Locality Sensitive Hashing and its Application

Rice University

Anshumali Shrivastava

anshumali At rice.edu

31th August 2015

Pairwise Comparisons Everywhere

- Near Duplicate Detections over web. (mirror pages)
- Plagiarism Detection
- Find Customers With Similar Taste.
- Movie Recommendations. (Find Similar profiles)

Activity : Exact Duplicates

Remove all repeated items in an array
example {1,2,3,8,2,7,3,3,4,8,9}

Activity : Exact Duplicates

Remove all repeated items in an array
example {1,2,3,8,2,7,3,3,4,8,9}

$O(n)$ or $O(n^2)$

Activity : Exact Duplicates

Remove all repeated items in an array
example $\{1,2,3,8,2,7,3,3,4,8,9\}$

$O(n)$ or $O(n^2)$

Array of vectors instead of numbers ?

Documents as Sets

Given 3 short documents

- "Earth is the third planet"
- "USA is the third largest country"
- "Pluto is the nineth planet"

How do we mathematically represent documents and compare between them ?

Documents as Sets

Given 3 short documents

- "Earth is the third planet"
- "USA is the third largest country"
- "Pluto is the nineth planet"

How do we mathematically represent documents and compare between them ?

A very reasonable and practical idea

- Two documents with more words overlap are likely to be similar.
- Represent documents as set of words appearing in it. (Bag of Words)

Documents as Sets

Given 3 short documents

- "Earth is the third planet"
- "USA is the third largest country"
- "Pluto is the nineth planet"

How do we mathematically represent documents and compare between them ?

A very reasonable and practical idea

- Two documents with more words overlap are likely to be similar.
- Represent documents as set of words appearing in it. (Bag of Words)

Problems

- Different but similar meaning words (synonyms) ?
- Order information ?

Better Representation: *k*-Shingles

Definition

- A document is a string.
- k-shingles is the set of all length k substrings that appear one or more times within that document. (character k-grams)
- **Popular Variant:** Treat words as basic tokens. (word k-grams)

Example 1: Document "abc dab d" for k =2.
The set of 2-shingles is {ab, bc, c , d, da, b , d}.

Example 2: Document "This is Rice University" for k =2.
The set of 2-word grams is {This is, is Rice, Rice University}.

Bottom Line: Documents can be reasonably represented as sets.

Better Representation: *k*-Shingles

Definition

- A document is a string.
- k-shingles is the set of all length k substrings that appear one or more times within that document. (character k-grams)
- **Popular Variant:** Treat words as basic tokens. (word k-grams)

Example 1: Document "abc dab d" for k =2.
The set of 2-shingles is {ab, bc, c , d, da, b , d}.

Example 2: Document "This is Rice University" for k =2.
The set of 2-word grams is {This is, is Rice, Rice University}.

Bottom Line: Documents can be reasonably represented as sets.

What are the universal sets in these examples ?

Jaccard Similarity

The popular **resemblance (Jaccard) similarity** between two sets $X,\ Y \subset \Omega$ is defined as:

$$\mathcal{R} = \frac{|X \cap Y|}{|X \cup Y|} = \frac{a}{f_x + f_y - a},$$

where $a = |X \cap Y|$, $f_x = |X|$, $f_y = |Y|$ and $|.|$ denotes the cardinality.

Jaccard Similarity

The popular **resemblance (Jaccard) similarity** between two sets $X, \ Y \subset \Omega$ is defined as:

$$\mathcal{R} = \frac{|X \cap Y|}{|X \cup Y|} = \frac{a}{f_x + f_y - a},$$

where $a = |X \cap Y|$, $f_x = |X|$, $f_y = |Y|$ and $|.|$ denotes the cardinality.

Question: Why not just the intersection $|X \cap Y|$?

Jaccard Similarity

The popular **resemblance (Jaccard) similarity** between two sets $X,\ Y \subset \Omega$ is defined as:

$$\mathcal{R} = \frac{|X \cap Y|}{|X \cup Y|} = \frac{a}{f_x + f_y - a},$$

where $a = |X \cap Y|$, $f_x = |X|$, $f_y = |Y|$ and $|.|$ denotes the cardinality.

Question: Why not just the intersection $|X \cap Y|$?

Sets $\iff$ Binary Vectors

$$a = |X \cap Y| = x^T y; \quad f_x = nonzeros(x); \quad f_y = nonzeros(y),$$

where x and y are the binary vector equivalents of sets X and Y respectively.

Cosine Similarity

Cosine similarity between two sets $X, \ Y \subset \Omega$ is defined as:

$$\mathcal{R} = \frac{|X \cap Y|}{\sqrt{|X||Y|}} = \frac{a}{\sqrt{f_x f_y}},$$

where $a = |X \cap Y|$, $f_x = |X|$, $f_y = |Y|$ and $|.|$ denotes the cardinality.

Recent Results: Cosine and Jaccard only differs in normalization.

- Both are distortions of each other.
- We actually don't need two, doing good on any one is enough.
- Check "Shrivastava and Li *In Defense of Minhash over Simhash* AISTATS 2014"

So Far

- Shingle Representation
- Documents as sets
- Two popular similarities over sets
  - Jaccard Similarity
  - Cosine Similarity

Subroutine of Interest : Similarity Search

Given a query $q \in \mathbb{R}^D$ and a giant collection $\mathcal{C}$ of N vectors in $\mathbb{R}^D$, search for $p \in \mathcal{C}$ s.t.,

$$p = \arg\max_{x \in \mathcal{C}} \quad sim(q, x)$$

Subroutine of Interest : Similarity Search

Given a query $q \in \mathbb{R}^D$ and a giant collection $\mathcal{C}$ of N vectors in $\mathbb{R}^D$, search for $p \in \mathcal{C}$ s.t.,

$$p = \arg\max_{x \in \mathcal{C}} \quad sim(q, x)$$

- *sim* is the similarity, like Cosine Similarity, Resemblance, etc.
- Worst case $O(N)$ for any query. N is huge.
- Querying is a very frequent operation.

Subroutine of Interest : Similarity Search

Given a query $q \in \mathbb{R}^D$ and a giant collection $\mathcal{C}$ of N vectors in $\mathbb{R}^D$, search for $p \in \mathcal{C}$ s.t.,

$$p = \arg\max_{x \in \mathcal{C}} \quad sim(q, x)$$

- *sim* is the similarity, like Cosine Similarity, Resemblance, etc.
- Worst case $O(N)$ for any query. N is huge.
- Querying is a very frequent operation.

Our goal is to find sub-linear query time algorithm.

Subroutine of Interest : Similarity Search

Given a query $q \in \mathbb{R}^D$ and a giant collection $\mathcal{C}$ of N vectors in $\mathbb{R}^D$, search for $p \in \mathcal{C}$ s.t.,

$$p = \arg\max_{x \in \mathcal{C}} \; sim(q, x)$$

- *sim* is the similarity, like Cosine Similarity, Resemblance, etc.
- Worst case $O(N)$ for any query. N is huge.
- Querying is a very frequent operation.

Our goal is to find sub-linear query time algorithm.

1. Approximate answer suffices.
2. We are allowed to pre-process $\mathcal{C}$ once. (offline costly step)

Locality Sensitive Hashing

Hashing: Function (randomized) h that maps a given data vector $x \in \mathbb{R}^D$ to an integer key $h : \mathbb{R}^D \mapsto \{0, 1, 2, ..., N\}$

Locality Sensitive Hashing

Hashing: Function (randomized) h that maps a given data vector $x \in \mathbb{R}^D$ to an integer key $h : \mathbb{R}^D \mapsto \{0, 1, 2, ..., N\}$

Locality Sensitive: Additional property

$$Pr_h\big[h(x) = h(y)\big] = f(sim(x, y)),$$

where f is monotonically increasing. *sim* is any similarity of interest.

Locality Sensitive Hashing

Hashing: Function (randomized) h that maps a given data vector $x \in \mathbb{R}^D$ to an integer key $h : \mathbb{R}^D \mapsto \{0, 1, 2, ..., N\}$

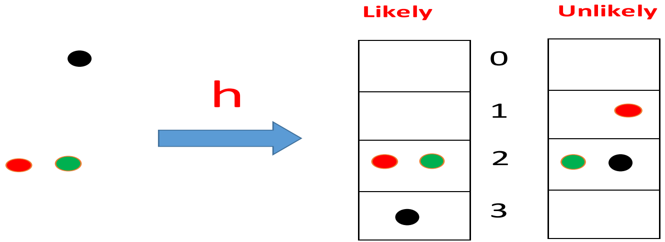
Locality Sensitive: Additional property

$$Pr_h\left[h(x) = h(y)\right] = f(sim(x, y)),$$

where f is monotonically increasing. *sim* is any similarity of interest.

Similar points are more likely to have the same hash value (hash collision).

Question: Does this definition implies the definition given in the book ?



Minwise Hashing

A random permutation π is performed on Ω, i.e.,
$\pi: \ \Omega \longrightarrow \Omega, \qquad \text{where} \quad \Omega = \{0, 1, ..., D-1\}$. is the universal set

For $S_1, S_2 \subset \Omega$ we always have

$$\mathbf{Pr}\left(\min(\pi(S_1)) = \min(\pi(S_2))\right) = \frac{|S_1 \cap S_2|}{|S_1 \cup S_2|} = R \quad (\textbf{Jaccard Similarity.}).$$

Example:
$D = 5$. $S_1 = \{0, 3, 4\}$, $S_2 = \{1, 2, 3\}$, $R = \frac{|S_1 \cap S_2|}{|S_1 \cup S_2|} = \frac{1}{5}$.
One realization of the permutation π can be

$$0 \Longrightarrow 3 \quad 1 \Longrightarrow 2 \quad 2 \Longrightarrow 0 \quad 3 \Longrightarrow 4 \quad 4 \Longrightarrow 1$$

$$\pi(S_1) = \{3, 4, 1\}, \qquad \pi(S_2) = \{2, 0, 4\}$$

In this example, $\min(\pi(S_1)) \neq \min(\pi(S_2))$.

Minwise Hashing: Example Binary Vectors

1. Uniformly sample a permutation over attributes $\pi : [0, D] \mapsto [0, D]$.
2. Shuffle the vectors under π.
3. The hash value is **smallest index which is not zero**.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
S_1:	0	1	0	0	1	1	0	0	1	0	0	0	0	0	0	0
S_2:	0	0	0	0	0	0	0	0	1	0	1	0	1	0	1	0
S_3:	0	0	0	1	0	0	1	1	0	0	0	0	0	0	1	0

Minwise Hashing: Example Binary Vectors

1. Uniformly sample a permutation over attributes $\pi : [0, D] \mapsto [0, D]$.
2. Shuffle the vectors under π.
3. The hash value is **smallest index which is not zero**.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
S_1:	0	1	0	0	1	1	0	0	1	0	0	0	0	0	0	0
S_2:	0	0	0	0	0	0	0	0	1	0	1	0	1	0	1	0
S_3:	0	0	0	1	0	0	1	1	0	0	0	0	0	0	1	0

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$\pi(S_1)$:	0	0	1	0	1	0	0	1	0	0	0	0	0	1	0	0
$\pi(S_2)$:	1	0	0	1	0	0	1	0	0	0	0	0	0	1	0	0
$\pi(S_3)$:	1	1	0	0	0	0	0	0	0	0	1	0	1	0	0	0

Minwise Hashing: Example Binary Vectors

1. Uniformly sample a permutation over attributes $\pi : [0, D] \mapsto [0, D]$.
2. Shuffle the vectors under π.
3. The hash value is **smallest index which is not zero**.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
S_1:	0	1	0	0	1	1	0	0	1	0	0	0	0	0	0	0
S_2:	0	0	0	0	0	0	0	0	1	0	1	0	1	0	1	0
S_3:	0	0	0	1	0	0	1	1	0	0	0	0	0	0	1	0

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$\pi(S_1)$:	0	0	1	0	1	0	0	1	0	0	0	0	0	1	0	0
$\pi(S_2)$:	1	0	0	1	0	0	1	0	0	0	0	0	0	1	0	0
$\pi(S_3)$:	1	1	0	0	0	0	0	0	0	0	1	0	1	0	0	0

$$h_\pi(S_1) = 2, \quad h_\pi(S_2) = 0, \quad h_\pi(S_3) = 0$$

Minwise Hashing: Example Binary Vectors

1. Uniformly sample a permutation over attributes $\pi : [0, D] \mapsto [0, D]$.
2. Shuffle the vectors under π.
3. The hash value is **smallest index which is not zero**.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
S_1:	0	1	0	0	1	1	0	0	1	0	0	0	0	0	0	0
S_2:	0	0	0	0	0	0	0	0	1	0	1	0	1	0	1	0
S_3:	0	0	0	1	0	0	1	1	0	0	0	0	0	0	1	0

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$\pi(S_1)$:	0	0	1	0	1	0	0	1	0	0	0	0	0	1	0	0
$\pi(S_2)$:	1	0	0	1	0	0	1	0	0	0	0	0	0	1	0	0
$\pi(S_3)$:	1	1	0	0	0	0	0	0	0	0	1	0	1	0	0	0

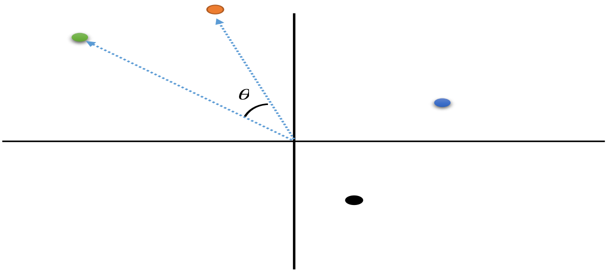
$$h_\pi(S_1) = 2, \quad h_\pi(S_2) = 0, \quad h_\pi(S_3) = 0$$

For any two binary vectors S_1, S_2 we always have

$$\mathbf{Pr}\left(h_\pi(S_1) = h_\pi(S_2)\right) = \frac{|S_1 \cap S_2|}{|S_1 \cup S_2|} = R \quad (\textbf{Jaccard Similarity.}).$$

Proof (On Board)

Signed Random Projections (SimHash)

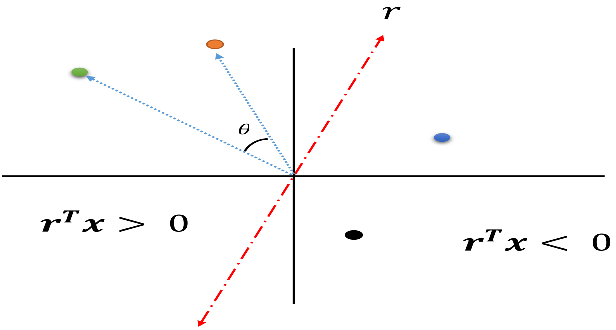


$$h_r(x) = \begin{cases} 1 & \text{if } r^T x \geq 0 \\ 0 & \text{otherwise} \end{cases} \qquad r \in \mathbb{R}^D \sim N(0, \mathcal{I})$$

$$Pr_r(h_r(x) = h_r(y)) = 1 - \frac{\theta}{\pi}, \quad \text{monotonic in cosine similarity } \theta = cos^{-1}\mathcal{S}$$

A classical result from Goemans-Williamson (95)

Signed Random Projections (SimHash)



$$h_r(x) = \begin{cases} 1 & \text{if } r^T x \geq 0 \\ 0 & \text{otherwise} \end{cases} \qquad r \in \mathbb{R}^D \sim N(0, \mathcal{I})$$

$$Pr_r(h_r(x) = h_r(y)) = 1 - \frac{\theta}{\pi}, \quad \text{monotonic in cosine similarity } \theta = cos^{-1}\mathcal{S}$$

A classical result from Goemans-Williamson (95)

For Binay Data, MinHash is better than SimHash

Recent Results: Cosine and Jaccard only differs in normalization.

- Both similarities are distortions of each other.
- For Binary Data, MinHash is more informative and better for similarity search and estimation compared to SimHash.
- Check "Shrivastava and Li *In Defense of Minhash over Simhash* AISTATS 2014"

LSH for Estimation

We have

$$Pr_h\Big[h(x) = h(y)\Big] = f(sim(x, y)),$$

where f is monotonically increasing.

LSH for Estimation

We have

$$Pr_h\big[h(x) = h(y)\big] = f(sim(x, y)),$$

where f is monotonically increasing.

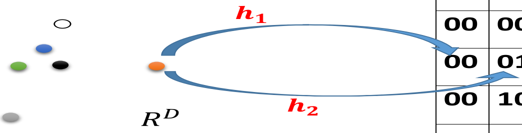
Activity: Design a strategy for estimating $sim(x, y)$ given access to values of $h(x)$ and $h(y)$, with h sampled independently.

Sub-linear Near Neighbor Search: Idea

Given: $Pr_h\left[h(x) = h(y)\right] = f(sim(x, y))$, f is monotonic.

Sub-linear Near Neighbor Search: Idea

Given: $Pr_h\left[h(x) = h(y)\right] = f(sim(x, y))$, f is monotonic.

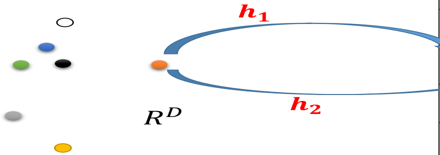


$h_1, h_2 : R^D \rightarrow \{0, 1, 2, 3\}$

h_1	h_2	Buckets (pointers only)
00	00	
00	01	
00	10	
...	...	
11	11	

Sub-linear Near Neighbor Search: Idea

Given: $Pr_h\left[h(x) = h(y)\right] = f(sim(x, y))$, f is monotonic.

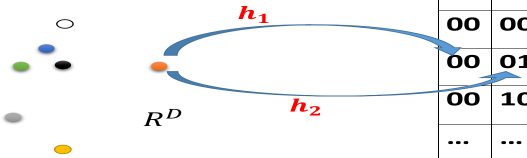


$h_1, h_2 : R^D \rightarrow \{0, 1, 2, 3\}$

h_1	h_2	Buckets (pointers only)
00	00	...
00	01	...
00	10	Empty
...	...	...
11	11	...

Sub-linear Near Neighbor Search: Idea

Given: $Pr_h\left[h(x) = h(y)\right] = f(sim(x, y))$, f is monotonic.



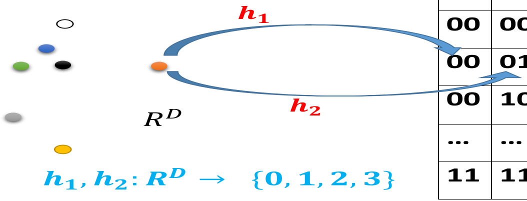
$h_1, h_2 : R^D \rightarrow \{0, 1, 2, 3\}$

h_1	h_2	Buckets (pointers only)
00	00	...
00	01	...
00	10	Empty
...	...	...
11	11	...

- Given query q, if $h_1(q) = 11$ **and** $h_2(q) = 01$, then probe bucket with index **1101**. It is a good bucket !!

Sub-linear Near Neighbor Search: Idea

Given: $Pr_h\left[h(x) = h(y)\right] = f(sim(x, y))$, f is monotonic.



h_1	h_2	Buckets (pointers only)
00	00	...
00	01	...
00	10	Empty
...	...	...
11	11	...

- Given query q, if $h_1(q) = 11$ **and** $h_2(q) = 01$, then probe bucket with index **1101**. It is a good bucket !!
- (Locality Sensitive) $h_i(q) = h_i(x)$ implies high similarity.
- Doing better than random !!

The Classical LSH Algorithm

Table 1

h_1^1	...	h_K^1	Buckets
00	...	00	...
00	...	01	...
00	...	10	Empty
...	...	...	...
11	...	11	...

- We use K concatenation.

The Classical LSH Algorithm

Table 1

h_1^1	...	h_K^1	Buckets
00	...	00	...
00	...	01	...
00	...	10	Empty
...	...	...	...
11	...	11	...

...

Table L

h_1^L	...	h_K^L	Buckets
00	...	00	...
00	...	01	...
00	...	10	
...	...	...	...
11	...	11	Empty

- We use K concatenation.
- Repeat the process L times. (L Independent Hash Tables)

The Classical LSH Algorithm

Table 1

h_1^1	...	h_K^1	Buckets
00	...	00	...
00	...	01	...
00	...	10	Empty
...	...	...	...
11	...	11	...

...

Table L

h_1^L	...	h_K^L	Buckets
00	...	00	...
00	...	01	...
00	...	10	
...	...	...	...
11	...	11	Empty

- We use K concatenation.
- Repeat the process L times. (L Independent Hash Tables)
- **Querying :** Probe one bucket from each of L tables. Report union.

The Classical LSH Algorithm

Table 1

h_1^1	...	h_K^1	Buckets
00	...	00	...
00	...	01	...
00	...	10	Empty
...	...	...	...
11	...	11	...

...

Table L

h_1^L	...	h_K^L	Buckets
00	...	00	...
00	...	01	...
00	...	10	
...	...	...	...
11	...	11	Empty

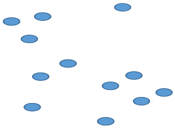
- We use K concatenation.
- Repeat the process L times. (L Independent Hash Tables)
- **Querying :** Probe one bucket from each of L tables. Report union.

1. Two knobs K and L to control.
2. **Theory says we have a sweet spot.** Provable sub-linear algorithm. (Indyk & Motwani 98)

A Real Problem: Avoiding Quadratic

Dataset of around 250,000 Syrian death records from 7 sources.

- A very short noisy text description of who died.
- Arabic suffixes and prefixes have many ambiguities.
- Selection biases.



A Real Problem: Avoiding Quadratic

Dataset of around 250,000 Syrian death records from 7 sources.

- A very short noisy text description of who died.
- Arabic suffixes and prefixes have many ambiguities.
- Selection biases.

A Real Problem: Avoiding Quadratic

Dataset of around 250,000 Syrian death records from 7 sources.

- A very short noisy text description of who died.
- Arabic suffixes and prefixes have many ambiguities.
- Selection biases.

Many records correspond to the same individual.

Problem: Can we estimate how many people died ? (**Record Linkage**)

A Real Problem: Avoiding Quadratic

Dataset of around 250,000 Syrian death records from 7 sources.

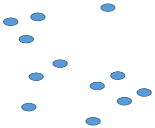
- A very short noisy text description of who died.
- Arabic suffixes and prefixes have many ambiguities.
- Selection biases.

Many records correspond to the same individual.

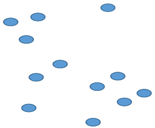
Problem: Can we estimate how many people died ? (**Record Linkage**)

Reasonable Idea: Try predicting match/mismatch given a pair.
Concern: Just too many pairs ! (3.1×10^{10})

Reducing Potential Pairs via Hashing



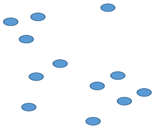
Reducing Potential Pairs via Hashing



h_1	h_2	Buckets (pointers only)
00	00	...
00	01	...
00	10	Empty
...	...	...
11	11	...

h_3	h_4	Buckets (pointers only)
00	00	...
00	01	...
00	10	
...	...	...
11	11	Empty

Reducing Potential Pairs via Hashing

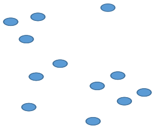


h_1	h_2	Buckets (pointers only)
00	00	...
00	01	...
00	10	Empty
...	...	...
11	11	...

h_3	h_4	Buckets (pointers only)
00	00	...
00	01	...
00	10	
...	...	...
11	11	Empty

- Co-occurrence in bucket mean high resemblance between records.

Reducing Potential Pairs via Hashing

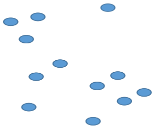


h_1	h_2	Buckets (pointers only)
00	00	...
00	01	...
00	10	Empty
...	...	...
11	11	...

h_3	h_4	Buckets (pointers only)
00	00	...
00	01	...
00	10	
...	...	...
11	11	Empty

- Co-occurrence in bucket mean high resemblance between records.
- Only form pairs within each bucket.

Reducing Potential Pairs via Hashing

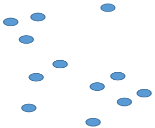


h_1	h_2	Buckets (pointers only)
00	00	...
00	01	...
00	10	Empty
...	...	...
11	11	...

h_3	h_4	Buckets (pointers only)
00	00	...
00	01	...
00	10	
...	...	...
11	11	Empty

- Co-occurrence in bucket mean high resemblance between records.
- Only form pairs within each bucket.
  1. All operations near linear.
  2. **99% recall** and only evaluate **1% of the total pairs**.

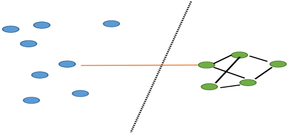
Reducing Potential Pairs via Hashing



h_1	h_2	Buckets (pointers only)
00	00	...
00	01	...
00	10	Empty
...	...	...
11	11	...

h_3	h_4	Buckets (pointers only)
00	00	...
00	01	...
00	10	
...	...	...
11	11	Empty

- Co-occurrence in bucket mean high resemblance between records.
- Only form pairs within each bucket.
  1. All operations near linear.
  2. **99% recall** and only evaluate **1% of the total pairs**.
- Connect to get a **sparse graph**. Graph cuts to reduce more.



Brain Strom Activity : Graph Matching !

- Given a collection of n graphs find a reasonable routine to remove isomorphic (identical or duplicates) graphs
- Assume you have an subroutine $isIsomorphic(G_1, G_2)$. Try to avoid quadratic call to this subroutine.

Brain Strom Activity : Graph Matching !

- Given a collection of n graphs find a reasonable routine to remove isomorphic (identical or duplicates) graphs
- Assume you have an subroutine $isIsomorphic(G_1, G_2)$. Try to avoid quadratic call to this subroutine.

Any real application ?