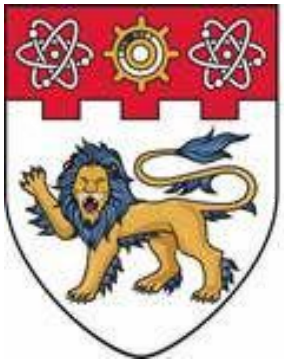

NANYANG TECHNOLOGICAL UNIVERSITY
SINGAPORE

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

CZ4041 - MACHINE LEARNING

PROJECT REPORT

Sberbank Russian Housing Market Prediction

STUDENT NAME	MATRICULATION NUMBER
Andre Lim	U2020397A
Andrew Ng	U2023820F
Chan Eu Ching	U2021000F
Chua Yu Hao	U2022690F
Keith Lee Mun Wei	U2023321A

1. Roles and Contributions **4**
2. Kaggle evaluation score and rank **4**
3. Problem Statement **5**
4. Challenges of the problem **5**
5. Proposed solution **6**
5.1. Preprocessing 6
5.1.1. Data Cleaning 6
5.1.2. Feature Selection 6
5.1.3. Further Cleaning 7
5.2. XGBoost [1] 8
5.3. Light Gradient Boosting Machine (Light GBM) [2] 9
6. Experiments **10**
6.1. Hyperparameters Tuning 10
6.2. Identifying Top K Features 11
6.3. Testing of Models 13
6.4. Final Parameters 13
7. Further Improvements **14**
7.1. Preprocessing 14
7.1.1 Data Cleaning 14
7.1.2 Feature Engineering 14
7.2 LightGBM 15
8. Weighted Model Averaging **15**
9. Conclusion **15**
References **16**

1. Roles and Contributions

STUDENT NAME	MATRICULATION NUMBER	CONTRIBUTION
Andre Lim	U2020397A	Everything
Andrew Ng	U2023820F	Everything
Chan Eu Ching	U2021000F	Everything
Chua Yu Hao	U2022690F	Everything
Keith Lee Mun Wei	U2023321A	Everything

2. Kaggle evaluation score and rank

Our Light GBM model obtained a final score and rank of **0.30787** and **12** respectively.

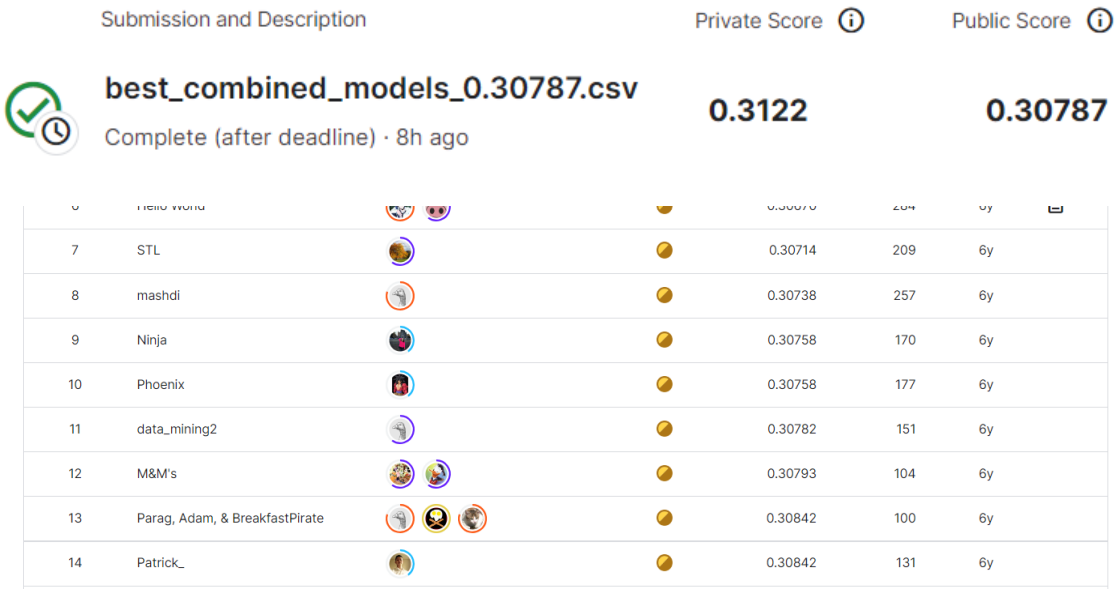

Figure 1. Results of our model

3. Problem Statement

Housing costs represent a substantial financial commitment for both individuals and developers. Whether managing personal finances or corporate budgets, uncertainty surrounding significant expenditures is often an unwelcome challenge. The problem at hand centres around the need to address this uncertainty in the Russian Housing Market by using machine learning.

In this competition, we are tasked to develop algorithms which use a broad spectrum of features to predict realty prices. An accurate forecasting model will allow Sberbank to provide more certainty to their customers in an uncertain economy.

The evaluation metric for this competition is RMSLE (Root Mean Squared Log Error):

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

Where:

n is the total number of observations in the (public/private) data set,

p_i is your prediction of target, and

a_i is the actual target for i.

$log(x)$ is the natural logarithm of x ($log_e(x)$.

4. Challenges of the problem

1. Given the context of this regression problem and complex dataset, we had to identify a model that is best suited for us to make accurate predictions on property prices.
2. There was erroneous data present in the dataset. For example in ‘train.csv’, the ‘build year’ column has values such as ‘4965’ and ‘20052009’
3. Depending on the model that we choose, we might need to encode categorical values into numerical values to allow the dataset to be trained on the model.
4. There were many missing values in the dataset (51 columns and more than 24000 rows had missing values). One of the challenges is to figure out how we can deal with these NaN values.
5. The given train and test dataset is large with a total of 290 columns (before encoding). This presents the difficulty of identifying the important and relevant columns for training the model.

5. Proposed solution

We attempted to solve the challenges identified previously, through the different sections of our proposed solution.

5.1. Preprocessing

5.1.1. Data Cleaning

Firstly, we analysed the data to have a better understanding of what needs to be done for data cleaning. We discovered that there was some erroneous data (challenge 2). Since there were only 2 rows that contained the erroneous data, we decided to remove these 2 rows so that it does not affect our model.

In order to tackle the categorical features, a plethora of encoding methods were trialed and tested (challenge 3). These encoding methods include Ordinal, One-hot and Target encoding. Ordinal encoding involves encoding features such as ‘ecology’ into ordered, numerical values (1, 2, 3 etc) representing the different conditions. Next, One-hot encoding was applied to ‘product_type’ because the values were simply labels, with no order to it. Finally, target encoding was used on ‘sub-area’, whereby the encoded categorical values would be represented by the average ‘price_doc’ for that specific ‘sub_area’. Overall, different permutations of these encoders were tested, such as using all three, only using One-hot or Ordinal encoders. Ultimately, after testing the performance of each of these, we decided on only using One-hot encoding because it produced the lowest error value. Additionally, we also tested on dropping rows containing NaN for categorical columns;however, the model performed better when such rows were not dropped.

5.1.2. Feature Selection

Now, with the data cleaned up and processed, we had to identify the columns with greater importance (challenge 5). For this, SHAP was used. SHAP values are values of importance assigned to each feature in a model. These values indicate how an attribute/feature will affect the choices the model makes/predicts with the usage of Game Theory. In order to get a general sense on the feature importance, XGBoost was used. This is due to its efficient and good performance on complex datasets (challenge 1). With the help of the SHAP library’s Tree Explainer class, the XGBoost model was explained and analysed, helping to identify the key features.

We arbitrarily extracted the top 50 features for our model.

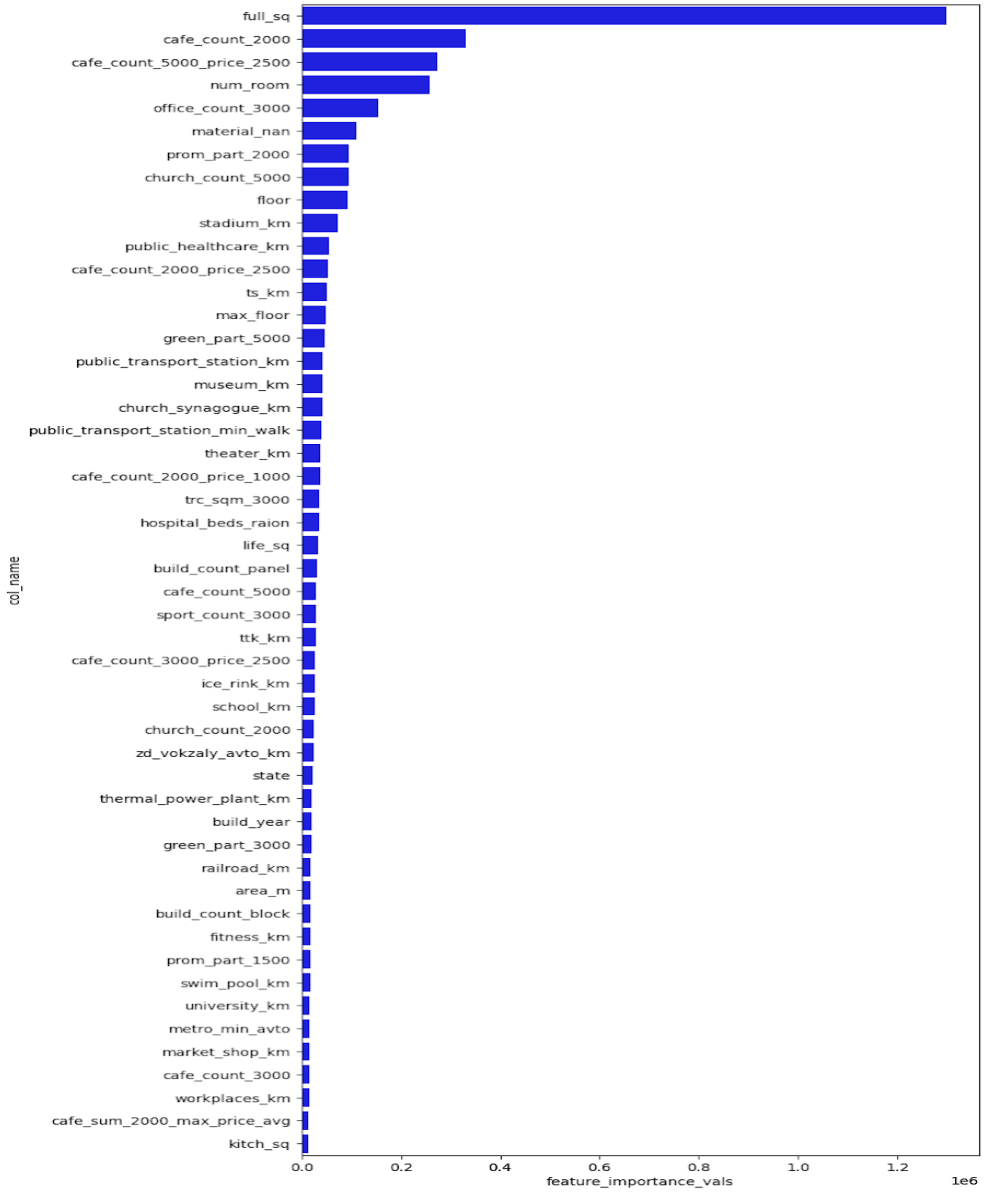

Figure 2: Top 50 features for XGBoost

5.1.3. Further Cleaning

At first, we generally filled in the missing values (challenge 4) of each column with the mean value for that column. After identifying the potential top features, a more granular approach was taken to fill in the missing values for the columns of higher importance extracted by the SHAP library.

Some of these specific methods include:

1) For ‘max_floor’, ‘build_year’ and ‘state’, we fill in the missing values with the median value of each column respectively, grouped by the properties in the same sub_area. For example, if there is a row with missing value in the ‘max_floor’ column belonging to the ‘Bibirevo’ sub_area, we will fill it in using the median ‘max_floor’ value for properties belonging to the ‘Bibirevo’ sub_area.

2) For ‘floor’, we simply fill in the missing values with the ‘max_floor’ value of each row respectively.
3) For ‘num_room’, we split the ‘full_sq’ values into different ranges (0-30, 31-52, 53,80, >80) and calculated the average num_room values for each range. Then, we filled in the rows with missing ‘num_room’ values using the calculated average num_room value depending on which range of ‘full_sq’ values the row belongs to.
4) For ‘kitch_sq’, we calculated the average ‘kitch_sq’/‘life_sq’ proportion grouped by sub_area and filled in the rows with missing ‘kitch_sq’ values using the ‘life_sq’ value of the rows multiplied by the kitch_sq/life_sq proportion depending on which sub_area the row belongs to.

5.2. XGBoost [1]

We chose XGBoost as one of our models due to the Gradient boosting algorithm’s capacity to handle outliers effectively. This algorithm exhibits robustness to outlier data points, ensuring that the presence of extreme values in the dataset does not unduly influence the model's predictions. This is crucial since our dataset does contain many outliers, but we are unable to simply drop the rows containing them since it would greatly reduce the dimension of our dataset, hence affecting our model’s training.

Furthermore, its efficiency is a noteworthy advantage, allowing for the streamlined processing of large and intricate datasets that might otherwise prove challenging for other machine learning models. This efficiency is vital in dealing with the complex nature of our large dataset. For example, Parallel Tree boosting, a feature of this algorithm, has the ability to train multiple decision trees in parallel, significantly reducing computational time, making it a pragmatic choice for handling our data.
In terms of performance, the ensemble learning approach employed by this algorithm combines results from multiple individual trees, translating to accelerated learning and enhanced model accuracy. This boosts the algorithm's predictive power, a crucial factor for our dataset.

Comparatively, the Gradient boosting algorithm demonstrates a favorable balance between Boosting and Bagging techniques when contrasted with a simple Random Forest model. These techniques adopt an additive approach, where future trees are built upon the results of their predecessors. This leads to a more efficient learning and prediction.

Lastly, the algorithm's user-friendliness is a notable benefit. Its ease of use simplifies the modeling process and allows for a more straightforward integration into our data analysis workflow.

5.3. Light Gradient Boosting Machine (Light GBM) [2]

Light Gradient Boosting Machine (Light GBM) is an efficient, powerful and flexible gradient boosting framework that uses decision tree algorithms to learn and make decisions. It can be used for ranking, classifications and many other machine-learning tasks.

One significant difference between Light GBM and other boosting machines is it splits the tree leaf-wise with best fit while other boosting algorithms split the tree depth-wise or level-wise.

The utilization of two novel techniques Gradient-Based One-Side Sampling (GOSS) as well as Exclusive Feature Bundling (EFB) allows the model to have a faster execution rate while maintaining good accuracy levels. GOSS maintains the accuracy of data by keeping instances with larger gradients and executes random sampling on instances with smaller gradients.

EFB is a method to decrease the number of effective features that is near lossless, hence allowing the model to incur lower loss as compared to level-wise algorithm, producing a more accurate result that could hardly be achieved by other existing boosting algorithms.

Not only that, it is also very efficient hence gaining the name of 'Light', and Figure 5.4a and Figure 5.4b will illustrate the reason for efficiency.

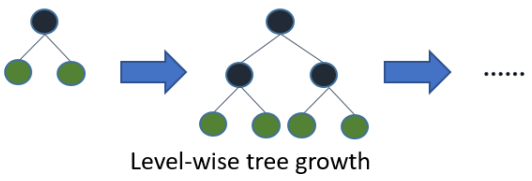

Figure 3a: XGBOOST

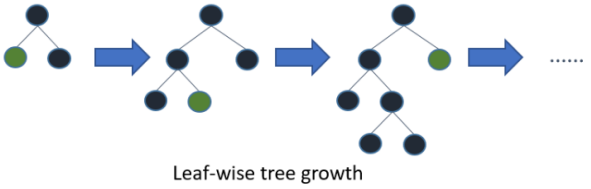

Figure 3b: LightGBM

Even though leaf-wise splits increase complexity, making it prone to overfitting, it can be solved by indicating a parameter max-depth which specifies the depth at which splitting occurs, preventing the tree from growing too large and complex hence improving the generalization performance of the model.
Not only that, Light GBM can perform well with large datasets, and compared to XGBoost, it has a more efficient training time.

6. Experiments

6.1. Hyperparameters Tuning

With the models decided, we carried out hyperparameters tuning to evaluate the optimal set of parameters for each model. However, with a wide range of potential parameters to tune, it was inefficient to do it with only a simple grid-search. Table 1 shows some of the relevant hyperparameters we have tuned specific to our models [2, 3].

Parameter	Definition
num_leaves	Max number of leaves in a tree. Affects accuracy and the fitting of the tree (overfit, underfit etc).
min_data_in_leaf	Minimum number of data in a leaf. Affects fitting of the tree (overfit, underfit, etc).
max_depth	Max depth of the tree. Increase in depth results in more complex relationships in data to be captured. This affects the fitting of the tree.
bagging_fraction	Fraction of the data used in each iteration. A higher fraction results in longer training.
feature_fraction	Fraction of features used in each iteration. A higher fraction results in longer training.
max_bin	Max number of bins that feature values will be bucketed in. Smaller bins may reduce training accuracy but reduce overfitting

Table 1: Definition of hyperparameters

Therefore, we conducted the search via the *Optuna* library. Optuna is an open-source library that enables efficient search of hyperparameters through cutting-edge algorithms [4]. Firstly, we indicate the parameters that require tuning for each model, and the respective value search space. This is shown in Figure 4. Next, different combinations of the search space will be trialed and tested, optimising based on criteria specified. The criteria used for evaluation of our models was the mean squared error. The combinations of parameters will be applied to the respective model, and fitted and tested against the Sberbank train dataset. Finally, the parameters giving the best results will be returned [6].

```python
f objective(trial):
  params = {
      "learning_rate": trial.suggest_float("learning_rate", 1e-3, 0.1, log=True),
      "n_estimators":  trial.suggest_int("n_estimators", 100, 500),
      "colsample_bytree": trial.suggest_float("colsample_bytree", 0.05, 1.0),
      "max_depth": trial.suggest_int("max_depth", 3, 10),
      "min_child_weight": trial.suggest_int('mind_child_weight', 0, 12),
      "gamma": trial.suggest_float("gamma", 0, 1),
  }

  regressor =  XGBRegressor(
                        device=xgb_params['device'],
                        objective=xgb_params['objective'],
                        eval_metric=xgb_params['eval_metric'],
                        enable_categorical=xgb_params['enable_categorical'],
                        early_stopping_rounds=xgb_params['early_stopping_rounds'],
                        n_jobs=xgb_params['n_jobs'],
                        **params
                    )

  regressor.fit(X_train, y_train, eval_set=[(X_val, y_val)])
  y_pred = regressor.predict(X_val)
  rmse = mean_squared_error(y_val, y_pred, squared=False)
  return rmse


udy = optuna.create_study(direction='minimize')
udy.optimize(objective, n_trials=20)
st_params = study.best_params
```

XGBoost

```python
from sklearn.metrics import mean_squared_error
import optuna

def objective(trial):
    params = {
        "objective": "regression",
        "metric": "rmse",
        "verbosity": -1,
        "n_estimators": trial.suggest_int("n_estimators", 600, 1000),
        "boosting": trial.suggest_categorical("boosting", ["gbdt", "rf", "dart"]),
        "lambda_l2": trial.suggest_float("lambda_l2", 0, 10),
        "bagging_fraction": trial.suggest_float("bagging_fraction", 0, 1),
        "bagging_freq": 1,
        "num_leaves": trial.suggest_int("num_leaves", 2, 2**10),
        "feature_fraction": trial.suggest_float("feature_fraction", 0.5, 1),
        "max_depth": trial.suggest_int("max_depth", 1, 50),
        "learning_rate": trial.suggest_float("learning_rate", 1e-3, 0.1, log=True),
        "subsample": trial.suggest_float("subsample", 0.05, 1.0),
        "colsample_bytree": trial.suggest_float("colsample_bytree", 0.05, 1.0),
        "min_data_in_leaf": trial.suggest_int("min_data_in_leaf", 10, 100),
#         "max_bin": trial.suggest_int("min_data_in_leaf", 128, 512),
    }

    model = lgb.LGBMRegressor(**params)
    model.fit(X_train, y_train, verbose=False)
    predictions = model.predict(X_val)
    rmse = mean_squared_error(y_val, predictions, squared=False)
    return rmse
```

LightGBM

Figure 4: Hyperparameters tuning with Optuna library

6.2. Identifying Top K Features

With the newly identified optimal hyperparameters, 5-fold cross-validation is used on the models to determine the top features based on average SHAP values (as mentioned in 5.1.2 and seen in Figure 5). The top 100 features are then iteratively searched upon, from a range of 10 to 100, with steps of 5. In each iteration, 5-fold cross validation is also used in order to get the mean squared loss. Figure 6 shows an example of the resulting scatter plot comparing the average loss and number of features used.

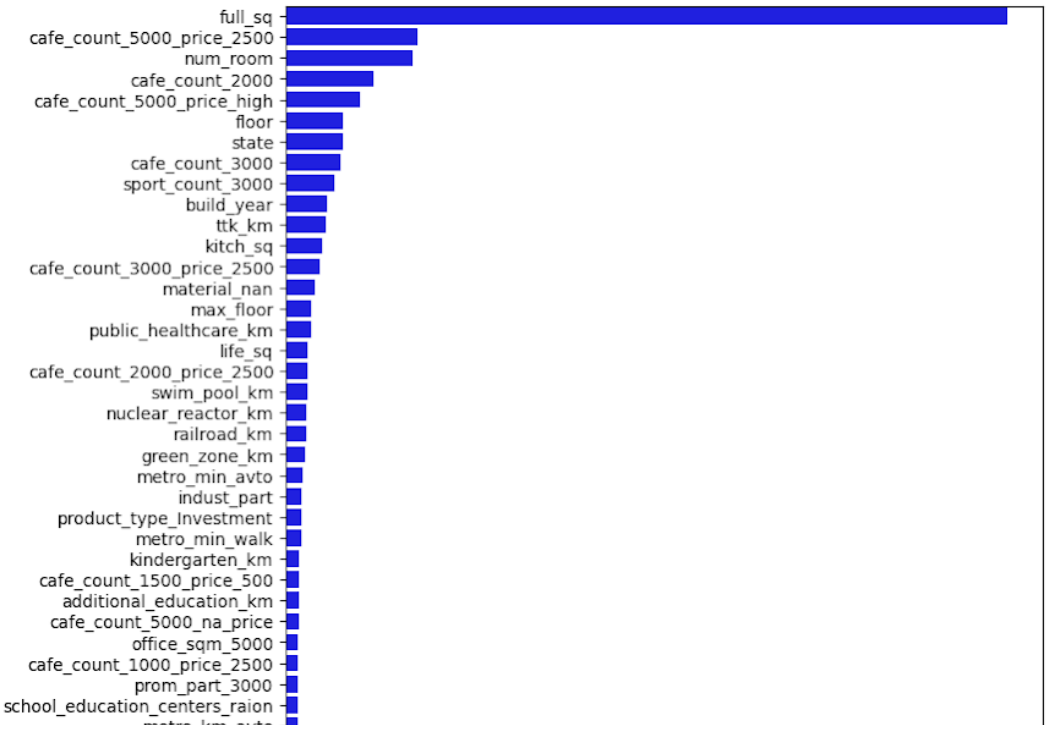

Figure 5: Snippet of top features

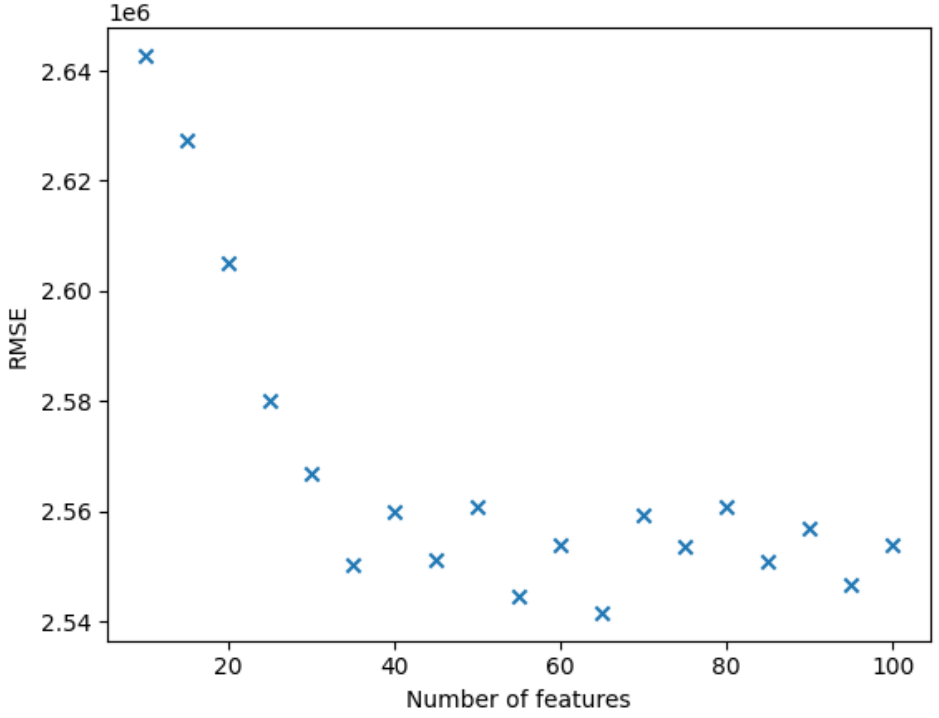

Figure 6: XGBoost RMSE against Number of features

6.3. Testing of Models

With both the current optimal hyperparameters and top features to use, the models are re-trained with a 10-fold cross-validation using the train dataset. The model with the median mean-squared-error was chosen. This is because through constant testing, we realised that a model with the lowest mean-squared-error need not necessarily result in the best prediction results when evaluated against the test dataset. This could be due to the model fitting more to the train set and not being generalised enough.

6.4. Final Parameters

The process from 6.1. to 6.3. was repeated in order to improve the model. Ultimately, Figure 7 shows the optimal hyperparameters after several iterations.

```
{'learning_rate': 0.04560121985309734,
 'n_estimators': 284,
 'colsample_bytree': 0.8872442400708895,
 'max_depth': 5,
 'mind_child_weight': 7,
 'gamma': 0.1795763847094346}
```

XGBoost

```
params4 = {
    'boosting': 'gbdt',
    'lambda_l2': 7.068023653932577,
    'bagging_fraction': 0.6261319161311689,
    'num_leaves': 151,
    'feature_fraction': 0.976804739696195,
    'max_depth': 8,
    'learning_rate': 0.01552248392804131,
    'subsample': 0.4408875142865268,
    'colsample_bytree': 0.3829262582959811,
    'min_data_in_leaf': 21,
    "objective": "regression",
    "metric": "rmse",
    "n_estimators": 825,
    "verbosity": -1,
}
```

LightGBM

Figure 7: Best result hyperparameters for respective models

7. Further Improvements

Following the submission of predictions from the aforementioned model, our team identified areas for improvement. To address these shortcomings, we delved into Kaggle discussions involving leaderboard competitors [7]. By examining various data cleaning strategies, a more refined approach was devised and implemented.

7.1. Preprocessing

7.1.1 Data Cleaning

It was observed that there were a few features that required further cleaning to remove outliers.

It was noticed that *full_sq, life_sq,* and *kitch_sq* required more cleaning as there were more erroneous entries. For instance, as explained in the data dictionary, we expect the *'full_sq'* (total square footage) not to surpass the *'life_sq'* (living area). Any data in both datasets that do not comply with this rule will result in *life_sq* being set as NaN.

Another feature that has erroneous data was built year. Built year is the year in which the house was built. Hence years that were built before 1500 were replaced as NaN in both test and training datasets.

Next, we clean the num_rooms data. For example, we set a minimum of 5 square meters for a room and all room areas below the minimum limit were set to NaN. All rooms with no rooms (*num_room* == 0) were also set to NaN. For any outliers, such as *life_sq* and *full_sq* being very small but *num_rooms* being large, we also set the data to NaN.

Finally, the data for *floor* and *max_floor* is cleaned. *Floor* or *max_floor* cannot be 0. Therefore, we changed these numbers to NaN. Since *floor* cannot be greater than the *max_floor*, we changed *max_floor* to NaN. Similarly, outliers were checked.

This data-cleaning process was designed to ensure that the data is accurate, consistent, and complete. This will improve the performance of any machine learning models that are trained on the data.

7.1.2 Feature Engineering

We created some new features to improve the training of the Light GBM model. Some new features include:

1) Relative floor size, *'rel_floor*', which is calculated by dividing *'floor'* by *'max_floor'*.
2) Room size, *'room_size*', which is calculated by dividing *'life_sq'* by *'num_rooms'*.
3) Month,*'month'*, which is extracted from the timestamp itself.
4) Age of property at time of transaction, *'bought_minus_built'*, which is the difference between the year from the timestamp and *'build_year'*.
5) Price per area, which is calculated by dividing the target price by *'full_sq'*

We also transformed categorical values to binary values, for example, values like *‘yes’* were transformed to 1 while *‘no’* were transformed to 0. *‘Investment’* and *‘OwnerOccupied’* were transformed to 0 and 1 respectively.

7.2 LightGBM

We identified a significant difference in the distribution of ‘price_doc’ in relation to the feature ‘product_type’. Therefore, two LightGBM models were trained separately on the distinct values of product_type, *OwnerOccupied and Investment*. Consequently, the hyperparameters of both models are tuned separately.

8. Weighted Model Averaging

Weighted model averaging (WMA) is a technique for combining the predictions from multiple machine learning models into a single prediction. Each model in the ensemble is assigned a weight, which represents its importance in the final prediction. The weights are assigned based on a variety of factors, such as the model's accuracy on a held-out dataset or the model's complexity
In our case, the best result in our proposed solutions (chapter 5) and the result in further improvements (chapter 7) are assigned weights of 0.41 and 0.59 respectively. This was largely determined by their performance in predicting the target variable. The final prediction, obtained by taking the weighted average of the predictions, nets us the position of 12th on the public leaderboard.

9. Conclusion

In many real-world datasets and problems, there are no clear-cut answers. Instead, one needs constant testing and refining, in order to formulate an innovative solution. The Sberbank competition is no exception. Through our extensive cleaning and processing, we were able to tackle the problematic dataset presented to us. We successfully dealt with erroneous and missing data with a myriad of techniques. This, coupled along with the selection of appropriate models allowed us to perform significantly better than the majority of submissions on Kaggle.

Both LightGBM (LGBM) and XGBoost (XGB) are powerful gradient-boosting frameworks known for their efficiency and effectiveness in handling large datasets and complex problems. This is due to their robust algorithms and ability to deal with outliers. To further improve on our methodology, Weighted Model Averaging was used to leverage the strengths of both of these models to create an even more robust and accurate prediction. This improvement enabled us to achieve our best possible score, securing us the 12th position on the Kaggle leaderboards.

References

[1] T. Chen and C. Guestrin, “XGBoost,” *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2016. doi:10.1145/2939672.2939785

[2] E. Khandelwal, “Which algorithm takes the crown: Light GBM vs XGBOOST?,” Analytics Vidhya, https://www.analyticsvidhya.com/blog/2017/06/which-algorithm-takes-the-crown-light-gbm-vs-xgboost/#:~:text=of%20Light%20GBM.-,Advantages%20of%20Light%20GBM,result%20in%20lower%20memory%20usage (accessed Nov. 19, 2023).

[3] G. Ke et al., “Lightgbm: Proceedings of the 31st International Conference on Neural Information Processing Systems,” Guide Proceedings, https://dl.acm.org/doi/10.5555/3294996.3295074 (accessed Nov. 19, 2023).

[4] S. Saha, “XGBoost vs lightgbm: How are they different,” neptune.ai, https://neptune.ai/blog/xgboost-vs-lightgbm (accessed Nov. 19, 2023).

[5] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, “Optuna,” *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2019. doi:10.1145/3292500.3330701

[6] S. S, “Hyperparameter tuning using optuna,” Analytics Vidhya, https://www.analyticsvidhya.com/blog/2020/11/hyperparameter-tuning-using-optuna/#:~:text=Optuna%20is%20a%20software%20framework,%2C%20bayesian%2C%20and%20evolutionary%20algorithms (accessed Nov. 19, 2023).

[7] Keremt, “Very extensive cleaning by Sberbank discussions,” Kaggle, https://www.kaggle.com/code/keremt/very-extensive-cleaning-by-sberbank-discussions/notebook (accessed Nov. 19, 2023).