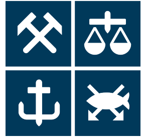
NHH



Norwegian School of Economics
Bergen, Fall 2020

Machine Learning as a Tool for Improved Housing Price Prediction

The applicability of machine learning in housing price prediction and the economic implications of improvement to prediction accuracy

Henrik I W. Wolstad and Didrik Dewan

Supervisor: Jonas Andersson

Master thesis, Economics and Business Administration

Major: Business Analytics and Financial Economics

NORWEGIAN SCHOOL OF ECONOMICS

This thesis was written as a part of the Master of Science in Economics and Business Administration at NHH. Please note that neither the institution nor the examiners are responsible – through the approval of this thesis – for the theories and methods used, or results and conclusions drawn in this work.

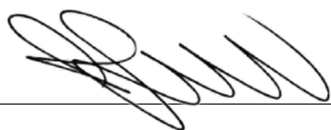
Acknowledgements

This thesis is written as a part of the Master of Science in Economics and Business Administration with specialization in Business Analytics and Financial Economics at the Norwegian School of Economics.

We would like to express our sincere gratitude to our supervisor Jonas Andersson for providing us with invaluable guidance throughout the process of writing this thesis. His extensive knowledge within the field of statistical modeling has been integral for our research. We would also like to thank Anders Francke Lund at Eiendomsverdi for providing the data used in this thesis and for valuable insights about the Norwegian real estate market. Finally, we thank each other for a productive, challenging and rewarding semester.

Norwegian School of Economics

Bergen, December 2020



Henrik I W. Wolstad

Didrik Dewan

Abstract

This thesis investigates whether non-linear machine learning algorithms can produce more accurate predictions of Norwegian housing prices compared to linear regression models. We find that the non-linear XGBoost algorithm increases out-of-sample prediction accuracy by 8.5% in terms of Root Mean Squared Error compared to the linear model used by Statistics Norway. Using additional property-specific and macroeconomic variables such as coordinates, common debt, story, inflation rate and interest rate, we find that a non-linear Stacked Regression model improves out-of-sample prediction accuracy by 39.52% in terms of Root Mean Squared Error compared to a linear model.

Keywords – Housing price prediction, AVM, Machine Learning, Deep Learning, XGBoost, Deep Neural Network, Stacked Regression, Random Forest

Contents

1 Introduction 1

2 Background and Theory 4
2.1 Literature Review . . . 4
2.2 Machine Learning Fundamentals . . . 6
2.2.1 Prediction Versus Inference . . . 6
2.2.2 Data Partitioning . . . 7
2.2.3 Overfitting . . . 8
2.2.4 Bias-Variance Trade-Off . . . 8
2.2.5 Resampling Methods . . . 10
2.3 Models . . . 12
2.3.1 Linear Regression . . . 12
2.3.2 Random Forest . . . 13
2.3.3 Extreme Gradient Boosting . . . 15
2.3.4 Deep Feedforward Neural Network . . . 17
2.3.4.1 Hyperparameters and Tuning . . . 20
2.3.5 Stacked Regression . . . 23

3 Data 26
3.1 Phase 1: SSB Replication . . . 26
3.1.1 Data Replication and Descriptive Statistics . . . 26
3.2 Phase 2: Additional Variables . . . 29
3.2.1 Data Cleaning and Feature Engineering . . . 29
3.2.2 Outliers . . . 32
3.2.3 Descriptive Statistics . . . 34

4 Methodology 38
4.1 Approach . . . 38
4.1.1 Phase 1: SSB Replication . . . 38
4.1.2 Phase 2: Additional Variables . . . 39
4.2 Model Selection . . . 39
4.2.1 Software and Computing Platforms . . . 40
4.2.2 Model Training . . . 40
4.2.2.1 Data Partitioning and Resampling Methods . . . 40
4.2.2.2 Hyperparameter Tuning . . . 41
4.3 Model Assessment . . . 43

5 Analysis 45
5.1 Phase 1: SSB Replication . . . 45
5.1.1 Model Assessment . . . 45
5.1.1.1 Accuracy Distribution . . . 46
5.1.1.2 Computing Time . . . 47
5.1.1.3 Model Interpretability . . . 49
5.2 Phase 2: Additional Variables . . . 52
5.2.1 Model Assessment . . . 52
5.2.1.1 Accuracy Distribution . . . 53

5.2.1.2 Computing Time 54
5.2.1.3 Model Interpretability 55
5.3 Summary of Results 59

6 Discussion **65**
6.1 The Economic Implications of Improved Accuracy 65
6.1.1 Property and Wealth Tax Estimation 65
6.1.1.1 Property Taxes 66
6.1.1.2 Wealth Tax 70
6.1.2 Other Applications 70
6.2 Interpretability and Industry Acceptance 73
6.3 Limitations and Further Research 73

7 Conclusion **77**

References **79**

Appendix **85**
A1 Variable Description and Imputation Methods 85
A2 Anomaly Detection with Isolation Forest 86
A3 Descriptive Statistics 88
A4 The Boruta Algorithm 89
A5 Feature Selection with the Boruta Algorithm 90
A6 SHapley Additive exPlanations 92
A6.1 SHAP Plots 95
A7 Random Forest Hyperparameter Tuning Grids 95
A8 XGBoost Hyperparameter Tuning Values 96
A9 DFNN Hyperparameter Tuning Values 97
A10 Predicted Versus Actual Values 98
A11 Phase 1 Model Selection 99
A11.1 XGBoost 99
A11.2 Random Forest 100
A11.3 Deep Feedforward Neural Network 101
A11.4 Linear Regression 103
A11.5 Stacked Regression 103
A12 Phase 2 Model Selection 107
A12.1 XGBoost 107
A12.2 Random Forest 108
A12.3 Deep Feedforward Neural Network 108
A12.4 Linear Regression 110
A12.5 Stacked Regression 112
A13 Property Tax Calculation Procedure 115

List of Figures

2.1 Visual illustration of the bias-variance trade-off 10
2.2 Visual illustration of a Regression Tree 14
2.3 Visual Illustration of a Single-layer FNN 18
3.1 Anomaly plot of the response variable using Isolation Forest 34
5.1 SHAP Summary plot Phase 1 51
5.2 SHAP Summary plot Phase 2 57
5.3 SHAP Dependence plot Phase 2 59
A5.1 Feature selection using the Boruta algorithm 91
A10.1 Predicted versus actual values - Phase 1 98
A10.2 Predicted versus actual values - Phase 2 99

List of Tables

2.1 Main Hyperparameters in XGBoost 16
2.2 DFNN Hyperparameters 21
3.1 SSB's housing criteria 27
3.2 Descriptive statistics for Phase 1 29
3.3 Descriptive statistics for a subset of the continuous variables used in Phase 2 36
3.4 Descriptive statistics for a subset of the dummy variables used in Phase 2 37
4.1 Hyperparameter Tuning Methods and Combinations Sampled 42
5.1 RMSE of final models 45
5.2 Accuracy distribution for XGBoost and SSB 47
5.3 Model Tuning and Computing Time Comparison 48
5.4 RMSE of final models - Phase 2 52
5.5 Accuracy distribution for Stacked Regression and LM for Phase 2 54
5.6 Model Tuning and Computing Time Comparison for Phase 2 55
5.7 Summary of results from Phase 1 and Phase 2 60
5.8 Comparison of RMSE between Stacked and SSB's model 62
5.9 Total property value estimates 63
6.1 Property and Wealth taxes Applicable to Oslo 66
6.2 Total Tax Liability per Model 67
6.3 Proportion of Properties Liable for Property Tax 68
6.4 Property tax estimation by quartiles of actual property values 69
A1.1 Variable description and imputation methods 86
A3.1 Descriptive statistics for all continuous variables in Phase 2 88
A3.2 Descriptive statistics for all dummy variables in Phase 2 89
A7.1 Random Forest Hyperparameter Tuning Values - Phase 1 95
A7.2 Random Forest Hyperparameter Tuning Values - Phase 2 95
A8.1 XGBoost Hyperparameter Tuning Values - Phase 1 96
A8.2 XGBoost Hyperparameter Tuning Values - Phase 2 96
A9.1 DFNN Hyperparameter Tuning Values - Phase 1 & 2 97
A11.1 XGBoost final hyperparameters 100
A11.2 Random Forest final hyperparameters 101
A11.3 Deep Feedforward Neural Network Tuned Hyperparameters 102
A11.4 Linear Regression Coefficients 103
A11.5 Stacked Regression Phase 1 - Final regularization parameters and weights 105
A11.6 Stacked model configuration - Phase 1 106
A12.1 XGBoost final hyperparameters - Phase 2 107
A12.2 Random Forest Hyperparameter Configuration - Phase 2 108
A12.3 Deep Feedforward Neural Network Tuned Hyperparameters - Phase 2 . . 109
A12.4 Linear Regression Coefficients in Phase 2 111
A12.5 Stacked Regression - Final regularization parameters and weights - Phase 2 113
A12.6 Stacked model configuration - Phase 2 114

1 Introduction

In this thesis we investigate how well non-linear machine learning models are suited for predicting housing prices in the Norwegian residential property market. We do so by training four non-linear machine learning models on property transaction data for five Oslo boroughs and comparing their performance to a linear regression benchmark.

Estimating the market value of housing has several practical economic applications, including taxation, mortgage refinancing, and risk management in banks. It is therefore important for organizations and individuals to obtain accurate market values for housing. However, estimating the market value of a property without a sale taking place is a considerable challenge due to the multitude of factors unique to each property. Physical appraisals of market values based on a broker's or appraiser's expertise is one of the most basic ways of estimating housing values in lieu of a transaction. However, such appraisals are time consuming and expensive, and often underestimate true market value (Benedictow and Walbækken, 2020). For this reason, different Automatic Valuation Models (AVM) have been developed to quickly and cheaply estimate property values.

AVMs must often follow strict requirements for interpretability and transparency. Because of this, such models are often developed using a few intuitive variables and simple estimating procedures. Consequently, much data about the individual properties and the macroeconomic conditions are excluded. Even in applications where interpretability is not a major concern, limitations in computing power and data availability have until recently inhibited adoption of more sophisticated and complex prediction models. Recent technological advancements in computing power, software development and big data (Chollet and Allaire, 2017) are making machine learning algorithms much more pragmatic both in terms of implementation and interpretation. This makes them an increasingly viable alternative to more conventional AVM approaches such as linear regression models and repeat sales methods.

Previous research on the topic of housing price prediction has applied machine learning algorithms to predict housing prices for the US and Asian residential housing market (Park and Bae, 2015; Chen et al., 2017; Lu et al., 2017; Truong et al., 2020). Comparisons between non-linear machine learning models and linear regression models have also been

conducted on real estate markets outside Norway (Tay and Ho, 1992; Do and Grudnitski, 1992; Lenk et al., 1997). In the Norwegian setting, Birkeland and D'Silva (2018) used a Stacked Regression approach to develop an AVM for the residential real-estate market in Oslo, and compared its performance against estimates by real estate agents.

In this thesis we expand on existing literature by investigating the prediction accuracy of a diverse selection of non-linear machine learning models against a linear regression AVM in a Norwegian setting, and by discussing the results in an economic context with consideration to model interpretability computational costs.

Our research question is:

Can non-linear machine learning models improve housing price predictions compared to linear regression models in the Norwegian residential housing market?

To investigate this, we predict residential housing prices by training four non-linear machine learning models on second-hand property transactions for five Oslo boroughs in the period 2005-2020. The algorithms employed are Random Forest, XGBoost, Deep Feedforward Neural Network and Stacked Regression. We compare the performance of said models to a benchmark linear regression model developed by Statistics Norway (SSB) for applications in taxation.

Our analysis is split into two phases. In Phase 1, we replicate the data filtering process employed by SSB and compare our models' performance against the benchmark SSB model. This is done to ensure a reliable comparison with the model currently employed by SSB. In Phase 2, we train the same models on an expanded data set that follows a different pre-processing and feature engineering regime. The expanded data set includes additional property-specific and macroeconomic variables such as inflation, coordinates, common debt, story and interest rates. In this manner, Phase 1 investigates the isolated effect of adopting non-linear machine learning models, while Phase 2 investigates the possible prediction accuracy gains that can be obtained from combining use of non-linear machine learning models with inclusion of additional property-specific and macroeconomic data.

Our findings indicate that significant improvements in prediction accuracy can be gained by switching from linear to non-linear models. In Phase 1, all machine learning models

outperform the linear benchmark model. XGBoost demonstrates the largest improvement, increasing out-of-sample prediction accuracy in terms of Root Mean Squared Error (RMSE) by 8.5% over the SSB benchmark. The performance difference between the machine learning models and the benchmark model is amplified in Phase 2. In this phase, the Stacked Regression model has the best out-of-sample performance with 39.52% increased accuracy in terms of RMSE compared to the linear benchmark model. This suggests that there exist non-linear relationships between housing prices and property-specific and macroeconomic variables, which the linear model naturally fails to capture.

The increased accuracy of the non-linear machine learning models comes at a cost. Training the machine learning models is several orders of magnitude slower than training the linear model in both phases due to increased computational complexity. Fitting the linear model is done in less than a second, while tuning and training the machine learning models takes between 2-35 hours depending on the model. The black-box nature of these algorithms also leads to a loss of interpretability compared with our benchmark model and other existing valuation methods. This poses perhaps the greatest challenge to adoption of non-linear models since many practical applications are subject to customer requirements for transparency in the models employed.

Nonetheless, we find that increased prediction accuracy may yield economic gains in applications such as taxation, property transactions, mortgage refinancing, risk management and monitoring of property-backed financial instruments. Preliminary estimates of property tax effects for Oslo using our best performing model show a reduction in the proportion of properties unduly charged with property taxes due to overestimation, as well as less underestimation of taxes for the most expensive properties compared to the benchmark SSB model[1].

[1] Which today is applied by tax authorities

2 Background and Theory

This chapter covers relevant literature and theory for our analysis. We begin by reviewing existing literature on housing price prediction and presenting the benchmark model used in our analysis. This is followed by a theoretical section in which we elaborate upon what machine learning is and the most important theoretical concepts in this regard. Lastly, we present the theoretical framework behind the models used in this thesis.

2.1 Literature Review

The theoretical foundation of housing- and real estate valuations are based on *hedonic price theory*, which states that a commodity can be viewed as a bundle of attributes or characteristics (Griliches, 1971). Rosen (1974) and Lancaster (1966) were the first to develop hedonic price models, an approach by which the price of a good is estimated based on the implicit prices of its attributes, which in turn can be estimated from observed prices of differentiated goods (Rosen, 1974). For housing, typical attributes include size, age, location and the number of bedrooms. Since the inception of hedonic price models a considerable amount of research has been devoted to investigate the efficacy of hedonic models for residential property valuation by means of standard regression methods (Kang and Reichert, 1991; Birch et al., 1991; de Haan and Diewert, 2011).

In Norway hedonic models have been used to develop AVMs for the residential real estate market as a cost-effective alternative to physical appraisals. SSB has developed their own valuation model for residential properties (Takle and Melby, 2020). SSB estimates the average price per square meter of a property based on property size, location, age, year and urbanization. The average market value can then be found by multiplying the estimated value by the size of the property. This model is currently used by Norwegian tax authorities to estimate property values for tax purposes (Takle and Melby, 2020). Since the model is used as a basis for estimating wealth and property taxes for households and individuals, its accuracy and reliability is important. In this thesis we use the exact model developed by SSB as a benchmark when evaluating the prediction accuracy of non-linear machine learning algorithms.

Due to the significant increase in computing power and availability of data in the last

couple of decades, more sophisticated machine learning algorithms have been employed in an attempt to achieve more accurate predictions for housing prices. Tay and Ho (1992) were among the first to train an Artificial Neural Network (ANN) to predict residential apartment prices in Singapore. They found that the ANN outperformed a standard multiple regression model in terms of mean percentage error. Others have argued against the application of neural networks in housing price prediction. Worzala et al. (1995) investigated the extent to which ANNs could be used as a tool for automatic real estate appraisals. Due to inconsistent predictions between software packages and model runs, the authors deemed ANNs to be unsuitable as an automatic appraisal tool. Lenk et al. (1997) concluded that there were significant estimation error costs associated with ANNs, and that such models did not consistently outperform standard multiple regression models across different data sets and accuracy metrics.

Despite this, there seems to be growing consensus within the literature that non-linear machine learning techniques and neural networks are able to consistently outperform standard linear regression models. Limsombunchai (2004) compared an ANN to a hedonic price model and found that even with a small sample size of 200 dwellings, the ANN was able to outperform a standard hedonic regression model. Park and Bae (2015) utilized several machine learning algorithms such as Repeated Incremental Pruning to Produce Error Reduction (RIPPER), Adaptive Boosting (AdaBoost) and Naïve Bayesian to predict housing prices in Virginia, and found that the RIPPER algorithm consistently outperformed the other models. Using time series data for the period 2004-2016 for major Chinese cities, Chen et al. (2017) found that a Long Short-Term Memory (LSTM) Neural Network outperformed an Autoregressive Integrated Moving Average (ARIMA) model.

In more recent years, *Stacked Regression models*[2] have become increasingly popular in the literature as well as in machine learning competitions (Kaggle, 2020a). Through extensive feature engineering and hyperparameter tuning, Lu et al. (2017) were able to predict housing prices with remarkable accuracy by stacking Lasso and XGBoost regression models. Truong et al. (2020) compared the performance of a Stacked Generalization regression model against popular machine learning techniques such as Random Forests (RF), XGBoost and LightGBM, and found that the Stacked model outperformed any individual model at the expense of a higher time complexity. Birkeland and D'Silva (2018)

[2]Models that combine several base learners (i.e. individual models).

used a Stacked Generalization approach consisting of four ensemble methods and a repeat sales index method to develop an AVM for the residential real estate market in Oslo. They found that the performance of the Stacked model was comparable to price estimates by local real estate agents, with a median absolute percentage error of 5.4%.

2.2 Machine Learning Fundamentals

Machine learning refers to the process of using statistical tools to learn from and understand data (James et al., 2013). These tools are usually divided into two main categories, namely *supervised* and *unsupervised* (James et al., 2013). The former refers to the process of building a statistical model to predict or estimate a predefined output variable based on one or more input variables (predictors). By feeding these statistical models data they are able to recognize and learn from complex patterns and relationships in the data, which are then used to make predictions (James et al., 2013).

In unsupervised learning there is no predefined output (James et al., 2013). Consequently, the goal is not to predict a predefined output based on the available variables, but rather to explore the data in order to identify relationships and structures in the data (James et al., 2013). The models employed in this thesis fall into the category of supervised learning, as we are predicting a predefined output (i.e. housing prices) based on a set of predictors.

2.2.1 Prediction Versus Inference

In general, the goal of any supervised learning process is to predict a quantitative or qualitative variable y based on a set of p predictors $x_1, x_2, ..., x_p$. Moreover, we assume that there exists a relationship between y and x_p. In the most simple case, this relationship can be expressed as

$$y = f(x_p) + \epsilon \tag{2.1}$$

Where f is an unknown but fixed function of $x_1, ..., x_p$, and ϵ is an error term that captures all variables that are associated with y but are not included in the model. In statistics, we assume that the error term is independent of x_p and has a mean of zero. In other words,

f is an (unknown) function that maps the relationship between the response variable[3] and the predictors. Since the function is unknown, we must estimate this function based on the observed data points. In statistical analysis, there are two main areas of interest when it comes to estimating the function f, namely *inference* and *prediction*.

Machine learning is mainly interested in prediction. More specifically, it is interested in identifying the set of predictors that yield the most accurate predictions for the output y, and less concerned about the nature of the relationship. In other words, it is irrelevant whether there is a causal relationship between a predictor x_p and the response variable y, as long as the the predictive power of the predictor is high and consistent. Consequently, it is not necessary to make assumptions about the data and the exact form of f. Since the error term is zero on average, we can predict y based on the set of predictors such that

$$\hat{y} = \hat{f}(x) \tag{2.2}$$

Where $\hat{f}$ is the estimated function f and $\hat{y}$ is the predicted values for y. In machine learning, the goal is typically to estimate a function $\hat{f}$ that minimizes the prediction error.[4]

2.2.2 Data Partitioning

When implementing machine learning models for prediction purposes, proper data partitioning is crucial in order to be able to objectively evaluate the models' performance and to avoid data leakage. In data rich situations Hastie et al. (2009) recommends splitting the data randomly into three groups: a *training set*, a *validation set* and a *test set*. The training set, as the name suggests, is used to train and fit the models. The validation set is used to obtain the optimal hyperparameter values and assist in model selection, and the test set is used to evaluate the out-of-sample performance of the final model.

There are no clear rules regarding the respective size of the different groups, as this largely depends on the availability and the signal-to-noise ratio[5] of the data (Hastie et al., 2009). However, the training set typically constitutes the largest fraction of the data, as machine

[3] Also referred to as independent variable or output variable.
[4] The difference between the actual and predicted values.
[5] The amount of relevant data relative to the amount of random irrelevant data.

learning models need to be trained on large amounts of data in order to be effective (James et al., 2013). The actual split between training and test data will be subject to the *bias-variance trade-off*, which we discuss in detail in Section 2.2.4. A larger training set will lead to lower bias and higher variance, and vice versa. A large training set will also increase computation time during training relative to a smaller one. Hence, the split chosen is a trade-off between bias, variance and computation time. In this thesis, we use a random sample of 70% of the data as the training set, whereas the test set constitutes the remaining 30%. We find that this split provides a good balance between bias, variance and computing time.

2.2.3 Overfitting

Overfitting refers to a situation in which a statistical model follows the *noise*[6] or errors too closely, instead of following the signal[7] in the data (James et al., 2013; Mullainathan and Spiess, 2017). If a machine learning model demonstrates very good performance on the training data (low training error) but performs poorly when tested on new data (high test error), this is usually an indication that the model is overfitting. This is unfortunate because it means that the model conforms too much to the training data and is unable to generalize feature relationships to new data.

2.2.4 Bias-Variance Trade-Off

In machine learning, the relationship between model complexity, training and test error is the result of the two competing properties *bias* and *variance* (James et al., 2013). Bias refers to the error that is introduced when trying to use a simple model to solve a complex real-world problem (James et al., 2013). In other words, it is the inability of a machine learning model to capture the true relationship in the data. For instance, if we are trying to use linear regression to estimate a non-linear relationship, the model will have high bias. This is because a straight line can never be flexible enough to capture a non-linear relationship. Variance refers to the amount by which the function $\hat{f}$ would change if it was estimated using a different set of observations (James et al., 2013). As such, variance is the difference in fit between data sets. When a model is overfitting, for example, it has

[6]Random, irrelevant information in the data.
[7]Relevant, meaningful information that the model is trying to detect.

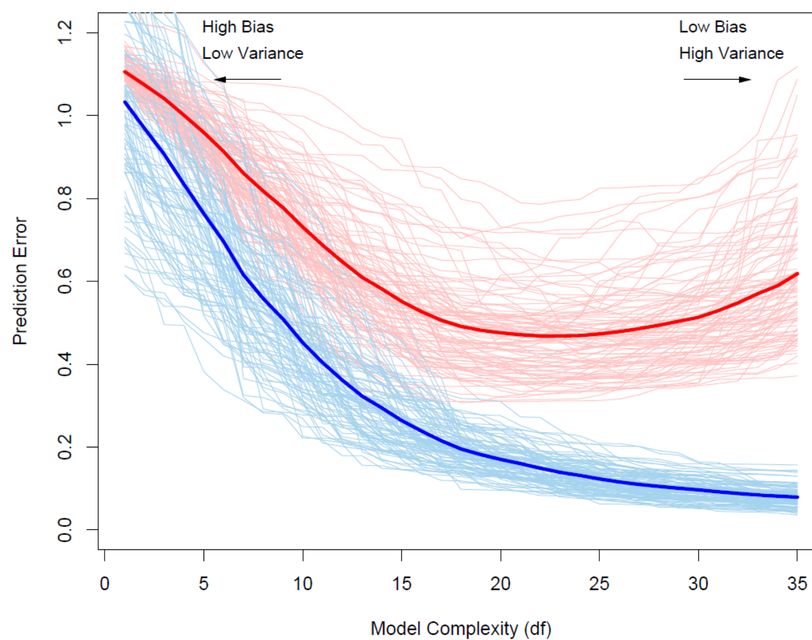
high variance, as the prediction error is vastly different for the training and test set.

The idea of the bias-variance trade-off is that we can reduce the training error of any machine learning model by increasing the *complexity* of the model, but we cannot reduce the test error (James et al., 2013). The training error is reduced because a higher complexity enables the model to follow the data more closely. The test error, on the other hand, is the sum of the variance and squared bias of the estimated function $\hat{f}(x)$, plus the variance of the error term ϵ (James et al., 2013). Thus, to minimize the test error, we need to select the model with lowest possible bias *and* variance. In general, more complex models have higher variance (James et al., 2013). This is because a complex model is able to follow the specific data it was fit on more closely. However, because a complex model follows the data more closely, it is also more likely to represent the true relationship in the training data, resulting in lower bias. Hence, selecting a model with relatively lower bias can only be achieved at the expense of higher variance.

This trade-off is illustrated in Figure 2.2.4, which shows the relationship between model complexity, bias and variance. The plot shows that after a certain point we cannot achieve a lower test error by increasing model complexity. Instead, the test error will follow a U-shape because of the bias-variance trade-off.

Figure 2.1: Visual illustration of the bias-variance trade-off

This figure shows the behaviour of the training and test error as the model complexity increases. The blue curves illustrate the training error and the red curves illustrate the test error. As complexity increases, both the training error and test error initially decrease. The test error initially decreases because the bias decreases at a higher rate than the variance increases to begin with. After a certain point, increasing complexity has little impact on the bias, but a large effect on the variance. Consequently, the test error starts to increase beyond this point. The figure is reprinted from Hastie et al. (2009) with permission from the authors.



2.2.5 Resampling Methods

Often there is not enough data to follow the recommendation by Hastie et al. (2009) to split the data three ways. In such cases, we can utilize clever *resampling methods* that replicate the validation set by efficient re-use of the training data. The most widely used resampling methods are *Validation Set*, *K-Fold Cross-Validation* and *Leave-One-Out Cross-Validation*.

The validation set approach is the most basic approach of the three, and involves splitting the data randomly into a training set and a test set. The training set typically accounts for 50-80 % of the data, whereas the remaining observations make up the test set (James et al., 2013). As before, the training set is used to fit the models and the test set is used to validate the predictions of the final model.

The rationale behind k-Fold Cross-Validation (k-fold CV) is to split the training data into k groups, or *folds*, of roughly the same size. The first k fold is held out and used as a validation set for the predictions, whereas the remaining $k-1$ folds are used to fit the model (James et al., 2013). The prediction error[8] is computed using the observations in the hold-out fold. The procedure is then repeated k times; each time, a new fold is treated as the hold-out fold and a corresponding prediction error is computed.

Instead of arbitrarily splitting the data into k parts, Leave-One-Out Cross-Validation (LOOCV) only uses a single data point (x_i, y_i) for the validation set, and the remaining $N-1$ observations constitute the training set and are used to fit the model (James et al., 2013). LOOCV is a special case of k-fold CV in which $k = n$. The single data point i is then used to predict the response. This procedure is repeated N times, until all observations have been excluded exactly once (James et al., 2013).

In this thesis we opt for k-fold CV in the model selection stage where this is possible to implement. For models where this is not feasible[9], we use the validation set approach. These methods are preferred because they have a major computational advantage over LOOCV as long as $k < n$ and the training split used for the validation set accounts for less than $\frac{N-1}{N}$ of the data (James et al., 2013). Additionally, given the appropriate value for k and training split, both methods are found to provide a good balance between bias and variance.

Regarding k-fold CV, there is no optimal value for the number of folds k. A higher value will be more computationally expensive than a lower value (James et al., 2013). Apart from the computational aspect, the number of folds to use is also subject to the bias-variance trade-off. In general, a large value for k will on average yield a less biased estimate of the validation error, but the estimates will in turn have higher variance (James et al., 2013), and vice versa. We set $k = 5$ for our analyses as we find this to provide a good balance between computing time, bias and variance.

[8]Error metric, for example RMSE.

[9]The Deep Feedforward Neural Network. In our case due to R package limitations.

2.3 Models

Machine learning encompasses a wide range of algorithms with different characteristics. In this thesis we implement a selection of four models in addition to the benchmark linear regression model: Random Forest, XGBoost, Deep Feedforward Neural Network, and Stacked Regression. They are among the best-performing and most widely used machine learning algorithms employed in industry, research and competitions.

2.3.1 Linear Regression

In this thesis we use *Linear Regression* as the benchmark model for both phases. For Phase 1, this corresponds to replicating SSB's AVM model. In the most simple case linear regression involves predicting a quantitative response variable Y based on a single predictor X. However, this can easily be extended to cases with multiple predictors, where it is referred to as multiple regression. The fundamental idea of multiple regression is to fit a hyperplane through the data points by assuming a linear relationship between the response variable and the predictors (James et al., 2013). If we have a set of p predictors, the relationship between the response and the predictors can be expressed as:

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + ... + \beta_p X_p + \epsilon \tag{2.3}$$

Where Y is the response variable that we want to predict, X_j is the jth predictor, β_j is the regression coefficient for a given predictor and ϵ is the error term that captures all factors that are associated with the response but not included in the model. The regression coefficients are not known in advance and therefore have to be estimated. The procedure used to estimate the coefficients is known as *Ordinary Least Squares* (OLS). In short, OLS chooses the coefficients such that the sum of squared residuals[10] (RSS) is minimized, as shown in Equation 2.4.

$$RSS = \sum_{i=1}^{n} (y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij})^2 \tag{2.4}$$

[10]Sum of the squared difference between the observed value of the response and the predicted values.

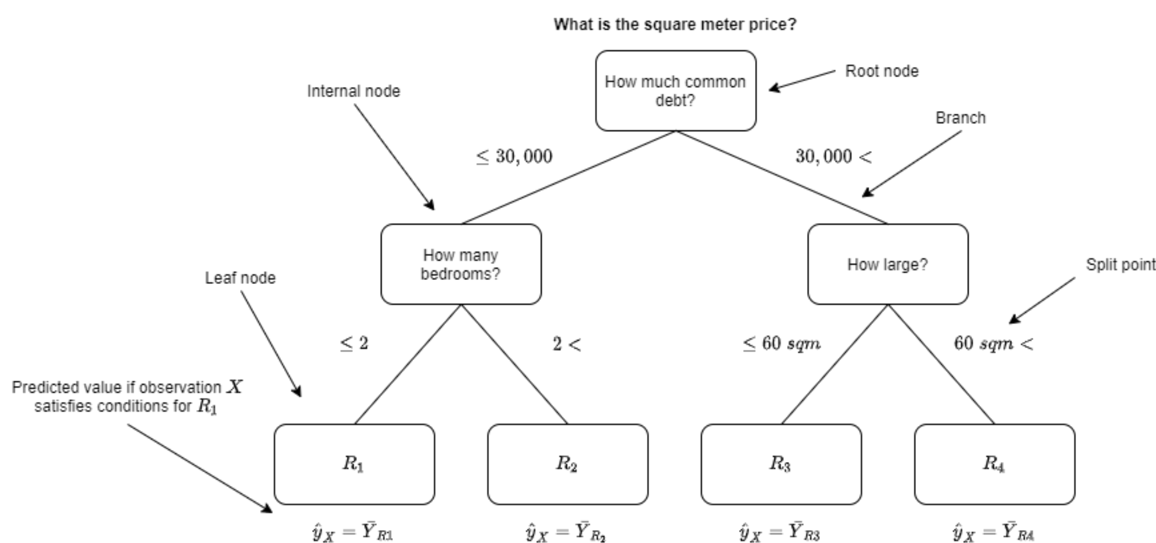
2.3.2 Random Forest

The Random Forest algorithm is a machine learning method that can be used for both regression and classification problems. The model is versatile and widely recognized for its robustness and performance, and is a popular choice in Kaggle competitions (Chollet and Allaire, 2017; Kaggle, 2012).

Random Forest regression models are based on regression trees. Essentially, a regression tree is an algorithm that splits the predictor space into a number of non-overlapping regions $R_1, R_2, ..., R_J$ according to a set of splitting rules, and makes predictions based on which region a particular observation belongs to (James et al., 2013). Each rule takes the form of a logical query that checks whether the value of the splitting predictor variable is higher or lower than the chosen *split point*, and subsequently assigns the observation to one of two predictor space regions.

A *Recursive Binary Splitting* approach is used to select the predictor variable and split point at each node. This approach starts with all observations belonging to the same region and successively splits the predictor space at each step using the predictor and split point that maximises the reduction in the loss function at that step (James et al., 2013). We can visualize these splitting rules as trees, where each split point is associated with a *node* that branches off into two non-overlapping predictor spaces. The depth of the tree decides how many sequential splitting rules a decision tree has before an observation reaches a *leaf node* representing one of the final R_J regions. The final predicted values for each observation assigned to a region R_j corresponds to the mean of the response variable of the training observations in that region. Figure 2.2 shows a simple illustration of a hypothetical regression tree for our problem.

Figure 2.2: Visual illustration of a Regression Tree



A problem with independent regression trees is that they often exhibit high variance (James et al., 2013). The Random Forest algorithm is therefore based on the *bootstrap aggregated (bagged)* decision tree approach, which involves building a large number of parallel decision trees on bootstrapped training samples and then averaging their outputs (James et al., 2013). This bagging procedure reduces the variance in the overall model. Although each tree will have high variance, the bootstrap method allows us to build many trees that, taken together, have low variance. In contrast to regular bagged trees, Random Forest only considers a limited number of predictors at each split. Each time a split is considered a random sample of m predictors is chosen as candidates. As in a regular regression tree, the predictor that contributes the most to reducing the loss function is chosen as the split predictor at that node. The rationale behind the random selection of predictors is to prevent the trees from choosing the same strong predictor in the top split, which would lead to very similar trees with highly correlated predictions and thus a model with high variance.

Using notation from the *randomForest* R package (Breiman et al., 2018), the algorithm's two hyperparameters are the number of trees to grow *(ntrees)* and the number of predictors to consider at each split *(mtry)*. *mtry* is the only hyperparameter that needs tuning, since the Strong Law of Large Numbers prevents the model from overfitting due to too many trees (Breiman, 2001). It is therefore sufficient to set a large enough number of trees to ensure convergence, and only tune for *mtry* (Breiman, 2001; James et al., 2013).

The *mtry* parameter has a minimum value of one and a maximum value equal to the number of predictor variables, p. To ensure that the model is aligned with the principles of the algorithm, it should never equal p, as a Random Forest with $mtry = p$ constitutes standard bagging. According to Breiman et al. (2018), the suggested value for *mtry* in regression problems is $p/3$. However, tuning the parameter model can yield performance gains.

2.3.3 Extreme Gradient Boosting

Extreme Gradient Boosting (XGBoost) is another approach that has gained much recognition and success in research and competitions (Kaggle, 2020b). XGBoost was developed by Chen and Guestrin (2016), and is largely based on the Gradient Tree Boosting algorithm by Friedman (2001, 1999). Like Random Forest, Gradient Boosting is based on ensembling the outputs of many weak base learners to collectively construct a strong prediction model. In our case these base learners are regression trees. However, in contrast to the parallel development of regression trees in Random Forests, Gradient Boosting applies an additive approach where trees are sequentially fitted, using information from the previously fitted models to "boost" the current model's performance. More specifically, each successive tree is fitted to the residuals of the previous iteration of the current model, these residuals being the gradient of the loss function being minimized at each particular step (Hastie et al., 2009; Friedman et al., 2000). In this manner, each iteration, or additional tree, picks up variance that has not yet been captured by previous trees. This incrementally improves the overall model in areas where it does not perform well (James et al., 2013).

The XGBoost algorithm is an advanced implementation of the Gradient Boosting algorithm. While based on the same algorithm, XGBoost introduces several enhancements and improvements, including advanced tree construction algorithms and the ability to perform column subsampling (Chen and Guestrin, 2016). Inspired by the traditional row subsampling employed in Gradient Boosting, column subsampling introduces randomness to the learning procedure in an attempt to combat overfitting and reduce computational requirements. Another key difference between XGBoost and traditional Gradient Boosting is that the former uses a more regularized model formulation to prevent overfitting, which has been shown to yield significant performance gains (Chen and Guestrin, 2016). In

terms of speed, it is also substantially faster than regular Gradient Boosting, as it allows for parallel processing.

Mathematically, the model tries to minimize the regularized objective shown in Equation 2.5 using gradient boosting (Chen and Guestrin, 2016).

$$\begin{aligned} L(\phi) &= \sum_i l(\hat{y}_i, y_i) + \sum_k \Omega(f_k) \\ \text{where} \quad \Omega(f) &= \gamma T + \frac{1}{2}\lambda \parallel w \parallel^2 \end{aligned} \tag{2.5}$$

In this setting, $l(\hat{y}_i, y_i)$ is the loss function measuring the difference between predicted values and the actual values, while $\Omega(f_k)$ is a term to penalize complexity. Each f_k represents a tree with structure q, T leaves (end nodes), and w leaf weights. γ and λ are regularization terms to reduce complexity. This setup reflects the bias-variance tradeoff and aims to ensure that the final model balances a solution that has both low variance and low bias (Chen, 2014).

We use notation from the R library *XGBoost* (Chen et al., 2020) to describe the algorithm's hyperparameters. A summary of the hyperparameters used in this thesis can be seen in Table 2.1.

Table 2.1: Main Hyperparameters in XGBoost

Parameter	Range	Description
eta	[0, 1]	Learning rate
gamma	[0, ∞]	Minimum loss reduction required at leaf nodes
max_depth	[0, ∞]	Maximum depth of a tree
min_child_weight	[0, ∞]	Minimum sum of instance weight in child
num_round	[0, ∞]	Number of trees
early_stopping_rounds	[0, ∞]	Early stopping parameter
subsample	[0, 1]	Ratio of training set sampled for each tree
colsample_bytree	[0, 1]	Ratio of columns sampled for each tree

eta is the model's learning rate, which functions as a factor that scales each additional tree's weights as it is added to the current model. In effect, it reduces each individual tree's influence, leaving room for improvement by growing additional trees. It thereby acts to reduce overfitting (Chen and Guestrin, 2016). Typical values are 0.01 or 0.001 (Hastie et al., 2009), reflecting the findings by Friedman (2001) that smaller values in the

learning rate improve the overall results. A smaller learning rate will on the other hand require a larger number of trees to be grown, since each individual tree will have less of an impact on the overall model.

num_round sets the total number of trees to grow, which must be large enough to ensure strong predictions but low enough to avoid overfitting. An appropriate value can be found through cross-validation.

max_depth, *gamma*, and *min_child_weight* relate to the structure of each tree. *gamma* sets the minimum loss reduction required to make a split at any given node on a tree. Larger values make the model more conservative and reduce complexity. Similarly, *min_child_weight* sets a criteria for the minimum sum of instance weight needed in a child node for splitting to continue. Larger values increase conservatism and reduce complexity (DML, 2020). *max_depth* sets the maximum depth of each tree, meaning the maximum number of splits possible. This also serves as a control for the interaction order of the model, since it sets how many variables can be involved in splits in each tree.

The final two parameters are regularization parameters intended to prevent overfitting. *subsample* sets the ratio of the training set to be sampled for each tree, while *colsample_bytree* sets the ratio of columns to be sampled for each tree (DML, 2020).

2.3.4 Deep Feedforward Neural Network

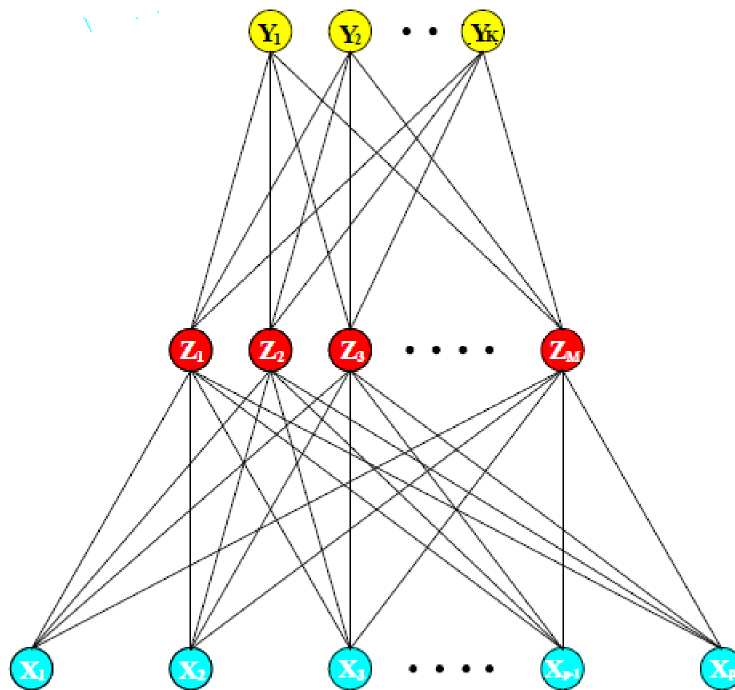
Deep Feedforward Neural Networks (DFNN) are a type of ANN, which constitutes a class of supervised learning techniques that have proved themselves excellent at prediction in classification and regression problems. These techniques currently dominate Kaggle competitions alongside boosted models and ensemble methods such as XGBoost and Stacked Regressions (Chollet and Allaire, 2017; Kaggle, 2020c).

DFNNs are organized in a layer-based manner, consisting of an input layer, two or more hidden layers, and an output layer (Goodfellow et al., 2016). These layers are arranged in a chain structure, so that each layer is a function of the preceding layer. Data is fed forward from the input layer to the hidden layers where it is transformed, and finally to the output layer to produce the final predictions. Figure 2.3 shows a diagram of a classic Feedforward Neural Network[11] (FNN) with one hidden layer.

[11] A DFNN is a FNN with more than one hidden layer.

Figure 2.3: Visual Illustration of a Single-layer FNN

This figure shows a diagram of a classic FNN with one hidden layer. $X_1, X_2, X_3, ..., X_{P-1}, X_P$ are the input features corresponding to the input layer. $Z_1, Z_2, Z_3, ..., Z_M$ are the hidden units corresponding to the hidden layer. $Y_1, Y_2, ..., Y_K$ are the output units corresponding to the output layer. For regression problems such as ours there is only one output unit, corresponding to the predicted value. Reprinted from Hastie et al. (2009) with permission from the authors.



Each layer consists of multiple *units*, also known as *neurons*. The units in the input layer correspond to the features (variables) in the data set. Each unit in the hidden layers, known as *hidden units*, represent linear combinations of its inputs. The connections between units are known as *weights* and represent the parameters that can be adjusted in order to increase or reduce the importance of a particular unit's output. In this thesis we use *dense* layers, meaning that each hidden unit in a layer is connected to all the units in the preceding and ensuing layers. A mathematical representation of the model can be seen in Equation 2.6 (Hastie et al., 2009).

$$
\begin{aligned}
Z_m &= \sigma(\alpha_{0m} + \alpha_m^T X),\ m = 1, ..., M, \\
T_k &= \beta_{0k} + \beta_k^T Z,\ k = 1, ..., K, \\
f_k(X) &= g_k(T),\ k = 1, ..., K
\end{aligned}
\tag{2.6}
$$

where $Z = (Z_1, Z_2, ..., Z_M)$ and $T = (T_1, T_2, ..., T_K)$

Each hidden unit corresponds to a derived feature Z_m that is created as a linear combination of the inputs. The target, $\hat{Y}_k$, is subsequently modeled as a function $f_k(X)$ of linear combinations of Z_m. It is common to add an additional *bias* unit that feeds in to each unit in the hidden and output layers. This bias corresponds to α_{0_m} and β_{0_k} in Equation 2.6. K is the number of units in the output layer, which corresponds to the number of prediction classes. For regression problems such as ours, we use $K = 1$, and the output function $g_k(T) = T_k$ (Hastie et al., 2009).

Each hidden unit Z_m's function parameters are learned from the data without any user input (Hastie et al., 2009). However, the activation function $\sigma(v)$ must be set by the user. In our case we will use the *Rectified Linear Unit* function (ReLU), which is in line with recommendations from leading publications (Goodfellow et al., 2016). The ReLU function takes the form of $g(z) = max\{0, z\}$ (Goodfellow et al., 2016), such that the derived feature Z_m from Equation 2.6 can be written as $Z_M = max\{0,\ \alpha_{0m} + \alpha_m^T X\}$.

The learning process in a DFNN consists of adjusting the values of the weights between all the Z_m units of the network so that the model fits the training data well. The complete set of weights is given by Equation 2.7 (Hastie et al., 2009).

$$\begin{aligned} \{\alpha_{0m}, \alpha_m; m = 1, 2, ..., M\}\ M(p+1) \text{ weights,} \\ \{\beta_{0k}, \beta_k; k = 1, 2, ..., K\}\ K(M+1) \text{ weights.} \end{aligned} \tag{2.7}$$

Adjusting the weight values is done in a manner that minimizes the loss function defined. In our case we use Mean Squared Error (MSE), as shown in Equation 2.8.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{f}(x_i))^2 \tag{2.8}$$

The exact method in which the weight values are adjusted is done through *backpropagation* in combination with a defined *optimizing algorithm.* Backpropagation is the process of feeding the information from the loss function backwards through the network. It does so by computing the gradient of the loss function with respect to the weights at each unit by utilizing the chain rule for differentiation (Goodfellow et al., 2016; Hastie et al., 2009). Through this process, the model identifies the contribution that each parameter in the model has on the loss function. Using the gradients found through

backpropagation, the optimizing algorithm employed in the model can subsequently adjust the value of each weight in the direction that reduces the loss function. This constitutes the learning procedure of the model. Repeating this process several times enables the model to incrementally "learn" the relationships between features in the data so that the loss function is minimized and the prediction accuracy increased.

This thesis will implement the *Minibatch Stochastic Gradient Descent* (SDG)[12], *RMSProp* (with momentum) and *Adadelta* optimizing algorithms, which are among the most common and successful optimizers employed today (Goodfellow et al., 2016). The SGD algorithm draws batches of random samples from the training set and uses gradient descent to incrementally improve the model by adjusting the weights by a *step* factor (learning rate) in the opposite direction of the gradient (Chollet and Allaire, 2017). RMSProp is an extension of SGD which divides the computed gradient with a running average of its recent magnitude and employs an adaptive learning rate in order to allow it to converge faster to an optimal solution (Tieleman and Hinton, 2012; Goodfellow et al., 2016). Adadelta is an extension of SGD which employs an adaptive learning rate and reduces the need for manual tuning (Zeiler, 2012; Keras, 2020a).

Finally, in addition to the common data processing described in Chapter 3, the DFNN requires further pre-processing in the form of scaling the inputs in order to function optimally (Hastie et al., 2009). Before tuning and fitting the DFNN model we therefore center and scale the inputs by subtracting the mean from the predictor values and then dividing them by their standard deviation.

2.3.4.1 Hyperparameters and Tuning

The DFNN is perhaps the most complex model implemented in this paper solely due to the sheer number of hyperparameters that must be chosen. In the following section we will briefly describe the hyperparameters set in this thesis, some of whose final values will be tuned for and others whose final values will be preset based on existing literature. A full overview of the hyperparameters can be seen in Table 2.2.

[12]With and without momentum.

Table 2.2: DFNN Hyperparameters

Hyperparameter	Description
Architecture	
Number of layers	Number of hidden layers in model
Number of hidden units	Number of hidden units in each hidden layer
Activation function	Activation function in hidden units
Bias	Whether to include bias units or not
Weight initializer	Process employed to set the initial values of the model's weights
Bias initializer	Process employed to set the initial values of the model's bias weights
Learning	
Batch size	Number of samples to draw from training data at each iteration
Epochs	Number of times the model is exposed to the whole training data set
Optimizer	Optimizing algorithm employed to adjust weights and improve loss
LR*	Step size controlling the size of weight updates at each iteration
LR annealing patience**	Number of epochs with no loss improvement before learning rate is reduced
LR annealing factor**	Factor by which learning rate is reduced once annealing patience has been reached
Momentum**	Velocity parameter that changes step size (learning)
Regularization	
Dropout	Rate of weigths to be dropped at each layer in each epoch
L2 regularization	Weight decay regularization
Early stopping patience	Number of epochs with no loss improvement before model training stops

*Learning Rate. **Only applicable for SGD optimizer. RMSProp and Adadelta use adaptive learning rates and built-in adaptive momentum.

The number of layers sets how many hidden layers the model should include. According to the universal approximation theorem, any Feedforward Network with a single layer is in theory sufficient to represent any function (Goodfellow et al., 2016). However, the single layer required to do so may be indefensibly large and may fail to learn or generalize properly. A deeper network usually requires fewer units per layer and can generalize quite well, but is harder to optimize. In this thesis we will implement a DFNN with two hidden layers, while the number of hidden units for each layer will be found through tuning. The activation function in these hidden units is the ReLU activation function discussed earlier. We also retain bias units in the model. It is common to initialize biases heuristically. We try two different values: 0 and 0.1. The first is compatible with most weight initialization schemes. The second helps us avoid causing too much saturation in the ReLU units, because it increases the likelihood that the initial input to the ReLU units will be positive, thus making it active (having non-zero derivatives) at initialization for most inputs (Goodfellow et al., 2016). We will tune for the final value of the two.

Since the model learns by updating weights according to their gradient loss, a set of initial weights must be set. Due to the structure of the model, setting initial weight values

to zero inhibits the algorithm. This is because the initial weights will then have zero derivatives and the algorithm will not be able to update the weights according to their gradient - thus failing at start (Hastie et al., 2009). Setting larger values often gives poor results. Preferably, the weights should be initialized randomly so that they avoid symmetry between units. We implement the widely adopted *Glorot Uniform* initialization of the weights, which draws values for the weights from a uniform distribution with limits $[-\sqrt{6/(m+n)}, \sqrt{6/(m+n)}]$, where m is the number of inputs to the layer and n is the number of outputs (Glorot and Bengio, 2010; Goodfellow et al., 2016).

The number of *epochs* signify the number of times the model iterates over the whole training set. Too many epochs causes the model to overfit; too few and it underfits. To select the optimal number we set the number of epochs to a high value, but avoid overfitting by implementing early stopping after 20 epochs of no improvement in the loss function. *Batch size* regulates the number of samples drawn during each training iteration within an epoch. Small batches lead to a better fit, larger batches to better generalization. This hyperparameter will be set through tuning.

The optimizing algorithm will be set through hyperparameter tuning as described earlier, but the three algorithms are treated differently with regards to learning rates and momentum. In the DFNN, the *learning rate* controls the increment at which the model learns at each iteration and should usually be set to a low initial value. SGD implements a fixed learning rate by default which must be manually reduced. For models using SGD we therefore tune for the initial learning rate and for the factor by which it should be reduced (annealed). We set a fixed learning rate annealing patience of five epochs. We also tune for momentum in the SGD algorithm manually. RMSProp and Adedelta both have adaptive learning rates that vary over time. We tune for the initial learning rate in RMSProp, but otherwise we leave the optimizer hyperparameters at default values in accordance with recommendations (Keras, 2020a,b).

As mentioned before, a major problem in fitting neural networks is the risk of overfitting during training. To avoid this, we implement two main regularization parameters: *dropout* (in each layer) and *L2 regularization*[13]. Dropout approaches regularization by simply setting a fraction of the input units in a layer to zero, effectively removing them from the

[13] In addition to early stopping.

network (Goodfellow et al., 2016). The fraction of units to be randomly set to zero depends on the dropout rate, which we will tune to find the optimal value. L2 regularization, or *weight decay*, is analogous to the regularization performed by Ridge regression[14], in which a penalty term λ is introduced to the error function (Hastie et al., 2009). Larger values for λ penalizes the weights by shrinking them towards zero.

2.3.5 Stacked Regression

The last model we consider is a *Stacked Regression* approach based on the SuperLearner algorithm by Van der Laan et al. (2007). Stacking was first introduced by Wolpert (1992), who coined the term *Stacked Generalization*. The concept was later later formalized by Breiman (1996). The fundamental idea of Stacking is to leverage the strengths of several base learners to achieve more accurate predictions (Hastie et al., 2009; Boehmke and Greenwell, 2019). Unlike ensemble methods that combine several *weak* learners, Stacked methods are designed to combine several *strong* learners to form optimal predictions (Boehmke and Greenwell, 2019). Stacked models have become increasingly popular in recent years, as they have shown strong performance in machine learning competitions (Kaggle, 2020a).

The SuperLearner algorithm consists of three main steps (Boehmke and Greenwell, 2019). First the ensemble is constructed by specifying a list of L base learners and selecting a metalearning algorithm[15] (Boehmke and Greenwell, 2019). To avoid selecting and tuning additional models, we use the models previously described in this chapter as our base learners. The metalearner can be any machine learning algorithm, but a regularized regression model is most commonly used (Boehmke and Greenwell, 2019). In this thesis, we use a regularized linear regression model as the metalearner. A regularized model is chosen to mitigate problems related to overfitting and highly correlated base learners.

The linear metalearner has two regularization parameters: α and λ. The regularization parameters are closely related to the penalty terms from the Lasso and Ridge regression models, often referred to as L1 and L2, respectively. Although an in-depth description of these models are outside the scope of this section, a discussion about the regularization parameters in the context of Stacking is warranted. In short, α dictates the distribution

[14]See Section 2.3.5 on Stacked Regression for in-depth explanation of Ridge regularization.
[15]The algorithm that is used to train the base learners. Hereafter referred to as metalearner.

between the penalty terms of the Lasso and Ridge regression models. Both models are very similar to OLS regression in the sense that the coefficients are estimated such that RSS is minimized. In addition to minimizing RSS, they also include a penalty term λ that performs regularization (James et al., 2013).

Ridge performs regularization by shrinking the coefficients of highly correlated predictors towards zero when λ increases (James et al., 2013), but never sets any of them to zero. Lasso works in a similar fashion, except that it has the power to set coefficients to zero for highly correlated predictors if the penalty term is sufficiently large (James et al., 2013). In the metalearner a high α means that regularization will be performed by means of Lasso, and a low α means that Ridge regularization will be performed. In the context of Stacking, this means that a value for α close to 0 will shrink the weights of highly correlated base learners towards zero, whereas a value closer to 1 means that highly correlated base learners can be excluded from the final ensemble if λ is sufficiently large. As such, λ controls the degree to which regularization should be performed.

In the second step the ensemble is trained. This involves using k-fold CV to train each of the base learners and collect their respective cross-validated predictions (Boehmke and Greenwell, 2019). The N cross-validated predictions from each model is then combined in a $N \times L$ feature matrix, as represented by matrix Z in Equation 2.9 (Boehmke and Greenwell, 2019). The feature matrix along with the original response vector y make up the *level-one* data

$$n\left\{\begin{bmatrix} p1 \end{bmatrix} \ldots \begin{bmatrix} p_L \end{bmatrix} \begin{bmatrix} y \end{bmatrix}\right. \rightarrow n\left\{\overbrace{\begin{bmatrix} \quad Z \quad \end{bmatrix}}^{\text{L}} \begin{bmatrix} y \end{bmatrix}\right. \tag{2.9}$$

where $p_1, ..., p_L$ is the cross-validated predictions obtained from each of the algorithms and n is the number of rows in the training set. The metalearning algorithm is then trained on the level-one data, as shown in Equation 2.9, which contains the individual predictions (Boehmke and Greenwell, 2019). The optimal weights for the base learners in the final model is obtained through cross-validation. In the third and final step the ensemble generates out-of-sample predictions. This is achieved by first generating predictions from the base learners, which are subsequently fed into the metalearner to generate the final

ensemble predictions (Boehmke and Greenwell, 2019).

3 Data

In this Chapter we describe the data cleaning process and present descriptive statistics of the final data used in the two phases of our analysis. In Phase 1 we replicate the data cleaning process implemented by SSB and construct a data set using the same variables as SSB. This is done to ensure a reliable comparison in the first part of the analysis. In Phase 2 we leverage additional property-specific and macroeconomic variables in an attempt to optimize the data for predictive modeling. This includes extensive feature engineering and advanced outlier detection.

The housing data used in this thesis is obtained from *Eiendomsverdi* and contains second-hand residential property transactions from the period January 2005 to September 2020 for five Oslo boroughs. The data covers approximately 121,000 unique transactions across 43 variables, including sales price, sales date and whether the transaction was a market sale. Moreover, the data contains variables that indicate the specific type of housing, such as estate type, estate sub-type and ownership type. The data also includes technical specifications and amenities. Technical specifications include variables such as square meters of primary area, gross total area, site area and build year. Examples of amenities include the number of bedrooms and whether the specific property has a balcony or an elevator. As for geographical variables, we have information about city district, coordinates, distance to coast, altitude and sunlight conditions.

3.1 Phase 1: SSB Replication

3.1.1 Data Replication and Descriptive Statistics

The data used by SSB is obtained from *Finn.no*, and contains second-hand residential property transactions for the period 2010-2019 for all regions in Norway[16] (Takle and Melby, 2020). Based on this data, SSB develops an AVM for each housing type in each region (such as Oslo). Each regional AVM estimates the market value[17] of properties in that region. This means that even though we only have data for Oslo, we can still ensure a reliable comparison. Moreover, since Finn.no get their data from Eiendom Norge, which

[16] Cities, municipalities and counties.
[17] Price per square meter of primary area.

in turn get their data from Eiendomsverdi, we have access to the exact same data as SSB. However, our data only covers transactions for 5 of the 15 boroughs in Oslo, whereas SSB uses data for all boroughs. This prevents us from perfectly replicating SSB's model for all of Oslo and represents a limitation of the analysis. Regardless, since the SSB benchmark model used in this thesis is trained on the same data as our machine learning models we can still compare their relative performances.

To ensure a fair comparison, we start by replicating the filtering process proposed by SSB (Takle and Melby, 2020). First, all missing values are removed from the data[18]. This is done to ensure that we only use data for which all features are known for all transactions. Furthermore, we restrict our sample to transactions in the period 2010-2019. Thereafter, we remove all transactions where the sales price is below 350,000 NOK. We also remove transactions where the price per square meter is below 10,000 NOK, which is the lower limit for dwellings located in cities (Takle and Melby, 2020). Moreover, SSB has established certain criteria for each housing type in terms of area and price per square meter. Table 3.1 shows a summary of these criteria.

Table 3.1: SSB's housing criteria

Retrived from Takle and Melby (2020).

	Area (m^2)	Price per square meter (NOK)
Detached houses	50-550	5,000-150,000
Semi-detached houses	40-350	5,000-150,000
Apartments	12-350	8,000*-200,000

*The lower limit is 10,000 NOK for apartments located in cities.

Since 97.12% of the properties in our data are apartments, and our coverage of the remaining types of housing from the SSB data is minimal[19], we restrict our sample to only consider apartments. It is important to note that this does not prevent a fair comparison between the models, as SSB estimates individual models for each housing type and region (Takle and Melby, 2020). Hence, we can still use the model developed for apartments as a benchmark. Finally, we filter out observations that do not satisfy the apartment criteria in Table 3.1. The final sample consists of 79,899 unique transactions, whereas SSB uses

[18] 495 missing data points are removed.
[19] The three remaining estate types constitutes 3,396 observations.

162,225 observations. The difference in sample size can be attributed to the difference in data coverage for the different boroughs in Oslo. We excluded a total of 135 transactions from the sample of apartments as a result of the filters, which corresponds to 0.2% of the data.

SSB uses three main variables to estimate the price per square meter of an apartment: the natural logarithm of primary area, price zone[20] and age of the property at the time of sale, which is divided into four age intervals (Takle and Melby, 2020). Additionally, the two dummy variables urbanization and year are included to control for population effects[21] and inflation (Takle and Melby, 2020), respectively. The population dummy is divided into six intervals depending on the population of a region. Since we only use data for Oslo, which is an urban area with a total population above the highest interval of the dummy variable[22], we do not need to include this variable in our models, as all transactions would assume a value of 1.

Table 3.2 shows descriptive statistics for the variables used in the first phase of our analysis. Note that we report all variables on their original scale in the interest of interpretability. Log-transformations for the variables specified by SSB[23] will be carried out before fitting the models in Chapter 5. The table shows that the average apartment has a primary area of 63 square meters and is valued at approximately 62,000 NOK per square meter. The minimum and maximum values reveal that there are large variations inherent in both price and primary area of apartments. The price per square meter, for instance, varies between 10,000 NOK and 194,940 NOK, suggesting that it is highly skewed. Primary area exhibits similar properties. Furthermore, the table illustrates that the vast majority of apartments (77.9%) are older than 34 years at the time of sale. As for location, Grünerløkka appears to be the area with the most sales (27.6%), closely followed by Sagene and Frogner.

[20] Borough or city district

[21] Whether a region is considered an urban area.

[22] 100,000 is the upper limit of this variable.

[23] Price per square meter and primary area.

Table 3.2: Descriptive statistics for Phase 1

This table shows descriptive statistics for all variables used in Phase 1 of the analysis. Mean, Min and Max are the average, minimum and maximum value for each variable, respectively. Price per square meter is reported on its original scale, but will be log-transformed before fitting the models, as described in Chapter 5. The data contains transactions for the period 2010-2019 and covers properties from Grünerløkka, Sagene, St. Hanshaugen, Ullern and Frogner. The sample is restricted to apartments only. All missing values have been removed. N = 79,899. The following table is constructed by combining the final training and test data after pre-processing. The training and test data remains separated for all other purposes.

Variable	Mean	Min	Max
Price per square meter (NOK)	62,075.710	10,000.000	194,940.500
Primary area (m^2)	63.229	12	297
Year	2015	2010	2019
Age $<$ 10	0.105	0	1
Age 10-19	0.060	0	1
Age 20-34	0.056	0	1
Age $>$ 34	0.779	0	1
Grünerløkka	0.276	0	1
Sagene	0.254	0	1
St. Hanshaugen	0.168	0	1
Ullern	0.067	0	1
Frogner	0.235	0	1

3.2 Phase 2: Additional Variables

3.2.1 Data Cleaning and Feature Engineering

In Phase 2 we utilize all available variables from Eiendomsverdi. Additionally, we include mortgage rates (Bloomberg, 2020), oil price (Eikon, 2020) and inflation rate (Bloomberg, 2020) as complementary macroeconomic variables. Since we have the date for each sales transaction, the external variables can be matched to either the year or specific date of a transaction. We use the 3-month Norwegian Interbank Offered Rate (NIBOR) as a proxy for mortgage rates, as NIBOR is typically used as a reference for mortgage rates in Norway. The rationale for including NIBOR is that it affects demand for housing. Lower NIBOR rates translate to lower mortgage rates which in turn should increase demand for housing as loans would become cheaper. Conversely, a high NIBOR rate results in higher mortgage rates, which should decrease demand for housing.

For the oil price, we obtain daily spot prices in USD for Norwegian Brent Spot. The idea behind including oil prices is that the petroleum industry is Norway's largest industry as a percentage of GDP. Because of this, oil prices are often used as a proxy for the condition of the Norwegian economy. While the effect of the oil price may not be consistent throughout the data period, we still consider it to be a relevant macroeconomic variable. Lastly, inflation rates from the period 2005-2020 are included to control for the general increase in prices during the period. We have obtained the Norwegian Consumer Price Index (CPI) for the period 2005-2020 as our measure for inflation.

Missing values, encoding and transformation of variables must be addressed in order to optimize the data for predictive modeling. In machine learning literature, this process is usually referred to as *feature engineering*, which involves extracting features[24] from raw data and transforming them into formats that are suitable for machine learning models (Zheng and Casari, 2018).

Initial analysis of the data reveals that there are over 1 million missing data points[25]. As most machine learning algorithms are unable to deal with missing values, this must be addressed before predictions can be made (Kaiser, 2014). A simple and accepted solution when dealing with missing data is to remove all missing values (Kantardzic, 2003). However, doing so would reduce our original data from 121,000 observations (rows) to a mere 1,800 observations. Simply omitting all missing values would consequently be unfortunate, as this would drastically reduce the amount of data we can use to train the models. It would also reduce the quality of the data, as we might end up removing data that actually contains important patterns and relationships, which may lead to increased bias (Kaiser, 2014).

To avoid wasting important data and significantly reducing our data set, we instead try to use the existing set of features to *impute* missing values. Imputation of missing values is an accepted and popular technique to deal with missing values (Kantardzic, 2003; Kaiser, 2014; Troyanskaya et al., 2001; Zainuri et al., 2015). As a general rule, we omit features in which more than 40 percent of the data is missing, while making exceptions for features that are deemed important (or unimportant). Highly suspect and unreliable transactions

[24]Numeric representation of an aspect of raw data, i.e variable (Zheng and Casari, 2018).
[25]A missing data point is defined as any row where the value of one feature (variable) is missing.

are also removed. This includes transactions below 350,000[26] NOK and transactions where all square meter metrics are zero or missing.

We utilize two main imputation techniques to handle missing values, namely imputation by default value and imputation by grouped median. Variable descriptions and imputation methods for all variables can be found in Appendix A1.1. Detailed explanations for the reasoning behind different imputation methods can also be seen in Appendix A1. For features that only take one of two values, such as whether a particular housing type has balcony (yes or no), we simply assume that missing values indicate that the particular housing type does not have a balcony. Hence, we set a default value of zero[27] for such variables.

For numerical features, we adopt a grouped median approach, as proposed by Maniruzzaman et al. (2018). The rationale for using grouped median as opposed to the median value is that the median value of a subgroup is often a more accurate proxy than the median of all values for a given feature. Median is preferred over mean as the median is more robust to outliers (Kantardzic, 2003). When using grouped median (or similar aggregated imputation techniques) to impute missing values, it is extremely important to split the data into a training and a test set *before* imputing the values in order to avoid data leakage. Imputation is therefore done separately for the training and test set.

Imputation by grouped median is applied to all numeric features with the exceptions of the different metrics for area[28]. Since at least one square meter metric is available for all transactions, we use the different metrics as a proxy for each other, as it is reasonable to assume that this is a better proxy than the median value of all apartments. In cases where the value of a feature likely depends on the geographical location of a property, such as for altitude, distance to coast and coast direction, we use city district as a grouping variable, with the rationale that these these features are most likely very similar for properties in the same area. For features that likely depend on the specific type of housing, such as the number of bedrooms, we use housing type as grouping variable. In cases where grouped median cannot be computed because all values for a given group are missing, we use the

[26] The lower limit of property values established by SSB (Takle and Melby, 2020).
[27] Where 1 represents yes and 0 represents no.
[28] Primary area, gross area and gross total area.

median value. Lastly, missing values for categorical variables are imputed using a default value that we set for each relevant variable.

After missing values have been imputed, we investigate all features and ensure that they have been encoded in a way that makes it easy for machine learning algorithms to understand the inputs. Categorical variables are encoded using *one-hot encoding* (dummy variables), as recommended by Kuhn and Johnson (2019) and Zheng and Casari (2018). Some of the categories in our categorical variables appear very infrequently. To handle this we combine the least frequent categories into a joint category.

Following the encoding process, we construct new features from our existing set of variables. We use the original BuildYear variable to create a new variable for the age of a property at the time of the sale. Using SiteArea and SiteAreaUndeveloped, we also create a new variable to indicate undeveloped site area as a percentage of total site area. This will serve as a proxy for the potential of a property in terms of investments and expansion, and in general as a measure of how spacious the property is. To capture seasonal differences in sales prices, we use the MarketSaleDate variable to make two new date variables that indicate the month and year of a sale (SalesMonth and SalesYear). Lastly, we use Floor and FloorNumber to create dummy variables that indicate whether an apartment[29] lies on the ground floor, top floor, or somewhere in between. This is done to enhance the signaling effect of the floor variables, as the most important aspect is usually not the actual floor number, but the positioning of the apartment relative to the total number of floors. Apartments with missing values for these variables are imputed as lying on a middle floor.

3.2.2 Outliers

Outliers are extreme observations that fall outside the normal range of the data to the degree that they raise suspicion about the veracity of the observations. While such observations should not automatically be removed, they do warrant further investigation. We have been made to understand that, while uncommon, our data set may contain errors in both the response variable and the predictor variables. The sources of such errors may include typing errors, recording errors, incomplete data migration, or data corruption. As

[29] Floor variables are only reported for apartments.

such, we deem that outliers in our data set should be addressed and potentially removed in the interest of maximizing prediction accuracy. Identifying the erroneous outliers and handling them is a challenge that is compounded by the number of observations, the large number of variables, the lack of an identifiable error pattern and our own lack of industry experience. This makes it hard to heuristically determine a set of criteria for addressing outliers in each variable.

In this thesis, we therefore employ *Isolation Forest* (IF), a model-based anomaly detection algorithm developed by Liu et al. (2008). This approach is preferred over a heuristic approach in our situation because it is relatively inexpensive to implement and has been shown to work well with data that contains a large number of possibly irrelevant features (Liu et al., 2008). By addressing outliers in a statistical manner, the algorithm allows us to perform data cleaning in an unbiased manner which has been proven to be robust (Liu et al., 2008) and has been successfully adopted in machine learning contests (Kaggle, 2018). While the algorithm does not guarantee that only erroneous data is identified, we still deem it to be less flawed than a heuristic approach given the large number of variables and our lack of industry expertise. A detailed description of the algorithm can be found in Appendix A2.

For each row in the training data the algorithm computes an anomaly score, as shown in Equation .1.

$$s(x,n) = 2^{-\frac{E(h(x))}{c(n)}} \tag{3.1}$$

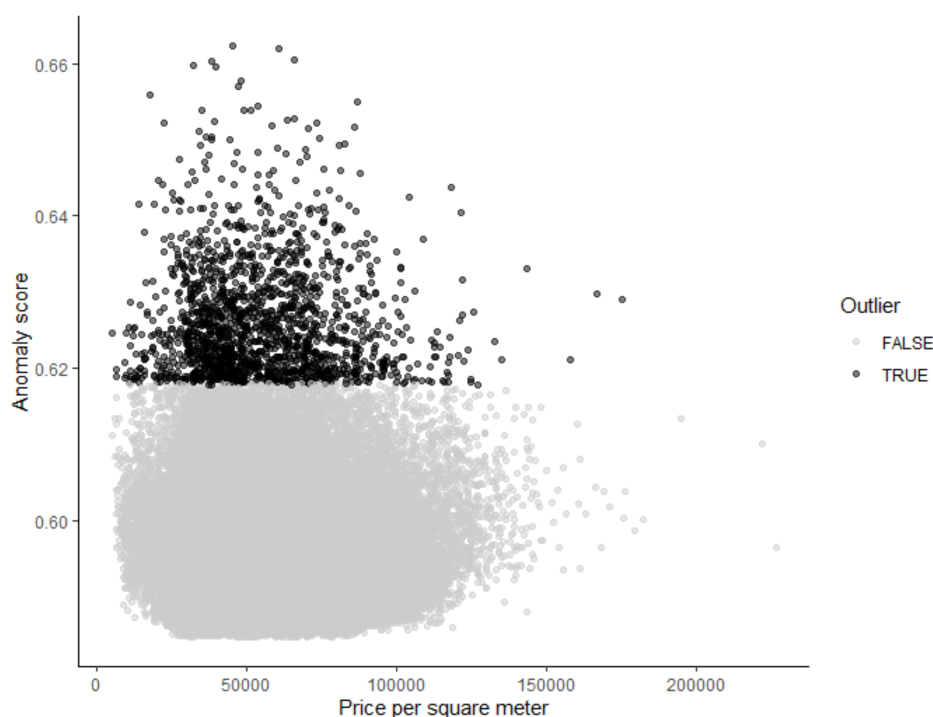
In this setting, $h(x)$ is the path length of observation x, $E(h(x))$ is the expected or average path length of observation x, $c(n)$ is the average path length of unsuccessful search in a binary search tree and n is the number of external nodes in a tree (Liu et al., 2008). A score closer to 1 indicates an anomaly, whereas a score closer to 0.5 indicates normal observations. We train IF using a random sample of 512 observations from the training data, and apply it on the remaining training set. The algorithm is not employed on the test set to avoid introducing bias and overestimating our results. To establish a threshold for removing outliers, we remove all observations with an anomaly score that lie in the 99th percentile (of the anomaly score) or higher. This is done to ensure that we only

remove the most extreme observations that are most likely to represent true outliers. In total, 1648 observations are classified as anomalies and consequently removed from the training data. This corresponds to 1.4% of the data set.

Figure 3.1 illustrates the classification of observations of the response variable. The figure shows that the three observations with highest price per square meter are not classified as anomalies. A possible explanation for this is that these observations had predictor values that were in line with extraordinarily high sales prices. As such, they do not represent outliers when considering the rest of the feature space.

Figure 3.1: Anomaly plot of the response variable using Isolation Forest

This figure shows anomaly scores of the response variable for each observation in the training set. Points with light shading (lower half) are classified as normal observations (True), while darker shaded points are classified as anomalies (False) by the IF algorithm. Black shading indicates clustering of observations among the outliers.



3.2.3 Descriptive Statistics

As in Phase 1, the vast majority (98.2%) of properties in the Phase 2 data set are apartments. In the interest of prediction accuracy, we therefore limit the analysis to only consider apartments in Phase 2 as well, as we do not have a sufficient amount of data to produce reliable and accurate predictions for other housing types. Upon completing

feature engineering and data cleaning, we end up with 116,126 observations and 60[30] predictor columns in total. Table 3.3 shows descriptive statistics for a subset of the continuous variables from the final sample, and Table 3.4 shows descriptive statistics for a subset of the dummy variables. Complete descriptive statistics for all variables can be found in Appendix A3.1 and A3.2.

Table 3.3 shows that the average apartment in Oslo is a two-bedroom apartment covering approximately 63 square meters of primary area, that lies 51 meters above sea level and roughly 2.1 km from the coastline. Moreover, the building is on average 72 years old, lies in an area with approximately 4,100 units in adjoining squares and has a total site area of 3,700 square meters. The value of such property is on average 56,500 NOK per square meter with 121,200 NOK in shared debt. From the table we see that there are large variations in some of the variables, as evidenced by the large gap between minimum and maximum values. The average sales price, for instance, is approximately 56,500 NOK per square meter, but it varies from 1,200 to 226,900 NOK per square meter. This means that the distribution of the response variable is highly skewed. Large variations are also found in the different metrics for square meters (primary and gross area) as well as total and undeveloped site area.

[30]Includes dummy variables.

Table 3.3: Descriptive statistics for a subset of the continuous variables used in Phase 2

This table shows descriptive statistics for a subset of the continuous variables. Mean, Min and Max are the average, minimum and maximum value for each variable, respectively. Price per square meter is reported on its original scale, but will be log-transformed before fitting the models as described in Chapter 5. All observations with an anomaly score that fall in the 99th percentile (of the anomaly score) or above have been removed from the training set, but not the test set. Missing values for all variables have been imputed using grouped median, median or default value. Unreliable observations have been removed. The data contains transactions for apartments in the period 2005-2020 and covers properties from Frogner, Grünerløkka, St. Hanshaugen, Sagene and Ullern. N = 116,126. Note that the data was split into a training and test set before missing values and outliers were handled. The following table is constructed by combining the final training and test data after pre-processing, and is only used to illustrate the entire data set in this chapter. The training and test data remain separated for all other purposes.

Variable	Mean	Min	Max
Price per square meter (NOK)	56,535.94	1,209.68	226,851.90
Shared debt in joint ownership (NOK)	121,240.60	0	5,201,000
Primary area (m^2)	62.80	10	386
Gross area (m^2)	68.08	10	386
Age of building (years)	71.87	0	256
Number of bedrooms	1.69	0	23
Site area (m^2)	3,671.85	0.00	61,435.60
Undeveloped site area (m^2)	2,741.13	0.20	49,273.60
Altitude (meters)	51.28	0	190
Distance to coast (meters)	2,120.45	5	10,000
Slope of site (degrees)	3.65	0	37
Number of units in adjoining squares	4,146.67	258	6,653
Consumer price index	96.94	81.20	112.90
NIBOR 3-month (%)	2.17	0.23	7.91
Brent Spot (USD)	75.57	19.33	146.08

Table 3.4 shows the distribution of a subset of the dummy variables. We see that the majority of apartments are block apartments (51.2%), and that a large fraction of the apartments do not have a specific sub-type (47.8%). Moreover, the majority of apartments have a balcony (66.8%), and 38.9% of them have an elevator installed. Furthermore, we see that the vast majority of these apartments are middle-floor apartments (76.8%), whereas top- and ground-floor apartments only account for 5.9% and 17.3%, respectively. The data indicate that the areas with highest turnover are Grünerløkka (27.7%), Sagene (25.7%) and Frogner (23.9%). When it comes to ownership, the two most common forms

of ownership are condominiums[31] (52.2%) and joint housing cooperatives (37%). Finally, the vast majority of lot sites are owned by the home owners (88.4%), and the remaining 11.6% of sites are leased.

Table 3.4: Descriptive statistics for a subset of the dummy variables used in Phase 2

This table shows descriptive statistics for a subset of the dummy variables. Mean is the average value for each dummy variable. All observations with an anomaly score that fall in the 99th percentile (of the anomaly score) or higher have been removed from the training set, but not the test set. Missing values for all variables have been imputed using a default value. The data contains transactions for the period 2005-2020 for apartments only and covers properties from Frogner, Grünerløkka, St. Hanshaugen, Sagene and Ullern. N = 116,126. The data was split into a training and test set before missing values and outliers were handled. The following table is constructed by combining the final training and test data after pre-processing, and is only used to illustrate the entire data set in this chapter. The training and test data remain separated for all other purposes.

Variable	Mean
Balcony	0.668
Elevator	0.389
Ground floor	0.173
Middle Floor	0.768
Top floor	0.059
Frogner	0.239
Grünerløkka	0.277
Sagene	0.257
St. Hanshaugen	0.171
Ullern	0.056
Apartment	0.478
Apartment in block	0.512
Other	0.010
Stock apartment	0.108
Housing cooperative	0.370
Cooperative apartment	0.00002
Condominium	0.522
Leasehold site	0.116
Site owner	0.884

[31] Independent apartment that is owned solely by the owner.

4 Methodology

In this chapter we outline the approach for our analysis and describe the methodology by which we implement and evaluate our machine learning models. The first section describes the analyses we will perform and the order of actions taken. The second section deals with our model selection process, and will outline how we implement hyperparameter tuning and resampling methods for each model. Finally, the last section describes the metrics used for model assessment.

4.1 Approach

The aim of our analyses is to investigate whether our non-linear machine learning models can achieve increased prediction accuracy for housing prices in the Norwegian housing market compared to linear models. To investigate this, we apply a two-pronged approach where we split the analyses into two separate phases. In Phase 1 of our analyses, we replicate the housing data methodology developed by SSB and compare SSB's model against our non-linear machine learning models trained on the same data set utilized by SSB. In Phase 2, we implement our machine learning models and the benchmark linear model (LM) on the expanded data set. The expanded data set includes additional property-specific variables obtained from Eiendomsverdi in addition to external macroeconomic variables[32]. In this manner, Phase 1 isolates the effects of the different algorithms and demonstrates the effects of changing the learning algorithm employed. Phase 2 investigates how much accuracy can be increased by including additional variables and implementing different pre-processing, and how the different algorithms capture this additional information.

4.1.1 Phase 1: SSB Replication

In the first phase of our analysis, we predict the log-adjusted price per square meter in accordance with the methodology developed by SSB (Takle and Melby, 2020). Primary area, age, location[33], and year are used as predictors.

The data cleaning process proposed by SSB is replicated[34], ensuring that all models are

[32]See Chapter 3 for a detailed overview of the data sets in Phase 1 and Phase 2.
[33]Referred to as Price Zone in the SSB paper.
[34]Described in Section 3.1.1.

fitted on the same data that SSB uses. Each model is tuned according to the tuning regimen specified in Section 4.2.2.2, with the final fitted models representing the best tuning of each algorithm. The final models are evaluated based on their prediction accuracy in terms of RMSE and computation speed as specified below in Section 4.3.

4.1.2 Phase 2: Additional Variables

In Phase 2 we utilize all available property-specific variables obtained from Eiendomsverdi as well as the additional macroeconomic variables[35] described in Section 3.2.1. The expanded data set also makes use of observations from earlier dates, and undergoes a more extensive pre-processing and feature engineering regimen than in Phase 1[36]. We log-adjust the price per square meter (response variable) in order to allow for comparison between the two phases and to reduce the variable's skew.

Because we operate with a large number of features, we apply the Boruta algorithm (Kursa et al., 2010) for feature selection to remove any unimportant variables and verify the importance of the remaining ones. A detailed description of how the algorithm works can be found in Appendix A4. This is motivated by two considerations. First, it allows us to carry out feature selection in a systematic and unbiased manner. Second, many machine learning algorithms exhibit decreasing performance when the number of features are significantly higher than what is optimal (Kohavi et al., 1997). Applying the Boruta algorithm will help us to remove features that may lead to poor performance. Finally, each model is tuned and fitted according to the tuning regimens we specify below. The final models are evaluated according to the same metrics as in Phase 1.

4.2 Model Selection

All the models implemented must be tuned and trained for each phase of our analysis. The following sections describe our specific process in this regard.

[35] Oil price, NIBOR and CPI.
[36] Described in Chapter 3.

4.2.1 Software and Computing Platforms

All models and analyses are implemented using the R programming language. We make use of different packages for each model, the selection of which has been made in order to comply with best practices and facilitate ease of implementation. Each model makes use of the same packages in both phases.

The linear regression benchmark is implemented with the built-in *stats* library in R. For Random Forest and XGBoost, we use the *caret* (Kuhn, 2020) package in combination with the algorithm-specific *randomForest* (Breiman et al., 2018) or *XGBoost* (Chen et al., 2020) packages. The DFNN is implemented through the *keras* (Allaire and Chollet, 2020) package with *tensorflow* (Allaire and Tang, 2020) as its backend. A difference between the two phases is that both *keras*, *tensorflow* and *XGBoost* are implemented with GPU support in Phase 2 in order to facilitate accelerated computing for XGBoost and DFNN. *H2o* (LeDell et al., 2020) is used to implement the Stacked Regression model.

All Phase 1 models are run on a Windows machine with eight logical processors and 16 GB RAM. For Phase 2 of our analyses, we utilize Amazon Web Services (AWS) to set up cloud computing virtual machines with enhanced memory, more logical processors and with Tensorflow-compatible GPUs. These AWS machines all run on Ubuntu OS. Regrettably, AWS policies prohibit us from immediately ramping up to our preferred processing and memory specifications in Phase 2. As a consequence, the models in Phase 2 are run on different machine specifications, with the first models being run on lower specification machines and the last models being run on machines with higher specifications. The Phase 2 machine specifications for each model are described in Chapter 5.

4.2.2 Model Training

4.2.2.1 Data Partitioning and Resampling Methods

All models will be trained using a random sample of 70% of the data. The remaining 30% constitute the test set which will only be used to generate out-of-sample predictions and compute the test error of the models. As discussed in Section 2.2.5, hyperparameter tuning will be performed by means of k-fold CV. The exception is for the DFNN, where we use a validation set approach due to *keras* package limitations. As mentioned previously,

we set $k = 5$ when performing k-fold CV because we find it to provide a good balance between computing time, bias and variance.

4.2.2.2 Hyperparameter Tuning

As discussed in Chapter 2, the machine learning models we employ have several hyperparameters that must be tuned in order to optimize performance. To do so, we first define each model's hyperparameter feature space[37], and thereafter try different combinations of hyperparameter values according to our tuning method.

In this thesis we employ *grid search* and *random search* as tuning methods. Grid search systematically fits a model for each combination of hyperparameters across the whole hyperparameter feature space grid in order to find the model that yields the lowest validation error. This is the most thorough method since the whole hyperparameter feature space is trialed. This is useful in cases where we implement an efficient algorithm and there are few possible combinations of hyperparameters. In cases where there are a large number of hyperparameters to tune and the hyperparameter value ranges are large, it becomes prohibitively expensive to implement grid search for most algorithms. In such cases, it is more efficient to implement random search. This involves fitting randomly sampled hyperparameter combinations from the hyperparameter feature space. While this method is not as exhaustive as grid search, it has been shown to be more efficient in finding good models than grid search (Bergstra and Bengio, 2012) because it is able to effectively search a larger hyperparameter configuration space. Although random search samples a relatively low number of random hyperparameter combinations, it still produces a high probability of finding a model that is close to the optimal hyperparameter combination. This can be shown by

$$1 - (1 - \frac{v}{V})^T = p \tag{4.1}$$

which shows the probability of finding a target model through random search (Bergstra and Bengio, 2012). Here $\frac{v}{V}$ is the volume of the target relative to the hyperparameter

[37]The value range each hyperparameter can have.

hypercube, T is the number of sampled hyperparameter combinations, and p is the probability of finding a model in the target volume (Bergstra and Bengio, 2012; Scriven, 2018). If the target is to find a model within 5% of the optimal model ($\frac{v}{V} = 0.05$) with a 95% probability ($p = 0.95$), then solving for T shows that we must sample 59 models.

Table 4.1 shows the hyperparameter tuning methods and the number of combinations tried for each model in both phases of our analysis.

Table 4.1: Hyperparameter Tuning Methods and Combinations Sampled

Phase 1					
Model	XGBoost	Stacked	RF	DFNN	LM
Tuning method	Random	Automatic	Grid	Random	None
# Combinations tried	120	100	8	120	1
# Combinations possible	504,000	100	9	100,018,800	1
Phase 2					
Model	XGBoost	Stacked	RF	DFNN	LM
Tuning method	Random	Automatic	None	Random	None
# Combinations tried	120	100	1	120	1
# Combinations possible	448,000	100	60	100,018,800	1

The tuning method applied to each model largely reflects the number of hyperparameters and hyperparameter values each model can have, while also taking into consideration other implementation issues such as R package compatibility and computing requirements. The benchmark LM has no hyperparameters to tune and is therefore not a concern. XGBoost and DFNN, as we have seen, have numerous tunable hyperparameters with broad value ranges. For the DFNN, some hyperparameters are heuristically set as fixed values, as described in Chapter 2. Apart from those, we set value ranges for the XGBoost and DFNN hyperparameters by reviewing existing literature and best practices from competition platforms such as Kaggle. The final tuning ranges of these models' hyperparameters can be seen in Appendix Sections A8 and A9. The DFNN tuning range remains the same in both phases and amounts to over 100 million unique combinations of hyperparameter values. For XGBoost, the tuning range in Phase 1 amounts to 504,000 unique combinations. In Phase 2 we have to reduce the tuning range of the *max depth* parameter because we do not have enough memory to process a model with the highest *max depth value* (30). The number of hyperparameter combinations has consequently been reduced to 448,000 in

Phase 2. Nonetheless, the number of possible combinations is too high to tune through grid search, and we therefore opt for random search with 120 sampled models fitted for both XGBoost and DFNN. According to Equation 4.1, this corresponds to a 95.5% probability that our best performing model for each algorithm lies within 2.5% of the optimal model.

Our implementation of Random Forest calls only for tuning the *mtry* hyperparameter, while the *ntree* is set to a sufficiently high number such that the error stabilizes. *mtry*'s hyperparameter value range is limited by the number of variables in the data set, but should not equal the number of variables. For Phase 1, where the data set only contains nine[38] predictor columns, we conduct a grid search of eight models. Our preference would be to implement the same method for Phase 2, but the expanded data set used in our second phase increases the computational requirements of the Random Forest algorithm to the point where we are unable to fit more than one model. Hyperparameter tuning of the Random Forest model in Phase 2 has therefore been abandoned in favour of using the default value of *mtry*[39]. The Phase 1 and Phase 2 tuning grids for Random Forest can be seen in Appendix A7.1 and A7.2.

The Stacked Regression model differs from the others since we have to consider both the base learners as well as the metalearner wrapper. Tuning each base learner is extremely time- and resource consuming. To avoid selecting and tuning new models we instead use the models previously described[40] as our base learners, and use the hyperparameters obtained from tuning these models. The metalearner we have chosen has two hyperparameters to tune: α and λ. The package we use to implement the Stacked Regression model inhibits an efficient tuning of α, but has an automatic tuning functionality for λ. It tunes λ by trying a high λ value first and decreasing it until it starts overfitting (LeDell et al., 2020). 100 different λ values are tried in our implementation.

4.3 Model Assessment

This thesis is primarily concerned with the prediction accuracy of our models in terms of RMSE. We therefore evaluate the performance of the models based on their test error. We

[38] Including $K-1$ dummy variables for each categorical variable

[39] $p/3$

[40] RF, DFNN, XGBoost and LM.

also record the computing time of each model in order to gain an indication about each model's computational complexity and to shed light on the trade-off between accuracy and computing time, although we consider this a secondary concern.

We use RMSE as a universal metric for assessing the accuracy of our prediction models. This metric is preferred over MSE because it is reported on the same scale as the response variable, making it easier to interpret (Hyndman and Koehler, 2006). RMSE is computed for both the training and test data. The training RMSE is used as a baseline for evaluation, whereas the test RMSE is used to evaluate the out-of-sample performance of the models. Equation 4.2 shows the formula used to compute RMSE.

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{f}(x_i))^2} \tag{4.2}$$

In this setting, n is the number of observations, y_i is the observed (i.e. true) value and $\hat{f}(x_i)$ is the predicted value for the ith observation. A small value for RMSE indicates that the predicted values are close to the true values, whereas a large value for RMSE indicates the opposite. A convenient property of RMSE is that it uses squared residuals to compute the error, implying that large residuals will be penalized (James et al., 2013).

5 Analysis

In this chapter we carry out the procedures described in Chapter 4 and apply the models described in Chapter 2. The following analysis is divided into two phases. In Phase 1 we compare the predictive performance of the non-linear machine learning algorithms against the linear model developed by SSB using the same data and variables used by SSB. In Phase 2 we seek to investigate the models' performance on an expanded data set with additional variables. Feature selection is performed by means of the Boruta algorithm. In each phase we first evaluate the performance of the models in regards to accuracy and computing time. This is followed by a discussion on model interpretability.

5.1 Phase 1: SSB Replication

Phase 1 of our analysis compares the performance of our non-linear machine learning algorithms against the linear benchmark model developed by SSB, using the exact same data and feature selection procedure. The training data used in this phase contains 55,929 observations and consists of the variables primary area, location, year and age. The data has been cleaned in accordance with SSB's filtering process[41].

5.1.1 Model Assessment

The resulting training- and test errors of the final models can be seen in Table 5.1. An overview and discussion of the final tuned hyperparameter configurations for each model in Phase 1 can be seen in Appendix A11.

Table 5.1: RMSE of final models

Metric	XGBoost	Stacked	RF	DFNN	SSB
Training RMSE	0.1812	0.1811	0.1821	0.1892	0.2045
Test RMSE	0.1851	0.1856	0.1872	0.1875	0.2023

The results reveal that XGBoost has the best out-of-sample performance among all the models, closely followed by the Stacked model. With a test RMSE of 0.1851, XGBoost

[41] As described in Section 3.1.1.

delivers an accuracy gain of 8.5%[42] compared to the benchmark model in terms of RMSE. By back-transforming[43] the predictions to get values in NOK, we find that this corresponds to a test RMSE of 10,033 NOK per square meter for XGBoost, compared to a test RMSE of 11,415 NOK for the benchmark SSB model.

Furthermore, none of the models demonstrate signs of overfitting or underfitting, as seen from the relative differences between training error and test error for each model. This indicates that all models generalize well and demonstrate a good balance between bias and variance. The results also show that all four machine learning algorithms outperform SSB's linear model in terms of RMSE. This indicates that nonlinear learning algorithms are better suited to model the relationship between price per square meter and the predictors, at least with regards to prediction accuracy. This may suggest that the data exhibits non-linear relationships that a linear regression model cannot effectively capture.

5.1.1.1 Accuracy Distribution

A breakdown of the out-of-sample predictions of the best performing model and the benchmark SSB model is shown in Table 5.2. The table illustrates how much the predicted square meter prices correspond to the actual prices in the test data for the two models. The table reveals that 48.67% of the predictions made by XGBoost have a predicted value that corresponds to 81-100% of the actual market price, and that 35.39% of the predictions lie within 101-120% of the actual price. This means that 84.06% of the XGBoost predictions lie within ± 20% of the actual price. This represents an improvement over the benchmark model, where 79.5% of the predictions lie within the same range. The table also shows that the tails are small for both models. This is especially true for the left tail. For XGBoost, only 6.53% of the predictions have a predicted value that corresponds to less than 80% of the actual market value, whereas 9.17% of the predictions fall into the same group for SSB's model. This demonstrates that there are few grossly underestimated observations. The accuracy on the opposite side of the distribution is slightly worse for both models, but XGBoost performs better for these groups as well. The results indicate that XGBoost is overall more accurate than the benchmark linear model and that XGBoost is less prone to extreme underestimation and overestimation. A visual illustration of the overall accuracy

[42]Percentage reduction in test RMSE.
[43]e raised to the power of x, where x is the natural logarithm of the predicted price for instance x.

of both models can be found in Appendix A10.1.

Table 5.2: Accuracy distribution for XGBoost and SSB

This table illustrates the accuracy distribution of the best performing model (XGBoost) and the benchmark SSB model. Distribution is how much the predicted values correspond to the actual values in the test set, divided into 10 groups. n is the number of predictions that falls into a given group. % is n as a percentage of the total number of observations in the test set. Cumulative % is the cumulative percentages per group. For example, for XGBoost, the first row shows that only 3 observations have a predicted value that is less than 40% of the actual value and that the predictions in this group accounts for 0.01% of the test data. The table is inspired by Takle and Melby (2020).

	XGBoost			SSB		
Distribution (%)	n	%	Cumulative %	n	%	Cumulative %
0-40	3	0.01	0.01	1	0.004	0.004
41-60	74	0.31	0.32	89	0.37	0.38
61-80	1488	6.21	6.53	2110	8.80	9.17
81-100	11666	48.67	55.20	10863	45.32	54.49
101-120	8484	35.39	90.59	8193	34.18	88.67
121-140	1453	6.06	96.65	1788	7.46	96.13
141-160	296	1.23	97.88	349	1.46	97.59
161-180	162	0.68	98.56	173	0.72	98.31
181-200	82	0.34	98.90	115	0.48	98.79
201+	262	1.09	100.00	289	1.21	100.00

5.1.1.2 Computing Time

While our main concern in this thesis is prediction accuracy, it is also important to address computing time. Different models have different computational requirements, which is reflected in the time taken to train them[44]. The total time consumed tuning a model is a function of the inherent computational efficiency of the algorithm as well as the number of tuning iterations. An overview of the computing time of the Phase 1 models can be seen in Table 5.3.

[44]Assuming models are run on similar computational platforms.

Table 5.3: Model Tuning and Computing Time Comparison

This table provides an overview of the methods and time spent tuning each model. All models were run on a similar computational platform consisting of eight logical processors and 16 GB RAM. Computing time is measured as time from the start of model tuning until the final optimal model has been fitted. Mins per model is the average time in minutes consumed per model tuned for each algorithm. *Computing time for Stacked Regression includes model fitting time and tuning time for the base learners.

	XGBoost	Stacked	RF	DFNN	SSB
Tuning method	Random	Automatic	Grid	Random	NA
Computing time	1.48 h	30.02* h	13.89 h	14.65 h	0.12 s
Models tuned	120	100	8	120	1
Mins per model	0.74	18.01	104.19	7.32	0.00204

The results reveal that increased accuracy comes at a cost. The DFNN, which demonstrates an accuracy gain of 7.32%[45] compared to SSB's model, has a total computation time of 14 hours and 65 minutes. This is a dramatic difference compared to SSB's model which only takes 0.12 seconds to fit. A poor trade-off between accuracy and training time is also found for RF, which underwent 13 hours and 89 minutes of training to achieve less than 8% increase in accuracy[46] compared to the benchmark model. Among the non-linear models, XGBoost achieves by far the best trade-off in this regard, as it is able to outperform[47] all models in a much shorter time. However, it is still substantially slower than the benchmark model. The Stacked Regression model requires the longest time to fit. This is because we not only fit and tune the metalearner algorithm, but also have to account for the time it takes to tune the base learner models. Total training time thus equates to the sum of training time for the XGBoost, RF, DFNN and LM, plus the training time of the metalearner algorithm. The isolated training time of the metalearner algorithm, however, was only 30 minutes.

While DFNN takes the largest tuning time of the individual models[48], it ranks third when considering time per model. Random Forest is the most time consuming in this regard, consuming a total of 104.19 minutes per model fitted. The benchmark LM is the least time consuming, fitting in less than a second. XGBoost seems to be the most efficient

[45]In terms of test RMSE
[46]In terms of test RMSE
[47]In terms of test RMSE
[48]All models except Stacked Regression.

with regards to accuracy and computing time, taking only 44.4[49] seconds to fit each model. The computing times demonstrate that any practical implementation of the above algorithms for housing price prediction should also carefully review the computational power available and the time frame for analysis. However, if the models only need to be re-trained on a monthly or quarterly basis to accommodate new data, computing time should not pose any practical problems for industry application.

5.1.1.3 Model Interpretability

A common critique against machine learning algorithms is their black box nature. Although they are capable of providing highly accurate predictions, they are notoriously hard to interpret. Because of this, machine learning often represents a clear trade-off between accuracy and interpretability. Thanks to recent advancements in the machine learning literature it is increasingly possible to interpret the output of these models and investigate the relationship between the variables. This is important because it allows us to gain an understanding of which features are the most important in predicting housing prices, and in which direction they affect prices.

Several frameworks have been developed to explain the output of different machine learning models (Štrumbelj and Kononenko, 2014; Datta et al., 2016; Ribeiro et al., 2016; Shrikumar et al., 2017; Lundberg and Lee, 2017). In this thesis, we adopt the SHapley Additive exPlanations (SHAP) framework by Lundberg and Lee (2017). SHAP provides a mathematical framework for interpreting the output of machine learning models. Based on the concept of Shapley values[50], SHAP values represent the average magnitude change in the model output when a feature is hidden from the model (Lundberg et al., 2020). A detailed description of the theoretical framework and relevant plots can be found in Appendix A6 and A6.1. Unlike standard variable importance plots, which only show the relative importance of the features, SHAP plots also show the direction in which a given feature affects the response variable. To apply this framework for the best prediction model in Phase 1 (XGBoost), we utilize the *SHAPforxgboost* library in R by (Liu and Just, 2019). It is important to note that while we can use SHAP to investigate the relationship between variables, we cannot use it to establish causal relationships.

[49] 60s $\times$ 0.74 minutes.

[50] A solution in cooperative game theory that distributes costs or gains depending on the average marginal contributions of the players (Shapley, 1953).

Figure 5.1 shows a SHAP Summary plot. The plot shows that Year and PRom[51] are the most important features in terms of predictive power[52], followed by the location dummies for Grünerløkka and Sagene. A possible explanation as to why year is the most important variable is that it serves as a proxy for both inflation as well as market specific price growth in housing due to increasing demand over time. Hence, it captures the general increase in price levels during the period, which, as expected, explains a lot of the variation in the sales price. The fact that primary area has high predictive power is not surprising, as it is closely related to our response variable and because size is a fundamental metric for evaluating a property.

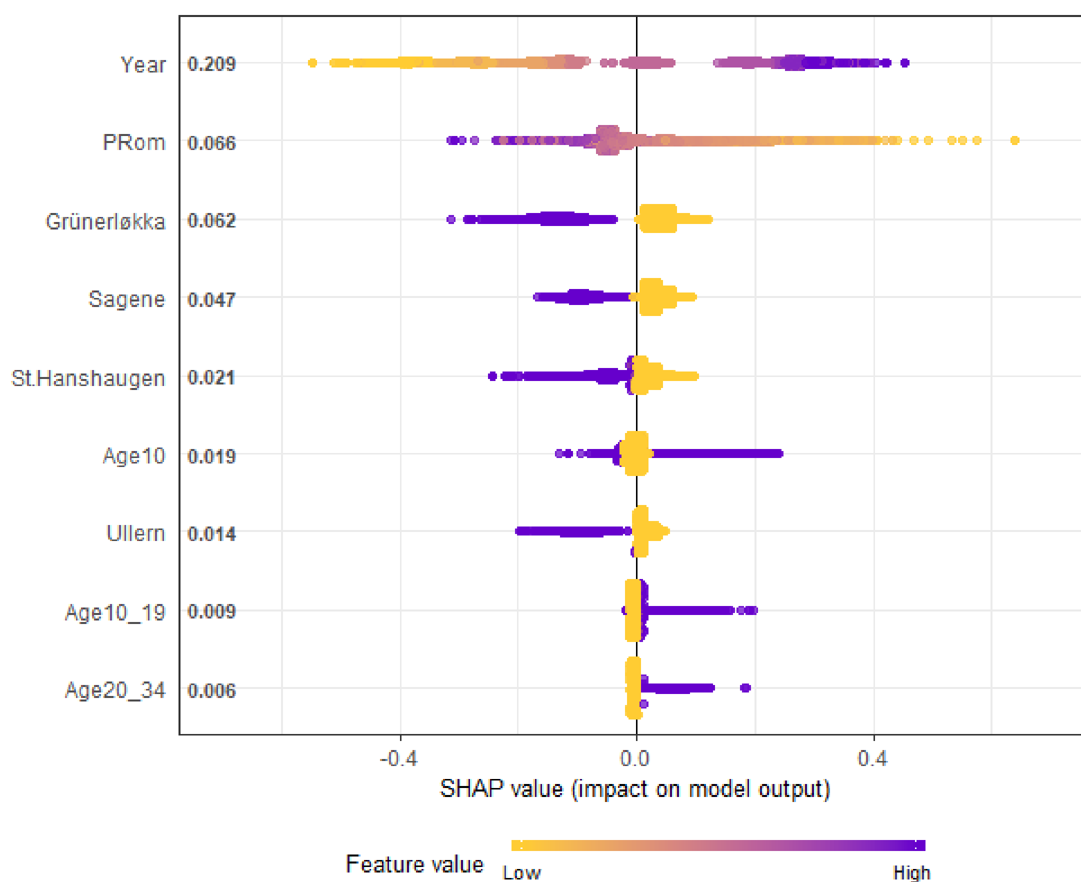
We see from the figure that there is a positive correlation between year and sales price. Lower values for year (yellow) are associated with lower prices, and higher values (purple) are associated with higher prices. This is intuitive given that the feature captures the general increase in price levels. The plot also reveals that there is a negative correlation between the size[53] of an apartment and the sales price. This may seem counter-intuitive at first glance. However, this is because prices are quoted per square meter. Hence, for a given property value, an increase in square meters will lead to a lower price per square meter, all else equal. This should not be confused with total price values, which likely increase with the number of square meters.

The location dummies show the effect of an apartment being located in a particular location relative to Frogner, as this category is the reference category. The plot shows that all locations have a negative effect on the sales price relative to Frogner. This is reasonable, as Frogner is considered to be the most expensive area of the ones covered in our data. For the age dummies, apartments older than 34 years at the time of sale is used as reference. They show that apartments between 10 and 34 years are on average more expensive than apartments older than 34 years. The net effect is unclear for apartments that were built less than 10 years ago, as such apartments are associated both with higher and lower sales prices.

[51] Primary area
[52] SHAP value.
[53] Square meters of primary area

Figure 5.1: SHAP Summary plot Phase 1

This figure displays a SHAP summary plot for all variables used in Phase 1 for the XGBoost model. The vertical axis shows the features ranked according to their importance (average SHAP value per feature across all values of the feature - See Appendix A6.1) and the horizontal axis shows the SHAP value for each observation (dot) for each feature. Each unique transaction is represented by one dot on every row. Clustering of dots indicates higher density. The colors indicate the value of a feature; yellow indicates low values and purple indicates high values. The horizontal location of the dots shows whether the effect of a feature value is associated with a higher or a lower predicted value; it shows the impact that a given feature has on the model's prediction. The dots that lie left of center are associated with lower predicted prices and values that lie right of center are associated with higher predicted prices.



5.2 Phase 2: Additional Variables

Phase 2 implements our models on the expanded data set which includes additional property-specific variables and macroeconomic data. The training data used in this phase consists of 80,781 observations and 60 predictor variables. These variables include the original variables from Phase 1 as well as additional variables such as distance to coast, sunlight conditions, common debt, story, inflation rate and NIBOR. Before training our models, we use the *Boruta* algorithm to perform feature selection on all features. The results from the algorithm suggest that all features should be included for predictive purposes. An in-depth discussion of the results from the Boruta algorithm and possible concerns in this regard can be found in Appendix A5. An overview and discussion of the final tuned hyperparameter configurations for each model in Phase 2 can be seen in Appendix A12.

5.2.1 Model Assessment

The performance of the final models in Phase 2 are summarized in Table 5.4. It reveals that all models perform significantly better when trained on the expanded data set with additional variables.

The Stacked model demonstrates best out-of-sample performance amongst the models, closely followed by the XGBoost and the DFNN. With a test RMSE of 0.1073, the Stacked model demonstrates an accuracy gain of 39.52%[54] relative to the benchmark LM. By back-transforming the Stacked model's predictions, we find that this amounts to a total test RMSE of 5,631 NOK. For the LM, the test RMSE corresponds to 9,778 NOK. Hence, the relative performance between the best model and the benchmark model is consistent with the results from Phase 1, but the difference in accuracy is substantially larger.

Table 5.4: RMSE of final models - Phase 2

Metric	XGBoost	Stacked	RF	DFNN	LM
Training RMSE	0.0721	0.0718	0.0475	0.0992	0.1682
Test RMSE	0.1076	0.1073	0.1197	0.1159	0.1774

The RF model has the lowest RMSE on the training data, suggesting that it captures

[54] In terms of test RMSE.

the training data particularly well. However, the test error reveals that the model is overfitting. This may be a consequence of the lack of tuning for this model in this phase[55]. The remaining models do not show egregious signs of overfitting, indicating a better balance between bias and variance.

While Stacked Regression failed to deliver superior performance in Phase 1, the algorithm now outperforms the best individual model, although by a small margin. This may indicate that the expanded data set with additional variables used in Phase 2 contains more non-linear relationships for each algorithm to learn and that the different algorithms capture different subsets of these relationships, thereby complementing each other better in a Stacked regression model. This seems plausible considering the drastic increase in feature relationships we would expect from increasing the data set to include many times[56] the number of variables as in Phase 1.

5.2.1.1 Accuracy Distribution

To further investigate the performance of the models, we examine the accuracy distribution of the best performing model and the benchmark LM. Table 5.5 shows that the relative differences in accuracy between the models are larger than in Phase 1. Only 1.63% of the Stacked model's predictions have an estimated value that corresponds to less than 80% of the actual market price, whereas 7.87% of the predictions made by the benchmark LM fall in the same group. The results demonstrate that extremely few observations are grossly underestimated by the Stacked model, while the LM is more prone to gross underestimation. Moreover, Table 5.5 that 95% of the predictions made by the Stacked model fall within ± 20% of the actual sales prices. For the LM, 81.14% of the predictions fall in the same range. The Stacked model also appears to be more robust on the right tail of the distribution, as significantly fewer observations are grossly overestimated compared to the LM. We see that there are significant accuracy gains to be made by opting for a Stacked model as opposed to a linear regression model for this particular data. However, both models seem to struggle more with overestimation than underestimation, suggesting that an overall loss in accuracy can to a larger extent be attributed to overestimation. This is consistent with the findings from Phase 1. Predicted versus actual values for both

[55]The default value for mtry was used due to computational limitations.
[56]From 9 to 60 predictors, including dummy variables.

models are visualized in Appendix A10.2, which further shows that the Stacked model is consistently more accurate than the benchmark LM.

Table 5.5: Accuracy distribution for Stacked Regression and LM for Phase 2

This table shows the accuracy distribution of the best performing model (Stacked) and the benchmark LM. Distribution is how much the predicted values correspond to the actual values, divided into 10 groups. n is the number of predictions that fall into a given accuracy group. % is n as a percentage of the total number of observations in the test set. Cumulative % is the cumulative percentages per group. For example, for Stacked, the first row shows that 2 observations have an estimated value that is less than 40% of the actual value and that the predictions in this group accounts for 0.01% of the test data. The table is inspired by Takle and Melby (2020).

	Stacked			LM		
Distribution (%)	n	%	Cumulative %	n	%	Cumulative %
0-40	2	0.01	0.01	23	0.06	0.06
41-60	29	0.08	0.09	175	0.49	0.56
61-80	545	1.54	1.63	2582	7.30	7.87
81-100	17111	48.41	50.04	14705	41.60	49.47
101-120	16467	46.59	96.63	13977	39.54	89.01
121-140	986	2.79	99.42	3116	8.82	97.83
141-160	107	0.30	99.72	456	1.29	99.12
161-180	51	0.14	99.87	122	0.34	99.46
181-200	12	0.03	99.90	57	0.16	99.62
201+	35	0.10	100.00	132	0.37	100.00

5.2.1.2 Computing Time

Table 5.6 shows the computing time and processing specifications for the models in Phase 2. Although the models were not run on similar processing specifications and tuning methods, the time spent is still indicative of computational requirements. All models take significantly longer time to fit compared to Phase 1. However, the relative performance in terms of speed of each model is more or less consistent with our expectations and earlier findings.

The biggest surprise is the increase in Random Forest's computational load. This algorithm has a drastic increase in computing time compared to Phase 1, as it spends 12.53 hours to fit just one model in Phase 2. This is despite reducing the number of trees grown from 1000 to 500. XGBoost once again demonstrates high computational efficiency and has perhaps the best trade-off between prediction accuracy and speed, producing predictions that are

on average 39.35% more accurate than the benchmark while spending only 2.37 minutes on each model. While demonstrating superior prediction accuracy, the Stacked model is still the slowest model to fit since its computing time also includes tuning of each base learner. If we disregard the base learners, the isolated training time for the metalearner in the Stacked Regression was 164.42 minutes. The benchmark LM requires no tuning and is fitted in less than a second. However, it has the worst out-of-sample performance among the models, resulting in a poor trade-off between accuracy and efficiency. The results thus far show that significant prediction accuracy gains can be made if one is willing to invest up to 35 hours to tune the prediction model chosen. As long as models are not required to be re-trained frequently to accommodate new data, this should not pose a barrier for industry adoption.

Table 5.6: Model Tuning and Computing Time Comparison for Phase 2

This table provides an overview of the tuning methods, time spent, and computational platform for each fitted model in Phase 2. The models were run on machines with varying specifications. XGBoost and DFNN were fitted using GPUs for accelerated processing, while the LM, Random Forest and the Stacked Regression only used CPUs. Computing time is measured as time from the start of model tuning until the final optimal model has been fitted. Mins per model is the average time in minutes consumed per model tuned for each algorithm. Computing time for Stacked Regression includes both model fitting time and tuning of the base learners.
*Denotes system CPUs available, not necessarily parallel workers used. **GPU-memory

	XGBoost	Stacked	RF	DFNN	LM
Tuning method	Random	Auto	NA	Random	NA
Computing time	4.74 h	34.82 h	12.53 h	17.55 h	0.5513 s
Models tuned	120	1	1	120	1
Mins per model	2.37	2089.21	751.8	8.77	0.0092
Processors*	1 GPU	16 CPU	32 CPU	1 GPU	16 CPU
Memory	16 GB**	64 GB	128 GB	12 GB**	64 GB
Ubuntu version	18.04	18.04	18.04	16.04	18.04

5.2.1.3 Model Interpretability

To investigate which features are most important in this phase, we compute and visualize their SHAP values[57]. Figure 5.2 shows a SHAP summary plot for the 10 most important features used in Phase 2. The plot reveals that inflation, sales year and primary area (CPI,

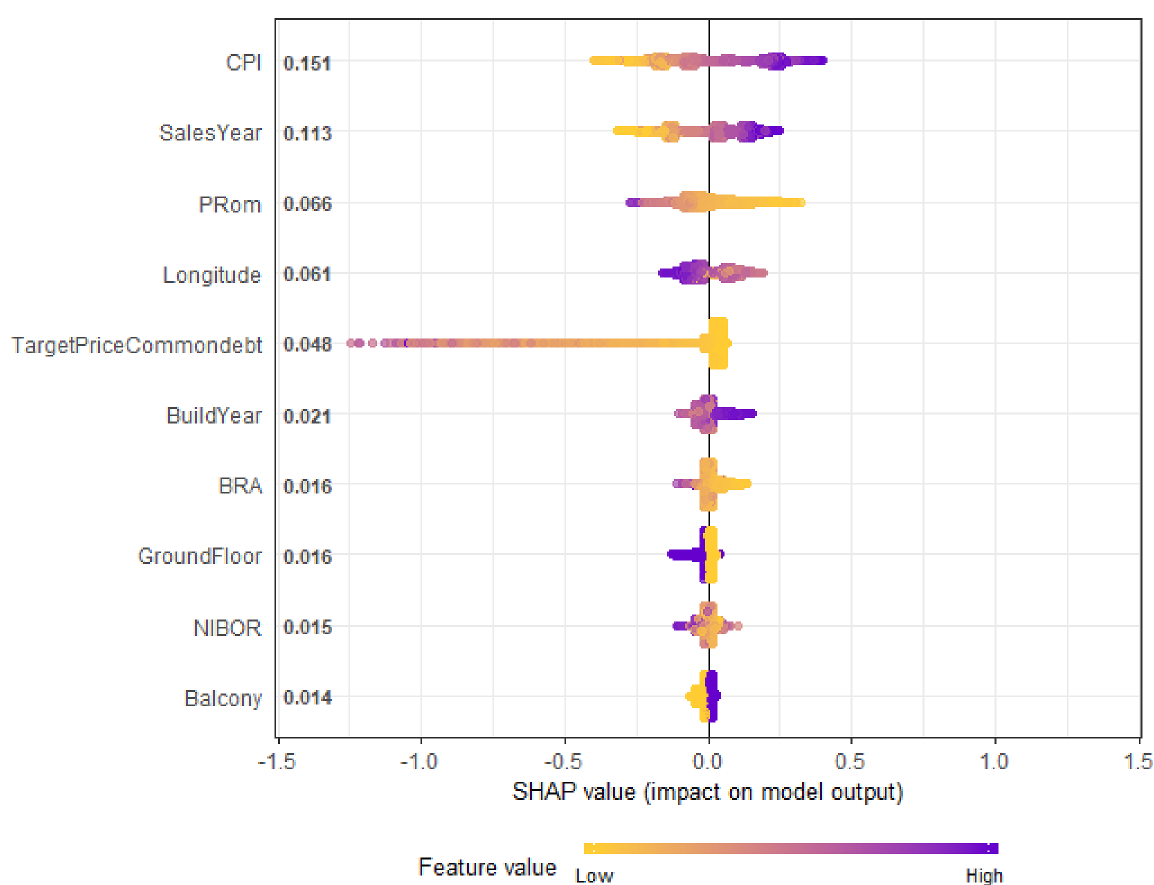
[57]SHAP values are computed using XGBoost instead of Stacked Regresion. This is because the *h2o* package in R does not offer SHAP compatibility for Stacked models.

SalesYear, and PRom) are the most important features for predicting property prices. The fact that inflation and sales year are the two most important features corroborates the findings from Phase 1 that general increases in price levels have strong predictive power for property prices. The figure indicates that higher inflation levels are associated with higher predicted prices and vice versa. The negative relationship between prices and primary area persists in this phase as well, which is reasonable given that we are still predicting prices per square meter. Longitude appears to be the most important geographical feature for predicting prices. The fact that none of the location dummies nor latitude are among the top features suggest that it is the east-west position of an apartment that is the more important aspect in terms of prices.

We also see that both high and low values for TargetPriceCommondebt, which is the shared debt for housing cooperatives, are associated with lower predicted prices on average. This suggests that common debt has a negative impact on the value of an apartment relative to independently owned apartments without such debt, and that the magnitude of the debt matters. This is expected because higher levels of common debt will result in higher monthly debt service, thus warranting compensation to the buyer in the form of a lower up-front transaction price. BuildYear and Balcony appear to be positively correlated with prices. This is reasonable, as newer apartments with balconies are typically more desirable. Apartments situated on the ground floor are associated with lower prices relative to middle floor apartments. Lastly, higher levels of NIBOR are associated with lower prices, and vice versa. As previously mentioned, this is consistent with financial theory that higher mortgage rates reduce demand for housing, which in turn leads to lower prices.

Figure 5.2: SHAP Summary plot Phase 2

This figure displays a SHAP summary plot of the 10 most important features for XGBoost in Phase 2. The vertical axis shows the features ranked according to their importance (average SHAP value per feature across all values of the feature - See Appendix A6.1) and the horizontal axis shows the SHAP value for all observations for each feature. Each unique transaction is represented by one dot on every row. Clustering of dots indicates higher density. The colors indicate the value of a feature; yellow indicates low values and purple indicates high values. The horizontal location of the dots shows whether the effect of a feature value is associated with a higher or a lower predicted value; it shows the impact that a given feature has on the model's prediction. The dots that fall left of center are associated with lower predictions and values right of center are associated with higher predicted values.



To further investigate the relationships between the features and the response variable, we plot the SHAP values for a feature against the values of that feature. This is known as SHAP Dependence plots. A SHAP dependence plot for CPI, NIBOR, Primary Area and Longitude is shown in Figure 5.3. The CPI plot clearly demonstrates a positive linear correlation between the inflation index and apartment prices. This further illustrates that inflation explains a lot of the price variation in the data. The plot also shows that there

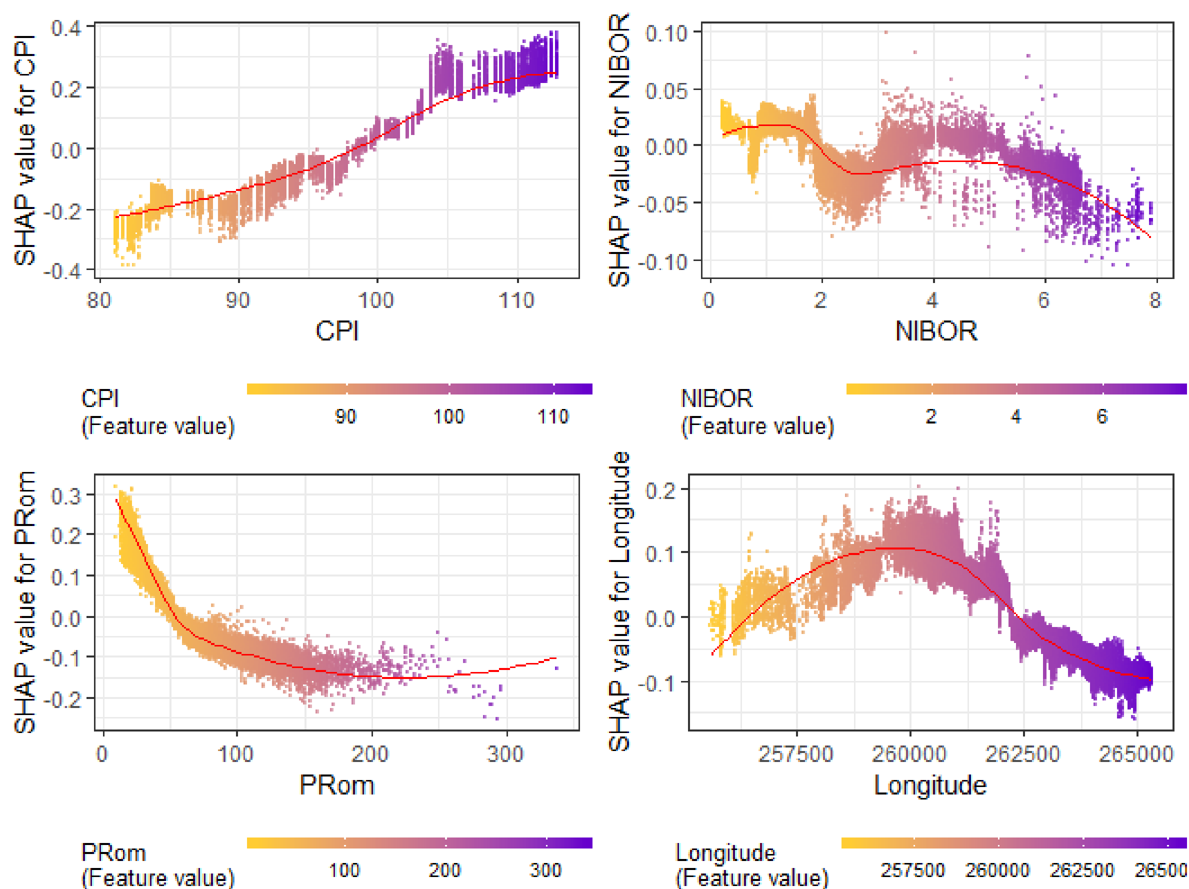
is a negative correlation between NIBOR and sales price, but the relationship is less clear. Low NIBOR rates are indeed associated with higher average prices, but prices seems to respond slowly to changes in NIBOR. Average prices stay roughly the same when NIBOR increases from 0.2-1.5% but decrease rapidly from 1.5-2.5%. From 2.5-5% prices seem to increase modestly before decreasing sharply for NIBOR rates above 5%. One plausible explanation for this is that it usually takes some time before changes in NIBOR result in changes in mortgage rates. It is important to note that there are large variations inherent in both CPI and NIBOR. This is revealed by the large vertical spread associated with each feature value, indicating that there may be interaction effects between these variables and other variables.

The plot for PRom exhibits a clear negative relationship between the price per square meter and size of an apartment. However, the relationship is not linear. Instead, the plot reveals that there are diminishing returns associated with square meter prices and size; prices decrease sharply up to a certain point before flattening out for apartments that are larger than 100 square meters. This relationship seems to hold up to 250 square meters in size, after which the price per square meter starts to increase again. However, we should be careful not to establish general patterns for apartments larger than 200 square meters, as we have limited data for apartments of this size.

The bottom right plot shows the relationship between Longitude and price per square meter. We see that there are clear price trends related to the east-west location of an apartment. Lower values for longitude, which corresponds to Ullern in our data, are associated with lower average prices. This is also true for apartments located in eastern parts of Oslo, such as Grünerløkka. Middle values, on the other hand, corresponding to Frogner and St. Hanshaugen, are associated with significantly higher average prices. This is consistent with empirical price data for Oslo. The large vertical dispersion of dots indicate that there are large variations in how important longitude is in prediction of prices, and that there may be interaction effects at play between longitude and other variables.

Figure 5.3: SHAP Dependence plot Phase 2

This figure shows SHAP Dependence plots for CPI, NIBOR, PRom and Longitude. The vertical axis in each of the four plots indicates the SHAP value for each observation (change in the margin output of the model - see Appendix A6.1) and the horizontal axis shows the value of the feature. Each dot corresponds to a sales transaction. Dots located higher in the two-dimensional space have a feature value that is associated with higher predicted prices and vice versa. Yellow indicates lower values for a feature and purple indicates higher values. The vertical dispersion of dots for a given feature value indicates the range of predicted prices associated with that feature value. High vertical dispersion may indicate interaction effects. The red curve shows the locally estimated scatterpot smoothing (LOESS) curve.



5.3 Summary of Results

A summary of our best performing models and the benchmark models from both phases is presented in Table 5.7.

Table 5.7: Summary of results from Phase 1 and Phase 2

	Phase 1			Phase 2		
	XGBoost	SSB	Improvement	Stacked	LM	Improvement
Test RMSE	0.1851	0.2023	**8.50%**	0.1073	0.1774	**39.52%**

The results of our Phase 1 analysis show that XGBoost has the best out-of-sample performance, with 8.5%[58] increased prediction accuracy compared to the benchmark linear model currently employed by SSB. As with most machine learning algorithms, this comes at the price of reduced insight into the resulting model and increased computational requirements. However, frameworks such as SHAP mitigate these issues by offering visual insights into the relationship between the response variable and the predictors. By implementing the SHAP framework, we find that sales year and the square meterage of an apartment are the most important predictors for apartment prices given that an XGBoost model is used.

As for computational requirements, the machine learning models are very slow compared to the benchmark SSB model. If accuracy is the main concern, our results clearly indicate that XGBoost yields the best results. Furthermore, the analysis reveals that all our non-linear machine learning algorithms improve the accuracy of predictions, suggesting that the non-linear algorithms capture more complex relationships in the data than the linear model. Immediate improvements can thus be made to the benchmark model simply by changing the algorithm.

The results from Phase 2 demonstrate even starker differences between the non-linear machine learning algorithms and the LM benchmark in terms of prediction accuracy. By including additional property-specific and macroeconomic variables, as well as implementing different pre-processing and feature selection methods, we find that the prediction accuracy of all models is improved. The expanded data set contains more information and more feature relationships. The non-linear machine learning algorithms consequently increase their performance lead over the linear model. In terms of variable importance[59], we find that inflation (CPI), primary area and longitude are the most important predictors of housing prices, where the two latter clearly demonstrate non-linear

[58] Percentage reduction in test RMSE

[59] For the XGBoost model.

relationships with housing prices.

The two best models from Phase 1, XGBoost and Stacked Regression, retain their leading positions in terms of out-of-sample prediction accuracy in Phase 2. The Stacked model has the best performance, with an increased accuracy of 39.52%[60] compared to the benchmark LM. The improvements in accuracy come at a cost: an increase in the number of variables increases the computational requirements for each model despite using more powerful machines and enabling GPU computation. Although prediction accuracy is our main concern in this thesis, it should be noted that the Stacked model requires the longest time to tune and fit, since it requires individual tuning of each base learner. Because of this, XGBoost, performing only marginally poorer and requiring the least tuning and fitting time, is perhaps the better trade-off in a practical situation if computational requirements and training time are important factors. However, the tuning times of both XGBoost and Stacked Regression are reasonable as long as industry applications do not require models to be re-fit frequently (e.g. daily or weekly) to accommodate new data. In summary, the results from Phase 2 indicate that considerable prediction accuracy can be gained by combining non-linear machine learning algorithms with data that includes additional property-specific and macroeconomic variables such as CPI, longitude, common debt and NIBOR.

While we cannot draw conclusive comparisons between the models in the two phases due to differences in sample size, pre-processing and feature engineering, we can still gain valuable insight and indications about the relative performance of the models. Table 5.8 shows the back-transformed test RMSE in NOK for the Phase 2 Stacked Regression and Phase 1 SSB model for the period 2010-2019. The overall back-transformed[61] Phase 2 Stacked model's test RMSE of 4,918[62] NOK per square meter is much lower than the Phase 1 SSB benchmark model's test RMSE of 11,415 NOK. The table also reveals that the Phase 2 Stacked model yields a considerably lower RMSE for all price ranges, especially for the tails of the distribution. For the 25th percentile, the Stacked model's RMSE of 2,348 NOK corresponds to a reduction in RMSE of 57,42% compared to the Phase 1 linear benchmark for the same percentile. Although the Stacked model undoubtedly

[60]Percentage reduction in test RMSE.

[61]e raised to the power of x where x is the natural logarithm of the predicted price for instance x.

[62]This deviates from the back-transformed Phase 2 Stacked RMSE presented in Phase 2 because the test sample is restricted to 2010-2019 in this section to increase comparability across the two phases.

demonstrates prediction accuracy gains compared to the benchmark model, such accuracy improvements must be considered in light of the increased computational requirements discussed earlier and a potential loss of interpretability for real-world applications.

Table 5.8: Comparison of RMSE between Stacked and SSB's model

In this table the predictions of our best model (Phase 2 Stacked Regression) are compared with SSB's model (Phase 1) grouped by quartiles of actual square meter prices. Predicted is the average predicted price per square meter per quartile, Actual is the average true price per square meter per quartile. RMSE is test RMSE per quartile. Stacked Regression (Phase 2) test sample has been limited to observations for the period 2010-2019[63] in order to ensure a more reliable comparison.

	Stacked			SSB		
Percentile	Predicted	Actual	RMSE	Predicted	Actual	RMSE
25	41282.93	39709.72	2348.49	45509.70	39829.34	5515.71
50	54221.13	53985.71	2385.48	54203.40	54162.30	4064.30
75	66974.91	67031.44	2801.56	65869.30	67006.73	4720.25
100	85275.56	88017.99	4078.96	78270.28	87717.33	7815.19
All	61944.20	62192.60	4917.77	60963.25	62179.06	11414.88

Similar performance is seen on an aggregated level. Table 5.9 shows cumulative[64] predicted property values compared to the actual[65] cumulative property values for the Phase 2 Stacked model and the SSB linear benchmark model in each quartile and in total[66].

[64] Sum of all property values within a quartile or in total.
[65] Sum of the true transaction prices
[66] These values are found by multiplying the *predicted* and *actual* square meter prices with each property's primary area in the test set and computing the sums of the resulting property values.

Table 5.9: Total property value estimates

This table shows total predicted and actual property values in billion NOK grouped by quartiles of actual total property values for the Phase 2 Stacked model and Phase 1 SSB benchmark model. Predicted and Actual are the sum of the predicted and actual property values for each quartile and for the entire test set. Error is how much the difference between the sum of the predicted and actual property values account for the actual property values. The sample is restricted to only consider predictions for the period 2010-2019 for both models in order to make predictions across the two phases more comparable.

	Stacked			SSB		
Percentile	Predicted	Actual	Error (%)	Predicted	Actual	Error (%)
25	11.94	11.66	2.34	12.75	11.71	8.83
50	17.89	17.74	0.84	17.80	17.73	0.45
75	23.74	23.77	-0.14	23.59	23.63	-0.15
100	38.61	39.44	-2.11	36.18	39.25	-7.83
All	92.17	92.61	-0.48	90.32	92.31	-2.16

The table shows that the SSB linear model underestimates the sum of total property values by 2.16%, compared to only 0.48% for the Phase 2 Stacked model. As such, both models demonstrate a small negative bias in predictions on the test set. The per-quartile breakdown in the table provides a more nuanced picture of the models' biases. Both models overestimate the cheapest properties (25th and 50th percentiles) and underestimate the more expensive properties (75th and 100th percentile). They have somewhat similar errors for the 50th and 75th percentile, with the SSB benchmark model outperforming the Stacked model in the 50th percentile. However, the Stacked model has a considerably lower error than the SSB benchmark for the 25th and 100th quartiles, corresponding respectively to the least and most expensive properties. Once again, this may indicate that our non-linear machine learning models are able to capture more feature relationships than the linear benchmark and that property values vary in a non-linear fashion.

In conclusion, our Phase 1 analysis demonstrates that switching from linear models to non-linear machine learning algorithms can yield improvements in prediction accuracy, likely due to the prevalence of non-linear feature relationships in the data. Phase 2 demonstrates that utilizing non-linear machine learning algorithms on expanded data sets with additional property-specific and macroeconomic variables can yield significant improvements in accuracy over linear models. The improvements demonstrated may have

considerable economic impact if adopted, both for the benchmark SSB model's original taxation purpose and for other applications. The economic impact of increased prediction accuracy will be discussed in the following chapter.

6 Discussion

In this chapter we discuss some of the economic implications of our results as well as the trade-offs from adopting machine learning models with regards to interpretability. We also highlight the limitations of this thesis and promising avenues for further research.

6.1 The Economic Implications of Improved Accuracy

Our analysis demonstrates that there is a large potential to improve estimates of property values by adopting non-linear machine learning algorithms and including additional property specific and macroeconomic variables in the data set. This suggests that there also exists a strong potential for economic gains to be made. For one, the residential real estate market is by itself a huge market where the property value of a house is a key metric for all players involved. If we also consider ancillary services and industries which in some way utilize property prices, there is clearly potential for economic gains from more accurate housing price predictions.

The applications for improved prediction models are numerous. A natural point of departure for discussion in this thesis is tax estimation for property and wealth taxes, since this is the original application of the SSB benchmark model. However, we also see potential benefits in credit evaluation, property transactions, risk assessment and valuation of financial instruments. In the following sections we discuss these applications in an economic context.

6.1.1 Property and Wealth Tax Estimation

There are two kinds of property tax in Norway: a direct property tax[67] and an indirect wealth tax. Details relating to each tax are listed in Table 6.1. Property tax is charged as a percentage of a property's value minus a minimum deduction. Wealth taxes are charged as a percentage of a person's net wealth, including property value, that exceeds a minimum threshold. By law, both these taxes are calculated as a function of a property's presumed market value, which we in this thesis assume to be the price a property would fetch in an open market sale at any given time. This is also the assumption about market

[67] Only applicable in certain municipalities, such as Oslo.

prices made by Norwegian tax authorities, since they also accept observed transaction prices as documentation for market value (Skatteetaten, 2020). The *correct* taxes, as decided by society through legislation, should therefore reflect true property market values. Given existing tax policies, correct taxation is therefore the situation where everyone pays taxes according to the true market value of their properties.

Table 6.1: Property and Wealth taxes Applicable to Oslo

This table presents an overview of the information used to assess taxes. *Tax rate specific for Oslo. **Minimum standard deduction for Oslo. ***The activation threshold after which the wealth tax is triggered based on net wealth.

	Tax base	Tax rate	Minimum standard deduction
Property tax	70% of market value	0.30%*	4,000,000 NOK**
Wealth tax	25% of market value	0.85%	1,500,000 NOK***

As discussed in the introduction, the market value of a property is easy to establish if a recent transaction has taken place, but hard to find if the previous transaction was long ago. Norwegian tax authorities therefore use SSB's linear model (our Phase 1 benchmark) as an AVM to estimate property market values for tax purposes. An issue with this approach is that the SSB model's estimation may be relatively inaccurate, as discussed in Section 5.3. This is especially problematic if the SSB model systematically under- or overestimates certain price ranges or wealth brackets, as our results seem to indicate. Any such bias could shift the weight of tax burden to specific brackets of the population, typically in the same socio-economic group. A more precise model, such as our Phase 2 Stacked model, may provide more accurate assessments of property values and thus more *correct* taxes. In the following two sections we will examine model-specific differences in tax estimations between the SSB benchmark model and our best performing Phase 2 model, Stacked Regression.

6.1.1.1 Property Taxes

We use our test data from Phase 1 and Phase 2 to predict the property tax effects of the Phase 1 SSB model and our Phase 2 Stacked model. By doing this we assume that our transaction data can act as an acceptable proxy for the real property distribution in the five boroughs[68] it covers. In order to increase comparability across the two phases, we

[68]Frogner, St. Hanshaugen, Grünerløkka, Ullern and Sagene.

restrict the sample size for the Stacked model to only include observations from 2010 to 2019[69]. Differences in sample sizes between the two phases due to different pre-processing regimes and additional variables still inhibit an accurate and conclusive comparison of results. Moreover, since we do not split the observations by individual years, the estimates made only reflect the approximated relative sizes of the tax burden. Even so, we believe such an exercise to be useful as an indication of the effects of model choice on taxation.

First we look at the aggregated *predicted*[70] property taxes compared to the aggregated *correct*[71] taxes. Predicted and correct property taxes are computed as $\max\{0,\ ((Property\ value \times 0.70 - 4,000,000) * 0.003)\}$, in accordance with Table 6.1. A detailed description of how predicted and correct taxes are computed for this example can be seen in Appendix A13.

Since our test samples only act as a proxy for the real distribution of properties in the boroughs, we are only interested in the relative differences between *predicted* taxes due and *correct* taxes due. Table 6.2 shows the results of this exercise.

Table 6.2: Total Tax Liability per Model

This table shows the property tax liabilities of the properties in the SSB benhcmark LM (Phase 1) and Stacked Model (Phase 2) test sets. Predicted taxes are the sum of property taxes calculated using each model's predicted property values as base value. Correct taxes are the sum of property taxes calculated using the true market values (transaction prices) of each property in the test sets. Error is the percentage difference between correct and predicted taxes.

	SSB	Stacked
Predicted taxes	9,876,247	13,511,782
Correct taxes	13,940,275	14,568,365
Error	**-29.15%**	**-7.25%**

Both the Stacked Regression and the SSB benchmark models underestimate total property tax liability, but the degree of underestimation differs considerably. The benchmark model underestimates total taxes by 29.15%[72], while the Stacked model underestimates by only

[69] In accordance with the Phase 1 test sample.

[70] *Predicted* taxes are calculated based on the predicted market value of a property. This corresponds to the back-transformed predicted $\hat{Y}$ variable from our analyses multiplied with the true apartment size.

[71] *Correct* taxes are calculated based on the true market value of a property. This corresponds to the back-transformed true Y variable in our test set multiplied with the true size of the apartment.

[72] Difference between the sum of predicted and actual taxes divided by actual taxes for the entire test set for both phases.

7.25%. The difference in underestimation between the models is significant considering that Oslo municipality collected NOK 655.5 million in residential property taxes in 2019 (SSB, 2020).

To further investigate tax effects, we isolate the properties who *should* pay property taxes[73] and those who *should not* pay property taxes. The proportion of properties in each Phase's test sample that should and should not pay property taxes according to their transaction price is shown in Table 6.3.

Table 6.3: Proportion of Properties Liable for Property Tax

This table shows the proportion of observations in each Phase's test sample that are and are not liable to pay property taxes based on their actual market values (transaction prices). Properties that should pay property taxes have an actual market value × 0.70 in excess of NOK 4 million.

	Phase 1 SSB	Phase 2 Stacked
Should pay	13.97%	14.17%
Should not pay	86.03%	85.83%

We first consider the properties whose actual market values imply that they *should not* pay property taxes. This is done to investigate whether any of the models overestimate property values to the extent that they become liable for property taxation[74] when in reality they should not pay property taxes. For the SSB benchmark model, 2.19% of properties in our test set are liable to pay property taxes despite their true market value indicating they should not. For the Stacked model, only 1.5% of properties are unduly charged due to overestimation. Hence, although the absolute difference is small, the Stacked model reduces the proportion of homeowners who are unduly charged with property taxes by 31.5%. Assuming the relative differences between the models can be extrapolated to the whole of Oslo, a total of 5279[75] properties are unduly charged by the SSB model, and 3616[76] properties are unduly charged by the Stacked model. Roughly 1,663 fewer apartments will be charged with property taxes by adopting the Phase 2 Stacked model for housing price prediction.

[73] All properties whose actual market value (transaction prices) × 0.70 exceeds 4,000,000 NOK.

[74] Observations whose true market value × 0.70 is below NOK 4 million, but whose predicted market value × 0.70 is above NOK 4 million

[75] 241,058 apartments in Oslo × 0.0219

[76] 241,058×0.015

Next, we consider the properties whose actual values imply that they *should* pay taxes (13.97% of observations in Phase 1 and 14.17% of observations in Phase 2) to examine the accuracy of the tax estimates for different property value brackets. A comparison between the tax predictions of our best model and the benchmark SSB model is shown in Table 6.4. The table reveals that the SSB model underestimates total property taxes by almost 33%, while the Stacked model only underestimates taxes by 9.02%[77]. The table also breaks down these tax predictions by quartile of actual total property values. With an error range spanning from -3.74% to -12.17% for the 50-100th percentiles, the Stacked model seemingly estimates property taxes much more accurately than the benchmark SSB model, whose error ranges from -30.68% to -36.16% of actual tax for the same percentiles. The exception is for the 25th percentile, where the SSB model is more accurate.

Table 6.4: Property tax estimation by quartiles of actual property values

This table shows the predicted and actual property taxes in millions NOK for each model grouped by quartile of actual property values in the test set. The test sample is restricted to homeowners whose actual property values imply that they *should pay* property taxes. Predicted and Actual are the sum of the predicted and actual property taxes for each quartile and for the entire (restricted) test sample. Error is how much the difference between the sum of the predicted and actual property taxes account for the actual taxes for each quartile. The sample is restricted to only consider predictions for the period 2010-2019 for both models in order to make predictions across the two phases more comparable.

	Stacked			SSB		
Percentile	Predicted	Actual	Error (%)	Predicted	Actual	Error (%)
25	0.67	0.52	29.81	0.62	0.53	15.67
50	1.55	1.61	-3.74	1.09	1.57	-30.68
75	3.12	3.43	-9.09	2.18	3.29	-33.73
100	7.91	9.01	-12.17	5.46	8.55	-36.16
All	13.25	14.57	-9.02	9.34	13.94	-32.99

Assuming that correct taxes are the ones that are computed from the actual tax values based on the true market value of a property, our findings indicate that adopting non-linear machine learning algorithms on expanded data sets can improve the accuracy of tax estimates. The Phase 2 Stacked model considerably improves the accuracy of the overall tax take. We also find that the Stacked model reduces the number of properties subjected

[77] These values differ from previous results (29.5% and 7.25%) because the sample is now restricted to those properties whose actual values imply they should pay property taxes.

to undue tax charges by 31.5% compared to the SSB model. While the relatively poorer performance of the benchmark model likely can be attributed to non-linear relationships in the data, it may also be due to omitted variable bias in the SSB linear model. Our results indicate beneficial economic gains in taxation applications can be achieved by adopting non-linear machine learning models such as our Stacked model.

6.1.1.2 Wealth Tax

Wealth tax effects are more difficult to estimate than property taxes. This is because people are taxed based on their net wealth exceeding the activation threshold, and not on an asset-by-asset basis. Hence, we cannot provide wealth tax estimates for property values since we do not know each property owner's holdings of debt or other assets. However, we *can* say something about the models' effects on properties' wealth tax base value, which is set by law at 25% of market value. As with property taxes, this market value is usually estimated using SSB's linear model (our Phase 1 benchmark). The relative inaccuracy of the SSB model compared to our Phase 2 Stacked model demonstrated in Section 5.3 indicates that the Stacked model can provide more accurate predictions of property market values and thus property wealth tax base values. This is especially true for the least and most valuable properties. Considering the fact that over 511,000 Norwegians had a net wealth that exceeded the activation threshold in 2018 (SSB, 2018), the change in levied taxes that stems from more accurate estimates could be profound.

While we leave any discussions about the morality and utility of property and wealth taxation to others, our findings suggest that implementing non-linear machine learning algorithms to predict property values can contribute significantly to producing more *correct* taxes on an individual and aggregated level. Any practical application to this effect must however weigh the benefits from increased accuracy against any loss in transparency, as we will discuss in Section 6.2.

6.1.2 Other Applications

Improved prediction accuracy can yield economic benefits in numerous business applications other than taxation. However, due to the proprietary nature of these applications in business practices, it is harder for us to estimate quantitative effects.

Economic implications for some applications also require extensive research before quantification can be made, and thus lie outside the scope of this thesis. In this section we will nevertheless outline some potential uses for housing price predictions and describe the general effects from improving predictions of housing prices.

An obvious benefit from improved housing price predictions is better transparency and risk reduction in second hand real estate markets. Individuals seldom have extensive knowledge about the housing market or particular properties, making it hard for them to make accurate value assessments of a property. Instead, they rely on physical appraisals and model valuations. However, we have seen that common valuation models such as SSB's linear model can be relatively inaccurate. Similarly, research shows that up to 74% of properties are sold above the appraised value (Benedictow and Walbækken, 2020). Such inaccuracies can lead to uncertainty about the fair market value of a property, and subsequently cause considerable uncertainty and friction in the market. Research also indicates that appraisals have an anchoring effect (Benedictow and Walbækken, 2020), limiting the range of bids in housing auctions. Consequently, adopting AVMs for housing prices that increase prediction accuracy can help by reducing buyer uncertainty and helping the market find true property value. This could benefit both buyer and seller by enabling a fair exchange while mitigating problems relating to asymmetrical information.

Better prediction accuracy may also have positive economic effects for residential mortgage refinancing. A bank needs an accurate valuation of a mortgage's underlying property in order to negotiate appropriate loan terms for new customers wishing to refinance a mortgage from another bank. This is needed both for fulfilling regulatory requirements (such as Loan-to-Value (LTV) limits) and for the bank's own profitability and risk requirements. Finding the market value of a property is trivial for fresh mortgage applications, since this corresponds to the transaction price. For mortgage refinancing, however, transaction prices are usually dated, and the property market value must be estimated. Physical appraisals for this purpose are impractical due to their cost. Many banks therefore prefer using AVMs to estimate the current property value (Lund, 2020). Applying non-linear machine learning algorithms to improve accuracy of housing valuation can therefore yield economic benefits in refinancing situations as it allows the banks to provide more appropriate loan terms to each customer, thereby reducing risk and increasing competitiveness. Better

predictions of property values will also enable banks to offer more appropriate loan terms in other types of financing situations, as they will allow banks to gauge each individual's net worth and credit worthiness better.

Banks also use AVMs to monitor the value of the collateral values of their residential mortgage portfolios. These portfolios account for approximately 42.6% of Norwegian banks' loan portfolios (Jansrud, 2017), and are usually secured on the underlying property. It is therefore imperative that banks monitor the value of their collateral portfolio in order to ensure prudent risk management. Increased accuracy from AVMs through the use of non-linear machine learning models can potentially provide a better overview of the total value of the collateral property and thus improve banks' risk management and profitability.

Finally, improved property price prediction accuracy may yield benefits in pricing and monitoring of housing-related financial instruments whose value depends on the underlying property value, such as mortgage-backed securities and covered bonds[78]. Norway has a large covered bond market that may directly benefit from adopting non-linear machine learning models to monitor the value of underlying properties. The outstanding volume of Norwegian covered bonds amounted to NOK 1158 billion in 2017 (Finance Norway, 2018). These are backed almost exclusively by residential mortgages. Since these bonds are ultimately secured by the value of the underlying properties, regulation and prudent risk management requires that the LTV ratio of each property is monitored and updated regularly (Finance Norway, 2018). Being able to make accurate predictions of property values is therefore important to ensure fair prices and low risk. Norwegian covered bond issuers typically monitor property valuations of the underlying portfolio based on Eiendomsverdi's proprietary AVM (Finance Norway, 2018). Any increase in prediction accuracy that can be gained through adoption of non-linear machine learning algorithms such as those we have developed can thus add value by reducing risk and contributing to improved asset pricing.

[78] Covered Bonds are debt securities issued by banks and mortgage institutions that are collateralized against a pool of underlying assets (BIS, 2014).

6.2 Interpretability and Industry Acceptance

As previously mentioned, a common critique of machine learning methods is the black box nature of many algorithms. This characteristic makes it difficult to observe what actually goes on in a particular model, and why its predictions turn out as they do. Even when using advanced methods for interpretation such as the SHAP-methods employed in this thesis, interpretability is still much more limited than linear models and other classic approaches. SHAP approaches will for example demonstrate general relationships between the features, but cannot say anything about their incremental impact in the same way that linear regression coefficients can. Such losses in interpretability represent important obstacles to adoption of machine learning models, and there is still some distance to travel before models such as the ones developed in this paper can see widespread adoption. According to Lund (2020), many commercial customers of AVM services today require in-depth insight into the workings of the models employed and the effects of different variables in the data set. They may need such information for a variety of purposes, including regulatory requirements, their own due diligence requirements, or because their customers demand it in turn.

While the discussion in the section above indicates that our models can yield economic benefits, difficulties with interpretability suggest that adoption of machine learning will be carefully scrutinized despite the advantages in prediction accuracy. Adoption will most likely be staggered across industries and subject to individual customer needs. Machine learning models such as ours will likely not supplant existing methods for the foreseeable future, but we expect to see an increase in their use to complement existing methodologies. We believe that the growth of big data, enhancements in computing power and the potential gains in accuracy ensure an economic future for machine learning in housing price prediction.

6.3 Limitations and Further Research

The methodologies and analyses in this thesis have certain limitations. Some stem from implementation issues, some from resource limits, some from data availability and others from methodological choices we have made.

A general limitation of our thesis is the limited scope of our data, which only covers five boroughs in Oslo. While this still suffices to answer our research question, we cannot expect the results to generalize to Oslo or Norway as a whole. These five boroughs also have very few observations of other housing types than apartments, forcing us to focus only on this type of housing. Furthermore, the five boroughs may not be representative for Oslo as a whole, as they are commonly considered among the more popular and expensive districts. This may have biased our results. Future analyses on this topic should preferably cover larger parts of the country and incorporate housing types such as detached houses, duplexes, and semi-detached houses.

In addition to comparing the performance of our machine learning models with our benchmark model, we have also compared our machine learning models with each other in terms of prediction accuracy and computing time. A fair comparison should also ideally involve setting equal computational budgets for each model, for example by allowing each model to tune and train for a predefined length of time. Despite the large differences in model fitting time we have not done so in this thesis, mostly due to resource constraints. An ideal analysis would include sufficient computing power to implement computational budgets without sacrificing model variety.

A related issue concerns the machines used to run our Phase 2 analyses. These were run on AWS virtual machines. Due to AWS policies that required us to demonstrate usage before ramping up to our desired specifications, some of our models in Phase 2 were run on more powerful machines than others. These machines should ideally have been similarly configured. However, the end results in terms of accuracy and time still seem ordinally in line with expected performance.

The limited resources available in terms of time and computing power also limited the thoroughness of our analysis. As we have seen, many machine learning models take a long time to train and fit. This problem is compounded by the need for hyperparameter tuning. The computational resources available to us included personal laptops and limited AWS virtual machines. This severely limited the scope of our hyperparameter tuning for each model in both phases. Although we still believe our results are indicative of each algorithm's performance, we cannot conclude that changes to the hyperparameter configurations would not lead to different results in terms of which models perform best.

In general we believe there is potential to improve each model by increasing the scope of hyperparameter tuning, especially so for the DFNN and XGBoost models, both of which sampled only a fraction of possible hyperparameter combinations. Limits due to the available computing power also required us to reduce the number of trees and abandon tuning of the Random Forest model in Phase 2. Future implementations should seek to conduct even more extensive hyperparameter tuning, including separate tuning of the Stacked Regression's base learners.

The software packages used to implement our models also impose certain limitations. Our implementation of the DFNN uses the *keras* package in R. While this package is widely renowned and versatile, it does not allow for easy implementation of k-fold CV resampling when conducting random hyperparameter search. Our implementation therefore makes use of the validation set resampling method for the DFNN. This differs from the k-fold CV resampling method used by the other models. Similarly, the *h2o* package used to implement the Stacked Regression model does not have an efficient method for customized tuning of the metalearner. Our recourse was to utilize the automatic tuning function in this package, but this only tuned *lambda*, leaving *alpha* at the default value. Furthermore, an ideal implementation of the Stacked Regression model would have allowed us to fully replicate the best tunings of our individual models. However, differences between *h2o* and the other model packages used inhibited a full replication of each base learner, forcing us to make heuristic decisions for some hyperparameters in the base learners.

The different sample sizes of our data sets in Phase 1 and Phase 2 can also be considered a limitation, although this choice was made consciously. Since the two phases utilize different data sets and cleaning procedures, a direct comparison of results across the phases is not possible. Related future analyses may want to utilize similar sample sizes for both phases in order to mitigate this. However, the decision to use a larger sample size and different pre-processing and feature engineering procedures in Phase 2 was motivated by a desire to fully utilize available data to investigate the models' potential accuracy levels.

Although mentioned in Section 6.1.1, we feel obliged to also note that the discussion of taxation effects from using machine learning models is a simplified estimation exercise intended only for indicative purposes. The relatively small sample size of our data is a

limitation in this estimation. Since the tax effects must be estimated using the test set, we are only using a random sample of 30% of our total data. This makes any year-by-year tax estimation difficult because each year would have a relatively small number of observations. Since both the training and test data are sampled randomly, we also run the risk of using more recent data to predict taxes back in time, which is not ideal for practical purposes. A more robust approach would look at the effects of our models on taxation by year, in addition to providing a generally more extensive and methodically sound analysis of property taxes.

The discussion of economic impacts as a whole is limited by the proprietary nature of business practices, with few businesses willing to freely share such information. Deeper analysis of each application and the potential economic benefits from better housing price predictions are therefore promising avenues for further research. This should preferably be done in collaboration with an industry partner such as a bank or a covered bond issuer.

In this thesis we have proved that there is potential to improve housing price predictions considerably by implementing any one of only four different non-linear machine learning models. Future research could benefit from investigating the performance of other machine learning models than the ones used here.

One particularly promising idea for research can be found by combining machine learning models with property image data. Property listings on platforms such as Finn.no usually include numerous images of the property in question. Training a specialized image classifier, such as a Convolutional Neural Network, on this image data in order to predict housing prices or expected deviations from appraised value can provide valuable insight into the aesthetics or types of images associated with higher or lower prices. Such a model could yield valuable predictions by itself, or be incorporated into a Stacked model along with models such as the ones trained in this paper.

7 Conclusion

In this thesis we have examined whether non-linear machine learning models can improve housing price predictions compared to linear regression models in the Norwegian residential housing market.

In the first phase of our analysis we replicated the data filtering process employed by SSB and compared the performance of our machine learning algorithms against the benchmark SSB model using the same variables and data that SSB uses. This was done to isolate the effect of switching to more advanced non-linear machine learning models. We found that the XGBoost model performed the best and increased out-of-sample prediction accuracy by 8.5% in terms of RMSE compared to SSB's linear model. Our results suggest that a modest increase in accuracy can be made by simply opting for non-linear machine learning algorithms over linear regression models due to the existence of non-linear feature relationships in the data.

In the second phase, we examined whether prediction accuracy could be further increased by training the same models on a data set with additional property-specific and macroeconomic variables. We found that the Stacked Regression model performed the best and was able to produce 39.52% more accurate predictions compared to the benchmark linear model in terms of RMSE. Moreover, all the non-linear models outperformed the linear benchmark. The results demonstrate that significant accuracy gains can be made by adopting a non-linear machine learning approach in conjunction with additional variables.

Increased housing price prediction accuracy from non-linear machine learning models can yield economic gains in applications including taxation, property transactions, refinancing and risk management. While the proprietary nature of business practices limits the extent of our estimations of economic gains, preliminary tax estimates indicate that adoption of our best machine learning model may have several beneficial effects that lead to more *correct* property taxes. This includes a 31.5% reduction in properties being unduly charged with property taxes, as well as a considerable reduction in underestimation of property taxes for the most valuable dwellings.

Future research should focus on deeper analyses of the economic effects and applications for improved housing price prediction. We also recommend further exploration of different

types of machine learning algorithms, especially in conjunction with other data types such as image data from property listings.

References

(2020). Xgboost documentation. https://xgboost.readthedocs.io/en/latest/index.html#.

Allaire, J. and Chollet, F. (2020). *keras: R Interface to 'Keras'*. R package version 2.3.0.0.

Allaire, J. and Tang, Y. (2020). *tensorflow: R Interface to 'TensorFlow'*. R package version 2.2.0.

Benedictow, A. and Walbækken, M. (2020). Kvantitativ analyse av boligomsetning og budgivning. https://www.regjeringen.no/contentassets/79704749185444af9cad00cb5f95c17c/samfunnsokonomisk-analyse.pdf.

Bergstra, J. and Bengio, Y. (2012). Random search for hyper-parameter optimization. *The Journal of Machine Learning Research*, 13:281–305.

Birch, J. W., Sunderman, M. A., and Hamilton, T. W. (1991). Estimating the importance of outliers in appraisal and sales data. *Property Tax Journal*, 10(4):361–376.

Birkeland, K. and D'Silva, A. D. (2018). Developing and evaluating an automated valuation model for residential real estate in oslo. Master's thesis, Norwegian University of Science and Technology (NTNU). Retrieved 2020-12-13, from https://ntnuopen.ntnu.no/ntnu-xmlui/bitstream/handle/11250/2575523/19153_FULLTEXT.pdf?sequence=1&isAllowed=y.

BIS (2014). Standards - supervisory framework for measuring and controlling large exposures. Bank for International Settlements. https://www.bis.org/publ/bcbs283.pdf.

Bloomberg (2020). Securities: Nibor3m index and nocpimom index. Retrieved 2020-10-27.

Boehmke, B. and Greenwell, B. M. (2019). *Hands-on machine learning with R*. CRC Press.

Breiman, L. (1996). Stacked regressions. *Machine learning*, 24(1):49–64.

Breiman, L. (2001). Random forests. *Machine learning*, 45(1):5–32.

Breiman, L., Cutler, A., Liaw, A., and Wiener, M. (2018). *randomForest: Breiman and Cutler's Random Forests for Classification and Regression*. R package version 4.6-14.

Chen, T. (2014). Introduction to boosted trees. https://web.njit.edu/~usman/courses/cs675_fall16/BoostedTree.pdf.

Chen, T., Benesty, M., He, T., and Tang, Y. (2018). Understand your dataset with xgboost. https://cran.r-project.org/web/packages/xgboost/vignettes/discoverYourData.html#special-note-what-about-random-forests.

Chen, T. and Guestrin, C. (2016). Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794.

Chen, T., He, T., Benesty, M., Khotilovich, V., Tang, Y., Cho, H., Chen, K., Mitchell, R., Cano, I., Zhou, T., Li, M., Xie, J., Lin, M., Geng, Y., and Li, Y. (2020). *xgboost: Extreme Gradient Boosting*. R package version 1.2.0.1.

Chen, X., Wei, L., and Xu, J. (2017). House price prediction using lstm. *arXiv preprint arXiv:1709.08432*.

Chollet, F. and Allaire, J. (2017). *Deep Learning with R*. Manning.

Datta, A., Sen, S., and Zick, Y. (2016). Algorithmic transparency via quantitative input influence: Theory and experiments with learning systems. In *2016 IEEE symposium on security and privacy (SP)*, pages 598–617. IEEE.

de Haan, J. and Diewert, W. (2011). Handbook on residential property price indexes. *Luxembourg: Eurostat*.

Do, A. Q. and Grudnitski, G. (1992). A neural network approach to residential property appraisal. *The Real Estate Appraiser*, 58(3):38–45.

Eikon (2020). Ice brent crudelcoc1. Retrieved 2020-10-27.

Finance Norway (2018). Norwegian covered bonds. https://www.finansnorge.no/en/covered-bonds/covered-bonds/.

Friedman, J. (1999). Stochastic gradient boosting. *Computational Statistics Data Analysis*, 38:367–378.

Friedman, J., Hastie, T., and Tibshirani, R. (2000). Additive logistic regression: a statistical view of boosting (with discussion and a rejoinder by the authors). *Ann. Statist.*, 28(2):337–407.

Friedman, J. H. (2001). Greedy function approximation: A gradient boosting machine. *Ann. Statist.*, 29(5):1189–1232.

Glorot, X. and Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In Teh, Y. W. and Titterington, M., editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 249–256, Chia Laguna Resort, Sardinia, Italy. JMLR Workshop and Conference Proceedings.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press. http://www.deeplearningbook.org.

Griliches, Z. (1971). *Price indexes and quality change: Studies in new methods of measurement*. Harvard University Press.

Hastie, T., Tibshirani, R., and Friedman, J. (2009). *The elements of statistical learning: data mining, inference, and prediction*. Springer Science & Business Media.

Hyndman, R. J. and Koehler, A. B. (2006). Another look at measures of forecast accuracy. *International journal of forecasting*, 22(4):679–688.

James, G., Witten, D., Hastie, T., and Tibshirani, R. (2013). *An introduction to statistical learning*, volume 112. Springer.

Jansrud, A. (2017). Bank: Rammebetingelser, regelverk og risikostyring. https://www.finansnorge.no/globalassets/presentasjoner/2017/bank-risikostyring-regelverk-og-rammebetingelser.pdf.

Kaggle (2012). Titanic - machine learning from disaster. Retrieved 2020-11-05, from https://www.kaggle.com/c/titanic/notebooks?competitionId=3136&searchQuery=random+forest.

Kaggle (2018). Credit card fraud detection. Retrieved 2020-12-13, from https://www.kaggle.com/mlg-ulb/creditcardfraud/notebooks?datasetId=310&searchQuery=isolation+forest.

Kaggle (2020a). House prices - advanced regression techniques. Retrieved 2020-11-05, from https://www.kaggle.com/c/house-prices-advanced-regression-techniques/notebooks?competitionId=5407&searchQuery=stacked.

Kaggle (2020b). House prices - advanced regression techniques. Retrieved 2020-11-05, from https://www.kaggle.com/c/house-prices-advanced-regression-techniques/notebooks?competitionId=5407&searchQuery=xgboost.

Kaggle (2020c). House prices - advanced regression techniques. Retrieved 2020-11-05, from https://www.kaggle.com/c/house-prices-advanced-regression-techniques/notebooks?competitionId=5407&searchQuery=neural+network.

Kaiser, J. (2014). Dealing with missing values in data. *Journal of systems integration*, 5(1):42–51.

Kang, H.-B. and Reichert, A. K. (1991). An empirical analysis of hedonic regression and grid-adjustment techniques in real estate appraisal. *Real Estate Economics*, 19(1):70–91.

Kantardzic, M. (2003). Data mining: Concepts, models, methods, and algorithms. *Technometrics*, 45(3):277.

Keras (2020a). Adadelta. https://keras.io/api/optimizers/adadelta/.

Keras (2020b). Rmsprop. https://keras.io/api/optimizers/rmsprop/.

Kohavi, R., John, G. H., et al. (1997). Wrappers for feature subset selection. *Artificial intelligence*, 97(1-2):273–324.

Kuhn, M. (2020). *caret: Classification and Regression Training*. R package version 6.0-86.

Kuhn, M. and Johnson, K. (2019). *Feature engineering and selection: A practical approach for predictive models*. CRC Press.

Kursa, M. B., Rudnicki, W. R., et al. (2010). Feature selection with the boruta package. *J Stat Softw*, 36(11):1–13.

Lancaster, K. J. (1966). A new approach to consumer theory. *Journal of political economy*, 74(2):132–157.

LeDell, E., Gill, N., Aiello, S., Fu, A., Candel, A., Click, C., Kraljevic, T., Nykodym, T., Aboyoun, P., Kurka, M., and Malohlava, M. (2020). *h2o: R Interface for the 'H2O' Scalable Machine Learning Platform*. R package version 3.32.0.1.

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

Limsombunchai, V. (2004). House price prediction: hedonic price model vs. artificial

neural network. In *New Zealand agricultural and resource economics society conference*, pages 25–26.

Liu, F. T., Ting, K. M., and Zhou, Z.-H. (2008). Isolation forest. In *2008 Eighth IEEE International Conference on Data Mining*, pages 413–422. IEEE.

Liu, Y. and Just, A. (2019). *SHAPforxgboost: SHAP Plots for 'XGBoost'*. R package version 0.0.3.

Lu, S., Li, Z., Qin, Z., Yang, X., and Goh, R. S. M. (2017). A hybrid regression technique for house prices prediction. In *2017 IEEE International Conference on Industrial Engineering and Engineering Management (IEEM)*, pages 319–323. IEEE.

Lund, A. F. (2020). Interview with anders f. lund, head of research at eiendomsverdi.

Lundberg, S. M., Erion, G., Chen, H., DeGrave, A., Prutkin, J. M., Nair, B., Katz, R., Himmelfarb, J., Bansal, N., and Lee, S.-I. (2019). Explainable ai for trees: From local explanations to global understanding. *arXiv preprint arXiv:1905.04610*.

Lundberg, S. M., Erion, G., Chen, H., DeGrave, A., Prutkin, J. M., Nair, B., Katz, R., Himmelfarb, J., Bansal, N., and Lee, S.-I. (2020). From local explanations to global understanding with explainable ai for trees. *Nature Machine Intelligence*, 2(1):2522–5839.

Lundberg, S. M., Erion, G. G., and Lee, S.-I. (2018). Consistent individualized feature attribution for tree ensembles. *arXiv preprint arXiv:1802.03888*.

Lundberg, S. M. and Lee, S.-I. (2017). A unified approach to interpreting model predictions. In *Advances in neural information processing systems*, pages 4765–4774.

Maniruzzaman, M., Rahman, M. J., Al-MehediHasan, M., Suri, H. S., Abedin, M. M., El-Baz, A., and Suri, J. S. (2018). Accurate diabetes risk stratification using machine learning: role of missing value and outliers. *Journal of medical systems*, 42(5):92.

Molnar, C. (2019). *Interpretable Machine Learning*. https://christophm.github.io/interpretable-ml-book/.

Mullainathan, S. and Spiess, J. (2017). Machine learning: an applied econometric approach. *Journal of Economic Perspectives*, 31(2):87–106.

Park, B. and 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.

Paul, R. K. (2006). Multicollinearity: Causes, effects and remedies. *IASRI, New Delhi*, 1(1):58–65.

Ribeiro, M. T., Singh, S., and Guestrin, C. (2016). " why should i trust you?" explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1135–1144.

Rosen, S. (1974). Hedonic prices and implicit markets: product differentiation in pure competition. *Journal of political economy*, 82(1):34–55.

Scriven, A. (2018). Introducing random search. https://www.datacamp.com/

courses/hyperparameter-tuning-in-python. https://s3.amazonaws.com/assets.datacamp.com/production/course_15167/slides/chapter3.pdf.

Shapley, L. S. (1953). A value for n-person games. *Contributions to the Theory of Games*, 2(28):307–317.

Shrikumar, A., Greenside, P., and Kundaje, A. (2017). Learning important features through propagating activation differences. *arXiv preprint arXiv:1704.02685*.

Skatteetaten (2020). Bolig, eiendom og tomt. https://www.skatteetaten.no/person/skatt/hjelp-til-riktig-skatt/bolig-og-eiendeler/bolig-eiendom-tomt/.

SSB (2018). Tax for personal tax payers: Wealth tax and taxable net property. Statistics Norway (SSB). Statbank. https://www.ssb.no/en/statbank/table/08231.

SSB (2020). 12843: Eiendomsskatt, etter region, statistikkvariabel og år. https://www.ssb.no/statbank/table/12843/.

Stock, J. H. and Watson, M. W. (2015). *Introduction to econometrics*.

Štrumbelj, E. and Kononenko, I. (2014). Explaining prediction models and individual predictions with feature contributions. *Knowledge and information systems*, 41(3):647–665.

Takle, M. and Melby, P. (2020). Modell for beregning av boligformue. Statistics Norway. https://www.ssb.no/priser-og-prisindekser/artikler-og-publikasjoner/modell-for-beregning-av-boligformue--415134.

Tay, D. P. and Ho, D. K. (1992). Artificial intelligence and the mass appraisal of residential apartments. *Journal of Property Valuation and Investment*.

Tieleman, T. and Hinton, G. (2012). Lecture 6.5—RmsProp: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural Networks for Machine Learning.

Troyanskaya, O., Cantor, M., Sherlock, G., Brown, P., Hastie, T., Tibshirani, R., Botstein, D., and Altman, R. B. (2001). Missing value estimation methods for dna microarrays. *Bioinformatics*, 17(6):520–525.

Truong, Q., Nguyen, M., Dang, H., and Mei, B. (2020). Housing price prediction via improved machine learning techniques. *Procedia Computer Science*, 174:433–442.

Van der Laan, M. J., Polley, E. C., and Hubbard, A. E. (2007). Super learner. *Statistical applications in genetics and molecular biology*, 6(1).

Wolpert, D. H. (1992). Stacked generalization. *Neural networks*, 5(2):241–259.

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

Zainuri, N. A., Jemain, A. A., and Muda, N. (2015). A comparison of various imputation methods for missing values in air quality data. *Sains Malaysiana*, 44(3):449–456.

Zeiler, M. D. (2012). ADADELTA: an adaptive learning rate method. *CoRR*, abs/1212.5701.

Zheng, A. and Casari, A. (2018). *Feature engineering for machine learning: principles and techniques for data scientists.* " O'Reilly Media, Inc.".

Appendix

A1 Variable Description and Imputation Methods

Table A1.1 presents an overview of all variables along with a description and imputation method for each variable in Phase 2. For features that only take one of two values, such as whether a particular estate type has a balcony (yes or no), we simply assume that missing values indicate that the particular estate type does not have a balcony. Hence, we set a default value of 0 for such variables (where 1 represents yes and 0 represents no). This also applies to features where missing values likely represent a value of 0, such as the value of the debt of housing associations with cooperative ownership. It is reasonable to assume that such debt is only reported for estate types that have such debt (such as apartment blocks), but not for other property types. Hence, it is reasonable to set a default value of 0 in case of missing values for these types of instances, as they are not missing at random.

Numerical features are imputed using grouped median or median, except for the square meter metrics. Since at least one square meter metric is available for all transactions, we use the different metrics as a proxy for each other, as it is reasonable to assume that this is a better proxy than the median value of all apartments. We impute missing values for primary area as gross area, and vice versa. Similarly, we impute missing values for gross area as total area, and vice versa. In cases where the value of a feature likely depends on the geographical location of a property, such as for altitude, coast distance and coast direction, we use city district as a grouping variable. The reasoning behind this is that these features are most likely very similar for properties in the same area, whereas there might be huge differences for properties that lie on different sides of the city. For features that likely depend on the specific type of property, such as the number of bedrooms, we use estate type as grouping variable. In cases where grouped median cannot be computed because all values for a given group are missing, we use the median value.

Table A1.1: Variable description and imputation methods

This table shows variable description amd imputation method for all variables.

Variable	Description	Type	Imputation method	Source
CityDistrict	Location (borough)	Dummy	None	Eiendomsverdi
PricePerSquareMeter	Sales price per square meter (NOK)	Continuous	None	Eiendomsverdi
TargetPriceCommondebt	Shared debt of housing cooperative (NOK)	Continuous	None	Eiendomsverdi
TargetPriceMarketSaleDate	Sales date	Date	None	Eiendomsverdi
EstateType	Type of property	Dummy	None	Eiendomsverdi
EstateSubType	Sub category for property type	Dummy	Default (EstateType)	Eiendomsverdi
OwnershipType	Type of ownership	Dummy	None	Eiendomsverdi
SiteOwnershipType	Site ownership type	Dummy	None	Eiendomsverdi
PRom	Square meters of primary area	Continuous	Default (BRA/BTA)	Eiendomsverdi
BRA	Square meters of gross area	Continuous	Default (Prom/BTA)	Eiendomsverdi
BTA	Square meters of gross total area	Continuous	Default (BRA/Prom)	Eiendomsverdi
BuildYear	Building build year	Continuous	Grouped median	Eiendomsverdi
NumberOfBedrooms	Number of bedrooms	Continuous	Grouped median	Eiendomsverdi
Balcony	Balcony (yes/no)	Dummy	Default (0)	Eiendomsverdi
Elevator	Elevator (yes/no)	Dummy	Default (0)	Eiendomsverdi
SiteArea	Total square meters of site	Continuous	Grouped median	Eiendomsverdi
SiteAreaUndeveloped	Undeveloped site area (square meters)	Continuous	Grouped median	Eiendomsverdi
Longitude	Longitude coordinates	Continuous	None	Eiendomsverdi
Latitude	Latitude coordinates	Continuous	None	Eiendomsverdi
CoastDistance	Distance to coast (meters)	Continuous	Grouped median	Eiendomsverdi
CoastDirection	Direction of coast (degree)	Continuous	Grouped median	Eiendomsverdi
Altitude	Altitude above sea level (meters)	Continuous	Grouped median	Eiendomsverdi
SunsetHour	Sunset time during summer	Continuous	Grouped median	Eiendomsverdi
SiteSlope	Slope of site (degrees)	Continuous	Grouped median	Eiendomsverdi
SiteSlopeDirection	Site slope direction (degrees)	Continuous	Grouped median	Eiendomsverdi
NumberOfUnitsInAdjoiningSquares	Number of housing units within one square of the property	Continuous	Median	Eiendomsverdi
NumberOfUnitsInAdjoiningSquaresX2	Number of housing units within two squares of the property	Continuous	Median	Eiendomsverdi
NumberOfDetachedHousesInAdjoiningSquares	Number of detached houses within one square	Continuous	Median	Eiendomsverdi
NumberOfDetachedHousesInAdjoiningSquaresX2	Number of detached houses within two squares	Continuous	Median	Eiendomsverdi
NumberOfFlatsInAdjoiningSquares	Number of flats within one square	Continuous	Median	Eiendomsverdi
NumberOfFlatsInAdjoiningSquaresX2	Number of flats within two squares	Continuous	Median	Eiendomsverdi
NumberOfAttachedHousesInAdjoiningSquares	Number of attached houses within one square	Continuous	Median	Eiendomsverdi
NumberOfAttachedHousesInAdjoiningSquaresX2	Number of attached houses within two square	Continuous	Median	Eiendomsverdi
NumberOfSemiDetachedHousesInAdjoiningSquares	Number of semi-detached houses within one square	Continuous	Median	Eiendomsverdi
NumberOfSemiDetachedHousesInAdjoiningSquaresX2	Number of semi-detached houses within two squares	Continuous	Median	Eiendomsverdi
PreviousValue	Market value at previous sale	Continuous	Default (0)	Eiendomsverdi
PreviousValueCommondebt	Value of debt at previous sale	Continuous	Default (0)	Eiendomsverdi
PreviousPriceValueCategory	Sales category of previous sale	Dummy	Default (No sale)	Eiendomsverdi
CPI	Norwegian Consumer Price Index	Continuous	None	Bloomberg
NIBOR	NIBOR 3-month	Continuous	None	Bloomberg
Brent	Norwegin Brent Spot	Continuous	None	Eikon
SalesYear	Year of sale	Continuous	None	Eiendomsverdi
GroundFloor	Ground floor (yes/no)	Dummy	None	Eiendomsverdi
MiddleFloor	Between ground floor and top floor (yes/no)	Dummy	Default (MiddleFloor)	Eiendomsverdi
TopFloor	Top floor (yes/no)	Dummy	None	Eiendomsverdi
Age	Age at time of sale	Continuous	None	Eiendomsverdi
SiteAreaUndevelopedPerc	Undeveloped site area as a percentage of total area	Continuous	None	Eiendomsverdi
SalesMonth	Month of sale	Continuous	None	Eiendomsverdi

A2 Anomaly Detection with Isolation Forest

Isolation Forest is an unsupervised learning algorithm that, unlike most model based anomaly detection procedures, identifies anomalies instead of identifying normal points (Liu et al., 2008). Because of this, Isolation Forests only needs to be trained on a small fraction of the data in order to effectively identify anomalies (Liu et al., 2008). Just like other Ensemble methods, it is based on the idea of building a large number of decision trees. In each of these decision trees, which are built using only a small subset of the data, branches are created by randomly selecting a feature. The data contained in each feature is then divided by randomly selecting a split value between the minimum and

maximum value of the feature (Liu et al., 2008). If a given observation has a lower value than the split value, it will follow the left branch in the tree. Otherwise, it will follow the right branch. The process of splitting data and creating new branches within a tree continues until a single point is isolated or a maximum depth is reached (Liu et al., 2008). The key idea is that since the split value is chosen randomly, all split values have equal probabilities. Then, since outliers in theory are few and lie further away from the normal observations in the feature space (which are usually clustered together), it takes on average significantly fewer splits to isolate an outlier as opposed to normal observations (Liu et al., 2008). Hence, outliers should lie closer to the root of the tree and therefore have a shorter average path length compared to normal observations. After all features have been considered, an anomaly score is computed for each unique observation in the data. Equation .1 shows the formula used to compute the anomaly score.

$$s(x,n) = 2^{-\frac{E(h(x))}{c(n)}} \tag{.1}$$

In this setting, $h(x)$ is the path length of observation x, $E(h(x))$ is the expected or average path length of observation x, $c(n)$ is the average path length of unsuccessful search in a binary search tree and n is the number of external nodes in a tree (Liu et al., 2008). A score closer to 1 indicates an anomaly, whereas a score closer to 0.5 indicates normal observations.

A3 Descriptive Statistics

Table A3.1: Descriptive statistics for all continuous variables in Phase 2

Variable	Mean	St. Dev.	Min	Max
Price per square meter (NOK)	56,535.940	21,358.970	1,209.677	226,851.900
Shared debt in joint ownership (NOK)	121,240.600	258,578.800	0	5,201,000
Primary area (m^2)	62.803	27.597	10	386
Gross area (m^2)	63.424	28.018	10	401
Gross total area (m^2)	68.082	30.925	10	386
Build year	1,940.937	38.362	1,750	2,020
Number of bedrooms	1.685	0.729	0	23
Site area (m^2)	3,671.846	7,499.272	0.000	61,435.600
Undeveloped site area (m^2)	2,741.128	6,910.578	0.200	49,273.600
Longitude	262,251.300	1,721.930	255,603	265,311
Latitude	6,651,022.000	827.326	6,648,000	6,653,021
Distance to coast (meters)	2,120.448	1,002.593	5	10,000
Coast direction (degrees)	222.139	21.468	29	375
Site slope (degrees)	3.646	3.148	0	37
Site slope direction (degrees)	199.281	60.740	0	351
Number of units in adjoining squares	4,146.666	1,141.798	258	6,653
Number of units in adjoining squares (X2)	10,712.780	2,749.728	1,482	15,781
Number of detached houses in adjoining squares	27.457	31.834	0	316
Number of detached houses in adjoining squares (X2)	85.849	80.033	3	700
Number of flats in adjoining squares	3,883.720	1,119.208	38	6,270
Number of flats in adjoining squares (X2)	9,956.705	2,718.613	328	14,692
Number of attached houses in adjoining squares	17.577	23.578	0	255
Number of attached houses in adjoining squares (X2)	47.565	52.924	0	524
Number of semi detached houses in adjoining squares	23.102	19.465	0	171
Number of semi detached houses in adjoining squares (X2)	76.471	46.718	3	384
Previous value (NOK)	755,361.100	1,329,970.000	0	14,400,000
Previous value shared debt (NOK)	35,597.810	149,869.500	0	5,201,000
Consumer price index	96.938	9.085	81.200	112.900
NIBOR 3-month (%)	2.174	1.359	0.230	7.910
Brent Spot (USD)	75.567	25.186	19.330	146.080
Sunset time during summer	20.651	0.426	17.250	21.390
Altitude (meters)	51.277	27.509	0	190
Sales year	2,012.807	4.497	2,005	2,020
Age	71.870	38.272	0	256
Undeveloped site area (%)	1.658	15.160	0.001	1,341.070

Table A3.2: Descriptive statistics for all dummy variables in Phase 2

Variable	Mean
Balcony	0.668
Elevator	0.389
Ground floor	0.173
Middle floor	0.768
Top floor	0.059
January	0.077
February	0.076
March	0.090
April	0.089
May	0.107
June	0.110
July	0.042
August	0.113
September	0.100
October	0.091
November	0.075
December	0.030
Frogner	0.239
Grünerløkka	0.277
Sagene	0.257
St. Hanshaugen	0.171
Ullern	0.056
Apartment	0.478
Apartment in block	0.512
Other	0.010
Stock apartment	0.108
Housing cooperative	0.370
Cooperative apartment	0.00002
Condominium	0.522
Leasehold site	0.116
Site owner	0.884
Market sale	0.295
No previous sale	0.705

A4 The Boruta Algorithm

The Boruta algorithm utilizes the original Random Forest algorithm to identify the relevance of attributes (i.e features) by comparing their relevance to random probes (Kursa et al., 2010). The algorithm works by creating a shadow attribute for each attribute in the data, where the values of a given shadow attribute is obtained by randomly shuffling

the original value of the attribute across objects (Kursa et al., 2010). The algorithm then leverages the Random Forest algorithm to classify the importance of the shadow attributes. The idea is that given the random nature of the shadow attributes, their importance can only be nonzero due to random fluctuations (Kursa et al., 2010). Hence, the importance of the shadow attributes can be used as a reference to evaluate the importance of the real attributes. After all attributes have been classified, a Z-score, as shown in Equation .2, is computed for all decision trees to determine the relevance of an attribute.

$$Z = \frac{Average\ loss}{Standard\ deviation\ of\ variable} \tag{.2}$$

The algorithm then finds the maximum Z-score amongst the shadow attributes. If an attribute has significantly lower score than the maximum Z-score, it is deemed unimportant and permanently removed (Kursa et al., 2010). Otherwise, it is deemed important. At the end of the iteration, all shadow attributes are removed. This process is then repeated until importance is assigned to all attributes, or until the algorithm reaches the predetermined maximum number of runs (Kursa et al., 2010). If the importance of an attribute cannot be deemed within the maximum number of runs, importance can be assigned by comparing the median attribute Z-score against the median Z-score of the most important attribute (Kursa et al., 2010).

A5 Feature Selection with the Boruta Algorithm

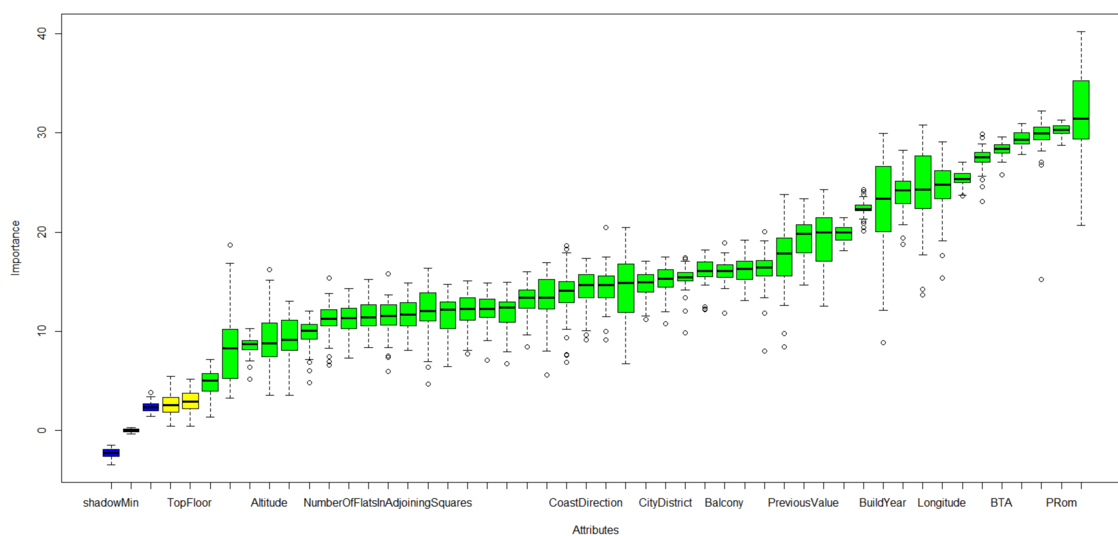
We use the *Boruta* library in R (Kursa et al., 2010) to perform feature selection using the Boruta algorithm in Phase 2. All features are included in the model, and price per square meter is declared as the response. The algorithm is performed on 50 percent of the data to speed up computation time. Figure A5.1 shows a graph of all features and their corresponding importance. Unsurprisingly, square meter metrics such as PRom and BTA are considered highly relevant. After 50 iterations, all features are confirmed important except TopFloor and SalesMonth, which are marked as tentative. A possible explanation for this is that the features had Z-scores very close to the maximum Z-score, rendering the algorithm unable to make a decision with the desired statistical confidence within a realistic number of iterations. However, after performing a *tentative fix* on the remaining

features in which the median Z-score of the feature is compared against the median Z-score of the most important shadow attribute, both tentative features are confirmed as important.

Hence, the results from the Boruta algorithm suggest that all features should be included for predictive purposes. A possible concern in this regard is multicollinearity, a situation in which two or more predictors are strongly correlated, meaning that they explain a lot of the same variation in the response variable (Paul, 2006; Stock and Watson, 2015). The main implication of multicollinearity is that statistical inference becomes unreliable, as it gets harder for the models to isolate the relationship between the response variable and each of the predictors (Stock and Watson, 2015; Paul, 2006). However, since we are only interested in prediction and not inference, multicollinearity will not be a problem (Paul, 2006). It is also worth noting that because the Boruta algorithm uses the Random Forest classifier, it is not affected by problems with multicollinearity in the fitting process (Kursa et al., 2010; Chen et al., 2018).

Figure A5.1: Feature selection using the Boruta algorithm

This figure shows a plot of the features and their corresponding relevance according to the Boruta algorithm. The horizontal axis maps the features and the vertical axis displays their corresponding importance. Blue boxplots convey the minimal, average and maximum Z-score of an attribute, while red and green boxplots represent the same information for the actual features. The higher the feature is located in the two-dimensional space, the more important it is considered to be.



A6 SHapley Additive exPlanations

SHapely Additive exPlanations (SHAP) are based on the concept of Shapley Values, a solution in cooperative game theory that assigns payouts to the players in a coalition depending on their marginal contributions (Shapley, 1953). In the context of machine learning Shapley values can be used to explain the output of models by assuming a game where each feature value is a player and the prediction is the output (Molnar, 2019). As in the original case, Shapley values can then be used to determine how to fairly distribute the payoff among the features (i.e. players) (Shrikumar et al., 2017).

In order to understand how SHAP values are computed, we must first understand how Shapley values are computed in the context of interpreting the output of machine learning models. The Shapley Value for *one* feature is the average marginal contribution of a feature value across all possible coalitions of features (Molnar, 2019). Mathematically, the Shapley value can be interpreted as the contribution ϕ_j of the jth feature on the prediction of a particular instance (Molnar, 2019). The Shapley value is defined by a value function val consisting of a subset S of players, where the Shapley value of feature j is the average contribution of j to the payout across all possible feature value combinations (Molnar, 2019):

$$\phi_j(val) = \sum_{S \subseteq \{x_1,\ldots,x_p\}/\{x_j\}} \frac{|S|!(p-|S|-1)!}{p!}(val(S \cup \{x_j\}) - val(S)) \tag{.3}$$

Where S is a subset of the features, x is a vector of feature values of the particular instance that we seek to explain and p is the number of features (Molnar, 2019). Then, $val_x(S)$ is the prediction for the feature values contained in set S that are marginalized over the features that are not included in this set (Molnar, 2019):

$$val_x(S) = \int \hat{f}(x_1, ..., x_p) d\mathbb{P}_{x \notin S} - E_X(\hat{f}(X)) \tag{.4}$$

It can be shown that the cooperative solution represented by the Shapley value is the only solution that satisfies the properties required for a fair distribution of the payout, namely *Efficiency*, *Symmetry*, *Dummy* and *Additivity*. We refer to Shapley (1953) for

details regarding these properties.

To estimate the Shapley Value of feature j, we must evaluate all possible coalitions (i.e. sets) of feature values with and without the jth feature. The exact estimation of the Shapley value is what separates the original solution by Shapley (1953) from the ones proposed by Lundberg and Lee (2017). In their original paper, Lundberg and Lee (2017) proved that SHAP values satisfy the same properties as the Shapely Value[79].

In the SHAP framework, the Shapley value is represented as an additive feature attribution method, meaning that it is a linear function of binary variables (Lundberg et al., 2018). This allows for connecting Local Interpretable Model-agnostic Explanations (LIME), which is a local surrogate model, and Shapely values. The goal is to explain the prediction of an instance x by quantifying how much each feature contributes to the prediction (Molnar, 2019; Lundberg and Lee, 2017), which is specified as:

$$g(z') = \phi_0 + \sum_{j=1}^{M} \phi_j z'_j \tag{.5}$$

Where g is the linear explanation model, $z' \in \{0,1\}^M$ is the simplified features or the "coalition vector", M is the number of input features (i.e maximum coalition size) and $\phi_j \in \mathbb{R}$ is the feature attribution for feature j (i.e the Shapley value) (Molnar, 2019; Lundberg et al., 2018). In the coalition vector, a value of 1 means that the feature value is present ($z'_j = 1$) and a value of 0 means that it is absent ($z'_j = 0$) (Molnar, 2019; Lundberg et al., 2018). Simulation is then used to dictate which features are present (i.e. playing) and which are absent (i.e. not playing) (Molnar, 2019). For a particular instance x, the coalition vector x' is a vector containing only 1's[80], which simplifies to:

$$g(x') = \phi_0 + \sum_{j=1}^{M} \phi_j \tag{.6}$$

Lundberg and Lee (2017) show that ϕ satisfy the properties Local accuracy, Missingness and Consistency, which correspond to the properties that the Shapely value satisfy. Hence, SHAP values are consistent with Shapley values.

[79] Referred to as Local accuracy, Missingness and Consistency in Lundberg and Lee (2017)
[80] Meaning that all feature values are present

Based on the original SHAP framework, the SHAP authors proposed TreeSHAP (Lundberg et al., 2018, 2019), which is what we use in this thesis. TreeSHAP is a model-specific SHAP framework for ensemble methods (Lundberg et al., 2018). TreeSHAP differs from the original SHAP framework in that it uses the conditional expectation $E_{X_S|X_C}(f(x)|x_S)$ to define the value function instead of the marginal expectation (Molnar, 2019). In short, TreeSHAP computes the Shapley values of an ensemble of decision trees as the weighted average of the Shapley values of the individual decision trees (Molnar, 2019). The computation of Shapley values for a single decision tree can be extended to a tree ensemble due to the Additivity property of Shapley values.

In order to evaluate the exact effect that missing (i.e. absent) features has on a model f, a mapping function h_x must be defined (Lundberg et al., 2018). According to Lundberg et al. (2018), h_x maps between a binary pattern of missing features represented by z' (the coalition vector) and the original function. This allows for evaluating $f(h_x(z'))$ and thereby compute the effect of a feature being present ($z'_j = 1$) or absent ($z'_j = 0$) (Lundberg et al., 2018).

To compute SHAP values for a model f, Lundberg et al. (2018) define

$$f_x(S) = f(h_x(z')) = E[f(x)|x_S] \tag{.7}$$

Where S is a set of non-zero indexes in the coalition vector z' and $E[f(x)|x_S]$ is the expected value of the function conditioned on the subset S containing the input features (Lundberg et al., 2018). SHAP values are then computed by combining the conditional expectations with the original Shapley values from Equation .3:

$$\phi_i = \sum_{S \subseteq N/\{i\}} \frac{|S|!(M - |S| - 1)!}{M!} [f_x(S \cup \{i\}) - f_x(S)] \tag{.8}$$

Where N is the set of all features. Hence, to compute SHAP values for a tree model, one must estimate $E[f(x)|x_S]$ and then use Equation .8 where $f_X(S) = E[f(x)|x_S]$. The exact algorithms for estimating $E[f(x)|x_S]$ can be found in Lundberg et al. (2018)[81].

[81]Referred to as Algorithm 1 (exponential time) and Algorithm 2 (polynomial time) in the paper.

A6.1 SHAP Plots

In Chapter 5 we utilize SHAP Summary plots and SHAP Dependence plots. SHAP Summary plots show global feature importance and feature effects, sorted by importance. Feature importance is based on SHAP values, which in turn are based on Shapley values. Features with large absolute SHAP values are deemed important, and vice versa (Lundberg et al., 2019). Global importance is found by computing the average absolute SHAP value per feature across the data, as shown in Equation .9.

$$I_j = \sum_{i=1}^{n} |\phi_j^{(i)}| \tag{.9}$$

In SHAP Summary plots, each point along the horizontal axis represents a SHAP value $\phi_j^{(i)}$ for a given instance for a feature (Molnar, 2019). The positioning on the horizontal axis shows in which direction a feature affects the response variable and the Shapley value for each instance of a feature (Molnar, 2019).

In SHAP Dependence plots, a feature is plotted against the SHAP values of that feature for each data instance (Molnar, 2019). The feature value is shown on the horizontal axis and the SHAP value is shown on the vertical axis. Mathematically, the plot contains the points $\{(x_j^{(i)}, \phi_j^{(i)})\}_{i=1}^{n}$.

A7 Random Forest Hyperparameter Tuning Grids

Table A7.1: Random Forest Hyperparameter Tuning Values - Phase 1

Hyperparameter	Value
Ntree	1000
Mtry	(1, 2, 3, 4, 5, 6, 7, 8)

Table A7.2: Random Forest Hyperparameter Tuning Values - Phase 2

Hyperparameter	Value
Ntree	500
Mtry	20

A8 XGBoost Hyperparameter Tuning Values

Table A8.1: XGBoost Hyperparameter Tuning Values - Phase 1

Hyperparameter	Value
Number of trees	(50, 100, 300, 500, 700, 1000, 1300, 1500)
Maximum Tree Depth	(2, 3, 6, 8, 10, 15, 20, 25, 30)
Learning Rate	(0.01, 0.02, 0.03, 0.1, 0.3, 0.6, 1)
Gamma	(0, 0.001, 0.01, 0.1, 0.2, 0.3, 0.4, 0.5)
Minumum Child Weight	(1, 3, 5, 10, 15)
Column Subsampling	(0.500, 0.625, 0.750, 0.875, 1.000)
Row Subsampling	(0.500, 0.625, 0.750, 0.875, 1.000)

Table A8.2: XGBoost Hyperparameter Tuning Values - Phase 2

Hyperparameter	Value
Number of trees	(50, 100, 300, 500, 700, 1000, 1300, 1500)
Maximum Tree Depth	(2, 3, 6, 8, 10, 15, 20, 25)
Learning Rate	(0.01, 0.02, 0.03, 0.1, 0.3, 0.6, 1)
Gamma	(0, 0.001, 0.01, 0.1, 0.2, 0.3, 0.4, 0.5)
Minumum Child Weight	(1, 3, 5, 10, 15)
Column Subsampling	(0.500, 0.625, 0.750, 0.875, 1.000)
Row Subsampling	(0.500, 0.625, 0.750, 0.875, 1.000)

A9 DFNN Hyperparameter Tuning Values

Table A9.1: DFNN Hyperparameter Tuning Values - Phase 1 & 2

Hyperparameter	Value
Batch size	(16, 32, 64, 128, 256, 512, 1024)
Layer 1 hidden units	(16, 32, 64, 128, 256, 512, 1024)
Layer 2 hidden units	(16, 32, 64, 128, 256, 512, 1024)
Layer 1 dropout rate	(0.0, 0.1, 0.2, 0.3, 0.4, 0.5)
Layer 2 dropout rate	(0.0, 0.1, 0.2, 0.3, 0.4, 0.5)
Layer 1 bias initializer	(0, 0.01)
Layer 2 bias initializer	(0, 0.01)
Initial LR*	(0.0001, 0.0005, 0.001, 0.005, 0.01)
LR* annealing factor	(0.01000, 0.03375, 0.05750, 0.08125, 0.10500, 0.12875, 0.15250, 0.17625, 0.20000)
L2 regularization	(0, 0.0001, 0.001, 0.01, 0.1)
Optimizer	(SGD, RMSprop, Adadelta)
Momentum	(0, 0.5, 0.9)
Preset Values	**Value**
Hidden layers	2
LR* annealing patience	5
Activation function	Rectified Linear Unit (ReLU)
Max epochs	1000
Early stopping patience	20
Layer 1 weight initializer	Glorot Uniform
Layer 2 weight initializer	Glorot Uniform
Include bias layer 1	True
Include bias layer 2	True

*Learning Rate

A10 Predicted Versus Actual Values

Figure A10.1: Predicted versus actual values - Phase 1

This figure shows predicted versus actual values for the SSB Linear Regression (OLS) (top) and XGBoost (bottom). Actual values in the test data are displayed on the vertical axis and the predicted values are displayed along the horizontal axis. The blue diagonal line represents 100 % accuracy.

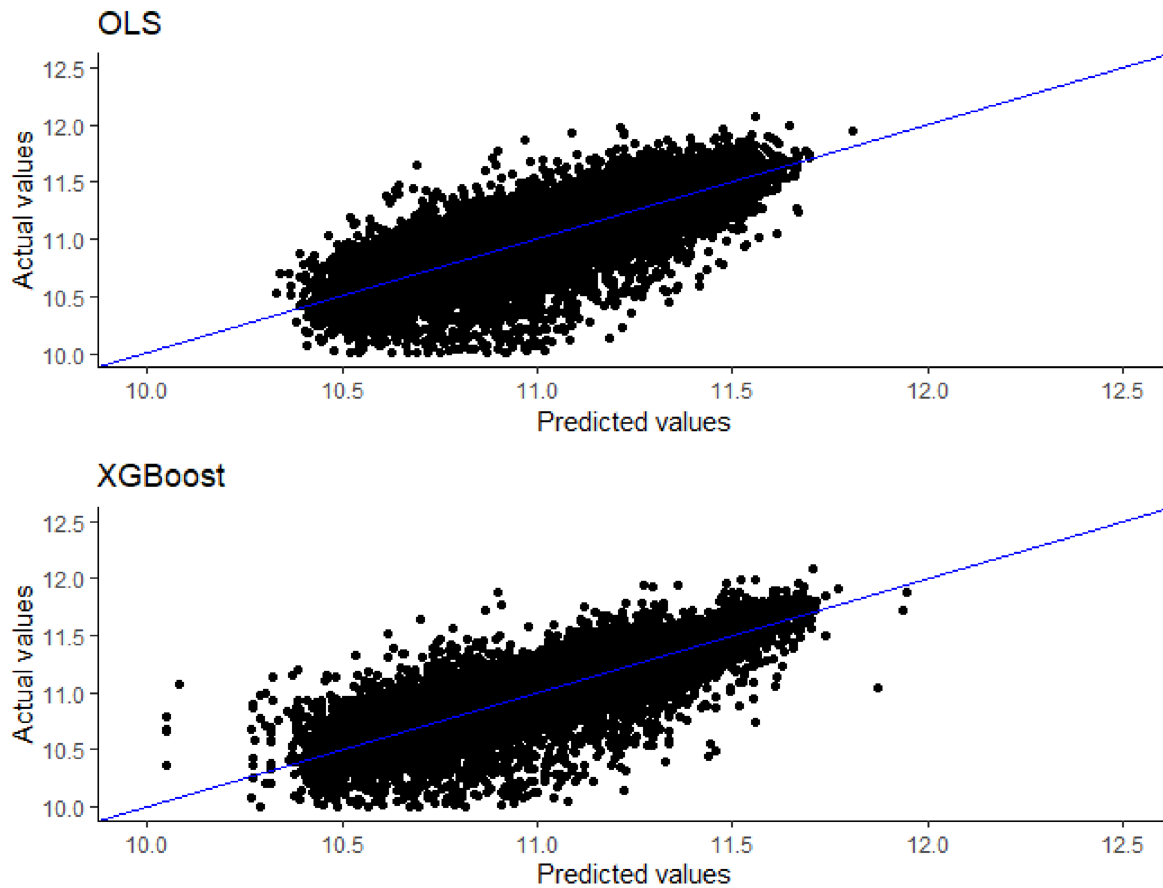
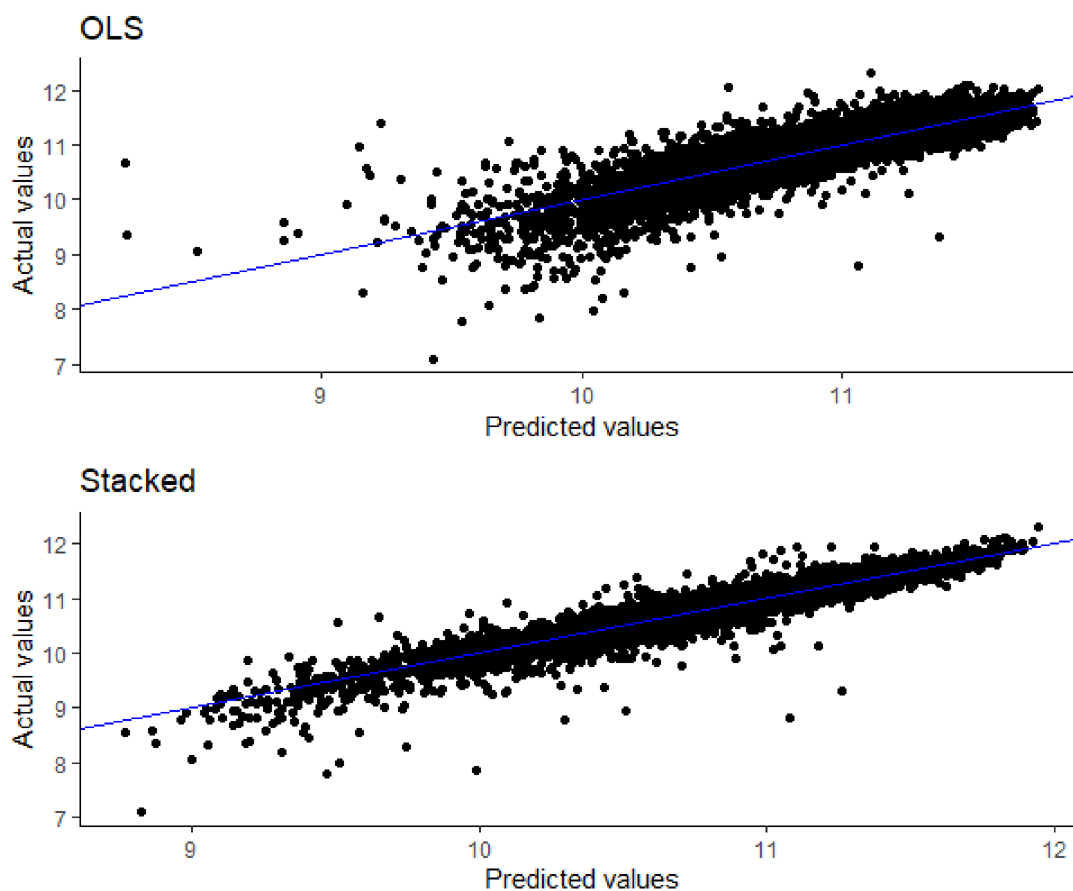


Figure A10.2: Predicted versus actual values - Phase 2

This figure shows predicted versus actual values for the linear benchmark model (OLS) (top) and Stacked model (bottom). Actual values in the test data are displayed on the vertical axis and the predicted values are displayed along the horizontal axis. The blue diagonal line represents 100 % accuracy.



A11 Phase 1 Model Selection

As described in the methodology chapter, hyperparameter tuning is conducted by means of Grid Search or Random Search for the individual models. Hyperparameter values for the base learners that make up the Stacked model are obtained from the individual tuning of these models. In cases where perfect replication of the individual models was not possible in the base learners, other similar values and approaches have been selected.

A11.1 XGBoost

Hyperparameter tuning for XGBoost was conducted via random search on a pre-defined grid, sampling 120 different combinations of hyperparameter values. The specifications of

the resulting model are shown in Table A11.1.

Table A11.1: XGBoost final hyperparameters

Hyperparameter	Value
Number of trees	1000
Maximum Tree Depth	10
Learning Rate	0.01
Gamma	0.2
Minumum Child Weight	5
Column Subsampling	1
Row Subsampling	0.875

The hyperparameter values provide insight into this model's learning process and the balancing between regularization and data fitting. The high number of trees grown makes the model, in isolation, prone to overfitting, since each additional tree captures ever more of the variance in the fitted data set. However, this is offset by the selection of a low learning rate. The net effect is that, although the model grows many trees, each tree's influence is relatively low, such that the overall model learns much of the relationships in the data set while still generalizing well.

With a maximum tree depth of 10, each tree is relatively deep, thereby allowing for more complexity to be captured than a shallower tree. This increases the risk of overfitting. However, non-zero values of Gamma and Minimum Child Weight act as regularizing factors to counteract this complexity. Similarly we see regularization effects from row subsampling only allowing each tree to sample a subset of the data when training.

A11.2 Random Forest

Grid Search was implemented to tune hyperparameters for the Random Forest algorithm. The resulting best model can be seen in Table A11.2. As mentioned in our theory section, Random Forest is robust to overfitting from the number of trees grown. As as long as the number of trees is set sufficiently large, the only hyperparameter that must be tuned is the number of predictor variables to consider at each node split, known as *mtry*. Our Phase 1 model includes only four variables, but due to one-hot encoding, the dataset encompasses nine columns. The tuning grid consisted of eight models with 1000 trees

each, and mtry ranging from one to eight [82].

Table A11.2: Random Forest final hyperparameters

Hyperparameter	Value
mtry	4
ntree	1000

Allowing the algorithm to only sample four out of nine predictors at each split forces the model to sometimes grow trees that consider the weaker predictors in the data set. This is very much in line with the essential principles behind Random Forest described in our theory section, thereby decorrelating the trees and reducing variance. Furthermore, this number is also closely aligned with the recommended number of predictors to sample in a Random Forest regression, which Breiman et al. (2018) suggest should be set to $p/3$ [83]. In our case this corresponds to three predictors being sampled at each split.

A11.3 Deep Feedforward Neural Network

As with XGBoost, the DFNN was tuned via random hyperparameter search sampling 120 unique combinations over a pre-defined set of values for each hyperparameter. The resulting model configuration can be seen in Table A11.3. The full hyperparameter grid sampled is found in Table A9.1.

[82] The upper limit of eight predictors was set to ensure the algorithm's randomness.
[83] p is the total number of predictors available.

Table A11.3: Deep Feedforward Neural Network Tuned Hyperparameters

Hyperparameter	Value
Number of epochs completed	92
Batch size	16
Layer 1 hidden units	256
Layer 2 hidden units	512
Layer 1 dropout rate	0
Layer 2 dropout rate	0.4
Layer 1 bias initializer	0
Layer 2 bias initializer	0.01
Initial learning rate	0.005
Learning rate annealing factor	0.0812
L2 regularization (Weight Decay)	0.0001
Optimizer	Minibatch Stochastic Gradient Descent (SGD)
Momentum	0.9
Preset Values	**Value**
Hidden layers	2
Learning rate annealing patience	5
Activation function	Rectified Linear Unit (ReLU)
Max epochs	1000
Early stopping patience	20
Layer 1 weight initializer	Glorot Uniform
Layer 2 weight initializer	Glorot Uniform
Include bias layer 1	True
Include bias layer 2	True

The model implements SGD with momentum as the optimizer, which, combined with a relatively large initial learning rate, allows the model to learn and converge faster. This is particularly useful in the early phase of training when large improvements can be made quickly. However, the model compensates for the large initial learning rate by including a relatively large annealing factor, which reduces the learning rate by a factor of 8.12 when loss plateaus, ensuring improved learning can be made in smaller increments. Although SGD is the non-adaptive optimizer in the tuning grid, our implementation of learning rate annealing when loss plateaus ensures that the learning rate in fact does change as training proceeds. Although the allowed number of epochs was 1000 iterations, the final model returned only completed 92 epochs before early stopping halted further learning. The relaitvely low number of epochs reduces the model's potential complexity. However, the small batch size of 16 is in general better for gaining a good fit to the data at the expense of increased complexity and poorer generalization. Furthermore, the large number of

hidden units in each layer also adds to the complexity of the model, allowing it to find more patterns in the data. However, further regularization of the model is done through the introduction of a large dropout rate in Layer 2 and L2 regularization that penalizes complexity in the weights between nodes.

A11.4 Linear Regression

The final Phase 1 linear regression model is shown in Table A11.4. The results indicate that all predictors are associated with the response variable. It is important to stress that we cannot use the estimated model to infer relationships between the variables or interpret the coefficients explicitly, as the least squares assumptions may not be satisfied. This implies that the coefficients and corresponding p-values may not be accurate. Although OLS diagnostics can be performed, it is outside the scope of this thesis. That being said, the estimated coefficients seem to be consistent with the findings from the SHAP plot, suggesting that there is a negative relationship between the price and square meters, and that the apartments located at Sagene, Grünerløkka, St. Hanshaugen and Ullern are associated with lower prices relative to Frogner.

Table A11.4: Linear Regression Coefficients

	Estimate
Primary area (m^2)	-0.190^{***}
Age (0-10)	0.098^{***}
Age (10-19)	0.085^{***}
Age (20-34)	0.063^{***}
Sagene	-0.208^{***}
Grünerløkka	-0.262^{***}
St. Hanshaugen	-0.128^{***}
Ullern	-0.143^{***}
Year	0.080^{***}
Constant	-149.789^{***}
Observations	55,929
Note:	*p<0.1; **p<0.05; ***p<0.01

A11.5 Stacked Regression

We use the R library *h2o* by LeDell et al. (2020) to implement a Stacked regression model, as this allows us to stack models that we have already tuned. The hyperparameters for the

base learners[84] are therefore the same as the ones obtained during tuning for these models, apart from a few instances where incompatibilities between R packages prohibit us from implementing the exact same configuration. A regularized linear regression is used as the metalearner to mitigate problems related to overfitting and highly correlated base learners. Regularization is introduced through incorporation of *alpha* and *lambda*-parameters for the metalearner. Hyperparameter tuning is only conducted for lambda, through the package's automatic *Lambda Search* functionality. The lack of tuning for alpha is a limitation in the analysis. The optimal weights for the base learners is obtained through k-fold CV using $k = 5$.

The final regularization parameters and weights for the Stacked model are shown in Table A11.5[85]. An alpha value of 0.5 implies that the distribution of the penalty terms from the Lasso and Ridge regression should be equally weighted. In the context of stacking, this allows for shrinking the weights of highly correlated base learners (Ridge) as well as completely shutting off base learners (Lasso). The lambda parameter controls the extent to which regularization should be performed. A final value of 0.0035 indicates that the penalty terms should not be heavily emphasized.

The final regularization parameters have direct implications for the final weights. The XGBoost and DFNN models, which have the best and second best CV-performance amongst the base learners, also receive the largest relative weights. However, the individual performance of the base learners does not exclusively determine the weights. The RF model, which has the worst performance of the base learners, is weighted more aggressively than the linear model. This is likely due to regularization, or because Random Forest captures some particular feature relationships better than the other models even though the overall performance is worse. Finally, the linear regression is marginally weighted in the Stacked model. This may suggest that linear regression does not contribute enough to variance reduction, and likely that it does not capture much new information vis-a-vis the other base learners. It also demonstrates that even with a small value for lambda, the metalearner will still largely disregard base learners that lead to deteriorating performance.

[84] XGBoost, RF, DFNN and Linear Regression.

[85] A full overview of the Stacked model and the configuration of the base learners can be seen in Table A11.6

Table A11.5: Stacked Regression Phase 1 - Final regularization parameters and weights

This table shows the final configuration of the Stacked Regression model for Phase 1. All base learners were trained using optimal values found during tuning of the individual models. Where we were unable to implement those values, similar or default values and approaches were chosen. Alpha was fixed at default value, while lambda was tuned through the package's automatic lambda search tuning function.
*Cross-validation error

Parameter/Weight	Value	Test RMSE	CV RMSE*
α	0.5		
λ	0.00351		
XGBoost	0.656567	0.1861	0.1892
FNN	0.22709	0.1878	0.1909
RF	0.11609	0.2038	0.2089
Linear Regression	0.002917	0.2022	0.2046
Intercept	-0.02868		

Table A11.6: Stacked model configuration - Phase 1

Stacked Model Configuration			
Stacked Ensemble			
Hyperparameter		**Value**	
Metalearner		Regularized Linear Model	
Alpha		0.5	
Lambda		0.000534	
Lambda Search		True	
Lambda max		Default	
Lambda min		Default	
Family		Gaussian	
CV-folds		5	
CV-folds per base learner		5	
CV-fold assignment		Modulo	
Base Learners			
XGBoost		**DFNN**	
Hyperparameter	**Value**	**Hyperparameter**	**Value**
Booster	gbtree	Epochs completed	92
Number of Trees	1000	Batch size	16
Max Tree Depth	10	Layer 1 Hidden Units	256
Learning Rate	0.01	Layer 2 Hidden Units	512
Gamma	0.2	Layer 1 Dropout rate	0
Minimum Child Weight	5	Layer 2 Droupout rate	0.4
Column Subsampling	1	Layer 1 Bias Initializer*	NULL
Row Subsampling	0.875	Layer 2 Bias Initializer*	NULL
		Initial Learning rate	0.005
		Learning rate annealing*	1e-06
		L2 Regularization (Weight Decay)	0.0001
		Optimizer	SGD
		Momentum start*	0.9
		Momentum stable*	0.9
		Hidden Layers	2
		Learning rate annealing patience*	0
		Activation function	ReLu
		Max Epochs	92
		Early stopping patience	20
		Layer 1 Weight initializer*	UniformAdaptive
		Layer 2 Weight initializer*	UniformAdaptive
		Include bias layer 1	True
		Include bias layer 2	True
		Loss*	Quadratic
Random Forest		**Linear Regression**	
Hyperparameter	**Value**	**Hyperparameter**	**Value**
Number of trees	1000	Alpha	0
Number of predictors sampled	4	Lambda	0
		Solver	IRLSM
		Family	Gaussian

*Values deviate from individual tuned models.

A12 Phase 2 Model Selection

A12.1 XGBoost

The optimal Phase 2 configuration for XGBoost resulting from our hyperparameter tuning process can bee seen in Table A12.1. A natural assumption is that the optimal configuration of the model changes when new variables are added to the data set, and the results do indeed show that different values for regularizing parameters are employed. While the number of trees, maximum depth, and row subsampling retain the same values as in Phase 1, the learning rate has increased to 0.02. This allows each sequential tree to retain more information than in Phase 1. This seems natural considering that the expanded data set used in Phase 2 likely also exhibits more relationships between features. Furthermore, gamma, controlling how much loss is required to partition a leaf node, has been reduced to zero, thereby increasing the potential complexity of the model. However, this is likely offset by a higher minimum child weight, this time requiring at least 10 instances to continue tree building. Column subsampling also plays a major role in regularization. Roughly two thirds of the predictor variables are sampled for each tree grown, thus reducing the risk of overfitting and decorrelating the model's trees. Although further tuning of the model is out of this paper's scope, the discrepancy between in-sample and out-of-sample errors show that improvements in generalization may be made by adding more regularization to this configuration.

Table A12.1: XGBoost final hyperparameters - Phase 2

Hyperparameter	Value
Number of trees	1000
Maximum Tree Depth	10
Learning Rate	0.02
Gamma	0
Minumum Child Weight	10
Column Subsampling	0.625
Row Subsampling	0.875

A12.2 Random Forest

Our preferred approach for model selection for the Random Forest algorithm was to conduct a full or limited grid search of possible *mtry* values. However, due to the algorithm's extremely long model fitting time, any meaningful implementation of a hyperparameter search was impossible with our available computational power, budget and time frame. We therefore fitted the final Random Forest model for Phase 2 using the default recommendation for mtry value[86], as well as a reduced number of trees compared to our implementation in Phase 1. While the lack of tuning for *mtry* weakens our analysis, we still believe the current model to be indicative of the Random Forest algorithm's performance potential. It is in line with the relative performance of the models observed in Phase 1, and tuning results from Phase 1 indicate that the default recommended value for *mtry* might anyway be close to the optimal value. The regrettable reduction in *ntree* was a computational necessity, as the memory required to fit our original number of trees exceeded the available memory. This does not seem to have undermined model performance. Our final Random Forest model configuration for Phase 2 can be seen in Table A12.2.

Table A12.2: Random Forest Hyperparameter Configuration - Phase 2

Hyperparameter	Value
mtry	20
ntree	500

A12.3 Deep Feedforward Neural Network

Tuning the DFNN model was conducted in a similar manner as in Phase 1 by implementing random search of 120 models on the same tuning grid of values. The resulting final configuration can be seen in Table A12.3.

[86] $p/3$, where p is the number of predictor variables.

Table A12.3: Deep Feedforward Neural Network Tuned Hyperparameters - Phase 2

Hyperparameter	Value
Number of epochs completed	111
Batch size	16
Layer 1 hidden units	256
Layer 2 hidden units	512
Layer 1 dropout rate	0
Layer 2 dropout rate	0.4
Layer 1 bias initializer	0
Layer 2 bias initializer	0.01
Initial learning rate	0.005
Learning rate annealing factor	0.0812
L2 regularization (Weight Decay)	0.0001
Optimizer	Minibatch Stochastic Gradient Descent (SGD)
Momentum	0.9
Preset Values	**Value**
Hidden layers	2
Learning rate annealing patience	5
Activation function	Rectified Linear Unit (ReLU)
Max epochs	1000
Early stopping patience	20
Layer 1 weight initializer	Glorot Uniform
Layer 2 weight initializer	Glorot Uniform
Include bias layer 1	True
Include bias layer 2	True

Interestingly, the resulting configuration is the same as the one identified in Phase 1 of the analysis. The only difference between the configurations in Phase 1 and Phase 2 is the number of epochs completed, with 92 epochs completed in Phase 1 compared to 111 in Phase 2. This allows for a little more complexity in the model compared to before. Otherwise the hyperparameter configurations have similar effects as in Phase 1; we therefore refer to the discussion in Appendix A11.3 for a deeper analysis of the final configuration.

While we expected larger differences in model configuration due to the increased number of variables in the data set, the results indicate that the optimal model found in Phase 1 was robust to increasing the dimensionality of the data. It may also indicate that this model configuration is well suited for the characteristics of our data, and that the additional variables did not change the overall character of the data set. Furthermore, it

should be noted that, due to reproducibility considerations, the tuning process in Phase 2 sampled the same 120 configurations as were sampled in Phase 1. An obvious consequence is that the effective ability to select a new configuration is therefore limited to those 120 configurations.

A12.4 Linear Regression

The final linear model using all available predictors is shown in Table A12.4. Note that the age variable and the dummy variable for housing cooperative are removed before fitting the linear model to avoid problems with multicollinearity. The final estimates show that the vast majority of predictors are associated with price and statistically significant. However, there are some predictors that do not seem to be associated with the response variable at all, such as the percentage of undeveloped site area and number of semi-detached houses in adjoining squares. Again, since the least squares assumptions may not be satisfied, we cannot use the estimated model for statistical inference. Although many of the coefficients are consistent with with the SHAP plot in Figure 5.2, many of them are not. This may indicate that the least squares assumptions are in fact not satisfied, or that there are non-linear properties inherent in some of the features which a linear regression model cannot capture, resulting in high bias. The SHAP dependence plot in Figure 5.3 provides evidence that both NIBOR, PRom and Longitude have a non-linear relationship with price per square meter. This may explain why the estimated coefficients for CPI and NIBOR, for instance, is counter-intuitive.

Table A12.4: Linear Regression Coefficients in Phase 2

	Estimate
TargetPriceCommondebt	−0.00000***
PRom	−0.006***
BRA	0.001***
BTA	0.001***
BuildYear	0.001***
NumberOfBedrooms	0.026***
Balcony	0.015***
Elevator	0.030***
SiteArea	0.00000*
SiteAreaUndeveloped	−0.00000*
Longitude	−0.00002***
Latitude	0.00005***
CoastDistance	−0.0001***
CoastDirection	−0.0001**
SiteSlope	−0.001**
SiteSlopeDirection	0.00001
NumberOfUnitsInAdjoiningSquares	0.0003***
NumberOfUnitsInAdjoiningSquaresX2	−0.0002***
NumberOfDetachedHousesInAdjoiningSquares	−0.0003***
NumberOfDetachedHousesInAdjoiningSquaresX2	−0.0001*
NumberOfFlatsInAdjoiningSquares	−0.0003***
NumberOfFlatsInAdjoiningSquaresX2	0.0002***
NumberOfAttachedHousesInAdjoiningSquares	−0.001***
NumberOfAttachedHousesInAdjoiningSquaresX2	0.0003***
NumberOfSemiDetachedHousesInAdjoiningSquares	0.0001
NumberOfSemiDetachedHousesInAdjoiningSquaresX2	−0.00000
PreviousValue	0.000***
PreviousValueCommondebt	0.00000***
CPI	−0.005***
NIBOR	0.011***
Brent	−0.001***
SunsetHour	0.025***
Altitude	0.0001*
SalesYear	0.079***
GroundFloor	−0.059***
TopFloor	0.052***
SiteAreaUndevelopedPerc	0.0001
SalesMonth.2	0.015***
SalesMonth.3	0.020***
SalesMonth.4	0.030***
SalesMonth.5	0.040***
SalesMonth.6	0.040***
SalesMonth.7	0.047***
SalesMonth.8	0.061***
SalesMonth.9	0.055***
SalesMonth.10	0.061***
SalesMonth.11	0.054***
SalesMonth.12	0.047***
CityDistrict.GRUNERLOKKA	−0.096***
CityDistrict.SAGENE	−0.078***
CityDistrict.ST.HANSHAUGEN	−0.036***
CityDistrict.ULLERN	−0.117***
EstateSubType.Annen.subtype	0.044***
EstateSubType.Leilighet	−1.681***
OwnershipType.Aksjeleilighet	1.649***
OwnershipType.Borettslag	1.696***
SiteOwnershipType.Festet	−0.011***
PreviousPriceValueCategory.Markedsalg	0.012***
Constant	−472.710***
Observations	80,781
Note:	*p<0.1; **p<0.05; ***p<0.01

A12.5 Stacked Regression

Table A12.5 shows the regularization parameters, base learner weights and base learner performances of the Stacked Regression model in Phase 2. See Table A12.6 for a full overview of the base learners' configurations. As in Phase 1, XGBoost is the best performing base learner, but this time Random Forest performs marginally better than the DFNN. The linear benchmark model clearly performs the worst. The relative weighting of the base learners reflect these performances by weighting XGBoost the highest, and directly penalizing the benchmark model with a negative weight. Although the RF model outperforms the DFNN base learner, the stacked model gives the latter a higher weight than RF, which is weighted only marginally above zero. This may indicate that the two tree-based models XGBoost and RF are highly correlated and capture much of the same information, while the DFNN is able to complement the XGBoost model better than RF due to an ability to capture different feature relationships in the data. The low value for λ indicates almost no regularization is performed in the metalearner.

As in Phase 1, the base learner performances do not fully replicate the performance of our individually tuned models due to package differences that force different implementations. In fact, all base learners have a worse test-error performance than their individually tuned counterparts, apart from XGBoost, which performs as well as the individual model. The superior performance of Stacked Regression in Phase 2, although by a small margin compared to the individual XGBoost model, may indicate that the stacking procedure allows the different base learners to compliment each other in identifying relationships between features.

Table A12.5: Stacked Regression - Final regularization parameters and weights - Phase 2

This table shows the final configuration of the Stacked Regression model for Phase 2. All base learners were trained using optimal values found during tuning of the individual models. Where we were unable to implement those values, similar or default values and approaches were chosen. Alpha was fixed at default value, while lambda was tuned through the package's automatic lambda search tuning function.

Parameter/Weight	Value	Test RMSE	CV RMSE*
α	0.5		
λ	0.0007756108		
XGBoost	0.9243904	0.1076	0.1033
FNN	0.08342137	0.1240	0.1184
RF	0.003390168	0.1201	0.1164
Linear Regression	-0.006353007	0.1774	0.1686
Intercept	-0.05300023		

*Cross-validation error

Table A12.6: Stacked model configuration - Phase 2

*Values deviate from individual tuned models.

Stacked Model Configuration			
Stacked Ensemble			
Hyperparameter		**Value**	
Metalearner		Regularized Linear Model	
Alpha		0.5	
Lambda		0.0007756108	
Lambda Search		True	
Lambda max		Default	
Lambda min		Default	
Family		Gaussian	
CV-folds		5	
CV-folds per base learner		5	
CV-fold assignment		Modulo	
Base Learners			
XGBoost		**DFNN**	
Hyperparameter	**Value**	**Hyperparameter**	**Value**
Booster	gbtree	Epochs completed	111
Number of Trees	1000	Batch size	16
Max Tree Depth	10	Layer 1 Hidden Units	256
Learning Rate	0.02	Layer 2 Hidden Units	512
Gamma	0	Layer 1 Dropout rate	0
Minimum Child Weight	10	Layer 2 Droupout rate	0.4
Column Subsampling	0.625	Layer 1 Bias Initializer*	NULL
Row Subsampling	0.875	Layer 2 Bias Initializer*	NULL
		Initial Learning rate*	NULL
		Learning rate annealing*	NULL
		L2 Regularization (Weight Decay)	0.0001
		Optimizer*	Adadelta
		Momentum start*	0.9
		Momentum stable*	0.9
		Hidden Layers	2
		Learning rate annealing patience*	0
		Activation function	ReLu
		Max Epochs	111
		Early stopping patience	20
		Layer 1 Weight initializer*	UniformAdaptive
		Layer 2 Weight initializer*	UniformAdaptive
		Include bias layer 1	True
		Include bias layer 2	True
		Loss*	Quadratic
Random Forest		**Linear Regression**	
Hyperparameter	**Value**	**Hyperparameter**	**Value**
Number of trees	500	Alpha	0
Number of predictors sampled	20	Lambda	0
		Solver	IRLSM
		Family	Gaussian

A13 Property Tax Calculation Procedure

To estimate the aggregated property taxation effects of the Phase 2 Stacked Regression model and Phase 1 SSB benchmark model, we undertake the following process:

1. Back-transform[87] the predicted price per square meter from our analyses in Chapter 5 and the actual price pre square meter to get predicted values and actual values in NOK.

2. Compute each property's *correct* property value and *predicted* property value by multiplying the actual[88] square meter prices (y) and predicted square meter prices ($\hat{y}$) with the size[89] of the property.

3. For each property, compute *predicted* and *correct* property tax (PT) using the following formula:[90]

$$PT = \max\{0,\ ((Property\ value * 0.70 - 4,000,000) * 0.003)\} \quad (.10)$$

4. Summarize the correct and predicted property taxes for each observation in the test set of each model.

5. Compute the total tax error (TE) for each model using the formula:

$$TE = \frac{Predicted\ tax - Correct\ tax}{Correct\ tax} \quad (.11)$$

[87] e raised to the power of x where x is the natural logarithm of the predicted value for instance x.
[88] Obtained from the test set.
[89] Square meters of primary area (PRom)
[90] Derived from tax details shown in Table 6.1.