Introduction to Algorithms

Locality Sensitive Hashing

Andres Mendez-Vazquez

September 27, 2020

Outline

Outline

We have the following problem [1, 2]

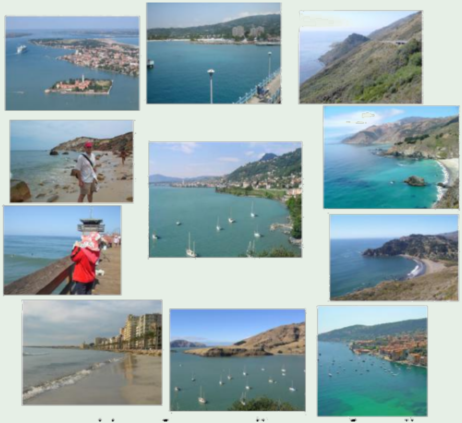
We have an image as a Query...

Image Retrieval

We want to ask a large database of images for the most similar

Scene Completion Problem

Then, we want to get the possible nearest elements to it



Outline

A Common Problem

Problems

- Many problems can be expressed as finding "similar" sets:
  - Basically... Finding 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...

Others

- Images with similar features...
- Users who visited the similar websites

Examples

Pages with similar words

- For duplicate detection, classification by topic...

Customers who purchased similar products

- Products with similar customer sets...

Others

- Images with similar features...
- Users who visited the similar websites

Examples

Pages with similar words

- For duplicate detection, classification by topic...

Customers who purchased similar products

- Products with similar customer sets...

Others

- Images with similar features...
- Users who visited the similar websites

Outline

Given a database D with n items [3]

Define the following query $NN(q, r, c)$ (Nearest Neighborhood (NN))

- Given query q and two parameters $r \geq 0$ and $c \geq 1$.

If there exists $x \in D$ such that $D(q, x) \leq r$

- Then report some $y \in D$ such that $D(y, r) \leq cr$

If there is no $x \in D$ such that $D(q, x) \leq cr$

- Report Failure...

Given a database D with n items [3]

Define the following query $NN(q, r, c)$ (Nearest Neighborhood (NN))

- Given query q and two parameters $r \geq 0$ and $c \geq 1$.

If there exists $x \in D$ such that $D(q, x) \leq r$

- Then report some $y \in D$ such that $D(y, r) \leq cr$

If there is no $x \in D$ such that $D(q, x) \leq cr$

- Report Failure...

Given a database D with n items [3]

Define the following query $NN(q, r, c)$ (Nearest Neighborhood (NN))

- Given query q and two parameters $r \geq 0$ and $c \geq 1$.

If there exists $x \in D$ such that $D(q, x) \leq r$

- Then report some $y \in D$ such that $D(y, r) \leq cr$

If there is no $x \in D$ such that $D(q, x) \leq cr$

- Report Failure...

Then, we have that

When $c = 1$

- The query is precise

If there is a point at distance at most r from q, the algorithm reports such a point

- else it reports failure

Therefore

- We can now perform binary search over r and compute the nearest neighbor to an arbitrarily good approximation.

Then, we have that

When $c = 1$

- The query is precise

If there is a point at distance at most r from q, the algorithm reports such a point

- else it reports failure

Therefore

- We can now perform binary search over r and compute the nearest neighbor to an arbitrarily good approximation.

Then, we have that

When $c = 1$

- The query is precise

If there is a point at distance at most r from q, the algorithm reports such a point

- else it reports failure

Therefore

- We can now perform binary search over r and compute the nearest neighbor to an arbitrarily good approximation.

When c is much larger than 1

Say 5

- the algorithm is good for distinguishing two cases

If all points in D are very far away from q

- At distance at least $cr = 5r$, the algorithm correctly reports failure.

If there is a point at most distance r from q

- The algorithm will report some point, but this could be a point further away at most $5r$.
  - Then, we need to do another test so look for equality

When c is much larger than 1

Say 5

- the algorithm is good for distinguishing two cases

If all points in D are very far away from q

- At distance at least $cr = 5r$, the algorithm correctly reports failure.

If there is a point at most distance r from q

- The algorithm will report some point, but this could be a point further away at most $5r$.
  - Then, we need to do another test so look for equality

When c is much larger than 1

Say 5

- the algorithm is good for distinguishing two cases

If all points in D are very far away from q

- At distance at least $cr = 5r$, the algorithm correctly reports failure.

If there is a point at most distance r from q

- The algorithm will report some point, but this could be a point further away at most $5r$.
  - Then, we need to do another test so look for equality

Therefore

At the paper [3] by the legendary Rajeev Motwani

- They described how to solve the Approximate Near-Neighbor Problem using Point Location in Equal Balls (PLEB) defined as

$$B(x, r) = \{p | d(x, p) \leq r\}$$

Point Location in Equal Balls

- Given n radius r balls centered at $C = \{c_1, c_2, ..., c_n\}$ in $\mathbb{R}^d$ then devise a data structure which for any query point $q \in \mathbb{R}^d$ does the following
  - If there exists $c_i \in C$ such that $q \in B(c_i, r)$ then return c_i else return NO

Note: Here n elements are the pre-processed elements at the database.

Therefore

At the paper [3] by the legendary Rajeev Motwani

- They described how to solve the Approximate Near-Neighbor Problem using Point Location in Equal Balls (PLEB) defined as

$$B(x, r) = \{p | d(x, p) \leq r\}$$

Point Location in Equal Balls

- Given n radius r balls centered at $C = \{c_1, c_2, ..., c_n\}$ in $\mathbb{R}^d$ then devise a data structure which for any query point $q \in \mathbb{R}^d$ does the following
  - If there exists $c_i \in C$ such that $q \in B(c_i, r)$ then return c_i else return NO

Note: Here n elements are the pre-processed elements at the database.

Therefore

The Question Arises

- Given $B(c_i, r)$, How do we define a distance d?

$$B(x, r) = \{p | d(x, p) \leq r\}$$

Outline

Distance Measures

Goal

- Find near-neighbors in **high-dimensional** 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-dimensional** 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

Jaccard Similarity

Jaccard Similarity/Distance of two sets is [4]

$$\frac{\text{size of their intersection}}{\text{the size of their union}}$$

This allows to define the function as

$$sim\left(C_1, C_2\right) = \frac{|C_1 \cap C_2|}{|C_1 \cup C_2|}$$

It allows to define a distance

$$d\left(C_1, C_2\right) = 1 - \frac{|C_1 \cap C_2|}{|C_1 \cup C_2|}$$

Jaccard Similarity

Jaccard Similarity/Distance of two sets is [4]

$$\frac{\text{size of their intersection}}{\text{the size of their union}}$$

This allows to define the function as

$$sim\left(C_1, C_2\right) = \frac{|C_1 \cap C_2|}{|C_1 \cup C_2|}$$

It allows to define a distance

$$d\left(C_1, C_2\right) = 1 - \frac{|C_1 \cap C_2|}{|C_1 \cup C_2|}$$

Jaccard Similarity

Jaccard Similarity/Distance of two sets is [4]

$$\frac{\text{size of their intersection}}{\text{the size of their union}}$$

This allows to define the function as

$$sim\left(C_1, C_2\right) = \frac{|C_1 \cap C_2|}{|C_1 \cup C_2|}$$

It allows to define a distance

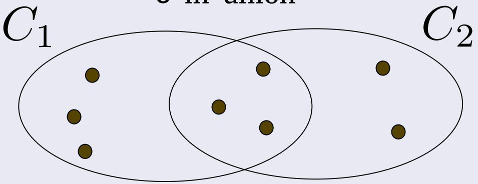
$$d\left(C_1, C_2\right) = 1 - \frac{|C_1 \cap C_2|}{|C_1 \cup C_2|}$$

Example

Jaccard Similarity between sets

3 in intersection

8 in union



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

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

Now, a concept on representations

Remember the Radix representation of a number

$$1011 = 1 \times 2^0 + 1 \times 2^1 + 0 \times 2^2 + 1 \times 2^3$$

This is a positional representation of a number

- Each block is based in the position of a representative belonging to the set $\{0, 1\}$:

Then, we have

- This idea of representative... SHINGLES...
  - A a rectangular tile of asphalt composite, wood, metal, or slate used on walls or roofs.

Now, a concept on representations

Remember the Radix representation of a number

$$1011 = 1 \times 2^0 + 1 \times 2^1 + 0 \times 2^2 + 1 \times 2^3$$

This is a positional representation of a number

- Each block is based in the position of a representative belonging to the set $\{0, 1\}$:

Then, we have

- This idea of representative... SHINGLES...
  - A a rectangular tile of asphalt composite, wood, metal, or slate used on walls or roofs.

Now, a concept on representations

Remember the Radix representation of a number

$$1011 = 1 \times 2^0 + 1 \times 2^1 + 0 \times 2^2 + 1 \times 2^3$$

This is a positional representation of a number

- Each block is based in the position of a representative belonging to the set $\{0, 1\}$:

Then, we have

- This idea of representative... SHINGLES...
  - A a rectangular tile of asphalt composite, wood, metal, or slate used on walls or roofs.

Outline

Finding Similar Documents

Goal

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

What kind of problems can you have?

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.

What kind of problems can you have?

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.

What kind of problems can you have?

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.

Therefore, we need do something

First, a representation of the documents

- Documents consists of words
  - One Shot Representation

Then, Shingle = Word

- This works well for small documents, but a lot of them

Therefore, we need do something

First, a representation of the documents

- Documents consists of words
  - One Shot Representation

Then, Shingle = Word

- This works well for small documents, but a lot of them

Another Example

Words in a Dictionary

- Shingles = Fonts

Or something different?

- Think about it...

Another Example

Words in a Dictionary

- Shingles = Fonts

Or something different?

- Think about it...

Outline

Trying to solve the Approximate Near-Neighbor Problem

If we define the following idea of Neighbor Balls

$$B(x, r) = \{p | d(x, p) \leq r\}$$

It is possible to define two Neighbors to solve such problem

- Basically, a ball where the query is successful
- An another ball where the query fails

Trying to solve the Approximate Near-Neighbor Problem

If we define the following idea of Neighbor Balls

$$B(x, r) = \{p | d(x, p) \leq r\}$$

It is possible to define two Neighbors to solve such problem

- Basically, a ball where the query is successful
- An another ball where the query fails

For this, we can define the following

Definition [3]

- A family $\mathcal{H} = \{h : S \longrightarrow U\}$ is called (r_1, r_2, p_1, p_2)-sensitive for D if for any $q, p \in S$
  - If $p \in B(q, r_1)$ then $Pr_{\mathcal{H}}[h(q) = h(p)] \geq p_1$
  - If $p \notin B(q, r_2)$ then $Pr_{\mathcal{H}}[h(q) = h(p)] \leq p_2$

In order to have something useful

- A Locality-Sensitive family to be useful, it has to satisfy $p_1 > p_2$ and $r_1 < r_2$

For this, we can define the following

Definition [3]

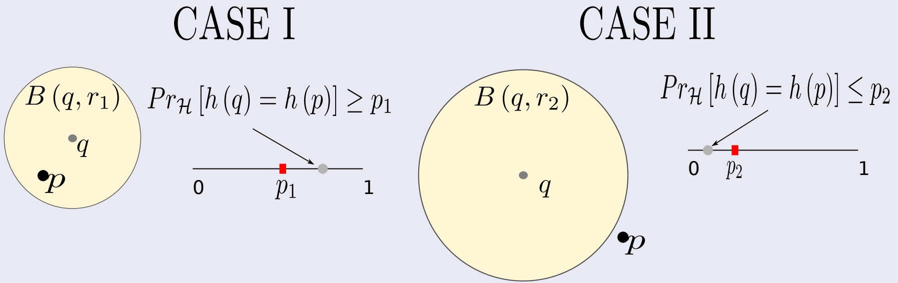
- A family $\mathcal{H} = \{h : S \longrightarrow U\}$ is called (r_1, r_2, p_1, p_2)-sensitive for D if for any $q, p \in S$
  - If $p \in B(q, r_1)$ then $Pr_{\mathcal{H}}[h(q) = h(p)] \geq p_1$
  - If $p \notin B(q, r_2)$ then $Pr_{\mathcal{H}}[h(q) = h(p)] \leq p_2$

In order to have something useful

- A Locality-Sensitive family to be useful, it has to satisfy $p_1 > p_2$ and $r_1 < r_2$

For example

We have rings and intervals



They described the following algorithm

Locality Sensitive Hashing

- Preprocessing
  - Define a function family $G = \{g : S \to U^k\}$ such that $g(p) = [h_1(p), ..., h_k(p)]$ where $h_i \in \mathcal{H}$
  - For an integer l, we choose l functions $g_1, ..., g_l \in G$ independently and uniformly at random.
  - We store each p at the database through the use of Hashing into the buckets.

- If the number of points encountered are greater than $2l$ we interrupt the search.
- Given the found points $p_1, ..., p_t$
  - For each p_j, if $p_j \in B(q, r_2)$ then return YES and p_j else we return NO

They described the following algorithm

Locality Sensitive Hashing

- Preprocessing
  - Define a function family $G = \{g : S \to U^k\}$ such that $g(p) = [h_1(p), ..., h_k(p)]$ where $h_i \in \mathcal{H}$
  - For an integer l, we choose l functions $g_1, ..., g_l \in G$ independently and uniformly at random.
  - We store each p at the database through the use of Hashing into the buckets.

Given a query q in the Search Process

1. We search all buckets $g_1(q), ..., g_l(q)$
2. If the number of points encountered are greater than $2l$ we interrupt the search
3. Given the found points $p_1, ..., p_t$
   1. For each p_j, if $p_j \in B(q, r_2)$ then return YES and p_j else we return NO

They described the following algorithm

Locality Sensitive Hashing

- Preprocessing
  - Define a function family $G = \{g : S \to U^k\}$ such that $g(p) = [h_1(p), ..., h_k(p)]$ where $h_i \in \mathcal{H}$
  - For an integer l, we choose l functions $g_1, ..., g_l \in G$ independently and uniformly at random.
  - We store each p at the database through the use of Hashing into the buckets.

Given a query q in the Search Process

1. We search all buckets $g_1(q), ..., g_l(q)$
2. If the number of points encountered are greater than $2l$ we interrupt the search
3. Given the found points $p_1, ..., p_t$
   1. For each p_j, if $p_j \in B(q, r_2)$ then return YES and p_j else we return NO

They described the following algorithm

Locality Sensitive Hashing

- Preprocessing
  - Define a function family $G = \left\{ g : S \rightarrow U^k \right\}$ such that $g(p) = [h_1(p), ..., h_k(p)]$ where $h_i \in \mathcal{H}$
  - For an integer l, we choose l functions $g_1, ..., g_l \in G$ independently and uniformly at random.
  - We store each p at the database through the use of Hashing into the buckets.

Given a query q in the Search Process

1. We search all buckets $g_1(q), ..., g_l(q)$
2. If the number of points encountered are greater than $2l$ we interrupt the search
3. Given the found points $p_1, ..., p_t$
   1. For each p_j, if $p_j \in B(q, r_2)$ then return YES and p_j else we return NO

They described the following algorithm

Locality Sensitive Hashing

- Preprocessing
  - Define a function family $G = \left\{ g : S \to U^k \right\}$ such that $g(p) = [h_1(p), ..., h_k(p)]$ where $h_i \in \mathcal{H}$
  - For an integer l, we choose l functions $g_1, ..., g_l \in G$ independently and uniformly at random.
  - We store each p at the database through the use of Hashing into the buckets.

Given a query q in the Search Process

1. We search all buckets $g_1(q), ..., g_l(q)$
2. If the number of points encountered are greater than $2l$ we interrupt the search
3. Given the found points $p_1, ..., p_t$
   1. For each p_j, if $p_j \in B(q, r_2)$ then return YES and p_j else we return NO

They described the following algorithm

Locality Sensitive Hashing

- Preprocessing
  - Define a function family $G = \left\{ g : S \rightarrow U^k \right\}$ such that $g(p) = [h_1(p), ..., h_k(p)]$ where $h_i \in \mathcal{H}$
  - For an integer l, we choose l functions $g_1, ..., g_l \in G$ independently and uniformly at random.
  - We store each p at the database through the use of Hashing into the buckets.

Given a query q in the Search Process

1. We search all buckets $g_1(q), ..., g_l(q)$
2. If the number of points encountered are greater than $2l$ we interrupt the search
3. Given the found points $p_1, ..., p_t$
   1. For each p_j, if $p_j \in B(q, r_2)$ then return YES and p_j else we return NO

They described the following algorithm

Locality Sensitive Hashing

- Preprocessing
  - Define a function family $G = \left\{ g : S \to U^k \right\}$ such that $g(p) = [h_1(p), ..., h_k(p)]$ where $h_i \in \mathcal{H}$
  - For an integer l, we choose l functions $g_1, ..., g_l \in G$ independently and uniformly at random.
  - We store each p at the database through the use of Hashing into the buckets.

Given a query q in the Search Process

1. We search all buckets $g_1(q), ..., g_l(q)$
2. If the number of points encountered are greater than $2l$ we interrupt the search
3. Given the found points $p_1, ..., p_t$
   1. For each p_j, if $p_j \in B(q, r_2)$ then return YES and p_j else we return NO

Therefore

Then, we choose k and l to ensure that with constant probability the following properties hold

1. If there exists $p \in B(q, r_1)$ then $g_j(p) = g_j(q)$ for some $j = 1, ..., l$.
2. The total number of collisions of q with points from $P - B(q, r_1)$ is less than $2l$:

$$\sum_{j=1}^{l} \left| P - B(q, r_1) \cap g_j^{-1}(g_j(q)) \right| < 2l$$

Something Notable

- If (1) and (2) hold, the algorithm is correct.

Therefore

Then, we choose k and l to ensure that with constant probability the following properties hold

1. If there exists $p \in B(q, r_1)$ then $g_j(p) = g_j(q)$ for some $j = 1, ..., l$.
2. The total number of collisions of q with points from $P - B(q, r_1)$ is less than $2l$:

$$\sum_{j=1}^{l} \left| P - B(q, r_1) \cap g_j^{-1}(g_j(q)) \right| < 2l$$

Something Notable

- If (1) and (2) hold, the algorithm is correct.

Outline

Theorem

(r_1, r_2, p_1, p_2)-sensitive family $\mathcal{H}$ for D ($p_1 > p_2$ and $r_1 < r_2$)

- Then, there exists and algorithm for (r_1, r_2)-Point Location in Equal Balls under measure D which uses $O\left(dn + n^{1+\rho}\right)$ space and $O\left(n^{\rho}\right)$ evaluations of the hash function for each query where

$$\rho = \frac{\log \frac{1}{p_1}}{\log \frac{1}{p_2}}$$

Proof

For this, we only need (1) and (2) hold

- With probability P_1 and P_2 strictly greater than half

Assume that $p \in B(q, r_1)$

- Set $k = \log_{\frac{1}{p_2}} n$, an arbitrary number of dimensions for $g(p) = [h_1(p), ..., h_k(p)]$

We have that

$$\left(\frac{1}{p_2}\right)^k = n \log_{\frac{1}{p_2}} \frac{1}{p_2} \Rightarrow p_2^k = \frac{1}{n}$$

Proof

For this, we only need (1) and (2) hold

- With probability P_1 and P_2 strictly greater than half

Assume that $p \in B(q, r_1)$

- Set $k = \log_{\frac{1}{p_2}} n$, an arbitrary number of dimensions for $g(p) = [h_1(p), ..., h_k(p)]$

We have that

$$\left(\frac{1}{p_2}\right)^k = n \log_{\frac{1}{p_2}} \frac{1}{p_2} \Rightarrow p_2^k = \frac{1}{n}$$

Proof

For this, we only need (1) and (2) hold

- With probability P_1 and P_2 strictly greater than half

Assume that $p \in B(q, r_1)$

- Set $k = \log_{\frac{1}{p_2}} n$, an arbitrary number of dimensions for $g(p) = [h_1(p), ..., h_k(p)]$

We have that

$$\left(\frac{1}{p_2}\right)^k = n \log_{\frac{1}{p_2}} \frac{1}{p_2} \Rightarrow p_2^k = \frac{1}{n}$$

Proof

Then the probability that $g(p) = g(q)$ for $p \in P - B(q, r_2)$

- It is at most $p_2^k = \frac{1}{n}$ assuming that the hash functions are randomly independently selected.

Thus, the expected number of elements from $P - B(q, r_2)$

- Colliding with q under fixed g_j is at most 1.

Then, the expected number of such collisions with any g_j is at most l

- Then we can use the Markov inequality

Proof

Then the probability that $g(p) = g(q)$ for $p \in P - B(q, r_2)$

- It is at most $p_2^k = \frac{1}{n}$ assuming that the hash functions are randomly independently selected.

Thus, the expected number of elements from $P - B(q, r_2)$

- Colliding with q under fixed g_j is at most 1.

Then, the expected number of such collisions with any g_j is at most l

- Then we can use the Markov inequality

Proof

Then the probability that $g(p) = g(q)$ for $p \in P - B(q, r_2)$

- It is at most $p_2^k = \frac{1}{n}$ assuming that the hash functions are randomly independently selected.

Thus, the expected number of elements from $P - B(q, r_2)$

- Colliding with q under fixed g_j is at most 1.

Then, the expected number of such collisions with any g_j is at most l

- Then we can use the Markov inequality

Markov Inequality

If X is a non-negative random variable and $a > 0$

- Then, the probability that X is at least a is at most the expectation of X divided by a:

$$P\left(X \geq a\right) \leq \frac{E\left(X\right)}{a}$$

Therefore for any g_j a random variable

$$P\left(g_j \geq 2l\right) \leq \frac{E\left(g_j\right)}{2l} = \frac{l}{2l} = \frac{1}{2}$$

Markov Inequality

If X is a non-negative random variable and $a > 0$

- Then, the probability that X is at least a is at most the expectation of X divided by a:

$$P(X \geq a) \leq \frac{E(X)}{a}$$

Therefore for any g_j a random variable

$$P(g_j \geq 2l) \leq \frac{E(g_j)}{2l} = \frac{l}{2l} = \frac{1}{2}$$

cinvestav

Then

At property (2)

- The total number of collisions of q with points from $P - B(q, r_1)$ is less than $2l$:

$$\sum_{j=1}^{l} \left| P - B(q, r_1) \cap g_j^{-1}(g_j(q)) \right| < 2l$$

Therefore, we have

Then, we have that $\sum_{j=1}^{l}\left|P-B\left(q,r_1\right)\cap g_j^{-1}\left(g_j\left(q\right)\right)\right|=*$ is also a random variable

$$
\begin{aligned}
P\left(*<2l\right) &= 1-P\left(*\geq 2l\right)\\
&> 1-\frac{1}{2}=\frac{1}{2}
\end{aligned}
$$

Now

Consider now the probability of $g_j(p) = g_j(q)$

- Given that $p \in B(q, r_1)$ then $Pr_{\mathcal{H}}[h(q) = h(p)] \geq p_1$

$$P(g_j(p) = g_j(q)) \geq (p_1)^k = p_1^{\log_{\frac{1}{p_2}} n} = n^{-\frac{\log 1/p_1}{\log 1/p_2}} = n^{-\rho}$$

Thus, the probability that such a g_j exists is at least

$$P_1 \geq 1 - (1 - n^{-\rho})^l$$

Now

Consider now the probability of $g_j(p) = g_j(q)$

- Given that $p \in B(q, r_1)$ then $Pr_{\mathcal{H}}[h(q) = h(p)] \geq p_1$

$$P(g_j(p) = g_j(q)) \geq (p_1)^k = p_1^{\log_{\frac{1}{p_2}} n} = n^{-\frac{\log 1/p_1}{\log 1/p_2}} = n^{-\rho}$$

Thus, the probability that such a g_j exists is at least

$$P_1 \geq 1 - \left(1 - n^{-\rho}\right)^l$$

Why?

We have $P\left(g_j\left(p\right) \neq g_j\left(q\right)\right) = 1 - P\left(g_j\left(p\right) = g_j\left(q\right)\right) \leq 1 - n^{-\rho}$

- Thus, we have
  $P_1 = P\left(p \in \mathrm{B}(q, r_1) \text{ then } g_j\left(p\right) = g_j\left(q\right) \text{ for some } j = 1, ..., l\right)$

$$P\left(p \in \mathrm{B}(q, r_1) \text{ then } g_j\left(p\right) = g_j\left(q\right), \text{ for some } j = 1, ..., l\right) \geq 1 - \left(1 - n^{-\rho}\right)^l$$

By Setting $l = n^{\rho}$

$$P_1 \geq 1 - \frac{1}{e} \geq \frac{1}{2} \text{ Q.E.D.}$$

Why?

We have $P\left(g_j\left(p\right) \neq g_j\left(q\right)\right) = 1 - P\left(g_j\left(p\right) = g_j\left(q\right)\right) \leq 1 - n^{-\rho}$

- Thus, we have
  $P_1 = P\left(p \in \mathrm{B}(q, r_1) \text{ then } g_j\left(p\right) = g_j\left(q\right) \text{ for some } j = 1, ..., l\right)$

$$P\left(p \in \mathrm{B}(q, r_1) \text{ then } g_j\left(p\right) = g_j\left(q\right), \text{ for some } j = 1, ..., l\right) \geq 1 - \left(1 - n^{-\rho}\right)^l$$

By Setting $l = n^{\rho}$

$$P_1 > 1 - \frac{1}{e} > \frac{1}{2} \text{ Q.E.D.}$$

Outline

We can use this [3]

That Jaccard Similarity is a way to define distance

- There are others, for example the Hamming Distance...

For example

- Consider the Hamming cube $\{0,1\}^d$ the there is $\ell_1 - distance$ defined has

$$D(x,y) = \sum_{i=1}^{d} |x_k - y_k|$$

  - It simply counts the number of coordinates where the points differ.

We can use this [3]

That Jaccard Similarity is a way to define distance

- There are others, for example the Hamming Distance...

For example

- Consider the Hamming cube $\{0,1\}^d$ the there is $\ell_1 - distance$ defined has

$$D(x,y) = \sum_{i=1}^{d} |x_k - y_k|$$

  - It simply counts the number of coordinates where the points differ.

Now, What if we introduce a hash family?

Consider the following hash family of functions

$$\mathcal{H} = \left\{ h_k | h_k(x) = k^{th} \text{ bit of } x \right\}$$

Then, a Direct Application

Proposition - remember $p_1 > p_2$ and $r_1 < r_2$ for utility

- Let $S = \mathcal{H}^d$ and $D(p,q)$ be a Hamming metric. Then for any $r, \epsilon > 0$ then $\mathcal{H}$ is $\left(r, r(1+\epsilon), 1 - \frac{r}{d}, 1 - \frac{r(1+\epsilon)}{d}\right)$-senstive.

From this the following Corollary [3]

- For any $\epsilon > 0$, there exists an algorithm for ϵ-PLEB in $\mathcal{H}^d$ or l_p^d for any $p \in [1,2]$ using $O\left(dn + n^{1+\frac{1}{1+\epsilon}}\right)$ space and $O\left(n^{\frac{1}{1+\epsilon}}\right)$ hash function for each query (n is the size of the database). The hash function can be evaluated using $O(d)$ operations.

Then, a Direct Application

Proposition - remember $p_1 > p_2$ and $r_1 < r_2$ for utility

- Let $S = \mathcal{H}^d$ and $D(p,q)$ be a Hamming metric. Then for any $r, \epsilon > 0$ then $\mathcal{H}$ is $\left(r, r(1+\epsilon), 1 - \frac{r}{d}, 1 - \frac{r(1+\epsilon)}{d}\right)$-senstive.

From this the following Corollary [3]

- For any $\epsilon > 0$, there exists an algorithm for ϵ-PLEB in $\mathcal{H}^d$ or l_p^d for any $p \in [1,2]$ using $O\left(dn + n^{1+\frac{1}{1+\epsilon}}\right)$ space and $O\left(n^{\frac{1}{1+\epsilon}}\right)$ hash function for each query (n is the size of the database). The hash function can be evaluated using $O(d)$ operations.

Outline

From this

We can actually do better

- If we assume sparse data

Proposition

- Let S be the set of all subsets of $X = \{1, ..., x\}$ (Shingles/Set Representation) and let D be the set resemblance measure (Jaccard). Then, for $1 > r_1 > r_2 > 0$ the following hash family is (r_1, r_2, r_1, r_2)-sensitive

$$\mathcal{H} = \left\{ h_\pi | h_\pi (A) = \max_{a \in A} \pi (a) \text{ , } \pi \text{ is a permutation of } X \right\}$$

From this

We can actually do better

- If we assume sparse data

Proposition

- Let S be the set of all subsets of $X = \{1, ..., x\}$ (Shingles/Set Representation) and let D be the set resemblance measure (Jaccard). Then, for $1 > r_1 > r_2 > 0$ the following hash family is (r_1, r_2, r_1, r_2)-sensitive

$$\mathcal{H} = \left\{ h_\pi | h_\pi (A) = \max_{a \in A} \pi (a), \pi \text{ is a permutation of } X \right\}$$

Here

Shingles

- A way to represent objects using power set elements when having basic set construction elements of such objects

For example, in short documents

- You can disregard the order (Although in modern algorithms, we have seen the utility of such order) and have a set representation of the document

Where the elements

- They are the words at the language dictionary.

Here

Shingles

- A way to represent objects using power set elements when having basic set construction elements of such objects

For example, in short documents

- You can disregard the order (Although in modern algorithms, we have seen the utility of such order) and have a set representation of the document

Where the elements

- They are the words at the language dictionary.

Here

Shingles

- A way to represent objects using power set elements when having basic set construction elements of such objects

For example, in short documents

- You can disregard the order (Although in modern algorithms, we have seen the utility of such order) and have a set representation of the document

Where the elements

- They are the words at the language dictionary.

Therefore

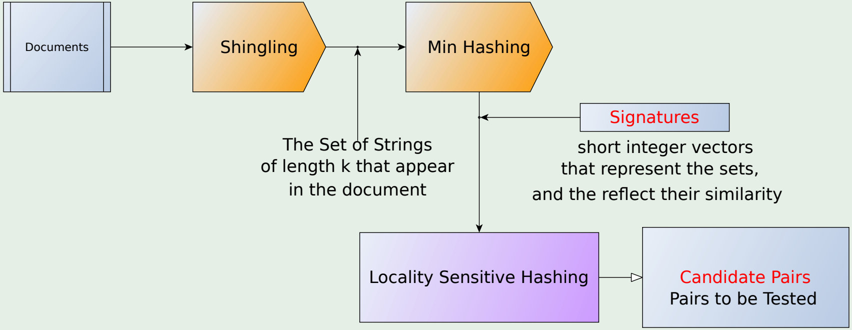
We have the following Corollary

- For $0 < \epsilon, r < 1$, there exists an algorithm for $(r, \epsilon r)$-PLEB under D using $O\left(dn + n^{1+\rho}\right)$ space and $O\left(n^{\rho}\right)$ evaluations for each query, where $\rho = \frac{\log r}{\log \epsilon r}$.

Outline

The Pipeline for the Locality Sensitive Hashing

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.

We can define

- **Document = set of words** appearing in document.
- **Document = set of "important" words.**
- Problem, they do not work well for this application. Why?

- Think about Sets: Use Shingles!!!

Documents as High-Dimensional Data

Step 1: Shingling

- Convert documents to sets.

We can define

- **Document = set of words** appearing in document.
- **Document = set of "important" words**.
- Problem, they do not work well for this application. Why?

We want to avoid to get tangled in the text structure

- Avoid taking in account the ordering of words!
- Think about Sets: Use Shingles!!!

Documents as High-Dimensional Data

Step 1: Shingling

- Convert documents to sets.

We can define

- **Document = set of words** appearing in document.
- **Document = set of "important" words**.
- Problem, they do not work well for this application. Why?

We want to avoid to get tangled in the text structure

- Avoid taking in account the ordering of words!
- Think about Sets: Use Shingles!!!

Documents as High-Dimensional Data

Step 1: Shingling

- Convert documents to sets.

We can define

- **Document = set of words** appearing in document.
- **Document = set of "important" words**.
- Problem, they do not work well for this application. Why?

We want to avoid to get tangled in the text structure

- Avoid taking in account the ordering of words!
- Think about Sets: Use Shingles!!!

Documents as High-Dimensional Data

Step 1: Shingling

- Convert documents to sets.

We can define

- **Document = set of words** appearing in document.
- **Document = set of "important" words**.
- Problem, they do not work well for this application. Why?

We want to avoid to get tangled in the text structure

- Avoid taking in account the ordering of words!
- Think about Sets: 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: $S(D_1) = \{ab, bc, ca\}$
  - Another 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: $S(D_1) = \{ab, bc, ca\}$
  - Another 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: $S(D_1) = \{ab, bc, ca\}$
  - Another 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: $S(D_1) = \{ab, bc, ca\}$
  - Another 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.

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 (Use the sensitivity hash family).

- Hash the shingles using the Universal Hash 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 (Use the sensitivity hash family).

Example

- $k = 2$; document $D_1 = abcab$ Set of 2-shingles: $S(D_1) = \{ab, bc, ca\}$
- Hash the shingles using the Universal Hash 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 (Use the sensitivity hash family).

Example

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

Outline

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

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

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

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

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

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

Outline

However

This is assuming a non-sparse representation

- But using an array of int to represent the shingles at the documents by bits 0 or 1

However

- We can use sparse vector (Hit in speed but less space)

However

This is assuming a non-sparse representation

- But using an array of int to represent the shingles at the documents by bits 0 or 1

However

- We can use sparse vector (Hit in speed but less space)

How do we can implement this? SWAR-Popcount

Code - SWAR-Popcount - Divide and Conquer

```
// This works only in 32 bits
int PopCount(int vector){

  int i = vector;

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

  return i;

}
```

We can use this (There are better.)

Together with AND and OR to implement the Jaccard similarity

How do we can implement this? SWAR-Popcount

Code - SWAR-Popcount - Divide and Conquer

```
// This works only in 32 bits
int PopCount(int vector){

  int i = vector;

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

  return i;

}
```

We can use this (There are better)

Together with AND and OR to implement the Jaccard similarity

Therefore

We have

```
long Jacard(int *C1, int *C2, int n){
        int i;
        long union, intersection;
        union = 0;
        intersection = 0;
        for(i = 0; i < n; i++){
                union = union +...
                        (long)PopCount( C1[i] | C2[i] );
                intersection = intersection +...
                        (long)PopCount( C1[i] & C2[i] );
        }
        return union/intersection;
}
```

Outline

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.

Another Motivation for Min-Hash/Locality Sensitive Hashing

Imagine the following

- We need to find near-duplicate documents with data sets of size in the millions, for example, $N = 1,000,000$.

Another Motivation for Min-Hash/Locality Sensitive Hashing

Imagine the following

- We need to find near-duplicate documents with data sets of size in the millions, for example, $N = 1,000,000$.

Compute pairwise Jaccard similarities

- Naively, we would have to compute pairwise Jaccard similarities for every pair of docs.
  - Not a god idea when, $\frac{N(N-1)}{2} \approx 5 * 10^{11}$ comparisons.
  - At 10^5 seconds per day and 10^6 comparisons per second, it would take 5 days.

Another Motivation for Min-Hash/Locality Sensitive Hashing

Imagine the following

- We need to find near-duplicate documents with data sets of size in the millions, for example, $N = 1,000,000$.

Compute pairwise Jaccard similarities

- Naively, we would have to compute pairwise Jaccard similarities for every pair of docs.
  - Not a god idea when, $\frac{N(N-1)}{2} \approx 5 * 10^{11}$ comparisons.
  - At 10^5 seconds per day and 10^6 comparisons per second, it would take 5 days.

For something larger

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

Another Motivation for Min-Hash/Locality Sensitive Hashing

Imagine the following

- We need to find near-duplicate documents with data sets of size in the millions, for example, $N = 1,000,000$.

Compute pairwise Jaccard similarities

- Naively, we would have to compute pairwise Jaccard similarities for every pair of docs.
  - Not a god idea when, $\frac{N(N-1)}{2} \approx 5 * 10^{11}$ comparisons.
  - At 10^5 seconds per day and 10^6 comparisons per second, it would take 5 days.

For something larger

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

Another Motivation for Min-Hash/Locality Sensitive Hashing

Imagine the following

- We need to find near-duplicate documents with data sets of size in the millions, for example, $N = 1,000,000$.

Compute pairwise Jaccard similarities

- Naively, we would have to compute pairwise Jaccard similarities for every pair of docs.
  - Not a god idea when, $\frac{N(N-1)}{2} \approx 5 * 10^{11}$ comparisons.
  - At 10^5 seconds per day and 10^6 comparisons per second, it would take 5 days.

For something larger

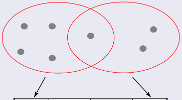
- For $N = 10$ million, it takes more than a year...

Outline

Encoding Sets as Bit Vectors

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

- Encode sets using 0/1 (bit, Boolean) vectors.
  - One dimension per element in the universal set.



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

Encoding Sets as Bit Vectors

As we said it

- Interpret set intersection as bit-wise **AND**, and set union as bit-wise **OR.**

Example

$C_1 = 10111$ and $C_2 = 10011$

- Size of intersection $= 3$ and size of union $= 4$,

Jaccard similarity

$$sim\left(C_1, C_2\right) = \frac{3}{4}$$

Thus, the distance

$$d\left(C_1, C_2\right) = 1 - \frac{3}{4} = \frac{1}{4}$$

Example

$C_1 = 10111$ and $C_2 = 10011$

- Size of intersection $= 3$ and size of union $= 4$,

Jaccard similarity

$$sim\left(C_1, C_2\right) = \frac{3}{4}$$

Thus, the distance

$$d\left(C_1, C_2\right) = 1 - \frac{3}{4} = \frac{1}{4}$$

Example

$C_1 = 10111$ and $C_2 = 10011$

- Size of intersection $= 3$ and size of union $= 4$,

Jaccard similarity

$$sim\left(C_1, C_2\right) = \frac{3}{4}$$

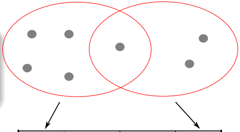
Thus, the distance

$$d\left(C_1, C_2\right) = 1 - \frac{3}{4} = \frac{1}{4}$$

From Sets to Boolean Matrices

Rows

- Rows are equal to elements (shingles)



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

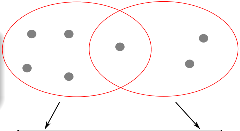
From Sets to Boolean Matrices

Rows

- Rows are equal to elements (shingles)

Columns

- The Columns are equal to sets (documents)
  - ONE 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 ONE)



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

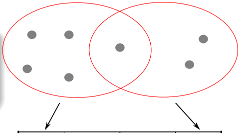
From Sets to Boolean Matrices

Rows

- Rows are equal to elements (shingles)

Columns

- The Columns are equal to sets (documents)
  - ONE 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 ONE)



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

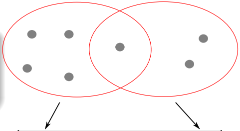
From Sets to Boolean Matrices

Rows

- Rows are equal to elements (shingles)

Columns

- The Columns are equal to sets (documents)
  - ONE 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 ONE)



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

Here, a problem arises

Column similarity is the Jaccard similarity of the corresponding sets (rows with value 1)

- **Such matrix is typically sparse!**

We need to solve this

- After all sparsity is problematic for the use of memory.

Here, a problem arises

Column similarity is the Jaccard similarity of the corresponding sets (rows with value 1)

- **Such matrix is typically sparse!**

We need to solve this

- After all sparsity is problematic for the use of memory.

Outline

Finding Similar Columns

Documents → Sets of shingles

- We have been able to represent them as sets vectors in a matrix

We can now try to reduce the size of the sparse representations

- Using a technique called Min-Hash to find small signatures...

However, we still have a problem

- Because comparing all pairs is too expansive...

Finding Similar Columns

Documents → Sets of shingles

- We have been able to represent them as sets vectors in a matrix

We can now try to reduce the size of the sparse representations

- Using a technique called Min-Hash to find small signatures...

However, we still have a problem

- Because comparing all pairs is too expansive...

Finding Similar Columns

Documents $\rightarrow$ Sets of shingles

- We have been able to represent them as sets vectors in a matrix

We can now try to reduce the size of the sparse representations

- Using a technique called Min-Hash to find small signatures...

However, we still have a problem

- Because comparing all pairs is too expansive...

How do we accomplish something like that

First than anything

- What are going to be our signatures of columns?
  - Which in addition keeps a specific property!!!

Which property?

- Once new signatures are generated...
  - if $s(C_1, C_2) \rightarrow 1$, the similarity of such signatures is also high!!!

How do we accomplish something like that

First than anything

- What are going to be our signatures of columns?
  - Which in addition keeps a specific property!!!

Which property?

- Once new signatures are generated...
  - if $s(C_1, C_2) \rightarrow 1$, the similarity of such signatures is also high!!!

Outline

We can use Hashing!!!

Hashing the Columns

- Hash each column C to a small signature $h(C)$

Such that

- $h(C)$ is small enough that the signature fits in RAM
- $sim(C_1, C_2)$ is the same as the "similarity" of signatures $h(C_1)$ and $h(C_2)$

We can use Hashing!!!

Hashing the Columns

- Hash each column C to a small signature $h(C)$

Such that

- $h(C)$ is small enough that the signature fits in RAM
- $sim\left(C_1, C_2\right)$ is the same as the "similarity" of signatures $h\left(C_1\right)$ and $h\left(C_2\right)$

Therefore, we want

Find a hash function $h(\cdot)$ such that

- if $sim(C_1, C_2)$ is high, then with high probability $h(C_1) = h(C_2)$.
- if $sim(C_1, C_2)$ is low, then with high probability $h(C_1) \neq h(C_2)$.

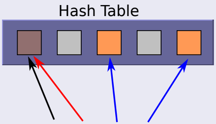
We can use the buckets of the Hash Table for this

Therefore, we want

Find a hash function $h(\cdot)$ such that

- if $sim\left(C_1, C_2\right)$ is high, then with high probability $h\left(C_1\right) = h\left(C_2\right)$.
- if $sim\left(C_1, C_2\right)$ is low, then with high probability $h\left(C_1\right) \neq h\left(C_2\right)$.

We can use the buckets of the Hash Table for this



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

Thus, we can do the following

Thus, we hash documents into buckets

- And we expect that the hash respect the similarity of "near" duplicates.

Outline

Min-Hashing

Similarity Metric

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

Min-Hashing

Similarity Metric

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

Hash Functions

- There is a suitable hash function for the Jaccard similarity, Min-Hashing

Min-Hashing

Similarity Metric

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

Hash Functions

- There is a suitable hash function for the Jaccard similarity, Min-Hashing

Remember the Corollary about the Permutation Hash Family

Random permutation

- Imagine the rows of the Boolean matrix permuted under random permutation π .

Define a Hash function $h_\pi(C)$

- $h_\pi(C)$ = the number of the first row, in order π, in which column C has value 1,

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

Thus, we can use this permutations

- Use many independent hash functions to create a signature of a column.

Remember the Corollary about the Permutation Hash Family

Random permutation

- Imagine the rows of the Boolean matrix permuted under random permutation π .

Define a Hash function $h_\pi(C)$

- $h_\pi(C) =$ the number of the first row, in order π, in which column C has value 1,

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

Thus, we can use this permutations

- Use many independent hash functions to create a signature of a column.

Remember the Corollary about the Permutation Hash Family

Random permutation

- Imagine the rows of the Boolean matrix permuted under random permutation π .

Define a Hash function $h_\pi(C)$

- $h_\pi(C) =$ the number of the first row, in order π, in which column C has value 1,

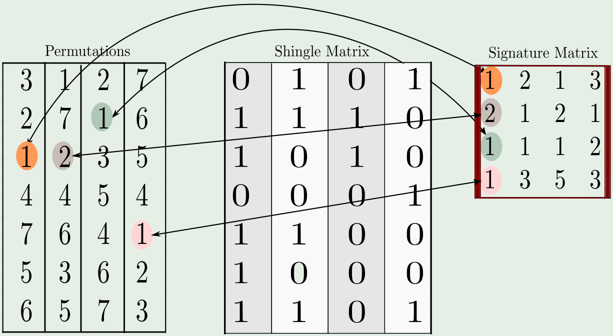
$$h_\pi(C) = \min_\pi \{\pi(C)\}$$

Thus, we can use this permutations

- Use many independent hash functions to create a signature of a column.

Min-Hashing Example

We have the following mapping



Surprising Property

When choosing a random permutation π

- We claim having the following equality:

$$Pr\left[h_{\pi}\left(C_1\right) = h_{\pi}\left(C_2\right)\right] = sim\left(C_1, C_2\right)$$

How is this possible?

- Let X be a document (set of shingles)

We have that given $|X|$ shingles, then under random uniform permutation

$$Pr\left[\pi\left(x\right) = \min\left(\pi\left(X\right)\right)\right] = \frac{1}{|X|}$$

Surprising Property

When choosing a random permutation π

- We claim having the following equality:

$$Pr\left[h_{\pi}\left(C_1\right) = h_{\pi}\left(C_2\right)\right] = sim\left(C_1, C_2\right)$$

How is this possible?

- Let X be a document (set of shingles)

We have that given $|X|$ shingles, then under random uniform permutation

$$Pr\left[\pi\left(x\right) = \min\left(\pi\left(X\right)\right)\right] = \frac{1}{|X|}$$

Surprising Property

When choosing a random permutation π

- We claim having the following equality:

$$Pr\left[h_{\pi}\left(C_1\right) = h_{\pi}\left(C_2\right)\right] = sim\left(C_1, C_2\right)$$

How is this possible?

- Let X be a document (set of shingles)

We have that given $|X|$ shingles, then under random uniform permutation

$$Pr\left[\pi\left(x\right) = \min\left(\pi\left(X\right)\right)\right] = \frac{1}{|X|}$$

Why is this possible

It is equally likely that any $x \in X$ is mapped to the min element

- Thus, we have an x such that

$$\pi(x) = \min\left[\pi\left(C_1 \bigcup C_2\right)\right]$$

Then either

- $\pi(x) = \min(\pi(C_1))$ if $x \in C_1$, or $\pi(x) = \min(\pi(C_2))$ if $x \in C2$

Thus, we have

- One of the two cols had to have 1 at position x.

Why is this possible

It is equally likely that any $x \in X$ is mapped to the min element

- Thus, we have an x such that

$$\pi\left(x\right) = \min\left[\pi\left(C_1 \bigcup C_2\right)\right]$$

Then either

- $\pi\left(x\right) = \min\left(\pi\left(C_1\right)\right)$ if $x \in C_1$, or $\pi(x) = \min\left(\pi\left(C_2\right)\right)$ if $x \in C2$

Thus, we have

- One of the two cols had to have 1 at position x.

Why is this possible

It is equally likely that any $x \in X$ is mapped to the min element

- Thus, we have an x such that

$$\pi(x) = \min\left[\pi\left(C_1 \bigcup C_2\right)\right]$$

Then either

- $\pi(x) = \min(\pi(C_1))$ if $x \in C_1$, or $\pi(x) = \min(\pi(C_2))$ if $x \in C2$

Thus, we have

- One of the two cols had to have 1 at position x.

Then, we have that

We realize that when $x = C_1 \cap C_2$

$$Pr\left[\min\left(\pi\left(C_1\right)\right) = \min\left(\pi\left(C_2\right)\right)\right] = \frac{|C_1 \cap C_2|}{|C_1 \cup C_2|} = sim\left(C_1, C_2\right)$$

Now, we have Four Types of Rows between Documents

Given cols C_1 and C_2, rows may be classified based on its similarity

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

Then, we define

The following cardinalities

1. a =Number of Rows of type A,
2. b =Number of Rows of type B,
3. c =Number of Rows of type C,
4. d =Number of Rows of type D.

Then, we have

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

Then, we define

The following cardinalities

1. a =Number of Rows of type A,
2. b =Number of Rows of type B,
3. c =Number of Rows of type C,
4. d =Number of Rows of type D.

Then, we have

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

Then, we have

Look down the cols C_1 and C_2 until we see a 1

$$Pr\left[h\left(C_1\right)=h\left(C_2\right)\right]=sim\left(C_1,C_2\right)$$

Something Notable

- If it's a type-A row, then $h\left(C_1\right)=h\left(C_2\right)$
- If a type-B or type-C row, then not.

Finally, as they say

$$Pr\left[h_\pi\left(C_1\right)=h_\pi\left(C_2\right)\right]=sim\left(C_1,C_2\right)$$

Then, we have

Look down the cols C_1 and C_2 until we see a 1

$$Pr\left[h\left(C_1\right) = h\left(C_2\right)\right] = sim\left(C_1, C_2\right)$$

Something Notable

- If it's a type-A row, then $h\left(C_1\right) = h\left(C_2\right)$
- If a type-B or type-C row, then not.

Finally, as they say

$$Pr\left[h_\pi\left(C_1\right) = h_\pi\left(C_2\right)\right] = sim\left(C_1, C_2\right)$$

Then, we have

Look down the cols C_1 and C_2 until we see a 1

$$Pr\left[h\left(C_1\right) = h\left(C_2\right)\right] = sim\left(C_1, C_2\right)$$

Something Notable

- If it's a type-A row, then $h\left(C_1\right) = h\left(C_2\right)$
- If a type-B or type-C row, then not.

Finally, as they say

$$Pr\left[h_\pi\left(C_1\right) = h_\pi\left(C_2\right)\right] = sim\left(C_1, C_2\right)$$

Similarity for Signatures

We know $Pr\left[h_{\pi}\left(C_1\right) = h_{\pi}\left(C_2\right)\right] = sim\left(C_1, C_2\right)$

- 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 $Pr\left[h_{\pi}\left(C_1\right)=h_{\pi}\left(C_2\right)\right]=sim\left(C_1,C_2\right)$

- 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 $Pr\left[h_{\pi}\left(C_1\right) = h_{\pi}\left(C_2\right)\right] = sim\left(C_1, C_2\right)$

- 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

Example

Similarity	$C_1 \vert C_2$	$C_1 \vert C_3$	$C_1 \vert C_4$	$C_2 \vert C_3$	$C_2 \vert C_4$	$C_3 \vert C_4$
Vector Shingles	$\frac{3}{6}$	$\frac{2}{5}$	$\frac{1}{7}$	$\frac{1}{5}$	$\frac{2}{5}$	0
Vector Signatures	$\frac{1}{3}$	$\frac{2}{3}$	0	$\frac{1}{4}$	$\frac{1}{4}$	0

Permutations

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

Shingle Matrix

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

Signature Matrix

1	2	1	3
2	1	2	1
1	1	1	2
1	3	5	3

We can see that

We have the following

Similarity	$C_1\|C_2$	$C_1\|C_3$	$C_1\|C_4$	$C_2\|C_3$	$C_2\|C_4$	$C_3\|C_4$
Vector Shingles	$\frac{3}{6}$	$\frac{2}{5}$	$\frac{1}{7}$	$\frac{1}{5}$	$\frac{2}{5}$	0
Vector Signatures	$\frac{1}{3}$	$\frac{2}{3}$	0	$\frac{1}{4}$	$\frac{1}{4}$	0

Therefore the need to have more permutations

- Remember the Corollary?

We can see that

We have the following

Similarity	$C_1\|C_2$	$C_1\|C_3$	$C_1\|C_4$	$C_2\|C_3$	$C_2\|C_4$	$C_3\|C_4$
Vector Shingles	$\frac{3}{6}$	$\frac{2}{5}$	$\frac{1}{7}$	$\frac{1}{5}$	$\frac{2}{5}$	0
Vector Signatures	$\frac{1}{3}$	$\frac{2}{3}$	0	$\frac{1}{4}$	$\frac{1}{4}$	0

Therefore the need to have more permutations

- Remember the Corollary?

After Many Proofs

Corollary [3]

- For $0 < \epsilon, r < 1$, there exists an algorithm for $(r, \epsilon r)$-PLEB under D using $O\left(dn + n^{1+\rho}\right)$ space and $O\left(n^{\rho}\right)$ evaluations for each query, where $\rho = \frac{\log r}{\log \epsilon r}$.

Min-Hash Signatures

We increase the number of signatures too look more like the original sim Vector Shingle based

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

We have that

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

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

- The signature of the document can be made small ~ 100 bytes!

Min-Hash Signatures

We increase the number of signatures too look more like the original sim Vector Shingle based

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

We have that

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

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

- The signature of the document can be made small ~ 100 bytes!

Outline

Implementation Trick

However

- Permuting rows is prohibitive!!!

And Hashing come to the rescue again!!!

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

Implementation Trick

However

- Permuting rows is prohibitive!!!

And Hashing come to the rescue again!!!

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

One-pass implementation

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

1. Initialize all $sig(C)[i] = \infty$
2. Scan rows looking for 1's
3. Suppose row j has 1 in column C
4. Then for each g_i:
5. If $g_i(j) < sig(C)[i]$, then $sig(C)[i] = g_i(j)$

One-pass implementation

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

1. Initialize all $sig(C)[i] = \infty$
2. Scan rows looking for $1's$
3. Suppose row j has 1 in column C
4. Then for each g_i:
5. If $g_i(j) < sig(C)[i]$, then $sig(C)[i] = g_i(j)$

One-pass implementation

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

1. Initialize all $sig(C)[i] = \infty$
2. Scan rows looking for $1'$s
3. Suppose row j has 1 in column C
4. Then for each g_i:
5. If $g_i(j) < sig(C)[i]$, then $sig(C)[i] = g_i(j)$

One-pass implementation

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

1. Initialize all $sig(C)[i] = \infty$
2. Scan rows looking for $1'$s
3. Suppose row j has 1 in column C
4. Then for each g_i:
5. If $g_i(j) < sig(C)[i]$, then $sig(C)[i] = g_i(j)$

One-pass implementation

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

1. Initialize all $sig(C)[i] = \infty$
2. Scan rows looking for $1'$s
3. Suppose row j has 1 in column C
4. Then for each g_i:
5. If $g_i(j) < sig(C)[i]$, then $sig(C)[i] = g_i(j)$

Selecting such hash functions

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

- Universal Hashing

For example, $h_{a,b}(x) = ((a \cdot x + b) \mod p) \mod N$ where:

- a, b random integers
- p a prime number $(p > N)$

Selecting such hash functions

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

- Universal Hashing

For example, $h_{a,b}(x) = ((a \cdot x + b) \mod p) \mod N$ where:

- a, b random integers
- p a prime number $(p > N)$

Outline

Locality Sensitive Hashing

Find documents with Jaccard similarity at least s

- For some similarity threshold, for example, $s = 0.8$

Locality Sensitive Hashing – 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 Min-Hash matrices

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

Locality Sensitive Hashing

Find documents with Jaccard similarity at least s

- For some similarity threshold, for example, $s = 0.8$

Locality Sensitive Hashing – 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 Min-Hash matrices

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

Locality Sensitive Hashing

Find documents with Jaccard similarity at least s

- For some similarity threshold, for example, $s = 0.8$

Locality Sensitive Hashing – 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 Min-Hash matrices

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

Candidates from Min-Hash

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

- Around this, we need to design the Min-Hash

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.

Candidates from Min-Hash

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

- Around this, we need to design the Min-Hash

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.

Why

Remember

- For $0 < \epsilon, r < 1$, there exists an algorithm for $(r, \epsilon r)$-PLEB under D using $O\left(dn + n^{1+\rho}\right)$ space and $O\left(n^{\rho}\right)$ evaluations for each query, where $\rho = \frac{\log r}{\log \epsilon r}$.

Something Notable

- We expect documents x and y to have the same (Jaccard) similarity as is the similarity of their signatures

Why

Remember

- For $0 < \epsilon, r < 1$, there exists an algorithm for $(r, \epsilon r)$-PLEB under D using $O\left(dn + n^{1+\rho}\right)$ space and $O\left(n^{\rho}\right)$ evaluations for each query, where $\rho = \frac{\log r}{\log \epsilon r}$.

Something Notable

- We expect documents x and y to have the same (Jaccard) similarity as is the similarity of their signatures

Outline

Locality Sensitive Hashing for Min-Hash

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

Locality Sensitive Hashing for Min-Hash

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

Locality Sensitive Hashing for Min-Hash

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

Outline

Basically

From the main Theorem with $\rho = \frac{\log \frac{1}{p_1}}{\log \frac{1}{p_2}}$

$$P\left(p \in \mathsf{B}(q, r_1) \text{ then } g_j\left(p\right) = g_j\left(q\right), \text{ for some } j = 1, ..., l\right) \geq 1 - \left(1 - n^{-\rho}\right)^l$$

Given that (Under $p_1 > p_2$)

$$n^{-\rho} = p_1^k$$

Basically

From the main Theorem with $\rho = \frac{\log \frac{1}{p_1}}{\log \frac{1}{p_2}}$

$$P\left(p \in \mathrm{B}(q, r_1) \text{ then } g_j\left(p\right) = g_j\left(q\right), \text{ for some } j = 1, ..., l\right) \geq 1 - \left(1 - n^{-\rho}\right)^l$$

Given that (Under $p_1 > p_2$)

$$n^{-\rho} = p_1^k$$

Therefore

Then, if each signature is split in l bands and k bits

- Then, we have that two signatures at a certain band are equal with probability greater than a certain threshold s:

$$P\left(\text{All elements at the band ae equal}\right) \geq (s)^k$$

Then

- We need to play with l and r to reach our objectives.

Therefore

Then, if each signature is split in l bands and k bits

- Then, we have that two signatures at a certain band are equal with probability greater than a certain threshold s:

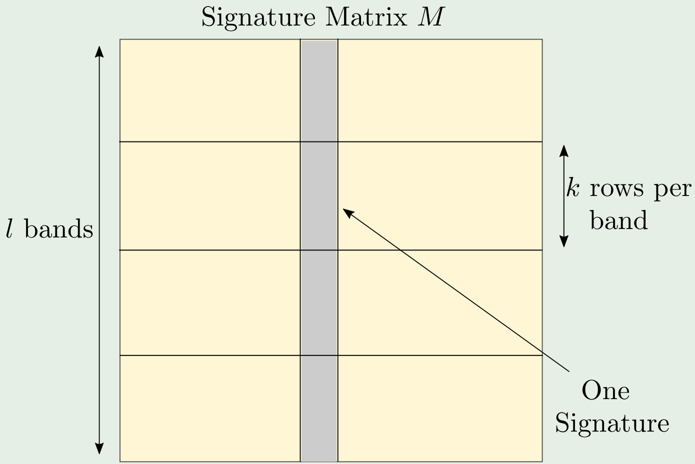
$$P\left(\text{All elements at the band ae equal}\right) \geq (s)^k$$

Then

- We need to play with l and r to reach our objectives.

Partition M into b Bands

Example



For this

Partition M into l Bands

- Divide matrix M into l bands of r rows.

For each band

- Hash its portion of each column to a hash table with k buckets.

Therefore

- Make k as large as possible

For this

Partition M into l Bands

- Divide matrix M into l bands of r rows.

For each band

- Hash its portion of each column to a hash table with k buckets.

Therefore

- Make k as large as possible

For this

Partition M into l Bands

- Divide matrix M into l bands of r rows.

For each band

- Hash its portion of each column to a hash table with k buckets.

Therefore

- Make k as large as possible

Then, the candidates have certain properties

Proposition

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

Then, the candidates have certain properties

Proposition

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

Catch most similar pairs

- Tune l and k to catch most similar pairs, but few non-similar pairs.

Then, the candidates have certain properties

Proposition

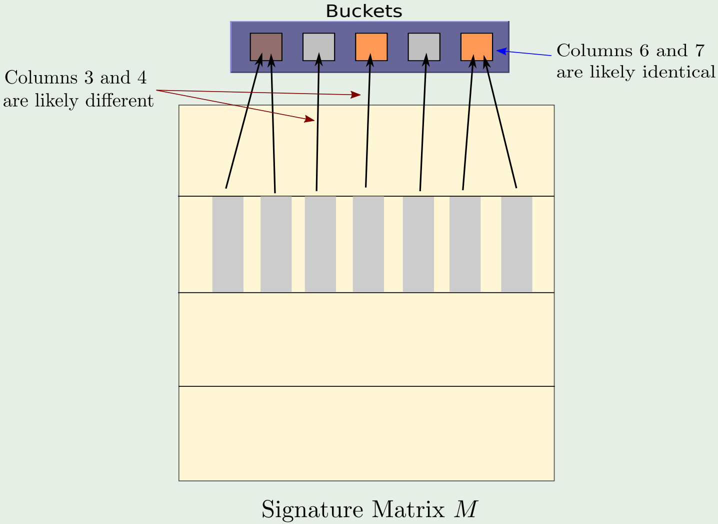
- Candidate column pairs are those that hash to the same bucket for ≥ 1 bands.

Catch most similar pairs

- Tune l and k to catch most similar pairs, but few non-similar pairs.

Hashing Bands

Example



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

Outline

Example of Bands

Assume the following case

- Suppose $100,000$ columns of M ($100,000$ documents)
- Signatures of 100 integers (rows) each integer taking 32 bits $= 4$ bytes
- Therefore, signatures can take around 38 Megabytes of Memory Space

Example of Bands

Assume the following case

- Suppose $100,000$ columns of M ($100,000$ documents)
- Signatures of 100 integers (rows) each integer taking 32 bits $=$ 4 bytes
- Therefore, signatures can take around 38 Megabytes of Memory Space

If we choose $l = 20$ bands of $k = 5$ integers/band, our objective is

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

Example of Bands

Assume the following case

- Suppose $100,000$ columns of M $(100,000$ documents)
- Signatures of 100 integers (rows) each integer taking 32 bits = 4 bytes
- Therefore, signatures can take around 38 Megabytes of Memory Space

If we choose $l = 20$ bands of $k = 5$ integers/band, our objective is

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

Example of Bands

Assume the following case

- Suppose $100,000$ columns of M $(100,000$ documents)
- Signatures of 100 integers (rows) each integer taking 32 bits $=$ 4 bytes
- Therefore, signatures can take around 38 Megabytes of Memory Space

If we choose $l = 20$ bands of $k = 5$ integers/band, our objective is

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

Now, if C_1,C_2 have a high 80% similarity

Find pairs of $\geq s = 0.8$ similarity

- set $l = 20$ and $k = 5$

If $sim(C_1, C_2) = 0.8$

- 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

- We have that the probability C_1, C_2 are identical in one particular band l_i is

$$P\left(C_1^{l_{i1}} = C_2^{l_{i1}}, \ldots, C_1^{l_{ik}} = C_2^{l_{ik}}\right) = \prod_{j=1}^{k} P\left(C_1^{l_{ij}} = C_2^{l_{ij}}\right) = (0.8)^5 = 0.328$$

Now, if C_1,C_2 have a high 80% similarity

Find pairs of $\geq s = 0.8$ similarity

- set $l = 20$ and $k = 5$

If $sim\left(C_1, C_2\right) = 0.8$

- 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

- We have that the probability C_1, C_2 are identical in one particular band l_i is

$$P\left(C_1^{l_{i1}} = C_2^{l_{i1}}, ..., C_1^{l_{ik}} = C_2^{l_{ik}}\right) = \prod_{j=1}^{k} P\left(C_1^{l_{ij}} = C_2^{l_{ij}}\right) = (0.8)^5 = 0.328$$

Now, if C_1,C_2 have a high 80% similarity

Find pairs of $\geq s = 0.8$ similarity

- set $l = 20$ and $k = 5$

If $sim(C_1, C_2) = 0.8$

- 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

- We have that the probability C_1, C_2 are identical in one particular band l_i is

$$P\left(C_1^{l_{i1}} = C_2^{l_{i1}}, ..., C_1^{l_{ik}} = C_2^{l_{ik}}\right) = \prod_{j=1}^{k} P\left(C_1^{l_{ij}} = C_2^{l_{ij}}\right) = (0.8)^5 = 0.328$$

What is the Probability of not being similar at all?

We use the complement to answer that over $l = 20$

$$P\left(C_1^{l_{i1}} \neq C_2^{l_{i1}}, ..., C_1^{l_{ik}} \neq C_2^{l_{ik}}\right) = \left[1 - \prod_{j=1}^{k} P\left(C_1^{l_{ij}} = C_2^{l_{ij}}\right)\right]^{20}$$

Thus, we have that

$$P\left(C_1^{l_{i1}} \neq C_2^{l_{i1}}, ..., C_1^{l_{ik}} \neq C_2^{l_{ik}}\right) = 0.00035$$

What is the Probability of not being similar at all?

We use the complement to answer that over $l = 20$

$$P\left(C_1^{l_{i1}} \neq C_2^{l_{i1}}, ..., C_1^{l_{ik}} \neq C_2^{l_{ik}}\right) = \left[1 - \prod_{j=1}^{k} P\left(C_1^{l_{ij}} = C_2^{l_{ij}}\right)\right]^{20}$$

Thus, we have that

$$P\left(C_1^{l_{i1}} \neq C_2^{l_{i1}}, ..., C_1^{l_{ik}} \neq C_2^{l_{ik}}\right) = 0.00035$$

Meaning

We have that

- About $\left(\frac{1}{3000}\right)^{th}$ of the 80% similar column pairs are false negatives i.e. we miss them

But, and this is important

- We would find 99.965% pairs of truly similar documents

Meaning

We have that

- About $\left(\frac{1}{3000}\right)^{th}$ of the 80% similar column pairs are false negatives i.e. we miss them

But, and this is important

- We would find 99.965% pairs of truly similar documents

Now, if C_1,C_2 have a low 30% similarity

Find pairs of $\geq s = 0.3$ similarity

- set $l = 20$ and $k = 5$

If $sim(C_1, C_2) = 0.3$

- 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

- We have that the probability C_1, C_2 are identical in one particular band l_i is

$$P\left(C_1^{l_{i1}} = C_2^{l_{i1}}, ..., C_1^{l_{ik}} = C_2^{l_{ik}}\right) = \prod_{j=1}^{k} P\left(C_1^{l_{ij}} = C_2^{l_{ij}}\right) = (0.3)^5 = 0.00243$$

Now, if C_1,C_2 have a low 30% similarity

Find pairs of $\geq s = 0.3$ similarity

- set $l = 20$ and $k = 5$

If $sim\left(C_1, C_2\right) = 0.3$

- 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

- We have that the probability C_1, C_2 are identical in one particular band l_i is

$$P\left(C_1^{l_{i1}} = C_2^{l_{i1}}, ..., C_1^{l_{ik}} = C_2^{l_{ik}}\right) = \prod_{j=1}^{k} P\left(C_1^{l_{ij}} = C_2^{l_{ij}}\right) = (0.3)^5 = 0.00243$$

Now, if C_1,C_2 have a low 30% similarity

Find pairs of $\geq s = 0.3$ similarity

- set $l = 20$ and $k = 5$

If $sim\left(C_1, C_2\right) = 0.3$

- 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

- We have that the probability C_1, C_2 are identical in one particular band l_i is

$$P\left(C_1^{l_{i1}} = C_2^{l_{i1}}, ..., C_1^{l_{ik}} = C_2^{l_{ik}}\right) = \prod_{j=1}^{k} P\left(C_1^{l_{ij}} = C_2^{l_{ij}}\right) = (0.3)^5 = 0.00243$$

cinvestav

What is the Probability of not being similar at all?

We use the complement to answer that over $l = 20$

$$P\left(C_1^{l_{i1}} \neq C_2^{l_{i1}}, ..., C_1^{l_{ik}} \neq C_2^{l_{ik}}\right) = \left[1 - \prod_{j=1}^{k} P\left(C_1^{l_{ij}} = C_2^{l_{ij}}\right)\right]^{20}$$

Thus, we have that

$$P\left(C_1^{l_{i1}} \neq C_2^{l_{i1}}, ..., C_1^{l_{ik}} \neq C_2^{l_{ik}}\right) = 0.0474$$

What is the Probability of not being similar at all?

We use the complement to answer that over $l = 20$

$$P\left(C_1^{l_{i1}} \neq C_2^{l_{i1}}, ..., C_1^{l_{ik}} \neq C_2^{l_{ik}}\right) = \left[1 - \prod_{j=1}^{k} P\left(C_1^{l_{ij}} = C_2^{l_{ij}}\right)\right]^{20}$$

Thus, we have that

$$P\left(C_1^{l_{i1}} \neq C_2^{l_{i1}}, ..., C_1^{l_{ik}} \neq C_2^{l_{ik}}\right) = 0.0474$$

Meaning

We have that

- 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.

Meaning

We have that

- 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.

Outline

Locality Sensitive Hashing Involves a Trade-off

You need to pick

- The number of Min-Hashes (rows of M).
- The number of bands l.
- The number of rows k per band to balance false positives/negatives.

Locality Sensitive Hashing Involves a Trade-off

You need to pick

- The number of Min-Hashes (rows of M).
- The number of bands l.
- The number of rows k 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

Locality Sensitive Hashing Involves a Trade-off

You need to pick

- The number of Min-Hashes (rows of M).
- The number of bands l.
- The number of rows k 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

Locality Sensitive Hashing Involves a Trade-off

You need to pick

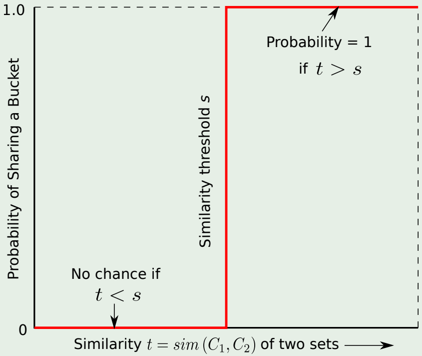
- The number of Min-Hashes (rows of M).
- The number of bands l.
- The number of rows k per band to balance false positives/negatives.

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

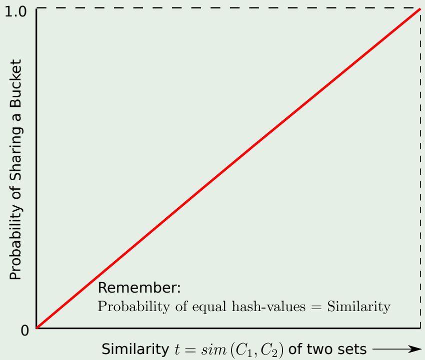
Analysis of Locality Sensitive Hashing - What We Want

The Ideal detection of similar objects



What One Band of One Row Gives You

Not Great at ALL



Given that probability of two documents agree 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^k.
2. The probability that the signatures disagree in at least one row of a particular band is $1 - s^k$.
3. The probability that the signatures disagree in at least one row of each of the bands is $\left(1 - s^k\right)^l$.
4. The probability that the signatures agree in all the rows of at least one band, and therefore become a candidate pair, is $1 - \left(1 - s^k\right)^l$.

Given that probability of two documents agree 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^k.
2. The probability that the signatures disagree in at least one row of a particular band is $1 - s^k$.
3. The probability that the signatures disagree in at least one row of each of the bands is $\left(1 - s^k\right)^l$.
4. The probability that the signatures agree in all the rows of at least one band, and therefore become a candidate pair, is $1 - \left(1 - s^k\right)^l$.

Given that probability of two documents agree 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^k.
2. The probability that the signatures disagree in at least one row of a particular band is $1 - s^k$.
3. The probability that the signatures disagree in at least one row of each of the bands is $\left(1 - s^k\right)^l$.
4. The probability that the signatures agree in all the rows of at least one band, and therefore become a candidate pair, is $1 - \left(1 - s^k\right)^l$.

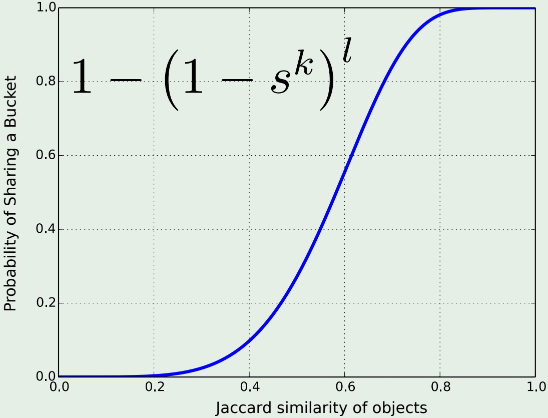
Given that probability of two documents agree in a row is s

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

If you fix k and l

Something Notable



Example: $l = 20$; $k = 5$

Given

- Similarity threshold s

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

s	$1 - \left(1 - s^k\right)^l$
.2	0.006
.3	0.047
.4	0.186
.5	0.470
.6	0.802
.7	0.975
.8	0.9996

Example: $l = 20$; $k = 5$

Given

- Similarity threshold s

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

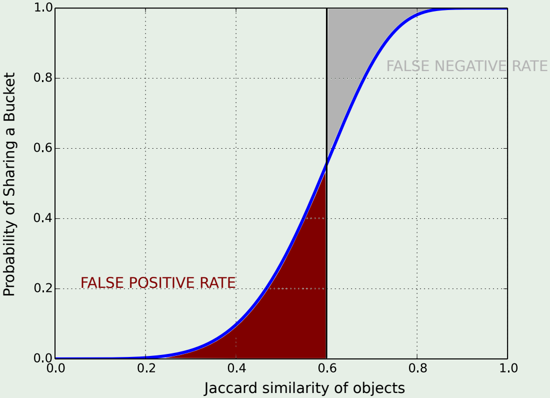
s	$1 - \left(1 - s^k\right)^l$
.2	0.006
.3	0.047
.4	0.186
.5	0.470
.6	0.802
.7	0.975
.8	0.9996

cinvestav

Picking k and l: The S-curve

Picking k and l to get the best S-curve

- For example, for 50 hash-functions $(k = 5, l = 10)$



Locality Sensitive Hashing, a Brief Summary

Tune M, l, k

- Tune M, l, k 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

Locality Sensitive Hashing, a Brief Summary

Tune M, l, k

- Tune M, l, k 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

Locality Sensitive Hashing, a Brief Summary

Tune M, l, k

- Tune M, l, k 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

Outline

The Final Pipeline

Convert Objects using Vector Shingling Representation

- Convert Objects into sets via shingling

The Final Pipeline

Convert Objects using Vector Shingling Representation

- Convert Objects into sets via shingling

Convert large sets to short signatures, while preserving similarity using Min-hashing

- Use similarity preserving hashing to generate signatures with property

$$Pr\left[h_{\pi}\left(C_1\right) = h_{\pi}\left(C_2\right)\right] = sim\left(C_1, C_2\right).$$

- Use hashing to get around generating random permutations.

The Final Pipeline

Convert Objects using Vector Shingling Representation

- Convert Objects into sets via shingling

Convert large sets to short signatures, while preserving similarity using Min-hashing

- Use similarity preserving hashing to generate signatures with property

$$Pr\left[h_{\pi}\left(C_1\right) = h_{\pi}\left(C_2\right)\right] = sim\left(C_1, C_2\right).$$

- Use hashing to get around generating random permutations.

Finally

Locality-Sensitive Hashing

- Them, focus on pairs of signatures that are likely to be from similar documents.
  - Use hashing to find candidate pairs of similarity $\geq s$

Finally

Locality-Sensitive Hashing

- Them, focus on pairs of signatures that are likely to be from similar documents.
  - Use hashing to find candidate pairs of similarity $\geq s$

Y. Gong, S. Lazebnik, A. Gordo, and F. Perronnin, "Iterative quantization: A procrustean approach to learning binary codes for large-scale image retrieval," *IEEE transactions on pattern analysis and machine intelligence*, vol. 35, no. 12, pp. 2916–2929, 2012.

M. Muja and D. G. Lowe, "Scalable nearest neighbor algorithms for high dimensional data," *IEEE transactions on pattern analysis and machine intelligence*, vol. 36, no. 11, pp. 2227–2240, 2014.

P. Indyk and R. Motwani, "Approximate nearest neighbors: towards removing the curse of dimensionality," in *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pp. 604–613, 1998.

M. Levandowsky and D. Winter, "Distance between sets," *Nature*, vol. 234, no. 5323, pp. 34–35, 1971.