Machine Learning for Data Mining

Finding Similar Items in High Dimensional Spaces

Andres Mendez-Vazquez

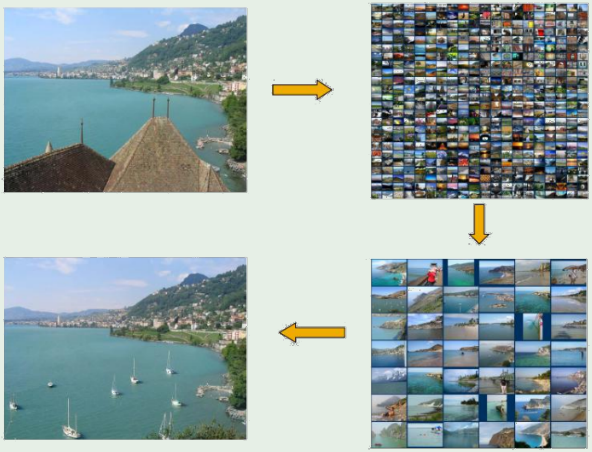
August 17, 2018

Outline

Outline

Scene Completion Problem

Example

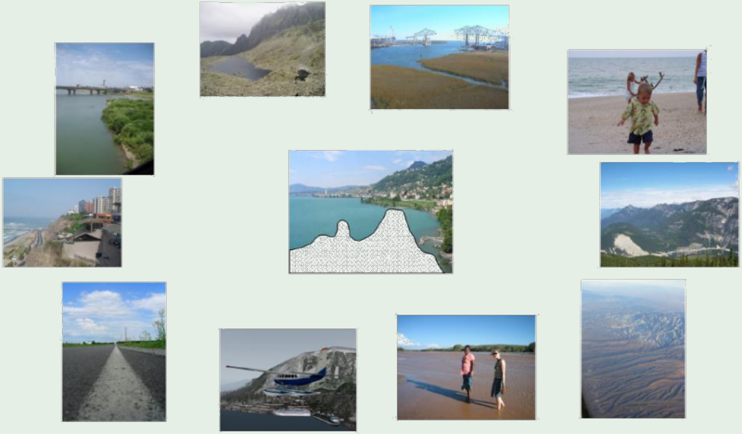


Scene Completion Problem

Example

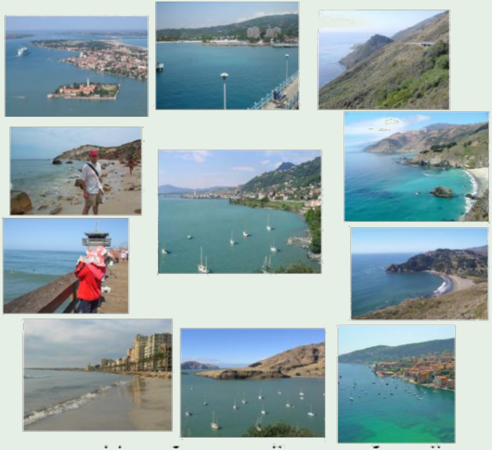
Scene Completion Problem

10 nearest neighbors from a collection of 20,000 images



Scene Completion Problem

10 nearest neighbors from a collection of 2 million images



Outline

A Common Idea

Problems

- Many problems can be expressed as finding "similar" sets:
  - Find near-neighbors in **high-dimensional** space

A Common Idea

Problems

- Many problems can be expressed as finding "similar" sets:
  - Find near-neighbors in **high-dimensional** space

Examples

- Pages with similar words
  - For duplicate detection, classification by topic
- Customers who purchased similar products
  - Products with similar customer sets
- Images with similar features
- Users who visited the similar websites

A Common Idea

Problems

- Many problems can be expressed as finding “similar” sets:
  - Find near-neighbors in **high-dimensional** space

Examples

- Pages with similar words
  - For duplicate detection, classification by topic
- Customers who purchased similar products
  - Products with similar customer sets
- Images with similar features
- Users who visited the similar websites

A Common Idea

Problems

- Many problems can be expressed as finding "similar" sets:
  - Find near-neighbors in **high-dimensional** space

Examples

- Pages with similar words
  - For duplicate detection, classification by topic
- Customers who purchased similar products
  - Products with similar customer sets
- Images with similar features
- Users who visited the similar websites

A Common Idea

Problems

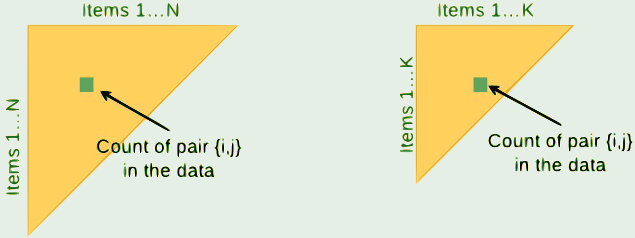
- Many problems can be expressed as finding "similar" sets:
  - Find near-neighbors in **high-dimensional** space

Examples

- Pages with similar words
  - For duplicate detection, classification by topic
- Customers who purchased similar products
  - Products with similar customer sets
- Images with similar features
- Users who visited the similar websites

Relation to Previous Lecture

Last time: Finding frequent pairs



We had the Naïve solution

Single pass but requires space quadratic in the number of items:

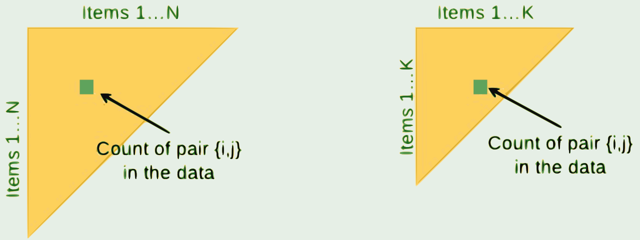
- N = number of distinct items
- K =number of items with support $\geq s$

However the A-priori Algorithm

- First pass: Find frequent singletons
  - For a pair to be a candidate for a frequent pair, its singletons have to be frequent!
- Second pass:
  - Count only candidate pairs!

Relation to Previous Lecture

Last time: Finding frequent pairs



We had the Naïve solution

Single pass but requires space quadratic in the number of items:

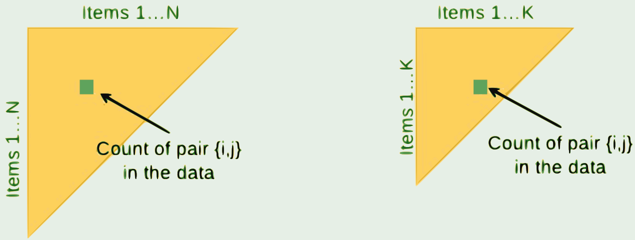
- N = number of distinct items
- K =number of items with support $\geq s$

However the A-priori Algorithm

- First pass: Find frequent singletons
  - For a pair to be a candidate for a frequent pair, its singletons have to be frequent!
- Second pass:
  - Count only candidate pairs!

Relation to Previous Lecture

Last time: Finding frequent pairs



We had the Naïve solution

Single pass but requires space quadratic in the number of items:

- $N =$ number of distinct items
- $K =$number of items with support $\geq s$

However the A-priori Algorithm

- First pass: Find frequent singletons
  - For a pair to be a candidate for a frequent pair, its singletons have to be frequent!
- Second pass:
  - Count only candidate pairs!

Relation to Previous Lecture

Last time

Finding frequent pairs.

Relation to Previous Lecture

Last time

Finding frequent pairs.

Further improvement using PCY

- Pass 1:
  - Count exact frequency of each item:

Items $1 \ldots N$

  - Take pairs of items $\{i, j\}$, hash them into B buckets and count of the number of pairs that hashed to each bucket:

Relation to Previous Lecture

Last time

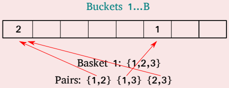
Finding frequent pairs.

Further improvement using PCY

- Pass 1:
  - Count exact frequency of each item:

Items $1 \ldots N$

  - Take pairs of items $\{i, j\}$, hash them into B buckets and count of the number of pairs that hashed to each bucket:

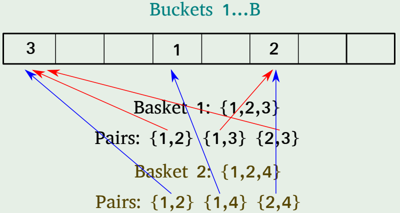


Relation to Previous Lecture

Further improvement: PCY

Pass 2:

- For a pair $\{i, j\}$ to be a candidate for a frequent pair, its singletons have to be frequent and it has to hash to a frequent bucket!



Thus, we have

Previous Lecture: A-Priori

- Main Idea: Candidates
  - Instead of keeping a count of each pair, only keep a count for candidate pairs!

Today's Lecture

- Main Idea: Candidates
  - Pass 1: Take documents and has them to buckets such that **documents that are similar hash to the same bucket.**
  - Pass 2: Only compare documents that are candidates (Hashed into the same bucket)
- Thus, we need $O(N)$ instead of $O(N^2)$.

Thus, we have

Previous Lecture: A-Priori

- Main Idea: Candidates
  - Instead of keeping a count of each pair, only keep a count for candidate pairs!

Today's Lectuire

- Main Idea: Candidates
  - Pass 1: Take documents and has them to buckets such that **documents that are similar hash to the same bucket.**
  - Pass 2: Only compare documents that are candidates (Hashed into the same bucket)
- Thus, we need $O(N)$ instead of $O(N^2)$.

Outline

Distance Measures

Goal

- Find near-neighbors in high-dim. space
  - We formally define "near neighbors" as points that are a "small distance" apart.

Application

- For each application, we first need to define what "distance" means

Distance Measures

Goal

- Find near-neighbors in high-dim. space
  - We formally define "near neighbors" as points that are a "small distance" apart.

Application

- For each application, we first need to define what "distance" means

Distance Measures

Today: Jaccard distance (/similarity)

- The **Jaccard Similarity/Distance** of two **sets** is the size of their intersection / the size of their union:
  - $sim(C_1, C_2) = |C_1 \bigcap C_2| / |C_1 \bigcup C_2|$
  - $d(C_1, C_2) = 1 - |C_1 \bigcap C_2| / |C_1 \bigcup C_2|$

Distance Measures

Today: Jaccard distance (/similarity)

- The **Jaccard Similarity/Distance** of two **sets** is the size of their intersection / the size of their union:
  - $sim(C_1, C_2) = |C_1 \bigcap C_2| / |C_1 \bigcup C_2|$
  - $d(C_1, C_2) = 1 - |C_1 \bigcap C_2| / |C_1 \bigcup C_2|$

Distance Measures

Today: Jaccard distance (/similarity)

- The **Jaccard Similarity/Distance** of two **sets** is the size of their intersection / the size of their union:
  - $sim(C_1, C_2) = |C_1 \bigcap C_2| / |C_1 \bigcup C_2|$
  - $d(C_1, C_2) = 1 - |C_1 \bigcap C_2| / |C_1 \bigcup C_2|$

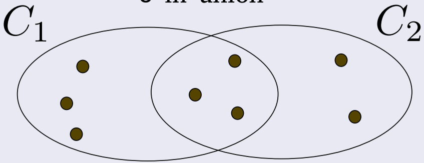
Distance Measures

Today: Jaccard distance (/similarity)

- The **Jaccard Similarity/Distance** of two **sets** is the size of their intersection / the size of their union:
  - $sim(C_1, C_2) = |C_1 \bigcap C_2| / |C_1 \bigcup C_2|$
  - $d(C_1, C_2) = 1 - |C_1 \bigcap C_2| / |C_1 \bigcup C_2|$

3 in intersection

8 in union



$$sim(C_1, C_2) = \frac{3}{8}$$

$$d(C_1, C_2) = \frac{5}{8}$$

Outline

Finding Similar Documents

Goal

- Given a large number (N in the millions or billions) of text documents, find pairs that are "near duplicates."

Finding Similar Documents

Goal

- Given a large number (N in the millions or billions) of text documents, find pairs that are "near duplicates."

Applications

- Mirror websites, or approximate mirrors.
  - We do not want to show both of them in a search.
- Similar news articles at many news sites.
  - Cluster articles by "same story."

Finding Similar Documents

Goal

- Given a large number (N in the millions or billions) of text documents, find pairs that are "near duplicates."

Applications

- Mirror websites, or approximate mirrors.
  - We do not want to show both of them in a search.
- Similar news articles at many news sites.
  - Cluster articles by "same story."

Finding Similar Documents

Goal

- Given a large number (N in the millions or billions) of text documents, find pairs that are "near duplicates."

Applications

- Mirror websites, or approximate mirrors.
  - We do not want to show both of them in a search.
- Similar news articles at many news sites.
  - Cluster articles by "same story."

Finding Similar Documents

Goal

- Given a large number (N in the millions or billions) of text documents, find pairs that are "near duplicates."

Applications

- Mirror websites, or approximate mirrors.
  - We do not want to show both of them in a search.
- Similar news articles at many news sites.
  - Cluster articles by "same story."

Finding Similar Documents

Goal

- Given a large number (N in the millions or billions) of text documents, find pairs that are "near duplicates."

Applications

- Mirror websites, or approximate mirrors.
  - We do not want to show both of them in a search.
- Similar news articles at many news sites.
  - Cluster articles by "same story."

Finding Similar Documents

Problems

- Many small pieces of one document can appear out of order in another.
- Too many documents to compare all pairs.
- Documents are so large or so many that they cannot fit in main memory.

Finding Similar Documents

Problems

- Many small pieces of one document can appear out of order in another.
- Too many documents to compare all pairs.
- Documents are so large or so many that they cannot fit in main memory.

Finding Similar Documents

Problems

- Many small pieces of one document can appear out of order in another.
- Too many documents to compare all pairs.
- Documents are so large or so many that they cannot fit in main memory.

3 Essential Steps for Similar Docs

Step 1: Shingling

Convert documents to sets

Step 2: Minhashing

Convert large sets to short signatures, while preserving similarity

Locality-sensitive hashing

Focus on pairs of signatures likely to be from similar documents

- Candidate pairs!

3 Essential Steps for Similar Docs

Step 1: Shingling

Convert documents to sets

Step 2: Minhashing

Convert large sets to short signatures, while preserving similarity

Locality-sensitive hashing

Focus on pairs of signatures likely to be from similar documents

- Candidate pairs!

3 Essential Steps for Similar Docs

Step 1: Shingling

Convert documents to sets

Step 2: Minhashing

Convert large sets to short signatures, while preserving similarity

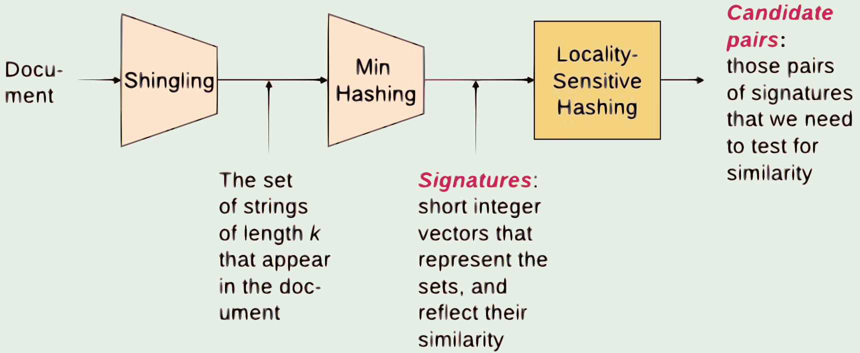
Locality-sensitive hashing

Focus on pairs of signatures likely to be from similar documents

- Candidate pairs!

The Big Picture

The Process of Identification



Outline

Documents as High-Dimensional Data

Step 1: Shingling

Convert documents to sets.

Documents as High-Dimensional Data

Step 1: Shingling

Convert documents to sets.

Simple approaches

- Document = set of words appearing in document.
- Document = set of "important" words.
- Don't work well for this application. Why?

- A different way: Use Shingles!!!

Documents as High-Dimensional Data

Step 1: Shingling

Convert documents to sets.

Simple approaches

- Document = set of words appearing in document.
- Document = set of "important" words.
- Don't work well for this application. Why?

We want to avoid to get tangled in the text structure

- Need to account for ordering of words!
- A different way: Use Shingles!!!

Documents as High-Dimensional Data

Step 1: Shingling

Convert documents to sets.

Simple approaches

- Document = set of words appearing in document.
- Document = set of "important" words.
- Don't work well for this application. Why?

We want to avoid to get tangled in the text structure

- Need to account for ordering of words!
- A different way: Use Shingles!!!

Documents as High-Dimensional Data

Step 1: Shingling

Convert documents to sets.

Simple approaches

- Document = set of words appearing in document.
- Document = set of "important" words.
- Don't work well for this application. Why?

We want to avoid to get tangled in the text structure

- Need to account for ordering of words!
- A different way: Use Shingles!!!

Documents as High-Dimensional Data

Step 1: Shingling

Convert documents to sets.

Simple approaches

- Document = set of words appearing in document.
- Document = set of "important" words.
- Don't work well for this application. Why?

We want to avoid to get tangled in the text structure

- Need to account for ordering of words!
- A different way: Use Shingles!!!

Outline

Shingles

k-shingle

- A k-shingle (or k-gram) for a document is a sequence of k tokens that appears in the doc.
    - Tokens can be characters, words or something else, depending on the application.
    - Assume tokens = characters for the examples.

Shingles

k-shingle

- A k-shingle (or k-gram) for a document is a sequence of k tokens that appears in the doc.
  - Tokens can be characters, words or something else, depending on the application.
  - Assume tokens = characters for the examples.

Example

- $k = 2$; document $D_1 = abcab$ Set of 2-shingles: $S(D_1) = \{ab, bc, ca\}$
  - One possible option: Shingles as a bag (multiset). Thus, count ab twice: $S'(D_1) = \{ab, bc, ca, ab\}$

Shingles

k-shingle

- A k-shingle (or k-gram) for a document is a sequence of k tokens that appears in the doc.
  - Tokens can be characters, words or something else, depending on the application.
  - Assume tokens $=$ characters for the examples.

Example

- $k = 2$; document $D_1 = abcab$ Set of 2-shingles: $S(D_1) = \{ab, bc, ca\}$
  - One possible option: Shingles as a bag (multiset). Thus, count ab twice: $S'(D_1) = \{ab, bc, ca, ab\}$

Shingles

k-shingle

- A k-shingle (or k-gram) for a document is a sequence of k tokens that appears in the doc.
  - Tokens can be characters, words or something else, depending on the application.
  - Assume tokens = characters for the examples.

Example

- $k = 2$; document $D_1 = abcab$ Set of 2-shingles: $S(D_1) = \{ab, bc, ca\}$
  - One possible option: Shingles as a bag (multiset). Thus, count ab twice: $S'(D_1) = \{ab, bc, ca, ab\}$

Shingles

k-shingle

- A k-shingle (or k-gram) for a document is a sequence of k tokens that appears in the doc.
  - Tokens can be characters, words or something else, depending on the application.
  - Assume tokens = characters for the examples.

Example

- $k = 2$; document $D_1 = abcab$ Set of 2-shingles: $S(D_1) = \{ab, bc, ca\}$
  - One possible option: Shingles as a bag (multiset). Thus, count ab twice: $S'(D_1) = \{ab, bc, ca, ab\}$

Compressing Shingles

Compress

- To compress long shingles, we can hash them to (say) 4 bytes.

Idea

Compressing Shingles

Compress

- To compress long shingles, we can hash them to (say) 4 bytes.

Represent a doc

- Represent a doc by the set of hash values of its k-shingles.
  - Idea: Two documents could (rarely) appear to have shingles in common, when in fact only the hash-values were shared.

- Hash the shingles using the division method to a hash table.

Compressing Shingles

Compress

- To compress long shingles, we can hash them to (say) 4 bytes.

Represent a doc

- Represent a doc by the set of hash values of its k-shingles.
  - Idea: Two documents could (rarely) appear to have shingles in common, when in fact only the hash-values were shared.

Example

- $k = 2$; document $D_1 = abcab$ Set of 2-shingles: $S(D_1) = \{ab, bc, ca\}$
- Hash the shingles using the division method to a hash table.

Compressing Shingles

Compress

- To compress long shingles, we can hash them to (say) 4 bytes.

Represent a doc

- Represent a doc by the set of hash values of its k-shingles.
  - Idea: Two documents could (rarely) appear to have shingles in common, when in fact only the hash-values were shared.

Example

- $k = 2$; document $D_1 = abcab$ Set of 2-shingles: $S(D_1) = \{ab, bc, ca\}$
- Hash the shingles using the division method to a hash table.

Compressing Shingles

Compress

- To compress long shingles, we can hash them to (say) 4 bytes.

Represent a doc

- Represent a doc by the set of hash values of its k-shingles.
  - Idea: Two documents could (rarely) appear to have shingles in common, when in fact only the hash-values were shared.

Example

- $k = 2$; document $D_1 = abcab$ Set of 2-shingles: $S(D_1) = \{ab, bc, ca\}$
- Hash the shingles using the division method to a hash table.

Similarity Metric for Shingles

Document

- Document D_1 = set of k-shingles $C_1 = S(D_1)$

Similarity Metric for Shingles

Document

- Document D_1 = set of k-shingles $C_1 = S(D_1)$

$0/1$ vector

- Equivalently, each document is a $0/1$ vector in the space of k-shingles
  - Each unique shingle is a dimension.
  - Problem!!! Vectors are very sparse.
    - We need a measure that can handle this situation.

Similarity Metric for Shingles

Document

- Document D_1 = set of k-shingles $C_1 = S(D_1)$

$0/1$ vector

- Equivalently, each document is a $0/1$ vector in the space of k-shingles
  - Each unique shingle is a dimension.
  - Problem!!! Vectors are very sparse.
    - We need a measure that can handle this situation.

A natural similarity measure is the Jaccard similarity

$$sim(D_1, D_2) = \frac{|D_1 \cap D_2|}{|D_1 \cup D_2|} \quad (1)$$

Similarity Metric for Shingles

Document

- Document D_1 = set of k-shingles $C_1 = S(D_1)$

$0/1$ vector

- Equivalently, each document is a $0/1$ vector in the space of k-shingles
  - Each unique shingle is a dimension.
  - Problem!!! Vectors are very sparse.
    - We need a measure that can handle this situation.

A natural similarity measure is the Jaccard similarity

$$sim\left(D_1, D_2\right) = \frac{|D_1 \cap D_2|}{|D_1 \cup D_2|} \tag{1}$$

Similarity Metric for Shingles

Document

- Document D_1 = set of k-shingles $C_1 = S(D_1)$

$0/1$ vector

- Equivalently, each document is a $0/1$ vector in the space of k-shingles
  - Each unique shingle is a dimension.
  - Problem!!! Vectors are very sparse.
    - We need a measure that can handle this situation.

A natural similarity measure is the Jaccard similarity

$$sim\left(D_1, D_2\right) = \frac{|D_1 \cap D_2|}{|D_1 \cup D_2|} \quad (1)$$

Similarity Metric for Shingles

Document

- Document D_1 = set of k-shingles $C_1 = S(D_1)$

$0/1$ vector

- Equivalently, each document is a $0/1$ vector in the space of k-shingles
  - Each unique shingle is a dimension.
  - Problem!!! Vectors are very sparse.
    - We need a measure that can handle this situation.

A natural similarity measure is the Jaccard similarity

$$sim\left(D_1, D_2\right) = \frac{|D_1 \cap D_2|}{|D_1 \cup D_2|} \quad (1)$$

Remember the SWAR-Popcount

Code - SWAR-Popcount - Divide and Conquer

```
// This works only in 32 bits
int product(int row, int vector){

  int i = row & vector;

  i = i - ((i >> 1) & 0x55555555);
  i = (i & 0x33333333) + ((i >> 2) & 0x33333333);
  i = (((i + (i >> 4)) & 0x0F0F0F0F) * 0x01010101) >> 24;

  return i & 0x00000001;

}
```

We can use this

Together with AND and OR to implement the Jaccard similarity

Remember the SWAR-Popcount

Code - SWAR-Popcount - Divide and Conquer

```
// This works only in 32 bits
int product(int row,int vector){

  int i = row & vector;

  i = i - ((i >> 1) & 0x55555555);
  i = (i & 0x33333333) + ((i >> 2) & 0x33333333);
  i = (((i + (i >> 4)) & 0x0F0F0F0F) * 0x01010101) >> 24;

  return i & 0x00000001;

}
```

We can use this

Together with AND and OR to implement the Jaccard similarity

Working Assumption

Similar text

- Documents that have lots of shingles in common have similar text, even if the text appears in different order.

Working Assumption

Similar text

- Documents that have lots of shingles in common have similar text, even if the text appears in different order.

Caveat

- You must pick k large enough, or most documents will have most shingles.
- It seems to be that
  - $k = 5$ is OK for short documents.
  - $k = 10$ is better for long documents.

Working Assumption

Similar text

- Documents that have lots of shingles in common have similar text, even if the text appears in different order.

Caveat

- You must pick k large enough, or most documents will have most shingles.
- It seems to be that
  - $k = 5$ is OK for short documents.
  - $k = 10$ is better for long documents.

Working Assumption

Similar text

- Documents that have lots of shingles in common have similar text, even if the text appears in different order.

Caveat

- You must pick k large enough, or most documents will have most shingles.
- It seems to be that
  - $k = 5$ is OK for short documents.
  - $k = 10$ is better for long documents.

Working Assumption

Similar text

- Documents that have lots of shingles in common have similar text, even if the text appears in different order.

Caveat

- You must pick k large enough, or most documents will have most shingles.
- It seems to be that
  - $k = 5$ is OK for short documents.
  - $k = 10$ is better for long documents.

Motivation for Minhash/LSH

Imagine the following

We need to find near-duplicate documents among $N = 1,000,000$ documents.

Motivation for Minhash/LSH

Imagine the following

We need to find near-duplicate documents among $N = 1,000,000$ documents.

Compute pairwise Jaccard similarites

- Naïvely, we would have to compute pairwise Jaccard similarites for every pair of docs.
  - i.e, $N(N-1)/2 \approx 5 * 10^{11}$ comparisons.
  - At 10^5 secs/day and 10^6 comparisons/sec, it would take 5 days.

Motivation for Minhash/LSH

Imagine the following

We need to find near-duplicate documents among $N = 1,000,000$ documents.

Compute pairwaise Jaccard similarites

- Naïvely, we would have to compute pairwaise Jaccard similarites for every pair of docs.
  - i.e, $N(N-1)/2 \approx 5 * 10^{11}$ comparisons.
  - At 10^5 secs/day and 10^6 comparisons/sec, it would take 5 days.

For something larger

For $N = 10$ million, it takes more than a year...

Motivation for Minhash/LSH

Imagine the following

We need to find near-duplicate documents among $N = 1,000,000$ documents.

Compute pairwaise Jaccard similarites

- Naïvely, we would have to compute pairwaise Jaccard similarites for every pair of docs.
  - i.e, $N(N-1)/2 \approx 5 * 10^{11}$ comparisons.
  - At 10^5 secs/day and 10^6 comparisons/sec, it would take 5 days.

For something larger

For $N = 10$ million, it takes more than a year...

Motivation for Minhash/LSH

Imagine the following

We need to find near-duplicate documents among $N = 1,000,000$ documents.

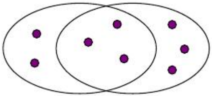
Compute pairwaise Jaccard similarites

- Naïvely, we would have to compute pairwaise Jaccard similarites for every pair of docs.
  - i.e, $N(N-1)/2 \approx 5 * 10^{11}$ comparisons.
  - At 10^5 secs/day and 10^6 comparisons/sec, it would take 5 days.

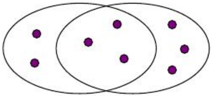
For something larger

For $N = 10$ million, it takes more than a year...

Outline



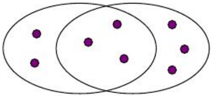
Encoding Sets as Bit Vectors



Something Notable

- Many similarity problems can be formalized as finding subsets that have significant intersection.

Encoding Sets as Bit Vectors



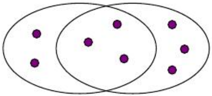
Something Notable

- Many similarity problems can be formalized as finding subsets that have significant intersection.

Encode sets

- Encode sets using 0/1 (bit, boolean) vectors.
  - One dimension per element in the universal set.
- Interpret set intersection as bitwise **AND**, and set union as bitwise **OR**.

Encoding Sets as Bit Vectors



Something Notable

- Many similarity problems can be formalized as finding subsets that have significant intersection.

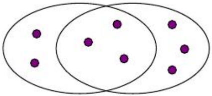
Encode sets

- Encode sets using 0/1 (bit, boolean) vectors.
  - One dimension per element in the universal set.
- Interpret set intersection as bitwise **AND**, and set union as bitwise **OR**.

Example

- $C_1 = 10111$; $C_2 = 10011$.
  - Size of intersection $= 3$; size of union $= 4$, Jaccard similarity (not distance) $= 3/4$
  - $d(C_1, C_2) = 1-$(Jaccard similarity) $= 1/4$

Encoding Sets as Bit Vectors



Something Notable

- Many similarity problems can be formalized as finding subsets that have significant intersection.

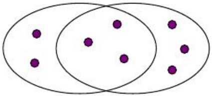
Encode sets

- Encode sets using 0/1 (bit, boolean) vectors.
  - One dimension per element in the universal set.
- Interpret set intersection as bitwise **AND**, and set union as bitwise **OR.**

Example

- $C_1 = 10111; C_2 = 10011.$
  - Size of intersection $= 3$; size of union $= 4$, Jaccard similarity (not distance) $= 3/4$
  - $d(C_1, C_2) = 1-$(Jaccard similarity) $= 1/4$

Encoding Sets as Bit Vectors



Something Notable

- Many similarity problems can be formalized as finding subsets that have significant intersection.

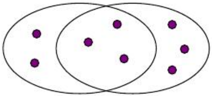
Encode sets

- Encode sets using 0/1 (bit, boolean) vectors.
  - One dimension per element in the universal set.
- Interpret set intersection as bitwise **AND**, and set union as bitwise **OR.**

Example

- $C_1 = 10111$; $C_2 = 10011$.
  - Size of intersection = 3; size of union = 4, Jaccard similarity (not distance) = 3/4
  - $d(C_1, C_2) = 1-$(Jaccard similarity) $= 1/4$

Encoding Sets as Bit Vectors



Something Notable

- Many similarity problems can be formalized as finding subsets that have significant intersection.

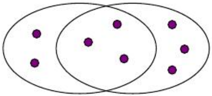
Encode sets

- Encode sets using 0/1 (bit, boolean) vectors.
  - One dimension per element in the universal set.
- Interpret set intersection as bitwise **AND**, and set union as bitwise **OR.**

Example

- $C_1 = 10111$; $C_2 = 10011$.
  - Size of intersection $= 3$; size of union $= 4$, Jaccard similarity (not distance) $= 3/4$
  - $d(C_1, C_2) = 1-$(Jaccard similarity) $= 1/4$

Encoding Sets as Bit Vectors



Something Notable

- Many similarity problems can be formalized as finding subsets that have significant intersection.

Encode sets

- Encode sets using 0/1 (bit, boolean) vectors.
  - One dimension per element in the universal set.
- Interpret set intersection as bitwise **AND**, and set union as bitwise **OR.**

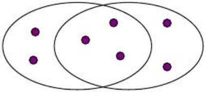
Example

- $C_1 = 10111$; $C_2 = 10011$.
  - Size of intersection $= 3$; size of union $= 4$, Jaccard similarity (not distance) $= 3/4$
  - $d(C_1, C_2) = 1-$(Jaccard similarity) $= 1/4$

From Sets to Boolean Matrices

Rows

- Rows are equal to elements (shingles)



1	1	1	0
1	1	0	1
0	1	0	1
0	0	0	1
1	0	0	1
1	1	1	0
1	0	1	0

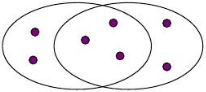
From Sets to Boolean Matrices

Rows

- Rows are equal to elements (shingles)

Columns

- The Columns are equal to sets (documents)
  - 1 in row e and column s if and only if e is a member of s
  - Column similarity is the Jaccard similarity of the corresponding sets (rows with value 1)
  - **Typical matrix is sparse!**



1	1	1	0
1	1	0	1
0	1	0	1
0	0	0	1
1	0	0	1
1	1	1	0
1	0	1	0

From Sets to Boolean Matrices

Rows

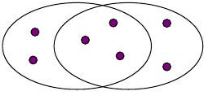
- Rows are equal to elements (shingles)

Columns

- The Columns are equal to sets (documents)
  - 1 in row e and column s if and only if e is a member of s
  - Column similarity is the Jaccard similarity of the corresponding sets (rows with value 1)
  - **Typical matrix is sparse!**

Each document is a column

- Example: $sim(C_1, C_2) = ?$
  - Size of intersection $= 3$; size of union $= 6$, Jaccard similarity (not distance) $= 3/6$
  - $d(C_1, C_2) = 1-$(Jaccard similarity) $= 3/6$



1	1	1	0
1	1	0	1
0	1	0	1
0	0	0	1
1	0	0	1
1	1	1	0
1	0	1	0

From Sets to Boolean Matrices

Rows

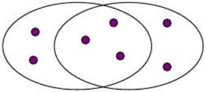
- Rows are equal to elements (shingles)

Columns

- The Columns are equal to sets (documents)
  - 1 in row e and column s if and only if e is a member of s
  - Column similarity is the Jaccard similarity of the corresponding sets (rows with value 1)
  - Typical matrix is sparse!

Each document is a column

- Example: $sim(C_1, C_2) = ?$
  - Size of intersection $= 3$; size of union $= 6$, Jaccard similarity (not distance) $= 3/6$
  - $d(C_1, C_2) = 1-$(Jaccard similarity) $= 3/6$



1	1	1	0
1	1	0	1
0	1	0	1
0	0	0	1
1	0	0	1
1	1	1	0
1	0	1	0

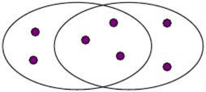
From Sets to Boolean Matrices

Rows

- Rows are equal to elements (shingles)

Columns

- The Columns are equal to sets (documents)
  - 1 in row e and column s if and only if e is a member of s
  - Column similarity is the Jaccard similarity of the corresponding sets (rows with value 1)
  - **Typical matrix is sparse!**



1	1	1	0
1	1	0	1
0	1	0	1
0	0	0	1
1	0	0	1
1	1	1	0
1	0	1	0

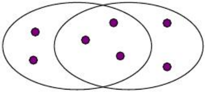
Each document is a column

- Example: $sim(C_1, C_2) = ?$
  - Size of intersection $= 3$; size of union $= 6$, Jaccard similarity (not distance) $= 3/6$
  - $d(C_1, C_2) = 1-$(Jaccard similarity) $= 3/6$

From Sets to Boolean Matrices

Rows

- Rows are equal to elements (shingles)



Columns

- The Columns are equal to sets (documents)
  - 1 in row e and column s if and only if e is a member of s
  - Column similarity is the Jaccard similarity of the corresponding sets (rows with value 1)
  - **Typical matrix is sparse!**

1	1	1	0
1	1	0	1
0	1	0	1
0	0	0	1
1	0	0	1
1	1	1	0
1	0	1	0

Each document is a column

- Example: $sim(C_1, C_2) = ?$
  - Size of intersection $= 3$; size of union $= 6$, Jaccard similarity (not distance) $= 3/6$
  - $d(C_1, C_2) = 1-$(Jaccard similarity) $= 3/6$

From Sets to Boolean Matrices

Rows

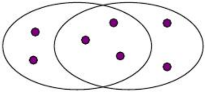
- Rows are equal to elements (shingles)

Columns

- The Columns are equal to sets (documents)
  - 1 in row e and column s if and only if e is a member of s
  - Column similarity is the Jaccard similarity of the corresponding sets (rows with value 1)
  - **Typical matrix is sparse!**

Each document is a column

- Example: $sim(C_1, C_2) = ?$
  - Size of intersection $= 3$; size of union $= 6$, Jaccard similarity (not distance) $= 3/6$
  - $d(C_1, C_2) = 1-$(Jaccard similarity) $= 3/6$



1	1	1	0
1	1	0	1
0	1	0	1
0	0	0	1
1	0	0	1
1	1	1	0
1	0	1	0

From Sets to Boolean Matrices

Rows

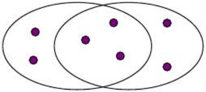
- Rows are equal to elements (shingles)

Columns

- The Columns are equal to sets (documents)
  - 1 in row e and column s if and only if e is a member of s
  - Column similarity is the Jaccard similarity of the corresponding sets (rows with value 1)
  - **Typical matrix is sparse!**

Each document is a column

- Example: $sim(C_1, C_2) = ?$
  - Size of intersection $= 3$; size of union $= 6$, Jaccard similarity (not distance) $= 3/6$
  - $d(C_1, C_2) = 1-$(Jaccard similarity) $= 3/6$



1	1	1	0
1	1	0	1
0	1	0	1
0	0	0	1
1	0	0	1
1	1	1	0
1	0	1	0

Outline

Outline: Finding Similar Columns

So far and next goal

- So far:
  - Documents → Sets of shingles
  - Represent sets as boolean vectors in a matrix
- Next Goal: Find similar columns, Small signatures

Outline: Finding Similar Columns

So far and next goal

- So far:
  - Documents → Sets of shingles
  - Represent sets as boolean vectors in a matrix
- Next Goal: Find similar columns, Small signatures

Approach

1. Signatures of columns: small summaries of columns
2. Examine pairs of signatures to find similar columns
   - Essential: Similarities of signatures & columns are related
3. Optional: Check that columns with similar signatures are really similar

- These methods can produce false negatives, and even false positives (if the optional check is not made)

Outline: Finding Similar Columns

So far and next goal

- So far:
  - Documents → Sets of shingles
  - Represent sets as boolean vectors in a matrix
- Next Goal: Find similar columns, Small signatures

Approach

1. Signatures of columns: small summaries of columns
2. Examine pairs of signatures to find similar columns
   - Essential: Similarities of signatures & columns are related
3. Optional: Check that columns with similar signatures are really similar

Warnings

- Comparing all pairs may take too much time: Job for Locality Sensitive Hashing (LSH)
  - These methods can produce false negatives, and even false positives (if the optional check is not made)

Outline: Finding Similar Columns

So far and next goal

- So far:
  - Documents → Sets of shingles
  - Represent sets as boolean vectors in a matrix
- Next Goal: Find similar columns, Small signatures

Approach

1. Signatures of columns: small summaries of columns
2. Examine pairs of signatures to find similar columns
   - Essential: Similarities of signatures & columns are related
3. Optional: Check that columns with similar signatures are really similar

Warnings

- Comparing all pairs may take too much time: Job for Locality Sensitive Hashing (LSH)
  - These methods can produce false negatives, and even false positives (if the optional check is not made)

Outline: Finding Similar Columns

So far and next goal

- So far:
  - Documents → Sets of shingles
  - Represent sets as boolean vectors in a matrix
- Next Goal: Find similar columns, Small signatures

Approach

1. Signatures of columns: small summaries of columns
2. Examine pairs of signatures to find similar columns
   - Essential: Similarities of signatures & columns are related
3. Optional: Check that columns with similar signatures are really similar

Warnings

- Comparing all pairs may take too much time: Job for Locality Sensitive Hashing (LSH)
  - These methods can produce false negatives, and even false positives (if the optional check is not made)

Outline: Finding Similar Columns

So far and next goal

- So far:
  - Documents → Sets of shingles
  - Represent sets as boolean vectors in a matrix
- Next Goal: Find similar columns, Small signatures

Approach

1. Signatures of columns: small summaries of columns
2. Examine pairs of signatures to find similar columns
   - Essential: Similarities of signatures & columns are related
3. Optional: Check that columns with similar signatures are really similar

Warnings

- Comparing all pairs may take too much time: Job for Locality Sensitive Hashing (LSH)
  - These methods can produce false negatives, and even false positives (if the optional check is not made)

Outline: Finding Similar Columns

So far and next goal

- So far:
  - Documents → Sets of shingles
  - Represent sets as boolean vectors in a matrix
- Next Goal: Find similar columns, Small signatures

Approach

1. Signatures of columns: small summaries of columns
2. Examine pairs of signatures to find similar columns
   - Essential: Similarities of signatures & columns are related
3. Optional: Check that columns with similar signatures are really similar

Warnings

- Comparing all pairs may take too much time: Job for Locality Sensitive Hashing (LSH)
  - These methods can produce false negatives, and even false positives (if the optional check is not made)

Outline: Finding Similar Columns

So far and next goal

- So far:
  - Documents → Sets of shingles
  - Represent sets as boolean vectors in a matrix
- Next Goal: Find similar columns, Small signatures

Approach

1. Signatures of columns: small summaries of columns
2. Examine pairs of signatures to find similar columns
   - Essential: Similarities of signatures & columns are related
3. Optional: Check that columns with similar signatures are really similar

Warnings

- Comparing all pairs may take too much time: Job for Locality Sensitive Hashing (LSH)
  - These methods can produce false negatives, and even false positives (if the optional check is not made)

Outline: Finding Similar Columns

So far and next goal

- So far:
    - Documents → Sets of shingles
    - Represent sets as boolean vectors in a matrix
- Next Goal: Find similar columns, Small signatures

Approach

1. Signatures of columns: small summaries of columns
2. Examine pairs of signatures to find similar columns
    - Essential: Similarities of signatures & columns are related
3. Optional: Check that columns with similar signatures are really similar

Warnings

- Comparing all pairs may take too much time: Job for Locality Sensitive Hashing (LSH)
    - These methods can produce false negatives, and even false positives (if the optional check is not made)

Outline: Finding Similar Columns

So far and next goal

- So far:
  - Documents → Sets of shingles
  - Represent sets as boolean vectors in a matrix
- Next Goal: Find similar columns, Small signatures

Approach

1. Signatures of columns: small summaries of columns
2. Examine pairs of signatures to find similar columns
   - Essential: Similarities of signatures & columns are related
3. Optional: Check that columns with similar signatures are really similar

Warnings

- Comparing all pairs may take too much time: Job for Locality Sensitive Hashing (LSH)
  - These methods can produce false negatives, and even false positives (if the optional check is not made)

Hashing Columns (Signatures)

Key idea

- "Hash" each column C to a small signature $h(C)$, such that:
  - (1) $h(C)$ is small enough that the signature fits in RAM.
  - (2) $sim(C_1, C_2)$ is the same as the "similarity" of signatures $h(C_1)$ and $h(C_2)$.

Hashing Columns (Signatures)

Key idea

- "Hash" each column C to a small signature $h(C)$, such that:
  - (1) $h(C)$ is small enough that the signature fits in RAM.
  - (2) $sim(C_1, C_2)$ is the same as the "similarity" of signatures $h(C_1)$ and $h(C_2)$.

Goal

- Find a hash function $h(\cdot)$ such that:
  - if $sim(C_1, C_2)$ is high, then with high prob. $h(C_1) = h(C_2)$.
  - if $sim(C_1, C_2)$ is low, then with high prob. $h(C_1) \neq h(C_2)$.

Hashing Columns (Signatures)

Key idea

- "Hash" each column C to a small signature $h(C)$, such that:
  - (1) $h(C)$ is small enough that the signature fits in RAM.
  - (2) $sim(C_1, C_2)$ is the same as the "similarity" of signatures $h(C_1)$ and $h(C_2)$.

Goal

- Find a hash function $h(\cdot)$ such that:
  - if $sim(C_1, C_2)$ is high, then with high prob. $h(C_1) = h(C_2)$.
  - if $sim(C_1, C_2)$ is low, then with high prob. $h(C_1) \neq h(C_2)$.

Buckets

- Thus, we hash documents into buckets, and expect that "most" pairs of near duplicate docs hash into the same bucket!

Hashing Columns (Signatures)

Key idea

- "Hash" each column C to a small signature $h(C)$, such that:
  - (1) $h(C)$ is small enough that the signature fits in RAM.
  - (2) $sim(C_1, C_2)$ is the same as the "similarity" of signatures $h(C_1)$ and $h(C_2)$.

Goal

- Find a hash function $h(\cdot)$ such that:
  - if $sim(C_1, C_2)$ is high, then with high prob. $h(C_1) = h(C_2)$.
  - if $sim(C_1, C_2)$ is low, then with high prob. $h(C_1) \neq h(C_2)$.

Buckets

- Thus, we hash documents into buckets, and expect that "most" pairs of near duplicate docs hash into the same bucket!

Hashing Columns (Signatures)

Key idea

- "Hash" each column C to a small signature $h(C)$, such that:
  - (1) $h(C)$ is small enough that the signature fits in RAM.
  - (2) $sim(C_1, C_2)$ is the same as the "similarity" of signatures $h(C_1)$ and $h(C_2)$.

Goal

- Find a hash function $h(\cdot)$ such that:
  - if $sim(C_1, C_2)$ is high, then with high prob. $h(C_1) = h(C_2)$.
  - if $sim(C_1, C_2)$ is low, then with high prob. $h(C_1) \neq h(C_2)$.

Buckets

- Thus, we hash documents into buckets, and expect that "most" pairs of near duplicate docs hash into the same bucket!

Hashing Columns (Signatures)

Key idea

- "Hash" each column C to a small signature $h(C)$, such that:
  - (1) $h(C)$ is small enough that the signature fits in RAM.
  - (2) $sim(C_1, C_2)$ is the same as the "similarity" of signatures $h(C_1)$ and $h(C_2)$.

Goal

- Find a hash function $h(\cdot)$ such that:
  - if $sim(C_1, C_2)$ is high, then with high prob. $h(C_1) = h(C_2)$.
  - if $sim(C_1, C_2)$ is low, then with high prob. $h(C_1) \neq h(C_2)$.

Buckets

- Thus, we hash documents into buckets, and expect that "most" pairs of near duplicate docs hash into the same bucket!

Hashing Columns (Signatures)

Key idea

- “Hash” each column C to a small signature $h(C)$, such that:
  - (1) $h(C)$ is small enough that the signature fits in RAM.
  - (2) $sim(C_1, C_2)$ is the same as the “similarity” of signatures $h(C_1)$ and $h(C_2)$.

Goal

- Find a hash function $h(\cdot)$ such that:
  - if $sim(C_1, C_2)$ is high, then with high prob. $h(C_1) = h(C_2)$.
  - if $sim(C_1, C_2)$ is low, then with high prob. $h(C_1) \neq h(C_2)$.

Buckets

- Thus, we hash documents into buckets, and expect that “most” pairs of near duplicate docs hash into the same bucket!

Outline

Min-Hashing

Goal

- Find a hash function $h(\cdot)$ such that:
  - if $sim(C_1, C_2)$ is high, then with high prob. $h(C1) = h(C2)$
  - if $sim(C_1, C_2)$ is low, then with high prob. $h(C_1) \neq h(C_2)$

Min-Hashing

Goal

- Find a hash function $h(\cdot)$ such that:
  - if $sim(C_1, C_2)$ is high, then with high prob. $h(C1) = h(C2)$
  - if $sim(C_1, C_2)$ is low, then with high prob. $h(C_1) \neq h(C_2)$

Similarity metric

- Clearly, the hash function depends on the similarity metric:
  - Not all similarity metrics have a suitable hash function.

Min-Hashing

Goal

- Find a hash function $h(\cdot)$ such that:
  - if $sim(C_1, C_2)$ is high, then with high prob. $h(C1) = h(C2)$
  - if $sim(C_1, C_2)$ is low, then with high prob. $h(C_1) \neq h(C_2)$

Similarity metric

- Clearly, the hash function depends on the similarity metric:
  - Not all similarity metrics have a suitable hash function.

Hash function

- There is a suitable hash function for Jaccard similarity: Min-hashing.

Min-Hashing

Goal

- Find a hash function $h(\cdot)$ such that:
  - if $sim(C_1, C_2)$ is high, then with high prob. $h(C1) = h(C2)$
  - if $sim(C_1, C_2)$ is low, then with high prob. $h(C_1) \neq h(C_2)$

Similarity metric

- Clearly, the hash function depends on the similarity metric:
  - Not all similarity metrics have a suitable hash function.

Hash function

- There is a suitable hash function for Jaccard similarity: Min-hashing.

Min-Hashing

Goal

- Find a hash function $h(\cdot)$ such that:
  - if $sim(C_1, C_2)$ is high, then with high prob. $h(C1) = h(C2)$
  - if $sim(C_1, C_2)$ is low, then with high prob. $h(C_1) \neq h(C_2)$

Similarity metric

- Clearly, the hash function depends on the similarity metric:
  - Not all similarity metrics have a suitable hash function.

Hash function

- There is a suitable hash function for Jaccard similarity: **Min-hashing**.

Min-Hashing

Goal

- Find a hash function $h(\cdot)$ such that:
  - if $sim(C_1, C_2)$ is high, then with high prob. $h(C1) = h(C2)$
  - if $sim(C_1, C_2)$ is low, then with high prob. $h(C_1) \neq h(C_2)$

Similarity metric

- Clearly, the hash function depends on the similarity metric:
  - Not all similarity metrics have a suitable hash function.

Hash function

- There is a suitable hash function for Jaccard similarity: **Min-hashing.**

Min-Hashing

Goal

- Find a hash function $h(\cdot)$ such that:
  - if $sim(C_1, C_2)$ is high, then with high prob. $h(C1) = h(C2)$
  - if $sim(C_1, C_2)$ is low, then with high prob. $h(C_1) \neq h(C_2)$

Similarity metric

- Clearly, the hash function depends on the similarity metric:
  - Not all similarity metrics have a suitable hash function.

Hash function

- There is a suitable hash function for Jaccard similarity: **Min-hashing.**

Min-Hashing

Random permutation

Imagine the rows of the boolean matrix permuted under random permutation π .

"Hash" function $h_\pi(C)$

- Define a "hash" function $h_\pi(C)$ = the number of the first (in the permuted order π) row in which column C has value 1:

$$h_\pi(C) = min_\pi \pi(C)$$

What can we do?

- Use several (e.g., 100) independent hash functions to create a signature of a column

Min-Hashing

Random permutation

Imagine the rows of the boolean matrix permuted under random permutation π .

"Hash" function $h_\pi(C)$

- Define a "hash" function $h_\pi(C)$ = the number of the first (in the permuted order π) row in which column C has value 1:

$$h_\pi(C) = min_\pi \pi(C)$$

What can we do?

- Use several (e.g., 100) independent hash functions to create a signature of a column

Min-Hashing

Random permutation

Imagine the rows of the boolean matrix permuted under random permutation π .

"Hash" function $h_\pi(C)$

- Define a "hash" function $h_\pi(C)$ = the number of the first (in the permuted order π) row in which column C has value 1:

$$h_\pi(C) = min_\pi \pi(C)$$

What can we do?

- Use several (e.g., 100) independent hash functions to create a signature of a column

Min-Hashing Example

Note: Another (equivalent) way is to store row indexes:

1	5	1	5
2	3	1	3
6	4	6	4

Something Notable

2nd element of the permutation is the first to map to a 1

Permutation π

2	4	3
3	2	4
7	1	7
6	3	2
1	6	6
5	7	1
4	5	5

Input matrix (Shingles x Documents)

1	0	1	0
1	0	0	1
0	1	0	1
0	1	0	1
0	1	0	1
1	0	1	0
1	0	1	0

Signature matrix *M*

2	1	2	1
2	1	4	1
1	2	1	2

4th element of the permutation is the first to map to a 1

Surprising Property

0	0
0	0
1	**1**
0	0
0	1
1	0

- Choose a random permutation π
- Claim: $Pr[h_\pi(C_1) = h_\pi(C_2)] = sim(C_1, C_2)$ Why?

Surprising Property

0	0
0	0
1	1
0	0
0	1
1	0

- Choose a random permutation π
- Claim: $Pr[h_\pi(C_1) = h_\pi(C_2)] = sim(C_1, C_2)$ Why?

Why?

- Let X be a document (set of shingles)
- Then: $Pr[\pi(x) = min(\pi(X))] = 1/|X|$
- It is equally likely that any $x \in X$ is mapped to the min element
- Let x be s.t. $\pi(x) = min(\pi(C_1 \bigcup C_2))$
- Then either: $\pi(x) = min(\pi(C_1))$ if $x \in C_1$, or $\pi(x) = min(\pi(C_2))$ if $x \in C2$
  - One of the two cols had to have 1 at position x
- So the prob. that both are true is the prob. $x \in C_1 \bigcap C_2$

$$Pr[min(\pi(C_1)) = min(\pi(C_2))] = \frac{|C_1 \bigcap C_2|}{|C_1 \cup C_2|} = sim(C_1, C_2) \qquad (2)$$

Surprising Property

0	0
0	0
1	**1**
0	0
0	1
1	0

- Choose a random permutation π
- Claim: $Pr[h_\pi(C_1) = h_\pi(C_2)] = sim(C_1, C_2)$ Why?

Why?

- Let X be a document (set of shingles)
- Then: $Pr[\pi(x) = min(\pi(X))] = 1/|X|$
- It is equally likely that any $x \in X$ is mapped to the min element
- Let x be s.t. $\pi(x) = min(\pi(C_1 \bigcup C_2))$
- Then either: $\pi(x) = min(\pi(C_1))$ if $x \in C_1$, or $\pi(x) = min(\pi(C_2))$ if $x \in C2$
  - One of the two cols had to have 1 at position x
- So the prob. that both are true is the prob. $x \in C_1 \bigcap C_2$

$$Pr[min(\pi(C_1)) = min(\pi(C_2))] = \frac{|C_1 \bigcap C_2|}{|C_1 \cup C_2|} = sim(C_1, C_2) \quad (2)$$

Surprising Property

0	0
0	0
1	1
0	0
0	1
1	0

- Choose a random permutation π
- Claim: $Pr[h_\pi(C_1) = h_\pi(C_2)] = sim(C_1, C_2)$ Why?

Why?

- Let X be a document (set of shingles)
- Then: $Pr[\pi(x) = min(\pi(X))] = 1/|X|$
- It is equally likely that any $x \in X$ is mapped to the min element
- Let x be s.t. $\pi(x) = min(\pi(C_1 \bigcup C_2))$
- Then either: $\pi(x) = min(\pi(C_1))$ if $x \in C_1$, or $\pi(x) = min(\pi(C_2))$ if $x \in C2$
  - One of the two cols had to have 1 at position x
- So the prob. that both are true is the prob. $x \in C_1 \bigcap C_2$

$$Pr[min(\pi(C_1)) = min(\pi(C_2))] = \frac{|C_1 \bigcap C_2|}{|C_1 \cup C_2|} = sim(C_1, C_2) \qquad (2)$$

Surprising Property

0	0
0	0
1	1
0	0
0	1
1	0

- Choose a random permutation π
- Claim: $Pr[h_\pi(C_1) = h_\pi(C_2)] = sim(C_1, C_2)$ Why?

Why?

- Let X be a document (set of shingles)
- Then: $Pr[\pi(x) = min(\pi(X))] = 1/|X|$
- It is equally likely that any $x \in X$ is mapped to the min element
- Let x be s.t. $\pi(x) = min(\pi(C_1 \bigcup C_2))$
- Then either: $\pi(x) = min(\pi(C_1))$ if $x \in C_1$, or $\pi(x) = min(\pi(C_2))$ if $x \in C2$
  - One of the two cols had to have 1 at position x
- So the prob. that both are true is the prob. $x \in C_1 \bigcap C_2$

$$Pr[min(\pi(C_1)) = min(\pi(C_2))] = \frac{|C_1 \bigcap C_2|}{|C_1 \cup C_2|} = sim(C_1, C_2) \quad (2)$$

Surprising Property

0	0
0	0
1	1
0	0
0	1
1	0

- Choose a random permutation π
- Claim: $Pr[h_\pi(C_1) = h_\pi(C_2)] = sim(C_1, C_2)$ Why?

Why?

- Let X be a document (set of shingles)
- Then: $Pr[\pi(x) = min(\pi(X))] = 1/|X|$
- It is equally likely that any $x \in X$ is mapped to the min element
- Let x be s.t. $\pi(x) = min(\pi(C_1 \bigcup C_2))$
- Then either: $\pi(x) = min(\pi(C_1))$ if $x \in C_1$, or $\pi(x) = min(\pi(C_2))$ if $x \in C2$
  - One of the two cols had to have 1 at position x
- So the prob. that both are true is the prob. $x \in C_1 \bigcap C_2$

$$Pr[min(\pi(C_1)) = min(\pi(C_2))] = \frac{|C_1 \bigcap C_2|}{|C_1 \cup C_2|} = sim(C_1, C_2) \qquad (2)$$

Surprising Property

0	0
0	0
1	1
0	0
0	1
1	0

- Choose a random permutation π
- Claim: $Pr[h_\pi(C_1) = h_\pi(C_2)] = sim(C_1, C_2)$ Why?

Why?

- Let X be a document (set of shingles)
- Then: $Pr[\pi(x) = min(\pi(X))] = 1/|X|$
- It is equally likely that any $x \in X$ is mapped to the min element
- Let x be s.t. $\pi(x) = min(\pi(C_1 \bigcup C_2))$
- Then either: $\pi(x) = min(\pi(C_1))$ if $x \in C_1$, or $\pi(x) = min(\pi(C_2))$ if $x \in C2$
  - One of the two cols had to have 1 at position x
- So the prob. that both are true is the prob. $x \in C_1 \bigcap C_2$

$$Pr[min(\pi(C_1)) = min(\pi(C_2))] = \frac{|C_1 \bigcap C_2|}{|C_1 \cup C_2|} = sim(C_1, C_2) \quad (2)$$

Surprising Property

0	0
0	0
1	1
0	0
0	1
1	0

- Choose a random permutation π
- Claim: $Pr[h_\pi(C_1) = h_\pi(C_2)] = sim(C_1, C_2)$ Why?

Why?

- Let X be a document (set of shingles)
- Then: $Pr[\pi(x) = min(\pi(X))] = 1/|X|$
- It is equally likely that any $x \in X$ is mapped to the min element
- Let x be s.t. $\pi(x) = min(\pi(C_1 \bigcup C_2))$
- Then either: $\pi(x) = min(\pi(C_1))$ if $x \in C_1$, or $\pi(x) = min(\pi(C_2))$ if $x \in C2$
  - One of the two cols had to have 1 at position x
- So the prob. that both are true is the prob. $x \in C_1 \bigcap C_2$

$$Pr[min(\pi(C_1)) = min(\pi(C_2))] = \frac{|C_1 \bigcap C_2|}{|C_1 \bigcup C_2|} = sim(C_1, C_2) \quad (2)$$

Surprising Property

0	0
0	0
1	1
0	0
0	1
1	0

- Choose a random permutation π
- Claim: $Pr[h_\pi(C_1) = h_\pi(C_2)] = sim(C_1, C_2)$ Why?

Why?

- Let X be a document (set of shingles)
- Then: $Pr[\pi(x) = min(\pi(X))] = 1/|X|$
- It is equally likely that any $x \in X$ is mapped to the min element
- Let x be s.t. $\pi(x) = min(\pi(C_1 \bigcup C_2))$
- Then either: $\pi(x) = min(\pi(C_1))$ if $x \in C_1$, or $\pi(x) = min(\pi(C_2))$ if $x \in C2$
  - One of the two cols had to have 1 at position x
- So the prob. that both are true is the prob. $x \in C_1 \bigcap C_2$

$$Pr[min(\pi(C_1)) = min(\pi(C_2))] = \frac{|C_1 \bigcap C_2|}{|C_1 \bigcup C_2|} = sim(C_1, C_2) \quad (2)$$

Surprising Property

0	0
0	0
1	1
0	0
0	1
1	0

- Choose a random permutation π
- Claim: $Pr[h_\pi(C_1) = h_\pi(C_2)] = sim(C_1, C_2)$ Why?

Why?

- Let X be a document (set of shingles)
- Then: $Pr[\pi(x) = min(\pi(X))] = 1/|X|$
- It is equally likely that any $x \in X$ is mapped to the min element
- Let x be s.t. $\pi(x) = min(\pi(C_1 \bigcup C_2))$
- Then either: $\pi(x) = min(\pi(C_1))$ if $x \in C_1$, or $\pi(x) = min(\pi(C_2))$ if $x \in C2$
  - One of the two cols had to have 1 at position x
- So the prob. that both are true is the prob. $x \in C_1 \bigcap C_2$

$$Pr[min(\pi(C_1)) = min(\pi(C_2))] = \frac{|C_1 \bigcap C_2|}{|C_1 \cup C_2|} = sim(C_1, C_2) \quad (2)$$

Four Types of Rows between two Documents

Given cols C_1 and C_2, rows may be classified as

	C_1	C_2
A	1	1
B	1	0
C	0	1
D	0	0

$a = \#$rows of type A, etc.

Note

$$sim(C_1, C_2) = \frac{a}{a+b+c} \quad (3)$$

Then

- Then: $Pr[h(C_1) = h(C_2)] = sim(C_1, C_2)$

Four Types of Rows between two Documents

Given cols C_1 and C_2, rows may be classified as

	C_1	C_2
A	1	1
B	1	0
C	0	1
D	0	0

$a = \#$rows of type A, etc.

Note

$$sim(C_1, C_2) = \frac{a}{a+b+c} \quad (3)$$

Then

- Then: $Pr[h(C_1) = h(C_2)] = sim(C_1, C_2)$
  - Look down the cols C_1 and C_2 until we see a 1.
  - If it's a type-A row, then $h(C_1) = h(C_2)$. If a type-B or type-C row, then not.

Four Types of Rows between two Documents

Given cols C_1 and C_2, rows may be classified as

	C_1	C_2
A	1	1
B	1	0
C	0	1
D	0	0

$a = \#$rows of type A, etc.

Note

$$sim(C_1, C_2) = \frac{a}{a+b+c} \tag{3}$$

Then

- Then: $Pr[h(C_1) = h(C_2)] = sim(C_1, C_2)$
  - Look down the cols C_1 and C_2 until we see a 1.
  - If it's a type-A row, then $h(C_1) = h(C_2)$ If a type-B or type-C row, then not.

Four Types of Rows between two Documents

Given cols C_1 and C_2, rows may be classified as

	C_1	C_2
A	1	1
B	1	0
C	0	1
D	0	0

$a = \#$rows of type A, etc.

Note

$$sim(C_1, C_2) = \frac{a}{a+b+c} \quad (3)$$

Then

- Then: $Pr[h(C_1) = h(C_2)] = sim(C_1, C_2)$
  - Look down the cols C_1 and C_2 until we see a 1.
  - If it's a type-A row, then $h(C_1) = h(C_2)$ If a type-B or type-C row, then not.

Similarity for Signatures

We know

- $Pr[h_\pi(C_1) = h_\pi(C_2)] = sim(C_1, C_2)$
- Now generalize to multiple hash functions

Similarity for Signatures

We know

- $Pr[h_\pi(C_1) = h_\pi(C_2)] = sim(C_1, C_2)$
- Now generalize to multiple hash functions

Similarity

- The similarity of two signatures is the fraction of the hash functions in which they agree

Similarity for Signatures

We know

- $Pr[h_\pi(C_1) = h_\pi(C_2)] = sim(C_1, C_2)$
- Now generalize to multiple hash functions

Similarity

- The similarity of two signatures is the fraction of the hash functions in which they agree

Note

- Because of the minhash property, the similarity of columns is the same as the expected similarity of their signatures

Similarity for Signatures

We know

- $Pr[h_\pi(C_1) = h_\pi(C_2)] = sim(C_1, C_2)$
- Now generalize to multiple hash functions

Similarity

- The similarity of two signatures is the fraction of the hash functions in which they agree

Note

- Because of the minhash property, the similarity of columns is the same as the expected similarity of their signatures

Similarity for Signatures

We know

- $Pr[h_\pi(C_1) = h_\pi(C_2)] = sim(C_1, C_2)$
- Now generalize to multiple hash functions

Similarity

- The similarity of two signatures is the fraction of the hash functions in which they agree

Note

- Because of the minhash property, the similarity of columns is the same as the expected similarity of their signatures

Min-Hashing Example

Note: Another (equivalent) way is to store row indexes:

1	5	1	5
2	3	1	3
6	4	6	4

Example

Permutation π

2	4	3
3	2	4
7	1	7
6	3	2
1	6	6
5	7	1
4	5	5

Input matrix (Shingles x Documents)

1	0	1	0
1	0	0	1
0	1	0	1
0	1	0	1
0	1	0	1
1	0	1	0
1	0	1	0

Signature matrix M

2	1	2	1
2	1	4	1
1	2	1	2

Similarities:

	1-3	2-4	1-2	3-4
Col/Col	0.75	0.75	0	0
Sig/Sig	0.67	1.00	0	0

MinHash Signatures

- Pick $K = 100$ random permutations of the rows
- Think of $sig(C)$ (Signature of C) as a column vector

MinHash Signatures

- Pick $K = 100$ random permutations of the rows
- Think of $sig(C)$ (Signature of C) as a column vector

- $sig(C)[i]$ =according to the i-th permutation, the index of the first row that has a 1 in column C

$$sig(C)[i] = min(\pi i(C))$$

- Note: The sketch (signature) of document C is small – ~ 100 bytes!

MinHash Signatures

- Pick $K = 100$ random permutations of the rows
- Think of $sig(C)$ (Signature of C) as a column vector

- $sig(C)[i]$ =according to the i-th permutation, the index of the first row that has a 1 in column C

$$sig(C)[i] = min(\pi i(C))$$

- Note: The sketch (signature) of document C is small – ~ 100 bytes!

- We achieved our goal! We "compressed" long bit vectors into short signatures

MinHash Signatures

- Pick $K = 100$ random permutations of the rows
- Think of $sig(C)$ (Signature of C) as a column vector

- $sig(C)[i]$ =according to the i-th permutation, the index of the first row that has a 1 in column C

$$sig(C)[i] = min(\pi i(C))$$

- Note: The sketch (signature) of document C is small – ~ 100 bytes!

- We achieved our goal! We "compressed" long bit vectors into short signatures

MinHash Signatures

- Pick $K = 100$ random permutations of the rows
- Think of $sig(C)$ (Signature of C) as a column vector

- $sig(C)[i]$ =according to the i-th permutation, the index of the first row that has a 1 in column C

$$sig(C)[i] = min(\pi i(C))$$

- Note: The sketch (signature) of document C is small – ~ 100 bytes!

- We achieved our goal! We "compressed" long bit vectors into short signatures

MinHash Signatures

- Pick $K = 100$ random permutations of the rows
- Think of $sig(C)$ (Signature of C) as a column vector

- $sig(C)[i]$ =according to the i-th permutation, the index of the first row that has a 1 in column C

$$sig(C)[i] = min(\pi i(C))$$

- Note: The sketch (signature) of document C is small – ~ 100 bytes!

- We achieved our goal! We "compressed" long bit vectors into short signatures

Outline

Implementation Trick

- Permuting rows even once is prohibitive

Implementation Trick

- Permuting rows even once is prohibitive

Row hashing!

- Pick $K = 100$ hash functions k_i
- Ordering under k_i gives a random row permutation!

- Initialize all $sig(C)[i] = \infty$
- Scan rows looking for 1s
  - Suppose row j has 1 in column C
  - Then for each k_i:

Implementation Trick

- Permuting rows even once is prohibitive

Row hashing!

- Pick $K = 100$ hash functions k_i
- Ordering under k_i gives a random row permutation!

One-pass implementation

- For each column C and hash-function k_i keep a "slot" for the min-hash value

- Initialize all $sig(C)[i] = \infty$
- Scan rows looking for 1s
  - Suppose row j has 1 in column C
  - Then for each k_i:

If $k_i(j) < sig(C)[i]$, then $sig(C)[i] \leftarrow k_i(j)$

How to pick a random hash function $h(x)$?
Universal hashing:
$h_{a,b}(x) = ((a \cdot x + b) \bmod p) \bmod N$
where:
a, b... random integers
p... prime number ($p > N$)

Implementation Trick

- Permuting rows even once is prohibitive

Row hashing!

- Pick $K = 100$ hash functions k_i
- Ordering under k_i gives a random row permutation!

One-pass implementation

- For each column C and hash-function k_i keep a "slot" for the min-hash value

- Initialize all $sig(C)[i] = \infty$
- Scan rows looking for 1s
  - Suppose row j has 1 in column C
  - Then for each k_i:

If $k_i(j) < sig(C)[i]$, then $sig(C)[i] \leftarrow k_i(j)$

How to pick a random hash function $h(x)$?
Universal hashing:
$h_{a,b}(x) = ((a \cdot x + b) \bmod p) \bmod N$
where:
a, b... random integers
p... prime number ($p > N$)

Implementation Trick

- Permuting rows even once is prohibitive

Row hashing!

- Pick $K = 100$ hash functions k_i
- Ordering under k_i gives a random row permutation!

One-pass implementation

- For each column C and hash-function k_i keep a "slot" for the min-hash value
  - Initialize all $sig(C)[i] = \infty$
  - Scan rows looking for 1s
    - Suppose row j has 1 in column C
    - Then for each k_i:

If $k_i(j) < sig(C)[i]$, then $sig(C)[i] \leftarrow k_i(j)$

How to pick a random hash function $h(x)$?
Universal hashing:
$h_{a,b}(x) = ((a \cdot x + b) \bmod p) \bmod N$
where:
a, b... random integers
p... prime number ($p > N$)

Implementation Trick

- Permuting rows even once is prohibitive

Row hashing!

- Pick $K = 100$ hash functions k_i
- Ordering under k_i gives a random row permutation!

One-pass implementation

- For each column C and hash-function k_i keep a "slot" for the min-hash value
  - Initialize all $sig(C)[i] = \infty$
  - Scan rows looking for 1s
    - Suppose row j has 1 in column C
    - Then for each k_i:

If $k_i(j) < sig(C)[i]$, then $sig(C)[i] \leftarrow k_i(j)$

How to pick a random hash function $h(x)$?
Universal hashing:
$h_{a,b}(x) = ((a \cdot x + b) \bmod p) \bmod N$
where:
a, b... random integers
p... prime number ($p > N$)

Implementation Trick

- Permuting rows even once is prohibitive

Row hashing!

- Pick $K = 100$ hash functions k_i
- Ordering under k_i gives a random row permutation!

One-pass implementation

- For each column C and hash-function k_i keep a "slot" for the min-hash value
  - Initialize all $sig(C)[i] = \infty$
  - Scan rows looking for 1s
    - Suppose row j has 1 in column C
    - Then for each k_i:

If $k_i(j) < sig(C)[i]$, then $sig(C)[i] \leftarrow k_i(j)$

How to pick a random hash function $h(x)$?
Universal hashing:
$h_{a,b}(x) = ((a \cdot x + b) \bmod p) \bmod N$
where:
a, b... random integers
p... prime number ($p > N$)

Implementation Trick

- Permuting rows even once is prohibitive

Row hashing!

- Pick $K = 100$ hash functions k_i
- Ordering under k_i gives a random row permutation!

One-pass implementation

- For each column C and hash-function k_i keep a "slot" for the min-hash value
  - Initialize all $sig(C)[i] = \infty$
  - Scan rows looking for 1s
    - Suppose row j has 1 in column C
    - Then for each k_i:

If $k_i(j) < sig(C)[i]$, then $sig(C)[i] \leftarrow k_i(j)$

How to pick a random hash function $h(x)$?
Universal hashing:
$h_{a,b}(x) = ((a \cdot x + b) \bmod p) \bmod N$
where:
$a, b \ldots$ random integers
$p \ldots$ prime number ($p > N$)

Implementation Trick

- Permuting rows even once is prohibitive

Row hashing!

- Pick $K = 100$ hash functions k_i
- Ordering under k_i gives a random row permutation!

One-pass implementation

- For each column C and hash-function k_i keep a "slot" for the min-hash value

- Initialize all $sig(C)[i] = \infty$
- Scan rows looking for 1s
  - Suppose row j has 1 in column C
  - Then for each k_i:

If $k_i(j) < sig(C)[i]$, then $sig(C)[i] \leftarrow k_i(j)$

How to pick a random hash function $h(x)$?

Universal hashing:

$h_{a,b}(x) = ((a \cdot x + b) \bmod p) \bmod N$

where:

a, b... random integers

p... prime number ($p > N$)

Implementation Trick

- Permuting rows even once is prohibitive

Row hashing!

- Pick $K = 100$ hash functions k_i
- Ordering under k_i gives a random row permutation!

One-pass implementation

- For each column C and hash-function k_i keep a "slot" for the min-hash value

- Initialize all $sig(C)[i] = \infty$
- Scan rows looking for 1s
  - Suppose row j has 1 in column C
  - Then for each k_i:

If $k_i(j) < sig(C)[i]$, then $sig(C)[i] \leftarrow k_i(j)$

How to pick a random hash function $h(x)$?
Universal hashing:

$h_{a,b}(x) = ((a \cdot x + b) \bmod p) \bmod N$
where:
$a, b \ldots$ random integers
$p \ldots$ prime number $(p > N)$

Implementation Trick

- Permuting rows even once is prohibitive

Row hashing!

- Pick $K = 100$ hash functions k_i
- Ordering under k_i gives a random row permutation!

One-pass implementation

- For each column C and hash-function k_i keep a "slot" for the min-hash value

- Initialize all $sig(C)[i] = \infty$
- Scan rows looking for 1s
  - Suppose row j has 1 in column C
  - Then for each k_i:

If $k_i(j) < sig(C)[i]$, then $sig(C)[i] \leftarrow k_i(j)$

How to pick a random hash function $h(x)$?
Universal hashing:
$h_{a,b}(x) = ((a \cdot x + b) \bmod p) \bmod N$
where:
a, b... random integers
p... prime number ($p > N$)

Implementation Trick

- Permuting rows even once is prohibitive

Row hashing!

- Pick $K = 100$ hash functions k_i
- Ordering under k_i gives a random row permutation!

One-pass implementation

- For each column C and hash-function k_i keep a "slot" for the min-hash value

  - Initialize all $sig(C)[i] = \infty$
  - Scan rows looking for 1s
    - Suppose row j has 1 in column C
    - Then for each k_i:

If $k_i(j) < sig(C)[i]$, then $sig(C)[i] \leftarrow k_i(j)$

How to pick a random hash function $h(x)$?
Universal hashing:
$h_{a,b}(x) = ((a \cdot x + b) \bmod p) \bmod N$
where:
a, b... random integers
p... prime number ($p > N$)

Implementation Trick

- Permuting rows even once is prohibitive

Row hashing!

- Pick $K = 100$ hash functions k_i
- Ordering under k_i gives a random row permutation!

One-pass implementation

- For each column C and hash-function k_i keep a "slot" for the min-hash value

- Initialize all $sig(C)[i] = \infty$
- Scan rows looking for 1s
  - Suppose row j has 1 in column C
  - Then for each k_i:

If $k_i(j) < sig(C)[i]$, then $sig(C)[i] \leftarrow k_i(j)$

How to pick a random hash function $h(x)$?
Universal hashing:
$h_{a,b}(x) = ((a \cdot x + b) \bmod p) \bmod N$
where:
a, b... random integers
p... prime number $(p > N)$

Implementation Trick

- Permuting rows even once is prohibitive

Row hashing!

- Pick $K = 100$ hash functions k_i
- Ordering under k_i gives a random row permutation!

One-pass implementation

- For each column C and hash-function k_i keep a "slot" for the min-hash value

- Initialize all $sig(C)[i] = \infty$
- Scan rows looking for 1s
  - Suppose row j has 1 in column C
  - Then for each k_i:

If $k_i(j) < sig(C)[i]$, then $sig(C)[i] \leftarrow k_i(j)$

How to pick a random hash function $h(x)$?
Universal hashing:
$h_{a,b}(x) = ((a \cdot x + b) \bmod p) \bmod N$
where:
a, b... random integers
p... prime number $(p > N)$

Outline

Trying to define LSH

2	1	4	1
1	2	1	2
2	1	2	1

Goal

- Find documents with Jaccard similarity at least s (for some similarity threshold, e.g., $s = 0.8$)

Trying to define LSH

2	1	4	1
1	2	1	2
2	1	2	1

Goal

- Find documents with Jaccard similarity at least s (for some similarity threshold, e.g., $s = 0.8$)

LSH – General idea

- Use a function $f(x, y)$ that tells whether x and y is a candidate pair: a pair of elements whose similarity must be evaluated.

Trying to define LSH

2	1	4	1
1	2	1	2
2	1	2	1

Goal

- Find documents with Jaccard similarity at least s (for some similarity threshold, e.g., $s = 0.8$)

LSH – General idea

- Use a function $f(x, y)$ that tells whether x and y is a candidate pair: a pair of elements whose similarity must be evaluated.

For MinHash matrices

- Hash columns of signature matrix M to many buckets.
- Each pair of documents that hashes into the same bucket is a candidate pair.

Trying to define LSH

2	1	4	1
1	2	1	2
2	1	2	1

Goal

- Find documents with Jaccard similarity at least s (for some similarity threshold, e.g., $s = 0.8$)

LSH – General idea

- Use a function $f(x, y)$ that tells whether x and y is a candidate pair: a pair of elements whose similarity must be evaluated.

For MinHash matrices

- Hash columns of signature matrix M to many buckets.
- Each pair of documents that hashes into the same bucket is a candidate pair.

Trying to define LSH

2	1	4	1
1	2	1	2
2	1	2	1

Goal

- Find documents with Jaccard similarity at least s (for some similarity threshold, e.g., $s = 0.8$)

LSH – General idea

- Use a function $f(x, y)$ that tells whether x and y is a candidate pair: a pair of elements whose similarity must be evaluated.

For MinHash matrices

- Hash columns of signature matrix M to many buckets.
- Each pair of documents that hashes into the same bucket is a candidate pair.

Candidates from Minhash

2	1	4	1
1	2	1	2
2	1	2	1

- Pick a similarity threshold s $(0 < s < 1)$.

Candidates from Minhash

2	1	4	1
1	2	1	2
2	1	2	1

- Pick a similarity threshold s $(0 < s < 1)$.

Candidate pair

- Columns x and y of M are a candidate pair if their signatures agree on at least fraction s of their rows:
  - $M(i,x) = M(i,y)$ for at least fraction s of values of i
    - We expect documents x and y to have the same (Jaccard) similarity as is the similarity of their signatures

Candidates from Minhash

2	1	4	1
1	2	1	2
2	1	2	1

- Pick a similarity threshold s $(0 < s < 1)$.

Candidate pair

- Columns x and y of M are a candidate pair if their signatures agree on at least fraction s of their rows:
  - $M(i,x) = M(i,y)$ for at least fraction s of values of i
    - We expect documents x and y to have the same (Jaccard) similarity as is the similarity of their signatures

Candidates from Minhash

2	1	4	1
1	2	1	2
2	1	2	1

- Pick a similarity threshold s $(0 < s < 1)$.

Candidate pair

- Columns x and y of M are a candidate pair if their signatures agree on at least fraction s of their rows:
  - $M(i, x) = M(i, y)$ for at least fraction s of values of i
    - We expect documents x and y to have the same (Jaccard) similarity as is the similarity of their signatures

LSH for Minhash

2	1	4	1
1	2	1	2
2	1	2	1

Big idea

- Hash columns of signature matrix M several times

Likely to hash

- Arrange that (only) similar columns are likely to hash to the same bucket, with high probability

Candidate pairs

- Candidate pairs are those that hash to the same bucket

LSH for Minhash

2	1	4	1
1	2	1	2
2	1	2	1

Big idea

- Hash columns of signature matrix M several times

Likely to hash

- Arrange that (only) similar columns are likely to hash to the same bucket, with high probability

Candidate pairs

- Candidate pairs are those that hash to the same bucket

LSH for Minhash

2	1	4	1
1	2	1	2
2	1	2	1

Big idea

- Hash columns of signature matrix M several times

Likely to hash

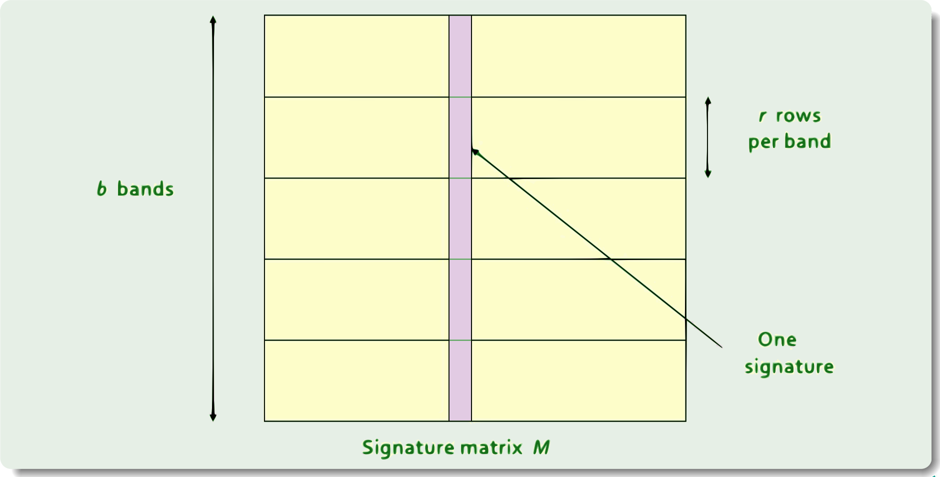
- Arrange that (only) similar columns are likely to hash to the same bucket, with high probability

Candidate pairs

- Candidate pairs are those that hash to the same bucket

Partition M into b Bands

2	1	4	1
1	2	1	2
2	1	2	1



Partition M into b Bands

Divide Matrix

- Divide matrix M into b bands of r rows.
- For each band, hash its portion of each column to a hash table with k buckets.
  - Make k as large as possible.

Partition M into b Bands

Divide Matrix

- Divide matrix M into b bands of r rows.
- For each band, hash its portion of each column to a hash table with k buckets.
    - Make k as large as possible.

Candidate

- Candidate column pairs are those that hash to the same bucket for ≥ 1 bands.

Partition M into b Bands

Divide Matrix

- Divide matrix M into b bands of r rows.
- For each band, hash its portion of each column to a hash table with k buckets.
  - Make k as large as possible.

Candidate

- Candidate column pairs are those that hash to the same bucket for ≥ 1 bands.

Catch most similar pairs

- Tune b and r to catch most similar pairs, but few non-similar pairs.

Partition M into b Bands

Divide Matrix

- Divide matrix M into b bands of r rows.
- For each band, hash its portion of each column to a hash table with k buckets.
  - Make k as large as possible.

Candidate

- Candidate column pairs are those that hash to the same bucket for ≥ 1 bands.

Catch most similar pairs

- Tune b and r to catch most similar pairs, but few non-similar pairs.

Partition M into b Bands

Divide Matrix

- Divide matrix M into b bands of r rows.
- For each band, hash its portion of each column to a hash table with k buckets.
    - Make k as large as possible.

Candidate

- Candidate column pairs are those that hash to the same bucket for ≥ 1 bands.

Catch most similar pairs

- Tune b and r to catch most similar pairs, but few non-similar pairs.

Partition M into b Bands

Divide Matrix

- Divide matrix M into b bands of r rows.
- For each band, hash its portion of each column to a hash table with k buckets.
  - Make k as large as possible.

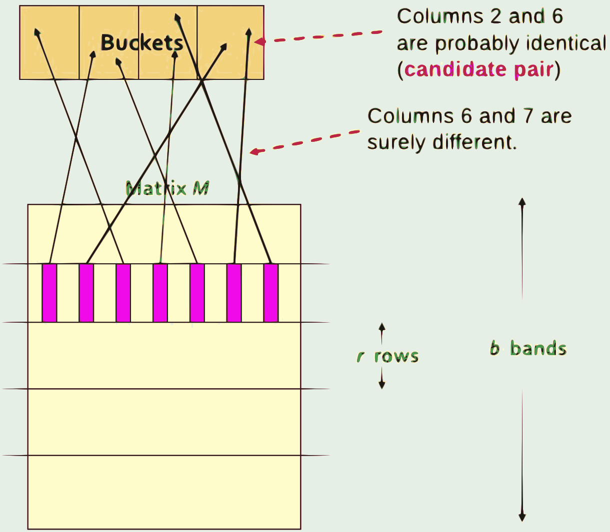
Candidate

- Candidate column pairs are those that hash to the same bucket for ≥ 1 bands.

Catch most similar pairs

- Tune b and r to catch most similar pairs, but few non-similar pairs.

Hashing Bands



Simplifying Assumption

Identical

- There are enough buckets that columns are unlikely to hash to the same bucket unless they are identical in a particular band

Same bucket

- Then, we assume that "same bucket" means "identical in that band"

Not for correctness

- Assumption needed only to simplify analysis, not for the correctness of algorithm

Simplifying Assumption

Identical

- There are enough buckets that columns are unlikely to hash to the same bucket unless they are identical in a particular band

Same bucket

- Then, we assume that “same bucket” means “identical in that band”

Not for correctness

- Assumption needed only to simplify analysis, not for the correctness of algorithm

Simplifying Assumption

Identical

- There are enough buckets that columns are unlikely to hash to the same bucket unless they are identical in a particular band

Same bucket

- Then, we assume that "same bucket" means "identical in that band"

Not for correctness

- Assumption needed only to simplify analysis, not for the correctness of algorithm

Example of Bands

2	1	4	1
1	2	1	2
2	1	2	1

Assume the following case

- Suppose $100,000$ columns of M $(100k$ docs)
- Signatures of 100 integers (rows)
- Therefore, signatures take $40Mb$
- Choose $b = 20$ bands of $r = 5$ integers/band

Example of Bands

2	1	4	1
1	2	1	2
2	1	2	1

Assume the following case

- Suppose $100,000$ columns of M ($100k$ docs)
- Signatures of 100 integers (rows)
- Therefore, signatures take $40Mb$
- Choose $b = 20$ bands of $r = 5$ integers/band

Goal

- Find pairs of documents that are at least $s = 0.8$ similar

Example of Bands

2	1	4	1
1	2	1	2
2	1	2	1

Assume the following case

- Suppose $100,000$ columns of M ($100k$ docs)
- Signatures of 100 integers (rows)
- Therefore, signatures take $40Mb$
- Choose $b = 20$ bands of $r = 5$ integers/band

Goal

- Find pairs of documents that are at least $s = 0.8$ similar

Example of Bands

2	1	4	1
1	2	1	2
2	1	2	1

Assume the following case

- Suppose $100,000$ columns of M ($100k$ docs)
- Signatures of 100 integers (rows)
- Therefore, signatures take $40Mb$
- Choose $b = 20$ bands of $r = 5$ integers/band

Goal

- Find pairs of documents that are at least $s = 0.8$ similar

Example of Bands

2	1	4	1
1	2	1	2
2	1	2	1

Assume the following case

- Suppose $100,000$ columns of M ($100k$ docs)
- Signatures of 100 integers (rows)
- Therefore, signatures take $40Mb$
- Choose $b = 20$ bands of $r = 5$ integers/band

Goal

- Find pairs of documents that are at least $s = 0.8$ similar

Now, if C_1,C_2 are 80% Similar

2	1	4	1
1	2	1	2

Assume

- Find pairs of $\geq s = 0.8$ similarity, set $b = 20$, $r = 5$
- Assume: $sim(C_1, C_2) = 0.8$
  - Since $sim(C_1, C_2) \geq s$, we want C_1, C_2 to be a candidate pair: We want them to hash to at least 1 common bucket (at least one band is identical)

Now, if C_1,C_2 are 80% Similar

2	1	4	1
1	2	1	2

Assume

- Find pairs of $\geq s = 0.8$ similarity, set $b = 20$, $r = 5$
- Assume: $sim(C_1, C_2) = 0.8$
  - Since $sim(C_1, C_2) \geq s$, we want C_1, C_2 to be a candidate pair: We want them to hash to at least 1 common bucket (at least one band is identical)

In one particular band

- Probability C_1, C_2 identical in one particular band: $(0.8)^5 = 0.328$

  - i.e., about 1/3000th of the 80%-similar column pairs are false negatives (we miss them)
  - We would find 99.965% pairs of truly similar documents

Now, if C_1,C_2 are 80% Similar

2	1	4	1
1	2	1	2

Assume

- Find pairs of $\geq s = 0.8$ similarity, set $b = 20$, $r = 5$
- Assume: $sim(C_1, C_2) = 0.8$
  - Since $sim(C_1, C_2) \geq s$, we want C_1, C_2 to be a candidate pair: We want them to hash to at least 1 common bucket (at least one band is identical)

In one particular band

- Probability C_1, C_2 identical in one particular band: $(0.8)^5 = 0.328$

What is the Probability of not being similar at all?

- Probability C_1, C_2 are not similar in all of the 20 bands: $(1 - 0.328)^{20} = 0.00035$
  - i.e., about $1/3000$th of the 80%-similar column pairs are false negatives (we miss them)
  - We would find 99.965% pairs of truly similar documents

Now, if C_1,C_2 are 80% Similar

2	1	4	1
1	2	1	2

Assume

- Find pairs of $\geq s = 0.8$ similarity, set $b = 20$, $r = 5$
- Assume: $sim(C_1, C_2) = 0.8$
  - Since $sim(C_1, C_2) \geq s$, we want C_1, C_2 to be a candidate pair: We want them to hash to at least 1 common bucket (at least one band is identical)

In one particular band

- Probability C_1, C_2 identical in one particular band: $(0.8)^5 = 0.328$

What is the Probability of not being similar at all?

- Probability C_1, C_2 are not similar in all of the 20 bands: $(1 - 0.328)^{20} = 0.00035$
  - i.e., about $1/3000$th of the 80%-similar column pairs are false negatives (we miss them)
  - We would find 99.965% pairs of truly similar documents

Now, if C_1,C_2 are 80% Similar

2	1	4	1
1	2	1	2

Assume

- Find pairs of $\geq s = 0.8$ similarity, set $b = 20$, $r = 5$
- Assume: $sim(C_1, C_2) = 0.8$
  - Since $sim(C_1, C_2) \geq s$, we want C_1, C_2 to be a candidate pair: We want them to hash to at least 1 common bucket (at least one band is identical)

In one particular band

- Probability C_1, C_2 identical in one particular band: $(0.8)^5 = 0.328$

What is the Probability of not being similar at all?

- Probability C_1, C_2 are not similar in all of the 20 bands: $(1 - 0.328)^{20} = 0.00035$
  - i.e., about 1/3000th of the 80%-similar column pairs are false negatives (we miss them)
  - We would find 99.965% pairs of truly similar documents

Now, if C_1,C_2 are 80% Similar

2	1	4	1
1	2	1	2

Assume

- Find pairs of $\geq s = 0.8$ similarity, set $b = 20$, $r = 5$
- Assume: $sim(C_1, C_2) = 0.8$
  - Since $sim(C_1, C_2) \geq s$, we want C_1, C_2 to be a candidate pair: We want them to hash to at least 1 common bucket (at least one band is identical)

In one particular band

- Probability C_1, C_2 identical in one particular band: $(0.8)^5 = 0.328$

What is the Probability of not being similar at all?

- Probability C_1, C_2 are not similar in all of the 20 bands: $(1 - 0.328)^{20} = 0.00035$
  - i.e., about $1/3000$th of the 80%-similar column pairs are false negatives (we miss them)
  - We would find 99.965% pairs of truly similar documents

Now, if C_1,C_2 are 80% Similar

2	1	4	1
1	2	1	2

Assume

- Find pairs of $\geq s = 0.8$ similarity, set $b = 20$, $r = 5$
- Assume: $sim(C_1, C_2) = 0.8$
  - Since $sim(C_1, C_2) \geq s$, we want C_1, C_2 to be a candidate pair: We want them to hash to at least 1 common bucket (at least one band is identical)

In one particular band

- Probability C_1, C_2 identical in one particular band: $(0.8)^5 = 0.328$

What is the Probability of not being similar at all?

- Probability C_1, C_2 are not similar in all of the 20 bands: $(1 - 0.328)^{20} = 0.00035$
  - i.e., about $1/3000$th of the 80%-similar column pairs are false negatives (we miss them)
  - We would find 99.965% pairs of truly similar documents

C_1,C_2 are 30% Similar

2	1	4	1
1	2	1	2

Assume

- Find pairs of $\geq s = 0.8$ similarity, set $b = 20$, $r = 5$
- Assume: $sim(C_1, C_2) = 0.3$
  - Since $sim(C_1, C_2) < s$ we want C_1, C_2 to hash to NO common buckets (all bands should be different).

C_1,C_2 are 30% Similar

2	1	4	1
1	2	1	2

Assume

- Find pairs of $\geq s = 0.8$ similarity, set $b = 20$, $r = 5$
- Assume: $sim(C_1, C_2) = 0.3$
  - Since $sim(C_1, C_2) < s$ we want C_1, C_2 to hash to NO common buckets (all bands should be different).

Identical in one particular band

- Probability C_1, C_2 identical in one particular band: $(0.3)^5 = 0.00243$.

C_1,C_2 are 30% Similar

2	1	4	1
1	2	1	2

Assume

- Find pairs of $\geq s = 0.8$ similarity, set $b = 20$, $r = 5$
- Assume: $sim(C_1, C_2) = 0.3$
  - Since $sim(C_1, C_2) < s$we want C_1, C_2 to hash to NO common buckets (all bands should be different).

Identical in one particular band

- Probability C_1, C_2 identical in one particular band: $(0.3)^5 = 0.00243$.

Properties

- Probability C_1, C_2 identical in at least 1 of 20 bands: $1 - (1 - 0.00243)20 = 0.0474$.
  - In other words, approximately 4.74% pairs of docs with similarity 0.3% end up becoming candidate pairs.
    - They are false positives since we will have to examine them (they are candidate pairs) but then it will turn out their similarity is below threshold s.

C_1,C_2 are 30% Similar

2	1	4	1
1	2	1	2

Assume

- Find pairs of $\geq s = 0.8$ similarity, set $b = 20$, $r = 5$
- Assume: $sim(C_1, C_2) = 0.3$
  - Since $sim(C_1, C_2) < s$we want C_1, C_2 to hash to NO common buckets (all bands should be different).

Identical in one particular band

- Probability C_1, C_2 identical in one particular band: $(0.3)^5 = 0.00243$.

Properties

- Probability C_1, C_2 identical in at least 1 of 20 bands: $1 - (1 - 0.00243)20 = 0.0474$.
  - In other words, approximately 4.74% pairs of docs with similarity 0.3% end up becoming candidate pairs.
    - They are false positives since we will have to examine them (they are candidate pairs) but then it will turn out their similarity is below threshold s.

C_1,C_2 are 30% Similar

2	1	4	1
1	2	1	2

Assume

- Find pairs of $\geq s = 0.8$ similarity, set $b = 20$, $r = 5$
- Assume: $sim(C_1, C_2) = 0.3$
  - Since $sim(C_1, C_2) < s$we want C_1, C_2 to hash to NO common buckets (all bands should be different).

Identical in one particular band

- Probability C_1, C_2 identical in one particular band: $(0.3)^5 = 0.00243$.

Properties

- Probability C_1, C_2 identical in at least 1 of 20 bands: $1 - (1 - 0.00243)20 = 0.0474$.
  - In other words, approximately 4.74% pairs of docs with similarity 0.3% end up becoming candidate pairs.
    - They are false positives since we will have to examine them (they are candidate pairs) but then it will turn out their similarity is below threshold s.

C_1,C_2 are 30% Similar

2	1	4	1
1	2	1	2

Assume

- Find pairs of $\geq s = 0.8$ similarity, set $b = 20$, $r = 5$
- Assume: $sim(C_1, C_2) = 0.3$
  - Since $sim(C_1, C_2) < s$we want C_1, C_2 to hash to NO common buckets (all bands should be different).

Identical in one particular band

- Probability C_1, C_2 identical in one particular band: $(0.3)^5 = 0.00243$.

Properties

- Probability C_1, C_2 identical in at least 1 of 20 bands:
  $1 - (1 - 0.00243)20 = 0.0474$.
  - In other words, approximately 4.74% pairs of docs with similarity 0.3% end up becoming candidate pairs.
    - They are false positives since we will have to examine them (they are candidate pairs) but then it will turn out their similarity is below threshold s.

C_1,C_2 are 30% Similar

2	1	4	1
1	2	1	2

Assume

- Find pairs of $\geq s = 0.8$ similarity, set $b = 20$, $r = 5$
- Assume: $sim(C_1, C_2) = 0.3$
  - Since $sim(C_1, C_2) < s$we want C_1, C_2 to hash to NO common buckets (all bands should be different).

Identical in one particular band

- Probability C_1, C_2 identical in one particular band: $(0.3)^5 = 0.00243$.

Properties

- Probability C_1, C_2 identical in at least 1 of 20 bands: $1 - (1 - 0.00243)20 = 0.0474$.
  - In other words, approximately 4.74% pairs of docs with similarity 0.3% end up becoming candidate pairs.
    - ★ They are false positives since we will have to examine them (they are candidate pairs) but then it will turn out their similarity is below threshold s.

LSH Involves a Tradeoff

2	1	4	1
1	2	1	2
2	1	2	1

You need to pick

- The number of minhashes (rows of M).
- The number of bands b.
- The number of rows r per band to balance false positives/negatives.

LSH Involves a Tradeoff

2	1	4	1
1	2	1	2
2	1	2	1

You need to pick

- The number of minhashes (rows of M).
- The number of bands b.
- The number of rows r per band to balance false positives/negatives.

Example

- if we had only 15 bands of 5 rows, the number of false positives would go down, but the number of false negatives would go up

cinvestav

LSH Involves a Tradeoff

2	1	4	1
1	2	1	2
2	1	2	1

You need to pick

- The number of minhashes (rows of M).
- The number of bands b.
- The number of rows r per band to balance false positives/negatives.

Example

- if we had only 15 bands of 5 rows, the number of false positives would go down, but the number of false negatives would go up

LSH Involves a Tradeoff

2	1	4	1
1	2	1	2
2	1	2	1

You need to pick

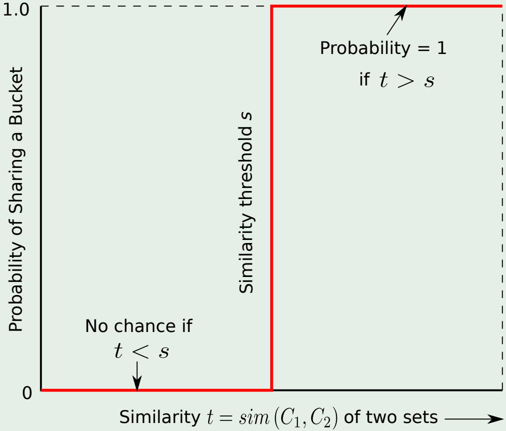
- The number of minhashes (rows of M).
- The number of bands b.
- The number of rows r per band to balance false positives/negatives.

Example

- if we had only 15 bands of 5 rows, the number of false positives would go down, but the number of false negatives would go up

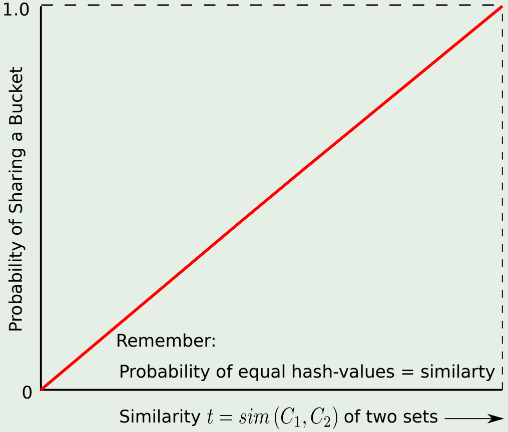
Analysis of LSH - What We Want

The Ideal detection of similar objects



What 1 Band of 1 Row Gives You

Not so great



Given that probability of two documents aggree in a row is s

We can calculate the probability that these documents become a candidate pair as follows

1. The probability that the signatures agree in all rows of one particular band is s^r.
2. The probability that the signatures disagree in at least one row of a particular band is $1 - s^r$.
3. The probability that the signatures disagree in at least one row of each of the bands is $(1 - s^r)^b$.
4. The probability that the signatures agree in all the rows of at least one band, and therefore become a candidate pair, is $1 - (1 - s^r)^b$.

Given that probability of two documents aggree in a row is s

We can calculate the probability that these documents become a candidate pair as follows

1. The probability that the signatures agree in all rows of one particular band is s^r.
2. The probability that the signatures disagree in at least one row of a particular band is $1 - s^r$.
3. The probability that the signatures disagree in at least one row of each of the bands is $(1 - s^r)^b$.
4. The probability that the signatures agree in all the rows of at least one band, and therefore become a candidate pair, is $1 - (1 - s^r)^b$.

Given that probability of two documents aggree in a row is s

We can calculate the probability that these documents become a candidate pair as follows

1. The probability that the signatures agree in all rows of one particular band is s^r.
2. The probability that the signatures disagree in at least one row of a particular band is $1 - s^r$.
3. The probability that the signatures disagree in at least one row of each of the bands is $(1 - s^r)^b$.
4. The probability that the signatures agree in all the rows of at least one band, and therefore become a candidate pair, is $1 - (1 - s^r)^b$.

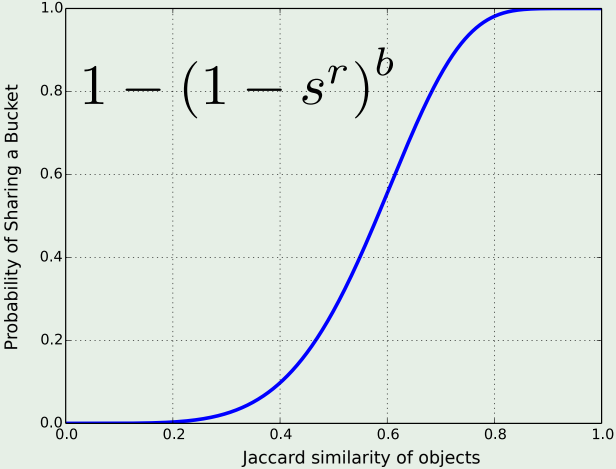
Given that probability of two documents aggree in a row is s

We can calculate the probability that these documents become a candidate pair as follows

1. The probability that the signatures agree in all rows of one particular band is s^r.
2. The probability that the signatures disagree in at least one row of a particular band is $1 - s^r$.
3. The probability that the signatures disagree in at least one row of each of the bands is $(1 - s^r)^b$.
4. The probability that the signatures agree in all the rows of at least one band, and therefore become a candidate pair, is $1 - (1 - s^r)^b$.

If you fix r and b

Something Notable



Example: $b = 20; r = 5$

Given

- Similarity threshold s

Similarity threshold s Prob. that at least 1 band is identical

s	$1-(1-s^r)^b$
.2	0.006
.3	0.047
.4	0.186
.5	0.470
.6	0.802
.7	0.975
.8	0.9996

Example: $b = 20$; $r = 5$

Given

- Similarity threshold s

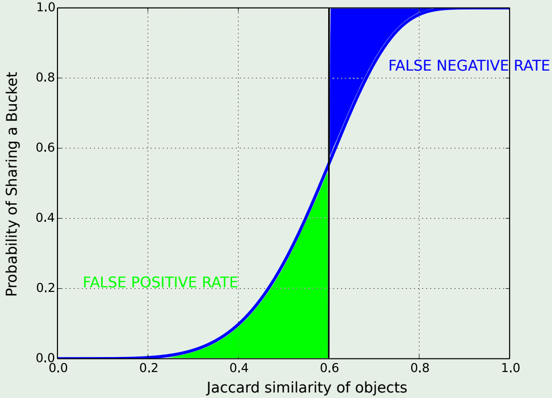
Similarity threshold s Prob. that at least 1 band is identical

s	$1-(1-s^r)^b$
.2	0.006
.3	0.047
.4	0.186
.5	0.470
.6	0.802
.7	0.975
.8	0.9996

Picking r and b: The S-curve

Picking r and b to get the best S-curve

- 50 hash-functions $(r = 5, b = 10)$



LSH Summary

Tune M, b, r

- Tune M, b, r to get almost all pairs with similar signatures, but eliminate most pairs that do not have similar signatures

Check in main memory

- Check in main memory that candidate pairs really do have similar signatures

Optional

- In another pass through data, check that the remaining candidate pairs really represent similar documents

LSH Summary

Tune M, b, r

- Tune M, b, r to get almost all pairs with similar signatures, but eliminate most pairs that do not have similar signatures

Check in main memory

- Check in main memory that candidate pairs really do have similar signatures

Optional

- In another pass through data, check that the remaining candidate pairs really represent similar documents

LSH Summary

Tune M, b, r

- Tune M, b, r to get almost all pairs with similar signatures, but eliminate most pairs that do not have similar signatures

Check in main memory

- Check in main memory that candidate pairs really do have similar signatures

Optional

- In another pass through data, check that the remaining candidate pairs really represent similar documents

Summary: 3 steps

Shingling

- Convert documents to sets
  - We used hashing to assign each shingle an ID Min-hashing: Convert large sets to short

Summary: 3 steps

Shingling

- Convert documents to sets
  - We used hashing to assign each shingle an ID Min-hashing: Convert large sets to short

Min-hashing

- Convert large sets to short signatures, while preserving similarity.
  - We used similarity preserving hashing to generate signatures with property $Pr[h_\pi(C_1) = h_\pi(C_2)] = sim(C_1, C_2)$.
  - We used hashing to get around generating random permutations.

  - We used hashing to find candidate pairs of similarity $\geq s$

Summary: 3 steps

Shingling

- Convert documents to sets
  - We used hashing to assign each shingle an ID Min-hashing: Convert large sets to short

Min-hashing

- Convert large sets to short signatures, while preserving similarity.
  - We used similarity preserving hashing to generate signatures with property $Pr[h_\pi(C_1) = h_\pi(C_2)] = sim(C_1, C_2)$.
  - We used hashing to get around generating random permutations.

Locality-Sensitive Hashing

- Focus on pairs of signatures likely to be from similar documents.
  - We used hashing to find candidate pairs of similarity $\geq s$

Summary: 3 steps

Shingling

- Convert documents to sets
  - We used hashing to assign each shingle an ID Min-hashing: Convert large sets to short

Min-hashing

- Convert large sets to short signatures, while preserving similarity.
  - We used similarity preserving hashing to generate signatures with property $Pr[h_\pi(C_1) = h_\pi(C_2)] = sim(C_1, C_2)$.
  - We used hashing to get around generating random permutations.

Locality-Sensitive Hashing

- Focus on pairs of signatures likely to be from similar documents.
  - We used hashing to find candidate pairs of similarity $\geq s$

Summary: 3 steps

Shingling

- Convert documents to sets
  - We used hashing to assign each shingle an ID Min-hashing: Convert large sets to short

Min-hashing

- Convert large sets to short signatures, while preserving similarity.
  - We used similarity preserving hashing to generate signatures with property $Pr[h_\pi(C_1) = h_\pi(C_2)] = sim(C_1, C_2)$.
  - We used hashing to get around generating random permutations.

Locality-Sensitive Hashing

- Focus on pairs of signatures likely to be from similar documents.
  - We used hashing to find candidate pairs of similarity $\geq s$

Summary: 3 steps

Shingling

- Convert documents to sets
  - We used hashing to assign each shingle an ID Min-hashing: Convert large sets to short

Min-hashing

- Convert large sets to short signatures, while preserving similarity.
  - We used similarity preserving hashing to generate signatures with property $Pr[h_\pi(C_1) = h_\pi(C_2)] = sim(C_1, C_2)$.
  - We used hashing to get around generating random permutations.

Locality-Sensitive Hashing

- Focus on pairs of signatures likely to be from similar documents.
  - We used hashing to find candidate pairs of similarity $\geq s$

Summary: 3 steps

Shingling

- Convert documents to sets
  - We used hashing to assign each shingle an ID Min-hashing: Convert large sets to short

Min-hashing

- Convert large sets to short signatures, while preserving similarity.
  - We used similarity preserving hashing to generate signatures with property $Pr[h_\pi(C_1) = h_\pi(C_2)] = sim(C_1, C_2)$.
  - We used hashing to get around generating random permutations.

Locality-Sensitive Hashing

- Focus on pairs of signatures likely to be from similar documents.
  - We used hashing to find candidate pairs of similarity $\geq s$