Michał Krzyżański

Kaggle: Michał Krzyżański https://www.kaggle.com/michakrzyaski
Student ID: 3172260

1. Research about real estate prices

In order to gain intuition and better understand the data I read several articles regarding what influences property prices. The most significant are:

- Location
- Size and usable space
- Age and condition
- Interest rates

In the dataset we have all the data but the last position. However interest rates influence property prices over time and we are considering the housing market at a single point, therefore they wouldn't be helpful anyways.

2. Look at the properties on google maps

The properties range from kiosks to a colosseum. This might pose problems as in some cases we might not be able to infer from the data itself whether the property is for example a big hall or a big villa.

3. Data pre-processing

Imputation and encoding

I assumed the data was obtained using web scraping, therefore:

a) In case of binary variables "nan" was imputed to be 0: in my experience with scraping where the bot fails there i probably no field on the website which indicates lack of the feature

b) In the case of categorical variable conditions I assumed livable conditions in all nan cases, the reason being it's the most average. I assumed for bad conditions people would be required to state it and for better conditions they would be incentivised to provide it. One could argue the use of KNN here could yield better results, however I do not believe so, since the condition is not correlated with any of the other features in the real world. The features were encoded on 1-4 scale, since there is a clear order to property states.

c) In the case of bathrooms and rooms I assumed 1 if not given, since again it is the " most default" similar to the case of binary variables. Properties without any bathroom are not very popular and by definition property with 0 rooms does not exist.

d) I have thought about what to do with n_floors and total_floors for a long time. Imputing it with KNN would make some sense, however I think I figured out a better way to do this.
   - I assumed that if total_floors is nan, then floor is the number of total floors as well.
   - I assumed that if n_floors is nan then total

    - For the rest of nans, assume that there is only one floor: the ground floor
    - For properties that ended up with negative total_floors I switch the number and add one (for ground floor)

e) I have imputed surface as follows: average surface*(n_rooms+n_bathrooms)

f) For the rest of missing data I used KNN imputer with sqrt(n_data) neighbors

Dealing with outliers

Knowing that most outlier removal methods based on normality assumptions
I have decided to manually remove outliers rather than use statistical methods or kernels, because most of the data does not seem to fit any known distribution.

a) They are in the data properties with 0 surface. The max surface of a property in the dataset is 999. Therefore at 1000 and above the values are overloaded to 0. I will set all to 1000 and add a categorical variable “massive” to adjust for that.

a) Some properties have expenses unproportionally high to the price. I removed those observations.

b) I removed data points with price over 20000000 EUR

c) Removed observations with above 50 total floors

Feature engineering

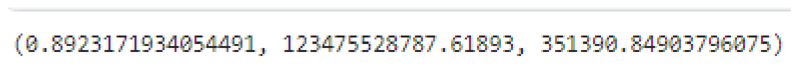
Since price varies greatly from other variables in magnitude I decided to regress the logarithm of price instead of price itself. Additionally I created new dummy variables. 1 for each city the properties find themselves in, 2 for size of the properties (big and small) and variable for properties that had 0 surface. This will allow the model to catch the relationships more easily. I have also decided to drop features with 0 correlation. Since I will be using a decision tree based model, I will not drop 0 correlation features, since there might be some non-linear complicated relation that the model will pick up.

The model

I decided to use XGBoost for 2 reasons: It performs well with multidimensional data and huge sample size at the same time being very fast to train and validate. XGBoost optimises loss function by fitting regression trees on the negative gradient of a given loss function.

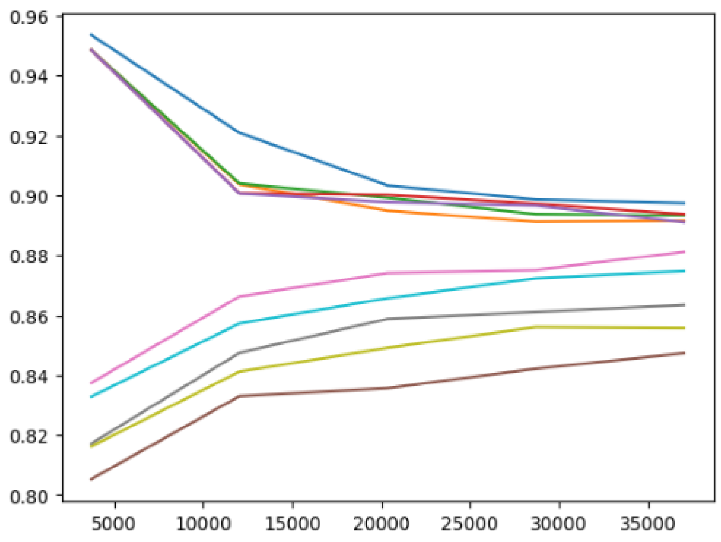
I searched for the most optimal parameter for XGBoost: max_leaf_nodes. It is a parameter that controls overfitting as it controls the size of the tree. I have estimated it should be around 30-50. I have added l_2 regularisation to further prevent overfitting.

Then I ran the model obtaining following results: score, MSE, sqrt(MSE)

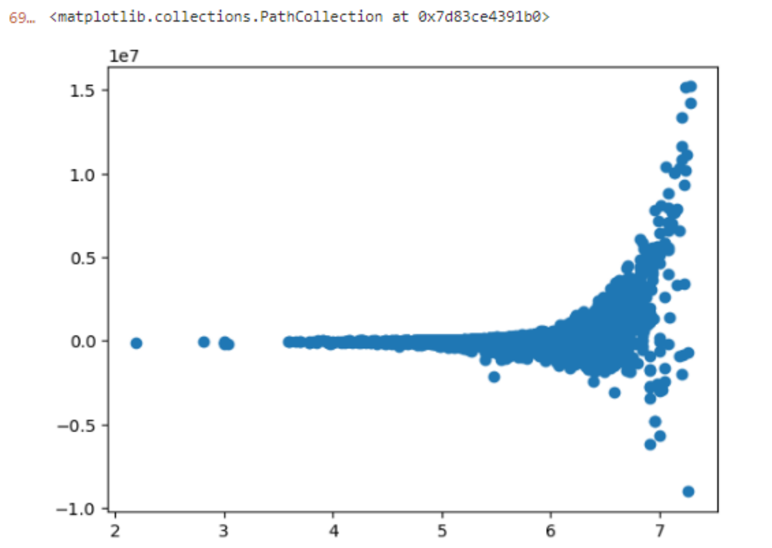


```
(0.8923171934054491, 123475528787.61893, 351390.84903796075)
```

Learning curves



Residual plot (log price vs actual price difference)



```
69… <matplotlib.collections.PathCollection at 0x7d83ce4391b0>
```

The score is satisfactory with r^2 being at 0.89. The MSE is pretty high.The model is not overfitted, which is confirmed by learning curves. However the model cannot properly predict values of properties with log price above 6. It is just possible that we don't have enough data. At the extremes, the price might be influenced by other factors, such as for example, material from which the house is built, furniture, etc.

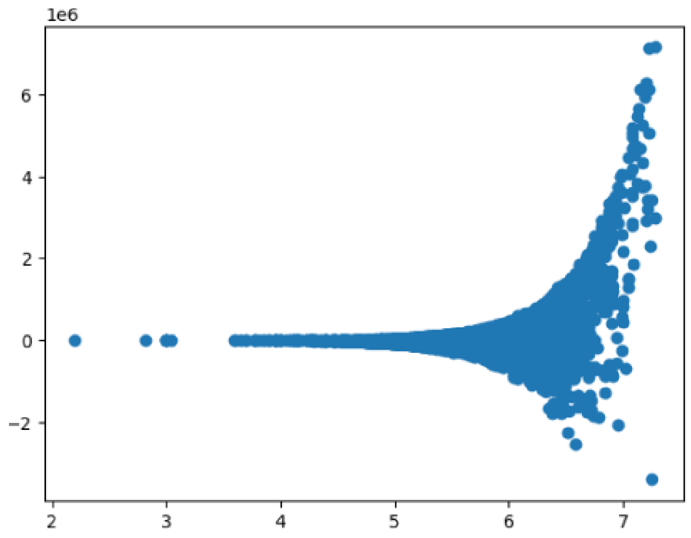
However I will try to further reduce mean squared error.

I decided to modify the model by putting it in ADABoost. ADABoost fits additional copies of regressors on the same dataset, but where the weight of instances are adjusted by the error of the prediction. This should mainly optimize the model at the extremes, where the biggest error occurs. I set ADAboost number of estimators to conservative 25 not counter overfitting. Results are given below.

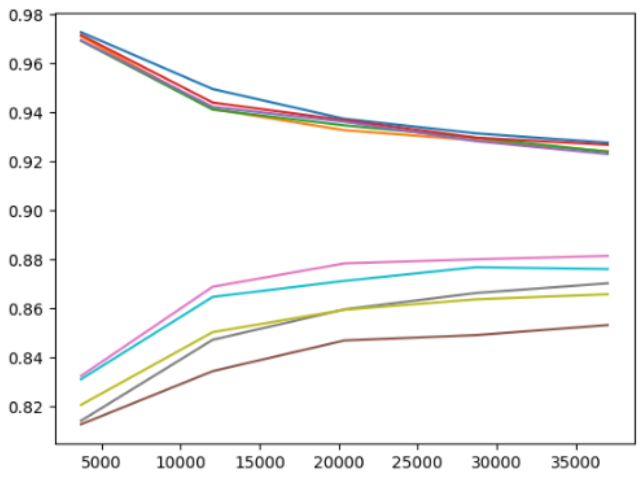
Score, MSE, sqrt(MSE)

```
(0.9207971551183655, 55711629452.30285, 236033.11092366438)
```

Residual plot (log price vs actual price difference)



Learning curves



As expected ADABoost improved the performance above 6 log price. Maximum error went down from above 15 million to 7 million (halved) and with it halved mean squared error. Training score improved but the testing score remained the same. In terms of MSE it is an improvement, however it is not so clear if bias-variance tradeoff in this case is at the optimal point. Yes, the model had an even lower MSE on the testing set, but if we were to take the property prices from next year I am not so sure that this model would be the best. I think it would adapt to novelties worse than the first model.