# Predicting House Prices with Machine Learning

**Diana Dai, Wenqing Lan**
University of Washington
dy2018@uw.edu, lanw3@uw.edu

## Abstract

Our goal is to predict house prices using a data set including variables that cover almost every aspect of residential homes in Ames, Iowa. We explore the benefits of different data pre-processing techniques and the strength of different models. We compare the basic data pre-processing techniques to the advanced ones, including different imputation method and data engineering. The processed data is passed into the Random Forest Regressor, Gradient Boosting Regressor, and CatBoost Regressor with fine-tuned hyper-parameters. Finally, we compared and analyzed the performance of three models.

## 1 Introduction

Housing is a big investment. Therefore, it is important to have an accurate prediction on the house price to help people get a sense of the price ranges to make plans and decisions accordingly. A good prediction model can be beneficial for homeowners, potential buyers, investors, and many other people who are interested in real estate market.

We used the data provided in the "House Prices - Advanced Regression Techniques" competition to build the model. The dataset provides 79 explanatory variables about the house, such as the overall condition of the property, style of dwelling, number of bedrooms, etc. The goal is to predict the final sale price of each house in the test dataset.

## 2 Exploratory Data Analysis

Our data can be divided into two types: numerical and categorical. We analyze each type to understand the data set in more depth.

### 2.1 Numerical Variables

Excluding the target 'SalePrice', there are 36 numerical features. We incorporated a correlation matrix that display the correlation coefficients for different variables. It is a useful tool to identify important features in the given data set. We then zoom in on 10 variables that have stronger correlation than others with the coefficient displayed .

From this matrix, we see that some variable are strongly correlated to the 'SalePrice', such as 'OverallQual' and 'GrLivArea'. These two are followed by variables about the garage, 'GarageCars' and 'GarageArea'. The next two variables are 'TotalBsmtSF' and '1stFlrSF', which deal with the square feet of the basement and the first floor. 'FullBath' counts the number of bathrooms in the house, while 'TotRmsAbvGrd' consider the number of rooms above grade. 'YearBuilt' is in the list as well. Intuitively, the newer houses are more expensive than older ones.

34th Conference on Neural Information Processing Systems (NeurIPS 2020), Vancouver, Canada.

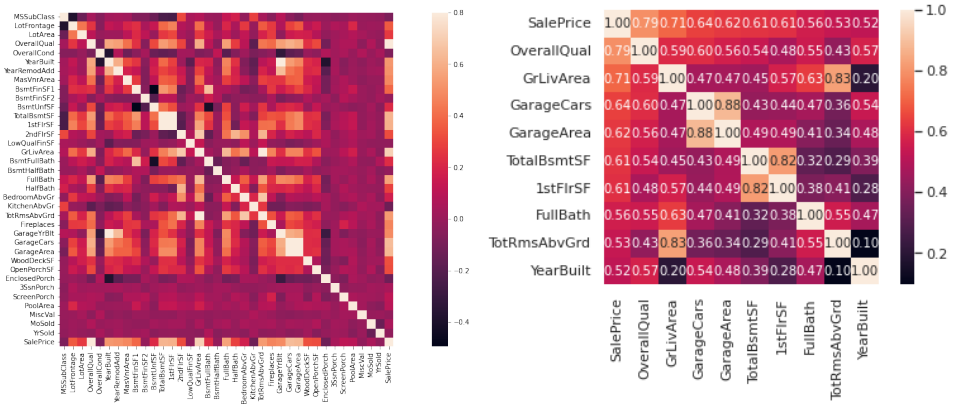


Figure 1: Heatmap of numerical data correlations

### 2.2 Categorical Variables

There are 43 categorical variables in the training dataset. Categorical variables usually need pre-processing before being fitted into most machine learning models, which cannot process NaN's or NA's. Specifically, some categorical data can have large percentage of NAs. These NAs usually indicate that the house doesn't have the feature instead of the value being missing. For example, from our data set, we can see some features, like 'PoolQC' (Pool Quality) and 'Alley' (Type of alley access to property), have more than $90\%$ data marked as NA.

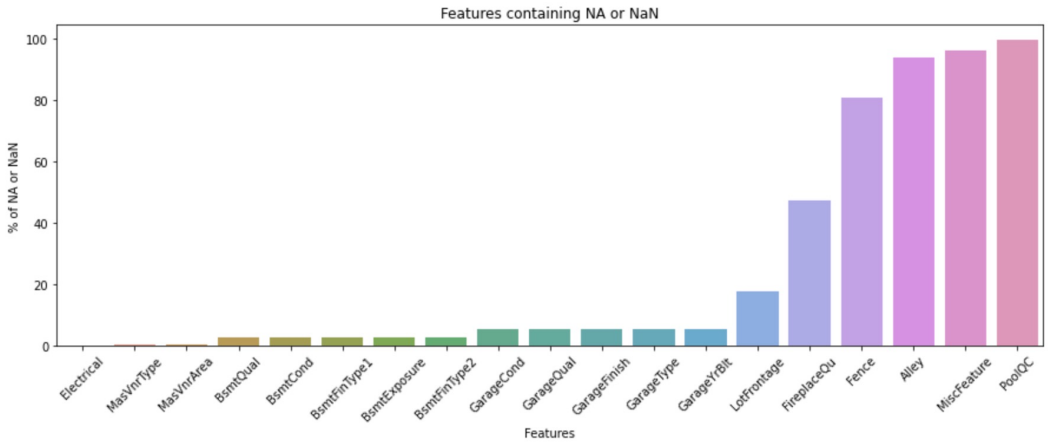


Figure 2: Percentage of data marked NA

## 3 Data pre-processing

### 3.1 Missing Values

19 attributes in the training dataset have missing values. Most machine learning libraries, such as the scikit-learn we use, will give an error if we try to build a model based on data with missing values. We use imputation to fill the missing values. We compared the accuracy by using (1) simple imputation and (2) advanced imputation together with data engineering techniques. We also record the accuracy with data simply dropping the columns with lots of missing values.

### 3.2 Basic Data Processing

#### 3.2.1 Simple Imputation

For numerical variables, simple imputation method directly fills missing values with the mean, the median, the most frequent value, or a constant. For categorical variables, we have two ways: (1) assign each unique value in that categorical variable a different integer, which is called ordinal encoding, or (2) create new columns specifying the presence or absence of each possible value in the original dataset, which is called one-hot encoding. Since some categorical variables may have many values to take, we only focus the categorical variable with less than 10 unique possible values.

### 3.3 Advanced Data Processing

#### 3.3.1 K-Neighbor Imputation

Before using advanced imputation, we choose to change the naming of NA's to indicate 'not available' instead of 'missing'. For example, for 'GarageCars', we changed NA's to 0 meaning there is 0 cars. For features like 'PoolQC' and 'Alley', we change NA's into 'None' string. Next, we use K-Neighbor Regressor to predict the missing values.

#### 3.3.2 Data engineering

To further improve the models' performance, we create new features based on existing ones. An example is: $data['TotalBathrooms'] = (data['FullBath'] + (0.5 * data['HalfBath']) + data['BsmtFullBath'] + (0.5 * data['BsmtHalfBath']))$. This step is applied to the data after K-Neighbors Imputation.

## 4 Training

### 4.1 Random Forest Regressor

The first model we use is random forest. Random forest is an ensemble learning method that operates by averaging the predictions made by many decision trees. It starts with bootstrapping, which aims to make the trees less sensitive to the original data. Then, it select features and build decision trees based on those features. By only building around a few features instead of all, it reduces the correlation between the trees. Random forest gives better predictive accuracy compared to predict based on one single decision tree. Also, this method works well even with default parameters, so it is good to use as a starting point.

### 4.2 Gradient Boosting Regressor

We also use another ensemble model called gradient boosting. This method operates by building a model and optimize it in a forward stage-wise fashion. It starts the ensemble with a single naive model, which may produce wildly inaccurate predictions. Those predictions are then used to calculate the loss. A loss function is used later to fit a new model. The new model will tune the parameters so that by adding this model to the ensemble, the loss will decrease. Then the new model will be added to the ensemble, make new predictions, as the process repeats. Those subsequent additions to the ensemble help to address the errors in the first naive model, and optimize it iteratively. Gradient boosting is flexible with multiple tuning options on hyperparameters. It is also good to capture complex, non-linear patterns in the data.

### 4.3 CatBoost Regressor

The third model is CatBoost Regressor, which is based on gradient boosted decision trees. A set of decision trees is created consecutively with reduced loss. It has about 10 hyper-parameters that provide high flexibility of handling different kinds of data. It can handle categorical data without encoding, which is usually needed for the previous two models. Additionally, it provides the GirdSearch method that can search the best from a combination of parameters. This function saves time from manual hyper-parameter tuning.

# 5 Experiments and Results

## 5.1 Evaluation Method

The metrics we use to measure model quality is the mean absolute error (MAE). We chose this as it shows how much our predictions are off on average. It is a linear score, which means that all the individual differences are weighted equally. This is important because we do not want to favor particular houses.

We also apply cross-validation for evaluation, which means we run the modelling process on different subsets of the data to get multiple measures. Together, we use the averaged MAE for each built model on the validation to compare their performances.

## 5.2 Results

Using techniques discussed in 3.2, the best results after hyperparameter tuning are the following.

| Random Forest | Gradient Boosting | CatBoost |
|---|---|---|
| 17243.31742 | 16542.05295 | 15739.88873 |

For Random Forest, the combination of parameters are: numerical_imputer = median, categorical_imputer = constant, categorical_encoder = one hot encoder, n_estimators = 300.

For Gradient Boosting and CatBoost, the combination of parameters are: numerical_imputer = mean, categorical_imputer = constant, categorical_encoder = one hot encoder, n_estimators = 950, learning_rate=0.05.

During the experiments, we find that simple imputation outperforms imputation with extensions on new columns with a lower averaged MAE value by a small amount. We also figured that for numerical variables, using the median has a better performance compared to using mean or mode.

We also found that using ordinal encoding outperform one-hot encoding on categorical variables, though the difference is small. Also, the relative performances for the four numerical imputers (constant, mean, median, most_frequent) do change even with the same manipulation on the categorical imputers.

Next, using techniques discussed in 3.3, the best results after hyperparameter tuning are the following.

| Random Forest | Gradient Boosting | CatBoost |
|---|---|---|
| 16444.17456 | 16396.74122 | 14557.84784 |

All three models have lower MAE scores compared to using simple imputers. The best test MAE is achieved by CatBoost using advanced pre-processed data. From here, we can see that using more advanced data pre-processing techniques can further improve the models' accuracy. Interestingly, the best score come from the data that has no column dropped.

# 6 Conclusion

We pre-process the training data using simple imputers and advanced imputers combined with data engineering techniques. We fit the processed data to three models, Random Forest, Gradient Boosting, and Cat Boost Regressor and optimized the testing accuracy through cross validation and hyper-parameter tuning. CatBoost Regressor achieved the best score using the advancedly processed training data.

# 7 References

1. House Prices - Advanced Regression Techniques

2. Our code notebook for basic data pre-processing techniques.

3. Our code notebook for advanced data pre-processing techniques.

# 8 Appendix

## 8.1 Choose The Best Simple Imputer Combination

Since we have figured simple imputation is good for handling missing values, we incorporate imputation when we experiment with different models. Some experiment results on random forest models are shown below:

Table 1: Random Forest results using simple imputation

| Numerical Imputer | Categorical Imputer | Categorical Encoder | N_estimators | Validation MAE | Test MAE |
|---|---|---|---|---|---|
| constant | / drop | / | 100 | 18029.09058 | 18017.66597 |
| mean | / | / | 100 | 18054.6604 | 18062.89461 |
| median | / | / | 100 | 18009.76786 | 17791.599 |
| most_frequent | / | / | 100 | 18104.75778 | 18104.75778 |
| constant | most_frequent | ordinal encoder | 100 | 17875.63456 | 17473.5325 |
| mean | most_frequent | ordinal encoder | 100 | 17863.55346 | 17373.73558 |
| median | most_frequent | ordinal encoder | 100 | 17837.4233 | 17360.63065 |
| most_frequent | most_frequent | ordinal encoder | 100 | 17835.94789 | 17390.19606 |

The pattern that we observed is that it is always better to add the categorical variables than simply dropping those, since they may contain important features for building the models.

## 8.2 Hyper-parameter Tuning

For random forest models, we tune the n_estimators parameter, which determines the number of trees in the forest. For gradient boosting models, we tune both the n_estimators parameters and the learning_rate parameters. The n_estimators is still about the number of trees in the ensemble, which is equivalent of the number of boosting rounds. The learning rate determines the learning rate when building the models. Some experiment results on the gradient boosting models are shown below:

Table 2: Gradient Boosting results using the best imputation found previously

| Numerical Imputer | Categorical Imputer | Categorical Encoder | N_estimators | Learning Rate | Validation MAE | Test MAE |
|---|---|---|---|---|---|---|
| constant | constant | one hot encoder | 50 | 0.5 | 20643.49956 | 20125.09586 |
| constant | constant | one hot encoder | 50 | 0.1 | 16731.18512 | 17399.02392 |
| constant | constant | one hot encoder | 50 | 0.05 | 21799.05915 | 21740.79021 |
| constant | constant | one hot encoder | 50 | 0.01 | 110372.6835 | 110909.3765 |
| constant | constant | one hot encoder | 50 | 0.005 | 141272.3367 | 141903.7615 |
| constant | constant | one hot encoder | 100 | 0.5 | 20699.50218 | 20142.44578 |
| constant | constant | one hot encoder | 100 | 0.1 | 16380.56888 | 16944.79786 |
| constant | constant | one hot encoder | 100 | 0.05 | 16655.09916 | 17511.70409 |
| constant | constant | one hot encoder | 100 | 0.01 | 68374.11898 | 68512.86576 |
| constant | constant | one hot encoder | 100 | 0.005 | 110518.3479 | 111046.2366 |

We found out that for the random forest models, the best n_estimators is 300. The average validation MAE does go down for about 100-200 for each combination of imputers and encoders once we use 300 trees instead of the default value of 100 trees. The best test MAE we obtained with random forest

models is 17243.31742, using mean imputation on the numerical variable and constant imputation on categorical variables, which later gets encoded with one-hot encoding.

For the gradient boosting models, the default n_estimators and learning_rate yield worse results compared to random forests models. With tuning, the MAE gets reduced a lot. We found that a learning rate of 0.1 or 0.05 works the best among all training data. Learning rates that are too large or too small drastically increase the MAE. Also, when the n_estimators goes above 500, the averaged validation MAE does not change much.