
David C. Anastasiu\textsuperscript{1,2}, Huzefa Rangwala\textsuperscript{3}, and Andrea Tagarelli\textsuperscript{4}

\textsuperscript{1}Computer Engineering, San Jose State University, CA
\textsuperscript{2}Computer Science & Engineering, Santa Clara University, CA
\textsuperscript{3}Computer Science & Engineering, George Mason University, VA
\textsuperscript{3}DIMES, University of Calabria, Italy

25th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, August 2019
Part V: Filtering-Based Search

David C. Anastasiu, San José State University  [ david.anastasiu@sjsu.edu ]

Starting September:
Department of Computer Science and Engineering
Santa Clara University
Tutorial Outline

- **Part I: Problems and Data Types**
  - Dense, sparse, and asymmetric data
  - Bounded nearest neighbor search
  - Nearest neighbor graph construction
  - Classical approaches and limitations

- **Part II: Neighbors in Genomics, Proteomics, and Bioinformatics**
  - Mass spectrometry search
  - Microbiome analysis

- **Part III: Approximate Search**
  - Locality sensitive hashing variants
  - Permutation and graph-based search
  - Maximum inner product search

- **Part IV: Neighbors in Advertising and Recommender Systems**
  - Collaborative filtering at scale
  - Learning models based on the neighborhood structure

- **Part V: Filtering-Based Search**
  - Massive search space pruning by partial indexing
  - Effective proximity bounds and when they are most useful

- **Part VI: Neighbors in Learning and Mining Problems in Graph Data**
  - Neighborhood as cluster in a complex network system
  - Neighborhood as influence trigger set
Talk Outline

• What is filtering-based search?

• Massive search space pruning by partial indexing [and other pruning strategies]
  • So what in the world is partial indexing?
  • Search using a partial index
  • The case of k-NNG construction
  • Approximate methods could use filtering too
  • What if we used parallelism
  • When both length and angles matter

• Effective proximity bounds and when they are most useful
  • Not all pruning is created equal
  • When less is more

• Open questions
What is filtering-based search?

• Given a similarity bounding threshold, there is no need to compute the full similarity between a pair of vectors to tell if they are similar enough
  • Compute an upper bound similarity estimate
  • Prune/filter the pair if the similarity estimate is below the threshold

\[ \widehat{\text{sim}}(q, c) \geq \text{sim}(q, c) \]

\[
\text{if: } \widehat{\text{sim}}(q, c) < \epsilon \\
\text{then: } \text{sim}(q, c) < \epsilon \\
\text{prune}
\]
How to prune the search space

- All Pairwise Similarities
- Sparsity
- Vector lengths
- Vector angles
- Angles & lengths
- True neighbors
Take advantage of sparsity

**Input matrix**

<table>
<thead>
<tr>
<th></th>
<th>(f_1)</th>
<th>(f_2)</th>
<th>(f_3)</th>
<th>(f_4)</th>
<th>(f_5)</th>
<th>(f_6)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(d_1)</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>(d_2)</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(d_3)</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(d_4)</td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(d_5)</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[
T(d_i, d_j) = \frac{\langle d_i, d_j \rangle}{||d_i||_2^2 + ||d_j||_2^2 - \langle d_i, d_j \rangle}
\]

\[
C(d_i, d_j) = \frac{\langle d_i, d_j \rangle}{||d_i||_2 \times ||d_j||_2}
\]

\[
\langle d_2, d_3 \rangle = d_{2,1} \times d_{3,1} + d_{2,2} \times d_{3,2} + d_{2,3} \times d_{3,3} + d_{2,4} \times d_{3,4} + d_{2,5} \times d_{3,5} + d_{2,6} \times d_{3,6}
\]
Index and Accumulator: a match made in heaven

- **Inverted index**: set of lists, one for each feature, containing documents and their associated values

```
A[d_2] += d_{3,1} \times d_{2,1}
A[d_5] += d_{3,1} \times d_{5,1}
A[d_1] += d_{3,2} \times d_{1,2}
A[d_4] += d_{3,2} \times d_{4,2}
A[d_5] += d_{3,2} \times d_{5,2}
```

[...]

```
d_1
d_2
\infty
d_4
d_5
```

Accumulator
IdxJoin: A straight-forward solution

• Method:
  
  Compute and store vector norms
  Construct an inverted index from the objects
  *For each query object:*
  • Compare only with objects with features in common
  • Select neighbors

• Results in *EXACT* solution

• Advantage:
  • Skips some object comparisons and many meaningless multiply-adds
## Datasets

<table>
<thead>
<tr>
<th>Dataset</th>
<th>$n$</th>
<th>$m$</th>
<th>$nnz$ (M)</th>
<th>$mrl$</th>
<th>$mcl$</th>
<th>type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Patents</td>
<td>100,000</td>
<td>759,044</td>
<td>46.3</td>
<td>464</td>
<td>61</td>
<td>text</td>
</tr>
<tr>
<td>WW100k</td>
<td>100,528</td>
<td>339,944</td>
<td>79</td>
<td>787</td>
<td>233</td>
<td>text</td>
</tr>
<tr>
<td>Twitter</td>
<td>146,170</td>
<td>143,469</td>
<td>200</td>
<td>1370</td>
<td>1395</td>
<td>graph</td>
</tr>
<tr>
<td>WW500</td>
<td>243,223</td>
<td>660,600</td>
<td>202</td>
<td>830</td>
<td>306</td>
<td>text</td>
</tr>
<tr>
<td>MLSMR</td>
<td>325,164</td>
<td>20,021</td>
<td>56.1</td>
<td>173</td>
<td>2803</td>
<td>chem</td>
</tr>
<tr>
<td>WW500k</td>
<td>494,244</td>
<td>343,622</td>
<td>197</td>
<td>399</td>
<td>574</td>
<td>text</td>
</tr>
<tr>
<td>RCV1</td>
<td>804,414</td>
<td>43,001</td>
<td>61</td>
<td>76</td>
<td>1417</td>
<td>text</td>
</tr>
<tr>
<td>WW200</td>
<td>1,017,531</td>
<td>663,419</td>
<td>437</td>
<td>430</td>
<td>659</td>
<td>text</td>
</tr>
<tr>
<td>Wiki</td>
<td>1,815,914</td>
<td>1,648,879</td>
<td>44</td>
<td>24</td>
<td>27</td>
<td>graph</td>
</tr>
<tr>
<td>Orkut</td>
<td>3,072,626</td>
<td>3,072,441</td>
<td>223</td>
<td>73</td>
<td>73</td>
<td>graph</td>
</tr>
<tr>
<td>SC</td>
<td>11,519,370</td>
<td>7,415</td>
<td>1,784.5</td>
<td>155</td>
<td>262,669</td>
<td>chem</td>
</tr>
</tbody>
</table>
The sparsity advantage: IndexJoin vs. $n^2/2$

<table>
<thead>
<tr>
<th>Dataset</th>
<th># sims</th>
<th>% all sims</th>
</tr>
</thead>
<tbody>
<tr>
<td>Orkut</td>
<td>23,308,569,172</td>
<td>0.12</td>
</tr>
<tr>
<td>Wiki</td>
<td>71,700,509,475</td>
<td>1.09</td>
</tr>
<tr>
<td>Twitter</td>
<td>8,516,651,351</td>
<td>19.93</td>
</tr>
<tr>
<td>RCV1</td>
<td>289,612,531,857</td>
<td>22.38</td>
</tr>
<tr>
<td>WW100k</td>
<td>5,052,763,937</td>
<td>25.00</td>
</tr>
<tr>
<td>WW500k</td>
<td>122,095,368,297</td>
<td>25.00</td>
</tr>
</tbody>
</table>

- A large number of comparisons are filtered by the sparsity constraints
- For Orkut and Wiki, 99% or more of the comparisons are simply ignored
LSH vs. IndexJoin

- In all experiments, LSH parameters were tuned to achieve at least 95% accuracy.
- LSH outperforms IndexJoin at high thresholds.
- Performs poorly at low thresholds and for high dimensional datasets (Orkut, Wiki).
A prelude: prefix and suffix vectors

\[ \mathbf{d}_j \]

\[
\begin{bmatrix}
\mathbf{f}_1 & \mathbf{f}_2 & \mathbf{f}_3 & \mathbf{f}_4 & \mathbf{f}_5 & \mathbf{f}_6 \\
.27 & & .72 & & .64 \\
\end{bmatrix}
\]

\[ \mathbf{d}_{\leq p}^j \]

\[
\begin{bmatrix}
\mathbf{f}_1 & \mathbf{f}_2 & \mathbf{f}_3 & \mathbf{f}_4 & \mathbf{f}_5 & \mathbf{f}_6 \\
.27 & & & & & \\
\end{bmatrix}
\]

\[ \mathbf{d}_{> p}^j \]

\[
\begin{bmatrix}
\mathbf{f}_1 & \mathbf{f}_2 & \mathbf{f}_3 & \mathbf{f}_4 & \mathbf{f}_5 & \mathbf{f}_6 \\
 & & .72 & & .64 \\
\end{bmatrix}
\]

\[
\mathbf{d}_i = \mathbf{d}_{\leq p}^i + \mathbf{d}_{> p}^i
\]

\[
\| \mathbf{d}_i \|_0 = \| \mathbf{d}_{\leq p}^i \|_0 + \| \mathbf{d}_{> p}^i \|_0
\]

\[
\| \mathbf{d}_i \|_2 = \| \mathbf{d}_{\leq p}^i \|_2 + \| \mathbf{d}_{> p}^i \|_2
\]

\[
\langle \mathbf{d}_i, \mathbf{d}_j \rangle = \langle \mathbf{d}_i, \mathbf{d}_{\leq p}^j \rangle + \langle \mathbf{d}_i, \mathbf{d}_{> p}^j \rangle
\]

\[
\langle \mathbf{d}_i, \mathbf{d}_j \rangle = \langle \mathbf{d}_{\leq p}^i, \mathbf{d}_{\leq p}^j \rangle + \langle \mathbf{d}_{> p}^i, \mathbf{d}_{> p}^j \rangle
\]
Main idea:

Only need to index enough non-zeros to guarantee correct result.

\[
\text{sim}(d_1^\leq, d_c) \leq \|d_1^\leq\|_2 \times \|d_c\|_2 \\
< \|d_1^\leq\|_2 \times 1 \\
< \epsilon
\]

- We’ll focus initially on the min-\(\epsilon\) graph construction problem.
Partial indexing in practice

- L2AP indexes fewer non-zeros than previous approaches
- Leads to greatly improved execution runtime
Search using a partial index

L2AP follows a two-step process:

1. Accumulate similarity using partial inverted index
   - Only need to compute a subset of similarities:
     - Can do further filtering

2. For each un-pruned object, finish similarity computation using forward index
   - \( \mathbf{D} = \mathbf{D}^T + \mathbf{D} \mathbf{A}^T + \mathbf{D} \mathbf{B}^T \)
   - \( \text{sim}(d_q, d_3) \) & \( \text{sim}(d_q, d_5) \)

http://davidanastasiu.net/software/l2ap/
Angle/Suffix Filtering

Filter/prune object pairs not in final graph based on similarity estimates

\[
\langle d_q^p, d_c^p \rangle \leq \|d_q^p\|_2 \times \|d_c^p\|_2
\]

(Cauchy-Schwarz inequality)

Filter \( \operatorname{sim}(d_q, d_c) \) if

\[
A[d_c] + \|d_q^p\|_2 \times \|d_c^p\|_2 < \epsilon
\]
Angle/Suffix Filtering

Filter/prune object pairs not in final graph based on similarity estimates

\[ \langle \mathbf{d}_q^p, \mathbf{d}_c^p \rangle \leq \min(||\mathbf{d}_q^p||_0, ||\mathbf{d}_c^p||_0) \times ||\mathbf{d}_q^p||_\infty \times ||\mathbf{d}_c^p||_\infty \]

Filter \( \text{sim}(d_q, d_c) \) if

\[ A[d_c] + \min(||\mathbf{d}_q^p||_0, ||\mathbf{d}_c^p||_0) \times ||\mathbf{d}_q^p||_\infty \times ||\mathbf{d}_c^p||_\infty > \epsilon \]
But won’t it take longer to compute those norms?

- We pre-compute all necessary norms and store them in a compressed sparse row (CSR) –like data structure
  - \(O(nnz)\) time and space for this step

<table>
<thead>
<tr>
<th>rowval</th>
<th>.75</th>
<th>.25</th>
<th>.25</th>
<th>.50</th>
<th>.25</th>
</tr>
</thead>
<tbody>
<tr>
<td>.27</td>
<td>.72</td>
<td>.64</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.72</td>
<td>.49</td>
<td>.49</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.67</td>
<td>.33</td>
<td>.67</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.65</td>
<td>.44</td>
<td>.44</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| l2norm | .66  | .61  | .56  | .25  | .00  | .96  | .64  | .00  | .69  | .49  | .00  | .75  | .67  | .00  | .76  | .62  | .44  | .00  |
|--------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|
| rowind | 1    | 2    | 3    | 4    | 5    | 0    | 3    | 5    | 0    | 1    | 4    | 1    | 3    | 5    | 0    | 1    | 3    | 4    |
| rowptr | 0    | 5    | 8    | 11   | 14   | 18   |

Same idea for \(\|d^p\|_0\), or \(\|d^p\|_\infty\)
Filtering in practice

- L2AP filters most objects without computing their similarity
How fast is it?

- L2AP outperforms all exact and most approximate baselines
The case of k-NNG construction

• In the k-NN problem, we don’t have a nice global minimum similarity threshold we can use for filtering
  • It exists, but
    • (1) we don’t know it
    • (2) it is likely too low to make a difference
• We can use local incomplete (approximate) neighborhood thresholds
• L2Knng strategy:
  • Build an initial approximate graph $\hat{G}$
    • Provides thresholds for filtering
  • Improve $\hat{G}$ until exact
    • Search for objects that can improve neighborhoods

http://davidanastasiu.net/software/l2knng/
Step 1: Approximate graph construction

a) Heuristically choose candidates likely to succeed
   • Find an initial neighborhood for each object

   - Sort vectors and inverted lists in non-increasing weight order
   - Traverse inverted lists and gather $\mu \geq k$ candidates

   $C_{\mu=3} = [d_5, d_1, d_4]$

   • Compute similarities with candidates and keep the $k$ nearest neighbors

   • This step results in an initial approximate graph
Step 1: Approximate graph construction

b) Improve initial approximate graph

• Find potential better neighbors for each object ($\gamma$ times)
  • Visit neighbors in non-increasing similarity order
  • Consider $d_s$, a neighbor of $d_r$, as a candidate if:
    • Have collected less than $\mu$ candidates
    • $\text{sim}(d_s, d_r) \geq \text{sim}(d_r, d_q)$
  • Compute similarities with candidates and update both neighborhoods of $d_s$ and $d_q$

$$C_{\mu=3}(d_1) = [d_6, d_3, d_4]$$
How important is Step 1?

Influence of initial graph quality toward exact graph construction
Step 2: Filtering

• For each query object $d_q$,
  • Find previously processed objects such that:
    • $sim(d_c, d_q) > \sigma_{d_q}$: can improve query obj. neighborhood
    • $sim(d_c, d_q) > \sigma_{d_c}$: can improve neighborhood of previously processed objects
  • Verify list of candidate objects:
    • prune object pair as soon as possible
    • else update neighborhoods of both objects
  • Index processed query object

• Caveats
  • Neighborhoods stored in max-heaps
  • Index tiling used to improve neighborhoods quicker
How well does the filtering work?

Number of candidates pruned in different stages of the filtering framework

RCV1-400k

WW200-250k
How fast is it?

Approximate Baselines

- NN-Descent
- Greedy Filtering
- L2KnnngApprox
- L2Knnng

Exact Baselines

- kldxJoin
- kL2AP
- Maxscore
- BMM
- L2Knnng

Total time (s), log-scaled

k

10 25 50 75 100

10 25 50 75 100

10 25 50 75 100

10 25 50 75 100

10 25 50 75 100

10 25 50 75 100

10 25 50 75 100

10 25 50 75 100

10 25 50 75 100

10 25 50 75 100

10 25 50 75 100

10 25 50 75 100

10 25 50 75 100

10 25 50 75 100

10 25 50 75 100

10 25 50 75 100

10 25 50 75 100

10 25 50 75 100

10 25 50 75 100

10 25 50 75 100

10 25 50 75 100

10 25 50 75 100

10 25 50 75 100

10 25 50 75 100

10 25 50 75 100

10 25 50 75 100

10 25 50 75 100

10 25 50 75 100

10 25 50 75 100

10 25 50 75 100

10 25 50 75 100

10 25 50 75 100

10 25 50 75 100

10 25 50 75 100

10 25 50 75 100

10 25 50 75 100

10 25 50 75 100

10 25 50 75 100

10 25 50 75 100

10 25 50 75 100

10 25 50 75 100

10 25 50 75 100

10 25 50 75 100

10 25 50 75 100

10 25 50 75 100

10 25 50 75 100

10 25 50 75 100

10 25 50 75 100

10 25 50 75 100

10 25 50 75 100

10 25 50 75 100

10 25 50 75 100

10 25 50 75 100

10 25 50 75 100

10 25 50 75 100

10 25 50 75 100

10 25 50 75 100

10 25 50 75 100

10 25 50 75 100

10 25 50 75 100

10 25 50 75 100

10 25 50 75 100

10 25 50 75 100

10 25 50 75 100

10 25 50 75 100

10 25 50 75 100

10 25 50 75 100

10 25 50 75 100

10 25 50 75 100

10 25 50 75 100

10 25 50 75 100

10 25 50 75 100

10 25 50 75 100

10 25 50 75 100

10 25 50 75 100

10 25 50 75 100

10 25 50 75 100

10 25 50 75 100

10 25 50 75 100

10 25 50 75 100

10 25 50 75 100

10 25 50 75 100

10 25 50 75 100

10 25 50 75 100

10 25 50 75 100

10 25 50 75 100

10 25 50 75 100

10 25 50 75 100

10 25 50 75 100

10 25 50 75 100

10 25 50 75 100

10 25 50 75 100

10 25 50 75 100

10 25 50 75 100

10 25 50 75 100

10 25 50 75 100

10 25 50 75 100

10 25 50 75 100

10 25 50 75 100

10 25 50 75 100

10 25 50 75 100
Approximate methods could use filtering too

• CANN applies the ideas in L2AP and L2Knnng-a to the approximate min-\(\epsilon\) graph construction problem

• Approximate solution, in 2 steps:
  1. Construct approximate min-\(\epsilon\) \(k\)-NN graph \(\mathcal{G}\)
     1) Heuristically choose objects that area likely neighbors:
        a. Build partial inverted index for min-\(\epsilon\) search
        b. Sort query vectors and partial inverted index in decreasing order
        c. Choose objects with high weights in common
  2. Use \(\mathcal{G}\) to construct final min-\(\epsilon\) NN graph
     1) Zero or more graph improvement steps
        a. Chose candidates among the neighbors of my neighbors that have higher similarity with my neighbor than me and my neighbor do
Filtering helps improve efficiency

- L2-Norm bound is most useful (ignore others)
- Helpful to hash the query vector (e.g., make it dense)

Bounded similarity computation with pruning:

```
1: function BOUNDED_SIM(d_q, d_c, \epsilon)
2:     s ← 0
3:     for each j = 1, ..., m s.t. d_{c,j} > 0 do
4:         if d_{q,j} > 0 then
5:             s ← s + d_{q,j} \times d_{c,j}
6:         if s + \|d_q^>^j\| \times \|d_c^>^j\| < \epsilon then
7:             return -1
8:     return s
```

\[
\langle d_q, d_c \rangle = \langle d_q^{\leq p}, d_c^{\leq p} \rangle + \langle d_q^{> p}, d_c^{> p} \rangle
\]
How fast is it?

Recall = 0.9
What if we used parallelism?

• Many processes/threads means potential contention over data structures or output space
  • Must carefully design methods that delineate independent work for the threads while ensuring load balance

• If all threads have working data, they may overwhelm the cache and cause delays due to cache misses
  • Cache tiling and memory-efficient data structures can help reduce the cache footprint of each thread
Cache tiling

\[ \zeta \text{ non-zeros per index} \]

\[ \eta \text{ objects in each query block} \]
Partial linear overflow scan during collision lookup.
Neighborhood updates

• Local neighborhood updated during search
• Candidate neighborhood updates staged for cooperative update at the end of query block
Tiling in practice

$pL2AP$ – tiled index
$pL2AP_{rr}$ – full inverted index

### Load Balance

![Load Balance Chart]

### Percent Instructions Leading to Cache Misses (Collisions)

**24 threads, $\varepsilon = 0.3$**

- RCV1, $pL2AP$
- RCV1, $pL2AP_{rr}$
- WW200, $pL2AP$
- WW200, $pL2AP_{rr}$
How fast is it?

24 threads @ 2.5 GHz Intel Xeon E5-2680v3 \w 30 Mb Cache
When both length and angles matter

- Tanimoto min-\( \epsilon \) NNG Construction

For each object \( d_i \) from a set \( D \), find all neighbors \( d_j \) with \( T(d_i, d_j) \geq \epsilon \).

\[
T(d_i, d_j) = \frac{\langle d_i, d_j \rangle}{\|d_i\|^2 + \|d_j\|^2 - \langle d_i, d_j \rangle}
\]

http://davidanastasiu.net/software/tapnn/
Length-based filtering

• $d_q$ and $d_c$ cannot be neighbors unless

\[ \|d_c\| \in [(1/\alpha)\|d_q\|, \alpha\|d_q\|], \]

\[ \alpha = \frac{1}{2} \left( (1 + \frac{1}{\epsilon}) + \sqrt{(1 + \frac{1}{\epsilon})^2 - 4} \right) \]

• $\alpha$ bound due to Marzena Kryszkiewicz, IIDS 2013

• Relabel objects in non-decreasing length order

\[
\begin{array}{ccccccc}
& f_1 & f_2 & f_3 & f_4 & f_5 & f_6 \\
\hline
\boxed{d_1} & d_2 & d_1 & d_1 & d_1 & d_1 & d_1 \\
\boxed{d_2} & \boxed{d_3} & \boxed{d_3} & d_2 & d_3 & d_2 & d_2 \\
\boxed{d_3} & \boxed{d_4} & \boxed{d_4} & d_4 & d_5 & d_4 & d_4 \\
d_5 & d_5 & d_5 & d_5 & d_5 & d_5 & d_5
\end{array}
\]

Inverted Index
Subset of cosine neighborhood

- The following inequalities hold for our domain:
  \[ T(d_i, d_j) \leq C(d_i, d_j) \]
  \[ T(d_i, d_j) \geq \epsilon \Rightarrow C(d_i, d_j) \geq \epsilon \]
  \[ C(d_i, d_j) < \epsilon \Rightarrow T(d_i, d_j) < \epsilon \]

- Potential solution
  - Store vector norms and normalize vectors
  - Find cosine neighbors (L2AP-like filtering here)
  - Transform \( C(d_i, d_j) \) to \( T(d_i, d_j) \)
  - Remove non-Tanimoto neighbors

- Tighter bound due to Lee et al., DEXA 2010
  \[ T(d_i, d_j) \geq \epsilon \Rightarrow C(d_i, d_j) \geq \frac{2\epsilon}{1 + \epsilon} = t \]
Length + Angle-Based Pruning

• \(d_q\) and \(d_c\) cannot be neighbors unless

\[
\|d_c\| \in [(1/\beta) \|d_q\|, \beta \|d_q\|], \quad \beta = \frac{s}{t} + \sqrt{\left(\frac{s}{t}\right)^2 - 1}
\]

where \(s\) is any cosine similarity upper bound such as the ones we compute during filtering.
How fast is it?

Scales well as data set size increases.
Effective proximity bounds and when they are most useful
What will the output look like?

And, a related question, how do I choose $\epsilon/k$?

- Output of similarity search/graph construction is data-dependent
  - For some data sets, you get no neighbors at $\epsilon = 0.95$ cosine similarity; for others, you get many neighbors
  - A given $\epsilon$ threshold means different things in different contexts
- Number of neighbors is dependent on dimensionality
  - By the pigeon hole principle, when $n \gg m$, more likely to see collisions (features in common)
- Filtering effectiveness is dependent on stdev of feature weights
  - If all features weight the same, it will take longer to accumulate similarity
- Parameter choices are often dependent on subsequent analysis that the neighbors are sought for
Pruning Effectiveness Comparison

**Cosine**

<table>
<thead>
<tr>
<th>Percent Pruned</th>
<th>0.3</th>
<th>0.4</th>
<th>0.5</th>
<th>0.6</th>
<th>0.7</th>
<th>0.8</th>
<th>0.9</th>
</tr>
</thead>
<tbody>
<tr>
<td>l2cg</td>
<td>80</td>
<td>70</td>
<td>60</td>
<td>50</td>
<td>40</td>
<td>30</td>
<td>20</td>
</tr>
<tr>
<td>cvlen</td>
<td>20</td>
<td>30</td>
<td>40</td>
<td>50</td>
<td>60</td>
<td>70</td>
<td>80</td>
</tr>
<tr>
<td>other</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

**Tanimoto**

<table>
<thead>
<tr>
<th>Percent Pruned</th>
<th>0.3</th>
<th>0.4</th>
<th>0.5</th>
<th>0.6</th>
<th>0.7</th>
<th>0.8</th>
<th>0.9</th>
</tr>
</thead>
<tbody>
<tr>
<td>l2cg</td>
<td>70</td>
<td>60</td>
<td>50</td>
<td>40</td>
<td>30</td>
<td>20</td>
<td>10</td>
</tr>
<tr>
<td>cvlen</td>
<td>30</td>
<td>40</td>
<td>50</td>
<td>60</td>
<td>70</td>
<td>80</td>
<td>90</td>
</tr>
<tr>
<td>other</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

RCV1
### Neighborhood Graph Statistics

<table>
<thead>
<tr>
<th>$\epsilon$</th>
<th>$\mu$</th>
<th>$\rho$</th>
<th>$\mu$</th>
<th>$\rho$</th>
<th>$\mu$</th>
<th>$\rho$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>WW500k</td>
<td>RCV1</td>
<td>Orkut</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.1</td>
<td>1,749</td>
<td>3.5e-03</td>
<td>10,986</td>
<td>1.4e-02</td>
<td>76</td>
<td>2.5e-05</td>
</tr>
<tr>
<td>0.2</td>
<td>233</td>
<td>4.7e-04</td>
<td>2,011</td>
<td>2.5e-03</td>
<td>21</td>
<td>6.9e-06</td>
</tr>
<tr>
<td>0.3</td>
<td>64</td>
<td>1.3e-04</td>
<td>821</td>
<td>1.0e-03</td>
<td>7.2</td>
<td>2.4e-06</td>
</tr>
<tr>
<td>0.4</td>
<td>25</td>
<td>5.1e-05</td>
<td>355</td>
<td>4.4e-04</td>
<td>2.3</td>
<td>7.6e-07</td>
</tr>
<tr>
<td>0.5</td>
<td>10</td>
<td>2.2e-05</td>
<td>146</td>
<td>1.8e-04</td>
<td>0.69</td>
<td>2.3e-07</td>
</tr>
<tr>
<td>0.6</td>
<td>4.7</td>
<td>9.5e-06</td>
<td>57</td>
<td>7.2e-05</td>
<td>0.22</td>
<td>7.2e-08</td>
</tr>
<tr>
<td>0.7</td>
<td>2.1</td>
<td>4.2e-06</td>
<td>25</td>
<td>3.2e-05</td>
<td>0.09</td>
<td>3.1e-08</td>
</tr>
<tr>
<td>0.8</td>
<td>0.93</td>
<td>1.9e-06</td>
<td>14</td>
<td>1.8e-05</td>
<td>0.07</td>
<td>2.1e-08</td>
</tr>
<tr>
<td>0.9</td>
<td>0.28</td>
<td>5.7e-07</td>
<td>8.1</td>
<td>1.0e-05</td>
<td>0.06</td>
<td>2.0e-08</td>
</tr>
</tbody>
</table>

$\mu$: Average neighborhood size  
$\rho$: Output graph density
### Not all pruning is created equal

- Designed many filtering criteria for the min-$\epsilon$ and $k$-NN problems. E.g.,

<table>
<thead>
<tr>
<th>bound</th>
<th>stage</th>
<th>target</th>
<th>estimate</th>
</tr>
</thead>
<tbody>
<tr>
<td>$idx$</td>
<td>idx</td>
<td>$\text{sim}(d_q^{\leq j}, d_{&gt;q})$</td>
<td>$\min(\langle d_q^{\leq j}, mx_{\geq q} \rangle, |d_q^{\leq j}|_2)$</td>
</tr>
<tr>
<td>$sz$</td>
<td>c.g.</td>
<td>$\min(</td>
<td></td>
</tr>
<tr>
<td>$rs$</td>
<td></td>
<td>$\text{sim}(d_q^{\leq j}, d_{&lt;q})$</td>
<td>$\min(\langle d_q^{\leq j}, mx \rangle,</td>
</tr>
<tr>
<td>$l2cg$</td>
<td></td>
<td>$\text{sim}(d_q^{&lt;j}, d_c^{&lt;j})$</td>
<td>$|d_q^{&lt;j}|_2 \times</td>
</tr>
<tr>
<td>$ps$</td>
<td>c.v.</td>
<td>$\text{sim}(d_q, d_c^{\leq})$</td>
<td>$\min(\langle d_c^{\leq}, mx_{\geq c} \rangle,</td>
</tr>
<tr>
<td>$dps_1$</td>
<td></td>
<td>$\text{sim}(d_q, d_c^{\leq})$</td>
<td>$\min(</td>
</tr>
<tr>
<td>$dps_2$</td>
<td></td>
<td>$\text{sim}(d_q, d_c^{\leq})$</td>
<td>$\min(</td>
</tr>
<tr>
<td>$l2cv$</td>
<td></td>
<td>$\text{sim}(d_q^{&lt;j}, d_c^{&lt;j})$</td>
<td>$|d_q^{&lt;j}|_2 \times</td>
</tr>
</tbody>
</table>

- Some of the criteria are problem-specific (L2-Norm-bound is not)
- Of all criterial, the L2-Norm bound is the most productive (by far)
- Some have suggested less pruning may be more efficient
  - E.g., De Francisci Morales & Gionis, VLDB’16 (extended L2AP to streaming case)
  - May be data specific, but has not been my finding so far
When less is more

- The amount of pruning is not directly proportional to efficiency
Some filtering may cover other filtering
Summary and Open Questions
In summary

• Creating sparsity-aware algorithms goes a long way towards efficient solutions to hard problems

• Filtering is a very effective technique for similarity search, especially for sparse data and asymmetric proximity measures

• L2-norm filtering is extremely effective for cosine similarity and Tanimoto coefficient – may also be beneficial in other proximity measures (e.g., Euclidean distance)

• More research is needed to:
  • Derive new even tighter filtering bounds
  • Identify optimum balance between checking and not checking bounds
  • Characterize NNS pruning and output based on feature statistics
    • Yuliang Li et al. (ICDT2019) prove optimality guarantees for L2-norm filtering of skewed data
    • They also propose alternate and partial inverted list traversal + lower-bound filters
Questions?
References


References


