Project 1 Report

Dreycen Foiles [dfoiles2],
Olek Yardas [yardasol]

October 17, 2022

1 Introduction

We report on our efforts to use different regression techniques to predict the price of a house in Ames, IA given various different parameters. The two techniques that we present in this report is the ridge regression and the gradient boosted regression algorithm.

2 Methods

2.1 Data Preprocessing

For this project, we are given a data set made up numerous data points. These include numerical data like basement area, lot size, number of floors, year built, as well as categorical data like zoning type, heating type, street type. Our goal is to use these values to predict the price of the corresponding house. Before we can build a model, we must first clean the data in two steps: 1) remove missing and low-importance values, and 2) convert non-numerical data into a form that can be used with our regression techniques.

We used two data cleaning approaches for the linear and tree model. For the linear model we removed the following columns of data: PID, Garage_Yr_Blt, Street, Utilities, Condition_2, Roof_Matl, Heating, Pool_QC, Misc_Feature, Low_Qual_Fin_SF, Pool_Area, Longitude, Latitude. For the tree model we only removed PID because it was not data and Garage_Yr_Blt because it contained NaNs. For numeric data stored as strings, we used the `pandas.to_numeric` function to convert the data into the appropriate numeric datatypes. For categorical data, we used the common approach of creating a vector of length n with the ith entry being 1 and the remaining values being 0, where n is the number of different categorical values, and i corresponds a particular categorical value. Such a technique is called one-hot-encoding.

We used 95% winsorization on the data to remove extreme values to improve performance. This improves performance because after some threshold value, an increase in a feature's value does not contribute to an increase in a house's price. Therefore, placing a cap on the upper limits of a data set can improve a model's performance significantly.

3 Linear Model

For the linear model, we used a ridge regression. We had initially experimented with a LASSO regression but eventually saw that a ridge regression gave superior performance. We optimized the α for the model by cross validation and we found that an α value of 2 did the best on test sets.

Overall, the ridge regression was less flexible than the gradient boosted regressor because we needed to remove several columns before the performance was near the benchmark. On the other hand, ridge regression was orders of magnitude faster than the decision tree so linear models may be worth the lack of flexibility when working with very large data sets. In the end, the ridge regression performed better than the tree-based model, which we found surprising.

3.1 Results

Split #	RMSE	Runtime (sec.)
1	0.123	0.007
2	0.114	0.007
3	0.124	0.007
4	0.131	0.008
5	0.129	0.008
6	0.123	0.009
7	0.114	0.009
8	0.124	0.007
9	0.131	0.009
10	0.129	0.007

4 Tree-based Model

For our tree-based model, we used the `GradientBoostingRegressor` from scikit-learn. We first tried random forests but found it was textbfmuch slower than gradient boosting and quite a bit less accurate. We experimented with two main hyper-parameters, the number of decision trees and the max depth of the decision trees. After some experimentation, we found that a max depth of 4, just 1 above the default value, was optimal. On the other hand, the number of decision trees seemed to be optimal around 500. This was significantly higher than the default value. There were still some performance benefits of increasing the number of trees further but they were quickly diminishing and the computational cost of running the model was becoming intractable.

The gradient boosted regressor was much better equipped to handle bad data than ridge regression. We did not need to remove any columns and, in fact, removing columns worsened its performance. Therefore, I think we could say that decision trees work better for when you have data and you are not yet sure which features are actually valuable.

4.1 Results

Split #	RMSE	Runtime (sec.)
1	0.123	10.316
2	0.117	10.428
3	0.124	10.131
4	0.134	10.132
5	0.131	10.102
6	0.123	10.108
7	0.117	10.131
8	0.121	10.133
9	0.129	10.141
10	0.132	10.386

5 Conclusion

The important result we observed was that preprocessing was **essential** to have good performance. It did not matter what hyper-parameters we used for the two models, if the data was not processed properly, we would get poor results. Tuning the hyper-parameters of the two models would reduce our RMSE by only a small amount compared to the effects of One-hot-encoding and Winsorization. However, even with our preprocessing, we were unable to reach the benchmark for one of the test sets.