

## Module 5: Probabilistic Blocking, Part II

Rebecca C. Steorts

# Agenda

- ▶ Data Cleaning Pipeline
- ▶ Blocking
- ▶ Locality Sensitive Hashing (LSH)
- ▶ Hash functions
- ▶ Hashed shingles
- ▶ Signatures
- ▶ Characteristic Matrix
- ▶ Minhash (Jaccard Similarity Approximation)
- ▶ Back to LSH

# Load R packages

```
library(RecordLinkage)
library(blink)
library(knitr)
library(textreuse) # text reuse/document similarity
library(tokenizers) # shingles
library(devtools)
library(cora)
library(ggplot2)
# install_github("resteorts/cora")
data(cora) # load the cora data set
```

# Data Cleaning Pipeline



Figure 1: Data cleaning pipeline.

## Blocking

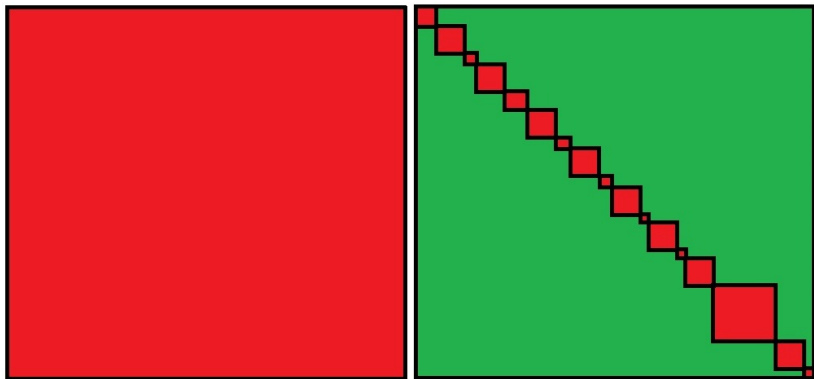


Figure 2: Left: All to all record comparison. Right: Example of resulting blocking partitions.

# LSH

Locality sensitive hashing (LSH) is a fast method of blocking for record linkage that originates from the computer science literature.

# Hash function overview

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

Technical reading on this: Chen et al. (2018) and Shrivastava and Steorts (2018)

# Hash function

A *hash function*  $h()$  is defined such that

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



# Hashing shingles

1. Instead of storing the strings as shingles, we store the **hashed values**.
2. These are integers (instead of strings).

We do this because the integers take up less memory, so we are performing a type of **dimension reduction**.

## Hashing shingles (continued)

## Hashing shingles (continued)

## Hashing shingles (continued)

- ▶ To hash the shingles, it took  $6.53296 \times 10^5$  bytes.
- ▶ To store the shingles, it took  $8.411816 \times 10^6$  bytes.

**The entire pairwise comparison still took the same amount of time ( $\approx 1.14$  minutes) for both approaches, so keep in mind we have not improved this aspect of our approach.**

## Similarity preserving summaries of sets

Sets of shingles are large (larger than the original data set)

If we have millions of records in our data set, it may not be possible to store all the shingle-sets in memory

We can replace large sets by smaller representations, called *signatures*

We can use the *signatures* to **approximate** Jaccard similarity (using a cool fact)

# Characteristic matrix

How do we build signatures?

1. Build a characteristic matrix
2. Then build the signature matrix
3. Find the minhash

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

# Picture

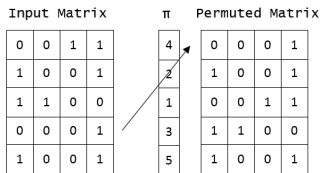


Figure 3: Permuted matrix from the characteristic one. The  $\pi$  vector is the specified permutation.

## Helper Function for Building Characteristic Matrix

```
# return if an item is in a list  
item_in_list <- function(item, list) {  
  as.integer(item %in% list)  
}
```



## Characteristic matrix

```
# get the characteristic matrix
# items are all the unique hash values
# columns will be each record
# we want to keep track of where each hash is included
char_mat <- data.frame(item = unique(unlist(hashled_shingles)))
# for each hashed shingle, see if it is in each row
contained <- lapply(hashled_shingles, function(col) {
  vapply(char_mat$item, FUN = item_in_list,
    FUN.VALUE = integer(1), list = col)
})
# list to matrix
char_mat <- do.call(cbind, contained)
# row names
rownames(char_mat) <- unique(unlist(hashled_shingles))
# column names
colnames(char_mat) <- paste("Record", seq_len(nrow(dat)))
```

## Characteristic matrix (continued)

```
# inspect results  
kable(char_mat[10:15, 1:5])
```

	Record 1	Record 2	Record 3	Record 4	Record 5
1146728047	1	1	1	1	1
-468563035	1	0	0	0	0
-1206963121	1	0	0	0	0
1666629753	1	1	1	1	1
-1773283382	1	1	1	1	1
1421345234	1	1	1	1	1

The result is a  $3562 \times 1879$  matrix

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

# Minhashing

We want to 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)

```
# set seed for reproducibility
set.seed(02082018)
# function to get signature for 1 permutation
get_sig <- function(char_mat) {
  # get permutation order
  permute_order <- sample(seq_len(nrow(char_mat)))
  # get min location of "1" for each column (apply(2, ...))
  t(apply(char_mat[permute_order, ], 2,
          function(col) min(which(col == 1))))
}
# repeat many times
m <- 360
sig_mat <- matrix(NA, nrow=m,
                  ncol=ncol(char_mat)) #empty matrix
for(i in 1:m) {
  sig_mat[i, ] <- get_sig(char_mat) #fill matrix
}
colnames(sig_mat) <- colnames(char_mat) #column names
```

## Minhashing (cont'd)

```
# inspect results  
kable(sig_mat[1:10, 1:5])
```

Record 1	Record 2	Record 3	Record 4	Record 5
3	3	3	3	3
38	38	38	38	38
32	32	32	32	32
3	3	3	3	3
101	101	101	101	101
70	70	70	70	70
153	153	153	153	153
36	36	36	36	36
55	55	55	55	55
44	44	44	44	44

## 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

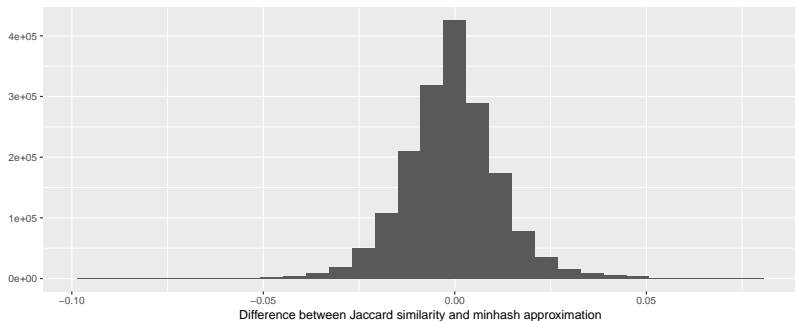
## Jaccard similarity approximation

```
# add jaccard similarity approximated from the minhash to compare  
# number of agreements over the total number of combinations  
hashed_jaccard$similarity_minhash <-  
  apply(hashed_jaccard, 1, function(row) {  
    sum(sig_mat[, row[["record1"]]]  
      == sig_mat[, row[["record2"]]])/nrow(sig_mat)  
  })  
  
# how far off is this approximation? plot differences  
qplot(hashed_jaccard$similarity_minhash - hashed_jaccard$similarity_minhash,  
       xlab("Difference between Jaccard similarity and minhash approximation"))
```

```
## Warning: `qplot()` was deprecated in ggplot2 3.4.0.  
## This warning is displayed once every 8 hours.  
## Call `lifecycle::last_lifecycle_warnings()` to see where this  
## generated.  
  
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`
```

# Jaccard similarity approximation

```
## `stat_bin()` using `bins = 30`. Pick better value with
```



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)



Wait did I miss something?

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

# Locality Sensitive Hashing (LSH)

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

```
# view the signature matrix
```

```
print(xtable::xtable(sig_mat[1:10, 1:5]), hline.after = c(-
```

	Record 1	Record 2	Record 3	Record 4	Record 5
1	3	3	3	3	3
2	38	38	38	38	38
3	32	32	32	32	32
4	3	3	3	3	3
5	101	101	101	101	101
6	70	70	70	70	70
7	153	153	153	153	153
8	36	36	36	36	36
9	55	55	55	55	55
10	44	44	44	44	44

# 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 $b$

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

```
# library to get divisors of m
library(numbers)

# look at probability of binned together for various bin s
bin_probs <- expand.grid(s = c(.25, .75), h = m, b = divisors(m))
bin_probs$prob <- apply(bin_probs, 1, function(x) lsh_prob(x))

# plot as curves
ggplot(bin_probs) +
  geom_line(aes(x = prob, y = b, colour = factor(s), group = factor(s))) +
  geom_point(aes(x = prob, y = b, colour = factor(s)), size = 100) +
  xlab("Probability") +
  scale_color_discrete("s")
```

```
## Warning: Using `size` aesthetic for lines was deprecated
## i Please use `linewidth` instead.
```

## “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)
```

## “Easy” LSH in R (Continued)

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

## “Easy” LSH in R (Continued)

```
# perform lsh to get buckets
```

```
buckets <- lsh(corpus, bands = b, progress = FALSE)
```

```
## Warning: `gather_()` was deprecated in tidyr 1.2.0.
```

```
## i Please use `gather()` instead.
```

```
## i The deprecated feature was likely used in the textreus
```

```
## Please report the issue at <https://github.com/ropensc>
```

```
## This warning is displayed once every 8 hours.
```

```
## Call `lifecycle::last_lifecycle_warnings()` to see where
```

```
## generated.
```

```
# grab candidate pairs
```

```
candidates <- lsh_candidates(buckets)
```

```
# get Jaccard similarities only for candidates
```

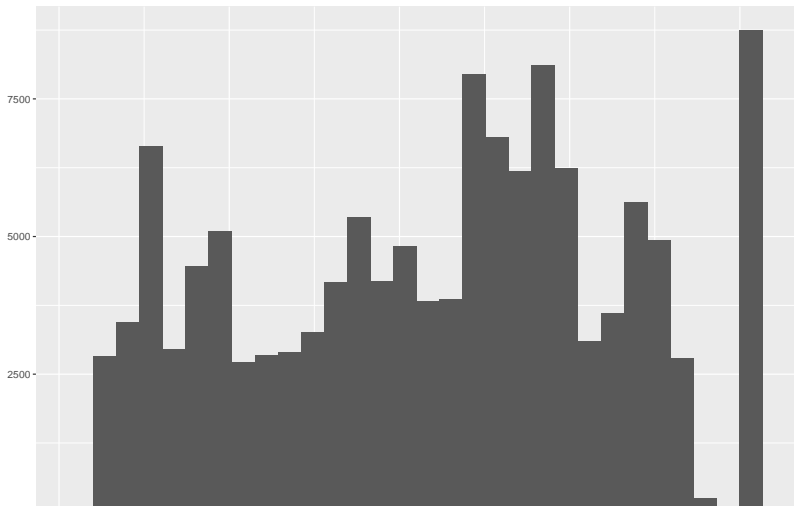
```
lsh_jaccard <- lsh_compare(candidates, corpus, jaccard_sim)
```



## “Easy” LSH in R (cont'd)

```
# plot jaccard similarities that are candidates  
qplot(lsh_jaccard$score)
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `
```



## 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
```

## Putting it all together

```
g <- make_empty_graph(n, directed = FALSE) # empty graph
g <- add_edges(g, is.vector((candidates[, 1:2]))) # candidates
g <- set_vertex_attr(g, "id", value = dat$id) # add id

# get clusters, 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     2
## 3   3     3
## 4   4     4
## 5   5     5
## 6   6     6
```

## 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