Kaggle: Home Default Risk Competition
Classification of Bad Accounts in Credit Card Industry

Jijun Du
Instructor: Farid Alizadeh

September 30, 2018

1.Problem & Datasets Description

1.1 Problem

Home Credit is an institution that , and eager to use historical data with machine learning methods to make these predictions. clients capable of repayment are not rejected and that loans are given with a principal, maturity, and repayment.

1.2 Dataset

Total datasets add up to 2.68 GB. Data is publicly available at Kaggle- Home Default Risk competition home page. There are eight csv files from three sources: loan applicants previous and current data, applicants data that report to bureau, clients account balance includes prior installments loans and credit card loans. Dataset relationship shows within the following entity relationship diagram.

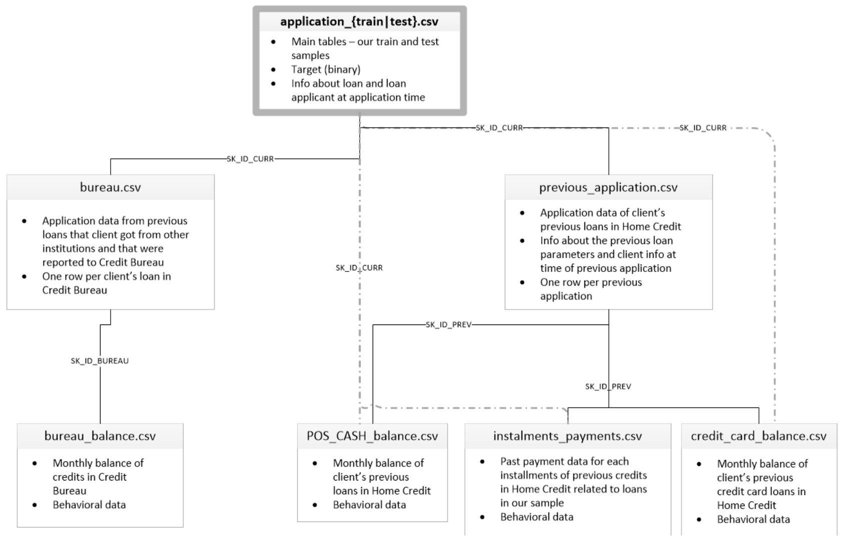

application_{train|test}.csv
Main tables – our train and test samples
Target (binary)
Info about loan and loan applicant at application time
SK_ID_CURR
SK_ID_CURR
SK_ID_CURR
SK_ID_CURR
bureau.csv
Application data from previous loans that client got from other institutions and that were reported to Credit 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. Relational schema of HDR competition. Provided by Kaggle, url: https://www.kaggle.com/c/home-credit-default-risk/data

1.3 Resources:

Learning resource: relevant Rutgers courses, open source machine learning package in Python & R, "How to Win a data science competition" from Coursera, specialists blogs.

Kaggle resource: other entrants' kernel for visualization purpose, discussion forum, online GPU with limited access time.

Discussion forums: Since Kaggle competition have great discussion forums, I learned and adapt Credit card company's suggestion in field knowledge. Mostly from user Anh, a former Home Credit Vietnam senior financial analyst & product manager.

Such as:

- HC mainly does business in CIS and SEA countries, data might be combined from data of Kaz, Russia, Vietnam, China, Indonesia, Phillipines. Therefore, findings on FICO.com from US is really not useful
- Bureau data is usually not sufficient for credit scoring loan application here, as people have low exposure to banking.
- Current loan & previous loan are more reliable data for scoring

2. Idea Generation & Preparation

2.1 Flow chart of strategy

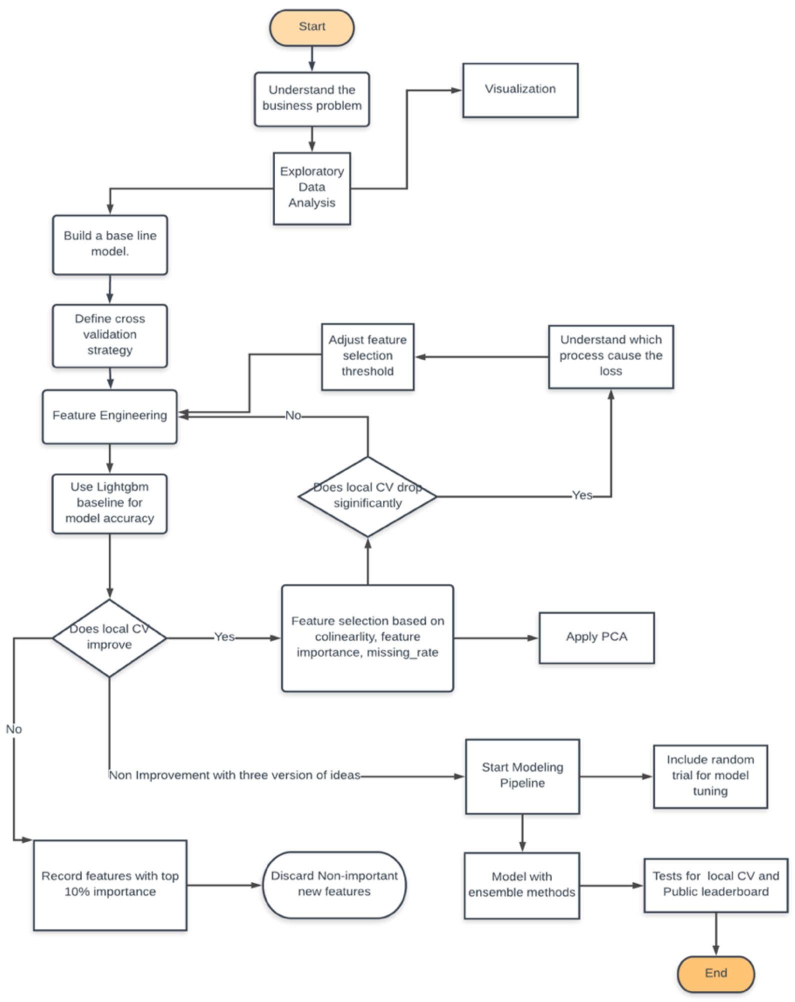

Start
Understand the business problem
Visualization
Exploratory Data Analysis
Build a base line model.
Define cross validation strategy
Adjust feature selection threshold
Understand which process cause the loss
Feature Engineering
No
Does local CV drop significantly
Yes
Use Lightgbm baseline for model accuracy
Does local CV improve
Yes
Feature selection based on colinearlity, feature importance, missing_rate
Apply PCA
No
Non Improvement with three version of ideas
Start Modeling Pipeline
Include random trial for model tuning
Record features with top 10% importance
Discard Non-important new features
Model with ensemble methods
Tests for local CV and Public leaderboard
End

Fig. 2. My workflow for this competition

In terms of competition strategy, I distribute my time based on 80/20 rule. For methods in core stages, preprocessing, feature engineering, and modeling, I record the reasoning, time consumption, and improvements.

In order to save computational time, I turned on debug mode with a smaller sample of data purpose.

Instead of learning the complex visualization techniques with Python or R, I read other competitors' visualization files and use existing tools such as Tableau & Orange (similar to SPSS) for idea generation.

The flow chart is my final one, changed over time based on my understanding to machine learning and strategy for the competition.

2.2 Strategies & Baseline model set up

Data Exploration: Used Orange (toolbox for machine learning and visualization) for subset data, other competitors' instructional kernel files.

Random Sampling: I chose random sampling dataset of 3~5%, limit 7 sub files with size 1.7~8.3 MB. The sample datasets maintain the statistical significance, and are easy for debug in feature engineering process, constructing baseline model, and feed in Orange.

Baseline model: a simple program contains read in datasets, transforms data to feed in models, builds lightgbm model with default parameters, and generate output file for test dataset ["SK_ID_CURR"] and it's corresponding probability ['Target'].

ROC Evaluation: ROC curve is created by plotting the true positive rate (TPR) against the false positive rate (FPR) at various threshold settings.

Cross validation strategy: 5 fold or 10 fold based on the size of datasets.

2.3 Exploration with Orange

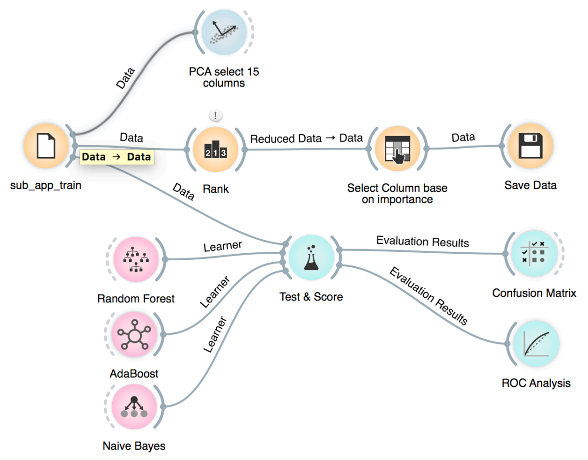


Fig. 3. Orange pipeline

- PCA for sub dataset
- Rank feature importance for top 15
- Set three baseline models
- Evaluation based on selected models.

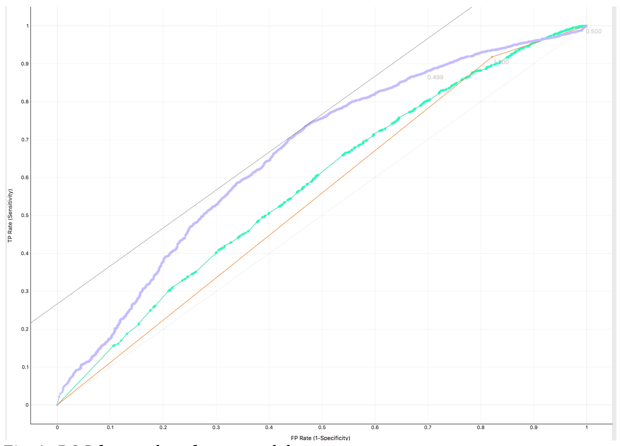
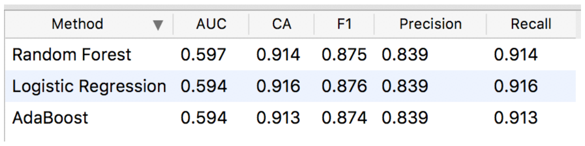


Fig. 4. ROC for random forest model



Method	AUC	CA	F1	Precision	Recall
Random Forest	0.597	0.914	0.875	0.839	0.914
Logistic Regression	0.594	0.916	0.876	0.839	0.916
AdaBoost	0.594	0.913	0.874	0.839	0.913

Fig. 5. Evaluation for baseline mode

3. Feature preprocessing and engineering:

"Data and features determine the upper limit of machine learning, while models and algorithms just approximate this upper limit."

3.1 Preprocessing: Exploratory Data Analysis

3.2 Data pattern distribution:

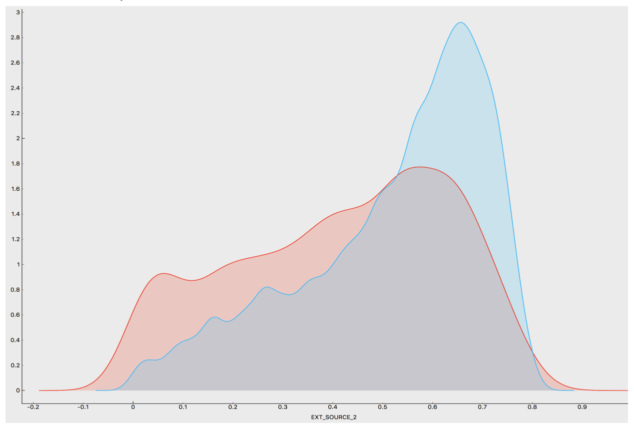


Fig. 6. Unlabeled & significant feature ['external_source_2'] in application_train.csv

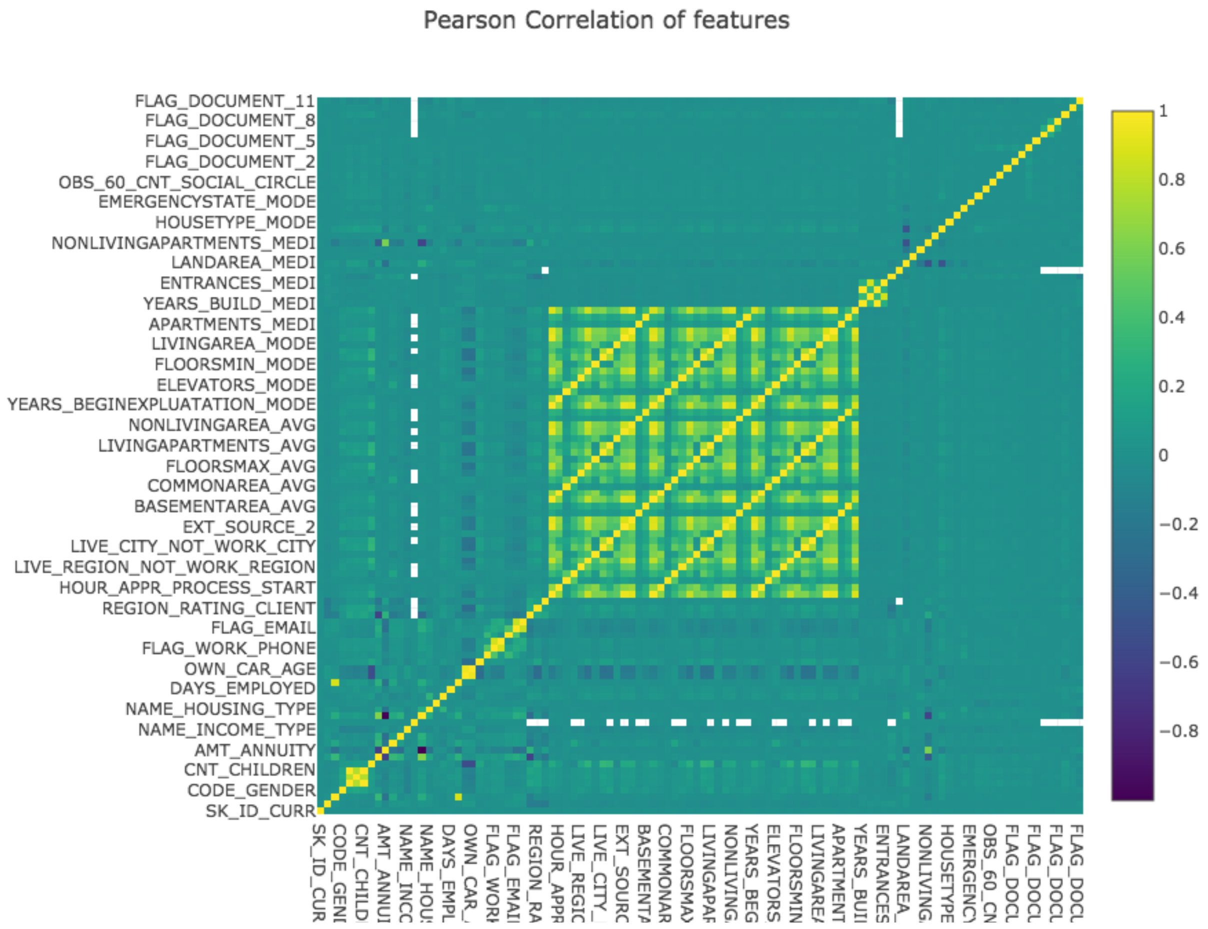
Research on sub datasets help me to understand categorical features meaning and types, and distribution frequency of numerical data.

'0' for credit repaid, '1' for not repaid.

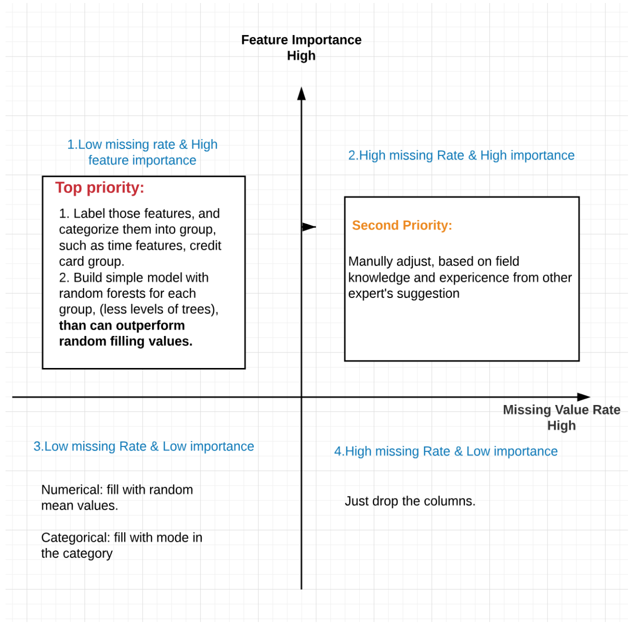
The process includes data distribution by Orange and question driven exploration by Jupyter notebook.

3.3 Explore dataset with specific purposes

Pearson Correlation of features



3.4 Missing Value imputation:


Feature Importance
High
Missing Value Rate
High
1.Low missing rate & High feature importance
2.High missing Rate & High importance
3.Low missing Rate & Low importance
4.High missing Rate & Low importance

Top priority:

1. Label those features, and categorize them into group, such as time features, credit card group.
2. Build simple model with random forests for each group, (less levels of trees), **than can outperform random filling values.**

Second Priority:

Manully adjust, based on field knowledge and expericence from other expert's suggestion

Numerical: fill with random mean values.

Categorical: fill with mode in the category

Just drop the columns.

Fig. 9. My data cleaning four quadrants

To deal with feature engineering in a timely manner, I divided data into four category, two aspects. X axis for missing value rate, Y axis for feature importance achieve from. Each threshold is tested and adjusted with trails.

Steps to

1. Read expertise field experience, reduce unnecessary workload.
2. Separate data-frame into numerical and categorical. Label their columns.
3. Compute percentage of missing value, set missing value threshold ranging from 0.65~0.75, feature importance threshold > 0.15. (Based on column #)

I apply similar methods to all seven relational data table. Description is based on application_train.csv .

Imputation for Type I , Type II data:
Type I: Applying random forest classifier to application's specific columns. Since it's easily to implement with less computation required. For example, the regressor has layer of 3 and impute the 30 percent of missing value.
Type II: Applying missing value imputation for a based-line model, improved score from 0.753-0.759. With principal component analysis (PCA), I know the feature importance and relationship for all variables. The output files are come from Python (mice) package, 1. feature importance matrix, 2. Principle components values. Therefore, I reduced 130 features into 15 variables. The correlation result shows that feature ["external_source_1~3"] ["payment_rate"] has most feature importance. I choose top 10 variables with high correlation to implement.

3.5 Features engineering

My feature engineering process includes feature encoding, feature selection, feature construction. Encoding: for categorical data, I used label encoding if (distinct categorical values of a variable <=5) and treat the rest with one-hot encoding. Then, transform numerical data with min-max.

Construction:
Selection:

4. Modeling

4.1 Model Selection:

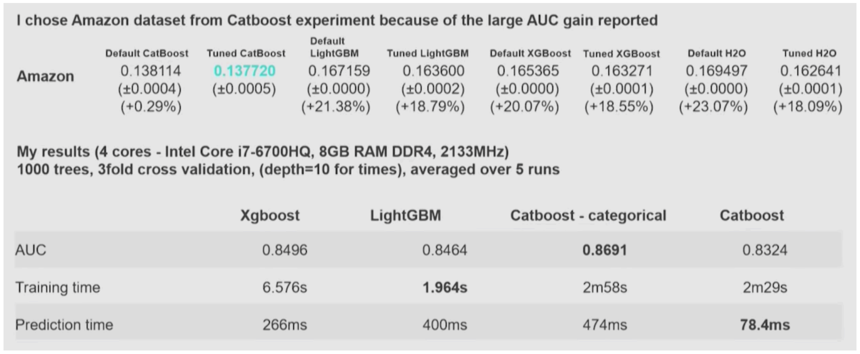
For choosing models in the whole process, I emphasize on model accuracy achieved, tolerance to missing value, computational speed (training and tuning model). At last, I select two groups of models. Random forest for missing value imputation, and lightgbm, Xgboost for stacking models for general prediction.

4.2 Random Forest

Random forest is a stacking method with collection of decision trees, it is easy to apply and explain. There are three parameters we need to tune for RF: ntree (controls how many trees) and mtry. Ntree controls how many trees to grow and mtry controls how many variables to draw each time. For missing value imputation, I set ntree to 100 and mtry = 3 for random forest regressor.

4.3 Lightgbm

Unlike most boosting methods (random forest,) using pre-sort-based algorithms, Lightgbm use histogram-based algorithm to increase efficiency. During this competition, it's the most reliable model I used for most of the times. After retrieving an ideal sets of model parameters, I only make improvement on feature engineering side. Since lightgbm's logic
For each fold of 5-fold validation method, record time spent on training, best iteration's AUC score and its parameter. Passing indexes of categorical feature.



I chose Amazon dataset from Catboost experiment because of the large AUC gain reported

	Default CatBoost	Tuned CatBoost	Default LightGBM	Tuned LightGBM	Default XGBoost	Tuned XGBoost	Default H2O	Tuned H2O
Amazon	0.138114 (±0.0004) (+0.29%)	0.137720 (±0.0005)	0.167159 (±0.0000) (+21.38%)	0.163600 (±0.0002) (+18.79%)	0.165365 (±0.0000) (+20.07%)	0.163271 (±0.0001) (+18.55%)	0.169497 (±0.0000) (+23.07%)	0.162641 (±0.0001) (+18.09%)

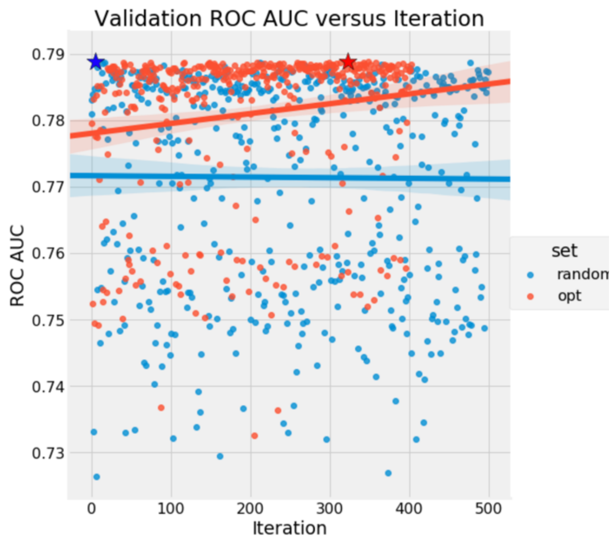
My results (4 cores - Intel Core i7-6700HQ, 8GB RAM DDR4, 2133MHz)
1000 trees, 3fold cross validation, (depth=10 for times), averaged over 5 runs

	Xgboost	LightGBM	Catboost - categorical	Catboost
AUC	0.8496	0.8464	**0.8691**	0.8324
Training time	6.576s	**1.964s**	2m58s	2m29s
Prediction time	266ms	400ms	474ms	**78.4ms**

Fig. 10. Mateusz Susik's comparison for benchmark model performance from McKinsey & Company

4.4 Xgboost

4.5 Model Parameter Tuning



Among model selection methods, I choose grid search, random search, Bayesian optimization with sample dataset. Grid search requires a decent understanding of parameter's range, a broad range would result intensive calculation that last for hours. From my baseline model, Bayesian optimization outperforms random search in both accuracy and time spent on finding the best iteration. For subsequent model training and tuning parameters, I only use Bayesian optimization.

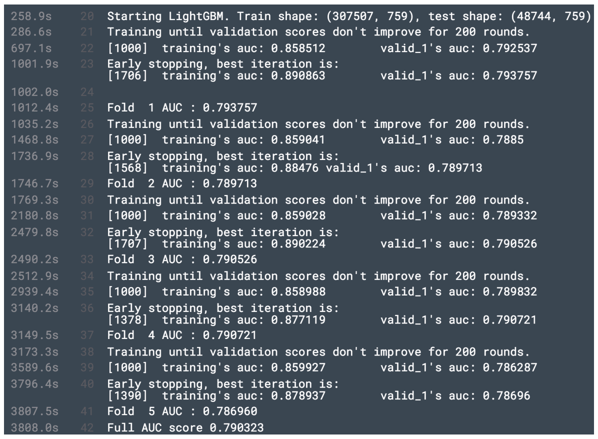
Method	Cross Validation Score	Test Score (on 6000 Rows)	Submission to Leaderboard	Iterations to best score
Random Search	0.73110	0.73274	0.782	996
Bayesian Hyperparameter Optimization	0.73448	0.73069	0.792	596

4.6 Model pipeline

A pipeline method helps to make minimum change in code and show modeling results easily. Each previous pipeline method chunk's output is the current chunk's input; besides, modeling process can be simplified to pipeline. The process is down by separating each process into code chunk, that feeds in the next.

Improvements on existing pipeline:

Each chunk is combined with a timer, so I can understand each step in this process take. For modeling speed efficiency, I implement debug mode into the pipeline. When set debug in pipeline method as true, read in only minimum of (8% of rows, 1000 rows) of the data in 6 relational tables. The complexity for model tuning can reduce significantly from 10-hour preprocessing time to 7 minutes.



```
258.9s     Starting LightGBM. Train shape: (307507, 759), test shape: (48744, 759)
286.6s     Training until validation scores don't improve for 200 rounds.
697.1s     [1000]  training's auc: 0.858512        valid_1's auc: 0.792537
1001.9s    Early stopping, best iteration is:
           [1706]  training's auc: 0.890863        valid_1's auc: 0.793757
1002.0s
1012.4s    Fold  1 AUC : 0.793757
1035.2s    Training until validation scores don't improve for 200 rounds.
1468.8s    [1000]  training's auc: 0.859041        valid_1's auc: 0.7885
1736.9s    Early stopping, best iteration is:
           [1568]  training's auc: 0.88476 valid_1's auc: 0.789713
1746.7s    Fold  2 AUC : 0.789713
1769.3s    Training until validation scores don't improve for 200 rounds.
2180.8s    [1000]  training's auc: 0.859028        valid_1's auc: 0.789332
2479.8s    Early stopping, best iteration is:
           [1707]  training's auc: 0.890224        valid_1's auc: 0.790526
2490.2s    Fold  3 AUC : 0.790526
2512.9s    Training until validation scores don't improve for 200 rounds.
2939.4s    [1000]  training's auc: 0.858988        valid_1's auc: 0.789832
3140.2s    Early stopping, best iteration is:
           [1378]  training's auc: 0.877119        valid_1's auc: 0.790721
3149.5s    Fold  4 AUC : 0.790721
3173.3s    Training until validation scores don't improve for 200 rounds.
3589.6s    [1000]  training's auc: 0.859927        valid_1's auc: 0.786287
3796.4s    Early stopping, best iteration is:
           [1390]  training's auc: 0.878937        valid_1's auc: 0.78696
3807.5s    Fold  5 AUC : 0.786960
3808.0s    Full AUC score 0.790323
```

Fig. 10. Full model training Results with run time

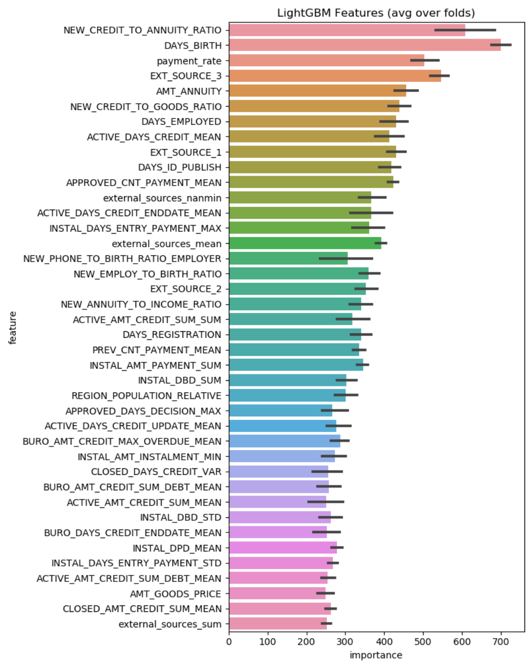


Fig. 11. Feature Importance for lightgbm model

5. Evaluation

5.1 Local cross validation

Prevent overfitting with datasets.

5.2 Public leaderboard

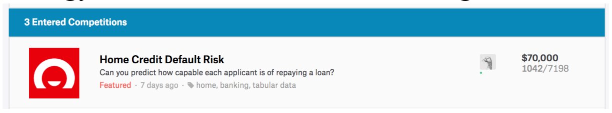
Record new process and take screenshots of scores and place I get.

5.3 AUC score under ROC curve

6. Conclusion and Future

Among these models, Lightgbm performs the best. With ensemble model blend in Xgboost, it out performs all three models.

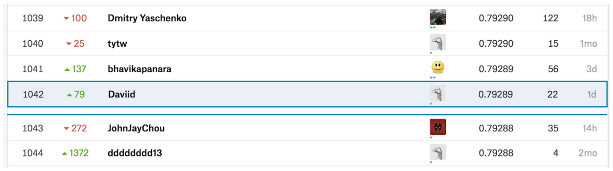
Since "Home default risk" is my first featured competition, I learn the whole cycle of the machine learning competition for the credit card risk, understands my deficiency and weakness (limited field knowledge, this time I relied on public post). Besides, I get familiar with the resources stored in Kaggle, experts' (1st,2nd,3rd place winner) idea to solve the problem. Next time, I could decide the cooperate strategy and find teammates to work together.



3 Entered Competitions

Home Credit Default Risk
Can you predict how capable each applicant is of repaying a loan?
Featured · 7 days ago · home, banking, tabular data

$70,000
1042/7198



1039	▾100	Dmitry Yaschenko	0.79290	122	18h
1040	▾25	tytw	0.79290	15	1mo
1041	▴137	bhavikapanara	0.79289	56	3d
1042	▴79	Daviid	0.79289	22	1d
1043	▾272	JohnJayChou	0.79288	35	14h
1044	▴1372	dddddddd13	0.79288	4	2mo

Fig. 15. Ranking in Kaggle

Further improvement on this competition:

- Data visualization leads to modeling ideas.
- Missing value imputation methods.
- Add feature extraction process.
- Looking for data leakage
- Try stacking layers of models
- Understand into TensorFlow, neural network.

REFERENCES

[1] Bradley, A. P. (1997). The use of the area under the ROC curve in the evaluation of machine learning algorithms. Pattern recognition, 30(7), 1145-1159.

[2] James, G., Witten, D., Hastie, T., & Tibshirani, R. (2013). An introduction to statistical learning (Vol. 112). New York: springer.

[3] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... & Vanderplas, J. (2011). Scikit-learn: Machine learning in Python. Journal of machine learning research, 12(Oct), 2825-2830.

[4] Chen, T., & Guestrin, C. (2016, August). Xgboost: A scalable tree boosting system. In Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining (pp. 785-794). ACM.

[5] Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., ... & Liu, T. Y. (2017). Lightgbm: A highly efficient gradient boosting decision tree. In Advances in Neural Information Processing Systems (pp. 3146-3154).

[6] Microsoft Team (2018). Available at: https://docs.microsoft.com/en-us/azure/machine-learning/studio/algorithm-choice

[7] Khandani, A. E., Kim, A. J., & Lo, A. W. (2010). Consumer credit-risk models via machine-learning algorithms. Journal of Banking & Finance, 34(11), 2767-2787.

[8] Orange team blog. Available at https://blog.biolab.si/tag/data-mining/