Erasmus University Rotterdam

Erasmus School of Economics

Master Thesis Econometrics & Management Science

Program: Business Analytics & Quantatative Marketing

Predicting prices of newly built houses using various machine learning methods

---

Author:

Luuk Jonkergouw

Student number:

431511

Thesis supervisor:

dr. Paul Bouman

Second assessor:

dr. Olga Kuryatnikova

Company supervisor:

Tijn van Dongen

Abstract

Neural Network, XGBoost, and Heterogeneous Feature Augmentation are employed to predict newly built house prices. Out of these three, XGBoost combined with Bayesian Hyperparameter Optimization demonstrated the best performance with a Mean Average Percentage Error equal to 2.502.

Date final version:

December 10, 2021

Contents

1 Introduction

The problem's relevance and description are detailed in this section, along with the three research questions.

1.1 Veneficus

This thesis is written at Veneficus, a consultancy firm that aims to help its clients make better decisions by providing data-driven insights. The firm, which was founded in 2007 by three econometricians, currently employs about 35 people. The majority of Veneficus' employees have an econometrics background. The four sectors in which Veneficus operates are retail, real estate, insurance, and the public sector (Veneficus, 2021). The firm offers solutions in traditional consultancy project format, but they also offer platform-based solutions. Veneficus has developed the Houzr platform for the real estate sector, allowing residential real estate developers to model and optimize new building projects more effectively. Machine learning algorithms are utilized in the prediction of property values based on home characteristics and location factors. This thesis aims to improve the underlying model that is used to forecast property prices.

1.2 Relevance

Real estate valuation is the process of estimating the price of real estate, the accuracy of which is essential for a multitude of reasons. For mortgage lenders, the valuation can make or break the sale of a house, depending on how it compares to the offer price. Furthermore, the banks use the appraisal to calculate the mortgage amount for their customers, which needs to be accurately given. Additionally, the valuation of newly built real estate is relevant for project developers to estimate the project's profitability.

Moreover, transactions in real estate often require appraisals as every property is unique, given its characteristics, condition, and location. Many factors determine the price of a house. For example, suppose five terrace houses lie next to each other, all of which are of the same type and thus have identical characteristics. The leftmost house is situated next to a busy road, and the rightmost house is adjacent to a quiet canal. Although the functionalities of

these two houses are the same, the prices will likely differ, as the demand for the rightmost house will be higher since it is positioned next to the canal. Veneficus currently employs a model for real estate valuation that values these two houses at the same price, which does not correspond to reality, as illustrated by the aforementioned example.

Several methods can be employed for predicting house prices. Park and Bae (2015) state that there are two main research trends, namely using hedonic-based regression approaches, such as the Hedonic Pricing Model (HPM), or machine learning methods, such as a feedforward Neural Network, to develop real estate price-prediction models. Literature related to these methods is discussed in Section 2.

Machine learning methods require careful tuning of their hyperparameters. In this thesis, Bayesian Hyperparameter Optimization is proposed, which is explained in Section 4.5.

1.3 Problem Description

The data used in this thesis consists of existing and newly built house data. Predicting house prices, specifically the square meter price, differs between existing and newly built houses. From now on, the price per square meter will be referred to as the m^2 price. Additionally, house markets differ from apartments markets and can differ per city, meaning that market heterogeneity exists. Machine learning methods presume that the training and test data have equivalent distributed features. However, this is not the case with market heterogeneity. It would imply that the existing house data cannot effectively predict m^2 prices of newly built houses in machine learning. Nonetheless, this data may contain helpful information. Ideally, one would like to develop a model to account for this market heterogeneity and exploit this information.

Another difficulty that arises is that the data of the newly built houses have a different feature space compared to the existing house data. This means that the data of newly built houses contains additional features about the house's location, such as whether the backyard points towards the sun. These extra features are used to predict newly constructed houses on a building number level, making a separate prediction for each property. Having different feature spaces causes problems when employing traditional machine learning techniques, as the input data must have the same dimension.

In this thesis, the focus is on the valuation of newly built houses. This valuation will depend on all the house characteristics and idiosyncratic factors that vary for similar types of houses. To perform the valuation, newly built and existing house data are used, to learn from the relations present in it. This would preferably result in better model performance. Additionally, market heterogeneity and the difference in feature space need to be accounted for and are discussed in Section 4.

1.4 Research Questions

As mentioned before, Veneficus uses a model that values the same types of houses at the same price. Therefore, a new model is constructed in this thesis that accounts for this issue. Due to market heterogeneity, the assumption that the train and test set features follow the same distribution does not hold. Standard machine learning methods cannot deal with differences in feature spaces in the data, so one needs to account for these issues, which is done using transfer learning. With transfer learning, the existing house data can be used to predict the m^2 price of the newly built houses more accurately. Currently, Veneficus uses grid-search for tuning its models' hyperparameters, which is an inefficient method. Therefore, it is valuable to analyze a more advanced hyperparameter tuning method. For Veneficus, it is crucial to provide a clear interpretation of the results of their models so that their clients can understand what drives these results. From this, the following research questions can be derived:

- Is it possible to estimate the m^2 price of a newly constructed house (i.e., on a building number level) accurately using transfer learning while dealing with market heterogeneity and differences in feature space, compared to baseline methods?
- How does adopting Bayesian Hyperparameter Optimization affect model performance, and will it result in a more efficient algorithm in terms of running time?
- How can one provide an interpretation of the model results? What are the features most influential in the valuation?

2 Related work

In this section, literature related to this research is discussed. Firstly, several real estate valuation methods are described. After this, Bayesian Hyperparameter Optimization is discussed. In the third paragraph, transfer learning applications are given, and finally, literature related to the interpretation of machine learning models is discussed.

2.1 Real Estate Valuation Methods

Several methods can be used for real estate valuation. The most traditional approach is the Hedonic Pricing Model (Sirmans et al., 2005). This method is ubiquitous in existing research and considers several internal and external features in the estimation of the price of a house through a regression (Miller Jr & Markosyan, 2003). In the real estate setting, the internal features are the different characteristics of a house, such as the size, the number of rooms, or the construction year. The external features relate to the neighborhood or environment of the property, such as whether there is a school nearby or the average income per neighborhood. The internal and external features together define the hedonic price of the property.

Pagourtzi et al. (2003) provide a brief overview of some widely used methods in real estate valuation. They state that the above-mentioned HPM method is relatively straightforward and that deep learning methods such as a Neural Network (NN) belong to the more advanced methods. Peterson and Flanagan (2009) state that relative to hedonic pricing models, Neural Networks generate significantly lower dollar pricing errors and have greater out-of-sample pricing precision. In addition, they extrapolate better from more volatile pricing environments when predicting house prices, as multiple layered Neural Networks can model complex non-linearities.

Selim (2011) examined house prices in Turkey and compared a hedonic property valuation method with a NN. The authors chose this method as it can identify potential non-linear patterns. The study demonstrated that the NN resulted in better prediction performance relative to the hedonic regression.

Limsombunchai (2004) also conducted a comparison between the predictive power of a hedonic price model and an NN, using a sample of 200 houses in New Zealand. The out-

of-sample forecasts of the hedonic pricing models were poorer than the NN models, so they did not outperform them. They speculate that the poor performance of the hedonic price models is due to the non-linear relationship between features and house prices. Furthermore, hidden layers of a NN are commonly referred to as 'black-box,' which in their opinion is an unfavorable characteristic since this may cause problems regarding the interpretability of the model.

Furthermore, other machine learning techniques are used for real estate appraisal. Zhao et al. (2019) use deep learning combined with eXtreme Gradient Boosting (XGBoost) for making price predictions in real estate, as they analyze historical sale records to make a house price prediction. XGBoost has become a prevalent machine learning method over the last few years and is recognized for its good performance. The research demonstrates that replacing the last output layer with XGBoost improves the performance of house price prediction accuracy. According to this literature, Neural Networks and XGBoost models are more efficient than traditional methods like HPM when predicting real estate prices.

However, not all studies point in the same direction. Worzala et al. (1995) doubt whether Neural Networks are reliable when they are used for real estate appraisal. The authors claim that even though the same data is utilized, the results from various NN software packages might differ. Also, regression models are not always outperformed by NN models. Lenk et al. (1997) found that the hedonic models and NN models showed similar results, which contradicts some previous research that argues for the supremacy of Neural Networks. It may be dependent on the available data which method shows the best performance in predicting house prices.

2.2 Bayesian Hyperparameter Optimization

Hyperparameters in machine learning must be pre-set before the learning process begins, and they must be tuned carefully (Snoek et al., 2012). Zhang et al. (2020) declare that grid-search is commonly used to tune them but that the number of model evaluations increases exponentially as more hyperparameters are taken into account. Zhang et al. (2020) recommend Bayesian Hyperparameter Optimization (BHO) as an alternative for grid-search. BHO uses previous model evaluations to find the best set of hyperparameters that minimizes

the objective function. This optimization approach uses fewer iterations and results in better performance of the models, as it iteratively focuses on the hyperparameter spaces where previous sets of hyperparameters yielded good performance (Bergstra et al., 2013). Veneficus mostly uses grid-search to tune the models' hyperparameters. BHO is implemented in this thesis to tune these hyperparameters.

2.3 Transfer Learning

Pan and Yang (2009) emphasize that the training and test data must span the same feature space and have the same distribution, which is a crucial assumption in many machine learning applications. However, this assumption may not hold in many real-world examples. In this research, one deals with market heterogeneity and differences in feature spaces of the data. Transfer learning can address this problem, as it allows the feature spaces and the data distributions to differ. The analysis of TL is inspired by the fact that people can apply previously acquired knowledge to solve new problems quicker or more precisely. TL aims to take information from one or more source tasks and apply it to a target task. In this thesis, the source data is the existing house data, while the target data is the newly built house data. Ideally, one wants to use relevant information from the former data to make more accurate predictions on the latter.

There are different types of transfer learning, such as instance-based approaches and feature-based approaches (Weiss et al., 2016). In the instance-based method, each training instance's effect in the target data is estimated using a pre-defined model learned on the source data. Based on this influence, it is determined whether a training sample is used or not in the target data. The general idea is thus to take instances from the source data that are relatively close to the target domain data and use these to make predictions on the target data. Feature-based approaches aim to minimize the difference in the domain between the source and target data. The instance-based approach can be used in situations with market heterogeneity. The feature-based approach is suitable in situations where there is a difference between feature spaces in the data. This thesis must find a method that can deal with both market heterogeneity and different feature spaces.

Desautels et al. (2017) conducted a study using TL where they predicted in-hospital

mortality. The researchers utilized a publicly available dataset as their source material, including over 50.000 critical care unit stays from 2001 to 2012. The target data consists of 110.000 inpatient contacts from a San Francisco hospital. The wards and departments from which these datasets are derived differ in a variety of ways. While retaining high predictive accuracy, the researchers emphasize that the TL approach significantly decreases the target data collection workload. They used weighted combinations of source and target data to train the classifier. As a result, the performance of the classifier based on the target data was significantly enhanced.

Fawaz et al. (2018) use transfer learning for a time series task with deep Neural Networks. The researchers performed experiments with 85 datasets to investigate the potential of TL. They pre-trained a model on each dataset and then fine-tuned it on the other datasets, yielding 7.140 different deep Neural Networks. They note that TL can both improve or degrade the model predictions, depending on the dataset. Using inter-dataset similarities, they designed a method that chooses the best source dataset for a target dataset, which improves accuracy on 84% of the datasets.

2.4 Model Interpretation

Lundberg and Lee (2017) state that understanding why a model makes specific predictions is in many applications equally crucial as the prediction's accuracy. The highest accuracy for large modern datasets is often achieved by complex models that are hard to interpret, such as machine learning models. This phenomenon creates a trade-off between accuracy and interpretability. They present a framework that can be used for interpreting predictions called SHAP (SHapley Additive exPlanations). This framework assigns a value to each feature used in a particular model, which describes its importance. These values measure the change in the prediction of the expected model conditional on that feature. In this way, predictions from complex models can be related to human intuition. These SHAP values are exploited in this thesis as complex deep learning models are used, and the outcomes of these models should be interpretable for the clients of Veneficus.

3 Data Exploration

This section describes the data that is used in this thesis and how the data is collected. Moreover, it is described how outliers are treated, and a dependent variable analysis is conducted. Finally, correlations between the m^2 price and some independent variables are presented.

3.1 Data Description

The primary data that is used in this research consists of existing and newly built house data. The existing house data consists of all data that is not from newly built house projects. The data ranges from 2019 up until the present. The existing house data and the newly built house data consist of approximately 154.000 observations and 14.000 observations, respectably. This data is scraped from several websites. The existing house data is scraped from http://www.jaap.nl, and the newly built house data is scraped from:

- http://www.nieuwbouw-nederland.nl/gemeenten;
- http://www.nieuwbouw-nuenenwest.nl;
- http://www.am.nl;
- http://woningaanbod.heijmans.com;
- http://www.vanwanrooij.nl;
- http://www.vorm.nl;
- http://www.bergenopzoom.hetnieuwbouwplatform.nl;
- http://www.{amsterdam, utrecht, breda, groningen, meppel}woont.nl.

However, some observations from both datasets are not helpful, so the data must be filtered first. The two datasets consist of various types of houses, with all their respective characteristics. These include, for example, the m^2 price, the size of the floor space, and the number of bedrooms. The newly built house data is enriched with extra features of the

houses or apartments that can not be scraped accurately. For that reason, these features are added manually to the data. These include, for example, whether a house has a backyard and in which direction it points, or whether an apartment has a balcony or a terrace, and how many square meters it is. Other data used for both the existing and newly built house data is neighborhood data obtained from Centraal Bureau Statistiek (CBS). CBS is an office that keeps track of disparate kinds of statistics in the Netherlands. The CBS data used includes all characteristics of a particular neighborhood, such as a school, a sports club, or a supermarket nearby the house or apartment. Hereafter, feature engineering is applied to create dummies for the house type and whether the city is Amsterdam.

Moreover, based on the scraped date, all house m^2 prices are adjusted with the quarterly house inflation rate per province. The data mentioned above is available in the SQL database of Veneficus. An overview of the features that are available for the existing and newly built house data is presented in Table 1 in the Appendix. The data pipeline for the existing and newly built house data is shown in Figure 1:

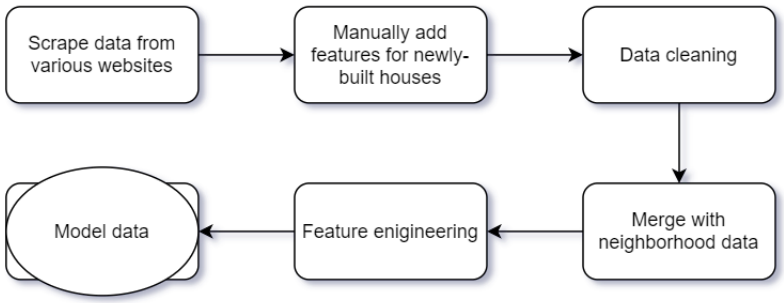


Figure 1: Data pipeline for existing and newly built house data

3.2 Outliers

For outlier detection, the lowest or highest quantiles are removed from the data for four crucial features, which significantly influence the employed models. The following criteria are used to remove outliers:

- The latitude should be between 50.7 and 53.6
- The longitude should be between 3.09 and 7.3

- The living area should be between 25 and 250
- The price should be between 75.000 and 1.500.000

Table 1 shows the number of observations that are left after applying these criteria:

Table 1: Remaining observations after data cleaning for both existing and newly built house data.

	Existing	Newly built
Houses	154.310	11.247
Features	23	37

3.3 Dependent Variable Analysis

This thesis aims to predict the m^2 price of newly built houses, implying that the m^2 price is the dependent variable. Figure 2 displays the density plots of the m^2 price for both the existing and newly built house data. It is observed that in general, the newly built houses have a higher price per square meter.

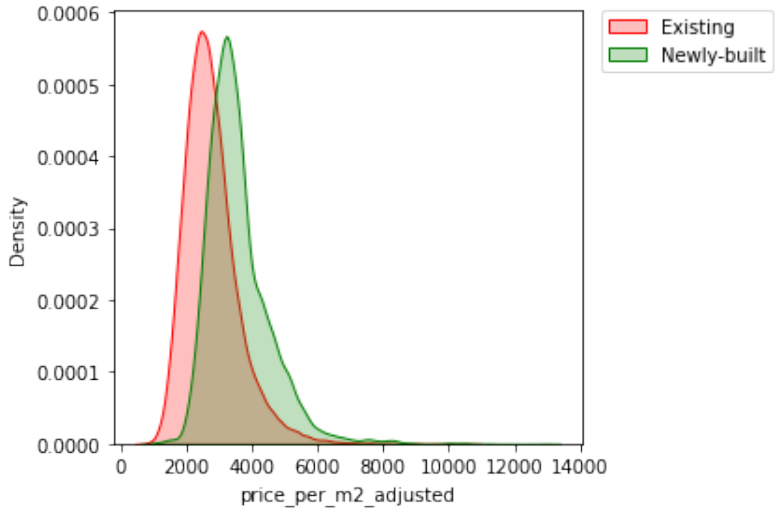


Figure 2: Density plots of the m^2 price for existing (red) and newly built (green) house data.

The data distributions of the m^2 price seem to differ between the existing and the newly built house data. Table 2 highlights the descriptive statistics for the two datasets. The kurtosis indicates how tall and sharp the central peak is in the m^2 pricing distribution. Furthermore, the skewness indicates how much the data distribution is symmetrical, where

the density plots in Figure 2 suggest that the data is skewed to the right. Table 2 shows that the skewness of the existing house data is 1.4, and 1.74 for the newly built house data, which confirms that the data is skewed to the right, as both are different from zero. One may consider a data transformation, such as taking the natural logarithm, which results in a more symmetrical distribution of the dependent variable.

Furthermore, one notices that the minimum and the maximum m^2 prices are higher for newly built houses than existing ones, as is the case for the mean, median, and mode. The standard deviation is greater for the newly built houses, which can be elucidated in Table 1. With 154.310 observations, the dataset of existing houses is significantly more extensive relative to the 11.247 observations of the newly built houses. Since the spread of the observations is narrower for larger samples, the standard deviation of the sample mean decreases.

Table 2: Descriptive statistics for the existing and newly built house data.

	Existing	Newly built
Mean	2.768	3.579
Standard deviation	850	980
Median	2.641	3.379
Mode	2.500	3.000
Skewness	1.40	1.74
Kurtosis	4.38	6.20
Mimimum	670	705
Maximum	10.761	12.987

Moreover, the correlation between the m^2 price and the independent features is calculated. As these features seem to influence the price of the house, it is expected that these features are essential drivers for the model predictions, which is discussed in Section 5.4. The correlations between some crucial features and the m^2 price are shown in Table 3. Noticeably, the sign of the correlations is the same for the existing and the newly built house data. However, the magnitude of the correlations differs. 'gem_woningwaarde_2019' and 'inkomen_gem_per_inwoner_2017' are indicators for the average house price in 2019 and the

average income in 2017. These features seem to significantly impact the m^2 price, as the magnitude of its correlations is the largest for both the existing and newly built house data.

Table 3: Correlations between the m^2 price of the house and some crucial features for the existing and newly built house data.

Feature	Existing	Newly built
latitude	0.04	0.15
longitude	-0.30	-0.26
property_type_vrijstaand	0.17	0.23
kaveloppervlakte_m2	0.15	0.26
afstandtotbelangrijkoverstapstation	-0.19	-0.35
adressendichtheid_2019	0.20	0.18
bevolkingsdichtheid_2019	0.17	0.04
dummy_amsterdam	0.19	0.37
gem_woningwaarde_2019	0.61	0.43
inkomen_gem_per_inwoner_2017	0.51	0.48
prc_oppervlakte_water_2019	0.10	0.21
distance_shoreline	-0.09	-0.31

The main differences between the existing and newly built house data are reflected in the m^2 price and the number of features. It is expected that data containing more features has more predictive power. Transfer learning may be used to create a model that predicts prices using both datasets, which is further discussed in Section 4.8. The differences in m^2 price can be explained by the construction year of the data. As mentioned in Section 3.1, the newly built data is scraped from 2019 until the present, which means that this data spans a relatively short period. In the past, many circumstances influenced the housing market, such as the Second World War creating a decline in house prices. Figure 3 shows the average m^2 price of houses from 1920 until 2021, and observably, the m^2 price has fluctuated over the years. Moreover, around 1960, many new houses were built, resulting in a significant dip in house prices, but over the last 40 years, house prices upsurged tremendously. However, a dip in house prices was noticed around 2010 as a result of the Great Recession. The differing construction years in the two datasets can therefore be an explanation for the difference in

the density plots in Figure 2.

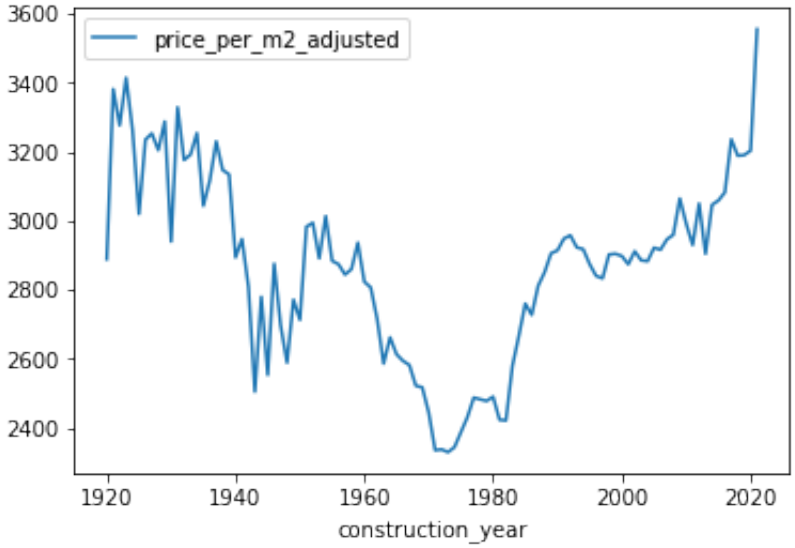


Figure 3: The average m^2 price of houses in the period 1920-2021.

4 Methodology

In this section, the methodology is discussed that answers the research questions. Firstly, the generalization performance is deliberated. Secondly, model selection, regression and classification approach, and scaling, are debated. Hereafter, Bayesian Hyperparameter Optimization is explained. The following section describes how Neural Networks, boosting methods, transfer learning, and Heterogeneous Feature Augmentation work and how these methods are implemented. Lastly, SHAP values are discussed.

4.1 Generalization Performance

The generalization performance of the models in the regression context is measured using the Mean Squared Error (MSE) and Mean Average Percentage Error (MAPE). These metrics are defined as follows:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}\left(Y_i - \hat{Y}_i\right)^2,$$

$$MAPE = \frac{1}{n} \sum_{i=1}^{n} \left| \frac{Y_i - \hat{Y}_i}{Y_i} \right|,$$

where n stands for the number of observations, Y_i for the actual m^2 price of the property, and $\hat{Y}_i$ for the predicted m^2 price of the property. The Mean Squared Error (MSE) measures the average squared difference between the actual and the predicted prices. The MAPE measures the average percentage absolute price error. When the MSE is minimized, observations with a significant error due to the square are given relatively more weight. Both metrics are widely used and are desired to be as small as possible (Hyndman et al., 2006). The MSE is scale-dependent, so it does not make sense to compare accuracies on different scales. Furthermore, from the business perspective, MAPE is popular, as percentage errors are easy to interpret.

For the classification context, the generalization performance is measured using the accuracy and the macro F_1-score. Recall that accuracy is defined as the number of correct predictions divided by the total number of predictions. Furthermore, the F_1 score for class i is defined as:

$$F_{1i} = \frac{2 \cdot \text{recall}_i \cdot \text{precision}_i}{\text{recall}_i + \text{precision}_i}.$$

The macro-F_1-score is the average of the F_1 scores over all K classes as $\frac{1}{K} \sum_{i=1}^{K} F_{1i}$. The macro-$F_1$-score is utilized, as predicting each class is considered equally important.

4.2 Model Selection

As the NN and the XGBoost model cannot deal with different feature spaces, the existing and newly built house data are merged into one dataset that contains the features of the existing house data and hence does not contain the extra features that are used in the TL model. This dataset is divided into a training set, containing 80% of the data and a test set, containing 20% of the data. Observations from the same building project in the training and the test data may cause a prediction bias. This is the case since these houses have similar characteristics and prices and are therefore relatively easy to predict in the test set if the models are trained on the train set. Therefore, for constructing the training and test set, it is ensured that observations from the same building project are either in the train set or in the test set. In order to select the best sets of hyperparameters for the models,

five-fold cross-validation is used. In k-fold cross-validation, the training data is divided in k equally sized folds. Following that, k rounds of training and validation are carried out, with each iteration holding out a different fold of the data for validation and the remaining $k-1$ folds being utilized for learning (Refaeilzadeh et al., 2009). Hence, five different folds of the training data are used as a validation set for hyperparameter tuning of the models. The tuning of the hyperparameters is carried out using Bayesian Hyperparameter Optimization and Grid-Search as baseline method, these methods are discussed in Section 4.5.

For the heterogeneous transfer learning model, which is discussed in Section 4.8.2, it is preferable to exploit all observations of the existing house data (source data) as training data, along with 80% of newly built house data (target data). Hence, only 20% of the newly built house data is used as test data. Here, it is also the case that the test and training data do not contain observations from the same building project. Figure 4 shows a visualisation of the model selection process, based on the complete data and the subset of the data. The generalization performance is measured on the test set to prevent bias for the regression and classification approaches. However, it turned out that using all the observations in the heterogeneous transfer learning model was not computationally feasible, the reason for this non-feasibility is clarified in Section 4.8.2. Therefore, the following sub-sample of the data is taken for the heterogeneous transfer learning method: the source data consists of 8.000 observations, the target data of 4000 observations, and the test data of 300 observations. This method is compared with the NN and the XGBoost model using the same sub-sample of the data.

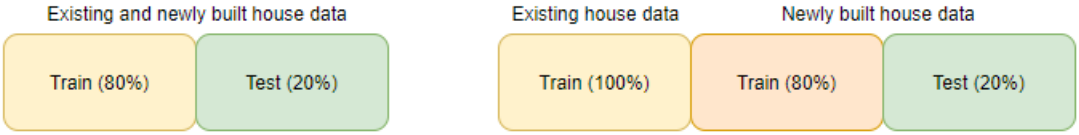


Figure 4: Model selection process for Neural Network and XGBoost, using the complete dataset (left), and for Heterogeneous Feature Augmentation, Neural Network, and XGBoost, using a subset of the data (right).

4.3 Regression and Classification Approach

This thesis aims to predict newly built house prices, which corresponds to a regression problem, as a numerical value is predicted. This prediction is carried out by adopting a NN and

an XGBoost model, described in detail in Sections 4.6 and 4.7, respectively. These methods' hyperparameters are tuned using BHO and Grid-Search, which are described in Section 4.5.

In Heterogeneous Feature Augmentation (HFA), which is described in detail in Section 4.8.2, Shi et al. (2010) propose a framework where HFA is applied in a classification context, and the global solution is found using Support Vector Classification (SVC). The authors state that HFA can be "readily incorporated into different methods, such as Support Vector Regression (SVR)". However, putting this method into practice in a regression environment was not as simple as described. After putting in much effort, the underlying mathematics turned out to be too complicated to derive. Therefore, HFA is utilized in a classification context. The m^2 prices of each observation are assigned to m ordered buckets, where the buckets are ordered, and bucket zero corresponds to the observations with the lowest m^2 price. Bucket m corresponds to the observations with the highest m^2 price. Hereafter, HFA is applied, and it is evaluated whether this method assigns the observations into its actual buckets, this process is executed using 5, 8, and 11 buckets. HFA is compared with NN and XGBoost in the classification context by looking at the accuracy and the micro F_1-score, which is done using a sub-sample of the data as described in Section 4.2. The hyperparameters of the NN and XGBoost model are tuned using BHO. Table 4 shows how the buckets are composed, which is based on the distribution of the m^2 price shown in Figure 2.

4.4 Scaling

In Section 3.3 it is concluded that the m^2 price of the houses is skewed to the right. Therefore, this variable's natural logarithm is taken to make the data distribution more symmetrical in the regression context. Furthermore, in the regression and classification context, the data is normalized, as Al Shalabi et al. (2006) mentioned that normalizing the data improves model performance for machine learning methods. They tested two normalization methods which are Z-score normalization and min-max normalization. The min-max method demonstrated the best performance, so this method is used in this thesis for scaling the data. Each feature is scaled as follows so that every observation has values between one and zero:

$$\frac{x_i - \min(\mathbf{x})}{\max(\mathbf{x}) - \min(\mathbf{x})},$$

Table 4: Composition of the m buckets, with $m = 5, 8, 11$

	Five buckets	Eight buckets	Eleven buckets	bucket
m^2 price	(0, 1500]	(0, 1500]	(0, 1500]	1
	(1500, 2500]	(1500, 2000]	(1500, 1900]	2
	(2500, 3500]	(2000, 2500]	(1900, 2300]	3
	(3500, 4500]	(2500, 3000]	(2300, 2700]	4
	(4500, ∞]	(3000, 3500]	(2700, 3100]	5
		(3500, 4000]	(3100, 3500]	6
		(4000, 5000]	(3500, 3900]	7
		(5000, ∞]	(3900, 4300]	8
			(4300, 4700]	9
			(4700, 5500]	10
			(5500, ∞]	11

where x_i is a value of a feature vector $\mathbf{x}$ with length n. The scaling is executed for all p feature vectors for the NN, XGBoost, and HFA. However, XGBoost is unaffected by monotonic feature transformations, so establishing a split on one scale results in a similar split on the transformed scale (Meng et al., 2020). Thus, scaling has no effect for XGBoost, but the features of all methods are scaled for simplicity.

4.5 Bayesian Hyperparameter Optimization

The key goal of hyperparameter optimization is to find the best set of hyperparameters for which the highest generalization performance on the validation set is obtained. For optimization of the hyperparameters, the following objective value $f(x)$ is minimized on the validation set with respect to its hyperparameters:

$$x^\star = \arg\min_{x \in \mathcal{X}} f(x). \tag{1}$$

In this case, x denotes the set of hyperparameters while $\mathcal{X}$ denotes the hyperparameter space, and x^* indicates the hyperparameters for which the lowest mean squared error on the validation set is obtained. Grid search, which is currently exploited at Veneficus, is

a method for determining the best hyperparameters by experimenting with various sets of hyperparameters that the researcher specifies. However, the method is inefficient as it is not informed of the previous evaluations of the sets of hyperparameters. As a result, grid search is a tedious method as it attempts to try sets of hyperparameters with low generalization performance making it a time-consuming approach. On the other hand, BHO algorithms iteratively focus on hyperparameter spaces that produce strong results.

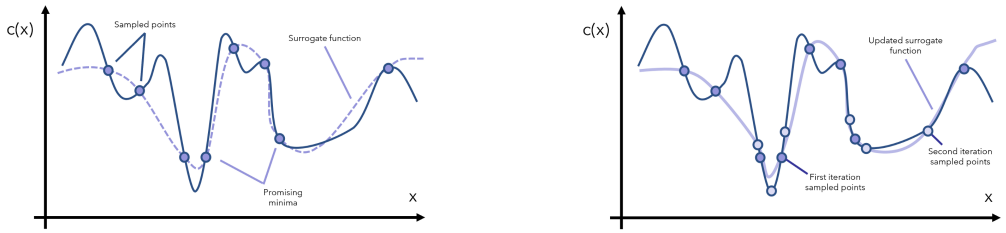
However, it may be the case that BHO focuses too much on promising regions. In contrast, the unpromising regions are left unexplored, where in theory, the global minimum may happen to lie there. However, as stated by Zhang et al. (2020), BHO produces promising results, and therefore, the hyperparameter tuning for the NN, the XGBoost model, and the HFA model is done using BHO. Most of the hyperparameter optimization algorithms minimize the objective function, therefore the MSE is provided to the algorithm. The sets of hyperparameters that provide the minimum MSE score on the validation set are selected. The `HyperOpt` package in `Python` is used to implement BHO.

In the regression approach in this thesis, the NN and the XGBoost models' hyperparameters are tuned using both Grid-Search and BHO. The hyperparameter spaces of BHO and Grid-Search contain the same extreme values per hyperparameter, but BHO contains all numbers in the span between these extremes, while Grid-Search contains sets of pre-selected values. The performance, the time, and the number of function evaluations are compared for both the hyperparameter optimization methods. These results are discussed in Section 5.3.

Evaluating the objective function $f(x)$ in (1) is computationally expensive. To evaluate the function, the algorithm must be trained using the hyperparameter set $x \in \mathcal{X}$. This means that the algorithm needs to be trained from scratch for evaluating each different set of hyperparameters. Therefore, BHO uses a surrogate function $\mathbf{\Omega}$ that is less costly to optimize relative to the original objective function $f(x)$. It is possible to represent $\mathbf{\Omega}$ as a probabilistic model defined as $\mathbf{\Omega} = p(y|x)$ with y the objective value and x the set of hyperparameters. Subsequently, the $\mathbf{\Omega}$ function is used to build the selection function $\mathbf{S}$ for finding the most promising collection of hyperparameters. This is accomplished by using the Sequential Model-Based Optimization (SMBO) algorithm (Bergstra et al., 2011). Moreover, the Tree-structured Parzen estimator serves as the surrogate function $\mathbf{S}$.

4.5.1 Sequential Model-Based Optimization

The SMBO algorithm builds the selection function $\mathbf{S}$ by using the initial selection function $\mathbf{\Omega_0}$, whereafter the set of hyperparameters x_i is picked that minimizes the selection function. Hereafter, the objective function value for iteration i is computed using the computationally costly objective function $f(x_i)$. It is expensive to compute the objective value with a fresh set of hyperparameters since it necessitates training the algorithm from scratch. Optimizing the selection function allows examining only the most promising sets of hyperparameters. At each iteration i, the generalization performance and the corresponding hyperparameters are stored, and $\mathbf{\Omega}$ is updated according to the posterior beliefs. Finally, the algorithm saves the values y and the hyperparameters x, and the collection of hyperparameters that yield the best results is chosen. A hypothetical example illustrates this process. Suppose that there is an objective function $c(x)$, given some input x that needs to be minimized. The true shape of this function is hidden from the optimizer, where a surrogate function is formed based on some sample points, shown in Figure 5a. Based on this surrogate function, points that are promising minima are identified, and hereafter, more points in the promising regions are sampled, and the surrogate function is updated accordingly, which is shown in Figure 5b.



(a) First iteration of the algorithm.

(b) Second iteration of the algorithm.

Figure 5: The objective function, denoted in blue, and the surrogate function, denoted in dashed and light-blue. The circled points denote the promising minima. The surrogate function is updated from the left graph to the right graph.

4.5.2 Tree-structured Parzen Estimator

The surrogate function $\mathbf{\Omega}$ is the probabilistic representation of the objective function f, and it needs to be computationally cheap. As mentioned above, this surrogate function is updated at each iteration with the posterior beliefs. Surrogate functions often utilized include Gaussian Processes, Random Forests, and the Tree Parzen Estimators (TPE). The latter is

used in this thesis, as suggested by Bergstra et al. (2011). This estimators models $p(y|x)$ by utilizing Bayes' law:

$$p(y|x) = \frac{p(x|y)p(y)}{p(x)}$$

In every iteration of the SMBO algorithm, the set of hyperparameters that minimizes $\mathbf{S}$ is evaluated using the objective function f. One of the most popular options for $\mathbf{S}$ is Expected Improvement and is used in this thesis (Bergstra et al., 2011). The SMBO algorithm places a greater emphasis on the hyperparameter space where previous hyperparameters demonstrated exemplary performance. Thus, the algorithm incorporates knowledge from earlier iterations and concentrates on the hyperparameter space with the highest potential.

4.6 Neural Network

The Universal Approximation Theorem states that a NN can approximate any continuous function with a sufficiently sizeable hidden layer (Hornik et al., 1989). Neural Networks can capture complex relationships in the data. For estimating house prices, a NN is employed since it can deal with potential non-linear patterns in the data. The NN together with BHO is implemented by combining the `Keras` and `HyperOpt` packages in `Python`. The following three subsections discuss the design that is chosen for the NN.

4.6.1 Architecture

Géron (2019) states that a feedforward NN can model almost every continuous function by using only one or two hidden layers. The feedforward NN architecture has an input, a hidden, and an output layer. In the input layer, the data is fed to the network. The number of neurons in the input layer equals the number of features that the input data contains. Weights are assigned to the input data by the hidden layer, which then delivers them to the output layer through activation functions. Moreover, by using an activation function, the output layer generates the final prediction. Figure 6 highlights an example of a feedforward NN design in a regression context, which contains one input layer with four neurons, one hidden layer with five neurons, and one output layer with one neuron. In practice, Géron (2019) emphasizes that using more layers and neurons than one requires, in combination

with regularization approaches, prevents the model from overfitting. Typically, the number of hidden layers ranges from one to two, and the number of neurons per hidden layer ranges from ten to a hundred. These are included in the hyperparameter space, which is shown in Table 5 (Géron, 2019).

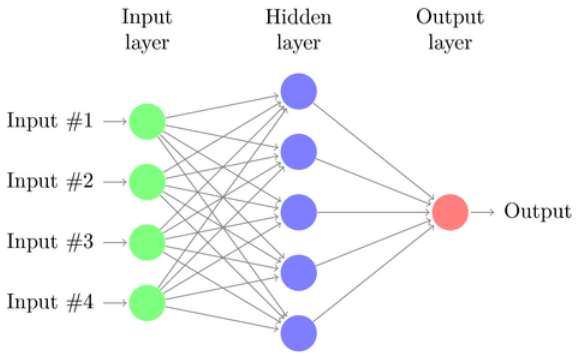


Figure 6: Architecture of a feedforward NN in a regression context.

4.6.2 Activation Functions

In a feedforward NN, activation functions are utilized in the hidden layer(s) and output layer. When the network is given input data $\mathbf{X} = X_1, ..., X_k$, with k neurons, the inputs are multiplied with the coefficients, and these serve as input for the first hidden layer:

$$a_j = \sum_{i=1}^{k} \beta_{ji}^{(1)} X_i + \beta_{j0}^{(1)},$$

where $\beta_{ji}^{(1)}$ is the coefficient from input neuron i to hidden neuron j, and $\beta_{j0}^{(1)}$ is the bias term. These inputs for the hidden layer a_j are transformed through the activation function h, after which the output z_j from the hidden layer is obtained:

$$z_j = h\left(a_j\right), \quad j = 1, \ldots, M,$$

where z_j is the output of the j-th neuron in the hidden layer and M the number of neutrons in the hidden layer. In the case of only one hidden layer, z_j is multiplied by the second set of coefficients and forms the output layer's input:

$$a_k = \sum_{j=1}^{M} \beta_{kj}^{(2)} z_j + \beta_{k0}^{(2)},$$

where a_k is the input for the output layer. In the final step, a_k is imputed in the output layer activation function, denoted by σ, where the final prediction is made:

$$\hat{y} = \sigma(a).$$

The output activation function has a direct link to the problem objective. For numerical prediction, the linear activation function is used in the output layer:

$$\sigma(a) = a.$$

For multiclass prediction, the softmax activation function is used in the output layer:

$$\sigma(a)_i = \frac{e^{a_i}}{\sum_{i-1}^{K} e^{a_j}} \text{ for } i = 1, \ldots, K,$$

where K stands for the number of classes.

The Rectified Linear Unit (ReLU) is regarded as an activation function for hidden layers since it is one of the most often employed activation functions. The ReLU activation function is defined as follows:

$$\mathrm{h}_{\mathrm{ReLU}}(a) = \max(0, a).$$

However, this function is not flawless, as it may suffer from the dying ReLU problem, which states that specific neurons always present a zero as output. This is because the weighted sum of the neuron inputs is negative for all occurrences in the training set. This means that Gradient Descent does not change the fact that the neurons keep outputting zeros since the ReLU's gradient is zero when its input is negative (Géron, 2019). Clevert et al. (2015) suggest the Exponential Linear Unit (ELU) activation function as a solution to this challenge, which shortened training time and demonstrated fewer mean squared errors in their studies when compared to the ReLU function. Moreover, the ELU function has a nonzero gradient for $a < 0$, which solves the dying ReLU problem. The ELU activation function is defined as follows:

$$\mathrm{h}_{\mathrm{ELU}}(z) = \begin{cases} \exp(a) - 1 & \text{if } a < 0; \\ a & \text{if } a \geq 0. \end{cases}$$

Both the ReLu and ELU activation functions are included in the hyperparameter space that is shown in Table 5.

4.6.3 Hyperparameters

Besides the architecture and the activation function hyperparameters, Géron (2019) states that the learning rate, the optimizer, and the batch size are the essential hyperparameters to tune. The NN is trained by optimizing the coefficients. To do so, the gradients of the coefficients need to be obtained. The backpropagation algorithm is used to calculate these gradients. The details of this algorithm can be found in the paper by Rumelhart et al. (1986).

Furthermore, Adaptive Moment Estimation (Adam) and Nesterov-accelerated Adaptive Moment Estimation (Nadam) are included in the hyperparameter space as optimizers, as suggested by Géron (2019). These algorithms are used for updating the coefficients and thus for the training of the NN. Each time the model weights are updated, the learning rate determines how much the model is altered in response to the predicted error. The batch size controls the accuracy of the estimate of the error gradient. See Table 5 for the list of hyperparameters of the NN that is optimized with BHO.

4.6.4 Regularization

The overfitting problem is a severe issue when training Neural Networks. When a model learns the details and noise in the training data to the point where it degrades the model's performance on new data, this is known as overfitting. One can reduce the risk of overfitting by applying regularization techniques. The dropout method is one of the most popular and effective regularization algorithms for training Neural Networks, according to Géron (2019). In the dropout method, each neuron has a chance p that it is ignored in a training step. This hyperparameter p needs to be tuned. Therefore, the dropout hyperparameter is also included in the hyperparameter space.

Furthermore, the early stopping method is used as another method to avoid overfitting. If the loss function does not decrease after ten iterations, the training of the algorithm is stopped. The entire hyperparameter space that is used for hyperparameter optimization is shown in Table 5.

Table 5: Hyperparameter space for the Neural Network for Bayesian Hyperparameter Optimization.

Hyperparameter	**Range**	**Description**
num.dense.layers	$[1, 2]$	Number of hidden layers
num.dense.nodes	unif$(10, 100)$	Number of neurons for each hidden layer
learning.rate	unif$(0.0001, 0.2)$	Learning rate for optimization algorithm
batch.size	unif$(1024, 5120)$	Number of samples that are propagated
activation.function	[ReLU, ELU]	Activation functions for the hidden layer
dropout.rate	unif$(0, 0.5)$	Probability of being temporarily dropped out
optimizer.name	[Adam, Nadam]	Optimization algorithm

As mentioned earlier, the hyperparameters are also tuned using Grid-search. The combinations of hyperparameters that are evaluated using Grid-Search are listed in Table 6. Note that per hyperparameter, the lowest and highest value of the hyperparameter space of BHO is included in the Grid-Search hyperparameter space. The description of the features is the same as in Table 5.

Table 6: Hyperparameter space for the Neural Network for Grid-Search.

Hyperparameter	**Values**
num.dense.layers	$[1, 2]$
num.dense.nodes	$[10, 55, 100]$
learning.rate	$[0.0001, 0.001, 0.01, 0.02]$
batch.size	$[1024, 2048, 3072, 4096, 5120]$
activation.function	[ReLU, ELU]
dropout.rate	$[0.1, 0.25, 0.5]$
optimizer.name	[Adam, Nadam]

4.7 Boosting Methods

Boosting is an ensemble learning method that combines several weak learners into a strong learner. Weak learners are known for their limited predictive ability and are often decision trees with a limited depth. Boosting combines their strengths into a more robust model with a strong performance relative to the weak learners. Three boosting methods are discussed, which are AdaBoost, Gradient Boosting, and XGBoost. The following subsections briefly

explain how boosting methods work, and these subsections work towards the explanation of Extreme Gradient Boosting. This thesis utilizes only Extreme Gradient Boosting since XGBoost is an optimized version of Gradient Boosting. However, XGBoost is based on Gradient Boosting, and Gradient Boosting is based on AdaBoost. Therefore, all these three methods are described.

4.7.1 Relevance

Boosting methods can be used in a regression setting, in which numerical values are predicted. The idea of boosting is to train weak learners by focusing on observations where the model showed poor performance. Hence, the previous model is boosted or updated. Therefore, compared to single machine learning approaches, boosting represents a significant improvement.

4.7.1.1 AdaBoost

Freund and Schapire (1997) suggested the AdaBoost boosting algorithm. AdaBoost is a boosting method that adopts single split decision trees as weak learners. Besides classification, AdaBoost can also be used for regression purposes. This is done by minimizing the mean squared error loss function. The regression problem is then reduced to a binary classification problem. For classification purposes, the accuracy is minimized. The algorithm is considered fast and straightforward to implement. However, AdaBoost is sensitive for outliers, as it builds each tree on previous trees' errors. This means that excessive weight is assigned to these outlying observations (Freund & Schapire, 1997). AdaBoost can be generalized by considering it as an additive model, where a prediction $\hat{f}(x)$ is constructed as a weighted sum of weak learners:

$$\hat{f}(x) = \sum_{m=1}^{M} \beta_m b\left(x, \gamma_m\right),$$

where β_m's are the expansion coefficients, and $b\left(x, \gamma_m\right)$'s are weak learners with parameter γ_m. In every iteration i, β_m, and γ_m are calculated to minimize the exponential loss function when imputed in the former model. The complete AdaBoost algorithm can be found in the work of Freund and Schapire (1997).

4.7.1.2 Gradient Boosting

As mentioned before, Gradient Boosting is based on AdaBoost. The main idea of Gradient Boosting is to obtain an improved model based on an initial guess by deploying the Gradient Descent algorithm. Using the gradient of the loss function, which is approximated by the first-order Taylor expansion, a regression tree can be fitted on observations computed in the gradient. Here, a constant can be computed that optimizes the loss function in the former model. Hence, the model is updated using these optimized constants, after which the procedure is repeated. Furthermore, Gradient Boosting deploys a general loss function, which implies more flexibility than AdaBoost, which uses an exponential loss function. Gradient Boosting also pays less attention to observations with a significant error and is, for that reason, more robust to outliers (Hastie et al., 2001).

4.7.1.3 Extreme Gradient Boosting

Extreme Gradient Boosting is introduced by Chen and Guestrin (2016) and is an extended version of the aforementioned Gradient Boosting. In XGBoost, regression trees are similarly fitted to the gradient as in Gradient Boosting. The second-order Taylor expansion is adopted in XGBoost to produce a more precise approximation of the gradient. The primary difference is that XGBoost finds an optimum using the hessian rather than the gradient, making it more accurate. XGBoost also provides more possibilities for regularization, the training is fast, and it can deal with missing values, the complete XGBoost algorithm can be found in the work of Chen and Guestrin (2016).

4.7.2 Implementation

XGBoost together with BHO is implemented by combining the `XGBoost` and the `HyperOpt` packages in `Python`. Again, the hyperparameters are tuned using BHO. These include the tree booster, the learning task, the maximum depth, and α. A smaller value of the learning task prevents overfitting. Setting a maximum depth of the tree also prevents overfitting. α functions as a regularization term on the weights, where a higher value for α leads to a more conservative model.

Furthermore, early stopping is implemented. If the loss function on the validation set does

not decrease for ten iterations, the training is stopped. The remaining hyperparameters are set at their default value. Furthermore, the MSE function is used as the objective function in the regression context, and the accuracy in the classification context. See Table 7 for the hyperparameter space of the XGBoost model, which is optimized with BHO.

Table 7: Hyperparameters space for XGBoost for Bayesian Hyperparameter Optimization.

Hyperparameter	Range	Description
max_depth	unif(2, 10)	Maximum depth of a tree
learning_rate	unif(0.05, 0.5)	Step size shrinkage to prevent overfitting
reg_alpha	unif(0, 50)	L1 regularization term on weights

Moreover, the hyperparameters are tuned using Grid-search, and the combinations of hyperparameters that are evaluated using Grid-Search are listed in Table 8. Note that again the lowest and highest value of the hyperparameter space of BHO is included in the Grid-Search hyperparameter space. The description of the features is the same as in Table 7.

Table 8: Hyperparameters space for XGBoost for Grid-Search.

Hyperparameter	Range
max_depth	[2, 4, 6, 8, 10]
learning_rate	[0.05, 0.2, 0.35, 0.5]
reg_alpha	[0, 10, 20, 30, 40, 50]

4.8 Transfer Learning

Almost all machine learning models are built under the assumption that the feature space of the training and test data are equal and that they follow the same distribution. The abovementioned assumption is unrealistic in many real-world situations (Pan & Yang, 2009). In such cases, transfer learning can be applied.

When a machine learning model that has already been trained is utilized to address a different but related problem, it is referred to as transfer learning. The following example

demonstrates this: If a simple classifier is trained to predict if a picture contains a dog, the model's knowledge may be used to distinguish other animals such as a cat. In TL, it is the aim to use the information learned in one task to boost generalization in another task. The weights that are learned by a network in task 1 are transferred to a new task 2. Hence, TL approaches strive to transfer information from primary tasks to a target task, while standard machine learning approaches aim to learn each task from scratch (Pan & Yang, 2009).

Wu et al. (2017) state that transfer learning can be split into two main categories: homogeneous and heterogeneous transfer learning. It holds that the data distributions of the source and target data set are different for both categories, the feature spaces of the source and target data are equal in homogeneous transfer learning. For heterogeneous transfer learning, the dimensions of the feature spaces differ. Day and Khoshgoftaar (2017) state that heterogeneous transfer learning can be applied in many real-world examples. This data heterogeneity is challenging to deal with since one must devise a mechanism for bridging the different feature spaces and data distributions. They present an overview of current methodologies that deal with these kinds of situations. In TL, the source tasks must be somehow related to the target task. Negative transfer learning occurs when the information on the source data affects the predictions on the target data in a negative way. In such a case, the source task is irrelevant for the target task. For example, a data collection comprising of bird images will not assist in predicting the nationality of photographs of humans worldwide. The model that was trained on bird data will not enhance the picture data set predictions but may even impair it.

4.8.1 Heterogeneous Transfer Learning

Heterogeneous transfer learning is characterized by different feature spaces in the source and target domains, but it can also be coupled with other problems like different data distributions. This thesis deals with both disparities in feature space and data distributions, so heterogeneous transfer learning is applied.

Day and Khoshgoftaar (2017) provide an overview of existing heterogeneous transfer learning methods. The authors distinguish between methods based on whether the source and target data are labeled. This label can be a class value in a classification context and a

numerical value in a regression context. The authors discuss several types of heterogeneous transfer learning. They begin to discuss supervised heterogeneous transfer learning methods that require labeled source data and limited labeled target data. There is no strict rule on the minimum number of observations for which the target data is considered 'limited'.

Moreover, semi-supervised approaches that do not require labeled target data are discussed. Methods with restricted target labels and no source labels are also examined. Lastly, unsupervised approaches, which do not require labeled source and target data, are described. Each observation in the data utilized in this thesis has a label, which is the house's m^2 price. As all observations are labeled, only supervised heterogeneous transfer learning methods are considered that require labeled source data and limited labeled target data. The method that is used in this thesis is called Heterogeneous Feature Augmentation, which is described in detail in Section 4.8.2

4.8.2 Heterogeneous Feature Augmentation

Li et al. (2013) introduced a heterogeneous transfer learning method called Heterogeneous Feature Augmentation (HFA). Firstly, the main idea of HFA is discussed, after which the method's theory is explained in detail. The authors published an example script of the code in `MATLAB`. As Veneficus prefers to have their code in `Python`, much time is spent implementing this method in `Python`. To accomplish this, it was necessary to understand the method thoroughly. As this took considerable time, the theory of HFA is discussed in this much detail in Section 4.8.2.2

4.8.2.1 Main Idea

This method transforms the data from the source and target domains into a common subspace by using two different projection matrices $\mathbf{P}$ and $\mathbf{Q}$. After that, two feature mapping functions are proposed to augment the transformed data with their original features and zeros. The zeros are used to match the different dimensions between the source and target data features. As a result, the differences in dimensions between the two feature spaces are matched. The projection matrices are learned using Support Vector Machines (SVM). This is an optimization problem that is simplified by introducing a transformation metric $\mathbf{H}$ that

combines $\mathbf{P}$ and $\mathbf{Q}$. Here $\mathbf{H}$ is decomposed into a linear combination of a set of rank-one positive semi-definite (PSD) matrices so that the problem is reformulated into a convex optimization problem. This is similar to a Multiple Kernel Learning (MKL) problem, for which the global optimum can be found by using existing MKL solvers.

4.8.2.2 Theory

In this thesis, the same notation is used as proposed by Li et al. (2013). In the notation for HFA, an apostrophe is used to denote the transpose of a vector or a matrix. The $n \times n$ identity matrix is defined as $\mathbf{I}_n$ and the $n \times m$ matrix of all zeros as $\mathbf{O}_{n \times m}$. Furthemore, $\mathbf{0}_n, \mathbf{1}_n \in \mathbb{R}^n$ are defined as the $n \times 1$ column vectors of zeros and ones, respectively. The ℓ_p -norm of a vector $\mathbf{a} = [a_1, \ldots, a_n]'$ is defined as $\|\mathbf{a}\|_p = (\sum_{i=1}^{n} a_i^p)^{\frac{1}{p}}$, and $\|\mathbf{a}\|$ is used to denote the ℓ_2 -norm. The inequality $\mathbf{a} \leq \mathbf{b}$ means that $a_i \leq b_i$ for $i = 1, \ldots, n$. Moreover, $\mathbf{a} \circ \mathbf{b}$ denotes the element-wise product between the vectors $\mathbf{a}$ and $\mathbf{b}$, i.e., $\mathbf{a} \circ \mathbf{b} = [a_1 b_1, \ldots, a_n b_n]'$. And $\mathbf{H} \succeq 0$ means that $\mathbf{H}$ is a PSD matrix. In their work, it is assumed that there is only one source domain and one target domain, which corresponds to the setting in this thesis, where there is a source domain of existing house data and a target domain with newly built house data. The set of labeled training samples from the source domain is denoted by $\{(x_i^s, y_i^s)|_{i=1}^{n_s}\}$ and the limited number of labeled samples from the target domain by $\{(x_i^t, y_i^t)|_{i=1}^{n_t}\}$, where y_i^s and y_i^t, in this case the property prices, are the labels of the samples x_i^s and x_i^t, respectively, and $y_i^s, y_i^t \in \mathbb{R}_{\geq 0}$. The dimensions of $\mathbf{x}_i^s$ and $\mathbf{x}_i^t$ are denoted by d_s and d_t, respectively. Note that $d_s \neq d_t$, since one deals with heterogeneous transfer learning. Moreover, $\mathbf{X}_s = \left[x_1^s, \ldots, x_{n_s}^s\right] \in \mathbb{R}^{d_s \times n_s}$ and $\mathbf{X}_t = \left[x_1^t, \ldots, x_{n_t}^t\right] \in \mathbb{R}^{d_t \times n_t}$ are defined as the matrices that represent the source and target domains, respectively. Daumé III (2009) proposed Feature Replication to augment the original feature space $\mathbb{R}^d$ into a larger space $\mathbb{R}^{3d}$ by replicating the source and target data for homogeneous TL. Specifically, for any data point x $\in \mathbb{R}^d$, the feature mapping functions φ_s and φ_t for the source and target domains are defined as $\varphi_s(\mathbf{x}) = [\mathbf{x}', \mathbf{x}', 0_d']'$ and $\varphi_t(\mathbf{x}) = [\mathbf{x}', 0_d', \mathbf{x}']'$. It is not useful to use this exact method by adding zeros to match the different feature dimensions of the source and target data, as in that case there would be no correspondences between the heterogeneous features. To successfully utilize the heterogeneous features from the two domains, a common subspace

is first generated for the source and target data so that the heterogeneous features from the two domains may be compared. The common subspace is defined as $\mathbb{R}^{d_c}$, and using two projection matrices $\mathbf{P}$ and $\mathbf{Q}$, any source sample x^s and target sample x^t can be projected onto it, where $\mathbf{P} \in \mathbb{R}^{d_c \times d_s}$ and $Q \in \mathbb{R}^{d_c \times d_t}$, respectively. Motivated by Daumé III (2009), the original features are also incorporated and any source and target domain samples $x^s \in \mathbb{R}^{d_s}$ and $x^t \in \mathbb{R}^{d_t}$ are augmented by using the augmented feature mapping functions φ_{s} and φ_t in the following way:

$$\varphi_s\left(\mathbf{x}^s\right) = \begin{bmatrix} \mathbf{P}\mathbf{x}^s \\ \mathbf{x}^s \\ \mathbf{0}_{d_t} \end{bmatrix}, \varphi_t\left(\mathbf{x}^t\right) = \begin{bmatrix} \mathbf{Q}\mathbf{x}^t \\ \mathbf{0}_{d_s} \\ \mathbf{x}^t \end{bmatrix}.$$

The feature weight vector $\mathbf{w}$ is defined as $[\mathbf{w}_c', \mathbf{w}_s', \mathbf{w}_t']' \in \mathbb{R}^{d_c+d_s+d_t}$ for the augmented feature space, where $\mathbf{w}_c \in \mathbb{R}^{d_c}, \mathbf{w}_s \in \mathbb{R}^{d_s}$ and $\mathbf{w}_t \in \mathbb{R}^{d_t}$ are also the weight vectors that are defined for the common subspace, the source domain and the target domain, respectively. Hereafter, the projection matrices $\mathbf{P}$ and $\mathbf{Q}$ are learnt as well as the weight vector w by minimizing the structural risk functional of SVM. As a result, the HFA method is presented as follows:

$$\min_{\mathbf{P},\mathbf{Q}} \min_{\mathbf{w},b,\xi_i^S,\xi_i^t} \frac{1}{2}\|\mathbf{w}\|^2 + C\left(\sum_{i=1}^{n_S} \xi_i^s + \sum_{i=1}^{nt} \xi_i^t\right), \tag{2}$$

$$\text{s.t.} \quad y_i^s\left(\mathbf{w}'\varphi_s\left(\mathbf{x}_i^s\right) + b\right) \geq 1 - \xi_i^S, \xi_i^S \geq 0; \tag{3}$$

$$y_i^t\left(\mathbf{w}'\varphi_t\left(\mathbf{x}_i^t\right) + b\right) \geq 1 - \xi_i^t, \xi_i^t \geq 0; \tag{4}$$

$$\|\mathbf{P}\|_F^2 \leq \lambda_p, \|\mathbf{Q}\|_F^2 \leq \lambda_q,$$

where $C > 0$ is a trade-off parameter that balances the model complexity on the training samples from both the domains. $\lambda_p, \lambda_q > 0$ control the complexities of $\mathbf{P}$ and $\mathbf{Q}$, and are predefined parameters. Hereafter, the dual form of optimization problem (2) is derived by introducing dual variables $\{\alpha_i^s|_{i=1}^{n_s}\}$ and $\{\alpha_i^t|_{i=1}^{n_t}\}$ for constraints (3) and (4), respectively. Furthermore, the derivatives of the Lagrangian of (2) with respect to $\mathbf{w}$, b, ξ_i^s, ξ_i^t are set to zero, so that the Karush-Kuhn-Tucker conditions are obtained, such that the dual problem becomes:

$$\min_{\mathbf{P},\mathbf{Q}} \max_{\boldsymbol{\alpha}} \mathbf{1}'\boldsymbol{\alpha} - \frac{1}{2}(\boldsymbol{\alpha} \circ \mathbf{y})'\mathbf{K}_{\mathbf{P},\mathrm{Q}}(\boldsymbol{\alpha} \circ \mathbf{y}), \tag{5}$$

$$\text{s.t. } \mathbf{y}'\boldsymbol{\alpha} = 0,$$

$$0 \leq \boldsymbol{\alpha} \leq C\mathbf{1},$$

$$\|\mathbf{P}\|_F^2 \leq \lambda_p, \|\mathbf{Q}\|_F^2 \leq \lambda_q,$$

where $\boldsymbol{\alpha} = \left[\alpha_1^s, \ldots, \alpha_{n_s}^s, \alpha_1^t, \ldots, \alpha_{n_t}^t\right]' \in \mathbb{R}^{n_s+n_t}$ is a vector of the dual variables. $\mathbf{y} = [\mathbf{y}_s', \mathbf{y}_t']'$ is the label vector of all training samples, $\mathbf{y}_s = \left[y_1^s, \ldots, y_{n_s}^s\right]'$ is the label vector of samples from the source domain, $\mathbf{y}_t = \left[y_1^t, \ldots, y_{n_t}^t\right]'$ is the label vector of samples from the target domain, and $\mathbf{K}_{\mathbf{P},\mathbf{Q}} = \begin{bmatrix} \mathbf{X}_s' \left(\mathbf{I}_{d_s} + \mathbf{P}'\mathbf{P}\right) \mathbf{X}_s & \mathbf{X}_s'\mathbf{P}'\mathbf{Q}\mathbf{X}_t \\ \mathbf{X}_t'\mathbf{Q}'\mathbf{P}\mathbf{X}_s & \mathbf{X}_t' \left(\mathbf{I}_{d_t} + \mathbf{Q}'\mathbf{Q}\right) \mathbf{X}_t \end{bmatrix} \in \mathbb{R}^{(n_s+n_t)\times(n_s+n_t)}$ is the derived kernel matrix for the samples from both domains. An intermediate variable $\mathbf{H}$ is introduced that is defined as $\mathbf{H} = [\mathbf{P}, \mathbf{Q}]'[\mathbf{P}, \mathbf{Q}] \in \mathbb{R}^{(d_s+d_t)\times(d_s+d_t)}$. $\mathbf{H}$ is chosen this way, as the projection matrices $\mathbf{P}$ and $\mathbf{Q}$ always appear in the forms of $\mathbf{P}'\mathbf{P}$, $\mathbf{P}'\mathbf{Q}$, $\mathbf{Q}'\mathbf{P}$, and $\mathbf{Q}'\mathbf{Q}$ so that optimization problem (5) is reformulated as:

$$\min_{\mathbf{H}\succeq 0} \max_{\alpha} \mathbf{1}'\boldsymbol{\alpha} - \frac{1}{2}(\boldsymbol{\alpha} \circ \mathbf{y})'\mathbf{K}_{\mathbf{H}}(\boldsymbol{\alpha} \circ \mathbf{y}), \tag{6}$$

$$\text{s.t. } \mathbf{y}'\boldsymbol{\alpha} = 0,$$

$$\mathbf{0} \leq \boldsymbol{\alpha} \leq \mathbf{C1},$$

$$\text{trace}(\mathbf{H}) \leq \lambda,$$

where $\mathbf{K}_{\text{H}} = \mathbf{X}'(\mathbf{H} + \mathbf{I})\mathbf{X}$, $\mathbf{X} = \begin{bmatrix} \mathbf{X}_s & \mathbf{O}_{d_s\times n_t} \\ \mathbf{O}_{d_t\times n_s} & \mathbf{X}_t \end{bmatrix} \in \mathbb{R}^{(d_s+d_t)\times(n_s+n_t)}$ and $\lambda = \lambda_p + \lambda_q$. Instead of learning two projection matrices $\mathbf{P}$ and $\mathbf{Q}$, only a transformation metric $\mathbf{H}$ has to be learned. Thus, only $\mathbf{H}$ and $\boldsymbol{\alpha}$ have to be optimized. In the next part, kernelization is applied, so that optimization problem (6) can be solved in a kernel space by learning a non-linear transformation metric with its size independent from the feature dimensions. This is done since $\mathbf{H}$ is linear, and it can be computationally expensive to learn the linear metric $\mathbf{H}$ when the dimensions of the source and target data, d_s and d_t, grow.

4.8.3 Non-linear Feature Transformation

In this part, it is shown that by using kernelization, the transformation metric $\mathbf{H}$ is unaffected by the feature dimension and grows solely in proportion to the quantity of training data from

both domains. The kernel of the source domain is defined as $\mathbf{K}_s = \Phi_s' \Phi_s \in \mathbb{R}^{n_s \times n_s}$ where $\Phi_s = \left[\phi_s\left(\mathbf{x}_1^s\right), \ldots, \phi_s\left(\mathbf{x}_{n_s}^s\right)\right]$, with $\phi_s(.)$ being the non-linear feature mapping function induced by $\mathbf{K}_s$. The kernel of the target domain is defined similarly. the dimensions of the non-linear features $\phi_s\left(\mathbf{x}^s\right)$ and $\phi_t\left(\mathbf{x}^t\right)$ are denoted as $\tilde{d}_s$ and $\tilde{d}_t$. Based on these, a similar optimization problem as in (6) can be derived, where a transformation metric $\mathbf{H} \in \mathbb{R}^{\left(\tilde{d}_s+\tilde{d}_t\right)\times\left(\tilde{d}_s+\tilde{d}_t\right)}$ is solved that maps the different non-linear features from two domains into a common feature space. However, the dimensions of $\mathbf{H}$ cannot be determined, as the explicit forms of the non-linear feature mapping functions are unknown. For that reason, a non-linear transformation matrix $\tilde{\mathbf{H}} \in \mathbb{R}^{(n_s+n_t)\times(n_s+n_t)}$ is defined, which satisfies that $\mathbf{H} = \Phi \mathbf{K}^{-\frac{1}{2}} \tilde{\mathbf{H}} \mathbf{K}^{-\frac{1}{2}} \Phi'$, where $\mathbf{K} = \begin{bmatrix} \mathbf{K}_s & \mathbf{O}_{n_s \times n_t} \\ \mathbf{O}_{n_t \times n_s} & \mathbf{K}_t \end{bmatrix} \in \mathbb{R}^{(n_s+n_t)\times(n_s+n_t)}$ and $\mathbf{K}^{\frac{1}{2}}$ is the symmetric square root of $\mathbf{K}$. Thus, the kernelization version of (6) can be derived as an optimization problem on $\tilde{\mathbf{H}}$ instead of $\mathbf{H}$. Hereafter, the reformulation of optimization problem (6) is as follows:

$$\min_{\tilde{\mathbf{H}} \succeq 0} \max_{\boldsymbol{\alpha}} \mathbf{1}'\boldsymbol{\alpha} - \frac{1}{2}(\boldsymbol{\alpha} \circ \mathbf{y})' \mathbf{K}_{\tilde{\mathbf{H}}} (\boldsymbol{\alpha} \circ \mathbf{y}), \tag{7}$$

$$\text{s.t. } \mathbf{y}'\boldsymbol{\alpha} = 0,$$

$$\mathbf{0} \leq \boldsymbol{\alpha} \leq C1,$$

$$\text{trace}(\tilde{\mathbf{H}}) \leq \lambda.$$

The main differences with the optimization problem (6) are thus that $\tilde{\mathbf{H}}$ is optimized, which depends on the number of training samples, instead of $\mathbf{H}$, which depends on the number of feature dimensions. However, this implies that one needs to calculate an enormous kernel if the data consists of many samples. As discussed in Section 4.2, this would imply calculating a kernel with dimensions $(n_s + n_t) \times (n_s + n_t)$, where n_s is equal to 154.310 and n_t to $11.247 * 0.8 \approx 9.000$. This kernel could by far not be calculated on the laptop provided by Veneficus, and on any other 'normal' laptop, due to a lack of memory. A solution might be to run this operation in a cloud that has more memory. However, this kernel is also used to fit the SVM in each iteration, a non-parallelizable operation. Therefore, it would take too much time per iteration to fit such a big kernel matrix to the SVM model. This is the reason that a sub-sample of the data is taken to execute this method, as discussed in Section 4.2.

In order to solve the optimization problem in (7) for $\tilde{\mathbf{H}}$ and $\boldsymbol{\alpha}$, a couple of steps are taken that are not discussed in this thesis, as it would adversely affect the readability. However, the intuition that comes along with these steps is discussed.

The optimization problem (7) is reformulated into a convex MKL problem by replacing the Ivanov regularization with the Tikhonov regularization. After this, $\mathbf{H}$ is decomposed as a linear combination of a set of PSD matrices. After this, the problem is reformulated as an Infinite Kernel Learning (IKL) problem, where a closed-form solution can be obtained by using the cutting plane algorithm. Please see the work of Li et al. (2013) for the mathematical details of these steps. After obtaining the optimal solution for $\mathbf{H}$ and $\boldsymbol{\alpha}$, any test sample $\mathbf{x}$ from the target domain can be predicted with the following target decision function:

$$f(\mathbf{x}) = (\boldsymbol{\alpha} \circ \mathbf{y})' \mathbf{K}^{\frac{1}{2}} (\mathbf{H} + \mathbf{I}) \begin{bmatrix} \mathbf{O}_{n_s \times n_t} \\ \mathbf{K}_t^{-\frac{1}{2}} \end{bmatrix} \mathbf{k}_t + b,$$

where $\mathbf{k}_t = \left[k\left(\mathbf{x}_1^t, \mathbf{x}\right), \ldots, k\left(\mathbf{x}_{n_t}^t, \mathbf{x}\right) \right]'$ and $k\left(\mathbf{x}_i, \mathbf{x}_j\right) = \phi_t\left(\mathbf{x}_i\right)' \phi_t\left(\mathbf{x}_j\right)$ is a predefined kernel function for two data samples $\mathbf{x}_i$ and $\mathbf{x}_i$ in the target domain.

There are a few hyperparameters in HFA that need to be set. These hyperparameter include *tau*, *max_iter*, C_s, and C_t. *tau* determines how strict the stopping condition is for the algorithm, *max_iter* stands for the maximum number of algorithm iterations, and C_s and C_t stand for the cost hyperparameters for training the SVM on respectively the source and target dataset. These hyperparameters are left at their default values, which are 0.001, 100, 1, and 1, respectively. This decision is made, as training the HFA model takes quite some time, and with grid search, one needs to train the algorithm from scratch for every unique set of hyperparameters. Besides, it is challenging to implement BHO into this method, as the code for HFA is written from scratch, and there are no pre-built packages that make it relatively easy to implement BHO.

4.9 SHAP values

SHAP values are exploited to give an interpretation to machine learning models. These values show how much each feature contributes to the models' prediction by calculating the change in the expected model prediction when conditioning on a particular feature (Lundberg &

Lee, 2017). The authors state that the base value of a model is equal to $E[f(x)]$, where f is the original prediction model. $E[f(x)]$ is the value that would be predicted if one does not know the features to the current output $f(x)$. In this explanation model, simplified inputs x' are used. These inputs relate to the original inputs through a mapping function that is defined as follows:

$$x = h_x(x'),$$

where the input data is denoted by x. These type of models assign an effect ϕ_i to each feature. Taking the sum of the base value $E[f(x)]$ and all the effects ϕ_i for $i = 1, \ldots, M$ for all M features, approximates the output $f(x)$ of the original model, that is: $\sum_{i=1}^{M} \phi_i + E[f(x)] = f(x)$ (Lundberg & Lee, 2017).

Starting from the base value $E[f(x)]$, one conditions on each feature, one by one, to see how the base value changes. However, the order in which the features are added is influential for non-linear models or dependent features. Therefore, the SHAP values are calculated by taking the average contribution of feature i in each permutation. See Figure 7 for a visualisation of this idea (Lundberg & Lee, 2017).

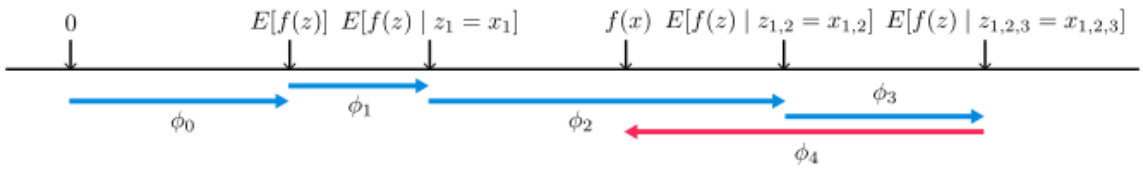


Figure 7: Visualization of SHAP values. $E[f(z)]$ denotes the base value, ϕ_i denotes the contribution of feature i to the base value when conditioning on that feature. When one conditions on all the four features in this example, $E[f(z)|z_{1,2,3,4} = x_{1,2,3,4}] = f(x)$.

Furthermore, an example is presented to illustrate the idea of SHAP values further. Suppose one wants to predict whether a soccer team contains the man-of-the-match player. One could then wonder how much this prediction is driven by a team scoring two goals. However, with SHAP values, one approximates how much a prediction is driven because the team scores two instead of some baseline number of goals. Suppose that the baseline chance that a team contains the man-of-the-match player is 0.498. Figure 8 shows the contribution of all the different features to obtain the output value of the model. Thus, the two goals that

the team scored positively affected the chance of having the man-of-the-match player, while the 38% ball possession had a negative effect on the chance. By taking the sum of all the impact ϕ_i for $i = 1, \ldots, M$, and adding the sum to the baseline chance of 0.498, one comes to the output value of the model, which is 0.700.

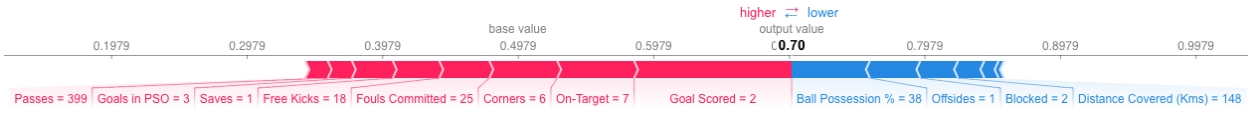


Figure 8: Example that illustrates the process of SHAP values where the predicted value, equal to 0.498, is obtained from the base value, equal to 0.700, by adding the contributions of all the different features to the base value.

5 Results

In this part, the models' findings are discussed. In the context of regression and classification, the results of the approaches are compared in terms of MSE, MAPE, accuracy, and the macro $F1$-score. The first research question is addressed in this way. Secondly, to answer the second research question, model performance is assessed using the two hyperparameter optimization methods in terms of MAPE, running time, and the number of objective function evaluations. Finally, the interpretation of the best-performing model, the XGBoost model, is explored. This is executed with the help of SHAP values, which corresponds to the third research question.

5.1 Individual Regression Model Results

Table 9 highlights the outcomes of the Neural Network (NN) and XGBoost models employed in the regression context. The natural logarithm of the dependent variable is taken, and the features are standardized using min-max scaling, as described in Section 4.4. The best BHO results for the NN and XGBoost models are based on 25 algorithm iterations, implying that the objective function is evaluated 25 times. The results of Grid-Search for the NN and XGBoost model are based on 600 and 3600 fits of various sets of hyperparameters, meaning that the objective function is evaluated 600 and 3600 times, respectively. The hyperparameter optimization results are further discussed in Section 5.3. Noticeably, the

XGBoost model outperforms the NN model since the MSE and the MAPE are significantly lower for BHO. The MAPE is lower for XGBoost in grid-search, whereas the MSE is lower for NN. The lowest MSE and MAPE, equal to 0.08 and 2.502, respectively, are obtained for the XGBoost model using BHO. Therefore, the XGBoost model appears to be most suitable for forecasting house prices, and further evaluations are described in Section 5.3 and 5.4.

Table 9: Mean Squared Error and Mean Average Percentage Error for the Neural Network and XGBoost model, for Bayesian Hyperparameter Optimization and Grid-Search, in the regression context.

	BHO		Grid-Search	
	MSE	MAPE	MSE	MAPE
Neural Network	0.158	3.686	**0.051**	3.365
XGBoost	**0.080**	**2.502**	0.074	**2.714**

5.2 Individual Classification Model Results

Table 10 presents the outcomes of the NN model, the XGBoost model, and the Heterogeneous Feature Augmentation model, employed in the classification context, based on a sub-sample of the data, and the results of the NN and the XGBoost model based on the complete data. The models' hyperparameters are optimized using BHO with 25 iterations, as it demonstrated better performance and is more time-efficient compared to Grid-Search. Section 5.3 delves into the efficiency of the optimization methods in terms of running time and model performance. The XGBoost model, once again, displays the best overall performance. The model constructs the most promising results for buckets 5 and 11. However, for the setting with 8 buckets, the NN exhibits higher accuracy but a lower macro F_1-score. Thus, in comparison to XGBoost, NN and HFA fall short and conceive inferior results. However, the results are based on a relatively small sample of the data, so whether these are reliable is debatable. Moreover, a reason for this difference in results can be perhaps resulted by outlying observations within the test set. Nevertheless, the results are not promising, as XGBoost scores more promising results on the same sub-sample of the data. As a consequence, integrating the manually added features that were utilized for HFA did not improve model performance. By combining existing and newly built housing data into a single dataset, the NN and XGBoost

models disregard market heterogeneity, resulting in superior model performance relative to HFA.

Predictably, the results of the NN and the XGBoost model based on the complete data demonstrate that most accuracies and macro F_1-scores are greater for both models. This is due to the fact that these models are trained on a dataset consisting of roughly 163.000 data points instead of the 12.000 data points in the sub-set. As a result, the models are more accurately trained and better understand the available data patterns.

Table 10: The accuracy and the F_1-score for the Neural Network, XGBoost, and Heterogeneous Feature Augmentation model, for Bayesian Hyperparameter Optimization, in the classification context, based on a sub-sample of the data, and the results of the Neural Network and the XGBoost model based on the complete data.

	Five buckets		Eight buckets		Eleven buckets	
	Accuracy	macro F_1-score	Accuracy	macro F_1-score	Accuracy	macro F_1-score
Sub-sample						
Neural Network	0.572	0.254	**0.502**	0.120	0.104	0.027
XGBoost	**0.689**	**0.368**	0.314	**0.246**	**0.338**	**0.300**
HFA	0.359	0.240	0.173	0.115	0.082	0.060
Complete data						
Neural Network	0.54	0.46	0.29	0.21	0.22	0.18
XGBoost	**0.70**	**0.63**	**0.49**	**0.50**	**0.46**	**0.42**

5.3 Hyperparameter Optimization

The optimal sets of hyperparameters in the regression context for the NN and XGBoost model are presented in Table 11. These sets of hyperparameters are based on 25 iterations of the BHO algorithm and 3600 and 600 fits of different sets of hyperparameters for the Grid-Search algorithm, for the NN, and XGBoost model, respectively.

Table 11: Optimized hyperparameters of the Neural Network and XGBoost model for Bayesian Hyperparameter Optimization and Grid-Search, in the regression context.

Model	Hyperparameters	BHO	Grid-search
Neural Network	num_dense_layers	1	1
	num_dense_nodes	89	100
	learning_rate	0.020	0.02
	batch_size	3566	1024
	activation_function	ELU	ReLu
	dropout_rate	0.006	0.25
	optimizer	Nadam	Adam
XGBoost	max_depth	7	10
	learning_rate	0.379	0.2
	reg_alpha	1.900	0

The number of objective function evaluations and the running time in minutes of the two optimization methods for varying iteration values in the regression context are shown in Table 12. It is worth noticing that with BHO, 25 iterations are sufficient to achieve a relatively low MSE and MAPE. Running the algorithm for 50 iterations is more time-consuming while having no significant increase in the models' performance. This is best illustrated by looking at the difference in MAPE for NN with 25 and 50 iterations of the BHO algorithm. The MAPE remains the same, but the running time has more than doubled. As a result, in the context of this thesis, 25 iterations of the BHO algorithm appear to be sufficient. Furthermore, it is concluded that BHO outperforms Grid-Search, in terms of model performance. Grid-Search is commonly known to be an inefficient method, but as Veneficus still employs this method, it was only fair to compare it with BHO. Therefore, it might be valuable to consider another hyperparameter optimization method like BHO. However, it should be emphasized that Grid-Search outperforms BHO in some circumstances. This happens for 3600 objective functions evaluations of Grid-Search. The MAPE, equal to 3.365, is lower than the MAPE for 50 iterations of the BHO algorithm. This is most likely due to BHO's overemphasis on promising regions of hyperparameters rather than examining all regions. BHO may overlook an unpromising region where the global optimum may be found. Grid-Search, on the other hand, does not skip unpromising regions because it tries all sets of

hyperparameters in the pre-defined space, which is most likely the cause of this outcome.

Additionally, it is noticed that BHO outperforms Grid-Search in terms of running time. BHO is faster in every case, except for 600 objective function evaluations of Grid-Search, compared to 50 iterations of BHO. However, as 25 iterations of BHO seems sufficient for outperforming Grid-Search, this result is negligible. Noticeably, BHO takes less running time in almost every case. However, when comparing the number of function evaluations, one might expect a more significant difference in running time between BHO and Grid-Search. It takes 11 minutes to conduct 50 BHO objective function evaluations, but only four times as long to do 3600 Grid-Search objective function evaluations, which are 72 times as many objective function evaluations. One probable cause is that in the context of this thesis, assessing the Tree Parzen Estimator, the surrogate function utilized for BHO, takes longer than expected. (Bergstra et al., 2011) recommended this surrogate function, although BHO may be quicker if Gaussian Processes or Random Forests were used instead.

The different number of objective function evaluations of the algorithm provides the same results for XGBoost utilizing Grid-Search. This is the case, as the optimal hyperparameters correspond to the extremes in the hyperparameter space, included in all three regions to make a fair comparison.

Table 12: The number of objective function evaluations and running time in minutes for the Bayesian Hyperparameter Optimization and Grid-Search algorithms for different iteration values.

	BHO			Grid-Search		
Model	Function evaluations	Running time	MAPE	Function evaluations	Running time	MAPE
Neural Network	10	**2**	**8.306**	1.080	14	9.645
	25	**4**	**3.686**	2.160	31	10.096
	50	**11**	3.686	3.600	45	**3.365**
XGBoost	10	**25**	**2.504**	135	40	2.714
	25	**77**	**2.502**	320	99	2.714
	50	180	**2.709**	600	**175**	2.714

5.4 Model Interpretation

The best-performing model is XGBoost, which hyperparameters are tuned using 25 iterations of the BHO algorithm. This model is extensively evaluated using SHAP values. With

these values, one can investigate which contribution each feature has in the models' predictions. Figure 9 shows the feature contributions of the first m^2 price prediction in the test set, utilizing SHAP values. Here, the base value $E[f(x)]$ that corresponds to the expected prediction of the model is equal to 3.115 euro per m^2. The features shown in blue and red contribute positively and negatively to the m^2 price prediction and hence contain a positive and negative SHAP value, respectively. Seemingly, the construction year, being equal to 2021 for this observation, contributes positively to the model's price prediction as it is shown in red in Figure 9.

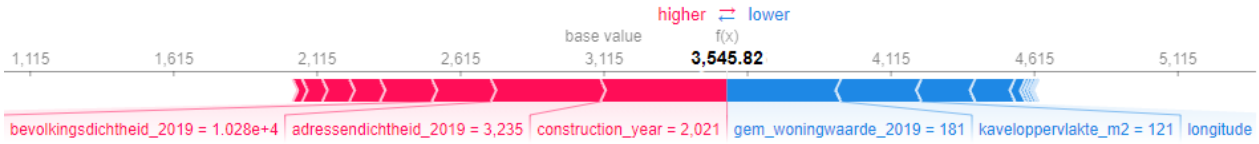


Figure 9: Feature contributions from base value, equal to 3115, to m^2 price prediction of the XGBoost model, equal to 3545.82, for the first observation in the test set. The predicted m^2 price is obtained by adding the contributions of all the different features to the base value.

Depending on the value of that feature for a particular observation, it can affect the model's prediction. For example, a high value of 'inkomen_per_inwoner_2017', which specifies the average income per capita in 2017, may positively contribute to the model's prediction and vice versa. An overview of how each feature value affects the model's prediction is presented in Figure 10. The different features are listed in the rows, where each dot represents an observation used to calculate the SHAP value. If the dot is blue, the feature value is relatively low and relatively high if the dot is red. The horizontal axis indicates whether a feature contributes negatively or positively to the model's price prediction. Most of the dots of 'gem_woningwaarde_2019', which stands for the average price of a home in 2019, correspond to a positive SHAP value. Therefore, it seems that this feature contributes on average favorably to the model's price forecast.

In Section 3.3, a couple of features' correlations with the m^2 price were investigated. The features 'gem_woningwaarde_2019' and 'inkomen_gem_per_inwoner_2017' were deemed significant predictors, as the magnitude of its correlations was the highest. As the SHAP values are relatively high, Figure 10 reveals that, in particular, 'gem_woningwaarde_2019' is indeed a crucial predictor.

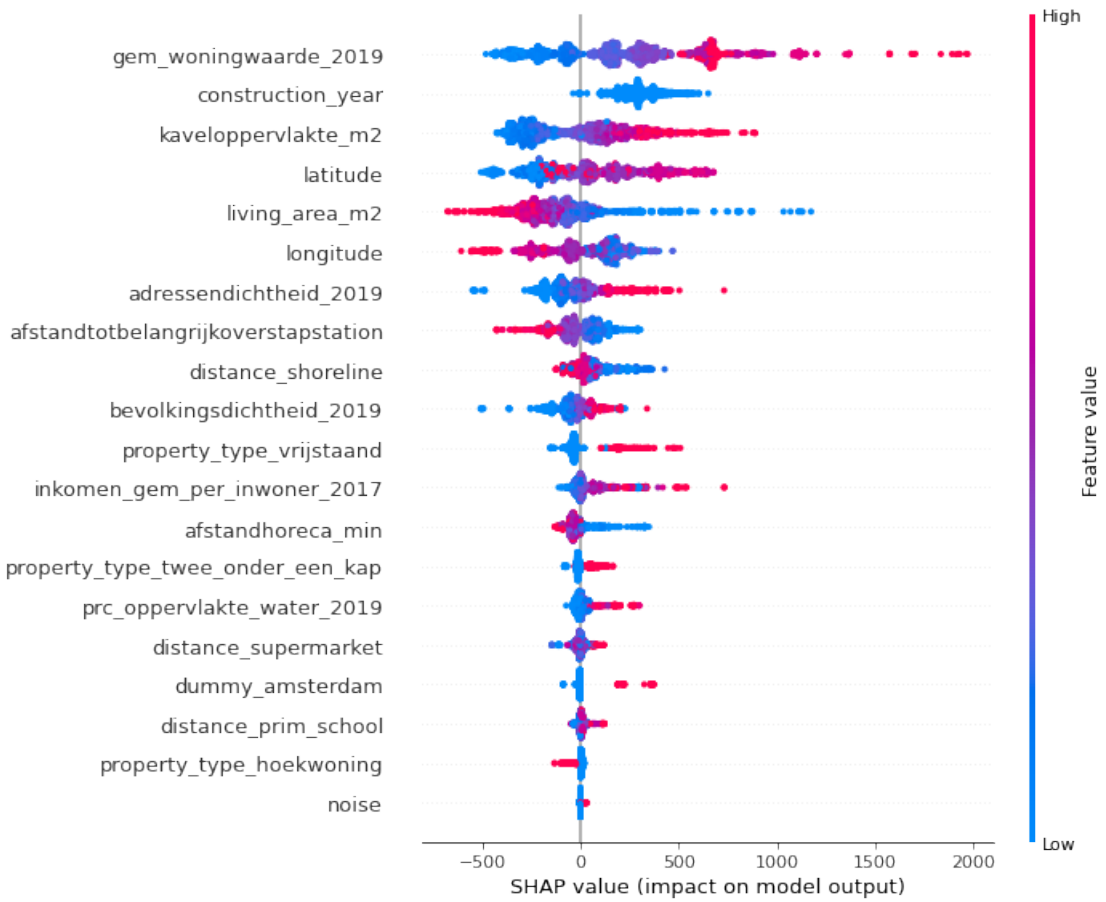


Figure 10: Summary plot of the SHAP values of the XGBoost model. The different features are listed in the rows, where each dot represents an observation used to calculate the SHAP value. If the dot is blue, the feature value is relatively low and relatively high if the dot is red. The horizontal axis indicates whether a feature contributes negatively or positively to the model's m^2 price prediction.

Moreover, Figure 11 depicts the average relevance of each feature to the model. The average magnitude of the SHAP values per feature is taken across the dataset, plotted as a bar chart. One notices that 'gem_woningwaarde_2019' and 'kaveloppervlakte_m2', 'latitude', 'longitude', and 'living_area_m2' have the SHAP values with the largest absolute magnitude. The exact definitions of these features can be found in Table 1 in the Appendix. It is worth noting that just because a feature has a lower magnitude does not mean it is irrelevant, given the possibility that low- and high magnitude features are correlated. This is further discussed in Section 5.5.

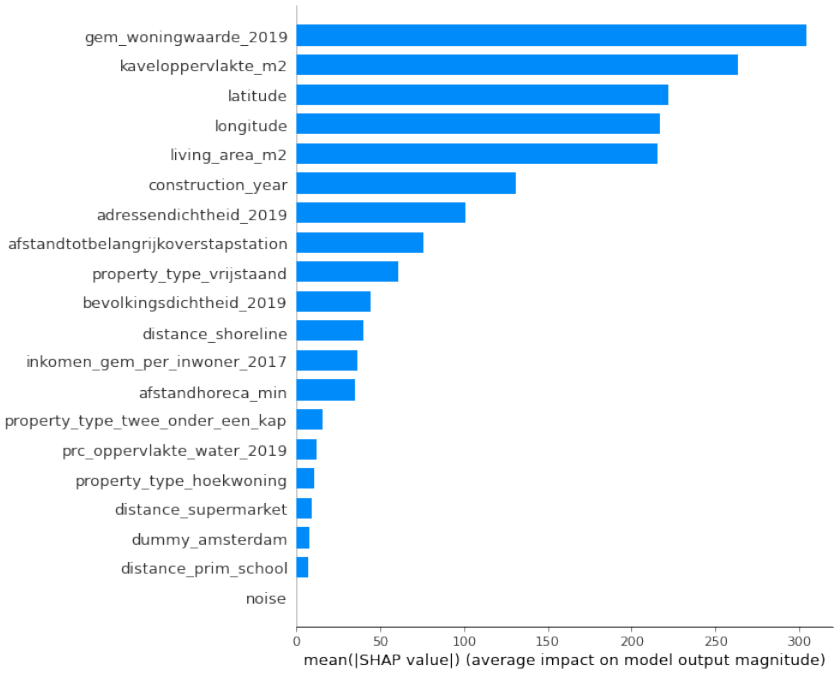
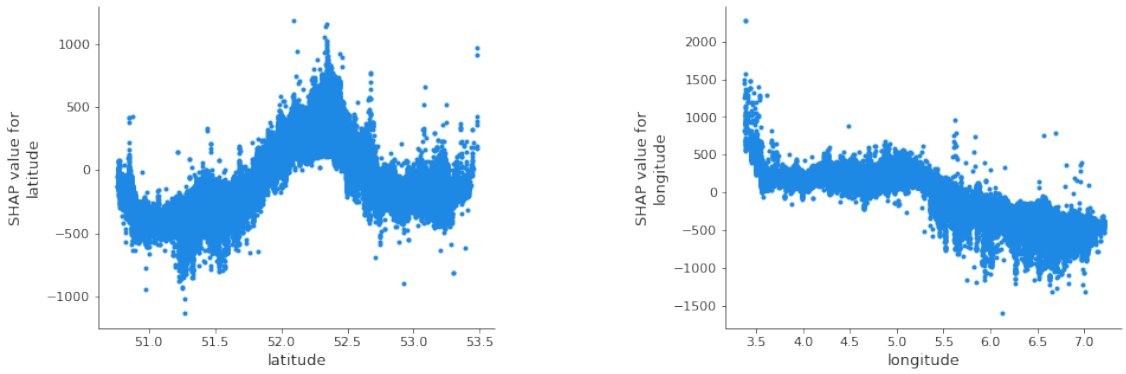


Figure 11: Feature importance of the XGBoost model, which shows the average relevance of each feature to the model's m^2 price prediction.

The contribution of each feature to the models' prediction is investigated in more detail. Expectations concerning the impact of specific features were offered in Section 3.3, based on the magnitude of their correlation with the m^2 price. Figures 12a and 12b show the dependence plots of 'latitude' and 'longitude' between the contribution to the models' prediction and the feature values. A dependence plot is a scatter plot that illustrates the impact of a single feature on the model's m^2 price prediction. The higher the 'latitude' is, the more north to the equator the place is located. Figure 12a depicts that for low and high latitude values, one notices low SHAP values, where for average latitude values, one notices high SHAP values. This is clarified by the fact that the largest cities in the Netherlands, where the m^2 price is high, are located in the center of the country. For the longitude, one notices a negative linear pattern. How further away one moves to the east of the country, how lower the SHAP values are. This corresponds to the lower m^2 prices in the east of the country.

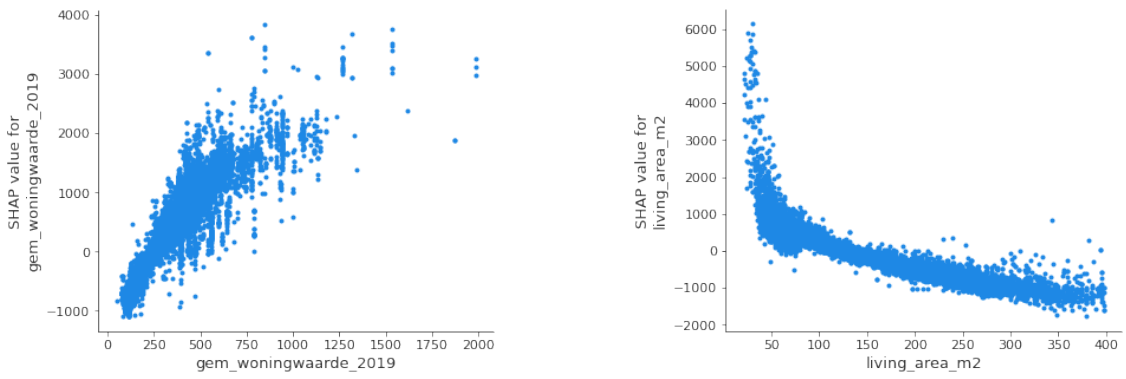


(a) Latitude

(b) Longitude

Figure 12: Dependence plots of 'latitude' and 'longitude' features in the XGBoost model, which illustrate the features' contributions to the model's m^2 price prediction.

Furthermore, the features 'gemiddelde_woningwaarde_2019' and 'living_are_m2' are highlighted. Their dependence plots are presented in Figures 13a and 13b. There is a clear positive correlation between the value of 'gemiddelde_woningwaarde_2019' and its SHAP values, which implies that high values for 'gemiddelde_woningwaarde_2019' indicate a significant positive contribution to the model's prediction, this relationship is rather intuitive. However, from Figure 13b it is clear that 'living_area_m2' demonstrates a negative relation with the SHAP value, which perhaps appears counter-intuitive, as its correlation with the m^2 price is positive, as demonstrated in Section 3.3. The order in which the features were implemented might play a role here. For example, the features 'kaveloppervlake_m2' and 'living_area_m2' are positively correlated, with a correlation of 0.43. For determining the SHAP values, this implies that the latest added feature of these two has a relatively small contribution to the m^2 price prediction. The dependency plots of the remaining features can be found in Figures 1-8 in the Appendix.



(a) Gemiddelde_woningwaarde_2019

(b) Living_are_m2

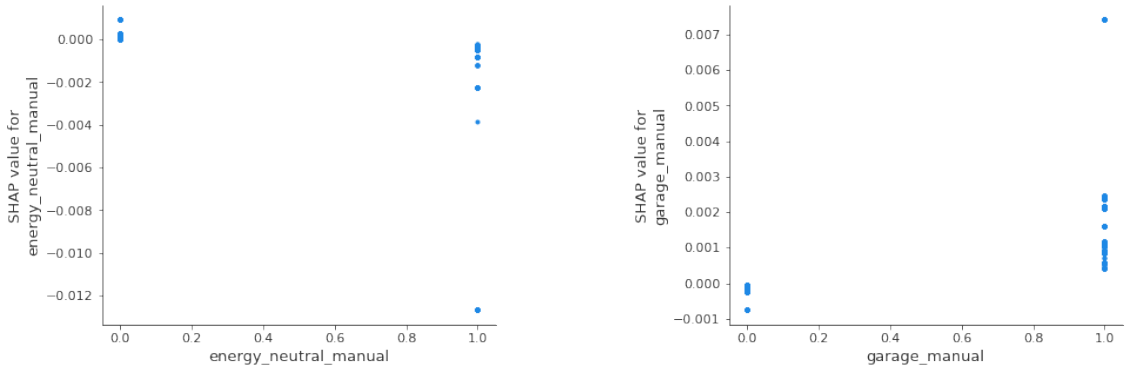
Figure 13: Dependence plots of 'gemiddelde_woningwaarde_2019' and 'living_are_m2' features in the XGBoost model.

5.5 Sensitivity Analysis

The influence of features on the model's prediction is the subject of sensitivity analysis. Sensitivity analysis is performed on the best predicting model, which is the XGBoost model. This analysis is performed using two methods, which are feature importance and feature ablation. The outcome of the feature importance method is presented in Figure 11. The features 'gem_woningwaarde_2019' and 'kaveloppervlakte_m2', 'latitude', 'longitude', and 'living_area_m2' have the highest absolute magnitude and are therefore the most important features. Feature ablation measures the contribution of each feature by excluding each feature from the data once. Hereafter, the algorithm is trained on the data without that feature, and the prediction is made on the test set, from which the MAPE is obtained. This is done for every feature, and the higher the MAPE is compared to the MAPE with all features, which is 2.502, the higher the contribution of that particular feature is. The three features for which the MAPE increased the most when that feature is excluded from training the model are: 'kaveloppervlakte_m2' (3.127), 'contruction_year' (2.803), and 'gem_woningwaarde_2019' (2.687). This is in line with the results of the importance plot, except for 'latitude' and 'longitude'. A possible explanation is that 'latitude' and 'longitude' share a correlation of 0.25. This implies that if one of them is deleted, its information is partly captured by the other.

5.6 Impact Added Features

Venficus specifically requested to provide an interpretation regarding the impact of the manually added features. However, HFA did not prove to be a success. NN and XGBoost did not employ the manually inserted features. To explain the impact of these features, an XGBoost model is trained on newly built house data so that the SHAP values can be calculated for the manually added features. Veneficus' interest in the impact of the m^2 price is due to the varying features that change the worthiness of a house. These features include a house with a garden pointing to the south, a garage, a built-in kitchen and bathroom, or being energy-neutral. However, the predictive power of these features turned out to be negligible. The abovementioned features' importance is so insignificant that they are not even included in the feature importance plot, which can be found in Figure 9 in the Appendix. Figure 14a and 14b illustrate that no conclusions about these features can be drawn, as the magnitude of the SHAP values is negligible. Figure 14b emphasizes a minimal positive effect on the model's prediction of having a garage. However, Figure 14a depicts that having an energy-neutral house negatively contributes to the price prediction, which seems counter-intuitive.



(a) Energy_neutral_manual

(b) Garage_manual

Figure 14: Dependence plots of 'energy_neutral_manual' and 'living_area_m2' features.

6 Conclusion

From Veneficus' perspective, it is best to utilize an XGBoost to predict the m^2 price of newly built houses relative to Neural Network (NN) and Heterogeneous Feature Augmentation (HFA). XGBoost is the best performing method with a MAPE of 2.502 in the regression context, using 25 iterations of the Bayesian Hyperparameter Optimization (BHO) algorithm. Additionally, NN and XGBoost are compared with HFA in a classification context, as putting HFA into practice in a regression environment was not as simple as the authors of HFA described. Moreover, a subset of the data was used, as HFA took significant time to execute, which made it unfeasible to use the entire data. Unfortunately, HFA showed poor performance relative to NN and XGBoost. XGBoost was again the best performing method in the cases where the m^2 prices were divided into 5, 8, and 11 different buckets. NN and XGBoost were also compared based on the complete data, where XGBoost showed the best performance for all the tested buckets. Regarding the research question, this means that the $m2$ price could not be predicted accurately using transfer learning. Therefore, the manually added features could not be utilized for predicting the m^2 price of newly built houses, as XGBoost outperforms HFA without using these extra features.

Furthermore, it is suggested to utilize BHO instead of Grid-Search for tuning its models' hyperparameters, as BHO is faster and shows better performance in terms of MAPE. Thus, BHO affects running time and model performance positively and results in a more efficient algorithm. In this thesis' setting, it is sufficient to use only 25 iterations of the BHO algorithm because the performance did not increase further by using more iterations. However, this may differ per dataset.

Finally, the interpretation of the model results is obtained by exploiting SHAP values, and the features having the most considerable contribution to the XGBoost model's predictions are found. Based on feature importance and ablation, essential predictors turned out to be the average price of a home, the plot area, the living area, the longitude, and the latitude. Furthermore, these features showed logical relations with the SHAP values. Houses located in major cities result in higher m^2 prices, and the same holds for cities with a high average house price.

7 Limitations and Further Research

One might further consider investigating putting HFA in a regression context. Creating buckets to predict the m^2 price is cumbersome, instead, one wants to estimate the m^2 price directly. The fact that HFA did not perform well in a classification context does not imply that it provides poor performance in a regression context, so it might be worth investigating. A flaw encountered while running the models was the lack of computing power and laptop memory, as it was not sufficient to perform HFA on the complete data. This was a weakness as the HFA may have resulted in better predictions. For future calculations, it can be considered to run the model in a cloud. However, this approach would still be costly and take considerable time.

Furthermore, although HFA did not turn out successful in this thesis, heterogeneous transfer learning (HTL) has excellent potential to advance further and be applicable in a more general way. Other HTL methods not considered in this thesis may outperform HFA, which should be investigated. As of now, HTL is mainly used in the field of image recognition, but many real-life scenarios deal with circumstances in which HTL can offer a solution, such as predicting newly built house prices.

Finally, from Veneficus' perspective, it is encouraged to extend the manually enriched dataset further. Although this is a costly and time-consuming process, it creates a unique dataset, which serves the opportunity to outperform other companies operating in the same field. This dataset could be used for other HTL methods than HFA, which have a shorter running time and can utilize the complete enriched dataset.

References

Al Shalabi, L., Shaaban, Z., & Kasasbeh, B. (2006). Data Mining: A Preprocessing Engine. *Journal of Computer Science*, *2*(9), 735–739.

Bergstra, J., Bardenet, R., Bengio, Y., & Kégl, B. (2011). Algorithms for hyper-parameter optimization. *25th annual conference on neural information processing systems (NIPS 2011)*, *24*.

Bergstra, J., Yamins, D., Cox, D. D., et al. (2013). Hyperopt: A Python Library for Optimizing the Hyperparameters of Machine Learning Algorithms. *Proceedings of the 12th Python in science conference*, 13–19.

Chen, T., & Guestrin, C. (2016). XGBoost: A Scalable Tree Boosting System. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 785–794.

Clevert, D.-A., Unterthiner, T., & Hochreiter, S. (2015). Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289*.

Daumé III, H. (2009). Frustratingly easy domain adaptation. *arXiv preprint arXiv:0907.1815*.

Day, O., & Khoshgoftaar, T. M. (2017). A survey on heterogeneous transfer learning. *Journal of Big Data*, *4*(1), 1–42.

Desautels, T., Calvert, J., Hoffman, J., Mao, Q., Jay, M., Fletcher, G., Barton, C., Chettipally, U., Kerem, Y., & Das, R. (2017). Using transfer learning for improved mortality prediction in a data-scarce hospital setting. *Biomedical informatics insights*, *9*, 1178222617712994.

Fawaz, H. I., Forestier, G., Weber, J., Idoumghar, L., & Muller, P.-A. (2018). Transfer learning for time series classification. *2018 IEEE international conference on big data (Big Data)*, 1367–1376.

Freund, Y., & Schapire, R. E. (1997). A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting. *Journal of Computer and System Sciences*, *55*(1), 119–139.

Géron, A. (2019). *Hands-on machine learning with scikit-learn, keras, and tensorflow: Concepts, tools, and techniques to build intelligent systems*. O'Reilly Media.

Hastie, T., Tibshirani, R., & Friedman, J. (2001). *The Elements of Statistical Learning*. Springer New York Inc.

Hornik, K., Stinchcombe, M., & White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural networks*, *2*(5), 359–366.

Hyndman, R. J. et al. (2006). Another look at forecast-accuracy metrics for intermittent demand. *Foresight: The International Journal of Applied Forecasting*, *4*(4), 43–46.

Lenk, M. M., Worzala, E. M., & Silva, A. (1997). High-tech valuation: Should artificial neural networks bypass the human valuer? *Journal of Property Valuation and Investment*.

Li, W., Duan, L., Xu, D., & Tsang, I. W. (2013). Learning with augmented features for supervised and semi-supervised heterogeneous domain adaptation. *IEEE transactions on pattern analysis and machine intelligence*, *36*(6), 1134–1148.

Limsombunchai, V. (2004). House price prediction: Hedonic price model vs. artificial neural network. *New Zealand agricultural and resource economics society conference*, 25–26.

Lundberg, S., & Lee, S.-I. (2017). A unified approach to interpreting model predictions. *arXiv preprint arXiv:1705.07874*.

Meng, C., Zhou, L., & Liu, B. (2020). A case study in credit fraud detection with smote and xgboost. *Journal of Physics: Conference Series*, *1601*(5), 052016.

Miller Jr, N. G., & Markosyan, S. (2003). The academic roots and evolution of real estate appraisal. *The Appraisal Journal*, *71*(2), 172.

Pagourtzi, E., Assimakopoulos, V., Hatzichristos, T., & French, N. (2003). Real estate appraisal: A review of valuation methods. *Journal of Property Investment & Finance*.

Pan, S. J., & Yang, Q. (2009). A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, *22*(10), 1345–1359.

Park, B., & Bae, J. K. (2015). Using machine learning algorithms for housing price prediction: The case of fairfax county, virginia housing data. *Expert systems with applications*, *42*(6), 2928–2934.

Peterson, S., & Flanagan, A. (2009). Neural network hedonic pricing models in mass real estate appraisal. *Journal of real estate research*, *31*(2), 147–164.

Refaeilzadeh, P., Tang, L., & Liu, H. (2009). Cross-validation. *Encyclopedia of database systems*, *5*, 532–538.

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning representations by back-propagating errors. *nature*, *323*(6088), 533–536.

Selim, S. (2011). Determinants of house prices in turkey: A hedonic regression model. *Doğuş Üniversitesi Dergisi*, *9*(1), 65–76.

Shi, X., Liu, Q., Fan, W., Philip, S. Y., & Zhu, R. (2010). Transfer learning on heterogenous feature spaces via spectral transformation. *2010 IEEE international conference on data mining*, 1049–1054.

Sirmans, S., Macpherson, D., & Zietz, E. (2005). The composition of hedonic pricing models. *Journal of real estate literature*, *13*(1), 1–44.

Snoek, J., Larochelle, H., & Adams, R. P. (2012). Practical bayesian optimization of machine learning algorithms. *arXiv preprint arXiv:1206.2944*.

Veneficus. (2021). Www.veneficus.nl.

Weiss, K., Khoshgoftaar, T. M., & Wang, D. (2016). A survey of transfer learning. *Journal of Big data*, *3*(1), 1–40.

Worzala, E., Lenk, M., & Silva, A. (1995). An exploration of neural networks and its application to real estate valuation. *Journal of Real Estate Research*, *10*(2), 185–201.

Wu, Q., Wu, H., Zhou, X., Tan, M., Xu, Y., Yan, Y., & Hao, T. (2017). Online transfer learning with multiple homogeneous or heterogeneous sources. *IEEE Transactions on Knowledge and Data Engineering*, *29*(7), 1494–1507.

Zhang, W., Wu, C., Zhong, H., Li, Y., & Wang, L. (2020). Prediction of undrained shear strength using extreme gradient boosting and random forest based on Bayesian optimization. *Geoscience Frontiers*, *12*(1), 469–477.

Zhao, Y., Chetty, G., & Tran, D. (2019). Deep learning with xgboost for real estate appraisal. *2019 IEEE Symposium Series on Computational Intelligence (SSCI)*, 1396–1401.

8 Appendix

Table 1: Features which are available for the existing and newly built house data, together with the variable type and a description of each feature.

Existing	**Newly built**	**Type**	**Description**
price_per_m2_adjusted	price_per_m2	Float	The square meter price of a house, adjusted for inflation
latitude	latitude	Float	How far north or south of the equator a place is located
longitude	longitude	Float	How far east or west of the equator a place is located
poperty_type_hoekwoning	poperty_type_hoekwoning	Float	Dummy that indicates a type of house
property_type_twee_onder_een_kap	property_type_twee_onder_een_kap	Float	Dummy that indicates a type of house
property_type_vrijstaand	property_type_vrijstaand	Float	Dummy that indicates a type of house
living_area_m2	living_area_m2	Float	Living area in square meters
kaveloppervlakte_m2	kaveloppervlakte_m2	Float	Plot area in square meters
afstandtotbelangrijkoverstapstation	afstandtotbelangrijkoverstapstation	Float	Distance to station (km)
distance_supermarket	distance_supermarket	Float	Distance to supermarket (km)
adressendichtheid_2019	adressendichtheid_2019	Float	Address density in 2019
bevolkingsdichtheid_2019	bevolkingsdichtheid_2019	Float	Population density in 2019
dummy_amsterdam	dummy_amsterdam	Integer	Dummy that indicates whether the city is Amsterdam
gem_woningwaarde_2019	gem_woningwaarde_2019	Float	Average price of a home in 2019
inkomen_gem_per_inwoner_2017	inkomen_gem_per_inwoner_2017	Float	Average income in 2017
prc_oppervlakte_water_2019	prc_oppervlakte_water_2019	Float	Water area in square meters
distance_prim_school	distance_prim_school	Float	Distance to primary school (km)
afstandhoreca_min	afstandhoreca_min	Float	Distance to hospitality industry (km)
construction_year	construction_year	Float	Construction year
pc2	pc2	Integer	Two-digit postal code
distance_shoreline	distance_shoreline	Float	Distance to the shoreline (km)
noise	noise	Integer	Dummy that indicates whether noise is above a threshold
project_name_label	project_name_label	String	Label that indicates the name of the project
	kitchen_manual	Integer	Dummy that indicates whether the kitchen is included in the price
	bathroom_manual	Integer	Dummy that indicates whether the bathroom is included in the price
	energy_neutral_manual	Integer	Dummy that indicates whether the house provides its own energy
	front_orien	Category	Direction which the frontyard faces
	back_orien	Category	Direction which the frontyard faces
	huiz_parking	Category	Feature that says what the parking opportunities of the house are
	garage	Integer	Dummy that indicates whether the house has a garage
	number_of_floors	Integer	Number of floors of the house
	outhouse	Integer	Dummy that indicates whether the house has a outhouse
	achterom	Integer	Dummy that indicates whether the house can be entered from the back
	corner	Integer	Dummy that indicates whether the house lies in a corner position
	next_to	Category	Feature that tells what the most important factor is near the house, which can be water, green, street, and walkway
	backyard	Integer	Dummy that indicates whether the house has a backyard
	frontyard	Integer	Dummy that indicates whether the house has a frontyard

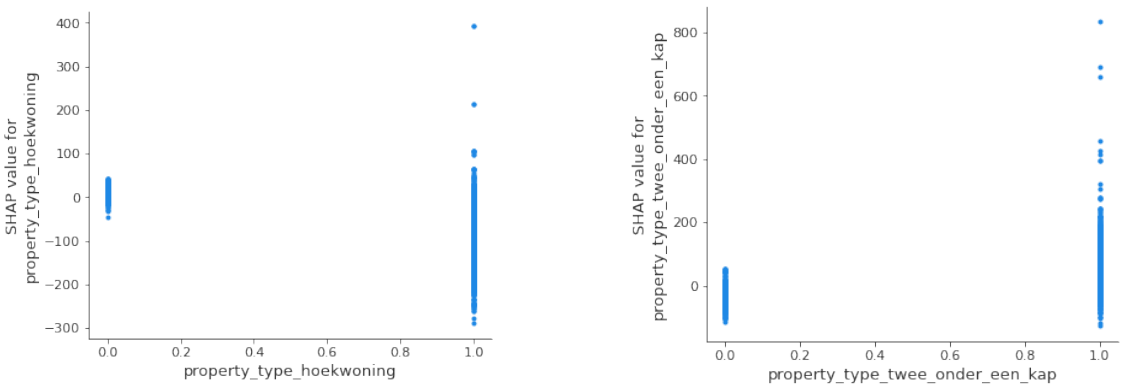


Figure 1: Dependence plots of 'hoekwoning' and 'twee_onder_een_kap' features.

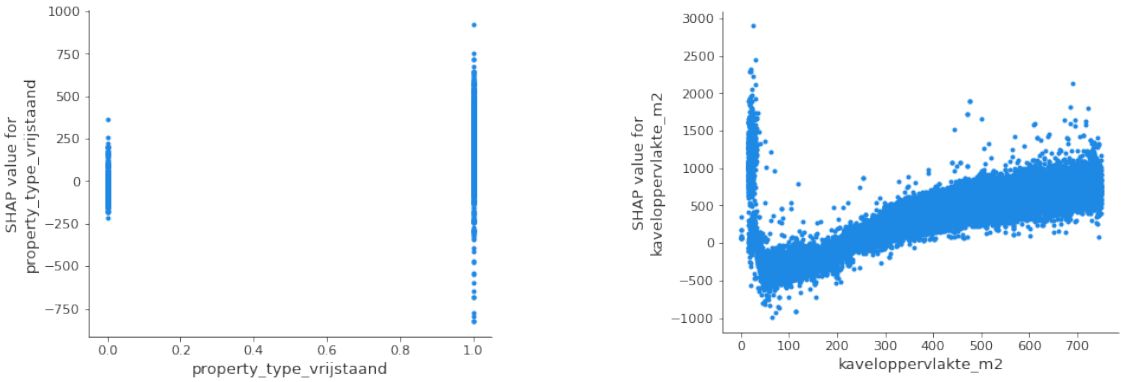


Figure 2: Dependence plots of 'vrijstaand' and 'kaveloppervlakte_m2' features.

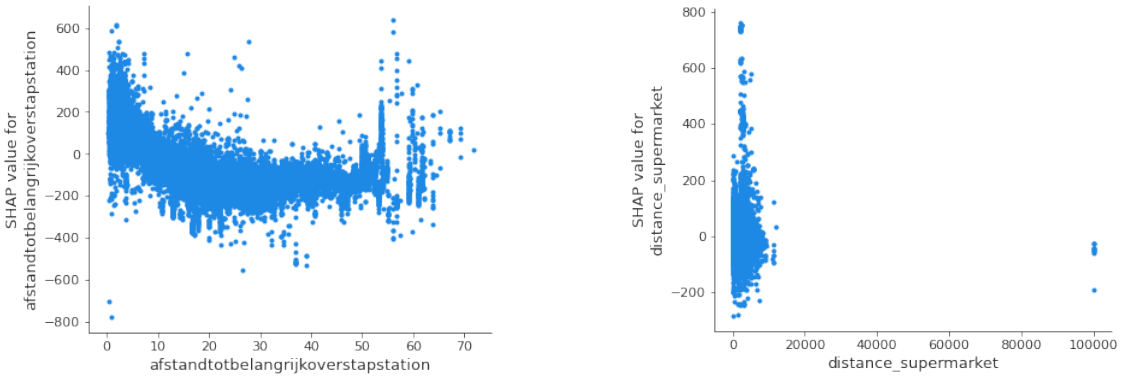


Figure 3: Dependence plots of 'afstand_tot_station' and 'distance_supermarket' features.

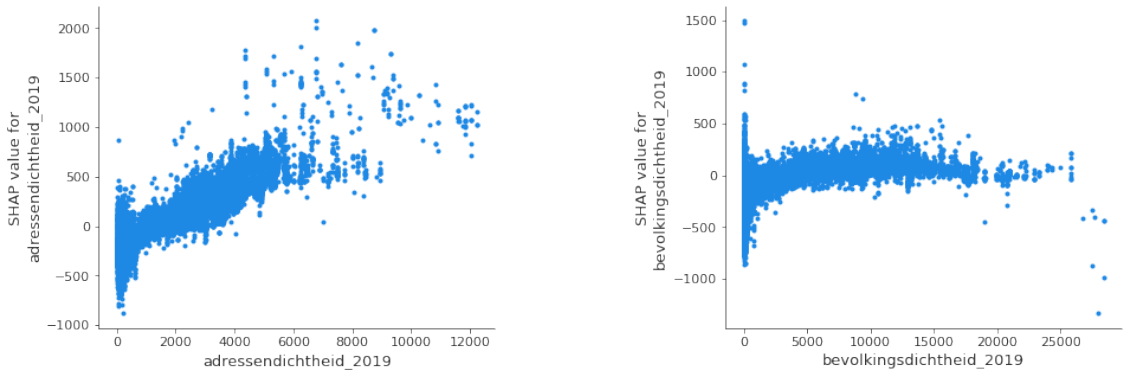


Figure 4: Dependence plots of 'adressendichtheid' and 'bevolkingsdichtheid' features.

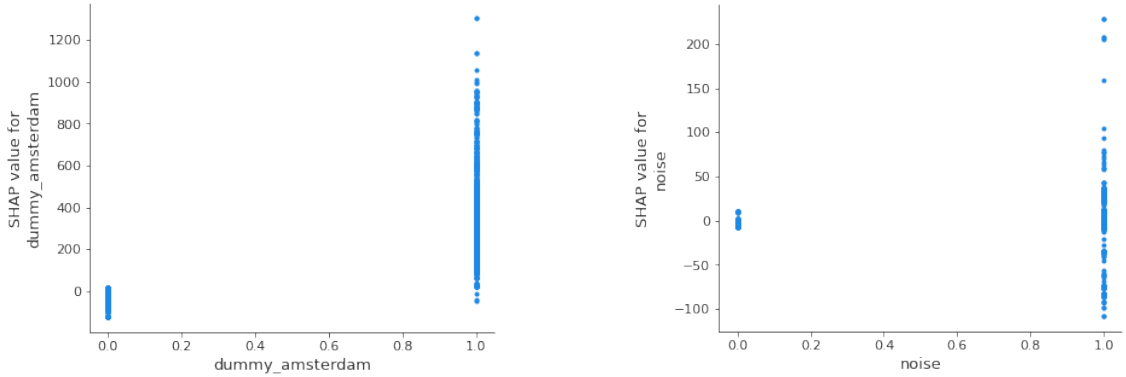


Figure 5: Dependence plots of 'dummy_amsterdam' and 'noise' features.

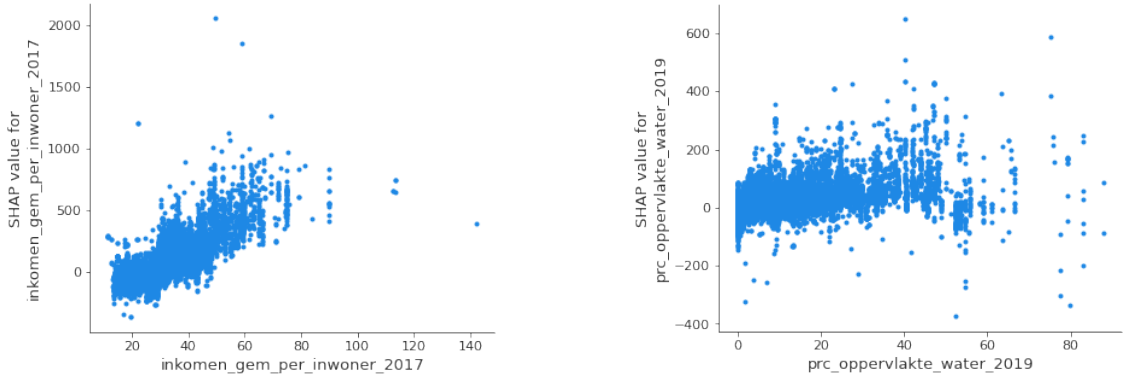


Figure 6: Dependence plots of 'inkomen_per_inwoner' and 'oppervlakte_water' features.

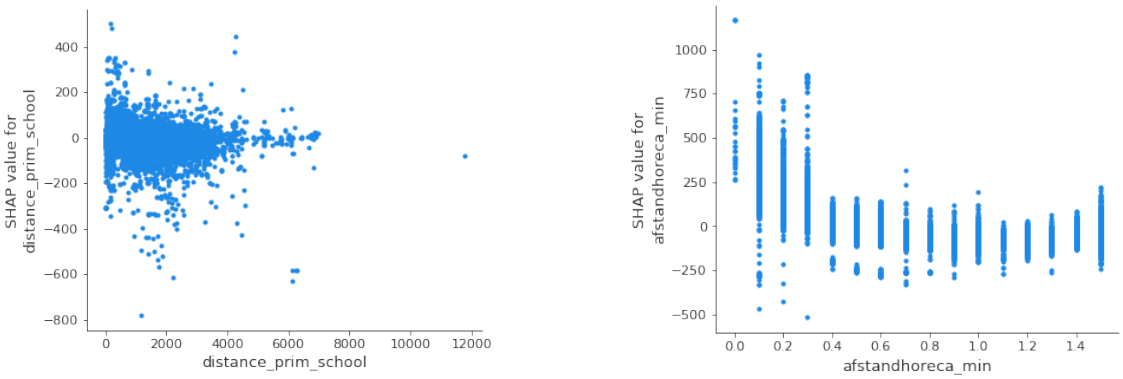


Figure 7: Dependence plots of 'distance_school' and 'afstand_horeca' features.

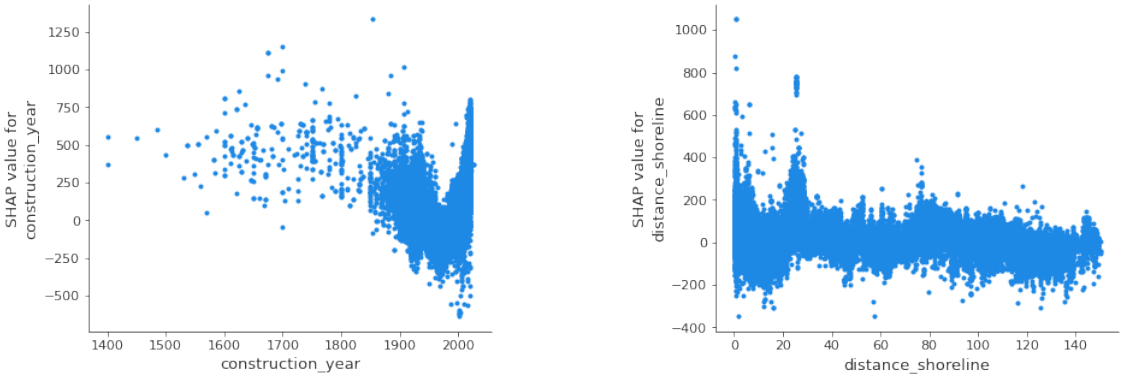


Figure 8: Dependence plots of 'construction_year' and 'distance_shoreline' features.

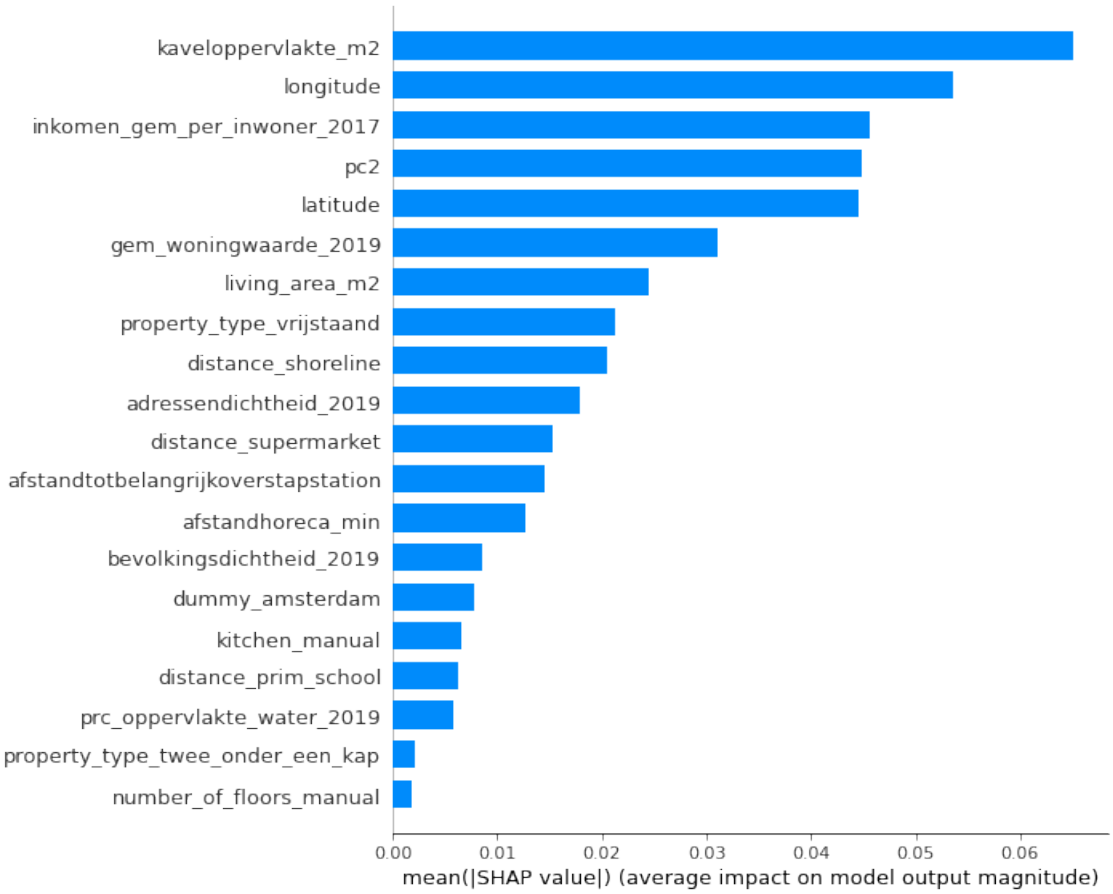


Figure 9: Feature importance of the XGBoost model with the extra features.