Team and Contribution

Xiaozhu Ma (xiaozhu3) - data processing and final report
Fan Yang (fanyang3) - model training

Project Goal

The purpose of this project is to predict the house price using Ames housing dataset. Two categories of models were explored in the project: linear model and tree based model. Multiple models were trained in this project with the aim to:

- Built the final model that can provide the highest accuracy
- Compare the two types of models in terms of model performance (bias vs. stability), model expandability and training time, etc.

Data Description and Preprocessing

The data used in the project is Ames house data set with a total of 2930 records and 83 columns, which includes:

- The response variable house price: Sale_Price
- 81 predictors describing (almost) every aspect of residential homes. 35 numeric variables and 46 categorical
- Parcel identification number PID, a merge key and thus not used in the final modeling process

The response variable is converted to log scale in this project.
Based on initial data exploration, the following preprocessing was added to predictors:

1) Missing value.
   One variable 'Garage_Yr_Blt' has missing value in the data. We replace NA with 0. This variable was later dropped due to high correlation with 'Year_Built'.
2) One hot encoding for categorical variables.
3) Remove 8 highly imbalanced categorical variables:
   ['Utilities', 'Condition_2', 'Roof_Matl', 'Heating','Pool_QC', 'Misc_Feature', 'Low_Qual_Fin_SF', 'Pool_Area']
4) Remove 3 variables that are not intuitive.
   ['Street', 'Longitude', 'Latitude']
5) Drop 24 highly correlated variables. If the correlation between two variables is greater than 0.8, we only keep the one that is more correlated to the response variable.
   ['Bldg_Type__0', 'Bldg_Type__4','BsmtFin_SF_1', 'BsmtFin_Type_1__4', 'BsmtFin_Type_2__4', 'Bsmt_Cond__3', 'Bsmt_Qual__3', 'Exterior_1st__0', 'Exterior_1st__11', 'Exterior_1st__4','Exterior_1st__5', 'Exterior_1st__6','Exterior_2nd__13','Exterior_2nd__2', 'Garage_Area', 'Garage_Cond__3', 'Garage_Qual__3', 'Garage_Type__6', 'House_Style__1', 'House_Style__4','House_Style__5', 'MS_SubClass__6', 'MS_Zoning__2', 'Sale_Condition__5']
6) Feature engineer- see the 'feature engineer' section in the code for calculation details
   - We think house price are impacted by additional factors including house age, remodel, and total bathrooms. So we created additional variables to reflect this information.
   - The price covers time window from 2006-2010 which include both before and after financial crisis, so the house price is u-shape for this period. Since liner model does not work well in capturing u-shape pattern, we create year indicator in the hope to capture the pattern.

A total of 294 variables were included in the following model training process

Model Training

10 sets of training /test data were generated from the total house data set, with 2051 records in training and 879 in test data. The values of the hyper-parameters described in the following section is based on one set of

training data. We did not observe big differences in the parameter values across the 10 set of training data. So we fix the value for the sake of computation time.

1. Linear model

We first build a lasso model with L1 regularization parameter alpha selected through 5 fold cross-validation

- Package: liner_model.lasso from sklearn
- Model configuration: fit_intercept=True, normalize=True
- Candidate alpha values: np.logspace(-6, 0.1, 20)
- Final select alpha value: 8.439481965654006e-05

Then variables with beta greater than 0 are included in a Ridge model. L2 regularization parameter alpha is selected through 5 fold cross-validation.

- Package: liner_model.ridge from sklearn
- Model configuration: fit_intercept=True, normalize=True
- Candidate parameter: np.logspace(-2, 0.2, 20)
- Final select parameter: 0.32008340465997676

Final model performance and training time (for both training and test data, but not including the data processing time, like one-hot-encoding) for the 10 data set.

test_id	testing_error	train_error	run_time
0	0.133489	0.121695	0.672242
1	0.131904	0.116967	0.543091
2	0.143080	0.112938	0.597039
3	0.144494	0.113520	0.594220
4	0.144480	0.116482	0.588838
5	0.133544	0.121689	0.634058
6	0.131891	0.116954	0.540732
7	0.143323	0.112959	0.629905
8	0.144329	0.113516	0.600858
9	0.144514	0.116467	0.583402

2. Tree based model

We first tried random forest with hyperparameter selected through 5 fold random search.
Four parameters with the following values were selected to build the final model. However, the model performance is similar to what we got in Lasso/Ridge.
{'n_estimators': 800, 'min_samples_split': 0.001, 'max_samples': 0.5, 'max_depth': 10}

So we built Gradient Boosting Tree model with hyperparameter selected through 5 fold random search. Candidate parameter values and selected parameter values are listed below

- Package: XGBRegressor from xgboost
- Parameters we tuned:
    - n_estimator: number of boosting rounds
    - max_depth: max tree depth for base learner
    - learning_rate: boosting learning rate
    - gamma: minimum loss reduction required to make a further partition on a leaf node of the tree
    - subsample: subsample ratio of the training instance
    - colsample_bytree: subsample ratio of columns when constructing each tree

  Of these parameters, n_estimator and max_depth increase model accuracy, while others help control overfitting.
- Candidate and final values:

```
params = {
        'learning_rate' :[0.05, 0.03, 0.01], # best: 0.03
        'gamma': [0, 0.1, 0.5, 1, 1.5], # best: 0
        'subsample': [0.6, 0.8, 1.0], # best: 0.8
        'colsample_bytree': [0.6, 0.8, 1.0], # best: 0.8
        'max_depth': [4, 5, 6], # best: 4
        'n_estimators': [300, 500, 700] # best: 700
        }
```

Final model performance and training time (for both training and test data) for the 10 data set:

test_id	testing_error	train_error	run_time
0	0.124233	0.055658	7.446666
1	0.113829	0.055536	7.397985
2	0.122747	0.053265	7.352140
3	0.130424	0.054933	7.286645
4	0.129209	0.054720	7.426310
5	0.123453	0.055384	7.323329
6	0.113756	0.055956	7.418424
7	0.123991	0.053383	7.300784
8	0.129795	0.055180	8.041485
9	0.129959	0.054334	7.995538

Observation and Conclusion

In terms of final model performance, it shows that Xgboost model outperforms others. This is consistent to our expectations. Linear model does not capture non-liner relation and variable intercept, while tree based model overcome this shortcoming.

However, linear model is better in expandability. We can easily see the factors impacting the house price and their relation to house price. But for tree based model, even though variable importance shows the top predictors, it's not clear how those factors drive house price.

Unfortunately, the final model performance does not meet the performance target. For linear model, we tried winsorization, but did not see big improvements. More in-depth data exploration or feature engineer may be necessary to improve performance. For tree-based model, we observe a better performance, but it's still below the target. We tried larger tree depth and boosting round. Even though they help decrease the test error, we observed an even larger difference between training and test performance, which could be a sign of overfitting. Due to the time constraint, we are unable to explore further options.

There are several interesting findings:

- The L1 regularization parameter selected by sklearn is 8.43e-05, which is much smaller than what we saw in the R-examples in the lecture
- Comparing the base learner tree parameters between random forest and xgboost, it shows that random forest prefers deep tree(max_depth = 10), while xgboost select shallow tree (max_depth = 4)
- Even though Xgboost outperforms other models in terms of test data performance, we observe a larger variance between training and testing error of Xgboost. Further tuning other hyperparameters may help reduce the variance.

Computer system: MacBook Pro, 2GHz, 8GB Memory
Python package version: pandas==1.1.5, scikit-learn==0.24.1, numpy==1.20.1, xgboost==1.3.3