Kaggle Project Report

ChipiChipi Team

CHAFIK Hala
EL OMARI Chaimae
DEMRI Lina

P2025

February 2024

Contents

Chapter 1

Feature Engineering

1.1 Introduction

Identifying variations in multi-date, multi-image remote sensing data assists in uncovering and comprehending worldwide conditions. This challenge leverages geographic features derived from satellite imagery. Techniques from computer vision have been applied to prepare the data for analysis through machine learning approaches.
The goal of the challenge is to categorize a specified geographic region into one of six classes. In this objective, the first step is processing the data to convert it into an appropriate format. We then explore creating a new set of features from existing ones, and removing what is deemed as marginally useful. Next, we tried different learning algorithms to solve the problem, some (greatly) more efficient than others. Finally, we used different metrics to evaluate our models, mainly **f1-macro** and **accuracy**.

1.2 Handling non-numerical features and missing values

Data preprocessing holds a pivotal role, if not the most crucial, in any machine learning project.
Initially, we employed one-hot encoding on the features **'urban_type'** and **'geography_type'** in order to deal with numerical features rather than strings. However, upon closer examination, we could clearly see that, for the majority of data points, there was not only one geography or urban type corresponding to it. Therefore, the first step was to split these categorical features before applying one-hot encoding.

Another issue we encountered was dealing with the **'N,A'** values; denoting missing or Non-Available values. Opting not to discard rows with missing values, we retained them while excluding columns **N** and **A** generated during the one-hot encoding process.

In the same context, we adopted the Label Encoding technique to convert categorical features related to **'change_status_date'**. Given the ordinal nature of these categories (*e.g* **'Construction Done', 'Construction Midway'** etc.), we deemed this type of

encoder suitable for the task.

1.3 Ordering dates and creating new temporal features

Following that, the temporal aspect was addressed by ordering the dates of each observation's construction process. In fact, while examining the data, we could see that the dates were not arranged in sequential order. In other words, **date0**, for instance, would not necessarily align with the first step of the construction process.

However, it is crucial to note that there are several features which depend on the dates, namely **change_status_datei**, **img_color_mean_datei**, **img_color_std_datei**, where **color** $\in$ {*red, blue, green*}, and **i** $\in$ {0,1,2,3,4}. Thus, the data must be reorganized.

We then computed the number of days between two consecutive dates, acknowledging the construction pace variability across different geographic zones.

We also computed the difference between **'date0'** and a reference date for each datapoint and discarded accordingly the date columns as they no longer retained significance for our analysis.

1.4 Introducing geometric features

Furthermore, considering the primary objective of predicting the class of a geographic zone, it is evident that computing geometric properties of each polygon holds significant importance. With that being said, calculations were performed for the **area**, **perimeter**, **number of edges**, and **aspect ratio** of the polygons. At first, we executed a Random Forest algorithm on the data solely considering the area, but upon introducing other geometric properties, the performance of the algorithm increased significantly.
Furthermore, we continued to incorporate additional geometric properties to evaluate the models' performance. Ultimately, we introduced features such as the ratio of the area to the perimeter squared, as well as the width and height of the minimum bounding box.

1.5 Dimensionality Reduction

We first wanted to execute PCA in order to perform a Dimensionality Reduction. However, the performance of the models drastically dropped. This could be due to the absence of an underlying linear data structure. With that being said, out dataset was composed of 200.000 data points and 65 features.

Chapter 2

Model Tuning and Comparison

Below are the models we explored to determine which best fits our problem:

2.1 Logistic Regression

We applied the multinomial logistic regression model and sought to enhance its performance by incorporating regularization techniques (L2 for instance to prevent overfitting). We also kept adjusting other hyperparameters such as the regularization strength (**'C'** parameter and the maximum number of iterations).
Despite being computationnally efficient, the accuracy is very poor here. This is surely due to the fact that this model is too simple and that our problem requires more complexity.

2.2 Support Vector Machines

It takes forever to run due to the enormous amount of data. Indeed, although SVMs work well in high-dimensional spaces, they are only suitable for small to medium datasets, which is not the case here.

2.3 k-Nearest Neighbors

We tried the k-NN model with different values of k ranging from 5 to 20, with peak performance achieved for 11-NN (highest **f1-score** and a hardly better accuracy) after applying PCA retaining 95% of the original variance, in order to avoid the curse of dimensionality.

2.4 XGBoost

XGBoost, short for Extreme Gradient Boosting, is an ensemble learning method that combines the predictions of multiple weak learners, typically decision trees, to create a robust and accurate predictive model.
It prevents overfitting by incorporating L1 regularization (Lasso) and L2 regularization

(Ridge) into its optimization process.
In our parameter tuning procedure for the XGBoost classifier, we opted for a manual approach due to the time-intensive nature of grid search. We focused on two key hyper-parameters, namely **n_estimators** and **learning_rate**.
N.B: Although XGBoost reaches an accuracy of 0.966 on the Kaggle test, it still does not outperform Random Forest, despite the increasing of the number of trees.

2.5 Random Forest

Same as for XGBoost, we kept manually varying the number of estimators as well as the maximum depth. The most optimal prediction was achieved using about 200 estimators for a depth of 45 and the running time for training was about 40 minutes. **Random Forest was our most effective model**. Random Forests are known for their high accuracy in many real-world applications. By averaging or taking the majority vote of predictions from multiple decision trees, they reduce the risk of overfitting and often provide more accurate predictions than individual decision trees. However, the creation of hundreds of trees contributes to the model's complexity and increases the demand for computational resources and memory. Additionally, the process of aggregating predictions from numerous decision trees can result in slower prediction times compared to some alternative algorithms.

Below is a table summarizing our findings:

Model	**k-NN**	**XGBoost**	**Random Forest**	**Log-Reg**
f1-macro metric	0.184	0.382	0.472	0.2878
accuracy metric	0.4175	0.587	0.7669	0.5619
Kaggle score	0.4174	0.966	**0.97318**	0.426