Finding Similar Items and Locality Sensitive Hashing

STA 325, Supplemental Material

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.

Libraries

```
# load libraries
library(rvest) # web scraping
library(stringr) # string manipulation
library(dplyr) # data manipulation
library(tidyr) # tidy data
library(purrr) # functional programming
library(scales) # formatting for rmd output
library(ggplot2) # plots
library(numbers) # divisors function
library(textreuse) # detecting text reuse and
# document similarity
```

Background

We first go through a bit of R programming that you have not seen. This will not be a complete review, so please be sure to go through this on your own after the lecture.

Piping

There is a helpful R trick (used in many other languages) known as piping that we will use. Piping is available through R in the `magrittr` package and using the `{%>%}` command.

How does one use the pipe command? The pipe operator is used to insert an argument into a function. The pipe operator takes the left-hand side (LHS) of the pipe and uses it as the first argument of the function on the right-hand side (RHS) of the pipe.

Examples of Piping

We give two simple examples

```
# Example One
1:50 %>% mean
```

```
## [1] 25.5
```

```
mean(1:50)
```

```
## [1] 25.5
```

Examples of Piping

```
# Example Two
years <- factor(2008:2012)
# nesting
as.numeric(as.character(years))
```

```
## [1] 2008 2009 2010 2011 2012
```

```
# piping
years %>% as.character %>% as.numeric
```

```
## [1] 2008 2009 2010 2011 2012
```

The dplyr function

A very useful function that I recommend that you explore and learn about is called dplyr and there is a very good blogpost about it below: https://www.r-bloggers.com/useful-dplyr-functions-wexamples/

The `dplyr` function

Why should you learn and use `dplyr`?

- It will save you from writing your own complicated functions.
- It's also very useful as it will allow you to perform data cleaning, manipulation, visualisation, and analysis.

The `dplyr` function

There are many useful cheat sheets that have been made and you can find them here reading R and Rmarkdown in general that will help you with this module and optimizing your R code in general: `https://www.rstudio.com/resources/cheatsheets/`

We won't walk through an example as the blogpost above does a very nice job of going through functions within `dplyr`, such as `filter`, `mutate`, etc. Please go through these and your own become familiar with them.

Information Retrieval

One of the fundamental problems with having a lot of data is finding what you're looking for.

This is called **information retrieval**!

In this module, we will look at the problem of how to examine data for similar items for very large data sets.

Finding Similar Items

Suppose that we have a collection of webpages and we wish to find near-duplicate pages. (We might be looking for plagarism).

We will introduce how the problem of finding textually similar documents can be turned into a set using a method called shingling.

Then, we will introduce a concept called minhashing, which compresses large sets in such a way that we can deduce the similarity of the underlying sets from their compressed versions.

Finally, we will discuss an important problem that arises when we search for similar items of any kind, there can be too many pairs of items to test each pair for their degree of similarity. This concern motivates locality sensitive hashing, which focuses our search on pairs that are most likely to be similar.

Data

We will scrape lyrics from `http://www.metrolyrics.com`. (Please note that I do not expect you to be able to scrape the data, but the code is on the next two slides in case you are interested in it.)

```
# get beatles lyrics
links <- read_html("http://www.metrolyrics.com/beatles-lyrics.html") %>% # lyrics site
  html_nodes("td a") # get all links to all songs

# get all the links to song lyrics
tibble(name = links %>% html_text(trim = TRUE) %>% str_replace(" Lyrics", ""), # get song names
       url = links %>% html_attr("href")) -> songs # get links

# function to extract lyric text from individual sites
get_lyrics <- function(url){
  test <- try(url %>% read_html(), silent=T)
  if ("try-error" %in% class(test)) {
    # if you can't go to the link, return NA
    return(NA)
  } else
    url %>% read_html() %>%
      html_nodes(".verse") %>% # this is the text
      html_text() -> words

    words %>%
    paste(collapse = ' ') %>% # paste all paragraphs together as one string
      str_replace_all("[\r\n]" , ". ") %>% # remove newline chars
      return()
}
```

Data (continued)

```
# get all the lyrics
# remove duplicates and broken links
songs %>%
  mutate(lyrics = (map_chr(url, get_lyrics))) %>%
  filter(nchar(lyrics) > 0) %>% #remove broken links
  group_by(name) %>%
  mutate(num_copy = n()) %>%
  # remove exact duplicates (by name)
  filter(num_copy == 1) %>%
  select(-num_copy) -> songs
```

We end up with the lyrics to 62 Beatles songs and we can start to think about how to first represent the songs as collections of short strings (*shingling*). As an example, let's shingle the best Beatles' song of all time, "Eleanor Rigby".

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 $\mathrm{SIM}(S, T)$.

Figure 1: 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.

Shingling of Documents

The most effective way to represent document as sets is to construct from the document the set of short strings that appear within it.

Then the documents that share pieces as short strings will have many common elements.

Here, we introduce the most simple and common approach — shingling.

k-Shingles

A document can be represented as a string of characters.

We define a k-shingle (k-gram) for a document to be any sub-string of length k found within the document.

Then we can associate with each document the set of k-shingles that appear one or more times within the document.

k-Shingles

Example: Suppose our document is the string `abcdabd` and $k = 2$.

The set of 2-shingles is $\{ab, bc, cd, da, bd\}$.

Note the sub-string `ab` appears twice within the document but only once as a shingle.

Note it also makes sense to replace any sequence of one or more white space characters by a single blank. We can then distinguish shingles that cover two or more words from those that do not.

Back to the Beatles

- How would we create a set of k-shingles for the song "Eleanor Rigby"?
- We want to construct from the lyrics the set of short strings that appear within it.
- Then the songs that share pieces as short strings will have many common elements.
- We will get the k-shingles (sub-string of length k words found within the document) for "Eleanor Rigby" using the function `textreuse::tokenize_ngrams()`.

Shingling Eleanor Rigby

```
# get k = 5 shingles for "Eleanor Rigby"
# this song is the best
best_song <- songs %>% filter(name == "Eleanor Rigby")
# shingle the lyrics
shingles <- tokenize_ngrams(best_song$lyrics, n = 5)

# inspect results
head(shingles) %>%
  kable() # pretty table
```

ah look at all the
look at all the lonely
at all the lonely people
all the lonely people ah
the lonely people ah look
lonely people ah look at

Shingling Eleanor Rigby

- These are the $k = 5$-shingles for Eleanor Rigby, but what should k be?
- How large k should be depends on how long typical songs are.
- The important thing to remember is k should be picked large enough such that the probability of any given shingle in the document is low.

Shingling all Beatles songs

We can now apply the shingling to all 62 Beatles songs in our corpus.

```
# add shingles to each song
# note mutate adds new variables and preserves existing
songs %>%
  mutate(shingles = (map(lyrics, tokenize_ngrams, n = 3)))
```

Shingling Eleanor Rigby

- We will assume $k = 3$ here is sufficient.
- How could we validate our choice of k? What would we need?

(Think about this on your own for a few minutes or talk to the person next to you).

Choosing the shingle size

We could pick k to be any value that we wanted, but this leads to problems.

Suppose $k = 1$. Then most webpages will have most of the common characters and few other characters, so almost all webpages will have high similarity.

How large k should be depends on how long typical documents are and how large the set of typical characters is. The important thing to remember is:

- k should be picked large enough such that the probability of any given shingle in the document is low.

Choosing the shingle size

For example, if our corpus of documents is emails, then k=5, should be fine. Why?

Suppose that only letters and general white space characters appear in emails.

If so, there would be $27^5 = 14,348,907$ possible shingles.

Since the typical email is much smaller than 14 million characters long, we would expect $k = 5$ to work well and it does.

Choosing the shingle size

Surely, more than 27 characters appear in emails!

However, not all characters appear with equal probability.

Common letters and blanks dominate, while "z" and other letters that have high point values in Scrabble are rare.

Thus, even short emails will have many 5-shingles consisting of common letters and the chances of unrelated emails sharing these common shingles is greater than implied by the previous calculation.

A rule of thumb is to imagine there are 20 characters and estimate the number of shingles as 20^k. For large documents (research articles), a choice of $k = 9$ is considered to be safe.

Jaccard similarity of Beatles songs

Now that we have shingled records (sets), we can now look at the Jaccard similarity for each song by looking at the shingles for each song as a set (S_i) and computing the Jaccard similarity

$$\frac{|S_i \cap S_j|}{|S_i \cup S_j|} \text{ for } i \neq j.$$

We will use `textreuse::jaccard_similarity()` to perform these computations.

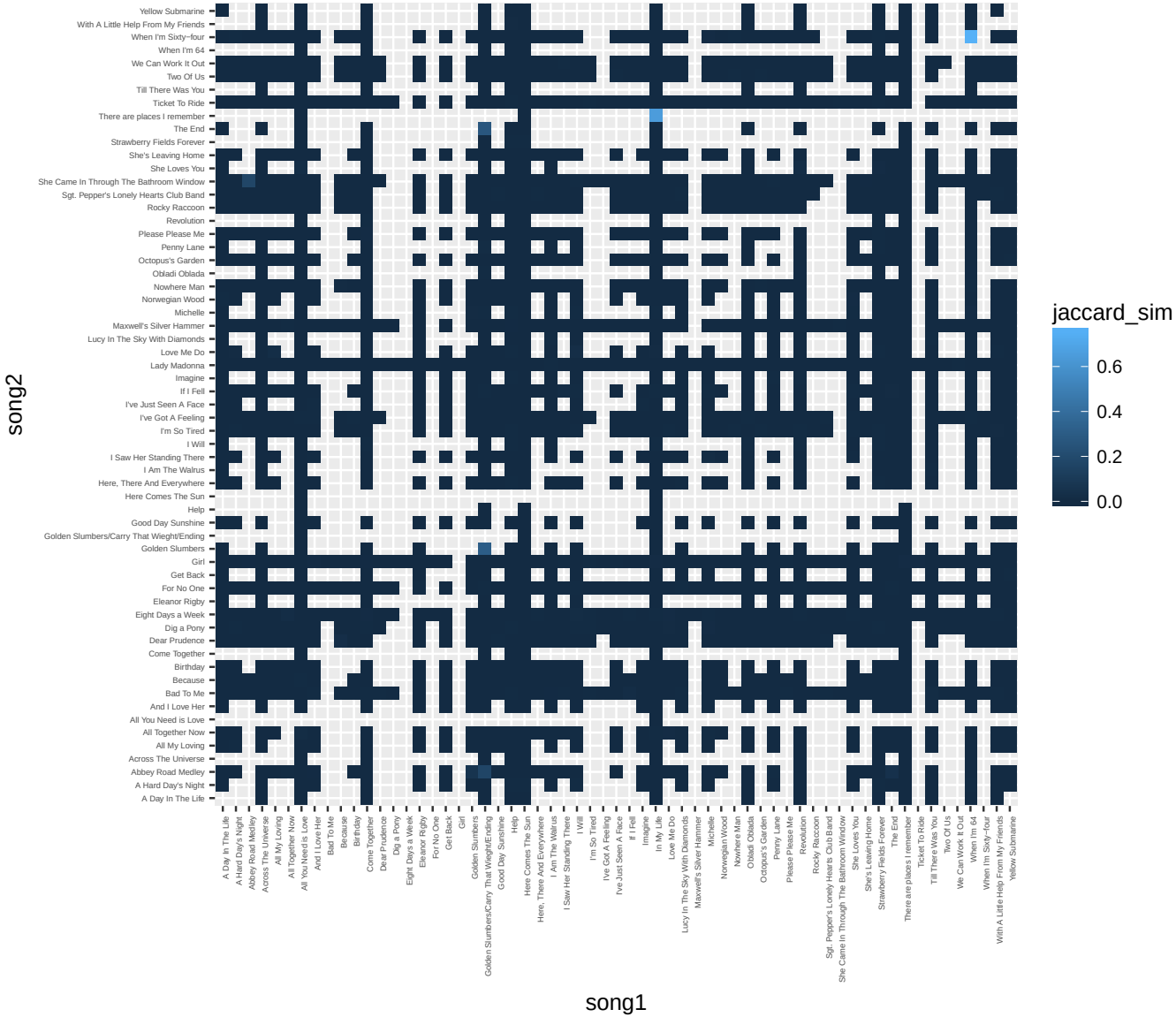
Jaccard similarity of Beatles songs

```
# create all pairs to compare then get the jacard similarity of each
# start by first getting all possible combinations
song_sim <- expand.grid(song1 = seq_len(nrow(songs)), song2 = seq_len(nrow(songs))) %>%
  filter(song1 < song2) %>% # don't need to compare the same things twice
  group_by(song1, song2) %>% # converts to a grouped table
  mutate(jaccard_sim = jaccard_similarity(songs$shingles[song1][[1]],
                                          songs$shingles[song2][[1]])) %>%
  ungroup() %>%  # Undoes the grouping
  mutate(song1 = songs$name[song1],
         song2 = songs$name[song2]) # store the names, not "id"

# inspect results
summary(song_sim)
```

```
##     song1              song2            jaccard_sim
##  Length:1891        Length:1891        Min.   :0.000000
##  Class :character   Class :character   1st Qu.:0.000000
##  Mode  :character   Mode  :character   Median :0.000000
##                                        Mean   :0.001826
##                                        3rd Qu.:0.000000
##                                        Max.   :0.757895
```

Jaccard similarity of Beatles songs



It looks like there are a couple song pairings with very high Jaccard similarity. Let's filter to find out which ones.

Jaccard similarity of Beatles songs

```
# filter high similarity pairs
song_sim %>%
  filter(jaccard_sim > .5) %>% # only look at those with similarity > .5
  kable() # pretty table
```

song1	song2	jaccard_sim
In My Life	There are places I remember	0.6488550
When I'm 64	When I'm Sixty-four	0.7578947

Two of the three matches seem to be from duplicate songs in the data, although it's interesting that their Jaccard similarity coefficients are not equal to 1. (This is not suprising. Why?). Perhaps these are different versions of the song.

Jaccard similarity of Beatles songs

```
# inspect lyrics
print(songs$lyrics[songs$name == "In My Life"])
```

```
## [1] "There are places I'll remember. All my life, though some have changed. Some forever, not for bette
```

```
print(songs$lyrics[songs$name == "There are places I remember"])
```

```
## [1] "There are places I remember all my life. Though some have changed. Some forever, not for better. S
```

By looking at the seemingly different pair of songs lyrics, we can see that these are actually the same song as well, but with slightly different transcription of the lyrics. Thus we have identified three songs that have duplicates in our database by using the Jaccard similarity. What about non-duplicates? Of these, which is the most similar pair of songs?

Jaccard dissimilarity of Beatles songs

```
# filter to find the songs that aren't very similar
# These are songs that we don't want to look at
song_sim %>%
  filter(jaccard_sim <= .5) %>%
  arrange(desc(jaccard_sim)) %>%  # sort by similarity
  head() %>%
  kable()
```

song1	song2	jaccard_sim
Golden Slumbers/Carry That Wieght/Ending	Golden Slumbers	0.3063063
Golden Slumbers/Carry That Wieght/Ending	The End	0.2666667
Abbey Road Medley	She Came In Through The Bathroom Window	0.1688742
Golden Slumbers/Carry That Wieght/Ending	Abbey Road Medley	0.1611842
Golden Slumbers	Abbey Road Medley	0.0560132
The End	Abbey Road Medley	0.0413907

These appear to not be the same songs, and hence, we don't want to look at these. Typically our goal is to look at items that are most similar and discard items that aren't similar.

Jaccard similarity of Beatles songs

Note, although we computed these pairwise comparisons manually, there is a set of functions that could have helped — `textreuse::TextReuseCorpus()`, `textreuse::pairwise_compare()` and `textreuse::pairwise_candidates()`.

Think about how you would do this and investigate this on your own (Exercise).

Hashing

A hash function maps objects to integers such that dissimilar objects are mapped far apart.

Formally a hash function $h()$ is defined such that

- if $\text{SIM}(A, B)$ is high, then with high probability $h(A) = h(B)$.
- if $\text{SIM}(A, B)$ is low, then with high probability $h(A) \neq h(B)$.

Hashing shingles

Instead of using substrings as shingles, we can use a hash function, which maps strings of length k to some number of buckets and treat the resulting bucket number as the shingle.

The set representing a document is then the set of integers of one or more k-shingles that appear in the document.

Example: We could construct the set of 9-shingles for a document. We could then map the set of 9-shingles to a bucket number in the range 0 to $2^{32} - 1$.

Thus, each shingle is represented by four bytes intead of 9. The data has been compressed and we can now manipulate (hashed) shingles by single-word machine operations.

Similarity Preserving Summaries of Sets

Sets of shingles are large. Even if we hash them to four bytes each, the space needed to store a set is still roughly four times the space taken up by the document.

If we have millions of documents, it may not be possible to store all the shingle-sets in memory. (There is another more serious problem, which we will address later in the module.)

Here, we replace large sets by smaller representations, called signatures. The important property we need for signatures is that we can compare the signatures of two sets and estimate the Jaccard similarity of the underlying sets from the signatures alone.

It is not possible for the signatures to give the exact similarity of the sets they represent, but the estimates they provide are close.

Characteristic Matrix

Before describing how to construct small signatures from large sets, we visualize a collection of sets as a characteristic matrix.

The columns correspond to sets and the rows correspond to the universal set from which the elements are drawn.

There is a 1 in row r, column c if the element for row r is a member of the set for column c. Otherwise, the value for (r, c) is 0.

Below is an example of a characteristic matrix, with four shingles and five records.

```
library("pander")
element <- c("a","b","c","d","e")
S1 <- c(0,1,1,0,1)
S2 <- c(0,0,1,0,0)
S3 <- c(1,0,0,0,0)
S4 <- c(1,1,0,1,1)
my.data <- cbind(element,S1,S2,S3,S4)
```

Characteristic Matrix

```
pandoc.table(my.data,style="rmarkdown")
```

```
##
##
## | element | S1 | S2 | S3 | S4 |
## |:-------:|:--:|:--:|:--:|:--:|
## |    a    | 0  | 0  | 1  | 1  |
## |    b    | 1  | 0  | 0  | 1  |
## |    c    | 1  | 1  | 0  | 0  |
## |    d    | 0  | 0  | 0  | 1  |
## |    e    | 1  | 0  | 0  | 1  |
```

Why would we not store the data as a characteristic matrix (think sparsity)?

Minhashing

The signatures we desire to construct for sets are composed of the results of a large number of calculations, say several hundred, each of which is a "minhash" of the characteristic matrix.

Here, we will learn how to compute the minhash in principle. Later, we will learn how a good approximation is computed in practice.

To minhash a set represented by a column of the characteristic matrix, pick a permutation of the rows.

The minhash value of any column is the index of the first row in the permuted order in which the column has a 1.

Minhashing

Now, we permute the rows of the characteristic matrix to form a permuted matrix.

The permuted matrix is simply a reordering of the original characteristic matrix, with the rows swapped in some arrangement.

Figure **??** shows the characteristic matrix converted to a permuted matrix by a given permutation. We repeat the permutation step for several iterations to obtain multiple permuted matrices.

Figure 2: Permuted matrix from the characteristic one. The π vector is the specified permutation.

The Signature Matrix

Now, we compute the signature matrix.

The signature matrix is a hashing of values from the permuted one.

The signature has a row for the number of permutations calculated, and a column corresponding with the columns of the permuted matrix.

We iterate over each column of the permuted matrix, and populate the signature matrix, row-wise, with the row index from the first 1 value found in the column.

The row index inputted to the signature matrix is the original row index the row was associated with, rather than the new index value. The signature matrix is below.

Signature Matrix

```
library("pander")
signature <- c(2,4,3,1)
pandoc.table(signature,style="rmarkdown")
```

```
##
## |   |   |   |   |
## |:-:|:-:|:-:|:-:|
## | 2 | 4 | 3 | 1 |
```

Minhashing and Jaccard Similarity

Based on the results of the signature matrix, we observe the pairwise similarity of two records. 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.

Minhashing and Jaccard Similarity

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

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

The equation means that the probability that the minimum values of the given hash function, in this case h, is the same for sets A and B is equivalent to the Jaccard Similarity, especially as the number of record comparisons increases.

We use this relationship to calculate the similarity between any two records.

We look down each column, and compare it to any other column: the number of agreements over the total number of combinations is equal to Jaccard measure.

General idea of LSH

Our general approach to LSH is to hash items several times such that similar items are more likely to be hashed into the same bucket (bin).

Candidate Pairs

Any pair that is hashed to the same bucket for any hashings is called a candidate pair.

We only check candidate pairs for similarity.

The hope is that most of the dissimilar pairs will never hash to the same bucket (and never be checked).

False Negative and False Positives

Dissimilar pairs that are hashed to the same bucket are called false positives.

We hope that most of the truly similar pairs will hash to the same bucket under at least one of the hash functions.

Those that don't are called false negatives.

False Negative and False Positives

Draw an example of false negatives and false positives.

How to choose the hashings

Suppose we have minhash signatures for the items.

Then an effective way to choose the hashings is to divide up the signature matrix into b bands with r rows.

For each band b, there is a hash function that takes vectors of r integers (the portion of one column within that band) and hashes them to some large number of buckets.

We can use the same hash function for all the bands, but we use a separate bucket array for each band, so columns with the same vector in different bands will hash to the same bucket.

Example

Consider the signature matrix

The second and fourth columns each have a column vector $[0, 2, 1]$ in the first band, so they will be mapped to the same bucket in the hashing for the first band.

Regardless of the other columns in the other bands, this pair of columns will be a candidate pair.

Example (continued)

It's possible that other columns such as the first two will hash to the same bucket according to the hashing in the first band.

But their column vectors are different, $[1, 3, 0]$ and $[0, 2, 1]$ and there are many buckets for each hashing, so an accidential collision here is thought to be small.

Analysis of the banding technique

You can learn some about the analysis of the banding technique on your own in Mining Massive Datasets, Ch 3, 3.4.2.

Combining the Techniques

We can now give an approach to find the set of candidate pairs for similar documents and then find similar documents among these.

1. Pick a value k and construct from each document the set of k-shingles. (Optionally, hash the k-shingles to shorter bucket numbers).
2. Sort the document-shingle pairs to order them by shingle.
3. Pick a length n for the minhash signatures. Feed the sorted list to the algorithm to compute the minhash signatures for all the documents.
4. Choose a threshold t that defines how similar doucments have to be in order to be regarded a similar pair. Pick a number b bands and r rows such that $br = n$ and then the threshold is approximately $(1/b)^{1/r}$.
5. Construct candidate pairs by applying LSH.
6. Examine each candidate pair's signatures and determine whether the fraction of components they agree is at least t.
7. Optionally, if the signatures are sufficiently similar, go to the documents themselves and check that they are truly similar.