Cours - Présentation de l'Intelligence Artificielle

I Introduction

La notion d'Intelligence Artificielle voit le jour dans les années 1950 grâce au mathématicien Alan Turing. Dans son livre *Computing Machinery and Intelligence*, ce dernier soulève la question d'apporter aux machines une forme d'intelligence. Il décrit alors un test aujourd'hui connu sous le nom « Test de Turing » dans lequel un sujet interagit à l'aveugle avec un autre humain, puis avec une machine programmée pour formuler des réponses sensées. Si le sujet n'est pas capable de faire la différence, alors la machine a réussi le test et, selon l'auteur, peut véritablement être considérée comme « intelligente ».

Le but de l'intelligence Artificielle (IA) est de concevoir des systèmes capables de reproduire le comportement de l'humain dans ses activités de raisonnement. L'IA se fixe comme but la modélisation de l'intelligence prise comme phénomène (de même que la physique, la chimie ou la biologie qui ont pour but de modéliser d'autres phénomènes).

Nous allons découvrir dans ce chapitre quelles sont les champs d'application de l'IA, ainsi qu'une rapide classification des différents types d'IA utilisés.

L'intelligence artificielle (IA) comprend plusieurs domaines :

— Le dialogue automatique : se faire comprendre d'un ordinateur en lui parlant ;
— la traduction automatique, si possible en temps réel ou très légèrement différé ;
— le raisonnement automatique (systèmes experts) ;
— l'apprentissage automatique ;
— la reconnaissance de formes, des visages et la vision en général ;
— l'intégration automatique d'informations provenant de sources hétérogènes ;
— l'aide aux diagnostics ;
— l'aide à la décision ;
— la résolution de problèmes complexes, tels que les problèmes d'allocation de ressources ;
— l'assistance par des machines dans les tâches dangereuses, ou demandant une grande précision.

L'intelligence artificielle est utilisée (ou intervient) dans une variété de domaines tels que :

— la banque, avec des systèmes experts d'évaluation de risque lié à l'octroi d'un crédit ;
— le militaire, avec les systèmes autonomes tels que les drones ;
— les jeux, domaine qui sera développé dans un chapitre ultérieur ;
— la médecine, avec les systèmes experts d'aide au diagnostic ;
— la logistique, au travers d'approches heuristiques de type résolution de problème de satisfaction de contraintes.

II Machine Learning

II.1 Principes généraux et utilisation du Machine Learning

Le Machine Learning est un domaine d'étude de l'IA qui vise à donner aux machines la capacité d'apprendre. Cette technologie très puissante a permis le développement des voitures autonomes, de

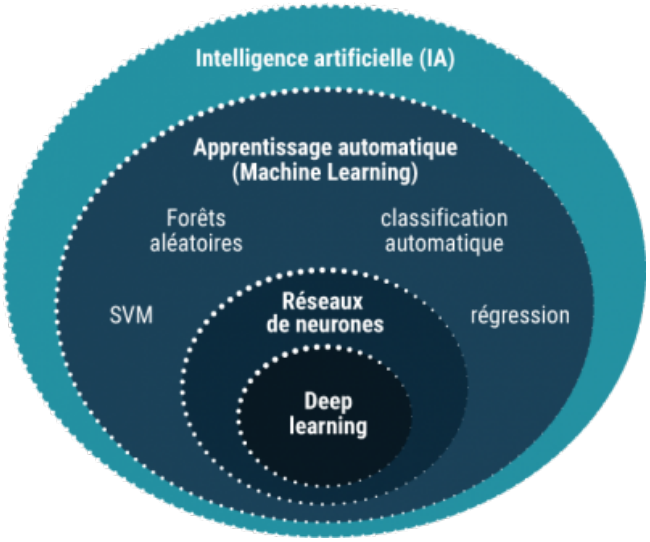

FIGURE 1 – Entités au sein de l'IA

la reconnaissance vocale, et de tous les systèmes dits "intelligents" depuis le début du siècle.

Le Machine Learning a été inventé par Arthur Samuel en 1959, après que celui-ci ait développé le premier programme de jeu de Dames doté d'une intelligence artificielle. Ce programme avait appris à jouer aux Dames tout seul, sans recevoir la moindre instruction de son développeur.

II.2 Trois méthodes d'apprentissage

On distingue en général trois grandes catégories de *machine learning* :

— L'apprentissage supervisé avec la classification qui permet de labelliser des objets comme des images et la régression qui permet de réaliser des prévisions sur des valeurs numériques. L'apprentissage est supervisé car il exploite des bases de données d'entraînement qui contiennent des labels ou des données contenant les réponses aux questions que l'on se pose. En gros, le système exploite des exemples et acquiert la capacité à les généraliser ensuite sur de nouvelles données de production.

— L'apprentissage non supervisé avec le clustering et la réduction de dimensions. Il exploite des bases de données non labellisées. Ce n'est pas un équivalent fonctionnel de l'apprentissage supervisé qui serait automatique. Ses fonctions sont différentes. Le clustering permet d'isoler des segments de données spatialement séparés entre eux, mais sans que le système donne un nom ou une explication de ces clusters. La réduction de dimensions (ou embedding) vise à réduire la dimension de l'espace des données, en choisissant les dimensions les plus pertinentes. Du fait de l'arrivée des big data, la dimension des données a explosé et les recherches sur les techniques d'embedding sont très actives.

— L'apprentissage par renforcement pour l'ajustement de modèles déjà entraînés en fonction des réactions de l'environnement. C'est une forme d'apprentissage supervisé incrémental qui utilise des données arrivant au fil de l'eau pour modifier le comportement du système. C'est utilisé par exemple en robotique, dans les jeux ou dans les chatbots capables de s'améliorer en fonction des réactions des utilisateurs. Et le plus souvent, avec le sous ensemble du machine learning qu'est le deep learning. L'une des variantes de l'apprentissage par renforcement est l'apprentissage supervisé autonome notamment utilisé en robotique où l'IA entraîne son modèle en déclenchant d'elle même un jeu d'actions pour vérifier ensuite leur résultat et ajuster son comportement.

Dans les trois catégories décrites sommairement ci-dessous, on peut classer différents algorithmes pour chaque type d'apprentissage automatique (Figure 2) :

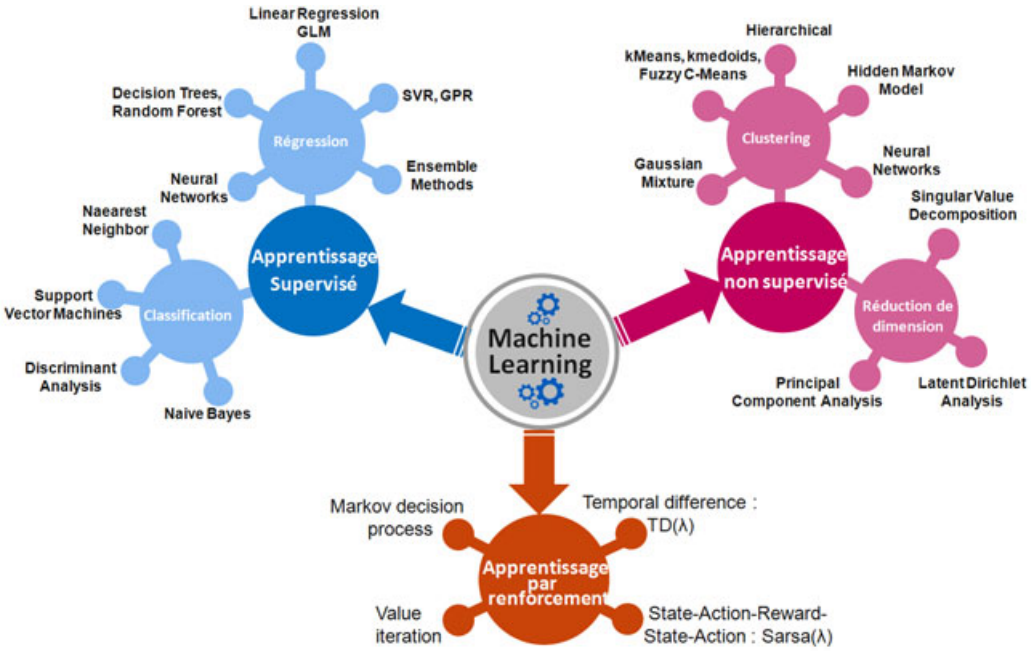

FIGURE 2 – Types d'apprentissages automatiques

Nous développerons plus particulièrement les méthodes d'apprentissage supervisé et non supervisé qui sont au programme.

On parle de modèle d'intelligence artificielle supervisé lorsque celle-ci apprend par le biais de données d'exemple. Cette méthode d'apprentissage s'oppose au non-supervisé qui désigne des intelligences artificielles pour lesquelles on fournit des données uniquement, la tâche étant de les ranger/grouper de manière "logique". La figure ci-dessous illustre de manière très simplistes les différences entre apprentissage supervisé et non-supervisé.

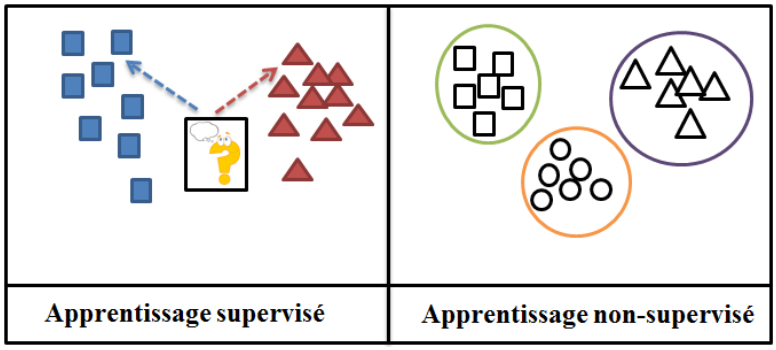

FIGURE 3 – Différences entre apprentissage supervisé et non supervisé

II.3 Principes généraux de l'apprentissage supervisé

L'apprentissage supervisé est utilisé pour développer des modèles prédictifs, c'est-à-dire des modèles capables de prédire une variable y, qui peut être un vecteur, en fonction de plusieurs variables x_1, x_2, etc.

Exemples : Prédire le nom d'une personne (y) à partir d'une photo (x), ou bien prédire le prix d'un appartement (y) en fonction de sa surface habitable (x_1) et du nombre de pièces (x_2)

Pour développer de tels modèles, il faut en premier lieu fournir à la machine une grande quantité de données (x, y). On appelle cela un dataset (un jeu de données). Ensuite, on demande à la machine de développer une fonction d'approximation qui représente au mieux la relation $x \to y$ présente dans des données. Pour cela, on utilise un algorithme d'optimisation qui minimise les écarts entre la fonction et les données du dataset.

Les applications de l'apprentissage supervisé sont nombreuses. On peut néanmoins les diviser en deux catégories de problèmes : les régressions, et les classifications. Les deux graphes ci-dessous montrent les différences principales entre un problème de classification (à gauche) et régression (à droite).

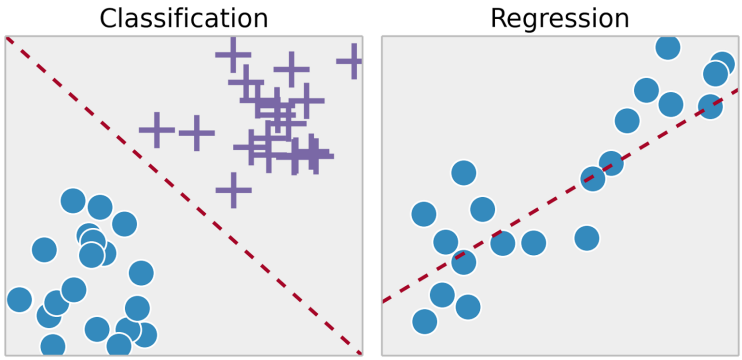

FIGURE 4 – Différences entre classification (à gauche) et régression (à droite)

II.3.1 Quelques définitions et notations

On considérera dans la suite que les données sont représentées par leurs entrées et leurs sorties.

— L'entrée X est représentée par un vecteur de valeurs d'attributs réels (représentation factorisée) Par exemple : une image est représentée par un vecteur contenant la valeur de chacun des pixels.

— La sortie désirée ou cible y aura une représentation différente selon le problème à résoudre. Si c'est un problème de régression, ce sera une valeur réelle ou continue ; si c'est un problème de classification en C classes, se sera une valeur discrète (index de 0 à C-1)

Un problème d'apprentissage supervisé peut être formulé de la façon décrite ci-dessous.

On donne un ensemble d'entraînement D de N exemples : $D = \{(X^{(1)}, y^{(1)}), (X^{(2)}, y^{(2)}), ..., (X^{(N)}, y^{(N)})\}$. Dans cet ensemble d'entraînement, chaque y_j a été généré par une fonction inconnue f, telle que $y = f(X)$. L'objectif est de trouver une nouvelle fonction h, appelée modèle (ou hypothèse) qui sera une bonne approximation de f ; c'est à dire : $f(X) \approx h(X)$.

En somme, un algorithme d'apprentissage peut donc être vu comme étant une fonction A à laquelle on donne un ensemble d'entraînement D et qui donne en retour cette fonction h :

$$A(D) = h$$

Les données d'entraînement pourront être indicées par la lettre "t" (pour training) : X_t, y_t

Nous verrons plus loin que le problème de l'apprentissage supervisé peut être formulé comme un problème d'optimisation. Dans lequel, pour chaque exemple d'entraînement, on souhaite minimiser une certaine distance J entre la cible y_t et la prédiction $h(X_t)$. Cette distance est appelée perte ou fonction coût : $J(y_t, h(X_t))$. Si la prédiction est bonne, ce coût tend vers 0.

II.3.2 Problèmes de régression

Littéralement en mathématiques, la régression est le fait d'approcher une variable (le prix d'un appartement) à partir d'autres qui lui sont liées (la superficie et le nombre de pièces).

Pour atteindre cet objectif, plusieurs modèles d'approches sont possibles. Par exemple, approcher les données par une droite (régression linéaire), par un polynôme (régression polynomiale), par une fonction logarithmique, etc... Par extension, on appelle régression en apprentissage supervisé tout problème qui consiste à prédire une (ou plusieurs) variable(s) quantitatives. Les domaines d'application sont nombreux, des finances (prédiction du cours de la Bourse...) au commerce (stocks futurs à prévoir...) en passant par la maintenance prédictive (anticiper une panne).

Dans le cas des problèmes de régression la cible est un nombre réel $y \in \mathbb{R}$.

Ci-dessous deux exemples permettant de prédire un chiffre d'affaire, avec deux modèles prédictifs (régression linéaire et polynomiale, qui est une régression linéaire multiple).

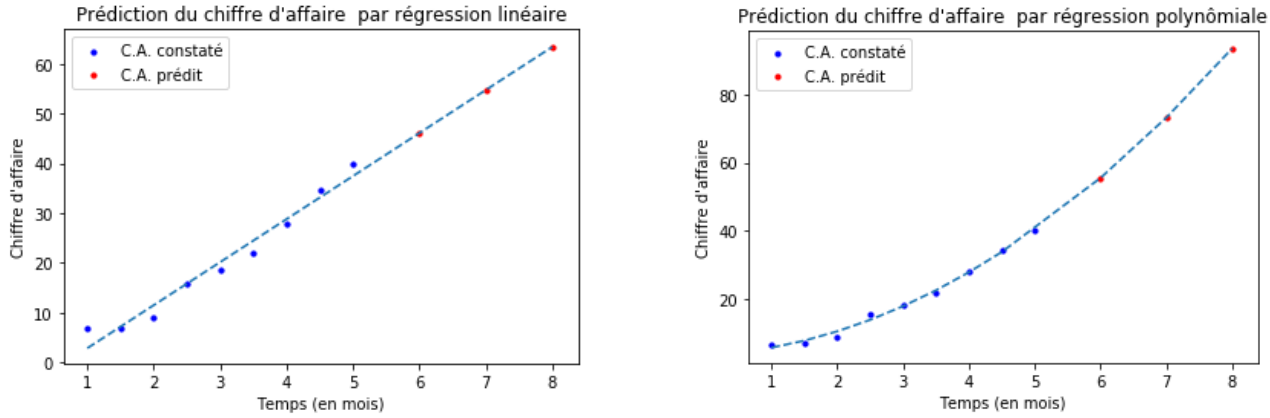

FIGURE 5 – Exemples d'estimation par régression linéaire et polynomiale

On pourra citer en guise d'exemples les algorithmes de régression suivants :

— Régression linéaire uni-variable ou multi-variables,

— Arbres de régression,

— Régression vectorielle de support,...

II.3.3 Problèmes de classification

La classification, comme le nom l'indique, repose sur des classes : ce sont les problèmes pour lesquels l'objectif est de prédire une classe parmi un ensemble fini. Quelques exemples : Déterminer si l'animal présent dans l'image est un chat, un chien ou un lapin, évaluer la gravité d'un cancer (bénin ou malin, l'équivalent en régression serait de prédire le risque de décès en pourcentage),...

La classification présentée ainsi ressemble à un problème lié à de l'apprentissage non-supervisé, mais ce n'est pas le cas. La différence est qu'en non-supervisé les classes ne sont pas connues et sont déterminées à la volée, tandis qu'ici la prédiction de l'apprentissage supervisé est une classe parmi un ensemble de classe définies.

On pourra citer en guise d'exemples les algorithmes de classification suivants :

— Régression logistique,

— Arbres de classification (dont Forêts aléatoires),

— K plus proches voisins (KNN),

— Machine à vecteurs de support

— Classificateur naïf de Bayes,

— Réseaux de neurones,...

II.4 Principes généraux de l'apprentissage non supervisé

Mathématiquement, l'apprentissage non supervisé est celui où vous n'avez que des données d'entrée (X) et aucune variable de sortie correspondante. L'objectif de l'apprentissage non supervisé est de modéliser la structure sous-jacente ou la distribution dans les données afin d'en savoir plus sur les données. Dans l'approche d'apprentissage non supervisé, l'échantillon d'un ensemble de données d'apprentissage n'a pas de sortie attendue qui lui est associée. En utilisant les algorithmes d'apprentissage non supervisé, vous pouvez détecter des modèles en fonction des caractéristiques typiques des données d'entrée. Le clustering peut être considéré comme un exemple de tâche d'apprentissage automatique qui utilise l'approche d'apprentissage non supervisé. La machine regroupe ensuite des échantillons de données similaires et identifie différents groupes dans les données.

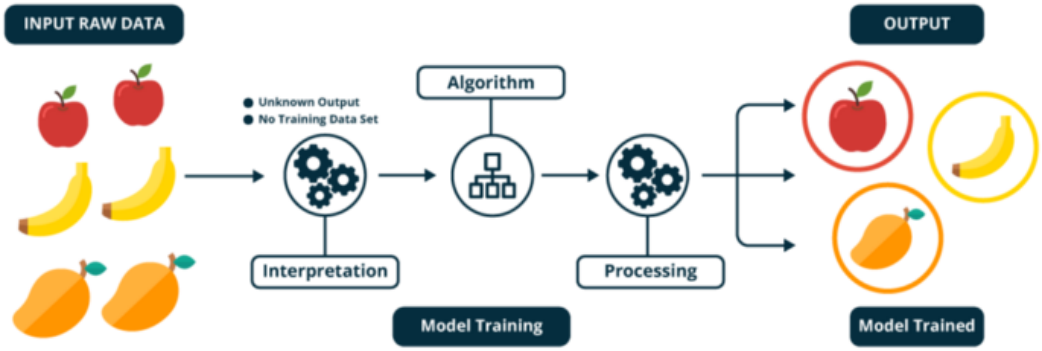

FIGURE 6 – Exemple d'apprentissage Automatique non supervisé

Regroupement ou Clustering :

La mise en cluster consiste à séparer ou à diviser un ensemble de données en un certain nombre de groupes, de sorte que les ensembles de données appartenant aux mêmes groupes se ressemblent davantage que ceux d'autres groupes. En termes simples, l'objectif est de séparer les groupes ayant des traits similaires et de les assigner en grappes.

Voyons cela avec un exemple. Supposons que vous soyez le chef d'un magasin de location et que vous souhaitiez comprendre les préférences de vos clients pour développer votre activité. Vous pouvez regrouper tous vos clients en 10 groupes en fonction de leurs habitudes d'achat et utiliser une stratégie distincte pour les clients de chacun de ces 10 groupes. Et c'est ce que nous appelons le Clustering.

Association :

L'association consiste à découvrir des relations intéressantes entre des variables dans de grandes bases de données. Par exemple, les personnes qui achètent une nouvelle maison ont aussi tendance à acheter de nouveaux meubles. Il découvre la probabilité de co-occurrence d'éléments dans une collection.

En résumé, le clustering consiste à grouper des points de données en fonction de leurs similitudes, tandis que l'association consiste à découvrir des relations entre les attributs de ces points de données.

Voici une liste de certains algorithmes d'apprentissage automatique non supervisés :

— K-Moyennes

— Réduction de la dimensionnalité
— Réseaux de neuronnes
— Analyse des composants principaux
— Décomposition en valeur singulière
— Analyse en composantes indépendantes
— Modèles de distribution
— Classification hiérarchique

III Exemple d'algorithmes simples utilisés en machine learning

III.1 Régression linéaire uni-variable

On rappelle que dans le cadre de l'apprentissage supervisé, dans un problème de régression, la cible est un nombre réel $y \in \mathbb{R}$ Exemple : prédiction de la valeur d'une action à la bourse x : vecteur contenant l'information sur l'activité économique de la journée y : valeur d'une action à la bourse le lendemain.

III.1.1 Modèle de régression linéaire

.

Le modèle de régression linéaire est le suivant :

$$h(X, W) = w_0 + w_1.x_1 + ... + w_D.x_D$$

où :

— $X = (x_1, ..., x_D)^T$ est le vecteur des entrées du modèle.
— $W = (w_0, w_1, ..., w_D)^T$ est le vecteur des paramètres du modèles (biais et poids)

La prédiction correspond donc à une droite dans l'espace en D dimensions. Dans ce modèle, nous pouvons considérer que w_0 est le biais du modèle car il ne dépend pas des composantes de l'entrée X. Tandis que les $w_1, ..., w_D$ sont les poids du modèle. Ces derniers pondèrent les changements dans les valeurs prises par $x_1, ..., x_D$.

Prenons l'exemple en une dimension. Dans ce cas le modèle s'écrira : $h(X, w) = w_0 + w_1.x_1$.

Pour simplifier, on notera : $x_1 = x$. L'objectif est donc à partir d'un ensemble d'entraînement, composé de N couples $(x_t^{(i)}, y_t^{(i)})$, de déterminer les deux paramètres du modèle w_0 et w_1 de la fonction h. Cet ensemble d'entraînement ou "Dataset" peut alors être vu pour une problème en une dimension comme un nuage de points dans le plan :

Si le modèle est correct, il devra donner de petites erreurs entre les sorties du modèle y et ses prédictions avec $h(x)$.

III.1.2 Fonction coût et sa minimisation par l'algorithme de descente de gradient

On peut définir l'erreur unitaire entre une valeur observée $y_t^{(i)}$ et une valeur prédite $h(x_t^{(i)})$ de la façon suivante :

$$\left(h(x_t^{(i)}) - y_t^{(i)}\right)^2$$

Trouver le meilleur couple (w_0, w_1) revient à minimiser le coût global des erreurs unitaires qui se définit comme suit :

$$\sum_{i=1}^{N} \left(h(x_t^{(i)}) - y_t^{(i)}\right)^2$$

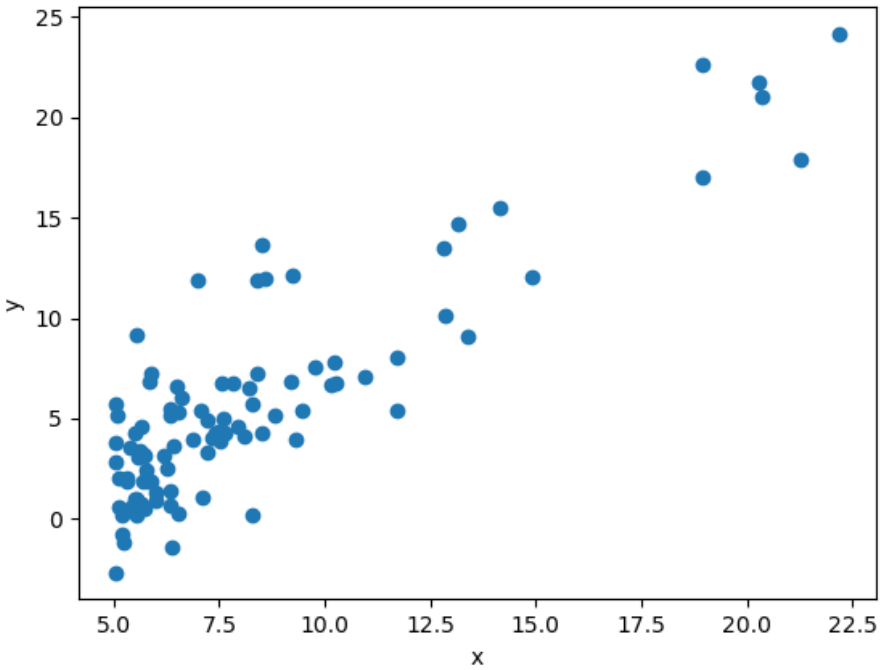

FIGURE 7 – Nuage de points pour exemple

La fonction coût est une erreur quadratique moyenne définie de la façon suivante :

$$J(W) = \frac{1}{2.N} \sum_{i=1}^{N} \left(h(x_t^{(i)}) - y_t^{(i)} \right)^2$$

Dans le cas d'une régression linéaire uni-variable, nous avons : $W = (w_0, w_1)$. Ainsi, on obtient :

$$J(w_0, w_1) = \frac{1}{2.N} \sum_{i=1}^{N} \left(w_0 + w_1.x_t^{(i)} - y_t^{(i)} \right)^2$$

Cette fonction J est convexe (Figure 8), ce qui implique qu'un seul minimum global existe. Par conséquent, trouver les valeurs w_0 et w_1 qui minimisent la fonction J c'est trouver les valeurs optimales pour la fonction modèle h.

Pour minimiser la fonction coût, on va utiliser l'algorithme de descente de gradient, qui se présente de manière simplifiée avec le pseudo-code suivant :

```
1 Debut algorithme Descente de gradient :
2     Initialiser aleatoirement les valeurs w0,w1
3     Tant que le minimum n est pas atteint :
4         Pour j={0,1}:
5             wj <- wj - pas.dJ/dwj
6         Fin Pour
7     Fin Tant que
8 Fin algorithme
```

La ligne 5 du pseudo-code précédent fait intervenir le pas α, que l'on nommera aussi taux d'apprentissage, ainsi qu'une dérivée partielle $\frac{\partial J}{\partial w_j}$. A chaque itération dans la boucle "Tant que", les paramètres

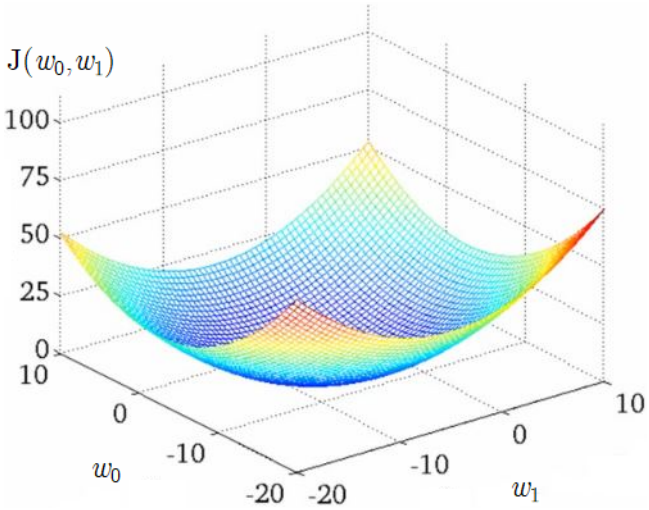

FIGURE 8 – Représentation graphique de la fonction J pour la régression linéaire uni-variable

du modèle sont mis à jour en leur soustrayant la plus grande pente possible multipliée par le taux d'avancement (ou pas) α. Le taux d'apprentissage α est en règle général variable pour optimiser au mieux et au plus vite la convergence vers le minimum fonction coût. Dans notre cas, on le conservera constant. Mais il est évident qu'il doit être bien choisi :

— S'il est trop petit, le modèle peut mettre longtemps à être entraîné (convergence lente) ;

— S'il est trop grand, on risque de ne jamais atteindre le minimum et ne pas converger...

Dans le cas de la régression linéaire uni-variable, les dérivées partielles de la fonction coût sont simplement calculables :

$$\frac{\partial J(w_0, w_1)}{\partial w_0} = \frac{1}{N} \sum_{i=1}^{N} \left(w_0 + w_1 . x_t^{(i)} - y_t^{(i)} \right)$$

$$\frac{\partial J(w_0, w_1)}{\partial w_1} = \frac{1}{N} \sum_{i=1}^{N} x_t^{(i)} \left(w_0 + w_1 . x_t^{(i)} - y_t^{(i)} \right)$$

```
1  def J(X,Y,w0,w1):
2      cout=0; N=len(Y)
3      for i in range(N):
4          cout += (w0+w1*X[i]-Y[i])**2
5      return (1/(2*N))*cout
```

```
1  def Gradient(X,Y,w0,w1):
2      N = len(X)
3      dJ_w0, dJ_w1 = 0,0
4      for i in range(N):
5          dJ_w0 += w0 + w1*X[i] - Y[i]
6          dJ_w1 += X[i] * (w0 + w1*X[i] - Y[i])
7      return [(1/N)*dJ_w0,(1/N)*dJ_w1]
```

```python
def Regression_Lineaire(X,Y,nbre_iter=10000,alpha=float(0.001)):
    w0 = 0.; w1 = 0.
    Histoire_cout = []
    for k in range(nbre_iter):
        w0 -= alpha * Gradient(X,Y,w0,w1)[0]
        w1 -= alpha * Gradient(X,Y,w0,w1)[1]
        Histoire_cout.append(J(X,Y,w0,w1))
    return w0,w1,Histoire_cout
```

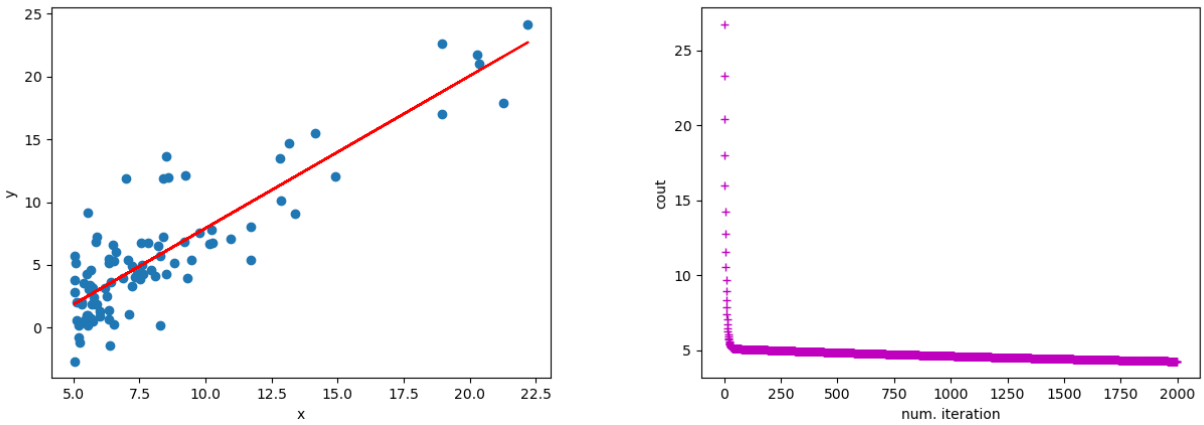

FIGURE 9 – Résultat de la régression linéaire et évolution de la fonction coût en fonction du numéro d'itération

III.2 Régression linéaire multiple et polynomiale (multi-variables)

Il est possible d'étendre la régression linéaire à un problème multi-variables (le modèle ne sera plus linéaire, mais polynomiale à plusieurs dimensions).

Explicitons dans un premier temps dans un premier temps ce que représentent chacun des termes qui seront utilisés par la suite. Comme nous sommes dans un problème à D dimensions, on aura :

— Le vecteur W des paramètres (avec le biais), de dimensions $(D+1,1)$: $W = [w_0, w_1, ..., w_D]^T$

— Le vecteur y_t des sorties de taille $(N,1)$, où N est la taille du jeu de données : $y_t = [y_t^{(1)}, y_t^{(2)}, ..., y_t^{(D)}]^T$

— La matrice X_t représente les entrées : $\begin{bmatrix} 1 & x_{t1}^{(1)} & x_{t2}^{(1)} & \dots & x_{tD}^{(1)} \\ 1 & x_{t1}^{(2)} & x_{t2}^{(2)} & \dots & x_{tD}^{(2)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{t1}^{(N)} & x_{t2}^{(N)} & \dots & x_{tD}^{(N)} \end{bmatrix}$, les termes $x_j^{(i)}$ avec $j > 1$ peuvent représenter une autre dimension, ou un des termes de degrés strictement supérieurs à 1, dans le cas d'une régression polynomiale. On remarque que la première colonne de 1 est là pour pouvoir associer le biais lors de l'apprentissage.

Il va être possible de définir les fonctions suivantes :

— Fonction modèle (hypothèse) : $h(X, W) = X.W$

— Fonction cout : $J(X_t, y_t, W) = \frac{1}{2.N} \sum_{i=1}^{N} (h(X_t, W) - y_t)^2$

— Descente de Gradient : $W = W - \alpha.\dfrac{\partial J}{\partial W}$

Pour travailler sur des problèmes à plusieurs dimensions, il est, nous venons de voir, nécessaire d'utiliser le calcul matriciel. Sous python, on pourra dont travailler avec des tableaux du module `numpy`. Dans ce cas, nous aurons besoin des fonctions du numpy telles que `np.dot`, `np.array`, `np.shape`, etc.

Prenons un exemple : on souhaite modéliser par une fonction polynomiale de degré 2 (deuxième exemple de la Figure 5). Dans ce cas, on a : $W = (w_0, w_1, w_2)^T$, et $X_t = \begin{bmatrix} 1 & x_{t1}^{(1)} & x_{t2}^{(1)} \\ 1 & x_{t1}^{(2)} & x_{t2}^{(2)} \\ \vdots & \vdots & \vdots \\ 1 & x_{t1}^{(N)} & x_{t2}^{(N)} \end{bmatrix}$ En notant : $x_t^{(i)} = (1, x_{t1}^{(i)}, x_{t2}^{(i)})$ Pour chaque ligne i on doit donc trouver la fonction h, telle que : $h(x_t^{(i)}) = w_0 + w_1.x_{t1}^{(i)} + w_2.x_{t2}^{(i)}$, où le terme $x_{t2}^{(i)} = {x_{t1}^{(i)}}^2$. La première colonne de la matrice X_t pourra être obtenue avec la fonction `np.ones((N,1))`, après avoir extrait le nombre N, tandis que la troisième colonne pourra être obtenue en faisant tout simplement `X**2` (avantage des tableaux Numpy...).

L'assemblage complet de la matrice X_t peut se faire avec les fonctions `np.concatenate` ou `np.hastack`. Ainsi on aurait :

```
Xt = np.concatenate((np.ones((N,1)),X,X**2),axis=1)
```

ou :

```
Xt = np.hstack((np.ones((N,1)),X,X**2))
```

Nous pouvons à présent combiner l'ensemble des résultats, pour pouvoir définir les trois fonctions précédentes (`J`, `Gradient` et `Regression_Lineaire`). On ajoutera pour simplifier les notations la fonction modèle (qui dans notre cas est une modélisation linéaire) :

```
def h(X,W):
    return X.dot(W)
```

```
def J(X,Y,W):
    N=np.shape(X)[0]
    cout = sum((h(X,W)-Y)**2)
    return (1/(2*N))*cout
```

```
def Gradient(X,Y,W):
    N = np.shape(X)[0]
    return 1/N * X.T.dot(h(X,W)-Y)
```

```
def Regression_Lineaire(X,Y,alpha,nbre_iter=10000):
    N,D = np.shape(X)
    W = np.random.randn(D,1)
    Histoire_cout = [J(X,Y,W)]
    k=0
    while k<nbre_iter :
        W = W - alpha*Gradient(X,Y,W)
        Histoire_cout.append(J(X,Y,W))
        k+=1
    return W,Histoire_cout
```

Remarque 1 : En lieu et place des lignes `np.shape(X)[0]`, il aurait été possible de saisir `X.shape[0]`...

Remarque 2 : `np.random.randn(D,1)` permet de générer un tableau 1 dimension de D nombres d'une distribution gaussienne standard (moyenne 0, écart-type 1)

Une autre possibilité d'arrêt est d'observer lorsque la différence de coût entre deux itérations successives tend vers une valeur proche de 0 (le modèle ne pourra être meilleur, on atteint le minimum de la fonction coût...). Dans ce cas on définirait la fonction ainsi :

```
def Regression_Lineaire(X,Y,nbre_iter=10000,alpha,ecart_erreur):
```

La boucle `while` serait alors réécrite, en ajoutant une variable c :

```
    c = np.inf
    while k<nbre_iter and abs(c-Histoire_cout[-1])>ecart_erreur:
        c = Histoire_cout[-1]
```

III.3 Régression logistique et problème de classification binaire

L'algorithme de régression logistique, comme son nom ne le laisse pas suggérer est à classer dans la catégorie des algorithmes de classification. On souhaite par exemple déterminer si un courriel est un spam. Parmi les critères quantitatifs qui peuvent être utiliser, on peux par exemple compter le nombre de fautes d'orthographe. Si ce nombre dépasse 15, alors le courriel en considérer qu'il s'agit d'un spam, sinon, il n'en est pas un.

Dans ce genre de problème, la variable cible y_t prendra uniquement 2 valeurs, soit par exemple 0 ou 1. (0 pour non spam et 1 si spam pour notre exemple). Dans un tel problème de classification binaire, on dira que l'on a deux classes.

On ajoute alors une frontière de décision qui permet de classer si le mail est un spam ou non. La fonction modèle h permettant de faire créer cette frontière sera telle que : $\begin{cases} h(x) = 0, \, si \, x <= 0,5 \\ h(x) = 1, \, si \, x <= 0,5 \end{cases}$.

Il est évident que pour un problème de classification binaire, un modèle de type linéaire vu dans le paragraphe précédent ne peut pas convenir. La Figure 10 ci-dessous illustre ces propos.

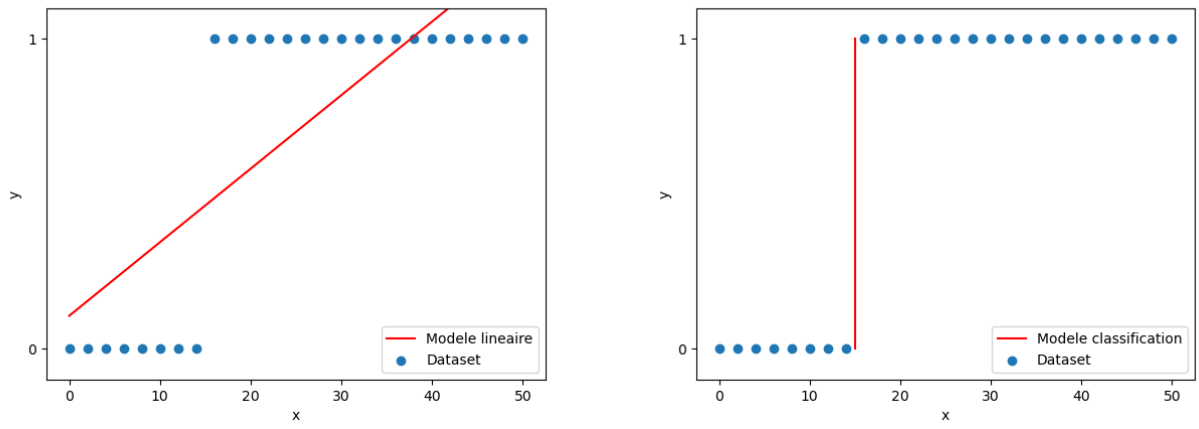

FIGURE 10 – Exemple de classification à une variable (spam)

III.3.1 Fonction logistique

On développe alors une nouvelle fonction pour ce genre de problèmes. Il s'agit de la fonction logistique, aussi appelée fonction sigmoïdale ou sigmoïde S :

$\sigma(z) = \dfrac{1}{1 + \exp -z}$

La Figure 11 donne un aperçu de la courbe représentative de cette fonction. On remarque que cette fonction est toujours comprise entre 0 et 1

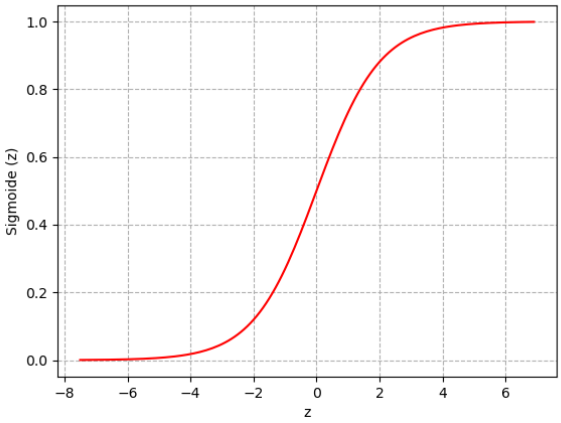

FIGURE 11 – Représentation graphique de la fonction Sigmoïde

Il existe d'autre fonctions de ce type, comme par exemple, la fonction Heaviside ou la fonction tangente hyperbolique, qui ont elles aussi la particularité d'être bornées. Dans la suite, on se focalisera sur la fonction logistique sigmoïde. A partir de cette fonction logistique, il est possible de définir une frontière ou seuil de décision, qui en règle générale est égal à 0,5.

Dans ce cas : $\begin{cases} y = 0, \, si \, \sigma(z) <= 0,5 \\ y = 1, \, si \, \sigma(z) = 0,5 \end{cases}$.

A noter que z est obtenu en faisant le produit $X_t.W$, X_t représente la même matrice que dans le cas de la régression linéaire et W le même vecteur des poids. Ainsi, pour l'exemple de classification à une variable, donnée en Figure 10, nous aurions une matrice X_t réduite à 2 colonnes (une colonne de 1 et une colonne prenant les valeurs des abscisses x des points du dataset. le vecteur W comporterait 2 valeurs : $(w_0, w_1)^T$.

III.3.2 Fonction coût associée à la régression logistique

On rappelle que la fonction coût associée à la régression linéaire J, s'écrivait : $J(X_t, y_t, W) = \dfrac{1}{2.N} \sum_{i=1}^{N} (h(X_t, W) - y_t)^2$. Celle-ci donnait une fonction (convexe) admettait un unique minimum, qui faisait que l'algorithme de descente de gradient fonctionnait. Cependant, pour le modèle logistique, cette fonction admet plusieurs minima (dû à la non-linéarité du modèle). Ceci implique l'aldorithme de descente de gradient risque de bloquer dans un minimum local qui n'est pas le minimum global...

Il s'avère donc nécessaire de développer une fonction coût compatible avec le modèle de régression logistique.

Cette fonction s'appuie sur la fonction logarithme et on devra expliciter celle-ci en séparant les cas où $y = 0$ et les cas où $y = 1$. En effet, dans le cas d'une classification binaire, seules ces deux valeurs sont possibles.

Expression de la fonction coût dans le cas où $y = 1$:

L'objectif est de pénaliser la machine si elle prédit une valeur proche de 0, alors que la sortie vaut 1. Dans ce cas, le coût associé J sera grand, et il faudra chercher de nouveaux paramètres dans W qui vont faire baisser le coût et ce successivement, jusqu à se rapprocher d'un coût nul.

La fonction coût peut donc dans ce cas s'écrire : $J(X_t, 1, W) = -log(\sigma(X_t.W))$

Expression de la fonction coût dans le cas où $y = 0$:

A l'inverse, cette fois-ci il faudra pénaliser la machine par une grande erreur si celle-ci prédit 1, alors que $y = 0$.

La fonction coût peut donc dans ce cas s'écrire : $J(X_t, 0, W) = -log(1 - \sigma(X_t.W))$

La Figure 12 illustre l'allure du coût J pour les deux possibilités $y = 1$ et $y = 0$.

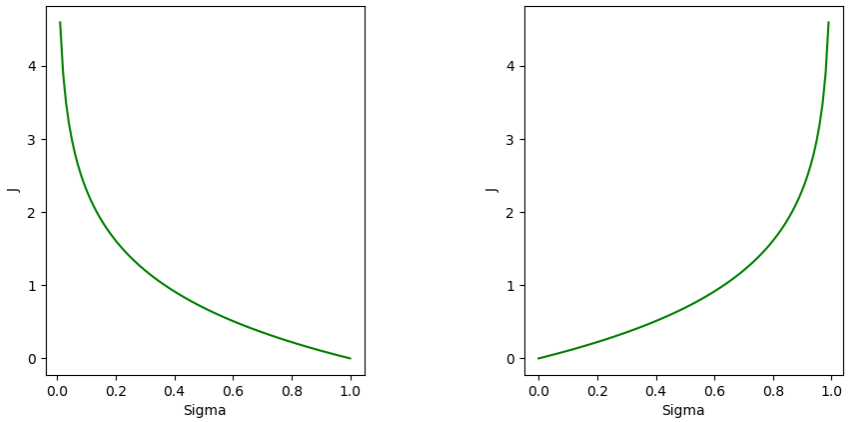

FIGURE 12 – Fonction coût pour les cas $y = 1$ (à gauche) et $y = 0$ (à droite)

Fonction coût complète :

La fonction coût peut s'écrire, en combinant les deux expressions précédentes, et en la moyennant par rapport au nombre N d'échantillons ou exemples du dataset :

$$J(X_t, y_t, W) = \frac{-1}{N}.\sum_{i=1}^{N}(y_t.\log(\sigma(X_t.W)) + (1 - y_t).\log(1 - \sigma(X_t.W)))$$

III.3.3 Algorithme de descente de Gradient pour la régression logistique

Cet algorithme se présente de la même manière que pour la régression linéaire.

La dérivée de la fonction coût s'écrit :

$$\frac{\partial J}{\partial W} = \frac{1}{N}\sum_{i=1}^{N}((\sigma(X_t.W) - y_t).X)$$

Résultat qui se démontre en calculant rigoureusement la dérivée partielle de J par rapport à la variable W.

A chaque itération, voilà ce que donne l'algorithme de descente de Gradient : $W = W - \alpha.\frac{\partial J}{\partial W}$.

α est toujours le taux d'apprentissage.

III.3.4 Résumé de la méthode de régression logistique et code python correspondant

— Modèle : $\sigma(X_t.W) = \dfrac{1}{1+e^{-X_t.W}}$

— Fonction coût : $J(X_t, y_t, W) = \dfrac{-1}{N}.\sum_{i=1}^{N}(y_t.\log(\sigma(X_t.W)) + (1-y_t).\log(1-\sigma(X_t.W)))$

— Gradient : $\dfrac{1}{N}.X^T.(\sigma(X_t.W) - y_t)$

— Descente de gradient : $W = W - \alpha.\dfrac{\partial J}{\partial W}$

Voici pour conclure les différentes fonctions écrites dans le langage Python :

```
#Fonction Sigmoide
def Sigmoide(z):
     return 1 / (1 + np.exp(-z))
def h(X,W):
    return Sigmoide(X.dot(W))
```

```
def J(X,Y,W):
    N=np.shape(X)[0]
    S=h(X,W)
    cout = sum(Y*(np.log(S)) + (1-Y)*(np.log(1-S)))
    return (-1/N)*cout
```

```
def Gradient(X,Y,W):
    N = np.shape(X)[0]
    return 1/N * X.T.dot(h(X,W)-Y)
```

```
def Regression_Logistique(X,Y,alpha,nbre_iter=10000):
    N,D = np.shape(X)
    W = np.zeros((D,1))
    Histoire_cout = [J(X,Y,W)]
    k=0
    while k<nbre_iter :
        W = W - alpha*Gradient(X,Y,W)
        Histoire_cout.append(J(X,Y,W))
        k+=1
    return W,Histoire_cout
```