PREDICTING FOREST COVER TYPE - CHALLENGE KAGGLE

APM-51053-EP - Foundations of Machine Learning

10 Janvier 2025

Ethan Cohen, Arthur Iffenecker, Tanguy Azema, Jules Cognon

TABLE OF CONTENTS

INTRODUCTION

The objective of the Kaggle challenge we participated in was to predict forest cover types ("Cover Type") based on a set of information related to a given geographical area. To achieve these predictions, we developed classification models that predict the Cover Type of an area based on the provided dataset.

To assist us in building our model, three CSV files were provided : a train.csv file containing the training data, a test-full.csv file serving as the dataset to test our model, and a full-submission.csv file providing an example of an acceptable submission format.

The data included 55 parameters (features) of various types. Some parameters were continuously distributed (for example, Elevation, Slope), while others were represented in a binary manner (for instance, Soil-Type1). The target parameter for our model was Cover Type.

We thus worked on optimizing the provided datasets for training our models, and then we utilized several models such as Random Forest, Extra Trees, KNN, and XGBoost. This allowed us to obtain initial results, become familiar with these tools, and understand their operation. Finally, we employed several model enhancement methods.

The final model we used to make our predictions allowed us to obtain a score of **0.84241**

1 DATA STUDY

1.1 Exploratory Data Analysis (EDA)

Let's first understand the structures and relationships within our data. To do this, we will examine the various features, their correlations, and distributions. This will help us determine appropriate approaches to solve our problem.

1.1.1 • Types of Variables

Our variables are numerous, describing geographical and environmental characteristics of forest plots such as elevation, slope, distance to water points, or soil type. They provide key information to differentiate forest cover types present in each plot. Some variables are continuous, others discrete, and others binary.

Continuous variables (numeric) :

- **Elevation** : Altitude of a plot, expressed in meters (**continuous numeric**).
- **Aspect** : Orientation of the plot relative to the sun, measured in azimuth degrees (**continuous numeric**).
- **Slope** : Slope inclination, measured in degrees (**continuous numeric**).
- **Horizontal Distance to Nearest Water Point** : Measured in meters (**continuous numeric**).
- **Vertical Distance to Nearest Water Point** : Measured in meters (**continuous numeric**).
- **Horizontal Distance to Nearest Road** : Measured in meters (**continuous numeric**).
- **Hillshade at 9am** : Shading index at the summer solstice at 9am, scale 0 to 255 (**continuous numeric**).
- **Hillshade at noon** : Shading index at the summer solstice at noon, scale 0 to 255 (**continuous numeric**).
- **Hillshade at 3pm** : Shading index at the summer solstice at 3pm, scale 0 to 255 (**continuous numeric**).
- **Horizontal Distance to Fire Points** : Distance to fire ignition points, measured in meters (**continuous numeric**).

Binary categorical variables (0 or 1) :

- **Wilderness Areas** : Presence or absence of a specific wilderness area. These variables are represented by 4 binary columns (**binary qualitative**).
- **Soil Type** : Soil type of the plot. These variables are encoded in 40 binary columns where 1 indicates the specific soil type (**binary qualitative**).

Target Variable :

- **Forest Cover Type** : Indicates the forest type present on the plot, with 7 possible categories (**multiclass categorical variable**).

1.1.2 • Correlation Between Variables

Let's study correlations among the different features, starting with continuous variables.

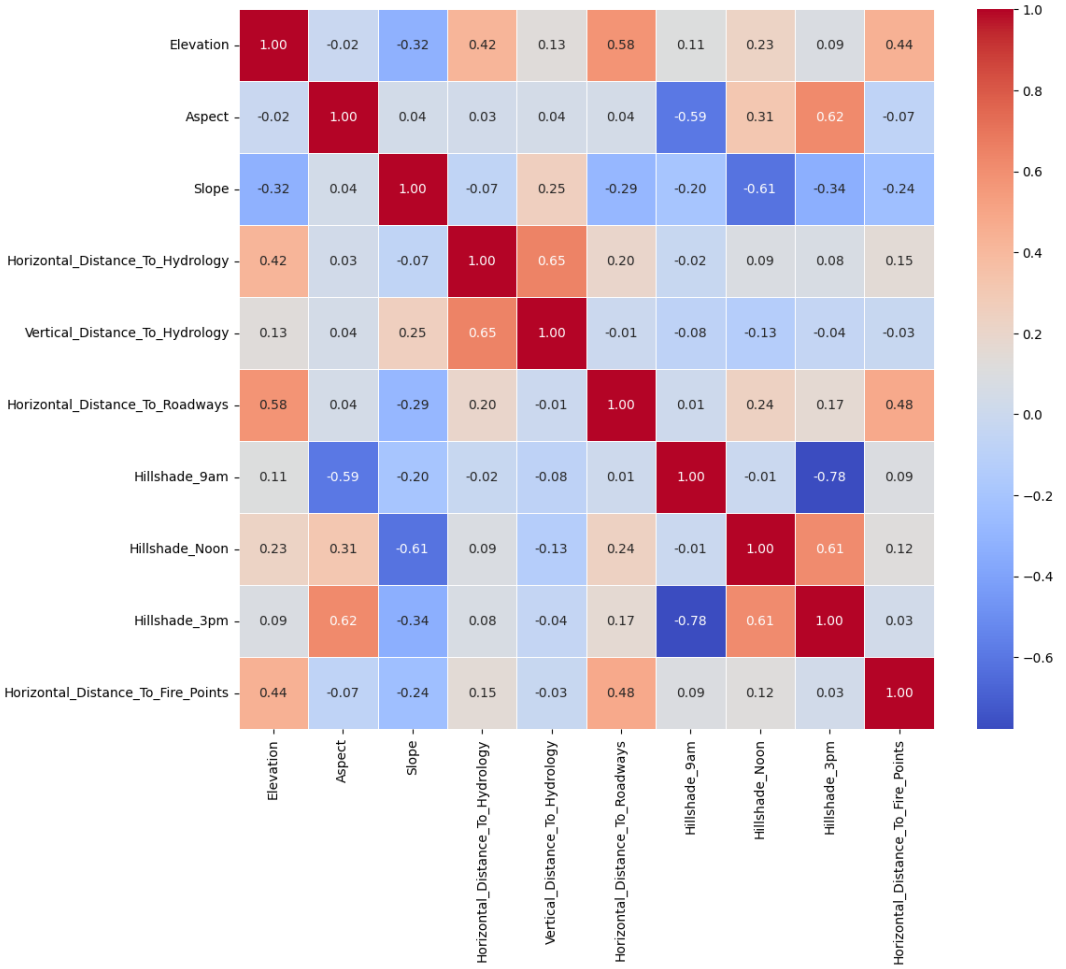


Figure 1 – Correlation matrix of continuous variables (train set)

The correlation matrix shows logical and interesting relationships between continuous variables. For example, **Vertical Distance To Hydrology** and **Horizontal Distance To Hydrology** have moderate correlation (0.65), suggesting related aspects of water distance. Shade indices, especially **Hillshade 9am** and **Hillshade 3pm**, have a strong negative correlation (-0.78), indicating opposite slope orientations depending on the hour. Variables such as **Aspect** show very weak correlations with others, suggesting relative independence in the dataset.

To identify trends with different categories of our target variable, we look at correlations between each variable and each forest cover type.

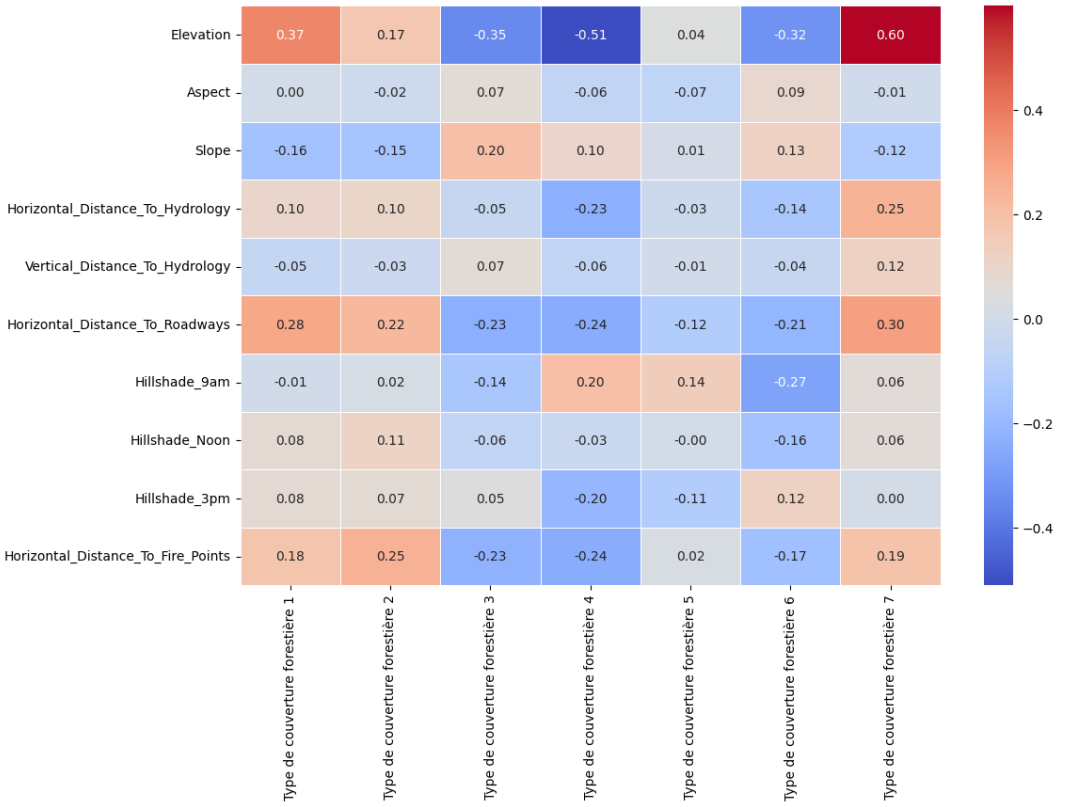


Figure 2 – Correlation matrix of continuous variables with each forest cover type

We can already notice important relationships :

- **Separability of Forest Cover Types** :
  - Type **7** is distinct, with strong positive correlation to **Elevation** (0.60).
  - Types **5** and **6** are difficult to separate, as their correlations with continuous variables are weak and similar.
  - **Forest cover types 1 and 2 will be difficult to separate.**
- **Main Discriminative Variable** :
  - **Elevation** is the most discriminative variable, with significant correlations for multiple classes (positive for **1** and **7**, negative for **4**).
- **Secondary Variables** :
  - **Horizontal_Distance_To_Fire_Points** slightly helps distinguish certain classes, notably type **2** (0.25).
  - Shade indices (**Hillshade_9am**, **Hillshade_Noon**, **Hillshade_3pm**) and **Aspect** show weak correlations, limiting their discriminative role.

Finally, correlations remain weak overall, suggesting the need for feature engineering later.

Now, let's examine non-continuous variables through correlations with forest cover types, using two heatmaps.

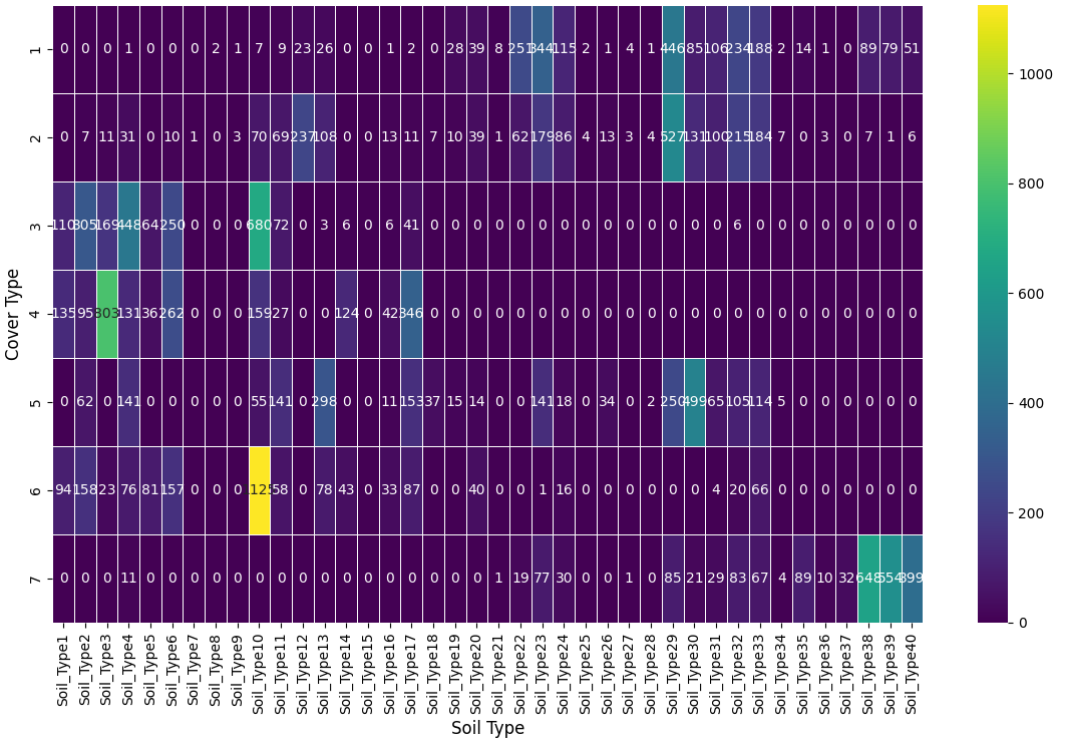


Figure 3 – Correlation matrix of soil types with forest cover types

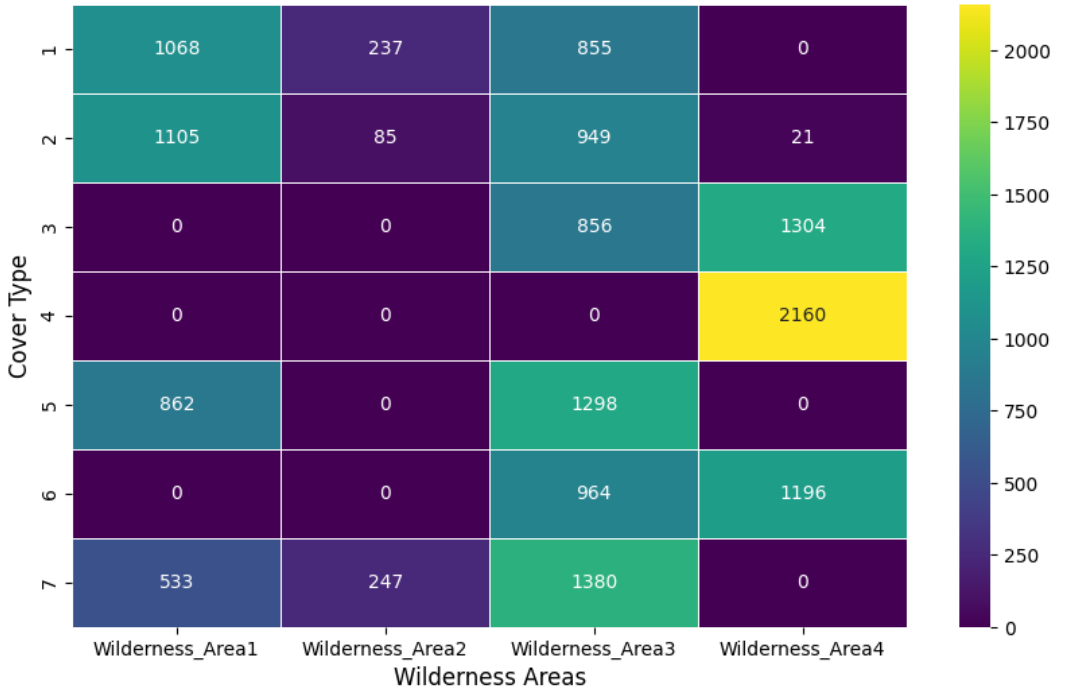


Figure 4 – Correlation matrix of wilderness areas with forest cover types

Analysis of soil types and wilderness areas heatmaps provides further insights :

The soil type heatmap reveals strong associations between specific categories and forest cover types. For instance, **Soil_Type40** is nearly exclusively linked to cover type **7**, while **Soil_Type10** clearly corresponds to type **6**. Conversely, types like **Soil_Type1** to **Soil_Type9** show diffuse relationships across multiple cover types, indicating less influence.

The wilderness areas heatmap highlights even stronger relationships. **Wilderness_Area4** strongly relates to cover type **4**, making it a key indicator. Similarly, **Wilderness_Area3** primarily links to types **5** and **7**. Other areas like **Wilderness_Area1** and **3** are shared among multiple cover types (**1**, **5**, **7**), showing less exclusivity.

Together, wilderness areas and soil types provide complementary information. Wilderness areas capture global geographic characteristics, useful for identifying specific classes like type **4**. Soil types add fine-grained differentiation for other classes. Jointly leveraging these variables should improve classification performance.

1.2 Data Preprocessing

Upon examining the distribution of Cover Types in the datasets, we noticed that the Cover Types in the "train" file were evenly distributed, whereas this was not the case for the test file.

```
Répartition des cover type pour train_data
Cover_Type
1    0.142857
2    0.142857
3    0.142857
4    0.142857
5    0.142857
6    0.142857
7    0.142857
```

(a) Distribution of Cover Types in the "train" file

```
Répartition des cover type pour test_data
Cover_Type
1    0.369400
2    0.418418
3    0.065448
4    0.006454
5    0.046342
6    0.041235
7    0.052703
```

(b) Distribution of Cover Types in the "test" file

Figure 5 – Distribution of Cover Types

The test file does not contain a Cover Type column. Therefore, to obtain the distribution for this file, we used our submissions as an approximation.

We observed that Cover Types 1 and 2 were significantly more represented proportionally than the other five types. Moreover, as previously highlighted in our exploratory data analysis, these two Cover Types are relatively challenging to distinguish. To address this and create a training dataset more consistent with the test dataset, we increased the proportion of Cover Types 1 and 2 in our training set. We duplicated rows for Cover Types 1 and 2 (using the RandomOverSampler method) to achieve the desired proportions.

Finally, we generated a new training file with proportions closer to those of the test file.

```
Répartition des cover type pour augmented_train_data
Cover_Type
1    0.338068
2    0.377841
3    0.056818
4    0.056818
5    0.056818
6    0.056818
7    0.056818
```

Figure 6 – New distribution of the training dataset after duplication

1.3 Feature Engineering

To enhance the predictions of our models, it is essential to extract relevant information from the data to better capture underlying patterns. Therefore, we created new variables from the raw data.

- **Creation of New Continuous Variables**

We developed several new variables of interest. To capture the total distance to water points, we created the variable *Distance_To_Hydrology* using vertical and horizontal distances. We also introduced *Average_Hillshade*, averaging shade indices across different times of day to characterize the overall brightness of the location, and *combined_distances*, averaging distances to roads, water points, or fire ignition points. Additionally, to capture non-linear relationships, we added logarithmic and quadratic transformations of elevation, namely *Elevation_log* and *Elevation_square*. These transformations particularly help in distinguishing Cover Types 1 and 2.

- **Addition of Binary Variables**

Using the ELU codes of the 40 soil types, we created new binary variables to account for climatic and geological zones. The variables named *climatic_zone_i* and *geological_zone_i* provide more precise information. By observing their correlation with Cover Types, we removed some and retained others. For example, *climatic_zone_8* effectively differentiates Cover Type 7, whereas *geological_zone_7* is evenly distributed across all Cover Types and was thus discarded.

We attempted to use PCA to reduce data dimensionality, but this resulted in poorer performance, so it was not retained.

- **Variable Interactions**

Capturing interactions between variables can also improve predictions. We created a variable *wilderness_area_elevation_i* multiplying elevation by the wilderness area type index to better capture their combined effect.

- **Normalization of Continuous Data**

We experimented with scaling data using various normalization methods to balance variable importance. Testing standard scaling and min-max scaling led to lower model performance, so these methods were not retained.

2
MODELS USED

In this section, we describe the different models we used throughout this project.

2.1 First Approach Using Simple Algorithms

Initially, we explored classic and relatively simple algorithms to familiarize ourselves with the problem.

2.1.1 • K-Nearest Neighbors Algorithm

This algorithm, also called `K-NN`, unlike models like `Random Forest` or `Extra-Trees` (which we detail later), does not rely on explicit training : it is an instance-based model where no function is actually learned from the training data. When a new data point needs to be predicted, the model searches for the k nearest neighbors in the feature space. However, this principle implies both a high computation time for the dataset we worked on and limited accuracy, which quickly led us to use other algorithms.

2.1.2 • Decision Trees

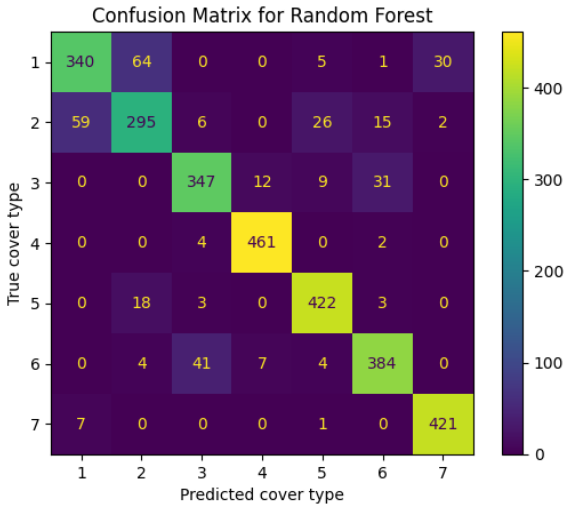
`Decision Trees` are simple yet powerful algorithms. They work by splitting the data into successive segments based on optimal split criteria, forming a tree-like structure. `Decision Trees` are a good starting point due to their simplicity and interpretability. They are often used as base components in ensemble models (`Random Forest`, `Gradient Boosting`) to overcome their limitations in generalization. Understanding this algorithm helped us later to better approach more complex algorithms, which we now present and which allowed us to improve our model's performance.

2.2 Bagging Algorithms

Bagging (Bootstrap Aggregating) is an ensemble method that combines several weak learners (often the same type of model) to create a more robust and accurate global model. The key idea is to train each model on a different subset of the data obtained through bootstrapping (sampling with replacement) and to combine their predictions, usually by voting in classification tasks. We specifically used two bagging algorithms : `Random Forest` and `Extra Trees`.

2.2.1 • Random Forest

`Random Forest` is a machine learning algorithm based on an ensemble of decision trees. It combines the predictions of multiple trees (a "forest") to improve accuracy and reduce overfitting risk. Each tree is built using a random subset of data and features, introducing diversity and enhancing robustness. This algorithm accepts a maximum tree depth as a parameter. Increasing this depth improved results, but deeper trees also increased overfitting risk. Thus, we had to find a balance to ensure strong performance while avoiding overfitting.



(a) Confusion matrix for the `Random Forest` algorithm

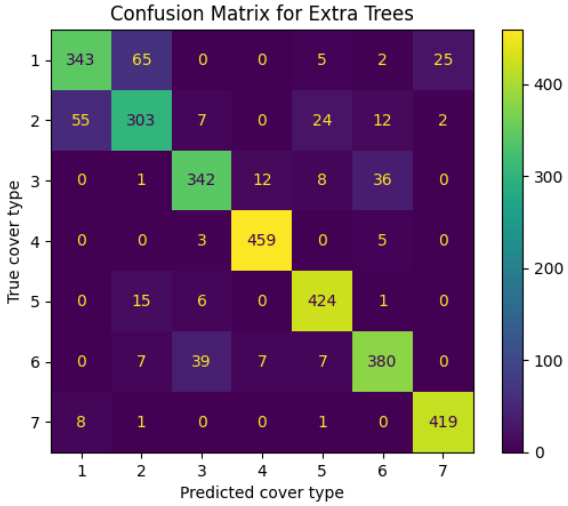
	precision	recall	f1-score	support
1	0.84	0.77	0.80	440
2	0.77	0.73	0.75	403
3	0.87	0.87	0.87	399
4	0.96	0.99	0.97	467
5	0.90	0.95	0.92	446
6	0.88	0.87	0.88	440
7	0.93	0.98	0.95	429
accuracy			0.88	3024
macro avg	0.88	0.88	0.88	3024
weighted avg	0.88	0.88	0.88	3024

(b) Results by Cover Type

We observe that `Random Forest` performs very well for Cover Types 3 to 7. However, predictions are slightly less accurate for Cover Type 1 and significantly lower for Cover Type 2, which is often confused with Type 1, confirming the similarity between these two types.

2.2.2 • Extra Trees

`Extra Trees` (or Extremely Randomized Trees) is a similar algorithm to `Random Forest` but with even more randomness in tree construction. Like `Random Forest`, it uses an ensemble of decision trees for predictions, but with a key difference : it selects split thresholds at random at each node rather than searching for the best one based on a purity criterion. `Extra Trees` is faster to train and has lower variance due to this randomness, although this comes at the cost of increased bias.



(a) Confusion matrix for the `Extra Trees` algorithm

	precision	recall	f1-score	support
1	0.84	0.78	0.81	440
2	0.77	0.75	0.76	403
3	0.86	0.86	0.86	399
4	0.96	0.98	0.97	467
5	0.90	0.95	0.93	446
6	0.87	0.86	0.87	440
7	0.94	0.98	0.96	429
accuracy			0.88	3024
macro avg	0.88	0.88	0.88	3024
weighted avg	0.88	0.88	0.88	3024

(b) Results by Cover Type

The results for `Extra Trees` are very close to those of `Random Forest`, which is expected given the similarity of the two algorithms.

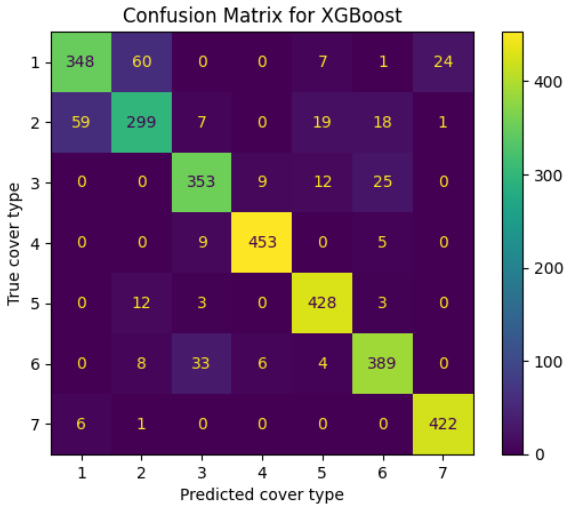
2.3 Boosting Algorithms

Boosting algorithms aim to transform weak learners into a strong learner. The idea is to build models sequentially, where each new model corrects the errors of the previous ones.

2.3.1 • XGBoost

`XGBoost` (Extreme Gradient Boosting) is also based on decision trees. However, `XGBoost` belongs to the boosting family, where trees are built sequentially to correct previous errors. The algorithm performs gradient boosting, optimizing a loss function by computing gradients and adjusting predictions accordingly. Aside from gradient descent, it is quite similar to `Random Forest`.

However, `XGBoost`, when used for multi-class classification problems, requires the target variable to be in integer form (0, 1, 2, etc.) rather than categorical or text form. In our dataset, the Cover Type target variable was originally represented as integers from 1 to 7. We therefore transformed it using LabelEncoding.



(a) Confusion matrix for the `XGBoost` algorithm

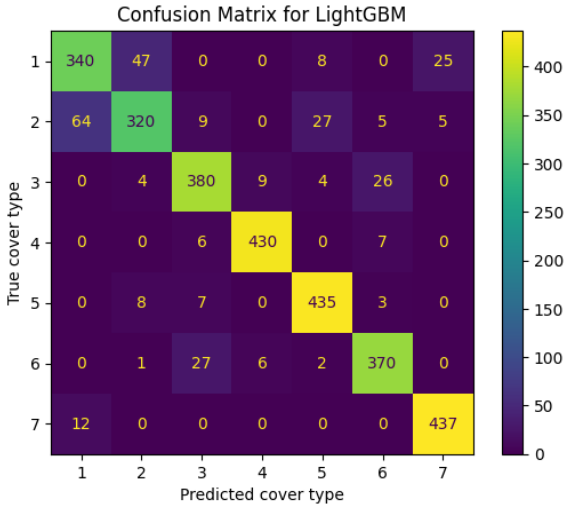
	precision	recall	f1-score	support
1	0.84	0.79	0.82	440
2	0.79	0.74	0.76	403
3	0.87	0.88	0.88	399
4	0.97	0.97	0.97	467
5	0.91	0.96	0.93	446
6	0.88	0.88	0.88	440
7	0.94	0.98	0.96	429
accuracy			0.89	3024
macro avg	0.89	0.89	0.89	3024
weighted avg	0.89	0.89	0.89	3024

(b) Results by Cover Type

The results after using XGBoost are overall more satisfactory for Cover Types 2 to 7, and similar for Cover Type 1 compared to bagging algorithms

2.3.2 • LightGBM

`LightGBM` is also a gradient boosting algorithm. Like `XGBoost`, it builds trees sequentially, with each new tree correcting the errors of the previous ones by minimizing the loss function. However, unlike `XGBoost`, `LightGBM` always splits the leaf with the greatest gain, allowing for better local optimization. This approach can result in unbalanced trees that are nonetheless highly effective.



(a) Confusion matrix for the `LightGBM` algorithm

	precision	recall	f1-score	support
1	0.82	0.81	0.81	420
2	0.84	0.74	0.79	430
3	0.89	0.90	0.89	423
4	0.97	0.97	0.97	443
5	0.91	0.96	0.94	453
6	0.90	0.91	0.91	406
7	0.94	0.97	0.95	449
accuracy			0.90	3024
macro avg	0.89	0.90	0.89	3024
weighted avg	0.90	0.90	0.90	3024

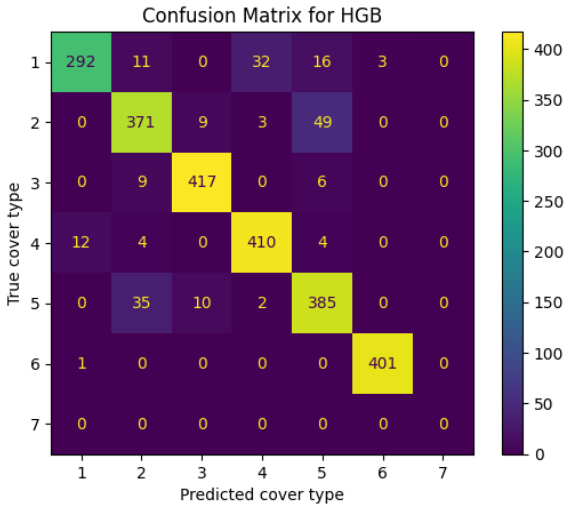
(b) Results by Cover Type

The most significant difference in results between `LightGBM` and the other models is its improved performance on Cover Type 2.

2.3.3 • Histogram Gradient Boosting

The `Histogram Gradient Boosting (HGB)` algorithm is a variant of Gradient Boosting that uses histograms to accelerate the learning process. Continuous data is grouped into bins, reducing computational cost while maintaining good performance. Like `XGBoost` and `LightGBM`, `HGB` builds trees sequentially, with each tree correcting the errors of the previous ones by minimizing the loss function.

For `Histogram Gradient Boosting`, results on Cover Type 1 are significantly lower than those of the other algorithms, making it less effective unless optimized.



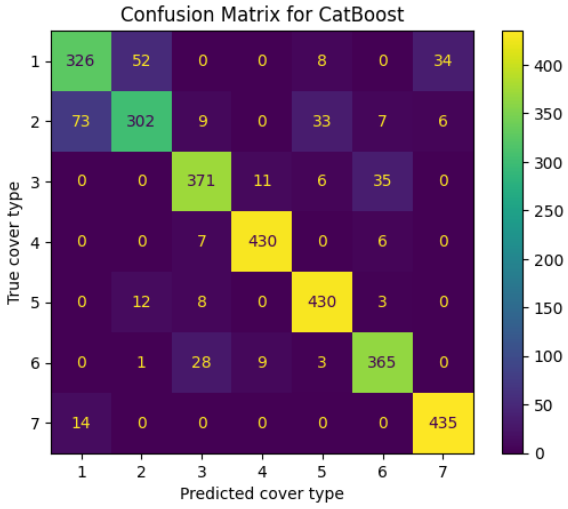
(a) Confusion matrix for the `Histogram Gradient Boosting` algorithm

	precision	recall	f1-score	support
0	0.75	0.75	0.75	432
1	0.77	0.68	0.72	432
2	0.86	0.86	0.86	432
3	0.96	0.97	0.96	432
4	0.91	0.95	0.93	432
5	0.84	0.89	0.86	432
6	0.92	0.93	0.93	432
accuracy			0.86	3024
macro avg	0.86	0.86	0.86	3024
weighted avg	0.86	0.86	0.86	3024

(b) Results by Cover Type

2.3.4 • CatBoost

Finally, we also used the `CatBoost` model, which did not achieve the same level of performance as the other algorithms. The main difference between `CatBoost` and an algorithm like `XGBoost` lies in its encoding : `CatBoost` uses permutations and relies on the random order of the data to reduce overfitting and bias.



(a) Confusion matrix for the `CatBoost` algorithm

	precision	recall	f1-score	support
1	0.79	0.78	0.78	420
2	0.82	0.70	0.76	430
3	0.88	0.88	0.88	423
4	0.96	0.97	0.96	443
5	0.90	0.95	0.92	453
6	0.88	0.90	0.89	406
7	0.92	0.97	0.94	449
accuracy			0.88	3024
macro avg	0.88	0.88	0.88	3024
weighted avg	0.88	0.88	0.88	3024

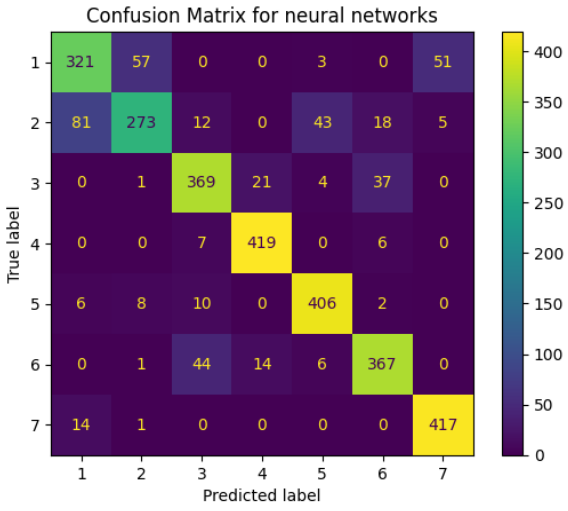
(b) Results by Cover Type

We observed similar results for `CatBoost` and `Histogram Gradient Boosting` regarding Cover Type 1, though `CatBoost` performed better on Cover Type 2.

2.4 Neural Networks

A neural network is a machine learning algorithm inspired by the functioning of the human brain. It relies on an architecture composed of layers of interconnected neurons that learn to detect complex patterns in the data. Each neuron applies an activation function to a weighted linear combination of inputs, allowing the model to capture non-linear relationships. Learning is performed through backpropagation, where connection weights are adjusted to minimize a loss function.

We used a multilayer model for our project. The model's performance is highly dependent on several hyper-parameters, including the number of hidden layers, the number of neurons per layer, the number of epochs, the learning rate, and the dropout rate (used to prevent overfitting). Since the data lacks specific structure, neural networks are not necessarily the most suitable. In testing, our model achieved a cross-validation score of 0.79 on the test set.



(a) Confusion matrix for the neural network

	precision	recall	f1-score	support
1	0.76	0.74	0.75	432
2	0.80	0.63	0.71	432
3	0.83	0.85	0.84	432
4	0.92	0.97	0.95	432
5	0.88	0.94	0.91	432
6	0.85	0.85	0.85	432
7	0.88	0.97	0.92	432
accuracy			0.85	3024
macro avg	0.85	0.85	0.85	3024
weighted avg	0.85	0.85	0.85	3024

(b) Results by Cover Type

These results are unsatisfactory, particularly for Cover Types 1, 3, and 6. This contrasts with the other models, which generally performed well on Cover Types 3 to 7.

3
REFINING THE RESULTS

After identifying a shortlist of promising models (`XGBoost`, `Random Forest`, `Extra Trees`, and `Histogram Gradient Boosting`), we set out to optimize them using various approaches.

3.1 Hyperparameter Optimization

First, we aimed to optimize the hyperparameters. To do this, we used the `GridSearchCV` tool from the Scikit-Learn library, specifying the hyperparameters to test and their potential ranges. However, our models had large hyperparameter spaces. For example, for the Histogram Gradient Boosting model, we explored five main hyperparameters : *learning rate*, *max_leaf_nodes*, *max_iter*, *l2_regularization*, and *max_bins*.

The exhaustive approach of `GridSearchCV`, which tests all possible combinations, quickly became too time-consuming, especially for complex models. Therefore, we adopted `RandomizedSearchCV`, which limits the number of iterations while effectively exploring the hyperparameter space. This strategy allowed us to quickly identify promising ranges.

Next, once these narrower ranges were defined, we used `GridSearchCV` again to refine results by exhaustively searching within this smaller region. Below is an example of the narrowed hyperparameter ranges for the XGBoost model used in this refinement phase :

```
param_grid_xgb = {
    'n_estimators': [50, 100, 200],
    'max_depth': [4, 5, 6],
    'learning_rate': [0.01, 0.05, 0.1],
    'subsample': [0.7, 0.8, 0.9],
    'colsample_bytree': [0.8, 0.9, 1.0]
}
```

Figure 14 – Reduced hyperparameter ranges for XGBoost

Ultimately, this strategy enabled us to improve the performance of our models.

We then further boosted our scores, especially for the XGBoost model, by using the generic optimization library `Optuna`, discussed in PC. This method was particularly relevant for exploring our mixed parameters, both continuous and discrete. We defined the search ranges via `trial.suggest_float`, building on the optimal results obtained with previous optimization methods. This adaptive approach allowed for more targeted, efficient hyperparameter exploration while reducing computation time. As a result, we observed an improvement in the overall performance of our XGBoost model.

In parallel, we also included Bayesian optimization in our strategy to combine exploration and exploitation, using the `BayesSearchCV` tool from `scikit-optimize`. This method, especially suited to complex hyperparameter spaces, helped us refine the performance of the `Histogram Gradient Boosting` model and achieve our best score on 30% of the test data (0.84198). The optimization was performed over 100 iterations (`n_iter=100`) with 5-fold cross-validation (`cv=5`) and accuracy as the metric (`scoring='accuracy'`).

3.2 Ensemble Methods

Next, we used Ensemble Methods to combine the models that achieved the best performances. We began by implementing a `Voting Classifier` that combined `XGBoost`, `Histogram Gradient Boosting`, `RandomForest`, and `ExtraTrees`. We first tested hard voting, which aggregates each classifier's predictions and chooses the class that received the most votes. However, we found better results with soft voting, where the predicted class is the one with the highest average probability across all classifiers. To further refine results, we assigned weights to the classifiers according to their respective performances, as illustrated below :

```python
# Pondération des modèles
weights = [0.3, 0.3, 0.2, 0.2]
weighted_voting_clf = VotingClassifier(
    estimators=[
        ('xgb', xgb_model),
        ('hgb', hgb_model),
        ('rf', rf_model)
        ('extra', et_model)
    ],
    voting='soft',
    weights=weights
)
```

Figure 15 – Classifier weighting in the Voting Classifier

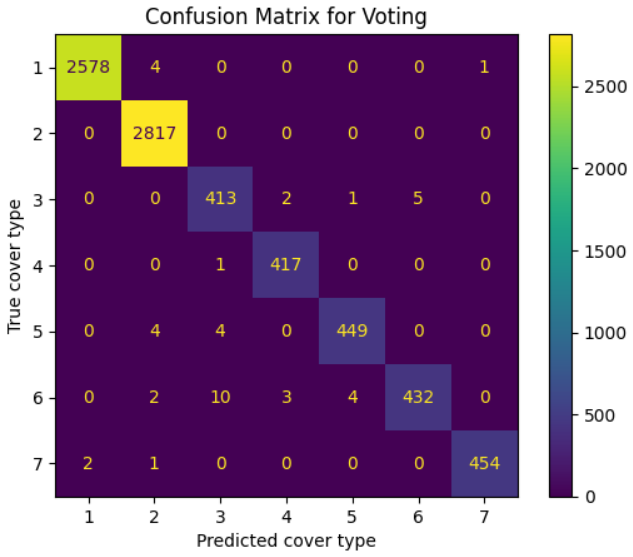
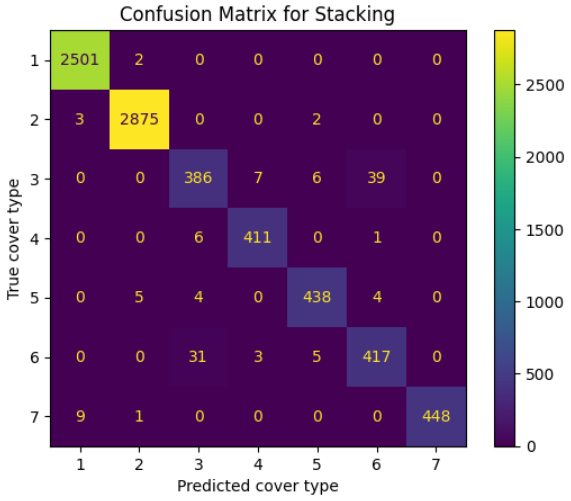


Figure 16 – Confusion matrix for the voting method

The results shown in this matrix are particularly high, especially compared to the significantly lower final results upon submission. Thus, we can infer that our voting method may have overfitted the test dataset.

Finally, we explored stacking. Unlike voting, this approach involves training a model (called a blender) to combine the predictions of various predictors. Our final blender was trained on the predictions of three base models : `RandomForest`, `XGBoost`, and `Histogram Gradient Boosting`. We used `LogisticRegression` as our final estimator.



(a) Confusion matrix for the `stacking` method

	precision	recall	f1-score	support
1	1.00	1.00	1.00	2503
2	1.00	1.00	1.00	2880
3	0.90	0.88	0.89	438
4	0.98	0.98	0.98	418
5	0.97	0.97	0.97	451
6	0.90	0.91	0.91	456
7	1.00	0.98	0.99	458
accuracy			0.98	7604
macro avg	0.96	0.96	0.96	7604
weighted avg	0.98	0.98	0.98	7604

(b) Results by Cover Type

Based on the confusion matrix and the Cover Type results chart, stacking likely also caused overfitting with respect to the training dataset, as the results upon submission were significantly lower.

We first observed that neither of these two ensemble methods significantly improved overall performance. Moreover, the `VotingClassifier` produced better results than stacking. This is likely because the base models used are already high-performing and fairly homogeneous in terms of accuracy. In this context, a voting method benefits from the models' complementarity, yielding a slight performance boost. However, stacking, while powerful in other situations, does not appear to provide significant added value in this particular case.

CONCLUSION

In conclusion, this project allowed us to build a high-performing predictor, achieving a maximum score of **0.84241** through the `Histogram Gradient Boosting` method optimized via a Bayesian approach.

Our process began by exploring simple models, such as the K-Nearest Neighbors algorithm and decision trees. We then expanded our analysis by testing more sophisticated methods, including neural networks, as well as bagging- and boosting-based models. Ensemble methods showed interesting potential but did not yield as strong results as expected.

This project highlights areas for improvement, particularly in differentiating classes 1 and 2. More advanced feature engineering techniques and strategies for handling imbalanced data could be crucial in further separating these classes.

Ultimately, this project provided valuable experience in applying and comparing different Machine Learning methods, underscoring the importance of model adaptation and hyperparameter tuning. These insights will offer a solid foundation for tackling similar challenges in the future.