Notes sur les techniques de régularisation en machine learning

Franck Jeannot

Montréal, Canada, Juin 2023, AA795, v1.0

---

1. Introduction

La **régularisation** est une technique essentielle en apprentissage automatique, ou « machine learning ». Elle sert à éviter un problème très courant appelé "sur-apprentissage", sur-ajustement [1] ou **"overfitting"** en anglais. Lorsqu'on entraîne un modèle sur des données, on souhaite que ce modèle puisse généraliser son apprentissage à de nouvelles données. Cependant, sans régularisation, un modèle peut trop se concentrer sur les détails et le bruit dans l'ensemble de données d'apprentissage, au point de "mémoriser" ces données plutôt que d'apprendre les tendances générales. En conséquence, il se comporte mal lorsqu'il rencontre des données inédites. En bref, la régularisation est une méthode pour empêcher le sur-apprentissage en imposant une certaine forme de pénalité sur la complexité du modèle. Plus précisément, elle décourage l'apprentissage de modèles trop complexes en ajoutant un coût à leur fonction d'objectif (la fonction que le modèle essaie de minimiser pendant l'apprentissage).

La régularisation est une technique qui empêche un modèle de "sur-ajuster" en lui ajoutant des informations supplémentaires. Il s'agit d'une forme de régression qui rétrécit les estimations des coefficients vers zéro [2]. Cette technique apporte de légères modifications à l'algorithme d'apprentissage de sorte que le modèle se généralise mieux, améliorant les performances du modèle sur des données inédites ou "invisibles" [2].

2. Des techniques standard

Il existe plusieurs techniques de régularisation couramment utilisées pour contrôler la complexité des modèles d'apprentissage automatique.
La "régularisation L1", aussi appelée régularisation **Lasso** (3) (26), est une technique d'optimisation qui vise à minimiser la complexité du modèle en ajoutant une pénalité égale à la **somme** absolue des coefficients du modèle (c'est-à-dire la norme L1 des coefficients) à la fonction de coût. Elle favorise des solutions parcimonieuses, c'est-à-dire avec beaucoup de coefficients nuls.
La "régularisation L2", ou régression régression **crête** ou ***Ridge*** (18) [3] [4] [5], est une autre technique d'optimisation qui ajoute une pénalité égale à la **somme des carrés** des coefficients du modèle (c'est-à-dire la norme L2 des coefficients) à la fonction de coût. Elle a tendance à réduire les coefficients sans les rendre nuls. Ces deux techniques aident à prévenir le surapprentissage.
Au final, la régression Lasso et la régression Ridge cherchent toutes deux à minimiser la somme des carrés des résidus, mais elles diffèrent dans la manière dont elles ajoutent une pénalité aux coefficients du modèle pour éviter le surapprentissage. En résumé, alors que la régression Ridge réduit la magnitude des coefficients, la régression Lasso peut les rendre nuls, conduisant à un modèle plus simple et plus interprétable.

3. La notation L (L1, L2)

Les espaces L^p sont une classe d'espaces vectoriels normés introduits par **Henri Lebesgue** [6], et sont des exemples d'espaces de Banach, nommés d'après **Stefan Banach** [7] qui a formalisé leur étude. Les normes dans les espaces L^p sont des généralisations de la **norme euclidienne** (ou L^2) et de la norme de **Manhattan** 4 (ou L^1), qui sont respectivement les cas $p = 2$ et $p = 1$ de la norme L^p. Ces normes sont basées sur les travaux de **Hermann Minkowski** [8], [9] sur les géométries à p-normes. Les régressions L1 et L2 tirent leur nom de l'utilisation de ces normes comme termes de régularisation : la régression L1 utilise la norme L^1 (somme des valeurs absolues des coefficients), et la régression L2 utilise la norme L^2 (somme des carrés des coefficients). La notation L fait référence aux normes de Minkowski et aux espaces L^p. Ces derniers généralisent simplement la notion de distances de taxicab et euclidiennes à $p > 0$ dans l'expression suivante :

$$\|x\|_p = \left(|x_1|^p + |x_2|^p + \ldots + |x_n|^p\right)^{\frac{1}{p}}$$

Il est important de noter que seul $p \geq 1$ définit une distance métrique ; $0 < p < 1$ ne satisfait pas l'inégalité triangulaire, donc ce n'est pas une distance selon la plupart des définitions.

4. Le terme "Lasso"

Le terme **"Lasso"** est un acronyme pour "Least Absolute Shrinkage and Selection Operator". Il a été introduit par Robert Tibshirani en 1996 dans un article intitulé "Regression shrinkage and selection via the lasso" [10].

Le terme "Lasso" a été choisi pour deux raisons. Premièrement, comme mentionné, c'est un acronyme qui décrit la méthode. Deuxièmement, le terme "lasso" en anglais désigne un lasso, qui est un outil utilisé pour attraper des objets à distance en les entourant. De manière similaire, la **régression Lasso** (26) "attrape" un sous-ensemble de prédicteurs (c'est-à-dire qu'elle effectue une sélection de caractéristiques) tout en réduisant les coefficients de régression vers zéro (c'est-à-dire qu'elle effectue une contraction ou un "shrinkage").

5. Techniques de régularisation L1 et L2

Parmi les techniques standard les plus couramment utilisées sont la régularisation L1 et L2. La régularisation L1 à tendance à pousser les poids des caractéristiques moins importantes vers zéro, effectuant ainsi une sorte de sélection de caractéristiques. D'autre part, la régularisation L2 réduit simplement l'importance de certaines caractéristiques mais ne les élimine pas complètement.

Les techniques de régularisation L1 (Lasso [10]) et L2 (Ridge (18)) sont des méthodes de régularisation traditionnelles utilisées dans de nombreux types de modèles d'apprentissage automatique, y compris les réseaux de neurones. Elles ajoutent un terme de pénalité à la **fonction de coût** [11] du modèle qui est proportionnel à la taille des poids du modèle. Cela encourage le modèle à avoir de petits poids, ce qui peut rendre le modèle plus simple et moins susceptible de surajuster les données d'entraînement.

6. Ajustement de données (Data fitting)

Le **"Data fitting"** est un processus qui consiste à créer un modèle mathématique qui s'adapte le mieux possible à un ensemble de données. Cet ajustement est généralement effectué en minimisant la différence entre les valeurs prédites

par le modèle et les valeurs réelles des données. Les techniques d'ajustement de données sont largement utilisées en science des données et en apprentissage automatique pour créer des modèles prédictifs. Le processus consistant à tracer une série de points de données et à dessiner la ligne d'ajustement la plus appropriée pour comprendre la relation entre les variables est aussi appelé **ajustement de données** (3).

7. Modèle de régression

Un **modèle de régression** [1] est un type de modèle statistique qui est utilisé pour prédire une variable cible en fonction d'une ou plusieurs variables indépendantes, en supposant une relation spécifique entre elles. Ainsi **la régression linéaire** (11) peut être vue comme une technique d'*ajustement de données* (3) qui utilise une fonction linéaire pour modéliser la relation entre les variables.

8. La norme L1

La norme L1 , également appelée **norme Manhattan** (4) ou **norme taxicab** (4) [2], est une mesure de la taille d'un vecteur. Pour un vecteur w dans un espace à N

1. Le terme "régression" a été utilisé pour la première fois en statistique par **Francis Galton**, un cousin de Charles Darwin, dans le cadre de ses études sur l'hérédité. Galton a utilisé le terme "régression" pour décrire le phénomène selon lequel les traits des parents "régressent" vers la moyenne dans la génération suivante. Par exemple, il a observé que les enfants de parents très grands ont tendance à être plus petits que leurs parents, se rapprochant ainsi de la taille moyenne de la population. Le terme "régression" a ensuite été adopté dans le domaine de l'analyse statistique pour décrire les méthodes qui cherchent à modéliser la relation entre une variable dépendante et une ou plusieurs variables indépendantes. Ainsi, un "modèle de régression" est un modèle qui utilise la régression pour prédire une variable à partir d'une ou plusieurs autres.

2. Les termes "norme Manhattan" et "norme taxicab" sont des métaphores géographiques pour la norme L1. Norme Manhattan : Ce nom provient de la structure en grille des rues de Manhattan, un arrondissement de la ville de New York. Si vous voulez vous déplacer d'un point à un autre dans Manhattan, vous devez vous déplacer le long des rues, qui sont disposées en grille. Par conséquent, la distance que vous parcourez (en supposant que vous ne pouvez pas couper à travers les bâtiments) est la somme des distances horizontale et verticale, ce qui correspond à la norme L1. Norme taxicab : Ce nom a la même origine. Il fait référence à la distance qu'un taxi doit parcourir dans une ville où les rues sont disposées en grille. Encore une fois, la distance est la somme des distances horizontale et verticale, car le taxi doit suivre les rues et ne peut pas couper à travers les bâtiments. Ces deux noms sont des façons intuitives de comprendre la norme L1, qui est la somme des valeurs absolues des différences entre les coordonnées de deux points. Le terme "norme taxicab" a été introduit par Hermann Minkowski, et il est parfois appelé "distance de Minkowski" pour p=1.

dimensions, la norme L1 est définie comme la somme des valeurs absolues de ses composantes. Mathématiquement, elle est définie comme suit :

$$\|w\|_1 = \sum_{j=1}^{N} |w_j| \tag{1}$$

où w_j est la j-ième composante du vecteur w. La notation $|w_j|$ représente la valeur absolue de la j-ième composante du vecteur w. La somme $\sum_{j=1}^{N}$ signifie que l'on additionne ces valeurs absolues pour toutes les composantes du vecteur, de la première ($j = 1$) à la N-ième ($j = N$). Dans le contexte de la régularisation L1, cette norme est utilisée pour mesurer la "taille" des coefficients de poids dans un modèle de machine learning. La régularisation L1 a tendance à faire tendre certains des poids exactement vers zéro, ce qui peut être utile pour la sélection de caractéristiques.

Par exemple, la norme L1 du vecteur $w = [1, -2, 3, -4, 5]$ est calculée en prenant la somme des valeurs absolues de chaque élément du vecteur. En décomposant, cela donne :

$$\|w\|_1 = \sum_{j=1}^{N} |w_j| = |1| + |-2| + |3| + |-4| + |5| = 1 + 2 + 3 + 4 + 5 = 15$$

Donc, la norme L1 de ce vecteur est 15.

Exemple d'implémentation python :

```python
import numpy as np
w = np.array([1, -2, 3, -4, 5])
L1_norm = np.sum(np.abs(w))
print("La norme L1 du vecteur est :", L1_norm)
```

```
La norme L1 du vecteur est : 15
```

9. Fonction de coût régularisée L1

L'équation plus bas est celle de la **fonction de coût régularisée L1 (ou LASSO)** [3] :

Fonction de coût régularisée L1 (ou LASSO)

$$\tilde{J}(\theta, X, y) = J(\theta, X, y) + \lambda \|w\|_1 = J(\theta, X, y) + \lambda \sum_{j=1}^{N} |w_j| \tag{2}$$

L'équation se compose de deux termes : la fonction de coût originale $J(\theta, X, y)$ et le terme de régularisation $\lambda \|w\|_1$.

A noter que le tilde ($\tilde{}$) est un symbole utilisé pour indiquer une approximation ou une équivalence. Dans le contexte de l'équation de la fonction de coût régularisée L1, $\tilde{J}(\theta, X, y)$, le tilde indique que cette fonction est une version modifiée ou régularisée de la fonction de coût originale $J(\theta, X, y)$. En d'autres termes, $\tilde{J}(\theta, X, y)$ est la fonction de coût $J(\theta, X, y)$ à laquelle on a ajouté un terme de régularisation.

La fonction de coût $J(\theta, X, y)$ [4] mesure à quel point le modèle prédit mal les données d'entraînement. Par exemple, dans la régression linéaire, $J(\theta, X, y)$ est la somme des carrés des erreurs de prédiction.

Le terme de régularisation $\lambda \|w\|_1$ [5] pénalise les grands poids dans le modèle. Le paramètre λ contrôle l'importance de cette pénalité. Si λ est grand, les poids seront fortement pénalisés, ce qui conduira à un modèle plus simple. Si λ est petit, les poids seront moins pénalisés, ce qui conduira à un modèle plus complexe. L'objectif de l'apprentissage est de trouver les paramètres θ qui minimisent cette

3. La fonction de coût régularisée L1, également connue sous le nom de Lasso (Least Absolute Shrinkage and Selection Operator) [10], a été introduite par Robert Tibshirani en 1996. Cette méthode a été développée comme une modification de la méthode de régression Ridge 18 (régularisation L2) pour permettre la sélection de variables. En ajoutant une pénalité L1 à la fonction de perte, certaines des estimations des coefficients de régression peuvent être réduites à zéro, ce qui permet d'exclure les variables correspondantes du modèle.

4. J de theta, X, y

5. lambda fois la norme L1 de w

fonction de coût régularisée.

Voici un exemple de calcul avec $\lambda = 0.1$:

1. On calcule la fonction de coût $J(\theta, X, y)$:

$$J(\theta, X, y) = \sum_{i=1}^{N} (y_i - \hat{y}_i)^2 \tag{3}$$

où y_i sont les valeurs réelles, $\hat{y}_i$ sont les valeurs prédites par le modèle, et N est le nombre total d'échantillons.

On a :

```
J = \sum_{i=1}^{N} (\hat{y}_i - y_i)^2
converti en python :
J = np.sum((reg.predict(X) - y) ** 2)
```

2. On calcule la norme L1 des coefficients du modèle :

$$\|w\|_1 = \sum_{j=1}^{p} |w_j| \tag{4}$$

où w_j sont les coefficients du modèle, et p est le nombre total de caractéristiques.

3. Finalement on calcule la fonction de coût régularisée L1 :

$$\tilde{J}(\theta, X, y) = J(\theta, X, y) + \lambda \|w\|_1 \tag{5}$$

où λ est le paramètre de régularisation.

Dans l'exemple précédent, la valeur de λ est définie à 0.1 et la norme L1 des coefficients du modèle, $\|w\|_1$, est calculée comme la somme des valeurs absolues des coefficients. Dans cet exemple, le modèle a trois coefficients, tous égaux à 1. Par conséquent, la norme L1 des coefficients est $|1| + |1| + |1| = 3$.

Lorsqu'on multiplie λ et $\|w\|_1$ ensemble pour obtenir $\lambda\|w\|_1$, on a $0.1 \times 3 = 0.3$. C'est pourquoi $\lambda\|w\|_1$à la valeur 0.3 dans cet exemple.

Dans l'exemple précédent, la fonction de coût régularisée L1 est donc calculée comme suit :

$$\tilde{J}(\theta, X, y) = J(\theta, X, y) + \lambda \|w\|_1 = 0 + 0.1 \times 3 = 0.3 \tag{6}$$

On crée alors un programme Python qui implémente une fonction de coût régularisée L1 équivalente. Dans cet exemple, on utilise une régression linéaire simple (11) comme fonction de coût de base J. Dans ce code, on utilise la méthode **fit** de **LinearRegression** pour ajuster le modèle aux données, puis on calcule la fonction de coût $J(\theta, X, y)$ en utilisant la méthode **predict** pour obtenir les prédictions du modèle et en calculant la somme des carrés des résidus. Enfin, on la pénalité de régularisation L1, qui est le produit de lambda et de la somme des valeurs absolues des coefficients du modèle :

```python
import numpy as np
from sklearn.linear_model import LinearRegression

# Définir les données
X = np.array([[1, 1], [1, 2], [2, 2], [2, 3]])
y = np.dot(X, np.array([1, 2])) + 3

# Créer une instance de LinearRegression
reg = LinearRegression().fit(X, y)

# Calculer la fonction de coût J(theta, X, y)
J = np.sum((reg.predict(X) - y) ** 2)

# Définir le paramètre de régularisation lambda
lambda_ = 0.1

# Calculer la fonction de coût régularisée L1
J_L1 = J + lambda_ * np.sum(np.abs(reg.coef_))

print("La fonction de coût régularisée L1 est :", J_L1)
```

```
La fonction de coût régularisée L1 est : 0.29999999999999993
(la valeur exacte est 0.3 : l'erreur est due aux erreurs de précision
numérique en python)
```

10. Notion d'argument de minimisation

Pour signifier (écrire en termes mathématiques) "l'argument qui minimise X", on écrit :

arg min (X)

$$\arg\min \ X \tag{7}$$

En d'autres termes, cela indique la valeur de l'argument pour laquelle la fonction X atteint sa valeur minimale.

L'origine de cet usage remonte à l'analyse mathématique, où il est souvent nécessaire de trouver les valeurs des paramètres qui minimisent une fonction donnée. Par exemple, en optimisation, on cherche souvent à minimiser une fonction de coût qui mesure l'erreur entre les valeurs prédites et les valeurs observées.

Voici un exemple simple pour illustrer cette notion. Supposons que nous avons une fonction $f(x) = x^2 + 2x + 1$. Nous voulons trouver la valeur de x qui minimise cette fonction. En utilisant l'expression $\arg\min \ X$, nous pouvons écrire cela comme suit :

arg min f(x)

$$x = \arg\min f(x) \tag{8}$$

En résolvant [6] cette équation, on trouve que la valeur de x qui minimise $f(x)$ est $x = -1$.

6. Pour trouver la valeur de x qui minimise la fonction $f(x) = x^2 + 2x + 1$, nous pouvons utiliser la méthode de la dérivation. La dérivée première de $f(x)$ est $f'(x) = 2x + 2$. Pour trouver les points critiques de $f(x)$, nous résolvons l'équation $f'(x) = 0$, ce qui donne $x = -1$. En évaluant la dérivée seconde de $f(x)$ en ce point, nous trouvons que $f''(-1) = 2 > 0$, ce qui indique que $x = -1$ est un minimum local de $f(x)$. Comme $f(x)$ est une fonction quadratique, elle a un seul minimum, donc $x = -1$ est également le minimum global de $f(x)$.

11. Régression Lasso

D'après (Tibshirani, 1996)[10], la régression **Lasso** (Least Absolute Shrinkage and Selection Operator), utilise la régularisation L1 pour réduire la complexité du modèle en ajoutant un terme de pénalité à la fonction de coût.

Une équation de la formulation du problème de la régression Lasso est :

Régression LASSO

$$(\hat{\alpha}, \hat{\boldsymbol{\beta}}) = \arg\min \left\{ \sum_{i=1}^{N} \left(y_i - \alpha - \sum_j \beta_j x_{ij} \right)^2 \right\} \quad \text{avec} \sum_j |\beta_j| \leqslant t. \tag{9}$$

La régression Lasso cherche à *minimiser* (9) la **somme des carrés des résidus** dits **RSS** (21) (c'est-à-dire la différence entre les valeurs observées y_i et les valeurs prédites $\alpha + \sum_j \beta_j x_{ij}$), tout en limitant la somme des valeurs absolues des coefficients β_j à être inférieure ou égale à un certain seuil t. Elle utilise l'**algorithme de descente de gradient coordonnée** (16).

Cela encourage une solution parcimonieuse où certains coefficients β_j sont exactement zéro, ce qui peut aider à la sélection des variables dans un modèle de régression.

L'objectif de cette méthode est de trouver les valeurs de α et $\boldsymbol{\beta}$ qui minimisent la somme des carrés des résidus, c'est-à-dire la différence entre les valeurs observées y_i et les valeurs prédites tout en satisfaisant la contrainte que la somme des valeurs absolues des coefficients β_j soit inférieure ou égale à un seuil t. Cette contrainte a pour effet de réduire la complexité du modèle en forçant certains coefficients β_j à être égaux à zéro, ce qui équivaut à supprimer les variables correspondantes du modèle.

Valeurs prédites

$$\alpha + \sum_j \beta_j x_{ij} \tag{10}$$

12. Régression linéaire

Dans une régression linéaire, l'équation est de la forme :

Valeurs prédites

$$y = \alpha + \beta_1 x_1 + \beta_2 x_2 + ... + \beta_n x_n + \epsilon \tag{11}$$

où y est la variable dépendante, $x_1, x_2, ..., x_n$ sont les variables indépendantes, α est l' **intercept**, $\beta_1, \beta_2, ..., \beta_n$ sont les coefficients des variables indépendantes, et ϵ est l'erreur.

L'**intercept** α est le point où la ligne de régression croise l'axe des ordonnées (c'est-à-dire lorsque toutes les variables indépendantes sont égales à zéro).

En ajoutant une colonne de 1 à la matrice des variables indépendantes X, nous permettons à notre modèle d'apprendre l'intercept α comme un autre coefficient. Cela simplifie l'équation de régression en $y = X\beta + \epsilon$, où X est maintenant la matrice augmentée des variables indépendantes et β est le vecteur des coefficients, y compris l'intercept.

Exemple de code Python qui génère 20 points aléatoires et effectue une régression linéaire sur ces points. Le graphe affiche les points en rouge et la ligne de régression en bleu :

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression

# Générer 20 points aléatoires
np.random.seed(0)
X = np.random.rand(20, 1)
y = 2 + 3 * X + np.random.rand(20, 1)

# Créer et entraîner un modèle de régression linéaire
model = LinearRegression()
model.fit(X, y)
```

```python

# Prédire les valeurs de y pour les valeurs de X
y_pred = model.predict(X)

# Afficher les points et la ligne de régression
plt.scatter(X, y, color='red')
plt.plot(X, y_pred, color='blue')
plt.show()

```

Dans ce code, on utilise la bibliothèque sklearn pour créer et entraîner un modèle de régression linéaire. On utilise ensuite ce modèle pour prédire les valeurs de y pour les valeurs de X, et on affiche les points et la ligne de régression sur un graphique.

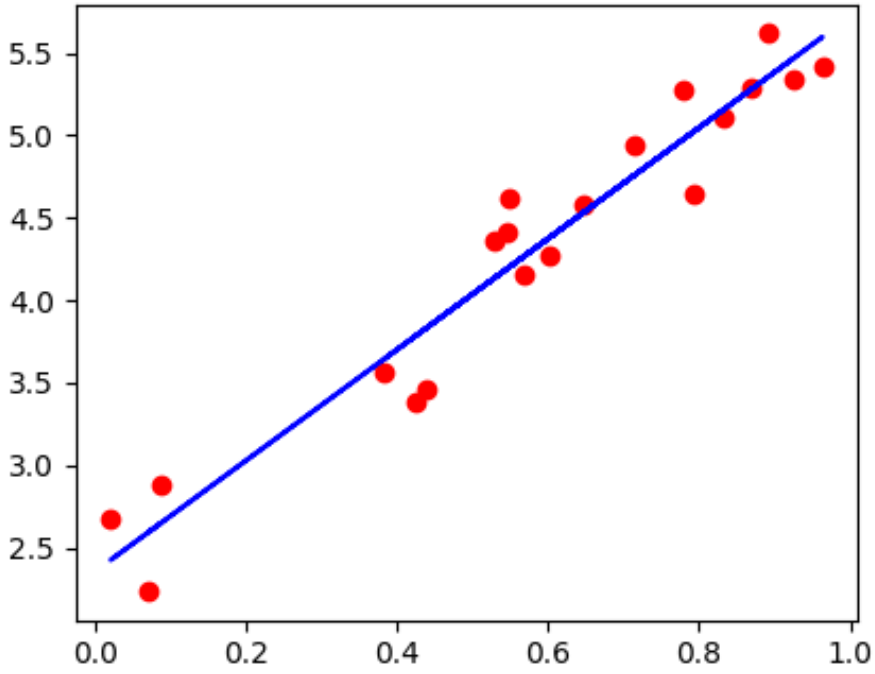

Figure (1) – Régression linéaire sur 20 points (sklearn)

Pour visualiser l'intercept et afficher les valeurs des coefficients ont peut optimiser le code python :

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression

# Générer 20 points aléatoires
np.random.seed(0)
X = np.random.rand(20, 1)
y = 2 + 3 * X + np.random.rand(20, 1)
# Créer et entraîner un modèle de régression linéaire
model = LinearRegression()
model.fit(X, y)
# Prédire les valeurs de y pour les valeurs de X
y_pred = model.predict(X)
# Afficher les coefficients et l'intercept
print("Coefficients:", model.coef_)
print("Intercept:", model.intercept_)

# Afficher les points et la ligne de régression
plt.scatter(X, y, color='red', label='Data points')
plt.plot(X, y_pred, color='blue',
         label=f'Regression line: y = {model.coef_[0][0]:.2f}x + {model.intercept_[0]:.2f}')

# Ajouter une ligne verticale qui coupe
#l'axe X au niveau de l'intercept
x_intercept = -model.intercept_/model.coef_
plt.axvline(x=x_intercept, color='green', linestyle='--',
            label=f'x-intercept: {x_intercept[0][0]:.2f}')

# Ajuster les limites de l'axe des y
#pour visualiser l'intersection de la régression linéaire avec l'axe des x
plt.ylim(min(0, y.min()), y.max())

# Afficher la légende
plt.legend()

plt.show()
```

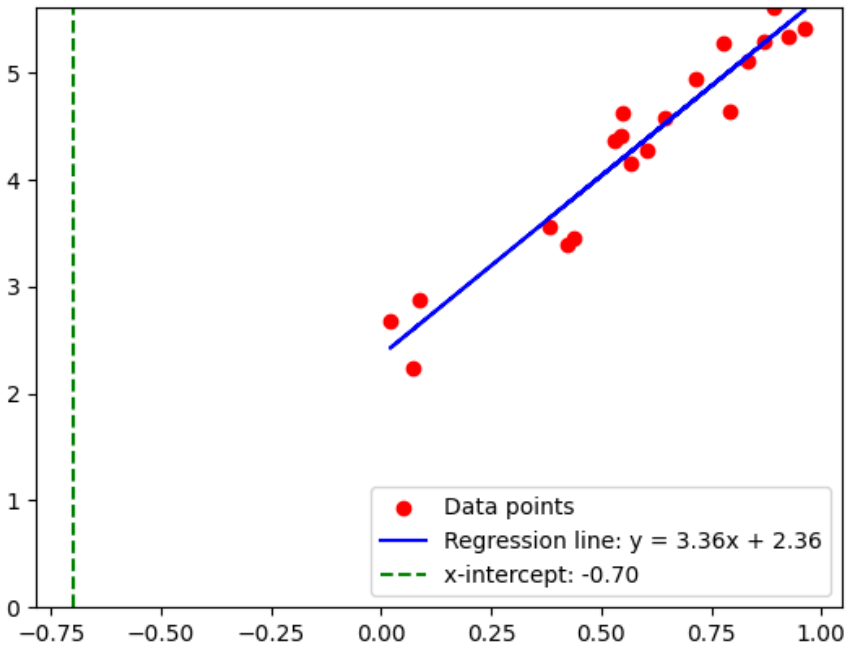

Figure (2) – Régression linéaire avec visualisation de l'intercept et des coefficients (Intercept alpha : -0.7)

13. La résolution de l'équation de la régression Lasso

La résolution de l'équation de la régression Lasso sans utiliser une bibliothèque comme **sklearn (python)** peut être assez complexe car elle nécessite l'implémentation d'un algorithme d'optimisation, comme l'**algorithme de descente de gradient coordonnée** (16) ou l'**algorithme de descente de gradient stochastique**.

En python on peut résoudre une version simplifiée de l'équation pour une régression linéaire (11) ordinaire (sans la contrainte L1) en utilisant uniquement le package **numpy**. Voici un exemple de code Python qui effectue une **régression linéaire ordinaire** [7] sur un ensemble de données synthétiques [8] :

```python
import numpy as np

# Générer un ensemble de données de régression
np.random.seed(0)
X = np.random.rand(100, 1)
y = 2 + 3 * X + np.random.rand(100, 1)
# Ajouter une colonne de 1 à X pour le biais
X_b = np.c_[np.ones((100, 1)), X]
# Calculer les coefficients beta en utilisant l'équation normale
beta_hat = np.linalg.inv(X_b.T.dot(X_b)).dot(X_b.T).dot(y)
print("Intercept:", beta_hat[0])
print("Coefficients:", beta_hat[1:])
```

```
python lasso_non_l1_simplified.py
Intercept: [2.55808002]
Coefficients: [[2.93655106]]
```

7. méthode statistique qui modélise la relation linéaire entre une variable dépendante et une ou plusieurs variables indépendantes

8. c'est a dire généré artificiellement

Dans ce code, on génére un ensemble de données de régression synthétiques où la véritable relation est $y = 2 + 3x + \epsilon$, où ϵ est un bruit aléatoire. Ensuite, on ajoute une colonne de 1 à 'X' pour le biais (c'est-à-dire l'intercept α). Enfin, on calcule les coefficients β en utilisant l'équation normale, qui est une solution analytique pour le problème de la **régression linéaire ordinaire**. Il faut noter que ce code ne résout pas exactement l'équation de la régression LASSO car il ne prend pas en compte la **contrainte L1**. La résolution de l'équation de la régression Lasso nécessite un algorithme d'optimisation plus complexe.

14. L'algorithme de descente de gradient standard

L'algorithme de descente de gradient standard est une méthode d'optimisation [9] qui vise à trouver le minimum local d'une fonction. Il fonctionne en ajustant itérativement un ensemble de paramètres dans la direction opposée au gradient de la fonction à un taux d'apprentissage donné. Cela permet de réduire progressivement la valeur de la fonction jusqu'à atteindre un minimum.

15. Algorithme de descente de gradient coordonnée

L' **algorithme de descente de gradient coordonnée** [12] [13] est une méthode d'optimisation utilisée pour minimiser une fonction objectif. Au lieu de mettre à jour tous les paramètres simultanément comme dans la descente de gradient standard (16), cet algorithme met à jour un seul paramètre à la fois, en gardant tous les autres paramètres fixes. Il sélectionne le paramètre à mettre à jour en fonction d'une règle spécifique, par exemple, le paramètre qui réduira le plus la fonction objectif. Cet algorithme est particulièrement utile pour les problèmes de grande dimension et pour les problèmes où la fonction objectif est séparable, c'est-à-dire qu'elle peut être écrite comme une somme de fonctions, chacune dépendant d'un seul paramètre.

L'algorithme de descente de gradient coordonnée est une méthode d'optimisation qui vise à minimiser une fonction objectif en mettant à jour une seule coordonnée à la fois. Pour une fonction objectif $f(x)$, où x est un vecteur de paramètres, l'algorithme de descente de gradient coordonnée peut être décrit par les formules mathématiques suivantes :

9. Augustin-Louis Cauchy a fait de nombreuses contributions importantes aux mathématiques, y compris le développement de la méthode de Cauchy pour la résolution de problèmes d'optimisation, qui est une précurseur de l'algorithme de descente de gradient.

Pour chaque itération t jusqu'à la convergence :
Pour chaque j dans $1, 2, ..., d$:

L'algorithme de descente de gradient coordonnée

$$x_j^{(t+1)} = \arg\min_{z \in \mathbb{R}} f(x_1^{(t+1)}, ..., x_{j-1}^{(t+1)}, z, x_{j+1}^{(t)}, ..., x_d^{(t)}) \tag{12}$$

Dans ces formules, d est la dimension de x, et $x_j^{(t)}$ est la valeur du j-ième paramètre à l'itération t. La mise à jour de chaque coordonnée x_j est effectuée en minimisant la fonction objectif par rapport à cette coordonnée, tout en gardant les autres coordonnées fixes.

Voici un exemple minimaliste d'implémentation de l'**algorithme de descente de gradient coordonnée** en Python. Il utilise une fonction simple $f(x, y) = x^2 + y^2$ et part du point (1, 1). L'algorithme effectue 100 itérations avec un pas de 0.1. Après 100 itérations, les valeurs de x et y sont très proches de zéro (environ 2.04×10^{-10}), ce qui est le minimum de la fonction $f(x, y)$.

```python
import numpy as np
# Définition de la fonction et de son gradient
def f(x, y):
    return x**2 + y**2
def gradient(x, y):
    return np.array([2*x, 2*y])
# Paramètres de l'algorithme
step = 0.1
x, y = 1, 1
# Boucle principale
for _ in range(100):
    grad = gradient(x, y)
    x, y = x - step * grad[0], y - step * grad[1]
print(f"x: {x}, y: {y}")
```

```
python algo-descente-gradient-coordo.py
x: 2.0370359763344877e-10, y: 2.0370359763344877e-10
```

16. Régression Ridge ou régularisation L2

La régularisation L2, également connue sous le nom de régression Ridge, est une technique utilisée en apprentissage automatique et en statistiques pour prévenir le surapprentissage et améliorer la généralisation du modèle. Elle fonctionne en ajoutant un terme de pénalité à la fonction de coût, qui est proportionnel à la somme des carrés des coefficients du modèle. Cela a pour effet de réduire la magnitude des coefficients, ce qui rend le modèle moins complexe et donc moins susceptible de surapprendre.

La régularisation L2 et la régression Ridge sont essentiellement la même chose. Le terme "Ridge" est plus couramment utilisé en statistiques, tandis que "L2" est plus couramment utilisé en apprentissage automatique. Les deux termes font référence à la même technique de régularisation, qui consiste à ajouter un terme de pénalité égal à la somme des carrés des coefficients du modèle à la fonction de coût.

La principale différence entre la régularisation L2/Ridge et d'autres techniques de régularisation, comme la régularisation L1/Lasso (3), réside dans la manière dont la pénalité est calculée. Dans la régularisation L2/Ridge, la pénalité est proportionnelle à la somme des carrés des coefficients, tandis que dans la régularisation L1/Lasso, la pénalité est proportionnelle à la somme des valeurs absolues des coefficients. Cela conduit à des différences dans la manière dont ces techniques affectent les coefficients du modèle : la régularisation L2/Ridge tend à réduire tous les coefficients de manière égale, tandis que la régularisation L1/Lasso peut réduire certains coefficients à zéro, effectuant ainsi une sélection de variables.

La pénalité sur la somme des carrés des résidus (21), ou **Ridge Regression** [10], est une forme de **régression linéaire régularisée** [11] qui impose une pénalité à la taille des coefficients pour éviter l'overfitting.

10. Le terme "Ridge Regression" vient du mot anglais "ridge", qui signifie "crête". Cela est dû à la façon dont cette méthode de régularisation affecte le paysage de la fonction d'objectif, c'est-à-dire la fonction que le modèle essaie de minimiser pendant l'apprentissage.

11. technique d'apprentissage automatique supervisé utilisée pour prédire une variable continue, ou variable cible, à partir d'une ou plusieurs variables d'entrée, ou variables explicatives. Le modèle de régression linéaire suppose que la relation entre les variables d'entrée et la variable cible est linéaire.

Exemple d'implémentation en Python à l'aide de la bibliothèque sklearn :

```python
import matplotlib.pyplot as plt
from sklearn.linear_model import Ridge
from sklearn.model_selection import train_test_split
from sklearn.datasets import fetch_california_housing

# Load the California housing dataset
housing = fetch_california_housing()
X = housing.data
y = housing.target

# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Create a Ridge Regression instance
ridge = Ridge(alpha=1.0)

# Train the model
ridge.fit(X_train, y_train)

# Make predictions
predictions = ridge.predict(X_test)

# Visualize the results
plt.scatter(y_test, predictions)
plt.xlabel('True Values')
plt.ylabel('Predictions')
plt.title('Ridge Regression Predictions vs True Values')
plt.grid(True)
plt.show()
```

Dans cet exemple, on utilise le jeu de données **"fetch california housing"** sur le logement en Californie et une valeur de pénalité (alpha) de 1.0. Le jeu de données sur le logement en Californie contient des informations sur le logement en Californie, telles que le revenu médian, l'âge médian des logements, le nombre moyen de pièces, le nombre moyen de chambres, la population, le nombre moyen de ménages, la latitude et la longitude. La variable cible est la valeur médiane des

maisons.

On génère un graphique de dispersion qui compare les valeurs réelles aux prédictions de la régression Ridge. Chaque point représente une observation, avec la valeur réelle sur l'axe des x et la prédiction sur l'axe des y. Une ligne de référence (ligne pointillée noire) qui représente l'endroit où les prédictions seraient parfaitement égales aux valeurs réelles :

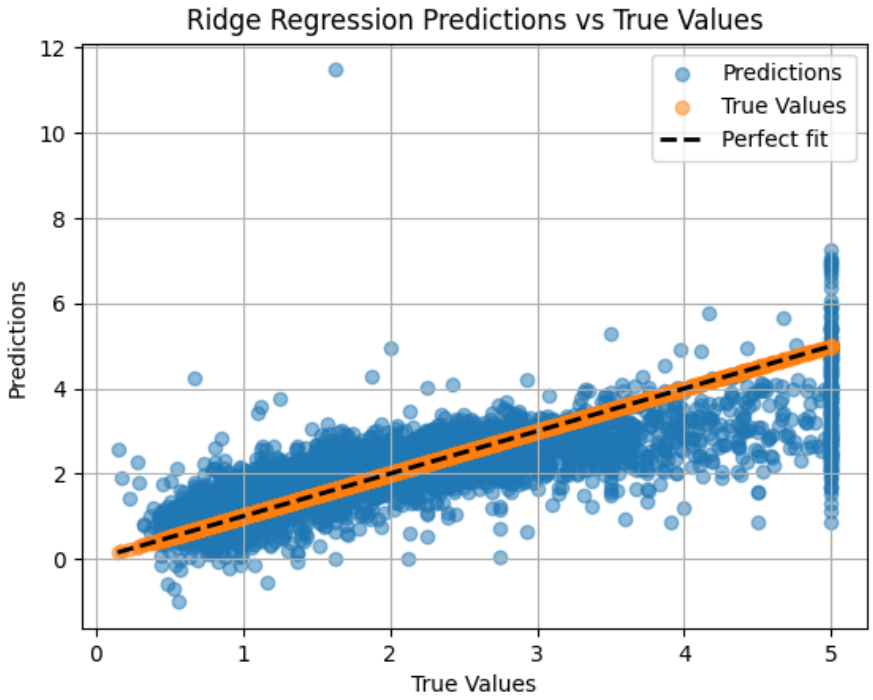

Figure (3) – Graphique de dispersion qui compare les valeurs réelles aux prédictions de la régression Ridge

La technique de la régression **Ridge** a été introduite par le statisticien américain Arthur E. **Hoerl** et son collègue Robert W. Kennard en 1970. Dans leur article intitulé *"Ridge Regression : Biased Estimation for Nonorthogonal Problems"*, ils ont présenté [4] la régression Ridge comme une solution aux problèmes de multicollinéarité [12] dans les modèles de régression multiple.

12. Les problèmes de multicollinéarité surviennent lorsque des variables prédictives dans un modèle de régression sont fortement corrélées entre elles

17. Somme des Carrés des Résidus - RSS

Le **RSS** "**Residual Sum of Squares**" ou *Somme des Carrés des Résidus* est une mesure couramment utilisée pour évaluer la performance d'un modèle de régression.

Ainsi, dans la régression Ridge (18), au lieu de minimiser simplement le RSS, on minime la somme du RSS et de la pénalité de régularisation. Cette pénalité de régularisation est la somme des carrés des coefficients de la régression, multipliée par un paramètre de régularisation (généralement noté lambda ou alpha). En formule, cela donne :

```
minimiser(RSS + alpha * (somme des carrés des coefficients))
```

$$\text{RSS} = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 \tag{13}$$

Dans cette équation, y_i représente la i-ème valeur réelle, $\hat{y}_i$ représente la i-ème valeur prédite, et n est le nombre total d'observations. La somme s'étend sur toutes les observations.

La **somme des carrés des résidus** est une mesure de l'écart entre les données et un modèle d'estimation. On a RSS = $\sum_{i=1}^{n} (y_i - \hat{y}_i)^2$, où n est le nombre de points de données, y_i est la valeur observée pour le i-ème point de données et $\hat{y}_i$ est la valeur prédite pour le i-ème point de données.

$$\text{RSS} = \sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij} \right)^2 \tag{14}$$

L'équation est une représentation mathématique de la somme des carrés des résidus (RSS) dans un modèle de régression linéaire multiple. Dans ce modèle, Y représente la variable dépendante, $X_1, X_2, \ldots, X_p$ représentent les variables indépendantes ou les prédicteurs pour Y, et $\beta_0, \beta_1, \ldots, \beta_p$ représentent les estimations des coefficients pour les différentes variables ou prédicteurs. Le RSS est une mesure de l'écart entre les valeurs observées de Y et les valeurs prédites par le modèle de régression linéaire. Un petit RSS indique un ajustement serré du modèle aux données .

Dans cette équation, RSS $= \sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij} \right)^2$, n est le nombre de points de données, y_i est la valeur observée pour le $i - me$ point de données et $\beta_0 + \sum_{j=1}^{p} \beta_j x_{ij}$ est la valeur prédite pour le i-ème point de données. La valeur prédite est calculée comme une combinaison linéaire des variables indépendantes x_{ij}, où

x_{ij} représente la valeur de la $j-me$ variable indépendante pour le $i-mepoint$ de données et β_j représente l'estimation du coefficient pour la $j-me$ variable indépendante

Les résidus sont la différence entre les valeurs réelles de la variable cible et les valeurs prédites par le modèle. Lorsqu'on parle de minimiser le RSS, cela signifie qu'on cherche à minimiser la somme de ces résidus au carré, c'est-à-dire à rendre les prédictions du modèle aussi proches que possible des valeurs réelles.

La Regression **Ridge** est une forme de régression linéaire régularisée qui ajoute une pénalité à la taille des coefficients de la régression. Cette pénalité est en réalité une forme de contrainte sur la taille de ces coefficients :

```python
import matplotlib.pyplot as plt
from sklearn.linear_model import Ridge
from sklearn.model_selection import train_test_split
from sklearn.datasets import fetch_california_housing
import numpy as np

# Charger le jeu de données sur le logement en Californie
housing = fetch_california_housing()
X = housing.data
y = housing.target

# Séparer les données en ensemble d'apprentissage et de test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Créer une instance de Ridge Regression
ridge = Ridge(alpha=1.0)

# Entraîner le modèle
ridge.fit(X_train, y_train)

# Faire des prédictions
predictions = ridge.predict(X_test)

# Calculer le RSS
rss = np.sum((predictions - y_test)**2)

# Afficher le RSS
```

```python
print(f"Somme des Carrés des Résidus (RSS) : {rss}")

# Visualiser les résultats
plt.scatter(y_test, predictions, alpha=0.5, label='Prédictions')
plt.scatter(y_test, y_test, alpha=0.5, label='Valeurs réelles')
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], \
'k--', lw=2, label='Correspondance parfaite')
plt.xlabel('Valeurs réelles')
plt.ylabel('Prédictions')
plt.title('Prédictions de la régression Ridge vs Valeurs réelles')
plt.legend()
plt.grid(True)
plt.show()
```

```
python rss.py
Somme des Carrés des Résidus (RSS) : 2294.356711748009
```

18. Comment fonctionne la régularisation en synthèse

La régularisation [14] [2] fonctionne en ajoutant une pénalité ou un terme de complexité ou de rétrécissement avec la **somme des carrés des résidus** [15] (RSS). Si on prends l'équation de régression linéaire simple. Ici, Y représente la caractéristique dépendante ou la réponse qui est la relation apprise. Ensuite, Y est approximé à : $Y \approx \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_p X_p$

Ici, $X_1, X_2, \ldots, X_p$ sont les caractéristiques indépendantes ou les prédicteurs pour Y, et $\beta_0, \beta_1, \ldots, \beta_n$ représentent les estimations des coefficients pour les différentes variables ou prédicteurs (X), qui décrivent les poids ou l'ampleur attachés aux caractéristiques, respectivement. En régression linéaire simple, notre fonction d'optimisation ou fonction de perte est connue sous le nom de somme des carrés des résidus (RSS) (21).

C'est là que la régularisation entre en jeu, qui rétrécit ou régularise ces estimations apprises vers zéro, en ajoutant une fonction de perte avec des paramètres d'optimisation pour faire un modèle qui peut prédire la valeur précise de Y.

19. Régularisation des réseaux de neurones et Deep learning

Les algorithmes d'optimisations stochastiques marchent bien en pratique. Ils convergent souvent vers un très bon minimum local de la fonction de coût. Comme ces modèles ont beaucoup de paramètres, il y a souvent plus de paramètres que d'observations, il faut faire attention au surapprentissage. Le surapprentissage correspond à un ajustement trop étroit ou exact à un ensemble particulier de données. Le modèle apprend "par coeur" les données d'apprentissage et ne peut pas généraliser sur de nouvelles données. Pour ne pas choisir un mauvais modèle, on découpe l'ensemble d'apprentissage en deux : une majorité des données pour estimer les paramètres du modèle (ensemble d'entrainement). Une minorité de données pour estimer la généralisation du modèle (ensemble de validation). En anglais, cette méthode porte le nom de **"hold-out"**.

Bien qu'on choisisse le meilleur modèle à l'aide d'un ensemble de validation, Il est préférable, de controler la puissance de modélisation du MLP [13]. Actuellement, il y a essentiellement quatre méthodes utilisées en Deep Learning :

- Le **weight-decay** qui pénalise la fonction de coût en fonction de la taille des paramètres. L'avantage de cette technique est de stabiliser l'algorithme d'optimisation du gradient stochastique en empéchant les poids de devenir trop grands.

13. multi-layer perceptron [16]

- Le **drop out**, tous les poids ne sont pas mis-à-jours à chaque mini-batch. C'est comme si devant chaque unitées cachées d'une couche, il y avait un masque qui laisse passer ou non l'information avec une probabilité p fixée, mais à choisir.
- Le **mixup**, qui consiste à créer des observations virtuelles qui sont un mélange des vraies observations.
- La "Batch normalization" a la réputation de régulariser le modèle mais on ne sait pas très bien quelle en est la raison. Cette technique a été introduite pour rendre la descente de gradient plus efficace.

Aucune de ces méthodes n'est exclusive, on peut très bien utiliser plusieurs en même temps[17]. Voici comment les techniques de régularisation principalement utilisées en deep learning se rapportent à la régularisation L1 et L2 :

Weight decay : Il s'agit essentiellement de la régularisation L2 appliquée aux réseaux de neurones. Le terme de pénalité est proportionnel à la somme des carrés des poids, ce qui encourage le modèle à avoir de petits poids.

Dropout : Il s'agit d'une technique de régularisation différente qui fonctionne en désactivant aléatoirement certains neurones pendant l'entraînement. Cela encourage le modèle à être plus robuste et moins dépendant de tout neurone individuel. Bien que différente de la régularisation L1 et L2, elle vise le même objectif général de prévention du surapprentissage.

Mixup : Cette technique génère de nouvelles données d'entraînement en mélangeant aléatoirement les exemples d'entraînement et leurs étiquettes. Bien que cela ne soit pas directement lié à la régularisation L1 et L2, cela peut aider à prévenir le surapprentissage en encourageant le modèle à être plus robuste aux variations des données d'entrée.

Batch normalization : Cette technique normalise les activations des neurones pour chaque mini-lot pendant l'entraînement. Cela peut aider à stabiliser l'apprentissage et réduire le surapprentissage, bien que ce ne soit pas une forme de régularisation au sens traditionnel.

En résumé, bien que ces techniques soient différentes de la régularisation L1 et L2, elles visent toutes à prévenir le surapprentissage et à rendre le modèle plus robuste, ce qui est l'objectif général de toute technique de régularisation.

20. Régression Lasso

Le graphique plus bas (fig 4) montre le chemin de rétrécissement des coefficients de régression Lasso (3). La régression LASSO (Least Absolute Shrinkage and Selection Operator) est une méthode qui peut être utilisée pour ajuster un modèle de régression lorsque la multicolinéarité est présente dans les données. Elle réduit les coefficients vers zéro en introduisant un facteur de pénalisation appelé valeurs alpha (α).

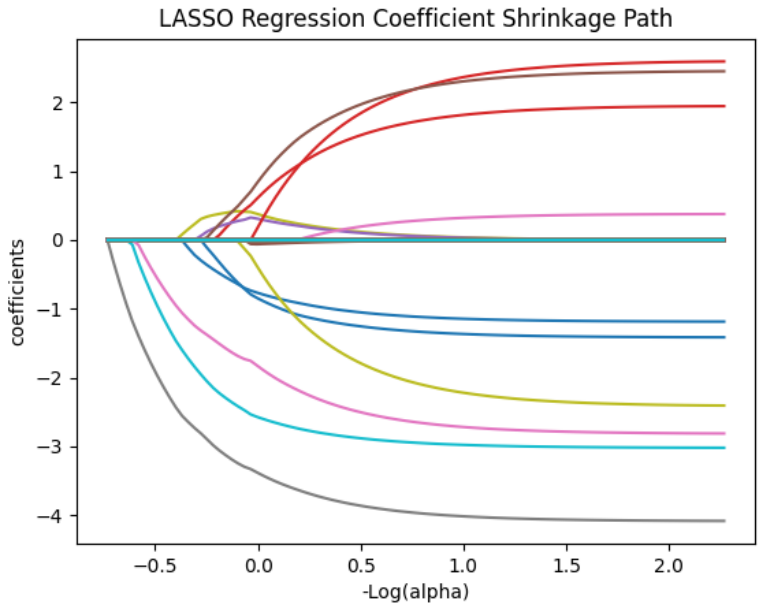

Figure (4) – Les coefficients standardisés de la régression LASSO en fonction de Log(α)

Le graphique précédent a été généré avec le code qui suit et qui génère des données et utilise la fonction **lasso_path** du module **linear_model** de scikit-learn pour calculer le chemin Lasso avec une **"descente de coordonnées"** (16) :

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import lasso_path
# Generate some data
np.random.seed(0)
n_samples, n_features = 50, 200
X = np.random.randn(n_samples, n_features)
coef = 3 * np.random.randn(n_features)
inds = np.arange(n_features)
np.random.shuffle(inds)
coef[inds[10:]] = 0  # sparsify coef
y = np.dot(X, coef)

# Add noise
y += 0.01 * np.random.normal(size=n_samples)

# Compute Lasso path with coordinate descent
alphas_lasso, coefs_lasso, _ = lasso_path(X, y)

# Display results
plt.figure()
neg_log_alphas_lasso = -np.log10(alphas_lasso)
for coef_l in coefs_lasso:
    l1 = plt.plot(neg_log_alphas_lasso, coef_l)

plt.xlabel('-Log(alpha)')
plt.ylabel('coefficients')
plt.title('LASSO Regression Coefficient Shrinkage Path')
plt.axis('tight')
plt.show()
```

Références

[1] Empêcher le sur-ajustement et les données déséquilibrées avec le ML automatisé (2023).
URL https://learn.microsoft.com/fr-fr/azure/machine-learning/concept-manage-ml-pitfalls?view=azureml-api-2

[2] Chirag Goyal, Complete guide to regularization techniques in machine learning, https://www.analyticsvidhya.com/blog/2021/05/complete-guide-to-regularization-techniques-in-machine-learning/ (2021).

[3] Hoerl, Arthur E and Kennard, Robert W, Ridge regression : Biased estimation for nonorthogonal problems, Technometrics 8 (1970) 27–51.

[4] Hoerl, Arthur E and Kennard, Robert W, Ridge Regression: Biased Estimation for Nonorthogonal Problems, Technometrics 12 (1) (1970) 55–67. doi:10.1080/00401706.1970.10488634.
URL https://homepages.math.uic.edu/~lreyzin/papers/ridge.pdf

[5] Hoerl, Arthur E. and Kennard, Robert W., Ridge Regression : Biased Estimation for Nonorthogonal Problems, Technometrics 42 (1) (2000) 80, https://sci-hub.st/10.2307/1271436. doi:10.2307/1271436.

[6] W. Rudin, Real and Complex Analysis, McGraw-Hill, 1987.
URL https://59clc.files.wordpress.com/2011/01/real-and-complex-analysis.pdf

[7] Banach, Stefan, Théorie des opérations linéaires, Warsaw, 1932.
URL http://eudml.org/doc/268537

[8] Minkowski, Hermann, Allgemeine Lehrsätze über die konvexen Polyeder, Nachrichten von der Gesellschaft der Wissenschaften zu Göttingen, Mathematisch-Physikalische Klasse (1896) 198–219.

[9] Minkowski, Hermann, Geometrie der Zahlen, Teubner, 1910.

[10] Tibshirani, Robert, Regression shrinkage and selection via the lasso, Journal of the Royal Statistical Society : Series B (Methodological) 58 (1) (1996) 267–288.
URL https://homepages.math.uic.edu/~lreyzin/papers/lasso.pdf

[11] Franck Jeannot, Fonctions de coût et Machine Learning (2018).
URL https://franckybox.com/wp-content/uploads/cost_function.pdf

[12] Wright, Stephen J, Coordinate descent algorithms, Mathematical Programming 151 (1) (2015) 3–34.

[13] Tseng, Paul, Convergence of a Block Coordinate Descent Method for Nondifferentiable Minimization, Journal of Optimization Theory and Applications 109 (3) (2001) 475–494.

[14] Ritwick Roy, Regularization in machine learning.
URL https://towardsdatascience.com/regularization-in-machine-learning-6fbc4417b1e5

[15] Residual sum of squares.
URL https://en.wikipedia.org/wiki/Residual_sum_of_squares

[16] Franck Jeannot, Intelligence Artificielle et réseaux neuronaux (2023).
URL https://franckybox.com/wp-content/uploads/Intelligence_Artificielle_et_reseaux_neuronaux_v1.2.pdf

[17] J. Rynkiewicz, Introduction au Deep Learning Régularisation et sélection de modèles (2022).
URL https://samm.univ-paris1.fr/IMG/pdf/coursdeep3.pdf