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

Kevin L. Chang

Max-Planck Insititute for Computer Science

October 2, 2007
Theoretical Abstractions for Massive Data Set Computation

- The input on disk/storage is modeled as a read-only array. Elements may only be accessed through a sequential pass.
  - Input elements can be arbitrarily ordered.
- Main memory is modeled as extra space used for intermediate calculations.
- Algorithm is allowed some extra computing time before and after each pass.
- **Goal**: Design algorithms that minimize memory usage, number of passes, extra computing time.
Models of Computation

- **Streaming Model**: Algorithm may make a single pass over the data. Space must be $o(n)$.
- **Pass-Efficient Model**: Algorithm may make a small, constant number of passes. Ideally, space is $O(1)$.
- Other models: external memory, sublinear algorithms

Note that Pass-Efficient is more flexible than streaming, but not suitable for “streaming” data arriving that is processed immediately and then “forgotten”.
Why do we need multiple passes?

- Is it better, in terms of other resources, to make 3 passes instead of 1 pass?
- Example: Find the mean of an array of \( n \) integers.
  - 1 pass requires \( O(1) \) space.
  - More passes don’t help.
- Example: Find the median. [MP ’80]
  - 1 pass algorithm requires \( \Omega(n) \) space.
  - 2 pass needs only \( O(n^{1/2}) \) space.
- We study the trade-off between passes and memory.
Generative clustering model: Mixtures of $k$ distributions in $\mathbb{R}^d$

- $k$ probability distributions $F_1, \ldots, F_k$, each with weight $w_i > 0$ such that $\sum w_i = 1$.
- To generate a sample $x$ from the mixture: Pick a distribution $F_i$ with probability $w_i$. Draw a point according to $F_i$.
- If each $F_i : \Omega \to \mathbb{R}$ is a density function, then density of mixture is $F(x) = \sum_i w_i F_i(x)$. 

Place the samples in a data stream, and order arbitrarily. 

Goal of algorithm: from data stream, approximate $F$ with a function $G$. Error measured by $L_1$ norm: 

$$\int_{\Omega} |F - G|$$
Generative clustering model: Mixtures of $k$ distributions in $\mathbb{R}^d$

- $k$ probability distributions $F_1, \ldots, F_k$, each with weight $w_i > 0$ such that $\sum w_i = 1$.
- To generate a sample $x$ from the mixture: Pick a distribution $F_i$ with probability $w_i$. Draw a point according to $F_i$.
- If each $F_i : \Omega \to \mathbb{R}$ is a density function, then density of mixture is $F(x) = \sum_i w_i F_i(x)$.
- Place the samples in a data stream, and order arbitrarily.
- Goal of algorithm: from data stream, approximate $F$ with a function $G$. Error measured by $L^1$ norm: $\int_\Omega |F - G|$. 
Mixtures of $k$ uniform distributions in $\mathbb{R}^d$

- Each mixture component $F_i$ is a uniform distribution over some cell $R_i$ in $[0, 1]^d$: $R_i = (a_1, b_1) \times \ldots \times (a_d, b_d) = \{ x \in \mathbb{R}^d | a_i < x_i < b_i, i = 1, \ldots, d \}$. 

![Diagram of mixtures of uniform distributions](image)
Main Result: mixture of $k$ uniform distributions in $\mathbb{R}^d$

- For any $\ell > 0$, a $2\ell + 1$ pass algorithm that, with probability at least $1 - \delta$, learns $F$ to within error $\epsilon$ such that
  - Memory required:
    $$\tilde{O}\left(\frac{k^3d^3}{\epsilon^2/\ell} \log(1/\delta) + (2k)^d\right)$$
  - Sample complexity: Requires $\tilde{\Omega}\left((wkd/\epsilon)^{O(d)}\right)$ samples in data stream.
- This exhibits a trade-off between the number of passes and memory.
  - Error stays the same, but the memory decreases significantly with each extra pair of passes!
  - If 3 passes, requires $\approx 1/\epsilon^2$ space, 5 passes: $\approx 1/\sqrt{\epsilon}$, 9 passes $\approx 1/\sqrt[3]{\epsilon}$.
  - Suffers from curse of dimensionality.
Main Result (second interpretation)

For any $\ell > 0$, a $2\ell + 1$ pass algorithm that, with probability $1 - \delta$, learns $F$ to within error $\epsilon^\ell$ such that

- Memory required:
  \[
  \tilde{O} \left( \frac{k^3 d^3}{\epsilon^2} \log(1/\delta) + \frac{\ell k d^2}{\epsilon} \log(1/\delta) + (2k)^d \right)
  \]

- A few extra passes doesn’t affect the memory requirement much, but the error drops exponentially!

- Rest of talk will focus on this error guarantee.
Related Work

- Same problem, \( d = 1, 2 \) dimension [Chang & Kannan '06], [Guha & McGregor '07].
- Learning mixtures of Gaussian distributions in \( \mathbb{R}^d \). Not streaming. [Dasgupta '99, Arora & Kannan '05, Vempala & Wang '04, KSV '06]
- Multidimensional histograms in data streams. [TGIK '02]
  - Streaming alg, similar to our problem, but instead of assuming generative model of data, just find the best fit function.
  - also suffers from curse of dimensionality.
Preliminaries: Structure of cells in $\mathbb{R}^d$

- Cell $R_i = \{ x \in \mathbb{R}^d | a_i \leq x_i \leq b_i \}$ has $2d$ boundary faces (for $d = 2$, edges of the rectangle).

- For $m \in [d]$, the two $m$-boundary faces are: \\
  $\{ x \in R_i | x_m = a_m \}$ and $\{ x \in R_i | x_m = b_m \}$.

- Each boundary face can be thought of as a $d - 1$ dimensional cell that is completely contained in a $d - 1$ dimensional hyperplane with fixed $m$th coordinate.

$(x, y)$, → first coordinate, ↑ second coordinate.
Subroutine 1: Find those hyperplanes!

Input: $m$, data stream.
Find the set of hyperplanes with fixed $m$th component that contain all $m$-boundary faces.

- Uses $2\ell$ passes, space $\tilde{O}(k^3 d^2 / \epsilon^2 + k d \ell / \epsilon)$.
- Get $2k$ hyperplanes that contain all $m$-boundary faces.
- Warning: real subroutine does this with some error $\approx \epsilon^{\ell} / d$.

More details on how to accomplish this later.
Using SR1 to learn the mixture

- Run SR1 for all $m = 1, \ldots, d$, in parallel. $2\ell$ passes.
Using SR1 to learn the mixture

- Run SR1 for all \( m = 1, \ldots, d \), in parallel. \( 2\ell \) passes.
- The output hyperplanes partition bounding box into a set of cells \( \mathcal{C} \) such that \( F \) is constant when restricted to each cell. Number of such cells: \( |\mathcal{C}| \leq (2k)^d \).
Using SR1 to learn the mixture

- Run SR1 for all $m = 1, \ldots, d$, in parallel. $2\ell$ passes.
- The output hyperplanes partition bounding box into a set of cells $C$ such that $F$ is constant when restricted to each cell. Number of such cells: $|C| \leq (2k)^d$. 
In one more pass: count the number of points of data stream $X$ that falls in each $C \in C$.

- Since $F$ is constant on each $C$, $|X \cap C|/(|X| \text{Vol}(C))$ will approximate $F(x)$, $x \in C$.
- With $|X|$ sufficiently large, approximation of $F$ will have $L^1$ error at most $\epsilon^\ell$.

Memory used: $\tilde{O}(k^3 d^3/\epsilon^2 + kd^2 \ell/\epsilon + (2k)^d)$.

Passes used: $2\ell + 1$. 

Using SR1 to learn the mixture, continued
Finding boundary hyperplanes: Subroutine 1

Iterative algorithm, in $\ell$ pairs of passes.

1. First pass: Take a sample of size $s = O(d^2 k^2 / \epsilon^2)$ from the data stream.
   - Partition the bounding box $R$ into a set of cells $\{P_i\}$ such that each $m$-boundary face is completely contained in one $P_i$.
   - Using the sample, can do this such that $\int_{P_i} F = \Theta(\epsilon/dk)$, whp.
   - Number of cells will be $\Theta(dk/\epsilon)$. 
Iterative algorithm, in $\ell$ pairs of passes.

1. First pass: Take a sample of size $s = O(d^2k^2/\epsilon^2)$ from the data stream.
   - Partition the bounding box $R$ into a set of cells $\{P_i\}$ such that each $m$-boundary face is completely contained in one $P_i$.
   - Using the sample, can do this such that $\int_{P_i} F = \Theta(\epsilon/dk)$, whp.
   - Number of cells will be $\Theta(dk/\epsilon)$.

2. Second pass: Test each partition cell $P_i$ to see if $P_i$ contains an $m$-boundary face.
   - Test algorithm requires $\tilde{O}(d\ell)$ space to make.
Iterative algorithm, in $\ell$ pairs of passes.

1. First pass: Take a sample of size $s = O(d^2 k^2 / \epsilon^2)$ from the data stream.
   - Partition the bounding box $R$ into a set of cells $\{P_i\}$ such that each $m$-boundary face is completely contained in one $P_i$.
   - Using the sample, can do this such that $\int_{P_i} F = \Theta(\epsilon/dk)$, whp.
   - Number of cells will be $\Theta(dk/\epsilon)$.

2. Second pass: Test each partition cell $P_i$ to see if $P_i$ contains an $m$-boundary face.
   - Test algorithm requires $\tilde{O}(d\ell)$ space to make.

3. Then recurse on those partition cells that do contain $m$-boundary face in parallel.
Finding boundary hyperplanes: Subroutine 1

Iterative algorithm, in \( \ell \) pairs of passes.

1. First pass: Take a sample of size \( s = O(d^2 k^2 / \epsilon^2) \) from the data stream.
   - Partition the bounding box \( R \) into a set of cells \( \{P_i\} \) such that each \( m \)-boundary face is completely contained in one \( P_i \).
   - Using the sample, can do this such that \( \int_{P_i} F = \Theta(\epsilon/dk) \), whp.
   - Number of cells will be \( \Theta(dk/\epsilon) \).

2. Second pass: Test each partition cell \( P_i \) to see if \( P_i \) contains an \( m \)-boundary face.
   - Test algorithm requires \( \tilde{O}(d\ell) \) space to make.

3. Then recurse on those partition cells that do contain \( m \)-boundary face in parallel.

Warning: real algorithm more complicated.
An example for $d = 2, m = 1$. 

What happens when you iterate?

- Volume of each partition cell will decrease by at least a multiplicative factor of $\epsilon$.
- After $\ell$ iterations (2$\ell$ passes), each partition cell that contains a boundary hyperplane has volume at most $\approx \epsilon \ell / (d^k)$.
- Thus, after $\ell$ iterations, will have isolated each boundary face to within a partition cell of volume $\epsilon \ell / (d^k)$. 
An example for $d = 2, m = 1$. 

What happens when you iterate?

- Volume of each partition cell will decrease by at least a multiplicative factor of $\epsilon$.
- After $\ell$ iterations ($2^\ell$ passes), each partition cell that contains a boundary hyperplane has volume at most $\approx \epsilon \ell / (d k)$.
- Thus, after $\ell$ iterations, will have isolated each boundary face to within a partition cell of volume $\epsilon \ell / (d k)$. 
An example for $d = 2, m = 1$. 

What happens when you iterate?

- Volume of each partition cell will decrease by at least a multiplicative factor of $\epsilon$.
- After $\ell$ iterations ($2^\ell$ passes), each partition cell that contains a boundary hyperplane has volume at most $\approx \frac{\epsilon \ell}{d k}$.
- Thus, after $\ell$ iterations, will have isolated each boundary face to within a partition cell of volume $\epsilon \ell / (d k)$. 
An example for $d = 2, m = 1$. 

What happens when you iterate?

- Volume of each partition cell will decrease by at least a multiplicative factor of $\epsilon$.
- After $\ell$ iterations (2\$^\ell$ passes), each partition cell that contains a boundary hyperplane has volume at most $\approx \epsilon \ell / (dk)$.
- Thus, after $\ell$ iterations, will have isolated each boundary face to within a partition cell of volume $\epsilon \ell / (dk)$. 
An example for $d = 2, m = 1$.

What happens when you iterate?

- Volume of each partition cell will decrease by at least a multiplicative factor of $\epsilon$.
- After $\ell$ iterations ($2\ell$ passes), each partition cell that contains a boundary hyperplane has volume at most $\approx \epsilon^\ell/(dk)$.
- Thus, after $\ell$ iterations, will have isolated each boundary face to within a partition cell of volume $\epsilon^\ell/(dk)$. 
What’s the space complexity?

- Each iteration takes a sample of size $s$ and runs the test algorithm for $kd/\epsilon$ cells.
- $\tilde{O}(s + d\ell/\epsilon) = \tilde{O}(k^2d^2/\epsilon^2 + d\ell)$.
- Recursion only occurs on at most $2k$ cells in parallel. Total memory: $\tilde{O}(k^3d^2/\epsilon^2 + d\ell k)$. 
Fundamental one pass algorithm [Indyk 1999] for computing the $\ell_1$ length of a vector $v \in \mathbb{R}^n$ given as a stream of dynamic updates.

Input: a data stream that consists of pairs $\langle a, i \rangle$, $i \in [n]$ and $a \in (-M, M)$.

Suppose we have a vector $v \in \mathbb{R}^n$. Each pair is an update of $v$ with semantics: add $a$ to $i$th component of $v$.

Approximate in one pass $\|v\|_1$ or $\|v\|_2$ with $o(n)$ space.
Thanks for your time!