Project 1
Predict the Housing Prices in Ames
CS598 Fall 2020

oorona2 - Oswaldo Orona

Prediction Methods

For this project I decided to focus on 2 methodologies.

1. Gradient Boosting Trees. The first method was mainly selected based on the ability to speed up results using the package xgboost and use a model that would not required feature selection.
2. Linear Regression using Lasso. The main purposed of this methods was to be able to find a good model and by doing also do feature selection using lasso penalty. The data set contains a lot of predictors and many of them are categorical predictors with many levels, therefore I considered that it was important for me to reduce the model size. Eventually, the original idea evolved to include a combination of lasso and ridge using Elasticnet penalty.

Data Pre-processing

Data Processing was done very similar in both cases but there are slightly different steps followed for each method. Below, I described the steps for each method.

Gradient Boosting Tree

1. It was obvious that the main problem was going to be the categorical variables. The data set includes multiple categorical variables and some of the them have a lot of levels. There is not a good way to assign values based on some kind of order, there was no way to easy consider many of this variables to be ordinal. The decision was to follow the recommendation from piazza and create dummy variables for K levels if the categorical variable had K>2 levels.
2. A list of columns from the train matrix is capture so that we can apply the same levels to the test data before prediction.

Linear Regression with Elasticnet

1. Some columns were dropped. BsmtFin_SF_2, Bsmt_Unf_SF, Total_Bsmt_SF, Second_Flr_SF and First_Flr_SF. The list of columns was discovered by doing a combination of winsorization and dropping columns. At some point during one of the runs I realized that not dropping columns and focusing on winsorizing the data was more effective reducing RMSE. After focusing on winsorization and selecting some columns, I decided to include some of those columns on the list of dropped columns with a better result overall.
2. Winsorization of data was done to a small selection of columns with a 95 percentile. The list of columns is Lot_Frontage, Lot_Area, Mas_Vnr_Area, Gr_Liv_Area, and Garage_Area. During part of the experimentation I discovered that winsorizing the data was more critical to the results than dropping columns.
3. A small set of corrections were done to eliminate NA from the Garage_Yr_Blt column and a fix of a data problem of a particular value listed outside the normal range.
4. The next 2 steps are the same as described for the first process.

Training models

Training the models and finding the optimal values for the 2 models were done in similar way.

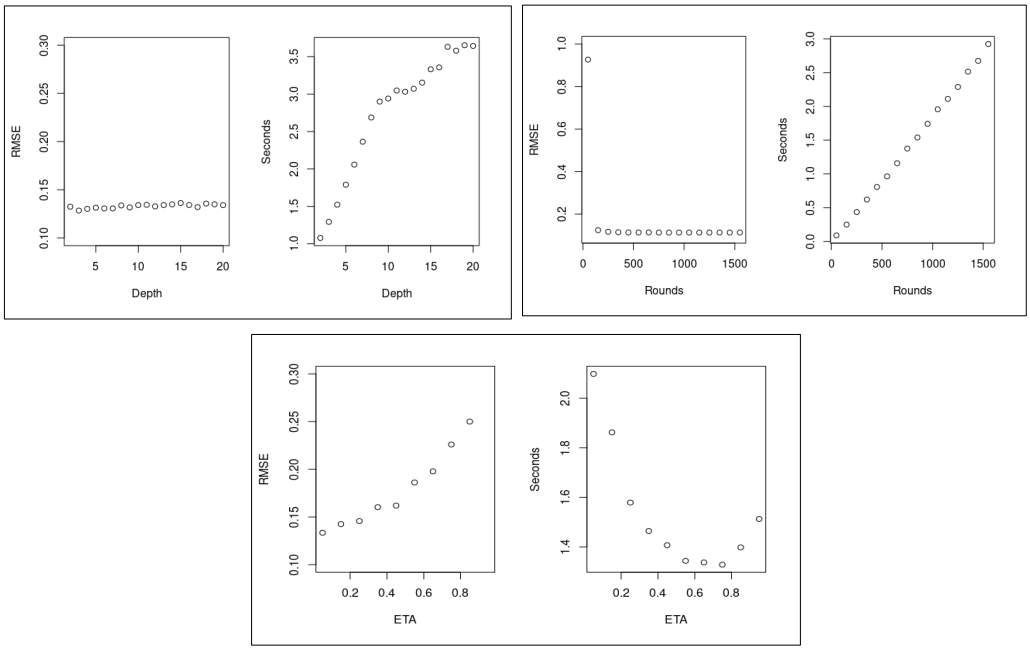
1. I ran the model with default values and doing a modification of only one parameter. i.e. eta for xgboost or lamda for glmnet.
2. After doing this I would choose the parameters that would have a bigger impact on the RMSE of the model and define a good estimate range of values. i.e. for xgboost max-depth range from 2 to 20 or lambda from 0 to 1 increment 0.1.
3. After choosing the parameters that look promising I did a parameter Cartesian grid search with the chosen parameters.
4. During each Cartesian search I used cross validation from both packages xgboost and glmnet to find the best response.
5. One the best parameters were chosen I would use those values in the actual code for submission.

Interesting Findings

The following points were of interest during the training of the models.

For Xgboost.

1. It gave good results very early on. No need to really do a lot of pre-processing of data but finding the best model was challenging because the model it has a lot of parameters that can be tuned. My Cartesian search at some point included 345 different combination of parameters. This was even with a limited set of values for each parameter. This process took at some point close to 3 hours to complete on a fairly powerful computer.
2. Xgboost was able to use all cpus and thread of the machine during the training phase.
3. Based on this approach not only I was able to find a good set of parameters but I was able to conclude that limited value of 532 nrounds was way more efficient and faster to compute without losing a lot predicting performance. The change was within of 0.001 of RMSE by going from 2000 nrounds to 532 reducing the compute time by 4 times.
4. Choosing bigger nrounds would not provide a significant improvement in accuracy and it would increase the computing time in a linear model. Other parameters would impact computation time without really improving the accuracy. As can be seeing the the below graphs.



5. The biggest increase of accuracy was by early on when the nrounds when to around 100.

For Linear regression.

1. Using all the predictors or removing some did not have a significant impact.
2. The biggest impact was winsorization of the data.

Results

Below is a table with all the results of this project.

Split	Rmse1(Gbm)	Rmse2(Linear)	Time(s)
1	0.112627	0.127717	3.300
2	0.12019	0.123091	3.509
3	0.111805	0.12349	3.477
4	0.11214	0.124654	3.586
5	0.107169	0.116973	3.215
6	0.130049	0.138518	3.378
7	0.130961	0.131745	3.268
8	0.126075	0.123347	3.283
9	0.128332	0.133683	3.256
10	0.118735	0.129256	3.293

Equipment

These results were obtained on a desktop computer running Ubuntu 20.04 using a processor AMD 3950x 3.8Ghz Base clock and boost of 4.6Ghz with 16cores 32 threads, 128Gb of ram.