FACULTY OF ECONOMICS AND BUSINESS ADMINISTRATION

InClass Kaggle Competition: loan default prediction

Technical report

Course: Machine Learning

Teacher: Prof. Dr. Dries Benoit
Arthur Thuy (TA)

Students: Team 08

Lauren Janssens	02108568	lauren.janssens@ugent.be
Wouter Dewitte	01806106	wodwitte.dewitte@ugent.be
Lennert Van den Broeck	01706979	lennert.vandenbroeck@ugent.be
Tibo Walrave	01607568	tibo.walrave@ugent.be

Academic year: 2021–2022

GHENT UNIVERSITY

Contents

1 Introduction

Prediction on loan default is essential for banks because it has a strong impact on their profitability. Gaining insight in the factors that play a role in these defaults could enable decision makers to change their loan offers. For example, if an applicant is likely to default the bank could simply deny the loan, reduce the amount or charge a higher interest.

In this report we will elaborate on the decisions we've made in order to come up with our predictive model for the in-class kaggle competition. We let us guide by the CRISP-DM process model.

2 Data Preprocessing

2.1 Data Understanding

In order to get a better understanding of the data we made use of the standard visual descriptive tools such as boxplots, histograms and in some cases barplots. This enabled us to get a quick overview of each feature at hand. We recognized that some features could be divided into more specific features, for example the address variable. In addition we made use of the 'ggplot2' library to run some dependency plots for each variable.

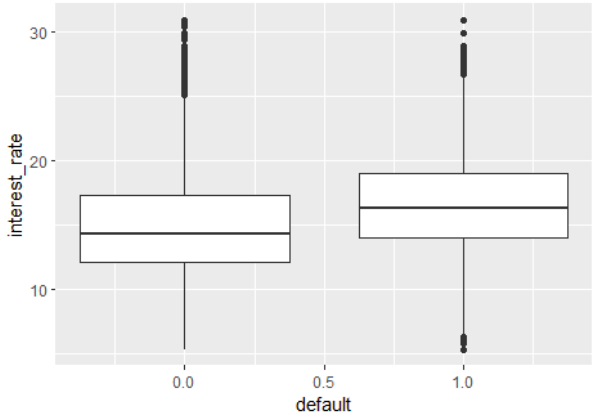

Figure 1: Dependency plot: interest rate

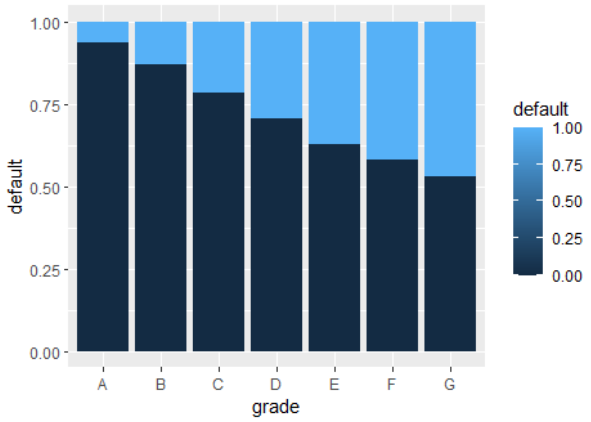

Figure 2: Dependency plot: grade

2.2 Data Cleaning

The first step in our data cleaning process was to create some functions to avoid repetitive tasks. We coded functions to replace missing values, handle outliers, calculate the mode, create categories and create NA flags. Next we checked which columns included missing values and we applied the NA flag function to indicate the data points with missing values. We also handled outliers with our predefined function.

Then we zoomed in on each feature in more detail. We handled the missing values by imputing the appropriate value. This value could be the mean, median or mode depending on the situation.

2.3 Feature Engineering

In step (2.1) it was immediately clear some feature distributions were skewed. To these certain features we applied transformations to get a more symmetric distribution. We applied a square root transformation on monthly payment, for instance. Moreover we encoded our categorical features and also binned some numerical variables. Furthermore we standardized and scaled our data, we exported them into 2 separate files. In the modelling phase we work with these files.

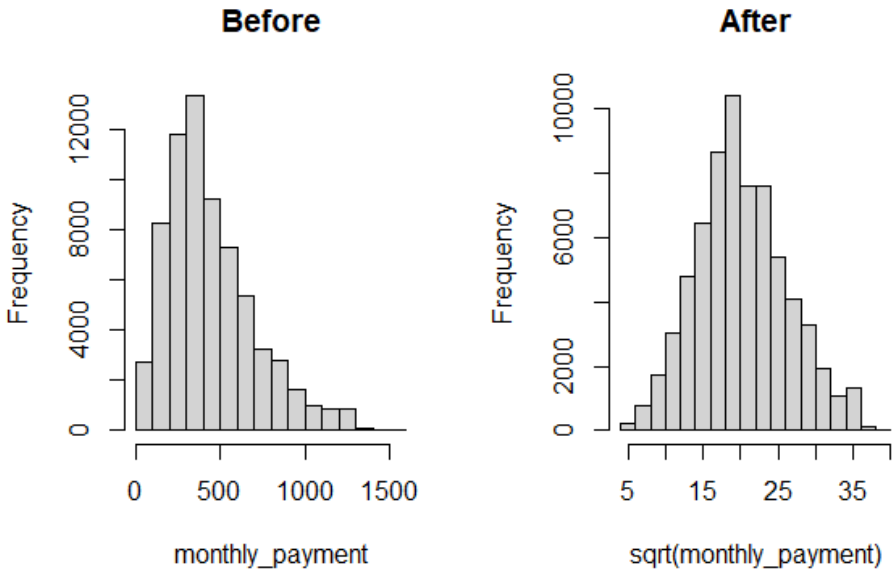

Figure 3: Histograms before and after square root transformation

2.4 Class Imbalance

When creating our model we did not take class imbalance into account. But afterwards, we ran our LBGM model on balanced data to see if it had a significant effect. These results are further discussed in 3.5 Light Gradient Boosting.

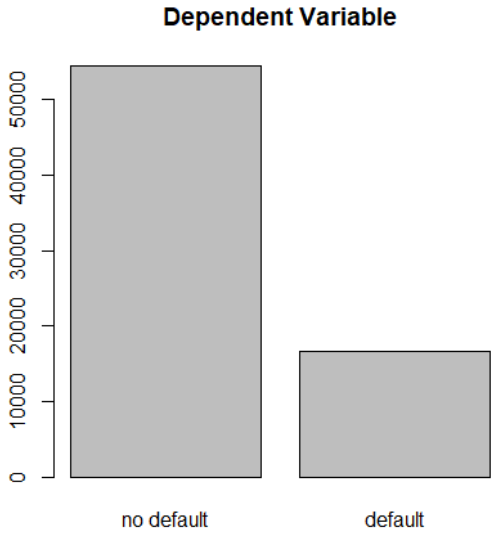

Figure 4: Class imbalance between default - no default

3 Modeling

3.1 Modeling Framework

When creating our model, it was important for us that we could test it independent of the limit on Kaggle. We created an internal and an external way of testing. The internal way was created by splitting the training csv in 80% train set and 20% validation set. This was done in the "BasetableCreation_Splitting.R" file. When we were happy with our internal AUC, we used the model to perform external testing. For the external testing, we had to run our data preprocessing again but on the whole training csv, to optimize the training ability of our model. To switch efficiently between the two ways of testing, we implemented both code possibilities and used commenting to switch.

3.2 Feature Selection

The dataset at hand contains ±70.000 observations with 24 raw features. These raw features can not be fed directly into our model. In the preprocessing step, we performed a first feature selection step based on intuition (field knowledge) and basic dependency plots (See 2.1). The remaining features were encoded and preprocessed in order to obtain numeric scaled features (See 2.2 and 2.3). This results in 46 clean features which are likely to have predictive power in our classification model (see appendix A). Because features with no predictive power can introduce noise and potentially lead to overfitting (James et al., 2021), we conducted a quantitative feature selection step in order to further extract the number of relevant parameters.

There exist a number of feature selection methods like best/stepwise subset selection which are explicit methods. Alternatively we could use implicit feature selection methods like shrinkage or even by making use of dimensionality reduction techniques (James et al., 2021). We implemented shrinkage methods as well as the efficient forward stepwise selection method, both applied on logistic regression. The stepwise selection method resulted in the best AUC performance and outputs the optimal combination of parameters in the logistic regression model explicitly. The number of features are reduced from 46 to 26. The selected features are highlighted in appendix A as well. In the rest of this project, we worked with these parameters. The code for both implementations of feature selection can be found in the R files *Logistic_Regression.R* and *Feature_selection_ridge_classification.R* respectively.

We made the assumption that the features that contributed to the optimal logistic regression model will also be relevant for the other models **and** that the features that didn't contribute for logistic regression will not contribute to the other models either. We are aware that non linear and interaction effects can play a role and irrelevant classified features in the logistic model can be of high interest in other models. However this is the assumption we made in this project.

3.3 Logistic Regression (lr)

3.3.1 Introduction

The logistic regression model is a binary classification model. It models the probability that Y belongs to a certain class. lr is considered as a traditional regression model. This model was used to set a baseline for our feature engineering. The benefit of this model is that it is basic but still performs relatively good when compared to more complicated machine learning models, when using few optimized features. (Lynam et al., 2020)

3.3.2 Model

The lr model uses a logistic function to model a binary dependent variable. It estimates the parameters of a logistic model.lr has the dependent variable, default, with two possible outcomes, labeled "0" and "1". The model converts the log-odds to probabilities, so the outcome is a probability of default between "0" and "1". There are no variables that need to be tuned, so that makes it also a more convenient method.

3.3.3 Code

The code can be found in the R files called *Logistic_ Regression.R*. In this file, we also performed the forward stepwise selection to determine the most important features.

3.3.4 Results

Logistic regression gave us good results to start our modeling with. We mostly used this model to determine if our feature engineering was succesfull. Firstly, we used this model on all our features. Secondly, we used it on our manually selected features and we finished by performing it on the forward stepwise selected features. When testing on our internal validation set, the best AUC we got was **0.9049**. When testing on our external validation set, our optimal AUC was **0.8954**.

More details about the final lr model and its estimated coefficients (absolute values and sign) and significance in terms of p-values can be found in appendix B.

3.4 Random Forest (rf)

3.4.1 Introduction

Random forest is a tree based method. In rf a large number of trees is grown where a split only considers a random sample of m predictors. (James et al., 2021) The random forest was chosen because in most cases, it performs better than the logistic regression, according to Couronné et al., 2018. It should be noted that performance still depends on the data-set that is used.

3.4.2 Model

As mentioned above the rf builds multiple trees while only selecting a random sample of m predictors. As a result the most important parameters of the method are the mtry, maxnodes, nodesize and ntree. Mtry is the number of variables that are randomly sampled, which resolves the correlation issues between the trees. Maxnodes limits the depth of the tree, thus curbing over-fitting. The nodesize is the size of the terminal nodes. The larger the value, the less the method over-fits.(Breiman, 2002) To determine these parameters a tuning approach was used. Due to limited computing power only the nodesize and mtry have been tuned. After tuning the optimal parameters were applied to a rf model.

3.4.3 Code

The code can be found in the R files called *RandomForest_ Tuning* and *tree-based_ methods*. First the tuning has to be applied. In the tuning file parallelization has been used to speed up the calculations. First a task is made. Then the parameters grid is created. Then the validation and optimization strategy are determined. Finally the tuneParams function is called. After receiving the optimal parameters, listed in appendix C.1. These parameters are used in the *tree-based_ methods* R file.

3.4.4 Results

Unfortunately the tuning and random forest were computationally very expensive. Rf also did not give any improvement over logistic regression. The AUC value was on the test set was **0.8925**. Which was a decrease compared to the logistic regression. The method was abandoned and a more advanced method was sought after.

3.5 Light Gradient Boosting (lgb)

3.5.1 Introduction

The light gradient boosting is a method that tree based learning algorithms. The algorithm has proven its worth in practical applications, as mentioned by ML6. The main advantage is the combination of speed and quality. (Ke et al., 2017) Having limited computing power the method seemed appropriate.

3.5.2 Model

The main difference between lgb and more traditional tree based algorithms is that it builds trees horizontally. This means that the tree is build in a lead-wise.(Pushkar, 2017) A graphical representation can be seen in figure 5.

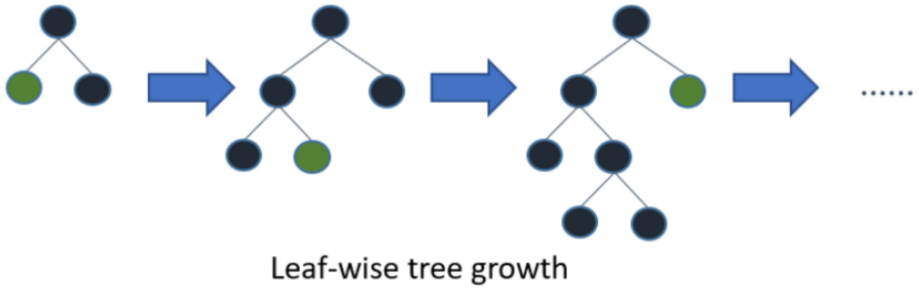

Figure 5: Building process lgb (Pushkar, 2017)

The parameters that have been used in our model are the num_leaves, max_depth, metric, boosting, learning_rate, num_iterations, min_data_in_leaf. The num_leaves is the number of end leaves of the tree. Setting this to a low value results in a robust model. A high value introduces over-fitting. The max_depth is the maximum number of levels of the tree. A higher max_depth results in a better fit on the training data. The metric is the objective of the method. The boosting parameter is the boosting algorithm that is performed on the data. The learning_rate is how fast the model learns when growing trees.A slower learning rate often has the advantage of creating more robust and efficient model. But a low learning rate should be accompanied by a high number of iterations so that the model has enough time to learn. The num_iterations should also not be too high because that could introduce over-fitting. The min_data_in_leaf is mainly used to combat over-fitting. (Bahmani, 2021)

3.5.3 Code

First of all the required packages have to be installed. These are lightgbm and pROC. Next the required data has to be read in. The *lgb.train* function requires the data to be of the form *lgb.Dataset*. To transform the data into a *lgb.Dataset* the data has to be transformed into a matrix. After creating the correct dataframe the parameter grid has to be defined. After assigning the parameters the model can be trained by the *lgb.train* function. Then we apply the predict function to receive the predictions. The predictions can then be used to get the auc value for the validation set.

3.5.4 Results

The eventual parameters setting for our optimal model can be found in appendix C.2. The number of leaves and maximum depth are rather low to curb over-fitting on the training data. The learning rate is low so that the model can achieve a more robust solution by taking small steps. As a consequence a rather large number of iterations was needed to compensate for the low learning rate. The actual values for the parameters were found by re-running the program and changing them accordingly. Note that extensive tuning

could result in a better parameter setting, but this was not in the scope of our computing power. Using the insight behind the parameters a rather good solution has been achieved.

In terms of AUC the model scored around **0.9085** on the internal validation set. The confusion matrix for the internal validation set can be found in table 4. The AUC of the test was **0.9032** which is a only a minor decrease.

We also took a look at the results if we took the class imbalance into account. When we did oversampling to fix the class imbalance we have an AUC of **0.9085** on our internal validation set, which is not significantly different from the AUC with class imbalance. We also did undersampling to fix the imbalance. This gave us an AUC of **0.9077**, which is also not significantly different. After doing these steps, we concluded that fixing this class imbalance problem would not give us a significant better model.

4 Evaluation & Deployment

The resulsts of the previous section are summarised in table 1. For every model we notice a drop in the performance when applied on the external test set. Next we notice that the best performing model in terms of AUC is the lgb model.

	validation set (internal)	test set (external)
lr	0.9049	0.8954
rf	0.8990	0.8925
lgb	0.9085	0.9032

Table 1: Overview of AUC scores of the different models (logistic regression, random forest and light gradient boosting) on the internal validation and external test set of the public leaderboard on Kaggle.

In the previous sections, our goal was to find the best performing model on predicting loan defaults in terms of AUC. This measure is however general and difficult to interpret by itself. Because different errors of a deployed model translates into different costs, we better have an idea where the model is likely to fail. This can be achieved by looking at the ROC curve (see figure 6) and/or look into some (threshold dependent) confusion matrices.

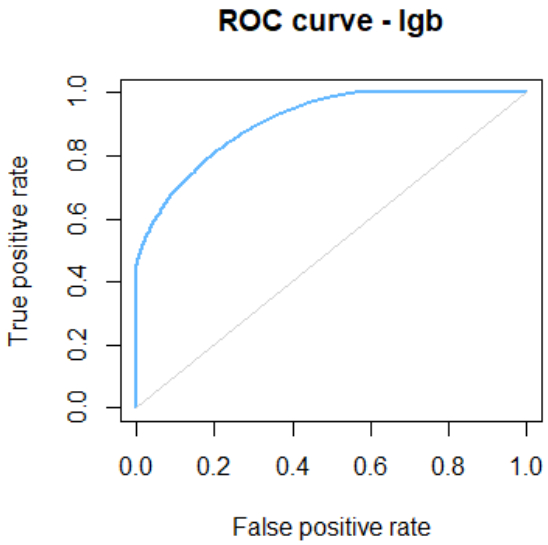

Figure 6: ROC curve optimal model (lgb), corresponding to an AUC score of 0.9085.

The confusion matrix of the most likely outcome (table 2), a more relevant table in the context of this project (table 3) and a table for a context where an extreme high cost for False Negatives exists (table 4) were created. By changing the threshold, the false positive and false negative rates can be balanced in order to optimise for the business setting at hand.

	True 0	True 1
Predicted 0	10761	1686
Predicted 1	110	1633

Table 2: Confusion matrix of optimal glb model with threshold of 0.5

	True 0	True 1
Predicted 0	8089	476
Predicted 1	2782	2843

Table 3: Confusion matrix of optimal glb model with threshold of 0.2

	True 0	True 1
Predicted 0	4812	2
Predicted 1	6059	3317

Table 4: Confusion matrix of optimal glb model with threshold of 0.05

It is worth mentioning that the accuracy drops from 0.8734 to 0.7704 and even further to 0.5729 when lowering the threshold from 0.5 to 0.2 to 0.05. However it is very likely that

the value of the model (in contrast to the accuracy) in this specific business context will increase. The code on which the analysis in this section is based upon can be found in the R file *Deployment_and_metrics.R*.

5 Reflection

During our modelling phase, we noticed that logistic regression explains a lot. This is in line with Occam's Razor. The rough translation of this principle is *"the simplest explanation is usually the best one"* (Howard, 2012). We also chose to use LGB instead of creating a neural network to keep our modeling as simple as possible.

Our number of submissions is rather low compared to other groups. There was a strong reasoning behind this. We internally evaluated our progress diligently before submitting. Also the concern of potentially over-fitting on the available test set made us reluctant to submit a lot of times.

There are a lot of possible additions for the future. A lot of potential methods haven't been explored. A good example would be exploring a neural network. The light gradient boosting could have also been tuned more precisely by cross validation further improving our final model. The data pre-processing could also be improved further. An example is that we noticed that in the addresses a lot of nature elements were listed. Grouping these with NLP could potentially add an interesting feature.

In summary, we believe that we approached the kaggle competition from a realistic perspective with a practical approach. We made sure that we thought through our method before submitting. This in turn also prevented us from over-fitting. As a result we hope that our score on the hidden leaderboard will also be quite high.

References

Bahmani, M. (2021). Understanding LightGBM Parameters (and How to Tune Them). https://neptune.ai/blog/lightgbm-parameters-guide

Breiman, L. (2002). "Manual On Setting Up, Using, And Understanding Random Forests V3.1". https://www.stat.berkeley.edu/~breiman/Using_random_forests_V3.1.pdf

Couronné, R., Probst, P., & Boulesteix, A. L. (2018). Random forest versus logistic regression: A large-scale benchmark experiment. *BMC Bioinformatics*, *19*(1), 1–14. https://doi.org/10.1186/s12859-018-2264-5

Howard, R. W. (2012). Selecting the better model by using data, occam's razor, and a little common sense: Reply to ericsson. *Applied Cognitive Psychology*, *26*(4), 654–656. https://doi.org/10.1002/acp.2836

James, G., Witten, D., Hastie, T., & Tibshirani, R. (2021). *An Introduction to Statistical Learning with application in R* (second edition).

Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., Ye, Q., & Liu, T. Y. (2017). LightGBM: A highly efficient gradient boosting decision tree. *Advances in Neural Information Processing Systems*, *2017-December*(Nips), 3147–3155.

Lynam, A. L., Dennis, J. M., Owen, K. R., Oram, R. A., Jones, A. G., Shields, B. M., & Ferrat, L. A. (2020). Logistic regression has similar performance to optimised machine learning algorithms in a clinical setting: Application to the discrimination between type 1 and type 2 diabetes in young adults. *Diagnostic and Prognostic Research*, *4*(1), 6. https://doi.org/10.1186/s41512-020-00075-2

Pushkar, M. (2017). What is LightGBM, How to implement it? How to fine tune the parameters? https://medium.com/@pushkarmandot/https-medium-com-pushkarmandot-what-is-lightgbm-how-to-implement-it-how-to-fine-tune-the-parameters-60347819b7fc

Appendices

A Selected Features

debt_to_income	address_postal_00813
emp_length	address_postal_05113
grade	**address_postal_11650**
income_verif_status	**address_postal_22690**
interest_rate	address_postal_29597
num_bankrupts	**address_postal_30723**
num_mortgages	**address_postal_48052**
num_open_credit	**address_postal_70466**
num_records	**address_postal_86630**
purpose	**address_postal_93700**
revol_balance	application_type_DIRECT_PAY
revol_util	**application_type_INDIVIDUAL**
term	application_type_JOINT
annual_income_flag	**date_funded_year**
emp_length_flag	**earliest_cr_line_year**
emp_title_flag	home_status_MORTGAGE
home_status_flag	home_status_OTHER
monthly_payment_flag	home_status_OWN
num_bankrupts_flag	**home_status_RENT**
num_mortgages_flag	**sqrt_amount**
num_records_flag	**log_annual_income**
num_total_credit_flag	sqrt_monthly_payment
revol_util_flag	sqrt_num_total_credit

Table 5: Encoded features before and after (**highlighted**) feature selection

B Coefficients logistic regression

```
Coefficients:
                            Estimate Std. Error z value Pr(>|z|)
(Intercept)                -19.60347   75.98773  -0.258 0.796421
debt_to_income               0.90133    0.08261  10.910  < 2e-16 ***
grade                       -1.75050    0.07545 -23.202  < 2e-16 ***
income_verif_status          0.12999    0.03449   3.769 0.000164 ***
num_bankrupts               -0.18333    0.06367  -2.880 0.003983 **
num_mortgages               -0.44715    0.06912  -6.469 9.88e-11 ***
num_open_credit              0.34329    0.08633   3.976 7.00e-05 ***
num_records                  0.32064    0.09155   3.502 0.000461 ***
revol_balance               -0.37816    0.10777  -3.509 0.000450 ***
revol_util                   0.52341    0.08786   5.957 2.57e-09 ***
term                        -0.40456    0.03504 -11.547  < 2e-16 ***
emp_title_flag               0.51568    0.07764   6.642 3.10e-11 ***
num_mortgages_flag          -0.19649    0.07530  -2.610 0.009067 **
num_total_credit_flag        0.26433    0.15304   1.727 0.084122 .
address_postal_11650        38.99745  252.21081   0.155 0.877119
address_postal_22690        18.28342   75.98692   0.241 0.809855
address_postal_30723        18.31203   75.98692   0.241 0.809563
address_postal_48052        18.35127   75.98692   0.242 0.809163
address_postal_70466        18.32888   75.98692   0.241 0.809392
address_postal_86630        38.98078  249.06552   0.157 0.875633
address_postal_93700        39.01284  248.58431   0.157 0.875292
application_type_INDIVIDUAL  0.90276    0.29004   3.113 0.001855 **
date_funded_year             0.38764    0.11934   3.248 0.001162 **
earliest_cr_line_year        0.24595    0.13508   1.821 0.068638 .
home_status_RENT             0.16695    0.03132   5.331 9.75e-08 ***
sqrt_amount                  0.74053    0.09271   7.987 1.38e-15 ***
log_annual_income           -1.68631    0.15383 -10.962  < 2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

Figure 7: Coefficients final logistic regression model

C Parameters

C.1 Random forest parameters

Table 6: Parameter settings of the random forest

Name of parameter	Value
Mtry	12
node_size	37

C.2 Light Gradient Boosting Parameters

Table 7: Parameter settings of light gradient boosting

Name of parameter	Value
Objective	Binary
num_leaves	10
max_depth	12
metric	AUC
boosting	gbdt
learning_rate	0.005
num_iterations	2950
min_data_in_leaf	40