Machine Learning
Techniques de base d’apprentissage supervisé

Introduction

- En Machine Learning, le **modèle** vise à **modéliser et estimer la relation entre les variables d'entrée (features)** et une **variable de sortie (target)**.
- On distingue deux grandes catégories de techniques selon la manière dont la relation est modélisée :
    - **Techniques paramétriques** : le modèle est défini par un ensemble fixe de **paramètres** à estimer. Exemples :
        - Régression linéaire,
        - Régression logistique,
        - Réseaux de neurones
    - **Techniques non-paramétriques** : le modèle **ne fait pas d'hypothèse forte** sur la forme de la relation entre les variables. Exemples :
        - Arbres de décision,
        - k-plus proches voisins (KNN)
- Dans ce chapitre, nous allons nous concentrer sur **deux techniques paramétriques fondamentales** :
    - La **régression linéaire**
    - La **régression logistique**

Partie 1 : Régression Linéaire

Présentation

Contexte

- **Tâche** : Prédiction de **valeurs continues**
- Exemple : prédire le prix d'une maison à partir de sa superficie.

Modèle

- Forme du modèle :

$$y = f_{w,b}(x) = w * x + b$$

$$y = x_1 * w_1 + \ ... + x_m * w_m + b$$

Tels que :

- **y** est un scalaire et elle est appelée variable cible
- **x** est en général un vecteur composé des valeurs x_1, ..., x_1 appelées variables prédictives
- **b** est l'ordonnée à l'origine (ou l'intercept)
- **w** est un vecteur composé des valeurs w_1 , ... , w_m et elles sont appelées coefficients ou poids

Chaque coefficient/poids correspond à une variable prédictive.

Estimation des paramètres

- Pour estimer les paramètres, il y a deux approches :
  - 1. Résolution analytique
    - Utilise la méthode des **moindres carrés ordinaires (OLS)** :
    - Efficace pour les petits jeux de données
    - Implémentation pratique : **LinearRegression** de Scikit-learn
  - 2. Résolution algorithmique
    - Utilise **la descente de gradient** (batch, stochastique ou mini-batch)
    - Efficace pour les grands jeux de données
    - Implémentation pratique : **SGDRegressor** de Scikit-learn

Résolution algorithmique

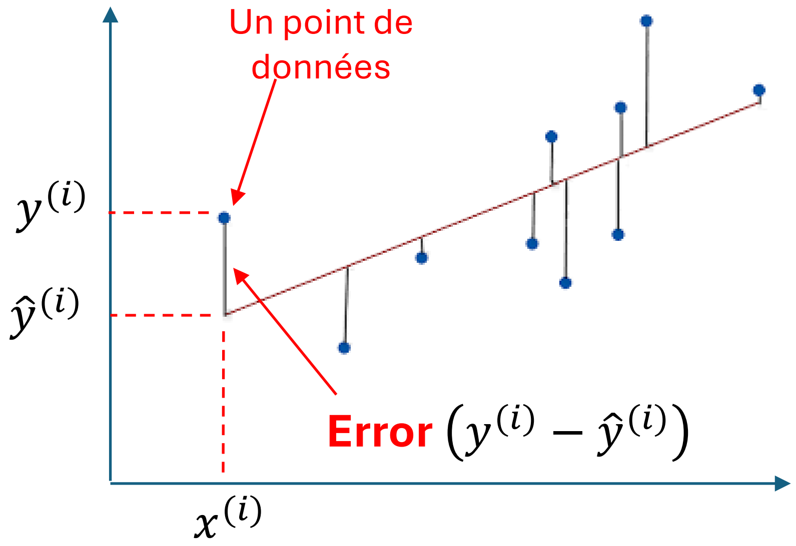
Fonction de coût

- C’est l’erreur quadratique moyenne (Mean Squared Error, MSE) :

$$J(w,b) = \frac{1}{2n}\sum_i \left(y^{(i)} - \hat{y}^{(i)}\right)^2$$

$$J(w,b) = \overset{\text{Mean}}{\boxed{\frac{1}{2n}\sum_i}} \left(\overset{\text{Error}}{\boxed{y^{(i)} - \hat{y}^{(i)}}}\right)^{\overset{\text{Squared}}{\boxed{2}}}$$

$$J(w,b) = \frac{1}{2n}\sum_i \left(y^{(i)} - (x^{(i)} * w + b)\right)^2$$



Estimation des paramètres

Mise à jour des paramètres

- Mise à jour des poids :

$$\partial w_j = \frac{1}{n}\sum_i \left(\hat{y}^{(i)} - y^{(i)}\right) x_j^{(i)}$$

- Mise à jour du biais :

$$\partial b = \frac{1}{n}\sum_i \left(\hat{y}^{(i)} - y^{(i)}\right)$$

Hyperparamètres

- Taux d'apprentissage α
- Nombre d'itérations
- Taille du mini-batch

Activité 2

Partie 1 : Regression linéaire univariée

1. Créer les données d'entrée X (les surfaces des maisons) et de sortie y (prix de maisons)
    - X est un tableau bidimensionnel dont :
        - chaque ligne est un individu (une maison)
        - Chaque colonne est un feautre (la surface)
    - Y est un tableau unidimensionnel
2. Créer et configurer le modèle de la régression linéaire avec Descente de Gradient Stochastique (SGD)

   -> Utiliser soit la classe SGDRegressor de sklearn.linear_model

   -> Spécifier les arguments suivants :
    - learning_rate='constant'
    - eta0=0.001
    - max_iter=10000
    - tol=0.001
    - loss='squared_error'
    - penalty=None
    - verbose=1
3. Entrainer le modèle en utilisant la function fit()
4. Vérifier les valeurs optimales des paramètres :
    - coefficients (w)
    - intercept (b)
5. En utilisant le modèle entrainé, appeler la fonction predict() pour prédire la sortie d'un nouvel individu

Activité 2

Partie 2 : Regression linéaire multivariée

Scénario 1 :

On va enrichir les données d'entrée X avec une autre feature, par exemple le nombre de chambres.

- ➔ PB : Les features surface et nb_chambre ont éventuellement des échelles différentes.
- ➔ Solution : Normaliser les données X avant l'apprentissage avec StandardScaler

Scénario 2 :

On va enrichir les données d'entrée X avec une autre feature, par exemple Type d'une maison qui peut être soit Normal soit Haut Standing.

- ➔ PB : Feature type a des valeurs discrètes
- ➔ Solution : Encoder ce feature en utilisant OneHotEncoder

Scénario 3 :

-> PB : Les features peuvent être nombreuses et trop corrélées

-> Solution : Appliquer l'analyse en composantes principales (ACP) pour réduire le nombre des features et les décorréler.

Partie 2 : Régression Logistique

Présentation

Contexte

- **Tâche** : Prédiction de **valeurs discrètes**
  - Classification **binaire (deux classes : 0 et 1)**
  - Classification **multiclasses** (plus que 2 classes)
- Exemple : prédire la catégorie de prix d'une maison (faible, moyen, elevé) à partir de sa superficie.

Modèle

- Forme du modèle :

$$y = P(y = 1|x) = f_{w,b}(x) = \frac{1}{1 + exp^{-(x*w+b)}}$$

- On peut voir le modèle comme suit :

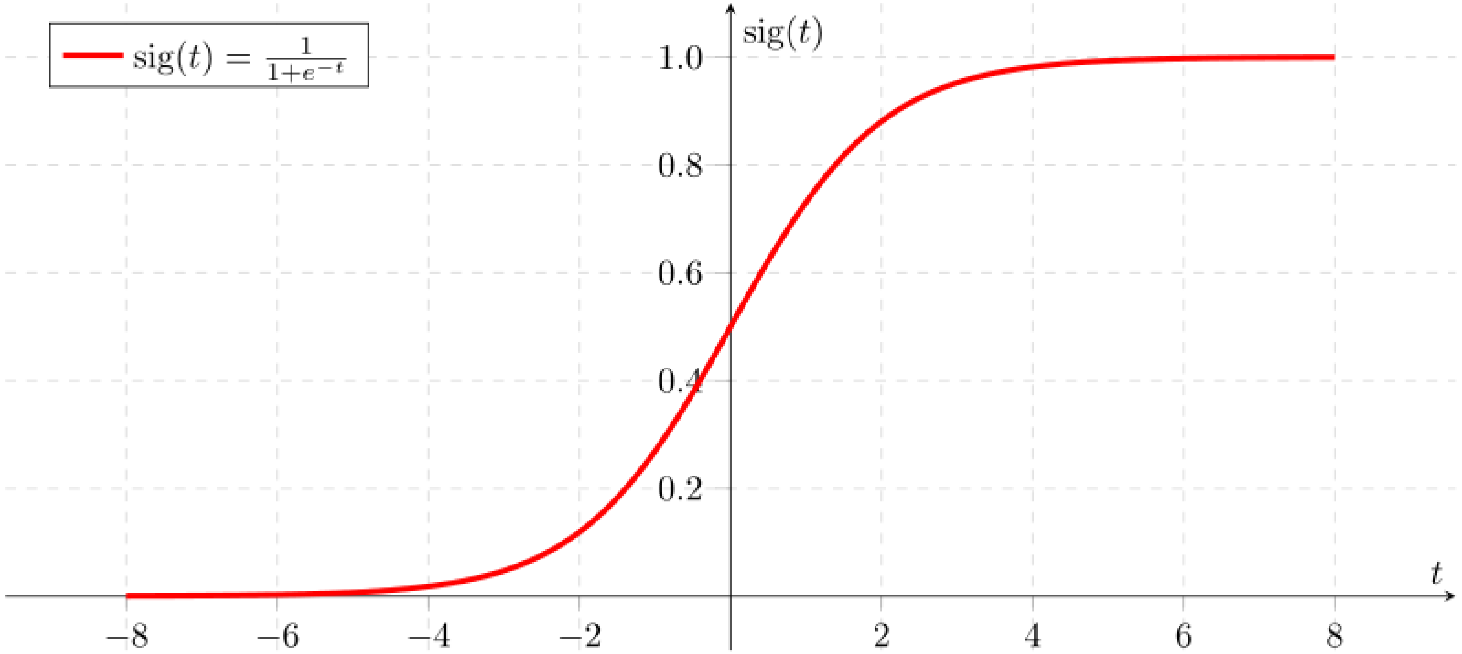
$$y' = x * w + b = x_1 * w_1 + \ \ldots + x_m * w_m$$
$$\text{y} = \boldsymbol{sig}(y')$$

Tels que :

- **y** est un scalaire et elle est appelée variable cible
- **x** est en général un vecteur composé des valeurs x_1, ..., x_1 appelées variables prédictives
- **b** est l'ordonnée à l'origine (ou l'intercept)
- **w** est un vecteur composé des valeurs w_1 , ... , w_m et elles sont appelées coefficients ou poids
- **sig( )** est une fonction d'activation (elle sera expliquée par la suite)

Fonction logistique

- La fonction d'activation utilisée dans la régression logistique ***sig( )*** est la fonction sigmoïde (ou fonction logistique) qui transforme la sortie pondérée en une probabilité normalisée entre 0 et 1.



- **Interprétation:**
  - La sortie $y' = x * w + b$ n'est pas normalisée, càd dire elle n'est pas entre 0 et 1.
  - On utilise la **sigmoïde** pour produire une **probabilité** d'appartenance à la classe positive.

Estimation des paramètres

- Pour estimer les paramètres, il y a deux approches :
  - 1. Résolution algorithmique 1
    - Utilise la méthode d'optimisation quasi-Newtonienne (notée solver='lbfgs')
    - Efficace pour les petits jeux de données
    - Implémentation pratique : **LogisticRegression** de Scikit-learn
  - 2. Résolution algorithmique 2
    - Utilise **la descente de gradient** (batch, stochastique ou mini-batch)
    - Efficace pour les grands jeux de données
    - Implémentation pratique : **SGDClassifier** de Scikit-learn

Résolution algorithmique 2

Fonction de coût

- C'est l'entropie croisée négative (cross-entropy loss) appelée encore Log de la vraisemblance ou erreur logistique ou log loss
- Elle mesure l'erreur entre les prédictions du modèle et les valeurs réelles de y.

- La fonction de coût est définie comme suit :

$$J(w,b) = -\frac{1}{m}\sum_i y^{(i)} * \log(\hat{y}^{(i)}) + (1-y^{(i)}) * \log\left(1-\hat{y}^{(i)}\right)$$

Tels que :

- $y^{(i)}$ est la classe réelle de l'entrée $x^{(i)}$
- $\hat{y}^{(i)}$ est la sortie prédite par le modèle, càd la probabilité d'appartenance à la classe positive

Résolution algorithmique 2

Fonction de coût

Il s'agit de l'entropie négative

Normaliser l'entropie

=1 si la classe est positive
= 0 si la classe est négative

$$J(w,b) = -\boxed{\frac{1}{m}} \sum_i \boxed{e^{(i)}} = \boxed{\boxed{y^{(i)}} * \log(\hat{y}^{(i)})} + \boxed{\boxed{(1 - y^{(i)})} * \log(1 - \hat{y}^{(i)})}$$

=0 si la classe est positive
= 1 si la classe est négative

exclusives

L'entropie croisée négative (cross-entropy loss)

Estimation des paramètres

Mise à jour des paramètres

- Mise à jour des poids :

$$\partial w_j = \frac{1}{n}\sum_i \left(\hat{y}^{(i)} - y^{(i)}\right) x_j^{(i)}$$

- Mise à jour du biais :

$$\partial b = \frac{1}{n}\sum_i \left(\hat{y}^{(i)} - y^{(i)}\right)$$

Hyperparamètres

- Taux d’apprentissage α
- Nombre d’itérations
- Taille du mini-batch

Régression logistique pour la classification multi-classes

Modèle

- Prédire la **probabilité** que x appartienne à l'une des K classes :

$$y = P(y = k|x) \quad , k = 0, 1, 2, .., K\text{-}1$$

- On peut voir le modèle comme suit :

$$y^{j'} = x * w_j + b = x_1 * w_j^1 + \ \dots + x_m * w_j^m$$

$$y = \boldsymbol{softmax}(y')$$

- La fonction **softmax** généralise l'utilisation de la fonction sigmoïde pour K classes :
  - Elle doit garantir que la somme des probabilités de K classes = 1
  - La formule est plus claire pour chaque classe :

$$\hat{y}_k = P(y = k \mid x) = \frac{e^{z_k}}{\sum_{j=1}^{K} e^{z_j}} = \frac{e^{w_k^\top x + b_k}}{\sum_{j=1}^{K} e^{w_j^\top x + b_j}}$$

Tels que :

y est la probabilité d'appartenance à la classe k

$\boldsymbol{W} = \begin{pmatrix} \boldsymbol{w_1} \\ \vdots \\ \boldsymbol{w_K} \end{pmatrix}$ est une matrice poids composée des vecteurs $w_1, \dots, w_K$.

Chaque vecteur $\boldsymbol{w_k}$ contient des valeurs poids appelées coefficients ou poids

$\boldsymbol{b} = (\boldsymbol{b_1}, \boldsymbol{b_2}, \dots, \boldsymbol{b_K})$ est un vecteur de dimension K appelé biais (ou l'intercept)

softmax( ) est une fonction d'activation (elle sera expliquée par la suite)

Activité 2

Partie 3 : Régression logistique

1. Reprendre le dataset de activité 2.
   - Garder les mêmes données d'entrée X (les surfaces des maisons), nombre de chambres et type de maison
   - X est un tableau bidimensionnel dont :
     - chaque ligne est un individu (une maison)
     - Chaque colonne est un feautre (la surface, nb_chambre, type)
   - Les données de sortie y (classe qui est la catégorie de prix de maisons : prix_faible, prix_moyen et prix_eleve)
     - Y est un tableau unidimensionnel

2. Créer et configurer le modèle de la régression logistique avec Descente de Gradient Stochastique (SGD)

   -> Utiliser soit la classe SGDClassifier de sklearn.linear_model

   -> Spécifier les arguments suivants :
   - learning_rate='constant'
   - eta0=0.001
   - max_iter=10000
   - tol=0.001
   - loss='log'
   - penalty=None
   - verbose=1

3. Entrainer le modèle en utilisant la function fit()

4. Vérifier les valeurs optimales des paramètres :
   - coefficients (w)
   - intercept (b)

5. En utilisant le modèle entrainé, appeler la fonction predict() pour prédire la sortie d'un nouvel individu