Project Report: House Prices Prediction with Advanced Regression Techniques on Kaggle

Author: Jinhang Jiang, Maren Olson

Contributors: Andrew Aviles, Tzu Hsuan Chang

Master of Science in Business Analytics at Arizona State University 2020 - 2021

Content Table

Project Summary

This project provided by Kaggle included a dataset of 79 explanatory variables describing aspects of residential homes in Ames, Iowa. You may find all the data sources and description of the project at https://www.kaggle.com/c/house-prices-advanced-regression-techniques.

The goal of this competition was to predict the final sales price for each home listed. We did this problem because we were looking to apply our regression techniques as well as our stacking and exploratory data analysis skills. This problem set allowed us to use all of the above practices. Moreover, the datasets contain many missing values, which offered us the opportunity to gain experience in handling missing data.

We ran multiple regression techniques such as XGB regressor, SGD regressor, MLP regressor, Decision Tree regressor, Random Forest regressor, CatBoost regressor, Light GBM, and SVR. We stacked some together and experimented to see which ones ended up with the smallest Root-Mean-Squared-Error.

The prediction that we submitted was two columns (Figure 1.1). The first column was the ID of the house and the second column was our predicted sales price. The prediction is scored based on the Root-Mean-Squared-Error between the predicted value logarithm and the log of the observed sales price. The smallest RMSE was the best prediction. After running base models and the tuned models, we found that our tubed CatBoost regression model got us the lowest score of 0.12392 of RMSE, which was 863rd place (out of 4525) on the Leaderboard.

Id	SalePrice
1461	124787.5
1462	165017.6
1463	182598.2
1464	189458.4
1465	185829.3

Figure 1.1 - Sample of Submission

Data

- Overview

The competition offered us two datasets to work on. One of them is the training data with 1460 observations and 81 columns, which contains each house's ID and the sale price. The other dataset is the holdout file, which includes 1459 observations.

We first looked into the types of features of the training dataset. Figure 2.1 shows that the whole dataset has 43 categorical features and 36 numeric variables, including "ID".

```
print(house.dtypes.astype(str).value_counts())
```

```
object     43
int64      35
float64     3
```

Figure 2.1 - The Numbers of Numeric & Categorical Features

Then, we took the efforts to detect if there were any missing values in the dataset. In the training dataset, 19 variables have missing values. The holdout dataset has 33 features with missing values. Some features have very high percentages of missing values. For example, the feature, PoolQC (Pool Quality), has as high as 99.5% missing values. Features like PoolQC, may hurt the accuracy of the prediction model and lead us to invalid conclusions. So, we removed the features with at least 80% of the values that are missing. The details are explained in the Data Processing section.

Before we started to process the data, we also studied the necessary statistical information of the 38 numeric features. In Figure 2.2, we found out that the ranges of some features, like the SalePrice, are relatively large. Many regression models we were going to use would need us to do some types of transformations of those features before processing.

ScreenPorch	PoolArea	MiscVal	MoSold	YrSold	SalePrice
1460.000000	1460.000000	1460.000000	1460.000000	1460.000000	1460.000000
15.060959	2.758904	43.489041	6.321918	2007.815753	180921.195890
55.757415	40.177307	496.123024	2.703626	1.328095	79442.502883
0.000000	0.000000	0.000000	1.000000	2006.000000	34900.000000
0.000000	0.000000	0.000000	5.000000	2007.000000	129975.000000
0.000000	0.000000	0.000000	6.000000	2008.000000	163000.000000
0.000000	0.000000	0.000000	8.000000	2009.000000	214000.000000
480.000000	738.000000	15500.000000	12.000000	2010.000000	755000.000000

Figure 2.2 - Statistical Info of All The Numeric Variables

- Data Processing

Missing values always hurt the accuracy of the prediction models. Before we did any transformations, we decided to check out for the columns with at least 80% of values missing

first. According to Figure 2.3, there are four variables with a large number of missing values, and they were “Alley”, “PoolQC”, “Fence”, and “MiscFeature”.

```
In  [7]: nullvalue=house.isnull().sum()
         nullvalue.where((nullvalue/1460)>0.8).dropna().astype(int)

Out[7]: Alley          1369
        PoolQC         1453
        Fence          1179
        MiscFeature    1406
        dtype: int32

In  [8]: nullvalue=holdout.isnull().sum()
         nullvalue.where((nullvalue/1460)>0.8).dropna().astype(int)

Out[8]: Alley          1352
        PoolQC         1456
        Fence          1169
        MiscFeature    1408
        dtype: int32
```

Figure 2.3 - The Variables With At Least 80% Missing Values

Making sure we made the same changes to the training dataset and holdout dataset was essential in processing the data. The difference in the number of columns in each dataset, inconsistent data formats, and the unmatched number of values in the categorical variables would all possibly cause troubles to our models. Therefore, to better prepare the data and make sure any transformations will reflect on both the training dataset and holdout dataset, we used the code in Figure 2.4 to combine those two datasets together temporarily with labels 0 and 1, and the code for splitting the dataset back to the original dataset is also provided.

```
combined_data = pd.concat([house, holdout], keys=[0,1])
train = combined_data.xs(0)
test = combined_data.xs(1).drop(['SalePrice'],axis=1)
```

Figure 2.4 - Code for Combine and Uncombine the Datasets

For all the categorical variables, we converted them to dummy variables to deal with the missing values. For all the missing values in the numeric variables, we filled them with the average of the current variable.

- Feature Engineering

We also created a few new variables based on the existing variables. For example, we used the data about when the house was renovated and when it was sold to generate a new variable called “AgeofHouse”, which helped us intuitively understand the relationship between the house prices and the quality of the house.

```
combined_data["AgeofHouse"]= combined_data["YrSold"] - combined_data["YearRemodAdd"]
print(combined_data[["AgeofHouse","YrSold","YearRemodAdd","SalePrice"]].head())
```

	AgeofHouse	YrSold	YearRemodAdd	SalePrice
0 0	5	2008	2003	208500
1	31	2007	1976	181500
2	6	2008	2002	223500
3	36	2006	1970	140000
4	8	2008	2000	250000

Figure 2.5 - New Feature “AgeofHouse”

We wondered if we needed to do a log transformation for the “SalePrice” since it contains some relatively large values that would have a negative impact on the accuracy of the models. And the Residuals vs Fitted plots (Figure 2.6 and Figure 2.7) before and after the log transformation confirmed our assumption that the transformation made the model more linear.

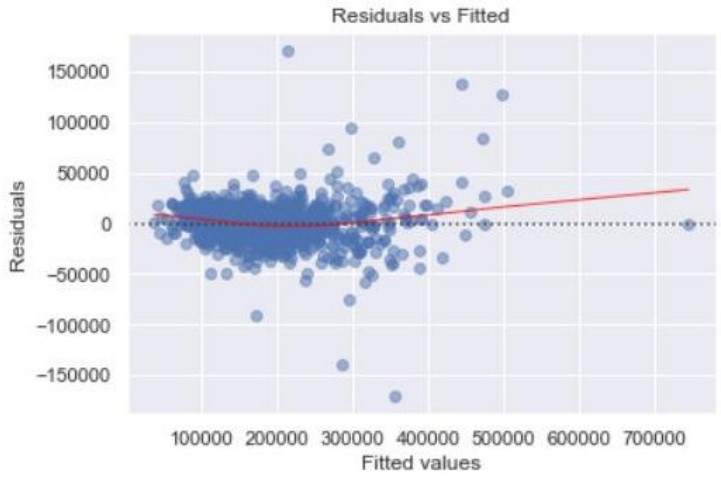

Figure 2.6 Residual vs Fitted Before Log Transformation

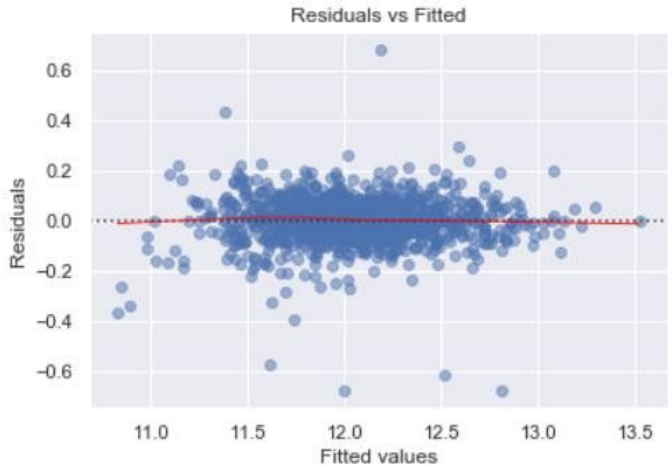

Figure 2.7 Residual vs Fitted After Log Transformation

- Plots for Correlation

We created some plots to study the data. We first looked at the correlation between the target variable and the rest of the variables. Figure 2.8 shows the top 30 features in terms of absolute correlation with SalePrice of the house. The dark color indicates the variable is negatively correlated with SalePrice.

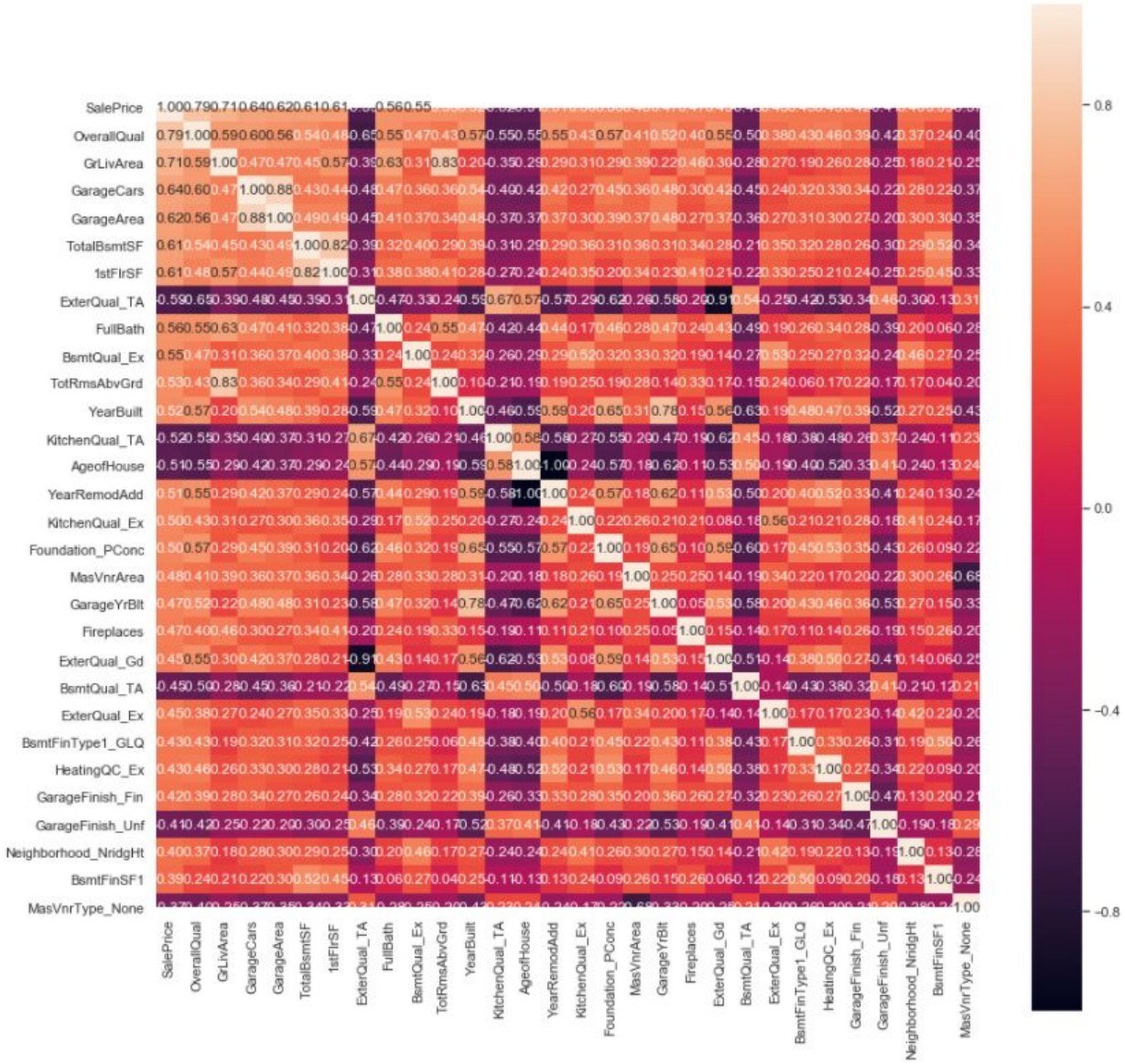

Figure 2.8 - Top 30 “Most” Correlated Variables

We also took a look at the relationship between the “SalePrice” and the individual variables in detail. We selected several interesting charts to present in this report. For example, we made a scatter plot (Figure 2.9) regarding the “SalePrice” and “AgeofHouse”, the feature we created in the previous step. As we can see, the newer the house, the more likely for the house to be sold

at a higher price. Thus, we are confident that the houses with prices over $500,000 were either built or renovated within the most recent 15 years.

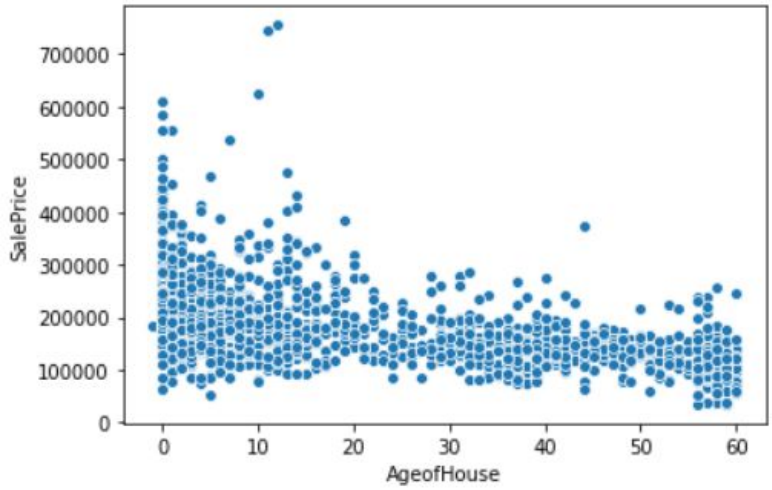

Figure 2.9 - AgeofHouse vs. SalePrice

Another example is the plot for “OverallQual” and “SalePrice”. “OverallQual” is the variable that is most positively related to the “SalePrice”, and the plot (Figure 2.10) clearly shows that the higher ratings will lead to higher prices.

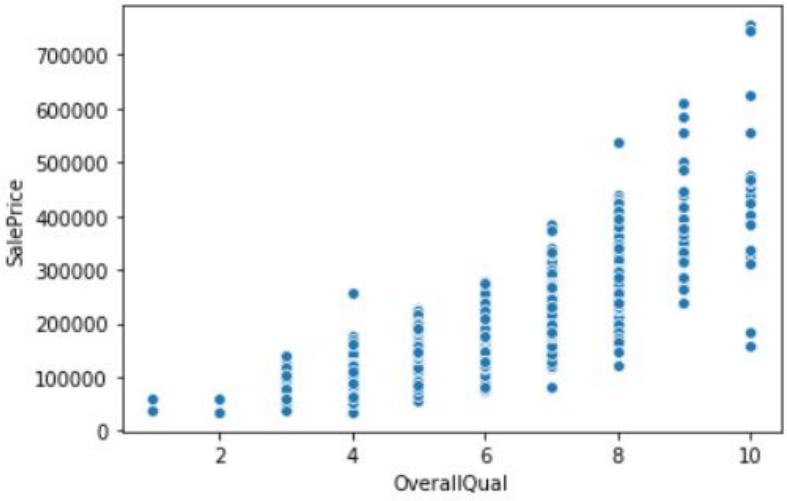

Figure 2.10 - OverallQual vs. SalePrice

Our final training dataset has 1460 observations and 289 variables; and the holdout dataset has 1459 observations and 289 variables.

Modeling

- Individual Models Performance (base vs. tuned model)

Nine regression models were picked in this case: OLS, XGBRegressor, SGDRegressor, MLPRegressor, DecisionTreeRegressor, RandomForestRegressor, SVR, CatBoostRegressor, and LightGBM.

We used the OLS as our base model, and we generated the regression results of the model in the Figure 3.1 :

OLS Regression Results

Dep. Variable:	SalePrice	R-squared:	0.943
Model:	OLS	Adj. R-squared:	0.928
Method:	Least Squares	F-statistic:	65.64
Date:	Sun, 25 Oct 2020	Prob (F-statistic):	0.00
Time:	18:15:09	Log-Likelihood:	1111.4
No. Observations:	1168	AIC:	-1753.
Df Residuals:	933	BIC:	-563.1
Df Model:	234		
Covariance Type:	nonrobust		

Figure 3.1 - OLS Regression Results

According to the OLS report, we learned that the R-squared is 0.943 which means 94.3% of "SalePrice" can be explained by the independent variables in the model. The kaggle score for this model is 0.15569.

In the following table (Figure 3.2), we list kaggle scores for all the models with both untuned and tuned parameters. CatBoostRegressor with tuned parameters had the best performance, 0.12392, among all the other models. This score put us at the number 863 out of 4525 teams (equivalently to the top 19%) by the time we wrote this report.

Kaggle Scores			
Models	**Base**	**Tuned**	**Rank**
XGBRegressor	0.13137	0.12434	***2***
SGDRegressor	0.80454	0.69097	***8***
MLPRegressor	0.41204	0.28506	***7***
DecisionTreeRegressor	N/A	0.19394	***6***
RandomForestRegressor	0.15354	0.15098	***4***
SVR	N/A	0.15540	***5***
CatBoostRegressor	0.12638	**0.12392**	***1***
LightGBM	N/A	0.14440	***3***

Figure 3.2 - Table for Kaggle Scores

- Stacking Models and Average Ensembles

We also created several stacking models and average ensemble models. These types of models normally can help reduce the error by having multiple models work together. However, none of these models can outperform CatboostRegressor. The output is listed in the Figure 3.3:

STACKED MODEL	KAGGLE SCORE
Decision Tree Regressor, Random Forest Regressor, SVM Regressor	.14097
XGB Regressor, Random Forest Regressor, Decision Tree Regressor	.12543

AVERAGE ENSEMBLE	KAGGLE SCORE
Average (Decision Tree, MLP, Random Forest, XGB)	0.14565
Weighted Average (0.4*XGB+0.3*Random Forest+ 0.2*Decision Tree+0.1*MLP)	0.13530

*For our stacked models we used the best parameters for each

Figure 3.3 - Kaggle Scores for Stacking And Average Ensemble Models

Conclusion

Overall, our main challenge with the "House Prices Advanced Regression Techniques" problem was that there was an ample amount of missing data. We tested the data with multiple solutions for missing value, but it was still hard to find a way to improve the accuracy of the models dramatically. Also, we believed that if we could have a bigger training dataset, it may also help improve the models.

After submitting various models, we noticed that our tuned CatBoost regression model was the best predictor of the house prices. We also ran four stacked and average ensemble models composed of three or more different tuned regression models, but none scored better than the CatBoost regressor.

We found this project interesting because we are using many different explanatory variables to predict a single price. This was also interesting because we can use the same methods on datasets from other cities or states to predict house pricing and compare the average prices of different locations. We could also use prediction models like this to help with future house market sales prices.