CENG-542 Project Report
Predicting German House Prices

Student: Joel Amarou Heuer, 202102201

Table of Contents

Goal

This project aims to find patterns which enable the prediction of prices for German houses based on different attributes. Due to this objective, exploratory (predictive) data mining is performed. More precisely, it is a regression problem – the prediction of continuous values (y-variable = house prices).

The following dataset is used to address the introduced problem: "germany_housing_data_14.07.2020"
(Source: https://www.kaggle.com/datasets/scriptsultan/german-house-prices). Two Experiments are conducted to identify patterns within this dataset to estimate valid house prices.
.

Getting to Know the Data

Features & Samples

The dataset used contains 10552 samples (rows) and the following 24 attributes (columns):

24 Attributes of the dataset		
Numerical (10)	**Categorical (13)**	**Text/Date (1)**
Living_space	Type	Free_of_Relation
Lot	Furnishing_quality	
Usable_area	Condition	
Rooms	Heating	
Bedrooms	Energy_source	
Bathrooms	Energy_certificate	
Floors	Energy_certificate_type	
Year_built	Energy_consumption	
Year_renovated	Energy_efficiency_class	
Garages	State	
	City	
	Place	
	Garagetype	

After a closer look at the features, it becomes obvious that State, City, and Place contain the same information only in different granularities. Thereby, the granularity of these features is hierarchized as follows: State > City > Place.

NaN Values

The number of NaN values (cells without value) is analyzed to understand the dataset better. This analysis shows that the number per NaN value varies enormously by feature (between 0% and ~77%).

Top 5 features with many NaNs	Top 5 features with few NaNs
Energy_consumption (76.94%)	Living_space (0.00%)
Year_renovated (49.31%)	Lot (0.00%)
Usable_area (47.23%)	Rooms (0.00%)
Energy_efficiency_class (45.67%)	City (0.01%)
Bedrooms (34.82%)	State (0.01%)

Statistical Properties

Then, the statistical properties of the dataset are investigated by applying:

```
df_source.describe(include="all")
```

Thus, a first insight into the nature of the different features concerning mean, standard variance, minimum, maximum, etc. can be obtained. It is noticeable that static properties cannot be calculated for many features since they are not numerical in nature but categorical or text-based. An average house in the dataset has about 7 rooms, 4 bedrooms, 2 bathrooms, 2 garages, a living area of 216m², and costs on average 556 693€.
An important finding is that the price is distributed very unevenly and therefore probably has some outliers. For example, the maximum house price in the data set is 13 000 000€, which is about 23 times higher than the average price.

Preprocessing

This section covers how data is prepared to be used as input for machine learning algorithms. Mainly *Data Reduction*, *Data Cleaning*, and *Data Transformation* are discussed.

Data Integration

Since the data set has many features (d = 24) and many samples (n = 10552), there is no need to integrate data into the data set. On the contrary, data must be removed.

Data Reduction (Dimensionality Reduction)

The first preprocessing step is to remove features that store the same information. *State*, *City*, and *Place* all indicate the location of a house, but at different granularities. For this analysis, it is sufficient to look only at the German states of Germany (*State*), so features *City* and *Place* are removed. Furthermore, all features with more than $\frac{1}{3}$ NaN values are removed. Although one could replace missing data using mean or mode. However, 33.333% missing data for a feature is, in my opinion, too high to obtain reliable data.

Discarded features are: *Energy_consumption*, *Year_renovated*, *Usable_area*, *Energy_efficiency_class*, *Bedrooms*, *Free_of_Relation*, *Energy_certificate_type*, *City*, *Place.*
→ The result of this preprocessing step is a DataFrame/table without duplicate and sparse features.

Data Cleaning (Handle Missing Data)

To clean the data, missing data (NaN values) are handled depending on the regarding feature. In the data cleaning process I undertook, handling these NaN values means they are replaced with their feature's mode. The mode is used instead of the mean since no outlier removal has been undertaken at this point. It is well known that the mean is very sensitive to outliers, whereas the mode is robust. Also, the mode can be applied to categorical data, but the mean cannot. However, the *State* feature was treated separately in advance. Since this column contains only 0.01% NaN values, all samples without a value for *State* were deleted.
→ The result of this preprocessing step is a DataFrame/table without NaN-values (mode mostly replaced NaN values).

Data Cleaning (Outlier Removal)

As a further step in data cleaning, outliers are identified and removed. The *z-score* method is used for this purpose. Since this method utilizes mean and standard deviation, it can only be applied to numeric features as well as for the y-variable (Price). Detecting a sample as an outlier is as follows: For each instance (row), the z-score is calculated for each numerical feature (column) as well as for the y-variable (Price). If at least one *z-score* z_k for a sample is not within the interval [-3, 3], it is considered an outlier. A total of 715 samples are detected as outliers by using the *z-score* method. Subsequently these samples are removed from the data set.

Looking more closely at the cleaned data set, it is noticeable that there are still outliers concerning the y-variable (Price). For example, there are some samples with a Price = 0€ and some samples with prices of several million €. These exceptional houses are not addressed in this work. Accordingly, all samples whose Price is not in the interval [50 000, 10 000 000] are also considered as outliers. In this context, it should be noted that the average Price for German houses is between [300 000, 1 000 000], depending on the location.

A total of 966 samples are detected as outliers by the z-score method and the interval method and removed from the data set.

Figure 1 - Effect of Outlier Detection

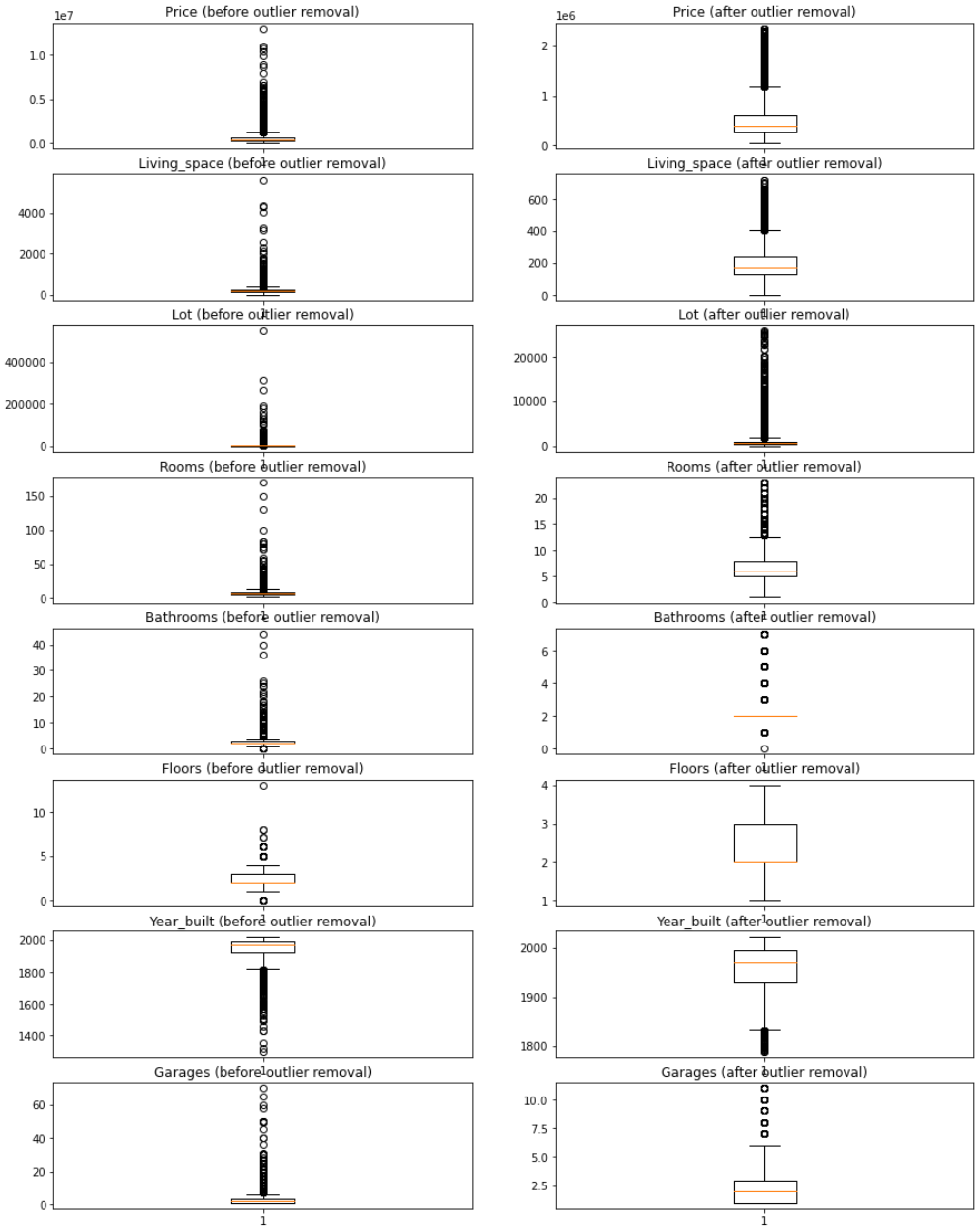

Figure 1 shows Data distribution for each numeric feature and y-variable (Price) in the dataset before (left) and after (right) outlier removal.

Figure 1 shows several graphs. Two boxplots belonging to the same feature are listed per row: *Left*: Data distribution of Feature F_i before Outlier-Removal. *Right*: Data distribution of Feature F_i after Outlier-Removal. It can be clearly seen that there exist a lot of outliers before outlier removal (left). After outlier removal with interval method for price and the *z-score* method (right) the most significant, but not all outliers are removed. Instead of the *z-score* method, one could have used the *interquartile range* method (*IQR*). However, the disadvantage of this would have been that too much data would have been discarded. It is worth noting that mean price and maximum price are strongly shifted after outlier removal:

	Before Outlier Removal	**After Outlier Removal**
Price (min.)	0 €	51 000 €
Price (mean)	556 685 €	499 877 €
Price (max.)	13 000 000 €	2 350 000 €

→ The result of this preprocessing step is a DataFrame/table with 966 fewer samples than before. These samples are identified as outliers by using (a) z-score method and (b) applying minimum threshold = 50 000€ and maximum threshold = 10 000 000€ for y-variable (Price).

Data Transformation (One-hot Encoding)

Next, the categorical features are tackled. Since categorical features cannot be given as input to learning algorithms, they must first be converted into a suitable form. This method is one-hot encoding, where each category from a column (feature) spans a separate binary column. The original categorical features are then deleted. This procedure extends the dimensionality of the dataset to 170 features (columns).
→ The result of this preprocessing step is a DataFrame/table with 170 features (all previous categorical features are encoded with one-hot encoding).

Data Transformation (Rearrange Dataframe)

Here, the data is arranged so that all features are at the beginning of the DataFrame and the y variable (Price) is at the end. Given that the DataFrame has d columns, the features are in the first (d-1), and the y-variable is in the last column. That structure of the DataFrame makes it easier to process the data later, mainly if the data is divided into training and test data.
→ The result of this preprocessing step is a DataFrame/table with such a structure that all features are placed in the first (d-1), and the y variable is set in the last column.

Data Reduction (Feature Selection)

Due to the fact that the DataFrame has increased enormously in dimensions (d = 170) by one-hot coding, it is interesting to see which features actually correlate highly with the y-variable (Price). As a result, one can discard rather irrelevant features and prevent complexity of learning algorithms.

Figure 2 - Correlation Heatmap

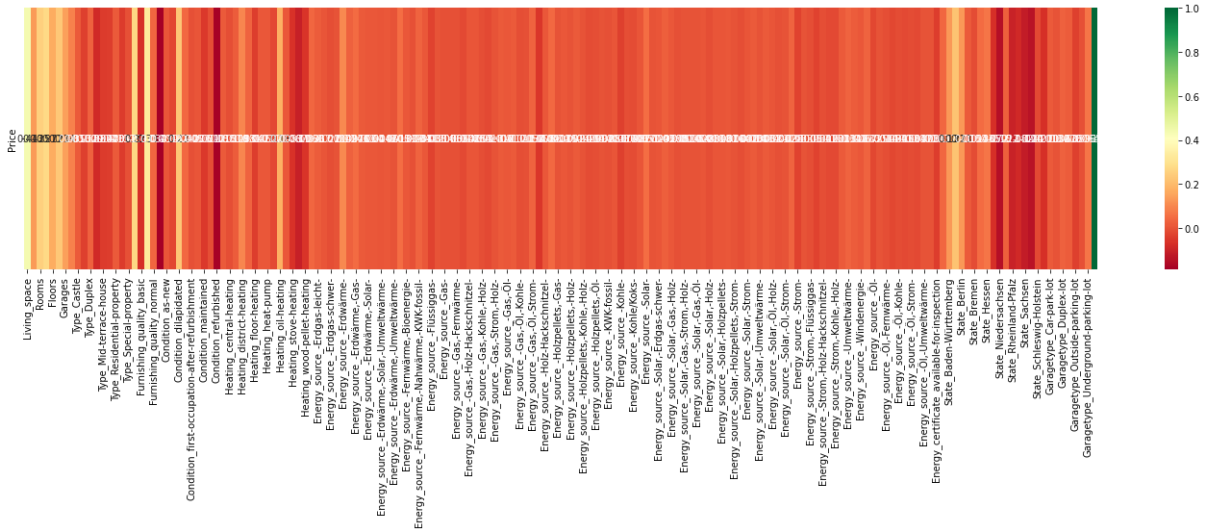

Figure 2 shows a heatmap representing how strong features correlate with the y-variable (Price), with following color encoding: red = little, yellow = moderate, and green = much correlation.

Using a correlation heat map (**Figure 2**), a visual examination is first made of which features correlate most highly with the y variable (Price). The color coding is red-yellow-green, where red means little correlation, yellow means moderate correlation, and green means high correlation. Clearly, most features correlate little with the y-variable (Price) and only some features correlate moderately.

The five features with the highest correlation are as follows:

Feature	Living_space	Furnishing_quality_luxus	Bathrooms	Type_Villa	Rooms
Cor. factor	0.44	0.32	0.27	0.27	0.25

For the final feature selection, a minimum threshold value for the correlation factor is set. All features with a lower correlation factor than 0.2 are not required for further analysis and are therefore discarded.

→ The result of this preprocessing step is a DataFrame/table with only those features that have a high correlation factor with respect to the y-variable (Price).

Data Transformation (Split Data in Test & Train)

One of the last steps of preprocessing is to divide the data into training data (80%) and test data (20%).
→ The result of this preprocessing step is that the DataFrame/table is now divided into training and test data.

optional: Data Transformation (Scale Data)

As an optional step, the numerical features of the training data are scaled using StandardScaler, i.e., for each sample, its z-scores are calculated with respect to all numerical (not one-hot encoded) features. The motivation behind this transformation is that some learning algorithms, such as K-nearest-neighbors, require a uniform scale. However, through repeated and varying execution of the learning algorithms, it has been found that it makes little difference whether the numerical features are scaled or not. Therefore this step is not used during code execution.
→ The result of this preprocessing step (if used) is that all numerical (not one-hot encoded) features of the training data are scaled by StandardScaler/z-score method, resulting in sharing the same scale for all feature

Learning Algorithms

The preprocessed data are used to estimate house prices. Two experiments are conducted for this purpose. In experiment-1, only univariate linear regression models are trained. In experiment-2 multivariate models are trained. For experiment-2, base learning algorithms (linear regression, K-nearest-neighbors) and one ensemble learning algorithm (random forest) are used. Named learning algorithms are used for this work because they can estimate continuous values, which means house prices in this case.
I choose these learning algorithms because they vary in their nature (base vs. ensemble learning algorithms, parametric vs. non-parametric learning algorithms) but try to solve the same problem. Using this mix of different learning algorithms, I hope to achieve good results.

(1) Linear regression is a parametric learning algorithm, which optimizes a function of the form

$$f(X) = b + \sum_{i=0}^{d} w_i x_i ,$$

where

- $f(X)$ is the estimate of the y variable (Price),
- X is are input features,
- d is the number of features,
- x_i is the i-th feature,
- w_i is the associated weight to feature x_i, and
- b is a bias term.

The learning algorithm gradually adjusts the weights w_i to make an optimal estimate. After the training phase, the weights are fixed. As the value of a weight w_i gets farther away from 0, the associated feature x_i has more influence in the estimation. For a sample's y-estimation, its features are given as input to the function, and then the calculation is performed.

(2) K-nearest-neighbors algorithm can be used for both classification and regression. It is a non-parametric algorithm. To estimate the y-variable (Price) for a sample, its k nearest neighbors are determined with respect to the used features using a distance metric. Then, from these k neighbors, the mean value of their y-variables (Price) is calculated. This value is the estimation for the sample.

(3) Random Forest is a non-parametric ensemble learning algorithm that uses decision trees as a base learning algorithm. Random Forests are used for both classification and regression. Multiple decision trees form a Random Forest. Each decision tree is created based on a subset of the training dataset. Each decision tree computes an estimate for a sample's y-estimate (price estimate). As a result, the estimation of the random forest is the mean of all predictions of the decision trees.

Experiment-1 – Univariate Lineare Regression Models

In the first experiment, five univariate linear regression models are trained. Each model is trained based on only one feature. Features used are those that have a high correlation factor with respect to the y-variable (Price), namely *Living_space* (0.44) *Furnishing_quality_luxus* (0.32) *Bathrooms* (0.27) *Type_Villa* (0.27) *Rooms* (0.25). Models are trained with the training data. After completion of the training phase, the performance for each model is determined using the Root-Mean-Squared-Error (RMSE) loss function. RMSE is calculated for both the training data and the test data. Both values are compared with each other.

Results of Experiment-1

Figure 3 - Univariate Linear Regression Models (Experiment-1)

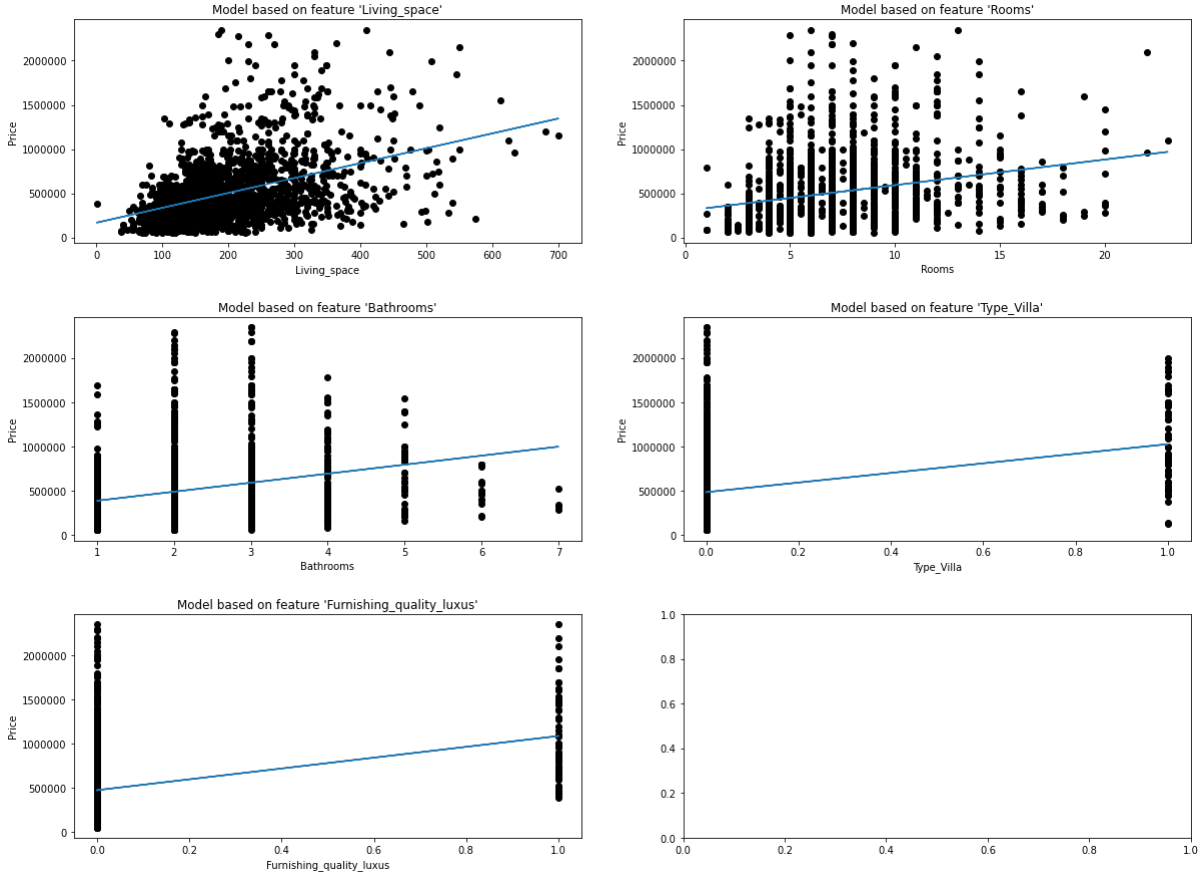

Figure 3 shows several graphs, each representing a different univariate linear regression model (blue line). The y-axis represents the y-variable (price), and the x-axis is the feature the model is trained with.

Figure 3 displays variously trained univariate linear regression models, each in a subplot. In summary, no strong linear relationship between x-variable (Feature) and y-variable (Price) can be found in any of the models. Data points are too scattered for a univariate linear regression model to predict a correct price.

Figure 4 – Root-Mean-Squared-Errors (Experiment-1)

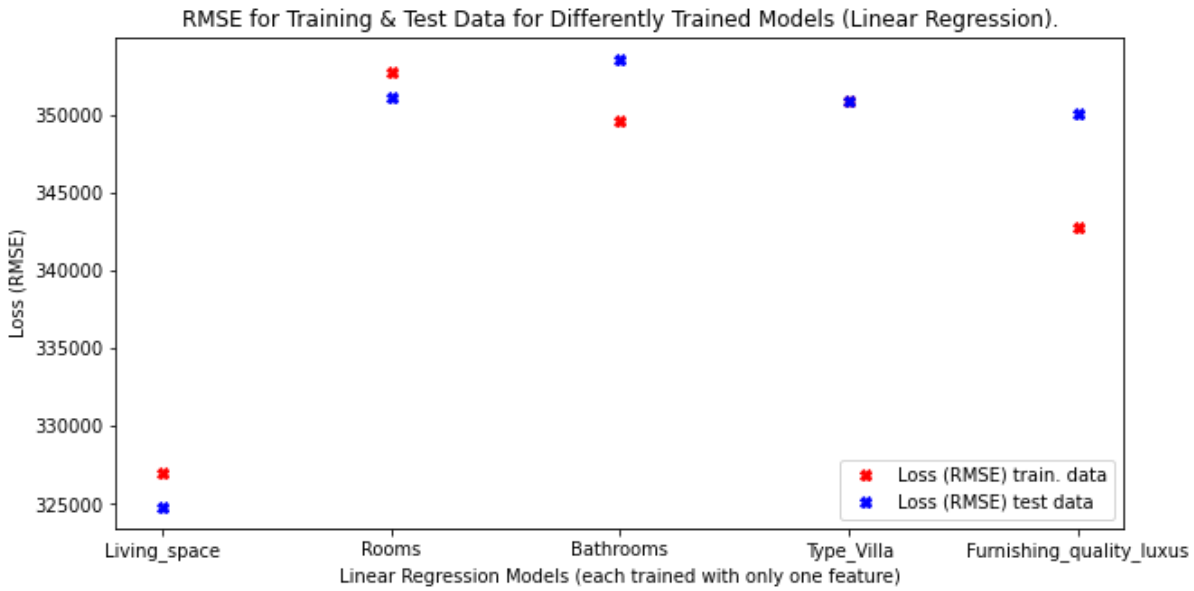

*Figure 4 visualizes the loss/*RMSE *(y-axis) for the different univariate linear regression models (x-axis). Red coded is the* RMSE *for the training data set; blue coded is the* RMSE *for the test data set.*

Figure 4 shows for each model (x-axis) the corresponding RMSE (y-axis) for training (red) & test data (blue). Overall, RMSE values are in the range of [325 000 - 355 000]. It can be clearly seen that RMSE for training data and RMSE for test data are very close to each other in all models. The best performing model was the model trained on the *Living_space* feature. As a reminder, this feature has the highest correlation factor with the y-variable (Price).

The results from experiment-1 conclude that none of the models can predict reliable prices based on a single feature. However, the best-performing model was the one that was trained based on the *Living_space* feature, which has RMSE-values around 325 000.

Experiment 2 – Multivariate Learning Algorithms

In Experiment-2, two base learning algorithms (Linear Regression, K-nearest-neighbor), as well as one ensemble learning algorithm (Random Forest), are used to estimate house prices. Each of the three learning algorithms is used to train models. In addition, multiple models are created for each learning algorithm. These models differ in that they train on different numbers of features. The number of features is either 1, 2, 3, 4, 5 (features with the highest correlation factor to the y-variable (Price)) or 170 (all features).

Results of Experiment-2

Figure 5 - Performance of Learning Algorithms (Experiment 2)

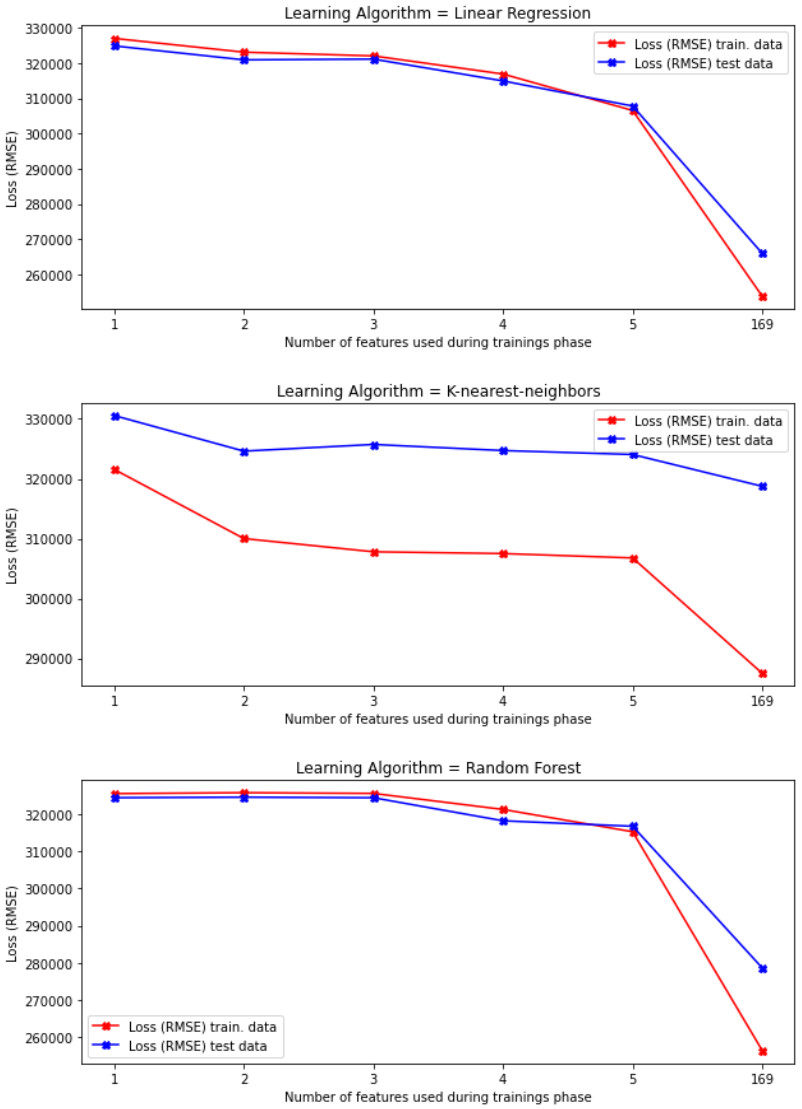

Figure 5 visualizes in three graphs the performance as RMSE *(y-axis) of different learning algorithms depending on how many features are used during the training phase (x-axis). The upper graph belongs to Linear Regression, the middle graph to K-nearest-neighbors, and the lower graph to Random Forest. Red coded is the* RMSE *for the training data set; blue coded is the* RMSE *for the test data.*

Figure 5 shows how the different learning algorithms' loss/performance (RMSE) behaves depending on the number of features used. The upper chart shows performance for linear regression, the middle chart for K-nearest-neighbors, lower graph for random forest.

In all learning algorithms, the RMSE for test data (coded in red) is higher than the RMSE for training data (coded in blue). However, the overall RMSE is approximately between 260000-335000, which is high. The RMSE values decrease with the increasing number of features used during the training phase for all learning algorithms. Accordingly, the lowest RMSE values for all learning algorithms are found in the models that use all 170 features.

K-nearest-neighbors performed worst when using all 170 features (RMSE for training data ~290 000, RMSE for test data ~325 000), followed by K-nearest-neighbors (RMSE for training data ~300000, for test data ~320000). Random Forest achieves the second-best performance (RMSE for training data ~260 000, for test data ~280 000), and the best by Linear Regression (RMSE for training data ~260 000, for test data ~270 000).

For Linear Regression as well as Random Forest learning algorithm, a large decrease in RMSE-values can be seen as soon as 170 features are used instead of 1-5 features. It is also noticeable that, in contrast to K-nearest-neighbors, RMSE for training and test data always have very little difference from each other.

The results from experiment-2 conclude that all models have a high loss (RMSE). The model with the lowest loss values uses Multiple Linear Regression and trains on all 170 features. Here, the loss for training & test data is almost identical (RMSE for training data ~260 000, for test data ~270 000) . However, even for this model, the RMSE values are so high that it cannot make reliable price predictions.

Final Result

Comparing Experiment-1 and Experiment-2, the model using Multiple Linear Regression using all 170 features during the training phase achieves the lowest loss. For training and test data, the RMSE is ~260 000 and ~270 000, respectively.

Figure 6 – Comparison between Ground Truth and Predictions

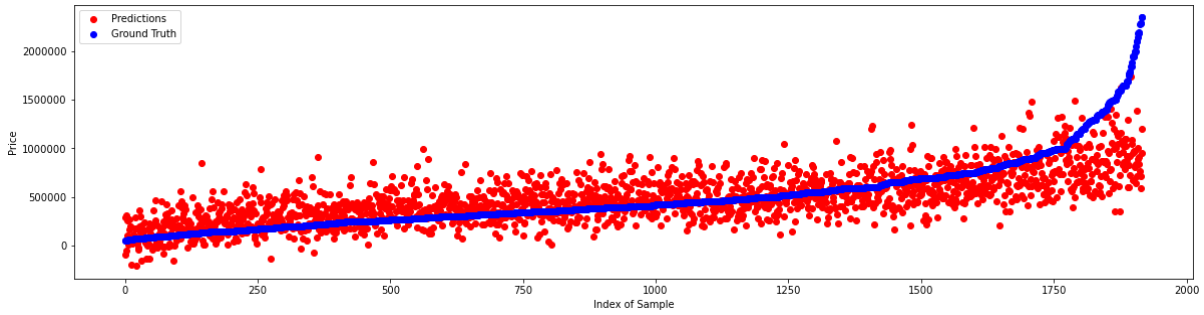

Figure 6 shows for each sample (x-axis) its corresponding price (y-axis). There is the actual price (blue) and a prediction (red) for each sample. Predictions are obtained by Multiple *Linear Regression trained on 170 features. The actual prices of the test data (blue) are sorted in ascending order.*

Finally, to visualize the performance of the best model (Multiple Linear Regression trained on 170 features) in a different form, Figure 6 is used. Figure 6 visualizes how well the model performs on the test data, i. e. the deviation between estimated (red) and actual (blue) prices. The x-axis represents the samples, and the y-axis the prices.

The ground truth (blue) is sorted in ascending order. Predictions follow this upward trend, although only very roughly. Particularly visible in the graph is the large spread of the predictions around the actual prices, reflecting the high RMSE for the test data (~270 000). Especially for high prices (> 10 000 000€) bad predictions are made. Another negative characteristic of the model is that it predicts negative prices for a small number of samples, although negative prices for houses do not exist in the ground truth.

This work aims to identify patterns from the used dataset to determine prices for German houses. In the end, the machine learning algorithms did not find any reliable patterns for accurate price predictions. In fact, the best model (Multiple Linear Regression trained on 170 features) only makes very rough price predictions that are on average 270 000€ (RMSE) apart. Considering that the average price for German houses is in the interval [300 000€, 1 000 000€] depending on the place of residence, this RMSE value is far from reliable. Nevertheless, the rough trend for prices is captured. Accordingly, the deployment of models is only of limited use. For a decision making process, the predictions of the model have only little reliability.