Algorithme de régression PLS : Présentation et implémentation

I. Introduction

La régression PLS a été développé comme une solution algorithmique dans les années 80 par deux chimio-métriciens Svante Wold et Harald Martens. Il s'agit d'une méthode d'analyse multivariée qui combine les avantages de la régression multiple et de l'analyse en composantes principales (PCA). Elle a été introduite dans le domaine de la chimio-métrie et a attiré avec le temps un nombre important de chercheurs et statisticiens. La régression PLS est d'une certaine manière très proche de l'analyse en composantes principales.

Comme l'ACP, la régression PLS vise à projeter l'information commune à X et Y sur des composantes non corrélées. Contrairement à l'ACP, elle prend en compte la variable dépendante Y dans l'extraction des composantes principales. Elle est particulièrement utile dans les situations où il y a une forte multi colinéarité entre les variables indépendantes et où le nombre de variables est plus élevé que le nombre d'observations.

De nos jours, il existe plusieurs versions de l'algorithme de PLSR. Dans la suite, nous présenterons brièvement l'algorithme standard de la méthode ainsi que notre propre implémentation de l'algorithme sous Python.

II. Principe de la méthode

L'idée de la méthode est de chercher des composantes $Z = [z_1, \dots, z_h]$ tels que :

- Ils approximent les prédicteurs : $X = ZP^T + E$
- Ils prédisent également la variable d'intérêt : $y = Zd + e$

Où :

- P est la matrice de chargement des X
- E est la matrice de résidus de X
- d est le vecteur de coefficient de régression de la variable d'intérêt
- e est le vecteur de résidus de y

Le modèle pour l'espace des X est formulé comme suit :

$$\hat{X} = ZP^T$$

Le modèle de prédiction pour la variable d'intérêt y est donné par :

$$\hat{y} = Zd$$

L'algorithme est itératif sur le nombre de composantes principales et le nombre optimal de composantes es déterminé par Cross Validation ou par critère d'information.

III. Etapes explicites de l'algorithme

Etape 1 : Standardisation des données

Etape 2 : Initialisation des matrices

- Initialiser les matrices W, Z, P, D, B à zéro. La matrice W est la composante permettant d'obtenir Z.

L'utilité de ces matrices réside dans le processus itératif de PLS, où à chaque étape, on cherche à maximiser la covariance entre les scores des variables indépendantes et dépendantes pour extraire les composantes qui capturent l'information commune entre X et Y. Ces matrices sont mises à jour à chaque itération pour atteindre cet objectif. Une fois que toutes les composantes ont été obtenues, elles sont utilisées pour prédire de nouvelles observations. Les poids dans ces matrices sont ajustés de manière à optimiser la relation entre les deux ensembles de variables.

Etape 3 : Boucle PLS

- Pour chaque composante :
- Calculer le vecteur de poids normalisé w qui correspond à la covariance entre X et y
- Calculer la première composante principale z : $z_1 = X_0 w_1$
- Calculer le vecteur de chargement des variables indépendantes p : $p_1 = X_0^T z_1 / z_1^T z_1$
- Calculer le coefficient de régression d : $d_1 = y_0^T z_1 / z_1^T z_1$
- Déflater de la matrice X la variance expliquée par la première composante principale afin d'obtenir la matrice résiduelle des X
- Faire la même déflation pour y

La boucle est itérative sur le nombre de composantes principales définies.

Etape 4 : Prédiction

- Utiliser les composantes obtenues pour prédire de nouvelles observations

Etape 5 : Validation

- Utiliser des techniques de validation comme la validation croisée ou les critères d'information pour évaluer les performances du modèle.

IV. Implémentation sous Python

Dans le cadre de l'implémentation de l'algorithme, nous nous basons sur un jeu de données simulé et nous comparons les résultats de notre algorithme avec celui obtenu à partir de la fonction du package PLSRegression et en utilisant la cross validation pour déterminer le nombre optimal de composantes.

Nous avons simulé aléatoirement une matrice de 10 variables indépendantes de 100 observations. Pour rendre plus adaptée notre jeu de données au contexte d'application de la méthode, nous avons aléatoirement créé une forte multi colinéarité entre nos variables indépendantes. A partir de cette matrice nous avons simulé notre variable d'intérêt suivant une relation linéaire.

Résultats obtenus

Les résultats de validation de notre algorithme PLSR pour un nombre de 5 composantes se présente suit :

Mean Squared Error (Component 1): 1.5595298389133752

Mean Squared Error (Component 2): 1.0810821825971557

Mean Squared Error (Component 3): 0.7270839818324687

Mean Squared Error (Component 4): 0.48539317637862256

Mean Squared Error (Component 5): 0.34687280754700095

On remarque à travers ces résultats qu'à priori notre algorithme fonctionne bien car le MSE diminue au fur et à mesure qu'augmente le nombre de composantes.

Après vérification à partir de la fonction intégrée de Python et en se basant sur une cross-validation pour obtenir le nombre optimal de composantes, on obtient : Mean Squared Error : 2.3232196877329447 avec n_components = 1.

Ce résultat montre une meilleure performance de notre algorithme comparé à celui de la fonction intégrée même en considérant le nombre optimal de composantes. Cependant les résultats restent assez proche et renforce notre assurance sur le bon fonctionnement de l'algorithme.