TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

JUUSO ERONEN

HOUSING UNIT PRICE PREDICTION SYSTEM

Master's thesis

Examiner: Associate Professor Heikki Huttunen

The examiner and topic of the thesis were approved on 8 May 2018

ABSTRACT

JUUSO ERONEN: Housing Unit Price Prediction System
Tampere University of Technology
Master of Science Thesis, 51 pages, 1 Appendix page
October 2018
Master's Degree Programme in Information Technology
Major: Data Engineering and Machine Learning
Examiner: Associate Professor Heikki Huttunen

Keywords: machine learning, regression, Amazon Web Services, housing unit pricing

The objective of this thesis was to compare different regression methods for predicting the price of housing unit, to design and develop a price estimation software and to determine which features are most important when determining the price of a housing unit. The software was developed for Alma Mediapartners Oy to be integrated with their housing marketplace Etuovi.com. Amazon's SageMaker cloud machine learning platform was used to develop and deploy the software.

The data consisted of Etuovi.com's housing unit advertisements posted by real estate agencies and individual customers. Comparing the machine learning algorithms and developing the software used data from a time period of one year. The features used for training the models included, for example, location, size of the housing unit, age of the building and house type. The tested algorithms included linear regression, regression tree, random forest, gradient boosting and extreme gradient boosting out of which extreme gradient boosting had the best performance. The final model showed that over half of the test samples had an error of less than one percent while 80% of the samples had an error of less than ten percent. Less than four percent of the test samples had an error of 25% or more.

The software was developed on Amazon SageMaker following SageMaker's developer guide. The software fetches the housing unit dataset from Etuovi.com's data warehouse and trains a model on the dataset on a virtual machine using the extreme gradient boosting -algorithm. The trained model is then hosted in the cloud and can be integrated with Etuovi.com as an independent component.

TIIVISTELMÄ

JUUSO ERONEN: Hinta-arviojärjestelmä kiinteistöille
Tampereen teknillinen yliopisto
Diplomityö, 51 sivua, 1 liitesivu
Lokakuu 2018
Tietotekniikan diplomi-insinöörin tutkinto-ohjelma
Pääaine: Data Engineering and Machine Learning
Tarkastaja: Apulaisprofessori Heikki Huttunen

Avainsanat: koneoppiminen, regressio, Amazon Web Services, kiinteistöjen hinnoittelu

Työn tarkoituksena oli vertailla erilaisia regressiomenetelmiä huoneistojen hintojen ennustamiseen, suunnitella ja kehittää hinta-arviosovellus sekä määrittää, mitkä huoneiston ominaisuudet ovat tärkeimmät sen hintaa määritettäessä. Sovellus kehitettiin Alma Mediapartners Oy:lle osaksi Etuovi.com -asuntoportaalia. Sovelluksen kehittämiseen ja pystyttämiseen käytettiin Amazonin SageMaker -koneoppimisalustaa.

Data rakentui kiinteistönvälittäjien ja yksityisten ilmoittajien Etuovi.comiin lisäämistä asuntoilmoituksista. Koneoppimisalgoritmien vertailuun ja sovelluksen kehitykseen käytettiin vuoden huoneistodataa. Opetukseen käytetyt piirteet sisälsivät muun muassa sijainnin, asunnon koon, rakennuksen iän sekä talotyypin. Testattuihin algoritmeihin lukeutuivat lineaarinen regressio, päätöspuu, random forest, gradient boosting ja extreme gradient boosting. Näistä parhaiten suoriutui extreme gradient boosting. Viimeistellyllä mallilla yli puolilla testikohteista virheen suuruus oliu vähemmän kuin yksi prosentti, kun taas 80%:lla kohteista kohteista virhe oli alle kymmenen prosenttia. Vain neljällä prosentilla kohteista virhe oli enemmän kuin 25%.

Sovelluksen kehitykseen käytettiin Amazonin SageMaker alustaa ja kehitys tapahtui SageMakerin kehitysohjetta seuraten. Sovellus noutaa asuntodatan Etuovi.comin tietovarastosta ja kouluttaa mallin datasetin avulla virtuaalikoneella käyttäen extreme gradient boosting -algoritmia. Koulutettu malli tuotteistetaan pilvipalveluna ja voidaan liittää osaksi Etuovi.comia.

PREFACE

This thesis was done in 2017-2018 for Alma Mediapartners Oy. I was given almost full discretion on the project and the option to use time to try out different tools and methods. This further fueled my interests in the topic and motivated me to overcome the challenges of the project.

The software was my first piece of work that ended up being used by large number of users from all around Finland. For this, I would like to express my gratitude to my examiner, Associate Professor Heikki Huttunen, for invaluable guidance and comments. I would also like to thank both Antti Siiskonen and Antti Koivisto of Alma Mediapartners Oy for the topic and giving me the chance to work on this project.

In Tampere, Finland, on 23 October 2018

Juuso Eronen

CONTENTS

1. INTRODUCTION 1
2. BACKGROUND 3
   2.1 Housing unit appraisal 3
   2.2 Previous work 3
3. THEORY 6
   3.1 Machine learning 6
   3.2 Feature engineering 7
      3.2.1 Feature transformation and construction 8
      3.2.2 Feature selection 8
   3.3 Linear models 9
      3.3.1 Simple linear regression 10
      3.3.2 Multiple linear regression 10
      3.3.3 Least-squares method 11
      3.3.4 Polynomial regression 13
      3.3.5 Ridge regression 14
      3.3.6 LASSO regression 14
   3.4 Classification and regression trees 15
   3.5 Tree ensembles 17
      3.5.1 Random forest 18
      3.5.2 Gradient tree boosting 18
      3.5.3 Extreme gradient boosting 21
   3.6 Model evaluation 22
      3.6.1 Validation 22
      3.6.2 Evaluation metrics 23
4. EXPERIMENTS 25
   4.1 Data and preprocessing 25
   4.2 Feature selection 27
   4.3 Validation 28
   4.4 Linear regression 28
   4.5 Regression tree 30
   4.6 Random forest 31
   4.7 Gradient boosting machine 33
   4.8 Extreme gradient boosting 35
   4.9 Summary of performances 37
5. IMPLEMENTATION 41
   5.1 Planning and experimenting 41
   5.2 Amazon SageMaker 41
   5.3 Development and deployment 42
   5.4 Challenges and future development 46

6. CONCLUSIONS 47
REFERENCES 49

APPENDIX A: TRAINING AND DEPLOYING THE MODEL 52

LIST OF FIGURES

Figure 2.1. *Rise of used housing unit prices per square meter in Finland [29].* 4
Figure 2.2. *Rise of used housing unit prices per square meter in Finland (smoothed, scaled to June 2014 = 100) [29].* 4

Figure 4.1. *Variable importances of decision tree regressor (price per square meter).* 31
Figure 4.2. *MAE of the tuned random forest model (price per square meter).* 32
Figure 4.3. *Variable importances of random forest (price per square meter).* 32
Figure 4.4. *MAE of the tuned gradient boosting model (price per square meter).* ... 34
Figure 4.5. *Variable importances of gradient boosting model (price per square meter).* 34
Figure 4.6. *MAE of the tuned XGBoost model (price per square meter).* 36
Figure 4.7. *Variable importances of XGBoost model (price per square meter).* 36
Figure 4.8. *Validation and testing MAE when predicting total price.* 38
Figure 4.9. *Validation and testing MAE when predicting price per square meter.* ... 38
Figure 4.10. *R^2 when predicting total price.* 39
Figure 4.11. *R^2 when predicting price per square meter.* 39
Figure 4.12. *Improvement of performance after hyperparameter optimization (total price).* 40
Figure 4.13. *Improvement of performance after hyperparameter optimization (price per square meter).* 40

Figure 5.1. *Machine learning application workflow.* 43
Figure 5.2. *Screenshot of the user interface of the finished application [29].* 44
Figure 5.3. *Architecture of the application.* 45

LIST OF SYMBOLS AND ABBREVIATIONS

XGBoost	eXtreme Gradient Boosting: An ensemble learning algorithm
LASSO	Least Absolute Shrinkage and Selection Operator: Regression analysis method that performs regularization
CART	Classification And Regression Tree: A decision tree algorithm
MAE	Mean Absolute Error: An error measure for interpreting model performance
RMSE	Root Mean Squared Error: An error measure for interpreting model performance
AWS	Amazon Web Services: Amazon's family of remote computing services
EC2	Elastic Compute Cloud: Amazon's cloud computing service
REST	Representational State Transfer: A protocol for creating web services
API	Application Programming Interface: Specifies how software components should interact
ECS	Elastic Container Service: Amazon's container orchestration service
ECR	Elastic Container Registry: Amazon's Docker container registry
R^2	Coefficient of determination: Quantity for how well the model explains the problem

1. INTRODUCTION

Many companies have massive amounts of unused data stored in a data warehouse. The data might be anything from user behavior data to, like in this case, housing unit sales advertisement data. A great amount of valuable data has been collected from the sales advertisements of Alma Mediapartners' housing marketplace service, Etuovi.com. The problem is how to turn the data into a product instead of just keeping it stored pointlessly.

Assessing the price of a housing unit is known to be quite difficult for consumers and housing transaction is one of the most important financial decisions in one's life. These often require assistance from a real estate agent. The idea of developing a pricing tool for consumers has been around for some time among companies that work in the field of real estate business. The real estate agent evaluates the housing unit by its features and amenities like location, size and age. Using these features it is possible to build a pricing application using mathematics, statistics and machine learning as proven by Antipov [2] and Masias [23] in their research on housing unit appraisal.

Machine learning is a process where a computer program learns to automatically recognize dependencies and patterns and make decisions based on data. A machine learning model is also able to generalize the learned information and make decisions on data outside of the training set. This means that the model can make a prediction whether the sample was in the training data or not. Everyday examples of machine learning applications include e-mail spam filters and the recommendation engines used by online stores. In practice, the machine learning model is trained by feeding it preprocessed data as input-output -pairs. The inputs are called explanatory variables and the outputs are called response variables. The algorithm automatically recognizes dependencies and patterns between the inputs and outputs. Different algorithms use different methods for finding the patterns and dependencies and are suited for modeling different kinds of data.

The aim of this thesis is to determine which features are the most important when considering the price of a housing unit, compare different regression methods for predicting the price of housing unit and finally and most importantly, design and develop a price estimation software to be integrated with Etuovi.com. The idea of this pricing application is to attract customers and create interest, leading to more registered users of the host service, Etuovi.com. The software should look trustworthy to consumers and it should be fairly simple and easy to use for an average consumer.

In Chapter 2, the background of this thesis is gone through in more detail. The chapter explains the events and choices that led to the development of the pricing application. Also, earlier research on the topic will be reviewed.

Chapter 3 concentrates on explaining the theoretical background of this thesis, mainly machine learning and the related algorithms. The point is to give the reader an insight to the principles and methodologies utilized in this thesis.

Chapter 4 contains the experiments from processing data to building the models. First, the dataset used and the flow of the experiments are introduced to the reader. Next, machine learning models are built using the chosen algorithms and their performance is reviewed and compared. The best solution will be taken to the implementation phase.

In Chapter 5 the acquired knowledge is taken to the production environment. The chapter explains the steps to create the price appraisal tool from planning to deployment. The chapter also introduces the implementation platform, Amazon SageMaker, and goes through the challenges and future possibilities.

The final chapter sums up the work of the thesis. The results of the experiments and the implementation of the application are reviewed. In Appendix A, the code for training and deploying the machine learning model using Amazon SageMaker can be found.

2. BACKGROUND

This chapter discusses the features and amenities that define the value of housing unit. Price evaluation for housing unit is especially difficult for consumers due to their infrequent occurrence and the uniqueness factor of every property. Thus, housing unit appraisals require the help of an expert almost without exception.

2.1 Housing unit appraisal

The point of an appraisal is to estimate the market value, the most likely price that would be paid for a housing unit under the current market situation and to determine the asking price [1]. The features that make up the price of a housing unit can be split into internal and external characteristics. The internal characteristics consist of the features of the property itself, for example, location, square footage, age and condition of the property. The external characteristics include the law of supply and demand and the current economic situation [1].

Most commonly price estimations are carried out by professional marketplace players like appraisers and real estate brokers. One disadvantage of such a method is the difficulty in ensuring that the appraiser conducts a neutral, unbiased analysis in arriving at the appraised value. This disadvantage also includes the fact that the appraiser may be connected to an interested party like a lender, mortgage broker, buyer, or seller in the transaction, for example through a paid contract [18]. To obtain less biased estimates, one can turn to statistics to obtain a mathematical model to aid in estimating the value of housing unit.

The prices of housing unit have been steadily rising in Finland during the last years as shown in Figures 2.1 and 2.2. It can be seen that the pace of increment is higher around the capital region.

2.2 Previous work

Alma Mediapartners Oy has collected extensive amounts of data on housing unit sales advertisements through their online housing marketplace Etuovi.com. There exists a service, ARVO, that aids real estate agents to explain the value of a housing unit to its seller. The service uses the sales advertisement records of Etuovi.com to calculate housing unit price estimations using a similarity-based approach for prediction.

The specification for Etuovi.com included an appraisal tool for consumers to evaluate their housing unit and help understand the value of housing unit. The purpose of this appraisal tool is to attract customers and raise their interest, leading to more registered

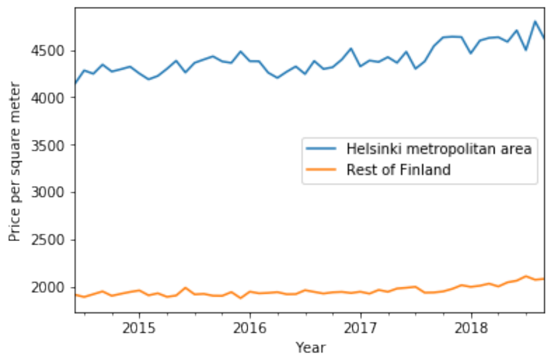


Figure 2.1. Rise of used housing unit prices per square meter in Finland [29].

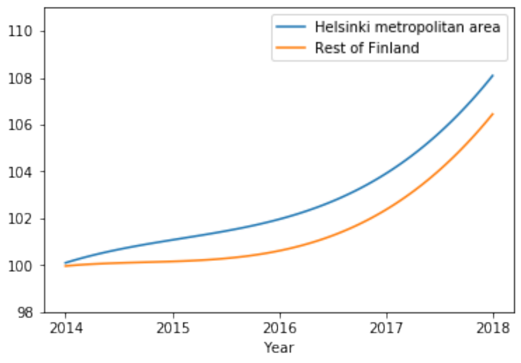


Figure 2.2. Rise of used housing unit prices per square meter in Finland (smoothed, scaled to June 2014 = 100) [29].

users of the host service, Etuovi.com. Some banking companies and housing unit agencies already possess such pricing applications. Integrating ARVO as a part of Etuovi.com was considered but dropped due to refactoring difficulties and the fact that it was built for real estate agents' use and a more consumer-friendly application was desired.

ARVO searches for advertisements with similar features to the target of appraisal from the same postal code area. A similarity value is calculated for each feature and the estimation is based on the distribution of the prices of similar housing units. The system uses the following features for price calculation:

- Location (Postal code)
- Age
- Living area
- Room count
- House type

The distribution is used to generate four different price estimations: low, medium, high and extremely high and it is up to the agent to determine the final asking price while considering other features of the housing unit. The agent can also see other similar units that are also for sale near the unit they are doing the price estimation for. The system proved to be quite accurate with its estimations as long as there is enough data available.

The service does not satisfy all the customer's needs though. The problems mainly came with the fact that the program calculates the estimation using similar properties in close proximity of the appraisal target. Only the properties with the same postal code were used in the calculation. This causes difficulties as the data mass inside one postal coded area might not be homogenous enough as there might be both affordable and high class housing units within the same area. For example, when estimating the price of a high class house in an area where there are only a few of them but a lot of affordable apartments, the estimator will incorrectly base the price of the high class house on the value of the affordable apartments. If there are no recent advertisements within the postal code area, the program is unable to provide an estimation. Also, with this approach even a small amount of biased data can negatively affect the accuracy of the estimate as the amount of data points used in the estimation is usually quite small in many areas in Finland.

As ARVO was designed for professionals to use, some of its features would have to be dropped in order to tailor it better towards consumers as it concentrates on aiding the real estate agent in providing an appraisal instead of simply assessing the value of a property. The application is not built in a modular fashion so major refactoring of the code would be needed in order to use the algorithm. The time needed for each appraisal was also quite long due to the fact that ARVO had to fetch data and make its calculations every time an appraisal was made.

Therefore, Alma Mediapartners decided to start seeking alternative methods to produce appraisals from data. The method of choice was to use machine learning to build a model for price estimation and base the new application on this.

A machine learning based approach for housing unit appraisals has been previously researched by Antipov [2] and Masias [23]. They both developed a system for housing unit appraisal using the Random forest algorithm, which will be introduced later in section 3.5.1. The researches were conducted in Russia and Chile respectively. Antipov states in his paper that there is a lack of research in using modern machine learning methods like the random forest algorithm for housing unit appraisal. Both researches used a wide variety of explanatory variables ranging from the properties of the housing units like living area and location to external amenities like distance from a train station or city center. Both researches were also limited to a small area like just one city.

Random forest was shown to be extremely accurate when predicting the housing unit prices. Both papers compares multiple machine learning algorithms for the housing unit appraisal problem like traditional linear regression, support vector machines, neural networks and random forest out of which the random forest seemed the most effective. The effectiveness of a tree based ensemble method like random forest suggests to research the effectiveness of other tree ensembles like gradient boosting for this problem.

3. THEORY

This chapter introduces the theoretical principles and methodologies of machine learning models. The chapter begins with a short overview of machine learning. This is followed by the key steps on building the model itself. The steps include a deeper look into different techniques used in the proposed system. This includes methods for feature selection, classification and prediction, proposed learning algorithms and the process of evaluating the performance of the trained model.

3.1 Machine learning

The basis of machine learning is to have a computer program automatically learn to recognize patterns and associations in the data and then use the acquired knowledge to make decisions in an operative environment. A machine learning model can generalize the acquired knowledge to explain observations outside the training data. [16]

Common fields that nowadays use machine learning are, for example security heuristics, image analysis, deep learning, object recognition, and pattern recognition [22]. An example of a simple machine learning problem is spam e-mail classification. This is a binary classification task where the objective of the machine is to learn to differentiate spam from normal e-mail to filter out the spam while letting others through by the examples in the training data.

The spam detection example was a classification problem. In classification tasks the program must learn to predict discrete values for the response variables from one or more explanatory variables [15]. This means that the machine must predict the most probable category or a label for new samples. Examples of classification algorithms include decision trees, k-nearest neighbor and support vector classifiers. In the case of classification, the dependent variable is always categorical. If the response variable is continuous, the task is called regression.

In regression the program has to learn to predict continuous values from the training samples [15]. This means that instead of categorical labels, the program learns a real valued function from the explanatory variables to predict the outcome. An example of a regression task is predicting the price of a product where the price takes a real value. Examples of regression algorithms are linear regression and logistic regression.

Both classification and regression are instances of supervised learning. In supervised learning, the goal is to predict the value of an outcome measure based on a number of input measures [15]. This essentially means that the machine learns a function to map the explanatory variables to the correct response variable. Sometimes there are no labels available for training. Providing the true labels for a data set is often labor-intensive and expensive [17].

The task of grouping data without having any prior information on the groups is called clustering [17]. This kind of learning that is done without training labels is called unsupervised learning. In clustering the instances are assigned into groups so that in a group the instances are more similar to each other than compared to other groups based on some similarity measure. [15]. An example of a clustering task is clustering customers into groups by their purchasing habits and then being able to tailor services for them.

There is a third learning method that utilizes both supervised and unsupervised learning. In a situation where we have some labeled samples but a lot of unlabeled data, one possibility is to use semi-supervised learning. A possible approach in semi-supervised learning is to train the model first using the small set of labeled data and then make predictions on the unlabeled data. Next, the most confident predictions can be used as new training data and the model can be trained again on this larger training set. [9]. The dataset used for the experiment in this thesis already contains example input-output pairs. Thus, the methods used fall under supervised learning category.

Models form the central concept in machine learning as they are learned from the data to solve the given task. [9] Models are learned from data through the use of machine learning algorithms. There is a wide range of machine learning models to choose from. In this thesis we concentrate specifically on ensemble methods like random forests [4], gradient boosting [12] and extreme gradient boosting (XGBoost) [8].

In supervised learning the machine learning algorithms require example input-output pairs for training. The dataset consists of instances which contain one or more explanatory variables and the response variable. The inputs are known as the feature matrix $\mathbf{X}$ *in* $R^{N \times D}$ and the outputs as the class labels $\mathbf{y}$ *in* R^{N}.

The used dataset is usually split into three parts before training: training data, validation data and testing data. The model uses the training data to learn connections between inputs and outputs. Validation data is used for evaluating the model during tuning of its hyperparameters. Test data is used to measure the performance of the final model after tuning. This set contains more carefully selected instances that reflect the problems the model would face in real situations.

3.2 Feature engineering

In machine learning, a feature is an individual measurable property of the process being observed [7]. Using a set of features and responses, a machine learning model is capable of performing prediction. The set of features used is one of the determining features considering how well the machine learning application will perform. This is because the model is only as good as the features used [9]. Thus, feature selection, transformation and construction play major roles in building machine learning models.

3.2.1 Feature transformation and construction

The purpose of feature transformation is to improve the usefulness of features by removing, changing or adding information [9]. For example, a feature can be normalized by bounding its values between zero and one. Another common technique is changing categorical variables to numerical. For example, if we have a rating scale with discrete values from one to five, we can transform it to a continuous scale of numerical values.

Feature construction means making new features from one or more existing features or by splitting a single feature to multiple ones. For example, the age of a car is a useful feature when predicting its value at the time of sale. This feature can be constructed by subtracting the car's manufacturing date from its selling date. Some machine learning algorithms can only work with numerical data and can't work with categorical variables. This is why categorical variables usually need to be encoded to make them processable by algorithms. Common way is to use one-of-K or one-hot encoding, in which the explanatory variable is encoded using a binary feature for each of the feature's possible values [15]. For example, we can have a categorical variable with possible values 'red', 'green', 'blue'. One-hot encoding will transform this into numerical by constructing one binary variable for each of the possible values: red: 0, 1, green: 0, 1 and blue: 0, 1.

New features can also be constructed by expanding the dimensionality of the original features [26]. This can aid in discovering information and revealing new patterns that would otherwise be out of sight. This operation is related to the kernel trick used with support vector machines where it is done implicitly. Polynomial feature construction is an example of feature construction by dimensionality expansion.

3.2.2 Feature selection

Feature selection focuses on selecting the most valuable variables from the data. The objective is to maximize prediction accuracy by finding the features that best describe the input data while minimizing the effects of noise and irrelevant variables [14]. For example, in the case of e-mail filtering one could look at what kind of words and linguistic structures define a spam e-mail. Feature selection brings many benefits with it. Feature selection helps to visualize and understand the data better. It reduces storage space, training time and utilization time needed by the data. It also helps with the curse of dimensionality problem and improves the performance of the predictor [14].

Feature selection methods can be split roughly to three categories which are filter methods, wrapper methods and embedded methods. Filter methods order the features by ranking them by their usefulness. The methods rank variables using different ranking techniques

[7]. Basically, these methods consider the relationship between features and the responses and suppress the least important features. Filter methods are quite fast compared to other feature selection methods as they do not require to train the predictor and are sometimes the only possible solution if the amount of features is extremely high. A drawback of this method is that it does not take into account the relationships between individual features. Examples of filter methods include for example, removing features with the lowest variance [7].

Wrapper methods use the predictor itself to evaluate variables. This is done by learning the model with a set of variables and testing how good the model is [7]. This means that the importances of features are evaluated on the model itself. It is possible to perform an exhaustive search to find out which variables contribute the most to the prediction. However, this might be problematic if the number of variables is very large due to high computational power requirements. Special search algorithms can be used to make the task computationally more feasible. These include best-first, branch-and-bound, simulated annealing and genetic algorithms [14]. Ways to search for the optimal set of features also include forward selection, backwards elimination and random search. Forward selection incrementally trains a model and selects the best feature(s) until convergence. Backwards elimination on the other hand tries to improve the performance by recursively eliminating least important features from the dataset [17]. These recursive selection methods can also be used at the same time to narrow the interval where the most optimal set lies.

The main approach of embedded methods is to incorporate the feature selection as part of the training process [14]. Embedded feature selection is computationally less expensive than wrapper methods but are specific to some learning algorithms, for example the decision tree based random forests and gradient boosting machines. Another method for feature selection for regression problems is the Least Absolute Shrinkage and Selection Operator or LASSO, which will be introduced further in Section 3.3. LASSO basically applies regularization to the parameters and shrinks some of them to zero, thus eliminating some of the features [10].

3.3 Linear models

Linear models belong to the category of geometric models. These kinds of models are constructed in instance space by using geometric shapes like lines, planes and hyperplanes [9]. The models use these shapes as predictors and decision boundaries. Linear models exist for both classification and regression problems.

Linear models are parametric. This means that they have a fixed form and have a small number of numeric parameters that need to be learned from the data. These models are also stable, which means that small variations in the training data only slightly alter the performance of the model. On the other hand, these models are not robust, meaning that even a single faulty observation, for example a completely absurd price for a housing unit, might cause the model to become completely useless. This can be mitigated by feature selection though. Because they have fewer parameters than other models, linear models are less likely to overfit the training data. The downside of this is that linear models can be prone to underfitting meaning that they cannot give a precise approximation of the target. [9]

As for types of linear models, this thesis concentrates on linear regression and its application in housing unit price assessment. The simplest form of linear regression basically tries to fit a straight line to the continuous-valued responses using one explanatory variable.

3.3.1 Simple linear regression

Simple linear regression is the case when we have only a single explanatory variable *x in R* to predict the response y. It is defined as:

$$y = \beta_0 + \beta_1 x + \epsilon \tag{3.1}$$

where the intercept β_0 and the slope β_1 are unknown constants and ϵ is a random error component. The intercept β_0 equals the value of the prediction when $x = 0$. Slope β_1 describes how the predicted value changes regarding to x. The error consists of residuals which are caused by random variance in the data. Residuals are the differences between the observed and the predicted values. The error is a random variable that is assumed to be distributed with $E(\epsilon) = 0$ and $Var(\epsilon) = \sigma^2$. The quantity σ^2 is called the error variance. It is also usually assumed that the errors are uncorrelated meaning that there is no dependency between them. Because the errors are uncorrelated the responses are also uncorrelated. [33], [25], [34]

3.3.2 Multiple linear regression

If there are more than one explanatory variable, the regression model is called multiple linear regression. It is defined as:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_k x_k + \epsilon, \tag{3.2}$$

where β_j are the regression coefficients. The model describes a hyperplane in the k - dimensional space of the regressor variables x_j. This can be written more conveniently in matrix form [25]:

$$\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\epsilon}, \tag{3.3}$$

where:

$$\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}, \qquad \mathbf{X} = \begin{bmatrix} 1 & x_{11} & x_{12} & x_{13} & \dots & x_{1k} \\ 1 & x_{21} & x_{22} & x_{23} & \dots & x_{2k} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n1} & x_{n2} & x_{n3} & \dots & x_{nk} \end{bmatrix},$$

$$\boldsymbol{\beta} = \begin{bmatrix} \beta_0 \beta_1 \\ \beta_2 \\ \vdots \\ \beta_k \end{bmatrix}, \qquad \boldsymbol{\epsilon} = \begin{bmatrix} \epsilon_1 \\ \epsilon_2 \\ \vdots \\ \epsilon_n \end{bmatrix}.$$

In general, $\boldsymbol{y}$ is an $n \times 1$ vector of the observations, $\boldsymbol{X}$ is an $n \times (k{+}1)$ matrix of the levels of the regressor variables, $\boldsymbol{\beta}$ is a $(k{+}1) \times 1$ vector of the regression coefficients, and $\boldsymbol{\epsilon}$ is an $n \times 1$ vector of random errors [25].

3.3.3 Least-squares method

Least-squares method can be used to estimate the regression coefficients β_j in 3.2. The first published treatment of the method of least squares was included in an appendix of Legendre's book in 1805 [20].

The regression model in 3.2 can be written as:

$$\begin{aligned} y_i &= \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \dots + \beta_k x_{ik} + \epsilon_i \\ &= \beta_0 + \sum_{j=1}^{k} \beta_j x_{ij} + \epsilon_i, i = 1, 2, \dots, n. \end{aligned} \tag{3.4}$$

The least-squares function is defined as:

$$S(\beta_0, \beta_1, \dots, \beta_k) = \sum_{i=1}^{n} \epsilon^2 = \sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{k} \beta_j x_{ij} \right)^2 . \tag{3.5}$$

The function S must be minimized in regard to β_0,β_1,...,β_k. The most convenient way to obtain the solution is to use the matrix solution 3.3 to find a vector $\hat{\boldsymbol{\beta}}$ that minimizes:

$$S(\boldsymbol{\beta}) = \sum_{i=1}^{n} \epsilon_i^2 = \boldsymbol{\epsilon}^T \boldsymbol{\epsilon} = (\mathbf{y} - \mathbf{X}\boldsymbol{\beta})^T (\mathbf{y} - \mathbf{X}\boldsymbol{\beta}). \tag{3.6}$$

This can be expressed as:

$$\begin{aligned} S(\boldsymbol{\beta}) &= \mathbf{y}^T\mathbf{y} - \boldsymbol{\beta}^T\mathbf{X}^T\mathbf{y} - \mathbf{y}^T\mathbf{X}\boldsymbol{\beta} + \boldsymbol{\beta}^T\mathbf{X}^T\mathbf{X}\boldsymbol{\beta} \\ &= \mathbf{y}^T\mathbf{y} - 2\boldsymbol{\beta}^T\mathbf{X}^T\mathbf{y} + \boldsymbol{\beta}^T\mathbf{X}^T\mathbf{X}\boldsymbol{\beta} \end{aligned} \tag{3.7}$$

because since $\boldsymbol{\beta}^T \boldsymbol{X}^T \boldsymbol{y}$ is a *1* × *1* matrix, or a scalar, and its transpose $(\boldsymbol{\beta}^T \boldsymbol{X}^T \boldsymbol{y})^T$ is the same scalar. The minimum appears when the gradient becomes zero, so the least-squares estimators must satisfy:

$$\left.\frac{\partial S}{\partial \boldsymbol{\beta}}\right|_{\hat{\boldsymbol{\beta}}} = -2\mathbf{X}^T\mathbf{y} + 2\mathbf{X}^T\mathbf{X}\hat{\boldsymbol{\beta}} = \mathbf{0}. \tag{3.8}$$

This simplifies to:

$$\mathbf{X}^T\mathbf{X}\hat{\boldsymbol{\beta}} = \mathbf{X}^T\mathbf{y}. \tag{3.9}$$

This is called least-squares normal equations [25], [34], [11].

To find the vector $\hat{\boldsymbol{\beta}}$, which is an estimation of the vector $\boldsymbol{\beta}$, solve 3.9 in regard to it by multiplying both sides with the inverse $(\mathbf{X}^T\mathbf{X})^{-1}$:

$$\hat{\boldsymbol{\beta}} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y} \tag{3.10}$$

provided that the inverse $(\mathbf{X}^T\mathbf{X})^{-1}$ exists. The matrix will always exist if the regressors are linearly independent. This means that no column of the $\boldsymbol{X}$ matrix is a linear combination of the other columns.

The vector of fitted values $\hat{y}_i$ corresponding to the observed values y_i is:

$$\hat{\mathbf{y}} = \mathbf{X}\hat{\boldsymbol{\beta}} = \mathbf{X}(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y} = \mathbf{H}\mathbf{y}. \tag{3.11}$$

The $n \times n$ matrix $\mathbf{H} = \mathbf{X}(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T$ is usually called the hat matrix. The hat matrix plays an important part in regression analysis as it maps the vector of observed values into a vector of fitted values. [25], [34]

The residual $e_i = y_i - \hat{y}_i$ is the difference between an observed value y_i and the corresponding fitted value $\hat{y}_i$. The n residuals can be expressed in matrix form:

$$\mathbf{e} = \mathbf{y} - \hat{\mathbf{y}}. \tag{3.12}$$

There are other ways to express the vector of residuala by combining the above equations [25]:

$$\mathbf{e} = \mathbf{y} - \mathbf{X}\hat{\boldsymbol{\beta}} = \mathbf{y} - \mathbf{H}\mathbf{y} = (\mathbf{I} - \mathbf{H})\mathbf{y}. \tag{3.13}$$

This allows a simpler way to calculate the residuals in the case when calculating $\hat{\mathbf{y}}$ is not required.

3.3.4 Polynomial regression

A polynomial regression model is a linear regression model where the dimensionality of the existing variables has been expanded to capture curvilinear or even complex nonlinear relationships [25]. For example the second order polynomial linear regression model $F(x_1, x_2)$ maps to:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_{11} x_1^2 + \beta_{22} x_2^2 + \beta_2 x_1 x_2 + \epsilon \tag{3.14}$$

where x^2 is the second order polynomial of a feature and $x_1 x_2$ is the product of the two variables.

Polynomial features are useful in situations when it is known that the response function contains curvilinear effects. Montgomery [25] lists important considerations that one must take into account when fitting a polynomial regression model.

- It is important to keep the order of the model as low as possible and avoid using higher than second order polynomials.
- For selecting the order of an approximating polynomial, one can successively fit models of increasing order (forward selection) or fit highest order model and delete terms one by one (backward elimination).
- Extrapolating polynomial models might have unexpected consequenses and turn the model in inappropriate directions.
- Increasing the order of the polynomial causes the $\mathbf{X}^T\mathbf{X}$ matrix to become ill-conditioned, meaning that the matrix inversion calculates become inaccurate and error might be introduced to the parameter estimates. The same might happen if the values of x are limited to a narrow range.
- The model is considered hierarchical if it contains all of the terms below the highest order.

3.3.5 Ridge regression

The ridge regression is a regression method that adds a penalty to the regression coefficients β. The ridge coefficients minimize a residual sum of squares: [17]

$$\hat{\beta}^{ridge} = argmin_{\beta} \left\{ \sum_{i=1}^{n} (y_i - \beta_0 - \sum_{j=1}^{k} \beta_j x_{ij})^2 + \lambda \sum_{j=1}^{k} \beta_j^2 \right\} \tag{3.15}$$

where λ controls the amount of coefficient shrinkage and thus the amount of regularization. Zero value for λ gives the ordinary least-squares while larger values shrink the coefficients towards zero. Another way to write the ridge problem is:

$$\begin{aligned} \hat{\beta}^{lasso} &= argmin_{\beta} \sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{k} \beta_j x_{ij} \right)^2 \\ &\text{subject to} \quad \lambda \sum_{j=1}^{k} \beta_j^2 \leq t. \end{aligned} \tag{3.16}$$

There is a one-to-one correspondence between λ from Equation (3.15) and t. For every λ exists a t with the same minimum. Ridge regression is used to prevent multicollinearity and reduce the complexity of the model. Also, shrinking the irrelevant attributes increases the accuracy of the model [17].

3.3.6 LASSO regression

The Least Absolute Shrinkage and Selection Operator is another regression method that adds a penalty term to the coefficients β similar to ridge regression but with important differences. LASSO coefficients minimize a penalized residual sum of absolute value: [17] [31]:

$$\begin{aligned} \hat{\beta}^{lasso} &= argmin_{\beta} \sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{k} \beta_j x_{ij} \right)^2 \\ &\text{subject to} \quad \lambda \sum_{j=1}^{k} |\beta_j| \leq t \end{aligned} \tag{3.17}$$

where t is the parameter determining the amount of regularization. This can also be expressed in Lagrangian form:

$$\hat{\beta}^{lasso} = argmin_{\beta} \left(\frac{1}{2} \sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{k} \beta_j x_{ij} \right)^2 + \lambda \sum_{j=1}^{k} |\beta_j| \right). \tag{3.18}$$

The LASSO constraint $\sum_1^k |\beta_j|$ makes the solutions nonlinear in y_i and there is no closed form expression [17].

LASSO can be used for both regularization and feature selection. The term λ controls the magnitude of the penalty. Zero value for λ gives the ordinary least-squares similar to ridge while very large values set the coefficients β to exactly zero causing underfitting. The fact that LASSO shrinks the coefficients of least important features to zero can be utilized in feature selection. By setting a larger value for λ more coefficients are reduced to zero [10]. Using LASSO also helps with overfitting because the coefficients not associated with the response variable are set to zero [10].

3.4 Classification and regression trees

Decision tree models are easy to understand and expressive. They are a very popular types of machine learning models and can be used with both classification and regression tasks. Decision trees are sequential models. This means that they consist of a sequence of tests that compare a numeric attribute against a threshold value or a nominal attribute against a set of possible values. As decision trees consist of logical rules and can be re-represented as sets of if-then rules to improve human readability. They are a lot easier to understand and interpret than ”black-box” models like neural networks. [9], [24], [19]

The trees handle instances by sorting them from the root node, which is the topmost node of the tree, to some leaf node, which are the last nodes of the tree. The leaf nodes contain the classification for the instance or a local regressor in case of a regression problem. Each tree node has a test for some variable of the instance and each branch contains a possible value for the variable. The instance moves from the root of the tree and ends up at one of the leaf nodes depending on its attributes. [24], [5]

Decision trees are very sensitive to even the smallest changes in data. The trees also tend to learn the outliers if grown too deep, causing them to easily overfit [24]. This thesis concentrates on the CART-methodology by Leo Breiman for decision tree construction. Other widely used algorithms include ID3 and its successor C4.5.

Classification and regression tree models are obtained by recursively partitioning the data space and fitting a simple prediction model within each partition. This partitioning can be represented graphically as a decision tree [21]. The tree is a binary decision tree constructed by splitting the root node, which contains the whole learning sample, into two child nodes and repeating for subsequent nodes [5]. In classification tree models the dependent variable is categorical, while in regression trees it is continuous [35]. CART was not the first decision tree model in machine learning but it is the first to be described with analytical correctness and supported by sophisticated statistics and probability theory [28].

The basic idea when growing the tree is to choose the best split at each node. The process is started by finding the best split for each predictor variable. Continuous and ordinal predictors are first sorted from the smallest to the largest. Then the best split is determined by examining each split point between the smallest and largest value and choosing the one which maximizes the splitting criterion. For nominal predictors, examine all possible subsets of categories to find the best split. After finding the best splits for the predictors, the one that maximizes the splitting criterion is chosen. The node is then split repeatedly if the stopping rules are not satisfied. [5]

The CART algorithm finds the optimal split using one of the following splitting rules. Gini, twoing, ordered twoing and symmetric gini for classification trees and least squares and least absolute deviation for regression trees [28].

For regression trees, the split is usually chosen by minimizing the least squares error criterion,

$$\frac{1}{N}\sum_{n}(y_n - r(\beta, \mathbf{x}_n))^2 \tag{3.19}$$

where n is the sample size, $< x_n, y_n >$ is a data point and $r(\beta, \mathbf{x}_n)$ is the prediction of the regression model $r(\beta, \mathbf{x})$ for the case $< x_n, y_n >$.

To minimize the criterion for $y(t)$, we use the average of y_n for all cases (x_n, y_n) falling into t; that is, the minimizing $y(t)$ is [5] [32]:

$$\bar{y}(t) = \frac{1}{N(t)}\sum_{\mathbf{x}_n \epsilon t} y_n. \tag{3.20}$$

Any path from the root node to a node t corresponds to a part of the input set n. The error of node t can be defined as the average of the squared between the y values of the instances. Taking the predicted value in any node as $\bar{y}(t)$ the error in an arbitrary leaf node is:

$$Err(t) = \frac{1}{N}\sum_{\mathbf{x}_n \epsilon t}(y_i - \bar{y}(t))^2. \tag{3.21}$$

The error of the tree T can be defined as a weighted average of the error of its leaves [5] [32]:

$$Err(T) = \frac{1}{N}\sum_{t \epsilon \tilde{T}}\sum_{\mathbf{x}_n \epsilon t}(y_i - \bar{y}(t))^2 = \sum_{t \epsilon \tilde{T}} Err(t) \tag{3.22}$$

where $\tilde{T}$ is the set of leaves in tree T.

A binary split splits a node in two. The goal of the splitting rule is to choose a split that most decreases the error of the resulting tree. The error of a split s is defined as the weighed average of the errors in the sub-nodes resulting from the split [32]:

$$Err(s,t) = \frac{N_{t_L}}{N_t} \times Err(t_L) + \frac{N_{t_R}}{N_t} \times Err(t_R) \tag{3.23}$$

where t_L is the left child node of t and n_{t_L} the cardinal of this set. t_R is the right child node of t and n_{t_R} the cardinal of this set. The best split for a node t given a set S of candidate splits is the one that maximizes:

$$\Delta Err(s,t) = Err(t) - Err(s,t). \tag{3.24}$$

To prevent the tree from becoming too deep and overfitting, pruning is implemented. The point of pruning is to make the tree less dependent on irrelevant features of the training set and thus making it able to perform better with unseen data. Pruning is divided in two parts: pre-pruning (stopping rules) and post-pruning. Stopping rules stop the growth of the tree to avoid it becoming too specific and to avoid the computationally more costly post-pruning process. The nodes continue to split until they meet a stopping criterion. Common stopping rules include the following [19]:

- All instances in the training set belong to a single value of y.
- The maximum tree depth has been reached.
- The number of cases in the terminal node is less than the minimum number of cases for parent nodes.
- If the node were split, the number of cases in one or more child nodes would be less than the minimum number of cases for child nodes.
- The best splitting criteria is not greater than a certain threshold.

Post-pruning means cutting out some branches of the tree after the growing phase if they do not meet some criterion [19]. CART trees are pruned by using a two-stage cost-complexity pruning algorithm. First, a sequence of increasingly smaller subtrees $\{T_k\}_{k=0}^{K}$ of the full tree T_0 are built on the training data. In the second stage, one of these trees is chosen as the pruned tree using a holdout dataset separate from the training data [5]. Another pruning method is called reduced error pruning where the terminal nodes that provide little to the tree are pruned and if the error of the tree decreases, these prunings are kept.

3.5 Tree ensembles

The idea of ensemble learning is to combine multiple weaker learners to build a more powerful model. Model ensembles are one of the most powerful types of models on machine learning. This comes at the cost of increased algorithmic and model complexity though [9]. Ensemble methods are used often with decision tree models. Commonly used ensemble techniques include bootstrap aggregating (bagging) and boosting.

Bagging is a simple and highly effective ensemble method that creates diverse models on different random samples of the dataset. The samples are taken uniformly with replacement so that the sample can contain duplicates and some parts of the original data can be missing from the bootstrap samples. A model is built on each of these samples and their average is taken as the prediction result. Bagging reduces variance so it works well with models that are sensitive to small variations in the data like decision trees. The famous random forest method by Leo Breiman utilizes bagging with CART trees. [17] [9]

In boosting, unlike bagging, the weak learners evolve over time and the members cast a weighted vote instead of simply generating random predictors and averaging their result [17]. In boosting each model in the sequence is trained to perform better on the residuals of the previous model [9]. Boosting appears to perform better than bagging on most problems, has thus become the preferred choice of method [17].

3.5.1 Random forest

A random forest is an ensemble of CART trees each trained on a random sample of the training data with a random subset of features [4]. The average of the predictions of these trees is used as the prediction result. The problem with decision trees is that they tend to overfit on the data, meaning that they also learn the noise, which makes them unable to perform well outside of the training data. Because in random forests each tree is trained from a different subsample of the original dataset, they are less prone to overfitting than individual trees as no single tree can learn from all of the instances [15].

The random forest algorithm itself is quite simple. It starts by first by taking a random bootstrap sample with replacement from the dataset and then selecting a random subset of features to reduce the dimensionality of the sample. Next, an unpruned CART tree is trained on this bootstrap sample. This process is repeated for the desired ensemble size. The predicted value of an unknown instance x is obtained by simply taking a majority vote in the case of classification or an average in the case of regression of the trees' predictions in the ensemble:

$$\hat{f}_{rf}^{B}(x) = \frac{1}{B}\sum_{b=1}^{B} T_b(x) \tag{3.25}$$

where b is a bootstrap sample and T_b is a tree trained on sample b [9].

3.5.2 Gradient tree boosting

Gradient boosting consecutively fits new weak models, typically CART trees, to correct the error of the previous model. It combines gradient-based optimization which utilizes gradient computations to minimize a model's loss function and boosting [12]. Even though gradient boosting reduces both variance and bias, it does not solve the problems decision trees have with overfitting.

Using a training set $(\mathbf{x}_i, y_i)_{n=1}^N$, the goal of gradient boosting is to find an approximation for a function f that minimizes some arbitrary loss function $\Psi(y,f)$. This is done by additively expanding the form

$$\hat{F}(\mathbf{x}) = \sum_{m=0}^{M} \hat{F}_m(\mathbf{x}) = \sum_{m=0}^{M} \beta_m \phi_m(\mathbf{x}) \tag{3.26}$$

where $\phi_m(\mathbf{x})$ are the base learning models, $\hat{F}_0$ is the initial guess and $\{\hat{F}_i\}_{i=1}^M$ are incremental functions or "boosts" and β_m are expansion coefficients (weight) [12] [13].

The coefficients $\{\beta_m\}_0^M$ and the models $\{\phi_m\}_0^M$ are jointly fit on the training data in a stage-wise manner. First, we start with an initial guess:

$$\hat{F}_0(\mathbf{x}) = arg\ min_\beta \sum_{i=1}^{N} \Psi(y_i, \beta). \tag{3.27}$$

Then for each m from 1 to M we update the model as:

$$(\beta_m, \phi_m) = arg\ min_{\beta,\phi} \sum_{i=1}^{N} \Psi(y_i, \hat{F}_{m-1}(\mathbf{x}_i) + \beta\phi_m(\mathbf{x}_i)) \tag{3.28}$$

and

$$\hat{F}_m(\mathbf{x}) = \hat{F}_{m-1}(\mathbf{x}) + \beta_m\phi_m(\mathbf{x}). \tag{3.29}$$

To simplify the calculation of an optimal base model ϕ in each step, the function $\phi_m(\mathbf{x})$ is trained on pseudo-residuals obtained by gradient descent instead of actual residuals. The pseudo residuals can be calculated by negative gradient [12]:

$$\hat{y}_{im} = -\hat{g}_{im} = -\left[\frac{\partial\Psi(y_i, \hat{F}(\mathbf{x}_i))}{\partial\hat{F}(\mathbf{x}_i)}\right]_{\hat{F}(\mathbf{x})=\hat{F}_{m-1}(\mathbf{x})}, \ i = 1, ..., N. \tag{3.30}$$

The function $\phi_m(\mathbf{x})$ is fitted by least squares

$$\phi_m = arg\ min_{\phi,\beta} \sum_{i=1}^{N} (\hat{y}_{im} - \beta\phi(\mathbf{x}_i))^2 \tag{3.31}$$

and then after finding $\phi_m(\mathbf{x})$, the value of step length β_m is determined by line search:

$$\beta_m = arg\,min_\beta \sum_{i=1}^{N} \Psi(y_i, \hat{F}_{m-1}(\mathbf{x}_i) + \beta\phi_m(\mathbf{x}_i)). \tag{3.32}$$

This way we can get around the difficult optimization problem of Equation (3.28) by solving it for the arbitrary loss function Ψ using this two-step procedure [12]. The ”step” taken at each iteration m is then given by:

$$\hat{f}_m(\mathbf{x}) = v\beta_m\phi_m(\mathbf{x}). \tag{3.33}$$

The regularization parameter $0 < v \leq 1$ controls the learning rate of the procedure. Lowering the learning rate increases the generalization ability of the model at the cost of increased computational time.

Gradient tree boosting is a specialized case of gradient boosting where the base learner $T(\mathbf{x}, \mathbf{a})$ is a regression tree with L leaf nodes. At each iteration m, a regression tree partitions the the input space into L-disjoint regions $\{R_{lm}\}_1^L$ and predicts a separate constant value in each one [13]

$$T(\mathbf{x}, \{R_{lm}\}_1^L) = \sum_{l=1}^{L} \hat{y}_{lm} I) \tag{3.34}$$

where $\hat{y}_{lm} = \frac{1}{n_{lm}} \sum_{\mathbf{x}_i \epsilon R_{lm}} \hat{y}_{im}$ is the mean of 3.30 in each region R_{lm}. I is a function that equals one when $\mathbf{x} \epsilon R_{lm}$ and zero otherwise.

When using regression trees, the optimal solution to Equation (3.32) can be solved separately in each region R_{lm} defined by the corresponding leaf node l of the mth tree. The prediction of the tree in Equation (3.34) is a constant value $\bar{y}_{lm}$ within each region R_{lm}. The solution for Equation (3.32) reduces to: [13]

$$\hat{w}_{lm} = arg\,min_{\beta_l} \sum_{\mathbf{x}_i \epsilon R_{lm}} \Psi(y_i, \hat{F}_{m-1}(\mathbf{x}_i) + \hat{w}_l), \; l = 1, ..., L \tag{3.35}$$

and the update rule becomes

$$\hat{F}_m(\mathbf{x}) = \hat{F}_{m-1}(\mathbf{x}) + v\hat{w}_{lm} 1(\mathbf{x} \epsilon R_{lm}) \tag{3.36}$$

where the regularization parameter v is the learning rate. Another common way of regularization in gradient boosting is subsampling, in which the model is fit on a subsample drawn from the training data at random without replacement. Subsampling adds randomness to the procedure and helps to prevent overfitting. Gradient boosting utilizing subsampling is known as stochastic gradient boosting [13].

3.5.3 Extreme gradient boosting

XGBoost (eXtreme Gradient Boosting) is currently one of the most popular machine learning algorithms. It is a scalable and optimized implementation of the gradient boosting algorithm. XGBoost is known to be both faster and less prone to overfitting than traditional gradient boosting [8].

Compared to gradient boosting, XGBoost attempts to solve the optimization problem 3.28 using Newton's boosting method. In addition to gradient, we need the Hessian [8] [27]

$$\hat{h}_{im} = \left[\frac{\partial^2 \Psi(y_i, \hat{F}(\mathbf{x}_i))}{\partial^2 \hat{F}(\mathbf{x}_i^2)} \right]_{\hat{F}(\mathbf{x})=\hat{F}_{m-1}(\mathbf{x})}, \; i = 1, ..., N. \tag{3.37}$$

Gradient boosting learns the model by fitting it to the negative gradient using least-squares. Newton boosting on the other hand learns the model by fitting the model to the negative gradient scaled by the Hessian using weighted least squares regression. The weights are defined by the Hessian $\{\hat{h}_{im}\}_{i=1}^{N}$ [27]. Now, the basis function ϕ can be obtained by

$$\begin{aligned} \phi_m &= arg\ min_{\phi} \sum_{i=1}^{N} (\hat{g}_{im}\phi(\mathbf{x}_i) + \frac{1}{2}\hat{h}_{im}\phi(\mathbf{x}_i)^2) \\ &= arg\ min_{\phi} \sum_{i=1}^{N} \frac{1}{2}\hat{h}_{im}\Big[(-\frac{\hat{g}_{im}}{\hat{h}_{im}}) - \phi(\mathbf{x}_i)\Big]^2. \end{aligned} \tag{3.38}$$

Newton's method naturally has a step length of *1* which leads to the step being

$$\hat{f}_m(\mathbf{x}) = v\phi_m(\mathbf{x}). \tag{3.39}$$

If the base learner is a tree, it can be found with:

$$T(\mathbf{x}, \{R_{lm}\}_1^L) = \sum_{l=1}^{L} \hat{w}_{lm} I. \tag{3.40}$$

The weights $\hat{w}$ are given directly by

$$\hat{w}_{lm} = -\frac{G_{lm}}{H_{lm}}, \; l = 1, ..., L \tag{3.41}$$

where $G_{lm} = \sum_{\mathbf{x}\epsilon R_{im}} \hat{g}_{im}$ is the sum of gradients in each region R_{lm} and $H_{lm} = \sum_{\mathbf{x}\epsilon R_{im}} \hat{h}_{im}$ is the sum of Hessians in each region R_{lm}. I is a function that equals one when $\mathbf{x}\epsilon R_{lm}$ and zero otherwise.

Newton boosting practically solves the optimization problem 3.28 by using a second order approximation of the loss function also optimizing the weights at the same time. This gives a more accurate estimation of the tree but does not readjust the weights after learning the tree. This saves time though by skipping the line search in the weight adjustment procedure.

XGBoost additionally offers a possibility to penalize the complexity of trees by adding a regularization component to the objective function [8]:

$$\sum_{i=1}^{N}(\hat{g}_{im}\phi(\mathbf{x}_i) + \frac{1}{2}\hat{h}_{im}\phi(\mathbf{x}_i)^2) + \Omega(\phi) \tag{3.42}$$

where Ω can be written as [8] [27]:

$$\Omega(\phi) = \sum_{m=1}^{M}(\gamma T_m + \frac{1}{2}\lambda\|w_m\|_2^2 + \alpha\|w_m\|_1) \tag{3.43}$$

where γ is a cost-complexity criterion for the tree T_m. The last two terms are *l2* and *l1* regularizations for the leaf weights respectively. Just like stochastic gradient boosting, XGBoost also utilizes subsampling for both columns and rows. [27]

XGBoost also has the ability to speed up its running time by running parallel computations. The build processes for trees have to be run sequentially but instead XGBoost opts to use parallel computing to speed up the creation of branches within individual trees. This way the trees can be built much quicker and the training process gets sped up by a large margin by using all of the system's CPU cores.

3.6 Model evaluation

Model evaluation is done to find the model that best fits the data and how well the model will work in future applications. The two methods of model validation are hold-out and cross-validation. After validation, the model is tuned to get the best possible results on the evaluation metric of choice. Evaluation metrics for regression problems include mean absolute error, root mean squared error and the coefficient of determination or ”R squared” (R^2).

3.6.1 Validation

After building a model and validating it using the validation set, feedback is received on how the model performed given by the evaluation metric of choice. This feedback is used to tune the model to get the best possible results. Final assessment of the model is done on the testing set, a part of the data the model has not seen before. This method is called hold-out.

In cross validation the dataset is divided into k disjoint folds of equal size. Then a model is build on each fold using it as the training data and the rest of the dataset for validation. This way we are able to get an unbiased estimate of the model's performance even with a dataset of smaller size at the cost of increased computational time.

3.6.2 Evaluation metrics

For regression problems the most common metrics to measure the performance of the model are mean absolute error (MAE), root mean squared error (RMSE) and R squared (R^2). MAE is the average of absolute differences between predicted and observed values:

$$MAE = \frac{1}{n}\sum_{j=1}^{n} |y_j - \hat{y}_j|. \tag{3.44}$$

MAE is a linear score, so each error has the same weight. It is also very easy to understand and compute.

RMSE is the quadratic mean of the difference between the predicted and observed values:

$$RMSE = \sqrt{\frac{1}{n}\sum_{j=1}^{n} (y_j - \hat{y}_j)^2}. \tag{3.45}$$

As RMSE squares the errors before averaging, it gives a relatively high weight for large errors [36] [6]. Thus, RMSE is preferred when one requires particular sensitivity to large errors [3].

The performance of a model can also be measured through the use of the coefficient of determination (R^2). It tells how well the model explains the problem, meaning how large portion of the data the model is able to explain. This is measured on scale from 0 to 1. The coefficient of determination is defined as

$$R^2 = 1 - \frac{SS_{Res}}{SS_T} \tag{3.46}$$

where

$$SS_T = \mathbf{y}^T\mathbf{y} - \frac{\left(\sum_{i=1}^{n} y_i\right)^2}{n} \tag{3.47}$$

is a measure of variability in y without considering the effect of x and

$$SS_{Res} = \mathbf{y}^T\mathbf{y} - \hat{\boldsymbol{\beta}}^T\mathbf{X}^T\mathbf{y} \tag{3.48}$$

is a measure of the variability remaining in y after x has been considered. As $0 \leq SS_{Res} \leq SS_T$ then also $0 \leq R^2 \leq 1$. This means that models with R^2 values closer to 1 are better in explaning the phenomenon. The problem with R^2 is that increasing the amount of variables also increases the value of R^2 [25] so it cannot be used to compare models with different amount of variables.

4. EXPERIMENTS

The goal was to implement a system that allows customers to evaluate housing unit asking price using the machine learning techniques introduced in chapter 3. It was important for the application to be easily maintainable and very simple for the customers (end-users) to use. The predictor had to also look trustworthy from the viewpoint of the customer. The data used for training the predictor originated from sales advertisements from Etuovi.com housing marketplace.

In this chapter we first go through the dataset used and select the feature selection methods to select the features used for the learning. Second, the results from different predictors are compared and their viability for production is assessed. One method is chosen for further development. The performances of the predictors are compared with different feature selection methods, using different error measures and against both the total price and price per square meter. As XGBoost is famous for its efficiency in solving diverse machine learning problems and currently being the top algorithm in various machine learning competitions, it is expected to outperform the older and and more simple algorithms.

We go through the regression algorithms introduced in chapter 3, starting with linear regression, followed by regression trees, random forest, gradient boosting machine and finally XGBoost. The implementations used are mostly from Python's Scikit-learn module and from Python package of H2O machine learning platform. Finally we define and train the models on the proposed dataset using multiple machine learning algorithms and assess their performances.

4.1 Data and preprocessing

The dataset consisted of housing unit sales advertisements from Etuovi.com from June 2017 until June 2018. The dataset was exported from Etuovi.com's data warehouse. The structure of the dataset was as follows: Each sales advertisement was considered one instance and the attributes of the property described by the seller were used as explanatory variables for the selling price and price per square meter. A sample of the used dataset is in table 4.1.

The following features were considered to have an impact on the price/price per square meter of housing unit

- Location by postal code (categorical) and coordinates (numerical)
- Age calculated from construction year (numerical)

Table 4.1. Dataset sample.

House type	Room count	Postal code	Lat	Lon	Condition	Living area (m^2)	Age	Plot area (m^2)	Plot holding type	Sauna	Price per square meter (EUR)	Total price (EUR)
Detached house	5h+	15900	60.9	25.6	Good	230	12	1027	Own	1	2434.78	560000
Row house	4h	37150	61.4	23.5	Not rated	120	48	1374	Own	1	1654.17	198500
Apartment house	2h	01710	60.2	24.8	Good	47.5	45	2886	Own	0	2729.47	129650
Apartment house	3h	47400	60.9	26.3	Not rated	71	86	3058	Rental	1	1900	134900
Detached house	5h	14200	60.9	24.6	Satisfactory	128	31	1585	Own	1	1000	128000
Apartment house	2h	00200	60.1	24.8	Satisfactory	50	77	1998.8	Own	0	5260	263000
Apartment house	2h	05800	60.6	24.8	Satisfactory	50	49	4822	Own	0	1940	97000
Row house	5h+	70800	62.8	27.6	Good	137	28	3185	Rental	1	2525.55	346000
Apartment house	1h	00380	60.2	24.8	Satisfactory	35	18	2632	Own	1	5285.71	185000
Detached house	3h	70910	62.9	27.6	Good	92	48	611	Rental	1	1793.48	165000

- Living area (numerical)
- Room count (categorical)
- House type (categorical)
- Condition (categorical)
- Sauna (binary)
- Plot area (area of the land, numerical)
- Plot holding type (binary).

This includes all of the variables used by the previous system, ARVO, as well as some new variables to consider.

For the case of linear regression we also constructed polynomial features of the dataset to try to improve the performance of the model. The dataset was also reduced using the following rules to remove irregularities and erroneous samples: Construction year had to be later than the year 1900, living area had to be between ten and one thousand square meters, house type had to be one of the following: apartment house, row house, detached house, semi-detached house, balcony-access-block, separate house. Open bidding advertisements, living right homes and new properties were not included in the dataset. The price range was from twenty thousand to two million euros.

Other features were also considered for the task. Room structure was dropped due to lack of consistency in the observations. This is due to the field in the advertisement form being free-text. Distance to amenities and services like public transport, supermarket, school etc. but this was dropped due to lack of data and inconsistencies. Floor level was also considered as one of the variables to be used but low quality of data caused by misinterpretations with using the said datafield made it unusable. The sauna-variable seemed to have many missing observations at the rate of almost 40% so its effect on improving the predictive capabilities of a model is probably diminished. Some of the categorical variables contained low amounts of missing values and those were filled with an ”unknown”-class. The continuous variables did not contain any missing values. Utilizing past renovations was also considered, but the idea was scrapped because the renovation data is in freeform text and working with it would be too resource intensive.

Categorical variables, room count, house type, floor level and condition had to be encoded for them to be used by some of the algorithms. Here one-hot encoding was used to create a dummy variable of values $(0, 1)$ for each of the possible classes in a categorical variable when needed.

4.2 Feature selection

For linear regression, using an exhaustive search to find the optimal feature set cost too much and thus instead both forward selection and backwards elimination were used. These methods were used to select or drop the feature that most improved the model. When none of the variables seemed to reduce the error any further, the least harmful variable was selected until exhaustion. Then the subset with the smallest error was chosen. The selection was not stopped at the point where no more improvement was seen, because the most optimal solution can lie beyond this as it’s unknown to us how the combinations of the variables affect the predictive power of the model. This is also one of the weaknesses of these methods. As another approach, LASSO regression was also applied to the data in order to suppress the least important features. In addition to the previously listed features, polynomial feature engineering was applied to the numerical features so that the regressor also utilized squared versions of the numerical features as well as their cross products.

Because constructing a tree model involves ranking the features by how much they contribute to the result, feature selection was not as important as it is with other types of models. For tree models, backwards elimination was applied by utilizing the feature importance property of the models to improve performance. This was done by removing the least important features one by one until the performance of the model improved no more. But due to the nature of the task it was important to get rid of the least important features to make the end application as simple as possible to use. This was handled after assessing the performance of every model type. For example, if a feature makes the predictor only slightly better, it will be considered for removal to enhance the user experience of the end application.

4.3 Validation

The models were validated using hold-out. The training dataset of 109000 observations was split into 70% training data and 30% validation data at random. This seemed sufficient considering the amount of data available

A separate test set made from the newest housing advertisements published after June 2018 was used for the final assessment. This should give a better picture of the performance of the model as it can be seen as a simulation of the use of the model in production. The size of the testing dataset was around 30000 rows.

4.4 Linear regression

The first of the machine learning algorithms was ordinary least squares. This Linear regression method was discussed in section 3.3. In short, the aim of the model is to choose the parameters for a set of explanatory variables so that the sum of squares between the difference of them and the response variable is minimized.

The implementation of ordinary least squares used here was from Python's Scikit-learn-module (sklearn.linear_model.LinearRegression). For ordinary least squares, the feature selection was done by utilizing both forward selection and backwards elimination. Another linear regression method tested here is the LASSO regression (sklearn.linear_model.Lasso). The performances of the models were evaluated using the mean absolute error (MAE) and the coefficient of determination (R^2). For the linear regression models, some extra features were constructed from the original feature set. This included squared features and cross products generated for the numerical variable. The whole feature set considered for linear regression being the original set with the addition of second order polynomials for age, living area and plot area (the area of the land) as well as the cross products of these variables.

The variables chosen with forward selection were:

Total price:

- Living area
- Age
- Latitude
- Age^2
- House type
- $Living\ area \times Age$
- Condition
- Sauna
- $Living\ area \times Plot\ area$
- Postal code
- Longitude

Price per square meter:

- Age
- Latitude
- House type
- Age^2
- Postal code

in that order. This resulted in a MAE of 42380 and R^2 of 0.681 when predicting total price and a MAE of 350.3 and R^2 of 0.763 when predicting price per square meter. When predicting price per square meter, the method selected less than half of the number of features it had selected when predicting total price. All of the features selected when predicting price per square meter were also selected when predicting total price. The inclusion of age squared indicates a curvilinear relationship. The inclusion of this variable allowed the capturing of the fact that very old housing units can have quite high cost. It is known that the cheapest housing units are those built in 1960's and 1970's as they are aging quickly, were sometimes built with poor quality and are often in need of renovation. The older, already renovated housing units have a higher price tag because of this.

With backwards elimination, the dropped variables were:

Total price:

- $Plot\ area^2$
- Room count
- $Living\ area^2$
- Longitude

Price per square meter:

- $Plot\ area^2$
- Latitude

in that order. The MAE and R^2 were 42430 and 0.688 respectively when predicting the total price and 318.8 and 0.805 when predicting price per square meter. It did not take many variables to reach the lowest error rate. Interestingly in both cases the eliminator dropped a coordinate variable which was not expected as they were selected earlier by the forward selection method. As the elimination stopped when these variables were dropped, it seems like the most optimal set lies somewhere else and cannot be found by backwards elimination.

Using LASSO regression resulted in a MAE of 41820 and R^2 of 0.696 when predicting total price and a MAE of 315.8 and R^2 of 0.811 when predicting price per square meter. Finding the optimal λ value for LASSO was done by first taking values evenly on a logarithmic scale from 10^-5 to 10^5 using numpy.logspace. When the right magnitude was found, the value was iterated further using numpy.linspace until the optimal value was found. Using LASSO gave a lower error rate than both forward selection and backwards elimination. When looking at the feature selection results, there were some interesting findings. First, the age squared -feature had a positive weight. This verifies the fact that cheapest housing units are those built around 1960's and 1970's. All the other polynomial and cross product features had almost zero weight. The LASSO completely zeroed the weights for semi-detached houses and some remote postal codes. The amount of observations for these variables was already very low to begin with. From all of the feature selection methods for linear regression, the LASSO had the best performance.

4.5 Regression tree

The second method was decision tree regression that was discussed in section 3.4. The goal of this model is to successively split the source data into smaller partitions using threshold-based rules. The terminal nodes of the tree are called leaf nodes, which contain a simple predictor (e.g. mean, least squares) to make a prediction.

In order to fit a regression tree, Scikit-learn's implementation of the CART-algorithm (sklearn.tree.DecisionTreeRegressor) was used. Feature selection is not very useful in tree models when building as accurate model as possible but still useful when removing the variables that contribute least to the tree's predictive abilities. The model was first fit with out of the box. After the first fit, hyperparameters of the model were tuned to get a more optimized model. The feature importances can be seen in Figure 4.1. The hyperparameters used to tune the tree model were:

- Maximum depth of the tree (32)
- Minimum number of samples for splitting (14)
- Minimum samples per leaf (14).

The hyperparameter tuning was done by hand. It proved difficult trying to lower the error rate as the tree overfitted very easily as it grew deeper. The relationships between variables seemed too complex to express using a single tree without overfitting.

After tuning the hyperparameters, the predictive capabilities of the model improved a little. The result was a MAE of 42330 and R^2 of 0.696 when predicting the total price and 354.8 and 0.781 when predicting price per square meter. The regression tree model's performance was worse than linear regression in both cases.

The decision tree regressor ranked house type as the most important feature followed by latitude, room count and living area. Interestingly, the importance of some of the features

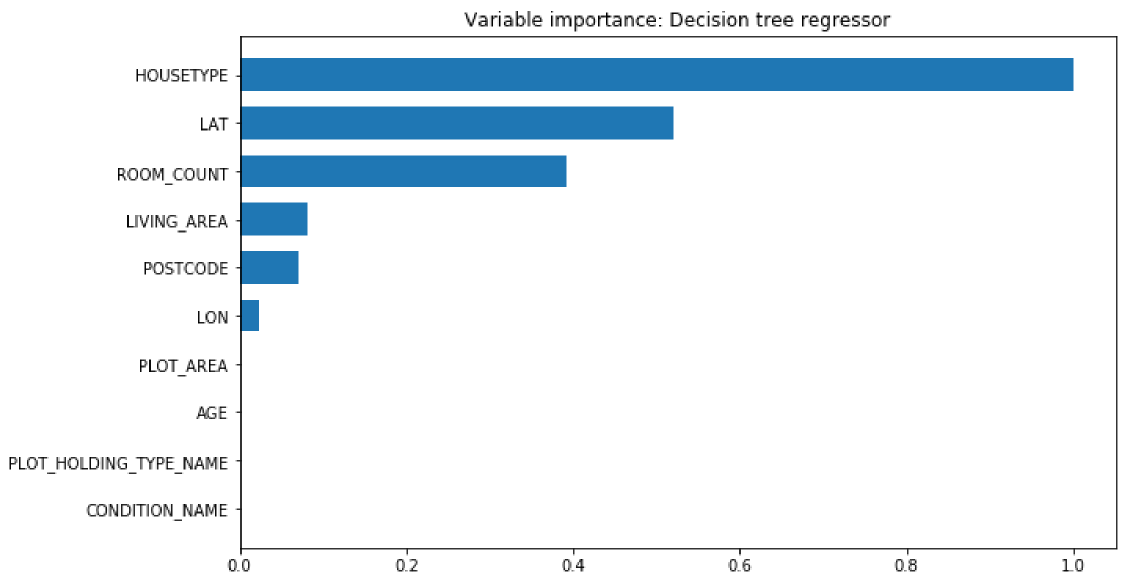


Figure 4.1. *Variable importances of decision tree regressor (price per square meter).*

was actually really close to zero. Even though importance was extremely low, removing these features greatly increased the error rate of the model so they were kept in.

4.6 Random forest

The next model was random forest regressor discussed in 3.5.1. The model is a bootstrap ensemble of regression trees trained on a random sample of the dataset. Then the average of the whole forest of trees is used as the prediction result.

The implementation of the random forest algorithm used here is from the H2O -machine learning platform (H2O.estimators.RandomForestEstimator). As feature selection does not play as big of a role when building a tree based model like a random forest regressor, we started directly by first building the model out-of the box and then started the tuning to get the optimal configuration of hyperparameters. Random forest is known for being easy to tune and should perform quite well with most problems almost by default.

The common hyperparameters utilized by random forests are:

- Number of trees (477)
- Maximum depth (Unlimited)
- Minimum leaf size (1)
- Sampling of rows and columns. (0.632, 1).

The hyperparameter tuning was started by making the number of trees unlimited, but stopping the training when no further improvement is seen. The maximum depth of the trees is left unlimited and minimum leaf size left set at one so individual trees are able to grow without any pruning. Sampling is applied to rows and columns by iterating the sample size values from 0.4 to 1. The improvement in the model with each new tree can be seen in Figure 4.2. The feature importances are shown in Figure 4.3.

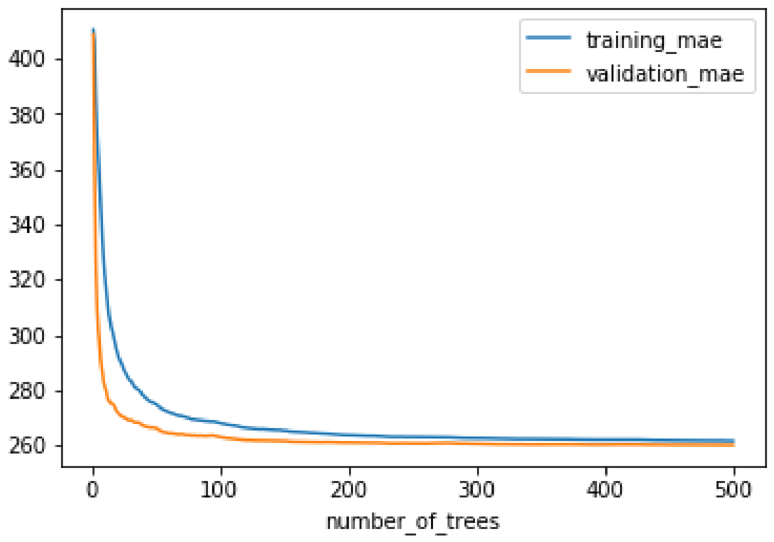


Figure 4.2. *MAE of the tuned random forest model (price per square meter).*

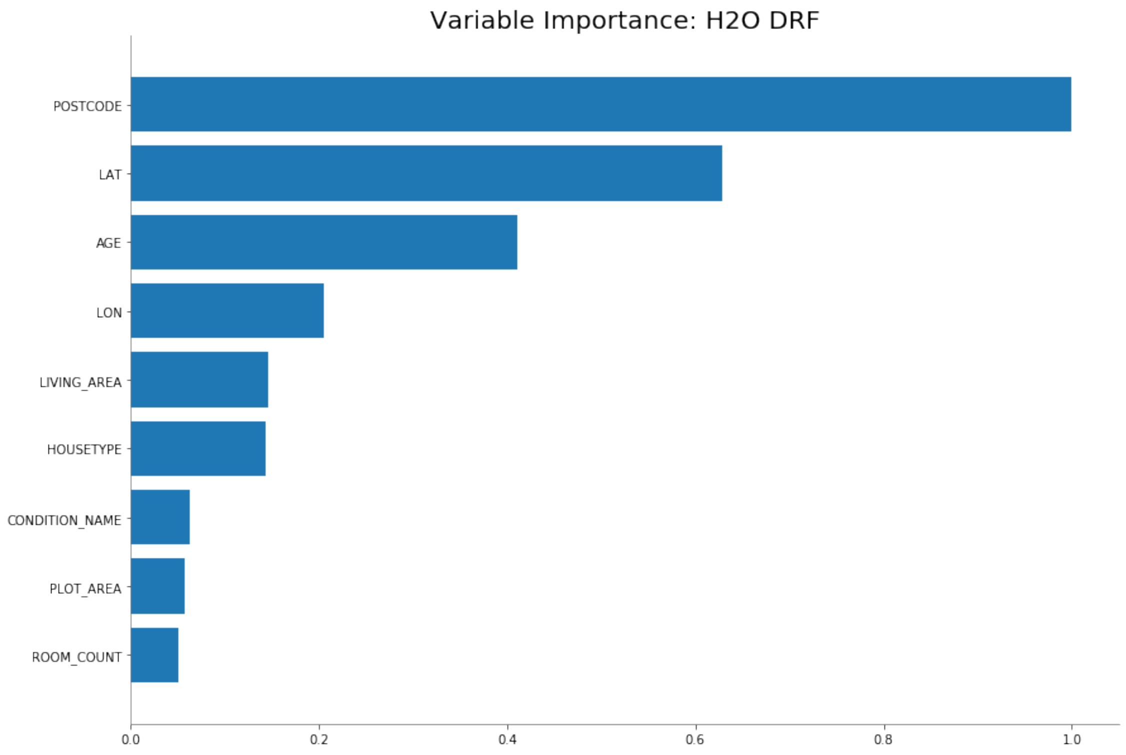


Figure 4.3. *Variable importances of random forest (price per square meter).*

Even though changing the hyperparameters from their default values did not seem to substantially affect the performance, a significant improvement is seen here over the more simple models. Dropping the least important features, sauna flag and plot holding type, also improved the performance slightly. The MAE and R^2 score of the tuned model were 26470 and 0.880 when predicting the total price and 262.9 and 0.921 when predicting the price per square meter.

The model ranked location by postal code as the number one feature followed by latitude, age and longitude. Interestingly, the importance of house type was very low compared to it's ranking in the case of a single decision tree. With random forest, overfitting was no longer an issue like when using a single decision tree. Even though individual trees can be very deep and learn complex patterns, the problem of overfitting is overcome with the facts that the trees are trained on different sets of data and the result is an average of all the deep decision trees. The substantial increase in performance compared to the previous models shows the effectiveness of model ensembles.

4.7 Gradient boosting machine

Gradient boosting is an ensemble model introduced in 3.5.2. The model is a boosting ensemble of sequentially trained regression trees where each tree tries to correct the error of its predecessor.

Because boosting includes building sequential models, it is not normally possible to utilize parallelism to speed up the training process. H2O's implementation can however use multiple cores to train a single tree. This is the same method XGBoost uses to make use of multiple cores. The H2O's implementation (H2O.estimators.GradientBoostingEstimator) is used as of this reason. Same as with other tree based models, the separate feature selection is not considered when building the model. Being a more complicated algorithm, gradient boosting requires careful hyperparameter tuning to achieve the best performance.

The hyperparameters considered for the gradient boosting model are:

- Number of trees (1700)
- Maximum depth (32)
- Minimum leaf size (10)
- Learning rate (0.008)
- Sampling of rows and columns. (0.4, 0.6).

The number of trees is found the same way as with random forest by setting the number of trees to unlimited and controlling the training using a stopping metric to stop the training process when no more improvement is seen between iterations. The rest of the parameters are tuned by hand in the following way. First the tree specific parameters are tuned in the order of higher impact parameters first (Max depth, then Min leaf size). Then, learning rate is reduced and tree-specific parameters tuned again until no significant improvement is seen. Lowering learning rate causes the number of trees to increase in order for the model to converge which increases the computational power requirements to iterate the optimal parameters in a reasonable amount of time. The improvement in the model with each new tree can be seen in Figure 4.4 and the feature importances in 4.5. Sampling of both rows and columns is increased as the number of trees increases to control overfitting and thus increase the predictive capabilities of the model.

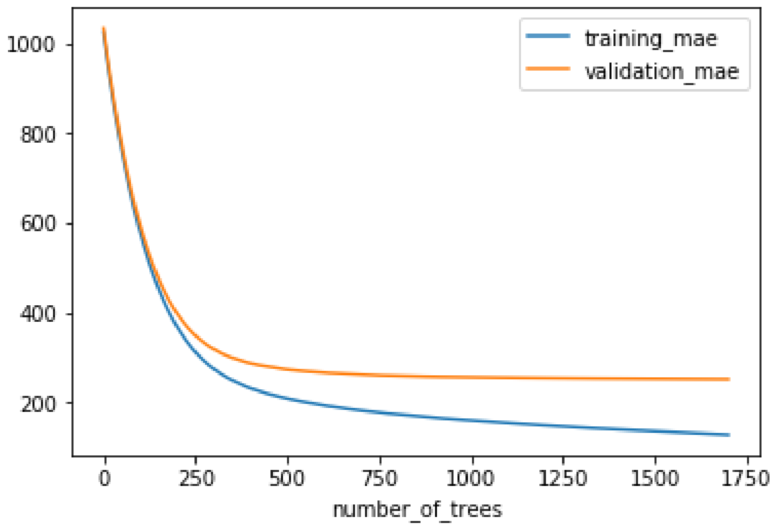


Figure 4.4. *MAE of the tuned gradient boosting model (price per square meter).*

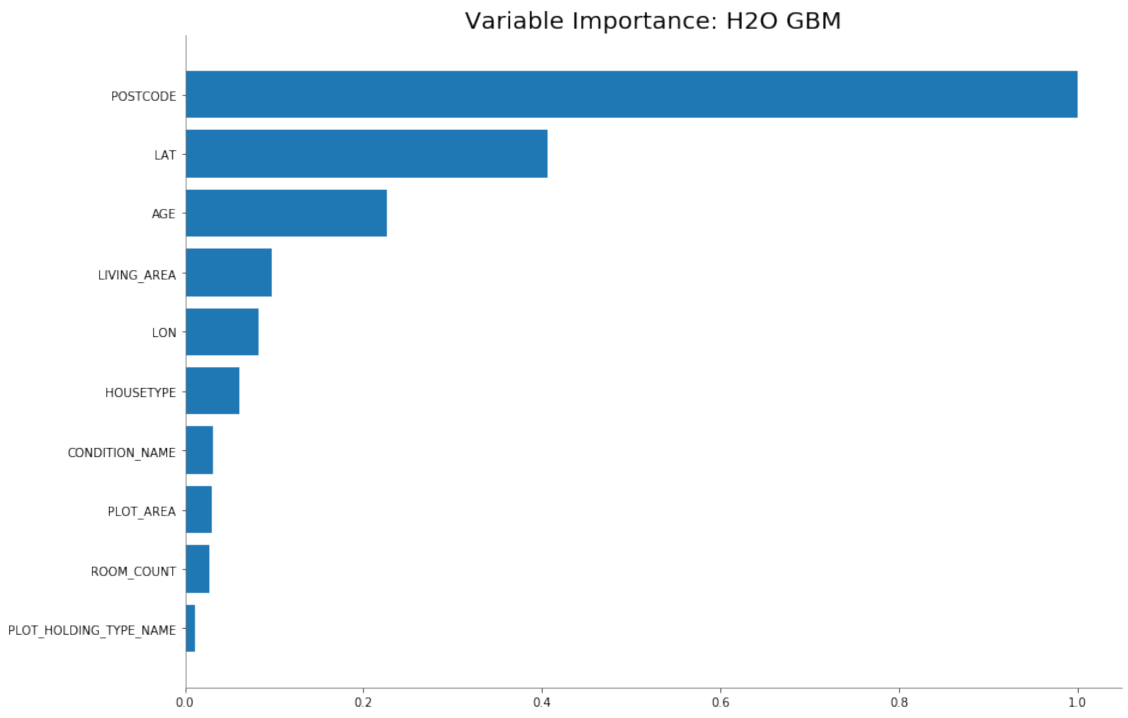


Figure 4.5. *Variable importances of gradient boosting model (price per square meter).*

The model did not perform so well out of the box as random forest, but with proper tuning we were able to achieve a slight improvement over random forest in predictive power. The error was decreased a little further by dropping the least valuable features similar to random forest. The MAE and R^2 score of the tuned model were 24420 and 0.886 when predicting the total price and 252.0 and 0.929 when predicting the price per square meter.

The feature importances show that the location contributes most to the model with postal code at the top and latitude following next. The importances look pretty similar as in the case of random forest with house type being again quite low in the ranking.

4.8 Extreme gradient boosting

XGBoost (3.5.3) is an optimized version of the gradient boosting algorithm. Same as with the other tree based models, feature selection was not considered here as the model is able to select the features it sees the most important. Similarly to gradient boosting, the algorithm requires careful tuning of its hyperparameters to achieve the best results. The implementation used here was from H2O (H2O.estimators.H2OXGBoostEstimator).

The hyperparameters for XGBoost are the same as those for gradient boosting:

- Number of trees (1000)
- Maximum depth (12)
- Minimum leaf size (10)
- Learning rate (0.03)
- Sampling of rows and columns. (0.8, 0.8)

Tuning the parameters works in a similar way to gradient boosting in 4.7. As XGBoost includes calculating the second order derivative of the loss function, the time to build a similar size model is increased. For comparison, it took the training instance less than ten minutes to build the plain gradient boosting model consisting of 1700 trees with a maximum depth of 32 while an XGBoost model of one thousand trees with maximum depth of 12 took the same machine over an hour to train. The slower training time is explained by the fact that XGBoost has to also calculate the hessian in addition to the gradient. On the other hand, this notably less complex and smaller model is able to achieve even better results than plain gradient boosting. MAE of the model after adding each tree can be seen in Figures 4.6 and the feature importances are presented in Figure 4.7.

Also with XGBoost, the out of the box performace was not even close to the optimal result. But with some tuning the algorithm ended up having the highest predictive power for this particular task out of the options proposed. The result was a MAE of 22350 and R^2 of 0.903 when predicting the total price and 233.5 and 0.932 when predicting price per square meter. The tuning was not optimized completely due to limitations on computational power available. It seemed like the error could be decreased a little further but at the current state each iteration took hours and it was already proven that the model performed the best out

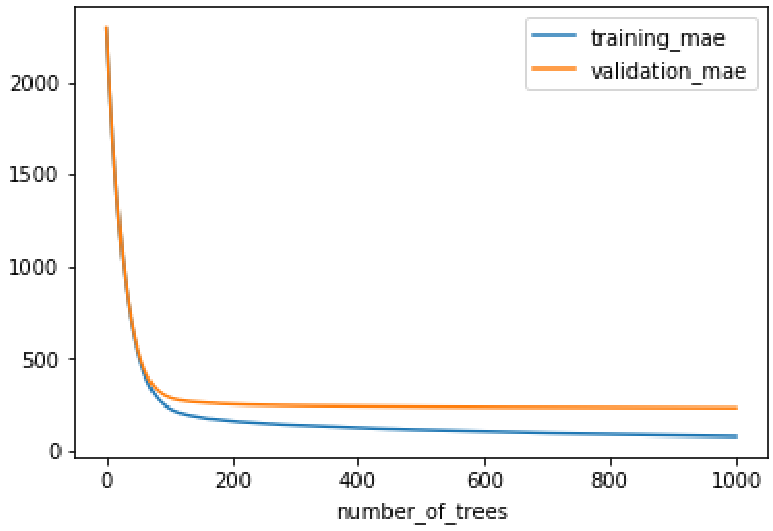


Figure 4.6. *MAE of the tuned XGBoost model (price per square meter).*

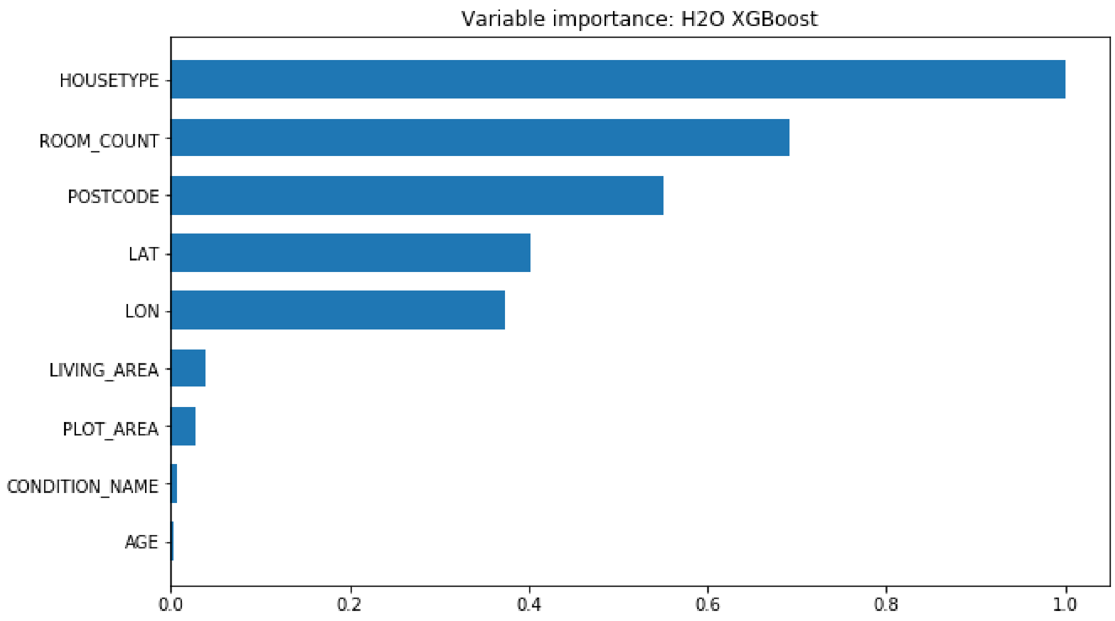


Figure 4.7. *Variable importances of XGBoost model (price per square meter).*

of the options proposed. The performance of the model can still be improved slightly in the production environment.

The variable importances look somewhat different from random forest and gradient boosting. House type was actually the most important feature followed by room count. After these came all three location variables in the order of postal code, latitude and longitude. Rest of the variables had substantially lower importances. The age variable has a low importance, similar to the case of the single regression tree. Because all the buildings in an area are usually built around the same time, the location features also contain at least some of the information of the age feature. Also, the room count and living area both describe the size of the housing unit.

4.9 Summary of performances

After experimenting with the models, it was found out that extreme gradient boosting had the highest performance on this particular task as expected with an R^2 score of 0.932 when predicting price per square meter. The MAE of all the models can be see in Figures 4.8 and 4.9.

It is noticeable how much better the tree ensembles performed compared to linear regression or the regression tree. The R^2-scores can be seen in Figures 4.10 and 4.11. The results show that predicting price per square meter tends to give higher values for the coefficient of determination. This might have something to do with the normalization of the response.

The linear regression models saw a huge improvement in performance with feature selection. The decision tree only improved minimally as it overfitted extremely easily so not much could be done to it's hyperparameters. It should be noted that random forest performed best out of the box and only improved minimally after tuning. This can make it a good baseline model to compare other solutions against.

A significant difference in performance can be seen between the decision tree regressor and the random forest model when going from a simple learner to an ensemble of multiple ones. This truly shows the effectiveness of ensembles in forming a powerful predictor using multiple weak ones. The performance of XGBoost here can be improved further by more hyperparameter tuning as the limitations on computational capacity became an obstacle. As XGBoost had the highest performance out of the options, it was chosen for the implementation phase using the same explanatory variables as in the comparison experiment and price per square meter as the response variable as it produced a higher value for R^2 than the total price.

The XGBoost model seemed quite trustworthy when tested. Over half of the test samples had an error of less than one percent while 80% of the samples had an error of less than ten percent. Less than four percent of the test samples had an error of 25% or more. The model is not perfect by any means but it should be sufficient for giving consumers an asking price estimation.

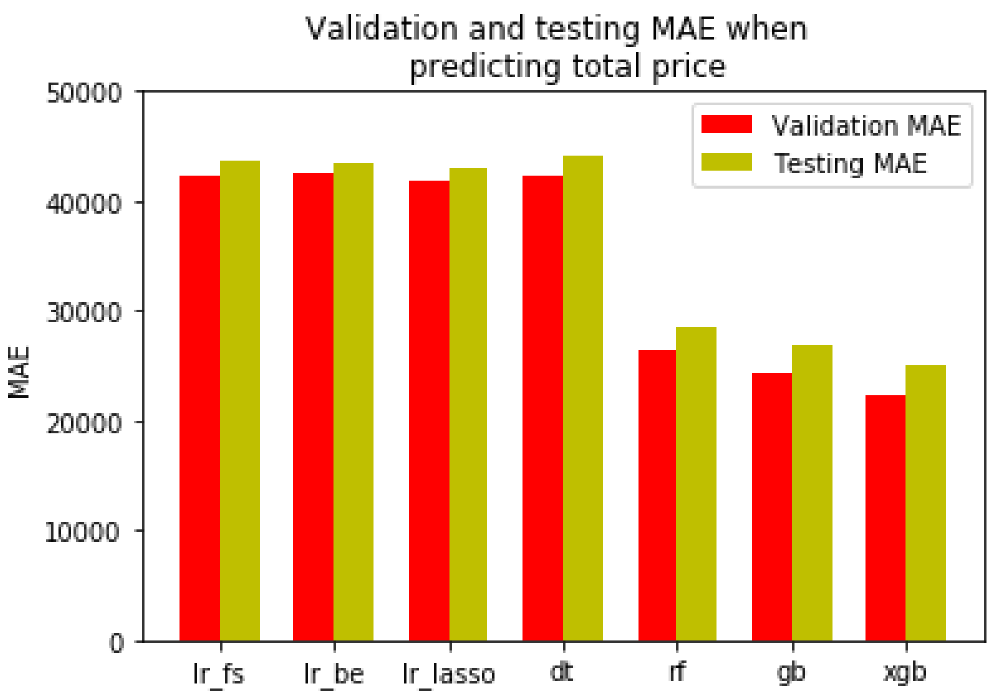


***Figure 4.8**. Validation and testing MAE when predicting total price.*

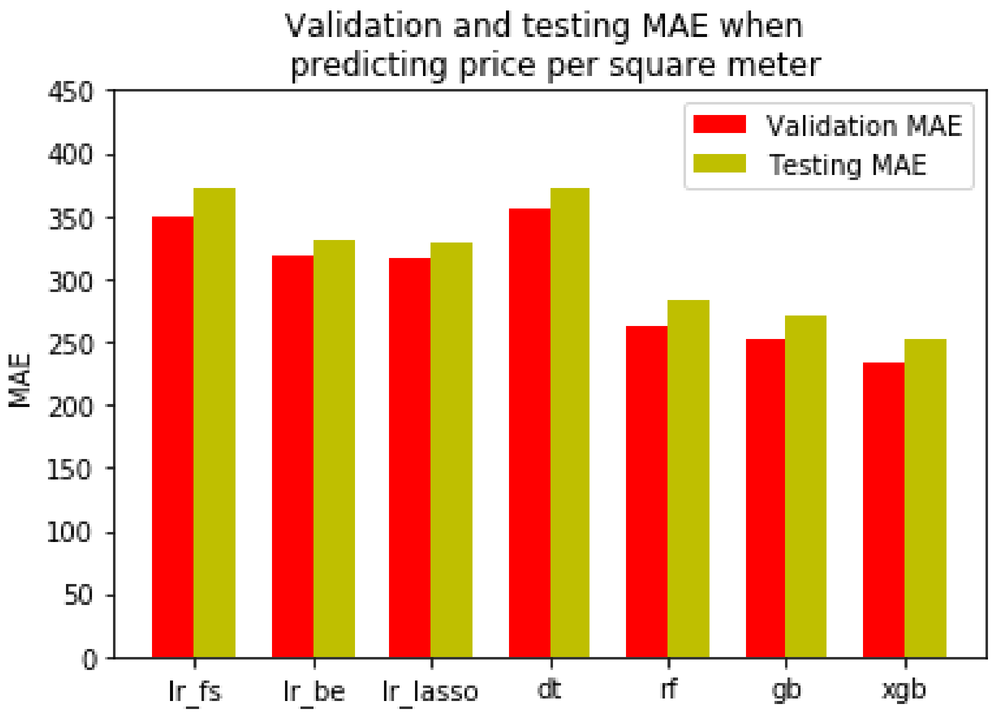


***Figure 4.9**. Validation and testing MAE when predicting price per square meter.*

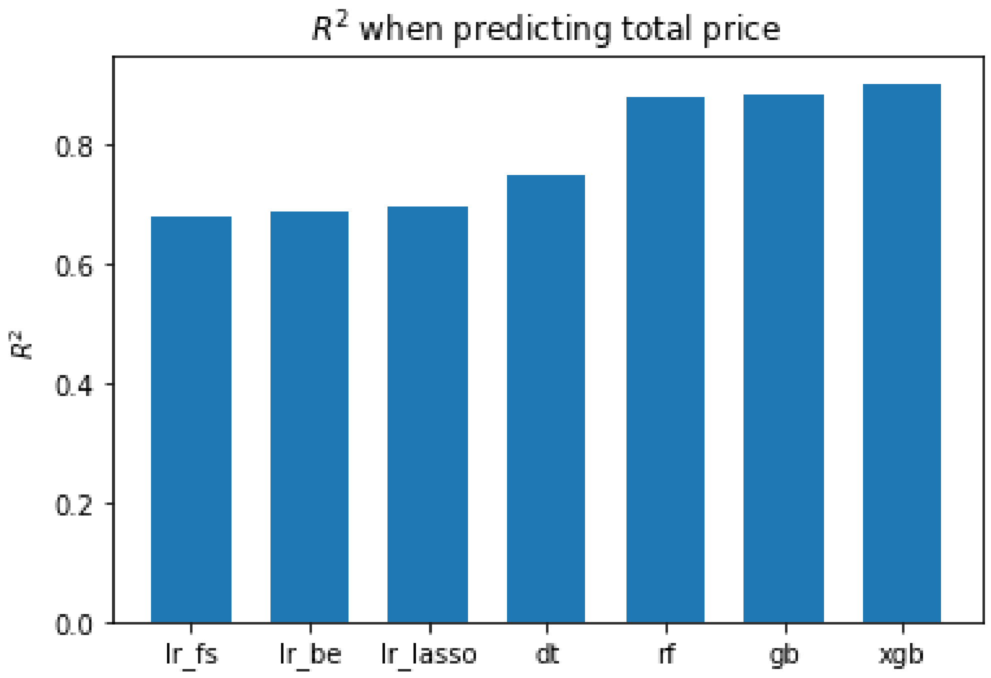


Figure 4.10. *R^2 when predicting total price.*

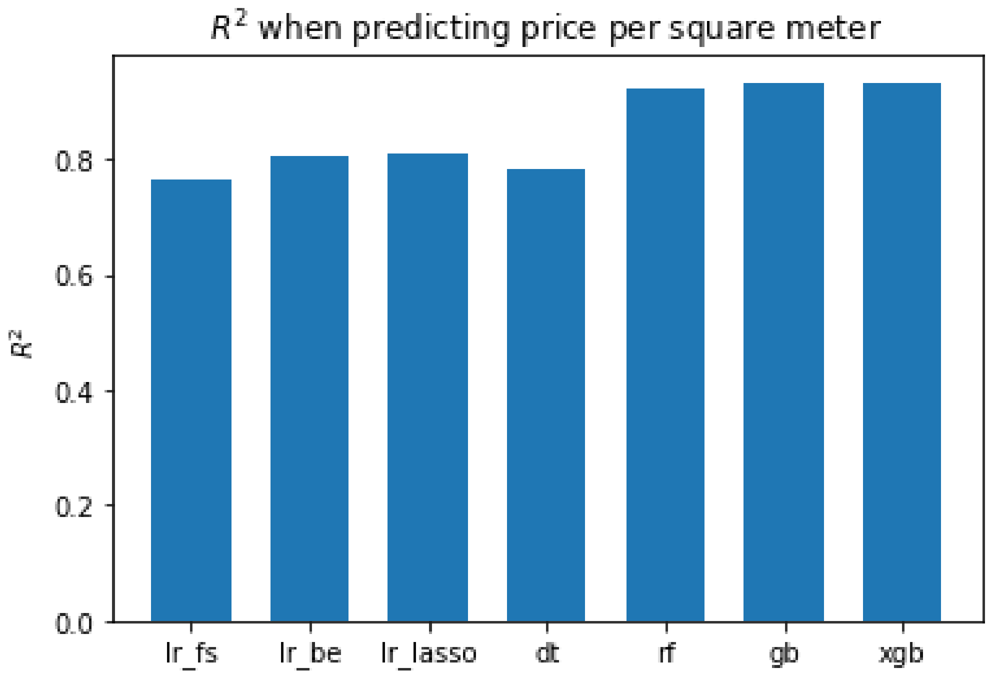


Figure 4.11. *R^2 when predicting price per square meter.*

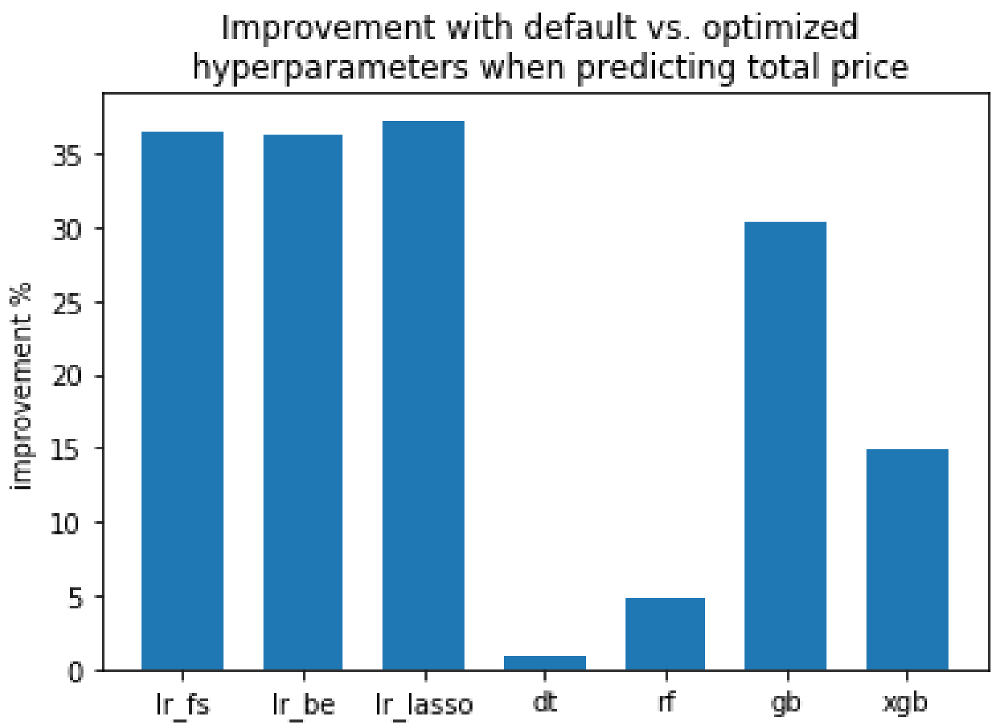


Figure 4.12. *Improvement of performance after hyperparameter optimization (total price).*

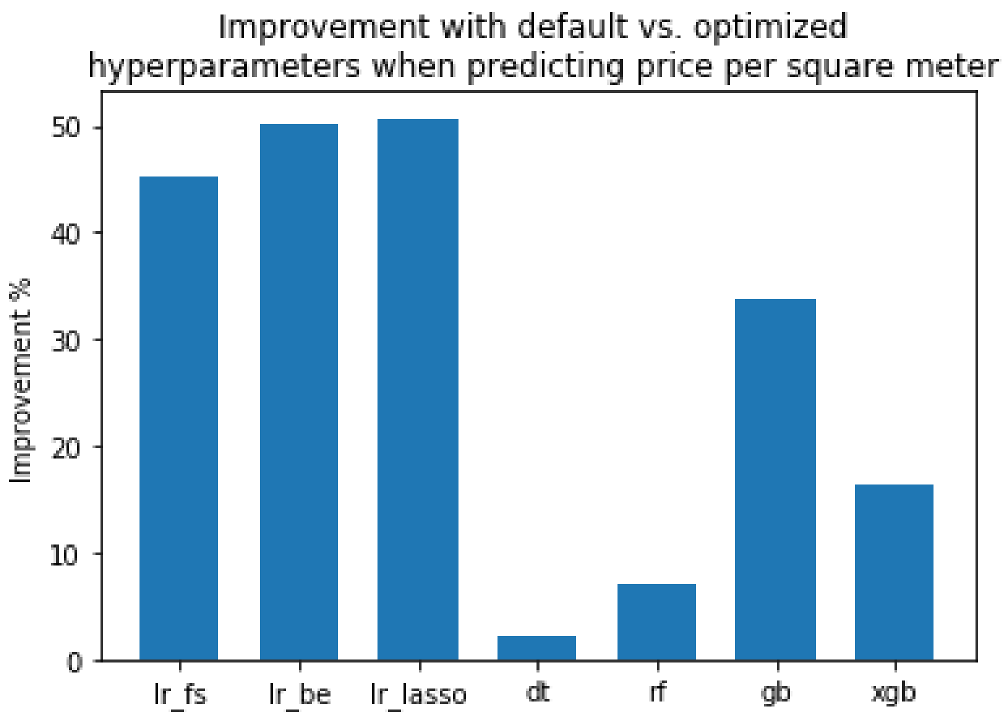


Figure 4.13. *Improvement of performance after hyperparameter optimization (price per square meter).*

5. IMPLEMENTATION

After comparing the algorithms and choosing the best one, we are ready for the implementation phase. Here, the acquired knowledge is taken into the production environment and a housing unit appraisal application is designed and built. The platform used for the implementation is the Amazon SageMaker -machine learning platform. In this chapter we go through the whole process of creating a machine learning application from planning to deployment. Amazon SageMaker is introduced and the design and implementation of the proposed solution reviewed. The platform chosen for ease of use and compatibility because the infrastructure of Etuovi.com is on the Amazon Web Services (AWS) platform.

5.1 Planning and experimenting

Working with the problem was started by thinking what features would be important when considering the price of a housing unit, checking what features are available in the data warehouse and what is the state and quality of this data. In this case ARVO already existed so the features it was using were taken along with some supplementary ones.

The easiest way to begin is to take a sample dataset with the chosen features and start experimenting locally using, for example, Scikit-learn. The point is to do as much experimenting locally as possible to avoid the costs of using a virtual cloud machine. If the data requires a lot of computational power, it is possible to first build a small scale working prototype locally only using a part of the intended dataset and then later move to the cloud environment with the full dataset. If more computational power is needed for tuning the model, one option is to use a virtual machine like Amazon Elastic Compute Cloud (EC2) to do the calculations. Recommended Python libraries for prototyping would be, in addition to Scikit-learn, Numpy, Scipy and Pandas for data handling and matplotlib for visualization.

With a working prototype, it is time to move to the deployment phase. In this case, Amazon's SageMaker platform is used for the final implementation and deployment of the application. Next, the environment will be introduced with plans for managing and maintaining the application.

5.2 Amazon SageMaker

SageMaker is Amazon's machine learning platform for building and deploying machine learning models) for production use. It aids in creating a complete pipeline for creating and maintaining software components that utilize machine learning. The deployed machine learning software can be queried through a REST API created with Amazon API gateway which is especially useful in web software development.

The platform includes innate support many common machine learning methods and algorithms including linear models, nearest neighbor, XGBoost and Deep Learning. SageMaker uses Docker containers to train and deploy models. This allows the packaging of the software into standardized units that can run on any platform running Docker. The use of containers packages code, runtime and all tools and settings into a single unit isolated from it's surroundings. It also allows the use of custom algorithms through the use of Amazon Elastic Container Service (ECS) which is Amazon's container orchestration service for running and scaling containerized applications. One could for example package their Scikit-learn implementation in a container and run it with SageMaker [30].

The Figure 5.1 shows the workflow of developing and deploying machine learning models into production. The process starts from fetching the data from a database or data warehouse and then continues to cleaning and preparing the fetched dataset for modeling using the techniques mentioned in chapter 3. After the data is ready, models are trained and tuned using the chosen algorithm. The trained models are then evaluated until a satisfactory model is obtained. SageMaker scales well with the computing requirements of the training process. One can utilize different types and sizes of computing instances to train the model depending on the requirements of the dataset and used algorithm. For example, if the task requires training a deep neural network, it is possible to allocate the task to a powerful GPU instance or even a cluster of them for maximum training efficiency [30].

When an acceptable model is obtained, you can directly deploy it on SageMaker's hosting service without needing any further work. After being deployed, the model can be integrated for example as a part of a web service and can be queried using the endpoint created by SageMaker upon deployment. After monitoring the performance of the deployed model, the accuracy can be increased further by doing required maintenance and re-training the model with the newest data [30].

5.3 Development and deployment

The production version of the housing unit price prediction system was developed on Amazon's SageMaker machine learning platform using the SageMaker development guide [30] as a reference. The architecture of the system was based on Amazon's suggestion described in the guide. The system is to be integrated to a web service, as a part of a pricing tool included in Etuovi.com. The data originates from the sales advertisements from Etuovi.com, the same as it was in the experimenting phase. The algorithm used for training the model was XGBoost trained in the same fashion as in the previous chapter with the same explanator variables and using data from a period of one year. Price per square meter was be used as the response.

The architecture of the solution can be seen in Figure 5.3. The Amazon services used by the solution are the S3 data buckets, SageMaker, Elastic Container Registry (ECR) and Lambda functions. S3 is amazon's cloud object storage that allows storing and retrieving data in

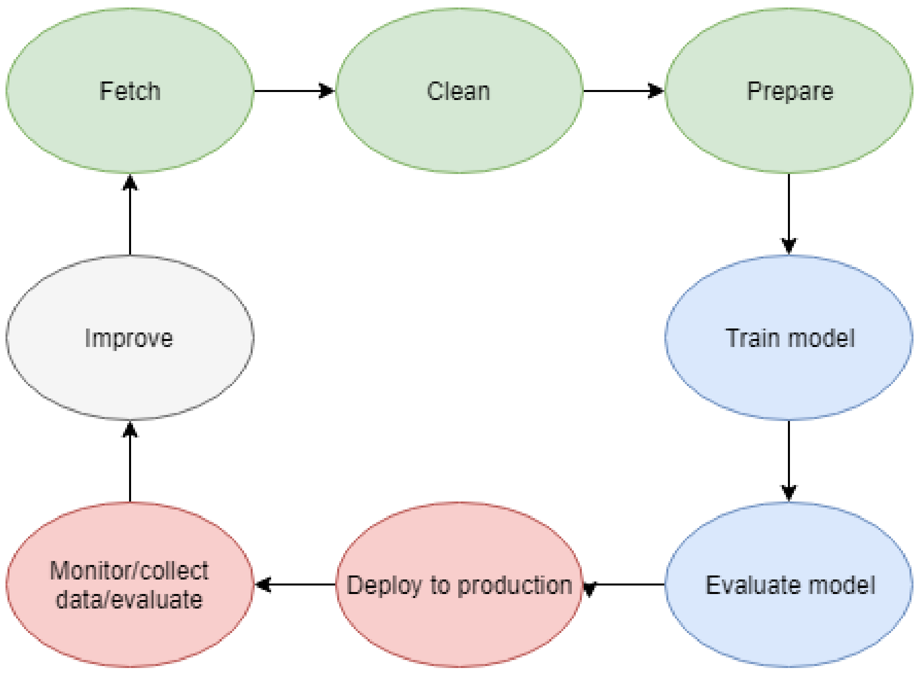


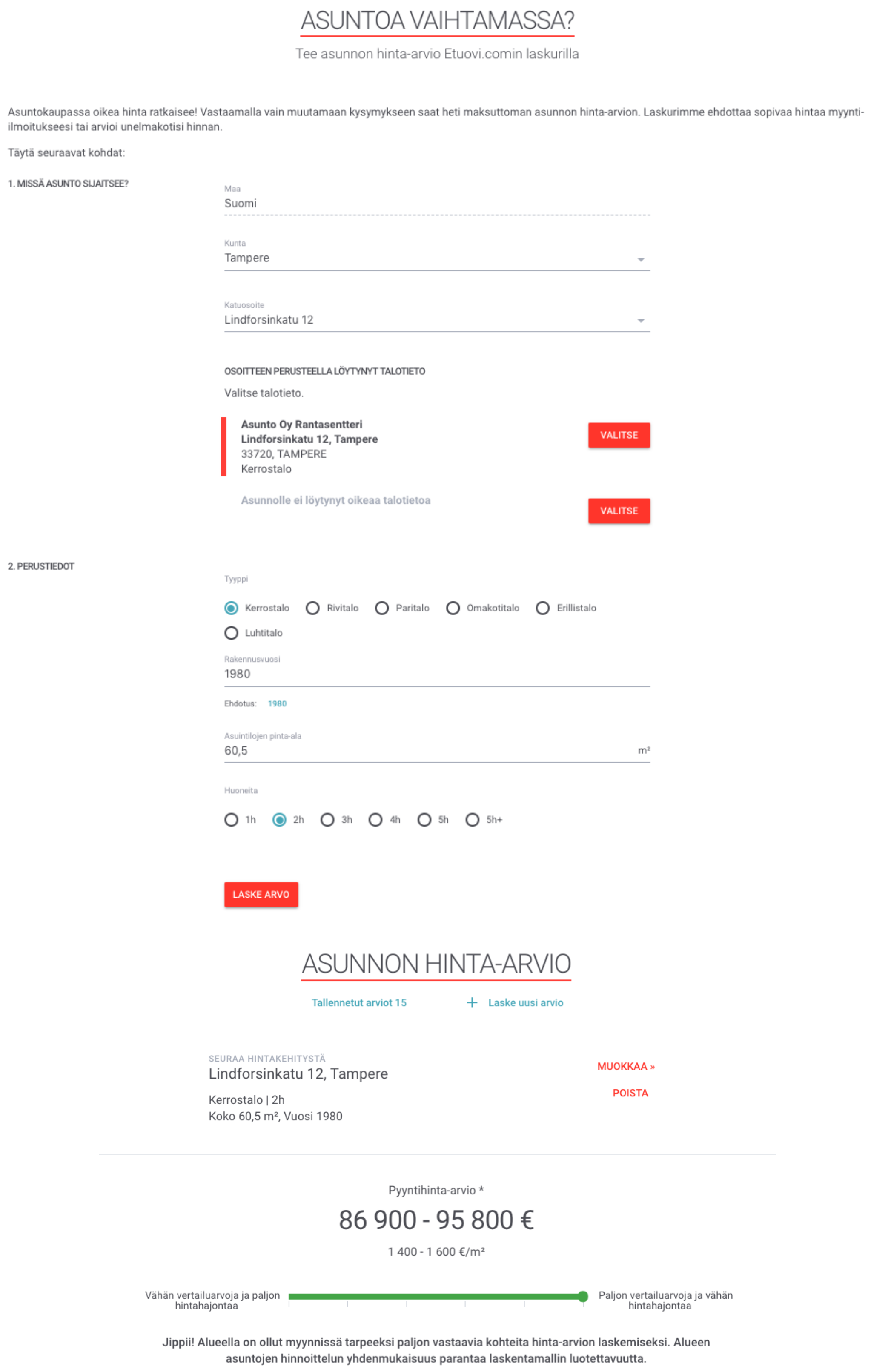
***Figure 5.1**. Machine learning application workflow.*

and from the cloud, ECR is for storing the docker images needed for training and inference and Lambda functions are for running code without the need of servers. Also, Amazon API gateway is used for building a REST API for the application. The S3 buckets are used to store the training data, the trained models and the prediction results. SageMaker is used for training and hosting the models. The models are trained and hosted in SageMaker's machine learning computing instances. The application exposes an endpoint where the hosted model is able to perform inferences. The Elastic Container Registry contains the code images for the XGBoost algorithm provided by Amazon. A Lambda function is used to periodically update the model and to invoke the endpoint when requesting a prediction through the Amazon API gateway.

The goal was to use the solution architecture provided by Amazon off-the-shelf with slight modifications as it fits very well to the general machine learning workflow. The whole environment is under Amazon Web Services for the best integrateability with the parent project Etuovi.com. The code for data preprocessing, model building and deploy was written in Python.

Two use cases exist for the solution. First being the customer using the prediction service for housing unit evaluation. On customer's request, the properties of the housing unit to be evaluated are sent to the endpoint. The deployed model makes a prediction using the inference code with these properties. The inference code uses SageMaker's inference image (XGBoost) from EC2 container registry. The result is then returned to the customer.

The second use case is creating/updating the model which is on the responsibility of the administrator. When the creation/update is run, the newest one year dataset is fetched from


ASUNTOA VAIHTAMASSA?

Tee asunnon hinta-arvio Etuovi.comin laskurilla

Asuntokaupassa oikea hinta ratkaisee! Vastaamalla vain muutamaan kysymykseen saat heti maksuttoman asunnon hinta-arvion. Laskurimme ehdottaa sopivaa hintaa myynti-ilmoitukseesi tai arvioi unelmakotisi hinnan.

Täytä seuraavat kohdat:

1. MISSÄ ASUNTO SIJAITSEE?

Maa: Suomi

Kunta: Tampere

Katuosoite: Lindforsinkatu 12

OSOITTEEN PERUSTEELLA LÖYTYNYT TALOTIETO

Valitse talotieto.

Asunto Oy Rantasentteri
Lindforsinkatu 12, Tampere
33720, TAMPERE
Kerrostalo

VALITSE

Asunnolle ei löytynyt oikeaa talotietoa

VALITSE

2. PERUSTIEDOT

Tyyppi: [✓ Kerrostalo] Rivitalo Paritalo Omakotitalo Erillistalo Luhtitalo

Rakennusvuosi: 1980

Ehdotus: 1980

Asuintilojen pinta-ala: 60,5 m²

Huoneita: 1h [✓ 2h] 3h 4h 5h 5h+

LASKE ARVO

ASUNNON HINTA-ARVIO

Tallennetut arviot 15 + Laske uusi arvio

SEURAA HINTAKEHITYSTÄ
Lindforsinkatu 12, Tampere

Kerrostalo | 2h
Koko 60,5 m², Vuosi 1980

MUOKKAA »
POISTA

Pyyntihinta-arvio *

86 900 - 95 800 €

1 400 - 1 600 €/m²

Vähän vertailuarvoja ja paljon hintahajontaa
Paljon vertailuarvoja ja vähän hintahajontaa

Jippii! Alueella on ollut myynnissä tarpeeksi paljon vastaavia kohteita hinta-arvion laskemiseksi. Alueen asuntojen hinnoittelun yhdenmukaisuus parantaa laskentamallin luotettavuutta.

***Figure 5.2**. Screenshot of the user interface of the finished application [29].*

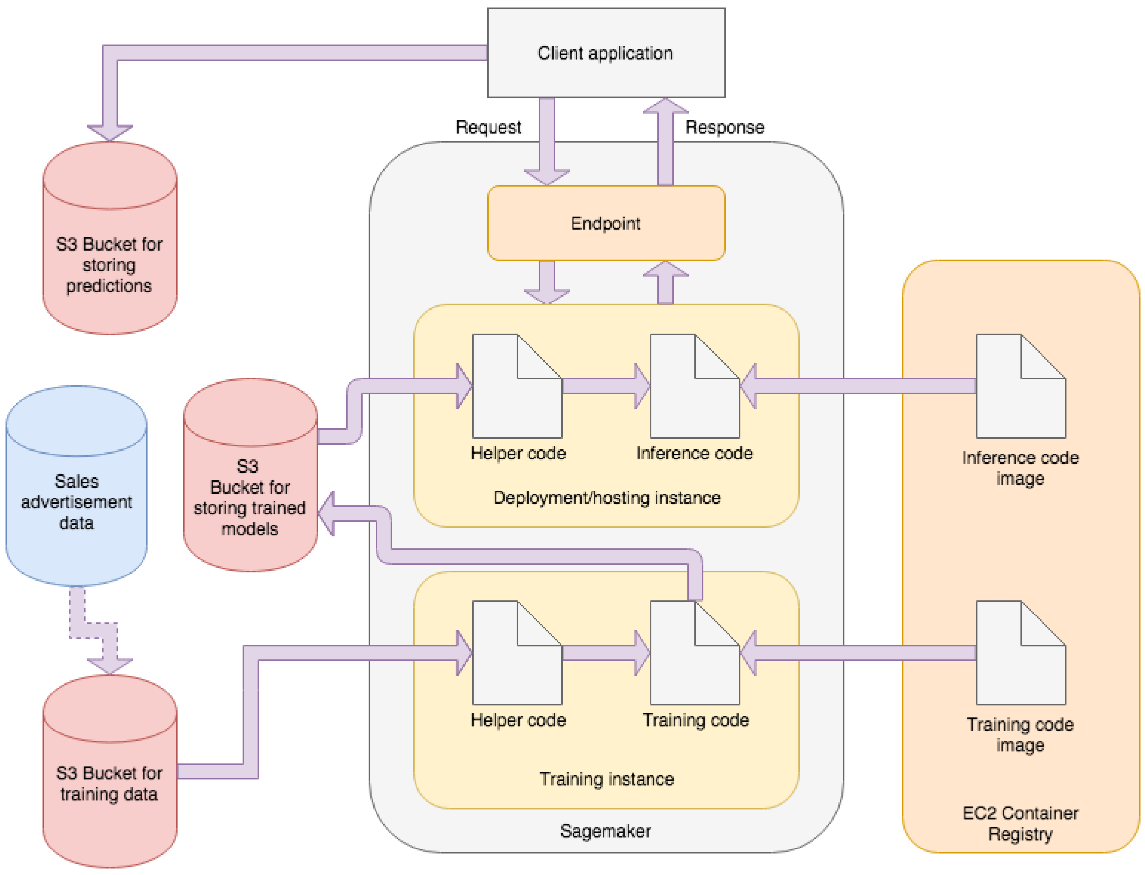


***Figure 5.3**. Architecture of the application.*

the data bucket. The data is then cleaned by the preprocessing code. Then, the cleaned dataset is given to the model trainer code for training a new model. The trainer code uses SageMaker's training code image (XGBoost) from EC2 container registry. After training, the model is validated for any errors or mistakes using cross validation. If the model is not satisfactory, an alert is sent to the administrator through Amazon CloudWatch and the process is ran from the beginning. If the model is still erroneous, the administrator uses the logs to find out the error and the process is run again after it is fixed. When the model is satisfactory, the process will continue to the deployment phase. The model is first stored in a bucket with the previous models and then deployed to SageMaker's hosting service. During the deployment process the endpoint will be created and the solution will be ready for predictions. The model is designed to be updated each month, total 12 times a year. Updating the model will be an automated task run with a periodic AWS Lambda function.

Setting up the solution requires first creating the S3 data buckets, one of them containing the data needed for training. After this, you can train fetch your data from the bucket, do any necessary preprocessing, train the model on a training instance of chosen size. Next, store the validated model in a data bucket and in the end deploy the model on the hosting instance. After deployment, the model will be ready for predictions. The code for training and deploying the model can be seen in Appendix A.

Querying the endpoint for predictions works the following way. The client calls an Amazon API gateway REST API that passes the query to a Lambda function which then sends it to the SageMaker endpoint. The model performs the prediction and the result is returned to the Lambda function which sends it to the API which responds to the client with that value. The API gateway is a layer that provides the API to the client and acts as a protective layer between the client and the backend.

The solution is integrated as a part of a web service through which it is queried for predictions. It is a completely independent module that the host service uses through the endpoint. The prediction results made through the host service will be stored in a S3 bucket to aid in future improvement of the models. All events will be logged in Amazon CloudWatch where the administrator can monitor the performance.

5.4 Challenges and future development

The documentation and tutorials for the platform made the development process quite straightforward. The problem with the platform being that even though it was almost ”plug and play” the prebuilt options available were quite limited. For example, if you would like to use some specific algorithm, there is a good chance that you have to containerize it yourself. SageMaker did not have integrated support for some components of the application. These include the fetching of data from the data warehouse, the Amazon API Gateway REST API and the retraining of the model. Including these as an integral part of SageMaker could make it more simple to use.

In the current state, the application only provides a single price estimation whereas from the consumer’s point of view it would be better to see a price range for the housing unit. As the current implementation of XGBoost in SageMaker does not support quantile loss, there are two options to produce the price range. The first being using a custom XGBoost implementation where the use of quantile regression would be possible and obtain the desired confidence range this way. Basically this means training two predictors, one for the lower limit and one for the upper limit and using the two results to form the price range. Another option is to produce the range using, for example, the error of the model and base the price range on that. This hardcoded option would look less trustworthy as the range would not be based on the single property but the whole mass instead.

There were also some features that were left out of the initial dataset due to the poor quality of the data. These features include for example amenities of the unit, nearby services and past renovations. The effect of these features on the predictive power of the model is currently not known but can be researched in the future if better quality data becomes available. A larger scale improvement to the application would be to also make predicting sales times possible. This way the seller would be able to see the estimated time for their housing unit to sell with the current price. Showing the probability of the unit being sold after a specific time period would give the user valuable information.

6. CONCLUSIONS

The main objective of this thesis was to create a housing unit price prediction application to be used in Etuovi.com. Integrating an existing pricing tool, ARVO, with Etuovi.com was considered, but the idea was scrapped and replaced by a machine learning solution built as an independent component. The application was developed on Amazon's SageMaker -machine learning platform using the XGBoost algorithm.

Developing the application started with exploring the available data and looking for existing solutions for building a price prediction tool. The studies of Antipov [2] and Masias [23] showed the suitability of random forest for predicting housing unit prices so the focus was put to experimenting with tree ensemble models. Amazon Web Service's SageMaker platform was chosen for compatibility reasons. This also enables the development of the entire component using Amazon Web Services.

As for the data, the following features were chosen for the experiments: postal code, latitude, longitude, age of the building, living area, room count, house type, condition of the unit, sauna, plot area and plot holding type. These included all the features that ARVO used along with some supplementary ones. There were some issues with the state and quality of the data. Mainly, some of the features were unusable due to inconsistent entries or had a lot of missing values.

The algorithms chosen for the experiments were linear regression, decision tree, random forest, gradient boosting and extreme gradient boosting. Out of these, the decision tree was the worst and extreme gradient boosting had the highest performance. Thus, XGBoost was chosen for the implementation. The most important features for the XGBoost model were house type, location and room count. The R^2 score of the model was 0.932. The test results showed the model's effectiveness in practice. Over half of the test samples had an error of less than one percent while 80% of the samples had an error of less than ten percent. Less than four percent of the test samples had an error of 25% or more.

The application was built using SageMaker while being supported by other Amazon Web Services' products. The flow of the application is as follows: The data is exported from Etuovi.com's data warehouse, cleaned and divided into training and validation sets. The data is stored to an S3 bucket. SageMaker uses the data from the bucket to train, validate and store the model for future revision after which it is deployed. The deployed model exposes an endpoint which can be used through a REST API created with Amazon API gateway to make predictions. This way the software is independent from the service it is integrated with.

The experimenting and developing the application went mostly without problems although in the beginning it took some time to find the correct tools and methods to proceed as there was only a little experience in the company for developing machine learning applications. In the end, the deployed application was successful. The accuracy of the application is very good in most cases, especially in highly populated areas that have a homogenous housing unit population. Detached and semi-detached houses were the most problematic probably due to them having more uniqueness and extra amenities compared to apartments. Of course, the application is only usable in Finland due to data limitations. The application built by the proposed guidelines is completely viable and can be integrated easily as a part of a web service as an independent component.

Possible improvements to be considered in the future included developing a method to show a price range for the target housing unit instead of a single price. Also, the features that were left out of the dataset due to poor data quality might be of use in improving the performance of the model. Lastly, also having the option to predict sales times would give the user some valuable information.

Even though the system is able to give an accurate prediction for the price of a housing unit with high probability, it is nowhere near perfect. In the end, every apartment is unique and the law of supply and demand determines the final contract price, which will differ from the asking price of the apartment almost without question. So, as interesting and entertaining the application is, estimating the price when actually selling a housing unit is the job of a real estate agent.

REFERENCES

[1] K. Adetiloye, P. Eke, *A REVIEW OF REAL ESTATE VALUATION AND OPTIMAL PRICING TECHNIQUES*, Asian Economic and Financial Review, Vol. 4, Iss. 12, 2014, pp. 1878–1893.

[2] E. Antipov, *Mass appraisal of residential apartments: An application of Random forest for valuation and a CART-based approach for model diagnostics*, Expert Systems with Applications, Vol. 39, Iss. 2, 2012, pp. 1772–1778.

[3] G. Brassington, *Mean absolute error and root mean square error: which is the better metric for assessing model performance?*, Geophysical Research Abstracts, Vol. 39, Iss. 1, 2017.

[4] L. Breiman, *Random Forests*, Machine Learning, Vol. 45, Iss. 1, 2001, pp. 5–32.

[5] L. Breiman, J. Friedman, C. Stone, R. Olshen, *Classification and Regression Trees*, Taylor and Francis, 1984, 368 p.

[6] T. Chai, R. Draxler, *Root mean square error (RMSE) or mean absolute error (MAE)? – Arguments against avoiding RMSE in the literature*, Geosci. Model Dev., Vol. 7, Iss. 1, 2014, pp. 1247–1250.

[7] G. Chandrashekar, F. Sahin, *A survey on feature selection methods*, Computers and Electrical Engineering, Vol. 40, Iss. 1, 2014, pp. 16–28.

[8] T. Chen, *XGBoost: A Scalable Tree Boosting System*, Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2016, pp. 785–794.

[9] P. Flach, *Machine Learning: The Art and Science of Algorithms that Make Sense of Data*, Cambridge University Press, 2012, 709 p.

[10] V. Fonti, *Feature Selection using LASSO*, Available: https://-beta.vu.nl/nl/Images/werkstuk-fonti_tcm235-836234.pdf, 2017.

[11] J. Freund, *Mathematical Statistics with Applications*, Pearson, 2014, 554 p.

[12] J. Friedman, *Greedy function approximation: A Gradient Boosting Machine*, The Annals of Statistics, Vol. 29, Iss. 5, 2001, pp. 1189–1232.

[13] J. Friedman, *Stochastic gradient boosting*, Computational Statistics and Data Analysis, Vol. 38, Iss. 4, 2002, pp. 367–378.

[14] I. Guyon, A. Elisseeff, *An Introduction to Variable and Feature Selection*, Journal of Machine Learning Research, Vol. 3, Iss. 1, 2003, pp. 1157–1182.

[15] G. Hackeling, *Mastering Machine Learning with scikit-learn*, Packt Publishing Limited, 2014, 238 p.

[16] J. Han, J. Pei, M. Kamber, *Data Mining Concepts and Techniques*, Morgan Kauffman, 2012, 744 p.

[17] T. Hastie, R. Tibshirani, J. Friedman, *The Elements of Statistical Learning*, Springer, 2008, 745 p.

[18] A. Jost, J. Nelson, K. Gopinathan, C. Smith, *Real estate appraisal using predictive modelling*, US5361201A, 1994. Available: https://patents.google.com/patent/US5361201A/en

[19] S. Kotsiantis, *Decision trees: A recent overview*, Artificial Intelligence Review, Vol. 39, Iss. 4, Apr. 2013, pp. 1–23.

[20] A. Legendre, *Nouvelles méthodes pour la détermination des orbites des comètes*, F. Didot, 1805, 290 p.

[21] W.Y. Loh, *Classification and regression trees*, WIREs, Vol. 1, Iss. 1, 2011, pp. 14–23.

[22] P. Louridas, C. Ebert, *Machine Learning*, IEEE Software, Vol. 33, Iss. 5, 2016, pp. 110–115.

[23] V. Masias, *Property Valuation using Machine Learning Algorithms: A Study in a Metropolitan-Area of Chile*, International Conference on Modeling and Simulation in Engineering, Economics and Management (MS'2016), 2016.

[24] T. Mitchell, *Machine Learning*, McGraw-Hill Education, 1997, 432 p.

[25] D. Montgomery, E. Peck, G.G. Vining, G.G. Vining, *Introduction to Linear Regression Analysis*, John Wiley and Sons, 2012, 645 p.

[26] H. Motoda, H. Liu, *Feature Extraction, Construction and Selection: A Data Mining Perspective*, Springer, 1998, 410 p.

[27] D. Nielsen, *Tree Boosting With XGBoost - Why Does XGBoost Win "Every" Machine Learning Competition?*, Norwegian University of Science and Technology, Available: https://brage.bibsys.no/xmlui/bitstream/handle/11250/2433761/16128_FULLTEXT.pdf?sequence=1&isAllowed=y, 2016.

[28] D. Steinberg, *CART: Classification and Regression Trees*, Available: https://www.researchgate.net/profile/Dan_Steinberg2/publication/265031802_Chapter_10_CART_Classification_and_Regression_Trees/links/567dcf8408ae051f9ae493fe/Chapter-10-CART-Classification-and-Regression-Trees.pdf, 2009.

[29] *Etuovi.com*, Etuovi.com, website, 2018. Available (accessed on 5.10.2018): https://www.etuovi.com/

[30] *Sagemaker Developer Guide*, Amazon Web Services, Inc., 2018. Available: https://-docs.aws.amazon.com/sagemaker/latest/dg/sagemaker-dg.pdf.

[31] R. Tibshriani, *Regression Shrinkage and Selection via the Lasso*, Journal of the Royal Statistical Society, Vol. 58, Iss. 1, 1996, pp. 267–288.

[32] L. Torgo, *Inductive Learning of Tree-based Regression Models*, University of Porto, Available: http://www.dcc.fc.up.pt/ ltorgo/PhD/th1.pdf, Sept. 1999.

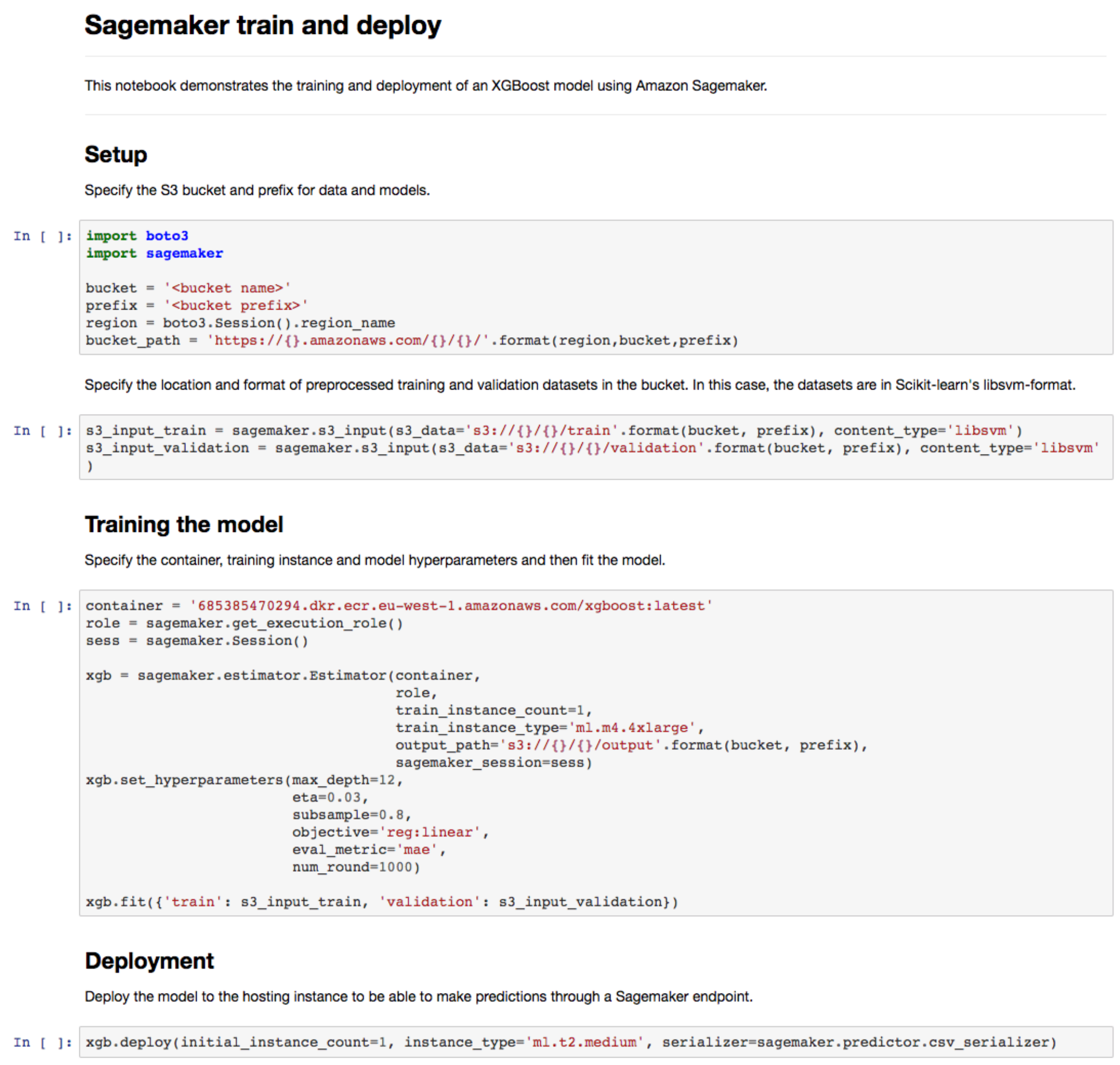
[33] R. Walpole, R. Myers, S. Myers, K. Ye, *Probability and Statistics for Engineers and Scientists*, Pearson, 2012, 791 p.

[34] S. Weisberg, *Applied Linear Regression*, Wiley, 2013, 336 p.

[35] L. Wilkinson, *Tree Structured Data Analysis: AID, CHAID and CART*, Sawtooth/SYSTAT Joint Software Conference, 1992.

[36] C. Willmott, K. Matsuura, *Advantages of the mean absolute error (MAE) over the root mean square error (RMSE) in assessing average model performance*, Climate Research, Vol. 30, Iss. 1, 2005, pp. 79–82.

APPENDIX A: TRAINING AND DEPLOYING THE MODEL



Sagemaker train and deploy

This notebook demonstrates the training and deployment of an XGBoost model using Amazon Sagemaker.

Setup

Specify the S3 bucket and prefix for data and models.

```
In [ ]: import boto3
        import sagemaker

        bucket = '<bucket name>'
        prefix = '<bucket prefix>'
        region = boto3.Session().region_name
        bucket_path = 'https://{}.amazonaws.com/{}/{}/'.format(region,bucket,prefix)
```

Specify the location and format of preprocessed training and validation datasets in the bucket. In this case, the datasets are in Scikit-learn's libsvm-format.

```
In [ ]: s3_input_train = sagemaker.s3_input(s3_data='s3://{}/{}/train'.format(bucket, prefix), content_type='libsvm')
        s3_input_validation = sagemaker.s3_input(s3_data='s3://{}/{}/validation'.format(bucket, prefix), content_type='libsvm'
        )
```

Training the model

Specify the container, training instance and model hyperparameters and then fit the model.

```
In [ ]: container = '685385470294.dkr.ecr.eu-west-1.amazonaws.com/xgboost:latest'
        role = sagemaker.get_execution_role()
        sess = sagemaker.Session()

        xgb = sagemaker.estimator.Estimator(container,
                                            role,
                                            train_instance_count=1,
                                            train_instance_type='ml.m4.4xlarge',
                                            output_path='s3://{}/{}/output'.format(bucket, prefix),
                                            sagemaker_session=sess)
        xgb.set_hyperparameters(max_depth=12,
                                eta=0.03,
                                subsample=0.8,
                                objective='reg:linear',
                                eval_metric='mae',
                                num_round=1000)

        xgb.fit({'train': s3_input_train, 'validation': s3_input_validation})
```

Deployment

Deploy the model to the hosting instance to be able to make predictions through a Sagemaker endpoint.

```
In [ ]: xgb.deploy(initial_instance_count=1, instance_type='ml.t2.medium', serializer=sagemaker.predictor.csv_serializer)
```