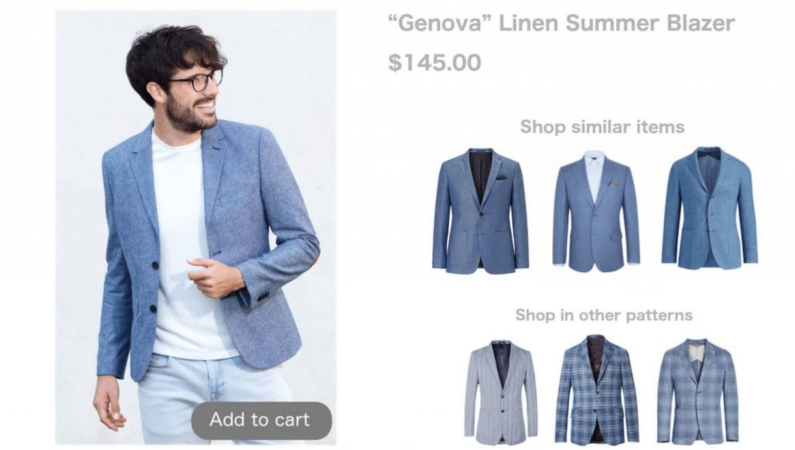
SDSC3002
Finding Similar Items: Min Hashing

Yu Yang
yuyang@cityu.edu.hk

Outline

Item Recommendation



Near-Duplicate Docs Detection

- Online doc sharing system
- Remove near-duplicate docs
- Also used in text summarization, news recommendation, ...

Similarity as Kernel

N = size of training data

$$f(\mathbf{x}) = \sum_{i}^{N} \alpha_i y_i k(\mathbf{x}_i, \mathbf{x}) + b$$

weight (may be zero)

support vector

Gaussian kernel $k(\mathbf{x}, \mathbf{x}') = \exp\left(-||\mathbf{x} - \mathbf{x}'||^2 / 2\sigma^2\right)$

Radial Basis Function (RBF) SVM

$$f(\mathbf{x}) = \sum_{i}^{N} \alpha_i y_i \exp\left(-||\mathbf{x} - \mathbf{x}_i||^2 / 2\sigma^2\right) + b$$

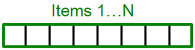
The Major Challenge

- Problem: find all pairs of data points (x_i, x_j) such that $sim(x_i, x_j) \geq \theta$
- Brute force search: $O(n^2)$ time to enumerate all pairs
- Goal: only use $O(n)$ time (compromising some accuracy)
  - Consider $n = 10^7$, $O(n^2) = 10^14$ takes more than a year if 10^6 comparisons per sec

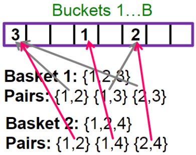
Inspiration by Pattern Mining

- Recall the Hashing and Pruning technique
  - Put multiple patterns in one same bucket
  - Itemsets in infrequent patterns can be pruned

- **Pass 1:**
  - Count exact frequency of each item:



  - Take pairs of items {i,j}, hash them into *B* buckets and count of the number of pairs that hashed to each bucket:
- **Pass 2:**
  - For a pair {i,j} to be a **candidate for a frequent pair**, its singletons {i}, {j} have to be frequent and the pair has to hash to a frequent bucket!



- Hashing helps pruning!

Similarity of Sets

- Given two sets A and B, Jaccard similarity is defined as

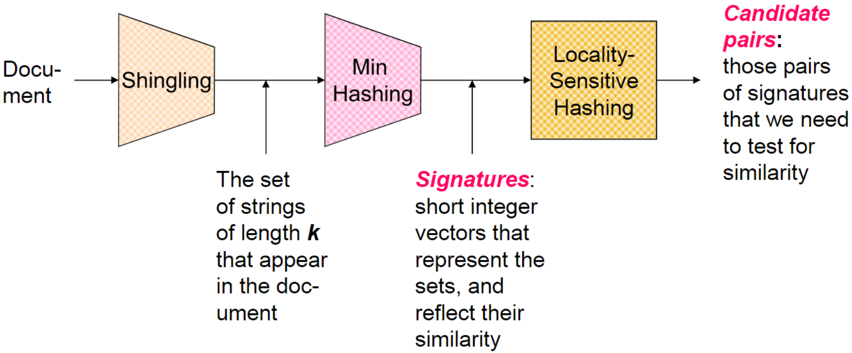
$$JS(A,B) = \frac{|A \cap B|}{|A \cup B|}$$

- $A = \{0,1,2,5,6\}$, $B = \{0,2,3,5,7,9\}$, $A \cap B = \{0,2,5\}$, $A \cup B = \{0,1,2,3,4,5,6,7,9\}$, $JS(A,B) = \frac{3}{9}$
- Properties of Jaccard similarity
  - Within the range $[0,1]$
  - Convert to distance metrics: $1 - JS(A,B)$ or $\sqrt{JS(A,A)^2 + JS(B,B)^2 - 2JS(A,B)}$
  - A valid kernel as the Jaccard Index Matrix is positive semi-definite

Essential Steps for Similar Docs

- **Shingling**: convert docs to sets (feature extraction)
- **Min Hashing**: convert large sets to short signature vectors preserving (approximate) similarity
- **Locality Sensitive Hashing**: only consider pairs of signatures likely to be from similar docs (candidate pairs)

The Big Picture



Outline

Turning Docs to Sets

- Naive ideas
  - Document as set of words
  - Document as set of important words
- Ordering of words matters for Natural Languages

Shingles

- A *k-shingle* (*k-gram*) for a doc is a sequence of of k tokens appearing in the doc
  - Tokens: can be characters, words or even phrases, ...

- **Example: k=2**; document $\mathbf{D_1}$ = abcab
  Set of 2-shingles: $\mathbf{S(D_1)}$ = {ab, bc, ca}
  - **Option:** Shingles as a bag (multiset), count ab twice: $\mathbf{S'(D_1)}$ = {ab, bc, ca, ab}

Compressing Shingles

- We do not have to store the whole shingles
- Hash each shingle to an integer (4 bytes)
- Represent a doc by the set of hash values of its k-shingles
  - Then use Jaccard similarity as the similarity measure

- **Example: k=2**; document $\mathbf{D_1}$= abcab
  Set of 2-shingles: $\mathbf{S(D_1)}$ = {ab, bc, ca}
  Hash the singles: $\mathbf{h(D_1)}$ = {1, 5, 7}

Assumption Behind the Framework

- Docs sharing lots of shingles have similar text, although the text appears in different orders
- Caveat: you must pick k large enough, otherwise most docs will have most shingles
  - $k = 5$ is okay for short docs
  - $k = 10$ is better for long docs
- Such modeling is not perfect
  - The computation does not have to be perfect

Outline

Encoding Sets as Bit Vectors

- Let N be the number of all possible k-shingles
- Convert a set of shingles to an N-dimensional bit vector
  - The i-th bit is 1 if the set contains the i-th k-shingle
- Interpret set intersection as bitwise AND, and set union as bitwise OR
- Example: $C_1 = 10111, C_2 = 10011$

From Sets to Boolean Matrices

- Rows: elements (shingles)
- Columns: sets (docs)
- We usually have a sparse matrix

Element	S_1	S_2	S_3	S_4
1	1	0	0	1
2	1	0	1	0
3	0	1	1	0
4	0	0	1	1
5	1	0	0	0
6	0	0	1	1

represents matrix $M = \begin{pmatrix} 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{pmatrix}$

From Columns to Signatures

- Can we further compress representations of docs?
- Key idea: hash each column C to a small **signature** $h(C)$ such that
  - $h(C)$ is small enough to fit in RAM
  - similarity of columns $\approx$ "similarity" of signatures
- Goal: find a hash function $h(\cdot)$ such that
  - If $JS(C_1, C_2)$ is high, then $h(C_1) = h(C_2)$ with high prob.
  - If $JS(C_1, C_2)$ is low, then $h(C_1) \neq h(C_2)$ with high prob.

Min Hashing

- Imagine the rows of the boolean matrix permuted under a random permutation π
- Define the hash function $m(C)$ as the index of the **first** row (under π) where the column C has value 1
- Compare $m(C_1)$ and $m(C_2)$ to estimate their Jaccard similarity

Example

Step 1: Randomly permute the items (by permuting the rows of the matrix).

Element	S_1	S_2	S_3	S_4
2	1	0	1	0
5	1	0	0	0
6	0	0	1	1
1	1	0	0	1
4	0	0	1	1
3	0	1	1	0

Step 2: Record the first 1 in each column, using a map function m. That is, given a permutation, applied to a set S, the function $m(S)$ records the element from S which appears earliest in this permutation.

$$m(S_1) = 2$$
$$m(S_2) = 3$$
$$m(S_3) = 2$$
$$m(S_4) = 6$$

Step 3: Estimate the Jaccard similarity $\mathsf{JS}(S_i, S_j)$ as

$$\hat{\mathsf{JS}}(S_i, S_j) = \begin{cases} 1 & m(S_i) = m(S_j) \\ 0 & \text{otherwise.} \end{cases}$$

Analysis of Min Hashing

- Three types of rows
  - Tx: there are x rows with 1 in both columns
  - Ty: there are y rows with 1 in one column and 0 in the other
  - Tz: there are z rows with 0 in both column
- Random permutation
  - Start from $\pi = [1, \ldots, n]$
  - Randomly select an index j in $[i, i+1, \ldots, n]$ and swap $\pi[i]$ and $\pi[j]$ for the final $\pi[i]$
  - Rows of 0 can be ignored
- Every element in C becomes the min one with equal prob.
  - $m(S_1) = m(S_2)$ means among $|S_1 \cup S_2|$ rows, some row in $S_1 \cap S_2$ is picked as the min for both S_1 and S_2, prob.$=JS(S_1, S_2)$
- $E[\hat{JS}(S_1, S_2)] = E[\mathbf{I}(m(S_1) = m(S_2))] = JS(S_1, S_2)$

Analysis of #Permutations

- To boost the estimation accuracy, we adopt r independent random permutations
  - The i-th random permutation results in $m_i(C)$ for each column C
  - $\hat{JS}(S_1, S_2) = \frac{1}{r}\sum_{i=1}^{r} \mathbf{I}(m_i(S_1) = m_i(S_2))$
- By **Chernoff bound**

$$\Pr\{|\hat{JS}(S_1, S_2) - JS(S_1, S_2)| \geq \epsilon\} \leq 2e^{-2\epsilon^2 r}$$

- Set $\epsilon = 0.05$, $\delta = 2e^{-2\epsilon^2 r} = 1\% \rightarrow r \approx 1060$

Fast Min Hashing Algorithm

- Generating r random permutations is costly: building the matrix, r permutations, ...
- Fast implementation
  - We pick r random hash functions $\{h_1, h_2, \ldots, h_r\}$ from a Hash family, where $h_i : \mathcal{V} \to [n']$ ($n' \geq n$) and $\mathcal{V}$ is the set of all possible elements
  - Initialize $m_1(S), m_2(S), \ldots, m_r(S)$ as ∞ for each S

Algorithm Min Hash on set S

```
for i ∈ S do
  for j = 1 to r do
    if (h_j(i) < m_j) then
      m_j ← h_j(i)
```

We only scan the data once without explicitly storing the permutations

Random Hash Functions: Universal Hashing

- Choose a large prime $M > |\mathcal{V}|$
- $h_{cd}(a) = ca + d (\text{mod } M)$
- $H = \{h_{cd} \mid c, d = 0, 1, \ldots, M-1\}$
- A random hash function is represented by a 3-tuple (c, d, M)

Acknowledgement

- Some of the contents originate from Jure Leskovec's slides for CS246 at Stanford