**INF 8245AE : Machine Learning**

# Assignment 1

Julien SEGONNE

Septembre 2024


POLYTECHNIQUE
MONTRÉAL

# Table des matières



# Table des figures

# Introduction

Ce rapport présente mon travail sur les différentes méthodes de régression étudiées en cours de Machine Learning et leur implémentation.

# 1 Simple Linear Regression

Dans cette partie, nous implémentons une régression linéaire avec entraînement puis évaluation.

## 1.1 Préliminaires

Avant d'entraîner et d'évaluer le modèle, nous devons tout d'abord implémenter des fonctions de base :

— *data_matrix_bias* : ajoute la colonne correspondant au biais à la matrice X,

— *linear_regression_predict* : effectue la multiplication matricielle $Xw$,

— *linear_regression_optimize* : calcule le $w$ optimal grâce à l'équation suivante $\boxed{w^* = (X^T X)^{-1} X^T y}$,

— *rmse* : calcule la RMSE entre la prédiction $\hat{y}$ et les réelles valeurs $y$.

## 1.2 Résultats

Nous pouvons maintenant entrainer notre modèle sur les datasets d'entraînement et calculer une prédiction des salaires en fonction de l'expérience et de leur score au test. Les résultats sont présentés ci-dessous.
La RMSE après régression linéaire entre $\hat{y}$ et $y$ est : $\boxed{RMSE(\hat{y}, y) = 2662.938}$.

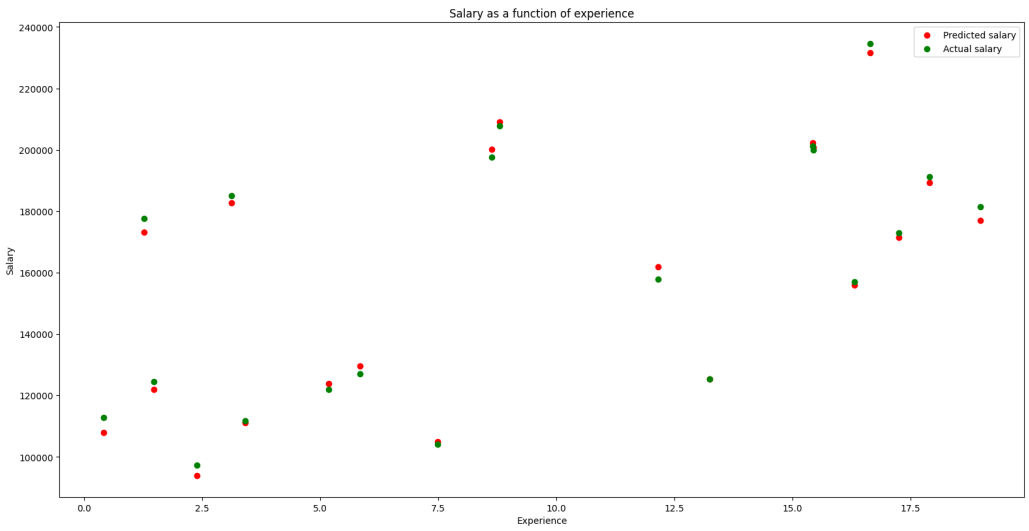


FIGURE 1 – Salaire en fonction de l'expérience

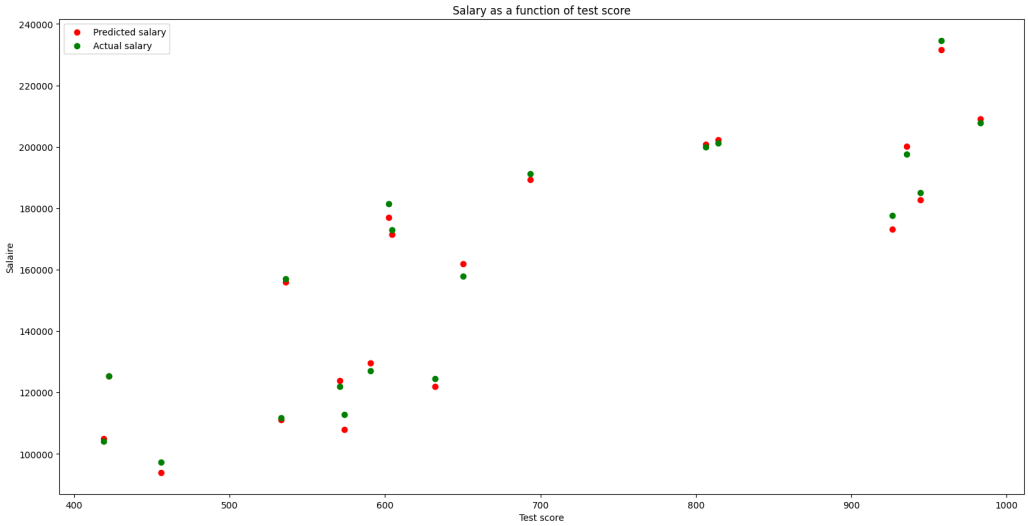


FIGURE 2 – Salaire en fonction du score au test

On constate que les prédictions sont globalement cohérentes, il n'y a pas de valeur aberrante. De plus, la RMSE est assez faible témoignant des faibles écarts entre prédiction et réalité.

# 2 Ridge Regression and Cross Validation

Dans cette partie nous implémentons une ridge regression et une validation croisée afin de trouver l'hyperparamètre $\lambda$ offrant les meilleurs résultats.

## 2.1 Closed form solution

En différenciant l'expression de la loss function, on obtient le $w$ optimal suivant :

$$\boxed{w^* = (XX^T + \lambda I)^{-1} X^T y}.$$

## 2.2 Cross Validation

Nous pouvons ainsi implémenter une validation croisée.

— Pour chaque valeur de $\lambda$, on crée différent datasets d'entraînement et de validation en sélectionnant une portion de validation différente à chaque itération dans le dataset d'entraînement initial.

— Pour chaque nouveau dataset, nous pouvons entrainer le modèle et calculer la RMSE entre la prédiction faite sur le dataset de validation et les réelles valeurs.

— Nous pouvons ensuite faire la moyenne de toutes les RMSE propres à un $\lambda$ spécifique.

— Finalement, nous choisissons le $\lambda$ qui offre la RMSE moyenne la plus faible.

Dans le cas particulier où $k\_fold = 1$, j'ai procédé comme ceci : pour chaque $\lambda$, nous séparons le dataset une seule fois (les 80% premières valeurs sont destinées à l'entraînement et les 20% restants à la validation), nous entrainons le modèle et calculons la RMSE. Enfin, nous sélectionnons le $\lambda$ qui offre la RMSE la plus faible.

## 2.3 Résultats

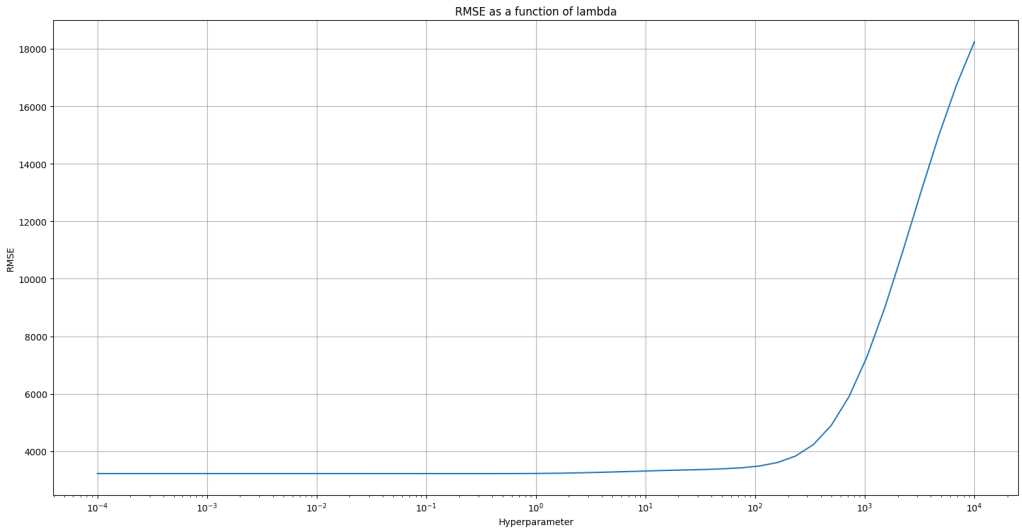


FIGURE 3 – RMSE en fonction de $\lambda$

On constate que la RMSE est faible et quasiment constante jusqu'à $\lambda = 10^3$ puis qu'elle croit très rapidement jusqu'à environ 18 000.
Pour $\lambda$ petit, le modèle est similaire à un modèle de régression linéaire classique, on obtient une RMSE proche de celle obtenue dans la partie précédente.
Pour $\lambda$ grand, la régularisation est trop forte et le modèle ne tient plus vraiment compte des données : il va principalement chercher à minimiser le membre $\lambda \cdot \|w\|_2^2$ déjà très grand. En minimisant $\lambda \cdot \|w\|_2^2$, le modèle va simplement diminuer l'influence de certains paramètres et donc ses prédictions seront moins fidèles et précises.

## 2.4 Preuve d'inversibilité

$XX^T$ est une matrice symétrique semi-définie positive donc toutes ses valeurs propres sont positives ou nulles : $\forall \mu \in spec(XX^T),\ \mu \geq 0$.
Ensuite, $\forall u,\ XX^T u = \mu u$ où $\mu \in spec(XX^T)$, on peut écrire :
$(XX^T + \lambda I)u = XX^T u + \lambda u = \mu u + \lambda u = (\mu + \lambda)u$.
Or $\lambda > 0$, donc $\boxed{\mu + \lambda > 0}$.
Ainsi, les valeurs propres de $XX^T + \lambda I$ sont toutes strictement positives donc cette matrice est inversible.

## 2.5 Influence de B

Si l'on exprime le nouveau biais sous la forme suivante : $w_0' = \frac{w_0}{B}$, en isolant le biais des autres coefficients, on obtiendrait un terme équivalent (ce ne serait pas le terme exacte car $w_0$ se trouve dans la norme) à $\frac{\lambda}{B} w_0$. Ainsi, en choisissant la constante B, nous pouvons choisir l'influence qu'a la régularisation sur le biais.
Donc avec une très grande valeur de B, $\lambda$ n'aurait que très peu d'impact sur le biais tout en gardant son influence sur les autres termes de $w$.

# 3 Full-Batch Gradient Descent

Plutôt que d'utiliser une closed-form solution pour calculer le $w$ optimal, nous allons utiliser ici la descente de gradient.

## 3.1 Préliminaires

Nous calculons dans un premier temps les gradients selon la forme de la régression (*linear* ou *ridge*). Puis nous pouvons procéder à la descente de gradient en choisissant si l'on utilise la régularisation ou non. Nous prenons en argument le learning rate et le nombre d'epochs.

## 3.2 Résultats

Lors du lancement du code, j'ai rencontré un problème de divergence : les gradients explosaient après quelques itérations et l'ordinateur n'était plus capable d'effectuer les calculs. Deux solutions s'offraient à moi :

— Imposer une borne supérieure à la valeur absolue des gradients et s'ils la dépassent, ils prennent cette valeur.

— Diminuer le learning rate afin de donner moins d'impact au gradient et de leur imposer de plus faibles valeurs.

J'ai décidé d'employer les deux solutions.

### 3.2.1 q3_3.py

Paramètres :

— $learning\_rate = 10^{-8}$

— $num\_epochs = 1000$

— $ridge\_hyperparameter = 0.1$

— $sup\_gradient = 10^{8}$

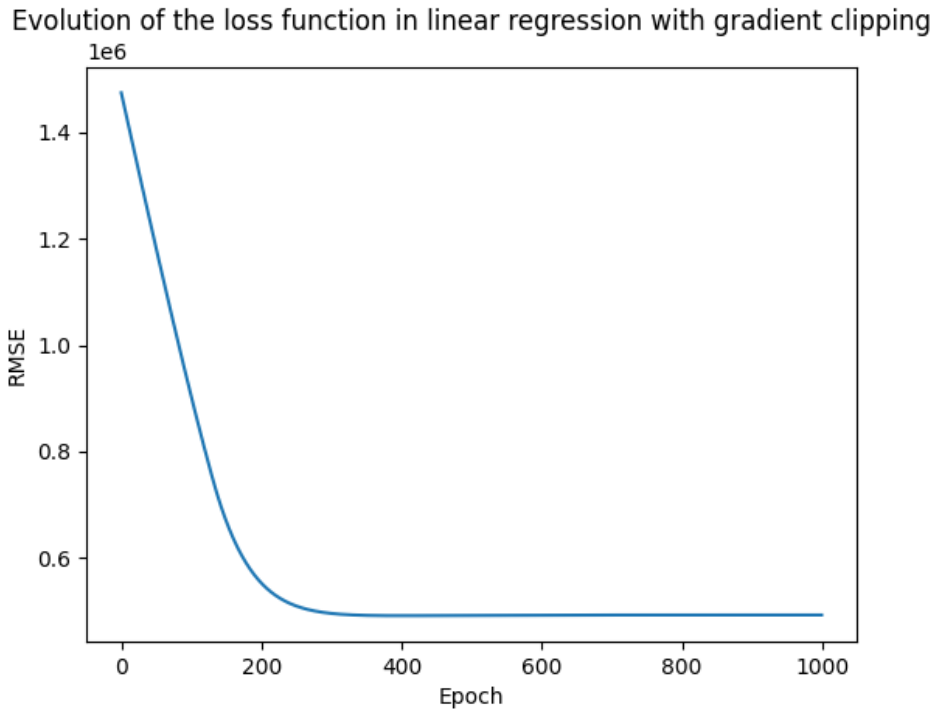


FIGURE 4 – Evolution de la fonction de perte (régression linéaire)

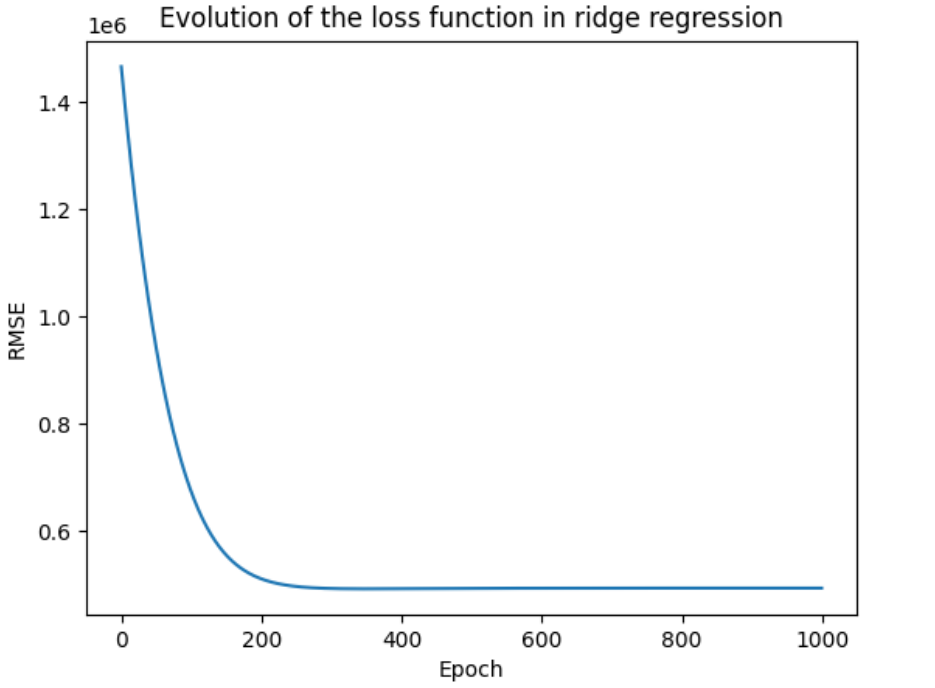


FIGURE 5 – Evolution de la fonction de perte (ridge regression)

On observe sur ces deux graphiques que dès les premières itérations, il y a une diminution drastique de la RMSE témoignant de l'ajustement rapide du modèle entre l'initialisation et les premières itérations, les gradients sont assez élevés car le modèle est loin de l'optimum, les sauts entre les w(i) et w(i+1) sont grands. Ensuite le modèle ne s'améliore plus vraiment et la RMSE

converge, en se rapprochant de l'optimum, les gradients deviennent beaucoup plus faibles et les sauts sont plus petits, la décroissance de la RMSE est donc de moins en moins significative.

### 3.2.2 q3_4.py

Paramètres :

— $learning\_rates$ = [0.0001, 0.001, 0.01, 0.05, 0.1, 0.2, 0.5, 1.0]

— $num\_epochs$ = 1000

— $ridge\_hyperparameter$ = 0.1

— $sup\_gradient$ = $10^3$

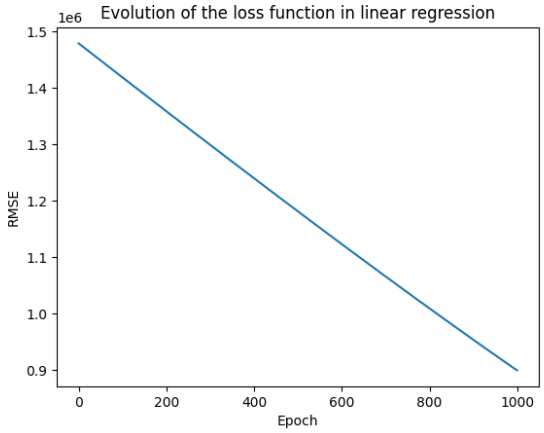


FIGURE 6 – learning_rate = 0.0001

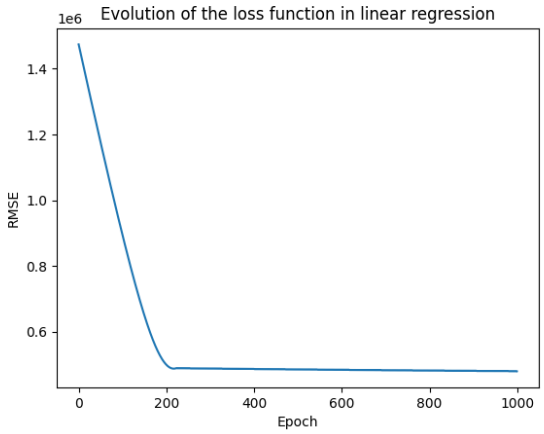


FIGURE 7 – learning_rate = 0.001

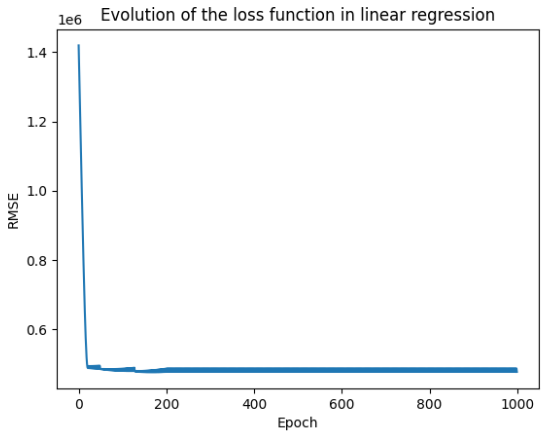


FIGURE 8 – learning_rate = 0.01

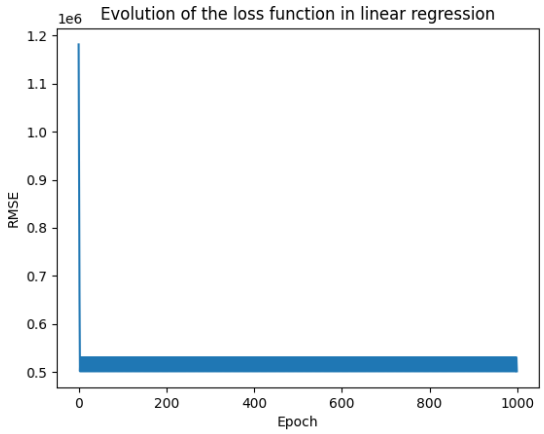


FIGURE 9 – learning_rate = 0.05

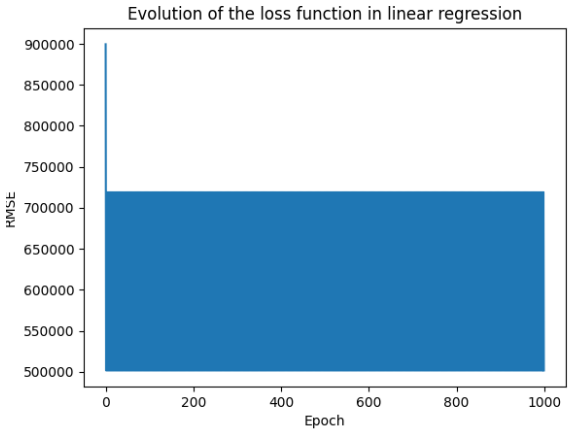


FIGURE 10 – learning_rate = 0.1

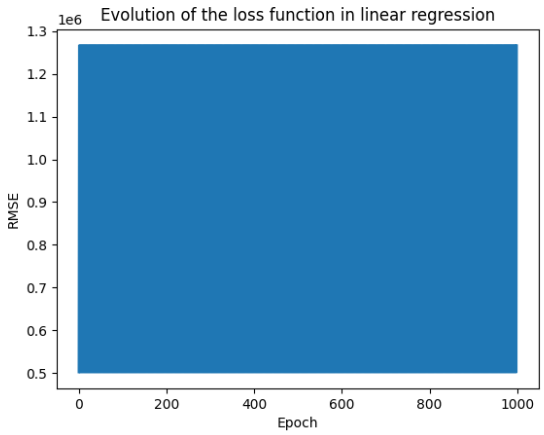


FIGURE 11 – learning_rate = 0.2

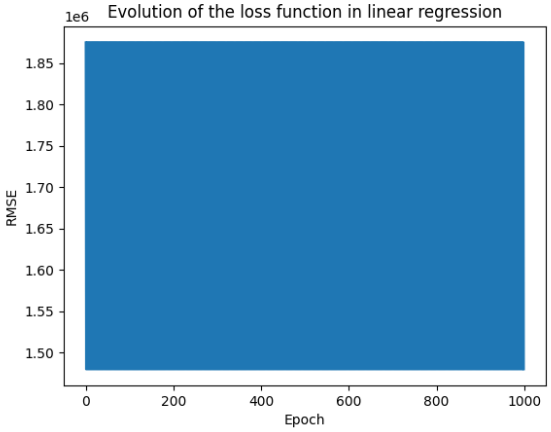


FIGURE 12 – learning_rate = 0.5

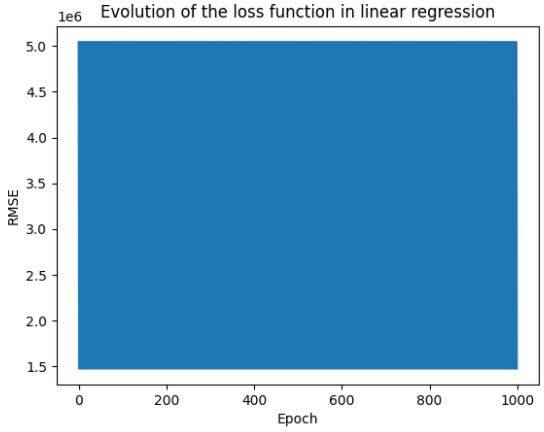


FIGURE 13 – learning_rate = 1.0

On constate que pour le learning rate le plus faible la décroissance est linéaire : cela est dû à la limite imposée dans mon implémentation, le premier gradient est toujours très élevé (donc il prend la valeur de la borne supérieur) or le learning rate étant faible, il n'impacte que très

peu la nouvelle valeur de w donc le gradient suivant sera encore très grand et aura encore la même valeur (car supérieur à la borne supérieure). Ainsi, les performances offertes ne sont pas optimales.
En revanche, pour les learning rate suivants (de 0.001 à 0.05), on observe une convergence de plus en plus rapide, ce qui est explicable par le fait que le gradient a de plus en plus de poids dans l'expression du $w$ suivant. Ainsi, le modèle s'améliore de plus en plus rapidement.
Enfin, pour de grands learning rate, le graphique n'est plus vraiment exploitable. Je suppose que la valeur de la RMSE fluctue énormément donnant un tel graphique. Ceci doit être dû au fait que le gradient ayant un trop grand impact sur w, son signe alterne car il dépasse à chaque fois le minimum que l'on cherche à atteindre. Nous obtenons donc de grandes fluctuations brouillant le graphique.

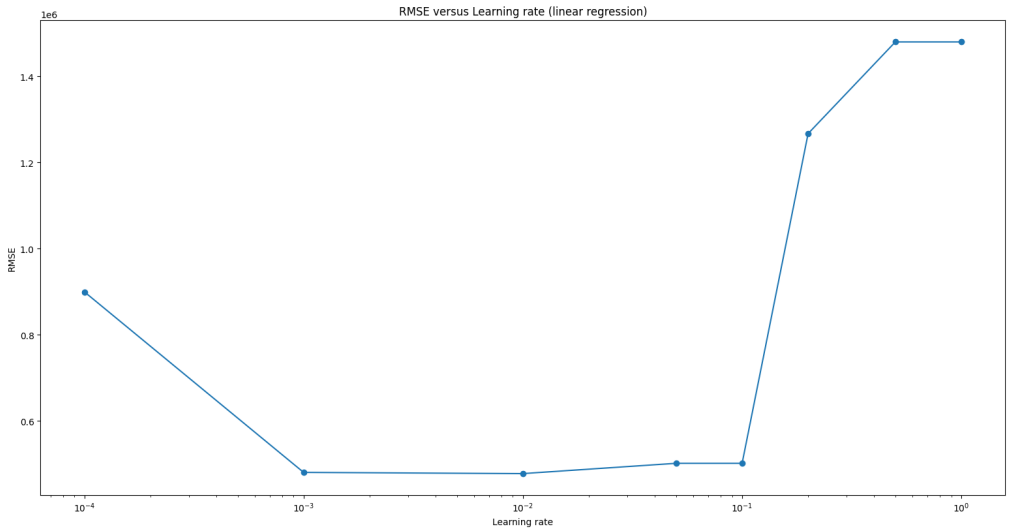


FIGURE 14 – RMSE versus Learning rate (linear regression)

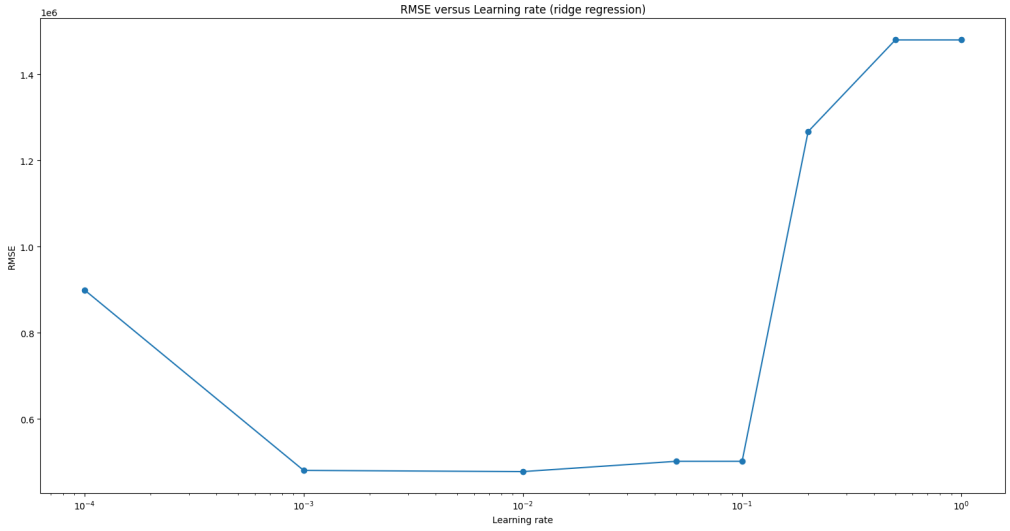


FIGURE 15 – RMSE versus Learning rate (ridge regression)

On constate sur ces deux graphiques qu'un trop faible ou trop fort learning rate n'offre pas les meilleurs résultats :

— Pour le plus faible learning rate, le gradient n'a pas assez de poids lorsqu'on calcule le nouveau $w$, ainsi l'apprentissage est trop lent et en 1000 epochs, le modèle n'a pas le temps de se rapprocher assez près de la valeur optimale (minimum de la RMSE).

— Pour les plus grands learning rate, le gradient a un poids trop élevé dans l'expression du nouveau $w$, ainsi la valeur optimale n'est jamais atteinte car elle est dépassée et le gradient change de signe à chaque itération, donc à la fin des itérations on ne s'est pas vraiment rapproché du minimum (on peut imaginer en deux dimensions une parabole avec un minimum qui n'est jamais atteint car on fait des grands sauts d'un côté et de l'autre de la parabole).

En conclusion, les meilleurs learning rate sont ceux pour lesquels le gradient a assez d'impact dans le calcul du nouveau $w$ pour converger rapidement vers le minimum dans le temps imparti mais pas un impact trop élevé afin que l'on ne fasse pas de trop grands sauts qui fassent converger trop lentement (des sauts n'offrant pas de progression significative vers l'optimum) ou bien diverger le modèle (avec des gradients qui ne cessent de grandir).