TP03 IA

Dans le but d'appliquer notre première méthode de classification, il est primordial de préparer notre base de données. Le nettoyage des données manquantes, la normalisation des données et la division de notre base en une partie apprentissage et une partie teste et la spécification des variables dépendantes de celles indépendantes sont les étapes les plus importantes à suivre.

Entrée [28]:
```
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, OrdinalEncoder
from sklearn.neural_network import MLPClassifier
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score, recall_score, \
f1_score, classification_report,ConfusionMatrixDisplay
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import classification_report
file1= open ("E:/cours doctoraux/Cours essec/IA/TP/titanic/train.csv")
file2= open ("E:/cours doctoraux/Cours essec/IA/TP/titanic/test.csv")
titanic=pd.read_csv(file1, index_col='PassengerId')
titanic_test=pd.read_csv(file2, index_col='PassengerId')
file3= open ("E:/cours doctoraux/Cours essec/IA/TP/titanic/gender_submission.csv")
titanic_tY=pd.read_csv(file3, index_col='PassengerId')
```

Il suffit vous avez appliqué le prétraitement nécessaire pour l'élimination des données manquantes, vus pouvez suivre le suivant :

Entrée [35]:
```
#diviser la BD en variable cible et variables d'entrées
target = titanic['Survived']
inputs = titanic.drop(['Survived'],axis=1)
```

Division de la base, pour ce faire nous aurons besoin de la méthode train_test_split de scikit learn

Entrée [36]:
```
#Division de la BD en partie apprentissage et partie test 20%
x_train, x_test, y_train, y_test = train_test_split(inputs, target, test_size=0.2, random_state=365, stratify = target)
```

Entrée [37]:
```
y_train.value_counts(normalize = True)
```

Pour coder les variables cardinales en des valeurs numériques nous utilisons la méthode encoder de scikit learn

Entrée [38]:
```
#Définir un encodeur pour Target et Inputs
EnI = OrdinalEncoder()
EnT = LabelEncoder()
```

```
Entrée [ ]:
#Appliquer la transformation nécessaire
x_train_transf = enc_i.fit_transform(x_train)
x_test_transf = enc_i.transform(x_test)

y_train_transf = enc_t.fit_transform(y_train)
y_test_transf = enc_t.transform(y_test)
```

Par la suite, il vient l'étape de la normalisation qui représente une étape essentielle pour l'application de certaines méthodes de classification tel est le cas de SVM.

```
Entrée [70]:
#Normalisation des données dans l'intervalle [-1,1]
scaling = MinMaxScaler(feature_range=(-1,1)).fit(x_train_transf)
x_train_rescaled = scaling.transform(x_train_transf)
```

Maintenant nous pouvons classer les données. La première étape vise à définir une méthode MLP avec une seule couche cachée en spécifiant les différents paramètres.

```
Entrée [71]:
#Définition de MLP
ANN = MLPClassifier( solver='Adam', hidden_layer_sizes=(6,), random_state=1, max_iter=1000)
```

```
Entrée [74]:
#Model fitting
ANN.fit(x_train_rescaled, y_train_transf)
```

```
Entrée [84]:
#Tester le modèle
predictions = ANN.predict(x_test_rescaled)
```

Et maintenant nous allons évaluer notre modèle :

```
Entrée [85]:
print(classification_report(predictions, y_test_transf ))
```

Travail demandé : varier les paramètres de votre modèle et interpréter les résultats.