Elo- Merchant Category Recommendation

Kaggle competition

Kai Mo
School of Computing and information
University of Pittsburgh
Pittsburgh Pennsylvania USA
kam455@pitt.edu

Tianqi Xie
School of Computing and information
University of Pittsburgh
Pittsburgh Pennsylvania USA
tix19@pitt.edu

ABSTRACT
This article describes the methods that we have tried to solve the ELO-Merchant Category recommendation problem on Kaggle. The main objection of this competition is to find a way to recommend merchant categories to different cardholder based on their purchase behavior. We did data processing based on feature engineering and feature selection. In addition, we compared two the-state-of-art gradient boosting decision tree: LightGBM and CatBoost.

INTRODUCTION
Laying advertisement precisely is a popular advertising method in the internet advertising industry, which is different from the traditional advertising industry that has time, region and choice of audience limitation. The Elo company wants to take this advertising method to the next level. Instead lay different ads for different people group, the Elo company wants to lay different ads for each individual based on their consuming style. In other word, we have to put label on each cardholder other than put labels on different groups of people. This extreme precisely advertising method not only help customers to improve their consuming experience, but also help merchants to acknowledge their customer loyalty better.

In this paper, we describe and analysis the challenges for Elo company's merchant category recommendation datasets from the Kaggle competition. We faced challenges due to large datasets that include incomplete features, which are also anonymized, the special formatted data, and so on. We used some supervised machine learning algorithms to solve these problems. Our dedication on this project could be concluded as following:

- We extended the feature space, from low dimension feature space to high dimension space, based on existed features. We created new features to capture the potential patterns to improve our model performance.
- We applied embedded feature elimination on datasets to help us reduce feature dimensionality.
- We learned from Bayesian optimization to tune the hyperparameters of the models we selected.
- We compared the performance of LightGBM and CatBoost models on our project.

CCS CONCEPTS
• Feature Engineering • Feature Selection • Data Processing • LightGBM • CatBoost

KEYWORDS

• LightGBM • CatBoost

1. Detailed Problem Description

The Kaggle competition we joined is Elo merchant category recommendation. The Elo is a payment brands in Brazil and the company has already grouped the cardholders according to their important aspects and preferences of consuming styles. The problem Elo faces right now is that the company wants to offer cardholders deals and promotions based on their personal lifestyle instead offer the deals and promotion based on group features. We need to analyze the data Elo provided and predict cardholders' loyalty scores to help the company to create better promotion target strategy for each customer.

2. Related Work

In recent years, ensemble learning methods like boosting and bagging outperform others machine learning methods in many competitions and datasets, shows its outstanding performance and generalized capability. Boosting is a method that resamples or reweights samples in each training stage in order to correct current model prediction error. In another word, it integrates a set of weak estimators' outputs to produce better performance than a single weak estimator. A popular branch of boosting method is Gradient Boosting Decision Tree (GBDT), which boosts a set of decision tree models and use gradient boosting instead of adaptive boosting (J. Friedman, Hastie, & Tibshirani, 2000; J. H. Friedman, 2001, 2002).

There are three most successful state-of-the-art implementations of GBDT, i.e., XGBoost, LightBGM and CatBoost. XGBoost is an ensemble machine learning algorithm have been dominating applied machine learning and Kaggle competitions due to its speed and performance. The main idea of XGBoost is identical with GBDT which use gradient boosting for correcting current stage error(Chen & Guestrin, 2016). LightBGM, released by Microsoft which is claimed to be more efficient than XGBoost, is also an implementation of GBDT which is similar with XGBoost, but it has several difference with XGBoost, the main difference is that LightGBM uses Gradient-based One-Side Sampling(GOSS) to filter out the data instances to find a split value while XGBoost uses pre-sorted algorithm and Histogram-based algorithm for finding the best split(Ke et al., 2017). A new popular GBDT model named CatBoost, achieves the best results on the benchmark, which significantly improve the performance on those datasets with categorical features. There are two main ideas introduced by CatBoost, using a permutation-driven ordered boosting algorithm to construct tree structure and innovative methods to handle categorical features – ordered target statistic(Dorogush, Ershov, & Gulin, 2018; Prokhorenkova, Gusev, Vorobev, Dorogush, & Gulin, 2018).

3. Methodology

3.1 Data description

During the competition, the dataset was split into public and hidden parts. Public part was used for model training and evaluation while final models were evaluated on private test set, named hidden part. The public part has seven different tables, but we only retrieved useful information from three tables i.e. train.csv, historical_transaction.csv, new_merchant_transaction.csv. The information of these tables shows below.

Columns	Description	Type
card_id	Unique card identifier	String
first_active_month	'YYYY-MM', month of first purchase	String
feature_1	Anonymized card categorical feature	Category
feature_2	Anonymized card categorical feature	Category
feature_3	Anonymized card categorical feature	Category
target	Loyalty numerical score calculated 2 months after historical and evaluation period	Numeric

Table1: Train.csv (with target feature): 0.2 million rows

Columns	Description	Type
card_id	Card identifier	String
month_lag	month lag to reference date	Numeric
purchase_date	Purchase date	String
authorized_flag	Y' if approved, 'N' if denied	Category
category_3	anonymized category	Category
installments	number of installments of purchase	Numeric
category_1	anonymized category	Category
merchant_category_id	Merchant category identifier (anonymized)	String
subsector_id	Merchant category group identifier (anonymized)	String
merchant_id	Merchant identifier (anonymized)	String
purchase_amount	Normalized purchase amount	Numeric
city_id	City identifier (anonymized)	String
state_id	State identifier (anonymized)	Numeric
category_2	anonymized category	Category

Table 2: Historical_transaction.csv: 29.11 million rows

Columns	Description	Type
card_id	Card identifier	String
month_lag	month lag to reference date	Numeric
purchase_date	Purchase date	String
authorized_flag	Y' if approved, 'N' if denied	Category
category_3	anonymized category	Category
installments	number of installments of purchase	Numeric
category_1	anonymized category	Category
merchant_category_id	Merchant category identifier (anonymized)	String
subsector_id	Merchant category group identifier (anonymized)	String
merchant_id	Merchant identifier (anonymized)	String
purchase_amount	Normalized purchase amount	Numeric
city_id	City identifier (anonymized)	String
state_id	State identifier (anonymized)	Numeric
category_2	anonymized category	Category

Table 3: New_merchant_transaction: 1.96 million rows

3.2 feature engineering

As we can see from the data description section, some features like First_active_month and purchase_date are in the date format. However, we need to calculate the data for future usage. In order to tackle this problem, we introduced a new feature called reference_date, which is the date of purchase_date minus month_lag. Then, we use Month_diff feature to present the time information. The month_diff equals to the current_data minus reference_date, which is a number. For categorical features, we used one-hot-encoding method to process them.

Here is a problem that the features we got are not enough, so we decided to expend our feature space by creating new features based on what we have now. First of all, we calculated the statistics values of each feature, for example, we calculated the sum and mean of category_1. Then, we aggregate these features through group by card_id. Second, we calculate count, sum, mean, min, max and std of feature purchase_amount and installments, and then aggregate them group by card_id. Third, we aggregated all features' mean and std group by month and card_id.

3.3 feature selection

For the feature election or feature elimination part, we based on the model's in-build feature importance calculation method, then apply adjusted permutation feature importance (PIMP) method to correct the model calculated feature importance. For LightGBM in-build feature importance calculate methods we use split and gain feature importance, where split importance is calculated as numbers of times the feature is used in a model and gain importance calculated as total gains of splits which use the feature. For CatBoost we use PredictionValuesChange feature importance and the formula shows below:

$$feature\_importance = \sum_{trees,leafs} \left(v_1 - \frac{v_1 \cdot c_1 + v_2 \cdot c_2}{c_1 + c_2}\right)^2 c_1 + \left(v_2 - \frac{v_1 \cdot c_1 + v_2 \cdot c_2}{c_1 + c_2}\right)^2 c_2$$

The reason of applying PIMP is that, as the paper of Altmann mentioned, "complex model like random forest and gradient boosting decision tree are biased in such way that categorical variables with a large number of categories are preferred"(Altmann, Tolosi, Sander, & Lengauer, 2010). To tackle this problem, the author proposed a new method to normalizing feature importance which will eliminate the bias that we mentioned above. The main idea of our adjusted feature importance is we first repeatedly train the model many times with different permutation of target value which is called non-informative setting, then we collect the importance distribution calculated by model as *null importance*. Second, we train model on original target value without permutation and collect the actual feature importance. Finally, we divided actual feature importance by the 75th percentile value of *null importance* and take log function of the result as the PIMP. The formula shows below.

$$PIMP = \log \frac{actual\_importance}{75^{th} null\_importance} (1)$$

We calculated three PIMP based on LightGBM and CatBoost feature importance respectively.

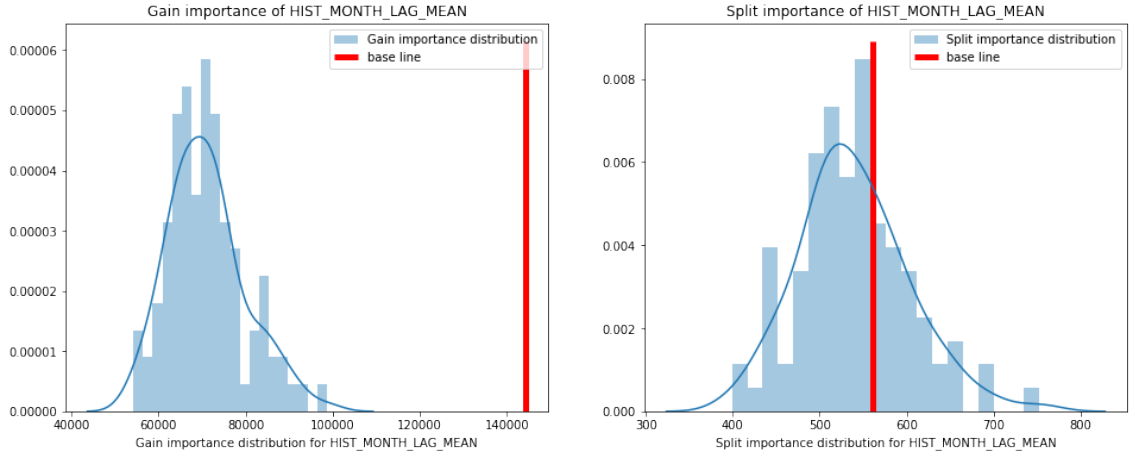

Figure 1. Null importance and actual importance of HIST_MONTH_LAG_MEAN of LightGBM

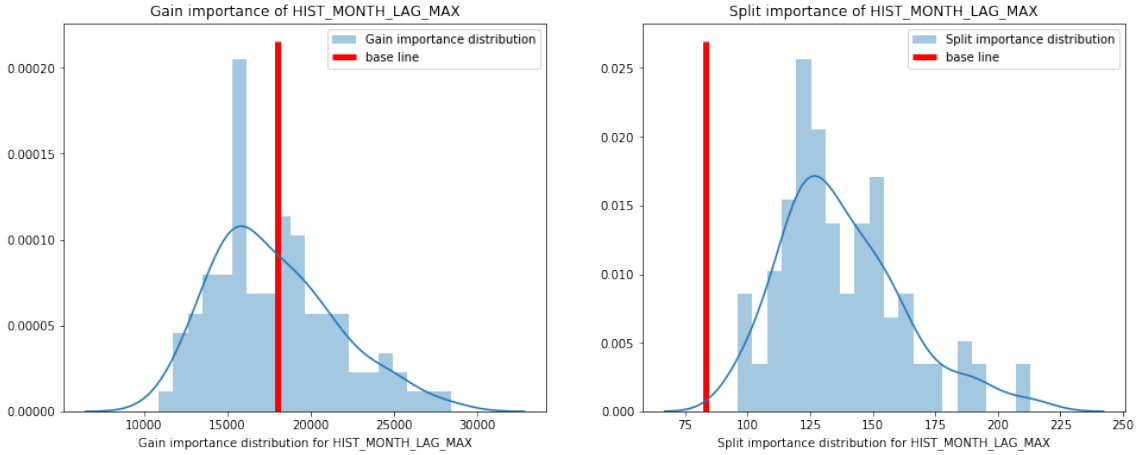

Figure 2. Null importance and actual importance of HIST_MONTH_LAG_MAX of LightGBM

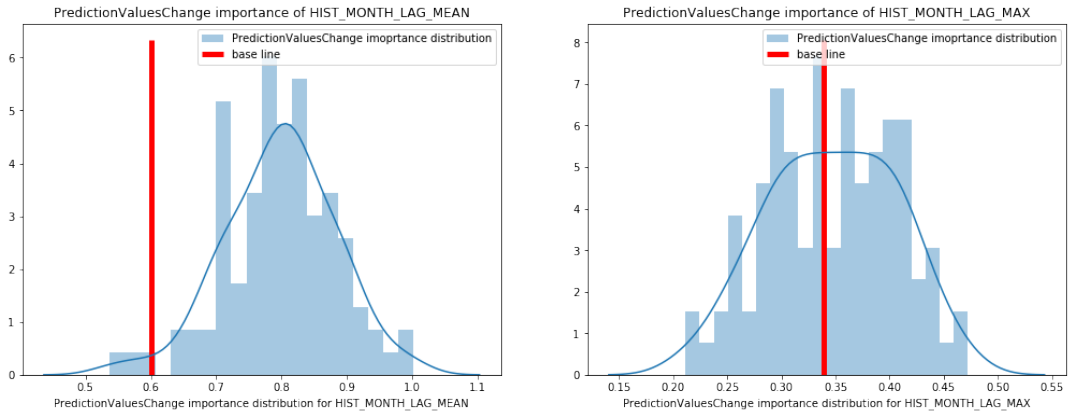

Figure 3. Null importance and actual importance of HIST_MONTH_LAG_MAX and HIST_MONTH_LAG_MEAN of CatBoost

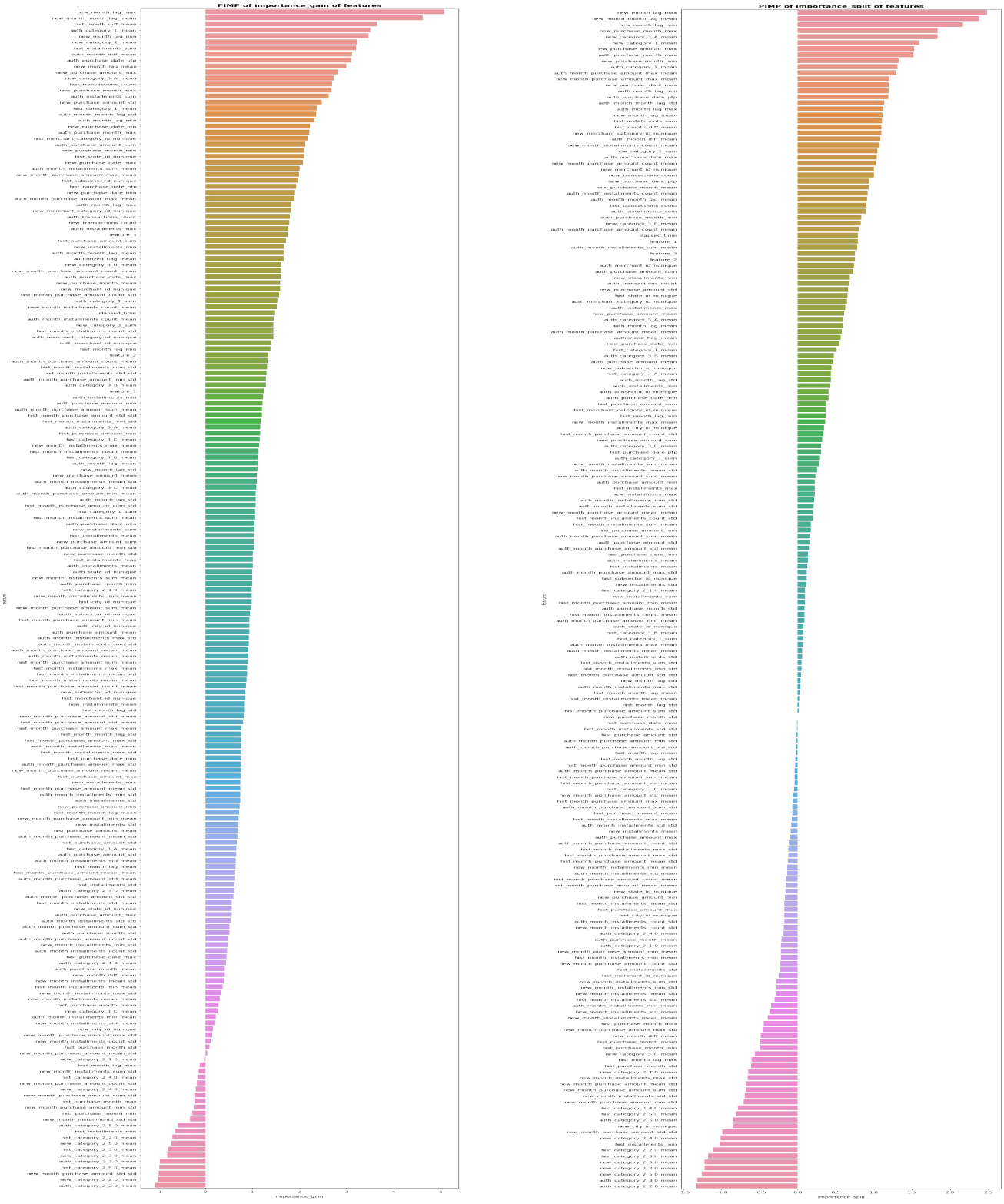

Figure 4. PIMP of split and gain importance of LightGBM

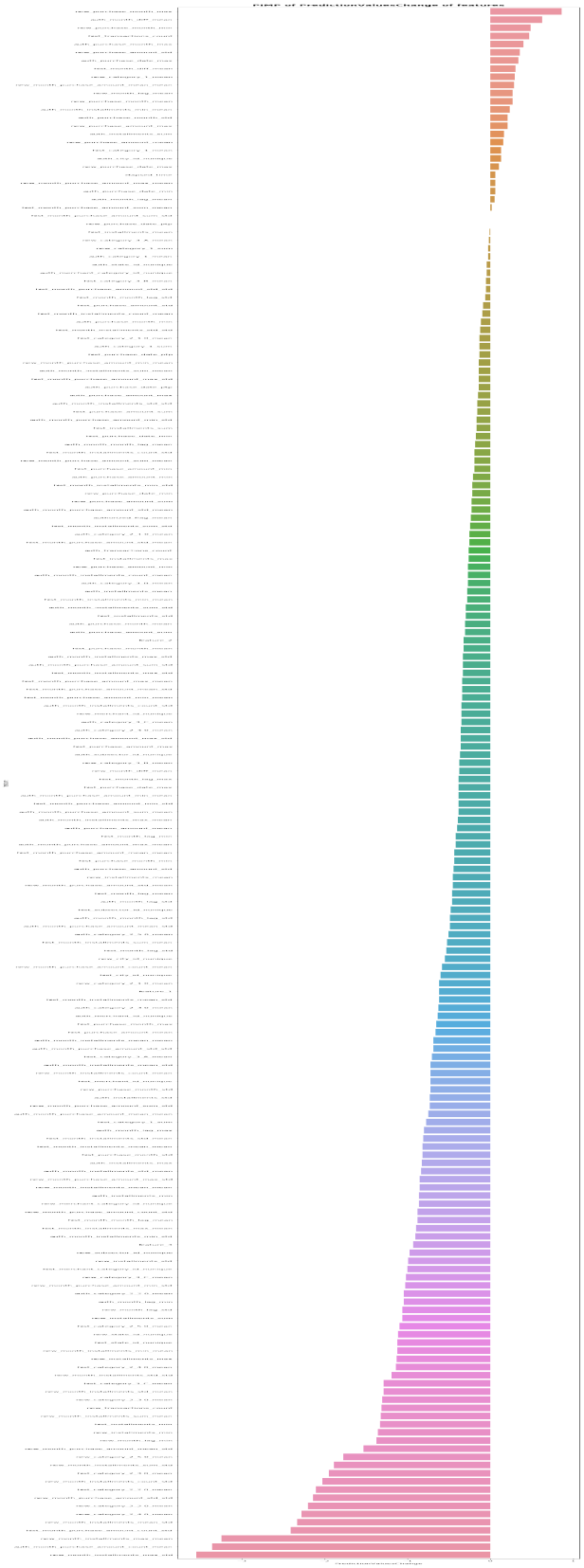

Figure 5. PIMP of importance of CatBoost

Figure 1. and figure 2. show the *null importance* of gain and split importance distribution of two features i.e. HIST MONTH LAG MEAN and HIST MONTH LAG MAX in LightGBM. As can be seen there, actual gain importance of HIST MONTH LAG MEAN is much larger than 75th *null importance*, which means that gain importance of HIST MONTH LAG MEAN is significant, in other word, the feature is important to target. On the contrary, the actual split importance of HIST MONTH LAG MAX is much less than 75th *null importance*, which means that the split importance of HIST MONTH LAG MAX is not significant, i.e. the feature is not importance. Figure 3. shows the *null importance* distribution and actual feature importance of HIST MONTH LAG MAX in CatBoost, which is similar with the later situation in LightGBM, the actual importance is much less than *null importance*.

Figure 4. and figure 5. show the PIMP ranking for LightGBM and CatBoost. In the two type of importance of LightGBM PIMP ranking plot, i.e. figure 4, feature's ranking is similar with the PIMP importance of CatBoost ranking plot. That is, the more important features have higher ranking in both three PIMP plots.

3.4 hyperparameter optimization

As we all known, hyperparameter optimization is an essential part in model tuning, which will diminish model performance without properly setting, otherwise, improve model performance. Unfortunately, hyperparameter optimization is a "black art", in general, it needs expert experience or brute force search. For instance, grid search is a brute force method for finding the optimal combination of hyperparameters which minimizes loss function or maximizes object. At 2012, some researchers proposed a more efficient method named random search, but the computational complexity is still not affordable and it is time consuming (Bergstra & Bengio, 2012). After that, Bayesian optimization method was proposed by Snoek and Larochelle et. al(Snoek, Larochelle, & Adams, 2012). The main idea of this method is assuming the learning algorithm's generalization performance is based on Gaussian process priors. Bayesian optimization basically builds a surrogate for objective function, qualifies the uncertainty in the surrogate we mentioned before, and uses the surrogate's acquisition functions to make a decision on where to collect the samples. The Bayesian optimization has two major parts: a Bayesian statistical model and an acquisition function. The main reason that we used expected improvement is that it is the most popular acquisition function and it performs well and is easy to use (Frazier, 2018; Xia, Liu, Li, & Liu, 2017).

3.5 model training

As mentioned above, after preprocessing, feature engineering, feature selection and model hyperparameters tuning, we have all ingredients for training models, i.e. constructed features with PIMP ranking, models' hyperparameters, it is the time to feed these ingredients to the LightGBM and CatBoost models. For both of these two models we use Root Mean Squared Error (RMSE) metric to evaluate training and test performance.

4. Conclusion

The training and test result are showed in Table 5, we train these two models based on different number of most important features. E.g. for the first row of our table, we train LightGBM and CatBoost models using the first 60 features in PIMP ranking list. By comparing these results of two models, we can see that, when feed same number of features to LightGBM and CatBoost model, LightBGM outperforms CatBoost. Moreover, the best result of LightGBM is much better than CatBoost. CatBoost as an innovative method, outperforms other models in most of benchmark datasets especially in categorical datasets like Amazon. However, the permutation-

driven ordered boosting method do not perform as well as LightGBM in that case with almost numeric features.

Number of features	RMSE	
	LightGBM	*CatBoost*
60	3.62825	3.65745
70	**3.62789**	3.65858
80	3.62931	3.65742
90	3.62838	3.65808
100	3.62876	**3.65362**
120	3.63027	3.65793
150	3.63066	3.65857
160	3.63095	3.65760
170	3.63059	3.65835
180	3.63083	3.65652

Table 4: Submission results

REFERENCES

Altmann, A., Tolosi, L., Sander, O., & Lengauer, T. (2010). Permutation importance: a corrected feature importance measure. *Bioinformatics, 26*(10), 1340-1347. Retrieved from https://www.ncbi.nlm.nih.gov/pubmed/20385727. doi:10.1093/bioinformatics/btq134

Bergstra, J., & Bengio, Y. (2012). Random Search for Hyper-Parameter Optimization. *Journal of Machine Learning Research, 13*(Feb), 281-305. Retrieved from <Go to ISI>://WOS:000303046000003.

Chen, T., & Guestrin, C. (2016). *Xgboost: A scalable tree boosting system*. Paper presented at the Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining.

Dorogush, A. V., Ershov, V., & Gulin, A. (2018). CatBoost: gradient boosting with categorical features support.

Frazier, P. I. (2018). A tutorial on Bayesian optimization. *arXiv preprint arXiv:1807.02811*.

Friedman, J., Hastie, T., & Tibshirani, R. (2000). Additive logistic regression: a statistical view of boosting (With discussion and a rejoinder by the authors). *The Annals of Statistics, 28*(2), 337-407. Retrieved from https://projecteuclid.org:443/euclid.aos/1016218223. doi:10.1214/aos/1016218223

Friedman, J. H. (2001). Greedy function approximation: A gradient boosting machine. *Annals of Statistics, 29*(5), 1189-1232. Retrieved from <Go to ISI>://WOS:000173361700001. doi:DOI 10.1214/aos/1013203451

Friedman, J. H. (2002). Stochastic gradient boosting. *Computational Statistics & Data Analysis, 38*(4), 367-378. Retrieved from <Go to ISI>://WOS:000173918200002. doi:Doi 10.1016/S0167-9473(01)00065-2

Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., . . . Liu, T.-Y. (2017). *Lightgbm: A highly efficient gradient boosting decision tree*. Paper presented at the Advances in Neural Information Processing Systems.

Prokhorenkova, L., Gusev, G., Vorobev, A., Dorogush, A. V., & Gulin, A. (2018). *CatBoost: unbiased boosting with categorical features*. Paper presented at the Advances in Neural Information Processing Systems.

Snoek, J., Larochelle, H., & Adams, R. P. (2012). *Practical bayesian optimization of machine learning algorithms*. Paper presented at the Advances in neural information processing systems.

Xia, Y. F., Liu, C. Z., Li, Y. Y., & Liu, N. N. (2017). A boosted decision tree approach using Bayesian hyper-parameter optimization for

credit scoring. *Expert Systems with Applications, 78*, 225-241. Retrieved from <Go to ISI>://WOS:000398877400017. doi:10.1016/j.eswa.2017.02.017