Trinkaus, Oliver; Kauermann, Göran

Can machine learning algorithms deliver superior models for rental guides?

AStA Wirtschafts- und Sozialstatistisches Archiv

Provided in Cooperation with:
Springer Nature

Suggested Citation: Trinkaus, Oliver; Kauermann, Göran (2023) : Can machine learning algorithms deliver superior models for rental guides?, AStA Wirtschafts- und Sozialstatistisches Archiv, ISSN 1863-8163, Springer, Berlin, Heidelberg, Vol. 17, Iss. 3, pp. 305-330, https://doi.org/10.1007/s11943-023-00333-x

This Version is available at:
https://hdl.handle.net/10419/313164

AStA Wirtschafts- und Sozialstatistisches Archiv (2023) 17:305–330
https://doi.org/10.1007/s11943-023-00333-x

ORIGINALVERÖFFENTLICHUNG

Can machine learning algorithms deliver superior models for rental guides?

Oliver Trinkaus · Göran Kauermann

Received: 11 April 2023 / Accepted: 13 November 2023 / Published online: 12 December 2023

Abstract In this paper we discuss the use and potential advantages and disadvantages of machine learning driven models in rental guides. Rental guides are a formal legal instrument in Germany for surveying rents of flats in cities and municipalities, which are today based on regression models or simple contingency tables. We discuss if and how modern and timely methods of machine learning outperform existing and established routines. We make use of data from the Munich rental guide and mainly focus on the predictive power of these models. We discuss the "black-box" character making some of these models difficult to interpret and hence challenging for applications in the rental guide context. Still, it is of interest to see how "black-box" models perform with respect to prediction error. Moreover, we study adversarial effects, i.e. we investigate robustness in the sense how corrupted data influence the performance of the prediction models. With the data at hand we show that models with promising predictive performance suffer from being more vulnerable to corruptions than classic linear models including Ridge or Lasso regularization.

Keywords Machine learning · Statistical learning · Regression · Adversarial regression · Mietspiegel · Rental guide

✉ Oliver Trinkaus
EMA-Institut für empirische Marktanalysen, Regensburg, Germany
E-Mail: trinkaus@ema-institut.de

✉ Göran Kauermann
Ludwig-Maximilians-Universität München, München, Germany
E-Mail: Goeran.Kauermann@stat.uni-muenchen.de

1 Introduction

Rental guides for flats are an official instrument in the German rental market, see e.g. Kauermann and Windmann (2016). Based on regular surveys, city councils issue the average rent for flats given the flat's facilities, like floor space, year of construction and facilities such as well equipped kitchen or high standard bathroom. Given available survey data one is interested in constructing a prediction model for the rent per squared meter, given the input variables, which we call features subsequently. Denoting the features as x and the rent per squared meter as y we are interested in finding a good prediction model

$$y = f(x) + \varepsilon \tag{1}$$

where ε is considered as noise, mirroring market price variation. We denote y subsequently also as response variable.

Before discussing the statistical approaches to tackle function $f(x)$ in (1) we want to add some more explanation about rental guides in Germany. These are used as an instrument to infringe on a landlord's constitutional rights to his property (Art. 14 German Constitution). Therefore, a judge, in order to deny a landlord a rent increase, will need a solid base for his judgment. Rental guides aim to fulfill this purpose. This, in turn, seem to exclude "black box models" from the set of instruments that one should use. Moreover, there is a significant amount of vested interest involved in the process of rental guide creation which of course increases the likeliness of data corruption to occur. Hence, careful and detailed guidelines for rental guides are inevitable. With this being said, we can now approach rental guides formally as a prediction model. We also refer to Kauermann and Windmann (2016) and Fitzenberger and Fuchs (2017) for more details.

In times of increased usage of machine learning methods we can consider Eq. (1) as a supervised learning setting. Hence, we may take advantage of the toolbox of available machine learning algorithms to train or estimate a suitable prediction model f. A classical model and in fact the commonly used model in practice (so far) is to build the prediction model f via regression techniques. One approach, developed by Aigner et al. (1993), is to fit a multiplicative-additive regression model in two stages. The more common strategy is to use an additive regression model, as discussed for instance in Fahrmeir et al. (2022). Regression models are one of the legally permitted models for rental guides, besides very simple models based on (contingency) tables, which we do not consider here in this article. Instead, we go beyond the legally permitted models and want to explore more advanced tools of machine learning as introduced, e.g. by James et al. (2017) and Hastie et al. (2017).

Regression models allow for interpretation due to their open box character. In contrast, more complex machine learning makes direct interpretations often difficult. This leads to extended flexibility but ends up with a "black-box" character. Still, one can achieve higher prediction accuracy. The recent developments in machine learning suggest to investigate their potential usage for rental guides. This is the scope of this paper. We use classical regression models including penalized regression and contrast these to regression trees (Breiman 1984) and ensemble models

like averaging and boosting (Freund and Schapire 1999; Breiman 2001; Friedman 2001; Chen and Guestrin 2016) but also neural networks (Goodfellow et al. 2016). A comparison is given in terms of model performance and predictive power.

Besides interpretability, the question of robustness of these models gets in the foreground. We thereby focus on adversarial effects, see e.g. Biggio et al. (2013); Szegedy et al. (2014); Biggio and Roli (2018); Madry et al. (2018) or Tsipras et al. (2019). Adversarial effects are changes of the input variables to a machine learning model that cause the model to make wrong predictions. We use the concept of adversarial risk proposed in Javanmard et al. (2020); Mehrabi et al. (2021) to quantify the robustness of machine learning based rental guides and compare these to adversarial effects in regression models.

The paper is organized as follows: Sect. 2 shortly describes the data at hand. In Sect. 3 we introduce all prediction models, emphasize some differences and explain their essential hyper-parameters. Sect. 4 introduces the notion of standard and adversarial risk which is then applied it to the rental data. The results of our data analysis are given in Sect. 5 and a conclusion follows in Sect. 6.

2 Data and Software

As database we make use of the Munich rental guide data from 2019 containing $n = 3024$ sampled apartments for which we include $p = 19$ selected features Windmann and Kauermann (2019, Table 2.7). The features and their respective description are listed in Table 1. The features are apparently not independent and their Pearson correlation coefficients are visualized in Fig. 1.

For developing and coding we use Python 3, see Van Rossum and Drake (2009). As IDE (Integrated Development Environment) we use Spyder 4.5.0, see Raybaut (2009). For building statistical models we use Statsmodel (Seabold and Perktold 2010) and Scikit (Pedregosa et al. 2011) API's (Application Interface) with its latest versions.

3 Prediction Models

In this section we shorty describe the different types of prediction models used in this paper. A short summary including the models' standard performance hyper-parameters are provided in Table 3.

3.1 Regression Model

Given features $\mathbf{x} = (x_1, \ldots, x_p)$, we predict the rent per square meter via the model

$$f(\mathbf{x}; \hat{\boldsymbol{\beta}}) = \hat{\beta}_0 + \hat{\beta}_1 x_1 + \ldots + \hat{\beta}_p x_p = \hat{\beta}_0 + \sum_{j=1}^{p} x_j \hat{\beta}_j, \tag{2}$$

Table 1 List of features

#	Feature	Content	Min	Max	Freq	Mean
y	netRentSquareMeter	Net rent per square meter	2.88	27.90		11.72
1	lsBySmooth	Function depending on living space and building year using penalized regression splines	9.82	18.00		11.72
2	macroArea	Macro centrality of residential area	0.00	1.00	0: 2017, 1: 1007	
3	wwsIncomplete	Incomplete warm water supply	0.00	1.00	0: 2958, 1: 66	
4	noThermostaticValve	No thermostatic valve	0.00	1.00	0: 2566, 1: 458	
5	incompleteHeatingSupply	Incomplete heating supply	0.00	1.00	0: 2983, 1: 41	
6	underfloorHeating	Underfloor heating	0.00	1.00	0: 2692, 1: 332	
7	openKitchen	Open kitchen	0.00	1.00	0: 2666, 1: 358	
8	modernizedFloor	Modernized floor	0.00	1.00	0: 2715, 1: 309	
9	specialEquipment	Special equipment	0.00	1.00	0: 2794, 1: 230	
10	noTerraceBalcony	Neither terrace nor balcony	0.00	1.00	0: 2405, 1: 619	
11	terraceRoofTerrace	Terrace or roof terrace	0.00	1.00	0: 2664, 1: 360	
12	windowsSingleGlazing	Windows single glazed	0.00	1.00	0: 2923, 1: 101	
13	rearBuilding	Rear building	0.00	1.00	0: 2824, 1: 200	
14	score2Kitchen	Score for kitchen equipment	0.00	3.00	0: 2249, 2: 236, 3: 317, 1: 222	
15	residentialArea	Score for the residential area	1.00	3.00	1: 1663, 2: 1222, 3: 139	
16	buildingType19	Types of buildings	0.00	2.00	0: 2343, 2: 453, 1: 228	
17	houseType19	New building type	0.00	3.00	0: 2240, 3: 487, 2: 130, 1: 167	
18	floorType	Different types of floor equipment	0.00	2.00	2: 2182, 0: 528, 1: 314	

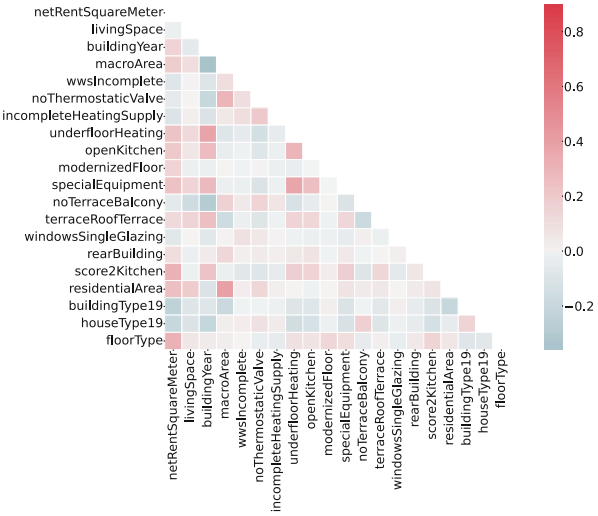

Fig. 1 Pearson correlation coefficients between all selected features

where $\hat{\beta}_0$ denotes the intercept and $\hat{\beta}_0, \ldots, \hat{\beta}_p$ are estimated by least squares method, i.e. by minimizing the residual sum of squares (*RSS*)

$$RSS(\boldsymbol{\beta}) = \sum_{i=1}^{n} \left(y^{(i)} - \mathbf{x}^{(i)} \boldsymbol{\beta} \right)^2, \tag{3}$$

where $\mathbf{x}^{(i)} = (1, x_1^{(i)}, \ldots, x_p^{(i)})$ and $\boldsymbol{\beta} = (\beta_0, \ldots, \beta_p)$, with superscript (i) referring to the observed data and $i = 1, \ldots, n$. We also write $\hat{f}(\mathbf{x}) := f(\mathbf{x}; \hat{\boldsymbol{\beta}})$ and $\hat{\mathbf{y}} := \mathbf{x}\hat{\boldsymbol{\beta}}$ as shorthand. In the application, the model is extended by including non-linearities for the metrical covariates. To be explicit, we replace the linear fit by a spline-based fit using tools extensively described in Wood (2017).

Given our response variable y, our inputs $\mathbf{x}$ and our prediction model $\hat{f}(\mathbf{x})$, the loss function for measuring errors between y and $\hat{f}(\mathbf{x})$ is denoted by $\ell(\mathbf{y}, \hat{f}(\mathbf{x}))$. We use the quadratic loss

$$\ell(\mathbf{y}, \hat{f}(\mathbf{x})) := \left(\mathbf{y} - \hat{f}(\mathbf{x}) \right)^2, \tag{4}$$

for the applications in this paper.

3.2 Regression Trees

Basically all tree-based methods arise from partitioning the feature space into a set of hyperrectangles, and then fit a simple model in each of the hyperrectangles which are summed up as an ensemble of subtrees to give a final prediction model. Amongst several types of algorithms, here we focus on the most common CART-algorithm (Classification and Regression Trees). To be specific, regression trees, following Breiman (1984) and Hastie et al. (2017), are built by dividing the fea-

ture space $\mathbb{R}^p$ into K distinct and non-overlapping hyperrectangles $R_1, \dots R_K$ that minimize the *RSS* given by

$$\sum_{k=1}^{K} \sum_{l \in R_k} \left(y^{(l)} - \hat{y}_{R_k} \right)^2 \tag{5}$$

where $\hat{y}_{R_k}$ is the mean response for the observations within the kth hyperrectangle, i.e. $\hat{y}_{R_k} = \sum_{x^{(l)} \in R_k} y^{(l)} / |R_k|$. For computational runtime reasons one takes a top-down, greedy approach that is known as recursive binary splitting, see Breiman (1984). In order to perform recursive binary splitting, the algorithm needs a starting point. Therefore, we first need to find a feature x_k and a cutpoint s such that splitting the feature space into the regions $\{x \mid x_k < s\}$ and $\{x \mid x_k \geq s\}$ leads to the greatest possible reduction in *RSS*. We consider all features $x_1, \dots x_p$, and all possible values of the cutpoint s for each of the features, and then choose the feature and cutpoint such that the resulting tree has the lowest *RSS*. Once found, this feature is called the *root* of the tree. More precisely, for any $\tilde{p}$ and s, we define the pair of half-planes

$$R_1(\tilde{p}, s) := \{x \mid x_{\tilde{p}} < s\} \text{ and } R_2(\tilde{p}, s) := \{x \mid x_{\tilde{p}} \geq s\}, \tag{6}$$

and we seek the value of $\tilde{p}$ and s that minimize the equation

$$\sum_{i \mid x^{(i)} \in R_1(\tilde{p}, s)} \left(y^{(i)} - \hat{y}_{R_1} \right)^2 + \sum_{i \mid x^{(i)} \in R_2(\tilde{p}, s)} \left(y^{(i)} - \hat{y}_{R_2} \right)^2, \tag{7}$$

where $\hat{y}_{R_1}, \hat{y}_{R_2}$ are the mean response for the observations in $R_1(\tilde{p}, s)$ and $R_2(\tilde{p}, s)$, respectively. The splitting procedure is now continued on each half-plane and the splitting process is continued until a stopping criterion is reached. For instance, we may continue until no region contains more than five observations. This procedure comes often with a very complex resulting tree, which is likely to overfit. Therefore, a procedure called cost-complexity tree pruning, suggested by Breiman (1984), is obtained by removing a sequence of subtrees. This procedure is applied after fully growing the tree and is described as follows: A large tree T_0 is grown and the splitting process only is stopped when some minimum node size (say 5) is reached. Then, this tree is "pruned" by finding a subtree $T \subsetneq T_0$. Let $|T|$ denote the number of terminal nodes in tree T and let $RSS(T)$ be the residual sum of squares given in Eq. (3) for tree T. We define the criterion

$$C_\alpha(T) = RSS(T) + \alpha |T| .$$

For given α we aim to find the subtree $T_\alpha \subseteq T_0$ which minimizes $C_\alpha(T)$. The tuning parameter $\alpha \geq 0$ governs the tradeoff between the tree size and its goodness of fit to the data. Large values of α result in smaller tress T_α, and conversely for smaller values of α. As the notation suggests, with $\alpha = 0$ the solution is the full tree T_0. For each α one can show that there is a unique smallest subtree T_α that minimizes $C_\alpha(T)$. To find T_α we use weakest link pruning, that is we successively

collapse the internal node that produces the smallest per-node increase in *RSS*, and continue until we produce the single-node (root) tree. This gives a (finite) sequence of subtrees, and one can show this sequence must contain T_α, see Breiman (1984); Hastie et al. (2017). Estimation of α is achieved using cross-validation. The final prediction model is then contained in the final tree $T_{\hat{\alpha}}$.

3.3 Random Forests

If we consider a regression tree as our statistical model, **bagging** regression trees, as proposed by Breiman (1996), is an aggregation of B bootstrap samples

$$\left(\mathbf{x}^{*1}, \mathbf{y}^{*1}\right), \left(\mathbf{x}^{*2}\mathbf{y}^{*2}\right), \ldots, \left(\mathbf{x}^{*B}, \mathbf{y}^{*B}\right),$$

sampled randomly from the original training data with replacement, having all the same size. Then for each bootstrap sample $\mathbf{x}^{*b}$, a corresponding bootstrap replication regression tree T^{*b} is grown, for all $b = 1, \ldots, B$. Together all B regression trees $T^{*1}, T^{*2}, \ldots T^{*B}$ are summed up to fit a final prediction model $\hat{f}_{\text{bag}}(\mathbf{x})$. This model is given by the arithmetic mean of the predictions obtained from the B trees $T^{*1}, T^{*2}, \ldots T^{*B}$, i.e.

$$\hat{f}_{\text{bag}}^{B}(\mathbf{x}) = \frac{1}{B}\sum_{b=1}^{B} T^{*b}(\mathbf{x}). \tag{8}$$

Random forests, see e.g. Breiman (2001) and James et al. (2017), are basically an extension of bagging. It fits regression trees $\hat{T}^{*b}$ as base regressors on each of all the $b = 1, \ldots, B$ bootstraps. However, when splitting each node during the construction of a tree (as described in Sect. 3.2), the best split is found from a random subset of the features (see hyper-parameter max_features in Table 3). Hence, instead of considering all p features $x_1, \ldots, x_p$ a split is considered from a random sample of $m < p$ features chosen as split candidates. The final prediction model is then given by

$$\hat{f}_{\text{rf}}^{B}(\mathbf{x}) = \frac{1}{B}\sum_{b=1}^{B} \hat{T}^{*b}(\mathbf{x}), \tag{9}$$

for $b = 1, \ldots, B$.

With these two variations of randomness, we decrease the prediction error of random forest prediction models and increase prediction performance. This will also be visible on our experiments later in the paper.

3.4 Boosting

A weak learner is defined to be a regression model that is only slightly correlated with the true prediction. Boosting, originally proposed by Schapire (1990) for classification tasks, answers the question, if a set of weak learners can be put together

to form a strong learner. As in the above supervised learning problems, the goal is to find a function $\hat{f}$ that best predicts the output variable $\mathbf{y}$ from the input variables $x_1, \ldots, x_p$. Let $\ell(\mathbf{y}, f(\mathbf{x}))$ be the ℓ_2-loss. Then, we want to minimize this loss formally by

$$\hat{f} = \text{argmin}_{f(\mathbf{x})} \mathbb{E}_{\mathbf{y},\mathbf{x}}[\ell(\mathbf{y}, f(\mathbf{x}))]. \tag{10}$$

The idea of boosting is to predict the response y by fitting an additive model

$$\hat{f}(\mathbf{x}) = \sum_{m=1}^{M} \gamma_m t_m(\mathbf{x}) \tag{11}$$

for $M \in \mathbb{N}$, $\gamma_m \in \mathbb{R}$ as weight and $t_m(x) \in T$, where T denotes the class of base or weak learners, e.g. the class of regression trees. This is optimized in a forward stage-wise manner, meaning at each stage, one fixes the errors of its predecessor.

3.4.1 Steepest Descent

Unfortunately, choosing the best function t at each step for an arbitrary loss function ℓ is a computational infeasible optimization problem in general. Therefore, we restrict our approach to a simplified version of the problem. There are basically two methods: The very first algorithm is called Adaboost using a specific exponential loss function and small stumps which are usually smaller than the trees build with gradient tree boosting (gradie), as explained in the next section, see also Freund and Schapire (1996). In this paper we use a more flexible way of updating the model to its predecessors called *gradient descent*. It is a first-order iterative optimization algorithm for finding a local minimum of a differentiable convex loss function going in the opposite direction of the gradient at a point. This is the direction of the steepest descent (Cauchy 1847), which is given by the negative gradient $-\mathbf{g}(f)$ of a function f. The gradient $\mathbf{g}(f)$ of a real-valued, p-dimensional function f is defined as

$$\mathbf{g}(f) := \frac{\partial f}{\partial x_1} e_1 + \cdots + \frac{\partial f}{\partial x_p} e_p = \begin{pmatrix} \frac{\partial f}{\partial x_1} \\ \vdots \\ \frac{\partial f}{\partial x_p} \end{pmatrix}, \tag{12}$$

where the $e_1, \ldots, e_p$ denoting the unit vectors.

3.4.2 Gradient Tree Boosting

Gradient Tree Boosting (Breiman 1997; Friedman 2001) uses regression tress of fixed size as base learners. It specializes the approach above to the case where the base learner $t_m(\mathbf{x})$ is a J_m-terminal leaf regression tree. More precisely, let F be the set of regression trees. Each tree comes with a respective partition of the feature space R_j, $j = 1, 2, \ldots, J$ induced by the terminal node of the tree. In this case, at

the mth step one fits a regression tree $f_m(\mathbf{x})$ to the previous pseudo-residuals using steepest descent by

$$r_m^{(i)} := -g_{f_{m-1}}^{(i)} = -\left[\frac{\partial \ell(y^{(i)}, f(\mathbf{x}^{(i)}))}{\partial f(\mathbf{x}^{(i)})}\right]_{f(\mathbf{x}^{(i)})=f_{m-1}(\mathbf{x}^{(i)})}, \tag{13}$$

where f_{m-1} represents the combined prediction from the ensemble of trees up to the $(m-1)$th iteration in the gradient boosting process. These pseudo-residuals forming a new data set $\{(\mathbf{x}^{(i)}, r_m^{(i)})\}_{i=1}^n$ which t_m gets fed with. Now, the output $t_m(\mathbf{x})$ for input x can be written as the sum:

$$f_m(\mathbf{x}) = \sum_{j=1}^{J_m} b_{jm} \mathbf{1}_{R_{jm}}(\mathbf{x}), \tag{14}$$

where $b_{jm} = \bar{y}_{jm}$, the mean of the $y_{jm}^{(i)}$ predicted in region R_{jm} and $\mathbf{1}_{R_{jm}}(x)$ denotes the indicator function. We optimize this expression and replace the b_{jm}'s by calculating the one-dimensional γ_{jm}'s in each of the trees regions R_{jm}. Hence we write

$$\begin{aligned} \hat{f}_m(\mathbf{x}) &= \hat{f}_{m-1}(\mathbf{x}) + \sum_{j=1}^{J_m} \gamma_{jm} \mathbf{1}_{R_{jm}}(\mathbf{x}), \\ \gamma_{jm} &= \text{argmin}_\gamma \sum_{x^{(i)} \in R_{jm}} \frac{1}{2}\left(y^{(i)} - \left(\hat{f}_{m-1}(\mathbf{x}^{(i)}) + \gamma\right)\right)^2. \end{aligned} \tag{15}$$

To control overfitting we take a sensible amount of trees (parameter: n_estimators) and a further shrinkage by ν, the *learning rate* (parameter: learning_rate), which can be plugged into the update rule (15) as follows:

$$\hat{f}_m(\mathbf{x}) = \hat{f}_{m-1}(\mathbf{x}) + \nu \cdot \sum_{j=1}^{J_m} \gamma_{jm} \mathbf{1}_{R_{jm}}(\mathbf{x}), \quad 0 < \nu \leq 1. \tag{16}$$

3.4.3 Stochastic Gradient and Extreme Boosting

Another way to apply gradient boosting is to fit the trees only on subsamples. This is called Stochastic Gradient Boosting (Friedman 2001). The size of each tree can be controlled either by setting the tree depth via max_depth or by setting the number of leaf nodes via max_leaf_nodes, see Table 3.

In this paper we use an implementation of a gradient tree boosting algorithm (gradie) as described in (Pedregosa et al. 2011). We further use extreme-boosting(xgbreg) (Chen and Guestrin 2016) which uses a second order Taylor approximation in the loss function to weight the leafs inside of a tree, see Eq. (31) in Appendix. The idea was proposed by Friedman et al. (2000). For parameter details see Table 3.

3.5 Nearest Neighbours

The principle behind the k-Nearest-Neighbour method (KNN) used for regression, see Altman (1992), is to find k observations that are closest in distance to the new point $\mathbf{x}^{(0)}$, denoted by $\mathcal{N}^{(0)}$. Then $f(\mathbf{x}^{(0)})$ is estimated using the average of all the training responses in $\mathcal{N}^{(0)}$, namely

$$\hat{f}(\mathbf{x}^{(0)}) = \frac{\sum_{x^{(i)} \in \mathcal{N}^{(0)}} y^{(i)}}{\left|\mathcal{N}^{(0)}\right|}. \tag{17}$$

The number of samples can be a user-defined constant (K-nearest neighbour learning), or vary based on the local density of points (radius-based neighbour learning). The distance can, in general, be any metric measure: standard Euclidean distance is the most common choice. Sometimes it make sense to assign more weight to the nearer neighbours. A common technique to achieve this is to define weights by the inverse of the distance between $\mathbf{x}^{(0)}$ and a neighbour $\mathbf{x}^{(k)}$, doing so for all $\mathbf{x}^{(k)} \in \mathcal{N}^{(0)}$, see also Cunningham and Delany (2020).

Following Pedregosa et al. (2011), the problem of finding the right nearest neighbour algorithm relies on the size of the sample data and the feature space. For small data sets in both directions, the sample size and the features space, respectively, one usually uses the brute-force algorithm. It computes the distance of all pairs of points in the data set. In our experiment we use this method, since the data-set is relatively small. For details on the hyperparameters see Table 3. As the number of data grows, calculating all these distances is computational infeasible. Therefore a method called *K-D-tree* can be applied, see Mehlhorn (1988); Pedregosa et al. (2011); Bentley (1975) for details.

3.6 Neural Networks

In this paper we focus on multilayer perceptrons (mlpreg) (Goodfellow et al. 2016) as a neural network regression model. Since the data flow only in forward direction, it is also known as feed-forward neural network. This network defines a mapping $\hat{y} = f(\mathbf{x}; \hat{\boldsymbol{\beta}})$ with $\hat{y}$ as predictor for given input variables $\mathbf{x}$, where the values of the parameter $\boldsymbol{\beta}$ are learned (estimated) from the data. Typically, one aims to minimize the squared prediction error $E((\hat{y} - y)^2)$. Such a network usually is represented by a composition of many different functions. For example we might have three functions $f^{[1]}$, $f^{[2]}$, and $f^{[3]}$ connected to form $f(\mathbf{x}) = f^{[3]}(f^{[2]}(f^{[1]}(x)))$. The function $f^{[1]}$ is called first layer, $f^{[2]}$ is called the second layer and so on. The overall length of the chain gives the depth of the model. The final layer is called the output layer and the layers before the final layer are so-called hidden layers. The hidden layer functions themselves are multivariate but simple in their structure. They incorporate a weighted sum of the input combined with an activation function. One can rewrite the j-th component of the functions as $f_j^{[k]}(\mathbf{x}^{[k-1]}; \mathbf{w}_{k,j}, b_{k,j}) = \phi(\mathbf{x}^{[k-1]^T} \mathbf{w}_{k,j} + b_{k,j})$, where $\mathbf{x}^{[k-1]}$ is the (multivariate) output of the previous layer with $\mathbf{x}^{[0]} = \mathbf{x}$. The weights $w_{k,j}$ and the intercept $b_{k,j}$(the so-called bias) for $k = 1, 2, \ldots$ are the parameters, which need to be determined data driven. The set

of all parameters defines β and leads to the trained model $f(\mathbf{x}; \hat{\boldsymbol{\beta}})$. The function $\phi(.)$ is called the activation function, which is known. With this setup we can now find optimal weights such the prediction error gets minimized. To do so one can use cross-validation so that the model is trained on one part of the data and tested on the other part. We do not want to get in more technical details here, since the field of neural networks became so massive with numerous introductory literature. We refer to Goodfellow et al. (2016) or Hastie et al. (2017) for more details, or to Borth et al. (2023).

4 Performance Measures

After having introduced the different models which will be used for constructing rental guides, we need to define the performance measures applied subsequently. We will thereby look not only on the prediction error but also on the robustness, utilizing the concept of standard and adversarial risk introduced in Javanmard et al. (2020); Mehrabi et al. (2021), just in case of the ℓ_2-Loss.

4.1 Standard Risk

We define the **standard risk**

$$\mathsf{SR}(f) := \sqrt{\mathbb{E}_{y,x}\left[(y - f(\mathbf{x}))^2\right]} \tag{18}$$

to be the prediction loss of an estimator f on an (uncorrupted) test data point $\mathbf{x}$ and where $(\mathbf{x}, y) \sim \mathcal{P}$ is drawn from some common law $\mathcal{P}$. An empirical estimate for SR is given by

$$\widehat{\mathsf{SR}}(\hat{f}) = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(y^{(i)} - \hat{f}_{-i}\left(x^{(i)}\right)\right)^2}, \tag{19}$$

where $\hat{f}_{-i}$ is the prediction model fitted on data omitting the ith observation.

4.2 Adversarial Risk

An adversarial attacked model $\hat{f}_{\mathcal{S}}$ is a prediction model $\hat{f}$ just with a corruption of the data coming from predefined perturbation sets $\mathcal{S} := \{\delta \in \mathbb{R}^p : \|\delta\|_{\ell_2} \leq \epsilon\} \subset \mathbb{R}^p$. The adversary has the power of ϵ perturbing each data point $x^{(i)}$ by adding an element of $\mathcal{S}$. The main idea of assessing the adversarial risk is to measure the robustness of the model. We quantify how much does the prediction change if a single or multiple input variables are false, i.e. perturbed from its original value.

We define the **adversarial risk** as

$$\mathsf{AR}(f, \mathcal{S}) := \sqrt{\mathbb{E}_{y,x}\left[\max_{\boldsymbol{\delta} \in \boldsymbol{\mathcal{S}}} (y - f(\mathbf{x} + \boldsymbol{\delta}))^2\right]} \tag{20}$$

which is the expected prediction loss of a predictor f on an adversarially corrupted data point according to some attack or mistake model. Stated differently, the adversarial risk measures how adverse the predictor f can perform in prediction when it is fed with adversarially corrupted data.

To motivate the adversarial attack in more detail, we first have to consider our two types of features, namely metrically scaled and binary (categorical) variables. For noising a metric feature x_j, say, a value $\delta \in \mathbb{R}$ is added to x_j in such a way, that the sum $(x_j + \delta)$ is still is inside the range of values of x_j. This means the sum ranges between the minimum and the maximum realization of x_j. For binary features x_l this goes analogously, except that all possible values are 0 or 1. Hence, in binary case, the perturbation is achieved by $x_l + (-1)^{x_l}$. This defines the perturbation set $\mathcal{S}$. We additionally restrict the number of perturbated variables and define the sets $\mathcal{S}_k \subset \mathcal{S}$ such that each element of $\mathcal{S}_k$ has exactly k elements which are unequal to zero. In other words, in $\mathcal{S}_k$ we perturb exactly k features and leave the remaining features unchanged. This leads to the adversarial risk

$$\mathsf{AR}(f, k) := \sqrt{\mathbb{E}_{y,x}\left[\max_{\boldsymbol{\delta} \in \mathcal{S}_k} (y - f(\mathbf{x} + \boldsymbol{\delta}))^2\right]} \tag{21}$$

Apparently, $\mathsf{AR}(f, \mathcal{S}) = \max_{k=1,\dots p} \mathsf{AR}(f, k)$.

We estimate (21) through

$$\widehat{\mathsf{AR}}(\hat{f}, k) := \sqrt{\max_{\boldsymbol{\delta} \in \mathcal{S}_k} \left(\frac{1}{n} \sum_{i=1}^{n} \left(y^{(i)} - \hat{f}_{-i}(x^{(i)} + \delta\right)^2\right)} \tag{22}$$

where $\hat{f}_{-i}$ is the prediction model fitted on data excluding the i-th observation.

5 Results

In this section we compare the performance of the models introduced above with respect to the standard and adversarial risk, respectively. We abbreviate the different models as follows: linear regression model (linear), decision tree (decisi), bagging tree (baggin), random forest (random), gradient boosting (gradie), extreme-boosting tree (xgbreg), k-neighbours ($k$neigh), multi-layer perceptron (mlpreg).

5.1 Standard Risk

The standard risk for the different models is shown in Fig. 2, see also Table 2. For the standard risk we have in descending direction decisi (2.70), baggin (2.61),

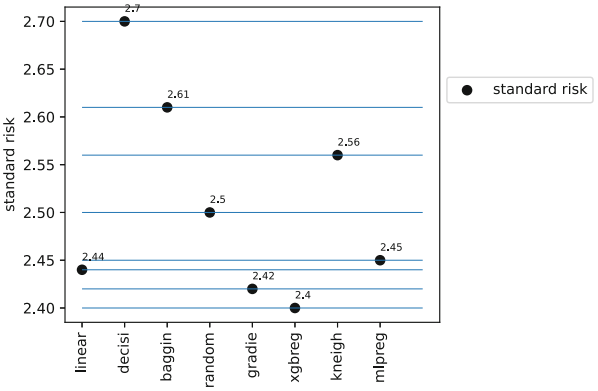

Fig. 2 Standard risk (y-axis) of all models (x-axis)

Table 2 SR, AR and ρ

Model	SR	AR	ρ
linear	2.44	3.22	1.32
decisi	2.70	3.62	1.34
baggin	2.61	4.17	1.59
random	2.50	3.10	1.23
gradie	2.42	3.26	1.35
xgbreg	2.40	3.63	1.51
kneigh	2.56	3.24	1.27
mlpreg	2.45	3.16	1.29

$k$neigh (2.56), random (2.50), mlpreg (2.45), linear (2.44), gradie (2.42), xgbreg (2.40). Hence, the best results are obtained by the boosting approaches xgbreg and gradie followed by the classical linear and neural network mlpreg. Then we have $k$neigh and random with almost equal results. At the end we have baggin and decisi. Both of the last two regression models are also almost equal since baggin is fitted using decisi as base regressors. It is reassuring to see that the "open box" regression models performs quite well.

5.2 Adversarial Risk

5.2.1 Model Robustness (global Robustness)

We describe the results for $\boldsymbol{\delta} \in \mathcal{S}_k$ with $k = 1$. In other words, a single input variable is perturbed. The highest adversarial risk is obtained by baggin (4,17), followed by xgbreg (3,63). This is followed by decisi (3,62), gradie (3,26), $k$neigh (3,24), linear (3,22). The lowest adversarial risks are given by mlpreg (3,16) and random (3,10), see Fig. 3. Interestingly enough, linear regression is performing well, while some models with small standard risk suffer from larger adversarial risk.

To get a sense of the robustness of the algorithms against adversarial attacks, we also calculate the robustness measure $\rho := \mathsf{AR}/\mathsf{SR} \in \mathbb{R}$, the ratio between the

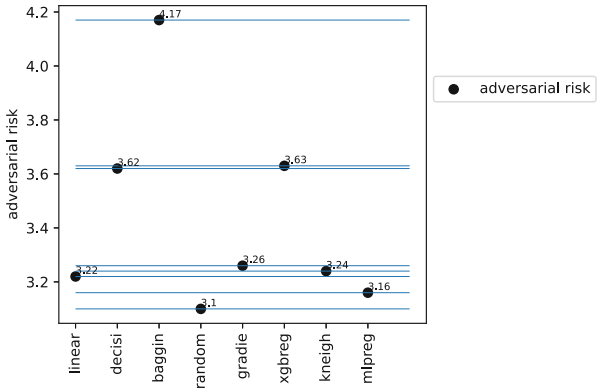

Fig. 3 Adversarial risk (y-axis) for $\mathcal{S}_1$ of all models (x-axis)

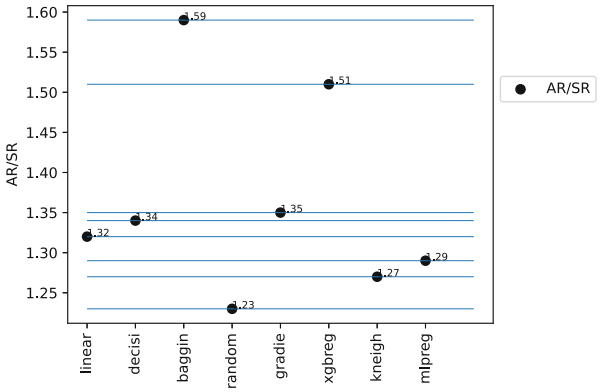

Fig. 4 Ratio ρ (y-axis) of all models (x-axis)

adversarial risk and the standard risk. Usually we expect the adversarial risk to be higher than the standard risk for corrupted data. This means that we expect ρ to be always greater or a least equal 1.

For ρ in descending direction we have baggin (1.59), xgbreg (1.51), gradie (1.35), decisi (1.34), linear (1.32). The models mlpreg (1.29), $k$neigh (1.27) showed equal robustness. Finally random (1.23) performend the best in terms of robustness, see Fig. 4 and Table 2.

5.2.2 Feature Robustness (local Robustness)

We can take a look at the adversarial risk and see how perturbing the different features contribute to the adversarial risk. That is we look at the quantities

$$\mathsf{AR}(f, \boldsymbol{\delta}) := \sqrt{\mathbb{E}_{y,x}\left[(y - f(\mathbf{x} + \boldsymbol{\delta}))^2\right]} \tag{23}$$

for all $\boldsymbol{\delta} \in \mathcal{S}_k$. This helps to specify which feature causes unrobustness of the prediction. We do this for every feature and every learning model. This approach can also serve as a robust feature importance measure. Not surprisingly at all, we see that the different features have different effects on the models' adversarial risk,

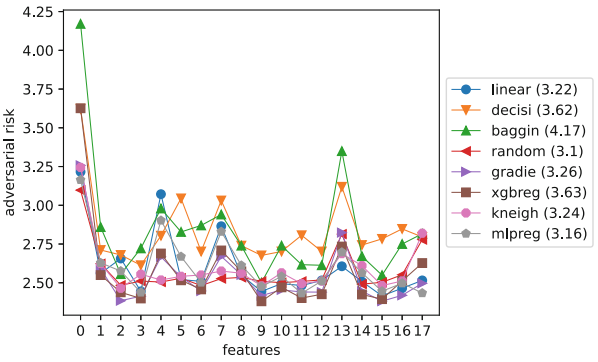

Fig. 5 Adversarial risk of all models for $\boldsymbol{\delta} \in \mathcal{S}_1$

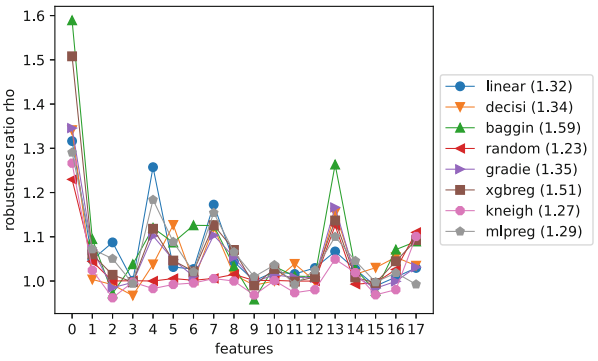

Fig. 6 Robustness of all models for $\boldsymbol{\delta} \in \mathcal{S}_1$

see Fig. 5. Also we can observe that even if one model performs better in terms of a lower adversarial risk, it may have a higher unrobustness measure ρ, compare, for example, $k$neigh and mlpreg in Figs. 5, 6 and Table 2.

Our experiment also shows that in total, there is no model which outperforms all the other models in terms of standard and adversarial risk.

6 Conclusions

This paper studies the use and performance of machine learning regression models when used for rental guides. We also study the models' behaviour under adversarial data corruption using data from the Munich rental guide. In Sect. 5 we compared different machine learning regression models which seem to suite most for the case of explaining rental prices for flats in German Rental Guides. Even if boosting algorithms showed their equal prediction performance compared to the classic prediction models like linear regression, the black-box character still remains: Rental Guides need a differentiated and reliable (robust) information about which criteria are influencing the net rental price in both, the positive and negative direction.

In Sects. 5.1 and 5.2 we showed the standard and adversarial performance of the given models. In Sects. 4.2 and 5.2.1 we addressed adversarial regression by building prediction models on top of corrupted data. We also compared the (local) standard and adversarial risk of features which are highly influential of a model's

performance (see Fig. 5). This means that a model $\hat{f}$ may have a lower adversarial risk than another model $\hat{f}'$ in terms of its predictive power. But, we found that $\hat{f}'$ may possibly be more robust than $\hat{f}$ by looking at the robustness measure $\rho = \mathsf{AR}/\mathsf{SR}$, see for example, $k$neigh and mlpreg in Figs. 5 and 6 and Table 2. Local examples, meaning a feature-wise comparison can be found in Table 4. For example within the feature "specialEquipment" baggin shows a higher adversarial risk but lower robustness than xgbreg.

Concretely, in terms of both, the risk measures SR and AR on the one hand, and the robustness measure ρ on the other hand, random, mlpreg, linear, and gradie showed good predictive power with low adversarial risk and high robustness against adversarial attacks, see Fig. 5. The model $k$neigh showed an overall moderate performance and xgbreg showed a high adversarial risk.

Both models, decisi and baggin showed a higher standard and adversarial risk than their competitors, but decisi has a similar robustness measure compared to xgbreg. The worst performance is obtained by baggin with second highest standard risk and highest adversarial risk resulting in the highest robustness measure. Additionally, it should be mentioned that several features being highly vulnerable when considered within the models baggin and decisi, see Table 2.

As a résumé, we see the resulting volatility of the models applied to (slightly) corrupted data as explained in Sect. 5.2.2. This contradicts both demands for prediction models being explained well on the one hand, and robust on the other hand. It is therefore recommended to search for a model minimizing the trade-off between predictive performance and adversarial risk. This helps to reduce the danger of generating machine learning regression models with misleading explanation. Therefore, we conclude that the best way to overcome this issue, is to use classic regression models. classic regression models may be supported by machine-learning tools to make them more effective. But, machine learning models should not be used as a replacement for classic regression models in the rental guide context.

7 Appendix

7.1 Tables

Table 3 Overview of all used models and essential performance hyper-parameters settings

Linear Regression	Linear models	Ordinary least squares Linear Regression	fit_intercept: Whether to calculate the intercept for this model. If set to False, no intercept will be used in calculations (i.e. data is expected to be centered), data type = bool, value = True, default value = True; normalize: This parameter is ignored when fit_intercept is set to False. If True, the regressors X will be normalized before regression by subtracting the mean and dividing by the l2-norm, data typpe = bool, value = False, default value = False, copy_X: Ift True, X will bie copied, else, it may be overwritten, value = True, default value = False; n_jobs: The number of jobs to use for the computation, data type = int, value = None, default value = none;
Nearest Neighbour	Neighbours	Regression based on k-nearest neighbours.	n_neighbours: Number of neighbours to use by default for kneighbours queries; data type = int; value = 25 (default value = 5), weights: Weight function used in prediction; value = 'uniform', 'distance' or callable, value = 'uniform' (default value = 'uniform'), algorithm: Algorithm used to compute the nearest neighbours; values = 'auto', 'ball_tree', 'kd_tree', 'brute', value set = 'brute' (default value = 'auto'); leaf_size: Leaf size passed to BallTree or KD tree. This can affect the speed of the construction and query, as well as the memory required to store the tree. The optimal value depends on the nature of the problem. When p = 1, this is equivalent to using manhattan_distance (11), and euclidean_distance (l2) for p = 2. For arbitrary p, minkowski_distance (l_p) is used; value = 30 (default value = 30), p: Power parameter for the Minkowski metric; data type = int; value = 2, default value = 2;
Trees	Tree	Regression based on splitting the predictor space via recursive binary splitting	criterion: The function to measure the quality of a split, data type = string, values = "squared_error", "friedman_mse", "absolute_error", "poisson", value = 'mse', default value = 'mse', splitter: The strategy used to choose the split at each node, data type = string, values = "best", "random", value set = 'best', default = 'best'; max_depth: The maximum depth of the tree. If None, then nodes are expanded until all leaves are pure or until all leaves contain less than min_samples_split samples, data type = int, value = None, default value = None, min_samples_split: the minimum number of samples required to split an internal node, data type = int or float, value = 2, default value = 2, min_samples_leaf: the minimum number of samples required to be at a leaf node; data type = int or float, value = 1, default value = 1;

Table 3 (Continued)

Random Forest	Ensemble	Meta estimator that fits a number of regression trees (ensemble) on various sub-samples of the dataset	n_estimators: The number of trees in the forest, data type = int, value = 25, default value = 100; criterion: The function to measure the quality of a split, data type = string, value = "squared_error", "absolute_error", "poisson", default value = "squared_error"; max_depthint: The maximum depth of the tree. If None, then nodes are expanded until all leaves are pure or until all leaves contain less than min_samples_split samples, data type = int, value = None, default value = None; min_samples_split: The minimum number of samples required to split an internal node, data type = int or float, value = 2, default value = 2; min_samples_leaf: The minimum number of samples required to be at a leaf node. A split point at any depth will only be considered if it leaves at least min_samples_leaf training samples in each of the left and right branches, data type = int or float, value = 25, default value = 1; max_features: The number of features to consider when looking for the best split, data type = string, int or float, values = "sqrt", "log2", None, value = 1.0, default value = 1.0; max_leaf_nodes: Grow trees with max_leaf_nodes in best-first fashion. Best nodes are defined as relative reduction in impurity. If None then unlimited number of leaf nodes, data type = int, value = None, default value = None. bootstrap: Whether samples are drawn with replacement. If False, sampling without replacement is performed, data type = bool, value = True, default value = True; oob_score: Whether to use out-of-bag samples to estimate the generalization error. Only available if bootstrap = True, value = bool, default value = False; ccp_alpha: Complexity parameter used for Minimal Cost-Complexity Pruning, data type = non-negative float, value = 0.0, default value = 0.0;
Gradient Boosting	Ensemble	Regression trees fit on the negative gradient of the given loss function.	loss: 'squared_error', 'absolute_error', 'huber', 'quantile', default = 'squared_error', Loss function to be optimized; learning_rate: float, default = 0.1. Learning rate shrinks the contribution of each tree, There is a trade-off between learning_rate and n_estimators. n_estimators: int, default = 100. The number of boosting stages to perform. subsample: float, default = 1.0. criterion: 'friedman_mse', 'squared_error', 'mse', default = 'friedman_mse'. The function to measure the quality of a split. min_samples_split: int or float, default = 2. The minimum number of samples required to split an internal node. min_samples_leaf: int or float, default = 1. The minimum number of samples required to be at a leaf node;
Extrem Boosting	Ensemble	Gradient boosting framework that uses tree based learning algorithms	n_estimators: The number of trees, data type = int, value = 250, default value = 100, learning_rate: Step size shrinkage used in update to prevents overfitting, data type = float in range [0,1], value = 0.07, default value = 0.3; min_split_loss or gamma: Minimum loss reduction required to make a further partition on a leaf node of the tree. The larger gamma is, the more conservative the algorithm will be, data type = int, value = 0, default value = 0; max_depth: Maximum depth of a tree. Increasing this value will make the model more complex and more likely to overfit, data type = float, value = 3, default value = 6, min_child_weight: Minimum sum of instance weight (hessian) needed in a child, data type = float, value = 1.5, default value = 1; reg_alpha: L1 regularization term on weights, data type = float, value = 0.01, default value = 0, reg_lambda: L2 regularization term on weights, data_type = float, value = 0.45, default value = 1, subsample: Subsample ratio of the training instances. Setting it to 0.5 means that XGBoost would randomly sample half of the training data prior to growing trees, data type = float, value = 0.8, default value = 1.0; colsample_bytree: The subsample ratio of columns when constructing each tree, data type = float, value = 0 .4, default value = 1; gamma: Minimum loss reduction required to make a further partition on a leaf node of the tree: data type = float, value = 0, default_value = 0;

Table 3 (Continued)

Bagging	Ensemble	Meta-estimator that fits on random subsets of the original dataset and then aggregate their individual prediction	base_estimator: The base estimator to fit on random subsets of the dataset. If None, then the base estimator is a DecisionTreeRegressor; value = None, n_estimators: The number of base estimators in the ensemble; data type = int, value = 25; default value = 10, max_samples: The number of samples to draw from X to train each base estimator; data type = int or float, value = 1.0, default value = 1.0, max_features: The number of features to draw from X to train each base estimator, data type = int or float, value = 1.0; default value = 1.0,bootstrap: Whether samples are drawn with replacement. If False, sampling without replacement is performed, data type = bool, value = True, default value = True; oob_score: Whether to use out-of-bag samples to estimate the generalization error. Only available if bootstrap = True, value = bool, default value = False;
Multi-layer Perceptron	Neural network	Multi-layer Perceptron regressor optimizes the squared error using LBFGS or stochastic gradient descent	hidden_layer_sizes:tuple, length = n_layers - 2, default = (100,). The ith element represents the number of neurons in the ith hidden layer. Activation: 'identity', 'logistic', 'tanh', 'relu', default = 'relu'. Activation function for the hidden layer. solver: 'lbfgs', 'sgd', 'adam', default = 'adam'. The solver for weight optimization; alpha: Strength of the L2 regularization term. The L2 regularization term is divided by the sample size when added to the loss, data type = float, value = 0.0001, default value = 0.0001; batch_size: Size of minibatches for stochastic optimizers, data type = int, value = 'auto', default value = 'auto'; learning_rate: Learning rate schedule for weight updates, values = 'constant', 'invscaling', 'adaptive', value = 'constant', default = 'constant'; learning_rate_init: The initial learning rate used. It controls the step-size in updating the weights. Only used when solver = 'sgd' or 'adam'. data type = float, value = 0.001, default value = 0.001; max_iter: Maximum number of iterations. The solver iterates until convergence (determined by 'tol') or this number of iterations. For stochastic solvers ('sgd', 'adam'), note that this determines the number of epochs (how many times each data point will be used), not the number of gradient steps, data type = int, value = 200, default value = 200;

Table 4 Standard and adversarial risk of all features for $\delta \in S_1$.

Model	Feature	SR	AR	ρ
linear	buildingType19	2,443	2,416	0,989
decisi	buildingType19	2,702	2,783	1,030
baggin	buildingType19	2,608	2,549	0,977
random	buildingType19	2,513	2,503	0,996
gradie	buildingType19	2,421	2,381	0,984
xgbreg	buildingType19	2,405	2,394	0,996
kneigh	buildingType19	2,562	2,482	0,969
mlpreg	buildingType19	2,452	2,445	0,997
linear	floorType	2,443	2,515	1,030
decisi	floorType	2,702	2,796	1,035
baggin	floorType	2,585	2,816	1,089
random	floorType	2,503	2,778	1,110
gradie	floorType	2,421	2,497	1,031
xgbreg	floorType	2,405	2,627	1,093
kneigh	floorType	2,562	2,820	1,101
mlpreg	floorType	2,452	2,434	0,993
linear	houseType19	2,443	2,465	1,009
decisi	houseType19	2,702	2,848	1,054
baggin	houseType19	2,568	2,750	1,071
random	houseType19	2,487	2,552	1,026
gradie	houseType19	2,421	2,419	0,999
xgbreg	houseType19	2,405	2,512	1,045
kneigh	houseType19	2,562	2,512	0,980
mlpreg	houseType19	2,452	2,497	1,018
linear	incompleteHeatingSupply	2,443	3,072	1,257
decisi	incompleteHeatingSupply	2,702	2,804	1,038
baggin	incompleteHeatingSupply	2,660	2,979	1,120
random	incompleteHeatingSupply	2,505	2,505	1,000
gradie	incompleteHeatingSupply	2,421	2,668	1,102
xgbreg	incompleteHeatingSupply	2,405	2,689	1,118
kneigh	incompleteHeatingSupply	2,562	2,518	0,983
mlpreg	incompleteHeatingSupply	2,452	2,902	1,184
linear	lsBySmooth	2,443	3,216	1,316
decisi	lsBySmooth	2,702	3,623	1,341
baggin	lsBySmooth	2,624	4,171	1,589
random	lsBySmooth	2,519	3,098	1,230
gradie	lsBySmooth	2,421	3,258	1,346
xgbreg	lsBySmooth	2,405	3,626	1,508
kneigh	lsBySmooth	2,562	3,244	1,266
mlpreg	lsBySmooth	2,452	3,164	1,291
linear	macroArea	2,443	2,561	1,048
decisi	macroArea	2,702	2,711	1,003
baggin	macroArea	2,612	2,859	1,095
random	macroArea	2,512	2,623	1,044

Table 4 (Continued)

Model	Feature	SR	AR	ρ
gradie	macroArea	2,421	2,591	1,070
xgbreg	macroArea	2,405	2,550	1,060
kneigh	macroArea	2,562	2,624	1,024
mlpreg	macroArea	2,452	2,628	1,072
linear	modernizedFloor	2,443	2,865	1,173
decisi	modernizedFloor	2,702	3,031	1,122
baggin	modernizedFloor	2,613	2,940	1,125
random	modernizedFloor	2,512	2,527	1,006
gradie	modernizedFloor	2,421	2,671	1,103
xgbreg	modernizedFloor	2,405	2,709	1,126
kneigh	modernizedFloor	2,562	2,576	1,005
mlpreg	modernizedFloor	2,452	2,830	1,154
linear	noTerraceBalcony	2,443	2,444	1,000
decisi	noTerraceBalcony	2,702	2,677	0,991
baggin	noTerraceBalcony	2,610	2,501	0,958
random	noTerraceBalcony	2,507	2,509	1,001
gradie	noTerraceBalcony	2,421	2,417	0,998
xgbreg	noTerraceBalcony	2,405	2,380	0,990
kneigh	noTerraceBalcony	2,562	2,480	0,968
mlpreg	noTerraceBalcony	2,452	2,476	1,010
linear	noThermostaticValve	2,443	2,445	1,001
decisi	noThermostaticValve	2,702	2,613	0,967
baggin	noThermostaticValve	2,622	2,722	1,038
random	noThermostaticValve	2,509	2,511	1,001
gradie	noThermostaticValve	2,421	2,413	0,997
xgbreg	noThermostaticValve	2,405	2,399	0,997
kneigh	noThermostaticValve	2,562	2,555	0,998
mlpreg	noThermostaticValve	2,452	2,438	0,994
linear	openKitchen	2,443	2,509	1,027
decisi	openKitchen	2,702	2,702	1,000
baggin	openKitchen	2,549	2,870	1,126
random	openKitchen	2,478	2,482	1,002
gradie	openKitchen	2,421	2,447	1,011
xgbreg	openKitchen	2,405	2,457	1,022
kneigh	openKitchen	2,562	2,550	0,996
mlpreg	openKitchen	2,452	2,502	1,020
linear	rearBuilding	2,443	2,515	1,029
decisi	rearBuilding	2,702	2,702	1,000
baggin	rearBuilding	2,587	2,613	1,010
random	rearBuilding	2,519	2,518	1,000
gradie	rearBuilding	2,421	2,440	1,008
xgbreg	rearBuilding	2,405	2,426	1,009
kneigh	rearBuilding	2,562	2,511	0,980
mlpreg	rearBuilding	2,452	2,512	1,024

Table 4 (Continued)

Model	Feature	SR	AR	ρ
linear	residentialArea	2,443	2,508	1,026
decisi	residentialArea	2,702	2,743	1,015
baggin	residentialArea	2,602	2,671	1,027
random	residentialArea	2,509	2,491	0,993
gradie	residentialArea	2,421	2,446	1,010
xgbreg	residentialArea	2,405	2,424	1,008
kneigh	residentialArea	2,562	2,610	1,019
mlpreg	residentialArea	2,452	2,564	1,046
linear	score2Kitchen	2,443	2,606	1,067
decisi	score2Kitchen	2,702	3,119	1,155
baggin	score2Kitchen	2,651	3,349	1,263
random	score2Kitchen	2,500	2,814	1,126
gradie	score2Kitchen	2,421	2,824	1,166
xgbreg	score2Kitchen	2,405	2,734	1,137
kneigh	score2Kitchen	2,562	2,688	1,049
mlpreg	score2Kitchen	2,452	2,697	1,100
linear	specialEquipment	2,443	2,549	1,043
decisi	specialEquipment	2,702	2,740	1,014
baggin	specialEquipment	2,651	2,739	1,033
random	specialEquipment	2,497	2,535	1,015
gradie	specialEquipment	2,421	2,551	1,054
xgbreg	specialEquipment	2,405	2,574	1,070
kneigh	specialEquipment	2,562	2,562	1,000
mlpreg	specialEquipment	2,452	2,612	1,065
linear	terraceRoofTerrace	2,443	2,495	1,021
decisi	terraceRoofTerrace	2,702	2,702	1,000
baggin	terraceRoofTerrace	2,652	2,739	1,033
random	terraceRoofTerrace	2,497	2,500	1,001
gradie	terraceRoofTerrace	2,421	2,454	1,014
xgbreg	terraceRoofTerrace	2,405	2,470	1,027
kneigh	terraceRoofTerrace	2,562	2,564	1,001
mlpreg	terraceRoofTerrace	2,452	2,541	1,036
linear	underfloorHeating	2,443	2,520	1,032
decisi	underfloorHeating	2,702	3,044	1,127
baggin	underfloorHeating	2,602	2,828	1,087
random	underfloorHeating	2,515	2,529	1,006
gradie	underfloorHeating	2,421	2,537	1,048
xgbreg	underfloorHeating	2,405	2,516	1,046
kneigh	underfloorHeating	2,562	2,542	0,992
mlpreg	underfloorHeating	2,452	2,669	1,088
linear	windowsSingleGlazing	2,443	2,482	1,016
decisi	windowsSingleGlazing	2,702	2,807	1,039
baggin	windowsSingleGlazing	2,588	2,617	1,011
random	windowsSingleGlazing	2,509	2,509	1,000

Table 4 (Continued)

Model	Feature	SR	AR	ρ
gradie	windowsSingleGlazing	2,421	2,440	1,008
xgbreg	windowsSingleGlazing	2,405	2,402	0,999
kneigh	windowsSingleGlazing	2,562	2,494	0,974
mlpreg	windowsSingleGlazing	2,452	2,433	0,992
linear	wwsIncomplete	2,443	2,656	1,087
decisi	wwsIncomplete	2,702	2,679	0,992
baggin	wwsIncomplete	2,630	2,556	0,972
random	wwsIncomplete	2,484	2,484	1,000
gradie	wwsIncomplete	2,421	2,384	0,985
xgbreg	wwsIncomplete	2,405	2,438	1,014
kneigh	wwsIncomplete	2,562	2,465	0,962
mlpreg	wwsIncomplete	2,452	2,576	1,050

7.2 Algorithms

Algorithm 1 (Gradient boosting)

1: Input: Data set $\{(x^{(i)}, y^{(i)}\}_{i=1}^{n}$, loss function $\ell(y, f(x))$, number of iterations M.
2: Initialize model with a constant value:

$$\hat{f}_0(x) = \arg\min_{\gamma} \sum_{i=1}^{n} \ell(y_i, \gamma) \tag{24}$$

3: **for** index $m = 1$ to M **do**
4: Compute pseudo-residuals:

$$r_m^{(i)} := -g_{f_{m-1}}^{(i)} \quad \text{for } i = 1, \ldots, n. \tag{25}$$

5: fit a base learner (e.g. tree) $t_m(x)$ to pseudo-residuals
6: Compute multiplier γ_m by solving the following one-dimensional optimization problem:

$$\gamma_m = \arg\min_{\gamma} \sum_{i=1}^{n} \ell(y^{(i)}, \hat{f}_{m-1}(x^{(i)}) + \gamma t_m(x^i)) \tag{26}$$

7: Update the model:

$$\hat{f}_m(x) = \hat{f}_{m-1}(x) + \gamma_m t_m(x). \tag{27}$$

8: Output: $\hat{f}(x) = \hat{f}_M(x) = \sum_{m=0}^{M} \hat{f}_m(x)$.
9: **end for**

Algorithm 2 ((unregularized) XGBoost Algorithm)

1: Initialize model with a constant value:

$$\hat{f}_0(x) = \arg\min_{\gamma} \sum_{i=1}^{n} \ell(y^{(i)}, \hat{\gamma}). \tag{28}$$

2: **for** $m = 1$ to M: **do**

3: Compute the 'gradients' g_m and 'hessians' $\mathcal{H}_m$:

$$g_m(x^{(i)}) = \left[\frac{\partial \ell(y^{(i)}, f(x_i))}{\partial f(x^{(i)})}\right]_{f(x)=\hat{f}_{m-1}(x)}. \tag{29}$$

$$\mathcal{H}_m(x^{(i)}) = \left[\frac{\partial^2 \ell(y^{(i)}, f(x^{(i)}))}{\partial f(x^{(i)})^2}\right]_{f(x)=\hat{f}_{m-1}(x)}. \tag{30}$$

4: Fit a base learner (or weak learner, e.g. tree) using the training set $\{x^{(i)}, -\frac{\hat{g}_m(x^{(i)})}{\hat{\mathcal{H}}_m(x^{(i)})}\}_{i=1}^{n}$ by solving the optimization problem below:

$$\hat{\gamma}_m = \arg\min_{\gamma} \sum_{i=1}^{n} -\frac{1}{2}\hat{\mathcal{H}}_m(x^{(i)}) \left[-\frac{\hat{g}_m(x^{(i)})}{\hat{\mathcal{H}}_m(x^{(i)})} - \hat{\gamma}(x^{(i)})\right]^2. \tag{31}$$

$$\hat{f}_m(x) = \nu \cdot \hat{\gamma}_m(x) \tag{32}$$

5: Update the model:

$$\hat{f}_m(x) = \hat{f}_{m-1}(x) + \hat{f}_m(x). \tag{33}$$

6: Output: $\hat{f}(x) = \hat{f}_M(x) = \sum_{m=0}^{M} \hat{f}_m(x)$.

7: **end for**

Algorithm 3 (Calculating the standard and adversarial risk in case of $\delta \in S_1$ using LOOCV)

1: **for** index i in set of observations $N = \{1, \ldots, n\}$ **do**
2: drop ith observation from dataset
3: fit a regression-model $\hat{f}_{-i}$ on the original dataset with ith observation removed
4: predict $\hat{f}_{-i}(x^{(i)})$ on the original dataset
5: calculate $\mathbf{mse}^{(i)}$
6: **end for**
7: calculate SR
8: **for** index i in set of observations $N = \{1, \ldots, n\}$ **do**
9: drop ith observation from dataset
10: fit a regression-model $\hat{f}_{-i}$ on the original dataset with ith observation removed
11: corrupt i-th observation $x^{(i)}$ by $\delta \in S_1 \subset \mathcal{S}$
12: add corrupted ith observation back to get the original amount of N observations
13: predict $\hat{f}_{-i}(x^{(i)})$ on the original dataset
14: calculate $\mathbf{mse}^{(i)}$
15: **end for**
16: calculate AR
17: compare SR with AR

Funding Open Access funding enabled and organized by Projekt DEAL.

Declaration of competing interest The first author is involved in creating rental guides. This work consists of surveying rents of flats and doing statistical analyses in behalf of German cities and municipalities. He was not involved in creating the Munich rental guide. The second author was involved in the statistical analyses of the Munich rental guides in 2013, 2015, 2017, 2019, 2021, 2023. The data used for the Munich rental guide 2019, which are used in this paper, have been collected by KANTAR (TNS Infratest). These involvements of both authors could be interpreted as conflict of interest appeared to influence the work reported in this paper.

References

Aigner, Oberhofer, Schmidt (1993) Eine neue methode zur erstellung eines mietspiegels am beispiel der stadt regensburg. Wohnungswirtschaft und Mietrecht WM 1993(1/2/93):16–21

Altman NS (1992) An introduction to kernel and nearest-neighbor nonparametric regression. The American Statistician 46(3):175–185, http://www.jstor.org/stable/2685209

Bentley JL (1975) Multidimensional binary search trees used for associative searching. Commun ACM 18(9):509–517, https://doi.org/10.1145/361002.361007

Biggio B, Roli F (2018) Wild patterns: Ten years after the rise of adversarial machine learning. Pattern Recognition 84(3):317–331, https://doi.org/10.1016/j.patcog.2018.07.023, http://arxiv.org/pdf/1712.03141v2

Biggio B, Corona I, Maiorca D, Nelson B, Srndic N, Laskov P, Giacinto G, Roli F (2013) Evasion attacks against machine learning at test time 7908(1):387–402, https://doi.org/10.1007/978-3-642-40994-3_25, http://arxiv.org/pdf/1708.06131v1

Borth D, Hüllermeier E, Kauermann G (2023) Maschinelles Lernen, Springer Berlin Heidelberg, Berlin, Heidelberg, pp 19–49. https://doi.org/10.1007/978-3-662-66278-6_4

Breiman L (1984) Classification and regression trees. The Wadsworth statistics, probability series

Breiman L (1996) Bagging predictors. Machine Learning 24(2):123–140, https://doi.org/10.1007/BF00058655

Breiman L (1997) Arcing the edge. University of California, 486

Breiman L (2001) Random forests. Machine Learning 45(1):5–32, https://doi.org/10.1023/A:1010933404324

Cauchy A (1847) Methode generale pour la resolution des systemes d'equations simultanees. CR Acad Sci Paris 25:536–538, https://ci.nii.ac.jp/naid/10026863174/en/

Chen T, Guestrin C (2016) Xgboost: A scalable tree boosting system. CoRR abs/1603.02754, http://arxiv.org/abs/1603.02754

Cunningham P, Delany SJ (2020) k-nearest neighbour classifiers: 2nd edition (with python examples). CoRR abs/2004.04523, https://arxiv.org/abs/2004.04523

Fahrmeir L, Kneib T, Lang S, Marx BD (2022) Regression: Models, methods and applications, second edition edn. Springer eBook Collection, Springer, Berlin, Heidelberg, https://doi.org/10.1007/978-3-662-63882-8

Fitzenberger B, Fuchs B (2017) The residency discount for rents in germany and the tenancy law reform act 2001: Evidence from quantile regressions. German Economic Review 18(2):212–236

Freund Y, Schapire RE (1996) Experiments with a new boosting algorithm. In: In Proceedings of the Thirteenth Internations Conference on Machine Learning, Morgan Kaufmann, pp 148–156

Freund Y, Schapire RE (1999) A short introduction to boosting. In: In Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence, Morgan Kaufmann, pp 1401–1406

Friedman JH (2001) Greedy function approximation: A gradient boosting machine. The Annals of Statistics 29(5):1189–1232, https://doi.org/10.1214/aos/1013203451, https://projecteuclid.org/journals/annals-of-statistics/volume-29/issue-5/Greedy-function-approximation-A-gradient-boostingmachine/10.1214/aos/1013203451.full

Friedman J, Hastie T, Tibshirani R (2000) Additive logistic regression: a statistical view of boosting (With discussion and a rejoinder by the authors). The Annals of Statistics 28(2):337 – 407, https://doi.org/10.1214/aos/1016218223

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

Hastie T, Tibshirani R, Friedman JH (2017) The elements of statistical learning: Data mining, inference, and prediction, second edition, corrected at 12th printing 2017 edn. Springer series in statistics, Springer, New York, NY

James G, Witten D, Hastie T, Tibshirani R (2017) An introduction to statistical learning: With applications in R, corrected at 8th printing edn. Springer texts in statistics, Springer, New York and Heidelberg and Dordrecht and London

Javanmard A, Soltanolkotabi M, Hassani H (2020) Precise tradeoffs in adversarial training for linear regression. CoRR abs/2002.10477, https://arxiv.org/abs/2002.10477

Kauermann G, Windmann M (2016) Mietspiegel heute: Zwischen realität und statistischen möglichkeiten. Wirtschafts- und sozialstatistisches Archiv : ASTA : eine Zeitschrift der Deutschen Statistischen Gesellschaft 10(4):205–223

Madry A, Makelov A, Schmidt L, Tsipras D, Vladu A (2018) Towards deep learning models resistant to adversarial attacks. In: International Conference on Learning Representations, https://openreview.net/forum?id=rJzIBfZAb

Mehlhorn K (1988) Datenstrukturen und effiziente Algorithmen: Band 1: Sortieren und Suchen. Datenstrukturen und effiziente Algorithmen, Vieweg+Teubner Verlag, https://books.google.de/books?id=EmxIAQAAIAAJ

Mehrabi M, Javanmard A, Rossi RA, Rao A, Mai T (2021) Fundamental tradeoffs in distributionally adversarial training. CoRR abs/2101.06309, https://arxiv.org/abs/2101.06309

Pedregosa F, Varoquaux G, Gramfort A, Michel V, Thirion B, Grisel O, Blondel M, Prettenhofer P, Weiss R, Dubourg V, Vanderplas J, Passos A, Cournapeau D, Brucher M, Perrot M, Duchesnay E (2011) Scikit-learn: Machine learning in Python. Journal of Machine Learning Research 12:2825–2830

Raybaut P (2009) Spyder-documentation. Available online at: pythonhosted org

Schapire RE (1990) The strength of weak learnability. Machine Learning 5(2):197–227, https://doi.org/10.1007/BF00116037

Seabold S, Perktold J (2010) statsmodels: Econometric and statistical modeling with python. In: 9th Python in Science Conference

Szegedy C, Zaremba W, Sutskever I, Bruna J, Erhan D, Goodfellow IJ, Fergus R (2014) Intriguing properties of neural networks. In: Bengio Y, LeCun Y (eds) 2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings, http://arxiv.org/abs/1312.6199

Tsipras D, Santurkar S, Engstrom L, Turner A, Madry A (2019) Robustness may be at odds with accuracy. In: International Conference on Learning Representations, https://openreview.net/forum?id=SyxAb30cY7

Van Rossum G, Drake FL (2009) Python 3 Reference Manual. CreateSpace, Scotts Valley, CA

Windmann M, Kauermann G (2019) Mietspiegel für München 2019 - Statistik, Dokumentation und Analysen. Sozialreferat der Landeshauptstadt München

Wood SN (2017) Generalized Additive Models: An Introduction with R, Second Edition. Chapman & Hall / CRC Texts in Statistical Science, CRC Press, Portland, https://ebookcentral.proquest.com/lib/gbv/detail.action?docID=4862399