TP : Base de l'apprentissage
La Classification : Decision Tree

Pierre Mahé
pierre.mahe@univ-tln.fr

3 janvier 2023

Dans ce TP, nous allons nous intéresser au fonctionnement de classification de données et plus particulièrement aux arbres de décision.

Les TPs seront notés, ils seront donc a terminer et à envoyer à l'adresse email : pierre.mahe@univ-tln.fr. Le compte rendu devra contenir : le code, figures ainsi qu'un court un résumé des explorations ainsi que des observations.

Nous allons implémenter un arbre de décision pour classifier des espèces de fleurs. L'arbre de décision sera implémenté sous forme de fonction récursive. Une ébauche de l'implémentation peut être trouvée dans les fichiers fournis. Le dataset est composé de 3 types d'iris : Iris Setosa, Iris Versicolor, Iris Virginica. Chaque donnée x est composée de 4 features : 'sepal_length', 'sepal_width', 'petal_length', 'petal_width' et d'un label y (toutes les caractéristiques sont données en millimètre).

Exercice 1

1.1) Pour nous familiariser avec les données, nous allons implémenter une fonction qui calcule le nombre d'éléments pour chaque classe et qui affiche les données avec la bibliothèque matplotlib et la fonction matplotlib.pyplot.scatter. Le nombre de dimension de x étant de 4, seule une visualisation partielle est possible, plusieurs affichages sont nécessaires.

1.2) Avant de nous lancer dans l'implementation de la fonction build_tree. Nous allons réaliser les fonctions de base utilisables par la suite. Tout d'abord, print_tree(self, tree=None) qui prend un arbre en entrée et qui affiche sous forme de texte l'arbre. Pour les feuilles, les informations sont : la classe majoritaire. Pour les noeuds de décision : l'indice de la feature utilisée, le seuil ainsi que le gain d'information du split. Pour tester votre affichage, créer à la main un arbre avec plusieurs noeuds et plusieurs feuilles.

Pour réaliser ces fonctions, il est important d'avoir une vision claire du type des variables qui vont être utilisées dans build_tree. Vous êtes invités a lire attentivement l'ensemble du code avant d'implémenter les fonctions.

1.3) Lors de la construction de l'arbre, il faut sélectionner la feature et le seuil qui permet de séparer le mieux les données. Pour calculer cela, il faut utiliser l'indice de Gini et le gain d'information. Implémentez la fonction gini_index((self, y) qui calcule l'indice de Gini pour un dataset donné. Puis la fonction information_gain(self, y_parent, y_left_child, y_rigth_child) qui retourne le gain obtenu avec la nouvelle séparation entre le dataset du noeud parent par rapport aux noeuds fils.

Exercice 2

Nous allons maintenant nous intéresser aux 2 fonctions principales qui sont build_tree et get_best_split.

2.1) La fonction build_tree prend un dataset et construit un arbre de décision. C'est une fonction récursive, A chaque noeud, la fonction va s'appeler elle-même pour la construction du sous-arbre gauche et droit (noeud gauche et droit). Il est important de prévoir les conditions d'arrêt de la récursion. La fonction doit vérifier les conditions d'arrêt qui sont : la profondeur maximum de l'arbre et le nombre d'échantillon dans le dataset. Si elle est dans une condition d'arrêt, elle doit retourner une feuille. Sinon, elle doit appeler la fonction get_best_split pour trouver la meilleure séparation possible du dataset. Puis, appeler build_tree pour les deux portions du dataset et retourner le noeud de décision.

2.2) Implémentons maintenant la fonction get_best_split. La fonction prend un dataset en paramètre et doit trouver la meilleure séparation possible pour ce dataset afin d'avoir une séparation la plus homogène possible. Pour cela, la fonction va itérer sur toutes les features et sur toutes les valeurs possibles pour trouver la séparation qui permette d'avoir un gain d'information le plus important par rapport au dataset avant la séparation. Les informations de la meilleure séparation seront stockées dans un dictionnaire et retournées par la fonction. Ces informations seront : la feature utilisée, la valeur seuil, le sous-dataset gauche et sous-dataset droit ainsi que le gain d'information.

Exercice 3

3.1) Pour évaluer les performances du modèle, il faut programmer trois fonctions : la fonction fit(self, x, y) qui construit le Decision Tree à partir d'une dataset. La fonction predict_one(self, x) qui prédit la classe d'un élément et la fonction predict_all(self, dataset) qui prédit la classe pour l'ensemble d'un dataset.

3.2) Maintenant, nous allons faire l'évaluation de votre modèle, à l'aide de la bibliothèque scikit-learn calculer l'accuracy, le recall et la precision de votre arbre de décision (les fonctions sont disponibles dans sklearn.metrics). Quels scores obtenez vous ?

3.3) (Bonus) À partir de votre Decision Tree, vous pouvez maintenant essayer d'implémenter une Random Forest avec les données de l'Iris dataset.