Machine Learning Capstone Project

-

Final Report

Project Overview

Moscow's housing market flourished in the last decade, setting and outstripping multiple records. In 2015 3.8 million square metres of new housings were built. This is four times as much as were build in New York City in 2014. This upward trend is in sharp contrast to Russia's overall economic situation. Since investment in property is a major financial endeavour for private investors as well as companies, a reliable machine learning model to determine real estate prices would be very beneficial. The task of building such a machine learning model was posted as part of a kaggle competition. Participants were asked to reliably predict prices of real estates in Moscow. Sberbank, the competition's host, provided datasets of real estates (e.g. number of rooms, living space) and macro-economic parameters (e.g. oil price, stock market). Based on this data real estate prices of a test dataset had to be predicted.

Problem Statement

The problem to solve is a regression problem. A continuous target variable (i.e. housing price) has to be predicted using various features. The features, as mentioned above, are composed of property-related features and macro-economy-related features. Most of the features are continuous and numeric but there are also categorical, ordinal and discrete features. What makes this tasks challenging is that Russia's economy is extremely volatile and the housing prices are not obviously influenced by it. Another problems arises from the data itself. The provided datasets are extremely noisy, incomplete and features show high collinearity. This makes accurate predictions virtually impossible. Incomplete data cannot be handled by most machine learning algorithms, therefore those values have to be imputed. Collinearity causes an inflation of the standard error and therefore it might cause significant features to become non-significant. The most important step for a successful will be a thoroughly data preprocessing step.

Metrics

The Root Mean Squared Logarithmic Error (RMSLE) was used in combination with a 5-fold cross validation (CV) as the evaluation metric. The RMSLE will penalize (big) differences between big numbers less than would the Root Mean Squared Error. For the CV the data will be split in a 80:20 ratio (test,train*, respectively).
The RMSLE is calculated the following [11]:

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

ϵ : RMSLE value (score)
n : total number of observations in the (public/private) data set
pi : prediction
ai : actual response for *i*
log(*x*) : natural logarithm of *x*

The macro dataset shows a high degree of collinearity and has to be reduced. In order to estimate the degree of collinearity the variance inflation factor was calculated. A high VIF shows a high collinearity of the feature, while a VIF of 1 shows no correlation of the features. Features with a VIF higher than 5 were excluded from further analysis. Since the calculation of the VIF is extremely time intensive, the results from such a calculation were taken from https://www.kaggle.com/robertoruiz/dealing-with-multicollinearity.

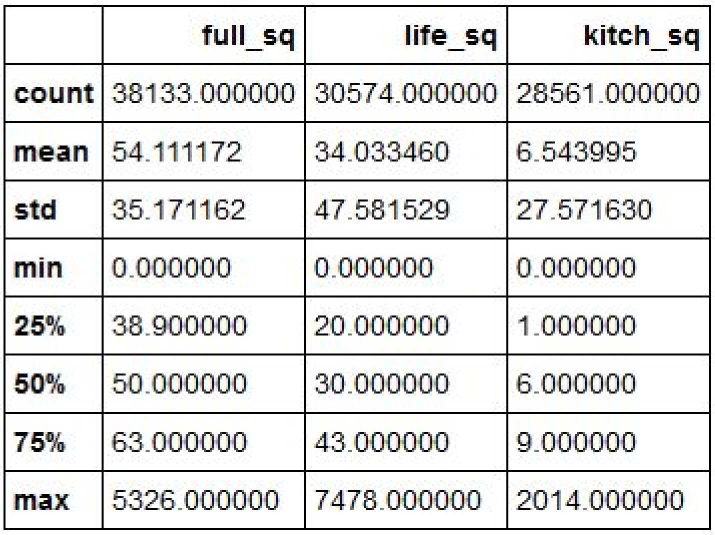
The Variance Inflation Factor is calculated the following:
VIF = $1/(1-R^2)$

II. Analysis

Data Exploration

The datasets used are provided by the Sberbank Russia as part of a Kaggle challenge. The data comes in three separate files, namely a training and a testing csv file and a csv file with various data regarding Russia's economics (train.csv, test.csv and macro.csv, respectively). The training dataset has 30,471 entries with 291 features and one target variable (sale price). The testing dataset has 7,662 entries with 291 features. The macro.csv file has 2484 entries with 100 features. The macro.csv file as well as the test.csv and train.csv all share a timestamp feature. The test.csv and train.csv files are mostly composed of discrete and continuous numeric values (157 features of integer type and 119 features of float type) and only some nominal (e.g. sub_area) and ordinal (e.g. ecology) features (16 features). The macro.csv file is composed of 96 continuous numeric values (2 integer and 94 float types) and 4 features displayed as object type. The macro includes features of importance for the Russian economic. For instance the country's GDP, the exchange rate of rubel toward other currencies like euro. Further, information about the general population is provided like mortality, childbirth or number of marriages. Train.csv and test.csv contain features about the specific houses. Important features, for instance, are square meter, living room m^2, build_year, population density of the neighbourhood. Besides property specific information also features regarding the neighbourhood are provided. Those are for instance number of schools, shopping centers or cafes in the neighbourhood. Also some information of the location of the property can be found in the features. For example in features like kremlin_km, railroud_1line etc.
When analyzing the data for missing entries, I found that from 302 features, 56 have missing values. Which equals to roughly
19 % of features with missing entries. Of those, some show a very high percentage of missing entries (e.g. museum_visitis_per_100_cap - ~60% missing), while others show only some (e.g. floor < 5% missing) (Fig 1).
Also, looking at the features it becomes apparent that some of them are correlated to each other. For example “floor” and “max_floor” or “full_sq” and “kitch_sq”. It is obvious that “max_floor” must always equal or less than “floor” and the same holds true for “full_sq” and “kitch_sq”. Also values must be in a reasonable range.

Table 1. Descriptive statistics of full_sq, life_sq and kitch_sq



	full_sq	life_sq	kitch_sq
count	38133.000000	30574.000000	28561.000000
mean	54.111172	34.033460	6.543995
std	35.171162	47.581529	27.571630
min	0.000000	0.000000	0.000000
25%	38.900000	20.000000	1.000000
50%	50.000000	30.000000	6.000000
75%	63.000000	43.000000	9.000000
max	5326.000000	7478.000000	2014.000000

Looking at table 1 it becomes obvious that there are outliers (max values) and that some values are probably infeasible. For example when comparing the maximum values for full_sq and life_sq. Also entries with 0 are not possible and must be handled appropriately.

Exploratory Visualization

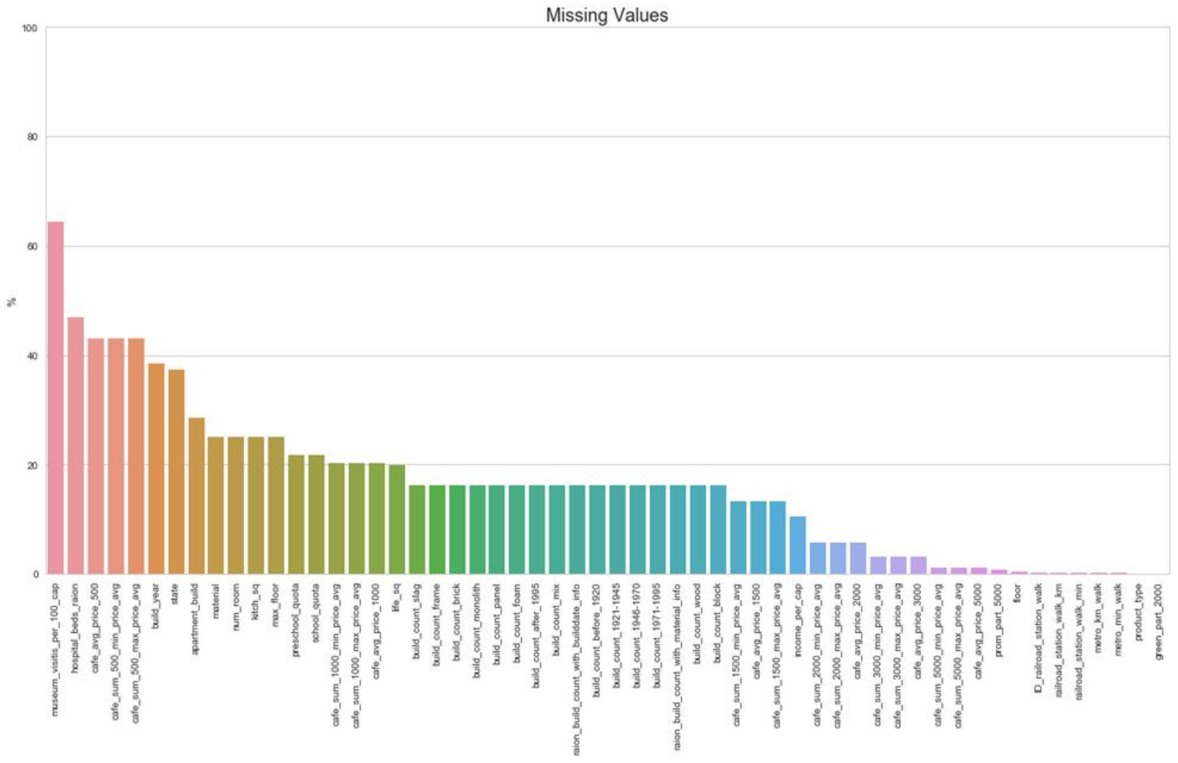


Fig. 1 Missing values from shown features. From left to right the percentage of missing values decreases. Looking at features one can notice that some very important features show a high degree of missing values (e.g. kitch_sq, build_year)

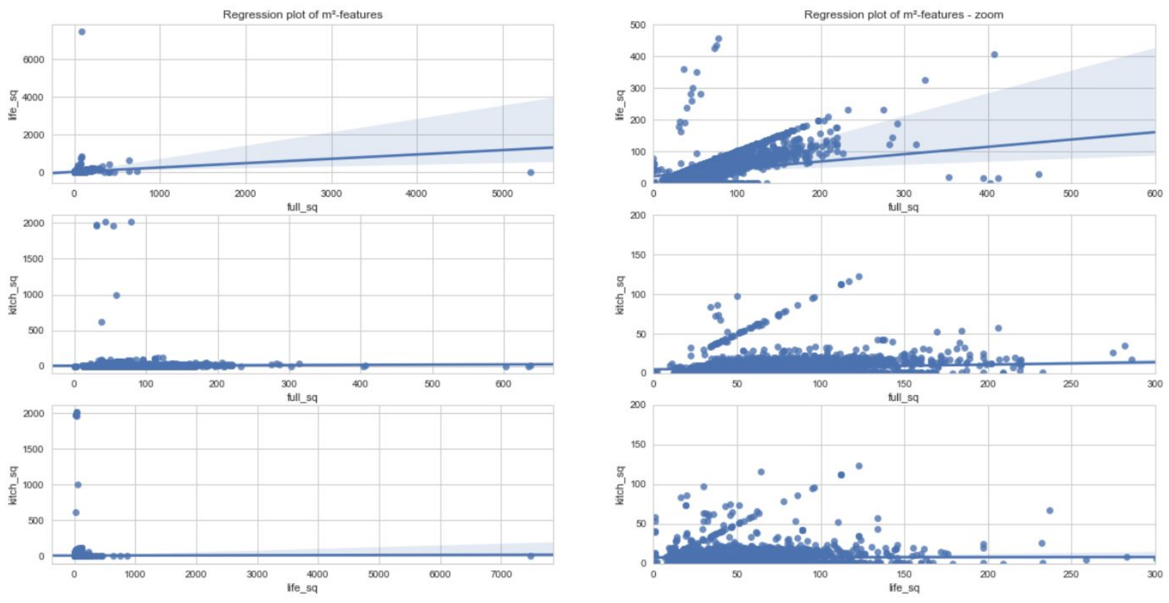


Fig. 2 Regression plots of property area features. Area features are plotted against each other. On the left side all values are plotted, while on the right side a zoomed view is provided. One can notice a couple of things when looking at the plots.

1. For some values ***life_sq > full_sq***.
2. For some values ***kitch_sq > full_sq***.
3. All three features show some outliers.
4. The ratio of ***full_sq : life_sq*** seems to be terribly off for some cases.

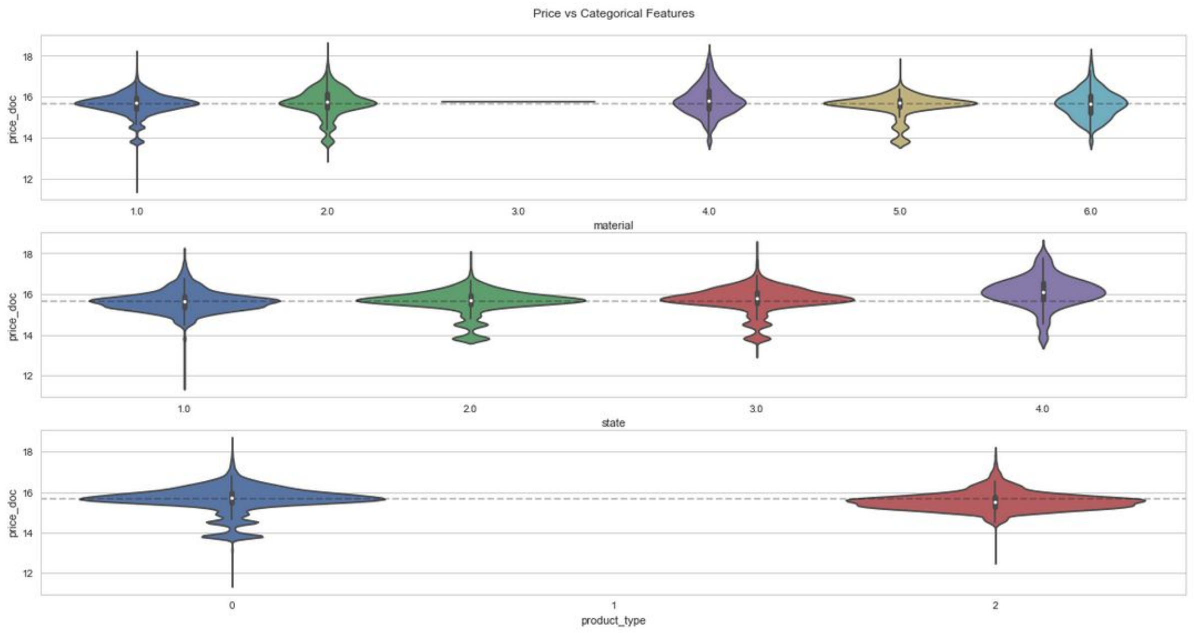


Fig 3. Price vs. Categorical Features. **Upper row** Material type vs price. Two things can be said. First, The material type does not seem to have a significant influence on the properties price and second, material #3 is very rare (only one occurrence). Material #1 seems to have

a slightly higher variance and the minimum price. **Middle row** State vs price. The state of the property is related to the price. State 1 is the least valuable while state 4 is the most valuable. However, while a relationship is undeniable, it seems not to be very strong. **Lower row** Product type vs price. Looking at both product types, there seems to be no influence of the product type on the price.

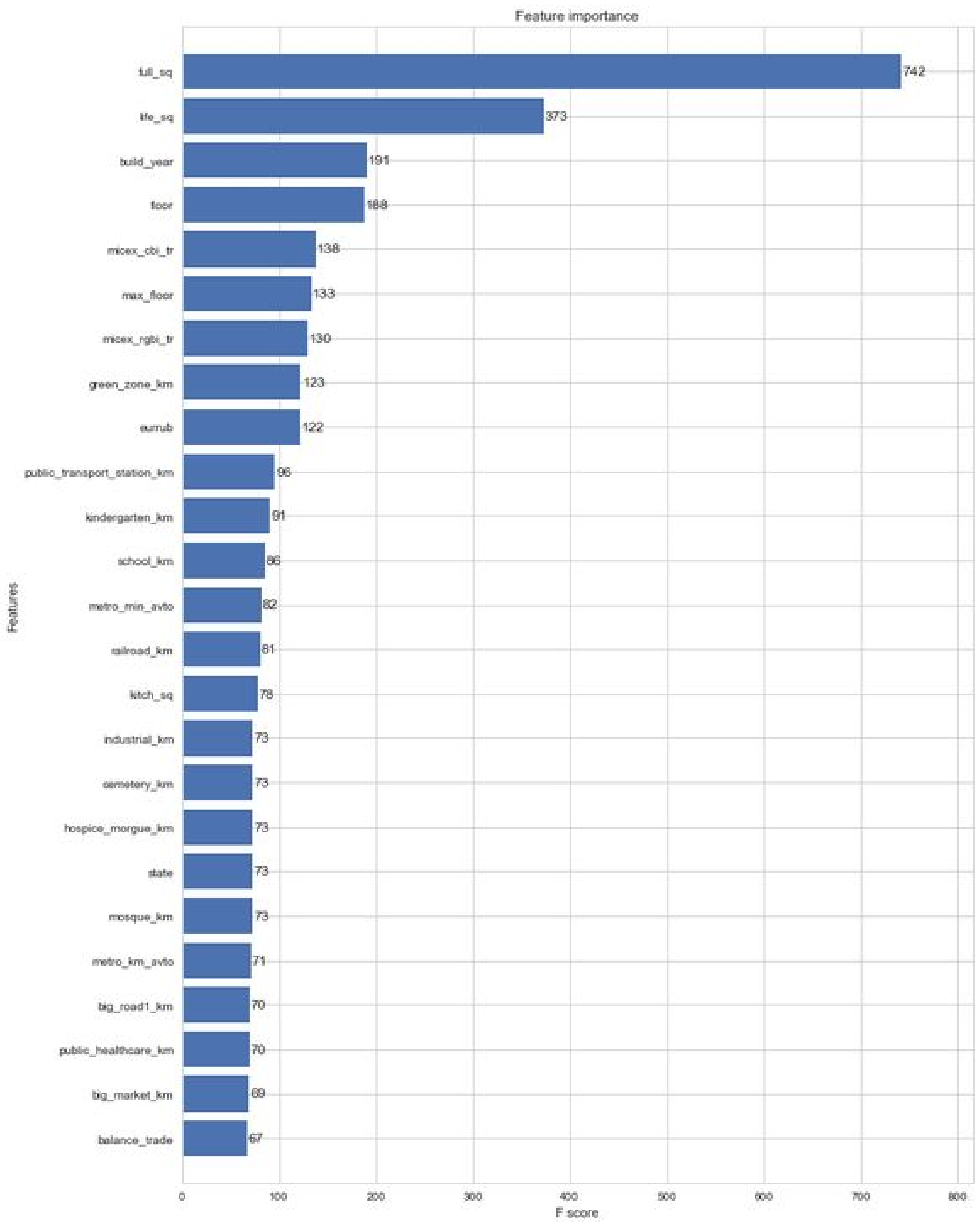


Fig. 4 XGBoost for determination of feature importance. Here the 25 most important features are shown. The feature full_sq is by far the most important feature, followed by life_sq. Those two are followed with a sharp drop by build_year and floor. This again puts emphasis on the idea that features directly attributed to the property itself are the most important ones.

Algorithms and Techniques

In order to solve this problem I will various machine learning algorithms and techniques. For the data preprocessing I will use variance inflation factor (VIF) analysis, Isolation Forest and principal component analysis (PCA). The VIF technique has already been briefly described above. Generally, it is a technique in which a regression analysis of each feature is performed and features that have a VIF above a certain threshold (5 - 10 as a rule of thumb) are deemed highly collinear and are therefore discarded for further analysis. The Isolation Forest algorithm belongs to the family of ensemble methods. This algorithm picks a feature and a random split value in the range of feature_min and feature_max. The picking of the split value is recursively repeated. Abnormal features tend to have shallow trees. The forest's shallowness is therefore taken as a measure for abnormality. The PCA analysis is performed in order to reduce the dimensionality of the dataset. The number of dimensions will be chosen in such a manner that around 90% of the underlying variability can be explained. Since the PCA can be influenced by a features intrinsic variance, the PCA analysis will be performed on a normalized and scaled dataset. Besides a traditional PCA a partial PCA will be performed. Here, the 10 most important features of the dataset are being determined via a boosting algorithm (XGBoost; Fig. 4). Those features will then be excluded from the PCA. This has the advantage that, while still greatly reducing the dimensionality of the dataset, it will be easier to interpret the results. PCA analysis might also bring some insight into which features of the dataset are having the greatest influence on the data's variance. For the actual real estate price prediction part I will use two boosting algorithms, one random forest and lasso and ridge linear regression algorithms. The boosting algorithms are XGBoost and LightGBM. Both gradient boosting algorithms will generate numerous weak learners, which then will predict the prices based on the prediction of the previous learner. The correctly predicted instances will get a small weight, the wrongly predicted ones a high weight. Therefore, the algorithm focus on the hard to predict instances. When looking at the hyperparameters of the boosting algorithm, those can be divided roughly into three categories: tree-related, boosting-related and others. Those algorithms will take the subsets of input data and build an ensemble of trees in order to cumulatively predict the price. The random forest algorithm, just like the boosting algorithms, is based on a set of classification and regression trees (CART). In contrast to the boosting algorithms, the random forest algorithm is not based on weak learners. The CARTs are not as shallow as those generated for the boosting algorithm (stronger learner). The lasso and ridge regressions are regularized linear regressions. Lasso and ridge are using L1- and L2-error penalization, respectively. While ridge regression cannot zero out the coefficients of the linear model (all features are selected), lasso regression can do so and therefore only a subset of features can be selected. In general both are trying to build a linear model **ß1X1 + … + ßnXn** which explains the target variable **y**. When considering the data at hand it is not possible to use it unprocessed for the prediction. While, the boosting algorithms do possess an integrated mechanism to cope with missing features, while for the random forest and the linear regression models, missing features have to be imputed. Further categorical features have to be encoded and ideally the data has to be scaled and normalized.

Benchmark

The benchmark for my model’s performance will be the average price. Predictions of the model will be compared towards the most simplistic model which just assumes the target variable to be the mean price. Here the RMSE of the prediction using the mean price of the differently preprocessed datasets is:

The rmse of df1 prediction is: 0.6121502702
The rmse of df2 prediction is: 0.6121502702
The rmse of df3 prediction is: 0.587188741106
The rmse of df4 prediction is: 0.587188741106
The rmse of df5 prediction is: 0.587188741106

Further, I will take the Kaggle leaderboard into account to evaluate my model’s performance.

III. Methodology

Data Preprocessing

A look at the macro-dataset set shows a lot of features which might be correlated. For example unemployment vs. employment. Such features might inflate the standard error of the coefficients of the regression analysis and thereby causing significant features to become insignificant. A common practice to estimate the collinearity of any feature a variance inflation factor analysis can be performed. Here, I performed a VIF analysis on the macro-dataset with a threshold of five. After VIF calculation only the following thirteen features remained in the macro-dataset:
"timestamp","balance_trade","balance_trade_growth","eurrub","average_provision_of_build_contract", "micex_rgbi_tr", ”micex_cbi_tr", "deposits_rate", "mortgage_value", "mortgage_rate", "income_per_cap", "museum_visitis_per_100_cap", "apartment_build".
Those remaining features were fused to the test and the train dataset based on their timestamp feature. (*The calculation are very time intensive and therefore the results of such a VIF analysis were taken from: https://www.kaggle.com/robertoruiz/dealing-with-multicollinearity/notebook/notebook*)

I intuitively deemed the features directly describing the property, like full square meter, most important for predicting the price. This has also been strengthened by XGBoost feature importance analysis (Fig 4). Therefore, I investigated those features in depth to identify possible data abnormalities and correct those.
First, I investigated the continuous features full_sq, life_sq and kitch_sq. A plot revealed some severe outliers, which have to be removed.
When we look at the features full_sq, life_sq and kitch_sq we can set up following rules:

1. full_sq >= life_sq > kitch_sq
2. full_sq > 5 m^2 & life_sq > 5 m^2
3. full_sq < 500 m^2
4. life_sq < 500 m^2
5. (kitch_sq >= 1 m^2 & full_sq 1 m^2
6. life_sq/full_sq > 0.4

I chose a cutoff of 5 m^2 since there are micro apartments with comparable sizes. Entries that do not fulfill the above mentioned criteria are replaced with NaN.

Next, I cleaned the discrete features, which are max_floor, num_floor, build_year and num_room. When we look at max_floor, then the value should not be larger than 81. This is because the Oko1 is Moscow’s largest skyscraper with 81 floors. Further, for any entry the num_floor cannot be larger than max_floor. Also, the life_sq : full_sq ratio should be reasonable. I chose 0.4 as a threshold here. The num_room must be larger than 0 and should have a reasonable size. Therefore, the life_sq is divided by the num_room. The average room size shouldn't be smaller than 4 m^2.

When investigating the build_year there are two obvious outliers which will be manually replaced, also the build year shouldn't be earlier than 1500. Again, all variables that violate the stated rules are replaced with NaN. The categorical features are seemingly without errors. Only one is a state which is 33 instead of 3.
Further, I generated some new features which are relative floor, average room size, year, month and day of the weak. Relative floor was calculated by dividing floor by max_floor. The average room size by dividing life_sq by num_room. The dates were extracted from the timestamp feature, which was dropped afterwards.
Finally, when we look at the target variable it shows a strong skew. Therefore, a log transformation of the target variable is performed.

After correcting the data abnormalities of the above stated features further preprocessing steps were performed. Since, the algorithms I chose have different prerequisites for their input data, I generated various datasets.

First, a dataset representing the original unaltered data, without addition of macro-economy data. This dataset will serve as benchmark for the refined datasets.
Second, a dataset which is a merge of property-related, encoded features and macro-economy data (**note**: the macro dataset has already been reduced via VIF analysis). In this second dataset the categorical features already have been label encoded.
Third, a dataset which further implements a imputation of missing features. Further, the features are scaled and outlier were removed via an Isolation Forest algorithmus.
Forth, a dataset which's dimensions were reduced via PCA.
Fifth, a dataset which is the result of a partial PCA. Here, the 10 most important features were identified via XGBoost and the remaining features were fed into PCA analysis.

Implementation

The label encoding has been performed on all features with "object" type. Classes between 0 and n-1 have been assigned to the values of those features. For the imputation of the missing features the percentage of missing values of each feature has been calculated. Features with more than 5% missing values were dropped from further analysis. The cutoff was empirically determined by comparing prediction results with various thresholds. The remaining features (less than 5% missing values) were imputed by replacing missing values with the median value of the respective feature. I chose the median to ensure that the imputation is not negatively influenced by outliers.
Further, the data was scaled to be in the range of 0,1 using sklearn's MinMaxScaler. After scaling a further normalization using sklearn's normalize was performed.
Afterwards, outlier were detected via a Isolation Forest algorithmus. Here, the default parameters were taken to identify possible outliers. 3814 out of the 38133 samples were deemed to be outlier and therefore removed from the dataset.

For the fourth dataset, a PCA was performed. As basis for PCA the imputed dataset just described was taken. First the PCA components were analysed on their ability to explain the underlying data variability. The first 40 dimensions explained about 90 % of the data's underlying variance and were therefore taken for the generation of the reduced fourth dataframe.

For the fifth dataframe the 10 most important features were identified via XGBoost algorithm. An untuned XGBoost algorithm was trained on the dataset and the .plot_importance function was used to show the 25 most important features based on their F-score. The ten most important ones were then removed from the dataframe and stored separately. The remaining features were used for PCA analysis. The first ten PCA components explained roughly 60 % of the data’s underlying variance and I chose them to be, in combination with the 10 most important features, sufficient. Finally, the excluded 10 features and the 10 PCA components were merged into the fifth dataframe.

The target variable was log transformed (np.log1p) since many machine learning models work better with normal distributed data. Therefore, I decided to use the Root Mean Squared Error (RMSE) instead of the RMSLE for the evaluation of the models performance. For the final submission, the results of the prediction were transformed back into the original dimensionality and the RMSLE was applied.

The different machine learning algorithms and techniques I employed had different requirements that had to be fulfilled. The boosted tree methods (XGBoost and LightGBM) did not require any data refinement. Both are able to handle missing data as well as categorical data. The other methods on the other hand did require data imputation of missing values and encoding of categorical features. The different preprocessing approaches led to five different datasets. The linear regression as well as the random forest algorithm were only tested on the imputed dataset and the PCA/partial PCA datasets.

In the first attempt of predicting the target variable the algorithms were run out of the box without tuning their hyperparameters.

Refinement

After initially testing the out of the box machine learning models on the learning tasks, the RMSE of the models on the individual datasets was calculated.

```
BENCHMARK
The RMSE for BENCHMARK for df_train1 is 0.612150270200498
The RMSE for BENCHMARK for df_train2 is 0.612150270200498
The RMSE for BENCHMARK for df_train3 is 0.5709465648655305
The RMSE for BENCHMARK for df_train4 is 0.5709465648655305
The RMSE for BENCHMARK for df_train5 is 0.5709465648655305
 ----------------------------------------------------------------------
LIGHT GBM
The RMSE for light GBM for df_train1 is 0.46548861110578665
Training took 8.79 s
The RMSE for light GBM for df_train2 is 0.464205793161556
Training took 10.23 s
```

```
The RMSE for light GBM for df_train3 is 0.4485461853903543
Training took 8.61 s
The RMSE for light GBM for df_train4 is 0.5651728176073144
Training took 1.07 s
The RMSE for light GBM for df_train5 is 0.45738539685426666
Training took 1.19 s
 ------------------------------------------------------------------
XGBOOST
The RMSE for XGBoost for df_train1 is 0.4140799690031782
Training took 64.44 s
The RMSE for XGBoost for df_train2 is 0.40942231376312505
Training took 73.18 s
The RMSE for XGBoost for df_train3 is 0.39548621998248024
Training took 81.86 s
The RMSE for XGBoost for df_train4 is 0.5460140063788276
Training took 11.42 s
The RMSE for XGBoost for df_train5 is 0.42418695140737794
Training took 9.36 s
 ------------------------------------------------------------------
RANDOM FOREST REGRESSOR
The RMSE for light RFR for df_train3 is 0.47643443717211803
Training took 80.29 s
The RMSE for light RFR for df_train4 is 0.6219574804018945
Training took 17.89 s
 -----------------------------------------------------------------
Ridge Regression
The RMSE for RIDGE REGRESSION for df_train3 is 0.4579025186823458
Training took 0.48 s
The RMSE for RIDGE REGRESSION for df_train4 is 0.5635890716840916
Training took 0.02 s
 ------------------------------------------------------------------
Lasso Regression
The RMSE for LASSO REGRESSION for df_train3 is 0.5709465648655311
Training took 0.27 s
The RMSE for LASSO REGRESSION for df_train4 is 0.5709465648655311
Training took 0.04 s
 ------------------------------------------------------------------
```

Looking at the RMSE one can observe two things. First, the boosting algorithms perform better than the random forest regressor or the linear regressors. Second, reduction of dimensionality via PCA or partial PCA led to a decrease of prediction accuracy. This becomes especially obvious when comparing the results from the random forest regressor on df_4 (PCA dataframe) with the benchmark. Here, the random forest regressor behaved significantly worse than the benchmark solution. The boosting algorithms still bet the benchmark solution but performed way better without PCA. The best solution of the out of the box algorithms was achieved by the XGBoost on the df_3. This dataframe is composed of the macro data and is label encoded as well as imputed. Based on this preliminary results I decided to further improve the prediction by tuning the hyperparameters of the XGBoost algorithm. For this I performed a five-fold cross validated grid search with the df_train3 as training dataset. Before hyperparameter tuning I fed the following parameters into the model:

```
xgb_params = {
   "n_jobs" : 4,
    "n_estimators" : 100,
   'learning_rate': 0.05,
   'subsample': 0.8,
   'colsample_bytree': 0.8,
   "min_child_weight":1,
   "gamma":0,
   'objective': 'reg:linear',
   'silent': 1
}
```

In the first round of parameter tuning I decided to tune ‘max_depth’ and ‘min_child_weight’. Min_child_weight is the minimum sum of weights in a child node. High values prevent overfitting, while too high values might result in uderfitting. The max_depth describes the maximum depth of the tree, again this can be tuned to control overfitting. Max depth of 3,5,7 and 9 and min child weight of 1, 3 and 5 was tested.
A max_depth of 5, as well as a min_child_weight of 5 were found to be the best of the selected parameters. A RMSE on the generated train data resulted in 0.47058647047405894. In the second round the parameters max_depth and child_weight were again optimized in a smaller value window of 4,5,6 each. The min_child_weight parameter remained at 5, while the max_depth performed better with 6. The new RMSE is 0.46986457846928686.
The next hyperparameter is gamma. The gamma value sets the threshold for the minimum positive reduction of the loss function needed in order to perform a split on a certain node. Here, 0.01, 0.02, 0.03, 0.04 and 0.05 were initially fed into the grid search. A gamma value of 0.01 was found to be the best, resulting in a new RMSE of 0.46980745216770498.
Next, subsample and colsample_bytree were tuned. The subsample parameter determines the percentage of entries to be samples for the tree. The colsample_bytree determines the percentage of features used for the tree. Values between 0.6 and 1 were fed into the grid search for subsample and colsample_bytree. The best value for colsample_bytree was 0.8 and the best for subsample 0.8. However, the resulting RMSE was again 0.46980745216770498, which is no improvement. Therefore, the grid search was repeated with finer steps between the values. Therefore values between 0.79 and 0.82 were chosen. The grid search returned that the 0.8 is indeed the best suited value and that the RMSE cannot be improved by altering colsample_bytree and subsample.
The next parameter, reg_alpha, is L1 regularization term on the weight. It was initiated with values 1e-5, 1e-2, 0.1, 1, 100. 1e-5 was best, and slightly improved the RMSE up to 0.46980745627430431.
After tuning those parameters via five-fold cross validated grid search the learning_rate was decreased to 0.01 and the number of estimators increased to 5000. With the optimized

parameters the RMSE was improved up to 0.4591709387232905. Afterwards the optimized parameters were used for the prediction of number of boost rounds and the final prediction on the train set resulted in a RMSE of 0.38671094547858137. So the tuning of the hyperparameters resulted in a significant improvement of the model. The final hyperparameters are:

```
xgb_params = {
        "n_estimators" : 1000,
        'learning_rate': 0.01,
        'subsample': 0.8,
        'colsample_bytree': 0.8,
        "min_child_weight":5,
        "max_depth" : 6,
        "gamma":0.01,
        'reg_alpha': 1e-05,
        'objective': 'reg:linear',
        'silent': 1,
        "n_jobs" : 4,
}
```

IV. Results

Model Evaluation and Validation

XGBoost was chosen as the final machine learning algorithm. It performed best out of the box compared to the other tested algorithms. In terms of speed it was significantly worse than LightGBM. When looking at the various trainings dataframes we can see a similar trend of RMSE of the predictions when looking at XGBoost and LightGBM. Both algorithms already showed a huge improvement when compared to the naive solution. When the post-VIF macro features are added to the dataframe a further improvement was achieved. This improvement was more prominent for XGBoost, than for LightGBM. While RMSE of LightGBM was reduced by ~ 0.001 the RMSE of XGBoost was reduced by ~ .005. A larger improvement was achieved after throughout data cleaning, scaling and imputation. LightGBM's score improved by ~0.016 and XGBoost's by 0.014. Therefore, the data processing equally benefitted both algorithms. The reduction of dimensionality via PCA or partial PCA worsen the predictive power of the algorithms. PCA resulted in predictions only slightly better than the benchmark, which scored ~ 0.5709466. LightGBM and XGBoost had a RMSE of ~ 0.561728 and ~ 0.546014, respectively. The partial PCA performed much better with RMSE of ~ 0.457384 and ~ 0.42418695 for LightGBM and XGBoost, respectively. The most prominent improvement of the partial PCA is the reduction of processing time. XGBoost took only 9.36 s on the partial PCA dataset, while for the complete dataset the computation took up to 81.86 s for the imputed dataset. However, for the hyperparameter tuning of XGBoost I chose the imputed, complete dataset since it performed somewhat better in terms of RMSE.
The hyperparameter tuning further improved the model, whereas tuning colsample_bytree and min_child_weight parameters led to to the biggest improvement. The final model uses a learning rate of 0.01, this should ensure a robust model by shrinking the assigned weights. A subsample and colsample_bytree of 0.8 should counter overfitting. The values are still large enough so that the trees will not underfit the data. The default value of min_child_weight is 1, here cross validation resulted in a optimal value of 5 which is higher. Higher values of min_child_weights are good to prevent the algorithm from learning not significant correlations of features. The max_depth was tuned to a value of 6, which is the default value. Too high values could result in overfitting. A gamma of 0.01 was found to lead to a slight improvement of the algorithm.
The XGBoost algortihm by itself is robust, this can be explained by the XGBoost algorithm itself. The colsample_bytree and subsample parameter, define that only a subset of the data will be used for training of the trees. Therefore, the final model is composed of various inputs which together ensure that the model generalizes well and do not overfit, at least when those parameters are properly set. Further, by training and testing the model on the differently preprocessed datasets, one can see that the model's capacity to predict data is not greatly influenced. For instance, compare the third dataset (imputed, outlier removed, scaled) towards the second. The second dataset has some outliers (noise) added and also shows more missing values. Still, the algorithm performs only slightly worse than it does on the third dataset. This emphasises the robustness of the XGBoost algorithm used here.
However, there is a major drawback which becomes apparent when predicting the real estate prices of the competition's dataset. Here, the algorithms achieves a RMSLE of

0.32789 in the public leaderboard, which is not too good. The second dataset, with less preprocessing steps resulted in a better score. There are two possible explanations for this. On the one hand, the model possibly overfits to the preprocessed training data, or on the other hand, there might be a difference between the datasets. The preprocessing steps were performed for both, the training datasets as well as for the competition's test dataset. Also, the XBGoost's hyperparameter colsample_bytree, min_child_weight and subsample should control overfitting. So it might be that the competition's test data has a different distribution, or different price determination, than the competition's train data.

Justification

The final result of the improved XGBoost model is significantly better than those of the benchmark. Also the LightGBM algorithm performed nicely on the training data, especially it's fast computation time were impressive. After performing a train-test-split on the training data, the XGBoost algorithm performed good and it was possible to further improve it by cleaning the data and tuning it's hyperparameters. However, on the competition's test data it performed not as good when training on the fully imputed data. This indicates that the underlying price distribution of the final test set is somewhat different than of the training data. Therefore, excessive preprocessing led to some overfit. The noisy data from the beginning was harder to predict for the algorithm, it showed a higher variance as seen by a higher RMSE. However, in the end it was able to generalize better to the test data. Again, this is only the case since the data seems to somehow differ from each other. So, is the final solution significant enough to have solved the problem? This really depends, on the one hand it is significant better than the benchmark solution. When training on the second dataset, which was not as thoroughly preprocessed than the third, the best RMSLE was achieved. The RMSLE of 0.32496 would be place 1771 of 3274 in the competition. On the other hand, a big part was the preprocessing of the data, which yielded to nothing in the end. Therefore, the preprocessing steps, even though they greatly optimized the algorithm on the training data, led to no improvement when predicting the competition's test data. Taken together, the chosen algorithm led to a significant enough solution, the major step of data preprocessing, apart from an educational aspect, was worthless.

V. Conclusion

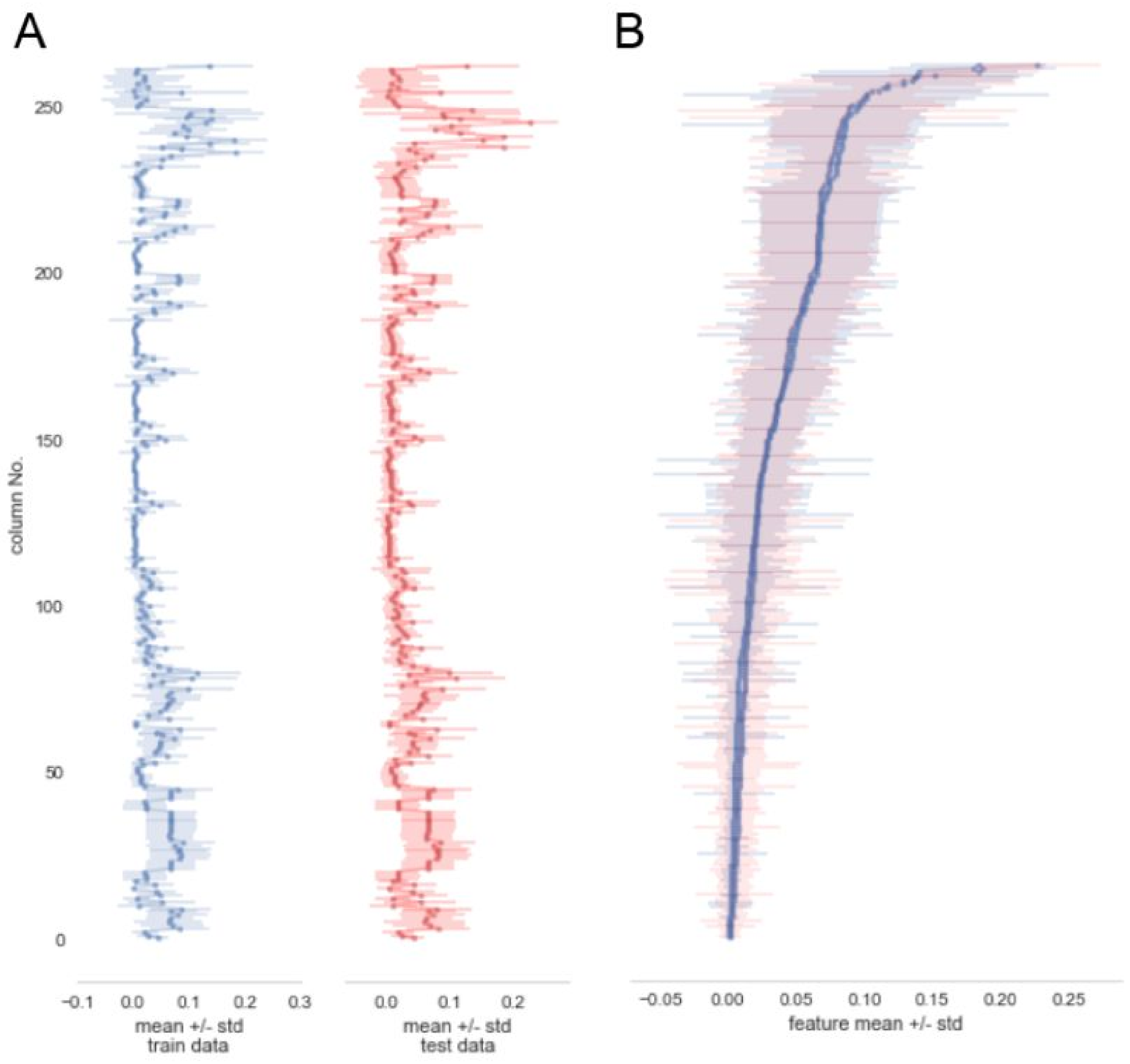


Fig. 5. Mean +/- standard deviation of the features. Mean and standard deviation are calculated on the third dataset. **A)** left: training data, right: test data. **B)** sorted overlay of training and test data.

Looking at the means and standard deviation of the training and testing data we can observe two things. First both dataset show a very similar structure. One cannot say that they significantly differ from each other. It therefore strange that the XGBoost algorithm's score on the training data differs so strongly from the testing data. Second thing one notices are the really high standard deviations of some of the features. This again emphasises the noise underlying the data.

Reflection

When first looking at the data, it became apparent that the data is quite exhaustive and by far not complete. Some of the features showed high importance for the property prices and were directly describing the property (e.g. full_sq). Those features were isolated and extensively analysed and cleaned. For the remaining features algorithms were employed for the preprocessing. Since the impact of the various preprocessing steps was not clear, five different datasets were generated. The training datasets were then split into train and test data and the mean price of the train data was used as a benchmark prediction for the test data. Five algorithms (LightGBM, XGBoost, RandomForest, Ridge and Lasso Regression) were used and tested against this benchmark. LightGBM and XGBoost both performed good. While LightGBM was much faster XGBoost yielded a lower RMSE and was therefore chosen. Afterwards, the parameters of the XGBoost algorithm were tuned via a five fold cross validated grid search. The resulting parameters were taken to make the final prediction on the competition’s test set. The final prediction yielded a RMSLE of 0.32496 would be place 1771 of 3274 in the leaderboard.

It was very interesting to start exploring the dataset, looking at some features in great detail and employing all these different machine learning techniques. Also getting to know some state of the art ensemble algorithms was really exciting. Reading a lot of XGBoost really helped to build a solid base and feeling for the algorithm’s potential and its hyperparameters. However, I really struggled with the project when the cross validation and improvement of my model on the training set did not yield any significant improvements on the competition’s test data. Here, I was really glad of choosing the capstone project as part of a Kaggle competitions. Reading the forum and seeing that I was not alone with this problem helped somewhat.
In the end, I think the general approach to solve this kind of problem is absolutely appropriate. It was somewhat frustrating and unlucky though that the work intensive data preprocessing did not yielded the expected improvement.

Improvement

While my model beat the proposed benchmark, there are better solutions. My model scored 0.32496 while the competition’s winner scored 0.30087. So, there definitely are things to improve. To improve the model’s predictive power I would suggest mainly two things. First, even more feature engineering might yield the desired improvement. Since the features which show the highest impact on the price prediction of the training data are not necessarily the ones most important for the prediction of the test data, it might be worth to analyse the data for data structures that might be a good indicator for both sets. One whole group of features describes distances to Moscow’s important building or subway stations. Using those features together with the suburb they are located in, it might be possible to exactly locate the property in Moscow. Such a feature could reduce the datasets dimensions (since distances are not that important anymore) and might be a more universal predictor for the property price.

On the other hand, one could munge the data in such a way that the algorithm's performance is not accessed via a train-test-split, but on the competition's test data. While this probably would lead to better results on the leaderboard, it would also lead to a overfitting towards the test data. Therefore, such a method might be applicable here, but in general it should not be considered good machine learning practice.