Master's thesis

Master's Programme in Computer Science

Rental Price Prediction on Greater Helsinki Apartments

Lauri Karanko

May 11, 2022

Faculty of Science

University of Helsinki

Supervisor(s)

Prof. Jukka K. Nurminen

Examiner(s)

Prof. Jukka K. Nurminen, Assoc. Prof. Timo Asikainen

Contact information

P. O. Box 68 (Pietari Kalmin katu 5)
00014 University of Helsinki,Finland

Email address: info@cs.helsinki.fi
URL: http://www.cs.helsinki.fi/

HELSINGIN YLIOPISTO – HELSINGFORS UNIVERSITET – UNIVERSITY OF HELSINKI

Tiedekunta — Fakultet — Faculty Faculty of Science	Koulutusohjelma — Utbildningsprogram — Study programme Master's Programme in Computer Science

Tekijä — Författare — Author

Lauri Karanko

Työn nimi — Arbetets titel — Title

Rental Price Prediction on Greater Helsinki Apartments

Ohjaajat — Handledare — Supervisors

Prof. Jukka K. Nurminen

Työn laji — Arbetets art — Level Master's thesis	Aika — Datum — Month and year May 11, 2022	Sivumäärä — Sidoantal — Number of pages 51 pages

Tiivistelmä — Referat — Abstract

Determining the optimal rental price of an apartment is typically something that requires a real estate agent to gauge the external and internal features of the apartment, and similar apartments in the vicinity of the one being examined. Hedonic pricing models that rely on regression are commonplace, but those that employ state of the art machine learning methods are still not widespread. The purpose of this thesis is to investigate an optimal machine learning method for predicting property rent prices for apartments in the Greater Helsinki area. The project was carried out at the behest of a client in the real estate investing business. We review what external and inherent apartment features are the most suitable for making predictions, and engineer additional features that result in predictions with the least error within the Greater Helsinki area. Combining public demographic data from Tilastokeskus (Statistics Finland) and data from the online broker Oikotie Oy gives rise to a model that is comparable to contemporary commercial solutions offered in Finland. Using inverse distance weighting to interpolate and generate a price for the coordinates of the new apartment was also found to be crucial in developing an performant model. After reviewing models, the gradient boosting algorithm XGBoost was noted to fare the best for this regression task.

ACM Computing Classification System (CCS)
Computing methodologies → Modeling and simulation → Model development and analysis→ Modeling methodologies
Applied computing → Operations research → Forecasting

Avainsanat — Nyckelord — Keywords

forecasting, modeling, housing

Säilytyspaikka — Förvaringsställe — Where deposited

Helsinki University Library

Muita tietoja — övriga uppgifter — Additional information

Algorithms study track

Contents

1 Introduction

1.1 Topic Introduction

Determining the proper rental price at which an apartment should be rented at often requires a domain expert to evaluate the apartment features. Knowledge of how other landlords have priced their rental units in the same geographic area or city district is also necessary to find the sweet spot at which the rent should be set. A far too high rent will see the property vacant or without a tenant for an extended time, while a too low rent will diminish the returns from the rental unit, hurting the bottom-line of the proprietor. The ability to estimate a suitable rental price for a property also helps investors decide whether housing units might have an acceptable return of investment prior to consulting other domain experts, and identify sites of remarkable return of investment due to the speed of analysis when investigating a multitude of properties.

With the help of contemporary machine learning methods and a suitable repository of historic data on rental unit transactions, it is possible to create a predictive model that is able to properly price rental units at a satisfactory level. With this model, analysts can support the rent prices real estate brokers have chosen with the help of the predictions of the model, assuaging their worries that a rental unit might be priced unsuitably (Truong et al., 2020).

1.2 Research Questions

The purpose of the study is to find whether it is possible to accurately predict the proper rental price of apartment buildings. This can be best tested by attempting to create a machine learning model that is as accurate as possible when it comes to predicting the rental prices of new housing units, comparing the results given by the model to those made by professional real estate agents or landlords. As several commercial rental price prediction applications also already exist online, an additional aim of the study is to determine how accurate the models built in this study are relative to these similar commercial systems in use.

The main focus of the study will lie in developing and testing a model to predict rental prices in the greater Helsinki area, containing the cities of Helsinki, Espoo and Vantaa. The city of Kauniainen, which is a part of greater Helsinki, is sparsely populated and will not be considered in this study due to the low amount of rental data available from the region. As the aim of this thesis was to create a production ready model for a client and deploy it to be used on a custom online platform, the main indicator of success was the accuracy and precision of the final model.

1.3 Thesis Overview

This thesis has feature engineering as a large focus, as well as the techniques used to generate the models. Modelling pitfalls and possibilities will be discussed, and any insights that were found in the tuning of hyperparameters and the choosing of parameters are shared. The two machine learning techniques that during an exploration phase resulted in the best models were Random Forest Ensemble and XGBoost (shorthand from *eXtreme Gradient Boost / Gradient Boosted Trees*, or *Optimized Gradient Boost*), and thus the chapter Theoretical Background will be largely focused on them.

2 Theoretical Background

2.1 Introduction to Rental Price Prediction

Prior to the revolutionary rise of machine learning in predictive analysis, traditional statistical tools for estimating the future prices of financial instruments such as capital stocks or other assets of value such as housing units were employed by technical analysts. Unfortunately, traditional statistical methods are not sufficient for handling apartment rental price prediction without difficulty if there are too many explanatory variables to track, thus we need machine learning to discover the patterns and relationships present in the data we have (Kok et al., 2017). With the use of the proper machine learning techniques, it is possible to turn the the experience of people (training data) into expertise (model) that allows making accurate assessments on rental housing pricing. Years of real estate agent experience pricing apartments can be generalized into models that make predictions for us for previously unseen apartment locations.

2.1.1 Contemporary Real Estate Price Prediction in Finland

Currently several companies are tackling the same problem of estimating rental or sales prices for apartment units in Finland, and have working solutions for sale. *Vuokraovi* provides a public version of their price prediction model which accepts fewer parameters called *Vuokralaskuri*[†], but offers a pro-version for 1500€ per month. *SkenarioLabs*[‡] has created automatic valuation models for real estate portfolio management for which they sell API access. Among other analytics, price estimation for property value is provided. *Cityfier*[§] by A-insinöörit is a system to estimate current real estate prices and promises to give price accurate predictions up to the year 2040 by evaluating zoning maps.

Some businesses leverage using ML-assisted price estimation as a part of their real estate brokerage or agency. *Kodit.io*[¶] provides a service in which they will immediately give a cash offer for any housing unit given that the user answers 12 questions on the condition

[†]https://vuokraovi.com/vuokralaskuri

[‡]https://www.skenariolabs.com/

[§]https://www.cityfier.com/

[¶]https://www.kodit.io

and properties of the housing unit on the *Kodit.io* web interface. A model will calculate a suitable price with no real estate agent visiting the physical apartment prior to the sale transaction. The real estate agency *Blok.ai*[*] requests for 13 parameters from the user, after which their model will give a price estimate for the housing unit.

There is a reason for the emergence of rental estimation systems like this, as researchers such as *Kok et al.* claim that instant automated valuation models have surpassed traditional appraisal methods when using the median absolute error as the criterion of comparison (Kok et al., 2017).

2.2 Data Overview

The data used to train the model has been collected from the Finnish housing broker Oikotie. The API for the Oikotie Real Estate service[†] can be perused for all of the data available for this thesis. Additional data from Tilastokeskus[‡] was used as improve the accuracy of the model with information specific to certain postal codes. The data used is in many ways quite homogeneous as the only type of dwelling represented in the final model training data were apartment units.

2.3 Prediction models

When it comes to selecting apt machine learning models to estimate rental unit prices, we look into the use of supervised models, as our data is already labeled and we seek to understand the relationships between the features of apartments and the target rental price. Due to predicting a numerical value, we build a regressor to do so (James et al., 2021).

2.3.1 Regression models

At its core, regression involves fitting a function as well as possible through some data points. A numerical prediction on the price of an apartment y_i is made based upon a vector of features x_i of the apartment $f(x_i) = y_i$ (Friedman, 2001).

[*]https://www.blok.ai

[†]https://api.asunnot.oikotie.fi/

[‡]Statistics Finland; https://www.stat.fi/

Multiple regression analysis is a very suitable method for forecasting the values of our target variable rental price y (in statistics dependant variable) that is dependant on other variables (or in other words, independent variables) that can be given numerical representations x. In regression analysis some variables have an association between them called a correlation, causing changes in one variable to result in a change in the other. Regression methods represent how changes of any given parameter change the target value, and can be used to predict a target value with previously unseen parameters (Varma et al., 2018). When using a simple regression analysis to predict rental housing prices, the resulting model would be considered a hedonic pricing model (Kok et al., 2017), with the estimated rental price being affected by the characteristics of the apartments themselves collected from Oikotie, and other external factors collected from Tilastokeskus related to the apartments or area the apartment is located at. In the hedonic method, the price of apartments for sale or rent on the market is expected to be determined by a function of measurable characteristics such as apartment age, the presence of a balcony or the distance to a school. It attempts to identify these characteristics and the effect on the price for each characteristic. The hedonic regression pricing method is often used in real-estate to estimate the influence various factors have on the price of housing, with the price being the dependant variable and all other factors being independent factors (Wing and Chin, 2003). An issue that hedonic pricing models run into is that they are limited to simple regression that is unable to map non-linear relationship to the target value: *rental price*. They typically include a search engine that compares features of the subject property with comparable properties in a radial range (Kok et al., 2017).

$$\hat{y} = \sum_{i=1}^{n} x_i * w_i \tag{2.1}$$

$$\hat{y} = rooms * w_1 + apartment\_size * w_2 + floor * w_3 + minimum\_rent * w_4 \tag{2.2}$$

In equation 2.1 and 2.2, n is the number of features, x are the individual features and w_i are weights corresponding to the features x_i. Equation 2.2 shows an example for rental price calculation with regression. $\hat{y}$ is the predicted value, the weights w_n for the features promote some features over others in the calculation of $\hat{y}$.

Besides the linear regression model, regression trees are the simplest tree-based regression method, which are able to capture the non-linear interactions between input features and the target variable that simple linear regression is not capable to. A greedy tree is generated by selecting a root node, at which we iterate through all the features and attempt to find the feature that maximizes the splitting criteria we are using. At the leaf nodes created from the split, we continue to build the tree using the same method until a stopping criteria we have set is met. The main methods to control how the tree is built are limiting maximum tree depth, limiting the maximum number of nodes or leaves, preventing splits if there are not enough observations at a split and implementing a constraint on the minimum gain threshold from any split. Post-training, tree pruning can be used to remove nodes from the tree that don't contribute much to the accuracy of the tree by comparing the accuracy of the tree prior to pruning and post-pruning (James et al., 2021).

Nevertheless, *Madhuri et al.* suggest that when it comes to regression methods for predicting housing prices, gradient boosting methods fare the best out of all regression methods. Regression trees are likely to over fit data during training (Madhuri et al., 2019).

2.3.2 Gradient Boosting

When working with data, we have some set of data points $X = [x_1, x_2, ..., x_n]$ through which we are interested in plotting a curve through. This curve is the model, and using it we can predict the value of some value of y_i given x_i. The curve that represents the model can be decomposed to simpler functions, each representing some aspect of the curve. It is also possible to begin modeling the curve by using several simpler functions, which when combined (added together) generate a composite function that models some data points on the curve. This technique is called additive modeling, and it is the foundation of the gradient boosting method. This more complicated function represents the model that maps some observed feature to a scalar value. Gradient Boosting Machines use additive modeling by progressively adjusting and adding sub-functions to the more complex composite function, working towards a better approximation of a good model for the given data points (Friedman, 2001). In a machine learning context, these simple models are often called weak learners. If all the base model and weak models that are added together are linear, the composite model is still linear akin to shown in Equation 2.2, meaning in this situation there is no need for boosting to solve a linear equation.

$$F_m(x) = F_{m-1}(x) + f_m(x) \tag{2.3}$$

In equation 2.3, boosting is expressed as a recursive function, in which the previous model $F_{m-1}(x)$ is altered by the weak model $f_m(x)$ to get the next composite model $F_m(x)$. $f_m(x)$ never alters the previous models, but only adds to it. Typically we stop adding

$f_m(x)$ when the increases to the performance of the model stagnate or we are content with the model. The first step F_0 is only a weak model $f_0(x)$ that attempts to learn y with x with no additional alterations.

As both random forests and gradient boosting methods find their foundations in the tree-based supervised learning algorithm, it is prudent to define decision trees as well. Something common to all decision trees is that they involve segmenting the predictor space into various simpler regions, and then typically use the mean value for the predicted feature to determine the region into which the predicted value belongs. These regions when viewed in the tree-representation are typically called terminal nodes or leaves of the tree (Breiman, 2001).

The first node of the decision tree called the root node contains the full training data set. Splits attempt to minimize variance in a regression line for both subsamples or resulting leaf nodes using the best predictor available in the data set in the node . This is repeated in the leaf nodes with the best predictor or independent variable available in the leaf node data set until a previously specified limit such as tree depth is reached. The same process is repeated recursively in all branches of the tree (Kok et al., 2017).

Splits in trees at their most basic form are generated using a greedy algorithm called *recursive binary splitting*, which takes the best possible split at each junction in the tree. This method attempts to minimize the RSS (residual sum of squares) at each step since considering all possible partitioning schemes would be far too computation intensive (James et al., 2021). *Chen et al.* in their implementation of XGBoost call this algorithm the *exact greedy algorithm*, with the algorithm being the same as implemented in *scikit-learn* for tree-splitting. XGBoost does not use this algorithm when executing on several machines simultaneously (T. Chen and Guestrin, 2016a). Other more sophisticated methods exist as well, such as initially generating a larger tree with more robust branches, that is then pruned of the less useful branches by estimating the test error of the sub-tree using cross-validation. Due to this method having to compute the test error for each sub-tree, pruning can rather be implemented with *cost complexity pruning* (weakest link pruning, included in *scikit-learn 0.22* and above), in which sequences of sub-trees are viewed (Buitinck et al., 2013) (James et al., 2021).

Gradient Boosting was born from a generalization of the Adaptive Boosting (AdaBoost) method, in which observations are are given more weight if they are difficult to classify, and new learners are added to focus on these difficult features. To calculate loss, it minimizes the exponential loss function making it somewhat sensitive to outliers in the data. Finally, the prediction is made by a collection or ensemble of weak learners (James et al., 2021).

Similar to AdaBoost, Gradient Boosting is a method for improving the accuracy of any machine learning model. Gradient Boosting does not share the fixed loss function that AdaBoost utilizes. Gradient Boosting Machines (GBM) are suitable for regression tasks such as relta price estimation and differ in their approach compared to random forest in that the method only generates a single tree, which is evaluated for it's accuracy, prior to using it as a guide for creating the next tree. Thus, the trees are combined sequentially and additively to incrementally increase the prediction accuracy with the training data (James et al., 2021).

Gradient Boosting typically combines the use of gradient descent and boosting, and identifies the weak learners using the gradients from gradient descent (James et al., 2021). Often a weak or basic learning algorithm is called several times in series, and the prediction accuracy is increased as more weak learners are chained (Friedman, 2001). In the end, the final model is dictated by the forms of the weak models, for which the form is not specified by boosting. Nevertheless, the composite model is gradually tweaked towards a better approximation by the step-wise changes made during training. Each step is evaluated by comparing the prediction from the model during the step $F_m(x)$ against y, with $y - F_m(x)$ being $f_m(x)$, typically referred to as Δ_m due to its vector nature. Thus the difference between the predicted value $\hat{y} = F_m(x)$ is the residual vector which is then used to nudge the training of the next weak model $f_m(x)$ towards the direction of y (Friedman, 2001). While $\hat{y}$ approaches y, speed how fast it approaches y is often slowed down by a learning-rate factor η to prevent the model from over fitting.

$$F_m(X) = F_{m-1}(X) + \eta f_m(X) \tag{2.4}$$

Equation 2.4 shows boosting with the addition of the learning rate η. Typically $M = [m_1, m_2, ..., m_n]$ and η are decided by the use of grid-search or some other method to approximate a good value (Friedman, 2001).

Gradient Descent

$$x_{i+1} = x_i - \gamma \nabla f(x_i) \tag{2.5}$$

The process of gradient descent is used to minimize the loss when adding new trees, thus it attempts to find the position of x for $f(x)$ that returns the smallest possible value. An initial value for x_0 selected at random or by some prior convention, which is then continuously nudged towards the position in which $f(x)$ is the smallest. The equation 2.5 shows a single step of the gradient descent algorithm of function f at point x_i with the step size γ. Examining the negative gradient $-\nabla f(x_{(i)})$ of $f(x)$ allows to identify the upwards or downwards direction of the curve relative to x_{i-1} (Friedman, 2001). The gradient is calculated using first order derivatives for $f(x)$ (Zhou et al., 2019). The position x_{i-1} is updated to x_i by adding the direction vector multiplied by the learning rate η to x_{i-1}. An example of this can be seen in Figure 2.1. Thus, gradient descent measures the local gradient of the loss function at a point, then moves in the direction of descending gradient, moving towards a gradient of zero which is a minima. It is possible to perform gradient descent on any loss function that can have its differential calculated (Friedman, 2001).

In addition to training weak learners on the residual vectors $\hat{y}$ from the intermediate predictions which encode the distance to the target, weak learners can also be trained on sign vectors $sign(y_i - F_{m-1}(x_i))$, which record the direction (sign) of the residual vector with -1, 0, or +1. The sign vector is used to predict the median residual value instead the median average value. While regression trees are trained using sign vector values, the tree instead predicts for the median of the residuals in the leaf nodes when the sign vectors are used for grouping and splitting in the trees (Moulay et al., 2019).

The sign vectors are a component extracted from the direction vector, recording only the direction of the of the direction vector without any magnitude towards the direction. The direction vector is calculated using the partial derivative of the slope at $f(x, y)$ in an example with only 2 dimensions x and y (Moulay et al., 2019).

2.3.3 Random Forest Ensemble

A singular decision tree can grow in a vertical fashion to a size at which every single node/leaf contains only a single observation. Even prior to reaching this state, the tree has likely overfitted, becoming unlikely to predict unseen data proficiently. By limiting the depth of the tree, it can become more robust against overfitting, but leaving the tree too shallow will cause the opposite effect, underfitting (Kok et al., 2017). Nonetheless, these limitations can be sidestepped. Decision trees by themselves don't fare well when compared to other regression methods when it comes to their predictive accuracy, but methods that utilize several trees simultaneously to generate a better prediction are competitive. Decision trees are also suitable for data containing qualitative predictors outside classification tasks as they can be transformed into quantitative predictors for regression

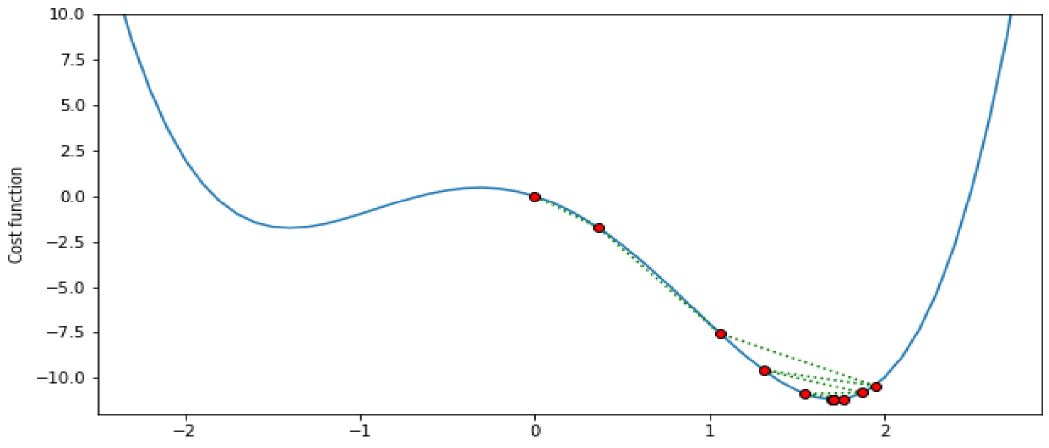


Figure 2.1: Example of gradient descent searching for the local minima on a one single variable, with 2 mimina shown. The green dotted line connects the continuous steps. The learning rate η of the descent has a decay, causing the "movement" towards the minima to keep shortening (James et al., 2021).

trees, or these predictors can just be assigned into different tree branches (James et al., 2021).

In the seminal paper on the Random Forest technique "RANDOM FORESTS", *Breiman* defines a Random Forest classifier as (Breiman, 2001):

> *"A random forest is a classifier consisting of a collection of tree-structured classifiers h(x,k), k=1, ... where the k are independent identically distributed random vectors and each tree casts a unit vote for the most popular class at input x."*

Thus *Breiman* establishes the ensemble nature of the method. Despite the quote being aimed toward explaining classification, the case for regression is similar. With regression we would have tree-structured regressors and average the results at input x. The many decision trees used in the Random Forest method are trained on random selections of the whole data set instead of the full data set (Breiman, 2001). This method of selecting the training data is called bootstrapping, known also as random sampling with replacement, as the same records can be selected multiple times into different training samples. The term bagging (bootstrap aggregation) is used when several bootstapped weak models are combined. In Random Forests, when the trees are built using bootstrapped training data, a predetermined number of randomly chosen predictors is chosen as the split candidates for the nodes in the tree from the full set of predictors. Each split only uses a single predictor (James et al., 2021), but there are several possible implementations for handling splitting with Random Forest. A random selection of input features can be used at the nodes for splits or random linear combinations of the features can be used for the splits (Breiman, 2001). As previously mentioned for decision trees, the Random Forest Ensemble method is also able to use categorical values as features as opposed to general regression methods such as linear or logarithmic regression used in hedonic regression models which must convert categorical values into in dummy values (Kok et al., 2017). The use of bootstrapped samples from the training data means that trees will be dissimilar to each other. These different trees will then convey the final prediction as an average of their given predictions, or as a weighted average of their predictions (Kok et al., 2017).

This random selection is designed to reduce the over fitting of the model by utilizing the reduction to the accuracy of the model by not having all of the records present from the training data. It also reduces the amount of sampling bias, especially when combined with having many learners (James et al., 2021). *Sagi et al.* describes ensemble learning methods to combine the outputs of several machine-learning models to create one decision, usually for a classification or regression task. Several weak learners can compensate for the errors of single models, improving the predictive performance. Thus, in our regression task with Random Forest we can average the results of all the weak rental price predictions to generate one rental price prediction that is better than those made individual weak models. According to *Sagi et al.*, their use is seen as a best practise for data sets containing features of clearly different natures, such as age or gender, compared to deep learning methods that excel in tasks related to image or sequence-like data *Sagi et al.*. That makes the use of this ensemble method apt for the data available for predicting rental prices (Sagi and Rokach, 2021). Random Forest is also resistant to over fitting due to the strong law of large numbers causing the average result to become close to the expected value (Breiman, 2001).

Breiman further elucidates that growing trees for Random Forest is fast due to there being no need to grow trees sequentially, as the trees are trained independently in parallel (Breiman, 2001). The predictor space containing all the possible values is divided into X_n distinct non-overlapping regions. Then, for all observations the same prediction region, the mean of the observations is used as the predicted response (James et al., 2021).

2.3.4 Regularization

Complex models are more prone to over fitting, so regularization methods are used to minimize the objective function which calculates loss during model training by shrinking function coefficients. XGBoost can use L1 (lasso regression) and L2 (ridge regression) regularization to prevent over fitting by reducing the complexity of the model being trained. When used simultaneously, the combination of L1 and L2 is called elastic net. L1 and L2 work by penalizing the loss function of the algorithm, thus the making the leaf nodes less likely to create new partitions in the tree (T. Chen and Guestrin, 2016a). L1 and L2 regularization for gradient boosting and XGBoost differ in that XGBoost must use Taylor approximation to transform the objective function into an euclidean function prior to utilizing L1 and/or L2. With L1 regularization, when the sum of the residuals is greater than the alpha value (L1 parameter in XGBoost), the numerator of the output value is decreased by alpha. The output value is increased by alpha when the sum of the residuals is less than negative alpha, with both cases causing the value of the numerator to approach 0. When the sum of the residuals is less than the absolute value of alpha, the output value is 0. Thus, L1 regularization will encourage sparsity by increasing the number of zeroes, removing the features that are not likely meaningful (T. Chen and Guestrin, 2016a).

With L2 regularization (hyper-parameter lambda in XGBoost), the steps towards an alpha of zero are achieved by increasing the denominator. L2 regularization will only result in an output value of zero for the weight if the minimum MSE vector contains a 0, otherwise the value will only inch closer to zero. L2 regularization ensures that features that are less important will have a smaller, yet non-zero influence on the final prediction, so it does not eliminate features. Thus the L2 does not strive towards a sparse solution like L1 does, and making L1 robust against outliers and allows it to perform feature selection by reducing feature weights to zero. These regularization methods combat overfitting by setting constraints on the fitting process (Friedman, 2001).

Random Forest Ensemble on the other hand does not include regularization as a penalty to the cost function. This is due to the effect of averaging several random trees with high variance but low bias already creates a low-bias and low-variance estimator by itself. (Breiman, 2001).

2.3.5 XGBoost

XGBoost (eXtreme Grandient Boosting) is a datamining algorithm that has been implemented as an open-source software package for python and several other programming languages. It contains an implementation of a regularizing gradient boosting framework, which is used to execute a supervised learning algorithm that belongs into the boosters algorithm category (T. Chen and Guestrin, 2016b). As previously explained with Gradient Boosting, boosters attempt to create a strong learner from several weaker learners by analyzing the predictions of the previous weaker learners. XGBoost utilizes the tree model for generating decision trees. Over fitting is prevented by using tuning parameters such as the regularization term to set a minimum loss reduction to create new partitions on the decision tree leaves (γ in XGBoost) (Mo et al., 2019). As *Chen et al.* so succinctly explain it, lowering the learning rate η, also known as the shrinkage, reduces the influence of each individual tree and leaves space for future trees to improve the model due to η scaling all newly added weights after each tree boosting step (T. Chen and Guestrin, 2016a).

Compared to traditional gradient tree boosting, XGBoost contains several optimizations that are inherent to its success. As previously mentioned, XGBoost dynamically selects the a tree splitting algorithm that is most effective for the situation based on the volume of the training sets if allowed to. For smaller data sets, the *exact greedy algorithm* will be used as every single splitting point is examined with it, but as data sets grow or calculations need to be performed in a distributed manner, a XGBoost *approximate algorithm* is employed (T. Chen and Guestrin, 2016a). Then each candidate split is evaluated with the approximate algorithm at the feature level, but no approximation happens when comparing the selected candidate features.

Distributed platforms such as Spark, Hive or Apache Hadoop can also be used to train a model with XGBoost (Sagi and Rokach, 2021).

The approximate algorithm has two variants, with the one to be selected for use depending on the proposed splits on the feature distribution percentiles. The global variant proposes all splits during each tree construction phase, while the local variant adjusts the proposal during each split. Thus, the global variant requires less proposal steps, but has a higher starting cost, while the local variant is better suited for deeper trees due to it being able to adjust the splits. With enough split candidates, the global variant can be as accurate as the local variant (T. Chen and Guestrin, 2016a).

The method through which the candidate splits are proposed is called the *weighted quantile*

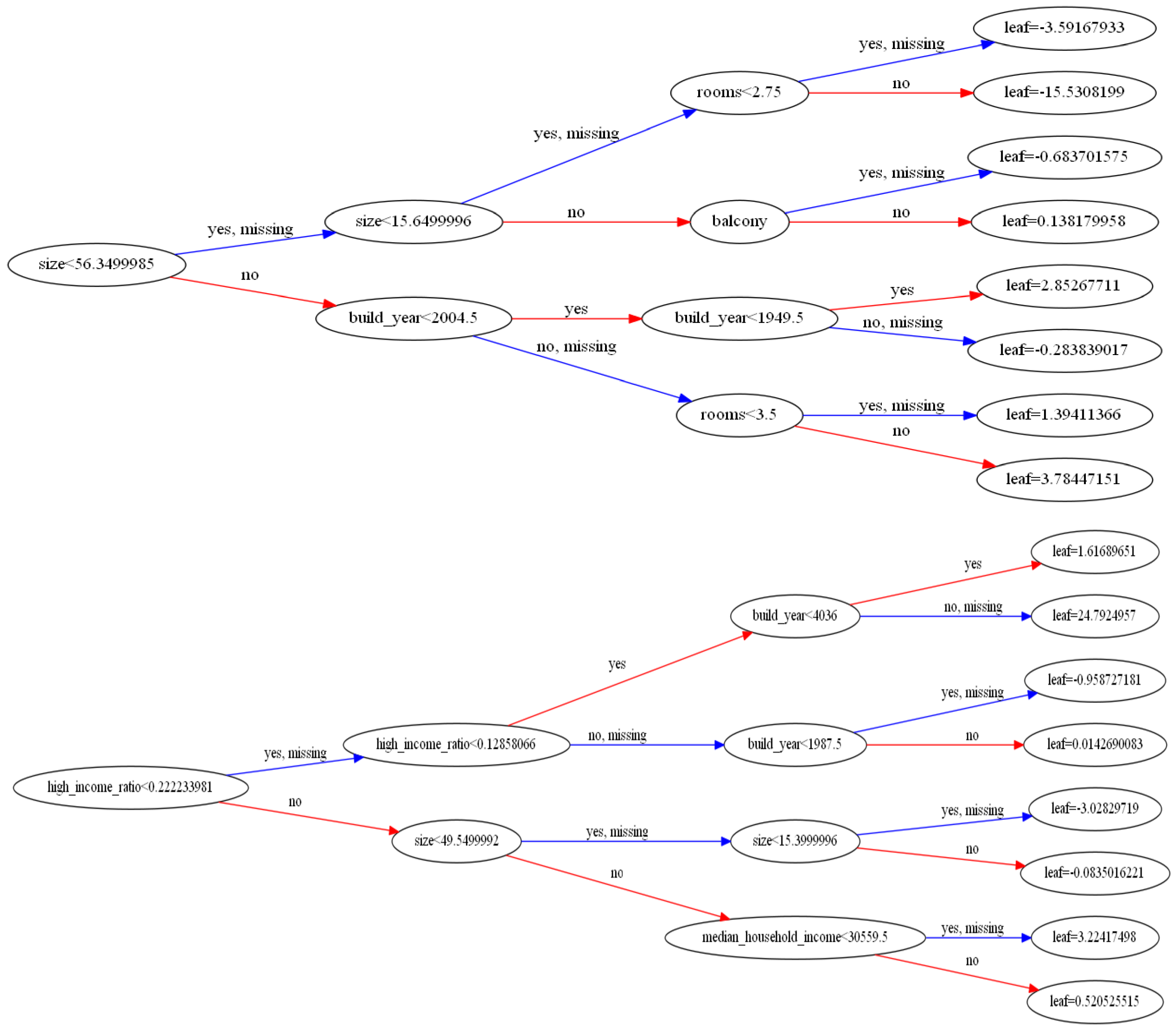


Figure 2.2: Example of two decision tree stumps from an XGBoost model, limited to the depth of 3. The final prediction for the input is the mean of all predictions from each tree.

sketch (T. Chen and Guestrin, 2016a). The method weights data points by their diagonal matrix on the Hessian, which guarantees an approximation that only grows linearly in time complexity with the number of non-zero elements in the matrix compared to the full Hessian matrix. The quantile sketch is used to approximate the distribution of values, with the eponymous quantile referring to its use of the quantiles, cutting points in a continuous probability distribution. These quantiles are the Hessians in a regression problem. The Hessian matrix is a square matrix of second partial derivatives used in optimization tasks when solving a function using Newton's method. Newton's method is used as an extension to traditional gradient descent using the stochastic gradient descent, utilizing more time to compute the direction to the solution. Gradient descent on the other hand only uses first order derivatives when updating the steepness of the descent/ascent of $f(x)$ (T. Chen and Guestrin, 2016a).

Quantiles sketch is a mergeable streaming algorithm to estimate the distribution of values, and approximately answer queries about the rank of a value, probability mass function of the distribution (PMF) or histogram, cumulative distribution function (CDF), and quantiles (median, min, max, 95th percentile and such).

In XGBoost all other values except the diagonal values in the Hessian matrix are discarded to prevent quadratic growth on the number of second partial derivatives that need to be calculated. The large error caused by approximating using only the diagonal matrix on the Hessian value is tolerated due to the nature of boosting reducing the model misfitting each round, and due to calculating the full resulting Hessian value being infeasible computationally (Zhou et al., 2019). The Hessians are stored in the CPU memory cache for quick access during execution (T. Chen and Guestrin, 2016a). The reason XGBoost must use Taylor series of the values in the Hessian is to use it as an approximation for the objective function, allowing it to evaluate better splits in the trees. The second order approximation for the loss is computationally superior, as several parts of the equation need to be computed only once for a particular split, and instead of only knowing the direction of the gradient descent $\nabla f(x^{(i)})$ at point $x^{(i)}$, the approximation points towards the minima of the loss function $f(x)$ (T. Chen and Guestrin, 2016a).

$$x^{(i+1)} = x^{(i)} - \frac{\nabla f(x^{(i)})}{h_i f(x^{(i)})} \tag{2.6}$$

2.6. Calculating the direction of the next point in gradient descent with the Hessian h_i (T. Chen and Guestrin, 2016a).

$$\tilde{\mathcal{L}}^{(t)} = \sum_{i=1}^{n} \left[g_i f_t(\mathbf{x}_i) + \frac{1}{2} h_i f_t^2(\mathbf{x}_i) \right] + \Omega(f_t) \tag{2.7}$$

$$\tilde{\mathcal{L}}^{(t)} = \sum_{i=1}^{n} \frac{1}{2} h_i \left(f_t(\mathbf{x}_i) - \frac{g_i}{h_i} \right)^2 + \Omega(f_t) + c \tag{2.8}$$

Equation 2.8, the Weighted Quantile Sketch algorithm for the scoring function of XGBoost uses weighted square loss for finding the candidate splits. g_i is the gradient (first derivative), h_i is the Hessian (containing second derivative) for the value i that is being used as an approximate weight. $\Omega(f_t)$ is a penalizing factor to the computational time complexity of the model. According to *Chen et al.*, this algorithm replaces the gradient tree approximation step of gradient tree boosting shown in equation 2.7 during approximated split finding, giving a more accurate value than could be attained using just the first derivative/gradient (T. Chen and Guestrin, 2016a). However, equation 2.8 contains a typo as the derivation from equation 2.7 is likely incorrect, but is represented like this in the paper by *Chen et al.*[*].

Another optimization that is exclusive to XGBoost compared to more traditional gradient boosting is the sparsity-aware split finding feature. It allows XGBoost to deal with missing features, extending to situations in which one-hot encoded values have been used. This feature helps XGBoost with the handling of missing data by sorting all the input data into two groups. One group contains no missing values for features, while the other group contains only records with some missing feature values. The training data group with no missing data is sorted (T. Chen and Guestrin, 2016a). The implementation of sparsity aware split finding is not without its faults, as it can mistake the lack of some values (missing values) as sparsity.

Column sub-sampling is another improvement on naive gradient boosting which is also present in Random Forest Ensemble. With column sub-sampling, a subset of all features is chosen for the training of each weak learner. Column sub-sampling speeds up execution times and prevents over-fitting (T. Chen and Guestrin, 2016a). The execution of the split finding phase in XGBoost can also be parallelized by saving data on multiple drives, resulting in a speed up when executed on several threads.

Initially all the records will have their predicted price be set to a default value of for all the prices. It doesn't matter which number is used, but 1 is typically the initial value

[*]Discussion on this, see: https://datascience.stackexchange.com/questions/10997/need-help-understanding-xgboosts-approximate-split-points-proposal, accessed 12.04.2022

for XGBoost. This value is used to generate residuals for each record by substracting the target value from this initial predicted value (T. Chen and Guestrin, 2016a).

To borrow from the definition of a increasing monotonic relationship from the XGBoost python package documentation, here is the mathematical definition 2.9:

$$f(x_1, x_2, \ldots, x, \ldots, x_{n-1}, x_n) \leq f(x_1, x_2, \ldots, x', \ldots, x_{n-1}, x_n) \tag{2.9}$$

Equation 2.9 shows a representation for an increasing monotonous relationship, which is used to constrain the target value in the model. x' has a predicted response larger or equal to that of x. An increasing constraint sees that $x \leq x'$ remains true for any value of x (T. Chen and Guestrin, 2016a).

In XGBoost, the monotonic constraint is implemented in a manner that during the tree growing phase of the estimator, each node is given a weight for that correlates with the predicted value for the target (T. Chen and Guestrin, 2016a)*.

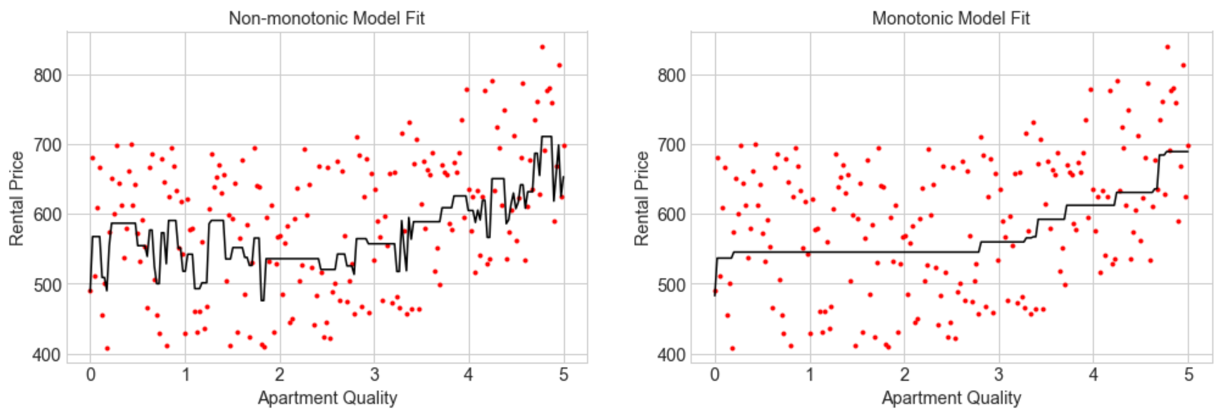


Figure 2.3: With no monotonic constraints, prices can fall despite the apartment quality increasing in this example model. With monotonic constraints, increasing apartment quality can only result in increased rental prices.

*The maintainers of the XGBoost python package request documentation citations to be attributed to the original XGBoost article, thus it is cited here and elsewhere despite monotonicity not being discussed in the 2015 article by *Chen et al.*

2.4 Interpreting a Black-box Method

How a singular decision tree works is easy to understand for a human with any level of expertise according to *Sagi et al.*, and can be explained in plain language. The hierarchical structure is also easy to visualize to a person with images, and the principle of each classification or regression end result being associated with a path in the decision tree that was used to reach that result can be shown (Sagi and Rokach, 2021). Boosting on the other hand sacrifices interpretability of the path of a decision due to having to take into account the possibly thousands trees being part of the ensemble method, but discards the inability of a singular decision tree to model complex interactions between the input features (Sagi and Rokach, 2021). In some studies, this interpretability is also known as simulatability; the degree to which the predictions of the model are comprehensible and understandable to humans. In other words, the degree to which humans can understand the cause of the decision and thus the degree to which we can consistently predict the model's result. This simulatability of a model can be of utmost importance for companies after the implementation of GDPR (*EU General Data Protection Regulation*) which forces companies to divulge how the logic of their decision making systems function that use the data collected from the EU denizens (ElShawi et al., 2021).

In the field of finance or housing, the ability to understand the model output can be critical when making a decision based on the results, as black-box methods might be snubbed despite their predictive performance. *Gradient Boosting Decision Trees* (alongside with deep neural networks) are some of the most advanced predictive methods currently, but suffer from a lack of transparency in their workings that can cause the experts relying on them to be skeptical on trusting the results (Sagi and Rokach, 2021). For example, the Google operated global influenza tracker *Google Flu Trends* was found to have trained a satisfactorily accurate model, but the black-box model could not be used to find insights related to the data (Kandula and Shaman, 2019). Thus, a solution is needed that can breach the concealed workings of the decision process by the decision forest, or ensemble. Solutions for this are ensemble pruning, in which a subset of models (or trees) is searched for which has at least the same level of accuracy as the original ensemble model, or transforming the decision forest into a single interpretable model by extracting meaningful rules from the ensemble model, bringing back the interpretability that was lost (Sagi and Rokach, 2021).

An advanced single-tree approximation method as described by *Sagi et al.* involves first pruning the ensemble trees before training the next trees when working sequentially, then extracting and evaluating conjunctive rules that are likely applied by the pruned forest model, and finally creating a new decision tree using the garnered conjunctive rules. This method is universal to all decision tree ensemble methods, and could be applied to the XG-Boost or Random Forest Ensemble models used in this thesis project to better explain the process behind the otherwise black-box methods (Sagi and Rokach, 2021). Nevertheless, the method was not used for the project, but the option to implement it remains.

Furthermore, the use of an interpretability framework called a *local model agnostic interpretability technique* can help when it comes to understanding the chosen black-box model. Model agnostic means that the methods can be applied to any machine learning model and are applied after the model is trained. Model-agnosticism is reached as the framework needs no information on the model itself, only the data and the predictions of the model (Molnar, 2022). According to *Elshawi et al.*, currently favoured frameworks usable in a *python* environment are LIME, Anchors, SHAP, LORE, ILIME and MAPLE. The frameworks have differing execution speeds, levels of bias and similarity, but serve the same purpose (ElShawi et al., 2021). Due to the project only testing LIME and SHAP, only they are looked into in the following paragraphs.

With LIME (Local Interpretable Model-agnostic Explanations), initially a prediction instance is selected that we are interested to have an explanation for, then the data set used to train the model is perturbed around the decision boundaries for our interested features (ElShawi et al., 2021). New predictions are made with the perturbed data, weighing the proximity of the altered features to those of the original feature. A new interpretable model, for example utilizing linear regression, is trained on the perturbed data set, which is then used to explain the prediction by interpreting the non-black box model (Molnar, 2022).

SHAP on the other hand uses the Shapley value from coalitional game theory designed to assess how large a payout each game *"player"* receives in a fair system. When assessing black box machine learning models, it calculates the proportional weight of each feature for a specific prediction from the model. These values for features can be positive or negative, and sum up to the value of the prediction. According to *Molnar*, the advantage of using the Shapley value that is utilized in the SHAP framework is that it is likely the only framework that is legally compliant with GDPR's *“right to explanations”* due to the theory behind the method having a more solid foundation. The unfortunate issue with the Shapley value is that the execution time is very long due to the need to assay all 2^k

possible game theory feature coalitions (k = number of dependant variables), although the SHAP framework has modifications to make this process faster. Explanations using the Shapley value also always use all the features (Molnar, 2022). The SHAP values for features chosen for the best performing model in this project can be referenced in Figure 4.2.

$$I_j = \frac{1}{n}\sum_{i=1}^{n} |\phi_j^{(i)}| \tag{2.10}$$

Shown in equation 2.10 is the absolute Shapley value ϕ for each feature j (n is the number of features) being averaged on a per feature basis to calculate the global feature importances I_j (Molnar, 2022).

3 Materials and Methods

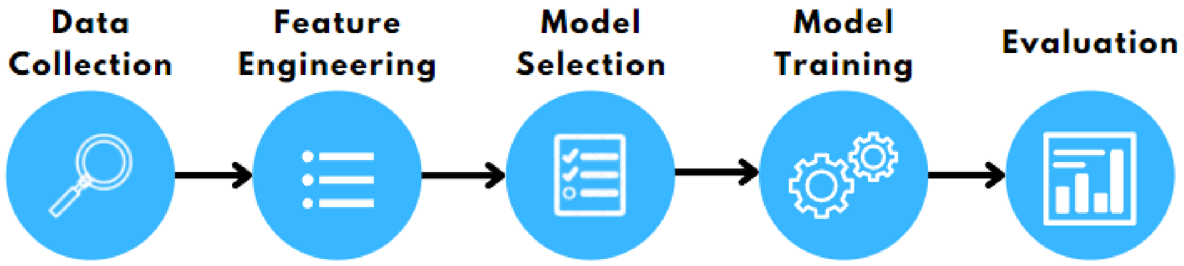


Figure 3.1: Model implementation pipeline. Model versions with significant improvements to the previous versions were made testable by the client.

3.1 Data Overview and Data Preprocessing

The rental data that was used to generate the predictive models was acquired by the client from Oikotie Oy. The raw data was not suitable for building predictive modelling as is, and had to be transformed in several ways for it to be satisfactory for training the needed models. Explorative data-analysis was the initial step before beginning to build the regression models. This stage helps to identify patterns in the data that can be used to choose the most suitable machine learning methods to employ.

Additional data was also collected from external sources. Statistics Finland (Tilastokeskus) collects demographic data for Finland and makes it available through their *Paavo-API*, which can be polled for data grouped by postal zip codes. The *high income earner ratio* and *median house income* from each postal area in Greater Helsinki was added as a feature for each apartment based on their location.

Due to the data being user input, many of the attributes were missing from the apartment sales adverts as the broker Oikotie does not require filling all possible fields. For most of the apartment adverts with missing data, the missing data was not limited to only one feature. The apartment rental and purchase advert information submitted by the users of *www.oikotie.fi* was made available for this thesis from the beginning of 2018 in totality, but we also had a smaller sample of data from the years 2010-2017 that had been given by Oikotie Oy prior to purchasing the formerly mentioned data. This additional data was also used to train the models, bringing the total of usable data points from Helsinki, Vantaa and Espoo to 203213 before further data processing which reduced the amount of data to 190819. Missing data consisted mainly due to the lack of the *build year* and *apartment condition* fields.

The rental price for an apartment in the Oikotie service was also not necessarily the price that would be agreed upon by the renter and rentee, as the used in the study is the price shown in the rental property advert. Using data from 1984–2010, *Cannon et al.* found that on average prices appraised by real-estate real estate agent were 12% above or below sales prices in the United States of America (Cannon and Cole, 2011). This does not mean that the same stands for rental prices, but even so it should be taken into consideration. Future experiment design could include a follow-up in which the true rental prices of a number apartments are ascertained to calculate the average appraisal error to be used during feature engineering (Cannon and Cole, 2011).

The Oikotie API* could be polled for over 60 useful non-metadata attributes that could be examined for each apartment advertisement. Meta-data such as when the advert was published and removed from the broker-site or how many page views the advert garnered during it's up-time could not be used as parameters for the model, as these are not general features of apartment units that are available when estimating the price of an apartment outside the data set. Still, the *publish date* of the advert could be used to adjust the prices of older records using the compounding interest formula with quarterly interest. With this change, the price of an apartment from 2018 could be updated to a likely price of the apartment in 2021.

Initially one-hot encoding was used for some of the categorical data, such as city (Helsinki, Vantaa, Espoo), and apartment condition. In the end, the model performed better with a smaller footprint when dropping features generated by one-hot encoding for non-general regression models.

Thus, categorical data was changed to numerical representations when applicable for easier handling with the random forest algorithm. A scheme to score *apartment conditions* from 0 to 13 was devised together with the thesis client. Leaning into their extensive domain knowledge, they suggested scoring the apartment conditions at intervals from 0 (huono / bad), 4 (välttävä / passable), 8 (tyydyttävä / adequate), 9 (hyvä / good), 12 (erinomainen / excellent), 13 (uusi / new). According to the client the distinction between an adequate

*Oikotie *swagger* URL: https://api.asunnot.oikotie.fi/

and good apartment is very small in practise, and warranted only a small increment between them when scoring the apartments. The same held for apartments in excellent or new condition. The whole apartments data set only contained 13 instances of bad quality apartments, with most apartments being in the adequate and good range.

Initially we tested *mean substitution* as method for imputation, but after some concerns that bias was being introduced to the data, list-wise deletion, in which records with missing data were removed, was used for the values we believed to have the largest impact if altered to contain a mean value of similar records. This reduced the amount of data available, but the losses were deemed acceptable (203213 → 190819 records). The more apartments there is in the training data for any particular set of apartment features, the better the estimate for the true rental price is that we can reach with the particular set of features, especially in the value ranges expressed in the data (Park and Bae, 2015). Compared to similar studies, such as *"Using machine learning algorithms for housing price prediction: The case of Fairfax County, Virginia housing data"* by *Park et al.* that had only 15135 records, we are better positioned to create an accurate model (Park and Bae, 2015).

Some data such as information on future renovations (*renovation_future_info* in the API) could have been very important when it came to determine the rental price, although less important than it would be for a selling price. This field from Oikotie was a free text field, so it could contain long rambling strings that didn't share any specific structure. Deciphering the field could have lead to better gains, but time was not spent on this for this study as cleaning the data would have taken a lot of effort with no certainty of gains to prediction accuracy.

Another field that could also be used in the future was the *publish length* that denoted how long the advert had been shown in the Oikotie website. This could not directly be used for training, but could act as a tell on whether some apartment rental prices had been set too high by the renter.

In the Oikotie data, some values can also be justified to be *missing not at random*, such as an apartment missing a boolean flag on their balcony or sauna (Mack et al., 2018). If the apartment had a sauna or balcony, it would most likely be divulged by the leaser as this would increase the amount of rent that could be charged for the apartment property. Thus, missing values in these situations are imputed as a false value, as the apartment is certainly missing the amenity.

When it came to choosing the features for training, a lot of empirical testing with different features from the Oikotie API data was done initially, but specialized tools like the *pandas_profiling* exploratory data analysis package were used as well to map correlations between features and the target value. Another good starting point for choosing features was examining what features companies working on the housing price estimation problem in Finland had disclosed they were using. Articles by *Sirmans et al. (2005), Guntermann et al. (1987)* and *Park et al. (2015)* also served as material that helped choose features. Kok et al. also guided discussion after the experiments with their observations on apartment features (Kok et al., 2017). The features that were being examined could be divided into physical features of the housing unit, and features of the area the housing was located at. *Sirmans et al.* had previously done research on hedonic pricing models, and according to their meta study into the features used in previous studies, they and *Guntermann et al.* denote the following to be important "Internal House Features" with significant positive effect on the dependant variable price; *room count, fireplace, air-conditioning, central heating, bathroom count, full baths, basement, dining area, wet bar.* "External House Features" that *Sirmans et al.* show to be important are *garage spaces, decks and pools* (Sirmans et al., 2005).

The presence of any of these features were noted to have an average increase of 3-18% per feature to the price of the apartment in the USA. Features of the neighbourhood and environment such as *golf courses and trees* had a significant positive impact on the price, while *crime and schools* a mainly negative effect. Other factors that *Sirmans et al.* identified to have a large effect on the price were *location, apartment condition, the unemployment rate in the neighbourhood, time on market* for the housing unit and whether the apartment was being *sold for cash.* (Sirmans et al., 2005). *Park et al.* on the other hand reiterate the importance of location based features as the most important factors that determine the target value price (Park and Bae, 2015). *Guntermann et al.* note that some features that can raise the price of an apartment are not easily quantifiable or can have proxies created for them, such as the value of privacy at a unit (Guntermann and Norrbin, 1987).

Guntermann et al. list the following as physical characteristics as important features in their research into hedonic pricing models; apartment unit size, bedroom count, number of bathroom, age, condition (Guntermann and Norrbin, 1987). As environmental features, they mention accessibility to the central business district and other location considerations. Guntermann et al. also note that for some apartments the rental price includes the price of the utilities, affecting the rental price (Guntermann and Norrbin, 1987). This was something that we could not determine from our data without investing a large amount of time due to there being no field for it in the Oikotíe data.

Not all of these are applicable to apartment housing in the Greater Helsinki area, but the findings could help design future apartment brokerage listings or machine learning models.

When selecting hyperparameters for the training phases of the models, initially optimal parameters were searched with the *sklearn RandomizedSearchCV* module that allowed testing different parameters within a wide range, after which the *GridSearchCV* module was used from the same *sklearn* package to further search for the best parameters near those found using the randomized search (Buitinck et al., 2013).

GridSearch eased searching for decent values for the learning rate (η, the weighting factor for new trees which slows how fast learning in a gradient boosting model), after which the tree parameters were tuned (maximum depth, minimum child weight, gamma (minimum loss required to create a new partition on a tree leaf node), subsample ratio, column sample by tree). The optimal η was found to be 0.05, while the other parameters that were used to train the final model were: maximum depth: 22, alpha (L1 regularization conservative value): 15, number of estimators: 150, gamma γ: 0.1, subsample ratio: 0.906, min_child_weight: 0 (T. Chen and Guestrin, 2016a). The monotonous constraints were also given as a parameter for the final model training. MAE was used as the evaluation metric and linear regression (squared error to determine how far prediction is from the regression line) was objective of training due to this being a problem for which we are predicting a continuous output value.

For tuning the hyperparameters prior to implementing GridSearch, initially the *Optuna* hyperparameter optimization framework was used to test the first manually tuned values, which helped down-size the tree-sizes for the Random Forest and XGBoost models by recommending less deep trees (Akiba et al., 2019). *HyperOpt* was also tested, but returned a similar result to Optuna. The *hgboost* python module fared worse than the previous two, but suggested that an improvement could be reached by creating an ensemble with XGBoost, CatBoost and LightGBM when it came to the RMSE of the model. *GridSearchCV* was used to test the hyperparameters adjacent to those used in the final XGBoost model. All of the models were trained with the following hardware using *Google Cloud Platform Jupyter Notebooks* with the CPU *Intel Xeon 2.20 GHz*, GPU *Tesla K80 12 GB* and 12 GB of RAM. Nonetheless, a package was created to run the the full data collection, pre-processing and training process as a script within a *python* environment outside the Notebook environment.

The *Google Cloud Platform Vertex AI*[*] was also tested, but the system could not generate a more accurate model with our data in 8 hours of execution time than our current model at the time. It is possible that it could have achieved better results if allowed to execute as long as needed, but there is running cost for the execution.

The *mljar-supervised* Automated Machine Learning Python package was also used to test our results against a reliable regression task baseline with the following ensemble of algorithms in AutoML: Baseline, DecisionTree, Default_XGBoost, Default_NeuralNetwork and Default_RandomForest, with AutoML This ensemble reached a RMSE of 116.23 training, while our solution using only XGBoost reached a RMSE of 112.6830 and MAE of 49.4883 with only XGBoost.

Features - Data Overview		
Attribute Name	Data Type	Description
zip_code	string	postal code of apartment unit
apartment condition	float	6 different values expressed as numbers between 0 to 13
size	float	size in m^2
sauna	boolean	has sauna in apartment
rooms	int	room count
median_household_income	float	median household income in postal area
build_year	int	year apartment building built
interpolated_mean_price	float	mean price of n nearby apartments
interpolated_mean_price_sqm	float	mean price per m^2 of n nearby apartments
high_income_ratio	float	ratio of denizens in postal code with high income
balcony	boolean	apartment has balcony
first_floor	boolean	apartment on first floor

Table 3.1: List of attributes used for final model training.

[*]https://cloud.google.com/vertex-ai

3.1.1 Feature Importance

Feature importance has been calculated by counting the amount of times the feature was used in a split within a decision tree (T. Chen and Guestrin, 2016a) and with the use of the SHAP framework. When analyzing the feature importance chart from XGBoost in Figure 3.2 and Figure 4.3, the top 5 features are not unexpected. The size of the apartment is the most important feature when it comes to determining the rental price of the apartment. Unsurprisingly, as the size of the apartment increases, so does the rental price. *Guntermann et al.* also corroborate that the size has great explanatory power in hedonic models, and is highly significant (Guntermann and Norrbin, 1987). The next two most important features have been generated using inverse distance weighting, with the value given to the record of an apartment being interpolated from the *price per* m^2 of the closest 8 and *mean price* of the closest 8 apartments. The interpolation method is described in detail in it's own section. These two features give an estimated rental price for the coordinates of the area being examined. The rental prices of apartments around the estimated apartment are very important when estimating the price of a new apartment. It is likely that the location of the apartment would have a larger effect on the rental price if the model training data contained locations outside the greater Helsinki Metropolitan area. Apartments outside the large cities could have made the coordinates a more important feature than the size of the apartment, as far larger apartments can be rented for cheaper outside the capital. The categorical feature *zip code* was also an important explanatory value as it is another location based feature.

The *build year* of the apartment has a large effect on the rental price according to XG-Boost splits, and is likely to follow a trend in which newer apartments have higher rents, unless the location of the older apartment is more prestigious or has nearby services that haven't been taken into account in the model features. According to *Guntermann et al.*, standard appraisal practises use chronological age as a proxy for apartment depreciation. Nevertheless, the effect of age is less important as the apartment condition also has its own feature, and the age is less important than the perceived condition of the apartment. The *apartment condition* feature is completely subjective, but is logically related to the age. The findings of *Guntermann et al.* were at odds with the *apartment condition* feature importance being higher than the effect of the chronological age. Their findings showed that age had little significance, while age held more significance in our model than the apartment condition with both SHAP and split based feature importance (Guntermann and Norrbin, 1987).

Prior to setting the monotonic constraints, increases to the *build year* feature did not increase the price of an apartment in a linear fashion, but had dips for certain year ranges at some locations.

The *sauna* feature has a relatively low feature importance despite a sauna being an expensive fixture for an apartment. This is likely due to a sauna being uncommon within smaller apartments, and the net effect of a sauna on the price of the apartment being less than changes on the size of the apartment. The presence of a sauna could be considered a predictor for the *size* of the apartment, through which it can predict the target value rental price, but with the *size* known, the sauna feature is less useful for estimating the rental price.

Features that were not part of this project, but would likely be helpful when training a model are closeness to universities as *Guntermann et al.* mention that universities act as focal points for increased rental prices, and that as distances from universities increased, rental prices decreased. Thus it could be prudent to include metrics that measure their distance to apartment units (Guntermann and Norrbin, 1987). *Kok et al.* also emphasize the *closeness of greenspaces and schools*, the *delinquency rate* of the area, *median income* and *land value* as important explanatory values, although their study used the metric *return of investment* on real estate rather than rental price (Kok et al., 2017).

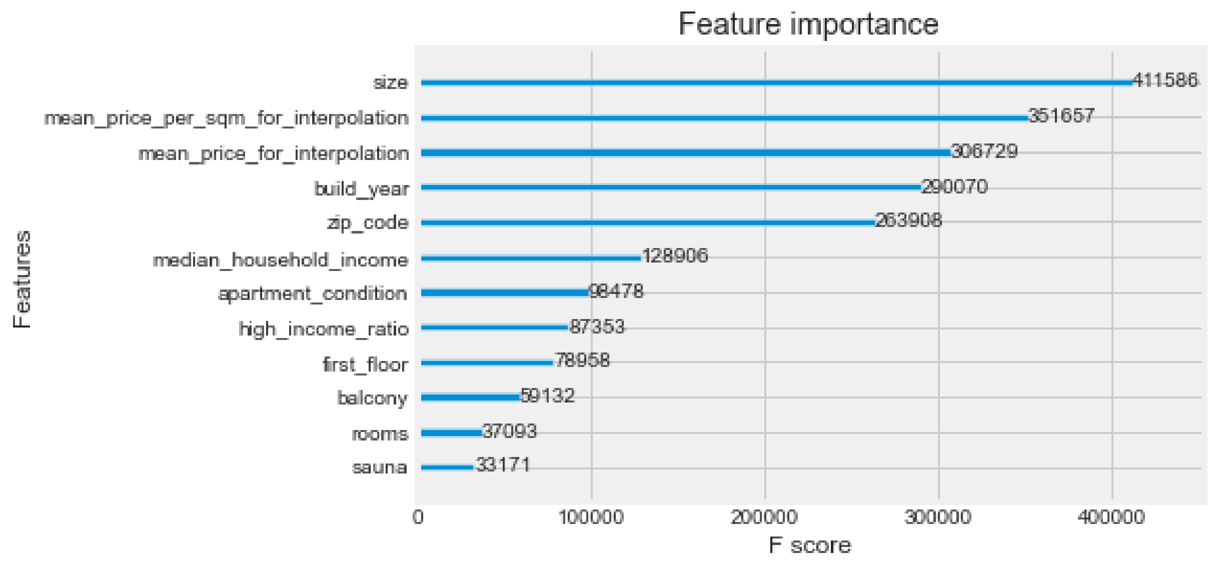


Figure 3.2: Feature importance chart generated using the XGBoost package *plot_importance* function, matching used features to their significance to the model accuracy. The F-Score in this figure denotes how many times the decision trees were split using the variable. This frequency metric is calculated from the model tree map.

3.1.2 Location Data and Inverse Distance Weighting

When it came to using coordinate data for training and predictions, several different methods were tried. Initially, we set to divide the greater Helsinki area into smaller subsections, bagging all the data in each into 100m^2 to 250m^2 squares covering the whole area. The limits of the Helsinki-Vantaa-Espoo area were set to be between latitude 60.077503 and 60.402547, between longitude 24.5041 and 25.252731 to prevent apartments not within the official Helsinki, Vantaa and Espoo borders being included in the training. Apartments outside these limits needed to be culled, as some users of the Oikotie broker had mislabeled apartments outside the city limits to be part of Helsinki, Vantaa and Espoo. Calculating the mean prices within each square gave an estimate for the prices of apartments within a certain latitude and longitude, allowing the generation of a rental price heat map. Nevertheless, this method was inefficient, as the sizes of the bags should have varied according to the densities of areas in greater Helsinki. Many bags were completely without data, while some large construction projects had clearly attributed all apartments built to a single coordinate point for hundreds of apartments, causing many apartments being sold to have the same coordinates, preventing their data from being in neighbouring bags in the location matrix.

Dividing areas into price areas was also considered, using GeoJSONs and GeoPandas. This approach was too time-consuming due to the large amount of manual work, so it was abandoned. The use of the Gaussian Process Regression (also known as Kriging) method (Schulz et al., 2018) was also considered , but the gains from implementing it were deemed not much better than those gained from inverse distance weighting (Lu and Wong, 2008).

As previously mentioned, good results were received using inverse distance weighting (IDW) as described by Lu et al. in their work in *An adaptive inverse-distance weighting spatial interpolation technique* (Lu and Wong, 2008). The previously mentioned city limits were still used. IDW is a method to interpolate new multivariate values using a scattered set of points. IDW assumes that the similarity of any given points in a plane is related to the distance between the points. Thus, the closer a point is from another, the larger the spatial relationship between the points should be, while the inverse holds true the further these points are from each other (Lu and Wong, 2008).

At the city scale we can work with a Cartesian coordinate system, not having to take into consideration the curvature of the Earth. Thus, the distances between two points are calculated using a basic distance formula instead of using the Haversine formula to calculate the distance of two points on a sphere. As the value being interpolated was the rental price of an apartment at a specific coordinate, the new price was generated using the rental prices of the surrounding apartments, giving more weight to closer apartments. The weighting would prevent apartments in unpopular areas to have their rental prices dictated by housing units too far away. Results suggested that taking into account the rental prices of the closest eight apartments gave suitable values. IDW thus allowed the use of the target value *rental price* for prediction as a feature in a manner of speaking.

Methods similar to our implementation of inverse distance weighting were employed in hedonic pricing methods as well according to *Kok et al.*, as this method often included steps in which a radial search was made to scrutinize the attributes of housing near the estimation target (Kok et al., 2017).

```python
from scipy.spatial import cKDTree as KDTree
import numpy as np

def inverse_dt(X, z, leafsize=10, q, nnear=8, eps=.1, weights):
    # X:training data coords, z:training target (rental price), q:new
  query coords, nnear: number of nearest apartments
    tree = KDTree(X, leafsize=leafsize)  # build the tree
    distances, ix = tree.query(np.asarray(q), k=nnear, eps=eps)
    interpol = np.zeros((len(distances),) + np.shape(z[0]))
    iteration = 0
    for dist, ix in zip(distances, ix):
        if dist[0] < 1e-10:
            wz = z[ix[0]]
        else:
            w = 1 / dist**2 # inverse power of distance
            if weights is not None:
                w *= weights[ix]
            w /= np.sum(w)
            wz = np.dot(w, z[ix])
        interpol[iteration] = wz
        iteration += 1
    return interpol
```

Listing 3.1: Inverse Distance Tree implementation using numpy and scipy cKDTree

The method for inverse distance weighting is similar to k-nearest neighbours, but with the addition of weights.

3.2 Evaluation Metrics

This research uses the commonly established evaluation metrics MAE (mean absolute error) and RMSE (root mean squared error) to evaluate the performance of the model (James et al., 2021), and served as the main cost, or loss functions in the machine learning models this study. MAE also serves as the key performance indicator for this study (Friedman, 2001) Both methods are residual standard error calculation methods (James et al., 2021).

MAE is a metric to denote the similarity of two sets, derived from finding the error between the corresponding values in the sets and finding the mean of found errors, which is then divided by the sample size. RMSE is similar to MAE, except the absolute errors are squared before they are summed, and the result is square-rooted after dividing by the sample size. The calculations for the metrics are shown in the equations 3.1 and 3.2 (Y. Chen et al., 2021). Generally, if the RMSE of the test data is larger than the RMSE of the training data, then we have over fitted the data, and when the RMSE of the test is smaller than the training, then the data has been under fit. With a good model, the RMSE of the test data is similar to the RMSE of the train data set. With RMSE, the outliers in the data have a greater effect on the metric due to the squaring, while MAE does not pronounce the effect of outliers due to not squaring the residuals (Willmott and Matsuura, 2005).

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - F_m(x_i)| \tag{3.1}$$

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\Big(y_i - F_m(x_i)\Big)^2} \tag{3.2}$$

3.3 Client Needs

The client expected a software system that would be able to predict the rental price of a housing unit when given information pertaining to the unit. The predictions from this tool and 2 other commercial systems would be used by their real-estate analysts to compliment their own analyses. When the prices suggested by the models would deviate sufficiently from the analysts own estimations, the real-estate analyst can delve into the reasons for the discrepancies in the suggested prices by the predictors and appraise whether real-estate rental prices need to be altered.

For cross-validation, the data set was split into train/test groups using a 80/20 split. Due to the large data set available for this project, the split used was adequate. A stratified data splitting method where the test sets contained a more diverse selection of apartments was tested using the *StratifiedKFold* splitting method from the *scikit-learn* python package and dividing the apartments into sets using their coordinates, but found unnecessary (Buitinck et al., 2013).

The Google Geocoding API* is used to identify the coordinates for apartment addresses from the provided data, which would then be used by the inverse distance weighting method to interpolate a rental price for the coordinate based on the prices of the surrounding apartments.

Prior training the models, rental prices were adjusted to the current rental apartment market prices by using the custom quarterly compounding interest formula 3.4, with the amount of interest for each quarter is was taken from Tilastokeskus from their open data on rent changes per quarter for Helsinki, Espoo and Vantaa. The postal areas of each city have also been divided into expensiveness areas by Tilastokeskus, for which more granular data exists when it comes to the quaterly increases in rental price compared to the previous quarter. This data extended back to 2015 Quarter 2.

$$A = P\left(1 + \frac{r}{100}\right)^{n} \tag{3.3}$$

Seen above in equation 3.3 is the compounding interest formula. P is the principal amount, A is accumulated interest + the principal, r is the rate of interest, n is the number of interest periods. In our implementation, r changes depending on the quarter data from Tilastokeskus.

$$A = P\prod_{i=1}^{n}\left(1 + \frac{a_i}{100}\right), a \in \mathbb{Q} \tag{3.4}$$

The compounding interest formula 3.4 is used in our code. $\boldsymbol{a}$ is the sequence of percentile price changes for the city and price region for the apartment since the apartment was

*https://developers.google.com/maps/documentation/geocoding published on the brokerage site. An apartment added 5 quarters ago could have an $\boldsymbol{a}$ like {0.3, 0.2, -0.1, 0.4, 0.2}, which would be used to calculate a new adjusted rental price.

```
def calculate_cumulative_interest(df):
    # get quaterly changes dataframe, remove data for which there is no
   quaterly interest information (prior to 2015Q2).
    df.drop(df[df.quarters < (df_quaterly_stats.shape[1]-2)].index)
    df = df.apply(lambda x: add_adjusted_rent(x, df_quaterly_interests.
   loc[df_quaterly_interests['town_area'] == x.town_area]
    return df

def add_adjusted_rent(single_apartment_df, interest_list):
    principal = apartment_df['rental_price']
    for quarter in interest_list:
        interest = 1.0 + (quarter*0.01)
        principal += (principal * interest) - principal
    apartment_df['adjusted_rent'] = principal
    return apartment_df
```

Listing 3.2: Adjusting Prices for Compounding Interest

The working models were integrated into a *Streamlit.io*[*] front-end separate from the training processes, and dockerized to be served online using *Google Cloud Platform* containers[†] for the client to test their use through a browser interface. An image of the application can be seen in Figure 4.5.

[*]https://streamlit.io/

[†]https://cloud.google.com/containers

4 Results

XGBoost was found to be the most performant algorithm for modelling rental price prediction and fulfilled all client needs. Using the hyperparameters that were found as the best for each model, XGBoost performed the best with with a MAE of 49.49 as seen in Table 4.1. Compared to the second best option, Random Forest Ensemble (MAE of 50.77), the difference was small. Of note is that when setting a monotonic constraint on the *apartment quality* feature, XGBoost was still more performant than Random Forest Ensemble with a MAE of 49.92 as seen in Figure 4.4. Due to the XGBoost package having a ready made implementation of adding monotonic constraints to building decision trees, XGBoost with monotonic constraints was a natural pick as the project focus as the feature was a client requirement after the lack of monotonic constraints was seen to cause an inconsistency in pricing apartments. Multivariate linear regression and plain Gradient Boosting are shown to further establish the baseline that the models were compared against.

Model	MAE	MSE	RMSE	R^2
Linear Regression	93.9056	32192.3707	179.4223	0.7162
Random Forest Regressor	50.7696	13390.5756	115.7177	0.8819
Gradient Boosting	71.3259	20189.6374	142.0902	0.8220
XGBoost	49.4883	12697.4609	112.6830	0.8880
XGBoost monotone constraints	49.9231	13154.0355	114.6910	0.8840

Table 4.1: Model Performance with metrics. Variance R^2 is also shown, describing the model robustness (Kok et al., 2017), explaining how well the model would perform with outliers and unseen data similar to the training data.

Prediction precision is better at the rental price ranges for which there is the most data. Most rental prices fall between 600 and 1300 euros per month as seen in 4.1. Inaccuracy increased when the model needed to make predictions on apartments that were outside of this range. From Figure 4.2, we can see the SHAP framework given feature impacts on the XGBoost model with monotonic constraints. The SHAP values were computed on a smaller 10000 observation sample of the data set to reduce the execution time of SHAP.

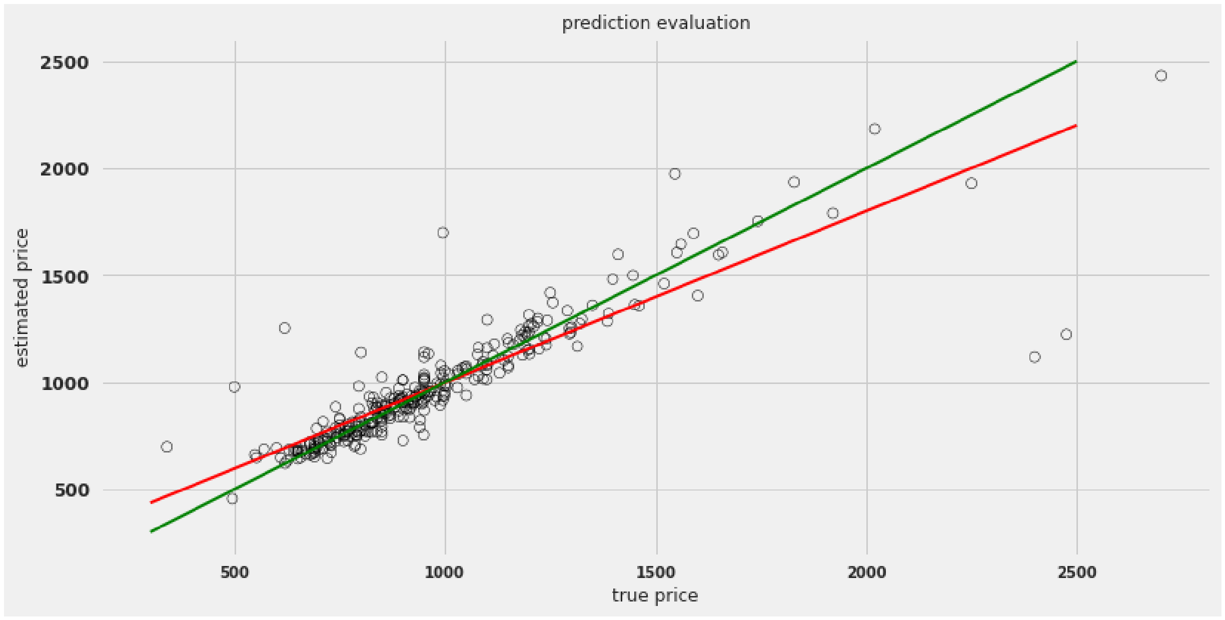


Figure 4.1: The closer the prediction is to the green line, the more closer it was to the true value y. The red line is the best fit line created with estimated and true values. A random sample of 300 apartments from the test set is shown in the figure as the circles, but lines represent the full test data.

It is meaningful to note how the feature importance based on the number of splits in XGBoost decision trees shown in Figure 3.2 differs compared to SHAP's average impact on model output magnitude seen in Figure 4.3. In SHAP, *size*, *interpolated prices* and the *zip code* of the apartment can be seen to contribute the largest swings to the target value price. The first floor feature has the least impact.

Of special interest is the differing importance placed on the *room count* and *build year* shown by the different methods to gauge feature importance. *Room count* is not used for many splits in the decision trees, but has a relatively high importance given by SHAP. The *build year* feature has more splits than the qualitative feature *zip code*, but it has a low SHAP value.

4.1 Comparing Model Performance

To test the performance of the model, the client selected a sample of new apartment listings that the model has not been trained on. The population from which the samples are taken from is apartments units within Helsinki, Espoo and Vantaa. A sample size of only 26 apartments was tested, as the client needs to manually test the sample performance on a subscription based service for each apartment. The models were tested on the week of

29.10.2021.

Table 4.2 contains features as well as predicted rental prices for the apartment units tested on commercially available models and the models created for this study. The commercial models by *SkenarioLabs* and *Vuokraovi* (AlmaVuokra) were tested and their results are recorded in Table 4.2. It was found that the *SkenarioLabs* model was on average 8.03% off from the real estate agent given prices. *AlmaVuokra* had a mean difference of 7.46% from the real estate agent given prices. Our best model with monotonic constraints applied was on average 7.78% off from the real estate agent given price. In another set of prediction, the values were 5.70% for *SkenarioLabs*, 4.96% for *AlmaVuokra* and 5.37% for our best XGBoost model. The calculation to determine how far off all the predictions were was a simple averaging of the percentile differences of single predictions compared to the real estate agent provided true values.

4.2 GUI for Model Testing

One of the goals of the project was to provide the client a way to test and evaluate the work that was performed for the thesis. A range of the models developed during the project can be tested with a single page web application created for the purpose using *Steamlit.io*. Figure 4.5 shows the layout of the page. The tool allows the selection of the model, after which the page updates the fillable form to include those features that need to be provided to the model for a prediction to be made. When the button named "estimate rent price" is clicked, the estimate is provided under the button alongside with the full address taken from the *Google Geolocation API.* The API is also used to retrieve the coordinates of the selected address, and the coordinates are then used to find the prices and price per m^2 of the 8 closest apartments. These prices are used by inverse distance weighting to interpolate a price for the current coordinates which is used as a feature to make the prediction. An Excel or .csv file with features can be uploaded for the selected model to estimate rents for more than one location at a time, with the predictions added to the Excel or .csv file that was uploaded as a new column. The tool is hosted on a *Google Cloud Platform* container.

address	city	size	sauna	rooms	condition	build year	floor	floors count	market price
Isonnevantie 6 B	Helsinki	21	0	1	4	1966	3	5	685
Seljatie 2	Helsinki	20	0	1	3	1965	3	5	690
Bahamankatu 4	Helsinki	32	0	1	5	2021	2	6	935
Bahamankatu 4	Helsinki	37	0	1	5	2021	1	6	980
Bahamankatu 4	Helsinki	49	0	2	5	2021	6	6	1260
Bahamankatu 4	Helsinki	61	0	2	5	2021	5	6	1440
Kalliolanrinne 4	Helsinki	25	0	1	5	2020	3	6	842
Kalliolanrinne 4	Helsinki	31	0	1	5	2020	2	6	882
Kalliolanrinne 4	Helsinki	55	0	2	5	2020	3	6	1435
Työpajankatu 6	Helsinki	25	0	1	5	2021	5	12	820
Työpajankatu 6	Helsinki	35	0	1	5	2021	8	12	970
Työpajankatu 6	Helsinki	39	0	2	5	2021	4	12	1090
Työpajankatu 6	Helsinki	58	0	2	5	2021	3	12	1320
Työpajankatu 6	Helsinki	59	0	3	5	2021	8	12	1600
Työpajankatu 6	Helsinki	90	0	4	5	2021	2	12	2100
Ylismäentie 12	Espoo	28	0	1	5	2021	3	8	785
Tulkinkuja 3	Espoo	21	0	1	5	2021	7	8	725
Tulkinkuja 3	Espoo	28	0	1	5	2021	3	8	770
Unikkotie 5	Vantaa	24	0	1	5	2021	4	8	745
Unikkotie 5	Vantaa	29	0	1	5	2021	2	8	790
Unikkotie 5	Vantaa	47	0	2	5	2021	4	8	995

market price	**our XGBoost**	**% difference**	**SkenarioLabs**	**% difference**	**AlmaVuokra**	**% difference**
685	684	-0.2 %	706	3.0 %	670	2.2 %
690	684	-0.8 %	651	-6.0 %	676	2.1 %
935	995	6.1 %	955	2.1 %	1010	7.4 %
980	988	0.8 %	1027	4.6 %	1050	6.7 %
1260	1225	-2.8 %	1373	8.2 %	1320	4.5 %
1440	1353	-6.4 %	1516	5.0 %	1440	0.0 %
842	853	1.3 %	879	4.2 %	859	2.0 %
882	944	6.5 %	972	9.3 %	953	7.5 %
1435	1224	-17.2 %	1418	-1.2 %	1470	2.4 %
820	871	5.8 %	954	14.0 %	882	7.0 %
970	982	1.2 %	1053	7.9 %	1050	7.6 %
1090	1229	11.3 %	1239	12.0 %	1220	10.7 %
1320	1287	-2.6 %	1571	16.0 %	1500	12.0 %
1600	1441	-11.0 %	1882	15.0 %	1590	0.6 %
2100	2291	8.3 %	2262	7.2 %	2070	1.4 %
785	787	0.3 %	748	-4.9 %	756	3.8 %
725	630	-15.0 %	704	-3.0 %	701	3.4 %
770	772	0.2 %	777	0.9 %	756	1.9 %
745	706	-5.5 %	711	-4.8 %	704	5.8 %
790	781	-1.1 %	749	-5.5 %	757	4.4 %
995	931	-6.9 %	977	-1.8 %	924	7.7 %
MAPE %	5.37%		5.70%		4.96%	

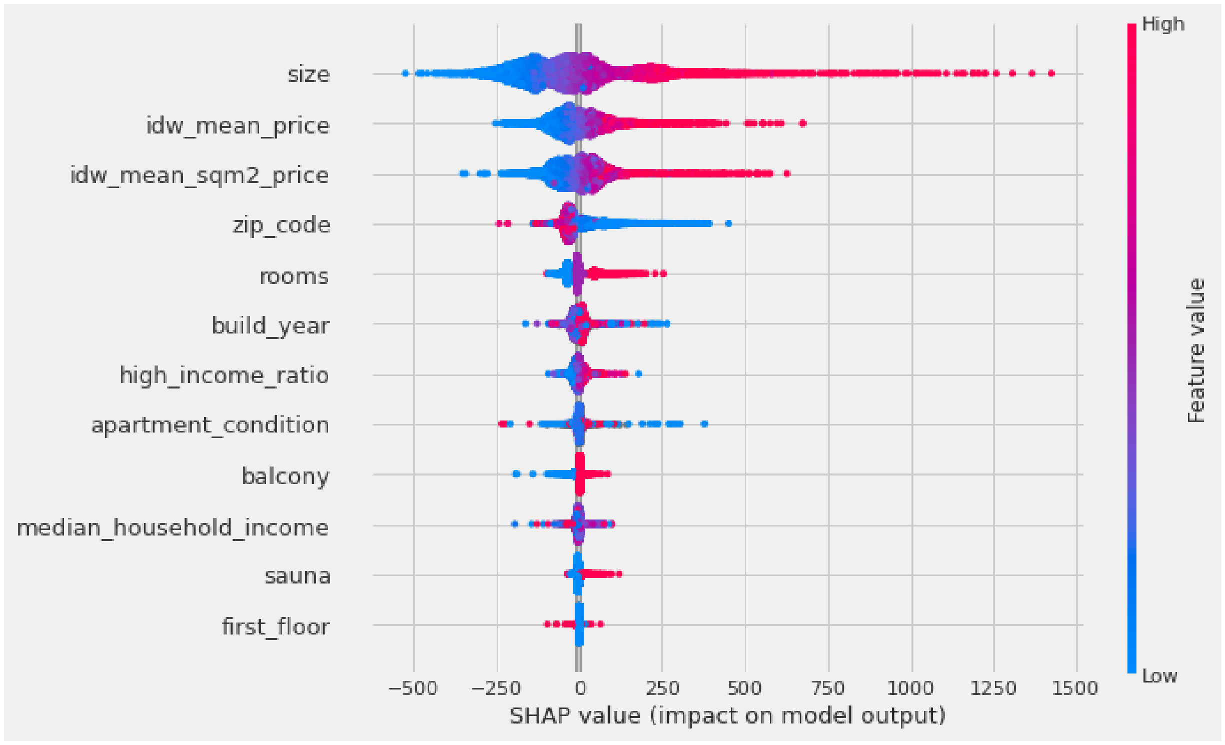


Figure 4.2: XGBoost with monotonic constraints SHAP summary; a more red colour represents a higher feature value, while blue colour stands for a lower feature value.

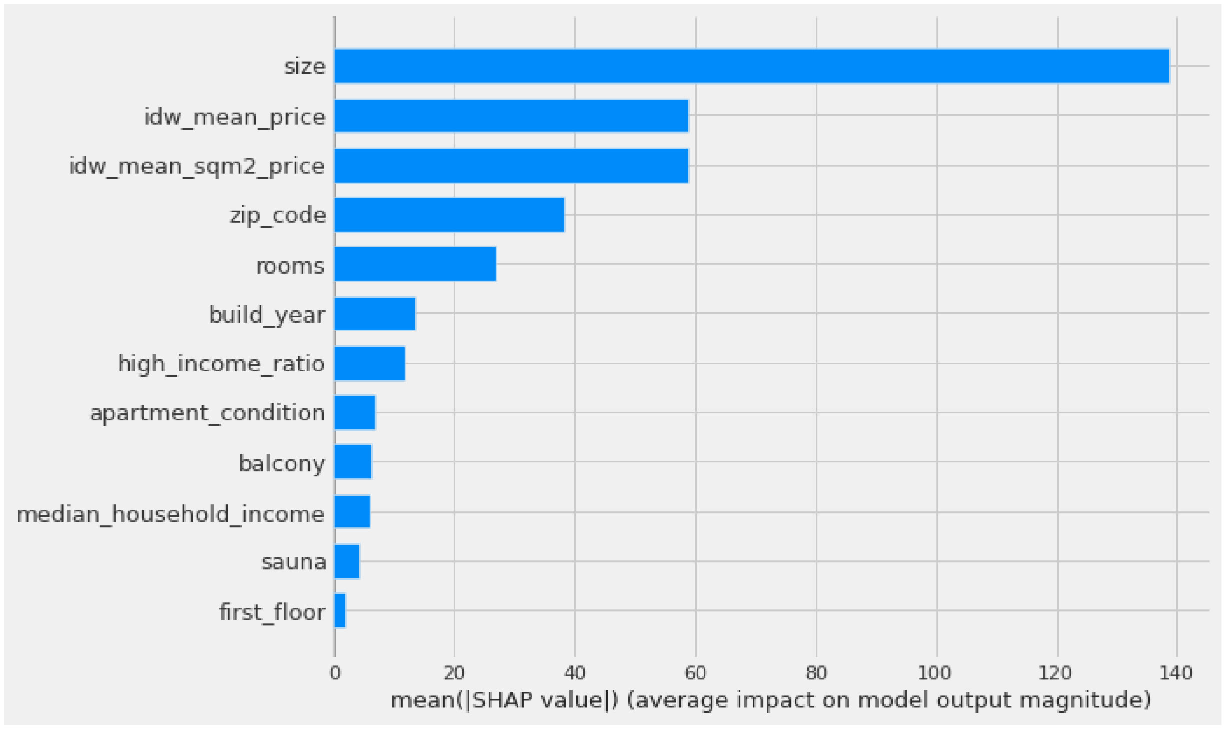


Figure 4.3: The feature importance from SHAP is dissimilar to the feature importance plot exported from the XGBoost model using the *XGBoost* library. Mean feature importance is calculated for SHAP as seen in equation 2.10.

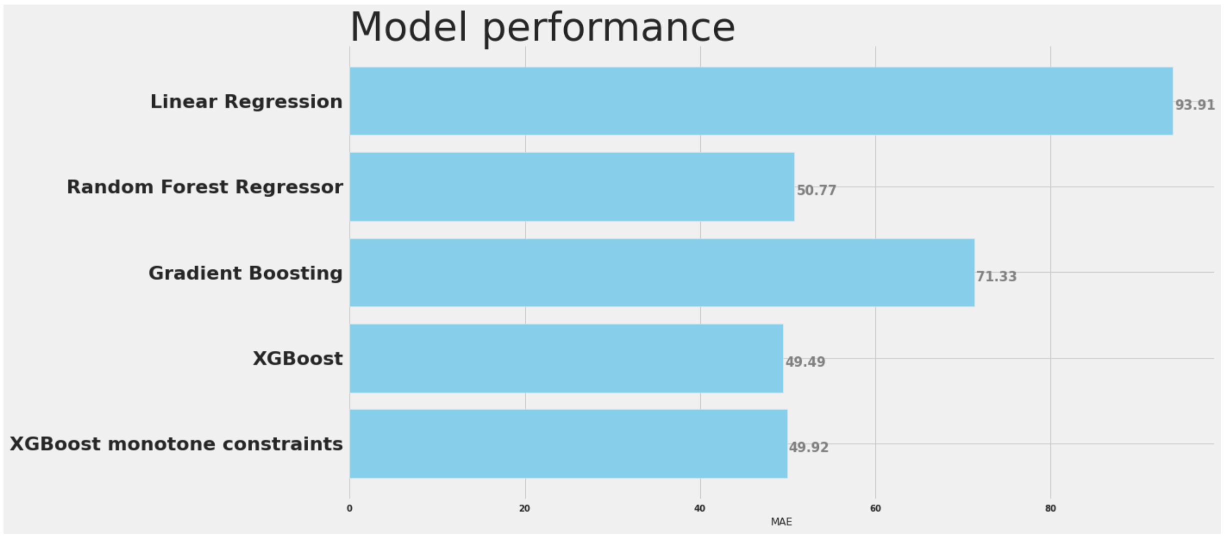
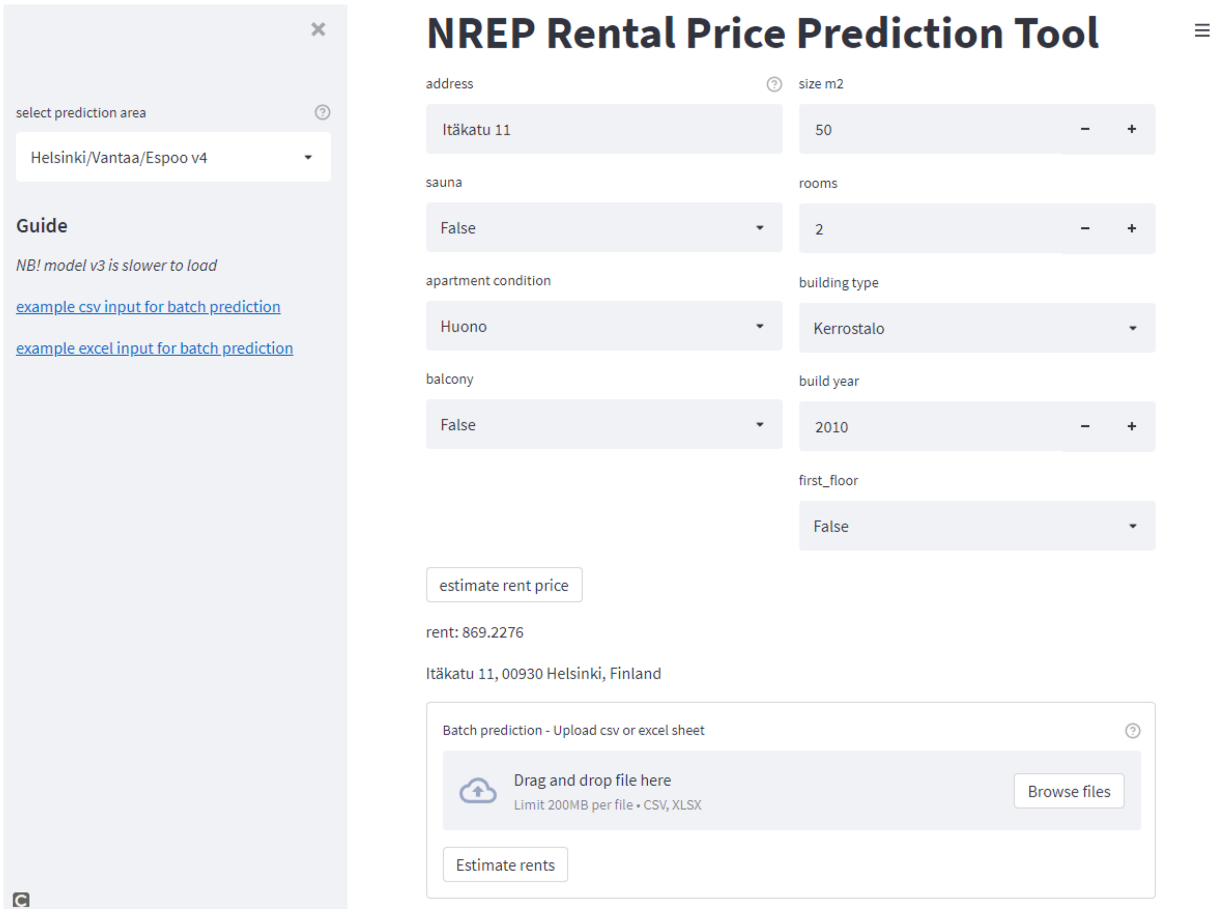


Figure 4.4: A comparison of model performance based on MAE which was used to train the models.



rent: 869.2276

Itäkatu 11, 00930 Helsinki, Finland

Batch prediction - Upload csv or excel sheet
Drag and drop file here
Limit 200MB per file • CSV, XLSX
Browse files
Estimate rents

Figure 4.5: The interface created to test the models made for the project. Application built using *streamlit.io*.

5 Discussion

The decent predictions made by our model compared to other models can be explained by a number of elements. The high F scores of the features mean_price_per_sqm_for_interpolation and mean_price_for_interpolation in Figure 3.2 show the importance of using inverse distance weighting in including the prices of nearby apartments in the estimation of new apartments.

The favourable results from testing the model are in partly explained by the comparatively large amount of data available to train the models as explained by *Yihao et al.* (Y. Chen et al., 2021). CatBoost by *Yandex* replaces high-dimensional categorical features with single numerical values using target statistics (Sagi and Rokach, 2021). It could have been used with our data containing several categorical features instead XGBoost for speeding up the training phase at the cost of prediction accuracy, but with the relatively fast (under one minute) training with the full Greater Helsinki data set for apartments, it was not necessary to consider its use (Sagi and Rokach, 2021).

Advantages of utilizing XGBoost are it's ease of use, due to there being no need to normalize parameters or change categorical values into numerical ones for regression tasks. The data preparation stage will take less time due to XGBoost working despite null values in the data.

Differences in the feature importance derived from SHAP values and the XGBoost feature importance reliant on split counts can likely be attributed to feature importance in XGBoost favouring large numerical features that could be split several times. Thus, testing different interpretability methods can yield worthwhile results, and the SHAP model average impact values are likely to explain the true importance of the features compared to the XGBoost split count method.

Initially, when it came to estimating the price of an unit, an unfortunate development was that the model had learned for some locations that an increase in the apartment condition of an apartment would correlate with a decrease in rental price. Predicted rental prices could be higher for apartments in bad and excellent conditions than those in neutral or good conditions with all other parameters remaining the same. Due to some apartments being in a worse off condition yet staying expensive due to the effects of other criteria, a user testing the model could see the recommended rental price drop by increasing the condition of an apartment. A more general explanation to this phenomenon is that the apartment data could be too sparse in some areas to learn the proper relationship between price and apartment condition that we believe to exist. Some other noise in the data could also cause the unsound pricing. Nevertheless, in this situation the apartment condition has a monotonic relationship with the target value **price**, which should manifest in the predictions made with the model. The use of monotonic constraints on parameters such as the apartment condition corrected the error, giving the increasing parameters a linear relation to the increase of the price. The use of monotonic constraints resulted in a higher RMSE and MAE, but allowed the implementation of the prior belief that *as the apartment condition rises, the price should remain the same or increase as well when all other parameter are unchanged.* Another cost of using monotonic constraints with XGBoost was the additional time spent training the model and the increased amount of memory required to save the decision trees during the training phase (T. Chen and Guestrin, 2016a).

As said, the use of monotonic constraints will decrease the average predictive power of the model, but will remove nonsensical instances in which predicted apartment prices can fall while the quality of the apartment unit increases due to the model having learnt to correlate the wrong sequential order for a feature with increased prices. To utilize the coordinates of the apartment in the study, inverse distance weighting of the rental price added a new feature to the training data that had great importance in tree generation. Besides the *apartment size*, the top features were location dependent, thus the old housing related adage of "location, location, location" still likely holds merit.

5.1 Validity and Reliability

One potential threat to the internal validity of this study is the general unpredictability in explaining the workings of black-box methods such as XGBoost. Small changes in the hyperparameters can give very different models that lead to very different predictions despite the average prediction error being the same for the models. In some tests, apartments at a certain postal code could be hundreds of euros more expensive, while another model was able to give predictions that were closer to the expected value despite suffering in similar manner at some other locations. However, the validity of the tested models was still sufficiently high to claim that the models could be of actual use. The cross-validation train/test split of 80/20 that was utilized resulted in large validation sets with the nigh

200k observation data set used in the project. Another potential threat is that some of the data that had imputed fields could cause the model to learn some faults despite best practises when imputing data. Some of the included features also had a very minor effect on the model performance, such as the presence of a sauna, but they were kept in the training phase. Due to the validation phase after training, we are confident in the predictive capabilities of the models despite any errors that could affect the training phase. The model comparisons on the other hand are less reliable for making conclusions, as the number of observations that was available for us was low.

Regarding the external validity, due to the data being from 2018-2021, the covid-19 pandemic could have affected some of the features in the data, such as the rental price of the apartments due to different rental behaviour of people expressed during the duress of the pandemic. The effect of increased remote work could impact the data from Oikotie, but this is not yet visible in the data. Nevertheless, this could have added a trend to the data that would not have been present in a more typical time period.

Due to the decent performance of the models, tests could be done whether using the models generated on the Greater Helsinki data used as a base for a model for other cities with additive modelling or reinforcement learning. Without the glut of apartment building data that Greater Helsinki has, this could give a leg up to creating working models for other large cities with apartment buildings.

5.2 Further Research

In the future, testing stacked methods that incorporate several different machine learning systems in series would be interesting when attempting to estimate housing prices. Something of interest is exploring which methods can give the best return to those working with smaller data sets and meager budgets when it comes to model training infrastructure.

Generating distance matrices for nearest railway stations, bus stops, shops, restaurants, bars or other remarkable landmarks that have an affect on the rental price of an apartment is something that should be tested, and used as features for training a model. For apartments that have 3-4 rooms, it would be prudent to test whether the ratings of the nearest elementary and lower upper secondary schools have an effect on the price, as students of these levels are automatically admitted to their nearest schools without any entrance exam.

The difference between the estimated prices and set prices for apartment units for the housing brokerage companies offering these apartments can be examined to determine whether some brokers are more likely to undersell or oversell.

Features that were deemed insignificant in their importance could be attempted to be incorporated into the study using principal component analysis (PCA) or an autoencoder to combine several of these features, and test whether the variance between the new records is large enough to differentiate them and use the new values as a training feature. Techniques similar to this were used in the past by *Guntermann et al.* (Guntermann and Norrbin, 1987).

The method by which samples are chosen for the comparison between commercial model and the model being created in the study should be changed in the future. Random sample selection could be done using the random sample function of *scikit-learn* on 2022 apartment listings from the Oikotie API (Buitinck et al., 2013).

As mentioned earlier in its own subsection, being able to rationalize how the model arrives to a certain outcome could be important in order to convince stake-holders of its operation, especially in a domain such as finance for housing or rentals. Thus, it would be of interest to attempt to implement the method described by *Sagi et al.* to simplify the decision forest into a performant decision tree (Sagi and Rokach, 2021).

It would also be good to test how the XGBoost implementation compares to Microsoft's LightGBM Gradient Boost library. LightGBM is apparently faster and more efficient than XGBoost in the training phase, and builds the decision trees leaf-wise instead of depth-wise like XGBoost, reducing overfitting to data. Due to these properties, it should be more suitable for larger data sets (Ke et al., 2017). Also, testing and documenting the performance of ensemble models that involve several different models should be worked on, especially since the *hgboost* python module presented workable ensembles using XGBoost, CatBoost and LightGBM.

The American online real estate market place Zillow has been providing online estimates utilizing hedonic and ML models using their *"zestimate"* system. In 2021, they reported testing a ML solution employing neural networks to reduce the average estimation error*. Developing and assessing the practicality of a neural network solution to tackle estimating rental prices could be another avenue of future research.

*https://www.prnewswire.com/news-releases/zillow-launches-new-neural-zestimate-yielding-major-accuracy-gains-301312541.html

As a model localized to Helsinki, Vantaa and Espoo has good predictive power, a semi-automated system could be tested in which a hierarchy of models is created for cities and/or expensiveness areas. This would mean that when estimating prices, different models are used depending on the availability of data for the subset of postal codes the apartments are located in, or whether the model has a good enough MAE or RMSE score when validated. If there is not a sufficient amount of data in the current model, then a higher level model is used, which has been trained on a larger set of apartment data within postal code areas containing the previous subset of apartments.

6 Conclusions

The purpose of this study was to test and develop an optimal method for predicting rental prices for Greater Helsinki area apartment units. After testing, the use of the XGBoost algorithm was deemed as the main focus of the project due to the superior performance and for it's ability to impute data for each feature. We compared the XGBoost against several other models such as Random Forest Ensemble, variants of linear regression and non-XGBoost gradient boosting. We compared the final model against two commercially available solutions and noted that the model performed as well as one of the solutions. The large amount of available data was crucial in this study in generating models with sensible estimations. We created an interface for the client to use the developed models during the project to receive feedback during development and act as a proof-of-concept for a future system. Machine learning models are certain to have a place in predicting apartment prices and systems relying on XGBoost can become competitive with comparatively less effort than expected.

Bibliography

Akiba, T., Sano, S., Yanase, T., Ohta, T., and Koyama, M. (2019). *Optuna: A Next-generation Hyperparameter Optimization Framework*. arXiv: 1907.10902 [cs.LG].

Breiman, L. (2001). "Random forests". In: *Machine Learning* 45.1, pp. 5–32. URL: www.scopus.com.

Buitinck, L., Louppe, G., Blondel, M., Pedregosa, F., Mueller, A., Grisel, O., Niculae, V., Prettenhofer, P., Gramfort, A., Grobler, J., Layton, R., VanderPlas, J., Joly, A., Holt, B., and Varoquaux, G. (2013). "API design for machine learning software: experiences from the scikit-learn project". In: *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, pp. 108–122.

Cannon, S. and Cole, R. (2011). "How Accurate are Commercial Real Estate Appraisals? Evidence from 25 Years of NCREIF Sales Data". In: *The Journal of Portfolio Management* 35, pp. 68–88. DOI: 10.2139/ssrn.1824807.

Chen, T. and Guestrin, C. (2016a). "XGBoost: A Scalable Tree Boosting System". In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '16. San Francisco, California, USA: ACM, pp. 785–794. ISBN: 978-1-4503-4232-2. DOI: 10.1145/2939672.2939785. URL: http://doi.acm.org/10.1145/2939672.2939785.

– (2016b). "XGBoost: A Scalable Tree Boosting System". In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '16. San Francisco, California, USA: Association for Computing Machinery, pp. 785–794. ISBN: 9781450342322. DOI: 10.1145/2939672.2939785. URL: https://doi-org.libproxy.helsinki.fi/10.1145/2939672.2939785.

Chen, Y., Xue, R., and Zhang, Y. (2021). "House price prediction based on machine learning and deep learning methods". In: *2021 International Conference on Electronic Information Engineering and Computer Science (EIECS)*, pp. 699–702. DOI: 10.1109/EIECS53707.2021.9587907.

ElShawi, R., Sherif, Y., Al-Mallah, M., and Sakr, S. (2021). "Interpretability in healthcare: A comparative study of local machine learning interpretability techniques". In: *Computational Intelligence* 37.4, pp. 1633–1650. DOI: https://doi.org/10.1111/coin.12410. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/coin.12410. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/coin.12410.

Friedman, J. H. (2001). “Greedy function approximation: A gradient boosting machine”. eng. In: *The Annals of statistics* 29.5, pp. 1189–1232. ISSN: 0090-5364.

Guntermann, K. L. and Norrbin, S. C. (1987). “Explaining the Variability of Apartment Rents”. In: *Real Estate Economics* 15, pp. 321–340.

James, G., Witten, D., Hastie, T., and Tibshirani, R. (2021). “Tree-Based Methods”. In: *An Introduction to Statistical Learning: with Applications in R*. New York, NY: Springer US, pp. 327–365. ISBN: 978-1-0716-1418-1. DOI: 10.1007/978-1-0716-1418-1_8. URL: https://doi.org/10.1007/978-1-0716-1418-1_8.

Kandula, S. and Shaman, J. (2019). “Reappraising the utility of Google Flu Trends”. In: *PLOS Computational Biology* 15, e1007258. DOI: 10.1371/journal.pcbi.1007258.

Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., Ye, Q., and Liu, T.-Y. (2017). “Lightgbm: A highly efficient gradient boosting decision tree”. In: *Advances in neural information processing systems* 30, pp. 3146–3154.

Kok, N., Koponen, E.-L., and Martínez-Barbosa, C. A. (2017). “Big Data in Real Estate? From Manual Appraisal to Automated Valuation”. In: *The Journal of Portfolio Management* 43, pp. 202–211. DOI: 10.3905/jpm.2017.43.6.202.

Lu, G. Y. and Wong, D. W. (2008). “An adaptive inverse-distance weighting spatial interpolation technique”. In: *Computers Geosciences* 34.9, pp. 1044–1055. ISSN: 0098-3004. DOI: https://doi.org/10.1016/j.cageo.2007.07.010. URL: https://www.sciencedirect.com/science/article/pii/S0098300408000721.

Mack, C., Z, S., and D., W. (2018). *Managing Missing Data in Patient Registries: Addendum to Registries for Evaluating Patient Outcomes: A User's Guide, Third Edition [Internet]. Rockville (MD): Agency for Healthcare Research and Quality (US).* URL: https://www.ncbi.nlm.nih.gov/books/NBK493614/.

Madhuri, C. R., Anuradha, G., and Pujitha, M. V. (Mar. 2019). “House Price Prediction Using Regression Techniques: A Comparative Study”. In: *2019 International Conference on Smart Structures and Systems (ICSSS)*, pp. 1–5. DOI: 10.1109/ICSSS.2019.8882834.

Mo, H., Sun, H., Liu, J., and Wei, S. (2019). “Developing window behavior models for residential buildings using XGBoost algorithm”. In: *Energy and Buildings* 205, p. 109564. ISSN: 0378-7788. DOI: https://doi.org/10.1016/j.enbuild.2019.109564. URL: https://www.sciencedirect.com/science/article/pii/S0378778819324971.

Molnar, C. (2022). *Interpretable Machine Learning. A Guide for Making Black Box Models Explainable*. 2nd ed. URL: https://christophm.github.io/interpretable-ml-book.

Moulay, E., Léchappé, V., and Plestan, F. (2019). "Properties of the sign gradient descent algorithms". In: *Information Sciences* 492, pp. 29–39. ISSN: 0020-0255. DOI: https://doi.org/10.1016/j.ins.2019.04.012. URL: https://www.sciencedirect.com/science/article/pii/S0020025519303135.

Park, B. and Bae, J. K. (2015). "Using machine learning algorithms for housing price prediction: The case of Fairfax County, Virginia housing data". In: *Expert Systems with Applications* 42.6, pp. 2928–2934. ISSN: 0957-4174. DOI: https://doi.org/10.1016/j.eswa.2014.11.040. URL: https://www.sciencedirect.com/science/article/pii/S0957417414007325.

Sagi, O. and Rokach, L. (2021). "Approximating XGBoost with an interpretable decision tree". In: *Information Sciences* 572, pp. 522–542. ISSN: 0020-0255. DOI: https://doi.org/10.1016/j.ins.2021.05.055. URL: https://www.sciencedirect.com/science/article/pii/S0020025521005272.

Schulz, E., Speekenbrink, M., and Krause, A. (2018). "A tutorial on Gaussian process regression: Modelling, exploring, and exploiting functions". In: *Journal of Mathematical Psychology* 85, pp. 1–16. ISSN: 0022-2496. DOI: https://doi.org/10.1016/j.jmp.2018.03.001. URL: https://www.sciencedirect.com/science/article/pii/S0022249617302158.

Sirmans, G. S., Macpherson, D. A., and Zietz, E. N. (2005). "The composition of hedonic pricing models". English. In: *Journal of Real Estate Literature* 13.1, pp. 3–43. URL: www.scopus.com.

Truong, Q., Nguyen, M., Dang, H., and Mei, B. (2020). "Housing Price Prediction via Improved Machine Learning Techniques". In: *Procedia Computer Science* 174, pp. 433–442. DOI: 10.1016/j.procs.2020.06.111.

Varma, A., Sarma, A., Doshi, S., and Nair, R. (2018). "House Price Prediction Using Machine Learning and Neural Networks". In: *2018 Second International Conference on Inventive Communication and Computational Technologies (ICICCT)*, pp. 1936–1939. DOI: 10.1109/ICICCT.2018.8473231.

Willmott, C. J. and Matsuura, K. (2005). "Advantages of the mean absolute error (MAE) over the root mean square error (RMSE) in assessing average model performance". In: *Climate Research* 30.1, pp. 79–82. ISSN: 0936577X, 16161572. URL: http://www.jstor.org/stable/24869236.

Wing, C. K. and Chin, T. (2003). "A Critical Review of Literature on the Hedonic Price Model". In: *International Journal for Housing Science and Its Applications* 27, pp. 145–165.

Zhou, D. C., Jin, Z., and Zhang, T. (2019). “A Fast Sampling Gradient Tree Boosting Framework”. In: *CoRR* abs/1911.08820. arXiv: 1911.08820. URL: http://arxiv.org/abs/1911.08820.

Acknowledgements

I would like to thank my supervisor Prof. Jukka Nurminen for the aid and advice that he has freely shared during the thesis process, NREP and Oivan for funding this research and Perttu R. for all the helpful discussions and professional insights that lead to the success of the thesis. I would also like to give my thanks to Elina K. and my family for all the support.