Rapport machine learning

Régression linéaire + Ridge et Lasso

L'expérience de l'overfitting dans un modèle augmente le degré de liberté dans les modèles de régression, cet overfitting se réduit à l'aide de la régularisation avec deux techniques, régression Lasso et Ridge, présentées ci-après, permettent de réduire ces problèmes. Voir ce concept de régression Lasso, en quoi il est similaire et diffère de la régression Ridge.

Qu'est-ce que la régularisation ?

Le terme régularisation signifie rendre les conditions acceptables ou régulières.

Dans l'apprentissage machine, c'est une technique clé qui vise à limiter le « sur-apprentissage » **(*overfitting*)** et à contrôler l'erreur de type variance ***Standard Error*** (***SE***) pour aboutir à de meilleures performances, en réduisant l'erreur et en améliorant la généralisation de la solution. Il existe de nombreuses formes de régularisation, qui dépendent de l'objectif recherché et des hypothèses fixées sur le problème.

Types de régularisation

Il existe deux types de techniques de régularisation de base. Ces dernières sont la régression par crête (RIDGE) et la régression par Lasso. Leur méthode de pénalisation du coefficient est différente. Cependant, les deux techniques permettent de réduire l'overfitting d'un modèle.

Régression Ridge

La régression qualifiée de ***Ridge*** consiste à minimiser la somme des carrées des coefficients, le tout pondéré par un facteur noté λ. Ridge est représenté par la formule ci-après :

$$\beta^*_{ridge} = \arg\min_{\beta} \sum_{i=1}^{n}(y_i - x_i^T\beta)^2 + \overbrace{\lambda \sum_{j=1}^{d} |\beta_j^2|}^{Régularisation\ L2}$$

Ridge utilise la pénalité ou la régularisation L2 qui diminue l'impact sur les coefficients pour être maîtrisé sans que ces coefficients soient ramenés à 0.

> ⚠️ **Note importante :**
>
> Si les variables indépendantes x ne sont pas sur la même échelle, nous devons modifier ces valeurs pour obtenir une moyenne de 0 et une variance égale à 1 par la méthode de standardisation (étape de feature scaling).

Régression Lasso

La régression ***Lasso*** consiste à minimiser la somme des valeurs absolus des coefficients, le tout pondéré par un facteur noté λ. Lasso est représenté par la formule suivante :

$$\beta^*_{lasso} = \arg\min_{\beta} \sum_{i=1}^{n}(y_i - x_i^T\beta)^2 + \overbrace{\lambda \sum_{j=1}^{d} |\beta_j|}^{Régularisation\ L1}$$

Lasso utilise la pénalité ou la régularisation L1 en réalisant une ***variable selection*** pour l'optimisation, c'est-à-dire, qu'il va éteindre totalement certaines variables en mettant leur valeur à 0.

En quoi les régressions Ridge et Lasso sont différentes ?

Les régressions Ridge et Lasso utilisent deux fonctions de pénalités différentes pour la régularisation. Ridge utilise la régularisation L2, tandis que Lasso utilise la régularisation L1.

Dans la régression Ridge, la pénalité est égale à la somme des carrés des coefficients et dans le Lasso, elle est la somme des valeurs absolues des coefficients.

La régression Lasso, il s'agit du rétrécissement vers zéro en utilisant une valeur absolue (pénalité ou technique de régularisation L1) plutôt qu'une somme de carrés

(pénalité ou technique de régularisation L2).

Comme nous savons que dans la régression Ridge, les coefficients ne peuvent pas être nuls. Ici, nous considérons soit tous les coefficients, soit aucun des coefficients, alors que la technique de l'algorithme de régression Lasso effectue simultanément et automatiquement le rétrécissement des paramètres et la sélection des caractéristiques, car elle annule les coefficients des caractéristiques colinéaires. Cela permet de sélectionner la ou les variables parmi les n variables données tout en effectuant une régression Lasso plus facilement et plus précisément.

Lasso n'est pas adaptée si le nombre de variables indépendantes noté p est plus grand que le nombre d'enregistrements n (situation plutôt rare) alors Lasso ne sélectionnera que n variables indépendantes.

Pour finir, s'il y a deux variables ou plus, et qu'elles sont colinéaires, alors Lasso sélectionnera l'une d'entre elles de manière aléatoire ce qui peut compliquer l'interprétation des données.

L'implémentation de Ridge et LASSO avec Python + visualisation

Afin d'appliquer et de visualiser ces deux modèles, on a choisi un cas d'étude sur des données des salaires de joueur du Baseball [https://rdrr.io/cran/ISLR/man/Hitters.html].

Modèle Ridge et LASSO :

1. On importe les librairies Python nécessaires

```
# Importer les librairies
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import Lasso, LassoCV, Ridge, RidgeCV
from sklearn.metrics import mean_squared_error
```

2. Nous importons le dataset choisi pour cette implémentation « Hitters.csv » qui contient (322 lignes et 20 colonnes). Ensuite, on a fait une analyse en supprimant les valeurs manquantes et en traitant chaque variables. Apres cela, on sélectionne les « features » et le « target ». Par la suite, on transforme les données en données d'entrainement et de test, et on applique leur mise à l'échelle. **(Annexe 2).**

3. Dans cette partie, on initialise les deux modèles (Ridge et Lasso) et on ajuste les données d'entrainement.

```
# Initialisation du modèle Lasso
reg_lasso = Lasso(alpha=1)
reg_lasso.fit(X_train, y_train)
```

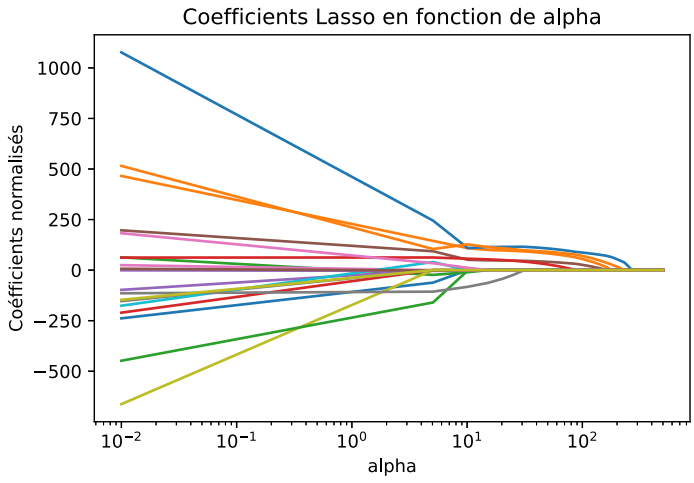
```
# Initialisation du modèle Ridge
reg_ridge = Ridge(alpha=1)
reg_ridge.fit(X_train, y_train)
```

4. Après l'ajustement des données, on fait la prédiction dans un jeux de données d'entrainement et puis dans un jeu de test.

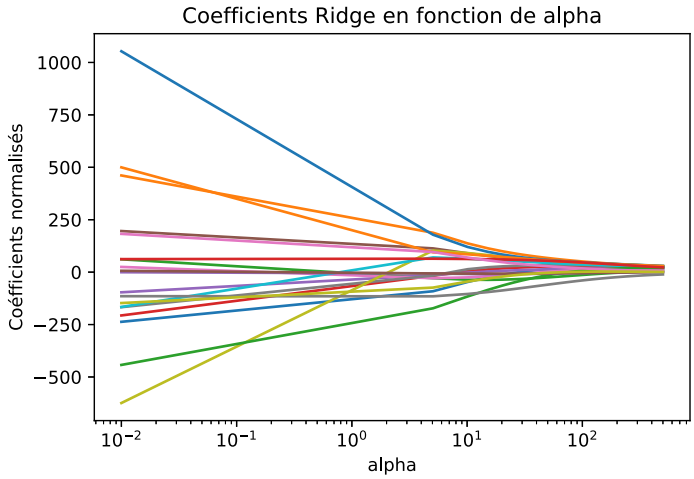
```
# Prediction Ridge
pred_ridge = reg_ridge.predict(X_test)
mse_ridge = mean_squared_error(y_test, pred_ridge)
print(f"MSE Ridge : {round(mse_ridge, 2)}")

# Prediction Lasso
pred_lasso = reg_lasso.predict(X_test)
mse_lasso =mean_squared_error(y_test, pred_lasso)
print(f"MSE Lasso : {round(mse_lasso, 2)}")
```

5. Afin de comprendre l'influence de α, on affiche dans une figure les coefficients en fonctions de α (*max_iter* : le nombre maximum d'itérations). Pour simplifier notre implémentation, on va mettre dans une fonction en mettant en paramètre Ridge ou Lasso et afficher les résultats correspondants. **(<u>Annexe 3</u>)**
6. Appel aux fonctions

```
model_lasso = Lasso(max_iter=10000)
PlotCoefficientsEnFoctionDeAlpha(model_lasso, "Lasso")
```



```
model_ridge = Ridge(max_iter=10000)
PlotCoefficientsEnFoctionDeAlpha(model_ridge, "Ridge")
```



Un lien pour executer le programme sur google collab : https://drive.google.com/file/d/1m14y6FGBXUce31SYbvf2ge6B6zo9rYAu/view?usp=sharing

Annexes

Annexe 1 : *Régression linéaire*

La ***régression linéaire simple*** est représentée par l'équation suivante :

$$f(x) = \beta_0 + \beta_1 x$$

La ***régression linéaire multiple*** est représentée par l'équation suivante :

$$f(x;\theta) = \beta_0 + \sum_{j=1}^{d} \beta_j x_j \qquad with\ \theta = \{\beta_j\}_{j=0,\dots,d}$$

```
# importer la librairie "LinearRegression" depuis le module "sklearn.linear_model"
from sklearn.linear_model import LinearRegression

# importer les données du dataset
donnees = pd.read_csv('dataset.csv')
```

```
# créer l'objet depuis la classe "LinearRegression"
model = LinearRegression()

# créer "y" et "X" depuis le dataframe "donnees"
y = donnees.Sales
X = donnees.columns.drop('Sales')

model.fit(X,y)
```

Annexe 2 : *Traitement des données*

```
# Importer le dataset
df = pd.read_csv("./Hitters.csv")
df.shape
# Traitement valeurs manquantes
df = df.dropna()
# Traitement des variables catégoriques
dummies = pd.get_dummies(df[['League', 'Division','NewLeague']])
X_numerical = df.drop(['Salary', 'League', 'Division', 'NewLeague'], axis=1).astype('float64')
list_numerical = X_numerical.columns
# Features and Target
y = df['Salary']
X = pd.concat([X_numerical, dummies[['League_N', 'Division_W', 'NewLeague_N']]], axis=1)
# Repartition des données en Train et Test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=10)
# Mise à l'echelle des données
scaler = StandardScaler().fit(X_train[list_numerical])
X_train[list_numerical] = scaler.transform(X_train[list_numerical])
X_test[list_numerical] = scaler.transform(X_test[list_numerical])
```

Traitement des données

Annexe 3 : *Fonction d'affichage de graph*

```
def PlotCoefficientsEnFoctionDeAlpha(model, model_name):
    alphas = np.linspace(0.01,500,100)
    coefs = []
    for a in alphas:
        model.set_params(alpha=a)
        model.fit(X_train, y_train)
        coefs.append(model.coef_)
    ax = plt.gca()
    ax.plot(alphas, coefs)
    ax.set_xscale('log')
    plt.axis('tight')
    plt.xlabel('alpha')
    plt.ylabel('Coéfficients normalisés')
    plt.title(f"Coefficients {model_name} en fonction de alpha")
    plt.savefig(f"{model_name}.svg")
```

Fonction d'affichage de graph