Machine Learning Capstone Project

Yang Yang

Hoboken, New Jersey, 07039, USA
E-mail: yyang64@stevens.edu

Introduction

Project Overview This is a kaggle competition sponsored by Sberbank, Russias oldest and largest bank. It challenge kagglers by making predictions of house price using house transaction data and macroeconomic data. A good algorithm will mean a lot for both the bank and its customers, with which Sberbank can provide better service and analysis for their customers and at the same time their customers can be more confident whether to lease or purchase a house. Tons of reseach has already been done to understand what factors influence house prices, but, in a more finance oriented aspect. In this project, I will try to predict the house price using machine learning.

Data has already been provided by Kaggle,and it's available on the project main page. It onsists of 3 parts `train.csv, test.csv, macro.csv`. The train set is 30000*200 big, the test set is 7000*200 and the macro data set is 2500*100. More than 90% of features are numeric features and the left are categorical features. Train & test set are information about individual transactions. The rows are indexed by the "id" field, which refers to individual transactions(particular properties might appear more than once, in separate transactions). These files also include supplementary information about the local area of each property. The left part is macro, which is data on Russia's macroeconomy and financial sector. For more details, please

refer to the `README.md` file.

Problem Statement In this competition, we will predict the house price with the given data.More specifically, we will use more than 400 both numeric and categorical features to predict a numeric target, the house price. Firstly I will process the data, including filling null values, removing outliers, selecting features. And then I will build some single models and tune each of them with grid search cross validation. To increase the diversity among model I will try random forest, adaboost, neuron nets, and of course, xgboost. And in the final stage, I will build a second level model to ensemble them to further minimize the error. Details will be discussed later.

Metrics According to Kaggle, we will use RMSLE (Root Mean Squared Logarithmic Error) for this competition.This is quite similar to RMSE but it gives less punishment on large predicted values. This error will be calculated on my local cross validation folds and Kaggle leaderboard (a place where you compete your solution with other data scientists).

Analysis

Data Exploration and Visualization Since we have more than 40 thousand data points and more than 300 featres, so I won't be able to show every features here. I will take some samples to illustrate some characteristics of the data sets. Some characteristics are really problematic, which I will spend some time dealing with them later. Firstly, let's take a peek at the data. Figure 1 is the first few features of train data and Figure 2 is the first few features of the macro data. I skip the test data here but I will come back to it in the following sections.

For the first glance, we can see already that the data has many null values. We need to fill them with some values before training our models since most of algorithms don't accept null

Figure 1: Train samples

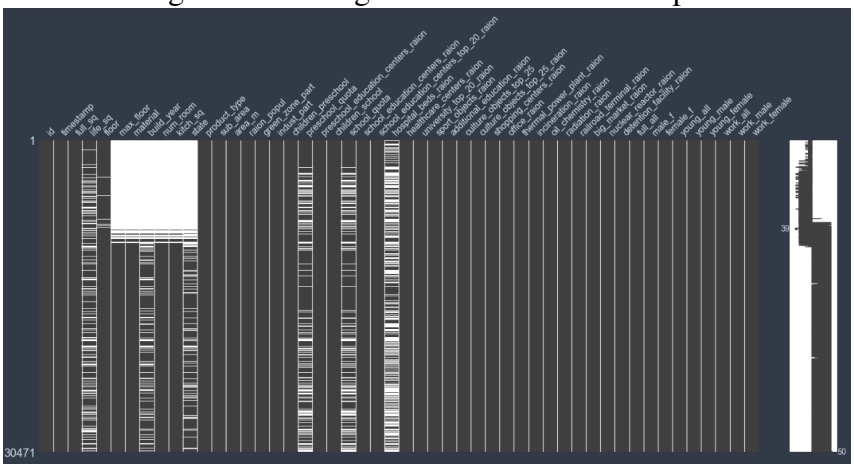
	id	timestamp	full_sq	life_sq	floor	max_floor	material	build_year	num_room	kitch_sq	state	product_type	sub_area
0	1	2011-08-20	43	27.0	4.0	NaN	NaN	NaN	NaN	NaN	NaN	Investment	Bibirevo
1	2	2011-08-23	34	19.0	3.0	NaN	NaN	NaN	NaN	NaN	NaN	Investment	Nagatinskij Zaton
2	3	2011-08-27	43	29.0	2.0	NaN	NaN	NaN	NaN	NaN	NaN	Investment	Tekstil'shhiki
3	4	2011-09-01	89	50.0	9.0	NaN	NaN	NaN	NaN	NaN	NaN	Investment	Mitino
4	5	2011-09-05	77	77.0	4.0	NaN	NaN	NaN	NaN	NaN	NaN	Investment	Basmannoe

Figure 2: Macro samples

	timestamp	oil_urals	gdp_quart	gdp_quart_growth	cpi	ppi	gdp_deflator	balance_trade	balance_trade_growth	usdrub	eurrub
0	2010-01-01	76.1	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
1	2010-01-02	76.1	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2	2010-01-03	76.1	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
3	2010-01-04	76.1	NaN	NaN	NaN	NaN	NaN	NaN	NaN	29.905	43.4054
4	2010-01-05	76.1	NaN	NaN	NaN	NaN	NaN	NaN	NaN	29.836	42.9600

vlaues. But before that, I want to see how spare exactly the data is, here I use a python package named `missingno` to make a more intuitive graph about missing values. From Figure 3, we can see some abnormalities of the data that the first quarter of some important features like max floor, build year, number of rooms are competely missing. Meanwhile some other features like hospital_beds_raion is very sparse.

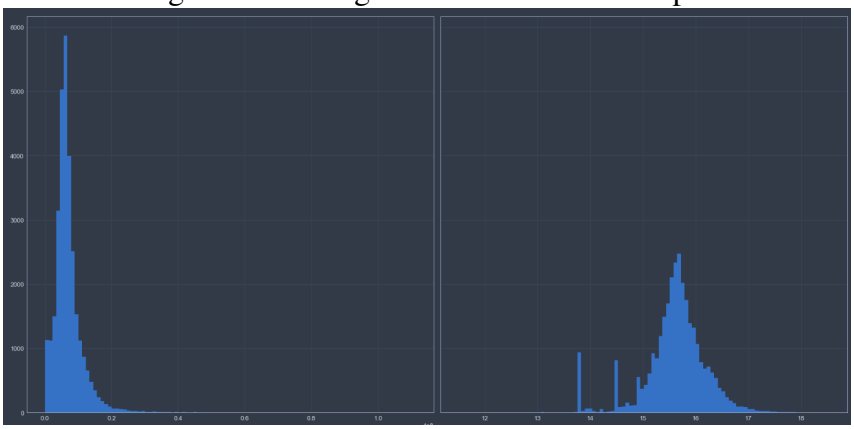
Figure 3: Missing values distribution sample



Also, from Figure 4, the target is highly skewed, we can apply a log on it. But interesting

thing is that, after transformation, the distrubition is somewhat normal while many samples certer around the lower tail abruptly. We have reasons to doublt the quality of such samples.

Figure 4: Missing values distribution sample



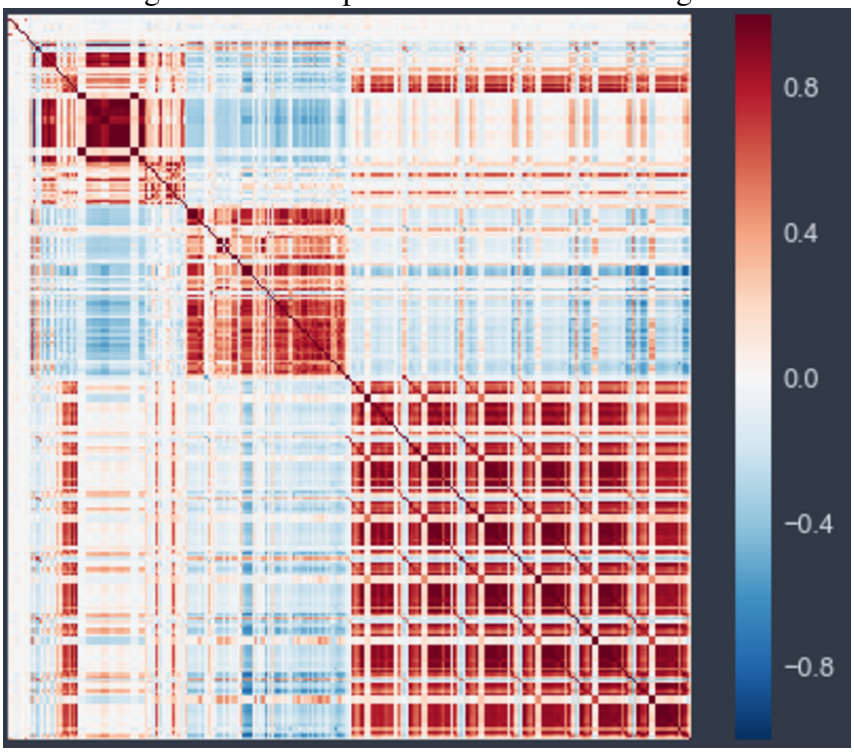
What's more, from Figure 5, we can see that some features are heavily correlated. From a higher perspective, they can gernerally be divided into 3 groups. Features within groups have positive correlation while correlation between groups is almost zero.

Some more eploration were done by calculation some usefull statistics, eg. by looking into the min/max values for each features, I found that some features have some impossible values such as zero room. I won't list them here, but all of them are treated differently during the stage of preprocessing.

Algorithms and Techniques In terms of algotithms, the first candidate is xgboost because we have a data set which is relatively large, filled with missing values and having different scale. When running an xgboost, it automatically deals with missing values, is invariant to scale, and designed to run fast by handling cache and memory. We can even cutomize the object we want to optimize, which therefore makes it almost suitable for every machine learning senarios. That's the reason you see xgboost everywhere in Kaggle competitions.

However, without genius feature engineering, a simple model usually won't bring you to

Figure 5: Heatmap for all features in training set



the top of the kaggle competition. A more common and realistic way is to ensemble many decnet models to make a final model. In other words, I will train a bunch of single models treated as frist level models, get out-of-samples from these models and treating them as training samples to train the second level model. This is called stacking, more specifically, I will use cross validation stacking. I will discuss about it in the later implementation section.

For now, given I will use stacking methods, I should also try larger variety of models to get better perfomance. And for this specific complex data set, I will choose these models apart from xgboost:

- Random forest & Extra trees: These two models are similar and have slightly different in terms of speed and randomization methods. The main idea behind is to randomly select multiple row sampes and column samples and train a weak tree models and then emsem-

ble them to reduce bias and variance. It's a different ensemble way so called bagging, which is different campared with boosting. So, with the idea of incresing diversity for our first level models, we should add them to our candidates.

- Feed forward neuron network: This is also called multi layer perceptron which can be thought as a combination of many groups of logistic regression. It linearly transforms input and apply some none linear function (activation function). Based on this main idea, it is theoretically able to simulate every function. But there are many problems, the most practical one is that there are too many hyper parameters to be tuned so actually, the FNN is not always good enough. But after all, it is also a promising candidate.

Benchmark As mentioned above, since xgboost is so powerful and convenient, it's always a good idea to just feed the raw data (with some simple label encoding to deal with categotical feature) to xgboost, xgboost will always yield a decent start point for me to treat as the benchmark. In this case, it scored with a RMSLE around 0.34.

Methodology

Data Preprocessing The data preprocessing stage is really difficult and troublesome. The data is messy– full of false data and null values, and outliers. Also the feature is not intuitive enough for me to select and drop. And also, given it's a kaggle competition, this stage is an iterative process that require you do experiments and verify them on both local cross validation error and kaggle leader board(in this weird competition, they always response differently). What's more, you should always be very careful to avoid overfitting the public leader board beacuse the final score is calculated on the hidden data, rather than those already shown us.

Regardless of problems listed above, below is my completed preprocessing details:

- Deal with false values: Some values like 0 max_floor are impossible and we should replace them with `np.Nan` and later fill them. There many possiblities, the main method to find them out is to scatter plot each feature in response to target price.

- Further cleaning: some rows are filled with null values or 0s, we should definitely drop them.

- Fill null values: For most features with null values, I choose to fill them with median since median is invariant to outliers.

- Feature encoding: For most categorical features, I choose to use label encoding. For a few of them, I use one-hot encoding if the levels for that feature are small. I also encode one feature `sub_area` by myself using the median of house price in that sub area.

- Feature engineering: I created some features based on timestamp, relative floor(floor number divided by max floor number), relative kitchen square(kitchen square divided by full suqare), macro effect. Also I applied partial PCA on highly correlated 'count' features to reduce dimension.

- Some more options: There are many other options I tried but I am not sure about the influence on the performance. Things like: scaling, merge macro data set to the training set. I set them as a knob to be tweak

Implementation The implementation can mainly be divided into two parts. After preprocessing the data, the first part is to use the processed data to build a fine tuned single model that has a good error both on local CV and kaggle leader board. The second part is to build a pipeline that can run a bunch of fine tuned models at one run and generate the out-of-sample and prediction, which are treated as the training and test data for the second level model.

- First part: The first part is bscially to tune model to get a decent RMSLE. I used `GridSearchCV` from `sklearn` package, where I set a range reasonable values for each hyper parameters. If I fit this estimator, it will go through every possible pair of hyper paramster combinations and run cross validation on the data. It will finally give us the best estimator and parameters that have a best mean cross validation error. I did the same thing on each of those selected models. It's hard to plot graphs to compare models, so what I did is to create a log file and add all infomation to it.

- Second part: for the stacking part, I build a class that can create out-of-sample and prediction for me and put those results in a folder so that when building second level model, I can directly read from the folder. When creating an instance of this class, all it takes is a dictionary that contains all the models I want, and for every model,they are also a dictionary that have every parameters that I will possibly tweak, including some flags telling the program how to preprocess the raw data.

- Submit: For the last step, I create another xgboost to be my second level ensemble model. It read the data generated from the level one models, trained on out-of-samples and target house price. And make prediction using the predictions given by level one models. I also applied bagging method to reduce variance.

Refinement For the xgboost bechamrk, without preprocessing the data, I scored around 0.34 RMSLE on leaderboard. For the fine tuned xgboost with missing values, outliers, false values being removed, I scored around 0.32 on leaderboard. And after I ensemble xgboost with fine tuned random forest, extra trees, neuron nets, sadly, I am still stucked around 0.32 on leaderboard. Further refinement such as some more feature engineering and methods like bagging didn't improve the score.

Results

Model Evaluation and Validation As already mentioned above, for every single model, I tuned them with grid search cross validation. By trying different features and preprocessing options, I think I have more than 100 versions of single models. Generally speaking, the performance for xgboost is better than other models as expected. All these models scored from 3.16 to around 3.50 on kaggle leaderboard. I carefully chose 3 xgboost, 1 random forest, 1 neuron nets(3 layers) and 1 extra trees by taking both diversity and performance into consideration.

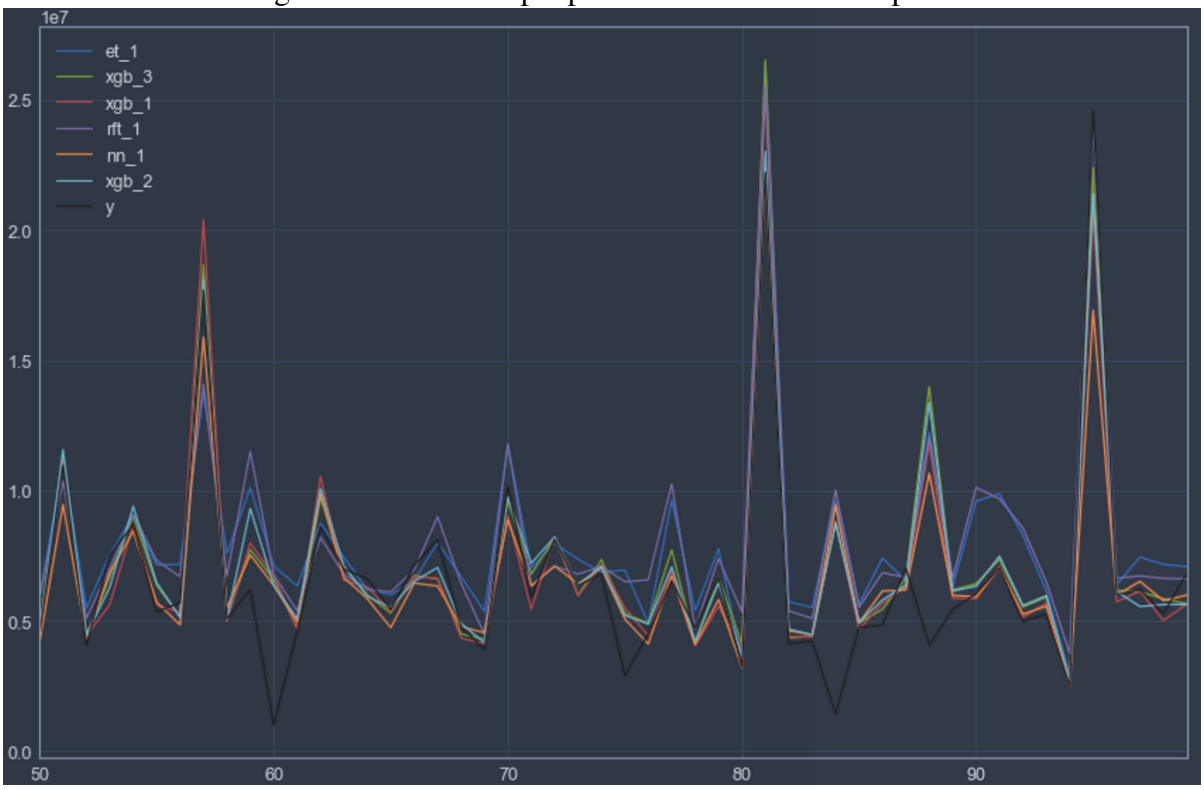
Justification Compared with the benchmark model, I gained 0.02 RMSLE improvement. Afterwards, by ensemble around 6 different models, although it did not make further improvement, it's almost certain that it becomes more robustic to predict future house prices. Since I have tried everything I can and the best score RMSLE on leaderboard is merely 0.30, I take this as my final solution.

Conclusion

Free-Form Visualization In this project I try different ways to build many single models, from which I carefully chose 6 model for ensemble. Figure 6 shows how well these model performed. For brevity, I only use 50 samples to plot this graph, but we can already see that these models generally performed well. Also by taking model diversity into consideration, we can see that for most of the data points, the predicted values are either a little higher or lower around the true values. If we further ensemble them, we will get a model closer to the truth. But we should also notice that the all my models don't predict well for extreme low values.

Reflection To sum up, I did the following things for my project: Explore the data and find a proper way to clean or transform it. Sencondly, build many single model and see their peer-

Figure 6: Out-of-sample prediction VS real house price



formance under each different settings. Thridly, choose some models under consideration of performance and diversity. Last but not least, ensemble them to get a better model.

There are a few things I might not do well compared with those kagglers who rank in the top. Feature engineering is the most influencial one. Although I have tried my best to do something useful, but there are still a bunch of features out there that I did not treat properly. For some features with more than 20%, I still simply filled it with median. Also I have already found some correlation between features while all I did is simply applying a partial PCA. Also I am sure that there are many potential features that might be usefull in prediction, I just don't how to create them.

Improvement My current ranking is not quite satisfying. Further improvement can be achieved by adding more different fisrt level models. Also further cleaning on the data should also be useful. Creating new features is the most important and it can be done by better uderstanding the meaning of each features. Also, a way to handle outlier can be useful as well,especially for those low prices, my models don't predict them well. There are so many things can be done, but for now, I think I might stop right here.