Home Price Prediction

Eric Rubtsov, Hannah Kim, Jessica Lee

DSCI 478 Kaggle Project

February 21, 2024

**

**Table of Contents:**

Introduction …………………………………………………………………………………….. 2

Data Preprocessing …………………………………………………………………………… 2

Exploratory Data Analysis & Feature Engineering ………………………………………… 3

Training,Testing, Hyperparameter Tuning ………………………………………………….. 6

Linear Regression Model …………………………………………………………………….. 7

Random Forest ………………………………………………………………………………... 7

Decision Tree ………………………………………………………………………………… 10

Bagging ……………………………………………………………………………………….. 12

Gradient Boosting ……………………………………………………………………………. 13

Lasso & Ridge Regression ……………………………………………………………...….. 15

Support Vector Regression (SVR) …………………………………………………………. 16

Conclusion ..………………………………………………………………………………….. 18

Works Cited ………………………………………………………………………………….. 20

Appendix ……………………………………………………………………………………… 22

***Introduction***

Markets across all industries are riddled with dynamic and complex relationships. Real estate markets on various scales are a prime example of these deeply layered relationships, with their associated effects impacting everyone from buyers and sellers to renters. Modeling these relationships and subsequently predicting outcomes with data science tools have cemented their roles in finance and real estate solutions due to their flexibility, scalability, and interpretability.

The objective of this project is to use features of homes to predict their sale prices. As is standard data science practice in every industry, the data must first be cleaned to be useful and explore its statistical properties. Following this preprocessing and exploratory step involves transforming the data for the objective and applying various statistical learning methods to predict the outcome of unseen data. Tuning the hyperparameters of these models is a critical step in ensuring their predictive potential is maximized, while also balancing computational efficiency. Lastly is the analysis of 8 statistical models (including machine learning models as well) to see which model best predicts housing prices. The models used in this project are linear regression model, random forest, decision tree, bagging, gradient boosting, lasso regression, ridge regression, and support vector regression (SVR).

***Data Preprocessing***

Data collection processes are often, if not always, incomplete and inconsistent. Before exploring the relationships between the features and the outcome, missing values should be handled. A simple method for handling missing values is simply dropping their associated data point. However, this may cause a loss of valuable information, so we decided to drop the features that have over 80% of NA values. For the rest of the data, we used the imputation method, which fills in missing values depending on specified criteria.

The features that determine housing prices are a combination of categorical and continuous values. The imputation technique that requires the least amount of domain knowledge is to simply use the mean of a feature to fill the missing continuous values and the mode of a feature to fill the missing categorical values. The project accomplishes this with the use of a class that automatically determines whether a feature is continuous or categorical, and then handles the imputation accordingly. Using the categorical features also involves encoding the words or abbreviations that represent categories into numbers for a program to be able to make use of them. The class handles this with minimal domain knowledge by using a label encoder that simply attributes a number to a label. These numbers are not ordinal, which is an important consideration when fitting a linear model. The resulting full data set is then ready to be explored.

***Exploratory Data Analysis & Feature Engineering***

Illustrating the distribution of the response variable was the first course of action. Unsurprisingly, the response variable is skewed, meaning that the mean is higher than the median and mode of house sale prices due to the fewer very expensive houses. Outliers in data have the potential to notably impact statistical models and must be carefully evaluated on their contribution to their relationship with the response variable. This evaluation involves context consideration, so the project considered that that larger living area is correlated with a greater sale price. These outliers were determined visually from the correlation plot. Figure 1 illustrates the distribution of sale price and the removal of outliers based on the living size and sale price relationship.

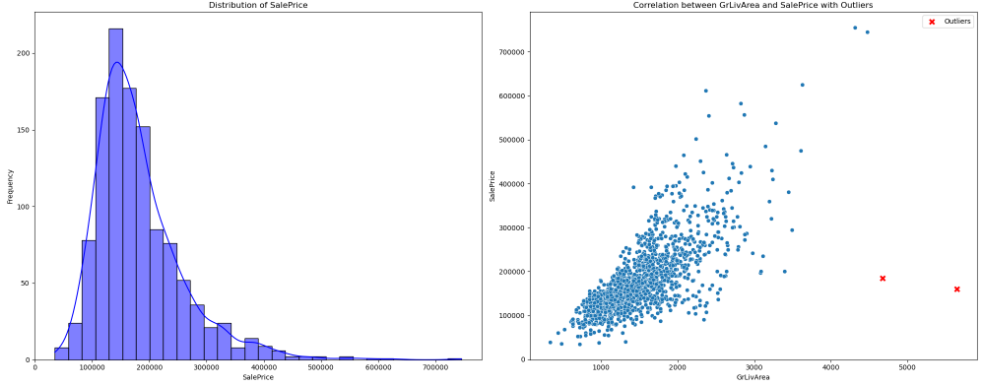
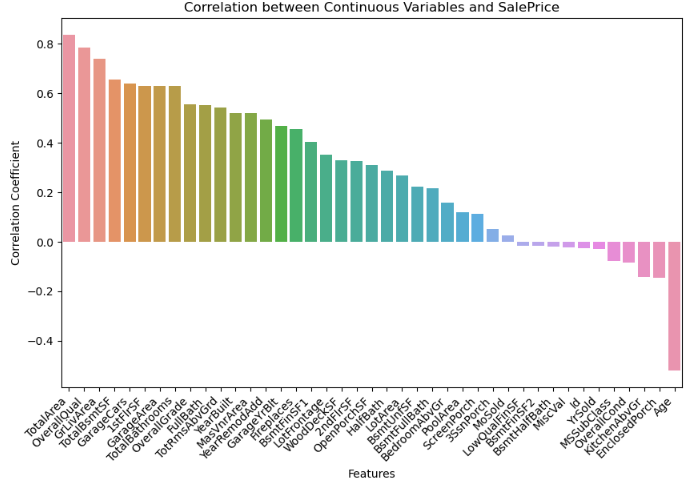


Figure 1: Sale price distribution and removal of outliers

Feature engineering is a cornerstone of extracting useful information from data. Creating a new feature by combining relevant information from separate features is an incredibly useful technique in uncovering initially unseen relationships. The features that were created with already existing features in this project include the total area: adding up all the square footage, house age: the difference between the year that the house was sold and the year that it was built, the number of bathrooms: adding up all the bathrooms with consideration of half bathrooms, the overall grade: compacts the overall condition multiplied by the overall quality, and a binary indicator to represent whether the home was remodeled or not. The feature importance to the sale price can then be visualized with Pearson's correlation coefficient for the continuous features and the chi-squared test for the categorical features, as shown in the visuals below.



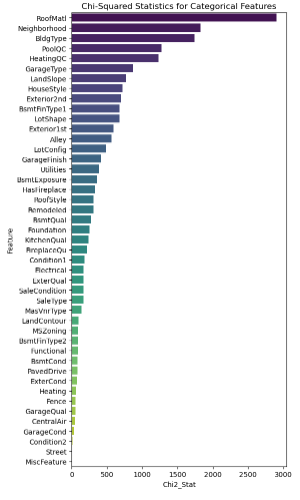


Figure 2: Correlation plot (continuous variables) and Chi-Squared plot (categorical variables) for feature importance as it relates to sales price

As can be seen in the visual, added continuous features such as TotalArea, Age, and TotalBathrooms have strong correlations with the sale price of a house. The indicator of whether a home was remodeled or not has a moderate impact on sales price.

Feature importance is revisited in training the Random Forest model for dimension reduction. The results of the Random Forest’s feature selection will then be cross-referenced with the findings from the correlation and chi-squared assessments to ensure the most important features are indeed being accounted for in the model.

***Training, Testing, Hyperparameter Tuning***

The train\_test\_split module from the Scikit learn library was implemented to train the various models on 80% of the training housing data and tested for their accuracy on the remaining 20%. Splitting the data into training and testing subsets allows the models to prevent overfitting by evaluating the performance of the model on unseen data and tuning the hyperparameters such that the machine learning model is not simply recognizing patterns from the data it was trained on. The module also provides the option of randomness control, by setting a reproducibility seed.

Hyperparameter tuning is essential for the learning process of machine learning models. The various learning algorithms configure these hyperparameters before the learning process begins, where they remain fixed and influence the resulting learning parameters that were obtained through the training process. Examples of such hyperparameters for the tree-based models used in this project include tree depth, number of samples per leaf, stopping criteria (impurity metrics), and the number of features to split on. The Support Vector Regression model considers the kernel type for training the model on either linear or non linear relationships, the regularization parameter for balancing a trade off between complexity and overfitting, and the tolerable margin of error for predictions represented by the epsilon parameter. Although beyond the scope of this project, the options for tuning machine learning algorithms also encompass the choice of optimizer algorithms like stochastic gradient descent.

This project utilized Scikit Learn’s GridSearchCV module to define a grid of hyperparameters for each model, evaluates the performance of each model with each combination of these hyperparameter values using k-fold cross-validation, and chooses the model with the highest cross validation score. K-fold cross validation is a holdout method of cross validation where the model is trained on k-1 partitions of the data and tested on the remaining partition. This is repeated until every data partition has been a test set and the performance metrics on all k test sets are averaged. The module allows to specify the number of “folds” or data partitions and allows for the specification of the CV score; r^2 was the CV score used for this project. This method of cross validation was also used for feature selection in the random Forest model.

***Linear Regression Model***

As shown in Figure 1, the sale price of homes is positively skewed. This skewness can be normalized by log-transforming the sale price variable. A linear regression model was fit to this log-transformed sale price value. The sale price was then back-transformed to make predictions on the test set and to plot the Actual versus predicted sales price. The result was a remarkable r^2 value of 0.92.

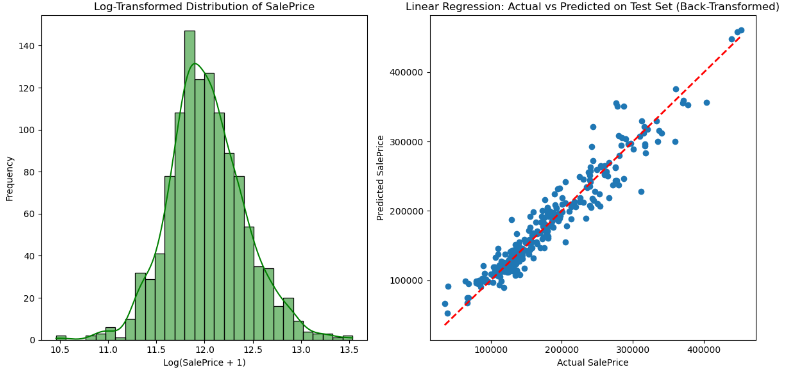


Figure 3: Linear Regression model fit to log-transformed sales price

***Random Forest***

Random Forest is an ensemble method for decision trees that is an extension of the bootstrapping aggregation method. The Random forest algorithm combines random sampling of data with replacement by randomly generating a subset of the features to create an uncorrelated forest of decision trees. The layers of randomness from selecting subsets of features and samples within those features greatly reduce the correlation between trees, resulting in the averaging of the individual trees (necessary for regression) to have reduced variance and not overfit the training data.

Once the subsets of features are randomly selected, training samples from these subsets are randomly selected, as would be done in bootstrap aggregation. A third of each of the training samples is set aside as testing data, also known as the out-of-bag (OOB) sample. This OOB sample is used for cross-validation once the individual trees are averaged. Random Forest advantages include reduced risk of overfitting, flexibility between regression and classification tasks, and feature selection. The major drawback of Random Forest is complexity and computational resources since predictions for several individual trees must be computed (IBM).

This project leveraged the feature selection capability of random forest with Scikit Learn’s SelectFromModel module. Before filtering out features, a visual of all the features and their importance was included [1]. This module filters out unimportant features from a dataset. The importance of each feature is determined by a threshold value, which corresponds to that feature's coefficient in a predictive model. This means that a lower threshold means more features will be selected because the criteria for a feature to be considered important are lower. This project determined the threshold by looping through threshold values, passing a RandomForestRegressor object to a SelectFromModel object given each threshold, storing these selected features, and using Scikit Learn’s cross-validation module to determine the random forest model with the subset of features that have the highest r^2 value. These selected features were then used later in the project to test other models on a reduced number of features. Figure 4 provides a visual of the cross-validation process for feature selection using Random Forest.

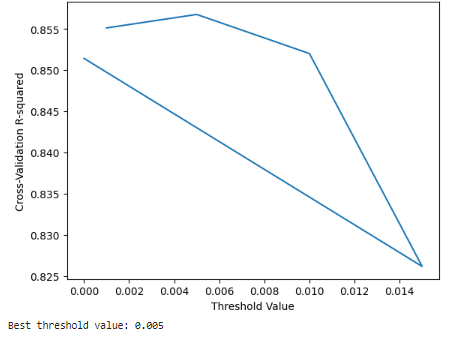


Figure 4: Cross-validation to determine the best threshold value for feature selection using Random Forest

We also used random forest as one of our models to predict housing prices. The hyperparameters that were considered in this model were the maximum depth of the tree, the minimum number of samples split, and the number of estimators (trees). According to our findings, a minimum sample split of 2 and 30 estimators in the ensemble is the most optimal for this dataset. The r^2 value obtained during the hyperparameter tuning process with grid search was 0.8694 or 86.94%. This tells us that the model explains around 86.94% of the variance in the target variable and captures a good amount of the underlying patterns in the data. We also obtained an RMSE of 28567.0534 which tells us that on average, our random forest model’s predictions on the validation set have about a $28,567.05 difference from the actual values of the houses. Lastly, our r^2 on the validation set came out to be 0.8936 or 89.36%. When using our random forest model on new and unseen data that was not used during hyperparameter tuning, we got a higher r^2 value than what was originally obtained.

Looking at the learning curve for our random forest model (Figure 5), the model improves rapidly with an increase of training samples, however, it slows down as the number of training samples increases.

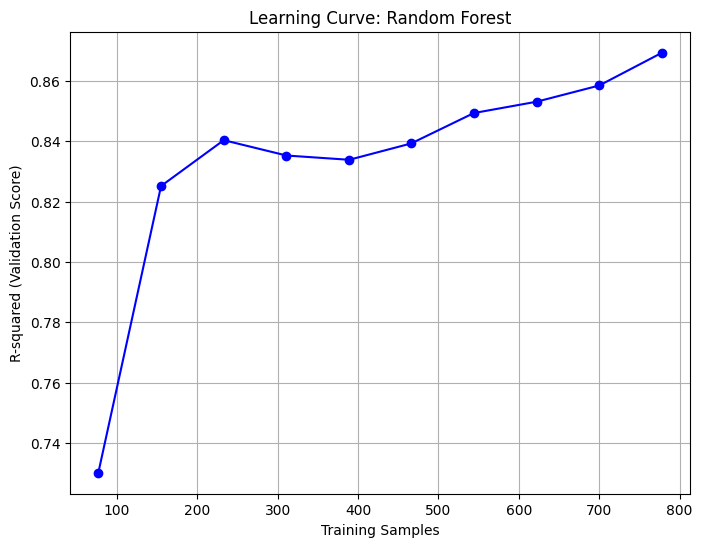


Figure 5: Training samples versus predictive accuracy learning curve of random forest model.

***Decision Tree***

Decision trees are a learning algorithm used for both regression and classification. The structure of a decision tree is reflected by its name which resembles a hierarchical tree shape. The structure of a decision tree represents the variables’ relative importance in the model. It goes from the most important variables on the top to the least important variables at the bottom. There are two types of decision trees; classification trees and regression trees. In our tests, we used regression trees using the DecisionTreeRegressor from the sklearn.tree library.

This project evaluates the base model decision tree on its variance-bias tradeoff. This base model uses the reduced number of features determined from the random forest model. Determining the overfitting of this “weak learner” to be used in the bootstrap aggregation is done by plotting the learning curve of the validation set as the number of training samples increases. A plateauing or decrease in predictive accuracy as the number of samples increases indicates overfitting. Figure 6 illustrates the learning curve on the validation set of the decision tree which uses the selected features from the random forest mode.

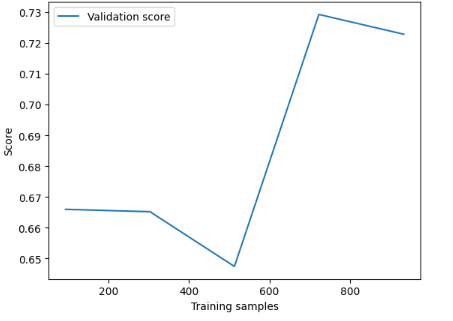


Figure 6: Training samples versus predictive accuracy learning curve of base decision tree model.

The r^2 score of the decision tree model during the hyperparameter tuning process is around 0.999992 while the r^2 score on the validation set is 0.8124, strengthening the suspicions of the presence of overfitting and high variance. This suggests that the model is capturing randomness from the training data, leading to an overestimation of prediction performance.

The hyperparameters considered for our decision tree model were the number of these base models to average together, the maximum depth of the tree, which can control the complexity of the relationships to capture, the proportion of training data to be randomly sampled for each base model, and the use of an out of bag cross-validation to finalize the prediction. Upon tuning the hyperparameters with grid search, it becomes clear that the highest-performing model requires the greatest number of base learners, unlimited tree depth, and a high proportion of training data to sample. The predictive accuracy of this model does not surpass Random Forest. We also obtained a RMSE on the validation set of 37,929.7233 which tells us that our decision tree model’s predictions on the validation set have about a $37,929.7233 difference from the actual values of the houses.

***Bagging***

Bagging, otherwise known as bootstrap aggregation, is an ensemble method that randomly samples data from a training set and trains many base models in parallel to then aggregates into a strong learning decision tree. Averaging base model decision trees trained from several random samples reduces the variance. Therefore, bagging methods are typically used when base models exhibit high variance that is prone to overfitting and low bias.

This project tests bagging to predict housing prices using the hyperparameters of the number of bootstrapped samples used to train each base estimator, the maximum depth of a tree, the maximum number of samples, and out-of-bag samples which are not used in the bootstrap. According to our findings, a maximum number of samples of 0.9, 200 number of bootstrapped samples, and using out-of-bag samples in the ensemble is the most optimal for this data set. We obtained a r^2 value of 0.8747 during the hyperparameter tuning process using grid search. The bagging model explains around 87.47% of the variance in the target variable and captures the underlying patterns in our data. We also obtained a RMSE on the validation set of 27,763.54 which implies that on average the bagging model’s predictions have about a $27,763.54 difference from the actual values of the houses. Lastly, we got a r^2 value on the validation set of 0.8995 or 89.95%. We can see that our bagging model performed better on new unseen data than on training data and was able to make more accurate predictions.

Looking at Figure 7, we can see that the learning curve of the bagging model is not a smooth increase. Initially, it rapidly improves but as the training samples increase the model’s overall performance goes up and down in a repetitive pattern. However, in the long run, we can see that as training samples increase, the model does improve. This pattern can be explained by the nature of bootstrapping samples. Due to the random nature of bootstrap sampling, each base model encounters different subsets acquired from the training data. This randomness can be reflected in up-and-down fluctuations in the learning curve graph.

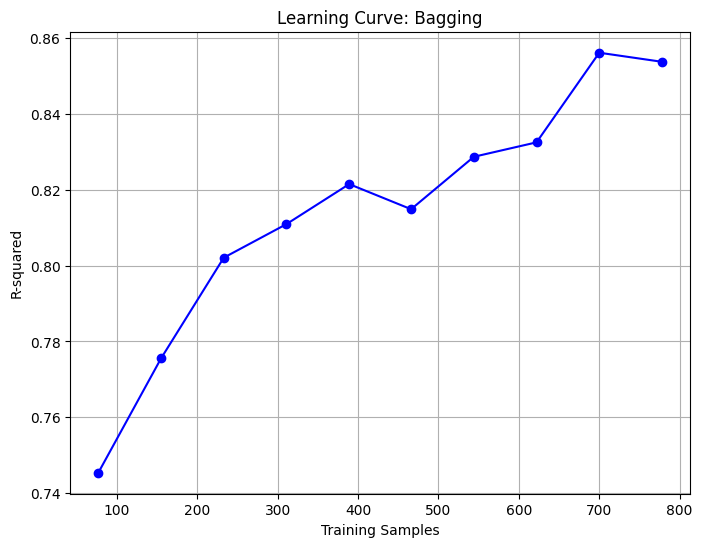


Figure 7: Training samples versus predictive accuracy learning curve of bagging model.

***Gradient Boosting***

Gradient boosting is an ensemble method that aims to minimize training errors by combining weak learners into strong learners. This is done so by selecting a random sample from the data and fitting it with a model which is then trained. With each iteration, each model focuses on correcting the errors (residuals) made by the previous models using gradient descent optimization. These models then work together to learn from other models’ shortcomings in order to improve overall performance. In the end, gradient boosting makes the model a stronger and more reliable prediction model. Looking at Figure 8, the learning curve of our gradient boosting model we can see that this model improves fast with more and more training samples, however, it reaches a plateau as the training samples increase.

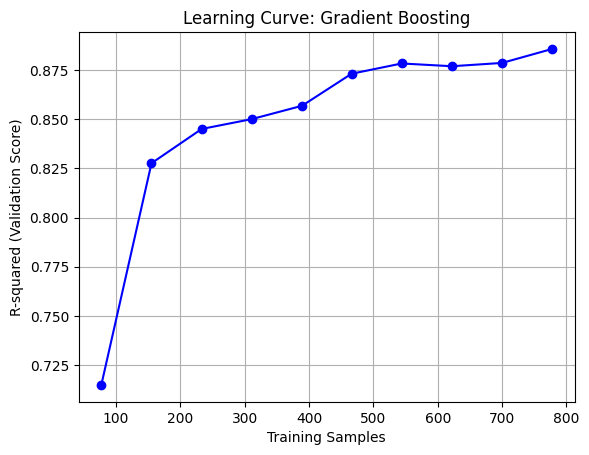


Figure 8: Training samples versus predictive accuracy learning curve of gradient boosting model.

This project tests gradient boosting to predict housing prices and the hyperparameters considered in this model are learning rate, maximum tree depth, and number of estimators (trees). According to our findings, a learning rate of 0.3, a maximum tree depth of 3, and 30 estimators in the ensemble are the most optimal for this dataset. The r^2 value obtained during the hyperparameter tuning process with grid search was 0.8707 or 87.07%. This tells us that the model explains around 87.07% of the variance in the target variable and captures a good portion of the underlying patterns in the data. Our RMSE on the validation set was 26565.0302. This indicates that on average, our gradient boosting model’s predictions on the validation set have about a $26,565.03 difference from the actual values of the houses. Finally, our r^2 on the validation set came out to be 0.908 or 90.8%. When applying our gradient boosting model to new and unseen data that was not used during hyperparameter tuning, we got a higher r^2 value than what was originally obtained during training. This tells us that our model performs well on new data and has the ability to make accurate predictions in real-world settings.

***Ridge & Lasso Regression***

Lasso regression, which is well known for feature selection and regularization, is adept at preventing overfitting and managing high-dimensional data. While making the sum of the absolute values of the weights to be minimal, the lasso also finds the weights and the biases that will minimize the MSE. All elements of the weight will be zero or close to zero. In other words, lasso regression itself selects the features that it thinks are important to include, therefore, we can see that some of the features are removed to create the model (Figure 9).

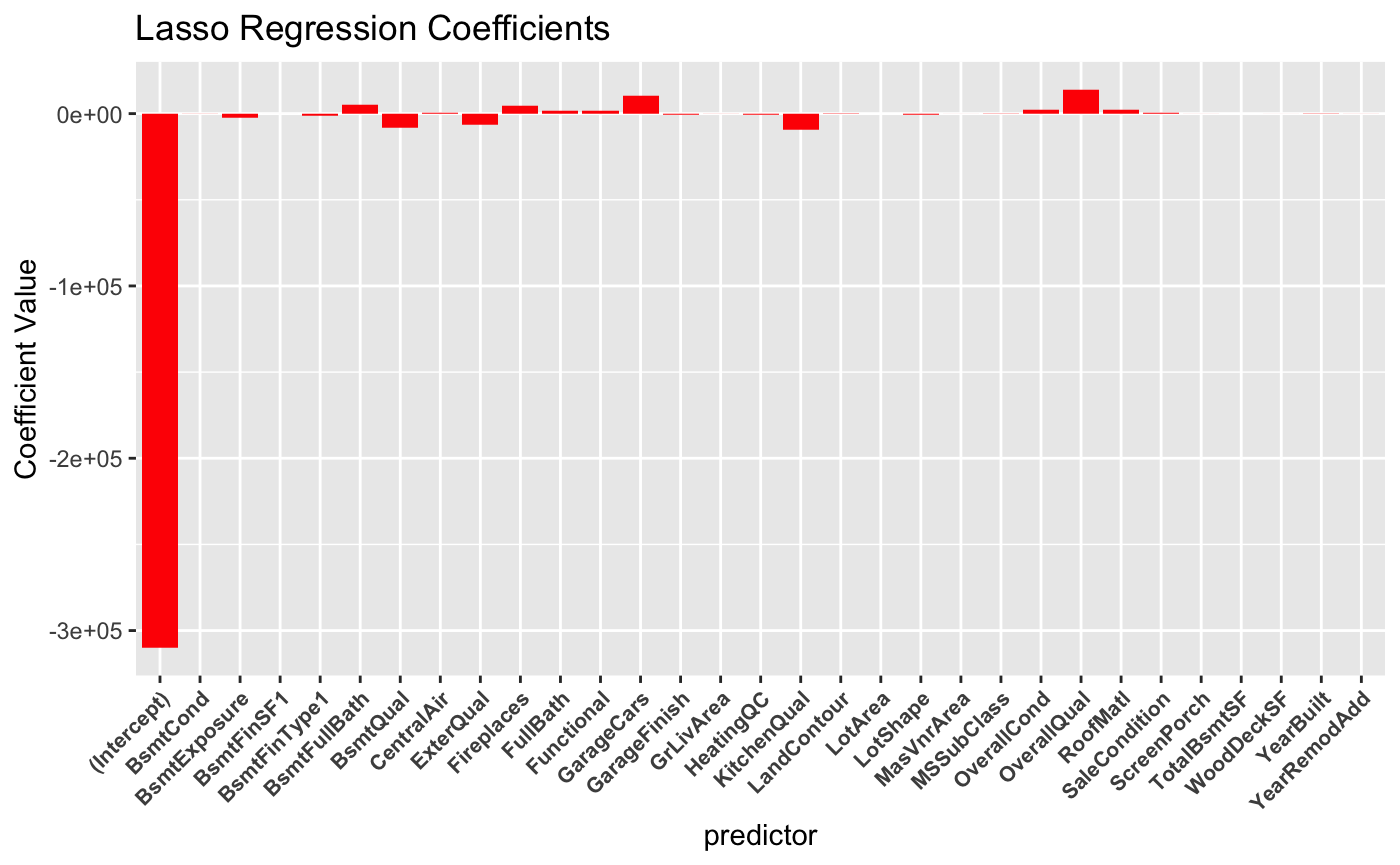


Figure 9: Variables used in Lasso regression

Ridge regression is known for its ability to deal with multicollinearity, overfitting, and reducing coefficients. However, unlike Lasso regression, ridge regression reduces model complexity without the complete removal of some features. So rather than selecting variables, ridge regression will distribute appropriate weights to all variables as we can see in Figure 10.

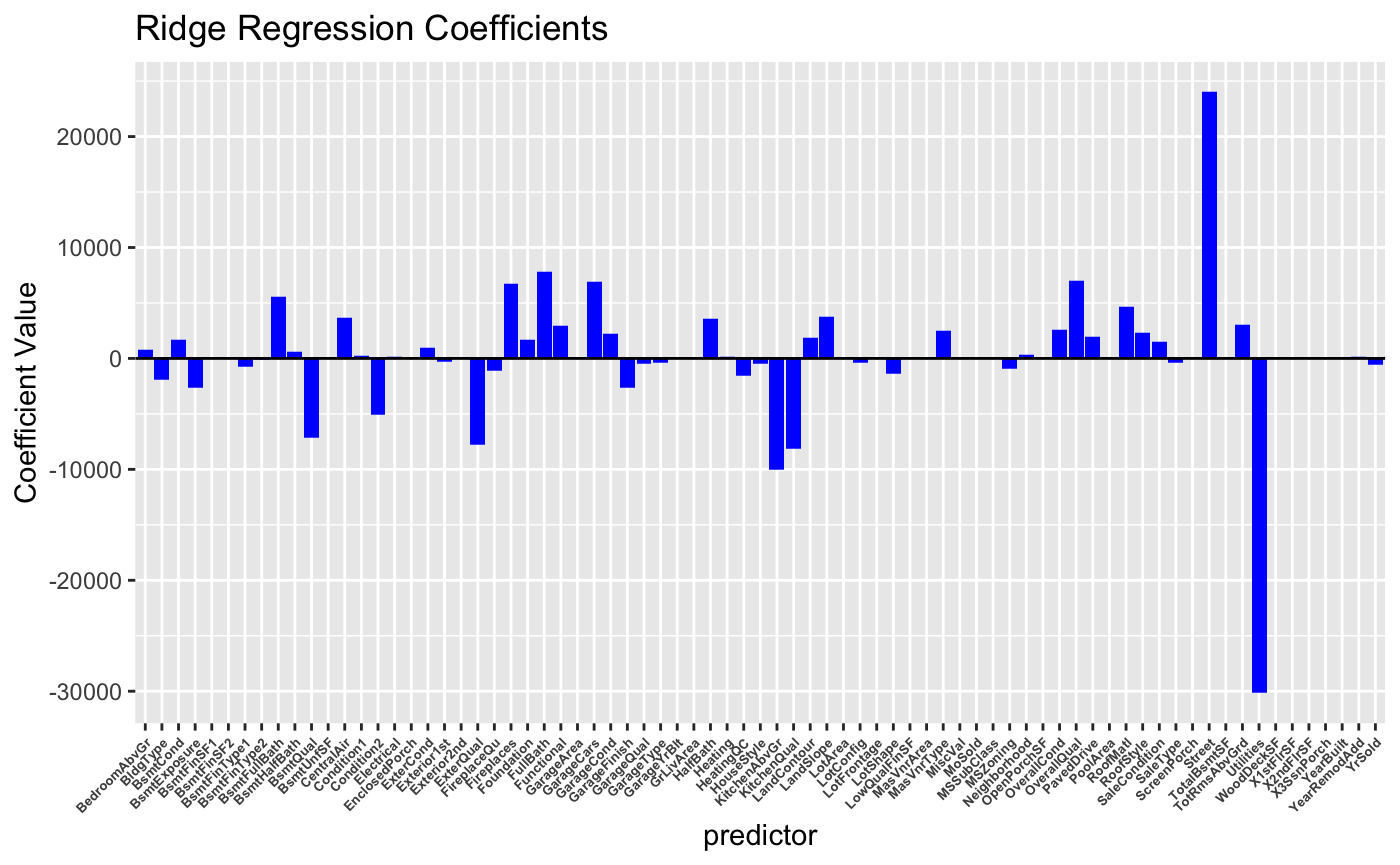


Figure 10: Variables used in Ridge regression

Both Ridge and Lasso find the optimal model by adjusting λ and cross-validation is used in the process. To be specific, they first select the grid of λ to consider and find the cross-validation error for each λ value. Then the λ value that makes the error the smallest is used for prediction.

The best lambda value used for the ridge regression model is 43935.23 and 2186.67 for the lasso regression. With these lambda values, the prediction on the validation set results in a r-squared value of 0.8899 for lasso and 0.8935 for ridge.

***Support Vector Regression (SVR)***

Support Vector Machines are a method that finds a hyperplane in high-dimensional space to separate classes. SVM’s use kernel functions to map the input variables to higher dimensional feature spaces for classification tasks, depending on linearity or non-linearity. The idea behind SVR’s is similar, except the hyperplane aims to fit the data points in a continuous space. The objective is to find a decision boundary on each side of the hyperplane that contains the most data points from the training sample. Looking at Figure 11, we can see that the learning curve of the SVR model initially increases but right away it starts to decrease as the training samples increase. This tells us that there is a possibility of overfitting occurring. The model may be memorizing the noise from the training data and overestimating predictions.

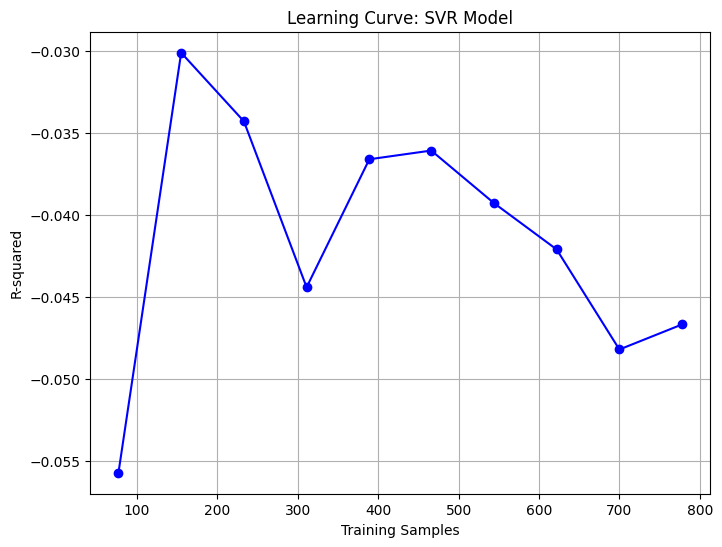


Figure 11: Training samples versus predictive accuracy learning curve of SVR model.

This project tests SVR to predict housing prices. The hyperparameters considered in this model are the kernel, the regularization parameter, which is meant to determine how closely fits the model to the training data at the risk of overfitting in the testing data, and the epsilon value which is the tolerance for prediction error. Scikit Learn’s StandardScaler() and SVR() functionality is passed into the makePipeline feature, which allows for chaining transformations and model fitting on data. SVR is sensitive to the scale of features, so the model resulting from the pipeline is fed into the grid search for the best hyperparameters after standardizing the feature scales.

The best kernel turned out to be linear, which checks out, considering the linear regression model on the log-transformed sale price variable was so high-performing.

***Conclusion***

The predictive accuracy of machine learning models can take on a few metrics. However, we specifically used Root-Mean-Squared-Error (RMSE) for evaluation to meet Kaggle’s scoring criteria. We also calculated the R-square, which describes how much variability in the response variable is associated with the input variables. Below is a table that organizes the performance of our various models by RMSE and R-square:

|  | Best R^2 | RMSE on Validation Set | R^2 on Validation Set |
| --- | --- | --- | --- |
| Gradient Boosting | 0.8824 | 26,565.0302 | 0.908 |
| Bagging | 0.8707 | 29,031.6843 | 0.8995 |
| Log-transformed Linear Regression | 0.8601 | 27,794.9894 | 0.8993 |
| Random Forest | 0.8694 | 28,567.0534 | 0.8936 |
| Lasso | 0.8233 | 30,133.44 | 0.8901 |
| Ridge | 0.8288 | 31,227.57 | 0.8888 |
| Decision tree | 0.999992 | 39,616.6329 | 0.7954 |
| SVR | 0.504 | 56,078.4508 | 0.59 |

From this table, we can see that the gradient boosting model had the highest r^2 on the validation set (90.8%), as well as the lowest RMSE on the validation set (26,565.0302). This tells us that the gradient boosting model performs the best out of all of the models we created when encountering new data and has the best ability to make accurate predictions in real-world settings. The best-performing models in predicting housing prices on the validation set are in sequential order; gradient boosting, bagging, log-transformed linear regression, random forest, lasso regression, ridge regression, decision tree, and lastly SVR.

Gradient boosting and bagging resulted as the best top two models when predicting housing prices. Gradient boosting continues to improve the errors from previous models while bagging trains multiple models on different sections of the data. Both of these models’ characteristics help in reducing the models' tendency to memorize the noise present in the training data, which avoids overfitting and improves overall performance. Additionally, both gradient boosting and bagging are ensemble methods that use decision trees as base models. This makes them capable of capturing non-linear relationships between features which makes them suitable for predicting house prices based on a variety of features like type of alley access, roof material, type of heating, garage quality, etc.

This housing price prediction project demonstrates how we can apply data science concepts to daily life decisions, such as real estate investments, to both individuals and companies by optimizing prediction models to assist in major decision-making processes. We were able to analyze the nature and performance of a few statistical machine-learning models. Specifically, linear regression, random forest, decision tree, bagging, gradient boosting, lasso regression, ridge regression, and SVR are the models that we explored.

Works Cited

“A Complete Guide to Dealing with Missing Values in Python” Analytics Vidhya, 27 Feb.

2023, [https://www.analyticsvidhya.com/blog/2021/10/a-complete-guide-to- dealing-](https://www.analyticsvidhya.com/blog/2021/10/a-complete-guide-to-dealing-)with-missing-values-in-python/

“All You Need to Know about Gradient Boosting Algorithm - Part 1.Regression” Medium,

19 Jan. 2022, <https://towardsdatascience.com/all-you-need-to-know-about-gradient-boosting-algorithm-part-1-regression-2520a34a502>

“All You Need to Know about Gradient Boosting Algorithm - Part 2.Classification"

Medium, 7 Feb. 2022, https://towardsdatascience.com/all-you-need-to-know-about-gradient-boosting-algorithm-part-2-classification-d3ed8f56541e

“Model-Based and Sequential Feature Selection.” *Scikit*,

scikit-learn.org/stable/auto\_examples/feature\_selection/plot\_select\_from\_model\_diabetes.html#:~:text=Selecting%20features%20based%20on%20importance,-Now%20we%20want&text=SelectFromModel%20accepts%20a%20threshold%20parameter,of%20third%20most%20important%20feature. Accessed 20 Feb. 2024.

Sethi, Alakh. “Support Vector Regression Tutorial for Machine Learning.” *Analytics*

*Vidhya*, 9 Feb. 2024, www.analyticsvidhya.com/blog/2020/03/support-vector-regression-tutorial-for-machine-learning/.

“sklearn.ensemble.GradientBoostingRegressor” sckitlearn,

<https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingRegressor.html>. Accessed 15 Feb. 2024

“sklearn.ensemble.RandomForestRegressor” sckitlearn,

<https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html>. Accessed 14 Feb. 2024

“What Is Bagging?” *IBM*,

www.ibm.com/topics/bagging#:~:text=Bagging%2C%20also%20known%20as%20bootstrap,be%20chosen%20more%20than%20once. Accessed 20 Feb. 2024.

“What Is Boosting” IBM, <https://www.ibm.com/topics/boosting>. Accessed 20, Feb. 2024

“What Is Random Forest?” *IBM*, www.ibm.com/topics/random-forest. Accessed 20 Feb. 2024.

Singh, V. (n.d.). *Ridge regression vs lasso regression - shiksha online*. shiksha.

https://www.shiksha.com/online-courses/articles/ridge-regression-vs-lasso-regression/

**Appendix**

*[1] All feature’s importance plot*

