Project 1: Predict the Housing Prices in Ames

CS598: Practical Statistical Learning

Naomi Bhagat - nbhagat3, Michael Miller - msmille3, Joe May - jemay3

03 October 2023

Assignment Data

Program: MCS-DS Assignment post: campuswire

Team contributions:

Person	Contribution
Naomi Bhagat	Data pre-procesing, traning/testing general process, report Section 1
Michael Miller	Training/testing refinement and finalization, general process
Joe May	Report Section 1 and 2, debugging, and quality control

Section 1: Technical Details

The goal of this project is to predict the price of a home using certain features present in the Ames housing dataset. Importantly, the Sale price is predicted on a logarithmic scale. To achieve this goal, we built two prediction models: one using linear regression and one using a tree model to yield an RMSE of 0.125 for the first five splits and 0.135 on the final five splits.

Pre-Processing

Our implementation begins by installing the packages necessary to run the remainder of the script: `glmnet` and `xgboost`. We have a variable for debugging that is `FALSE` by default. When set to `TRUE`, the RMSE output (Section 2) is printed to the console; this is omitted when not running debug mode.

There were several pre-processing steps necessary for models. The `clean_input` function completes the pre-processing steps detailed in this section by taking a dataframe as an input and returning a cleaned dataframe. First, we replace NULL or missing values in the `Garage Year Built` column, as it is the only column in the data with such values.

Next, we removed a set of imbalanced categorical variables, and variables that do not offer additional insights. Most of these variables are highly biased, meaning that the entries for these variables skew heavily in favor of one category rather than a more normal distribution. This set includes `Street`, `Utilities`, `Condition_2`, `Roof_Matl`, `Heating`, `Pool_QC`, `Misc_Feature`, `Low_Qual_Fin_SF`, and `Pool_Area`. We also removed `Latitude` and `Longitude`, as they don't lend themselves to interpretable insights, especially since `Neighborhood` does a clearer and better job.

The next step is winsorization, which applies transformations to numerical variables, which we perform on `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`. This identifies values above the 95% quantile with that value. This creates a cap because there are diminishing returns for considering larger values for these numerical columns.

Finally, we convert any leftover categorical variables to 1-hot vectors. Our pre-processing step counts the unique categorical values for any character vector, creates a column for each value, and assigns a one to

the column representing a row's category, and sets the rest equal to zero. The last part of our 'clean_input' function puts 'Sale_Price' at the end and returns the cleaned dataframe.

Afterward, we have functions `force_col_match`', `print_formatted`, and `get_rmse`. `force_col_match` handles padding columns with zeros to ensure our training and test data have the same number of columns. The function `print_formatted` prints our predictions into the two submission files. Lastly, `get_rmse` function, which the sum of squares difference in actual and predicted y values, and takes the square root of the sum of squares divided by n.

Modeling

Our `train_and_eval` function takes the test and training data as inputs, processes the data, and creates the models. We first clean both the training and testing data with `clean_input` and `force_col_match`. Then we format the data as matrices as to properly fit as inputs into our modeling functions, described in the following sections.

To test our models, we import the appropriate test and training data. In debug mode, we split the test data into ten folds, iterate through the ten folds, train, evaluate, and print the RMSE for the linear and tree model for its performance and the time it took to train that fold. When not in debug mode, we train and evaluate the models, and write predicted `Sale_Price` values to the submission text files.

Linear Model

For the linear model, we first use 'glmnet' to create a temporary model using with an alpha of one (Lasso regression), which is then used to determine the selection of model variables. Then, to create the actual model, we use `glmnet` again with an alpha of zero (Ridge regression), restricting our data to the variables selected by the temporary model. Once the modeling is finished, we make our predictions of the test data.

Tree Model

We also create a tree model. While we initially attempted to use a randomforest model, abysmal results led us to instead utilize `xgboost` with a maximum depth of 6, an eta of 0.05, an nthread of 2, an nrounds value of 500, and verbose and print_every_n flags set to 0. Then, we make our predictions on the test data.

Section 2: Performance Metrics

The computer system this was run on was a Dell Inspiron 3501, 1.00 GHz with 8.00 GB of installed RAM for all 10 training/test splits. Below is a summary of the fold #, the RMSEs for the linear and tree models respectively and the time it took to train the models:

Fold	Linear RMSE	Tree RMSE	Linear Time	Tree Time
1	0.1248	0.1171	1.415	6.725
2	0.1211	0.1211	1.919	10.118
3	0.1201	0.1138	1.672	10.393
4	0.1208	0.1181	1.730	9.897
5	0.1140	0.1152	2.155	10.121
6	0.1341	0.1308	1.473	10.330
7	0.1266	0.1340	1.379	10.355
8	0.1192	0.1283	1.756	10.378
9	0.1305	0.1315	1.547	10.226
10	0.1257	0.1269	1.528	10.348