CS376 Term Project

Team 19

December 14, 2018

Please refer to section 4.6 when you want to run the code

1 Model Descriptions

We use XGBRegressor for our model. which is the boosting for regression problem.

1.1 Boosting

The concept of Boosting:

Boosting (Freud and Shapire, 1996) - algorithm allowing to fit many weak classifiers to reweighted versions of the training data. Classify final examples by majority voting.

To do regression problem, XGBoost uses the concept of Classification and regression tree (CART). CART contains the score for leaf node while the normal decision tree contain only decision value. With the score for leaf node, CART can use in classification problem.

This is the example of how CART work. For example, the question is whether, which person like computer given the demographic inputs.

The tree based model categorizes the each of user to each categories.

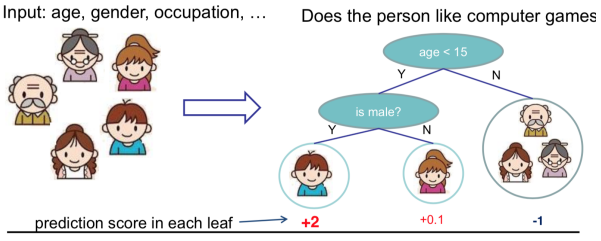

Figure 1: Regression Tree.

The single regression is not enough for solving the problem. So, we need to use Ensemble Tree. The following is the combination of two trees. Two trees

help each other to make the overall better performance.

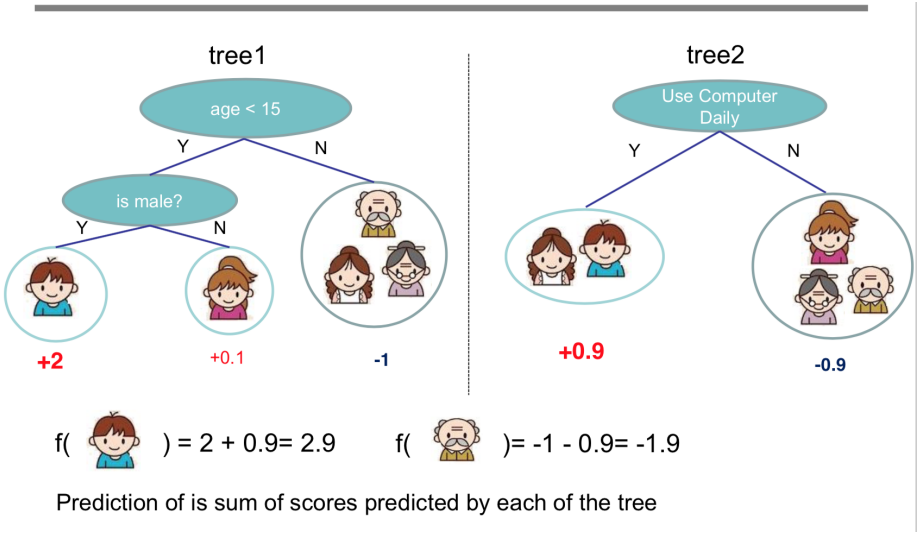

Figure 2: Tree Emsembles.

XGboost uses gradient tree boosting with regularization, which is one type of tree ensemble.

1.2 Objective function

The model can written in the following mathematical formula.

$$\hat{y}_i = \sum_{k=1}^{K} f_k(x_i), f_k \in \mathcal{F}$$

Figure 3: Mathematical formula.

1. K: the number of tree
2. f: function in the in F
3. F: set of all possible CARTs

The objective function should include lost and regularization. The reason for lost is to make the model has lower error in training set while the regularization made the model more stable and has the test error doesn't vary to much from the low error in training set.

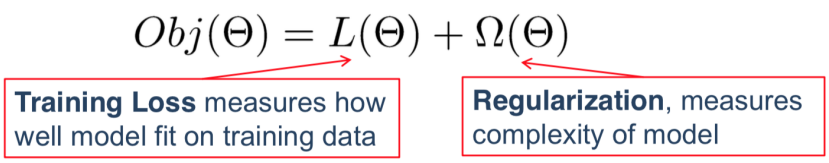

Figure 4: Objective function.

For XGBoost, the complexity of the tree or regularization contains number of tree and L2 norm of leaves score.

1.3 Learning tree ensembles

To train the model, fi that contain the structure of tree and the score on each leaf. The method such as SGD cannot be used in this problem and we use Additive Training instead.

1. First start from empty model.
2. Fix what we have learn and adding new tree by keep the remaining the same. The new tree should optimize the objective.

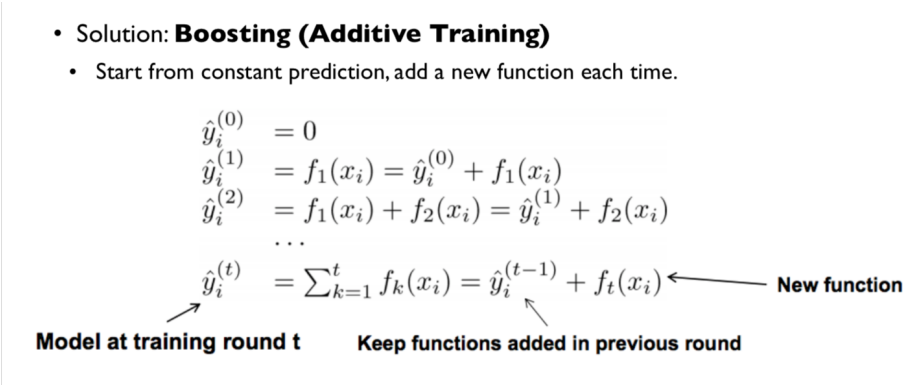

Figure 5: Additive Training.

The final objective function that we obtained by combined lost and regularization is in the following form.

$$\text{obj}^{(t)} = \sum_{j=1}^{T} [G_j w_j + \frac{1}{2}(H_j + \lambda) w_j^2] + \gamma T$$

Figure 6: final objective function.

The best weight for given struture q[x] can obtained from above objective function and best objective function is represented below.

$$w_j^* = -\frac{G_j}{H_j + \lambda}$$

$$\text{obj}^* = -\frac{1}{2} \sum_{j=1}^{T} \frac{G_j^2}{H_j + \lambda} + \gamma T$$

Figure 7: final objective function.

To obtained the best split, We can calculate the structure score of all possible spilt obtained above.

2 Unique Methods

2.1 Dropout

In order to measure the relative weights for the features we have used Pearson correlation coefficient, to compare how certain features have contributed to the price. Using this information we knew which features we can discard in the preprocessing stage.

1. According to the correlation statistics we keep the columns which have high correlation coefficient such as 'floor', 'area', 'avg_management_fee', 'number_of_cars_in_parking_lot' and etc.

2. we drop built_year, construction_completion_date and contract_date because we change it to another columns as we will discuss in section 2.2

2.2 Other unique methods

1. We add new column day_diff calculating the time between the start date of the construction of apartment and finished date, then we drop those two columns, since we have merged them into day_diff. We are doing this because the time period between starting and ending of construction can affect the price. In reality, the longer it takes to construct the apartment, the higher the price it should be.

2. We add new column "age" which calculate year 2018 subtracted by the built year column, then we discard built_year column. The reason behind this is quite similar to daycolumn, since the longer the age, the lower the price because old apartment should have low price in reality.

3. Since we have 1st class region id which is categorical data, we decided to make 6 more columns as ["one", "two", "three", "four","five"] for each value of 1st class region. As far as we can see in both training data and test data, the highest value of 1st class region id is 6. But we created only 5 columns for first values of 1,2,3,4,5 because we do not know the limitations of id, so we just discard the id higher than 5. We label them using LabelBinarizer which we will explain more in Library part.

4. We used LightGBM Regressor along with XGBoost and ensembled it. We chose LightGBM because shrinking and removing the coefficients can reduce variance without a substantial increase of the bias, this is very useful for us since we have a lot of features, and it can provide better result using the average and ensembling them together.

For more description about unique method we have used, please refer to this link
https://www.kaggle.com/anuartb/unique-py?scriptVersionId=8447458

3 Libraries

3.1 Numpy

Version: 1.15.2
Purpose: numerical computation on array of a value

3.2 Other libraries

1. Pandas
   Version: 0.23.4
   Purpose: to create data as csv file, easily handle data for each column, adding new column, dropping columns, using date_time for each column

2. sklearn
   Version: 0.20.1

Purpose: RandomizedSearchCV for finding best hyper parameter, KFold for finding score of validation for the prediction result, preprocessing to label the value using LabelBinarizer function, 3 sub-libraries BaseEstimator, RegressorMixin, TransformerMixin

3. scipy
   Version: 1.1.0
   Purpose: For parameter using in xgboost test which is to find the best hyper parameter for XGBoost.

4. xgboost
   Version: 0.81
   Purpose: to create xgboost model, using fit, predict, and score function from the model

4 Source codes

4.1 Model

The data and our code lies in the 'data' folder. The main components of our system lie across several python scripts. For code without unique method, we mainly use XGBoost only with some columns dropped. For the unique method, we use ensembling model on 2 gradient boosting models to average our bias and reduce the variance. Also, we removed unrelated features by checking their correlation with prices.

We chose XGBoost and LightGBM because they are famous for their fast training speed and higher efficiency. LightGBM tree expands horizontally compared to other popular algorithms and converges much faster. As a result, our optimized learning rate for LightGBM is much higher than learning rate of XGBoost.

We actually also tried Lasso Regression, but the result after fitting was not very well done, so we decided to discard it from the prediction.

4.2 Model component

- For based method (without unique method) we do the following
  1. The file "demo_train.py" contains several models that we compared. Open it.
  2. The function "xgboost" trains and validates the training set.
  3. We use "XGBRegressor" to fit the training set.
- For unique method, we do the following
  1. The file "train_test_unique.py" includes main function which will work the whole process, starting from preprocessing in preprocess function where we do unique method as described in section 2.2. Then we apply and Lightgbm and ensemble them together.

4.3 Testing part

- For based method (without unique method) we do the following
    1. The file "demo_test.py" is the place for testing part.
    2. The test set is preprocessed by using the function "pre_process" from "demo_preprocess_test"
    3. The file uses function "run_xgboost" to train the model and predict outcome.
- For unique method, we do the following
    1. We use 'perf' function which needs the model and data as arguments, then we average the models and get the final score.

4.4 Testing the mode with another test set

1. Go to the file "demo_preprocess_test.py"
2. Change the path from f = open('data_test.csv') to your path directory test file.

4.5 Tuning Hyperparameter

RandomSearchCV was use to randomly test the the value of hyperparameter in the specify range. It is from sklearn library.

1. Set the hyperparameter that we want to test in params, which includes "n estimators","learning_ rate", "gamma", "subsample", "colsample_bytree", "max_depth".
2. Using the RandomizedSearchCV with 5 fold cross validation and 100 iterations. It is important to make the cross validation in the step of choosing the hyperparameter as well as the train parameter.
3. The hyperameter was obtained to used for our model.

4.6 Running and testing the program

- Base test
  You can just run 'demo_test.py' normally using python command
- Unique test
  The file 'train_test_unique.py' contains our unique model with tuned hyperparameters. As we have used XGBoost regressor and LGBMRegressor for our model, it is possible to specify ratio hyperparameters for both models. In order to see complete list of hyperparameters. **The important part is you need to specify 2 files name when you run 'train_test_unique.py' file which are 1. Train data file name. 2. Test data file name.**

5 Performance

5.1 Scoring

We use 5-fold cross validation with pre-shuffled data, as it seems that data in the train set is grouped and it affects the score substantially. We import KFold from sklearn.model_selection by XGBRegressor which provide the score function, and we can get accuracy from it.
We also calculates the score for unique method by using another formula:

$$1 - \frac{1}{N}\sum_{t=1}^{N}\left|\frac{A_t - F_t}{A_t}\right|$$

5.2 Execution time

For base method, we took around 20 seconds to run the program

Our unique method takes 450 seconds to run fit and calculate their performance score using 5-fold cross-validation.

5.3 Analysis

For the base code which we did not use our unique method, we have accuracy around 93.50% and 95.56% with 5-fold cross-validation. We did not get accuracy as high as possible, since the way we drop the columns is still cannot be proved that they can get rid of those adversarial input. But they are most likely to give much better results, since we did not drop columns randomly, but using correlation of each columns regarding to the price which we want to predict.

Method/Model	5-fold CV	All data
Base/XGBoost	95.56%	93.50%
Unique/XGBoost	95.41%	93.62%
Unique/LightGBM	98.31%	96.99%
Unique/Ensemble	N/A%	96.52%

For code with unique method, we got 93.62% from XGBoost, 96.33% from LightGBM and Ensemble score is 94.44%. Again, we actually still use XGBoost as our base code, then we add LightGBM, merging some columns into new column such as finished date of construction and start date of construction which make sense if we think about the period of the construction, the longer the time it takes, the higher the price is, and this is one reason we create this column. Likewise, age for each apartment also contribute the same reason. Apart from this, we try to create new random by some categorized column such as "1st class region id" which we can create new columns from them using their values.

And as you can see, for unique method, we literally base on the result of ensemble which we have slightly better result compared to Base method.