Stanford

Covariance Matrices & All-pairs Similarity

Reza Zadeh

STANFORD UNIVERSITY

Matroid

May 2018, Stanford DAO

Notation for matrix A

- Given $m \times n$ matrix A, with $m \gg n$.

$$A = \begin{pmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m,1} & a_{m,2} & \cdots & a_{m,n} \end{pmatrix}$$

- A is tall and skinny, example values $m = 10^{12}, n = \{10^4, 10^6\}$.
- A has sparse *rows*, each row has at most L nonzeros.
- A is stored across hundreds of machines and cannot be streamed through a single machine.

Computing $A^T A$

- We compute $A^T A$.
- $A^T A$ is $n \times n$, considerably smaller than A.
- $A^T A$ is dense.
- Holds dot products between all pairs of columns of A.

Guarantees

There is a knob γ which can be turned to preserve similarities and singular values. Paying $O(nL\gamma)$ communication cost and $O(\gamma)$ computation cost.

- With a low setting of γ, preserve similar entries of $A^T A$ (via Cosine, Dice, Overlap, and Jaccard similarity).
- With a high setting of γ, preserve singular values of $A^T A$.

Computing All Pairs of Cosine Similarities

- We have to find dot products between all pairs of columns of A
- We prove results for general matrices, but can do better for those entries with $\cos(i, j) \geq s$
- Cosine similarity: a widely used definition for “similarity" between two vectors

$$\cos(i, j) = \frac{c_i^T c_j}{||c_i|| ||c_j||}$$

- c_i is the $i'th$ column of A

Example matrix

Rows: users.
Columns: movies.

Distributed Computing Environment

- With such large datasets, we must use many machines.
- Algorithm code available in Spark and Scalding.
- Described with Maps and Reduces so that the framework takes care of distributing the computation.

Naive Implementation

1. Given row r_i, Map with NaiveMapper (Algorithm 1)
2. Reduce using the NaiveReducer (Algorithm 2)

Algorithm 1 NaiveMapper(r_i)

for all pairs (a_{ij}, a_{ik}) in r_i **do**
  Emit $((j, k) \to a_{ij}a_{ik})$
end for

Algorithm 2 NaiveReducer($(i, j), \langle v_1, \ldots, v_R \rangle$)

output $c_i^T c_j \to \sum_{i=1}^{R} v_i$

Analysis for First Pass

- Very easy analysis
- 1) Shuffle size: $O(mL^2)$
- 2) Largest reduce-key: $O(m)$
- Both depend on m, the larger dimension, and are intractable for $m = 10^{12}, L = 100$.
- We'll bring both down via clever sampling
- Assuming column norms are known or estimates available

Algorithm 3 DIMSUMMapper(r_i)

for all pairs (a_{ij}, a_{ik}) in r_i **do**
  With probability $\min\left(1, \gamma \frac{1}{||c_j||||c_k||}\right)$
  emit $((j,k) \to a_{ij}a_{ik})$
end for

Algorithm 4 DIMSUMReducer($(i,j), \langle v_1, \ldots, v_R \rangle$)

if $\frac{\gamma}{||c_i||||c_j||} > 1$ **then**
  output $b_{ij} \to \frac{1}{||c_i||||c_j||} \sum_{i=1}^{R} v_i$
else
  output $b_{ij} \to \frac{1}{\gamma} \sum_{i=1}^{R} v_i$
end if

Analysis for DIMSUM

The algorithm outputs b_{ij}, which is a matrix of cosine similarities, call it B.
Four things to prove:

1. Shuffle size: $O(nL\gamma)$
2. Largest reduce-key: $O(\gamma)$
3. The sampling scheme preserves similarities when $\gamma = \Omega(\log(n)/s)$
4. The sampling scheme preserves singular values when $\gamma = \Omega(n/\epsilon^2)$

Shuffle size for DIMSUM

Theorem

For $\{0, 1\}$ matrices, the expected shuffle size for DIMSUMMapper is $O(nL\gamma)$.

Proof.

The expected contribution from each pair of columns will constitute the shuffle size:

$$\sum_{i=1}^{n} \sum_{j=i+1}^{n} \sum_{k=1}^{\#(c_i,c_j)} \Pr[\text{DIMSUMEmit}(c_i, c_j)]$$

$$= \sum_{i=1}^{n} \sum_{j=i+1}^{n} \#(c_i, c_j) \Pr[\text{DIMSUMEmit}(c_i, c_j)]$$

Shuffle size for DIMSUM

Proof.

$$\leq \sum_{i=1}^{n} \sum_{j=i+1}^{n} \gamma \frac{\#(c_i, c_j)}{\sqrt{\#(c_i)}\sqrt{\#(c_j)}}$$

Shuffle size for DIMSUM

Proof.

$$\leq \sum_{i=1}^{n} \sum_{j=i+1}^{n} \gamma \frac{\#(c_i, c_j)}{\sqrt{\#(c_i)}\sqrt{\#(c_j)}}$$

$$\text{(by AM-GM)} \quad \leq \frac{\gamma}{2} \sum_{i=1}^{n} \sum_{j=i+1}^{n} \#(c_i, c_j)\left(\frac{1}{\#(c_i)} + \frac{1}{\#(c_j)}\right)$$

Proof.

$$\leq \sum_{i=1}^{n} \sum_{j=i+1}^{n} \gamma \frac{\#(c_i, c_j)}{\sqrt{\#(c_i)}\sqrt{\#(c_j)}}$$

$$\text{(by AM-GM)} \quad \leq \frac{\gamma}{2} \sum_{i=1}^{n} \sum_{j=i+1}^{n} \#(c_i, c_j)\left(\frac{1}{\#(c_i)} + \frac{1}{\#(c_j)}\right)$$

$$\leq \gamma \sum_{i=1}^{n} \frac{1}{\#(c_i)} \sum_{j=1}^{n} \#(c_i, c_j)$$

Shuffle size for DIMSUM

Proof.

$$\leq \sum_{i=1}^{n} \sum_{j=i+1}^{n} \gamma \frac{\#(c_i, c_j)}{\sqrt{\#(c_i)}\sqrt{\#(c_j)}}$$

$$\text{(by AM-GM)} \quad \leq \frac{\gamma}{2} \sum_{i=1}^{n} \sum_{j=i+1}^{n} \#(c_i, c_j)\left(\frac{1}{\#(c_i)} + \frac{1}{\#(c_j)}\right)$$

$$\leq \gamma \sum_{i=1}^{n} \frac{1}{\#(c_i)} \sum_{j=1}^{n} \#(c_i, c_j)$$

$$\leq \gamma \sum_{i=1}^{n} \frac{1}{\#(c_i)} L\#(c_i) = \gamma L n$$

$\square$

Shuffle size for DIMSUM

- $O(nL\gamma)$ has no dependence on the dimension m, this is the heart of DIMSUM.
- Happens because higher magnitude columns are sampled with lower probability:

$$\gamma \frac{1}{||c_1||||c_2||}$$

Shuffle size for DIMSUM

- For matrices with real entries, we can still get a bound
- Let H be the smallest nonzero entry in magnitude, after all entries of A have been scaled to be in $[-1, 1]$
- E.g. for $\{0, 1\}$ matrices, we have $H = 1$
- Shuffle size is bounded by $O(nL\gamma/H^2)$

Largest reduce key for DIMSUM

- Each reduce key receives at most γ values (the oversampling parameter)
- Immediately get that reduce-key complexity is $O(\gamma)$
- Also independent of dimension m. Happens because high magnitude columns are sampled with lower probability.

Correctness

- Since higher magnitude columns are sampled with lower probability, are we guaranteed to obtain correct results w.h.p.?
- Yes. By setting γ correctly.
- Preserve similarities when $\gamma = \Omega(\log(n)/s)$
- Preserve singular values when $\gamma = \Omega(n/\epsilon^2)$

Correctness

Theorem

Let A be an $m \times n$ tall and skinny ($m > n$) matrix. If $\gamma = \Omega(n/\epsilon^2)$ and D a diagonal matrix with entries $d_{ii} = ||c_i||$, then the matrix B output by DIMSUM satisfies,

$$\frac{||DBD - A^T A||_2}{||A^T A||_2} \leq \epsilon$$

with probability at least $1/2$.

Relative error guaranteed to be low with constant probability.

Proof

- Uses Latala's theorem, bounds 2nd and 4th central moments of entries of B.
- Really need extra power of moments.

Latala's Theorem

Theorem

(Latala's theorem). Let X be a random matrix whose entries x_{ij} are independent centered random variables with finite fourth moment. Denoting $||X||_2$ as the matrix spectral norm, we have

$$\mathbb{E}\,||X||_2 \leq C[\max_i \left(\sum_j \mathbb{E}\,x_{ij}^2\right)^{1/2} + \max_j \left(\sum_i \mathbb{E}\,x_{ij}^2\right)^{1/2} + \left(\sum_{i,j} \mathbb{E}\,x_{ij}^4\right)^{1/4}].$$

Proof

Prove two things

- $\mathbb{E}[(b_{ij} - Eb_{ij})^2] \leq \frac{1}{\gamma}$ (easy)
- $\mathbb{E}[(b_{ij} - Eb_{ij})^4] \leq \frac{2}{\gamma^2}$ (not easy)

Details in paper.

Correctness

Theorem

For any two columns c_i and c_j having $\cos(c_i, c_j) \geq s$, let B be the output of DIMSUM with entries $b_{ij} = \frac{1}{\gamma} \sum_{k=1}^{m} X_{ijk}$ with X_{ijk} as the indicator for the k'th coin in the call to DIMSUMMapper. Now if $\gamma = \Omega(\alpha / s)$, then we have,

$$\Pr\left[||c_i||||c_j||b_{ij} > (1+\delta)[A^T A]_{ij} \right] \leq \left(\frac{e^{\delta}}{(1+\delta)^{(1+\delta)}} \right)^{\alpha}$$

and

$$\Pr\left[||c_i||||c_j||b_{i,j} < (1-\delta)[A^T A]_{ij} \right] < \exp(-\alpha\delta^2/2)$$

Relative error guaranteed to be low with high probability.

Correctness

Proof.

- In the paper
- Uses standard concentration inequality for sums of indicator random variables.
- Ends up requiring that the oversampling parameter γ be set to $\gamma = \log(n^2)/s = 2\log(n)/s$.

$\square$

Observations

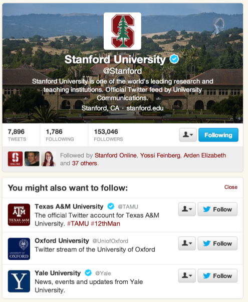
- DIMSUM helpful when there are some popular columns
- e.g. The Netflix Matrix (some columns way more popular than others)
- Power-law columns are effectively neutralized

In practice

- Forget about theoretical settings for γ
- Crank up γ until application works
- Estimates for $||c_i||$ can be used, expectations still hold, but concentration isn't guaranteed
- If using for singular values, watch for ill-conditioned matrices

Experiments

- Large scale production live at `twitter.com`



Experiments

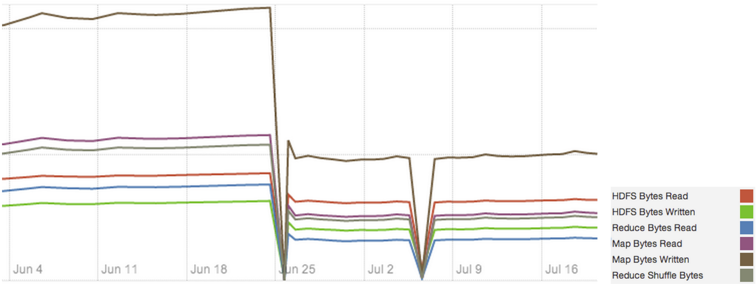


Figure: Y-axis ranges from 0 to 100s of terabytes

Implementation


```
// Load and parse the data file.
val rows = sc.textFile(filename).map { line =>
  val values = line.split(' ').map(_.toDouble)
  Vectors.dense(values)
}
val mat = new RowMatrix(rows)

// Compute similar columns perfectly, with brute force.
val simsPerfect = mat.columnSimilarities()

// Compute similar columns with estimation using DIMSUM
val simsEstimate = mat.columnSimilarities(threshold)
```

Figure: Widely distributed with Spark as of version 1.2

Other Similarity Measures

Picking out similar columns work for some other similarity measures.

Similarity	Definition	Shuffle Size	Reduce-key size
Cosine	$\frac{\#(x,y)}{\sqrt{\#(x)}\sqrt{\#(y)}}$	$O(nL\log(n)/s)$	$O(\log(n)/s)$
Jaccard	$\frac{\#(x,y)}{\#(x)+\#(y)-\#(x,y)}$	$O((n/s)\log(n/s))$	$O(\log(n/s)/s)$
Overlap	$\frac{\#(x,y)}{\min(\#(x),\#(y))}$	$O(nL\log(n)/s)$	$O(\log(n)/s)$
Dice	$\frac{2\#(x,y)}{\#(x)+\#(y)}$	$O(nL\log(n)/s)$	$O(\log(n)/s)$

Table: All sizes are independent of m, the dimension.

Locality Sensitive Hashing

- MinHash from the Locality-Sensitive-Hashing family can have its vanilla implementation greatly improved by DIMSUM.
- Another set of theorems for shuffle size and correctness in DISCO paper.
  `stanford.edu/~rezab/papers/disco.pdf`

Conclusion

- Consider DIMSUM if you ever need to compute $A^T A$ for large sparse A
- Many more experiments and results in paper at `stanford.edu/~rezab`

Combiners

- All bounds are without combining: can only get better with combining
- For similarities, DIMSUM (without combiners) beats naive with combining outright
- For singular values, DIMSUM (without combiners) beats naive with combining if the number of machines is large (e.g. 1000)
- DIMSUM with combining empirically beats naive with combining
- Difficult to analyze combiners since they happen at many levels. Optimizations break models.
- DIMSUM with combiners is best of both.

Combiners

With k machines

- DIMSUM shuffle with combiner: $O(\min(nL\gamma, kn^2))$
- DIMSUM reduce-key with combiner: $O(\min(\gamma, k))$
- Naive shuffle with combiner: $O(kn^2)$
- Naive reduce-key with combiner: $O(k)$

DIMSUM with combiners is best of both.

Experiments

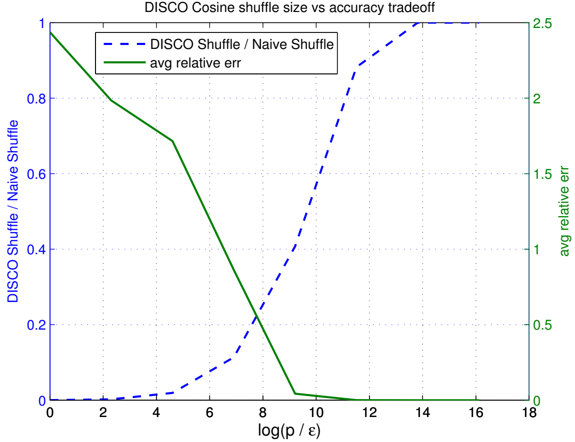


Figure: As $\gamma = p/\epsilon$ increases, shuffle size increases and error decreases. There is no thresholding for highly similar pairs here.