Rapport TDM5 : Introduction aux sciences des données et à l'intelligence artificielle

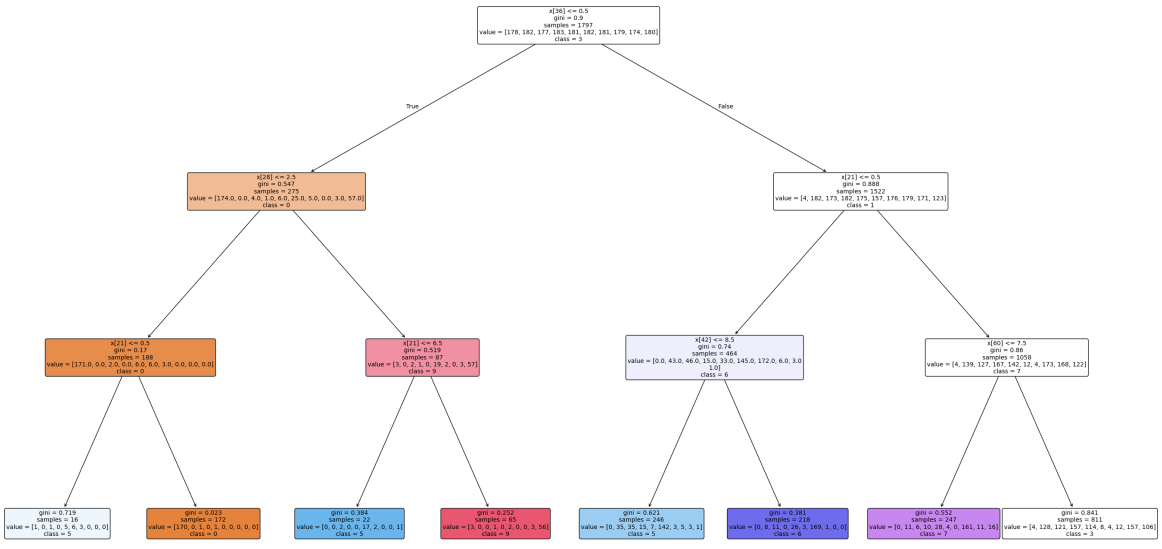
Fanani Selma et Amsaf Rim (Groupe 4)

October 11, 2024

1 Mise en œuvre des arbres de décision

Dans cette partie du TDM, nous avons étudié l'utilisation des arbres de décision pour la classification supervisée en utilisant le jeu de données digits, et nous avons exploré l'impact des hyper-paramètres sur leurs performances. L'objectif était de comprendre comment optimiser ces hyper-paramètres pour améliorer la qualité de la classification. Nous avons commencé par évaluer les performances d'un arbre de décision avec les paramètres par défaut, via une validation croisée à 5 plis. Cela nous a permis d'obtenir une première approximation du taux de réussite sans ajustement particulier. Le score moyen obtenu était de 0.7913, ce qui indique que l'arbre de décision dans sa configuration par défaut classifie correctement environ 79 % des chiffres du jeu de données.

Cependant, l'arbre obtenu était très complexe et difficile à visualiser dans son intégralité. Pour mieux comprendre sa structure et analyser les nœuds et les feuilles de manière plus aisée, nous avons décidé de réentraîner un nouvel arbre avec une profondeur maximale fixée à 3. Cette démarche nous a permis d'obtenir un arbre plus lisible, facilitant ainsi l'interprétation des décisions prises à chaque nœud. Comme vous pouvez le constater dans l'arbre ci-dessous :



Dans cet arbre, les décisions sont prises en fonction de caractéristiques discriminantes telles que X[28], X[21], et X[36]. Chaque nœud interne divise les données en sous-ensembles de plus en plus homogènes, avec l'objectif de réduire l'impureté à chaque division. Par exemple, le nœud racine utilise

la caractéristique X[28] ≤ 2.5 pour séparer les données en deux groupes : les échantillons à gauche sont ceux pour lesquels cette condition est vraie, tandis que ceux à droite ne la respectent pas. À chaque étape, l'arbre sélectionne les caractéristiques qui minimisent l'impureté de Gini pour créer des sous-groupes où les classes sont de plus en plus distinctes.

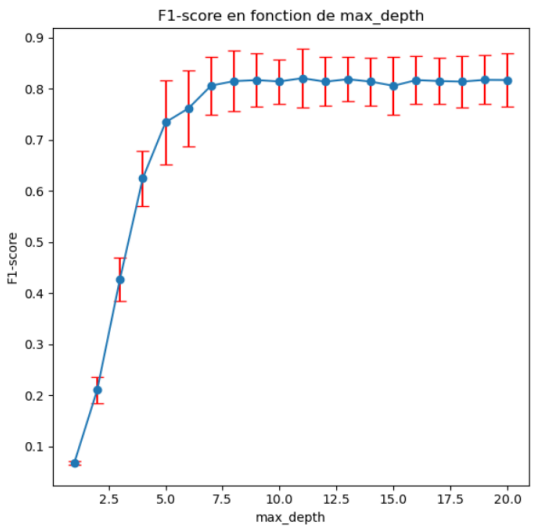
Certaines feuilles atteignent un niveau d'impureté très faible (indice de Gini proche de 0), indiquant une séparation claire des classes. Par exemple, la feuille avec un Gini de 0.023 reflète une classification presque parfaite pour la classe 0. En revanche, d'autres nœuds, comme celui avec un Gini de 0.88, contiennent un mélange plus important de classes, ce qui suggère que la division effectuée à cet endroit n'a pas permis de bien distinguer les classes. Cela peut se produire dans des zones du jeu de données où les classes sont plus difficiles à séparer.

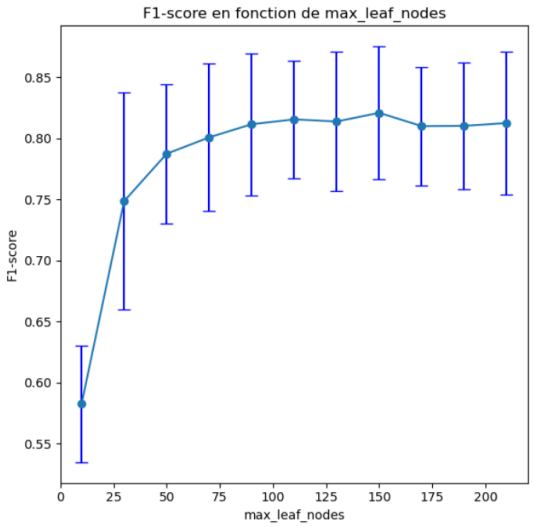
2 Étude de l'impact des hyperparamètres `max_depth` et `max_leaf_nodes` sur l'efficacité d'un modèle

Nous avons d'abord procédé à une estimation du F1-score d'un arbre de décision en utilisant les paramètres par défaut. À l'aide d'une validation croisée à 5 plis, le F1-score moyen obtenu était de 0.785, indiquant des performances satisfaisantes sans ajustement des hyper-paramètres. Après cette première évaluation, nous avons ensuite exploré l'optimisation des hyper-paramètres pour améliorer ces résultats.

Nous avons commencé par évaluer l'impact du critère de choix des tests, criterion, qui peut prendre les valeurs **gini** ou **entropy**, en utilisant cette fois une validation croisée à 10 plis. Les résultats montrent que pour gini, le taux de réussite moyen est de 0.8230, avec un F1-score de 0.8219. Pour entropy, le taux de réussite est légèrement supérieur, à 0.8235, mais le F1-score est un peu plus faible, à 0.8203. Bien que les différences soient minimes, on observe que le critère gini obtient un F1-score légèrement supérieur, suggérant un meilleur équilibre entre précision et rappel. Ainsi, bien que l'impact du choix entre gini et entropy soit faible sur ce jeu de données, gini semble offrir des performances globales légèrement meilleures.

Nous allons ensuite examiner l'impact de deux autres hyper-paramètres : **max_depth**, que nous ferons varier entre 1 et 20, et **max_leaf_nodes**, que nous ferons varier entre 10 et 210 par paliers de 20. Ces paramètres seront étudiés de manière indépendante, en utilisant entropy comme critère de sélection.





En analysant l'évolution du F1-score en fonction de max_depth, nous constatons que le score aug-

mente rapidement dans les premières étapes, avant de se stabiliser autour d'une profondeur de 7.5. Nous avons sélectionné une profondeur de 13 pour ce paramètre, car cette valeur permet d'obtenir un F1-score élevé tout en maintenant un écart type relativement faible.
De la même manière, en étudiant l'évolution du F1-score en fonction du paramètre max_leaf_nodes, nous observons que le score progresse rapidement avant de se stabiliser autour de 75 feuilles. Nous avons retenu la valeur 112, car elle permet de maintenir un F1-score élevé et un écart type modéré. Cette valeur évite de créer un modèle trop complexe, c'est-à-dire un modèle avec trop de feuilles qui risquerait de s'adapter excessivement aux données d'entraînement (phénomène de surapprentissage), tout en assurant des performances optimales.

Avec ces deux hyper-paramètres optimisés, ainsi que le critère criterion='entropy', nous avons validé notre configuration en utilisant la validation croisée. Le F1-score moyen obtenu avec ces réglages était de 0.814. Ce résultat représente une amélioration notable par rapport au F1-score initial de 0.785 obtenu avec les paramètres par défaut. Cela confirme que l'optimisation des hyper-paramètres permet d'améliorer la performance globale du modèle.

3 Recherche des hyperparamètres optimaux à l'aide de la méthode GridSearch

Jusqu'ici, nous avons pu améliorer notre modèle en testant quelques valeurs d'hyperparamètres, sans explorer toutes les combinaisons possibles entre les valeurs de `max_depth` et `max_leaf_nodes`. Il existe dans la bibliothèque `sklearn` une méthode appelée `GridSearchCV`, qui permet de tester systématiquement toutes les combinaisons possibles entre des intervalles de valeurs de nos trois hyperparamètres : `criterion`, `max_depth`, et `max_leaf_nodes`, tout en réalisant également une validation croisée.

Dans un premier temps, nous allons appliquer la méthode `GridSearchCV` en utilisant les mêmes intervalles que précédemment pour les deux hyperparamètres `max_depth` et `max_leaf_nodes`. Cette approche permettra d'identifier les valeurs optimales des hyperparamètres grâce à l'attribut `best_params_`, qui retourne la meilleure combinaison ayant donné les meilleurs résultats. De plus, la méthode fournira le score moyen le plus élevé obtenu lors de la validation croisée à l'aide de l'attribut `best_score_`.

Grâce à cette nouvelle approche, nous avons obtenu les valeurs d'hyperparamètres suivants :
Meilleurs paramètres : {'criterion': 'entropy', 'max_depth': 10, 'max_leaf_nodes': 170}
Meilleur F1-score moyen : 0.8147.
Il est intéressant de noter que les valeurs de `max_depth` et `max_leaf_nodes` sont proches de celles choisies précédemment en observant uniquement les graphiques. De plus, bien que cette approche par `GridSearchCV` ait permis d'améliorer légèrement le F1-score moyen, celui-ci n'est pas significativement plus élevé que celui obtenu avec les valeurs définies manuellement. Cela montre que, dans ce cas, une optimisation complète des hyperparamètres n'apporte pas une amélioration drastique des performances du modèle, suggérant une bonne estimation initiale des paramètres.

Lorsque nous utilisons la méthode de recherche d'hyperparamètres avec validation croisée, il est important de noter qu'un très grand nombre de modèles (dans ce cas, des arbres de décision) sont créés afin d'évaluer leur efficacité. C'est pourquoi nous avons choisi de tester uniquement 3 hyperparamètres, car il serait difficile de tester toutes les combinaisons possibles entre tous les hyperparamètres existants. L'approche idéale consiste à tester quelques hyperparamètres et à identifier ceux qui ont le plus d'impact sur l'efficacité du modèle, avant de réaliser une `GridSearchCV` sur ces hyperparamètres clés.
Par exemple, pour notre premier `GridSearchCV`, nous avons testé les hyperparamètres suivants :

```
param_grid = {
    'criterion': ['entropy', 'gini'],  % Critère de sélection
    'max_depth': range(1, 21),  % Profondeurs maximales de 1 à 20
    'max_leaf_nodes': range(10, 221, 20)  % Feuilles maximales de 10 à 210, par paliers de 20
}
```

Avec une validation croisée à 5 plis (`cv=5`), nous avons généré un total de :

$$2 \times 20 \times 11 \times 5 = 2200$$

modèles. Ce chiffre n'est pas anodin, car il souligne l'importance de l'évaluation rigoureuse de chaque combinaison d'hyperparamètres. En effet, plus le nombre de modèles testés est élevé, plus nous sommes en mesure de trouver les configurations qui optimisent la performance de notre modèle. Cela nécessite également un temps de calcul considérable, ce qui doit être pris en compte lors de la planification de nos expérimentations. Une approche systématique comme celle-ci garantit que nous ne négligeons pas des solutions potentiellement plus performantes.

Il est également important de souligner l'intérêt de l'attribut `cv_results_`, qui répertorie un ensemble d'informations pour chaque arbre appris. En sélectionnant uniquement les informations pertinentes et en triant les modèles par ordre décroissant en fonction du *F1-score* obtenu, nous avons pu afficher les cinq modèles ayant obtenu le score le plus élevé. Dans notre cas, voici les résultats obtenus pour les cinq meilleures combinaisons d'hyperparamètres :

param_criterion	**param_max_depth**	**param_max_leaf_nodes**	**mean_test_score**
entropy	15	150	0.815657
entropy	14	130	0.814605
entropy	11	190	0.814011
entropy	13	170	0.813313
entropy	18	170	0.811261

Table 1: Les 5 meilleures combinaisons d'hyperparamètres selon le F1-score moyen.

Ces résultats nous permettent de raffiner notre approche en effectuant une nouvelle recherche d'hyperparamètres *GridSearchCV* mais cette fois-ci sur un intervalle plus restreint Nous ajusterons cet intervalle en fonction des 5 meilleures combinaisons d'hyperparamètres obtenues précédemment
Nous ferons les ajustements suivants :

- **Criterion** : Nous conserverons uniquement la valeur *'entropy'* puisque cette valeur a systématiquement donné les meilleurs résultats.

- **Max Depth** : Nous réduirons l'intervalle des profondeurs d'arbre *max_depth* en le limitant aux valeurs comprises entre 11 et 18 car ces valeurs ont produit les meilleures performances.

- **Max Leaf Nodes** : Pour le nombre maximal de feuilles *max_leaf_nodes* nous limiterons l'intervalle aux valeurs comprises entre 130 et 190 qui ont également montré de bons résultats.

En affinant ces trois hyperparamètres sur ces plages plus restreintes nous espérons améliorer davantage la performance du modèle tout en réduisant le temps de calcul nécessaire pour explorer l'espace des hyperparamètres, nous obtenons donc les 5 meilleurs modèles ci-dessous :

param_criterion	**param_max_depth**	**param_max_leaf_nodes**	**mean_test_score**
entropy	13	170	0.818046
entropy	16	130	0.811880
entropy	11	170	0.810518
entropy	11	130	0.809420
entropy	13	150	0.809073

Table 2: Les 5 meilleures combinaisons d'hyperparamètres selon le F1-score moyen.

Nous pouvons donc conclure que les hyperparamètres qui produisent le meilleur arbre de décision sur ces données sont : une profondeur d'arbre de 13 et une valeur de 170 pour le paramètre *max_leaf_nodes*. Cependant il existe de nombreuses autres techniques qui pourraient encore améliorer la performance de notre modèle de prédiction, par exemple, nous pourrions explorer l'impact d'autres hyperparamètres en utilisant à nouveau une recherche par grille *GridSearchCV*

En outre la méthode *RandomizedSearchCV* pourrait être une alternative intéressante. Contrairement à *GridSearchCV* qui teste toutes les combinaisons possibles d'hyperparamètres dans une grille prédéfinie *RandomizedSearchCV* sélectionne de manière aléatoire un sous-ensemble de combinaisons d'hyperparamètres. Cela permet de réduire le temps de calcul tout en explorant efficacement un plus grand espace d'hyperparamètres. Cette méthode peut s'avérer particulièrement utile lorsque le nombre d'hyperparamètres ou les intervalles sont larges.

En explorant ces différentes méthodes d'optimisation nous pourrions découvrir des configurations d'hyperparamètres encore plus efficaces pour améliorer notre modèle de prédiction.

References

[1] GridSearchCV Documentation. *GridSearchCV User Guide*. https://scikit-learn.org/dev/modules/generated/sklearn.model_selection.GridSearchCV.html

[2] Vous retrouverez le code directement en suivant ce lien. https://github.com/SelmaFanani/TDM5