RAPPORT - CHALLENGE KAGGLE

21 décembre 2024

Stéphane EILLES-CHAN WAY et Hugo PERCOT

TABLE DES MATIÈRES

1
DESCRIPTION DU PROBLÈME ET DU DATA SET

L'objectif de ce challenge est de parvenir à prédire le `Cover_Type`, c'est-à-dire le type d'arbre qui couvre une parcelle (30m par 30m) d'une forêt. Cette prédiction s'appuie sur différentes observations réalisées sur la parcelle en question. Nous avons accès à un dataset d'entraînement composé de 15120 entrées pour construire un modèle le plus performant possible. Le `Cover_Type` est une variable multi-catégorielle qui peut prendre une valeur entre 1 et 7. La signification de cette codification peut être retrouvée en annexe.

Les observations sont de deux types, catégorielles ou quantitatives. Le premier type couvre les attributs :
- `Wilderness_Area` : valeur entière entre 1 et 4. Il s'agit de catégories d'espaces géographiques qui possèdent chacune des caractéristiques particulières.
- `Soil_Type` : valeur entière entre 1 et 40. Il s'agit de la catégorie du sol dans la parcelle.

La signification de cette codification peut être retrouvée en annexe. Notons que ces deux attributs ne sont a priori pas dans un espace métrique, par exemple, `Soil_Type = 1` n'est pas plus proche de `Soil_Type = 2` que de `Soil_Type = 40`.

Le second type comprend dix observations :
- `Elevation` : L'altitude en mètre
- `Aspect` : C'est l'azimut auquel le terrain fait face (suppose une pente non nulle)
- `Slope` : La pente en degré (moyenne sur la parcelle a priori)
- `Horizontal_Distance_To_Hydrology` : Distance horizontale du point d'eau le plus proche en mètre
- `Vertical_Distance_To_Hydrology` : Distance verticale du point d'eau le plus proche en mètre
- `Horizontal_Distance_To_Roadways` : Distance horizontale de la route la plus proche en mètre
- `Hillshade_9am` : Ombrage à 9h du matin au solstice d'été (entre 0 et 255)
- `Hillshade_noon` : Ombrage à midi au solstice d'été
- `Hillshade_3pm` : Ombrage à 3h de l'après-midi au solstice d'été
- `Horizontal_Distance_To_Fire_Point` : Distance horizontale du départ de feu sauvage le plus proche en mètre

Pour évaluer la performance finale de notre modèle, nous avons un dataset de test, qui permet de calculer la précision des prédictions.

2
ANALYSE EXPLOIRATOIRE DES DONNÉES

Dans cette section, nous présentons le travail qui a été réalisé avant d'entreprendre l'apprentissage de modèle : l'analyse exploratoire des données. L'objectif est de mieux comprendre les données et les relations entre les attributs. Cela permet alors de justifier certaines initiatives de prétraitement et de feature engineering sur ces données pour améliorer les modèles obtenus. Ainsi, cette partie présente les observations qui nous ont poussés à essayer ces techniques, et la section 3.2 en donne les performances expérimentales.

Avant toute chose, nous nous sommes assurés que les données étaient propres, c'est-à-dire sans données manquantes ou entrées redondantes, mais aussi qu'elles étaient du bon type (en l'occurrence entier pour notre dataset). Sur ce point, le dataset d'entraînement était propre.

2.1 Analyse univariée des données et outliers

Nous nous sommes intéressés à la possible présence de données aberrantes. Parmi les indicateurs statistiques qui peuvent être utilisés pour les repérer, nous avons choisi le diagramme boîte à moustaches qui fait ressortir les valeurs supérieures à $Q_3 + 1.5\Delta Q$ ou inférieures à $Q_1 - 1.5\Delta Q$, où Q_1, Q_3 et ΔQ sont respectivement le premier, le troisième quartile et l'écart inter-quartile.

Par exemple, la figure 1 montre assez facilement que l'attribut `Vertical_Distance_To_Hydrology` possède deux valeurs suspectes (les deux valeurs maximales) qui sont très éloignées de la moyenne par rapport à l'écart-type. De la même façon, plusieurs autres attributs pourraient présenter des valeurs aberrantes. Des données erronées comme celles-ci risquent de provoquer un overfitting du modèle, et donc réduire la précision sur le dataset de test. Ainsi, le prétraitement pourrait consister à retirer les lignes comprenant ces *outliers*.

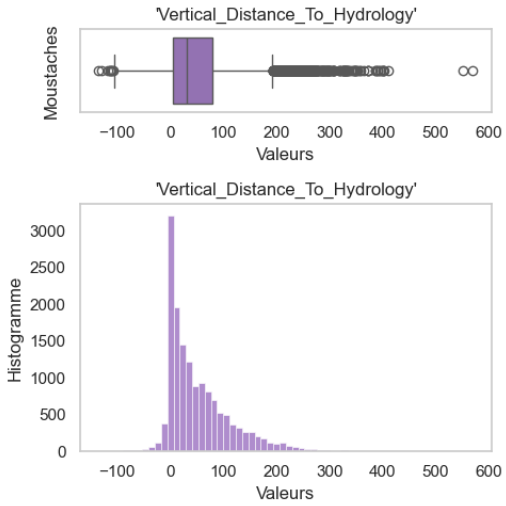
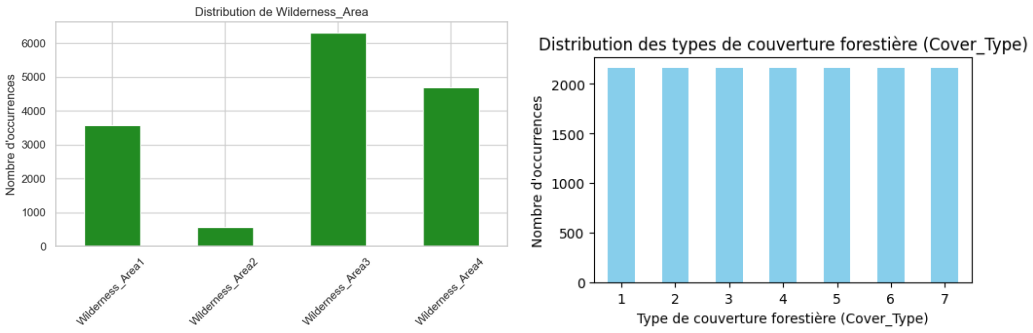


Figure 1 – Box plot de `Vertical_Distance_To_Hydrology`

Enfin, nous avons observé la distribution des classes `Cover_Type` (figure 2b). Il s'est avéré que dans le dataset d'entraînement, chacun des 7 types était présent en même quantité (2160 par type). Or, comme le confirmeront de futures observations, les proportions de `Cover_Type` dans le dataset de test ne sont pas uniformes.

De plus, selon le court texte de présentation des données sur la page du challenge Kaggle, il est dit pour l'attribut `Wilderness_Area` que les catégories 1 et 3 sont les plus communes parmi les données étudiées, devant



(a) `Wilderness_Area` (b) `Cover_Type`

Figure 2 – Distributions

la catégorie 2, puis la catégorie 4 qui serait bien plus rare. Or lorsque l'on observe la distribution du dataset d'entraînement (figure 2a), on observe que la catégorie 4 est sur-représentée et que la catégorie 2 n'est que très rare. Ces deux contradictions entre distributions réelle et celle du dataset d'entraînement laissent penser que les données qui y figurent n'ont pas été prises au hasard. On devra prendre garde aux risques de biais et d'overfitting.

2.2 Analyse bivariée des données

Dans cette section, nous étudions les relations entre les attributs. L'objectif est double :
- Comprendre les relations entre les attributs afin de pouvoir éventuellement supprimer des redondances, créer des nouveaux attributs, etc
- Identifier des règles de séparation, c'est à dire des partitions des ensembles dans lequels vivent les attributs qui permettent de discriminer une classe de `Cover_Type` par rapport à une autre ou par rapport à toutes les autres. Il faut donc étudier les relations entre `Cover_Type` et les attributs.

• Relation `Cover_Type` versus attributs qualitatifs

Pour commencer avec les attributs catégoriels, on construit la matrice de contingence entre `Cover_Type` et ces attributs. Par exemple, pour `Soil_Type`, on obtient la figure 3.

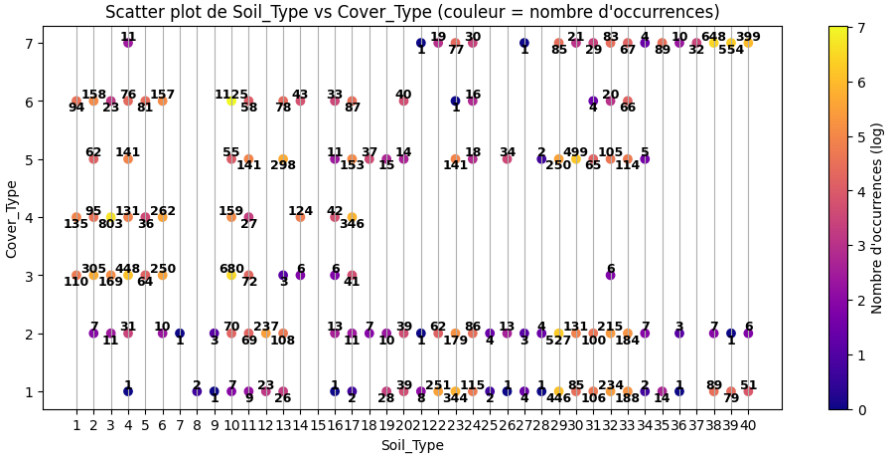


Figure 3 – Matrice de contingence entre `Cover_Type` et `Soil_Type`

On s'aperçoit que l'attribut `Soil_Type` permet de séparer presque parfaitement `Cover_Type = 7` de `Cover_Type = 3` et `Cover_Type = 4`. On s'aperçoit que le `Soil_Type8` n'est présent que pour `Cover_Type = 1`. On pourrait donc implémenter une règle de prédiction qui prédit 1 si `Soil_Type = 8`, puis entraîner un modèle sur les données `Soil_Type != 8`. Cependant, il faut se méfier de l'overfitting, car on voit qu'il n'y a que 2 apparitions de `Soil_Type8`, le comportement observé pourrait être un coup du hasard ou de la séléction des données. De la même façon, `Soil_Type7` n'est présent que pour `Cover_Type = 2` (1 apparition) et `Soil_Type37` n'est présent que pour `Cover_Type = 7` (32 apparitions).

En réalité, les `Soil_Type` ne sont pas de simples catégories indépendantes. On pourrait voir ces données sous un autre jour, notamment pour prendre en compte la proximité entre certaines catégories. Une première manière de faire cela simplement est de contracter les 40 variables binaires `Soil_Type` en une seule variable quantitative (entière) dont la valeur est le numéro du `Soil_Type`. Cela est possible car le `Soil_Type` est unique pour chaque ligne.

En fait, chaque catégorie `Soil_Type` renferme trois informations qui peuvent recouper d'autres catégories `Soil_Type` :

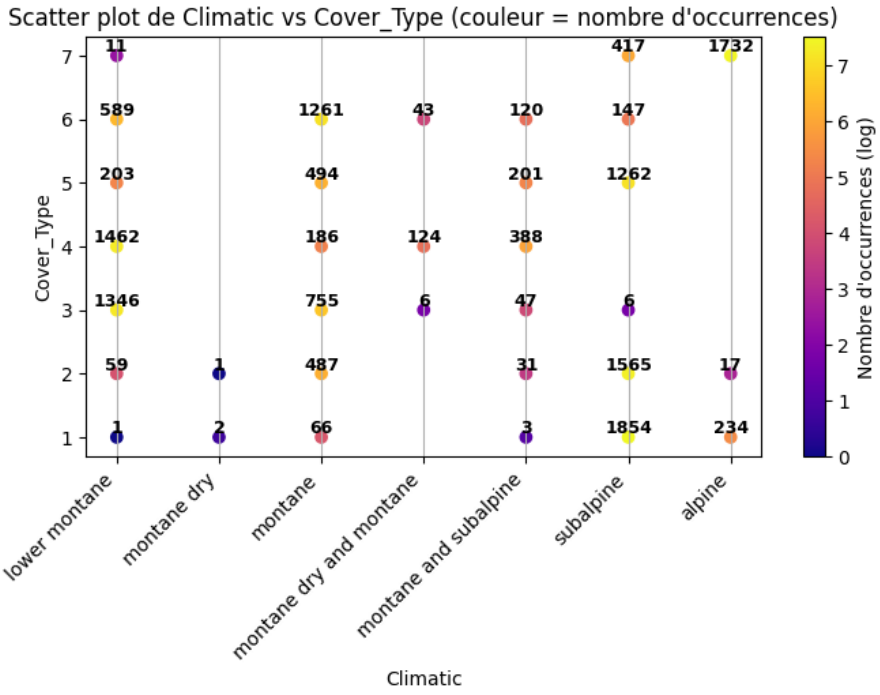


Figure 4 – Relation entre les types de couverture et les zones climatiques

— La zone climatique (catégorielle) : basse montagne, haute montagne, sec, etc
— La zone géologique (catégorielle) : schiste, glace, volcanique, etc
— Une description textuelle du sol.

On peut alors regrouper des `Soil_Type` selon leurs caractéristiques climatiques ou géologiques. Ainsi, on obtient la figure ci-contre. On s'aperçoit que certains `Cover_Type` sont très minoritaires pour certaines zones climatiques (e.g. `Cover_Type = 1` dans les 'lower montane' ou 'montane and subalpine'). Supprimer ces données minoritaires pourrait déboucher sur une meilleure précision du modèle.

Enfin, la page du projet Kaggle donne une description textuelle courte de chaque `Soil_Type`. Celle-ci peut laisser transparaître la répartition selon la zone climatique et géographique comme évoquée précédemment, mais pourrait donner des informations supplémentaires. L'utilisation de techniques de **NLP** pourrait permettre d'apprendre un modèle de représentation des `Soil_Type` pour en utiliser toute l'information contenue dans ces variables.

- Relation `Cover_Type` versus attributs quantitatifs

Ensuite, avec les attributs quantitatifs, on trace les densités selon la classe du `Cover_Type`. Par exemple, pour `Elevation`, on obtient la figure 5. Si des courbes de densité possèdent des supports disjoints, alors il sera a priori facile de séparer les `Cover_Type` associés. Dans cet exemple, il y a plusieurs sous-ensembles de classe facile à discriminer. C'est le cas de $\{4, 5, 7\}$, mais pas de $\{1, 2\}$ par exemple. `Elevation` est le seul attribut quantitatif pour lequel on a repéré de telles séparations flagrantes.

Pour toutes les séparations repérées, on s'attend à ce que notre modèle ne fasse aucune (ou très peu) de confusion, étant donné qu'on pourrait implémenter nous-mêmes la règle à la main.

- Relation entre attributs quantitatifs

Analyser les relations entre des attributs quantitatifs peut nous donner des idées de transformations à appliquer sur ces derniers.

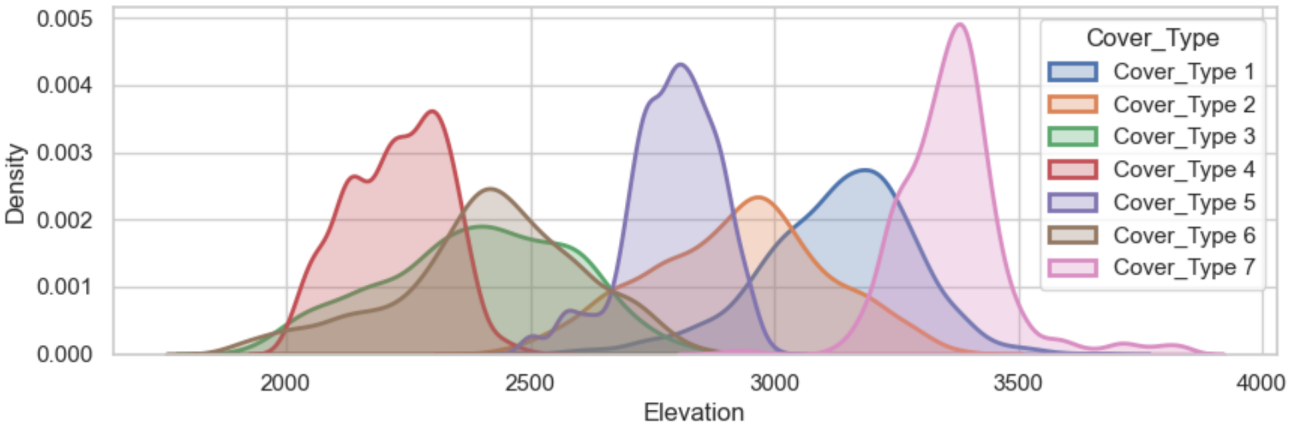


Figure 5 – Densité de `Elevation` selon la classe `Cover_Type`

Premièrement, les variables `Vertical_Distance_To_Hydrology` et `Horizontal_Distance_To_Hydrology` ont une corrélation de 0,65. On peut supposer alors que ces deux attributs apportent des informations similaires, et qu'ils peuvent surtout être combinés en une distance sur le plan. On pourrait construire une distance $||.||_\infty$ en prenant le maximum de ces deux distances, ou une distance euclidienne $||.||_2$ en prenant :

$$\texttt{Distance\_To\_Hydrology} = \sqrt{\texttt{Vertical\_Distance\_To\_Hydrology}^2 + \texttt{Horizontal\_Distance\_To\_Hydrology}^2}$$

On peut alors remplacer les deux premiers attributs par cette nouvelle distance.

D'autre part, les variables `Hillshade_9am` et `Hillshade_3pm` ont de fortes corrélations avec plusieurs variables, comme en témoigne la figure 6.

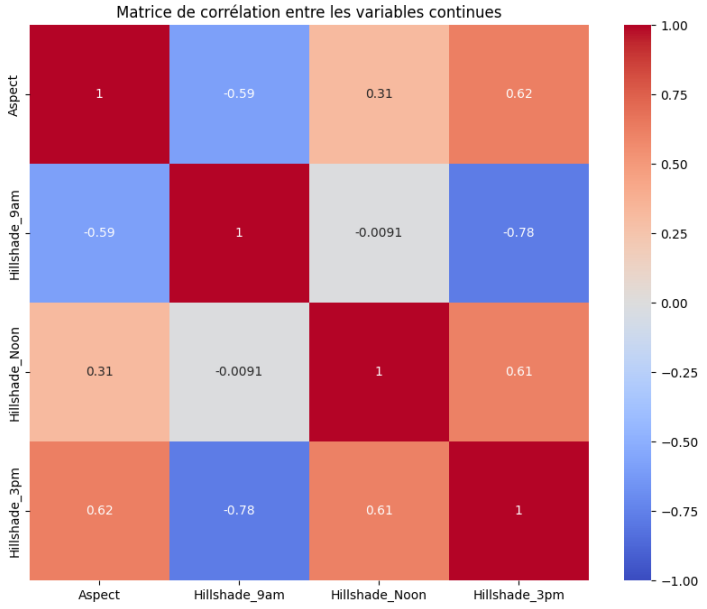


Figure 6 – Matrice de corrélation

Ainsi, on observe que la variable `Hillshade_3pm` crée en partie une redondance avec les attributs `Hillshade_9am`, `Hillshade_Noon` et `Aspect`. Cela n'est pas étonnant, car il est commun que l'ensoleillement et l'ombrage soient fortement dépendants de l'orientation d'une parcelle. On peut alors envisager de supprimer l'attribut

`Hillshade_3pm` ou `Hillshade_9am`.

Enfin, il est intéressant de se pencher sur les variables `Aspect` et `Slope`. Ces attributs étant exprimés en degrés, il est tentant de leur appliquer une fonction trigonométrique. Cette intuition est également motivée par la relation entre `Aspect` et d'autres attributs, comme le montre en particulier la figure 7.

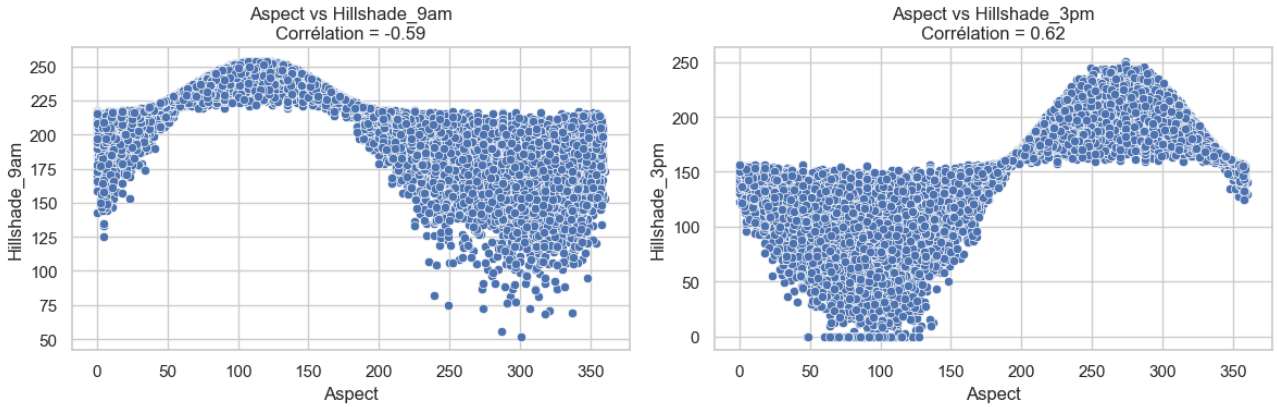


Figure 7 – Des relations sinusoidales

- Un aperçu de l'importance des attributs

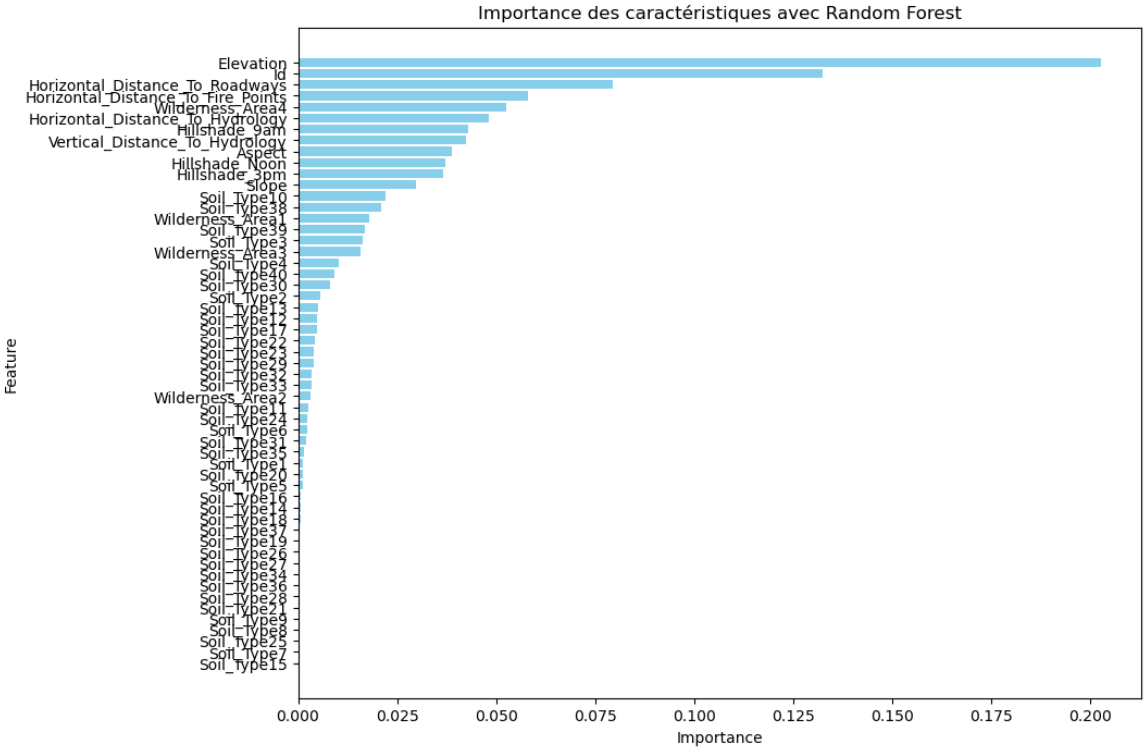


Figure 8 – L'importance de chaque attribut dans la prédiction par le modèle Random forest

Une première application d'un modèle **Random Forest** nous donne une idée de l'importance de chaque attribut dans la prédiction du `Cover_Type`. La figure 8 montre que `Elevation` est l'attribut le plus important

pour la prédiction, mais, de manière plus surprenante, on voit que l'attribut `Id` se classe deuxième en termes d'importance. Assez étonnant, car habituellement, on a souvent le réflexe de supprimer cette information jugée purement artificielle. Pourtant, dans notre cas, elle semble contenir de l'information importante pour une bonne prédiction.

3 MÉTHODOLOGIE

Dans cette partie, nous retraçons la méthodologie et le cheminement vers notre solution finale.

3.1 Choix du modèle

- **Test de modèles classiques**

La première chose que nous avons faite après l'étude exploratoire des données a été de tester les modèles classiques pour nous donner une idée des modèles les plus performants pour le problème. Les hyperparamètres à ce stade sont arbitraires.

Pour pouvoir évaluer les modèles, nous avons divisé le data set issu de *train.csv* en quatre data set : *X_train*, *y_train* les données d'entraînement, et *X_test*, *y_test* les données de validation.

Certaines méthodes de machine learning, comme les algorithmes basés sur des distances (par exemple, k-nearest neighbors, k-means, SVM), sont sensibles à l'échelle des caractéristiques. Nous avons normalisé nos données pour cette étude.

Régression logistique : Ce modèle linéaire simple va nous donner une base de performance à battre dans la suite. La précision globale (*accuracy*) obtenue pour le modèle de régression logistique est de **0.7229**.Cela donne un premier ordre d'idée des performances que l'on peut obtenir. Le rapport de classification détaillé est donné par le tableau 1.

Classe	Precision	Recall	F1-Score	Support
1	0.67	0.67	0.67	648
2	0.62	0.54	0.58	648
3	0.66	0.58	0.62	648
4	0.88	0.90	0.89	648
5	0.76	0.79	0.77	648
6	0.61	0.69	0.65	648
7	0.85	0.89	0.87	648
Accuracy	0.72 (sur 4536 échantillons)			
Macro avg	0.72	0.72	0.72	4536
Weighted avg	0.72	0.72	0.72	4536

Table 1 – Rapport de classification pour le modèle de régression logistique.

Random Forest : La précision globale obtenue pour le modèle Random Forest est de **0.8769**.Ce modèle est donc déjà plutôt performant, et est prometteur pour la suite. Le rapport de classification détaillé est donné par

le tableau 2.

Classe	Precision	Recall	F1-Score	Support
1	0.80	0.78	0.79	648
2	0.80	0.71	0.76	648
3	0.85	0.88	0.87	648
4	0.95	0.98	0.96	648
5	0.90	0.95	0.93	648
6	0.87	0.88	0.88	648
7	0.94	0.96	0.95	648
Accuracy	0.88 (sur 4536 échantillons)			
Macro avg	0.87	0.88	0.88	4536
Weighted avg	0.87	0.88	0.88	4536

Table 2 – Rapport de classification pour le modèle Random Forest.

XGBoost : La précision globale obtenue pour le modèle XGBoost est de **0.8891**, ce qui témoigne d'une excellente performance. On ne met pas le rapport de classification détaillé à partir de ce modèle et pour tous les autres par souci de longueur, mais ils ont bien sûr été réalisés et pris en compte dans notre étude. Avec une telle précision, ce modèle XGBoost se révèle particulièrement prometteur pour répondre au problème étudié.

LightGBM : La précision globale obtenue pour le modèle LightGBM est de **0.8918**, ce qui est aussi très bon et prometteur.

Support Vector Machine (SVM) : La précision globale obtenue pour le modèle SVM est de **0.3893** avec le noyau RBF, ce qui est plutôt décevant. D'autres noyaux comme le noyau Polynomial ou Sigmoïde ont été utilisés, mais cela n'a pas beaucoup amélioré la performance.

K-Nearest Neighbors (KNN) : La précision globale obtenue pour le modèle KNN est de **0.5516**, ce qui est aussi assez faible. En essayant d'autres métriques et d'autres nombres de voisins, nous avons atteint **0.6651** ce qui reste faible.

Extra Trees : La précision globale obtenue pour le modèle Extra Trees est de **0.8741**, ce qui est bon.

Gradient Boosting : La précision globale obtenue pour le modèle Gradient Boosting est de **0.8893**, ce qui est très performant. Cependant, ce modèle met plus de temps que les autres à s'entraîner : il met 1 min 30 alors que pour les autres, le temps d'entraînement est de l'ordre de quelques secondes.

CatBoost : La précision globale obtenue pour le modèle catBoost est de **0.8578**, ce qui est bon mais un peu moins que les autres modèles utilisant des arbres.

Cette étude permet de conclure que les modèles basés sur des arbres, tels que Random Forest (0.8769), Extra Trees (0.8741), et surtout les modèles de boosting comme XGBoost (0.8891), LightGBM (0.8918), et Gradient Boosting (0.8893), se sont montrés nettement supérieurs en termes de précision.

On remarque aussi dans les rapports de classification que pour tous ces modèles, les classes 1 et 2 sont celles avec le moins de précision avec environ 0.80, alors qu'elle est d'au moins 0.87 pour les autres classes. En générant une **matrice de confusion**, on remarque que les classes 1 et 2 sont souvent confondues par ces modèles.

- **Test du Deep Learning**

Nous avons ensuite testé des modèles de Deep learning. Nous avons exploré l'utilisation des Multilayer Perceptrons (MLP) pour résoudre le problème de classification. Les MLP, étant des réseaux de neurones entièrement connectés, sont particulièrement adaptés pour les données tabulaires où les relations entre les caractéristiques ne présentent pas nécessairement de structure spatiale ou temporelle. Cette approche se justifie par le fait que les données de notre problème ne contiennent pas de dépendances locales (comme les images pour les CNN) ou séquentielles (comme les séries temporelles pour les RNN/LSTM).

Ainsi, nous avons développé un réseau avec une architecture profonde et riche en neurones. L'objectif principal était de réduire l'underfitting auquel nous étions confrontés lors du Deep Learning. Nous avons conçu un réseau comportant 5 couches cachées avec un grand nombre de neurones (`1024, 512, 512, 256, 128`) et une couche de sortie correspondant au nombre de classes de classification.

Encore dans le but de réduire l'underfitting, nous avons utilisé la fonction d'activation `LeakyReLU`, qui, contrairement à la fonction `ReLU`, atténue le problème des "neurones morts" grâce à sa pente non nulle pour les valeurs négatives.

L'optimisation du modèle a été réalisée avec un taux d'apprentissage initial de 0.001, ajusté dynamiquement via un `scheduler` (`ReduceLROnPlateau`) pour réduire le taux lorsque la perte sur l'ensemble de validation stagnait.

Malgré toutes ces modifications et un long entraînement, la précision de ce modèle était de **0.713**, ce qui montre que les modèles de Deep Learning ne sont pas les plus performants pour notre problème. La nature tabulaire et relativement simple des données favorise davantage les modèles basés sur des arbres, qui sont souvent mieux adaptés aux caractéristiques structurées et catégoriques.

- **Stacking des modèles performants**

Pour améliorer les performances globales de notre solution, nous avons opté pour une approche de **stacking**, qui combine les prédictions de plusieurs modèles performants afin de tirer parti de leurs forces respectives. Nous avons sélectionné cinq modèles performants pour cette tâche : **Random Forest**, **XGBoost**, **LightGBM**, **Extra Trees** et **Gradient Boosting**. Ces modèles ont été entraînés sur les mêmes données et leurs prédictions ont ensuite été utilisées comme nouvelles features pour un méta-modèle.

Au début, nous avons choisi d'utiliser une **régression logistique** comme méta-modèle pour le stacking. Ce choix initial, simple mais efficace, a permis d'améliorer les performances par rapport aux modèles de base, démontrant la pertinence de la méthode de stacking. Cependant, afin d'explorer davantage les possibilités d'optimisation, nous avons essayé d'autres méta-modèles, notamment **LightGBM**, **Gradient Boosting**, et **XGBoost**. Après une évaluation comparative, nous avons finalement retenu **XGBoost** comme méta-modèle final, car il offrait les meilleurs résultats en termes de précision globale et de robustesse, consolidant ainsi les prédictions issues des modèles de base.

Ce modèle atteint ainsi la précision de **0.9019** sur le data set de validation, sans avoir fait de feature engineering, ni d'optimisation des hyperparamètres à ce stade.

3.2 Prétraitement et feature engineering

Une fois que nous avons trouvé notre modèle de stacking qui a une bonne performance avec les données inchangées, nous avons alors cherché à améliorer la performance avec le feature engineering que suggère notre analyse exploratoire des données. Nous avons donc repris les idées de la partie 2 pour les tester sur les cinq modèles classiques sélectionnés (Random Forest, XGBoost, LightGBM, Extra Trees et Gradient Boosting) et voir quel feature engineering est bénéfique.

- **Suppression de Hillshade 9am**

Nous avons supprimé `Hillshade_9am`, ce qui a bien amélioré la performance des modèles classiques, passant par exemple de **0.8891** à **0.8953** pour XGBoost. Cette modification a donc été gardée pour la suite.

Conformément à ce qui a été présenté dans la partie précédente, nous aurions pu essayer de supprimer `Hillshade_3pm`, mais en pratique, les résultats étaient moins bons.

- **Distance To Hydrology**

Nous avons ajouté un attribut `Distance_To_Hydrology` défini comme ceci :

$$\texttt{Distance\_To\_Hydrology} = \sqrt{\texttt{Vertical\_Distance\_To\_Hydrology}^2 + \texttt{Horizontal\_Distance\_To\_Hydrology}^2}$$

Cependant, ce changement n'a pas amélioré la performance des modèles, il n'a donc pas été retenu.

- **Modification des angles en leur sinus ou cosinus**

Nous avons changé l'attribut `Aspect` en $sin($`Aspect`$)$, puis en $cos($`Aspect`$)$, mais aucun des deux changements n'a amélioré la performance des modèles. Ils n'ont donc pas été retenus.

Nous avons fait de même avec l'attribut `Slope`, mais de même cela n'a pas augmenté la précision, donc on ne retient pas la modification non plus.

- **Soil Type en un seul attribut**

Nous avons supprimé les nombreuses variables binaires des `Soil_Type` pour en faire un seul attribut `Soil_type` qui vaut le numéro du Soil Type. Cependant, cela n'a pas eu d'impact sur les résultats, donc cette modification n'a pas été gardée.

- **Ajout de Climatic Zone et Geologic Zone**

Nous avons extrait les informations du code ELU des `Soil_Type` et pour les utiliser directement dans les modèles en ajoutant des nouvelles colonnes `climatic_zone` et `geologic_zone` aux caractéristiques d'entrée des modèles. Cela n'a pas amélioré la performance non plus, nous n'avons donc pas gardé cette modification.

- **Exploitation des descriptions des Soil Type**

Pour exploiter les informations textuelles associées aux attributs `Soil_Type`, nous avons utilisé une méthode d'encodage de texte afin de capturer les similitudes sémantiques entre les descriptions. L'objectif était de représenter chaque description sous forme d'un vecteur dense (**embedding**), permettant ainsi aux descriptions textuelles proches d'avoir des vecteurs similaires.

Pour cela, nous avons commencé par créer un dictionnaire associant chaque valeur de `Soil_Type` à sa description textuelle. Ensuite, nous avons utilisé la bibliothèque `SentenceTransformers`, un outil performant pour l'extraction d'*embeddings* textuels, afin de transformer ces descriptions en vecteurs. Ces vecteurs, représentant les caractéristiques sémantiques des descriptions de `Soil_Type`, ont ensuite été intégrés au jeu de données comme nouvelles caractéristiques.

La précision de certains modèles a été légèrement améliorée mais celle d'autres a été diminuée. De plus, les vecteurs étaient de taille 377 ce qui a augmenté considérablement la taille des entrées dans les modèles et qui, par conséquent, les a rendus beaucoup plus lents. Nous avons alors fait le choix de ne pas garder cette modification car, en vue de l'optimisation des hyper-paramètres, travailler avec des modèles trop lents n'allait pas être agréable.

- Attribution automatique pour certains Soil Type

Nous avons vu que certains `Soil_Type` ne sont présents que pour un `Cover_type` spécifique. On décide alors d'attribuer automatiquement le `Cover_Type` 7 aux points avec un `Soil_type` égal à 37. Puis nous passons le reste des points qui n'ont pas été attribués dans le modèle de stacking.

Nous remarquons que les performances de ce procédé et du modèle de stacking seul sont exactement les mêmes, on en déduit que notre modèle attribuait déjà de lui-même les `Soil_type` 37 au `Cover_Type` 7. On ne retient donc pas ce changement.

3.3 Optimisation des hyperparamètres

Maintenant que nous avons le type de modèle et le feature engineering, nous avons pu optimiser les hyperparamètres du modèle.

Nous avons d'abord optimisé les hyperparamètres de chaque modèle classique en utilisant la bibliothèque **Optuna**, qui repose sur des méthodes bayésiennes pour chercher les combinaisons optimales d'hyperparamètres. Cette approche permet d'explorer intelligemment l'espace des hyperparamètres en se concentrant sur les régions les plus prometteuses, ce qui améliore l'efficacité et la qualité de l'optimisation par rapport à des recherches classiques comme le *grid search*.

Un avantage supplémentaire d'**Optuna** est qu'il fournit une analyse de l'importance des hyperparamètres, ce qui permet de mieux comprendre quels paramètres influencent le plus les performances des modèles. Cette capacité nous a aidés à affiner davantage nos choix et à prioriser les hyperparamètres critiques pour chaque modèle.

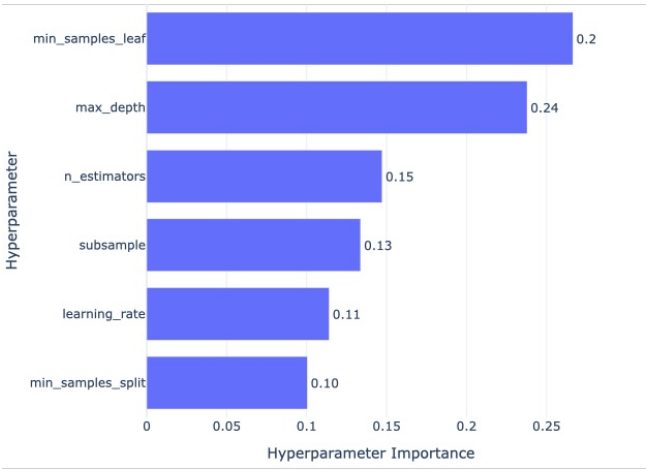


Figure 9 – Exemple d'importances d'hyperparamètres données par optuna

Concernant l'optimisation des hyperparamètres du méta-modèle, il était plus compliqué d'utiliser une bibliothèque comme **Optuna** ou une autre méthode automatisée en raison de la complexité supplémentaire qu'apporte le stacking et du temps de calcul.

Nous avons donc procédé à une recherche des hyperparamètres de manière plus manuelle, en testant différentes configurations et en observant celles qui donnaient les meilleurs résultats.

Cela a été guidé par l'idée qu'un méta-modèle doit rester relativement simple pour éviter le sur-apprentissage, car il repose sur les prédictions des modèles de base déjà performants. Cette approche empirique nous a permis d'obtenir une configuration équilibrée et efficace pour le méta-modèle.

Pour l'apprentissage du méta-modèle, nous avons choisi d'utiliser une validation croisée avec un nombre de splits élevé, à savoir **20**. Cette décision a été motivée par l'objectif de réduire le risque d'overfitting.

Tous les hyperparamètres choisis sont ainsi présentés dans la solution finale dans la partie suivante.

4
SOLUTION FINALE ET SA PERFORMANCE

4.1 Description du modèle

Notre solution finale repose donc sur l'approche de stacking, qui combine plusieurs modèles de base et un méta-modèle.

- **Modèles de base**

Pour les modèles de base, nous avons donc sélectionné les cinq modèles et les hyperparamètres suivants :

Random Forest
— **n_estimators** : 300 (nombre d'arbres dans la forêt)
— **max_depth** : None (profondeur maximale non limitée, permettant une division jusqu'à ce que toutes les feuilles soient pures ou contiennent moins de 2 échantillons)
— **random_state** : 42 (pour assurer la reproductibilité)
— **max_features** : sqrt (le nombre maximal de caractéristiques considérées lors d'une division est la racine carrée du total des caractéristiques)
— **min_samples_split** : 2 (nombre minimal d'échantillons requis pour effectuer une division interne)
— **min_samples_leaf** : 1 (nombre minimal d'échantillons requis pour former une feuille)
— **bootstrap** : True (échantillonnage avec remise)

XGBoost
— **n_estimators** : 200 (nombre maximal d'arbres construits)
— **max_depth** : 9 (profondeur maximale des arbres, permettant de capturer des relations complexes)
— **learning_rate** : 0.112 (taux d'apprentissage utilisé pour ajuster la contribution de chaque arbre)
— **colsample_bytree** : 0.9936 (fraction des caractéristiques à utiliser pour chaque arbre)
— **gamma** : 0.0129 (réduction minimale de la perte requise pour effectuer une division)
— **subsample** : 0.7079 (fraction des échantillons utilisés pour chaque arbre)
— **objective** : multi :softmax (pour un problème de classification multiclasse)
— **num_class** : 7 (nombre de classes dans la classification)
— **random_state** : 42 (pour assurer la reproductibilité)

LightGBM
— **n_estimators** : 200 (nombre d'itérations de boosting)
— **max_depth** : 15 (profondeur maximale des arbres)
— **learning_rate** : 0.0843 (taux d'apprentissage pour les mises à jour des arbres)
— **colsample_bytree** : 0.6285 (fraction des caractéristiques utilisées pour chaque arbre)
— **subsample** : 0.6895 (fraction des échantillons utilisés pour chaque arbre)
— **num_leaves** : 100 (nombre maximal de feuilles dans chaque arbre)
— **objective** : multiclass (adapté à un problème de classification multiclasse)
— **num_class** : 7 (nombre de classes dans la classification)
— **random_state** : 42 (pour assurer la reproductibilité)
— **verbosity** : -1 (pour réduire les sorties inutiles)

ExtraTrees
— **n_estimators** : 300 (nombre d'arbres dans la forêt)
— **max_depth** : None (profondeur maximale non limitée)
— **random_state** : 42 (pour assurer la reproductibilité)
— **max_features** : None (toutes les caractéristiques sont considérées pour chaque division)
— **min_samples_split** : 2 (nombre minimal d'échantillons requis pour effectuer une division interne)
— **min_samples_leaf** : 1 (nombre minimal d'échantillons requis pour former une feuille)
— **n_jobs** : -1 (utilise tous les cœurs disponibles pour accélérer l'entraînement)

Gradient Boosting
— **n_estimators** : 300 (nombre d'étapes de boosting à effectuer)
— **learning_rate** : 0.1178 (taux d'apprentissage pour ajuster les contributions des arbres)
— **max_depth** : 16 (profondeur maximale des arbres, permettant de capturer des relations complexes)
— **subsample** : 0.7410 (fraction des échantillons utilisés pour construire chaque arbre)
— **min_samples_split** : 5 (nombre minimal d'échantillons requis pour effectuer une division interne)
— **min_samples_leaf** : 6 (nombre minimal d'échantillons requis pour former une feuille)
— **random_state** : 42 (pour assurer la reproductibilité)

- Méta-modèle

Le méta-modèle est alors XGBoost et ses hyperparamètres sont :

— **n_estimators** : 160 (nombre maximal d'arbres construits)
— **max_depth** : 2 (profondeur maximale des arbres, limitant la complexité pour éviter le surapprentissage)
— **learning_rate** : 0.1 (taux d'apprentissage pour ajuster les contributions des arbres)
— **colsample_bytree** : 0.92 (fraction des caractéristiques à utiliser pour chaque arbre)
— **subsample** : 0.75 (fraction des échantillons utilisés pour chaque arbre)
— **random_state** : 42 (pour assurer la reproductibilité)
— **use_label_encoder** : False (pour éviter l'encodage automatique des labels)
— **eval_metric** : mlogloss (fonction de perte utilisée pour évaluer la qualité du modèle)

Concernant la validation croisée servant à évaluer les performances du modèle de stacking et ajuster ses paramètres, nous avons choisi un nombre split égal à **20**.

- Prétraitement et Feature Engineering

Malgré les nombreux essais qui ont été détaillés dans la partie précédente, le seul prétraitement que nous avons retenu est donc l'élimination de l'attribut `Hillshade_9am`.

4.2 Performance

Sur le dataset de validation, on obtient ainsi une précision de **90,08%**. Cette précision est très satisfaisante, et lorsque l'on réalise les prédictions sur le dataset de test fourni sur la page Kaggle, la précision atteint **85,40%**.

Pour analyser plus en détail les performances de prédictions, analysons les prédictions sur le data de validation, étant donné que l'on a aussi accès aux véritables valeurs de `Cover_Type`. Le tableau 3 donne, entre autres, le détail de la précision selon la véritable valeur de `Cover_Type`.
Alors que les classes $\{4, 5, 7\}$ sont très bien prédites, les classes $\{1, 2\}$ ne connaissent pas la même performance. Plus précisément, on peut regarder la matrice de confusion (figure 10) qui donne plus d'informations sur les classes qui sont souvent confondues.

Classe	Precision	Recall	F1-Score	Support
1	0.83	0.80	0.81	648
2	0.80	0.80	0.80	648
3	0.89	0.92	0.91	648
4	0.98	0.97	0.97	648
5	0.95	0.94	0.95	648
6	0.90	0.93	0.92	648
7	0.96	0.94	0.95	648
Accuracy	0.90 (sur 4536 échantillons)			
Macro avg	0.90	0.90	0.90	4536
Weighted avg	0.90	0.90	0.90	4536

Table 3 – Rapport de classification pour le modèle de stacking final

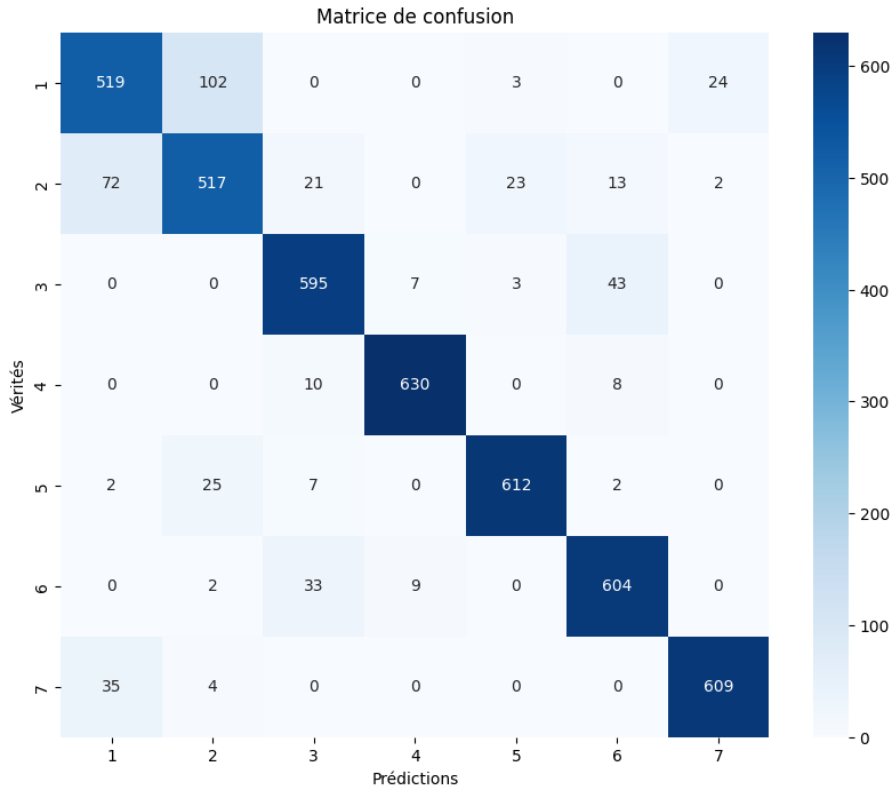


Figure 10 – Matrice de confusion sur les prédictions finales

On retrouve le fait que les classes $\{1, 2\}$ sont souvent confondues, alors que les classes $\{4, 5, 7\}$ sont parfaitement distinguées entre elles, comme on l'avait déjà anticipé lors de l'analyse de données en amont.

On peut aussi relier cette confusion entre $\{1, 2\}$ au fait que ces deux classes sont sous-représentées dans le dataset d'entraînement par rapport au dataset de test, sur lequel la précision est moins bonne. On peut alors faire l'hypothèse que, comme vu lors de l'analyse des données, la mauvaise représentativité du jeu de données d'entraînement entraîne de l'overfitting qui réduit alors la précision obtenue au final sur le dataset de test.

4.3 Étude du modèle

Pour comprendre et interpréter notre modèle, nous avons étudié l'importance que le méta-modèle accordait à chaque sous-modèle en analysant leurs contributions respectives dans le cadre du stacking. Cette analyse permet de comprendre quels modèles de base jouent un rôle plus significatif dans les prédictions finales.

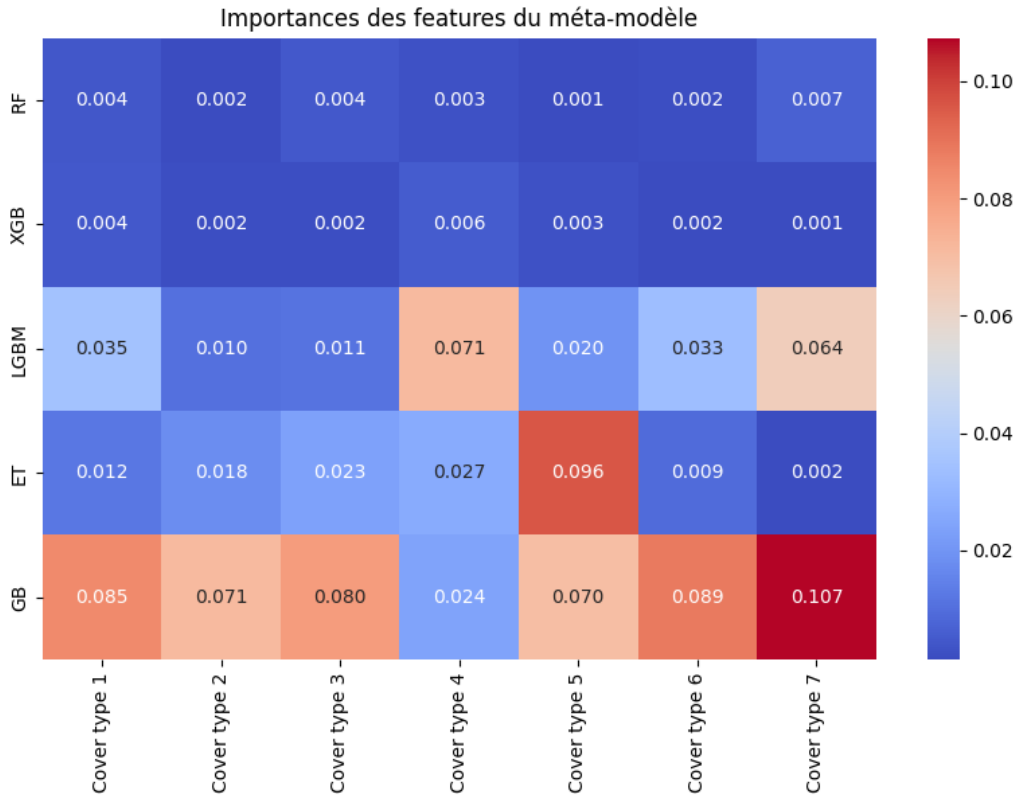


Figure 11 – Importance donnée à chaque sortie de chaque sous-modèle

En observant les pondérations attribuées aux sorties des sous-modèles (figure 11), nous remarquons que Gradient Boosting est le modèle qui a les plus grandes valeurs d'importance, ce qui est normal car, individuellement, c'est celui qui a la meilleure précision.

Cependant, ponctuellement, les autres modèles peuvent avoir une grande importance lorsqu'ils prédisent un certain `Cover_Type`, qui correspond au `Cover_Type` qu'ils prédisent le mieux comparativement aux autres modèles.

Prenons l'exemple de **Extra Trees**, qui a une grande importance lorsqu'il prédit le `Cover_Type` 5. En observant son rapport de classification (tableau 4), on remarque que le `Cover_Type` 5 est en effet celui où il a le plus de précision, c'est-à-dire **0.95**.

Cela confirme ainsi la complémentarité des modèles dans la combinaison des prédictions.

Classe	Précision	Recall	F1-Score	Support
1	0.79	0.79	0.79	648
2	0.80	0.72	0.76	648
3	0.88	0.90	0.89	648
4	0.95	0.98	0.97	648
5	0.92	0.95	0.94	648
6	0.89	0.89	0.89	648
7	0.94	0.95	0.94	648
Accuracy	0.88 (sur 4536 échantillons)			
Macro avg	0.88	0.88	0.88	4536
Weighted avg	0.88	0.88	0.88	4536

Table 4 – Rapport de classification pour le modèle ExtraTrees

A
SIGNIFICATION DES CATÉGORIES DES ATTRIBUTS QUALITATIFS

La signification des valeurs que peut prendre `Cover_Type` est celle-ci :
- 1 = "Spruce/Fir"
- 2 = "Lodgepole Pine"
- 3 = "Ponderosa Pine"
- 4 = "Cottonwood/Willow"
- 5 = "Aspen"
- 6 = "Douglas-fir"
- 7 = "Krummholz"

La signification des valeurs que différentes catégories de `Wilderness_Area` est celle-ci :
- 1 = "Rawah Wilderness Area"
- 2 = "Neota Wilderness Area"
- 3 = "Comanche Peak Wilderness Area"
- 4 = "Cache la Poudre Wilderness Area"