TP6 – Classification par arbres de décision et forêts aléatoires

Les arbres de décision sont des modèles d'apprentissage automatique qui organisent les décisions en forme de structure arborescente. Ils représentent un processus de prise de décision en utilisant des règles simples basées sur la valeur des variables. Chaque noeud de l'arbre représente une condition sur la valeur d'une variable, et les branches indiquent les résultats possibles de cette condition. En utilisant un ensemble de règles hiérarchiques, les arbres de décision permettent de résoudre des problèmes de classification ou de régression en prédisant des résultats à partir des données d'entrée, offrant ainsi une compréhension intuitive du processus de décision.

Dans ce TP, nous vous proposons tout d'abord d'implémenter et de visualiser un arbre de décision ainsi qu'une forêt aléatoire sur le même ensemble de données que pour les précédents TPs. Dans un second temps vous appliquerez votre code à une nouvelle base de données, et chercherez à obtenir la meilleure performance possible.

Rappel

L'impureté d'un ensemble $\mathcal{D}$ de données labellisées mesure le degré de mélange des étiquettes de classe à l'intérieur de cet ensemble. L'indice de Gini est l'une des mesures couramment utilisées pour quantifier cette impureté dans le contexte des arbres de décision. Si on appelle n_j la proportion d'éléments de la classe j dans l'ensemble de données, et $n = \sum_{j=1}^{K} n_j$ le nombre total d'élements de l'ensemble, alors on a :

$$\text{Gini}(\mathcal{D}) = i_n = \sum_{j=1}^{K} \frac{n_j}{n}(1 - \frac{n_j}{n}) = 1 - \sum_j \left(\frac{n_j}{n}\right)^2$$

où K désigne le nombre de classes du problème. Pour calculer la perte d'impureté associée à la séparation des données lors de la construction d'un arbre de décision, on compare généralement l'indice de Gini de l'ensemble initial avant la séparation avec la somme pondérée des indices de Gini des ensembles résultants après la séparation. La diminution de l'impureté, ou la perte d'impureté, est alors la différence entre l'impureté initiale et la somme pondérée des impuretés des ensembles divisés.

$$\Delta i_n = i_n - P_G i_G - P_D i_D$$

avec $P_G = \frac{n_G}{n}, P_D = \frac{n_D}{n}$ les proportions des ensembles D_G, D_D (gauche et droit).

Cette mesure est utilisée pour sélectionner la meilleure séparation possible lors de la construction de l'arbre de décision, visant à maximiser la pureté des noeuds fils résultants.

Exercice 0 : meilleure première séparation

Complétez la fonction `premiere_coupure`, appelée par le script `exercice_0`, permettant de déterminer la coupure qui maximise la perte d'impureté mesurée par l'indice de Gini sur les données mélanomes. En l'occurrence, plutôt de que de maximiser la perte d'impureté, on minimise l'impureté pondérée de l'ensemble séparé $P_G i_G + P_D i_D$, ce qui est plus simple à calculer.

Exercice 1 : construction d'un arbre de décision

En appliquant récursivement l'algorithme de coupure implémenté dans l'exercice 0, on peut construire un arbre de décision complet. L'algorithme CART (pour *Classification And Regression Trees*), abordé en cours, nécessite un paramètre supplémentaire qui est le nombre de données minimal pour une feuille, c'est-à-dire le nombre de de données en-deça duquel on ne cherche plus à séparer les données restantes.

Complétez le script `entrainement_arbre` qui permet d'entraîner un arbre de décision à partir des données `X_app` et des labels associés `Y_app` ainsi que du nombre de données minimal pour une feuille `taille_minimale_feuille`. Vous utiliserez pour cela la fonction `fitctree` qui construira cet arbre pour vous : à vous de déterminer comment l'appeler. Complétez ensuite le script `classification_arbre` qui calcule la prédiction `Y_pred` de l'arbre de décision `arbre` sur des données `X`. Pour cela, vous pouvez utiliser la fonction `predict` de Matlab.

Observez la forme de la frontière décision qui est composée d'une union de régions rectangulaires, en raison des séparations qui se font sur chaque variable séparément. Testez différentes valeurs pour le paramètre `taille_minimale_feuille` et observez l'évolution des résultats.

Exercice 2 : forêts aléatoires

Une technique courante d'apprentissage automatique consiste à créer un ensemble (*bag*) de prédicteurs dits faibles (c'est-à-dire, dont la performance est limitée) pour former un prédicteur performant. L'algorithme des forêts aléatoires appartient à cette famille d'algorithmes de *bagging*. Cet algorithme consiste à entraîner un ensemble d'arbres de décision complémentaires : ces arbres ne sont pas tous entraînés sur les mêmes données, ni en utilisant nécessairement les mêmes variables pour réaliser les séparations.
Les arbres sont ensuite utilisés lors de l'inférence pour la prédiction finale via un vote de majorité, comme illustré sur la Figure 1.

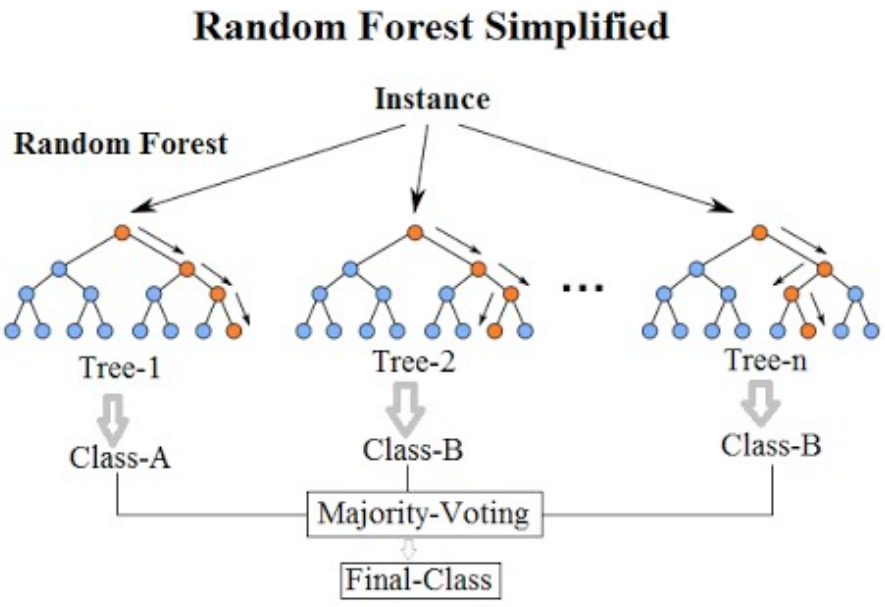

Figure 1 – Diagramme illustrant le fonctionnement des forêts aléatoires

Écrivez le script `entrainement_foret` qui permet d'entraîner un ensemble d'arbre de décision à partir des données `X_app` et des labels associés `Y_app`. Cette fonction compte deux paramètres additionnels : `nb_arbres` qui définit le nombre d'arbres composant la forêt, et `proportion_individus` qui désigne le pourcentage des données sur lequel on va entraîner chaque arbre de décision.

Conseils de programmation :

- Vous aurez besoin de la structure de données `cell` pour manipuler une forêt d'arbre comme étant un tableau de cellules. L'appel à chaque case se fait avec des accolades : `foret{i}`.
- Pour chaque arbre de décision, vous devez commencer par tirer aléatoirement un sous-ensemble des données d'apprentissage (dans une proportion indiquée par le paramètre `proportion_individus`). Ce tirage s'effectue **avec remise**, c'est-à-dire que chaque tirage se fera à partir de l'ensemble des données d'apprentissage. Vous pouvez utiliser la fonction `randperm` pour obtenir un ordre aléatoire sur les indices de cet ensemble.
- La séparation de ces arbres de décision ne doit pas prendre en compte toutes les n_{var} variables mais seulement un sous-ensemble de $\sqrt{n_{var}}$ variables, ce qui garantit une bonne variabilité des prédictions des arbres. Ceci peut être réalisé grâce au paramètre `NumVariablesToSample` de la fonction `fitctree`.

Complétez ensuite le script `classification_foret` qui calcule la prédiction de tous les arbres de décision de la forêt sur des données `X`, puis qui en calcule un vote de majorité pour obtenir les prédictions `Y_pred`. Pour cela, vous pouvez utiliser la fonction `mode` de Matlab qui permet de retrouver la valeur majoritaire d'un ensemble d'échantillons.

Testez différentes valeurs des paramètres `nb_arbres` et `proportion_individus` dans le script `exercice_2` et observez comment ces changements affectent la frontière de décision.

Exercice 3 - classification de type de forêt par forêt aléatoire !

Dans cet exercice, nous vous fournissons une nouvelle base de données permettant de classifier 7 types de couvert végétal (Épinette/Sapin, Pin tordu, Pin ponderosa, Peuplier/ Saule, Peuplier faux-tremble, Douglas de Menzies, Krummholz) en fonction d'un grand nombre de variables (54 en tout). Cette base de données est décrite plus en détail sur `https://archive.ics.uci.edu/dataset/31/covertype`.

Complétez le script `exercice_3` pour entraîner et tester une forêt aléatoire sur cette nouvelle base de données. Ce script calcule le pourcentage de bonnes classifications ainsi qu'une matrice de confusion, qui permet de visualiser les erreurs du prédicteur entre les différentes classes. A vous de chercher les paramètres qui permettent d'obtenir les meilleurs résultats sur l'ensemble de test !