This housing price prediction project is the combined work of three members: Qinghe Gao, Zeyu Zhou, and Fan Wu ， all of whom have just completed the Machine Learning course provided by THE IVY DATA SCIENCE PROGRAM. We divided the workload equally between all members and completed the coding (excluding the writing of the final report) within five weeks. We attempted the implementation of different size machine learning models, finally adopting four of them, including Ridge model, Lasso model, Elasticnet model, and Gradient Boosting model.

Instructor: Shuo Zhang, Ph.D., Columbia University
Contact: shuozhang1985@gmail.com

HOW TO ACHIEVE RANK 2% FOR KAGGLE COMPETITION: HOUSE PRICING

1. Introduction

Upon commencement, we had some experience of R, Python, and machine learning basics. During the process, we gained knowledge of various different machine learning algorithms, from supervised or unsupervised to reinforcement learning. Now it's time to apply them to solving real problems and testing what we have learned. Fortunately, we discovered an ongoing and fascinating competition on kaggle.com, a practical competition whose goal is to predict house prices in Ames, Iowa using data covering the different features of the houses collected in 2010. There are 79 explanatory features describing every aspect of residential homes in Ames, Iowa. We considered this competition approachable because detailed explanations of the features had been provided in full to the participants. It was an invaluable opportunity to both practice and hone our skills and knowledge of advanced machine learning, such as creative feature engineering and advanced regression techniques like random forest and gradient boosting.

In five weeks, we have performed EDA, feature engineering, ensembling,

stacking, and have constantly fine tuned our parameters. Our own codes are score-oriented for the sake of further improving the prediction score. Currently, and after several submissions of our machine learning project, we are placed in the top 2% out of more than 4,500 teams on the Kaggle leader board.

1. Workflow

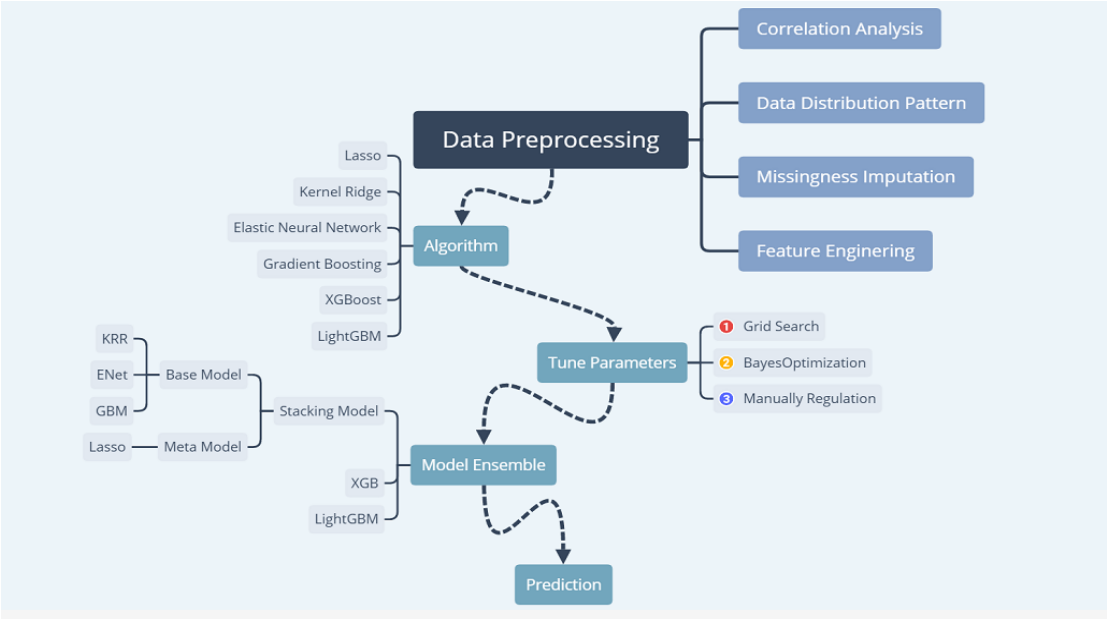


Figure1. Workflow of the program

3. Pre-Processing

3.1 Data Distribution Pattern

Our aim is to predict the house price (abbreviated as HP), thus it's important to determine the distribution of the HP of the sample.

As shown in the first map below, the shaded blue area represents the HP density histogram plot, and the blue and black lines represent the HP density line and fitted HP density line respectively. It's obvious that the distribution of the HP is not a normal one, which may cause considerable errors in our predictions. We can also infer this from the second map, which compares the theoretical quantile value with the real quantile value. We can see that most of the points are beyond the boundary, meaning that the distribution of HP is non-normal.

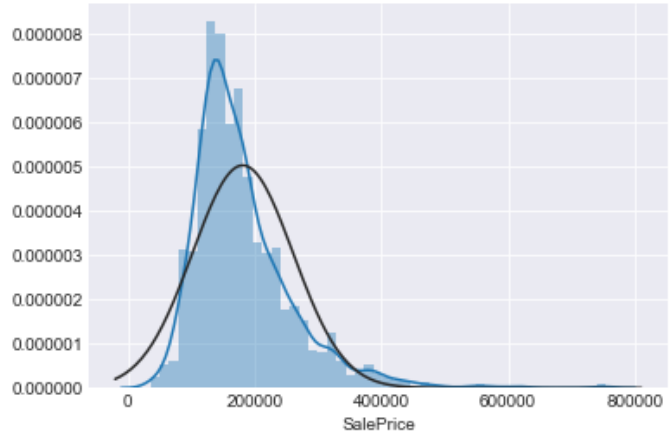


Figure2. Original Distribution of the Sale Price

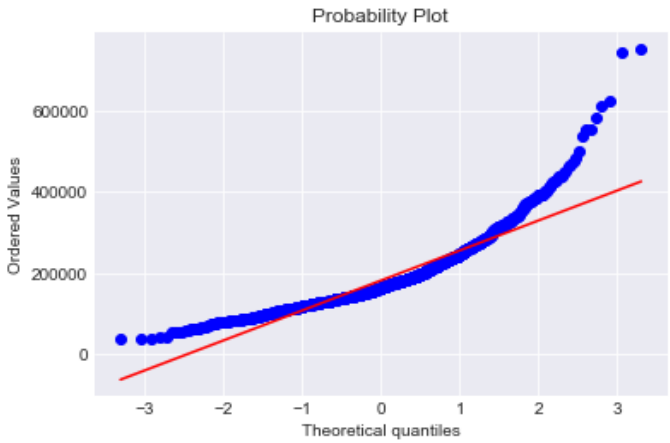


Figure3. Original Probability Plot

This non-normal distribution shows a trailing trend, namely that most of the samples are clustering around the lesser values. Therefore, we can employ the cox-box method to handle this problem.

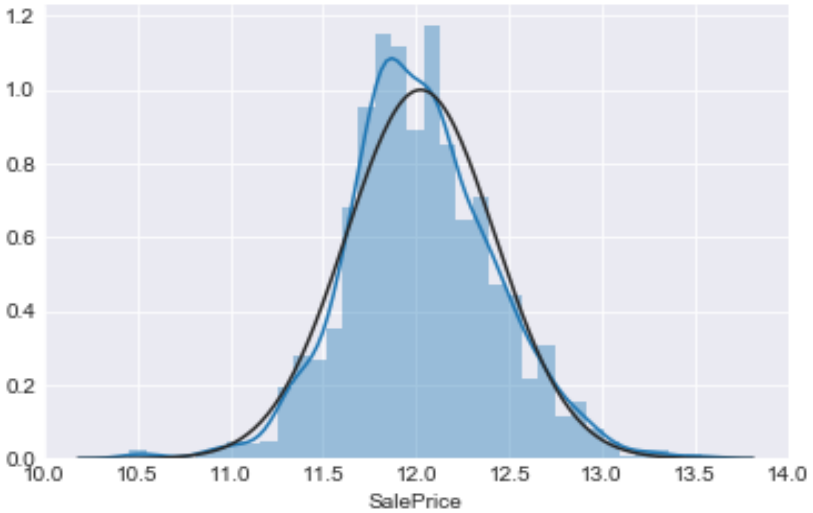
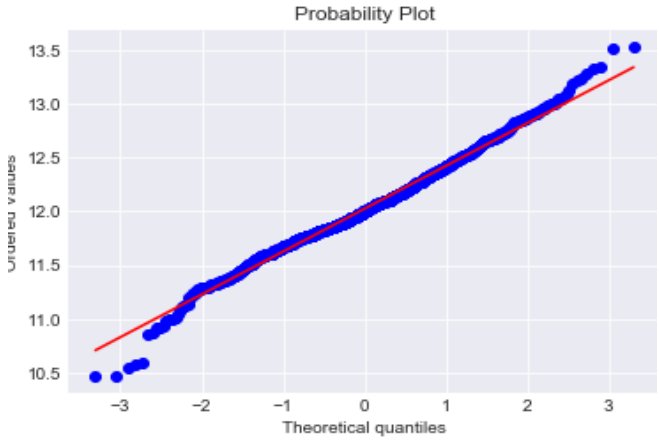


Figure4. Distribution of the Sale Price after Normalization



Original Figure 5. Probability Plot after Normalization

We can observe that the samples are close to the normal distribution after using the cox-box method.

3.2 Missing Values

The next important process is dealing with missing values. In the code, it is easy to see that 30 columns contained missing values. We dealt with the missing values as the following table shows.

	Missing Ratio
PoolQC	99.657417
MiscFeature	96.402878
Alley	93.216855
Fence	80.438506
FireplaceQu	48.646797
LotFrontage	16.649538
GarageQual	5.447071
GarageCond	5.447071
GarageFinish	5.447071
GarageYrBlt	5.447071
GarageType	5.378554
BsmtExposure	2.809181
BsmtCond	2.809181
BsmtQual	2.774923
BsmtFinType2	2.740665
BsmtFinType1	2.706406
MasVnrType	0.822199
MasVnrArea	0.787941
MSZoning	0.137033
BsmtFullBath	0.068517
BsmtHalfBath	0.068517
Utilities	0.068517
Functional	0.068517
Electrical	0.034258
BsmtUnfSF	0.034258
Exterior1st	0.034258
Exterior2nd	0.034258
TotalBsmtSF	0.034258
GarageArea	0.034258
GarageCars	0.034258

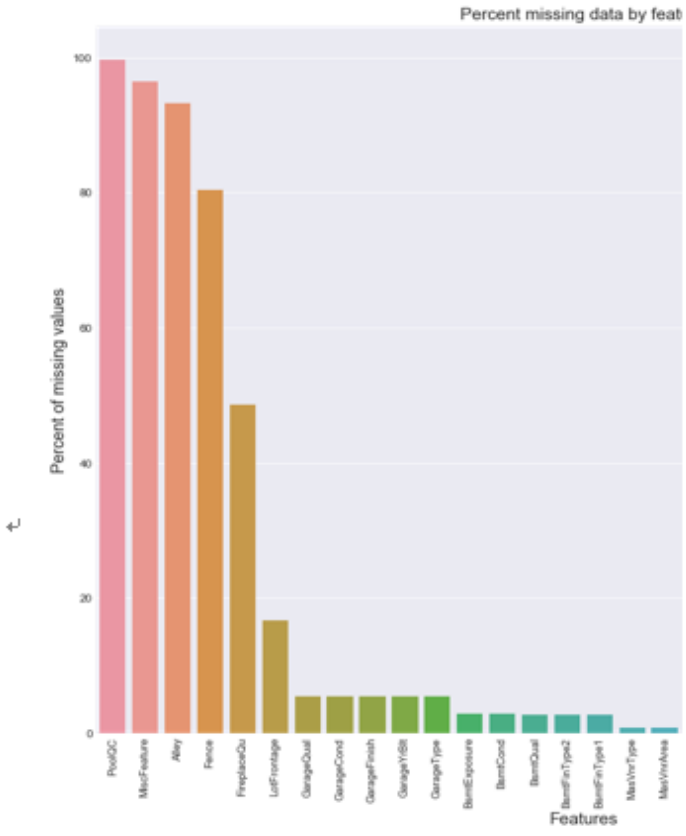


Figure6. Missing Values

Table1. Imputation of the Missing Values

Method	Feature
None	PoolQC, Alley, Fence, FireplaceQu, MiscFeature, GarageType, GarageFinish, GarageQual, GarageCond, BsmtQual, BsmtCond, BsmtExposure, BsmtFinType1, BsmtFinType2, MasVnrType, Functional
0	GarageYrBlt, GarageArea, GarageCars, BsmtFinSF1, BsmtFinSF2, BsmtUnfSF, TotalBsmtSF, BsmtFullBath, BsmtHalfBath, MasVnrArea
Most Frequent Class	MSZoning, Electrical, KitchenQual, Exterior1st
Median Replacement	LotFrontage
Drop	Utilities

For the “None” group, which includes features such as ‘PoolQC’, ‘MiscFeature’, ‘Alley’, ‘Fence’, ‘FireplaceQu’, etc, missing values perhaps occur because most of the houses don’t have these facilities, which may cause ‘missing’. It is therefore safe to fill the NA with “None”.

Secondly, for those houses with no basement, garage, or wall cladding, this may also cause missing values to occur, as can be seen in the features 'GarageYrBlt, 'BsmtFinSF1', 'BsmtFinSF2', etc. So we fix them with the value 0.

By observing the feature ‘MSZoning’, 'Electrical', 'KitchenQual', 'Exterior1st', 'SaleType', 'Exterior2nd', we can observe that all of them frequently show a value which is not determined by the sample’s other features. Therefore it’s a good idea to fill these missing values with the most frequent value within the sample.

The ‘LotFrontage’ usually shows a definite value in a definite area, thus we can simply fill in this feature’s missing value with a median value of this area which is grouped by ‘Neighborhood’.

For the feature ‘Utilities’, all records are "AllPub", except for one "NoSeWa" and 2 NA . Since the house with 'NoSewa' is in the training set, this feature won't benefit our predictive model. We can then safely remove it for that very reason.

We have officially filled in all the missing values.

3.3 Correlation Analysis

In order to analyze the correlation between each feature and the housing price, we plotted the correlation heat map of these features. We discovered that some numerical features, which are correlate highly with the price. In particular, there are strong positive correlations between “TotaLBsmtSF” and “1stFlrSF”, as well as “GarageAreas” and “GarageCars”. Such information is useful in determining the correlation of features and offering evidences beneficial to performing imputations. We also know that multi-collinearity may make it more difficult to make inferences about the relationships between our independent and dependent variables.

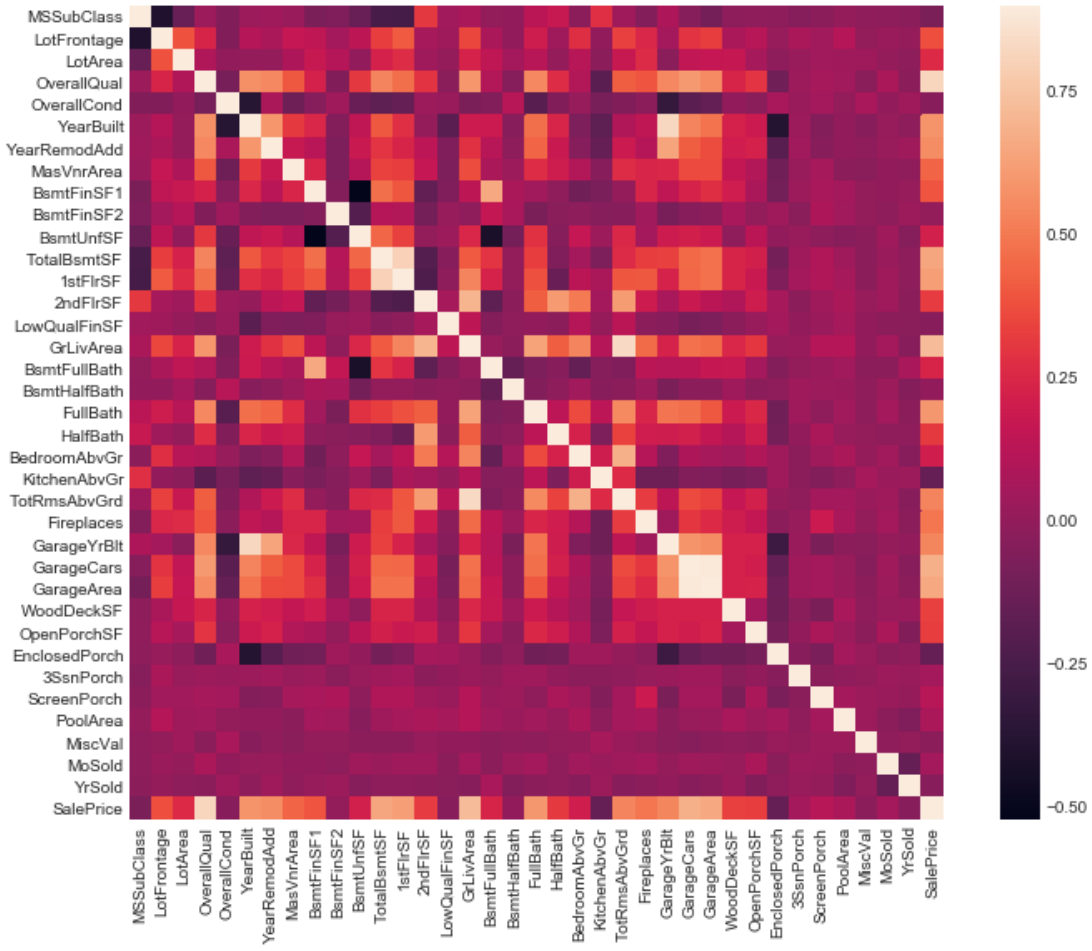


Figure7. Correlation Before Imputation

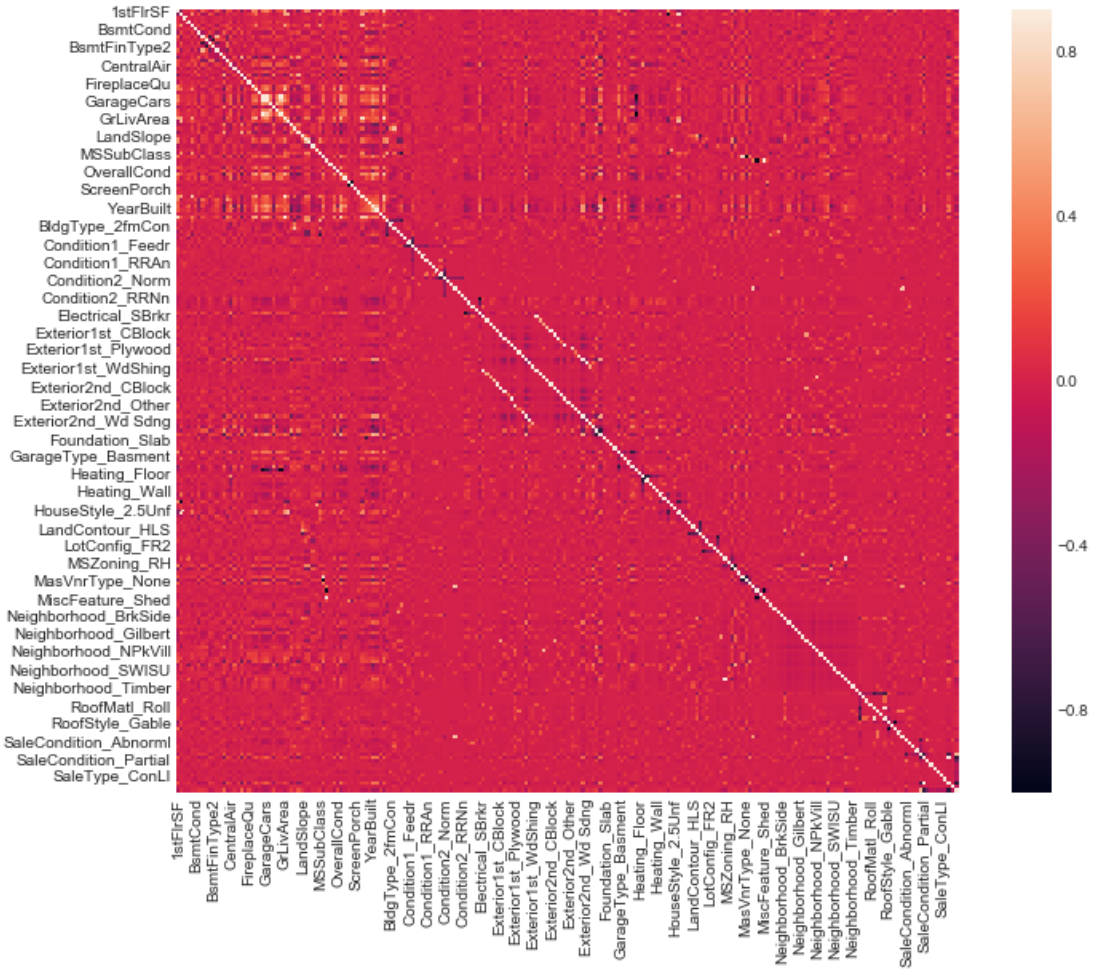


Figure8. Correlation After Imputation

3.4 Feature Engineering

3.4.1 TRANSFORMING THE DATA TYPE

In the algorithm to be applied next, the data type should be legal, which means that we should transform some of this data.

In one method, we transferred some numeric features to categorical types including 'MSSubClass', 'OverallCond', ‘'YrSold', 'MoSold'.

Another way, which allows for categorical features with three or more classes, is to employ the method of ‘LabelEncoder’ to transform them, which includes 'FireplaceQu', 'BsmtQual', 'BsmtCond', 'GarageQual', 'GarageCond', 'ExterQual', 'ExterCond','HeatingQC', 'PoolQC', 'KitchenQual', 'BsmtFinType1', 'BsmtFinType2', 'Functional', 'Fence', 'BsmtExposure', 'GarageFinish', 'LandSlope','LotShape', 'PavedDrive',

'Street', 'Alley', 'CentralAir', 'MSSubClass', 'OverallCond', 'YrSold', 'MoSold'. At last we implement ‘One-Hot-Encoding’ in order to make dummy variables out of all the categorical data which will have better effects on the algorithms.

3.4.2 ADDING A NEW FEATURE

We know that the mean house price for different parts of the house is the same. In this way, we can combine the features 'TotalBsmtSF', '1stFlrSF', and '2ndFlrSF' into a new feature, namely 'TotalSF'.

3.4.3 FEATURE SKEWNESS

In most of the algorithms, data with Gauss distribution can minimize error. Thus, we need to discover the skewing feature and then correct it by calculating the relative asymmetry of the data.

	Skew
MiscVal	21.939672
PoolArea	17.688664
LotArea	13.109495
LowQualFinSF	12.084539
3SsnPorch	11.372080
LandSlope	4.973254
KitchenAbvGr	4.300550
BsmtFinSF2	4.144503
EnclosedPorch	4.002344
ScreenPorch	3.945101

Figure9. Skewness of the Feature

We select the top 10 skewing features, all of which display high rates of skewness. For all of the numerical data within the data set, we set 0.75 as the threshold of the skewness. In this condition, we come to find that we have 59 features in total to be corrected. By histograming the density map of the features, we can employ box-cox as our method to correct the skewing (LAM = 0.15).

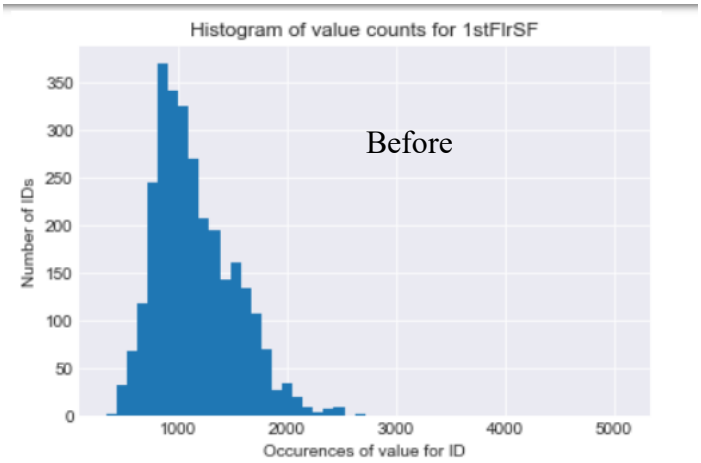


Figure10. Skewness of 1stFirSF Before Imputation

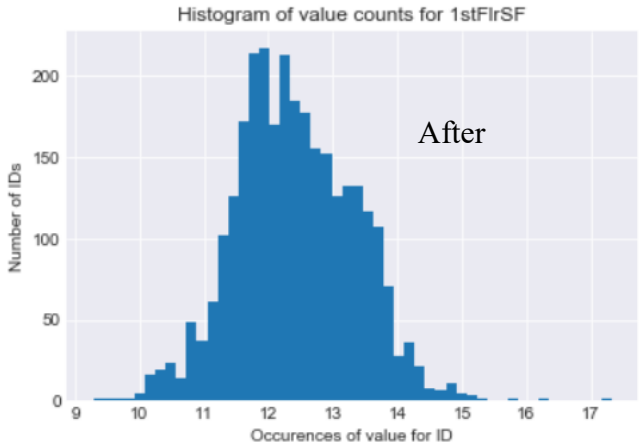


Figure11. Skewness of 1stFirSF after Imputation

4. Models

4.1 Elastic Net

In statistics, and in particular the fitting of linear or logistic regression models, the elastic net is a regularized regression method that linearly combines the lasso and ridge methods.

There are two parameters we need to tune for: alpha, l1_ratio.

Alpha: the constant that multiplies the penalty terms

l1_ratio: the ElasticNet mixing parameter, with 0 <= l1_ratio <= 1. For l1_ratio = 0, the penalty is an L2 penalty. For l1_ratio = 1 it is an L1 penalty. For 0 < l1_ratio < 1, the penalty is a combination of L1 and L2.

Based on the process, we decided: Alpha[0.001,0.1], l1_ratio[1,2]

We then added the new parameters “random_state”, the seed of the pseudo random number generator that selects a random feature to update.
We used the MSE in order to judge the score and to discover the most ideal parameters.

4.2LightGBM:

LightGBM is a gradient boosting framework that employs tree based learning algorithms. It is designed to be distributed and efficient, and has the following advantages: faster training speed and higher efficiency, lower memory usage, better accuracy, is parallel and GPU learning supported, and is capable of handling large-scale data.
There are nine parameters to tune

Table2. LightGBM

parameter	note	range	Chosen value
num_leaves	The main parameter to control the complexity of the tree model. It should be smaller than 2^(max_depth) to prevent over-fitting	$(1,+\infty)$	[5,8]
learning_rate	Shrinkage rate and in dart, it also affects the normalization weights of dropped trees.	$[0,+\infty)$	[0.01,0.1]
n_estimators	Iterations or how many trees. Too often will lead to over-fitting and too seldom will lead to decreased accuracy	$(0,+\infty)$	[500,3000]

max_bin	(1) Max number of bins that features values will be bucketed in (2) Min number of bins may reduce training accuracy but may increase general power (thus dealing with over-fitting)	$(1, +\infty)$	55
bagging_fraction	(1) Like feature_fraction, but this will randomly select part of the data without resampling (2) Can be used to speed up training (3) Can be used to deal with over-fitting (4) ***Note:*** to enable bagging, bagging_freq should be set to a non zero value as well	(0,1]	[0.6,0.8]
bagging_freq	(1) frequency of bagging (2) 0 means disable bagging; k means perform bagging at every k iteration (3) ***Note:*** to enable bagging, bagging_fraction should be set to a value less than 1.0	-	[5,10]

feature_fraction	(1) LightGBM will randomly select part of the features within each iteration if feature_fraction is lesser than 1.0. E.g. if you set it to 0.8, LightGBM will select 80% of features before training each tree (2) can be used to speed up training (3) can be used to deal with over-fitting	(0,1]	[0.2,0.5]
feature_fraction_seed	random seed for feature_fraction	-	[5,10]
bagging_seed	random seed for bagging	-	[5,10]
min_data_in_leaf	minimal number of data in one leaf. Can be used to deal with over-fitting	$[0,+\infty)$	[6,8]
min_sum_hessian_in_leaf	minimal sum hessian in one leaf. Like min_data_in_leaf, it can be used to deal with over-fitting	$(0,+\infty)$	[10,15]

4.3GBM

The next step is to apply gradient boosting. There are 6 tuning parameters: n_estimators, learning_rate, max_depth, max_features, min_samples_leaf, min_samples_split.

Table3. GBM

parameter	note
n_estimators	The number of sequential trees to be modeled.
learning_rate	This determines the impact of each tree on the final outcome. GBM works by starting with an initial estimate which is updated using the output of each tree. The learning parameter controls the magnitude of this change in the estimates.
max_depth	The maximum depth of a tree.
max_features	The number of features to consider while searching for the best split.
min_samples_leaf	Defines the minimum samples (or observations) required by a terminal node or leaf
min_samples_split	Defines the minimum number of samples (or observations) which are required by a node to be considered for splitting.

There are two types of parameter that we need to tune here, tree based and boosting parameters. We knew that there were no optimum values for learning rates as low values always work better, given that we trained on a sufficient number of trees. While GBM is robust enough to prevent over-fitting within an increased number of trees, a high number for a particular learning rate can still lead to over-fitting. As we reduced the learning rate and increased trees, the computation took a long time to run and produce results.

In such a case, we adopted the following approach. Initially, we used 0.05, however it did not produce a favorable result. Thus, our second attempt saw us choosing the default value of 0.1. Meanwhile, we lowered the learning rate and increased the estimators proportionally in order to produce more robust models. Additionally, it is worth mentioning that you should pay careful attention to the order of tuning variables. For example, the parameters that exert greater impacts on the result should be tuned firstly, such as max_depth and min_samples_split, and then you could tune min_samples_leaf and max_features.

4.4 KernelRidge

Let's have a look at our liner machine learning model, Kernel Ridge Regression (KRR). KRR combines ridge regression (linear least squares with l2-norm regularization) with the kernel trick. It thus learns and adapts a linear function in the space induced by the respective kernel and the data. For non-linear kernels, this corresponds to a non-linear function within the original space. In our case, there were four parameters, including alpha, kernel, degree and coef0. The table below is their definitions.

Table 4. KernelRidge

parameter	note
alpha	Small positive values of alpha improve the conditioning of the problem and reduce the variance of the estimates. Alpha corresponds to (2*C)^-1 in other linear models such as LogisticRegression or LinearSVC. If an array is passed, penalties are assumed to be specific to the targets. Hence they must correspond in number.
kernel	Kernel mapping is used internally. A callable should accept two arguments and the keyword arguments passed to this object as kernel_params, and should return a floating point number.
degree	Degree of the polynomial kernel. Ignored by other kernels.
coef0	Zero coefficient for polynomial and sigmoid kernels. Ignored by other kernels.

4.5 XGBoost

We choose XGBoost as one of our base models. In this case, we mainly tuned nine features: colsample_bytree, gamma, learning_rate, max_depth, min_child_weight, n_estimators, reg_alpha, reg_lambda, subsample. Introduction to the parameters:

Table5. XGBoost

parameter	note
Colsample_bytree	Colsample_bytree denotes the fraction of columns to be randomly sampled for each tree.
Gamma	Gamma specifies the minimum loss reduction required to make a split.
Learning_rate	Learning_rate determines the impact of each tree on the final outcome. XGBoost starts with some initial estimates and the learning_rate controls the further change.
Max_depth	Max_depth is the max depth of a tree.
Min_child_weight	Min_child_weight defines the sum of weights of all observations required in a child.
N_estimators	N_estimators is the number of sequential trees to be modeled.
Reg_alpha	Reg_alpha represents the L1 regularization term which is helpful to control over-fitting.
Reg_lamda	Reg_lamda represents the L2 regularization term which is helpful to control over-fitting.
Subsample	Subsample denotes the fraction of observations to be randomly sampled for each tree

Over the course of this project, we employed the BayesianOptimization in order to determine the rough interval. After that, we used the grid search method to choose the elaborate parameter values of n_estimators, max_depth, colsample_bytree, subsample, learning_rate and others in order to minimize any chance of error.

In employing the single XGB model to predict the test data set, we discovered a considerable bias between the training set and the testing set. We attempted to tune the parameters responsible for controlling over-

fitting, but the result was unsatisfactory. For this reason, we employed the XGB predictions in the ensemble strategy while giving XGB a lower weight.

4.6 Lasso

Lasso is a powerful technique generally used for creating parsimonious models in the presence of a large number of features. The loss function is based on the 'Least squares'. Lasso adds the L1 penalty in order to penalize a large number of coefficients. What's more, it can also minimize the coefficient to 0 when certain features fail to work within the model.

We used the grid search method to determine the parameter's alpha value. Lasso is a linear method, and therefore it can be used to enhance the generalization of the model in the sections of the samples where the price is highly related to the area of the house. But the actual model is so much more complicated than that, and thus we chose to use Lasso as the meta model for the stacking model, and the implementation has been a success, ultimately producing very accurate predictions.

4.7 Summary

Table6. Summary of all Models

Model	Description	Pros	Cons
Lasso	• Lasso regression is a powerful techniques generally used for creating parsimonious models in the presence of a 'large' number of features	• Adds a regularization term which restrains the efficiency brought about by the increased data size. • The regularization term can minimize to zero in order to totally move the non-relative features.	• Sensitive to the outliers, which might cause over-fitting.
Elastic Net	• In statistics, and in the fitting of linear or logistic regression models in particular, the elastic net is a regularized regression method that linearly combines the L1 and L2 penalties of the lasso and ridge methods	• Enforces sparsity • No limitation on the number of selected variable • Encourages a grouping effect in the presence of highly correlated predictors	• Native elastic net suffers from double shrinkage

KRR	• Kernel ridge regression (KRR) combines ridge regression (linear least squares with l2-norm regularization) with the kernel trick. It thus learns a linear function within the space induced by the respective kernel and the data. For non-linear kernels, this corresponds to a non-linear function within the original space.	• Strong theoretical guarantees. • Generalization to outputs in the single matrix inversion • Uses kernels	• Solution not sparse. • Training time for large matrices is a low-rank approximations of kernel matrix,
GBM	Produces a prediction model in the form of an ensemble of decision trees Builds the model in a stage-wise fashion Generalizes results by allowing for the optimization of an arbitrary differentiable loss function	High-performing Uses all features Lower possibility of over-fitting with more trees	Susceptible to outliers. Lack of interpretability and higher complexity. Harder to tune the parameters than other models. Slow training speed.

LightGBM	• An advanced implementation of gradient boosting. • Uses a more regularized formalization to control over-fitting which gives a better performance within the model.	□ Faster training speed and higher efficiency □ Lower memory usage □ Better accuracy □ Parallel and GPU learning supported □ Capable of handling large-scale data	□ Easily over-fitted
XGBOOST	• An advanced implementation of gradient boosting. • Uses a more regularized formalization to control over-fitting which produces a better performance within the model.	• Uses regularization to reduce over-fitting. • Support parallel processing. • Allows for splits up to the max_depth specified and then starts pruning the tree backwards and removes splits beyond which there is no positive gain. • Built-in cross validation.	• Susceptible to outliers. • Lack of interpretability and higher complexity. • Harder to tune the parameters than other models. • Slow training speed.

5. Averaging Based Ensemble Methods and Stacked Ensemble Methods

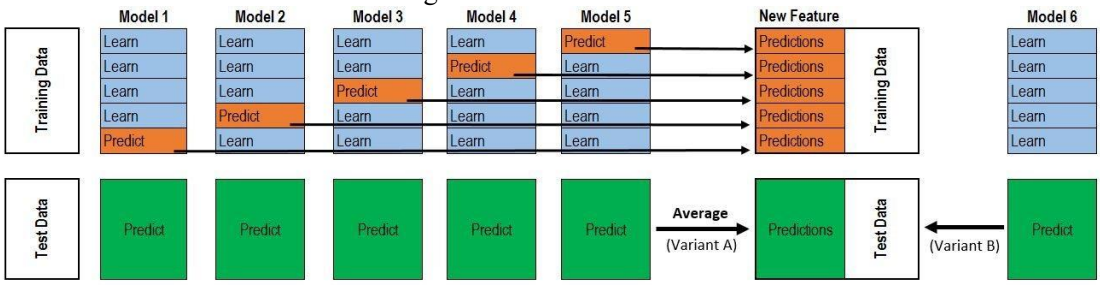
The implementation of only one model cannot independently produce the best score, so we decided to ensemble all the models in order to figure out the best way to predict the outcomes.

5.1Simple averaging model

Within the simple averaging method, for every sample of test datasets, the average predictions are calculated. This method often reduces over-fitting and creates a smoother regression model. The scoring method is MSE, and it turn out that the more linear the regression, the better the score is. Ultimately, the best score was produced by the combined efforts of lasso, elastic net, KRR, and GBoost

5.2 Stacked:

Figure12. Stacked Method



As you can see from the two layers of stacking within the figure above, the first layer has five models and the second has only one. The role of the first layer model is to train an Rn×m feature matrix for inputting the second layer model training, where n is the number of training data rows and m is the number of first layer models. We used lasso for the second model and the other models for the first layer. If the first layer has less than five models, we can ensemble the left model by changing the weight. We employed the following equation: “ensemble = stacked_pred*0.70 + averaged_pred*0.2 +xgb_pred*0.1”, and after careful processing, the results were as follows: the first layer is made up of KRR, elastic net, and GBoost, and the second layer is lasso. In this way, the best score is achieved.

6. Final score

House Prices: Advanced Regression Techniques

Predict sales prices and practice feature engineering, RFs, and gradient boosting

4,414 teams · Ongoing

House Prices: Advanced Re...

Ongoing - Top 2%

78th

of 4413
