Introduction to Data Science

GIRI NARASIMHAN, SCIS, FIU

Jaccard Similarity

- Defined on 2 sets, S and T
  - SIM(S,T) = $|S \cap T| / |S \cup T|$
- E.g., Documents and Web pages can be thought of as set of words
- *Bag Similarity* uses **bags** instead of sets

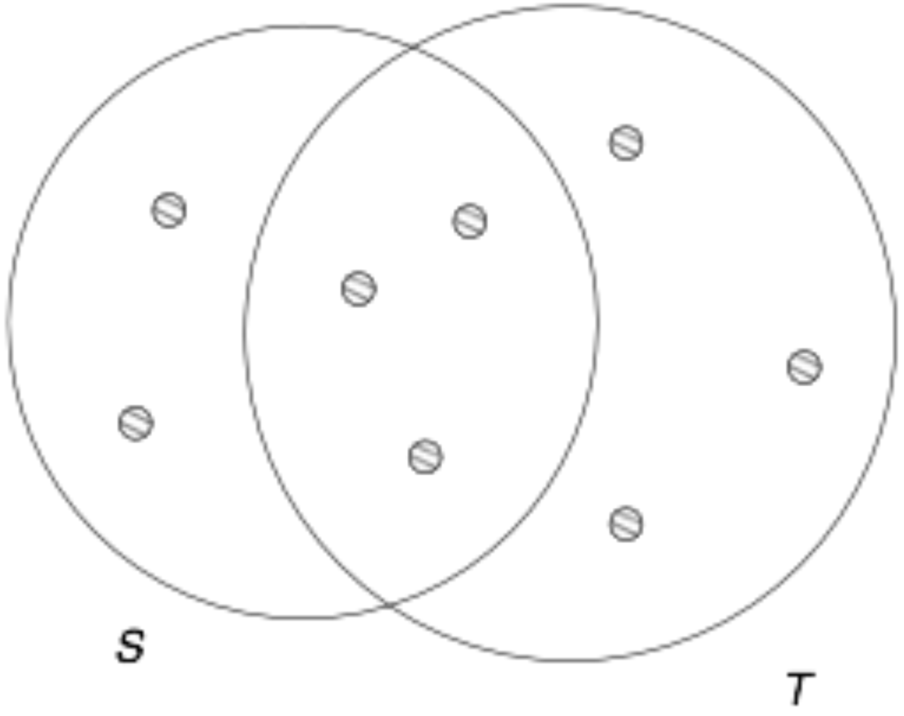


Figure 3.1: Two sets with Jaccard similarity 3/8

Small Signatures and MinHash

- Permute the rows
- Minhash(S_i) = row number of the first 1 in column S_i
- Minhash of the 4 columns are:
  - (a, c, b, a)
- Pr{Minhash(S_i) = Minhash(S_j)} equals
  - Jaccard similarity SIM(S_i, S_j)
- MinhashSignature(S_i) = result from N perm
  - Say N = 100

Element	S_1	S_2	S_3	S_4
b	0	0	1	0
e	0	0	1	0
a	1	0	0	1
d	1	0	1	1
c	0	1	0	1

Computing Minhash Signatures

- **Permuting** a large characteristic matrix is too **expensive**
- **Simulate** permutations using **hashing**
  - It is a close **approximation**, except for collisions
  - Ignore **collisions**, which cause **errors** in the computation
  - **Sparsity** helps in lowering the errors
  - Instead of N permutations, we pick N hash functions
    - $h_1, h_2, \ldots, h_N$

Computing Minhash Signatures

- Given hash function h_1, h_2, …, h_N, we want to compute MinHash values
- Let SIG(k,c) = signature matrix for k-th hash function and column c
- For row r, compute $h_1(r)$, $h_2(r)$, …, $h_N(r)$
- If col c has 0 in row r, do nothing
- Else, for each k = 1, 2, …, N,
  - set SIG(k,c) = min{SIG(k,c), $h_k(r)$}
- Initialize all SIG values to infty

Row	S_1	S_2	S_3	S_4	$x + 1 \bmod 5$	$3x + 1 \bmod 5$
0	1	0	0	1	1	1
1	0	0	1	0	2	4
2	0	1	0	1	3	2
3	1	0	1	1	4	0
4	0	0	1	0	0	3

	S_1	S_2	S_3	S_4
h_1	∞	∞	∞	∞
h_2	∞	∞	∞	∞

	S_1	S_2	S_3	S_4
h_1	1	∞	∞	1
h_2	1	∞	∞	1

	S_1	S_2	S_3	S_4
h_1	1	∞	2	1
h_2	1	∞	4	1

	S_1	S_2	S_3	S_4
h_1	1	3	2	1
h_2	1	2	4	1

	S_1	S_2	S_3	S_4
h_1	1	3	2	1
h_2	0	2	0	0

	S_1	S_2	S_3	S_4
h_1	1	3	0	1
h_2	0	2	0	0

Pair	True SIM	Approx SIM
(1,2)	0	0
(1,4)	2/3	1
(3,4)	1/5	1/2

Row	S_1	S_2	S_3	S_4	$x + 1 \mod 5$	$3x + 1 \mod 5$
0	1	0	0	1	1	1
1	0	0	1	0	2	4
2	0	1	0	1	3	2
3	1	0	1	1	4	0
4	0	0	1	0	0	3

h_1 h_2

Minhash Overview

- Takes very large documents and computes small signatures such that
  - Jaccard Similarity is (approximately) retained
- **Example**: 1 M docs, N = 250 hash functions; 4 bytes per hash value
  - 1 KB per doc signature
  - 1 GB to store all signatures for all 1 M docs
  - 0.5 Trillion pairs of docs
  - Similarity computation = 1 microsec
  - To compute all pairs = ~ 6 days (= 0.5184 trillion microsecs)

Find Closest Pair of Documents

- Cannot wait 6 days for an answer
- Clustering algorithms need this repeatedly
- **Approach**: Use a special hash function
  - Hash items so that similar items are likely to end up in the same bucket.
  - Avoid pairs in different buckets & reduce number of pairs to inspect
- These hash functions are called **Locality Sensitive Hashing** (**LSH**)
- Small Prob of error due to hashing
  - False Positives (cause extra work) and False Negatives (miss good pairs)

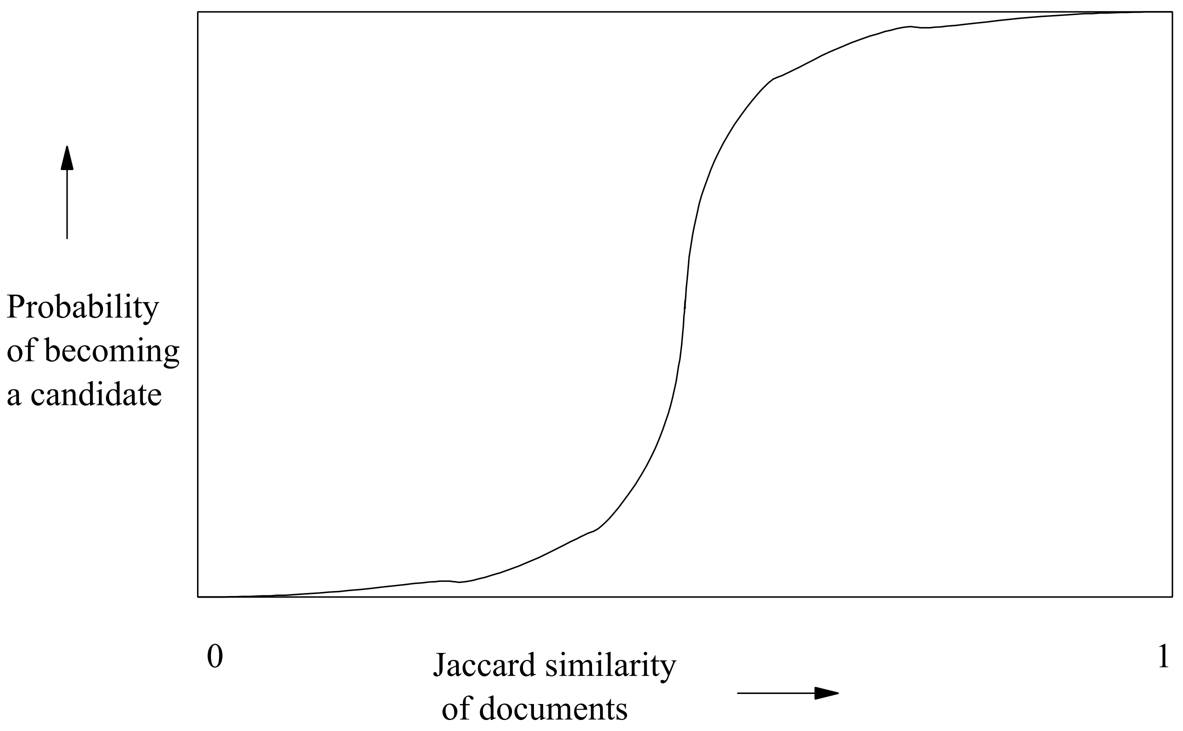
LSH for MinHash

- Divide signature matrix into b bands of r rows each
- For each band, hash column vector of r items to large # of buckets
- Use same hash function for each band but use separate buckets
  - Use different sets of buckets for different bands
- Any pair that appears in the same bucket in any band becomes a candidate for further inspection. All other pairs are discarded.
- If 2 columns are similar, then they must be identical in at least 1 band
- Each pair gets b chances to be in the same bucket

Analysis of LSH with Banding

- Assume b bands and r rows
- Consider a pair of docs with similarity value s
- Prob that their Minhash signatures agree in any particular row = s
- We want prob that this pair of docs becomes a candidate
- Prob signatures agree in all rows of one band = s^r
- Prob signature disagrees in at least one row of a band = $1 - s^r$
- Prob signatures disagree in at least one row in each band = $(1-s^r)^b$
- Prob that signatures agree in all rows of at least one band = $1 - (1-s^r)^b$

Behavior of $1 - (1-s^r)^b$



- **Independent** of **b** and **r**
  - Curve has to get from (0,0) to (1,1)
  - It's always an **S-curve**
- Threshold = value of s at steep rise
  - > threshold, pair is likely a candidate
  - Set (b,r) to achieve desired threshold

LSH-based Algorithm for Similar Items

- Pick k and construct k-shingles from each document
- Pick t, b, and r ($t \sim (1/b)^{1/r}$)
- Pick n = br hash functions
- Apply LSH technique, find candidates, check true similarity

Distance Measures

- A distance measure D must satisfy the following properties
  - **Non-negativity**: D(x,y) >= 0
    - D(x,y) = 0 if and only if x = y
  - **Symmetry**: D(x,y) = D(y,x)
  - **Triangle Inequality**: D(x,y) < = D(x,z) + D(z,y)

Important Distance Measures

- $D([x1, \ldots, xn], [y1, \ldots, yn]) = (|x1-y1|^r + \ldots + |xn-yn|^r)^{1/r}$
- If r= 2, this is the standard **Euclidean distance**
- Other values are commonly referred to as **Euclidean norms**
- **Jaccard Distance** = 1 – Jaccard Similarity
- **Cosine Distance** = **Dot Product** of 2 vectors
- **Edit Distance** = measure of changes to turn **x** into **y**
- **Hamming Distance** = # of components in which 2 vectors differ

Finding Identical Items

- LSH works for items with low similarity
- What if we only want to find identical items
  - Not good just to look at say first few characters
  - Not good to compare entire documents to check
  - Even if we hashed, we would need too many buckets
  - Idea: **Compute hash value based on random positions**

Finding near-identical items

- Advanced topic – please read from text.

Streaming

The Stream Model

- Data arrives in a stream
- Data is arriving rapidly
- Data cannot be stored in local storage, but in archival storage
- Archival storage, if any, is too large and cannot be accessed quickly
- Archival storage cannot be searched quickly
- If stream data is not processed immediately, then it is lost
- Decisions have to be made based on the data
- Quick approximate answer is often better than slow exact answer

Examples

- Wall street stock market data
- Satellite image data
- Internet and web traffic data
- Sensor data
  - 4-byte data every 0.1 sec = 3.5 MB/day
  - 1 million sensors in the ocean corresponds to one e
  - 40 MB every sec

Modern Times

Queries

- Alert when temperature is above 25 degrees
- Sliding window concept
  - Maximum temperature for period X
  - Alert when average for X is above 25 degrees
  - Number of unique elements for X

Standard Trick: Random Sampling

- **Random Sampling**: Pick a random integer from [0 .. N-1] and if 0, process the stream data, else ignore it.
  - Samples 1/N items
- It artificially **slows down the stream** to manageable levels

Sampling Woes

- **Stream**: Tuples (**user**, **query**, **time**); **Sampling**: 1 in 10
  - Each user has 1/10 of their queries processed
- **Query**: Fraction of typical user's queries repeated over last month
- **Correct Answer**: Suppose user has s unique queries and d queries twice and NO queries more than twice in the last month; Answer = d/(s+d)
- **Problem**: Reported fraction would be wrong
  - In the sampled stream, s/10 are unique queries and d/100 queries appear twice
  - The remainder of the queries that should appear twice will appear once 18d/100
  - We will report d/(10s + 19d) [d/100 twice and s/10 + 18d/100 once

Improved Solution for Sampling Woes

- Problem is that we are picking 1/10 of the queries
- We need to pick 1/10 of the users and pick all their queries
- If we can store 1/10 of the users, then for every query we can decide either to process or not
- **Improved Solution**: Hash user ID (actually, IP address) to 0 … 9
  - Pick only those that hash to 0
- **Sampling Question**: How to sample at rate of 1/70?
- **Sampling Question**: How to sample at rate of 23/70?

Sampling

- Sampling can be applied if the filtering test is easy (e.g., hash value = 0? Temperature > 22 degrees?)
- Sampling is harder if it involves a lookup (e.g., has this query been asked before by this user? Is this user among the top 10% of the frequent users list?)
- Other techniques are available for filtering
  - **Bloom Filters**

Example: Bloom Filters for Spam

- **White lists**: allowed email addresses
  - Assume we have **1 Billion** allowed email addresses
  - Assume black list is much larger than white list
  - If each email address is 20 bytes, this takes 20 GB to store
- **Bloom Filters**: store white lists as bit hash arrays
  - Every email address is hashed and a 1 is stored in the location if it is in white list
  - In 1 GB, we can store hash array of size 8 Billion
- Strict White Lists: use bloom filters and then verify with real white list
- Stricter White List: use cascade of bloom filters

Bloom Filters: Test for Membership

- Array of n bits, initially all 0's
- Collection of k hash functions. Each hash func maps a key to n buckets
- Given key K, compute K hash values and
  - Check that each location in bit array is a 1
  - Even if one is 0, then it fails the test

False Positive Rate

$(1\text{-}h)^{1/h} = e$ f
small h

- Assume we have **x** targets and **y** darts
- Prob a dart will hit a specific target = 1/x
- Prob a dart does not hit a specific target = 1 – (1/x) = (x-1)/x
- Prob that y darts miss a specific target = $((x-1)/x)^y$
- Prob that y darts miss a specific target = $e^{-y/x}$
- Let x = 8B; y = 1B; Then prob of missing a target = $e^{-1/8}$
- Prob of hitting a target = false positive rate = $1 - e^{-1/8} = 0.1175$
- If k = 2, the prob becomes $(1 - e^{-1/4})^2 = 0.0493$

False Positive Rate

- Let n = bit array length = 8B
- Let m = # of members = 1B
- Let k = # of hash functions = 1
- Prob that a white list email hashes to a location = 10^{-9}

Counting distinct elements

- How many unique users in a give period?
- How many users (IP addresses) visited a webpage?
  - Each IP address is 4 bytes = 32 bits
  - 4 billion IP addresses are possible = 16 GB
  - If we need this for each webpage and there are thousands, then we cannot store in memory

Flajolet-Martin Algorithm

- For each element obtain a sufficiently long hash
  - Has to be more possible results of hash than elements in the universal set
  - Example, use 64 bits ($2^{64} \sim 10^{19}$) to hash URLs (4 Billion)
  - High prob that different elements get different hash values
  - Some fraction of these hash values will be "unusual"
- We will focus on the ones that have r 0s at the end of its hash value
  - Prob of hash value to end in r 0s is 2^{-r}
  - Prob that m unique items have has values that don't end in r 0s is $(1-2^{-r})^m = e^{-m2^{-r}}$

Summary

- Look at the probability = $e^{-m2^{-r}}$
- If m is much larger than 2^r, then prob approaches 1
- If m is much smaller than 2^r, then prob approaches 0
- Thus 2^R is a good choice, where R is the largest tail of 0s

Moments

- i-th Momemt
- Zeroth Moment
- First Moment
- Average = ?
- Variance = ?

$$\frac{1}{m}\sum_{s=1}^{m}\left(f_s - \frac{n}{m}\right)^2 = \frac{1}{m}\sum_{s=1}^{m}\left(f_s^2 - 2\frac{n}{m}f_s + \left(\frac{n}{m}\right)^2\right) = \left(\frac{1}{m}\sum_{s=1}^{m}f_s^2\right) - \frac{n^2}{m^2}$$