الجمهورية الجزائرية الديمقراطية الشعبية

وزارة التعليم العالي والبحث العلمي

People's Democratic Republic of Algeria

Ministry of Higher Education and Scientific Research

University of Algiers 1 Benyoucef Benkhedda

Faculté des Sciences

Département Informatique

Spécialité : Master 2 Analyse et Sciences de Données

Module : Analyse et Exploitation des Données

Rapport de Projet

Régression Logistique Binaire

Présenté par :

Guechtouli Anis

Hammamid Ahmed Issam

2024 / 2025

Table des matières

1. Introduction 3
2. Partie Théorique : 4
2.1 - Définition 4
2.2 – La fonction d'activation (Sigmoïde) 4
2.3 - La classification en utilisant la Régression Logistique Binaire 6
2.4 - Entraînement du Modèle 7
2.4.1 - La fonction de perte d'entropie croisée binaire 7
2.4.2 - Descente de gradient 8
2.5 – Avantages et limites de la Régression Logistique Binaire 10
2.6 - Domaines d'applications de Régression Logistique Binaire 12
3. Partie Pratique : 13
3.1 – Base de données 13
3.2 – Régression Logistique Binaire dans le domaine médical 13
3.2.1 Importation des bibliothèques nécessaires 13
3.2.2 Lecture de dataset 13
3.2.3 Prétraitement des données 14
3.2.4 Entrainement du Modèle 16
3.2.5 Evaluation du Modèle 16
4. Conclusion 21
5. Références 22

1. Introduction

La régression est une méthode fondamentale en analyse prédictive, principalement utilisée pour modéliser les relations entre une variable dépendante continue et une ou plusieurs variables indépendantes. Elle permet de prédire une valeur numérique en se basant sur des données d'entrée, en établissant une relation mathématique entre les variables. Cette approche est largement utilisée dans des domaines comme l'économie, la finance, et les sciences sociales, où la compréhension des liens entre différentes variables est essentielle pour faire des prévisions précises.

Cependant, la régression peut aussi être étendue à des tâches de classification, en particulier dans le cas de la régression logistique. Bien qu'initialement conçue pour prédire des variables continues, la régression logistique permet de traiter des problèmes de classification binaire, où l'objectif est de prédire l'appartenance d'une observation à l'une des deux classes possibles. En remplaçant la fonction linéaire classique par la fonction sigmoïde, qui transforme la sortie en une probabilité, la régression logistique devient un outil puissant pour modéliser des décisions dichotomiques. Ainsi, bien que la régression soit principalement associée à des variables continues, son extension à la classification ouvre de nombreuses possibilités dans des domaines aussi variés que la médecine, le marketing, la finance, et bien d'autres, où il est nécessaire de prédire des événements binaires comme la réussite/échec, l'achat/non achat, ou là présence/absence d'une maladie.

2. Partie Théorique :

2.1 - Définition

La régression logistique binaire est une méthode statistique utilisée pour modéliser la probabilité qu'un événement binaire (ayant deux issues possibles, comme "succès/échec" ou "oui/non") se produise, en fonction d'un ensemble de variables prédictives, également appelées caractéristiques. Utilisant une fonction d'activation appelée sigmoïde, cette méthode est essentielle dans les sciences sociales et naturelles et constitue un outil de base pour les algorithmes supervisés de classification.

Particulièrement adaptée pour identifier les relations entre les caractéristiques observées et une issue particulière, la régression logistique permet de classer les observations en fonction de leur probabilité d'appartenir à l'une des deux classes. Elle est également étroitement liée aux réseaux de neurones, car la régression logistique peut être considérée comme un réseau de neurones à une seule couche, où la fonction d'activation sigmoïde joue le rôle d'activation., ce qui en fait une technique fondamentale dans le domaine de l'apprentissage automatique.

2.2 – La fonction d'activation (Sigmoïde)

Dans la régression logistique binaire, l'objectif est de créer un classificateur capable de prédire la probabilité qu'une observation appartienne à une classe particulière, souvent représentée par 1 (membre de la classe) ou 0 (non membre de la classe). Pour ce faire, la régression logistique utilise la fonction d'activation sigmoïde.

Considérons une observation d'entrée x, représentée par un vecteur de caractéristiques $[x_1, x_2, \dots, x_n]$. Le classificateur utilise ces caractéristiques pour prédire la probabilité que l'observation appartienne à la classe 1. Cette probabilité, notée $P\ (y = 1|x)$, est calculée en passant un calcul linéaire des caractéristiques à travers la fonction sigmoïde.

Avant d'appliquer la fonction sigmoïde, on calcule une somme pondérée des caractéristiques, incluant un terme de biais. Chaque caractéristique x_i est multipliée par un poids w_i, et ces produits sont additionnés. Ce calcul donne une valeur z qui représente l'«évidence » pour la classe, c'est-à-dire la somme pondérée des caractéristiques, augmentée du biais b :

$$z = \left(\sum_{i=1}^{n} w_i x_i \right) + b$$

Cette équation est similaire à celle de la régression linéaire, où l'objectif est de prédire une valeur continue. Cependant, cette somme n'est pas une probabilité en soi. En effet, elle peut être un nombre quelconque, allant de -∞ à ∞. Pour obtenir une probabilité, nous appliquons la fonction sigmoïde à cette somme z, ce qui permet de contraindre le résultat dans l'intervalle $[0, 1]$, interprétable comme une probabilité.

La fonction sigmoïde est définie par l'équation suivante :

$$\sigma(z) = \frac{1}{1 + e^{-z}} = \frac{1}{1 + \exp(-z)}$$

Le graphe associé à la fonction sigmoïde est le suivant :

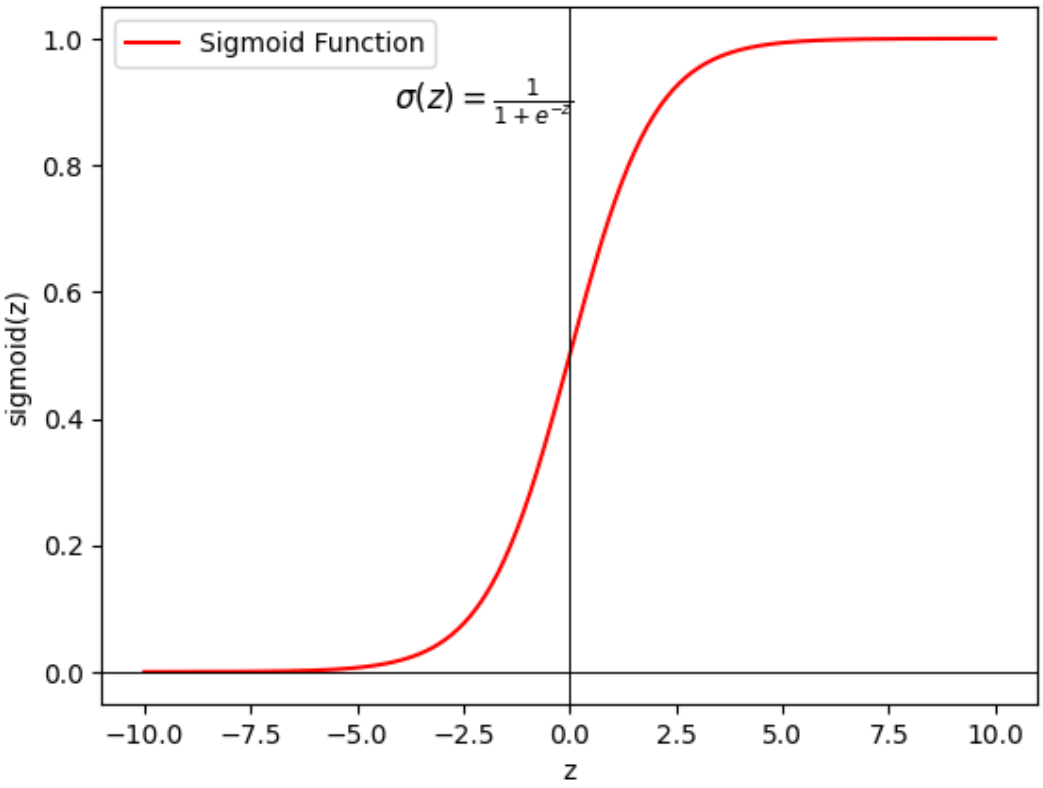

Figure 01 : Représentation graphique de la fonction sigmoïde

La fonction transforme ainsi la sortie z en une valeur comprise entre 0 et 1. Elle est souvent qualifiée de fonction d'activation car elle "active" ou "désactive" une décision en fonction du résultat de la somme pondérée. En d'autres termes, elle permet de "caler" ou de "décider" du passage de la sortie du classificateur à une probabilité entre 0 et 1, ce qui est crucial dans la classification binaire. Elle sert à déclencher l'activation de la classe en fonction de l'entrée, d'où le terme "activation".

Enfin, un point terminologique important. L'entrée de la fonction sigmoïde, $z = w \cdot x + b$, est souvent appelée le logit. Cela s'explique par le fait que la fonction logit est l'inverse de la sigmoïde.

L'utilisation du terme "logit" pour désigner z permet de rappeler que, en utilisant la fonction sigmoïde pour transformer z (qui varie de $-\infty$ à ∞) en une probabilité, on interprète implicitement z non seulement comme un nombre réel, mais spécifiquement comme un logarithme des cotes aussi appelé en anglais “log odds” qui est une transformation mathématique utilisée pour exprimer la probabilité d'un événement dans le cadre de modèles de régression logistique.

2.3 - La classification en utilisant la Régression Logistique Binaire

La fonction sigmoïde présentée dans la section précédente nous permet de calculer la probabilité $P\ (y = 1|x)$ pour une instance xxx. Mais comment prendre une décision quant à la classe à attribuer à une instance de test x? Pour une instance donnée xxx, nous classons l'observation comme appartenant à la classe 1 si la probabilité $P\ (\ y = 1 \mid x\ )$ est supérieure au seuil choisi, et comme appartenant à la classe 0 sinon.

Ce seuil d’est appelé **seuil de décision** ou **decision boundary**. Le processus de décision peut être formulé comme suit :

$$decision(x) = \begin{cases} 1\ si\ P(\ y = 1 \mid x) > seuil \\ 0\ sinon \end{cases}$$

Le seuil de 0.5 est couramment utilisé, mais il n'est pas toujours optimal. Dans certains cas, il peut être préférable de l'ajuster en fonction des besoins spécifiques. Par exemple, si l'on privilégie la précision (pour réduire les faux positifs) ou le rappel (pour réduire les faux négatifs), il est possible d'ajuster le seuil en conséquence. Il existe des techniques pour choisir judicieusement le seuil de décision.

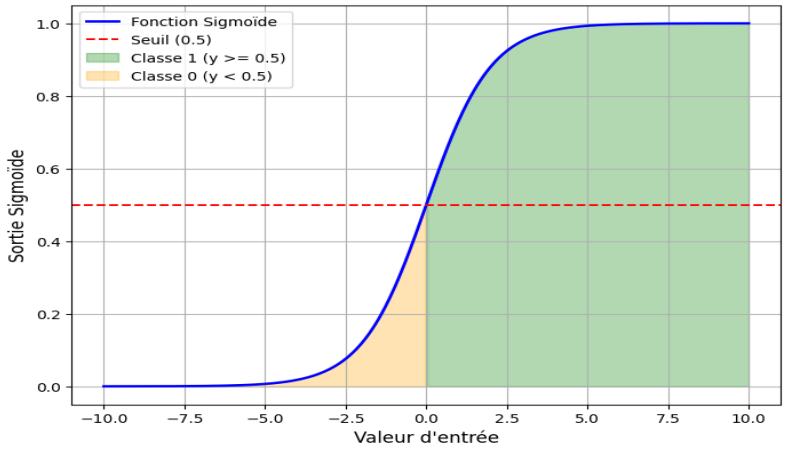

Figure 02 : Fonction sigmoïde logistique et classification

2.4 - Entraînement du Modèle

Lors de l'entraînement d'un modèle de régression logistique, l'objectif est d'apprendre les paramètres du modèle, à savoir les poids w et le biais b, de manière à ce que la prédiction du modèle soit aussi proche que possible de la vérité observée, c'est-à-dire la véritable étiquette y. Dans le cas de la régression logistique binaire, chaque observation x est associée à une étiquette correcte y, qui peut être soit 0 soit 1. Ce que le modèle produit, c'est une estimation $\hat{y}$ de la véritable étiquette y, basée sur la fonction logistique. L'objectif est de faire en sorte que $\hat{y}$ soit aussi proche que possible de y pour chaque observation du jeu d'entraînement.

Cela nécessite deux éléments clés. Le premier est une mesure de la proximité entre l'étiquette actuelle ($\hat{y}$) et l'étiquette réelle (y). Plutôt que de mesurer la similarité, on parle généralement de l'opposé de cela : la distance entre la sortie du système et la sortie idéale, et on appelle cette distance la fonction de perte ou **la fonction de coût**. Dans la section suivante, nous introduirons la fonction de perte couramment utilisée pour la régression logistique et aussi pour les réseaux neuronaux, **la perte d'entropie croisée binaire.**

La deuxième chose dont nous avons besoin est un algorithme d'optimisation pour mettre à jour itérativement les poids afin de minimiser cette fonction de perte. L'algorithme standard pour cela est **la descente de gradient** ; nous introduisons l'algorithme de descente de gradient stochastique dans la section suivante.

2.4.1 - La fonction de perte d'entropie croisée binaire

La fonction de perte d'entropie croisée binaire mesure l'écart entre la prédiction du modèle $\hat{y}$ et la véritable étiquette y, qui peut être 0 ou 1. Cette fonction de perte est utilisée pour évaluer la qualité des prédictions dans les tâches de classification binaire.

L'entropie croisée est dérivée de la probabilité que le modèle attribue à chaque étiquette y. Pour chaque observation z, la probabilité $p(\, y \mid z\,)$ est calculée à l'aide de la fonction sigmoïde, qui génère des valeurs entre 0 et 1. La perte d'entropie croisée est ensuite définie comme :

$$L_{CE}(\hat{y}, y) = -[\, y \log \hat{y} + (1 - y) \log (1 - \hat{y})\,]$$

Cette fonction de perte pénalise les erreurs de classification, avec une perte plus grande lorsque la prédiction du modèle s'éloigne de la vérité. L'objectif est de minimiser cette fonction en ajustant les paramètres du modèle w et b pour rendre les prédictions aussi proches que possible des étiquettes réelles.

2.4.2 - Descente de gradient

L'objectif principal de la descente de gradient est de trouver les paramètres optimaux du modèle, c'est-à-dire les poids $\boldsymbol{w}$ et le biais $\boldsymbol{b}$, qui minimisent la fonction de perte définie pour le modèle. Dans le cas de la régression logistique binaire, cette fonction de perte est la Binary Cross-Entropy Loss, qui dépend des paramètres $\boldsymbol{\theta} = \{\boldsymbol{w}, \boldsymbol{b}\}$

Le but est donc de minimiser cette fonction de perte en moyenne sur l'ensemble des exemples du jeu de données.

2.4.2.1 - Principe de la Descente de Gradient

La descente de gradient est une méthode d'optimisation utilisée pour trouver le minimum d'une fonction. Elle repose sur l'idée d'identifier la direction dans laquelle la pente de la fonction augmente le plus rapidement, et de se déplacer dans la direction opposée afin de réduire la valeur de la fonction.

Mathématiquement, cette pente correspond à la **dérivée** (ou au gradient, dans le cas de fonctions multivariées) de la fonction au point considéré. En procédant par itérations successives, la méthode ajuste les paramètres de manière à minimiser progressivement la fonction cible.

Pour mieux comprendre, imaginez que vous êtes dans un canyon et que vous essayez de descendre le plus rapidement possible vers la rivière au fond. Vous regardez autour de vous pour déterminer où le terrain est le plus en pente et descendez dans cette direction.

2.4.2.2 - Convexité de la Fonction de Perte

Une fonction convexe est une fonction mathématique dont la courbe, entre deux points quelconques de son domaine, se situe toujours en dessous ou sur la droite qui relie ces deux points. Cela se traduit par la propriété suivante : pour tout x_1, x_2 appartenant au domaine de la fonction et pour tout $\lambda \in [0,1]$, la fonction f vérifie :

$$f\left(\lambda x_1 + (1-\lambda)x_2\right) \leq \lambda f(x_1) + (1-\lambda) f(x_2)$$

Dans le cadre de l'apprentissage automatique, une propriété importante des fonctions convexes nous intéresse : elles possèdent **un seul minimum global** et pas de minimum local. Cette propriété garantit que la méthode de la descente de gradient, quel que soit le point de départ, trouvera toujours le minimum global.

Dans le cas de la régression logistique, la fonction de perte est convexe, ce qui facilite l'optimisation. En revanche, dans des contextes comme ceux des réseaux de neurones, la fonction de perte peut être non convexe, ce qui rend la recherche du minimum global plus complexe et susceptible de converger vers un minimum local.

2.4.2.3 - Algorithme de la Descente de Gradient

L'algorithme procède par étapes successives en mettant à jour les paramètres $\boldsymbol{w}$ et $\boldsymbol{b}$ dans la direction opposée au gradient de la fonction de perte $\boldsymbol{L}$. Le gradient est le vecteur qui pointe dans la direction où la fonction augmente le plus. Pour une fonction d'une seule variable (par exemple, $\boldsymbol{w}$), le gradient correspond simplement à la dérivée.

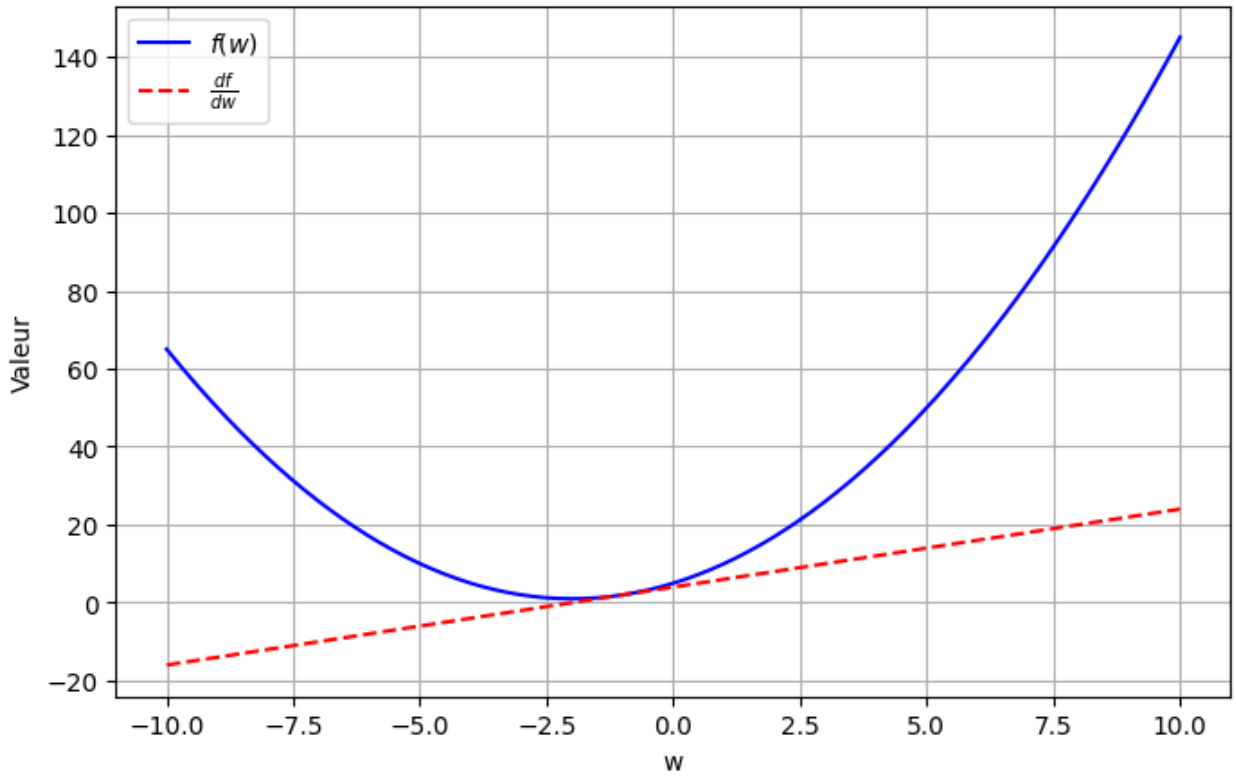

Figure 03 : Fonction Convexe f(w) et son Gradient

1- Initialisation : On commence par initialiser les paramètres w et b à des valeurs aléatoires.

2- Calcul du gradient : À chaque itération, on calcule le gradient de la fonction de perte L par rapport aux paramètres actuels.

3- Mise à jour des paramètres : Les paramètres sont mis à jour selon la formule suivante :

$$w := w - \eta \cdot \frac{\partial L}{\partial w} \quad , \quad b := b - \eta \cdot \frac{\partial L}{\partial b}$$

Où η est **Le taux d'apprentissage** (learning rate) qui est un hyperparamètre du modèle qui contrôle la vitesse à laquelle les paramètres sont ajustés. Le choix d'un bon taux d'apprentissage est crucial pour un entraînement efficace du modèle. Il est généralement déterminé par expérimentation, à

l'aide de techniques comme la recherche sur grille (grid search) ou la recherche aléatoire (random search).

4- Répétition des étapes 2 et 3 jusqu'à ce qu'un des critères suivants soit satisfait :

Convergence de la fonction de perte : L'algorithme s'arrête lorsque la fonction de perte (ou le coût) atteint un minimum ou ne diminue plus de manière significative entre deux itérations successives. Cela signifie que l'algorithme a trouvé un minimum local ou global, et que les ajustements des paramètres n'ont plus d'impact notable sur la réduction de l'erreur.

Nombre d'itérations maximal atteint : L'algorithme peut être configuré pour s'arrêter après un certain nombre d'itérations (epochs), même si la fonction de perte n'est pas complètement convergée. Cela permet de limiter le temps d'entraînement, notamment dans les cas où celui-ci pourrait durer trop longtemps.

Seuil de tolérance pour la mise à jour des paramètres : L'algorithme peut être configuré pour s'arrêter si les mises à jour des paramètres (ou du gradient) sont inférieures à un seuil défini. Si les changements dans les paramètres deviennent suffisamment petits, cela indique que l'optimisation approche de son optimum, et l'algorithme peut alors être arrêté.

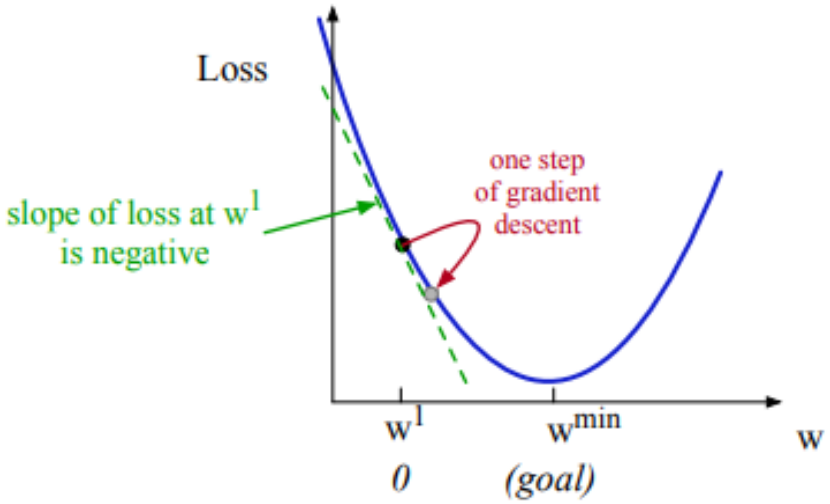

Figure 04 : Evaluation de la fonction de perte lors d'une étape de descente gradient

2.5 – Avantages et limites de la Régression Logistique Binaire

L'évaluation des bénéfices de la régression logistique binaire permet d'identifier les situations où cette méthode est particulièrement pertinente, tandis que l'analyse de ses contraintes permet de cerner ses limites et d'envisager des approches alternatives lorsque nécessaire. Nous allons maintenant lister quelques exemples des avantages associés à cette méthode :

1. **Facilité d'interprétation :**
   Les coefficients de la régression logistique peuvent être directement liés à l'effet des variables indépendantes sur la probabilité de l'événement cible, ce qui facilite leur interprétation.

2. **Robustesse avec peu de données :**
   Elle fonctionne bien même avec des ensembles de données relativement petits, à condition que les hypothèses de base soient respectées.

3. **Simplicité de mise en œuvre :**
   Elle est facile à comprendre et à implémenter à l'aide de bibliothèques statistiques ou de machine learning courantes.

Il est aussi important de noter que la régression logistique binaire repose sur certaines conditions qui doivent être respectées pour garantir la validité des résultats obtenus. Voici les principales hypothèses du modèle :

1. **Hypothèse de linéarité**
   La régression logistique suppose une relation linéaire entre les variables indépendantes et la log-odds (logarithme des cotes), ce qui peut limiter son efficacité pour des relations complexes.

2. **Hypothèse de la variable dépendante binaire**
   La régression logistique suppose que la variable dépendante prend uniquement deux valeurs distinctes (par exemple, Oui/Non, Vrai/Faux). Cette hypothèse est essentielle, car le modèle est conçu pour prédire des résultats binaires.

3. **Hypothèse de l'absence de multi colinéarité entre les variables explicatives**
   La régression logistique suppose qu'il n'y a pas de multi colinéarité forte entre les variables explicatives. La multi colinéarité se produit lorsque deux ou plusieurs variables sont fortement corrélées et fournissent des informations redondantes au modèle, ce qui peut entraîner des divergences dans l'ajustement du modèle.

4. **Hypothèse de l'absence de valeurs aberrantes extrêmes**
   La régression logistique binaire suppose qu'il n'y a pas de valeurs aberrantes extrêmes ou d'observations influentes qui pourraient fausser les résultats du modèle.

2.6 - Domaines d’applications de Régression Logistique Binaire

La régression logistique binaire est largement utilisée dans divers domaines grâce à sa simplicité et son efficacité pour résoudre des problèmes de classification binaire. Voici les principaux domaines d’application :

- **Santé et médecine** :

  Prédiction de maladies (par exemple, la probabilité qu’un patient ait une maladie à partir de symptômes ou d’analyses médicales), évaluation des risques de rechute ou de mortalité, ou encore classification des images médicales comme étant normales ou anormales.

- **Marketing et commerce** :

  Prédiction de l’engagement des clients (par exemple, probabilité qu’un client achète un produit ou clique sur une annonce), identification des clients susceptibles de se désabonner (churn), ou classification des leads comme qualifiés ou non.

- **Finance et assurance** :

  Analyse des risques de crédit (évaluer si un individu peut rembourser un prêt), détection de fraudes dans les transactions financières, ou évaluation des probabilités d'indemnisation dans les réclamations d’assurance.

3. Partie Pratique :

3.1 – Base de données

Les applications de la Régression Logistique Binaire couvrent des domaines variés : de la classification des e-mails à la détection du cancer, en passant par la prédiction météo, la différenciation entre mines et roches dans les données sonar, l'évaluation des prêts bancaires, la reconnaissance vocale et la prévention des maladies cardiaques. Dans cette section, nous illustrerons ces usages à travers des exemples pratiques.

3.2 – Régression Logistique Binaire dans le domaine médical

La régression logistique binaire est importante en médecine car elle aide à prédire si une personne a ou n'a pas une maladie en fonction de différents facteurs. Par exemple, elle peut être utilisée pour estimer le risque de cancer ou de maladies cardiaques. Cela aide les médecins à prendre de meilleures décisions pour le traitement et le suivi des patients.

3.2.1 Importation des bibliothèques nécessaires

```python
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split, learning_curve, cross_val_score
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, accuracy_score, confusion_matrix,log_loss
```

3.2.2 Lecture de dataset

```python
data = pd.read_csv('heart.csv')
data.head()
```

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	63	1	3	145	233	1	0	150	0	2.3	0	0	1	1
1	37	1	2	130	250	0	1	187	0	3.5	0	0	2	1
2	41	0	1	130	204	0	0	172	0	1.4	2	0	2	1
3	56	1	1	120	236	0	1	178	0	0.8	2	0	2	1
4	57	0	0	120	354	0	1	163	1	0.6	2	0	2	1

Figure 05 : Affichage des cinq premières lignes des données

3.2.3 Prétraitement des données

Filtrage des colonnes et gestion des valeurs manquantes.

```
data.isna().sum()
```

```
age         0
sex         0
cp          0
trestbps    0
chol        0
fbs         0
restecg     0
thalach     0
exang       0
oldpeak     0
slope       0
ca          0
thal        0
target      0
dtype: int64
```

Figure 06 : Aucune valeurs nulles

3.2.3.1 – Boxplot pour détecter les outliers

```
fig, axs = plt.subplots(ncols=2, figsize=(10, 5))

sns.boxplot(data=data.drop(['trestbps', 'chol', 'thalach', 'age', 'target', 'sex', 'fbs', 'thal'],
                           axis=1),
                           ax=axs[0])
axs[0].set_title('Boxplot pour les variables avec de petites valeurs')

sns.boxplot(data=data[['trestbps', 'chol', 'thalach', 'age']],
            ax=axs[1])
axs[1].set_title('Boxplot pour les variables avec de grandes valeurs')

plt.tight_layout()
```

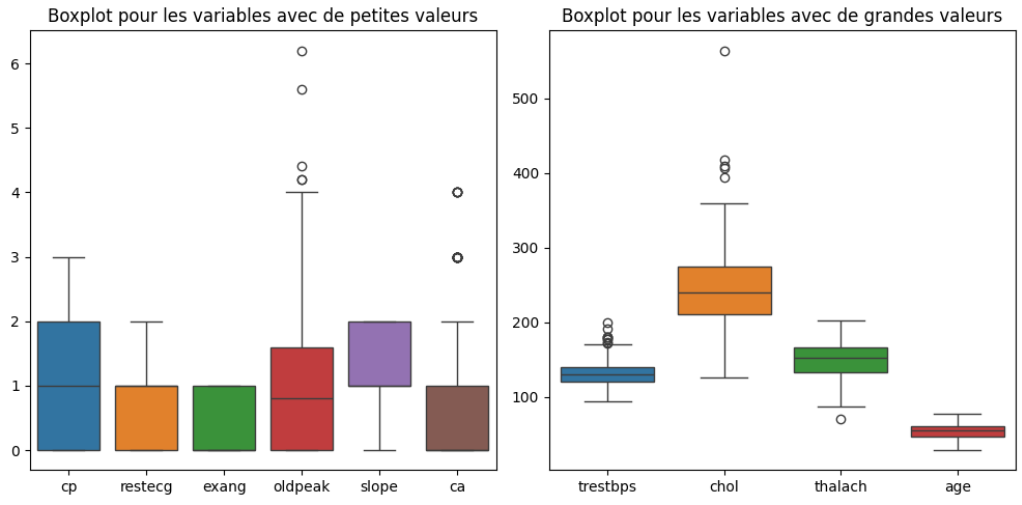

Figure 07 : Boxplot variables

3.2.3.2 – Retirer les outliers

```
for feature in features:
    Q1 = data[feature].quantile(0.25)
    Q3 = data[feature].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    data = data[(df[feature] >= lower_bound) & (df[feature] <= upper_bound)]
```

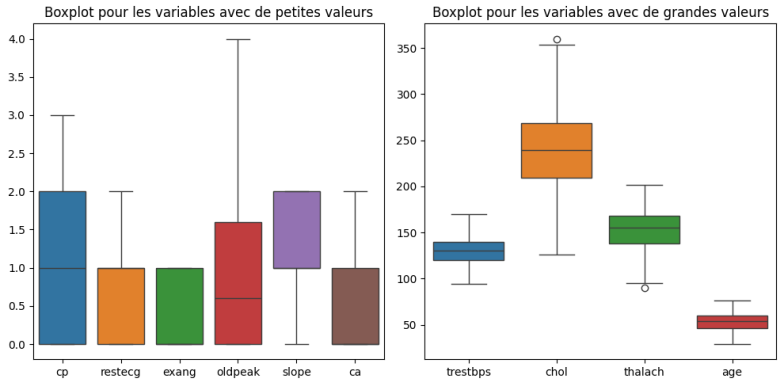

Figure 08 : Boxplot des variables sans outliers

3.2.3.3 – Corrélation entre les variables

```
plt.figure(figsize=(10, 10))
sns.heatmap(data.corr(), annot=True, fmt='.2f', cmap='coolwarm',vmin=-1,vmax=1)
plt.show()
```

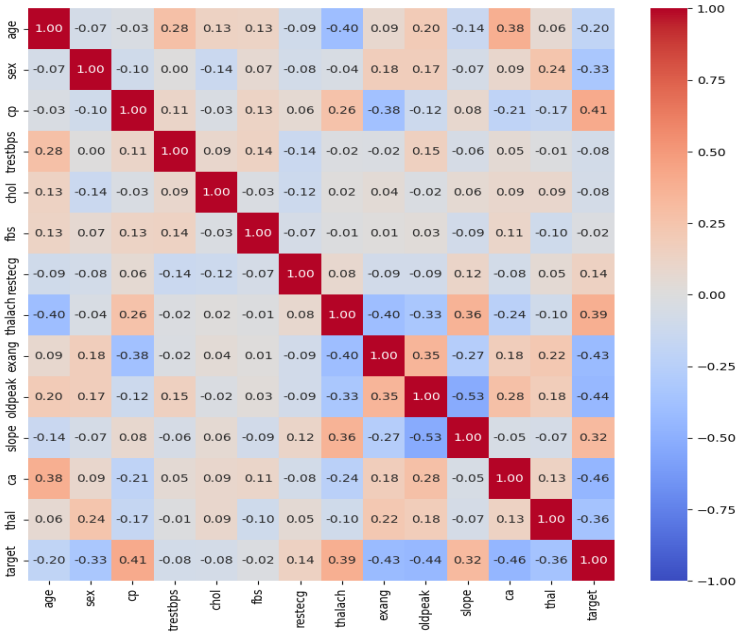

Figure 09 : Matrice de corrélation entre les variables

3.2.3.4 – Séparation des variables

Cette section du code sépare les caractéristiques (X) de la variable cible (y). Ensuite, elle applique une normalisation aux données avec *StandardScaler* pour standardiser les variables explicatives. Enfin, les données sont divisées en ensembles d'entraînement et de test (75 % / 25 %) pour préparer l'évaluation du modèle.

```
X = df.drop('target', axis=1)
y = df['target']
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=40)
```

3.2.4 Entrainement du Modèle

Un modèle de régression logistique est initialisé et ajusté aux données d'entraînement (X_train, y_train). Ensuite, il effectue des prédictions sur l'ensemble de test (X_test) pour évaluer sa performance.

```
model = LogisticRegression()
model.fit(X_train,y_train)

y_pred = model.predict(X_test)
```

3.2.5 Evaluation du Modèle

3.2.5.1 – L'exactitude du modèle

L'exactitude du modèle est calculée et affichée, suivie du rapport de classification qui donne des informations détaillées sur la précision, le rappel et le score F1 pour chaque catégorie. Le rapport montre aussi les scores moyens pour l'ensemble des catégories.

```
print(f'L\'exactitude : {accuracy_score(y_test, y_pred)*100:.2f} %')
print('Le rapport de classification')
print(classification_report(y_test,y_pred))
```

```
L'exactiture : 96.23%
classification_report
              precision    recall  f1-score   support

           0       1.00      0.92      0.96        26
           1       0.93      1.00      0.96        27

    accuracy                           0.96        53
   macro avg       0.97      0.96      0.96        53
weighted avg       0.96      0.96      0.96        53
```

Figure 10 : Résultats d'évaluation du modèle

3.2.5.2 – Exactitude de chaque classe

```python
accuracy_0 = accuracy_score(y_test[y_test==0],y_pred[y_test==0])
accuracy_1 = accuracy_score(y_test[y_test==1],y_pred[y_test==1])

print(f'Exactitude pour la catégorie 0 : {accuracy_0*100:0.2f}')
print(f'Exactitude pour la catégorie 1 : {accuracy_1*100:0.2f}')
```

```
Exactitude pour la catégorie 0 : 92.31
Exactitude pour la catégorie 1 : 100.00
```

Figure 11 : Résultats d'exactitude pour chaque catégorie

3.2.5.3 - Matrice de Confusion

```python
conf_matrix = confusion_matrix(y_test, y_pred)
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='viridis')
plt.xlabel('Predicted')
plt.ylabel('Vraie Valeur')
plt.title('Confusion matrix')
plt.show()
```

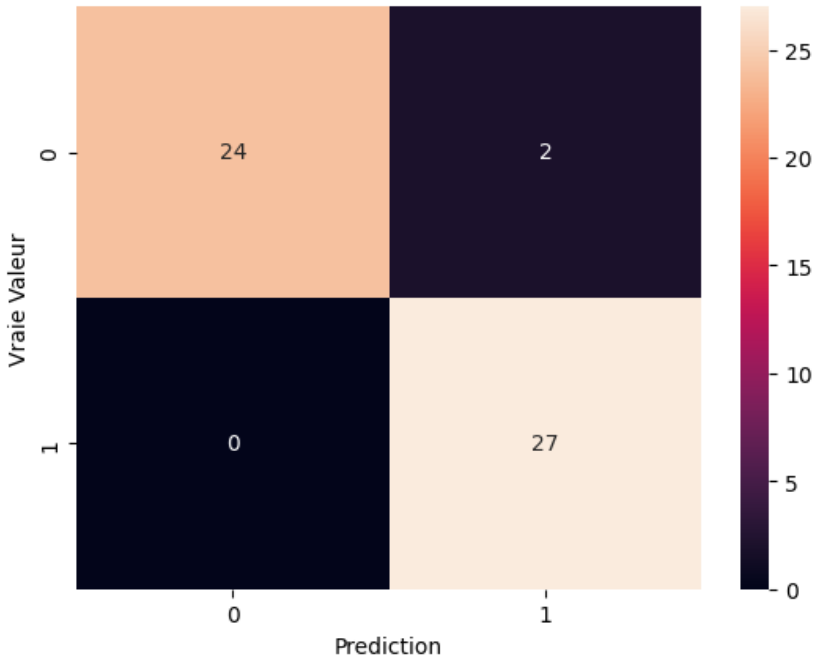

Figure 12 : Matrice de confusion

Le modèle a correctement prédit 24 cas sans maladie cardiaque et 27 cas avec maladie cardiaque (classe 1), avec 2 faux positifs (cas sans maladie cardiaque prédit comme malades) et 0 faux négatif (cas de maladie cardiaque non détecté).

Cela suggère que le modèle a une très bonne capacité à identifier les cas de maladies cardiaques ainsi que les cas sans maladie cardiaque, bien qu'il y ait quelques erreurs de classification, principalement des faux positifs.

3.2.5.4 - Courbe d'apprentissage

La courbe d'apprentissage montre l'évolution du score d'entraînement et du score de validation en fonction de la taille de l'ensemble d'entraînement. Elle permet d'évaluer si le modèle surapprend (overfitting) ou sous-apprend (underfitting).

```
n, tr_score, val_score = learning_curve(model, X_train, y_train, train_sizes=np.linspace(0.2, 1, 5), cv=5)
plt.plot(n, tr_score.mean(axis=1), label="Score d'entraînement")
plt.plot(n, val_score.mean(axis=1), label="Score de validation")
plt.title("Courbe d'apprentissage")
plt.xlabel("Taille de l'ensemble d'entraînement")
plt.ylabel("Score")
plt.legend()
plt.show()
```

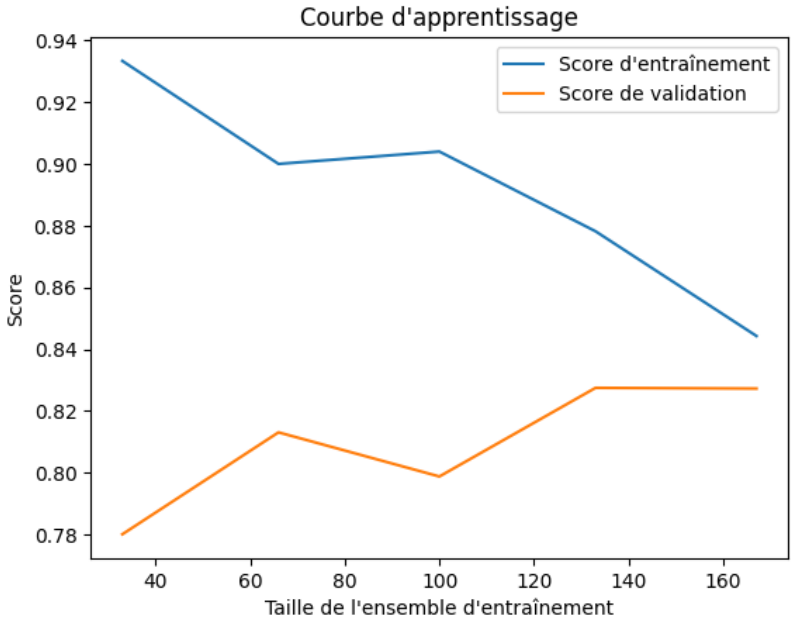

Figure 13 : Visualisation de la courbe d'apprentissage

Le modèle semble bien équilibré, avec des scores d'entraînement et de validation proches les uns des autres. Il n'y a pas de signe évident de surapprentissage ou de sous-apprentissage, ce qui suggère qu'il est performant à mesure que l'ensemble d'entraînement augmente.

3.2.5.5 – Validation croisée

La validation croisée permet de mieux évaluer la performance du modèle en le testant sur plusieurs sous-ensembles des données d'entraînement. Elle aide à éviter la variance des résultats causée par une division aléatoire des données.

```python
scores = cross_val_score(model, X_train, y_train, cv=5)
print(f'Cross-validated scores:')
for i in range(len(scores)):
    print(f'Fold {i+1}: {scores[i]:.2f}')
```

```
Cross-validated scores:
Fold 1: 0.87
Fold 2: 0.87
Fold 3: 0.90
Fold 4: 0.85
Fold 5: 0.81
```

Figure 14 : Résultats de la validation croisée

Le modèle semble bien généralisé, avec des performances stables sur différents sous-ensembles de données. Les petites variations dans les scores (entre 0.96 et 1.00) sont normales et indiquent que le modèle ne souffre pas de surapprentissage.

3.2.5.6 – Log-Loss vraisemblance

La Log Loss (ou log-vraisemblance) mesure l'écart entre les probabilités prédites par le modèle et les vraies étiquettes. Une Log Loss plus faible indique que le modèle fait des prédictions plus proches de la réalité, et une Log Loss plus élevée indique un écart plus grand.

```
y_pred_prob = model.predict_proba(X_test)[:, 1]
log_loss_value = log_loss(y_test, y_pred_prob)
print(f"Log Loss: {log_loss_value}")
```

Une Log Loss de **0.2302** indique que le modèle est **modérément performant**, avec des erreurs plus significatives dans ses prédictions de probabilité

4. Conclusion

La régression logistique binaire représente un outil fondamental en analyse prédictive, offrant une méthode rigoureuse pour transformer des données complexes en décisions binaires. Sa polyvalence se manifeste à travers ses applications diversifiées, des diagnostics médicaux aux stratégies marketing.

Cependant, sa simplicité n'est pas sans limites. Les hypothèses de linéarité et de relations binaires strictes peuvent restreindre sa performance face à des problèmes plus complexes. Pour des scénarios nécessitant une modélisation plus sophistiquée, des alternatives telles que les arbres de décision, les forêts aléatoires, ou les réseaux de neurones offrent des approches plus flexibles et adaptatives.

Au-delà de sa simplicité apparente, cette technique statistique révèle la puissance de l'analyse mathématique pour comprendre et prédire des comportements complexes. Chaque modèle construit raconte une histoire de probabilités, où la réalité se dessine entre les lignes d'un calcul précis.

La régression logistique demeure un pilier essentiel de l'apprentissage automatique, témoignant de l'élégance et de la précision des méthodes statistiques modernes.

5. Références

Panda, Nihar. (2022). A Review on Logistic Regression in Medical Research. National Journal of Community Medicine. 13. 265-270. 10.55489/njcm.134202222.

Ma, Qianyou. (2024). Recent applications and perspectives of logistic regression modelling in healthcare. Theoretical and Natural Science. 36. 185-190. 10.54254/2753-8818/36/20240614.

Banerjee, Debashmita & Kumar, Roshan & Tripathi, Srishti & Murry, Benrithung. (2024). Application of Binary Logistic Regression in Biological Studies. Journal of the Practice of Cardiovascular Sciences. 10. 48-52. 10.4103/jpcs.jpcs_82_23.

Stephenson, B. (2008). Binary response and logistic regression analysis. www.public.iastate.edu

https://www.youtube.com/watch?v=pOsqQ0MAHrY