Comparison of Ridge/Lasso and XGBoost Regression for Housing Price Prediction

Shaun Niemeyer

Fall 2022

Introduction

The goal of this project was to predict the sale price of homes (in log scale) in Ames, Iowa, using a data set that contains 81 explanatory variables which describe various attributes of each home. We developed two models, one based on linear regression using both Ridge and Lasso penalties, and one based on a boosted trees model. Each model was evaluated on 10 training/test splits. The predictions from each model were required to result in an RMSE of less than 0.125 for the first 5 splits and less than 0.135 for the remaining 5 splits.

Pre-Processing

Both models used the same set of pre-processing steps, and the pre-processing for the training set was the same as for the test set.

The first step was to remove any variables that would not be useful for prediction. We removed the variables `Street`, `Utilities`, `Condition_2`, `Roof_Matl`, `Heating`, `Pool_QC`, `Misc_Feature`, `Low_Qual_Fin_SF`, `Pool_Area`, `Longitude`, and `Latitude`.

Next, we replaced any `NA` values in the data with zeros. Although only the `Garage_Yr_Blt` variable contained `NA` values, for future reusability, we used a generalized function to replace `NA`s with zeros for all variables in the dataset.

We decided to apply Winsorization to the variables `Lot_Frontage`, `Lot_Area`, `Mas_Vnr_Area`, `BsmtFin_SF_2`, `Bsmt_Unf_SF`, `Total_Bsmt_SF`, `Second_Flr_SF`, `First_Flr_SF`, `Gr_Liv_Area`, `Garage_Area`, `Wood_Deck_SF`, `Open_Porch_SF`, `Enclosed_Porch`, `Three_season_porch`, `Screen_Porch`, and `Misc_Val`. For these variables, we determined the upper 95% quantile of the variable based on the training data. We will refer to this value as the "threshold value." Any values for a variable, in either the training or test data sets that were above the threshold value were subsequently replaced with the threshold value, thus eliminating any extreme values in the explanatory variables.

Finally, we identified all non-numeric categorial variables in the data and converted these to dummy variables. For instance, the original data set contained the levels "Normal," "Partial," "Abnorml," "Family," "Alloca," and "AdjLand" for the variable `Sale_Condition`. So we created a new set of dummy variables `Sale_Condition_Normal`, `Sale_Condition_Partial`, etc., for each of the levels of `Sale_Condition` found in the data, and then removed the original variable.

When the test set contained different levels for a particular categorical variable than existed in the training set, this would result in the training and test sets having different numbers of dummy variables. So, as a final step, we added any variables to the test set that were in the training set but not the test set, and we removed any variables in the test set that did not exist in the training set.

As the predictions provided by the model were to be in log scale, we also transformed the response, `Sale_Price`, to be the natural logarithm of `Sale_Price`.

Model Development

For the model based upon linear regression, we used a two-step approach. We first used `cv.glmnet` with `alpha = 1` to perform Lasso-based variable selection. We then identified the variables selected from Lasso when lambda = `lambda.min` (i.e. the lambda penalty term resulting in the minimum mean cross-validated error). The training and test sets were truncated to include only this set of predictors. Next, we performed Ridge regression using `cv.glmnet` with `alpha = 0` on this reduced data set to obtain a new `lambda.min` value. Using this Ridge model and our new `lambda.min`, we then performed our predictions on the test set.

For the boosted trees model, we utilized the `xgboost` package. XGBoost was trained using the following parameters:

- `max_depth = 3`
- `eta = 0.05`
- `nrounds = 5000`
- `min_child_weight = 4`
- `verbose = FALSE`

The `max_depth` parameter indicates the number of splits to be used for each tree. Unlike bagging, boosting often works well with a small number of splits, because each tree is built sequentially, and depends on the trees that have previously been grown. By using smaller trees, we make small improvements to the model at each step, learning slowly.

In addition to using small trees, we want the learning rate (`eta`) to also be small. So we reduce `eta` to 0.05 from its default value of 0.3. This again reduces the chance of overfitting by slowing the learning process.

When we are using a slow learning approach, with a low number of splits and a low learning rate, we naturally have to increase the number of "rounds" or boosting iterations. So we set `nrounds` to 5000, a rather large number. This gives the model the time it needs to fully capture the signal in the data.

`min_child_weight` influences whether the algorithm will continue splitting leaf nodes that contain few observations. Specifically, if a split would result in a leaf node whose instance weight is less than the specified `min_child_weight`, then no further partitioning of that node will occur. Higher values for this parameter will presumably reduce overfitting in a rare number of cases. The default value is `1`, but we are using `4` as additional insurance against overfitting. This parameter has a very slight impact on the model, but was necessary in order to obtain the benchmark RMSE for one of the splits.

The `verbose` parameter simply indicates whether performance metrics will be printed. The project instructions indicated that this parameter should be set to `FALSE`.

Results

Both models successfully predicted the `Sale_Price` with an RMSE below the benchmarks for all 10 splits. As previously noted, the benchmark for the first 5 splits was 0.125 and the benchmark for the remaining 5 splits was 0.135.

RMSEs for each split:

Split

Linear Regression with Lasso/Ridge

XGBoost

1

0.1245623

0.1192514

2

0.1218657

0.1226989

3

0.1197646

0.1127228

4

0.1199335

0.1189489

5

0.1127674

0.1129485

6

0.1342772

0.1303821

7

0.1258317

0.1348335

8

0.1195389

0.1242578

9

0.1294987

0.1314821

10

0.1250598

0.1224601

Average

0.1233100

0.1229986

Run-Times for each split (in seconds):

Split

Linear Regression with Lasso/Ridge

XGBoost

1

1.0307181

14.69859

2

0.9788921

14.72654

3

0.9582829

21.75545

4

2.8028619

27.99387

5

2.2444160

28.23148

6

2.2266629

28.05070

7

2.2622979

16.07699

8

1.2945008

19.09704

9

0.9075341

19.45840

10

0.9379189

20.01674

Average

1.56440856

21.01058

This performance was achieved using the following system specifications:

- Dell Latitude 5520 Laptop
- Processor: 11th Gen Intel Core i7-1185G7 @ 3.00GHz 1.80 GHz
- Installed RAM: 16.0 GB (15.7 GB usable)
- System Type: 64-bit operating system, x64-based processor
- Operating System: Windows 10 Pro, Build 19042.1706

Discussion

The Linear Regression model and the Boosted Trees model performed very comparably; the performance of the two models, from the perspective of accuracy, can hardly be distinguished. However, the performance in terms of model training and prediction time is remarkably different. The XGBoost model took, on average, 13 times as long as the Linear Regression model.

It was also interesting to note how well both methods performed on such a complex data set with very minimal pre-processing. No feature engineering was required in order to meet the benchmark thresholds, although there were many potential opportunities for improving the data set with the addition of calculated or modeled variables. Nor did we need to remove any outliers or perform standardization and scaling. Therefore it is reasonable to expect that we could substantially improve upon these results with some dedicated effort and a more rigorous approach to pre-processing.