KSE525: Spring Term Project

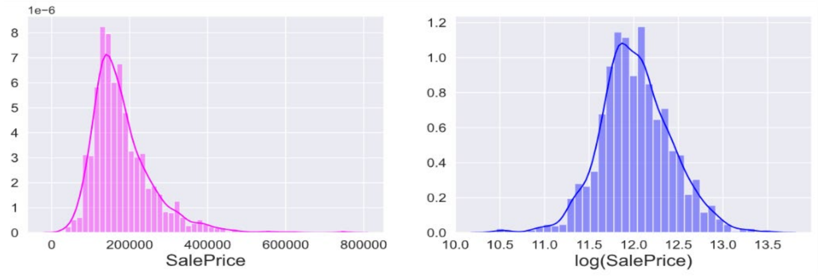
20203221 민향숙

1. Data Preprocessing And Feature Engineering

The datasets are divided into train dataset and test dataset. Train dataset has 81 features including a target variable and 1460 instances. And test dataset has 80 features and 1459 instances.

1) Rescale Target Value (SalePrice)

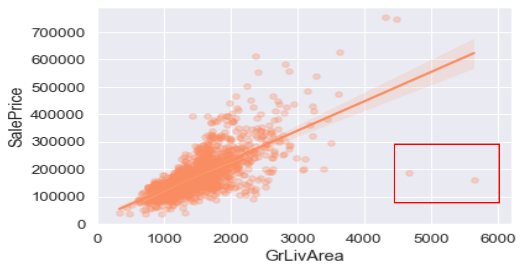
The competition evaluates the results on Root Mean Square Logarithmic Error (RMSLE) between the predicted value and the observed sales price. Thus, it would be better to train the model with the target value (SalePrice) changed into logarithm form. In addition, taking log on the value reduces the skewness and kurtosis, which makes the distribution of the target values close to normal distribution. It makes us have more precise results and finally improves the performance. It is described in Figure1.



[Figure 1]

2) Imputes Missing Value

The 19 features in train dataset and the 33 features in test dataset have missing values. Some features have more than 90% missing values of entire rows. And some features are dominated by a single value. Thus, I deleted 'PoolQC', 'Street' and 'Alley'. But for 'MiscFeature', which means "Miscellaneous feature not covered in other categories" in the data description, it is not deleted because the feature is about additional function of the house. And I create new features ('Has_Shed', 'Has_Gar2', and 'Has_TenC') to emphasize 'MiscFeature'.



[Figure 2]

Before diving into the data preprocessing, most kaggle kernels deleted the rows whose "GrLivArea" is over 4500. Since the row with "GrLivArea" over 4500 has a lower "SalePrice"(Target value) compared to other rows with the same condition, and these types of rows are outliers. (Reference Figure 2).

After deleting outliers and features that have lots of NA, the train and test dataset are preprocessed in 4 steps: Impute NA, Feature Engineering, Adjust Skewness and Delete Outliers. First step is imputing the missing values. There are 3 ways imputing missing values, (1) considering existence of a value in other higher-order features, (2) filling mode or mean by considering another highly correlated feature, or (3) imputing 0 or 'None' for all features. As an example of (1), if value of "GarageArea" is 0, which means the house doesn't have a Garage, all features related to Garage such as GarageCars, GarageType, and so on should be 'None' or 0. Thus, the missing value in "GarageYrBlt", "GarageCars", "GarageTye", "GarageFinish", "GarageCond", and "GarageQual" are filled with 'None' or 0 if the GarageArea of corresponding rows have 0. On the other hand, if GarageArea has a positive value, they are imputed by looking through the values of similar rows. As an example of (2), there is a feature named "LotFrontage", which means "Linear feet of street connected to property". We can assume that "LotFrontage" is varied depending on the neighborhood. Thus, I fill NA of "LotFrontage" with a mode of a same neighborhood. (3) Features except corresponding to 1) and 2), are imputed by 0 or 'None'.

3) Create New Features

Second step of data preprocessing is feature engineering. In an effort to make the performance better, I make new features by mixing some features. For example, "Total_Bathrooms", are created by combining all 4 features related to "Bathroom". Furthermore, some features such as "GarageCars", "GarageQual", "BsmtCond", and "FireplacesQu" can have the real values only when "GarageArea", "TotalBsmtSF", and "Fireplaces" have values greater than 0. In order to simplify the existence of Garage, Basement and so on, the categorical features are newly created. For example, if "GarageArea" has a value greater than 0, new feature has 1 if not it has 0. Additionally, I create the features about whether house was remodeled or not, by comparing the features "YearBuilt" and "YearRemodAdd".

4) Adjust Skewness

Third step of data preprocessing is adjusting skewness. After imputing missing values, we need to adjust the skewness of the features to reduce the variance of the features and improve the performance. In addition, since we already took the log of the target value, we want to make them similar to it. To do this, the features with high skewness are rescaled with Box-cox transformation.

5) Delete Outliers

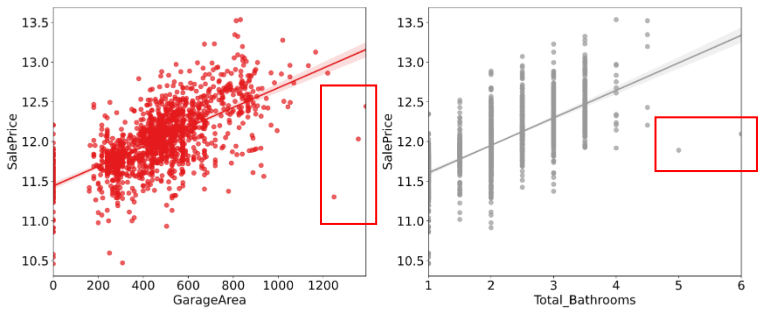


Figure 3

In order to make the training work better, I try to detect outliers in train dataset using scatter plot between SalePrice(target value) and features. (Figure 3) First, I draw multiple scatter plots between SalePrice and other features. For example, in scatter plots with "GarageArea", and "Total_Bathrooms", I find some outliers. The outliers have higher values in each feature, but their target values are much lower than trend. I decide that they would be obstacle to train the model, so I delete them from train dataset.

6) One-hot Encoding

For categorical features, in order to make models work well, we need to convert the categorical variables into dummy variables. By using one-hot encode method, dummy variables are created for every categorical variable.

2. MODELING

Since predicting the house price is the regression problem, linear models such as *BayesianRidge, Ridge, Lasso* are used. Boosting methods such as *GradientBoostingRegressor*, *LGBMRegressor*, *XGBRegressor* are also, employed. Additionally, *StackingCVRegressor* is utilized in order to improve the performance.

In *StakcingCVRegressor*, 6 models (*Ridge, BayesianRidge, Lasso, GradientBoostingRegressor, LGBMRegressor, XGBRegressor*) are used as normal regressors and XGBRegressors are used as a meta regressor since XGBRegressors gives the best result. I use all models published in scikit-learn package, also utilized another open source lightgbm, xgboost, and mlxtend package.

Before training the models, for linear models, train and test dataset are rescaled by *RobustScaler,* because linear models are very sensitive to scale of the value. Finally, I run every model and get the RMSLE for train dataset, and they are described in Table 1.

Model	Hyperparameter	RMSLE
RidgeCV	*alpha = 15.5, cv = KFold(n_splits=10, shuffle = True)*	0.08870
Bayesian Ridge	*alpha_2=88, lambda_1 = 6.5, lambda_2=0.4*	0.09056
LassoCV	*alpha = 0.0006, cv = KFold(n_splits=10, shuffle = True)*	0.09404
GradientBoosting-Regressor	*n_estimators=3000, learning_rate=0.05, max_depth=4, max_features='sqrt', min_samples_leaf=15, min_samples_split=10, loss='huber', random_state=0*	0.03776
LGBMRegressor	*objective='regression', num_leaves=4, learning_rate=0.01, n_estimators=5000, max_bin=200, bagging_fraction=0.75, bagging_freq=5, bagging_seed=100, feature_fraction=0.2, feature_fraction_seed=100*	0.06741
XGBRegressor	*learning_rate=0.01, n_estimators=3500, max_depth=3, min_child_weight=0, gamma=0, subsample=0.7, colsample_bytree=0.7, reg_alpha=0.00006, seed = 100*	0.04508
StackingCVRegressor	*Regressor = (Ridge, BayesianRidge, Lasso, GradientBoostingRegressor, LGBMRegressor, GBRegressor) Meta_regressor = XGBRegressor*	0.04192

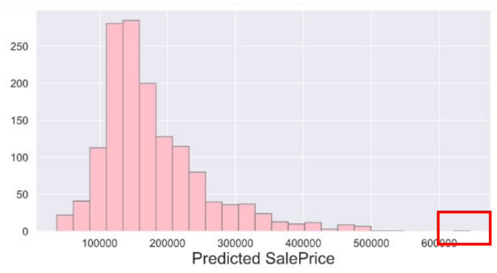
[Table 1] RMSLE for each Model

In order to improve the prediction results, I blend multiple results of models. After several attempts, I found the best combination that improves the results. They are described in Table 2. I use Ridge, LGBMRegressor and StackingCVRegressor, and they get different weights. The result is obtained by multiplying the weights Ridge by 0.1, LGBMRegressor by 0.35, and StackingCVRegressor by 0.55. For train dataset, the model has 0.05062 RMSLE. It is higher than StackingCVRegressor, but usually blending several methods gives the better results than using a single result when predicting test dataset. And submitting the results in Kaggle gives **0.11721**.

Model	Weight	RMSLE
Ridge	**0.1**	**0.05062**
Bayesian Ridge	-	
Lasso	-	
GradientBoostingRegressor	-	
LGBMRegressor	**0.35**	
XGBRegressor	-	
StackingCVRegressor	**0.55**	

[Table 2] Weights for Blending Method and RMSLE of Blending Method

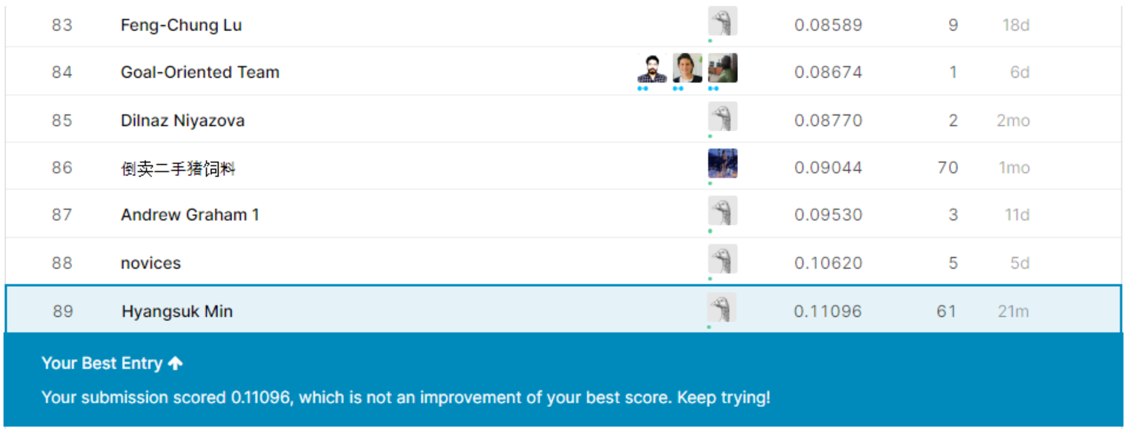
Another attempt to improve the results is using Brutal Force Method. Regression problems are vulnerable to predict the edge cases such as small values of predictors. So, I multiply some constant to adjust the edge values. The values lower than 0.45% of the results are multiplied by 0.79 and the values over than 99% of the results are multiplied by 0.925. Consequently, it reduces RMSLE about 0.00149, and final submission is evaluated as **0.11572.**



[Figure 4]

One more attempt to improve the results is rescaling the outlier. The Brutal Force method reduces RMSLE a lot. It gives me an idea that rescaling extreme edge value again might give a better result. In histogram of the predicted SalePrice, I find that one row is predicted much higher than other rows. (Figure 4) So, I rescale it by multiplying a constant, I can get much lower RMSLE in Kaggle. I search the optimal constant by submitting multiple times. Finally multiplying the extreme edge value by 0.28 reduces RMSLE about 0.00476. Consequently, final submission is evaluated as **0.11096**.

3. RESULT



83	Feng-Chung Lu	0.08589	9	18d
84	Goal-Oriented Team	0.08674	1	6d
85	Dilnaz Niyazova	0.08770	2	2mo
86	倒卖二手猪饲料	0.09044	70	1mo
87	Andrew Graham 1	0.09530	3	11d
88	novices	0.10620	5	5d
89	Hyangsuk Min	0.11096	61	21m

Your Best Entry

Your submission scored 0.11096, which is not an improvement of your best score. Keep trying!

2020-07-03 19:18 pm

FINAL_SUBMISSION-20203221.csv

Reference Kernels

https://www.kaggle.com/shaitender/ensembles

https://www.kaggle.com/wangqiyuan/7-lines-of-code-to-reach-6th-place

https://www.kaggle.com/agehsbarg/top-10-0-10943-stacking-mice-and-brutal-force