Introduction to locality sensitive hashing

Andee Kaplan

Duke University
Department of Statistical Science
`andrea.kaplan@duke.edu`

February 8, 2018

Slides available at `http://bit.ly/cimat-lsh`

Goal and outline

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

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

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.

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

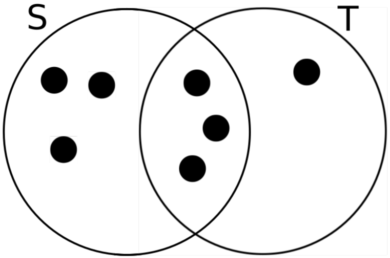


Figure: Two sets S and T with Jaccard similarity 3/7. The two sets share 3 elements in common, and there are 7 elements in total.

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" and $k = 2$, 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:

	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 {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 16 items total {mi, ic, ch, ha, ae, el, lv, vo, og, ge, el, lm, me, ey, ye, er}, so the Jaccard similarity is $\frac{6}{16} = \frac{3}{8} = 0.375$

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
# detecting text reuse and document similarity + shingles
library(textreuse)
library(tokenizers)
```

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)

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

Example data

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

```
library(RLdata)
data(cora)
str(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. Fahl
##  $ 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`).

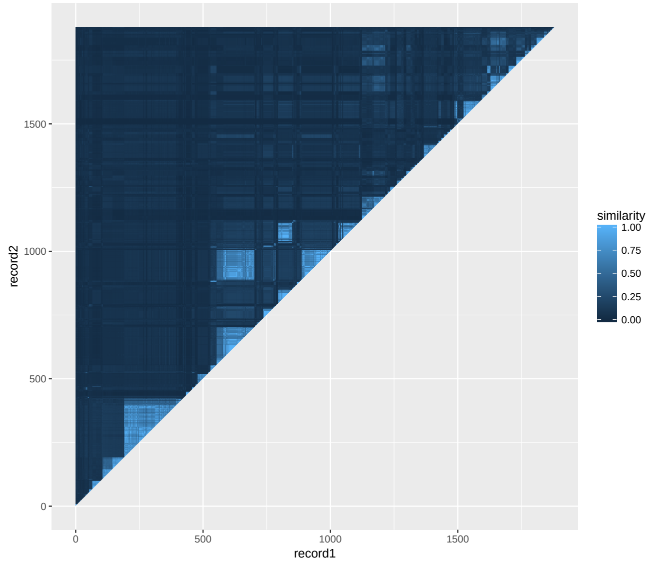
2. Obtain the Jaccard similarity between each pair of records (**hint:** use `jaccard_similarity`).

Your turn solution

```r
# get only the columns we want
dat <- cora[, c("title", "authors", "journal")]

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

# 2. Jaccard similarity between pairs
jaccard <- expand.grid(record1 = seq_len(nrow(dat)),
                       record2 = seq_len(nrow(dat)))

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

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

This took took 150.11 seconds $\approx$ 2.5 minutes

Your turn solution (cont'd)



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, collapse=" ")
  shingles <- tokenize_character_shingles(string, n = 3)[[1]]
  hash_string(shingles)
})
```

This took up 6.38256×10^5 bytes, while storing the shingles took 7.36544×10^6 bytes. However, the whole pairwise comparison still took the same amount of time (≈ 2.58 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
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, 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

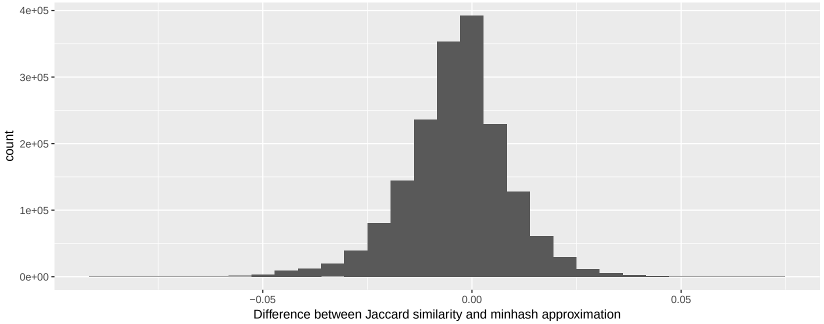
$$Pr\{\min[h(A)] = \min[h(B)]\} = \frac{|A \cap B|}{|A \cup B|}$$

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



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.

Any pair that is hashed to the same bucket for any hashing is called a *candidate pair* and we only check candidate pairs for similarity.

1. Divide up the signature matrix into b bands with r rows such that $m = b * r$ where m is the number of times that we drew a permutation of the characteristic matrix in the process of minhashing (number of rows in the signature matrix).
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. Each pair of documents has as many chances to be considered a candidate as there are bands, and the fewer rows there are in each band, the more likely it is that each document will match another.

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

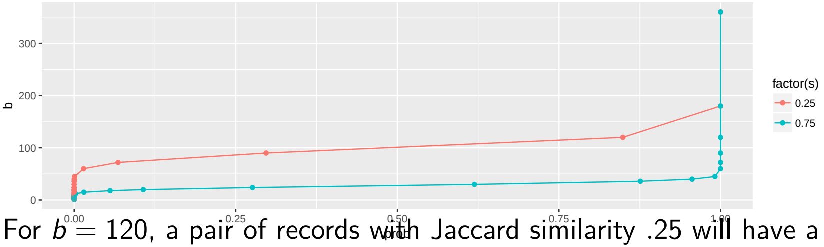
How large k should be depends on how long our data strings are. The important thing to remember is k should be picked large enough such that the probability of any given shingle is *low*.

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



For $b = 120$, a pair of records with Jaccard similarity .25 will have a 84.9% chance of being matched as candidates and a pair of records with Jaccard similarity .75 will have a 100% 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):

```r
b <- 120

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

# build the corpus using textreuse
docs <- apply(dat, 1, paste, collapse=" ")
corpus <- TextReuseCorpus(text = docs,
                          tokenizer = tokenize_character_shingles, n = 3, simplify = TRUE,
                          progress = FALSE,
                          keep_tokens = TRUE,
                          minhash_func = 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
lshed <- lsh_compare(candidates, corpus, jaccard_similarity, progress = FALSE)

kable(head(lshed[order(lshed$score),]))
```

a	b	score
doc-1014	doc-1256	0.0125000

Putting it all together

Your turn