Compte rendu

Régression Linéaire : Implémentation et Application

Souheib Ben Mabrouk

INDP2E

15 octobre 2021

Introduction

La régression linéaire est une technique d'analyse prédictive de base qui utilise des données historiques pour prédire une variable de sortie. Elle est populaire pour la modélisation prédictive car elle est facile à comprendre.[1]

Les modèles de régression linéaire ont de nombreuses applications dans le monde réel, dans un large éventail de secteurs tels que l'économie (par exemple, la prévision de la croissance), les affaires (par exemple, la prévision des ventes de produits, les performances des employés), les sciences sociales (par exemple, la prévision des tendances politiques à partir du sexe ou de la race), les soins de santé (par exemple, la prévision des niveaux de pression artérielle à partir du poids, l'apparition de maladies à partir de facteurs biologiques), etc.

Comprendre comment mettre en œuvre des modèles de régression linéaire peut permettre de découvrir des histoires dans les données pour résoudre des problèmes importants.
L'idée de base est que si nous pouvons adapter un modèle de régression linéaire aux données observées, nous pouvons alors utiliser le modèle pour prédire toute valeur future.

L'objectif de ce rapport est de présenter la méthode dans le cadre du cours de l'optimisation ainsi que implémenter quelques cas d'étude tout en tirant des conclusions.

Pour ce faire, j'ai choisi de travailler avec **Python 3.8** disponible en ligne par le recours à **Google Colaboratory** qui me facilite le travail grâce à sa simplicité et son efficacité en termes de temps d'exécution.
D'autre part, j'utiliserai le système de composition de documents **LATEX** afin d'avoir un rapport bien conçu avec les formulations mathématiques nécessaires.

Fig. 1 – Logo de python

Fig. 2 – Logo de google colab

Fig. 3 – Logo de LATEX

Table des matières

Table des figures

Chapitre 1

Principe

1.1 Présentation

L'analyse de régression linéaire sert à prévoir la valeur d'une variable en fonction de la valeur d'une autre variable. La variable dont vous souhaitez prévoir la valeur est la variable dépendante. La variable que vous utilisez pour prévoir la valeur de l'autre variable est la variable indépendante.

Ce type d'analyse estime les coefficients de l'équation linéaire, impliquant une ou plusieurs variables indépendantes, qui estiment le mieux la valeur de la variable dépendante. La régression linéaire consiste en la détermination d'une droite ou d'une surface qui réduit les écarts entre les valeurs de sortie prévues et réelles. Il existe des calculatrices de régression linéaire simple qui utilisent une méthode des moindres carrés pour découvrir la ligne la mieux adaptée pour un ensemble de données appariées. La valeur de X (variable dépendante) est ensuite estimée à partir de Y (variable indépendante). [2]

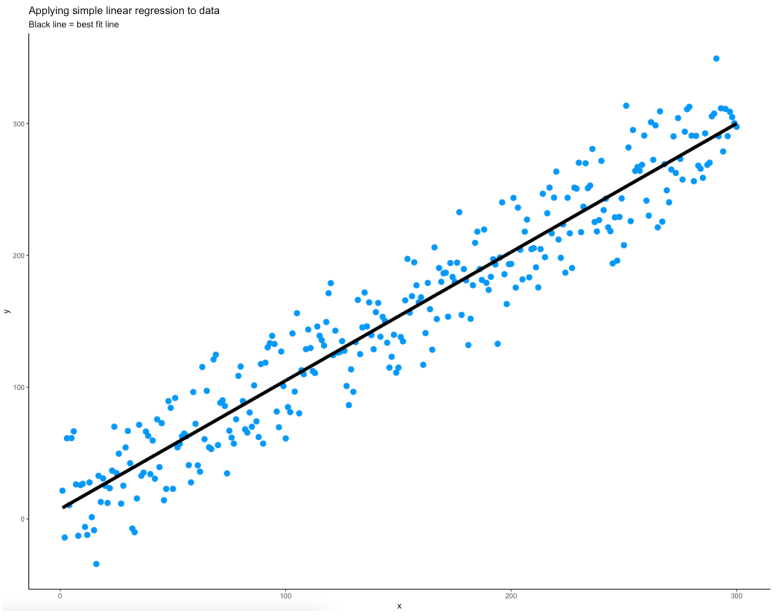

Fig. 1.1 – Exemple d'une regression linéaire

1.2 Modélisation Mathématique

1.2.1 Motivation

Pour savoir si le modèle linéaire est oui ou non pertinent pour l'étude de notre phénomène, Le graphique est au départ un nuage de points avec lequel on relève la tendance qu'a la forme de ce nuage de points. [3]

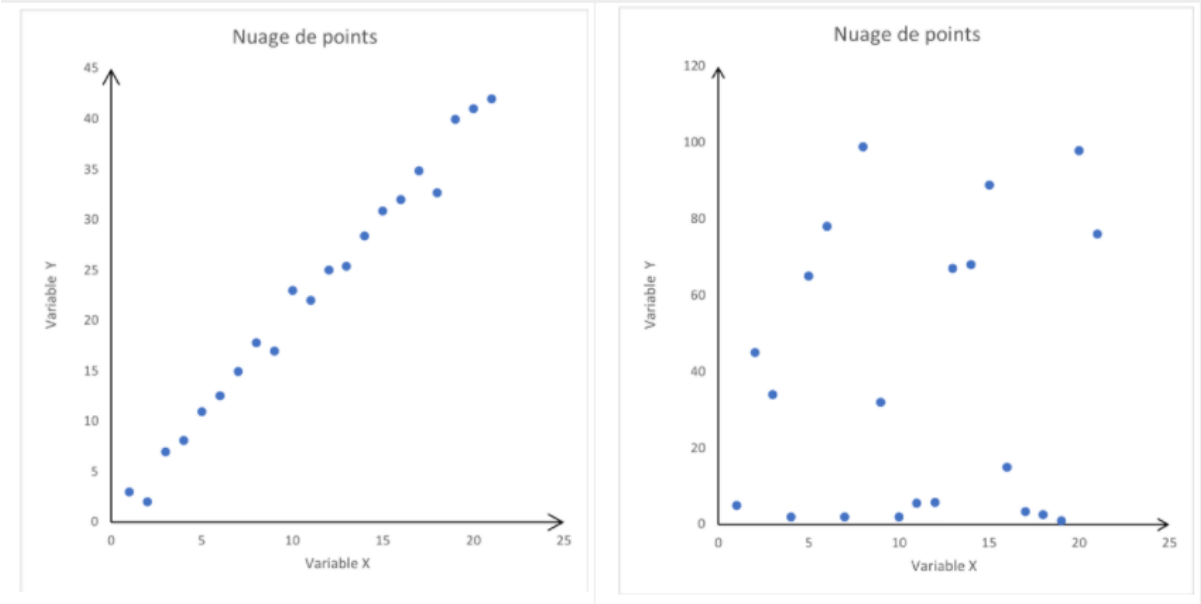

Fig. 1.2 – Motivation à la modélisation linéaire

Au vu de ces deux graphiques, il semble approprié d'utiliser le modèle linéaire pour la première image et pas pour la deuxième qui ne laisse transparaitre aucune tendance connue. Dans la suite nous expliquerons la modélisation et l'estimation des paramètres de la fonction de prédiction pour pouvoir tracer cette droite.

1.2.2 Regression simple / multiple

Modélisation	Nature de la régression
Une seule variable explicative X	Régression simple
Plusieurs variables explicatives Xj	Régression multiples

Un modèle de régression linéaire simple est de la forme :

$$\begin{cases} Y = \theta X + b + \varepsilon \\ f(X) = \theta X + b \end{cases} \tag{1.1}$$

Avec :
— Y, la variable cible, aléatoire dépendante
— θ et b, les coefficients (pente et ordonnée à l'origine) à estimer
— X, la variable explicative, indépendante
— ε, une variable aléatoire qui représente l'erreur

Un modèle de régression linéaire multiple est de la forme :

$$\begin{cases} Y = \theta 1x1 + \theta 2x2 + \theta 3x3 + ... + b + \varepsilon \\ f(X) = \theta X + b \end{cases} \tag{1.2}$$

Avec :
— Y, la variable cible, aléatoire dépendante
— θ = (θ1,θ2, .. ,θn) et b, les coefficients à estimer
— X =(x1,x2, .. ,xn), la variable explicative, indépendante
— ε, une variable aléatoire qui représente l'erreur

Pour obtenir la **droite optimale** qui représente de la meilleure façon Y en fonction de X, on a tendance à minimiser ε.
Plusieurs modèles ont été conçus pour minimiser cette fonction dont le plus célèbre est celui de **Moindres carrés** ou **Least Square**.

1.2.3 Moindres carrés

La méthode des moindres carrés consiste à minimiser la somme des carrés des écarts, écarts pondérés dans le cas multidimensionnel, entre chaque point du nuage de régression et son projeté, parallèlement à l'axe des ordonnées, sur la droite de régression. [4]

Le principe des moindres carrés ordinaires consiste à choisir les valeurs de a et b qui minimisent les erreurs de prédiction ou les résidus sur un jeu de données d'apprentissage :

$$\varepsilon = \sum_{i=0}^{N} (Y_i - (\theta X_i + b))^2 \tag{1.3}$$

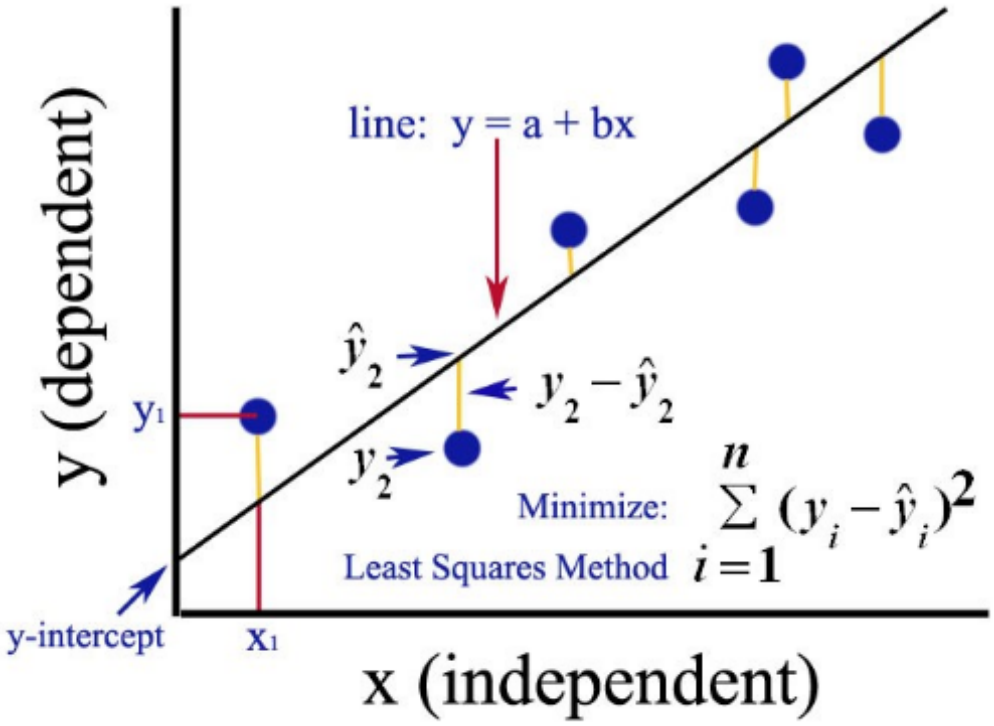

Fig. 1.3 – Représentation de la méthode des Moindres Carrés

Notre Objectif sera donc de mettre en oeuvre la méthode des moindres carrés et de l'implémenter sous python pour concevoir la regression linéaire de la dataset étudiée.

Chapitre 2

Implémentation

2.1 Préparation de la dataset

On implémente dans cette partie un exemple d'application à partir de la bibliothèque **sklearn** de python. On choisit pour cela une dataset à 100 exemples et présentant une variable Y corrélée avec X avec un bruit gaussien qui n'affecte pas fortement la forme des données.
Le code suivant traduit la création et la visualisation de la dataset.

```
import numpy as np
from sklearn import datasets
import matplotlib.pyplot as plt
x,y = datasets.make_regression(n_samples=100, n_features=1,noise=20 , random_state=4)
plt.scatter(x, y, color='b',marker="o", s=30)
```

Le résultat de ce code est donnée par la figure :

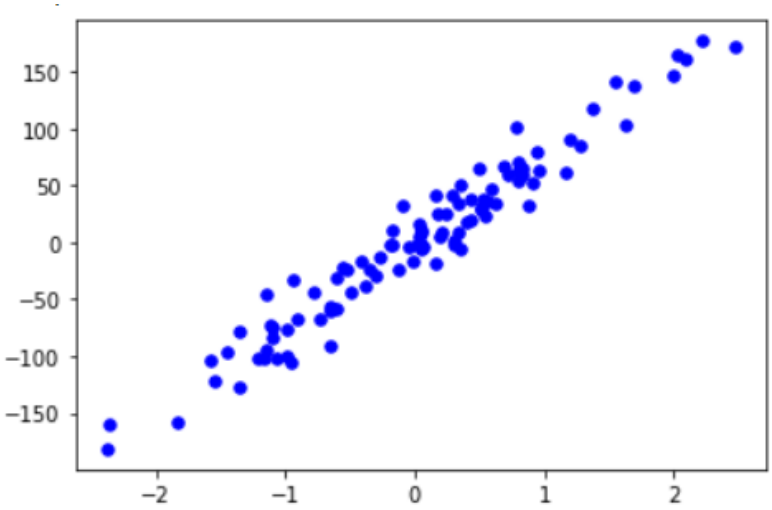

Fig. 2.1 – Visualisation de la dataset

On remarque que, comme prévu, les données sont susceptibles à être modélisées par une droite affine .

2.2 Optimisation par la méthode des moindres carrés

Partant de l'équation (1.3) On pose J la **fonction coût** qu'on a l'objectif de la minimiser :

$$J(\theta, b) = \frac{1}{N}\sum_{i=0}^{N}(Y_i - (\theta X_i + b))^2 \tag{2.1}$$

Cette fonction est **différentiable et convexe** donc calculons son gradient :

$$\begin{cases} \frac{dJ}{d\theta}(\theta, b) = \frac{1}{N}\sum_{i=1}^{n} -2x_i(y_i - (\theta x_i + b)) \\ \frac{dJ}{db}(\theta, b) = \frac{1}{N}\sum_{i=1}^{n} -2(y_i - (\theta x_i + b)) \end{cases} \tag{2.2}$$

On applique **l'algorithme du gradient descendant à pas fixe p** pour la recherche d'un minimum absolu de la fonction J (étant une fonction convexe un minimum existe forcément et il est absolu)

$$\begin{cases} p = 0.001 \quad , \theta = O_n \quad , b = 0 \\ \quad Pour\ k = 0...N_{iterations} \\ \quad\quad \theta = \theta - p \cdot \frac{dJ}{d\theta} \\ \quad\quad b = b - p \cdot \frac{dJ}{db} \\ \quad fin\ pour \end{cases} \tag{2.3}$$

Remarque :
Le minimum cherché peut ne pas être unique comme l'exemple ci dessous :

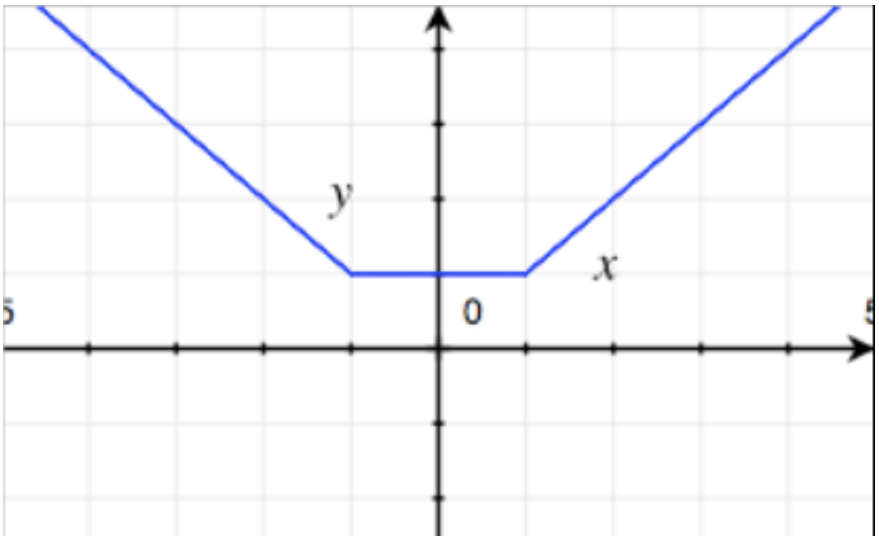

Fig. 2.2 – Exemple d'une fonction convexe mais pas strictement convexe

- On a utlisé, le critère de moindres carrés dans ce programme pour trouver les bonnes valeurs de θ et b

- La fonction J est **convexe** donc le gradient descendant converge essentiellement vers **un minimum absolu** de la fonction.

2.3 Implémentation de l'algorithme en Python

Après avoir mettre en oeuvre la modélisation mathématique du problème et l'optimiser dans le cadre des moindres carrés, on code le raisonnement antérieur sous forme d'un code python avec des commentaires à chaque étape pour expliquer le choix des variables et des opérations effectuées.

```
import numpy as np
from sklearn import datasets
import matplotlib.pyplot as plt

class MY_L_R:
    def __init__(self, pas=0.001, N_iterations=10000):
        self.pas = pas
        self.N_iterations = N_iterations #nbre d'N_iterations
        self.teta = np.zeros(1)
        self.b = 0
# Creation de la classe avec les initialisations nécessaires de l'equation (2.3)

    def training(self, X, Y):
        for i in range(self.N_iterations):
            y1 = np.dot(X, self.teta) + self.b
            self.teta = self.teta + (1/100)*self.pas*(np.dot(X.T, (Y-y1)))
            self.b = self.b + (1/100)*self.pas*(np.sum(Y-y1))

# La Methode qui va calculer teta et b avec (2.3)

    def predict(self, X, Y):
        print('start of prediciton')
        y1 = np.dot(X, self.teta) + self.b
        print('prediction: ')
        print(y1)
        print('True Value: ')
        print(Y)
        print('MSE: ')
        print(np.mean((Y-y1)**2))
```

```
        return y1
        # La methode qui va faire la prédiction

liner = MY_L_R()
liner.training(x,y)
liner.predict(x[10],y[10])
# L'appel des fonctions fait


y1_line = liner.predict(x,y)
cmap = plt.get_cmap("viridis")
fig = plt.figure(figsize=(8, 6))
print(x.shape, y.shape)
m1 = plt.scatter(x, y, color='b', s=30)

plt.plot(x, y1_line, color="red", linewidth=1, label="Prediction")
plt.show()
# Visualisation de model
```

Le résultat de ce code est donnée par la figure :

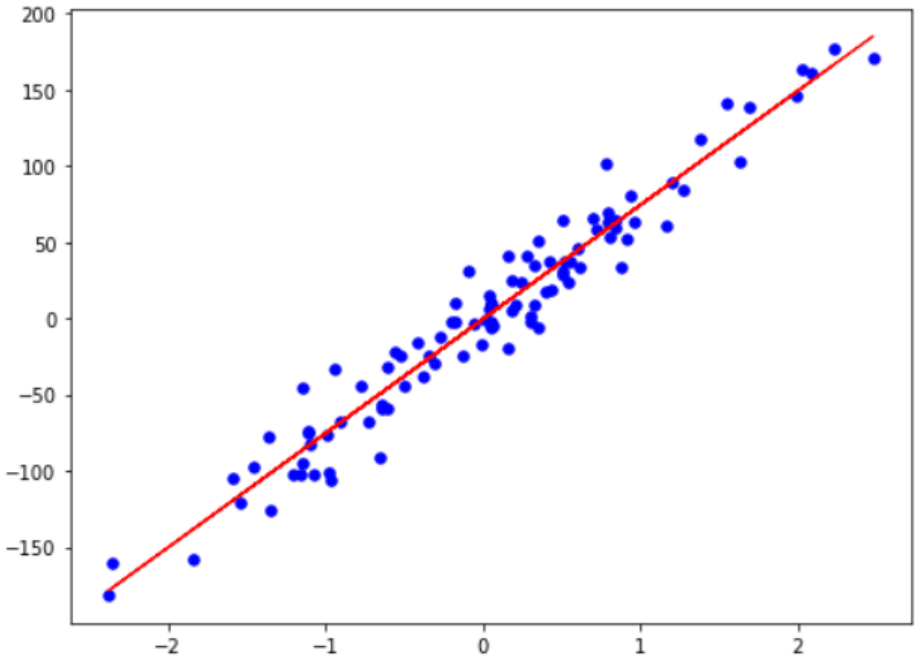

Fig. 2.3 – Résultat final

Remarque :

Pour ces paramètres, On obtient un MSE = 290.1597501271552

2.4 Interprétations des résultats

On a réussi à mettre en oeuvre la méthode des moindres carrés dans notre exemple de dataset et le code obtenu est généralisable pour toutes dataset à éléments corrélés linéairement.
On remarque qu'en jouant sur les paramètres pas et Nitérations, on peut avoir des résultats différents et plus que Nitérations est grand et lamda est petite, on aura un classement plus performant mais qui coute plus en terme de temps et de la mémoire.
Donc, on peut contrôler ses paramètres selon les capacités Hardware ainsi que la nature du problème et la précision qu'il exige.

N=10, MSE=5509 **N=1000**, MSE=1089 **N=10000**, MSE=290 **N=100000**, MSE=290

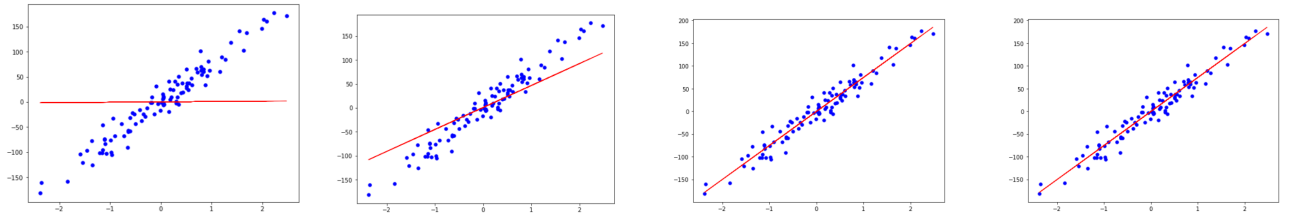

Fig. 2.4 – Effet de changement de $N_{iterations}$ *pour* $p = 0.001$

On remarque que pour un nombre d'itérations relativement faible, l'algorithme du gradient à pas fixe p=0.001 ne converge pas et on a un MSE relativement important. Par contre à partir de 10000 itérations, on a la convergence de l'algorithme pour **un MSEmin=**290 **.**

p=0.00001, MSE=4690 **p=0.0001**, MSE=1089 **p=0.001**, MSE=290 **p=0.01**, MSE=290

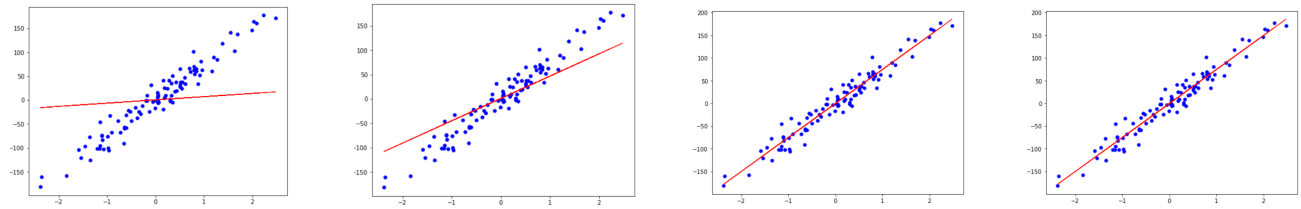

Fig. 2.5 – Effet de changement de pas p à N=10000

On remarque que pour un nombre d'itérations fixe N=10000, l'algorithme du gradient à pas fixe p ne converge qu'à partir de p>= pmin=0.001.

Chapitre 3

Creuser d'avantage

3.1 Problème

Le modèle de régression linéaire est probablement le modèle de prédiction le plus simple et le plus couramment utilisé. L'utilisation d'un modèle de régression linéaire présente de nombreux avantages. Le plus important est que, sous l'hypothèse d'une distribution normale i.i.d. des termes d'erreur, les estimateurs MCO (moindres carrés ordinaires) du modèle de régression linéaire sont sans biais (sur la base du théorème de Gauss-Markov), ce qui permet de tirer des conclusions utiles.

Mais il y a des cas ou le modèle de régression linéaire classique ne gère pas bien : Quand il y a multicolinéarité.Multicolinéarité est le phénomène selon lequel une (ou plusieurs) des variables indépendantes peut être exprimée comme la combinaison linéaire d'autres variables indépendantes. En fait, ce problème existe presque partout dans le monde réel. Lorsque le nombre de variables indépendantes est supérieur au nombre d'observations. Lorsque cela se produit, les estimations OLS ne sont pas valides principalement parce qu'il existe des solutions infinies pour nos estimateurs.

Mais il existe des cas que le modèle de régression linéaire classique ne gère pas bien. [5]

1- Lorsqu'il y a **multicollinéarité.** [1] En fait, ce problème existe presque partout dans le monde réel.

2- Lorsque le nombre de variables indépendantes est supérieur au nombre d'observations. Lorsque cela se produit, les estimations des selon Moindres carrés ne sont pas valides parce qu'il existe des solutions infinies aux estimateurs.

Par conséquent, nous avons besoin de meilleures alternatives pour résoudre ces problèmes.

1. La multicollinéarité est le phénomène selon lequel une (ou plusieurs) des variables indépendantes peut être exprimée comme une combinaison linéaire d'autres variables indépendantes.

3.2 Idée de la régularisation

La régularisation est un processus qui consiste à introduire des informations supplémentaires afin de résoudre un problème mal posé ou d'éviter un surajustement. Une façon de régulariser est d'ajouter une contrainte à la fonction de perte :

Perte régularisée = Fonction de perte + Contrainte

Il existe de nombreuses formes différentes de contraintes que nous pouvons utiliser pour régulariser. La forme la plus populaire est **la régression Ridge.**

3.3 La régression Ridge

La régression ridge est une méthode d'ajustement de modèle qui est utilisée pour analyser toutes les données qui souffrent de multicollinéarité. Cette méthode effectue une régularisation L2. Lorsque le problème de la multicollinéarité se pose, les moindres carrés ne sont pas biaisés et les variances sont importantes, ce qui fait que les valeurs prédites sont très éloignées des valeurs réelles. [6]
La fonction cout est de la forme :

$$J = \sum_{i=0}^{N} (Y_i - (\theta X_i + b))^2 + \lambda ||\theta||^2 \tag{3.1}$$

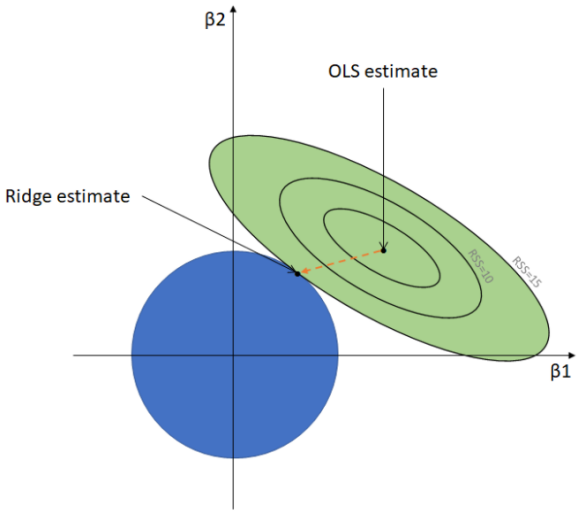

Fig. 3.1 – Regression Ridge

Pour minimiser la fonction de perte régularisée, nous devons choisir λ pour minimiser la somme de l'aire du cercle et de l'aire de l'ellipsoïde choisi par la tangence.
Notez que lorsque λ tend vers zéro, la fonction de perte régularisée devient la fonction de perte MCO. Lorsque λ tend vers l'infini, nous obtenons un modèle à intercept seul (car dans ce cas,

les coefficients de régression ridge tendent vers zéro). Nous avons alors une variance plus faible mais un biais plus important. Une critique de la régression ridge est que toutes les variables ont tendance à se retrouver dans le modèle. Le modèle ne réduit que les coefficients.

Conclusion

J'ai réussi dans ce travail à se familiariser avec les fondements mathématiques de la méthode Regression Linéaire et de l'explorer dans le cadre d'une implémentation appliquée dans l'étude des données.

En effet, le travail de documentation que j'ai fait m'a permis de faire face à la problématique de modélisation linéaire présentée dans le cadre du cours de l'optimisation et d'implémenter l'étude théorique en code exécutable.

De plus, dans la dernière partie du travail j'ai allé encore plus loin dans l'exploration de la méthode en montrant ses limitations et une méthode de correction possible par **La régularisation** qu'on va étudier dans un prochain atelier.

Souheib Ben Mabrouk.

Bibliographie

[1] Lorraine Li. Introduction to linear regression in python. https://towardsdatascience.com/introduction-to-linear-regression-in-python-c12a072bedf0.

[2] Régression linéaire. https://www.ibm.com/fr-fr/analytics/learn/linear-regression.

[3] Concepts mathématiques derrière le machine learning : la régression linéaire. https://www.actuia.com/tutoriel/concepts-mathematiques-derriere-le-machine-learning-la-regression-lineaire/.

[4] Méthode des moindres carrés ordinaire. https://fr.wikipedia.org/wiki/M%C3%A9thode_des_moindres_carr%C3%A9s_ordinaire.

[5] The classical linear regression model is good. why do we need regularization ? https://medium.com/@zxr.nju/the-classical-linear-regression-model-is-good-why-do-we-need-regularization-c89dba10c

[6] What is ridge regression ? https://www.mygreatlearning.com/blog/what-is-ridge-regression/.