Apprentissage supervisé avec arbre de décision

I. Rappel

Les arbres de décision sont des méthodes d'apprentissage utilisées pour des problèmes de classification et de régression. L'objectif est de créer un modèle qui prédit les valeurs de la variable cible (target), en se basant sur un ensemble de séquences de règles de décision déduites à partir des données d'apprentissage. Ainsi, l'arbre approxime la cible par une succession de règles if-then-else. Plus l'arbre généré est complexe, mieux le modèle « explique » les données d'apprentissage mais plus le risque de sur-apprentissage (over-fitting) est élevé. Les arbres de décision ont plusieurs avantages qui les rendent intéressants dans des contextes où il est utile de comprendre la séquence de décisions prise par le modèle :

- ♣ Ils sont simples à comprendre et à visualiser.
- ♣ Ils nécessitent peu de préparation des données (normalisation, etc.).
- ♣ Ils sont capables d'utiliser des données catégorielles et numériques.
- ♣ Ils sont capables de traiter des problèmes multi-classes.

Ces modèles présentent néanmoins deux désavantages majeurs :

♣Sur-apprentissage : parfois les arbres générés sont trop complexes et généralisent mal. Choisir des bonnes valeurs pour les paramètres profondeur maximale (max_depth) et nombre minimal d'exemples par feuille (min_samples_leaf) permet d'éviter ce problème.

♣ Les arbres générés peuvent êtres non équilibrés. Il est recommandé d'ajuster la base de données avant la construction, pour éviter qu'une classe domine largement les autres (en termes de nombre d'exemples d'apprentissage).

II. Apprentissage d'arbres de classification en python :

Dans la bibliothèque Scikit-learn, la classe sklearn.tree.DecisionTreeClassifier permet de réaliser une classification multi-classes à l'aide d'un arbre de décision. Pour cela, il est nécessaire d'importer le module tree pour construire l'objet arbre.

```
from sklearn import tree
abr = tree.DecisionTreeClassifier(max_depth=3 , min_samples_leaf =2)
```

- **max_depth :** désigne la profondeur (nombre de niveaux) maximale de l'arbre.
- **min_samples_ leaf (*par défaut=1*) :** Définit le nombre minimum d'échantillons qu'une feuille (nœud terminal) doit contenir après une division.

Pour un jeu de données tel que X (les données) et y (les labels), l'arbre se construit (est entraîné) à l'aide de la méthode « .**fit**(X, y) » :

```
abr = abr.fit(X, y)
```

La prédiction sur de nouveaux échantillons se fait de façon habituelle avec « .**predict**(z) » :

```
abr.predict([[2., 2.]])
```

On peut aussi prédire la probabilité de chaque classe pour un échantillon (qui est calculée comme la fraction de données d'apprentissage dans chaque feuille) :

```
abr.predict_proba([[2., 2.]])
```

III. Application

Dans ce TP, nous allons utiliser le jeu de données qui contient 76 attributs, mais toutes les expériences publiées se réfèrent à l'utilisation d'un sous-ensemble de 14 d'entre eux. Le champ "target" fait référence à la présence d'une maladie cardiaque chez le patient ou non. Il s'agit d'un nombre entier 0 = pas/moins de risque de crise cardiaque et 1 = plus de risque de crise cardiaque.

Parmi les attributs :

- Age
- Sex
- chest pain type : type de douleur thoracique (4 valeurs)
- Pression artérielle au repos
- Cholestorale sérique en mg/dl
- Glycémie> 120 mg/dl
- Fréquence cardiaque maximale atteinte
- Angine induite par l'exercice
- Etc.

On peut télécharger ce jeu de données à partir de lien suivant:

https://www.kaggle.com/nareshbhat/health-care-data-set-on-heart-attack-possibility

1. **Importer le jeu de données.**
2. **Analyser les données :** Effectuer une visualisation des données à l'aide de **matplotlib** ou **seaborn**.
3. **Afficher la matrice de corrélation des caractéristiques des patients.**

4. **Séparer le jeu de données en deux : 70% pour l'apprentissage et 30% pour le test.**
5. **Apprendre un arbre de classification pour la prédiction des individus à risque.**
6. **Représenter l'arbre.**

```
from sklearn.tree import plot_tree
import matplotlib.pyplot as plt
fig = plt.figure(figsize=(15,7))
fn = ["age","sex","cp","trestbps","chol","fbs","restecg","thalach","exang","oldpeak","slope","ca","thal"]
cn = ['0', '1']
tree.plot_tree(abr, feature_names =fn, class_names=cn, filled=True)
plt.show()
```

7. **Utiliser le modèle pour calculer le taux d'erreurs sur l'échantillon de test.**