Pawel Linek and Joseph Burns
12/5/19
Final Project Report

Dataset:

House Prices: Advanced Regression Techniques

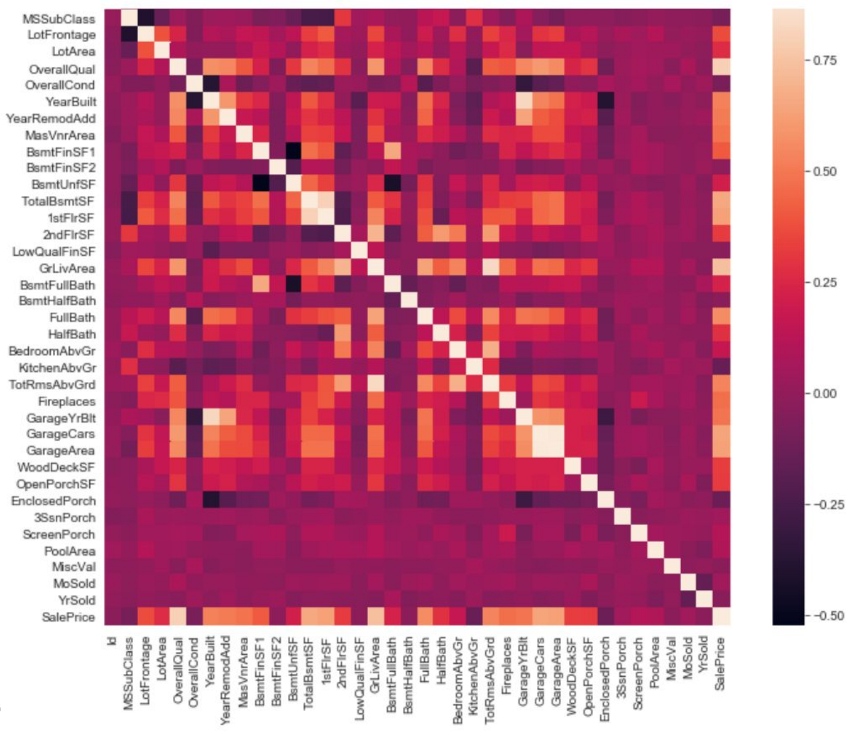
Abstract:

For this project we decided to compare various classifiers on a relatively simple dataset. Our goal was to try to predict house prices based on the data in the Ames Housing dataset. We attempted to solve this problem using 2 classifiers, Decision Trees, and Multiple Regression and Boosting hybrid. Below are the results:

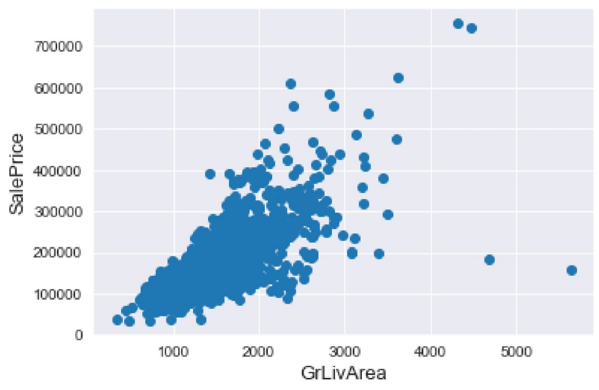
Introduction:

The House Prices dataset is an 80+ feature dataset consisting of some features which have a large impact on the sales price, and some that do not. An example of the top 6 variables for the dataset which have the largest impact on the sales price are the:

- Overall Quality
- Above Ground Living Area
- Garage Cars/ Garage Area
- Total Basement Square Feet
- Total First Floor Square Feet



This heatmap is the basis for the above features. It can be seen as significant because many of the above features are some of the first features our first classifier, the decision tree, looks at when deciding how to branch.

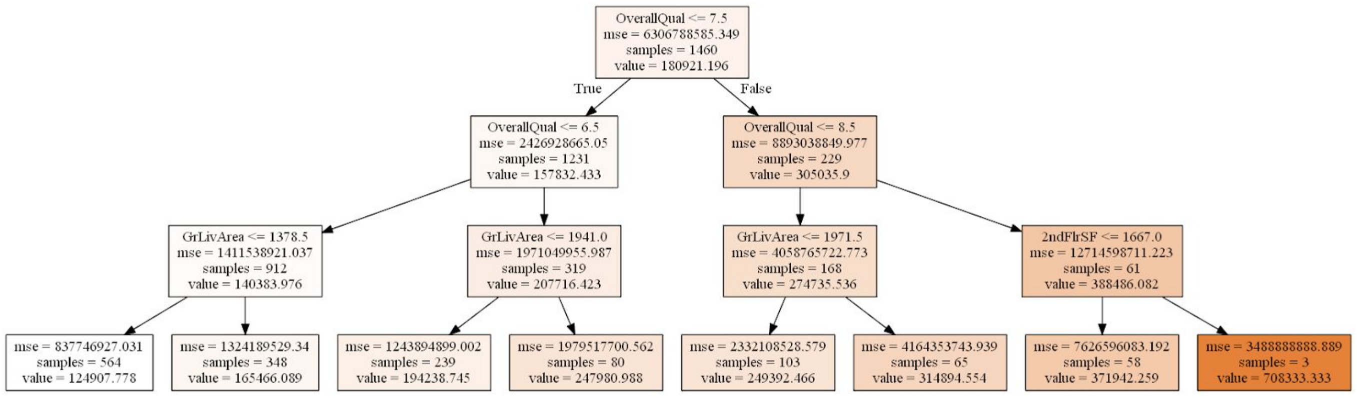


The scatter plot to the left shows how “Above Ground Living Area”, [the second most significant] is correlated to the Sales Price [what we are looking to predict]. As it can be seen the correlation looks relatively positive.

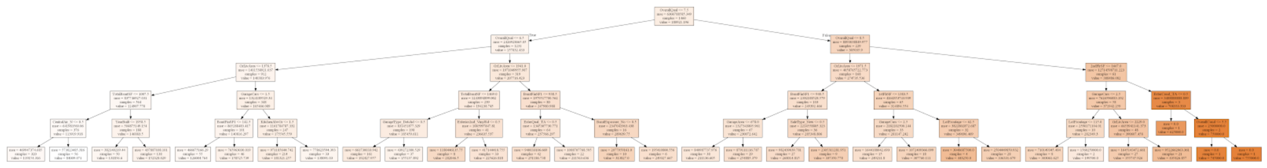
Decision Tree:

The main purpose of the decision tree classifier, was to see the effects of under and overfitting. We tested the decision tree algorithm while limiting the depth to 3, 5, 7, 9 and unrestricted. As can be imagined, the 3 and 5 trees seemed to be a little underfit, while the unrestricted tree overfit. Below are some images of the trees of each associated depth:

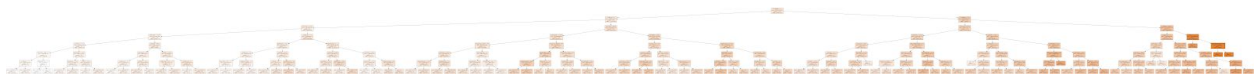
Depth 3:



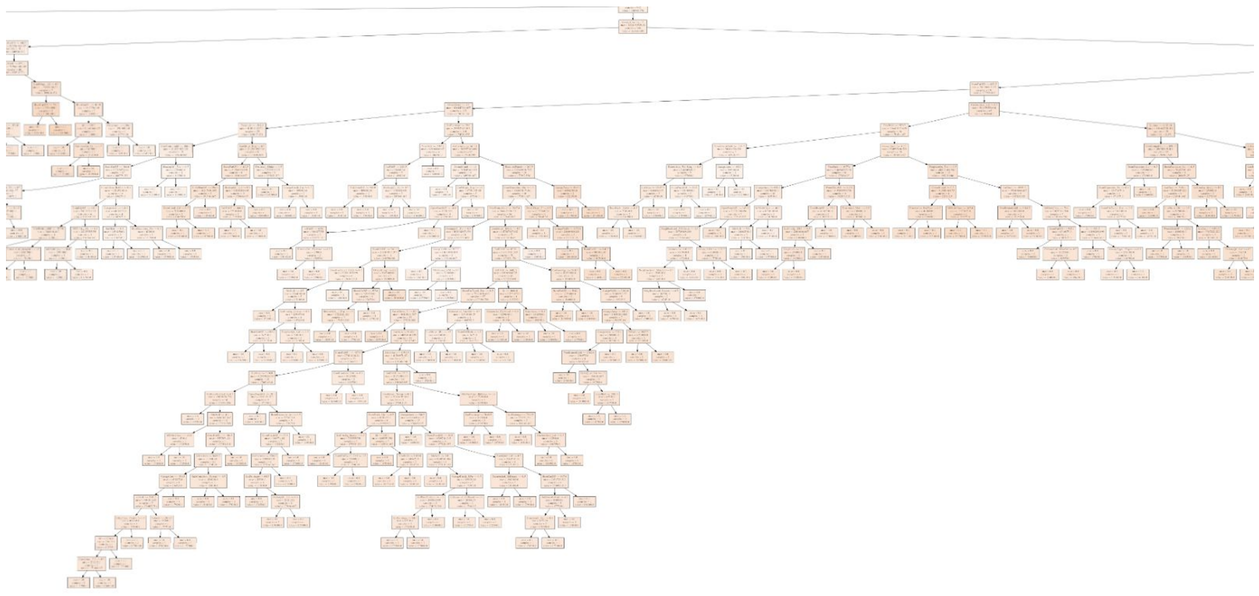
Depth 5:



Depth 7:

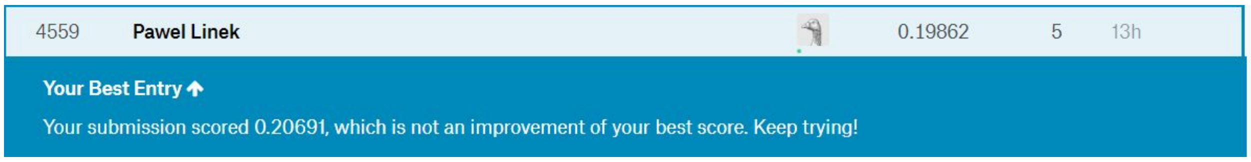


Depth9: *(Too large for google docs)*
Part of the Unlimited Depth: *(Approximately Depth 28. Also too large for google docs)*



Because the images of depth 9 and unlimited depth were too large for google docs to handle, I simply put an image of about ¼ of the full unrestricted decision tree. The Mean squared error (MSE) of the leaf nodes of the unrestricted decision tree were all 0. This means that the nodes at the bottom represented perfectly some Sales price of a house in the original dataset. It essentially memorizes the data and results in overfitting. Similarly, the depth 3 decision tree's leaf nodes had MSE errors nearing 1 billion. This obviously means that the decision tree doesn't generalize the data too well, (or in other words, underfits).

For the decision tree we used Kaggle as our main accuracy test. By submitting the outputs our classifiers of various depths produced into kaggle, we generate the Root Mean Squared Logarithmic Error which peaked at 0.19862 Error for the classifier with Depth 9. Our top score placed us in spot 4559/5489 just using a simple decision tree classifier.

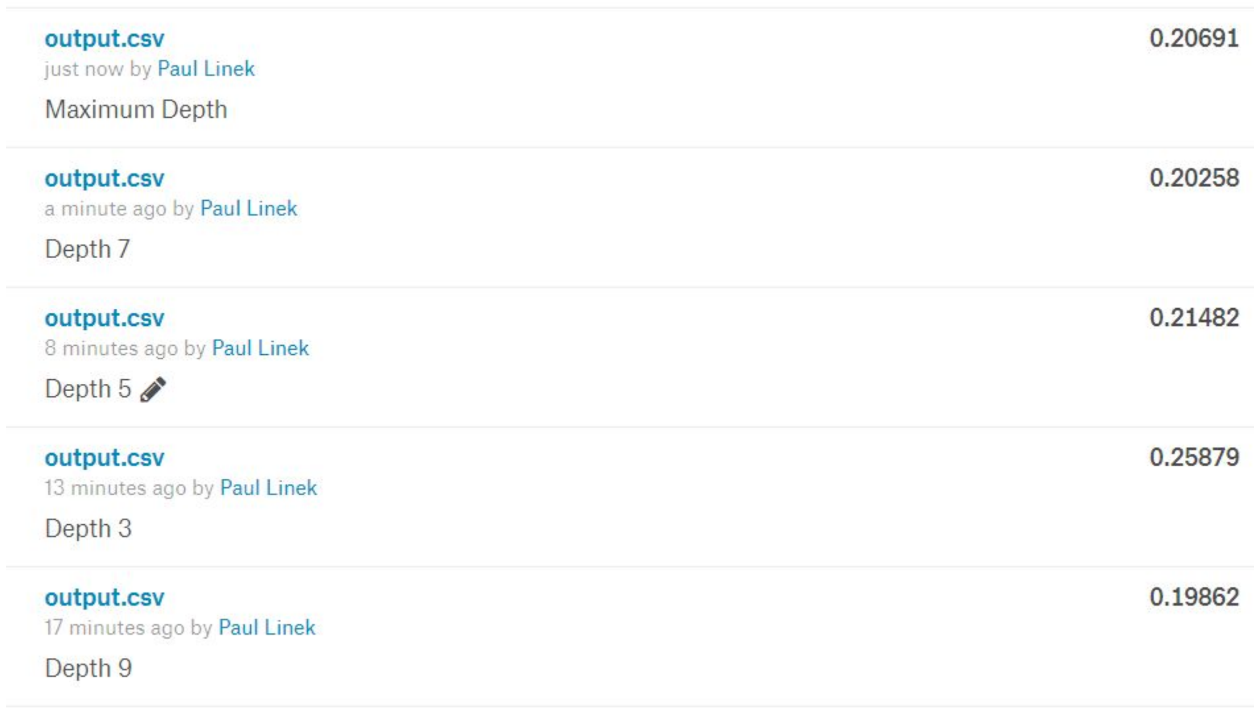


4559	Pawel Linek	0.19862	5	13h

Your Best Entry ↑

Your submission scored 0.20691, which is not an improvement of your best score. Keep trying!

Below is the list of submissions on kaggle. The score is on the right, and the description about depth is on the left:



output.csv
just now by Paul Linek
Maximum Depth
0.20691

output.csv
a minute ago by Paul Linek
Depth 7
0.20258

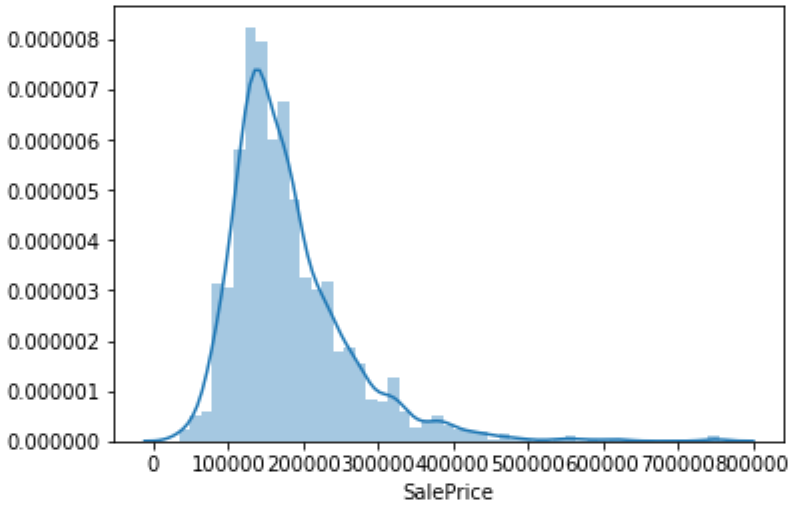
output.csv
8 minutes ago by Paul Linek
Depth 5
0.21482

output.csv
13 minutes ago by Paul Linek
Depth 3
0.25879

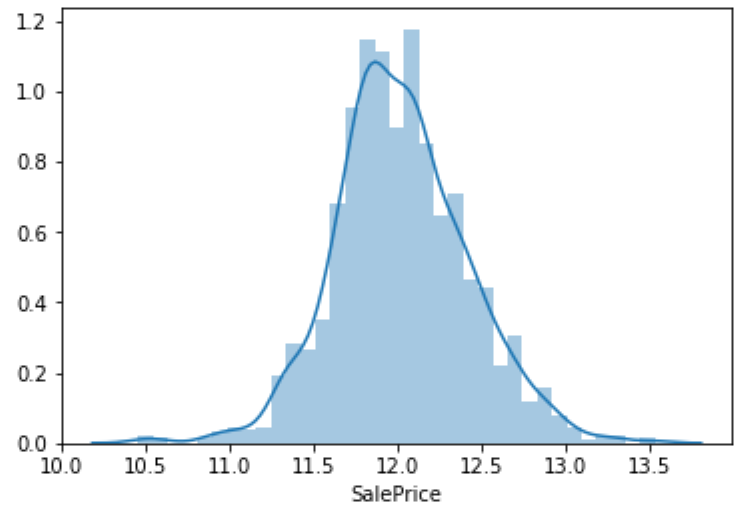
output.csv
17 minutes ago by Paul Linek
Depth 9
0.19862

Multiple Regression:

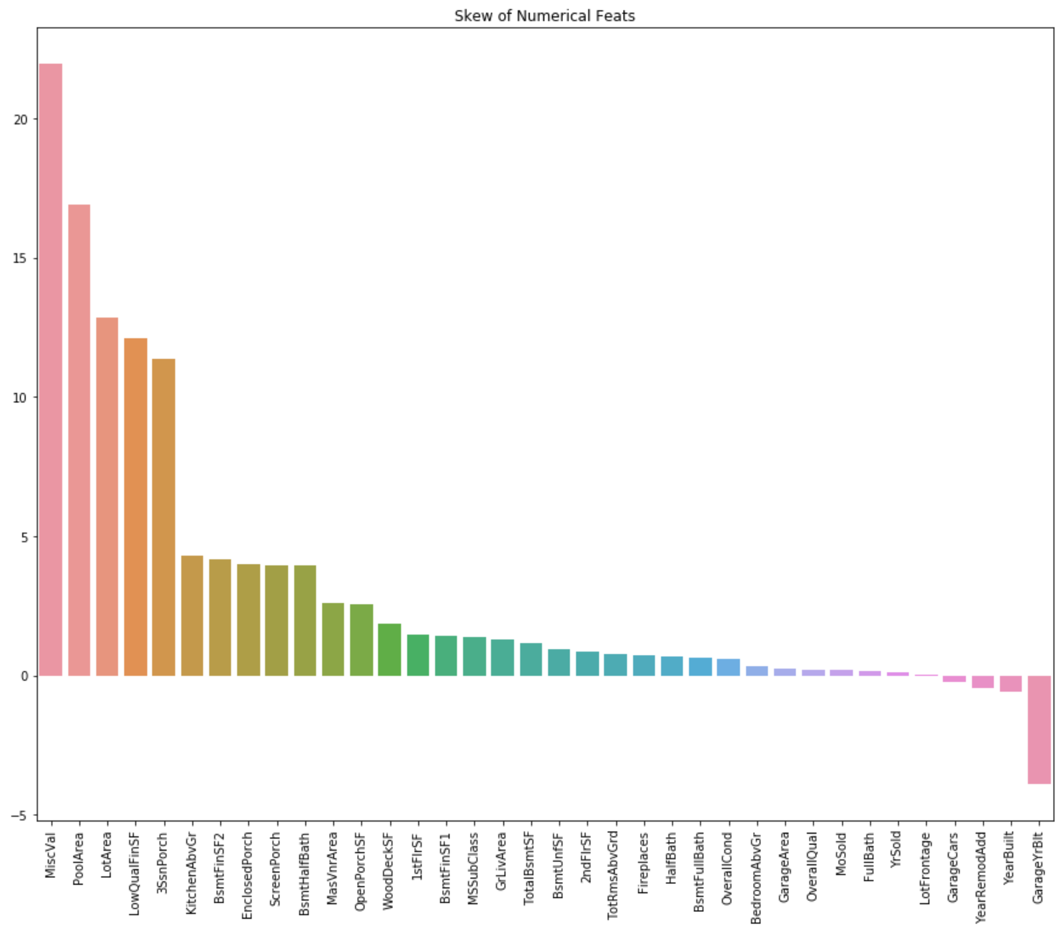
Looking at the dataset it was easy to tell that regression would be an optimal choice. The data when plotted over the SalePrice showed favorable trends. Seeing that we decided with regression the first thing we did was look at the distribution of SalePrice shown below.



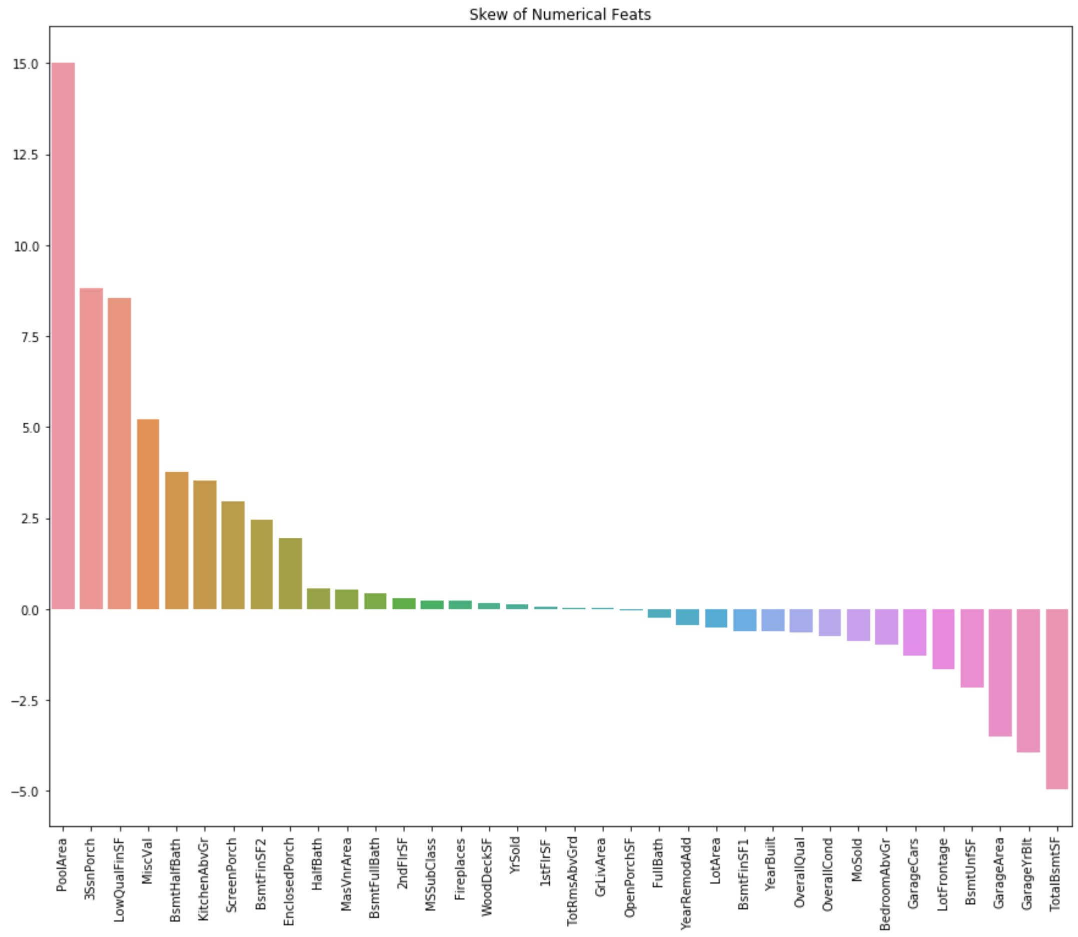
The graph shows the data is heavily right-skewed. This is a problem because our models would not be as accurate because it is very prone to outliers that will pull the predicted model in a direction we do not want to go.



This new distribution plot shows the SalePrice after applying ln(1+SalePrice) to the data. This gives us a more centered distribution that we can work with. The next challenge we found was fixing the skewness of the numerical training data. The data has 43 categorical and 36 numerical columns. To fix the categorical data it was done with just filling NA data entries with ‘None’. This allowed them to have no weight in the regression process.

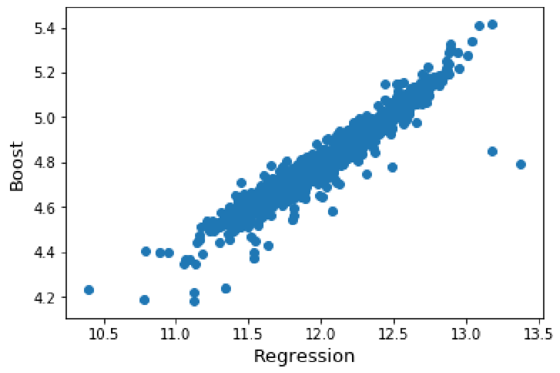


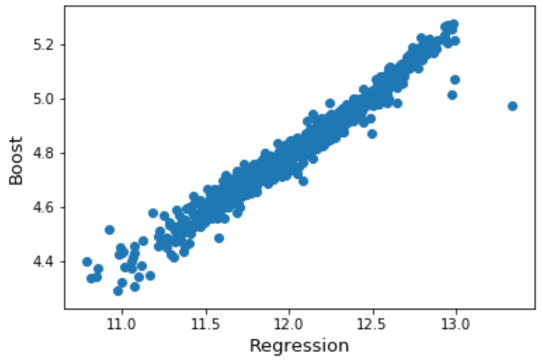
Above shows the skewness of the numerical data. You can see that the data is also positively skewed, but there are also still some negatively skewed data. To fix the skewness of the data wasn't easy and still wasn't completely perfect as there was still some skewness on both sides. But the same graph with the same features shows the middle (around 0 skewness) to have much more of an impact as it plateaus. This is shown in the graph below.



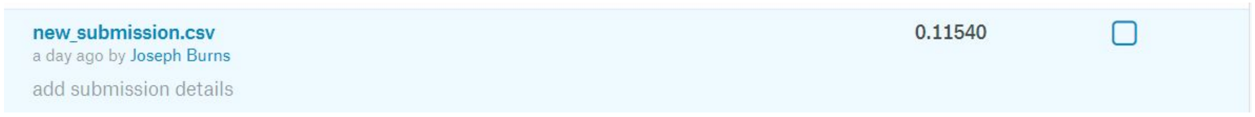
For modelling we chose to use 6 different regression techniques: Lasso, Ridge Regression, Support Vector Machine, Elasticnet, Gradient Boosting, and XGB Boost. We then blended them into one model, assigning different weights. For the regression techniques, we used a RobustScaler() to help even more with outliers, hoping they would be ignored. For boosting techniques, we messed around with some hyperparameters to help cut down on compile time. Gradient Boosting and XGB used a smaller learning rate and a high number of estimators, this was used to better learn the data at the cost of a little more time to compile. To combat this we made the max_depth of the tree smaller to make the model more compact and to prevent overfitting the training data. The huber loss helps Gradient Boosting because we are not expecting main outliers remaining in the data after preprocessing. Increasing alpha on XGB Boost allows the model to be more conservative. On both, they need a random_state value or the tree selected will be random every time meaning the results will not be consistent.

The models are then scored using cross-validation and k-folds = 10. Using this we can see the Root Mean Squared Logarithmic Error (RMSLE) of each model on the training data. Using this helps us with the weights when blending the model. XGB Boosting performed best on the model having the lowest RMSLE. After fitting each of the models to the data, we combined the regression techniques to the boosting techniques. This resulted in the graph below left when predicted on the training data. This shows the data staying closely together with few outliers that may have been missed through the cleaning of the data. The graph on the shows the prediction of the test data with the regression and boosting techniques. This data is much more evenly spread and has fewer outliers than the training data.





After tweaking with some weights to try and get the best model, the regression techniques held a 60% weight and the boosting held a 40% bringing us in total to the top 1000 in the leaderboard on kaggle with a RMSLE of .11540.

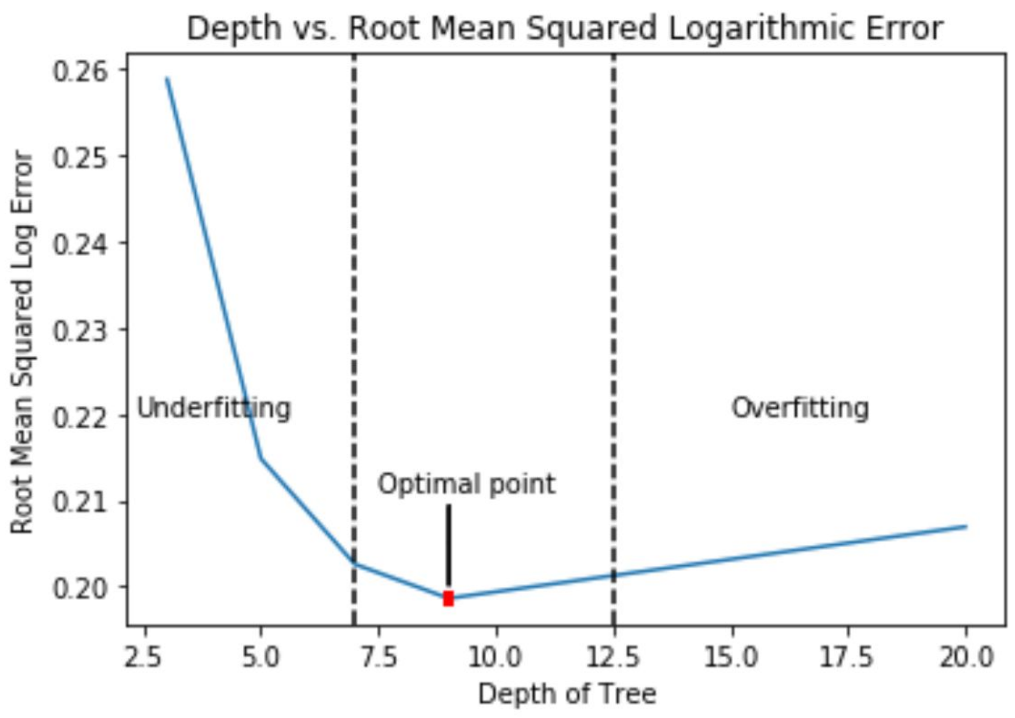


new_submission.csv 0.11540
a day ago by Joseph Burns
add submission details

Conclusion:

Ultimately, from this assignment we were able to determine that the decision tree classifier performs much better than expected acquiring a root mean squared logarithmic error(RMSLE) of 0.19862 at an optimal depth of 9. Any depth below that is considered underfitting as the RMSLE increases, and any depth greater than 9 is considered overfitting as RMSLE also increases.

The decision tree had some limitations for this specific dataset as a decision tree isn't the best fit for classifying numeric data. A decision tree is more so used for classifying groups like whether something is an apple, a banana, or an orange. We used a decision tree classifier to see how it would perform on a dataset it isn't necessarily meant for as well as to visualize over and underfitting which can be seen in the below plot:



In contrast to the decision tree, we used a multiple regression hybrid classifier to perform classification on the dataset which seems like regression would work well for (see scatterplot in Introduction section). Since a significant portion of the dataset seems positively correlated to the price of the home, multiple regression with ridge regression, elasticnet, lasso, SVR, Gradient boosting and XGB boosting. Cleaning the data proved to be the most important part of regression. Dropping data from the dataset that we believed to be unimpactful, or categories of data that had too much missing data. The data in this set was also severely skewed. Not fixing this would mean our prediction distribution would be too closely related to the training set. Fixing this allowed for more complex regression.

Ridge Regression is involved in our hybrid model as it is what helps penalize sum of squared(L2) coefficients, which seem to be outliers to have a less significant weight on our model. Similarly Lasso looks to penalize the sum of absolute values (L1). Our Elasticnet part of the model looks to minimize the following loss function:

$$L_{enet}(\hat{\beta}) = \frac{\sum_{i-1}^{n}(y_i - x_i'\hat{\beta})^2}{2n} + \lambda(\frac{1-\alpha}{2}\sum_{j-1}^{m}\hat{\beta}_j^2 + \alpha\sum_{j-1}^{m}|\hat{\beta}_j|),$$

Above, alpha is a mixed variable of, Ridge alpha=0 and, Lasso alpha=1.

SVR (Support Vector Regression) is the part of the model which tries to place the error within a certain threshold. This means we try to limit as much and as little range of error as we can for our model. This works similarly to how Support Vector Machine's work due to the nature of creating a decision boundary and slowly expanding the gap until it is comprised of an optimal amount of data points.

Finally XBG boosting and Gradient Boosting, uses many decision trees and combines them to create a more robust and powerful model. This is another reason a simple decision tree was used for the first classifier, to compare its performance to an XGB boosted model. XBG boosting was by far the model that took the longest to train on this dataset. We can see that without optimizing the hyperparameters within model creation. The same can be said for Gradient Boosting. This however can train a little faster with fewer hyperparameters defined. A lot of the tuning took place in during the process by changing the parameters higher and lower based on the base model. On both XGB Boost and Gradient Boost, one such parameter is max_depth, which is all the depths of the trees it makes. Obviously as concluded in the decision tree section a max depth of 9 is optimal, but time consuming and due to time restraints paired with the boosting algorithms proneness to overfit data we were not able to test this outcome. Selecting a smaller depth allowed the boosting regression to be conservative.

This model performed significantly better than a simple classifier model and was able to place within the top 1000 submissions on kaggle compared to the decision tree's placement of 4500. This is true for our initial guess as well by looking at graphs created by the data.