Final Project Report:

Developing a Prediction Model for Sales Prices using the Ames Housing Dataset

Aditya Pendyala

Michigan State University

STT 481: Capstone in Statistics

Dr. Yue Xing

April 12th, 2024

Contents

1. Introduction, Exploratory Data Analysis, and Data Pre-Processing

Introduction

This project involves the use of *Ames Housing Dataset*. This dataset consists of 79 explanatory variables describing (almost) every aspect of residential homes in Ames, Iowa. It was compiled by Dean De Cock for use in data science education. It's an incredible alternative for data scientists looking for a modernized and expanded version of the often-cited Boston Housing dataset.

Using the above-mentioned dataset, this project aims to develop a prediction model that utilizes many of the 79 variables to predict sales prices for each property. The dataset provided comes in two parts: *test.csv* and *train.csv*. The goal is to use the *train.csv* file to develop a prediction model and then use the *test.csv* file to determine the model's accuracy.

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	...	PoolArea	PoolQC	Fence	MiscFeature	MiscVal	M
0	1	60	RL	65.0	8450	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	NaN	NaN	0	
1	2	20	RL	80.0	9600	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	NaN	NaN	0	
2	3	60	RL	68.0	11250	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	NaN	NaN	0	
3	4	70	RL	60.0	9550	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	NaN	NaN	0	
4	5	60	RL	84.0	14260	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	NaN	NaN	0	

5 rows × 81 columns

Figure 1: *A glimpse into train.csv. Displaying the first 5 rows of the dataset. By excluding 'Id' and 'SalePrice' variables, there are 79 variables left to work with.*

Exploratory Data Analysis and Data Preprocessing

Before loading the datasets, essential libraries such as Matplotlib, Pandas, NumPy, Seaborn, and XGBoost, along with modules for splitting data and evaluating models, were imported. Next, the *train.csv* file was imported using pandas as a data-frame, *train_df*. After inspecting the new data-frame using *.info* method, it was found that many variables had very few non-null values. Some of these variables included: *Alley, MasVnrType, FireplaceQu, PoolQC, Fence, and MiscFeature*. Next, the previously mentioned variables, along with the *Id* variable were dropped from the *train_df* and was renamed *train_df_col_drop*. The *Id* variable was dropped due to its lack of relation to *SalePrice*.

Since dropping rows with Na values would lead to the loss of data, each column of *train_df_col_drop* was first sorted in an ascending order, followed by filling the Na values with forward fill (*bfill)* method. This data-frame was renamed as *filled_train_df*. To further understand each variable, a histogram (*Figure 2*) was plotted using only the float and integer type data. Looking at the plot, it was found that some variables like *LotFrontage, YearBuilt, YearRemodAdd, BsmtUnfSF, TotalBsmtSF, GarageYrBlt, and GarageArea had noticeable variability, almost showing bell curves.*

To encode string variables, I used the pandas *get_dummies* method, and renamed the data-frame as *train_encoded*. To further find variables that have a strong relation to *SalePrice* I plotted a correlation plot (*Figure 3*) with correlation values greater than 0.60. The variables shown in this plot match the variables in the histogram. These variables will likely play a strong role in the prediction model.

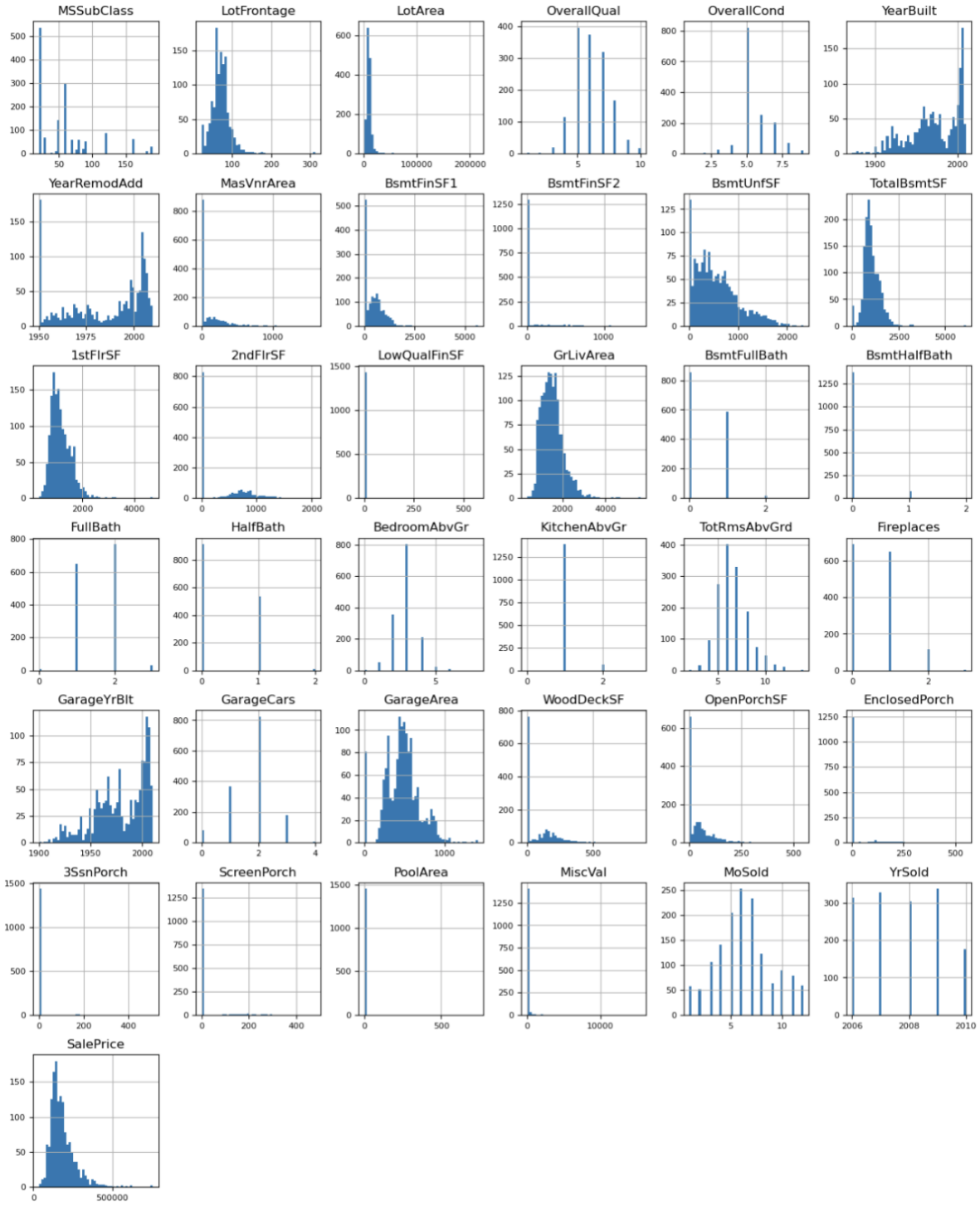

Figure 2: *Understanding the nature of variables in the data through histograms. Some variables show bell-curve like trends, and many show large variability.*

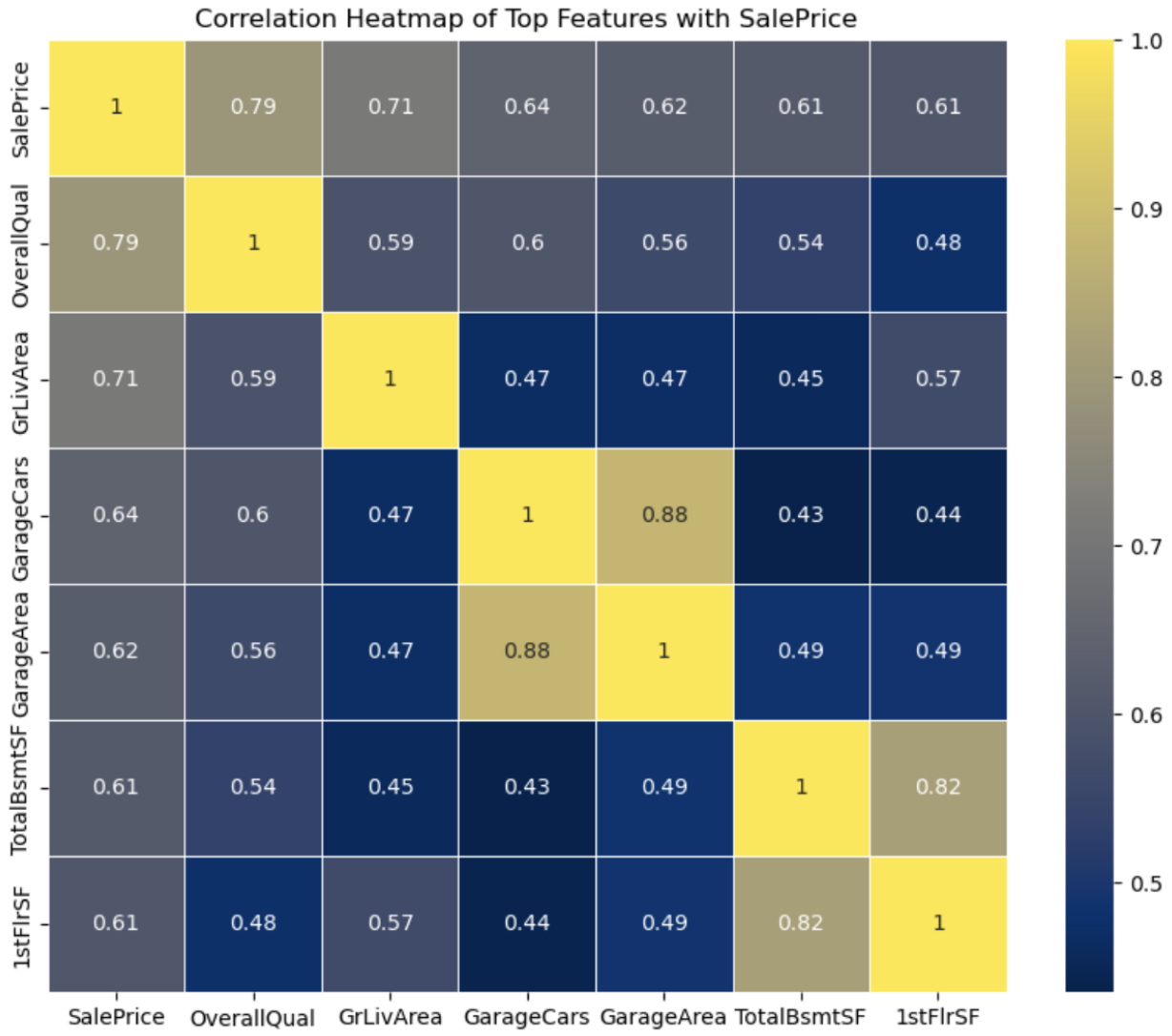

Figure 3: *Correlation plot showing variables with values above 0.60.*

2. Algorithm, Cross Validation, and Hyper-parametric Tuning

Algorithm and Cross Validation

Firstly, the *train_encoded* data-frame is split into features (X) and *SalePrice* (y). Next, the data is split into training and testing data (*X_train, X_test, y_train, y_test)*, with the size ration 70:30. Keeping the testing data aside, the training data is used to

train a XG Boost based model. Since the goal of the project is to create a prediction model that prioritizes squared error reduction, the XG Boost regressor's objective is set to *"reg:squarederror"*. This model is a preliminary model, with no additional parameters. After testing this preliminary model on testing data, the R-squared score was 0.902.

Hyper-parametric Tuning

Few of the most important parameters for a XG Boost regressor are:

1. *max_depth: Depth of each tree. Higher the depth, stronger the learning and the ability for the tree to overfit.*
2. *learning_rate: Higher the value, faster the algorithm will converge to the local minima, but too high might overshoot the minima, and too low might never reach the minima.*
3. *n_estimators: Number of boosting rounds.*
4. *subsample: Sample of training data to be used in each boosting round. It is important to control overfitting.*
5. *colsample_bytree: Number of columns to use when growing a tree. This is also very important to control overfitting.*

One of the ways to find the best model using those parameters is to use Grid Search. So, using Grid Search, the following values for the parameters were used to find the best performing model.

```
from sklearn.model_selection import GridSearchCV

parameters = {
    'max_depth': [3, 6, 10],
    'learning_rate': [0.01, 0.05, 0.1],
    'n_estimators': [100, 500, 1000],
    'subsample': [0.5, 0.7, 1.0],
    'colsample_bytree' : [0.3, 0.5, 0.7]
}

xgb_reg = xgb.XGBRegressor(objective='reg:squarederror', random_stat
grid_search = GridSearchCV(estimator=xgb_reg, param_grid=parameters,
grid_search.fit(X_train, y_train)

print("Best parameters:", grid_search.best_params_)
print("Best score:", grid_search.best_score_)

```

```
Fitting 3 folds for each of 243 candidates, totalling 729 fits
Best parameters: {'colsample_bytree': 0.3, 'learning_rate': 0.05, 'max_
depth': 3, 'n_estimators': 1000, 'subsample': 0.5}
Best score: 0.8820152536433256
```

Figure 4: *Using GridSearch CV to find the best combination of parameters.*

The output shows that 729 fits were performed out of which the best model had the parameters:

1. *max_depth: 3*
2. *learning_rate: 0.05*
3. *n_estimators: 1000*
4. *subsample: 0.5*

5. *colsample_bytree: 0.3*

This model had the score 0.88, and when used for testing, the R-squared value was 0.923, which is an improvement over the preliminary model.

We can see the performance of both the preliminary XG Boost model and the Hyper-parametrized model in the graph below.

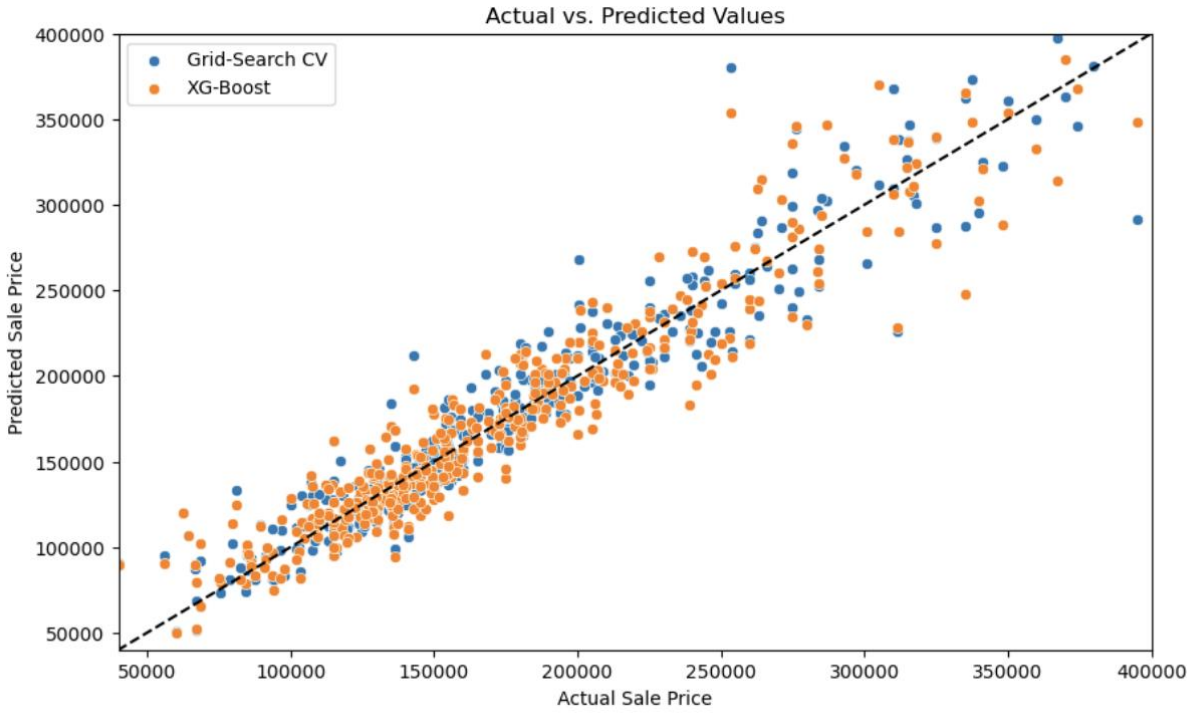

Figure 5: *Plotting both the preliminary and hyper-parametrized XG-Boost models. The closer the datapoints to the dashed line, more the accuracy. We can see that the blue dots are fairly closer to the dashed line than the orange dots.*

3. Methodology and technical challenges

Methodology

XG Boost, short for Extreme Gradient Boosting, is a machine learning algorithm that can handle large amounts of data efficiently. Since the dataset consisted of over 70 variables, XG Boost proved to be a suitable regressor. Furthermore, after handling categorical variables with dummy variables, the number of columns in the data-frame increased to almost 200, making XG Boost the preferred regressor. Moreover, XG Boost includes L1 and L2 regularization (Lasso and Ridge Regression) which prevents overfitting and improved model generalization. This is crucial in complex datasets like those involving real estate where the model needs to generalize well from training data to unseen data.

Technical challenges

The dataset has over 70 variables, making it difficult to look at each variable separately. This makes handling Na values difficult. By using a custom function and *ffill* method, there's not a lot of proper missing data value handling. Some columns had too many missing values, making them non-viable for regression, but could have been proven useful. Furthermore, making visualizations to understand the relationship of every variable with *SalePrice* is difficult due to the large number of variables.

Another major component of technical difficulties is the hyper-parametrization of the regression model. By using multiple values for each parameter, the algorithm had to run 729 different fits, making the process computationally challenging. I overcame this problem by using the argument *" device = 'cuda' "*, which is used to specify that the GPU should be used for computation and not the CPU. This can significantly accelerate the training of models.

4. Model Interpretation

Interpreting an XG Boost model involves understanding the influence of features on the predictions and the decision-making process within the model. XGBoost, while highly effective, can sometimes be considered a "black box" because of its complexity. But there are some ways to make it interpretable. One of them is to find the variables that have the highest F-scores. For the hyper-parametrized model, the top 10 features of most importance are:

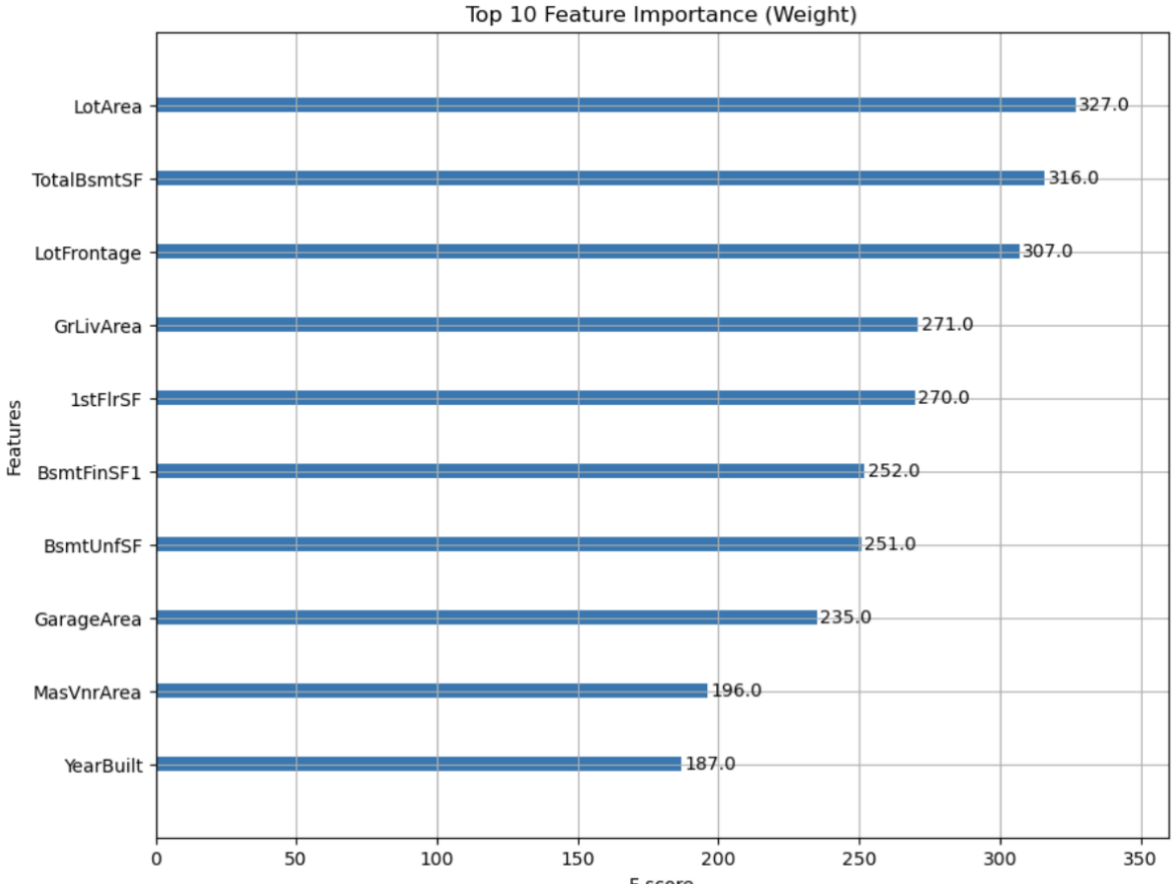

Figure 6: *Plotting the top 10 features with the highest importance to the XG Boost model.*

We can see that many of these features were seen in the initial plots (figure 2).

5. Limitations and potential improvements

As mentioned earlier, hyper-parametrization is a key component to developing an effective and accurate model. Due to computational restrictions, the hyper-parametrization carried out was not the best possible. There's still room for improvement in that space and addition of more parameters could lead to better results. Another space for improvement could be the use of k-fold cross validation, which can lead to better results and lesser overfitting.