Stratified Sampling for Regression

Nadja Sauter nadja.sauter@campus.lmu.de
Department of Statistics
Ludwig Maximilian University of Munich

Anne Gritto anne.gritto@campus.lmu.de
Department of Statistics
Ludwig Maximilian University of Munich

Supervisors
Prof. Dr. Matthias Feurer and Dr. Giuseppe Casalicchio

Abstract

Stratification is typically applied in classification to sample proportionally to the respective class sizes in order to preserve the original class distribution in resulting subsets. In this way, stratification can address issues arising from imbalanced and small datasets, overall leading to a better generalization performance of machine learning models. In our experiments, we investigate the application of stratification in regression, focusing on its potential to improve hyperparameter optimization and the final performance of regression models. We apply a group-sized discretization of the target variable to create bins from which one can sample proportionally during cross-validation of Random Search. The results indicate that our stratification approach compared to naive random sampling increases the overlap of the training and validation data within cross-validation, but does not improve the final performance of the models in our setting. However, the error of the Mean Squared Error (MSE) estimator and its variance are reduced through stratification, indicating a better estimation. The findings suggest that stratification could be particularly beneficial for smaller datasets. We think that stratification should be considered as an optional sampling approach within cross-validation for hyperparameter tuning and model selection.

1. Introduction

Traditional random sampling techniques can fail to adequately represent the underlying population especially when dealing with imbalanced and small datasets (Hornung et al., 2023; Sechidis et al., 2011; Thabtah et al., 2020). In the continuous case, imbalanced datasets are characterized by skewed distributions, whereas in classification tasks the number of observations between classes are not equally distributed. The bigger the imbalance, the higher is the risk of a bias towards the majority class of a trained model, i.e. the class with more samples (Brus, 2021). For example, if we create a train and test dataset by random sampling from an imbalanced categorical target variable, in the worst-case scenario the train set does not contain any instances of the minority class at all. In this case, the model cannot learn anything about the minority class during training.

In the following, we want to investigate stratification as an alternative approach to random sampling in order to overcome issues arising especially during training of classical machine

learning algorithms on small and imbalanced datasets. We focus on the less common application of stratification in regression. Our goal is to investigate if stratification within cross-validation of the Random Search algorithm improves hyperparameter optimization and the final performance of the trained regression models. Moreover, we analyze if any special experimental parameters of the data generation or stratification algorithm are particularly relevant for a possible improvement of our stratified approach.

2. Related Work

Looking at the literature, there are different approaches how to tackle issues related to random sampling in the creation of train and test splits for the application of standard machine learning algorithms. For example, downsampling and upsampling both aim to rebalance a dataset before training. Studies have shown that these methods can significantly improve the overall performance of several types of classifiers (He and Garcia, 2009; Visa and Ralescu, 2005). On the one hand, downsampling techniques reduce the size of the majority class to create a more balanced dataset. On the other hand, upsampling duplicates random instances of the minority class to achieve a similar balance. While rebalancing the data in this way, the approach may also result in information loss or overfitting, particularly in cases of severe imbalance or limited data availability (He and Garcia, 2009). A more sophisticated sampling method for classification tasks is Synthetic Minority Oversampling Technique called SMOTE (Chawla et al., 2002). Synthetic minority samples are created by interpolating the k-nearest neighbors of an original data point of the minority class in the feature space. Balancing seems to not improve the performance of strong classifiers but helps in the case of weak classifiers (Chawla et al., 2002). This idea can be expanded to regression with the SMOTER algorithm (Torgo et al., 2013). In order to better handle imbalanced and small datasets than random sampling without reducing or replicating data observations, stratification is another common approach. As we are going to apply this method in our experiments, we now go into the details of this method and its application in classification and in regression as well as within cross-validation.

2.1 Stratification in Classification

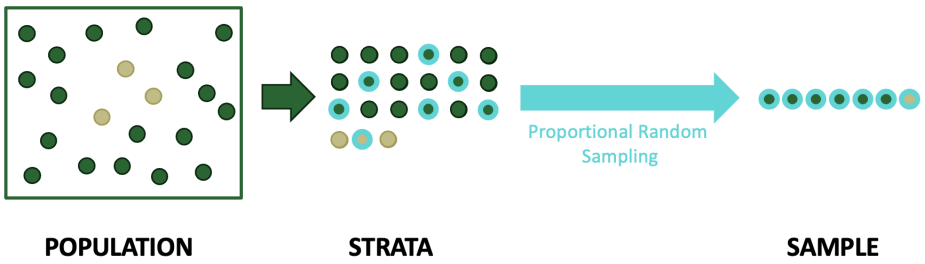


Figure 1: Illustration of the concept of stratification in classification. Data points of a population with uneven class distribution are sampled. However, they are not sampled randomly from the population, but proportionally according to the strata.

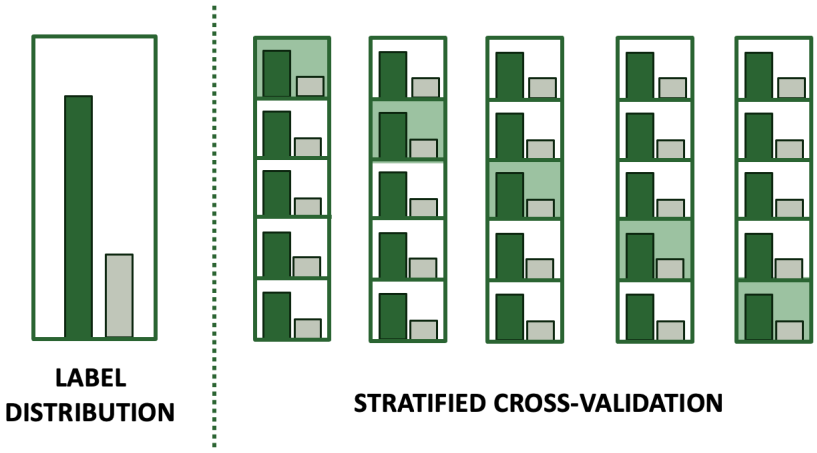


Figure 2: Illustration of the concept of stratification in cross-validation. The label distribution is maintained within the folds.

Typically, stratification is applied in the context of classification. The main idea (see illustration 1) is that we sample the data proportionally to their respective strata size in order to preserve the original class proportion in resulting subsets (Raschka, 2020). In this way, stratification helps to ensure that each data split accurately represents the overall dataset. This can lead to better generalization performance of machine learning models since by maintaining the proportional representation of classes in each subset, models trained on stratified data splits are less likely to be biased towards the majority class (Brus, 2021; Diamantidis et al., 2000). Overall, stratification has shown to improve performance of classifiers especially in the case of small and imbalanced datasets (Sechidis et al., 2011), whereas for relatively large and balanced datasets, random and stratified sampling result in similar results (Raschka, 2020).

2.2 Stratification in cross-validation for Classification

A common technique in machine learning for the evaluation and selection of a model and its hyperparameters is K-fold cross-validation (Browne, 2000). This method randomly partitions the original dataset into K (nearly) equal-sized subsets, called folds. One of these K subsets is retained as the validation set for testing the fitted model, while the remaining $(K-1)$ folds are used as training data. This process is repeated K times, where each fold serves as the validation set exactly once. The performance metric, for example accuracy in classification, is then averaged across all K iterations to obtain an overall robust estimate of the model's performance.

Applying the idea of stratified sampling (see section 2.1) to cross-validation in classification, each fold is sampled proportionally from the discrete label distribution. While usually the assignment of a data point to a fold is random, stratification aims to maintain the label distribution within each fold as illustrated in Figure 2. This leads to more reliable evaluation metrics and helps prevent overestimation or underestimation of model performance.

In this way, stratification within cross-validation can reduce the variance and bias of the estimator (Kohavi, 1995). The stratified K-fold cross-validation algorithm for classification is summarized in Algorithm 1. Currently, an implementation of stratified cross-validation exists in scikit-learn (Pedregosa et al., 2011) only for classification, called *StratifiedKFold*.

Algorithm 1 Stratified K-fold cross-validation in classification

$K \in \{2, 3, 4, ...\}$

$data = \biguplus_{k \in K} Fold_k$ where $|Fold_k| = data/K \ \forall k$

Stratification: Sample elements of $Fold_k$ proportional to label distribution

for k *in range* (0, K) **do**

  $V \leftarrow Fold_k$ ▷ Validation Set

  $T \leftarrow data \backslash V$ ▷ Training Set

  Train model on training data T with model parameter λ

  $Acc_k \leftarrow$ Evaluate trained model on validation data V

end

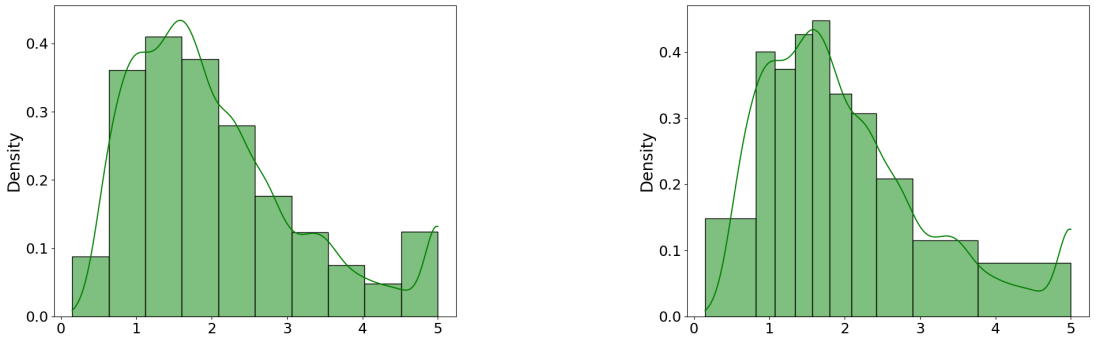
$Acc \leftarrow \frac{1}{K} \sum_{k=1}^{K} Acc_k$

2.3 Stratification in Regression

Now applying stratification to regression instead of classification, we encounter the problem that the target distribution is not categorical but continuous. Consequently, we cannot directly sample in proportion to the classes anymore. In order to stratify, groups need to be created from the target distribution. There exists several approaches how to create these bins of the continuous target variable.

Firstly, **equidistant discretization** creates groups based on a fixed bin width as it is depicted in the left plot of Figure 3. However, problems can arise here if too little observations are present in one bin, so that it is not possible to sample proportionally K-times during the cross-validation. This form of discretization is implemented within the python package *verstack* as the function *stratified_continuous_split* which creates stratified train-test-splits for continuous target variables. In this approach however, the assigned bins are corrected if some include none or a singular value by combining them to their nearest neighbors (Danil, 2023).

Secondly, **quantile-based groups** can be formed based on the quantiles of the data distribution such that each group contains the same number of data points, or ±1 if the data is not evenly separable. For example, in the right plot of Figure 3, each bin contains 10% of the data. In this way we can always sample the K folds proportionally within cross-validation. However, for skewed data, the bin width gets partly very high. For instance, in the example of Figure 3 the bin widths at the tails of the distribution are higher than in the center. This weakens the stratification approach in this area as we sample from quite a big area at the tails for each fold. Moreover, the number of data points in a group varies for fixed quantiles depending on the dataset size. If we create, for instance, groups based on 10% quantiles for a dataset that contains 1000 versus one that only has 200 instances, the groups consist of 100 and 20 data points, respectively. The larger the datasets are, the



(a) Equidistant binning

(b) Quantile-based binning

Figure 3: Illustration of a binned continuous label distribution with 10 groups by a) equidistant and b) quantile-based discretization.

more data points are in a bin from which instances are sampled such that the distribution may not be represented well again in a fold.

Thirdly, **group-sized discretization** defines a fixed group size from which we sample our data for each fold of the cross-validation. This is a reformulation of the quantile-based discretization. In the quantile-based discretization, the number of data points in a resulting group depend on the quantile chosen and the size of the dataset. Group-sized discretization, however, aims to keep the number of instances in the created bins fixed for any dataset size. Therefore, the respective quantiles vary depending on the dataset size. The idea is to order the data based on the target variable and divide the sorted data into groups of a fixed size. Then, $\frac{group\_size}{K}$ instance(s) of a group are randomly allocated to each fold. This random allocation ensures that the stratification process preserves some randomness, while maintaining balance within each partition and considering e.g. the margins of a distribution.

Fourthly, instead of sampling randomly from equidistant bins or quantile-based groups, **sorted stratification** orders all data points of the target variable and assigns each data point directly to one fold following this order without any sampling (Scott, 2016). Having a small group/quantile/bin size like in the discretizations before is similar to the sorted method.

Overall, different approaches can be chosen for the left over data points if the data is not dividable by the number of folds K. For instance we can just randomly drop these values or assign them to some random fold. In our experiments, we decided to use the latter approach and implement the group-sized discretization for stratifying the target distribution during cross-validation which is illustrated in Figure 4. In this way, we hold the group size constant, avoiding the described issues of the equidistant and quantile-based discretization. By choosing small group sizes, we are close to the stratified ordering while still allowing some randomness during sampling within the groups into the K-Folds. To the best of our knowledge, there is currently no standard implementation of the stratified cross-validation in regression in Python. Furthermore, we also did not find any elaborated benchmarks about different stratification approaches.

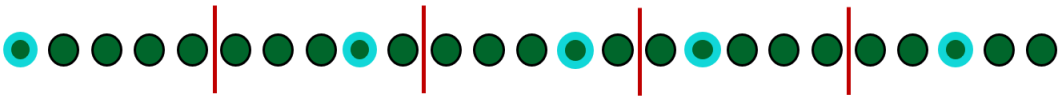


Figure 4: Illustration of the concept of group-sized discretization for stratification. Let the data points be ordered with a defined group size of 5, indicated by a red line. The data points marked in turquoise are randomly sampled from each group and assigned to the same fold. We experiment with group sizes of 5 and 10 for 5-fold cross-validation.

3. Research Question and Hypothesis

Our main research question examines the possible improvement of stratification compared to random sampling in regression. Specifically, we aim to investigate whether stratification increases the performance of regression models and under what conditions this approach could be most beneficial. We address these questions through three hypotheses, each of them focusing on different aspects:

1. **Performance:** The use of stratification is at least not worse than without stratification. If there is an impactful improvement needs to be investigated through experiments. We assess the model performance using metrics such as Mean Squared Error (MSE), Mean Absolute Error (MAE) as well as R^2 on the test data. We also evaluate the choice of the best hyperparameters and the error of the MSE estimator itself.

2. **Data Characteristics:** Stratification is especially helpful with imbalanced, small and noisy datasets. We think that stratification may help address challenges associated with these data characteristics, leading to improved model performance. We will explore this hypothesis by varying data transformations such as identity, logarithmic, and square root, training set sizes of 200 and 1,000 samples, and noise levels of 0 and 3.

3. **Method:** Stratification depends on the binning of the continuous target variable. A small group size indicates that ordered stratification might be a better approach than group-sized discretization. We investigate this hypothesis by varying the group size parameter as explained in Section 2.3, exploring values of 5 and 10.

4. Experiments

This section describes the design of our experiments in detail. It gives an overview of the models, the data, the explored hyperparameter grid of Random Search, the experimental parameters, the implementation of the stratified and unstratified cross-validation, and the methods used to evaluate the results (see Figure 5). The Python implementation and results can be found in our GitHub Repository[1].

1. `https://github.com/annegriddl/slds_stratification`

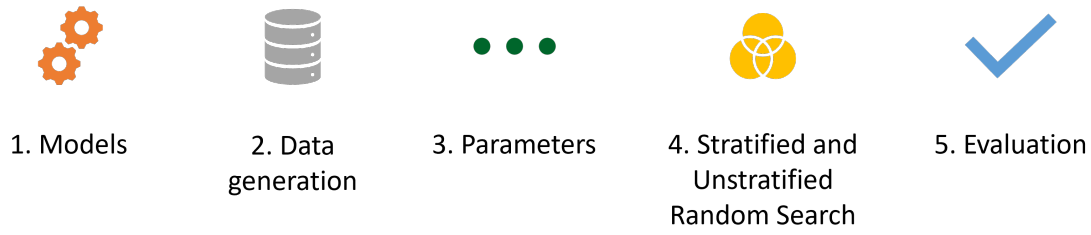


Figure 5: Overview of experimental set-up.

4.1 Models

We employ two different models to test our research hypothesis, a Random Forest regressor (RF) and the boosting algorithm XGBoost (XGB). Both models are ensemble learning methods suitable for regression tasks (Breiman, 2001; Chen and Guestrin, 2016).
A Random Forest constructs multiple decision trees, each based on a subset of the dataset, and aggregates their predictions to generate the final output. The advantages of Random Forests are their robustness to outliers and noise as well as their good performance and simplicity (Breiman, 2001). We use the Random Forest Regressor implemented by `scikit-learn` (Pedregosa et al., 2011).
`XGBoost` (Chen and Guestrin, 2016) is a scalable tree boosting system which is widely used in machine learning. It performs additive optimization and incorporates a regularized model to generalize well to new data and avoid overfitting. A key advantage of XGBoost is its high scalability, which leads to faster training times.
For hyperparameter tuning of the models we employ Random Search. The hyperparameter grids for Random Forest and XGBoost are shown in Table 1, respectively. The ranges of the hyperparameters have been chosen according to existing literature, specifically according to suggestions by Maulana et al. (2023), Kapoor and Perrone (2021) and Stoytchev (2020) for XGBoost as well as by Góralczyk et al. (2021), van Rijn and Hutter (2018) and Koehrsen (2018) for Random Forest. All other hyperparameters are kept as their default value for each model implementation.

4.2 Data generation

We generate the data using the "Friedman #1" regression problem in `scikit-learn` (Pedregosa et al., 2011) which is based on the work of Friedman (1991) and Breiman (1996). The features X are independent, uniformly distributed variables on the interval $[0, 1]$. The target variable y is generated by the following equation

$$y(X) = 10 \sin(\pi \cdot X[:,0] \cdot X[:,1]) + 20(X[:,2] - 0.5)^2 + 10 \cdot X[:,3] + 5 \cdot X[:,4] + noise \cdot N(0,1).$$

In order to compare the results of our experiments, we always maintain a large fixed test set size of $n = 100,000$ instances and keep the seed for generating the test data fixed. In this way our test data simulates the ground truth in our artificial experimental set-up. On the other

Model	Hyperparameter	Values	Count
Random Forest	n_estimators	500	1
	min_samples_split	{2, 3, ..., 10}	9
	min_samples_leaf	{1, 2, ..., 10}	10
	max_features	{1, 2, ..., 8}	8
XGBoost	learning_rate	Linear-space {0.001,..., 0.4}	10
	min_child_weight	{1, 2, ..., 9}	9
	gamma	Log-space {1, ..., 20}	10
	max_depth	Log-space {2, ..., 20}	20
	subsample	Uniform (0.5, 1)	10
	colsample_by_tree	Uniform (0.5, 1)	10

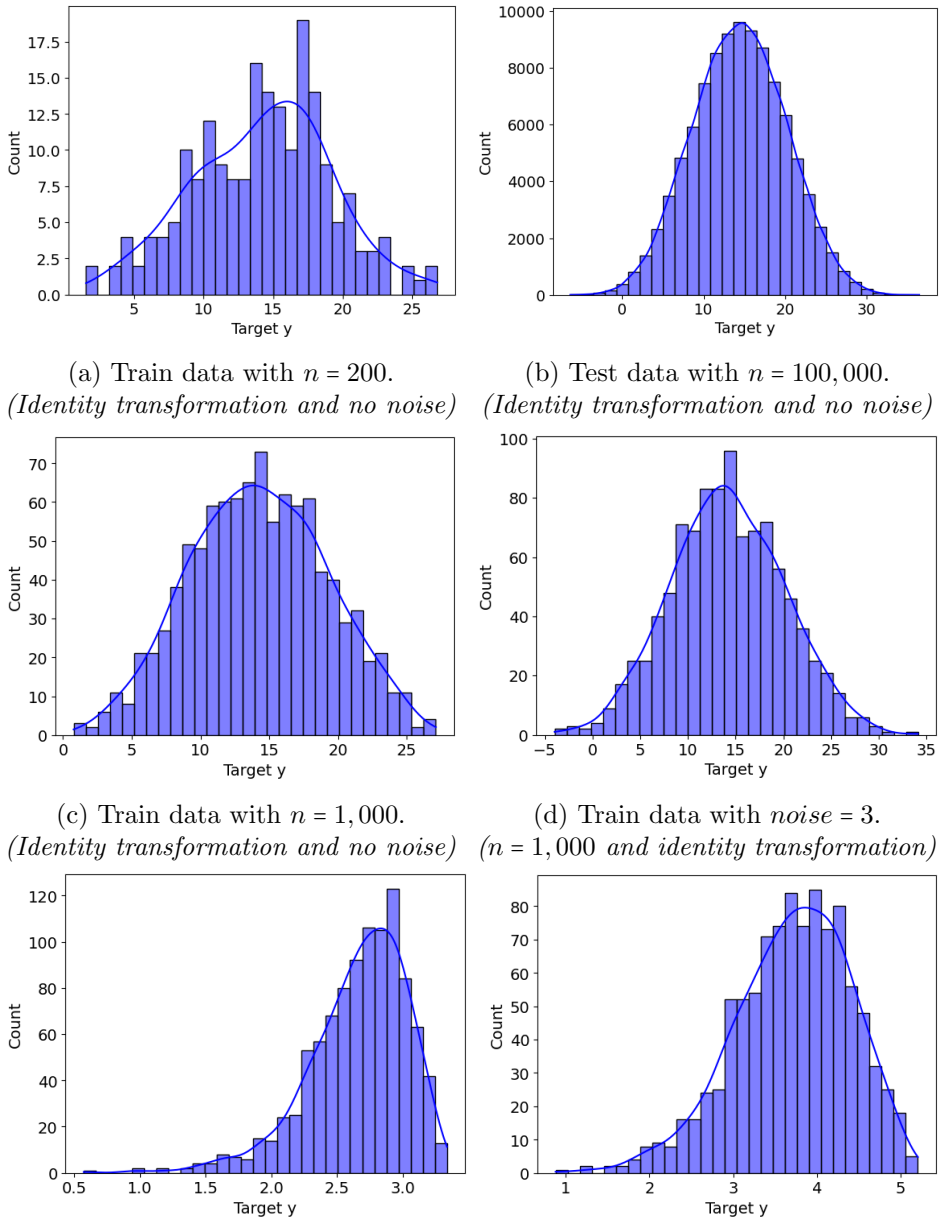
Table 1: Hyperparameter grid to be exploited in Random Search with 200 iterations for Random Forest and XGBoost, respectively. As indicated, parameters are from a linear, logarithmized or uniformly distributed space.

side, the train dataset varies. Firstly, we always use a different seed for data generation. Secondly, we explore different options of the experimental parameters. In order to investigate whether stratification in regression is particularly effective with small, unbalanced and noisy datasets, we compare different sizes by setting $n\_train = 200$ vs. $n\_train = 1,000$. Furthermore, we use different values for the noise variable, specifically $noise = 0$ vs. $noise = 3$. Finally, we apply different transformations to both, the train and test data, namely identity, natural logarithm (log) and square root (sqrt) transformations. The number of features is fixed and set to 8. Note that only 5 feature variables are used to generate the target variable and therefore 3 features are independent of y and act as additional noise.
Figure 6 shows the distribution of the target variable y for different generated datasets used in the experiments. Figures 6a, 6b and 6c show the distributions for the train datasets with 200 and 1,000 data points as well as the test dataset with 100,000 observations. It can be seen that the higher the number of data points in a dataset, the smoother the distribution of y. Figure 6d demonstrates the effect of noise on the variance of the distribution when it is set to 3. The range of the 1,000 data points with no noise is about 27 units, while the range of y with $noise = 3$ is about 38 units, also including negative data points. The Figures 6e and 6f depict the target variable after applying the natural logarithm and the square root transformations. Note that the values of y can also be negative. In this case, we shift the distribution to be positive to ensure that transformations like the logarithm can be applied. We shift the training as well as the testing data by

$$y_{new} = y + (|\min\{y_{train}, y_{test}\}| + 1.00000001).$$

4.3 Parameters

To test our hypotheses, we explore different experimental parameters denoted as $\omega^{(r)}$ at each repetition r for the models Random Forest and XGBoost. Experimental parameters



(a) Train data with $n = 200$.
(Identity transformation and no noise)

(b) Test data with $n = 100,000$.
(Identity transformation and no noise)

(c) Train data with $n = 1,000$.
(Identity transformation and no noise)

(d) Train data with $noise = 3$.
($n = 1,000$ and identity transformation)

(e) Train data with log transformation.
($n = 1,000$ and no noise)

(f) Train data with square root.
($n = 1,000$ and no noise)

Figure 6: Examples of differently generated datasets.

that vary during our experiments are the size of the train dataset, $n\_train = \{200, 1000\}$, the noise used to generate the data, $noise = \{0, 3\}$, the transformation applied to the generated data, $transformation = \{identity, log, sqrt\}$, and the group size of the discretization, $group\_size = \{5, 10\}$, introduced in section 2.3. Table 2 shows the relationship of the group size and the corresponding quantile, applying our group-based discretization, where we hold the group size constant for the different training sizes.

In total, the parameter settings result in 24 experimental parameter combinations which we repeat $r = 200$ times each. Parameters that are fixed, as described in the data generation (see section 4.2), are the number of features ($n\_features = 8$) and the size of the test set ($n\_train = 100,000$). Furthermore, we set the number of folds K in cross-validation to 5 and use $I = 200$ iterations in Random Search.

Group size	**Train size**	**Number of groups**	**Quantile**
5	200	40	2.5%
5	1000	200	0.5%
10	200	20	5%
10	1000	100	1%

Table 2: Quantiles that result from group-based discretization with 5 and 10 data points per group with a train data size of 200 and 1000 data points, respectively.

4.4 Stratified and Unstratified Random Search

To train our models, we tune the hyperparameters listed in Table 1 with Random Search. Let $\lambda^{(i,r)}$ be the hyperparameters at iteration $i \in \{1, ..., I\}$ with $I = 200$ within a repetition $r \in \{1, ..., R\}$ with $R = 200$ for the models Random Forest and XGBoost, respectively. 200 iterations of Random Search mean that 200 different hyperparameter combinations are used to find the setting that minimizes the error on the validation set in cross-validation measured in our case as the Mean Squared Error. Within Random Search, we perform 5-fold cross-validation to robustly evaluate the fitted models for each iteration. We use the same training data and initialization of the models as well as of Random Search specified by the same seed for one repetition. Within one repetition, we train the model first with a normal, unstratified cross-validation in each iteration, then re-initialize the model with the same seed and train again with the same data, but with stratified cross-validation splits. This ensures the comparability of the two sampling approaches, stratified versus unstratified.

To perform stratified cross-validation, we need to pass the indices of the data points with their respective fold allocation to the Random Search object. Therefore, we first create groups of the target variable with our group-based discretization approach, i.e. we use Pandas `qcut` function. This implementation uses quantiles for the discretization, but we specify this by the number of groups we need by dividing the number of train data points by the chosen group size (5 or 10), i.e. $n\_groups = \frac{n\_train}{group\_size}$. This results for different dataset sizes in different quantiles used to create the validation data (see Table 2. To split the data points in each group into folds, we utilize `StratifiedKFold` implemented by `scikit-learn` (Pedregosa et al., 2011). This stratification algorithm is used in our experiments and inspired by the implementation of Anil (2022). The Python code of our stratification implementation can also be found in the Appendix in function 1.

The cross-validation performances of each hyperparameter combination λ for each experimental parameter combination ω are saved as the averaged MSE formulated for $K = 5$ by

$$MSE_{VAL}^{(i,r)}\left(\lambda^{(i,r)}, \omega^{(r)}\right) = \frac{1}{K}\sum_{k=1}^{K} MSE_{k}^{(i,r)}\left(\lambda^{(i,r)}, \omega^{(r)}\right).$$

The best hyperparameter combination for one repetition r is then the one that results in the smallest MSE defined as

$$\lambda_{best}^{(r)} = \arg\min_{\lambda} MSE^{(r)} = \arg\min_{\lambda}\{MSE^{(1,r)}, ..., MSE^{(I,r)}\}.$$

Note that this is not in general the theoretical best hyperparameters, but the best choice of the options explored as defined in the parameter grid. This procedure of stratified Random Search in regression is summarized in Algorithm 2.

Algorithm 2 Stratified Random Search in regression

$K \in \{2, 3, 4, ...\}$

$data = \biguplus_{k \in K} Fold_k$ where $|Fold_k| = \frac{data}{K}\ \forall k$

Stratification: Create indices that allocate each data point into one fold in proportion to the bins created by group-sized discretization according to function 1 in the Appendix.

for i *in range* (0, I) **do**
  for k *in range* (0, K) **do**
    $Fold_k \leftarrow q_k$ ▷ Group-sized discretization
    $V \leftarrow Fold_k$ ▷ Validation Set
    $T \leftarrow data \backslash V$ ▷ Training Set
    Train model on training data T with parameter $\lambda^{(i,r)}$
    $MSE_k^{(i)} \leftarrow$ Evaluate trained model on validation data V
  end
  $MSE^{(i)} \leftarrow \frac{1}{K}\sum_{k=1}^{K} MSE_k^{(i)}$ ▷ Defined as: $MSE_{VAL}^{(i,r)}(\lambda^{(i,r)}, \omega^{(r)})$
end

$\lambda_{best}^{(r)} = \arg\min_{\lambda} MSE^{(r)} = \arg\min_{\lambda}\{MSE^{(1,r)}, ..., MSE^{(I,r)}\}$

4.5 Evaluation

During Random Search, we save different evaluation metrics for each hyperparameter combination λ and for each experimental parameter combination ω. We evaluate and compare the models with regard to unstratified vs. stratified cross-validation by a descriptive exploration of the folds, an analysis of the best chosen hyperparameters, the final performance on the test data and lastly by the estimation error of the MSE. The results are presented in Section 5.

To summarize the experimental set-up, we have 24 experimental parameter combinations ω with $R = 200$ repetitions per model each. Within one repetition, we employ stratified and unstratified cross-validation with $I = 200$ iterations of Random Search for hyperparameter tuning. This process results in a total of 9,600 individual results combined for both models.

In terms of computational execution, Random Forest models were primarily run on a remote computer equipped with an Intel® Core™ i7-12700 Processor with 12 cores, 20 simultaneous threads and a graphic card with 12 GB of VRAM. Conversely, XGBoost runs were mostly executed on a local laptop equipped with an 11th Gen Intel® Core™ i7-1165G7 CPU, 4 cores, 8 threads and 16 GB of RAM. Additionally, we compared the parallelization of the repetitions to the parallelization of the Random Search with regard to runtime improvements. The former is implemented with `Joblib` and the latter by setting the parameter n_jobs to -1 in scikit-learn's `RandomizedSearchCV`. Parallelizing the repetitions generally exhibited shorter running times compared to parallelizing Random Search, indicating that efficiency can be gained through the first approach. The average running times for one single repetition per model for each parallelization and hardware type are shown in the Appendix in Table 3. Based on the results, we observe notable differences in the running times across these set-ups and models. For Random Forest, runs on the remote computer with parallel repetitions exhibited the shortest running times at 3.88 minutes per repetition on average, while runs with parallel Random Search on the local machine took the longest time at an average of 33.46 minutes. Moreover, XGBoost shows compared to Random Forest very short running times. Runs with parallel repetitions on the Remote Computer are the fastest with on average 0.19 minutes. Overall, these findings show the impact of a computational environment and parallelization strategies on the efficiency of the training process of our models. However, it is important to note that the running times of repetitions do vary due to factors such as concurrent usage of the Remote Computer or the level of activity on the local laptop during the experiments.

5. Results

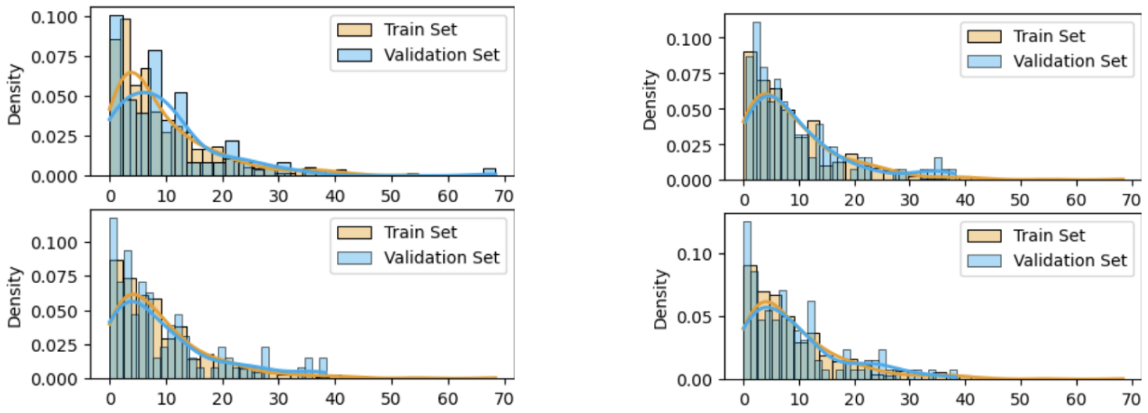
In the following section, we present the results of our experiments with regard to our hypotheses as defined is Section 3. Firstly, we conduct a descriptive analysis to determine whether the train and validation data more closely align with stratified sampling, thereby better reflecting the underlying distribution in each fold. Secondly, we extract the best hyperparameter combination of a repetition and examine whether stratified and unstratified cross-validation result in the same best chosen hyperparameter combination. Thirdly, we evaluate the final performance of the best hyperparameters combination per repetition on the test data based on the MSE, MAE and R^2. Lastly, we calculate the estimation error of the MSE to see if the estimation itself can be improved by stratification. Please note that for all plots we use

$$transformation\_train\_noise\_groups$$

as abbreviation on the x-axis for the 24 unique experimental parameter combinations ω. For example "$identity$_200_0_5" indicates that the data is not transformed, comprises 200 data observation with no noise in the data generation process and the group size of the stratification approach is set to 5.

5.1 Descriptive analysis

First, we investigate our stratification approach in a descriptive analysis. Through visual inspection (see Figure 7), we can already see that in the case of stratified sampling the



(a) With Random Sampling

(b) With Stratified Sampling

Figure 7: Visual inspection of a) unstratified random sampling and b) stratified sampling of two train test splits respectively.

train and validation data seem to align better. To analyze this observation analytically, we calculate the intersection of the train and validation data within cross-validation for each unique experimental parameter combination ω for stratification and random sampling in our experiments, resulting in $0 < Int_{stratified}, Int_{unstratified} < 1$. The ordered and grouped boxplots in Figure 8 clearly show that the stratified intersection per experimental parameter combination is on average higher than in the unstratified case. The pattern of the training size on the x-axis indicates that the intersection for $n\_train = 200$ results in a lower intersection than for $n\_train = 1000$ overall. Moreover, the logarithmized data for each training size results in a lower intersection respectively.

Comparing the intersection with stratification versus random sampling in more detail, the difference $Int_{diff} = Int_{stratified} - Int_{unstratified}$ is always positive for each unique parameter combination ω as the intersection for stratification is on average higher in each case. Table 4 in the Appendix shows the exact values of the differences Int_{diff} per ω. High differences of the means indicate that the train-validation-overlap with stratification compared to random sampling is higher in magnitude. For $n\_train = 200$ the differences Int_{diff} are higher than for $n\_train = 1000$, supporting our hypothesis that stratification can especially help in case of small datasets. Looking at each training data size independently, the logarithmized data has smaller differences Int_{diff} compared to the other transformation. Contrary to our assumption this indicates that our stratification approach does not help especially in the case of imbalanced data such as the logarithm. Furthermore, the differences of the standard deviations per experimental parameter combination $SD_{diff} = SD_{stratified} - SD_{unstratified}$ are always negative, indicating a smaller variation of the intersection for stratification.

We also test statistically if the training and validation data are from the same continuous distribution with a Kolmogorov–Smirnov Test (Massey Jr, 1951). Figure 13 in the Appendix shows that the null hypothesis for stratified and unstratified sampling is accepted, indicating that the train-validation data comes from the same distribution. However, for stratification, the p-value is higher and almost equal to 1. A paired t-test (Fahrmeir et al., 2013) per experimental parameter combination results in p-values smaller than 0.0001, indicating

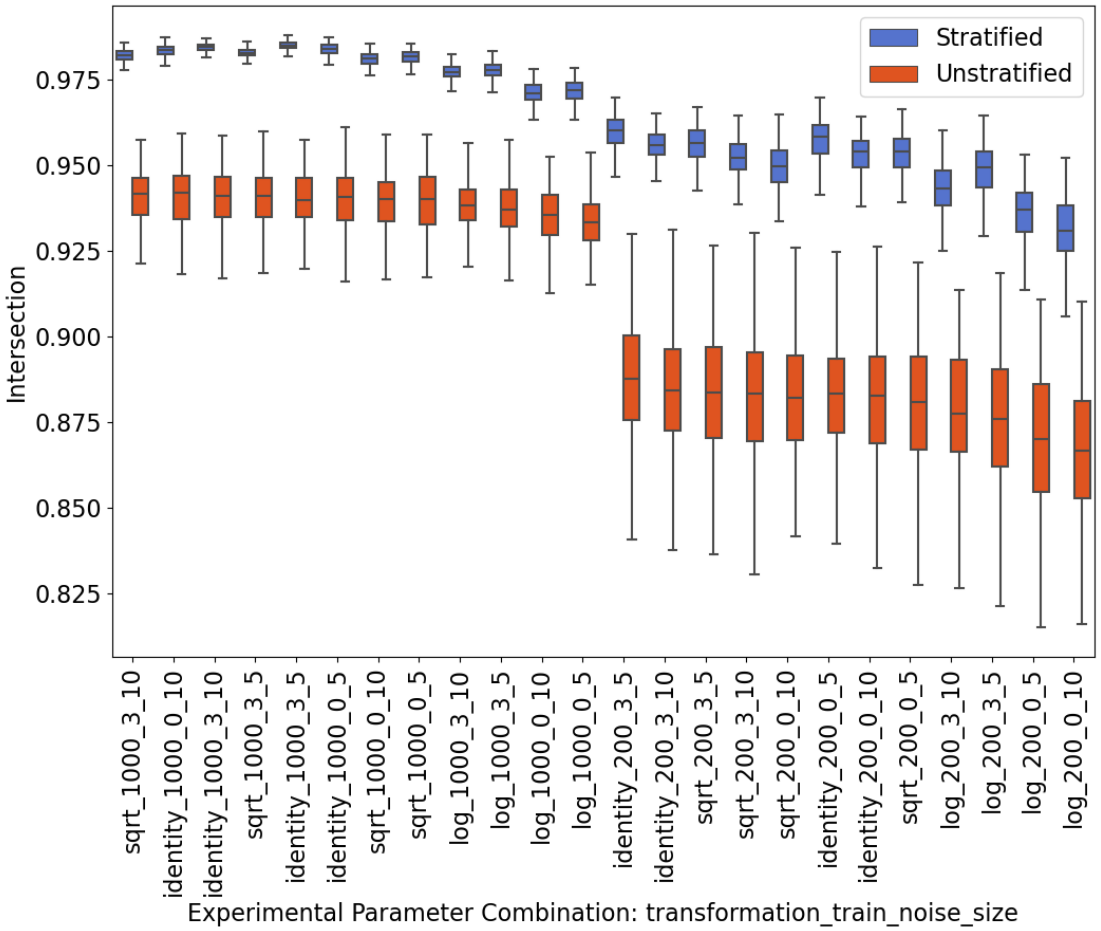


Figure 8: Grouped and ordered boxplots of the intersection of the train and validation data within cross-validation for each experimental parameter combination. The boxplots are ordered by the mean of the unstratified intersection. For better visualization, outliers are not depicted here.

that the differences between the intersection of the stratified and unstratified sampling are significantly different. The statistical tests support our descriptive analysis that the two distributions of train and validation set are more similar in the case of stratification.

5.2 Analysis of best hyperparameters

Now we want to analyze the best hyperparameters which are chosen over all iterations $i = 1, ..., 200$ by Random Search for each repetition r, defined as:

$$\lambda_{best}^{(r)} = \arg\min_{\lambda} MSE_{VAL}^{(r)} \text{ with } MSE_{VAL}^{(r)} = \arg\min_{\lambda} \{MSE_{VAL}^{(1,r)}, ..., MSE_{VAL}^{(200,r)}\}.$$

Comparing the best hyperparameter combinations $\lambda_{best}^{(r)}$ for each model for stratified versus unstratified sampling, Random Forest has overall around 66% different hyperparameter

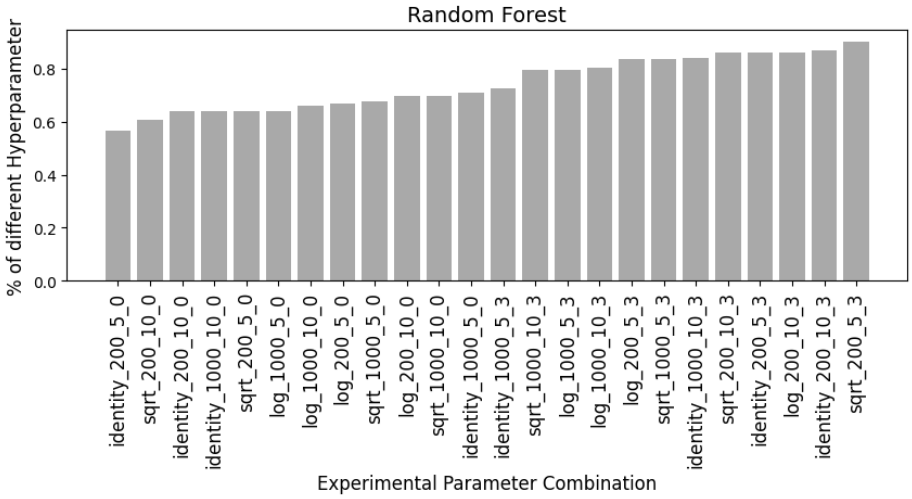


Figure 9: Percentage of different hyperparameter combinations of Random Search comparing the chosen best hyperparameters of stratification and random sampling approach summarized by each unique experimental parameter combination for Random Forest.

combinations of the 4,800 total runs, whereas XGBoost has around 77% different combinations. In the other case, both approaches result in the same best hyperparmeters and the same model performance. Looking at the percentage of different hyperparameter combinations per experimental parameter combination ω for the Random Forest in Figure 14, a clear order with respect to the random noise parameter of the data generation can be seen. With a random noise of three, a higher percentage of different hyperparameter combinations are found compared to no random noise. For XGBoost one can see the same tendency, but overall this pattern is not strictly followed (see Figure 14 in the Appendix).
Furthermore, we investigate the overall frequency of the chosen best hyperparameter values for Random Forest and XGBoost in order to see if any hyperparameters are of special interest in the stratified versus the unstratified case. However, the corresponding counts in Figures 15, 16 and 17 in the Appendix look very similar for both cases. In this way, it is not possible to identify a dominant hyperparameter for each sampling procedure. Nevertheless, the plots give interesting insights about the overall prevalence of the hyperparmeter values. For instance, $gamma = 1$ is clearly dominant for XGBoost and $min\_samples\_leaf = 1$ for Random Forest.

5.3 Final performance of best hyperparameters

Building on the results of the previous Section 5.2, it is important to note, that if the best hyperparameters $\lambda_{best}^{(r)}$ are the same in the stratified and unstratified case, the final performance in the evaluation on the test set is also equal. We have not filtered out the same hyperparameter configurations in the following analysis as we want to provide a full picture including all results of each repetition. Measuring the final performance by $MSE_{TEST}^{(r)}(\lambda_{best}^{(r)}, \omega^{(r)})$ and $MAE_{TEST}^{(r)}(\lambda_{best}^{(r)}, \omega^{(r)})$ for each repetition r on the test set, it

is important to analyze the evaluation metrics per transformation. Both depend on the scale of the values of the target variable which is changed by the data transformations. On the other hand, $R^{2(r)}_{TEST}(\lambda^{(r)}_{best}, \omega^{(r)})$ is scale-independent and therefore, the metrics can be compared across transformations. Exemplarily, the grouped and ordered boxplots of R^2 per experimental parameter combination for stratified and unstratified sampling for Random Forest and XGBoost are depicted in Figure 10. The analysis of the other two metrics can be found in the Figures 18 and 19 in the Appendix. Apart from the log-transformed data, for both models and all evaluation metrics, it can be overall seen that the performance is lower for $noise = 3$ compared to no noise. The R^2 measured on the log-transformed data also follows this pattern, whereas for MAE and MSE the performance drops for $n\_train = 200$ compared to $n\_train = 1000$.
Furthermore, the differences of the means and standard deviations between each evaluation metric of the stratified and the unstratified result can be found in the Tables 5, 7, 9, for Random Forest and in the Tables 6, 8, 10 for XGBoost. The differences of the means as well as of the standard deviation are close to zero including both negative and positive values over the different experimental parameter combinations. Therefore, there seems no clear better approach comparing stratification versus random sampling with regard to the final performance of the best hyperparmeters.

5.4 Error estimator of MSE

Although the descriptive analysis shows a better alignment of the train-validation split, the final performance does not show any notable improvement with stratification. It seems like overall the same or similar hyperparameters are chosen by both sampling approaches resulting in a similar final performance. For this reason, we further investigate the error of the estimator of the MSE to see if stratification improves our estimator from a more theoretical point of view. In general, the error of an estimator θ can be calculated by (Allen et al., 2009):

$$\mathrm{MSE}(\hat{\theta}) = \mathrm{E}\left[(\hat{\theta} - \theta)^2\right] \tag{1}$$

Following the idea of the *bias-variance decomposition*, it is possible to split the error into the square of the bias plus the variance:

$$\mathrm{MSE}(\hat{\theta}) = (\mathrm{Bias}(\hat{\theta}, \theta))^2 + \mathrm{Var}(\hat{\theta}) \tag{2}$$

This implies that when estimating a parameter, the error can arise from either excessive sensitivity to fluctuations in the data (high variance) or from a systematic tendency to consistently deviate from the true value (high bias)(James et al., 2013). In our setting with with $i = \{1, ..., 200\}$ iterations per repetition r, the error of the MSE is defined based on equation 1:

$$MSE^{(r)}_{ERROR} = \frac{1}{I}\sum_{i=1}^{I}(MSE^{(i,r)}_{VAL}(\lambda^{(i,r)}, \omega^{(r)}) - MSE^{(i,r)}_{TEST}(\lambda^{(i,r)}, \omega^{(r)}))^2 \tag{3}$$

$$\text{with } \hat{\theta} = MSE^{(i,r)}_{VAL}(\lambda^{(i,r)}, \omega^{(r)}) \text{ and } \theta = MSE^{(i,r)}_{TEST}(\lambda^{(i,r)}, \omega^{(r)}) \tag{4}$$

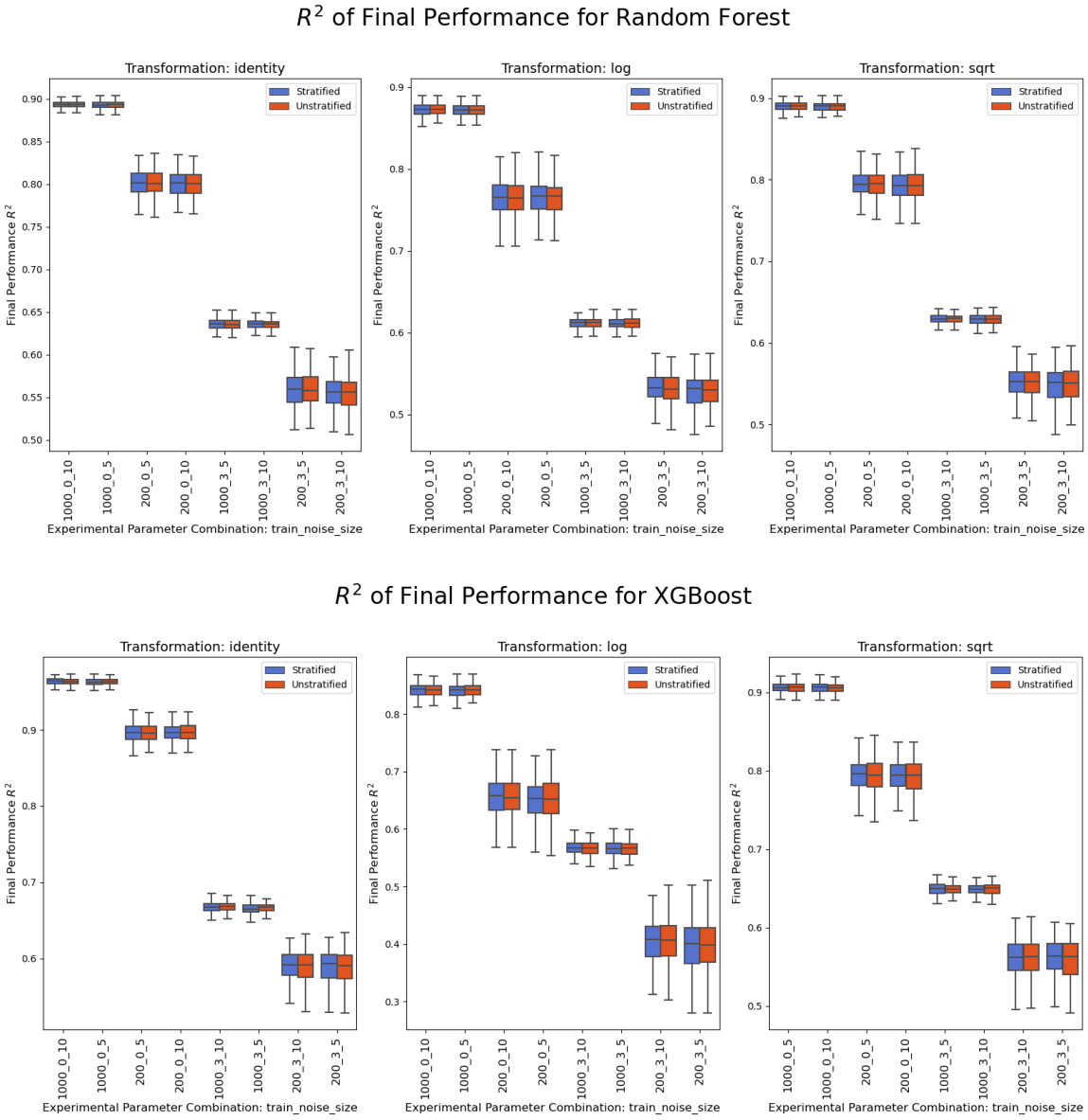


Figure 10: Final Performance measured by the $R^{2(r)}_{TEST}(\lambda^{(r)}_{best}, \omega^{(r)})$ for XGBoost and Random Forest grouped by stratified and unstratified sampling for each experimental parameter combination. Note that for better visualization the outliers are removed and ordered by the unstratified mean.

The $MSE^{(r)}_{ERROR}$ can be calculated separately for the stratified and unstratified cross-validation. Hence, we get 200 results for each repetition r defined as $MSE^{(r)}_{ERROR,stratified}$ and $MSE^{(r)}_{ERROR,unstratified}$ respectively, resulting in the error differences

$$MSE^{(r)}_{error\_difference} = MSE^{(r)}_{ERROR,stratified} - MSE^{(r)}_{ERROR,unstratified} \tag{5}$$

$$SD^{(r)}_{error\_difference} = SD^{(r)}_{ERROR,stratified} - SD^{(r)}_{ERROR,unstratified} \tag{6}$$

In Figure 11, the 200 repetitions performed for each parameter combination are summarized per model and data transformation in form of grouped and ordered boxplots. Similarly to the descriptive analysis of the intersection in chapter 5.1, all boxplots show the same pattern that the estimation error of the MSE is bigger for $n\_train = 200$ than for $n\_train = 1000$. It is also directly visible that the stratified mean and median of the error of the MSE are smaller than for random sampling in every case. For this reason all differences $MSE_{error\_difference}$ are negative (see Table 11 and 12). Looking at the $SD_{error\_difference}$, most values are negative, indicating a smaller variance of the error estimator for stratification. Furthermore, for the identity and square root transformations one can also see that the difference of the means for $n\_train = 200$ is bigger in magnitude, indicating that stratification has an increased improvement on the MSE_{error} in this case. Comparing the overall error of the MSE per model, the aggregated boxplots in Figure 20 in the Appendix show that the error and its variance are in general smaller for the Random Forest than for XGBoost for every data transformation.

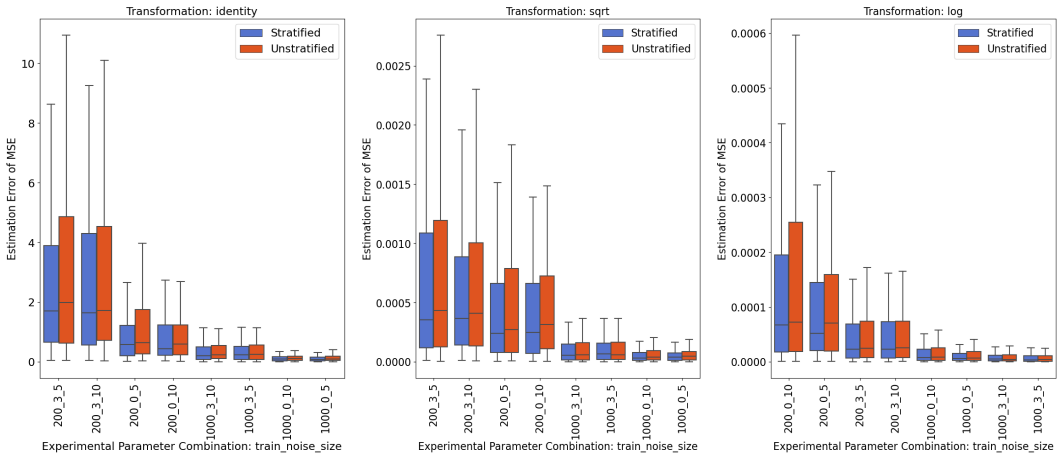


Figure 11: Estimation Error of the MSE of Random Forest

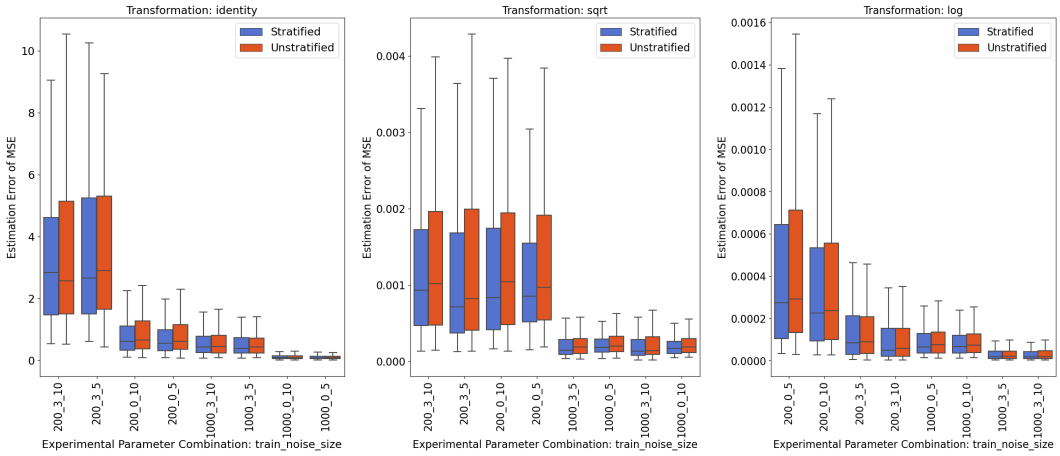


Figure 12: Estimation Error of the MSE of XGBoost

6. Conclusion

The descriptive analysis shows that the overlap of the train and validation data within cross-validation can be increased by stratification. However, in 34% of the runs for Random Forest and in 23% for XGBoost, the models result in the exact same best hyperparameter combinations and subsequently in the same final performance. With regard to our first hypothesis, the final performance between stratified and unstratified sampling is really similar showing no superiority of the two approaches. On the other hand, the error of MSE as well as its variance is reduced through stratification indicating a better estimation. In conclusion, although the estimation slightly improves, the choice of the best hyperparameters in Random Search and the final performance stay rather similar when comparing stratification with random sampling. Looking at the different experimental parameters, stratification seems to help especially with smaller dataset. However, no notable effect with regard to noise, transformation or group size can be found in our experiments. We believe that stratification should be further investigated and implemented as optional sampling approach within cross-validation for hyperparameter tuning and model selection. Our results show, that there is the chance of improving the results especially for small datasets. Further investigation is needed to analyze different datasets, models and stratification approaches.

7. Limitations & Outlook

In our experiments we are limited to the artificially generated Friedman dataset and the two models Random Forest and XGBoost. Within this framework, it is possible to implement the common approach of early stopping for XGBoost (Makarova et al., 2021). This could enhance the effect of stratification as based on each stratified cross-validation there is the option to stop a stagnating training procedure resulting in a possible faster convergence with less overfitting and consequently less variance of the estimator (Lever et al., 2016; Caruana et al., 2000). For time reasons we restricted our experiments to the basic implementation of the two models with a limited explorative hyperparameter grid of Random Search. Additional hyperparmeters and further models could be tested, whereby especially models which are sensitive to their hyperparameters should be considered.

Furthermore, exchanging the Friedman dataset with a real-world dataset would be another interesting modification. Especially a further analysis of real-world imbalanced data should be considered. In our experiments we are not able to show that stratification is especially helpful for the more imbalanced square root and log-transformed data compared to the untransformed data. However, the investigation of stratification on data with outliers and multiple modes is still missing. As we do not have the ground truth in the case of real-world data anymore, the general experimental set-up needs to be adopted. With regard to our stratification approach based on group-sized discretization, other implementations considering the exact ordering of the target values or incorporating upsampling in the algorithm are further possible changes. Moreover, a further analysis of the bias-variance decomposition of the error of the MSE could answer the question if stratification especially reduces the bias or variance of the estimator.

In general, also a theoretical analysis of stratification can give helpful insights how to address issues arising from random sampling within cross-validation and in which way stratification

can help in this case. Apart from using stratification for hyperparmeter tuning, the idea could be also incorporated in bootstrapping algorithms (Weinshel et al., 2022; D'Amato et al., 2012). Here it would be particularly interesting to analyze if the bias or variance of the resulting statistics of interest can be reduced. Another further idea would be to not only stratify on the the target values, but also applying this concept on the feature variables (Yang et al., 2021; Chen et al., 2018).

8. Contributions

Nadja focused on the evaluation and interpretation of the results. Anne led the implementation of the experimental setup, ensuring consistency and reproducibility. Both contributed to the literature research as well as designing the stratification algorithm and the experimental set-up.

References

G. Allen, J. Nabrzyski, E. Seidel, G.D. van Albada, J. Dongarra, and P.M.A. Sloot. *Computational Science – ICCS 2009: 9th International Conference Baton Rouge, LA, USA, May 25-27, 2009 Proceedings*. Computational Science - ICCS 2009: 9th International Conference, Baton Rouge, LA, USA, May 25-27, 2009 ; Proceedings. Springer, 2009. ISBN 9783642019722. URL https://books.google.de/books?id=R_eSs3BzufsC.

Ozturk Anil. Stratification on regression problems. https://medium.com/@nlztrk/stratification-on-regression-problems-e36b3b866079, 2022. [Accessed: 2024-03-17].

Leo Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996.

Leo Breiman. Random forests. *Machine Learning*, 45:5–32, 10 2001. doi: 10.1023/A:1010950718922.

Michael W Browne. Cross-validation methods. *Journal of mathematical psychology*, 44(1): 108–132, 2000.

Patrick Brus. Data Imbalance in Regression. https://towardsdatascience.com/data-imbalance-in-regression-e5c98e20a807, 2021. [Accessed: 2024-03-16].

Rich Caruana, Steve Lawrence, and C Giles. Overfitting in neural nets: Backpropagation, conjugate gradient, and early stopping. *Advances in neural information processing systems*, 13, 2000.

N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer. Smote: Synthetic minority over-sampling technique. *Journal of Artificial Intelligence Research*, 16:321–357, June 2002. ISSN 1076-9757. doi: 10.1613/jair.953. URL http://dx.doi.org/10.1613/jair.953.

Renjie Chen, Ning Sun, Xiaojun Chen, Min Yang, and Qingyao Wu. Supervised feature selection with a stratified feature weighting method. *IEEE Access*, 6:15087–15098, 2018.

Tianqi Chen and Carlos Guestrin. XGBoost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, pages 785–794, New York, NY, USA, 2016. ACM. ISBN 978-1-4503-4232-2. doi: 10.1145/2939672.2939785. URL http://doi.acm.org/10.1145/2939672.2939785.

Zherebtso Danil. Verstack documentation: verstack 3.9.2. https://verstack.readthedocs.io/en/latest/#stratified-continuous-split, 2023. [Accessed: 2024-03-19].

NA Diamantidis, Dimitris Karlis, and Emmanouel A Giakoumakis. Unsupervised stratification of cross-validation for accuracy estimation. *Artificial Intelligence*, 116(1-2):1–16, 2000.

Valeria D'Amato, Steven Haberman, and Maria Russolillo. The stratified sampling bootstrap for measuring the uncertainty in mortality forecasts. *Methodology and Computing in Applied Probability*, 14:135–148, 2012.

Ludwig Fahrmeir, Thomas Kneib, Stefan Lang, Brian Marx, Ludwig Fahrmeir, Thomas Kneib, Stefan Lang, and Brian Marx. *Regression models*. Springer, 2013.

Jerome H. Friedman. Multivariate Adaptive Regression Splines. *The Annals of Statistics*, 19(1):1 – 67, 1991. doi: 10.1214/aos/1176347963. URL `https://doi.org/10.1214/aos/1176347963`.

Mateusz Góralczyk, Anna Michalak, and Paweł Śliwiński. Drill bit deterioration estimation with the random forest regressor. *IOP Conference Series: Earth and Environmental Science*, 942:012013, 11 2021. doi: 10.1088/1755-1315/942/1/012013.

Haibo He and E.A. Garcia. Learning from imbalanced data. *Knowledge and Data Engineering, IEEE Transactions on*, 21(9):1263–1284, Sept 2009. ISSN 1041-4347. doi: 10.1109/TKDE.2008.239.

Roman Hornung, Malte Nalenz, Lennart Schneider, Andreas Bender, Ludwig Bothmann, Bernd Bischl, Thomas Augustin, and Anne-Laure Boulesteix. Evaluating machine learning models in non-standard settings: An overview and new findings, 2023.

Gareth James, Daniela Witten, Trevor Hastie, Robert Tibshirani, et al. *An introduction to statistical learning*, volume 112. Springer, 2013.

Sanyam Kapoor and Valerio Perrone. A simple and fast baseline for tuning large xgboost models, 2021.

Will Koehrsen. Hyperparameter tuning the random forest in python. https://towardsdatascience.com/hyperparameter-tuning-the-random-forest-in-python-using-scikit-learn-28d2aa77dd74, 2018. Accessed: 2024-03-19.

Ron Kohavi. A study of cross-validation and Bootstrap for accuracy estimation and model selection. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1137–1143. Morgan Kaufmann, 1995.

Jake Lever, Martin Krzywinski, and Naomi Altman. Points of significance: model selection and overfitting. *Nature methods*, 13(9):703–705, 2016.

Anastasia Makarova, Huibin Shen, Valerio Perrone, Aaron Klein, Jean Baptiste Faddoul, Andreas Krause, Matthias Seeger, and Cedric Archambeau. Overfitting in bayesian optimization: an empirical study and early-stopping solution. In *2nd Workshop on Neural Architecture Search (NAS 2021 collocated with the 9th ICLR 2021)*, 2021.

Frank J Massey Jr. The kolmogorov-smirnov test for goodness of fit. *Journal of the American statistical Association*, 46(253):68–78, 1951.

Aga Maulana, Farassa Rani Faisal, Teuku Rizky Noviandy, Tatsa Rizkia, Ghazi Mauer Idroes, Trina Ekawati Tallei, Mohamed El-Shazly, and Rinaldi Idroes. Machine learning approach for diabetes detection using fine-tuned xgboost algorithm. *Infolitika Journal of Data Science*, 1(1):1–7, Aug. 2023. doi: 10.60084/ijds.v1i1.72. URL https://heca-analitika.com/ijds/article/view/72.

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

Sebastian Raschka. Model evaluation, model selection, and algorithm selection in machine learning, 2020.

C. Lowe Scott. Stratified validation splits for regression problems. https://scottclowe.com/2016-03-19-stratified-regression-partitions/, 2016. 2023-03-18.

Konstantinos Sechidis, Grigorios Tsoumakas, and Ioannis Vlahavas. On the stratification of multi-label data. In *Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2011, Athens, Greece, September 5-9, 2011, Proceedings, Part III 22*, pages 145–158. Springer, 2011.

Marin Stoytchev. Xgboost hyperparameter tuning. https://meanderingscience.com/xgboost-hyperparameter-tuning/, 2020. [Accessed: 2024-03-21].

Fadi Thabtah, Suhel Hammoud, Firuz Kamalov, and Amanda Gonsalves. Data imbalance in classification: Experimental evaluation. *Information Sciences*, 513:429–441, 2020.

Luís Torgo, Rita P Ribeiro, Bernhard Pfahringer, and Paula Branco. Smote for regression. In *Portuguese conference on artificial intelligence*, pages 378–389. Springer, 2013.

Jan N. van Rijn and Frank Hutter. Hyperparameter importance across datasets. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery amp; Data Mining*, KDD '18. ACM, July 2018. doi: 10.1145/3219819.3220058. URL http://dx.doi.org/10.1145/3219819.3220058.

Sofia Visa and Anca Ralescu. Issues in mining imbalanced data sets-a review paper. In *Proceedings of the sixteen midwest artificial intelligence and cognitive science conference*, volume 2005, pages 67–73. sn, 2005.

Sarah Weinshel, David J Irwin, Panpan Zhang, Daniel Weintraub, Leslie M Shaw, Andrew Siderowf, and Sharon X Xie. Appropriateness of applying cerebrospinal fluid biomarker cutoffs from alzheimer's disease to parkinson's disease. *Journal of Parkinson's disease*, 12(4):1155–1167, 2022.

Yuzhe Yang, Kaiwen Zha, Ying-Cong Chen, Hao Wang, and Dina Katabi. Delving into deep imbalanced regression. *CoRR*, abs/2102.09554, 2021. URL https://arxiv.org/abs/2102.09554.

Appendix A.

```
def create_strat_folds(y,
                       n_folds,
                       n_groups,
                       seed):
    '''
    Function to create continuous folds.
    Inputs:
        y: the target variable
        n_folds: the number of folds
        n_groups: the number of groups
        seed: the seed to be used
    Outputs:
        cv_splits: the indices for the folds
    '''
    # create StratifiedKFold as for classification
    skf = StratifiedKFold(n_splits=n_folds, shuffle=True, random_state=seed)

    # create groups in y with pd.qcut: group-based discretization using different quantiles
    y_grouped = pd.qcut(y, math.ceil(n_groups), labels=False)

    # create fold numbers
    fold_nums = np.zeros(len(y))
    #split(X, y[, groups]): Generate indices to split data into training and test set
    for fold_no, (t, v) in enumerate(skf.split(y_grouped, y_grouped)):
        fold_nums[v] = fold_no

    cv_splits = []

    # iterate over folds, create train & test indices for each fold
    for i in range(n_folds):
        test_indices = np.argwhere(fold_nums==i).flatten()
        train_indices = list(set(range(len(y_grouped))) - set(test_indices))
        cv_splits.append((train_indices, test_indices))

    return cv_splits
```

Listing 1: Stratification algorithm used in the experiments with group-sized discretization inspired by the implementation of Anil (2022)

Model	Computer	Type of parallelization	Average time
Random Forest	Remote	Parallel repetitions	3.88 minutes
		Parallel Random Search	19.20 minutes
	Local	Parallel repetitions	14.18 minutes
		Parallel Random Search	33.46 minutes
XGBoost	Remote	Parallel repetitions	0.19 minutes
		Parallel Random Search	0.61 minutes
	Local	Parallel repetitions	0.59 minutes
		Parallel Random Search	0.80 minutes

Table 3: The average running time per repetition, i.e. for one experimental parameter combination (out of 24) based on the model, device and type of parallelization used.

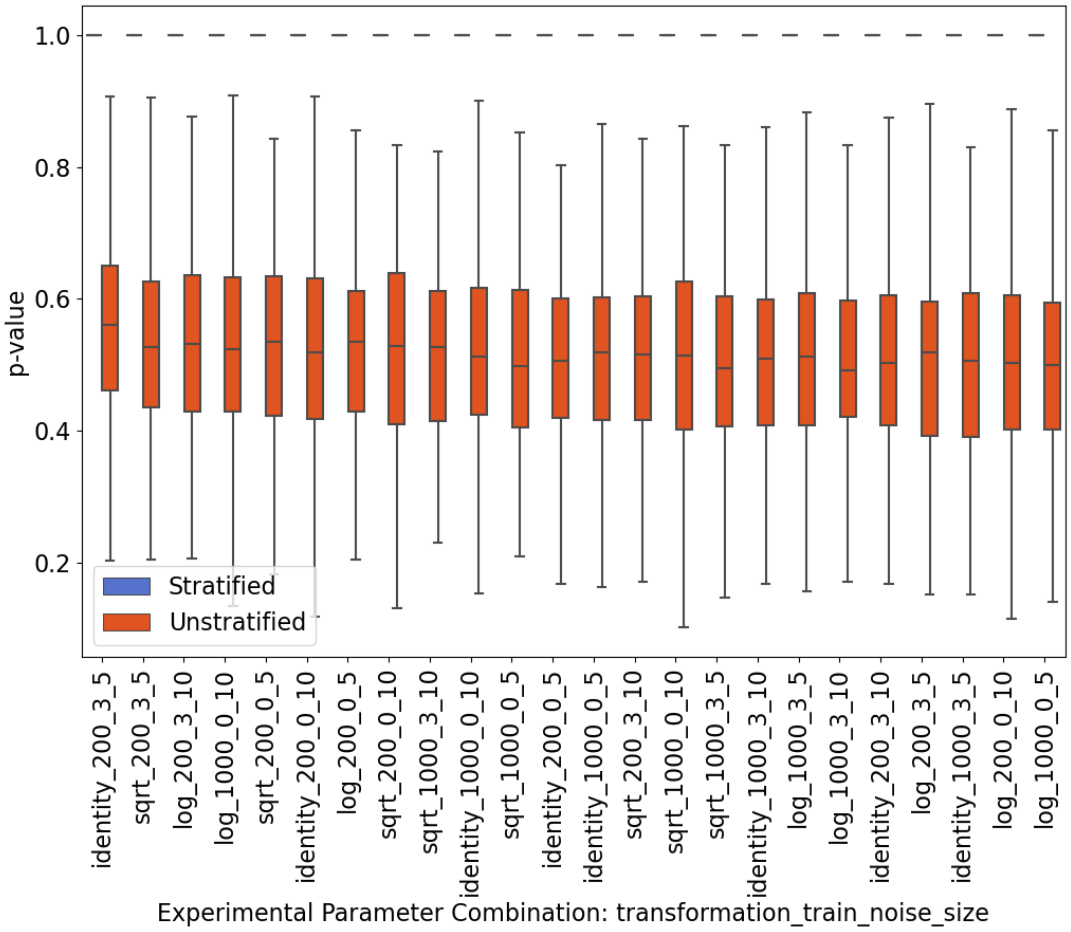


Figure 13: P-values of the Kolmogorov–Smirnov test of the train and validation data within the cross-validation for each experimental parameter combination depicted as grouped box-plots. Note that the p-values in the stratified case are so close to one that it is depicted as dashed line.

Experimental Parameter	Difference Mean Int_{diff}	Difference SD SD_{diff}	Stratified Mean $Int_{stratified}$	Unstratified Mean $Int_{unstratified}$
identity_200_0_5	0.075775	-0.011865	0.956914	0.881139
sqrt_200_0_5	0.072926	-0.012281	0.952688	0.879761
identity_200_3_5	0.072798	-0.013215	0.959407	0.886608
identity_200_0_10	0.072670	-0.014889	0.953106	0.880436
sqrt_200_3_5	0.072221	-0.011109	0.955647	0.883425
identity_200_3_10	0.071967	-0.014395	0.955543	0.883576
log_200_3_5	0.071911	-0.012318	0.947750	0.875839
sqrt_200_3_10	0.069765	-0.014595	0.951797	0.882032
sqrt_200_0_10	0.067591	-0.012417	0.948855	0.881264
log_200_0_5	0.065605	-0.012120	0.935508	0.869903
log_200_0_10	0.065211	-0.010804	0.930428	0.865218
log_200_3_10	0.064872	-0.011612	0.942236	0.877363
identity_1000_3_5	0.044919	-0.007212	0.984842	0.939923
identity_1000_3_10	0.044108	-0.007478	0.984281	0.940173
identity_1000_0_5	0.043961	-0.007285	0.983633	0.939672
identity_1000_0_10	0.042763	-0.007048	0.983260	0.940496
sqrt_1000_3_5	0.042675	-0.007588	0.982679	0.940004
sqrt_1000_0_5	0.042371	-0.007288	0.981574	0.939203
sqrt_1000_0_10	0.041607	-0.006986	0.980856	0.939249
sqrt_1000_3_10	0.041437	-0.006536	0.981961	0.940524
log_1000_3_5	0.040713	-0.006033	0.977720	0.937007
log_1000_3_10	0.039405	-0.006253	0.976997	0.937592
log_1000_0_5	0.038630	-0.004964	0.971823	0.933193
log_1000_0_10	0.036006	-0.005638	0.970957	0.934951

Table 4: Intersection per experimental hyperparameter combination for stratificaiton ($Int_{stratified}$) and unstratified random sampling ($Int_{unstratified}$). The difference of the means is defined as $Int_{diff} = Int_{stratified} - Int_{unstratified}$ and in an analogous manner the difference of the standard deviations is calculated by $SD_{diff} = SD_{stratified} - SD_{unstratified}$. The table is ordered by Int_{diff}.

Transformation	Experimental Parameter	Difference Mean	Difference SD	Stratified Mean	Unstratified Mean
Identity	200_3_10	5.001×10^{-3}	1.637×10^{-2}	14.643	14.638
	1000_0_10	2.310×10^{-3}	1.592×10^{-3}	2.547	2.544
	1000_3_5	1.733×10^{-3}	7.960×10^{-3}	11.966	11.964
	200_0_5	-6.505×10^{-4}	-7.778×10^{-3}	4.752	4.752
	1000_0_5	-8.409×10^{-4}	-4.396×10^{-4}	2.558	2.559
	200_0_10	-2.960×10^{-3}	-4.665×10^{-4}	4.779	4.782
	1000_3_10	-5.436×10^{-3}	-3.354×10^{-3}	11.964	11.969
	200_3_5	-1.319×10^{-2}	-1.727×10^{-3}	14.482	14.495
Logarithm	200_0_5	4.331×10^{-5}	9.401×10^{-5}	0.038	0.038
	1000_3_5	1.688×10^{-5}	-2.211×10^{-5}	0.035	0.035
	1000_3_10	9.139×10^{-6}	-3.465×10^{-6}	0.034	0.034
	1000_0_10	6.524×10^{-6}	-5.990×10^{-7}	0.020	0.020
	1000_0_5	-4.484×10^{-7}	-1.654×10^{-5}	0.020	0.020
	200_3_10	-5.465×10^{-6}	-2.307×10^{-5}	0.042	0.042
	200_0_10	-5.418×10^{-5}	6.003×10^{-5}	0.038	0.038
	200_3_5	-9.559×10^{-5}	-4.216×10^{-5}	0.042	0.042
Square Root	200_3_10	4.934×10^{-4}	2.176×10^{-4}	0.197	0.196
	200_0_10	9.667×10^{-5}	-2.416×10^{-4}	0.096	0.096
	1000_0_5	2.495×10^{-5}	-3.081×10^{-6}	0.051	0.051
	1000_0_10	1.845×10^{-5}	-5.579×10^{-5}	0.051	0.051
	200_0_5	1.633×10^{-5}	4.568×10^{-5}	0.095	0.095
	1000_3_5	2.527×10^{-6}	-1.928×10^{-5}	0.162	0.162
	1000_3_10	-1.607×10^{-5}	-1.058×10^{-4}	0.162	0.162
	200_3_5	-2.071×10^{-4}	1.749×10^{-4}	0.196	0.196

Table 5: Final Performance of Random Forest for stratification versus random sampling measured by the $MSE^{(r)}_{TEST}(\lambda^{(r)}_{best}, \omega^{(r)})$ on the test data. The differences are defined as the stratified values minus the unstratified values. The table is ordered by the differences of the mean per transformation.

Transformation	Experimental Parameter	Difference Mean	Difference SD	Stratified Mean	Unstratified Mean
Identity	1000_3_5	0.038192	0.026224	10.996778	10.958586
	1000_3_10	0.019491	0.020853	10.927986	10.908494
	1000_0_5	0.010136	0.012020	0.896512	0.886376
	200_3_5	0.004468	0.045564	13.549402	13.544934
	200_0_5	-0.000895	0.014166	2.486034	2.486929
	200_0_10	-0.002881	0.004485	2.489376	2.492257
	1000_0_10	-0.007913	0.003494	0.878039	0.885952
	200_3_10	-0.051725	-0.050925	13.464484	13.516209
Logarithm	200_0_5	0.000107	-0.000041	0.055980	0.055873
	1000_0_5	0.000075	0.000283	0.025468	0.025393
	1000_3_5	0.000049	0.000046	0.038535	0.038485
	1000_3_10	-0.000018	-0.000031	0.038424	0.038441
	200_3_10	-0.000037	-0.000244	0.053152	0.053190
	200_0_10	-0.000089	-0.000020	0.025257	0.025346
	200_3_5	-0.000079	0.000053	0.053614	0.053693
	1000_0_10	-0.000281	0.000065	0.043135	0.043416
Square Root	1000_3_5	0.000203	0.000542	0.153001	0.152798
	1000_3_10	0.000162	0.000050	0.152895	0.152732
	1000_0_5	-0.000056	0.000057	0.043090	0.043146
	200_0_5	-0.000072	-0.000660	0.094802	0.094874
	1000_0_10	-0.000281	0.000065	0.043135	0.043416
	$200_31$0	-0.000476	-0.001004	0.191491	0.191968
	200_0_10	-0.000558	-0.000621	0.094583	0.095141
	200_3_5	-0.001368	-0.001748	0.190753	0.192121

Table 6: Final Performance of XGBoost for stratification versus random sampling measured by the $MSE^{(r)}_{TEST}(\lambda^{(r)}_{best}, \omega^{(r)})$ on the test data. The differences are defined as the stratified values minus the unstratified values. The table is are ordered by the differences of the mean per transformation.

Transformation	Experimental Parameter	Difference Mean	Difference SD	Stratified Mean	Unstratified Mean
Identity	200_3_5	0.000402	-0.000053	0.558972	0.558570
	1000_3_10	0.000166	-0.000102	0.635683	0.635518
	200_0_10	0.000124	-0.000020	0.799862	0.799738
	1000_0_5	0.000035	-0.000018	0.892852	0.892817
	200_0_5	0.000027	-0.000326	0.800984	0.800956
	1000_3_5	-0.000053	0.000242	0.635604	0.635657
	1000_0_10	-0.000097	0.000067	0.893337	0.893433
	200_3_10	-0.000152	0.000498	0.554100	0.554252
Logarithm	200_3_5	0.001070	-0.000387	0.531665	0.530595
	200_0_10	0.000339	0.000376	0.765001	0.764662
	200_3_10	0.000061	-0.000259	0.527858	0.527797
	1000_0_5	0.000003	-0.000103	0.872060	0.872057
	1000_0_10	-0.000041	-0.000004	0.872622	0.872663
	1000_3_10	-0.000101	-0.000171	0.611341	0.611442
	1000_3_5	-0.000192	-0.000181	0.611381	0.611572
	200_0_5	-0.000271	0.000588	0.763511	0.763782
Square Root	200_3_5	0.000475	0.000401	0.550802	0.550327
	1000_3_10	0.000037	-0.000244	0.629139	0.629102
	1000_3_5	-0.000006	-0.000060	0.628675	0.628681
	200_0_5	-0.000035	0.000099	0.793661	0.793697
	1000_0_10	-0.000040	-0.000121	0.889725	0.889765
	1000_0_5	-0.000054	-0.000007	0.889484	0.889538
	200_0_10	-0.000210	-0.000525	0.791954	0.792165
	200_3_10	-0.001132	0.000496	0.548099	0.549231

Table 7: Final Performance of Random Forest for stratification versus random sampling measured by the $R^{2(r)}_{TEST}(\lambda^{(r)}_{best}, \omega^{(r)})$ on the test data. The differences are defined as the stratified values minus the unstratified values. The table is ordered by the differences of the mean per transformation.

Transformation	Experimental Parameter	Difference Mean	Difference SD	Stratified Mean	Unstratified Mean
Identity	200_3_10	0.001575	-0.001551	0.589975	0.588400
	1000_0_10	0.000331	0.000146	0.963226	0.962894
	200_0_10	0.000121	0.000188	0.895739	0.895618
	200_0_5	0.000037	0.000593	0.895879	0.895841
	200_3_5	-0.000136	0.001388	0.587389	0.587525
	1000_0_5	-0.000425	0.000503	0.962452	0.962876
	1000_3_10	-0.000594	0.000635	0.667217	0.667811
	1000_3_5	-0.001163	0.000799	0.665122	0.666285
Logarithm	200_3_5	0.000887	0.000599	0.397175	0.396288
	200_0_10	0.000837	-0.000385	0.656326	0.655489
	1000_0_10	0.000558	-0.000127	0.841990	0.841432
	200_3_10	0.000437	-0.002978	0.402150	0.401713
	1000_3_10	0.000194	-0.000273	0.566793	0.566599
	1000_0_5	-0.000470	0.001773	0.840669	0.841139
	1000_3_5	-0.000578	0.001164	0.565559	0.566137
	200_0_5	-0.000670	-0.000257	0.649781	0.650451
Square Root	200_3_5	0.003137	-0.003931	0.562027	0.558891
	200_0_10	0.001212	-0.001351	0.794333	0.793121
	200_3_10	0.001093	-0.002305	0.560492	0.559399
	1000_0_10	0.000610	0.000141	0.906204	0.905594
	200_0_5	0.000156	-0.001435	0.793857	0.793700
	1000_0_5	0.000122	0.000124	0.906303	0.906182
	1000_3_10	-0.000367	-0.000069	0.648502	0.648869
	1000_3_5	-0.000470	0.001427	0.648744	0.649214

Table 8: Final Performance of XGBosst for stratification versus random sampling measured by the $R^{2(r)}_{TEST}(\lambda^{(r)}_{best}, \omega^{(r)})$ on the test data. The differences are defined as the stratified values minus the unstratified values. The table is ordered by the differences of the mean per transformation.

Transformation	Experimental Parameter	Difference Mean	Difference SD	Stratified Mean	Unstratified Mean
Identity	1000_0_10	0.000570	0.000234	1.259281	1.258711
	200_3_10	0.000472	0.001661	3.051215	3.050743
	1000_3_5	0.000160	0.000825	2.757368	2.757208
	200_0_5	0.000142	-0.001772	1.736697	1.736555
	1000_0_5	-0.000147	-0.000260	1.263419	1.263565
	200_0_10	-0.000655	0.000139	1.741401	1.742056
	1000_3_10	-0.000679	-0.000428	2.757432	2.758110
	200_3_5	-0.001403	-0.000119	3.034408	3.035812
Logarithm	1000_3_5	0.000052	-0.000009	0.139977	0.139925
	200_0_5	0.000048	0.000195	0.140562	0.140515
	1000_0_10	0.000044	-0.000001	0.102384	0.102341
	200_3_10	0.000025	0.000055	0.154720	0.154695
	1000_0_5	0.000021	-0.000022	0.102581	0.102560
	1000_3_10	0.000016	-0.000005	0.139842	0.139825
	200_0_10	-0.000112	0.000036	0.140499	0.140612
	200_3_5	-0.000178	0.000016	0.154050	0.154228
Square Root	200_3_10	0.000393	0.000149	0.347939	0.347546
	200_0_10	0.000161	-0.000276	0.241233	0.241072
	1000_0_10	0.000053	-0.000101	0.175043	0.174990
	1000_0_5	0.000012	-0.000018	0.175165	0.175153
	200_0_5	0.000004	0.000067	0.240613	0.240610
	1000_3_5	-0.000009	0.000009	0.315354	0.315363
	1000_3_10	-0.000021	-0.000071	0.315310	0.315331
	200_3_5	-0.000192	0.000154	0.347230	0.347421

Table 9: Final Performance of Random Forest for stratification versus random sampling measured by the $MAE^{(r)}_{TEST}(\lambda^{(r)}_{best}, \omega^{(r)})$ on the test data. The differences are defined as the stratified values minus the unstratified values. The table is ordered by the differences of the mean per transformation.

Transformation	Experimental Parameter	Difference Mean	Difference SD	Stratified Mean	Unstratified Mean
Identity	1000_3_5	0.004164	0.003001	2.645429	2.641265
	1000_0_5	0.003348	0.003790	0.736776	0.733428
	1000_3_10	0.002067	0.002808	2.637032	2.634965
	200_3_5	0.000488	0.004389	2.931357	2.930869
	200_0_5	-0.000666	0.004139	1.227642	1.228308
	200_0_10	-0.000738	0.002164	1.229540	1.230278
	1000_0_10	-0.003840	0.000823	0.730832	0.734673
	200_3_10	-0.005299	-0.004395	2.924143	2.929442
Logarithm	1000_0_5	0.000285	0.000343	0.116069	0.115784
	200_0_5	0.000124	-0.000387	0.176090	0.175967
	1000_3_5	0.000111	0.000056	0.148068	0.147956
	200_3_10	-0.000005	-0.000438	0.175422	0.175427
	1000_3_10	-0.000046	-0.000147	0.147715	0.147761
	200_3_5	-0.000065	0.000193	0.175908	0.175974
	1000_0_10	-0.000141	0.000016	0.115696	0.115837
	200_0_10	-0.000345	-0.000117	0.174420	0.174765
Square Root	1000_3_5	0.000203	0.000509	0.306865	0.306663
	1000_3_10	0.000163	0.000136	0.306676	0.306514
	200_0_5	0.000035	-0.000633	0.241197	0.241163
	1000_0_5	-0.000099	0.000188	0.159955	0.160055
	200_3_10	-0.000431	-0.000542	0.343509	0.343940
	1000_0_10	-0.000581	0.000057	0.159955	0.160535
	200_0_10	-0.000678	-0.000443	0.241186	0.241863
	200_3_5	-0.001201	-0.001532	0.343005	0.344205

Table 10: Final Performance of XGBoost for stratification versus random sampling measured by the $MAE^{(r)}_{TEST}(\lambda^{(r)}_{best}, \omega^{(r)})$ on the test data. The differences are defined as the stratified values minus the unstratified values. The table is ordered by the differences of the mean per transformation.

Transformation	Experimental Parameter	Difference Mean	Difference SD	Stratified Mean	Unstratified Mean
Identity	1000_0_10	-5.509×10^{-3}	−0.005 997	0.116 733	0.122 242
	1000_0_5	-6.062×10^{-3}	−0.002 184	0.110 507	0.116 570
	1000_3_5	-7.437×10^{-3}	−0.008 894	0.559 764	0.567 201
	1000_3_10	-1.374×10^{-2}	0.013 535	0.642 586	0.656 325
	200_0_5	-6.681×10^{-2}	−0.048 686	0.786 194	0.853 004
	200_0_10	-6.778×10^{-2}	−0.042 828	0.848 809	0.916 592
	200_3_10	-9.239×10^{-2}	−0.438 139	4.010 839	4.103 226
	200_3_5	-3.970×10^{-1}	−0.822 474	3.689 919	4.086 906
Logarithm	1000_3_5	-1.378×10^{-6}	−0.000 002	0.000 036	0.000 037
	1000_3_10	-1.453×10^{-6}	−0.000 003	0.000 035	0.000 037
	200_3_5	-2.854×10^{-6}	−0.000 005	0.000 161	0.000 164
	200_3_10	-3.606×10^{-6}	−0.000 005	0.000 135	0.000 138
	1000_0_10	-7.034×10^{-6}	−0.000 006	0.000 092	0.000 099
	1000_0_5	-9.718×10^{-6}	−0.000 008	0.000 095	0.000 105
	200_0_10	-3.747×10^{-5}	−0.000 043	0.000 381	0.000 418
	200_0_5	-4.393×10^{-5}	−0.000 055	0.000 472	0.000 516
Square Root	1000_3_10	-1.160×10^{-5}	−0.000 008	0.000 225	0.000 236
	1000_0_10	-2.009×10^{-5}	−0.000 015	0.000 201	0.000 221
	1000_3_5	-2.470×10^{-5}	−0.000 020	0.000 229	0.000 254
	1000_0_5	-2.601×10^{-5}	−0.000 013	0.000 223	0.000 249
	200_3_5	-1.572×10^{-4}	−0.000 246	0.001 444	0.001 601
	200_3_10	-1.580×10^{-4}	−0.000 112	0.001 588	0.001 746
	200_0_5	-1.958×10^{-4}	−0.000 198	0.001 206	0.001 402
	200_0_10	-2.405×10^{-4}	−0.000 238	0.001 263	0.001 504

Table 11: Estimation Error MSE_{ERROR} for stratification versus random sampling for XGBoost per experimental parameter combination. The differences are defined as the stratified values minus the unstratified values. The table is ordered by the differences of the mean per transformation.

Transformation	Experimental Parameter	Difference Mean	Difference SD	Stratified Mean	Unstratified Mean
Identity	1000_0_5	-1.599×10^{-2}	−0.007 391	0.114 048	0.130 043
	1000_3_5	-1.616×10^{-2}	0.016 233	0.415 301	0.431 461
	1000_0_10	-1.797×10^{-2}	−0.014 962	0.118 066	0.136 038
	1000_3_10	-4.466×10^{-2}	−0.038 729	0.405 319	0.449 977
	200_0_10	-1.366×10^{-1}	−0.209 442	0.853 509	0.990 089
	200_0_5	-2.400×10^{-1}	−0.211 622	0.952 482	1.192 480
	200_3_10	-4.846×10^{-1}	−1.134 639	3.186 668	3.671 252
	200_3_5	-5.757×10^{-1}	−1.773 983	3.382 900	3.958 576
Logarithm	1000_3_5	-6.099×10^{-7}	−0.000 002	0.000 010	0.000 010
	200_3_10	-7.329×10^{-7}	0.000 027	0.000 065	0.000 066
	1000_3_10	-1.138×10^{-6}	−0.000 004	0.000 011	0.000 012
	1000_0_10	-1.667×10^{-6}	−0.000 001	0.000 016	0.000 018
	1000_0_5	-2.041×10^{-6}	−0.000 002	0.000 012	0.000 014
	200_3_5	-3.099×10^{-6}	0.000 012	0.000 079	0.000 082
	200_0_5	-1.393×10^{-5}	−0.000 019	0.000 132	0.000 146
	200_0_10	-2.342×10^{-5}	−0.000 028	0.000 159	0.000 183
Square Root	1000_3_5	-5.085×10^{-6}	−0.000 014	0.000 109	0.000 114
	1000_0_5	-6.659×10^{-6}	−0.000 004	0.000 053	0.000 060
	1000_0_10	-6.970×10^{-6}	−0.000 003	0.000 055	0.000 062
	1000_3_10	-7.007×10^{-6}	−0.000 010	0.000 109	0.000 116
	200_3_5	-3.720×10^{-5}	−0.000 003	0.000 850	0.000 888
	200_3_10	-3.997×10^{-5}	0.000 104	0.000 838	0.000 878
	200_0_10	-7.281×10^{-5}	−0.000 068	0.000 435	0.000 508
	200_0_5	-1.235×10^{-4}	−0.000 194	0.000 423	0.000 546

Table 12: Estimation Error MSE_{ERROR} for stratification versus random sampling for Random Forest per experimental parameter combination. The differences are defined as the stratified values minus the unstratified values. The table is ordered by the differences of the mean per transformation.

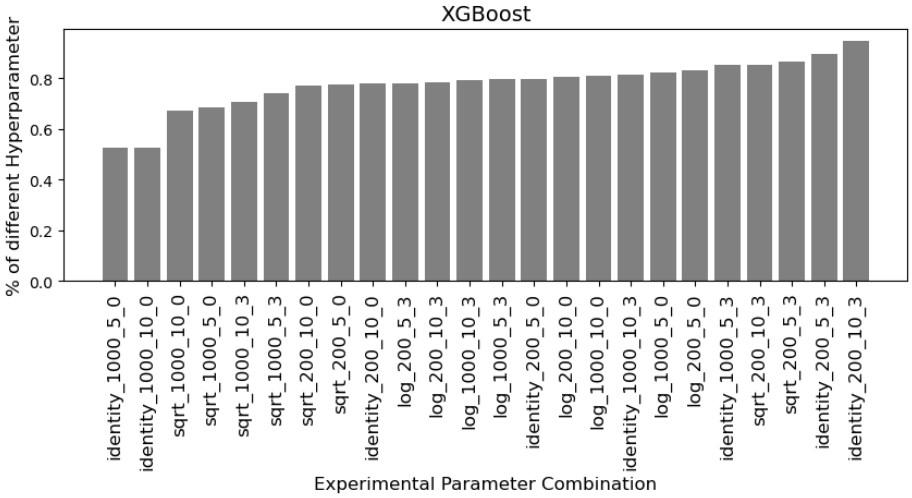


Figure 14: Percentage of different hyperparameter combinations comparing the chosen best hyperparameter combinations of stratification and random sampling of the Random Search summarized by each unique experimental parameter combination for XGBoost.

Transformation	**Model**	**Difference Mean**	**Difference SD**	**Stratified Mean**	**Unstratified Mean**
Identity	RF	-0.191453	-0.747593	1.178537	1.369989
	XGB	-0.082089	-0.295330	1.345669	1.427758
Logarithm	RF	-0.000006	-0.000007	0.000060	0.000066
	XGB	-0.000013	-0.000029	0.000176	0.000189
Square Root	RF	-0.000037	-0.000009	0.000359	0.000396
	XGB	-0.000104	-0.000154	0.000797	0.000902

Table 13: Estimation Error MSE_{ERROR} for stratification versus random sampling aggregated by Random Forest and XGBoost. The differences are defined as the stratified values minus the unstratified values.

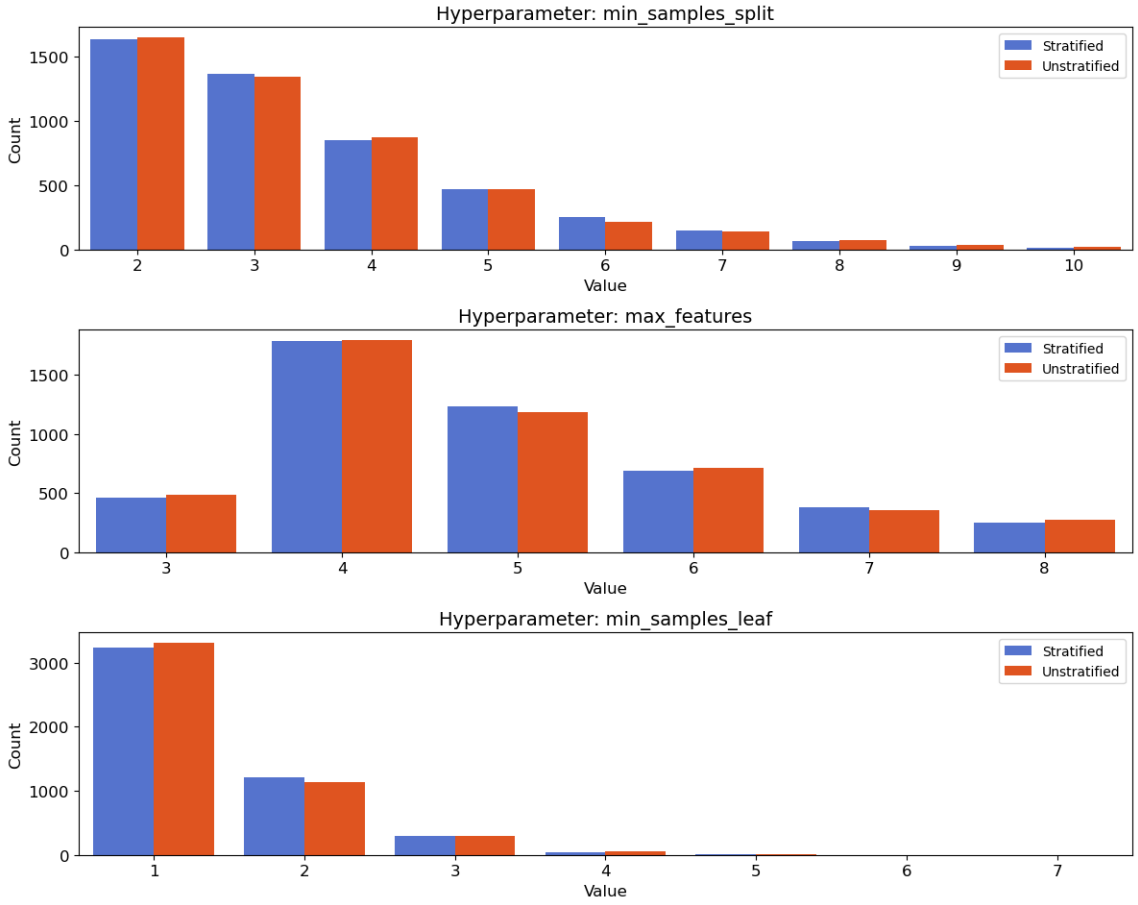


Figure 15: Counts of each hyperparameter value of Random Forest grouped by stratified versus unstratified sampling.

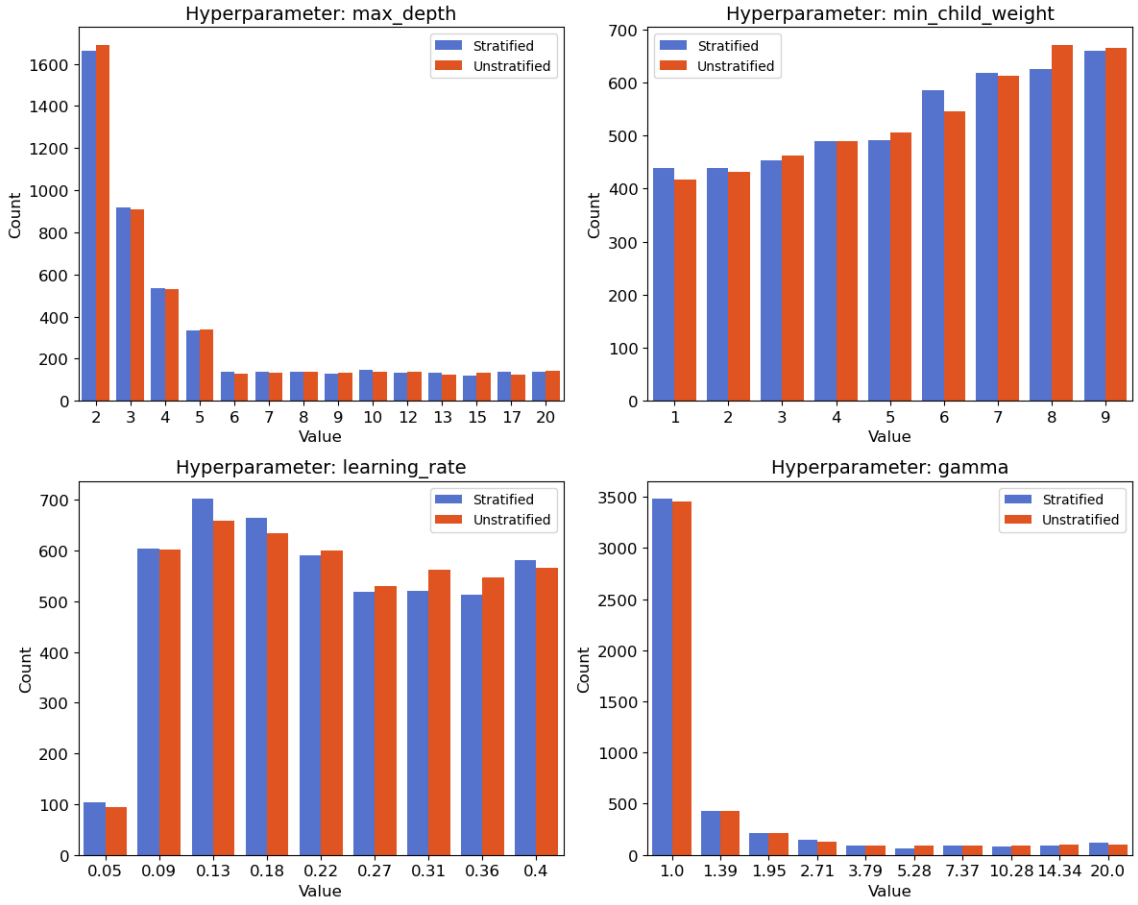


Figure 16: Counts of each discrete hyperparameter value of XGBoost grouped by stratified versus unstratified sampling.

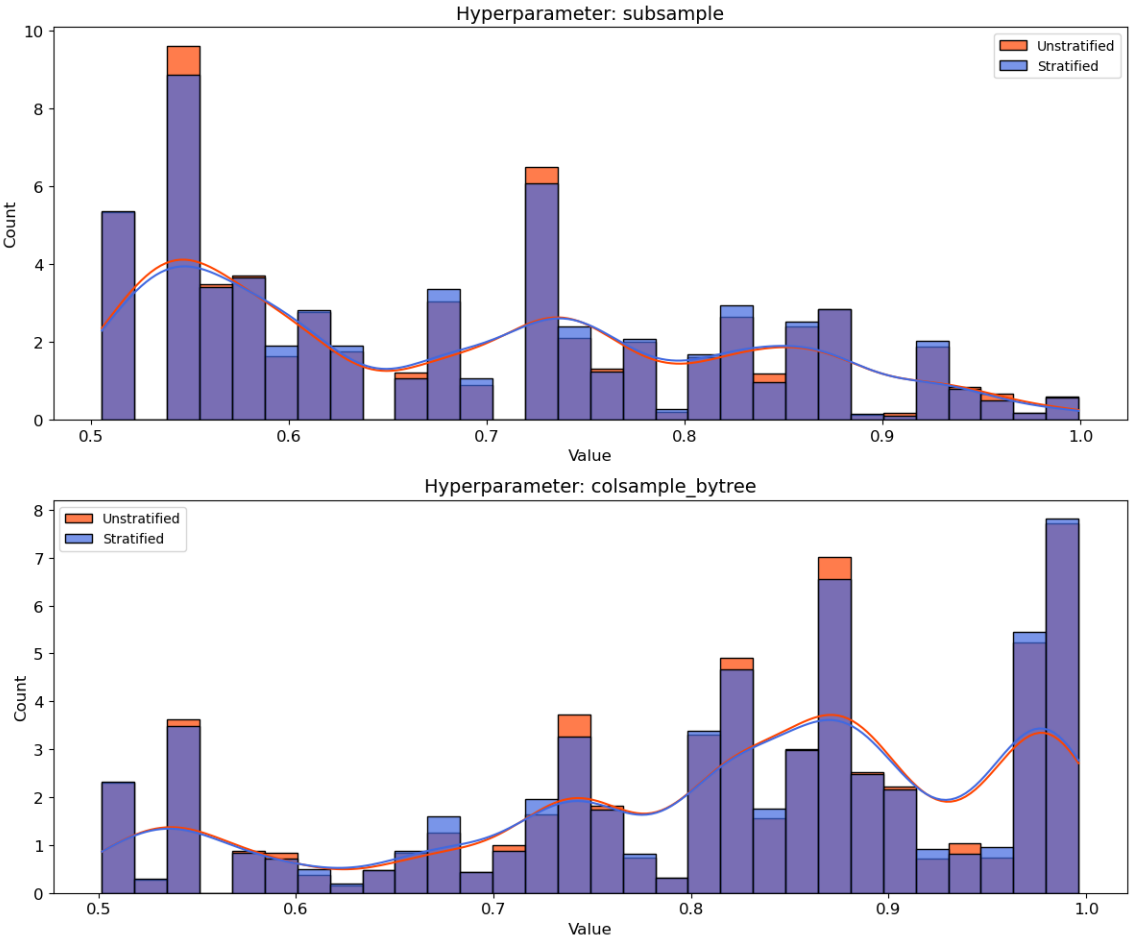


Figure 17: Counts of each randomly sampled hyperparameter value of XGBoost grouped by stratified versus unstratified sampling.

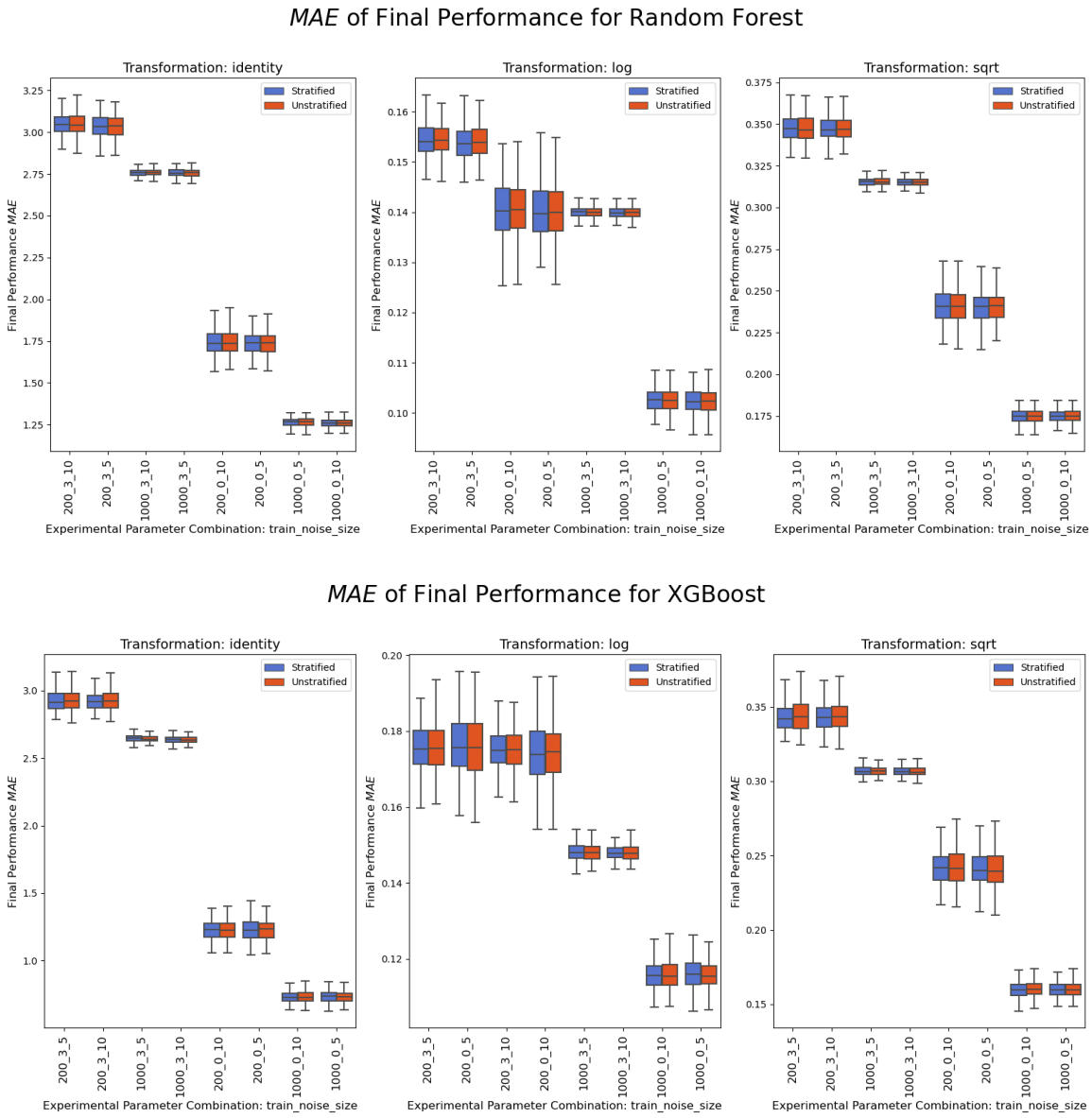


Figure 18: Final Performance measured by the $MAE^{(r)}_{TEST}(\lambda^{(r)}_{best}, \omega^{(r)})$ for XGBoost and Random Forest grouped by stratified and unstratified sampling aggregated by each unique experimental parameter combination. Note that for better visualization the outliers are removed.

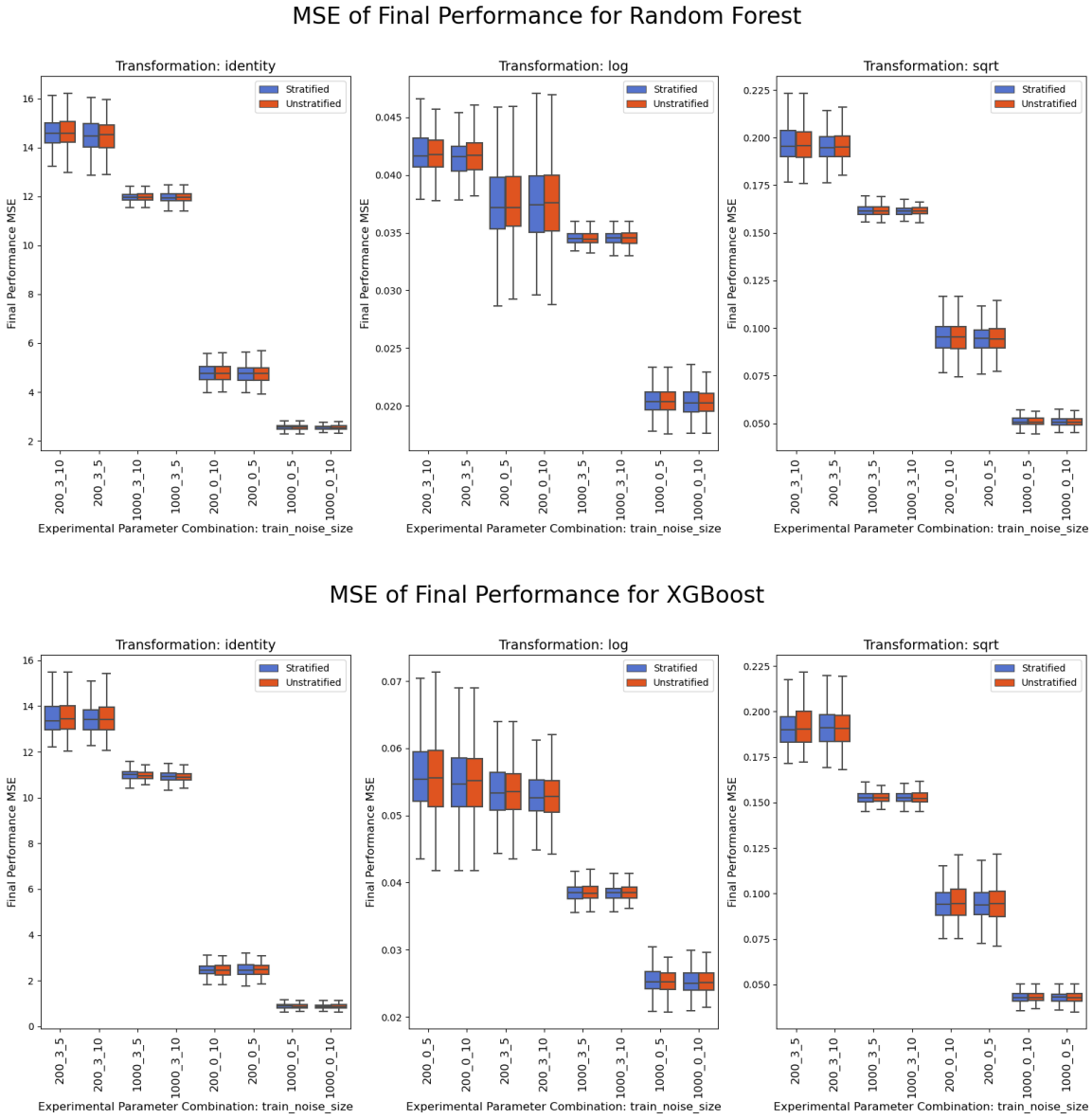


Figure 19: Final Performance measured by the $MSE^{(r)}_{TEST}(\lambda^{(r)}_{best}, \omega^{(r)})$ for XGBoost and Random Forest grouped by stratified and unstratified sampling aggregated by each unique experimental parameter combination. Note that for better visualization the outliers are removed and ordered by the unstratified mean.

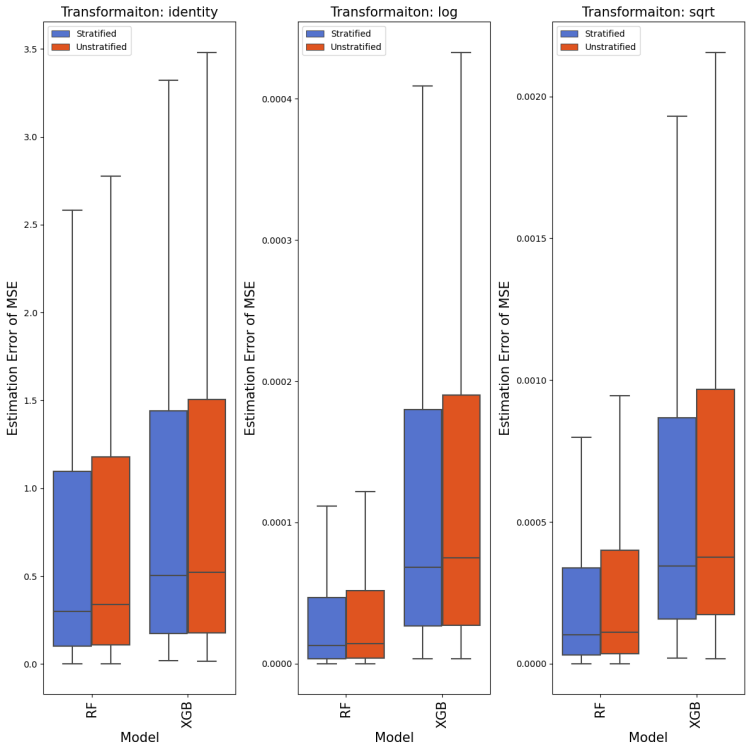


Figure 20: Comparison of the estimation error MSE_{ERROR} of stratification versus random sampling aggregated by all experimental parameter combinations per model.