Università Commerciale "Luigi Bocconi"
PhD School

PhD program in: STATISTICS

Cycle: 34th

Disciplinary Field (code): SECS-S/01 STATISTICS

Machine Learning Approaches for Computing Information Value and Information Density

Advisor: EMANUELE BORGONOVO

PhD Thesis by

Mariusz Budzinski

ID number: 3084829

Academic Year: 2023

1. Abstract

Information value (VoI) analysis is a key component for decision-making supported by quantitative simulations [37]. A key obstacle to the full utilization of VoI is computational efficiency. This thesis examines the use of machine learning approaches to estimating VoI for realistic simulators. We compare the smoothing approaches already introduced, and propose two novelties. First, an approach based on the nearest neighbors. We prove a central limit theorem and then we discuss the automatic selection of the number of neighbors through a LassoLars weighting approach. We also propose a modification of a previously introduced algorithm by using MARS regression. We compare the resulting estimators through a wide range of numerical experiments. We then adapt the algorithms for the estimation of a new quantity, the information density. Experiments show that the algorithms can be successfully modified and one obtains consistent indications about the regional importance of variables, both individually and in groups.

Contents

2. Acknowledgement

At the outset, I would like to express my deepest thanks to the entire academic staff of Bocconi University's department of Decicion Sciences for the knowledge they imparted to me, without which this work would not have been written. In particular, I would like to sincerely thank my supervisor and mentor, Prof. Emmanuele Borgonovo, for his support of my person, joint work and forbearance, without which also this work would not have been written. I would also like to thank my parents, my mother Elżbieta and especially my father Mirosław, for their great and continuous support at every stage of my life and education. Without your involvement in my upbringing and education, I would certainly not be where I am now. I sincerely thank you for that. Finally, I would like to dedicate this work, to my dearest daughter Sophia.

CHAPTER 1
ESTIMATING INFORMATION VALUE: A COMPARISON OF MACHINE LEARNING APPROACHES

3. Introduction

Decision-makers rely on the information provided by quantitative models in an increasing number of applications. As [11] highlights, proper uncertainty quantification plays a central role in making the analysis transparent and better informed. Within uncertainty quantification, factor prioritization, i.e., the identification of the factors that drive uncertainty in model predictions becomes a key task for analysts and decision-makers. For factor prioritization, analysts rely on global sensitivity measures. Indicators in this family range from variance-based ([33],[27]) to distribution-based [5]), to value of information-based indices [25]. Among these sensitivity measures, the value of information is specifically suited to all those applications in which simulation outputs are used to evaluate and compare alternative policies.

The suggestion of value of information (VoI) as a global sensitivity method comes from works such as [12, 13] in the context of medical decision-making. These works highlight that VoI can quantify input importance while taking into account whether the optimal policy changes when we receive perfect information about a feature. They summarize this fact suggesting that VoI provides value as well as decision sensitivity. Reviews of applications of VoI can be found in works such as [6, 21, 41].

Because VoI is a global sensitivity measure, its estimation can be challenging as recently highlighted in [24]. In fact, the definition of VoI requires a so-called double loop of Monte Carlo evaluations. First, one propagates uncertainty in the decision-support model and identifies the nominal optimal alternative. Then, one fixes the input of interest at a given value, samples from the conditional distribution and re-evaluates the model on this conditional sample (say with a sample of size N_{int}). This operation needs to be repeated for several values of the input of interest (say N_{ext}) and for each input, leading to a total double-loop estimation cost of $C^{\mathrm{DL}} = n_X N_{\mathrm{ext}} N_{\mathrm{int}}$ model evaluations, where n_X is the number of model inputs. Several works have then addressed this challenge. The combination of a double loop design and a Gaussian process has been pioneered for the calculation of VoI in [25] and [26]. To reduce computational burden, [35, 38] propose alternative one-loop approaches that require $C^{\mathrm{OL}} = N$ model evaluations. The approach in [35] relies on scatterplot partitioning. In addition, the approach is affected by the curse of dimensionality, and becomes unreliable when calculating the VoI of groups of factors of cardinality greater than two. The approach of [38], instead, considers a non-parametric regression over these scatterplots avoiding the partition size selection, and performs well also for input groups.

In this work, we propose two new approaches. First, we present a nearest neighbors VoI estimator and we prove a corresponding central limit theorem. Based on this proposal, we develop a new algorithmic procedure. We investigate the algorithm in detail and propose a machine learning approach for determining the number of neighbors. The approach relies on creating an artificial dataset of output values associated with each neighborhood and then processing this dataset with a shrinkage procedure to identify the influential neighbors. We also discover that when the VoI estimation concerns a group of features of cardinality greater than 2, the VoI nearest neighbors estimator requires a large sample to obtain reliable estimates, which translates into a very long computation time. This is due to the curse of dimensionality. To overcome this issue we notice that the nearest neighbors approach consists of modeling the neighborhood of a fixed point by taking an average. The generalization of such an approach would be to use a more complex model. Such intuition stands behind local regression methods which are characterized by greater statistical and generalization power. Methods that follow such an intuition are for example local linear regression and the MARS model. This leads us to replace the nearest neighbors approach with such methods when estimating VoI. The approach yields a notable reduction in computational time and makes VoI estimation possible for very large data sets with even millions of observations. Additionally, the VoI estimator becomes more reliable. This is due to the fact local regression modeling has greater generalization power than the nearest neighbors approach.

In order to compare the proposed algorithms with the state-of-the-art solution given by Strong and Oakley [37], we conduct a series of experiments with a case study with 2 scenarios: a multilinear model in which inputs are correlated, but with known analytic solutions for all conditional distributions, and the same model in which inputs are correlated but where sampling from the conditional distributions requires MCMC. The testing is done for different sample sizes and not only for individual inputs but also for input groups up to order 7.

We also demonstrate how to use VoI as a feature selection tool for decision problems when the decision maker needs to choose an option from a set of alternatives. We demonstrate the results of such selection for two case studies.

The remainder of the work is structured as follows. In Section 4, we review extant literature, with a focus on VoI and estimation algorithms. In Section 5, we introduce the proposed method and demonstrate the method in the two case studies, and compare the received results with other existing algorithms in the fourth section. In the end, we summarize obtained results.

4. Literature Review

This work intersects literature streams regarding VoI and machine learning approaches. We provide a synthetic review in this section, focusing on the works and aspects more closely related to our work. In Subsection 4.1 we introduce the notion of VoI and the estimation algorithms by Strong and Oakley [36] and Strong and Oakley [37]. In Subsection 4.2 we give a brief description of the Nearest Neighbor method for regression problems.

4.1. **Information Value.**

The notion of information value has been developed within the realms of decision analysis and probability theory. A first intuition can be found in an unpublished note by Frank P. Ramsey [30]. It has then been formally developed in works such as [29]. As in [25], we consider the formulation of VoI as an expected utility increase, which is also typically used in statistical applications [4].

Let $(\Omega, \mathcal{B}(\Omega), \mathbf{P})$ be a probability space. Consider a decision maker who is using a model to support the selection of an action from a finite set of possible alternatives A. The input and output of this model are represented by random vectors defined on the $(\Omega, \mathcal{B}(\Omega), \mathbf{P})$,

$\mathbf{X}\colon = (\mathbf{X}_1, ..., \mathbf{X}_k)$ and $\mathbf{Y}\colon = \left(Y_{a_1}, Y_{a_2}, ..., Y_{a_{|A|}}\right) = \left(g(\mathbf{X}, a_1), g(\mathbf{X}.a_2), ..., g(\mathbf{X}, a_{|\mathbf{A}|})\right)$, where $g(\mathbf{X}, a)$ represents the model's output for alternative $a \in A$. We also assume that the decision maker has assessed a utility function, $S\colon \mathbb{R} \to \mathbb{R}$, $S(Y_a)$.

The decision-making problem is to select an alternative such that

$$\operatorname*{argmax}_{a\in A} \mathbb{E}(S(Y_a)). \tag{1}$$

In case the analyst receives the posterior information $\mathbf{X}_u = (x_{i_1}, \dots x_{i_l})$, with $u = \{i_1, i_2, \dots, i_l\}$, $l \leq k$ and $u \subseteq (1, 2, \dots, k)$, where k is the number of features, two things can happen. The distribution of the Y_a might change as well as the preferred alternative. The maximization problem becomes

$$\operatorname*{argmax}_{a\in A} \mathbb{E}(S(Y_a(\mathbf{X}))|\mathbf{X}_u). \tag{2}$$

Problem (16) is called *prior expected value of action posterior to perfect information* ([28], p.252). We then have the following definition of VoI:

Definition 4.1 (**Information Value (VoI)**). *The VoI for getting to know* $\mathbf{X}_u$ *is given by*

$$\epsilon^S_{\mathbf{X}_u} = \mathbb{E}(\max_{a\in A} \mathbb{E}(S(Y_a)|\mathbf{X}_u)) - \max_{a\in A} \mathbb{E}(S(Y_a)). \tag{3}$$

VoI equals the expected increase in utility registers when the uncertainty about the model inputs or groups of inputs will be removed. Note that (17) is always greater than equal to zero. Also, VoI is null if information about $\mathbf{X}_u$never causes the optimal decision to change. Thus, a null VoI signals that the decision-maker's selection is not affected by uncertainty about X_u. This makes VoI a measure of decision sensitivity, besides of value sensitivity. In order to be able to use VoI in practical applications, efficient estimation methods are needed.

The efficient estimation of VoI has attracted notable interest in the literature. Coyle and Oakley [9] provides an overview of methods developed in the late 1990's and early 2000's. Of these, the double loop Monte Carlo approach is then subject to intensive numerical investigation in the work of [25]. Heath et al. [19] carry out a recent overview of calculation methods. They compare the Strong and Oakley single-loop (partition-based) estimation method [36], the method of Sadatsafavi et al. [32] and the non-parametric regression method of Strong and Oakley [37]. The results of Heath et al. [19] evidence the efficiency of the non-parametric regression methods of Strong and Oakley. We present these algorithms in detail next.

The starting point of the analysis is a dataset of input-output realizations obtained by propagating uncertainty in the decision support model. Then a scatterplot of N observations $\{x_i, y_a^i\}_{i\in 1:N, a\in A}$ can be formed and used to estimate (17).

Strong and Oakley (2013) approach [36]. We consider first the algorithm proposed in Oakley and Strong [36] (Algorithm 1).

Algorithm 1 Strong and Oakley [2013] [36]

INPUT: Input data set ($S(y_{i,a}), \tilde{x}_i)_{i\in 1,\bar{N}, a\in A}$

PARAMETERS:

- K - cardinality of the partition

OUTPUT: Information value for continuous feature $\tilde{\mathbf{X}}$

```
1: Sort input data set in ascending order by the value of $\tilde{x}$
2: Divide the input data set prepared in such a way into K blocks, each of J observations, $JK = N$.
3: Estimate the first term in the RHS of Eq (17) by
       $$\hat{\epsilon}^S_{\tilde{X},1} = \frac{1}{K}\sum_{k=1}^{K}\max_{a\in A}\frac{1}{J}\sum_{j=1}^{J} S(y^k_{a,j}),$$
   where k indexes block number.
4: Estimate the second term in the RHS of Eq (17) by simple average
       $$\hat{\epsilon}^S_{\tilde{X},2} = \max_{a\in A}\frac{1}{N}\sum_{j=1}^{N} S(y_{j,a})$$
5: Final estimate is given by the
       $$\hat{\epsilon}^S_{\tilde{X}} = \hat{\epsilon}^S_{\tilde{X},1} - \hat{\epsilon}^S_{\tilde{X},2}$$
```

In Step 1, for each alternative one forms the scatterplot between the considered continuous feature and the corresponding utility.

In Step 2 we need to distinguish two cases. In the case X_i is a continuous random variable, we need to partition the horizontal axis of the scatterplot. Usually, one assigns equi-populated partitions. Here a crucial step is the selection of the partition cardinality (K in Algorithm 1). The realizations of Y_a in each partition are averaged to yield an estimate of (16). In the case $\tilde{\mathbf{X}}$ is discrete, there is no need for partitioning. In Step 3, all such estimates are further averaged over the partitions to approximate the first term in the RHS of (17). Step 4 uses a simple average to estimate (15). The output of the algorithm is as in Step 5.

At small sample sizes, the estimates yielded by Algorithm 1 are very sensitive to the choice of the partition cardinality K. At large sample sizes, instead, Strong and Oakley (2013) [36] show a plateau effect in which estimates become insensitive to the actual partition number for a wide range of values of K. [36] then recommend picking a value in this plateau of estimates as the value of VOI to report. The estimation procedure suffers from the curse of dimensionality, as high-dimensional partitions may not be sufficiently populated by data to ensure the required statistical accuracy of the estimates.

An alternative algorithm is then proposed by Strong, Oakley, and Brennan (2014) [37].

Algorithm 2 Strong and Oakley (2014) [37] approach

```
INPUT: Input data set $\{S(y_{i,a}), \tilde{x}_i\}_{i\in\overline{1,N}, a\in A}$,where $\tilde{x}_i \in \mathbb{R}^k$, for $k \in \mathbb{N}$
OUTPUT: Information value for set of features $\tilde{\mathbf{X}}$
1: For each $a \in A$ fit Gaussian Process Regressor or Generalised Additive Model $g(\tilde{x})$ to
   the input data set $\{S(y_{i,a}), \tilde{x}_i\}$
2: The final estimate is given by the following
   $\hat{\epsilon}^S_{\tilde{X}} = \frac{1}{N}\sum_{i=1}^{N} \max_a g(\tilde{x}_i) - \max_a \frac{1}{N}\sum_{i=1}^{N} g(\tilde{x}_i)$
```

The idea is to treat the output of the model as noisy. In Step 1 a non-parametric smoothing procedure such as a *Gaussian Process Regressor* (GPR) [31] or a *Generalised Additive Model* (GAM) [17] is used to smooth the data for each $a \in A$ separately. In Step 2 these smoothed values are inserted in the VOI formula to yield the final estimate. In comparison to Algorithm 1, Algorithm 2 avoids to chose the partition cardinality which makes it attractive to use. The algorithm also is less exposed to the curse of dimensionality. On the other hand, some smoothing procedures may fail to work for a dataset of large dimensionality. Oakley and Strong (2014) [37] recommend the use of GAMs for moderate to big data sets.

4.2. **The Nearest Neighbors approach.** The nearest neighbors algorithm is a non-parametric method developed by Evelyn Fix and Joseph Hodges in 1951 [14] for classification problems and later adapted to regression [2]. Due to its versatility, many different variants of this method were later proposed. In the case of the classification, it is worth mentioning [39], where the Stabilized Nearest Neighbors classifier is introduced. Stability refers to obtaining stable classification results for samples drawn from the same population. In the case of regression, it is worth mentioning [10], where the problem of choosing a metric in the Nearest Neighbors regressor was emphasized and a novel metric is presented which is invariant on affine transformations and gives asymptotically consistent results. In our work, we also took into account the problem of choosing the optimal metric. The way we deal with this problem is described in the next section.

Apart from classification and regression problems, the nearest neighbors approach has found application in a much areas from statistical testing ([7],[23]) to machine learning applications such as pattern recognition [20]. In [20], a probabilistic version of the nearest neighbors method is introduced, which alows for automatic selection of the number of nearest neighbors. In our work, we also recognize this issue but our approach is different see Section 5.

4.3. **Lasso Lars.** The Lasso model (least absolute shrinkage and selection operator) is a linear regression model introduced in geophysics [34] and later developed by Robert Tibshirani [40], who coined the term. Its characteristic is the use of L1 regularization which reduces model dimensionality by decreasing the number of relevant coefficients. The loss function of the Lasso is not differentiable, but a wide variety of techniques from convex analysis and optimization theory have been developed to train the model. These include coordinate descent, subgradient methods, proximal gradient methods, and least-angle regression (LARS). The so-called LassoLars model is the Lasso model which uses the Lars algorithm for fitting. Such an algorithm is extremely efficient, requiring the same order of computation as that of a single least squares fit using the same number of features.

5. A Nearest Neibghour Approach to VoI Estimation

This section is divided into three parts. In Section 5.1, we introduce general results for the estimation, deriving a central limit theorem for the estimates. In Section 5.2, we discuss an approach for automating the selection of neighbors through cross-validation and LassoLars weighting.

5.1. **A Central Limit Theorem Result.**

If $\tilde{\mathbf{X}}$ is a set of continuous random variables, in order to define $\mathbb{E}(Y|\tilde{\mathbf{X}}=x)$, we have to specify what limiting procedure produces the set $\tilde{\mathbf{X}}=x$, to avoid the Borel–Kolmogorov paradox. Define the set $H_x^\epsilon := \{\omega \Big| \|\tilde{\mathbf{X}}(\omega)-x\| < \epsilon\}$ with the assumtion that H_x^ϵ is measurable with $\mathbf{P}(H_x^\epsilon)>0$ for all $\epsilon>0$ [16]. Then we can define $\mathbb{E}(Y|\tilde{\mathbf{X}}=x)$ as

$$\lim_{\epsilon\to 0}\mathbb{E}(Y|H_x^\epsilon).$$

Now focus on the first term on the RHS of (17). We can write

$$\mathbb{E}_{\tilde{\mathbf{X}}}(\max_{a\in A}\mathbb{E}_Y(S(Y_a)|\tilde{\mathbf{X}}=x)) = \mathbb{E}_{\tilde{\mathbf{X}}}(\max_{a\in A}\lim_{\epsilon\to 0}\mathbb{E}_Y(S(Y_a)|H_x^\epsilon)). \tag{4}$$

The question is whether we can approximate (4), selecting a sufficiently small $\delta>0$, so that the quantity

$$\varepsilon_X(\delta) = \mathbb{E}_{\tilde{\mathbf{X}}}(\max_{a\in A}\mathbb{E}_Y(S(Y_a)|H_x^\delta)), \tag{5}$$

tends to ε_X when $\delta\to 0$.

The problem then becomes whether we can exchange the limit and the max operations. We need to specify under which conditions for a function $f(a,\epsilon)$, $f: A\times\mathbb{R}\to\mathbb{R}$ the following holds true:

$$\max_{a\in A}\lim_{\mu\to 0} f(a,\mu) = \lim_{\mu\to 0}\max_{a\in A} f(a,\mu). \tag{6}$$

For this, it is sufficient that $\mu \to f(a,\mu)$ is continuous at $\mu = 0$ for each $a \in A$. To see this consider two cases

- Case 1: A **is a discrete set**
  Define the preferred action $a^* := \operatorname{argmax}_{a\in A} \lim_{\epsilon\to 0} f(a,\epsilon)$, the second best action as a^{**} and $\delta_{a^*,a^{**}} := \frac{1}{2}\lim_{\epsilon\to 0}(f(a^*,\epsilon) - f(a^{**},\epsilon))$. By continuity of f at $\epsilon = 0$, there exists $\zeta > 0$ such that $|f(a,\zeta) - \lim_{\epsilon\to 0} f(a,\epsilon)| < \delta_{a^*,a^{**}}$, for every $a \in A$ uniformly. Hence, $\operatorname{argmax}_{a\in A} f(a,\zeta) = a^*$.
- Case 2: $A = \mathbb{R}^{\mathbf{k}}$
  By the continuity of $f(a,\cdot)$ at 0 we can write

$$\forall \rho > 0 \; \exists \zeta > 0\colon \forall a \in A \qquad \lim_{\epsilon\to 0} f(a,\epsilon) - \rho < f(a,\zeta) < \lim_{\epsilon\to 0} f(a,\epsilon) + \rho$$

Applying the max operator to the above inequality and taking $\rho \to 0$, (6) is true.

Assume further that A is a discrete set. Notice that (17) can be rewritten in the following form

(7)
$$\epsilon^S_{\tilde{\mathbf{X}}} = \mathbb{E}(\max_{a\in A} \mathbb{E}(S(Y_a)|\tilde{\mathbf{X}} = x)) - \max_a \mathbb{E}(S(Y_a)) = \mathbb{E}(\mathbb{E}(S(Y_{a^*_x})|\tilde{\mathbf{X}} = x) - \mathbb{E}(S(Y_{a^{**}})|\tilde{\mathbf{X}} = x))$$
$$= \mathbb{E}(\lim_{\epsilon\to 0} \mathbb{E}(S(Y_{a^*_x}) - S(Y_{a^{**}})|H^\epsilon_x)),$$

where $a^*_x = \operatorname{argmax}_{a\in A} \mathbb{E}(S(Y_a)|\tilde{\mathbf{X}} = x)$ and $a^{**} = \operatorname{argmax}_{a\in A} \mathbb{E}(S(Y_a))$. We can then define

$$\epsilon^S_{\tilde{\mathbf{X}}}(\delta) = \mathbb{E}(\mathbb{E}(S(Y_{a^*_{H^\delta_x}}) - S(Y_{a^{**}})|H^\delta_x)), \tag{8}$$

where $a^*_{H^\delta_x} = \operatorname{argmax}_{a\in A} \mathbb{E}(S(Y_a)|H^\delta_x)$ and by (6)

$$\lim_{\delta\to 0} \epsilon^S_{\tilde{\mathbf{X}}}(\delta) = \epsilon^S_{\tilde{\mathbf{X}}}. \tag{9}$$

The outer expectation in (8) can be approximated by its empirical average so that estimates of $a^*_{H^\delta_x}$, a^{**} can be written as

$$\begin{aligned} \hat{a}^*_{H^\delta_x} &:= \underset{a\in A}{\operatorname{argmax}} \frac{1}{|H^\delta_x|} \sum_{(y'_a,x')|x'\in H^\delta_x} S(y'_a), \\ \hat{a}^{**} &:= \underset{a\in A}{\operatorname{argmax}} \frac{1}{N} \sum_{i=1}^{N} S(y_{i,a}), \end{aligned} \tag{10}$$

respectively, where N is the sample size and $|H^\delta_x|$ is the cardinality of the set H^δ_x. Then we have

(11)

$$\frac{1}{N}\sum_{i=1}^{N} \mathbb{E}_Y(S(Y_{\hat{a}^*_{H^\delta_{x_i}}}) - S(Y_{\hat{a}^{**}})|H^\delta_{x_i}) = \frac{1}{N}\sum_{i=1}^{N} \frac{1}{|H^\delta_{x_i}|} \sum_{(y_{j,\hat{a}^*_{H^\delta_{x_i}}},x_j)|x_j\in H^\delta_{x_i}} (S(y_{j,\hat{a}^*_{H^\delta_{x_i}}}) - S(y_{j,\hat{a}^{**}})).$$

The next step is to change the order of the summation in (11). Because action set A is discrete and $|A| = d$, we can define the following ordered set

$$\begin{aligned} \Delta\hat{S} = \{&S(y_{1,a_1}) - S(y_{1,\hat{a}^{**}}), \cdots, S(y_{1,a_d}) - S(y_{1,\hat{a}^{**}}), \cdots, \\ &S(y_{N,a_1}) - S(y_{N,\hat{a}^{**}}), \cdots, S(y_{N,a_d}) - S(y_{1,\hat{a}^{**}})\}. \end{aligned}$$

Then (11) can be further rewritten as

$$\sum_{i=1}^{N*d} \Delta\hat{S}_i \frac{1}{N} \sum_{x|x_{\lfloor i/d\rfloor}\in H^\delta_x} \frac{1}{|H^\delta_x|} * \delta_{\hat{a}^*{}_{H^\delta_x}=a_{\lfloor i/N\rfloor}} = \sum_{i=1}^{N*d} \Delta\hat{S}_i\hat{\alpha}_i = \hat{\epsilon}^S_{\tilde{\mathbf{X}}}(\delta). \tag{12}$$

In equation (12) we recognize a *central limit theorem* (CLT):

$$\hat{\epsilon}^S_{\tilde{\mathbf{X}}}(\delta) \to_d \mathcal{N}(\epsilon^S_{\tilde{\mathbf{X}}}(\delta), \sum_{i=1}^{N*d} \mathbb{V}(\hat{\alpha}_i\Delta\hat{S}_i)), \tag{13}$$

where $\sum_{i=1}^{N*d} \mathbb{E}(\hat{\alpha}_i\Delta\hat{S}_i) = \epsilon^S_{\tilde{\mathbf{X}}}(\delta)$. Equation (13) states that the estimate in (12) is asymptotically normal.

In the context of numerical applications, the estimation is made on the basis of a finite sample size. This leads to a few complications. First of all it is impossible to take the limit $\delta \to 0$, since it is required that $|H^\delta_x| > 0$, for all x. The direct consequence of this is an estimator whose bias increases as δ increases. On the other hand, the variance of such an estimator decreases with δ. It is the well-know bias-variance trade-off that influences directly accuracy and speed. This is where the problem of δ selection arises as well as the metric to determine H_x^δ. Moreover, when $\tilde{\mathbf{X}}$ is of high dimensionality, the approach is exposed to the curse of dimensionality. Since the number of points falling in a given neighborhood might be small, leading to inaccurate estimates. In our experiments, we found that the proposed estimator is very sensitive to the choices of H_x^δ, especially when feature space is multidimensional.

In our work, we decide to use one of the most popular versions of the nearest neighbors algorithm, namely *k-nearest neighbors* which is to choose the considered number of nearest neighbors and treat them as a neighborhood H_x^δ. In our experiments, we recognize the problem of selecting the number of neighbors and propose a new approach that allows for dynamic and automated selection of the number of neighbors. We explain the method in the next subsection.

5.2. **Cross-validation + LassoLars weighting.** The approach is presented in Algorithm (3) and it is divided into three stages. First, the algorithm performs a search for metric, where we limit ourselves to the weighted Euclidean metric which is equivalent to applying the linear transform of the p-dimensional feature vector $\tilde{x} \to \sqrt{\alpha} \circ \tilde{x}$, where $\alpha \in [0:1]^p$, $\circ$ is the Hadamard product $(\sqrt{\alpha} \circ \tilde{x})_i = \sqrt{\alpha_i}\tilde{x}_i$ and using not weighted Euclidean metric. This are Steps 1-12. The second stage, Steps 13-15, is to find weights for the weighted k- nearest neighbor approach using of the Lasso regression. The last stage, Steps 16-19, is to use the results of the previous stages in order to output the estimates.

Algorithm 3 Information value estimate for continuous input features $\tilde{\mathbf{X}}$ based on nearest neighbour algorithm

INPUT: Input data set ($S(y_i, a), \tilde{x}_i)_{i \in 1:\bar{N}, a \in A}$

PARAMETERS:

- K - number of folds in the validation procedure
- $k_{\max}$ - maximal number of nearest neighbors to consider
- n_{bayes} - number of steps for Bayesian search
- k_{bayes} - maximal number of nearest neighbours to consider during Bayesian search

OUTPUT: information value for $\tilde{\mathbf{X}}$

```
1:  Split randomly K times matrix $\{\tilde{x}_i\}_{i\in\bar{1n}}$ into train/validation subsets $F_i$, $G_i$ such that
    $F_i \sqcup G_i = \{\tilde{x}_j\}_{j\in 1:N}$ for each $i \in 1:K$.
2:  **for** each $a \in A$ **do**
3:      Perform **Bayesian search** to find optimal $\alpha$, Steps 4-10:
4:      For candidate vector $\alpha$ consider the transformed data set $\{\sqrt{\alpha}\circ\tilde{x}_i\}_{1:N}$, where $\circ$ is
        the Hadamard product. Note that train/validation splits do not change but they
        include now transformed features.
5:      **for** each test set $G_i$ **do**
6:          for each $\sqrt{\alpha}\tilde{x}_m \in G_i$ find its $k_{bayes}$ nearest neighbours among $F_i$ and
            create the nearest neighbours matrix $A$ such that
                $$A_{mn} \coloneqq \text{n-th nearest neighbour of } \tilde{x}_m \text{ among } F_i$$
7:          for each $j \le k$ calculate the validation absolute score $CV^a_{ij}$
                $$CV^{a,\alpha}_{ij} \coloneqq \frac{1}{|G_i|}\sum_{\tilde{x}_l\in G_i}|\frac{1}{m}\sum_{0\le m\le j} S(y_{A_{lm}},a) - S(y_l,a)|$$
8:          return $\min_j \text{mean}_i\, CV^{a,\alpha}_{ij}$
9:      **end for**
10:     After $n_{bayes}$ step of Bayesian search select optimal $\alpha^*_a$
                $$\alpha^*_a = \operatorname*{argmin}_{\alpha}\min_j \operatorname*{mean}_i CV^{a,\alpha}_{ij}$$
11:     Using $\alpha^*_a$ repeat Steps 5- 7 using $k_{\max}$ nearest neighbors instead of $k_{bayes}$ and return
        average validation score matrix $CV^{a,\alpha^*_a}_{ij}$ . Denote by
                $$k^*_{a,i} = \operatorname*{argmin}_j CV^{a,\alpha^*_a}_{ij}$$
12:     Using $(\sqrt{\alpha^*_a}\tilde{x}_i)_{i\in 1,N}$ contruct matrix $(B_{ij})_{i\in 1:N, j\in 1:\max_i k^*_{a,i}}$, where
                $$B^a_{ij} = \text{ j-th nearest neighbour of } \sqrt{\alpha^*_a}\tilde{x}_i$$
        and matrix $\left(C^a_{ij}\right)_{i\in 1:N, j\in 1:\max_i k^*_{a,i}}$, where
            $$C^a_{ij} = S(y_{B^a_{ij}}, a)$$
13: end for
14: for each $i \in 1 : K$ do
15:     Fit LassoLars model or other regression model to the $(C^a_{j,1:k^*_{a,i}}, S(y_j, a))_{j\in 1:N}$ and
        compute predictions $(\hat{S}(y_j, a))_{j\in 1:N}$ on training inputs $C^a_{j,1:k^*_{a,i}}$
16:     Compute the following
            $$\hat{\epsilon}^i_{\tilde{\mathbf{X}}} = \frac{1}{N}\sum_{i=1}^{N}(\hat{S}(y_i, \hat{a}^*_i) - \hat{S}(y_i, \hat{a}^{**})),$$
        where
            $$\hat{a}^{**} = \underset{a}{\operatorname{argmax}} \sum_{i=1}^{N} \hat{S}(y_i, a),$$
            $$\hat{a^*_i} = \underset{a\in A}{\operatorname{argmax}}\, \hat{S}(y_i, a)$$
17: end for
18: Define
            $$\hat{\epsilon}_{\tilde{\mathbf{X}}} = \underset{i}{\operatorname{mean}}\, \hat{\epsilon}^i_{\tilde{\mathbf{X}}}$$
19: return $\hat{\epsilon}_{\tilde{\mathbf{X}}}$
```

The algorithm works as follows. First of all the assignment of four parameters is required: an initial value k_{max} of neighbors, the number K of train/validation splits, the number k_{bayes} of nearest neighbors to consider during the n_{bayes} steps in a Bayesian search.

In Step 1 we split randomly K times the input data set into train/validation subsets. Step 2 states that each $a \in A$ is processed separately. In Step 3 a Bayesian search for metric begins. For candidate vector α the input data set is transformed to $\{\sqrt{\alpha}\tilde{x}_i\}_{1\in 1:N}$ as in Step 4. Each previously created train/validation split $F_i \sqcup G_i$ is processesd as in Steps 5-9. Firstly, k_{bayes} nearest neighbors for all observations from G_i are determined among the training part F_i, and the matrix A is created as in Step 6. In Step 7 the mean validation absolute error of the j-nearest neighbors regressor for each $j \leq k_{bayes}$ is calculated. The minimal average validation error across all train/validation partitions is returned in Step 8. After n_{bayes} steps of **Bayesian search** the optimal weights vector α_a^* is returned as in Step 10.

In Step 11 the optimal number of neighbors $k_{a,i}^*$ are computed for each train/validation split. This completes the search for a weighted Euclidean metric. The second stage begins.

In Step 12 matrix $\left(C_{ij}^a\right)_{i\in 1:N, j\in 1:\max_i k_{a,i}^*}$ is contructed, where

$$C_{ij}^a = S(y_{B_{ij}^a}, a)$$

and B_{ij}^a is the j-th nearest neighbour of $\sqrt{\alpha_a^*}\tilde{x}_i$. In Steps 14-17 each of the K train/validation splits is processed separately. Consider the i-th split and define the data set

$$C_{a.i} = \Big((C_{nm}^a)_{m\in 1:k_{a,i}^*}, S(y_n, a) \Big)_{n\in 1:N}.$$

In Step 15 the LassoLars model is fitted on $C_{a,i}$, where $(C_{nm}^a)_{m\in 1:k_{a,i}^*}$ is a feature vector and the corresponding response is $S(y_n, a)$. The prediction on $C_{a,i}$ is returned. The justification of this step is that the LassoLars model automatically assigns weights to neighbours through L^1 regularization, thus selecting the 'important' ones. Then in Step 16 an estimate of VoI is calculated based on the prediction from Step 15. The final estimate is an average of estimates given for each train/validation split as in Steps 18, 19.

Finally, we highlight a connection of our **Algorithm** 3 with Strong and Oakley's **Algorithm** 2, in the case a GP model is chosen as a smoothing technique. The trained GP model for a new observation $\tilde{x}^*$ returns the following prediction

$$y^* = (K(\tilde{x}^*, \tilde{x}_i))_{i\in 1:N} \, (K(\tilde{x}_i, \tilde{x}_j))^{-1}_{i\in 1:N, j\in 1:N} (S(y_i, a))_{i\in 1:N}, \tag{14}$$

where $K(\cdot,\cdot)$ is a kernel function. The prediction (14) is a weighted average of the input data set $(S(y_i, a))_{i\in 1:N}$ with a specific choice of the weights. This is equivalent to the weighted nearest neighbors algorithm. Thus, **Algorithm** 2 and **Algorithm** 3 differs only by the weights selection.

5.3. **Local learning approach.** The idea presented in **Algorithm** 3 is based on equation (8) which is a local approximation of the inner expectation. In section 5.2, we have proposed using the nearest neighbors algorithm for that purpose but we can be more general and use a local modeling approach to make our estimate more robust. By local modeling we mean using models that fits to the data locally. A well-known representative example is the *locally estimated scatterplot smoothing model* (LOESS) [8]. The obvious gain from using local modeling is in predictive power as local models are a more flexible solution than the ordinary nearest neighbors approaches, becasue they can incorporate not trivial local patterns in data. On the ther hand, LOESS model is computationally expensive, becasue it performs fitting in the neighborhood. A good alternative is to partition data into blocks and build a model for each block separately. Such an approach is expected to be much faster computationally and also very flexible if partitioning is done automatically and is optimized to the data. A good example of such algorithm is the multivariate adaptive regression splines model (MARS) [15]. The idea presented here is similar to the one presented in **Algorithm** 2 with the difference that instead of a GAM or a GP model we fit a MARS or a LOESS model.

6. Numerical Experiments

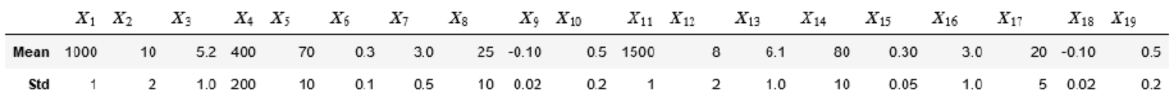
In this section, we compare the performance of the algorithms in a series of numerical experiments. In Section 6.1, we present results for the first case study in [37]. In Section 6.2, we present results for the second case study in [37]

6.1. **Case Study 1.[37].**

As a first case study, we rely on Case Study 1 of Strong's and Oaekly's [37]. The model represents a hypothetical decision tree with correlated inputs. The decision-maker is selecting between two alternatives whose utility $S(y, a_i)$ $(i = 1, 2)$ depends on 12 uncertain inputs as follows:

$$\begin{aligned} S(y,a_1)|\tilde{\mathbf{X}}=\tilde{\mathbf{x}}) &= \lambda(x_5x_6x_7+x_8x_9x_{10})-(x_1+x_2x_3x_4) \\ S(y,a_2)|\tilde{\mathbf{X}}=\tilde{\mathbf{x}}) &= \lambda(x_{14}x_{15}x_{16}+x_{17}x_{18}x_{19})-(x_{11}+x_{12}x_{13}x_4), \end{aligned}.$$

as in [37], in The inputs are assigned a joint normal distribution, with X_5, X_6, X_{14} and X_{16} pairwise correlated with a correlation coefficient equal to 0.6. Table 1 reports the means and standard deviations of the inputs. We also assign $\lambda = 10000$ as in [37]. We report a selection of several experiments developed to test the performance of the algorithms presented in the previous sections. We perform experiments at increasing sample sizes and



	X_1	X_2	X_3	X_4	X_5	X_6	X_7	X_8	X_9	X_{10}	X_{11}	X_{12}	X_{13}	X_{14}	X_{15}	X_{16}	X_{17}	X_{18}	X_{19}
Mean	1000	10	5.2	400	70	0.3	3.0	25	-0.10	0.5	1500	8	6.1	80	0.30	3.0	20	-0.10	0.5
Std	1	2	1.0	200	10	0.1	0.5	10	0.02	0.2	1	2	1.0	10	0.05	1.0	5	0.02	0.2

Table 1: Summary of means and standard deviations for case study 1

to obtain an uncertainty quantification of the estimates we perform 300 replicates sampling independent data sets from the input distributions and correspondingly re-running the model. The uncertainty in the estimates is then represented as a boxplot. Figures 1,2, 3. display the obtained results. Figure 1 present results for individual features, while Figures 2, 3 for features group. Above each box in parentheses is the mean and standard deviation of the estimates. Additionally, the mean is indicated by a solid line on each box. The graphs in each figure show results for sample sizes starting at $M = 500$ (first graph) to $M = 50000$ respectively (the sample size is displayed at the top of each graph).

Table 2 shows the average execution times (over 300 replicates) in seconds of the alternative algorithms (listed in the first column), for alternative sample sizes (second column) and for alternative group size (from one feature, column 3, to eight features, column 7).

Algorithm	Sample size	One feature	Two features	Four features	Five features	Eight features
Algorithm 3 (LassoLars weights)	500	0.35	3	3	4	7
	1000	0.4	4	4	6	8
	10000	10	14	10	12	20
	25000	30	16	15	15	24
	50000	35	20	27	27	30
Algorithm 3 (No LassoLars weights)	500	0.3	3	3	4	7
	1000	0.35	3	3	5	8
	10000	5	8	8	10	18
	25000	13	9	10	12	20
	50000	20	10	17	18	21
Algorithm 2 (LOESS)	500	0.05	0.05	0.15	Too large dimensionlity	Too large dimensionlity
	1000	0.06	0.1	0.5		

	10000	1	3	30	for	for
	25000	13	50	600	the R	the R
	50000	33	145	>600	implementation	implementation
Algorithm 2 (MARS)	500	<1	<1	1	<1	<1
	1000	<1	<1	1	<1	<1
	10000	<1	< 1	1	<1	<1
	25000	<1	<1	1	1	1
	50000	<1	<1	1	1	3
Algorithm 2 (GAM)	500	<1	<1	<1	2	<1
	1000	<1	<1	<1	3	<1
	10000	<1	<1	<1	<1	1
	25000	<1	<1	1	1	3
	50000	<1	1	3	3	6
Algorithm 2 (GP)	500	2	2	2	2	3
	1000	4	4	4	4	4
	10000	50	60	70	92	192
	25000	>500	>500	>500	>500	>500
	50000	>500	>500	>500	>500	>500

Table 2. Approximate running times in seconds of various algorithms for VoI estimation depending on the number of input variables.

The second row reports the performance of the LarroLars algorithm with weight selection. The values show that the algorithm takes about 0.35 seconds to estimate information value at a sample size $N = 500$, while it takes 35 seconds at $N = 50000$. For groups of larger size, we have a systematic increase with the sample size, but the increase with respect to the cardinality of the group is not systematic. For instance, at $N = 50000$, we register the lowest estimation time for groups of 2 features and the time for the 8-feature group is lower than the time for the 1-feature group (we discuss this aspect further). The third row reports the performance of the same algorithm without automatic weight selection.

The algorithm is systematically faster that the version with weight selection. Also for this algorithm we register a systematic increase with the sample size, but a non-systematic increase with the group size, with the group of two features being faster than the group with 1 feature. The fourth row reports results for the times of Algorithm 2 with LOESS as a metamodel. We see that the algorithm is substantially slower that the LassoLars algorithm and inapplicable for groups of four features for $N \geq 25000$ and not applicable for groups of 5 ot 8 features. Conversely, Algorithm 2 with MARS (row 5) or GAM (row 6) proves to be computationally fast, with running times smaller than 1 second in the majority of cases. Finally, row 7 shows that the GP algorithm fails to yield estimates within reasonable time at $N = 25000$ and $N = 50000$. This result is in line with previous literature findings [19].

Regarding the non-systematic increase with respect to the group size, we belive this is is mainly due to the fact that for higher dimensional feature groups it is optimal to consider a smaller number of neighbors than in low-dimensional cases. This is directly related to the curse of dimensionality and it is easily seen in the following example. Consider a single feature $\mathbf{X}_1$ and any point x_0^1 from the input data set. If we plot a sphere on the basis of $\mathbf{X}_1$ and a fixed radius centered at x_0^1, then there will be k_0 neighbors within its boundary (let's assume that $k_0 > 0$). On the other hand, if we want to include a larger number of features and plot a sphere with the same center and such that it includes the same number of neighbors k_0, its radius will have to increase with the increase of the number of features under consideration, which icrease a bias as in (13). Thus, for a fixed number of observations in the input set, as the number of considered features increases, due to the bias-variance tradeoff it is optimal to use a smaller number of neighbors.

Notice that the GP and LOESS algorithms are dropped in calculations for sample sizes equal to $N = 25000$ and $N = 50000$ because their execution time is larger than 10 minutes. Additionally, we use the R implementation of the LOESS algorithm, which allows us to perform calculations for groups of variables with a maximum size of 4. Overall, Algorithm 2 (GAM) and Algorithm 2 (MARS) turn out to be timewise inefficient, for sample sizes greater than $N = 10000$, while the remaining algorithms are fast and of practical applicability also at large sample sizes. (The LassoLars weights Algorithm 5 increases the computation time, but to such an extent that it is still possible to apply the algorithm to data sets with a large number of observations.) This aspect is important because we are guaranteed an asymptotic consistency and thus all these algorithms allow analysts to exploit from the largest possible sample size.

In Figure 1, the mean point estimates are stable at about 610 with all algorithms at all sample sizes. The horizontal axis reports the 95% confidence intervals for the estimates of X_6. One notes that the width of the confidence intervals shrinks as the sample size increases. At $N = 500$ for **Algorithm** 3 (with LassoLars weights) we register $CI_{0.95} = [490, 720]$, which corresponds to a $[-18\%, +20\%]$ around the mean, while at $N = 50000$ we register $CI_{0.95} = [594, 626]$, which corresponds to a variation of about 2% around the mean.

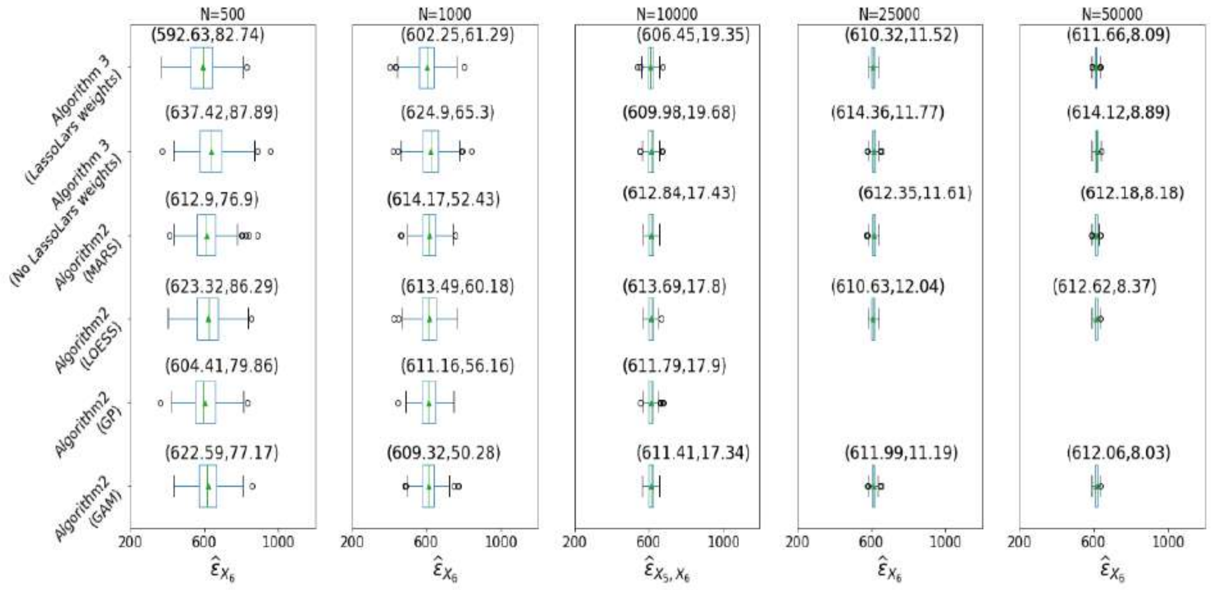


FIGURE 1. Result of VoI estimation for X_6 by different algorithms

We then consider the estimation of the joint information value of input groups. Figures 2 and 3 display results for the estimation of a 4-input group, X_5, X_6, X_{14}, X_{15} and a 5-input group $X_5, X_6, X_{14}, X_{15}, X_{19}$. Both figures show that the average point estimates of the tested algorithms differ for almost all sample sizes. Moreover, all algorithms except **Algorithm** 3(No LassoLars weights) produce stable estimates starting from a certain sample size. For example, **Algorithm** 2(MARS) produces stable estimates starting from a sample size $N > 1000$ for both Figures 2 and 3. The point estimate produces a value of

$\widehat{\epsilon}_{X_5,X_6,X_{14},X_{15}} = 850$ **Algorithm** 3 with LassoLars weights, is stable starting with a sample size greater than 10000 in Figure 2 (mean estimate around $\widehat{\epsilon}_{X_5,X_6,X_{14},X_{15}} = 845$) and than 25000 in Figure 3 (mean estimate around $\widehat{\epsilon}_{X_5,X_6,X_{14},X_{15},X_{19}} = 848$).

We can then establish a confidence interval by considering the area between the 25th and 75th percentiles. We say that two algorithms give consistent results if their confidence intervals have a common part. For both Figures 2 and 3 and samples larger than 500 we observe that the **Algorithm** 3 without LassoLars weights gives estimates not consistent with other approaches. The situation changes significantly when Lassolars weights are used. We observe then the consistency of the results with all tested versions of **Algorithm** 2. What is more, this consistency turns out to be even greater in comparison with the consistency among different versions of **Algorithm** 2. This means that **Algorithm** 3 in both cases gives results that are a good compromise between estimates derived from different versions of **Algorithm** 2.

6.2. **Case Study 2 of Strong and Oakley 2014 [37].** The case study contains a three-state state Markov Model and concerns the selection between two alternatives, a_1 and a_2, whose utilities depend on a set of 31 inputs as follows:

$$S(y,a_1)|\mathbf{X} = \lambda(\sum_{i=1}^{20} S_1^T M_1^i U_1 + X_8X_9X_{10}) - (X_1 + X_2X_3X_4)$$

$$S(y,a_2)|\mathbf{X} = \lambda(\sum_{i=1}^{20} S_2^T M_2^i U_2 + X_{17}X_{18}X_{19}) - (X_{11} + X_{12}X_{13}X_4).$$

Regarding the input distributions, we have $S_1 := (X_5, 1, 0)^T$,$S_2 := (X_{14}, 1 - X_{14}, 0)^T$, $U_1 := (X_6, 0, 0)^T$,$U_2 := (X_{15}, 0, 0)^T$ and

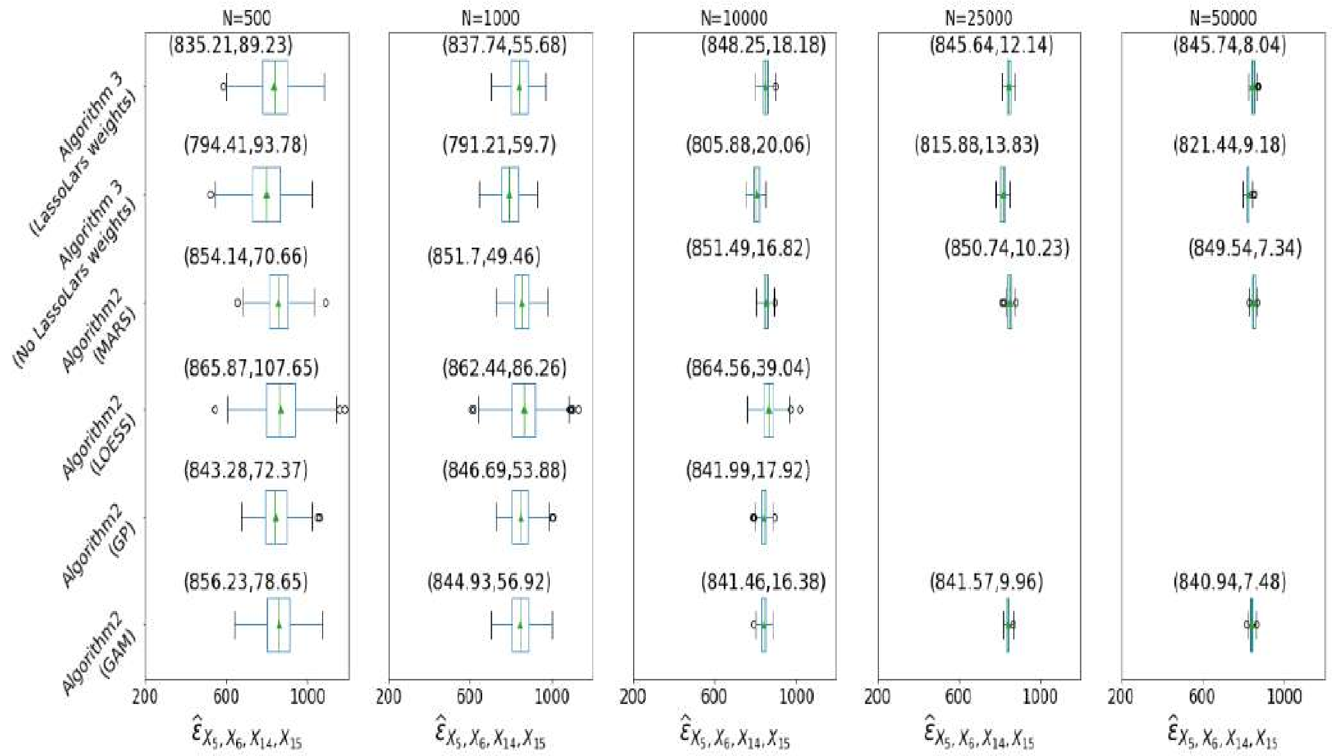


FIGURE 2. Result of VoI estimation for the input group $(X_5, X_6, X_{14}, X_{15})$ by different algorithms

$$M_1 := \begin{pmatrix} X_{20} & X_{21} & X_{22} \\ X_{23} & X_{24} & X_{25} \\ 0 & 0 & 1 \end{pmatrix} \qquad \mathrm{M}_2 := \begin{pmatrix} X_{26} & X_{27} & X_{28} \\ X_{29} & X_{30} & X_{31} \\ 0 & 0 & 1 \end{pmatrix},$$

[37] assign the following distributions to the uncertain inputs:

$$X_{20}, X_{21}, X_{22} \sim \text{Dirichlet}(70, 40, 10),$$

$$X_{23}, X_{24}, X_{25} \sim \text{Dirichlet}(10, 100, 20),$$

$$X_{26}, X_{27}, X_{28} \sim \text{Dirichlet}(70, 40, 10),$$

$$X_{29}, X_{30}, X_{31} \sim \text{Dirichlet}(10, 100, 20),$$

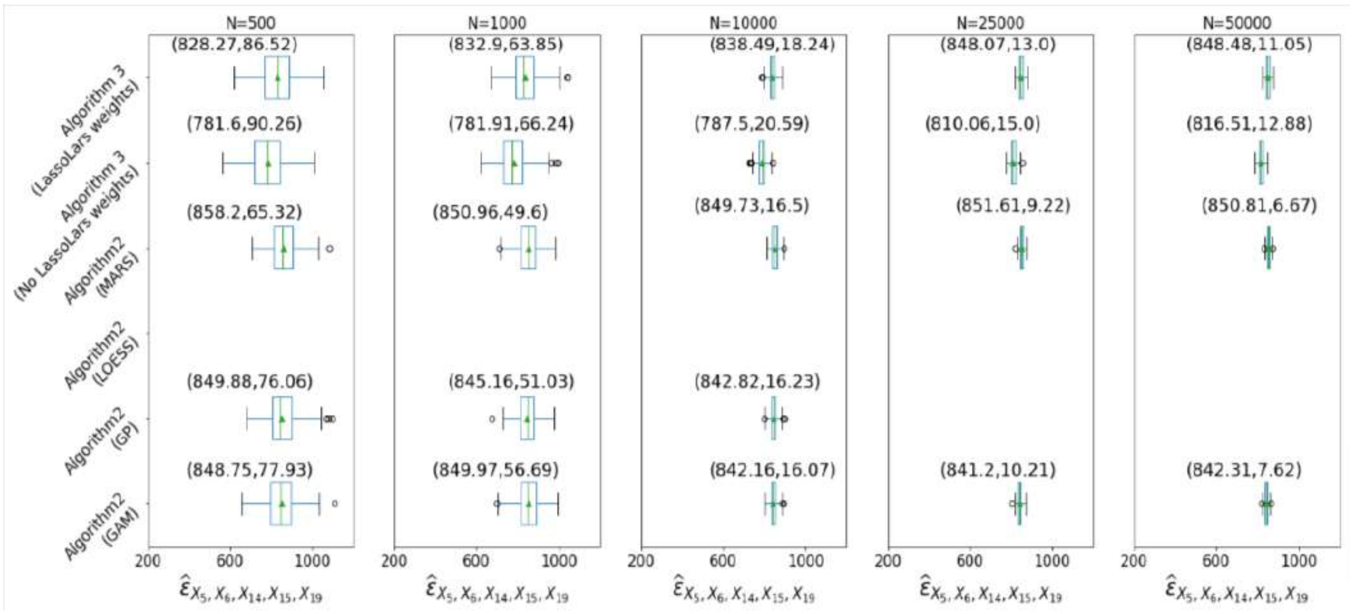


FIGURE 3. Result of VoI estimation for $(X_5, X_6, X_{14}, X_{15}, X_{19})$ by different algorithms

$X_2, X_5, X_8, X_{12}, X_{14}, X_{17}$ are assigned Beta distributions and $X_3, X_4, X_{10}, X_{13}, X_{19} \sim$ Gamma distributions, with means and standard divations reported in Table 1. Inputs X_1 to X_{19} are assumed independent of the others.

We report results for sample sizes equal to 500, 1000, 10000, 25000 and 50000 for different combinations of input features. In Figure 4 we present results for single features and in Figures 5,6,7 for groups of features.

In Figure 4, the mean point estimates are stable at an information value of about $\hat{\epsilon}_{X_6} = 510$ for all algorithms and at all analyzed sample sizes. The horizontal axis reports the 95% confidence intervals for the estimates of X_6. One notes that the width of the confidence intervals shrinks as the sample size increases. At $N = 500$ for **Algorithm** 3 (with LassoLars weights) we register $CI_{0.95} = [488, 536]$, which corresponds to a $\pm 4\%$ deviation around the mean, while at $N = 50000$ we register $CI_{0.95} = [502, 520]$, with a deviation of about $\pm 1.7\%$ around the mean.

We now report results for groups of higher dimensions. Figures 5, 6, and 7 display results for the estimation of a 2-input group, (X_5, X_{14}), a 4-input group $(X_5, X_6, X_{14}, X_{15})$ and a 8-input group $(X_{20}, X_{21}, X_{23}, X_{24}, X_{26}, X_{27}, X_{29}, X_{30})$, respectively.

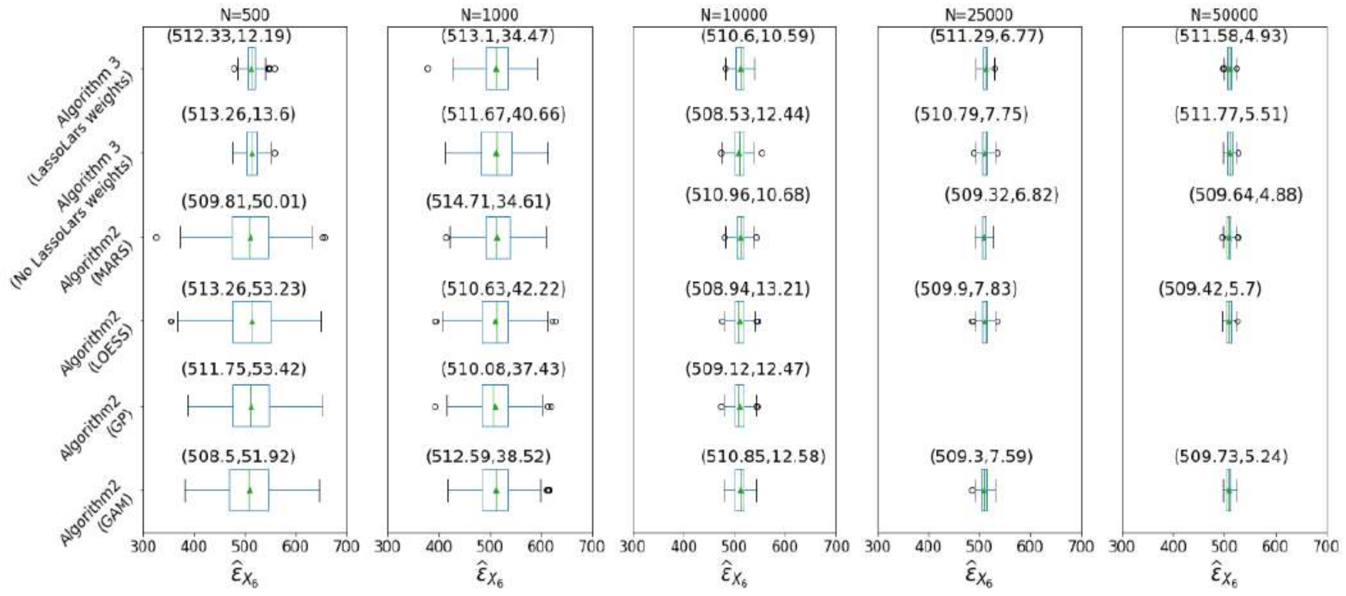


FIGURE 4. Result of VoI estimation for X_6 by different algorithms

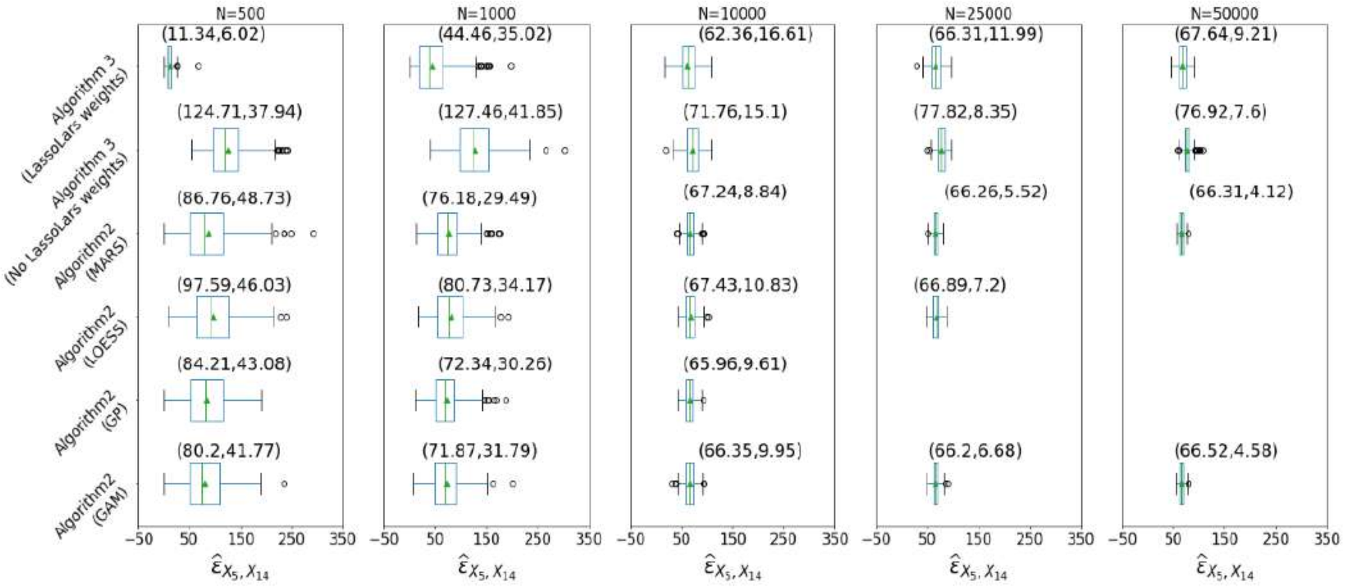


FIGURE 5. Result of VoI estimation for (X_5, X_{14}) by different algorithms

Figure 5 shows that the point estimates $\hat{\epsilon}_{X_5,X_{14}}$ are more sensitive to the sample size than in the one-dimensional case. Specifically, algorithm 1 fails to produce reasonable

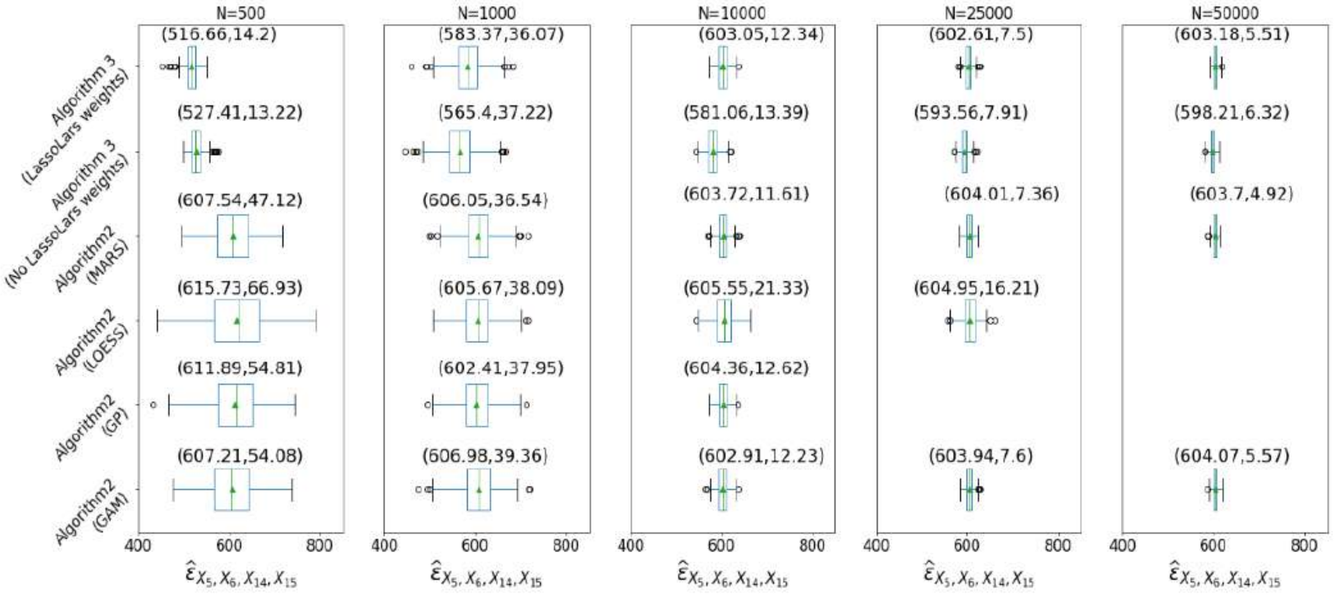


FIGURE 6. Result of VoI estimation for $(X_5, X_6, X_{14}, X_{15})$ by different algorithms

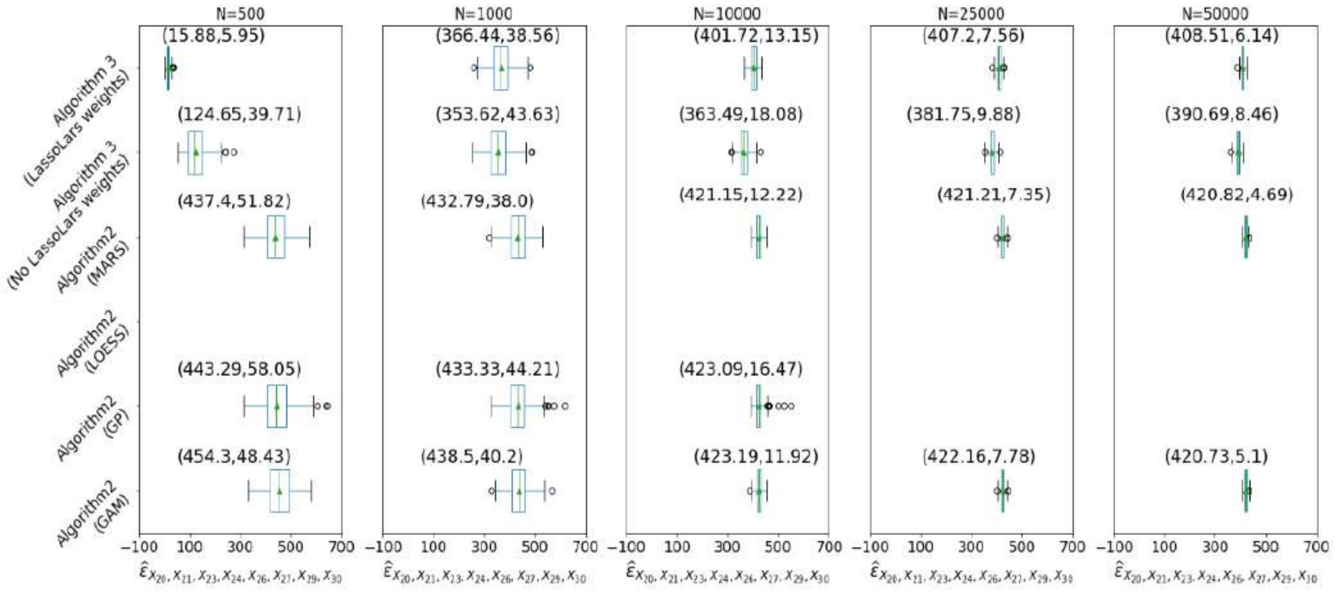


FIGURE 7. Result of VoI estimation for $(X_{20}, X_{21}, X_{23}, X_{24}, X_{26}, X_{27}, X_{29}, X_{30})$ by different algorithms

estimates at $N = 500$, however, it produces accurate estimates for $N \geq 10000$. All three figures show that the average point estimates of the tested algorithms differ for almost all sample sizes. Moreover, all algorithms except **Algorithm** 3 (No LassoLars weights) produce stable estimates starting from a certain sample size. For example, **Algorithm** 2 (MARS) gives results that show stability starting from a sample size greater than 10000 for all three figures. **Algorithm** 3 with LassoLars weights, is stable starting with a sample size greater than 10000 in Figure 6 (the mean point estimates around $\widehat{\epsilon}_{X_5,X_6,X_{14},X_{15}} = 602$) and than 25000 in Figures 5, 7 (the mean point estimates around $\widehat{\epsilon}_{X_5,X_{14}} = 67$; $\widehat{\epsilon}_{X_{20},X_{21},X_{23},X_{24},X_{26},X_{27},X_{29},X_{30}} = 408$ respectively).

Overall, we can draw analogous conclusions with respect to Study Case 1 that highlight the importance of using LassLars weights in **Algorithm** (3).

6.3. **VoI as feature selection tool for decision problems.** Consider now that time or resource constraints allow the decision-maker to collect information on the inputs sequentially and that they do not allow to collect simultaneously information on all features. The problem is then to find the most informative subset of features. This problem (feature selection) has been formulated in alternative ways and is, by nature, a combinatorial problem. In fact, there are $\binom{d}{k}$ ways to select k features out of n. In our case, we express the problem as that of finding the input subset D of minimal cardinality such that $\mathbf{X}_D \subseteq \mathbf{X}$ of minimal cardinality such that

$$\epsilon_{\mathbf{X}_D} \geqslant \alpha\epsilon_{\mathbf{X}},$$

where $0 \leq \alpha \leq 1$. For such a problem direct solution would be to consider all combinations of features and find the optimal subset. But such an approach is very computationally expensive. An approximate algorithmic solution is the step-forward feature selection where at each step a feature that gives the biggest increase in the VoI is selected.

In order to make this approach practical, we need a fast estimation procedure. We performed experiments for such forward-step feature selection for Case Study 1 (Section 6.1) and Case Study 2 (Section 6.2) using **Algorithm** 3 for VoI estimation.

The results are presented in Figures 8 and 9.

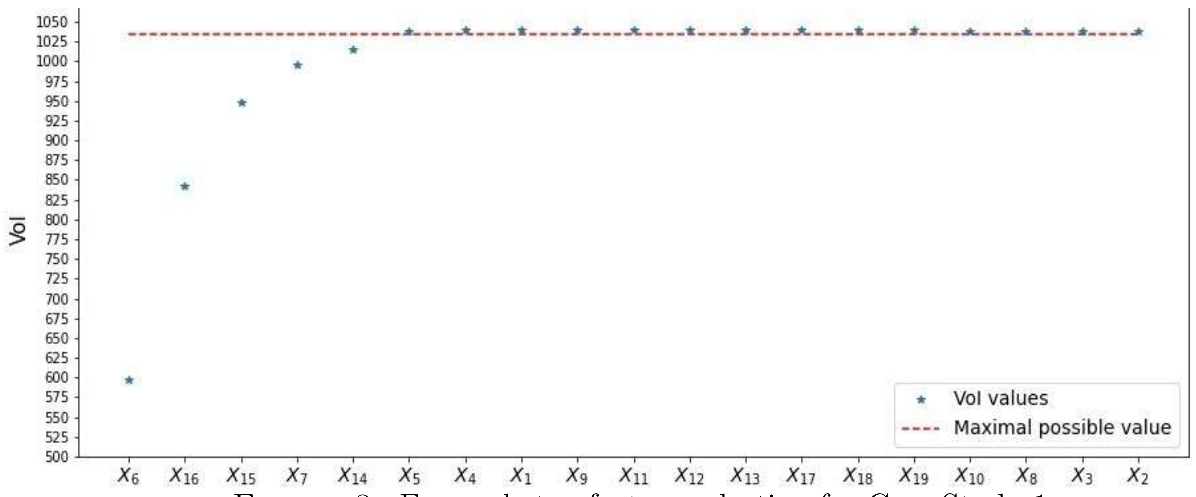


FIGURE 8. Forward-step feature selection for Case Study 1

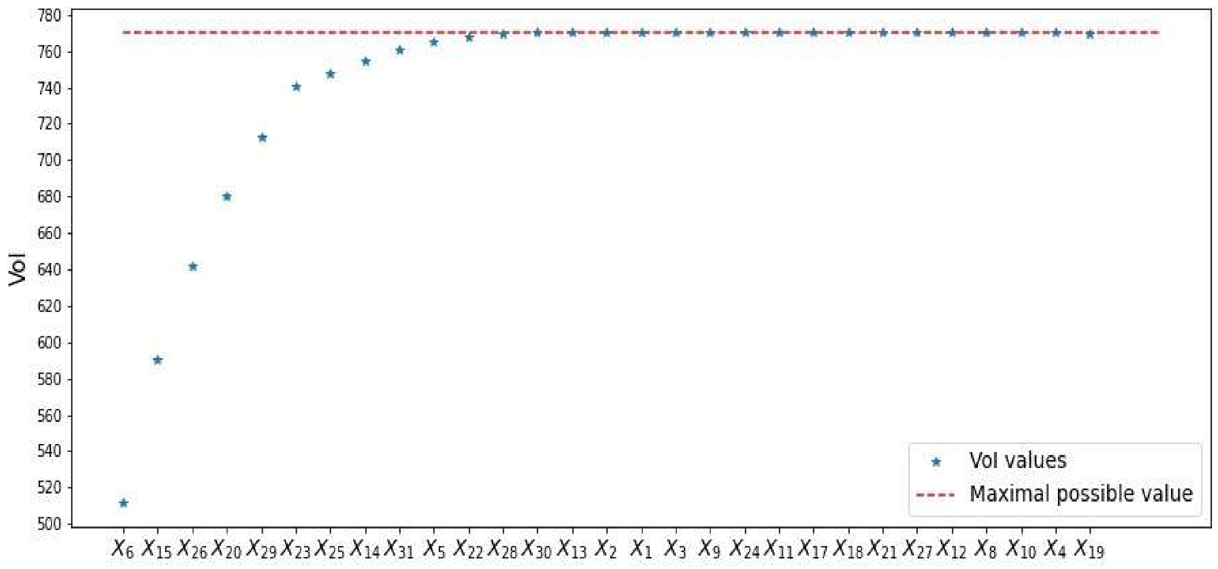


FIGURE 9. Forward-step feature selection for Case Study 2

In Figures, 8 and 9 the horizontal axis presents the selection order given by the feed-forward procedure and y-axis displays the VoI after adding a new feature to the current subset of features. Figure 8 shows that set of features $\{X_6, X_{16}, X_{15}, X_7\}$ carries over 97% of the total VoI of all 19 features. Similailry, Figure 9 shows that set of features $\{X_6, X_{15}, X_{26}, X_{20}, X_{29}, X_{23}\}$ carries over 96% of the total VoI of all 30 features. The such analysis brings a clear picture of the drivers of information value for these case studies.

CHAPTER 2
INFORMATION DENSITY ESTIMATION

7. Introduction

Global sensitivity measures, and therefore also the value of information in its various forms, summarize the contribution of inputs to output variability in a unique number. As such, they do not yield information on the region of the input support which is active in determining the relevance of an input. This directional concern has the merit, instead, of making the analysis of input importance more transparent. Here, we note that this type of sensitivity analysis is called regional sensitivity analysis with the term possibly being used for the first time in [22]. In the context of decision analysis, graphical tools are one-way and two-way sensitivity analysis methods. These methods, while providing regional information, are not engineered to take uncertainty into account. As underlined, for instance, in [3], their deterministic nature does not make them compatible with a decision problem under input uncertainty. The earlier analysis of [13] also reveals that one may obtain false impressions of sensitivity by relying on deterministic methods in the presence of uncertainty.

To overcome this limitation, Hazen (2014)[18] introduces the notion of information density, as a regional sensitivity method within VoI analysis. The intuition is to add a tool that can provide clear information about the critical directions of sensitivity - the same information that is captured in a graphical sensitivity diagram. A few years after the proposal, Hazen et al (2022) [1] give formal conditions under which information density exists and is well-posed. They perform some preliminary numerical experiments for computing information density in a black box context. Their approach, however, is based on a double-loop design which is computationally expensive. Indeed, Hazen et al (2022) point out the need for a fast and accurate way to estimate the information density.

In the remaining chapters, we present novel algorithms for the estimation of information density. Our intuition is to adapt the previously discussed algorithms for VoI estimation so as to extract simultaneously the regional information associated with information density. We then adapt the Strong and Oakley's [38] VoI estimation algorithm and the nearest neighbor approach. We conduct a series of experiments with a case study with 2 scenarios: a multilinear model in which inputs are correlated, but with known analytic solutions for all conditional distributions, and the same model in which inputs are correlated but where sampling from the conditional distributions requires MCMC. The testing is done for different sample sizes and we address both the one and the two-dimensional information density.

In Section 8, we present the formal definition of information density. In Section 8, we introduce the estimation methods. In Section 10, we present results of numerical experiments for two case studies.

8. Information Density

The work of Hazen (2014)[18] introduces the concept of information density. A more thorough discussion of the definition can be found in the work of Hazen, Borgonovo, and Lu (2022) [1]. These two works provide the theoretical foundations of information density. For instance, they show that information density remains meaningful when information value is defined as an expected utility increase, while information density becomes difficult to interpret if information value is expressed as a certainty equivalent increase or as a buying (or selling) price of information. Some numerical experiments and an application of information density are discussed in Hazen, Borgonovo, and Lu (2022) [1]. However, the authors adopt a double-loop approach of Monte Carlo simulation to provide an initial illustration and do not discuss numerical considerations for estimating information density. In this paper, we address this problem more extensively. Our intuition is to start with the designs for estimating information value discussed in the previous section and to adapt them for estimating information density.

Following Hazen et al (2022) [1], we present the definition of information density and discuss it intuitively by means of an example. The setup is the typical information value one. We consider a decision maker who is selecting an action from a set of possible actions A. The uncertain consequences are associated with random payoff V. The decision-maker possesses a utility function U that maps each possible value of V (and thus each possible consequence) to a real number. The problem faced by the decision-maker is then to find

$$\underset{a\in A}{\operatorname{argmax}}\, \mathbb{E}(U(V^a)). \tag{15}$$

We then consider that the decision-maker has the possibility of collecting information on random vector $\mathbf{X}$ (defined on the same underlying probability space — See Section 4). Then, the decision problem becomes to find

$$\underset{a\in A}{\operatorname{argmax}}\, \mathbb{E}(U(V^a)|\mathbf{X})). \tag{16}$$

Problem (16) is called *prior expected value of action posterior to perfect information* ([28], p.252). Comparing (16) with (15) gives the expected increase in utility registered when uncertainty in X is removed

$$\epsilon_{\mathbf{X}}^{U} = \mathbb{E}(\max_{a\in A} \mathbb{E}(U(V^a)|\mathbf{X})) - \underset{a\in A}{\operatorname{argmax}}\, \mathbb{E}(U(V^a)). \tag{17}$$

The (17) is known as the value of the information (VoI) of $\mathbf{X}$. Note that VoI is always greater or equal to zero and is null if and only if $\mathbf{X}$ do not change the preferred alternative.

Equation (17) can be rewritten in the following form

$$\epsilon_{\mathbf{X}}^{U} = \mathbb{E}(\max_{a\in A} \mathbb{E}(\Delta U^a|\mathbf{X})), \tag{18}$$

where

$$\Delta U^a = U(V^a) - U(V^{a^*})$$

is the gain for switching to alternative a. Note that this quantity is negative for all $a \in \mathcal{A}$ if no new information is received, because a^* is the preferred alternative in that case. In the case, with new information, a^* becomes sub-optimal, then this quantity can become positive for some a.

Equation (18) shows that $\epsilon_{\mathbf{X}}^{U}$ is the expected value of the random variable $\max_{a\in A} \mathbb{E}(\Delta U^a|\mathbf{X})$. Consider the case when $\mathbf{X}$ is absolutely continuous with density $f_{\mathbf{X}}(x)$ over some region $\Omega_{\mathbf{X}} \in \mathbb{R}^d$. Hazen et al (2022) prove that

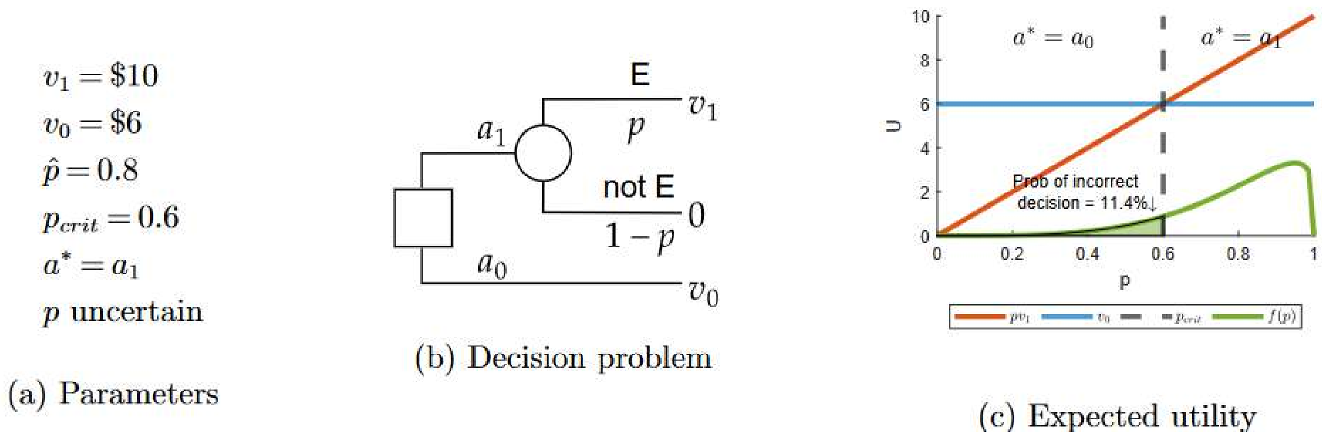
$$\iota_{\mathbf{X}}(x) = f_{\mathbf{X}}(x) \max_{a\in A} \mathbb{E}(\Delta U^a|\mathbf{X} = x) \tag{19}$$

is indeed a density of $\max_{a \in A} \mathbb{E}(\Delta U^a|\mathbf{X})$, that is, they prove that

$$\epsilon^U_{\mathbf{X}} = \int_{\mathcal{X}} \iota_{\mathbf{X}}(x)dx = \int_{\mathcal{X}} f_{\mathbf{X}}(x) \max_{a \in A} \mathbb{E}(\Delta U^a|\mathbf{X} = x), \tag{20}$$

where $\mathcal{X}$ is the support of $\mathbf{X}$. Equation (19) is called *information density* of $\mathbf{X}$ since it is non-negative and its integral is the overall information value. Hazen at al (2022) show that in the case a^* is not unique (it is a set), then also information density is not unique, and we have one information density in correspondence of each alternative in the set.

Example 8.1. *We illustrate the notion of information density with a toy decision problem taken from Hazen et al 2022 [1]. Consider a decision maker selecting among two alternatives, a_0 and a_1. The first alternative is associated with a sure payoff equal of 6\$, the second with a random payoff \$10 if event E occurs. The problem is visualized in graphs (a) and (b) of Figure 10. Let the probability of E occurring be denoted by p. Suppose that p is assigned the base-case value of $\hat{p} = 0.8$. Then, alternative a_1 is prefered, with an expected payoff equal to 8. The value of p at which we change from alternative a_1 to alternative*



(a) Parameters

(b) Decision problem

(c) Expected utility

FIGURE 10. Simple toy decision problem Hazen et al (2022) [1]

a_0 (critical value) is $p_{crit} = 0.6$. If p assumes values below this threshold, the preferred alternative changes (Figure 10 (c)) and the decision-maker selects a_0. However, p is uncertain and the analyst assigns a beta(a=4.8, b=1.2) distribution with mean $\hat{p}$ (Figure 10 (c)). If we perform an uncertainty analysis sampling values of p from this distribution, the

probabilistic sensitivity analysis would reveal an 11.4% *chance of switching from alternative* a_1 *to alternative* a_0*. We can also compute the VoI of* p *(either from the sample or analytically), which results equal to* 0.12\$*. This value is relatively small if compared to the optimal expected payoff of alternative* a_1 *(which, we recall, equals* 8\$*). This result might be interpreted as indicating that uncertainty in* p *might not be an issue despite the* 11.4% *chance of change. However, if we rely solely on the magnitude of VoI we do not obtain any insight into the fact that* p *needs to decrease below the threshold* $p_{crit} = 0.6$*, in order for the preferred alternative to change. This directional information is, instead, delivered by the one-way sensitivity plot. Thus, VoI itself does not allow us to determine the directions of concern in a sensitivity analysis. However, for this test case, it is possible to derive the information density of* p *analytically. The resulting graph is reported in Figure 11 taken from Hazen et al (2022).*

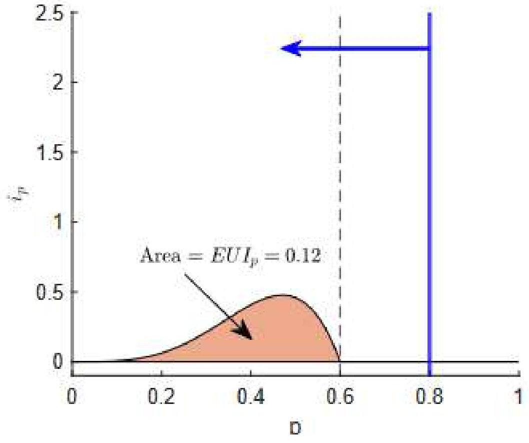


FIGURE 11. Information density for p and direction of concern, from Hazen et al (2022) [1]

The information density in Figure 11 evidences that information about p has its highest value when p is comprised between 0.3 and 0.6. Also it clearly indicates that the direction of concern for p is at the left of its base-case value. In particular, it is the part of the support of p below the threshold that makes p informative. Conversely, If p falls at the right of the threshold, the preferred alternative is not going to change.

The region of maximal information density as well as the direction of concern can also be sought in a two-dimensional setting. Consider that not only p but also v_1 *become uncertain, with base-case values* $(\hat{p}, \hat{v}_1) = (0.80, 80\$)$*, and assign* $v_0 = 65\$$ *with certainty. Suppose p and* v_1 *are regarded as independent and assigned the following distributions: p is beta with mean 0.80 and standard deviation 0.0026 (beta*$(48, 12)$*), and* v_1 *is normal with mean* $80\$$ *and standard deviation* $40\$$*. See the left panel of Figure 12, taken from Hazen et al (2022).*

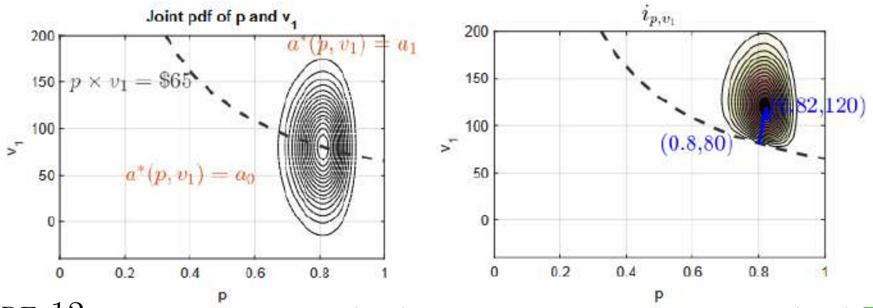


FIGURE 12. Information density for (p, v_1) and direction of concern, Hazen et al (2022) [1]

The dashed line in the graphs of Figure 12 marks the border between the regions where $a^* = a_1$ (above the line in this case) and $a^* = a_0$ (below the line in this case) are, respectively, optimal.

The left and right graphs in Figure 12 also present contour plots of the joint probability density $f_{p.v_1}(p, v_1)$ and the joint information density $\iota_{p.v_1}$, respectively. We recall that the joint value of information of (p, v_1) equals 12.24\$*, which is of the same order of the problem payoffs. Therefore, joint information about (p, v_1) is significant for the decision maker. Let us compare the contour plots in the left and right graphs in Figure 12. The region in which the joint density of (p, v_1) is different from zero covers values included in the rectangle* $[0.65, 0.9] \times [0, 150]$*. This region overlaps with both the regions where a_1 or a_0 is preferred. In the right graph, however, we have a null information density only in the region where a_1 is preferred, as a consequence of the fact that the information gain is non-null only in this region. The right graph also shows that the contour lines reveal a mode of the information density at about the point* $(0.82, \$120)$*. This is the region where information about (p, v_1) has its highest value. The arrow from the baseline* $(\hat{p}, \hat{v}_1) = (0.80, \$80)$ *to the mode of the information density evidence the direction of concern. Note that* $(\delta p, \delta v_1) = (0.02, \$40)$*, so that for both p and v_1, the direction of concern involves an increase (right of the current values).*

9. Estimation

In the previous example, information density can be found analytically. However, in order to use information density in practical applications, we need efficient estimation methods. Before this work, in Hazen et al (2022) a double-loop approach is used to calculate information density numerically. Such an approach works as follows. To fix ideas, consider a one-dimensional setting and let X_i be the parameter of interest. In a double-loop design, the first step is to preselect a set of values (called a grid) on which to compute information density. Let N_G be the number of points in the grid and let x_i^k denote a point in the grid ($k = 1, 2, \ldots, G$). Then, once X_j is fixed at any of these values, we re-evaluate the model through Monte Carlo propagation obtaining a sample of N_{Cond} points from which to compute the conditional expected utility of each alternative. From knowledge of the conditional expected utility one computes the information gain at a given point ($\zeta_{X_i}(x_i^k)$).

The computational cost for each input is then equal to $N_G \cdot N_{\text{Cond}}$ model evaluations. In the case the model is characterized by long running times, then this cost becomes prohibitive. Conversely, we have seen that there are computationally convenient algorithms to estimate information value and, in the previous chapter, we have also discussed new alternatives. We argue that these algorithms can be modified to yield information density as well.

As a first method, we propose an adaptation of the VoI estimation algorithm developed by Strong and Oakley [37].

Algorithm 4 Strong and Oakley (2014) [37] adaptation

```
INPUT: Input data set $\{U(v_{i,a}), x_i\}_{i\in\bar{1,N},a\in A}$,where $x_i \in \mathbb{R}^d$, for $d \in \mathbb{N}$
OUTPUT: Information density of $X_i$
1: For each $a \in A$ fit a machine learning regression model $g_a(\tilde{x})$ to the input data set
   $\{U(v_{i,a}), x_i\}$
2: Evaluate the kernel density estimator $\hat{f}_{X_i}(x_i)$ of $f_{X_i}(x_i)$ using the input data set
3: for each $x_i$ in the input data set do
       compute
           $\hat{\iota}_X^U(x_i) = (\max_a g_a(x_i) - g_{\hat{a}^*}(x_i))\hat{f}_{X_i}(x_i),$
       where $\hat{a}^* = \text{argmax}_a \frac{1}{N}\sum_{i=1}^N g(x_i)$
4: end for
5: return $(x_i, \hat{\iota}_X^U(x_i))$ and visualize it
```

The idea is the same as for Algorithm (2) and is to treat the model output as noisy. Step 1 is the same as Step 1 of Algorithm (2). Step 2 foresees rsees the use of a kernel density estimator to determine $\mathbf{X}$. Notice that this step becomes redundant if the probability density function of $\mathbf{X}$ is known. In that case, the value of $f_{X_i}(x_i)$ can be assigned directly in the density estimator.

In Step 3, the results of Steps 1 and 2 are inserted into the information density formula to obtain the final estimate. The algorithm finally returns all pair of points $(x_i, \hat{\iota}_X^U(x_i))$. These can be used to obtain a visual display. Notice that, potentially, it is not needed to calculate the pair $(x_i, \hat{\iota}^U_X(x_i))$ for all points in the dataset, but the analyst can also pre-determine a grid of selected points. The reason for selecting a subset of lower cardinality could be to reduce computational time. (However, in our experiments no issues have emerged in this respect).

A second proposal is the modification of the nearest neighbor approach based on Algorithm (3).

Algorithm 5 Information density estimate for continuous input features $\tilde{\mathbf{X}}$ based on nearest neighbour algorithm

```
INPUT: Input data set $( U(v_{i,a}), x_i)_{i\in 1:\bar{N}, a\in A}$
PARAMETERS:
  • K - number of folds in the validation procedure
  • $k_{\max}$ - maximal number of nearest neighbours to consider
  • $n_{bayes}$ - number of steps for Bayesian search
  • $k_{bayes}$ - maximal number of nearest neighbors to consider during Bayesian search
OUTPUT: information density for $\mathbf{X}$ at $x_i$
13: Steps 1-13 are the same as in Algorithm (3)
14: for each $i \in 1:K$ do
15:     Fit the LassoLars model or other another regression model to the $(C^a_{j,1:k^*_{a,i}}, U(v_{j,a}))_{j\in 1:N}$
        and compute the predictions $(\hat{U}^i(v_{j,a}))_{j\in 1:N}$ on training inputs $C^a_{j,1:k^*_{a,i}}$
16: end for
17: Compute
        $$\hat{U}(v_{j,a}) = \frac{1}{K}\sum_{i=1}^{K} \hat{U}^i(v_{j,a})$$
18: Evaluate the kernel density estimator $\hat{f}_{X_i}(x_i)$ of $f_{X_i}(x_i)$ using the input data set
19: The final estimate $\hat{\iota}^U_X(x)$ at input data point $x_i$ is given by
        $$\hat{\iota}^U_X(x_i) = (\max_a \hat{U}^i(v_{j,a}) - \hat{U}^i(v_{j,a^*}))\hat{f}_{\mathbf{X}}(x_i),$$
    where $\hat{a}^* = \text{argmax}_a \frac{1}{N}\sum_{i=j}^{N} \hat{U}^i(v_{j,a})$
20: return $(x_i, \hat{\iota}^U_X(x_i))$ and visualize it
```

Up to step 13, the algorithm foresees the same steps as of **Algorithm** (3). Then, differently from **Algorithm** (3), the algorithm foresees storing the value of the information gain at each location, multiply it by the input density and return the value of the information density as such location.

10. Numerical Experiments

In this section, we analyse the performance of the algorithms presented in Section 9 through a series of numerical experiments. We rely on the same case studies as in Section 6. The first step is the generation of a sample from the input distributions. This step is followed by an uncertainty propagation, so that to obtain the unconditional distribution of all the alternatives. We generate samples of sizes N=500, 1000, 10000, 25000 and 50000. In Section 10.1 we presents results for the toy model, for which the analytical values of the information gain, information density and VoI are available. In Section 10.2, we present results for the first case study in [37]. In Section 10.3, we present results for the second case study in [37].

We observe that analytical expressions for these two case studies are not available. Then, it becomes of interest to compare the estimates produced by the alternative algorithms.

10.1. **Results for the Analytical Example.** In this section, we report results for the numerical estimation of the information density for the univariate and bivariate cases discussed in Section 8, with reference to the toy model in Hazen at al (2022). Results are summarized in Figure 13.

On the horizontal axis of each graph of each graph in Figure 13, we report the support of the input and on the vertical axis we display the corresponding information density. In each graph, the green curve indicates the analytical information density.

The first row in Figure 13 repors the estimates of the information density of p with **Algorithm** 4 in which MARS is used as an emulator. We note that starting at N=10000 numerical estimates are close to analytical ones. Thus, the MARS estimator shows an asymptotically consistent behavior. The second row in Figure 13 displays results when the GAM is used in **Algorithm** 4. One notes that this algorithm is not as effective as MARS. For instance, at $N = 1000$, $N = 10000$ and $N = 25000$ one would obtain a visual impression

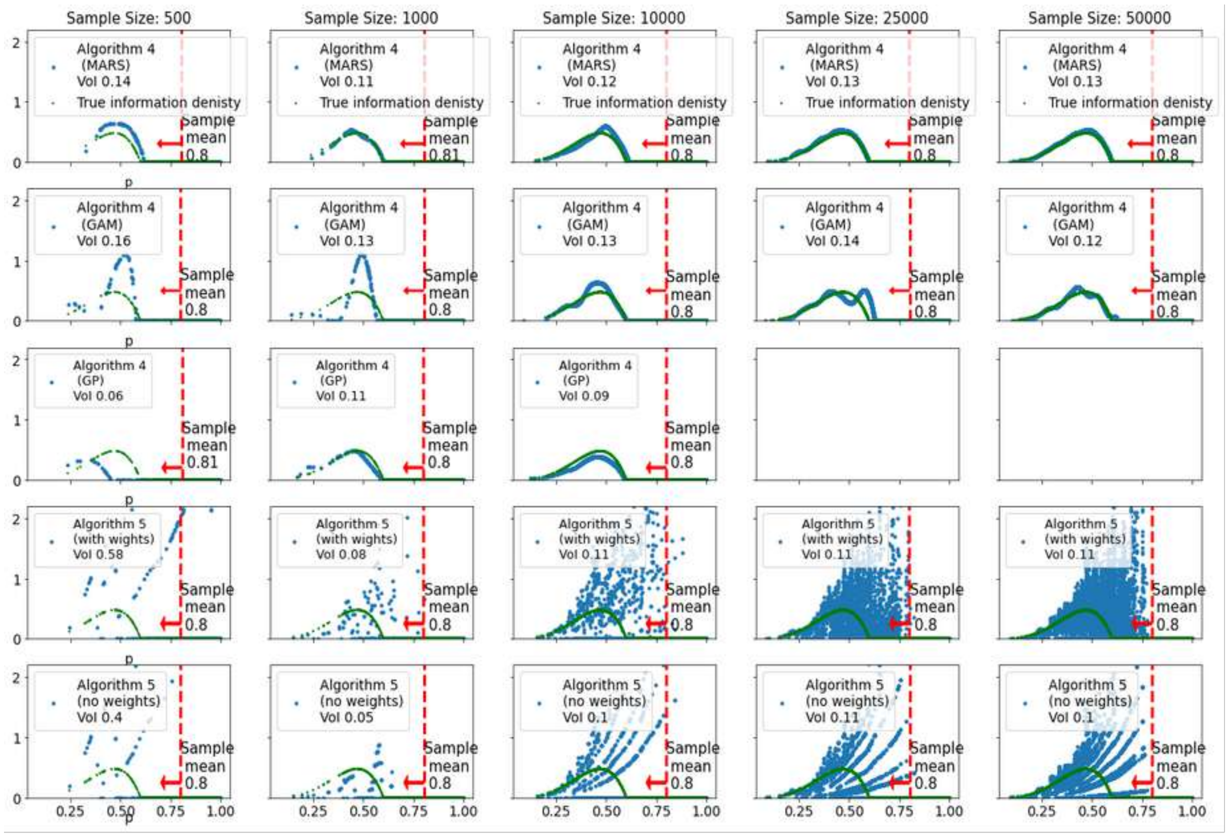


FIGURE 13. Result of information density estimates for p by different algorithms.

of a bimodal information density, which does not coincide with the analytical one. The impression still remains at $N = 50000$ either. The third row in Figure 13 displays results when the GDP is used in **Algorithm** 4. One notes that this choice visually as effective as MARS for small sample sizes. However, at large sample sizes the execution time of the algorithm prevents its utilization. The fourth and the fifth rows in Figure 13 display results for the **Algorithm** 5 with weights and without LassoLars weights, respectively. The graphs show that the estimates are highly volatile and the user will not get accurate information about the information density event at the largest sample sizes.

In this respect, as an additional information on the algorithm performance, it is also useful to take into account also the value of the VoI estimate corresponding to a given information density plot. Figure 13, displays in the legende of each plot the estimated value of ϵ_p.

The estimates of **Algorithm** 5 are less accurate then the estimates of the other algorithms, confirming the volatility impression generated by the plot. For instance at $N = 50000$, the GAM algorithm attains $\epsilon_p = 0.12$, which coincides with the analytical value, while **Algorithm** 5 exhibits $\epsilon_p = 0.10$, still with a 16% error.

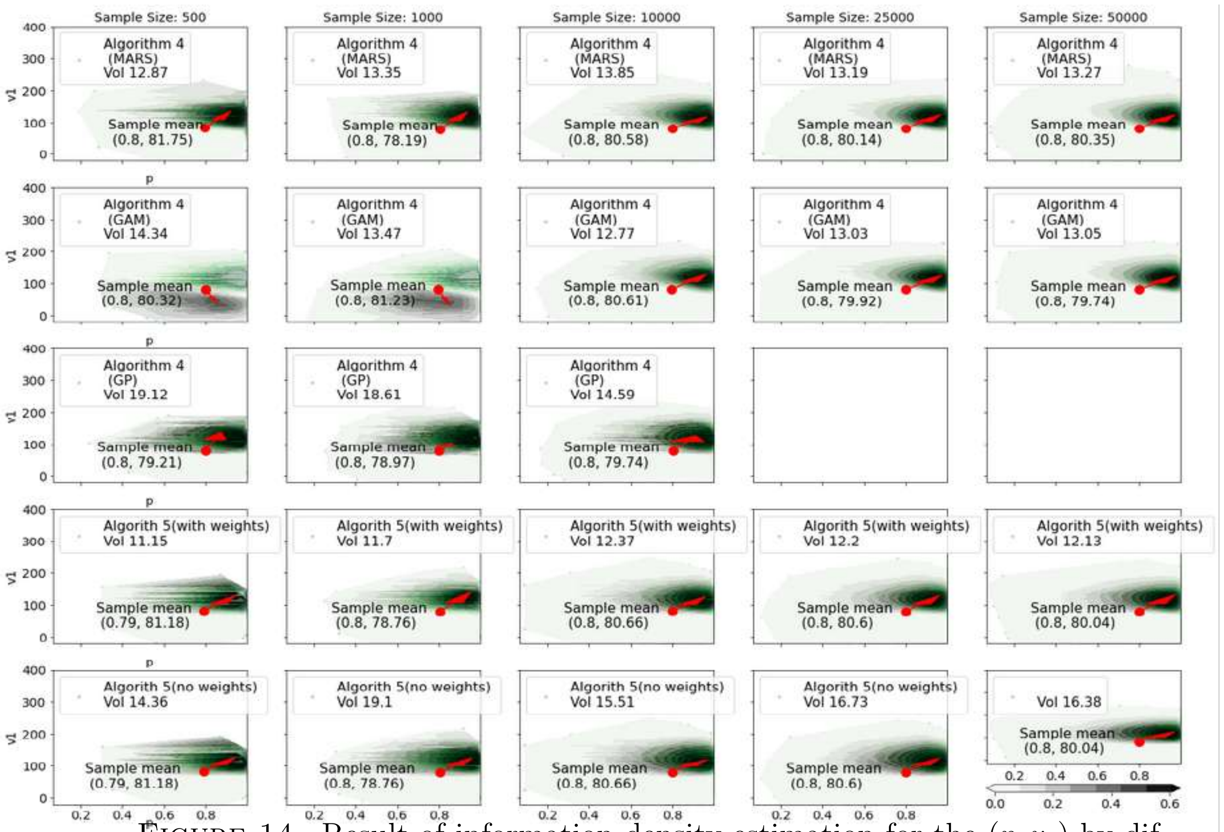


FIGURE 14. Result of information density estimation for the (p, v_1) by different algorithms

Figure 14 displays results for the estimation of the joint information density of (p, v_1). In this respect, the results in Figure 14 are the numerical counterpart of the bivariate calculations performed for the toy model in Example 9.1. The graphs in rows 1-5 of Figure 14 display estimates obtained by the same algorithms as in Figure 13. Also, in each graph, the green shadow indicates the true information density.

We note that in this case the LassoLars algorithm exhibits the best performance: the information value estimates are close to the analytical value of $\epsilon_{p,v_1} = 12.24$ across all samples. **Algorithm** 4 with GAM produces slightly less accurate estimates and the MARS algorithm follows. **Algorithm** 5 without weights produces unreliable information value estimates, while the GP algorithm shows the worst performance in this case, with large errors in the VoI estimates. We then focus on the information density indications produced by the three best performing algorithms. All graphs concur in indicating the active region as comprised within the rectangle $[0.65, 1] \times [100, 200]$, basically at all sample sizes, with the indications becoming more precise as N increases. Also the direction of concern is correctly represented in the majority of the graphs. An exception is registgered by the GAM algorithm at N=500 and N=1000. In this case, the maximal information density is registered in a region that does not coincide with the one obtained analytically. However, if one considers the indications of **Algorithm** 4 with MARS and GAM as well as of **Algorithm** 5 at sample sizes starting at $N = 10000$, one obtains a correct indication about the region in which (p, v_1) are active and about the direction of concern.

In the next two subsections, we examine the performance of the algorithms in estimating information density for the two case studies in [37]. For these case studies analytical results for the information density and VoI are not available and therefore an analyst needs to rely on the comparison of the graphs and estimates produced by the alternative algorithms. We start with Case Study 1 in [37].

10.2. **Case Study 1.[37].** Figure (15) presents results for the information density of $\mathbf{X}_6$, the input associated with the largest individual value of information. Similarly, to Figure 13 the graphs in the first row display estimates produced by **Algorithm** 4 with MARS, with sample sizes from $N = 500$ to $N = 10000$, the second row with **Algorithm** 4 with GAM, and similar.

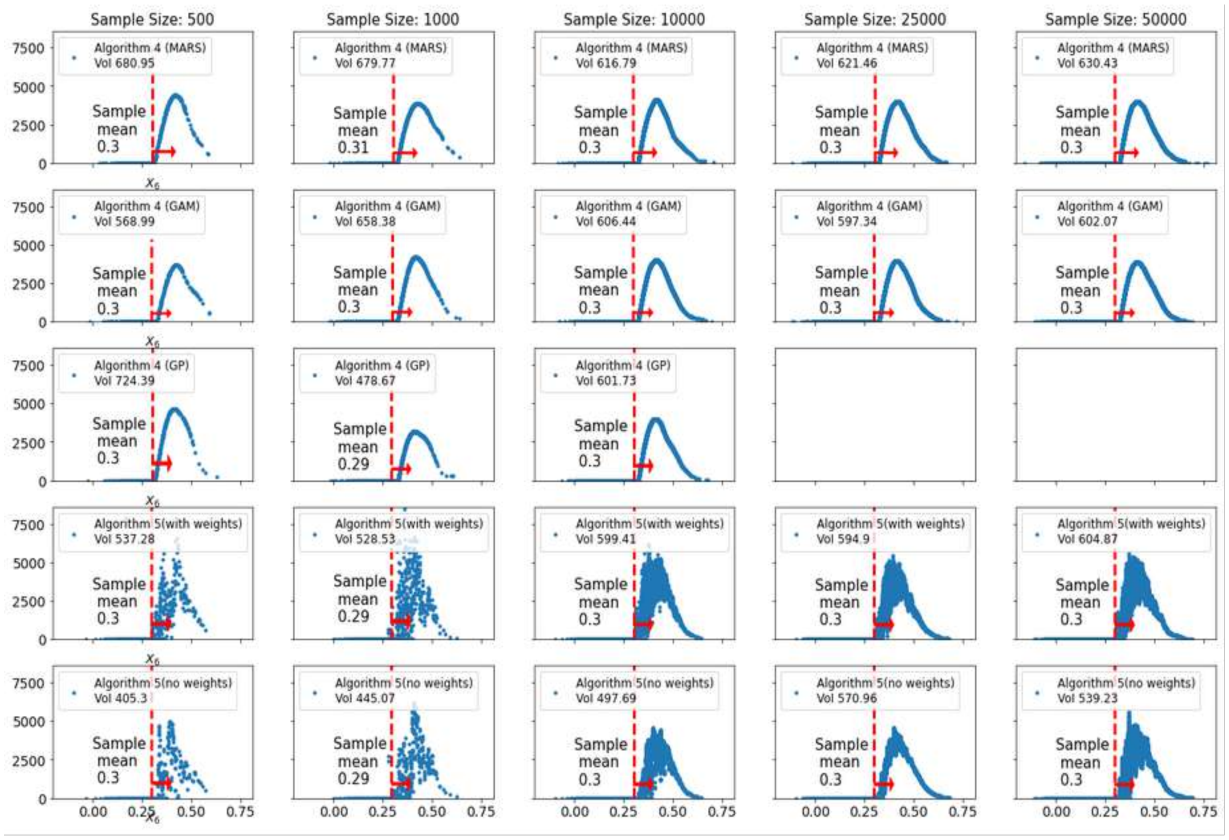


FIGURE 15. Result of information density estimation for the X_6 by different algorithms

The graphs in the first and second rows of Figure 15 show that the direction of concern for X_6 is at the left of $X_6 = 0.38$. All algorithms concur in this indication, with two exceptions. **Algorithm** 4 with GP fails at large sample sizes due to computational time (as in the previous cases). **Algorithm** 5, with and without wrights, produces noisy estimates, which are difficult to interpret at small sample sizes. Also, all algorithms agree in indicating that the maximum information density occurs at a value of $X_6 \sim 0.42$.

In order to assess the performance in the bivariate case, we use the same dataset to obtain estimates of the joint information density of the two most important variables (X_5, X_6). Figure(16) displays the results.

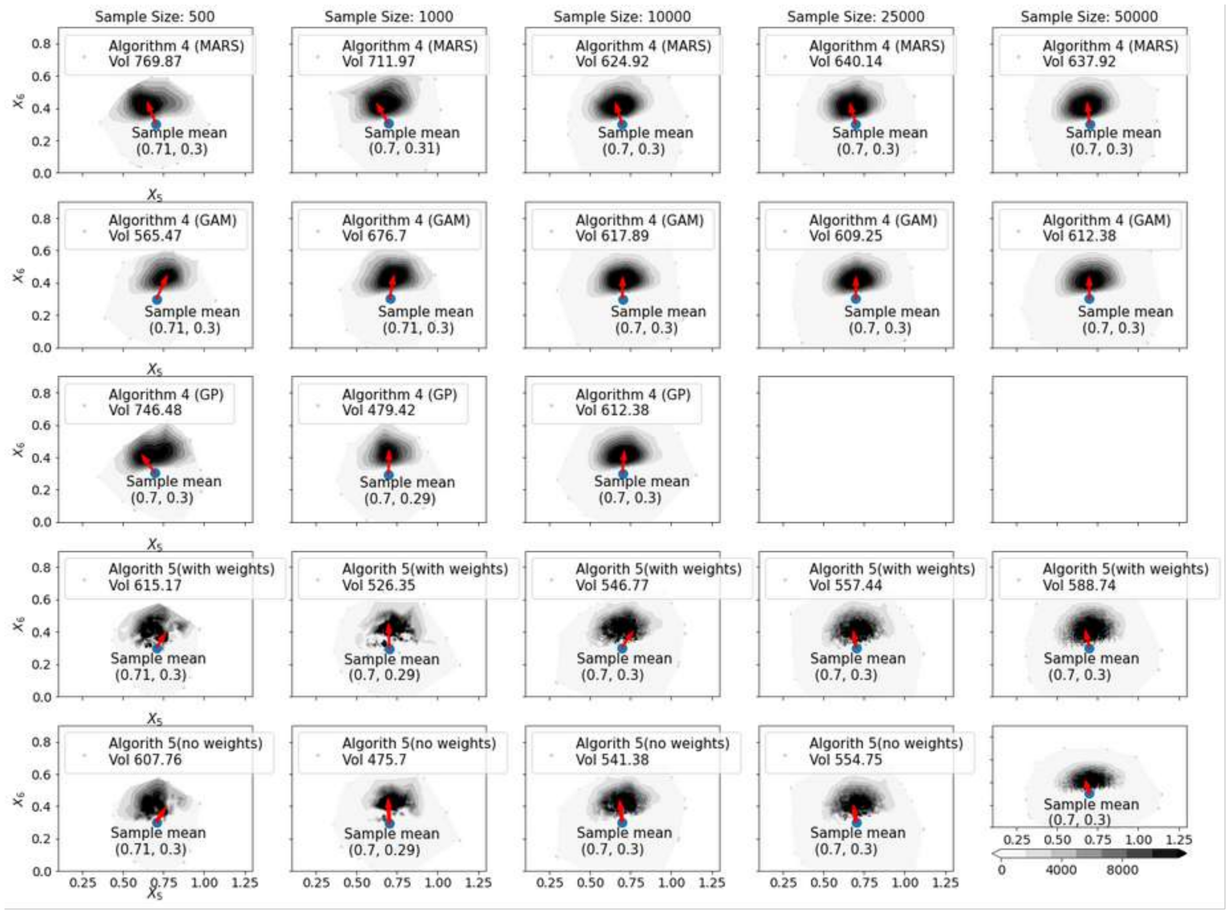


FIGURE 16. Result of information density estimation for the (X_5, X_6) by different algorithms

The graphs in Figure(16) show that all algorithms produce stabile visualizations starting from $N = 10000$ observations. We also find that the information density estimator given by **Algorithm** (4) (in each tested version) gives significantly smoother results with respect to **Algorithm** (5). At the same time, starting from a sample size above N=10000 observations, all graphs allow us to draw the same conclusions about the maximum information density, as well as about the areas where it is null. The area in which the pair (X_5, X_6) becomes informative is contained in the rectangle $[0.5, 0.9] \times [0.3, 0.6]$ at all sample sizes. The direction of concern is evidenced in each graph by the red arrows that start at the baseline value (the population mean in our case) and end at the point of maximal value. For instance, if we take the fifth graph in the first row, the arrow originates at $(\hat{\mathbb{E}}(X_5), \hat{\mathbb{E}}(X_6)) = (0.7, 0.3)$ and ends at $(0.68, 0.41)$, with a length of $(\Delta_{X_5} = 0.02, \Delta_{X_6} = 0.11)$. Note that the arrow is almost aligned with the direction of X_6, showing that information on X_6 is more relevant than X_5. This is, indeed, in line with the corresponding information value: we have $\epsilon_{X_6} \simeq 500$, while $\epsilon_{X_5} \simeq 30$.

10.3. **Case Study 2.[37].** Figure (17) displays the results of estimating the information density by different algorithms for a single feature X_6.

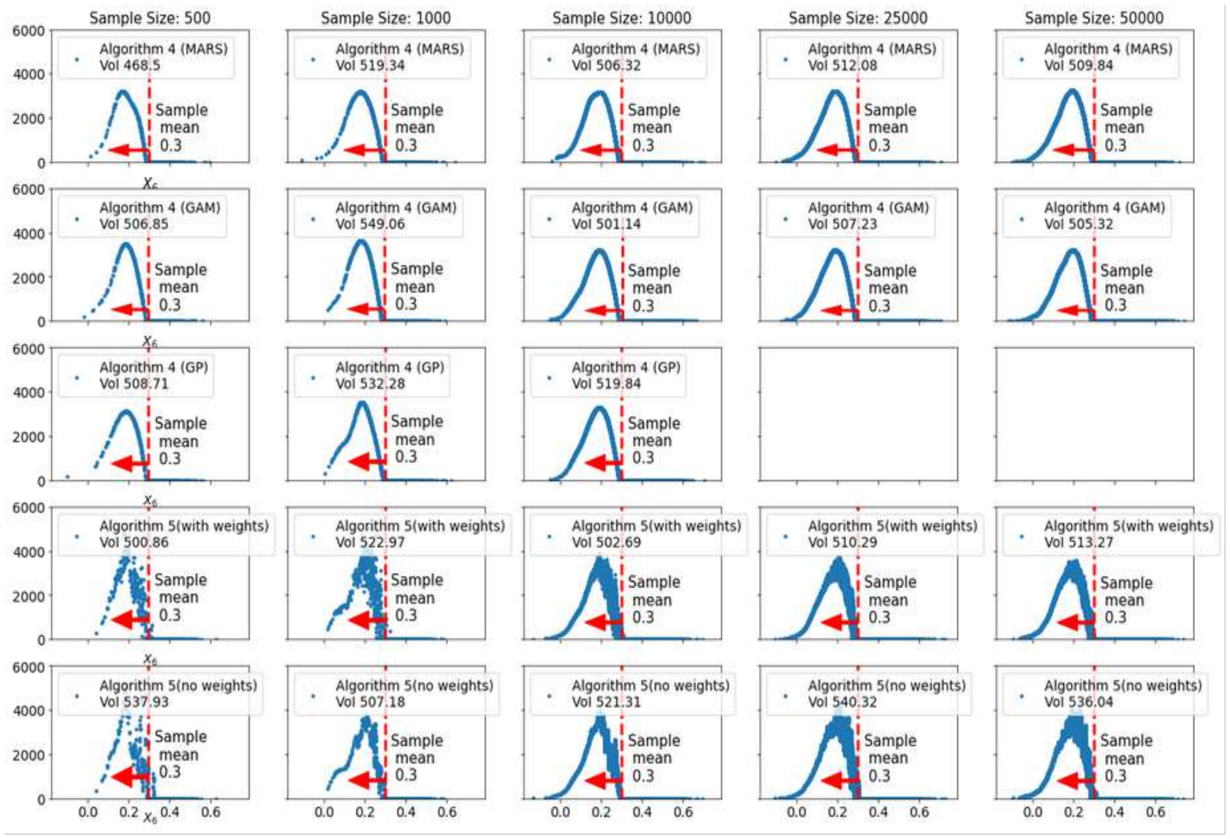


FIGURE 17. Result of information density estimation for the X_6 by different algorithms

All variants of **Algorithm** (4) give consistent results starting from sample size N= 1000 although the values of the estimated VoI vary slightly around N=500. In the case of the two versions of **Algorithm** (5) we can also observe stabilization of the estimates starting from sample size $N = 10000$. However, **Algorithm** (5) gives a relatively noisy estimator of the information density, the same as for first study case.

Based on **Algorithm** (4) and at all sample sizes, we conclude that is informative for values smaller than X_6=0.3. A similar conclusion is drawn based on the results of **Algorithm** (5), but the value threshold is noisy. Also, all algorithms at all sample sizes concur in indicating that information density is maximal at $X_6 = 0.2$.

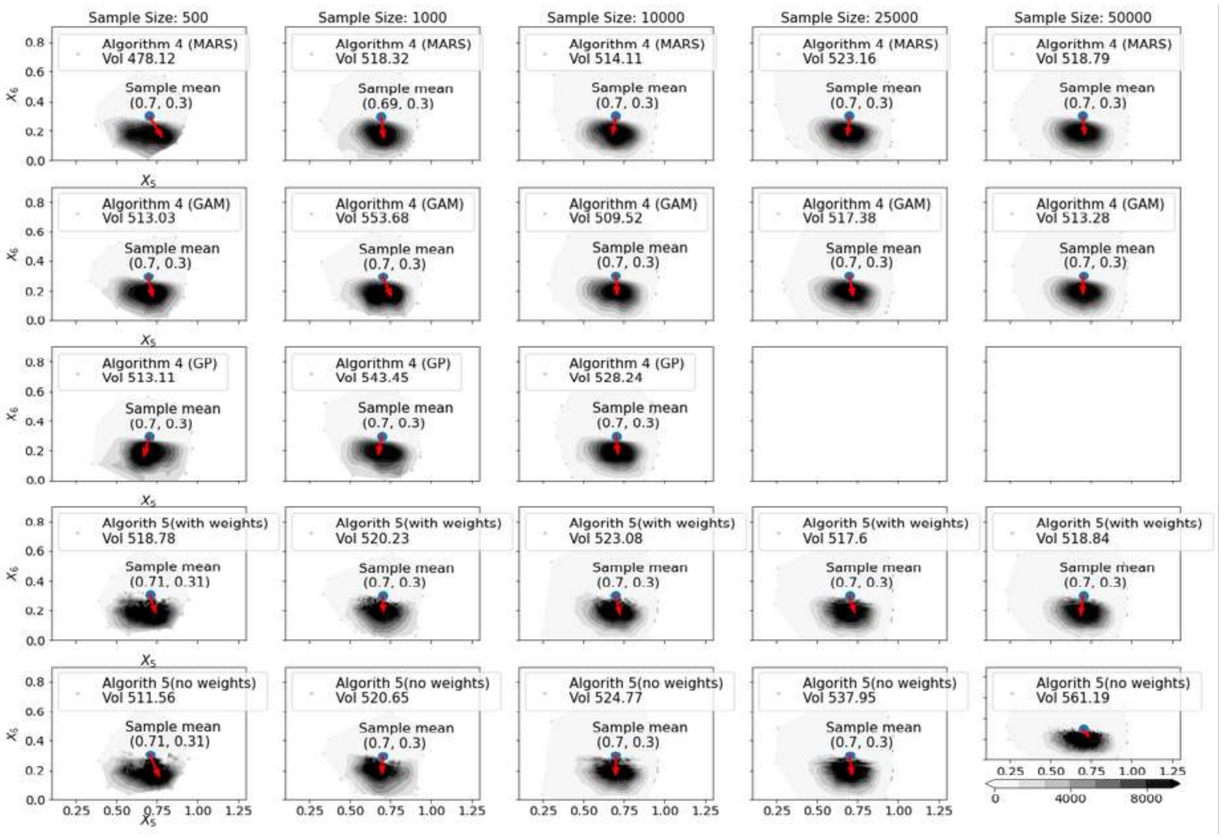


FIGURE 18. Result of information density estimation for the (X_5, X_6) by different algorithms

Figure (18) shows that all algorithms produce stabile visualizations starting from $N = 10000$ observations. At N=10000, 25000 and 50000, we have that most algorithms concur in indicating an estimate of the joint VoI of about $\hat{\epsilon}_{(X_5,X_6)} = 518$ with some exceptions. Specifically, **Algorithm** (5) with GP at N=1000 and N=10000 gives estimates of $\hat{\epsilon}_{(X_5,X_6)} \simeq 543$ and $\hat{\epsilon}_{(X_5,X_6)} \simeq 528$, respectively; also, **Algorithm** (5) without LassoLars weights tends yields upward biased estimates with $\hat{\epsilon}_{(X_5,X_6)} \simeq 538$ and $\hat{\epsilon}_{(X_5,X_6)} \simeq 561$ at $N = 25000$ and $N = 50000$, respectively. However, all graphs show that the region where information about the pair (X_5, X_6) is active is a region enclosed in the rectangle $[0.5, 0.9] \times [0, 0.3]$ at all sample sizes. The direction of interest marked by the red arrow becomes relatively stable above N=1000 observations. In this case, it takes its origin at $(\hat{\mathbb{E}}(X_5), \hat{\mathbb{E}}(X_6)) = (0.7, 0.3)$ and ends at the pont $(0.69, 0.2)$, with a length of $(\Delta_{X_5} = 0.01, \Delta_{X_6} = -0.1)$ $(0.4, 0.5)$. This clearly indicates that the shift in the direction of X_6 is greater than the one in the direction of X_5.

Let us now look at further insights about other inputs. We consider $N = 50000$ and study information density for additional individual inputs as well as input pairs. Figures 19 and 20 report results using **Algorithm** 5 with MARS.

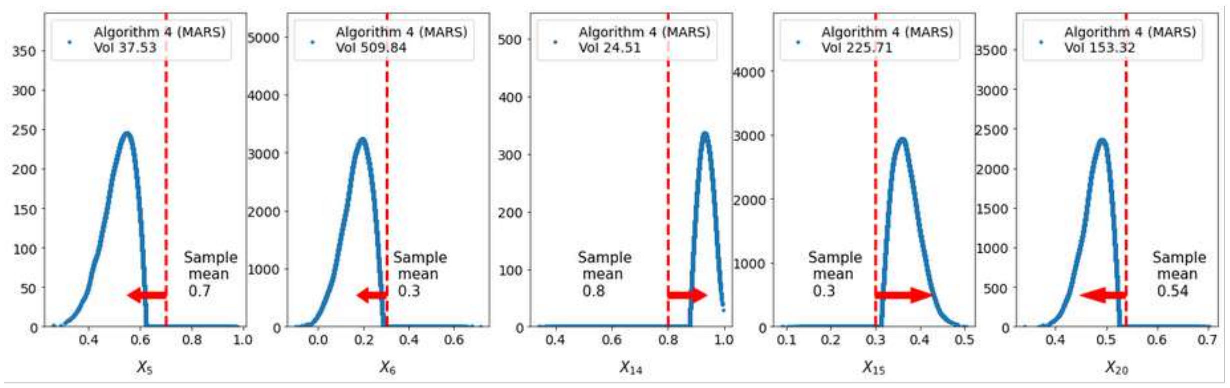


FIGURE 19. Result of information density estimation for individual inputs by **Algorithm** 4(MARS) for N=50000

The first graph in Figure 19 shows that the direction of concern for X_5 is left of its current base case. The second graph shows that the direction of concern for X_{14}, instead, is right of its current base case value. The input X_{14} becomes active for values $X_{14} \geq 0.9$ and information on this input is most valuable for values of X_{14} at about 0.95. Similarly, also for X_{15} the direction of concern is right of its current base case value. The region of maximum information value is concentrated around the value $X_{15} = 0.37$. The most informative feature in terms of VoI is X_6, for which $\hat{\epsilon}_{X_6} = 509$. Moreover, the direction of

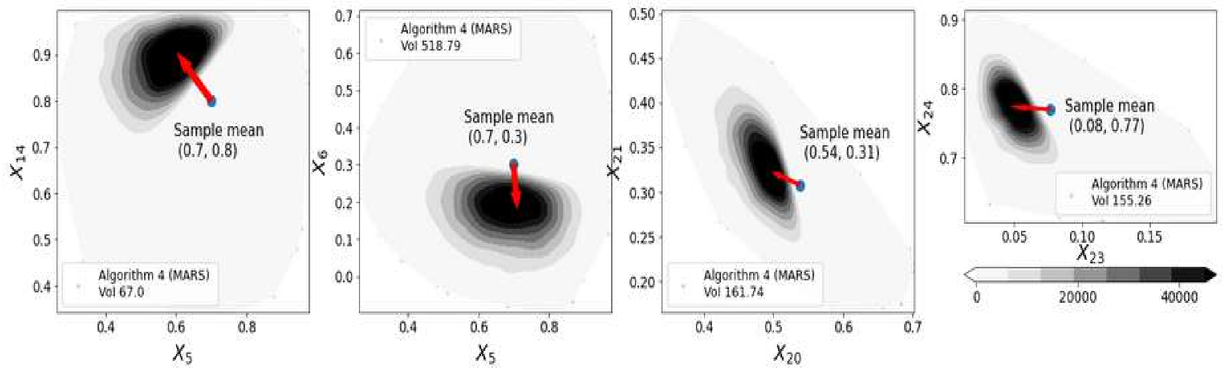


FIGURE 20. Result of information density estimation for different grups of variables by **Algorithm 4** (MARS) for N=50000

interest is to the left of the current base value, and the information density itself takes a maximum value around $X_6 = 0.18$.

The graphs in Figure 20 show the information density estimates for two-dimensional cases. The first two graphs show the estimates for (X_5, X_{14}) and (X_5, X_6), respectively. The activation area for the first graph is contained in the $[0.4, 0.8] \times [0.7, 1]$ area, while that for the second graph is contained in $[0.4, 0.8] \times [0, 0.3]$. The direction of concern for the first graph is characterized by length $(\Delta_{X_5}, \Delta_{X_{14}}) = (-0.2, 0.14)$, and for the second graph by $(\Delta_{X_5}, \Delta_{X_{14}}) = (0.01, -0.1)$. Intuitively, the direction of the arrows shows also the direction of greater informativeness. For instance, the arrow in the second graph is almost vertical, indicating that information on X_6 is more valuable than information on X_5. This fact is in agreement with the corresponding information values, as we have $\widehat{\epsilon}_{X_6} = 509$ and $\widehat{\epsilon}_{X_5} = 36$. The direction of concern is towards lower values of X_6. Similarly, last graph of Figure 20, the direction of concern is characterized by a length of $(\Delta_{X_{23}}, \Delta_{X_{24}}) = (-0.03, 0.003)$. Thus, we can conclude that the informativeness of X_{24} is very low in comparison with X_{23}. The direction of concern is towards lower values of X_{23}.

11. Summary

We have presented theoretical proof of the central limit theorem for the nearest neighbors estimator of the Value of the Information. We further proposed an improvement for the nearest neighbors algorithm by the weights derived from a model from the class of linear models. To test our claims we performed many experiments using two test cases widely used in the literature. Results confirmed that using weights in the nearest neighbors algorithm can increase the accuracy of the final estimate. The best algorithm to use when computing VoI estimate, based on our experiments is the MARS algorithm which suffers less than other algorithms from the curse of dimensionality and can be used for large data sets. We have also used the MARS algorithm to find the most important features for both case studies which can have a crucial impact on understanding the decision problem which is being solved.

We have then modified the algorithms to estimate information density, a recently presented quantity that enriches the VoI through a graphical method of sensitivity analysis. In this case, for the estimation of information density, we recommend using several algorithms at the same time, as we have observed in experiments that depending on the input data, different solutions may turn out to be more or less performing. Therefore, relying on multiple algorithms in the absence of closed-form expressions is the best guarantee for analysts to have robust conclusions to be then communicated to decision-makers.

References

[1] G B. Hazen , E Borgonovo and X. Lu. Information density in decision analysis. 2022.

[2] N S. Altman. An introduction to kernel and nearest-neighbor nonparametric regression. *American Statistician*, 46(3):175–185, August 1992.

[3] G. Baio and A P. Dawid. Probabilistic sensitivity analysis in health economics. *Statistical Methods in Medical Research*, 24(6):615–634, 2015. PMID: 21930515.

[4] J M. Bernardo and A F. M. Smith. *Bayesian Theory.* Wiley&Sons, New York, NY, USA, second edition, 2000.

[5] E Borgonovo. A New Uncertainty Importance Measure. *Reliability Engineering and System Safety*, 92(6):771–784, 2007.

[6] R B. Bratvold, J E. Bickel, and H P. Lohne. Value of information in the oil and gas industry: Past, present and future. *SPE Reservoir Evaluation and Engineering*, 12(4):630–638, 2009.

[7] H Chen and Y Xia. A normality test for high-dimensional data based on the nearest neighbor approach. *Journal of the American Statistical Association*, 0(0):1–13, 2021.

[8] W S. Cleveland and S J. Devlin. Locally weighted regression: An approach to regression analysis by local fitting. *Journal of the American Statistical Association*, 83:596–610, 1988.

[9] D. Coyle and J. Oakley. Estimating the expected value of partial perfect information: a review of methods. *Eur. J. Health Econ.*, 9(3):251–259, August 2008.

[10] L Devroye, L Gyorfi, A Krzyzak, and G Lugosi. On the Strong Universal Consistency of Nearest Neighbor Regression Function Estimates. *The Annals of Statistics*, 22(3):1371 – 1385, 1994.

[11] D B. Dunson. Statistics in the Big Data Era: Failures of the Machine. *Statistics and Probability Letters*, 136:4–9, 2018.

[12] J C. Felli and G B. Hazen. Sensitivity Analysis and the Expected Value of Perfect Information. *Medical Decision Making*, 18:95–109, 1998.

[13] J C. Felli and G B. Hazen. A Bayesian Approach to Sensitivity Analysis. *Health Economics*, 8:263–268, 1999.

[14] E Fix and J L. Hodges. *Discriminatory Analysis: Nonparametric Discrimination: Consistency Properties.* USAF School of Aviation Medicine, 1951.

[15] J H. Friedman. Multivariate adaptive regression splines. *Ann. Statist*, 1991.

[16] B E. Fristedt and Lawrence F G. *A modern approach to probability theory.* Springer, 1997.

[17] T Hastie and R Tibshirani. *Generalized additive models.* Wiley Online Library, 1990.

[18] G B. Hazen. Sensitivity analysis via information density. 2014.

[19] A Heath, I Manolopoulou, and G Baio. A Review of Methods for Analysis of the Expected Value of Information. *Medical Decision Making*, 37(7):747–758, 2017.

[20] C C. Holmes and N M. Adams. A probabilistic nearest neighbour method for statistical pattern recognition. *Journal of the Royal Statistical Society. Series B: Statistical Methodology*, 64(2):295–306, 2002.

[21] J M. Keisler, Z A. Collier, E Chu, N Sinatra, and I Linkov. Value of information analysis: The state of application, 2014.

[22] H Millwater, G Singh, and M Cortina. Development of a localized probabilistic sensitivity method to determine random variable regional importance. *Reliability Engineering & System Safety*, 107:3–15, 2012.

[23] P K. Mondal, M Biswas, and A K. Ghosh. On high dimensional two-sample tests based on nearest neighbors. *Journal of Multivariate Analysis*, 141:168–178, 2015.

[24] H O. V. Myklebust, J Eidsvik, I B. Sperstad, and D Bhattacharjya. Value of information analysis for complex simulator models: Application to wind farm maintenance. *Decision Analysis*, 17(2):134–153, 2020.

[25] J E. Oakley. Decision-theoretic Sensitivity Analysis for Complex Computer Models. *Technometrics*, 51(2):121–129, 2009.

[26] J E. Oakley, A Brennan, P Tappenden, and J Chilcott. Simulation sample sizes for Monte Carlo partial EVPI calculations. *Journal of Health Economics*, 29(3):468–477, 2010.

[27] J E. Oakley and A O'Hagan. Probabilistic Sensitivity Analysis of Complex Models: a Bayesian Approach. *Journal of the Royal Statistical Society, Series B*, 66(3):751–769, 2004.

[28] J W. Pratt, H Raiffa, and R Schlaifer. *Introduction to Statistical Decision Theory.* MIT Press, Cambridge Massachusetts (USA), 1995.

[29] H. Raiffa and R. Schlaifer. *Applied Statistical Decision Theory.* Harvard University Press, Boston, 1961.

[30] F P. Ramsey. Weight or the Value of Knowledge. *The British Journal for the Philosophy of Science*, 41(1):1–4, 1990.

[31] C E. Rasmussen and C K. I. Williams. *Gaussian processes for machine learning.* The MIT Press, 2006.

[32] Mohsen Sadatsafavi, Nick Bansback, Zafar Zafari, Mehdi Najafzadeh, and Carlo Marra. Need for speed: An efficient algorithm for calculation of single-parameter expected value of partial perfect information. *Value in Health*, 16(2):438–448, 2013.

[33] A Saltelli and S Tarantola. On the Relative Importance of Input Factors in Mathematical Models: Safety Assessment for Nuclear Waste Disposal. *Journal of the American Statistical Association*, 97(459):702–709, 2002.

[34] F Santosa and W W. Symes. Linear inversion of band-limited reflection seismograms. *SIAM Journal on Scientific and Statistical Computing*, 7(4):1307–1330, 1986.

[35] M Strong and J E. Oakley. An efficient method for computing partial expected value of perfect information for correlated inputs. *Medical Decision-Making*, 33:755–766, 2013.

[36] M Strong and J E. Oakley. An efficient method for computing single-parameter partial expected value of perfect information. *Medical Decision Making*, 33(6):755–766, 2013. PMID: 23275450.

[37] M Strong, J E. Oakley, and A Brennan. Estimating multiparameter partial expected value of perfect information from a probabilistic sensitivity analysis sample: A nonparametric regression approach. *Medical Decision Making*, 34(3):311–326, 2014. PMID: 24246566.

[38] M Strong, J E. Oakley, A Brennan, and P Breeze. Estimating the Expected Value of Sample Information Using the Probabilistic Sensitivity Analysis Sample: A Fast, Nonparametric Regression-Based Method. *Medical Decision Making*, 35(5):570–583, 2015.

[39] W W. Sun, X Qiao, and G Cheng. Stabilized nearest neighbor classifier and its statistical properties. *Journal of the American Statistical Association*, 111(515):1254–1265, 2016.

[40] R Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, 58(1):267–288, 1996.

[41] A Zabeo, J M. Keisler, D Hristozov, A Marcomini, and I Linkov. Value of information analysis for assessing risks and benefits of nanotechnology innovation. *Environmental Sciences Europe*, 31(1), 2019.

12. Appendix A

During the thesis, several more experiments were performed, in excess of the ones described in the main text of the thesis. In these appendices, we present additional ones, starting with experiments performed for VoI estimation for the first case study 6.1. Figures 21, 22, 23, 24 and 25 report VoI estimates yielded by **Algorithm** 2 and **Algorithm** 3 at $N = 500$, $N = 1000$, $N = 10000$, $N = 25000$ and $N = 50000$, respectively.

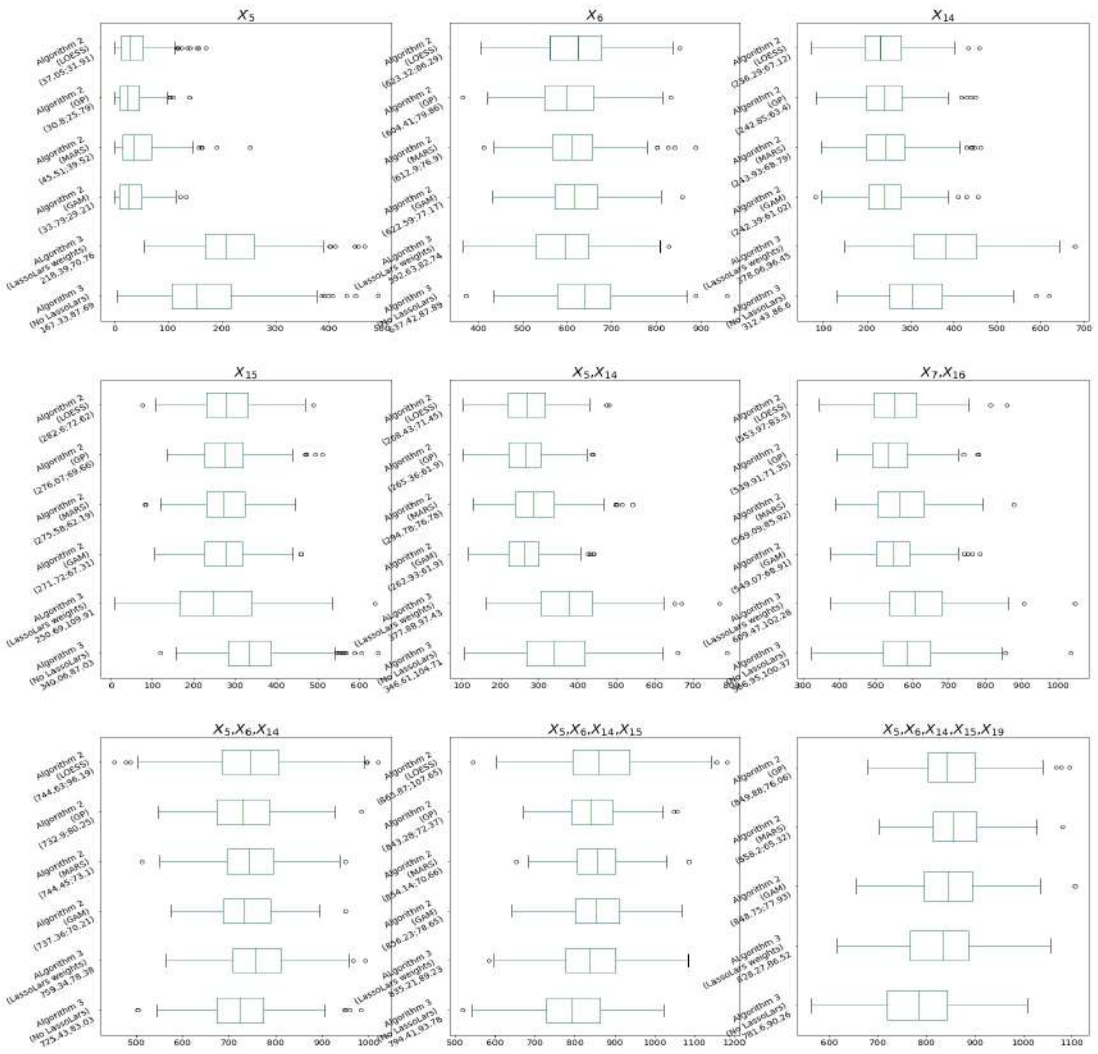


Figure 21. VoI estimation results for different groups of variables by **Algorithm** 2 and **Algorithm** 3 for N=500

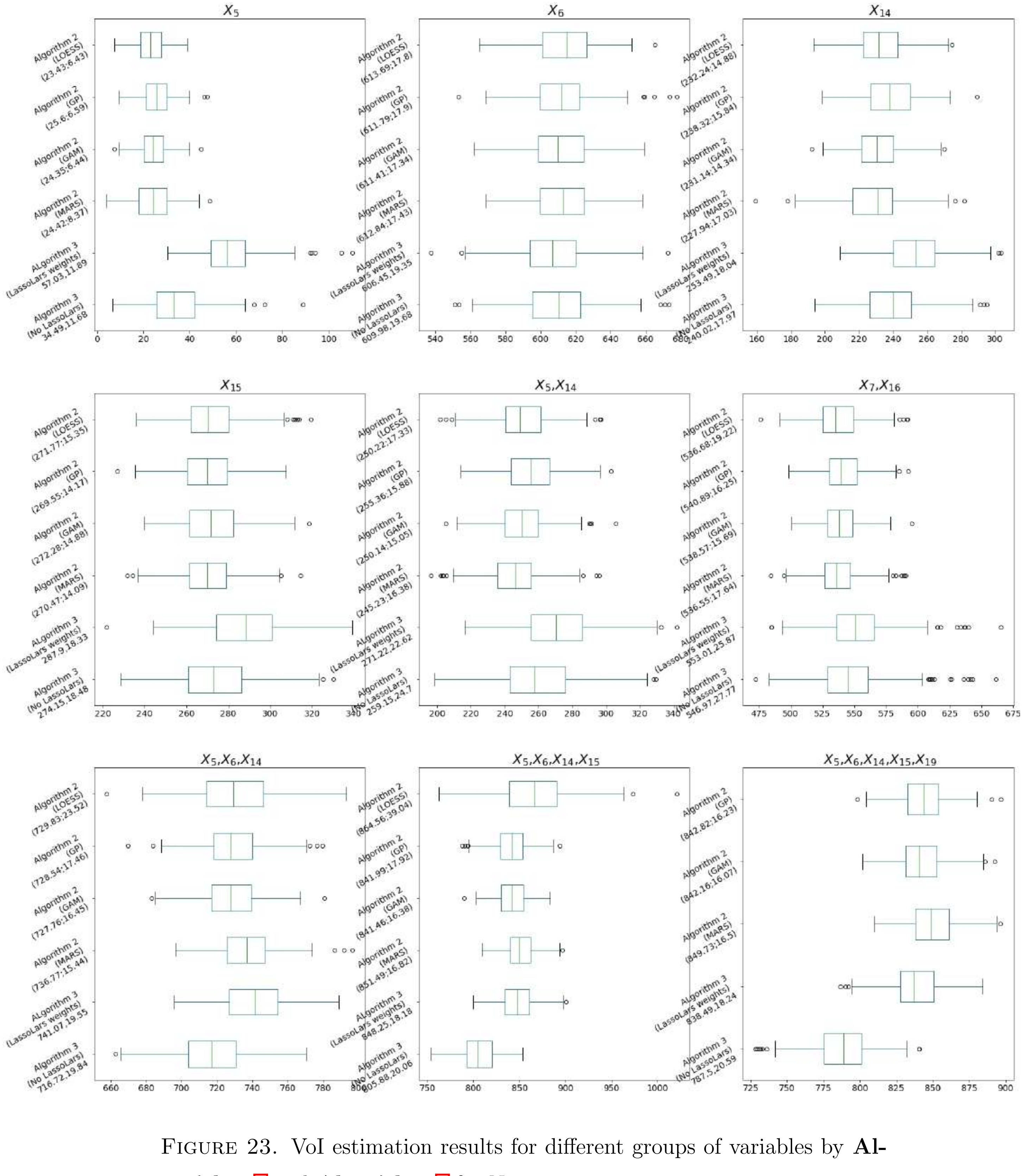


FIGURE 23. VoI estimation results for different groups of variables by **Algorithm** 2 and **Algorithm** 3 for N=10000

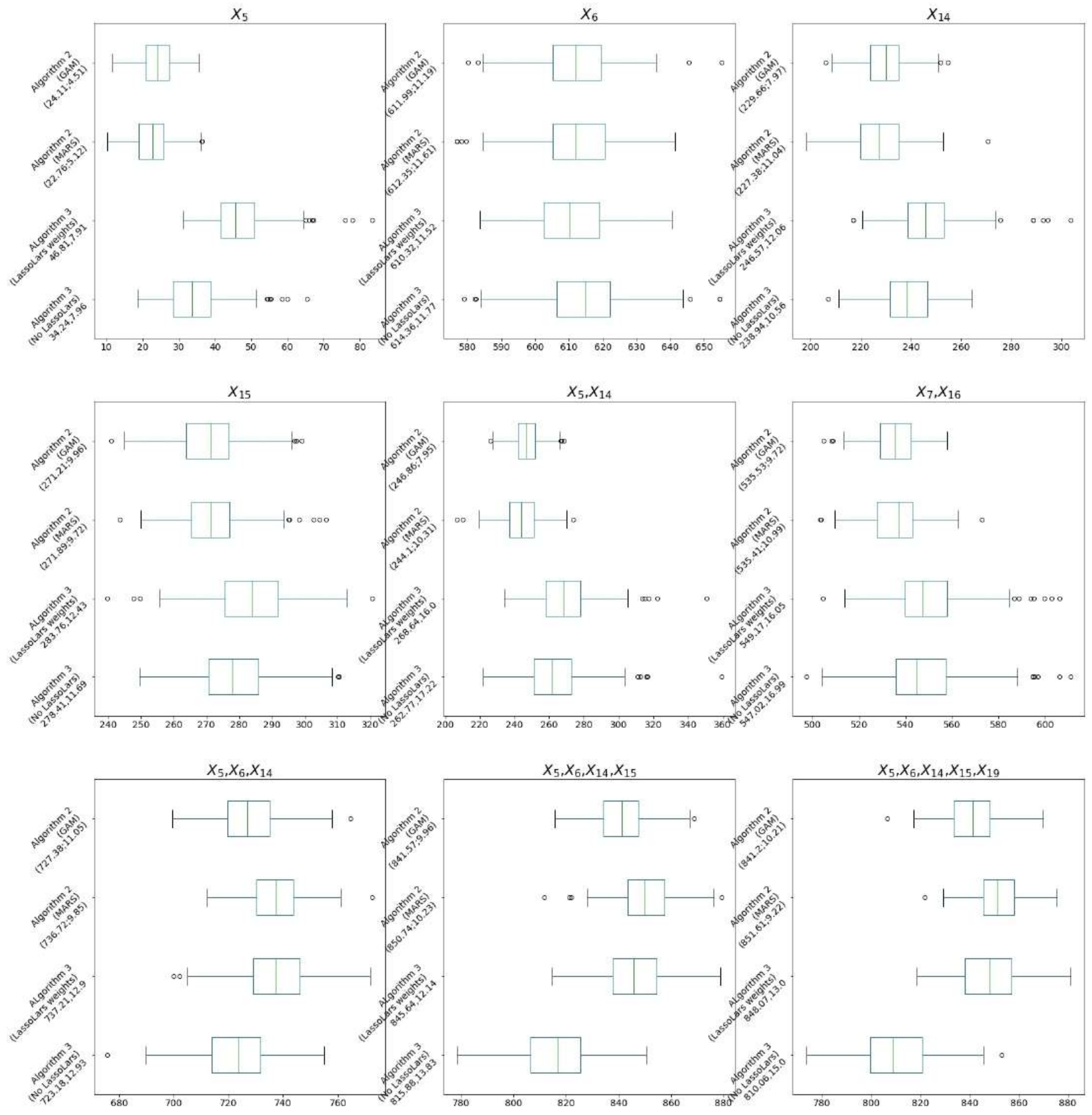


FIGURE 24. VoI estimation results for different groups of variables by **Algorithm** 2 and **Algorithm** 3 for N=25000

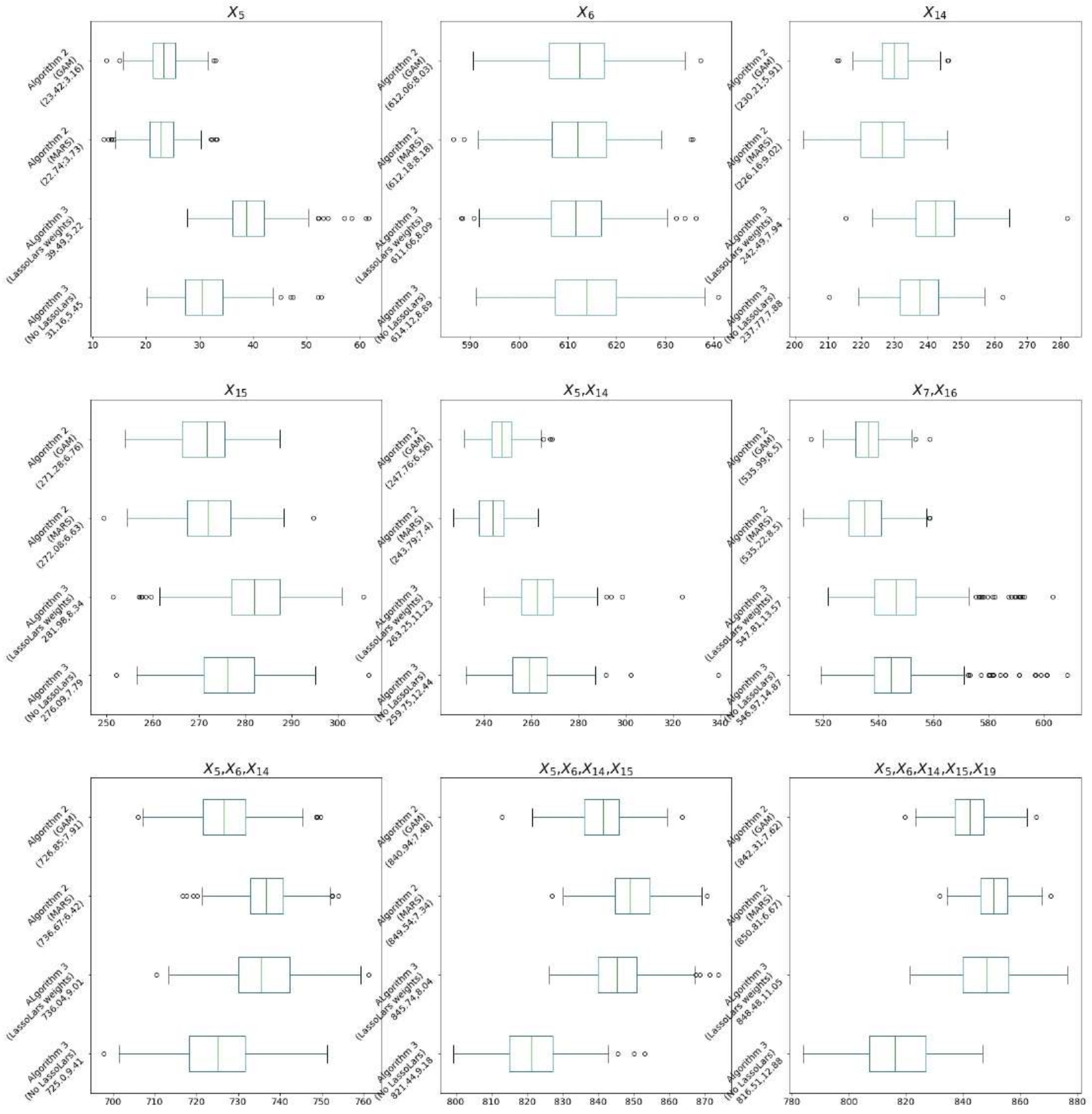


FIGURE 25. VoI estimation results for different groups of variables by **Algorithm** 2 and **Algorithm** 3 for N=50000

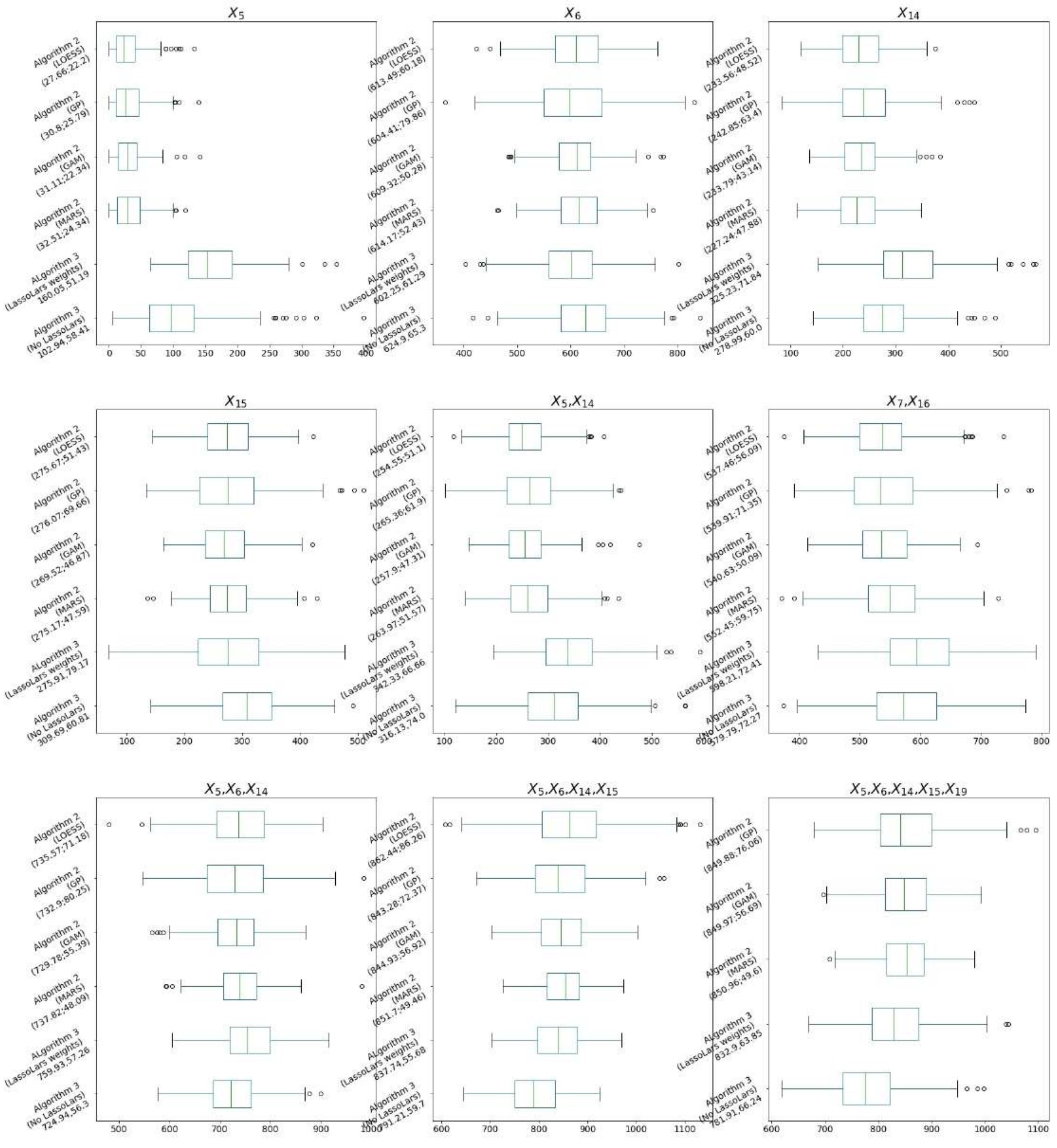


FIGURE 22. VoI estimation results for different groups of variables by **Algorithm** 2 and **Algorithm** 3 for N=1000

Here we present the results of further the experiments performed for VoI estimation for the second case study 6.2. Figures 26, 27, 28, 29 and 30 report results yielded by **Algorithm** 2 and **Algorithm** 3 at sample sizes of N=500, N=1000, N=10000, N=25000 and N=50000, respectively.

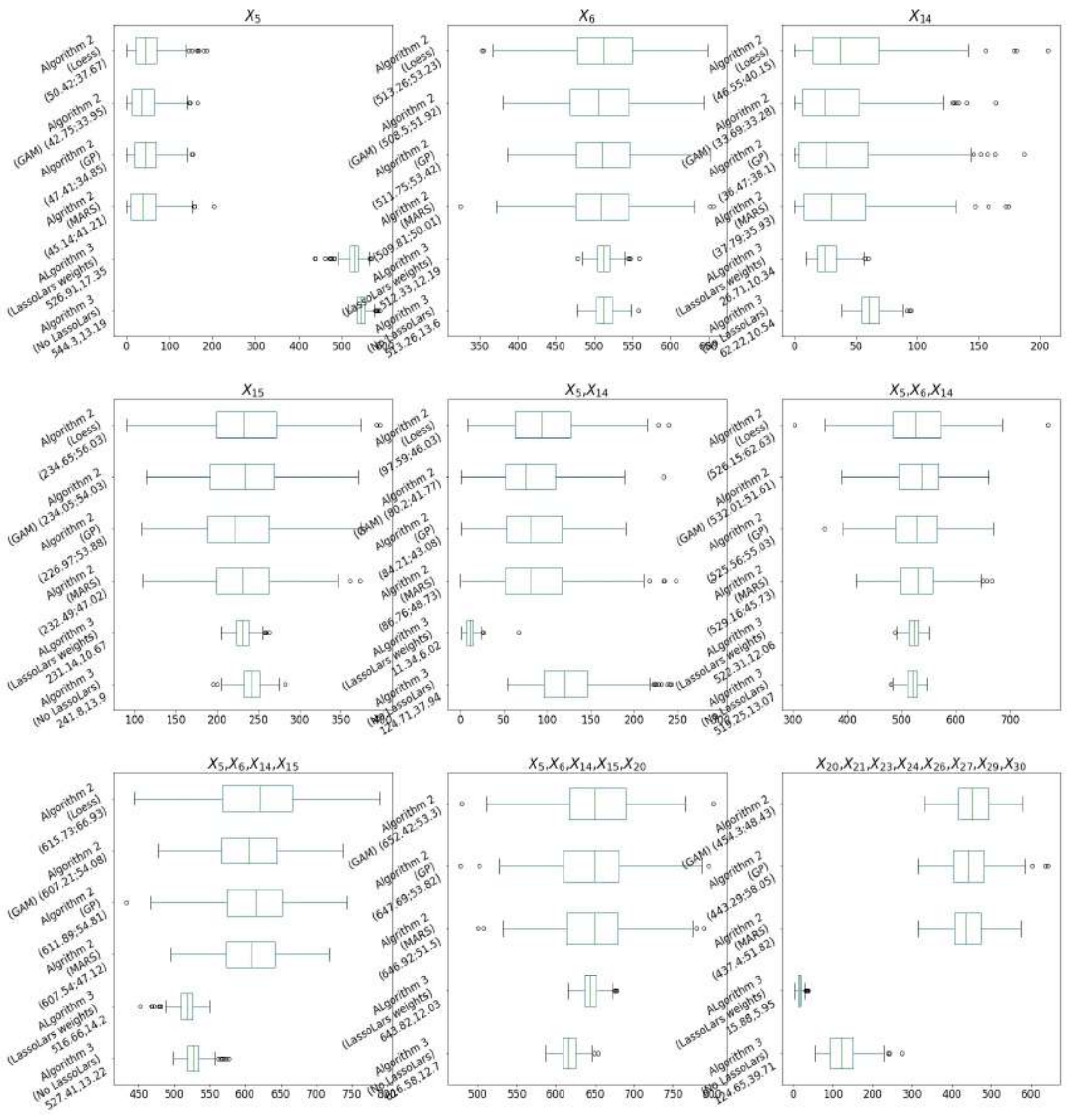


FIGURE 26. VoI estimation results for different groups of variables by **Algorithm** 2 and **Algorithm** 3 for N=500

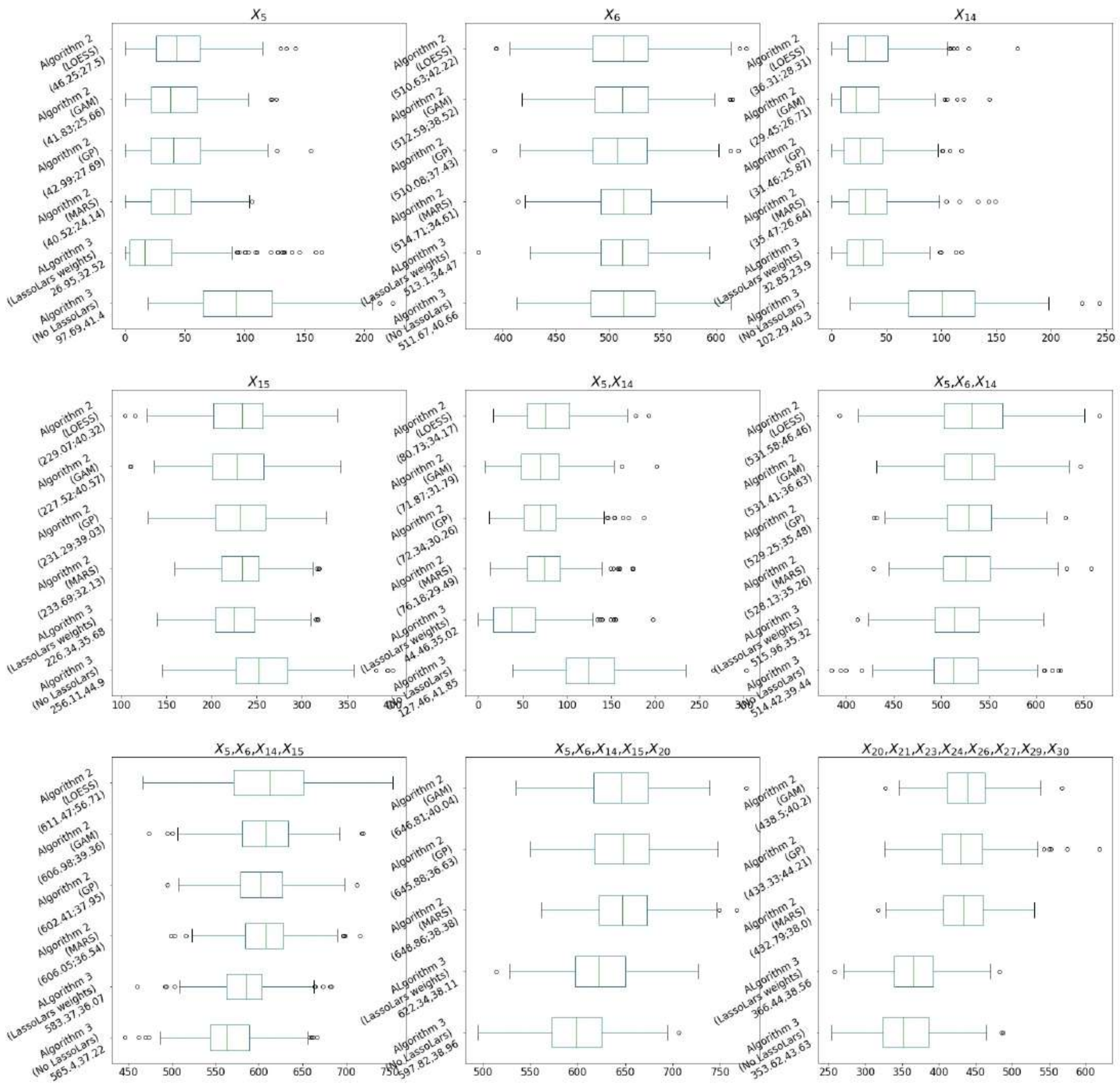


FIGURE 27. VoI estimation results for different groups of variables by **Algorithm** 2 and **Algorithm** 3 for N=1000

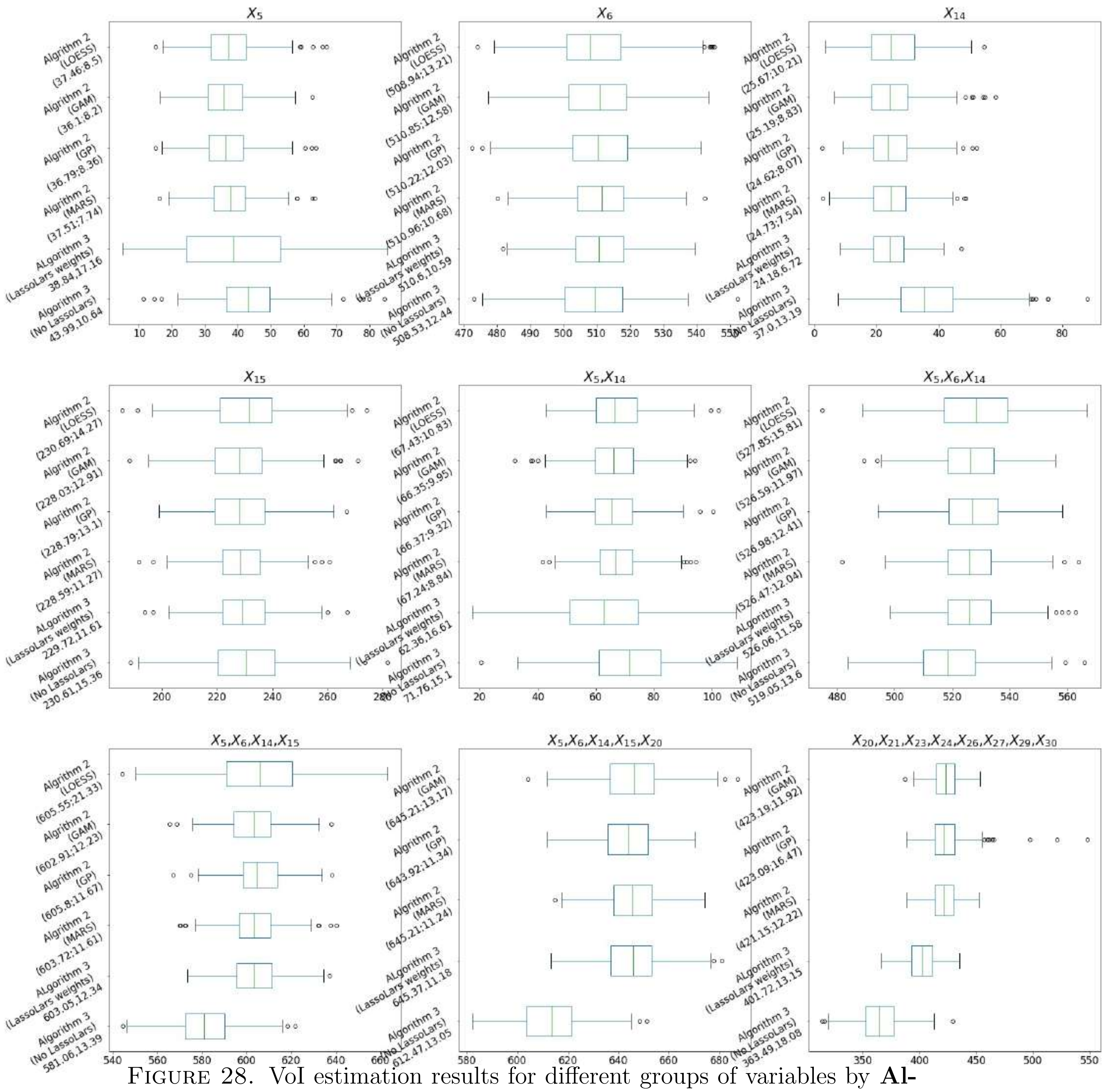


FIGURE 28. VoI estimation results for different groups of variables by **Algorithm** 2 and **Algorithm** 3 for N=10000

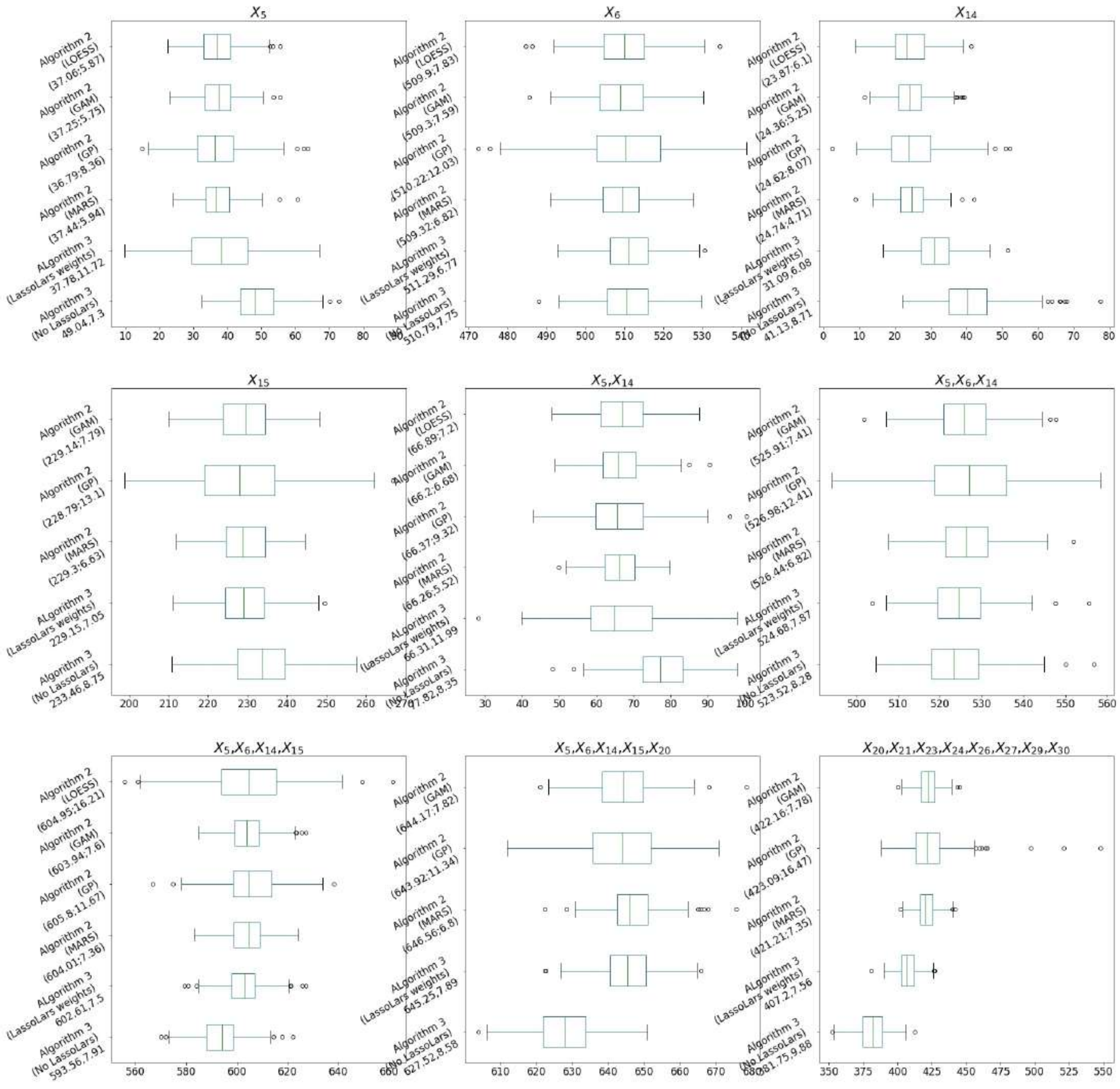


FIGURE 29. VoI estimation results for different groups of variables by **Algorithm** 2 and **Algorithm** 3 for N=25000

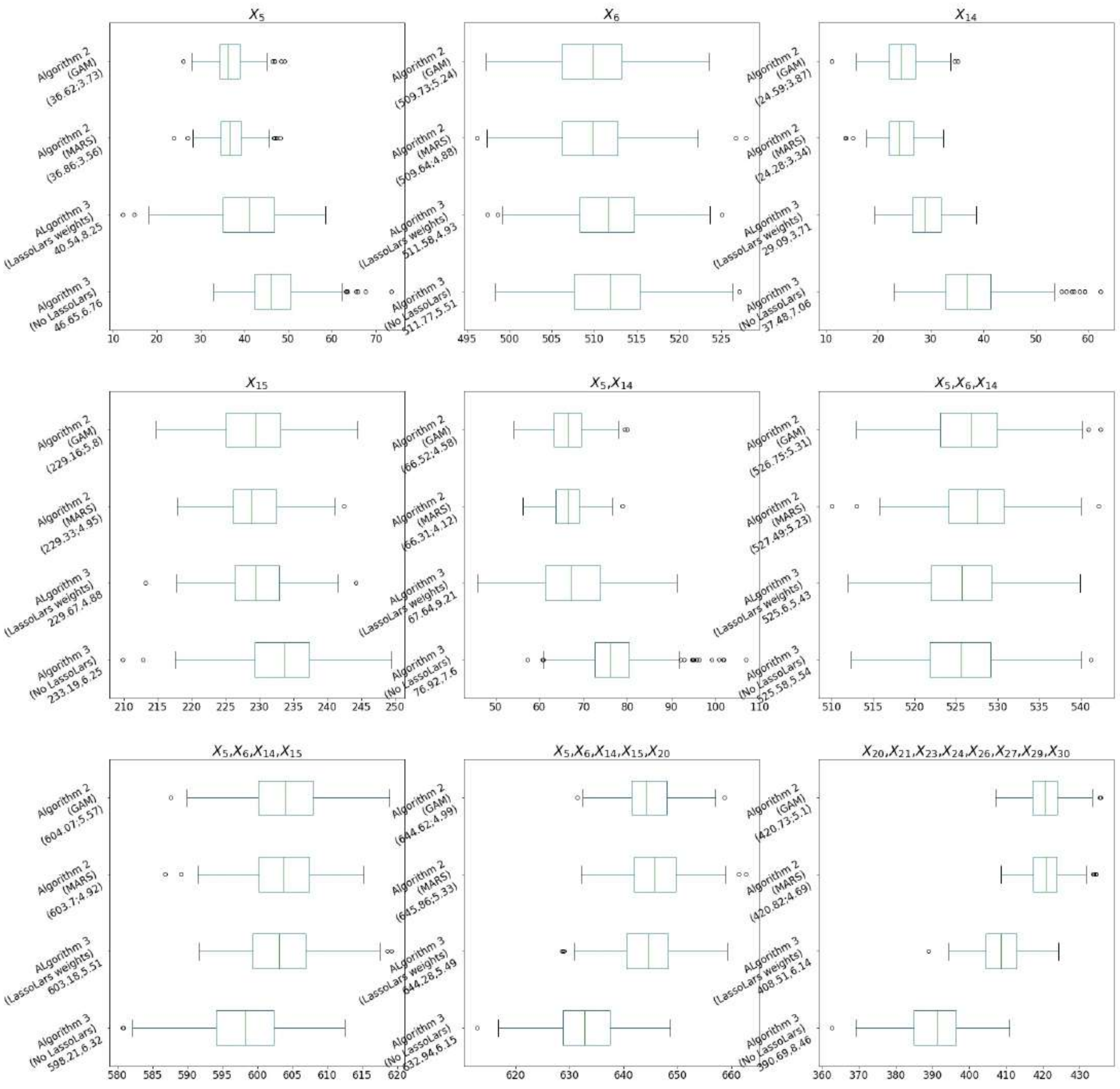


FIGURE 30. VoI estimation results for different groups of variables by **Algorithm** 2 and **Algorithm** 3 for N=50000

13. Appendix B

Here we present the rest of the experiments performed for information density estimation for the first case study 6.1. Figures 31, 32, 33, 34 and 35 report information density estimates yielded by **Algrithm** 4 and **Algrithm** 5 for different groups of variables.

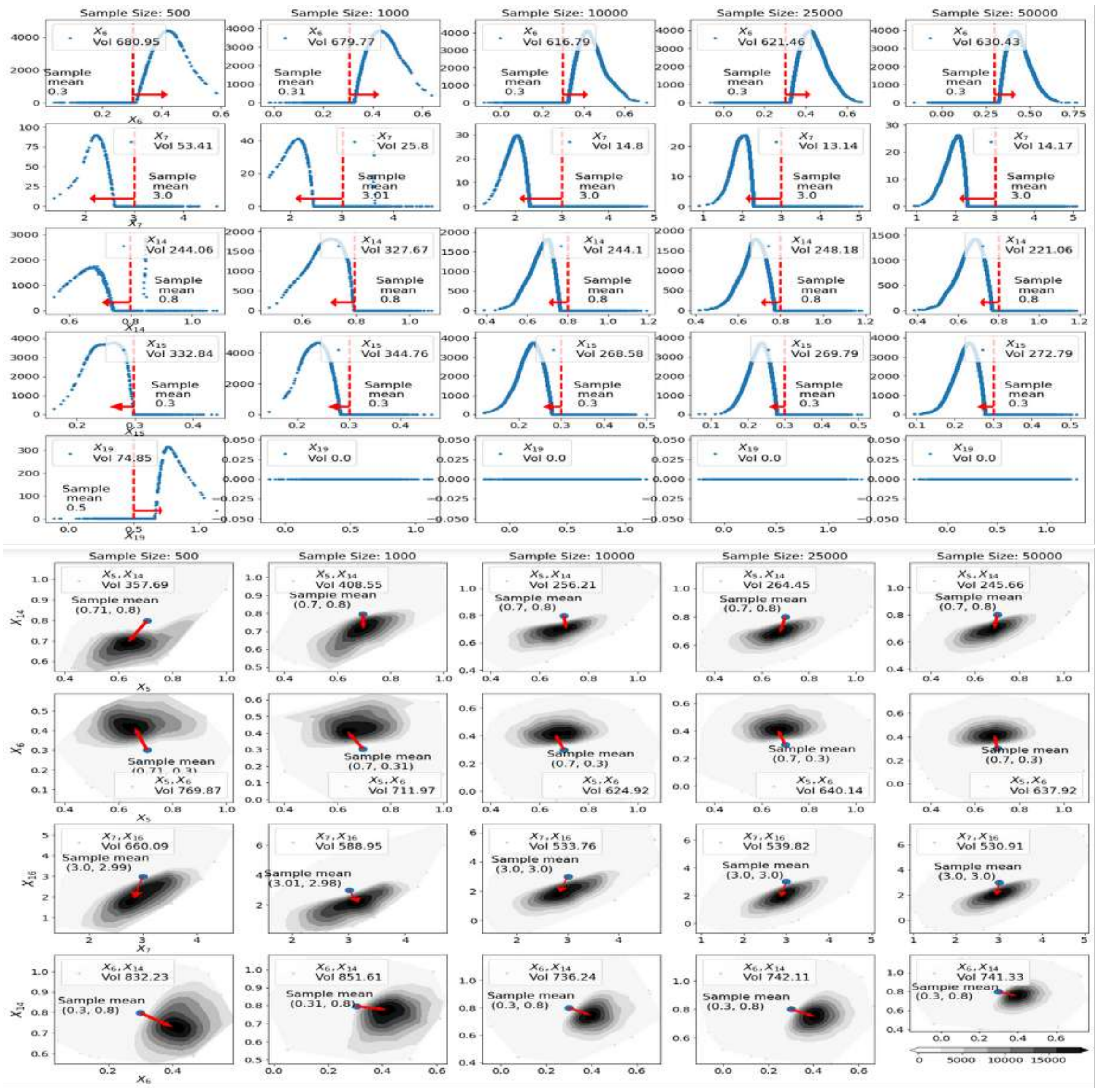


Figure 31. Result of information density estimation for different grups of variables by **Algorithm** 4 (MARS)

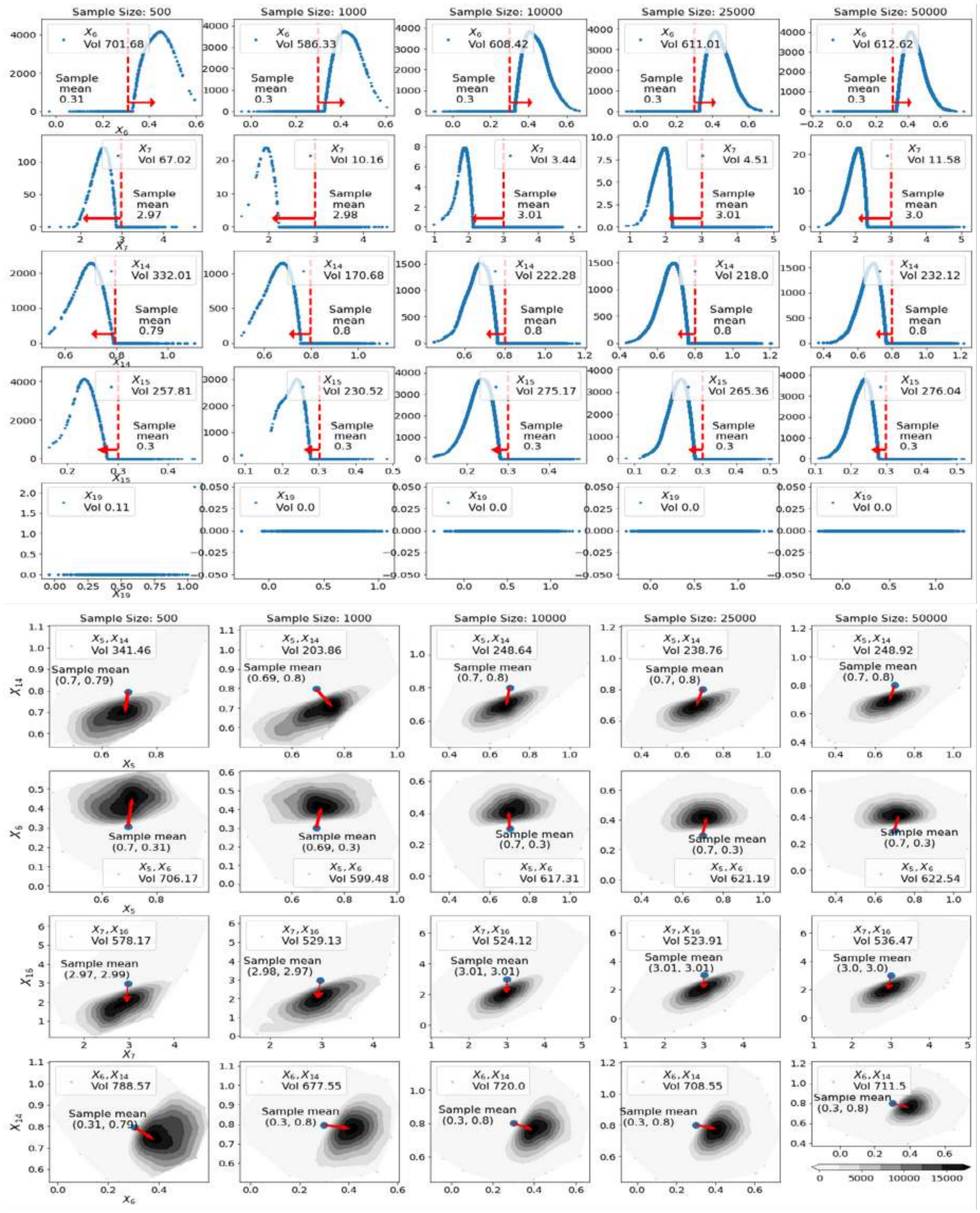


FIGURE 32. Result of information density estimation for different grups of variables by **Algorithm** 4 (GAM)

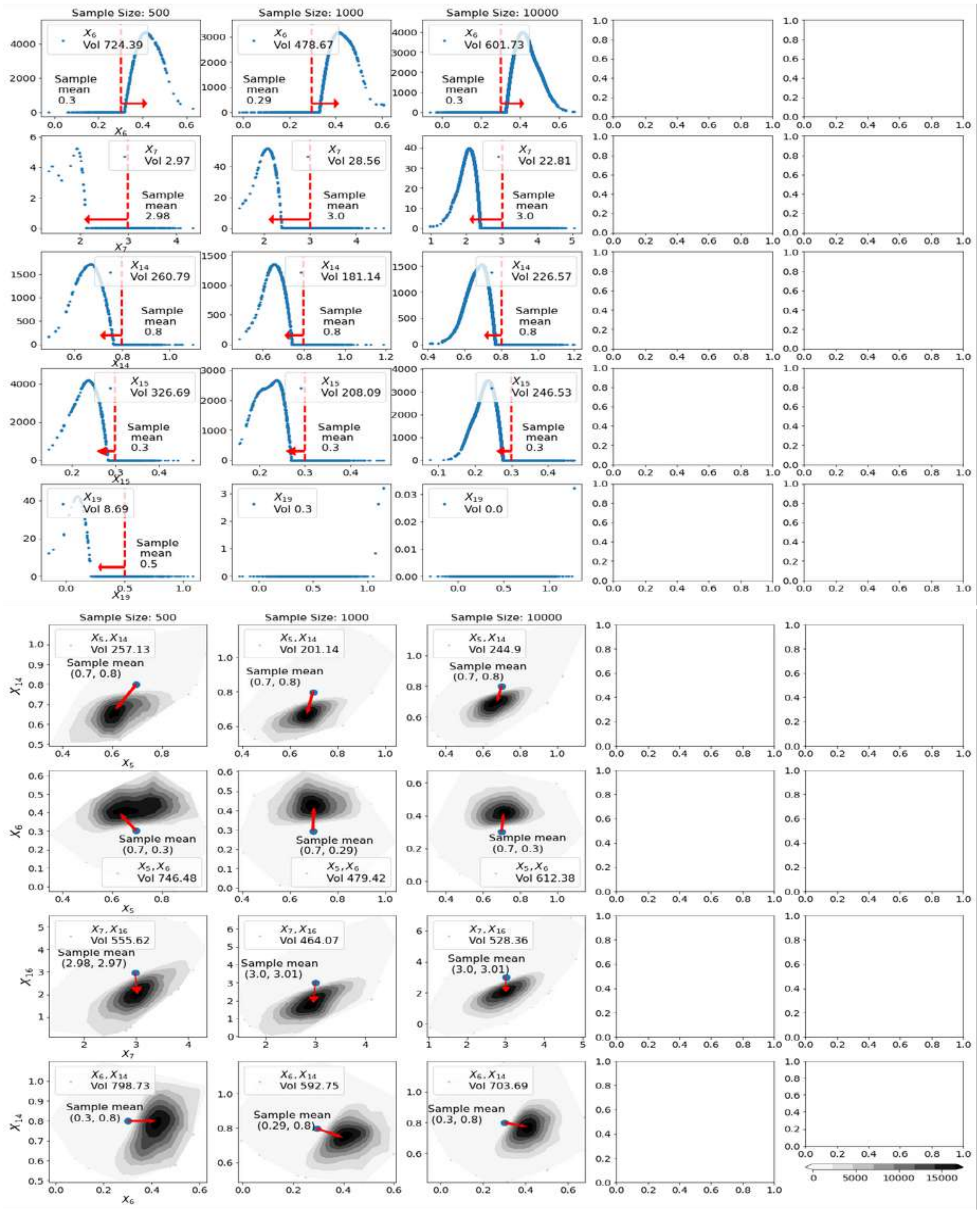


FIGURE 33. Result of information density estimation for different variables by **Algorithm** 4 (GP)

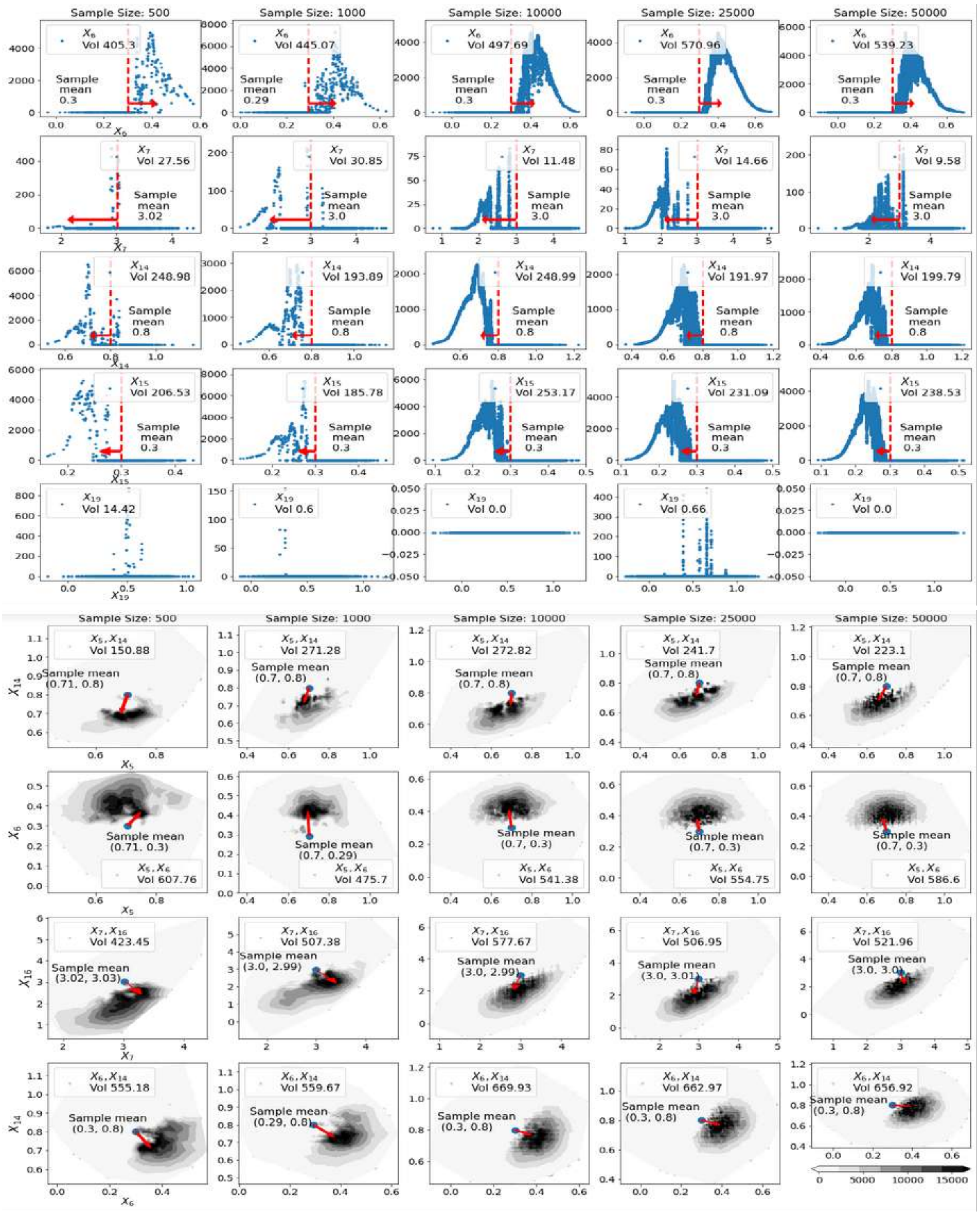
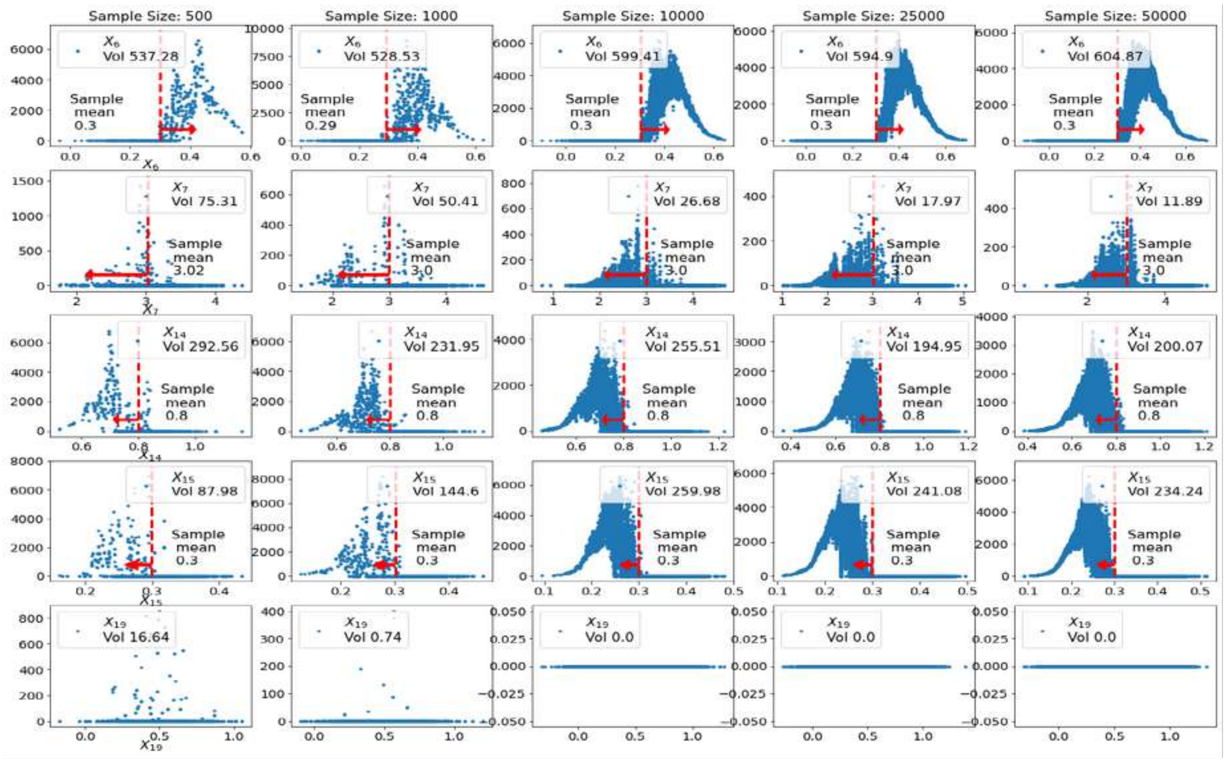


FIGURE 34. Result of information density estimation for different grups of variables by **Algorithm** 5 (without LassoLars weights)



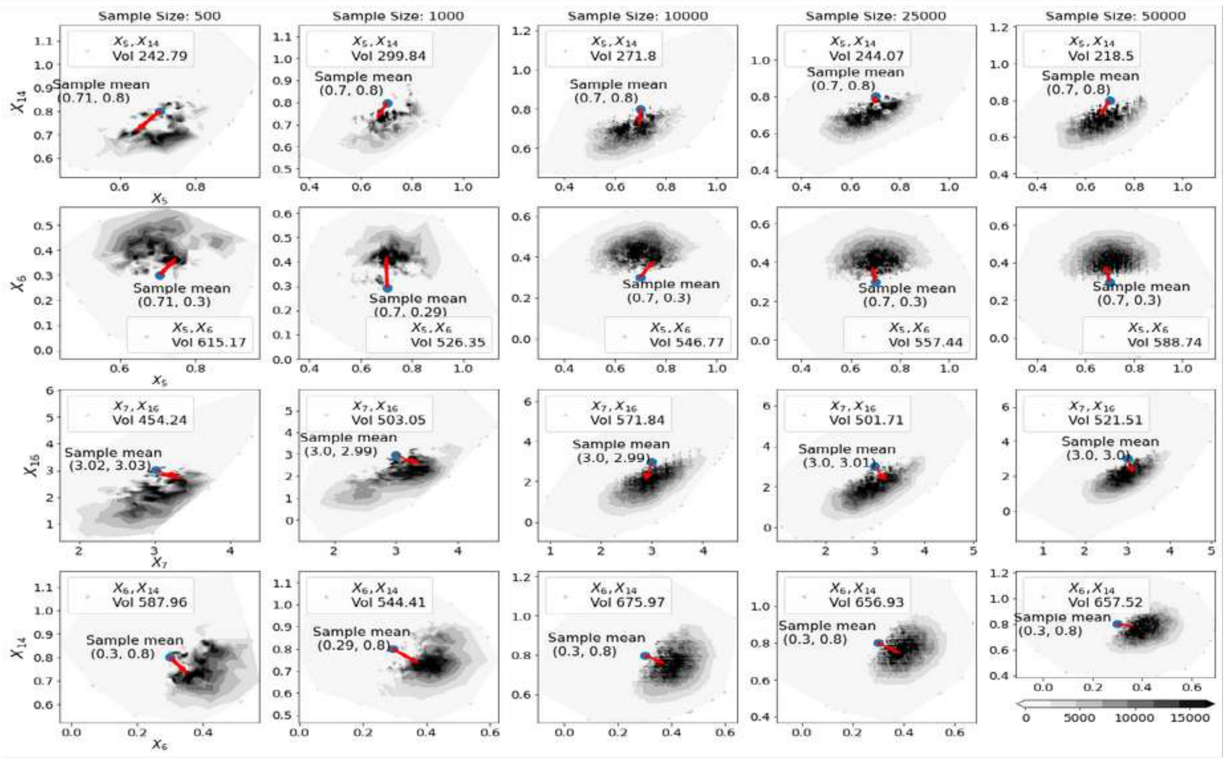


FIGURE 35. Result of information density estimation for different grups of variables by **Algorithm** 5 (with LassoLars weights)

Here we present the rest of the experiments performed for information density estimation for the second case study 6.2. Figures 31, 32, 33, 34 and 35 report information density estimates yielded by **Algrithm** 4 and **Algrithm** 5 for different groups of variables.

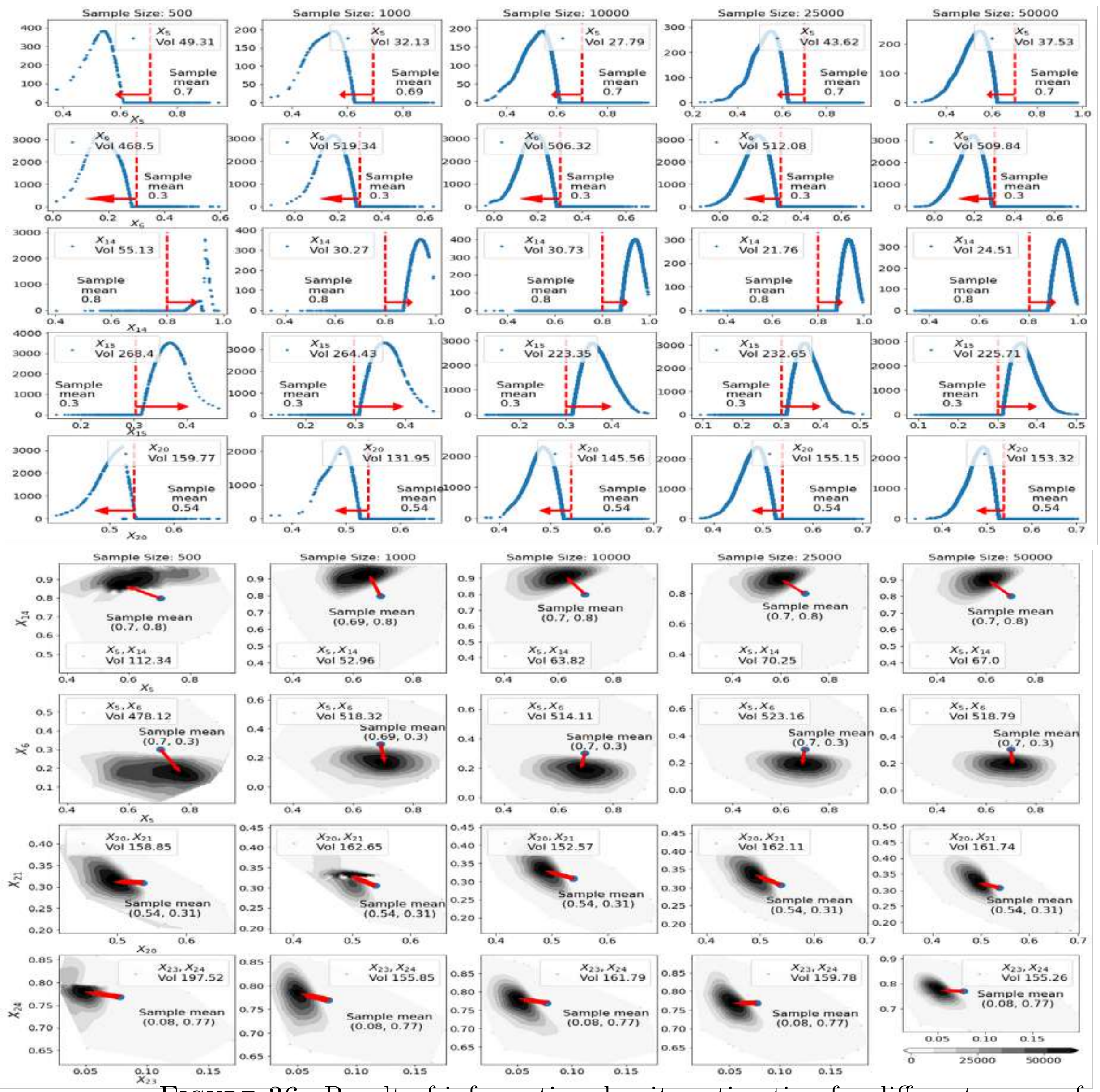


FIGURE 36. Result of information density estimation for different grups of variables by **Algorithm** 4 (MARS)

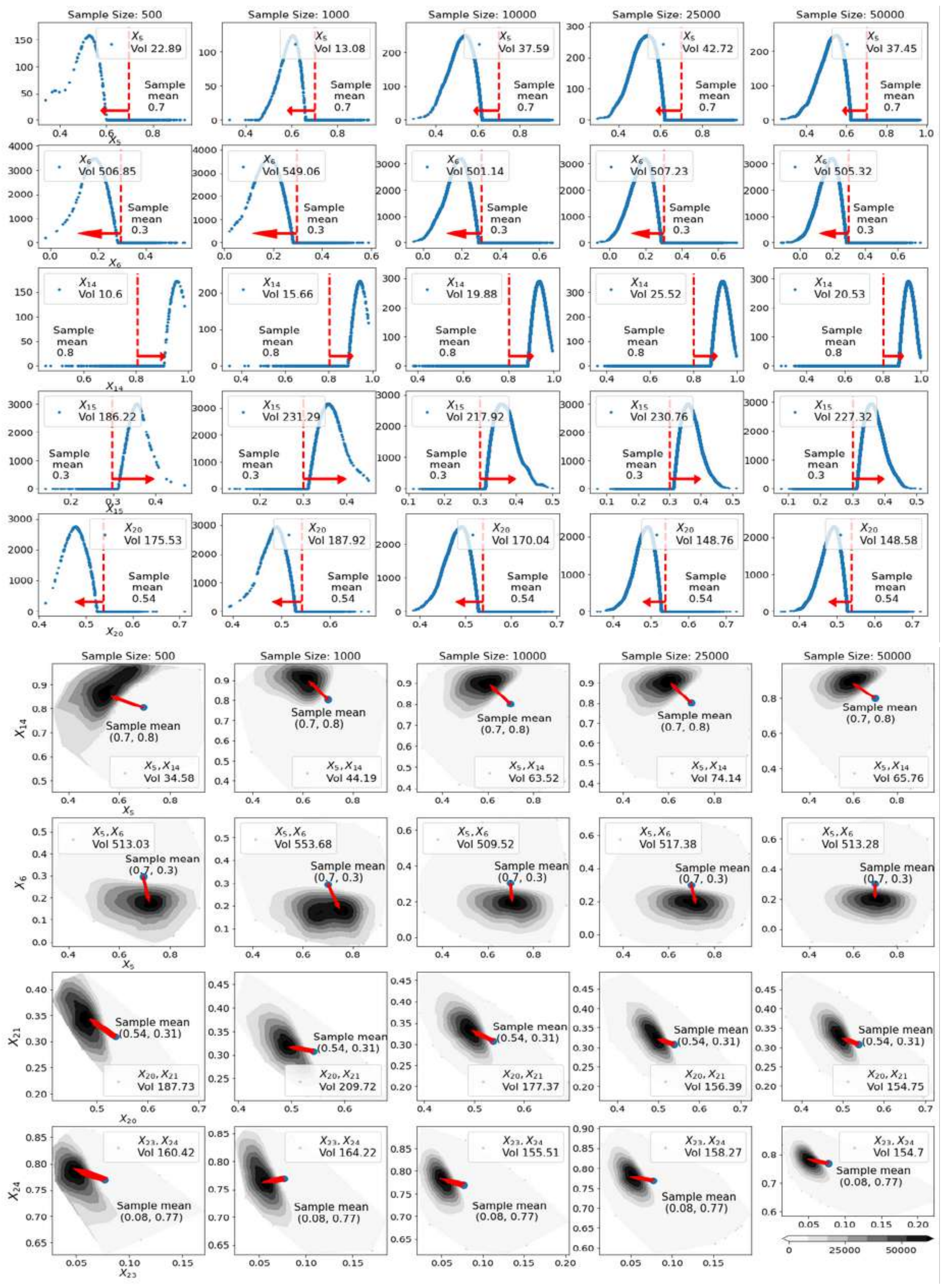


FIGURE 37. Result of information density estimation for different grups of variables by **Algorithm** 4 (GAM)

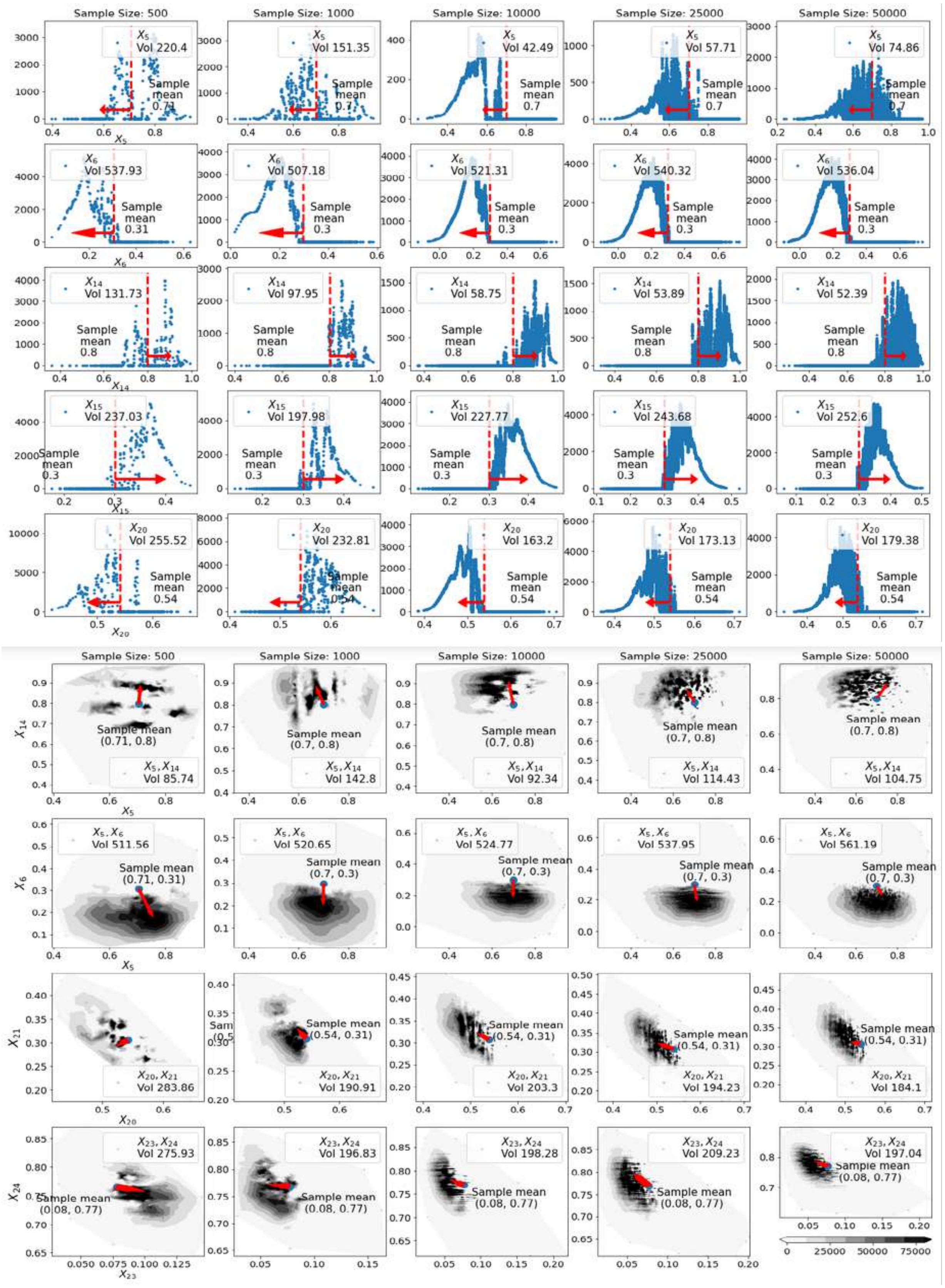


FIGURE 38. Result of information density estimation for different grups of variables by **Algorithm** 5 (without LassoLars weights)

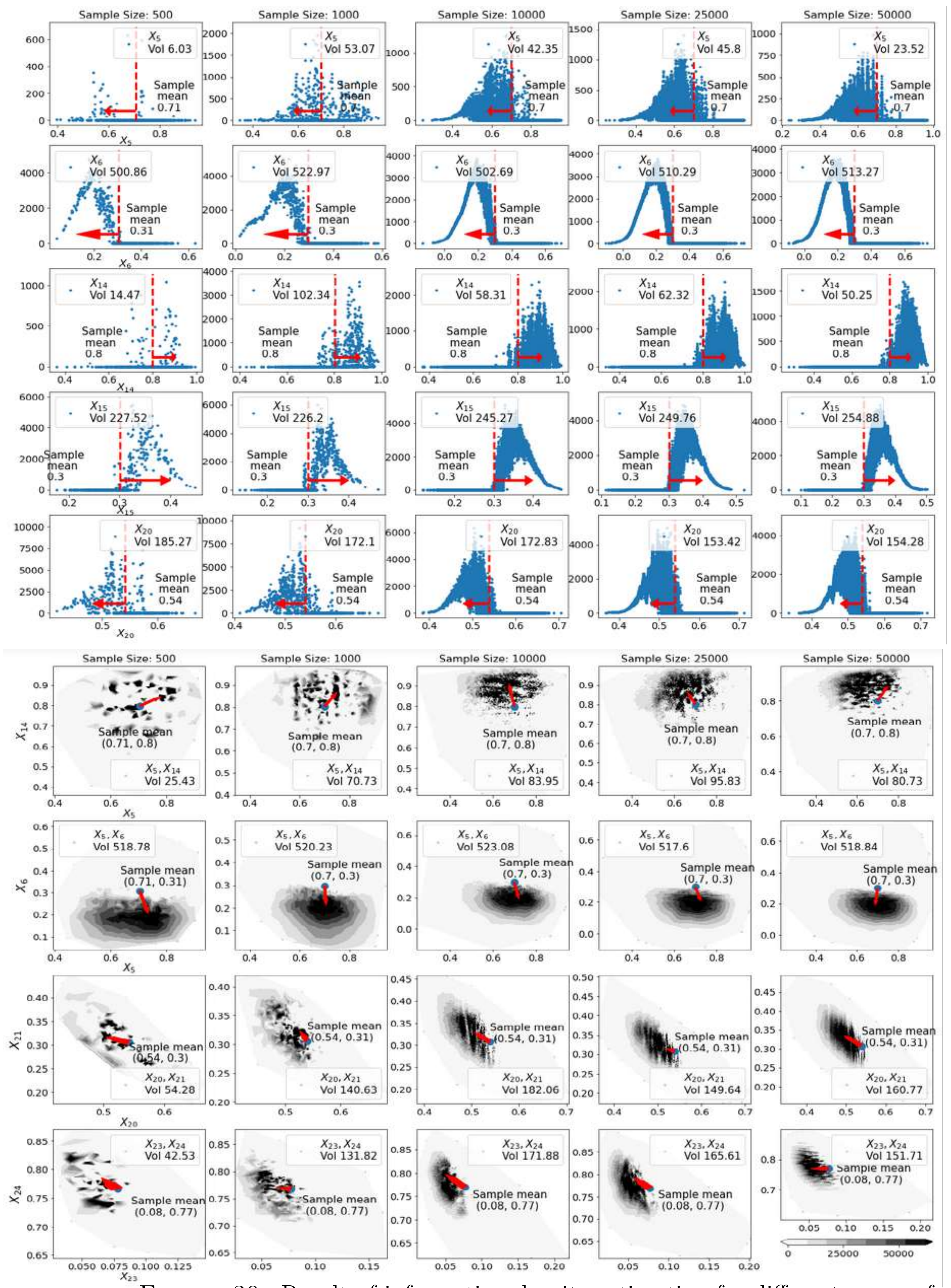


FIGURE 39. Result of information density estimation for different grups of variables by **Algorithm** 5 (with LassoLars weights)

Appendix C

Here we present the codes (Python 3.7) that implement the original algorithms for VoI estimation developed in this thesis.

[27]:
```
from __future__ import division
from __future__ import print_function
import re
from joblib import Parallel, delayed
from lightgbm import LGBMRegressor
from pandas import DataFrame
from rpy2.robjects import pandas2ri
from sklearn.linear_model import LassoLarsCV, LassoLarsIC
from sklearn.model_selection import train_test_split
from sklearn.neighbors import NearestNeighbors
from tqdm import tqdm

pandas2ri.activate()
from rpy2.robjects.packages import importr
from bayes_opt import BayesianOptimization
from time import time

import numpy as np
import time
import logging

import pandas
def experiment_knn_ordinar_val(inputs2, nb, alpha, k_max,␣
 ↪train_val_splits):
    X = inputs2 * np.sqrt(list(alpha))
    y = nb
    all_diffs = []
    for train_index, test_index in train_val_splits:
        X_train, X_test = X[train_index], X[test_index]
        y_train, y_test = y[train_index], y[test_index]
        clf = NearestNeighbors(n_neighbors=k_max).fit(X_train)
        # clf = FaissKNeighbors()
        # clf.fit(X_train, y_train)
        dist, indices = clf.kneighbors(X_test, k_max)
        predictions = np.array([np.average(y_train[indices[:, :k]][:, :,␣
 ↪0],
                                                   axis=1) for k in range(1,␣
 ↪k_max + 1)])

        difss = [np.abs(predictions[:, i] - y_test[i]) for i in␣
 ↪range(len(y_test))]

        all_diffs.append(difss)
    all_diffs_arr = np.vstack(all_diffs)
    return np.expand_dims(np.mean(all_diffs_arr, 0), 1)


def grid_search(nb, i, k, skf_list, indices_skf):
    fit_intercept = i[0]
    positive = i[1]
    norm = i[2]
    all_diffs = []
    for idx, (train_index, test_index) in tqdm(enumerate(skf_list)):
        y_train, y_test = nb[train_index], nb[test_index]
        skf2_train = np.random.choice(np.arange(len(indices_skf[0])),␣
 ↪size=int(len(indices_skf[0]) * 0.75),
                                      replace=False)
        skf2_test = np.array(list(set(np.arange(len(indices_skf[0]))).
 ↪difference(skf2_train)))

        indices = indices_skf[idx]
        estimator = LassoLarsCV(normalize=norm, n_jobs=-1, cv=5,␣
 ↪fit_intercept=fit_intercept, positive=positive)
        estimator.fit(y_train[indices[skf2_train, :k]],␣
 ↪y_test[skf2_train], )
        predictions = estimator.predict(y_train[indices[skf2_test, :k]])
        difss = np.abs(predictions - y_test[skf2_test])
        all_diffs.append(difss)

    all_diffs_arr = np.hstack(all_diffs)
    return np.mean(all_diffs_arr)


def objective(inputs,
              nb,
              k_max, train_val_splits,
              params, size_sample=10000):
    alphas = tuple([params[key]
                    for key in params.keys() if len(re.findall("alpha",␣
↪key)) > 0])

    # idx_under = np.random.choice(a=np.arange(len(inputs)),␣
↪size=size_sample, replace=False)
    difss = experiment_knn_ordinar_val(inputs2=inputs,
                                    nb=nb, alpha=alphas, k_max=k_max,␣
↪train_val_splits=train_val_splits)
    if difss.shape[1] == 2:
        loss = np.max([np.min(difss[:, 0]), np.min(difss[:, 1])])
    elif difss.shape[1] == 1:
        loss = np.min(difss[:, 0])
    return -loss


def inf_val_knn_lasso(nb, inputs, parameters_indx, k_max, k_bayes,␣
↪n_bayes_steps, n_bayes_sample, n_cv, init_points_bayes):
    start_t_knn=time.time()
    if n_bayes_sample == 'all':
        idx_bayes_search = np.arange(len(inputs))
    else:
        idx_bayes_search = np.random.choice(a=np.arange(len(inputs)),␣
↪replace=False, size=n_bayes_sample)

    inputs2 = inputs[idx_bayes_search, :][:, parameters_indx].
↪astype('float32')
    inputs2 = np.ascontiguousarray(inputs2)
    nb_bayes = nb[idx_bayes_search]
    train_val_splits = []
    for _ in range(n_cv):
        X_train, X_test, y_train, y_test =␣
↪train_test_split(DataFrame(inputs2), DataFrame(nb_bayes[:, 0]),
                                                    test_size=0.1)
        idx = X_train.index
        idx2 = X_test.index
        train_val_splits.append((idx, idx2))
    if len(parameters_indx) > 1:
        bounds = {
            'alpha' + str(i): (0.00001, 1)
            for i in range(0, inputs2.shape[1])
        }
        optimizer0 = BayesianOptimization(
            f=lambda **x: objective(inputs=inputs2,
                                    nb=nb_bayes[:, [0]],
                                    k_max=k_bayes,␣
↪train_val_splits=train_val_splits,
                                    params=x),
            pbounds=bounds,
            random_state=1,
        )
        optimizer0.maximize(init_points=init_points_bayes,␣
↪n_iter=n_bayes_steps)
        alpha_0 = optimizer0.max['params'].values()
        alpha_0 = tuple(alpha_0)
        optimizer1 = BayesianOptimization(
            f=lambda **x: objective(inputs=inputs2,
                                    nb=nb_bayes[:, [1]],
                                    k_max=k_bayes,␣
↪train_val_splits=train_val_splits,
                                    params=x),
            pbounds=bounds,
            random_state=1,
        )
        optimizer1.maximize(init_points=init_points_bayes,␣
↪n_iter=n_bayes_steps)
        alpha_1 = optimizer1.max['params'].values()
        alpha_1 = tuple(alpha_1)
    else:
        alpha_1 = [1]
        alpha_0 = [1]

    def find_neighbors(X, y, train_val_splits):
        dists = []
        indices_total = []
        for train_index, test_index in train_val_splits:
            X_train, X_test = X[train_index], X[test_index]
            clf = NearestNeighbors(n_neighbors=k_max).fit(X_train)
            dist, indices = clf.kneighbors(X_test)
            dists.append(dist)
            indices_total.append(indices)
        return dists, np.array(indices_total)
```

```
def find_k_cross_validation(nb, k_max, train_val_splits,␣
↪ind_train_val):
    y = nb
    all_diffs = []
    for idx, (train_index, test_index) in␣
↪tqdm(enumerate(train_val_splits)):
        y_train, y_test = y[train_index], y[test_index]
        indices = ind_train_val[idx]

        stds=np.array([np.std(y_train[indices[:, :k]][:, :, 0],
                             axis=1).mean() for k in range(1, k_max +␣
↪1)])
        pandas.Series(index=range(1, k_max + 1),data=stds).plot()

        predictions = np.array([np.average(y_train[indices[:, :k]][:,␣
↪:, 0],
                                          axis=1) for k in range(1,␣
↪k_max + 1)])
        predictions = np.expand_dims(predictions, 2)
        difss = np.mean(np.abs(predictions - y_test), 1)
        all_diffs.append(difss)
    return all_diffs

estimates_knn = []
estimates_knn_lassolars = []

X0 = inputs2 * np.sqrt(list(alpha_0))
X0 = X0[:, ~np.all(X0 == 0, axis=0)]
X1 = inputs2 * np.sqrt(list(alpha_1))
X1 = X1[:, ~np.all(X1 == 0, axis=0)]

y0 = nb_bayes[:, 0]
y1 = nb_bayes[:, 1]

dists0, ind_val_nn_in_train0 = find_neighbors(X0, y0,␣
↪train_val_splits)
dists1, ind_val_nn_in_train1 = find_neighbors(X1, y1,␣
↪train_val_splits)

exp_k0 = find_k_cross_validation(nb=nb_bayes[:, [0]],
                                 k_max=k_max,␣
↪train_val_splits=train_val_splits,
                                 ind_train_val=ind_val_nn_in_train0)
    exp_k1 = find_k_cross_validation(nb=nb_bayes[:, [1]],
                                     k_max=k_max,␣
↪train_val_splits=train_val_splits,
                                     ind_train_val=ind_val_nn_in_train1)

    k_0 = [1 + np.argmin(i[:, 0]) for i in exp_k0]
    k_1 = [1 + np.argmin(i[:, 0]) for i in exp_k1]
    # k_0 = [1 + np.argmin(np.concatenate(exp_k0, 1).mean(1))]
    # k_1 = [1 + np.argmin(np.concatenate(exp_k1, 1).mean(1))]
    print(f"K0 {k_0} and K1 {k_1}")

    inputs2 = inputs[:, parameters_indx].astype('float32')
    inputs2 = np.ascontiguousarray(inputs2)

    X0 = inputs2 * np.sqrt(list(alpha_0))
    X0 = X0[:, ~np.all(X0 == 0, axis=0)]
    X1 = inputs2 * np.sqrt(list(alpha_1))
    X1 = X1[:, ~np.all(X1 == 0, axis=0)]

    clf = NearestNeighbors(n_neighbors=max(k_0)).fit(X0)
    dist0, indices_total0 = clf.kneighbors(X0)
    clf = NearestNeighbors(n_neighbors=max(k_1)).fit(X1)
    dist1, indices_total1 = clf.kneighbors(X1)

    for i in range(len(k_0)):

        smooth0 = nb[indices_total0, 0][:, :k_0[i]].mean(1)
        smooth1 = nb[indices_total1, 1][:, :k_1[i]].mean(1)
        y_smooth = np.vstack((smooth0, smooth1)).T
        final_inf = np.mean(np.max(y_smooth, 1)) - np.max(np.
↪mean(y_smooth, 0))
        estimates_knn.append(final_inf)

        from sklearn.model_selection import GridSearchCV
        parameters = {'normalize': (True, False),
                      'fit_intercept': (True, False)}

        clf = GridSearchCV(LassoLarsIC(), parameters, verbose=4, cv=2,␣
↪scoring='neg_mean_absolute_error')
        clf.fit(nb[indices_total0, 0][:, 1:k_0[i]], nb[:, 0])
        smooth0 = clf.predict(nb[indices_total0, 0][:, 1:k_0[i]])
        clf2 = GridSearchCV(LassoLarsIC(), parameters, verbose=4, cv=2,␣
↪scoring='neg_mean_absolute_error')
        clf2.fit(nb[indices_total1, 1][:, 1:k_1[i]], nb[:, 1])
        smooth1 = clf2.predict(nb[indices_total1, 1][:, 1:k_1[i]])
        y_smooth = np.vstack((smooth0, smooth1)).T
        final_inf = np.mean(np.max(y_smooth, 1)) - np.max(np.
↪mean(y_smooth, 0))
        estimates_knn_lassolars.append(final_inf)

    return np.mean(estimates_knn), np.mean(estimates_knn_lassolars)
```

[28]:
```
import pyearth
def alg2MARS(nb,inputs,parameters_indx):
    '''Algorithm 2 (MARS)'''

    x=inputs[:,parameters_indx]
    y1=nb[:,[0]]
    y2=nb[:,[1]]
    model1=pyearth.Earth()
    model1.fit(x,y1)
    pred1=model1.predict(x)

    model2=pyearth.Earth()
    model2.fit(x,y2)
    pred2=model2.predict(x)

    nb_smooth=np.vstack((pred1,pred2)).T
    inf_val=nb_smooth.max(1).mean()-nb_smooth.mean(0).max()
    return inf_val
```

[ ]:

Appendix D

Here we present the codes (Python 3.7) that implement the original algorithms for information density estimation developed in this thesis.

```
[39]: from __future__ import division
      from __future__ import print_function
      import re
      from pandas import DataFrame
      from rpy2.robjects import pandas2ri
      from sklearn.linear_model import LassoLarsCV, LassoLarsIC
      from sklearn.model_selection import train_test_split
      from sklearn.neighbors import NearestNeighbors
      from statsmodels.nonparametric.kernel_density import KDEMultivariate
      from tqdm import tqdm
      import pandas as pd
      pandas2ri.activate()
      from bayes_opt import BayesianOptimization
      import numpy as np
      import time


      def experiment_knn_ordinar_val(inputs2, nb, alpha, k_max,␣
       ↪train_val_splits):
          X = inputs2 * np.sqrt(list(alpha))
          y = nb
          all_diffs = []
          for train_index, test_index in train_val_splits:
              X_train, X_test = X[train_index], X[test_index]
              y_train, y_test = y[train_index], y[test_index]
              clf = NearestNeighbors(n_neighbors=k_max).fit(X_train)
              # clf = FaissKNeighbors()
              # clf.fit(X_train, y_train)
              dist, indices = clf.kneighbors(X_test, k_max)
              predictions = np.array([np.average(y_train[indices[:, :k]][:, :,␣
       ↪0],
                                                          axis=1) for k in range(1,␣
      ↪k_max + 1)])

              difss = [np.abs(predictions[:, i] - y_test[i]) for i in␣
      ↪range(len(y_test))]

              all_diffs.append(difss)
          all_diffs_arr = np.vstack(all_diffs)
          return np.expand_dims(np.mean(all_diffs_arr, 0), 1)

      def objective(inputs,
                    nb,
                    k_max, train_val_splits,
                    params, size_sample=10000):
          alphas = tuple([params[key]
                          for key in params.keys() if len(re.findall("alpha",␣
      ↪key)) > 0])

          difss = experiment_knn_ordinar_val(inputs2=inputs,
                                             nb=nb, alpha=alphas, k_max=k_max,␣
      ↪train_val_splits=train_val_splits)
          if difss.shape[1] == 2:
              loss = np.max([np.min(difss[:, 0]), np.min(difss[:, 1])])
          elif difss.shape[1] == 1:
              loss = np.min(difss[:, 0])
          return -loss


      def inf_density_knn(nb, inputs, parameters_indx, k_max, k_bayes,␣
      ↪n_bayes_steps, n_bayes_sample, n_cv, init_points_bayes):
          '''Algorithm 5'''
          start_t_knn = time.time()
          if n_bayes_sample == 'all':
              idx_bayes_search = np.arange(len(inputs))
          else:
              idx_bayes_search = np.random.choice(a=np.arange(len(inputs)),␣
      ↪replace=False, size=n_bayes_sample)

          inputs2 = inputs[idx_bayes_search, :][:, parameters_indx].
      ↪astype('float32')
          inputs2 = np.ascontiguousarray(inputs2)
          nb_bayes = nb[idx_bayes_search]
          train_val_splits = []
          for _ in range(n_cv):
              X_train, X_test, y_train, y_test =␣
      ↪train_test_split(DataFrame(inputs2), DataFrame(nb_bayes[:, 0]),
                                                                        test_size=0.1)
              idx = X_train.index
              idx2 = X_test.index
              train_val_splits.append((idx, idx2))
          if len(parameters_indx) > 1:

              start = time.time()
              print(f"Start time: {start}")
              # Search for direction --- feature selection
              bounds = {
                  'alpha' + str(i): (0.00001, 1)
                  for i in range(0, inputs2.shape[1])
              }
              try:
                  optimizer0 = BayesianOptimization(
                      f=lambda **x: objective(inputs=inputs2,
                                              nb=nb_bayes[:, [0]],
                                              k_max=k_bayes,␣
      ↪train_val_splits=train_val_splits,
                                              params=x),
                      pbounds=bounds,
                      random_state=1,
                  )
                  optimizer0.maximize(init_points=init_points_bayes,␣
      ↪n_iter=n_bayes_steps)
                  alpha_0 = optimizer0.max['params'].values()
                  alpha_0 = tuple(alpha_0)
              except ValueError:
                  alpha_0=(1,1)
              optimizer1 = BayesianOptimization(
                  f=lambda **x: objective(inputs=inputs2,
                                          nb=nb_bayes[:, [1]],
                                          k_max=k_bayes,␣
      ↪train_val_splits=train_val_splits,
                                          params=x),
                  pbounds=bounds,
                  random_state=1,
              )
              optimizer1.maximize(init_points=init_points_bayes,␣
      ↪n_iter=n_bayes_steps)
              alpha_1 = optimizer1.max['params'].values()
              alpha_1 = tuple(alpha_1)
              end = time.time()
              print(f"Time for metric learning : {end - start}")
          else:
              alpha_1 = [1]
              alpha_0 = [1]

          def find_neighbors(X, y, train_val_splits):
              dists = []
              indices_total = []
              for train_index, test_index in train_val_splits:
                  X_train, X_test = X[train_index], X[test_index]
                  y_train, y_test = y[train_index], y[test_index]
                  clf = NearestNeighbors(n_neighbors=k_max).fit(X_train)
                  dist, indices = clf.kneighbors(X_test)
                  dists.append(dist)
                  indices_total.append(indices)
              return dists, np.array(indices_total)

          def find_k_cross_validation(nb, k_max, train_val_splits,␣
      ↪ind_train_val):
              y = nb
              all_diffs = []
              for idx, (train_index, test_index) in␣
      ↪tqdm(enumerate(train_val_splits)):
                  y_train, y_test = y[train_index], y[test_index]
                  indices = ind_train_val[idx]
                  predictions = np.array([np.average(y_train[indices[:, 1:k]][:
      ↪, :, 0],
                                          axis=1) for k in range(2,␣
      ↪k_max + 1)])
                  predictions = np.expand_dims(predictions, 2)
                  difss = np.mean(np.abs(predictions - y_test), 1)
                  all_diffs.append(difss)
              return all_diffs

          estimates = []
          estimates_knn = []

          print(f"ALPHA1 {alpha_0} and alpha2 {alpha_1}")
          X0 = inputs2 * np.sqrt(list(alpha_0))
          X0 = X0[:, ~np.all(X0 == 0, axis=0)]
          X1 = inputs2 * np.sqrt(list(alpha_1))
          X1 = X1[:, ~np.all(X1 == 0, axis=0)]
          y0 = nb_bayes[:, 0]
          y1 = nb_bayes[:, 1]

          dists0, ind_val_nn_in_train0 = find_neighbors(X0, y0,␣
      ↪train_val_splits)
          dists1, ind_val_nn_in_train1 = find_neighbors(X1, y1,␣
      ↪train_val_splits)

          exp_k0 = find_k_cross_validation(nb=nb_bayes[:, [0]],
                                           k_max=k_max,␣
      ↪train_val_splits=train_val_splits,
                                           ind_train_val=ind_val_nn_in_train0)

          exp_k1 = find_k_cross_validation(nb=nb_bayes[:, [1]],
                                           k_max=k_max,␣
      ↪train_val_splits=train_val_splits,
                                           ind_train_val=ind_val_nn_in_train1)

          k_0 = [2 + np.argmin(i[:, 0]) for i in exp_k0]
          k_1 = [2 + np.argmin(i[:, 0]) for i in exp_k1]

          print(f"K0 {k_0} and K1 {k_1}")

          inputs2 = inputs[:, parameters_indx].astype('float32')
          inputs2 = np.ascontiguousarray(inputs2)

          X0 = inputs2 * np.sqrt(list(alpha_0))
          X0 = X0[:, ~np.all(X0 == 0, axis=0)]
          X1 = inputs2 * np.sqrt(list(alpha_1))
          X1 = X1[:, ~np.all(X1 == 0, axis=0)]

          clf = NearestNeighbors(n_neighbors=max(k_0)+1).fit(X0)
          dist0, indices_total0 = clf.kneighbors(X0)
          clf = NearestNeighbors(n_neighbors=max(k_1)+1).fit(X1)
          dist1, indices_total1 = clf.kneighbors(X1)

          inf_val_den_lasso=[]
          inf_val_den=[]
          for i in range(len(k_0)):
              smooth0 = nb[indices_total0, 0][:, :k_0[i]-1].mean(1)
              smooth1 = nb[indices_total1, 1][:, :k_1[i]-1].mean(1)
              y_smooth = np.vstack((smooth0, smooth1)).T
              final_inf = np.mean(np.max(y_smooth, 1)) - np.max(np.
      ↪mean(y_smooth, 0))
              estimates_knn.append(final_inf)
              inf_val_den.append(y_smooth)
              end_t_knn = time.time()
              print(f"KNN {final_inf}", end_t_knn - start_t_knn)
              from sklearn.model_selection import GridSearchCV
              parameters = {'normalize': ( False,True),
                            'fit_intercept': ( False,True)}
              clf = GridSearchCV(LassoLarsCV(), parameters, verbose=4, cv=5,␣
      ↪scoring='neg_mean_squared_error')
              clf.fit(nb[indices_total0, 0][:, 1:k_0[i]], nb[:, 0])
              smooth0 = clf.predict(nb[indices_total0, 0][:, 1:k_0[i]])
              clf2 = GridSearchCV(LassoLarsCV(), parameters, verbose=4, cv=5,␣
      ↪scoring='neg_mean_squared_error')
              clf2.fit(nb[indices_total1, 1][:, 1:k_1[i]], nb[:, 1])
              smooth1 = clf2.predict(nb[indices_total1, 1][:, 1:k_1[i]])

              y_smooth_lasso_lars = np.vstack((smooth0, smooth1)).T
              final_inf_lasso_lars = np.mean(np.max(y_smooth_lasso_lars, 1)) -␣
      ↪np.max(np.mean(y_smooth_lasso_lars, 0))
              inf_val_den_lasso.append(y_smooth_lasso_lars)


          nb_smooth_lasso=np.vstack((np.concatenate([ j[:,[0]]for j in␣
      ↪inf_val_den_lasso ],1).mean(1),
          np.concatenate([ j[:,[1]]for j in inf_val_den_lasso ],1).max(1))).T

          nb_smooth_knn=np.vstack((np.concatenate([ j[:,[0]]for j in␣
      ↪inf_val_den ],1).mean(1),
          np.concatenate([ j[:,[1]]for j in inf_val_den ],1).max(1))).T


          idx_max=nb_smooth_lasso.mean(0).argmax()
          inf_val_density0=nb_smooth_lasso.max(1)-nb_smooth_lasso[:,idx_max]

          idx_max=nb_smooth_knn.mean(0).argmax()
          inf_val_density1=nb_smooth_knn.max(1)-nb_smooth_knn[:,idx_max]

          x=inputs[:,parameters_indx]
          clf=KDEMultivariate(data=x,var_type='c'*len(parameters_indx))
          x_density=clf.pdf(x)

          inf_val_density=inf_val_density0*x_density
          inf_val_density_prim=inf_val_density1*x_density


          inf_val_density_result_lasso_pd=pd.DataFrame(data=np.
      ↪hstack((x,x_density.reshape((-1,1)),
                                                          inf_val_density0.
      ↪reshape((-1,1)),
                                                          inf_val_density.
      ↪reshape((-1,1)))),
              columns=[f"x{i}" for i in range(x.
      ↪shape[1])]+['x_density','inf_val_gain_x','information_density'])

          inf_val_density_result_knn_pd=pd.DataFrame(data=np.
      ↪hstack((x,x_density.reshape((-1,1)),
                                                          inf_val_density1.
      ↪reshape((-1,1)),
                                                          inf_val_density_prim.
      ↪reshape((-1,1)))),
              columns=[f"x{i}" for i in range(x.
      ↪shape[1])]+['x_density','inf_val_gain_x','information_density'])

          return inf_val_density_result_lasso_pd,inf_val_density_result_knn_pd
```

[4]:
```
import  pyearth
def alg4MARS(nb,inputs,parameters_indx):
    '''Algorithm 4 (MARS)'''

    x=inputs[:,parameters_indx]
    y1=nb[:,[0]]
    y2=nb[:,[1]]
    model1=pyearth.Earth()
    model1.fit(x,y1)
    pred1=model1.predict(x)

    model2=pyearth.Earth()
    model2.fit(x,y2)
    pred2=model2.predict(x)

    nb_smooth=np.vstack((pred1,pred2)).T
    arg_max_total=nb_smooth.mean(0).argmax()
    inf_val_density0=nb_smooth.max(1)-nb_smooth[:,arg_max_total]
    clf=KDEMultivariate(data=x,var_type='c'*len(parameters_indx))
    x_density=clf.pdf(x)

    inf_val_density=inf_val_density0*x_density

    inf_val_density_result_pd=pd.DataFrame(data=np.hstack((x,x_density.
↪reshape((-1,1)),
                                                         ␣
↪inf_val_density0.reshape((-1,1)),
                                                         ␣
↪inf_val_density.reshape((-1,1)))),
                columns=[f"x{i}" for i in range(x.
↪shape[1])]+['x_density','inf_val_gain_x','information_density'])


    return inf_val_density_result_pd
```