Machine Learning Project

Classification of a geographic area

Auteur :
JACQUIER Nathanaël
AMIRAULT Tom
BARRAQUE Baptiste
LABORDE Julien

Février 2025

Preprocessing and feature engineering

The first task is to format the data in order to simplify its manipulation. We began by modifying our training and test geojson files by deleting the properties and geometry blocks and creating new structured features which simply contain all the properties, the coordinates of the polygon and *change_type* in the last column.

Next, we transformed our file into a dataframe, added new features, encoded the categorical variables, and handled missing values by replacing NaNs.
More specifically, we focused on computing the polygon's area, perimeter, and various additional geometric attributes, such as Convex Hull features, Compactness, Elongation ratio, Circularity, Rectangular fit, Bounding box aspect ratio, Number of vertices, and Pairwise distance features.. These geometric features are particularly useful for classifying different areas, as each area possesses distinct properties.

For each sample, the five dates, formatted as DD-MM-YYYY, serve as five distinct features but are not arranged chronologically. To provide meaningful temporal information, we first sorted them in ascending order and then calculated the number of days between each consecutive date, introducing a temporal dimension.

We initially applied one-hot encoding to the urban type and geographic type attributes, generating 5 and 11 new binary features. However, this approach proved ineffective, as the additional 16 features contributed negligible information compared to the other variables. As a result, we opted for target encoding, which significantly improved performance. By replacing each category with a numerical representation based on its relationship with the target variable, target encoding captures more relevant information while reducing feature dimensionality, making it a more efficient encoding method in this context.

To extract information from the color features, we first computed the difference between consecutive dates for mean color values and standard deviation. This helps capture how the color values change over time. For each color, we then computed overall statistics such as the average color intensity over the 5 dates, the standard deviation, the lowest color intensity recorded and the highest color intensity recorded.

Dimensionality Reduction

Depending on the models we were trying, we used different dimensionality reduction techniques. Here we will just detail what we did for the final model we chose, the Random Forest.
After feature engineering, we obtained a dataset with 85 features. To assess the importance of these features and eliminate less relevant or noisy ones, we trained a Random Forest model with 300 trees on the full feature set. This number of trees was chosen arbitrarily, as it provided a reasonable trade-off between performance and computational cost.
Using this trained model, we extracted and plotted feature importances, which allowed us to visualize which features contributed most to the predictions.

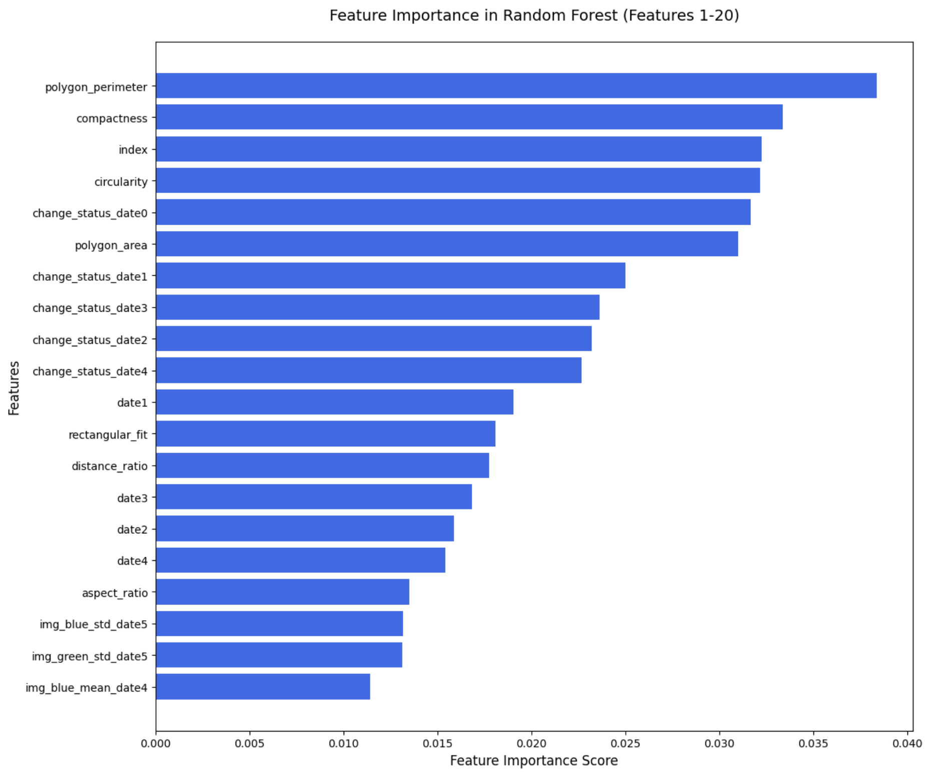

FIGURE 1 – Feature importance plot extracted from the Random Forest model.

By analyzing this plot, we initially considered selecting features based on a fixed importance threshold. However, we observed that the feature importance values were almost continuous, meaning there was no clear cutoff point to separate important and unimportant features. Because of this, setting an arbitrary threshold was not a pertinent approach.
Instead, we decided to make the number of selected features a variable and evaluated its impact on model performance. By keeping the number of trees constant, we experimented with different feature set sizes and measured the corresponding F1 validation scores.

Through this process, we found that the highest F1 score was achieved with approximately 40 features, leading us to retain the top 40 most important features for our final Random Forest model.
This feature selection process helped reduce overfitting, improve interpretability, and speed up training while maintaining strong predictive performance.

Models

Here are some of the models we tried.

Neural Network : We did not consider dimensionality reduction necessary for a neural network, as 85 features is a relatively small number. Despite hyperparameter tuning, we were unable to achieve an F1 score above 0.45, suggesting that the model struggled to capture meaningful patterns in the data.
Gradient Boosting Classifier : For this model, we first applied Principal Component Analysis (PCA), retaining 95 percent of the variance. This transformation helped reduce dimensionality while preserving most of the information. After training the model, we obtained a validation F1 score of 0.65, which was a promising improvement.
Stochastic Gradient Descent (SGD) Classifier : With this model, we were consistently unable to surpass an F1 score of 0.5, even after extensive tuning. The model likely struggled due to the dataset's complexity and the absence of well-separated decision boundaries.
XGBoost : To reduce dimensionality with this model we used LDA (Linear Discriminant Analysis). After training, we managed to get a validation F1 score of 0.69, and a score of 0,71 on Kaggle, suggesting the model was performant but not as much as the RandomForest algorithm.

We ultimately decided to use Random Forest as our final algorithm.
Tuning : After creating a new dataset with only the top 40 features for each example, we needed to tune the hyperparameters of our Random Forest model. To achieve this, we performed Bayesian optimization using the *Optuna* library. The objective function we maximized was the weighted F1 score on a validation set, with the primary goal of avoiding overfitting. The best Random Forest model we found consisted of 149 trees, each with a maximum depth of 42. This model achieved a validation F1 score of 0.74 and a score of 0.89 on Kaggle. The difference between these scores suggests that the model might be overfitting to the test set used on Kaggle. One possible explanation for this could be data leakage, where some examples from the test set are also present in the training set, artificially inflating the Kaggle score.