A Work Project, presented as part of the requirements for the Award of a
Master Degree in Finance from the NOVA School of Business And Economics

ESTIMATION OF CREDIT RISK FOR MORTGAGE PORTFOLIOS: SELECTION AND OPTIMIZATION OF MACHINE LEARNING MODELS

Field Lab Project Nova SBE & Moody's Analytics

Dominique Sebastian LADE, 33873

A project carried out on the Master in Finance Program, under the supervision of:

Moody's Analytics Advisors

Dr Bahar KARTALCIKLAR, PhD, Risk Modeller - Content Solutions

Dr Petr ZEMCIK, Senior Director - Content Solutions

Carlos CASTRO, Senior Director - Economics and Structured Analytics

Faculty Advisor

Professor João Pedro PEREIRA

January 3, 2020

Models

In this section, we present the reasoning for the choice of the dummy classifier as well as the logistic regression as a baseline model. Moreover, a comparison between the characteristics of each machine model discussed in Section 2.2 is done in order to evaluate which one suits better for the problem under analysis. Finally, a boosted tree algorithm is the one we select to use in the remainder of the paper, precisely, the specific implementation of our choice is the extreme gradient boosting (XGBoost) model.

Model Selection

Baseline Model

The goal of this paper is to build a model that is able to successfully measure the default risk of a pool of mortgages. Thus, to fairly evaluate the performance of the model developed in this thesis, it is necessary to define a baseline model to compare it with.

The most basic and important condition for the model is that it performs better than a random prediction or no prediction at all. The baseline is a prediction which does not take the available features into account, but instead makes a prediction based on a (deterministic) predefined rule. Since our dataset is heavily unbalanced with roughly 4% of defaulted loans vs. 96% of non-defaulted loans, a classifier which always predicts that the loan is not in the default state gets immediately 96% of the predictions right. Using this classifier as a baseline, we ensure that the machine learning model can extract valuable information from the available features at all. This basic classifier is called for the remainder of this paper dummy classifier.

Currently, the industry standard to calculate risk scorecards are logistic regressions (LR). These have the advantage that they are easy to implement and fast to train and retrain. Besides, a logistic regression model is easy to interpret. However, in mathematical terms, a logistic-regression is a linear model wrapped into a sigmoid function. Therefore, without tricks in the data pre-processing, the model can not capture non-linear behaviour or interaction between features. Furthermore, according to Siddiqi (2006, p. 88) a typical credit risk scorecard, as it is used in the industry, contains between 8 to 15 different variables. Eliminating variables certainly

helps on the interpretability side but it might weaken the performance of the model. Since the LR is a linear model, this is covered as good as possible during feature selection, as described in Section . However, given the fact that machine learning models take into account interactions between explanatory variables, eliminating variables based on univariate tests might lead to a significant drop in this type of models' performance.

Machine Learning Model

As an initial requirements list, we needed a model that can deal with a large dataset in a reasonable time, that is able to capture highly non-linear effects on the target variable and to capture interaction effects between features. Moreover, the model must be able to work under the presence of imbalanced classes, since the majority of the loans present in the data are of the non-default class.

In a world with unlimited available computational power, it would be desirable to train every available model with different sets of hyperparameters on the data to evaluate which model performs best. Since the dataset is rather large and there is a nearly unlimited combination of models and hyperparameters, we decided to conduct a qualitative analysis. In computational terms, the main effort is in optimizing the hyperparameters for the model of our choice.

Considering the advantages and disadvantages of each model, discussed in Section 2.2, it is possible to evaluate which model suits best for the machine learning problem under analysis.

Table 2: Comparison between characteristics of the considered machine learning models.

	Explainable	Large Datasets	Imbalanced Classes	Non-Linear Effects	Feature Interaction
Logistic Regression	✓	✓	✓	✗	✗
Naive-Bayes Classifier	✓	✓	✗	✗	✗
K-nearest neighbors	✓	✓	✗	✓	✗
Support Vector Machine	✓	✗	✗	✓	✓
Decision Tree	✓	✓	✓	✓	✓
Boosting Trees	✓	✓	✓	✓	✓
Voting Classifier	✗	✓	✓	✓	✓
Neural Networks	✗	✓	✓	✓	✓

As shown in Table 2, a boosted tree algorithm seems to be the one satisfying all of the required conditions for the type of data that is being used for the purpose of this project. More-

over, it provides a good trade off between expected performance on structural data and reasonable computational performance. Therefore, for the remainder of the paper, we are going to focus on this type of model.

Extreme Gradient Boosting

As discussed in Section , the model of our choice is a boosted tree algorithm. There are several state of the art implementations available which vary mainly in the computational speed and in the way how to deal with categorical variables and missing values. The most widely used and successful implementations are currently LightGBM[3] developed by Microsoft, CatBoost developed by Yandex[4] and XGBoost developed by Tianqi Chen and Carlos Guestrin, two researcher from the University of Washington (Chen and Guestrin, 2016). Similar to Section 2.2, it would be preferable to train all of the models and select the best performing one, however, due to limits in computational resources, the team focuses again on a qualitative comparison.

In our opinion, the XGBoost algorithm has the greatest odds to deliver the best results. As explained in Section 2.2, a boosting tree algorithm trains the subsequent tree on the residuals of the previous one. Usually, this kind of models is rather slow since, for most of the boosting approaches, it is not possible to train several trees in parallel. However, Chen and Guestrin developed an algorithm to train them in parallel and, therefore, reduce the computational runtime by a large magnitude. This allows for a greater exploration of the hyperparameter space. Moreover, for a normal machine learning algorithm, the researcher has to decide on how to encode missing values. For numerical variables, XGBoost is automatically able to detect the best direction of missing values. For our dataset, this is a strong argument in favour of this algorithm. Other algorithms, like CatBoost, are able to handle the encoding of categorical variables natively. This would be beneficial for our dataset as well. However, internally, these algorithms use a way of target based encoding for these variables. We mimic this effect by manually encoding the categorical variables with the weight-of-evidence encoding.

[3]LightGBM by Microsoft: `https://github.com/microsoft/LightGBM`
[4]CatBoost by Yandex: `https://catboost.ai/`

Hyper Parameter Optimization

A machine learning algorithm has parameters which are external and which are internal to the model. Internal parameters are learned based on the given input data. For a logistic regression without regularization, the internal parameter are the regression coefficients. In opposition, the algorithm can not determine the optimal value for the external data based on training data. However, the value for the hyperparameters greatly influences the overall model performance or, in particular, the performance on holdout data. External parameters are usually referred to as hyperparameters. An example of a hyperparameter for a Decision Tree would be the maximum depth that it can achieve. In case of a logistic regression with LASSO regularization, the main external parameter that would influence the results of the model would be the λ, or regularization, as explained in Section . For the remainder of this section, parameters are used as synonyms for external- and hyperparameters.

Optimization Techniques

Generally speaking, the process of optimizing the hyperparameters is as follows: one trains the model with a certain set of parameters. Evaluates the performance of the model and continues training the model with another set of parameters. Selection of parameters for which the models will be evaluated can either be done manually or computationally. The manual approach is used for simple models with a small set of (discrete) hyperparameters or, for models where a strong domain knowledge can lead to an advantage. This process can be used to optimize the hyperparameters for the baseline model, the logistic regression. The most important parameter for the baseline model is the choice of the regularization type. This could be either the so-called LASSO regularization, Ridge regularization, Elastic-Net regularization, which reflects a mixture of both, or none of these. The LASSO regularization could potentially set the coefficient for some parameters to zero, which would effectively eliminate these parameters. Due to the extensive statistically feature selection performed in Section , this was not the desired outcome. Moreover, at this stage, we wanted to keep statistically properties of the regression interpretable. Which would not be possible with any type of regularization. Therefore, the logistic regression was performed without any regularization applied.

While the manual approach was sufficient for the baseline model, it was impossible to apply for the gradient boosting algorithm. For the XGBoost model, 13 parameters were to be optimized. These parameters were all continuous. Some of these parameters were highly co-dependent on each other. Therefore, the parameters were computationally optimized. The standard method one could use is the so-called grid-search. This approach is based on a predefined set of values for each parameter. These values are combined in a way so that every possible combination of parameters is tested. Given that we want to optimize n different hyperparameters with k different possible values for each parameter, that would lead to k^n different combinations. In our case, given that we want to check just 10 different values per parameter, it would have been necessary to test 10^{13} different models. In reality, it is necessary to multiply this number by the number of cross-validation folds performed. Given infinitive small steps for the hyperparameters, grid-search would lead for sure to the optimal combination. However, this algorithm explores heavily also the inefficient parts of the parameter grid. Therefore, in a time and computational constraint setting, this is not feasible.

The next method one could try is to use a random-search algorithm. The parameters for which the model is to evaluate are sampled from a given distribution for each parameter. This reduces the risk of missing optimal parameter combinations due to a too wide-meshed parameter grid. However, the algorithm still explores inefficient areas of the parameter grid. Also, one has to specify a hard stop after a certain amount of time or evaluations.

Both the manual and random-search approaches are simplistic in the sense that all evaluation steps are calculated independently from each other. Earlier in 2019, Akiba et al. (2019) proposed a new state of the art hyper-parameter optimization framework, called Optuna. Similar to the random-search algorithm, the input for this algorithm is a distribution for every parameter. In contrast to the random search, Optuna uses a Tree Parzen Estimator to obtain the next parameter combination to test. The main idea is that the next set of parameters is obtained with the knowledge of the result of the previous evaluation. Therefore, the algorithm explores efficient areas of the parameter grid more extensively and tries to avoid inefficient ones. Moreover, Optuna implements the so-called pruning of inefficient trials, which is essentially an early stopping if a model is expected to show a bad performance. As already mentioned in the Section , our

final model is a tree boosting type. Therefore, several hundred trees are calculated, whereas the next tree is fitted upon the residual errors from the previous one. If a model shows a bad performance during the first 50 trees and is not expected to show a performance increase afterwards, Optuna automatically stops the evaluation of this model and continues with the next set of parameters.

Obtained Hyper Params

Table 3: Comparison of the hyper parameter obtained with Optuna for XGBoost.

Parameters	Default	Binned	Unbinned
# Estimators	-	141	656
Max Depth	6	17	14
Learning Rate	0.3	0.099	0.013
Scale Positive Weight	1	4.464	5.460
Max Delta Step	0	1.142	7.060
Alpha	0	25.005	0.000
Lambda	1	0.013	3.530
Gamma	0	0.000	0.000
Min Child Weight	1	16.480	0.000
Subsample	1	0.394	0.658
Colsample by Tree	1	0.923	0.822
Colsample by Node	1	0.554	0.777
Colsample by Level	1	0.922	0.740

This section is meant to give an overview of the obtained hyperparameters during the optimization with Optuna. For the sake of reproducibility, all obtained hyperparameters are reported in Table 3. However, just the most important ones are discussed in this section. Some of the parameters show no relevance at all whereas others are highly co-dependent on the value of others and therefore hard to interpret. For a detailed reference of all parameters please consult the parameters section of the official XGBoost documentation[5].

As already explained in Section , this thesis is based on two final datasets. One obtained with the process described in Section 3, while the other one consists of the exact same set of features, without being binned. On one hand, the binning of the data ensures easy interpretability of the models' results. Moreover, for models which are natively not able to discover feature interac-

[5] `https://xgboost.readthedocs.io/en/latest/parameter.html`

tions and non-linearity like the logistic regression, binning gives the model a chance to capture these interactions. With this approach, the logistic regression has not just one coefficient per feature but one coefficient per category. Therefore, binned data can improve the performance of certain models while improving the interpretability at the same time.

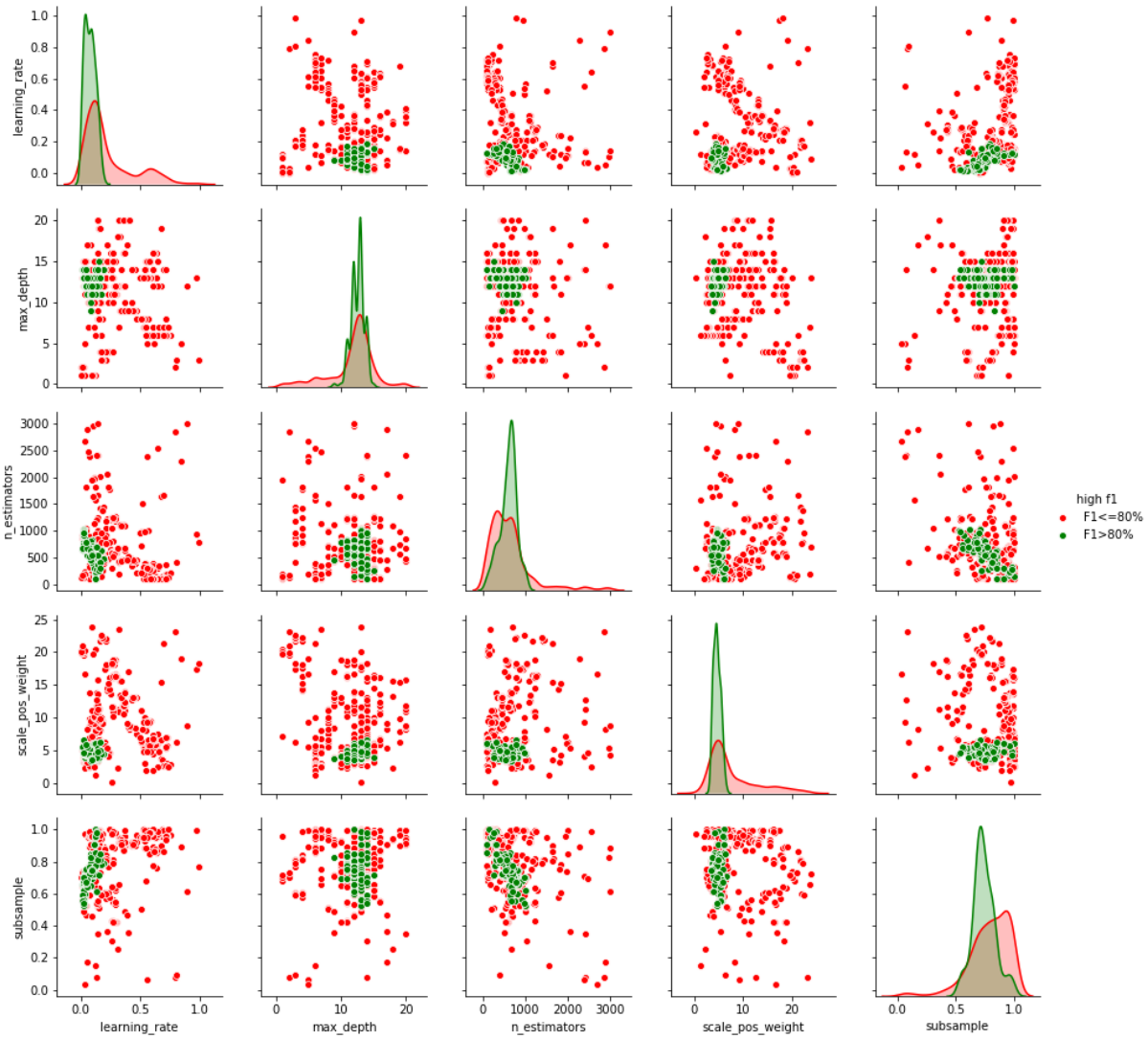
Nevertheless, binning also reduces the information-value in the data. In general, that tends to decreases the performance for high-performing machine learning algorithms due to the loss of information. As discussed in Section 4, this decrease is also true for the model used in this paper.

During the hyperparameter search we observed the relationship of tested hyperparameters and the improvement of the target metric. For the binned dataset, XGBoost finds the best performing model quite early. Several other tested combinations reach nearly the same performance, but none of it is able to outperform the one found earlier. The opposite is true for non-binned data. Even after several hundred attempts the algorithm still finds a model which performs slightly better. Even after the pruning, the distribution of the models' performance is wider. Which implies that XGBoost is more sensitive in a change of hyperparameters for the non-binned data in comparison to the binned data. These observations lead to the conclusion that the non-binned data indeed contain more valuable information and, more importantly, the boosting algorithm is also able to extract it.

Especially during the first trials, the parameters the algorithm uses to test the models' performance are randomly sampled from a given distribution. Even though the algorithm is empirically proven fairly stable and tends to deliver near-perfect results in a fraction of the time (Lindauer et al., 2019), we acknowledge that some of the interpretation might be influenced by a few luckily first trials. The results might change if one tries to reproduce them with a different initialization of the random number generator.

The number of estimators, which is the only parameters without a default value, is the most important parameter for a boosting algorithm. It defines how many decision trees are fitted in a subsequent order. Generally speaking, if the model performance improves with an increase of estimators, it means that the model is still able to extract information from the data. However, the performance has to be evaluated on a separate holdout set to prevent overfitting. The fact

Figure 6: The empirical distribution of a subset of hyperparameters which were tested during the hyperparameter optimization. Overall, 574 tree boosting models with different hyperparameters are trained. Tested parameters along with the obtained F1 score are visualized.



that the number of estimators is more than four times higher for the unbinned data compared to the binned data, is a strong indicator that the model can extract more valuable information out of the dataset.

The maximum depth parameter defines how many levels the model is allowed to construct a single decision tree. A depth of one would prevent the model to gain any insights about the interaction of features. The default value is a depth of six, whereas the optimal parameters for the Italian mortgage market are 17 and 14 for binned and non-binned data, respectively. These high values are a strong indicator that the interaction of features has high importance to estimate the default of a mortgage loan.

Another benefit of the XGBoost algorithm is the fact that it is possible to manipulate the weight of certain classes. The scale positive weight parameter allows putting more weight on defaulted loans. Due to the highly imbalanced dataset, this parameter prevents scenarios where the algorithm just predicts the majority class.

Most of the other remaining parameter, in particular, the learning rate, are meant to control overfitting. The biggest concern is that the algorithm has an outstanding performance on the training set, but fails to apply this knowledge on the test set. This happens if the algorithm is fitted to noise in the training set, which is not present in the test dataset. This parameter is more than seven times smaller for binned data compared to unbinned ones. Therefore, the algorithm incorporates information from a single decision tree just with a small fraction into the final estimate to prevent fitting to noise. This leads again to the conclusion that the unbinned data contain more information, which the boosting algorithm is aware of and able to use.

Generally speaking, the selected hyperparameters are a strong indicator that a machine learning model can extract more valuable information from non-binned data compared to binned ones. This is mostly due to the fact, that machine learning models can estimate non-linearity, non-monotonicity and feature interactions. Therefore, just changing the model on an existing data source is not sufficient. One has to revise the whole end-to-end credit risk pipeline.