Getting Started on LSH

Vinicius Vielmo Cogo

Navtalks – November 18, 2016
Locality Sensitive Hashing

LSH is an efficient algorithm to find similar objects using hashes
Recommendation Algorithms

Customers who bought this object also bought …

![Snake Oil Bottle](image)

- **Dr. Fakerington’s Genuine, Pure Snake Oil**
  - Relieves Instantaneously
  - Headache, Toothache, Backache, Earache, Swelling, Inflammation, Fever, Restless Sleep, Chest Pain
  - Glorious Health
  - **50€**

- **Dr. Fakerington’s Genuine, Pure Snake Oil**
  - Relieves Instantaneously
  - Headache, Toothache, Backache, Earache, Swelling, Inflammation, Fever, Restless Sleep, Chest Pain
  - Glorious Health
  - **35€**

- **Dr. Fakerington’s Genuine, Pure Snake Oil**
  - Relieves Instantaneously
  - Headache, Toothache, Backache, Earache, Swelling, Inflammation, Fever, Restless Sleep, Chest Pain
  - Glorious Health
  - **20€**
Recommendation Algorithms

- facebook → ... people you may know
- tinder → ... people you may like
- YouTube → ... videos you may like
- NETFLIX → ... movies you may like
- Spotify → ... music you may like
- amazon → ... products you may like
Recommendation Algorithms

Example:
Suggest something to users.

1) Find similar users (with similar preferences)
   Comparing the list of things they like

2) Suggest what one likes and the other doesn’t know yet
Example: Music Recommendation

P1

P2

P3
Example: Music Recommendation

Jaccard Distance*

Similarity = \frac{|A \cap B|}{|A \cup B|}

* See extra slides for other distances
Example: Music Recommendation

\[
\frac{|P_1 \circ P_2|}{|P_1 \cup P_2|} = \frac{4}{6} = 0.667
\]

\[
\frac{|P_2 \circ P_3|}{|P_2 \cup P_3|} = \frac{1}{8} = 0.125
\]

\[
\frac{|P_1 \circ P_3|}{|P_1 \cup P_3|} = \frac{1}{8} = 0.125
\]
Example: Music Recommendation

(P1, P2) are more similar than (P1, P3) and (P2, P3)

Suggestions:
F -> P1
C -> P2
Millions of users that listen thousands different songs each
Millions of users: 

**Users** are the **objects** to compare – $O(n^2)$

Inserting a **new** user = compare 1 to all

Thousands songs:

Each **song** is a **dimension*** to compare

Curse of dimension

*See extra slides for other dimension examples
Does it solve efficiently similarity search with more than 10 dimensions?
Locality Sensitive Hashing

Locality Sensitive Hashing

LSH is an efficient algorithm to find similar objects using hashes.
Locality Sensitive Hashing

Cluster similar objects into hash buckets
(with a similarity threshold)
Locality Sensitive Hashing

Crypto hashes:
Similar objects -> very different hashes

Locality Sensitive Hashing:
Similar objects -> similar hashes
Locality Sensitive Hashing

Calculate the distance between objects within the same bucket only
Locality Sensitive Hashing

Queries (search similarity):
- Nearest neighbor ★
- Near neighbors ★★★★
- Clustering 🌈

Nearest neighbor
Near neighbors
Clustering
Locality Sensitive Hashing

We need to calculate the distance within a bucket to validate the distances
Locality Sensitive Hashing

False positives and false negatives may happen (configurable)

- False positives: Computing overhead
- False negative: Reduces recall
Locality Sensitive Hashing

Insert: hash dimensions of new object + compare within bucket
Locality Sensitive Hashing

\[ \text{LSH} = \text{MinHash} + \text{MultiMap}[ ] \]
Hashing is crucial!
For each distance there is a different hash family*
Jaccard Distance -> MinHash*

* See extra slides for other hash families
MinHash

• An array with the minimal hashes from all dimensions for each hash function

\[
\text{minHash(P1)}
\]

\[
23 \quad 12 \quad 5 \quad 45 \quad 7 \quad 34 \quad 12 \quad 76 \quad 87 \quad \ldots
\]

\[
\text{minHash(Object o)}:
\]

```
num_hashes <- 200  // defined based on a similarity threshold
hashes <- new hash[num_hashes]  // 200 different hash functions
minHash <- new int[num_hashes]  // MinHash of the object
for i in 0..hashes:  // for each hash function
    for d in object.dimensions:  // for each dimension (music)
        hi <- hashes[i](d)  // calculate the hash of d
        minHash[i] <- min(minHash[i], hi)  // store the min
```

• Converts \textbf{variable} number of dimensions to a \textbf{fixed} configurable number
• Using the \textbf{same order} of hash functions is important to \textbf{find similar} objects
Locality Sensitive Hashing

break it into b bands and r rows (based also on the desired similarity threshold)

Each band of r rows is the key for a different hashtable

```
minHash(P1)
23 12 5 45 7 34 12 76 87 ...
```

```
23 12 5 45 7 34 12 76 87 ...
```

```
hashTable[0]
23 12 5 [P1]
```

```
hashTable[1]
45 7 34 [P1]
```

```
hashTable[2]
12 76 87 [P1]
```
Locality Sensitive Hashing

break it into \(b\) bands and \(r\) rows (based also on the desired similarity threshold)

Each band of \(r\) rows is the key for a different hashtable

\[
\begin{array}{cccccc}
23 & 12 & 5 & 73 & 22 & 15 \\
3 & 28 & 56 & \ldots
\end{array}
\]
Interfaces

distance(Object o1, Object o2)

insert(Object o)

query(Object o)

   nearestNeighbor(Object o)

   nearNeighbors(Object o, int maxNeighbors)

   clustering(Object o)
Challenges: Implementing

- Generic to any object
- Providing multiple hash function families (generic to all distances)
- Being efficient (space and time)
- Durability
Challenges: Scaling up

- MultiMaps (1:n)
- Off-heap implementation (avoid garbage collection)
- Bigger than memory (e.g., using RAM + SSD disk space)
- Multi-threaded (fine-grain locks or non-blocking)
- Using primitives (avoid space overhead)
Challenges: Scaling out

- Distributing hash tables in several machines
  
  \[
  \begin{align*}
  \text{hashTable}[0] & \rightarrow s1 \\
  \text{hashTable}[1] & \rightarrow s2 \\
  \text{hashTable}[2] & \rightarrow s3 \\
  \text{hashTable}[3] & \rightarrow s4 \\
  \text{hashTable}[4] & \rightarrow s5 \\
  \end{align*}
  \]

- Partitioning keys (require to inform hashTable number)
  
  \[
  \begin{align*}
  \text{Keys } [0 \rightarrow 1,000,000] & \rightarrow s1 (\text{hashTable}[0-4]) \\
  \text{Keys } [1,000,000 \rightarrow 2,000,000] & \rightarrow s2 (\text{hashTable}[0-4]) \\
  \text{Keys } [2,000,000 \rightarrow 3,000,000] & \rightarrow s3 (\text{hashTable}[0-4]) \\
  \text{Keys } [3,000,000 \rightarrow 4,000,000] & \rightarrow s4 (\text{hashTable}[0-4]) \\
  \end{align*}
  \]
Available LSH implementations

- OpenLSH (https://github.com/singhj/locality-sensitive-hashing)
- Datasketch (https://github.com/ekzhu/datasketch)
- TarsosLSH (https://github.com/JorenSix/TarsosLSH)
- E2LSH (https://github.com/JorenSix/TarsosLSH)
- Many others
Some LSH papers

- Similarity Search in High Dimensions via Hashing
- Locality-Preserving Hashing in Multidimensional Spaces
- Approximate Nearest Neighbors: Towards Removing the Curse of dimensionality
- Near-Optimal Hashing Algorithms for Approximate Nearest Neighbor in High Dimensions
- Fast Search in Hamming Space with Multi-Index Hashing
- b-Bit Minwise Hashing
- LSH forest: self-tuning indexes for similarity search
Who uses LSH for what?

Detect near-duplicate web pages
Detecting Near-Duplicates for Web Crawling
Google News recommendations
Google News Personalization: Scalable Online Collaborative Filtering

Detect very similar routes
https://spark-summit.org/2016/events/locality-sensitive-hashing-by-spark/

Detect spam and malicious messages for event organizers

Clustering People

Spotify recommender system
LSH forest - ANNOY

And others
Take outs

- LSH solves *similarity search*
- LSH is very *useful* for several applications
- *Similarity search* is usually a *step* to something bigger
- Think *what do with* the similarity *knowledge*
Take outs

- Implementing basic specific cases is simple
- Being generic is not
- Scaling requires good engineering and optimizations
- Take time to experiment the best parameters to your case
Thank you!

Navtalks – November 18, 2016
# Other Objects and Dimensions

| **1 Dimension** | Binary values: 0 or 1  
Numbers: age, height, weight, etc. |
|-----------------|---------------------------------------------------------------------------------|
| **2 Dimensions** | Cartesian coordinates: (x, y)  
Tuples: (k, v) |
| **3 Dimensions** | 3D coordinates: (x, y, z)  
Animation: (time, x, y) |
| **N Dimensions** | Characters in a string: “abcdefgh”  
Substrings of a string: “abc”, “bcd”, “cde”…  
Bits in a Byte array: 0011 1101  
Words in a sentence: “Foo bar bar foo”  
Sentences in a document  
Pixels in an image  
Notes in a music  
Properties in an object  
Columns in a DB row  
*Minutiae* of fingerprints |
## Distances and LSH families

<table>
<thead>
<tr>
<th>Distance</th>
<th>Description</th>
<th>LSH family</th>
</tr>
</thead>
<tbody>
<tr>
<td>Euclidean</td>
<td>Distance between two vectors</td>
<td>Random projections</td>
</tr>
<tr>
<td>Jaccard</td>
<td>(\text{len}(\text{intersection})/\text{len}(\text{union}))</td>
<td>MinHash</td>
</tr>
<tr>
<td>Cosine</td>
<td>Angular distance between vectors</td>
<td>SimHash</td>
</tr>
<tr>
<td>Hamming</td>
<td>Number of Substitutions</td>
<td>BitSampling</td>
</tr>
<tr>
<td>Levenshtein</td>
<td>Minimal number of substitutions, insertions and deletions</td>
<td></td>
</tr>
</tbody>
</table>
Links to used resources

- **Presentations:**
  - http://www.slideshare.net/j_singh/open-lsh-a-framework-for-locality-sensitive-hashing-45912645
  - http://www.slideshare.net/SameeraHorawalavithana/locality-sensitive-hashing (Tree on taxonomy)
  - http://www.slideshare.net/DmitriySelivanov/finding-similar-items-in-high-dimensional-spaces-locality-sensitive-hashing
  - http://www.slideshare.net/jsuchal/minhashing-fast-similarity-search
  - http://www.slideshare.net/SparkSummit/locality-sensitive-hashing-by-spark (Uber on similar routes)
  - http://www.slideshare.net/huitseeker/a-gentle-introduction-to-locality-sensitive-hashing-with-apache-spark

- **Blog posts:**

- **Videos:**
  - https://www.youtube.com/watch?v=dgH0NP8Qxa8
  - https://www.youtube.com/watch?v=bQAYY8INBxg
  - https://www.youtube.com/watch?v=Arni-zkqMBA
  - https://www.youtube.com/watch?v=t_8SpFVOl7A
  - https://www.youtube.com/watch?v=Lqcwaw2YE_c
  - https://www.youtube.com/watch?v=Ha7_Vf2eZvQ
  - https://www.youtube.com/watch?v=Dkomk2wPaoc