Data-Intensive Distributed Computing

CS 431/631 451/651 (Winter 2019)

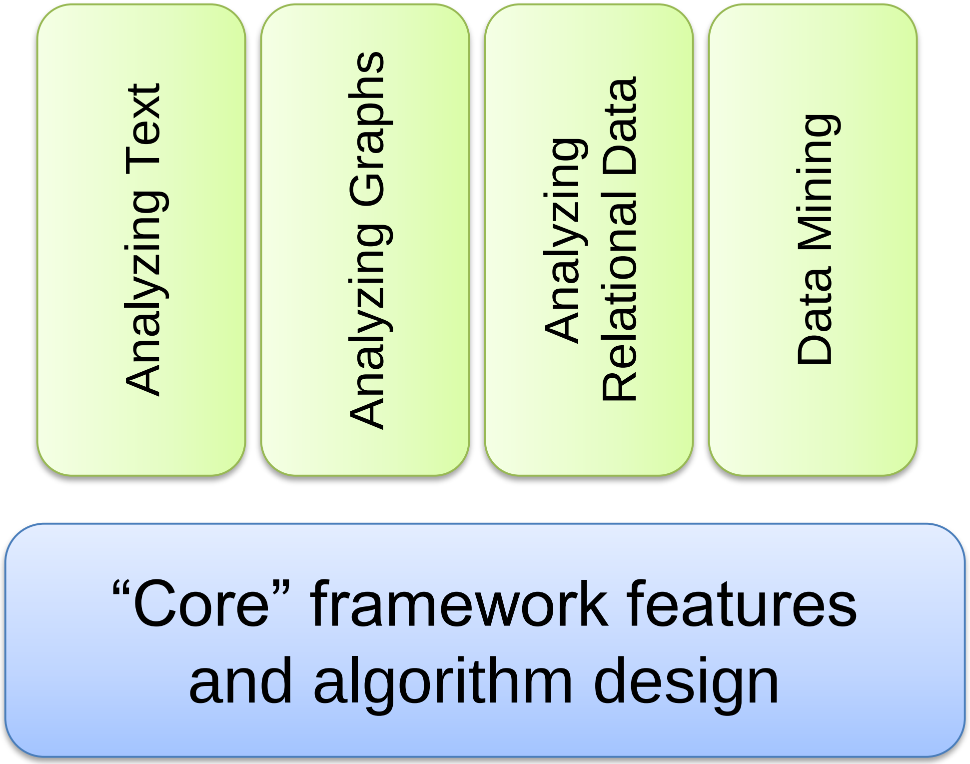
Part 6: Data Mining (3/4)

March 7, 2019

Adam Roegiest
Kira Systems

These slides are available at http://roegiest.com/bigdata-2019w/

Structure of the Course



Theme: Similarity

How similar are two items? How "close" are two items?

Equivalent formulations: large distance = low similarity

Lots of applications!

Problem: find similar items

Offline variant: extract all similar pairs of objects from a large collection

Online variant: is this object similar to something I've seen before?

Today!

Problem: arrange similar items into clusters

Offline variant: entire static collection available at once

Online variant: objects incrementally available

Next time!

Literature Note

Many communities have tackled similar problems:

Theoretical computer science

Information retrieval

Data mining

Databases

…

Issues

Slightly different terminology

Results not easy to compare

Four Steps

Specify distance metric

Jaccard, Euclidean, cosine, etc.

Compute representation

Shingling, tf.idf, etc.

“Project”

Minhash, random projections, etc.

Extract

Bucketing, sliding windows, etc.

Distance Metrics

Source: www.flickr.com/photos/thiagoalmeida/250190676/

Distance Metrics

1. Non-negativity:

$$\mathrm{d}(\mathrm{x},\mathrm{y}) \geq 0$$

2. Identity:

$$\mathrm{d}(\mathrm{x},\mathrm{y}) = 0 \iff \mathrm{x} = \mathrm{y}$$

3. Symmetry:

$$\mathrm{d}(\mathrm{x},\mathrm{y}) = \mathrm{d}(\mathrm{y},\mathrm{x})$$

4. Triangle Inequality

$$\mathrm{d}(\mathrm{x},\mathrm{y}) \leq \mathrm{d}(\mathrm{x},\mathrm{z}) + \mathrm{d}(\mathrm{z},\mathrm{y})$$

Distance: Jaccard

Given two sets A, B

Jaccard similarity:

$$\mathrm{J}(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

$$\mathrm{d}(A, B) = 1 - \mathrm{J}(A, B)$$

Distance: Norms

Given $$\mathrm{x} = [x_1, x_2, \ldots x_n]$$ $$\mathrm{y} = [y_1, y_2, \ldots y_n]$$

Euclidean distance (L_2-norm) $$\mathrm{d}(\mathrm{x}, \mathrm{y}) = \sqrt{\sum_{i=0}^{n} (x_i - y_i)^2}$$

Manhattan distance (L_1-norm) $$\mathrm{d}(\mathrm{x}, \mathrm{y}) = \sum_{i=0}^{n} |x_i - y_i|$$

L_r-norm $$\mathrm{d}(\mathrm{x}, \mathrm{y}) = \left[\sum_{i=0}^{n} |x_i - y_i|^r\right]^{1/r}$$

Distance: Cosine

Given $$\mathrm{x} = [x_1, x_2, \ldots x_n]$$
$$\mathrm{y} = [y_1, y_2, \ldots y_n]$$

Idea: measure distance between the vectors

$$\cos\theta = \frac{\mathrm{x} \cdot \mathrm{y}}{|\mathrm{x}||\mathrm{y}|}$$

Thus:

$$\mathrm{sim}(\mathrm{x}, \mathrm{y}) = \frac{\sum_{i=0}^{n} x_i y_i}{\sqrt{\sum_{i=0}^{n} x_i^2}\sqrt{\sum_{i=0}^{n} y_i^2}}$$

$$\mathrm{d}(\mathrm{x}, \mathrm{y}) = 1 - \mathrm{sim}(\mathrm{x}, \mathrm{y})$$

Distance: Hamming

Given two bit vectors

Hamming distance: number of elements which differ


Representations

Representations

(Text)

Unigrams (i.e., words)

Shingles = *n*-grams

At the word level

At the character level

Feature weights

boolean

tf.idf

BM25

…

Representations

(Beyond Text)

For recommender systems:

Items as features for users

Users as features for items

For graphs:

Adjacency lists as features for vertices

For log data:

Behaviors (clicks) as features

Minhash

Source: www.flickr.com/photos/rheinitz/6158837748/

Near-Duplicate Detection of Webpages

What's the source of the problem?

Mirror pages (legit)

Spam farms (non-legit)

Additional complications (e.g., nav bars)

Naïve algorithm:

Compute cryptographic hash for webpage (e.g., MD5)

Insert hash values into a big hash table

Compute hash for new webpage: collision implies duplicate

What's the issue?

Intuition:

Hash function needs to be tolerant of minor differences

High similarity implies higher probability of hash collision

Minhash

Naïve approach: N^2 comparisons: Can we do better?

Seminal algorithm for near-duplicate detection of webpages

Used by AltaVista

Setup:

Documents (HTML pages) represented by shingles (n-grams)

Jaccard similarity: dups are pairs with high similarity

Preliminaries: Representation

Sets:

A = {e_1, e_3, e_7}

B = {e_3, e_5, e_7}

Can be equivalently expressed as matrices:

Element	A	B
e_1	1	0
e_2	0	0
e_3	1	1
e_4	0	0
e_5	0	1
e_6	0	0
e_7	1	1

Preliminaries: Jaccard

Element	A	B
e_1	1	0
e_2	0	0
e_3	1	1
e_4	0	0
e_5	0	1
e_6	0	0
e_7	1	1

Let:

M_{00} = # rows where both elements are 0

M_{11} = # rows where both elements are 1

M_{01} = # rows where A=0, B=1

M_{10} = # rows where A=1, B=0

$$\mathrm{J}(A, B) = \frac{M_{11}}{M_{01} + M_{10} + M_{11}}$$

Minhash

Computing minhash

Start with the matrix representation of the set
Randomly permute the rows of the matrix
minhash is the first row with a "one"

Example

Element	A	B
e_1	1	0
e_2	0	0
e_3	1	1
e_4	0	0
e_5	0	1
e_6	0	0
e_7	1	1

$h(A) = e_3$ $h(B) = e_5$

Element	A	B
e_6	0	0
e_2	0	0
e_5	0	1
e_3	1	1
e_7	1	1
e_4	0	0
e_1	1	0

Minhash and Jaccard

Element	A	B	
e_6	0	0	M_{00}
e_2	0	0	M_{00}
e_5	0	1	M_{01}
e_3	1	1	M_{11}
e_7	1	1	M_{11}
e_4	0	0	M_{00}
e_1	1	0	M_{10}

$$P[h(A) = h(B)] = \mathrm{J}(A, B)$$

$$\frac{M_{11}}{M_{01} + M_{10} + M_{11}} \qquad \frac{M_{11}}{M_{01} + M_{10} + M_{11}}$$

Woah!

To Permute or Not to Permute?

Problem: Permutations are expensive

Solution: Interpret the hash value as the permutation

Only need to keep track of the minimum hash value

Can keep track of multiple minhash values at once

Extracting Similar Pairs

Task: discover all pairs with similarity greater than *s*

Naïve approach: N^2 comparisons: Can we do better?

Tradeoffs:

False positives: discovered pairs that have similarity less than *s*

False negatives: pairs with similarity greater than s not discovered

The errors (and costs) are asymmetric!

Extracting Similar Pairs (LSH)

We know: $P[h(A) = h(B)] = \mathrm{J}(A, B)$

Task: discover all pairs with similarity greater than *s*

Algorithm:

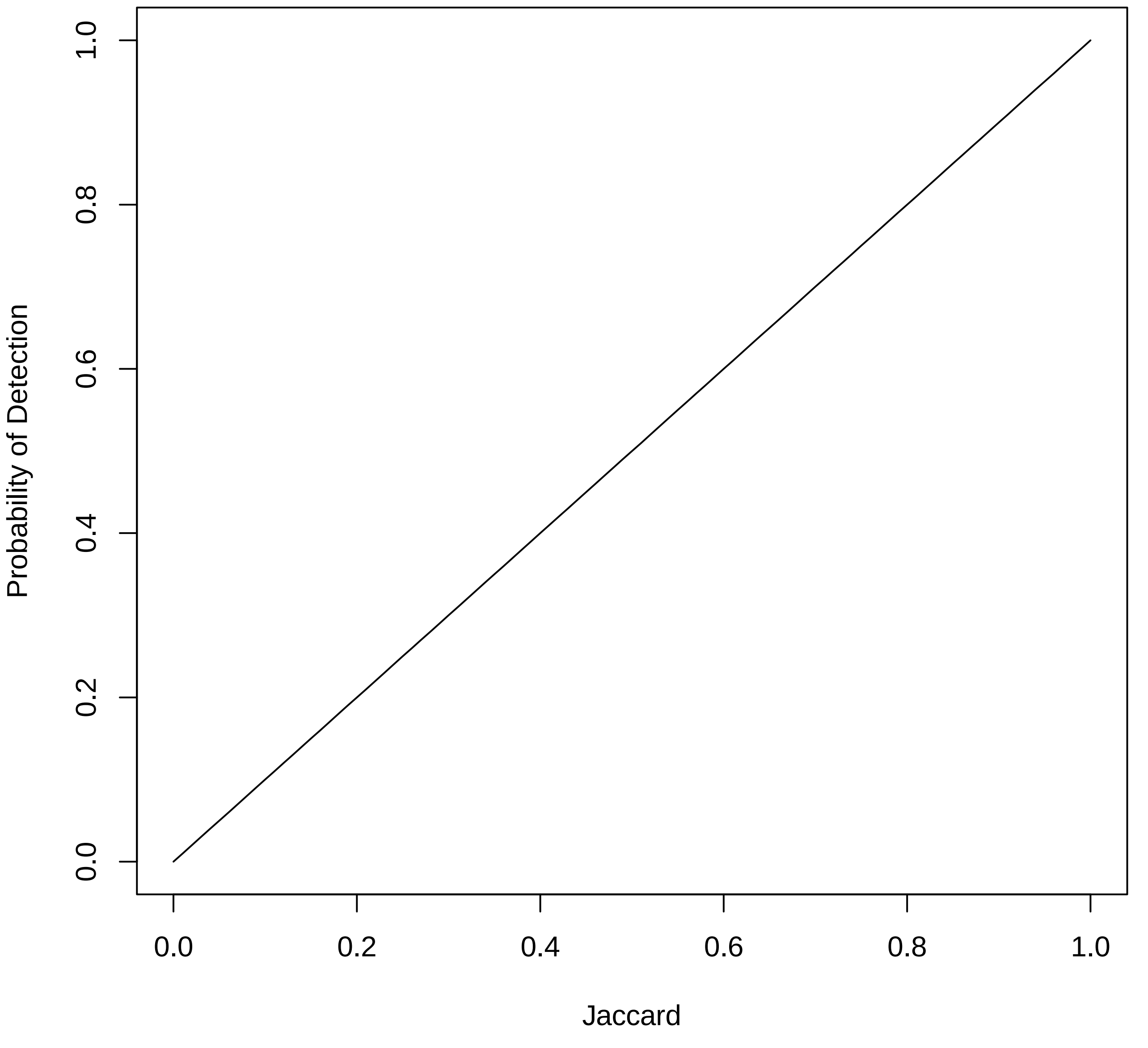
For each object, compute its minhash value

Group objects by their hash values

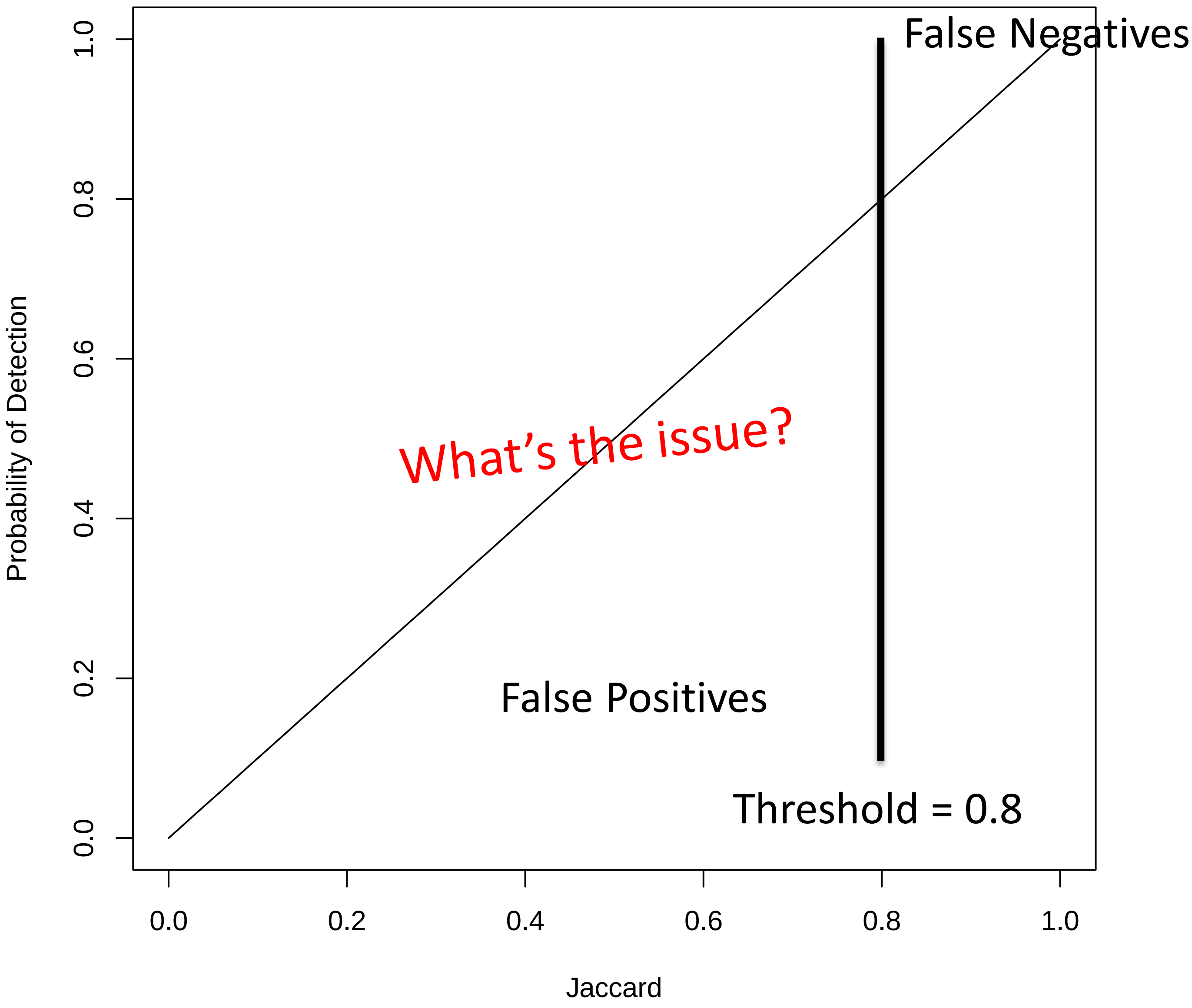
Output all pairs within each group

Analysis:

If J(A,B) = *s*, then probability we detect it is *s*



Probability of Detection
0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0
Jaccard



False Negatives
What's the issue?
False Positives
Threshold = 0.8
Probability of Detection
Jaccard
0.0
0.2
0.4
0.6
0.8
1.0

2 Minhash Signatures

We know: $P[h(A) = h(B)] = \mathrm{J}(A, B)$

Task: discover all pairs with similarity greater than *s*

Algorithm:

For each object, compute 2 minhash values and concatenate = signature

Group objects by their signatures

Output all pairs within each group

Analysis:

If J(A,B) = *s*, then probability we detect it is s^2

3 Minhash Signatures

We know: $P[h(A) = h(B)] = \mathrm{J}(A, B)$

Task: discover all pairs with similarity greater than *s*

Algorithm:

For each object, compute 3 minhash values and concatenate = signature

Group objects by their signatures

Output all pairs within each group

Analysis:

If J(A,B) = *s*, then probability we detect it is s^3

k Minhash Signatures

We know: $P[h(A) = h(B)] = \mathrm{J}(A, B)$

Task: discover all pairs with similarity greater than *s*

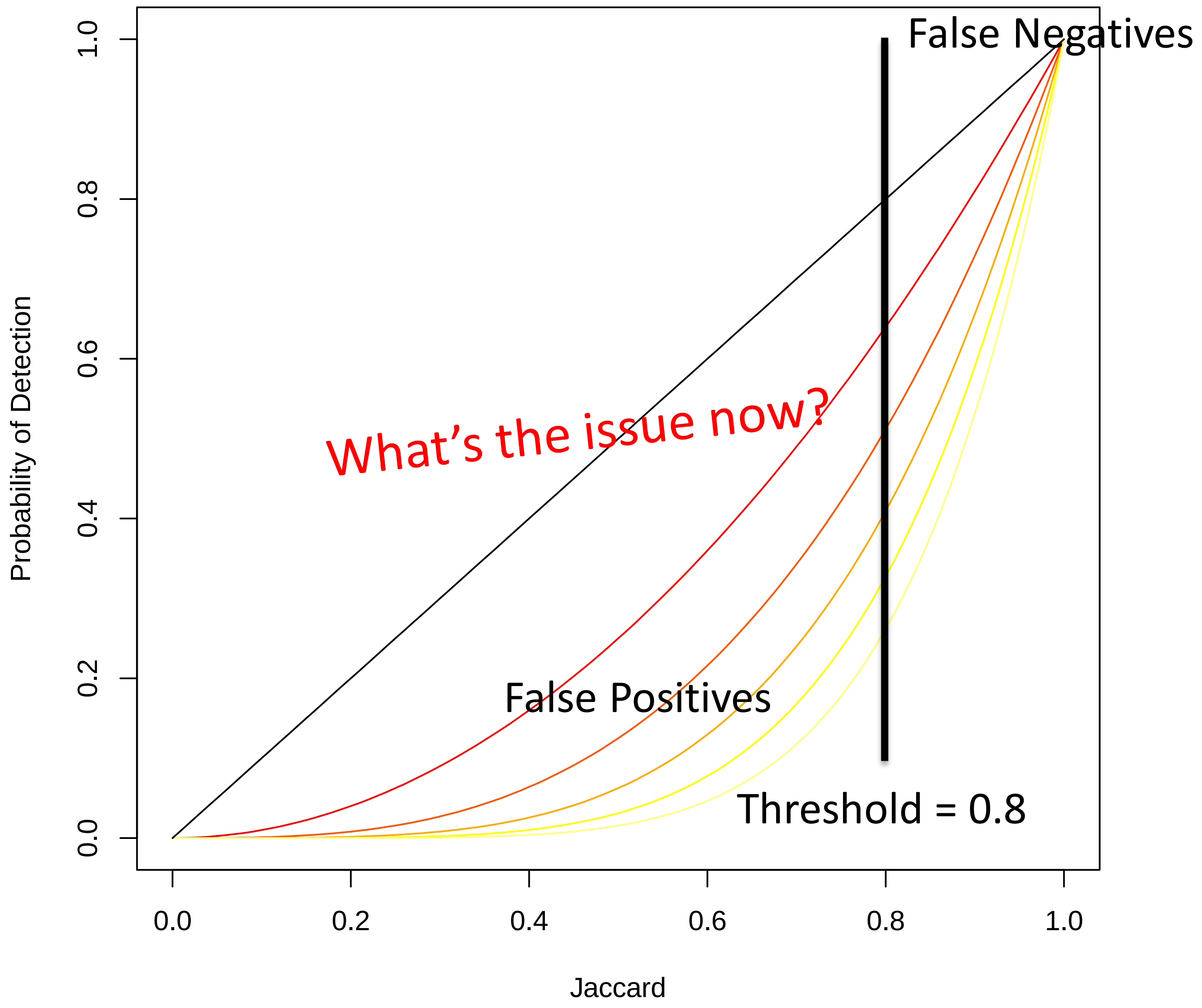
Algorithm:

For each object, compute *k* minhash values and concatenate = signature
Group objects by their signatures
Output all pairs within each group

Analysis:

If J(A,B) = *s*, then probability we detect it is s^k

k Minhash Signatures concatenated together



n different k Minhash Signatures

We know: $P[h(A) = h(B)] = \mathrm{J}(A, B)$

Task: discover all pairs with similarity greater than s

Algorithm:

For each object, compute n sets k minhash values

For each set, concatenate k minhash values together

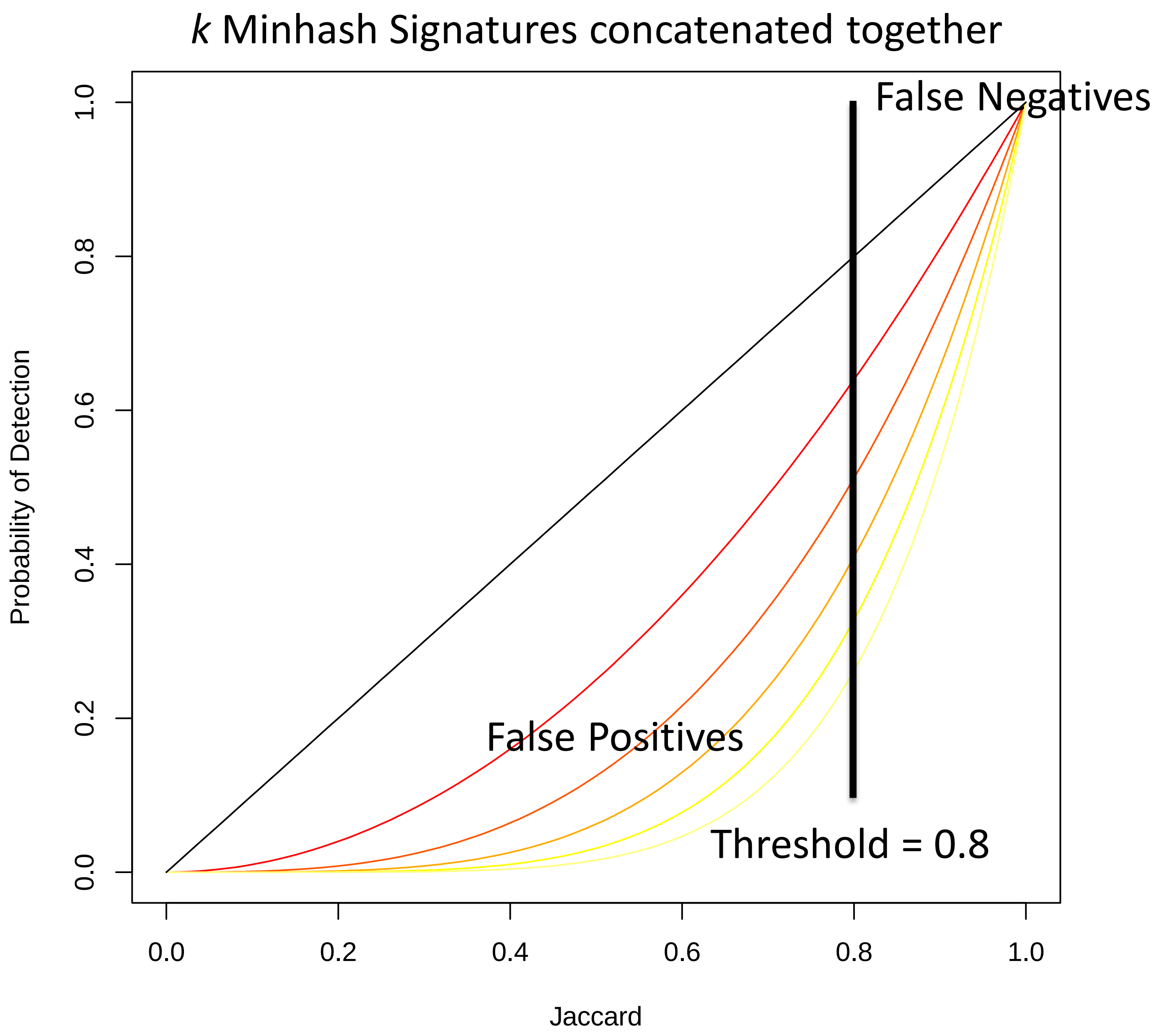
In each set: group objects by signatures, output all pairs in each group

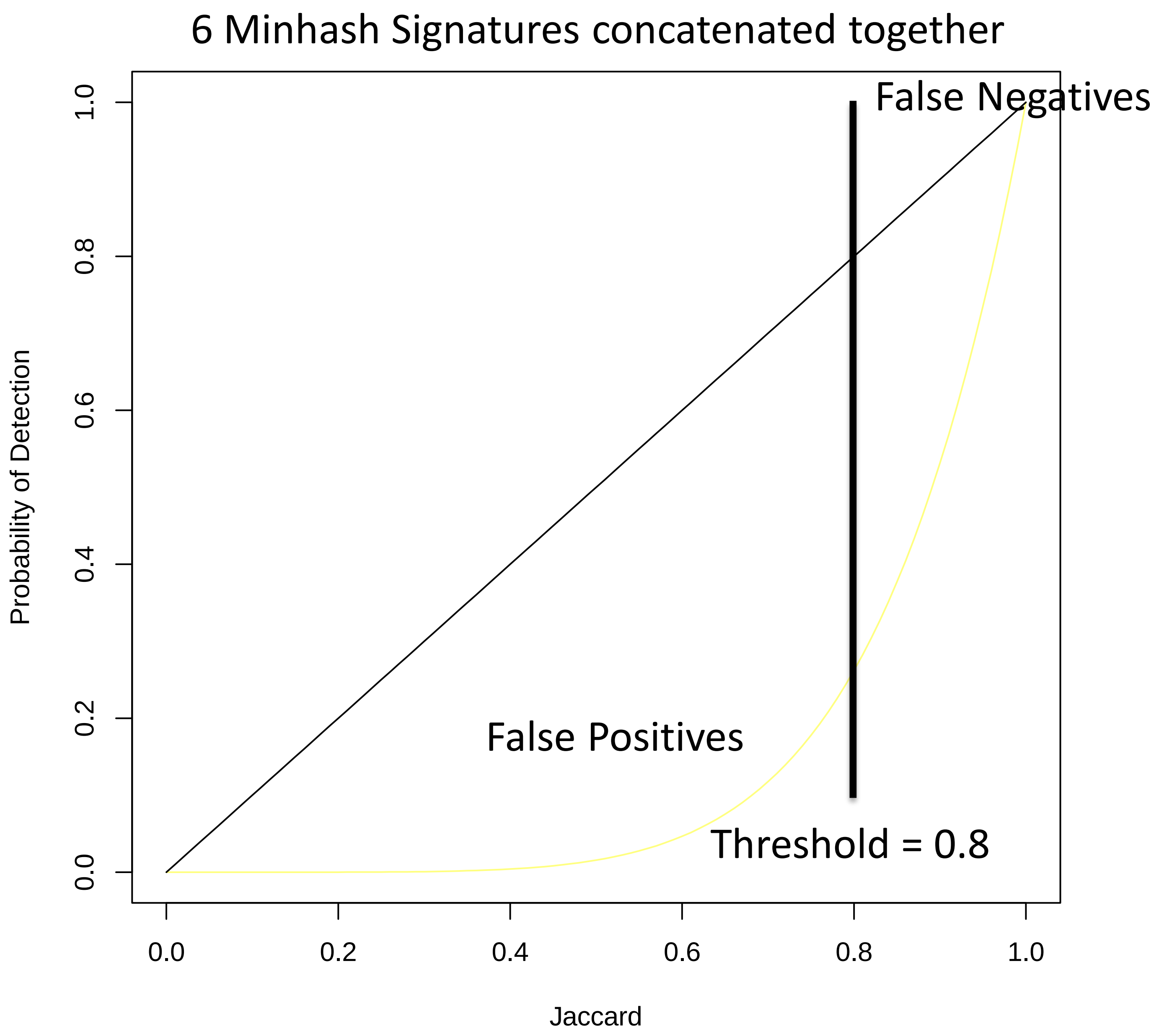
De-dup pairs

Analysis:

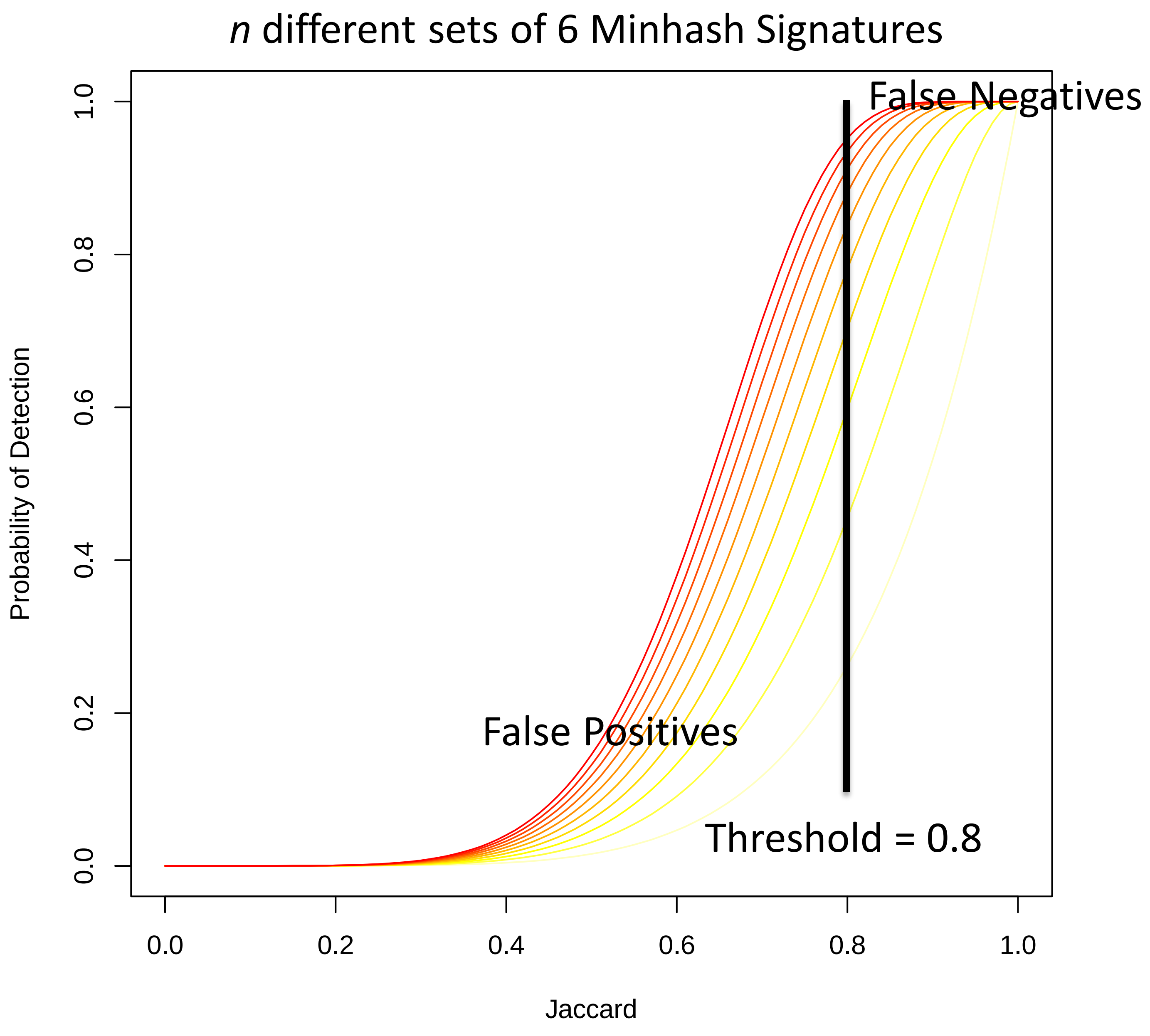
If J(A,B) = s, P(none of the n collide) = $(1 - s^k)^n$

If J(A,B) = s, then probability we detect it is $1 - (1 - s^k)^n$


k Minhash Signatures concatenated together
False Negatives
False Positives
Threshold = 0.8
Probability of Detection
Jaccard
0.0
0.2
0.4
0.6
0.8
1.0



6 Minhash Signatures concatenated together
False Negatives
False Positives
Threshold = 0.8
Probability of Detection
Jaccard
0.0
0.2
0.4
0.6
0.8
1.0



n different sets of 6 Minhash Signatures
False Negatives
False Positives
Threshold = 0.8
Probability of Detection
Jaccard
0.0
0.2
0.4
0.6
0.8
1.0

n different *k* Minhash Signatures

Example: J(A,B) = 0.8, 10 sets of 6 minhash signatures

P(*k* minhash signatures match) = $(0.8)^6 = 0.262$

P(*k* minhash signature doesn't match in any of the 10 sets) = $(1 – (0.8)^6)^{10} = 0.0478$

Thus, we should find $1 – (1 – (0.8)^6)^{10} = 0.952$ of all similar pairs

Example: J(A,B) = 0.4, 10 sets of 6 minhash signatures

P(*k* minhash signatures match) = $(0.4)^6 = 0.0041$

P(*k* minhash signature doesn't match in any of the 10 sets) = $(1 – (0.4)^6)^{10} = 0.9598$

Thus, we should find $1 – (1 – 0.262144)^{10} = 0.040$ of all similar pairs

n different *k* Minhash Signatures

s	$1 - (1 - s^6)^{10}$
0.2	0.0006
0.3	0.0073
0.4	0.040
0.5	0.146
0.6	0.380
0.7	0.714
0.8	0.952
0.9	0.999

What's the issue?

Practical Notes

Common implementation:

Generate M minhash values, select k of them n times

Reduces amount of hash computations needed

Determining "authoritative" version is non-trivial

MapReduce/Spark Implementation

Map over objects:

Generate *M* minhash values, select *k* of them *n* times

Each draw yields a signature, emit:

key = (*p*, signature), where *p* = [*1* … *n*] and value = object id

Shuffle/Sort

Reduce

Receive all object ids with same (*n*, signature), emit clusters

Second pass to de-dup and group clusters

(Optional) Third pass to eliminate false positives

Offline Extraction vs. Online Querying

Batch formulation of the problem:

Discover all pairs with similarity greater than *s*

Useful for post-hoc batch processing of web crawl

Online formulation of the problem:

Given new webpage, is it similar to one I've seen before?

Useful for incremental web crawl processing

Online Similarity Querying

Preparing the existing collection:

For each object, compute *n* sets of *k* minhash values

For each set, concatenate *k* minhash values together

Keep each signature in hash table (in memory)

Note: can parallelize across multiple machines

Querying and updating:

For new webpage, compute signatures and check for collisions

Collisions imply duplicate (determine which version to keep)

Update hash tables

Random Projections

Source: www.flickr.com/photos/roj/4179478228/

Limitations of Minhash

Minhash is great for near-duplicate detection

Set high threshold for Jaccard similarity

Limitations:

Jaccard similarity only

Set-based representation, no way to assign weights to features

Random projections:

Works with arbitrary vectors using cosine similarity

Same basic idea, but details differ

Slower but more accurate: no free lunch!

Random Projection Hashing

Generate a random vector *r* of unit length

Draw from univariate Gaussian for each component

Normalize length

Define:

$$h_r(\mathrm{u}) = \begin{cases} 1 & \text{if } \mathrm{r} \cdot \mathrm{u} \geq 0 \\ 0 & \text{if } \mathrm{r} \cdot \mathrm{u} < 0 \end{cases}$$

Physical intuition?

RP Hash Collisions

It can be shown that:

Proof in (Goemans and Williamson, 1995)

$$P[h_r(\mathrm{u}) = h_r(\mathrm{v})] = 1 - \frac{\theta(\mathrm{u},\mathrm{v})}{\pi}$$

Thus:

$$\cos(\theta(\mathrm{u},\mathrm{v})) = \cos((1 - P[h_r(\mathrm{u}) = h_r(\mathrm{v})])\pi)$$

Physical intuition?

Random Projection Signature

Given *D* random vectors:

$$[\mathrm{r}_1, \mathrm{r}_2, \mathrm{r}_3, \ldots \mathrm{r}_D]$$

Convert each object into a *D* bit signature:

$$\mathrm{u} \rightarrow [h_{r_1}(\mathrm{u}), h_{r_2}(\mathrm{u}), h_{r_3}(\mathrm{u}), \ldots h_{r_D}(\mathrm{u})]$$

Since:

$$\cos(\theta(\mathrm{u}, \mathrm{v})) = \cos((1 - P[h_r(\mathrm{u}) = h_r(\mathrm{v})])\pi)$$

We can derive:

$$\cos(\theta(\mathrm{u}, \mathrm{v})) = \cos\left(\frac{\mathrm{hamming}(\mathrm{s_u}, \mathrm{s_v})}{D} \cdot \pi\right)$$

Insight: similarity boils down to comparison of hamming distances between signatures

One-RP Signature

Task: discover all pairs with cosine similarity greater than *s*

Algorithm:

Compute *D*-bit RP signature for every object

Take first bit, bucket objects into two sets

Perform brute force pairwise (hamming distance) comparison in each bucket, retain those below hamming distance threshold

Analysis:

Probability we will discover all pairs: *

$$1 - \frac{\cos^{-1}(s)}{\pi}$$

Efficiency

$$N^2 \qquad \text{vs.} \qquad 2\left(\frac{N}{2}\right)^2$$

* Note, this is actually a simplification: see Ture et al. (SIGIR 2011) for details.

Two-RP Signature

Task: discover all pairs with cosine similarity greater than *s*

Algorithm:

Compute *D*-bit RP signature for every object
Take first two bits, bucket objects into four sets

Perform brute force pairwise (hamming distance) comparison
in each bucket, retain those below hamming distance threshold

Analysis:

Probability we will discover all pairs:

$$\left[1 - \frac{\cos^{-1}(s)}{\pi}\right]^2$$

Efficiency

$$N^2 \qquad \text{vs.} \qquad 4\left(\frac{N}{4}\right)^2$$

k-RP Signature

Task: discover all pairs with cosine similarity greater than s

Algorithm:

Compute D-bit RP signature for every object

Take first k bits, bucket objects into 2^k sets

Perform brute force pairwise (hamming distance) comparison in each bucket, retain those below hamming distance threshold

Analysis:

Probability we will discover all pairs:

$$\left[1 - \frac{\cos^{-1}(s)}{\pi}\right]^k$$

Efficiency

$$N^2 \qquad \text{vs.} \qquad 2^k\left(\frac{N}{2^k}\right)^2$$

m Sets of *k*-RP Signature

Task: discover all pairs with cosine similarity greater than *s*

Algorithm:

Compute *D*-bit RP signature for every object

Choose *m* sets of *k* bits; for each, use *k* selected bits to bucket objects into 2^k sets

Perform brute force pairwise (hamming distance) comparison in each bucket, retain those below hamming distance threshold

Analysis:

Probability we will discover all pairs:

$$1 - \left[1 - \left[1 - \frac{\cos^{-1}(s)}{\pi}\right]^k\right]^m$$

Efficiency

$$N^2 \qquad \text{vs.} \qquad m \cdot 2^k \left(\frac{N}{2^k}\right)^2$$

MapReduce/Spark Implementation

Map over objects:

Compute *D*-bit RP signature for every object

Choose *m* sets of *k* bits and use to bucket; for each, emit:

key = (*p*, *k* bits), where *p* = [*1* … *m*], value = (object id, rest of signature bits)

Shuffle/Sort

Reduce

Receive (*p*, *k* bits)

Perform brute force pairwise (hamming distance) comparison for each key, retain those below hamming distance threshold

Second pass to de-dup and group clusters

(Optional) Third pass to eliminate false positives

Online Querying

Preparing the existing collection:

Compute D-bit RP signature for every object

Choose m sets of k bits and use to bucket

Store signatures in memory (across multiple machines)

Querying:

Compute D-bit signature of query object, choose m sets of k bits in same way

Perform brute-force scan of correct bucket (in parallel)

Additional Issues to Consider

Emphasis on recall, not precision

Two sources of error:

From LSH

From using hamming distance as proxy for cosine similarity

Load imbalance

Parameter tuning

“Sliding Window” Algorithm

Compute *D*-bit RP signature for every object

For each object, permute bit signature *m* times

For each permutation, sort bit signatures

Apply sliding window of width *B* over sorted

Compute hamming distances of bit signatures within window

MapReduce/Spark Implementation

Mapper:

Compute *D*-bit RP signature for every object

Permute *m* times, for each emit:

key = (*p*, signature), where *p* = [*1* … *m*], value = object id

Shuffle/Sort

Reduce

Keep FIFO queue of *B* bit signatures

For each new bit signature, compute hamming distance wrt all in queue

Add new bit signature to end of queue, displacing oldest

Four Steps to Finding Similar Items

Specify distance metric

Jaccard, Euclidean, cosine, etc.

Compute representation

Shingling, tf.idf, etc.

“Project”

Minhash, random projections, etc.

Extract

Bucketing, sliding windows, etc.

Source: Wikipedia (Japanese rock garden)