

A Pass-Efficient Algorithm for Clustering Census Data

Kevin Chang ^{*}
Yale University

Ravi Kannan [†]
Yale University

Abstract

We present a number of streaming algorithms for a basic clustering problem for massive data sets. In this problem, we are given a set of n points drawn randomly according to a mixture of k uniform distributions, and wish to approximate the density function of the mixture. The points are placed in a datastream (possibly in adversarial order), which may only be read sequentially by the algorithm. We argue that this realistically models, among others, the datastream produced by a national census of the incomes of all citizens. Our algorithms conform to the pass-efficient model of computation; thus one of them has the novelty that it may make several passes over the data. The power of this approach is made apparent by the following property of our main algorithm: If the algorithm is allotted $2l$ passes, it will produce an approximation with error at most ϵ^l , using at most $O^*(k^3/\epsilon^2)$ read-write memory, the most critical resource for streaming computation. Thus, we demonstrate that more passes allow much sharper approximations, holding the amount of RAM used constant.

1 Introduction

The streaming model of computation has received much attention in the recent literature. In this model, the input to an algorithm may only be read in a single, sequential pass over the data; no random access of the input is allowed. Furthermore, the amount of RAM used by the algorithm must be small (typically $o(n)$, where n is the size of the datastream). The streaming model fits well with the constraints of computation with massive data sets, where the size of the input is much too large to fit into main memory. The data must be placed in secondary storage (for instance a tape, or disk that must be read in entire blocks), which may only be accessed sequentially.

The *pass-efficient* model has been proposed by Drineas and Kannan [7] as a more flexible version of the streaming model. Intuitively, the rigid restriction of the streaming model to a single pass over the data unnecessarily limits its potential utility. The pass-efficient model allows for multiple sequential passes over the input (ideally a small, perhaps constant, number of passes), and a small amount of extra space. While processing each element of the datastream, the algorithm may only use computing time that is independent of n , but after each pass, it is allowed more computing time (typically $o(n)$). In this model of computation, we are most concerned with two resources: *the number of passes* and *additional storage space (RAM)*. The model has been applied to a two pass algorithm for computing a succinct, approximate representation of a matrix [7].

In this paper, we consider a new clustering problem. Our problem, *learning a mixture of k uniform distributions*, is roughly stated as follows: Given a set of points drawn from a mixture of distributions (to be defined precisely below), compute the clusters (i.e. density function of the mixture) from which the points were drawn. The main motivation for our problem arises from considerations in the analysis of census data. Census data can naturally be modeled as a mixture

^{*}kevin.chang@yale.edu. Supported by NSF grant CCR-0331548.

[†]kannan@cs.yale.edu. Supported in part by NSF grant CCR-0310805.

of uniform distributions. Consider for instance the data set consisting of the personal incomes of a sampled set of citizens or, as in the case of the United States census, in principle all citizens. The incomes of individuals engaged in any single profession are certainly not exactly the same, but rather are distributed over some range of incomes. A reasonable first step in modeling such distributions is to assume that the incomes in a given profession are uniformly distributed over some interval. Thus, suppose profession i corresponds to the range of incomes (a_i, b_i) , and represents w_i proportion of the population. A sample drawn for the census can be viewed as distributed according to a mixture of uniform distributions in the following way: with probability w_i , the individual sampled belongs to profession i , in which case the individual's income will fall in the interval (a_i, b_i) . The whole of the census data may be adversarially ordered.

An interesting question to ask is: *assuming the census data is distributed as a mixture of uniform distributions, can we compute the distribution of incomes?* Since the amount of data in a census is, in principle, very large, the pass-efficient/streaming model of computation is appropriate.

More formally, a *mixture of k uniform distributions* is defined by k possibly intersecting intervals (a_i, b_i) for $i = 1, \dots, k$ and a corresponding *mixing weight* $w_i \geq 0$ for each interval, such that $\sum w_i = 1$. We assume that the mixture is not degenerate: $-\infty < a_i < b_i < \infty$, for all i . A mixture of k uniform distributions is a probability distribution on \mathbb{R} defined by choosing the i 'th interval with probability w_i , and then picking a point uniformly at random from the chosen interval.

Suppose that X is a set of n points randomly chosen from \mathbb{R} according to a mixture of k -uniform distributions with density function F . X can be ordered (possibly adversarially) to produce a sequence which constitutes the datastream. Our problem is then: Design a pass-efficient algorithm that approximates F from the datastream X .

Standard methods including Vapnik-Cervonenkis arguments can be used to show that from a random sample of $O^*(k^2/\epsilon^2)$ points from X , we can learn the mixture within error ϵ (see Section 3 for precise statements). The main result of our paper is that with multiple passes, we can gain significant accuracy without increasing the amount of RAM. Alternatively, the benefit of multiple passes can be viewed as holding approximation quality constant while decreasing the necessary amount of RAM; increasing the number of passes significantly decreases the RAM needed.

We now describe the general technique for our multiple pass algorithm. In a single pass, we partition the domain into a set of intervals, based on samples of the datastream. We estimate the density function F on each interval as follows. In a second pass, we count the number of points in X that lie in each interval, and run subroutine **Constant?**, which determines whether or not F is (approximately) constant on each interval. If F is constant on an interval I , then the density of F can be estimated easily by the number of points of X that fall in the interval. If F is not constant on I , the algorithm recursively estimates the density in the interval using subsequent passes, in effect "zooming in" on these more troublesome intervals until we are satisfied with our approximation.

A crucial part of the algorithm is subroutine **Constant?**, which solves a problem of independent interest: determining in a single pass over X whether or not F is approximately constant on some interval. For any $\beta > 0$, **Constant?** uses only $O(k)$ RAM and determines whether F is within β in L^1 distance of the uniform distribution, provided n is large enough ($\Omega^*((5k)^{8k+3}/\beta^{8k-4})$). Note that the space required is independent of β . For technical reasons, it is necessary to slightly perturb the data in X so that each point is drawn from an η -smoothed distribution, H , which is very close to F in variational distance. In a single pass, we can estimate the first $4k - 3$ raw moments of the perturbed distribution. If any of these moments deviate significantly from what we would expect if H were constant, then we know that F is not constant. Otherwise, we know that F is approximately constant; this will follow from our critical result: namely that the conditioning of the matrix of moments of an η -smoothed distribution is well-behaved.

1.1 Our Results

The L^1 distance with respect to the usual Lebesgue measure between measurable functions f and g is defined as $\int_{\mathbb{R}} |f - g|$. Given $\epsilon > 0$ and $\delta > 0$, we describe various randomized pass-efficient algorithms that find with probability at least $1 - \delta$ a function G that is an approximation to F , with error measured by L^1 distance: $\int_{\mathbb{R}} |F - G|$. Specifically, we present the following results:

1. In Section 3, a one pass algorithm with error at most ϵ that requires $O^*(k^2/\epsilon^2)$ RAM.
2. In Section 4, a $2l$ pass algorithm with error at most ϵ^l that uses $O^*(k^3/\epsilon^2)$ RAM, for any integer $l > 0$.
3. In Section 5, a four pass algorithm with error at most ϵ^2 that uses $O^*(k^3/\epsilon^2)$ RAM, but has a better sample complexity than the algorithm above when $l = 2$.

These results can alternatively be viewed as showing that making more passes over the data allows our algorithms to use less space, thus trading one resource in the pass-efficient model for the other. Viewed in this manner, the $2l$ pass algorithm will have an error of at most ϵ using $O^*(k^3/\epsilon^{2/l})$ space. This feature demonstrates the potential for multiple passes in designing a streaming algorithm with flexible performance guarantees on space needed and passes used; the algorithm can be tailored to the specific needs and limitations of a given application and system configuration.

1.2 Related Work

One pass streaming algorithms have been designed for a wide array of problems, including clustering. In contrast to our approximation schemes, these algorithms give constant factor (or weaker) approximations; for our purposes and motivations, such coarse approximations may not be acceptable. Charikar *et al.* [4] gave a streaming algorithm that achieved a constant factor approximation to the k -center problem, using $O(k)$ extra space. Guha *et al.* [14] designed a streaming algorithm that gave a factor $2^{O(1/\epsilon)}$ approximation to the k -median problem that uses at most $O(n^\epsilon)$ extra space, which is sublinear. Charikar *et al.* [5] improved this result, giving a constant factor approximation algorithm that uses only a poly-logarithmic amount of extra space.

Another related problem that has been heavily studied in the theory and database literature is the problem of histogram construction [1,9,12,15,16]. In the datastream version of this problem [13], one is given a stream of n real numbers a_1, \dots, a_n in this order, and the aim is to construct a piecewise constant function F with k pieces such that $\sum |F(i) - a_i|^2$ is minimized. Guha *et al.* [13] give a one pass $1 + \epsilon$ factor approximation algorithm using space $O(k^2/\epsilon \log n)$. A variant of the problem is that of histogram maintenance [10, 11] in which the a_i 's are presented as a stream of updates of the form “add 7 to a_3 ”, and the algorithm must always be able to output a histogram for the current state of the a_i 's. Gilbert *et al.* [11] provide a one pass $1 + \epsilon$ approximation algorithm that uses time per update and total space polynomial in $(k, 1/\epsilon, \|a\|_1$ or $\|a\|_2, \log n)$. Although these problems are similar to ours, there are some major distinctions. In the construction problem, elements in the datastream are presented in sorted order, whereas our datastream may be adversarially ordered. Furthermore for both the construction and maintenance problems, the function defined by the a_i 's is presented to the algorithm explicitly by its values or by sums of explicitly given values, whereas in our case the density function must be implicitly inferred from a sample drawn according to the mixture. Lastly, the algorithm of [11] uses much time per update and space (both polynomial in $(1/\epsilon, k, \log n)$), whereas our algorithm uses only $O(k \log k \log(1/\epsilon))$ time to process each element in the datastream, and significantly less space (in fact $O^*(k^3/\epsilon^{2/l})$ for an ϵ approximation if allowed $2l$ passes).

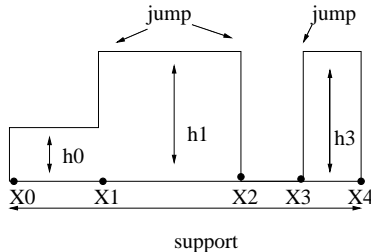


Figure 1: A mixture of 3 uniform distributions, labeled as a step function with 4 steps. Note that $h_2 = 0$ is not labeled; it is the density of the third step.

Algorithms for learning mixtures of Gaussian distributions in high dimension have been studied extensively. A mixture of Gaussians is the same as our mixture of uniform distributions, except each distribution in the mixture is a high dimensional Gaussian rather than a one dimensional uniform distribution. The objective is generally to learn the mean, covariance matrix, and mixing weight of each Gaussian in the mixture. Known algorithms make various assumptions regarding the Gaussians: DasGupta [6] assumes that the Gaussians are “well-separated” and spherical, Arora and Kannan [2] generalized this to non-spherical Gaussians, and Vempala and Wang [19] assume that they are spherical, in addition to their satisfying a weak separation criterion. These algorithms assume that the samples are ordered randomly, in contrast to our algorithm in which samples are ordered adversarially.

2 Preliminaries

We first give an alternate description of the structure of the probability density function induced by a mixture of uniform distributions.

Lemma 1 *Given a mixture of k uniform distributions $(a_i, b_i), w_i$, let F be the induced probability density function on \mathbb{R} . F (except possibly at at most $2k$ points) can be represented as a collection of at most $2k - 1$ disjoint open intervals, with a constant value on each interval.*

Since the above characterization of F holds except on a set of measure zero, it will suffice for our algorithm to learn F with this representation. We will be somewhat lax in our language, and say that a set of intervals partitions some larger interval I , even if there are a finite number of points of I that are not included in any interval of the partition.

Definition 1 *The representation of a mixture of k uniform intervals as a set of at most $2k - 1$ pairs, $(x_i, x_{i+1}), h_i$, for $i = 0, \dots, m - 1 \leq 2k$ such that the density of the mixture is a constant h_i on the interval (x_i, x_{i+1}) will be called the step function representation of the mixture. We call each interval (x_i, x_{i+1}) a step of the mixture.*

Implicit in this definition is the fact that the m steps form a partition of the interval (x_0, x_m) .

Definition 2 *The support of a mixture of uniform intervals given by step function representation $(x_i, x_{i+1}), h_i$ consisting of m steps is the interval (x_0, x_m) . If F is the density function of the mixture, we denote its support by $\text{Support}(F)$.*

Definition 3 *A jump of a step function is a point x where the density changes (i.e. an endpoint shared by two different steps of the mixture).*

See Figure 1. We will work exclusively with the Step function representation of the mixture.

3 A Single Pass Algorithm

In this section, we present a single pass algorithm that with probability $1 - \delta$ will learn the density function of a mixture of k uniform intervals within L^1 distance ϵ using RAM space $O^*(k^2/\epsilon^2)$

OnePass is a divide and conquer algorithm, reminiscent of merge sort. Given a uniform random sample of $m = O^*(k^2/\epsilon^2)$ points drawn from the datastream, **OnePass** first initializes the computation, then sorts the m points. It calls subroutine **Cluster**, which divides the sample into two halves and recursively approximates the density function F in each half. **Cluster** then applies a *merging* procedure that decides whether or not the last interval of one half can be merged with the first interval of the other half.

The exact statement of the algorithm can be found in the appendix. The analysis of **OnePass** views the algorithm from a bottom-up perspective. The basic idea of **OnePass** is that the base case consists of a partition of $\text{Support}(F)$ into small intervals containing only two sample points. Adjacent intervals are then repeatedly merged together if the empirical densities of the two intervals are sufficiently close. The density of a merged interval is then close to the densities of the original intervals, with an error on the order of $O(\epsilon/(k \log m))$. After all mergers, the resulting intervals define the steps of the output function, G . Care must be taken in choosing which adjacent intervals to consider for merging; if we consider merging indiscriminately, then we can potentially induce a large aggregate error if an interval participates in too many mergers. The recursion defined by **Cluster** gives a structure for choosing the mergers, such that no interval is involved in more than $\log m$ mergers. This fact is crucial for establishing the correctness of the following theorem, whose proof has been omitted from this preliminary version. Many of the omitted proofs in this paper make use of a powerful statistical tool called the *Vapnik-Cervonenkis (VC) Dimension*. See [17,18] for these results, and [3] for the classic application of these tools to PAC learning.

Theorem 2 *With probability at least $1 - \delta$, one pass algorithm **OnePass** will learn a mixture of k uniform intervals to within L^1 distance ϵ using RAM memory at most*

$$O\left(\frac{k^2 \log^2(k/\epsilon) + \log^2 \log(1/\delta)}{\epsilon^2} \log\left(\frac{k}{\epsilon} + \log \frac{1}{\delta}\right)\right) = O^*\left(\frac{k^2}{\epsilon^2}\right).$$

4 Multiple Passes

In the previous section, we showed that a single pass algorithm can approximate a mixture of k uniform intervals to within L^1 distance ϵ using $O^*(k^2/\epsilon^2)$ RAM. In this section, we show that using additional passes can significantly improve the quality of the approximation, while holding the amount of RAM needed roughly constant.

Given a mixture of k uniform distributions with density function F and an integer l , algorithm **SmallRam** can approximate F to within L^1 distance ϵ^l with $2l$ passes, using at most $O^*(k^3/\epsilon^2)$ RAM.

In order to achieve the ϵ^l approximation, **SmallRam** requires that $n = \Omega^*((5k)^{8k+4}/(\epsilon^l)^{8k-2})$. This large sample complexity may seem like a drawback. However, the main application of **SmallRam** is to massive data sets; in such instances, the input in the datastream is generally considered to be very large relative to available computational resources.

SmallRam makes use of a subroutine called **Constant?**. **Constant?** is a single pass algorithm that will determine whether or not a datastream contains points drawn from a distribution with a constant density function. In Section 4.2, we describe the algorithm and sketch the proof of the following theorem about its performance:

Theorem 3 Given a datastream X consisting of sufficiently many samples drawn according to a mixture of k uniform distributions with density function H , with probability at least $1 - \delta\epsilon/(10k^2l)$, single pass algorithm **Constant?** will accept if H is a constant function and reject if H is more than $\epsilon^l/2k$ in L^1 distance from a constant function. It uses $O(k)$ RAM space.

4.1 The Algorithm

Subroutine **Estimate** draws a random sample S of size $|S| = O^*(k^2/\epsilon^2)$ from the input stream. It sorts S and partitions it into $O(k/\epsilon)$ intervals. **Estimate** then calls **Constant?** on each of the intervals; **Constant?** uses the entire datastream X to compute with great accuracy certain moments in the interval to determine if the distribution is constant on the interval. If F is close to constant on an interval, we can output the interval as a step of the final output G with a very accurate constant density estimate, again using the large set of samples, X , read via the datastream. However, if F is far away from constant on the interval, then estimating its density by a constant is too coarse; thus we repeat the process by recursively calling **Estimate**. This recursive call can be thought of as “zooming in on the jumps.” See Figure 2.

Given a sequence of points X and an interval J , we define $X|_J$ to be the subsequence of X that consists of points that fall in J . $X|_J$ can be considered a datastream; a pass over X can simulate a pass over $X|_J$.

Algorithm **SmallRam** input: datastream X of samples from distribution with density F .

$$n = |X| = \Omega\left(\frac{(5k)^{8k+4}}{\epsilon^{l(8k-2)}} \cdot \log\left(\frac{kl}{\delta\epsilon}\right)\right) \quad (1)$$

output: step function G .

1. initialize $p \leftarrow 1$, $J \leftarrow \text{Support}(F)$ (Note that J can be found in the first pass of the algorithm).
2. Call **Estimate** on p , J .

Subroutine **Estimate** : input: interval J , and level p .

1. Make a fresh pass, extracting a set S of size $m = \Theta\left(\frac{8k^2}{\epsilon^2} \log\left(\frac{lk}{\epsilon\delta}\right)\right)$ from $X|_J$, and store it into RAM.
2. Sort the samples in S in ascending order. If $p = 1$, let $q \leftarrow \frac{9\epsilon}{20k}m$; otherwise let $q \leftarrow (9/10) \cdot \epsilon m$.
3. Partition J into $m/q = 20k/(9\epsilon)$ or $10/(9\epsilon)$ disjoint intervals J_i^p , such that $|S \cap J_i^p| = q$.
4. Make an additional pass to count the exact number of points of X , the entire datastream, that fall in each of the J_i^p 's.
5. Also in this pass, call subroutine **Constant?** on $X|_{J_i^p}$, for each of the m/q intervals J_i^p in parallel. Mark those intervals J_i^p that **Constant?** rejects. Let M^p be the set of marked intervals.
6. If interval J_i^p is not marked in the previous step, output it as a step of G with density $|X \cap J_i^p|/(m \cdot \text{length}(J_i^p))$.
7. If $p < l$, for each $J_i^p \in M^p$, run **Estimate** on $J \leftarrow J_i^p$, $p \leftarrow p + 1$, in parallel with all other level $p + 1$ copies of **Estimate**.
If $p = l$, output each interval $J_i^p \in M^p$ as a step of G with density 0.

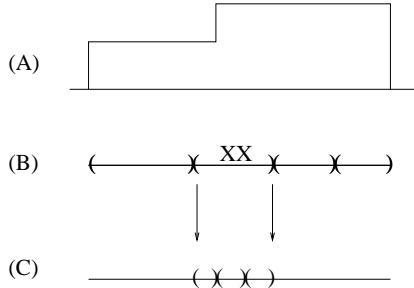


Figure 2: Zooming. (A) A step distribution. (B) Intervals have been created in Step 3 of some call to **Estimate**. Note that one of them contains a jump and has been marked. (C) **Estimate** is recursively called on the marked interval, and the process repeats.

Remark 1 *The order in which this recursive algorithm computes each call to **Estimate** has a natural structure defined by the level of the call (defined by its parameter p): All level p copies of **Estimate** are run in parallel, then all level $p+1$ copies are run in parallel, and so forth. All copies of **Estimate** running in parallel may read the datastream simultaneously, thus limiting the total number of passes per level to two. The total number of passes is therefore $2l$.*

The following two Lemmas are crucial to establishing the main result, Theorem 6. Proofs have been omitted from this preliminary version, and make use of VC-bounds and Chernoff bounds.

Lemma 4 *Let J_i^p be an interval created in Step 3 of a level p call to **Estimate**. With probability at least $1 - \delta\epsilon/(25k^2l)$, J_i^p contains at most $\epsilon^p/2k$ proportion of the total weight of F , and at least $(8/10)^p\epsilon^p/2k$ proportion of the total weight. That is,*

$$\left(\frac{8}{10}\right)^p \cdot \frac{\epsilon^p}{2k} \leq \int_{J_i^p} F \leq \frac{\epsilon^p}{2k}.$$

Lemma 5 *With probability $1 - \delta/2$, the following is true for all levels p : the aggregate number of intervals marked in level p calls to **Estimate** is at most $2k - 1$.*

The idea behind this Lemma is that an interval may only be marked if it contains a jump in the distribution, and there are at most $2k - 1$ jumps.

Theorem 6 *With probability at least $1 - \delta$, **SmallRam** will compute an approximation to F within L^1 distance ϵ^l of F , using at most $2l$ passes and $O(k^3/\epsilon^2(\log(lk/\epsilon\delta)))$ RAM.*

The intuition between Theorem 6 was sketched at the beginning of the section, but its technical proof involves many applications of various sampling principles.

4.2 Testing intervals for uniformity

We now describe the single pass subroutine **Constant?** used by **SmallRam**. Given a datastream of n samples from a mixture of k uniform distributions, with H as its density function, **Constant?** will accept H if it is constant, and reject if it is not within $\epsilon^l/2k$ in L^1 distance of constant.

By suitably scaling and translating the input, we may assume without loss of generality that the support of H is exactly the interval $[0, 1]$. Thus, our subroutine will determine whether $\int_0^1 |H - 1| \leq \epsilon^l/2k$ or not.

The high-level description of **Constant?** is that it computes estimates for the first $4k - 3$ moments of a slightly modified version of the probability distribution H . If these moments differ significantly from the moments that we would encounter from a uniform distribution, then we will know H is not uniform and will reject.

4.2.1 The η -smoothed distribution of H .

For technical reasons, our algorithm does not estimate the moments of H , but rather it estimates the raw moments of a probability distribution derived from H , which we call the η -smoothed distribution of H .

Definition 4 Given a mixture of k uniform distributions that defines a probability density H and a number $0 < \eta \leq 1$, define the η -smoothed distribution to be the following distribution, with density H_η . Let $i \leq 1/\eta - 1$ be the integer such that $x \in (i\eta, (i+1)\eta]$. Then

$$H_\eta(x) = \frac{1}{\eta} \int_{i\eta}^{(i+1)\eta} H(y) dy.$$

H_η satisfies the following useful properties.

Lemma 7 Given a random sample $x \in [0, 1]$ drawn according to H , we can derive a sample $x' \in [0, 1]$, drawn according to H_η in constant time.

Lemma 8 If H is a mixture of k uniform distributions, H_η is a mixture of uniform distributions, whose step function representation has at most $4k - 3$ steps, each of which has length at least η .

Lemma 9 If H is uniform, then H_η is also uniform.

Theorem 10 If $\eta \leq \epsilon^l/5k$ and $|H_\eta(x) - 1| \leq \epsilon^l/2k$ for all $x \in [0, 1]$, then H is within L^1 distance $\epsilon^l/2k$ of the uniform distribution.

The above theorem establishes the connection between H_η and H . Our algorithm will accept only if $|H_\eta - 1| \leq \epsilon^l/2k$, and thus will only accept if H is within $\epsilon^l/2k$ of uniform. The key point about the η -smoothed distribution is that its constituent steps/intervals all have length at least η . The proof of Theorem 3 in Section 4.2.3 will exploit this fact, specifically in its application of Lemma 11.

4.2.2 Description of Constant?

We are now ready to fully describe our subroutine for determining whether or not a given mixture is near uniform. The algorithm will accept if H is uniform, and reject if H is not within L^1 distance $\epsilon^l/2k$ from uniform. The following definition is standard:

Definition 5 If j is an integer, the j 'th raw moment of a distribution with density function μ is defined as $\int_0^1 x^j \mu(x) dx$.

The general idea of the algorithm is as follows. The j th moment of the uniform distribution is $1/(j+1)$. If H_η is uniform, its estimate should be close to $1/(j+1)$. Thus if the estimated moments deviate substantially, then we know that H_η is not close to uniform and reject. We are now able to present the algorithm:

Constant? input: datastream X with samples drawn according to mixture with density H , such that

$$|X| = \Omega \left(\frac{(5k)^{8k+3}}{\epsilon^{l(8k-4)}} \cdot \log \left(\frac{kl}{\delta\epsilon} \right) \right).$$

Let $X = \{x_1, \dots, x_m\}$.

Let ϕ be the function that, given a sample from H , returns a sample from H_η (see Lemma 7).

1. Set $\eta = \epsilon^l/(5k)$.
2. In a single pass, estimate the first $4k - 3$ raw moments of H_η , where the j 'th raw moment is estimated by $\hat{m}_j = \sum \phi(x_i)^j / |X|$.
3. If there exists a j such that

$$|(j+1)\hat{m}_j - 1| \geq \frac{\epsilon^{l(4k-2)}}{8(5k)^{4k+1/2}}$$

reject. Otherwise, **accept.**

4.2.3 Sketch of Proof of Theorem 3

Let M_j be the j 'th raw moment of H_η (distinct from \hat{m}_j , our *estimate* of M_j). By applying Hoeffding's bound, it can be shown that if \hat{m}_j is sufficiently close to $1/(j+1)$ then the difference between M_j and $1/(j+1)$ is small (call this difference $\alpha_j = (j+1)M_j - 1$), and if some \hat{m}_j is far from $1/(j+1)$, then H_η , and hence H , is not uniform. More precisely, it can be shown that if $|(j+1)\hat{m}_j - 1| \leq \epsilon^{l(4k-2)}/(8(5k)^{4k+1/2})$, then $|\alpha_j| \leq \epsilon^{(4k-2)l}/(2(5k)^{4k+1/2})$. If this is the case for all j , **Constant?** will accept. If some \hat{m}_j does not satisfy the inequality, then H_η is not uniform and **Constant?** will reject.

Thus, the proof of correctness of the algorithm reduces to proving that if $|\alpha_j| \leq \epsilon^{(4k-2)l}/(2(5k)^{4k+1/2})$ for all j , then H is sufficiently close to uniform. The general approach of our argument is as follows. Suppose H_η consists of $t \leq 4k - 3$ steps, defined by the points $0 = x_0 \leq x_1 \leq \dots \leq x_t = 1$, and heights h_i . That is, $(x_i, x_{i+1}), h_i$ is a step. The j 'th raw moment of H_η is then given by $M_j = 1/(j+1) \sum_{i=0}^{t-1} h_i(x_{i+1}^{j+1} - x_i^{j+1})$.

Consider the following problem: We are given the x_i s and the first t moments of H_η , and want to find the densities of the steps of H_η . This problem can be formulated as the following system of t linear equations in t variables, the h_i s:

$$\sum_{i=0}^{t-1} h_i(x_{i+1}^{j+1} - x_i^{j+1}) = (j+1)M_j, \quad j = 0, \dots, t-1. \quad (2)$$

Define $A = \{a_{ij}\}_{i,j=0}^{t-1}$ to be the $t \times t$ matrix (note that A 's indices run from 0 to $t-1$) corresponding to the linear system (2). More explicitly, $a_{ij} = x_{i+1}^{j+1} - x_i^{j+1}$. Linear system (2) can then be represented as $A\vec{h} = \vec{1} + \vec{\alpha}$, from which it follows that $|h_i - 1| \leq \|A^{-1}\|_2 \cdot |\vec{\alpha}|$, since $A^{-1}\vec{1} = \vec{1}$.

Of critical importance to our proof is the inequality $\|A^{-1}\|_2 \leq t^{3/2}/4\eta^t$ (which we prove in Lemma 11). Thus,

$$|h_i - 1| \leq \frac{t^{3/2}}{4\eta^t} |\vec{\alpha}| \leq \frac{(4k-3)^{3/2}}{4(\epsilon^l/5k)^{4k-3}} \cdot |\vec{\alpha}|.$$

Since $|\vec{\alpha}| \leq \epsilon^{(4k-2)l}/(2(5k)^{4k-1/2})$, then $|h_i - 1| < \epsilon^l/2k$ for all i . Recalling that the h_i s are the densities of the steps of H_η , Theorem 10 then implies that H is within L^1 distance $\epsilon^l/2k$ of uniform.

The last piece of the proof is the following lemma. Herein lies the importance of the η -smoothed distribution, because such distributions have the property that $x_1 \geq \eta$ and $|x_i - x_j| \geq \eta, \forall i, j$.

Lemma 11 *If $x_1 \geq \eta$ and $|x_i - x_j| \geq \eta$ for all i and j , then*

$$\|A^{-1}\|_2 \leq \frac{t^{3/2}}{4\eta^t}.$$

PROOF SKETCH: Let $V = \{v_{ij}\}_{i,j=0}^{t-1}$ be the Vandermonde matrix with $v_{ij} = x_{i+1}^j$. It can be shown with matrix algebra and manipulations of matrix norms that $\|A^{-1}\|_2 \leq t^{3/2}/x_1 \cdot \|V^{-1}\|_\infty$. Gautschi [8] proved upper bounds on $\|V^{-1}\|_\infty$, which for our purposes, yield:

$$\|A^{-1}\|_2 \leq \frac{t^{3/2}}{x_1} \cdot \max_j \prod_{i \neq j} \frac{1 + |x_i|}{|x_i - x_j|} \leq \frac{t^{3/2}}{4\eta^t}$$

The second inequality follows partially from the hypothesis of the Lemma that $|x_i - x_j| \geq \eta, \forall i, j$. \square

Note that if we had just used the moments of H rather than H_η , x_i and x_j could be arbitrarily close to each other, in which case A^{-1} would be prohibitively ill-conditioned.

5 Better Algorithm for Four Passes

As we saw in the previous section, **SmallRam** requires a large number of samples in its datastream. For the special case where we only wish to make four passes, we can significantly improve the sample complexity of the algorithm. The sample complexity is reduced by introducing subroutine **ConstantV2**, which is an analogue of **Constant?**, whose analysis avoids inverting a Vandermonde matrix. Thus, **ConstantV2** needs relatively few samples, but is only sufficient for a four pass algorithm. It can be found in the appendix. The proof of the following theorem follows closely that of Theorem 6.

Theorem 12 *Given a datastream X of size $|X| = \Omega(k^6/\epsilon^7 \cdot \log(k/\epsilon\delta))$ that contains samples drawn from a mixture of k uniform distributions with density function F , there exists a four pass algorithm that, with probability at least $1 - \delta$, will approximate F within L^1 distance ϵ^2 using at most $O(k^2/\epsilon^2 \cdot \log(k/\epsilon\delta))$ RAM.*

6 Conclusions

We have presented pass-efficient algorithms for the massive data set problem of approximately learning a mixture of k uniform distributions. The most striking feature of our work is the fact that our multiple pass algorithm has flexible performance guarantees: the number of passes allotted to the algorithm is an input parameter, such that each increase in this parameter will substantially decrease the error of the computation while holding memory requirements constant. Thus, we have demonstrated that allowing a streaming algorithm to make more than one pass over the data can substantially enhance its performance.

Possible extensions of this work include designing multiple pass algorithms for learning mixtures of multidimensional uniform distributions, or for learning mixtures of Gaussian distributions in low or high dimension.

Lastly, much future work lies in designing multiple pass algorithms for massive data set problems, with flexible performance guarantees trading number of passes for RAM used. Potentially, such algorithms may use far less RAM than known single pass algorithms and may even break lower bounds on RAM used for single pass algorithms.

References

- [1] A. Aboulnaga and S. Chaudhuri. Self-tuning histograms: Building histograms without looking at data. In *Proceedings ACM SIGMOD International Conference on Management of Data*, pages 181–192, 1999.
- [2] S. Arora and R. Kannan. Learning mixtures of arbitrary gaussians. In *Proceedings of the 33rd Annual ACM Symposium on the Theory of Computing*, pages 247–257, 2001.
- [3] A. Blumer, A. Ehrenfeucht, D. Haussler, and M. Warmuth. Learnability and the vapnik-chervonenkis dimension. *Journal of the Association for Computing Machinery*, 36(1):929–965, 1989.
- [4] M. Charikar, C. Chekuri, T. Feder, and R. Motwani. Incremental clustering and dynamic information retrieval. In *Proceedings of the 29th Annual ACM Symposium on the Theory of Computing*, pages 626–635, 1997.
- [5] M. Charikar, L. O’Callaghan, and R. Panigrahy. Better streaming algorithms for clustering problems. In *Proceedings of the 35th Annual ACM Symposium on the Theory of Computing*, pages 30–39, 2003.
- [6] S. DasGupta. Learning mixtures of gaussians. In *Proceedings of the 40th IEEE Symposium on Foundations of Computer Science*, pages 634–644, 1999.
- [7] P. Drineas and R. Kannan. Pass efficient algorithm for large matrices. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 223–232, 2003.
- [8] W. Gautschi. Norm estimates for inverses of vandermonde matrices. *Numerische Mathematik*, 23:337–47, 1975.
- [9] P. B. Gibbons, Y. Matias, and V. Poosala. Fast incremental maintenance of approximate histograms. In *Proceedings of 23th International Conference on Very Large Data Bases*, pages 266–275, 1997.
- [10] 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.
- [11] A. C. Gilbert, Y. Kotidis, S. Muthukrishnan, and M. Strauss. Surfing wavelets on streams: One-pass summaries for approximate aggregate queries. In *Proceedings of 27th International Conference on Very Large Data Bases*, pages 79–88, 2001.
- [12] S. Guha and N. Koudas. Approximating a data stream for querying and estimation: Algorithms and performance evaluation. In *Proceedings of the 18th International Conference on Data Engineering*, pages 567–, 2002.
- [13] S. Guha, N. Koudas, and K. Shim. Data-streams and histograms. In *Proceedings of the 33rd Annual ACM Symposium on the Theory of Computing*, pages 471–475, 2001.
- [14] S. Guha, N. Mishra, R. Motwani, and L. O’Callaghan. Clustering data streams. In *Proceedings of the 32nd Annual ACM Symposium on the Theory of Computing*, pages 359–366, 2000.
- [15] H. Jagadish, N. Koudas, S. Muthukrishnan, V. Poosala, K. C. Sevcik, and T. Suel. Optimal histograms with quality guarantees. In *Proceedings of 24th International Conference on Very Large Data Bases*, pages 275–286, 1998.

- [16] Y. Matias, J. S. Vitter, and M. Wang. Dynamic maintenance of wavelet-based histograms. In *Proceedings of 26th International Conference on Very Large Data Bases*, pages 101–110, 2000.
- [17] M. Talagrand. Sharper bounds for gaussian and empirical processes. *Annals of Probability*, 22(1):20–76, 1994.
- [18] V. Vapnik and A. Cervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probability and its Applications*, 16:264–280, 1971.
- [19] S. Vempala and G. Wang. A spectral algorithm for learning mixtures of distributions. In *Proceedings of the 43rd IEEE Symposium on Foundations of Computer Science*, pages 113–122, 2002.

Appendix

Description of Algorithm OnePass :

Before describing the algorithm, we make the following definition.

Definition 6 Suppose we have a set of points S and two intervals J and J' . We say that the empirical density of J is consistent with J' on S , or simply J is consistent with J' on S , if

$$\left| \frac{|J \cap S|}{|S| \text{length}(J)} - \frac{|J' \cap S|}{|S| \text{length}(J')} \right| \leq \frac{\epsilon}{97k \log m} \cdot \frac{1}{\text{length}(J)}.$$

OnePass input: datastream X of samples from mixture of k uniform distributions with density function F , such that $|X| \geq m$ (defined below).

output: approximate density function G .

1. In one pass, sample

$$m = O \left(\frac{k^2 \log^2(k/\epsilon) + \log^2 \log(1/\delta)}{\epsilon^2} \left(\log \frac{k}{\epsilon} + \log \frac{1}{\delta} \right) \right) \quad (3)$$

points from X , where m is a power of 2. Sort the samples, and call the sorted samples S .

2. Call **Cluster** on S , and let the resulting intervals be $\tilde{J}_1, \dots, \tilde{J}_v$.
3. Output the approximate density function, G , as the step function $\tilde{J}_i, |S \cap \tilde{J}_i| / (m \text{length}(\tilde{J}_i))$.

Cluster input: set of t sorted points, $T = \{s_1, \dots, s_t\}$.

Base case: If $t < 3$, then output interval (s_1, s_{t+1}) , where s_{t+1} is the point proceeding s_t in S .

1. Set $T_1 = \{s_1, \dots, s_{k/2}\}$ and $T_2 = \{s_{k/2+1}, \dots, s_k\}$
2. Call **Cluster** on T_1 , and let the resulting intervals be I_1, \dots, I_{k_1} .
3. Call **Cluster** on T_2 and let the resulting intervals be J_1, \dots, J_{k_2} .
4. Set $K = I_{k_1} \cup J_1$
5. If the empirical densities of both I_{k_1} and J_1 are consistent with K on T , then output the set of intervals $\{I_1, \dots, I_{k_1-1}, K, J_2, \dots, J_{k_2}\}$.
Otherwise, output the set of intervals $\{I_1, \dots, I_{k_1}, J_1, \dots, J_{k_2}\}$

Description of Algorithm ConstantV2 :

ConstantV2 : input: interval J , datastream X of samples from F such that

$$|X| = \Omega\left(\frac{k^6}{\epsilon^7} \cdot \log\left(\frac{k}{\epsilon\delta}\right)\right).$$

1. Partition J into $16k/\epsilon$ intervals of *equal length*; call these intervals the J_i 's.
2. In a single pass, count the number of points of X that fall in each of the J_i 's.
3. If all the J_i 's satisfy:

$$\left| \frac{|X \cap J_i|}{\text{nlength}(J_i)} - \frac{|X \cap J|}{\text{nlength}(J)} \right| \leq \frac{\epsilon^2}{8k^2} \frac{|X \cap J|}{\text{nlength}(J)} \quad (4)$$

then **accept**. Otherwise, **reject**.