Online Generation of Locality Sensitive Hash Signatures

BENJAMIN VAN DURME & ASHWIN LALL

human language technology center of excellence

JOHNS HOPKINS UNIVERSITY

Georgia Tech
Our access to data is growing fast
Our access to data is growing faster than our ability to process it.
Our access to data is growing faster than our ability to process it

Complementary solutions:
- Distributed environments (e.g., MapReduce)
- Streaming / Randomized Algorithms
Locality Sensitive Hashing

- Goal: fast comparison between points in very high dimensional space

- Indyk & Motwani (‘98) => Charikar (’02)
  - Randomly project points to low dimensional bit signatures such that cosine distance is approximately preserved

- Example Applications in HLT
  - Noun clustering [Ravichandran et al ’05]
  - Topic Detection and Tracking (TDT) [Petrovic et al ‘10]
Locality Sensitive Hashing
Locality Sensitive Hashing
Locality Sensitive Hashing
Locality Sensitive Hashing
Locality Sensitive Hashing

Benjamin Van Durme & Ashwin Lall

ACL 2010
Locality Sensitive Hashing

Benjamin Van Durme & Ashwin Lall

ACL 2010
Locality Sensitive Hashing

Benjamin Van Durme & Ashwin Lall

ACL 2010
Locality Sensitive Hashing
Locality Sensitive Hashing

Benjamin Van Durme & Ashwin Lall
ACL 2010
Locality Sensitive Hashing

Hamming Distance := $h = 1$
Signature Length := $b = 6$
Locality Sensitive Hashing

Hamming Distance := $h = 1$
Signature Length := $b = 6$

$\cos(\theta) \approx \cos\left(\frac{h}{b} \pi\right)$

$= \cos\left(\frac{1}{6} \pi\right)$
Accuracy as function of bit length

32 bit signatures

256 bit signatures

Approximate Cosine

True Cosine

Cheap

Accurate

Benjamin Van Durme & Ashwin Lall

ACL 2010
1. Online hash function

2. Pooling trick
Online Hash Function

\[ \vec{u} \in \mathbb{R}^d \]

\[ \vec{r}_i \sim N(0, 1)^d \]
Online Hash Function

\[ \vec{v} \in \mathbb{R}^d \]

\[ \vec{r}_i \sim \mathcal{N}(0, 1)^d \]

\[ h_i(\vec{v}) = \begin{cases} 
1 & \text{if } \vec{v} \cdot \vec{r}_i \geq 0, \\
0 & \text{otherwise}. 
\end{cases} \]
Online Hash Function

if $\vec{v} = \sum_j \vec{v}_j$
then $\vec{v} \cdot \vec{r}_i = \sum_j \vec{v}_j \cdot \vec{r}_i$

Break into local products
Online Hash Function

$$h_i(\vec{v}) = \begin{cases} 1 & \text{if } \vec{v} \cdot \vec{r}_i \geq 0, \\ 0 & \text{otherwise.} \end{cases}$$

$$h_{it}(\vec{v}) = \begin{cases} 1 & \text{if } \sum_j \vec{v}_j \cdot \vec{r}_i \geq 0, \\ 0 & \text{otherwise.} \end{cases}$$
The Pooling Trick

\[ \vec{r}_i \sim N(0, 1)^d \]

\[
\begin{pmatrix}
\vec{r}_1 \\
\vdots \\
\vec{r}_b
\end{pmatrix}
= 
\begin{pmatrix}
N(0, 1) & \cdots & N(0, 1) \\
\vdots & \ddots & \vdots \\
N(0, 1) & \cdots & N(0, 1)
\end{pmatrix}
\]
The random projection matrix can easily require gigabytes of memory

\[
\begin{pmatrix}
\vec{r}_1 \\
\vdots \\
\vec{r}_b
\end{pmatrix}
= \begin{pmatrix}
N(0, 1) & \cdots & N(0, 1) \\
\cdots & \cdots & \cdots \\
N(0, 1) & \cdots & N(0, 1)
\end{pmatrix}
\]
The Pooling Trick

\[ \tilde{p} \sim N(0, 1)^m \]

\[ \tilde{p} = ( N(0, 1) \ldots N(0, 1) ) \]

Define \( \tilde{r}_i[j] = \tilde{p}[\text{hash}(i, j) \text{ MOD } m] \)

\[ m \ll b \times d \]
The Pooling Trick

Define $\vec{r}_i[j] = \vec{p}[\text{hash}(i, j) \text{ MOD } m]$

$$
\begin{pmatrix}
\ldots & \ldots & \ldots \\
\ldots & (i, j) & \ldots \\
\ldots & \ldots & \ldots \\
\end{pmatrix}

\rightarrow

( N(0, 1) \ldots N(0, 1) )$$
The Pooling Trick

Pool Size

Mean Absolute Error

Mean Absolute Error

Pool Size

10

10^1

10^2

10^3

10^4

10^5

Benjamin Van Durme & Ashwin Lall

ACL 2010
The Pooling Trick

That is 400 kilobytes
Example

London
Milan_{97}, Madrid_{96}, Stockholm_{96}, Manila_{95}, Moscow_{95}
ASHER_0, Champaign_0, MANS_0, NOBLE_0, come_0
Prague_1, Vienna_1, suburban_1, synchronism_1, Copenhagen_2
Frankfurt_4, Prague_4, Taszar_5, Brussels_6, Copenhagen_6
Prague_12, Stockholm_12, Frankfurt_14, Madrid_14, Manila_14
Stockholm_20, Milan_22, Madrid_24, Taipei_24, Frankfurt_25
Closest based on true cosine

London

**Milan.97, Madrid.96, Stockholm.96, Manila.95, Moscow.95**

ASHER₀, Champaign₀, MANS₀, NOBLE₀, come₀

Prague₁, Vienna₁, suburban₁, synchronism₁, Copenhagen₂

Frankfurt₄, Prague₄, Taszar₅, Brussels₆, Copenhagen₆

Prague₁₂, Stockholm₁₂, Frankfurt₁₄, Madrid₁₄, Manila₁₄

Stockholm₂₀, Milan₂₂, Madrid₂₄, Taipei₂₄, Frankfurt₂₅
London
Milan.97, Madrid.96, Stockholm.96, Manila.95, Moscow.95
ASHER₀, Champaign₀, MANS₀, NOBLE₀, come₀
Prague₁, Vienna₁, suburban₁, synchronism₁, Copenhagen₂
Frankfurt₄, Prague₄, Taszar₅, Brussels₆, Copenhagen₆
Prague₁₂, Stockholm₁₂, Frankfurt₁₄, Madrid₁₄, Manila₁₄
Stockholm₂₀, Milan₂₂, Madrid₂₄, Taipei₂₄, Frankfurt₂₅

Closest based on 32 bit sig.’s

Cheap
London
Milan.97, Madrid.96, Stockholm.96, Manila.95, Moscow.95
ASHER0, Champaign0, MANS0, NOBLE0, come0
Prague1, Vienna1, suburban1, synchronism1, Copenhagen2
Frankfurt4, Prague4, Taszar5, Brussels6, Copenhagen6
Prague12, Stockholm12, Frankfurt14, Madrid14, Manila14
Stockholm20, Milan22, Madrid24, Taipei24, Frankfurt25

Closest based on 256 bit sig.’s

Cheap-ish

Benjamin Van Durme & Ashwin Lall

ACL 2010
No Questions.

See Poster.

Also: these algorithms distribute to the cloud.