Fast Similarity Graph Construction via Data Sketching Techniques

By: Hoorieh Marefat
Supervisors: Aijun An, Manos Papagelis
Introduction
Motivation

We explore how to build similarity graphs in an efficient way.
Applications

NN Search

Collaborative Filtering

Clustering

Link Prediction
Similarity-based Graphs

- Similarity Graph
- $\epsilon$-Graph
- Nearest Neighbour (NN) Graph
Similarity graph

The Dataset

<table>
<thead>
<tr>
<th>Entities</th>
<th>Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3 0 2 0 10</td>
</tr>
<tr>
<td>2</td>
<td>4 5 1 0 0</td>
</tr>
<tr>
<td>3</td>
<td>1 3 0 1 5</td>
</tr>
<tr>
<td>4</td>
<td>0 2 2 0 3</td>
</tr>
<tr>
<td>5</td>
<td>0 0 0 4 0</td>
</tr>
</tbody>
</table>

Dataset’s Similarity Graph

Entities

Similarity of Entities Based on Attributes
$\epsilon$-graph

$\epsilon = 15$
If we have $k$ nearest neighbours for each node, graph would be a $k$NN graph.
Similarity-based graphs

- Each of these graphs lead to different problems

- Each of them have different solutions

- In our work, we focus on the $\epsilon$-similarity graph, a similarity graph whose edges are above the $\epsilon$ threshold
Similarity Graph Construction Challenges

When we have to build similarity graphs many times like in data streams for different snapshots or windows.

Similarities should be computed for all pairs of entities based on all attributes.

\[ O(n^2d) \]

Scalability

<table>
<thead>
<tr>
<th>UserID</th>
<th>Item1</th>
<th>Item2</th>
<th>Item3</th>
<th>Item4</th>
<th>Item5</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>5</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>3</td>
<td>0</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>0</td>
</tr>
</tbody>
</table>

D Attributes

N Entities
Main Objective

Proposing an **efficient** and **effective** method for **similarity graph construction** from **high-dimensional** data
Approaches

• Distributed solutions on MapReduce

• GPU-based solutions

• Efficient algorithmic optimizations
  • Using inverted index
  • Sampling/sketching based methods

Our work
Inverted index

**Doc1:** New Home Sales

**Doc2:** Home Sales In July

### The Forward Index

<table>
<thead>
<tr>
<th>Doc1</th>
<th>New</th>
<th>Home</th>
<th>Sales</th>
</tr>
</thead>
<tbody>
<tr>
<td>Doc2</td>
<td>Home</td>
<td>Sales</td>
<td>In</td>
</tr>
</tbody>
</table>

### The Inverted Index

<table>
<thead>
<tr>
<th>New</th>
<th>Doc1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Home</td>
<td>Doc1</td>
</tr>
<tr>
<td>Sales</td>
<td>Doc1</td>
</tr>
<tr>
<td>In</td>
<td>Doc2</td>
</tr>
<tr>
<td>July</td>
<td>Doc2</td>
</tr>
</tbody>
</table>
Data Sketching

Summarizing data that might be thought of as a high dimensional vector, or matrix

Data sketches have mathematically proven error bounds
Problem Statement
Sparse Vector Representation of Matrix

The Matrix

<table>
<thead>
<tr>
<th></th>
<th>A1</th>
<th>A2</th>
<th>A3</th>
<th>A4</th>
<th>A5</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1</td>
<td>5</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>R2</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>R3</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>R4</td>
<td>3</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>R5</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
</tbody>
</table>

Its Sparse Vector Representation

- R1 = \{(A1, 5), (A2, 1), (A5, 4)\}
- R2 = \{(A3, 3)\}
- R3 = \{(A3, 2), (A5, 1)\}
- R4 = \{(A1, 3), (A2, 2), (A5, 4)\}
- R5 = \{(A2, 3), (A5, 2)\}
Approximate Similarity Graph Construction

• Given:
  • a similarity threshold $\epsilon$
  • a data matrix or its sparse vector representation

• the problem is:
  • to build a similarity graph $G(V, E)$ where
    • $V$ is the set of entities in the data matrix and
    • $E$ is the set of edges representing the similarity between two nodes and
  • the similarity is above the $\epsilon$ threshold
Proposed Methodology
Overview of the Algorithm

Step 1: Data Sketching

Sk[1] = {(2, 4), (3, 1)}
Sk[2] = {(1, 3)}
Sk[3] = {(1, 2), (2, 1)}
Sk[4] = {(2, 4), (3, 2)}
Sk[5] = {(2, 2), (3, 3)}

Step 2: Pairwise Similarity Computations

<table>
<thead>
<tr>
<th></th>
<th>(Sk[1], 4)</th>
<th>(Sk[3], 1)</th>
<th>(Sk[4], 4)</th>
<th>(Sk[5], 2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>(Sk[2], 3)</td>
<td>(Sk[3], 2)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Step 3: Similarity Graph Construction

```
1 -- 2 -- 3
    |   |   |
    v   v   v
4 --- 5
```
Step 1: Data Sketching

Main goal:
Start from a large dataset
Make it smaller

The dataset

Dataset’s sketches

Sketch size: $k \ll d$
Step 1: Data Sketching

• Input

  • Two different kinds of dataset
    • A data matrix
    • Sparse vector representation of a data matrix
  • The sketch size (k)

• Output

  • Sketches of data
Data Sketching From a Data Matrix

### Dataset: D

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
</tbody>
</table>

### Permuted Dataset

Random Column Permutation:

[1, 2, 3, 4, 5] → [4, 3, 1, 5, 2]

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>4</td>
<td>1</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>4</td>
<td>2</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>2</td>
<td>3</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

**Sketches of D**

- Sketch size (K) = 2
- \( Sk[1] = \{(2, 4), (3, 1)\} \)
- \( Sk[2] = \{(1, 3)\} \)
- \( Sk[3] = \{(1, 2), (2, 1)\} \)
- \( Sk[4] = \{(2, 4), (3, 2)\} \)
- \( Sk[5] = \{(2, 2), (3, 3)\} \)

- \( Sk\_max\_id[1] = 3 \)
- \( Sk\_max\_id[2] = 1 \)
- \( Sk\_max\_id[3] = 2 \)
- \( Sk\_max\_id[4] = 3 \)
- \( Sk\_max\_id[5] = 3 \)

**Sketch Max Column IDs**

Taking the first k column ID/value pairs with nonzero values.

Recording the highest column ID in each sketch.
Data Sketching From Sparse Vector Representation

Original Dataset
- R1 = {(1, 5), (2, 1), (5, 4)}
- R2 = {(3, 3)}
- R3 = {(3, 2), (5, 1)}
- R4 = {(1, 3), (2, 2), (5, 4)}
- R5 = {(2, 3), (5, 2)}

Permutated Dataset
- R1 = {(4, 5), (3, 1), (2, 4)}
- R2 = {(1, 3)}
- R3 = {(1, 2), (2, 1)}
- R4 = {(4, 3), (3, 2), (2, 4)}
- R5 = {(3, 3), (2, 2)}

Column Permutation
- [1, 2, 3, 4, 5] to [4, 3, 1, 5, 2]

Sketch Max IDs
- Sk_max_id[1] = 3
- Sk_max_id[2] = 1
- Sk_max_id[3] = 2
- Sk_max_id[4] = 3
- Sk_max_id[5] = 3

Sketches
- Sk[1] = {(2, 4), (3, 1)}
- Sk[2] = {(1, 3)}
- Sk[3] = {(1, 2), (2, 1)}
- Sk[4] = {(2, 4), (3, 2)}
- Sk[5] = {(2, 2), (3, 3)}
Data Sketching - Time Complexity

- From Data Matrix: $O(d) + O(nm)$

- From Sparse Vector Rep.: $O(d) + O(n \times (f + f \cdot \log(f) + k))$

- $O(n \times (f + f \cdot \log(f) + k)) < O(nm)$
Step 2: Pairwise similarity computations

Input: Data sketches

Output: Similarities of all pairs of sketches

Our similarity measure is the **inner-product**
Structure of An Inverted Index

Sketches of D (k = 2)

- $Sk[1] = \{(2, 4), (3, 1)\}$
- $Sk[2] = \{(1, 3)\}$
- $Sk[3] = \{(1, 2), (2, 1)\}$
- $Sk[4] = \{(2, 4), (3, 2)\}$
- $Sk[5] = \{(2, 2), (3, 3)\}$

The Inverted Index

2 -> $(Sk[1], 4)$, $(Sk[3], 1)$, $(Sk[4], 4)$, $(Sk[5], 2)$
3 -> $(Sk[1], 1)$, $(Sk[4], 2)$, $(Sk[5], 3)$
1 -> $(Sk[2], 3)$, $(Sk[3], 2)$
Pairwise similarity computations

**Sketches**

- $\text{Sk}[1] = \{(2, 4), (3, 1)\}$
- $\text{Sk}[2] = \{(1, 3)\}$
- $\text{Sk}[3] = \{(1, 2), (2, 1)\}$
- $\text{Sk}[4] = \{(2, 4), (3, 2)\}$
- $\text{Sk}[5] = \{(2, 2), (3, 3)\}$

**Inverted Index**

- $2 \rightarrow (\text{Sk}[1], 4), (\text{Sk}[3], 1)$
- $3 \rightarrow (\text{Sk}[1], 1)$
- $1 \rightarrow (\text{Sk}[2], 3), (\text{Sk}[3], 2)$

**Similar Sketches of $\text{Sk}[4]$**

- $(\text{Sk}[1], 16), (\text{Sk}[3], 4)$
- $(\text{Sk}[1], 16+2)$
- $(\text{Sk}[1], 18), (\text{Sk}[3], 4)$
Effective sample size ($d_s$) Computation

Our similarity measure is the **inner-product**

Original data:

$$a = \sum_{i=1}^{d} u_{1,i}u_{2,i}$$

Sketches:

$$\hat{a} = \frac{d}{d_s} \sum_{i=1}^{d_s} \tilde{u}_{1,i}\tilde{u}_{2,i}$$

$$_{d_s} = \min(\max(ID(\tilde{u}_1)), \max(ID(\tilde{u}_2)))$$
Three approaches for $d_S$ computation

**Online**

Sk[4] = {(2, 4), (3, 2)}

**Offline via Sorting**

Sorting sketches based on their max ID

Sk[2] = {(1, 3)}
Sk[3] = {(1, 2), (2, 1)}
Sk[1] = {(2, 4), (3, 1)}
Sk[4] = {(2, 4), (3, 2)}
Sk[5] = {(2, 2), (3, 3)}

**Offline via Matrix Precomputation**

$d_S$ pairwise matrix

![Matrix Diagram]

Preprocessing
Extra Space Needed for $d_s$ computations

- Online: $O(1)$
- Offline via sorting: $O(n)$
- Offline via matrix precomputations: $O(n^2)$
Pairwise Similarity Computation - Time Complexity

- Online: $O(c \times n \times k \times l)$
- Offline via sorting: $O(c' \times n \times k \times l) + O(n \cdot \log(n))$
- Offline via matrix precomputation: $O(c'' \times n \times k \times l) + O(n^2)$

$c', c'' < c$
Step 3: Similarity Graph Construction

Similarities of all pairs of entities
Evaluation
Evaluation Scenarios

- Runtime Cost
- Accuracy
- Effectiveness
## Datasets

<table>
<thead>
<tr>
<th>Name</th>
<th>Distribution</th>
<th>Density</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Normal2</td>
<td>Normal($\mu=200$, $\sigma=50$)</td>
<td>2%</td>
<td>10k × 10k</td>
</tr>
<tr>
<td>Normal4</td>
<td>Normal($\mu=400$, $\sigma=100$)</td>
<td>4%</td>
<td></td>
</tr>
<tr>
<td>Normal6</td>
<td>Normal($\mu=600$, $\sigma=100$)</td>
<td>6%</td>
<td></td>
</tr>
<tr>
<td>Normal8</td>
<td>Normal($\mu=800$, $\sigma=100$)</td>
<td>8%</td>
<td></td>
</tr>
<tr>
<td>Binomial2</td>
<td>Binomial($n=10k$, $p=0.02$)</td>
<td>2%</td>
<td>10k × 10k</td>
</tr>
<tr>
<td>Binomial4</td>
<td>Binomial($n=10k$, $p=0.04$)</td>
<td>4%</td>
<td></td>
</tr>
<tr>
<td>Binomial6</td>
<td>Binomial($n=10k$, $p=0.06$)</td>
<td>6%</td>
<td></td>
</tr>
<tr>
<td>Binomial8</td>
<td>Binomial($n=10k$, $p=0.08$)</td>
<td>8%</td>
<td></td>
</tr>
</tbody>
</table>
Evaluated methods

- **Original**
- **Normal Random Projection (NRP)**
- **Sk_naive** Data sketching without using an inverted index
- **Sk_online**
- **Sk_offline_sorted**
- **Sk_offline_matrix**

Our proposed methods

- sim. operations on the vectors of the original dataset

<table>
<thead>
<tr>
<th>UserID</th>
<th>Item1</th>
<th>Item2</th>
<th>Item3</th>
<th>Item4</th>
<th>Item5</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>3</td>
<td>0</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>5</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>0</td>
</tr>
</tbody>
</table>
Runtime vs. sample size

(a) normal4 dataset

Sample size (%)

Runtime (sec)

62 ×

Sketching-based methods: sketch size (k)

NRP: # dimensions in the projected dataset
Accuracy vs. sample size

Relative error for each pair \((i, j)\):

\[
e_{ij} = \left| \frac{s_{ij}^{\text{sample}} - s_{ij}^{\text{original}}}{s_{ij}^{\text{original}}} \right|
\]

The lower, the better
Effectiveness
Effectiveness

- *K*-nearest neighbours

- Node centrality values
  - show *importance* of the nodes in the graph
  - we work with *eigenvector centrality*

- Node rankings
  - Based on the centrality of the nodes in the graph, we have a ranking for them
  - We use *Spearman’s ranking correlation coefficient* ($\rho$)
Effectiveness - kNN

kNN recall:
- How many of the $k$ real nearest neighbours of each node we are returning
- Precision and recall are the same in this case
Effectiveness - Centrality Errors

Centrality error for node $i$:

$$e_i = \frac{|c_i^{sample} - c_i^{original}|}{c_i^{original}}$$

The lower, the better

(a) normal4 dataset
Effectiveness - Node Ranking Correlations

We use Spearman's ranking correlation coefficient:
A measure to see how well node rankings of the approximated graphs are compared to the original.
Conclusion & Future Work
Summary of contributions

building similarity graphs from high-dimensional data

\[ O(n^2d) \]

efficient, accurate and effective way of construction of similarity graphs

Inverted Index

Data Sketching

<table>
<thead>
<tr>
<th>2</th>
<th>(Sk[1], 4)</th>
<th>(Sk[3], 1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>(Sk[1], 1)</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>(Sk[2], 3)</td>
<td>(Sk[3], 2)</td>
</tr>
</tbody>
</table>
Summary of contributions

three algorithms each of which has a trade-off for speed and space

Time efficient
Accurate

Effective on different graph analysis tasks
Limitations

**Scalability** for very large datasets

**Storage overhead** for the inverted index + **maintenance cost**
Future Work

Providing theoretical bounds for the quality of graph downstream task results

Making the methods distributed to increase scalability

Working with multi-dimensional arrays instead of matrix
Thank You!
Appendix
Online Pairwise Similarities

**Algorithm 2: Online Pairwise Similarities**

**Input:** sketches of $D$: $Sk$, sketch max ids: $Sk\_Max\_Id$, similarity threshold: $\epsilon$, dimensionality of original data: $d$

**Output:** pairwise similarities: $S$

1. $S \leftarrow \emptyset$
2. $I_1, I_2, ..., I_d \leftarrow \emptyset$ ($I_i$ is the entry for the $i$th attribute in the inverted index. It will contain a list of $(x, v)$’s, where $x$ is a row/sketch id and $v$ is the value of $x$ for the $i$th attribute.)
3. for $x \in Sk$ do
4.     $M \leftarrow \emptyset$ ($M$ holds the similarity values between $x$ and each of the skethes before $x$ in $Sk$)
5.     for $(a, v) \in x$ do
6.         for $(y, y_v) \in I_a$ do
7.             $d_s = \min(Sk\_Max\_Id[x], Sk\_Max\_Id[y])$
8.             if $a \leq d_s$ then
9.                 $M[y] \leftarrow M[y] + (d/d_s) \cdot v \cdot y_v$
10.                $I_a \leftarrow I_a \cup \{(x, v)\}$
11.     $S_x \leftarrow \text{Filter\_Similarities}(M, \epsilon)$ (Remove similarities in $M$ whose value is less than $\epsilon$)
12.     $S \leftarrow S \cup (x, S_x)$
13. return $S$
Algorithm 3: Offline Pairwise Similarities via Sorting

Input: sketches of $D$: $Sk$, sketch max ids: $Sk\_Max\_Id$, similarity threshold: $\epsilon$, dimensionality of original data: $d$

Output: pairwise similarities: $S$

1. $S \leftarrow \emptyset$
2. $I_1, I_2, \ldots, I_d \leftarrow \emptyset$
3. $Sorted\_Sk\_Indices \leftarrow \text{arg sort}(Sk\_Max\_Id, \text{ascending})$
4. for $i \in \text{range}(|Sk|)$ do
   5. $x_{id} = Sorted\_Sk\_Indices[i]$
   6. $x \leftarrow Sk[x_{id}]$
   7. $M \leftarrow \emptyset$
   8. for $(a, v) \in x$ do
      9. for $(y, y_v) \in I_a$ do
         10. $d_s = Sk\_Max\_Id[y]$
         11. $M[y] \leftarrow M[y] + (d/s) \cdot v \cdot y_v$
         12. $I_a \leftarrow I_a \cup \{(x, v)\}$
   13. $S_x \leftarrow \text{Filter\_Similarities}(M, \epsilon)$
   14. $S \leftarrow S \cup (x, S_x)$
5. return $S$
Offline Pairwise Similarities via Matrix Precomputations

Algorithm 4: Offline Pairwise Similarities via Matrix Precomputations

Input: sketches of $D$: $Sk$, sketch max ids: $Sk\_Max\_Id$, similarity threshold: $\epsilon$, dimensionality of original data: $d$

Output: pairwise similarities: $S$

1 $S \leftarrow \emptyset$
2 $I_1, I_2, \ldots, I_d \leftarrow \emptyset$
3 $n \leftarrow |Sk|$
4 for $i \in \text{range}(n)$ do
5     for $j \in \text{range}(n)$ do
6         $d_s.pairwise[i][j] \leftarrow \min(Sk\_Max\_Id[i], Sk\_Max\_Id[j])$
7 for $x \in Sk$ do
8     $M \leftarrow \emptyset$
9     for $(a, v) \in x$ do
10        for $(y, y_v) \in I_a$ do
11            $d_s = d_s.pairwise[x][y]$
12            if $a < d_s$ then
13                $M[y] \leftarrow M[y] + (d/ds) \cdot v \cdot y_v$
14                $I_a \leftarrow I_a \cup \{(x, v)\}$
15     $S_x \leftarrow \text{Filter\_Similarities}(M, \epsilon)$
16     $S \leftarrow S \cup (x, S_x)$
17 return $S$
Bloom filters for set summarization

• Set membership

• The item has *definitely not been stored*, or the item has *probably been stored*

• Having $k$ hash functions and map each item with each of them

• Set all the corresponding bits to 1. If all were one for an item, say it is a member, if any of them were 0, say it is not a member
Counting with count-min sketch

- Counts the number of items of a certain type

- Sketch: an array of counters, and a set of hash functions which map items into the array

- Count of the desired item is to take the smallest of counters in each row as our estimate.
PCA vs. RP

• PCA

  • Extracting a small number of directions from the data which captures most of variation of dataset

  • Finding the direction requires finding eigenvectors of the covariance matrix

• Random projection

  • Rather than finding “the best” directions, it suffices to use random vectors.

  • Picking a moderate number of random directions captures a comparable amount of variation, while requiring much less computation.