Dimension Independent Matrix Square using MapReduce

Reza Bosagh Zadeh

ICME
INSTITUTE for COMPUTATIONAL & MATHEMATICAL ENGINEERING at STANFORD UNIVERSITY

STOC 2013

Outline

Computing $A^T A$

- Given $m \times n$ matrix A with entries in $[0, 1]$ and $m \gg n$, compute $A^T A$.

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

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

Guarantees

- Preserve singular values of $A^T A$ with ϵ relative error paying shuffle size $O(n^2/\epsilon^2)$ and reduce-key complexity $O(n/\epsilon^2)$. i.e. independent of m.
- Preserve specific entries of $A^T A$, then we can reduce the shuffle size to $O(n \log(n)/s)$ and reduce-key complexity to $O(\log(n)/s)$ where s is the minimum similarity for the entries being estimated. Similarity can be via Cosine, Dice, Overlap, or Jaccard.

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

Ubiquitous problem

MapReduce

- With such large datasets (e.g. $m = 10^{12}$), we must use many machines.
- Biggest clusters of computers use MapReduce
- MapReduce is the tool of choice in such distributed systems
- With so many machines (around 1000), CPU power is abundant, but communication is expensive
- 2 Minute description of MapReduce...

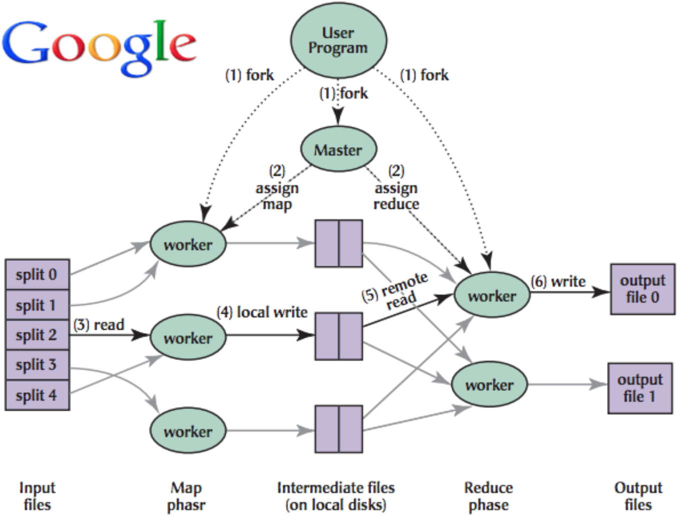
MapReduce


```
map(String key, String value):
    // key: document name
    // value: document contents
    for each word w in value:
        EmitIntermediate(w, "1");

reduce(String key, Iterator values):
    // key: a word
    // values: a list of counts
    int result = 0;
    for each v in values:
        result += ParseInt(v);
    Emit(AsString(result));
```

MapReduce

Google



MapReduce

- Input gets dished out to the mappers roughly equally
- Two performance measures
- 1) Shuffle size: shuffling the data output by the mappers to the correct reducer is expensive
- 2) Largest reduce-key: can't send too much of the data to a single reducer
- First pass at implementing $\cos(i, j)$ in MapReduce...

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 $((c_j, c_k) \to a_{ij}a_{ik})$
end for

Algorithm 2 NaiveReducer($(c_i, c_j), \langle v_1, \dots, v_R \rangle$)

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

DIMSUM Algorithm

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 $((c_j, c_k) \to a_{ij} a_{ik})$
end for

Algorithm 4 DIMSUMReducer($(c_i, c_j), \langle v_1, \ldots, v_R \rangle$)

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

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 $\gamma = \Omega(n/\epsilon^2)$

Analysis for DIMSUM

Some notation

1. $\#(c_i, c_j)$ is the number of times columns i and j have a nonzero in the same dimension
2. $\#(c_i)$ is the number of nonzeros in the vector c_i
3. Theorem will be about $\{0, 1\}$ matrices, but can be generalized

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:

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

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

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.

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

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

Shuffle size for DIMSUM

Proof.

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

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

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

Shuffle size for DIMSUM

Proof.

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

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

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

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

□

Shuffle size for DIMSUM

- It is easy to see via Chernoff bounds that the above shuffle size is obtained with high probability.
- $O(nL\gamma)$ 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 $[0, 1]$
- 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. But 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 high probability.

Proof

- Uses Latala's theorem, bounds 2nd and 4th central moments of entries of B.
- Latala's Theorem. 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,

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

and

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

Relative error guaranteed to be low with high probability.

Correctness

Proof.

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

□

Experiments

- Large scale experiment live at `twitter.com`

- Smaller scale experiment with points as words, and dimensions as tweets
- $m = 200M, n = 1000, L = 10$

Experiments

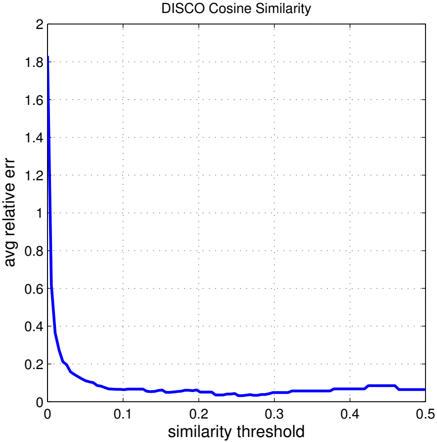


Figure : Average error for all pairs with similarity threshold *s*. DIMSUM estimated Cosine error decreases for more similar pairs.

Experiments

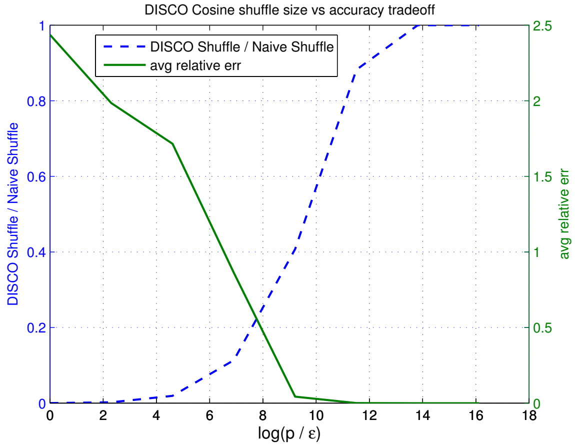


Figure : As $\gamma = p/\epsilon$ increases, shuffle size increases and error decreases. There is no thresholding for highly similar pairs here.

Other Similarity Measures

This all works for many 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. These are bounds for shuffle size without combining. Combining can only bring down these sizes.

Locality Sensitive Hashing

- MinHash from the Locality-Sensitive-Hashing family can have its vanilla implementation greatly improved by DIMSUM.
- Theorems for shuffle size and correctness in paper.

Fin.

- Consider DIMSUM if you ever need to compute $A^T A$ for large sparse A
- Many more experiments and results at `reza-zadeh.com`
- Thanks!