Machine Learning

CentraleSupélec - 2A

Kaggle project

2022-02-06

CentraleSupélec

Maelan Bernard
Rémi Boutonnier
Guillaume Grasset-Gothon
Théo Guilbert
Kaggle team : krb5

Table des matières

I Feature engineering

I.1 Preprocessing on the already existing features

Some features of the data are not usable by a Machine Learning algorithm using their original types. Therefore, we needed to modify the types of several attributes. Indeed, strings of characters must be changed into numbers to be considered by our algorithms.

The attributes described as words : change status dates, urban type and geography types, have therefore been transformed into integers. For the first two attributes it was sufficient to assign an integer for each possible value of the attribute. However, for the geography types attribute, each instance of this attribute is composed of a series of values. We have therefore created a new attribute for each possible value. Thus, if an instance has a certain geography type, the value of the new attribute associated with this geography type will be 1, otherwise 0.

The last strings attributes are the dates. Since we cannot simply transform them into integers as for the other attributes, we have restructured the information in a different way.

First, we calculated for each instance of the data the years, months and time intervals between each of the dates. In this way we had transcribed all the information contained in the date attributes. However, we noticed that these attributes were not very relevant because they did not improve the results of our algorithms.

We have therefore decided not to take into account years, months and time intervals, but only the total time related to the completion of the construction project. This new attribute is defined as the time between the first status change of the construction and the last status change. This choice is motivated by the fact that the duration of the project is, in principle, more correlated to the nature of the project than its date. However, there was no notable improvement of the score linked to this feature.

Finally, we decided not to take into account the dates features as we didn't manage to extract any satisfying result using features linked to the dates.

We also chose to not consider the geographic features as it did not led to a real improvement in the performance of our algorithms.

I.2 Research of new features

To extract new features of the data, we mainly used the geometry by computing properties on the polygons :

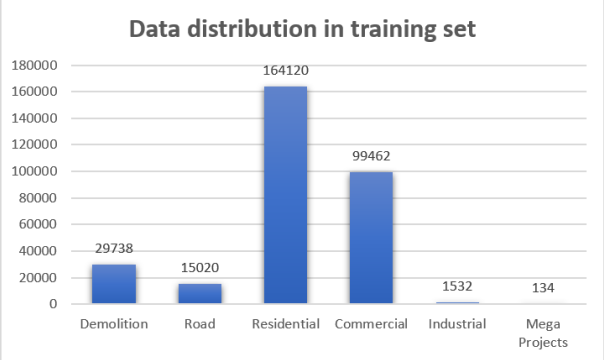
- The area as it was done in the example
- The perimeter
- The centroid coordinates of the the polygon
- The width and length of the smallest rectangle containing the polygon
- The angle of the diagonal and the base of the previous rectangle
- Angles between points of the geometry to try to catch information on the shape of it

However, wee did not keep the angles between points of the geometry because it was not improving the performance of our models.

Moreover, to catch the potential interactions between features, we created new features using the library PolynomialFeatures of Python. At the beginning, when we had too many features, it was not working well because we had too many features that were useless. However, when we reduced the number of features while using polynomial features, we had our best score. We used a degree of 2, as it was how we had the best results.

I.3 Dealing with unbalanced data

The distribution of the classes among the data set is very unbalanced, as you can see on this plot :



Therefore, we tried to take into account the imbalance of the data in the preprocessing of the data, using the imbalanced-learn library available in Python. We tried over-sampling with SMOTEENN and SMOTETomek, but it was not working well, we had a very high accuracy, but a medium score on the kaggle website. The model was overfitting to the data. Thus, we then tried to do under sampling with NearMiss, but we had difficulties using the libraries. We could only use it with logistic regression, and we had bad results.

II Model tuning and comparison

Looking for the best model to implement for this problem, we tried many models :

- Logistic regression
- SVM
- kNN (k-nearest neighbors)
- Neural network
- Simple decision tree
- Boosting with adaboost
- Bagging with random forest

When we tried different combination of features for all the models, in the end the best combination was always the same and we kept the features :

- change status date
- urban types
- the geometric features mentioned earlier (area, perimeter, etc. without the angles between points of the geometry)

This combination allowed us to perform well on training set and have the best generalization for the test set, avoiding overfitting. An advantage of considering less features is also to train the model faster. However, despite the reduction of the number of features, the computational cost of preprocessing the data and training a model was very high and took a lot of time. Therefore, we were limited to try every combination of features and models (with some models taking more than 2h to be trained). Using PCA with all or a part of the features (created or not) to perform dimensionality reduction could also have been a solution, but we did not get better results when we tried.

II.1 Logistic regression

We used a logistic regression to test how a simple model was doing on the problem.

We obtained a cross validation accuracy of 0.57789. The score obtained on kaggle with was 0.52469.

II.2 SVM

We wanted to try different types of kernel like linear, gaussian or polynomial. We use the One-vs-Rest strategy which splits a multi-class classification into one binary classification problem per class.

We obtained with SVM linear a cross validation accuracy of 0.52.

The score obtained on kaggle with SVM linear was 0.43310.

For the two other kernels we could not fit the model because it was too long (more than 3 hours at least).

II.3 kNN

We chose to use the classic number of neighbour at first to tune our model : $\sqrt{train\_size}$. This led to a 547-nearest-neighbour. We also tried to use other values for the parameter k but the more accurate value remained the square root.

We obtained a training accuracy of 0.67989 and a score on kaggle of 0.61581.

II.4 Neural network

To implement a neural network model we chose to use the Keras API. First of all, we began to use the entire features as presented in the first section, and we applied a PCA to choose only the relevant features.

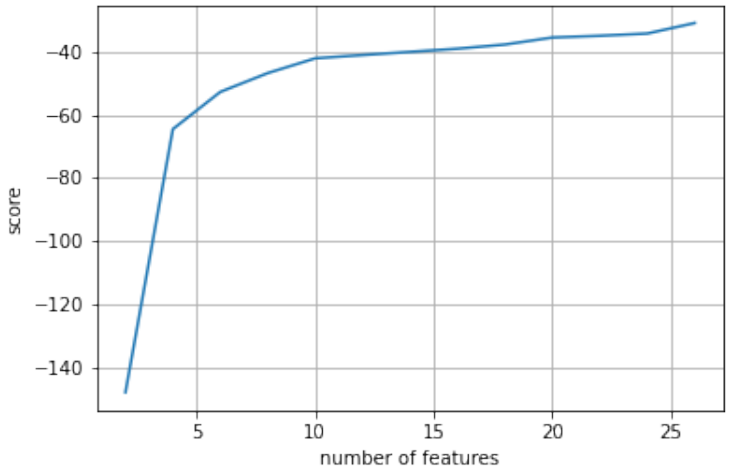


FIGURE 2.1 – PCA on our training set

Thus we chose to use the 10 first components to deal with our neural network.

Moreover to make a categorical prediction we reshaped our training set label as follows :

```
change_type_map = {'Demolition': np.array([1,0,0,0,0,0]),
    'Road': np.array([0,1,0,0,0,0]),
    'Residential': np.array([0,0,1,0,0,0]),
    'Commercial': np.array([0,0,0,1,0,0]),
    'Industrial': np.array([0,0,0,0,1,0]),
    'Mega Projects': np.array([0,0,0,0,0,1])}

train_y = train_df['change_type'].apply(lambda x: change_type_map[x])
```

Indeed, this representation of label enabled us to use the softmax activation function on our last layer of dimension 6 to compute the predicted probability for a sample to be in the i^{th} class.

At the beginning we tried a lot of possible combination between the number of layers and neurons with respect to a categorical cross-entropy loss and a gradient descent optimizer (SGD). Note that we only used dense and dropout layers from Keras to tune our model. We found that a good architecture was to decrease the number of neurons according to the depth of the layer.

The different results showed that the best performance was obtained with only 2 hidden layers composed by 40 to 60 neurons in the first one and 20 to 30 in the second one. Indeed, deep neural network was not performing very well, maybe because it was too hard to train, due to a large amount of parameters. We also found that the activation function that performed the best was the *relu* one.

Finally, we tried several optimizer such as *Adamax* and we found that the one which gave the best results was the *Adadelta* optimizer. Then, to prevent overfitting we adjusted the rate of the dropout layer (0.2 to 0.4) and added some regularizers on our layers thanks to the *keras.regularizers* module.

In the end, we used the selection of features used for all our models described at the beginning of the section. With these features, we had the best result we could get with neural network :

- 0.67556 accuracy on the full training set
- ≈ 0.66 validation accuracy when we were fitting our model
- 0.64205 score on kaggle website

II.5 Simple decision tree

We choose a max depth for the tree of 15 which is the depth maximizing the performance using our features.

We obtained a cross validation accuracy of 0.73. The score obtained on kaggle using simple decision tree was 0.64.

II.6 Boosting with adaboost and gradient boosting

— With adaboost and a simple decision tree as base classifier we obtained :
  - Training accuracy : 0.65084
  - Score on kaggle website : 0.60433

— With adaboost and random forest as base classifier we obtained :
  - Training accuracy : 0.97982
  - Score on kaggle website : 0.67626

  It is a better score than with a simple decision tree as base classifier, but it was much longer to train (almost 3h). Moreover, the accuracy is very high so it seems the model over-fitted on the training data set. We used the adaboost with 50 estimators, but we did not have the time to how it was doing with more or less estimators because it was too long to train for each iteration.

— With gradient boosting classifier, we obtained :
  - Training accuracy : 0.90640
  - Score on kaggle website : 0.65851

II.7 Bagging with random forest

This algorithm led to the best scores we have ever obtained. The max depth of our trees was set to 15, for the same reasons as for the simple decision tree.

After testing what were the best parameters and features, we chose to train the model on the whole data set (not to split it) to get the best result possible. This could lead to overfitting, but as we previously selected the maximum depth of the trees using cross validation, overfitting should be avoided.

With random forest classifier, we obtained :

- Training accuracy on the whole training set : 0.81556
- Score on kaggle website : 0.68970

To conclude, to obtain these results we used polynomial features of degree 2 based the combination of features we mentioned at the beginning of this section.