Project 1 Report

Paul Holaway (paulch2), Albert Li (xiangl9), & Matthew Schroeder (mas5)

October 14th, 2022

Statement of Contribution

All members of the group participated in multiple meetings, both in person and online, discussing the technical specifications of the modeling and the culmination of the project in a report. Specifically, Paul wrote the code for the gradient boosting model, while Matt and Albert were overlooking it. Paul also wrote the code for the elastic net model. Matt and Albert wrote the majority of the report, which Paul proofread and edited.

Overview

The goal for this project was to use two different models. One being a regression model with a lasso, ridge, or elastic net penalty, and the other being a tree based model, to predict the log transformed home price of homes in Ames, Iowa. For the regression based model, we selected an elastic net penalty and for the tree based approach, we used a gradient boosting model. The data set is composed of 2,930 observations and 83 variables that together give a clear picture of the qualities of each house. We were instructed to get the RMSE of the log of the sale price under 0.125 for the first five training/testing splits and under 0.135 for the remaining five training/testing splits, which was accomplished. The data set is being utilized in an ongoing Kaggle competition aimed at teaching advanced regression techniques.

Data Preprocessing

Reading in Libraries and Initializing Dataframes

First, we read in the Ames Data and test IDs files. We then imported the following libraries for our analysis: `fastDummies`, `gbm`, `glmnet`, and `tidyverse`. After that, we initialized a data frame to store the RMSE values for each of the ten splits for the two models and created a for loop that ran 10 times, one for each training/testing split.

Repeatable Preprocessing Process

For each of the 10 iterations of the for loop, the process was the same. First, we used the columns of the `testIDs` corresponding to the iteration of the for loop to create the `train`, `test`, and `test.y` data frames which we then wrote to CSV files of the corresponding name which we could later read back in. Immediately we loaded the `train.csv` file that we just wrote in the previous step. Then, we transformed the `Sale_Price` by taking the log and renamed it `log_Sale_Price` and moved it so that it was the first column in the data frame to make it easier to reference. Next, we filled all the values of `NA` with 0 using the `.isna()` function. After that, we conducted the winsorization of the training data for all of the non-factor variables.

The winsorization process helps reduce the effect of outlier variables without taking the observations out of the data set, instead it reduces the outlier values to a given quantile (chosen to be the 95th percentile in this case). Finally, we used the `fastDummies` package to create dummy variables for all of the categorical variables in the training data.

Modeling

Elastic Net

For the regression model, we tried both a Lasso and Ridge model at first, however, both turned out to be sub-optimal. Thus, we turned to an elastic net by utilizing the `glmnet` library in `R` to achieve the optimal threshold. Some specifications of our elastic net model are:

- All of the variables except for the response, the log of the sale price, and the ID number of each house were used to predict the response.
- Used both minimum and 1se methods for lambda to fit two separate models.
- Made two predictions based on the `new_train` data predictors with the two models, calculated the training RMSE for each model, and selected the one with the smaller training RMSE as the optimal elastic net model.

Gradient Boosting Machine

For the tree based model, we selected gradient boosting, which we implemented in `R` by using the `gbm` library. Here are the specifications of the model:

- We used all of the variables except for the response, (`log_sale_price`) and the ID numbers of each house to predict the response. We set the distribution parameter equal to Gaussian as we were facing a regression problem, not a classification problem.
- We made the number of trees large, specifically setting `n.trees = 2500` as we found that we were not getting under the expected RMSE of `log_sale_price` when we were using 500 trees.
- We set `interaction.depth equal = 4`, limiting the depth of each tree to 4. We chose a relatively small tree depth for computation efficiency reasons and fear of over fitting with a higher tree depth.
- Finally, we set the learning rate, `shrinkage = 0.05`. The default value of the learning rate for `gbm()` is 0.1, but since that was not hitting the benchmarks, we had to lower it. While this results in the model learning more slowly, it enables the model to predict a bit more accurately.

Testing Data Cleaning

What we did here was ensure that both the testing and the training design matrices had the same dimensions. To do this we created a new data frame named `fixed_test` with the numbers of rows as the testing data and the number of columns as the training data. We then used a nested for loop and only added the columns to `fixed_test` of the old testing data if they matched the column names of the training data. If we didn't do this we would have included dummy variable columns that were present in the testing data, but not in the training data, which would have not allowed us to fit the model. Any columns that were in the testing data but not the training data were filled in as 0 in `fixed_test`.

Results and Run Time

The standard for the accuracy on the testing data was stated as less than or equal to 0.125 for the first five splits and less than or equal to 0.135 for the last five splits. This was measured in terms of the RMSE of the

Table 1: RMSE and Computation Time vs. Model Iteration and Method

	Elastic Net		Boosting Tree	
	RMSE	Run Time	RMSE	Run Time
Iteration #1	0.1237140	3.49	0.1158990	69.95
Iteration #2	0.1192497	2.82	0.1187580	68.90
Iteration #3	0.1245197	2.87	0.1139657	69.56
Iteration #4	0.1227137	2.80	0.1164125	69.17
Iteration #5	0.1139809	2.79	0.1132320	69.12
Iteration #6	0.1336649	2.68	0.1272965	70.84
Iteration #7	0.1292647	3.08	0.1321661	70.51
Iteration #8	0.1216907	3.13	0.1240863	71.61
Iteration #9	0.1308148	2.53	0.1282775	69.83
Iteration #10	0.1251156	2.54	0.1248818	72.60

log of the sale price. Fortunately, the testing errors for both of our models were below the RMSE criteria for all of the testing errors. Our model creation times are listed next the the corresponding iteration's RMSE. On average, the elastic net models took 2.87 seconds to create and the boosting tree took 70.21 seconds to create.

Computer Specifications

- Processor: AMD Ryzen 3 3200G with Radeon Vega Graphics, 3.60 GHz
- Installed RAM: 16.0 GB
- System type: 64-bit operating system, x64-based processor

Conclusion

- **Elastic Net Model:** We realized the difficulty of finding the right alpha level when we first fit the model and using several acceptable values (0.5-0.8), we decided to utilize 0.5 for the final choice as it had the lowest RMSE.
- **Gradient Boosting Tree Model:** While trying to balance the running time and the RMSE for the tree model, we learned the fine balance between the learning rate and the number of trees. We then realized that the smaller the learning rate, the more trees we are going to need to achieve appropriate accuracy.
- **Comparison:** The two model has similar performance overall, with one performing slightly better than the other from time to time depending on which fold of the data was used. However, the Gradient Boosting Tree model consistently uses twice as much time as the Elastic Net model, which we think is a very important advantage for the elastic net model since the time difference could be more significant if the data size increased. It could translate into huge cost differences for real-world/corporate user cases. However, that does not mean Elastic Net Model is better from every perspective. At first, when we didn't perform winsorization, the boosting tree model was able to achieve a fairly good RMSE score, much better than the Elastic Net Model. Suggesting that the Boosting Tree model could be more resilient to extreme values and could be potentially used to set an initial benchmark at the very beginning of the model-building process.