École Nationale d'Ingénieurs de Tunis

Rapport de Projet

Machine Learning

Différents Types de Régularisations

Elaboré par :
Siwar Chouchi
Yassine Tisseoui
Mehdi Jebali

3AMIndS

Encadré par :
Azmi Makhlouf
Radhia Bessi

13 Décembre 2017

Table des matières

Table des figures

Introduction

L'apprentissage automatique (Machine Learning), plus précisément le domaine de la modélisation prédictive, vise principalement à minimiser l'erreur d'un modèle ou à rendre possibles les prédictions les plus précises, au détriment de l'explicabilité.

Dans l'apprentissage automatique appliqué, nous emprunterons et réutiliserons des algorithmes provenant de nombreux domaines différents, y compris des statistiques, et les utiliserons à ces fins.

A ce tire, la régression linéaire a été développée dans le domaine des statistiques et est étudiée comme un modèle pour comprendre la relation entre les variables numériques d'entrée et de sortie, mais a été empruntée par l'apprentissage automatique. C'est à la fois un algorithme statistique et un algorithme d'apprentissage automatique.

Différentes techniques peuvent être utilisées pour préparer ou former l'équation de régression linéaire à partir de données telles que la régression linéaire simple, les moindres carrés ordinaires, la descente en gradient, la régularisation ...

Ce qui nous intéresse dans ce projet, ce sont les différents types de régularisations. Nous verrons ainsi dans la suite l'intérêt et la manifestation de cette méthode de pénalisation dans la correction et l'amélioration du modèle de régression linéaire.

Dans le premier chapitre, nous présentons les définitions nécessaires et l'étude théorique des techniques de régularisation dans la régression linéaire ainsi qu'une étude numérique programmée avec Python sur une base de donnée choisie.

Chapitre 1

Etude Théorique

1.1 Régression linéaire

La régression linéaire est un modèle linéaire, par ex. un modèle qui suppose une relation linéaire entre les variables d'entrée (x) et la variable de sortie unique (y). Plus précisément, y peut être calculé à partir d'une combinaison linéaire des variables d'entrée (x).

Lorsqu'il existe une seule variable d'entrée (x), la méthode est appelée régression linéaire simple. Lorsqu'il existe plusieurs variables d'entrée, la littérature issue des statistiques se réfère souvent à la méthode sous forme de régression linéaire multiple. L'apprentissage d'un modèle de régression linéaire consiste à estimer les valeurs des coefficients utilisés dans la représentation avec les données dont nous disposons.

1.2 Régularisation

La régularisation est une technique utilisée pour tenter de résoudre le problème de surajustement (overfitting) dans les modèles statistiques. L'objectif de l'apprentissage automatique est maintenant de modéliser le motif et d'ignorer le bruit. Chaque fois qu'un algorithme essaie d'ajuster le bruit en plus du motif, il est en surapprentissage.

Pour corriger le surajustement, la régularisation ajoute une pénalité sur les différents paramètres du modèle pour réduire la liberté du modèle. Par conséquent, le modèle sera moins susceptible de s'adapter au bruit des données d'apprentissage et améliorera les capacités de généralisation du modèle.

Ainsi, les méthodes de régularisation cherchent à minimiser la somme de l'erreur quadratique du modèle sur les données d'apprentissage (en utilisant les moindres carrés ordinaires) mais aussi à réduire la complexité du modèle (comme le nombre ou la taille absolue de la somme de tous les coefficients dans le modèle) .

Deux exemples populaires de procédures de régularisation pour la régression linéaire sont :

Régression Lasso : où les moindres carrés ordinaires sont modifiés pour minimiser également la somme absolue des coefficients (appelée régularisation L1).

Régression Ridge : où les moindres carrés ordinaires sont modifiés pour minimiser également la somme absolue au carré des coefficients (appelée régularisation L2 ou régularisation de Tikhonov).

Régression Elastic Net : essentiellement une combinaison de la régularisation L1 et L2.

La procédure des moindres carrés ordinaires vise à minimiser la somme des résidus au carré. Cela signifie qu'étant donné une ligne de régression à travers les données, nous calculons la distance entre chaque point de données et la ligne de régression, le carré et somme toutes les erreurs au carré ensemble. C'est la quantité que les moindres carrés ordinaires cherchent à minimiser.

La régression Ridge et Lasso sont des techniques puissantes généralement utilisées pour créer des modèles parcimonieux en présence d'un «grand» nombre de caractéristiques et sont efficaces lorsqu'une colinéarité est présente dans les valeurs d'entrée et que les moindres carrés ordinaires surajustent les données d'entraînement.

Bien que Ridge et Lasso puissent sembler travailler vers un but commun, les propriétés inhérentes et les cas d'utilisation pratique diffèrent considérablement. Ils agissent en pénalisant l'ampleur des coefficients de caractéristiques tout en minimisant l'erreur entre les observations prédites et réelles.

Le modèle peut être bon mais il peut probablement être surajusté , c'est-à-dire qu'il aura probablement une mauvaise prédiction et une puissance de généralisation : il colle trop aux données et le modèle a probablement appris le bruit de fond tout en étant en forme.

Procès : La technique de régularisation est utile pour résoudre ce problème. Nous pénalisons la fonction de perte en ajoutant un multiple d'une norme L1 (Lasso) ou L2 (Ridge) du vecteur w (c'est le vecteur des paramètres appris dans votre régression linéaire). Nous obtenons l'équation suivante :

$$L(X, Y) + \lambda N(w)$$

(N est la norme L1, L2 ou toute autre norme)

Cela nous aidera à éviter le surapprentissage et effectuera, en même temps, la sélection des fonctionnalités pour certaines normes de régularisation.

Équilibrage de l'erreur Bien que la régularisation nous permettra de corriger les problèmes de sur-apprentissage, nous ne voulons en aucun cas arriver à modèle qui souffre d'un problème de sous-apprentissage. Cela peut arriver si l'on régularise trop le modèle. Afin de comprendre ce qui arrive lors de la régularisation il faut que l'on considère que l'erreur est composé de deux parties : Une erreur de biais, et une erreur de variation.

L'erreur de variation Supposons que nous répétons le processus de génération de notre modèle prédictif. Un même point de données peut à chaque fois donner une prédiction différente. Cette erreur de variance représente le degré de variabilité de la prédiction d'un point de données à plusieurs réalisations du modèle.

L'erreur de biais L'erreur de biais est la différence moyenne entre la prédiction généré par le modèle et la valeur réelle. La figure suivante illustre la différence entre les deux erreurs.

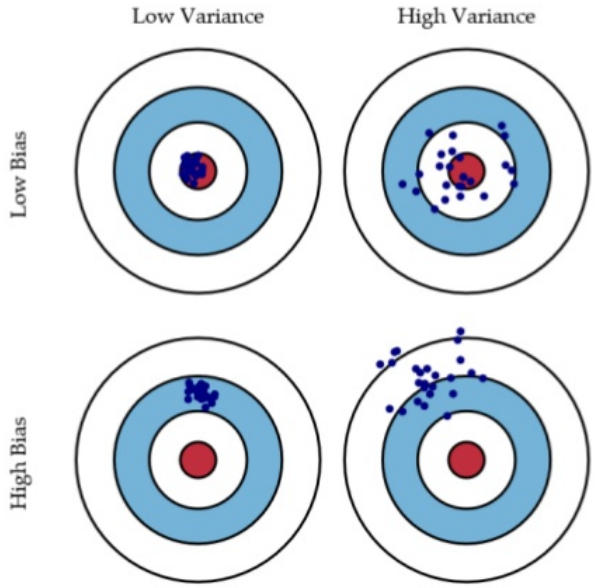

FIGURE 1.1 – Illustration des deux types d'erreur

Revenons à notre problème de régularisation, l'ajout du terme $\lambda N(w)$ ajoute de la complexité au modèle dans le but de décroître l'erreur de biais et d'avoir des prédictions plus proches aux valeurs réelles. Cependant, le fait d'augmenter la complexité augmentera l'erreur lié à la variance. Cela veut dire que si l'on régularise trop le modèle nous pouvons

avoir une forte erreur liée à la variation due à des coefficients devenus trop faible due à la forte régularisation. Ici le modèle a un problème dit de sous-apprentissage. Nous devons, en effet, assurer un modèle qui minimise l'erreur totale en essayant d'équilibrer entre les deux erreurs Comme l'illustre la figure suivante. Pour le que le modèle soit bon, nous avons

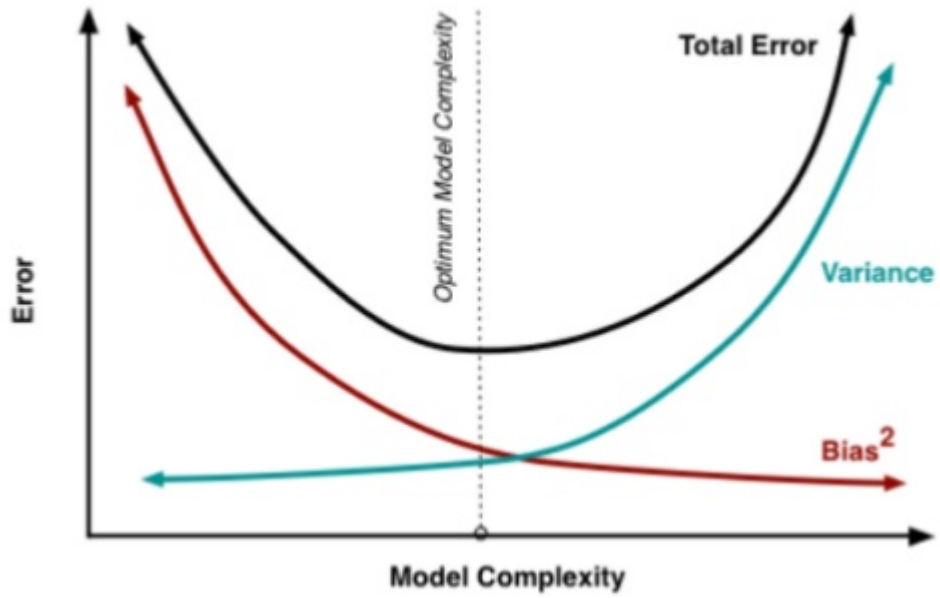

FIGURE 1.2 – Erreurs total, de variation et de biais en fonction de la complexité du modèle

besoin de choisir la norme N, et le paramètre λ afin d'avoir une erreur totale minimale. La première étape est de choisir la à utiliser. Ici nous allons uniquement considérer les régressions Ridge(L2), LASSO (L1) et Elastic Net(Combinaison de Ridge et Lasso). Après cela, nous aurons besoin de syntoniser le terme de régularisation λ. Une réponse possible est d'utiliser la validation croisée (cross-validation) : nous divisons nos données d'entraînement, nous formons notre modèle pour une valeur fixe de λ, puis nous le testons sur les sous-ensembles restants et nous répétons cette procédure en faisant varier λ. Ensuite, nous sélectionnons le meilleur λ qui minimise la fonction de perte.

1.2.1 Régression Ridge

La régression de Ridge sert à minimiser les coefficients en ajoutant un terme de pénalité en norme L^2. Il réduit les coefficients vers zéro. Cela introduit un certain biais mais on réduit la variance, ce qui engendre une meilleure erreur moyenne quadratique. La régression de Ridge est optimale lorsqu'il existe un sous-ensemble des coefficients qui sont quasiment nuls.

Le principe de cette régresion est de pénaliser la fonction de perte de telle sorte on obtient :

$$\sum_{i=1}^{n}(y_i - \sum_{j=1}^{p} x_{ij}\beta_j)^2 + \lambda \sum_{j=1}^{p} |\beta_j|^2$$

pour

$$\sum_{j=1}^{p} |\beta_j|^2 \leq c, \qquad c > 0$$

Ce qui est équivalent de minimiser la somme des résidus quadratiques :

$$\sum_{i=1}^{n}(y_i - \sum_{j=1}^{p} x_{ij}\beta_j)^2$$

La régression de Ridge impose d'autres contraintes sur les paramètres. Autrement, on va non seulement minimiser la somme des résidus mais aussi introduire un terme de pénalité λ suffisamment petit ($\lambda < 1$) pour que la fonction perte soit pénalisée dans le cas ou les coefficients β_j prennent des valeurs grandes.
La figure ci-dessous explique le concept de la méthode de régression de Ridge :

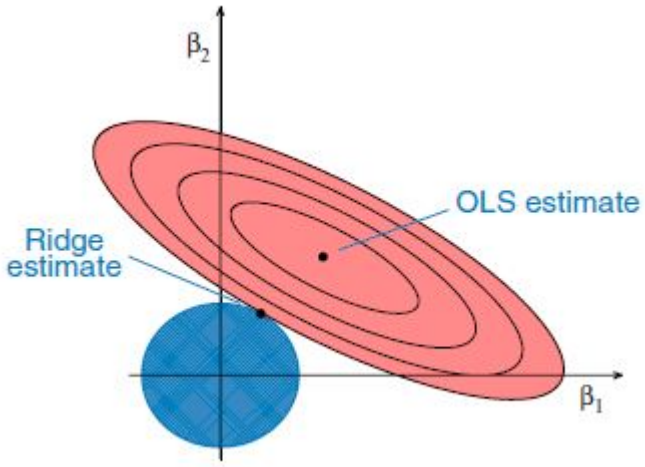

FIGURE 1.3 – Interprétation géométrique de la régression de Ridge

Les ellipses correspondent aux contours de la somme résiduelle des carrés (RSS) : l'ellipse interne a un RSS plus petit, et RSS est minimisée aux estimations ordinales par les moindres carrés (OLS).
Pour p=2, la contrainte de Ridge correspond à un cercle : $\beta_1^2 + \beta_2^2 < c$. L'objectif est de diminuer la taille de l'ellipse et du cercle simultanément. L'estimation de Ridge est donnée par le point auquel l'ellipse et le cercle se touchent.
Il y a un compromis entre la pénalité et RSS, même β_j grand vous donnerait une meilleure somme de carrés résiduelles, mais il augmentera le terme de pénalité. C'est pour cela on choisit généralemnt des coefficients petits.

1.2.2 Régression LASSO

LASSO veut dire : Least Absolute Shrinkage and Selection Operator. Et comme le nom l'indique une régression de type LASSO permet d'effectuer une sélection des variables, chose que la régression de type Ridge ne permet pas de faire vu que les coefficients ne tendent vers zéro que pour λ qui tend vers l'infini. Pour trouver les coefficients pour la régression LASSO il suffit de résoudre le problème d'optimisation avec contraintes suivant :
Minimiser

$$(y - X\beta)^T(y - X\beta)$$

sachant que

$$\sum_{j=1}^{p} |B_j| \leq t$$

Cela est équivalent à la fonction de perte :

$$PRSS(\beta)_{l1} = \sum_{i=1}^{n}(y_i - x_i^T\beta_i)^2 + \lambda \sum_{j=1}^{p} |\beta_j|$$

Nous avons ici également le paramètre λ qui correspond à $\frac{1}{t}$ et nous avons donc un ensemble de solutions qui sont chacune indexées par le paramètre λ. En effet et surtout quand nous avons un grand nombre de variables, nous voulons que certains des coefficients (ceux correspondants à des variables non utiles) soient nuls et nous voulons donc arriver à une solution "creuse". En effet, la nature de la norme l_1 fait que pour des valeurs de lambda assez grande nous aurons des coefficients qui seront égaux à zéro. Cela diffère d'une manière importante de la régression de Ridge où à des grandes valeurs de λ certains coefficients se stabiliserons à des valeurs proches de zéro sans jamais l'atteindre.

Il faut aussi noter que si nous augmentons λ nous une décroissance (shrinkage) plus grande chez les coefficients non nuls. Il faut donc choisir λ de façon à équilibrer entre ajuster un modèle linéaire de y sur x et rétrécir les coefficients.

Notons que tout comme pour la régression de Ridge nous avons pour $\lambda = 0$ notre régression linéaire de départ. Et pour λ qui tend vers ∞ tout les coefficients tendent vers zéro.

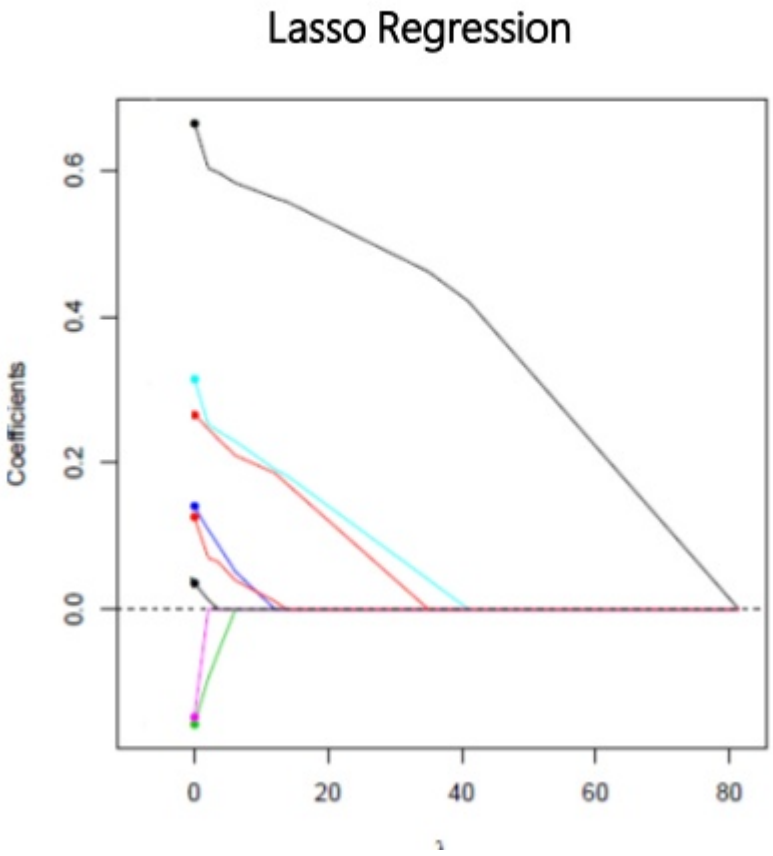

FIGURE 1.4 – Illustration du rétrécissement des coefficients pour la méthode de LASSO en fonction de λ

1.2.3 Régression Elastic Net

Si nous considérons maintenant, une data set avec un nombre élevé de variables. Il très probable que nous obtenons une forte corrélation entre les variables. Ces variables corrélés forment des groupement ou cluster et il est intéressant de choisir tout le cluster de variables corrélées au lieu d'une seule. La régression Elastic net permet de sélectionner des groupements de variables au lieu d'une seule comme un filet de pêche d'où son nom. Cette régression permet de corriger un problème important présent dans la régression LASSO qui est le fait que le nombre de variables qui peuvent être sélectionnées par une régression LASSO est limité par le nombre d'échantillons de données disponibles.

Elaboration de l'Elastic Net Cette régression permet de combiner les avantages des deux régressions Ridge et LASSO. La pénalité combine en fait une norme l_1 et une norme l_2 comme suit :

$$\beta^{elastique} = argmin(||y - X\beta||_2^2 + \lambda_1|\beta| + \lambda_2||\beta||_2^2)$$

Nous pouvons, en effet, considérer les régressions Ridge et Lasso comme étant des cas particuliers de la régression Elastic Net. Nous pouvons en fonctions des paramètres λ_1 et λ_2 rendre le "filet" plus lisse(souple) et nous obtiendrons une régression plus semblable à celle de ridge ou nous pouvons le rendre plus rigide et non linéaire, donc plus semblable à une régression de LASSO. Un bon choix des λ pourrait permettre de construire un modèle qui exploite au mieux les avantages des régressions Ridge et LASSO et minimise ainsi l'erreur.

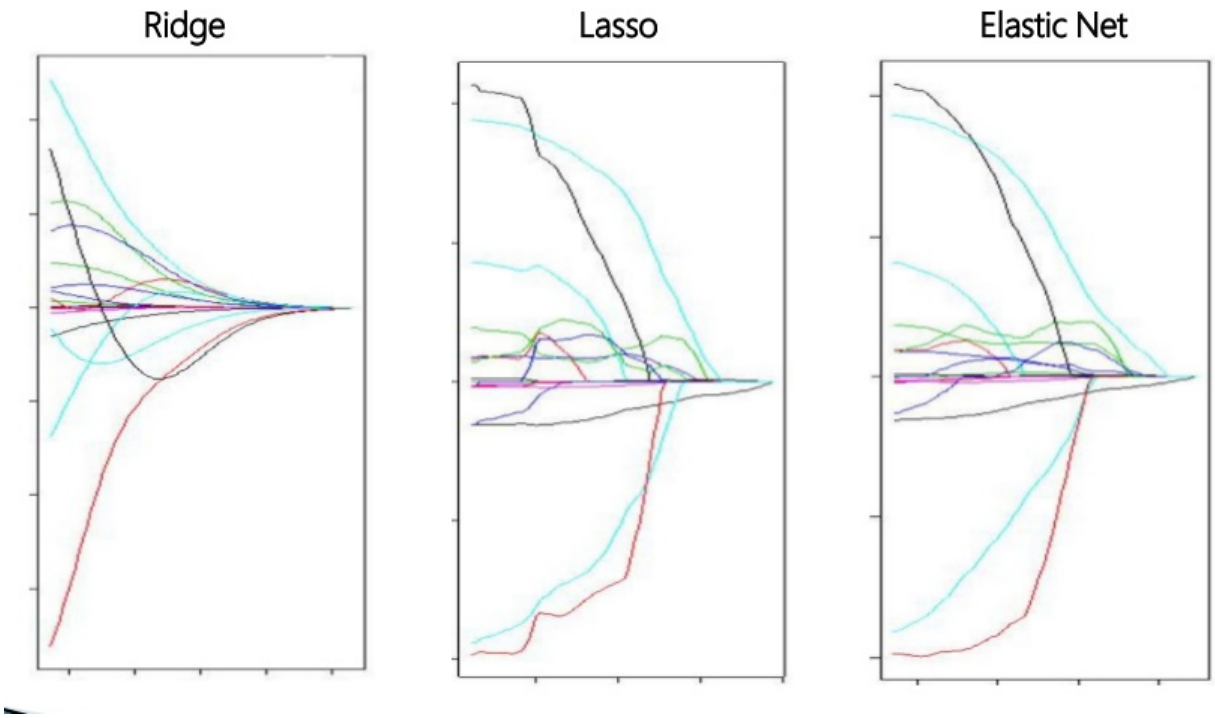

FIGURE 1.5 – Illustration du rétrécissement des coefficients pour les régressions Ridge, LASSO et Elastic Net en fonction de λ

Chapitre 2

Etude Numérique

Dans ce chapitre, nous présenterons l'approche numérique à celle théorique a déjà été expliqué dans le chapitre précédent. En fait, l'objectif de cette étude est de comparer les modèles de régularisation par rapport à l'erreur générée et la résolution du phénomène de surajustement.

2.1 Procédure

Les processus de construction du modèle de régression linéaire et de ses techniques de régularisation sont appliqués à l'ensemble de données acquis par ce site (`https://www.kaggle.com/apapiu/regularized-linear-models/data`) et programmés avec Python. La description de l'ensemble de données et de ses caractéristiques est fournie dans l'annexe.

Maintenant, nous allons utiliser le modèle de régression linéaire pour prévoir les ventes (SalePrice). En fait, la prédiction peut être faite en fonction d'une variable (feature) ou caractéristique, de deux variables, de multiples variables ou de toutes les variables.

Tout d'abord, nous allons utiliser toutes les variables de prédiction avec le modèle de régression linéaire, puis régulariser avec Lasso, Ridge et Elastic Net pour éventuellement être en mesure de sélectionner la bonne caractéristique pour notre modèle avec les bons paramètres de régularisation et selon à l'évaluation de l'erreur quadratique moyenne (root mean squared error = rmse) et le R-carré (cross-validated R_square).

Erreur quadratique moyenne : Pour évaluer la qualité du modèle, nous devons comprendre l'impact des prévisions erronées : si nous prédisons que les ventes sont supérieures à ce qu'elles pourraient être, le magasin dépensera beaucoup d'argent pour faire des arrangements inutiles qui entraîneront des dépassements inventaire. D'un autre côté, si nous le prédisons trop bas, nous perdrons des occasions de vente.

Ainsi, la façon la plus simple de calculer l'erreur sera de calculer la différence entre les valeurs prédites et réelles. Cependant, si nous les ajoutons simplement, ils pourraient s'annuler, donc nous corrigeons ces erreurs avant d'ajouter. Nous les divisons également par le nombre de points de données pour calculer une erreur moyenne car elle ne devrait pas dépendre du nombre de points de données. Ceci est connu comme l'erreur quadratique moyenne (rmse).

$$RMSE^2 = \frac{{e^2}_1 + {e^2}_2 + ... + {e^2}_n}{n}$$

Ici e_1, e_2 .... , e_n sont la différence entre les valeurs réelles et les valeurs prédites.

R_carré ou le R_square : Détermine la quantité de variation totale de Y (variable dépendante) expliquée par la variation de X (variable indépendante). Mathématiquement, il peut être écrit comme :

$$R\_carré = 1 - \frac{\sum(Y_{actual} - Y_{predicted})^2}{\sum(Y_{actual} - Y_{mean})^2}$$

La valeur de R-carré est toujours comprise entre 0 et 1, où 0 signifie que le modèle ne modélise pas et n'explique aucune variabilité dans la variable cible (Y) et 1 explique la variabilité totale de la variable cible.

2.2 Régression Linéaire

Pour construire un modèle de régression contenant toutes les variables, nous devons seulement utiliser des variables continues, car nous devons traiter différemment les variables catégorielles avant de les utiliser dans un modèle de régression linéaire. Il existe différentes techniques pour les traiter. Ici, nous créons des variables fictives pour convertir les variables catégorielles en valeurs numériques, puis nous imputons les valeurs manquantes (NaN) avec la moyenne de leurs colonnes respectives. Et avant tout cela, nous allons transformer les caractéristiques numériques inclinées en prenant log (feature + 1) pour rendre les variables (features) plus normales.

Nous aurons aussi besoin de définir une fonction qui renvoie l'erreur rmse de validation croisée afin que nous puissions évaluer nos modèles et choisir le meilleur paramètre. Son générateur de validation croisée est égale à 7 (70% apprentissage, 30% test).

Voici comment cela est mis en œuvre :

```
#importing all the necessary packages for the coding

import numpy as np
import pandas as pd
from scipy.stats import skew
```

```
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LinearRegression, Ridge, Lasso, ElasticNet


import matplotlib.pyplot as plt
matplotlib.rcParams['figure.figsize'] = (12.0, 6.0)

#loading the dataset
train = pd.read_csv('train.csv')

all_data = pd.concat((train.loc[:,'MSSubClass':'SaleCondition'],test.loc[:,'
    MSSubClass':'SaleCondition']))

train["SalePrice"] = np.log1p(train["SalePrice"])

#log transform skewed numeric features:

numeric_feats = all_data.dtypes[all_data.dtypes != "object"].index
skewed_feats = train[numeric_feats].apply(lambda x: skew(x.dropna()))
skewed_feats = skewed_feats[skewed_feats > 0.75]
skewed_feats = skewed_feats.index

all_data[skewed_feats] = np.log1p(all_data[skewed_feats])

#creating dummies
all_data = pd.get_dummies(all_data)
#filling NA's with the mean of the column:
all_data = all_data.fillna(all_data.mean())

#creating matrices for sklearn:
x_train = all_data[:train.shape[0]]
x_cv = all_data[train.shape[0]:]
y_train = train.SalePrice

#defining rmse
def rmse_cv(model):
    rmse= np.sqrt(-cross_val_score(model, x_train, y_train, scoring="
    neg_mean_squared_error", cv = 7))
    return(rmse)
```

Maintenant, nous construisons le modèle de régression linéaire.

```
#Linear model regression
lreg = LinearRegression()

#rmse evaluation
rmse_mod= rmse_cv(lreg.fit(x_train,y_train)).mean()

# evaluation using r-square
rscore_mod = lreg.score(x_train,y_train)
```

Le rmse de ce modéle est égale à 0.162994917466, alors que le R_carré est égale à 0.947334943997. Ces valeurs explique que notre modèle est capable de prédire des valeurs beaucoup plus proches des valeurs réelles.

2.3 Régression Ridge

Dans cette partie, nous présentons nos résultats de la régularisation Ridge et sa performance tout en variant le paramètre α. En effet, nous allons varier ce paramètre et évaluer le rmse et le R_carré à cet égard pour voir la flexibilité du modèle.

```
#Ridge
alphas = [0.05, 0.1, 0.3, 1, 3, 5, 10, 15, 30, 50, 75]
cv_ridge = [rmse_cv(Ridge(alpha = alpha).fit(x_train,y_train)).mean()
for alpha in alphas]
cv_ridge = pd.Series(cv_ridge, index = alphas)
cv_ridge.plot(title = "RMSE of Ridge Regression")
plt.xlabel("alpha")
plt.ylabel("rmse")
plt.show()

rsq_ridge=[]
for alpha in alphas:
    ridgeReg = Ridge(alpha=alpha, normalize=True)
    ridgeReg.fit(x_train,y_train)
    rsq_ridge.append(ridgeReg.score(x_train,y_train))
rsq_ridge = pd.Series(rsq_ridge, index = alphas)
rsq_ridge.plot(title= "R_square of Ridge Regression")
plt.xlabel("alpha")
plt.ylabel("r_square")
plt.show()
```

Lorsque l'α est trop grand, la régularisation est trop forte et le modèle ne peut pas capturer toutes les complexités dans les données. Si toutefois nous laissons le modèle être trop flexible (α petit), le modèle commence à surpasser. Une valeur de $\alpha = 5$ est à peu près juste en fonction du graphique ci-dessus puisqu'elle donne la plus petite erreur rmse = 0.127630758935. Alors que R_carré atteint son maximum à $\alpha = 0$.

FIGURE 2.1 – L'erreur quadratique moyenne de la régression Ridge

FIGURE 2.2 – Le R_carré de la régression Ridge

2.4 Régression Lasso

Passons au modèle Lasso et voyons l'évolution de la rmse et le R_carré en fonction de α comme déjà fait dans la partie de Ridge.

Ci-dessous le code implémenté pour la simulation de l'évolution.

```
## lasso

alphas2 = [1, 0.5, 0.3, 0.2, 0.1,0.01, 0.001, 0.0005, 0.0001]
cv_lasso = [rmse_cv(Lasso(alpha = alpha).fit(x_train, y_train)).mean()
for alpha in alphas2]
```

```
cv_lasso = pd.Series(cv_lasso, index = alphas2)
cv_lasso.plot(title = "RMSE of Lasso Regression")
plt.xlabel("alpha")
plt.ylabel("rmse")
plt.show()

rsq_lasso=[]
for alpha in alphas2:
    lassoReg = Lasso(alpha=alpha, normalize=True)
    lassoReg.fit(x_train,y_train)
    rsq_lasso.append(lassoReg.score(x_train,y_train))
rsq_lasso = pd.Series(rsq_lasso, index = alphas2)
rsq_lasso.plot(title= "R_square of Lasso Regression")
plt.xlabel("alpha")
plt.ylabel("r_square")
plt.show()
```

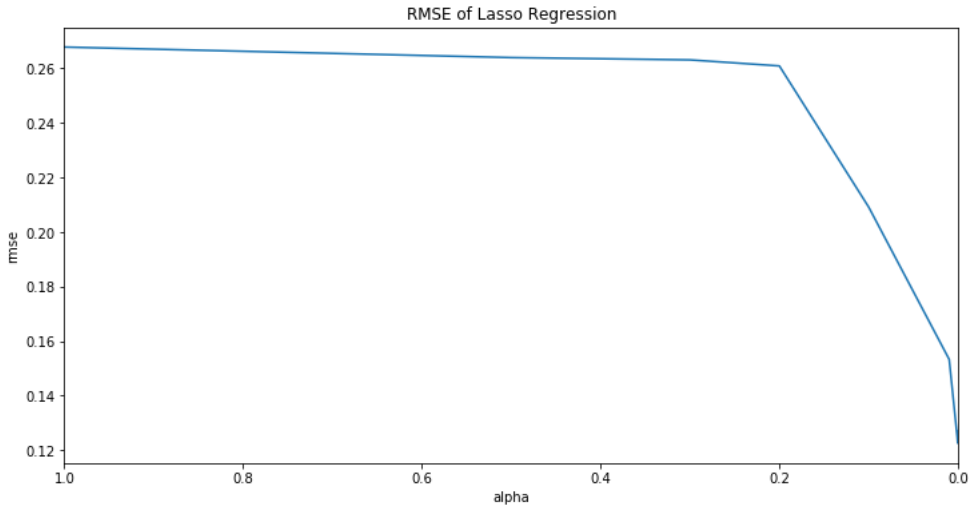

FIGURE 2.3 – L'erreur quadratique moyenne de la régression Lasso

D'après ces figures, nous remarquons qu'à $\alpha = 0.0005$, le rmse atteint une valeur de 0.122. D'où, Lasso a une meilleure performance que Ridge. D'ailleurs le R_carré est plus grand quand α tend vers 0. Un R_carré élevé indique que le modèle a un bon ajustement. Ce qui fait qu'on a un meilleur modèle régularisé.

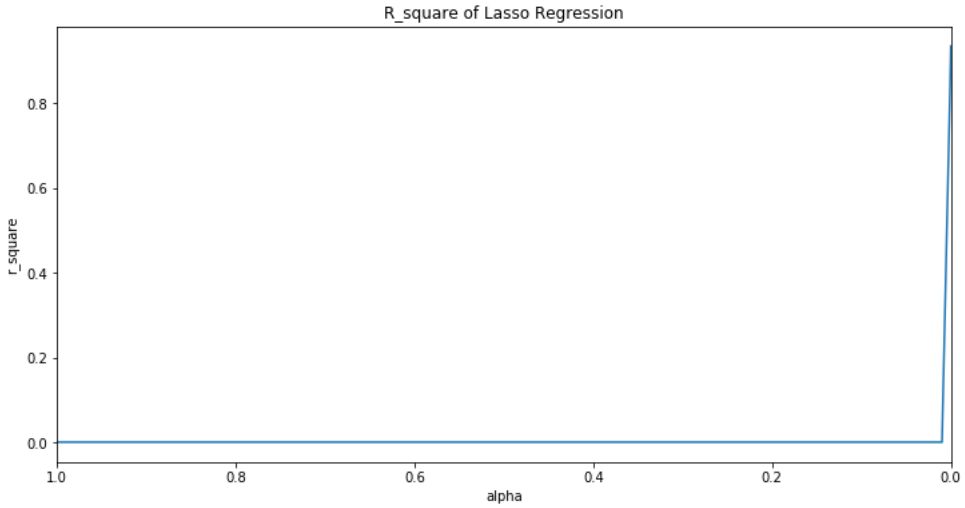

FIGURE 2.4 – Le R_carré de la régression Lasso

Une autre chose intéressante à propos du Lasso est qu'il fait une sélection de variable tout en les mettant des coefficients qu'il semble sans importance à zéro.

2.5 Régression Elastic Net

En ce qui concerne la régression Elastic Net, nous allons varier deux paramètres, le α et le $l1_{ratio}$ où :

$$\alpha = a + b$$

$$l1\_ratio = a/(a + b)$$

Où a et b pondèrent respectivement les termes L1 et L2.

Soit α (ou a + b) = 1, et considérons maintenant les cas suivants :

- Si $l1_{ratio} = 1$, donc si nous regardons la formule de $l1_{ratio}$, nous pouvons voir que $l1_{ratio}$ ne peut être égal à 1 que si a = 1, ce qui implique b = 0. Par conséquent, ce sera une pénalité au lasso.
- De même si $l1_{ratio} = 0$, implique a = 0. Ensuite, la pénalité sera une pénalité de crête.
- Pour $l1_{ratio}$ entre 0 et 1, la pénalité est la combinaison de crête et de lasso.

Par conséquent, nous allons prendre le cas de $\alpha = 1$ et $\alpha = 10$ et varier $l1_{ratio}$ pour voir l'évolution de rmse. Ci-dessous, le code utilisé.

```
#elastic net regression

l1_ratios= [0.5,0.2,0.1,0.01,0.001,0]
cv_elastic = [rmse_cv(ElasticNet(alpha = 1, l1_ratio=l1_ratio).fit(x_train,
    y_train)).mean()
for l1_ratio in l1_ratios]
cv_elastic = pd.Series(cv_lasso, index = l1_ratios)
cv_elastic.plot(title = "RMSE of Elastic Net Regression for alpha = 1")
plt.xlabel("l1_ratio")
plt.ylabel("rmse")
plt.show()

l1_ratios= [0.5,0.2,0.1,0.01,0.001,0]
cv_elastic = [rmse_cv(ElasticNet(alpha = 0.5, l1_ratio=l1_ratio).fit(x_train
    , y_train)).mean()
for l1_ratio in l1_ratios]
cv_elastic = pd.Series(cv_lasso, index = l1_ratios)
cv_elastic.plot(title = "RMSE of Elastic Net Regression for alpha = 0.5")
plt.xlabel("l1_ratio")
plt.ylabel("rmse")
plt.show()

l1_ratios= [1,0.5,0.2,0.1,0.01,0]
cv_elastic = [rmse_cv(ElasticNet(alpha = 10, l1_ratio=l1_ratio).fit(x_train,
     y_train)).mean()
for l1_ratio in l1_ratios]
cv_elastic = pd.Series(cv_lasso, index = l1_ratios)
cv_elastic.plot(title = "RMSE of Elastic Net Regression for alpha = 10")
plt.xlabel("l1_ratio")
plt.ylabel("rmse")
plt.show()

rsq_elas=[]
for l1_ratio in l1_ratios:
    ENreg = ElasticNet(alpha=1, l1_ratio=l1_ratio, normalize=False)
    ENreg.fit(x_train,y_train)
    rsq_elas.append(ENreg.score(x_train,y_train))
rsq_elas = pd.Series(rsq_elas, index = l1_ratios)
rsq_lasso.plot(title= "R_square of Elastic Net Regression for alpha = 1")
plt.xlabel("l1_ratio")
plt.ylabel("r_square")
plt.show()
```

Nous remarquons que nous avons des meilleurs résultats pour le $\alpha = 1$ et surtout quand le $l1_{ratio}$ est très proche de 0, où quand $l1_{ratio} = 0.001$, le rmse $= 0.124$.

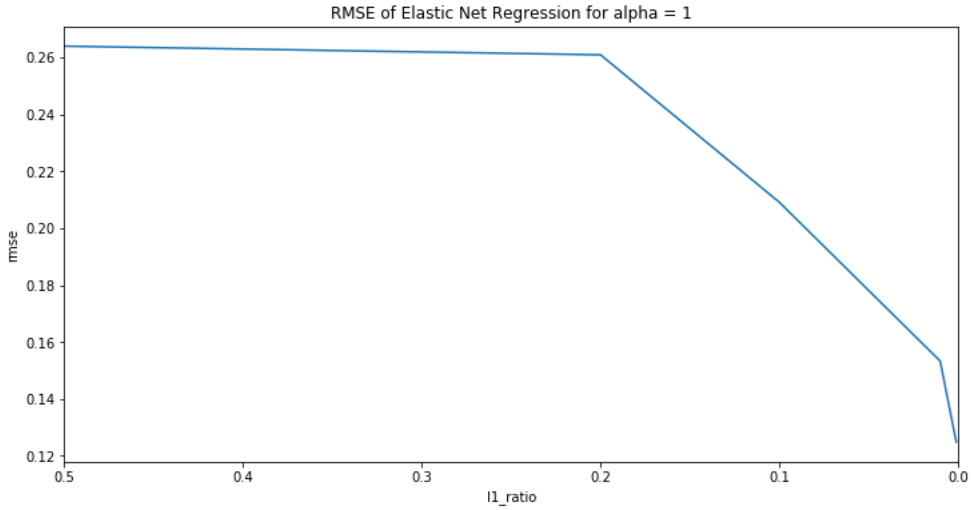

FIGURE 2.5 – L'erreur quadratique moyenne de la régression Elastic Net pour alpha = 1

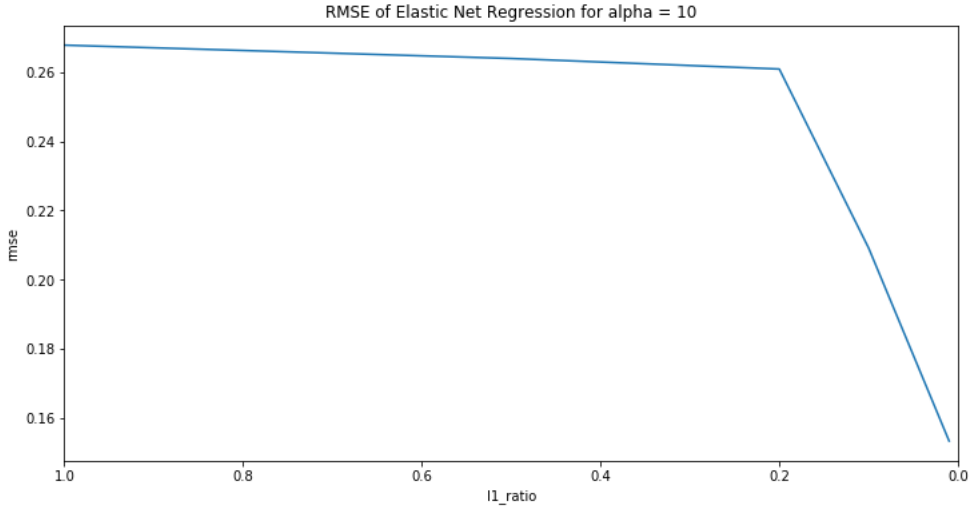

FIGURE 2.6 – L'erreur quadratique moyenne de la régression Elastic Net pour alpha = 10

Voyons maintenant comment le R_carré évolue en fonction de $l1_{ratio}$ quand $\alpha = 1$.

C'est presque comme dans le cas de Lasso, le R_square augmente quand $l1_{ratio}$ tend vers 0. Mais, toutefois, le modèle de Lasso est meilleur que Elastic Net selon ces paramètres.

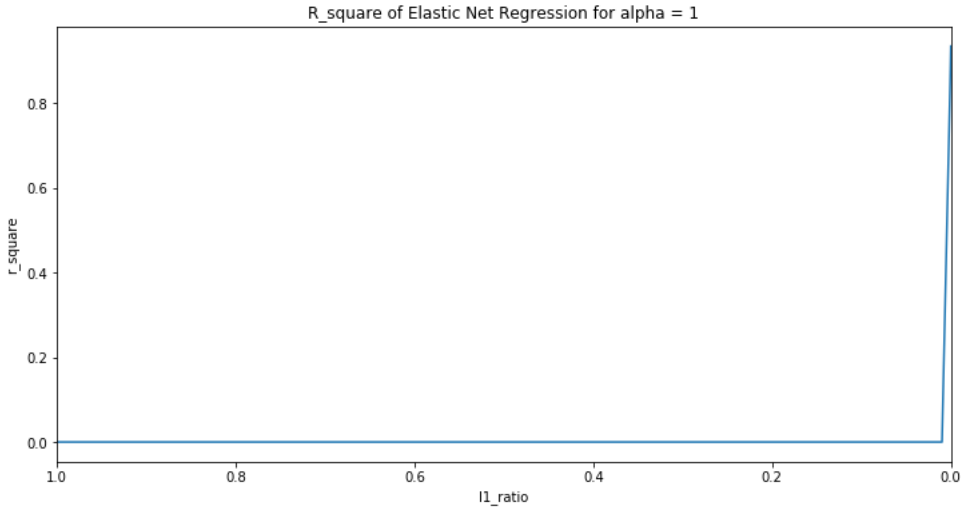

FIGURE 2.7 – Le R_carré de la régression Elastic Net pour alpha = 1

Conclusion D'après l'étude numérique faite sur les modèles de régularisation, nous avons obtenu que le modèle de Lasso est le meilleur sur cette base de donnée tout en prenant en compte que sa valeur de α est proche à 0, notamment, 0.001, vu qu'elle a une erreur quadratique moyenne de 0.122 comparant à Ridge qui est à 0.127, à Elastic Net qui est à 0.124 et même au modèle de régression linéaire non régularisé qui est à 0.163. Ainsi que le R_carré donne des résultats meilleurs quant au modèle lasso pour une valeur de α petite.

Conclusion

Ce projet nous a donné une bonne idée de la façon dont la régression de Ridge, de Lasso et de Elastic Net fonctionne dans la régularisation d'un modèle et nous avons pu consolider notre compréhension en les comparant et en essayant d'apprécier leurs cas d'utilisation spécifiques.

Et si nous essayons également les comparer avec d'autres approches : avec les progrès de l'apprentissage automatique, la régression de Ridge et de Lasso donne de très bonnes alternatives car elles offrent une meilleure sortie, nécessitent moins de paramètres de réglage et peuvent être automatisées dans une large mesure. Dans les cas d'utilisation typiques, il n'est pas difficile de voir pourquoi les techniques de sélection par étapes deviennent pratiquement très lourdes à mettre en œuvre dans les cas de forte dimensionnalité. Ainsi, le lasso offre un avantage significatif.

Par contre, avec la présence de variables hautement corrélées, Ridge fonctionne généralement bien même en présence de caractéristiques fortement corrélées car il inclura tous dans le modèle mais les coefficients seront répartis entre eux en fonction de la corrélation. Alors que Lasso sélectionne arbitrairement une caractéristique parmi les plus corrélées et réduit les coefficients du reste à zéro. De plus, la variable choisie change aléatoirement avec le changement des paramètres du modèle.

Ainsi, Elastic Net qui est une autre technique utile combinant à la fois la régularisation L1 et L2, peut être utilisé pour équilibrer les avantages et les inconvénients de la régression de Ridge et de Lasso. D'où, nous pouvons conclure que les techniques de régularisation dans la prédiction sont très utiles et que leurs analyses est assez crucial pour le choix des paramètres et de la meilleure régression.