Matière : **Machine Learning**

DSI 3

Chapitre IV : La Regression

Enseignantes :

Naïma Halouani
Hounaïda Moalla
ISET Sfax

Objectifs

- Découvrir la formulation d'un problème de régression et le fonctionnement d'un algorithme de régression linéaire simple et multiple en Python.
- Évaluer les modèles avec un ensemble de métriques spécifiques pour la régression linéaire.

Plan

1. Définition de RL
2. Applications
3. Principe
4. Types
5. Régression Linéaire simple
   - Principe
   - Implémentation
6. Régression Linéaire multiple
   - Backword Elimation
     1. Principe
     2. Etapes
     3. Implémentation
7. Les métriques de RL

Définition

La régression appartient à la classe des tâches d'**apprentissage supervisé** où la variable cible (target) est **continue**.

La régression Linéaire est un algorithme qui permet à partir des variables dites explicatives (***X***) de prédire la variable cible la variable à expliquer (***Y)***

Applications

Croissance économique
La régression linéaire est utilisée pour déterminer la **croissance économique** d'un pays ou d'un État au cours du trimestre à venir.

Prix du produit
La régression linéaire peut être utilisée pour prédire quel sera le **prix d'un produit** à l'avenir, si les prix augmenteront ou diminueront.

Ventes de logements
La régression linéaire peut être utilisée pour estimer le **nombre de maisons** qu'un constructeur vendra au cours des prochains mois et à quel **prix**.

Prédictions de score
La régression linéaire peut être utilisée pour prédire le **nombre de points** qu'un joueur de baseball marquera dans les matchs à venir en fonction des performances précédentes.

Principe

L'objectif est de trouver **une fonction de prédiction** ou une **fonction coût** qui décrit la relation entre ***X*** et ***Y.***

c'est-à-dire qu'à partir de valeurs connues de ***X***, on arrive à donner une prédiction des valeurs de ***Y***.

La fonction recherchée est de la forme :

Y= h(X) avec ***h(X)*** une fonction linéaire

Principe

Chercher à quel point les h(x(i)) sont proches des y(i) correspondants.

Minimiser la fonction coût :

$$J(\theta) = \frac{1}{2}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2$$

Avec :
- x_i représente une ligne d'entrée
- n est le nombre d'entrées,
- Θ_i sont des constantes

comment choisir les paramètres θ ?

modifier itérativement θ pour rendre J(θ) plus petit, jusqu'à ce que, la fonction converge vers une valeur de qui minimise J(θ)

Types

En fonction du nombre d'entités en entrée, la régression linéaire peut être de deux types :

Regression

SLR : Simple Linear Regression

MLR : Multi-linear Regression

Dans **SLR**, on a **une seule variable d'entrée** sur la base de laquelle nous prédisons la variable de sortie.

Dans **MLR**, nous prédisons la sortie en fonction de **plusieurs entrées**.

- Les **variables d'entrée** sont appelées variables **indépendantes/prédictives**.
- La **variable de sortie** est appelée **variable dépendante**.

SLR : Principe

L'équation pour SLR est $y=b_o+b_1x+\epsilon$

où,

- Y est la variable dépendante,
- X est le prédicteur,
- bo, b1 sont les coefficients/paramètres du modèle,
- et Epsilon(ϵ) est une variable aléatoire appelée Error Term .

SLR : Implémentation

1. Créer le modèle

```
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
regressor.fit(X_train,y_train)
```

2. Afficher valeur de ϵ

```
regressor.intercept_
```

3. Afficher les valeurs des coéfficients b0, b1, ...

```
regressor.coef_
```

SLR : Implémentation

4. Prédire les données de test

```
y_pred = regressor.predict(X_test)
```

5. Comparer les valeurs prédites et les valeurs réelles des données test

```
for i in range(10):
    print(y_test[i],y_pred[i])
```

MLR : Multiple Linear Regression

Types

La régression linéaire multiple dispose de plusieurs techniques pour construire un modèle efficace, à savoir :

- All in
- ***Backword elimination***
- Forword selection
- Bidirectional elimination
- Score comparison

MLR : Backword Elimination

Principe

L'élimination en amont est une technique de sélection de fonctionnalités lors de la création d'un modèle d'apprentissage automatique. Il est utilisé pour supprimer les caractéristiques qui n'ont pas d'effet significatif sur la variable dépendante (cible/la prédiction de la sortie).

MLR : Backword Elimination

Les étapes de BE

1. Sélectionnez un niveau de signification pour rester dans le modèle (par exemple SL = 0,05)
2. Ajuster le modèle avec tous les prédicteurs possibles (all features)
3. Considérez la variable avec la **valeur-p** la plus élevée. **Si P>SL**, passez à **l'étape 4** sinon aller à **Fin**
4. Supprimer la variable
5. Ajustez le modèle sans cette variable et répétez l'étape 3 jusqu'à ce que la condition devienne fausse.
6. **Fin**

MLR : Backword Elimination

Implémentation

1. Importation de la bibliothèque

```
import statsmodels.api as sm
```

2. Ajout d'une colonne dans la matrice de caractéristiques

Dans l'équation LR (1) , il existe un terme constant b_0 , mais ce terme n'est pas présent dans notre matrice de caractéristiques, nous devons donc l'ajouter manuellement. Nous allons ajouter une colonne ayant des valeurs $x_0 = 1$ associée au terme constant b_0 .

```
X = np.append(arr = np.ones((50,1)), values= X, axis=1)
```

MLR : Backword Elimination

Implémentation

3. créer un nouveau vecteur de caractéristiques x_opt
x_opt ne contiendra qu'un ensemble de caractéristiques indépendantes qui affectent de manière significative la variable dépendante.

```
x_opt=x [:, [ 0 , 1 , 2 , 3 , 4 , 5 ]]
```

4. Ajuster le modèle avec tous les prédicteurs possibles Classe OLS)
La régression des moindres carrés ordinaires (OLS) est une technique courante pour estimer les coefficients des équations de régression linéaire qui décrivent la relation entre une ou plusieurs variables indépendantes et une variable dépendante (régression linéaire simple ou multiple).

```
regressor_OLS = sm.OLS ( endog=y, exog=X_opt ).fit()
```

MLR : Backword Elimination

Implémentation

5. Calculer les p-valeur des xi et les comparer à SL (seuil)

p-valeur est le résultat d'un calcul de probabilité d'une variable dans un test statistique**. p-valeur** est généralement comparée à un **seuil.**

Etapes 1 et 2

Nous utiliserons la méthode **summary()** pour obtenir le tableau récapitulatif de toutes les valeurs

```
regressor_OLS.summary()
```

	coef	std err	t	P>|t|	[0.025	0.975]
const	5.013e+04	6884.820	7.281	0.000	3.62e+04	6.4e+04
x1	198.7888	3371.007	0.059	0.953	-6595.030	6992.607
x2	-41.8870	3256.039	-0.013	0.990	-6604.003	6520.229
x3	0.8060	0.046	17.369	0.000	0.712	0.900
x4	-0.0270	0.052	-0.517	0.608	-0.132	0.078
x5	0.0270	0.017	1.574	0.123	-0.008	0.062

Etape 3

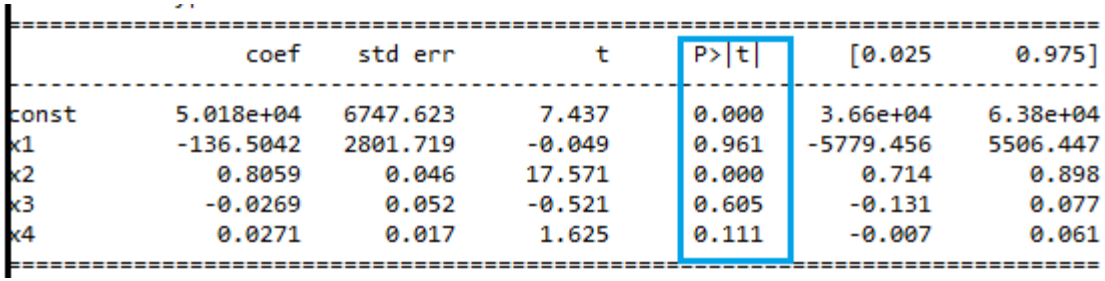
Dans le tableau, nous choisirons la valeur p la plus élevée, qui correspond à x2 = 0,990. Maintenant, nous avons la valeur p la plus élevée qui est supérieure à la valeur SL, nous allons donc supprimer la variable x2 (variable factice) du tableau et va remonter le modèle. Ci-dessous le code à exécuter :

MLR : Backword Elimination

Implémentation

Etape 4 et 5

```
x_opt=x[:, [ 0 , 1 , 3 , 4 , 5 ]]
regressor_OLS=sm.OLS(endog = y, exog=x_opt).fit()
regressor_OLS.summary()
```



	coef	std err	t	P>|t|	[0.025	0.975]
const	5.018e+04	6747.623	7.437	0.000	3.66e+04	6.38e+04
x1	-136.5042	2801.719	-0.049	0.961	-5779.456	5506.447
x2	0.8059	0.046	17.571	0.000	0.714	0.898
x3	-0.0269	0.052	-0.521	0.605	-0.131	0.077
x4	0.0271	0.017	1.625	0.111	-0.007	0.061

Etape 3

Comme nous pouvons le voir dans l'image de sortie, il reste maintenant cinq variables. Dans ces variables, la valeur p la plus élevée est de 0,961. Nous allons donc le supprimer à la prochaine itération.

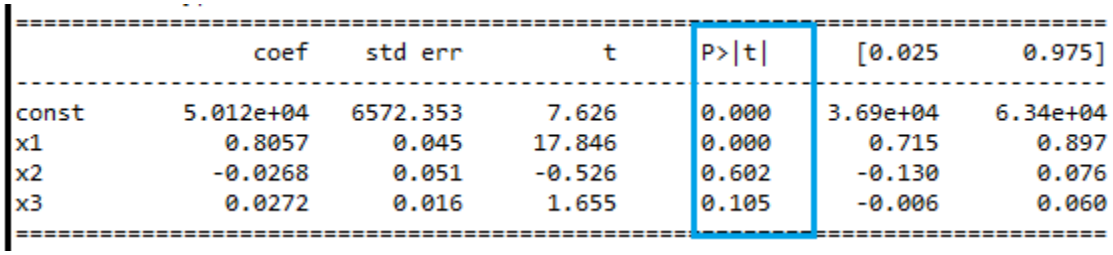
Maintenant, la prochaine valeur la plus élevée est 0,961 pour la variable **x1**, qui est une autre variable fictive.

MLR : Backword Elimination

Implémentation

Etape 4 et 5

```
x_opt= x[:, [ 0 , 3 , 4 , 5 ]]
regressor_OLS=sm.OLS(endog = y, exog=x_opt).fit()
regressor_OLS.summary()
```



	coef	std err	t	P>|t|	[0.025	0.975]
const	5.012e+04	6572.353	7.626	0.000	3.69e+04	6.34e+04
x1	0.8057	0.045	17.846	0.000	0.715	0.897
x2	-0.0268	0.051	-0.526	0.602	-0.130	0.076
x3	0.0272	0.016	1.655	0.105	-0.006	0.060

Etape 3

Dans l'image de sortie ci-dessus, nous pouvons voir que la variable factice (x2) a été supprimée. Et la prochaine valeur la plus élevée est .602, qui est toujours supérieure à 0.5, nous devons donc la supprimer.
Nous allons maintenant supprimer les dépenses d'administration qui ont une valeur p de 0,602 et réadapter à nouveau le modèle.

MLR : Backword Elimination

Implémentation

Etape 4 et 5

```
x_opt=x[:, [ 0 , 3 ,  5 ]]
regressor_OLS=sm.OLS(endog = y, exog=x_opt).fit()
regressor_OLS.summary()
```

	coef	std err	t	P>|t|	[0.025	0.975]
const	4.698e+04	2689.933	17.464	0.000	4.16e+04	5.24e+04
x1	0.7966	0.041	19.266	0.000	0.713	0.880
x2	0.0299	0.016	1.927	0.060	-0.001	0.061

Etape 3

Comme indiquer dans le resultat de summary(), il reste une variable, qui est les **dépenses marketing** ayant une valeur p élevée **(0,060)** . Nous devons donc l'enlever.

MLR : Backword Elimination

Implémentation

Etape 4 et 5

```
x_opt=x[:, [ 0 , 3 ]]
regressor_OLS=sm.OLS(endog = y, exog=x_opt).fit()
regressor_OLS.summary()
```

	coef	std err	t	P>|t|	[0.025	0.975]
const	4.903e+04	2537.897	19.320	0.000	4.39e+04	5.41e+04
x1	0.8543	0.029	29.151	0.000	0.795	0.913

Ainsi, seule la **variable indépendante R&D** est une variable significative pour la prédiction. Nous pouvons donc maintenant prédire efficacement en utilisant cette variable. ***Etape 6*** → ***Fin***

Réduction du problème à SLR

Métriques de LR

ce sont les métriques spécifiques pour le modèle de régression

- **Mean Squared Error (MSE),**
- **Mean Absolute Error (MAE),**
- **Root Mean Squared Error (RMSE) and**
- **R squared (R^2)**

n : nombre lignes de la base → nombre d'observation

MSE=1/n* sum(y_test[i]-y_pred[i])2**

MAE=1/n*sum(|y_test[i]-y_pred[i]|)

RMSE=SQRT(MSE)

R^2=1- sum(y_test[i]-y_pred[i])2/sum(y_test[i]-y_avg[i])**2**

Métriques de LR

Implémentation

```
from sklearn.metrics import mean_squared_error,
mean_absolute_error, r2_score
mean_squared_error(y_test, y_pred)
```

```
def rmse(targets, predictions):
    return np.sqrt(((predictions - targets) ** 2).mean())
```

```
rmse(y_test, y_pred)
```

```
mean_absolute_error(y_test, y_pred)
```

```
r2_score(y_pred,y_test)
```

Autres algorithmes

Régressions Régularisées (pour éviter l'overfitting)

Ridge Regression, Lasso Regression, Elastic Net

Modèles Basés sur des Arbres

Decision Tree Regression, Random Forest Regression, Gradient Boosting Machines (GBM, XGBoost, LightGBM, CatBoost)

Modèles Basés sur les Plus Proches Voisins

K-Nearest Neighbors Regression (KNN)

Modèles Bayésiens

Bayesian Regression

...