Université Moulay Ismail
École Nationale Supérieure D'arts Et Métiers
A.U : 2024/2025

TP3 – Classification et régression avec les arbres de décision

Exercice 1

DecisionTreeClassifier de **scikit-learn** permet de résoudre des problèmes de classification comportant plusieurs classes (par exemple : 0, 1, … K−1). Dans ce TP, nous utiliserons le jeu de données Iris, accessible directement via la bibliothèque sklearn. Ce jeu de données contient 150 observations d'une plante appelée *Iris*. Chaque observation correspond à une description morphologique d'une fleur, avec pour objectif de la classer dans l'une des trois espèces suivantes:

- ***Iris Setosa***
- ***Iris Versicolor***
- ***Iris Virginica***

Une des classes (Iris Setosa) est facilement identifiable (linéairement séparable), tandis que les deux autres sont plus difficiles à distinguer entre elles. Les variables (attributs) décrivant chaque fleur sont :

- **Longueur du sépale (sepal length)**
- **Largeur du sépale (sepal width)**
- **Longueur du pétale (petal length)**
- **Largeur du pétale (petal width)**
- **La classe cible : (*Setosa*, *Versicolor*, *Virginica*)**

Ce jeu étant très courant en apprentissage automatique, scikit-learn propose une fonction dédiée pour le charger facilement en mémoire.

1. Importer les bibliothèques nécessaires pour l'application d'un arbre de décision sur les données *Iris*.
2. Charger le jeu de données Iris à l'aide de la fonction fournie par scikit-learn.
3. Afficher un aperçu du jeu de données (dimensions, types de variables, premières lignes).
4. Combien y a-t-il d'exemples pour chaque classe ? Utiliser un tableau de fréquences pour le visualiser.
5. Diviser les données en un ensemble d'entraînement (80%) et un ensemble de test (20%).
6. Créer un modèle d'arbre de décision avec les paramètres par défaut (DecisionTreeClassifier) et l'entraîner sur les données d'entraînement.
7. Déterminer la profondeur de l'arbre et le nombre de feuilles terminales.
8. Sans visualisation, déterminer quelle variable est utilisée pour la première division (racine de l'arbre).
9. Visualiser l'arbre généré à l'aide de plot_tree().
10. Prédire les classes des données de test et afficher les résultats.
11. Évaluer les performances du modèle à l'aide de : La matrice de confusion, La précision (accuracy), Le rappel (recall) et Le F1-score.
12. Modifier les hyperparamètres (max_depth, min_samples_split, min_samples_leaf, min_impurity_decrease), visualiser les nouveaux arbres et comparer les performances.

13. Peut-on observer un phénomène d'overfitting avec les données Iris lorsqu'on utilise un arbre de décision ? Justifiez votre réponse et proposez une solution possible si c'est le cas.
14. Afficher l'importance des variables (features) pour comprendre lesquelles influencent le plus la classification.

Exercice 2 :

Objectif : Utiliser un arbre de décision pour classifier des types de vin à partir de leurs 13 caractéristiques chimiques.

1. Importer les bibliothèques nécessaires et charger le jeu de données load_wine().
2. Afficher les noms des caractéristiques, des classes, et le nombre d'instances pour chaque classe.
3. Diviser les données en un ensemble d'entraînement (80 %) et un ensemble de test (20 %).
4. Créer et entraîner un modèle d'arbre de décision avec les paramètres par défaut.
5. Afficher la précision sur l'ensemble d'entraînement et sur l'ensemble de test.
6. Identifier la variable la plus discriminante.
7. Visualiser l'arbre de décision entraîné.
8. Afficher le rapport de classification (classification_report) sur l'ensemble de test
9. Tester différentes valeurs de max_depth (de 1 à 10) : afficher les scores sur entraînement et test.
10. Tracer la courbe des scores en fonction de max_depth et analyser à partir de quelle profondeur le modèle sur-apprend.
11. En utilisant la meilleure max_depth, tester plusieurs valeurs de min_samples_leaf (1, 2, 3, 4, 5, 10, 20, 30, 40) et comparer les scores.
12. Utiliser GridSearchCV pour rechercher la meilleure combinaison de max_depth et min_samples_leaf avec validation croisée à 5 plis.
13. Afficher la meilleure combinaison (best_params_) et le score moyen (best_score_).
14. Réentraîner le modèle optimal et évaluer sa performance sur le test.
15. Afficher le rapport de classification final du modèle optimisé.
16. Selon vous, quelles configurations semblent les plus efficaces pour éviter le surapprentissage dans ce cas ?

Exercice 3 : Arbre de Régression – California Housing

1. Importez les bibliothèques nécessaires et chargez le jeu de données fetch_california_housing().
2. Affichez : Le nombre total d'instances et de variables et Les noms des variables (features).
3. Séparez les données en un ensemble d'entraînement (70 %) et un ensemble de test (30 %).
4. Créez un modèle DecisionTreeRegressor avec les paramètres par défaut.
5. Entraînez ce modèle sur les données d'entraînement.
6. Évaluez les performances du modèle :
    - Affichez le score R^2 sur l'ensemble d'entraînement.
    - Affichez le score R^2 et la RMSE sur l'ensemble de test.
7. Comparez les performances entre train et test. Y a-t-il un signe d'overfitting ? Expliquez.
8. Affichez l'arbre de régression généré. Que remarquez-vous sur sa taille ?

10. Quelle variable semble la plus utilisée dans les premières divisions de l'arbre ?
11. Entraînez plusieurs arbres avec des profondeurs (max_depth) allant de 1 à 20.
12. Pour chaque modèle, calculez le score R^2 sur le train et le test.
13. Tracez la courbe des scores R^2 en fonction de max_depth.
14. À partir de quelle profondeur le modèle commence-t-il à sur-apprendre ?
14. Utilisez GridSearchCV pour rechercher la meilleure combinaison parmi :
    - max_depth
    - min_samples_leaf
15. Affichez :
    - Les meilleurs paramètres obtenus
    - Le score moyen de validation croisée
16. Évaluez le modèle optimal sur l'ensemble de test :
    - Score R^2
    - RMSE
17. Quelle configuration de l'arbre offre le meilleur compromis entre performance et généralisation ?
18. Selon vous, quels sont les avantages et limites d'un arbre de régression sur ce jeu de données ?