Data Mining : Arbre de décision

TP N : 1

Objectif

Un arbre de décision est un outil d'aide à la décision représentant un ensemble de choix sous la forme graphique d'un arbre. Les différentes décisions possibles sont situées aux extrémités des branches (les « feuilles » de l'arbre), et sont atteintes en fonction de décisions prises à chaque étape.
L'objectif de ce TP est d'appliquer l'algorithme d'arbre de décision sur l'ensemble de données « kyphosis.csv » afin de décider si un patient est atteint de la cyphose ou pas. Les attributs sont : "Age", "Number" et "Start". Les patients sont répartis selon s'ils sont malade ou pas, alors deux classes sont identifié : present et absent

Ce TP couvre les aspects suivants :

1. Comment diviser l'ensemble de données en un ensemble d'apprentissage et de test à l'aide de Scikit-Learn.
2. Comment construire un modèle d'arbre de décision à l'aide de Scikit-Learn.
3. Comment évaluer un arbre de décision avec Scikit-Learn.
4. Comment imprimer la représentation textuelle de l'arbre avec la méthode Scikit-Learn.
5. Comment visualiser les arbres de décision à l'aide de Matplotlib.

I. Importez les librairies python

Pour réaliser ce TP, on a besoin d'importer plusieurs librairies à savoir :

- Importez la bibliothèque **pandas** pour importer le contenu du fichier « kyphosis.csv »
  → `import pandas as pd`
- Importez la bibliothèque **matplotlib** pour tracer et visualiser des données sous formes de graphiques.
  → `import matplotlib.pyplot as plt`
- Importez **train_test_split** à partir de la bibliothèque **sklearn.model_selection** pour diviser l'ensemble de données en un ensemble d'apprentissage et de test.
  → `from sklearn.model_selection import train_test_split`
- Importez **DecisionTreeClassifier** à partir de la bibliothèque **sklearn.tree** pour construire l'arbre de décision.
  → `from sklearn.tree import DecisionTreeClassifier`
- Importez **classification_report, confusion_matrix** à partir de la bibliothèque **sklearn.metrics** pour construire l'arbre de décision.
  → `from sklearn.metrics import classification_report,confusion_matrix`

- Importez **tree** à partir de la bibliothèque Scikit-learn pour visualiser l'arbre de décision.

  → `from sklearn import tree`

II. Obtenez les données

1. Utilisez **pandas** pour lire le contenu du fichier « kyphosis.csv » en tant qu'un objet DataFrame appelée **df**.

   → `df = pd.read_csv('kyphosis.csv')`

2. Affichez les dimensions de l'ensemble de données
3. Affichez le contenu des premières et dernières lignes.
4. Explorez le contenu de « kyphosis.csv » en utilisant les méthodes info() et describe().

III. Analyse exploratoire des données

La visualisation des données est la discipline qui consiste à essayer de comprendre les données en les plaçant dans un contexte visuel. Python propose plusieurs excellentes bibliothèques graphiques contenant de nombreuses fonctionnalités différentes, à savoir :

- ✓ **Matplotlib** : bas niveau, offre beaucoup de liberté
- ✓ **Seaborn** : interface de haut niveau, excellents styles par défaut
- ✓ **ggplot** : basé sur le ggplot2 de R, utilise la grammaire des graphiques
- ✓ **Plotly** : peut créer des tracés interactifs

Dans ce TP, nous allons créer quelques graphiques de base à l'aide de Matplotlib.

Matplotlib est la bibliothèque de traçage Python la plus populaire. Il s'agit d'une bibliothèque de bas niveau avec une interface de type Matlab qui offre beaucoup de liberté au prix d'avoir à écrire plus de code. Matplotlib est particulièrement adapté à la création de graphiques de base tels que des graphiques linéaires, des graphiques à barres, des histogrammes et bien d'autres.

Histogramme

Dans Matplotlib, nous pouvons créer un histogramme en utilisant la méthode **hist**. Si nous lui transmettons des données catégoriques comme la colonne 'Kyphosis' de l'ensemble de données, il calculera automatiquement la fréquence à laquelle chaque classe se produit.

```
import matplotlib.pyplot as plt

# create figure and axis
fig, ax = plt.subplots()

# plot histogram
ax.hist(df['Kyphosis'])

# set title and labels
ax.set_title('Taux de maladies')
ax.set_xlabel('Classes')
ax.set_ylabel('Fréquence')
```

Graphique à barres

Un graphique à barres peut être créé à l'aide de la méthode **bar**. Le graphique à barres ne calcule pas automatiquement la fréquence d'une catégorie, nous allons donc utiliser la fonction pandas **value_counts** pour ce faire. Le graphique à barres est utile pour les données catégorielles qui n'ont pas beaucoup de catégories différentes (moins de 30) car sinon cela peut devenir assez compliqué.

```
# create a figure and axis
fig, ax = plt.subplots()

# count the occurrence of each class
data = df['Kyphosis'].value_counts()

# get x and y data
points = data.index
frequency = data.values

# create bar chart
ax.bar(points, frequency)
ax.set_title('Taux de maladies')
ax.set_xlabel('Classes')
ax.set_ylabel('Fréquence')
```

IV. Ensemble d'apprentissage et de test

La bibliothèque Python **scikit-learn** fournit une implémentation de la procédure de fractionnement *train-test* via la fonction **train_test_split()**. La fonction prend un ensemble de données chargé en entrée et renvoie l'ensemble de données divisé en deux sous-ensembles.

Idéalement, vous pouvez diviser votre ensemble de données d'origine en colonnes d'entrée (X) et de sortie (y), puis appeler la fonction en passant les deux tableaux et les diviser de manière appropriée en sous-ensembles d'apprentissage et de test.

```
...
# split into train test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, ...)
```

La taille de la division peut être spécifiée via l'argument *"test_size"* qui prend un nombre de lignes (entier) ou un pourcentage (float) de la taille de l'ensemble de données entre 0 et 1. Ce dernier est le plus courant, avec des valeurs utilisées telles que 0,33 où 33% de l'ensemble de données seront alloués à l'ensemble de test et 67% seront alloués à l'ensemble d'apprentissage.

```
...
# split into train test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33)
```

Il est maintenant temps de diviser nos données en un ensemble d'entraînement et un ensemble de test ! Nous allons adopter la répartition suivante :

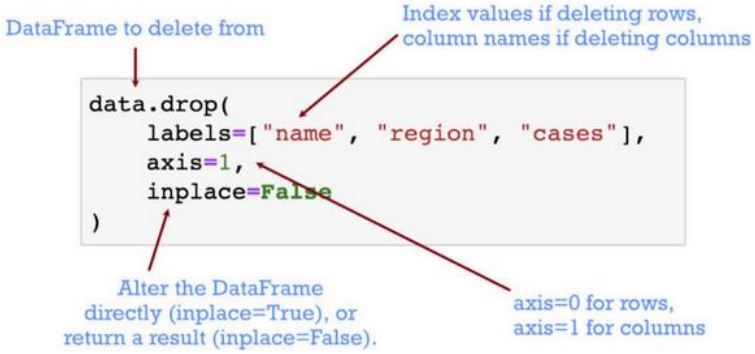
- Ensemble d'apprentissage = 70%
- Ensemble de test = 30%

Tout d'abord, il faut préciser les colonnes qui représentent les attributs et celle qui représente la classe.

```
from sklearn.model_selection import train_test_split

#Sélectionner juste les attributs en supprimant la classe
X = df.drop('Kyphosis',axis=1)

#Sélectionner la classe
y = df['Kyphosis']

#Diviser en 2 sous ensembles
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30)
```

Remarque :

Supprimer des colonnes et des lignes de votre DataFrame n'est pas toujours aussi intuitif qu'il pourrait l'être. Tout tourne autour de la commande "**DataFrame drop**". La fonction de suppression permet de supprimer des lignes et des colonnes de votre DataFrame, et une fois que vous l'aurez utilisé plusieurs fois, vous n'aurez aucun problème.



Construction d'un arbre de décision

1. **Étape 1 :** Importez le modèle que vous souhaitez utiliser

   → from sklearn.tree import DecisionTreeClassifier

2. **Étape 2 :** Créez une instance du modèle

   → clf = DecisionTreeClassifier()

3. **Étape 3 :** Former le modèle sur les données

   → clf.fit(X_train, Y_train)

4. **Étape 4 :** Prédire les étiquettes des données (test) invisibles

   → clf.predict(X_test)

```
from sklearn.tree import DecisionTreeClassifier

dtree = DecisionTreeClassifier()

dtree.fit(X_train,y_train)
```

Remarque :

DecisionTreeClassifier est une classe capable d'effectuer une classification multi-classe sur un ensemble de données.

```
class sklearn.tree.DecisionTreeClassifier(*, criterion='gini', splitter='best', max_depth=None, min_samples_split=2,
min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features=None, random_state=None, max_leaf_nodes=None,
min_impurity_decrease=0.0, class_weight=None, ccp_alpha=0.0)
```

Pramètres	Description
Criterion : **{"gini", "entropy"}, default="gini"**	La fonction pour mesurer la qualité d'un split
Splitter : **{"best", "random"}, default="best"**	La stratégie utilisée pour choisir la division à chaque nœud.
max_depth : **int, default=None**	La profondeur maximale de l'arbre.
min_samples_split : **int or float, default=2**	Le nombre minimum d'échantillons requis pour diviser un nœud interne
max_features : **int, float or {"auto", "sqrt", "log2"}, default=None**	Le nombre d'attribut à prendre en compte lors de la recherche du meilleur partage

Prédiction & Evaluation

Maintenant que notre classificateur a été entraîné, faisons des prédictions sur les données de test. Pour faire des prédictions, la méthode **predict()** de la classe DecisionTreeClassifier est utilisée.

→ predictions = dtree.predict(X_test)

Nous allons maintenant voir à quel point notre algorithme est précis. Pour les tâches de classification, certaines métriques couramment utilisées sont la matrice de confusion, la précision, le rappel et le score F1. La bibliothèque Scikit-Learn contient les méthodes **classification_report()** et **confusion_matrix()** qui peuvent être utilisées pour calculer ces métriques.

```
#Importer les bibliothèques nécessaires
from sklearn.metrics import classification_report,confusion_matrix

#Prédire les étiquettes des données
predictions = dtree.predict(X_test)

#Affichier la matrice de confusion
print(confusion_matrix(y_test,predictions))

#Afficher un les résultats obtenus
print(classification_report(y_test,predictions))
```

Visualisation de l'arbre de décision

Afficher la représentation textuelle :

L'exportation de l'arbre de décision vers la représentation textuelle peut être utile lorsque vous travaillez sur des applications sans interface utilisateur ou lorsque vous voulez enregistrer des informations sur le modèle dans un fichier texte. Pour cela, la bibliothèque Scikit-Learn dispose la méthode **export_text().**

```
#Importer la bibliothèque nécessaire
from sklearn import tree

text_representation = tree.export_text(dtree)
print(text_representation)
```

Visualisation de l'arbre:

La méthode **plot_tree** a été ajoutée à sklearn dans la version 0.21. Il nécessite l'installation de matplotlib. Il nous permet de produire facilement la figure de l'arbre (sans exportation intermédiaire vers graphviz).

```
#Importer la bibliothèque nécessaire
from sklearn import tree

# liste des attributs
fn=["Age","Number","Start"]

#Liste des classe
cn=["absent","present"]

#Agrandir la taille de l'arbre
fig = plt.figure(figsize=(25,20))

#Visualiser l'arbre
tree.plot_tree(dtree,
                    feature_names=fn,
                    class_names=cn,
                    filled=True)
```