CS376 Term Project

Team 16

December 14, 2018

1 Model Descriptions

We mainly used *XGBoost* to learn the data. XGBoost is a popular gradient boosting library that offers efficient supervised learning. We chose XGBoost because our TAs suggested it during the TA sessions and it is widely used among Kaggle winners.

XGBoost internally uses *decision tree ensemble*. When we train model, it classifies the data with multiple trees and based on decision, it gets score. Then based on the loss, it learns the tree weights. For instance, example below illustrates how multiple decision trees work to predict house price of the Gotham city.

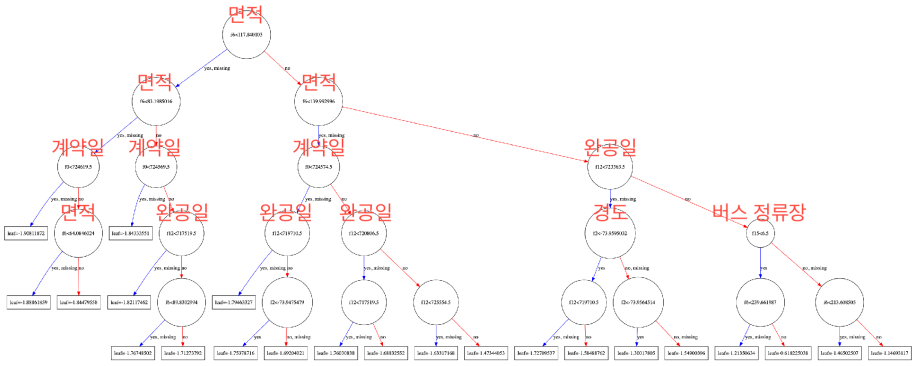

Figure 1: Example of boosted tree

For preprocessing, we first read the data with *pandas*. We replaced two date types *contract date*, *completion date* into three columns each: *year*, *month*, *day*. We shuffled the data to reduce variance and avoid overfitting. We intentionally

let the missing values intact for the base model, because XGBoost handles missing values appropriately; it learns how to treat missing values in each boosted tree.

We tuned the hyper-parameters including *max depth of the boosted tree* and *number of iterations*.

2 Unique Method

2.1 Drop outliers

We first carefully investigated whole dataset and found that there's outliers. We decided to drop records with price larger than 450,000 and altitude less than 30, i.e.,

$$price >= 450000 \vee altitude < 30$$

2.2 Drop unrelated columns

We dropped *completion month*, *completion day* as they are always january first. Also we deleted following columns as it seemed they are not contributing to the accuracy of final model.

- Altitude
- Road id
- Apartment id
- Whether an external vehicle can enter the parking lot
- The number of households in the apartment
- Average age of residents
- The number of schools near the apartment
- The number of bus stations near the apartment
- The number of subway stations near the apartment

One may find that many of them are categorical data. To handle categorical data, we first tried *one-hot encoding* from *scikit-learn*, but some of the columns make training take too much time while it doesn't contribute to the accuracy. When we made over 1400 columns, the process crashed as it uses too much memory. So we removed categorical data with too many values: *road id* and *apartment id*

2.3 Handling missing data with Imputer

We applied different missing value filling strategy to each columns. For instance, we filled *area* with mean value, and *builder id* with 0.

3 Libraries

3.1 Pandas

Version: 0.23.4
Purpose: To efficiently read data with appropriate labels.

3.2 xgboost

Version: 0.81
Purpose: To learn model with decision tree ensemble

3.3 matplotlib

Version: 3.0.2
Purpose: To visualize the data

3.4 scikit-learn

Version: 0.20.0
Purpose: To fill the missing values with *Imputer*, and *one-hot encode* the categorical data.

3.5 numpy

Version: 1.15.4
Purpose: To conveniently manipulate matrices.

4 Source codes

Git repository: `https://github.com/YujinGaya/ml-term-project/`

To inspect what we did, you only need to see *base.py* and *unique.py*. Following explanation is about the unique method. All the line numbers are line number of unique.py we submitted.

data_setting() function is to load and preprocess data, *performance()* is to measure the performance based on the criteria. and *train_n_predict()* is used to train the model and predict $\hat{y}$ for train and test dataset. *main()* calls the three functions appropriately.

If you want to test our source code with another train and test sets, you can edit the parameter of the *pd.read_csv()* call in line 30 to appropriate file name and train and test. Alternatively, if you only want to test our model without training it with new training set, you can just load the .model files

We tuned *depth of boosted tree* with *max_depth* parameter in line 147, *number of iterations* with third positional parameter of the *xgb.train()* in line 153.

5 Performance

We splitted the samples into 9:1 after shuffling the dataset. We used 9 part to train the model, and 1 part to validate.

For training set, performance of base model is *0.944927*, and performance of unique model is *0.950915*.

For validation set, performance of base model is *0.942786*, and performance of unique model is *0.948572*.

It takes 1 minute to train base model, 4 minutes to train unique model in the VM with 4 cores without CUDA (what we provided from the class).

Since there's not much difference between the performance from the training set and validation set, we think our model is not overfitted.