Project 1: Predict the Housing Prices in Ames

Sarah Yang (sarahxy2), Ryan Berry (rpberry2)

CS 598 Practical Statistical Learning (Fall 2020)

1. Team members and contributions

We, Sarah (sarahxy2) and Ryan (rpberry2) formed a two-member team for this project. Ryan contributed to doing the research of the tree model and implementing it. Sarah contributed to doing the research for linear regression models, implementing it, and writing the report.

2. Technical details

2.1 Goal

Ames Housing Project aims to build two prediction models, a Ridge regression model and a randomForest model to achieve the performance target of 0.125 for the first five training and test splits and 0.135 for the remaining five training/test splits. RMSE will be the performance metric on the log scale of the sale price.

2.2 Overview of steps

To predict the sale price, we utilized R glmnet and xgboost packages. We closely followed the guidance provided by the professor's Piazza postings. To ensure our solution meets the performance target, we created testing driver codes to validate all ten split prediction results by sourcing submitted mymain.R and calculating RMSE on a log scale. We processed training and test data in separate steps. For each train and test data set, we conducted the following steps:

2.3 Data Preprocessing

The steps we applied for preprocessing our linear model vs our tree model are similar but not identical. The linear model benefited from 1) removal of known bad features, 2) encoding categorical features into numerical values, 3) filling NaNs with 0s, 4) and winsorizing features with outliers. For the winsorization step, we cut off (essentially plateauing) the data at the 5th and 95th quintiles. This prevents outliers from having a large impact. The tree model did not see performance gains from step 1 and step 4, so we excluded those from its preprocessing.

2.4 Training of Model 1 (Linear regression with Ridge Penalty)

Using sklearn's linear regression package, we build a linear regression model with ridge penalty. We actually used RidgeCV in this case to use cross-validation to choose the optimal tuning parameter alpha. The alpha values searched through the 10-fold CV is 0 to 40, with 0 meaning the ridge regression line is just a least-squares line, and increasing alpha values meaning the predictions are less and less sensitive to the features used. The matrix by RidgeCV for determining the optimal alpha value is the negative square root value, which is just the negative of the MSE, and the more negative the better since this is an error and not an accuracy metric. Then by using mode on the 10 training and test set' optimal optimal alpha values by 10-fold CV, it was determined that alpha = 5. Then we refit our ridge regression model on the each test dataset, using alpha = 5. RMSE was then calculated by taking the square root of the mean squared error. Then by a simple function, we checked that RMSE for each data set is below the threshold.

2.5 Training of Model 2 (Gradient Boosting Regressor)

We used sklearn's GradientBoostingRegressor to achieve the best performance with a tree-related model. After some hyperparameter tuning (manually adjusted) we identified no major performance gain past the parameters of n_estimators=300, max_depth=5, min_samples_leaf=1, learning_rate=0.1. This means that our model performs 300 boosting stages with trees of depth up to 5. During these 300 stages, the model is able to leverage the predictions from weaker models to create a stronger ensemble model. Each stage helps to improve the accuracy of our overall model.

2.6 Testing Result

Based on the output of sour mymain.py script, we getting following RMSE result:

Model	rmse1	rmse2	rmse3	rmse4	rmse5	rmse6	rmse7	rmse8	rmse9	rmse10
Ridge Regression	0.1232	0.11765	0.11303	0.11629	0.10953	0.13403	0.13305	0.12461	0.1313	0.12289
GradientBoostingRegressor	0.1224	0.1253	0.1158	0.1158	0.1127	0.1319	0.1349	0.1291	0.1342	0.1233

Based the above result, it seems the linear regression model achieved the performance target of RMSE below 0.125 for first 5 data sets, and RMSE below 0.135 for last 5 data sets. The GradientBoosting model also did very well, with only one RMSE failing the threshold.

2.8 Running time

Our code runs in about 11s per train/test split on a Macbook Pro 2.6 GHz Quad-Core Intel Core i7.

2.9 Interesting findings

We noticed some interesting properties of random forests as compared to linear regression models. In general, there is less of a need for preprocessing, as the tree-based models seemed to be more resilient than the linear regression models. Specifically, winsorization actually had a negative effect on the tree models. This is likely due to how a linear model will interpret outliers for their literal values, while a tree model will throw the outliers into a bin of greater than or less than a certain value. This causes less unwanted side effects from outliers.

3 Conclusion

Though the interpretability and preprocessing ease of the Gradient Boosting Regressor were useful features, the Ridge Regression model outperformed the Gradient Boosting Regressor across most test/train splits.

4 Acknowledgment

- Kaggle competition (https://www.kaggle.com/c/house-prices-advanced-regression-techniques)
- Professor Liang's CampusWire post for the Project 1 (Fall 2022)