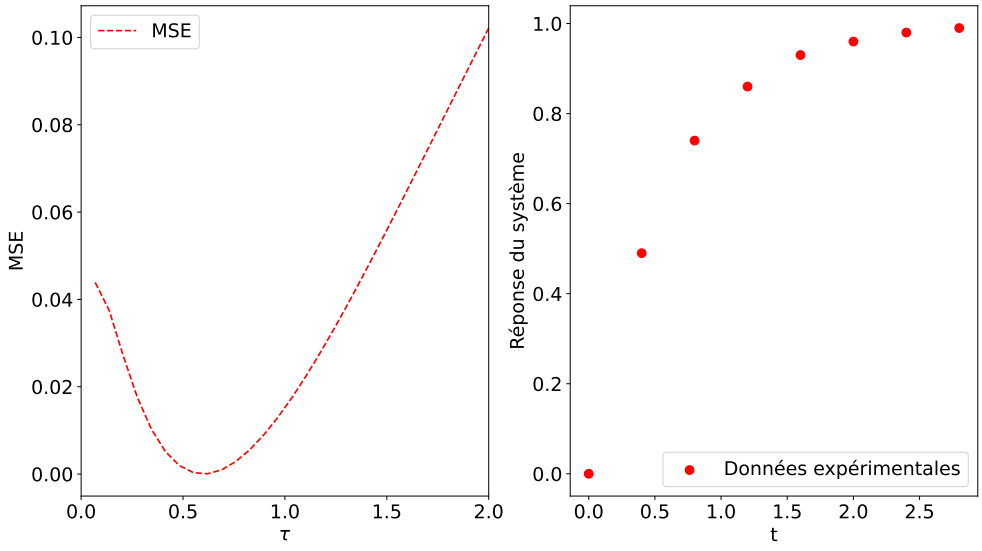


FIGURE 14 – Tracé de la fonction $MSE(\tau)$ et des points expérimentaux

4.4.2 Illustration de l'algorithme de descente de gradient

Le pas, *learning-rate* utilisé sur cet algorithme est $\alpha = 0.8$ et la valeur de τ utilisé pour initialiser l'algorithme est $\tau = 2$.

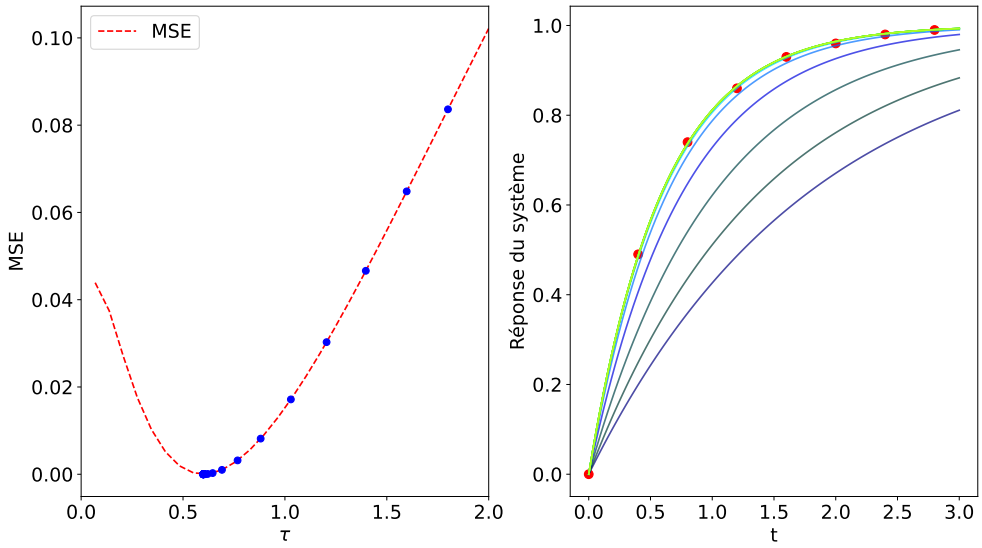


FIGURE 15 – Présentation de l'algorithme à descente de gradient

On trouve que la valeur de τ à utiliser pour notre modèle de comportement est $\tau = 0.6$ s.

4.4.3 Impact de la valeur de α

Le *learning rate* α est un paramètre clé qui contrôle l'amplitude des mises à jour du paramètre τ à chaque itération. Son choix influence fortement la convergence de l'algorithme.

Cas d'un learning rate trop petit

- **Avantages :** Convergence **assurée**.
- **Inconvénients :** Convergence **très lente**, nécessitant un grand nombre d'itérations.
- **Effets :** Risque de **stagnation** si le gradient devient trop faible.

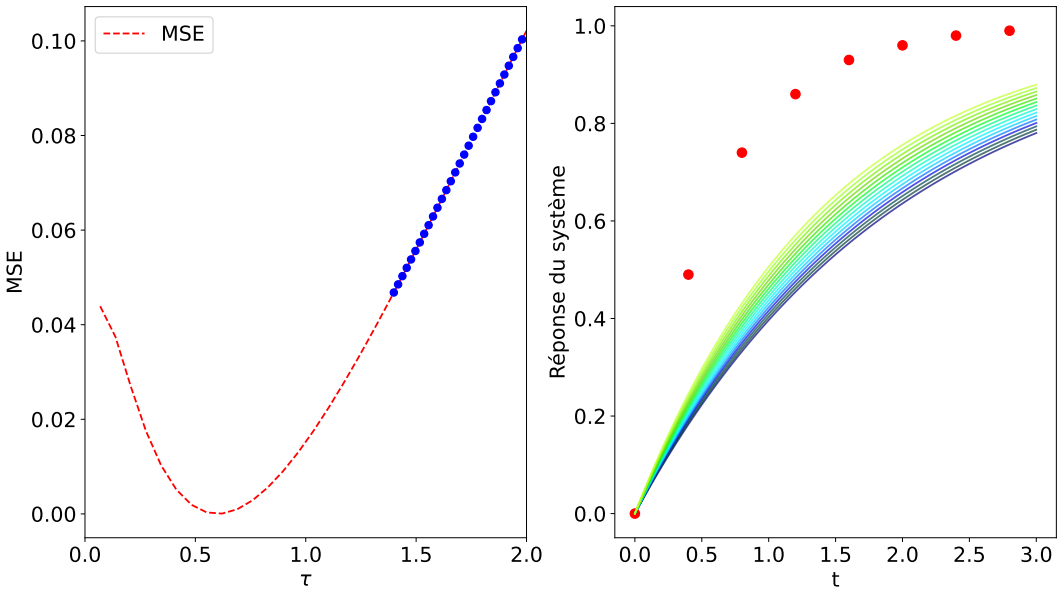


FIGURE 16 – Cas d'un learning rate trop faible, ici $\alpha = 0.01$.

Cas d'un learning rate bien choisi

- **Avantages :** Convergence **efficace et rapide**.
- **Effets :** La MSE diminue **régulièrement**, et le modèle atteint rapidement une bonne solution.

C'est le cas présenté dans la section précédente.

Cas d'un learning rate trop grand

- **Inconvénients :** L'algorithme **diverge** : les paramètres oscillent ou explosent.
- **Effets :** La MSE **augmente** au lieu de diminuer.
- **Conclusion :** L'algorithme **ne trouve jamais** une solution optimale.

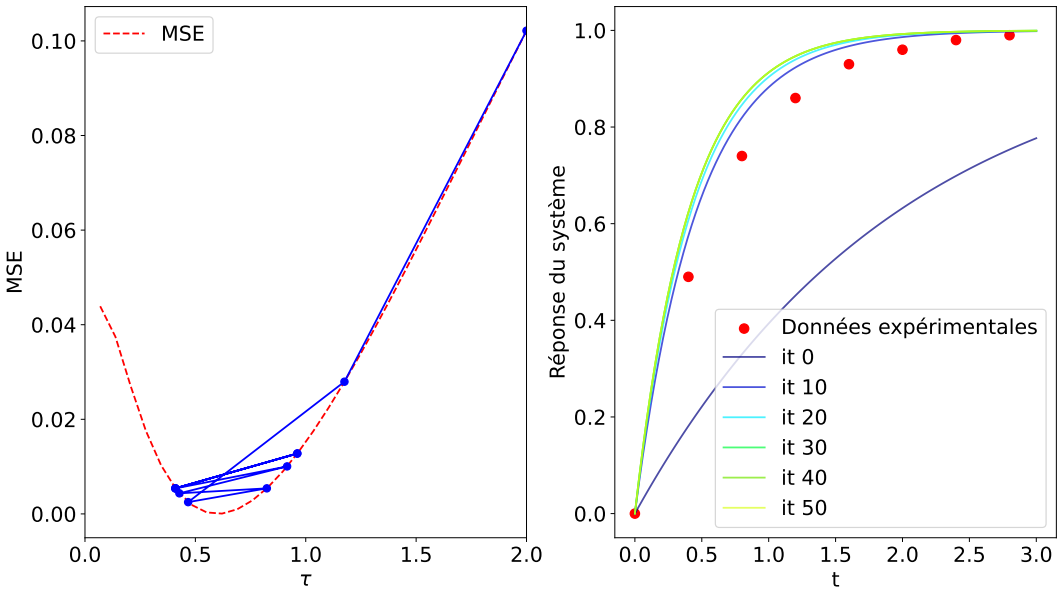


FIGURE 17 – Cas d'un learning rate trop grand, ici $\alpha = 3.1$.

À retenir

La régression est un outil fondamental pour modéliser des relations entre des données expérimentales et en extraire des tendances exploitables. Le choix du modèle et la méthode d'optimisation des paramètres influencent directement la qualité des prédictions.

L'erreur quadratique moyenne permet d'évaluer l'écart du modèle aux données, et des techniques comme la descente de gradient permettent de déterminer les valeurs des paramètres en minimisant cette erreur.

Nous avons vu également que le choix du pas d'apprentissage et la nature du modèle influencent fortement la convergence et la pertinence des résultats.

4.5 Et avec les k plus proches voisins ?

On peut effectuer une régression avec l'algorithme des k-plus proches voisins (kNN), c'est à dire estimer une valeur numérique en fonction des points les plus proches dans l'ensemble d'apprentissage. Le but étant de prédire la valeur d'un nouvesl échantillon, l'algorithme recherche ses k voisins les plus proches en utilisant une mesure de distance, comme la distance euclidienne. Jusqu'ici c'est la même idée que pour la classification sauf que cette fois ci la donnée de sortie, la valeur prédite, est obtenue en **faisant la moyenne (ou parfois une pondération)** des valeurs des k voisins.

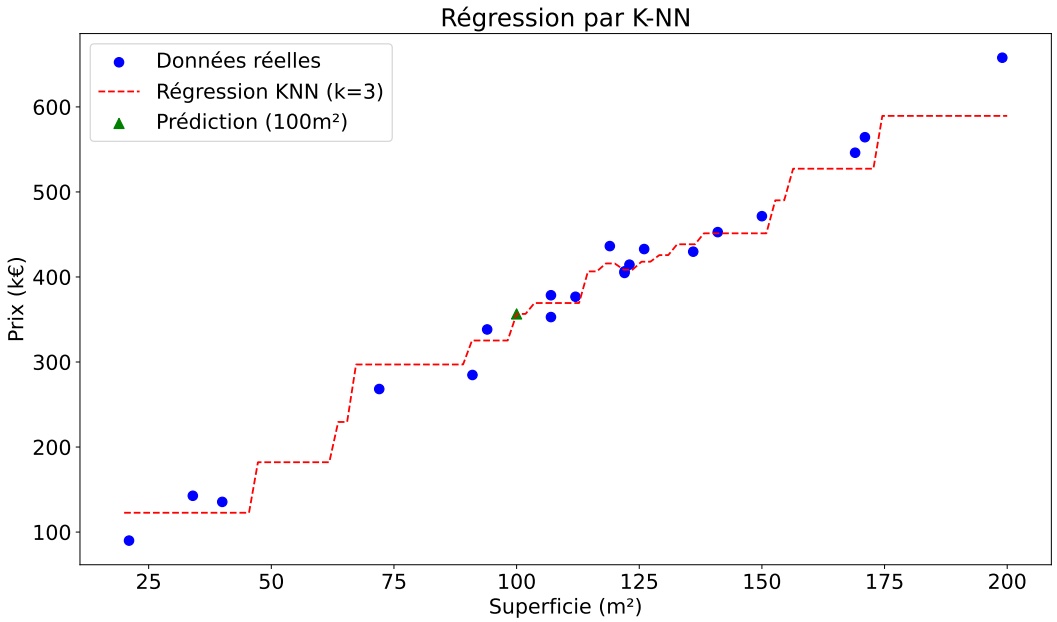


FIGURE 18 – Exemple d'une régressions par kNN sur le prix d'une maison en fonction de sa superficie (cas ultra simple évidemment avec seulement deux données)

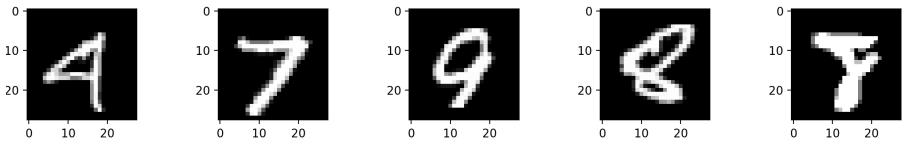
5 Modèle du perceptron

Cette section est rédigée en très grande partie grâce aux excellents sites :

- de Michael Nielsen
- de Grant Sanderson Chaine youtube à consommer sans modération.

5.1 Introduction

Le système visuel humain est absolument incroyable. Prenons la séquence suivante de chiffres écrits à la main :



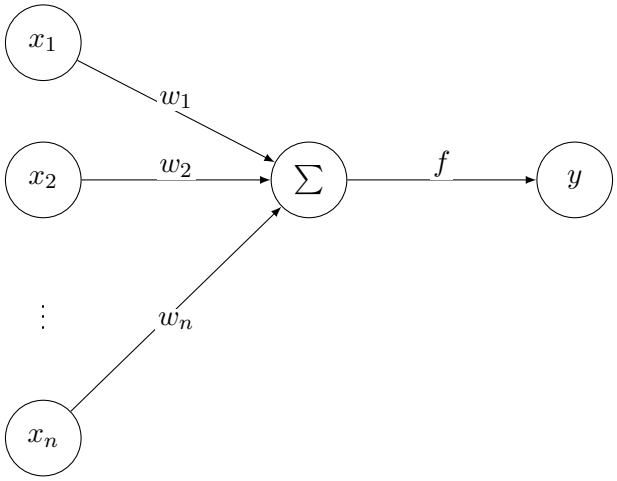
Normalement vous devriez reconnaitre sans peine ces chiffres comme étant 47988. Et pourtant, reconnaître des chiffres écrits à la main n'est pas facile bien au contraire, nous ne nous rendons généralement pas compte de la difficulté du problème résolu par notre système visuel.

La difficulté de cette reconnaissance des chiffres devient évidente si vous essayez d'écrire un programme informatique pour reconnaître des chiffres comme ceux ci-dessus. Ce qui semble facile lorsque nous le faisons nous-mêmes devient soudain extrêmement difficile. Des intuitions simples sur la façon dont nous reconnaissons les formes - « un 9 a une boucle en haut et un trait vertical en bas à droite » - s'avèrent moins simples à exprimer algorithmiquement. Bref, un enfer Pythonesque.

Les réseaux de neurones artificiels (RNA) sont des modèles inspirés du fonctionnement du cerveau humain. Ils sont constitués de neurones artificiels organisés en couches et sont largement utilisés en intelligence artificielle pour des tâches telles que la classification, la régression et la reconnaissance de formes. Ce chapitre vise à introduire les principes fondamentaux des réseaux de neurones, en se concentrant sur le perceptron et les notions associées.

5.2 Modèle du Perceptron

Le perceptron est un modèle de neurone artificiel utilisé pour la classification binaire. Il réalise une combinaison linéaire de ses entrées, suivie d'une activation par une fonction seuil permettant de produire une sortie.



5.2.1 Fonctionnement du Perceptron

Le perceptron fonctionne en trois étapes :

Somme pondérée des entrées Chaque entrée x_i est multipliée par un poids w_i. On effectue ensuite une somme pondérée de ces entrées avec un biais b :

$$S = \sum_{i=1}^{n} w_i x_i + b$$

où :
- S est la somme pondérée des entrées,
- b est le biais, un paramètre ajustable,
- w_i sont les poids associés aux entrées x_i.

Application d'une fonction d'activation Une fonction d'activation f est appliquée à la somme S pour produire la sortie y :

$$y = f(S)$$

Une fonction typique pour le perceptron est la fonction seuil :

$$f(S) = \begin{cases} 1, & \text{si } S \geq 0 \\ 0, & \text{si } S < 0 \end{cases}$$

Cette fonction permet de prendre une décision binaire.

Apprentissage La sortie de la fonction d'activation est la prédiction finale du perceptron. Pendant la phase d'apprentissage, cette prédiction est comparée à la vraie valeur pour calculer l'erreur. Si la prédiction est correcte, aucun ajustement n'est effectué. En revanche, si la prédiction est incorrecte, le perceptron ajuste ses poids et son biais dans le but de réduire l'erreur lors des prédictions futures.

Exemple de calcul Considérons un perceptron avec :

- Entrées : $x_1 = 0.5$, $x_2 = 0.3$, $x_3 = 0.7$
- Poids : $w_1 = 0.8$, $w_2 = -0.4$, $w_3 = 0.6$
- Biais : $b = -0.2$

Le calcul de S donne :

$$S = (0.8 \times 0.5) + (-0.4 \times 0.3) + (0.6 \times 0.7) + (-0.2)$$

$$S = 0.4 - 0.12 + 0.42 - 0.2 = 0.5$$

Si on applique la fonction seuil :

$$y = f(0.5) = 1$$

Le perceptron active donc la sortie $y = 1$.

5.2.2 Limites du perceptron

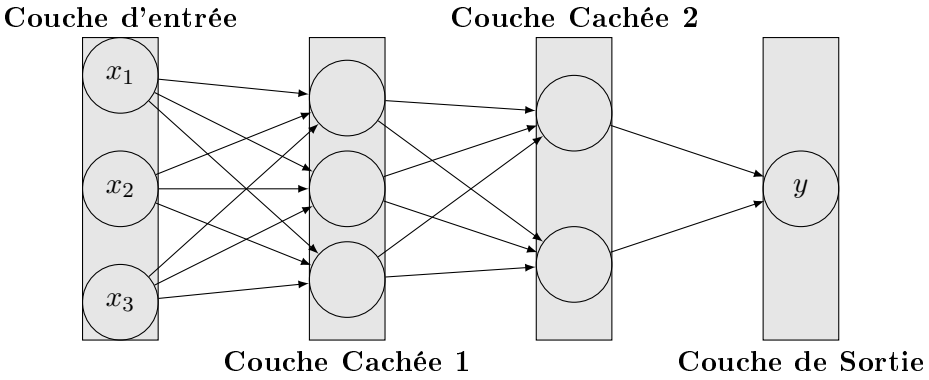
Le perceptron possède plusieurs limites te beaucoup sont hors programme, mais il y en a (au moins) deux qui l'excluent de toute application ou presque :

- **Classification binaire** En tant que classificateur binaire, il est incapable de gérer directement les problèmes de classification multiclasse. L'approche standard pour les surmonter implique l'utilisation de plusieurs perceptrons en parallèle.
- **Séparabilité linéaire** Il ne peut classer correctement que des données qui sont linéairement séparables. Par exemple, il échoue avec le problème du XOR (ou exclusif) où aucune ligne droite ne peut séparer les points positifs et négatifs.

C'est pourquoi nous allons parler de réseau de neurones, c'est à dire l'utilisation de plusieurs perceptron en série et en parallèles.

6 Réseau de Neurones

Un réseau de neurones artificiels est composé de plusieurs couches de neurones interconnectés.



Le réseau ci-dessus est composé de plusieurs couches :

- **Couche d'entrée** : contient les neurones x_1, x_2, x_3, qui reçoivent les données en entrée.
- **Première couche cachée** : transforme les entrées grâce aux connexions pondérées.
- **Deuxième couche cachée** : affine la représentation des données.
- **Couche de sortie** : produit le résultat final y.

Chaque connexion entre neurones est associée à un poids, qui est ajusté lors de l'apprentissage pour minimiser l'erreur du modèle.

6.1 Choix du Nombre de Neurones en Entrée et en Sortie

Le choix du nombre de neurones en entrée et en sortie est crucial pour la performance du réseau de neurones.

6.1.1 Nombre de Neurones en Entrée

Le nombre de neurones en entrée correspond au nombre de caractéristiques (ou variables) utilisées pour effectuer une prédiction. Par exemple :

- Pour la reconnaissance d'image en niveaux de gris de 28×28 pixels, il faut $28 \times 28 = 784$ neurones en entrée.
- Pour un problème de prédiction basé sur trois paramètres (ex. température, humidité, pression), on aura trois neurones en entrée.

Le bon choix du nombre d'entrées est fondamental, car un nombre insuffisant peut entraîner une perte d'information, tandis qu'un nombre trop grand peut augmenter la complexité inutilement.

6.1.2 Nombre de Neurones en Sortie

Le nombre de neurones en sortie dépend du type de tâche réalisée :

- Pour un problème de classification binaire (ex. chat ou chien), un seul neurone de sortie avec une activation sigmoïde est suffisant.

- Pour une classification multi-classes (ex. reconnaissance de chiffres manuscrits de 0 à 9), on utilise généralement un neurone par classe avec une activation softmax.
- Pour un problème de régression, la sortie est souvent un unique neurone donnant une valeur continue.

Le bon choix du nombre de neurones en entrée et en sortie est essentiel pour garantir les performances du modèle.

6.2 Choix de la Fonction d'Activation

La fonction d'activation permet d'introduire de la non-linéarité dans le réseau, ce qui est essentiel pour modéliser des relations complexes entre entrées et sorties. Voici quelques fonctions d'activation courantes :

- **Fonction Seuil (Heaviside)** : $f(z) = 1$ si $z \geq 0$, sinon $f(z) = 0$. Utilisée dans le perceptron classique.
- **Sigmoïde** : $f(z) = \frac{1}{1+e^{-z}}$, permet de lisser la transition entre 0 et 1.
- **Tangente hyperbolique** : $f(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$, centre les valeurs autour de 0.
- **ReLU (Rectified Linear Unit)** : $f(z) = \max(0, z)$, très utilisée en deep learning pour sa simplicité et son efficacité.

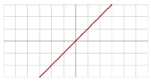
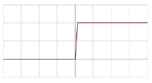
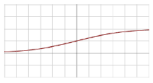
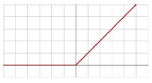
Identité	Heaviside	Logistique (sigmoïde)	Unité de rectification linéaire (ReLU)
			
$\sigma(x) = x$	$\sigma(x) = \begin{cases} 0 \text{ si } x < 0 \\ 1 \text{ si } x \geq 0 \end{cases}$	$\sigma(x) = \frac{1}{1+\mathrm{e}^{-x}}$	$\sigma(x) = \begin{cases} 0 \text{ si } x < 0 \\ x \text{ si } x \geq 0 \end{cases}$

FIGURE 19 – Une partie des différentes fonctions d'activation utilisées.

6.3 Mise à jour des biais et des poids

Dans un réseau de neurones, la mise à jour des poids et des biais s'effectue par un processus d'apprentissage basé sur la **descente de gradient** (ohhh mais quel hasard). Dans notre cas exemple de la reconaissance d'image, nous allons lui soumettre des images dont on connait l'étiquette (la valeur du chiffre écrit dessus). On établit alors une fonction de coût (typiquement les moindres carrés) caractérisant l'écrat de la prédiction à la réalité. Ce qui fait de notre réseau de neurones un algorithme de type **supervisé**.

Remarque

Les tâches du réseau de neurones sont souvent classées comme discriminatives (reconnaissance) ou génératives (imagination). Les tâches discriminatives utilisent souvent (mais pas toujours) des méthodes supervisées et les tâches génératives utilisent des méthodes non supervisées ; cependant, la séparation est très floue. (*Wikipedia*)

Ce que l'on vous demande de retenir, est surtout de savoir discriminer les cas où l'on connait l'étiquette de sortie et que l'on veut classifier des données (**supervisé**) et les cas où aucune étiquette n'est connue en sortie et l'algorithme génère ses propres partitions de données (**non supervisé**.

À chaque itération de l'entraînement :

1. **Propagation avant** : Les entrées traversent les couches du réseau pour produire une prédiction.
2. **Calcul de l'erreur** : La différence entre la sortie prédite et la sortie attendue est évaluée via une fonction de coût.
3. **Rétropropagation** : L'erreur est propagée en arrière dans le réseau pour calculer les gradients des poids et des biais par rapport à la fonction de coût.
4. **Mise à jour des paramètres** : Les poids et biais sont ajustés en fonction du gradient calculé, selon la règle :
$$w \leftarrow w - \eta \frac{\partial J}{\partial w}, \quad b \leftarrow b - \eta \frac{\partial J}{\partial b}$$
où η est le taux d'apprentissage et J la fonction de coût.

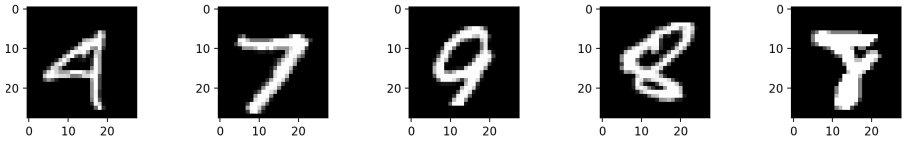
Ce processus est répété sur plusieurs échantillons pour minimiser l'erreur du réseau et améliorer sa capacité de prédiction. Ces calculs et cette démarche ne sont pas à connaitre.

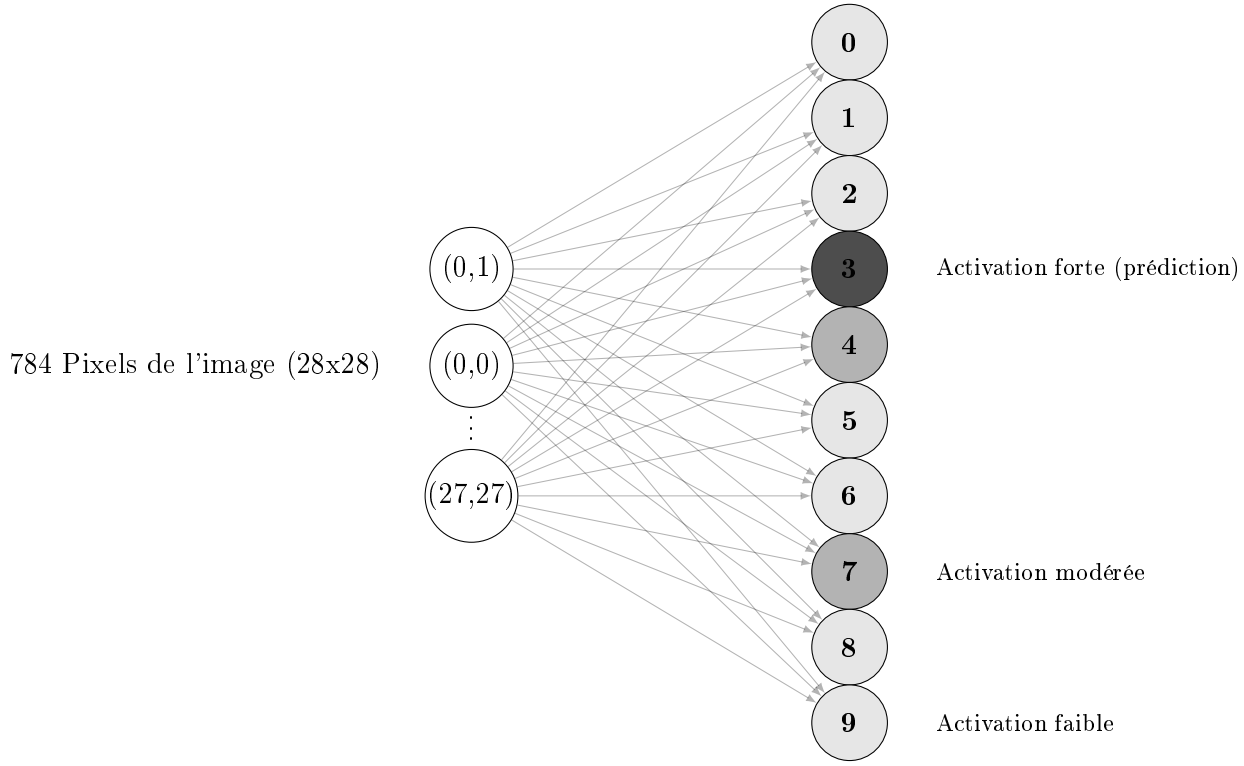
Une fois que le réseau est entrainé. On peut tester sur un jeu de Test et commencer à effectuer des prédictions : il s'agit de la phase dite **d'inférence**.

6.4 Exemple guidé :

6.4.1 Présentation du réseau

Reprenons le cas de la reconnaissance de caractère :



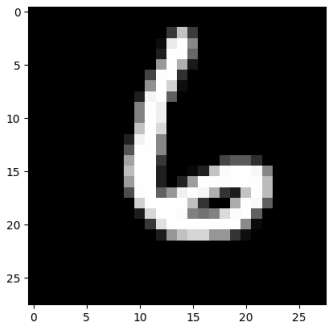


Ce graphe représente un réseau de neurones utilisé pour la reconnaissance de chiffres manuscrits (MNIST) sur Python. On prend le réseau de neurones le plus simple possible :

- La couche d'entrée contient 784 neurones (correspondant aux pixels d'une image 28x28). Pour des raisons de lisibilité, seuls quelques neurones sont affichés, et les autres sont indiqués par des pointillés.
- La couche de sortie contient 10 neurones, chacun représentant un chiffre entre 0 et 9. Chaque neurone a un niveau d'activation, visualisé ici par une échelle de gris :
  - ◇ **Gris foncé** : neurones fortement activés.
  - ◇ **Gris clair** : neurones faiblement activés.

6.4.2 Première prédiction

On initialise de manière aléatoire les poids et les biais de chacun des neurones. On lui donne une image quelconque et on regarde la valeur des 10 neurones de sortie. (Étape 1 : propagation avant



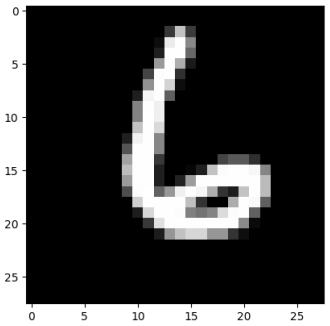
```
Prediction:  [9]
Label:  6
[[0.10967918]
 [0.11548322]
 [0.0196535 ]
 [0.22542634]
 [0.03481165]
 [0.02089338]
 [0.07810432]
 [0.0765011 ]
 [0.02443756]
 [0.29500975]]
```

Donc d'après notre réseau, le chiffre prédit est le neurone de sortie ayant la plus grosse activation : 9.

C'est faux ! Nous allons donc calculer la fonction de coût mettre à jour les biais et les poids. (Étapes 2, 3 et 4.)

6.4.3 Mis à jour des poids et des biais

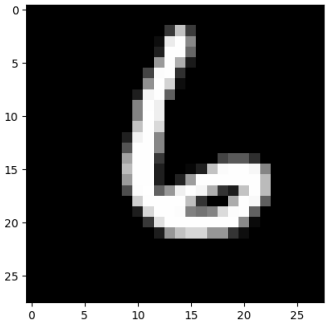
Le code et la démarche ici ne sont pas présentés, mais voilà le résultat pour **20 itérations** :



```
Prediction:  [9]
Label:  6
[[0.13821311]
 [0.06978264]
 [0.0692152 ]
 [0.09887596]
 [0.10809661]
 [0.06501081]
 [0.11841607]
 [0.10667198]
 [0.08194353]
 [0.14377409]]
```

Si le neurone 9 n'est plus autant activé, il est toujours majoritaire et la prédiction est ... fausse !

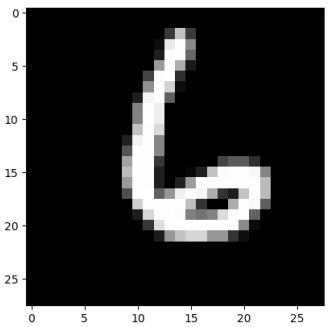
100 itérations :



```
Prediction:  [6]
Label:  6
[[1.21430109e-01]
 [6.01845526e-04]
 [4.19893884e-02]
 [8.17668668e-03]
 [1.83623796e-02]
 [4.80752911e-02]
 [7.35232319e-01]
 [1.38695889e-03]
 [5.61437960e-03]
 [1.91306423e-02]]
```

Ça y est la prédiction est juste ! Les activations des autres neurones sont relativement faibles devant celle du 6.

500 itérations :



```
Prediction:  [6]
Label:  6
[[3.16481510e-03]
 [1.04709613e-06]
 [1.70069602e-03]
 [1.94307896e-05]
 [1.44083879e-03]
 [1.45951593e-03]
 [9.91979001e-01]
 [9.07150946e-08]
 [1.56826465e-04]
 [7.77384068e-05]]
```

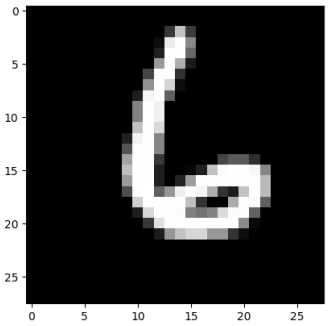
La prédiction est toujours juste, remarquez à quel point l'activation des autres neurones est très faible devant le 6.

Parfait on remarque que notre réseau s'améliore avec le nombre d'itérations de minimisation de la fonction de coût. La question que l'on peut se poser est alors la suivante :

Mais alors pourquoi ne pas faire des milliers et des milliers d'itérations de minimisation ?

7 Le sur-apprentissage

En effet, après **5000 itérations** :



```
Prediction:  [6]
Label:  6
[[3.03519794e-04]
 [1.79889862e-08]
 [3.29820018e-04]
 [1.95812536e-06]
 [1.54354232e-03]
 [6.43480039e-05]
 [9.97756154e-01]
 [3.24914956e-10]
 [4.24077596e-07]
 [2.15670280e-07]]
```

On peut dire que le réseau de neurones prédit bien un 6.

Seulement comme pour les KNN vu plus haut, nous avons entrainé notre réseau de neurones sur un jeu de données. Une toute petite base de donnée puisqu'elle comporte :

60000 **Caractères**

Comme auparavant, nous avons séparé cette base de données en données d'entrainement et en données de test.

Dans la figure ci-dessous on réalise un entrainement sur le jeu de données qui fait donc un cycle dans notre réseau de neurones : c'est la phase d'apprentissage. Á la fin d'un cycle on dispose déjà d'un premier modèle de réseau. On détermine alors l'erreur de prédiction commise en utilisant le jeu d'entrainement. La phase de validation consiste alors à tester le modèle du réseau de neurones déterminé. On détermine alors l'erreur de prédiction commise sur le jeu de test.

Remarque : le jeu de test ne modifie pas les poids et les biais

Et on itère cette démarche !

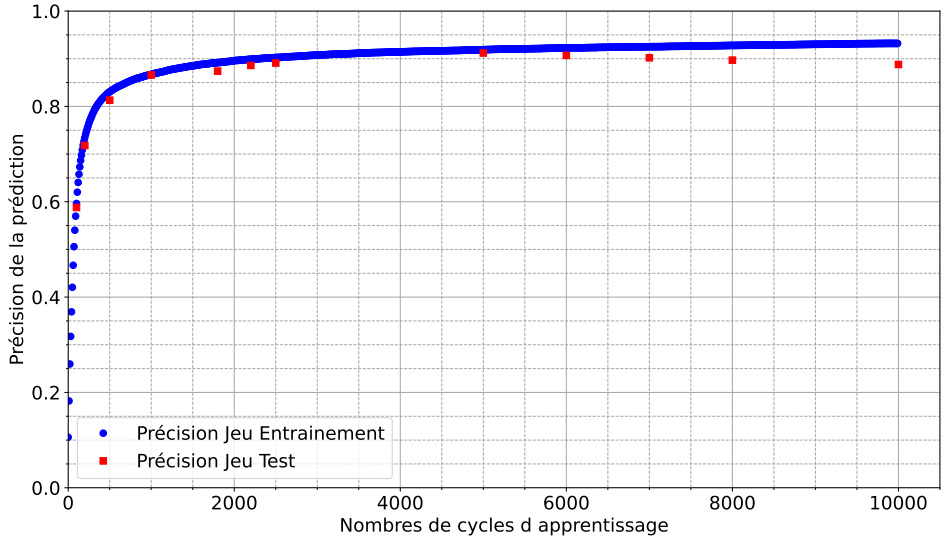
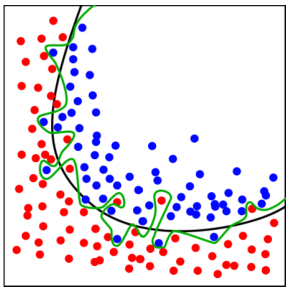


FIGURE 20 – Évolutions de la précision sur les jeux d'entrainement et de test en fonction du nombre de cycles.

« Logiquement » la précision croît toujours avec le jeu d'entraînement car la descente du gradient a pour objectif de réduire l'erreur de prédiction. Sur le jeu de test, l'erreur de prédiction décroît pendant un certain nombre de cycles puis augmente. Il existe en fait un stade à partir duquel le réseau de neurones se spécialise trop sur le jeu d'entraînement et devient donc incapable de réaliser des prédictions fiables sur un nouveau jeu de données. En effet le modèle s'est attaché à représenter/comprendre des détails du jeu d'entrainement. On parle de surentraînement ou sur-apprentissage (ou *overfitting*). La figure ci contre (*wikipedia*) permet d'illustrer ce phénomène. La ligne verte représente un modèle sur-appris et la ligne noire représente un modèle régulier. La ligne verte classifie trop parfaitement les données d'entrainement, elle généralise mal et donnera de mauvaises prévisions futur avec de nouvelles données. Le modèle vert est finalement moins bon que le noir.



L'objectif est donc d'arrêter l'entraînement du réseau de neurones lorsque l'erreur de prédiction sur le jeu de test. Il ne faut pas s'arrêter trop tôt car alors le réseau de neurones serait sous-entrainé.

Et pour les méthodes de régression et k-plus proches voisins ? Dans le cadre d'une régression polynomiale, un sous-apprentissage correspondrait au fait d'avoir un degré de régression trop faible. Un surapprentissage serait l'équivalent de faire une régression polynomiale avec un degré de régression trop important.

Sur cet exemple,[1] le vrai modèle à prédire est un polynôme de degré 3 + bruit.

En considérant les données (les cercles sur la figure précédente) comme le jeu de données d'apprentissage, on constate que l'erreur de prédiction du polynôme de degré 13 est nulle. Par contre son erreur de prédiction sur un nouveau jeu de données serait plus importante qu'avec le polynôme de de degré 3.

Pour les k-plus proches voisins, le surapprentissage est présent lorsque k est trop petit
Pour les k-plus proches voisins, le sous-apprentissage est présent lorsque k est trop grand

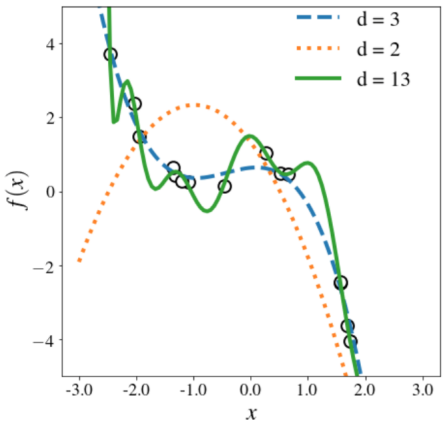


FIGURE 21 – Régression polynômiale de différents degré

1. Issu de la formation stage liesse proposé par l'Ecole des Mines ParisTech par Chloé-Agathe Azencott, Mai 2021.