Models with priori restrictions on the relationship between factors and the predicted value
Technical Report (Term 4: Early Research project)

Anna Nikolaeva [1]

Abstract

Advance machine learning models like DNNs and random forests have been made a huge breakthrough in the area of predictive analytics. Their performance is explained by high flexibility and the ability to catch complex relationships among data. One of the main challenges is that they in the majority of cases can not distinguish semantically meaningful inputs that obey known relationships. Recently the TensorFlow Lattice was presented as a set of prebuilt TensorFlow Estimators that that can handle the problem by allowing to build own lattice models. In other words, using the instrument, we now have the opportunity to make interpreted models with constraints. This property is especially important in the problem of predicting a client's default, since current models are not reliable for using them in practice due to a luck of explainability. Overall, this report contains the description of the first step of the project - implementation of the pipeline for the creation of entities to generate additional features. The so-called Deep Feature Synthesis approach is also presented in the document.

1. Introduction

We are working with the well-known Home Credit Default Risk dataset[1] from the competition that ended up in 2018. In addition to the interpretability of the model output, performance is no less important to us. Therefore, we are going to do a good-quality data prepossessing with the generation of additional features. To achieve this goal, we will use the Deep Feature Synthesis technique for generating new features from the featuretools library[2]. The approach gives us the opportunity to generate many features from existing ones without the direct participation of data engineers. The writing of requirement mathematical function is only needed. Moreover, with this help, we can create entities and relationships for generating features from secondary order tables (Figure 1) with the depths up to 3 for the data. Thus, the process of generating new features can be brought to automatism by applying one-factor evaluation for model. In our case, the quality metric will be a ROC AUC with a low threshold.

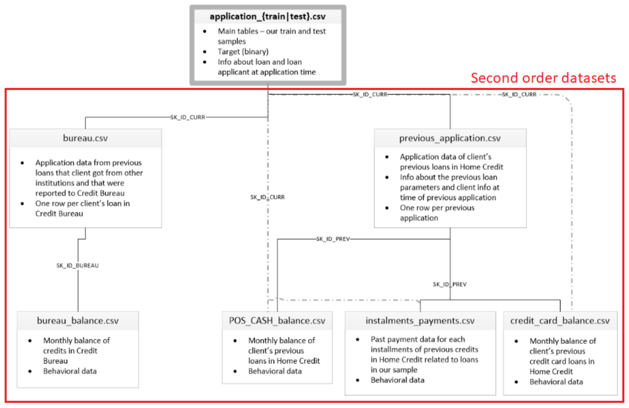

application_{train|test}.csv
Main tables – our train and test samples
Target (binary)
Info about loan and loan applicant at application time
Second order datasets
SK_ID_CURR
SK_ID_CURR
SK_ID_CURR
SK_ID_CURR
bureau.csv
Application data from previous loans that client got from other institutions and that were reported to Credit Bureau
One row per client's loan in Credit Bureau
previous_application.csv
Application data of client's previous loans in Home Credit
Info about the previous loan parameters and client info at time of previous application
One row per previous application
SK_ID_PREV
SK_ID_BUREAU
SK_ID_PREV
bureau_balance.csv
Monthly balance of credits in Credit Bureau
Behavioral data
POS_CASH_balance.csv
Monthly balance of client's previous loans in Home Credit
Behavioral data
instalments_payments.csv
Past payment data for each installments of previous credits in Home Credit related to loans in our sample
Behavioral data
credit_card_balance.csv
Monthly balance of client's previous credit card loans in Home Credit
Behavioral data

Figure 1. Diagram of the Home Credit Default Risk dataset

The main contributions of this report are as follows:

- Analysis of the data provided.
- Feature generation using featuretools library.
- Analysis of the impact of each feature on tree-based model output.

2. Deep Feature Synthesis

Deep Feature Synthesis (DFS) is an automated method for performing feature engineering on relational and temporal

[1]Skolkovo Institute of Science and Technology, Moscow, Russia. Correspondence to: Anna Nikolaeva <anna.nikolaeva@skoltech.ru>.

Technical report, Early Research project, Skoltech, Moscow, Russian Federation, 2020.

[1]https://www.kaggle.com/c/home-credit-default-risk

[2]https://www.featuretools.com/

data. DFS requires structured datasets, as we have (Figure 1), in order to perform feature engineering. The core idea and algorithm itself can be found in the original article (James Max Kanter, 2015). Overall, by implementing an autotuning process, it is possible to optimize the whole pathway without human involvement, enabling it to generalize to different datasets. There was also shown that the system is competitive with human solutions.

3. Experiments and Results

All source code is available in the folder[3]

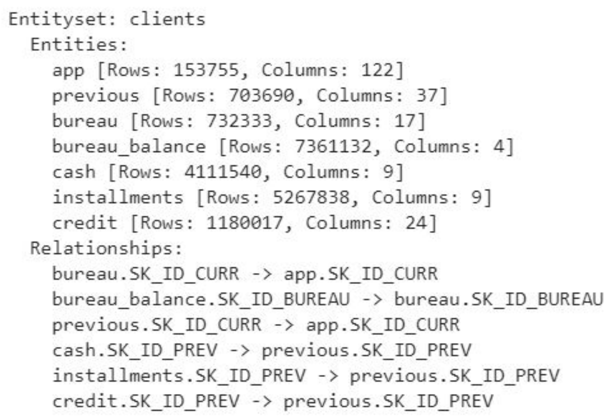
3.1. Analysis of the data provided

The application training data has 307511 observations (each one a separate loan) and 121 features (variables) in total. The target is highly imbalanced (282 686 versus 24 825). The majority of features has more than 60% of lost data, that are filled using imputation. Besides, I also used one-hot encoding, value counts techniques, and generation of percentage ratios. XGBClassifier with default parameters performed with the ROC AUC of 0.7 what was quite good result.

3.2. Feature generation using featuretools library

The major problems of the approach is that for the dataset it require more than 12 Gb of RAM to process train part, and takes several hours to generate hundreds of features. So, I continued only with 70% of train data and compressed them by changing the type, where possible. Basically, library allows generating “transformations”, that acting on a single table by creating new features out of one or more existing columns, and ”aggregations”, that performing across tables and use a one-to-many relationship (parent-to-child) to group observations and then calculate the statistics. In order to use first option I added time steps from the features provided.
All features should have appropriate types to be distinguished by the library in order to apply generations to the right variables. The minor changes that I have done was with changing categorical type on ordering for features that have this meaning (e.g. educational level, region rating, ect.).
After doing all above, we could generate entity set with relationship given by indexation (continuous lines on Figure 1, resulting entity on Figure 2).

[3] https://drive.google.com/drive/folders/1uUgRjlcts6x4y-jiIfmZs5g4h0X9fzON?usp=sharing



```
Entityset: clients
  Entities:
    app [Rows: 153755, Columns: 122]
    previous [Rows: 703690, Columns: 37]
    bureau [Rows: 732333, Columns: 17]
    bureau_balance [Rows: 7361132, Columns: 4]
    cash [Rows: 4111540, Columns: 9]
    installments [Rows: 5267838, Columns: 9]
    credit [Rows: 1180017, Columns: 24]
  Relationships:
    bureau.SK_ID_CURR -> app.SK_ID_CURR
    bureau_balance.SK_ID_BUREAU -> bureau.SK_ID_BUREAU
    previous.SK_ID_CURR -> app.SK_ID_CURR
    cash.SK_ID_PREV -> previous.SK_ID_PREV
    installments.SK_ID_PREV -> previous.SK_ID_PREV
    credit.SK_ID_PREV -> previous.SK_ID_PREV
```

Figure 2. The final entity set

3.3. Analysis of the impact of each feature on tree-based model output

For the first time, I used “count”, “percent true”, “min”, and “time since previous” basic primitives, an some custom functions, that are describe it details in the corresponding ipynb. Out of 1136 possible variants I randomly selected only 30 of them to check their influence on the performance. In Figure 4 and 3 we can see the outcomes. So, we can see the increase in performance by 6% after including the generated features. Moreover, the contribution of the new features can be seen on SHAP values plot.

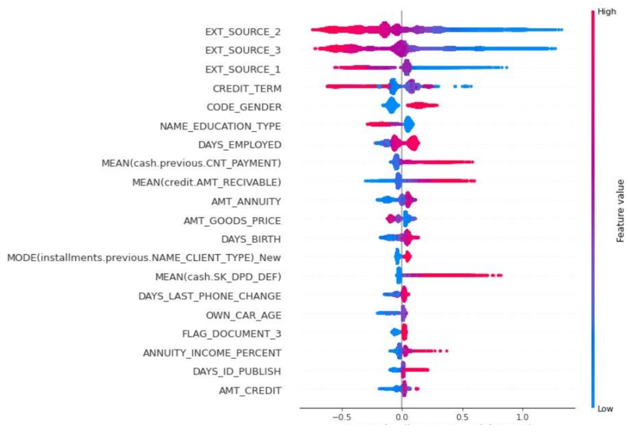


Figure 3. SHAP values (impact on model output)

4. Conclusion

Thus, the whole pipeline to generate new features and test their importance in the final model has been built. The next step is to make the pipeline work in an automatic mode by

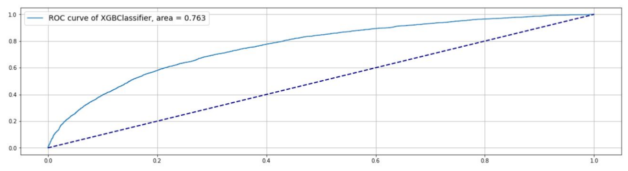


Figure 4. ROC AUC curve

selecting features according to the score obtained on every feature separately.

References

James Max Kanter, K. V. Generating features for target entity. 2015. URL https://perso.telecom-paristech.fr/bloch/papers/proceedings/CAP2017-Hadrien.pdf.