Information-theoretic partition-based models for interpretable machine learning

Yang, L.

Chapter 5

Summarizing Two-dimensional Data with MDL-based Discretization by Histograms

This chapter has been published as Yang, L, Baratchi, M, and van Leeuwen, M **Unsupervised discretization by two-dimensional mdl-based histogram**. *Machine Learning, 2023: 1-35.*

Chapter Abstract

Unsupervised discretization is a crucial step in many knowledge discovery tasks. The state-of-the-art method for one-dimensional data infers locally adaptive histograms using the minimum description length (MDL) principle, but the multi-dimensional case is far less studied: current methods consider the dimensions one at a time (if not independently), which result in discretizations based on rectangular cells of adaptive size. Unfortunately, this approach is unable to adequately characterize dependencies among dimensions and/or results in discretizations consisting of more cells (or bins) than is desirable.

To address this problem, we propose an expressive model class that allows for far more flexible partitions of two-dimensional data. We extend the state of the art for the one-dimensional case to obtain a model selection problem based on the normalized maximum likelihood, a form of refined MDL. As the flexibility of our model class comes at the cost of a vast search space, we introduce a heuristic algorithm, named PALM, which partitions each dimension alternately and then merges neighboring regions, all using the MDL principle. Experiments on synthetic data show that PALM 1) accurately reveals ground truth partitions that are within the model class (i.e., the search space), given a large enough sample size; 2) approximates well a wide range of partitions outside the model class; 3) converges, in contrast to the state-of-the-art multivariate discretization method IPD. Finally, we apply our algorithm to three spatial datasets, and we demonstrate that, compared to kernel density estimation (KDE), our algorithm not only reveals more detailed density changes, but also fits unseen data better, as measured by the log-likelihood.

5.1 Introduction

Discretization, i.e., the transformation of continuous variables into discrete ones, is part of numerous data analysis workflows, making it a crucial step for a wide variety of applications in knowledge discovery and predictive modeling. However, many different discretization methods exist and it is often not easy to determine which method should be used. As a result, naïve methods such as equal-length and equal-frequency binning are still widely used, often with the number of bins chosen more or less arbitrarily, which can lead to suboptimal discretization.

A good discretization strikes a balance between the amount of preserved information and the complexity of the representation of the discretized data, so as to avoid discretizations that are either too coarse—resulting in severe loss of information—or too fine-grained—resulting in a bin per data point in the extreme case.

Achieving an optimal balance has been thoroughly studied for supervised discretization, i.e., discretization using additional information from a target variable. Optimal discretizations have been formalized using 1) statistical quantities, e.g., Pearson's chi-square (Boullé 2004), 2) information-theoretic scores based on entropy or the minimum description length (MDL) principle (Fayyad and Irani 1993; Jin et al. 2009), and 3) Bayesian approaches (Boullé 2006).

In contrast, unsupervised discretization, which does not assume a target variable, has long been understudied (Kotsiantis and Kanellopoulos 2006). It serves a different purpose: supervised discretization aims to reduce the loss of information about the distribution of the target variable conditioned on the features (Boullé 2004; Fayyad and Irani 1993; Kerber 1992), whereas unsupervised discretization aims to preserve information about the probability distribution of the variable to be discretized (Biba et al. 2007; Schmidberger and Frank 2005).

This makes histograms well-suited to unsupervised discretization, and particularly adaptive histograms. An adaptive histogram is a probabilistic model that approximates probability density by piecewise constant densities, partitioning the data into bins such that 1) the probability density within each bin is approximately uniform (otherwise finer bins are needed), and 2) probability densities of neighboring bins are significantly different (otherwise they should be merged). Kontkanen and Myllymäki (2007b) formalized this goal for one-dimensional adaptive

histograms based on the minimum description length (MDL) principle (Rissanen 1978), which is now considered to be the state-of-the-art univariate discretization method (Kameya 2011; Marx et al. 2021; Nguyen et al. 2014).

The MDL principle (Grünwald and Roos 2019; Rissanen 1978) is arguably one of the best off-the-shelf approaches for model selection tasks such as selecting a histogram model for given data, as it provides a means to naturally trade-off goodness-of-fit with model complexity. It achieves this by defining the "best" probabilistic model for given data as the model that results in the best *compression* of data and model together, which has been widely used in data mining and machine learning tasks (Galbrun 2020).

Flexible multi-dimensional discretization. Traditional discretization methods are defined for one-dimensional (or univariate) data, and multi-dimensional (or multivariate) data is typically discretized by separately and independently discretizing each dimension, which ignores any dependencies between the dimensions. Multivariate discretization methods aim to take such dependencies into account, but they suffer from two problems. First, most methods focus on supervised discretization (Bay 2001; Ferrandiz and Boullé 2005; Kurgan and Cios 2004; Kwedlo and Kretowski 1999). Second, existing methods produce an adaptive grid based on the Cartesian product of the discretization results of individual dimensions. This approach ignores that the density of one dimension may change more drastically for certain values of another dimension; hence, appropriate binning of one dimension may depend on the values of the other dimensions.

For instance, consider a two-dimensional synthetic dataset sampled from a mixture of Gaussians as shown in Figure 5.1 (leftmost)[1]. To adequately discretize data from this distribution, the binning of the x-axis should be different depending on whether y is above or below the black dashed line, in order to capture the different density changes for the Gaussian distribution (above) and the Gaussian mixture (below). Similarly, the binning of the y-axis should be different depending on whether x is left or right to the red dashed line. This motivates us to consider partitions that are more flexible than adaptive grids: we consider all partitions that can be obtained by clustering the "cells" of a fine-grained fixed-grid. The remaining three plots in Figure 5.1 show the density plots obtained by

[1]For reproducibility, the data is generated by the mixture of $N[\binom{0}{1}, \left(\begin{smallmatrix}1 & 0\\ 0 & 1\end{smallmatrix}\right)]$, $N[\binom{1.5}{4}, \left(\begin{smallmatrix}2 & 0\\ 0 & 1\end{smallmatrix}\right)]$, $N[\binom{3}{0}, \left(\begin{smallmatrix}1 & 0\\ 0 & 1\end{smallmatrix}\right)]$, $N[\binom{7}{2}, \left(\begin{smallmatrix}1 & 0\\ 0 & 1\end{smallmatrix}\right)]$ with all mixing coefficients 0.25; sample size is 40 000.

1) IPD (Nguyen et al. 2014), the state-of-the-art multivariate unsupervised discretization method, 2) the one-dimensional MDL-based histogram method (Kontkanen et al. 1997) applied independently on each dimension, and 3) our method. Our method produces the density estimation that most resembles the shape of the original contour, as we allow the bins of one dimension to depend on the value of another dimension.

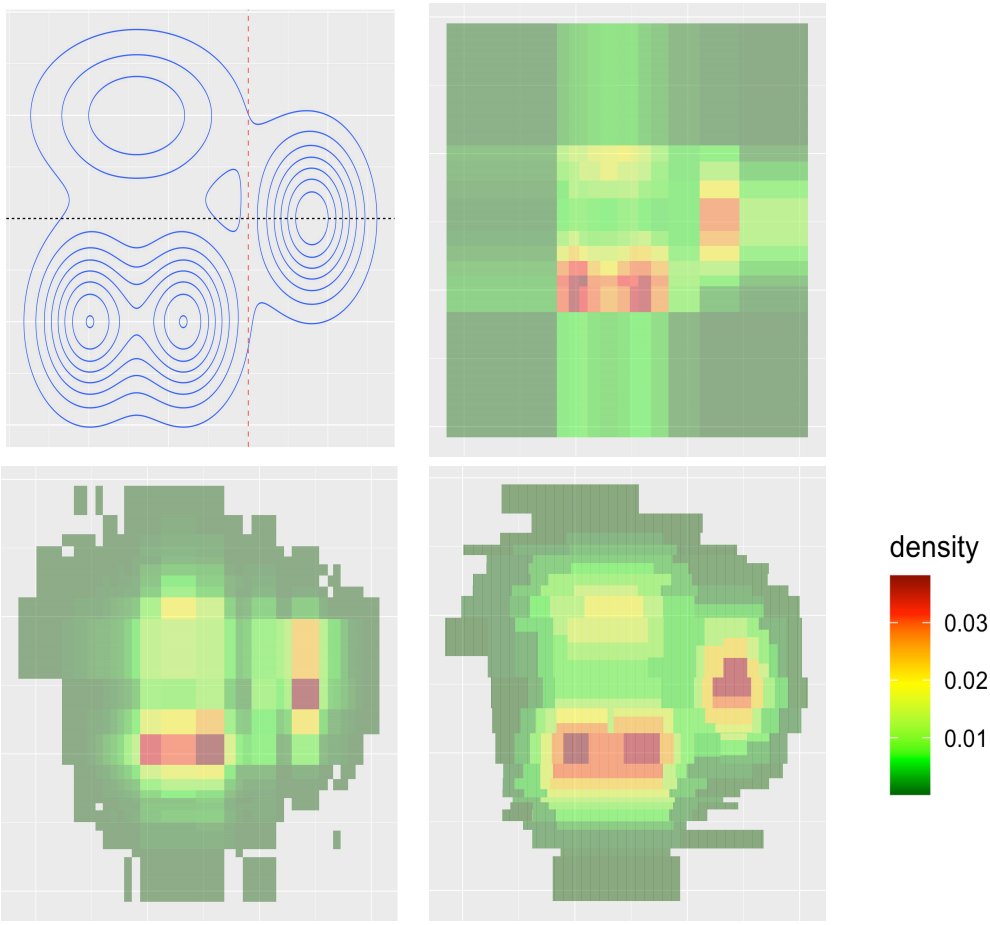

Figure 5.1: Distributions of a two-dimensional dataset simulated from a mixture of Gaussian distributions; from top-left to bottom-right: 1) true probability density contour, 2) partitioning by IPD (Nguyen et al. 2014), 3) partitioning by separately discretizing each dimension with the MDL histogram (Kontkanen and Myllymäki 2007b), 4) flexible partitioning by PALM, our algorithm.

Approach and contributions. We consider the problem of learning two-dimensional

histogram models that enable far more flexible partitions than regular adaptive grids. That is, we allow any partition that can be obtained by iteratively merging adjacent cells of a fixed grid, which allows for learning models that provide accurate density estimates while not having more bins than strictly necessary (thereby avoiding overfitting and providing clear region boundaries, i.e., adjacent bins must have different density estimates).

We formalize the two-dimensional histogram construction problem as a model selection task using the MDL principle. For this we build on the one-dimensional MDL-based histogram selection problem as introduced in the seminal work by Kontkanen and Myllymäki (2007b), because it is both theoretically elegant and practically fast. Specifically, it adopts the *normalized maximum likelihood* (NML) encoding scheme, a form of *refined MDL* (Grünwald 2007; Grünwald and Roos 2019) that provides minimax regret, and employs a fast dynamic programming algorithm to find the optimal solution.

The existing approach for one-dimensional histograms cannot be trivially extended to multiple dimensions though, hence we make a number of technical contributions.

First, we solve the challenge of computing the so-called *parametric complexity* (Grünwald and Roos 2019) for the multi-dimensional case.

Second, we observe that efficiently finding the MDL-optimal two-dimensional histogram is infeasible and propose PALM, a heuristic algorithm for learning two-dimensional histograms. PALM combines top-down (partition) and bottom-up (merge) search strategies by 1) first partitioning the data by iteratively splitting regions, and 2) then iteratively merging neighboring regions if their densities are similar. In each step, the MDL principle is used as decision criterion; as a result, our algorithm requires neither hyper-parameters[2] nor any pre-defined stopping criterion to be specified. It automatically adapts to both local density structure, as shown in the example in Figure 5.1 and, later, in Sections 5.7 and 5.8.

Third, we make several improvements to the dynamic programming algorithm used for the one-dimensional MDL histogram, which we use as a building block for our algorithm. Specifically, as described in Section 5.5, we 1) correct a minor theoretic flaw related to computing the code length that is needed to encode the histogram model, and 2) reduce the time complexity by simplifying the dynamic

[2]The precision with which the data is recorded can be used to set the granularity of the initial base grid.

programming recursion.

We perform extensive experiments to show that our algorithm 1) accurately recovers ground truth histograms, 2) approximates well ground truth partitions that are not within the model class, and 3) outperforms IPD (Nguyen et al. 2014), the state-of-the-art algorithm for unsupervised multi-dimensional discretization. Further, case studies on spatial data show that, compared to kernel density estimation (KDE), our algorithm not only reveals more detailed density changes, but also fits unseen data better, as measured by the log-likelihood.

We restrict the scope of this chapter to two-dimensional data for three reasons. First, two-dimensional discretization methods have many potential applications in the domain of spatial data analysis, e.g., using GPS data, where ad-hoc discretization methods are still widely used (Cao et al., 2014). The case studies demonstrate that our method can successfully reveal interesting patterns from GPS data. Second, as our approach uses more flexible partitions than adaptive grids, the search space is very large even for two-dimensional data. Our algorithm for the two-dimensional case should be regarded as a step towards solving the algorithmic challenge for higher dimensions, but does not solve it completely. Third, focusing on the two-dimensional case allows us to more easily examine the results empirically, e.g., to verify desired properties such as adaptivity to sample size and local density structure.

5.2 Related work

We briefly review previous work concerning discretization methods, histogram models, and tree-based models for density estimation.

Unsupervised univariate discretization. Most unsupervised univariate discretization methods are rather straightforward and concern equal-width or equal-frequency binning, which in practice usually involve ad-hoc choices for the number of bins or for the frequency in each bin.

Clustering techniques such as k-means (Friedman et al. 2001) or Bayesian clustering (Kontkanen et al. 1997) are also used in discretization; however, they ignore the possible heterogeneity within the cluster and choices of hyper-parameters are usually required.

More advanced criteria rely on density estimation and specifically construct-

ing adaptive histograms. Apart from the MDL-based histogram (Kontkanen and Myllymäki 2007b) already mentioned in Section 5.1, Schmidberger and Frank (2005) proposed to construct adaptive histograms by recursive binary partition with cross-validation. A local heuristic is used to decide the cut point, and cross-validation is used to choose the number of intervals; in contrast, the MDL-based histogram (Kontkanen and Myllymäki 2007b) uses a global score with a dynamic algorithm that optimizes the cut points and the number of bins simultaneously. Moreover, an adaptive histogram can also be selected as the one whose density estimation result is closest to the result of *kernel density estimation* (Biba et al. 2007), where cross-validation is used to prevent overfitting. As the true density is apparently not known, cross-validation is performed by Monte Carlo sampling-based methods. However, cross-validation is known to be computationally expensive, and the influence of choosing different kernels on discretization is not reported.

Bayesian approaches have been widely used in adaptive histograms (Gasparini 1996; Liu and Wong 2014; Lu et al. 2013; Scricciolo 2007; Van Der Pas and Rocková 2017). These methods treat all possible histograms as the model class and put a prior distribution on it, and the resulting posterior distribution is directly used for density estimation (by calculating the marginal distribution). Therefore, although these Bayesian approaches often provide theoretic guarantees as density estimation methods, they do not provide an individual adaptive histogram that can be used for discretization.

Unsupervised multivariate discretization. Since discretizing each dimension of multivariate data independently will ignore the dependencies among different dimensions, some methods attempt to reduce the dependencies by PCA- or ICA-based methods (Kang et al. 2006; Mehta et al. 2005)[3]. However, as both methods are based on *linear transformation* of the random vector, they may fail to eliminate nonlinear dependencies. Note that extending these methods to nonlinear PCA or nonlinear ICA may not be suitable for unsupervised discretization tasks, as the uniform distribution is not invariant under nonlinear transformation, and hence we cannot obtain an adaptive histogram of the original data by inversely transforming the adaptive histogram constructed on the nonlinearly transformed data.

[3]Note that the ICA-based method (Kang et al. 2006) is designed for supervised discretization, but we noticed that the ICA transformation there is not restricted to supervised discretization only.

Lud and Widmer (2000) proposed the so-called "relative unsupervised discretization". The core of this method is to perform clustering on an individual dimension, using different subsets of values. These different subsets are obtained by filtering the dataset using other dimensions, in order to keep the dependency among different dimensions. However, this method does not control the information loss about the probability distribution of the dimension that is to be discretized.

Further, methods trying to optimize the discretization of all dimensions simultaneously exist. One approach is to start from a very fine grid, and merge neighboring subintervals for each dimension if the multivariate probabilities of the data within these two consecutive subintervals are similar (Bay 2001; Nguyen et al. 2014). These methods are based on certain choices of similarity metrics, and require explicit specification of the similarity threshold. We empirically show in Section 5.7 that IPD, the method by Nguyen et al. (2014) that is also based on the MDL principle and is considered the state-of-the-art multivariate discretization method, does not converge in practice.

Finally, Kameya extended the one-dimensional MDL-histogram (Kameya 2011) specifically for time series data, who proposed to discretize time series data by iteratively adjusting the cut points on each dimension until convergence, using the coordinate descent optimization approach.

All these multivariate discretization methods try to optimize the adaptive grid and produce (hyper)rectangular regions. Our method, in contrast, is proposed to produce far more flexible segmentation, which allows the binning of one dimension to be dependent on the values of other dimensions.

Density estimation tree. Algorithmically, our method is very similar to methods using tree models for density estimation (Liu and Wong 2014; Ram and Gray 2011; Yang and Wong 2014), as partitioning the data space by iteratively partitioning each dimension is identical to growing a tree. However, these density estimation trees were developed by adapting the scores used in growing, stopping, and pruning (supervised) decision and regression trees. That is, while our algorithm employs a consistent MDL-based framework for selecting the best model, these density estimation trees use separate optimization scores respectively to fit the model and to control the model complexity, often with user-specified hyperparameters and/or computationally expensive cross-validation.

Moreover, these density estimation trees, as is like most supervised tree models, only do binary partitioning in a greedy manner. On the contrary, our method can split a dimension into multiple bins (from 1 to a pre-determined K_{max}) instead of just two, which is not only more flexible, but also more interpretable, as after partitioning on a certain dimension, within each bin the data points on that dimension can be regarded as approximately uniform.

Finally, our method has an additional merging step, which creates much more flexible partitions of data, resulting in models that are more informative for pattern mining and exploratory data analysis.

Supervised discretization. When discretization is needed for a supervised task such as classification, we can use *supervised discretization*, which means that the target variable is used to assess how much information on the target the discretization maintains. Several criteria can be put in this category, which are mostly based on statistical hypothesis testing or entropy, as summarized in the survey paper by Kotsiantis and Kanellopoulos (2006). The MDL principle has also been used for supervised discretization (Fayyad and Irani 1993; Ferrandiz and Boullé 2005; Gupta et al. 2010; Pfahringer 1995; Zhang et al. 2007), but all of them use the so-called *crude MDL* principle (Grünwald 2007), which is theoretically suboptimal.

5.3 Problem Statement

Informally, we consider the problem of inferring the best two-dimensional histogram for a given sample of continuous data. To make this problem precise, we start off by introducing our notation and definitions. Note that all $\log(\cdot)$ should be read as $\log_2(\cdot)$ unless specified otherwise.

5.3.1 Notation and definitions of data, model, and model class

Consider as data a vector of length n, i.e., $x^n = (x_1, ..., x_n)$, sampled independently from a *random variable* X.

The *sample space* of X, denoted as S, is a *bounded* subset of $\mathbb{R}^2$. Although the sample space of a random variable, e.g., a Gaussian, can be infinite in theory, we always assume it to be a bounded "box" when dealing with a given dataset.

The task of estimating S from the data directly is another research topic, usually referred to as "support estimation" in statistical literature (Cuevas, Fraiman, et al. 1997), and hence is out of the scope of our main focus in this article.

Conceptually, a histogram—no matter whether it is one- or multi-dimensional—is a *partition* of the sample space S, denoted by $\widetilde{S}$ and parametrized by a vector $\vec{f} = (f_1, \ldots, f_K)$. A partition $\widetilde{S}$ is defined as a set of *disjoint* subsets of S, and the union of all these subsets is S itself, i.e., $\widetilde{S} = \{S_1, S_2, \ldots, S_K\}$, where $\forall j \in \{1, \ldots, K\}$, $S_j \subseteq S$, $\bigcup_{j=k}^{K} S_j = S$, and $\forall j, k \in \{1, \ldots, K\}$, $S_j \cap S_k = \emptyset$. We also call these subsets, i.e., elements of $\widetilde{S}$, as *regions*.

Next, we assume that the probability density of X, denoted by $f(X)$, is given by

$$f(X) = \sum_{j \in \{1,\ldots,K\}} \mathbb{1}_{S_j}(X) f_j, \tag{5.1}$$

where $\mathbb{1}_{\{\cdot\}}(\cdot)$ is the *indicator function*. Each f_j is a *constant* and $\vec{f}$ satisfies $\sum_{i=1}^{K} f_j |S_j| = 1$, where $|S_j|$ denotes the geometric area of S_j, i.e., when $X \in S_j$, $f(X) = f_j$. We refer to any partition $\widetilde{S}$ as a *histogram model* that contains a family of probability distributions; i.e., $\forall \vec{f} \in \mathbb{R}^K$, we denote a single probability distribution by $\widetilde{S}_{\vec{f}}$.

We denote the model class as $\mathbb{M}$, representing all possible partitions with K regions that can be obtained by clustering cells of a fixed grid covering S, where $K \in \{1, \ldots, K_{max}\}$. The granularity of the grid, denoted as ϵ, and K_{max} are fixed in advance, but note that they can be set arbitrarily small and large, respectively.

Geometrically, this is equivalent to drawing *inner boundaries* within S along the fixed grid. In practice, ϵ can represent the precision up to which the data is recorded or that is useful for the given task. Although the model class we consider only has inner boundaries consisting of *line segments*, we will show that such a model class is flexible enough to approximate curved inner boundaries in Section 5.7.

5.3.2 Histogram model selection by the MDL principle

We now formally define the task of two-dimensional data discretization as an MDL-based model selection task, using histogram models as the model class.

The MDL principle is arguably one of the best off-the-shelf model selec-

tion methods and has been successfully applied to many machine learning tasks (Grünwald 2007; Hansen and Yu 2001). It has solid theoretical foundations in information theory and naturally prevents overfitting as the optimization criterion always includes the model complexity, defined as the code length (in bits) needed to encode that model (Grünwald 2007).

The basic idea is to *losslessly encode* the model and data together, by firstly encoding the model and then compressing the data using that model. The model resulting in the shortest total *code length* is defined to be MDL-optimal, i.e.,

$$\widetilde{S^*} = \arg\min_{\widetilde{S}\in\mathbb{M}} L(x^n, \widetilde{S}) = \arg\min_{\widetilde{S}\in\mathbb{M}} (L(\widetilde{S}) + L(x^n|\widetilde{S})), \tag{5.2}$$

where $L(\widetilde{S})$ and $L(x^n|\widetilde{S})$ are respectively the code length of the model and the code length of the data compressed by that model. Note that $L(\cdot|\cdot)$ denotes the *conditional* code length (Grünwald 2007); informally, $L(A|B)$ represents the code length of the message a *decoder* needs to receive in order to be able to losslessly reconstruct message A after having already received message B.

We will show in Section 5.4 that properly encoding the model and calculating its corresponding code length $L(\widetilde{S})$ turns out to be very difficult. As a result, we unfortunately cannot regard our model selection task simply as an optimization problem.

To alleviate this, we divide the model selection task into two steps, namely 1) partitioning alternately and 2) merging.

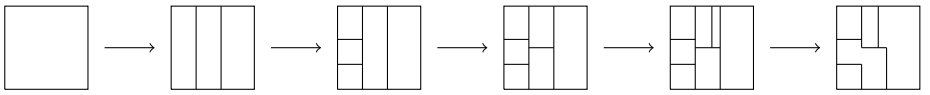

Figure 5.2: An illustration of the partitioning and merging steps. From left to right: alternatively partitioning each region until compression cannot be further improved, and finally merging some of the neighboring regions to further improve compression.

First, we alternately split each region within partition $\widetilde{S}$ (initially $\widetilde{S} = \{S\}$) in one of the two dimensions, then update $\widetilde{S}$ accordingly, and repeat the process. In other words, in each iteration we further split each region within $\widetilde{S}$ in one dimension (i.e., horizontally or vertically), which is equivalent to selecting the best set of horizontal or vertical *cut lines*.

Denote the subset of data points within a certain region $S' \in \widetilde{S}$ as $\{x^n \in S'\}$.

We formally define the task of selecting the set of MDL-optimal cut lines set as

$$\begin{aligned} C^*_{S'} &= \arg\min_{C_{S'} \in \mathbb{C}_{S'}} L(\{x^n \in S'\}, C_{S'}) \\ &= \arg\min_{C_{S'} \in \mathbb{C}_{S'}} (L(C_{S'}) + L(\{x^n \in S'\} | C_{S'})), \end{aligned} \tag{5.3}$$

where $\mathbb{C}_{S'}$ are all possible sets of cut lines, containing $K = \{0, 1, \ldots, K_{max}\}$ cut lines, for the certain region $S' \in \widetilde{S}$ in one certain dimension (i.e., horizontal or vertical), and K_{max} is predetermined a priori to be "large enough" given the task at hand.

In Section 5.5, we will show that searching for the MDL-optimal cut lines for (a subset of) two-dimensional data is the same as searching for the MDL-optimal cut points for the one-dimensional data that is the projection of the two-dimensional data onto the x- or y-axis.

The partitioning step will automatically stop once for each region the MDL-optimal set of cut lines is the null set, i.e., no further partitioning is needed.

Second, we search for all possible clusterings of neighboring regions gained in the previous partitioning step, in a greedy manner. In other words, we consider all possible clustering of regions of the partition gained by the previous partitioning step, which is actually a subset of the full model class $\mathbb{M}$ as defined in Section 5.3.1. We denoted this constrained model class by $\mathbb{M}_c$, and we formally define the merging step as selecting the MDL-optimal model within $\mathbb{M}_c$, i.e.,

$$\widetilde{S}^*_{merge} = \arg\min_{\widetilde{S} \in \mathbb{M}_c} L(x^n, \widetilde{S}) = \arg\min_{\widetilde{S} \in \mathbb{M}_c} (L(\widetilde{S}) + L(x^n | \widetilde{S})). \tag{5.4}$$

Figure 5.2 shows an illustrative example of the partitioning and merging process.

5.4 Calculating the code length

We now discuss the details of the code length (in bits) needed to encode the data and the model.

We first show the calculation of code length of data given a histogram model, encoded by the normalized maximum likelihood (NML) code (Grünwald 2007; Grünwald and Roos 2019). Specifically, we show that the *parametric complexity* term in the code length is independent of data dimensionality, which is an

important observation that makes it feasible to compute the NML code length.

Next, we discuss in detail the difficulties of encoding all possible models $\widetilde{S} \in \mathbb{M}$ if we would want to directly optimize over the full model class $\mathbb{M}$ using Equation (5.2), which motivates our (more pragmatic) solution of dividing the model selection task into two separate steps.

Finally, we discuss the calculation of the code length of a model in the partitioning and merging step respectively, i.e., $L(C_{S'})$ and $L(\widetilde{S})$ of Equations (5.3) and (5.4).

5.4.1 Code length of the data

Extending the work that was previously done for the one-dimensional case (Kontkanen and Myllymäki 2007b), we use the same code—i.e., the *Normalized Maximum Likelihood* (NML) code—to encode the two-dimensional data. This code has the desirable property that it is theoretically optimal because it has minimax regret. The code length of the NML code consists of two terms, namely the maximum likelihood and the parametric complexity (also referred to as *regret*), and is given by

$$L(x^n|\widetilde{S}) = -\log\left(\frac{P(x^n|\widetilde{S}_{\hat{\vec{f}}(x^n)})}{\mathrm{COMP}(n,\widetilde{S})}\right), \tag{5.5}$$

where $P(x^n|\widetilde{S}_{\hat{\vec{f}}(x^n)})$ is the probability of the data given $\widetilde{S}_{\hat{\vec{f}}(x^n)}$, i.e., the parameters $\vec{f} = (f_1, ..., f_K)$ are estimated by the *maximum likelihood estimator* given dataset x^n, denoted as $\hat{\vec{f}}(x^n) = (\hat{f}_1, \ldots, \hat{f}_K)$. The term $\mathrm{COMP}(n, \widetilde{S})$ is the so-called *parametric complexity*, which is defined as

$$\mathrm{COMP}(n,\widetilde{S}) = \sum_{y^n \in S^n} P(y^n|\widetilde{S}_{\hat{\vec{f}}(y^n)}), \tag{5.6}$$

where $\sum_{y^n \in S^n}$ is the sum over *all possible sequences* y^n within the Cartesian product of sample space S that can be *generated* by the histogram model $\widetilde{S}$, i.e., the order of individual values within vector y^n *does* matter.

We will now first describe the calculation of $P(x^n|\widetilde{S}_{\hat{\vec{f}}(x^n)})$, and then the calculation of $\mathrm{COMP}(n, \widetilde{S})$.

For any single data point $x_i \in x^n$, let $x_i = (x_{i1}, x_{i2})$ denote the pair of values

for its two dimensions. We then have

$$P(x^n|\widetilde{S}_{\hat{\hat{f}}(x^n)}) = \prod_{i=1}^{n} P(x_i|\widetilde{S}_{\hat{\hat{f}}(x^n)}) = \prod_{j=1}^{K} \left(\prod_{x_i \in S_j} P(x_i|\widetilde{S}_{\hat{\hat{f}}(x^n)}) \right), \tag{5.7}$$

as the data points are assumed to be independent. Note that K represents the number of regions of $\widetilde{S}$.

Since we assume our data to have precision ϵ, we can define the probability of the data, also referred to as its *maximum likelihood*, as

$$P(x_i|\widetilde{S}_{\hat{\hat{f}}(x^n)}) = P(X \in [x_{i1} - \frac{\epsilon}{2}, x_{i1} + \frac{\epsilon}{2}] \times [x_{i2} - \frac{\epsilon}{2}, x_{i2} + \frac{\epsilon}{2}] \mid \widetilde{S}_{\hat{\hat{f}}(x^n)}) = \hat{f}_j \epsilon^2. \tag{5.8}$$

The maximum likelihood estimator for the histogram model (Scott 2015) is

$$\hat{f}_j = \frac{h_j}{n\,|S_j|}, \forall j, \tag{5.9}$$

where h_j is the number of data points within S_j, and $|S_j|$ is the area of S_j. Thus, following Equations (5.7),(5.8), and (5.9),

$$P(x^n|\widetilde{S}_{\hat{\hat{f}}(x^n)}) = \prod_{j=1}^{K} (\hat{f}_j\,\epsilon^2)^{h_j} = \prod_{j=1}^{K} (\frac{h_j\,\epsilon^2}{n\,|S_j|})^{h_j}. \tag{5.10}$$

Next, we describe the calculation of COMP$(n, \widetilde{S})$. Although it may be surprising at first glance, we show that

Proposition 3. *The parametric complexity COMP$(n, \widetilde{S})$ of a histogram model is a function of sample size n and the number of bins K. Given n and K, COMP$(n, \widetilde{S})$ is independent of the dimensionality of the data.*

We leave the formal proof to Appendix A, but the proposition is based on the following important observations. First, as Kontkanen and Myllymäki (2007b) proved, COMP$(n, \widetilde{S})$ is a function of sample size n and the number of bins K for one-dimensional histograms. The remaining question is whether this holds for two (and higher) dimensional histograms as well. Observe that the maximum likelihood given a two-dimensional histogram model for any data is a function of h_j and $|S_j|/\epsilon^2$, respectively representing the number of data points in each region, and the total number of *possible positions* of data points in each region,

which are both some form of "counts" and hence are "dimensionality free". Finally, $\mathrm{COMP}(n, \widetilde{S})$, as defined in Equation (5.6), is just the sum of maximum likelihoods. Based on these observation, it is trivial to prove that $\mathrm{COMP}(n, \widetilde{S})$ has the same form for one- and multi-dimensional histograms.

Therefore, for both one- and multi-dimensional histogram models, we can denote $\mathrm{COMP}(n, \widetilde{S})$ as $\mathrm{COMP}(n, K)$, and as shown by Kontkanen and Myllymäki (2007b),

$$\mathrm{COMP}(n, K) = \sum_{h_1+\ldots+h_K=n} \frac{n!}{h_1!\ldots h_K!} \prod_{j=1}^{K} (\frac{h_j}{n})^{h_j}, \tag{5.11}$$

which turns out to be the same as the parametric complexity for the multinomial model (Kontkanen and Myllymäki 2007a). We can calculate $\mathrm{COMP}(n, K)$ in linear time (Kontkanen and Myllymäki 2007a) by means of the following recursive formula:

$$\mathrm{COMP}(n, K) = \mathrm{COMP}(n, K-1) + \frac{n}{K-2}\mathrm{COMP}(n, K-2). \tag{5.12}$$

5.4.2 Code length of the model

We first discuss in detail why properly encoding all models in the model class is difficult, and then describe the code length of model in the partitioning step and the merging step respectively.

Encoding all models in the model class is difficult

According to Kraft's inequality, encoding all models in the model class is equivalent to assigning a prior probability distribution to all models (Grünwald 2007). This prior distribution should reflect the model complexities (Grünwald 2004), especially when there exists some hierarchical structure in the model class. For models with similar model complexity, the prior distribution should be non-informative. Particularly, a common practice is to divide the model class into sub-classes according to the hierarchical structure, and then assign the prior distribution to each model by first assigning some prior to all the sub-classes and then assigning a uniform prior to all models within each sub-class.

The model class of all histogram models (i.e., all partitions of S) has an apparent hierarchical structure with respect to model complexity. That is, the model

class could be divided into sub-classes based on a combination of two factors: 1) the number of regions, and 2) the number of line segments composing the inner boundaries. Nevertheless, it is extremely challenging to assign a proper (or even an intuitively "natural") prior distribution based on this complexity hierarchical structure, because of the following two reasons.

First, it is difficult to specify a *joint* prior distribution on the number of regions and the number of line segments, as they are dependent on each other, though specifying marginal prior distributions for each of the factors may be feasible.

Second, given the number of regions, denoted by K, and the number of line segments composing the inner boundaries, denoted by T, it is challenging to count the number of models with K regions and T line segments. Hence, the prior probability of each model (with the uniform prior) within this sub-class is also difficult to obtain. On one hand, there is no analytical formula to obtain such count (to the best of our knowledge). On the other hand, to count this number algorithmically, we would first need to decide how many line segments each region has, i.e., to assign positive integers to $\{T_1, \ldots, T_K\}$ such that $T_1 + \ldots + T_K = T$. The number of possible values of $\{T_1, \ldots, T_K\}$ grows exponentially as K increases. Further, we would need to decide where to put these line segments to form K regions. The number of possible positions is enormous if ϵ is reasonably small. Finally, we would need to go over all individual cases to check for repeated counting for T, since regions can share line segments, which makes the counting computationally infeasible.

Code length of the model in the partitioning and merging steps

As properly encoding all possible models within $\mathbb{M}$ turns out to be too difficult, we now discuss how to calculate the code length of the model separately for the partitioning and merging step.

Partitioning. For a region $S' \in \widetilde{S}$, assume that there are E candidate positions for cut lines, either horizontally or vertically. To encode the set of cut lines, we first encode the number of regions $K \in \{1, \ldots, K_{max}\}$, where K_{max} is predetermined. We assign a uniform prior to K, and thus the code length needed to encode K becomes a constant, which has no effect on the result of the partitioning step. Given K, we then encode the positions of $(K-1)$ cut lines, with again a uniform

prior to all possible sets of $(K-1)$ cut lines. The code length needed in bits is

$$L(C_{S'}) = \log \binom{E}{K-1} \tag{5.13}$$

Merging. Next we discuss the code length of encoding all models in the constrained model class $\mathbb{M}_c$, which contains all possible models that can be obtained by merging neighboring regions of the partition after the partitioning step.

We argue that we should have a non-informative prior on $\mathbb{M}_c$. First, as discussed before, it is challenging to specify a joint prior to both the number of line segments and the number of regions. Second, if neighboring regions are merged, the partition of the sample space tends to have fewer regions but more geometric complexity. Hence, there exists no obvious ways to compare model complexities, even in an intuitive manner.

Thus, we treat the model complexities to be roughly equivalent and we assign a uniform prior to all models in $\mathbb{M}_c$. As a result, the code length of all models within $\mathbb{M}_c$ is a constant and has no effect on the result of the merging step. In other words, we only consider the code length of data in the merging step.

5.5 Revisiting MDL histograms for one-dimensional data

In this section, we elaborate the link of our work to the MDL-based histograms to one-dimensional data.

We first show that searching for the best cut lines on one certain dimension of given two-dimensional data is equivalent to searching for the best cut points for the corresponding one-dimensional data. We then review the algorithm for inferring MDL histograms for one-dimensional data as proposed by Kontkanen and Myllymäki (2007b), and describe how we improve it both theoretically and practically.

Notation and relation to our problem. To be able to distinguish it from two-dimensional data x^n, we denote one-dimensional data as $z^n = (z_1, ..., z_n)$, with precision equal to ϵ. Further, we define the sample space of z^n as $[\min z^n, \max z^n]$.

We define the one-dimensional histogram model with K bins as a set of

cut points, denoted as $C^K = \{C_0 = \min z^n, C_1, ..., C_K = \max z^n\} \subseteq C_a$, with $K \in \{0, 1, \ldots, K_{max}\}$, where K_{max} is pre-determined and C_a is defined as

$$C_a = \{\min z^n, \min z^n + \epsilon, ..., \min z^n + E \cdot \epsilon, \max z^n\}, \tag{5.14}$$

with $E = \lfloor \frac{\max z^n - \min z^n}{\epsilon} \rfloor$. Note that we assume all subintervals to be closed on the left and open on the right, except that the rightmost subinterval is closed on both sides.

The code length needed to encode the model C^K is

$$L(C^K) = \log \binom{E}{K-1}, \tag{5.15}$$

which is the same as Equation (5.13). Further, based on the calculation of maximum likelihood given any histogram model (Section 5.4.1) and Proposition 3, the code length needed to encode z^n given C^K by the NML code is

$$\begin{aligned} L(z^n|C^K) &= -\log P(z^n|C^K) + \log \mathrm{COMP}(n, K) \\ &= -\log \prod_{j=1}^{K} \left(\frac{h_j\, \epsilon}{n\,(C_{j+1} - C_j)}\right)^{h_j} + \log \mathrm{COMP}(n, K). \end{aligned} \tag{5.16}$$

If we compare $L(z^n|C^K)$ and $L(C^K)$ with Equations (5.10) and (5.13), we can see that the definition of the two-dimensional MDL-optimal cut lines and the one-dimensional MDL-optimal cut points only differ by a constant. Thus, given a two-dimensional dataset $x^n = \{(x_{11}, x_{21}), \ldots, (x_{1n}, x_{2n})\}$, the optimization task of searching for the MDL-optimal vertical (or horizontal) cut lines is equivalent to the task of searching for the MDL-optimal one-dimensional cut points based on one-dimensional dataset $z^n = \{x_{11}, \ldots, x_{1n}\}$ (or $z^n = \{x_{21}, \ldots, x_{2n}\}$). That is, z^n is the projection of x^n on the x- or y-axis.

In other words, the algorithm for constructing MDL-based one-dimensional histograms proposed by Kontkanen and Myllymäki (2007b) can be directly applied to the partitioning step of our model selection task. We now briefly review this algorithm and show how we improve it both theoretically and practically.

Improved one-dimensional MDL-based histograms. We improve the one-dimensional algorithm proposed by Kontkanen and Myllymäki (2007b) in two ways. First, in their previous work, the candidate cut points, denoted as C'_a, are

chosen based on the data z^n, i.e., $C'_a = \bigcup_{i=1}^{n} \{z_i \pm \epsilon\}$, and hence the code length of model is calculated *dependent* on given dataset, i.e., $L(C^K|z^n)$ is calculated instead of $L(C^K)$, which is theoretically sub-optimal, because generally

$$L(z^n, C^K) = L(z^n|C^K) + L(C^K) \neq L(z^n|C^K) + L(C^K|z^n). \tag{5.17}$$

In practice, this will cause significantly worse results when the sample size is very small. In such cases, the size of the set C'_a will be very small, and hence the code length of model will be significantly underestimated, leading to serious overfitting. We fix this problem by encoding the model independent of the data, as defined by Equations (5.14) and (5.15).

Further, we show that we do not need to consider *all* candidate cut points within C_a, but just those cut points with a data point near it from left or right, without other cut points in between. That is, we have the following.

Proposition 4. *For any two cut points $C_i, C_k \in C_a$, suppose $C_i < C_k$ and no data points exist in the interval $[C_i, C_k]$, then any cut point $C_j \in [C_i, C_k]$ would not be in the MDL-optimal set of cut points, i.e., we can skip all such C_j during the search process.*

This reduces the search space to a subset of C_a, and hence reduces the computational requirements. We include the proof in Appendix B.

Finally, we simplify the recursion formula for the dynamic programming proposed by Kontkanen and Myllymäki (2007b) in their original paper, which significantly reduces empirical computation time.

Dynamic programming algorithm. Kontkanen and Myllymäki (2007b) derived the recursion formula based on the total code length $L(z^n, C^K)$, i.e.,

$$\begin{aligned} L(z^n, C^K) &= L(z^n|C^K) + L(C^K) \\ &= -\log(P(z^n|C^K) + \log \mathrm{COMP}(n, K) + \log \binom{E}{K-1}. \end{aligned} \tag{5.18}$$

We show that we can simplify the recursion by only including the probability of the data, i.e., $P(z^n|C^K)$, instead of $L(z^n, C^K)$. Observe that when the number of bins K is fixed, $L(C^K)$ and $\mathrm{COMP}(n, K)$ become constant. Then, for fixed K, minimizing $L(z^n, C^K)$ is equivalent to minimizing $\{-\log(P(z^n|C^K)\}$, i.e., maximizing the likelihood.

Therefore, minimizing $L(z^n, C^K)$, for all $K \in \{1, \ldots, K_{max}\}$, can be done in two steps: 1) find the maximum likelihood cut points with fixed K, denoted as $\hat{C}^K$, for each K, using the following dynamic algorithm; and 2) calculate $L(z^n|\hat{C}^K)$ for each K, and find the $\hat{K} \in \{1, \ldots, K_{max}\}$ that minimizes $L(z^n, \hat{C}^K)$. Then,

$$\hat{C}^{\hat{K}} = \arg \min_{K \in \{1,\ldots,K_{max}\}, C^K \in C_a} L(z^n, C^K). \tag{5.19}$$

Now we describe the dynamic programming algorithm for finding $\hat{C}^K$ for each $K \in \{1, \ldots, K_{max}\}$. The (log) probability of z^n given any cut points is

$$\begin{aligned}
\log P(z^n|C^K) &= \sum_{i=1}^{n} \log P(z_i|C^K) \\
&= \sum_{j=1}^{K} \sum_{z_i \in [C_{j-1}, C_j)} \log P(z_i|C^K) \\
&= \sum_{j=1}^{K-1} \sum_{z_i \in [C_{j-1}, C_j)} \log P(z_i|\{C^K \setminus C_K\}) + \sum_{z_i \in [C_{K-1}, C_K]} \log P(z_i|C_K) \\
&= \log P(z^n_{C_{K-1}}|\{C^K \setminus C_K\}) + \sum_{z_i \in [C_{K-1}, C_K]} \log P(z_i|C_K)
\end{aligned} \tag{5.20}$$

where $z^n_{C_K}$ is a constrained dataset containing all data points smaller than C_K, i.e.,

$$z^n_{C_{K-1}} = \{z \in z^n | z < C_{K-1}\}. \tag{5.21}$$

Given the previous, the recursion formula is given by

$$\begin{aligned}
\max_{C^K \subseteq C_a} \log P(z^n|C^K) = \max_{C_K \in C_a} [&\max_{\{C^K \setminus C_K\} \subseteq C_a} \log P(z^n_{C_{K-1}}|\{C^K \setminus C_K\}) \\
&+ \sum_{z_i \in [C_{K-1}, C_K]} \log P(z_i|C_K)]
\end{aligned} \tag{5.22}$$

and hence a dynamic programming algorithm can be applied to search all $K \in \{1, \ldots, K_{max}\}$. In practice, K_{max} is pre-determined, and larger K_{max} should be investigated if $\hat{K} = K_{max}$.

The disadvantage of implementing the dynamic programming algorithm based on $L(z^n, C^K)$, $\forall K \in \{1, \ldots, K_{max}\}$, is that we would need to calculate the para-

metric complexity COMP(·) for every constrained dataset. Our improved version, in contrast, involves only $P(z^n|C^K)$, and thus we only need to calculate COMP(·) for the full dataset z^n when calculating $L(z^n, \hat{C}^K)$ for each K, which will be much faster in practice.

The essential component of the dynamic programming algorithm is to construct the constrained dataset $z^n_{C_{K-1}}$, $\forall K \in \{1, \ldots, K_{max}\}$. These constrained datasets are easy to construct in the one-dimensional case with a natural order, but infeasible for two or higher dimensional cases. Hence we resort to the heuristic algorithm presented in the next section.

5.6 The PALM Algorithm for Partitioning and Merging

We propose a heuristic algorithm named PALM, which infers histogram models for two-dimensional data by decomposing the overall model selection problem into two steps: 1) partition space S alternately based on the discretization result from previous iterations until it stops automatically; and then 2) merge neighboring regions if their densities are very similar. Both steps use the MDL principle as the decision criterion, with the code length defined in Section 5.4.

The PALM algorithm is given in Algorithm 5. Specifically, we first initiate $\widetilde{S} = \{S\}$ and choose the starting direction (line 1); then we iterate over all regions in $\widetilde{S}$ and partition each of them by searching for the MDL-optimal cut lines in the chosen direction (lines 3–5), and update $\widetilde{S}$ accordingly (lines 8–10); then, we keep iterating until $\widetilde{S}$ is no longer updated (lines 2 and 6–7), which completes the partitioning step.

Next, the merging step searches, in a greedy manner, for the MDL-optimal partition of S over all possible partitions that can be obtained by merging any two neighboring regions of the partition that is obtained in the partitioning step. That is, we list all the neighboring pairs of regions in $\widetilde{S}$, i.e., two regions that share part of their boundaries (line 15); then, we merge the pair that compresses the data most (or equivalently, decreases the MDL score most) and update the neighboring pairs list (lines 21–23); finally, we stop the merging step when no better compression can be obtained by merging any neighboring two pairs in $\widetilde{S}$ (lines 19–20).

Algorithm complexity. We now discuss the worst-case algorithm complexity for the partitioning and merging step respectively, and we will show the empirical runtime in Section 5.7.6.

For the first iteration of the partitioning step (i.e., when $\widetilde{S} = \{S\}$), the algorithm has a complexity of $\mathcal{O}(K_{max}E^2)$, the same as the one-dimensional case (Kontkanen and Myllymäki 2007b), where E is the number of possible locations for vertical (or horizontal) lines within the whole sample space S, based on the fixed grid with granularity ϵ. The second iteration has a worst-case time complexity of $\mathcal{O}(K_{max}^2E^2)$ when the first iteration produces exactly K_{max} regions. Following this line, the worst-case time complexity of the partitioning step is $\mathcal{O}(K_{max}^IE^2)$, where I is the number of iterations.

As for the merging step, the time complexity is bounded by K_pK_0, where K_0 denotes the number of regions after the partitioning step, and K_p denotes the number of neighboring pairs. That is, we can merge at most $(K_p - 1)$ times, and each merging requires going over all the neighboring pairs.

Although the worst-case time cost for the partitioning step is exponential, and K_0 and K_p could be large in practice, we will show in Section 5.7.6 that the empirical runtime may scale much better than exponential growth.

Choosing the hyper-parameter settings. We now briefly discuss how to choose ϵ and K_{max} in practice. First, we should set ϵ to be the same as the precision of the data by default; data is always recorded up to a precision in practice. Further, when prior knowledge exists given a specific task, ϵ may be larger than the recording precision, because the domain expert or data analyst may decide that the data is only meaningful up to a more coarse precision.

Second, theoretically we should set K_{max} to be sufficiently large, and hence in practice K_{max} is a "budget" rather than a hyper-parameter like the threshold or stopping criterion in other discretization methods (e.g., Nguyen 2014, Kerber 1992). That is, unlike these hyper-parameters, which can be either too large or too small and hence need to be carefully tuned, K_{max} can be simply picked to be as large as possible.

This makes our method practically hyper-parameter-free, in the sense that—given the guidelines above—no tedious hyper-parameter tuning should be necessary to obtain the best possible results.

5.7 Experiments

In this section, we investigate the performance of PALM using synthetic data, after which we will apply it to real-world data in the next section. We show that PALM can construct two-dimensional histograms that are adaptive to both local densities and sample size of the data.

We start off by defining the "loss" that we use for quantifying the quality of the "learned" partitions. We then present experiment results on a wide variety of synthetic data. Although our algorithm relies on heuristics, we show that it has a number of desirable properties, as follows.

First, if the data is generated by a histogram model within our model class $\mathbb{M}$, PALM is able to identify the "true" histogram given a large enough sample size. The results are discussed in Section 5.7.2.

Second, in Section 5.7.3 we show that PALM has the flexibility to approximate histogram models outside the model class $\mathbb{M}$. Specifically, we study the behavior of PALM on a dataset generated as follows: we set the sample space $S = [0, 1] \times [0, 1]$, and partition it by a sine curve; we then generate data points uniformly distributed above and below the sine curve, with different densities.

Third, we study the performance of PALM on data generated by two dimensional Gaussian distributions in Section 5.7.4. We show that it inherits the property of the one-dimensional MDL histogram method (Kontkanen and Myllymäki 2007b) that the bin sizes of the histogram are self-adaptive: the two-dimensional bin sizes become smaller locally where the probability density changes more rapidly.

Fourth, in Section 5.7.5 we compare PALM with the IPD algorithm (Nguyen et al. 2014), using a simple synthetic dataset that is almost identical to what has been used to study the performance of IPD (Nguyen et al. 2014).

Note that we always set $\epsilon = 0.001$, and all simulations are repeated 500 times unless specified otherwise[4]. The initial partitioning direction is fixed as vertical, to make the visualizations of the inferred partitions comparable.

[4]The code is available at: https://github.com/ylincen/PALM

5.7.1 Measuring the difference between two-dimensional histograms

As PALM produces a histogram model and can be regarded as a density estimation method, one of the most intuitive "loss" functions is the *Mean Integrated Squared Error (MISE)* (Scott 2015), defined as

$$\mathrm{MISE}(\hat{f}) = \mathbb{E}[\int_S (f(x) - \hat{f}(x))^2 dx], \tag{5.23}$$

where f is the true probability density and $\hat{f}$ is the histogram model density estimator. We report the empirical MISE by calculating the integral numerically, and estimating $\mathbb{E}[\cdot]$ by the empirical mean of results over all repetitions of the simulation.

As MISE cannot indicate whether there are more "bins" than necessary, we also propose two "loss" functions that directly quantify the distances between the inner boundaries of the learned and true partitions of a sample space S. We first break up the line segments of the inner boundaries into *pixels* with a precision set to $0.01 = 10\epsilon$ (merely to speed up the calculation). Then we introduce two loss functions based on the idea of *Hausdorff distance*, considering *false positives* and *false negatives* respectively. Namely, we propose L_{learn}, based on the learned partition, and L_{true}, based on the true partition:

$$L_{\text{learn}} = \sum_{p \in P} min_{q \in Q} ||p - q||^2; L_{\text{true}} = \sum_{q \in Q} min_{p \in P} ||p - q||^2 \tag{5.24}$$

where $||\cdot||$ denotes the Euclidean distance and P and Q are the sets of *pixels* on the line segments of the learned partition and the true partition, respectively.

The intuition for L_{learn} is that, for a given pixel on a line segment of the learned partition, we find on the line segments of the true partition the pixel closest to it, and measure their distance; for L_{true} it is the other way around. Thus, if L_{learn} is large, the learned partition must have unnecessary extra line segments, whereas if L_{true} is large, the learned partition fails to identify part of the line segments that actually exist.

5.7.2 Revealing ground truth two-dimensional histograms

We describe the settings for simulating the data and then our experiment results, to empirically show that our algorithm can identify the "true" histogram model if the data is generated by it.

Experiment settings. To randomly generate the "true" partitions, we use a generative process that is very similar to the search process of our algorithm: we fix a rectangular region, $S = [0, 1] \times [0, 1]$, randomly generate vertical cut lines to split it into K_1 regions, and randomly generate horizontal cut lines to split each of the K_1 regions into $(K_{21}, ..., K_{2,K_1})$ regions respectively. Then, for each pair of neighboring regions, we merge them with a pre-determined probability P_{merge}.

We set these hyper-parameters as follows:

$$K_1 = K_{21} = K_{22} = ... = K_{2,K_1} = 5; P_{merge} = 0.4; \epsilon = 0.001. \tag{5.25}$$

With these hyper-parameters, our generative process is able to generate a diverse subset of $\mathbb{M}$, as P_{merge} is chosen delicately to be not too small or too large. Figure 5.3 shows four random examples of the true partitions and learned partitions. These learned partitions are produced with the sample size set as 10 000.

After the partition is fixed, we generate "true" density parameters for the histogram model using a uniform distribution, i.e.,

$$f_j \sim \text{Uniform}(0, 1), \forall i = 1, 2, ..., K; \tag{5.26}$$

and normalize them such that $\sum_{j=1}^{K} f_j |S_j| = 1$, where K is the number of regions in total and $|S_j|$ is the geometric area of S_j. Note that we do not force the f_j to be different from each other.

Results. Figure 5.4 shows that MISE is already small for small sample size, and converges to almost 0 as the sample size increases. We also show, in Figure 5.5, that L_{learn} and L_{true} converge to almost zero except for some outliers.

The outliers of L_{learn} are due to sampling variance when generating data points, the number of which decreases significantly as the sample size grows.

The outliers of L_{true}, however, are due to the random generation of the density parameters f_j. As we do not force all f_j's to be different, they could accidentally turn out to be very similar. In that case, some of the "true" inner boundaries are

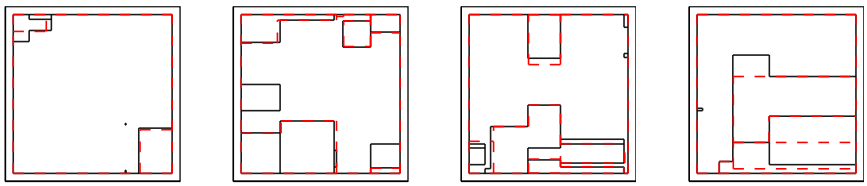

Figure 5.3: Random examples of true (black solid) and learned partitions (red dashed) of the experiment in Section 5.7.2, mainly to show that our experiment settings can produce very flexible partitions of $[0, 1] \times [0, 1]$. Note that the sample size is set as 10 000, which is *not* enough for MISE (Equation 5.23) to converge to almost 0, but the learned partitions by PALM already look promising: it can partly identify the true partitions.

actually unnecessary, and our algorithm will "fail" to discover them. Table 5.1 confirms that this is the cause of outliers when the sample size is large ($\geq 1e5$): when PALM fails to identify part of the "true" inner boundaries and $L_{true} > 1$, the learned histogram still estimates the density very accurately. The only explanation is then that some regions of the true partition accidentally have very similar f_j's.

Moreover, when the sample size is moderate, e.g., 5000, L_{learn} is already small, meaning that PALM can partly identify the true partition quite precisely, and rarely produces unnecessary extra regions. As the sample size increases, L_{true} decreases, indicating that the learned partition becomes more and more complex; i.e., it is shown that the model selection process is self-adaptive to sample size.

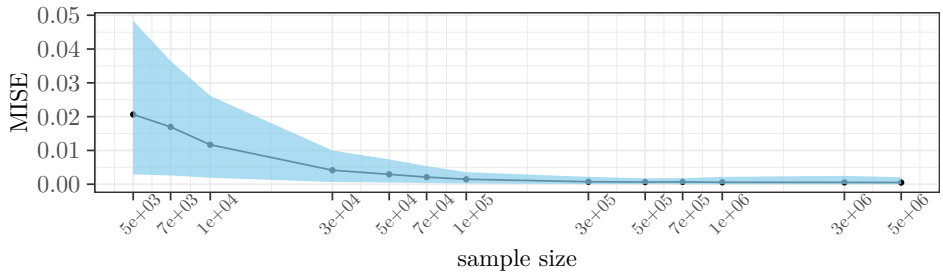

Figure 5.4: Sample size vs MISE: MISE converges to almost 0 when the sample size becomes larger than 100 000. The range between the 5th and 95th percentiles is shown in blue.

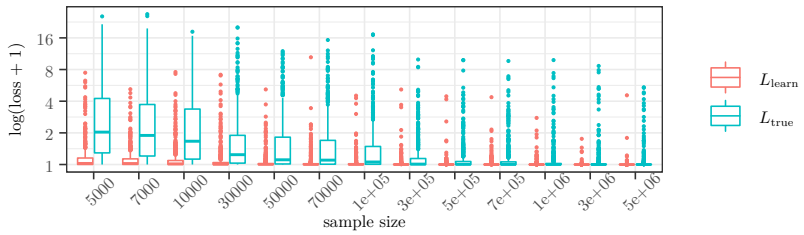

Figure 5.5: Boxplots showing the sample size versus L_{learn} and L_{true} as defined in Equation (5.24). Note that the y-axis has a logarithmic scale. L_{learn} is generally much smaller than L_{true}, meaning that it is very rare that PALM produces unnecessary extra regions. When the sample size is large enough for MISE to converge ($n \geq 1e5$), outliers of L_{true} are due to sampling variance when generating the true parameters f_j defined in Equation (5.26), see Table 5.1; the number of outliers for L_{learn} decreases rapidly as the sample size becomes larger, as they are due to sampling variance when generating the data.

5.7.3 Approximating histogram models outside model class $\mathbb{M}$

We now investigate the case where the true model is not within model class $\mathbb{M}$, while the data is still generated uniformly within each region.

We show that, although the model class $\mathbb{M}$ is based on a grid, it is indeed flexible and expressive: in practice, the learned partitions can approximate true partitions outside $\mathbb{M}$, and the approximation becomes more and more accurate as the sample size increases.

Experiment settings. As an illustrative example, we partition $S = [0,1] \times [0,1]$ by several sine curves, defined as

$$g(x) = \frac{1}{4} \sin 2m\pi x + \frac{1}{2} \tag{5.27}$$

and where m is a hyper-parameter.

We randomly generate data from a uniform distribution above and under the sine curve, and we set the probability density above $g(x)$ to be twice as large as below $g(x)$, i.e., we uniformly sample $\frac{2}{3}n$ data points above $g(x)$, and $\frac{1}{3}n$ data points below $g(x)$, where n is the total sample size.

Results. We empirically show that the learned partitions approximate the sine curves quite precisely, though occasionally a few extra undesired regions are produced. Figure 5.6 (left) shows the learned partitions on single simulated datasets, with $m \in \{2, 4, 6\}$ to control the degree of oscillation, and sample size $n \in \{1e4, 1e5, 1e6\}$. We see that, as the sample size grows, our approximation

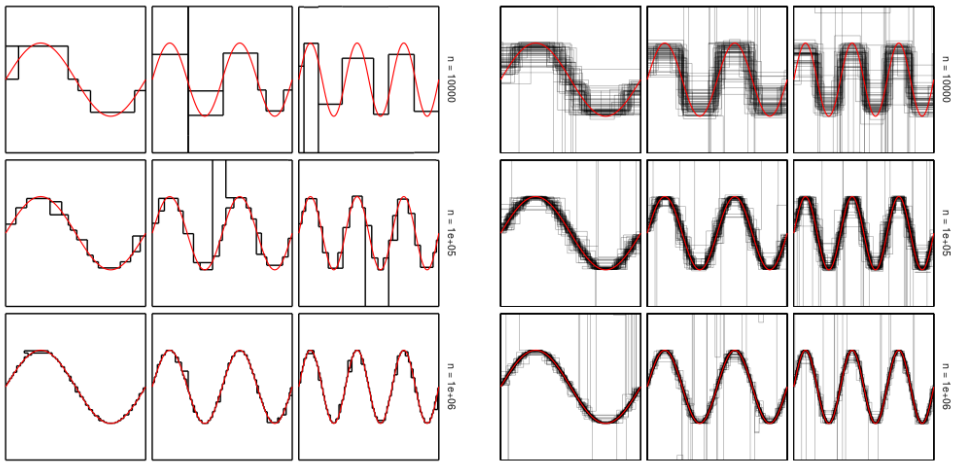

Figure 5.6: (Left) Sine curve defined in Equation (5.27) (red), with $m \in \{2, 4, 6\}$ from left to right on each row, and the learned partition by PALM (black). Data is randomly generated by uniforms distribution above and below the sine curve, within $S = [0, 1] \times [0, 1]$. Densities above and below the since curve are 2:1. From top to bottom, the sample sizes of the simulated data are $n \in \{1e4, 1e5, 1e6\}$. (Right) 50 partition results of 50 different simulated datasets are plotted *together*. It shows that PALM is not guaranteed to be absolutely stable, as it occasionally produces undesired extra line segments, but the line segments of the learned partitions mostly gather around the true sine curve.

becomes more and more accurate.

However, since our algorithm is greedy in nature, there is no guarantee to find the partition with the global minimum score. In practice, PALM will occasionally produce undesired, extra line segments. Thus, to investigate the stability of the learned partitions, we repeat the simulation 50 times for each combination of m and n, and plot *all* partition results in one single plot in Figure 5.6 (right).

Figure 5.6 (right) shows that the undesired extra regions are produced more frequently as m increases, but seems independent of sample size n. However, as sample size increases, the learned partitions become indeed more stable as they gather around the sine curves more closely.

5.7.4 Gaussian random variables

In this section, we show the performance of our algorithm on data generated from a two-dimensional Gaussian distribution. Specifically, we consider two of them, i.e., $N[\binom{0}{0}, \left(\begin{smallmatrix}1 & 0\\ 0 & 1\end{smallmatrix}\right)]$ and $N[\binom{0}{0}, \left(\begin{smallmatrix}1 & 0.5\\ 0.5 & 1\end{smallmatrix}\right)]$, of which the key difference is whether

the two dimensions are independent. We assume $S = [-5, 5] \times [-5, 5]$, as the true Gaussian density outside such S is negligible.

Figure 5.8 shows the learned partitions as well as the learned empirical densities from a random simulated dataset with different sample sizes, $n \in \{5\,000, 10\,000, 50\,000\}$. Note that bin size is self-adaptive with regard to sample size and local structure of the probability density. We also mention that the empirical runtime for a single dataset generated by such Gaussian distributions is at most a few minutes, for all $n \leq 50\,000$.

To quantify the quality of the learned partitions by PALM, we compare the MISE of PALM to the MISE of fixed equally-spaced grid partitions with different granularities. Figure 5.7 shows the mean and standard deviation of MISE for different cases, and we conclude that, to achieve roughly the same level of MISE with a fixed grid, a fixed grid needs to have five times as many regions as a partition learned by PALM.

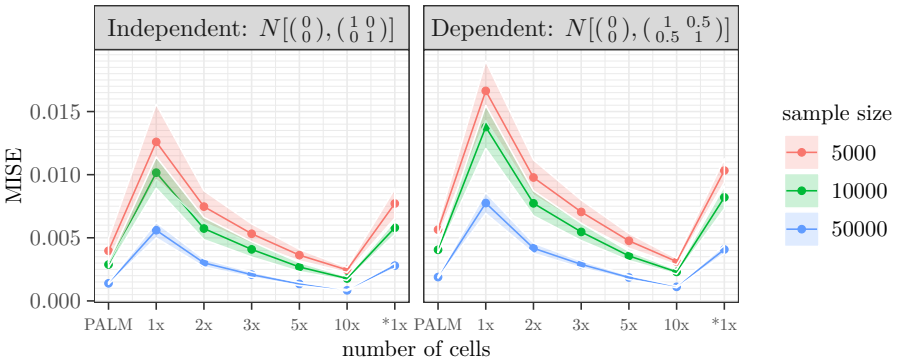

Figure 5.7: For data generated from a two-dimensional Gaussian distribution, described in Section 5.7.4, the mean and standard deviation of MISE is calculated for different partitions: (from left to right) PALM, fixed grid with the same number of regions as PALM (denoted as '1x'), fixed grid with two times number of regions as PALM (denoted as '2x'), ..., and fixed grid with the same number of regions before the merging step of PALM (denoted as '*1x'). We assume $S = [-5, 5] \times [-5, 5]$, as the true Gaussian density outside S is negligible.

5.7.5 Comparison with IPD

Since—to the best of our knowledge—no existing discretization method can produce partitions as expressive as PALM, it seems not so meaningful to compare

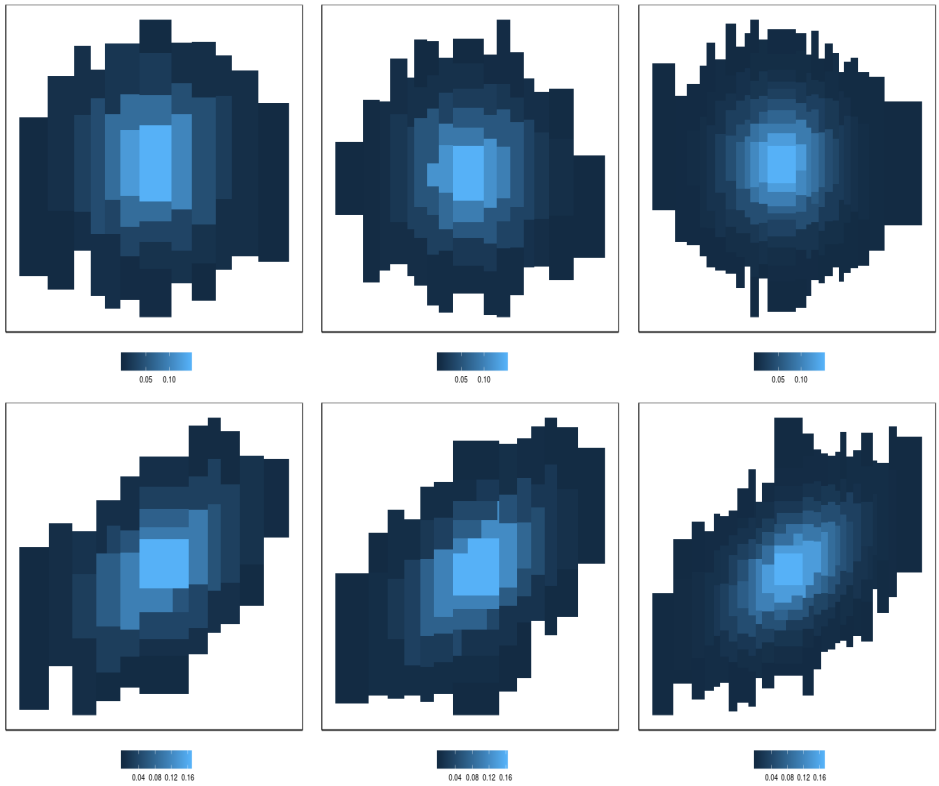

Figure 5.8: Learned partitions and estimated densities by PALM. The data is generated from two-dimensional Gaussian distributions, with sample size $n \in \{5\,000, 10\,000, 50\,000\}$, from left to right. The top and bottom row is respectively generated from independent and dependent two-dimensional Gaussian distributions.

with any existing algorithm. However, we do include a comparison with the IPD algorithm (Nguyen et al. 2014), mainly to show that our algorithm not only can produce more flexible partitions by definition, but also beats this state-of-the-art algorithm on a "simple" task, i.e., when the "true" partition is an adaptive two-dimensional grid.

We use simple synthetic data, similar to one of the synthetic datasets used to study the performance of IPD (Nguyen et al. 2014). The data is generated to be uniform within four regions in $S = [0, 1] \times [0, 1]$. These regions are produced by partitioning S by one vertical line $x = V_x$ and one horizontal line $y = H_y$, where $V_x, H_y \sim \text{Uniform}(0, 1)$. The number of data points within each region is equal.

We compare the loss, as defined in Equation (5.24), and we show in Figure 5.9 that 1) PALM has better performance on small datasets, and 2) as the sample size gets larger, PALM converges but IPD partitions S into more and more regions, as can be witnessed from an increasing L_{true}.

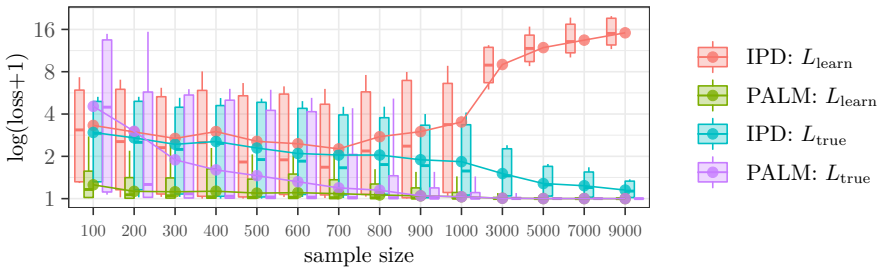

Figure 5.9: Comparison of PALM and IPD, using the box-plot and the mean of L_{learn} and L_{true}, as defined in Equation (5.24). PALM not only performs better when the sample size is small, but also converges as the sample size increases, while IPD does not converge.

5.7.6 Empirical runtime

We next discuss the empirical runtime with respect to K_{max}, the maximum number of bins to search, and E, the number of candidate cut points.

Specifically, we use two-dimensional datasets simulated from independent standard Gaussian distributions to examine the relationship between K_{max} and runtime, with fixed sample size equal to 500 and $\epsilon = 0.001$. The results are illustrated in Figure 5.10, showing that the runtime increases linearly with K_{max}. Further, to investigate the relationship between E and the runtime, we again simulate from two-dimensional Gaussian distributions with different variance σ^2 to control the E [5]. We fixed the sample size to be 1 000 and $\epsilon = 0.001$. The results show that, the runtime grows quadratically with E (as shown by the blue dashed curve), but the second-order coefficient is quite small (as it is very close to the red dashed line with a linear trend). We report the runtime based on 500 repetitions.

[5]For reproducibility, we first simulate 10 000 data points from $N[\binom{0}{0}, \begin{pmatrix} 0 & \sigma^2 \\ \sigma^2 & 0 \end{pmatrix}]$, where $\sigma^2 = E\epsilon/2$, where E is the desired number of candidate cut points. Since the corresponding desired data range with E candidate cut points is $[-E\epsilon/2, E\epsilon/2]$, we next remove the data points outside this desired data range, and we finally randomly select 1 000 data points from the remaining data points.

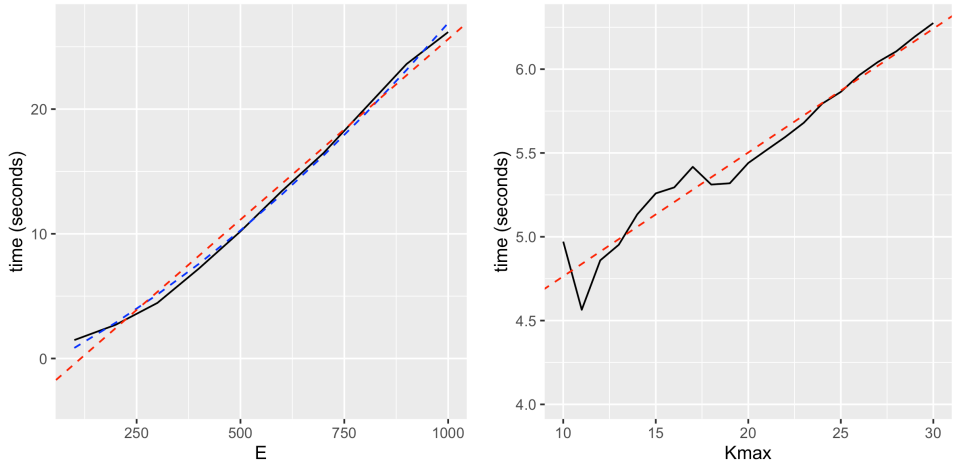

Figure 5.10: Empirical time complexity on simulated two-dimensional Gaussian data, with respect to E, the number of candidate cut points, and the runtime, and K_{max}, the maximum number of bins we search.

5.8 Case study

We now show the results of applying our algorithm to real-world spatial datasets. We start with describing the three datasets we use in Section 5.8.1. Next, we describe our case study tasks in Section 5.8.2. Specifically, we inspect the results by visualizing the histograms, to illustrate that our method can be used as an explanatory data analysis (EDA) tool. We also compare with kernel density estimation (KDE), arguably the most widely used EDA method for spatial datasets, both for the visualizations and the goodness-of-fit on unseen data. In Section 5.8.3, we report our results and show that 1) PALM can produce partitions that characterize more detailed density changes than KDE, and 2) PALM fits better on unseen data (i.e., a test dataset), in the sense that the partition of PALM has larger log-likelihood on the test dataset than KDE. Finally, we report the runtimes and detailed algorithm settings, respectively in Section 5.8.4 and 5.8.5.

5.8.1 Datasets

We consider three diverse real world datasets: locations of Airbnb housing in Amsterdam[6], GPS locations of destinations of DiDi taxi queries in Chengdu, China[7], and GPS recordings of visitors' movement in an amusement park[8].

Visitors movement data in the DinoFun amusement park. All visitors at the amusement park need to carry a device or use a smartphone app to check in at different attractions (e.g., roller coasters). Further, the amusement park is segmented into 100×100 cells (all of them are roughly 5 meters $\times$ 5 meters), and each cell has a sensor which can track the position of each visitor. The device (or the mobile app), together with the sensors, checks the position of the visitor every few seconds and records the position *if the visitor moves to another cell*. Thus, applying PALM on this dataset will reveal the densities of places that people have been in the amusement park. This data has a sample size of 9 078 623, in which every row represents a single position that one individual visitor visited (or passed by), with information like visitor's ID, timestamp, and location.

Amsterdam airbnb locations. This data has a sample size of 20 244, and the location of each house is recorded by its longitude and latitude. Applying PALM on this dataset shows the distribution of Airbnb housing in Amsterdam.

DiDi taxi data in Chengdu. The sample size of the data is 107 573, which collects the longitude and latitude of taxi destinations. Applying PALM on this dataset shows the densities of different regions that people visited by taxi in Chengdu, China.

5.8.2 Case study tasks

Explanatory data analysis and visualizations. We first partition the three two-dimensional datasets by PALM and estimate the densities of all regions, using the full datasets. We visualize the densities by the heat maps in Figure 5.11, and compare the visualizations obtained by two-dimensional kernel density estimation (KDE) (Duong et al. 2007), also with the full dataset. We also include the visualization results of the discretization obtained by IPD (Nguyen et al. 2014) for

[6] http://insideairbnb.com

[7] https://gaia.didichuxing.com

[8] http://vacommunity.org/VAST+Challenge+2015

comparison, although IPD is not primarily designed for two-dimensional datasets. The background of Figure 5.11 are the map of the DinoFun amusement part (provided together with the dataset), and the map of Amsterdam and Chengdu (from Google Maps API and the R package "ggmap" (Kahle and Wickham 2013)).

Comparison of KDE and PALM on the goodness-of-fit. Further, to quantitatively compare how KDE and PALM fit unseen data, and thus generalize to the underlying data distribution, we randomly split each dataset into training and testing set, obtain the PALM and KDE result from the training set, and compare the log-likelihoods on the testing dataset. We repeat the random splitting 100 times[9].

5.8.3 Case study results

We first analyze the result on each dataset respectively, based on which we give our concluding remarks for the case study at the end of this section.

Visitors movement data in the DinoFun amusement park. As shown in Figure 5.11, both KDE and PALM reveal the walking path of the amusement park purely from the movement data (i.e., without knowing the map as additional information). Although KDE seems to capture more density changes, we show that it fits unseen data much worse than PALM, measured by the log-likelihood on the test dataset, shown in Table 5.2. Thus, we conclude that KDE may overfit on this dataset.

Amsterdam airbnb locations. The visualizations of PALM and KDE look generally similar: if we treat red and orange regions in the center as the "dense region", the rigid boundary between the dense region and the rest obtained by PALM approximates well the corresponding curve boundary obtained by KDE. However, note that more density changes are captured within the dense region, and PALM revealed two dense spots outside the central areas that KDE neglects, respectively on the top right and the bottom right of the map[10]. Further, as shown in Table 5.2, the (average) log-likelihood of PALM and KDE on the test set is

[9] To speed up the process, we randomly sampled a subset of the Chengdu taxi dataset that contains 10% of the full sample size; also, for the amusement park movement dataset, we only use the subset of the data that is between 4 hours and 5 hours after the opening of the park, with sample size 713 846. Note that this is only for the comparison of goodness-of-fit but not for the visualizations and empirical runtime evaluation

[10] The top right dense spot is close to the "AMSTERDAM NOORD" text on the map (on the "T"), and bottom right dense spot is near "Amstel Business Park".

almost the same, which indicates that PALM does not overfit on this dataset, i.e., the dense spots revealed by PALM are valid.

DiDi taxi data in Chengdu. The visualizations of PALM and KDE lead to different understandings of this dataset: while KDE reveals several hot clusters of taxi destinations, PALM shows that the density can change drastically within very small range of areas. As PALM fits better on the testing dataset, we conclude that PALM does not overfit but KDE may over-smooth this dataset.

By default the PALM algorithm always starts by splitting the x-axis. Starting by splitting the y-axis leads to slightly different models, and thus somewhat different visualizations, but those differences are so minimal that they can be ignored for practical purposes. That is, the differences mostly appear in sparse regions, with very low densities, where no interesting patterns occur. To demonstrate that the differences are negligible, we compare the log-likelihoods obtained on unseen data when starting splitting on either the x-axis or the y-axis. The log-likelihoods are indeed very similar for both starting directions, as shown in Table 5.2, confirming that the resulting histogram models can only be different in sparse and less important regions; otherwise the log-likelihoods would be substantially different.

Based on the analysis above, we conclude that 1) although PALM partitions the dataset with rigid boundaries, PALM fits the data better than KDE when the datasets have drastic local density changes, such as the Chengdu taxi dataset and the amusement park dataset; 2) when we have smoother two-dimensional data such as the Amsterdam housing dataset, PALM and KDE fit the data equally well; 3) when we look at the visualizations, PALM tends to capture more density changes than KDE, and PALM can reveal dense spots that KDE neglects; in other words, KDE tends to over-smooth the dataset.

Last, we include the visualizations of the IPD discretization in Appendix C, in which we demonstrate that the discretization results obtained by IPD have much coarser granularity. Hence, our discretization results preserve more information from the original datasets.

5.8.4 Empirical runtime

We examine the empirical runtime on these three datasets in Table 5.3 (using the full datasets, without the split of training and testing set). We conclude that

KDE is generally much faster, except on the amusement park dataset, which has a very large sample size but small E.

Note that the runtime of KDE highly depends on the number of evaluation points, the bandwidth selection methods, and whether to use the binned kernel estimation as an approximation to the exact kernel estimation for bandwidth selection and/or density estimation. The runtime we report here is based on the following settings: 1) the number of evaluation points is the same as the number of pixels evaluated by PALM, i.e., the pixels on the fixed grid with the granularity ϵ; 2) the binned approximation for the plug-in bandwidth selection is used; otherwise it becomes too slow[11]; 3) the binned approximation for the density estimation is not used. Note that we use these same settings not only for the runtime evaluation, but also for visualizations and calculating the log-likelihood on the testing datasets.

5.8.5 Algorithm settings

We now describe some additional algorithm settings for reproducibility for PALM and KDE.

Kernel density estimation (KDE). We choose the Gaussian kernel for KDE, the most commonly used kernel by default. We also experiment with several bandwidth selection methods, including both plug-in methods and cross-validation methods. We find that plug-in methods are generally both more stable and much faster in these three cases, and we choose the one that is specifically designed for two-dimensional cases (Duong and Hazelton 2003).

Also, we visualize the KDE results by directly plotting the density of each "pixel"; another common practice is to use a contour function, which will further smooth the KDE results and hence hamper the straightforward comparisons with the PALM results.

PALM. We set $\epsilon = 1$ and $K_{max} = 100$ for the amusement park dataset, as the amusement park is divided into a 100×100 grid, so the data is recorded at precision of 1 and the maximum number of bins cannot exceed 100. For the other two datasets, the precision of the dataset is set as $\epsilon = 0.001$, which is roughly

[11] It cost more than 10 minutes for the Amsterdam housing data, and more than two days for the amusement park data, both on the full dataset (no splitting for training and testing set).

100 meters. During the partitioning step, we set $K_{max} = 300$ to make sure that $\hat{K} < K_{max}$.

5.9 Conclusions

We proposed to discretize two-dimensional data by histograms with far more flexible partitions than adaptive grids, as we observed that the appropriate binning of one dimension may depend on the value of the other dimension.

Next, we formalized this task based on the MDL principle. Building upon the one-dimensional MDL histogram, we made several technical contributions so as to extend both the formulation and algorithm to the two-dimensional case. Specifically, we solved the problem of calculating the parametric complexity for multi-dimensional cases. Also, we revisited and improved the algorithm for one-dimensional dataset by 1) correcting a minor flaw related to the model encoding, and 2) simplifying the dynamic programming recursion and hence improving the time complexity.

Further, we proposed a novel heuristic algorithm PALM, which combines the top-down and bottom-up search strategies, and we extensively examined the performance of the PALM algorithm on both synthetic and real-world datasets. That is, we verified our algorithm on various synthetic datasets, and showed that: 1) PALM reveals the ground-truth histogram and converges, in contrast to IPD that produces more and more bins as sample size increases; 2) PALM approximates well to the partitions outside the model class; 3) PALM is self-adaptive to local density structures and sample sizes.

Finally, we applied our algorithm on three diverse real-world spatial datasets, and demonstrated that PALM not only captures more densities changes than KDE, but also fits the unseen data better than KDE, as measured by the log-likelihood.

5.10 Appendix A: Proof of Proposition 3

Proposition 3: *The parametric complexity* $COMP(n, \widetilde{S})$ *of a histogram model is a function of sample size* n *and the number of bins* K*. Given* n *and* K*,* $COMP(n, \widetilde{S})$ *is independent of the dimensionality of the data.*

Assume $S \subset \mathbb{R}^l$, $\widetilde{S}$ is any partition of S with K regions, and $\forall S_j \in \widetilde{S}$, $|S_j|$ represents the (hyper-)volume of S_j; for any y^n that can be generated by $\widetilde{S}$, $h_j(y^n)$ denotes the number of data points in region S_j.

$$\begin{aligned}
\mathrm{COMP}(n, \widetilde{S}) &= \sum_{y^n \in S^n} P(y^n | \widetilde{S}_{\vec{f}=\hat{\vec{f}}(y^n)}) \\
&= \sum_{y^n \in S^n} [\prod_{j=1}^{K} (\frac{h_j(y^n)\,\epsilon^l}{n\,|S_j|})^{h_j}] \\
&= \sum_{h_1+\ldots+h_K=n, h_j \geq 0, \forall j} \quad \sum_{\{y^n : h_j(y^n)=h_j, \forall j\}} [\prod_{j=1}^{K} (\frac{h_j(y^n)\,\epsilon^l}{n\,|S_j|})^{h_j}]
\end{aligned} \tag{5.28}$$

To count the elements in the set $\{y^n : h_j(y^n) = h_j, \forall j\}$, we observe that the number of possible ways of distributing $(h_1, ..., h_K)$ data points into each region of $\widetilde{S}$ respectively is

$$\binom{n}{h_1}\binom{n-h_1}{h_2}\cdots\binom{n-h_1-\ldots-h_{K-1}}{h_K} = \frac{n!}{h_1!...h_K!}. \tag{5.29}$$

As we assume the precision to be ϵ, for any S_j, the number of possible locations for those $h_j(y^n)$ points is equal to $(\frac{|S_j|}{\epsilon^l})^{h_j}$. Thus, the number of elements in the set $\{y^n : h_j(y^n) = h_j, \forall j\}$ is

$$\frac{n!}{h_1!...h_K!} \prod_{j=1}^{K} \left(\frac{|S_j|}{\epsilon^l}\right)^{h_j} \tag{5.30}$$

Therefore,

$$\begin{aligned}
\mathrm{COMP}(n, \widetilde{S}) &= \sum_{h_1+\ldots+h_K=n} [\frac{n!}{h_1!...h_K!} \prod_{j=1}^{K} [(\frac{|S_j|}{\epsilon^l})^{h_j} \prod_{j=1}^{K} (\frac{h_j \cdot \epsilon^l}{n \cdot |S_j|})^{h_j}] \\
&= \sum_{h_1+\ldots+h_K=n} [\frac{n!}{h_1!...h_K!} \prod_{j=1}^{K} [(\frac{|S_j|}{\epsilon^l})^{h_j} (\frac{h_j \cdot \epsilon^l}{n \cdot |S_j|})^{h_j}] \\
&= \sum_{h_1+\ldots+h_K=n} \frac{n!}{h_1!...h_K!} \prod_{j=1}^{K} \left(\frac{h_j}{n}\right)^{h_j},
\end{aligned} \tag{5.31}$$

which completes the proof.

Note that for continuous data y^n, $\mathrm{COMP}(n, \widetilde{S})$ becomes an integral over

$y^n \in S^n$, but by the definition of Riemann integral, (which always exists since ϵ cancels out), the result of COMP$(n, \widetilde{S})$ is the same as Equation (5.31).

5.11 Appendix B: Proof of Proposition 4

Proposition 4: *For any two cut points $C_i, C_k \in C_a$, suppose $C_i < C_k$ and no data points exist in the interval $[C_i, C_k]$, then any cut point $C_j \in [C_i, C_k]$ would not be in the MDL-optimal set of cut points, i.e., we can skip all such C_j during the search process.*

Consider one-dimensional data z^n, and a partition of the data space S, by a set of cut points, denoted as $C^K = \{C_0 = \min z^n, C_1, ..., C_K = \max z^n\}$, the probability of data is

$$P(z^n|C^K) = \prod_{j=1}^{K} (\frac{h_j\,\epsilon}{n\,|S_j|})^{h_j} \tag{5.32}$$

where h_j is the number of data points within the subinterval S_j, and $|S_j|$ is the length of the subinterval S_j.

We regard $P(x^n|C^K)$ as a *continuous* function of the vector $\vec{S} = (|S_1|, ..., |S_K|)$, i.e., we forget about the granularity ϵ for now, and clearly all h_j's are fixed once we fix the $\vec{S}$.

On the other hand, if we keep all h_j's fixed, we can still "move" all the cut points to change $\vec{S}$ while keeping the h_j's fixed, i.e., we can move a cut point V_x within some closed interval, denoted as $[a, b]$, within which no data points exist.

We prove that the maximum of $P(x^n|C^K)$ will always achieved when $V_x = a$ or $V_x = b$ as we keep other cut points fixed. By doing this, we also prove that, given candidate cut points, we only need to consider cut points that are near to the data points, i.e., if for any candidate cut point, it is another two cut points that are closest to it, other than one or more data points, we can then skip this candidate cut point.

Since when we move one single cut point, it only affects the subinterval left and right to that cut point, while all other $|S_j|$'s remain the same, it is sufficient to just prove for the case $K = 2$.

Since now $C_0 = \min_{i\in[n]} x_{i1}$ and $C_2 = \max_{i\in[n]} x_{i1}$, $P(x^n|C^2)$ becomes a function of C_1, and equivalently a function of $|S_1|$, where both C_1 and $|S_1|$ are

bounded as we need to keep h_1 and h_2 fixed, i.e.,

$$\log P(x^n|C^2) = \log \left(\left(\frac{\epsilon h_1}{n|S_1|} \right)^{h_1} \left(\frac{\epsilon h_2}{n(|S| - |S_1|)} \right)^{h_2} \right) \tag{5.33}$$

where we assume $|S_1| \in [a, b]$ for some certain closed interval $[a, b]$. As we want to maximize $\log P(x^n|C^2)$, it is equivalent to *minimizing*

$$F(|S_1|) := h_1 \log |S_1| + h_2 \log(|S| - |S_1|) \tag{5.34}$$

as other terms in Equation (5.33) are constant. Since

$$F'(|S_1|) = \frac{h_1(|S| - |S_1|) - h_2|S_1|}{(|S| - |S_1|)|S_1|}, \tag{5.35}$$

by setting $F'(|S_1|) = 0$, we have

$$|S_1|^* = \frac{h_1}{h_1 + h_2} L. \tag{5.36}$$

We also have

$$F''(|S_1|) = \frac{-(h_1 + h_2)|S_1|^2 + 2h_1|S||S_1| - h_1|S|^2}{(|S| - |S_1|)^2|S_1|^2} < 0 \tag{5.37}$$

because 1) the denominator is always positive apparently, and 2) the numerator is a simple quadratic function which is always negative. The reason is that 1) $-(h_1 + h_2)|S_1| < 0$ and 2) the numerator has no real roots, since

$$(2h_1|S|)^2 - 4(-(h_1 + h_2))(h_1|S|^2) = -4h_2h_1|S_1|^2 < 0. \tag{5.38}$$

Therefore, if $|S_1|^* \notin [a, b]$, $F(|S_1|)$ is monotonic within $[a, b]$; if $|S_1|^* \in [a, b]$, $|S_1|^*$ reaches the *maximum*. In both cases, the minimum of $F(|S_1|)$ will be either a or b, which completes the proof.

5.12 Appendix C: IPD visualizations on case study datasets

Figure 5.12 shows the visualization of the IPD discretization results on two of the case study datasets.

Algorithm 5: PALM

Input: data x^n, data precision ϵ, sample space S, maximum number of splits per partitioning step K_{max}

Output: $\widetilde{S}$, a partition of S

$dir \leftarrow 0$ or 1;
while ***true*** **do**
  foreach $S_k \in \widetilde{S}$ **do**
    Partition S_k as $\widetilde{S_k}$ by finding the optimal cut lines for S_k in direction dir;
    $C^*_{S_k} = \arg\min_{C_{S_k}} L(\{x^n \in S_k\}, C_{S_k})$;
  if $\widetilde{S_k} = \{S_k\}$ *for all* $S_k \in \widetilde{S}$ **then**
    break;
  else
    $\widetilde{S} \leftarrow \bigcup \widetilde{S_k}$;
    $dir \leftarrow 1 - dir$;
$\widetilde{S}_{merge} \leftarrow \widetilde{S}$;
$K_{merge} \leftarrow$ the number of regions of $\widetilde{S}_{merge}$;
while ***true*** **do**
  Get all neighboring pairs of regions of $\widetilde{S}_{merge}$, $Pairs \leftarrow \{(S_j, S_k), \ldots\}$;
  foreach $(S_j, S_k) \in Pairs$ **do**
    $\widetilde{S'_{j,k}} \leftarrow$ merge the pair (S_j, S_k) in $\widetilde{S}_{merge}$;
    Calculate
    $$L(x^n, \widetilde{S'_{j,k}}) = -\log\left(P(x^n|\widetilde{S'_{j,k}})\right) + \log \mathrm{COMP}(n, K_{merge} - 1);$$
  if $\min_{S'_{j,k}} L(x^n, \widetilde{S'_{j,k}}) > L(x^n, \widetilde{S}_{merge})$ **then**
    return $\widetilde{S}_{merge}$;
  else
    $\widetilde{S}_{merge} \leftarrow \arg\min_{\widetilde{S'_{i,j}}} L(x^n, \widetilde{S'_{i,j}})$;
    $K_{merge} \leftarrow K_{merge} - 1$;

Sample size	MISE for subgroup: $L_{true} > 1$	overall MISE
100 000	0.00148	0.00148
300 000	0.00055	0.00074
500 000	0.00051	0.00065
700 000	0.00019	0.00069
1 000 000	0.00023	0.00058
3 000 000	0.00017	0.00055
5 000 000	0.00006	0.00051

Table 5.1: The average MISE of cases when $L_{true} > 1$, and the overall mean of MISE. We show that, when PALM fails to identify part of the true partitions, the learned histogram model still estimates the probability density accurately. The only explanation for these cases is that some neighboring regions in the true partitions have very similar "true" f_j as defined in Equation (5.26), as a result of which PALM does not deem it necessary to further partition these regions.

	Dataset	l_{palm}	l'_{palm}	l_{kde}	$(l_{palm} - l_{kde})/l_{kde}$
1	Amsterdam housing	29976.36	29983.31	30069.78	-0.00
2	Amusement park	270.56	262.0688	227.22	0.19
3	Chengdu taxi	14904.28	14742.05	14073.42	0.06

Table 5.2: The log-likelihood of PALM with partitioning vertically first, l_{palm}, and with partitioning horizontally first, l'_{palm}, and the log-likelihood of KDE, l_{kde}, on the test set for each of the three datasets.

	Dataset	sample size	PALM	KDE
1	Amsterdam housing	20 244	106.821	6.73
2	Amusement park	9 078 623	1134.581	2017.215
3	Chengdu taxi	107 573	60977.285	29.197

Table 5.3: Empirical runtime (in seconds) for the three case study datasets.

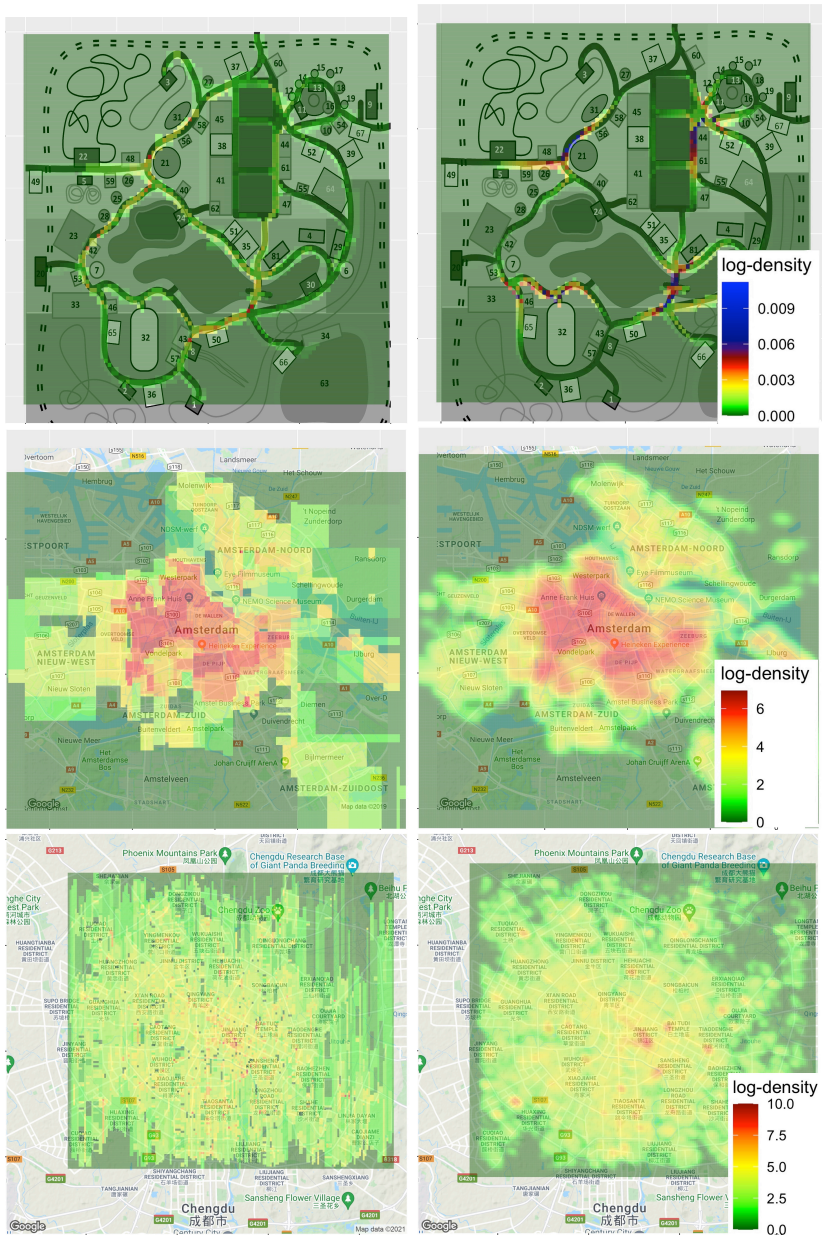

Figure 5.11: Estimated densities on three real-world datasets using PALM (left) and KDE (right); from top to bottom: DinoFun amusement park, Amsterdam Airbnb housing, and taxi destinations in Chengdu.

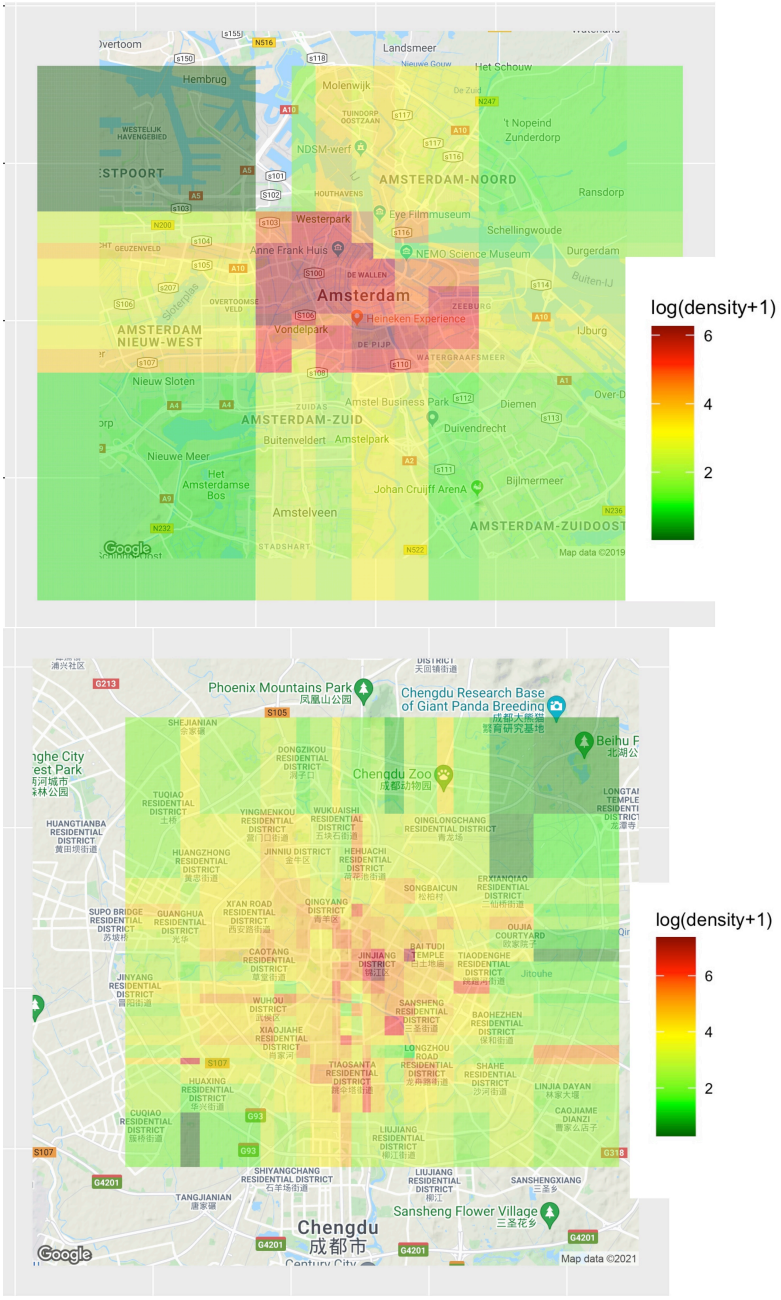

Figure 5.12: Visualization of the IPD discretization results on two of the case study datasets (we fail to obtain the result of IPD on the Amusement Park data within four hours).