Comparison of Tree-Based Algorithms with Google Merchandise Store Dataset

Fall 2018, SI 670

Yuheng Cai
20698962
yhcai@umich.edu

Shixu Lu
10186443
lushixu@umich.edu

Jiayi Zhang
85652257
zjiayi@umich.edu

I certify that the following paper represents my own independent work and conforms with the guidelines of academic honesty described in the UMich student handbook.

Abstract

Predicting the revenue from customers receives considerable attention in the analytics since it plays a crucial role in constructing promotional strategies. However, how to handle the categorical features and the polarized distribution of datasets has always been difficult. In our project, we compared the performances between five tree-based algorithms, which are Decision Tree, Random Forest, XGBoost, LightGBM and CatBoost. The result shows that Gradient Boosting Machines are able to handle our dataset containing plenty of categorical predictors better than Linear Regression, Decision Tree and Random Forest do. Additionally, among all the Gradient Boosting Machines, LightGBM can return the results in the shortest time and CatBoost has the lowest RMSE in the testing period.

Hypothesis

Basically, we formed four hypothesis:

- Hypothesis 1: XGBoost, LightGBM and CatBoost can significantly reduce variance and bias.
- Hypothesis 2: The length of the training period in LightGBM model can be shorter than that of other models.
- Hypothesis 3: CatBoost will need less number of iteration and it may have higher accuracy.
- Hypothesis 4: Linear Regression, Decision Tree and Random Forest model will experience some overfitting issues.

Background of Problem

Predicting the revenue collected from each customer is very important for online business because it serves as a reference for designing promotional strategy. According to the prevalent 20/80 rule, only a small portion of customers will actually spend money on the website. How to identify the non-

zero spending customers and accurately predict their spending is a great challenge for the modelers.

Based on Google Analytics Kaggle competition, the project aimed to predict revenue that each customer can bring to Google Merchandise Store, the goal of my project is to use different tree-based algorithms to make prediction in this competition and find their disadvantages and advantages with structured dataset like this.

Data Exploration

The dataset we used is provided by Kaggle. Because the size of the dataset is too large (25 GB for train data and 8 GB for test data), it is impossible for my PC to read it. To solve this problem, we divided the dataset into several batches and sequentially repeat the same step to drop obvious irrelevant columns. Finally, I save the preprocessed datasets as csv. file which are both smaller than 1GB.

Then we split them into three datasets, training, validation and testing, based on their observation dates.

	Time period	#. of observations
Train data	8/1/2016 – 5/31/2017	770000
Validation data	6/1/2017 – 4/30/2018	940000
Test data	5/1/2018 – 10/15/2018	400000

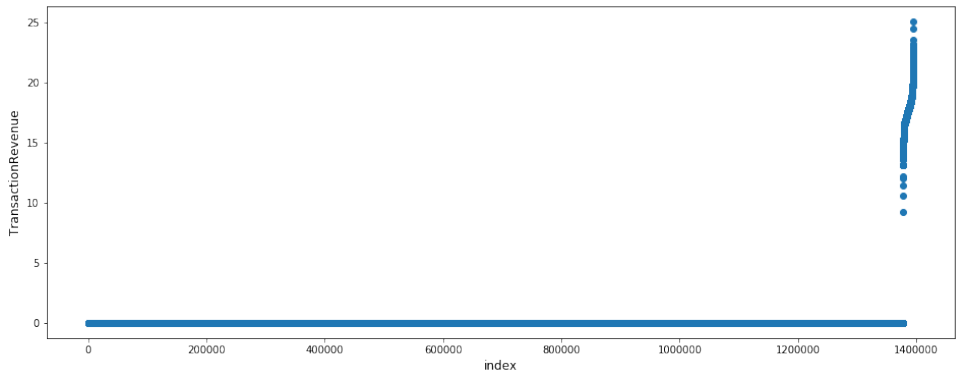
The output variable is transaction revenue of all customers in different days. As required by the competition, we first aggregate the output variable by each customer then take logarithm of them as prediction value to compute RMSE.

$$y_{user} = \sum_{i=1}^{n} transaction_{user_i}$$

$$target_{user} = \ln(y_{user} + 1)$$

$$RMSE = \sqrt{\frac{\sum_{t=1}^{T}(\hat{y}_t - y_t)^2}{N}}$$

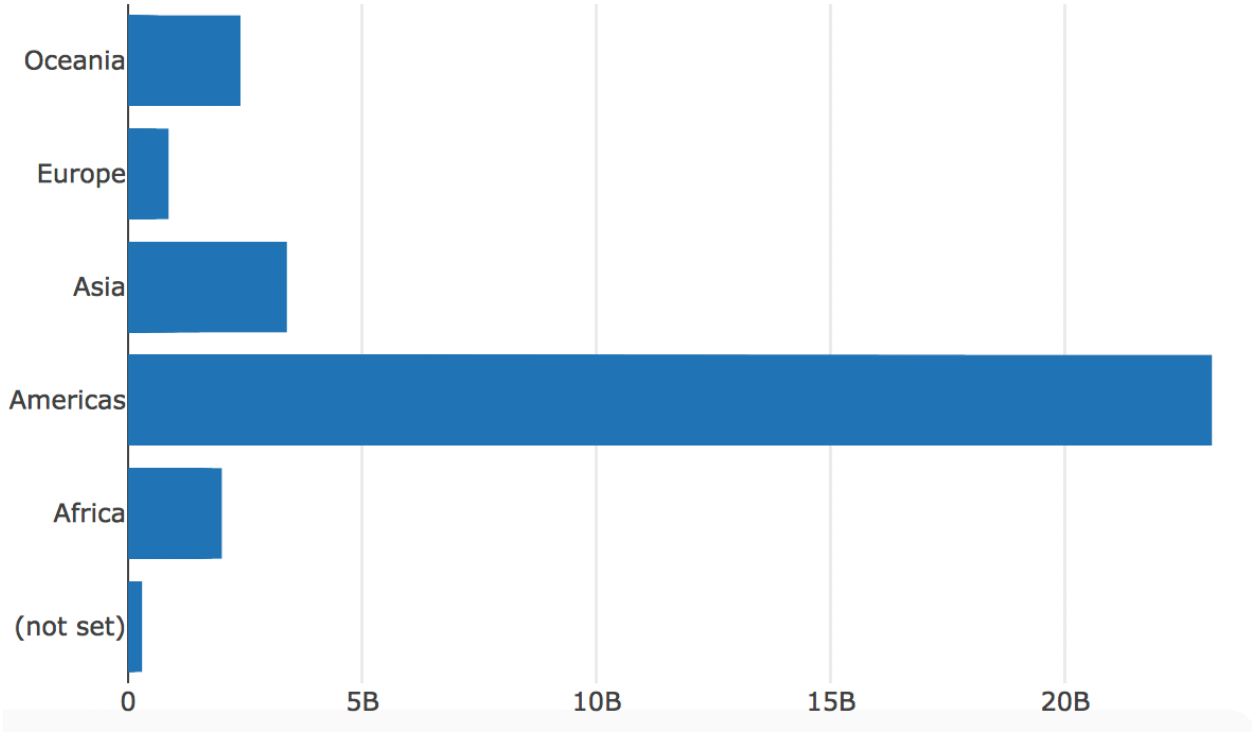
Here is the plot of our sorted total transaction for the whole dataset.

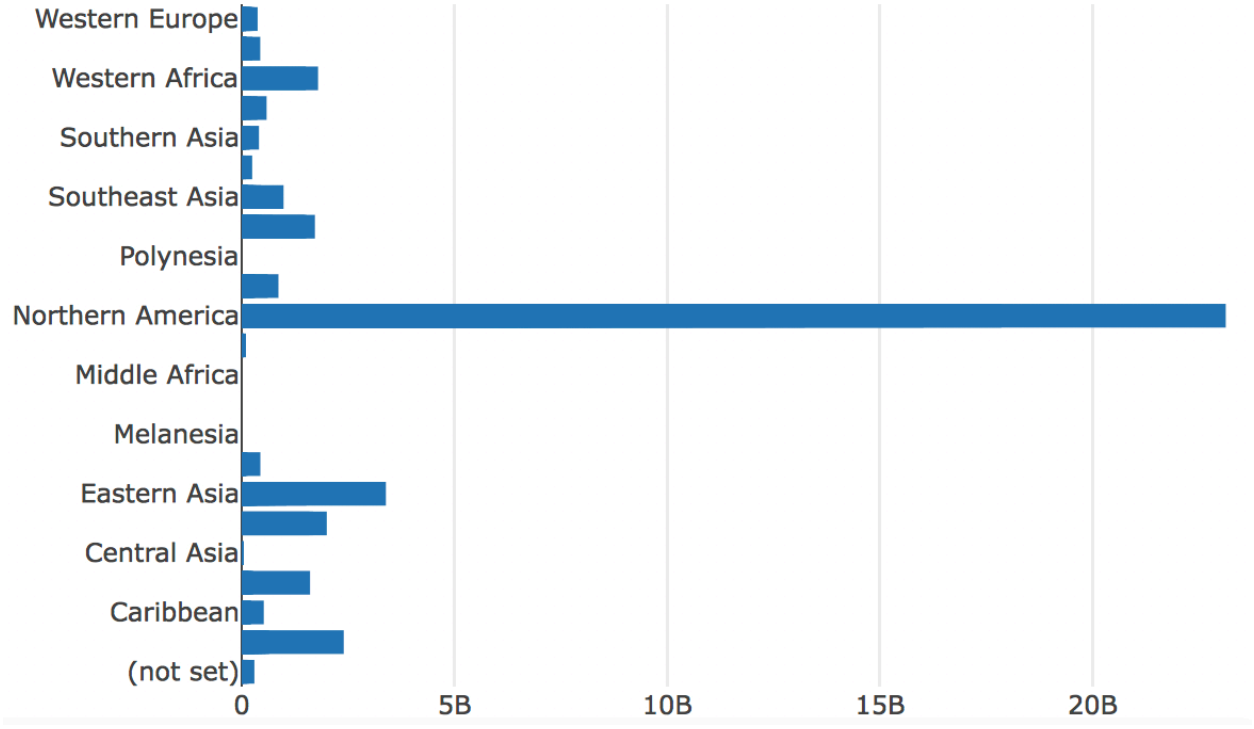


From the plot we can see that most people have no expenditure at the store which is reasonable and the rest also has polarized distribution. Therefore, we also take logarithm of them in data preprocessing to ensure prediction will be stable enough. Notice that we only transform the output variable in train and validation period in this way for modeling purpose. In test period, after getting the original prediction, we will transform them back and aggregate the result according to the formula above.

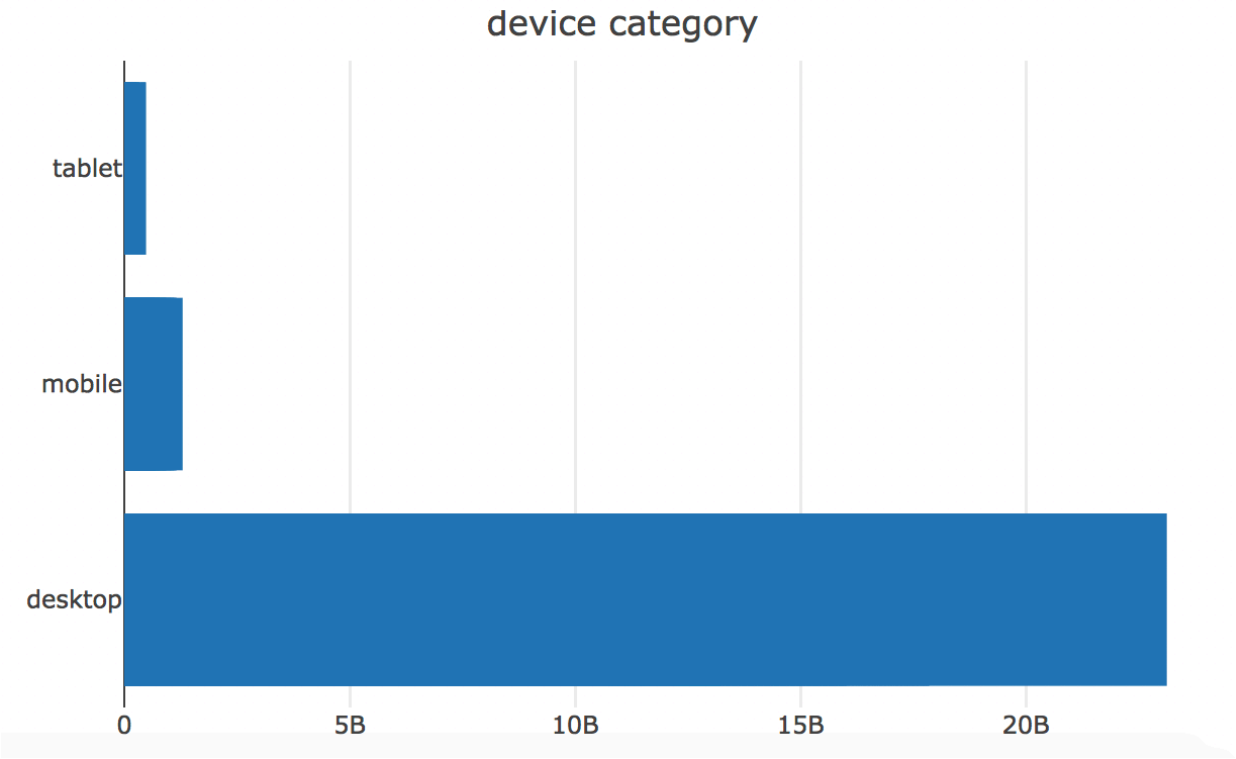
Here we would like to display some of the features we used and explained the reasons why we selected them.

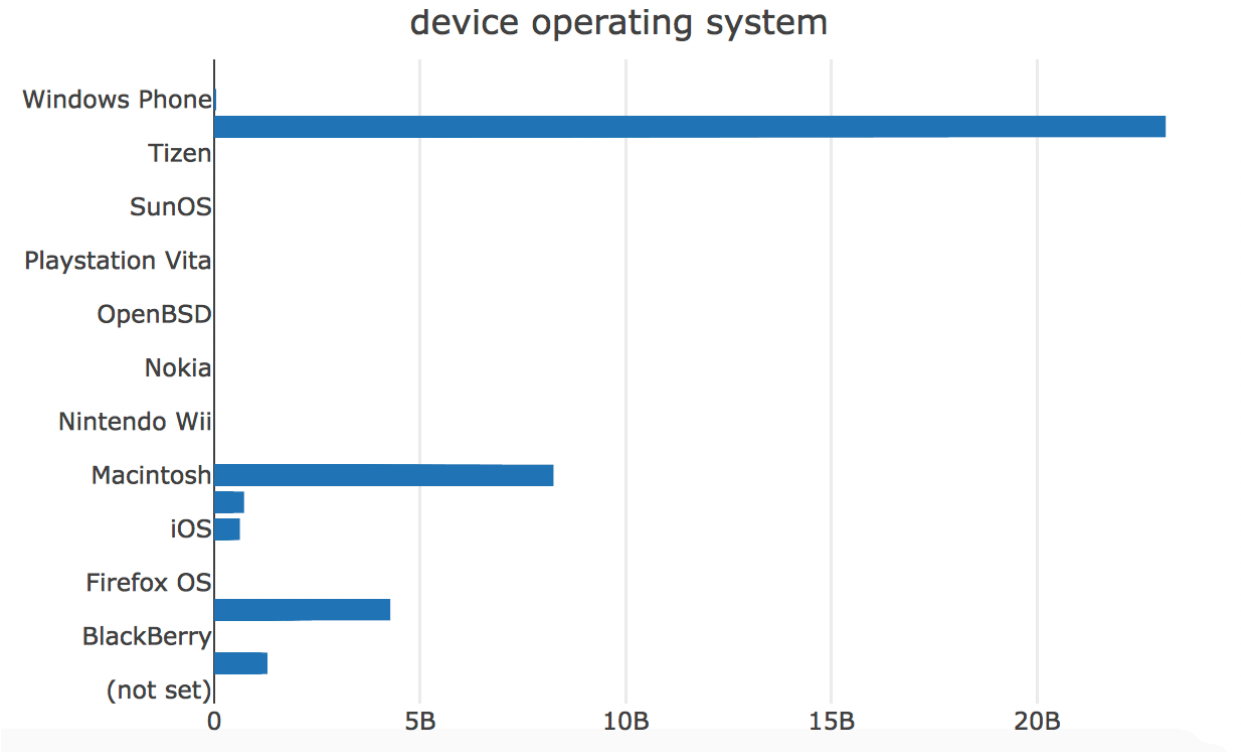
- GeoNetwork
  geoNetwork contains information about the geography of the user. We hypothesised that people from North America will spend most money at the store, because the majority of big fans of Google are in this area. From the plot, it is obvious that people from Americas contribute most to the revenue. Among these, Northern America is the main force. Below are the plots to illustrate:


Oceania
Europe
Asia
Americas
Africa
(not set)
0
5B
10B
15B
20B


Western Europe
Western Africa
Southern Asia
Southeast Asia
Polynesia
Northern America
Middle Africa
Melanesia
Eastern Asia
Central Asia
Caribbean
(not set)
0
5B
10B
15B
20B

- Device
  Device specifies the device used to access the store. We hypothesised that customers will use their mobile phone as their main access, however, more than 90% revenue is generated through desktops. Below are the plots to illustrate:





From those plots, we can see that these variables have high volatility, leading to higher feature importance, which can be viewed as our potential features.

Methods

The algorithm we used were mostly tree-based algorithms including **Decision Tree, Random Forest, XGBoost, LightGBM and CatBoost**. The reason why we chose tree-based algorithm is because these datasets have many categorical features which are hard to be managed by linear models. To make sure that these tree-based models really perform well, we built a linear regression model as the baseline model.

- Linear Regression
  Linear regression assumes that there is linear relationship between dependent variable y and explanatory variables. An error term is added to reveal the disturbance of the model.

$$y_i = \beta_0 + \beta_1 x_{i,1} + \beta_2 x_{i,2} + \cdots + \beta_k x_{i,k} + \cdots + \beta_l x_{i,l} + \varepsilon_i$$

  Most variables available are categorical, therefore we first transformed these variables through dummy coding. On account of the large number of variables exist, we used forward selection in our model. There was no candidate in our linear model at first, then we selected the variable with maximum adjusted R-squared (minimum RMSE). At each iteration, we selected the variable that makes the most contributions to the growth of adjusted R-squared until all left variables are insignificant.

- Decision Tree
  In our case, Decision trees aim to split the data based on categorical variables in order to get the minimum value of the entropy and increase the homogeneity of the elements within all classes.

$$Entropy(S) = \sum_{i=1}^{n} w_i Entropy(P_i) = \sum_{i=1}^{n} w_i \left(\sum_{j=1}^{c} -p_i \log_2(p_i) \right)$$

While making a decision tree, observations can be classified using as many splits as possible. This eventually might over classify the data. In order to avoid over classifying, we put the limitation on the number of the maximum depth of the Decision Tree, which is called the early stopping, or pre-pruning procedure. The package we used to build the Decision Tree is called 'party' and the function ctree in it.

We also applied C50 algorithm, which uses the post-trimming techniques methods to control the size of the decision tree. It first expands into an overfitting large tree and then removes nodes and branches of which the entropies are in relatively high values. Another way we used to reduce overclassfying is that we built validation sets to perform the post-trimming and evaluate the corresponding nodes with the C50 tree.

By comparing the results from the two Decision Trees, we selected the one with the lower RMSE, which is C50 tree.

- Random Forest

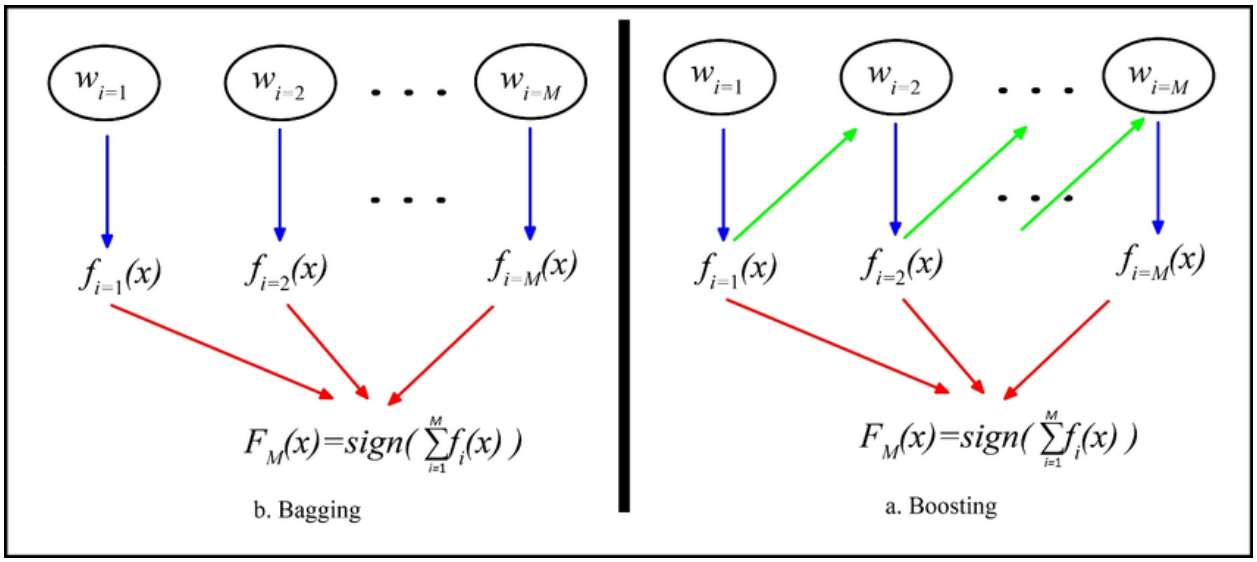
  The ideas of Random Forest are to repeat the generation of trees multiple times, predict each tree's performance, and finally ensemble those weighted votes into a combined classification result.

  In the Random Forest model, there are tradeoffs between increasing node-size and tree-size. Increasing the node size leads to smaller trees, which may compromise previous predictive power. On the flip side, increasing the tree size (maxnodes) and the number of trees (ntree) tends to increase the predictive accuracy. So we applied grid search in number of parameters and various maximum depth of tree in Random Forest to investigate the tradeoffs and come out with the best result. We also used cross validation in different elements in the grid to avoid the overfitting.

- XGBoost

XGBoost is an optimized distributed gradient boosting library designed to be highly efficient, flexible and portable. There are two major differences between it and traditional tree-based algorithms.

Firstly, instead of using bagging like random forest, this method uses boosting. The basic idea of boosting is to add one layer of model at a time which can minimize the loss function. Because of this, boosting can significantly reduce the bias of model. Additionally, because boosting will aggregate the prediction of all layer together, the variance will also be reduced.



Secondly, by taking Gradient of Loss Function over the value for each leaf, given a tree structure, this algorithm will identify the optimal value to assign to each leaf.

$$obj^{(t)} = \sum_{j=1}^{T} \left[G_j w_j + \frac{1}{2}(H_j + \lambda) w_j{}^2 \right] + \gamma T$$

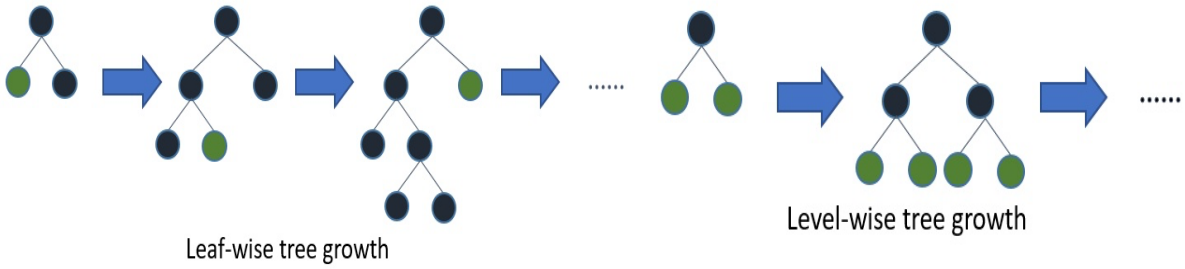
In order to make the prediction of XGBoost, LightGBM and CatBoost comparable to each other, I tried to set all the common parameters to be the same:

Evaluation Metric	RMSE

Learning Rate	0.0025
Max Depth	10
Subsample Fraction	0.7
Feature Fraction	0.5
L_1 regularization	0.001
Iteration	10000
Early Stopping	100

- LightGBM
  Based on XGBoost and designed to compete against XGBoost, LightGBM can produce almost the same accuracy while taking much less time. There are two reasons why it runs so fast.

  Firstly, it applies the leaf-wise algorithm to prevent wasting time on the leaf that can only produce little information gain. Secondly, it has two algorithms to reduce the time in finding the optimal split of each leaf. As we know that the complexity of finding the best split is ***O(#data*#features).*** LightGBM has two algorithm GOSS and Greedy Bundling to reduce the # data and # features in each leaf respectively, thus will run much faster than XGBoost.



- CatBoost
  Also based on XGBoost, this algorithm can save you plenty of time in encoding the categorical features because it can use the formula below to transform the categorical features directly into a numerical value.

$$value_{new} = \frac{countInClass + prior}{totalCount + 1}$$

Notice that there is a smoothing term, prior, in the numerator. The typical value of prior is the mean of all response. This will successfully help to prevent giving the rare category too much or too little numerical value, thus prevent the overfitting.

To understand the categorical variable transformation in CatBoost better, we also tried it externally to prepare datasets for both linear regression and decision tree.

Results

The table below display the RMSE and training time for each model, where * denotes that the categorical features are transformed. RMSE_Agg_Val and RMSE_Agg_Test are RMSE after taking summation for each customer,

while RMSE_Val and RMSE_Test are raw RMSE without taking summation. All these results correspond to our 4 hypothesis and provide support for conclusion and analysis.

Method	RMSE_Val	RMSE_Agg_Val	RMSE_Test	RMSE_Agg_Test	# Iteration Before Convergence	Training Time(s)
Linear Regression	1.616	1.7002	1.8659	2.0119	NA	1
Decision Tree	1.7219	1.7424	2.4326	2.531	NA	3.63
Random Forest	1.5464	1.5739	1.9326	2.0079	NA	1007.92
XGBoost	1.4648	1.5102	1.6603	1.7443	1802	4172.91
LightGBM	1.4629	1.5043	1.6615	1.7374	2300	247.65
CatBoost	1.4645	1.5105	1.6316	1.7209	986	3116.22
Linear Regression *	1.5985	1.6816	1.7858	1.9119	NA	0.89
Decision Tree *	2.074	2.1347	2.3713	2.4602	NA	4.4

- Hypothesis 1: As we can see from the table, the RMSEs of XGBoost, LightGBM and CatBoost in both validation and test period are all much lower than other models'.

- Hypothesis 2: Among three gradient boosting algorithms, LightGBM run with shortest time to produce almost some accuracy as the others, even though it takes much more iterations than XGBoost and CatBoost before convergence.

- Hypothesis 3: CatBoost has lowest RMSE in test period among three algorithms and its round of iteration is the fewest too.

- Hypothesis 4: Linear Regression, Decision Tree and Random Forest all have acceptable RMSEs in validation period but much higher RMSEs in test period, which indicate that these models are overfitted.

Discussion and Conclusion

The overfitting of linear regression, decision tree and random forest indicates that these traditional methods cannot handle the categorical features and numerical features well at the same time. Therefore, it is necessary to introduce some more advanced techniques designed for the structured datasets. Gradient Boosting Machine is well known for performing this kind of task.

As we can see from the results, these three algorithms can significantly reduce the bias and variance. Additionally, LightGBM can run much faster still produce similar accuracy because of two algorithms it applies to reduce number of features and number of instances used in each step.

As for CatBoosting, because of the algorithm it use to encode the categorical variables into its category's own label mean with smoothing term, it can not only converge faster but also help to prevent the overfitting. To further confirm its effect in preventing overfitting, we applied the encoding method externally to prepare the dataset for Linear Regression and Decision Tree. As we can see from the table, these two methods' RMSE during test period all reduced significantly.

With all these advanced algorithm in hand, I would highly recommend using LightGBM because it is not only reducing the bias and variance of prediction, but most importantly, it runs really fast. As is well-known in Kaggle, fancier algorithm can only help you to approach the upper bound of accuracy while feature engineering will determine the upper bound. Therefore, if we use

LightGBM in Kaggle competition, it will save us a lot of time so that we can concentrate more on feature engineering.

Acknowledgments and References

For now, there are thousands of teams competing on this Kaggle Competition. Most of the teams take LightGBM as their tools to model and their most of their difference comes from parameter tuning and feature engineering.

To understand these gradient boosting algorithm better, I read their documents and their link is as below:
LGBM:https://papers.nips.cc/paper/6907-lightgbm-a-highly-efficient-gradient-boosting-decision-tree.pdf
CatBoost:http://learningsys.org/nips17/assets/papers/paper_11.pdf
XGBoost:https://xgboost.readthedocs.io/en/latest/tutorials/model.html

I also read some Kernel from Kaggle about Data Exploration to help me understand and read in the data better. Such as these two:
https://www.kaggle.com/sudalairajkumar/simple-exploration-baseline-ga-customer-revenue
https://www.kaggle.com/julian3833/1-quick-start-read-csv-and-flatten-json-fields