Scalable Multivariate Histograms

Raazesh Sainudiin[1,2][0000−0003−3265−5565] and Tilo Wiklund[1][0000−0002−5465−999]

[1] Department of Mathematics, Uppsala University, Uppsala, Sweden
[2] AI & Analytics Centre of Excellence, Combient AB, Stockholm, Sweden
`raazesh.sainudiin@math.uu.se` `http://lamastex.org`
[3] `tilo.wiklund@math.uu.se`

Abstract. We give a distributed variant of an adaptive histogram estimation procedure previously developed by the first author. The procedure is based on regular pavings and is known to have numerous appealing statistical and arithmetical properties. The distributed version makes it possible to process data sets significantly bigger than previously. We provide prototype implementation under a permissive license.

Keywords: density estimation · penalized likelihood · statistical regular paving · multivariate histogram trees · Apache Spark · MapReduce

1 Introduction

Suppose our random variable X has an unknown density f on $\mathbb{R}^d$, then for all Borel sets $A \subseteq \mathbb{R}^d$,

$$\mu(A) := \Pr\{X \in A\} = \int_A f(x)dx \ .$$

Any density estimate $f_n(x) := f_n(x; X_1, X_2, \ldots, X_n) : \mathbb{R}^d \times \left(\mathbb{R}^d\right)^n \to \mathbb{R}$ is a map from $\left(\mathbb{R}^d\right)^{n+1}$ to $\mathbb{R}$. The objective in density estimation is to estimate the unknown f from an independent and identically distributed (IID) sample $X_1, X_2, \ldots, X_n$ drawn from f. Density estimation is often the first step in many learning tasks, including, anomaly detection, classification, regression and clustering. Current density estimators do not computationally cope with large datasets. We use a MapReduce implementation of an optimally smoothed multivariate density estimator over a dense class tree-based data-adaptive partitions.

2 Statistical regular pavings and histograms

Let $\boldsymbol{x} := [\underline{x}, \overline{x}]$ be a compact real interval with lower bound $\underline{x}$ and upper bound $\overline{x}$, where $\underline{x} \le \overline{x}$. Let the space of such intervals be $\mathbb{IR}$. The width of an interval $\boldsymbol{x}$ is $\mathrm{wid}\,(\boldsymbol{x}) := \overline{x} - \underline{x}$. The midpoint is $\mathrm{mid}\,(\boldsymbol{x}) := (\underline{x} + \overline{x})/2$. A box of dimension d with coordinates in $\Delta := \{1, 2, \ldots, d\}$ is an interval vector with ι as the first coordinate of maximum width:

$$\boldsymbol{x} := [\underline{x}_1, \overline{x}_1] \times \ldots \times [\underline{x}_d, \overline{x}_d] =: \underset{j \in \Delta}{\otimes} [\underline{x}_j, \overline{x}_j], \quad \iota := \min\left(\underset{i}{\mathrm{argmax}}(\mathrm{wid}\,(\boldsymbol{x}_i))\right) \ .$$

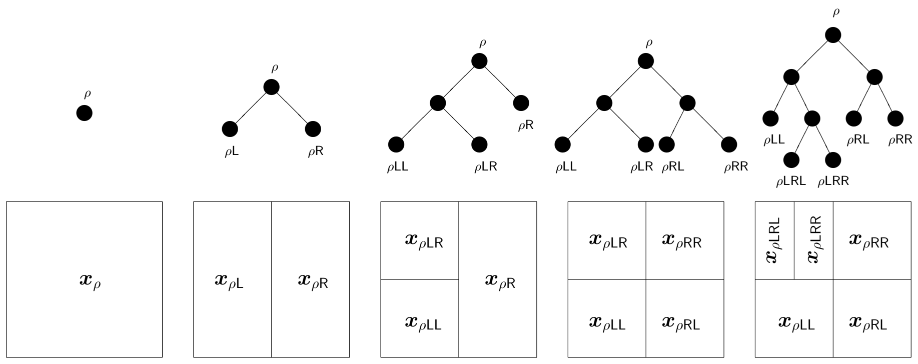


Fig. 1. A sequence of selective bisections of boxes (nodes) along the first widest coordinate, starting from the root box (root node) in two dimensions, produces an RP.

The set of all such boxes is $\mathbb{IR}^d$, i.e., the set of all interval real vectors in dimension d. A *bisection* or *split* of $\boldsymbol{x}$ perpendicularly at the mid-point along this first widest coordinate ι gives the left and right child boxes of $\boldsymbol{x}$

$$\boldsymbol{x}_{\mathsf{L}} := [\underline{x}_1, \overline{x}_1] \times \ldots \times [\underline{x}_\iota, \mathrm{mid}(\boldsymbol{x}_\iota)) \times [\underline{x}_{\iota+1}, \overline{x}_{\iota+1}] \times \ldots \times [\underline{x}_d, \overline{x}_d] \ ,$$

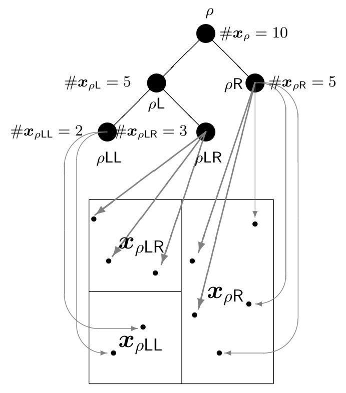
$$\boldsymbol{x}_{\mathsf{R}} := [\underline{x}_1, \overline{x}_1] \times \ldots \times [\mathrm{mid}(\boldsymbol{x}_\iota), \overline{x}_\iota] \times [\underline{x}_{\iota+1}, \overline{x}_{\iota+1}] \times \ldots \times [\underline{x}_d, \overline{x}_d] \ .$$

Such a bisection is said to be *regular*. Note that this bisection gives the left child box a half-open interval $[\underline{x}_\iota, \mathrm{mid}(\boldsymbol{x}_\iota))$ on coordinate ι so that the intersection of the left and right child boxes is empty. A recursive sequence of selective regular bisections of boxes, with possibly open boundaries, along the first widest coordinate, starting from the root box $\boldsymbol{x}_\rho$ in $\mathbb{IR}^d$ is known as a *regular paving* [5] or n-tree [12] of $\boldsymbol{x}_\rho$. A regular paving of $\boldsymbol{x}_\rho$ can also be seen as a binary tree formed by recursively bisecting the box $\boldsymbol{x}_\rho$ at the root node. Each node in the binary tree has either no children or two children. These trees are known as plane binary trees in enumerative combinatorics [13, Ex. 6.19(d), p. 220] and as finite, rooted binary trees (frb-trees) in geometric group theory [9, Chap. 10]. The relationship of trees, labels and partitions is illustrated in Figure 1 via a sequence of bisections of a square (2-dimensional) root box by always bisecting on the *first* widest coordinate.

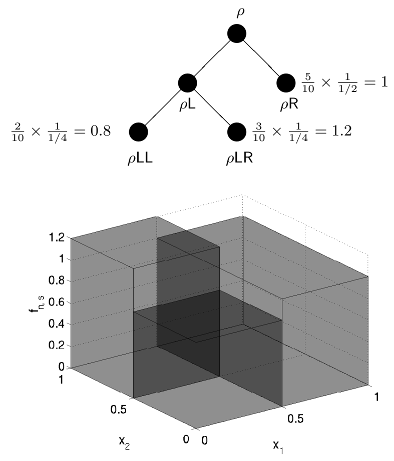
Let $\mathbb{N} := \{1, 2, \ldots\}$ be the set of natural numbers. Let the j-th interval of a box $\boldsymbol{x}_{\rho\mathsf{v}}$ be $[\underline{x}_{\rho\mathsf{v},j}, \overline{x}_{\rho\mathsf{v},j}]$, the volume of a d-dimensional box $\boldsymbol{x}_{\rho\mathsf{v}}$ be $\mathrm{vol}(\boldsymbol{x}_{\rho\mathsf{v}}) = \prod_{j=1}^{d}(\overline{x}_{\rho\mathsf{v},j} - \underline{x}_{\rho\mathsf{v},j})$, the set of all nodes of an RP be $\mathbb{V} := \rho \cup \{\rho\{\mathsf{L},\mathsf{R}\}^j : j \in \mathbb{N}\}$, the set of all leaf nodes be $\mathbb{L}$ and the set of internal nodes or splits be $\breve{\mathbb{V}}(s) := \mathbb{V}(s) \setminus \mathbb{L}(s)$. The set of leaf boxes of a regular paving s with root box $\boldsymbol{x}_\rho$ is denoted by $\boldsymbol{x}_{\mathbb{L}(s)}$ and it specifies a partition of the root box $\boldsymbol{x}_\rho$. Let $\mathbb{S}_k$ be the set of all regular pavings with root box $\boldsymbol{x}_\rho$ made of k splits. Note that the number of leaf nodes $m = |\mathbb{L}(s)| = k + 1$ if $s \in \mathbb{S}_k$. The number of distinct binary trees with k splits is equal to the Catalan number C_k. For $i, j \in \mathbb{Z}_+$, where $\mathbb{Z}_+ := \{0, 1, 2, \ldots\}$ and $i \leq j$, let $\mathbb{S}_{i:j} := \cup_{k=i}^{j} \mathbb{S}_k$ be the set of regular pavings with k splits where $k \in \{i, i+1, \ldots, j\}$. Let the set of all regular pavings be $\mathbb{S}_{0:\infty} := \lim_{j \to \infty} \mathbb{S}_{0:j}$.

A *statistical regular paving* (SRP) denoted by s is an extension of the RP structure that is able to act as a partitioned 'container' and responsive summarizer for multivariate data. An SRP can be used to create a histogram of a data set. A recursively computable statistic [1, 3] that an SRP node $\rho\mathsf{v}$ caches is $\#\boldsymbol{x}_{\rho\mathsf{v}}$, the count of the number of data points that fell into $\boldsymbol{x}_{\rho\mathsf{v}}$. A leaf node $\rho\mathsf{v}$ with $\#\boldsymbol{x}_{\rho\mathsf{v}} > 0$ is a non-empty leaf node. The set of non-empty leaves of an SRP s is $\mathbb{L}^+(s) := \{\rho\mathsf{v} \in \mathbb{L}(s) : \#\boldsymbol{x}_{\rho\mathsf{v}} > 0\} \subseteq \mathbb{L}(s)$.

Figure 2 depicts a small SRP s with root box $\boldsymbol{x}_\rho \in \mathbb{IR}^2$. The number of sample data points in the root box $\boldsymbol{x}_\rho$ is 10. Figure 2(a) shows the tree, including the count associated with each node in the tree and the partition of the root box represented by the leaf boxes of this tree, with the sample data points superimposed on the boxes. Figure 2(b) shows how the density estimate is computed from the count and the volume of leaf boxes to obtain the density estimate $f_{n,s}$ as an SRP histogram.



(a) An SRP tree and its constituents.



(b) An SRP histogram $f_{n,s}$.

Fig. 2. An SRP and its corresponding histogram.

An SRP histogram is obtained from n data points that fell into $\boldsymbol{x}_\rho$ of SRP s as follows:

$$f_{n,s}(x) = f_n(x) = \sum_{\rho\mathsf{v} \in \mathbb{L}(s)} \frac{\mathbb{1}_{\boldsymbol{x}_{\rho\mathsf{v}}}(x)}{n} \left(\frac{\#\boldsymbol{x}_{\rho\mathsf{v}}}{\operatorname{vol}(\boldsymbol{x}_{\rho\mathsf{v}})} \right) . \tag{1}$$

It is the maximum likelihood estimator over the class of simple (piecewise-constant) functions given the partition $\boldsymbol{x}_{\mathbb{L}(s)}$ of the root box of s. We suppress subscripting the histogram by the SRP s for notational convenience. SRP histograms have some similarities to dyadic histograms (for eg. [6, chap. 18], [7]). Both are binary tree-based and partition so that a box may only be bisected at the mid-point of one of its coordinates, but the RP structure restricts partitioning further by only bisecting a box on its first widest coordinate in order to make $\mathbb{S}_{0:\infty}$ closed under addition and scalar multiplication and thereby allowing for computationally efficient computer arithmetic over a dense set of simple functions. See [4] for statistical applications of this computer tree arithmetic, including conditional density regression, $1-\alpha$ confidence sets and associated tail probabilities, and model averaging across multivariate histograms with different partitions.

The set of regularly paved real-valued simple functions on $\boldsymbol{x}_\rho \in \mathbb{IR}^d$ satisfies the conditions of a Stone-Weierstrass theorem and is therefore dense in $\mathcal{C}(\boldsymbol{x}_\rho, \mathbb{R})$, the algebra of real-valued continuous functions over $\boldsymbol{x}_\rho$ [4, Theorem 4.1]. This ensures that histograms obtained from statistical regular pavings subject to an asymptotically L_1-consistent partitioning strategy can uniformly approximate any continuous density $f : \boldsymbol{x}_\rho \to \mathbb{R}$. Note that $\mathbb{S}_{0:\infty}$, the set of RP tree partitions, contains the set of dyadic partitions of $\boldsymbol{x}_\rho$ represented by complete dyadic binary trees: $\{\mathbb{S}_0, \mathbb{S}_2, \mathbb{S}_4, \mathbb{S}_8, \ldots\} \subset \mathbb{S}_{0:\infty}$. Thus $\mathbb{S}_{0:\infty}$ can be used in principle to obtain any σ-additive measure using standard measure-theoretic constructions (but such constructions without efficiency considerations may be limited in practice due to finite machine memory and computing time).

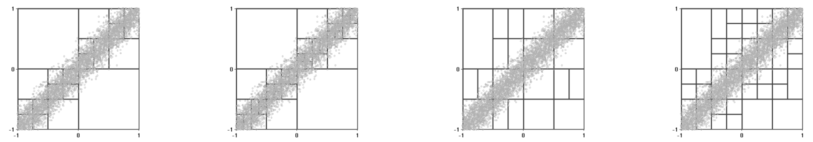
Priority-queued Markov chains The pseudo-code to generate sample paths from a generic priority-queued Markov chain (PQMC) over the state space of statistical regular pavings is given in Algorithm 1. A leaf node of a statistical regular paving (SRP) is splittable if it contains data and the child nodes from the split can be represented in the computer (as described in the next paragraph). A PQMC is a Markov chain on SRPs whose transition probabilities are given by ordering the elements of $\mathbb{L}^{\triangledown}(s)$, the splittable leaf nodes of the current SRP state s, by a randomized queue that is prioritized according to a given priority function $\psi : \mathbb{L}^{\triangledown}(s) \to \mathbb{R}$. This priority-queued collection of splittable leaf nodes is used to select the next node to be split from $\operatorname{argmax}_{\rho\mathsf{v} \in \mathbb{L}^{\triangledown}(s)} \psi(\rho\mathsf{v})$, the set of splittable leaf nodes of s which are equally 'large' when measured using ψ. If there is more than one such 'largest' node the choice is made uniformly at random from this set; this is the 'randomized' aspect of the process. Three criteria can be specified to stop the PQMC. A straightforward stopping condition is to stop partitioning when the number of leaves in the SRP reaches a specified maximum $\overline{m}$. The other stopping condition relates to the priority function so that partitioning stops when the value of the largest node under the priority function ψ is less than or equal to a specified value $\overline{\psi}$. A PQMC will also stop partitioning if there are no splittable leaf nodes in the SRP.

The output of $\texttt{PQMC}(s, \psi, \overline{\psi}, \overline{m})$ algorithm is $[s(0), s(1), \ldots, s(T)]$, a sequence of SRP states giving a sample path from the PQMC $\{S(t)\}_{t \in \mathbb{Z}_+}$ on $\mathbb{S}_{0:\overline{m}-1}$, such that $\mathbb{L}^{\triangledown}(s(T)) = \emptyset$ or $\psi(\rho\mathsf{v}) \leq \overline{\psi}\ \forall \rho\mathsf{v} \in \mathbb{L}^{\triangledown}(s(T))$ or $|\mathbb{L}(s(T))| \leq \overline{m}$ and T is a

```
ALGORITHM 1: PQMC($s, \psi, \overline{\psi}, \overline{m}$)
input      : $s$, initial SRP with root box $\boldsymbol{x}_\rho$,
             $x = (x_1, x_2, \dots, x_n)$, a data burst of size $n$,
             $\psi : \mathbb{L}^{\triangledown}(s) \to \mathbb{R}$, a priority function,
             $\overline{\psi}$, maximum value of $\psi(\rho\mathsf{v}) \in \mathbb{L}^{\triangledown}(s)$ for any splittable leaf node in the
final SRP,
             $\overline{m}$, maximum number of leaves in the final SRP.
output     : a sequence of SRP states $[s(0), s(1), \dots, s(T)]$ such that $\mathbb{L}^{\triangledown}(s(T)) = \emptyset$ or
             $\psi(\rho\mathsf{v}) \leq \overline{\psi}\ \forall \rho\mathsf{v} \in \mathbb{L}^{\triangledown}(s(T))$ or $|\mathbb{L}(s(T))| \leq \overline{m}$ .
initialize: $\boldsymbol{x}_\rho \leftsquigarrow x$, make $\boldsymbol{x}_\rho$ such that $\cup_i^n x_i \subset \boldsymbol{x}_\rho$ if $\nexists$ domain knowledge or
             historical data,
             $s \leftsquigarrow \boldsymbol{x}_\rho$, specify the root box of $s$,
             $\mathbf{s} \leftarrow [s]$
while $\mathbb{L}^{\triangledown}(s) \neq \emptyset$ & $|\mathbb{L}(s)| < \overline{m}$ & $\psi\left(\mathrm{argmax}_{\rho\mathsf{v} \in \mathbb{L}^{\triangledown}(s)} \psi(\rho\mathsf{v})\right) > \overline{\psi}$ do
    $\rho\mathsf{v} \leftarrow$ random_sample$\left(\underset{\rho\mathsf{v} \in \mathbb{L}^{\triangledown}(s)}{\mathrm{argmax}}\, \psi(\rho\mathsf{v})\right)$  // sample uniformly from nodes with
     largest $\psi$
    $s \leftarrow s$ with node $\rho\mathsf{v}$ split             // split the sampled node and update $s$
    s.append($s$)        // append the new SRP state with an additional split
end
```

corresponding random stopping time. Care must be taken in defining splittable nodes as well as the values of the stopping parameters $\overline{\psi}$ and $\overline{m}$, to ensure that $\psi(\rho\mathsf{v}) \leq \overline{\psi}\ \forall \rho\mathsf{v} \in \mathbb{L}^{\triangledown}(s(T))$ *and* $|\mathbb{L}(s(T))| \leq \overline{m}$. If the initial state $S(t = 0)$ is the root $s \in \mathbb{S}_0$ then PQMC $\{S(t)\}_{t \in \mathbb{Z}_+}$ on $\mathbb{S}_{0:\overline{m}-1}$ satisfies $S(t) \in \mathbb{S}_t$ for each $t \in \mathbb{Z}_+$, i.e., the state at time t has $t + 1$ leaves or t splits. Note that the initial state can be specified by a sample from any distribution on $\mathbb{S}_{0:\overline{m}-1}$. In fact, we will use a distribution defined from sample paths of one PQMC with a specific priority function to initialize several independent PQMCs with a different and complementary priority function to find our density estimate as described next.

Statistically Equivalent Blocks PQMC or SEB-PQMC A statistically equivalent block (SEB) partition of a sample space is some partitioning scheme that results in equal numbers of data points in each element (block) of the partition [14] (except possibly in blocks on the boundary of the partitioned space). A statistically equivalent blocks (SEB)-based SRP partitioning scheme specified by the PQMC with (i) priority function $\psi(\rho\mathsf{v}) = \#\boldsymbol{x}_{\rho\mathsf{v}}$, i.e., the number of sample points associated with a node $\rho\mathsf{v}$, (ii) ψ-related stopping condition $\overline{\psi} = \overline{\#}$ and (iii) $\overline{m}$, is denoted by SEB-PQMC. Thus, at stopping time T, the SRP s realized by the SEB-PQMC will be such that either $\mathbb{L}^{\triangledown}(s) = \emptyset$ or $|\mathbb{L}(s)| \leq \overline{m}$ or $\#\boldsymbol{x}_{\rho\mathsf{v}} \leq \overline{\#}\ \forall \rho\mathsf{v} \in \mathbb{L}^{\triangledown}(s)$. The operation may only be considered to be successful if $|\mathbb{L}(s)| \leq \overline{m}$ and $\#\boldsymbol{x}_{\rho\mathsf{v}} \leq \overline{\#}\ \forall \rho\mathsf{v} \in \mathbb{L}^{\triangledown}(s)$. Care must be taken to ensure that the operation is successful. Therefore, an SEB-PQMC can be used to create a final SRP at stopping time T such that each leaf node has at most $\overline{\#}$ of the sample data points associated with it and the total number of leaves is at most $\overline{m}$. Intuitively, SEB-PQMC prioritizes the splitting of leaf nodes with the largest numbers of data points associated with them.



(a) 20 leaves. (b) 40 leaves. (c) 20 leaves. (d) 40 leaves.

Fig. 3. Partition using an SEB-PQMC and SPC-PQMC.

Unfortunately, under an SEB-PQMC partitioning strategy over SRPs, the nodes with least data associated with them will remain unsplit for longer (and will possibly never be split). This tends to result in relatively large regions of very low density in the tails of the density estimate f_n formed from the SRP. Figure 3 shows two partitions of an SRP associated with a highly correlated dataset super-imposed on it. As the number of leaves in the partition increases from 20 (Figure 3(a)) to 40 (Figure 3(b)), large sub-boxes containing very little sample data remain unsplit. The effect can be especially distorting to the resulting histogram when the axis-aligned root box required by the SRP is a poor fit to the data (when large areas of the root box contain no data points). Thus, we would like to carve out empty space within the root box while remaining in $\mathbb{S}_{0:\infty}$. The next PQMC ameliorates this undesirable feature of SEB-PQMC and can be used collaboratively with SEB-PQMC to still ensure asymptotic L_1 consistency of the joint partitioning strategy.

Support Carving PQMC or SPC-PQMC A support-carving (SPC) SRP partitioning scheme specified by the PQMC with (i) $\psi(\rho\mathsf{v}) = \varXi(\rho\mathsf{v}) = (1 - \#\boldsymbol{x}_{\rho\mathsf{v}}/n)\mathrm{vol}\,(\rho\mathsf{v})$, the SPC priority function which prioritizes node $\rho\mathsf{v}$ according to the relative lack of its empirical measure when further scaled by its volume, (ii) ψ-related stopping condition $\overline{\varXi}$ and (iii) maximum number of leaves $\overline{m}^{\varXi}$, is denoted by SPC-PQMC. Using SPC-PQMC to carve out "empty space" in the complement of the empirical support can be thought of as an 'inversion' of the SEB-PQMC: instead of prioritizing splitting of nodes with the largest number of data points associated with them as done by SEB-PQMC, the SPC-PQMC prioritizes splitting of non-empty leaf nodes with large boxes but few data points and thus is likely to result in one of the child nodes being devoid of any sample data. Hence, the two PQMCs are thought to be complementary. The carving effect is illustrated in Figure 3. The partitions are created for the same set of correlated data shown in Figure 3 using SPC-PQMC. Figures 3(c) and 3(d) show the partition when the SRP has 20 and 40 leaves, respectively. Partitioning is concentrated in the regions of sparse sample data and the effect is to reduce the size of the sub-boxes of the partition into which these sparse data points fall, in effect more tightly enclosing the support of the data as desired while still remaining in $\mathbb{S}_{0:\infty}$.

Algorithm 1 with the SPC priority function $\varXi$ can be used to obtain a *core support-carved path* in $\mathbb{S}_{0:\overline{m}^{\varXi}}$ by initializing from the root SRP $s \in \mathbb{S}_0$ through the procedure $\texttt{PQMC}(s, \varXi, \overline{\varXi}, \overline{m}^{\varXi})$ with $\overline{\varXi} = 0.0$. Thus, the partitioning process of this core SPC-PQMC only stops when the SRP has $\overline{m}^{\varXi}$ leaves or aborts if there a no splittable nodes. More general support-carved paths can be generated by specifying $\overline{\varXi} > 0.0$ or imposing constraints on the minimum volume of splittable nodes if deemed appropriate for the underlying data generation process.

Joint exploration An SPC-PQMC of Section 2 alone will not give an effective data-driven partitioning strategy, but used in conjunction with an SEB-PQMC it can improve the SRP histogram. An initial SPC-PQMC can be run for a short time (specifying $\overline{\varXi} = 0.0$ and a relatively low value of $\overline{m}^{\varXi}$, say a small fraction of the total number of leaves $\overline{m}$), followed by an SEB-PQMC. The empty elements of the partition 'carved out' will be ignored by the SEB-PQMC, under which partitioning will be concentrated on the areas where most of the sample data has fallen. The core support-carved path from SPC-PQMC over $\mathbb{S}_{0:\overline{m}^{\varXi}}$ through $\texttt{PQMC}(s, \varXi, \overline{\varXi}, \overline{m}^{\varXi})$ with $\overline{\varXi} = 0.0$ will be used to determine $c^{\varXi}$ many spread-out initial conditions for launching multiple independent SEB-PQMCs. It is along these $c^{\varXi}$ many joint SPC/SEB-PQMC paths, which may be viewed as a system of $c^{\varXi}$ many SEB-PQMC tributaries spread along the core support-carved path of the SPC-PQMC, that we conduct MAP estimation for a given τ-specific prior by simply identifying the SRP state s with the highest log-posterior value. The search for the density estimate is conducted sequentially along each of the SPC/SEB-PQMC paths. The $c^{\varXi}$ many initial conditions from the core support-carved path for launching the SEB-PQMCs, should be well spread out in order to facilitate a better search strategy over $\mathbb{S}_{0:\overline{m}}$, and in particular, contain the SEB-PQMC launched from the root node as one of its joint SPC/SEB-PQMC paths. In essence, we can view the joint exploration as one that will necessarily improve upon the estimate found by the SEB-PQMC initialized from the root node (which satisfies the three conditions of [8] and thus asymptotically L_1-consistent as shown in [10]).

Smoothing Figure 4 shows two different SRP histograms constructed using two different values of $\overline{\#}$ for the same dataset of $n = 10^5$ points simulated under the standard bivariate Gaussian density. A small $\overline{\#}$ produces a histogram that is under-smoothed with unnecessary spikes (Fig. 4 left) while the other histogram with a larger $\overline{\#}$ used as the SEB stopping criterion is over-smoothed (Fig. 4 right). We will use the leave-one-out cross-validation to choose an optimally smoothed density.

Let $x_1, \ldots, x_n$ be the sample data, $\mathcal{L}\left(f_n\right)$ be the likelihood of the data given SRP s, $m = m(s) = |\mathbb{L}(s)|$ be the number of cells or leaf boxes in the partition

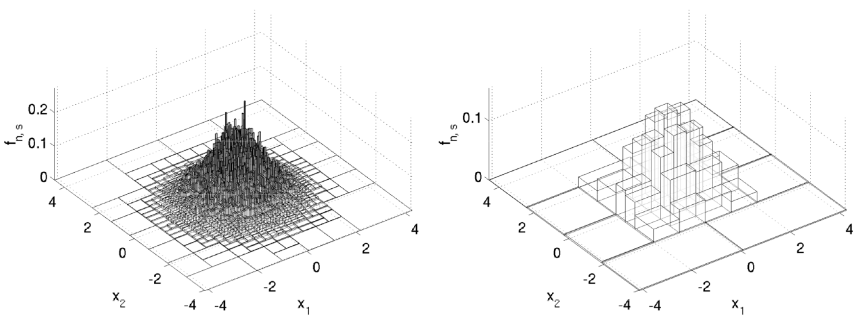


Fig. 4. Two histogram density estimates for the standard bivariate Gaussian density. The left figure shows a histogram with 1485 leaf nodes where $\overline{\#} = 50$ and the histogram on the right has $\overline{\#} = 1500$ resulting in 104 leaf nodes.

given by the SRP s, i.e., $m = k + 1$ if $s \in \mathbb{S}_k$. For a given τ, finding the regularized (penalized) maximum likelihood estimate, amounts to solving the following maximization problem over $\mathbb{S}_{0:\infty}$:

$$f_{n,\tau} := \underset{s \in \mathbb{S}_{0:\infty}}{\operatorname{argmax}} \log(\pi(s;\tau)) = \underset{s \in \mathbb{S}_{0:\infty}}{\operatorname{argmax}} \; \log \mathcal{L}\left(f_n\right) - \left(\frac{m(s)}{\tau}\right) \;.$$

We will solve this smoothing problem by minimizing Stone's leave-one-out cross-validation score $\widehat{J}(\tau)$, a nearly unbiased estimator of the expected L_2 loss $\int (f_{n,\tau}(x) - f(x))^2 dx$ (eg. [15]), given by:

$$\widehat{J}(\tau) = \int \left(f_{n,\tau}(x)\right)^2 dx - \frac{2}{n} \sum_{i=1}^{n} f_{n,\tau}^{(-i)}(x_i) \;,$$

where $f_{n,\tau}^{(-i)}$ is the estimate obtained from the data set by leaving out x_i.

3 Going distributed

Simply computing multiple histograms in parallel has a number of limitations. First of all the number of histograms to compute will typically be relatively small. More importantly one is still limited by the memory capacity of each individual machine, since each histogram would be based on all data points.

For simplicity we will consider only axis parallel splits through cell centres, with respect to a priority depending on volume and count (this covers support carving and SEB). The idea generalizes to priority functions depends on any recursively computable statistic and splitting hyper planes that may depend on some appropriate functions of both the cell and the data points it contains.

Recall that Algorithm 1 proceeded roughly as follows: given a sequence $x = (x_1, \ldots, x_N)$ of (data) points in $\mathbb{R}^d$ produce $s(0), s(1), \ldots$ the sequence of partitions given by splitting according to some priority $\psi\colon \mathbb{N}\times\mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$ starting at any $s(0)$. Thus $s(n+1)$ equals $s(n)$ with one cell split, according to the highest priority assigned by ψ. Schematically it thus proceeds as in Algorithm 2.

ALGORITHM 2: High level sketch of sequential splitting procedures

```
input      : $\psi$: Count × Volume $\to \mathbb{R}_{\geq 0}$, a priority function,
             $x_1, \dots, x_N$, (data) points in $\mathbb{R}^d$,
             $s(0)$, initial SRP,
             $\overline{\psi}$, maximum priority for any cell node in final output
output     : Increasing (refining) $s(0), \dots, s(K)$, cells of $s(K)$ have priority below $\overline{\psi}$
initialize: $r \leftarrow s(0)$
while $r$ has any cell of priority above $\overline{\psi}$ do
    output $r$
    $r \leftarrow r$ with the cell of highest priority split
end
output $r$
```

This is inherently sequential; modulo special properties of the priority functions one needs to split the current most high priority cell before being able to determine which cell to split next. One can make the procedure parallel by using two basic observations. Firstly, from $s(n)$ one can (again sequentially) reconstruct the path $s(n), s(n-1), \dots, s(0)$ without knowing x. Following the path backward entails a sequence of mergers of sibling leafs as opposed to a sequence of splits. To get the final path one merges in order of *least* priority of the parent.

ALGORITHM 3: Backtracking from an intermediate SRP

```
input      : $\psi$: Count × Volume $\to \mathbb{R}_{\geq 0}$, a priority function,
             $s(0)$, "intermediate" SRP
output     : Decreasing (coarsening) $s(0), s(1), \dots, s(K)$, with $s(K)$ trivial
initialize: $r \leftarrow s(0)$
while $r$ has more than one cell do
    output $r$
    $\mathbf{p} \leftarrow$ parents of leaf nodes in $r$
    $\mathbf{w} \leftarrow$ priority ($\psi$) of all $\mathbf{p}$ based on volume/count of children
    $r \leftarrow r$ with children of the least priority node according $\mathbf{W}$ merged
end
output $r$
```

Both count and volume, and thus the priority ψ, for any internal node can be computed by adding the counts and volumes of its children. Knowing only leaf counts and volumes one can therefore determine the last node to have been split. This is summarized in Algorithm 3 and illustrated in Figure 5.

The second observation is that given a threshold $c > 0$ the tree S given by starting at $s(0)$ and splitting leafs in an arbitrary order until all have a priority less than c will satisfy $S = s(k)$ for some k. This is most easily seen by letting $S' = s(l)$ where $l = \min\{k \mid \text{all cells in } s(k) \text{ priority below } c\}$ and noting that $S = S'$. The last statement holds since any cell split to get S will sooner or later have to be split when constructing S', and vice versa. Formally this follows by induction on the depth, within the tree, of the cell.

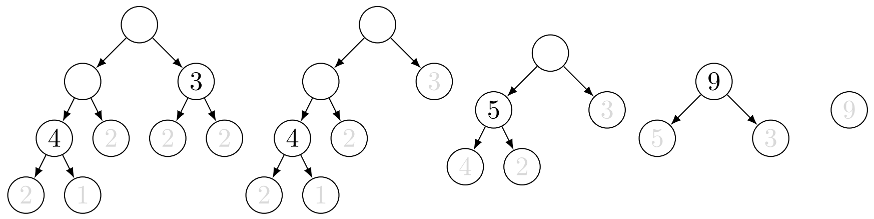


Fig. 5. Illustration of Algorithm 3, each tree is an SRP with nodes labelled with its priority and priorities of inner nodes computed recursively

This gives us the freedom to pick the order in which cells are split. In particular we can split multiple cells at once. Using the first observation the sequence of intermediary trees can be reconstructed. While this reconstruction is again sequential, performing mergers will generally be faster than splitting, since the latter involves traversals of all data points in the cell.

Once we can split cells simultaneously one may as well split *all* nodes with a priority over the threshold c. Note that any intermediate SRP computed this way need not lie along the path output by the original procedure; a node may well split into two where at least one has a higher priority than something split simultaneously (see Figure 6).

The path produced by Algorithm 2 can now be found using Algorithm 4 followed by Algorithm 3. This should yield a gain in performance when the number of data points is large, so that a merge is significantly faster than a split, assuming one can perform all splits in each iteration in parallel.

ALGORITHM 4: High level sketch of parallel splitting procedures

```
input/output: As in Algorithm 2
initialize   : $r \leftarrow s(0)$
while $r$ has any cell of priority above $\overline{\psi}$ do
    $r \leftarrow r$ with all cells of priority above $\overline{\psi}$ split
end
output $r$
```

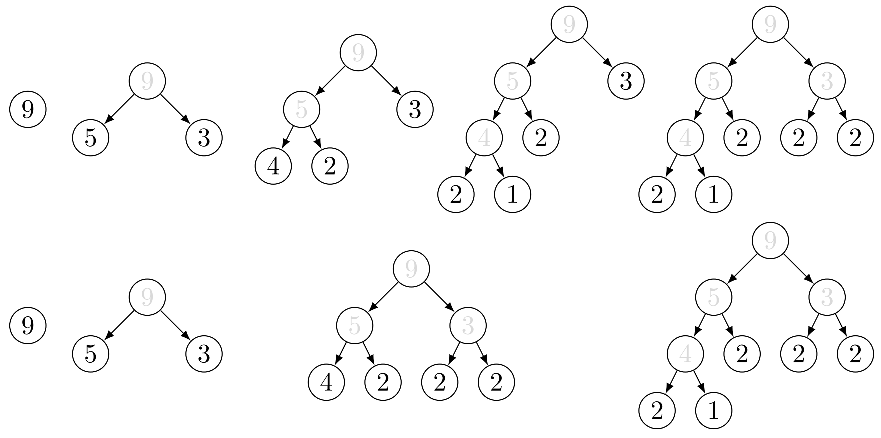


Fig. 6. Illustration of sequential (upper) and parallel (lower) splitting procedures, each tree represents an SRP with each node labeled by the priority of that node

In order for the procedure outlined above to translate into a distributed algorithm one needs a distributed representation that allows for efficient distributed and parallel computation of how a given set of cells split.

The way we achieve this is by storing the current tree implicitly by tagging each data point by the cell in which it lies. That is to say, the points $(x_1, \ldots, x_6)$ and partition $\{A, B, C\}$ in Figure 7 would be represented by the sequence $(A, x_1), (A, x_2), (B, x_3), (C, x_4), (B, x_5), (A, x_6)$.

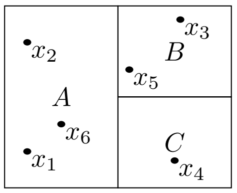


Fig. 7. Partitioned points represented by $(A, x_1), (A, x_2), (B, x_3), (C, x_4), (B, x_5), (A, x_6)$

Counting the current number of points in each cell is now a simple reduce operation available in, for example, Apache Spark under the name `countByKey`. The resulting map (counting the number of points for each cell) can either be collected to a single machine or itself remain distributed. It involves communication proportional to the current number of non-empty cells; in the worst case scenario every machine has to send its counts for each non-empty cell.

As volume is a function only of the cell this gives enough information for determining which cells needs to be split. Simply compute the priority (a function of count and volume) and filter to get only those cells with a priority exceeding the predetermined threshold c.

Once one has determined which cells to be split getting the new partition is a point-wise operation. For each pair (A, x) (point x in cell A) one checks if A is to be split and if so on which side of the splitting hyper plane x lies. If x lies below produce the pair (A', x) where A' is the lower part (relative to the hyper plane) of A and otherwise produce (A'', x) where A'' is the upper part of A.

Most of this computation fits directly into the MapReduce framework since it simply applies a function point-wise (thus a map operation). How to communicate of cells to be split depends on how the resulting map was stored; either one broadcasts it from a single machine or performs a distributed join operation.

ALGORITHM 5: MapReduce-friendly parallel splitting procedure

```
input      : $\psi$: Count × Volume → $\mathbb{R}_{\geq 0}$, a priority function,
             $x_1, \ldots, x_N$, (data) points in $\mathbb{R}^d$,
             $\overline{\psi}$, maximum priority for any cell node in final output
output     : SRP with cells of priority below $\overline{\psi}$ and cell count corresponding to the
             number of points in $x_1, \ldots, x_N$ in the cell
initialize:  $\rho \leftarrow$ root cell
             $y \leftarrow [(\rho, x_1), \ldots, (\rho, x_N)]$ // distributed array
             $c \leftarrow [(\rho, N)]$
while c has any cell/count pair with priority above $\overline{\psi}$ do
    // Implemented as a distributed map operation
    foreach $(a, x) \in y$ do
        if $\psi$(count of a in c, volume of a) $> \overline{\psi}$ then
            if x is below splitting hyperplane of a then
                $a' \leftarrow$ subcell of $a$ below splitting hyperplane
            end
            else
                $a' \leftarrow$ subcell of $a$ above splitting hyperplane
            end
            replace $(a, x)$ by $(a', x)$
        end
    end
    // Implemented as a distributed reduce operation
    $c \leftarrow [\,]$
    foreach $(a, x) \in y$ do
        increment count in $c$ at $a$ by 1 (adding key $a$ if needed)
    end
end
output SRP with leaves/counts corresponding to cells/counts in $c$
```

The procedure can be improved by pruning pairs corresponding to cells with a priority below the threshold. The relevant computations are already performed when determining, for each tuple, whether or not the cell is one to be split (see

Algorithm 5). Filtering these points avoids repeating this check in subsequent iterations; the threshold remains fixed so these cells will not be split in a future iteration either.

ALGORITHM 6: Optimization of Algorithm 4

```
while c has any cell/count pair with priority above $\overline{\psi}$ do
    // Now becomes a map and filter operation
    foreach $(a, x) \in y$ do
        if $\psi(count\ of\ a\ in\ c, volume\ of\ a) > \overline{\psi}$ then
            compute new cell $a'$ as before
            replace $(a, x)$ by $(a', x)$
        end
        else
            delete $(a, x)$
        end
    end
    update $c$ as before, keeping counts for cells that have passed threshold
end
output SRP with leaves/counts corresponding to cells/counts in $c$
```

For good performance one needs also a good representation for the cells in the distributed sequence of cell/point tuples. For our purposes a convenient representation is given by identifying cells with nodes in the (infinite) plane binary tree as described in Section 2 and illustrated in Figure 1.

For practical purposes such a node can be identified with a positive natural number as follows: the root node is assigned number 1, the left sub-child of the node with number n is assigned the number $2n$, and the right child of the node with number n is assigned number $2n+1$. This is simply a binary encoding given by an initial 1 followed by the unique path from the root node; left steps are encoded by 0 and right steps are encoded by 1.

Using multiple precision integer types (as provided by for example gmp[2]) this is yields a fairly compact representation (bits proportional to tree depth) where many operations can be implemented in terms of bit-level operations. For example, passing to ancestors corresponds to taking right shifts and the depth (distance to the root node) is the index of the most significant bit.

4 Implementation and Results

A complete, though non-parallel, `C++` implementation of regular paving trees is available in the open source library `mrs2` [11]. Our reference Scala implementation of the parallel density estimation procedure on top of Apache Spark is also publicly available [16].

Motivated by an industrial problem in fraud detection, we simulated $n = 13 \times 10^7$ points in 2 and 10 dimensions from a multivariate density. A typical computation on an Apache Spark cluster with 5 Workers (totaling 30.0 GB Memory and 4.4 Cores) in 2 and 10 dimensions took about 1.79 and 5.8 hours with estimated L1 errors of 0.03 and 1.13, respectively. Density estimates for such large datasets cannot be obtained in a single commodity machine.

References

1. Fisher, R.A.: Theory of statistical estimation. Mathematical Proceedings of the Cambridge Philosophical Society **22**, 700–725 (1925)
2. Granlund, T., the GMP development team: GNU MP: The GNU Multiple Precision Arithmetic Library, `http://gmplib.org/`
3. Gray, A.G., Moore, A.W.: Nonparametric Density Estimation: Towards Computational Tractability. In: SIAM International Conference on Data Mining. pp. 203–211. SIAM, San Francisco, California, USA (2003)
4. Harlow, J., Sainudiin, R., Tucker, W.: Mapped regular pavings. Reliable Computing **16**, 252–282 (2012)
5. Kieffer, M., Jaulin, L., Braems, I., Walter, E.: Guaranteed set computation with subpavings. In: Kraemer, W., Gudenberg, J. (eds.) Scientific Computing, Validated Numerics, Interval Methods, Proceedings of SCAN 2000, pp. 167–178. Kluwer Academic Publishers, New York (2001)
6. Klemelä, J.: Smoothing of Multivariate Data: Density Estimation and Visualization. Wiley, Chichester, United Kingdom (2009)
7. Lu, L., Jiang, H., Wong, W.H.: Multivariate density estimation by bayesian sequential partitioning. Journal of the American Statistical Association **108**(504), 1402–1410 (2013)
8. Lugosi, G., Nobel, A.: Consistency of Data-Driven Histogram Methods for Density Estimation and Classification. The Annals of Statistics **24**(2), 687–706 (1996)
9. Meier, J.: Groups, Graphs and Trees: An Introduction to the Geometry of Infinite Groups. Cambridge University Press, Cambridge, United Kingdom (2008)
10. Sainudiin, R., Teng, G.: Minimum distance estimation with universal performance guarantees over statistical regular pavings. Manuscript (2nd revision), LaMaStEx, Uppsala Centre, Box 480, Uppsala University, 75106 Uppsala, Sweden (2018), `http://lamastex.org/preprints/20180405_MDEYatracosThis.pdf`
11. Sainudiin, R., York, T., Harlow, J., Teng, G., Tucker, W., George, D.: MRS 2.0, a C++ class library for statistical set processing and computer-aided proofs in statistics. `https://github.com/raazesh-sainudiin/mrs2` (2008–2018)
12. Samet, H.: The Design and Analysis of Spatial Data Structures. Addison-Wesley Longman, Boston (1990)
13. Stanley, R.P.: Enumerative combinatorics. Vol. 2, Cambridge Studies in Advanced Mathematics, vol. 62. Cambridge University Press, Cambridge (1999)
14. Tukey, J.W.: Non-Parametric Estimation II. Statistically Equivalent Blocks and Tolerance Regions — The Continuous Case. The Annals of Mathematical Statistics **18**(4), 529–539 (1947)
15. Wasserman, L.: All of Statistics: A Concise Course in Statistical Inference. Springer, New York (2003)
16. Wiklund, T.: SparkDensityTree. `https://github.com/TiloWiklund/SparkDensityTree` (2018)