CSE 344
Lectures 26 and 28

Finding Similar Items

5/25/2011 and 6/1/2011

Announcements

- Today 5/25: Min-hashes (Ch. 22.3)
- Friday: guest lecturer YongChul Kwon on MR internals
- Monday 5/30: no classes
- Wednesday 6/1: LSH (Ch. 22.4)

Credits for this lecture

- Approximate String Joins in a Database (Almost) for Free, Luis Gravano et al., 2001
- A Primitive Operator for Similarity Joins in Data Cleaning, Chaudhuri, Ganti, Kaushik, 2006
- Efficient Exact Set-Similarity Joins, Arasu, Ganti, Kaushik, 2006
- Record Linkage Tutorial: Distance Metrics for Text, by William Choen

Problem Description

- Given two collections of strings, find pairs of similar strings

Application 1: Record Linkage

Relation 1

CName1	. . . Other attributes
Microsoft Corp	
Apple Computer	
Apples, Pears, and More	
. . .	

Relation 2

CName2	. . . Other attributes
Microsoft Inc	
Apple Corporation	
Apples and Pears Farm	
. . .	

Application 2: Similarity Join

```
SELECT *
FROM Company1, Company2
WHERE cname1 ≈ cname2
```

We want the strings to be "similar"

Same as record linkage, but done on-the-fly

Application 3: Collaborative Filtering

- We have n customers: 1, 2, …, n
- Each customer i buys a set of items S_i
- We would like to recommend items bought by customer j, if $S_i \approx S_j$

Example

S1 (customer 1)	S2	S3	S4	S5
Toothpaste floss	Floss mouthwash	Ipod PowerBook VideoAdapter	Ipod mouthwash	Floss toothpaste mouthwash

New customer buys

S = {mouthwash, floss, Ipad}

What do you recommend ?

Application 4: Similar Documents

- Given n documents 1, 2, …, n

- Want to find all pairs of "similar" documents, i.e. for which $S_i \approx S_j$

Example

- You work for a copyright violation detection company
- Customers: have documents 1, 2, 3, …., 10^6
- Web: has pages 1, 2, 3, …., 10^{11}

- Your job is to find "almost identical documents"
- What do you do ?

What is “Similar” ?

- Similarity function sim(S_1,S_2):
  - Sim(S_1,S_2) > k means s_1, s_2 are similar
- Distance function dist(S_1,S_2):
  - Dist(S_1,S_2) < k means S_1, S_2 are similar

Two Approaches

- Q-grams
  - Simple, and can lead to efficient optimizations
- Edit Distance
  - More accurate, but less amenable to optimizations

Q-Grams or Shingles

- Given a string S, a q-gram is a substring of length q
- Usually q = 3

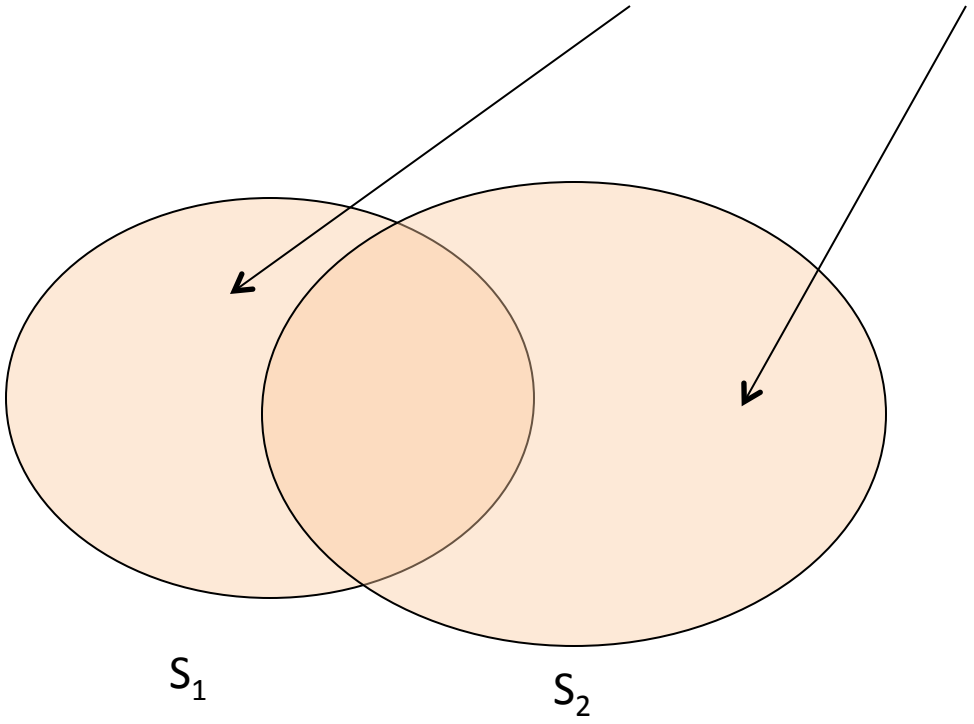
washington woshington

S_1 = {was, ash, shi, hin, ing, ngt, gto, ton}
S_2 = {wos, osh, shi, hin, ing, ngt, gto, ton}

Variation: may include beginning and end: ##w, #wa, on$, n$$

Hamming Distance

- $H(S_1 , S_2) = |S_1 \Delta S_2| = |S_1 - S_2| + |S_2 - S_1|$



"S_1 is similar to S_2" if $H(S_1, S_2) < c$, for some constant c

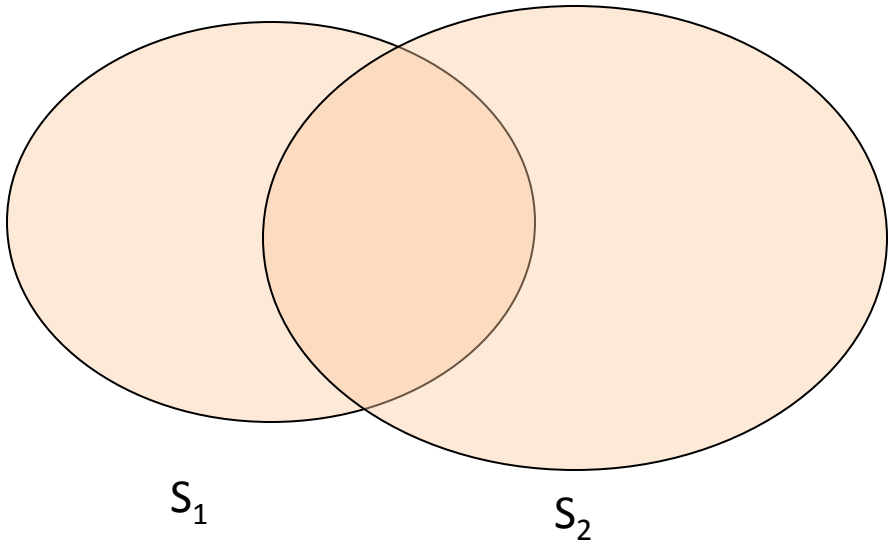
Hamming Distance

S_1 = {was, ash, shi, hin, ing, ngt, gto, ton}
S_2 = {wos, osh, shi, hin, ing, ngt, gto, ton}

$H(S_1, S_2)$ = ?

Jaccard Similarity

- $J(S_1, S_2) = |S_1 \cap S_2| / |S_1 \cup S_2|$



"S_1 is similar to S_2" if $J(S_1, S_2) > k$, for some constant k

Jaccard Similarity

S_1 = {was, ash, shi, hin, ing, ngt, gto, ton}
S_2 = {wos, osh, shi, hin, ing, ngt, gto, ton}

$J(S_1, S_2)$ = ?

They are related !

- Suppose $|S_1| = |S_2| = L$
- Denote $I(S_1, S_2) = |S_1 \cap S_2|$

- Then:
- $J(S_1, S_2) > k$ iff $I(S_1, S_2) > 2kL/(1+k)$

- $H(S_1, S_2) < c$ iff $I(S_1, S_2) > L - c/2$

WHY ?

Representing q-Grams

Company (id, name, …)

Id	Name	. . .
1	Washington	. . .
2	. . .	. . .

CQ(id, qgram)

Id	Qgram
1	was
1	ash
1	shi
1	hin
. . .	. . .
2	. . .

The q-gram schema

Naive Similarity Joins

```
SELECT *
FROM Company1, Company2
WHERE |cname1 ∩ cname2| > 6
```

Company1(id1, cname1, …)
Company2(id2, cname2, …)

CQ1(id1, qgram1)
CQ2(id2, qgram2)

Rewrite the query over the q-gram schema

Naive Similarity Joins

```
SELECT *
FROM Company1, Company2
WHERE |cname1 ∩ cname2| > 6
```

```
SELECT x.*, y.*
FROM Company1 x, Company2 y, CQ1 u, CQ2 v
WHERE x.id = u.id and y.id = v.id
  and u.qgram = v.qgram
GROUP BY x.id, y.id
HAVING count(*) > 6
```

Signatures

- The naïve method is exact, but slow
- Better method based on *signatures*
- A *signature* sig(s) is a set s.t.:
  $I(S_1, S_2) > k$ ➔ $sig(S_1) \cap sig(S_2) \neq$ emptyset
- New method:
  - Find all pairs S_1, S_2 for which $sig(S_1) \cap sig(S_2) \neq$ emptyset
  - Remove false positives by checking $I(S_1, S_2) > k$

Give examples of signatures !

Signature 1: The q-grams !

- Obviously: $I(S_1,S_2) > k$ ➔ $S_1 \cap S_2 \neq$ emptyset

```
SELECT DISTINCT x.*, y.*
FROM Company1 x, Company2 y, CQ1 u, CQ2 v
WHERE x.id = u.id and y.id = v.id
  and u.qgram = v.qgram
```

The query returns false positives, need to be removed

Signature 2: Prefix Filter

- Suppose all sets s_i have same cardinality L
- Order the q-grams lexicographically
- Define
  sig(S) = {the L-k+1 smallest q-grams in q}

Signature 2: Prefix Filter

- Example: L = 8, k = 6, L – k + 1 = 3

S_1 = {was, ash, shi, hin, ing, ngt, gto, ton}
= {ash, gto, hin, ing, ngt, shi, ton, was}
Sig(S_1) = {ash, gto, hin}

Signature 2: Prefix Filter

- Fact: if $|S_1 \cap S_2| > k$
  then $sig(S_1) \cap sig(S_2) \neq$ emptyset

- Why ?

Comments on Signatures

- More efficient signature schemes exists, but they are more complex and we won't discuss them
- In the implementation we need to also take into account $|S_1|$ and $|S_2|$, this adds to the complexity

Edit Distance

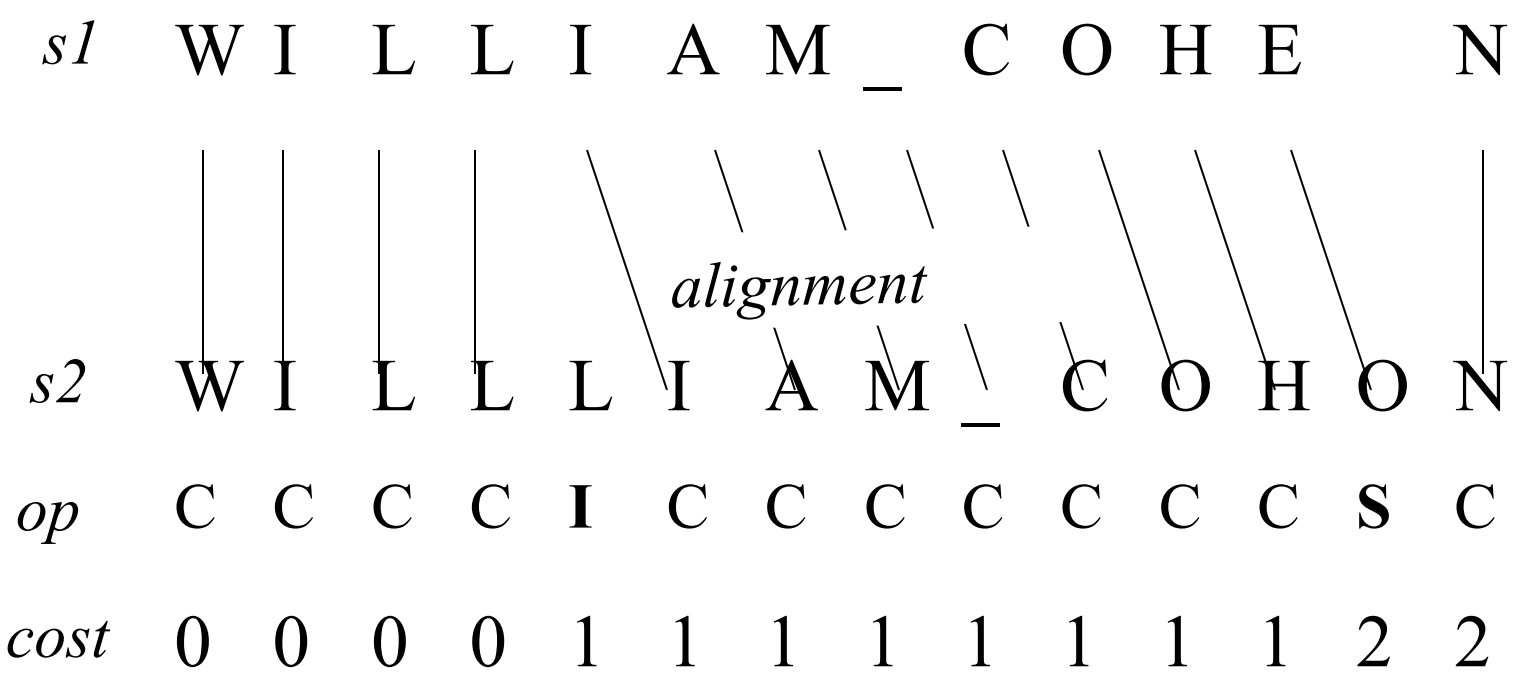
- Sometimes none of the similarity/distance measures are good enough
- Need to use *edit distance*
- Models more accurately the typing mistakes
- But harder to implement in SQL
- Instead: use Jaccard or Hamming for a first pruning, then remove false positives

String distance metrics: Levenstein

- Edit-distance metrics
  - Distance is **shortest sequence of edit commands** that transform *s* to *t*.
  - Simplest set of operations:
    - Copy character from *s* over to *t*
    - Delete a character in *s* (cost 1)
    - Insert a character in *t* (cost 1)
    - Substitute one character for another (cost 1)
  - This is "Levenstein distance"

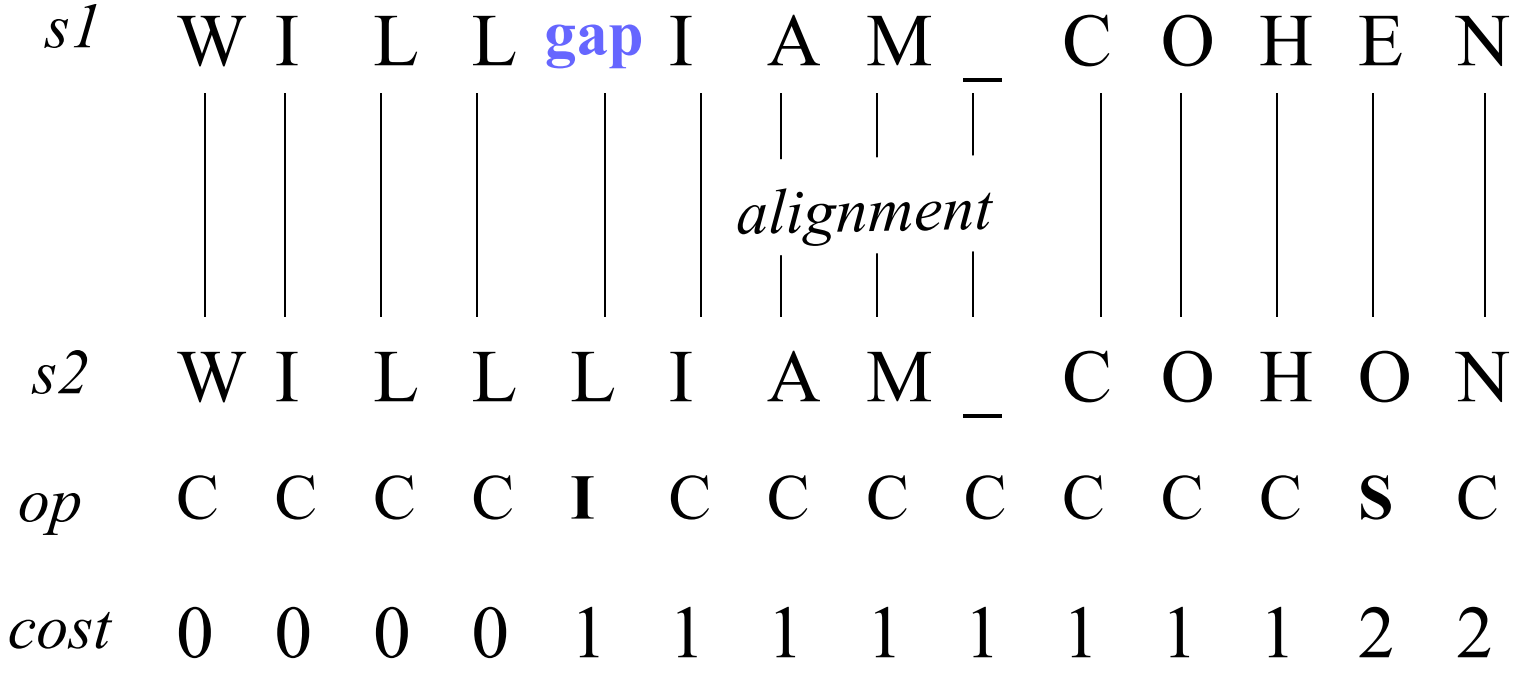
Levenstein distance - example

- distance("William Cohen", "Willliam Cohon")



Levenstein distance - example

- distance("William Cohen", "Willliam Cohon")



Computing Levenstein distance - 1

D(i,j) = score of **best** alignment from $s_1..s_i$ to $t_1..t_j$

$$= \min \begin{cases} D(i-1,j-1), \text{ if } s_i=t_j & \textit{//copy} \\ D(i-1,j-1)+1, \text{ if } s_i!=t_j & \textit{//substitute} \\ D(i-1,j)+1 & \textit{//insert} \\ D(i,j-1)+1 & \textit{//delete} \end{cases}$$

Computing Levenstein distance - 2

Simpler:

$$D(i,j) = \min \begin{cases} D(i-1,j-1) + d(s_i,t_j) & \textit{//subst/copy} \\ D(i-1,j)+1 & \textit{//insert} \\ D(i,j-1)+1 & \textit{//delete} \end{cases}$$

(simplify by letting d(c,d)=0 if c=d, 1 else)

also let D(i,0)=i *(for i inserts)* and D(0,j)=j

Computing Levenstein distance - 3

$$D(i,j) = \min \begin{cases} D(i-1,j-1) + d(s_i,t_j) & \textit{//subst/copy} \\ D(i-1,j)+1 & \textit{//insert} \\ D(i,j-1)+1 & \textit{//delete} \end{cases}$$

	C	**O**	**H**	**E**	**N**
M	1	2	3	4	5
C	1	2	3	4	5
C	2	2	3	4	5
O	3	2	3	4	5
H	4	3	2	3	4
N	5	4	3	3	3

$= D(s,t)$

Computing Levenstein distance – 4

$$D(i,j) = \min \begin{cases} D(i-1,j-1) + d(s_i,t_j) & \textit{//subst/copy} \\ D(i-1,j)+1 & \textit{//insert} \\ D(i,j-1)+1 & \textit{//delete} \end{cases}$$

A *trace* indicates where the min value came from, and can be used to find edit operations and/or a best *alignment* (may be more than 1)

	C	**O**	**H**	**E**	**N**
M	**1**	2	3	4	5
C	**1**	2	3	4	5
C	**2**	3	3	4	5
O	3	**2**	3	4	5
H	4	3	**2**	**3**	4
N	5	4	3	3	**3**

Needleman-Wunch distance

$$D(i,j) = \min \begin{cases} D(i-1,j-1) + d(s_i,t_j) & \textit{//subst/copy} \\ D(i-1,j) + G & \textit{//insert} \\ D(i,j-1) + G & \textit{//delete} \end{cases}$$

G = "gap cost"

d(c,d) is an arbitrary distance function on characters (e.g. related to typo frequencies, amino acid substitutibility, etc)

William Cohen

↓

Wukkuan Cigeb

Pre-filtering

- If editDistance(S_1, S_2) $\leq$ k,
  then $|Qgrams(S_1) \cap Qgrams(S_2)| \geq v$
- Where $v = \max(|S_1|,|S_2|) - 1 - (k-1) * q$
- Where q = length of the q-gram
- Thus, can use any of the signature schemes

The Signature Method

- For each S_i, compute a signature $Sig(S_i)$ s.t.

$J(S_i, S_j) > s$ ⟷ $Sig(S_i) \cap Sig(S_j) \neq$ emptyset

With high probability

The Signature Method

- Step 1: compute all pairs i,j for which $Sig(S_i) \cap Sig(S_j) \neq$ emptyset
  - This is a join operation !

- Step 2: for all such pairs, return (i,j) if $J(S_i, S_j) > s$
  - Hopefully only a few such pairs

Both false positives and false negatives are possible

The Signature Method

Will construct the signature in two steps:

1. Minhashes

2. LSH

Minhashing

- Let π be an arbitrary permutation of the domain

- For each i, let:

$mh(S_i)$ = {the smallest element in S_i according to π}

Example

- The entire domain is {a,b,c,d,e,f,g,h}
- The set S_i is

$$S_i = \{a,b,c,e,f\}$$

- Suppose we choose the permutation:

$$\pi = d,g,c,h,b,f,a,e$$

Then what is $mh(S_i)$ = ?

Minhashing

Main property:

$$\text{Probability}[mh(S_i) = mh(S_j)] = J(S_i, S_j)$$

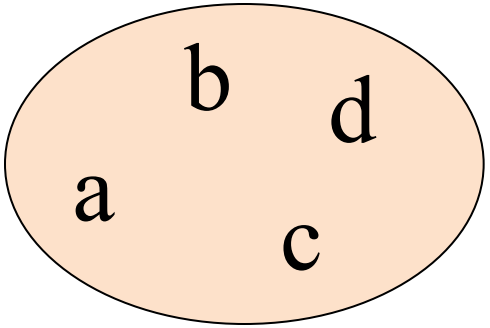
Why ?

Warmup Question

- Choose a *random* permutation π of {a,b,c…,z}
- Consider the set S={a,b,c,d}
- What is the probability that mh(S) = c ?

Warmup Question

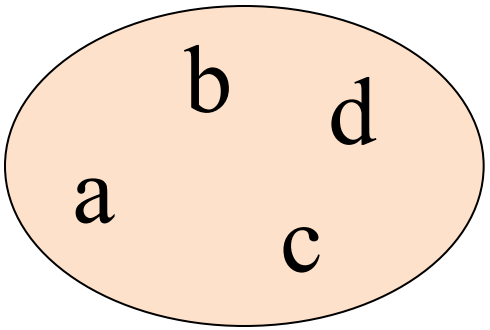
- Choose a *random* permutation π of {a,b,c…,z}

S= 

What is the probability that mh(S) = c ?

Warmup Question

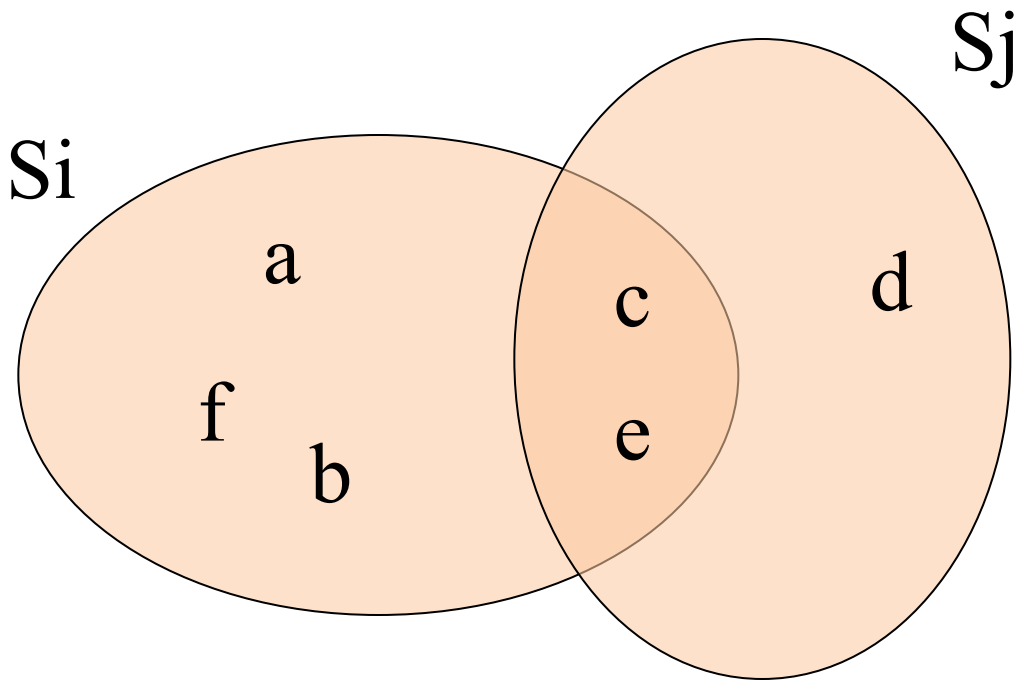
- Choose a *random* permutation π of {a,b,c…,z}

S= 

What is the probability that mh(S) = c ?

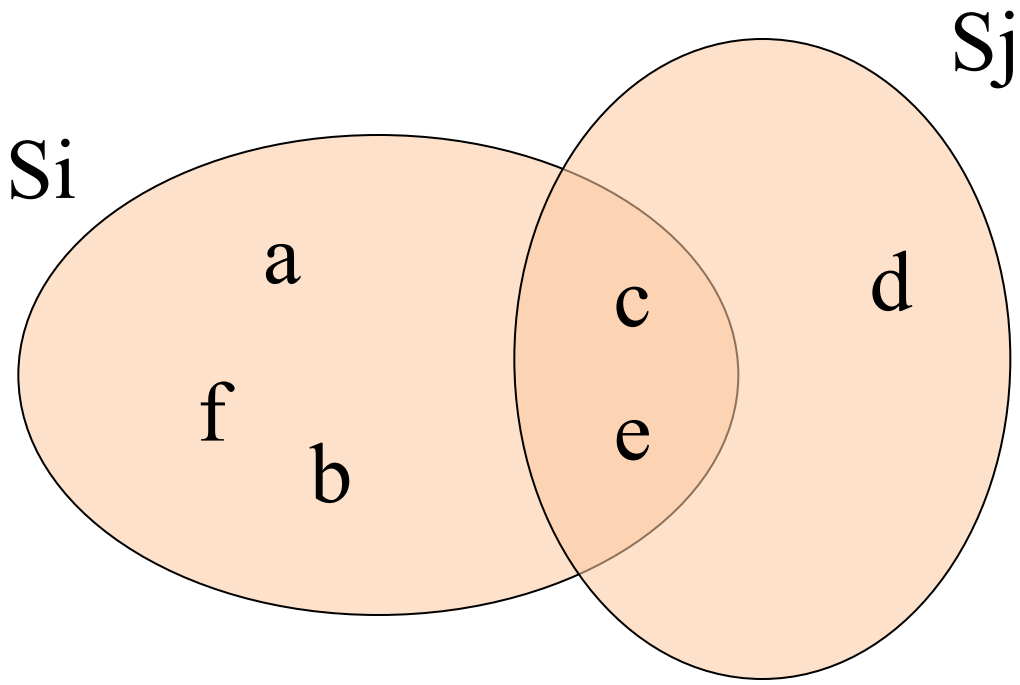
Answer: P = ¼ (each of a,b,c,d can be the min)

Main Property



What is Prob[$mh(S_i) = mh(S_j)$] ?

Main Property



What is Prob[$mh(S_i) = mh(S_j)$] ?

Answer: 2/6 = $J(S_i, S_j)$

Computing Minhashes

- We use a hash function (which we assume is random)

```
mh(S) {
    v = ∞;
    forall x in S do
        if h(x) < v then {v = h(x); y = x;}
     return y;
}
```

Example

- The set S_i is S_i = {a,b,c,e,f}
- Compute h:
  h(a)=77, h(b)=55, h(c)=33, h(e)=88, h(f)=66
- Then what is mh(S_i) = ?

Usage Idea

- Recall: we have n sets S_1, …, S_n

- Compute $mh(S_1)$, …, $mh(S_n)$
- Return pairs for which $mh(S_i)=mh(S_j)$
- Compute their Jaccard similarity
- But too many false negatives !
- How can we improve ?

Improvement

- Independent hash functions $h_1, \ldots, h_m$
- For each S_i, compute $MH(S_i)$ = the m minhashes for each j=1,…,m

Example

- The set S_i is S_i = {a,b,c,e,f}
- Compute h_1:
  $h_1(a)=77$, $h_1(b)=55$, $h_1(c)=33$, $h_1(e)=88$, $h_1(f)=66$
- Compute h_2:
  $h_2(a)=22$, $h_2(b)=66$, $h_2(c)=55$, $h_2(e)=11$, $h_2(f)=44$
- Then what is $MH(S_i)$ = ?
-

Example

- The set S_i is S_i = {a,b,c,e,f}
- Compute h_1:
  $h_1(a)=77$, $h_1(b)=55$, $h_1(c)=33$, $h_1(e)=88$, $h_1(f)=66$
- Compute h_2:
  $h_2(a)=22$, $h_2(b)=66$, $h_2(c)=55$, $h_2(e)=11$, $h_2(f)=44$
- Then what is $MH(S_i)$ = ?
- Answer: $MH(S_i)$ = (c, e) (an ordered pair)

Using Minhashes

- Compute $MH(S_1)$, …, $MH(S_n)$

- $J(S_i, S_j) \approx$ the fraction of positions where $MH(S_i)$ and $MH(S_j)$ agree

Example

n=7

m=6

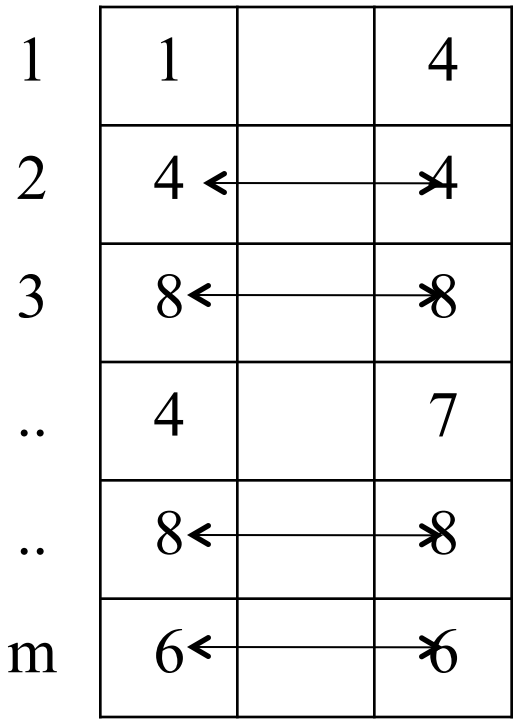
S_1	S_2	S_3	S_4	S_5	S_6	S_7
1	1	4	3	5	4	4
4	6	4	6	9	4	4
8	7	8	9	5	6	8
4	8	7	8	4	3	7
8	7	8	9	5	6	8
6	8	6	8	4	5	6

Estimate $J(S_1,S_2)$ = ?, then $J(S_1,S_3)$=?

Note: Minhashes still require Jaccard

- How do we compute the fraction of positions where MH_i and MH_j agree ?
- Annotate each position with its position number
- Then this is precisely $J(MH_i, MH_j)$

Note:
Minhashes still require Jaccard



MH_1={(1,1),(2,4),(3,8),(4,4),(5,8),(6,6)}
MH_2={(1,4),(2,4),(3,8),(4,7),(5,8),(6,6)}

$J(MH_1, MH_2) = 4/ (2m-4)$

Fraction of equal positions = 4/m

Comments on Minhashes

- It is not a signature yet !
- We have only reduced the problem of computing $J(S_i,S_j)$ to the problem of computing J on smaller sets, of size m

- The signature is provided by LSH (next)

Locality Sensitive Hashing

- $MH_1,\ldots, MH_n$ = sets of m minhashes
- Compute signatures $Sig(MH_i)$ for each set

- Desired property:

$Sig(MH_i) \cap Sig(MH_j) \neq \text{empty} \leftrightarrow J(MH_i, MH_j) > s$

LSH

- Each set Mh_i has m minhashes
- Divide the m minhashes into b "bands" of size r "rows", i.e. m = b*r
- For each band j=1,…,b, apply a hash function h_j to the string of values in band j in MH_i ➔ h_j
- Then $Sig(MH_i) = (h_1, h_2, \ldots, h_b)$

b signatures, each consisting of r minhashes

Example

n=7

m=6
b=3
r=2

S_1	S_2	S_3	S_4	S_5	S_6	S_7
1	1	4	3	5	4	4
4	6	4	6	9	4	4
8	7	8	9	5	6	8
4	8	7	8	4	3	7
8	7	8	9	5	6	8
6	8	6	8	4	5	6

r rows

Each rectangle (=band) becomes one new hash value

Analysis

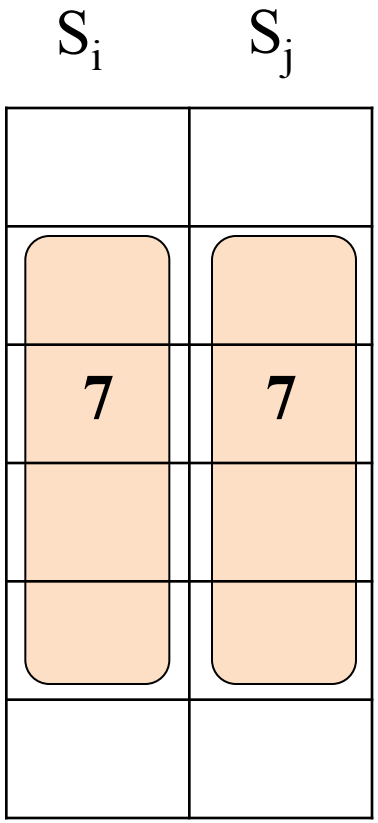
- Goal: want to compute the probability that $Sig(MH(S_i)) \cap Sig(MH(S_j)) \neq$ emptyset, as a function of $s = J(S_i, S_j)$

- So let $s = J(S_i, S_j)$ = fraction of equal positions

Analysis

$J(S_i,S_j) = s$

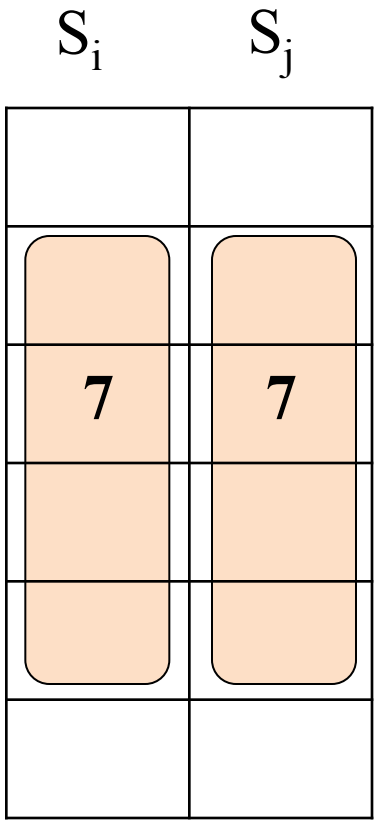
What is the probability that two entries are equal ?



Analysis

$J(S_i, S_j) = s$

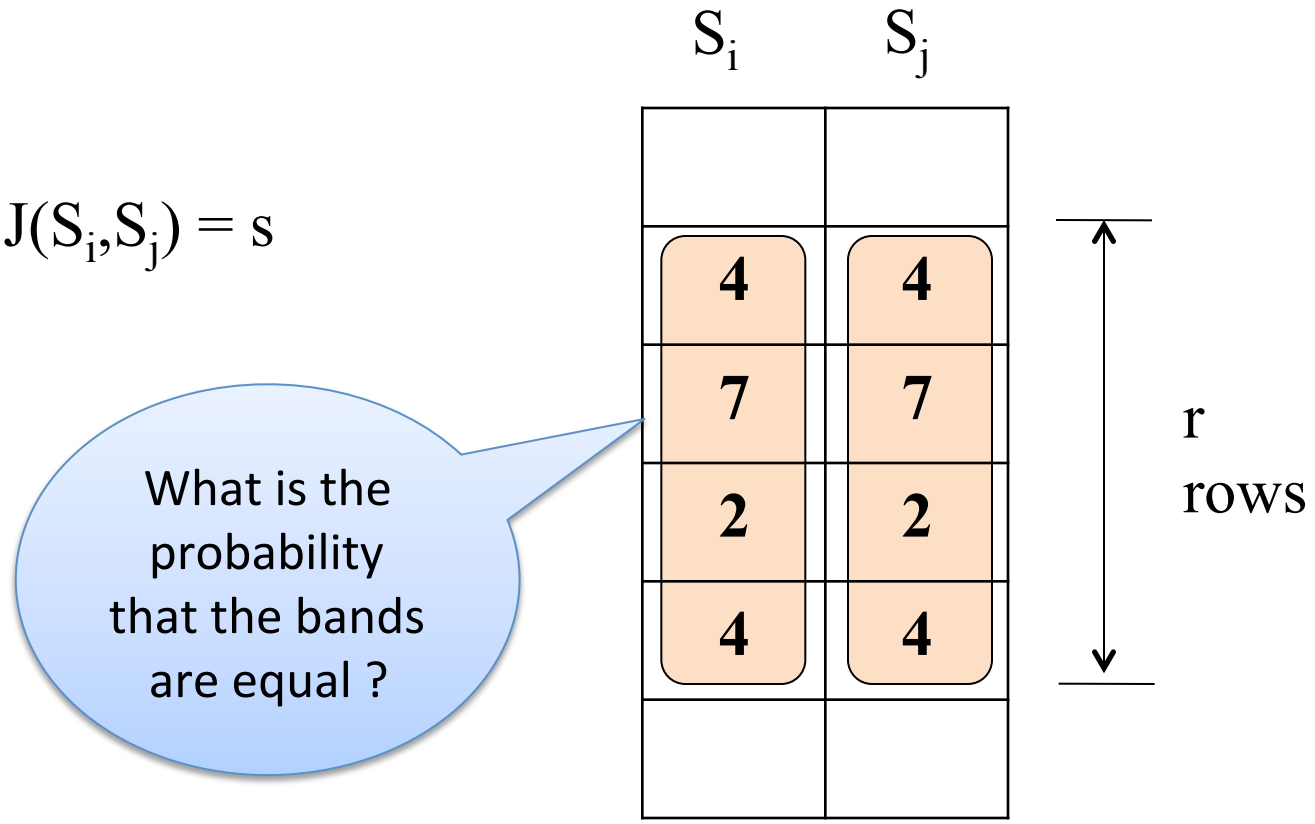
What is the probability that two entries are equal ?



Answer: s

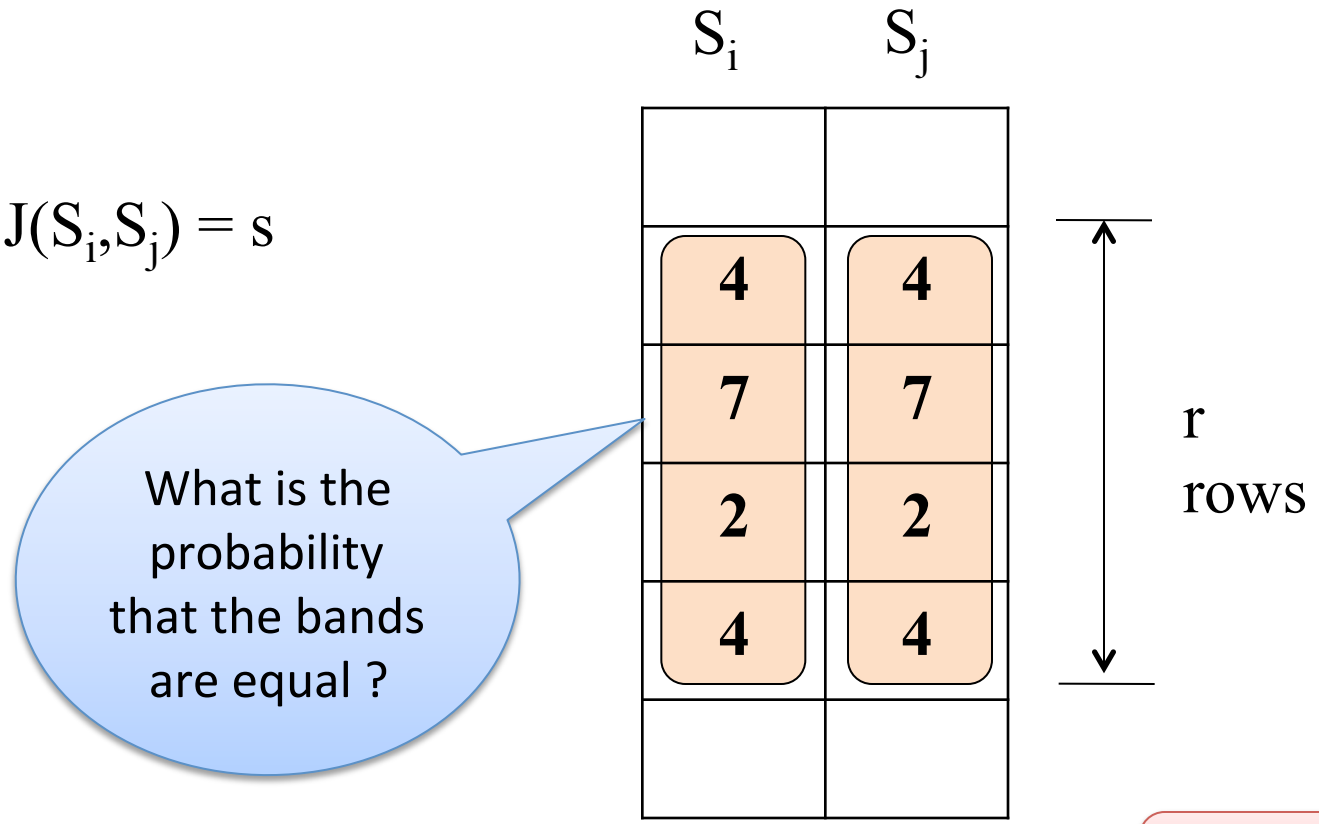
Analysis

$J(S_i,S_j) = s$



Analysis

$J(S_i, S_j) = s$

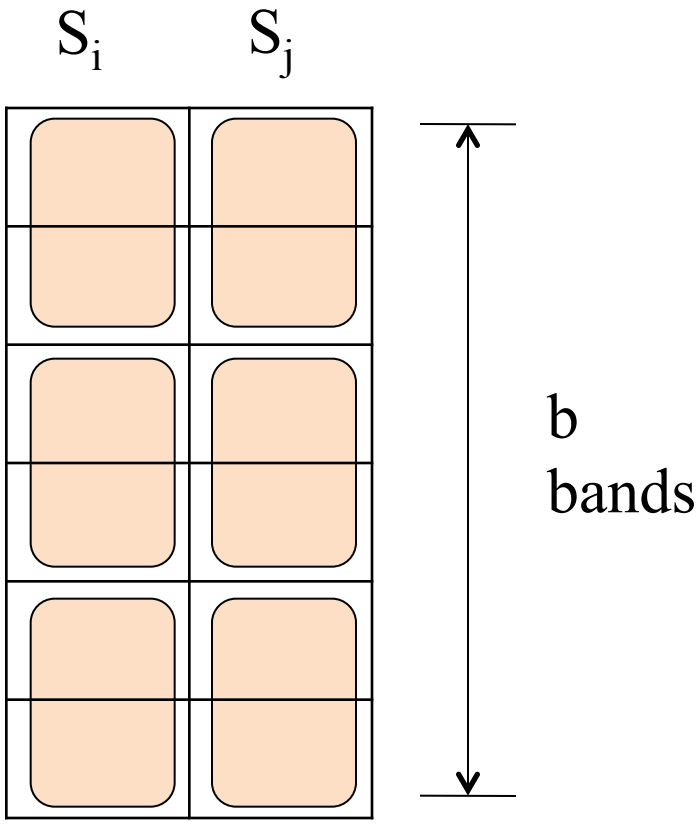


Answer: s^r

Analysis

$J(S_i,S_j) = s$

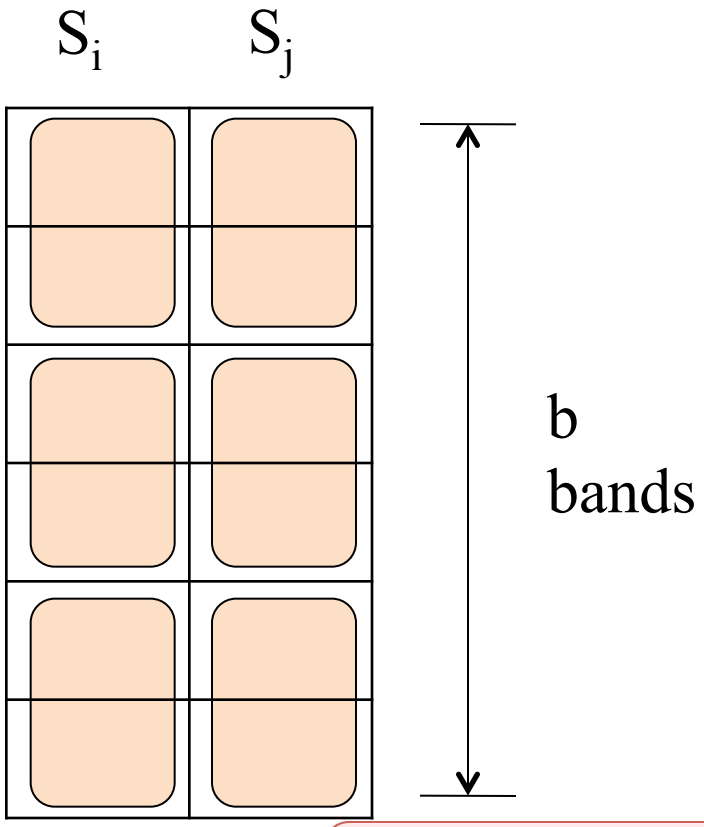
What is the probability that some pair of bands are equal ?



Analysis

$J(S_i,S_j) = s$

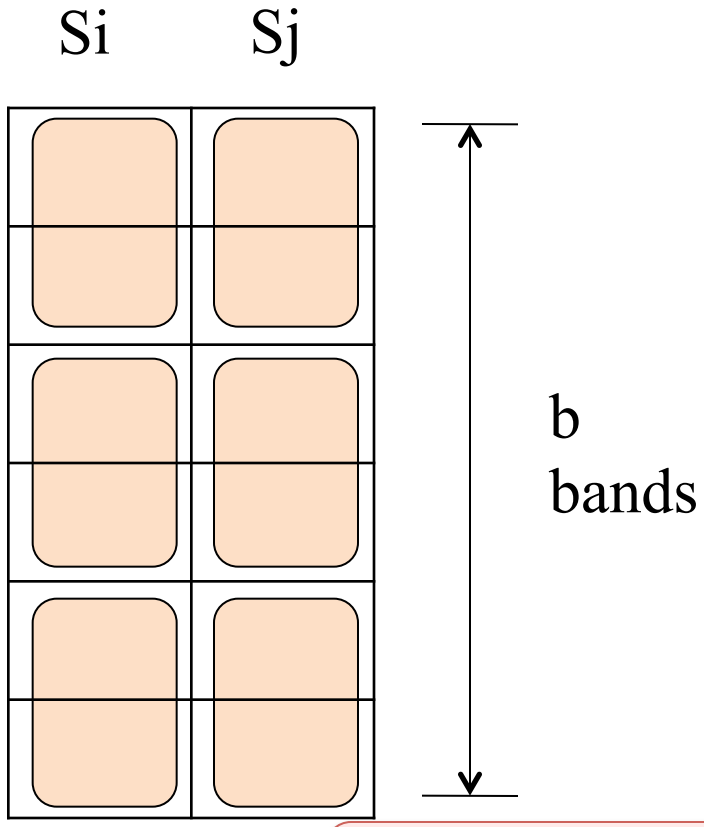
What is the probability that some pair of bands are equal ?



Answer: $1-(1-s^r)^b$

This is precisely
the probability that
Sig(Si) ∩ Sig(Sj)≠ emptyset

$J(S_i,S_j) = s$

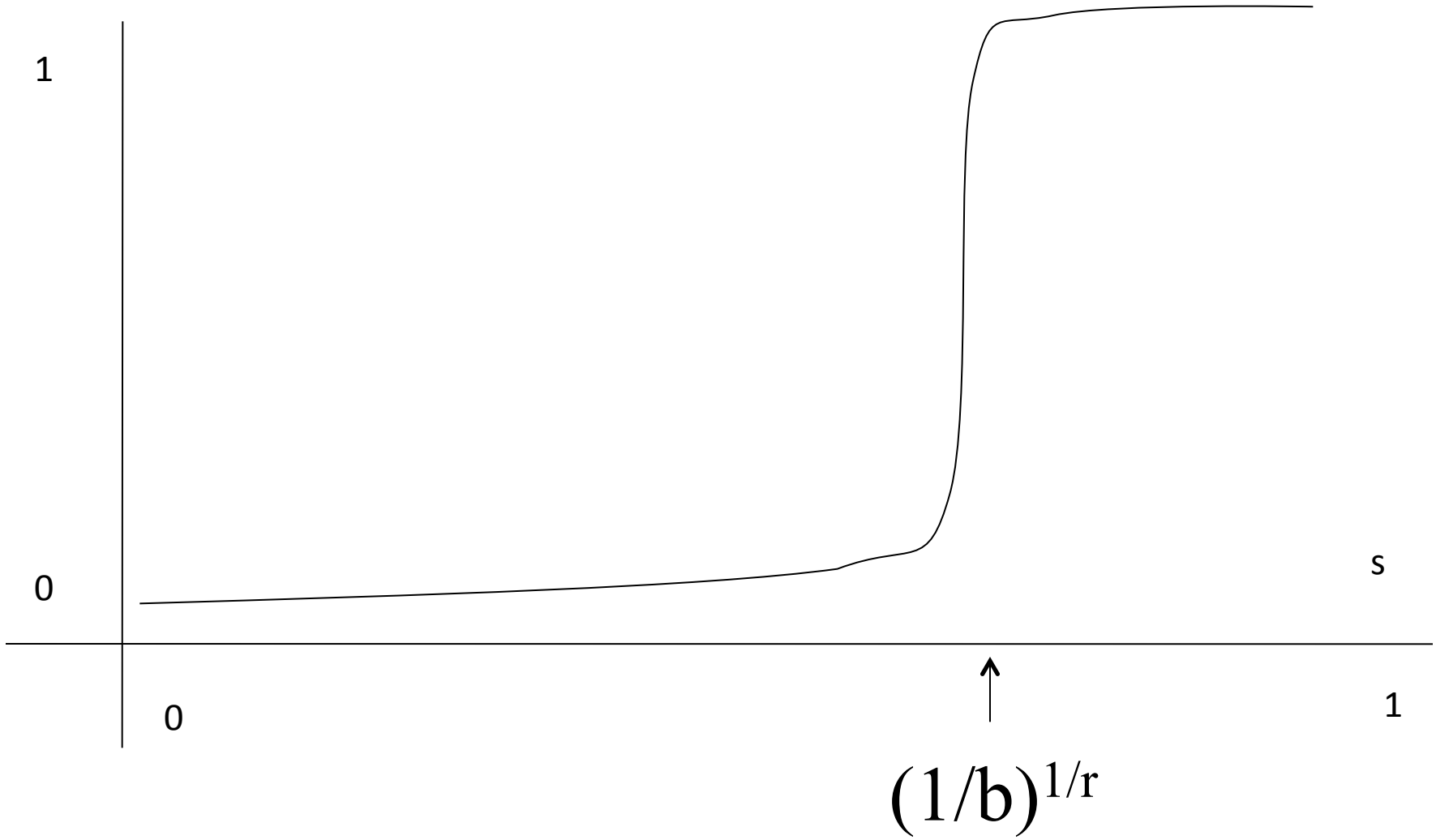


What is the probability that some pair of bands are equal ?

Answer: $1-(1-s^r)^b$

Analysis

Probability $1-(1-s^r)^b$



Putting it together

- You work for a copyright violation detection company
- Customers: have documents 1, 2, 3, …., 10^6
- Web: has pages 1, 2, 3, …., 10^{11}

- Your job is to find "almost identical documents"
- What do you do ???

Step 1: Q-grams

Doc

DocID	Qgram
1	To
1	o b
1	be
1	be
1	e o
1	or
1	or
1	r n
1	no
. . .	. . .
. . .	
2	. .
. . .	

Web

URL	Qgram
abc.com	Oba
abc.com	Bam
abc.com	ama
abc.com	ma
abc.com	a h
. . .	. . .
bcd.com	. . .
. . .	

Step 2: Compute m Min-hashes

docID	mh1	mh2	mh3	mh4	…	mh500
1	To	que	Ion	or		Not
2	. . .	. . .				

url	mh1	mh2	mh3	mh4	…	mh500
abc.com	sec	ret	def	ens		…
bcd.com	. . .	. . .				

Note: m = a few hundreds

Step 3: Compute Signatures

docID	h(mh1,…,mh20)	h(mh21,…,mh40)	. . .
1	2345234	3232	. . .
2	. . .		

docSIG

docID	Sig
1	2345234@1
1	3232@2
1	…
1	452342@25
2	23423@1
2	. . .

webSIG

url	Sig
abc.com	676876@1
abc.com	3232@2
abc.com	…
abc.com	787892@25
bcd.com	23423@1
…	. . .

Step 4: Find docs with common signatures

```
SELECT DISTINCT docSig.docID, webSig.url
FROM docSig, webSig
WHERE docSig.sig = webSig.sig
```

Note: this is a SINGLE JOIN query !!
Still need to filter out the false positives (easy)

Recap: Minhashing and LSH

The Set Similarity Problem

n=7

S1	S2	S3	S4	S5	S6	S7
a	d	d	f	f	a	b
			g		b	
	c		b	g	c	
c			c		f	d
	b		k	c	g	
			a		h	

Problem Find pairs Si, Sj s.t. J(Si, Sj) > s

Example: J(S2, S4)

n=7

S1	S2	S3	S4	S5	S6	S7
a	d	d	f	f	a	b
			g		b	
	c		b	g	c	
c			c		f	d
	b		k	c	g	
			a		h	

J(S2, S4) = 2 / 7

What Minhashes Do

n=7

m=6

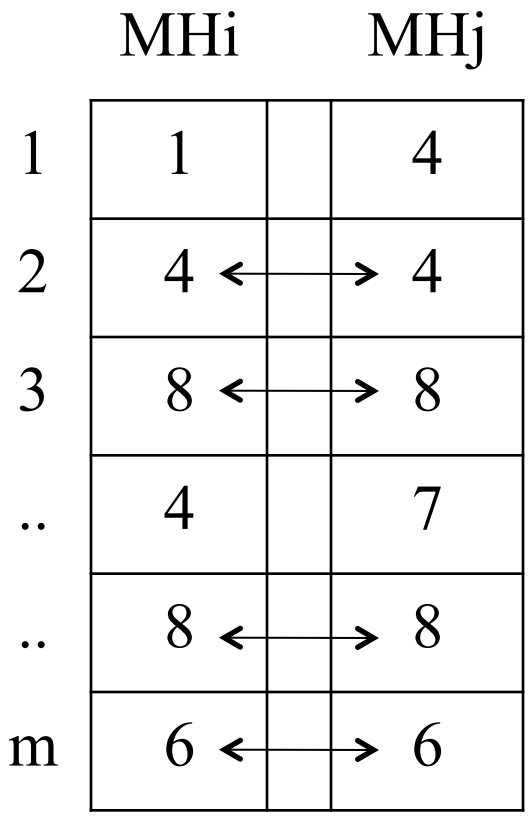
S1	S2	S3	S4	S5	S6	S7
1	1	4	3	5	4	4
4	6	4	6	9	4	4
8	7	8	9	5	6	8
4	8	7	8	4	3	7
8	7	8	9	5	6	8
6	8	6	8	4	5	6

J(S2, S4) = 3/6 = 0.5

What Minhashes Do

- Reduce the problem from comparing sets of variable sizes to comparing vectors of fixed size (m)
- Plus, now we compare only identical positions

What LSH Does



Initial problem:
find pairs of sets s.t. $J(S_i,S_j) > s$

After computing m minhashes it becomes this:

Problem: find pairs of minhashes s.t.
# of equal pairs is > s*m = threshold

What LSH Does

The basic similarity test:

```
Function TestSimilarity(MHi, MHj) {
   c = 0;
   for k = 1, m do {
      if (MHi[k] == MHj[k]) c++;
   }
   if (c>threshold) return TRUE;
   else return FALSE;
}
```

What LSH Does

	MH_i		MH_j
1	1		4
2	4		4
3	8		8
4	4		7
5	8		8
6	6		6

	Lh_i		LH_j
1	h14		h44
2	h84		h87
3	h86		h86

What LSH Does

The new similarity test:

```
Function NewTestSimilarity(LHi, LHj) {
  for k = 1, bdo {
     if (LHi[k] == LHj[k]) return TRUE;
  }
  return FALSE;
}
```

What LSH Does

- The key difference is that now we don't have to ***count*** the number of matches and check if > threshold
- Instead we have to find only ***one*** entry that matches
- Major difference: HAVING ➔ JOIN