Ame Housing Price Prediction

Chenzi Zhang, NetID chenziz2; Zhaoxu Qiu, NetID zhaoxuq2

Introduction

In our project, we are interested in analyzing the Ame Housing Price in Iowa, US. The goal is to build models to predict Ame Housing Price based on the original 81 features. The 10 train data sets we use to build models have 2051 observations with 83 columns. The 10 test data sets we used to examine accuracy have 879 observations with 83 columns.

In this report, we use three models. The XGBoost model reduced more work on manually remove variables but has more parameters for us to tune. The Lasso Regression model needs manually removing variables but has excellent variable selection performance, indicating features effect on Pirce explicitly. The GAM model is built based on Lasso feature selection, performancing as the most accurate. These three models have RMSE much more less than the benchmark and efficient running process.

Method

We choose to use three models: XGBoost, Lasso and GAM. We put these models seperately into three 10-round loops. In jth loop, we get jth train and test data set from original data, process data, build model, make prediction and do comperation between prediciton and `Sale_Price` in test data by RMSE. We will get one txt file for each model containing 10 RMSE's as result. Based on these result, we can compare the performaces of three models.

Missing value: There are 159 missing values in the Ames data, which are all from the variable `Garage_Yr_Blt`. Further investigation reveals that the 159 houses miss `Garage_Yr_Blt` because they never built a garage, so we replace the missing value in train data and test data by 0 using the function `ProcessMissingFixVal()`.

Model1: XGBoost

Before calling XGboost, we need to change the X matrix to a numerical matrix (no factors). There are two steps:

Step1: Seperate train and test data, remove varibale, obtain y and x data matrices, shrink values for response y. We set `PID` and `Sale_Price` as the variables we want to remove. Then, we get train.y and test.y by taking log on `Sale_Price` in train and test data. For predictor data matrices, we remove `PID` and `Sale_Price` by function `ProcessRemoveVars()`, gaining train.x and test.x. In train.x and test.x matrices, we set missing values as 0 by function `ProcessMissingFixVal()`.

Step2: Process binary model. For train.x and test.x matrices, we pick categorical variables, combine them with their levels as new column names for two new matrices, set appearance as 1 and absence as 0, get two binary matrices, combine these matrix with original numeric part by function `PreProcessingMatrixOutput()`, gaining numerical matrix for train.x and test.x data.

Then we build a xgboost model using `xgboost`. After tuning, we find the best parameter values are `nrounds = 300`, `max_depth = 3`, `eta = 0.2` and use them into our model. Then we do predict using this model based on test.x data, comparing the prediction result with `Sale_Price` in test data by RMSE.

Model2: Lasso

Before calling linear regression model with Lasso, we need to change the design matrix into a numeric matrix (no factors). There are four steps:

Step 1: Since values of the variables `Mo_Sold` and `Yr_Sold` are discrete, we set these as categorical variables.

Step 2: Seperate train and test data, remove varibale, obtain y and x data matrices, shrink values for response y. Some variables, such as `Street`, `Utilities`, `Condition_2`, `Roof_Matl`, `Heating`, `Pool_QC`, `Misc_Feature`, `Low_Qual_Fin_SF`, `Pool_Area` and `Longitude`,`Latitude`, tend to have just one level in the training data, since the other levels, except a dominate level, are rare and they can be confounded with the intercept, so we just remove these variables by function ProcessRemoveVars(). In train.x and test.x matrices, we set missing values as 0 by function `ProcessMissingFixVal()`.

Step 3: Process binary model by function `PreProcessingMatrixOutput()`, and do the same as XGBoost.

Step 4: Processs the outliers. Some variabels, such as `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`, have outliers, so we apply winsorization on these numerical variables using the function `ProcessWinsorization()`. In the function `ProcessWinsorization()`, we compute the upper 95% quantile of the train column, denoted by train.lim; then replace all values in the train and test that are bigger than train.lim by train.lim.

Up to now, the data preprocessing is finished. The next step is to use the function `cv.glmnet()` to find the lambda that minimizes the cv error, and then we use the minimum lambda to conduct lasso.model using the function `glmnet()`. Finally, we get the prediction and caculate the RMSE.

Model3: GAM

Before calling GAM, we need to change the X matrix to a numerical matrix (no factors). There are two steps:

Step 1: Seperate train and test data, remove varibale, obtain y and x data matrices, shrink values for response y. We remove `Street`, `Utilities`, `Condition_2`, `Roof_Matl`, `Heating`, `Pool_QC`, `Misc_Feature`, `Low_Qual_Fin_SF`, `Pool_Area`, `Longitude`, `Latitude`, `Mo_Sold`, `Year_Sold`, `PID` and `Sale_Price` by function `ProcessRemoveVars()`. In train.x and test.x matrices, we set missing values as 0 by function `ProcessMissingFixVal()`.

Step 2: Process linear variable and other numeric variable. Some variables, such as `BsmtFin_SF_1`, `Bsmt_Full_Bath`, `Bsmt_Half_Bath`, `Full_Bath`, `Half_Bath`, `Bedroom_AbvGr`, `Kitchen_AbvGr`, `Fireplaces` and `Garage_Cars`, only take a handful of unique values, which are not enough to fit a nonparametric curve, so we keep the linear term. And then we get the list of numerical variables whose nonlinear terms will be included in the gam model.

Step 3: Process categorical variables. Based on the coefficient of best Lasso model which has the smallest RMSE, we select 101 levels for important categorical variables, and use double "_" to separate variable name and the corresponding levels.

Step 4: Generate two data frames `train.dat.gam` and `test.dat.gam`. For linear.vars and num.vars, we take the corresponding columns from the original data. For select.binary.vars, we generate binary columns corresponding to "select.level.var".

Finally, we build a GAM model by function `gam()` using `gam.formula` we wrote above.Then we do predict using this model based on `test.dat.gam`, comparing the prediction result with Sale_Price in test data by RMSE.

Results

Evalution on Prediction Accuracy

Table 1: Prediction RMSE of First 5 Data Sets

model	set1	set2	set3	set4	set5
XGBoost	0.1179102	0.1191019	0.1156589	0.1196318	0.1131711
Lasso	0.1229074	0.1178241	0.1218115	0.1298183	0.1124678
GAM	0.1098040	0.1148686	0.1035657	0.1104255	0.1106502

Table 2: Prediction RMSE of Next 5 Data Sets

model	set6	set7	set8	set9	set10
XGBoost	0.1327133	0.1309916	0.1315819	0.1346636	0.1289655
Lasso	0.1331581	0.1264462	0.1206006	0.1300836	0.1237230
GAM	0.1173981	0.1186314	0.1086615	0.1182909	0.1193783

In general, the first 5 training/test splits perform better than the remaining 5 training/test splits for all these three models. We checked the first and following line in `testIDs`, finding that the first line of last 5 sets are the same, with number "1554". Then we checked the 1554th row in test data, seeing that this is an extreme observasion. After checking `PID` for 5 train/test splits, we find that pair sets(set1-set6, set2-set7, set3-set8, set5-set10) match each other for degree of over 60% and set4-set9 matches each other at over 50% degree. We further checked `Sale_Price` and other features for the unmathing rows, finding last 5 data sets tend to have extreme value. This may largely influence the result for last 5 sets.

The RMSEs of the last 5 splits are all less than 0.135, and the RMSEs of the first 5 splits are almost less than 0.125, except for the fourth training split using linear regression with Lasso. And Further investigation reveals that the variable `Overall_Cond_Poor`'s coefficient is -0.369233 and `Overall_Cond_Very_Poor`'s coefficient is -0.1349905 which is three times smaller than that of `Overall_Cond_Poor`. Then we check the fourth training/test split, and we find the Sale_Price of the housing whose overall condition is very poor in training dataset is much higher than that in test dataset, which may explain the weird coefficient and the bigger RMSE.

Overall, GAM model fits the dataset best among these three models. XGboost model fits better than Linear Regression with Lasso for the first 5 training/test splits while for the remaining 5 it's just the opposite.

Evalution on Efficiency

Table 3: Running Time (Unit: Second)

model	user	system	elapsed
XGBoost	71.341	1.460	21.862
Lasso	17.952	2.378	5.179
GAM	701.548	19.387	172.235

My computer system is MacBook Pro which has 2.3GHz Intel Core i5 Processor, 8 GB 2133 MHz LPDDR3 Memory. In order to improve the efficiency of our model, we use `parallel` package to achieve parallel programming. This method helps us reduce running time of GAM from over 600 seconds to 172 seconds. Running time is in the last column `elapsed`.

After comparing running time of these three models, we find GAM has the longest running time and Lasso has the shortest running time. Since GAM has the best prediction performance and Lasso has the same good prediction performance as XGBoost, we can conclude that Lasso model has the best efficiency – good performance and very short running time.

What's Also Interesting

After checking the coefficients from best Lasso model, we find that `Neighborhood_Green_Hills`, `Overall_Qual` with levels `Excellent`, `Very_Excellent` and `Garage_Qual_Excellent` have the largest magnitude positive effect on `Sale_Price`. Besides, we find that `Overall_Qual` with levels `Poor`, `Very_Poor`, `Functional_Sal` and `Functional_Sev` have the largest magnitude negative effect on `Sale_Price`.

Conclusion

For XGboost, we need to tune seven parameters and select the best set to make prediction, which is time-consuming. But we don't need to do much preprocess steps and the prediction performance seems good enough. For linear regression model with Lasso, the steps to preprocess is much more than other models. We need to change the design matrix into a numeric matrix by some complicated functions, but we don't need to tune parameters manually. Though Lasso model may rely more on the values of some important variables which may cause unstable predictions, it has the same good prediction performance as XGboost and has the shortest running time. For GAM, there are too many categorical level variables that we should select manually based on Lasso, which is an arduous task.

To sum up, it's better to choose XGboost model when we need too many steps to fix our original data. It's better to choose linear regression model with Lasso when we don't need to do much data cleaning and other preprocess steps, and it's a more efficient method. It's better to choose GAM model when we expect a more accurate prediction.