Capstone Project

Machine Learning Engineer Nanodegree

Pingping Chen
June 6th, 2018

Definition

Project Overview

The real estate industry is growing very fast in Bay Area. Zillow is one of the biggest online real estate database company which has a system called Zillow's Zestimate home valuation. This system is going to estimate home value based on million statistical and machine learning algorithms. My work here is going to improve the accuracy of Zestimate. The dataset is huge, and there are no patterns could be find by human eyes to predict the target. Now, Machine Learning algorithms could learn from the huge dataset and make predictions in the future.

Many Machine Learning algorithms have been talked about in recent years. For predictive problems just like this project, Regression was the first algorithm I thought about since it is the typical solution for decades. Multiple regression can be the benchmark since it utilizes the relation between more quantitative variables so that target variable can be predicted from the others[1].

Another algorithm called LGBM[2] which is a gradient boosting framework that uses tree based learning algorithm and it has high speed with handling the large size of data. And LGBM is going to be very popular because its high speed and focuses on accuracy of results. And LGBM could be tuned on loss function of mean absolute error.

Problem Statement

The target is simple: conduct machine learning algorithm to predict houses' value as accurate as possible. The value of house is highly related to its properties such as total area, the number of bedrooms, the number of bathrooms and so on. So, properties of each house as input of the model, and then train the model on thousands of houses. Evaluate the model on a testing dataset which includes houses the model hasn't seen them ever to see how the model predicts their values.
Here, the anticipated solution is LGBM which has high speed and focuses on accuracy of results.

Metrics

Mean Absolute Error between the predictions and the actual values is one of evaluation metrics for predicting problem.

$$Mean\ Absolute\ Error = \frac{\sum_{i=1}^{n} |prediction_i - acutal\ value_i|}{n}$$

MAE is the average of the absolute difference between the predicted values and observed value. The MAE is a linear score which means that all individual differences are weighted equally in the average. Smaller Mean Absolute Error means that predictions are very close to actual values. In this way, the model will have the better performance on predicting.

Another common metrics for predicting problem is Mean Squared Error between the predictions and actual values.

$$Mean\ Squared\ Error = \frac{\sum_{i=1}^{n} (prediction_i - acutal\ value_i)^2}{n}$$

MSE measures the average of the squares of the errors or deviations that is the difference between the estimator and actual value. The MSE assesses the quality of an estimator or a predictor. An MSE of zero, meaning that the estimator predicts observations with perfect accuracy, is the ideal, but is typically not possible.

Almost the same thing as Mean Absolute Error except it is going to square the distance between prediction and actual value. Smaller Mean Absolute Error means that the predictions have smaller distance to actual values. So, the model will have the better performance on predicting the actual value.

Analysis

Data Exploration

There are 60 features in total in the dataset. The target variable is called logerror. And the whole purpose is going to predict the target variable by using other features. The type of features is shown in the below table.

Feature Name	Type	Feature Name	Type
parcelid	int64	poolsizesum	float64
logerror	float64	pooltypeid10	float64
transactiondate	Datetime64[ns]	pooltypeid2	float64
airconditioningtypeid	float64	pooltypeid7	float64
architecturalstyletypeid	float64	propertycountylandusecode	object
basementsqft	float64	propertylandusetypeid	float64
bathroomcnt	float64	propertyzoningdesc	object
bedroomcnt	float64	rawcensustractandblock	float64
buildingclasstypeid	float64	regionidcity	float64
buildingqualitytypeid	float64	regionidcounty	float64
calculatedbathnbr	float64	regionidneighborhood	float64

decktypeid	float64	regionidzip	float64
finishedfloor1squarefeet	float64	roomcnt	float64
calculatedfinishedsquarefeet	float64	storytypeid	float64
finishedsquarefeet12	float64	threequarterbathnbr	float64
finishedsquarefeet13	float64	typeconstructiontypeid	float64
finishedsquarefeet15	float64	unitcnt	float64
finishedsquarefeet50	float64	yardbuildingsqft17	float64
finishedsquarefeet6	float64	yardbuildingsqft26	float64
fips	float64	yearbuilt	float64
fireplacecnt	float64	numberofstories	float64
fullbathcnt	float64	fireplaceflag	object
garagecarcnt	float64	structuretaxvaluedollarcnt	float64
garagetotalsqft	float64	taxvaluedollarcnt	float64
hashottuborspa	object	assessmentyear	float64
heatingorsystemtypeid	float64	landtaxvaluedollarcnt	float64
latitude	float64	taxamount	float64
longitude	float64	taxdelinquencyflag	object
lotsizesquarefeet	float64	taxdelinquencyyear	float64
poolcnt	float64	censustractandblock	float64
		transaction_month	Int64

The count of different types. Almost all are float variables with few object (categorical) variables

Column Type	Count
Int64	2
Float64	53
Datetime64[ns]	1
Object	5

Part of first five records are shown below

parcelid	**logerror**	**transactiondate**	**airconditioningtypeid**	**architecturalstyletypeid**	**basementsqft**	…
11016594	0.0276	1/1/16	1	**0**	NaN	…
14366692	-0.1684	1/1/16	NaN	**1**	NaN	…
12098116	-0.004	1/1/16	1	**2**	NaN	…
12643413	0.0218	1/2/16	1	**3**	NaN	…
14432541	-0.005	1/2/16	NaN	**4**	NaN	…

After getting the whole dataset, checking missing values is going to be the next step.

1. Check if there are any missing values in each feature.
   57 features out of 60 have missing values. Only ‘parcelid’, ‘logerror’, and ‘transactiondate’ do not have any missing value.
2. Check how many missing values in each feature.
   There are 20 features with more than 90% missing values, such as ‘architecturalstyletypeid’, ‘basementsqft’ and so on.
3. Five features with the most missing values are showing in the below table

	Column_name	Missing_count	Missing_ratios
5	basementsqft	167795	0.999446
8	buildingclasstypeid	167857	0.999815
15	finishedsquarefeet13	167813	0.999553
43	storytypeid	167795	0.999446
48	yardbuildingsqft26	167723	0.999017

Since there are many features, here I am only going to explore a few features which I think they are very important.

1. The target variable: logerror

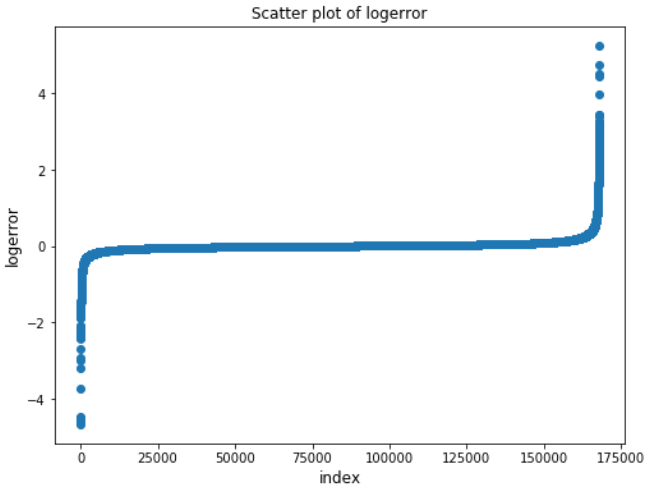
count	167888
mean	0.013906
std	0.165706
min	-4.65542
25%	-0.0253
50%	0.006
75%	0.0392
max	5.262999

   Form the above table which contains the basic statistics about logerror. There are 167,888 records in total, the mean of logerror is 0.013906, the median is 0.006, and the standard deviation is 0.165706. The maximum and minimum also included in the table.

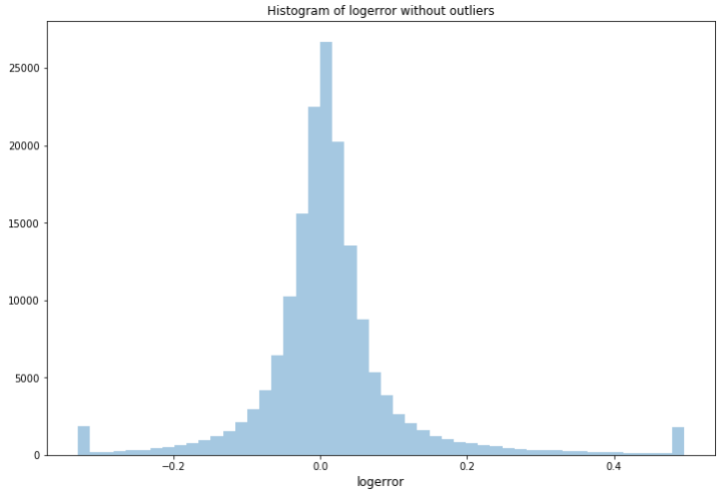
Exploratory Visualization

The target variable is logerror, so let’s see its distribution.

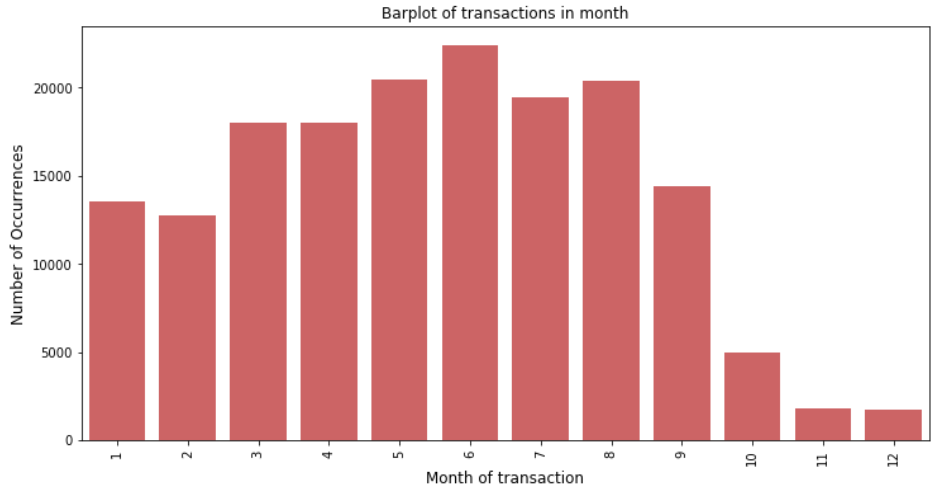
1. Scatter plot of the logerror: it looks nice with some outliers at both tails.



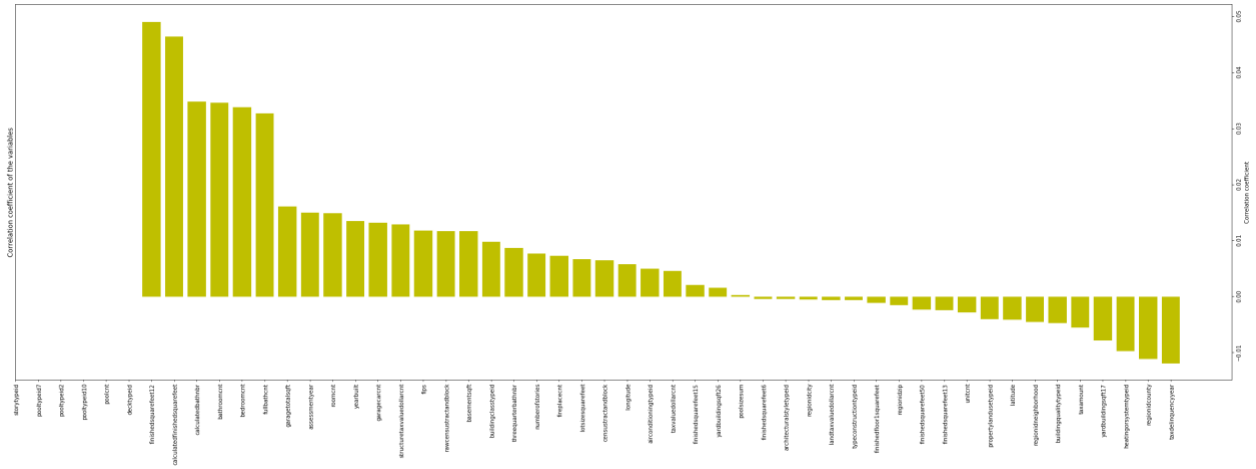
2. Let's remove these outliers at the both tails and plot the histogram. Logerror is normal distributed.



3. Check the number of transactions in each month.
   As we could see from the barplot as well, the data contains 1/1/2016 to 12/31/2016 and 1/1/2017 to 9/15/2017. Smaller number of transactions in the last three months. We are provided with a full list of real estate properties in three counties (Los Angeles, Orange and Venture, California) data in 2016 and 2017.



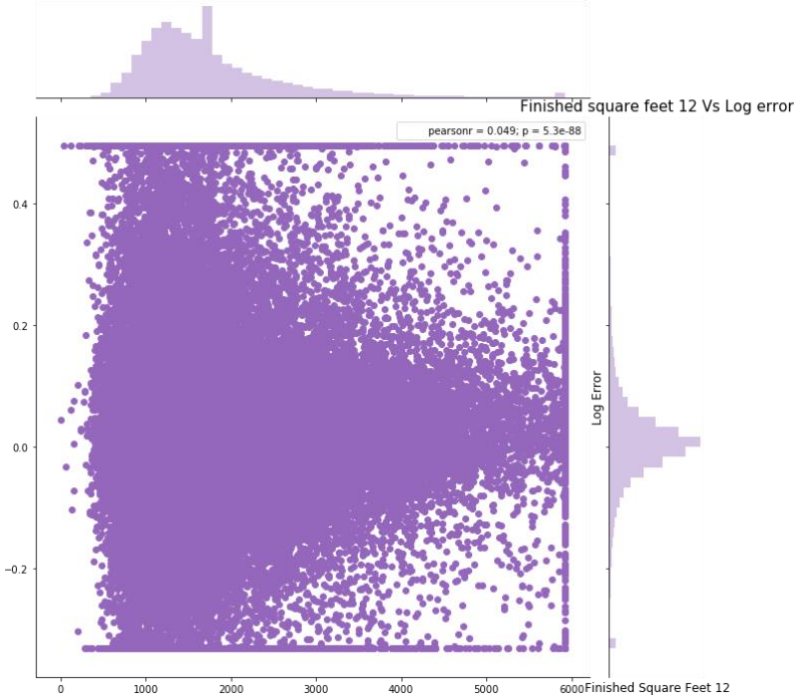
4. Get the correlation between float variables and the target variable to see how they are related.
   In order to compute the coefficients, let's impute the missing values with mean values.



The below table shows the variables with highest correlation values

Col_labels	Corr_values
taxdelinquencyyear	-0.011833
regionidcounty	-0.011037
fullbathcnt	0.032768
bedroomcnt	0.033868
bathroomcnt	0.034663
calculatedbathnbr	0.034883
calculatedfinishedsquarefeet	0.046444
finishedsquarefeet12	0.049083

5. Let's see how the finishedsquarefeet12 varies with the logerror. It seems that the range of logerror is smaller when larger finishedsquarefeet12 increases.



Algorithms and Techniques

I am going to use LGBM algorithm which is a gradient boosting framework that uses tree based learning algorithm and it has high speed with handling the large size of data. LGBM could be tuned on loss function of mean absolute error. The following parameters could be tuned to optimize the regressor:

- Control parameters:
  - Max_depth (describes the maximum depth of tree. It is used to handel model overfitting. If you feel that your model is overfitted, lower max_depth)
  - Min_data_in_leaf (is the minimum number of the records a leaf may have)
  - Feature_fraction (0.8 feature fraction means LightGBM will select 80% of parameters randomly in each iteration for building trees)
  - Bagging_fraction (specifies the fraction of data to be used for each iteration and is generally used to speed up the training and avoid overfitting.
  - Early_stopping_round (can help you speed up your analysis)
  - Lambda (specifies regularization)
  - Min_gain_to_split (describes the minimum gain to make a split)
- Core parameters
  - Task (specifies the task you want to perform on data. I may be either train or predict)
  - Application (specifies the application of your model, whether is a regression problem or classification problem. LightGBM will by default consider model as a regression model)
  - Boosting (defines the type of algorithm you want to run, default = gdbt
    - Gdbt: tranditional Gradient Boosting Decision Tree
    - Rf: random forest

    - Dart: Dropouts meet Multiple Additive Regression Trees
    - Goss: Gradient-based One-Side Sampling
  - Num_boost_round (number of boosting iterations, typically 100+)
  - Learning_rate (determines the impact of each tree on the final outcome)
  - Num_leaves (number of leaves in full tree, default 31)
  - Device (default: cpu)
- Metric parameter
  - Metric (specifies loss for model building)
    - Mae: mean absolute error
    - Mse: mean squared error

LightGBM is based on decision tree algorithms, it splits the tree leaf wise with the best fit. When growing on the same leaf in LightGBM, the leaf-wise algorithm can reduce more loss. For each leaf, the algorithm is going to reduce variance. The variance reduction of a node *N* is defined as the total reduction of the variance of the target variable *x* due to the split at this node.

During training, split the dataset into training and testing. Convert training dataset into LightGBM dataset format which is mandatory for LightGBM. Create a dictionary with parameters and their values. Accuracy of the model totally depends on the values I provide to parameters. Then fit the model on training dataset. Check results either using MAE (Mean Absolute Error) or MSE (Mean Squared Error).

Benchmark

To create an initial benchmark for the prediction, I used Multiple linear regression. The full multiple linear regression with 55 independent variables achieved the Mean Absolute Error is 0.0607, the Mean Squared Error is 0.0104 and the R-Squared is 0.0077 on training dataset.

After implemented feature selection on full multiple regression, I got the multiple regression with 12 independent variables and this model achieved the Mean Absolute Error is 0.0607, the Mean Squared Error is 0.0105, and the R-Square is 0.0035 on training. Almost the same value with the full multiple regression.

Methodology

Data Preprocessing

The most of preprocessing issues were dealing with missing values since there are many missing values in most of columns. For the multiple linear regression, I just simply imputed missing values with their averages.
For the LightGBM, I am going to filling the most missing values with their columns' median values except three columns.

- Hashottuborspa: replace missing values with 0

```
df_lgbm['hashottuborspa'] = df_lgbm.hashottuborspa.replace(np.NaN, 0)
df_lgbm['hashottuborspa'] = df_lgbm.hashottuborspa.replace(True, 1)
```

- Fireplaceflag: replace missing values with 0

```
df_lgbm['fireplaceflag'] = df_lgbm.fireplaceflag.replace(np.NaN, 0)
df_lgbm['fireplaceflag'] = df_lgbm.fireplaceflag.replace(True, 1)
```

- Taxdelinquencyflag: replace missing values with 0

```
df_lgbm['taxdelinquencyflag'] =
df_lgbm.taxdelinquencyflag.replace(np.NaN, 0)
df_lgbm['taxdelinquencyflag'] = df_lgbm.taxdelinquencyflag.replace('Y',
1)
```

Here, data preprocessing is very simple and easily to implement.

Create a new variable called age which is calculated by transaction year minus built year.

```
df_lgbm['transaction_year'] = df_lgbm['transactiondate'].dt.year
df_lgbm['age'] = df_lgbm['transaction_year'] - df_lgbm['yearbuilt']
```

Drop five features: ‘transactiondate’, ‘propertycountylandusecode’, ‘propertyzoningdesc’, ‘yearbuilt, ‘transaction_month’.

```
x = df_lgbm.drop(['parcelid', 'logerror', 'transactiondate',
'transaction_month',
                  "propertycountylandusecode", "propertyzoningdesc",
'yearbuilt'], axis = 1)
y = df_lgbm['logerror']
```

Implementation

The implementation process can be split into two stages:

- Get the training and testing data.
  - Split the data by using train_test_split(). Before implement LGBM model on the training dataset, I need to do feature scaling by using MinMaxScaler()

```
x = x.values
y = y.values

# split the data
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size =
0.2, random_state = 42)

# feature transformation
scaler = MinMaxScaler().fit(x_train)
x_train = scaler.transform(x_train)
x_test = scaler.transform(x_test)
```

- Convert the training data into LightGBM dataset format (this is mandatory for LightGBM training)

```
d_train = lgb.Dataset(x_train, label = y_train)
```

- Model building
  Create a python dictionary with parameters and their values. The performance of my model totally depends on the values I provide to parameters. There are many hyperparameters, here is just an initial list to implement the LightGBM model.

```
params = {}
params['learning_rate'] = 0.003
params['boosting_type'] = 'gbdt'
params['objective'] = 'regression'
params['metric'] = 'l1'
params['bagging_fraction'] = 0.8
params['sub_feature'] = 0.5
params['bagging_freq'] = 40
params['num_leaves'] = 1000
params['min_data'] = 50

clf = lgb.train(params, d_train, 100)
```

➢ Model prediction

- Write predict function on the testing dataset. This will give us the predicted values by the model. We could calculate the metrics by using the predictions.

```
y pred = clf.predict(x test)
```

Refinement

As mentioned in the Benchmark section, the Multiple Linear Regression trained with the training dataset with 15 independent variables achieved MSE (Mean Squared Error) around 0.0104, MAE (Mean Absolute Error) around 0.0606 and R-Square around 0.0079.

In my implementation, to get the initial result, I have applied LightGBM on training dataset with arbitrary values of parameters. The result achieved MSE around 0.0101, MAE around 0.0598 and R-Square around 0.0338 on training dataset. And the initial results on testing dataset are MSE is around 0.0105, MAE is around 0.0609, and R-Square is 0.0084.

This was improved upon by tuning the parameters' values and adding extra parameters:

- Adding max_bin
- Decrease the learning_reate
- Increase the number of leaves (num_leaves)
- Adding min_hessian (min_sum_hessian_in_leaf)
- Adding verbose
- Increase the number of round (num_round)

```
params = {}
params['max_bin'] = 10
params['learning_rate'] = 0.002
params['boosting_type'] = 'gbdt'
params['objective'] = 'regression'
params['metric'] = 'l1'
params['bagging_fraction'] = 0.85
params['sub_feature'] = 0.5
params['bagging_freq'] = 40
params['num_leaves'] = 1000
params['min_data'] = 50
params['min_hessian'] = 0.05
params['verbose'] = 0

clf = lgb.train(params, d_train, 440)
```

And this achieved the MSE is around 0.0095, MAE is around 0.0581, and R-Square is around 0.0906on training dataset. According all the metrics, this improved model has the better performance than the initial one.

Results

Model Evaluation and Validation

During development, a testing dataset was used to evaluate the model.

The final model and hyperparameters were chosen because they performed the best among the tried combinations.

With the parameters and their values in the refinement part, I got metric values for testing data. The MAE is around 0.0607, MSE is around 0.0104, R-Square is around 0.0183.

Justification

Comparing with the benchmark result, I have got below metrics on the testing dataset by using LightGBM:

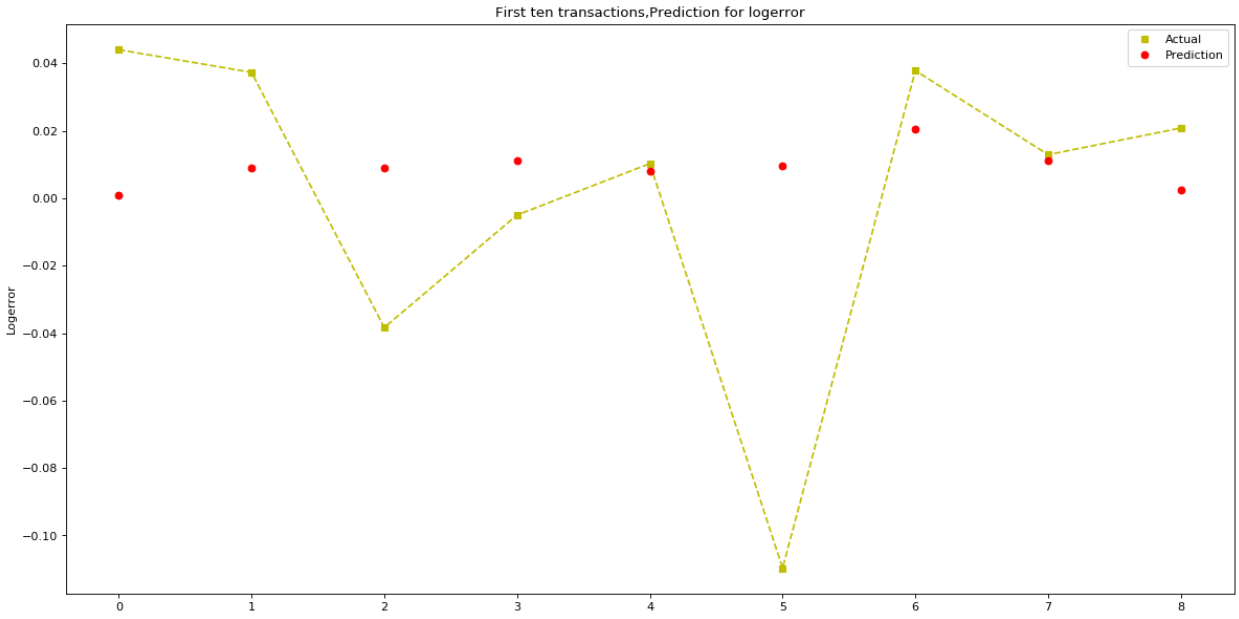
- MAE (Mean Absolute Error) is around 0. 0607which is almost the same with the benchmark
- MSE (Mean Squared Error) is around 0. 0104which is also almost the same with the benchmark
- R-Square is around 0. 0183which is greater than benchmark’s

Since there are many parameters in the LightGBM, I couldn't implement all of them in the model, and I could spend more time to tune the model to get the much better model. In summary, I am sure that LightGBM is going to be useful in predicting lager dataset.

Conclusion

Free-Form Visualization

Let's see the first ten transactions in the testing dataset, to see how the final LightGBM model predicts them. From the below graph, we could see that the model could predict a few transactions very accurate. However, I could say that the model isn't perfect and there still have many works could do.



Reflection

The process used for this project can be summarized using the following steps:

1. Find an interesting problem, and get the data from public website
2. Explore the dataset to know how many and what variables in there. Identify the target variable (logerror) and independent variables.
3. Exploratory data analysis to know the distributions of target variable and explanatory variables.
    - Calculate variance-covariance matrix of variables
    - Dealing with missing values
    - Visualize the target data with other independent variables

4. A benchmark was created for the prediction. Multiple Regression, to calculate the evaluation metrics (Mean Absolute Error, MSE and R-Square)
5. LigthGBM was built on training dataset
6. Tune hyperparameters to get the best performance on testing dataset.

I found the step 3 and step 6 the most difficult.

In step 3, I had to deal with many missing values in most columns. The variance-covariance matrix didn't show any strong correlations between target variable and independent variables, it was hard to see which variables would have the most importance. In the visualization part, no graph showed any patterns. This was frustrated when I couldn't find any patterns which could predict the target.

In the step 6, LigthGBM has many hyperparameters and their values could be any random numbers. All this is about experience, I think I should do more projects in real world to gain more experiences about which hyperparameters are matter and which possible values for them could give the better result.

Improvement

In this project, there are many missing values and many independent variables are highly correlated with each other. In the real world, this situation really happens a lot.

First thing, I think which could be improved is how to deal with missing values. Missing values mean that I lose information. So, I can't exclude all missing values, find appropriate way to treat or fill the missing values is very important.

Second thing is how to conduct feature selection which I done for Multiple Regression. Get rid of some variables which can't provide more information, in this way, the training time could be decrease a lot especially when I face huge dataset. Also feature selection could protect the model from overfitting.

Reference

[1] Applied Linear Regression Models; Kutner, Nachtsheim, Neter.

[2] What is LightGBM, How to implement it? How to fine tune the parameters?