Stockholms universitet

Living with Trees

Predicting Swedish Apartment Prices with eXtreme Gradient Boosting

Andreas Hörnqvist

Kandidatuppsats i matematisk statistik
Bachelor Thesis in Mathematical Statistics

Kandidatuppsats 2019:3
Matematisk statistik
Januari 2019

www.math.su.se

Matematisk statistik
Matematiska institutionen
Stockholms universitet
106 91 Stockholm

Mathematical Statistics
Stockholm University
Bachelor Thesis **2019:3**
http://www.math.su.se

Living with Trees
Predicting Swedish Apartment Prices with eXtreme Gradient Boosting

Andreas Hörnqvist*

January 2019

Abstract

The recent advent of statistical learning methods promises accurate predictions and insights. The ideas of what would later be the underpinnings of decision trees was first introduced in the nineteen-fifties. Today the use of serial ensembles of decision trees deliver unparalleled performance on a wide range of learning problems. One method that leverages such a procedure is eXtreme Gradient Boosting. The fair market value of real estate property is usually determined by a licensed appraiser, such as a real estate agent. While knowledgable professionals appraisers are due to make subjective estimates resulting in uncertain assessments. The purpose of this thesis is to produce a model that rivals the predictive performance of appraisers and provides both buyers and sellers objective price estimates. Applying eXtreme Gradient Boosting to a dataset containing records of sold apartments in Sweden during 2013 to 2018 results in such a model. Leveraging bespoke predictors and tuning meta-parameters a predictive model that in fact outperforms appraisers estimates, in terms of list prices, is achieved. The most important features for assessing the market value is determined to be local price point, living area and rent.

*Postal address: Mathematical Statistics, Stockholm University, SE-106 91, Sweden. E-mail: andreas.hornqvist@gmail.com. Supervisor: Chun-Biu Li and Disa Hansson.

Acknowledgment

This is a Bachelor's thesis of 15 ECTS in Mathematical Statistics at the Department of Mathematics at Stockholm University. I would like to thank my supervisors Chun-Biu Li and Disa Hansson for their support and advice. I am also grateful to Tomas Kasemets and Julia Kahlström for their suggestions and encouragement.

Contents

1 Introduction

"The fair market value is the price at which a property would change hands between a willing buyer and a willing seller, neither being under any compulsion to buy or to sell and both having reasonable knowledge of relevant facts."

- United states vs. Cartwright, 411 U.S. 546, 551 (1973)

Determining the fair market value of tenant-owned Swedish apartments (hereinafter referred to as apartments) is the responsibility of licensed appraisers, often real estate agents. The task of appraisal is integral in preparation for putting an apartment on the market as it is the basis of the advertised list price, Mäklarsamfundet (2017) [26]. Deliberately stating a list price substantially below the fair market value is considered malpractice and can result in the disqualification of the appraisers license.

While the actual market value of an apartment is determined in the final purchase agreement between buyer and seller, the estimation of a fair market value is of importance to to both parties as it is the baseline of price negotiations.

There are different methods for establishing the fair market value of apartments, among them the widely used *ortsprismetoden*. The outcome of ortsprismetoden is however dependent on the appraisers subjective assessment and consequently varies between practitioners.

A method for objective and accurate estimation of a fair market value would benefit sellers by reducing the uncertainty caused by the appraisers subjective assessment. Potential buyers would also be served by such a model as price expectations influenced by list price would closer match the final price.

The emergence of the field of *statistical learning* in combination with easy access to large datasets provides a great opportunity to create powerful predictive models. Devising such a model in an effort to predict apartment prices, and thus fair market value, is the aim of this thesis.

1.1 Background

Booli is a company that provides its users information on housing in Sweden. Among its services are an API which allows access to historical and current data on the Swedish housing market. The dataset encompasses most of the sold and listed housing objects from 2013 and forward, containing variables like price, day of sale, location, etc.

Valuation using ortsprismetoden takes into account the delimited relevant market, e.g. geographical area, similarities of objects, and temporal distance, amongst other criteria, Persson (2005) [27]. With ortsprismetoden in mind we make the observation that the Booli dataset contains a number

of the features required for making estimates of market values. This allows us to pursue a predictive model with some confidence.

To formulate a model using statistical learning methods it is helpful to identify some characteristics of the model and data. The model should output a continuous value, the predicted price. This is the dependent variable. The dataset contains the outcome of the dependent variable, that is to say that the data is labelled. Furthermore the method, and hence the model, preferably leverages aspects of the data that are known to be important for accurate predictions, e.g. geospatiality and temporality. We also know that housing markets can demonstrate non-linear behavior, Muellbauer & Murphy (2012) [25].

Given the above and using statistical learning nomenclature it can be concluded the problem is that of *supervised regression*, and should in the current case preferably be solved using a method that accounts for possible non-linearity.

The field of statistical learning provides many candidate methods for such a problem and in a practical setting one would often try, and combine, several of them. However, given the constraints of an academic thesis I have opted to choose one primary method, namely XGBoost. XGBoost have been proven to outperform other methods on a wide variety of datasets, Chen & Guestrin (2016) [5] and is thus a promising method for producing the predictive model.

1.2 Aim

The aim of this thesis is to create a predictive model that rivals the proficiency of professional appraisers. This means creating a model that does short term prediction on Swedish housing prices. Where short term reflects estimation under current market conditions. The model should be accurate enough to have practical use and the method used should be eXtreme gradient boosting, applied to the Booli dataset. In summary, the aspiration of this thesis is to answer if:

it is possible to achieve a satisfactory predictive model
for Swedish housing using Booli data and XGBoost?

1.3 Related Work

The intent of this section is to provide an overview of the research and evolution of methodology culminating in XGBoost. The overview is meant to be informal and many of the concepts and methods introduced may be unfamiliar but will be further discussed in Chapter 2.

In order to comprehend the method of gradient boosting in general and XGBoost in particular some background information will be necessary. We will see that XGBoost, a special case of gradient boosting, builds on a variety

of techniques, the most fundamental among them is decision trees, making them a good place to start.

In decision tree learning, predictions are inferred from observed values of a set of p predictor variables $X^T = X_1, \ldots, X_p$ and response y via a predictive model $\hat{Y} = f(X) + \epsilon$, where $\hat{Y}$ are the predictions and ϵ is random noise. The matching observations of y and X are usually referred to as a training or instance data. The model is constructed by recursive binary splitting where the dependent variable is successively split into two partitions given some criteria for a predictor variable. The aim is for each partition to be be as internally homogenous as possible while being different with regard to the dependent variable. This procedure generates a tree like structure where an observation can be sent down the tree from the initial root-node (decision trees are grown upside-down) to a final leaf-node containing a predictive value. Figure 2 on page 17 depicts the outcome of such a procedure, the left panel illustrates the partitions and the right panel the corresponding tree-structure. Tree-based models will be discussed more in-depth in Section 2.4.

Tree growing or hierarchical splitting has its origin in analysis of survey data according to Ritschard (2013) [30]. Belson (1959) [1] is often attributed with being the first to suggest generating tree-like structures by segmenting data. Belson proposed to develop relevant matching (i.e. representability) criteria by way of predictive composites, that is by establishing the predictive power of a range of possible predictor variables.

A few years after Belson's paper the first regression tree algorithm was published by Morgan & Sonquist (1963) [24]. The proposed algorithm was dubbed Automatic Interaction Detection (AID). AID predicts a regression-type (i.e. continuous) variable using a set of categorical predictors, which generally only take on a few possible values. AID follows the binary splitting procedure described above until a certain threshold of improvement, also referred to as a stopping criteria, is reached. The predicted value, or leaf weight, returned in each leaf node is then the node sample mean. The optimization procedure for finding leaf nodes and leaf weights are of particular interest and will be explored throughout Chapter 2.

Messenger & Mandel (1972) [23] broadened the AID approach to handle categorical response variables using the method Theta Automatic Interaction Detection (THAID). Like its predecessor THAID chooses splits based on some loss function. In the context of statistical learning a loss function is typically a measure of difference between estimated and actual values of instance data. Several loss functions are suggested in the paper, among them entropy $\phi(t) = -\Sigma_j p(j|t) log\ p(j|t)$ and the Gini index $\phi(t) = 1 - \Sigma_j p^2(j|t)$, where $p(j|t)$ is the proportion of class j observations at node t. The choice of loss function is integral for achieving satisfactory predictive performance and will be formally introduced in Section 2.2.

Tree based models did not immediately gain traction in the statistics

community however. Concerns about over-fitting were raised by Einhorn (1972) [10] as were problems with variable masking, Doyle (1973) [9]. Over-fitting occurs when a model corresponds too closely to a particular set of data and thus may fail at predicting independent data. Over-fitting will be discussed thoroughly in Section 2.2.1. Variable masking is a phenomenon that occurs when inferring importance of particular predictors. When masked, predictors known to be important might not receive an appropriate importance rank. Masking and variable importance will be discussed in Section 2.3.

Despite some initial doubts more efficient techniques for carrying out splits were proposed and with Breiman's (1984) [3] paper interest was reinvigorated. Brieman's proposed algorithm manages both classification and regression problems, as evident by the name Classification And Regression Trees (CART). CART uses the same splitting criteria as previous methods but Brieman suggest the use of a cross-validated pruning scheme instead of a stopping criteria. Pruning reduces the size of a tree by removing leaf-nodes that do not improve predictive performance, thus reducing model complexity and alleviating over-fitting. Cross-validation in turn is a method used for estimating model performance. Cross-validation will be discussed in Section 2.2.2. Pruning solves many of the over-fitting problems of AID and THAID. Among other improvements Brieman also devised a way to handle missing data values in a node by performing surrogate splits. That is splits on alternate variables other than that of the preferred split due to missing values. The appropriate action when dealing with missing values will be discussed in Section 2.7.2. CART are still prevalent today and are for example often used in conjunction with XGBoost.

Can a set of weak learners create a single strong learner? This was the question posed by Kearns & Valiant (1989) [16]. A learner in this context is just a predictive model. When the response variable is categorical such a learner is often referred to as a classifier, otherwise the learner is commonly called a regressor. In the paper a weak learner is defined as a classifier which only moderately correlates with the true classification, i.e. it is only marginally better than random guessing. A single weak learner is usually some variation of a learning model, e.g. CART. The question prompted a swift reply by Schapire (1990) [31] where he made the case for turning many weak learners into a strong learner and postulated the first boosting algorithm. Schapire & Freund (1997) [13] later developed a way for the method to adapt to the weak learners and thereby improving performance. Due to its adaptive nature the algorithm was dubbed Adaboost, short for adaptive boosting. They were later awarded the Gödel prize for their work on Adaboost. Boosting in general is the practice of iteratively training weak learners, usually on adjusted versions of the the instance data, and then adding them together to create a strong learner with enhanced predictive capabilities. Boosting will be discussed in Section 2.5.

Inspired by the observation that boosting algorithms could be interpreted as to perform optimization on a given loss function Brieman (1997) [4] developed the idea of gradient boosting. Following Briemans notion Friedman (2001) [12] developed the first regression gradient boosting algorithm. During the same time Mason et. al (1999) [22] suggested the view of boosting as iterative functional gradient descent and subsequently showed that Adaboost among others were special cases of gradient boosting algorithms. In Section 2.8 we will see that it is possible to mimic optimization by gradient descent with boosting using CART as weak learners. The realization that boosting can approximate gradient descent is the rationale for the naming convention of gradient boosting.

The advancement in gradient boosting inspired a variety of variations, among them XGBoost. XGBoost was created by Chen & Guestrin (2016) [5] and has gained tremendous popularity since its introcuction. Some of the algorithm's success is due to ease of implementation, complexity control and sparsity awareness, but the main appeal is state-of-the-art performance. XGBoost is the main method used in this thesis and will be explored thoroughly in sections 2.9 and 2.10.

1.4 Delimitations

The main delimitation of this thesis is the exclusive use of the XGBoost algorithm. There are many legitimate approaches but exploring more than one in-depth would cause the scope of this thesis to grow much too large.

As we shall see later XGBoost operates using a so called loss function. The loss function used in XGBoost can be any twice differentiable strictly convex function. In this thesis however only the squared error loss will be explored. The reason for the restriction to one loss function is the same as above.

1.5 Ethics and Sustainability

There are many possible outsets for discussion on ethics regarding statistical learning, ranging from dystopian killer machines to the societal implications of statistical bias in algorithms. In fact the ethical implications of statistical learning spans several fields and a multitude of topics. In this thesis however, a brief discussion on algorithmic bias will have to suffice. Algorithmic bias is a topic of special importance when constructing predictive models.

As learning algorithms become more ubiquitous the cause for scrutinizing algorithmic neutrality increases, Seaver (2013) [32]. Danks & London (2017) [8] describe algorithmic bias as when an algorithm is not merely a neutral transformer of data or extractor of information. The notion of bias in their discussion only pertains to deviating from some standard, moral, legal, statistical or other. In other words, the moral standard is one among

others and bias in this context does not necessarily imply something bad or immoral. Furthermore they provide a taxonomy for different kinds of algorithmic bias, dividing it into: training data bias, algorithmic focus bias, algorithmic processing bias, respectively, and lastly context bias. Training data bias is introduced when the dataset provided to the algorithm deviates from the standard in some way. This can lead to otherwise neutral algorithms yielding biased models.

Algorithmic focus bias arises when competing standards are pitted against each other, take for example a variable not legally permitted to be used. This data might be excluded for privacy reason but had it been included it might have influenced the final model, and thus the model would be biased to either a legal standard or statistical standard.

Algorithmic processing bias appear when the algorithm itself is biased in some way, for example by the use of statistically biased estimator. We shall see that algorithmic processing bias is prevalent in statistical learning and several sections of this thesis is concerned with its implications. Once again it is worth noting that not all bias is related to morality, although there certainly can be moral implications of processing bias.

Finally Algorithmic context bias stems from inappropriate use of algorithms. Most autonomous systems are deployed for a special purpose, in a special context. Employing such a model outside of its intended context can result in bias, moral, legal or otherwise. A trivial example would be an autonomous vehicle trained for driving in right-hand traffic being deployed in Great Britain, where left-hand traffic is the custom.

Not all of these examples of algorithmic bias are by themselves troublesome but it is crucial to be aware of the underlying causes to be able to identify the cases which might be.

1.6 Outline

The outline of the thesis is as follows: In section 2 a more rigorous treatment of the statistical and mathematical methods used in the thesis will be presented. Section 3 serves as an exposition on the Booli data set and how it is used. Section 4 will be dedicated to modeling with XGBoost where parameter tuning and feature selection will be discussed. In section 5 the resulting model and its metrics will be presented. Section 6 will contain discussion on method, model and provide broader perspective. Lastly Section 7 will conclude the thesis.

2 Theory

The aim of this chapter is to provide exposition on central concepts and ideas of Statistical Learning. Special attention will be given variations of

tree based boosting.The intention is that a student in mathematical statistics should find the thesis self contained with regards to the mathematical theory used. The theoretical presentation will follow that of The Elements of Statistical Learning, Hastie et al. (2017) [14] unless otherwise stated.

2.1 Statistical Learning

At its core statistical learning can be though of as learning from data. From the perspective of applied mathematics and statistics however learning is considered function approximation.

Given a *training set* of observations $\mathcal{T} = \{x_i, y_i\}$, $i = 1, \ldots, N$, the pairs $\{x_i, y_i\}$ of the training set are viewed as a point in some $(P+1)$-dimensional space, P being the dimensionality of the set of predictor variables, or *features*, X, where $X^T = (X_1, ..., X_P)$, and $x_i^T = (x_{i1}, ..., x_{iP})$. The other component of the training set y_i is the response measurement matching the predictors x_i. The domain of $f(x_i)$, the function to be approximated, is then equal to that of X and is related to the data via a model $y_i = f(x_i) + \epsilon_i$, where ϵ_i is the random error with expectation $E[\epsilon_i] = 0$. Achieving learning in this setting is consistent with obtaining a useful approximation $\hat{f}(x_i)$.

Furthermore statistical learning problems can be roughly categorized into two classes; supervised and unsupervised. In *supervised learning* the aim is to predict the values $\hat{Y} = \hat{f}(X) + \epsilon$ given the response variable Y and set of predictor variables X. In the case of *unsupervised learning* the goal is infer the properties of the joint density $Pr(X = x)$, given the N observations $(x_1, x_2..., x_N)$ of a random P-vector X without help of a supervisor Y. The key difference from supervised learning being the lack of examples y_i to learn from.

Supervised learning problems are commonly divided into *regression problems* and *classification problems*. The term regression in this context refers to the output, or response, Y taking values in $\mathbb{R}$, whereas classification implies that the output measure is qualitative and taking values in the finite set $\mathcal{G} = (\mathcal{G}_1, ..., \mathcal{G}_m)$, m being the number of possible classes in the response. It is worth noting that the input measures $X^T = (X_1, ..., X_P)$ can be a mix of both quantitative and qualitative measures in both cases.

The problem presented in this thesis is a supervised regression problem which is why the theoretic discussion will focus on issues related to such problems.

2.2 Model Selection and Assesment

The notion of performance of a learning model is related to how well it accomplishes prediction on independent test data. Evaluation of this performance is critical for model selection and assessment both guiding the ultimate choice of model.

We make the following distinction between model selection and assessment:

(a) **Model selection**
Evaluation of prediction performance to determine which of several competing models to choose.

(b) **Model assessment**
Evaluation of prediction performance on new data given a chosen model.

A common initial approach for both selection and assessment is to divide a dataset into three parts: training, validation and test set. The purpose of the training set is naturally to train the model. The validation set is meant to allow for comparison between competing models. Finally the test set is used to estimate the prediction error of the ultimate model using out-of-sample data.

Despite there being difficulty in providing general rules for a split-ratio, as it depends on signal to noise ratio and data abundance, there are hueristics for partitioning the data. Often a 50%-70% partition is recommended for the training set and 15%-25% respectively for validation and test set. In Section 2.2.2 we shall see however that there are ways to performs model assessment more efficiently than splitting the data three ways.

In order to either select or assess a model we need to first define the metric on which we base our conclusions. Given the response variable Y, predictors X and a *loss function* $L(Y, \hat{f}(X))$ that measures the error between the Y and a predictive model $\hat{f}(X)$

$$Err_{\mathcal{T}} = E_{X,Y}[L(Y, \hat{f}(X))|\mathcal{T}] \tag{1}$$

is referred to as the *test error*. The test error (1) is the prediction error over some independent test data with both X and Y drawn at random from their joint distribution and fixed training set $\mathcal{T}$. In practice it is difficult to estimate the conditional error effectively given some $\mathcal{T}$. Instead the *expected test error*

$$Err = E[Err_{\mathcal{T}}] = E[L(Y, \hat{f}(X))] \tag{2}$$

is routinely used for evaluation of performance. As we shall see in the following section the test error has different characteristics than the *training error*

$$\overline{err} = \frac{1}{N} \sum_{i=1}^{N} L(y_i, \hat{f}(x_i)), \tag{3}$$

the average loss over some training set $\mathcal{T}$.

2.2.1 The Bias-Variance Trade-off

Given some function-approximation $\hat{f}(x_0)$ evaluated at point x_0 and the squared error loss the following decomposition of the expected test *mean squared error* (MSE), $E[L(y_0, \hat{f}(x_0))] = E[(y_0 - \hat{f}(x_0))^2 | X = x_0]$ hold,

$$E[(y_0 - \hat{f}(x_0))^2 | X = x_0] = Var(\hat{f}(x_0)) + [Bias(\hat{f}(x_0))]^2 + Var(\epsilon). \quad (4)$$

Here $E[(y_0 - \hat{f}(x_0))^2 | X = x_0]$ refers to the average test MSE obtained if repeatedly estimating f using a large number of training sets, each $\hat{f}$ evaluated at x_0 from a test set.

The decomposition (4) allows some insights: The expected test MSE can't be lower than the variance of the random error ϵ and, which will show very important, that in order to achieve a low expected test MSE the model should have low variance as well as low bias. In this context bias and variance depends on the chosen models flexibility. We can think of model flexibility as "curvy-ness". Take for example a linear model in comparison to a high degree polynomial, the linear model will have a difficult time capturing non-linearity in the data while the polynomial might fit perfectly to the training data. The linear model would be biased since the model is too simple to capture relationships in the data. The polynomial meanwhile would have high variance in the sense that the model would change considerably if data in the training set $\mathcal{T}$ were altered. As a rule of thumb variance increases and bias decreases when a more flexible model is used. Balancing these properties is a key challenge in Statistical learning.

As discussed in the previous section we would like to calculate the expected test error (2). An impulse might be to try and estimate it from the training data $\mathcal{T}$. This would however be a foolish endeavor. As the model $\hat{f}$ becomes increasingly complex the fit to the training data would improve as bias decreased. However, the variance and test error would increase as well. This relationship is illustrated in Figure 1. In other words, a model with zero training error (3) would perform poorly on new data as the model would be overfit to the training data. In the next section a method for estimating the test error will be discussed.

2.2.2 Cross-Validation

Cross-validation is a widely used method for directly estimating the expected test error $E[L(Y, \hat{f}(X)]$. The idea of K-fold cross-validation is to split the available data into K folds of roughly equal size in accordance with the indexing function $k : \{1, 2, ..., N\} \mapsto \{1, 2, ..., K\}$ that indicates which observation $\{x_i, y_i\}$, $i = 1, 2, ..., N$ is allocated in which fold k. The assignment should be random. With the allocation done we fit a model using all but one of the k folds, denoted $\hat{f}^{-k}$. The expected test error is then estimated using the held out fold such that

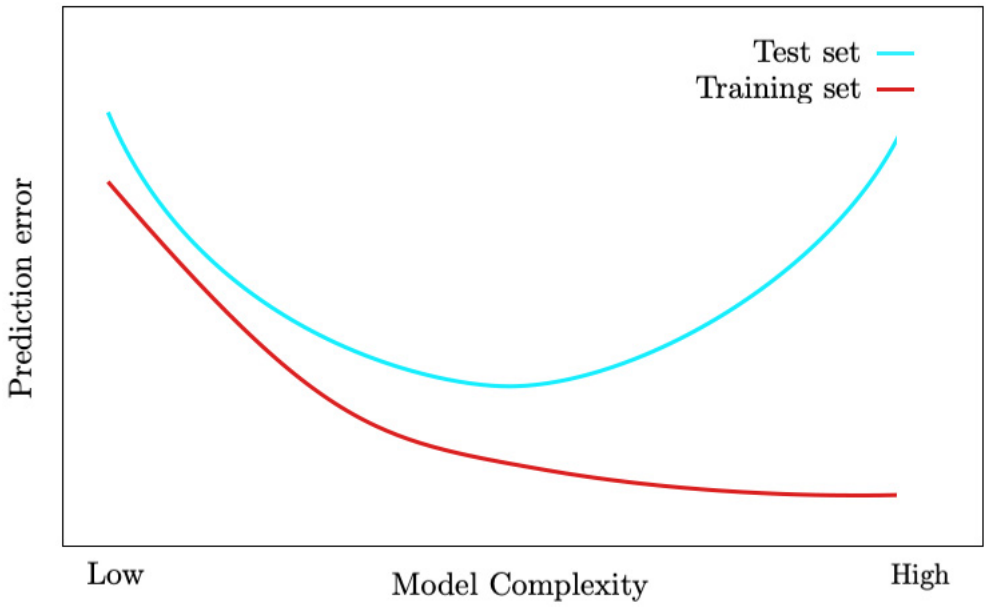

Figure 1: Prediction error of test set and training set as the model complexity is varied. High bias and low variance to the left. Low bias and high variance to the right.

$$Err_k = \frac{1}{N}\sum_{i=1}^{N} L(y_{k_i}, \hat{f}^{-k}(x_{k_i})), \tag{5}$$

where $\{x_{k_i}, y_{k_i}\}$ are the observations belonging to the kth fold. After repeating the process for all k folds we can calculate the cross-validation estimate of the prediction error

$$CV(\hat{f}) = \frac{1}{K}\sum_{k=1}^{K} Err_k. \tag{6}$$

In summary the cross-validation score (6) is an estimate of the test error and is commonly used for model selection. Before using cross-validation in practice however there are some further things to consider.

The choice of k will affect the performance of the cross-validation score, both in terms of computational efficiency and as a metric for model selection. As for the computational demand it is obvious that choosing $k = n$ will be more computational intensive then choosing $k \ll n$ but there are other advantages as well. The most important one being related to the bias-variance trade-off. Using n-fold cross validation would lead to an essentially unbiased estimate of the test error, it would however also lead to higher variance than training on $k \ll n$. The reason being that the cross-validation score would

be based on n almost identical training sets. It follows that the output from the n fitted models would be highly correlated. Using $k \ll n$ means averaging the output of models that are not as highly correlated. Recall the fact that averaging highly correlated quantities results in estimates with larger variance than the mean of quantities which are not as highly correlated. It follows that cross-validation using $k \ll n$ folds usually results in estimates with lower variance then using $n = k$. In summary there is a bias-variance trade-off to consider when choosing the number of folds. In practice typical choices of k is 5 or 10, which have been showed to neither suffer excessive bias nor variance. Furthermore the cross-validation score is best suited for selecting models not assessing them, a task better suited for the use of a explicit test set.

2.2.3 Model Diagnostics

A widely used loss function when solving regression problems is the squared error loss function

$$L(y, f(x)) = (y - f(x))^2. \tag{7}$$

The properties of the squared error loss will be discussed in depth later in this chapter but for now we will explore how it relates to the cross-validation score. Using the squared error loss in (5) yields the MSE for a given held out fold k

$$MSE_k = \frac{1}{N} \sum_{i=1}^{N} (y_{k_i} - f^{-k}(x_{k_i}))^2.$$

In general the MSE measures the average of the square of the residuals and is the estimation of the expected test error (2). The MSE's unit of measurement is the square of the quantity being estimated. While the MSE suffices mathematically as measure of predictive performance the *root mean squared error* (RMSE)

$$RMSE = \sqrt{MSE}$$

is easier to interpret as its unit of measurement is the same as the quantity being measured. RMSE will be used as the measure of predictive accuracy throughout this thesis.

2.3 Feature Attribution

Interpretation of predictive models is important for several reasons. One aspect is that of algorithmic bias as discussed in Section 1.5, in order to explain a model and motivate its decisions it helps to understand its inner workings. Another reason for model interpretation is that it can provide

insights on how to improve the model and aid understanding of the process that is being modeled.

2.3.1 Additive feature attribution methods

In the ideal case the best explanation of a model f is the model itself. Unfortunately complex models such as ensembles, while improving in predictive performance suffer in interpretability. One course of action in such cases is to use a simpler explanation model g to explain f. Through the use of *Additive feature attribution methods* a predictive model f's output can be explained as sum of real values attributed to each input feature p. Note that p just represents the feature, the instance data associated with p is referred to as x_p. Such an explanation model g is then defined as

$$g(z') = \phi_0 + \sum_{p=1}^{P} \phi_p z'_p \tag{8}$$

where $z' \in \{0,1\}_{p=1}^{P}$, and P is the number of input features, and $\phi_p \in \mathbb{R}$. The z'_i variables represents a feature being observed, $z'_i = 1$ or missing $z'_i = 0$. The sum of the effect of each feature ϕ_p then approximates the output $f(x_p)$ of the original model. The mathematical motivation of additive feature attribution methods are outside the scope of this thesis but the interested reader will find a thorough discussion in Lundberg & Lee (2018) [21].

2.3.2 SHAP

It can be shown that popular techniques for determining the importance of predictors are inconsistent and/or inaccurate. The theoretical justification for this claim will be omitted in this thesis, see Lundberg et al (2018) [20] for details. In the absence of a formal proof empirical inconsistencies will be demonstrated In Section 2.3.

To overcome these inconsistencies *Shapley additive explanation* (SHAP) values are proposed as consistent measures of attribution. SHAP values are derived from game theory's Shapley values. To compute Shapley values let $f_S(x_S)$ be a model trained on the set of features S with corresponding instance data x_S. Furthermore let $f_{S\cup\{p\}}(x_{S\cup\{p\}})$ be the model trained on the features in S and the additional feature p, and instance data corresponding to those features $x_{S\cup\{p\}}$. ϕ_p, $p = 1, \ldots, P$ is then the effect of each feature according to

$$\phi_p = \sum_{S\subseteq\mathcal{P}\setminus\{p\}} \frac{|S|!(P-|S|-1)!}{P!} \left[f_{S\cup\{p\}}(x_{S\cup\{p\}}) - f_S(x_S)\right] \tag{9}$$

where $\mathcal{P}$ is the set of all input features and S the set of features with non-zero indexes in z'. To evaluate the effect of missing features we need to

define a mapping h_x between z' and the original feature input space. With such a mapping the effect of observing or not observing a feature can be evaluated by $f_S(x_S) = f(h_x(z')) = E[f(x)|x_S]$ where $E[f(x)|x_S]$ is the expected value of the model conditioned on a subset S of the input features. Combining this definition of $f_S(x_S)$ with the Shapley values (9) we get SHAP values. Using SHAP values in conjunction with (8) we can measure the influence or attribution of a certain predictor on the model f. The intuition behind Shapely values is that one wants to calculate the weighted marginal contribution of a feature for all possible permutations of S. The marginal contribution of a feature p is represented by $f_{S\cup\{p\}}(x_{S\cup\{p\}}) - f_S(x_S)$. $|S|!$ is the number of possible permutations prior to p's addition, while $(P-|S|-1)!$ is the number of ways features added after p can be arranged. Finally summing over all possible sets S not containing p, $\sum_{S\subseteq\mathcal{P}\setminus\{p\}}$ and dividing by all possible permutations of features $P!$ gets us the Shapley values.

2.4 Tree-based Methods

In Section 2.5 we will see that gradient boosting uses an ensemble of statistical learning models not limited to any given model. In most practical applications however tree-based methods are more prolific as they are able to capture non-linear links and interactions among predictors. For this reasons such methods will be discussed in some detail below.

The basic idea of tree-based methods is to divide the domain of X into j disjunct rectangles R_j, $j = 1, \ldots, M$ where every split is parallel with an axis. Then, using a simple model, calculate some constant c_j to minimize a given criterion for each partition. Tree-based methods are useful for both classification and regression problems as reflected by the name of a popular algorithm, CART, short for *Classification And Regression Tree*. In a regression setting, restricted to recursive binary splitting, the model produced by CART can be represented by

$$\hat{f}(X) = \sum_{j=1}^{J} \gamma_j \mathbb{1}\{(X_1, ..., X_p) \in R_j\} \tag{10}$$

the details of which will be discussed in the following paragraphs.

The process of recursive binary splitting refers to splitting a given space into two half-planes followed by splitting one of the resulting half-planes into two and so on. The result of recursive binary splitting in some two-dimensional space spanned by (X_1, X_2) is visualized in Figure 2. The partition is the outcome of the following steps: first split the space at $X_1 = t_1$, then split $X_1 < t_1$ at $X_2 = t_2$ and finally split the region $X_1 > t_1$ at $X_1 = t_3$. The result is a partition into the four regions $R_1, ..., R_4$. The panel to the right in Figure 2 contains the decision tree associated with the partition and is an alternate way of representing (10). The diagram should

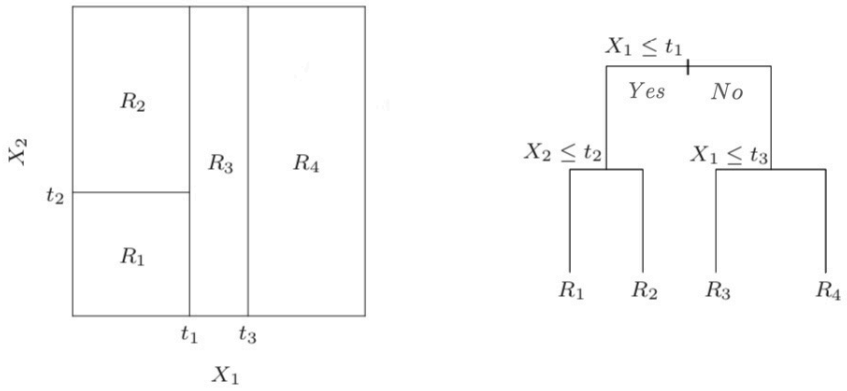

Figure 2: Partitions and CART. The left panel shows a partition of a two-dimensional space given recursive binary splitting, as used in CART. The right panel shows the corresponding decision tree.

be interpreted as follows; given a set of predictor variables (X_1, X_2) with observations $\{x_{1i}, x_{2i}\}$, $i = 1, 2, ..., n$ first determine if $x_{1i} \leq t_1$, this is called the root node. If the statement is $True$ then discern if $x_{2i} \leq t_2$. If the statement was $False$ check to see if $x_{1i} \leq t_3$. Depending on the outcome map x_{1i}, x_{2i} to one of the regions R_j, $j = 1, 2, ..., 4$. The bottom-most nodes of the tree is called leaf nodes.

The intuition behind growing a decision tree is that each region split should result in two new regions that respectively are as homogenous as possible by some metric. Assume that we have found such a split. The objective now is to find the c_j that minimizes the criterion by which we measure homogeneity, that is, to minimize the *training loss function* $l(y, f(x))$. The choice of training loss function depends on the problem at hand but in the case of regression problems the *residual sum of squares*, $RSS = \sum(y_i - f(x_i))^2$ is common and is the empirical metric corresponding to the squared error loss (7). The RSS will be the choice of training loss in this thesis.

Still using the two-dimensional example: given a partition R_m the function that minimizes the RSS is the mean of the y_i associated with the x_{1i}, x_{2i} in that region

$$\hat{\gamma}_j = ave(y_i | x_i \in R_j).$$

In practice the optimal splits, or binary partitions, are found using a *greedy* algorithm. The algorithm is greedy in the sense that it chooses the partition that is optimal at the current step, compared to searching through all possible splits at all possible steps.

Given the data (Y, X) where Y is the response with observations $(y_1, \ldots, y_n)$

and X a set of P predictor variables $X^T = (X_1, \ldots, X_P)$, with observations $x_i^T = (x_{i1}, \ldots, x_{iP})$ consider a partitioning on variable ν and split point s. The resulting half-planes would then be defined as

$$R_1(\nu, s) = \{X | X_\nu \leq s\} \text{ and } R_2(\nu, s) = \{X | X_\nu > s\}.$$

Seeking the splitting variable ν and point s that solve

$$\min_{\nu,s} \left[\min_{\gamma_1} \sum_{x_1 \in R_1(\nu,s)} (y_i - \gamma_1)^2 + \min_{\gamma_2} \sum_{x_1 \in R_2(\nu,s)} (y_i - \gamma_2)^2 \right]. \tag{11}$$

we have that for any ν and s the inner minimization is solved by

$$\hat{\gamma}_1 = ave(y_i | x_i \in R_1(\nu, s)) \text{ and } \hat{\gamma}_2 = ave(y_i | x_i \in R_2(\nu, s)). \tag{12}$$

After the best split has been determined the data is partitioned into the resulting regions and the process is then repeated on the new regions. The end result of the procedure is the mapping $x_i \mapsto R_j \mapsto \gamma_j$, where γ_j are often referred to as the leaf weights.

A remaining question is how many splits should be performed. A large tree, with many splits, might have large variance in the sense discussed in section 2.2 whereas a small tree might not capture important structure in the data and hence be biased. This is an important aspect of growing decision trees and is usually solved using some cost-complexity function that penalizes the model for being too flexible together with the concept of *pruning*. It turns out however that in the setting of gradient boosting the size of the trees grown will preferably be determined by cross-validation, motivating the exclusion of the details of using pruning.

As mentioned previously the trees discussed above are CARTs, specifically regression trees. This is the decision tree implementation used in XGBoost and will thus be the only decision tree variation explored in this thesis.

2.5 Boosting

Boosting is a technique that combines many weak learners into one strong learner, where a weak learner is a predictor which performs barely better than random guessing. The idea of boosting is to train weak learners on modified versions of data in a serial fashion. Formally boosting is a way of fitting an additive expansion in a set of elementary basis functions. An example of a basis function could be a CART, as described in the previous section.

2.5.1 Forward Stagewise Additive Modeling

General basis function expansions take the form

$$f(x) = \sum_{m=1}^{M} \beta_m b(x; \theta_m), \tag{13}$$

where β_m are the expansion coefficients and $b(x; \theta_m)$ the basis functions characterized by a set of parameters θ. In the case of trees for example, θ parameterizes the split variables, split points s and leaf weights γ_m at the regions R_m. Basis functions should be chosen depending on the current problem, typically though basis functions are chosen to have low variance but high bias. The only explicit basis function discussed in this thesis will be trees.

Fitting a function of the form (13) to data is often computationally infeasible, as it is a challenging combinatorial problem, but a model can be approximated by sequentially adding new basis functions without adjusting the parameters and coefficients of those previously calculated. One such iteration would then be solved by minimizing some loss function L over the training data $\mathcal{T}$,

$$\min_{\beta,\theta} \sum_{i=1}^{N} L(y_i, \beta b(x_i; \theta)).$$

Performing several such iterations is the basic idea of *Forward Stagewise Additive Modeling* (FSAM). The first step in FSAM is to initialize the model, setting $f_0 = 0$. Following the initialization: for each iteration m the optimal basis function $b(x_i; \theta_m)$ and coefficient β_m, are solved for

$$(\beta_m, \theta_m) = \arg\min_{\beta,\theta} \sum_{i=1}^{N} L(y_i, f_{m-1}(x_i) + \beta b(x_i; \theta)). \tag{14}$$

The coefficient and basis function $\beta_m b(x; \theta_m)$ is then added to the current expansion f_{m-1} which produces

$$f_m(x) = f_{m-1}(x) + \beta_m b(x; \theta_m,).$$

The procedure is then repeated until some M is reached. Note that the terms added before the current iteration m, $f_0, \ldots, f_{m-1}$, are not modified. The steps described above are the general form of boosting, summarized in Algorithm 1.

2.5.2 Boosted Trees

We recall that (10) is a representation of a tree. For further ease of notation we introduce the equivalent expression,

Algorithm 1 Forward Stagewise Additive Modeling

1: Input:
   Data set $\mathcal{D}$
   A Loss function L
   The number of Iterations M
2: Initialize $f_0 = 0$
3: **for** $m = 1$ to M **do**
4: Compute: $(\beta_m, \theta_m) = \arg\min_{\beta,\theta} \sum_{i=1}^{N} L(y_i, f_{m-1}(x_i) + \beta b(x_i; \theta))$
5: Update: $f_m(x) = f_{m-1}(x) + \beta_m b(x; \theta_m)$.
6: **end for**
7: **return** $\hat{f}(x) = f_M(x)$

$$T(x; \Theta) = \sum_{j=1}^{J} \gamma_j \mathbb{1}\{x \in R_j\} \tag{15}$$

where $\Theta = \{R_j, \gamma_j\}_{j=1}^{J}$. Θ then contains information on all γ_j and R_j for $j = 1, 2, ..., J$. Combining (13) and (15) we can express a function expansion with decision trees as basis function as

$$f_M(x) = \sum_{m=1}^{M} T(x; \Theta_m) = \sum_{m=1}^{M} \sum_{j=1}^{J} \gamma_j \mathbb{1}\{x \in R_j\}. \tag{16}$$

Calculating $f_M(x)$ can be done iteratively in accordance with Algorithm 1. Initialization is done minimizing the loss function over the non-partitioned original space

$$f_0 = \arg\min_{\gamma} \sum_{i=1}^{N} L(y_i, \gamma).$$

The expression of the optimization problem (14) using trees would be expresses as

$$\hat{\Theta}_m = \arg\min_{\Theta_m} \sum_{i=1}^{N} L(y_i, f_{m-1}(x_i) + T(x_i; \Theta_m)) \tag{17}$$

which is to be solved over the set of regions and leaf weights $\Theta_m = \{R_{j_m}, \gamma_{j_m}\}_1^{J_m}$ for the next tree, given the previous trees represented by $f_{m-1}(x)$. Solving for the constants γ_{j_m} in a given region $R_{j_m}$is usually a easy optimization problem,

$$\hat{\gamma}_{j_m} = \arg\min_{\gamma_{j_m}} \sum_{x_i \in R_{j_m}} L(y_i, f_{m-1}(x_i) + \gamma_{j_m}). \tag{18}$$

Algorithm 2 Boosted Trees

1: Input:
Data set $\mathcal{D}$
A Loss function L
The number of Iterations M
The learning rate η
2: Initialize: $f_0 = \arg\min_{\gamma} \sum_{i=1}^{N} L(y_i, \gamma)$
3: **for** $m = 1$ to M **do**
4: Compute: $\hat{\Theta}_m = \arg\min_{\Theta_m} \sum_{i=1}^{N} L(y_i, f_{m-1}(x_i) + T(x_i; \Theta_m))$
implying: $\hat{\gamma}_{j_m} = \arg\min_{\gamma_{j_m}} \sum_{x_i \in R_{j_m}} L(y_i, f_{m-1}(x_i) + \gamma_{j_m})$
5: Update: $f_m(x) = f_{m-1}(x) + \eta \sum_{j=1}^{J_m} \gamma_{j_m} \mathbb{1}(x \in R_{j_m})$
6: **end for**
7: **return** $\hat{f}(x) = f_M(x)$

In the last iterative step the model is updated

$$f_m(x) = f_{m-1}(x) + \eta \sum_{j=1}^{J_m} \gamma_{j_m} \mathbb{1}(x \in R_{j_m})$$

where $0 < \eta \leq 1$ is the learning rate, which will be discussed in Section 2.6.

Solving (17) usually implies optimizing (18) as well. There are however occasions when it is necessary to approximate (17) with a more convenient criterion for finding the R_j, in these cases the R_j are found using a better suited $\tilde{L}$

$$\dot{\Theta} = \{\dot{R}_j, \dot{\gamma}_j\}_{j=1}^{J} = \arg\min_{\Theta} \sum_{i=1}^{N} \tilde{L}(y_i, T(x_i; \Theta))$$

Given the regions $\dot{R}_j$, $j = 1, 2, ..., J$ the much easier optimization problem (18) can then be solved using the original loss function i.e. set $R_{j_m} = \dot{R}_{j_m}$ in (18). Considering (17) and (18) as separate steps might also aid intuition in later sections when comparing boosting with classical optimization methods.

To summarize the general boosted trees method is outlined in Algorithm 2.

2.5.3 The Squared Error Loss Function

A special case of FSAM arises when the loss function is chosen to be the squared error loss $L(y, f(x)) = (y - f(x))^2$, as is a common choice when solving regression problems.

$$
\begin{aligned}
L(y_i, f_{m-1}(x_i) + \beta b(x_i;\gamma)) &= (y_i - f_{m-1}(x_i) - \beta b(x_i;\gamma))^2 \\
&= (r_{im} - \beta b(x_i;\gamma))^2,
\end{aligned}
\tag{19}
$$

where $r_{im} = y_i - f_{m-1}(x_i)$ is the residual of the current model f_{m-1} and the ith observation, often referred to as a pseudo residual, this means that the basis model, and coefficient, $\beta_m b(x_i;\gamma_m)$ that best fits the psuedo residuals will be added to the expansion. The basis model is in that sense trained on the psuedo residuals.

As a consequence of training on the psuedo residuals r_{im}, squared error loss gives observations with large absolute residuals $\mid y_i - f(x_i) \mid$ big influence. This can lead to poor performance in cases when outliers are present or the distribution of Y is long-tailed. We will see however, that this can be a prize worth paying due to some pleasant mathematical properties of the squared error loss in conjunction with gradient boosting. It is also worth noting that outliers and long-tailed distributions to some extent can be accounted for with data transformations.

It was previously stated that the squared error loss had some pleasant properties when used in conjunction with boosted tree models. We shall now see the first reason why. Using the squared error loss (7), simplification (19) and using trees as basis function, (17) simplifies to

$$
\begin{aligned}
\hat{\Theta}_m &= \arg\min_{\Theta_m} \sum_{i=1}^{N} L(y_i, f_{m-1}(x_i) + T(x_i;\Theta_m)) \\
&= \arg\min_{\Theta_m} \sum_{i=1}^{N} (y_i - f_{m-1}(x_i) - T(x_i;\Theta_m))^2 \\
&= \arg\min_{\Theta_m} \sum_{i=1}^{N} (r_{i_m} - T(x_i;\Theta_m))^2
\end{aligned}
\tag{20}
$$

The solution is simply the tree that best predicts the psuedo residuals at the given iteration m. Equation (18) simplifies analogously to

$$
\begin{aligned}
\hat{\gamma}_{j_m} &= \arg\min_{\gamma_{j_m}} \sum_{x_i \in R_{j_m}} L(y_i, f_{m-1}(x_i) + \gamma_{j_m}) \\
&= \arg\min_{\gamma_{j_m}} \sum_{x_i \in R_{j_m}} (y_i - f_{m-1}(x_i) - \gamma_{j_m})^2 \\
&= \arg\min_{\gamma_{j_m}} \sum_{x_i \in R_{j_m}} (r_{i_m} - \gamma_{j_m})^2 \\
&= \bar{r}_{j_m}
\end{aligned}
\tag{21}
$$

where $\bar{r}_m$ is the the mean of the psuedo residuals in each partition R_{j_m}. This is the same result as for a single tree minimizing over RSS, comparison with Equation (11) and (12) is instructive.

2.6 Hyper Parameters

In Section 2.5.2 we saw that the two main hyper parameters for boosted trees are the number of iterations M and the learning rate η, both of which are important for model performance. Furthermore M and η need to be chosen in conjunction as they are not independent.

It was briefly mentioned that tree pruning was not necessary in the context of boosting. This notion will be explored further in discussion of the parameter *max tree depth*, T_{max}.

2.6.1 The Number of Iterations M

Assuming a fixed η, an increase of the number of iterations will affect the complexity of the boosting model. A larger M will allow the model to better fit data and at some point increase variance. For this reason the number of iterations is commonly determined by cross-validation and some early stoppage criterion.

2.6.2 The Learning Rate η

The learning rate η, or *shrinkage*, is intended to alleviate overfitting by shrinking the step taken in prediction space during boosting. A lower η will require a larger number of iterations M, hence impacting computational cost. A smaller learning rate have been shown to improve performance and is recommended to bet set as small as computationally feasible, Ridgeway (2006) [29]. Given a computationally affordable η the number of iterations M can then be determined via cross-validation as suggested in Section 2.6.1.

2.6.3 Max Tree depth T_{max}

First growing a large tree and then pruning it, as is the case for a single decision tree is poor practice in boosting. The reason being that growing a large tree to be pruned implies that the model is the last one in the expansion (16). Recall that the expansion is an ensemble of individual trees. Pruning each tree individually would imply that the tree represents the final model, which clearly is not the case in boosting. The solution is to choose a common tree depth T_{max} for all M trees such that $J_m = J \ \forall m$. Furthermore the number of splits $J-1$ can be interpreted as the order of interactions allowed between independent variables. This implies that J should be chosen to reflect the number of dominant interactions between them.

2.7 Other considerations

Two common practical considerations when construction learning models concern how to deal with categorical predictors and missing values. The following sections aim to provide some guidance.

2.7.1 Categorical Predictors

There are mainly two ways to treat categorical input variables in tree based models. Using *grouped categories* and *independent categories*.

When using grouped categories a predictor with q levels there are $2^{q-1}-1$ possible ways to divide the q categories into two groups. With such a large number of possibilities it is very likely to find a good one, which leads to overfitting.

In the case of independent categories the categorical predictors are encoded as q or $q-1$ binary variables, referred to as one-hot encoding and dummy-variable encoding respectively. Expanding somewhat on dummy variable encoding the process can be described in the following way: given a categorical predictor x_i with C classes dummy encoding entails creating a new set of binary dummy variables $x_i^{(d)} = x_{i_1}^{(d)}, \ldots, x_{i_{C-1}}^{(d)}$. Dummy variable encoding omits one of the C classes which instead is implied by $x_{i_1}^{(d)}, \ldots, x_{i_{C-1}}^{(d)} = 0$. One-hot encoding is very similar but creates the binary variables $x_i^{(o)} = x_{i_1}^{(o)}, \ldots, x_{i_C}^{(o)}$.

Neither of the methods described above deals with categorical predictors with many levels gracefully and such variables are best transformed or avoided.

2.7.2 Missing Values

Missing predictor values are a common problem in real world data sets. In general there are three approaches to missing values. Impute, discard or let the algorithm deal with the missing values. The best approach for dealing with missing data depends on the mechanism producing it. The missing data mechanisms are *missing completely at random* (MCAR), *missing at random* (MAR) and *missing not at random* (MNAR), Little & Rubin (2002, pp. 11-18) [18].

MCAR means that there is no dependency between the missing values and other values, neither observed or missing. MAR on the other hand allows a relationship between the mechanism behind the missing data and observed data. MAR does however require independence between the propensity for missing values and the missing values themselves.

Given a training set $\mathcal{T}$ with response y and predictor variables X, where X contains missing values, the mechanisms can be expressed as follows. Let X_{obs} denote the observed values in X, $Z = (y, X)$ and $Z_{obs} = (y, X_{obs})$. Furthermore, let R be an indicator matrix the size of X with entries ij having ones where x_{ij} is missing and zeroes otherwise. Data is said to be MAR if

$$Pr(R|Z, \theta) = Pr(R|Z_{obs}, \theta)$$

where θ are any parameters of R's distribution. If the distribution of R does not depend on the missing or observed data

$$Pr(R|Z, \theta) = Pr(R|\theta)$$

then data are said to be MCAR. If the missing data mechanism is dependent on the values themselves, e.g. the sickest people dropping out medical studies, the data is said to be MNAR. Discarding or imputing data that is MNAR is not recommended as it distorts the picture of the true population. For the same reason it is preferably avoided to let an algorithm handle MNAR data.

Determining whether or not data is MCAR often requires information about how data has been collected or certain domain knowledge, and is not always an easy task. However, dealing with missing values in categorical predictors can be done by creating a new level for the missing values.

Tree-based methods are well suited to handle missing values, as long as they are MCAR, and usually does so by creating *surrogate variables*. When finding the optimal splits only the observations that are non-missing are considered. After finding the best split a list of surrogate splits are formed. The surrogate list contains the variables and split points that best mimics the original split. The first surrogate mimics the original split the best, the second surrogate the second best and so on. During prediction the model leverages correlation among the predictors and uses surrogate variables to send observations with missing values down the tree.

XGBoost deals with missing values, and sparsity, in a similar fashion. Instead of learning the surrogate variables however, *default direction* at each node is determined. There are two choices of default direction in each branch and the optimal one are learnt from the data.

2.8 Gradient Boosted Trees

In this section we will see that the practice of tree boosting can be implemented to mimic the well known numerical optimization procedure gradient descent.

2.8.1 Gradient Descent

To understand the naming convention of gradient boosting trees some background in numerical optimization will be required.

Let $f(x)$ be a function (perhaps multivariate) defined and differentiable in the neighborhood of all points x in the domain of f. Then $f(x)$ decreases most rapidly in the direction of the negative gradient at point x, $-\frac{\partial f(x)}{\partial x} = -\nabla f(x)$. Given a point x_{m-1} it follows that, if

$$x_m = x_{m-1} - \rho_m \nabla f(x_{m-1}), \tag{22}$$

for some $\rho_m \in \mathbb{R}^+$, then $f(x_m) \geq f(x_{m+1})$. ρ_m, the step length, can be determined using line search

$$\rho_m = \arg\min_{\rho} f(x_{m-1} - \rho \nabla f(x_{m-1})), \tag{23}$$

The intuition behind gradient descent is that given a position x_{m-1} the negative gradient points in the direction the steepest descent in terms of some function $f(x)$. With the direction obtained it remains to determine the step length ρ_m, too big a step might overshoot the target hence it is optimized via line search (23). Given the step $-\rho_m \nabla f(x_{m-1})$ the position x_{m-1} is updated via (22) to x_m. The Gradient descent Algorithm 3 follows from performing these updates iteratively giving rise to a monotonic sequence $f(x_0) \geq f(x_1) \geq f(x_2) \geq ...$ converging to the local minima.

Algorithm 3 Gradient Descent

1: Input:
   Data set $\mathcal{D}$
   A function to be optimized f
   The number of Iterations M
2: Initialize x_0
3: **for** $m = 1$ to M **do**
4: Compute: $\nabla f(x_{m-1})$
5: Compute: $\rho_m = \arg\min_{\rho} f(x_m - \rho \nabla f(x_m))$
6: Update: $x_m = x_{m-1} - \rho_m \nabla f(x_{m-1})$
7: **end for**
8: **return** $\hat{x} = x_M$

2.8.2 Gradient Boosted Trees

We now have the necessary background for showing that Algorithm 1, using trees as basis function can be leveraged to perform an approximation of *gradient descent*.

Assume we would like to perform *gradient descent* over a differentiable loss function $L(y, f(x))$ given a training set $\mathcal{T}$ with the aim of performing prediction. We would soon realize that calculating the gradient $\nabla_f L(y_i, f(x_i))$ wouldn't be feasible since it is only defined at the training data points x_i and the goal is to generalize to new data not in $\mathcal{T}$. Instead we could opt to approximate the (negative) gradient using a tree $T(x_i; \Theta_m)$, using squared error as measure of closeness we have

$$\tilde{\Theta}_m = \arg\min_{\Theta} \sum_{i=1}^{N} (-\nabla_{f_{m-1}} L(y_i, f_{m-1}(x_i)) - T(x_i; \Theta))^2 \tag{24}$$

which fits the tree T to the negative gradient values $-\nabla_{f_{m-1}} L(y_i, f_{m-1}(x_i))$ at a given iteration m. Interpreting (24) as an approximation of the negative gradient and the implied optimization of $\hat{\gamma}_{j_m}$ in (18) as finding the step size ρ_m in (23) is key to realizing that gradient boosted trees performs (approximate) gradient descent.

The gradient tree boosting algorithm 4 is outlined below. Note there is no analogy for the learning rate, η in Algorithm 3. Setting $\eta = 1$ in Algorithm 4 makes comparison of the algorithms clearer.

Algorithm 4 Gradient Boosted Trees

1: Input:
   Data set $\mathcal{D}$
   A Loss function L
   The number of Iterations M
   The learning rate η
2: Initialize $f_0 = \arg\min_{\gamma} \sum_{i=1}^{N} L(y_i, \gamma)$
3: **for** $m = 1$ to M **do**
4: Compute: $g_m = \nabla_{f_{m-1}} L(y_i, f_{m-1}(x_i))$
5: Fit T to g_{i_m} : $\tilde{\Theta}_m = \arg\min_{\Theta} \sum_{i=1}^{N} (-g_m - T(x_i; \Theta))^2$
   For $j = 1, 2, ..., J_m$ Compute: $\gamma_{j_m} = \arg\min_{\gamma} \sum_{x_i \in R_{j_m}} L(y_i, f_{m-1}(x_i) + \gamma)$
6: Update: $f_m(x) = f_{m-1}(x) + \eta \sum_{j=1}^{J_m} \gamma_{j_m} \mathbb{1}(x \in R_{j_m})$.
7: **end for**
8: **return** $\hat{f}(x) = f_M(x)$

We illustrate the steps of Algorithm 4 using squared error (7) as loss function and training set $\mathcal{T}$. We start by initializing f_0,

$$f_0 = \arg\min_{\gamma} \sum_{i=1}^{N} L(y_i, \gamma) = \arg\min_{\gamma} \sum_{i=1}^{N} (y_i - \gamma)^2 = \bar{y}$$

For a given iteration m, the gradient of the squared error loss function (7) is

$$\frac{\partial L(y_i, f_{m-1}(x_i))}{\partial f_{m-1}(x_i)} = \frac{\partial}{\partial f_{m-1}(x_i)} (y_i - f(x_i))^2 \propto f_{m-1}(x_i) - y_i$$

which we recognize as the negative psuedo residuals $-r_{i_m}$. Fitting a Tree T to $g_{i_m} = -r_{i_m}$ yields

$$\tilde{\Theta}_m = \arg\min_{\Theta} \sum_{i=1}^{N} (r_{i_m} - T(x_i; \Theta))^2 \tag{25}$$

we note that the choice of squared error loss leads to same expression as in standard least squares boosting, compare with (20). This is true for the predictions γ_{j_m} as well and according to (21) we have $\gamma_{j_m} = \bar{r}_{j_m}$. Recognizing that the tree structure

$$T(x; \tilde{\Theta}) = \sum_{j=1}^{J_m} \bar{r}_{j_m} \mathbb{1}(x \in R_{j_m})$$

is analogous to the negative gradient and step size, together representing a step in prediction space. At the end of each iteration the model is updated

$$f_m(x) = f_{m-1}(x) + \eta \sum_{j=1}^{J_m} \bar{r}_{j_m} \mathbb{1}(x \in R_{j_m})$$

and finally, after M iterations, the ensemble model is returned

$$\hat{f}(x) = f_M(x).$$

2.9 eXtreme Gradient Boosting

As mentioned in the first chapter XGBoost, developed by Chen & Guestrin (2016) [5], is short for *eXtreme Gradient Boosting* and is similar to gradient boosting. Both methods fits additive tree models, as discussed in Section 2.5, but differ in regularization techniques and how they find the optimal tree structure and weights of the leaf-nodes. We will begin by determining the boosting algorithm used in XGBoost.

2.9.1 Newton's Method

In Section 2.8 we saw that Gradient boosted trees mimics the gradient descent algorithm. It turns out that XGBoost also approximates a well known optimization algorithm, namely Newtons method for optimization. Given a twice differentiable function f to be optimized the idea of newtons method is to approximate a second order function, around some point x_m, and to step to a local optimum of that approximation. By constructing a sequence of such steps a stationary point of f is eventually reached. It should be noted that convergence isn´t guarantied but further details on the method is not within the scope of this thesis, the interested reader can find rigorous discussion in Dahlquist & Björk (2012) [7]. In the one-dimensional case

the second order approximation of f around x_m follows from second order Taylor expansion

$$f(x) = f(x_m + \Delta x_m) \approx f(x_m) + f'(x_m)\Delta x + \frac{1}{2}f''(x_m)\Delta x_m^2. \tag{26}$$

We proceed by finding the stationary point of (26)

$$\arg\min_{\Delta x} \left[f'(x_m)\Delta x + \frac{1}{2}f''(x_m)\Delta x^2 \right]$$

which yields

$$\Delta x_m = -\frac{f'(x_m)}{f''(x_m)}. \tag{27}$$

Equation (27) is often referred to as the Newton step. We note that both the direction and size of the step is determined at once, in contrast to gradient descent. Finally we update the position.

The one-dimensional algorithm of Newtons method for optimization is summarized in Algorithm 5.

Algorithm 5 Newtons Method for Optimization

Input:
Data set $\mathcal{D}$
A function to be optimized f
The number of Iterations M
Initialize x_0
for $m = 1$ to M **do**
  Compute: $f'(x_{m-1})$
  Compute: $f''(x_{m-1})$
  Compute: $\Delta x_{m-1} = \arg\min_{\Delta x} \left[f'(x_m)\Delta x + \frac{1}{2}f''(x_m)\Delta x^2 \right]$
  Update: $x_m = x_{m-1} - \Delta x_{m-1}$.
end for
return $\hat{x} = x_M$

The representation of Algorithm 5 might seem cumbersome to the familiar reader. This is by intent, the notation is chosen so as to simplify generalization to tree boosting.

2.9.2 XGBoost

The following exposition on XGBoost is mainly based on Chen & Guestrin's (2016) [5] paper.

Given a twice differentiable, convex, Loss function L, and a training dataset $\mathcal{T}$ the function to be minimized at iteration m during tree boosting

can be expressed as

$$\sum_{i=1}^{N} L(y_i, f_m(x_i)) = \sum_{i=1}^{N} L(y_i, f_{m-1}(x_i) + T(x_i, \Theta)). \tag{28}$$

With a second order Taylor expansion around $f_{m-1}(x_i)$ Equation (28) equals

$$\sum_{i=1}^{N} \left[L(y_i, f_{m-1}(x_i)) + g_m(x_i)T(x_i, \Theta) + \frac{1}{2} h_m(x_i)T(x_i, \Theta)^2 \right]$$

where $g_m(x_i) = \frac{\partial L(y_i, f_{m-1}(x_i))}{\partial f_{m-1}(x_i)}$ and $h_m(x_i) = \frac{\partial^2 L(y_i, f_{m-1}(x_i))}{\partial^2 f_{m-1}(x_i)}$ are the empirical gradient and hessian, both solely based on the data points of $\mathcal{T}$. Removing the constant terms, with regard to Θ, yields the following expression

$$\arg\min_{\Theta} \sum_{i=1}^{N} \left[g_m(x_i)T(x_i, \Theta) + \frac{1}{2} h_m(x_i)T(x_i, \Theta)^2 \right]. \tag{29}$$

Leveraging the disjoint nature of trees we rewrite the criterion to be minimized (the sum in (29)) as

$$\begin{aligned} \sum_{i=1}^{N} \left[g_m(x_i) \sum_{j=1}^{J} \gamma_{j_m} \mathbb{1}\{x \in R_{j_m}\} + \frac{1}{2} h_m(x_i) \sum_{j=1}^{J} \gamma_{j_m} \mathbb{1}\{x \in R_{j_m}\}^2 \right] = \\ \sum_{j=1}^{J} \sum_{x_i \in R_{j_m}} \left[g_m(x_i)\gamma_{j_m} + \frac{1}{2} h_m(x_i)\gamma_{j_m}^2 \right] \end{aligned} \tag{30}$$

where $x_i \in R_{j_m}$ denote the set of x_i mapped to region R_{j_m}. We simplify the expression further using $G_{j_m} = \sum_{x_i \in R_{j_m}} g_m(x_i)$ and $H_{j_m} = \sum_{x_i \in R_{j_m}} h_m(x_i)$, (30) then equals

$$\sum_{j=1}^{J} \left[G_{j_m}(x_i)\gamma_{j_m} + \frac{1}{2} H_{j_m}(x_i)\gamma_{j_m}^2 \right]. \tag{31}$$

Given some fixed regions R_{j_m} the leaf weights are then given by

$$\begin{aligned} \tilde{\gamma}_{j_m} &= \arg\min_{\gamma_{j_m}} \sum_{j=1}^{J} \left[G_{j_m}(x_i)\gamma_{j_m} + \frac{1}{2} H_{j_m}(x_i)\gamma_{j_m}^2 \right] \\ &= -\frac{G_{j_m}}{H_{j_m}}, \ j = 1, 2, ..., J. \end{aligned} \tag{32}$$

Plugging (32) into (31) we get the best possible loss reduction for a given tree structure

$$-\frac{1}{2}\sum_{j=1}^{J}\frac{G_{j_m}^2}{H_{j_m}}. \tag{33}$$

It is helpful to regard (33) as a measure of how good a tree structure $T(x_i, \Theta)$ is. Given such a measure we can go ahead and find an optimal tree structure. Using a greedy approach the improvement in structure, or the *Gain*, when splitting a leaf node into two new nodes Left, L and Right, R can be decomposed into

$$Gain = \frac{1}{2}\left[\frac{G_L^2}{H_L} + \frac{G_R^2}{H_R} - \frac{(G_L+G_R)^2}{(H_L+H_R)}\right] = \frac{1}{2}\left[\frac{G_L^2}{H_L} + \frac{G_R^2}{H_R} - \frac{G_{j_m}^2}{H_{j_m}}\right] \tag{34}$$

where the first term represents the score on the new Left node, the second the new Right node and the last term is the score of the original leaf node.

Algorithm 6 eXtreme Gradient Boosting

1: Input:
   Data set $\mathcal{D}$
   A Loss function L
   The number of Iterations M
   The learning rate η
2: Initialize $f_0 = \arg\min_{\gamma} \sum_{i=1}^{N} L(y_i, \gamma)$
3: **for** $m = 1$ to M **do**
4: Compute: $g_m(x_i) = \frac{\partial L(y_i, f_{m-1}(x_i))}{\partial f_{m-1}(x_i)}$
5: Compute: $h_m(x_i) = \frac{\partial^2 L(y_i, f_{m-1}(x_i))}{\partial^2 f_{m-1}(x_i)}$
6: Determine $T(x_i, \tilde{\Theta}_m)$ by choosing splits that maximize

$$Gain = \frac{1}{2}\left[\frac{G_L^2}{H_L} + \frac{G_R^2}{H_R} - \frac{G_{j_m}^2}{H_{j_m}}\right]$$

and leaf weights $\{\tilde{\gamma}_{j_m}\}_{j=1}^{J_m}$

$$\tilde{\gamma}_{j_m} = -\frac{G_{j_m}}{H_{j_m}}$$

7: Update: $f_m(x) = f_{m-1}(x) + \eta \sum_{j=1}^{J_m} \tilde{\gamma}_{j_m} \mathbb{1}(x \in \tilde{R}_{j_m})$.
8: **end for**
9: **return** $\hat{f}(x) = f_M(x)$

Summation of these steps yields the XGBoost algorithm, outlined in Algorithm 6. Comparing Algorithm 5 and 6 we can draw the conclusion that XGBoost approximates Newtons method for optimization.

2.9.3 The Squared Error Loss Function

For the squared error loss $L(y_i, f_m(x_i)) = (y_i - f(x_i))^2$ the empirical gradient and hessian are respectively $g_m(x_i) = -2r_{i_m}$ and $h_m(x_i) = 2$. Plugging this into (29) yields

$$\arg\min_{\Theta} \sum_{i=1}^{N} \left[-2r_{i_m} T(x_i, \Theta) + T(x_i, \Theta)^2 \right]. \tag{35}$$

Completing the square in (35) and removing the constant results in the following expression to be minimized

$$\sum_{i=1}^{N} \left[(r_{i_m} - T(x_i, \Theta))^2 - r_{i_m}^2 \right] \propto \sum_{i=1}^{N} (r_{i_m} - T(x_i, \Theta))^2,$$

which we recognize as the same function to be minimized in (25). In other words, XGBoost performs gradient boosting when using the squared error loss function.

2.10 XGBoost Parameters

In section 2.6 we saw that the number of iterations M, learning rate η and maximum tree depth T_{max} all are hyper parameters associated with tree boosting. In the following sections we shall see that XGBoost offers a variety of other regularization options.

2.10.1 Tree Parameters

We already saw that setting the maximum tree depth T_{max} was a way of affecting the shape of the tree. Besides the number of splits we can influence the tree structure by determining a minimum leaf weight, h_{min} as well.

Every leaf node corresponds to a terminal region in the feature space. The observations in these regions contains observation weight equaling the hessian. To avoid overfitting a restriction of the minimum sum of observation weight h_{min} can be set. If the sum of instance weight (the hessian) is less than h_{min} the tree wont grow any bigger. In the case of squared error loss $\frac{1}{2}(y_i - f(x_i))^2$ the hessian h_i with regard to $f(x_i)$ is 1, it then follows that $h_{min} = \sum_{i=1}^{N} h_i = N$, the number of observations in the region. This shows how restricting the hessian helps avoiding overfitting as fitting to a small amount of observations would increase variance.

2.10.2 Complexity penalizing parameters

XGBoost offers several options for complexity control. This is achieved by introducing the objective function $\omega(\Theta) = l(\Theta) + \Omega(\Theta)$, the sum of the training loss function and a *regularization* term. The regularization term is defined as

$$\Omega(\Theta) = \Gamma J + \frac{1}{2}\lambda \sum_{j=1}^{J} \gamma_j^2 + \alpha \sum_{j=1}^{J} |\gamma_j|.$$

where $\Gamma, \lambda, \alpha > 0$.

In analogy with Equation (31) we derive the following expression of the objective function at iteration m

$$\omega_m(\Theta) = \sum_{j=1}^{J_m} \left[G_{j_m}\gamma_{j_m} + \frac{1}{2}(H_{j_m} + \lambda)\gamma_{j_m}^2 + \alpha|\gamma_{j_m}| \right] + \Gamma J_m. \tag{36}$$

We find the leaf weight that minimizes the objective function by differentiating with respect to γ_{j_m} and setting the result to zero. given a fixed tree structure, iteration m, and regularization term $\Omega(\Theta)$ we have two cases depending on if $\gamma_j \leq 0$ or if $\gamma_j > 0$.

Case 1: $\gamma_j \leq 0$

$$\frac{\partial \omega(\Theta)}{\partial \gamma_j} = G_j + (H_j + \lambda)\gamma_j - \alpha = 0$$

solving for γ_j yields

$$-\frac{(G_j - \alpha)}{(H_j + \lambda)} = \gamma_j \leq 0 \implies G_j > 0 \tag{37}$$

since $L(\Theta)$ is convex and H_j, under the current restrictions, is the second derivative of the loss function. Furthermore we need $G_j - \alpha \geq 0$ for the inequality of Equation (37) to hold. We can then write

$$G_j - \alpha = \text{sign}(G_j)\max(0, |G_j| - \alpha)$$

The derivation for **Case 2:** $\gamma_j > 0$ is analogous.

The leaf weight that minimizes the objective function given a fixed tree structure and regularization term is then

$$\ddot{\gamma}_j = -\frac{L_\alpha(G_j)}{H_j + \lambda}$$

where

$$L_\alpha(G_j) = \text{sign}(G_j)\max(0, |G_j| - \alpha)$$

Plugging this into Equation (36) for the cases $G_j < -\alpha$, $G_j > \alpha$ and $G_j = \alpha$ respectively we get the best possible regularized reduction for a given tree structure

$$-\frac{1}{2}\sum_{j=1}^{J_m}\frac{L_\alpha(G_{j_m})^2}{H_{j_m}+\lambda}+\Gamma J_m.$$

We can then express the regularized gain function at a given split as

$$Gain=\frac{1}{2}\left[\frac{L_\alpha(G_L)^2}{H_L+\lambda}+\frac{L_\alpha(G_R)^2}{H_R+\lambda}-\frac{L_\alpha(G_L+G_R)^2}{(H_L+H_R)+\lambda}\right]-\Gamma \tag{38}$$

A more in depth discussion of the regularization parameters follows below. For ease of discussion all regularization parameters except the current one will be assumed to be zero.

2.10.2.1 Gamma, Γ

Γ controls the number of leaf nodes in a tree by penalizing the *Gain* function (34).

$$Gain=\frac{1}{2}\left[\frac{G_L^2}{H_L}+\frac{G_R^2}{H_R}-\frac{(G_L+G_R)^2}{(H_L+H_R)}\right]-\Gamma$$

This translates to smaller *Gain* for more potential splits, resulting in shallower trees. The Γ parameter control the tree structure but not the leaf weights themselves. Using Γ for complexity control can be viewed as an adaptive option as compared to the hard constraint of T_{max} for example.

2.10.2.2 $l2$ regularization, λ

λ, the $l2$ *regularization* parameter and the shrinkage parameter η have similar effect. λ does in fact shrink the leaf weights as seen below

$$\ddot{\gamma}_j=-\frac{G_j}{H_j+\lambda}.$$

However, while η shrinks the leaf nodes by an equal amount the shrinkage provided by λ will vary depending on the node.

Examining the *Gain* function it is also clear that λ influences the tree structure.

$$Gain=\frac{1}{2}\left[\frac{G_L^2}{H_L+\lambda}+\frac{G_R^2}{H_R+\lambda}-\frac{(G_L+G_R)^2}{(H_L+H_R)+\lambda}\right]$$

It does so by impacting which splits are being made. The effect of λ is that regions with small hessian (corresponding to few observations in the case of the squared error loss) and thus higher variance will be penalized the most.

2.10.2.3 $l1$ regularization, α

α, the $l1$ *regularization* parameter performs a similar function as λ by shrinking the leaf weights. $l1$ regularization however may shrink the weights to zero analogous to lasso regression. When all other regularization parameters but α is zero we find that

$$\ddot{\gamma}_j = -\frac{L_\alpha(G_j)}{H_j}$$

as $L_\alpha(G_j) = \text{sign}(G_j)\max(0, |G_j| - \alpha)$. A large α will make the model less flexible as well as speed up calculations. Inspecting the $Gain$ function

$$Gain = \frac{1}{2}\left[\frac{L_\alpha(G_L)^2}{H_L} + \frac{L_\alpha(G_R)^2}{H_R} - \frac{L_\alpha(G_L + G_R)^2}{(H_L + H_R)}\right]$$

we can se that $l1$ regularization influences tree structure as well as leaf weights.

2.10.3 Randomization Parameters

Subsampling in combination with tree based ensemble methods have proven successful. Subsampling is the basic idea behind both *bagged trees* and *random forests*. The details of those methods are outside the scope of this thesis but the principle behind them both is to reduce variance be decreasing correlation between individual trees in the ensemble. Subsampling achieves this by fitting trees to different subsets of the training data, thus generating more diverse trees. An explicit advantage of randomization is that the influence of outliers are reduced as they are more rarely available when splits are chosen and weights calculated.

(a) **Row Subsampling, ζ_r**
The fraction $0 < \zeta_r \leq 1$ determines the subsample ratio of the training instances. Subsampling occurs once every boosting iteration.

(b) **Column Subsampling by Tree, ζ_{c_t}**
The subsample ratio of input variables. Subsampling occurs once every boosting iteration. $\zeta_{c_t} \in (0, 1]$.

(c) **Column Subsampling by level, ζ_{c_l}**
Like with subsampling by tree, ζ_{c_l} is the fraction of features to be used, the difference lies in when the sampling occurs. In the case of subsampling by level a new sample is drawn each time a new split is made. Just as with both of the above parameters ζ_{c_l} takes values larger than zero but no larger than one.

3 Data

The dataset was collected from Booli using their API and contains a little more than 300,000 observations across 30 variables and reflects almost four years of time. Booli uses web-scraping to collect the data themselves. The homepages web-scraped are those of individual real estate agencies and construction companies, who advertise the housing objects for sale.

The data pertain to the Swedish housing market and each observation represents the transaction of an apartment. Most of the variables describe the housing object e.g. list price, size, number of rooms and the like. Besides information regarding the property the data contains geo-spatial and temporal data, i.e. dates and location. A description of the variables can be found in Table 1.

3.1 Exploratory Data Analysis

To gain some preliminary insights about the Booli data and the interaction between the variables *exploratory data analysis* (EDA) will be performed.

One of the first proponents of EDA was Tukey (1962) [35] who was of the inclination that data analysis shouldn't only be inferential, in the sample-to-population sense, but also a guide for considerations in observations, experimentation and analysis.

The objectives of EDA in statistical learning are similar to those using more traditional methods. Where suggestion of hypothesis, assessing assumptions and selection of statistical tools are familiar objectives of data analysis. Besides those objectives EDA is performed in conjunction with statistical learning to aid in *feature engineering*, a topic which will be discussed more in depth in a later section.

3.1.1 Response Variable

The response variable is this thesis is *soldPrice*, the price paid for a given apartment at a certain time. SoldPrice is numerical and measured in Swedish Krona (SEK). To ease readability soldPrice will be referred to as sold price or just price in instances where there is no ambiguity.

Table 2 contains the five number summary for sold price. Inspecting the table we see that the most expensive apartment was sold for 57 million SEK and the cheapest for only 15,000 SEK. More importantly sold price might be characterized by an asymmetric distribution. This seems plausible given the differences between quantiles, min and max. The box- and density-plot of soldPrice in Figure 3 confirms this suspicion. We note the presence of outliers above 30 million SEK. It is difficult to distinguish more features of the data at this scale as the distribution are highly skewed. It may well prove worthwhile to transform sold price when fitting the model as

Table 1: Description of Variables

	Variable Name	Description
1	location.address.streetAddress	Street address
2	location.position.latitude	Coordinate: Latitude
3	location.position.longitude	Coordinate: Longitude
4	location.namedAreas	Local area name
5	location.region.municipalityName	Municipality name
6	location.region.countyName	County name
7	location.distance.ocean	Distance to ocean
8	listPrice	List price, SEK
9	rent	Rent, SEK
10	floor	Storey of building
11	livingArea	Size of living area, m^2
12	source.name	Data provider, e.g. real estate agencys name
13	source.id	Data provider ID
14	source.type	Type of data provider
15	source.url	Data provider URL
16	rooms	Number of rooms
17	published	Date when listed
18	constructionYear	Year of construction
19	objectType	Type of housing e.g. apartement
20	booliId	Internal Booli ID
21	soldDate	Date when sold
22	soldPrice	Sold price, SEK
23	soldPriceSource	How sold price was collected, e.g. bid history
24	url	URL to Booli advertisment
25	plotArea	Size of plot area, m^2
26	additionalArea	Size of additional area, m^2
27	apartmentNumber	Apartment number
28	isNewConstruction	Indicator of new construction
29	location.address.city	City name
30	location.position.isApproximate	Indicator of approximate position

Table 2: Summary of soldPrice

Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
15,000	1,200,000	1,995,000	2,338,789	2,975,000	57,000,000

tree based models using MSE loss tend to perform better on distributions without heavy tails, as mentioned in Section 2.5.3.

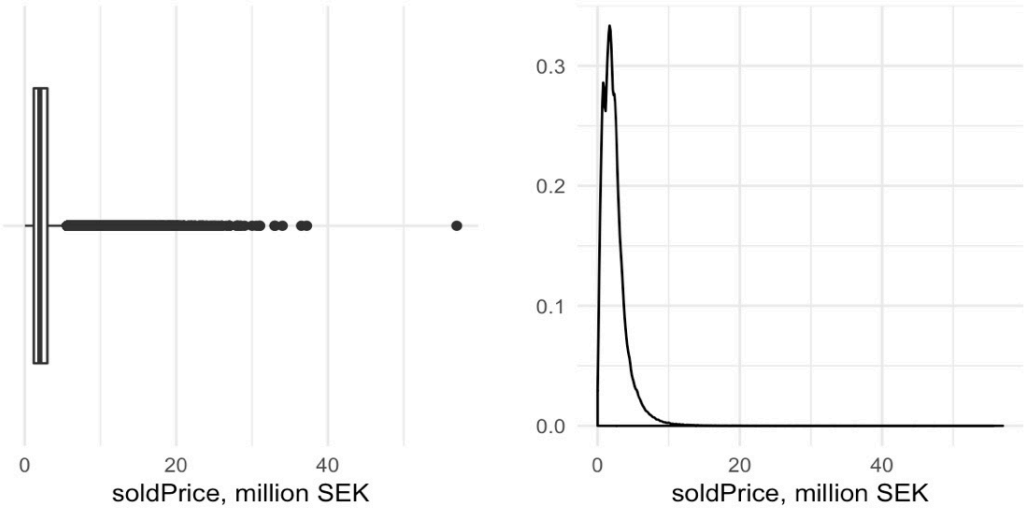

Figure 3: Box and density plot of soldPrice. The left panel shows the boxplot of sold price i million SEK. The panel to the right shows the corresponding densityplot.

3.1.2 Interactions with Predictor Variables

In order to gain further intuition about the Booli data we explore some of the interactions between the response and predictor variables. Leveraging domain knowledge seems like a good place to start and ortprismetoden provides initial guidance.

The aim is not to perform en exhaustive analysis, nor is it to declare statistical significance but rather to establish some rudimentary understanding of the data.

3.1.2.1 Sold Date

One of the strongest assumptions about housing prices are that they change over time. This assumption is central in ortsprismetoden as well. The left panel of Figure 4 shows an correlogram of monthly average apartment prices. The correlogram plots the autocorrelations versus time lags, each bar

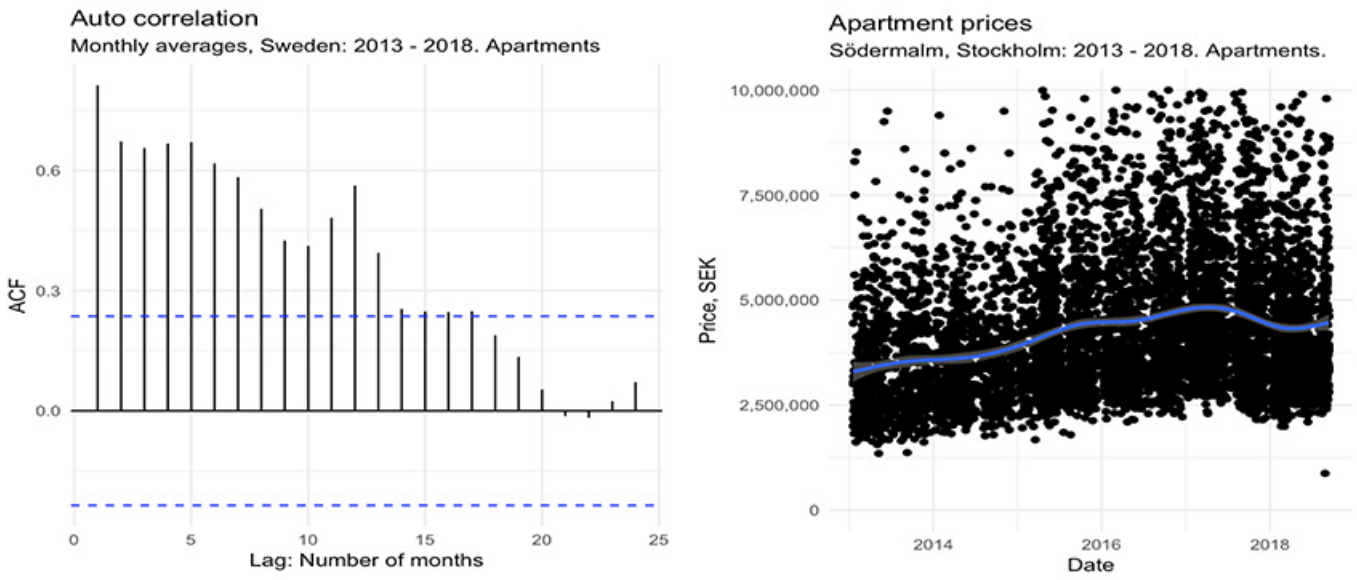

Figure 4: Correlogram and scatterplot of sold price over time.

representing autocorrelation at some lag. The dotted blue lines represent the significance bounds for when the autocorrelation differ from zero. We can clearly see that prices depend on previous values, even though correlations seems to dissipate after about a year. We also note the spike at 12 months lag, perhaps implying some seasonality effect. Over the course of two years it is clear however that prices fluctuate. This notion is supported by the panel to the right which shows changing apartment prices in Stockholm, Södermalm between early 2013 until late 2018.

3.1.2.2 Location

As implied by the well known expression: *'There are three things that matter in property: location, location, location.'* geo-spatial influence on housing prices are assumed to be strong. We assert to confirm this item of common knowledge and plot Swedish apartment prices i Figure 5. Judging by the figure this particular realtor catch-phrase seems to have merit. Most apparent is the concentration of expensive apartments in metropolitan areas, in particular Stockholm and Göteborg.

3.1.2.3 Living Area

Apartment prices are often assumed to scale linearly with the size of the living area. We aim to investigate. To reduce the temporal and geospatial influence we plot the relationship between Sold price and living area for a single month, both for the whole country and a selected city. The results are

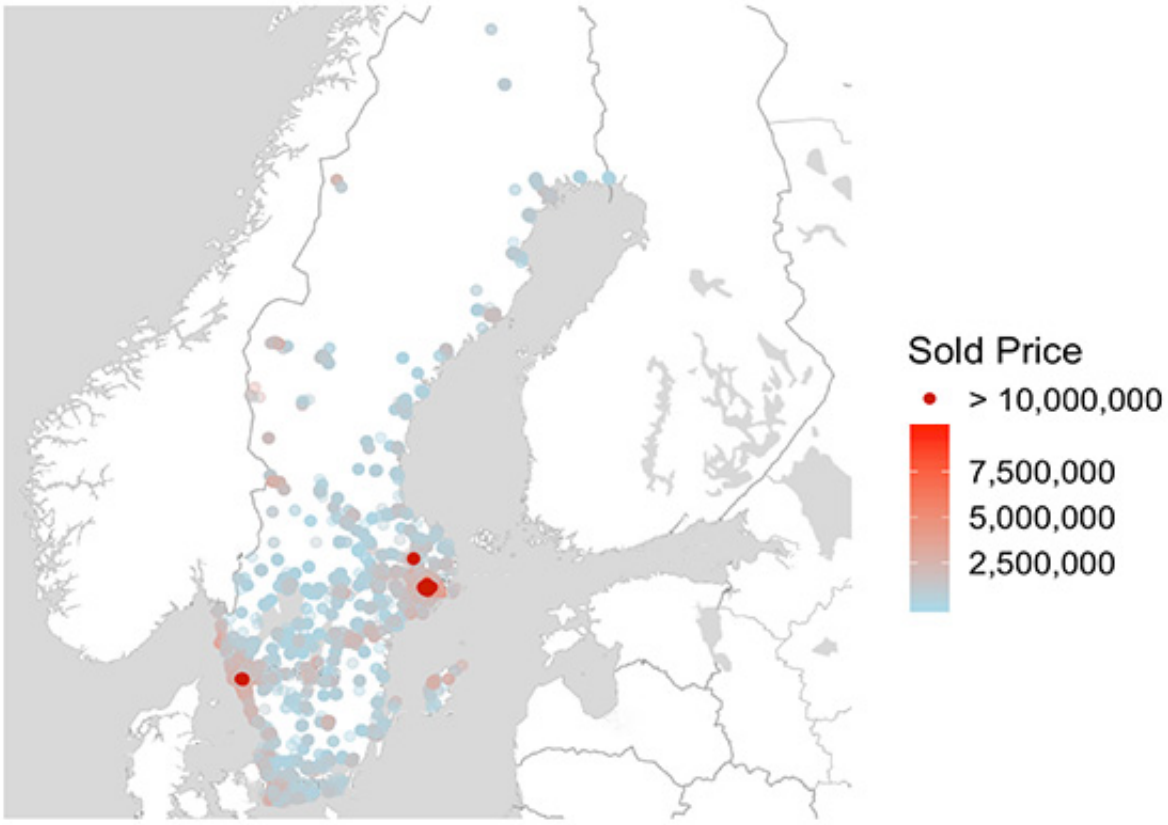

Figure 5: Swedish apartment prices by location during 2017.

presented in figure 6. The strength of the linear dependence is not obvious from the left panel which shows sold prices from all of Sweden. The plotted points resemble a cone rather than a straight line. A typical shape for a heteroscedastic random variable. Heteroscedasticity would be expected when plotting several subpopulations that are individually linearly dependent but with different coefficients. The plot in the right panel supports this notion as prices in the city of Solna is clearly linear with respect to living area. We also note that this data appears to show heteroscedasticity, perhaps for the same reason as above. There could of course be other reasons that variability increases as living area does but further speculation is refrained from.

3.1.2.4 Number of Rooms

Small apartments are often said to sell at a premium compared to their larger counterparts. This might sounds as opposing the idea of linearity between price and living area and perhaps it is, we really haven't investigated thoroughly enough. There is however another possibility, perhaps the number of rooms are what causes the premium. Looking at Figure 7 it does

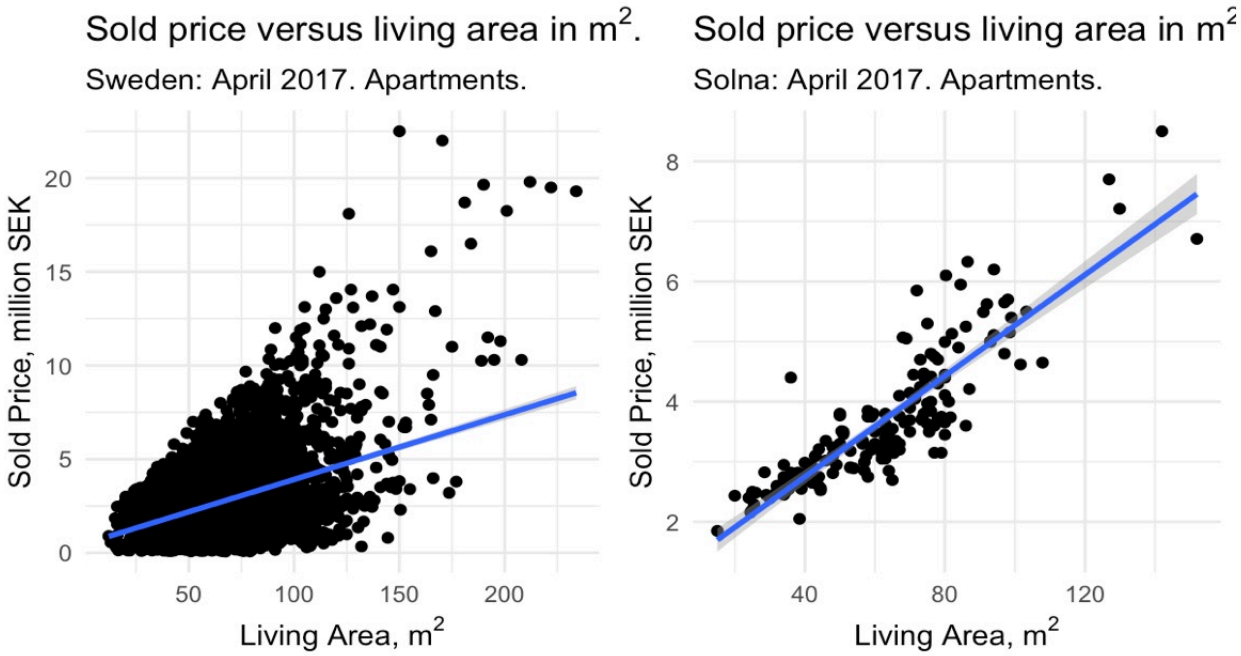

Figure 6: Swedish apartment prices versus living area in April 2017. The left pane shows observations from all of Sweden whereas the right shows data from the municipality of Solna.

indeed seem plausible that studios entail a higher price per square meter. Somewhat surprisingly large apartments, with six or more rooms, also exhibit a similar premium. For whatever reason the price per m^2 by number of rooms seem to have a nonlinear relationship.

3.2 Data Preparation

Creating a successful model using statistical learning depends on many factors. However, one of the single most important are the representation and quality of the data according to Kotsiantis et al (2006) [17]. Data preparation includes dealing with missing values and outliers, transformation and such. The product of the preparation will be the instance data by which the model is trained.

In the following paragraphs the data preparation performed in this thesis will be motivated.

3.2.1 Data Preprocessing

A first step of data preparation entails preprocessing. In the case of this thesis this includes managing datatypes, missing values and outliers.

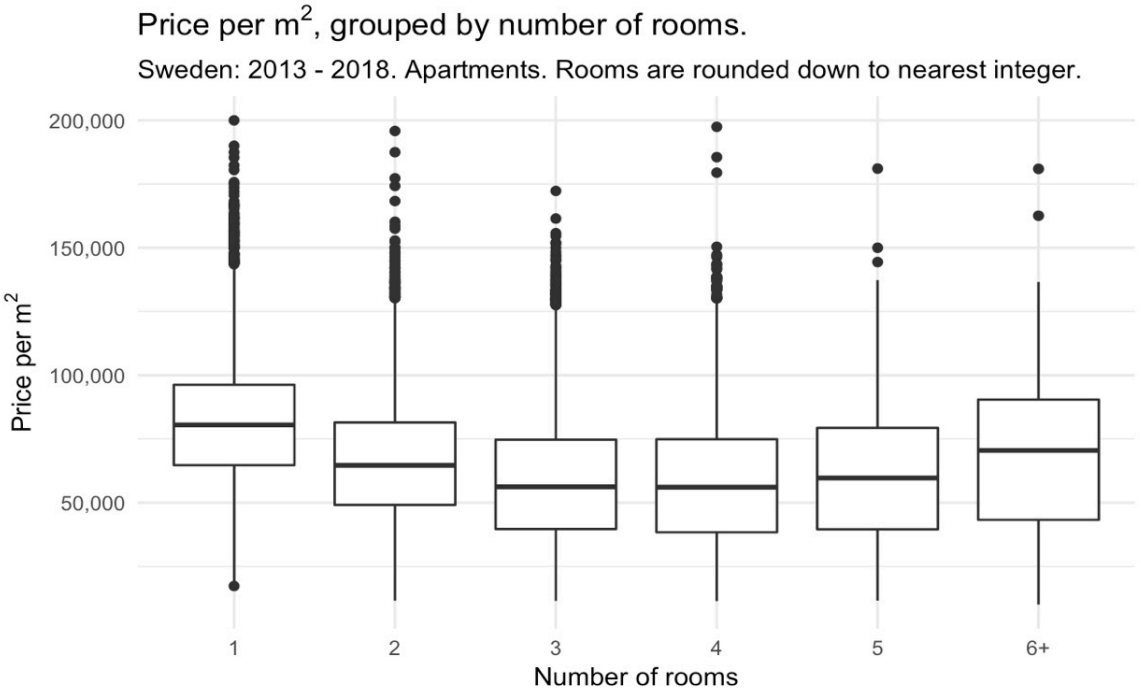

Figure 7: Distributions for sold price per m^2 by number of rooms.

3.2.1.1 Data Types

The Booli data set contains several variables containing character values. These variables need to be redefined to appropriate datatypes for them to make sense to the learning algorithm as well as aid feature engineering. Examples of datatype transformations in the Booli dataset are $'character' \rightarrow 'date'$ and $'character' \rightarrow 'factor'$.

3.2.1.2 Missing Values

As discussed in Section 2.7.2 the task of managing missing values (NAs) should begin by determining which kind of process that's producing them. Figure 8 visualizes the missing values of the Booli dataset. Some of the predictors almost only contain missing values! The missing values of those predictors are almost certainly not MCAR. While disheartening at first glance there seems to be an easy explanation. A plausible explanation is that Boolis data collecting web-scraping method encodes proper values when certain information e.g. apartment number, is present and NAs otherwise. This would also explain why some predictors only contain ones and NAs. Treating the categorical variables showing these signs are easy as it only takes recoding the missing values as a new level 'unknown'. In the cases where the predictor only takes the values 1 or NA the NAs are codes as 0. Among

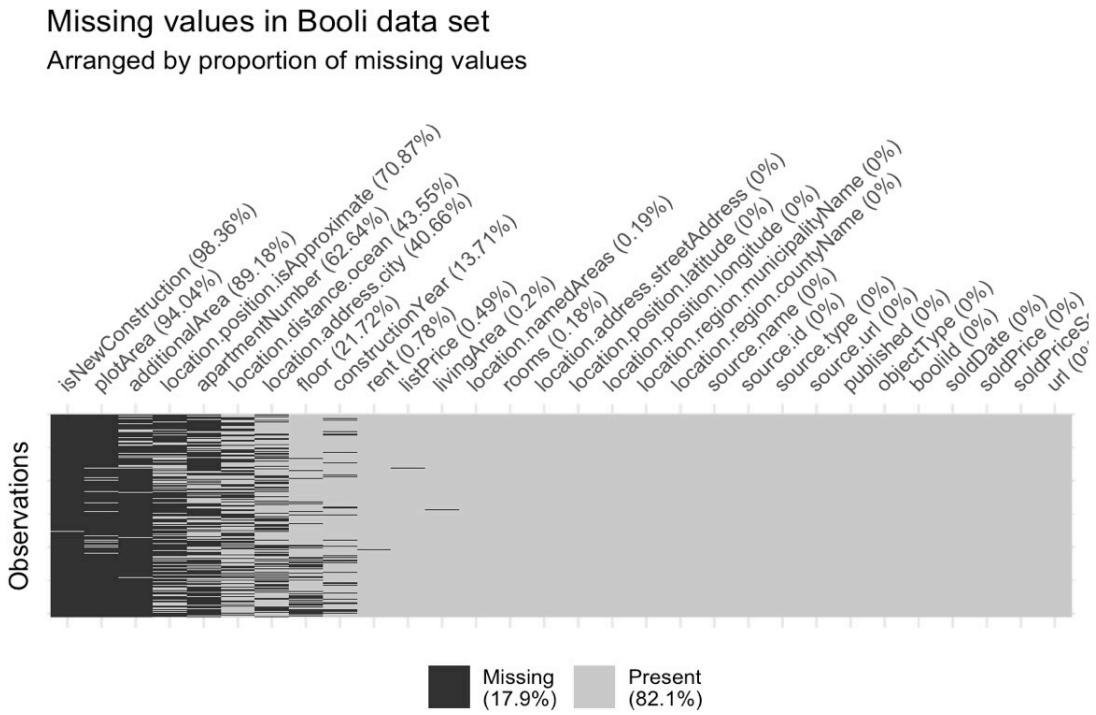

Figure 8: Visualization of missing values among the variables of the Booli dataset.

the numeric predictors containing many ($> 10\%$) missing values the solution is similar, take for example plotArea. It seems very likely that most of the apartments in Sweden do not have a plot, hence it is likely seldom specified in real estate ads. In these cases the NAs are recoded as 0. The remaining numerical predictors with many NAs are binned into discrete categories, e.g. constructionYear is binned into decades.

The few observations still containing missing values after the above process are discarded.

3.2.1.3 Outliers

As mentioned in Section 2.5.3 gradient boosting using the squared error loss is susceptible to extreme values. This is especially true for outliers in the response variable, but care should be given to the predictor variables as well. Outliers and extreme values are often used synonymously, as is the case in this thesis. There should however be a distinction between values arising from erroneous recordings and values arising from fat tailed distributions.

There are few ways to deal with outliers when using XGBoost. Given that a value of an observation is deemed erroneous one might remove them.

This does however require a robust way of determining which observations should be considered erroneous, an often subjective process which might degrade performance should proper care not be taken. A second option is to transform the data in ways that make the values less extreme. Another is to use the regularization options presented in Section 2.10. The randomization parameters are especially useful when dealing with extreme values. A last option is to use a loss function more robust to extreme values, like the absolute loss $L(y, f(x)) = |y - f(x)|$.

The approach to dealing with extreme values in this thesis will be a combination of discarding erroneous values, transformations and parameter tuning. Since the choice of squared-error loss is a delimitation of this thesis it will not be substituted. Values removed for being considered erroneous were observations with living area less then or equal to one.

3.2.2 Data Transformation

As mentioned in Section 3.2 the performance of statistical learning models is severely influenced by proper transformation of the instance data. Despite this fact specific recommendations on how to best transform input data is scarce in statistical learning literature. Heaton (2016) [15] does however provide some guidance for tree based models specifically, suggesting logarithmic transforms among others.

3.2.2.1 Encoding

While most tree based models do not require encoding of categorical variables into numerical ones, XGBoost does. For this reason we need to choose a proper way of doing so. Grouping categorical predictors into independent categories was summarily described in Section 2.7.1. When performing linear regression we would have to choose dummy variable encoding. The reason being that one-hot encoding would introduce multicollinearity. This follows from the fact that one of the introduced binary variables would be a linear combination of the others. This is not an issue with tree based models however. Alas dummy variable encoding do have the advantage of saving some memory and computational effort, which is why it is chosen in this thesis.

Also mentioned in Section 2.7.1 were the problems associated with categorical variables with high cardinality, we will see that this is an issue with the Booli data set in the following paragraph.

3.2.2.2 Feature Engineering

The practice of transforming instance data in order to improve prediction performance in a statistical learning context is often referred to as *feature engineering*. Unfortunately feature engineering is quite the informal topic with loose recommendations, often citing domain knowledge and the specificities

of the data and method. There is however a consensus about the importance of feature engineering for improving prediction performance. One common, albeit informal, advice is to create many engineered variables initially and worry about variable selection later. This is a recommendation that will be abided.

Applying XGBoost to the booli data set poses some specific challenges. These challenges will each each be discussed and suggestions to alleviate them proposed.

(a) **Distribution of the Response**
As we saw in Section 3.1.1 the empirical distribution of sold price is positively skewed. Among other things this means that the response contains values of large magnitude in comparison to the bulk of observations. In section 2.5.3 we determined that this could affect prediction performance and that a good course of action was to transform the response variable. Several transformations seem plausible, among them the logarithmic transform and sold price per square meter of living area.

(b) **Temporal**
There are two temporal predictors in the Booli dataset. Sold date, the date the apartment was sold and published date, the date the advertisement for the apartment was published. Variables of datatype date poses a challenge as they are represented as running numbers formatted to represent years, months and days. The algorithm, XGBoost, however won't be aware of the implicit information. As a consequence that information needs to be conveyed explicitly, through engineered predictors representing, for example, month and year.

(c) **Geospatial**
Since tree based models does not have an inherent way of interpreting geographic coordinates as location, or the difference between them as distance, some proxy variable is desirable to enhance performance. Such a feature is commonly zip codes. Zip codes are however a classic example of a high cardinality categorical variable. Besides the challenge discussed in Section 2.7.1 it is rarely computationally desirable to process thousands of predictor variables, as would often be the case when dummy encoding nationwide zip codes. A common solution to this problem is to engineer a feature that captures the significance of the geographical area. In the particular case of the Booli dataset one measure of importance might be the price per square meter of living area, at a certain period of time.

3.2.3 Partitioning

As we saw in Section 2.2.2 cross-validation estimates the expected test error $E[L(Y, \hat{f}(X)]$. This estimate is helpful for comparing models and tuning meta-parameters, it is not however an appropriate estimate of the true prediction error $Err_{\mathcal{T}}$. Too few folds might impact the size of the training instance in a way that leaves the model biased, overestimating the prediction error. Too many folds might instead lead to large variance and underestimation of the prediction error.

The solution is quite simple however. Since data is abundant we simply partition the full dataset into a training set and a test set. We split the data randomly into the training set, consisting of 85% of the original observations, and the test set, containing the remaining 15%.

Besides these data sets a completely unseen set containing future observations will be downloaded when the model has been finalized.

4 Modeling

Using XGBoost to create a predictive model is a proposition that differs substantially from more traditional methods. The key challenge using XGBoost is tuning the meta parameters (these include the regularization parameters) of the algorithm. This isn't to say that variable selection and other practices aren't of importance, but less so.

The aim of this chapter is to outline the process of tuning and diagnosing an XGBoost model.

4.1 XGBoost implementation

The theoretical framework of XGBoost was discussed in Chapter 2. For the practical implementation the open source package created by Chen & Guestrin (2017) [5] is used. The distributed version is built on top of rabit making it available to a number of platforms and languages. For this thesis the R implementation of XGBoost is used.

4.2 Parameter Tuning

Academic literature is sparse on the subject of configuring XGBoost parameters, though some studies have been made Probst et al (2018) [28]. Luckily the competitive machine learning scene provides some hueristics. These usually involve optimizing parameters with techniques like grid search, random search Bergstra & Bengio (2012) [2] or occasionally Bayesian optimization Snoek et al (2012) [33]. The approach in this thesis will be grid search as it is the most common one. No matter the approach though, the same issue is

faced, the parameter space. A conservative (some exotic XGBoost parameters were disregarded in this thesis) count of tuning parameters in XGBoost is nine. Using all nine parameters with, for example, six parameter values each would result in over ten million combinations! The conclusion is that an exhaustive approach isn't viable.

In order to tune the meta parameters we will use hueristics gathered from the likes of Hastie et al (2009) [14], Thakur (2016) [34] and Zhang (2015) [36].

1. **Default model**
   Define some set of default parameters using dataset and domain knowledge. Compromise between accuracy and speed when determining the learning rate η. A lower value of η often yields better tuning performance but increases the number of optimal iterations, increasing the computational demand for tuning. An initial range of iterations is often suggested to be a few hundred trees. Given a fixed value of η determine the optimal number of boosting iterations using 5 or 10-fold cross-validation.

2. **Grid search**
   The second step of the tuning process utilizes grid search. Grid search is simply the XGBoost evaluation of the ordered n-tuples resulting from the n-ary Cartesian product over the sets $X_1, \ldots, X_n$,

   $$X_1 \times \cdots \times X_n = \{(x_1, \ldots, x_n) | x_i \in X,\ \forall i \in \{1, \ldots, n\}\}, \tag{39}$$

   each X_i $i = 1, \ldots, n$, containing values for a hyper parameter. The idea is to first optimize the randomization and tree shape parameters and then determine the regularization parameters. Table 3 summarizes common hueristcs for parameter ranges.

Table 3: Hyper Parameter Tuning

Hyper parameter	Tuning approach	Range
No. of Trees, M	Initial tune $\Rightarrow$ Fine tune	$\{100, \ldots, 1000\}$
Learning rate, η	Initial tune $\Rightarrow$ Fine tune	$(2, 10)/M$
Max Tree depth T_{max}	Grid search	$\{3, 5, 7, 9, 12\}$
Row subsampling, ζ_r	Grid search	$\{.5, .75, 1\}$
Col. Subsampling, ζ_{c_t}	Grid search	$\{4, 6, 8, 1\}$
Gamma, Γ	Fixed	0
$l1$ regularization, α	Grid search	$\{0, .1, .5, 1\}$
$l2$ regularization, λ	Grid search	$\{(.01, .1), 1\}$
Min. leaf Weight, h_{min}	Fixed $\Rightarrow$ Fine tune	$\{1, 3, 5, 7\}$

3. **Fine tune**
   Using the best grid search parameters, the number of trees M and

the learning rate η is tuned once again. This time however the goal is accuracy, suggesting lower values of η and likely more iterations M. Finally the minimum leaf weight h_{min} is fine tuned.

It should be emphasized that the approach described above by no means is guaranteed to find optimal meta parameters. It has however proven successful in many machine learning competitions, Zhang (2015) [36].

4.3 Feature selection

Feature selection is often an integral part of optimizing performance of traditional learning techniques, as in the case of linear regression for example. We shall see however that this is of less importance when using XGBoost.

Minimizing (17) entails finding the optimal (in a greedy sense) predictor variable by which to partition the prediction space. This means that feature selection is done implicitly by the tree boosting algorithm. The same is true for XGBoost in particular. Reducing the number of input variables are actually often detrimental to model performance as correlated variables are used when dealing with missing values, as discussed in Section 2.7.2.

4.3.1 Other Consideration

Even though the XGBoost algorithm handles feature selection gracefully there are some remaining practical considerations regarding the Booli dataset.

(a) **Perfect Correlation**
The booli dataset contains some perfectly correlated variables, for example source.name and source.id. One of these can safely be omitted as no information is lost.

(b) **Noisy Features**
Some of the Booli dataset input variables are just random noise. Among these are internal observation IDs and URLs. These will be excluded as well.

(c) **List Price**
Relevant to the aim of this thesis is the feature List Price. List price is an estimate of sold price done by a trained real estate agent. Using this variable as a predictor would in a sense be "cheating" as the aim of this thesis is to create a data driven model. For this reason the list price variable will be dropped.

5 Results

In this chapter the model based on the XGBoost algorithm, transformed Booli data, and subsequent parameter tuning will be presented. The result-

ing mathematical model is formally defined by the parameters of the tree space Θ. However, a more practical representation is achieved by characterizing the model by its associated meta parameters and predictive performance.

5.1 Final model

In Section 3.2.2.2 feature engineering was proposed as way to resolve the issue of the skewed distribution of the response variable, sold price. This resulted in three possible approaches for creating an XGBoost model: using an unaltered version of sold price, the log of sold price and sold price per square meter of living area. A model was trained and tuned for each transformation, using the Booli data training set and the tuning scheme suggested in Section 4.2.

Table 4: The values in the table are the RMSE of the price per m^2 for each models prediction vs. actual values, evaluated on the test set.

Model	RMSE
Sold Price	4494
log(Sold Price)	4341
Sold Price per m^2	2742

Evaluation of the respective model was done with respect to the predictive performance, measured by RMSE for sold price per m^2, on the Booli data test set. The model trained on sold price per square meter of living area had the best predictive capabilities when applied to the test set and was the choice of final model. The performance of the different models are summarized in Table 4.

5.1.1 Parameters

The parameters of the final model, the result of the tuning procedure described in Section 4.2, are presented in Table 5. The regularization parameter Column Subsampling by level, ζ_{c_l} was not tuned due to suggested hueristics. ζ_{c_l} has a default value of one.

5.2 Prediction

The predictive performance of the final XGBoost model when applied to the test set and a set of future observations will be presented in this section. The prediction measure is sold price per m^2, in contrast to the original response sold price. Sold price per m^2 is chosen as metric as it is easier to interpret. The diagnostic metric for the predictive power of the model is the RMSE across all test observations.

Table 5: Final Parameters

Parameter	Value
No. of Trees, M	10,000
Learning rate, η	0.01
Max Tree depth T_{max}	5
Row subsampling, ζ_r	1
Col. Subsampling, ζ_{c_t}	1
Gamma, Γ	0
$l1$ regularization, α	0
$l2$ regularization, λ	0
Min. leaf Weight, h_{min}	1

5.2.1 Test Set

The Booli data test set contains 59370 observations randomly selected from the full booli dataset. Figure 10 depicts the relation between predicted and actual prices. The calculated RMSE of the test set is 2742, (very) roughly 2000 SEK less than the competing models.

5.2.2 Future Observations

The dataset of future observations was downloaded once the final model had been determined. The dataset contains the same variables as the original Booli dataset but spans a different interval of time, namely 2018-09-11 to

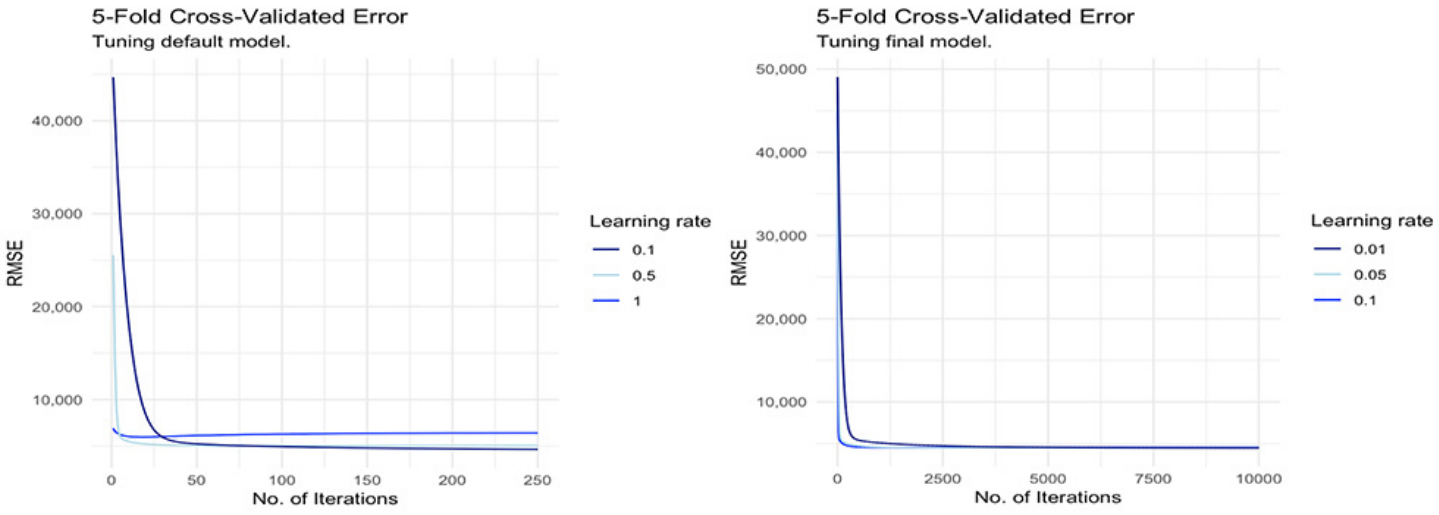

Figure 9: Cross-validated test error for different learning rates when tuning the (left) default model and (right) final model respectively.

2018-09-25. The observations correspond to the apartments published for sale in Sweden during the two weeks following the date the last apartment was sold in the training set.

Using the observations of the future dataset mimics the intended usage of the model more closely than the test set. This stands to reason as valuation in most cases are done in advance of a sale, rather than afterwards. The RMSE with regard to the future set is 4972. Performance will be further discussed in Section 6.2.

6 Discussion

The aim of this chapter is to examine the results presented in the previous sections with regard to underlying theory, data and modeling approach. Furthermore we will explore the model with the intent of unveiling its predictive behavior. Finally, drawing from insights gained during work with the thesis, some particular fields of interest are suggested for future research.

6.1 Fitting

With grid-search and cross-validation being computationally demanding it is important to strike a balance between test error and the number of boosting iterations when choosing a default model.

The left panel of figure 9 depicts the cross-validated test error of the default model (using sold price per m^2 as response) for three choices of η. It is clear that neither choice of learning rate results in excessive overfitting as all three curves level off as the number of iterations increases. This opposes the expectation of a u-shaped test-error curve discussed in Section 2.2.1, such results however, are not uncommon in practical application using large datasets and shrinkage, Hastie (2009, p. 371) [14]. The value of 0.1 is chosen as learning rate for the default model as it yields the lowest cross-validated test error.

Since overfitting does not appear to be an issue the use of regularization and randomization meta-parameters might not necessarily improve the model. This turned out to be the case with the Booli dataset and the final model; only Max Tree depth, T_{max}, learning rate, η and Number of Trees, M was changed in comparison to the default model. The right panel of Figure 9 shows the cross-validated test-error when tuning the final model. As suggested by Friedman (2001) [12] smaller values of η seem to result in better test error, however requiring a larger number of boosting iterations, M. Capping the number of iterations to 10,000, due to computational restraints, lead to the final choice of 10,000 iterations and a learning rate of 0.01. Test-performance might have continued to improve with a larger number of trees but such improvement where deemed too marginal in contrast

to computational demand. The reduction of cross-validated RMSE between the default model and final model was 4%.

All engineered features were kept in the final model as they (marginally) improved the cross-validated predictive performance of the model. This might obscure the interpretation of the model as many the engineered features are correlated and might interact. Sacrificing interpretability for predictive power is not ubiquitous, it is however in line with the purpose if this thesis: creating a useful predictive model.

6.2 Performance

The aim of this thesis is to create a useful model with satisfactory predictive performance. This is an admittedly vague criterion and as such will be further discussed in this section.

Figure 10 depicts the final model predictions versus actual prices, based on the test set. Observations being clustered closely around the diagonal suggests that the model is doing a good job of predicting sales prices. The implication of *good* is critical however. This is no trivial task and will often depend on circumstance and ambition. In Chapter 1 the notion of fair market value was discussed and it was concluded that it was part of the real estate agents responsibility to perform such an appraisal. The Booli dataset contains information on the professional opinion regarding an apartments fair market value at a given time, the list price.

Comparing the models predictive performance to the accuracy of realtor valuation will provide a notion of good that resonates with the purpose of this thesis. Table 6 contains the RMSE of the model prediction and realtor estimation respectively. Both measures have been computed for the test set and future set. The model outperforms the professional appraisals under the assumption that list prices correspond to estimates of fair market values.

6.3 Interpretation

As we saw in section 2.4 one of the advantages of single decision trees are their inherent interpretability. This quality is lost however, when using ensembles of trees. This leads to the predicament of having to choose between

Table 6: The values in the table are the RMSE of the price per m^2 for the model prediction vs. actual values and the list price vs. actual values. Both measures have been calculated for the test set and future set respectively.

Dataset	Model Prediction	List price
Test set	2742	6587
Future set	4972	5643

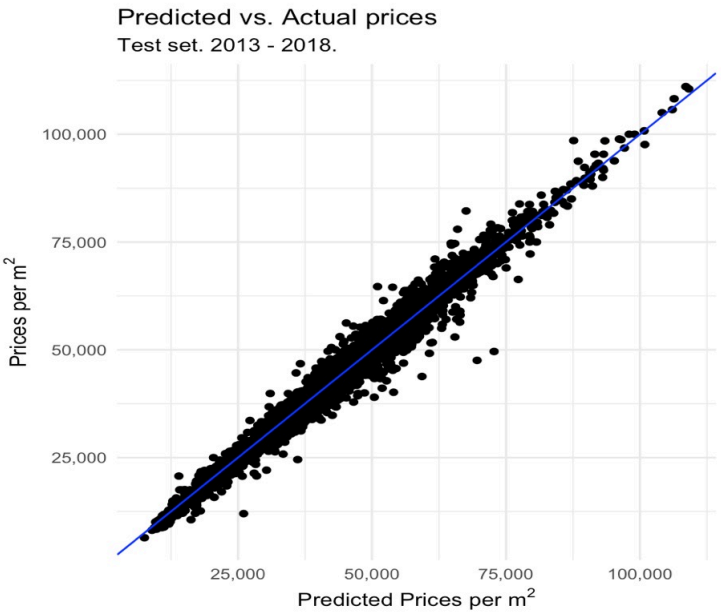

Figure 10: The figure shows predicted versus actual apartment prices per m^2 for the test set. Observations closer to the diagonal blue line are more accurate.

the superior predictive performance of ensembles and transparency of simpler models. The situation is further complicated by the fact that common approaches to model explanation is inconsistent as shown by Lundberg et al. (2017) [20] and Fisher et al. (2018) [11].

6.3.1 Global Attribution

Attribution is the practice of assigning importance measures to the predictor variables of a model. More specifically global attribution assigns features a measure representing the global impact of predictors on the model.

Common measures of global feature importance when using XGBoost are:

1. **Weight**
   The number of times a feature is used to split the data across all tree iterations.

2. **Cover**
   The number of times a feature is used to partition the data across all trees weighted by the number of training data points that goes through those splits.

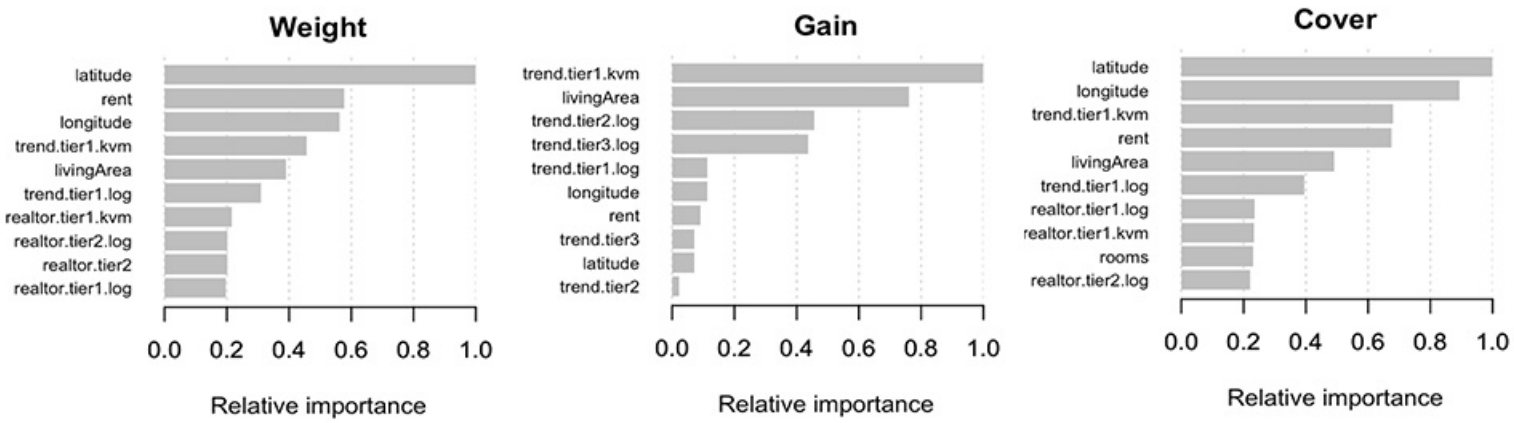

Figure 11: Feature Importance displayed by relative importance. Each panel shows a different measure of feature importance: (Left) weight, (Center) gain and (Right) cover.

3. **Gain**
   The average training loss reduction gained when using a feature for splitting. The gain for a given split is defined in Equation (38).

Figure 11 depicts the 10 most important features by each of the above measures. As we can see the feature importance depends on which measure is used. This is troubling as there are many situations where the prediction an algorithm makes needs to explained. There are several reasons the feature importance metrics are unreliable. In the case of correlated variables in boosting for example, when a specific link between a predictor and outcome has been established the algorithm will only place importance on one of the correlated features, XGBoost documentation (2016) [37]. Another example of an unreliable metric is the Gain. Splits taking place near the root of three are typically more important than splits taking place near the leaves. This results in potential bias as the Gain method attributes more importance to lower splits, see Lundberg & Lee (2018) [20] for details.

These results might seem discouraging but the SHAP method discussed in Section 2.3 promises consistent results. Figure 12 displays global and local SHAP measures. We note that the predictor *trend.tier1.kvm* (tt1.kvm) is determined to have the highest global impact on prediction magnitude. This is expected as tt1.kvm was constructed to represent the current local price point of apartments. The second most important predictor by SHAP is living area. We recall that the final model predicts sold price per m^2 of living area. We note that the first and second most important feature by SHAP coincides with the Gain measure in Figure 11, the order of importance

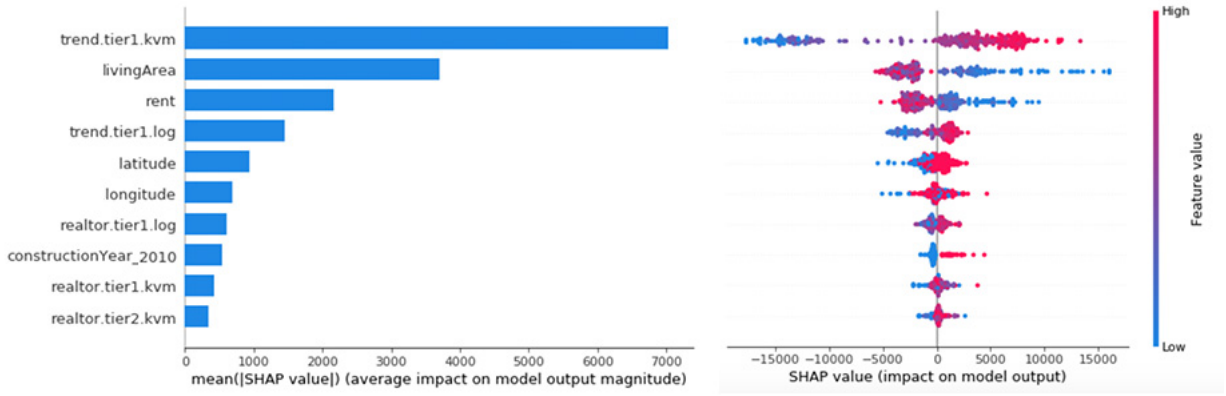

Figure 12: Feature importance displayed by SHAP Values. (Left) Average impact on model output magnitude. (Right) Local attribution by SHAP for a random subset of training observations. Red points denote larger predictor values, blue indicates smaller. (Right) Location along the x-axis implies the impact of given feature on the predicted value for a certain observation.

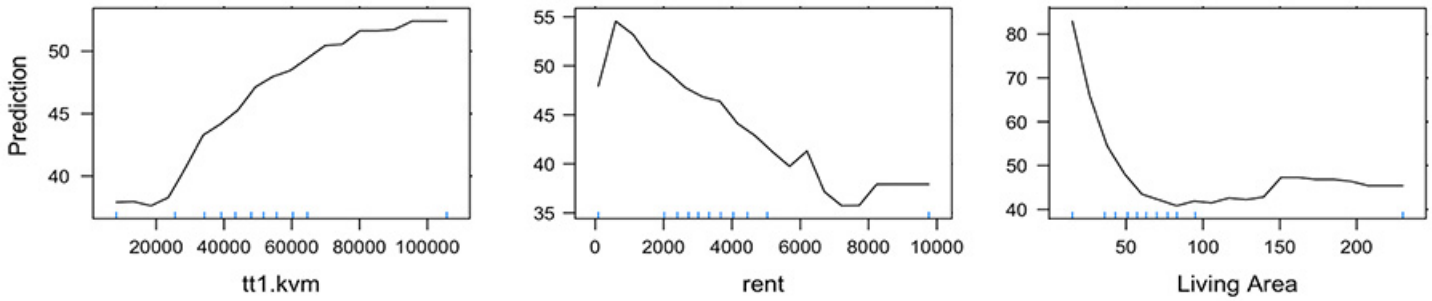

Figure 13: Partial dependence of sold price per m^2 on the 3 predictors with largest feature importance. The blue hash marks at the base of the plot are deciles of the input variables. Prediction values (y-axis) in 1000 SEK.

do however differ further down the importance ranking.

With a firm idea of the most important predictors a reasonable question is in which way those features affect the prediction. Figure 13 displays partial dependence plots of the three most important predictors measured by SHAP. The ticks at the bottom of each plot indicates the deciles of the input data. Larger separation of the ticks mean lower instance data density which in turn implies less certainty about the curve shape. The partial dependence curve represent the relationship between a feature and the response accounting for the average effect of the other predictors, Friedman (2001) [12]. The partial dependence of tt1.kvm on sold price (left panel) is essentially monotonic increasing. It should be noted however that the

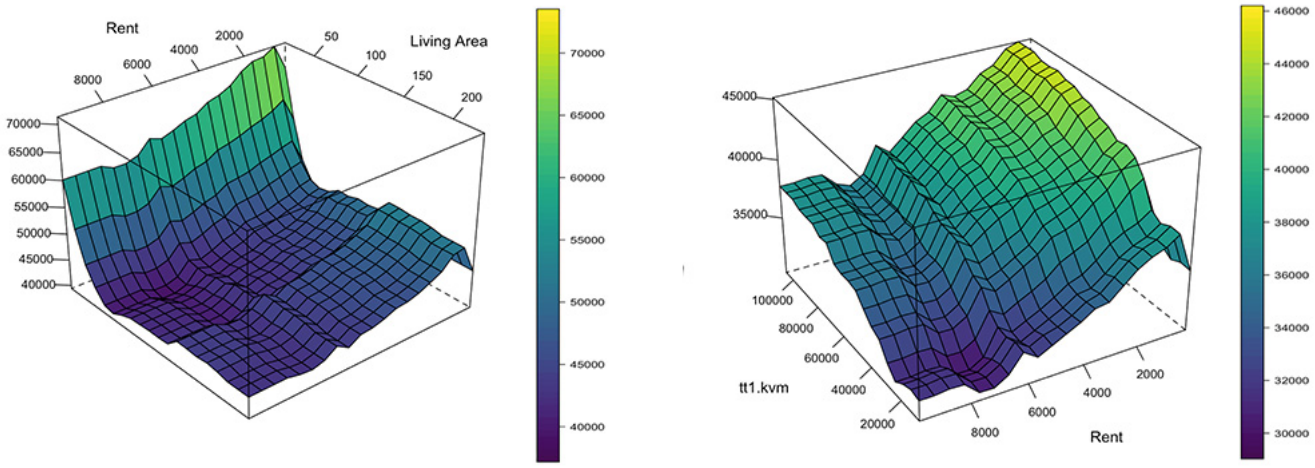

Figure 14: Two dimensional Partial dependence of sold price on (left) living area and rent and (right) tt1.kvm and rent.

y-axis is cropped and influence of tt1.kvm on sold price is less pronounced than what it seems at a glance. This might appear suspect as it scored high in all feature importance tests but it suggests that the weak main effect is masked by strong interaction effects with other predictors. The partial dependence curve of living area (right panel) shows a steep drop indicating that less living area give rise to higher prices per m^2. The effect of rent on price per m^2 (center panel) shows an irregular pattern, the relationship is however essentially linear decreasing in the more well defined second to nineth decile.

Beyond the individual importance of features on model output it is of interest to understand the interactions between predictors. Figure 14 shows some of these interactions. Exploring the two-way partial dependence of tt1.kvm and rent on price per m^2 we see a strong interaction effect on the model output. Decreasing rent and increasing tt1.kvm results in higher prices. The interactions between rent and living area produces some interesting results for small apartments with low rent. There seem to be an apparent premium on apartment prices associated with low values of size and rent.

6.3.2 Local Attribution

In contrast to global attribution local attribution represents the individualized impact of features for a single prediction. Figure 15 shows the individual SHAP-contribution of features towards the prediction value for a random test observation. The dominant feature appears to be living area which is balanced by the local price point. The right panel of Figure 12 shows local

attribution from a global perspective, each point represent an observation's feature value by color and that predictors individual contribution to the model output be position along the x-axis. We note that this representation appears consistent with Figure 13.

6.4 Future Work

While work with this thesis produced a usable model several shortcomings and questions have arisen during the process. Some of these warrant further research.

A field that would clearly benefit from further research is the effect of feature engineering. There's a consensus about it is importance yet the theoretical justifications are few. The same can be said to for hyper parameter tuning XGBoost, while novel approaches and empirical results are available the expected effects of parameter tuning lacks profound theoretical grounding, at least from the exposure gained while researching this thesis.

Suggestions for research questions could be; how are temporal and geospatial dependencies best modeled when using tree-based methods? Regarding the lack of academic literature on parameter tuning a question posed could be; which dataset characteristics lends themselves to tuning, and why? Another would be if it is possible to determine bounds for the effect of hyper parameter tuning?

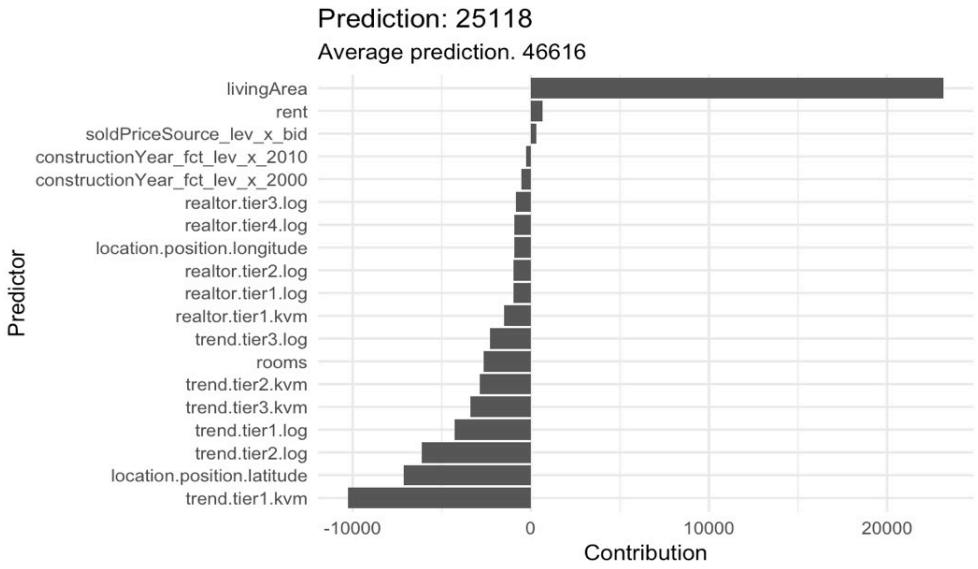

Figure 15: Contribution to prediction value per feature for a single observation. Features with small or zero contribution have been filtered out.

7 Conclusion

The fair market value is the estimate of the market value of a property given that both parties are knowledgeable, willing, and unpressured. The aim of this thesis was to produce a model capable of outputting such a quantity. Aiding the endeavor was XGBoost and the Booli dataset. Showing how XGBoost approximates gradient descent while using the mean squared loss function provided the foundation for the practical work of creating the predictive model. Adjusting meta parameters and creating bespoke predictors helped improve the model which proved to produce useful estimates.

It has been shown that it is possible to create a predictive model using XGBoost that outperforms professional appraisers and thus can be considered useful. While this only holds under the assumption that list prices reflects the realtors actual estimation of market values the result is encouraging non the less.

While complicated by interactions and disparity between metrics the three predictors with the largest influence on the fair market value was determined to be the local price point, reflecting the price per m^2 at a given point in time, the living area and rent.

While accuracy and inference of important features might suggest a relevant model a lot of work would remain before real world implementation including a thorough investigation of algorithmic bias.

Finally the work on this thesis have revealed some gaps in the academic literature regarding feature engineering and parameter tuning. Pursuing those topics in the future would surely be a rewarding endeavor.

References

[1] Belson, W. (1959) 'Matching and Prediction on the Principle of Biological Classification'. *JRSS, Series C, Applied Statistics*, Vol. 8, No. 2, June, 1959, pp. 65-75

[2] Bergstra, J. and Bengio Y.(2012) 'Random search for hyper-parameter optimization'. *J. Mach. Learn.* Res. 13, pp. 281-305

[3] Breiman, L. (1987) *Classification and Regression Trees.* New York, NY, USA: Routledge.

[4] Breiman, L. (1997) 'Arcing The Edge'. Technical Report 486 , Statistics Department University of California, Berkeley CA. 94720, [online]. Available at: `https://pdfs.semanticscholar.org/65b7/b1a0d61fd012f10cfce642d4aa4dec9a5829.pdf` (Accessed: 16 December 2018)

[5] Chen, T. and Guestrin, C. (2016) 'XGBoost: A Scalable Tree Boosting System'. The 22nd ACM SIGKDD International Conference, [online]. Available at: `https://arxiv.org/pdf/1603.02754.pdf` (Accessed: 16 December 2018)

[6] Dahlén, C., Lundblad, P., Stattin, C. and Clemedtson, P. (1995) *Företagsvärdering.* Stockholm, Sverige: Öhrlings Coopers och Lybrand.

[7] Dahlquist, G. and Björk, Å. (2012) *Numerical Methods*, New York, NY, USA: Dover publications.

[8] Danks, D. and London, A. (2017) 'Algorithmic Bias in Autonomous Systems'. *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence* pp. 4691-4697.

[9] Doyle, P. (1973) 'The use of automatic interaction detector and similar search procedures'. *Oper. Res. Quart.*, 24, pp. 465–467.

[10] Einhorn, H.J. (1972). 'Alchemy in the behavioural sciences'. *Public Opin. Quart.*, 36, pp. 367–378.

[11] Fisher, A., Rudin, C. and Dominici, F. (2018) 'Model Class Reliance: Variable Importance Measures for any Machine Learning Model Class, from the "Rashomon" Perspective', [online]. Available at: `https://arxiv.org/pdf/1801.01489v1.pdf` (Accessed: 16 December 2018)

[12] Friedman, J. H. (2001) 'Greedy Function Approximation: A Gradient Boosting Machine'. *Ann. Statist.* Volume 29, Number 5, pp. 1189-1232.

[13] Freund, Y. and Schapire, R.E. (1997) 'A decision-theoretic generalization of on-line learning and an application to boosting'. *Journal of Computer and System Sciences*, 55, pp. 119–139.

[14] Hastie, T., Tibshirani, R. and Friedman, J. (2009) *The Elements of Statistical Learning.* 2nd edn. New York, NY, USA: Springer Series in Statistics Springer New York Inc.

[15] Heaton, J. (2016) 'An Empirical Analysis of Feature Engineering for Predictive Modeling'. SoutheastCon, 2016. [online]. Available at: `https://arxiv.org/pdf/1701.07852.pdf` (Accessed: 16 December 2018)

[16] Kearns, M. and Valiant, L. (1989) 'Cryptographic limitations on learning Boolean formulae and finite automata'. *Journal of the ACM*, Volume 41 Issue 1, pp. 67-95.

[17] Kotsiantis, S.B., Kanellopoulos, D. and Pintelas, P.E. (2006) 'Data Preprocessing for Supervised Leaning'. *International journal of computer science*. Volume 1, number 1, pp. 1306-4428.

[18] Little, R. and Rubin, D. (2002) *Statistical Analysis with Missing Data*, 2nd edn. New York, NY, USA: Wiley

[19] Loh, W. (2014) 'Fifty Years of Classification and Regression Trees'. *International Statistical Review*, 82, 3, pp. 329–348

[20] Lundberg S.M., Erion G.G. and Lee, S.I. (2017) 'Consistent Individualized Feature Attribution for Tree Ensembles'. [online] Available at: `https://arxiv.org/pdf/1802.03888.pdf` [Accessed 9 December 2018].

[21] Lundberg, S.M. and Lee, S.I. (2017) 'A Unified Approach to Interpreting Model Predictions'. *Advances in Neural Information Processing Systems 30*, NIPS 2017

[22] Mason, L., Baxter, J., Bartlett, P. L. and Frean, M. (1999) 'Boosting Algorithms as Gradient Descent'. *Advances in Neural Information Processing Systems*. 12, MIT Press. pp. 512–518.

[23] Messenger, R. and Mandell, L. (1972). 'A modal search technique for predictive nominal scale multivariate analysis'. *J. Amer. Statist. Assoc.*, 67, pp. 768–772.

[24] Morgan, J.N. and Sonquist, J.A. (1963) 'Problems in the analysis of survey data, and a proposal'. *J. Amer. Statist. Assoc.*, 58, pp. 415–434.

[25] Muellbauer, J. and Murphy, A. (2012) 'Booms and busts in the UK housing market'. *The Economic Journal*, Volume 107, Issue 445.

[26] Mäklarsamfundet. (2017) 'Regler för prissättning och budgivning', [online]. Available at: http://www.maklarsamfundet.se/fragor-och-svar/regler-prissattning-och-budgivning. (Accessed 3 December 2018)

[27] Persson, E. (2005). 'Vad är fastigheten värd?' *Fastighetsekonomisk analys och fastighetsrätt, fastighetsnomenklatur*. Stockholm: Fastighetsnytt Förlags AB. pp. 343-406

[28] Probst, P., Bischl, B. and Boulesteix, A.L. (2018) 'Tunability: Importance of hyper- parameters of machine learning algorithms', [online]. Available at: https://arxiv.org/pdf/1802.09596.pdf (Accessed 16 December 2018)

[29] Ridgeway, G. (2006) 'gbm: Generalized Boosted Regression Models. R package version 1.5-7', [online]. Available at: http://www.i-pensieri.com/gregr/gbm.shtml (Accessed 3 December 2018)

[30] Ritschard, G. (2013). 'CHAID and Earlier Supervised Tree Methods'. *Contemporary Issues in Exploratory Data Mining in Behavioral Sciences*, Routeledge, New York, pp. 48–74.

[31] Schapire, R.E. (1990) 'The Strength of Weak Learnability'. *Machine Learning*, 5, pp. 197–227.

[32] Seaver, N. (2013) 'Knowing Algorithms'. *Media in Transition*, 8. Cambridge, MA.

[33] Snoek, J., Larochelle, H. and Adams, R.P. (2012) 'Practical Bayesian optimization of machine learning algorithms'. *NIPS'12 Proceedings of the 25th International Conference on Neural Information Processing Systems, Volume 2*, pp. 2951-2959.

[34] Thakur, A. (2016) 'Approaching (Almost) Any Machine Learning Problem', [online] Available at: https://www.linkedin.com/pulse/approaching-almost-any-machine-learning-problem-abhishek-thakur [Accessed 9 December 2018].

[35] Tukey, J.W. (1962) 'The Future of Data Analysis'. *Ann. Math. Statist.* 33, no. 1, pp. 1-67.

[36] Zhang, O. (2015) 'Winning data science competitions', [online] Available at: https://www.slideshare.net/ShangxuanZhang/winning-data-science-competitions-presented-by-owen-zhang. [Accessed 9 December 2018].

[37] XGBoost Documentation (2016) 'Understand your dataset with XGBoost'. [online] Available at: https://xgboost.readthedocs.io/en/latest/R-package/discoverYourData.html [Accessed 9 December 2018]