Sberbank Russian Housing Market

Machine Learning Nanodegree Capstone Project

Pramod Rao
August 20th, 2017

I. Project Definition

Project Overview

A lot of people working in data-related fields heard of Kaggle: a platform for running data science competitions. Currently Kaggle features more than 600K data scientists and dozens of well-known companies. A company defines its problem, quality metrics, publishes a dataset which may help solve it — and participants show off their creativity in solving company's problem.

Housing costs demand a significant investment from both consumers and developers. And when it comes to planning a budget—whether personal or corporate—the last thing anyone needs is uncertainty about one of their biggest expenses.

This project derived from Kaggle competition is brought by Sberbank, Russia's oldest and largest bank, helps their customers by making predictions about realty prices so renters, developers, and lenders are more confident when they sign a lease or purchase a building.

Problem Statement

This project is based on the Kaggle competition: "Sberbank Russian Housing Market". The aims to provide a solution by developing models using machine learning with deep learning to predict the reality prices. The project is largely rely large spectrum of features from the data set to predict the reality price.

The project will use the data provided by the Sberbank on the Kaggle competition forum. The dataset is classified as following:

- *train.csv, test.csv*: 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.
- *macro.csv*: data on Russia's macroeconomy and financial sector (could be joined to the train and test sets on the "timestamp" column)
- data_dictionary.txt: explanations of the fields available in the other data files

The whole dataset used in this can downloaded by agreeing to the rules of the competition.

The data provided by the Sberbank is completely anonymous and as well as uncleaned. Thus, the problem solving in the project starts from cleaning up of the data so as to make it fit for use in machine learning algorithms.

A brief break-down of the strategy used in the Sberbank Russian Housing Market challenge is:

1. Exploration of the raw dataset, identify anomalies/ outliers, create visualizations of different features. This is initial exploration is done in **Sberbank_Exploratory_Data_Analysis** notebook.
2. As mentioned earlier the datasets are unclean. The handling of outliers, feature engineering based on timestamp and validation on train-test data can be found in **DataClean** and **FeatureEngg** notebooks.
3. The modified data from the previous step is preprocessed so as to make it fit for ML algorithms (XGBoost and Multi-Layer Perceptron) and also a baseline score is obtained. This is accomplished in **XGBoost** and **NN_Model** notebooks.
4. Further tuning the hyper parameters of the algorithms to obtain significant improvements over the baseline models is achieved **XGBoost** and **NN_Model** notebooks.
5. Using stacking technique an ensemble of XGBoost and Multi-Layer Perceptron model is created to get a score better than the original model. This is accomplished in the **FinalStacking** notebook.

Metrics

Kaggle platform requires a company that runs the competition, define a transparent clear metric that participants compete on. Sberbank competition features Root Mean Square Log Error as a Metric. RMSLE is defined as:

$$\sqrt{\frac{1}{n}\sum_{i=1}^{n}(\log(p_i+1)-\log(a_i+1))^2}$$

Where p_i is the predicted value and a_i is the actual value.

Although the competition has set RMSLE as the metric, for convenience, throughout the project Root Mean Square Error is preferred. The main reason that, RMSE is readily available as an evaluation metric in most of the libraries, thus enabling smooth integration. In addition, RMSE and RMSLE do not differ much except that the latter doesn't heavily penalize large differences between actual and predicted values when they are very big numbers. Also, it can be observed that most of the public kernels and discussion in the completion use RMSE and have achieved very good results.

II. Analysis

Data Exploration:

Complete data exploration is done in "**Sberbank_Exploratory_Data_Analysis**" notebook.

The whole Sberbank dataset 30471 training sample and 7662 test samples. All the data points are indexed by "id". In this capstone project, the test dataset is untouched and is reserved for final prediction for the competition. It's noteworthy to mention, validating the train-test data it was found that test set is separated same as the train set, indication train-test sets were created by splitting from a single set.

Important findings from the exploratory analysis are as follows:

- The test and train datasets have 291 distinct features. In addition, train dataset has the label "price_doc", which is the sale price.
- Among the 272 features, there are 14 categorical variables and one "timestamp" which indicates the date of transaction.
- Most of the categorical variables (12 variables) are Boolean while other are enum values.
- A lot of features have missing data, they are handled in the data preprocessing stages of each algorithm.
- Target data, "price_doc" has is a few very high values, they have to be handled as outliers and also the data is right skewed, taking a logarithmic transformation normalizes the data. The outliers in the target variable are clipped.
- The features can be classified as:
    1. Housing Characteristics
    2. Demographic Characteristics
    3. School Characteristics
    4. Cultural/Recreational Characteristics
    5. Infrastructure Features
- A significant number of features of highly correlated, this leads to multicollinearity. This can affect the performance of the regression models.
- The "prices_doc" has constantly increased over the years while it fluctuates monthly.
- The timestamp feature is used to create new variables such as "month year", these help in making use of the timestamp feature in a more meaningful manner.
- Based on the suggestions in the competition forums, anomalies in different features are cleaned and data is made fit for further preprocessing.
- The train and test data distribution can be easily predicted Random Forest algorithm, indicating they are derived from different sources. This might mot case problems for this project but will have to use good cross-validation methods in order to get good results.

Exploratory Visualization

In this section, few important features visualization from the Sberbank dataset are presented. **Figure 1** show all the variables that have missing data in them. The percentage missing data is as high as 47%, which is a very high value. Also, a lot of features have data points missing, this emphasizes the missing data handling process.

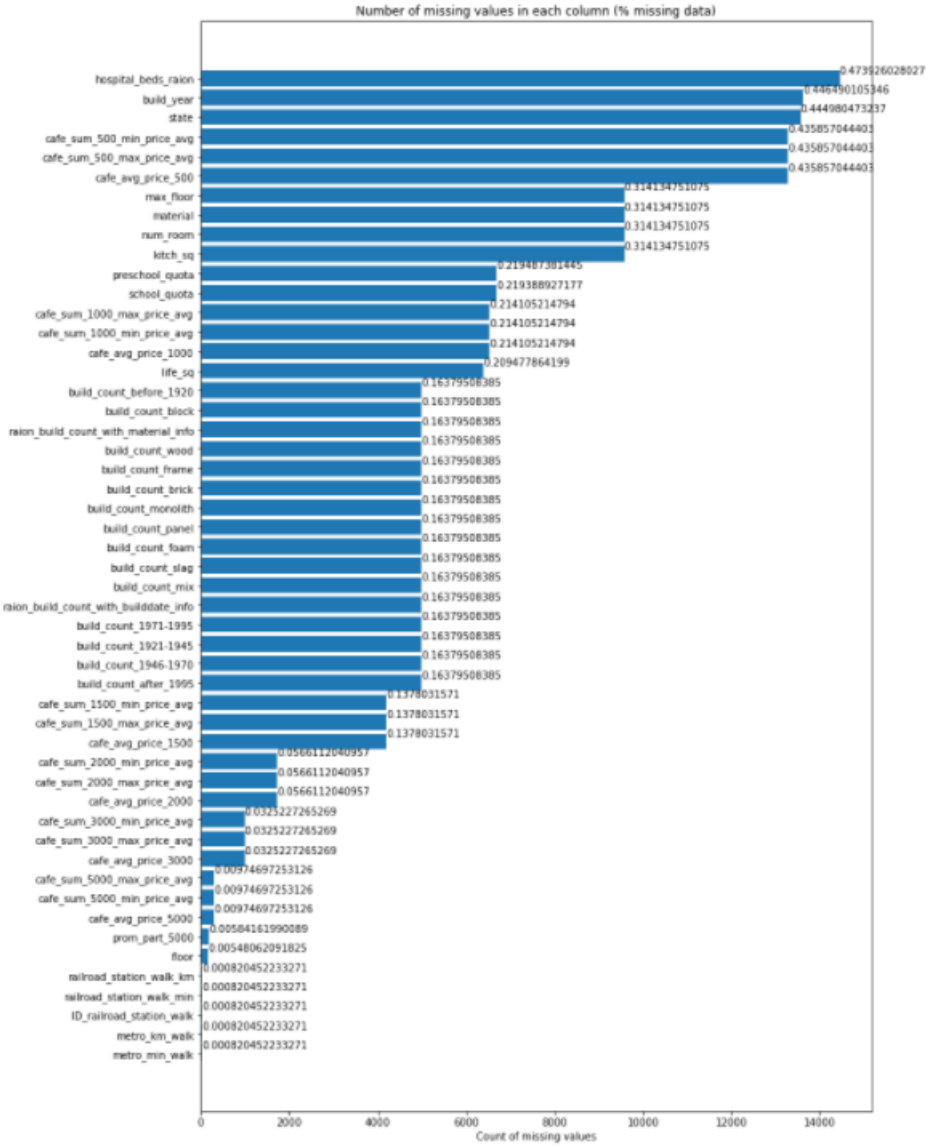
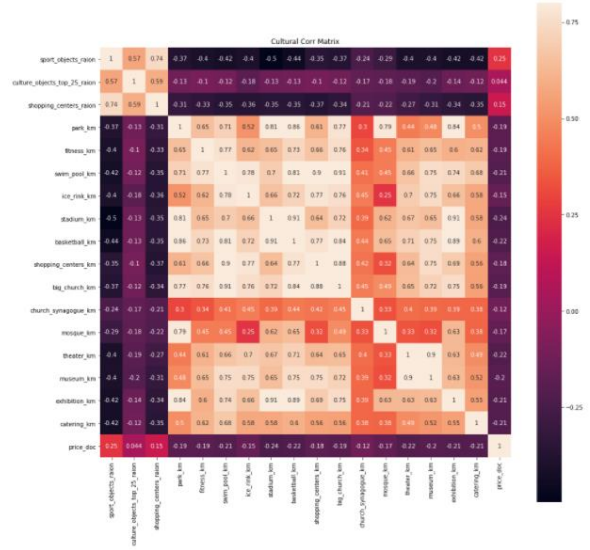
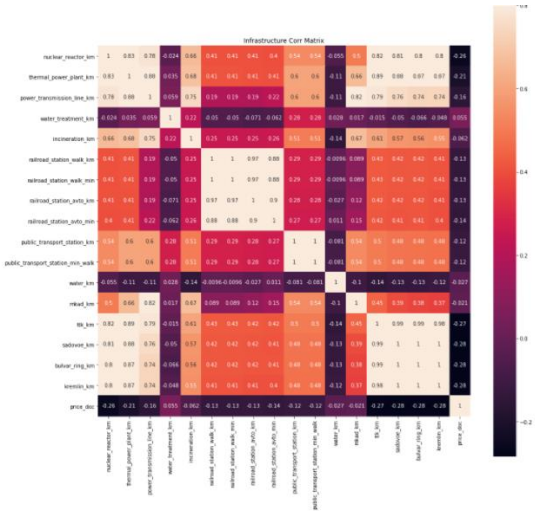
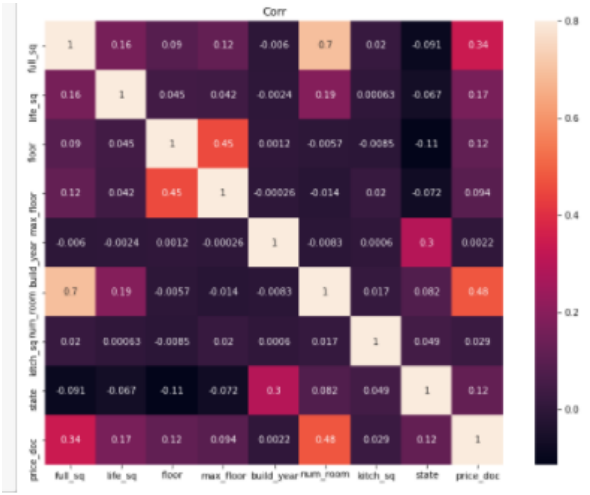
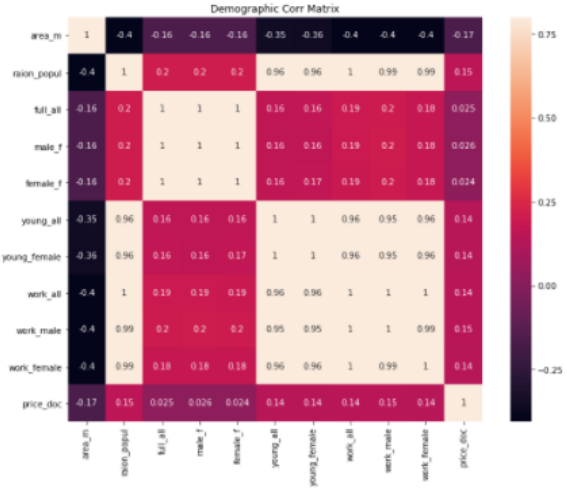


Figure1: Percentage of missing data









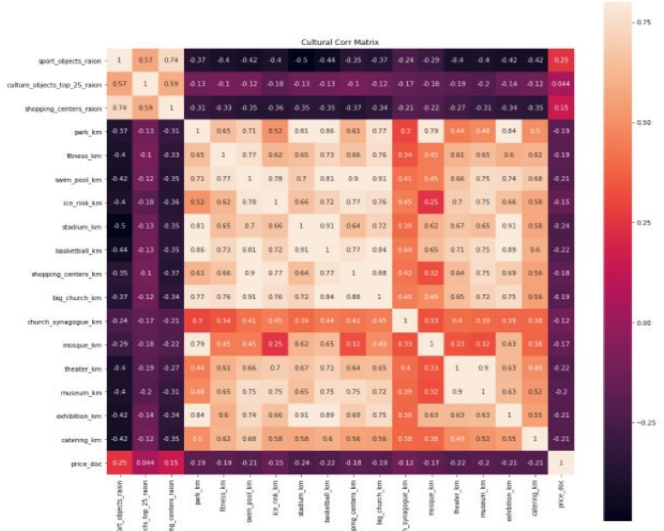


Figure 2: Correlation Matrix of different features

Figure 2 has five covariance matrices, each of them represent set of features. It can be seen that the many features are highly correlated (light colored regions). Among the five categories, the **Corr** plot which has basic house features like total area, year built, wall material number of rooms etc., has the least correlation.

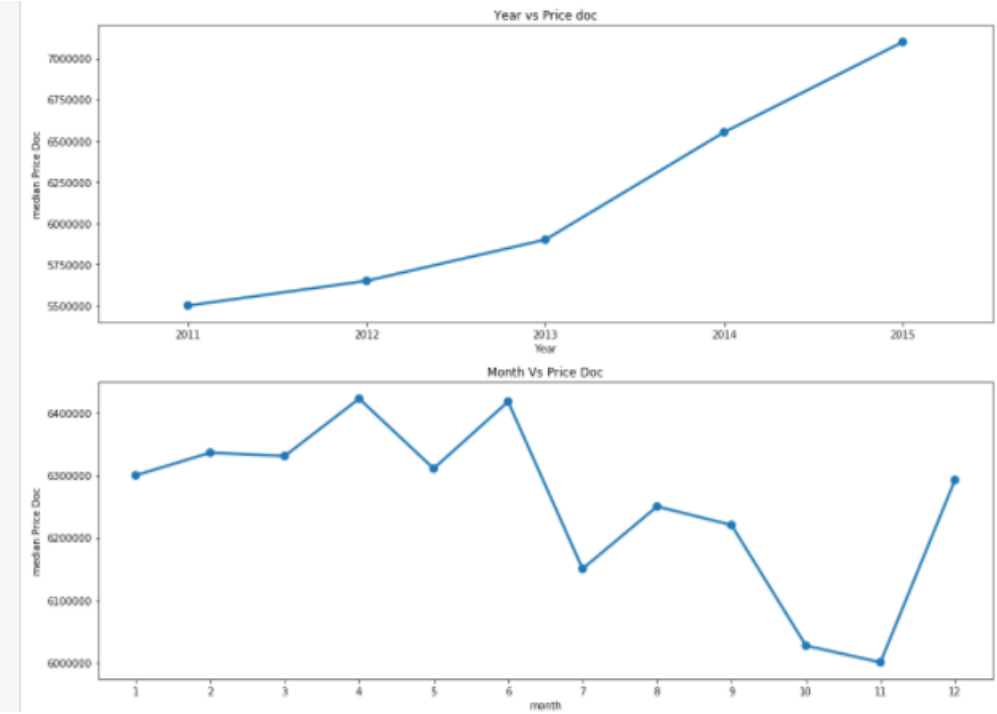


Figure 3: Target variable (**price_doc**) yearly (top) monthly (bottom)

Figure 3 gives an overview of price variations along the years and months as well. This plot clearly shows although the monthly prices, the real estate price has increased over the years without any hiccups.

Algorithms and Techniques

In this section, all the algorithms used is discussed briefly. The more detailed implementation of each algorithm can be seen in their respective notebooks dedicated for each algorithms (**XGBoost and NN_Model**). The choice of algorithms and techniques used are largely inspired by Kaggle methods of tackling competitions.

XGBoost:
The first algorithm of choice is XGBoost, this is a widely accepted and proved method most of the Kaggle competitions. It has a rich history of been successful in many recent competitions, this one of the major motivations to use XGBoost as the first algorithm.

Boosting is a sequential technique which works on the principle of ensemble. It combines a set of weak learners and delivers improved prediction accuracy. At any instant t, the model outcomes are weighed based on the outcomes of previous instant t-1. The outcomes predicted correctly are given a lower weight and the ones miss-classified are weighted higher. This technique is followed for a classification problem while a similar technique is used for regression.

Extreme Gradient Boosting or XGBoost is a variant of tree boosting method with deep considerations of system optimizations and fundamental principles machine learning. The main advantages of XGBoost are:

- Regularization: In machine learning, it is easy to build an algorithm that easily overfits the model. In XGBoost there is a more regularized model formalization to control overfitting. The parameters that aid this include: *max_depth(maximum depth of the tree), min_child_weight(the minimum sum of weights of all observations required in a child), gamma (minimum loss reduction required for a split), alpha(L1 regularization), lambda(L2 regualrization)*
- Parallel Processing: Compared to most of the boosting algorithms, XGBoost is exceedingly fast as it implements parallel processing.
- Handling Missing Data: XGBoost has a built-in routine to handle missing values. Upon encountering a missing value, the algorithm ties different methods and learns which path to take for a missing value in future.
- Built-in Crossvalidation: XGBoost allows user to run a cross-validation at each iteration of the boosting process and thus it is easy to get the exact optimum number of boosting iterations in a single run.

Multilayer Perceptron:
A multilayer perceptron is a network simple neuron called neurons. A perceptron uses computes a single output from a multiple inputs by forming a linear combination according to input weights and using a non-linear activation function. A single perceptron is not very useful because of its limited mapping ability. The perceptions can, however, be used as building blocks of a larger, much more practical structure. A typical multilayer perceptron (MLP) network consists of a set of source nodes forming the input layer, one or more hidden layers of computation nodes, and an output layer of nodes. The input signal propagates through the network layer-by-layer.

On the downside, increasing layers build a good model can easily result in overfitting. Thus, to reduce overfitting regularization techniques such as: *droupout(randomly drop units during training so that the doesn't adapt too much), batch-noramlization(a method to reduce internal covariate shift in neural networks) early stopping(stopping the training when validation error increases)*

In this project, Keras a high-level deep learning library built on top of Google's Tensorflow is used for easier and fast implementation.

GridSearch:
Gridsearch is a simple hyperparameter optimization method where, exhaustive searching/ sweeping of manually specified subset of hyperparameter space is learning algorithm is performed. The performance of a gridsearch technique is mostly guided by the cross validation score of the training set.

In this project, gridsearch method is derived from the scikit-learn library and 3-fold cross validation method is used to find the best set of hyperprameters for both XGBoost and MLP algorithms.

Stacking:
Stacking is an ensemble learning technique that combines multiple classification or regression models via a meta-classifier or a meta-regressor. The base level models are trained based on a complete training set, then the meta-model is trained on the outputs of the base level model as features.

In, this project two models, XGBoost and MLP, are stacked using simple linear regression method. The fitted meta-model is used to make the final predictions.

Benchmark

Kaggle leaderboard proves as a good benchmark for the problem. In fact, the competition has a predefined benchmark score as well (which can also be seen on the leaderboard). For this problem, training mean benchmark is a RMSLE value of 0.53347. This was bettered by most of the competitors. Also, from this project perspective as mention before, the evaluation metric used is Root Means Squared Error (RMSE) instead of RMSLE. As a consequence, the benchmark score would be different but this score is validated by submitting the predictions to the competition.

A simple XGBoost model with a default parameters gave a RMSE score of 2339300.07. Also, a basic MLP model had a score very close to 7404583.86.

In this project the aim would be design a model to achieve an error 200000, with would definitely result in a very good position on the Kaggle leaderboard as well.

III. Methodology

Data Preprocessing

The data preprocessing in this project is handled in two different stages. Firstly, the data is cleaned from the outliers or anomalies and are assigned as NaN. The main intension behind converting the outlier's points as NaN is that these would separately in the respective data preprocessing part of each algorithms. In this project, XGBoost handles the NaN internally and in MLP model they are manipulated for each feature using the following formula *min_value-max_value*2.*

The most the data cleaning is done in DataClean and **FeatureEngg** notebooks. In the notebook, major feature modifications are inspired from the discussions in the Kaggle forums, and they are:

- Addition of BAD_ADDRESS_FIX for GIS-related fixes as recommended
- Corrections rules for Full_Sq, Life_Sq and Kitchen_Sq
- Correction rules for Number of Rooms
- Fixing Floor and Max Floor; Material and State features
- Removal of extreme values in target variables
- Adding new features based on timestamp
- Handling correlated features by dropping highly correlated variables

Target Variable Processing

From the exploratory analysis, it was seen that the target variable, "price_doc" is right-skewed. To normalize the feature, logarithmic transformation servers the purpose. This is seen in **Figure 4.**

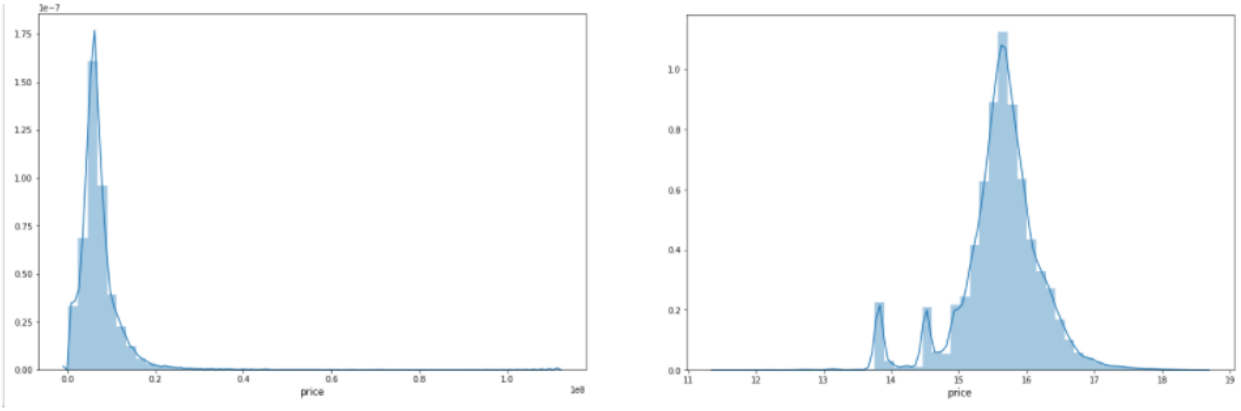


Figure 4: Raw and Log transformed target feature

In this project, the raw target feature gave better score in comparison with the log transformed. So in the project target feature is used as it is.

Processing Categorical Variables:

In majority of the datasets, there are two types of data types: categorical and continuous. Most of the machine learning algorithms, especially regression based methods cannot process categorical variables. In this project, like most of the datasets there are categorical variables and this are handled by standard strategies such as one-hot-encoding or label-encoding.

One-hot-encoding: For every categorical variable, a new binary variable is added for each unique value. The output is a sparse. The output will be a sparse matrix where each column corresponds to one possible value of one feature. This is widely used method since in this transformation no ordinal relationships exist but this method leads to explosion in the feature size. This curse of dimensionality leads to large training time.

Label-Encoding: This is simple method where each unique category value is assigned an integer value. This is not preferred method the model might assume a natural ordering between categories may result in poor performance or unexpected results.

In this project, as XGBoost can handle label-encoding efficiently, the categorical variables are converted to integer equivalent values. Neural networks do not perform well with label encoded features, but this method is a very slow algorithm so categorical variables should be handled carefully. For this reason, the binary feature are transformed using one-hot-encoding and other categorical variables are transformed using label encoder.

Implementation

In this capstone project, the implementation is done in two stages

- Stage 1: Build, train and tune base models (Level or L0):
    - XGBoost model
    - Neural Network model
- Stage 2: Build and validate a stacker model which gives results better than any of the L0 models

Level 0 Models: Building, training and tuning

This section can be further classified into XGBoost and MLP models

XGBoost Model
The strategy used in this building and tuning XBoost was inspired from Analytics Vidhya

1. Build a simple model with *max_depth = 5* and *num_boost_round = 50*. This acts a benchmark model with a baseline score of 2339300.07. All the further tuning is done to improve the model
2. In order to tweak the hyper parameters we use the scikit-learn wrapping for XGBoost is used. This XBGRegressor enables easy integration with the scikit-learn GridSearchCV for parameter tuning and cross-validation.
3. Important hyper-parameters, learning rate: 0.1, number of estimators: 50, evaluation metric: RMSE are fixed for further gridsearch tuning.

4. In the first step, max_depth and min_child_weight are tuned. The reason behind choosing max_depth and min_child_weight is that the former increased the model complexity while the latter acts a regularization parameter. On tuning these the best values were max_depth': 8, 'min_child_weight': 6 and the score improved from 2339300.07 to 2315526.983
5. Next parameter is the regularization parameter gamma. In this tuning, the scores remained unchanged since the model is over fitted. Thus the best score is unchanged ('gamma': 0.0) 2315526.9834.
6. Tuning fraction of features and observations to be randomly samples for each tree i.e., colsample_bytree and subsample we get the best parameters to be 'colsample_bytree': 0.60, 'subsample': 0.9. The score improved from 2315526.983 to 2294570.392.
7. The final tuning involves selecting the number of weak learners or estimators in such a way that the whole model performs better. On tuning, best results were obtained for n_estimators=100 and the score improved from 2294570.392 to 2286901.815

The results from the XGBoost tuning process are represented below:

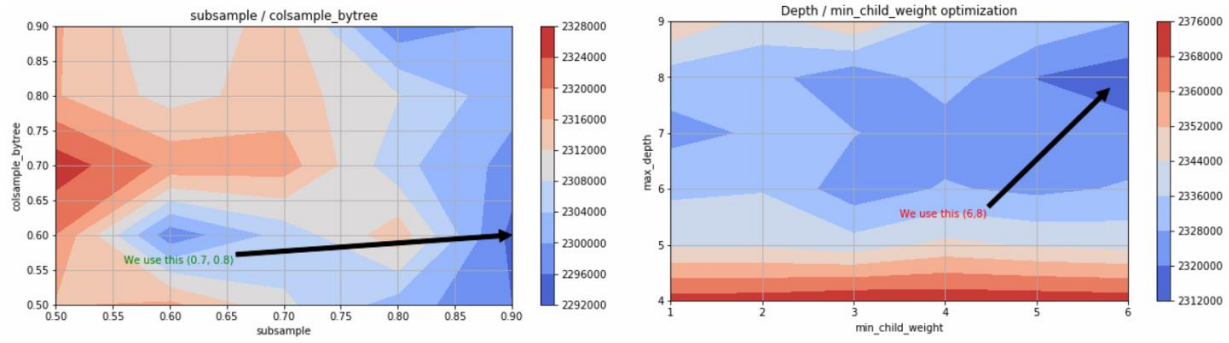


Figure 5: hyperparameter tuning for XGBoost model

Neural Network model

In this section a brief rundown of building and tuning of a neural network model is discussed.

1. Build a simple 2 layer neural network model with one hidden layer with one fully conncted layer, ReLU activation with adam optimizer. This model acts as the baseline model for the further neural network training. The base model has a score of 7404583.86.
2. As mentioned before, the MLP model is built upon Keras library and GridSearchCV along with KerasRegressor is used to tune the hyper-parameters.
3. By tuning batch size and units in hidden layers, the best fit was batch_size: 32 and 128 units of fully connected layer. Although by tuning the parameters model appears improves, but there are very high chances of overfitting. The improved score is 5761817.96.
4. In order to tackle the problem of overfitting, regularization techniques such as dropout and batch normalization are added.
5. Tuning different parameters such as number of epochs, batch size, weight initialization method, and finally dropout rate we end with a 3 layer layer with the following parameters: epochs': 100, 'batch_size': 32, init_mode': 'normal, 'dropout_rate': 0.1

6. The final model has CV error of 2639784.47 which is huge improvement from the baseline score.

Level 1 Model: Building, training and tuning

The best models from the L0 training (XGBoost and MLP) are stacked. For training L1 models, the dataset of out-of-fold predictions and test set is used to for the final assessment of ensemble model. The methodology is as follows:

1. **Splitting the dataset**: Split the train dataset into k folds
2. **Out-of-fold predictions**: Fit first stage models(Level 0 or L0) on k-1 folds and predict the kth fold
3. **Full set prediction**: Fit first stage models on the whole train set and get the predictions for the test dataset. The predictions from each model are combined into a test set where each feature is respective L0 model predicitons
4. **Stacker model**: Using the predictions from the out-of-fold set and labels of the training set, fit a stacker model (L1 model). Finally, the stacker model is used to predict test set labels based on the on the combined testset of L0 model.

For fitting the L0 models we use simple model such as LinearRegressor as the L1 model to reduce the chances of overfitting. The final stacker gives a result of 2236490.76654.

The stacker model is a linear combination of:

*Final_Stacker = 0.80*XGB_Pred + 0.23*MLP_Pred*

IV. Results

Model Evaluation and Validation

The final stacker model is a linear combination of the XGB and MLP models. This model performance is the best compared to individual models. In order to validate the model robustness, the final ensemble was trained and validated (stacked) various train subsets.

Five different subsets were chosen and all the models (L0 and L1) were trained. The final score of the training are aggregate in to a dataframe and the various metrics of the training error are shown below.

Metric	MLP	XGB	Stacker
Count	5.000000e+00	5.000000e+00	5.000000e+00
Mean	2.526834e+06	2.218198e+06	2.210825e+06
Std	5.695274e+04	3.209167e+04	3.431699e+04

Metric	MLP	XGB	Stacker
Min	2.466971e+06	2.178806e+06	2.171958e+06
25%	2.479117e+06	2.202010e+06	2.193288e+06
50%	2.517315e+06	2.207972e+06	2.196563e+06
75%	2.576244e+06	2.246863e+06	2.236356e+06
Max	2.594524e+06	2.255339e+06	2.255961e+06

All models has low standard deviation that the models have generalized well and less sensitive to changes. Also, the Stacker model has the least error compared to the best L0 model (XGB).

Justification

The baseline score set by the competition was 0.53347, in order to compare with the benchmark score, predictions on the test-set from the individual models (both L0 and L1) were submitted and the results were as follows:

Model	RMSLE
Benchmark Model	0.53347
Best MLP Model	0.39362
Best XGB Model	0.32665
Best Stacker Model	0.32621

The Stacker model has an improved performance when compared with the baseline Radomforest model. There is a significant improvement of 39% compared to the benchmark model.

V. Conclusion

Free-Form Visualization

In this section, a bird's eye view of the Capstone project is provided. After preprocessing the data two algorithms that were implemented and trained are XGBoost and Multi-Layer Perceptron. Initially, these models were immature for the project, in order to get the right settings, a series of steps were followed to tune the hyper parameters were tuned. After the building the base models of best fit, out-of-fold predictions were used to fit L1 model (Linear Regression) and base model predictions were combined using stacking technique. In order to validate that the final model has generalized well, the models was checked for consistency by training on 5 different seeds. The final score of L1 model was most optimal score compared to the individual L0 models.

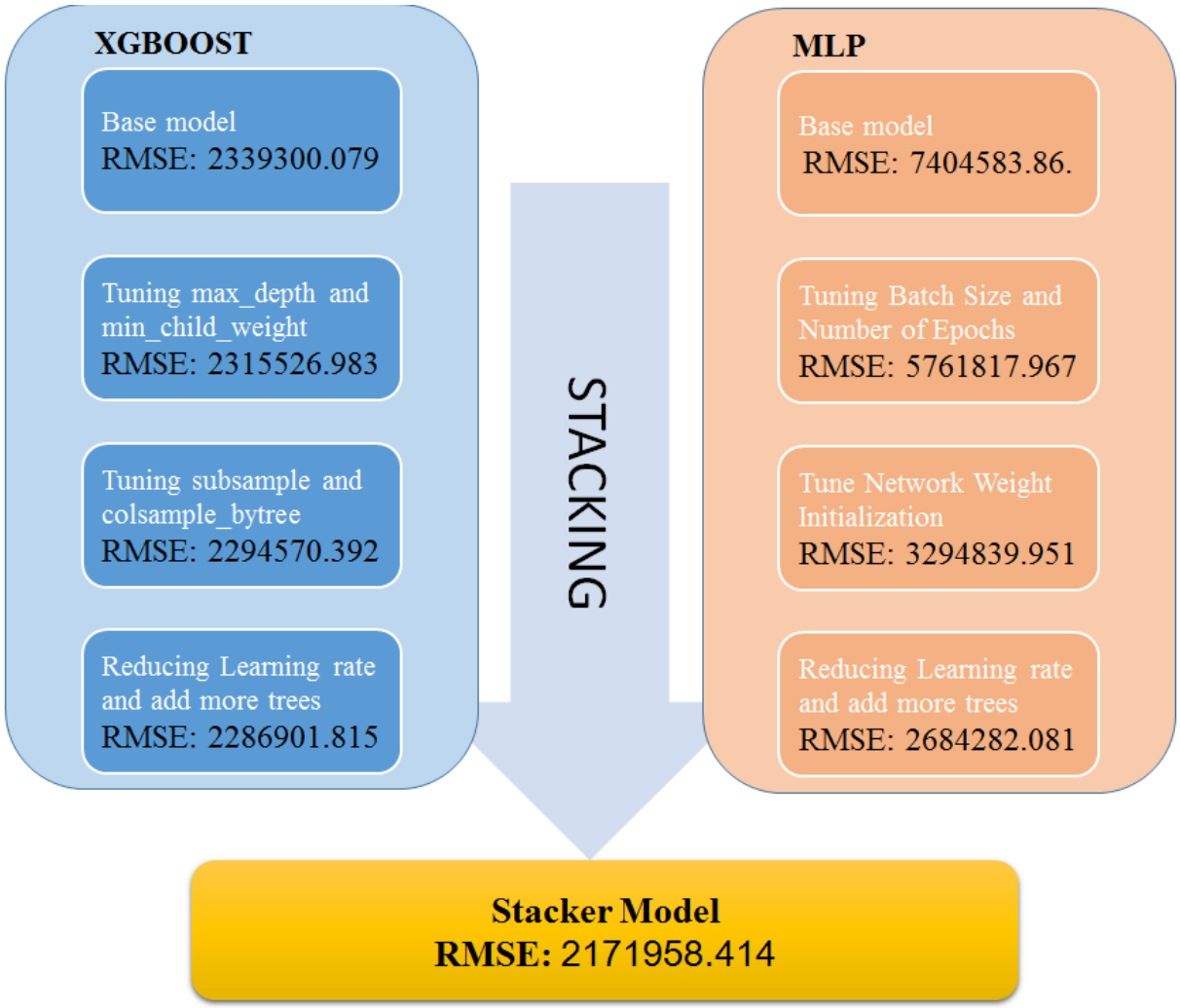


Figure 6: Free-Form visualization

Reflection

End-to-end problem solution

The main scope of the project is to work on a real-world problem dataset. Showcase the importance of data pre-processing, algorithm and their optimization. Another intention of choosing the Kaggle competition is to build a good XGBoost and Neural Network model and evaluate personal performance with the other competitors.

Interesting Aspects:

The working experience with such a feature rich real estate data itself is an exciting task and this was one of the major motivation for selection of the project. Producing models with significant

improvement over the benchmark score without using “magic numbers” or extensive feature engineering indeed is a good accomplishment.

Challenges

In this entire project I refrained myself from using extensive data enrichment methods or use of magic numbers to tweak the Leader Board. I also kept myself away from using public kernels which could have easily boosted my performance by a good margin. Finally the computation complexity that the competition expects is clearly out of scope of this capstone project.

Another serious challenge posed by the project is the requirement of a good computational infrastructure. This capstone project was accomplished with a Quadro 2000M GPU and 32 GB of memory. There are several parts that demanded heavy computation such as:

- GridSearch: Evaluating the hyper parameter space of the two algorithms XGBoost and MLP model took a lot of time. Neural Network Models notebook took as high as 6 hours to iterate through various hyperparamets.
- Cross validation and out-of-fold predictions for L0 and L1 models took substantial amount of time to validate the robustness.

In this section, you will summarize the entire end-to-end problem solution and discuss one or two particular aspects of the project you found interesting or difficult. You are expected to reflect on the project as a whole to show that you have a firm understanding of the entire process employed in your work. Questions to ask yourself when writing this section:

- *Have you thoroughly summarized the entire process you used for this project?*
- *Were there any interesting aspects of the project?*
- *Were there any difficult aspects of the project?*
- *Does the final model and solution fit your expectations for the problem, and should it be used in a general setting to solve these types of problems?*

Improvement

From this project perspective, there are many sections that can be improved. It could be from data preprocessing to the final ensemble method. Few important ones include:

- In the data preprocessing section: the competition organizers, Sberbank, have shared another set of features called “macros”. This includes various features such as “GDP”, “Inflation”, “Salary” many more. Using these additional features might improve the results.
- The XGBoost model could have been more complex by adding more estimators and using early stopping to prevent it from overfitting.
- In the Neural Network section, deeper architectures could have been tested and also hyper parameters could have been tuned as specific each hidden layer.
- Higher number of folds, say 10 folds could’ve been used while cross-validating the train datasets.

- With respect to the final stage, only two L0 models were used instead, more t models could such as LightGBM, SVM could have been used to combine the predictions. In addition, more L1 models could be used to fit a new L2 model which is quite common in Kaggle world.