TP2 : DATA PREPROCESSING

PREPROCESSING ?

Apprendre a préparer un data set avant de construire le model de ML. Dans cette section nous allons :

- Importer les librairies ;
- Importer le data set ;
- Gérer les données manquantes ;
- Gérer les variables catégoriques ;
- Diviser le jeu de données en Training set et Test set ;
- Appliquer le feature scaling (le mise des données sous la même échelle)

Données : Data.csv

1- Importer les librairies

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

2- # Importer le data set puis extraire la matrice des variables dépendantes et le vecteur des de la variable indépendante

 # Importer le data set

```
dataset = pd.read_csv('Data.csv')
```

Variables indépendantes : country, age, salary informations sur des clients d'une entreprise.

Variables dépendante : purshased ; le client à acheter ou non un produit.

 # extraire la matrice des variables indépendantes et le vecteur des de la variable dépendante

```
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, -1].values
```

3- # Gérer les données manquantes

Données distribuées selon une loi normale et absence d'outliers, imputer par la moyenne dans le cas contraire imputer par la médiane

 # importer la classe Imputer

```
from sklearn.preprocessing import Imputer
```

 # instancier la classe Imputer en précisant la stratégie d'imputation

```
imputer = Imputer (missing_values = 'NaN', strategy = 'mean', axis = 0)
```

 # Lire l'objet imputer aux colonnes de X à imputer

```
imputer.fit(X[:, 1:3])
```

 # remplacer les valeurs manquantes par la moyenne des colonnes

```
X[:, 1:3] = imputer.transform(X[:, 1:3])
```

4- # Gérer les variables catégoriques

Pourquoi ?

Les modelés ML sont basés sur des équations mathématiques qui implémentent difficilement les variables non numériques

 # importer les classes LabelEncoder et OneHotEncoder

```
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
```

 # creer une instance de LabelEncoder pour la variable categorique de X

```
labelencoder_X = LabelEncoder()
```

 # fiter et transformer sur la colonne a transformer. il transforme la variable catégorique a trois niveau en valeurs numérique 0 - 1- 2 pour utiliser OneHotEncoder qui ne travail qu'avec des variables numériques

```
X[:, 0] = labelencoder_X.fit_transform(X[:, 0])
```

 # instancier OneHotEncoder pour l'encodager sous forme de demie variable 0 entre crocher pour l'indice la/des colonnes a encoder

```
onehotencoder = OneHotEncoder(categorical_features = [0])
```

 # fiter puis transformer

```
X = onehotencoder.fit_transform(X).toarray()
```

 # instancier LabelEncoder pour y

```
labelencoder_y = LabelEncoder()
```

 # fiter et transformer

```
y = labelencoder_y.fit_transform(y)
```

5- # Diviser le dataset entre le Training set et le Test set

Pourquoi ?

Le modèle de ML va apprendre le model des corrélations entre la variable indépendante et les variables dépendantes sur un sous data set du jeu de données, le training set (80% du jeu de données) mais pour vérifier qu'il n'y a pas de sur apprentissage ie que le modèle n'a pas appris par cœur le modèle des corrélations. On va le tester le test set (20% de nouvelle valeurs).

Méthode (pour tester le modèle d'apprentissage) :

Calculer la précision :

P1 = nombre d'erreur sur le training / nombre d'observation du training

P2 = nombre d'erreur sur le test / nombre d'observation du test

Si P1 ≈ P2 il y a un bon training sur training set c-a-d qu'il n'y a pas sur apprentissage.

 # importer la fonction train_test_split

```
from sklearn.model_selection import train_test_split
```

 # test_size = 0.2

 # train_size = 0.8

 # random_state = 0 pour que deux executions differentes donne le meme training et test set

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2,
random_state = 0)
```

6- # Feature Scaling

Pourquoi ?

Mettre toute les variables a la même échelle pour ne pas qu'une variable écrase l'autre dans les modèles de ML.

Ex : age peut etre ecraser par le salaire dans le modèle de ML

- Standardisation : xstand = (x-mean())/sd(x)
- Normalisation : xnorm = (x – min(x))/(max(x) – min(x))

A la fin de cette phases les données devraient tous etre compris entre -2 et +2

 # pour le feature scaling importons la classe StandardScaler

```
from sklearn.preprocessing import StandardScaler
```

 # instancier StandardScaler

```
sc = StandardScaler()
```

 # fiter et transformer le training et le test set

```
X_train = sc.fit_transform(X_train)
X_test = sc.fit_transform(X_test)
Liens utiles
```

Quelques liens

https://paris-fire-brigade.github.io/data-challenge/tutoriels.html

https://towardsdatascience.com/setup-an-environment-for-machine-learning-and-deep-learning-with-anaconda-in-windows-5d7134a3db10

https://www.jetbrains.com/pycharm/

https://salishsea-meopar-docs.readthedocs.io/en/latest/work_env/python3_conda_environment.html