Practical Statistical Learning – Fall 2020

Project 1 (Ames Housing Data – Sale Price Prediction)

Team Information:
Balaji Sathyamurthy (balajis2)
Gowri Shankar Ramanan (gsr2)

Abstract:

This paper describes the study of optimizing prediction performance of the sale price with two selected prediction models (one based on linear regression model and another based on tree model), evaluated using a comprehensive housing dataset. The Ames Housing data set selected, contains 2930 observations and 83 explanatory variables involved in assessing home values in almost every aspect.

Goal:

Our goal is to predict the final sale price after splitting the dataset into 10 splits and evaluate accuracy of predictions against benchmarks set for each split. The first column is "PID", the Parcel Identification Number based on which 10 splits are made. The last column "Sale_Price" is the response variable that we are trying to predict. The prediction of final price is made in "log" scale with those explanatory variables. The following two prediction models were chosen for our evaluation and comparison study:

1. Linear Regression Model (Lasso)
2. Gradient Boosting Model (XGBoost)

Pre-processing/ Feature Engineering:

The dataset is split into 10 test/ evaluation sets based on a pre-identified list of PIDs in a file. For every split, the test set contains roughly 30% of the dataset and the remaining 70% of the dataset is used to train the model. The train and test data is loaded into a python pandas dataframe as a first step to start pre-processing. The following pre-processing and feature transformations were performed on both train and test data frames, and the feature selection was performed as a final step with the fitted Lasso model:

a. **Replace NA/ Missing values**
   All categorical features and numeric features are checked to see if there is a null or missing value. We have 46 categorical features and 37 numeric features in our dataset. The missing values are explicitly replaced to state as 'Missing' in case of categorical

variables. In case of numeric type variables, the value is replaced with the mode value calculated for that feature in that split. This action is performed on every train and test dataset splits separately.

b. **Same scale for all Year based temporal variables**
The Year based temporal features such as Year_Built, Year_Remod_Add, Garage_Yr_Built and Year_Sold are observed to be collinear in nature and can pretty much be brought in the same scale of Year_Sold as the reference year. This is done by subtracting other Years from the reference year of Year_Sold. This action is performed on every train and test dataset splits separately.

c. **Log transformation of the sale_price response numeric variables**
Applying log transformation on the Sale_Price numeric response variable. This action is performed on every train and test dataset splits separately.

d. **Replace Rare category labels in the upper 99th quantile with a constant "Rare"**
The distribution frequency of each categorical label data is checked individually and the top 1% rarest labels are obtained by calculating percentage counts of the label out of the total count falling below 1% threshold. These rare labels are normalized with a constant value "Rare" for reducing the impact of these outliers on the overall prediction. This action is performed on every train and test dataset splits separately.

e. **Apply Winsorization on the Rare Numeric variables**
The following numeric features were identified to contain many outliers in upper 95th and the lower 5th percentile ranges. The winsorization was performed using scipy.stats module to normalize these outliers by clipping and filtering the outliers in a symmetric fashion. This action is performed on every train and test dataset splits separately.
"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", "Misc_Val"

f. **Apply One Hot Encoding for every label in all categorical variables**
One Hot Encoding is performed on both train and test datasets separately on every split by creating dummy variables for every category label or levels. Pandas dataframe has the get_dummies function which can easily do this for each given reference categorical variable.

g. **Apply Standard Scaling**
Standard Scaling is applied for all numeric features which has shown to give better performance as the data is centered by removing the mean and scaling to unit variance.

h. **Perform Feature Selection using the fitted Lasso model**
The features are subsetted and selected from a fitted Lasso regression model with every train dataset in the split. In order to avoid any overfitting based on the train data, the alpha was carefully selected from various trials and finally tuned to the value of 0.001 which obtained the desired accuracy and performance

Overview of Selected Models:

Lasso is a regression analysis method that performs both variable selection and regularization in order to enhance the prediction accuracy and interpretability of the statistical model it produces. Though Lasso is originally formulated for Linear Regression models, it can be easily extended to a wide variety of statistical models. The alpha parameter is tuned to the value of 0.001 to avoid overfitting and to obtain the desired accuracy and performance.

XGBoost is an open-source software library which provides a gradient boosting framework for a wide variety of programming languages. We use python for this case study. It aims to provide a Scalable, Portable and Distributed GBM library. Salient Features of XGBoost which makes it different from other gradient boosting algorithms include:

- Clever penalization of trees
- A proportional shrinking of leaf nodes
- Newton Boosting
- Extra randomization parameter
- Implementation of single, distributed systems and out-of-core computations

After several tuning trials, the following hyperparameter values have been observed to give us optimal accuracy and performance for xgboost:
colsample_bytree=0.4, gamma=0, learning_rate=0.04, max_depth=60, min_child_weight=3.5, n_estimators=600, reg_alpha=0.75, reg_lambda=0.45, subsample=0.8, seed=42

Results & Performance:

The performance is evaluated based on Root-Mean-Squared-Error (RMSE) between the logarithm of the predicted price and the logarithm of the observed sale price. Please see the table below for the evaluation results observed from the studied models. Our performance is based on the minimal RMSE from the two models, and the following benchmarks were targeted and achieved:

- < 0.125 for the first 5 training/ test splits
- < 0.135 for the remaining 5 training/ test splits

As you can observe below, the benchmark is achieved consistently with XGBoost, often providing better performance than Lasso, thus XGBoost winning the game.

Table A: Prediction Accuracy Table for each split and model

Split#	Lasso (RMSE)	XGBoost (RMSE)
1	0.13004838275518266	0.12390215122663531
2	0.12206432054680316	0.11880925496627832
3	0.1338453193415672	0.11814894801686886
4	0.12732466741715032	0.11598279443746344
5	0.11705628005541024	0.10927103402589529
6	0.13790555912541722	0.13188370989812662
7	0.1337723346771136	0.1305690969089745
8	0.14000721688709813	0.1273737124366193
9	0.14185763316651626	0.13226795942074024
10	0.12446691884015162	0.1180592151071316

Table B: Total Time Taken (in milliseconds ms)

Split #	Train Prep	Feature Selection	Lasso Train	XGBoost Train	Test Prep/ Subsetting	Lasso Predict	XGBoost Predict	***Total Time Taken (in ms)***
1	236.715	40.511	13.391	354.796	232.749	1.422	33.651	913.235
2	221.159	36.469	20.253	757.117	241.312	1.167	27.248	1304.725
3	217.176	33.882	12.641	491.986	243.852	1.196	38.497	1039.23
4	219.954	43.248	18.012	660.25	535.027	1.739	42.708	1520.938
5	248.944	41.778	13.663	116.318	248.741	1.119	16.426	686.989
6	225.537	38.447	14.77	543.376	236.674	1.079	20.504	1080.387
7	430.472	57.936	49.793	944.931	246.121	0.944	22.551	**1752.748**
8	223.519	29.517	13.049	45.967	354.866	1.05	15.178	**683.146**
9	227.298	45.121	52.671	441.349	238.278	1.051	25.436	1031.204

10	228.744	33.968	12.889	666.139	226.468	0.948	21.375	1190.531

Evaluation Environment(s) Tech Specs:

- OS: MacOS Catalina 10.15.7 (19H2)
- Processor: 2.3 GHz Dual-Core Intel Core i5
- Memory: 8 GB 2133 MHz LPDDR3

Packages/ Modules	Version(s)
python	3.7.4
pandas	0.25.1
numpy	1.17.2
scipy	1.3.1
scikit-learn	0.21.3
xgboost	1.2.0

Interesting Observations & Conclusion:

It has been generally observed from the Prediction Accuracy Table (Table A) above that XGBoost is consistently performing better with prediction accuracy due to regularization and optimization which is performed as part of the boosting algorithm and it is very robust with respect to the outliers when compared to Lasso linear model. Interestingly, the split# 5 has the highest accuracy (lowest RMSE). As we observe the Table B for the total time taken, you can note that Lasso being a linear model takes very little run time overall to train and predict, whereas the XGBoost takes relatively more time to train and predict due to the in-built regularization and optimization through hypertuning. The run time observations show that split# 7 was the toughest split to train of all taking 1752.748 ms, while the split# 8 being the easiest with respect to workload with just 683.146 ms of total run time. This difference should be based on the variability of data and the outliers in the dataset for that split. The total time taken is mostly contributed by the time taken to train the dataset followed by any pre-processing needed. The prediction itself doesn't take much time once the model is trained and fit.

Team & Contributions:

Tasks	**Contributed by**
Data types and distribution analysis for every split	Balaji Sathyamurthy
Feature Engineering	
- Handle NA/ Missing Values	Balaji Sathyamurthy
- Handle temporal variables to reduce collinearity	Balaji Sathyamurthy
- Log transform numeric variables to increase prediction accuracy	Gowri Shankar Ramanan
- Handle/ Normalize Rare Category labels	Gowri Shankar Ramanan
- Winsorize Rare Numeric values	Gowri Shankar Ramanan
- Apply One Hot Encoding for all categorical features	Balaji Sathyamurthy
- Apply Standard Scaling for all numeric features	Balaji Sathyamurthy
- Perform Feature Selection using Lasso model	Balaji Sathyamurthy
Train/ Fit Lasso model from prepared datasets	Balaji Sathyamurthy
Train/ Fit XGBoost model from prepared datasets	Gowri Shankar Ramanan
Predict and Record RMSE for Lasso model	Balaji Sathyamurthy
Predict and Record RMSE for XGBoost model	Gowri Shankar Ramanan
Optimizations for Lasso model	Balaji Sathyamurthy
Optimizations/ Parameter tuning for XGBoost model	Gowri Shankar Ramanan
Summarize inputs, results & performance metrics (RMSE/ Run-time) for all splits	Gowri Shankar Ramanan Balaji Sathyamurthy
Documenting Report	Gowri Shankar Ramanan Balaji Sathyamurthy