Università Ca'Foscari Venezia

Home Credit Default Risk Prediction Project

Pierfrancesco Montello n° 887309, Riccardo Pasti n° 868552, Giovanni Rossato n° 860282

Master's Degree in Data Analytics for Business and Society
DATA ANALYTICS AND ARTIFICIAL INTELLIGENCE

March 2021

Data Import, Assessment and Preparation

The purpose of the project we were assigned is to deploy some methods/models able to make predictions, the more accurate possible, over the task described in https://www.kaggle.com/c/home-credit-default-risk/ page: we should try to predict, from some data we were given, if a person would be able to fully repay a loan in the terms or would encounter some difficulties (coded respectively as "0 and "1" in the column "TARGET" of the dataset we used).

Initially, we start by importing in our Jupyter Notebook file, some libraries and classes that wil come in handy during the whole course of the work. We import basic libraries like: pandas, numpy, scikit-learn (and some of its interesting sub-classes) for manipulating the data and building models, and matplot and seaborn for plotting purposes.

Since, we want to understand the nature of the data before starting with the manipulation, we upload the file named "HomeCredit_columns_description.csv" (specifying the type of encoding, considering that even if it is a .csv file, it's encoded differently from the usual files and the normal "pd.read_csv" shouldn't work without adding the right parameter value), and pair it with the files "application_train.csv" and "application_test.csv", the first will mainly be used to feed models or classifiers as training and the second is the one on which the predictions will be made and then uploaded to Kaggle for an evaluation.

We don't want to add any other information (from the other files downloadable from the contest page) since the already present in "application_train.csv" and "application_test.csv" are really big and we think that adding further columns to the data will only add more noise and computational cost the whole work, not providing with any benefits.

After that we proceed adding, from the start, some functions that will be useful in the following parts of the work.

It's clear from the start that, even an unexperienced analyst, would notice the large presence of missing values (coded as "NaN") in our data and that should be treated properly. Using one of the previous mentioned "useful functions" we manage to get a table with names, quantity and percent on the total of the columns containing missing values.

We make the choice to drop completely the columns that have 60% or more of them, made by "NaNs", assuming that with only 40% or less of information from that feature any type of use would be at least debatable if not totally misleading.

For the rest of the columns with missing values we adopt 2 different strategies, based upon the type of data contained: if the columns has numeric values the missing values will be replaced with the median of the values of the column whereas, for the columns that contains data defined as "object", each columns will find its "NaNs" replaced with the string "missing value", de facto creating a new category in each of them, assuming then that the fact of not having the data for different observations would be determined by some underlying common characteristic.

Later, looking closely to the data, we consider reasonably to add to the dataframes (both the train and the test) some features (4, indeed) that are obtained from computation over the already present ones but more consistently relatable to the target variable, also for interpretability purposes. However, despite there are understandable reasons to add the aforementioned features, we should be sure that they bring an augmented prediction capability to the model otherwise they would be useless for our purpose. To address this issue, we copy our main dataframe and in the duplicate we add the new 4 features so that there could be some way to compare the classifiers accuracy with and without them.

Nonetheless, before dive into this type of comparison there is still something that has to be done. In fact, right before the comparison we go on with the Label Enconding (turning every feature having only 2 different values in a dummy containing only 1s and 0s) and the One-Hot Encoding (for every feature having more than 2 values, adding columns accordingly to code correctly every category present, even the previously created "missing values"), exploiting some of the aforementioned useful functions put in the second part of the Notebook with this purpose.

After that, we remove from the training datasets (that we got from "application_train.csv") the features reporting categories or the features not present in the test datasets (got from "application_test.csv") except for the "TARGET" feature which is stored separately due to its obvious utility.

Finally, with all the data properly treated, we can proceed with the comparison between the dataset with the 4 "added features" and the one without them. The methodology of comparison we decide to use a 5-fold Cross-Validation. As classifier to compare in the CV, we take in consideration also several methods (decision tree, random forest, LightGBM option) but we end up with the logistic regression (the first model we will use for the effective classification) evaluating the two datasets through 2 different values for the parameter C (the one that addresses the desired "inverse of regularization strength).

With both the values for C, the dataset with 4 more features has a higher prediction accuracy in the validation test than its counterpart so, from now on, the dataset on which we will work is the one with the added features.

Logistic Regression Classification

Now we move onto the main part of classification task. Before trying to fit any model we must also scale the variables in our dataset, so that there's no excessive bias in the classification due to the dimension of some of them compared with the smaller ones. As mentioned before, the first classification method is conducted by fitting a Logistic Regression Method. The first logistic regression we computed is done by the common "LogisticRegression" method provided by class "linear_model" of scikit-learn.

However, we must also include in the code a way to balance the classes since in the training set there's a disproportion among the two target classes (we do by adding the parameter "class_weight") and, testing with different values of the "C" parameter the best method we got, (with "C" = 0.0001) has an accuracy of 0.70394, which for being the first attempt in classification is not bad.

Nevertheless, checking the documentation we discover that the default option for the regularization is a l2-regularization (a Ridge) for avoiding the possibility of over-fitting. So, we fit the logistic model once more, this time changing the type of penalty to a l1-regularization (a LASSO) and, with the "C" parameter set equal to 0.1, we get an even better model with the accuracy of 0.73729. Lastly, we want to test a logistic regression model without any type of regularization, which surprisingly is even slightly better with a 0.74001 accuracy score, being, until now, the most accurate classifier we built. All the predictions made by the different models on the test set were uploaded on Kaggle as "Late submissions" to get a score of accuracy.

If we effectively had taken part to the competition, with only 6 submission, we would have ranked 5507th.

Decision Tree Classification

Next, we want to build a different classifier for this issue, a Decision Tree. Anyhow, running a Decision Tree only with the default setting doesn't seem a great idea so, before the fitting we run a 3-fold Cross-Validation changing the depth of the tree (we initially try with 5, 7 and 10, then with 6,7 and 8) and we get that the highest accuracy in the validation test comes from a tree with a depth of 6. Therefore, we feed the tree with the training data and we classify the test data.

To our major surprise, submitting the classification results in Kaggle, we obtain an accuracy of just 0.50139. A value like that is slightly better than a coin-toss classification, that has 0.5 value accuracy, the baseline value which must be surpassed by the most possible to have a good classifier. It's almost impossible to have a classifier with the accuracy of 1.0 but a score that low is not good at all.

With these considerations in mind we try to change other parameters to the tree, to check if it's possible to get a better classifier score. We try to double some default parameters value, we bring the minimum sample needed to perform a split to 4 and the minimum sample needed to have a leaf to 2. This time the performance of our classifier is better, 0.53331, but still not satisfying.

Lastly, we build a Decision Tree without any constraint regarding the parameters, with the suspect that we may end up with over-fitting issues. Surprisingly enough, this last tree is the best performing among the three we used, with an accuracy of 0.53638 on the test set.

Until now, we followed two different approaches for solving the issue yet the best tree for accuracy is still more than 9% worse than our worst logistic model; this may lead us to think that the decision tree are not the most effective method to address this problem or, at least, to get a satisfying accuracy, there must be applied a very in-depth analysis and choice of the parameters and, possibly, the use of bagging, boosting or random forest (which will help us, too, shortly) could be helpful.

K-Means Clustering Classification

Finally, we want to face the task from a third perspective, with a k-means clustering classification. However, to produce at least purposeful k-means clustering with must define the dimension in what measure the distances among the different instances.

To fulfill this requirement, we decide to use, once again, a 3-fold Cross-Validation but, in this case, the model involved will be a Random Forest which will be tested with every different feature we have in our dataset to determine the two most important. After the execution of the CV, we plotted the first 15 features, importance-wise, at it turns out that "EXT_SOURCE_3" and "EXT_SOURCE_2" are two most important having a score almost at 0.16 each. Unfortunately, there's not much we can say about them in terms of interpretation since the description let us know that those are normalized score from external data source and nothing more.

Nonetheless, the CV served its purpose and now we are able to build a k-means classifier using the two most important features as its main dimensions. Trained, as usual, with the training set and tested on the test set, the classifier obtains an accuracy of 0.58052, which is impressive considering that it uses only two features and is better than any of the decision trees we built before.

Final Thoughts

Despite, having trained several models/classifiers to assess the issue the result, considering our best result, we are at 6% lower than the best method obtained and, virtually, out of the best 5000 who submitted at least a try.

Anyway, this work brought some insights along: the most effective model, with relatively low effort, that you can pull off is Logistic Model, in our case it was a pure logistic model with no regularization, but it is possible that with specific values of the hyperparameter, the reduction in variance can

overcome massively the increase in bias, that a Ridge and/or a Lasso can perform better. And we didn't even mention the fact that the two can be combined in an Elastic Net, perhaps, even better.

We were also able to experiment that one or both of the following statements are true: maybe the Decision Trees alone are not the most suited classifier for this task and/or to get a good results requires a lot of cautious "engineering" in their development. Obviously, the implementation of Bagging, Boosting or putting together a Random Forest should bring better results, at least in the long run.

As long as Clustering is concerned, our only attempt was with one type of clustering, maybe there are other types of clustering strategies or structures that can perform better, not even mentioning the fact that the SVC could be useful, too. It's anyway remarkable that the k-means clustering, despite using only two features, perform better than expected, meaning that it could greatly benefit from an increase in information available in form of more dimensions (even if, we would lose the representation opportunity).

In conclusion, we may hypothesize that the best performers in the competition may have made different decisions from the start on the data or, perhaps, used more advanced tools like Neural Networks.