Module X: Probabilistic Blocking

Rebecca C. Steorts

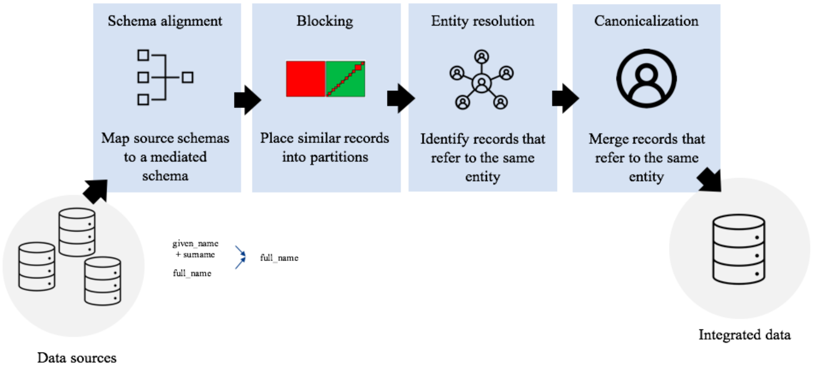
Agenda

- Data Cleaning Pipeline
- Blocking
- Probabilistic Blocking
- LSH

Load R packages

```
## Loading required package: DBI

## Loading required package: RSQLite

## Loading required package: ff

## Loading required package: bit

##
## Attaching package: 'bit'

## The following object is masked from 'package:base':
##
##     xor

## Attaching package ff

## - getOption("fftempdir")=="/var/folders/bv/xhclmwh90zg0

## - getOption("ffextension")=="ff"

## - getOption("ffdrop")==TRUE
```

Data Cleaning Pipeline



Schema alignment
Map source schemas to a mediated schema
Blocking
Place similar records into partitions
Entity resolution
Identify records that refer to the same entity
Canonicalization
Merge records that refer to the same entity
given_name + surname
full_name
full_name
Data sources
Integrated data

Figure 1: Data cleaning pipeline.

Blocking

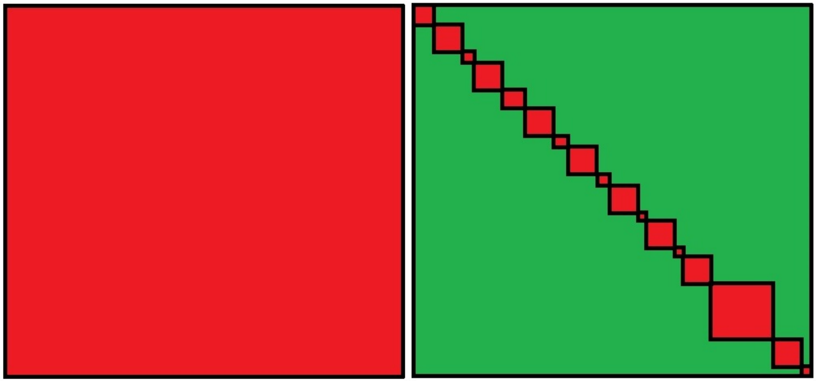


Figure 2: Left: All to all record comparison. Right: Example of resulting blocking partitions.

LSH

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

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

As already mentioned there are many ways to define similarity.

In this lecture, we will need the *Jaccard similarity*:

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

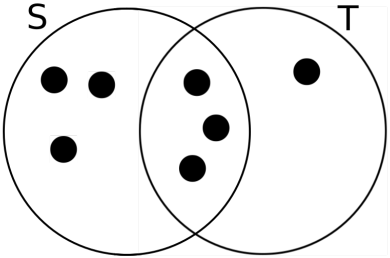


Figure 3: Two sets S and T with Jaccard similarity 3/7. The two sets share 3 elements in common, and there are 7 elements in total.

How to represent data as sets?

We want to talk about the similarity of our data (records)$\Rightarrow$ we need to compare sets of records!

- We can construct a set of **short strings** from the data
- This is useful because similar datasets will have many common elements (common short strings)
- We can do construct these short strings using *shingling*

k-shingling (how-to)

1. Think of our data set 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 data set the set of *k*-shingles that appear one or more times

Let's try

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

- the set of 2-shingles is $\{$he, el, ll, lo, ow, wo, or, rl, ld$\}$
- the set of 3-shingles is $\{$hel, ell, llo, low, owo, wor, orl, rld$\}$

Your turn

We have the following two records:

```r
# load RL data
data("RLdata500")

# select only 2 records
records <- RLdata500[129:130, c(1,3)]
names(records) <- c("First name", "Last name")

# inspect records
kable(records)
```

	First name	Last name
129	MICHAEL	VOGEL
130	MICHAEL	MEYER

Your turn (continued)

1. Compute the 2-shingles for each record

2. Using Jaccard similarity, how similar are they?

3. What do you learn from this exercise?

Your turn solution

1. The 2-shingles for the first record are $\{$mi, ic, ch, ha, ae, el, lv, vo, og, ge, el$\}$ and for the second are $\{$mi, ic, ch, ha, ae, el, lm, me, ey, ye, er$\}$
2. There are 6 items in common $\{$mi, ic, ch, ha, ae, el$\}$ and 15 items total $\{$mi, ic, ch, ha, ae, el, lv, vo, og, ge, lm, me, ey, ye, er$\}$, so the Jaccard similarity is $\frac{6}{15} = \frac{2}{5} = 0.4$
3. You should have learned that this is very tedious to do by hand!

Useful packages/functions in R

(Obviously) We don't want to do this by hand most times.

Here are some useful packages in R that can help us!

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

```
##
## Attaching package: 'tokenizers'

## The following objects are masked from 'package:textreuse
##
##     tokenize_ngrams, tokenize_sentences, tokenize_skip_
##     tokenize_words
```

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

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

Citation Data Set

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

```
library(devtools)
```

```
## Loading required package: usethis
```

```
install_github("resteorts/cora")
```

```
## Skipping install of 'cora' from a github remote, the SHA1 (70e32d5d) has not changed since last instal
##   Use `force = TRUE` to force installation
```

```
library(cora)
library(ggplot2)
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      : 'noquote' chr  "Inganas and M.R" NA NA NA ...
##  $ book_title : 'noquote' chr  NA NA NA NA ...
##  $ authors    : 'noquote' chr  "M. Ahlskog, J. Paloheimo, H. Stubb, P. Dyreklev, M. Fahlman, O" "M. Ah
##  $ address    : 'noquote' chr  NA NA NA NA ...
##  $ date       : 'noquote' chr  "1994" "1994" "1994" "1994" ...
##  $ year       : 'noquote' chr  NA NA NA NA ...
##  $ editor     : 'noquote' chr  NA NA NA NA ...
##  $ journal    : 'noquote' chr  "Andersson, J Appl. Phys." "JAppl. Phys." "J Appl. Phys." "J Appl.Phys."
##  $ volume     : 'noquote' chr  "76" "76" "76" "76" ...
##  $ pages      : 'noquote' chr  "893" "893" "893" "893" ...
##  $ publisher  : 'noquote' chr  NA NA NA NA ...
##  $ institution: 'noquote' chr  NA NA NA NA ...
##  $ type       : 'noquote' chr  NA NA NA NA ...
##  $ tech       : 'noquote' chr  NA NA NA NA ...
##  $ note       : 'noquote' chr  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`)

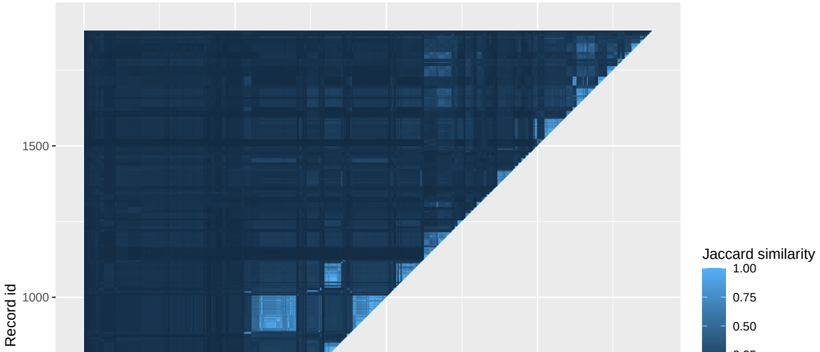
Your turn (solution)

```r
# 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 99.51 seconds ≈ 1.66 minutes

Your turn (solution, cont'd)

```
# plot the jaccard similarities for each pair of records
ggplot(jaccard) +
  geom_raster(aes(x = record1, y = record2,
                  fill=similarity)) +
  theme(aspect.ratio = 1) +
  scale_fill_gradient("Jaccard similarity") +
  xlab("Record id") + ylab("Record id")
```



Your turn (solution, cont'd)

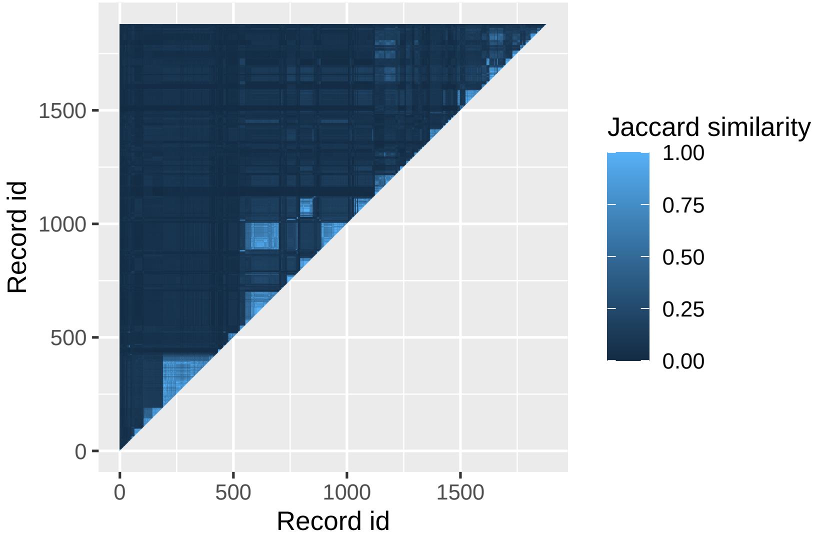


Figure 5: Jaccard similarity for each pair of records. Light blue indicates the two records are more similar and dark blue indicates less similar.

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 **low**.*

Hashing shingles

Instead of storing the strings as shines, we can instead store *hashed values*

These are integers, and they take up less space.

Hashing shingles

```r
# 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-shingles
  hash_string(shingles) # return hashed shingles
})
```

```r
# Jaccard similarity on hashed shingles
hashed_jaccard <-
  expand.grid(record1 = seq_len(n), record2 = seq_len(n))

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

time <- Sys.time() # see how long this takes
hashed_jaccard$similarity <-
  apply(hashed_jaccard, 1, function(pair) {
  jaccard_similarity(hashed_shingles[[pair[1]]],
                     hashed_shingles[[pair[2]]])
}) # get jaccard for each hashed pair
time <- difftime(Sys.time(), time, units = "secs") # timing
```

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

Characteristic matrix

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

# 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(hashed_shingles

# for each hashed shingle, see if it is in each row
contained <- lapply(hashed_shingles, function(col) {
  vapply(char_mat$item, FUN = item_in_list,
         FUN.VALUE = integer(1), list = col)
})

char_mat <- do.call(cbind, contained) # list to 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)

```r
# 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

# 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
46	46	46	46	46
36	36	36	36	36
31	31	31	31	31
124	124	124	124	124
21	21	21	21	21
9	9	9	9	9
85	85	85	85	85

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

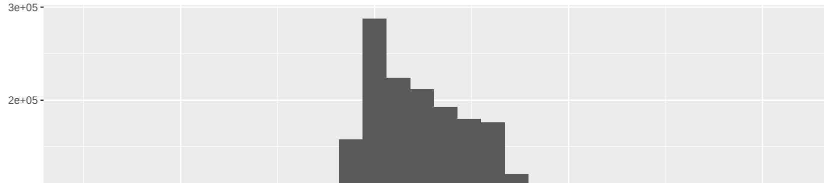
We look down each column of the signature matrix, and compare it to any other column

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

Jaccard similarity approximation

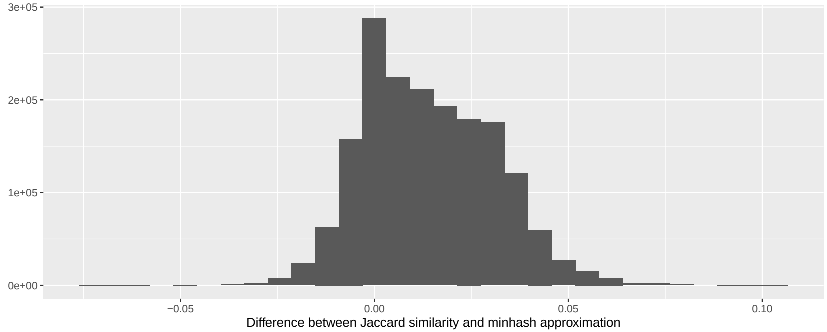
```
# add jaccard similarity approximated from the minhash to c
# number of agreements over the total number of combinatio
hashed_jaccard$similarity_minhash <-
  apply(hashed_jaccard, 1, function(row) {
  sum(sig_mat[, row[["record1"]]]
      == sig_mat[, row[["record2"]]])/nrow(sig_mat)
})

# how far off is this approximation? plot differences
qplot(hashed_jaccard$similarity_minhash - hashed_jaccard$si
  xlab("Difference between Jaccard similarity and minhash a
```

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



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)

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

```r
# 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	46	46	46	46	46
4	36	36	36	36	36
5	31	31	31	31	31
6	124	124	124	124	124
7	21	21	21	21	21
8	9	9	9	9	9
9	85	85	85	85	85
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*

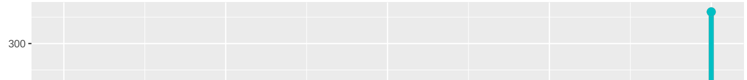
How to choose b

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 = diviso
bin_probs$prob <- apply(bin_probs, 1, function(x) lsh_proba


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



“Easy” LSH in R

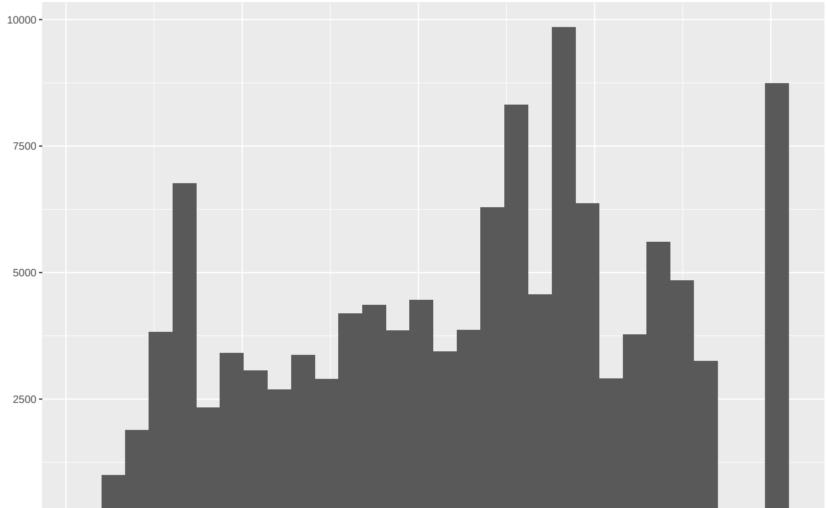
There an easy way to do LSH using the built in functions in the `textreuse` package via the functions `minhash_generator` and `lsh` (so we don’t have to perform it by hand):

```r
# 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)

```r
# 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

```r
library(igraph) #graph package
```

```
##
## Attaching package: 'igraph'

## The following objects are masked from 'package:stats':
##
##     decompose, spectrum

## The following object is masked from 'package:base':
##
##     union
```

```r
# 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, is.vector((candidates[, 1:2]))) # candid
g <- set_vertex_attr(g, "id", value = dat$id) # add id
```

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