APAN PS5430
Applied Text & Natural Language Analytics
Week 7: Locality Sensitive Hashing

Javid Huseynov, Ph.D.
Thursday, March 5, 2020

Week 7 Agenda

- Deduplication
- Dictionaries
- Hashing, Hash Functions & Tables
- Distance & Similarity Measures
  - Hamming
  - Jaccard
- Locality Sensitive Hashing
  - SimHash & MinHash
- Class Exercise

Background

With the growing volume of data on the Internet, the following tradeoffs are considered:

- Time vs. Space
- Exactness vs. Approximation

Duplicate Detection

- Given a large set of documents, find the near duplicates
- Documents:
  - Newsfeeds, webpages, etc.
- Distance/Similarity:
  - Hamming, Jaccard, Levenshtein, etc.
- All-pair similarity impractical

Clustering

- Given a set of objects with pairwise distances, group them into clusters
- Objects:
  - Documents, points in space, newsfeeds, etc.
- Distances = L1, L2, …
- Sub-quadratic solutions are desirable

Near Neighbor Search

- Given a set of objects and a new object, find the closest match in the set
- Objects:
  - Documents, points, images, …
- Distances = L1, L2, …
- Query time should be small

Dictionaries, e.g. in Python

- Stores **values** associated with user-specified **keys**:
  - **Keys** may be any homogenous comparable type
  - **Values** may be any homogenous type
  - Python Example: `ticker[”TSLA”]=“Tesla Inc.”`
- Operations:
  - Create, destroy, insert, find, delete
  - What is the efficiency?
- Representations:
  - Arrays, linked lists
  - Trees (Binary Search, etc.)
  - Hash tables

INSERT

FIND (‘WMT’)

Key	Value
AAPL	Apple Inc.
AMZN	Amazon Inc.
BA	Boeing Company
TSLA	Tesla Inc.
WMT	Walmart

Hashing

Rationale

- Sequential search (linked list, array) takes **O(*n*)** time, proportional to ***n*** samples
- Binary search improves on linear search with **O(*log n*)** time, but the worst-case still **O(*n*)**
- Suppose, we want to store 10,000 student records with 5-digit IDs:
  1. Linked list **O(*n*)** time
  2. Binary search **O(*log n*)** time
  3. Initialize with array of size 100,000 - **O(1)**, but big space use!!

Intuition

- Come up with a function to map the large range into small one, easier to manage
- Constant **O(1)** time per operation
- Example:
  - If Student ***A*** has ID ***k***, then ***A***'s record is stored in a table position ***h***(***k***), where ***h*** is a hash function

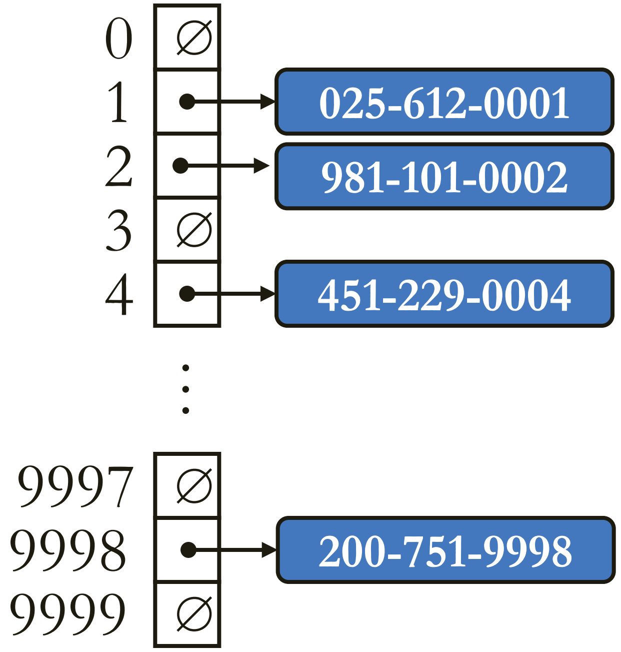
Hash Functions & Hash Tables

- A hash function h maps keys of a given type to integers in a fixed interval $[0, N - 1]$
- Example:

  $$h(k) = k \bmod N$$

  is a hash function for integer keys
- The integer $h(k)$ is called the hash value of key k
- A hash table consists of
  - Hash function h
  - Array (called table) of size N
- When implementing a map with a hash table, the goal is to store item (k, v) at index $i = h(k)$

- Hash Table for storing SS#
- h(x) = last 4 digits of SS#



- Collision?

Real Hash Functions

Name	Introduced	Weakened	Broken	Lifetime	Replacement
MD4	1990	1991	1995	1-5y	MD5
MD5	1992	1994	2004	8-10y	SHA-1
MD2	1992	1995	abandoned	3y	SHA-1
RIPEMD	1992	1997	2004	5-12y	RIPEMD-160
HAVAL-128	1992	-	2004	12y	SHA-1
SHA-0	1993	1998	2004	5-11y	SHA-1
SHA-1	1995	2004	not quite yet	9+	SHA-2 & 3
SHA-2 (256, 384, 512)	2001	still good			
SHA-3	2012	brand new			

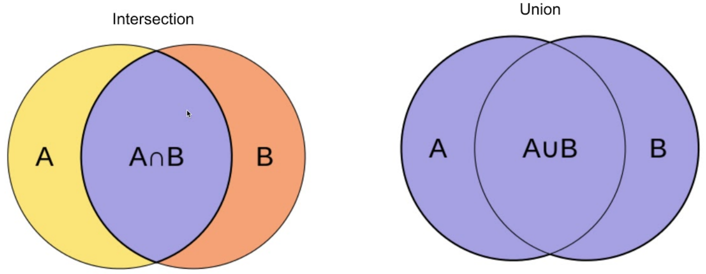
Distance and Similarity

Hamming Similarity (HS)

- Given two n-bit vectors x and y
  - $\mathbf{HS}(x, y) = \#\{\, i : x_i = y_i \,\}/n$
- Disjoint vectors have $\mathbf{HS}(x, y) = 0$
- $\mathbf{HS}(x, x) = 1$
- $x = 01001$, $y = 10011$, $\mathbf{HS}(x, y) = 2/5$
- Hamming distance $= 1 - \mathbf{HS}(x, y) * n$
- Examples:
  - "**karolin**" vs "**kathrin**" => **3**
  - **01001** vs **10011** => **3**

Jaccard Similarity (J)

- Given two sets **A** and **B**
  - $\mathbf{J}(A, B) = |A \cap B| \,/\, |A \cup B|$
- Disjoint sets have $\mathbf{J}(A, B) = 0$
- $\mathbf{J}(A, A) = 1$
- $A = \{1, 2\}$, $B = \{2, 3\}$, $J(A, B) = 1/3$
- Jaccard distance $= 1 - J(A, B)$



$$J(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

Locality Sensitive Hashing (LSH)

- U = Universe of objects
- $S: U \times U \rightarrow [0, 1]$ = Similarity function

An LSH for a similarity S is a probability distribution over a set $\mathcal{H}$ of hash functions such that for each $A, B \in U$:

$$\Pr_{h \in \mathcal{H}} [h(A) = h(B)] = S(A, B)$$

- LSHs represent similarities between objects
  - **SimHash**: based on Hamming distance, used by Google for deduplication of URLs
  - **MinHash**: based on Jaccard distance, used by Google for deduplication of newsfeeds

SimHash

- Given **f**-bit fingerprint (***F***), find all fingerprints that differ by at most ***k*** bits
- Create tables containing the fingerprints
  - Each table has a permutation $\boldsymbol{\pi}$ and a small integer ***p*** associated with it
  - Apply the permutation associated with the table of fingerprints
  - Sort the tables
- Store tables in main memory on a set of machines
  - Find all permutated fingerprints whose top $\boldsymbol{p}_i$ bits match the top $\boldsymbol{p}_i$ bits of $\boldsymbol{\pi}_i(\mathbf{F})$
  - For the fingerprints that matched, check if they differ from $\boldsymbol{\pi}_i(\mathbf{F})$ in at most ***k*** bits

SimHash (cont.)

- Very simple example
  - One web-page
    - Text: "*Simhash Technique*"
  - Reduced to two features
    - "Simhash" -> weight = 2
    - "Technique" -> weight = 4
  - Hash features to 4-bits
    - "Simhash" -> 1101
    - "Technique" -> 0110

- Start vector with all 0s

0
0
0
0

SimHash (cont.)

- Apply "Simhash" feature (weight = 2)

	feature's f-bit value	calculation		
0	1	0 + 2	→	2
0	1	0 + 2	→	2
0	0	0 - 2	→	-2
0	1	0 + 2	→	2

- Apply "Technique" feature (weight = 4)

	feature's f-bit value	calculation		
2	0	2 - 4	→	-2
2	1	2 + 4	→	6
-2	1	-2 + 4	→	2
2	0	2 - 4	→	-2

SimHash (cont.)

- Final vector:

-2
6
2
-2

- Sign of vector values is -,+,+,-
- Final 4-bit fingerprint = 0110

SimHash Example

- Simple example
  - F = 0100 1101
  - K = 3
  - Have a collection of 8 fingerprints
  - Create two tables

Fingerprints
1100 0101
1111 1111
0101 1100
0111 1110
1111 1110
0000 0001
1111 0101
1101 0010

SimHash Example (cont.)

Fingerprints
1100 0101
1111 1111
0101 1100
0111 1110
1111 1110
0010 0001
1111 0101
1101 0010

p = 3; π = Swap last four bits with first four bits
0101 1100
1111 1111
1100 0101
1110 0111

p = 3; π = Move last two bits to the front
1011 1111
0100 1000
0111 1101
1011 0100

SimHash Example (cont.)

p = 3; π = Swap last four bits with first four bits
0101 1100
1111 1111
1100 0101
1110 0111

SORT

p = 3; π = Swap last four bits with first four bits
0101 1100
1100 0101
1110 0111
1111 1111

p = 3; π = Move last two bits to the front
1011 1111
0100 1000
0111 1101
1011 0100

SORT

p = 3; π = Move last two bits to the front
0100 1000
0111 1101
1011 0100
1011 1111

SimHash Example (cont.)

F = 0100 1101

$\pi(F)$ = 1101 0100

p = 3; π = Swap last four bits with first four bits
0101 1100
1100 0101
1110 0111
1111 1111

$\pi(F)$ = 0101 0011

p = 3; π = Move last two bits to the front
0100 1000
0111 1101
1011 0100
1011 1111

Match!

p = 3; π = Swap last four bits with first four bits							
1	1	0	1	0	1	0	0
1	1	0	0	0	1	0	1

F

p = 3; π = Move last two bits to the front							
0	1	0	1	0	0	1	1
0	1	0	0	1	0	0	0

MinHash

- Given a universe **U**, pick a permutation $\boldsymbol{\pi}$ on **U** uniformly at random
- Hash each subset $S \subseteq U$ to the minimum value it contains according to $\boldsymbol{\pi}$
- Example: $A = \{1, 2\}$, $B = \{2, 3\}$
  - $\boldsymbol{\pi} = (1 < 2 < 3)$, $h(A) = 1$, $h(B) = 2$
  - $\boldsymbol{\pi} = (1 < 3 < 2)$, $h(A) = 1$, $h(B) = 3$
  - $\boldsymbol{\pi} = (2 < 1 < 3)$, $h(A) = 2$, $h(B) = 2$
  - $\boldsymbol{\pi} = (2 < 3 < 1)$, $h(A) = 2$, $h(B) = 2$
  - $\boldsymbol{\pi} = (3 < 1 < 2)$, $h(A) = 1$, $h(B) = 3$
  - $\boldsymbol{\pi} = (3 < 2 < 1)$, $h(A) = 2$, $h(B) = 3$

Practical Application

- Given a collection of web documents, find near-duplicate pairs
  - Need not be an identical copy of one another
  - Each document is represented by a set of ***k***-grams (shingles)
    - document = **abacacd**, ***k*** = 2
    - shingles = { **ab**, **ba**, **ac**, **ca**, **cd** }

- Two documents are similar if they share many shingles
  - Compute MinHash or SimHash using several hash functions
  - Concatenate the hashes to obtain a "signature"
  - Sort documents by their signatures