CentraleSupélec

Net4Geo : Geographical Area Classifier

AUTHORS

Marius Dragic
marius.dragic@student-cs.fr

May 4th 2024

Contents

1 Introduction

Identifying the variations in data in multiple images from different dates can help in the comprehension of global phenomena. Through the use of satellite imagery, this project relies on geographical attributes extracted from the images. The goal of the project is to categorize specific geographical regions into six different classes that are :

- Demolition
- Road
- Residential
- Commercial
- Industrial
- Mega Project

These classes corresponds to different types of local environments.

2 Feature Engineering

2.1 Data visualization

The main features we originally have are : the coordinates of the polygon that is to study, categorical values describing the state of this polygon at five different timesteps (dates), urban features of the neighbourhood and geographic features of the neighbourhood.

The first task is to load the data in python. This can be done with the geopandas librairy. Then, we will use pandas to manipulate the dataframes, which is a famous python librairy for data preprocessing.

We can instantly see that the dataset is not very balanced : there are only 151 Mega Projects for more than 140k Commercial or Residential type.

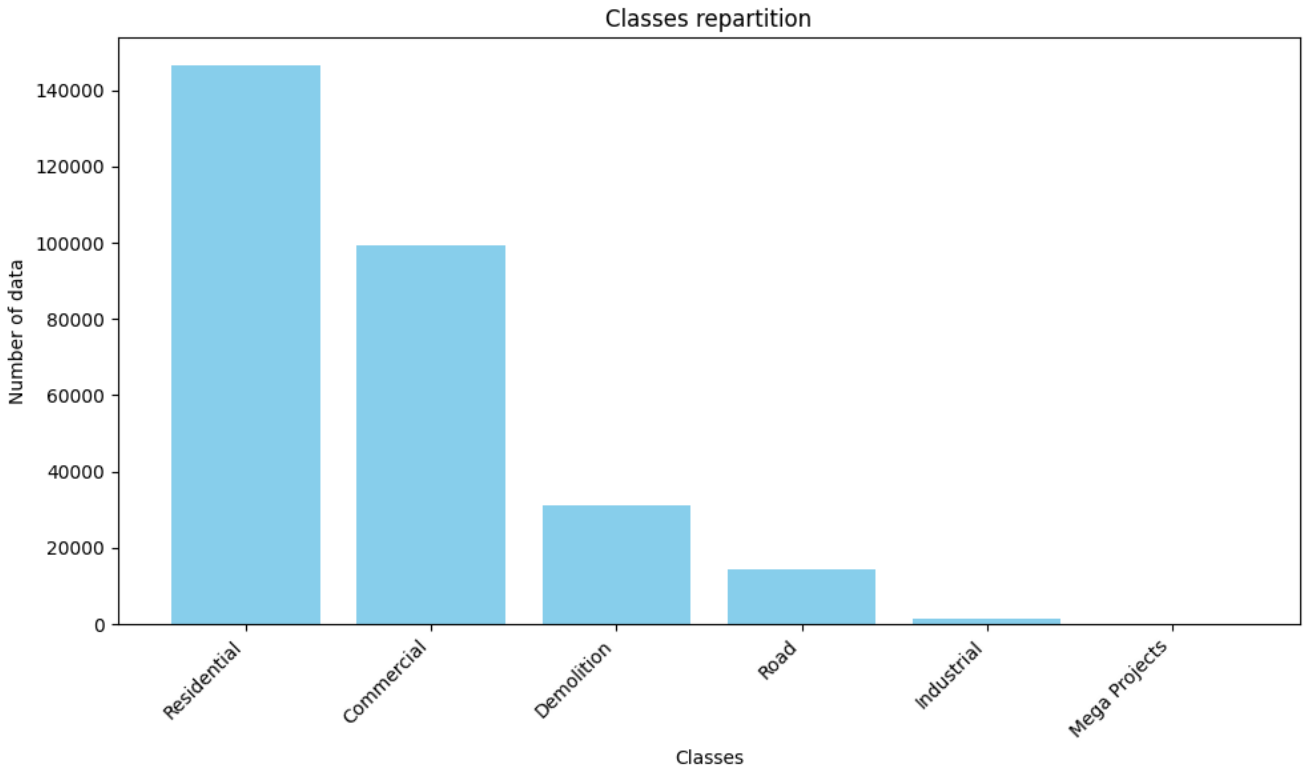

Figure 1: Graph showing classes repartition

We guess that Mega Project and Industrial will be difficult to classify due to the small amount of datas.

2.2 Data Preprocessing

2.2.1 Creation of features

In order to create an accurate prediction, we will try to create other features that our models will use.

The first features that we created are what we called 'diff dates'. It represents the difference between two dates of consecutive pictures that have been taken for one polygon. We calculate it simply with $df[\text{date}_i] - df[\text{date}_{i+1}]$.

Then, we implemented geographical metrics such as area and perimeter. These to are easy to compute and handled natively with methods like .area and .length on a polygon object. These features were very useful in the prediction since even with simple prediction models we got progress simply by adding these features.

For example, the mean area of a Mega Project is 1000 time larger than other classes. This will be used by models to help categorize these types.

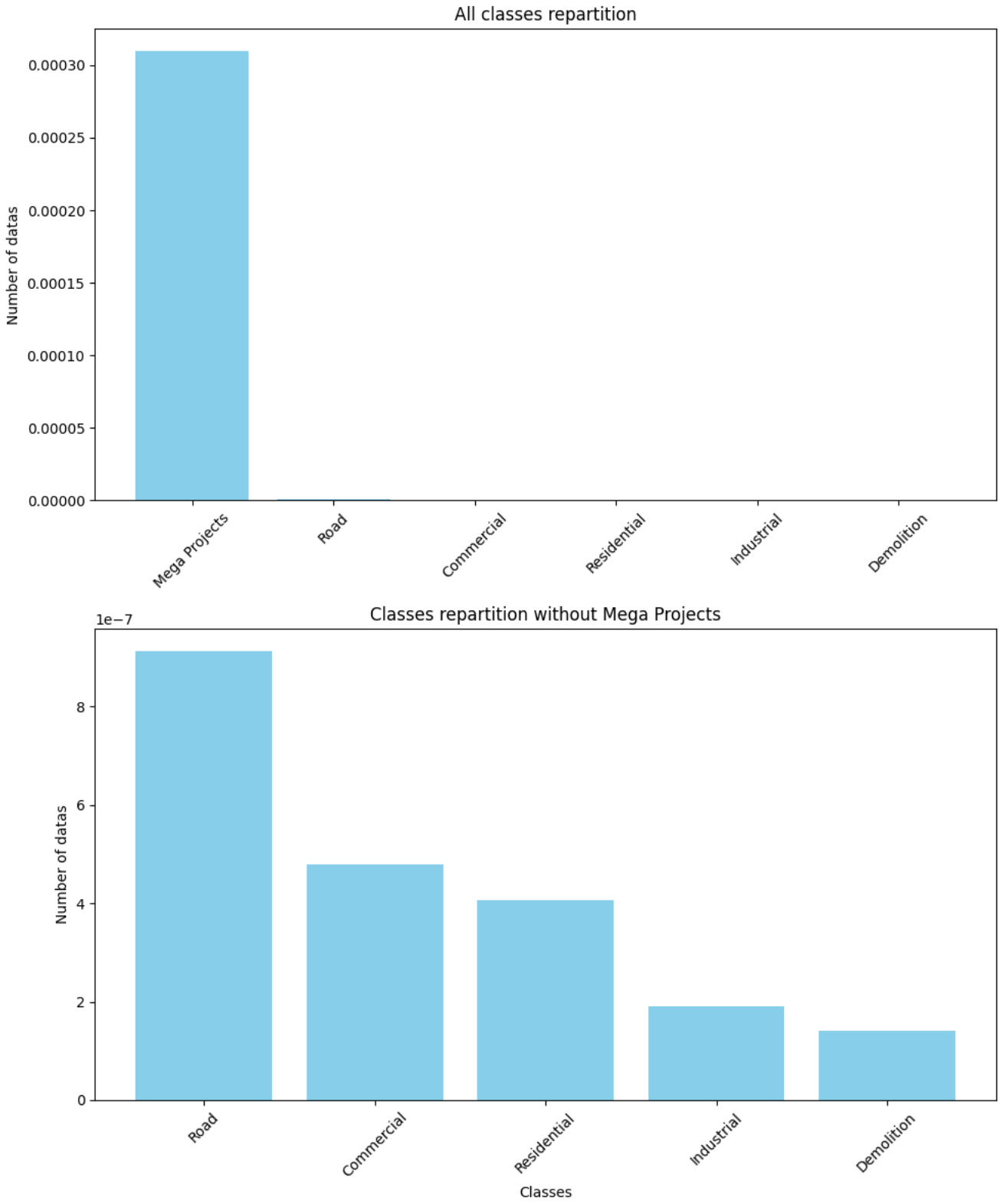

Figure 2: Graph showing classes area repartition

As suggest this graph the difference of area between "Mega projects" and other classes is so great that we could create a classifier that only classifies Mega Projects on area by finding a relevant threshold. We won't do this, however, to stay within the "Machine

Learning" objective and let our model learn on its own.

We also tried to use other features such as the squared area or two regularity features we found in the literature defined as

$$F_1 = \frac{\text{Polygon perimeter}}{\text{Perimeter of a circle with the same area}} \tag{1}$$

and

$$F_2 = \frac{\text{Polygon area}}{\text{Area of a circle with the same perimeter}} \tag{2}$$

But these three tests proved unsuccessful as they did not improve our performance. We therefore decided to withdraw these features for the rest of the project.

2.2.2 One Hot encoding

Features like **Urban type, change status date, geaography type** are very interesting data, but the problem is that they are categorical. To remedy this, there's **"onehot" encoding** which vectorizes each category of the feature under consideration. The main problem is that each category becomes a feature in itself. So we go from a 20 feature dataset to a N features, which considerably increases complexity and computation time, and could lead to dimensional curse.

2.2.3 Chi-Square statistical test

Finally, among all the remaining variables, the ones already present in the dataset and the ones we created we have to perform feature selection in order try to reduce dimension of our dataset. We selected the most useful using the **Chi square statistical test** χ^2. We tried different thresholds for the values of score and the p-values. We selected a **p-value of 0.05 and a score of 5**. They are used to filter out less significant features.

3 Model tuning and comparison

We performed different implementations of several machine learning algorithms in order to solve this problem.The metrics used to assess the quality of the model is the F1-Score. It is a number between 0 and 1 that represents the accuracy of the model. Our first ideas were to use **AdaBoost and a Neural Network** to make our predictions. Both of these methods did not perform well as we experienced performance not much better than the k-NN sample. Then, we implemented a **random forest** and got our first results above 0.5 in prediction precision. Actually, we even got up to a score of 0.68. At that point, we were not using all the feature engineering we discussed before but only 4 hand-chosen features which were the most relevant ones according to us : **urban type, geography type, area and perimeter**. By adding even more features, still hand-chosen, we went up to a score of 0.84. The features were the area, the perimeter and the temporal features. We were not using the color features at that time.

After that, we still tried to use other methods in order to find maybe something better. Our next try was to use **Fischer Discriminant Analysis** as well as **Principal Component Analysis** to both reduce dimensionality and maximize variance onto the space

where quantitatives datas are projected. The score were not much better than those of the baseline. In order to do that, we decided to try a Singular Vector Decomposition by taking only the first 13 components (which retained 85 percent of the energy) and noticed that we outperformed the baseline thanks to cross-validation with a score of 0.592 but when the code was tested on the not labelled data gave a score of 0.17 hence showing a significant difference from training data, it seems that the model is over-fitting. We first thought that we could add this SVD at the beginning of the random forest and that it could improve its performance but it proved to be wrong due to over-fitting.

Our last attempt was to still use the random forest but with preprocessing of the data. We performed a Principal Component Analysis again but this time, we put the selected features in the random forest. The score did not increase once again. Our last try was to use the χ^2 statistical test in order to select the interesting variables, which proved to be efficient. With this method combined to the random forest, we got up to a score of 0.96874.

4 Conclusion

The model we decided to keep at the end of this study is a χ^2 statistical test combined to a random forest. Before using these algorithms, we performed feature engineering to make some properties of the data stand out. We finally performed a cross validation with a GridSearch to find the best hyperparameters for our random forest. However, as the original train dataset is very large, we've created a reduce train dataset with the same class distribution to run the RandomForestClassifier faster several times in a row with GridSearch.

Listing 1: Grid Search cross validation implementation in Python to find best hyperparameters

```
from sklearn.model_selection import GridSearchCV

param_grid = {
    'n_estimators': [100, 200, 300, 400, 500, 600, 700, 800, 900,
       100],
    'random_state': [10, 20, 30, 40, 50, 60, 80, 90, 100]
}

random_forest = RandomForestClassifier()
grid_search = GridSearchCV(estimator=random_forest, param_grid=
   param_grid, cv=5, scoring='accuracy')
grid_search.fit(train_df_new, train_y)

print("Best hyperparameters :", grid_search.best_params_)
print("Best score Cross Validation :", grid_search.best_score_)
```

Finaly GridSearchCV method suggest the following hyperparameters to improve our Random Classifier : **n estimator = 700** and **random state = 50**.

These hyperparameters enable us to achieve a final performance of **0.86120** on the Kaggle test set.