

Multiple pass streaming algorithms for learning mixtures of distributions in \mathbb{R}^d

Kevin L. Chang
Yahoo! Labs
klchang@yahoo-inc.com

Mailing Address
Yahoo! Inc, 701 First Avenue, Sunnyvale, CA 94089, United States.
(408)-349-4699 (office phone)

January 13, 2009

Abstract

We present a multiple pass streaming algorithm for learning the density function of a mixture of k uniform distributions over rectangles in \mathbb{R}^d , for any $d > 0$. Our learning model is: samples drawn according to the mixture are placed in *arbitrary order* in a data stream that may only be accessed sequentially by an algorithm with a very limited random access memory space. Our algorithm makes $2\ell+2$ passes, for any $\ell > 0$, and requires memory at most $\tilde{O}(\epsilon^{-2/\ell} k^2 d^4 + (4k)^d)$, where ϵ is the tolerable error of the algorithm. This exhibits a strong memory-pass tradeoff in terms of ϵ : a few more passes significantly lowers its memory requirements, thus trading one of the two most important resources in streaming computation for the other. Chang and Kannan first considered this problem for $d = 1, 2$.

Our learning algorithm is especially appropriate for situations where massive data sets of samples are available, but computation with such large inputs requires very restricted models of computation.

1 Introduction

The rise of machine learning as an invaluable data analysis paradigm has coincided with the proliferation of massive data sets that stress computer systems in ways that render traditional models of computation inadequate. These two important considerations necessitate the theoretical study of algorithms for machine learning and statistical analysis that respect the resource constraints imposed by massive data set computation.

Of paramount importance is the observation that a large data set will not fit into the main memory of a computer system, but rather must be stored on disk or optical drives. For such data, well-designed memory access patterns are crucial, since access to data requires physical movement within storage devices. An algorithm will thus incur large time penalties for each random access; for large data sets, frequent random access is highly undesirable. Random access can be eliminated and I/O optimized by instead reading the data in a sequential fashion. The **streaming model** addresses these concerns and is popular in the theoretical computer science literature. The first few problems examined in the streaming model include sorting and selection [11] and approximating frequency moments [1]. In the streaming model, data in storage is modeled as a read-only array that can only be accessed sequentially in passes over the entire array. The algorithm may make a few passes over the array and use a small random-access memory space (usually sublinear in size, since one cannot hope to store the entire data set in memory) and may take constant time to process each element of the array. We will refer to this random-access memory space as simply “memory”. The important resources to be optimized are therefore passes and memory.

An important class of data mining and learning problems arises from generative clustering models. In these models, k clusters are defined by k probability distributions, F_1, \dots, F_k , over some universe Ω , each of which is given a weight $w_i \geq 0$ such that $\sum_1^k w_i = 1$. If the F_i s are density functions, then the mixture of these k distributions is defined by the density function $F = \sum_1^k w_i F_i$. The natural interpretation of a point drawn according to the mixture is that distribution F_i is picked with probability w_i , and then a point is drawn according to F_i . We consider the problem of estimating the probability density function of the mixture F given samples drawn according to the mixture.

In this paper, we will study the problem of learning **mixtures of k uniform distributions over axis-aligned rectangles in \mathbb{R}^d** , for any $d > 0$. In this case, each F_i is a uniform distribution over some cell in d dimensions $R_i = \{x \in \mathbb{R}^d \mid a_1 \leq x_1 \leq b_1, \dots, a_d \leq x_d \leq b_d\}$ for scalars $a_1, b_1, \dots, a_d, b_d$. The R_i s may intersect in arbitrary ways. Since the R_i s are arbitrary, learning the R_i s and w_i s from a set of samples from the mixture is an ill-defined problem, since different sets of rectangles and weights, when “mixed”, can form exactly the same distribution. Therefore, we will learn the density function, rather than the components, of the mixture. The output of the algorithm will be a function G that is an estimate of F .

One motivation behind learning mixtures of uniform distributions over rectangles is that these are among the simplest mixtures, and therefore any theory for learning mixture models in massive data set paradigms should start with this. Furthermore, these mixtures are building blocks for more complicated functions; continuous distribution in \mathbb{R}^d can be approximated as a mixture of sufficiently many uniform distributions over rectangles in \mathbb{R}^d . Our algorithm can then be used to learn this mixture.

Our learning and computational model is that samples drawn according to the mixture F are

placed in a data stream X , in **arbitrary order**.¹ Learning algorithms are required to be multiple-pass streaming algorithms, as described above. The output of the algorithm will be a function G that is an estimate of F , with error measured by L^1 distance: $\int_{\mathbb{R}^d} |F - G|$. An input parameter to the algorithm will be its probability of failure, $\delta > 0$. The approximation G will in general be *more complex* than simply the density function of a mixture of k uniform distributions.

Chang and Kannan [3] designed pass-efficient algorithms for learning a mixture of k uniform distributions over intervals in \mathbb{R} and axis-aligned rectangles in \mathbb{R}^2 . This work was subsequently improved by Guha and McGregor in [7]. In this paper, we use a similar high level approach, but develop new tools in order to generalize the algorithm to solve problems in arbitrary dimension.

1.1 Our results

Our main result is a multiple-pass algorithm for learning a mixture of k uniform distributions in \mathbb{R}^d with flexible resource requirements. The number of passes the algorithm may make is a function of an input parameter $\ell > 0$ that is independent of all other variables. The algorithm exhibits the power of multiple passes in the streaming learning model: if the algorithm is allotted just a few more passes and its error is held constant, then its memory requirements drop significantly as a function of ϵ . This is a strong **trade-off** between pass and memory requirements.

We will need the technical assumption that there exists a number $w > 0$, known to the algorithm, such that $F(x) \leq w$ for all $x \in \mathbb{R}^d$ and that all the probability mass of the mixture is contained in $[0, 1]^d$.

The main result of the paper is a $2\ell + 2$ pass algorithm that, with probability at least $1 - \delta$, will learn the mixture's density function to within L^1 distance ϵ and that uses memory at most

$$\tilde{O}\left(\frac{k^2 d^4}{\epsilon^{2/\ell}} + (4k)^d\right).$$

The algorithm requires the data stream to satisfy: $|X| = \tilde{\Omega}\left(\left(\frac{10}{8}\right)^\ell \frac{(kd)^{4d+3} w^{4d+2\ell}}{\epsilon^{4d+5\ell}}\right)$.²

1.1.1 Discussion of results

We note that the sample complexity and the memory requirements of the algorithm are exponential in the dimension d ; we partially justify the former requirement by our massive data set paradigm: we are working in the streaming model precisely because the data set is large. Despite this observation, the result is mostly of theoretical interest, since the sample complexity and memory requirements become unrealistic even for relatively modest values of d and k .

This does not preclude a strong pass-space trade-off for the algorithm, for many settings of the parameters d, k, ϵ . Since the memory requirement is $\tilde{O}(k^2 d^4 \epsilon^{-2/\ell} + (4k)^d)$, the trade-off between passes and memory is most significant in the case where the term involving ϵ dominates the memory requirement for small values of ℓ . This situation occurs when d and k are held constant and the tolerable error ϵ becomes very small. In this case, increasing ℓ will indeed reduce the memory requirement by very large factors. Again, this situation may be mostly of theoretical interest, since such small values of ϵ may not be required by applications.

¹Assuming that the data are randomly ordered is not always realistic; for instance if the data were collected from the census, then perhaps it would be ordered by address or some other attribute.

² $\tilde{O}(\cdot), \tilde{\Omega}(\cdot), \tilde{\Theta}(\cdot)$ denote asymptotic notation with polylogarithmic factors omitted.

1.2 Overview of methods

The main action of the algorithm is to learn the locations of the boundaries of the constituent mixture rectangles in $2\ell + 1$ passes. With this knowledge, **Learn**(\mathbf{d}, \mathbf{k}) can partition the domain into cells such that $F(x)$ is a constant function when restricted to each cell; in one more pass over the data stream, it can easily estimate these constants by counting the number of samples that fall in each of these cells.

Learn(\mathbf{d}, \mathbf{k}) requires Subroutine **FindBoundary**($\mathbf{d}, \mathbf{k}, \mathbf{m}$), which is a $2\ell + 1$ pass algorithm for finding the boundary edges of mixture rectangles that lie in a hyperplane that is perpendicular to the m th dimension; in one pass, this algorithm draws a sample from the data stream and uses the sample to partition the domain into a set of roughly $1/\epsilon^{1/\ell}$ cells that contain probability mass on the order of $\epsilon^{1/\ell}$ each. It then utilizes one pass subroutine **Invariant**($\mathbf{d}, \mathbf{k}, \mathbf{m}$) to test each of these cells for boundaries. Suppose **Invariant**($\mathbf{d}, \mathbf{k}, \mathbf{m}$) indicates that a boundary cell lies somewhere in partition cell P . In order to further localize this boundary cell (since it could be anywhere in the relatively large P), we iterate and partition P and then test the new subcells, which have probability mass approximately $\epsilon^{2/\ell}$. We continue iterating in this fashion. At each iteration, we are in essence “zooming in” on cells that we know contain the boundary. See Figure 1 for an example of an iteration of **FindBoundary**($\mathbf{d}, \mathbf{k}, \mathbf{m}$) in the $d = 2$ case.

Herein lies the trade-off between passes and memory. For example, if the size of the memory space is large, then a large sample can fit in memory, and therefore our partitioning of the domain at each round can be fine. The weight of F in each partition cell is small, and will shrink quickly with each extra round. Therefore, few passes will be necessary in order to sufficiently isolate the boundaries. If the memory space is small, then in each round the sample is small, and the weight of F in each partition cell will decrease slowly. Therefore, a larger number of passes will be required to isolate the boundaries.

The engine of our algorithm is **Invariant**($\mathbf{d}, \mathbf{k}, \mathbf{m}$), which determines if a cell C contains a boundary cell contained in a hyperplane that is perpendicular to the m th dimension. We formulate a statistic that can test this condition; since $|X|$ is large, the test will be very accurate and will tell us if F is within L^1 distance ϵ of what we would expect if C did not contain such a boundary cell. Computing the statistic in one pass using a small amount of memory presents an algorithmic challenge, and requires the application of a streaming algorithm by Indyk [9] for approximation of the ℓ_1 lengths of vectors using very little memory.

1.2.1 Comparison to \mathbb{R}^2 algorithm in [3]

We note that the approach in [3] for \mathbb{R}^2 is similar to the current approach, but differs in one key way, which yields a weaker algorithm. The high level idea of the previous algorithm is that the domain is partitioned into subcells, and each subcell is tested to see if it contains vertical edges of boundary rectangles. In rectangles *that do not contain vertical edges of boundary rectangles*, the problem could be reduced to the \mathbb{R} case, for which there was already a good algorithm. In rectangles that do contain such boundary edges, the algorithm iterates, as described above.

The current algorithm does not reduce the \mathbb{R}^d case to the \mathbb{R}^{d-1} case, but rather solves the problem directly by finding all the boundaries of the mixture rectangles. By avoiding this recursion, the current algorithm is much more efficient in terms of space and the number of passes.

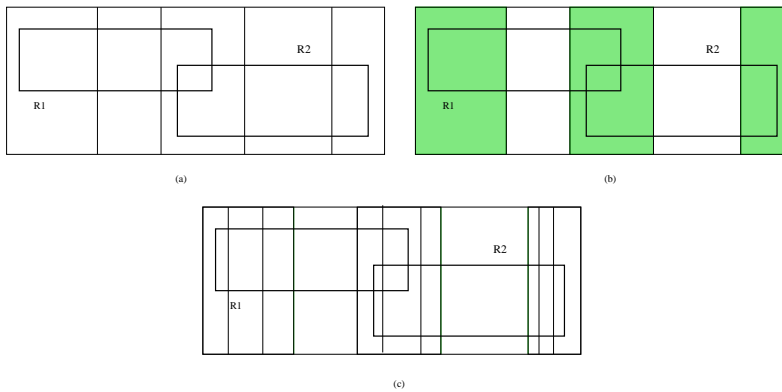


Figure 1: An example run of the algorithm **Invariant(d,k,m)** for $d = 2$, and learning the vertical boundary edges. a) The two mixture rectangles are given by R_1 and R_2 . The domain has been partitioned into a set of subcells. b) Each cell is tested, and the ones that contain vertical boundary edges are rejected (and shaded in the figure). c) These rejected partition cells are further partitioned in the next round.

1.3 Related work

Many algorithms for the unsupervised learning of mixtures of distributions have appeared in the learning and algorithms theory literature. Algorithms for learning mixtures of Gaussian distributions in \mathbb{R}^d [2, 5, 10, 15] generally estimate the means and covariance matrices of the constituent distributions from samples drawn according to the mixture. Algorithms for classification of sample points to their distribution of origin have been considered for more general distributions by Dasgupta *et al.* in [4]. We note that these algorithms are not suitable for massive data sets.

Many one and multiple pass algorithms for database and data mining problems appear in the theoretical computer science literature. Among the most relevant to this study are the algorithms for histogram maintenance. In the histogram maintenance problem, the algorithm is presented with a data stream of update pairs of the form “add 2 to a_j ”, where $j \in [n]$.³ During the pass, the algorithm must maintain a piecewise constant function $F(i)$, with k pieces, that minimizes $\sum_i |F(i) - a_i|$. In [6], Gilbert *et al.* gave a one pass algorithm for this problem with approximation ratio $1 + \epsilon$. This work gives the best piecewise constant approximation to arbitrary data (rather than assuming a generative data model) and is thus similar to our problem of learning the density function of a mixture of uniform distributions over intervals in \mathbb{R} . A d dimensional variant of the problem has been studied by Thaper *et al.* in [13] (their running time is also exponential in d).

Streaming algorithms with a statistical flavor include the work of Guha *et al.* [8], who consider one pass algorithms for estimating the entropy of a distribution from samples in a stream.

1.4 Problem Setup

Our algorithm will learn mixtures of distributions over axis-aligned rectangles in \mathbb{R}^d . For completeness, we define rectangles:

³ $[n]$ denotes the set $\{1, \dots, n\}$

Definition 1 (*d*-cell) For any positive integer $d > 0$, we define a *d*-cell to be a set $K \subset \mathbb{R}^d$ that satisfies $K = \{x \in \mathbb{R}^d | a_1 \leq x_1 \leq b_1, \dots, a_d \leq x_d \leq b_d\}$ for some choice of scalars $a_1, b_1, \dots, a_d, b_d$. We will sometimes write K as a cross product of d intervals in \mathbb{R} : $K = (a_1, b_1) \times \dots \times (a_d, b_d)$.

The **volume** of the *d*-cell K is given by $\text{vol}(K) = |b_1 - a_1| \cdot |b_2 - a_2| \cdot \dots \cdot |b_d - a_d|$.

Throughout this paper, the input will be the data stream X of length $|X| = N$, with samples drawn according to a mixture of k uniform distributions in \mathbb{R}^d , where the mixture cells may intersect arbitrarily. The density function of the mixture will be denoted by F . We assume that the algorithm knows a number $w > 0$ such that $F(x) \leq w$ for all $x \in \mathbb{R}^d$ and that all mixture cells are contained in the cell $[0, 1]^d$. We will call the smallest enclosing cell of the mixture the **bounding box**, $R \subseteq [0, 1]^d$.

2 Main algorithm: Learn(*d*, *k*)

We will present our $2\ell + 2$ pass algorithm as an algorithm with error ϵ^ℓ and roughly constant memory requirements in terms of ℓ . We will then show how to transform the parameters in order to derive an algorithm with error ϵ and a strong memory and pass trade-off.

2.1 Preliminaries

One of the main tools that we will use in our analysis is the VC bound, which was first developed by Vapnik and Červonenkis [14] and later improved by Talagrand [12]. This is a very fundamental result used often in learning theory. Below we state the bound that we use in this paper, noting that the statement is less general than in the papers cited above.

2.1.1 The VC bound

Given a (measurable) probability density function F , let $\mu(U) = \int_U F$ be the the probability that a point drawn from the distribution falls in the set U . Given a sequence X of m points randomly chosen according to the distribution F , $|X \cap U|$ is the set of points in X that lie in set U . Note that $|X \cap U|/m$ is the obvious empirical estimate of $\int_U F$.

Fact 1 (Vapnik and Červonenkis, Talagrand bound) (*Theorem 1.2 from [12]*) Let \mathcal{C} be a family of measurable sets with VC dimension d , and let $\epsilon > 0$, $\delta > 0$. Then there exists a constant c_0 such that if X is a set of m samples drawn according to μ , and

$$m \geq c_0 \frac{1}{\epsilon^2} \left(d \log \left(\frac{1}{\epsilon} \right) + \log \left(\frac{1}{\delta} \right) \right), \quad (1)$$

then

$$\Pr \left[\sup_{U \in \mathcal{C}} \left| \frac{|X \cap U|}{m} - \mu(U) \right| \geq \epsilon \right] \leq \delta.$$

The power of the VC bound lies in the fact that, when m is sufficiently large, the error between the empirical estimate $|X \cap U|/m$ and the true probability mass $\mu(U)$ is less than ϵ for all sets $U \in \mathcal{C}$ simultaneously with probability $1 - \delta$.

2.1.2 m th component invariance

Before our exposition of the main algorithm, we first introduce the concept of m th component invariance and an algorithm for testing this condition.

Definition 2 (m th component invariance) *A function f is m th component invariant in a set $K \subset R$ if it satisfies the following condition: if x and $y \in K$ satisfy $x_i = y_i$ for all $i \neq m$, then $f(x) = f(y)$. In words, m th component invariance is the condition where $f(x)$ is constant if all components are fixed except the m th component.*

Intuitively, if cell K is invariant in the m th component, then a learning algorithm does not need to consider the m th component when learning the function F in K . A key observation is that if F is invariant in all d components in cell K , then F is, in fact, constant in K .

The learning algorithm relies on the subroutine **FindBoundary**($\mathbf{d}, \mathbf{k}, \mathbf{m}$), which will learn a decomposition of the bounding box R into a set of cells such that F is invariant in the m th component in most of the cells. This subroutine is the engine for **Learn**(\mathbf{d}, \mathbf{k}); its proof of correctness will be presented in Section 3. To ease the proliferation of complicated expressions in the exposition, define

$$\text{SC}(d, k, \epsilon, \delta, \ell, w) = \left(\frac{10}{8}\right)^\ell \frac{(kd)^{4d+3} w^{4d+2\ell}}{\epsilon^{4\ell d+5\ell}} \log \frac{1}{\delta},$$

which is the sample complexity of **FindBoundary**($\mathbf{d}, \mathbf{k}, \mathbf{m}$). Note that $\text{SC}(d, k, \epsilon, \delta, \ell, w)$ is exponential in the dimension d . It also contains a term with ϵ^ℓ , but this is necessary, since the error of the algorithm is ϵ^ℓ .

Theorem 1 ***FindBoundary**($\mathbf{d}, \mathbf{k}, \mathbf{m}$) with postprocessing is a $2\ell+1$ pass algorithm that requires at most $\tilde{O}\left(\frac{\ell d^3 k^2}{\epsilon^2} + \frac{\ell^3 k^2 d}{\epsilon^2} + \frac{\ell^2 k^2 d^2}{\epsilon^2}\right)$ bits of memory and $O\left(\frac{\ell d^2 w k^2}{\epsilon^{2\ell}} \log(kw/\epsilon\delta)\right)$ per-element update time. If $|X| = \tilde{\Omega}(\text{SC}(d, k, \epsilon, \delta, \ell, w))$, then with probability at least $1 - \delta$, it will find a set of cells, \mathcal{V} , such that*

1. For all $V_i, V_j \in \mathcal{V}$ such that $V_i \neq V_j$, $V_i \cap V_j = \emptyset$ and $|\mathcal{V}| \leq 4k$.
2. There exists a function F_m such that F_m is invariant in the m th component in each $V \in \mathcal{V}$ and such that

$$\int_{\cup_{V \in \mathcal{V}} V} |F - F_m| \leq \frac{2\epsilon^{2\ell}}{9dw}$$

3. and

$$\int_{R \setminus (\cup_{V \in \mathcal{V}} V)} F \leq \frac{\epsilon^\ell}{3d},$$

where R is the bounding box.

The algorithm thus finds a set of disjoint cells, such that for all $V \in \mathcal{V}$, F restricted to V is very close to invariant in the m th component (note that the algorithm guarantees the existence of F_m

⁴We note that all of our algorithms' memory requirements have a factor of $\log(1/\delta)$, which is omitted due to the $\tilde{O}(\cdot)$ notation.

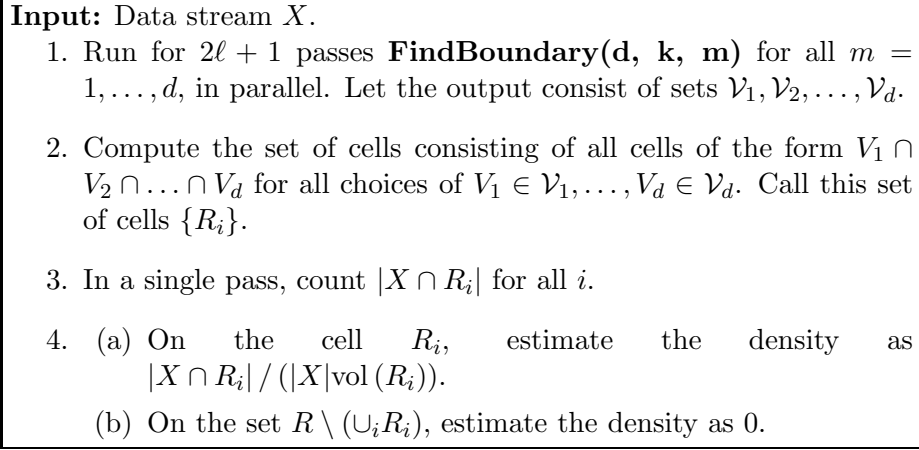


Figure 2: Algorithm **Learn**(\mathbf{d}, \mathbf{k})

but only finds \mathcal{V} , not F_m). The last condition implies that the cells in \mathcal{V} contain nearly all the weight of F .

Our learning algorithm **Learn**(\mathbf{d}, \mathbf{k}) will run **FindBoundary**($\mathbf{d}, \mathbf{k}, \mathbf{m}$) for all $m = 1, \dots, d$. The output of each call will consist of a set of cells \mathcal{V}_m such that F is nearly invariant in the m th component in each cell. Let $V_m \in \mathcal{V}_m$ be such a cell. For any two indices $1 \leq m_1 < m_2 \leq d$, consider the d -cell $C = V_{m_1} \cap V_{m_2}$. F , restricted to C , is close to a function that is invariant in the m_1 th component, and also close to another function that is invariant in the m_2 th component. Intuitively, such an F should be close to a function that is invariant in the m_1 th and m_2 th components simultaneously in C .

Extending the reasoning further, let $C = V_1 \cap \dots \cap V_d$, where $V_i \in \mathcal{V}_i$ for all i . Then F restricted to C is close to a function that is invariant in the m th component, for all m . We prove that this will imply that F is close to *constant* in C .

2.2 The Algorithm

We present **Learn**(\mathbf{d}, \mathbf{k}) in Figure 2. The algorithm will call **FindBoundary**($\mathbf{d}, \mathbf{k}, \mathbf{m}$) for each component $m = 1, \dots, d$; from the resulting sets \mathcal{V}_m , it will decompose R into cells R_i such that F is close to invariant in R_i in all d components. Lastly, it will then treat F as if it were constant on R_i , and estimate the density F in R_i by simply counting the number of sample points that lie in R_i .

Theorem 2 *If $|X| = \tilde{\Omega}(SC(d, k, \epsilon, \delta, \ell, w))$, then with probability at least $1 - \delta$, **Learn**(\mathbf{d}, \mathbf{k}) will compute an estimate G such that $\int_R |F - G| \leq \epsilon^\ell$ in $2\ell + 2$ passes, using memory at most $\tilde{O}(\frac{\ell d^4 k^2}{\epsilon^2} + \frac{\ell^3 k^2 d^2}{\epsilon^2} + \frac{\ell^2 k d^3}{\epsilon^2} + (4k)^d)$ and per-element update time $O(\frac{d^3 k^2}{\epsilon^{2\ell}} \log(kw/\epsilon^\ell \delta))$.*

Proof: From Theorem 1, we know that each call to **FindBoundary**($\mathbf{d}, \mathbf{k}, \mathbf{m}$) will output a set of d -cells \mathcal{V}_m such that there exists a function F_m that is invariant in the m th component on each $V \in \mathcal{V}_m$ and that satisfies $\int_{\cup_{V \in \mathcal{V}_m} V} |F_m - F| \leq \epsilon^\ell / 3d$. For each rectangle found in Step 2, this is true for all choices of m simultaneously. Note that since $|\mathcal{V}_m| \leq 4k$, $|\{R_i\}| \leq (4k)^d$.

Fix such a rectangle R_i . The following property is a precise statement of the intuitive idea that F should be close to constant in R_i . Let $\bar{F}_{R_i} = \int_{R_i} F / \text{vol}(R_i)$ be the *scalar* that is the average value of F in R_i and recall that w is defined to be an upper bound on F : $F(x) \leq w$ for all $x \in R$. The proof of the property can be found in Appendix A.

Property 1

$$\int_{R_i} |F - \bar{F}_{R_i}| \leq \left(2w \sum_{m=1}^d \int_{R_i} |F_m - F| \right)^{\frac{1}{2}}$$

Recall that $\int_{R_i} |F_m - F|$ is the distance between F and a function F_m that is invariant in the m th component in each $V \in \mathcal{V}_m$, returned by **FindBoundary**($\mathbf{d}, \mathbf{k}, \mathbf{m}$). Therefore, the above bound on the distance between F and the constant \bar{F}_{R_i} is in terms of the distances between F and functions that are invariant in each dimension. Intuitively, if the latter is small in all dimensions, then F is close to constant.

Let \mathcal{K} be the family of d -cells in \mathbb{R}^d . Since \mathcal{K} is comprised of axis-aligned cells, the family has VC dimension $2d$. Since X is drawn according to F , we have chosen our sample complexity so that the VC bound implies that

$$\Pr \left[\sup_{K \in \mathcal{K}} \left| \frac{|X \cap K|}{|X|} - \int_{R_i} F \right| \leq \frac{\epsilon^\ell}{3(2k)^d} \right] \geq 1 - \frac{\delta}{4}$$

and therefore

$$\Pr \left[\max_i \left| \frac{|X \cap R_i|}{N} - \int_{R_i} F \right| \leq \frac{\epsilon^\ell}{3(2k)^d} \right] \geq 1 - \frac{\delta}{4},$$

where $N = |X|$.

Recall that $|X \cap R_i| / N \text{vol}(R_i)$ is our algorithm's estimate of F in R_i . We now sum our bound on the error of our estimate in each rectangle R_i .

$$\begin{aligned} \sum_i \int_{R_i} \left| F - \frac{|X \cap R_i|}{\text{vol}(R_i) N} \right| &\leq \sum_i \left(\int_{R_i} \left| \frac{|X \cap R_i|}{\text{vol}(R_i) N} - \bar{F}_{R_i} \right| + \int_{R_i} |F - \bar{F}_{R_i}| \right) \\ &\leq \sum_i \left(\left| \frac{|X \cap R_i|}{N} - \int_{R_i} F \right| + \int_{R_i} |F - \bar{F}_{R_i}| \right) \\ &\leq \sum_i \left(\frac{\epsilon^\ell}{3(2k)^d} + \int_{R_i} |F - \bar{F}_{R_i}| \right) \\ &\leq (2k)^d \frac{\epsilon^\ell}{3(2k)^d} + \sum_i \int_{R_i} |F - \bar{F}_{R_i}| \\ &\leq \frac{\epsilon^\ell}{3} + \left(2w \sum_{m=1}^d \int_{\cup V \in \mathcal{V}_m V} |F_m - F| \right)^{\frac{1}{2}} \leq \frac{2\epsilon^\ell}{3}. \end{aligned}$$

The second inequality follows from the fact that the first integrand is a constant, the fifth inequality from Property 1, and the final inequality from **FindBoundary**($\mathbf{d}, \mathbf{k}, \mathbf{m}$)'s guarantee on $\int_{\cup V \in \mathcal{V}_m V} |F_m - F|$.

Lastly, we bound the error induced by estimating F as 0 on the set $R \setminus (\cup_i R_i)$. Let $\bar{V}_m = R \setminus (\cup_{V \in \mathcal{V}_m} V)$ be the set for which F_m is not invariant in the m th component. Therefore $R \setminus (\cup_i R_i) = \cup_m \bar{V}_m$. By Theorem 1, we know that $\int_{\bar{V}_m} F dx \leq \epsilon^\ell / (3d)$. Thus,

$$\int_{R \setminus (\cup_i R_i)} F \leq \sum_{m=1}^d \int_{\bar{V}_m} F \leq d \cdot \frac{\epsilon^\ell}{3d} \leq \frac{\epsilon^\ell}{3}.$$

The total error of the algorithm is therefore at most ϵ^ℓ . \square

Corollary 3 *There exists a $2\ell + 2$ pass algorithm that, with probability at least $1 - \delta$, will learn a mixture of uniform distributions in \mathbb{R}^d with error at most ϵ , using memory at most $\tilde{O}(\frac{k^2 d^4}{27\ell} + (4k)^d)$.*

Proof: If we transform the parameter ϵ to $\epsilon^{1/\ell}$, then we may assume that $\ell = O(\log 1/\epsilon)$. More passes than this would not decrease the memory requirement beyond a constant. \square

3 An algorithm for learning the location of boundary edges

Algorithm **FindBoundary**($\mathbf{d}, \mathbf{k}, \mathbf{m}$) computes a decomposition of the bounding box R into a set of cells \mathcal{V} such that F is invariant in the m th component in each cell. Roughly stated, it does so by ensuring that each cell $V \in \mathcal{V}$ does not contain an m th-component **boundary** cell of a mixture cell, as defined below.

Definition 3 (boundary) *The **boundary** of a d -cell $K = (a_1, b_1) \times \dots \times (a_d, b_d)$ consists of the $2d$ $d-1$ -cells defined by $(a_1, b_1) \times \dots \times (a_i, a_i) \times \dots \times (a_d, b_d)$ and $(a_1, b_1) \times \dots \times (b_i, b_i) \times \dots \times (a_d, b_d)$ for $i = 1, \dots, d$. The two cells for which $i = m$ are called **m th component boundary cells**.*

Informally, an m th component boundary cell is a $d-1$ -cell embedded in \mathbb{R}^d , such that the m th component of the boundary cell is the same for all points in the cell. Our interest in m th component boundary cells is summarized in the following, very intuitive, property.

Property 2 *If a d -cell K does not contain any m th component boundary cells of mixture rectangles, then F , restricted to K , is invariant in the m th component.*

Thus, boundary cells are just the natural geometric concept of the boundary of a rectangle in \mathbb{R}^d . See Figure 3 for an example in \mathbb{R}^2 .

Definition 4 *A **partition of the m th component** of d -cell $R = (a_1, b_1) \times \dots \times (a_d, b_d)$ is a set of cells of the form $P_i = \{x \in R | u_i \leq x_m < u_{i+1}\}$ (i.e. $P_i = (a_1, b_1) \times \dots \times (a_{m-1}, b_{m-1}) \times (u_i, u_{i+1}) \times (a_{m+1}, b_{m+1}) \times \dots \times (a_d, b_d)$), where $a_m = u_1 \leq u_2 \leq \dots \leq u_{|\mathcal{P}_m|} = b_m$, where \mathcal{P}_m is the set of partition cells $\mathcal{P}_m = \{P_i\}$.*

Note that an m th component boundary cell of a mixture rectangle is completely contained in a single partition rectangle. See Figure 4 for an illustration.

The algorithm **FindBoundary**($\mathbf{d}, \mathbf{k}, \mathbf{m}$) requires a subroutine **Invariant**($\mathbf{d}, \mathbf{k}, \mathbf{m}$) that will check if F is approximately m th component invariant when restricted to the d -cell K . We will defer the proof of the following theorem to Section 4.

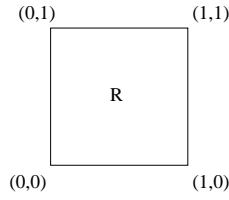


Figure 3: $R = (0, 1) \times (0, 1)$ is an example of a 2-cell. The boundary of R consists of the 4 line segments that outline the cell. The two 1-boundary cells are: $\{x \in \mathbb{R}^2 | x_1 = 0, 0 \leq x_2 \leq 1\}$ and $\{x \in \mathbb{R}^2 | x_1 = 1, 0 \leq x_2 \leq 1\}$ and the two 2 boundary cells are: $\{x \in \mathbb{R}^2 | 0 \leq x_1 \leq 1, x_2 = 0\}$ and $\{x \in \mathbb{R}^2 | 0 \leq x_1 \leq 1, x_2 = 1\}$.

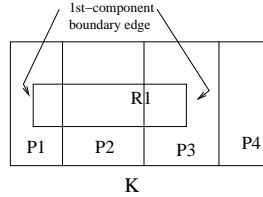


Figure 4: K and R_1 are 2-cells. The boundaries of K and R_1 are illustrated by their respective rectangular outlines. P_1, P_2, P_3, P_4 comprise a partition of the 1st component of K . Note that the 1st-component boundary edges of R_1 are completely contained in single partition cells.

Theorem 4 Let X be a data stream that contains samples from a mixture of k uniform distributions in \mathbb{R}^d with density function H and let $\beta > 0$ be some error parameter. If $|X| = \Omega\left(\frac{(kwd)^{2d}}{\beta^{2d+2}} \log\left(\frac{kdw}{\beta\delta}\right)\right)$ then with probability at least $1 - \delta$, one pass algorithm **Invariant**(d, k, m) will **accept** if H is invariant in the m th component and **reject** if there does **not** exist a function \hat{H} that is invariant in the m th component such that $\int_R |H - \hat{H}| \leq \beta$. **Invariant**(d, k, m) requires $\tilde{O}(d \log^2(kdw/\beta))$ bits of memory and $O(\frac{d}{\beta} \log(kw/(\beta\delta)))$ per-element update time.

3.1 The Algorithm

We first give an overview of algorithm **FindBoundary**(d, k, m). It is organized into ℓ pairs of passes. In the first pass of each pair, it takes a small sample from the data stream and uses it to find a partition of the m th component of the bounding box, such that all cells have roughly equal probability mass with respect to F . In a second pass, it tests each partition cell for invariance in the m th component. **Invariant**(d, k, m) uses the large amount of data in the data stream to perform this test with *very high accuracy*. A cell is rejected only if it contains an m th component boundary cell. We iterate on all rejected cells; the key observation is that these partition cells contain much less probability weight than the original bounding box and therefore will be sampled much more densely (but with the same overall sample size) and we will get better estimates for these interesting cells.

The cells that were rejected in the final iteration and that therefore contain m th component boundary cells have only a very small aggregate weight (less than $\epsilon^\ell/(3d)$). Thus, the set of partition

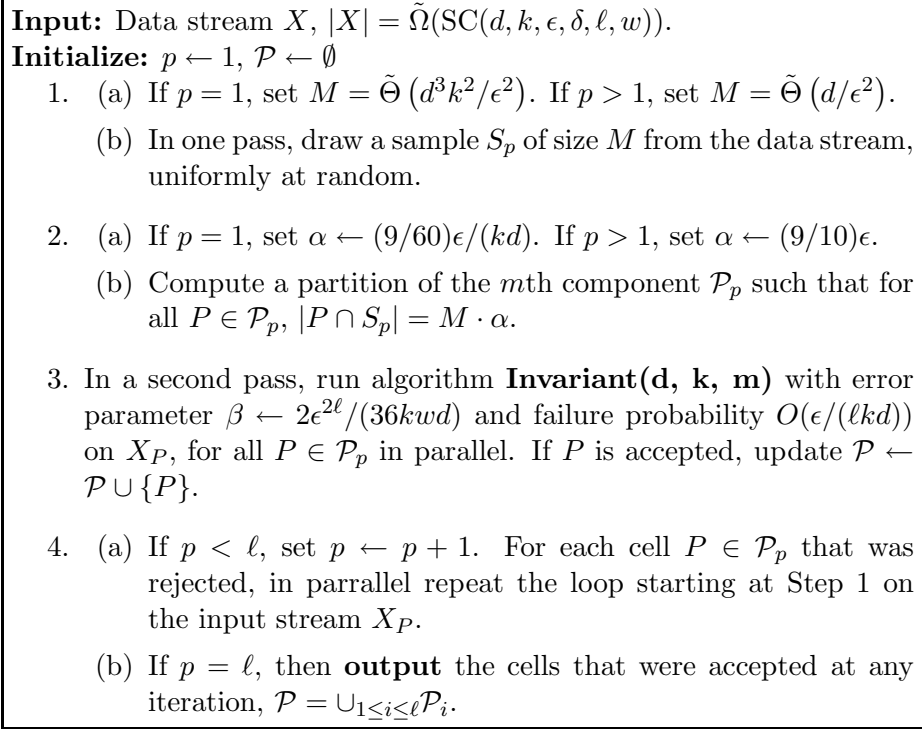


Figure 5: Algorithm **FindBoundary**($\mathbf{d}, \mathbf{k}, \mathbf{m}$)

cells consisting of those cells that were accepted at some iteration, in which F is guaranteed to be close to invariant in the m th component, could be a solution. However, the number of such cells is large, and in a final pass we will reduce the number to $4k$ with a postprocessing algorithm.

We describe the main algorithm in Figure 5, prove some of its properties, and then later describe the postprocessing algorithm. If $K \subset R$, then we denote by X_K the substream of X that consists of the samples in $K \cap X$. We note that a pass over X can simulate a pass over X_K .

Definition 5 *Since in step 4a, we run the algorithm in parallel on all rejected cells, we will call these **parallel instantiations** of the algorithm. We call the value of p the **level of the iteration**. We will refer to cells created in Step 2b of a level p iteration as **level p partition cells**.*

We first prove a simple lemma about the parallel instantiations of the algorithm.

Lemma 5 *With probability at least $1 - \delta/3$, the number of parallel instantiations of the algorithm at any level is at most $2k$.*

Proof: The total number of intervals created across all parallel instantiations of **FindBoundary**($\mathbf{d}, \mathbf{k}, \mathbf{m}$) is at most $O(\ell k / \epsilon + dk / \epsilon)$. Since **Invariant**($\mathbf{d}, \mathbf{k}, \mathbf{m}$) was called on each of these cells, by the union bound, the probability that at least one call to **Invariant**($\mathbf{d}, \mathbf{k}, \mathbf{m}$) failed is at most $\delta/3$. We condition on no calls failing.

Invariant($\mathbf{k}, \mathbf{d}, \mathbf{m}$) will only reject a cell P if P contains an m th component boundary cell of a mixture rectangle. Since there are at most k mixture d -cells, each of which has two such boundary

cells, the total number of boundary cells that can be rejected across all level p instantiations is $2k$. Therefore the algorithm iterates on only $2k$ cells across all parallel level p instantiations. \square

We now prove that the probability mass of all ℓ th level cells created in Step 2b of **FindBoundary**($\mathbf{d}, \mathbf{k}, \mathbf{m}$) decreases exponentially in the number of passes, ℓ .

Lemma 6 *With probability at least $1 - O((p-1)\delta/\ell d)$, for all cells $P \in \mathcal{P}_p$ created by the algorithm in Step 2b of iteration p :*

$$\left(\frac{8}{10}\right)^p \frac{\epsilon^p}{6kd} \leq \int_P F \leq \frac{\epsilon^p}{6kd}.$$

Proof: We first consider all level 1 partition cells $P \in \mathcal{P}_1$. Let \mathcal{K} be the family of d -cells in \mathbb{R}^d , which has VC dimension $2d$. Since we drew a sample S_1 of size $M = \tilde{\Theta}(d^3 k^2 / \epsilon^2)$ from the data stream, the VC bound implies that

$$\Pr \left[\sup_{K \in \mathcal{K}} \left| \int_K F - \frac{|K \cap S_1|}{|S_1|} \right| \leq \frac{1}{60} \cdot \frac{\epsilon}{kd} \right] \geq 1 - O\left(\frac{\delta}{dk\ell}\right).$$

Each $P \in \mathcal{P}_1$ created in Step 2b of the first iteration satisfies $|P \cap S_1| = \frac{9}{10} \frac{\epsilon}{6kd} |S_1|$. Therefore, for all such P , with probability at least $1 - O(\delta/(dk\ell))$,

$$\frac{8}{10} \cdot \frac{\epsilon}{6kd} \leq \int_P F \leq \frac{\epsilon}{6kd}.$$

Now suppose that level p partition cell $P \in \mathcal{P}_p$ was created by partitioning the cell $R_{p-1} \in \mathcal{P}_{p-1}$ (i.e. the input to the iteration) in Step 2b of a p th level iteration.

Claim 1 *With probability at least $1 - O(\delta/(dk\ell))$, for all level p partition cells P ,*

$$\frac{8}{10} \epsilon \int_{R_{p-1}} F \leq \int_P F \leq \epsilon \int_{R_{p-1}} F.$$

Proof: Since we draw a sample of size $\tilde{\Theta}(d/\epsilon^2)$, using the VC bound we determine that:

$$\Pr \left[\max_{P \in \mathcal{P}_p} \left| \frac{|S_p \cap P|}{|S_p|} - \int_P \frac{F}{\int_{R_p} F} \right| \leq \frac{1}{10} \epsilon \right] \geq 1 - O\left(\frac{\delta}{d\ell k}\right).$$

Since $|P \cap S_p| = \frac{9}{10} \epsilon |S_p|$, the claim follows. \square

With the claim, the lemma can be proved by induction as follows. Assume that Lemma 6 is true for level $p-1$. Since R_{p-1} was a $p-1$ th level cell created by the algorithm, it follows from the inductive hypothesis that with probability at least $1 - O((p-1)\delta/\ell d)$:

$$\left(\frac{8}{10}\right)^{p-1} \frac{\epsilon^{p-1}}{6dk} \leq \int_{R_{p-1}} F \leq \frac{\epsilon^{p-1}}{6dk}.$$

Applying the claim then yields: with probability at least $1 - O(p\delta/\ell d)$, for all $P \in \mathcal{P}_p$

$$\left(\frac{8}{10}\right)^p \frac{\epsilon^p}{6kd} \leq \int_P F \leq \frac{\epsilon^p}{6kd}.$$

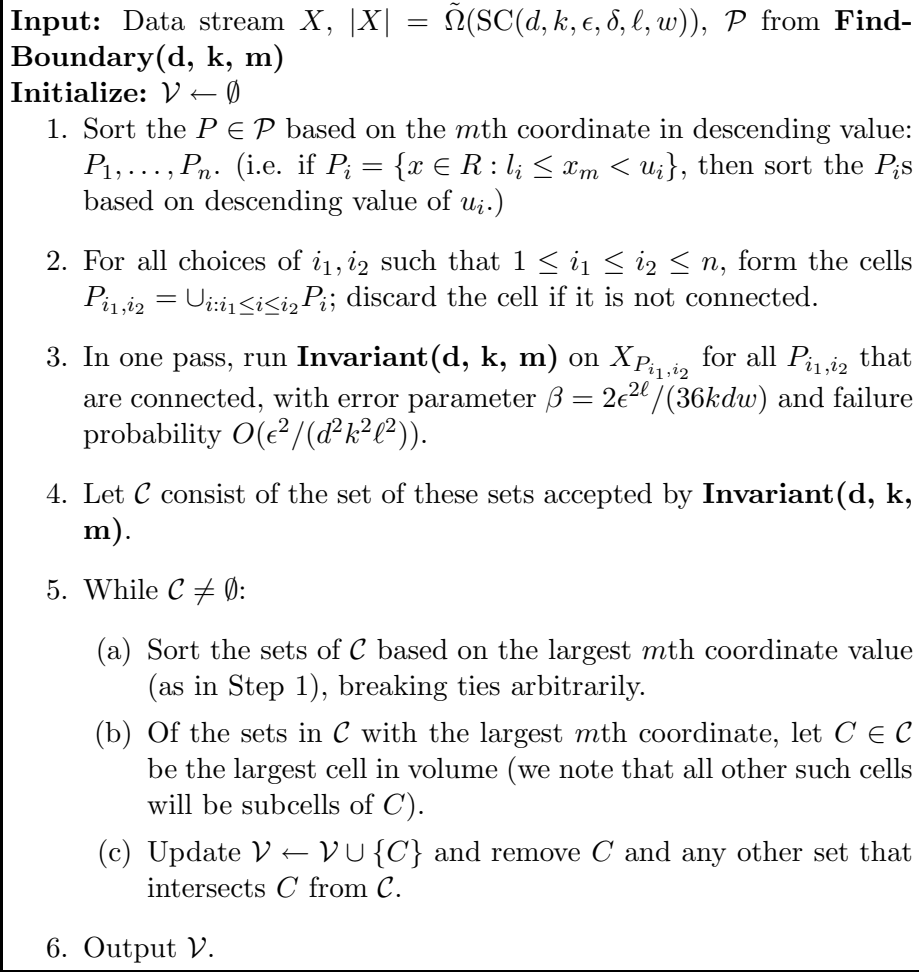


Figure 6: Algorithm **Postprocess**

□

Remark: The output of **FindBoundary**(\mathbf{d} , \mathbf{k} , \mathbf{m}) is the set \mathcal{P} of partition rectangles that were accepted by some call to **Invariant**(\mathbf{d} , \mathbf{k} , \mathbf{m}). Therefore, F is within β of an m th component invariant function for each cell $P \in \mathcal{P}$. Furthermore, the set $R \setminus (\cup_{P \in \mathcal{P}} P)$ contains at most $\epsilon^\ell/3d$ probability mass as we showed in the previous lemma.

We now consider the number of cells output by **FindBoundary**(\mathbf{d} , \mathbf{k} , \mathbf{m}), $|\mathcal{P}|$. Note that in the first iteration, the number of cells created by the algorithm is $|\mathcal{P}_1| = \Theta(dk/\epsilon)$. At each subsequent iteration, the aggregate number of cells created across all level p cells is at most $O(k/\epsilon)$. Therefore, $|\mathcal{P}| = O(\ell k/\epsilon + dk/\epsilon)$. We could output the set \mathcal{P} as the set to satisfy Theorem 1, but we would very much like to reduce the number of sets output to $O(k)$. This would significantly decrease the memory requirements of the overall learning algorithm.

Therefore, we will postprocess \mathcal{P} in a final pass over the data stream in order to reduce the number of intervals to $4k$. We describe this pass in Figure 6.

The next lemma shows that the output of **Postprocess**, \mathcal{V} , consists of a much smaller number

of cells.

Lemma 7 *With probability at least $1 - \delta/3$, after postprocessing, in each $V \in \mathcal{V}$, F is within β of invariant in the m th component and $|\mathcal{V}| \leq 4k$.*

Proof: Since the total number of cells \mathcal{C} created by the algorithm is at most $O(\ell^2 k^2 d^2 / \epsilon^2)$, and **Invariant**($\mathbf{d}, \mathbf{k}, \mathbf{m}$) was called on each cell, the probability that at least one call failed is at most $\delta/3$, from the union bound. We condition the remainder of the proof on no call failing.

Since each $V \in \mathcal{V}$ was accepted by **Invariant**($\mathbf{d}, \mathbf{k}, \mathbf{m}$), each V is within β of invariant in the m th component. Each set placed in $V \in \mathcal{V}$ must either

1. Contain an m th component boundary cell. Since none of the cells in \mathcal{V} intersect, there are at most $2k$ of these cells in \mathcal{V} .
2. Not contain such a boundary cell. Suppose P_{i_1, i_2} was a *connected* set that was created in Step 2 of **Postprocess**. We know that if it does not contain any m th component boundary cells, then **Invariant**($\mathbf{d}, \mathbf{k}, \mathbf{m}$) will accept P_{i_1, i_2} . Due to this fact and the fact that **Postprocess** takes in Step 5b the largest accepted cell, it follows that either **(a)** the previous cell placed in \mathcal{V} must have contained an m th component boundary cell, or **(b)** the previous cell placed in \mathcal{V} , V' , is not adjacent to V . This implies that between V and V' , there lies a cell that was rejected in the ℓ th level of **FindBoundary**($\mathbf{d}, \mathbf{k}, \mathbf{m}$), which has to contain an m th component boundary cell. Since there are at most $2k$ m th component boundary cells, there are a total of at most $2k$ of these cells of type **(a)** and **(b)** in \mathcal{V} .

□

We now have all the ingredients for the proof of Theorem 1.

Proof of Theorem 1: At the first level, there is only one copy of **FindBoundary**($\mathbf{d}, \mathbf{k}, \mathbf{m}$) running, which draws a sample of size $\tilde{O}(d^3 k^2 / \epsilon^2)$ from X and requires $O(kd/\epsilon)$ parallel calls to **Invariant**($\mathbf{d}, \mathbf{k}, \mathbf{m}$). Each of these calls requires at most $\tilde{O}(d\ell)$ memory. Therefore, the total memory required for the first level is $\tilde{O}(d^3 k^2 / \epsilon^2 + kd\ell/\epsilon)$.

At subsequent levels, there are at most $2k$ parallel instantiations of **FindBoundary**($\mathbf{d}, \mathbf{k}, \mathbf{m}$) running at any given moment. Each of these copies requires a sample of size $\tilde{\Theta}(d/\epsilon^2)$ and $O(1/\epsilon)$ parallel calls to **Invariant**($\mathbf{d}, \mathbf{k}, \mathbf{m}$). Therefore, the total memory requirement across all parallel instantiations is $\tilde{O}(kd/\epsilon^2 + kd\ell/\epsilon)$.

Lastly, **Postprocess** created $O(d^2 k^2 / \epsilon^2 + \ell^2 k^2 / \epsilon^2 + dk^2 \ell / \epsilon^2)$ cells in Step 3 and called **Invariant**($\mathbf{d}, \mathbf{k}, \mathbf{m}$) on each of these cells. Since this requires $\tilde{O}(d\ell)$ memory per call, the total memory requirement for **Postprocess** is therefore $\tilde{O}(\ell d^3 k^2 / \epsilon^2 + \ell^3 dk^2 / \epsilon^2 + d^2 k^2 \ell^2 / \epsilon^2)$.

Thus, the total amount of memory required by the algorithm is

$$\tilde{O}\left(\frac{\ell d^3 k^2}{\epsilon^2} + \frac{\ell^3 k^2 d}{\epsilon^2} + \frac{\ell^2 k^2 d^2}{\epsilon^2}\right).$$

The output of **Postprocess** is the set of sets \mathcal{V} , on which F is within β of a function that is invariant in the m th component, and such that $|\mathcal{V}| \leq 4k$. Naturally, we choose the function F_m that satisfies Theorem 1 to be this m th component invariant function on each set $V \in \mathcal{V}$, and 0 outside of $\cup_{V \in \mathcal{V}} V$. It then follows that:

$$\int_{\cup_{V \in \mathcal{V}} V} |F - F_m| = \sum_i \int_{V_i} |F - F_m| \leq 4k\beta \leq \frac{2\epsilon^{2\ell}}{9dw}.$$

The last part of the Theorem 1 follows from combining Lemma 7 with the fact that the set $R \setminus (\cup_{V \in \mathcal{V}} V)$ consists of the cells that were rejected in an ℓ th level iteration of **FindBoundary**(\mathbf{d} , \mathbf{k} , \mathbf{m}). There are at most $2k$ such cells, since each cell must contain at least one of the $2k$ m th boundary component. By Lemma 6, the total probability weight of these cells is at most $2k \cdot \epsilon^\ell / 6kd = \epsilon^\ell / 3d$. \square

4 Checking for m th component invariance

We now present the algorithm **Invariant**(\mathbf{d} , \mathbf{k} , \mathbf{m}) from Theorem 4. Recall that, given a data stream X containing samples from a mixture of k uniform distributions in \mathbb{R}^d with density function H , **Invariant**(\mathbf{d} , \mathbf{k} , \mathbf{m}) will accept if H is invariant in the m th component and will reject if H is not within distance β of an m th component invariant function. In order to ease our notation, we will assume without loss of generality that $m \leftarrow d$ and that the bounding box of H is given by $R = (0, b_1) \times (0, b_2) \times \dots \times (0, b_d)$, where $b_i \leq 1$.

Our exposition of **Invariant**(\mathbf{d} , \mathbf{k}) will compose of three parts: First, we define a sufficient condition for establishing that H is close to invariant in the d th component. We will then propose an estimator $\gamma_{\vec{j},i}$ and prove that $\gamma_{\vec{j},i}$ can be used to test the sufficient condition. Lastly, we give an algorithm for actually performing the test in a single pass with a small amount of memory. As in [3], the main algorithmic tool that we use is Indyk's one pass algorithm for computing the ℓ_1 length of a vector given as a stream of dynamic updates.

Definition 6 We define a *regular partition parallel to the d th component* \mathcal{P}_η to be the partition of the bounding box $R = (0, b_1) \times \dots \times (0, b_d)$ into $(1/\eta)^{d-1}$ d -cells $\{P_{\vec{j}}\}$: for all $\vec{j} \in [1/\eta]^{d-1}$,

$$P_{\vec{j}} = ((j_1 - 1)\eta b_1, j_1\eta b_1) \times \dots \times ((j_{d-1} - 1)\eta b_{d-1}, j_{d-1}\eta b_{d-1}) \times (0, b_d).$$

We will refer to the d -cells $P_{\vec{j}} \in \mathcal{P}_\eta$ as *partition cells*.

The partition is thus a partition of R into a set of d -cells with the same dimensions, such that each component except the d th is partitioned into $1/\eta$ intervals. Note that $\text{vol}(P_{\vec{j}}) = \eta^{d-1} \text{vol}(R)$ and that $|\mathcal{P}_\eta| = 1/\eta^{d-1}$.

Informally, each partition cell is a long, thin strip, with its long side along the d th component. The main idea is that if F is close to constant in most of these partition cells, then F should be close to invariant in the d th component.

More precisely, denote by

$$c_{\vec{j}} = \frac{\int_{P_{\vec{j}}} H}{\text{vol}(P_{\vec{j}})}.$$

Invariant(\mathbf{d} , \mathbf{k}) will check if the quantity $\alpha_{\vec{j}}$, defined by

$$\alpha_{\vec{j}} = \int_{P_{\vec{j}}} \left| H - \frac{\int_{P_{\vec{j}}} H}{\text{vol}(P_{\vec{j}})} \right| = \int_{P_{\vec{j}}} |H - c_{\vec{j}}|$$

is small. $\alpha_{\vec{j}}$ is the error of estimating $H(x)$ in $P_{\vec{j}}$ as simply the constant that is the average of H in $P_{\vec{j}}$.

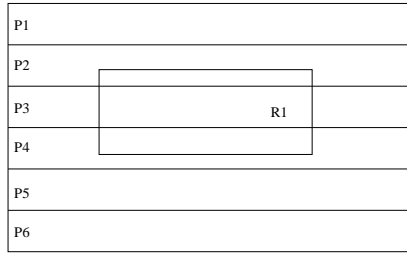


Figure 7: An example in \mathbb{R}^2 . Suppose we are testing to see if F in the bounding box varies with the horizontal component (i.e. if it contains any *vertical boundary edges* of mixture rectangles). Thus, $P_1, P_2, P_3, P_4, P_5, P_6$ constitute a regular partition parallel to the horizontal component. In this case, there is a mixture rectangle R_1 . P_2 and P_4 are **bad** partition cells (since they contain horizontal boundary edges), whereas all other partition cells are **good**. Note that in the \mathbb{R}^2 case, each horizontal boundary edge may only intersect one partition rectangle.

For the analysis, we need to classify partition cells as **good** or **bad**. Partition cell P is bad if it contains an m th component boundary cell of a mixture rectangle, for $m < d$. Otherwise, P is good. Note that a good P may contain d th component boundary cells. See Figure 7.

Let $\mathcal{G} \subset \mathcal{P}_\eta$ be the set of good partition cells and $\mathcal{B} = \mathcal{P}_\eta \setminus \mathcal{G}$ be the set of bad partition cells.

Lemma 8 $|\mathcal{B}| \leq \frac{2kd}{\eta^{d-2}}$

Proof: Any boundary cell of a mixture rectangle, except d th component boundary cells, can intersect at most $1/\eta^{d-2}$ partition cells. There are at most k mixture cells, each of which has $2d$ boundary cells. \square

The above lemma shows that the number of bad partition cells is small; therefore the aggregate volume of bad partition cells is negligible. We use this fact in the proof of the next Lemma.

Lemma 9 Suppose that $\eta \leq \frac{\beta}{4 \cdot k \cdot d \cdot w}$. If there exist constants $c_{\vec{j}}$ for all $P_{\vec{j}} \in \mathcal{G}$ such that

$$\sum_{P_{\vec{j}} \in \mathcal{G}} \alpha_{\vec{j}} = \sum_{P_{\vec{j}} \in \mathcal{G}} \int_{P_{\vec{j}}} |H(x) - c_{\vec{j}}| \leq \frac{\beta}{4},$$

then there exists a function \tilde{H} that is invariant in the d th component such that

$$\int_R |H(x) - \tilde{H}(x)| \leq \beta.$$

Proof: Consider choosing the function $\tilde{H}(x)$ as:

$$\tilde{H}(x) = \begin{cases} c_{\vec{j}} & \text{if } x \in P_{\vec{j}} \text{ such that } P_{\vec{j}} \in \mathcal{G}. \\ 0 & \text{if } x \in P_{\vec{j}} \text{ such that } P_{\vec{j}} \in \mathcal{B}. \end{cases}$$

Since $H(x) \leq w$ for all $x \in R$, $\int_{P_{\vec{j}}} |H(x) - \tilde{H}(x)| dx \leq w \text{vol}(P_{\vec{j}})$ for any $P_{\vec{j}}$. Therefore,

$$\begin{aligned} \int_R |H(x) - \tilde{H}(x)| dx &= \sum_{P_{\vec{j}} \in \mathcal{G}} \int_{P_{\vec{j}}} |H(x) - \tilde{H}(x)| + \sum_{P_{\vec{j}} \in \mathcal{B}} \int_{P_{\vec{j}}} |H(x) - \tilde{H}(x)| \\ &\leq \sum_{P_{\vec{j}} \in \mathcal{G}} \alpha_{\vec{j}} + \sum_{P_{\vec{j}} \in \mathcal{B}} w \cdot \text{vol}(P_{\vec{j}}) \\ &\leq \frac{\beta}{4} + \frac{2kd}{\eta^{d-2}} \cdot w \cdot \eta^{d-1} \leq \beta, \end{aligned}$$

where the second to last inequality follows from Lemma 8 and the fact that the volume of any partition cell is $\text{vol}(P_{\vec{j}}) = \eta^{d-1} \text{vol}(R) \leq \eta^{d-1}$. \square

4.1 Estimating $\alpha_{\vec{j}}$ from the data stream

We now describe an estimator for $\alpha_{\vec{j}}$ in a good partition cell $P_{\vec{j}} \in \mathcal{P}_\eta$, and prove properties of the estimator. Note that we do not provide an algorithm for computing this estimator until Section 4.2.

Recall that $P_{\vec{j}} = ((j_1 - 1)\eta b_1, j_1 \eta b_1) \times \dots \times ((j_{d-1} - 1)\eta b_{d-1}, j_{d-1} \eta b_{d-1}) \times (0, b_d)$ for the vector $\vec{j} \in [1/\eta]^{d-1}$. Let $\zeta > 0$ (assume that $1/\zeta$ is an integer). We further partition $P_{\vec{j}} \in \mathcal{P}_\eta$ into $1/\zeta$ d -cells of equal volume.

Definition 7 For each integer $i \in [1/\zeta]$, define the **sub-partition** d -cell $P_{\vec{j},i}$ by

$$P_{\vec{j},i} = \left\{ x \in P_{\vec{j}} \mid (i-1)\zeta b_d \leq x_d \leq i\zeta b_d \right\}.$$

Recall that $N = |X|$ is the number of samples in the data stream. We define the following random variables:

1. $N_{\vec{j},i} = |X \cap P_{\vec{j},i}|$
2. $\gamma_{\vec{j},i} = \frac{N_{\vec{j},i} - \zeta \sum_l N_{\vec{j},l}}{N}$

$N_{\vec{j},i}$ is the number of samples that lie in $P_{\vec{j},i}$; since $\zeta \sum_l N_{\vec{j},l}$ is the average number of points in each of the sub-partition cells of $P_{\vec{j}}$, $\gamma_{\vec{j},i}$ is the difference between $N_{\vec{j},i}$ and what we would expect if H were actually constant in $P_{\vec{j}}$. Therefore, if $\sum_i \gamma_{\vec{j},i}$ is small, then $\alpha_{\vec{j}}$ should be small as well:

Lemma 10 Let $\zeta \leq \frac{\beta}{64k \cdot w}$ and fix $P_{\vec{j}} \in \mathcal{G}$. If $|X| = \Omega\left(\frac{d^2}{\beta^2 \zeta^2 \eta^{2d-2}} \log(1/\zeta \beta \delta \eta)\right)$, then with probability at least $1 - \delta/2$, $\sum_i \gamma_{\vec{j},i} \geq \alpha_{\vec{j}} - \frac{\beta \eta^{d-1}}{8}$.

Proof: Let $c_{\vec{j}} = \zeta \sum_i N_{\vec{j},i}$. The VC dimension of the set of all d -cells is $2d$. Since X is drawn according to \tilde{H} , we have chosen our sample complexity so that the VC bound implies that

$$\Pr \left[\max_{\vec{j},i} \left| \frac{N_{\vec{j},i}}{N} - \int_{P_{\vec{j},i}} \tilde{H} \right| \leq \frac{\beta \cdot \zeta \cdot \eta^{d-1}}{16} \right] \geq 1 - \frac{\delta}{2}.$$

Since $P_{\vec{j}} \in \mathcal{G}$, at most $2k$ boundary d -cells of mixture rectangles can intersect it: d th component boundary cells of mixture rectangles, which are constant in the d th component and can thus be

written as $(a_1^i, b_1^i) \times \dots \times (a_{d-1}^i, b_{d-1}^i) \times (b^i, b^i)$ for appropriate choices of a_j^i s and b_j^i s. Note that any such boundary cell can only intersect $P_{\vec{j}}$ in one of its sub-partition cells $P_{\vec{j},i}$. Thus, at most $2k$ of the sub-partition cells contain any boundary cell.

Recall the definition of $\alpha_{\vec{j}}$:

$$\alpha_{\vec{j}} = \int_{P_{\vec{j}}} |H(x) - c_{\vec{j}}| = \sum_i \int_{P_{\vec{j},i}} |H(x) - c_{\vec{j}}|.$$

We will analyze the quantity $\int_{P_{\vec{j},i}} |H(x) - c_{\vec{j}}|$ separately for each sub-partition cell $P_{\vec{j},i}$, in two separate cases: sub-partition cells that do not contain any boundary cells, and sub-partition cells that do.

Case 1: A sub-partition cell that does not contain any boundary cells of the mixture rectangles. For such cells, H is constant. Thus,

$$\begin{aligned} \int_{P_{\vec{j},i}} |H(x) - c_{\vec{j}}| dx &= \left| \int_{P_{\vec{j},i}} (H(x) - c_{\vec{j}}) dx \right| \\ &\leq \left| \int_{P_{\vec{j},i}} H(x) dx - \frac{N_{\vec{j},i}}{N} \right| + \left| \int_{P_{\vec{j},i}} \left(\frac{N_{\vec{j},i}}{N} - c_{\vec{j}} \right) \right| \\ &\leq \gamma_{\vec{j},i} + \frac{\beta \zeta \eta^{d-1}}{16}. \end{aligned}$$

Case 2: A sub-partition cell $P_{\vec{j},i}$ that contains a boundary cell. There are at most $2k$ of these. Since $H(x) \leq w$ for all x , $\int_{P_{\vec{j},i}} |H(x) - c_{\vec{j}}| \leq 2w \cdot \text{vol}(P_{\vec{j},i}) \leq 2w\zeta\eta^{d-1}$.

Thus, we sum over all sub-partition cells to get a bound on $\gamma_{\vec{j}}$ in terms of $\alpha_{\vec{j}}$:

$$\begin{aligned} \alpha_{\vec{j}} &= \int_{P_{\vec{j}}} |H(x) - c_{\vec{j}}| \\ &= \sum_{P_{\vec{j},i}:\text{case 1}} \int_{P_{\vec{j},i}} |H(x) - c_{\vec{j}}| + \sum_{P_{\vec{j},i}:\text{case 2}} \int_{P_{\vec{j},i}} |H(x) - c_{\vec{j}}| \\ &\leq \sum_i \left(\gamma_{\vec{j},i} + \frac{\beta \cdot \zeta \cdot \eta^{d-1}}{16} \right) + 4kw\zeta\eta^{d-1} \\ &= \frac{\beta\eta^{d-1}}{8} + \sum_i \gamma_{\vec{j},i}. \end{aligned}$$

□

Corollary 11 *With probability at least $1 - \delta/2$, if $\sum_{\vec{j}} \sum_i \gamma_{\vec{j},i} \leq \beta/8$, then there exists a function \tilde{H} that is invariant in the d th component such that $\int_R |H(x) - \tilde{H}(x)| \leq \beta$.*

Proof: The corollary follows immediately from summing the $\gamma_{\vec{j},i}$ s and applying the previous two lemmas. □

Lemma 12 If $X = \Omega\left(\frac{d^2}{\beta^2 \zeta^2 \eta^{2d-2}} \log(1/\zeta\beta\delta\eta)\right)$, then the following is true with probability at least $1 - \delta/4$. If H is invariant in the d th component, then $\sum_{\vec{j}} \sum_i \gamma_{\vec{j},i} \leq \frac{\beta}{16}$.

Proof: If H is invariant in the d th component, then $\mathbb{E}\left[\frac{N_{\vec{j},i}}{N}\right] = \mathbb{E}\left[\frac{\zeta \sum_i N_{\vec{j},i}}{N}\right] = \zeta \int_{P_{\vec{j}}} H$ for all \vec{j}, i .

By applying the VC bound, we know that

$$\Pr\left[\max_{\vec{j},i} \left| \frac{N_{\vec{j},i}}{N} - \zeta \int_{P_{\vec{j}}} H \right| \leq \frac{\beta \cdot \zeta \cdot \eta^{d-1}}{32}\right] \geq 1 - \delta/8$$

and that

$$\Pr\left[\max_{\vec{j}} \left| \zeta \sum_i \frac{N_{\vec{j},i}}{N} - \zeta \int_{P_{\vec{j}}} H \right| \leq \frac{\beta \cdot \zeta \cdot \eta^{d-1}}{32}\right] \geq 1 - \delta/8.$$

Thus, with probability at least $1 - \delta/4$,

$$\gamma_{\vec{j},i} = \left| \frac{N_{\vec{j},i}}{N} - \frac{\zeta \sum_l N_{\vec{j},l}}{N} \right| \leq \left| \frac{N_{\vec{j},i}}{N} - \zeta \int_{P_{\vec{j}}} H \right| + \left| \zeta \int_{P_{\vec{j}}} H - \zeta \sum_i \frac{N_{\vec{j},i}}{N} \right| \leq \frac{\beta \cdot \zeta \cdot \eta^{d-1}}{16}$$

for all \vec{j}, i . The lemma follows by summing over \vec{j} and i . \square

4.2 A one pass, small space algorithm

Corollary 11 and Lemma 12 prove that an algorithm that **accepts** if $\sum_{\vec{j}} \sum_i \gamma_{\vec{j},i} \leq \beta/16$ and **rejects** if $\sum_{\vec{j}} \sum_i \gamma_{\vec{j},i} \geq \beta/8$, then it will accept if H is invariant in the d th component, and will reject if H is not within β of an invariant function.

A naive one pass algorithm to compute the estimator $\sum_{\vec{j},i} \gamma_{\vec{j},i}$ would explicitly keep one counter for each of the $1/(\eta \cdot \zeta) N_{\vec{j},i}$ s. This approach requires on the order of $(k \cdot w \cdot d/\beta)^d$ memory, which is far too much. With more powerful algorithmic tools, we can reduce the memory requirement to polylogarithmic in β, k, w and linear in d .

4.2.1 Approximating the length of a vector given as a stream of dynamic updates

Indyk [9] designed a one-pass algorithm for approximating the ℓ_1 length of a vector given as a stream of dynamic updates (very similar to the histogram problem mentioned in the related works section). First, we initialize a vector $v \in \mathbb{R}^d$: $v \leftarrow \vec{0}$. The input is a stream of update pairs $\langle a, i \rangle$, where $a \in [-M, M]$ and $i \in [n]$, that represent the semantics: add a to the i th component of vector $\vec{v} \in \mathbb{R}^n$. The problem is then to approximate $\|\vec{v}\|_1 = \sum_{i=1}^n |v_i|$ after processing all of the input pairs. The following theorem is an adaptation of a more general result:

Theorem 13 (Indyk[9]) *There exists a one pass algorithm that, with probability at least $1 - \delta$, will find an approximation ι such that $\frac{2}{3}\|\vec{v}\|_1 \leq \iota \leq \frac{4}{3}\|\vec{v}\|_1$ using at most $O(\log M \log(n/\delta))$ bits of memory and $O(\log(n/\delta))$ per-element update time.*

The high level idea of this algorithm is that instead of storing all n components of \vec{v} , it stores the components of a random projection of \vec{v} to a low dimensional subspace. The random matrix that defines the projection is compressed by only storing the seed of a pseudorandom number generator; the entries of the matrix are generated as needed during the pass.

We present in Figure 8 the details of our algorithm **Invariant(d, k)**.

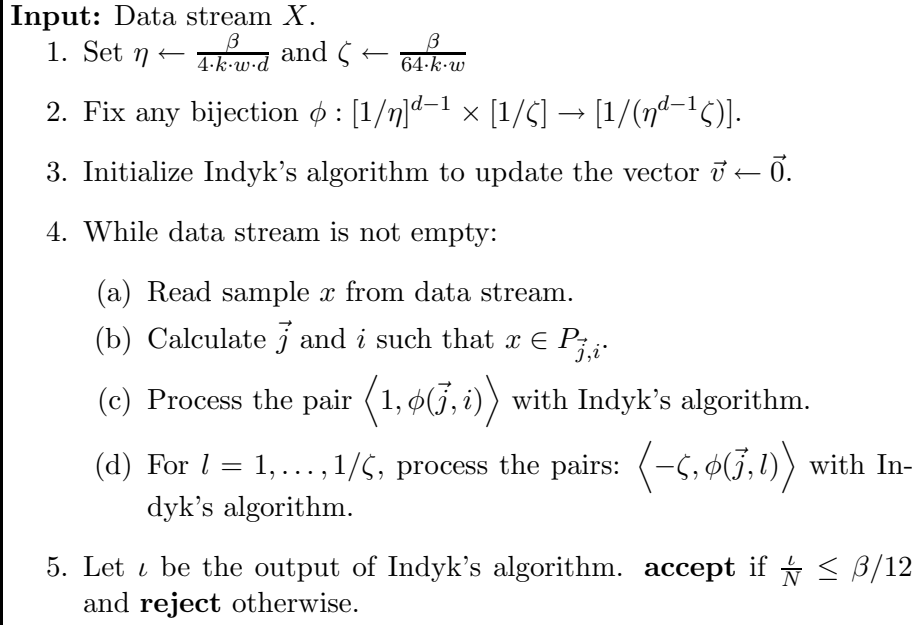


Figure 8: Algorithm **Invariant(d, k)**

Proof of Theorem 4: Due to the guarantees of Indyk's algorithm, $\frac{\iota}{N}$ will satisfy:

$$\frac{2}{3} \frac{\|v\|_1}{N} \leq \frac{\iota}{N} \leq \frac{4}{3} \frac{\|v\|_1}{N}.$$

Note that the $\phi(\vec{j}, i)$ th component of v is exactly: $v_{\phi(\vec{j}, i)} = N_{\vec{j}, i} - \zeta \sum_l N_{\vec{j}, l}$. Thus, $\frac{\|v\|_1}{N} = \sum_{\vec{j}} \sum_i \gamma_{\vec{j}, i}$. Therefore, the algorithm will accept if $\sum_{\vec{j}} \sum_i \gamma_{\vec{j}, i} \leq \beta/16$ and will reject if $\sum_{\vec{j}} \sum_i \gamma_{\vec{j}, i} > \beta/8$. The correctness of the algorithm follows from Corollary 11 and Lemma 12.

The memory usage and per-element update time of **Invariant(d, k)** are derived from the guarantees of Indyk's algorithm as stated in Theorem 13 by substituting the parameters $M \leftarrow |X|$ and $n \leftarrow 1/(\eta^{d-1}\zeta)$. \square

References

- [1] N. Alon, Y. Matias, and M. Szegedy. The space complexity of approximating the frequency moments. *Journal of Computer and System Sciences*, 58(1):137–147, 1999.
- [2] S. Arora and R. Kannan. Learning mixtures of separated nonspherical Gaussians. *Annals of Applied Probability*, 15(1A):69–92, 2005.
- [3] K. Chang and R. Kannan. The space complexity of pass-efficient algorithms for clustering. In *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1157–1166, 2006.
- [4] A. Dasgupta, J. E. Hopcroft, J. M. Kleinberg, and M. Sandler. On learning mixtures of heavy-tailed distributions. In *Proceedings of the 46th IEEE Symposium on Foundations of Computer Science*, pages 491–500, 2005.

- [5] S. Dasgupta. Learning mixtures of Gaussians. In *Proceedings of the 40th IEEE Symposium on Foundations of Computer Science*, pages 634–644, 1999.
- [6] A. C. Gilbert, S. Guha, P. Indyk, Y. Kotidis, S. Muthukrishnan, and M. Strauss. Fast, small-space algorithms for approximate histogram maintenance. In *Proceedings of the 34th Annual ACM Symposium on the Theory of Computing*, pages 389–398, 2002.
- [7] S. Guha and A. McGregor. Space efficient sampling. In *Proceedings of the Eleventh International Conference on Artificial Intelligence and Statistics*, 2007.
- [8] S. Guha, A. McGregor, and S. Venkatasubramanian. Streaming and sublinear approximation of entropy and information distances. In *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 733–742, 2006.
- [9] P. Indyk. Stable distributions, pseudorandom generators, embeddings, and data stream computation. *Journal of the Association for Computing Machinery*, 53(3):307–323, 2006.
- [10] R. Kannan, H. Salmasian, and S. Vempala. The spectral method for general mixture models. In *Proceedings of the 18th Annual Conference on Learning Theory*, pages 444–457, 2005.
- [11] J. I. Munro and M. Paterson. Selection and sorting with limited storage. *Theoretical Computer Science*, 12:315–323, 1980.
- [12] M. Talagrand. Sharper bounds for Gaussian and empirical processes. *Annals of Probability*, 22(1):28–76, 1994.
- [13] N. Thaper, S. Guha, P. Indyk, and N. Koudas. Dynamic multidimensional histograms. In *Proceedings of the 2002 ACM SIGMOD international conference on Management of data*, pages 428–439, New York, NY, USA, 2002. ACM Press.
- [14] V. Vapnik and A. Červonenkis. On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probability and its Applications*, 16:264–280, 1971.
- [15] S. Vempala and G. Wang. A spectral algorithm for learning mixtures of distributions. *Journal of Computer and System Sciences*, 68(4):841–860, 2004.

A Proof of Property 1 in Section 2.2

The proof of the property involves tools that reduce the dimension of the problem by 1. We therefore make the following definitions to ease the notation:

Definition 8 If $x \in \mathbb{R}^d$, we denote by x_{-i} the vector in \mathbb{R}^{d-1} defined by $x_{-i} = (x_1, \dots, x_{i-1}, x_{i+1}, \dots, x_d)$.

Definition 9 For $x \in \mathbb{R}^d$, let $\phi^m : \mathbb{R}^d \times \mathbb{R} \rightarrow \mathbb{R}^d$ be defined by $\phi^m(x, y) = (x_1, \dots, x_{m-1}, y, x_{m+1}, \dots, x_d)$ (in words, $\phi^m(x, y)$ has the value y in the m th component and the same value as x in every other component).

Property 3

$$\int_{R_i} |F - \bar{F}_{R_i}| \leq \left(2w \sum_{m=1}^d \int_{R_i} |F_m - F| \right)^{\frac{1}{2}}$$

Proof: We first prove a claim.

Claim 2 *There exists a constant c_{R_i} such that*

$$\int_{\tilde{R}_i} |F - c_{R_i}| \leq 2 \sum_{m=1}^d \int_{R_i} |F_m - F|. \quad (2)$$

Proof: Recall that all F_m are invariant in the m th component in R_i .

We prove the claim by induction. We assume that there exists a function H_m such that $\int_{R_i} |F - H_m| \leq 2 \sum_{j=1}^m \int_{R_i} |F - F_j|$ and H_m is invariant in the first m components, which means that $H_m(x)$ does not depend on the first m components of x . Suppose $R_i = (r_1, s_1) \times \dots \times (r_d, s_d)$. Define $R_i^{-(m+1)} = (r_1, s_1) \times \dots \times (r_m, s_m) \times (r_{m+2}, s_{m+2}) \times \dots \times (r_d, s_d) \subset \mathbb{R}^{d-1}$. For any scalar c , since F_{m+1} does not vary with the $m+1$ th component, the function:

$$g(x_{m+1}) = \int_{R_i^{-(m+1)}} |H_m(\phi^{m+1}(x, c)) - F_{m+1}(x)| dx_{-(m+1)}$$

is constant. Choose c^* to be the constant c that minimizes this quantity and define $H_{m+1}(x) = H_m(\phi^{m+1}(x, c^*))$. Since neither H_{m+1} nor F_{m+1} vary with the $m+1$ th component,

$$\begin{aligned} \int_{R_i} |H_{m+1}(x) - F_{m+1}(x)| dx &= \int_{r_{m+1}}^{s_{m+1}} \min_{c^*} \int_{R_i^{-(m+1)}} |H_m(\phi^{m+1}(x, c^*)) - F_{m+1}| dx_{-(m+1)} dx_{m+1} \\ &\leq \int_{R_i} |H_m(x) - F_{m+1}(x)| dx. \end{aligned}$$

By applying the triangle inequality and the inductive hypothesis, we get:

$$\begin{aligned} \int_{R_i} |F - H_{m+1}| &\leq \int_{R_i} |F - F_{m+1}| + \int_{R_i} |F_{m+1} - H_{m+1}| \\ &\leq \int_{R_i} |F - F_{m+1}| + \int_{R_i} |H_m - F_{m+1}| \\ &\leq \int_{R_i} |F - F_{m+1}| + \int_{R_i} |H_m - F| + \int_{R_i} |F - F_{m+1}| \\ &\leq \sum_{j=1}^{m+1} 2 \int_{R_i} |F - F_j|. \end{aligned}$$

□

We would like to use the existence of this constant to prove a bound on $\int_{R_i} |F - \bar{F}_{R_i}|$. One problem that we encounter is that $\arg \min_c \int_{R_i} |F - c| \neq \bar{F}_{R_i}$. However, it is true that

$$\arg \min_c \int_{R_i} (F - c)^2 = \bar{F}_{R_i}.$$

This can be proved by checking the first and second order necessary and sufficient conditions: If $L(c) = \int_{R_i} (F - c)^2$, then

$$\frac{\partial L}{\partial c} = -2 \int_{R_i} (F - c).$$

Thus, $\partial L/\partial c = 0$ when $c^* = \int_{R_i} F/\text{vol}(R_i)$. The second order condition

$$\frac{\partial^2 L}{\partial c^2} = 1 > 0.$$

implies that c^* is the global minimum.

Thus, \bar{F}_{R_i} minimizes the L^2 error norm induced by estimating F as a constant on R_i . We now relate the L^2 error with the L^1 error. For any constant c , a simple application of Hölder's inequality implies that

$$\int_{R_i} |F - c| \leq \left(\int_{R_i} (F - c)^2 \right)^{\frac{1}{2}}.$$

Also, recall that we assumed that $0 \leq F(x) \leq w$ for all $x \in R$. Therefore, for c such that $0 \leq c \leq w$,

$$\left(\int_{R_i} (F - c)^2 \right)^{\frac{1}{2}} \leq \left(\int_{R_i} w |F - c| \right)^{\frac{1}{2}} \leq \left(w \int_{R_i} |F - c| \right)^{\frac{1}{2}}.$$

Combining the above observations:

$$\begin{aligned} \int_{R_i} |F - \bar{F}_{R_i}| &\leq \left(\int_{R_i} (F - \bar{F}_{R_i})^2 \right)^{\frac{1}{2}} \\ &\leq \left(\int_{R_i} (F - c_{R_i})^2 \right)^{\frac{1}{2}} \\ &\leq \left(2w \sum_{m=1}^d \int_{R_i} |F - F_m| \right)^{\frac{1}{2}}, \end{aligned}$$

where the second inequality follows from the fact that $\bar{F}_{R_i} = \arg \min_c \int_{R_i} (F - c)^2$. \square