Team name: Bai Lan Le
Team mates & Kaggle IDs: Jiqiong Tang(Jiqiong(Janet) Tang), Jiaxin Wu(JiaxinWu778), Xinhui Cui(XINHUICUIKKKK), Yuefan Zhai(Yuefan Zhai)
Score on the private leaderboard: 20342.77484

495 FINAL Report

Ⅰ. Introduction

In the kaggle competition, students are asked to predict the housing Sale Price on the test.csv. This report explains how our team uses machine learning models to achieve the purpose, and it is elaborated in four parts: introduction, data preparation, training and model selection, and conclusion.

It's notable that, in order to make our models work well,we do a lot of data cleaning. But in the end, we find a very powerful package that simply deals with raw data produces the best result. Till then, it seems like all the data preparation process is meaningless. However, it's exactly after all the painful and cumbersome work on data preparation and model selection, we realize how powerful the package is. Therefore, we decide to present the whole data preparation process and explain how we find the best model step by step.

Ⅱ. Data preparation

The whole train data set has 79 features and 2052 records, which includes 36 numeric features and 43 text features. The value we intend to predict is the sale price of houses, not fitting a normal distribution. Thus, we use the log function to transform it for better performance of the linear models.

In order to simplify the data processing, we merge the train and test data first. It should be noticed that changes can be made on test data, but we can not drop any data in the test dataset. So the merge should happen after we drop outliers in train data. Then we analyze the numeric features and text features respectively and preprocess them by filling missing values, dropping features, normalizing and encoding.

For numeric features, we first drop the outliers. It's believed that sales price has a positive relationship with 'living area', thus we consider houses with a living area more than 4000 sold for less than 200,000 seem to be unreasonable. Then we fill all the nan with the number 0 rather than the mean value. In this case, many missing values illustrate the exact number of features such as the number of bedrooms. The mean value is highly possible to be a decimal, which is unreasonable if filled in. Year-related data are transferred into strings to make it reveal the information of time but not number. What's more, dropping insignificant numeric features or sequentially related factors is necessary. We consider this part by correlation between sale price and predictors, and correlation between predictors themselves. As seen in the heat map followed, the lighter the color, the higher the correlation.

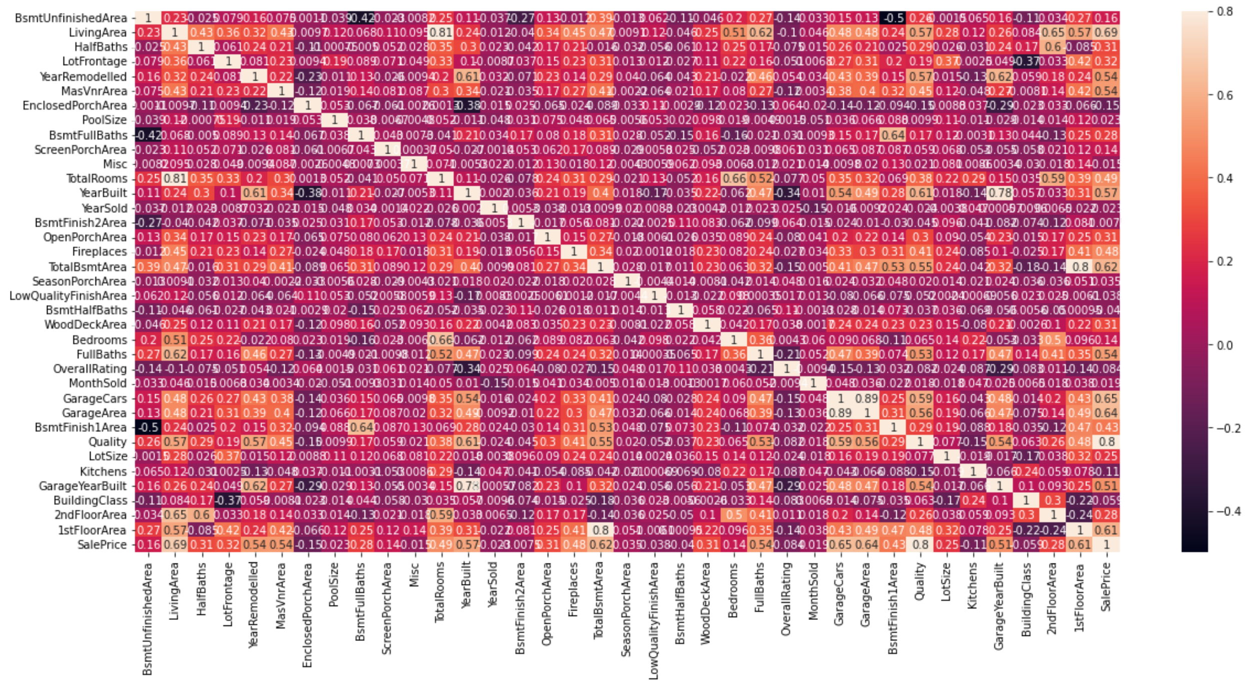
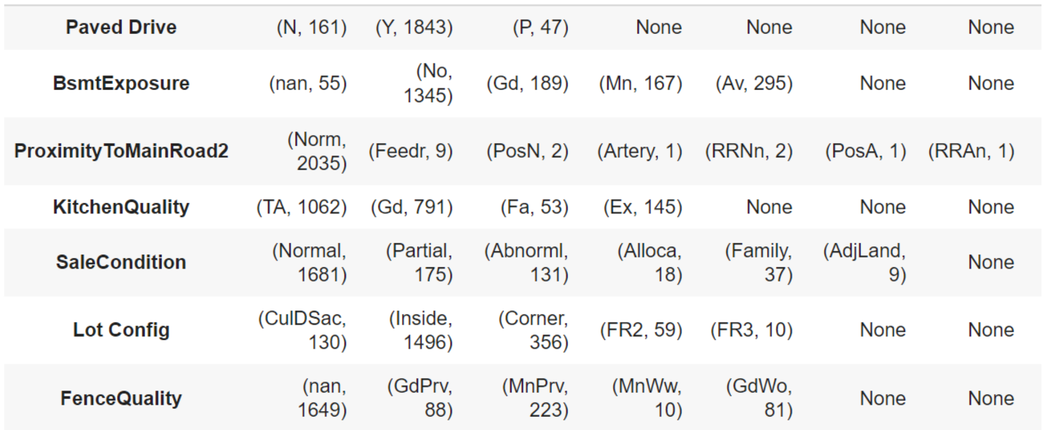


Exhibit 1 Correlation Heat Map

Features whose absolute value of correlation coefficient with the sale price are less than 0.1 are considered not important and deleted. In order to minimize the multiple collinear problem in regression, we calculate the correlation matrix between each feature and only retain one of the pairs of features with correlation coefficients more than 0.65 (absolute value). Additionally, the new feature named “TotalSF” is added by the following 3 columns, “TotalBsmtArea”, “1stfloorArea”, and “2rdFloorArea”. It is reasonable to take the total size of houses as an indicator of the sale price. At last, based on the assumption of normal distribution, we normalize the features with skewness higher than 0.75 (absolute value) for better fit of training models.

For text features, our first step is handling the missing value. Here, simply dropping “no” and “none” values seems to be feasible, but in fact unreasonable, since there are some instances where “no” is exactly the answer. For example, some houses don’t have a garage. Thus, we sort out features with the largest proportion value being “No”, and then fill them with “No”. We find that the column “Electrical” only has one missing value, so we fill it with the mode. And the remaining missing values are filled with “None”. Secondly, we need to transfer some text features into numeric features by label encoding. We import the collections function to get a statistical summary of the composition of each feature, creating a dataframe which shows pairs of unique values and their frequency.



Paved Drive	(N, 161)	(Y, 1843)	(P, 47)	None	None	None	None
BsmtExposure	(nan, 55)	(No, 1345)	(Gd, 189)	(Mn, 167)	(Av, 295)	None	None
ProximityToMainRoad2	(Norm, 2035)	(Feedr, 9)	(PosN, 2)	(Artery, 1)	(RRNn, 2)	(PosA, 1)	(RRAn, 1)
KitchenQuality	(TA, 1062)	(Gd, 791)	(Fa, 53)	(Ex, 145)	None	None	None
SaleCondition	(Normal, 1681)	(Partial, 175)	(Abnorml, 131)	(Alloca, 18)	(Family, 37)	(AdjLand, 9)	None
Lot Config	(CulDSac, 130)	(Inside, 1496)	(Corner, 356)	(FR2, 59)	(FR3, 10)	None	None
FenceQuality	(nan, 1649)	(GdPrv, 88)	(MnPrv, 223)	(MnWw, 10)	(GdWo, 81)	None	None

Exhibit 2 Part of Summaries of Unique Values

Besides, we plot boxplots, part of them shown as follows, to see the five-number summaries of each text feature and to analyze the central values of each group.

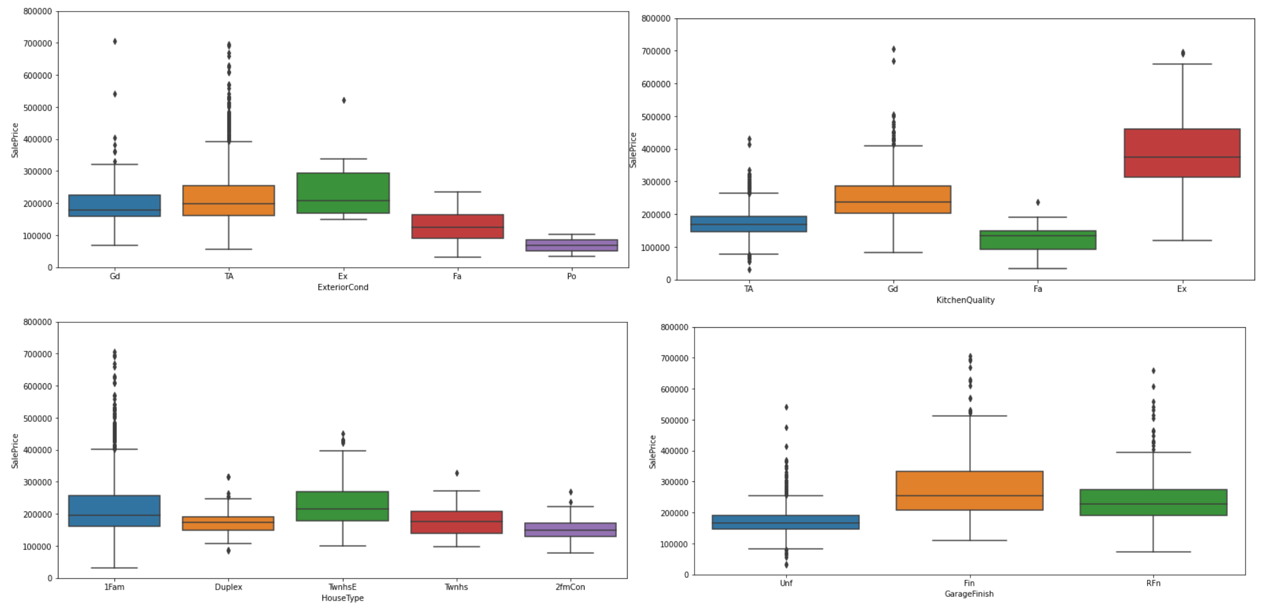


Exhibit 3 Box Plots Samples

After sorting and comparing the values, we find 22 features that are able to be ranked by some standards like quality, condition, type, and to be labeled into 3-5 classes. Therefore, we import the labelencoder function from sklearn to labelize these features, then each text data will transform into a number. Finally, we use a one-hot encoding function to convert categorical data variables.

Ⅲ.Training and Model Selection

After data preparation, we split the dataset into the training and test set, following that, we use five models to do the prediction and choose the one with the best result.

3.1 ElasticN, Linear Regression & Random Forest

At first we try some single models for prediction. For the ElasticN model, we import the package from sklearn.linear_model, and select the best parameter alpha that gets the highest score among the recommended alpha range in the official document. For the Linear Regression model, we simply import LinearRegression and run the model after fitting the train set. For the random forest model, we import RandomForestRegressor from sklearn.ensemble, do the hyperparameter tuning with the variable ‘max_features’ and train. Then, we can get the output of the three models. We are not satisfied with the results, so we move forward to our next model.

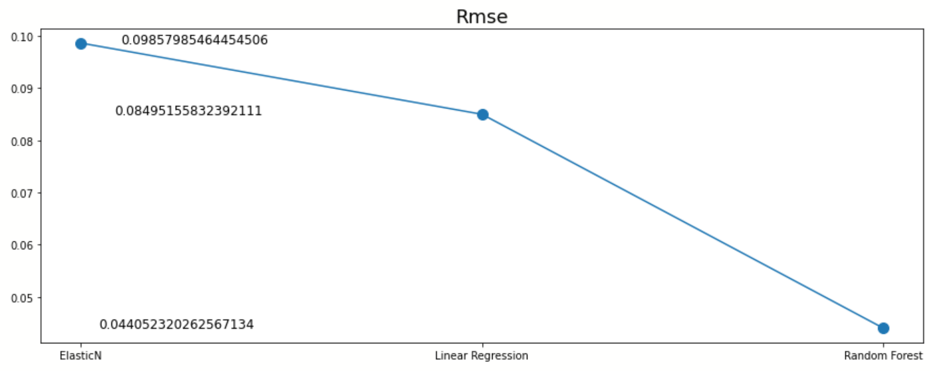


Exhibit 4 Rmse Scores

3.2 Stacking

After several trials with the models as we mentioned, we supposed that only one model may never produce a satisfying result. So we tried to find some models that integrate the advantages of several models. Stacking seemed to be what we wanted. The working process is as follows:

To begin with , it applies to 5 models including Lasso Regression, ElasticNet, Gboost, XGboost, and LightGBM. Secondly, it uses 5-fold cross validation and RMSE score to see which one has the best performance. Besides, the stacking model has two layers, and we combine Enet, Gboost and LightGBM in the first layer; we use Lasso regression in the second layer which has the function of avoiding overfitting. At last, we output the result and get RMSE score. According to our referenced website, some authors give different weights toward models, but we skip this step because we find no solid grounds to select the weights and justify ourselves.

The stacking model optimizes our result greatly, but we didn’t stop there. We found that the more data preparation we did, the better the score on the public board was. But as you may expect, we quickly realized that it leads us way closer to the overfitting problem. So we kept searching for a simpler model without much need for data preparation.

3.3 Autogluon

Autogluon is a very powerful package we found, which compares tens of packages inside itself and picks up the best one to best train automatically. So it's highly efficient and persuasive. Besides, it can run with nearly raw data. It saves the data preparation process and also avoids overfitting. On the other hand, it has one small drawback: this model needs a little bit more time and CPU space to run. So we just prompt one variable 'Lot Size' with extremely high variance and take the log to train the regression model. For this specific project in Kaggle, the best model that Autogluon chose for us was WeightedEnsemble_L2.

Ⅳ. Conclusion

Autogluon, as we have expected, produces the best result. We're amazed that only tens of codes can produce such astonishing results. At the same time, we know clearly that that engineer must have put a lot of effort perfecting his package. We admire his effort and contribution, and we also realize how important it is to refer to other powers appropriately.

We have to admit there are some imperfections in our coding and reports. For example, we failed to make much progress on Autogluon package because it is hard to find the balance between data processing and the model fitting. We know our knowledge and ability are highly limited, but we're enchanted by the magic of machine learning and we're inspired to improve our coding skills to a next level in the future.

References

[1] AutoGluon: Deep Learning AutoML,
https://towardsdatascience.com/autogluon-deep-learning-automl-5cdb4e2388ec

[2] Beginner's Guide to AutoML with an Easy AutoGluon Example,
https://www.analyticsvidhya.com/blog/2021/10/beginners-guide-to-automl-with-an-easy-autogluon-example/

[3] Simple Model Stacking, Explained and Automated,
https://towardsdatascience.com/simple-model-stacking-explained-and-automated-1b54e4357916

[4] Stacking Ensemble Machine Learning With Python,
https://machinelearningmastery.com/stacking-ensemble-machine-learning-with-python/

[5] Create a model to predict house prices using Python,
https://towardsdatascience.com/create-a-model-to-predict-house-prices-using-python-d34fe8fad88f