Interpretable House Price Prediction Using a Collection of Local Machine Learning Models

Anders Hjort*[1], Johan Pensar[2], Ida Scheel[3], and Dag Einar Sommervoll[4]

[1]*Department of Mathematics, University of Oslo, Norway and Eiendomsverdi AS*
[2]*Department of Mathematics, University of Oslo, Norway*
[3]*Department of Mathematics, University of Oslo, Norway*
[4]*School of Economics and Business, Norwegian University of Life Sciences and NTNU Trondheim Business School, Norway and Eiendomsverdi AS*

June 2022

Abstract

House price prediction models are used to estimate the price of a dwelling given its features such as location, size or number of bedrooms. Machine learning models like gradient boosted trees are the state-of-the-art for this in terms of prediction accuracy, but these models are often hard to interpret: Why exactly does the model predict what it predicts? We compare the performance of gradient boosted trees with the performance of Generalized Additive Models (GAMs), a class of model that is widely regarded as interpretable. We argue that GAMs are not optimal for data sets with categorical variables with a large number of levels, as is the case with the $K = 96$ city districts in the data set we study. As a possible solution for this, we propose using a collection of local models on clusters of city districts, where districts within the same cluster have a similar prediction model.

We evaluate the models on a data set consisting of $N = 29\,933$ transactions from Oslo, Norway, in 2018-2019. The results indicate that the gradient boosted trees have the best model performance, achieving an RMSE of 10.1% compared with 15.5% for the GAM model. The local GAM models achieve 11.5% at its best, while simultaneously remaining fully interpretable.

1 Introduction

Automated Valuation Models (AVMs) are statistical models that aim to estimate the value of a dwelling, or oftentimes a portfolio of dwellings. There might be many reasons why this is of interest, but one of the main reasons is risk management for banks that use dwellings as mortgage collateral when giving out mortgages. The financial crisis of 2007-2008 highlighted the need for having in place reliable and efficient mechanisms for monitoring of housing market valuations.

The idea of using statistical tools for house price prediction is not novel, and some early examples include the use of repeated sales models (Bailey et al. (1963)) and hedonic regression models (Rosen (1974)). Recent years have seen an increased use of more sophisticated machine learning models as AVMs. The term machine learning refers to a broad set of techniques and tools where models learn patterns from data, induce general learning rules from previous examples and use these rules to make predictions on new instances. Examples of popular machine learning techniques are decision trees, random forests (Breiman (2001)), gradient boosted trees (Freund et al. (1996), Friedman et al. (2000)) and neural networks (Schmidhuber (2015)). Many machine learning models have proven to yield high accuracy for the task of house price prediction: Ho et al. (2020), Park et al. (2015), Baldominos et al. (2018), Sing et al. (2021), Kim et al. (2021) and Hjort et al. (2022) all demonstrate the effectiveness of tree-based models such as random forest and gradient boosted trees for house price prediction.

*Corresponding author: anderdh@math.uio.no.

While these machine learning models often tend to give more accurate predictions than traditional regression models, the accuracy often comes at the cost of interpretability. Many of the above mentioned models are frequently referred to as *black box* models. This term captures the fact that the models often are difficult to interpret. To combat this, several explanatory frameworks exists, such as the SHAP (SHapley Additive exPlanations) framework proposed by Lundberg et al. (2017), and the LIME (Local Interpretable Model-Agnostic Explanations) framework introduced in Ribeiro et al. (2016). These methodologies consists of training a second explanation model on top of the prediction model, aiming to provide an explanation on why the model estimate is what it is.

Although explanatory tools like SHAP and LIME has gained popularity in recent years, they have also received critique. Rudin (2019) argues that all explanatory frameworks that creates *post hoc* explanations must be wrong by construction: The only way the explanatory model can give an exact explanation is if the explanatory model equals the prediction model, which would imply that the prediction model was fully interpretable in the first place.

A fundamental challenge in this regard is that there is no simple and universal definition of what it means for a model to be interpretable, and the requirements will vary with different applications and contexts. Whenever a prediction model outputs a prediction $\hat{y}_i$ given a set of input features $\mathbf{x}_i$, an interpretable model should be able to offer an insight into why $\hat{y}_i$ is the prediction returned. Many classical methods such as linear regression are inherently interpretable: The prediction is nothing but a sum of β coefficients times covariate values x_i, and it is therefore straightforward to explain how much each covariate x_i contributes to the final prediction. Some authors like Lou et al. (2012) and Lou et al. (2013) connect the notion of interpretability closely to visualization: If the model predictions can be visualized in a simple and user-friendly manner, the model is regarded as interpretable.

A class of models that is widely regarded as interpretable is the Generalized Additive Models (GAM). A GAM is a function that can be expressed as

$$f(x_1, x_2, ..., x_p) = f_0 + f_1(x_1) + f_2(x_2) + ... + f_p(x_p),$$

that is, as a sum of separate models of each covariate x_j. The models $f_j(x_j)$ are often called *shape functions*. As this class of models has no interaction terms, they can easily be examined visually: By plotting the pair $(x_j, f(x_j))$ we can examine exactly how a change in the covariate value x_j would impact the corresponding shape function. Furthermore, if the model outputs a certain prediction $\hat{y}$ we can decompose it to see how much each of the shape functions contribute to the final estimate.

A slight challenge with the GAMs arise when one of the covariates is a categorical covariate with K disjoint levels that have no intuitive order or ranking. This is highly relevant in the realm of house price data, since categorical covariates such as city district, municipality or postal code are typical and often very important determinants of the final sale price. One way of handling this is to introduce dummy variables for each of the K city districts such that each dummy variable is either zero or one. This is in many cases the intuitive approach, and it is also what many procedures like linear regression does "under the hood" when treating categorical variables. This solution requires little tuning or engineering and is easy to implement. The downside is that it effectively assumes the same form on the shape function for each of the K city districts.

If the between-group differences are sufficiently large, another approach is to create completely local models, that is, a separate prediction model per city district. The upside of this is that each model will be more refined as it is trained only on data from one district: We essentially get an ensemble of K local experts. This can be very good when there is enough available data in each city district, but it runs into trouble when some districts have few observations. If the number of city districts become large it can also becomes computationally and practically difficult to manage K completely separate models.

A trade-off between creating K completely local models and one global model, is to create a smaller number of models where each model is trained on a cluster of similar city districts. This process of collapsing similar groups together is referred to as *feature fusion* by Gertheiss et al. (2010). The motivation behind using feature fusion is that some city districts might display similar characteristics, and it will therefore make sense to group these together and in turn train a separate model on this subset of the data. Clustering together similar city districts have many advantages: Sharing of data between similar groups might lead to better models than if we train K completely separate models. Since the underlying premise is that the groups that are clustered together share many of the same attributes, each model might be less complex

than when we try to create a single global model. An obvious challenge with the categorical feature fusion is that the space of possible models quickly grows to be too large to exhaustively test every combination. This begs the question: How can we effectively fuse together groups in a way that creates the best set of $L < K$ local prediction models?

This problem can be tackled in many ways. Gertheiss et al. (2010) introduced a grouped lasso approach with regularization on the number of groups, effectively clustering similar groups together. More generally, Späth (1979) developed the framework of clusterwise linear regression (CLR), where the data set is partitioned into clusters and separate models are trained for each cluster. Zhang (2003) has introduced the same concept under the name regression clustering, an approach that Brusco et al. (2008) warned that might lead to overfitting of the training data. In the realm of neural networks, the notion of Mixture of experts (MoE) was introduced by Nowlan et al. (1990). This methodology consists of training multiple models to be experts in different parts of the feature space. On top of this, a *gating network* is trained to decide which of the underlying experts that should handle a new instance. Zeileis et al. (2008) presented model-based recursive partitioning, which is a framework for building separate models for different parts of the feature space. Among the p covariates, some are chosen as partition variables and some are chosen as regression variables.

This paper studies a data set consisting of $N = 29\,933$ housing market transactions from the Oslo (Norway) in 2018-2019. Each dwelling in the data set belongs to one of the $K = 96$ city districts. We investigate two closely related questions in this paper:

- Will interpretable models such as Generalized Additive Models match the performance of the commonly used gradient boosted tree algorithm, which is often regarded as the state-of-the-art machine learning model?
- Can we improve the GAMs by training an ensemble of local models rather than a single global model for the whole data set?

We propose a greedy algorithm that merges city districts iteratively, finding the best merger at each iteration based on model accuracy on a validation set. We compare the results of this procedure with that of a single global model and that of K completely local models.

The rest of this paper is organized as follows: The Oslo housing market data set is introduced in Section 2. Section 3 gives an introduction to gradient boosted trees, Generalized Additive Models and outline some similarities and differences between the two, while also demonstrating why the latter is considered a more interpretable model than the former. Section 4 introduces the greedy method for clustering of groups. Prediction performance on a separate test set, both with and without using the greedy algorithm to cluster together groups, is shown in Section 5.

2 About the data set

2.1 About the data set

We use a data set of all arms' length transactions of apartments in Oslo (Norway) in the time period 2019-2021. Oslo is the capital and largest city in Norway. The total number of observations in the data set is $N = 29\,933$, where each data point represents a single sale, i.e. a transaction of one dwelling. Each transaction includes a SalePrice (the response variable in the regression) and $p = 14$ features of the specific dwelling (the independent covariates in the regression). These features contain information that is typically of high importance to home owners, such as size of the dwelling (in m^2), the number of bedrooms, the exact location (represented by the Longitude and Latitude), the floor the apartment is on, the age of the dwelling and so on.

The data set also includes information about the neighborhood. This data is based on a division of Norway into grids of size 250×250 meters. We can count the number of homes and other buildings (shops, schools, churches, etc.) in the grid cell in which the dwelling lies, as well ass all (eight) adjacent grid cells. This information is summarized in the covariates NearbyHomes and NearbyBuildings. This information is valuable to potential buyers of an apartment, as it gives insight into what kind of area the dwelling is in.

Furthermore, Oslo is divided into $K = 96$ distinct and non-overlapping city districts. These city districts are depicted in Figure 1 with color coding showing the mean price per square meter. The prices are measured in thousands NOK. [1]

We have concentrated the analysis on only those sales registered as apartments, thus excluding other estate types such as row house, detached homes and duplexes. This will naturally lead to a higher density of transactions from central areas, as the areas outside of the city centre is often dominated by detached homes. Finally, sale date is given as SaleMonth with values ranging from 1 to 24 (two years corresponds to 24 months). This makes it easier to handle in a regression setting, and the sacrifice in model accuracy by doing this is minimal.

The data set is summarized in Table 1.

Table 1: The variables in the data set with summary statistics for the numerical variables.

Variable	Unit	Mean	St. Dev.	Min	Max	Type
Sale Price	NOK (mill.)	4.69	2.14	1.26	67.5	Numerical
City District[1]	-	-	-	-	-	Categorical
Sale Date	months	9.57	6.00	1.00	24.00	Numerical
Altitude	m	90.27	61.68	0	480	Numerical
Size [2]	m^2	65.63	24.24	15.00	370.00	Numerical
Floor[3]	-	3.02	1.89	-4	14	Numerical
Bedrooms	-	1.79	0.76	0	9	Categorical
Dwelling Age	years	61.27	37.4	0	218.00	Numerical
Balcony[4]	-	0.75	0.43	0	1	Binary
Elevator[4]	-	0.37	0.48	0	1	Binary
Units On Address[5]	-	20.54	27.49	0.00	274.00	Numerical
Coast Distance	m	3,160	2,395	5	12,201	Numerical
Lake Distance	m	966.60	497.37	31	3,183	Numerical
Nearby Homes[6]	-	2,815.72	1,589,6	100	6,746	Numerical
Nearby Buildings[7]	-	166.66	144.38	6	1,323	Numerical

[1] There are 96 distinct city districts in the data set.
[2] The living area in m^2.
[3] If the dwelling has multiple floor, this variable will be the lowest floor. For detached homes this is set to 1.
[4] In cases where the information is missing, this is set to 0.
[5] In some cases, e.g. in apartment buildings, multiple dwellings have the same address.
[6] The time of sunset as of July 1st.
[7] Norway is divided into squares of 250m×250m. This variable counts the number of homes or other buildings (stores, schools, churches etc.) in all the adjacent squares to the square that the target dwelling is in, i.e. the 8 neighbouring squares.
[8] This counts the number of homes or other buildings (stores, schools, churches etc.) in the adjacent eight 250m×250m squares, as well as the neighbors of these eight squares. In total, this includes 25 squares.

3 Tree stumps, gradient boosting and GAMs

In this section we briefly present gradient boosted trees and generalized additive models (GAMs), two popular methods for predictive tasks. We also show how gradient boosted trees can be considered a special case of a GAM when we use decision trees with a depth of one (so-called tree stumps).

3.1 Gradient boosted trees

Gradient boosted trees is an ensemble method that combines multiple decision trees. Denote $h(\boldsymbol{x}_i; q)$ to be the output of a decision tree with tree structure q. Encoded in the tree structure is the division of the feature space into J distinct and non-overlapping regions denoted $\mathcal{R}_j$, each equipped with a value $w(j)$ for

[1] 1 NOK $\approx$ 0.1 USD as of May 10th 2022.

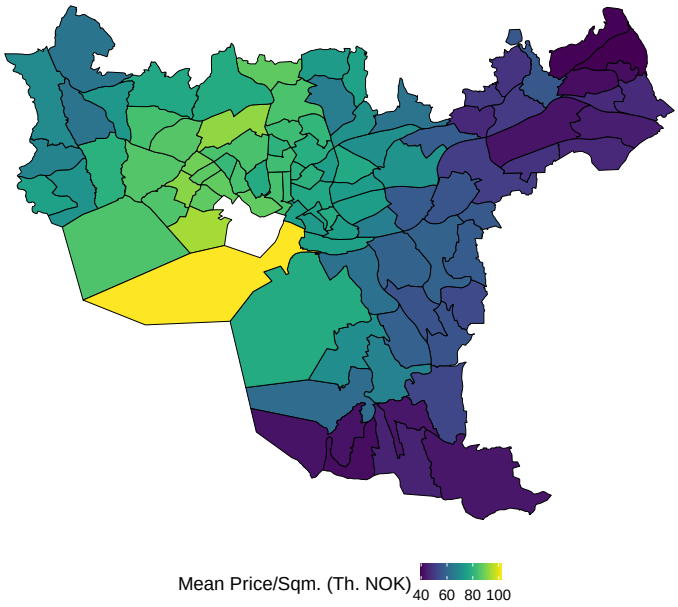

Figure 1: The $K = 96$ city districts in the data set with mean sale price per m^2 (measured in thousand NOK). The white area represents a city district ("Sentrum") that is left out of the data set due to too low sample size.

$j = 1, ..., J$. Formally,

$$h(\boldsymbol{x}; q) = \sum_{j=1}^{J} w(j) \cdot \mathbb{I}(\boldsymbol{x}_i \in \mathcal{R}_j).$$

The regions $\mathcal{R}_j$ are usually created in a way that minimizes the total variance of the y values in each of the nodes, meaning that the decision tree finds splits that group together observations with a similar response in each node. This was first introduced by Breiman et al. (1984).

Gradient boosted trees train multiple decision trees in a sequential manner where each tree is trained on the residuals from the previous trees. After training M such trees in a sequence, the final prediction is

$$f(\boldsymbol{x}) = \sum_{m=1}^{M} \eta \cdot h_m(\boldsymbol{x}; q_m)$$

where h_m is tree number m in the sequence and $\eta \in (0, 1]$ is a regularization parameter. This sequential model fitting to the residuals of the previous tree bears close resemblance to the gradient descent method often used in various optimization tasks, where one searches for the minimum in a step-wise fashion by repeatedly moving in the direction of the gradient. In fact, Friedman (2001) showed that gradient boosting is nothing but a gradient descent method in function space.

Even before this connection to the gradient descent method was established, the idea of boosting a model by re-training it on the residuals from the previous iteration was first introduced as a computational technique by Freund (1995), Schapire (1990) and Freund et al. (1996). In much of this original work the focus was on an ensemble of so-called *weak learners*, i.e., learners that on its own is merely slightly better than random guessing. Schapire (1990) showed that any strong learner, i.e. highly accurate model, can be expressed as an ensemble of weak learners via boosting. The boosting idea does not rely on a sequence of learners that on its own are very precise, but quite the contrary: Weak learners will yield precise predictions if we combine a large sequence of them, each one correcting the mistakes of the previous one. Schapire et al. (2012) likened boosting to "garnering wisdom from a council of fools". Despite this, boosting with stronger learners may often show better performance in practical applications, but too strong learners might quickly

lead to over fitting of the training data (Bentéjac et al. (2021), Ridgeway (2005)). The tree depth D and learning rate η should be picked based on the specific problem at hand, or found via systematic tuning.

A major drawback of gradient boosted trees, however, is the lack of interpretability. While a single decision tree is inherently easy to inspect and understand, a sequence of M trees quickly becomes hard to understand, especially as $M \sim 10^3$ in most applications. A single tree of depth D divides the feature space in up to 2^D distinct regions. Typical tree depths are $4 < D < 8$, yielding a large number of regions per tree. With these tree depths we also include a large number of interactions between the covariates, making it difficult to isolate the effect of each covariate.

Although the predictions from gradient boosted trees are difficult to interpret, some methods exist. Most notably, one can construct a *partial dependence plot*, which attempts to visualize the marginal effect of any of the covariates x_j on the dependent variable y. In order to generate a partial dependence plot we must find a way to marginalize out the effect of all the other covariates $x_1, ..., x_{j-1}, x_{j+1}, ..., x_p$. Greenwell (2017) presents a practical way of doing this: For a certain observation in the data set we modify the true observed x_j repeatedly while keeping all the other covariates fixed, each time using the gradient boosted tree to make a prediction based on this modified observation. If we do this repeatedly in a discrete grid, we can plot $(x_j, \hat{y})$ for this observation. If we repeat this for all N observations, we get N curves and can take the average curve as an approximation of how different values of x_j affects the prediction $\hat{y}$.

While the partial dependence plot gives a hint about the shape of the prediction $\hat{y}$, it does not give a perfect explanation in the following sense: Given a set of features $(x_1, ..., x_p)$ in a new observation, we can not simply look at the average partial dependence plots and use these as a prediction. Due to the interaction terms that are introduced by having deeper trees, a partial dependence plot of $(x_j, \hat{y})$ will not reveal to us the exact prediction corresponding to a set of features $(x_1, ..., x_p)$. Instead this plot will give us the prediction given x_j, averaged over all the other input variables. An example of a simple gradient boosted tree with interactions, together with a corresponding partial dependence plot, is shown in Appendix B.

3.2 Generalized Additive Models (GAM)

An Additive Model is a any model that takes the form

$$\mathbb{E}(y) = f_0 + f_1(x_1) + f_2(x_2) + ... + f_p(x_p), \tag{1}$$

where x_1, ..., x_p are features and $f_1(\cdot), ..., f_p(\cdot)$ are called *shape functions*. This framework allows for more flexible regressions than what can be achieved via traditional OLS linear regression or Generalized Linear Models (GLM), where a link function of the mean of the response y is modelled as a known function of covariate x_j, or specified transformations of x_j. In a additive model, the dependency on the covariate x_j is specified through an unknown function $fj(x_j)$, whose shape can be learned. The Additive Model can be expanded into a Generalized Additive Model (GAM) by adding a *link function* $g(\cdot)$ such that

$$g(\mathbb{E}(y)) = f_0 + f_1(x_1) + f_2(x_2) + ... + f_p(x_p). \tag{2}$$

The link function makes it possible to make assumptions about the mean of the underlying data. For instance, the link function $g(\cdot) = \log(\cdot)$ implies data that is Poisson distributed. By using the so-called identity link function $g(\mathbb{E}(y)) = \mathbb{E}(y)$ we retrieve the simple Additive Model. Note also that we can easily express standard linear regression in this framework by letting $f_j(x_j) = \beta_j \cdot x_j$ for every $j = 1, .., p$ and estimate β via maximum likelihood, as is normally used in linear regression. The GAM framework was introduced by Hastie et al. (1986).

The flexibility offered by the GAM framework has multiple benefits: First and foremost, it can be useful with a large variety of assumptions on the underlying data. Secondly, this kind of model is much more interpretable than a function that takes multiple covariates as input, thus creating interactions. The reason for this is that we can easily visualize each of the shape functions by plotting (x_j, f_j). This highlights the appeal of the additive nature of the model: By simply looking at all the shape functions $f_0, f_1, ..., f_p$ we can determine what the final prediction $\hat{y}$ is. It also makes it straight-forward to answer questions such as "how would $\hat{y}$ change if we change x_1 by one unit?" with certainty.

Traditionally, the preferred choice of shape functions in GAMs has been smoothing splines (Wood (2006)). Smoothing splines are piece-wise polynomial functions with the additional requirement that the polynomials

should have similar first and second derivative in the knots where two polynomials meet. This has the benefit of creating a smooth curve that interpolates the data in a highly flexible manner. In principle, however, all sorts of different models can be used to fit the shape functions $f_1, ..., f_p$. Recent years have seen an increased interest in using decision trees or ensembles of decision trees to fit each base learner. Lou et al. (2012) provides a comprehensive comparison of different shape functions in GAMs and note that splines tend to underfit, while tree-based methods are prone to overfitting and must thus be regularized carefully. They conclude that a bagging of shallow trees – defined by them as trees with depth of 1 or 2 – provide the best accuracy. Furthermore, Chang et al. (2021) note that even in cases where splines and tree-based shape functions have the same accuracy on a test set, the shape plots reveal unexpected and unreliable behaviour for some of the splines. For these reasons we will focus on tree-based GAMs.

The fitting of the GAMs depends on the choice of shape functions. Some shape functions, like linear regression or splines, can be obtained simultaneously in closed form. Other shape functions, like tree based methods, cannot be calculated in closed form and require iterative procedures. The historically most used method for this is backfitting (Hastie et al. (2001)).

3.3 Gradient boosted tree stumps as a GAM

At first glance, gradient boosted trees and GAMs seem to have few similarities. However, in the special case where each tree use tree depth of 1 we can indirectly turn a gradient boosted tree ensemble into a GAM. A tree of depth 1 is equivalent to dividing the feature space in two, i.e.

$$\mathcal{R}_1 = \{\boldsymbol{x} : x_j \leq a\}, \quad \mathcal{R}_2 = \{\boldsymbol{x} : x_j > a\}$$

for some value a. Note that each tree in the gradient boosted tree ensemble then will be a simple step function that only utilize one of the p covariates. A tree of depth 1 is often referred to as a tree stump. Although a simple tree stump in itself will provide a quite uninformative binary split of the x_j feature space, an accumulation of multiple binary splits on x_j will sum to a non-smooth function $f_j(x_j)$.

Training a full gradient boosted tree algorithm where each tree consists of a single split will effectively turn the gradient boosted tree into a GAM, since each tree will make a split on one of covariates x_j without creating any interactions. In order to visualize the pair (x_j, f_j) we can simply gather all the trees that split on x_j and discard the other trees. This can be repeated for all the p covariates, giving an additive structure. A simple example of how a sequence of tree stumps form an additive model is illustrated in Appendix B.

3.4 A simple example

We simulate data from the following model:

$$\begin{aligned} x_1 \sim \mathcal{N}(0,1), \quad x_2 \sim \mathcal{N}(0,1), \quad x_3 \sim \mathcal{N}(0,1) \\ y = 0.1x_1^2 + \sin(x_2) + \mathbb{I}\Big(\cos(x_3)\Big) + \varepsilon, \quad \varepsilon \sim \mathcal{N}(0, 0.5), \end{aligned} \tag{3}$$

where y is the response and x_1, x_2, x_3 are covariates. The function $\mathbb{I}(x)$ is an indicator function that is 1 if $x > 0$ and 0 otherwise. We train a GAM on $N = 1\,000$ samples from the above model, using boosting with $M = 1\,000$ iterations (tree stumps) and a learning rate of $\eta = 0.03$. The model results in prediction on the form $\hat{y} = f_0 + f_1(x_1) + f_2(x_2) + f_3(x_3)$, and the estimated shape functions are plotted in Figure 2. The plots show the constant intercept f_0 as well as the pairs $(x_1, f_1), (x_2, f_2), (x_3, f_3)$. The functions are not smooth, but rather a sum of many step functions (on average 333 binary splits on each covariate). The true function used to generate the data as per Equation 3 is displayed in a dotted blue line. The function that seems to best approximate the underlying truth is f_2. However, we must not be fooled by the fact that the f_1 and f_3 functions lie somewhat below the true functions $0.1 \cdot x_1^2$ and $\mathbb{I}\Big(\cos(x_3)\Big)$; the final prediction is a sum of all the shape functions given covariate values (x_1, x_2, x_3), including the intercept that is close to 1. Of major importance is therefore whether or not the functions f_1, f_2, f_3 are showing a similar shape as the underlying data. This seems to be the case, even though all of the shape functions diverge somewhat from the underlying truth in the edges of each plot. The shape functions f_1 and f_2 flatten out in the edges, due to the fact that there are less observations here for the model to train on.

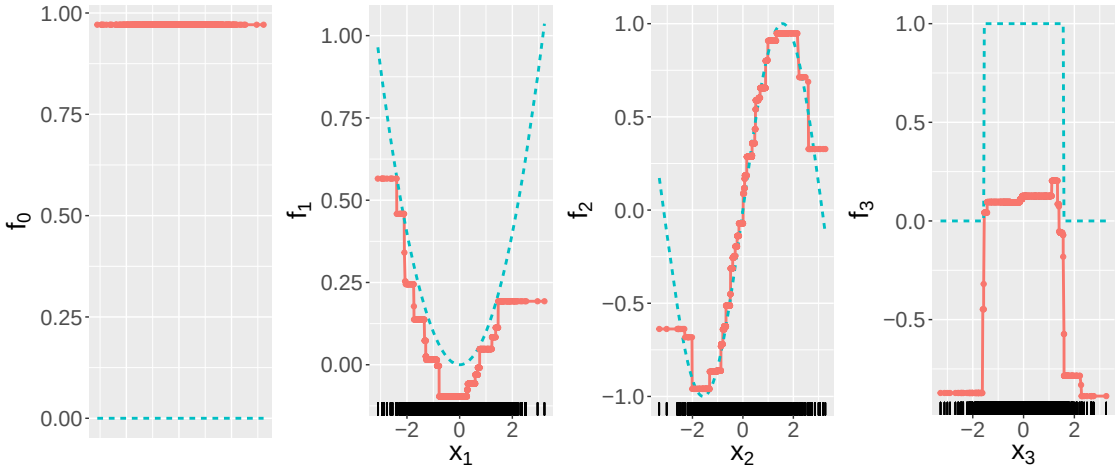

Figure 2: The shape functions from a GAM trained on the synthetic data set in (3) with $M = 1\,000$ sequential tree stumps. The first plot show the constant intercept f_0, while the other three plots show the pairs $(x_1, f_1), (x_2, f_2), (x_3, f_3)$. The black rug at the bottom of the plots show at what x values the observations are from. The blue dotted line is the true value that the data set is simulated from.

3.5 Preliminary results

We train two models on the data set of transactions from Oslo: a GAM with tree stumps as shape functions and a gradient boosted tree method with deeper trees. For the GAM models we use the `mboost` implementation by Hofner et al. (2014), while the deeper gradient boosted tree ensemble is trained by using the `XGBoost` implementation by Chen et al. (2016). We use a tree depth of $D = 8$ in the deep gradient tree ensemble. In both methods we use learning rate of $\eta = 0.05$ and $M = 10\,000$ iterations.

We randomly sample half of the data set for training and the other half for testing, giving $N_{train} = 14\,968$ and $N_{test} = 14\,965$. Table 2 show the performance of the three methods on the test set. We report Root Mean Squared Error (RMSE) and Median Absolute Error (MdAE), where all errors are reported as a percentage of the sale price. We also report R^2.

Table 2: Root Mean Squared Error (RMSE), Median Absolute Error (MdAE) and R^2 of the GAM and the gradient boosted trees. All errors are reported as percentages of sale price when the models are applied to a new test set of $N_{test} = 14\,965$ observations. All methods have used a learning rate of $\eta = 0.05$ and $M = 10\,000$ iterations. The gradient boosted trees is trained with XGBoost and with trees of depth $D = 8$.

Model	RMSE (%)	MdAE (%)	R^2(%)
GAM (tree)	15.5	8.4	82.3
Gradient boosted trees	10.1	5.4	89.1

The results in Table 2 clearly shows that the full complexity model (gradient boosted trees with depth 8) outperforms both the GAM on all performance metrics. Figure 3 shows a grid of 15 plots, corresponding to the shape plots of the 14 numerical covariates plus the intercept. The intercept, $f_0 = 4.58$ corresponds to the mean price in the training set. Any prediction will thus use this as the starting point, and then adjust the estimate up or down through the other shape functions. Notice that CityDistrict is a categorical covariate taking $K = 96$ levels. The shape function $f_{CityDistrict}$ is therefore simply a set of $K = 96$ constant values, one for each city district. The shape plot $f_{CityDistrict}$ displays these values from smallest to largest. Each tick mark represents one city district. The y axis in each of the plots is measured in million NOK. Although many of the plots, like f_{Size}, $f_{Bedrooms}$ or $f_{SaleMonth}$ tend to give rather smooth curves, other such as $f_{BuildingsNearby}$ or $f_{LakeDistance}$, are rather erratic and are less easy to understand. For instance, $f_{BuildingsNearby}$ has a sudden drop around 400 that is hard to explain. The shape function $f_{CoastDistance}$

shows a large hike for very small values before stabilizing around 0. This is intuitively easy to explain: People are willing to pay a lot to live very close to the sea, but the difference between living 5 000 meters and 6 000 meters from the sea will typically not matter much.

However, when considering these plots one must also look at the magnitude of the y axis. The erratic $f_{BuildingsNearby}$, for instance, varies between approximately -0.1 and 0.1, i.e. between -100 and 100 thousand NOK. This is a relatively modest amount, considering that the average price in the data set is 4.69 million NOK. It's not surprising that f_{Size} seems to have the biggest contribution to the final prediction, based on the magnitude of y axis.

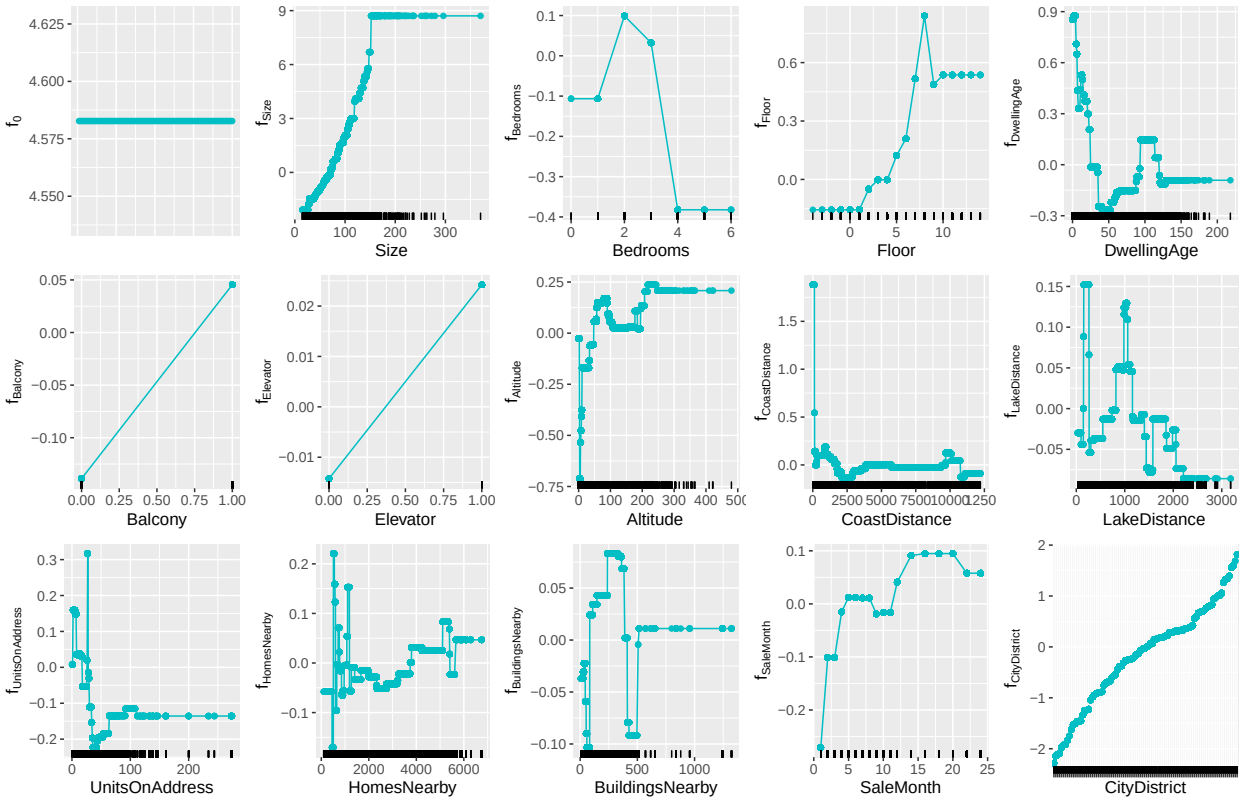

Figure 3: All the $p = 14$ shape functions (plus the intercept f_0) from the GAM trained with tree stumps. The values on the y axis is in million NOK. The covairates Balcony and Elevator are both boolean (that is, they take either the value 1 or 0). The CityDistrict covariate takes one of $K = 96$ different values. The shape plots $f_{CityDistrict}$ therefore shows 96 dots, sorted from smallest to largest for convenience.

Let us consider a specific example of an apartment in the training data. This has the true sale price of $y = 7.30$ (million NOK) and the GAM model predicts $\hat{y} = 7.48$. By calculating $f_j(x_j)$ for all the specific x_j features of this dwelling we can visualize exactly how we got to the prediction. Figure 4 shows this for one specific dwelling that is located in the city district Skillebekk. For this specific dwelling we observe that the two covariates that contribute most towards the final prediction is $f_{CityDistrict}$ and f_{Size}. The size of the bars in Figure 4 corresponds exactly to the f_j values from Figure 3, which highlights the explanatory nature of the GAMs.

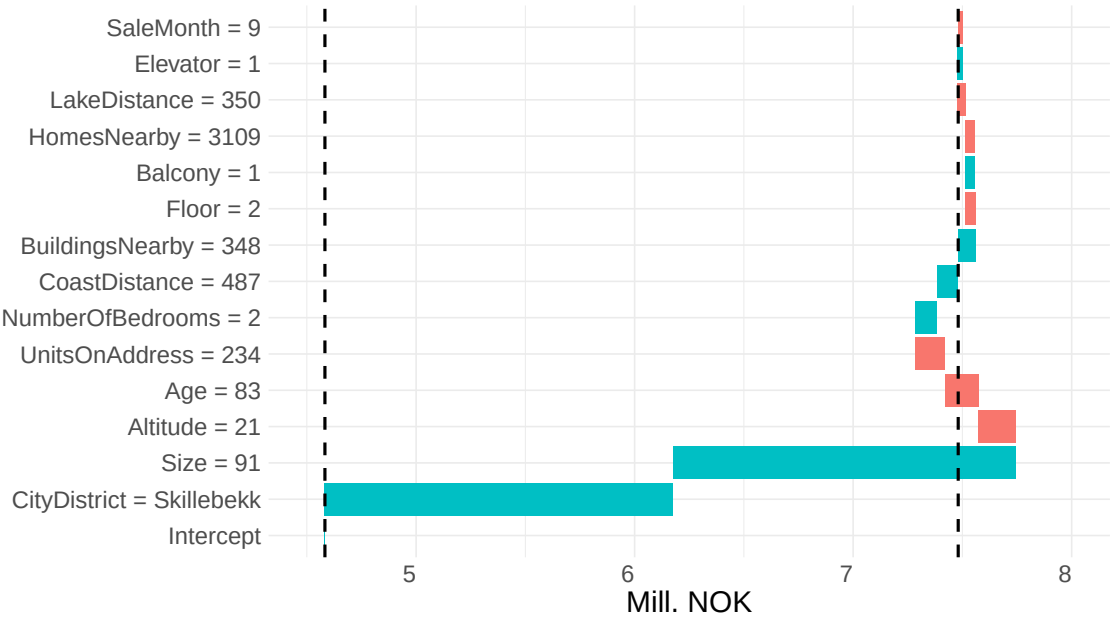

Figure 4: A plot showing the contribution from each shape function towards the final prediction. The left dashed line shows the intercept $f_0 = 4.58$ million NOK and the right dashed line shows the prediction $\hat{y} = 7.48$ million NOK. Covariates that give a postiive contribution to the prediction predictions are filled with blue, while covariates that give a negative contribution to the predictions are filled with red. The true sale price observed for this dwelling was $y = 7.30$ million NOK.

4 Towards more local models

It is clear from the previous section that the GAM models struggled to perform at the level of the full complexity model. One possible explanation is the large number of levels for the categorical features. With GAMs we can not encode interactions between different covariates, thus we are unable to encode that there is e.g. a different slope for the Size feature in different city districts. With the GAM framework we assume that the shape functions are equally shaped in each city district, but shifted up or down by a constant.

Is this a good assumption? In order to investigate this problem we take the polar opposite view: What if we trained K completely local models, that is, one model per city district? This would have the benefit of creating highly specialized models, but the obvious downside is the lack of available data for many city districts; some city districts have less than 10 observations. Figure 5 shows the shape plot of $f_{size}(\cdot)$ for the two city districts Bygdøy og Rommen. These are the city districts in the data set with highest and lowest mean sale price, respectively. They have thus been deliberately picked to demonstrate the point, namely that different city districts not necessarily have a similar shape plot. The right plot in Figure 5 uses a simple linear regression as an analogy for the shape plots produced by the GAMs: All city districts does not necessarily have the same β_{size}.

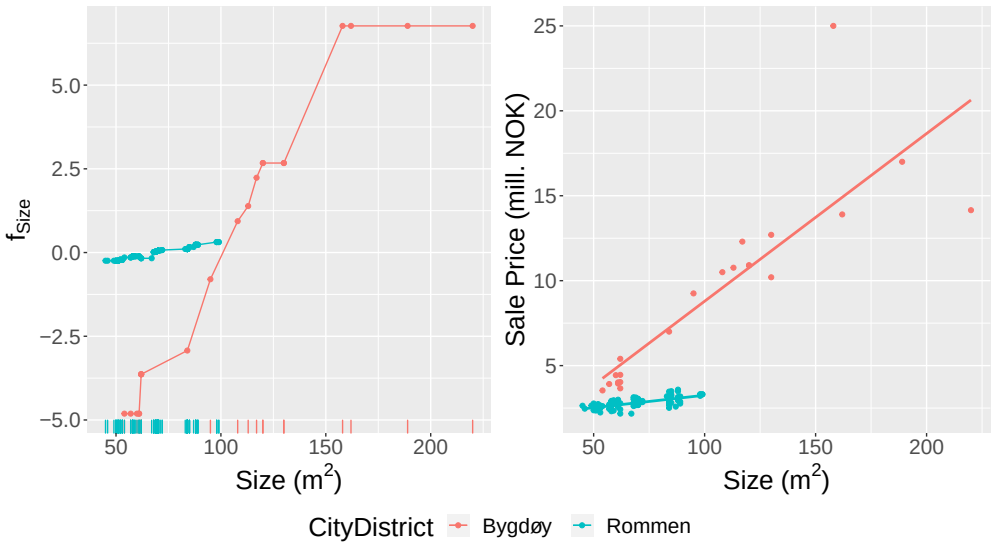

Figure 5: **Left:** Shape plots of f_{Size} from two local models trained on the city districts Bygdøy and Rommen. **Right:** A plot of the actual observations from the two city districts with a linear regression line showed for illustrative purposes.

Is the solution simply to train $K = 96$ completely local models? Not necessarily. The rug (i.e. the tick marks at the bottom of the plot) in Figure 5 reveal that some city districts contain quite few observations. This is often symptomatic of housing market data, as some parts of the market tend to have higher activity than others. Training K completely local models will in theory yield highly specialized experts in each of the city districts, but with so little data available for some city districts, there is no guarantee that this will be true in practice. Besides, many city districts display shape plots that are much more equal than the one displayed in Figure 5. Figure 6 shows the shape plots of f_{Size} for two local models trained on the the two neighbouring city districts Sagene and Torshov. The shape function from these models show a much clearer similarity, which is also evident from the right plot that shows β_{Size} from the two models.

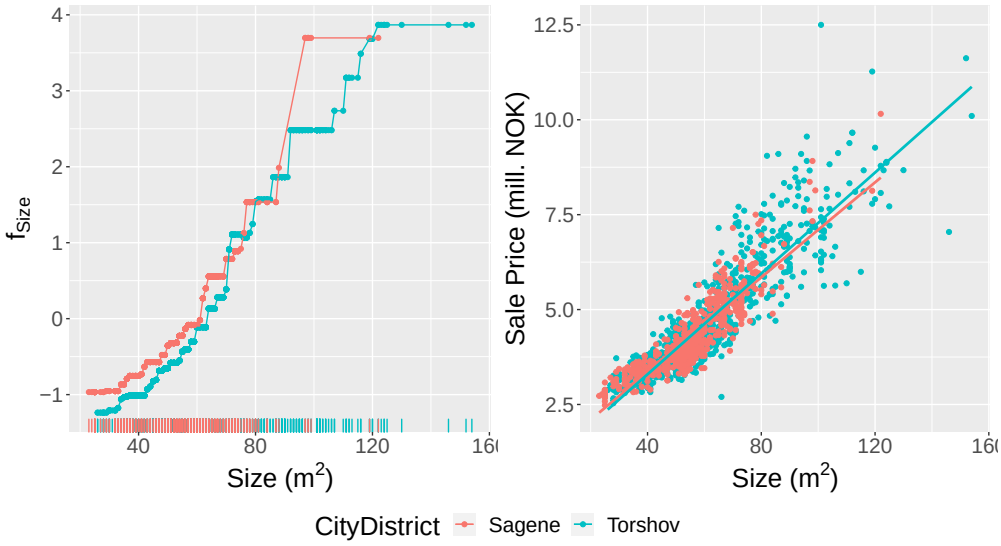

Figure 6: **Left:** Shape plots of f_{Size} from two local models trained on the city districts Sagene and Torshov. **Right:** A plot of the actual observations from the two city districts with a linear regression line showed for illustrative purposes.

The above examples illustrates that K completely local models not necessarily is an optimal choice, and some sharing of information between city districts can be beneficial. At the same time, Table 2 showed that the performance of a single global GAM model did not match the performance of full-complexity models. A compromise is to train $L < 96$ separate models, where similar city districts are merged together. The next subsection presents a *greedy* approach to this problem.

4.1 Greedy Algorithm

A greedy algorithm is a term for describing a paradigm in combinatorial optimization where an algorithm makes a locally optimal choice at every iteration of the algorithm without considerations of future implications of the choice. While this sequence of locally optimal choices not necessarily lead to the globally optimal solution, it is a good approximation when finding the global optimum is too expensive.

A greedy algorithm for the regression problem with a large number of city districts would consists of sequentially reducing the number of groups by merging two groups together at every iteration. This process is continued until a stopping criterion is reached. The stopping criterion will typically be related to the model performance or a pre-determined number of iterations. At every iteration, the algorithm should merge together the two groups that give the best improvement in performance.

How should we determine what gives the best improvement in performance? The intuitive approach is to test all combinations. That is, we try to merge every city district i and j and evaluate the change in performance measured by the prediction error on a validation set. When city districts i and j are merged we are left with $(K-1)$ groups. We then train $(K-1)$ completely local models and evaluate the model performance on a validation set. If we do this for every pair of city districts, we will obtain a $K \times K$ matrix, where element (i, j) contains the model performance in the scenario where city district i and j are grouped together. Note that the diagonal elements in this matrix will all contain the same number, since that would be the scenario where no groups are merged. We can now easily pick the smallest element in the matrix, and this would correspond to finding the city districts (i^*, j^*) that lead to the biggest reduction in error. We merge these and repeat the process until we find no further improvement. A detailed description of the algorithm is shown in Algorithm 1.

The benefits of this approach is that we actually test all combinations empirically on a validation set, thus giving a sequence of merge operations that are probably fairly good. It is also a model agnostic approach; the $(K-1)$ local models can be any type of statistical model, and even – in theory – different types of model. The downside of this approach, however, is the computational cost. We need to train $\binom{K}{2} = \frac{1}{2}K \cdot (K-1)$ models in order to initiate the algorithm (the $1/2$ is due to the symmetry involved; merging city district i and j is the same as merging city district j and i). After merging two groups we must train another $(K-1)$ model, then $(K-2)$ in the next iteration, and so on. With $K \approx 100$, as in the Oslo data set, we must train thousands of models. This is a significant computational challenge.

Algorithm 1 Greedy algorithm

procedure GREEDYMERGING(
)Train K completely local models on the K city districts
Initialize a symmetric error matrix $M \in \mathbb{R}^{K \times K}$ where $M[i, j]$ is the RMSE (%) on a validation set after city district i and j is merged
Normalize M s.t. $diag(M) = 1$
$SmallestError = min(M)$
while $SmallestError < 1$ **do**
$[i*, j*] = which(M == SmallestError)$
Merge city district $i*$ and $j*$
Delete row $j*$ and column $j*$
$Improvement = 1 - SmallestError$
$M = M - Improvement$
Recalculate row $i*$ and column $i*$ in M
$SmallestError = min(M)$

However, the greedy approach also allows for other choices of merging criterion. Another approach that is similar in spirit, but computationally faster, defines the merging criterion based on out-of-sample performance. Rather than actually merging city district i and j we can train a model on city district i and evaluate it on city district j. A good out-of-sample performance can be taken as an indicator that city district i and j are similar. This method drastically reduces the total number of models that need to be trained. In order to initiate the algorithm we must train K local models (as opposed to $\frac{1}{2}K \cdot (K-1)$), and for every iteration we must simply train one additional model. However, at every iteration we must evaluate one model on the $(K-1)$ other city districts. This also has a computational cost, but it is generally considered smaller than the cost of training $(K-1)$ models.

It must be noted that this approach does not have the same symmetry as the previous greedy approach: A model trained on city district i and evaluated on city district j does not necessarily give the same result as a model trained on city district j and evaluated on city district i. To make it easier to decide which two city districts to merge we enforce a symmetry by considering the average of element (i, j) and (j, i) in the matrix.

A detailed description of the greedy out-of-sample is described in Algorithm 2.

Algorithm 2 Greedy algorithm with out-of-sample merging criterion

procedure GREEDY ALGORITHM (OUT-OF-SAMPLE)(a, b)
  Train K completely local models on the K city districts;
  Divide a validation set into K parts such that part j consists of data only from city district j
  Initialize a symmetric error matrix $M \in \mathbb{R}^{K \times K}$ where $M[i, j]$ is the RMSE (%) when using a model i on validation set j
  Create $M_{symmetric}$ such that $M_{symmetric}[i, j] = \frac{1}{2}\Big(M[i, j] + M[j, i]\Big)$
  $SmallestError = min(M)$
  while iteration $<$ MaxIteration **do**
    $[i*, j*] = which(M_{symmetric} == SmallestError)$
    Merge city district $i*$ and $j*$
    Delete row $j*$ and column $j*$
    Recalculate row $i*$ and column $i*$ in M
    $SmallestError = min(M_{symmetric})$

5 Results

We saw in Table 3 that the gradient boosted tree significantly outperformed the GAM model, giving RMSE of 10.1% and 15.5%, respectively. We now seek to investigate if clustering city districts through a greedy procedure will improve the performance of the GAM. As touched upon in Subsection 4.1, the Greedy Algorithm requires us to train thousands of models, which is practically infeasible. We therefore use the much quicker GreedyOutOfSample algorithm to reduce the number of models iteratively from $K = 96$ (completely local models for each city district) to $K = 1$ (a single global model).In order to smooth out noise related to specific splits of the data into training set and test set, we perform 20 simulations and report the mean performance.

The mean results of the simulations can be seen in Figure 7. The plot shows performance, measured by MdAE, RMSE and R^2, against the number of groups, ranging from 1 to 96. The left side of the plot, where the number of groups is one, corresponds to training a single global model for the whole data set. These numbers are the same that was reported in Table 2. The red dashed line shows the performance of the gradient boosted tree with depth 8.

No matter the number of groups, we are not able to reach the performance of the gradient boosted tree on any of the performance measures. However, we do see some encouraging results: The RMSE is reduced from the original 15.5% to below 12% as long as the number of groups are above approximately 25. The MdAE is reduced from the original 8.4% towards approximately 6.2%. The R^2 improves quickly when going from one to approximately 15 groups, but stabilizes quickly thereafter.

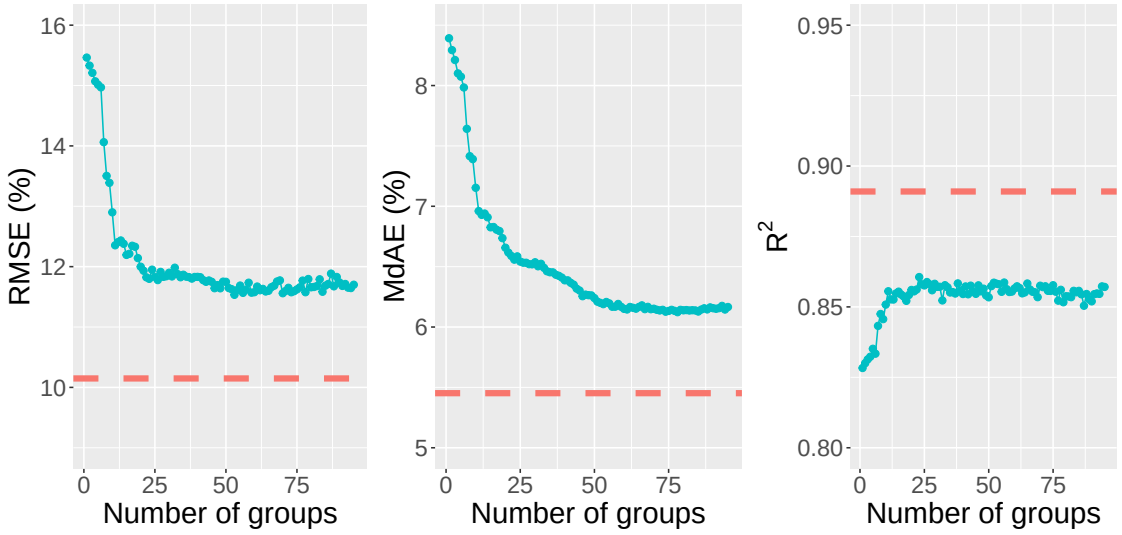

Figure 7: The mean results of running 20 simulations with the GreedyOutOfSample-algorithm. The plots should be read as follows: To the left in each plot, the number of groups is 1, i.e., we use a single global model for the whole data set. To the right in each plot the number of groups is 96, i.e., we use 96 completely local models (one for each city district. We use the GreedyOutOfSample-algorithm to go from 96 models to 1 model by merging two models in each iteration. We only consider tree-based GAM models. The red dashed line shows the performance of the gradient boosted trees with trees of depth 8, serving as a performance benchmark. **Left:** Root Mean Squared Error (RMSE). **Middle:** Median Absolute Error (MdAE). **Right:** R^2.

There seems to be no doubt that going towards more local model improves the performance of the GAMs. It is hard to determine the optimal number of groups, as it varies somewhat depending on what performance measure we choose. One can argue that 96 groups – i.e. one model for each city district – seems to be the optimal number. However, having a large number of models is also computationally and practically challenging. Table 3 reports the performance for different number of groups. With 53 groups we achieve the best RMSE. Note, however, that by using merely 20 groups we achieve an RMSE of 12.0%, a significant reduction from the original 15.5%.

Table 3: Model performance with different number of groups, where the GreedyOutOfSample algorithm is used to determine which city districts that are to be merged. The results above the dashed line are the one reported in Table 2. The 53 groups are chosen because it is the one minimizing the RMSE. As expected none of the GAM models are able to achieve performance at the level of the gradient boosted trees.

Model	RMSE (%)	MdE (%)	R^2(%)
GAM (1 group)	15.5	8.4	82.3
Gradient boosted trees	10.1	5.4	89.1
GAM (greedy, 96 groups)	11.7	6.2	85.7
GAM (greedy, 53 groups)	11.5	6.2	85.8
GAM (greedy, 25 groups)	11.8	6.5	85.8
GAM (greedy, 20 groups)	12.0	6.7	85.6
GAM (greedy, 10 groups)	12.9	7.2	85.1

Although it seems evident that going towards more local models improve the model accuracy of the GAMs, it also introduces another important question: Which city districts are being grouped together? Figure 8 shows three examples of how the clustering looks when the number of models is set to be 10. The three maps correspond to the clustering result for three different training sets, all sampled randomly from the

original data set presented in Section 2. Interestingly, all three plots show the existence of two super groups: One group that mainly includes the north-eastern and south-eastern city districts, and one group including the north-western, central northern and some eastern city districts. This has primarily two implications. Firstly, it is reassuring that the same structures seem to appear even with a slightly different training set. This strengthens the belief that these structures are indeed present, and not merely noise. Secondly, it is surprising and perhaps slightly problematic that all the other 8 groups consist of a single city district each. Rather than having 10 clusters with e.g. $5 - 15$ city districts in each, we see two super groups and eight completely separate city districts. One exception is the second map, which seems to have two neighboring city districts clustered together slightly left of the city centre.

The implications of this is not immediately obvious, and further analysis of the clustering results should be conducted. Although the number of models is indeed 10 in this case, one could argue that the actual number of clusters is three: the two large groups and a group of special cases that each need their own model. Utilizing this insight about the de facto number of groups might be able to improve the greedy clustering, either in terms of accuracy of by computational speed.

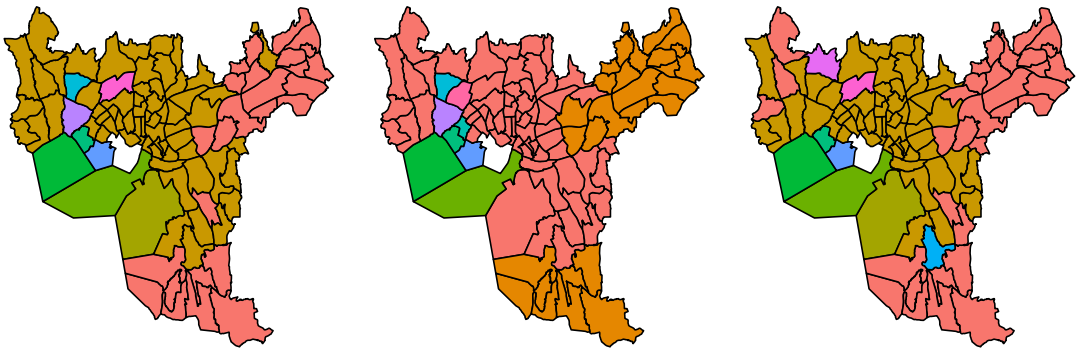

Figure 8: Three examples of how the greedy algorithm clusters when we order the number of groups to be 10. The three examples correspond to three different training sets that are randomly sampled from the same full data set presented in Section 2. The exact coloring is not of importance, and neither are color differences or similarities between the maps. The important thing is which city districts that are clustered together in each clustering.

6 Conclusion

This paper discusses the challenge of creating statistical models for house price prediction and evaluate the models both by their accuracy on a test set and by their interpretability. We argue, inspired by the works of Lou et al. (2012), Lou et al. (2013) and Chang et al. (2021), that Generalized Additive Models (GAMs) trained with tree stumps are more interpretable than the state-of-the-art machine learning models like gradient boosted trees.

The performance of the GAMs are considerable worse than the performance of gradient boosted trees on a data set of $N = 29\,933$ transactions from Oslo, Norway, from 2018 and 2019. However, using several local models rather than a global model improves model performance considerably, reducing RMSE from 15.5% to 11.5%. This is, however, still not as good as the performance of the state-of-the-art prediction model, gradient boosted trees with deeper trees, which yielded an RMSE of 10.1%.

Despite claims of the contrary from e.g. Rudin (2019), many empirical studies – including this one – point towards the existence of a trade off between how accurate a machine learning model is, and how intelligible it is. It is impossible to determine a general rule about how much accuracy one should sacrifice in order to get more interpretable models, especially since the notion of interpretability is hard to define on its own. By utilizing several local models rather than a single global we are able to get the performance of the GAMs closer to the gradient boosted tree. Whether or not the performance is close enough to prefer the former model depends entirely on the context the models should be used within, the importance of being able to

explain the prediction and the preference of the user.

There are many interesting directions for future research to improve the paper. The grouping of similar city districts is done via a greedy algorithm, as described in Subsection 4.1. This requires training a large number of models, and evaluating the model a large number of times. Another approach that makes sense both computationally and methodologically is to use the shape plots to measure the similarity between groups. To motivate this, let us imagine that we train K completely local linear regression model. We can then inspect the β coefficients of each of the model and argue that those city districts that have similar β coefficients share some characteristics and should thus be clustered together. Creating a similar methodology for GAMs is not straight forward, as it raises the question of how to quantify the similarity (or lack there of) between curves that are non-parametric and non-smooth. Numerical integration methods will likely be necessary.

Furthermore, it would be interesting to compare the results of both the GAM models and the full complexity model with the Catboost (Categorical Boosting) algorithm (Prokhorenkova et al. (2018)). This is an implementation of gradient boosted trees that is developed specifically for data sets with categorical variables. Recently, Nori et al. (2019) also introduced Explainable Boosting Machine, a framework for training GAMs with tree stumps that also allows for some interaction terms. This will also be highly relevant to compare with. It can also be interesting to compare this with mixture model approaches.

Finally, it is worth noting that we so far have neglected the spatial connection between the city districts. We argued in the beginning that the $K = 96$ city district have no inherent ranking or ordering, unlike the levels of other categorical covariates. Despite this, there is indeed a spatial relationship between the categories that might relevant to consider. Future work might include methods that use the spatial relationships as one factor when calculating the similarity between two city districts.

6.1 Acknowledgements

Thank you to Eiendomsverdi AS for providing the data set.

References

Bailey, Martin J., Muth, Richard F., and Nourse, Hugh O. (1963). "A regression method for real estate price index construction". In: *Journal of the American Statistical Association* 58.304, pp. 933–942.

Baldominos, Alejandro, Blanco, Iván, Moreno, Antonio, Iturrarte, Rubén, Bernárdez, Óscar, and Afonso, Carlos (2018). "Identifying Real Estate Opportunities Using Machine Learning". In: *Applied Sciences* 8.11, p. 2321.

Bentéjac, Candice, Csörgő, Anna, and Martínez-Muñoz, Gonzalo (Mar. 2021). "A comparative analysis of gradient boosting algorithms". In: *Artificial Intelligence Review* 54. DOI: 10.1007/s10462-020-09896-5.

Breiman, L. (2001). "Random forests". In: *Machine Learning* 45, pp. 5–23. DOI: https://doi.org/10.1023/A:1010933404324.

Breiman, L., Friedman, J. H., Olshen, R. A., and Stone, C. J. (1984). *Classification and Regression Trees*. Wadsworth and Brooks.

Brusco, Michael J., Cradit, J. Dennis, Steinley, Douglas, and Fox, Gavin L. (2008). "Cautionary Remarks on the Use of Clusterwise Regression". In: *Multivariate Behavioral Research* 43.1. PMID: 26788971, pp. 29–49. DOI: 10.1080/00273170701836653. eprint: https://doi.org/10.1080/00273170701836653. URL: https://doi.org/10.1080/00273170701836653.

Chang, Chun-Hao, Tan, Sarah, Lengerich, Ben, Goldenberg, Anna, and Caruana, Rich (2021). "How Interpretable and Trustworthy Are GAMs?" In: *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*. KDD '21. Virtual Event, Singapore: Association for Computing Machinery, pp. 95–105. ISBN: 9781450383325. DOI: 10.1145/3447548.3467453. URL: https://doi.org/10.1145/3447548.3467453.

Chen, Tianqi and Guestrin, Carlos (2016). "XGBoost: A Scalable Tree Boosting System". In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*.

Freund, Yoav (1995). "Boosting a Weak Learning Algorithm by Majority". In: *Information and Computation* 121.2, pp. 256–285. ISSN: 0890-5401. DOI: https://doi.org/10.1006/inco.1995.1136. URL: https://www.sciencedirect.com/science/article/pii/S0890540185711364.

Freund, Yoav and Schapire, Robert E. (July 1996). "Experiments with a new boosting algorithm". In: *Proceedings of the Thirteenth International Conference on International Conference on Machine Learning*. Morgen Kaufmann Publishers Inc, pp. 148–156.

Friedman, Jerome (2001). "Greedy Function Approximation: A Gradient Boosting Machine". In: *The Annals of Statistics* 29.5, pp. 1189–1232. DOI: https://doi.org/10.1214/aos/1013203451.

Friedman, Jerome, Hastie, Trevor, and Tibshirani, Robert (2000). "Special Invited Paper. Additive Logistic Regression: A Statistical View of Boosting". In: *The Annals of Statistics* 28.2, pp. 337–374.

Gertheiss, Jan and Tutz, Gerhard (Dec. 2010). "Sparse modeling of categorial explanatory variables". In: *The Annals of Applied Statistics* 4.4. ISSN: 1932-6157. DOI: 10.1214/10-aoas355. URL: http://dx.doi.org/10.1214/10-AOAS355.

Greenwell, Brandon M. (2017). "pdp: An R Package for Constructing Partial Dependence Plots". In: *The R Journal* 9.1, pp. 421–436. DOI: 10.32614/RJ-2017-016. URL: https://doi.org/10.32614/RJ-2017-016.

Hastie, Trevor and Tibshirani, Robert (1986). "Generalized Additive Models". In: *Statistical Science* 1.3, pp. 297–310. DOI: 10.1214/ss/1177013604. URL: https://doi.org/10.1214/ss/1177013604.

Hastie, Trevor, Tibshirani, Robert, and Friedman, Jerome (2001). *The Elements of Statistical Learning*. Springer Series in Statistics. New York, NY, USA: Springer New York Inc.

Hjort, Anders, Pensar, Johan, Scheel, Ida, and Sommervoll, Dag Einar (2022). "House price prediction with gradient boosted trees under different loss functions". In: *Journal of Property Research* 0.0, pp. 1–27. DOI: 10.1080/09599916.2022.2070525.

Ho, Winky K. O., Tang, Bo-Sin, and Wong, Sui Wai (2020). "Predicting property prices with machine learning algorithms". In: *Journal of Property Research*.

Hofner, Benjamin, Mayr, Andreas, Robinzonov, Nikolay, and Schmid, Matthias (2014). "Model-based Boosting in R: A Hands-on Tutorial Using the R Package mboost". In: *Computational Statistics* 29, pp. 3–35.

Kim, Jungsun, Won, Jaewoong, Kim, Hyeongsoon, and Heo, Joonghyeok (2021). "Machine-Learning-Based Prediction of Land Prices in Seoul, South Korea". In: *Sustainability* 13.23, pp. 202–211.

Lou, Yin, Caruana, Rich, and Gehrke, Johannes (2012). "Intelligible Models for Classification and Regression". In: *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '12. Beijing, China: Association for Computing Machinery, pp. 150–158. ISBN: 9781450314626. DOI: 10.1145/2339530.2339556. URL: https://doi.org/10.1145/2339530.2339556.

Lou, Yin, Caruana, Rich, Gehrke, Johannes, and Hooker, Giles (2013). "Accurate Intelligible Models with Pairwise Interactions". In: *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '13. Chicago, Illinois, USA: Association for Computing Machinery, pp. 623–631. ISBN: 9781450321747. DOI: 10.1145/2487575.2487579. URL: https://doi.org/10.1145/2487575.2487579.

Lundberg, Scott M and Lee, Su-In (2017). "A Unified Approach to Interpreting Model Predictions". In: *Advances in Neural Information Processing Systems 30*. Ed. by I. Guyon et al. Curran Associates, Inc., pp. 4765–4774. URL: http://papers.nips.cc/paper/7062-a-unified-approach-to-interpreting-model-predictions.pdf.

Nori, Harsha, Jenkins, Samuel, Koch, Paul, and Caruana, Rich (Sept. 2019). *InterpretML: A Unified Framework for Machine Learning Interpretability*. ArXiv. URL: https://www.microsoft.com/en-us/research/publication/interpretml-a-unified-framework-for-machine-learning-interpretability/.

Nowlan, Steven and Hinton, Geoffrey E (1990). "Evaluation of Adaptive Mixtures of Competing Experts". In: *Advances in Neural Information Processing Systems*. Ed. by R. P. Lippmann, J. Moody, and D. Touretzky. Vol. 3. Morgan-Kaufmann. URL: https://proceedings.neurips.cc/paper/1990/file/432aca3a1e345e339f35a30c8f65edce-Paper.pdf.

Park, Byeonghwa and Bae, Jae Kwon (2015). "Using machine learning algorithms for housing price prediction: The case of Fairfax County, Virginia housing data". In: *Expert Systems with Applications* 42.

Prokhorenkova, Liudmila, Gusev, Gleb, Vorobev, Aleksandr, Dorogush, Anna Veronika, and Gulin, Andrey (2018). "CatBoost: unbiased boosting with categorical features". In: *Advances in Neural Information Processing Systems*. Vol. 31.

Ribeiro, Marco Tulio, Singh, Sameer, and Guestrin, Carlos (2016). ""Why Should I Trust You?": Explaining the Predictions of Any Classifier". In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '16. San Francisco, California, USA: Association for Computing Machinery, pp. 1135–1144. ISBN: 9781450342322. DOI: 10.1145/2939672.2939778. URL: https://doi.org/10.1145/2939672.2939778.

Ridgeway, Greg (Jan. 2005). "Generalized Boosted Models: A Guide to the GBM Package". In: *Compute* 1, pp. 1–12.

Rosen, Sherwin (1974). "Hedonic prices and implicit markets: product differentiation in pure competition". In: *Journal of Political Economy* 82.1, pp. 34–55.

Rudin, Cynthia (May 2019). "Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead". In: *Nature Machine Intelligence* 1, pp. 206–215. DOI: 10.1038/s42256-019-0048-x.

Schapire, Robert E. (1990). "The strength of weak learnability". In: *Machine Learning* 5.2, pp. 197–227. DOI: 10.1023/A:1022648800760.

Schapire, Robert E. and Freund, Yoav (2012). *Boosting: Foundations and Algorithms*. The MIT Press. ISBN: 0262017180.

Schmidhuber, Jürgen (2015). "Deep learning in neural networks: An overview". In: *Neural Networks* 61, pp. 85–117. ISSN: 0893-6080. DOI: https://doi.org/10.1016/j.neunet.2014.09.003. URL: https://www.sciencedirect.com/science/article/pii/S0893608014002135.

Sing, Tien Foo, Yang, Jesse Jingye, and Yu, Shi Ming (2021). "Boosted Tree Ensembles for Artificial Intelligence Based Automated Valuation Models (AI-AVM)". In: *The Journal of Real Estate Finance and Economics* 43.

Späth, Helmuth (1979). "Algorithm 39 Clusterwise linear regression". In: *Computing* 22, pp. 367–373.

Wood, Simon (Jan. 2006). *Generalized Additive Models: An Introduction With R*. Vol. 66, p. 391. ISBN: 9781315370279. DOI: 10.1201/9781315370279.

Zeileis, Achim, Hothorn, Torsten, and Hornik, Kurt (2008). "Model-Based Recursive Partitioning". In: *Journal of Computational and Graphical Statistics* 17.2, pp. 492–514. DOI: 10.1198/106186008X319331. eprint: https://doi.org/10.1198/106186008X319331. URL: https://doi.org/10.1198/106186008X319331.

Zhang, B. (Dec. 2003). "Regression clustering". In: pp. 451–458. ISBN: 0-7695-1978-4. DOI: 10.1109/ICDM.2003.1250952.

A Example of tree stumps

We now present a simple example of how a set of tree stumps together form a Generalized Additive Model. We use the state-of-the-art implementation of gradient boosted trees, XGBoost, to train a model on the synthetic data set considered in Equation 3. Each tree is a tree stump. Figure 9 shows the first nine trees in this sequence. The root nodes are color coded based on which of the three variables x_1, x_2, x_3 that is chosen (by the algorithm) for splitting in each iteration. The resulting shape functions, based solely on these 9 trees, is shown in Figure 10. [2]

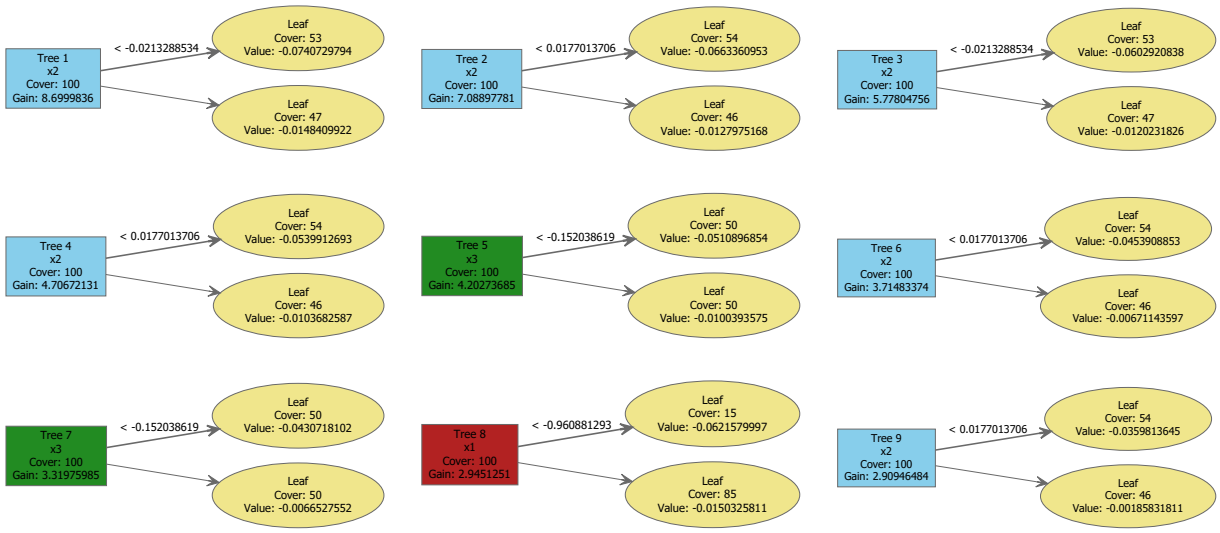

Figure 9: The first nine trees in a gradient boosted tree ensemble with depth = 1 trained on the simulated data set from Equation 3. The root nodes are color coded according to which variable they split on: Red (x_1), blue (x_2), green (x_3).

[2] This visualization is inspired by Cynthia Rudin's lecture notes on *Interpretable Generalized Additive Models via Boosting*. It can be accessed here: `https://users.cs.duke.edu/~cynthia/CourseNotes/AdditiveModelsBoosting.pdf`

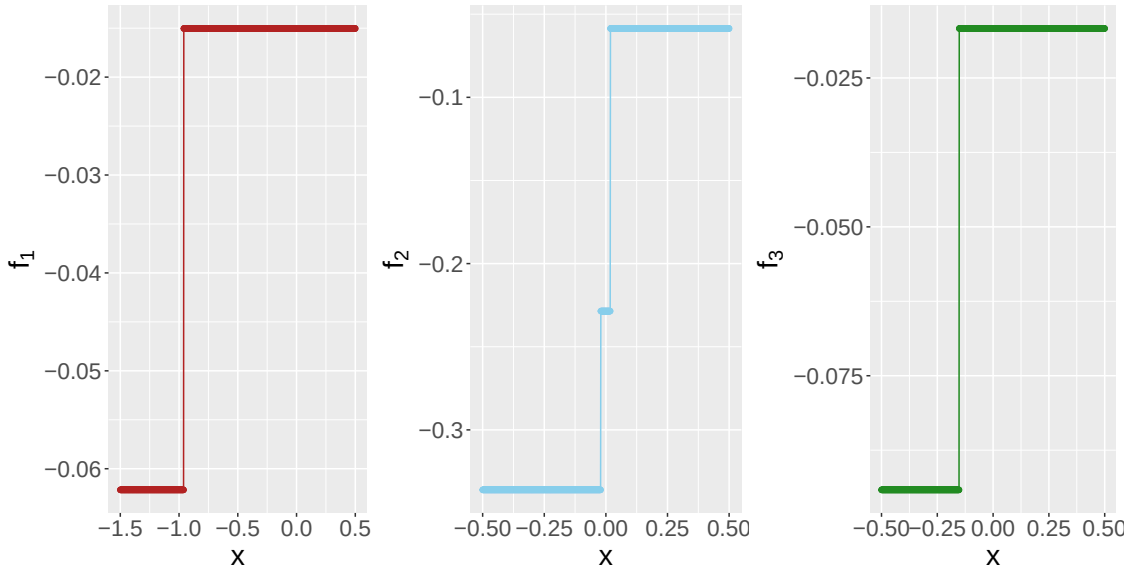

Figure 10: The shape functions f_1, f_2, f_3 based on the nine decision trees presented in Figure 9.

B Partial dependence plots

Consider a data set that is generated from the following scheme:

$$\begin{aligned} x_1 \sim \mathcal{N}(0,1), \quad x_2 \sim \mathcal{N}(0,1), \quad x_3 \sim \mathcal{N}(0,1) \\ y = \left(\frac{1}{2}x_1\right)^2 \cdot \sin(4 \cdot x_2) \cdot \frac{1}{2}|x_3| + \varepsilon, \quad \varepsilon \sim \mathcal{N}(0,1) \end{aligned} \tag{4}$$

This data set quite clearly has interactions, since we multiply instead of add, the three terms. To illustrate the effect of using trees of depth > 1, we train a gradient boosted procedure with $M = 1000$ sequential trees, each with tree depth 3. This is equivalent of including interactions terms, since each tree might first split on x_1 first, x_2 second and x_3 third, for instance. This is illustrated in Figure 11, where the left figure shows the first decision tree in the sequence. The first split is on the covariate x_1, putting every instance with $x_1 \lessapprox 0.99$ in one node and the rest in another node. The right plot shows a partial dependence plot between the prediction $\hat{y}$ and the covariate x_1. We show five lines, representing five of the rows in the training data set. Each line is generated by repeatedly changing x_1 while keeping the other covariates x_2, x_3 fixed, each time using the model to predict. Due to the interactions introduced by having trees with depth > 1, the lines are not parallel. If the model was trained by using tree stumps, we would see completely parallel lines.

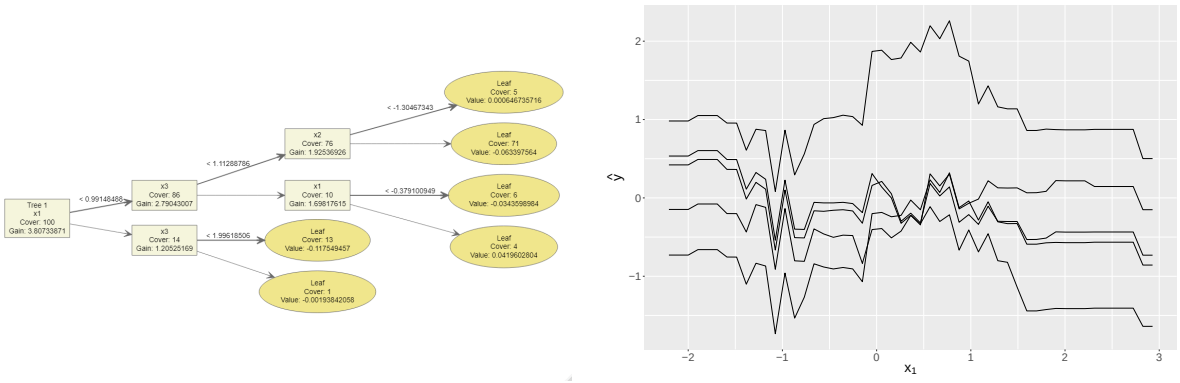

Figure 11: **Left:** A decision tree with depth 3 for a data set with interactions between x_1, x_2, x_3. The text in the square boxes indicate which covariate that is used to split on, while the values above the arrows show the numerical value the split is conducted at. For instance, the first split is done according to the rule $x_1 < 0.99$ such that the data points that satisfy this goes to the upper node and the rest goes to the lower node. **Right:** A partial dependence plot (PDP) after running a gradient boosted trees procedure for $M = 1000$ iterations, each time with trees of tree depth 3. The pdp shows five of the the final predictions $\hat{y}$ plotted against one of the covariates, x_1. Since we allow interactions, the lines will not always be parallel.