Technical Report: Project 1

Fan Yang (fanyang3) and Xiaozhu Ma (xiaozhu3)

February 20, 2021

We use Lasso and boosting tree (xgboost) to fit the data.

Pre-processing

- Log and centralize 'y': we first log the 'Sale_Price' variable and centralize it.
- One-hot encoder: for all category variables, we do the one-hot encoder using **sklearn.preprocessing.OneHotEncoder**.
- Standardization: For Lasso model, as the **Lasso** function in Python's **sklearn** does not offer the build-in standardization functionality, we have to use the **sklearn.preprocessing.StandardScaler** to standardize the feature matrix before we plug in the data to the Lasso model.

Implementation details

Hyperparameters:

We firs tune our model use the full data set to choose the hyperparameters.

For Lasso, we chose 'alpha (the constant that multiplies the L1 term)' as 0.1 from [0.5, 0.1, 0.01]; for the xgboost tree, we choose **{'max_depth': 3, 'num_round': 3}** from the following combinations, where **max_depth** is the max depth for a single tree, and **num_round** is the number of rounds for boosting.

```
[{'max_depth': 3, 'num_round': 3},
{'max_depth': 4, 'num_round': 4},
{'max_depth': 5, 'num_round': 4},
{'max_depth': 4, 'num_round': 5},
{'max_depth': 5, 'num_round': 5}
]
```

Missing Data in Prediction

After applying the one-hot encoder, the training data may not have the same variables as the testing data. Therefore, before we fit data into the model, we only use variables that appear both in the test data and training data.

Negative Predictions for Logged Sale Price

In the case the predicted logged sale price is negative, the we use the minimum value of sale price in training data as a substitute.

Accuracy of Test Data

Below is the test errors for the two models for each data split, and the mean test errors for all data splits.

```
# BoostingTreeMode
[0.045, 0.039, 0.048, 0.047, 0.048, 0.045, 0.039, 0.048, 0.047, 0.049]
# mean
0.046

# LassoModel
[0.054, 0.043, 0.058, 0.058, 0.051, 0.054, 0.043, 0.058, 0.058, 0.051]
# mean
0.053
```

Running Time

Running time is 62 seconds for all 10 data splits for the two models. To speed up the running time, we first hot-code encode the overall dataset, before split it to the 10 training/testing data pairs. In this way, we do not need to do the hot-code for each data split.

Below is our running environment:

```
# Mac
OS: Mac Catalina
Model Name:       MacBook Pro
Model Identifier:         MacBookPro16,1
Processor Name: 6-Core Intel Core i7
Processor Speed:          2.6 GHz
Number of Processors:     1
Total Number of Cores:    6
L2 Cache (per Core):      256 KB
L3 Cache:         12 MB
Memory: 16 GB

# Python version: 3.9
```