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 « iris_Excel.xlsx ». Ce dernier contient un échantillon d'espèces différentes de fleur d'iris. Les attributs sont : la longueur des sépales, la largeur et la longueur et la largeur des pétales. Les classes sont : Setosa, Versicolor et Virginica

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 « iris.xlsx »
  → `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_excel(' iris_Excel.xlsx')

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

III. 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 = 75%
- Ensemble de test = 25%

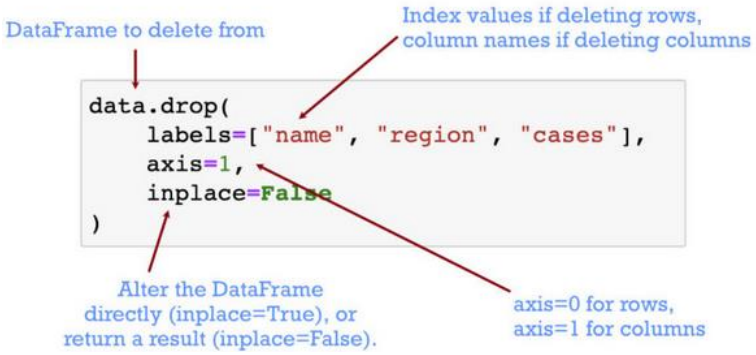
Tout d'abord, il faut préciser les colonnes qui représentent les attributs et celle qui représente la classe.

```python
from sklearn.model_selection import train_test_split

#Sélectionner juste les attributs en supprimant la classe
X = df.drop('variety',axis=1)
#Sélectionner les classes
y = df['variety']
#Diviser le dataset en 2 sous-ensemble
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25)
```

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

   → dtree = DecisionTreeClassifier()

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

   → dtree.fit(X_train, Y_train)

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

   → dtree.predict(X_test)

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).

```
#Agrandire la taille de l'arbre
fig=plt.figure(figsize=(15,10))

#Visualiser l'arbre
tree.plot_tree(dtree,
               feature_names=["sepal.length","sepal.width","petal.length","petal.width"],
               class_names=["Setosa","Versicolor","Virginica"],
               filled=True)
```

Exercice 1 :

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.

1. Importez les bibliothèques habituelles : pandas, sklearn.
2. Utilisez pandas pour lire *kyphosis.csv* en tant que **dataframe**.
3. Exécuter les méthodes **info(), head()** et **describe()** pour découvrir votre ensemble de données.
4. Utilisez sklearn pour diviser vos données en un ensemble d'entraînement et un ensemble de test comme nous l'avons déjà fait.
5. Créez une instance de **DecisionTreeClassifier()** appelée dtree
6. Entrainez votre modèle en utilisant la méthode **fit()**.
7. Créez des prédictions à partir de l'ensemble de test et créez un rapport de classification et une matrice de confusion.
8. Visualiser l'arbre de décision

Exercice 2 :

Pour ce projet, nous explorerons les données accessibles au public de LendingClub.com. Lending Club met en relation des personnes qui ont besoin d'argent (emprunteurs) avec des personnes qui ont de l'argent (investisseurs). Espérons qu'en tant qu'investisseur, vous voudriez investir dans des personnes qui ont montré un profil ayant une forte probabilité de vous rembourser. Nous allons essayer de créer un modèle qui aidera à prédire cela.

Nous utiliserons les données sur les prêts de 2007 à 2010 et essaierons de classer et de prédire si l'emprunteur a remboursé l'intégralité de son prêt ou non. Ces données sont disponibles dans le fichier csv déjà fourni.

Voici ce que représentent les colonnes :

- ***credit.policy*** : 1 si le client répond aux critères de souscription de crédit de LendingClub.com, et 0 sinon.
- ***purpose*** : l'objet du prêt (prend les valeurs "credit_card", "debt_consolidation", "educational", "major_purchase", "small_business" et "all_other").
- ***int.rate*** : Le taux d'intérêt du prêt, en tant que proportion (un taux de 11 % serait stocké sous la forme 0,11). Les emprunteurs jugés par LendingClub.com comme étant plus risqués se voient attribuer des taux d'intérêt plus élevés.
- ***mensualité*** : Les mensualités dues par l'emprunteur si le prêt est financé.
- ***log.annual.inc*** : Le logarithme naturel du revenu annuel autodéclaré de l'emprunteur.
- ***dti*** : Le ratio dette/revenu de l'emprunteur (montant de la dette divisé par le revenu annuel).
- ***fico*** : Le pointage de crédit FICO de l'emprunteur.
- ***days.with.cr.line*** : Le nombre de jours pendant lesquels l'emprunteur a eu une ligne de crédit.
- ***revol.bal*** : Le solde renouvelable de l'emprunteur (montant impayé à la fin du cycle de facturation de la carte de crédit).
- ***revol.util*** : taux d'utilisation de la ligne renouvelable de l'emprunteur (montant de la ligne de crédit utilisé par rapport au crédit total disponible).
- ***inq.last.6mths*** : Le nombre de demandes de renseignements de l'emprunteur par les créanciers au cours des 6 derniers mois.
- ***delinq.2yrs*** : le nombre de fois où l'emprunteur a été en retard de plus de 30 jours sur un paiement au cours des 2 dernières années.
- ***pub.rec*** : le nombre d'enregistrements publics dérogatoires de l'emprunteur (dossiers de faillite, privilèges fiscaux ou jugements).
- ***not.fully.paid*** : 1 si l'emprunteur a remboursé l'intégralité de son prêt, et 0 sinon.

Travail à faire

1. Importez les bibliothèques habituelles : pandas, sklearn.
2. Utilisez pandas pour lire loan_data.csv en tant que **dataframe** appelé loans.
3. Découvrez les méthodes **info(), head()** et **describe()** sur les prêts.
4. Notez que la colonne ¨**purpose**¨ est qualitative. Cela signifie que nous devons les transformer à l'aide de variables ***dummy*** en utilisant **pd.get_dummies** afin que sklearn puisse les comprendre. Afin de traiter ces colonnes, créez une liste de 1 élément contenant la chaîne 'purpose'. Appelez cette liste cat_feats.

   ```
   cat_feats = ['purpose']
   ```

   Utilisez maintenant **pd.get_dummies()** pour créer un dataframe fixe plus grand qui a de nouvelles colonnes avec des variables dummy. Définissez cet dataframe comme final_data.

```
final_data = pd.get_dummies(loans,columns=cat_feats,drop_first=True)

#visualiser le résultat
final_data.info()
final_data.head()
final_data.columns
```

5. Utilisez sklearn pour diviser vos données en un ensemble d'entraînement et un ensemble de test comme nous l'avons déjà fait.
6. Créez une instance de **DecisionTreeClassifier()** appelée dtree
7. Entrainez votre modèle en utilisant la méthode **fit()**.
8. Créez des prédictions à partir de l'ensemble de test et créez un rapport de classification et une matrice de confusion.