Introduction to Locality Sensitive Hashing

STA 325, Supplemental Material

Andee Kaplan and Rebecca C. Steorts

Load Libraries

```
## Loading required package: DBI

## Loading required package: RSQLite

## Loading required package: ff

## Loading required package: bit

## Attaching package bit

## package:bit (c) 2008-2012 Jens Oehlschlaegel (GPL-2)

## creators: bit bitwhich

## coercion: as.logical as.integer as.bit as.bitwhich whic

## operator: ! & | xor != ==

## querying: print length any all min max range sum summary
```

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.

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?

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).

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.

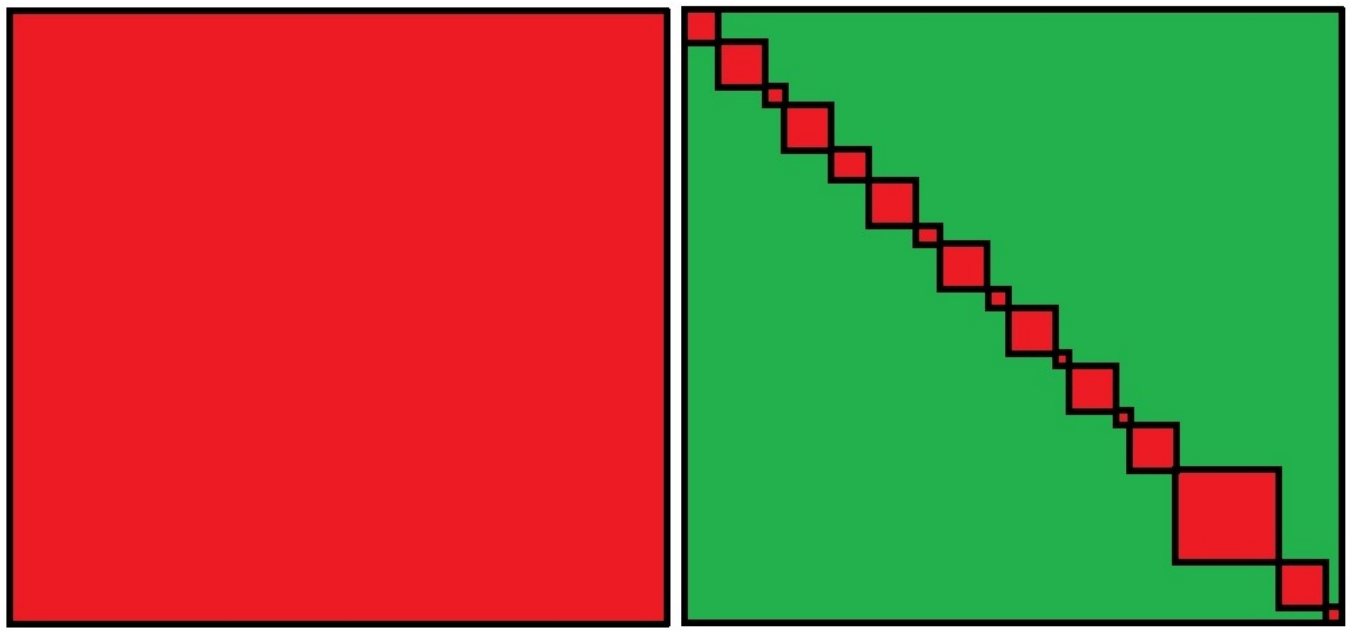


Figure 1: All-to-all data point comparisons (left) versus partitioning data points into blocks/bins (right).

Entity resolution

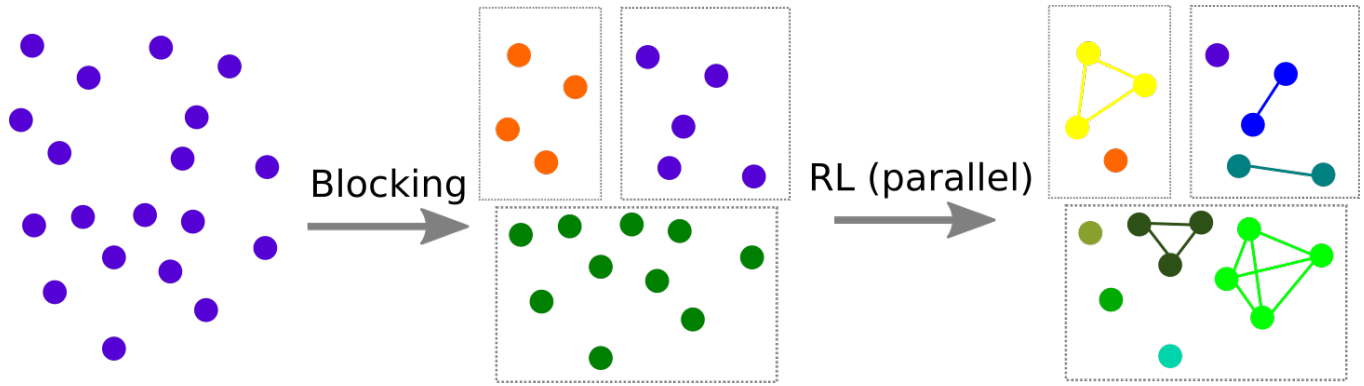


Figure 2: Entire process of the blocking step (via LSH) and then the record linkage step. Step 1: Dimension reduction via blocking. Step 2: Removing duplicate entities via record linkage.

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

Our motivating application will be a synthetic data set from the RecordLinkage package in R called RLdata500.

RLdata500 database

```
data(RLdata500)
head(RLdata500)
```

```
##   fname_c1 fname_c2 lname_c1 lname_c2   by bm bd
## 1  CARSTEN     <NA>     MEIER     <NA> 1949  7 22
## 2     GERD     <NA>     BAUER     <NA> 1968  7 27
## 3   ROBERT     <NA> HARTMANN     <NA> 1930  4 30
## 4   STEFAN     <NA>     WOLFF     <NA> 1957  9  2
## 5     RALF     <NA>  KRUEGER     <NA> 1966  1 13
## 6  JUERGEN     <NA>   FRANKE     <NA> 1929  7  4
```

```
head(identity.RLdata500)
```

```
## [1]  34  51 115 189  72 142
```

RLdata500 database

```
# 500 total records
dim(RLdata500)
```

```
## [1] 500   7
```

```
# 450 unique records (50 duplicate records)
length(unique(identity.RLdata500))
```

```
## [1] 450
```

Notions of Similarity

We have already discussed many notions of similarity, however, the main one that we work with in this module is the Jaccard similarity.

The Jaccard similarity of set S and T is

$$\frac{|S \cap T|}{|S \cup T|}$$

.

We will refer to the Jaccard similarity of S and T by $\mathsf{SIM}(S, T)$.

Jaccard Similarity

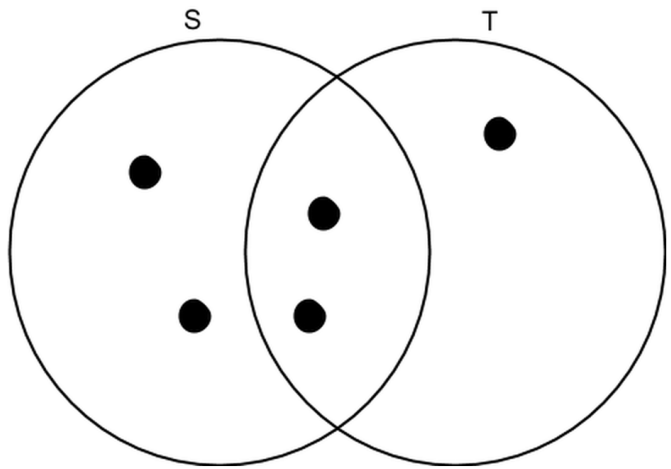


Figure 3: Two sets S and T with Jaccard similarity 2/5. The two sets share two elements in common, and there are five elements in total.

Overview of Locality Sensitive Hashing (LSH)

1. We first shingle a document (records, songs, etc), which breaks the document up into small subsets.
2. Next, we form a characteristic matrix using the shingled document. A characteristic matrix is an indicator matrix, where the columns of the matrix correspond to records (data points) in the document (so there is a column for each record) and the rows correspond to the shingles (so there is a row for each shingle).
3. The characteristic matrix is a large binary matrix. Next, we apply a permutation π to it to create a signature matrix which allows for some dimension reduction.
4. Based on the results of the signature matrix, we will observe the pairwise similarity of two records (data points). For the number of hash functions, there is an interesting relationship that we use between the columns for any given set of the signature matrix and a Jaccard similarity measure that we will eventually discover.

Shingling

- 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

Shingling (example)

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:

```
# 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

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

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

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

(Obviously) We don't want to do this by hand most times. Here are some useful packages in R that can help us!

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

```
## Warning: package 'tokenizers' was built under R version

##
## Attaching package: 'tokenizers'

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

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

Shingling and Jaccard similarity

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

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

Cora dataset

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 ...
```

Analyze cora dataset

```r
# get only the columns we want
(n <- nrow(cora)) # number of records
```

```
## [1] 1879
```

```r
dat <- data.frame(id = seq_len(n)) # create id column
dat <- cbind(dat, cora[, c("title", "authors", "journal")])
```

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)

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

Your turn (solution)

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

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

Your turn (solution)

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

This took took 131.41 seconds $\approx$ 2.19 minutes

Your turn (solution, cont'd)

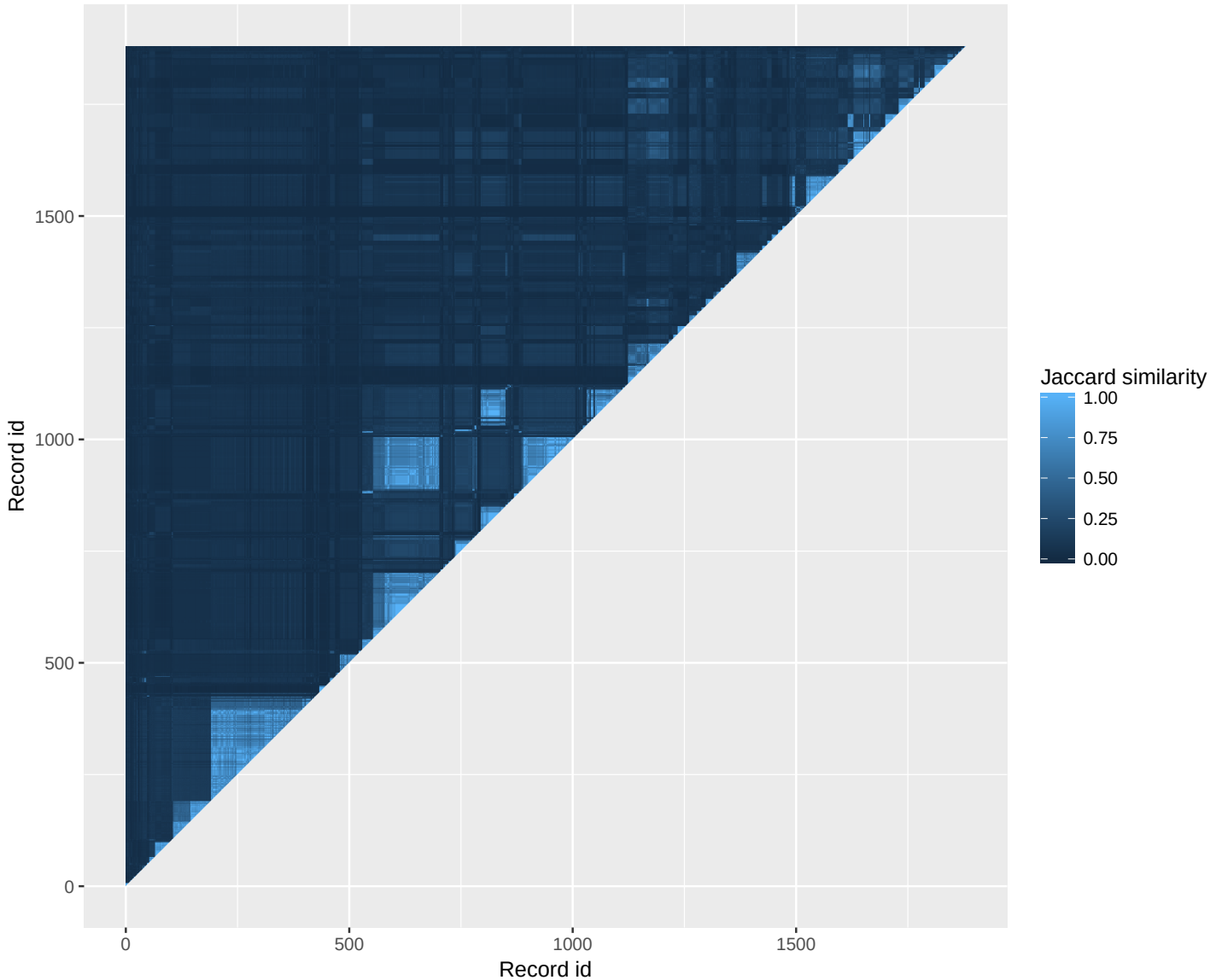


Figure 4: Jaccard similarity for each pair of records. Light blue indicates the two records are more similar and dark blue indicates less similar.

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 close together!

Locality sensitive hash (LSH) functions

LSH functions* $h()$ are defined in the following way:

- If records A and B have high similarity, then the probability that $h(A) = h(B)$ is **high**
- 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
})
```

```
# 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_jaccar

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[
}) # 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

```
# 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) {
```

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)

```
# 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(w
}

# repeat many times
m <- 360
sig_mat <- matrix(NA, nrow=m, ncol=ncol(char_mat)) #empty m
for(i in 1:m) {
  sig_mat[i, ] <- get_sig(char_mat) #fill matrix
}
```

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

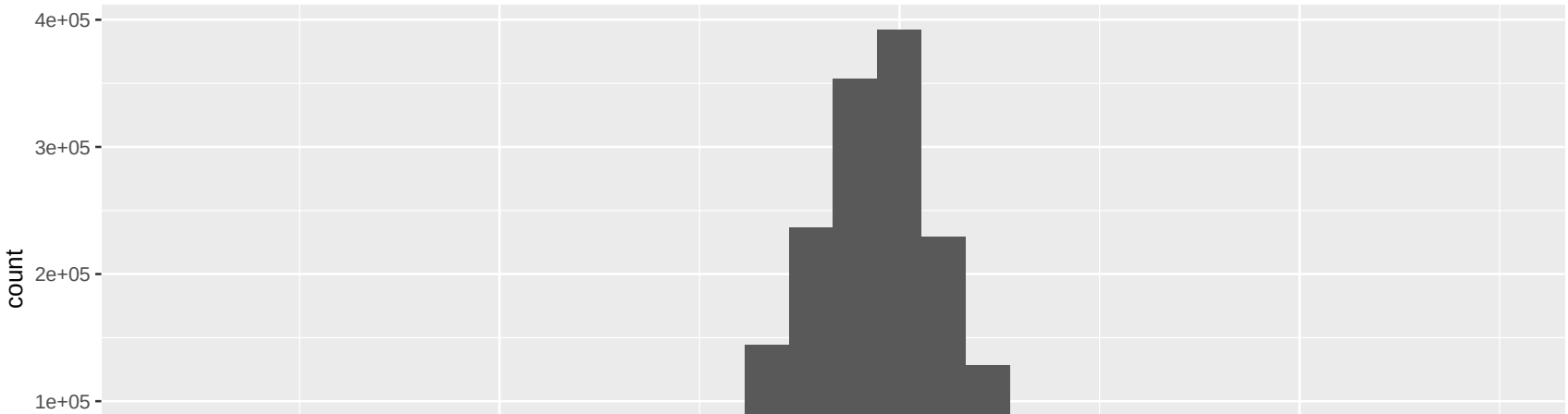
The number of agreements over the total number of combinations is an approximation to Jaccard measure

Jaccard similarity approximation

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

# 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 `
```



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**