République du Sénégal

Un peuple - Un but - Une foi

**

Ministère de l'Economie, du Plan et de la Coopération

****

Agence Nationale de la Statistique et de la Démographie (ANSD)

******

Ecole Nationale de la Statistique et de l'Analyse Economique (ENSAE)

TP DE DATA MINING
SUR LA NORMALISATION DES DONNEES

Rédigé par :
DEKOU VESSOU Jeff- Maruis
Elève Analyste Statisticien

M. GOMIS
Informaticien
Enseignant de DATA MINING

Novembre 2022

Tables de matières

I. Présentation des normalisations et principes 3
a. La Standardisation ou normalisation Z-score 3
b. La normalisation robuste.....4
c. La Transformation 'Box-Cox' 4
d. La transformation au vecteur unitaire 4
II. Différents codes pythons et explications des codes 5
a. La Standardisation ou normalisation 5
b. La Transformation 'Box-Cox' 5
c. La normalisation robuste.....6
d. La transformation au vecteur unitaire 7
e. La normalisation avec la fonction « Normalize » 8
III. Comparaison des types de normalisations.....9
IV. Intérêt des normalisations 9

I. Présentation des normalisations et principes

Nous allons nous intéresser à certains types de normalisations précises dont nous allons expliquer les principes sans oublier d'énoncer la façon dont elles sont calculées

Principalement dans notre travail nous allons mettre l'accent sur quelques normalisations dont :

- La normalisation Standard encore appelé la Standardisation ou normalisation Z-score
- La Normalisations robuste sous ses deux formes
- La Transformation 'Box-Cox'
- La normalisation au vecteur Unitaire
- La normalisation avec la fonction « Normalize »

Nous allons des à present présenter ces normalisations en première partie et ensuite joindre les codes pythons avec des explications de nos démarches.

a. La Standardisation ou normalisation Z-score

Encore appelé normalisation standard, elle donne à une variable une moyenne nulle et un écart-type de 1, similaire à une Loi normale centrée réduite.

On la calcule en retranchant à la variable sa moyenne et en divisant le résultat par l'écart-type :

$$\text{Stand} = \frac{x-u}{s}$$

Où x est la variable de départ, u sa moyenne et s son écart type.

Cette transformation assigne à la variable une majorité de valeurs comprises entre $[-1,1]$, le résultat étant moins dépendant des valeurs aberrantes, contrairement à la normalisation min-max.

b. La normalisation robuste

Cette technique transforme chaque variable en étant peu sensible aux outliers (valeurs aberrantes).

Le procédé est le suivant : on soustrait aux valeurs de la variable la médiane (la médiane est une valeur de la variable qui permet de séparer ses valeurs en deux parties égales) et on divise par l'écart interquartile IQR (IQR est une mesure de dispersion qui s'obtient en faisant la différence entre le troisième et le premier quartile) la formule est la suivante :

$$X\_rob = \frac{x - mediane}{IQR}$$

En python on peut l'effectuer de deux manières : Soit en utilisant RobustScaler () ou MaxAbsScaler ()

c. La Transformation 'Box-Cox'

La transformation Box-Cox fait partie des transformations appelées "power transform". Elles utilisent la puissance ou le log pour transformer rendre linéaire une courbe exponentielle. Elle est en grande partie utilisée pour annuler l'effet de l'hétéroscédasticité et rapprocher nos donner de la normalité. Elles sont plus utilisées lorsque os series ont l'air de suivre une loi exponentielle.

d. La transformation au vecteur unitaire

Si ce qui nous intéresse se rapporte à la direction du point p et non à sa distance à l'origine, on peut utiliser la normalisation à l'unité :

$$\mathbf{X\_Unit} = \frac{p}{L2(P)} \quad \mathbf{Ou} \quad \mathbf{L2(P)} = Racine\ carré\left(\sum_0^n P^2\right)$$

Elle utilise une notion mathématique : La norme d'un vecteur pour normaliser.
Elle est souvent utilisée dans les données du NLP et de la classification en général.

II. Différents codes pythons et explications des codes

a. La Standardisation ou normalisation

```
In [ ]: from sklearn.preprocessing import StandardScaler
        #visualisatio_des_normalisations_standard
        scaler1= StandardScaler()
        X_trans1=scaler1.fit_transform(X)
```

Pour débuter notre code, nous sommes allés chercher dès la première ligne le module qui permet d'effectuer cette normalisation :il s'agit de StandardScaler ().

Ce module se trouve dans la bibliothèque "sklearn" ; d'où l'importation effectué à la première ligne dans cette bibliothèque.

Ensuite nous avons appelé spécialement notre module que nous avons capter dans l'objet 'Scaler1'.

A la dernière ligne : Nous appliquons notre module à la nos données en adaptant cette dernière par le biais de l'instruction "fit_transform' et on obtient nos données transformées et normalisées

b. La Transformation 'Box-Cox'

```
#visualisatio_des_normalisation transformpower
from sklearn.preprocessing import PowerTransformer
scaler4= PowerTransformer()
X_trans4=PowerTransformer().fit_transform(X)
#print(X_trans4)
#plt.scatter(X_trans4[:,2],X_trans4[:,3])
```

Cette normalisation a une syntaxe particulière par rapport aux autres. En effet transform_power est constitué de deux méthodes : le méthode 'Box-Cox' et la méthode 'Joe Johnson'

Ces deux méthodes effectuent tous la normalisation mais la difference est que la méthode 'Box-Cox' ne s'applique qu'aux données positives contrairement à l'autre qui s'applique aux données positives et négatives. Pour plus de précisions et en fonction de nos objectifs, spécifier la méthode à utiliser avec la syntaxe suivante :

Transform power (data, method='Box-cox', *, standardize=True, copy=True)

Concernant notre code nous n'avons pas préciser la méthode et elle applique par défaut Joe Johnson la première ligne du code : aller chercher des la première ligne le module qui permet d'effectuer cette normalisation :il s'agit de Transform_Power ().
Ce module se trouve dans la bibliothèque "sklearn", d'où l'importation effectuée à la première ligne dans cette bibliothèque.
Ensuite nous avons appelé spécialement notre module que nous avons capter dans l'objet 'Scaler4'.

Nous appliquons notre module à la nos données transformé "fit_transform'
et on obtient nos données transformées et normalisées.

c. La normalisation robuste

En python on peut l'effectuer de deux manières : Soit en utilisant RobustScaler () ou MaxAbsScaler ()

- Première forme : RobustScaler

```
: #visualisatio_des_Normalisations robustes
from sklearn.preprocessing import RobustScaler
scaler2= RobustScaler()
X_trans2=scaler2.fit_transform(X)
#print(X_trans2)
```

Pour débuter notre code, nous sommes allés chercher dès la première ligne le module qui permet d'effectuer cette normalisation :il s'agit de RobustScaler ().

Ce module se trouve dans la bibliothèque "sklearn" ; d'où l'importation effectué à la première ligne dans cette bibliothèque.

Ensuite nous avons appelé spécialement notre module que nous avons capter dans l'objet 'Scaler2'.

A la dernière ligne : Nous appliquons notre module à la nos données en adaptant cette dernière par le biais de l'instruction "fit_transform' et on obtient nos données transformées et normalisées_X_trans2

- Deuxième forme : MaxAbsScaler

```
[80]: #visualisatio_des_Normalisations MaxAbsScaler ou Robuste
      from sklearn.preprocessing import MaxAbsScaler
      scaler6= MaxAbsScaler()
      X_trans6=scaler2.fit_transform(X)
      #print(X_trans6)
```

Pour débuter notre code, nous sommes allés chercher dès la première ligne le module qui permet d'effectuer cette normalisation :il s'agit de MaxAbsScaler ().

Ce module se trouve dans la bibliothèque "sklearn" ; d'où l'importation effectué à la première ligne dans cette bibliothèque.

Ensuite nous avons appelé spécialement notre module que nous avons capter dans l'objet 'Scaler6'.

A la dernière ligne : Nous appliquons notre module à la nos données en adaptant cette dernière par le biais de l'instruction "fit_transform' et on obtient nos données transformées et normalisées

d. La transformation au vecteur unitaire

```
#Normalisation au vecteur unitaire
import numpy as np
X_trans5= X / np.sqrt(np.sum(X**2))
#print(X_trans5)
#plt.scatter(X_trans5[:,2],X_trans5[:,3])
```

Cette normalisation n'a du moins rien à avoir avec notre bibliothèque sklearn.
Elle fait plutôt appel aux outils mathématiques. En premier lieu, nous avons importé de la bibliothèque Numpy (Bibliothèque permettant de faire des calculs numériques)
Et ensuite nous avons saisi la formule de façon manuel bien sûr, en se basant sur la formule pardonnée dans la premier partie (I).
En effet, faisant référence à notre formule d'en haut, on a par identification a notre code :
p=X
L(2P) = sqrt(sum(X**2))

Soit **X_Unit =** $\frac{X}{L2(X)}$ **Ou L2(X)=**$Racine\ carré(\sum_0^n X^2)$

Une fois cette formule appliquée, nous avons obtenus nos données transformées. Cette méthode est très utile dans les classifications.

e. La normalisation avec la fonction « Normalize »

```
from sklearn import preprocessing
X_trans3 = preprocessing.normalize(X)
#print(X_trans3)

#scaled_df = pd.DataFrame(d, columns=names)
#scaled_df.head()
```

Pour débuter notre code, nous sommes allés chercher dès la première ligne le module qui permet d'effectuer cette normalisation :il s'agit de 'preprocessing ()'. Ce module se trouve dans la bibliothèque "sklearn", d'où l'importation effectuée au départ dans cette bibliothèque. Ensuite nous appliquons directement ce module précédé de « preprocessing »

Nous venons de présenter quelques types de normalisations. Mais nous devons retenir qu'avoir beaucoup de méthodes de normalisation ne nous donne pas le libre doit de les appliquer comme l'on veut. Il se pose alors le problème de savoir

ou de se demander face a tous données la normalisation la plus adaptée à notre analyse. C'est ce qui fera d'ailleurs l'objet de notre dernière partie.

III. Comparaison des types de normalisations

A vrai dire l'on ne peut affirmer que telle normalisation ou telle normalisation est meilleure que l'autre car chacun à des avantages et des inconvénients.

Mais sur la base des documents et des recherches effectuées dans certains documents (Cités à la fin de notre travail) ; comparant la normalisation robuste, normalisation minmax et la normalisation Standard, la normalisation robuste semble être meilleure par rapport aux autres.

En effet ces deux premiers techniques (Normalisation standard et MinMax) techniques sont sensibles aux valeurs aberrantes, donc moins efficaces. Pour des variables ayant des valeurs aberrantes, il est préférable d'utiliser la normalisation robuste qui est peu sensible aux valeurs aberrantes.

Pour les autres normaisations _transform box-cox, cela dépendra de nos données et de l'objectif de nos analyses mais l'essentiel est de minimiser le nombre de valeurs abérantes possibles.

IV. Intérêt des normalisations

Les normalisations ont plusieurs objectifs.

- L'une de l'objectif de la normalisation est de modifier les valeurs des colonnes numériques de l'ensemble de données à une échelle commune, sans fausser les différences dans les plages de valeurs. Pour l'apprentissage automatique, chaque ensemble de données ne nécessite pas de normalisation. Il est requis uniquement lorsque les entités ont des plages différentes.

Exemple de cas pratique

Par exemple, considérons un ensemble de données contenant deux caractéristiques, l'âge et le revenu (x2). Où l'âge varie de 0 à 100 ans, tandis que le revenu varie de 0 à 100 000 et plus. Le revenu est environ 1 000 fois supérieur à l'âge. Ainsi, ces deux caractéristiques sont dans des gammes très différentes.

Lorsque nous effectuons une analyse plus approfondie, comme la régression linéaire multivariée, par exemple, le revenu attribué influencera intrinsèquement davantage le résultat en raison de sa valeur plus élevée. Mais cela ne signifie pas nécessairement qu'il est plus important en tant que prédicteur. Nous normalisons donc les données pour amener toutes les variables dans la même plage.

- La normalisation a également pour but de modifier les valeurs des colonnes numériques du jeu de données pour utiliser une échelle commune, sans que les différences de plages de valeurs ne soient faussées et sans perte d'informations.
- La normalisation des données permet également de rapprocher nous donner d'une même loi et souvent la loi normale avec moins de biais.

Bibliographies

Sources : Internet

- ***Sites visités** :*

- *https://www.alliage-ad.com/tutoriels-python/les-methodes-de-normalisation/*
- *https://scikitlearn.org/stable/modules/generated/sklearn.preprocessing.PowerTransformer.html*
- *https://www.arboretum.link/notes/mise-%C3%A0-l'%C3%A9chelle-des-donn%C3%A9es-standardisation-et-normalisation*
- *https://complex-systems-ai.com/analyse-des-donnees/normaliser-standardiser-redimensionner-vos-donnees/*