Introduction to locality sensitive hashing (LSH)

Andee Kaplan and Rebecca C. Steorts

STA 325, Supplemental Material

October 9, 2018

Agenda

1. Defining similarity
2. Representing data as sets (shingling)
3. Hashing
4. Hashing with compression (minhashing)
5. Too many pairs to compare! (LSH)
6. Evaluation
7. Even faster?

Reading

The reading for this module can be found in Mining Massive Datasets, Chapter 3.

`http://infolab.stanford.edu/~ullman/mmds/book.pdf`

There are no applied examples in the book, however, we will be covering these in class and homework.

Motivations

In information retrieval, recall that one main goal is understanding how similar items are in our task (application) at hand.

Documents

Suppose that we have a large number of documents (articles, songs, etc). We may wish to know which documents are the most similar.

Entity resolution

Entity resolution (record linkage or de-duplication) is the process of removing duplicate information from large noisy databases.

Often times we perform entity resolution and then perform a regression task (or some other inference/prediction task).

Information retrieval for comparing documents, records, etc

1. If we're looking to remove duplicate entities from a database, this requires comparing all n records in the database. (This is an quadratic operation.)
2. Typically, before performing entity resolution, we will want to perform dimension reduction to reduce down the space of records to something more manageable.

Overall questions

- How can we use take similar data points (records) and put them in clusters (buckets or bins)?
- How can we evalauate how well our method is doing? (Look at the precision and recall).

Blocking (partitioning)

Blocking partitions of data points so that we do not have to make all-to-all data point comparisons.

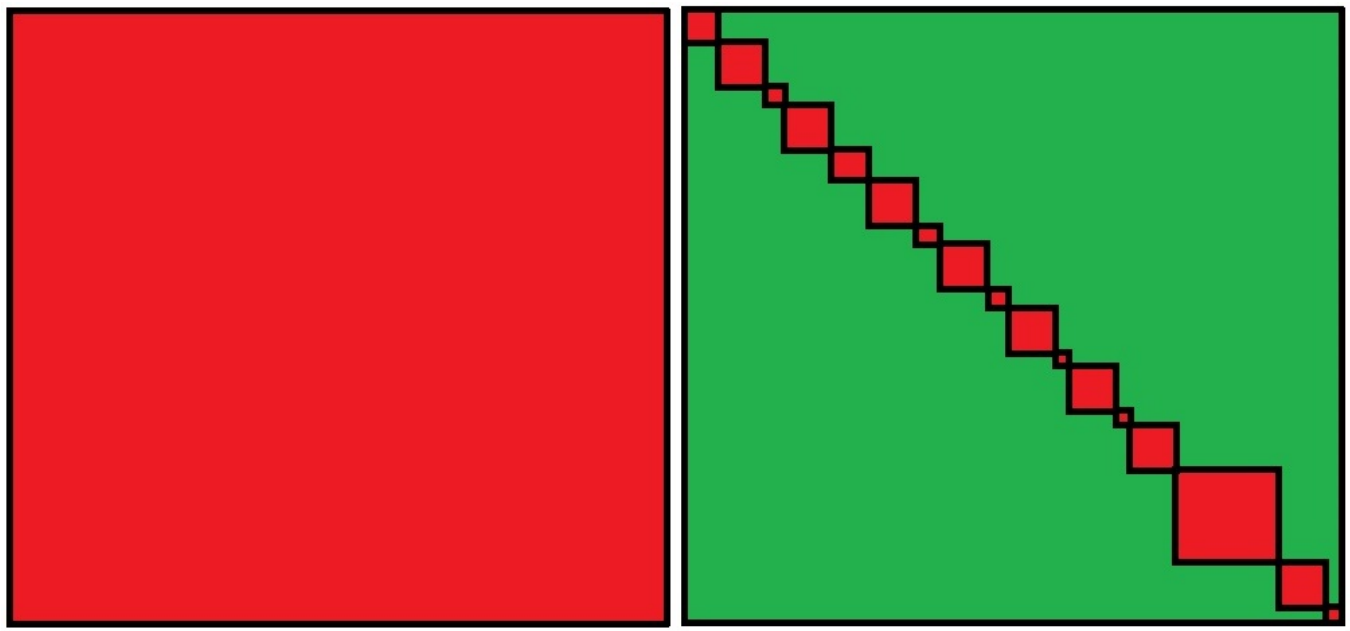


Figure: All-to-all data point comparisons (left) versus partitioning data points into blocks/bins (right).

Entity resolution

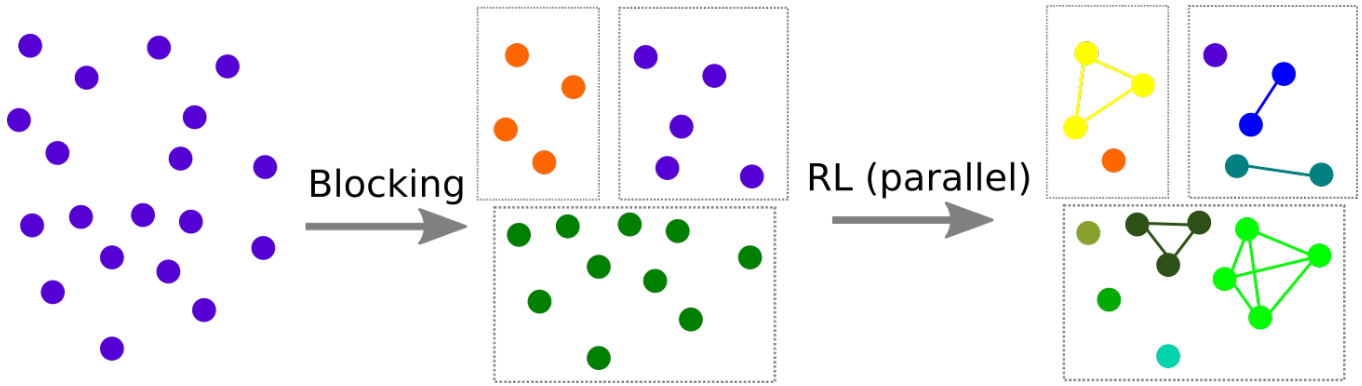


Figure: Entire process of the blocking step (via LSH) and then the record linkage step. Step 1: Dimension reduction via blocking. Step 2: Removing duplicate entities via record linkage.

Remark: In this module, we will focus on how to perform dimension reduction methods using LSH. For reading on how to perform the record linkage step, see Christen (2012).

Goal

Our overall goal in this lecture will be able to quickly compare the similarity of the following:

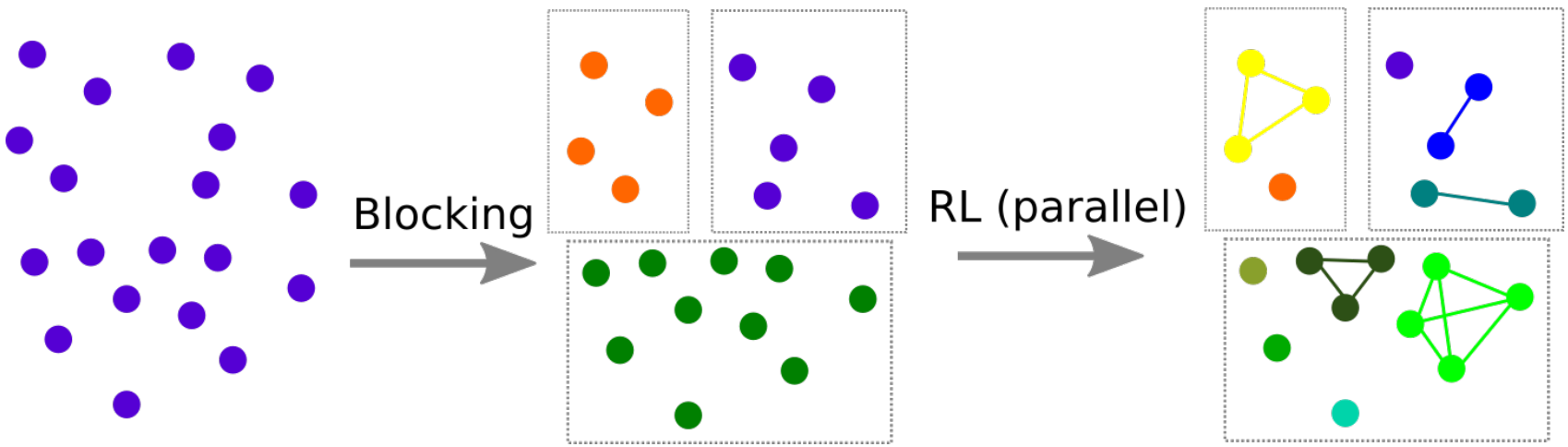
- documents
- songs
- data points
- other examples

We will work with two data sets. The first is a synthetic dataset from the `RecordLinkage` package in `R` called `RLdata500`. The second is a real data set on github (`RLdata`) called `cora`[1]

[1] This can be found at `https://github.com/resteorts/RLdata`.

Goal

Goal: Introduce locality sensitive hashing, a fast method of blocking for record linkage, and get some experience doing LSH in `R`



Figure

Finding similar items

- We want to find similar items

  - Maybe we are looking for near duplicate documents (plagiarism)

  - More likely, we are trying to block our data which we can later pass to a record linkage process

- How do we define *similar*?

Jaccard similarity

There are many ways to define similarity, we will use *Jaccard similarity* for this task

$$Jac(S, T) = \frac{| S \cap T |}{| S \cup T |}$$

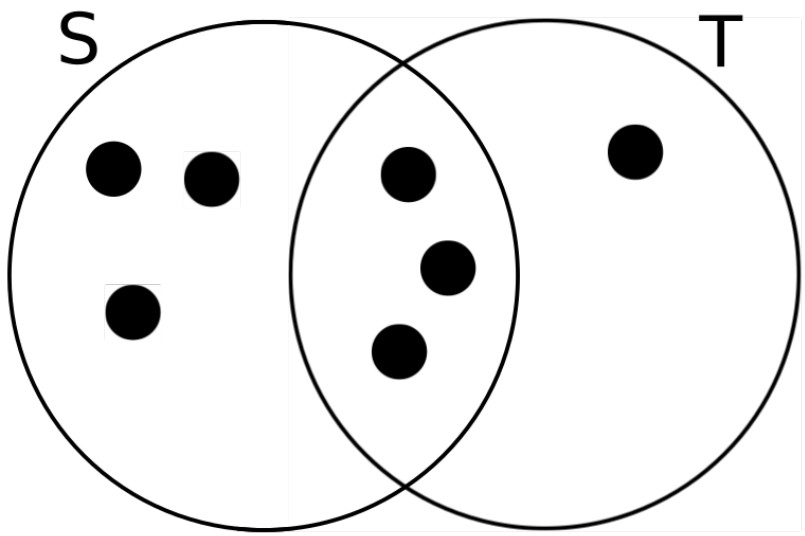


Figure: Two sets S and T with Jaccard similarity 3/7. The two sets share 3 elements in common, and there are 7 elements in total.

How to represent data as sets

We want to talk about similarity of data $\Rightarrow$ we need sets to compare!

- One way is to construct from the data the set of **short strings** that appear within it
- Similar documents/datasets will have many common elements, i.e. many commong short strings
- We can do construct these short strings using *shingling*

k-shingling (how-to)

1. Think of a document or record as a string of characters
2. A k-shingle (k-gram) is any sub-string (word) of length k found within the document or record
3. Associate with each document or record the set of k-shingles that appear one or more times within it

Let's try

Suppose our document is the string "Hello world", then

- the set of 2-shingles is

$$\{\text{he, el, ll, lo, ow, wo, or, rl, ld}\}$$

- the set of 3-shingles is

$$\{\text{hel, ell, llo, low, owo, wor, orl, rld}\}$$

Your turn

We have the following two records:

	First name	Last name
129	MICHAEL	VOGEL
130	MICHAEL	MEYER

1. Compute the 2-shingles for each record
2. Using Jaccard similarity, how similar are they?

Your turn solution

1. The 2-shingles for the first record are

$$\{\text{mi, ic, ch, ha, ae, el, lv, vo, og, ge, el}\}$$

and for the second are

$$\{\text{mi, ic, ch, ha, ae, el, lm, me, ey, ye, er}\}$$

2. There are 6 items in common

$$\{\text{mi, ic, ch, ha, ae, el}\}$$

and 15 items total

$$\{\text{mi, ic, ch, ha, ae, el, lv, vo, og, ge, lm, me, ey, ye, er}\},$$

so the Jaccard similarity is

$$\frac{6}{15} = \frac{2}{5} = 0.4$$

Useful packages/functions in R

It would be better to automate this. (And we can do this in R.)

```
library(textreuse) # text reuse/document similarity
library(tokenizers) # shingles
```

We can use the following functions to create k-shingles and calculate Jaccard similarity for our data

```
# get k-shingles
tokenize_character_shingles(x, n)

# calculate jaccard similarity for two sets
jaccard_similarity(a, b)
```

Cora data

Research paper headers and citations, with information on authors, title, institutions, venue, date, page numbers and several other fields

```
library(RLdata) # data library
data(cora) # load the cora data set
str(cora) # structure of cora
```

```
## 'data.frame':    1879 obs. of  16 variables:
##  $ id          : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ title       :Class 'noquote'  chr [1:1879] "Inganas and M.R" NA NA NA ...
##  $ book_title :Class 'noquote'  chr [1:1879] NA NA NA NA ...
##  $ authors     :Class 'noquote'  chr [1:1879] "M. Ahlskog, J. Paloheimo, H. Stubb, P. Dyreklev, M. Fahlr
##  $ address     :Class 'noquote'  chr [1:1879] NA NA NA NA ...
##  $ date        :Class 'noquote'  chr [1:1879] "1994" "1994" "1994" "1994" ...
##  $ year        :Class 'noquote'  chr [1:1879] NA NA NA NA ...
##  $ editor      :Class 'noquote'  chr [1:1879] NA NA NA NA ...
##  $ journal     :Class 'noquote'  chr [1:1879] "Andersson, J Appl. Phys." "JAppl. Phys." "J Appl. Phys."
##  $ volume      :Class 'noquote'  chr [1:1879] "76" "76" "76" "76" ...
##  $ pages       :Class 'noquote'  chr [1:1879] "893" "893" "893" "893" ...
##  $ publisher   :Class 'noquote'  chr [1:1879] NA NA NA NA ...
##  $ institution:Class 'noquote'  chr [1:1879] NA NA NA NA ...
##  $ type        :Class 'noquote'  chr [1:1879] NA NA NA NA ...
##  $ tech        :Class 'noquote'  chr [1:1879] NA NA NA NA ...
##  $ note        :Class 'noquote'  chr [1:1879] NA NA NA NA ...
```

Your turn

Using the `title`, `authors`, and `journal` fields in the `cora` dataset,

1. Get the 3-shingles for each record (**hint:** use `tokenize_character_shingles`)

2. Obtain the Jaccard similarity between each pair of records (**hint:** use `jaccard_similarity`)

Your turn (solution)

```
# get only the columns we want
n <- nrow(cora) # number of records
dat <- data.frame(id = seq_len(n)) # create id column
dat <- cbind(dat, cora[, c("title", "authors", "journal")]) # get columnds we want

# 1. paste the columns together and tokenize for each record
shingles <- apply(dat, 1, function(x) {
  # tokenize strings
  tokenize_character_shingles(paste(x[-1], collapse=" "), n = 3)[[1]]
})

# 2. Jaccard similarity between pairs
jaccard <- expand.grid(record1 = seq_len(n), # empty holder for similarities
                       record2 = seq_len(n))

# don't need to compare the same things twice
jaccard <- jaccard[jaccard$record1 < jaccard$record2,]

time <- Sys.time() # for timing comparison
jaccard$similarity <- apply(jaccard, 1, function(pair) {
  jaccard_similarity(shingles[[pair[1]]], shingles[[pair[2]]]) # get jaccard for each pair
})
time <- difftime(Sys.time(), time, units = "secs") # timing
```

This took took 164.95 seconds $\approx$ 2.75 minutes

Your turn (solution, cont'd)

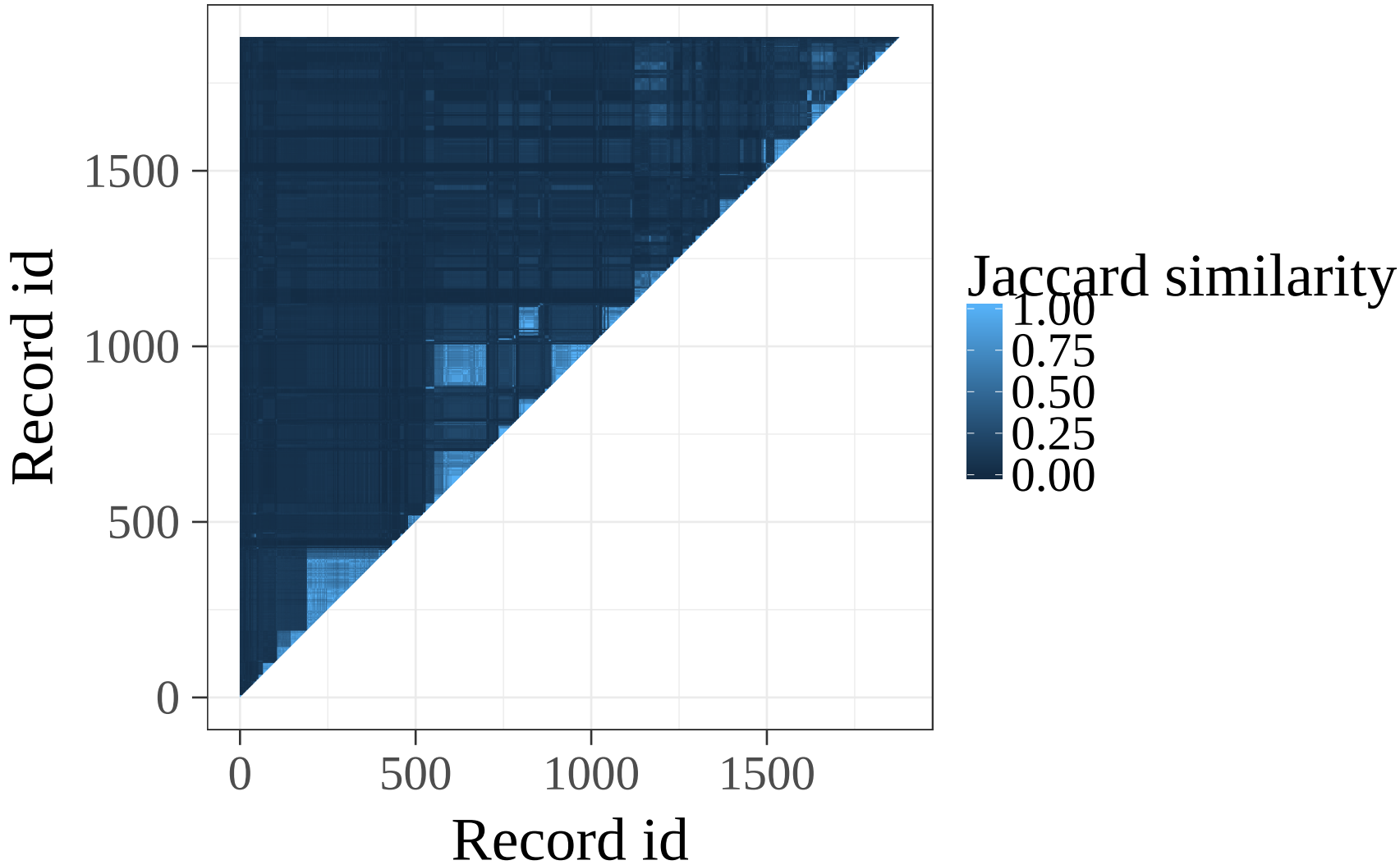


Figure: Jaccard similarity for each pair of records. Light blue indicates the two records are more similar and dark blue indicates less similar.

Hashing

For a dataset of size n, the number of comparisons we must compute is

$$\frac{n(n-1)}{2}$$

- For our set of records, we needed to compute $1,764,381$ comparisons

- A better approach for datasets of any realistic size is to use *hashing*

Hash functions

- Traditionally, a *hash function* maps objects to integers such that similar objects are far apart
- Instead, we want special hash functions that do the **opposite** of this, i.e. similar objects are placed closed together!

Definition: Hash function

Hash functions $h()$ are defined such that

> *If records A and B have high similarity, then the probability that $h(A) = h(B)$ is **high** and if records A and B have low similarity, then the probability that $h(A) \neq h(B)$ is **high**.*

Hashing shingles

Instead of storing the strings (shingles), we can just store the *hashed values*

These are integers, they will take less space

```
# instead store hash values (less memory)
hashed_shingles <- apply(dat, 1, function(x) {
  string <- paste(x[-1], collapse=" ") # get the string
  shingles <- tokenize_character_shingles(string, n = 3)[[1]] # 3-shing
  hash_string(shingles) # return hashed shingles
})
```

This took up 6.38256×10^5 bytes, while storing the shingles took 7.36544×10^6 bytes; the whole pairwise comparison still took the same amount of time ($\approx$ 2.35 minutes)

Similarity preserving summaries of sets

- Sets of shingles are large (larger than the original document)
- If we have millions of documents, it may not be possible to store all the shingle-sets in memory
- We can replace large sets by smaller representations, called *signatures*
- And use these signatures to **approximate** Jaccard similarity

Characteristic matrix

In order to get a signature of our data set, we first build a *characteristic matrix*

Columns correspond to records and the rows correspond to all hashed shingles

	Record 1	Record 2	Record 3	Record 4	Record 5
-78464425	1	1	1	1	1
-78234440	1	0	0	0	0
-78221717	1	0	0	0	0
-78235289	1	1	1	1	1
-78555255	1	1	1	1	1
-78132973	1	1	1	1	1

The result is a 3551×1879 matrix

Question: Why would we not store the data as a characteristic matrix?

Minhashing

Want create the signature matrix through minhashing

1. Permute the rows of the characteristic matrix m times
2. Iterate over each column of the permuted matrix
3. Populate the signature matrix, row-wise, with the row index from the first 1 value found in the column

The signature matrix is a hashing of values from the permuted characteristic matrix and has one row for the number of permutations calculated (m), and a column for each record

Minhashing (cont'd)

Record 1	Record 2	Record 3	Record 4	Record 5
30	30	30	30	30
8	8	8	8	8
6	1	1	1	1
2	2	2	2	2
102	102	102	102	102
16	16	16	16	16
8	8	8	8	8
112	161	161	161	161
1	1	1	1	1
76	27	27	27	27

Signature matrix and Jaccard similarity

The relationship between the random permutations of the characteristic matrix and the Jaccard Similarity is

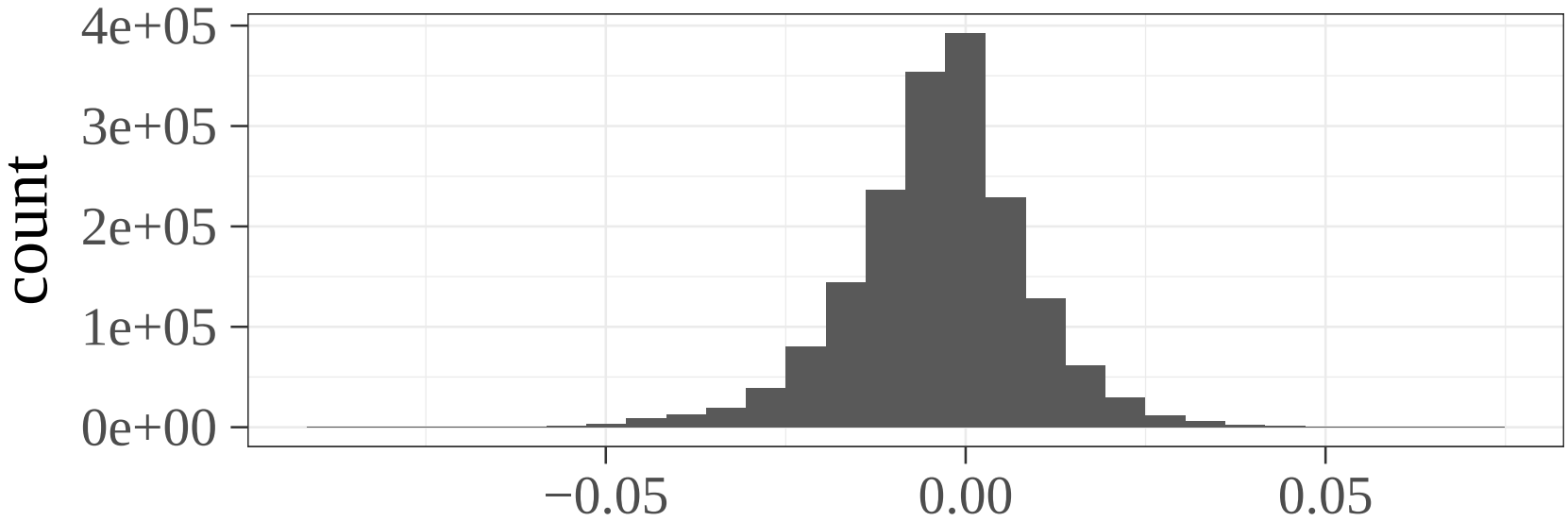
$$Pr\{\min[h(A)] = \min[h(B)]\} = \frac{|A \cap B|}{|A \cup B|}$$

We use this relationship to **approximate** the similarity between any two records

We look down each column of the signature matrix, and compare it to any other column

The number of agreements over the total number of combinations is an approximation to Jaccard measure

Jaccard similarity approximation



Difference between Jaccard similarity and minhash approx

Used minhashing to get an approximation to the Jaccard similarity, which helps by allowing us to store less data (hashing) and avoid storing sparse data (signature matrix)

We still haven't addressed the issue of **pairwise comparisons**

Avoiding pairwise comparisons

- Performing pairwise comparisons is time-consuming because the number of comparisons grows at $O(n^2)$
- Most of those comparisons are **unnecessary** because they do not result in matches due to sparsity
- We will use the combination of minhash and locality-sensitive hashing (LSH) to compute possible matches only once for each document, so that the cost of computation grows **linearly**

Locality Sensitive Hashing (LSH)

Idea: We want to hash items several times such that similar items are more likely to be hashed into the same bucket

1. Divide signature matrix into b bands with r rows each so $m = b * r$ where m is the number of times that we drew a permutation of the characteristic matrix in the process of minhashing
2. Each band is hashed to a bucket by comparing the minhash for those permutations
   - If they match within the band, then they will be hashed to the same bucket
3. If two documents are hashed to the same bucket they will be considered candidate pairs

We only check *candidate pairs* for similarity

Banding and buckets

	Record 1	Record 2	Record 3	Record 4	Record 5
1	30	30	30	30	30
2	8	8	8	8	8
3	6	1	1	1	1
4	2	2	2	2	2
5	102	102	102	102	102
6	16	16	16	16	16
7	8	8	8	8	8
8	112	161	161	161	161
9	1	1	1	1	1
10	76	27	27	27	27

Tuning

How to choose k

How large k should be depends on how long our data strings are

The important thing is k should be picked large enough such that the probability of any given shingle is *low*

How to choose b

b must divide m evenly such that there are the same number of rows r in each band

What else?

Choosing b

$P(\text{two documents w/ Jaccard similarity } s \text{ marked as potential match}) = 1-(1-s^{m/b})^b$

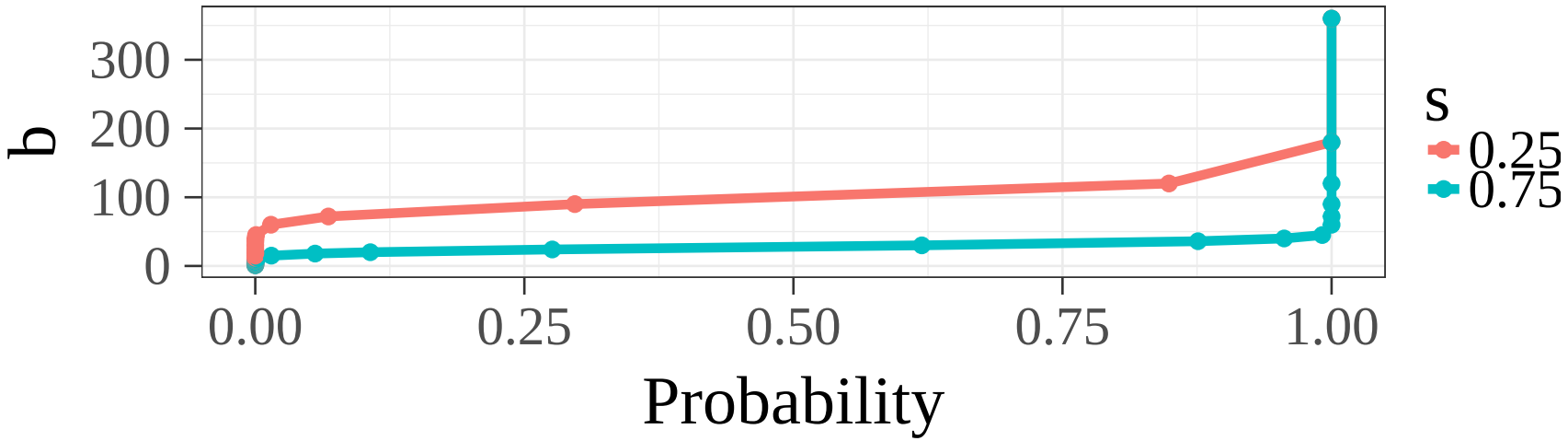


Figure: Probability that a pair of documents with a Jaccard similarity s will be marked as potential matches for various bin sizes b for $s = .25, .75$ for the number of permutations we did, $m = 360$.

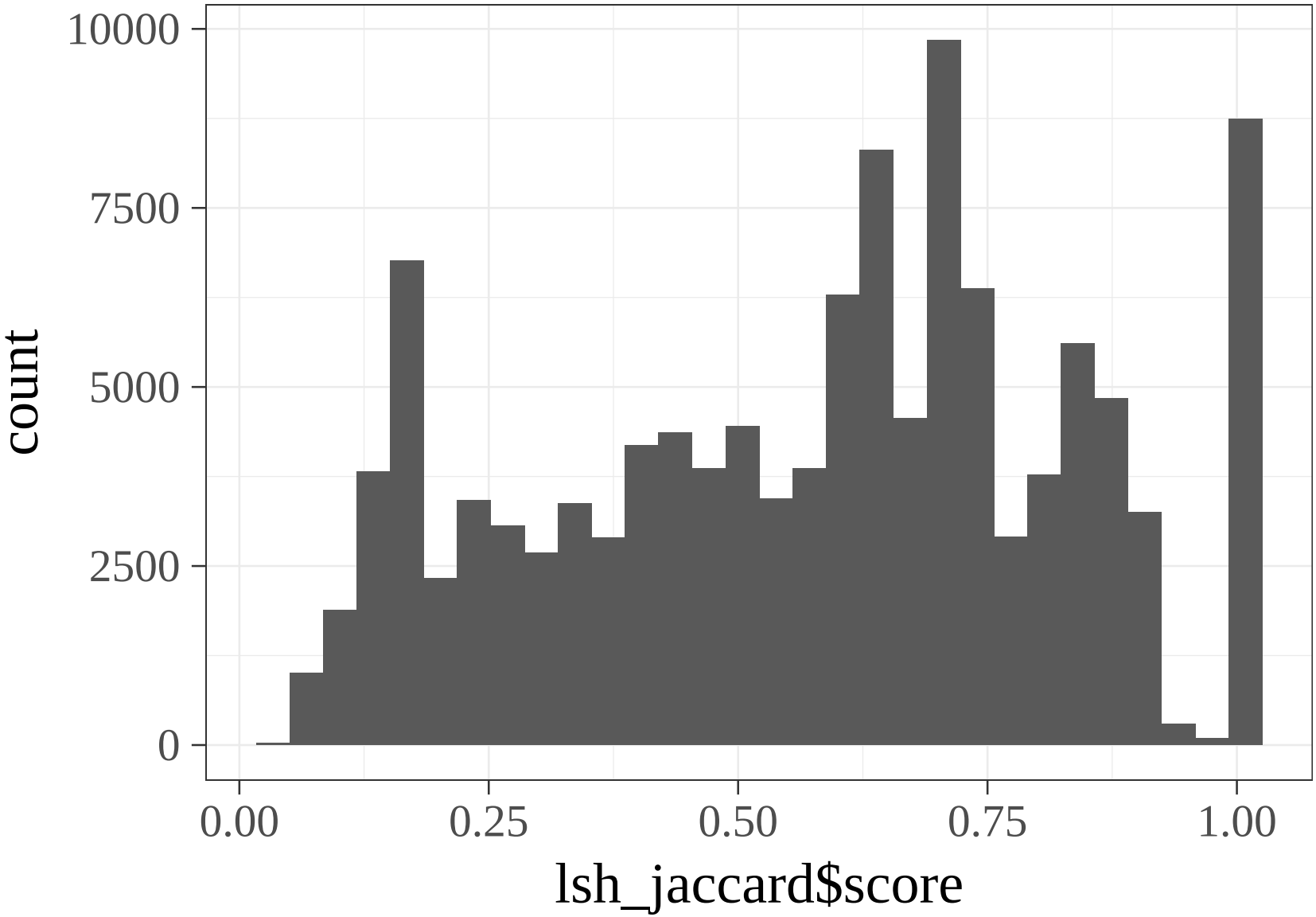
For $b = 90$, a pair of records with Jaccard similarity .25 will have a 29.7% chance of being matched as candidates and a pair of records with Jaccard similarity .75 will have a 100.0% chance of being matched as candidates

"Easy" LSH in R

There an easy way to do LSH using the built in functions in the `textreuse` package via the functions `minhash_generator` and `lsh` (so we don't have to perform it by hand):

```
# choose appropriate num of bands
b <- 90

# create the minhash function
minhash <- minhash_generator(n = m, seed = 02082018)

# build the corpus using textreuse
docs <- apply(dat, 1, function(x) paste(x[-1], collapse = " ")) # get strings
names(docs) <- dat$id # add id as names in vector
corpus <- TextReuseCorpus(text = docs, # dataset
                          tokenizer = tokenize_character_shingles, n = 3, simplify = TRUE, # shingles
                          progress = FALSE, # quietly
                          keep_tokens = TRUE, # store shingles
                          minhash_func = minhash) # use minhash

# perform lsh to get buckets
buckets <- lsh(corpus, bands = b, progress = FALSE)

# grab candidate pairs
candidates <- lsh_candidates(buckets)

# get Jaccard similarities only for candidates
lsh_jaccard <- lsh_compare(candidates, corpus, jaccard_similarity, progress = FALSE)
```

“Easy” LSH in R (cont’d)



Putting it all together

The last thing we need is to go from candidate pairs to blocks

```
library(igraph) #graph package

# think of each record as a node
# there is an edge between nodes if they are candidates
g <- make_empty_graph(n, directed = FALSE) # empty graph
g <- add_edges(g, as.vector(t(candidates[, 1:2]))) # candidate edges
g <- set_vertex_attr(g, "id", value = dat$id) # add id

# get custers, these are the blocks
clust <- components(g, "weak") # get clusters
blocks <- data.frame(id = V(g)$id, # record id
                     block = clust$membership) # block number
head(blocks)
```

```
##   id block
## 1  1     1
## 2  2     1
## 3  3     1
## 4  4     1
## 5  5     1
```

Your turn

Using the `fname_c1` and `lname_c1` columns in the `RecordLinkage::RL500` dataset,

1. Use LSH to get candidate pairs for the dataset

   - What k to use for shingling?
   - What b to use for bucket size?

2. Append the blocks to the original dataset as a new column, `block`

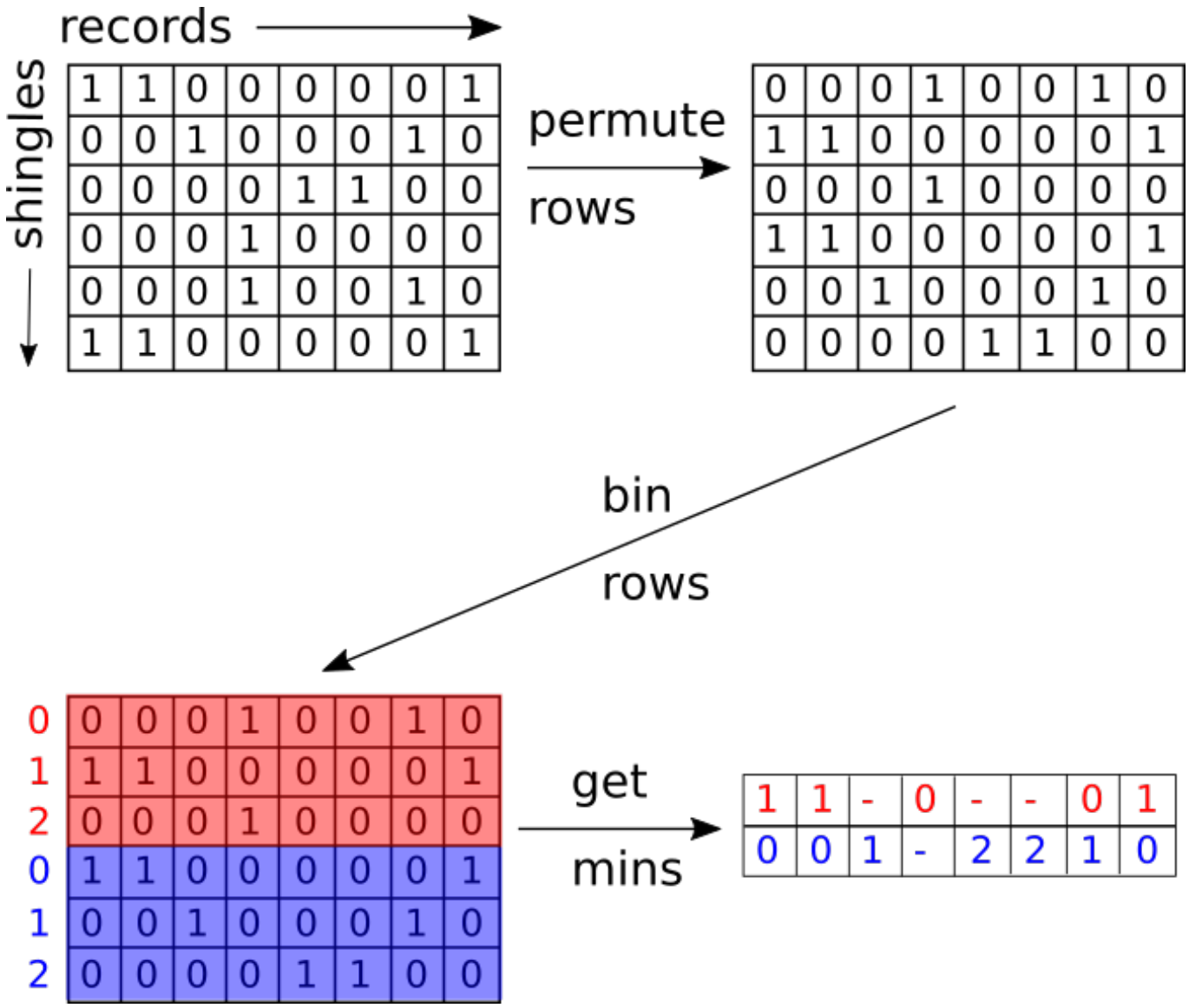
Even faster?

(**fast**): In minhashing we have to perform m permutations to create multiple hashes

(**faster**): We would like to reduce the number of hashes we need to create – "Densified" One Permutation Hashing (DOPH)

- One permutation of the signature matrix is used
- The feature space is then binned into m evenly spaced bins
- The m minimums (for each bin separately) are the m different hash values

“Densified” One Permutation Hashing (DOPH)



Figure