Prediction of credit clients repayment abilities with machine learning

Fabio Ferreira, Mario Baydar
Faculty Electrical Engineering and Information Technology
University of Applied Sciences Ravensburg-Weingarten
Weingarten, Germany,
{fabio.dacostaferreira, mario.baydar}@hs-weingarten.de

January 27, 2019

***Abstract*—We describe and analyze the approach used to attend the "Home Credit Default Risk" challenge on Kaggle. The goal of the challenge is to predict the ability for a client to repay a loan. The provided data contains personal and wealth information together with information about previous loans, credit card balances and ratings from other agencies. We will show different methods for handling missing values and selection of features. For training the classifiers we used two different approaches. In the first approach we used LightGBM and XGBoost with hyper-parameter optimization. For the second approach we used stacking, a ensemble technique that combines the predictions of multiple learning algorithms. At the end we compared both approaches and the obtained results are presented.**

***Index Terms*—kaggle, homecredit, machine learning, stacking**

I. Introduction

The "Home Credit Default Risk" is a machine learning challenge hosted by Kaggle [1]. The challenge took place from May 17, to August 29 of 2018 in partnership with Home Credit, a institution specialized to provide credit lines to people with non existing or insufficient banking history. With over 8000 participants and 7000 teams it was by far the biggest competition ever hosted by Kaggle. The dataset contains personal information about the applicant, like civil status, professional and wealth information but also information about previous applications, data from other financial institutions, monthly balance from previous credits and credit card balances. The target was to predict whether or not a applicant is able to repay a loan. The used classification metric for the challenge was the area under the curve also known as "ROC-AUC".

II. Exploratory Data Analysis

In this section we describe the competition dataset and the data analysis.

In the entity-relationship diagram shown in Fig. 1 we see that the data is divided into seven different sources. The main file "application_{train—test}.csv" consists of information about each loan for a applicant. Every applicant is identified by a unique key. The test dataset has 48744 rows with 120 features each and the train dataset has 307511 entries with an additional column TARGET, that describes if a applicant had difficulties or not repaying the loan. The other sources like for example "installments_payments.csv" and "bureau.csv" hold more detailed information about every repayment and previous credits of the applicant. The file "installments_payments.csv" has over 13,6 million samples. Even though we knew that for a excellent classification result, all provided data should be used, we decided due to hardware resource limitations to just use the data from the main file "application_train.csv".

The main file contains personal information about the applicant. Table I shows some of the 121 features together with their description. From the 120 features there are 76 continuous numerical, 8 categorical and 36 binary values. The rate of missing values was considerably high in the provided data. There were 57 features with missing values in which 41 features had a missing value rate over 50% (see Fig.2). This can be justified by the fact that around 50 features are related to applicant real estate. This implies that for customers without own real estate (94199 samples) this features must be missing.

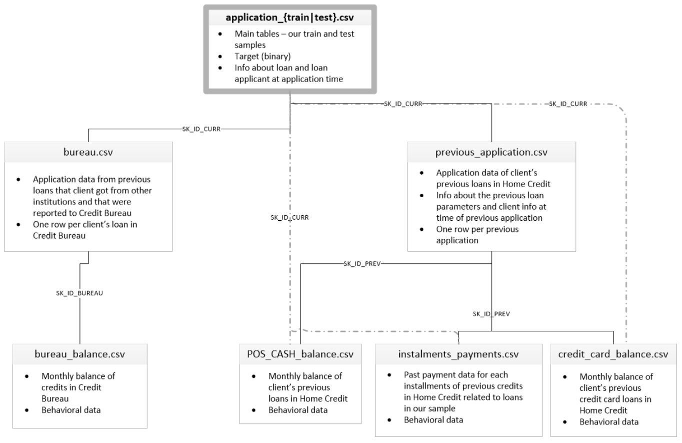
application_{train|test}.csv
Main tables – our train and test samples
Target (binary)
Info about loan and loan applicant at application time
SK_ID_CURR
SK_ID_CURR
SK_ID_CURR
SK_ID_CURR
bureau.csv
Application data from previous loans that client got from other institutions and that were reported to Credit 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_BUREAU
SK_ID_PREV
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

Fig. 1. Structure of the provided dataset [1]

III. Data Preprocessing

This section describes the preparation of the data, that is: handling of missing values, feature selection and feature

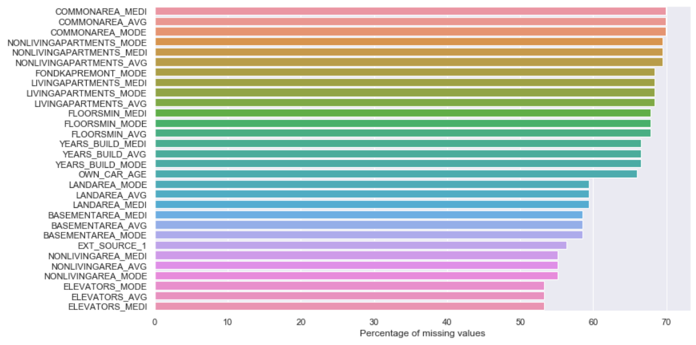

Fig. 2. Features with over 50% of missing values

engineering.

A. Data Cleaning

Date features, e.g. DAYS_BIRTH or DAYS_EMPLOYED (see Table I) were encoded as negative numbers representing the days relative to application time. For some clients the feature DAYS_EMPLOYED makes no sense because they are pensioner or unemployed. In those cases the high positive value 365243 was used. The days with negative sign were converted to positive years by dividing with -365 and rounding down the values. The values 365243 were changed to a minus one. For example -9461 will be converted to 25 and 365243 to -1. This conversion does not only bring a better understanding of the distribution (see Fig.3) but also turned the values from being stored as floating point to integer values which means for most algorithms the calculations will be faster.

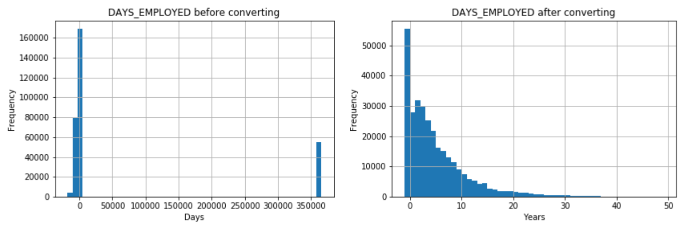

Fig. 3. Distribution comparison before and after conversion

Since a lot of classification algorithms implemented in common libraries are not able to learn from data with missing values, we handled them manually. Here different approaches for different features were used. For categorical features with high cardinality one hot encoding was used. This is needed because not only are a lot of algorithms prone to missing values but they are also not able to handle categorical features. To encode if the original feature was missing or not, a additionally binary feature is added. One of the features with high cardinality is OCCUPATION_TYPE (see Table I) which has 18 unique values (e.g. Laborers with 5838, Sales staff with 3092 or Medicine staff with 572 occurrences). Furthermore it has 96389 missing values, and after applying one hot encoding 19 columns are added to the data and the original feature is removed.

TABLE I

EXAMPLE FEATURES FROM APPLICATION_TRAIN.CSV

Feature name	**Description**
NAME_CONTRACT_TYPE	Type of the credit (loan cash or revolving)
CODE_GENDER	Gender of the Client
FLAG_OWN_CAR	Flag if the client owns a car
FLAG_OWN_REALTY	Flag if the client owns a house or flat
CNT_CHILDREN	Number of children the client has
CNT_FAM_MEMBER	How many family members the client has
AMT_INCOME_TOTAL	Income of the client
AMT_CREDIT	Credit amount of the loan
AMT_ANNUITY	Loan annuity
AMT_GOODS_PRICE	The price of the goods for which the loan is given
DAYS_BIRTH	Client's age in days (relative to application time)
DAYS_EMPLOYED	Days since client started current employment
OCCUPATION_TYPE	What kind of occupation does the client have
NAME_TYPE_SUITE	Who was with the client when applying for the loan
NAME_FAMILY_STATUS	Family status of the client (Married, Single, ...)

This is one way to handle missing values. Another way is to fill them with data that makes sense the most. The difficulty here is to not add information that is not really available and to figure out what makes sense the most. To fill the missing values for the feature AMT_GOODS_PRICE (see Table I) we analyzed other features and how they correlate with AMT_GOODS_PRICE. By that it could be observed that only 1% of the entries in AMT_GOODS_PRICE excluding the missing ones, differ from the entries in AMT_CREDIT (see Table I). Most of the time a loan is taken, the credit amount is as high as the good that will be bought. This is the reason why we decided to set the missing values equal to the AMT_CREDIT. A different approach which needs more data analyzing, is to find the nearest neighbor of a data point in case of features that are related the most to the one with the missing value. The feature NAME_TYPE_SUITE (see Table I) describes who was accompanying the client when applying for the loan. So the most related features are CNT_CHILDREN, NAME_FAMILY_STATUS, DAYS_BIRTH, CNT_FAM_MEMBER and FLAG_OWN_CAR (see Table I). All of this features could imply who was accompanying the client and were used to find a nearest neighbor. Additionally some thresholds were set so the possibility to find a neighbor would increase (e.g. a 5% difference for DAYS_BIRTH was allowed). This process of filling missing values was applied to several features. When more than one neighbor was found either the thresholds were lowered or the values were averaged. Sometimes even with a relative loose threshold no neighbor was found. In those cases the record with the missing value was handled as outlier and was removed from the dataset. The feature AMT_INCOME_TOTAL also debunked an outlier. The distribution reveals the anomaly because of the wide area with no occurrences similar to DAYS_EMPLOYED (see Fig.3 left). An income of 117000000 seems to be too high especially because the client did not own a car and according to the target had even difficulties to pay back the loan. Although the currency is unknown the big gap in the distribution still makes no sense. Because of this we decided to remove the record. Given a value that does not fit the normal data, it is important to ensure that it really makes sense to remove it by looking at different features of the record. A counterexample could be found for the feature AMT_ANNUITY (see Table I) where the highest value seems to be similar to the outlier for AMT_INCOME_TOTAL. But after analyzing additional features we could observe that the income of this client is also relative high so the value seems to be plausible. Supplementary other features and the target support this plausibility of the high loan annuity.

B. Feature Selection

After preprocessing we ended up with 218 features. As already mentioned, around 50 features were related to applicant real estate. Most of this features had a missing value rate over 50%, and therefore we decided to not use this for training. However since the possession of a real estate in general has a positive effect on a credit application, we kept only information about the type (e.g. Apartment, House, etc..) and number of real estates the applicant has. For simplicity we also decided to remove features related to the application time, hour and day of week, since there's no logical reason for this features to be relevant for the target predictions. After removing this features the dataset consists of 161 features.

C. Feature Engineering

Feature engineering normally requires domain knowledge to be successful. None of the authors can provide domain expertise in the field of loans. Fortunately a lot of competitors shared their thoughts about feature engineering on the competition forum which allowed us to incorporate some ideas into our own work. The goal of feature engineering is to combine existing features to create new ones that better represents the data and helps learning algorithms to achieve better results. A total of six engineered features were used as shown in Table II.

Most of the engineered features are ratios between two existing features. By using the ratio we gain the information where the single values are different but the ratio is equal or vice versa. By this we group applicants together in terms of dimensionality with respect to the engineered feature. The correlation between the new features and *TARGET* is shown in Figure 4. Specially to emphasize are the *EXT_SOURCE_X* features. This are positive values between 0 and 1 and they represent a score from a external rating agency. The meaning and how the value is calculated is not public. However the plot in Figure 5 shows the features with high correlation to *EXT_SOURCE_1* and gives some insights on how the value is composed.

IV. Classification

In this section the addressed approach for classification is described. We will present two methods for classification. In the first approach we used single machine learning algorithms with hyper-parameter optimization. In the second approach we used a technique called stacking, that combines multiple less optimized algorithms.

TABLE II
Engineered Features

Feature name	Formula
DAYS_EMPLOYED_PERC	DAYS_EMPLOYED / DAYS_BIRTH
INCOME_CREDIT_PERC	AMT_INCOME_TOTAL / AMT_CREDIT
INCOME_PER_PERSON	AMT_INCOME_TOTAL / CNT_FAM_MEMBERS
ANNUITY_INCOME_PERC	AMT_ANNUITY / AMT_INCOME_TOTAL
PAYMENT_RATE	AMT_ANNUITY / AMT_CREDIT
EXT_SOURCES_MEAN	(EXT_SOURCE_1 + ... + EXT_SOURCE_3) / 3
EXT_SOURCES_PRODUCT	(EXT_SOURCE_1 * ... * EXT_SOURCE_3)

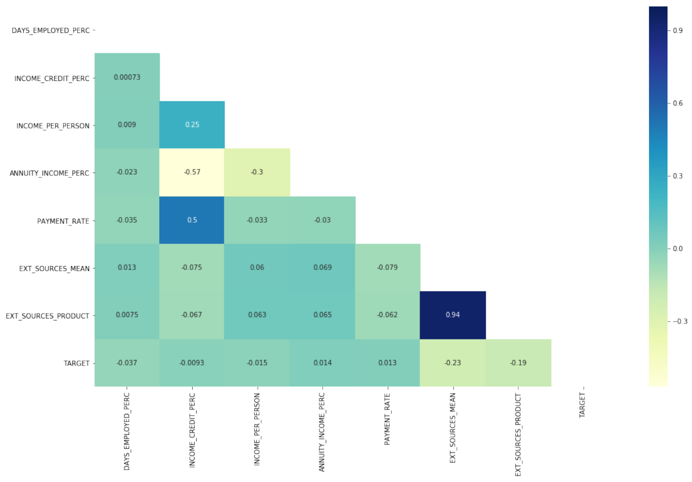

Fig. 4. Correlation between engineered features and *TARGET*

A. Scaling

Before training the algorithms the preprocessed data was scaled. Thereby we transformed each feature of the train data to have zero mean and unit variance, by subtracting the mean and dividing by the standard deviation of the feature (see equation 1). Those statistical values are stored to also be applied on the test data.

$$\frac{x_i - mean(\vec{y})}{stdev(\vec{x})} \tag{1}$$

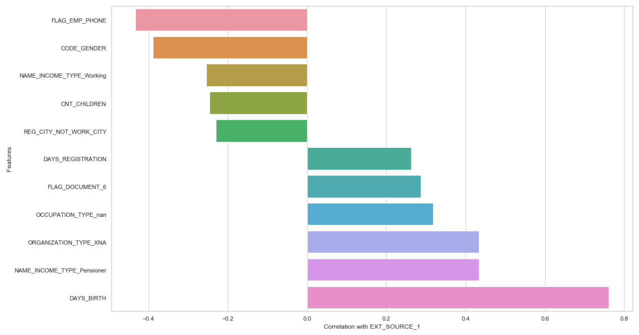

Fig. 5. High correlated features to *EXT_SOURCES_1*

B. Single Machine Learning Algorithms

At first a classical approach was addressed. Single machine learning algorithms were trained and optimized to classify the data. The following tree based boosting algorithms were chosen because most of the participants attend the challenge successfully with them. Furthermore these kinds of algorithms are fast in training and currently the common ones winning challenges on Kaggle [7].

e**X**treme **G**radient **B**oosting (**XGBoost**) [7]: This algorithm is a variant of the Gradient Boost Machine (GBM) [9] [10] where ensembles of weak models, like decision trees are trained sequentially. This is reasoned by the dependency of the previously trained and classified data, while training the next model. The false positive and false negative classified records will get higher weights for training the next classifier. This implies that those cases will be looked at more precise to correctly predict them over the training period, which is called boosting. At the end the learned models will be weighted and combined according to their error rate. XGBoost uses this technique but is efficient in training due to parallelizing the computation of statistics and the presorting of the data. This speeds up the algorithm. Like the most GBM, XGBoost builds trees level-wise (horizontally), which is shown in Figure 6 (bottom).

Light Gradient Boost Machine (**LightGBM**) [8]: The difference between LightGBM and XGBoost is mainly the growth of the trees. The trees do not grow level-wise like in the XGBoost algorithm, instead they grow leaf-wise like in Figure 6 (top). The vertically growth leads to different tree sizes with different depths. Thereby the trees are capable of reducing the loss in a specific leaf by growing that one more. Also improvements are made in terms of computations when dividing the data in leafs. Instead of going through all data instances to calculate the information gain for the feasible divisions only a small amount of data is used. The records with big gradients will be kept and the one with small gradients excluded. In [8] it is proved that by this method a quite good estimation of the information gain with a small amount of data is possible. Also a reduction of features by bundling those who strongly correlate together is done. By that an even greater reduction of the data is achieved. All of this speeds up the process of training with close to minimum loss of accuracy [8].

The two mentioned algorithms are optimized by using the grid search cross validation [6]. The hyper-parameter tuning works through an exhaustive generation of parameter combinations from a grid of parameters. The model with the best parameter combination regarding to accuracy in cross validation is than used as the estimator for classifying the data.

C. Stacking

In the second approach we used a ensemble technique called stacking (short for *stacked generalization* [2]). The idea behind stacking is to aggregate the predictions of multiple classifiers (called base- or level-1- learners) and use this predictions to create a new dataset for a final classifier (called a blender-

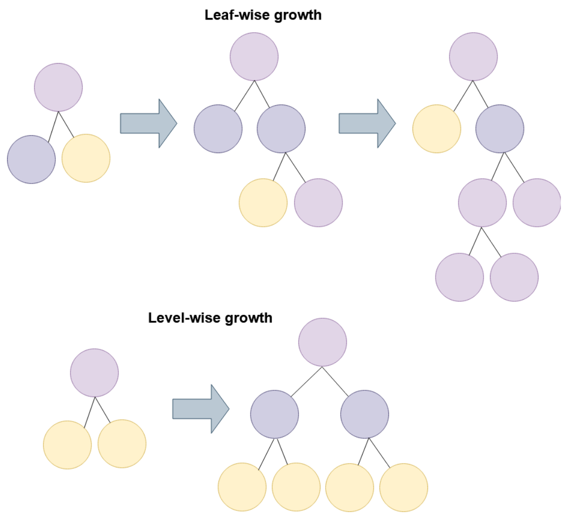

Fig. 6. Different Tree Growing

or meta-learner). The purpose of this technique is to learn a better function to aggregate the prediction of base learners, instead of just using trivial functions such as majority or average voters. Figure 7 shows the basic model architecture of stacking. Each of the base-learner predicts a different value, the meta-learner then takes those predictions as input and makes a final prediction.

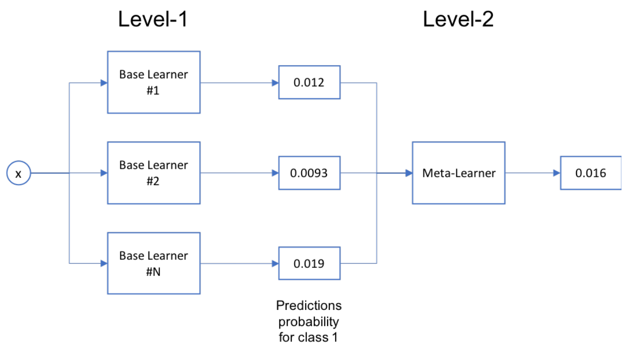

Fig. 7. Basic architecture of stacking

There are different approaches to train the base- and meta-learner [3], [5]. We tried two of them. In the first approach we splitted the train dataset into two subsets with 60% and 40% of the data. All splits were made with stratified subsets to preserve the percentage of samples for each target class. The first subset with 60% was used to train the base-learners and the remaining 40% to make predictions. These predictions were then used to create a new dataset (meta-dataset) with the single predictions of the base learners as features. Since the used metric for the evaluation of the challenge was ”ROC-AUC” we did not used the predicted class label of the base-learners but the probability for the class. The meta-learner was trained on the new meta dataset together with the original target values.

On the second approach the original train dataset was splitted into five stratified folds (see Figure 8). Each base-learner was trained five times on four different folds and then used to make predictions on the remaining fold as well as on the full original test dataset. At the end of training we had for each base-learner a full prediction on the train dataset and five different predictions on the test dataset. For training the meta-learner a new meta-dataset from the base-learner predictions on the original training set was created. Since we had five predictions per base-learner on the test dataset we took the average of the five predictions to create the the meta test dataset.

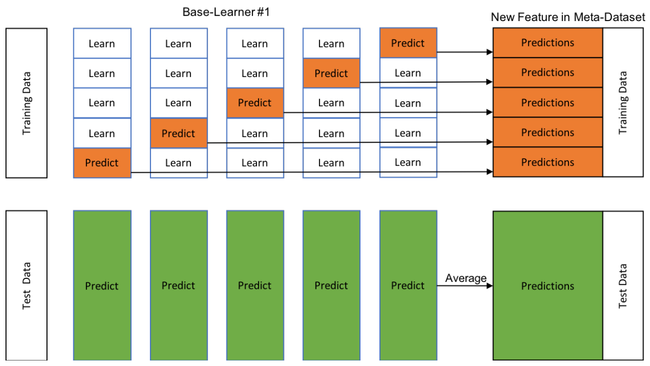

Fig. 8. Stacking with fold training

As base learners we used the two optimized models from the previous section IV-B (XGBoost and LightGBM) plus other nine models with mostly standard parameters. Table III shows the used algorithms for training the base learners and the number of models. For the tree-based algorithms we used different models with variation in the number of estimators.

TABLE III
USED BASE LEARNERS FOR STACKING

Algorithm	# Models
Logisitc Regression	1
Stochastic Gradient Descent	1
Random Forest	2
XGBoost	3
LightGBM	2
ExtraTrees	1
CatBoost	1
	Total: 11

For the meta-learner we tried three different algorithms (Logistic Regression, Stochastic Gradient Descent (SGD) and Multilayer Perceptron (MLP)). Hyper-parameters were tuned with Bayesian optimization [4]. The best performing model was a Multilayer Perceptron with 2 hidden layers (8 units on the first- and 2 on the second-layer) and

$$tanh(x) = \frac{2}{1 + \exp^{-2x}} - 1$$

as activation function. The originally input for the meta-learner was a $\mathbb{R}^{11}$ dimensional vector (base-learners). We also tried to do polynomial combinations of the inputs but with no success. A good performance boost was finally possible when we joined some of the engineered and original features (see section 4) as input for the meta-learner. Figure 9 shows the architecture of the meta-learner. We figured out that some of the base-learners including Logistic Regression and SGD lead to overfitting on the public leaderboard and therefore we didn't used this base-learners in our final stacking architecture.

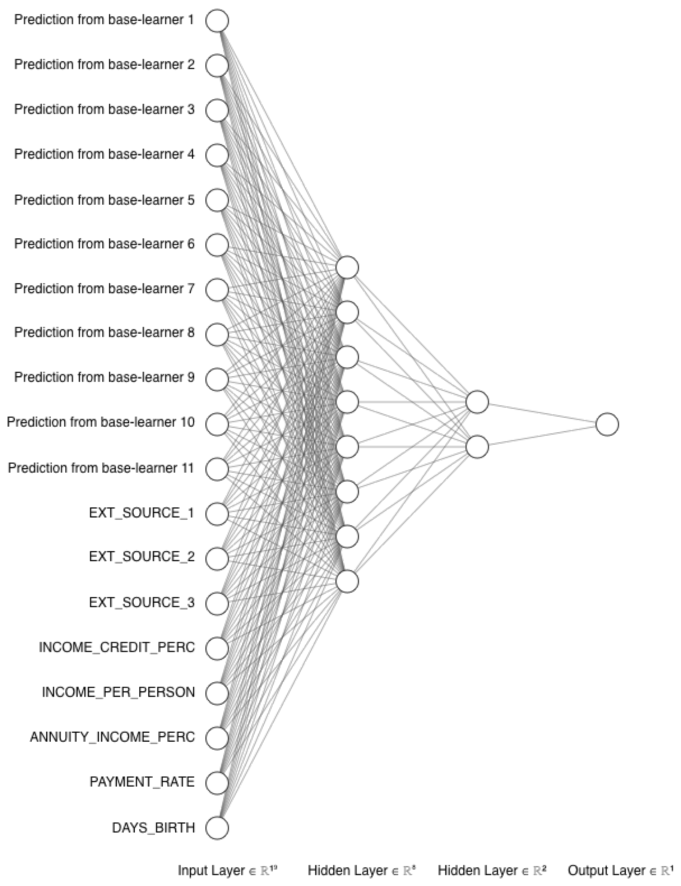
Prediction from base-learner 1
Prediction from base-learner 2
Prediction from base-learner 3
Prediction from base-learner 4
Prediction from base-learner 5
Prediction from base-learner 6
Prediction from base-learner 7
Prediction from base-learner 8
Prediction from base-learner 9
Prediction from base-learner 10
Prediction from base-learner 11
EXT_SOURCE_1
EXT_SOURCE_2
EXT_SOURCE_3
INCOME_CREDIT_PERC
INCOME_PER_PERSON
ANNUITY_INCOME_PERC
PAYMENT_RATE
DAYS_BIRTH
Input Layer ∈ ℝ¹⁹
Hidden Layer ∈ ℝ⁸
Hidden Layer ∈ ℝ²
Output Layer ∈ ℝ¹

Fig. 9. Architecture of meta-learner

V. RESULTS

In this section we will present the results of our previous described approaches. Since the provided test dataset from Kaggle doesn't include the targets, the presented results are derived with five fold cross validation (CV) on the training-dataset and the submission results on the public (PubLB) and private leaderboard (PriLB).

Figure 10 shows the results of the first approach (section IV-B), where the single optimized algorithms XGBoost and LightGBM are compared. The used datasets are with and without the engineered features (EF) from section III-C. We can see that XGBoost in general performs better than LightGBM, on CV as well as on the leaderboards. Also visible is that the engineered features improve the results for XGBoost in all cases and for LightGBM at least on the test data (PubLB and PriLB). So the best results here are achieved with XGBoost on the dataset with engineered features.

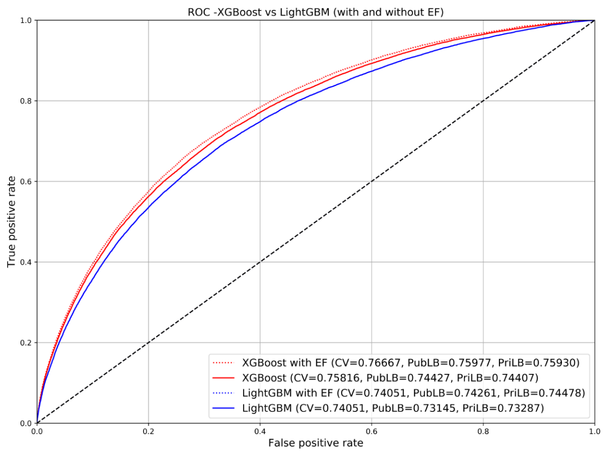

Fig. 10. ROC - XGBoost versus LightGBM

Figure 11 and 12 show the results of the stacking approaches for the three used meta-learner algorithms. We can see that there's no big difference between the meta-learners. Furthermore, it's visible that all models perform slightly better on CV than on the leaderboards. On the second stacking method (explained in Figure 8) the gap between CV and leaderboard scores is smaller.

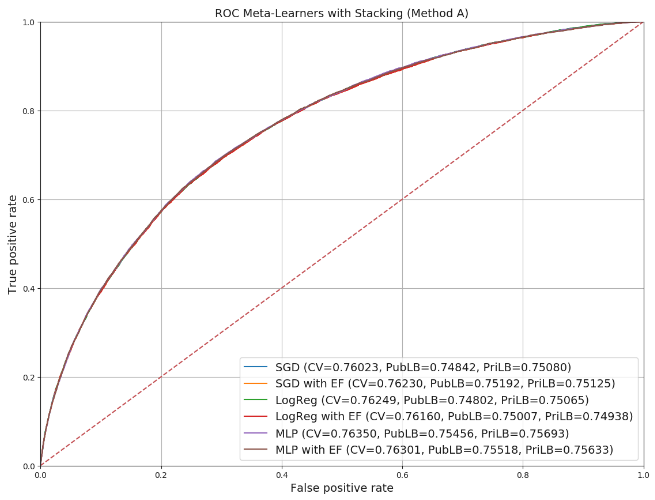

Fig. 11. ROC Meta-Learners

Our overall best result was achieved with the stacking method on the dataset together with the engineered features. For the meta-learner we used a Multilayer Perceptron (see Fig. 9). This submission equals place 4862 on the public and place 4803 on the private leaderboard.

VI. Conclusion

In this paper we described our approaches on attending the Kaggle challenge "Home Credit Default Risk". The preparation of the data as well as the analysis is explained to comprehend how the results are achieved. Two approaches, single

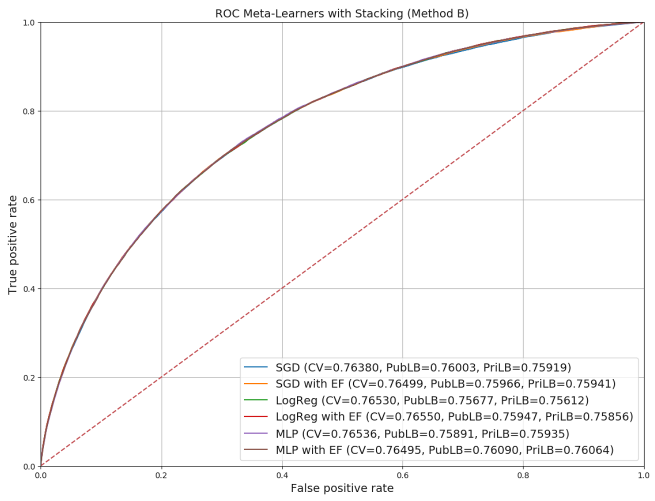

Fig. 12. ROC Meta-Learners

machine learning from section IV-B and stacking from IV-C are compared and it is demonstrated that both achieve similar results. This shows that one cannot decide in general which method performs better without testing. Although, stacking helped us to get better results on the leaderboard without the need to do computational expensive hyper-parameter optimization. It is important to mention that only a small amount of the provided data was used (explained in II). Nevertheless, our best approach could reach a score up to 76% which is only approximately 4% less than the first place solution (PriLB = 0.80570, PubLB = 0.80920).

References

[1] Kaggle Competition - Home Credit default Risk https://www.kaggle.com/c/home-credit-default-risk

[2] Wolpert, David H. "Stacked generalization." Neural networks 5.2 (1992): 241-259.

[3] Deroski, Saso, and Bernard enko. "Is combining classifiers with stacking better than selecting the best one?." Machine learning 54.3 (2004): 255-273.

[4] Snoek, Jasper, Hugo Larochelle, and Ryan P. Adams. "Practical bayesian optimization of machine learning algorithms." Advances in neural information processing systems. 2012.

[5] Sill, Joseph, et al. "Feature-weighted linear stacking." arXiv preprint arXiv:0911.0460 (2009).

[6] Pedregosa, Fabian, et al. "Scikit-learn: Machine learning in Python." Journal of machine learning research 12.Oct (2011): 2825-2830

[7] Chen, Tianqi, and Carlos Guestrin. "Xgboost: A scalable tree boosting system." Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining. ACM, 2016.

[8] Ke, Guolin, et al. "Lightgbm: A highly efficient gradient boosting decision tree." Advances in Neural Information Processing Systems. 2017.

[9] Friedman, Jerome H. "Greedy function approximation: a gradient boosting machine." Annals of statistics (2001): 1189-1232.

[10] Friedman, Jerome H. "Stochastic gradient boosting." Computational Statistics & Data Analysis 38.4 (2002): 367-378.