Mining of Massive Datasets (Stanford textbook)

Chapter 1

Data Mining = the process of extracting useful models of data. Sometimes, a model can be a summary of the data, or it can be the set of most extreme features of the data.

Models

- statistical modeling
- machine learning
- computational approaches to modeling
    1. summarize data
        - PageRank by Google
        - Clustering
    2. extract the most prominent features
        - Frequent itemsets
        - Similar items (collaborative filtering)

1.2.1 Total Information Awareness

Recall 911 example (before/after the attack)

1.2.2 Bonferroni's Principle

Bonferroni's Principle is an informal presentation of a statistical theorem that states if your method of finding significant items returns significantly more items that you would expect in the actual population, you can assume most of the items you find with it are bogus (虛假). This essentially means that an algorithm or method we think is useful for finding a particular set of data actually returns more FP (false positives) as it returns larger portions of the data than should be within that category.

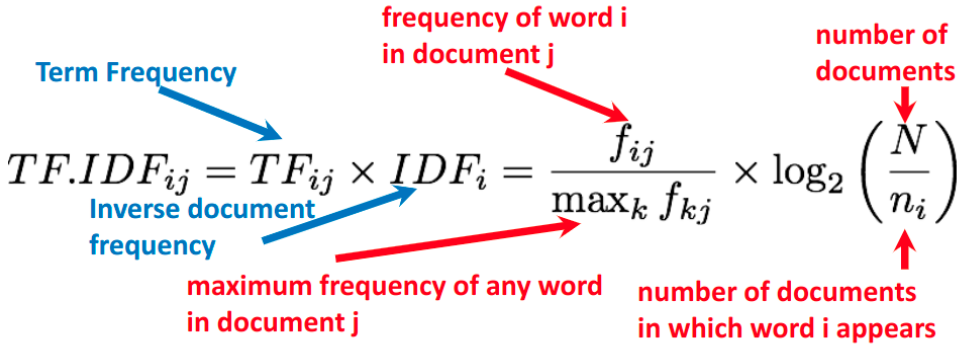
1.3.1 Importance of Words in Documents

Stop words = most common words in English ("the" "a" "and")

Term Frequency times Inverse Document Frequency (TF.IDF)

TF.IDF = measures how concentrated into few documents are the occurrences of a given word.

TF_{ij} is normalized by the *max* number of occurrences of any term (word)



Highest TF.IDF score = the term best characterize the topic of the document

1.3.2 Hash Functions

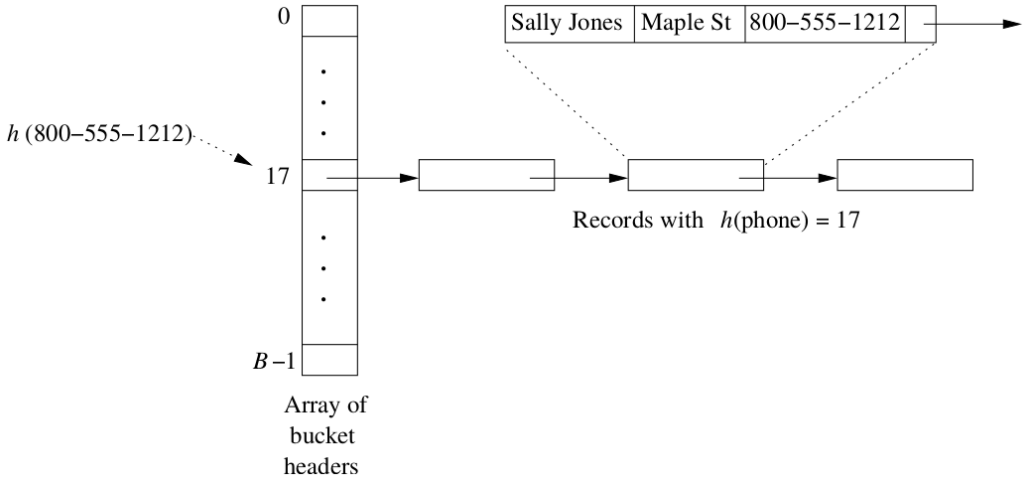
A hash function *h* takes a *hash-key* value as an argument and produces a *bucket number* as a result.

Suppose hash-keys are positive integers, we **usually** pick

$h(x) = x \bmod B$, where B = *prime number*

1.3.3 Indexes

An *index* is a data structure that allows us to store and retrieve data records efficiently, given the value in one or more of the fields of the record. Hashing is one way to build an index.



Here, a hash table is used as an index

1.3.4 Secondary Storage

Recall ECE embedded courses, that accessing data from main memory is way faster than accessing from disk.

1.3.5 The Base (e) of Natural Logarithms

Definition of *e*:

$$e = lim\ (1 + 1/x)^x$$

When *a* is small & *b* is large:

$$(1 + a)^b \approx e^{ab},$$

$$(1 - a)^b \approx e^{-ab}$$

1.3.6 Power Laws

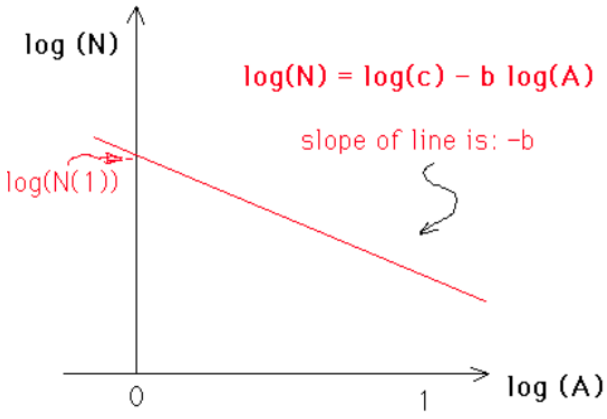
General form:

$$log\ y = b + a\ log\ x$$

let raise the base of the equations (e.g. *e*)

$$y = e^b \cdot e^{a log x}$$

$$y = c \cdot x^a$$

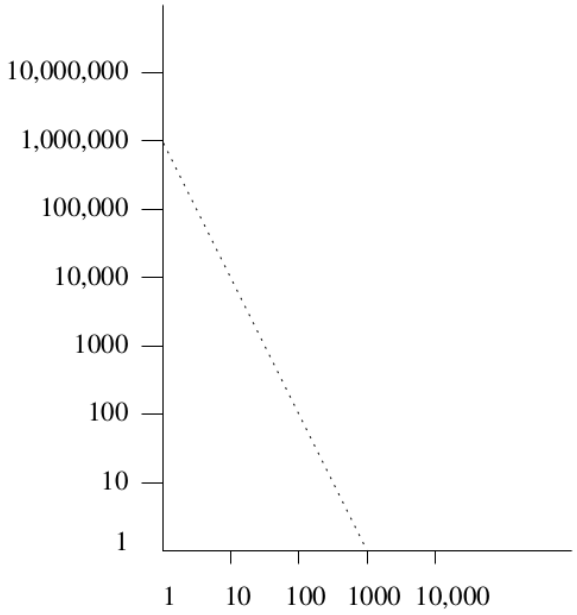


For example, for the pic on the right

$\log_{10} y = 6 - 2 \log_{10} x$

or

$y = 10^6/x^2$



Chapter 3 Finding Similar Items

3.1.1 Jaccard Similarity of Sets

Assume we have sets S and T, then

$$\text{Jaccard similarity } SIM(S, T) = |S \cap T| / |S \cup T|$$

3.1.3 Collaborative Filtering as a Similar-Sets Problem

Jaccard similarity for bags (used in movie ratings)

Example: bag {a,a,a,b}, {a,a,b,b,c} has a Jaccard bag-similarity of 1/3.

$$|S \cap T| = \{a,a,b\} = 3$$

$$|S \cup T| = \{a,a,a,b,a,a,b,b,c\} = 9$$

$$|S \cap T| / |S \cup T| = 1/3$$

Exercise: Suppose we have a universal set U of n elements, and we choose two subsets S and T at random, each with m of the n elements. What is the expected value of the Jaccard similarity of S and T?

Let the number of common elements between S and T be k. Then, as mentioned by ack_inc in the comment to his answer, Jaccard similarity $Sim(S,T) = k/(2m-k)$.

Now, $Pr(Sim(S,T) = k/(2m-k))$ will be $\dfrac{\binom{m}{k}\binom{n-m}{m-k}}{\binom{n}{m}}$ since there are n total elements, of which m are in S and k are common. So the number of ways we can choose m elements for T is given by $\binom{m}{k}$ (choosing the k common elements from S) times $\binom{n-m}{m-k}$ (choosing remaining $m-k$ elements).

Thus, $E(Sim(S,T)) = \sum_{k=0}^{m} \dfrac{k}{2m-k} \dfrac{\binom{m}{k}\binom{n-m}{m-k}}{\binom{n}{m}}$.

3.2 Shingling (瓦, 鹅卵石) of Documents

3.2.1. k-Shingles

document = a string of characters

k-shingle = any substring of length *k* in the document

Example: Suppose our document *D* is the string *abcdabd*, and we pick *k* = 2. Then the set of 2-shingles for *D* is {ab, bc, cd, da, bd}.

Note that the substring *ab* appears twice within *D*, but appears only once as a shingle.

You have to specify whether to eliminate white space (blank, tab, newline…) for exam questions.

3.2.2 Choosing the Shingle Size

"*k* should be picked **large enough** that the probability of any given shingle appearing in any given document is low"

3.2.3 Hashing Shingles

Memory is insufficient → hashing shingles

"Instead of using substrings directly as shingles, we can pick a hash function that maps strings of length *k* to some number of buckets and treat the resulting bucket number as the shingle."

3.3 Similarity-Preserving Summaries of Sets

Problem: space required to store a shingle set is large (even after hashing)

Goal: replace large sets by smaller representation, namely *signatures*.

3.3.1 Matrix Representation of Sets

Example

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

S_1 = {a, d}, S_2 = {c}, S_3 = {b,d,e}, S_4 = {a,c,d}

3.3.2 Minhashing

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 number of the first row, in the permuted order, in which the column has a 1.

Example (continue):

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

We scan until we find a first *1* in each column.

$h(S_1)$ = a, $h(S_2)$ = c, $h(S_3)$ = b, $h(S_4)$ = a

3.3.3 Minhashing and Jaccard Similarity

"The prob. that the minhash function for a random permutation of rows produces the same value for two sets = Jaccard similarity of those sets"

prob. of [$h(S_1) = h(S_2)$] == $SIM(S_1, S_2)$

Incomplete proof (pg. 82-83):

Assume we have two sets

- Type X row = {1,1}
- Type Y row = {1,0} or {0,1}
- Type Z row = {0,0}

Let x = *type X*, y = *type Y*, then $SIM(S_1, S_2) = |S \cap T| / |S \cup T| = x / (x+y)$

The probability that $h(S_1) = h(S_2)$ is also $x / (x+y)$, consider three cases:

first row we meet is (1) type X; (2) type Y; (3) type Z.

The probability of (1) is x/(x+y) and $h(S_1) = h(S_2)$; other cases $h(S_1) \neq h(S_2)$

3.3.5 Computing Minhash Signatures

Problem: random permutation of matrix (as in 3.3.2) is infeasible in real life

Solution: use random hash function to mimic the behavior

Algorithm

0. randomly pick n hash functions $h_1, h_2, \ldots, h_n$

1. set $SIG(i, c) = inf$ for all row i and column c

2. compute $h_1(r), h_2(r), \ldots, h_n(r)$

3. For each column c:

   (a) if c has 0 in row r, do nothing

   (b) if c has 1 in row r, then for each i = 1,2, …, n set $SIG(i,c)$ to $min(SIG(i,c), h_i(r))$

Example:

Step 0, we pick $h_1 = x+1 \bmod 5$, and $h_2 = 3x+1 \bmod 5$

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

	S1	S2	S3	S4
h_1	Inf	Inf	Inf	Inf
h_2	Inf	Inf	Inf	Inf

Row 0

	S1	S2	S3	S4
h_1	1	Inf	Inf	1
h_2	1	Inf	Inf	1

Row 1

	S1	S2	S3	S4
h_1	1	Inf	2	1
h_2	1	Inf	4	1

Row 2

	S1	S2	S3	S4
h_1	1	3	2	1
h_2	1	2	4	1

Row 3

	S1	S2	S3	S4
h_1	1	3	2	1
h_2	0	2	0	0

Row 4 (Final signature matrix)

	S1	S2	S3	S4
h_1	1	3	0	1
h_2	0	2	0	0

Estimated Jaccard similarity (from signature matrix)

SIM(S1, S4) = {(1,1), (0,0)} / {(1,1), (0,0)} = 2/2 = 1

True Jaccard similarity (from original matrix)

SIM(S1, S4) = {(1,1), (0,0)} / {(1,1), (0,1), (0,0)} = 2/3

Conclusion: The fraction of rows that agree in the signature matrix is only an ***estimate*** of the true Jaccard similarity

Creating h(x): Universal Hash Functions

h(x, a, b) = ((ax + b) mod p) mod m

a = [1, p-1]

b = [0, p-1]

p = prime number, p >> m

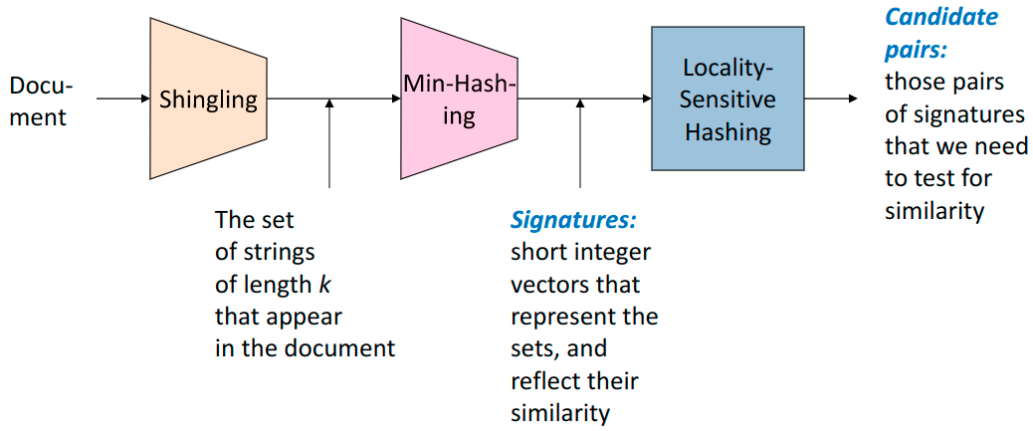
m = int_max + 1

3.4 Locality-Sensitive Hashing for Documents

Problem: cannot *efficiently* find the pairs (of documents) with greatest similarity in real life

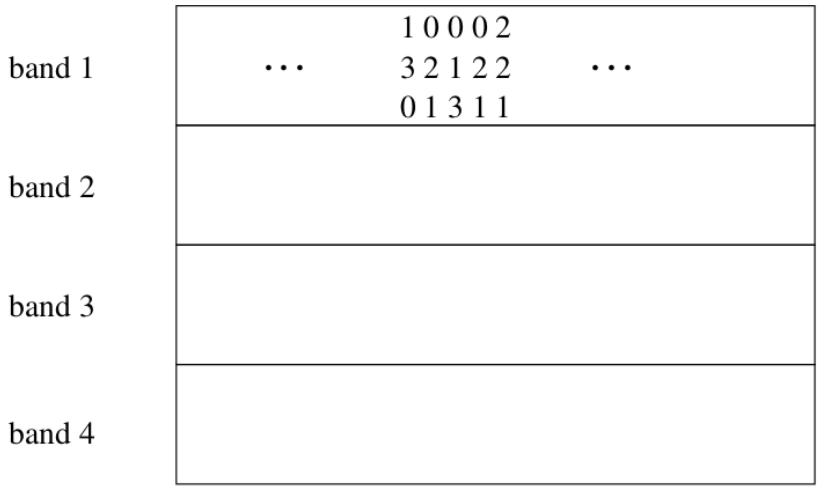
Solution:

- focus on pairs that are above some lower bound in similarity, (w/o investigating every pair)
- namely, *Locality-Sensitive Hashing (LSH)* or *near-neighbor search*



3.4.1 LSH for Minhash Signatures

- Do multiple hashes s.t. similar items are hashed to the same bucket
  - pairs in the same bucket = *candidate pair*
  - dissimilar pairs in the same bucket = *FP (false positives)*
- Only check *candidate pairs*
- divide signature matrix into ***b*** bands; each band consisting of ***r*** rows
- hash vectors of ***r*** integers within a band
- similar columns → likely to be *candidate pairs*
- e.g. in band 1, the 2nd and 4th col [0,2,1] are likely to be in the same bucket than any other pairs



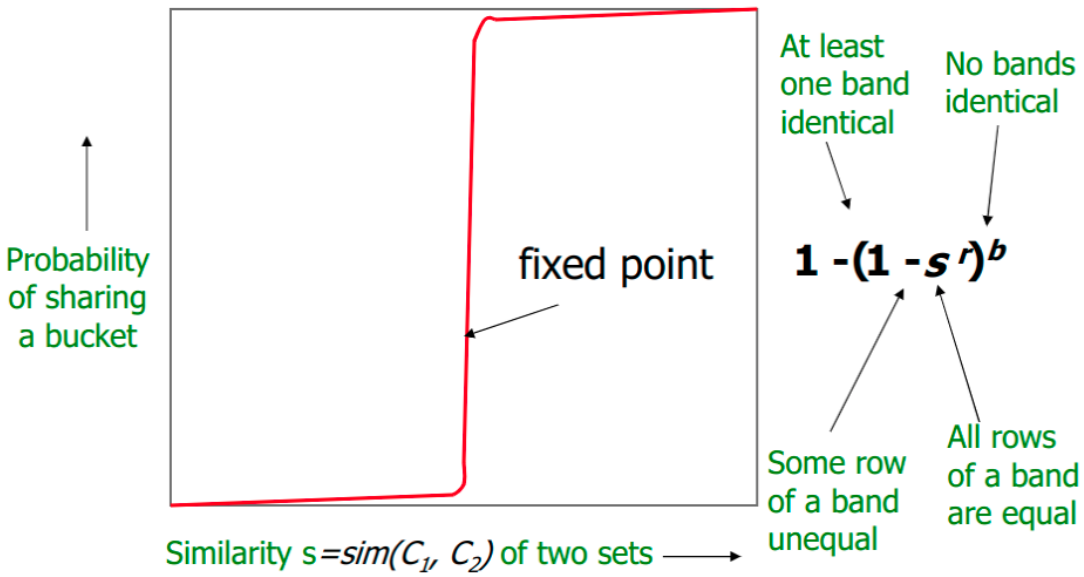
3.4.2 Analysis of the Banding Technique

- Assume we use ***b*** bands of ***r*** rows each
- suppose a particular pair have Jaccard similarity ***s*** → prob. that docs agree in particular row is ***s***
  - ***prob. of [$h(S_1) = h(S_2)$] == $SIM(S_1, S_2)$***

- prob. that signatures *agree* in *all* rows of a particular band
  - $\mathbf{s^r}$
- prob. that signatures *disagree* in *at least* one row of a particular band
  - $\mathbf{1 - s^r}$
- prob. that signatures *disagree* in *at least* one row of each of the bands
  - $\mathbf{(1 - s^r)^b}$
- prob. that signatures agree in *all* rows of *at least* one band
  - $\mathbf{1 - (1 - s^r)^b}$
  - candidate pair!

Threshold = ***s*** at which the prob. of becoming a candidate pair is ½

Approximation of threshold = $\mathbf{(1/b)^{1/r}}$



3.5 Distance Measures

3.5.1 Definition of a Distance Measure

A distance measure on a space (= a set of points) is a function d(x, y) that takes two points in the space as arguments and produces a real number, and satisfies the following **axioms**:

1. **d(x, y) ≥ 0**
2. **d(x, y) = 0 iff x = y**
3. **d(x, y) = d(y, x)**
4. **d(x, y) ≤ d(x, z) + d(z, y)**

Verify in exam (non-negativity, distant to itself, symmetry, triangle inequality)

3.5.2 Euclidean Distances (L_2-norm)

$$d([x_1, x_2, \ldots, x_n],\ [y_1, y_2, \ldots, y_n]) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$$

1. Non-negativity: +ve square root, x ≠ y → strictly positive
2. Distance to itself: if x = y, d = 0
3. Symmetry: $(x-y)^2 = (y-x)^2$
4. Triangle inequality: in Euclidean space, the sum of the lengths of any two sides of a triangle is no less than the length of the third side

L_r-norm

$$d([x_1, x_2, \ldots, x_n],\ [y_1, y_2, \ldots, y_n]) = (\sum_{i=1}^{n} |x_i - y_i|^r)^{1/r}$$

- r = 2 → Euclidean distance
- r = 1 → Manhattan distance
- r = inf → only largest difference matters, max($|x_i - y_i|$)

3.5.3 Jaccard Distance

$$d(x, y) = 1 - SIM(x, y) = 1 - | x \cap y | / | x \cup y |$$

1. Non-negativity: because the size of the intersection cannot exceed the size of the union
2. Distance to itself: $d(x, y) = 0$ *if* $x = y$, because $x \cup x = x \cap x = x$
3. Symmetry: because union and intersection are symmetric; $x \cup y = y \cup x$, $x \cap y = y \cap x$
4. Triangle inequality:
    1. *prob. of* $[h(S_1) = h(S_2)] == SIM(S_1, S_2)$
    2. Jaccard distance = the prob. that h() *does NOT* send x and y to the same value
    3. **d(x, y) ≤ d(x, z) + d(z, y)** == Pr[h(x) ≠ h(y)] ≤ Pr[h(x) ≠ h(z) ∪ h(y) ≠ h(z)]
    4. By the *union bound*, Pr[h(x) ≠ h(z) ∪ h(y) ≠ h(z)] ≤ Pr[h(x) ≠ h(z)] + Pr[h(y) ≠ h(z)]
        - Union Bound (Boole's inequality): for any finite or countable set of events, the probability that at least one of the events happens is no greater than the sum of the probabilities of the individual events.

3.5.4 Cosine Distance

$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}},$$

e.g x = [1,2,-1], y = [2,1,1], distance = 60 degrees

1. Non-negativity: because the value is in [0, 180]
    - theta in [0, pi]
2. Distance to itself: two vectors have angle 0 iff they are the same direction
3. Symmetry: angle between x and y == angle between y and x
4. Triangle inequality: One way to rotate from x to y is to rotate to z and thence to y. The sum of those two rotations cannot be less than the rotation directly from x to y.

3.5.5 Edit Distance

- This distance makes sense when points are strings
- The distance between two strings $x = x_1\, x_2 \cdots x_n$ and $y = y_1\, y_2 \cdots y_m$ is the smallest number of *insertions* and *deletions* of single characters that will convert x to y

e.g x = abcde, y = acfdeg; d(x, y) = 3

1. Non-negativity: no edit distance can be -ve
2. Distance to itself: only identical strings have an edit distance of 0
3. Symmetry: a sequence of insertions and deletions can be reversed
4. Triangle inequality:
   - s → u → t
   - It's impossible for $d(s, u) + d(u, t) \leq d(s, t)$

3.5.6 Hamming Distance

- In vector space, Hamming distance = # of differed components between two vectors

1. Non-negativity: cannot be -ve
2. Distance to itself: identical vectors → dist = 0
3. Symmetry: the distance does not depend on the order of vectors
4. Triangle inequality:
   - if d(x, z) = m and d(y, z) = n, then d(x, y) *cannot differ* more than m+n

3.6 The Theory of Locality-Sensitive Functions

Hash Functions Decide Equality

- A hash function *h* takes two elements x and y, and returns a decision whether x and y are candidates for comparison
  - returns h(x) = h(y) iff x==y

LSH Families Definition

- Suppose we have a space *S* of points with a distance measure *d*
- A family **H** of hash functions is said to be *(d_1, d_2, p_1, p_2)-sensitive* if for any x and y in S
  - d_1 is a small distance, d_2 is large
  - expect that probability p_1 is large, p_2 is small
  1. If d(x, y) ≤ d_1, then prob. over all h in **H**, that h(x) = h(y) is <u>at least</u> p_1
  2. If d(x, y) ≥ d_2, then prob. over all h in **H**, that h(x) = h(y) is <u>at most</u> p_2

Example: LS Family

- Let *S* = sets, *d* = Jaccard distance, **H** is formed from the minhash functions for all permutations
- Then Pr[h(x) = h(y)] = 1 – d(x, y)
  - restates theorem about Jaccard similarity & minhashing in terms of Jaccard distance
- Claim: **H** is a (1/3, 2/3, 2/3, 1/3)-sensitive family for *S* and *d*

LSH Families for Hamming Distance

(d_1, d_2, 1 – d_1/d, 1 – d_2/d)-sensitive

Amplifying a LSH-Family

- The “bands” technique for signature matrices carries over to this more general setting
  - Goal: the “S-curve” effect seen there
- AND construction ~= “rows in a band”
- OR construction ~= “combination of several bands”

AND of Hash Functions

- Given family **H**, construct family **H'** whose members each consist of *r* functions from **H**
- For $h = \{h_1, \ldots, h_r\}$ in **H'**, h(x) = h(y) iff $h_i(x) = h_i(y)$ for <u>all</u> $i = 1 \ldots r$
- **Theorem**: if **H** is (d_1, d_2, p_1, p_2)*-sensitive*, then **H'** is $(d_1, d_2, (p_1)^r, (p_2)^r)$*-sensitive*
  - Proof: h_i' is independent

OR of Hash Functions

- Given family **H**, construct family **H'** whose members each consist of *b* functions from **H**
- For $h = \{h_1, \ldots, h_b\}$ in **H'**, h(x) = h(y) iff $h_i(x) = h_i(y)$ for <u>some</u> *I*
- **Theorem**: if **H** is (d_1, d_2, p_1, p_2)*-sensitive*, then **H'** is $(d_1, d_2, 1-(1-p_1)^b, 1-(1-p_2)^b)$*-sensitive*
  - p = occur → (1-p) = not occur → $(1\text{-}p)^b$ = none occur → $1 - (1\text{-}p)^b$ = at least one occur

Effect of AND and OR Constructions

- AND makes all prob. shrink
  - small r → lower prob. p approaches 0 (while the higher does not)
- OR makes all prob. grow 推导易得
  - small b → higher prob. p approaches 1 (while the lower does not)

- **Goal**: lower prob → 0, higher prob → 1

AND-OR Composition

- p =AND=> p^r =OR=> $1 - (1 - p^r)^b$
- recall the S-curve

OR-AND Composition

- p =OR=> $1 - (1 - p^b)$ =AND=> $1 - (1 - p^b)^r$
- horizontally and vertically mirrored S-curve

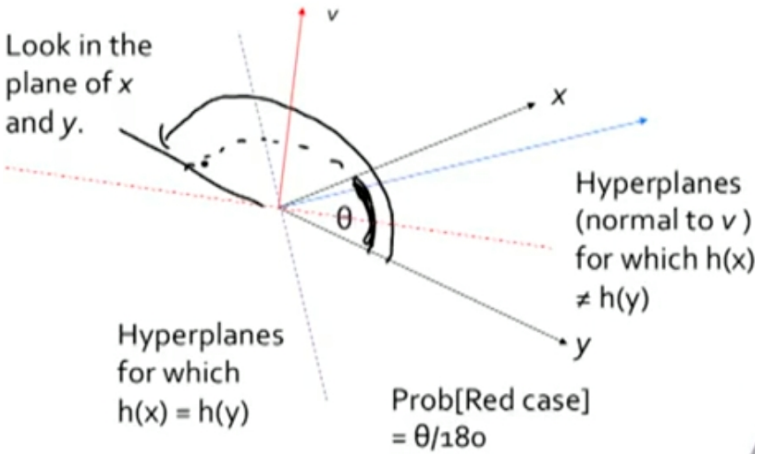
More LSH Families

An LSH Family for Cosine Distance

- **random hyperplanes**: *(d_1, d_2, $(1 - d_1/180)$, $(1 - d_2/180)$)-sensitive* family for any d_1 and d_2

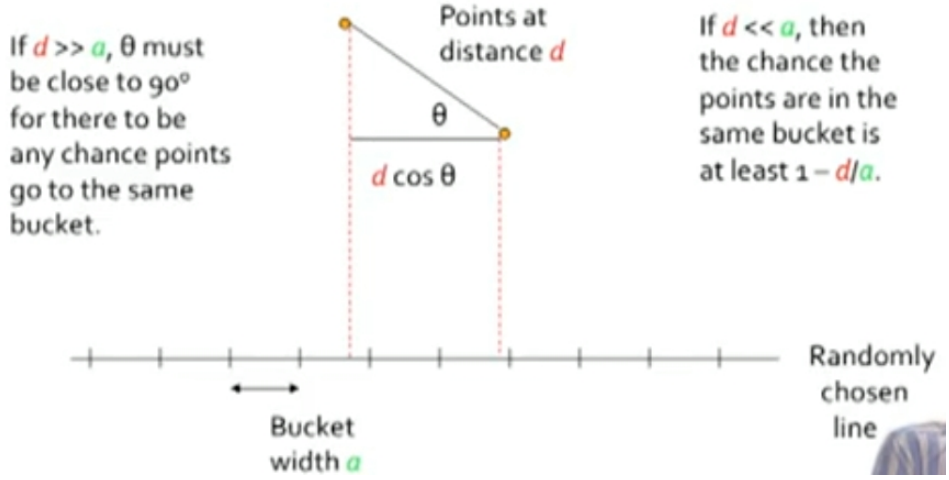
Random Hyperplanes

- Each vector *v* determines a hash function h_v with two buckets
- $h_v(x) = +1$ if dot(v, x) ≥ 0; = -1 if dot(v, x) < 0
- LS-family **H** = set of all functions derived from any vector
- Claim: **Pr[h(x) = h(y)] = 1 – (angle between x and y) / 180**



- for red case, x and y are on different side of the line, so h(x) ≠ h(y)
- for blue case, x and y are on the same side of the line, so h(x) = h(y)

Projection of Points



>> Finding similar sets of items without looking at all pairs of sets

Setting: Sets as Strings

- Represent sets by strings (lists of symbols)
  - order the universal set
  - represent a set by the string of its elements in sorted order

Example: Shingles

- If the universal set is k-shingles, there is a natural lexicographic order
- Think of each shingle as a single symbol
- 2-shingling of *abcad*
  - = set {ab, bc, ca, ad}, which is represented as list [ab, ad, bc, ca]

Jaccard and Edit Distances

- Suppose two sets have Jaccard distance **J** and are represented by strings s_1 and s_2
- Let the LCS of s_1 and s_2 have length **C** and the edit distance of s_1 and s_2 be **E**, then:
- **1 – J** = Jaccard similarity = **C / (C + E)**
- J = E / (C + E)

Length-based Indexes

- Create an index on the length of strings
- A set whose string has length L can be Jaccard distance J from a set whose string has length M only if
  - **L·(1-J) ≤ M ≤ L/(1-J)**
    - M < L case: 1 – J = M/L → M = **L·(1-J)**
    - M > L case: 1 – J = L/M → M = **L/(1-J)**

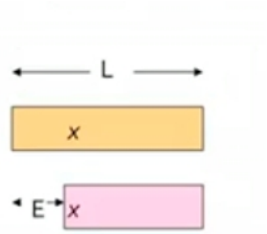
B-Tree Indexes

- B-tree is a perfect index structure for a length-based index
- Given a string of length L, we can find strings with length in the range [L(1-J), L/(1-j)]

>> If Jaccard distance is low, then two strings representing sets of the Jaccard distance must have a symbol in common among the prefixes, whose length are approximately the Jaccard distance times the length of the shorter string.

Example: Prefix-based Indexing

- If two strings are 90% similar, they must share some symbol in their prefixes whose length is just above 10% of the length of each string, then
- we can base an index on symbols in the first **floor(JL + 1)** positions of a string of length L
  - a.k.a *prefix* of the string
  - proof: J ≥ E/L, where E=prefix=differed part, colored bar are equal part



Indexing Prefixes

- Think of a bucket for each possible symbol
- Each string of length **L** is placed in the bucket for **each** of its first **floor(JL + 1)** positions
- A B-tree with symbol as key leads to the strings

Lookup

Given a probe string s of length L, with J=limit on Jaccard dist:

```
for (each symbol 'x' among the floor(JL+1) positions of 's'):
   look for other strings in the bucket for 'x'
```

Example: Indexing Prefixes

- Let J = 0.2
- *abcdef* is indexed unader *a* and *b*
  - L = 6, floor(JL+1) = 2 → a, b
- *acdfg* is indexed under *a* and *c*
- *bcde* is indexed under *b*

Chap 5 # Video Lecture 1 (Intro to Link Analysis & PageRank)

High Dim. Data	Graph Data	Infinite Data	Machine Learning	Apps
Locality sensitive hashing	PageRank, SimRank	Filtering data streams	SVM	Recommender systems
Clustering	Community Detection	Web advertising	Decision Trees	Association Rules
Dimensionality reduction	Spam Detection	Queries on streams	Perceptron, kNN	Duplicate document detection

Graph data

- Social networks
  - e.g. Facebook social network
- Media Networks
  - e.g. connections between political blogs
- Information Networks
  - e.g. citation networks and maps of science
- Communication Networks
  - e.g. Internet
- Technological Networks
  - e.g. bridges problems (only pass once)

Web as a **directed** Graph

- Nodes: webpages, Edges: Hyperlinks

Broad Question: How to organize the Web?

- First try: human curated (策划) web directories (such as Education, Politics…)
- Second try: Web Search (today)
    - **Information Retrieval** investigates:
        - find relevant docs in a small and trusted set
        - problem:
            - web is huge
            - full of untrusted documents, random things, web spam…

Web Search: 2 Challenges

2 challenges of web search:

- web contains many sources of information
    - soln: trustworthy pages may points to each other
- what is the “best” answer to query “newspaper”?
    - No single right answer
    - soln: pages that actually know about newspapers might all be pointing to many newspapers

Ranking Nodes on the Graph

- All web pages are not equally “important”
- There is a large diversity in the web-graph node connectivity

Link Analysis Algorithms

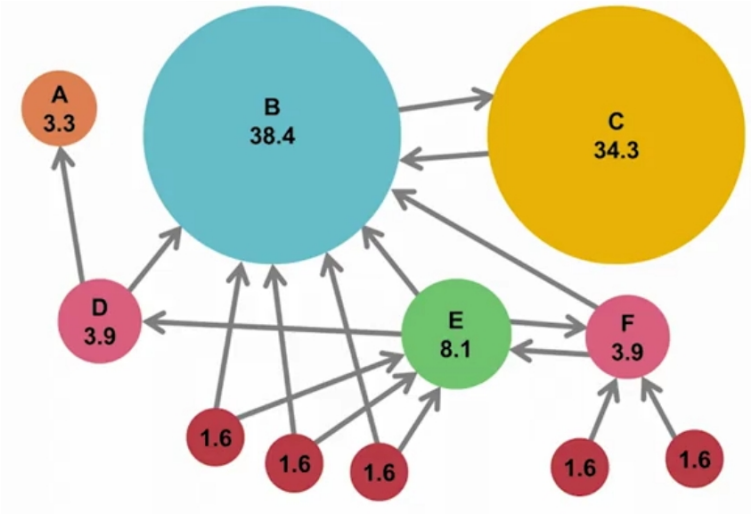
- Page Rank
    - initial algorithm behing Google search
- Hubs and Authorities (HITS)
- Topic-Specific (Personalized) Page Rank
- Web Spam Detection Algorithms

Video Lecture 2 (PageRank – The Flow Formulation)

Links as Votes

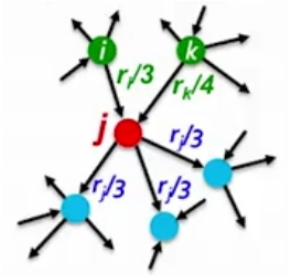
- Idea: links as votes
  - links +, importance +
  - in-links is harder to fake than out-links
  - Not all in-links are equal
  - In-links coming from important pages worth more

Example: PageRank Scores



- Sum = 100
- B have high score because there are a lot of other page points to it
- C have high score because a important node (B) pointing to it

Simple Recursive Formulation (to calculate PageRank score)



- Each link's vote is proportional to the importance of its source page
- If page ***j*** w/ importance r_j has n out-links, each link gets r_j / ***n*** votes
- Page ***j***'s own importance = sum of the votes on its in-links
- e.g. $r_j = r_i/3 + r_k/4$

PageRank: The "Flow" Model

- Define a rank r_j for page j
- r_j = sum of the rank of pages pointing to it, divide the out-degree of the pages pointing to it

$$r_j = \sum_{i \to j} \frac{r_i}{d_i}$$

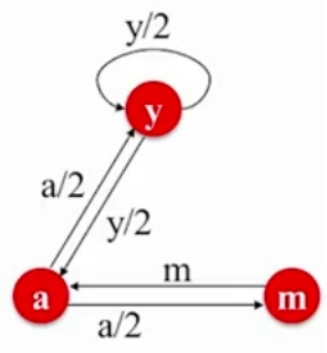
d_i … out-degree of node i

Example

$r_m = r_a / 2$ (only ***a*** points to ***m***, and ***a*** has two outgoing link)

$r_y = r_a/2 + r_y/2$

$r_a = r_y/2 + r_m$



Solving the Flow equations

$r_m = r_a / 2$

$r_y = r_a/2 + r_y/2$

$r_a = r_y/2 + r_m$

$r_a + r_y + r_m = 1$ (additional constraint forces uniqueness)

Limitations of the Flow model

Gaussian elimination method works for small examples, but we need a better method for a large web-size graphs.

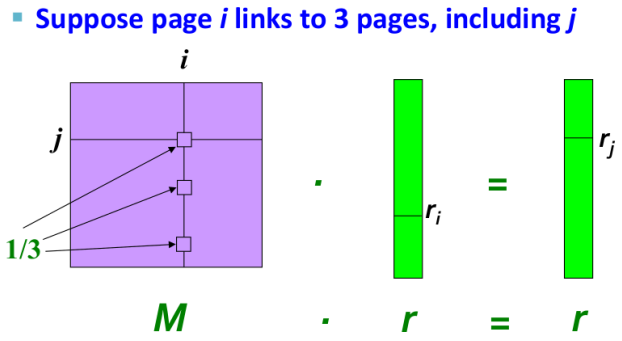
Video Lecture 3 (PageRank: Matrix Formulation)

Stochastic adjacency matrix M

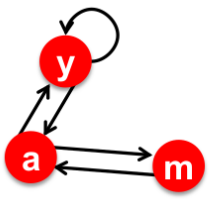
- Let page ***i*** has d_i out-links
- If $i \rightarrow j$, then $M_{ji} = 1 / d_i$; else $M_{ji} = 0$
  - columns sum to 1
- Rank vector ***r***: vector with an entry per page
- r_i = importance score of page ***i***
- $\sum_i r_i = 1$
- The flow equations can be written

$$r_j = \sum_{i \rightarrow j} \frac{r_i}{d_i}$$

  - $r = M \cdot r$
  - recall eigen-equiation: Ax = ƛx
  - rank vector ***r*** is an eigenvector of the stochastic web matrix ***M***
  - largest eigenvalue of ***M*** is 1
    - because (1) ***r*** is unit length (2) each column of ***M*** sums to 1 ==> $Mr \leq 1$
- Use **Power iteration** to solve for ***r***



Example



	y	a	m
y	½	½	0
a	½	0	1
m	0	½	0

$r = M \cdot r$

$$r_y = r_y/2 + r_a/2$$
$$r_a = r_y/2 + r_m$$
$$r_m = r_a/2$$

$$\begin{bmatrix} y \\ a \\ m \end{bmatrix} = \begin{bmatrix} ½ & ½ & 0 \\ ½ & 0 & 1 \\ 0 & ½ & 0 \end{bmatrix} \begin{bmatrix} y \\ a \\ m \end{bmatrix}$$

Video Lecture 4: Power Iteration

- suppose there are ***N*** web pages
- Initialize $\mathbf{r}^{(0)} = [1/N, \ldots, 1/N]^T$
  - $r^{(0)} = r$ at timestamp 0
- Iterate $\mathbf{r}^{(t+1)} = \mathbf{M} \cdot \mathbf{r}^{(t)}$
- stop when $| \mathbf{r}^{(t+1)} - \mathbf{r}^{(t)} |_1 < \varepsilon$
  - L1 norm (can use other vector norm as well)

$$r_j^{(t+1)} = \sum_{i \to j} \frac{r_i^{(t)}}{\mathrm{d_i}}$$

$\mathrm{d_i}$. out-degree of node i

python code

```
M = …
r = np.transpose(np.full((1, M.shape[0]), 1.0/M.shape[0]))
e = 10e-9
d = 10e5
while (d > e)
    new_r = np.dot(M, r)
    d = np.linalg.norm(new_r – r)
    r = new_r
```

Random Walk Interpretation

Imagine a random web surfer:

- At any time ***t***, surfer is on some page ***i***
- At time ***t+1***, the surfer follows an out-link from ***i*** uniformly at random
- Ends up on some page j linked from ***i***
- Process repeats indefinitely

Let

- ***p(t)*** = vector whose i^{th} coordinate is the prob. that the surfer is at page ***i*** at time ***t***
  - ***p(t)*** = prob. dist. over pages
- ***p(t+1) = M·p(t)***

Suppose the random walk reaches a state ***p(t+1) = M·p(t) = p(t)***, then

- ***p(t)*** is **stationary distribution** of a random walk

Recall that ***r = M·r***, where ***r*** is a stationary distribution for the random walk

A central result from the theory of random walks (a.k.a Markov process):

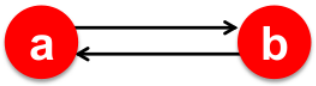
- For graphs that satisfy **certain conditions**, the stationary distribution is **unique** and eventually will be reached no matter what the initial probability distribution at time t = 0

 # Video Lecture 5 (PageRank: The Google Formulation)

$$r_j^{(t+1)} = \sum_{i \to j} \frac{r_i^{(t)}}{\mathrm{d_i}} \quad \text{or equivalently} \quad r = Mr$$

Does this converge?

The "Spider trap" problem



Example:

$$\begin{matrix} r_a \\ r_b \end{matrix} = \begin{matrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{matrix}$$

Iteration 0, 1, 2,

The "Dead end" problem

Example:

$$\begin{matrix} r_a \\ r_b \end{matrix} = \begin{matrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{matrix}$$

Iteration 0, 1, 2,

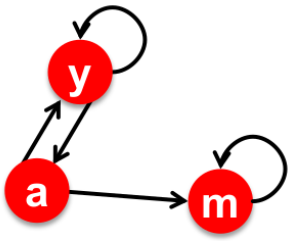
PageRank: Problems

- all out-links are within the group (spider traps)
  - eventually spider traps absorb all importance
- page has no out-links (dead ends); i.e out-degree = 0

- such pages cause importance to leak out

Spider Traps

$$\begin{bmatrix} r_y \\ r_a \\ r_m \end{bmatrix} = \begin{matrix} 1/3 & 2/6 & 3/12 & 5/24 & & 0 \\ 1/3 & 1/6 & 2/12 & 3/24 & \dots & 0 \\ 1/3 & 3/6 & 7/12 & 16/24 & & 1 \end{matrix}$$

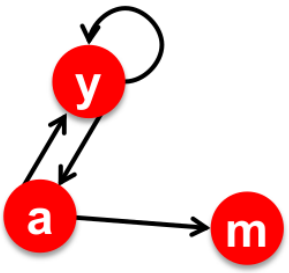


Random Teleports (solution to spider traps)

- At each time step, the random surfer has two options
  - with prob ***b***, follow a link at random
  - with prob ***1-b***, teleports/jumps 瞬移 to some random page
  - b usually = [0.8, 0.9]

Dead Ends

$$\begin{bmatrix} r_y \\ r_a \\ r_m \end{bmatrix} = \begin{matrix} 1/3 & 2/6 & 3/12 & 5/24 & & 0 \\ 1/3 & 1/6 & 2/12 & 3/24 & \dots & 0 \\ 1/3 & 1/6 & 1/12 & 2/24 & & 0 \end{matrix}$$



Always Teleports (solution to dead ends)

- When reach a dead-end, follow random teleport links with prob. = 100%

Video Lecture 6 (PageRank: Why Teleports Solve the Problem)

Markov chains

- is a set of states $\boldsymbol{X}$
- Transition matrix $\boldsymbol{P}$ where $\boldsymbol{P_{ij} = P(X_t = i \mid X_{t-1} = j)}$
- $\boldsymbol{\pi}$ specifying the stationary prob. of being at each state $\boldsymbol{x \in X}$
- Goal is to find $\boldsymbol{\pi}$ such that $\boldsymbol{\pi = P \cdot \pi}$
  - analogy; recall that $\boldsymbol{r = M \cdot r}$

Theory of Markov chains

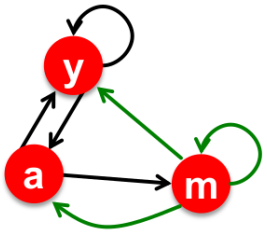
- For any start vector, the power iteration method applied to a Markov transition matrix $\boldsymbol{P}$ will converge to a unique positive stationary vector as long as $\boldsymbol{P}$ is **stochastic**, **irreducible** and **aperiodic**.

Why do teleport makes M stochastic

- M is stochastic if every column sums to 1
- solution: add **green** links

$$A = M + a^T\left(\frac{1}{n}e\right)$$

- a_i =1 if node *i* has out deg 0, =0 else
- **e** vector of all 1s

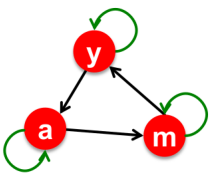


	y	a	m
y	½	½	1/3
a	½	0	1/3
m	0	½	1/3

$$r_y = r_y/2 + r_a/2 + r_m/3$$
$$r_a = r_y/2 + r_m/3$$
$$r_m = r_a/2 + r_m/3$$

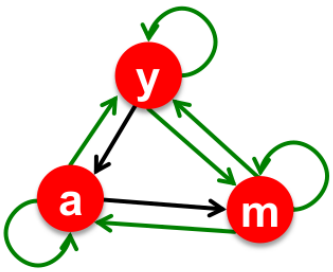
Why do teleport makes M aperiodic

- A chain is **periodic** if there exists $\boldsymbol{k > 1}$ such that the interval between two visits to some state $\boldsymbol{s}$ is always a multiple of $\boldsymbol{k}$
- solution: add green links (to jump out inf loop)



Why do teleport makes M irreducible?

- From any state, there is a non-zero probability of going from any state to another
- solution: add green links



Random Jumps (Google's solution)

- makes M stochastic, aperiodic, irreducible

PageRank equation [Brin-Page, 98]

$$r_j = \sum_{i \to j} \beta \frac{r_i}{d_i} + (1-\beta)\frac{1}{n}$$

d_i ... out-degree of node i

The above formulation assumes that $\boldsymbol{M}$ has no dead ends. We can either preprocess matrix $\boldsymbol{M}$ **(bad!)** or explicitly follow random teleport links with probability 1.0 from dead-ends.

The Google Matrix *A*:

$$A = \beta\, M + (1-\beta)\frac{1}{n}\boldsymbol{e} \cdot \boldsymbol{e}^T$$

e vector of all 1s

***A* is stochastic, aperiodic and irreducible, so**

$$\boldsymbol{r}^{(t+1)} = \boldsymbol{A} \cdot \boldsymbol{r}^{(t)}$$

What is β?

- In practice $\beta = 0.8, 0.9$ (make 5 steps and jump)

Video Lecture 7 (PageRank: How we really compute PageRank?)

Computing Page Rank

- key step: $\boldsymbol{r^{new} = A \cdot r^{old}}$
  - this is easy if we have enough main memory to hold $\boldsymbol{A}$, $\boldsymbol{r^{new}}$, $\boldsymbol{r^{old}}$
  - problem: we cannot store all the matrices in memory
  - solution: rearrange the equations (see below)

Matrix Formulation

- Suppose there are $\boldsymbol{N}$ pages
- Consider page $\boldsymbol{j}$, with $\boldsymbol{d_j}$ out-links
- $\boldsymbol{M_{ij}} = 1 / |\boldsymbol{d_j}|$ when j → i (j points to i); otherwise, $\boldsymbol{M_{ij}} = 0$
- The **random teleport** is equivalent to
  - Adding a teleport link from $\boldsymbol{j}$ to every other page with transition prob = $\boldsymbol{(1-\beta) / N}$
  - this reduces the prob. of each out-link from $\boldsymbol{1 / |d_j|}$ to $\boldsymbol{\beta / |d_j|}$
    - notes that $\boldsymbol{\beta}$ = [0.8, 0.9]
  - Equivalent: tax $\boldsymbol{(1-\beta)}$ of each page's score and redistribute it evenly

Rearranging the Equation

- assume M has no dead-ends; $[X]_N$ = a vector of length N with all entries as X

$$\boldsymbol{r} = \boldsymbol{A} \cdot \boldsymbol{r}, \text{ where } \boldsymbol{A_{ij}} = \boldsymbol{\beta}\, \boldsymbol{M_{ij}} + \frac{\boldsymbol{1-\beta}}{\boldsymbol{N}}$$

$$r_i = \sum_{j=1}^{N} A_{ij} \cdot r_j$$

$$r_i = \sum_{j=1}^{N} \left[\beta\, M_{ij} + \frac{1-\beta}{N}\right] \cdot r_j$$

$$= \sum_{j=1}^{N} \beta\, M_{ij} \cdot r_j + \sum_{j=1}^{N} \frac{1-\beta}{N} r_j$$

$$= \sum_{j=1}^{N} \beta\, M_{ij} \cdot r_j + \frac{1-\beta}{N} \quad \text{since } \sum r_j = 1$$

$$\textbf{So we get: } \boldsymbol{r} = \boldsymbol{\beta}\, \boldsymbol{M} \cdot \boldsymbol{r} + \left[\frac{\boldsymbol{1-\beta}}{\boldsymbol{N}}\right]_{\boldsymbol{N}}$$

- sum of r_j = 1 because r_j is prob. distribution
- after rearranging, M becomes sparse (only hundreds non-zero links out of billions entries)

Note if *M* contains dead-ends then $\sum_i r_i^{new} < 1$ and we also have to renormalize r^{new} so that it sums to 1

PageRank: The Complete Algorithm

- **Input: Graph G and parameter β**
  - Directed graph G with **spider traps** and **dead ends**
  - Parameter β
- **Output: PageRank vector r**
  - **Set:** $r_j^{(0)} = \frac{1}{N}, \quad t = 1$
  - **do:**
    - $\forall j$: $r'^{(t)}_j = \sum_{i \to j} \beta \frac{r_i^{(t-1)}}{d_i}$

      $r'^{(t)}_j = 0$ if in-deg. of j is 0
    - **Now re-insert the leaked PageRank:**

      $\forall j$: $r_j^{(t)} = r'^{(t)}_j + \frac{1-S}{N}$ where: $S = \sum_j r'^{(t)}_j$
    - $t = t + 1$
  - **while** $\sum_j \left| r_j^{(t)} - r_j^{(t-1)} \right| > \varepsilon$

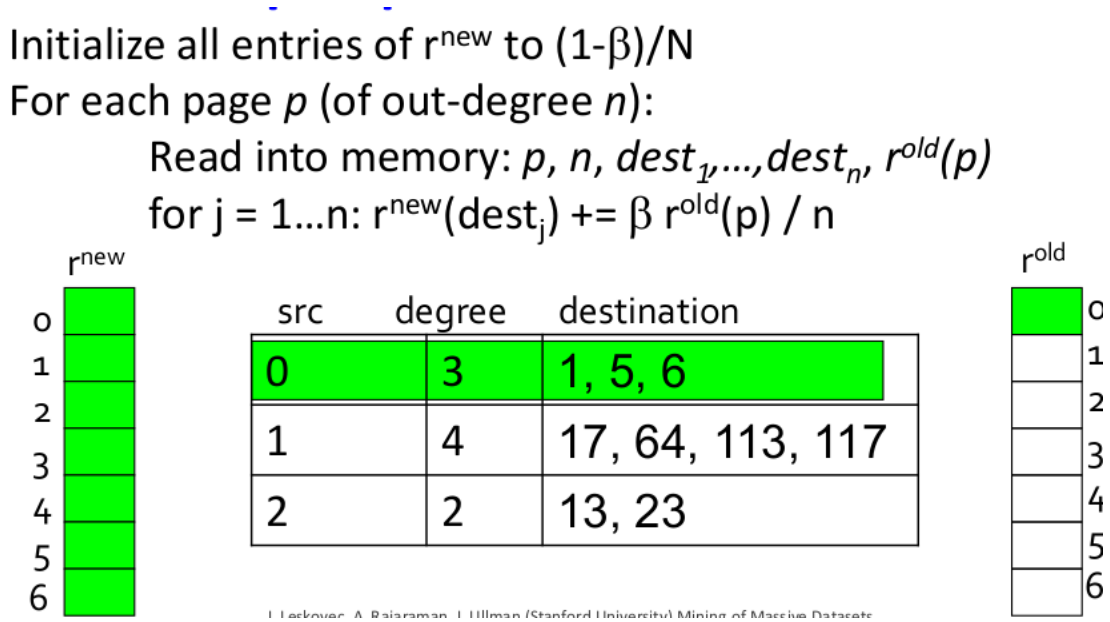
Video Lecture 8 Computing PageRank on Big Graphs

Sparse Matrix Encoding

- Encode sparse matrix using only nonzero entries
  - space proportional roughly to number of links
  - still won't fit in memory, but will fit on disk

Basic Algorithm: Update Step

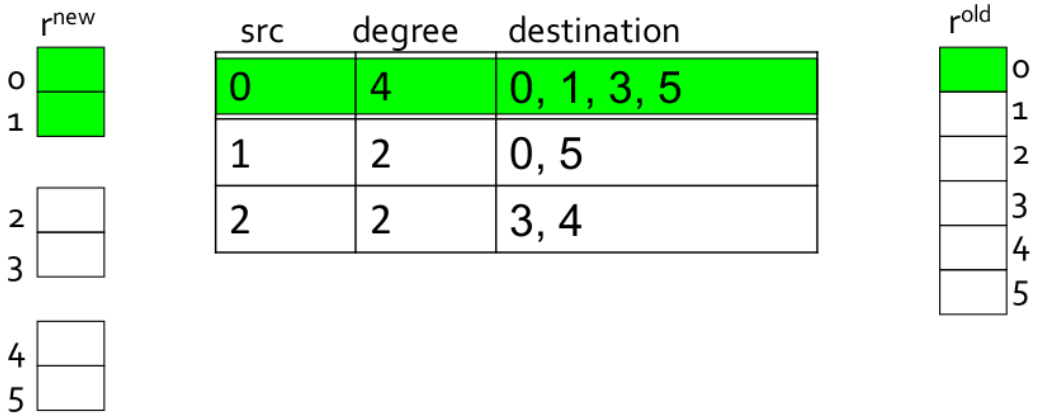
- Assume enough RAM to fit r^{new} into memory; store r^{old} and M on disk
- Then 1 step of power-iteration:



Analysis of Step-Update

- In each iteration, we have to
  - read $\boldsymbol{r^{old}}$ and $\boldsymbol{M}$
  - write $\boldsymbol{r^{new}}$ back to disk
  - Input/Output cost = **2| r | + | M |**
- problem: What if r^{new} doesn't fit in memory?
- solution: breaks r into blocks

Block-based Update Algorithm



Before: assume all of $\mathbf{r}^{new}$ fits in memory

Now: assume part of $\mathbf{r}^{new}$ fits in memory

Steps:

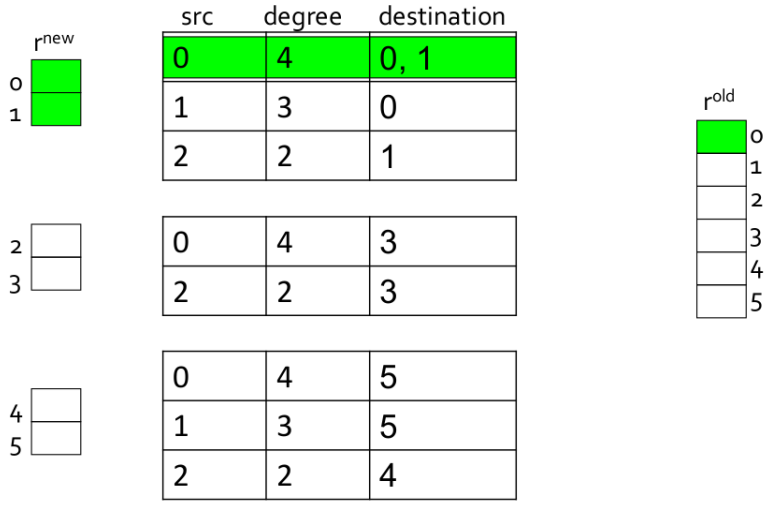
1. take vector ***r*** and split it into blocks
2. load part of $\boldsymbol{r}^{new}$ into memory
3. scan through ***M***
4. scan through rank vector $\boldsymbol{r}^{old}$
5. compute the corresponding score of given $\boldsymbol{r}^{new}$
6. repeat step 2-5 for remaining blocks

Analysis of Block-Update

- break $\boldsymbol{r}^{new}$ into ***k*** blocks that fit in memory
- scan ***M*** and $\boldsymbol{r}^{old}$ once for each block
- ***k*** scans of ***M*** and $\boldsymbol{r}^{old}$
  - $\boldsymbol{k(|M|+|r|)+|r|=k|M|+(k+1)|r|}$
    - the red $|\boldsymbol{r}|$ is the cost of write $\boldsymbol{r}^{new}$ back to memory
- problem: M.size >> r.size, so we have to read M multiple times
- solution: pre-process M

Block-Stripe Update Algorithm

- for each ***r*** block, only scan the corresponding ***M*** chunk



Analysis of Block-Stripe

- breaks ***M*** into chunks which corresponds to r^{new} blocks
- some additional (ε) overhead per stripe, but usually worth it
- Input/Output cost per iteration
  - $|M|(1+\varepsilon) + (k+1)|r|$
  - only needs to read M once, plus ε additional cost

Summary: some problems with PageRank

- PageRank measures generic popularity of a page
  - Biased against topic-specific authorities
  - solution: topic-specific PageRank
- PageRank uses a single measure of importance
  - Other models (e.g. hubs-and-authorities)
  - solution: Hubs-and-Authorities
- PageRank is susceptible to Link spam
  - Artificial link topographies created in order to boost page rank
  - solution: TrustRank

Video Lecture 9 Topic-Specific (Personalized) PageRank

Intuition: instead of generic popularity, can we measure popularity within a topic (sports, history…) ?

- Random walker has a small prob. of teleporting at any step
- Teleport can go to
  - **Standard PageRank (**any page with equal probability)
    - to avoid dead-end and spider-trap problems
  - **Topic-Specific PageRank**
    - a topic-specific set of "relevant" pages (teleport set)
- Idea: bias the random walk
  - when walker teleports, pick a page from set ***S***, which contains only topic-relevant pages
  - For each teleport set ***S***, we get a different vector r_s

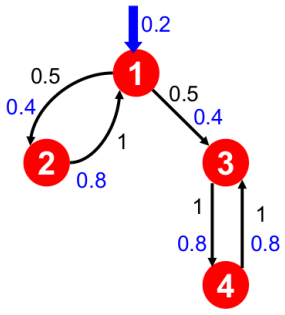
Matrix Formulation

$$A_{ij} = \begin{cases} \beta M_{ij} + (1-\beta)/|S| & \text{if } i \in S \\ \beta M_{ij} & \text{otherwise} \end{cases}$$

A is stochastic (every column sums to 1)

Example

- smaller S → higher score
- smaller β → higher score



Suppose $S = \{1\}$, $\beta = 0.8$

Node	Iteration			
	0	1	2	stable
1	0.25	0.4	0.28	0.294
2	0.25	0.1	0.16	0.118
3	0.25	0.3	0.32	0.327
4	0.25	0.2	0.24	0.261

S={1}, β=0.90:
r=[0.17, 0.07, 0.40, 0.36]
S={1} , β=0.8:
r=[0.29, 0.11, 0.32, 0.26]
S={1}, β=0.70:
r=[0.39, 0.14, 0.27, 0.19]

S={1,2,3,4}, β=0.8:
r=[0.13, 0.10, 0.39, 0.36]
S={1,2,3} , β=0.8:
r=[0.17, 0.13, 0.38, 0.30]
S={1,2} , β=0.8:
r=[0.26, 0.20, 0.29, 0.23]
S={1} , β=0.8:
r=[0.29, 0.11, 0.32, 0.26]

Discovering the Topic Vector S

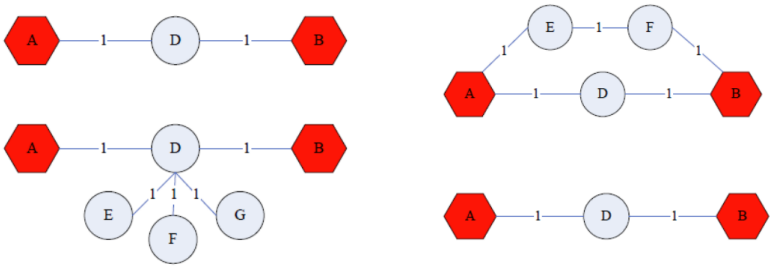
- Create different PageRanks for different topics
  - The 16 DMOZ top-level categories (arts, business, sports …)
- Which topic ranking to use?
  - User can pick from a menu
  - Classify query into a topic
  - Use the context of the query
    - e.g. query is launched from a web page talking about a known topic
    - history of queries (e.g. “basketball” followed by “Jordan”)
  - User context (e.g. user’s bookmarks…)

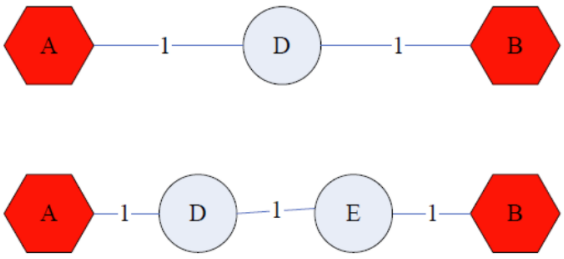
Video Lecture 10 Application to Measuring Proximity in Graphs

Proximity (Relevance, Closeness, “similarity”) on Graphs

How to measure proximity?

- Shortest path (bad)
  - no effect of degree-1 nodes
  - multi-faceted relationships
- Network flow (bad)
  - does not punish long paths
    - in the example, both are able to push 1 unit in the flow
- SimRank





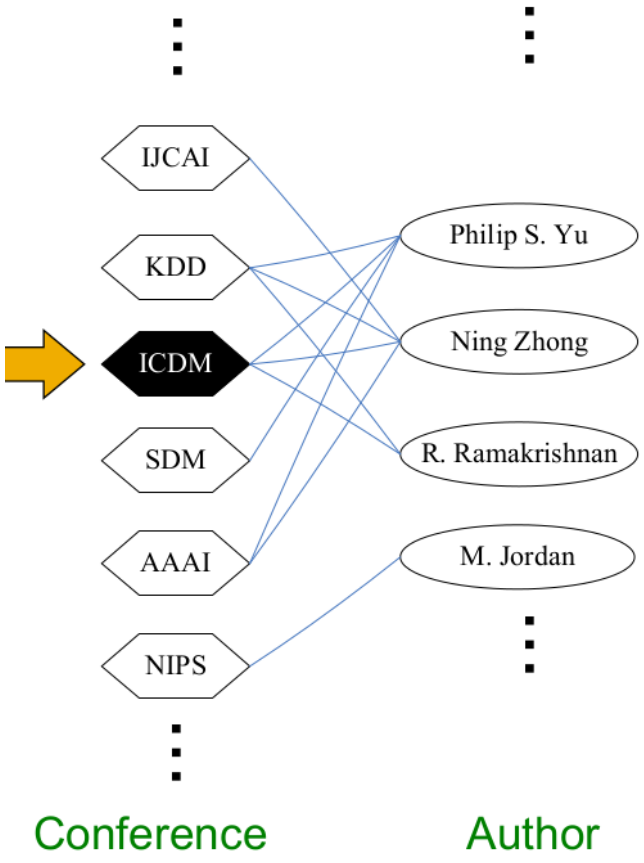
SimRank

- random walks from a ***fixed node*** on ***k***-partite graphs
- Setting: ***k***-partite graph with ***k*** types of nodes
  - Example: picture nodes and tag nodes
- Do a **Random Walk with Restarts** from node ***u***
  - this is a way to measure the similarity of pair images
  - teleport set ***S = {u}***
- Resulting scores measures similarity to node ***u***
- Problem
  - must be done once for each node ***u***
  - suitable for sub-Web-scale applications

SimRank: Example

What is the most related conference to ICDM?

1. create the bi-partite graph of “Conference” and “Author”
2. perform a random walk with a restart form “ICDM”
3. measure the visiting prob. of other walkers for other nodes in the “Conference” parition
    - Two “Conference” is close if they share a lot of “Author” in common

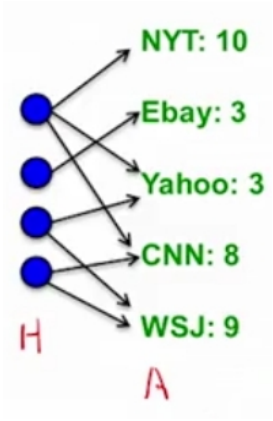


Video Lecture 11 Networks Links Analysis (Hubs and Authorities)

- HITS (Hypertext-Induced Topic Selection)
  - is a measure of importance of pages or documents, similar to PageRank
- Goal: find good newspapers
- Idea: links as votes
  - Page is more important if it has more links

Hubs and Authorities

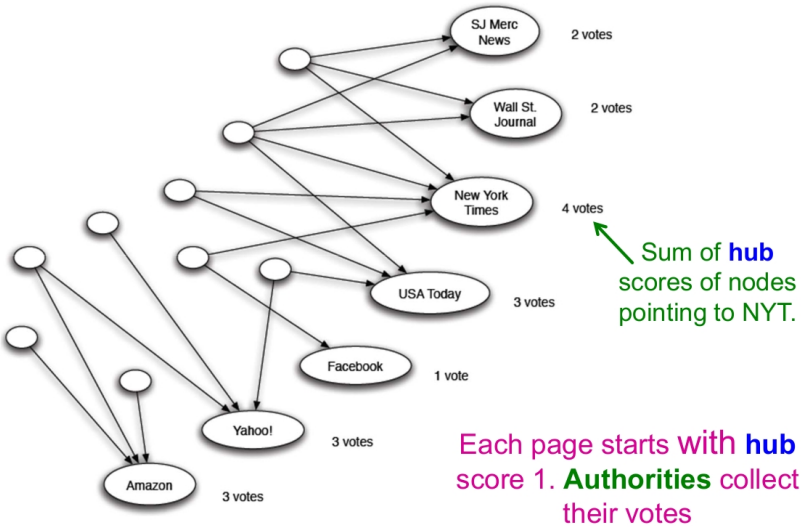
- Each page has two scores
  - Quality as an expert (hub score)
    - sum of votes of authorities pointed to
  - Quality as a content (authority score)
    - sum of votes coming from experts
- Principle of repeated improvement



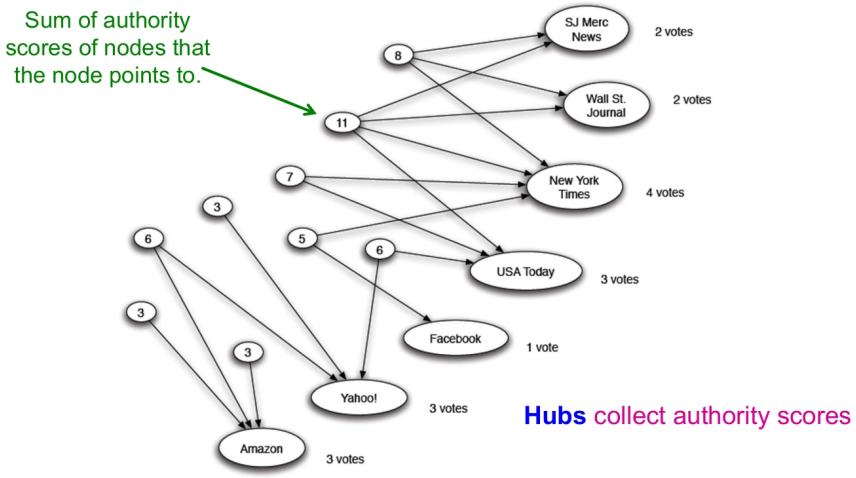
Interesting pages fall into two classes

- **Authorities** are pages containing useful information
  - newspaper home pages
  - course home pages
- **Hubs** are pages that link to authorities
  - List of newspapers
  - Course bulletin

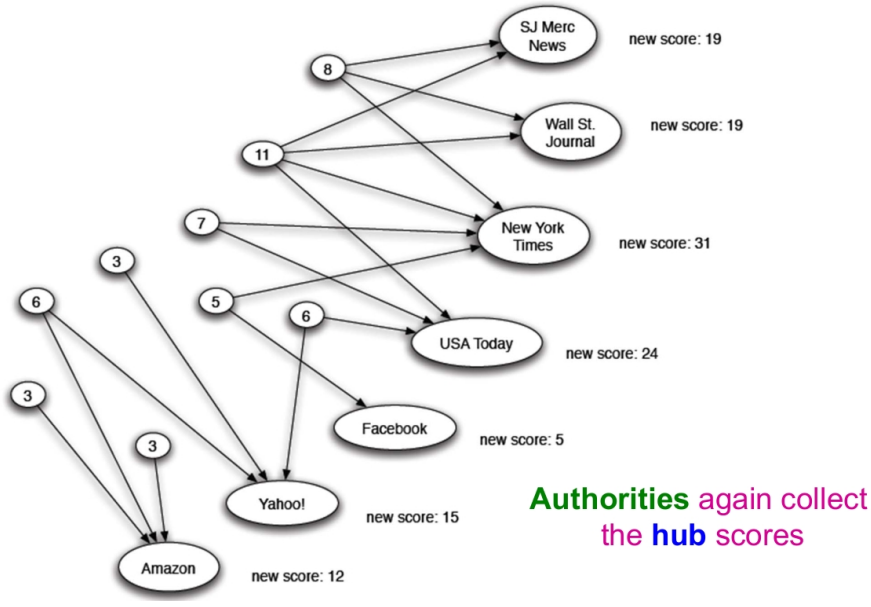
Counting in-links: Authority



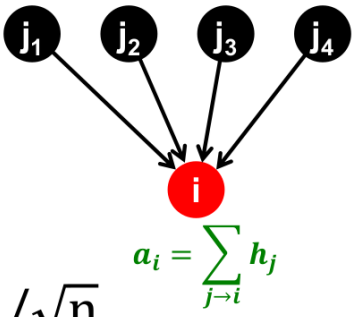
Expert Quality: Hub



Reweightings

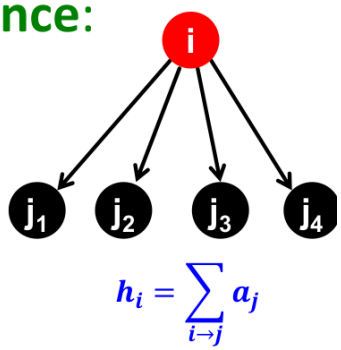


- **Each page i has 2 scores:**
  - Authority score: $\boldsymbol{a_i}$
  - Hub score: $\boldsymbol{h_i}$



HITS algorithm:

- Initialize: $a_j^{(0)} = 1/\sqrt{n}, \quad h_j^{(0)} = 1/\sqrt{n}$
- Then keep iterating until **convergence**:
  - $\forall \boldsymbol{i}$: Authority: $a_i^{(t+1)} = \sum_{j \to i} h_j^{(t)}$
  - $\forall \boldsymbol{i}$: Hub: $h_i^{(t+1)} = \sum_{i \to j} a_j^{(t)}$
  - $\forall \boldsymbol{i}$: Normalize:
    $\sum_i \left(a_i^{(t+1)}\right)^2 = 1, \sum_j \left(h_j^{(t+1)}\right)^2 = 1$



HITS algorithm in vector notation

- Set: $\boldsymbol{a_i} = \boldsymbol{h_i} = \frac{\mathbf{1}}{\sqrt{\boldsymbol{n}}}$

Repeat until convergence:

- $\boldsymbol{h} = \boldsymbol{A} \cdot \boldsymbol{a}$
- $\boldsymbol{a} = \boldsymbol{A^T} \cdot \boldsymbol{h}$
- Normalize $\boldsymbol{a}$ and $\boldsymbol{h}$

Convergence criterion:

$$\sum_i \left(h_i^{(t)} - h_i^{(t-1)}\right)^2 < \varepsilon$$

$$\sum_i \left(a_i^{(t)} - a_i^{(t-1)}\right)^2 < \varepsilon$$

Existence and Uniqueness

Under reasonable assumptions about $\boldsymbol{A}$, HITS converges to vectors $\boldsymbol{h}$' and $\boldsymbol{a}$'

- $\boldsymbol{h}$' is the principal eigenvector of matrix $\boldsymbol{AA^T}$
- $\boldsymbol{a}$' is the principal eigenvector of matrix $\boldsymbol{A^TA}$

PageRank and HITS

Question: What is the value of an in-link from node *u* to *v*?

- PageRank model the value of the link depends on the **links into u**
- HITS model the value of the link depends on the **links out of u**

PageRank is popular in industry while HITS is not (since post 1998)

Video Lecture 12 Web Spam: Intro

What is web spam?

- Spamming
  - Any deliberate action to boost a web page's position in search engine results, incommensurate with page's real value
- Spam
  - Web pages that are the result of spamming
- Approximately 10-15% of web pages are spam

Web Search

Early search engines

- Crawl the Web
- Index pages by the words they contained
- Respond to search queries (lists of words) with the pages containing those words

Early page ranking

- Attempt to order pages matching a search query by "importance"
- First search engines considered
  - Number of times query words appeared
  - Prominence of word position (e.g. title, header)

Spammer tried to exploit search engines to achieve high relevance/importance for a web page

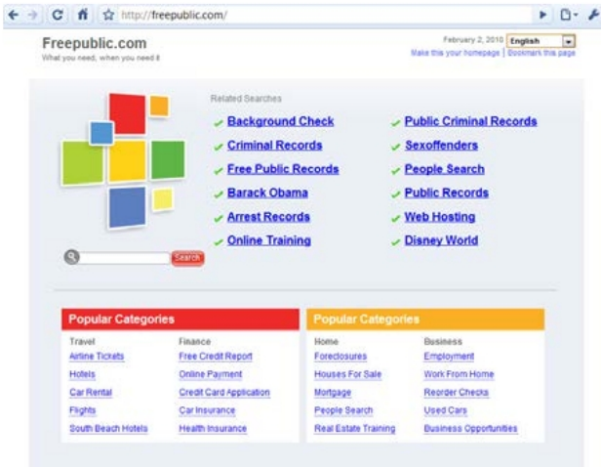
- **term spam**: add important word 1000 times to their pages and made the word "invisible" to users (e.g. set color = background color)

Google's solution to term spam: Use words in the anchor text (words that appear underlined to represent the link) and its surrounding text

Video Lecture 13 Web Spam: Spam Farming

Spam farms were developed to concentrate PageRank on a single page

Link spam: creating link structures that boost PageRank of a particular page

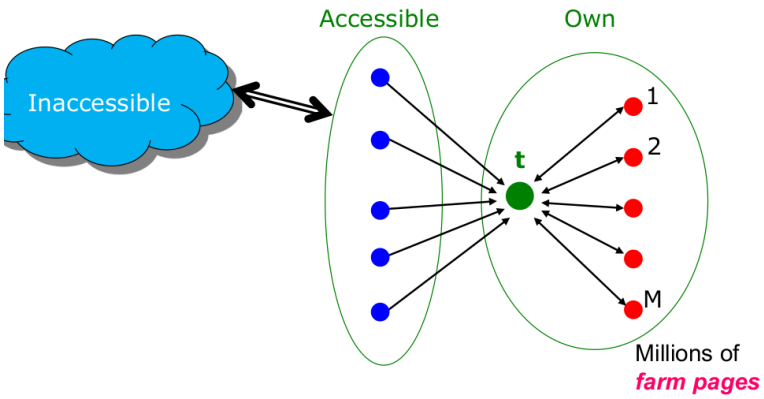


Link spamming (from spammer's view)

- Inaccessible pages
  - pages that spammer cannot touch
- Accessible pages
  - pages that spammer can touch
  - e.g. blog comments pages (spammer can post links to his pages)
- Own pages
  - completely controlled by spammer
  - may span multiple domain names

Link Farms

- Spammer's goal
  - maximize the PageRank of target page ***t***
- Technique
  - get as many links from accessible pages as possible to target page ***t***
  - construct "link farm" to get PageRank multiplier effect



Analysis of link farm

- **x**: PageRank contributed by accessible pages
- **y**: PageRank of target page ***t***
- Rank of each "farm" page $= \frac{\beta y}{M} + \frac{1-\beta}{N}$
- $y = x + \beta M \left[\frac{\beta y}{M} + \frac{1-\beta}{N}\right] + \frac{1-\beta}{N}$

  $= x + \beta^2 y + \frac{\beta(1-\beta)M}{N} + \boxed{\frac{1-\beta}{N}}$

  Very small; ignore

  Now we solve for ***y***
- $y = \frac{x}{1-\beta^2} + c\frac{M}{N}$ where $c = \frac{\beta}{1+\beta}$

By making ***M*** large, spammer can make ***y*** as large as they want

Video Lecture 14 TrustRank: Combating the Web Spam

- Combating term spam
  - analyze text using statistical methods; similar to email spam filtering
- Combating link spam (more expensive)
  - detection and blacklisting of structures that look like spam farms
    - problem: this leads to endless war between search engine and spammer

TrustRank (solution to link spam)

- TrustRank is a topic-specific PageRank with a teleport set of **trusted pages** (eg .edu domains)
- Basic principle: Approximate isolation
  - idea: it’s rare for a good page to point to a bad (spam) page
- Sample a set of **seed pages** from the web
- Have an oracle (human) to identify the good pages and the spam pages in the seed set
  - expensive task => seed set needs to be as small as possible

Trust Propagation

- **trusted pages** = subset of seed pages that are identified as good
- perform a topic-sensitive PageRank with teleport set of trusted pages
  - propagate trust through links
    - each page gets a trust value between 0 and 1
    - **SOLUTION I**: use a threshold value and mark all pages below the trust threshold as spam

Why is it a good idea?

- Trust attenuation
  - the degree of trust conferred by a trusted page decreases with the distance in the graph
- Trust splitting
  - the larger the # of out-links from a page → the less scrutiny (审查) the page author gives each out-link
  - trust is split across out-links

Picking the Seed Set

- 2 conflicting considerations
  - human has to inspect each seed page, so seed set must be as small as possible
  - must ensure every good page gets adequate trust rank, so need to make all good pages reachable from seed set by short paths

Approaches to Pick Seed Set

- PageRank
  - pick the top k pages by PageRank
- Use trusted domains
  - whose membership is controlled (e.g. .edu, .mil, .gov)

SOLUTION II: Spam Mass Estimation

estimate fraction of a page's PageRank that comes from **spam** pages

- r_p = PageRank of page p
- r_p^+ = PagerRank of p with teleport into trusted pages only
- $r_p^- = r_p - r_p^+$
- Spam mass of $p = r_p^- / r_p$
- This is better than solution I, as we calculate the relative score instead of absolute score

Chapter 6 Frequent Itemsets

6.1.1 Definition of Frequent Itemsets

s = support threshold

I = set of items

support for *I* = number of baskets for which *I* is a subset

frequent = *I* is frequent if its support >= *s*

The Market-Basket Model

The market-basket model of data is used to describe a common form of many-many relationship between two kinds of objects.

- To generate all subsets of size k for a basket with n items (n >> k)
  - n choose k ≈ n^k / k

Example

1. {Cat, and, dog, bites}
2. {Yahoo, news, claims, a, cat, mated, with, a, dog, and, produced, viable, offspring}
3. {Cat, killer, likely, is, a, big, dog}
4. {Professional, free, advice, on, dog, training, puppy, training}
5. {Cat, and, kitten, training, and, behavior}
6. {Dog, &, Cat, provides, dog, training, in, Eugene, Oregon}
7. {“Dog, and, cat”, is, a, slang, term, used, by, police, officers, for, a, male–female, relationship}
8. {Shop, for, your, show, dog, grooming, and, pet, supplies}

{} has support = 8, but we generally don’t consider {}

let s = 3

frequent singletons: {dog}, {cat}, {and}, {a}, {training}

frequent doubletons: {dog, a}, {dog, and}, {dog, cat}, {cat, a}, {cat, and}

frequent triples: {dog, cat, a}

6.1.3 Association Rules

$$I \rightarrow j, \text{ (I = a set of items, j = an item)}$$

If all of the items in *I* appear in some basket, **then** *j* is likely to appear in that basket as well.

Confidence (of association rule)

$$\text{conf}(I \rightarrow j) = \text{support}(I \cup j) / \text{support}(I)$$

Interest (of association rule)

Interest = absolute difference between its confidence and the fraction of baskets that contain *j*

$$\text{Interest}(I \rightarrow j) = \text{abs}(\text{conf}(I \rightarrow j) - \Pr[j])$$

$B_1 = \{m, c, b\}$ $B_2 = \{m, p, j\}$
$B_3 = \{m, b\}$ $B_4 = \{c, j\}$
$B_5 = \{m, p, b\}$ $B_6 = \{m, c, b, j\}$
$B_7 = \{c, b, j\}$ $B_8 = \{b, c\}$

Association rule: {m, b} →c

- Support = 2
- Confidence = 2/4 = 0.5
- Interest = |0.5 – 5/8| = 1/8

Mining Association Rules

Step 1. Find all frequent itemsets I

Step 2. Rule generation

For every subset A of I, generate a rule **A** → **I \ A** (I excludes A)

Output the rules above confidence threshold

Observation

1. Since I is frequent, A is also frequent
2. If A,B,C → D is below confidence, so is A,B → C, D

$B_1 = \{m, c, b\}$ $B_2 = \{m, p, j\}$
$B_3 = \{m, c, b, n\}$ $B_4 = \{c, j\}$
$B_5 = \{m, p, b\}$ $B_6 = \{m, c, b, j\}$
$B_7 = \{c, b, j\}$ $B_8 = \{b, c\}$

Support threshold $s = 3$, confidence $c = 0.75$

Step 1) Find frequent itemsets:

- {b,m} {b,c} {c,m} {c,j} {m,c,b}

Step 2) Generate rules:

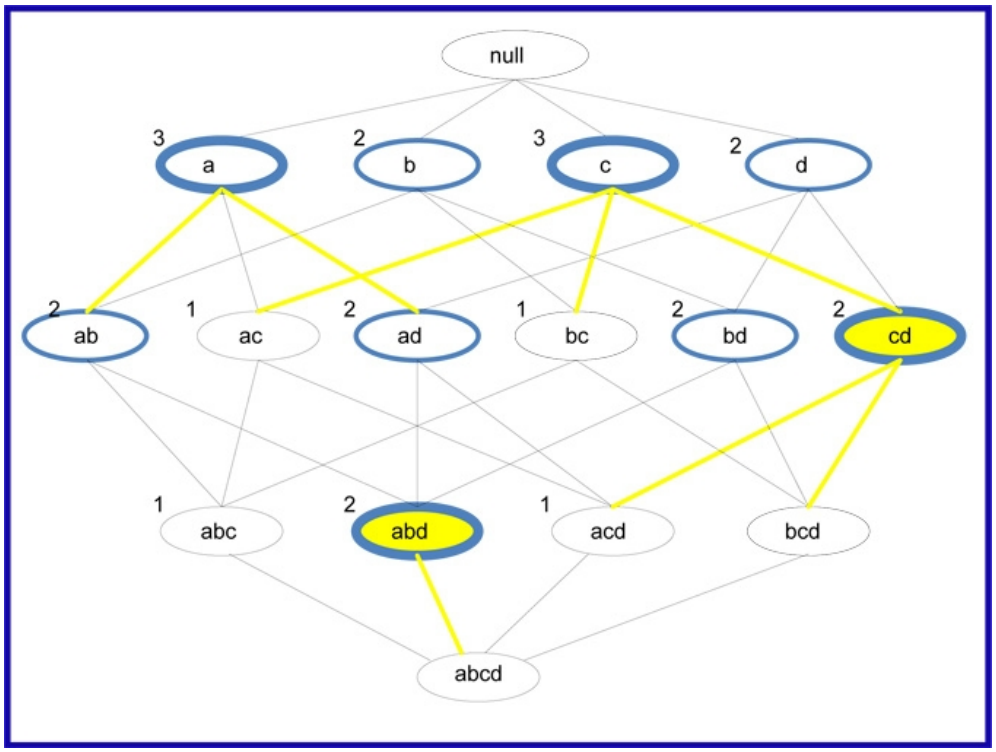
- ~~b→m: c=4/6~~ b→c: c=5/6 ~~b,c→m: c=3/5~~
- m→b: c=4/5 … b,m→c: c=3/4
  ~~b→c,m: c=3/6~~

Compacting the Output

To reduce the number of rules we can take

- **closed itemsets**: no immediate superset has the same support
- **maximal frequent itemsets**: no immediate superset is frequent

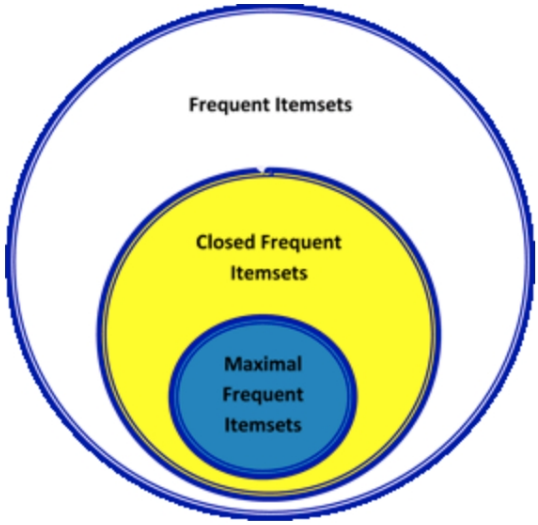
Example



circle with blue = frequent itemsets

circle with thick blue = closed itemsets

circle with thick blue + yellow fill = maximal frequent itemsets



Computation Model

- Typically, data is kept in flat files rather than in a DB system
- Data is stored on disk + stored basket-by-basket
- cost of mining disk-resident data = # of disk I/O

Main-Memory Bottleneck

For many frequent-itemset algorithms, main-memory is the critical resource

Finding Frequent Pairs

- It's hard to find the frequent pairs
- Probability of being frequent drops exponentially with size
- # of sets grows more slowly with size

Naive Algorithm

- Read file once, counting all pairs in main memory
- Use a double-loop to generate all the pairs
  - From each basket of **n** items, generating **n(n-1)/2** pairs
- Fails if (# of items)^2 exceeds main memory

Main-Memory Counting

Approach 1 (**Triangular-matrix approach**)

- count all pairs, with a triangular matrix
- n = total number items
- count pair of items {i, j} only if i < j
- keep pair counts in lexicographic order
  - {1, 2}, {1, 3}, …, {1, n}, {2, 3}, {2, 4}, …, {2, n}, {3, 4}, …, {3, n}, …, {n-1, n}
- When i < j, {i, j} is at position:
  - **(i – 1)(n – i/2) + (j – i)**
- total number of pairs = n(n-1)/2 ≈ n^2 / 2
- total bytes = 2·n^2 (4-byte per pair; assuming each integer is 4-byte)

Approach 2 (**Triples/Tabular approach**)

- Keep a table of triples [i, j, c] = "# of pair of items {i, j} is c" (where i < j)
- total bytes = 12·p, (12-byte per pair; p = # of pairs that actually occur)
- better than approach 1 if at most 1/3 of possible pairs actually occur

A-Priori Algorithm (used to find freq. Pairs)

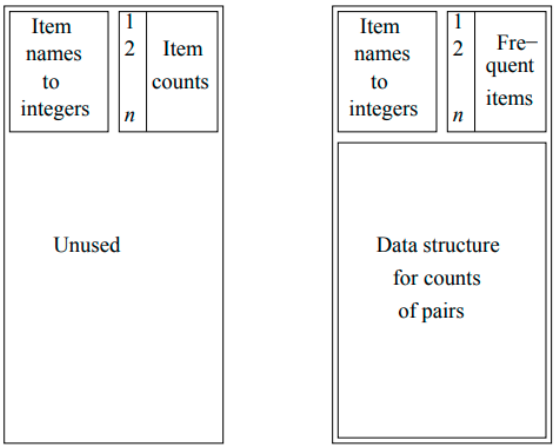
- Most cited work in data mining field
- Two-pass approach, limiting the need for main memory
- **monotonicity**
  - A set cannot be frequent unless all of its subsets are frequent
  - if a set of items ***I*** appears at lest ***s*** times, so does every subset ***J*** of ***I***
- **contrapositive** for pairs
  - if item ***i*** does not appear in ***s*** baskets, then no pair including ***i*** can appear in ***s*** baskets

Pass 1

- in memory, read baskets and count the occurrences of each item. Requires two table
  - Table 1 = mapping of item names to integers (from 1 to n)
  - Table 2 = counts of each item
- items that appear ≥ *s* times are the frequent items

Pass 2

- in memory, read baskets and count pairs of frequent (from pass 1)
- requires memory = (# of frequent items)^2 + a list of the frequent items (s.t. we know what is frequent)



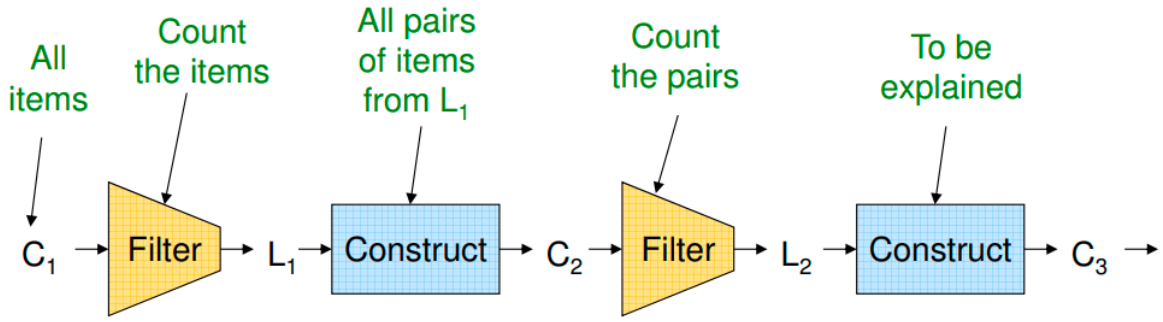
Detail for A-Priori

- Use triangular matrix method with *n* = # frequent items
- Trick: number frequent items 1,2, … and keep a table relating new numbers to original item numbers

>> The idea used on the 2nd pass extends to later passes that construct larger sets

Frequent Triples

- For each *k*, we construct two sets of *k-sets* (size of size *k*)
  - C_k = *candidate k-tuples* = possible frequent sets (support ≥ s) based on info. From the pass for *k-1*
  - L_k = the set of truly frequent *k-sets*



- 1st filter = 1st pass, 2nd filter = 2nd pass
- L_1 = frequent items, L_2 = frequent pairs

A-Priori for All Frequent Itemsets

- One pass for each *k*
- Needs main memory to count each candidate *k-sets*
- For typical market-basket data and reasonable support (e.g 1%), *k = 2* requires the most memory

>> Goal: minimizing # of pairs that actually have to be counted on 2nd pass of A-priori

Improvements to A-Priori

Park-Chen-Yu (PCY) Algorithm

- During 1st pass of A-priori, most memory is idle
- Use that memory to keep counts of buckets into which pairs of items are hashed
    - store count, not the pairs

```
for (each basket):
   for (each item in the basket):
      add 1 to item's count;

   for (each pair of items):
      hash the pair to a bucket;
      add 1 to the count for that bucket;
```

Observation

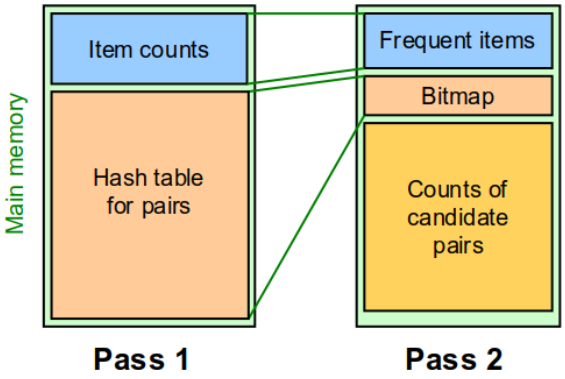
- If a bucket contains *frequent* pair, then the bucket is *frequent*
- :(However, a bucket can still be *frequent* without any *frequent* pair
    - we cannot use the hash to eliminate any member of a “frequent” bucket
- :) But, for a bucket (*count* < *s*), none of its pairs can be frequent
- For pass 2, only count pairs that hash to *frequent* buckets

PCY: Between Passes

- Replace the buckets by a bit-vector (1 = *frequent* bucket):
    - 1 = bucket count > s
    - 0 = bucket count < s
- 4-byte integer counts are replaces by bits
    - bit-vector thus requires 1/32 of original memory
- Prepare a list of *frequent* item for 2nd pass

PCY: Pass 2

- Count all pairs *{i, j}* that meet the conditions for being a candidate pair
  - both *i* and *j* are frequent items
  - the pair *{i, j}* hashes to a *frequent* bucket (bit = 1)
- The two conditions above are **necessary** for the pair to possibly be frequent



Main-Memory Details

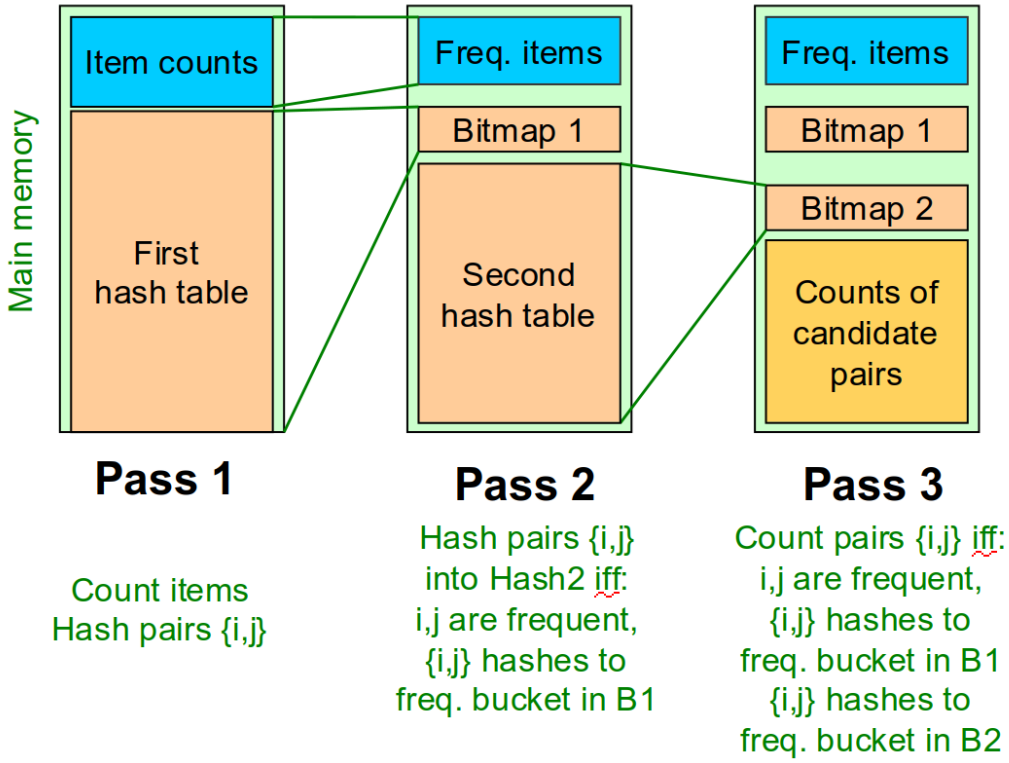
- Buckets require a few bytes each
  - note that we don't count past ***s***
  - # bucket is O(size of main memory)
- on 2nd pass, a table of triples (item, item, count) is essential
  - Goal: hash table must eliminate 2/3 of candidate pairs for PCY to beat A-Priori

Multistage Algorithm (Improvement of PCY)

- After 1st pass of PCY, only rehash those pairs that qualify for 2nd pass
- On middle pass, fewer pairs contribute to buckets, so fewer FP (freq. buckets w/o freq. pair)
- → Requires 3 passes over the data

Multistage: Pass 3

- Count only those *{i, j}* that satisfy following **candidate pair conditions**:
  1. both *i, j* are frequent items
  2. using the 1st hash function, the pair hashes to a bucket whose bit in 1st bit-vec = 1
  3. using the 2nd hash function, the pair hashes to a bucket whose bit in 2nd bit-vec = 1

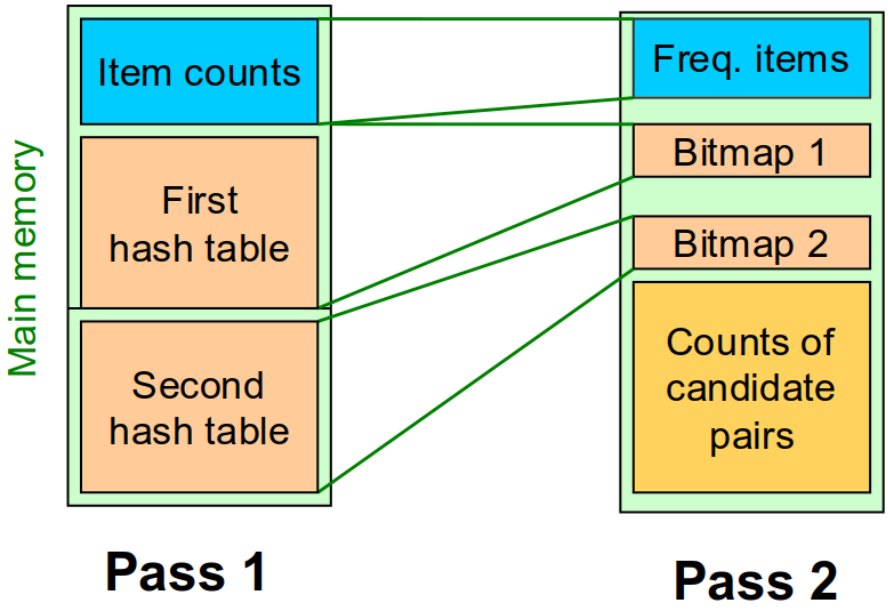


Important:

1. The hash functions must be independent
2. Need to check all hashes for new pass

Refinement: Multihash

- Idea: use several indep. hash tables on 1st pass; a hash table in multi-hash = a pass in multi-stage
- Risk: Halving # of buckets, doubles the average count
  - → need to ensure most buckets count < s



>> A-Priori, PCY take k passes to find frequent itemsets of size k; we want to reduce # of passes

Frequent Itemsets in ≤ 2 Passes

- Use ≤ 2 Passes for all sizes, but may miss some frequent itemsets
  - Random sampling
  - SON algorithm

Random Sampling

- Take a random sample of the market baskets
- Run A-priori or one of its improvements in main memory
  - so we don't pay for disk I/O (recall computation model) each time we increase the size of itemsets
  - reduce and match support threshold *s* to the sample size
- To decrease FP
  - verify candidate pairs are truly frequent by 2nd pass
- To decrease FN
  - use smaller threshold
    - → catch more truly frequent itemsets → requires more space

SON Algorithm

- Repeatedly read small subsets of the baskets into main memory and run an in-memory algo. To find all frequent itemsets
- An itemset is a candidate if it's found to be frequent in *any* one or more subsets of the baskets
- On 2nd pass, count all the candidate itemsets + determine which are frequent in the entire set
  - recall monotonicity

Frequent Items: Recap

- Frequent itemsets = sets of items that often co-occur in baskets
- association rules = Info on frequent itemsets
- confidence(I $\rightarrow$ i) = support($I \cup j$) / support (I)
- rules can be pruned based on interest
  - Interest($I \rightarrow j$) = abs(conf($I \rightarrow j$) – Pr[j])
- Frequent itemsets are monotonic
  - frequent triples contain only frequent pairs
- A-priori algorithm exploits monotonicity to find frequent itemsets
- PCY uses hashing to reduce the set size of candidate pairs C_2
- Multi-stage algorithms do a 2nd pass to further reduce candidate pairs
- Multi-hash algorithms use multiple hash tables at 1st pass
- Limited-pass algorithms perform standard algorithm on random data subset
- FP may be eliminated with ≥ 1 pass over the data
- SON performs randomized algorithms and eliminates FP

Chap 7 Clustering

Given a set of points, with a notion of distance between points, group the points into some number of clusters, so that

- members of a cluster are similar/close to each other
- members of a different clusters are dissimilar

Usually:

- Points are in a high-dimensional space
- Similarity is defined using a distance measure
  - Euclidean, Cosine, Jaccard, Edit distance

Document

- = set of words
  - Jaccard distance = 1 - $|A \cap B| / |A \cup B|$
- = point in space of words
  - $(X_1, X_2, \ldots, X_N)$, where X_i = 1 iff word I appears in doc
  - Euclidean distance
- = vector in space of words
  - vector from origin to $(X_1, X_2, \ldots, X_N)$
  - Cosine distance

Overview: Methods of Clustering

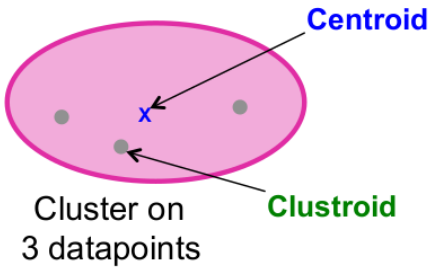
Hierarchical

- Agglomerative (bottom up)
  - Initially, each point is a cluster
  - repeatedly combine the two nearest clusters into one
- Divisive (top down)
  - start with one cluster and recursively split it

Video Lecture 16 Hierarchical Clustering

How to represent a cluster of many points?

- Represent each cluster by its **centroid** = average of its points
- clustroid = an existing point that is *closest* to all other points in the cluster



Possible meanings of closets

- smallest maximum distance to other points
- smallest average distance to other points
- smallest sum of squares of distances to other points

How to determine nearness of clusters?

- Measure cluster distances by distances of centroids

When to stop combining clusters?

- Approach 1: stop when we have ***k*** clusters
- Approach 2: stop when the next merge would create a cluster with low *cohesion*

Cohesion (3 measures)

- **diameter** of the merged cluster = maximum distance between points in the cluster
- **radius** = maximum distance of a point from centroid / clustroid
- **density-based approach**
  - density = number of points per unit volume
  - e.g. divide number of points in cluster by diameter or radius of the cluster
  - perhaps use a power of the radius (e.g square or cube)

Implementation

- Naive: at each step, compute pairwise distances between all pairs of clusters, then merge
  - O(compare) + O(merge) = $O(N^2 \cdot N) = O(N^3)$
- Careful: using priority queue
  - $O(N^2 \log N)$

Video Lecture 17 Clustering: K-Means Algorithm

k-means Algorithm

- assumes Euclidean space/distance
- start by picking k, the number of clusters
- initialize clusters by picking one point per cluster

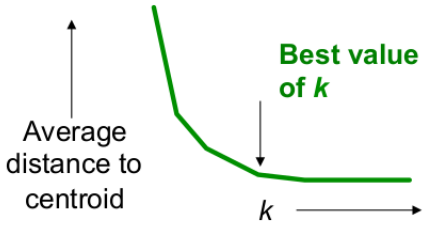
Populating Clusters

1. for each point, place it in the cluster whose current centroid it is nearest
2. after all points are assigned, update the locations of centroids of the k clusters
3. reassign all points to their closet centroid

- Repeat 2 and 3 until convergence (points don't move anymore)

Picking the right value for ***k***

- Average falls rapidly until right *k*, then falls much more slowly



Picking the initial ***k*** points

- Approach 1: Sampling
  - cluster a sample of the data using hierarchical clustering to obtain *k* clusters
  - pick a point from each cluster (e.g. point closet to centroid)
  - sample fits in main memory
- Approach 2: Pick dispersed set of points
  - pick first point at random
  - pick the next point to be the one whose minimum distance from the selected points is as large as possible
  - repeat until we have *k* points

Complexity

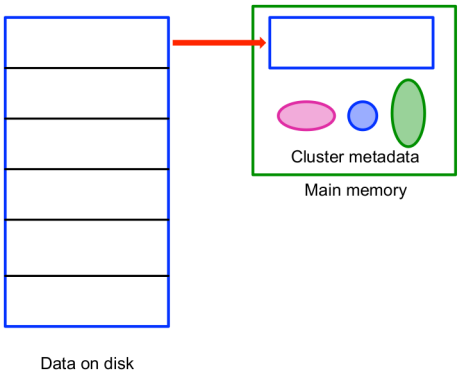
In each round, we have to examine each input point exactly once to find closet centroid

- each round = ***O(kN)*** for *N* points, *k* clusters

Video Lecture 18 Clustering BFR (Bradley-Fayyad-Reina) Algorithm

- BFR is a variant of k-means for very large data sets (that won't fit in memory)
- assume each cluster is **normally distributed** around a centroid in Euclidean space

- Data is large that it won't fit in main memory, we instead put one truck of data into memory
- At the same time, we maintain cluster metadata of all created clusters in main memory
- We pick the initial k centroids using the techniques from k-means algorithm



Three classes of Points (that we keep track of)

- Discard set (DS)
  - contain points close enough to a existing centroid (to be summarized)
  - we "discard" these points because they don't live long
- Compression set (CS)
  - contain groups of points that are close to each other but not close to any centriod
  - These points are summarized, but not assigned to a cluster
- Retained set (RS)
  - contain isolated points (that neither in DS nor in CS)

Summarizing Sets of Points

For each cluster, the DS is summarized by

- The number of points, ***N***
- The vector ***SUM***, ith component = sum of the coordinates of the points in the i^{th} dimension
- The vector ***SUMSQ***, ith component = sum of squares of coordinates in i^{th} dimension

Comments

- ***2d + 1*** (d = # of dimension) can represent any size cluster
- Average in each dimension
  - $\boldsymbol{SUM_i / N}$
- Variance of a cluster's discard set in dimension I
  - $\boldsymbol{(SUMSQ_i / N) – (SUM_i / N)^2}$

Actual clustering

1. Find those points that are *sufficiently close* to a cluster centroid
2. Add those points to that cluster and **DS**
   - then discard the point
3. **DS**: adjust statistics of each cluster to account for newly added points
   - add **N**s, **SUM**s, **SUMSQ**s
4. The remaining points are not close to any cluster
5. Use any main-memory clustering algorithm to cluster these points and the old **RS**
   - clusters go to the **CS**; outlying points go to new **RS**
   - e.g. Hierarchical clustering, k-means clustering
6. Consider merging mini-clusters in the **CS**
7. If this is the last round, merge all compressed sets in the **CS** and all **RS** points into their nearest cluster; if this is not the last round, keep the **CS** and **RS**, read the next trunk of data, and repeat step 1-7

Question: How close is *sufficiently close*?

- Use **Mahalanobis distance** to quantify the likelihood of a point belonging to a centroid

Mahalanobis distance

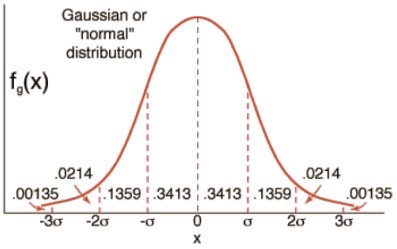
- Cluster *C* has centroid $(c_1,\ldots,c_d)$ and standard deviations $(\sigma_1,\ldots\sigma_d)$
- Point P = $(x_1,\ldots,x_d)$
- Normalized distance in dimension i:

  $y_i = (x_i - c_i)/\sigma_i$
- MD of point P from cluster C:

$$\sqrt{\sum_{i=1}^{d} y_i^2}$$

Mahalanobis Acceptance Criterion

- Suppose point P is one standard dimension away from centroid in each dimension
  - Each $y_i = 1$ and so the MD of P is $\sqrt{d}$



68% of points have MD $\leq \sqrt{d}$

95% of points have MD $\leq 2\sqrt{d}$

99% of points have MD $\leq 3\sqrt{d}$

Accept point P into cluster C if its MD from cluster centroid is less than a threshold e.g., $3\sqrt{d}$

Question: Should 2 CS mini-clusters be combined?

- Compute the variance of the combined mini-clusters
  - note that ***N***, ***SUM***, and ***SUMSQ*** is known
- Combine if the combined variance is below some threshold

Video Lecture 19 Cluster: CURE algorithm

Limitations of BFR Algorithm

- BFR assumes clusters normally distributed in each dimension
- BFR assumes axes are fixed

CURE Algorithm

- Clustering Using Representatives
- two-pass algorithm
- assumes a euclidean distance
- allows clusters to assume any shape
- uses a collection of representative points to represent clusters
    - instead of one point (centroid) in the BFR algorithm

1st Pass of CURE

- pick a random sample of points that fit in main memory
- cluster sample points hierarchically to create the initial clusters
- pick representative points:
    - for each cluster, pick *k* representative points, as dispersed as possible
    - move each representative point a fixed fraction (eg 20%) toward the centroid of the cluster

2nd Pass of CURE

- Now, rescan the whole dataset and visit each point *p* in the data set
- place p in the closet cluster
    - the cluster with the closet (to *p*) among all the representative points of all the clusters

Chap 11 Dimensionality reduction

- Assumption: Data lies on/near a low d-dimensional **subspace**

Rank of a Matrix

- Rank = # of linearly independent columns of a matrix
- we can use the nature of linear system to reduce the number of coordinates (= dimension)

why reduce dimensions?

- Discover hidden correlations/topics
- Remove redundant and noisy features
- Interpretation and visualization
- Easier storage and processing of the data

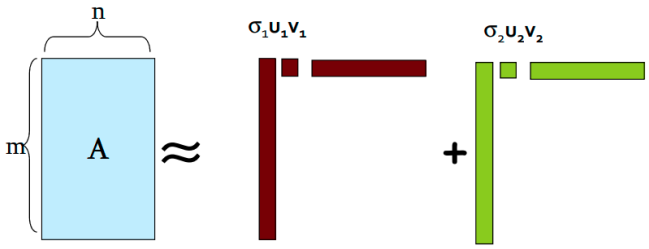
Singular Value Decomposition (SVD)

$$\mathbf{A}_{[m \times n]} = \mathbf{U}_{[m \times r]} \Sigma_{[r \times r]} (\mathbf{V}_{[n \times r]})^T$$

- **A**: **Input data matrix**
  - *m* x *n* matrix (e.g., *m* documents, *n* terms)
- **U**: **Left singular vectors**
  - *m* x *r* matrix (*m* documents, *r* concepts)
- **Σ**: **Singular values**
  - *r* x *r* diagonal matrix (strength of each 'concept') (*r* : rank of the matrix **A**)
- **V**: **Right singular vectors**
  - *n* x *r* matrix (*n* terms, *r* concepts)

Singular values is sorted from largest to smallest

$$\mathbf{A} \approx \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T = \sum_i \sigma_i \mathbf{u}_i \circ \mathbf{v}_i^T$$



SVD: Properties

It is **always** possible to decompose a real matrix A into $A = U\sum V^T$, where

- U, ∑, V: unique
- U, V: column orthonormal
  - $U^TU = I$, $V^TV = I$
  - columns are orthogonal (perpendicular) unit vectors
- ∑: diagonal
  - entries (singular values) are positive and sorted in decreasing order

SVD: Example

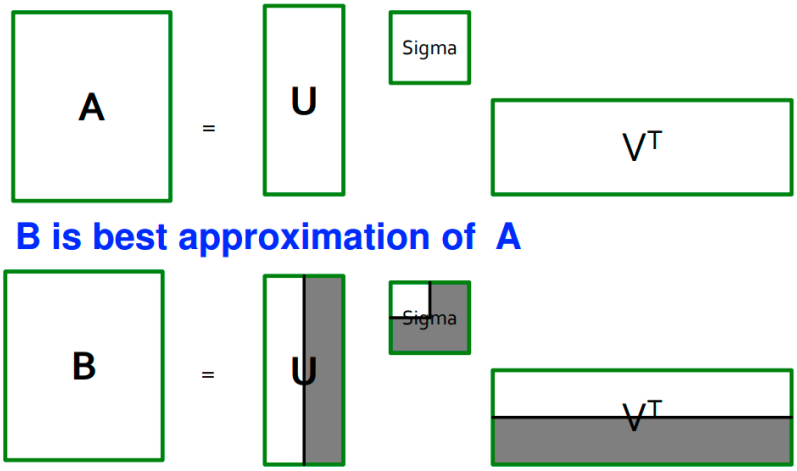
- U: user-to-concept similarity matrix
- V: movie-to-concept similarity matrix
- ∑: its diagonal elements: ‘strength’ of each concept

SVD – min reconstruction error

- SVD gives ‘best’ axis to project on
  - best = min sum of squares of projection errors

SVD: set smallest singular values to zero (and remove corresponding col/row in U, V)

SVD – Best Low Rank Approximation



Frobenius norm:

$\|M\|_F = \sqrt{\Sigma_{ij} M_{ij}^2}$

$\|A-B\|_F = \sqrt{\Sigma_{ij} (A_{ij}-B_{ij})^2}$ is "small"

SVD: Best Low Rank Approximation

Theorem:

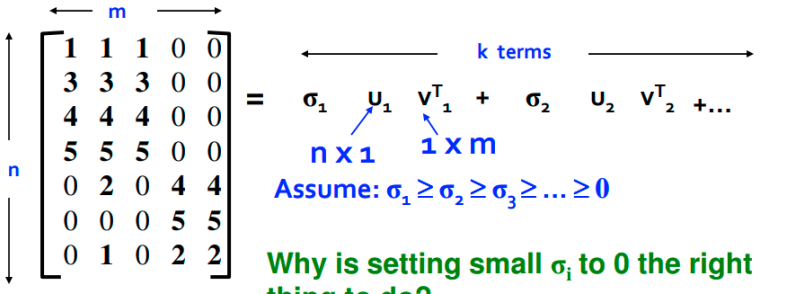
- Let $\boldsymbol{A = U \sum V^T}$ and $\boldsymbol{B = U S V^T}$
  - $\boldsymbol{S}$ = ***diagonal r x r matrix*** with $\boldsymbol{s_i = \sigma_i}$ $\boldsymbol{(i = 1 \ldots k)}$ else $\boldsymbol{s_i = 0}$
  - B is a solution to $\boldsymbol{min_B \| A - B \|_F}$ where ***rank(B) = k***
  - proof detail on the slides

$$\begin{pmatrix} x_{11} & x_{12} & \cdots & x_{1n} \\ x_{21} & x_{22} & \cdots & \\ \vdots & \vdots & \ddots & \\ x_{m1} & & & x_{mn} \end{pmatrix}_{m \times n} = \begin{pmatrix} u_{11} & \cdots & \\ \vdots & \ddots & \\ u_{m1} & & \end{pmatrix}_{m \times r} \begin{pmatrix} \sigma_{11} & 0 & \cdots \\ 0 & \ddots & \\ \vdots & & \end{pmatrix}_{r \times r} \begin{pmatrix} v_{11} & \cdots & v_{1n} \\ \vdots & \ddots & \\ & & \end{pmatrix}_{r \times n}$$

$$\|A - B\|_F = \sqrt{\sum_{ij} (A_{ij} - B_{ij})^2}$$

Equivalent:

'spectral decomposition' of the matrix



Why is setting small σ_i to 0 the right thing to do?

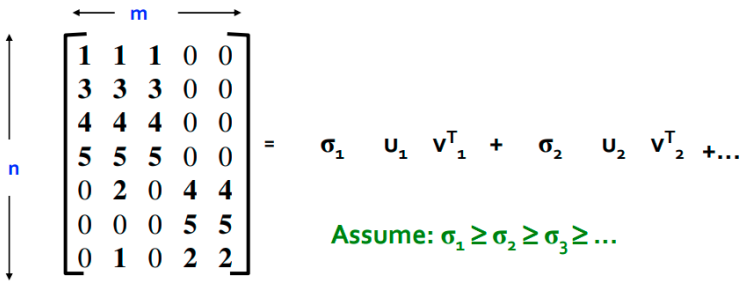
Vectors $\mathbf{u}_i$ and $\mathbf{v}_i$ are unit length, so σ_i scales them.

So, zeroing small σ_i introduces less error.

Q: How many σ_s to keep?

A: Rule-of-a thumb:

keep 80-90% of 'energy' $= \sum_i \sigma_i^2$



SVD: Conclusion

- Complexity: **min(O(nm^2), O(n^{2}m))**
- dim. reduction: keep the few largest singular values (80% - 90% of 'energy')
- **pro**
  - optimal low-rank approximation in terms of Frobenius norm
- **con**
  - hard to interpret (a singular vec = a lin comb of all input col/row)
  - lack of sparsity
    - singular vectors are dense
  - computation time

SVD: Relation to Eigen-decomposition

- SVD
  - $A = U \sum V^T$
- Eigen-decomposition
  - $A = X \lambda X^T$
    - A is symmetric
    - U, V, X are orthonormal (UTU = I)
    - λ, $\sum$ **are diagonal**

Compute SVD using eigen-decomposition

$X \Lambda^2 X^T$

- $AA^T = U\Sigma V^T(U\Sigma V^T)^T = U\Sigma V^T(V\Sigma^T U^T) = U\Sigma\Sigma^T U^T$
- $A^TA = V \Sigma^T U^T (U\Sigma V^T) = V \Sigma\Sigma^T V^T$

$X \Lambda^2 X^T$

SVD: Properties

- $A A^T = U \Sigma^2 U^T$
- $A^TA = V \Sigma^2 V^T$
- $(A^TA)^k = V \Sigma^{2k} V^T$
  - E.g.: $(A^TA)^2 = V \Sigma^2 V^T V \Sigma^2 V^T = V \Sigma^4 V^T$
- $(A^TA)^k \sim v_1 \sigma_1^{2k} v_1^T$ for k>>1

Check slide for exmaple (how to query?)

CUR Decomposition

- Better than SVD
- Goal: express matrix A as a product of matrices, C, U, R
  - make $||A- CUR||_F$ as small as possible
- C = a set of **c**olumns from A
- R = a set of **r**ows from A
- U = pseudo-inverse of the intersection of C and R

CUR: Provably good approx. to SVD

- Let A_k = the 'best' rank ***k*** approximation to A
  - i.e. A_k is SVD of A

Theorem [Drineas et al.]

CUR in O(**m·n**) time achieves

- $\|A-CUR\|_F \leq \|A-A_k\|_F + \varepsilon\|A\|_F$

with probability at least **1-δ**, by picking

- $O(k \log(1/\delta)/\varepsilon^2)$ columns, and
- $O(k^2 \log^3(1/\delta)/\varepsilon^6)$ rows

In practice:
Pick 4*k* cols/rows

CUR: Algorithm

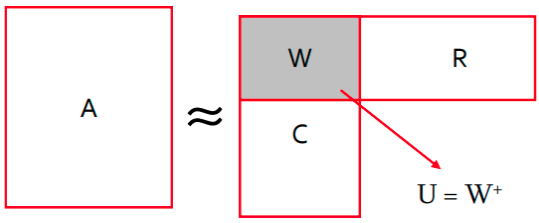
- Sampling columns (similar for rows)
- This is a randomized algorithm, same column can be sampled more than once!!

Input: matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$, sample size c
Output: $\mathbf{C}_d \in \mathbb{R}^{m \times c}$

1. for $x = 1 : n$ [column distribution]
2. $P(x) = \sum_i \mathbf{A}(i,x)^2 / \sum_{i,j} \mathbf{A}(i,j)^2$
3. for $i = 1 : c$ [sample columns]
4. Pick $j \in 1 : n$ based on distribution $P(x)$
5. Compute $\mathbf{C}_d(:,i) = \mathbf{A}(:,j) / \sqrt{cP(j)}$

CUR: Computing U

- Let **W** be the "intersection" of sampled **C** and **R**
  - Let SVD of **W = X Z Y^T**
- Then **U = W$^+$ = Y Z$^+$ X^T**
  - **Z^+_i = 1 / Z$_i$**
  - **W$^+$** is the pseudo-inverse



Why pseudo-inverse works?

W = X Z Y, then **W^{-1} = X^{-1} Z^{-1} Y^{-1}**

Due to orthonomality, **X^{-1} = X^T** and **Y^{-1} = Y^T**

Since **Z** is diagonal **Z^{-1} = 1 / Z$_i$**

Thus, if **W** is on-singular, pseudo-inverse is the true inverse

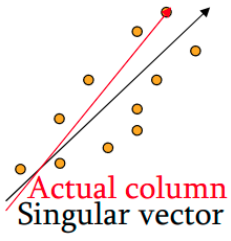
CUR: Example

- **For example:**
  - **Select $c = O\left(\frac{k \log k}{\varepsilon^2}\right)$ columns of A using ColumnSelect algorithm**
  - **Select $r = O\left(\frac{k \log k}{\varepsilon^2}\right)$ rows of A using ColumnSelect algorithm**
  - **Set $U = W^+$**
- **Then:** $\|A - CUR\|_F \leq (2+\epsilon)\|A - A_k\|_F$ (CUR error / SVD error) with probability 98%

In practice:
Pick 4*k* cols/rows
for a "rank-k" approximation

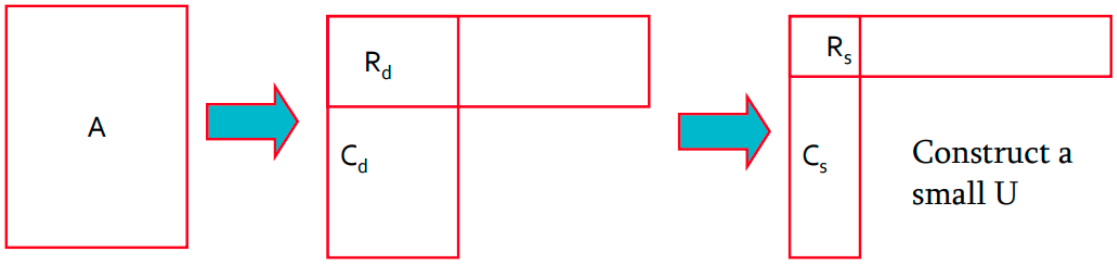
CUR: Conclusion

- Pro
  - easy interpretation (since basis vectors are actual cols and rows)
  - sparse basis (since basis vectors are actual cols and rows)
- Con
  - duplicate cols and rows (cols of large norms will be samples many times)

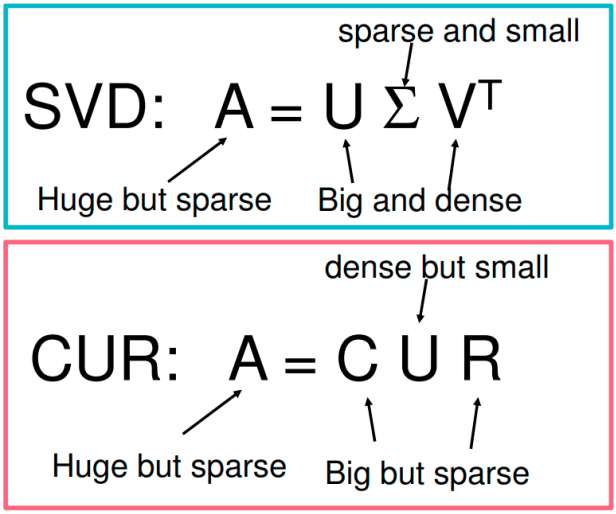


CUR: Remove Duplicates

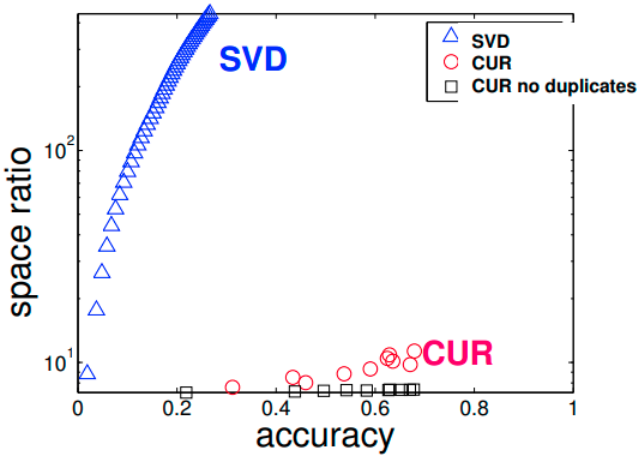
1. remove duplicates
2. scale (multiply) the cols/rows by √(# of dup)

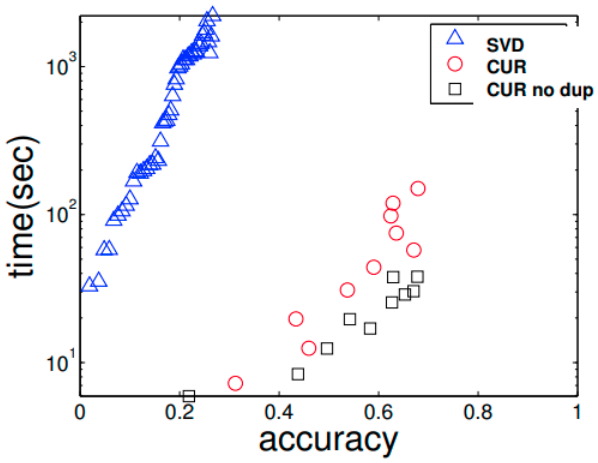


SVD vs. CUR



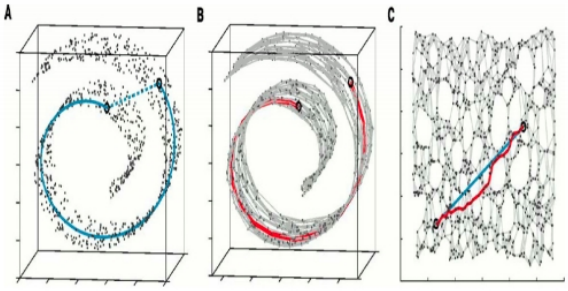
SVD vs. CUR: Example (DBLP – big sparse matrix)





Dim. Reduction: what aobut linearity assumption?

- SVD is limited to linear projections
  - Lower-dimensional linear projection that preserves Euclidean distances
- Non-linear methods: Isomap (related to CS4220-ML)
  - data lies on a nonlinear low-dim curve (a.k.a manifold)
    - use the distance as measured along the manifold
  - How?
    - Build adjacency graph
    - Geodesic distance is graph distance
    - SVD/PCA the graph pairwise distance matrix



Chap 9 Recommender System

From scarcity to abundance

- Shelf space is a scare commodity for traditional retailers
- The web enables near-zero-cost dissemination of information about products
    - **Long Tail** phenomenon

Types of Recommendations

- Editorial and hand curated (策划)
    - e.g. list of editor picked / favorites
    - depends on editors
- Simple aggregates
    - e.g Top 10, Most Popular
    - depends on other users
- Tailored to individual users
    - e.g Amazon, Netflix
    - depends on the user

Formal Model

- Utility function $\boldsymbol{u: C \times S \rightarrow R}$
- C = set of Customers, S = set of Items
- R = set of ratings; (R is a totally ordered set)

Key Problems

1. Gathering known ratings for matrix
2. Extrapolate unknown ratings from the known ones
    - we only interested in *high unknown ratings*
        - we’re interested in what user like than they don’t like
3. Evaluating extrapolation methods

Gathering Ratings

- Explicit
  - ask people to rate items
  - **issue**: doesn't scale (lots of user don't bother rating)
- Implicit
  - learn ratings from user actions (e.g purchase)
  - **issue**: hard to learn low ratings

In practice, companies user a combination of these two.

Extrapolating Utilities

- **Issue**: matrix U is sparse
  - most people haven't rated most items
  - **cold start**
    - new items have no ratings
    - new users have no history
- Three approaches to recommender systems
  1. Content-based
  2. Collaborative
  3. Latent factor based

Content-based Recommendations

Idea: recommend items to customer **x** similar to previous items rated highly by **x**

Model:

user → *likes item* → item profiles → *build* → user profile → *match* → products → *recommend* → user

Item Profiles

- For each item, create an item profile
- Profile is a set of features
  - movies: author, title, actor…
  - images: metadata, tags…
  - people: set of friends…

Text features

- Profile = set of “important” words in item (document)
- pick important words: use **TF-IDF**

User Profiles

- User has rated items with profiles $\boldsymbol{i_1 \ldots i_n}$
  - $\boldsymbol{i_i}$ is vector
- Simple: (weighted) average of rated item profiles
- Variant: Normalize weights using average rating of user

Example 1: Boolean Utility Matrix

- Items are movies, only feature is “Actor”
  - item profile: vector with 0/1 for each Actor
- Suppose user x has watched 5 movies
  - 2 movies featuring actor A
  - 3 movies featuring actor B
- User profile = np.mean(item profiles)
  - Feature A’s weight = 2/5 = 0.4; Feature B’s weight = 3/5 = 0.6

Example 2: Star Ratings

- Same example, 1-5 ratings
  - Actor A's movies rated 3 and 5
  - Actor B's movies rated 1, 2 and 4
- **issue**: users have different standards (e.g. 4 is high for user X while 4 is just average for user Y)
- **solution**: normalize ratings by subtracting user's mean rating
  - mean = (3 + 5 + 1 + 2 + 4) / 5 = 3
  - Actor A's norm. ratings = 0, +2
    - Profile weight = (0 + 2) / 2 = 1
  - Actor B's norm. ratings = -2, -1, +1
    - Profile weight = (-2 -1 + 1) / 3 = -2/3

Making predictions

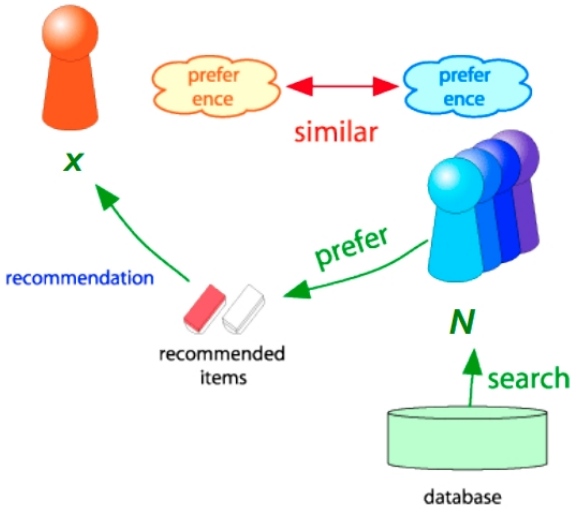
- Use ***cosine distance***
- Given a pair of (user profile ***x***, item profile ***I***)
  - Estimate $U(x, I) = cos(\theta) = np.dot(x, I) / (||x|| \cdot ||i||)$
  - the smaller θ, the higher similarity

Content-based Approach

Pro	Con
+ No need for data on other users + Able to recommend to users with unique tastes + Able to recommend new & unpopular items (no first-rater problem) + Explanations for recommended items	- Finding the appropriate features is hard - Over-specialization - Never recommends items outside user's content profile - People have numerous interests - Unable to exploit quality judgments of other users - Cold-start problem for new users

Collaborative Filtering

- Consider user ***x***
- Find set ***N*** of other users whose ratings are similar to ***x***'s ratings
- Estimate ***x***'s ratings based on ratings of users in ***N***



Similarity measures

- Jaccard similarity (bad)
  - only considers common movies, not ratings
- Cosine similarity (not so good)
  - different users have different standards → subtract row mean → Pearson correlation
- **Pearson correlation** (a.k.a centered cosine similarity)

Rating Predictions (User-User CF)

- Let $\boldsymbol{r_x}$ be the vector of user ***x***'s ratings
- Let ***N*** be the set of ***k*** users most similar to ***x***, who have also rated item ***i***
- Prediction for user ***x*** and item I
  - where $\boldsymbol{S_{xy}}$ **= *sim(x, y)***

$$\text{Option 1: } r_{xi} = 1/k \sum_{y \in N} r_{yi}$$

$$\text{Option 2: } r_{xi} = \sum_{y \in N} s_{xy} r_{yi} / \sum_{y \in N} s_{xy}$$

Item-item Collaborative Filtering

- For item ***i***, find other similar items
- Estimate rating for item ***i*** based on ratings for similar items
- Can use same similarity metrics and prediction functions as in user-user model

$$r_{xi} = \frac{\sum_{j \in N(i;x)} s_{ij} \cdot r_{xj}}{\sum_{j \in N(i;x)} s_{ij}}$$

s_{ij}… similarity of items ***i*** and ***j***
r_{xj}…rating of user ***u*** on item ***j***
N(i;x)… set items rated by ***x*** similar to ***i***

Example: Item-item CF (|N| = 2)

1. Calculate row mean for each row, and subtract the mean for known values; then fill blank as 0
   - row1 = [-2.6, 0, -0.6, …, 0.4, 0]
2. use centered cosine similarity to calculate the similarity score
3. Find highest N scores (items)
   - e.g. N = 2 and top 2 is row 3 (0.41) and last row (0.59)
4. ? = (0.41 * 2 + 0.59 * 3) / (0.41 + 0.59) = 2.6

users

movies	1	2	3	4	**5**	6	7	8	9	10	11	12
1	1		3		?	5			5		4	
2			5	4			4			2	1	3
3	2	4		1	2		3		4	3	5	
4		2	4		5			4			2	
5			4	3	4	2					2	5
6	1		3		3			2			4	

CF: Common Practice (evolved)

- μ = overall mean movie rating
- b_x = rating deviation of user ***x*** = *(avg. rating of user x) – μ*
- b_i = rating deviation of movie ***i***
- $\boldsymbol{b_{xi}}$ = *global baseline estimate for* $\boldsymbol{r_{xi} = \mu + b_x + b_i}$

$$r_{xi} = b_{xi} + \frac{\sum_{j \in N(i;x)} s_{ij} \cdot (r_{xj} - b_{xj})}{\sum_{j \in N(i;x)} s_{ij}}$$

CF: Complexity

- Expensive to find k most similar users
  - O(| size of utility matrix |)
- Expensive to do it at runtime
  - pre-compute
    - O(n·|U|) = O(#user · utility matrix)
- Some techniques
  - Near-neighbor search in high dimensions (Locality-Sensitive Hashing (LSH))
  - Clustering
  - Dimensionality reduction

CF: Evaluating Predictions

- Compare predictions with known ratings
  - RMSE (N = size of test set; r_{xi} is predition r_{xi}^* is actual)

    $$\sqrt{\frac{\sum_{(x,i)\in T}(r_{xi}-r^*_{xi})^2}{N}}$$
  - Precision at top 10
  - Rank Correlation
- 0/1 model
  - Coverage: # of items/users for which system can make predictions
  - Precision: Accuracy of predictions
  - Receiver operating characteristic (ROC): Tradeoff curve between FP and FN

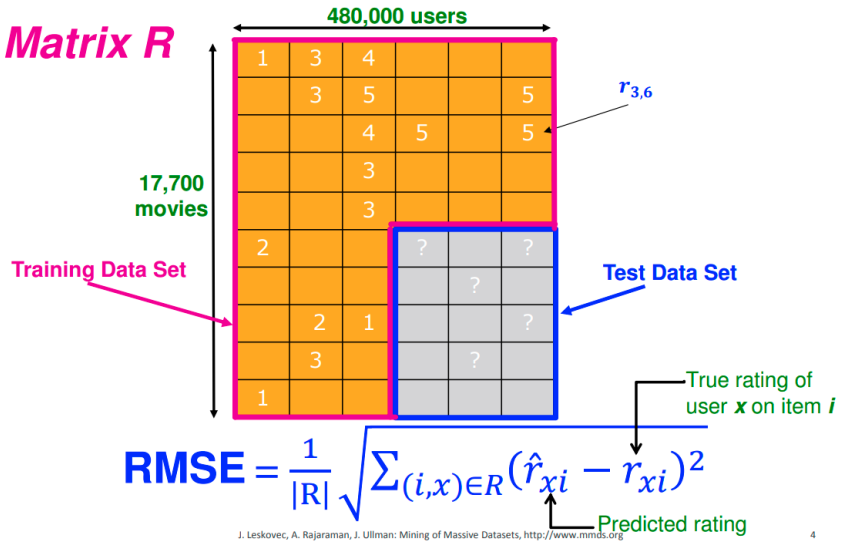
Collaborative Filtering: Conclusion

- In practice, item-item is better than user-user (items are much simpler to describe)

Pro	Con
+ Works for any kind of item No feature selection needed	- Cold Start Need enough users in the system to find a match - Sparsity - First rater - Popularity bias Cannot recommend items to someone with unique taste Tends to recommend popular items

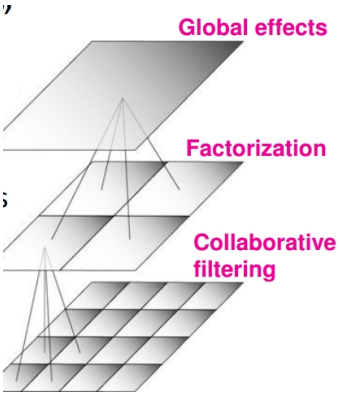
Recommender Systems: Latent 潜在 Factor Models

- The Netflix Prize: movie recommender system
- most user haven't seen most movies



BellKor Recommender System

- Multi-scale modeling of the data
  - **Global**: overall deviations of users/movies
  - **Factorization**: Addressing regional effects
  - **CF**: extract local patterns



Modeling Local & Global Effects

Recall:

$$\hat{r}_{xi} = b_{xi} + \frac{\sum_{j\in N(i;x)} s_{ij}\cdot(r_{xj} - b_{xj})}{\sum_{j\in N(i;x)} s_{ij}}$$

issues

- similarity measure are arbitrary
- pairwise similarities neglect interdependencies among users
- Takeing a weighted avg can be restricing

solution: instead of s_{ij} use w_{ij} that we estimate directly from data

Interpolation Weights w_{ij}

- $\widehat{r_{xi}} = b_{xi} + \sum_{j \in N(i,x)} w_{ij}(r_{xj} - b_{xj})$
- **How to set w_{ij}?**
  - Remember, error metric is: $\frac{1}{|R|}\sqrt{\sum_{(i,x)\in R}(\hat{r}_{xi} - r_{xi})^2}$ or equivalently **SSE:** $\sum_{(i,x)\in R}(\hat{r}_{xi} - r_{xi})^2$
  - Find w_{ij} that minimize **SSE** on **training data!**
    - Models relationships between item ***i*** and its neighbors ***j***
  - w_{ij} can be **learned/estimated** based on ***x*** and all other users that rated ***i***

Gradient Descent

$$J(w) = \sum_{x} \left(\left[b_{xi} + \sum_{j \in N(i;x)} w_{ij}(r_{xj} - b_{xj}) \right] - r_{xi} \right)^2$$

- **Iterate until convergence: $w \leftarrow w - \eta \nabla_w J$** η … learning rate
- **where $\nabla_w J$ is the gradient (derivative evaluated on data):**

$$\nabla_w J = \left[\frac{\partial J(w)}{\partial w_{ij}} \right] = 2 \sum_{x,i} \left(\left[b_{xi} + \sum_{k \in N(i;x)} w_{ik}(r_{xk} - b_{xk}) \right] - r_{xi} \right)(r_{xj} - b_{xj})$$

for $j \in \{N(i;x), \forall i, \forall x\}$

else $\frac{\partial J(w)}{\partial w_{ij}} = 0$

- **Note:** We fix movie ***i***, go over all r_{xi}, for every movie $j \in N(i;x)$, we compute $\frac{\partial J(w)}{\partial w_{ij}}$

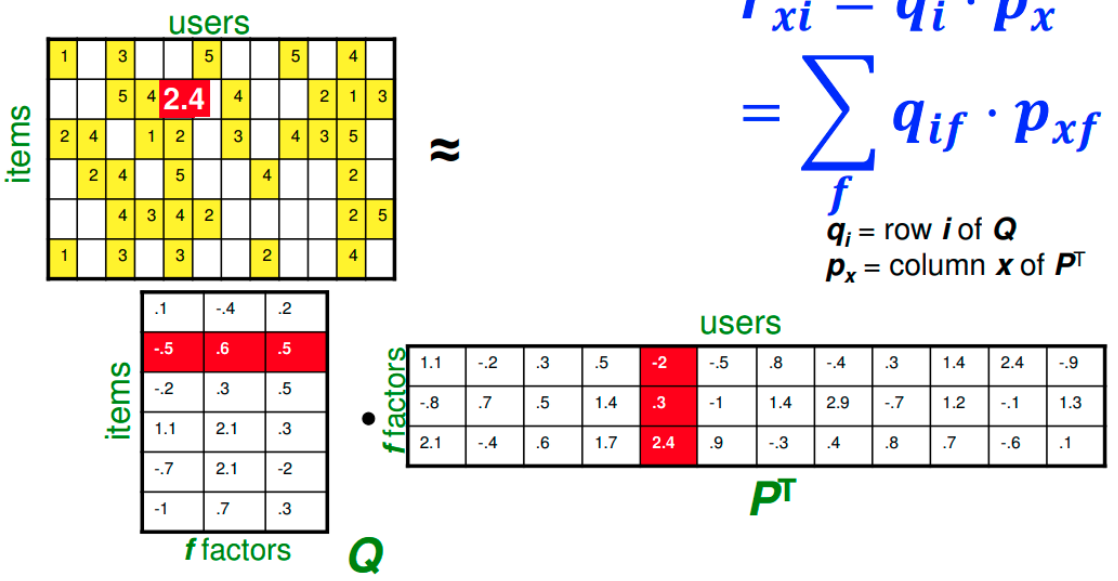
while $|w_{new} - w_{old}| > \varepsilon$:

$w_{old} = w_{new}$

$w_{new} = w_{old} - \eta \cdot \nabla w_{old}$

Latent Factor Models ($R \approx Q \cdot P^T$)

- **How to estimate the missing rating of user *x* for item *i*?**



"SVD" on Netflix data: $R \approx Q \cdot P^T$

$A = R,\ Q = U,\ P^T = \Sigma V^T$

SVD gives minimum reconstruction error

$$\min_{U,V,\Sigma} \sum_{ij \in A} \left(A_{ij} - [U\Sigma V^T]_{ij}\right)^2$$

Our goal is to find P and Q such tat:

$$\min_{P,Q} \sum_{(i,x) \in R} \left(r_{xi} - q_i \cdot p_x\right)^2$$

>> R has missing entries

Dealing with Missing Entries

$$\min_{P,Q} \sum_{training} (r_{xi} - q_i p_x)^2 + \left[\lambda_1 \sum_x \|p_x\|^2 + \lambda_2 \sum_i \|q_i\|^2 \right]$$

"error" — "length"

λ_1, λ_2 … user set regularization parameters

Stochastic Gradient Descent

- **Stochastic gradient decent:**
  - Initialize ***P*** and ***Q*** (using SVD, pretend missing ratings are 0)
  - Then iterate over the ratings (multiple times if necessary) and update factors:

    For each r_{xi}:
    - $\varepsilon_{xi} = 2(r_{xi} - q_i \cdot p_x)$ (derivative of the "error")
    - $q_i \leftarrow q_i + \mu_1 (\varepsilon_{xi} p_x - \lambda_2 q_i)$ (update equation)
    - $p_x \leftarrow p_x + \mu_2 (\varepsilon_{xi} q_i - \lambda_1 p_x)$ (update equation)

    μ … learning rate
- **2 for loops:**
  - For until convergence:
    - For each $\mathbf{r_{xi}}$
      - Compute gradient, do a "step"

New model

$$r_{xi} = \mu + b_x + b_i + q_i \cdot p_x$$

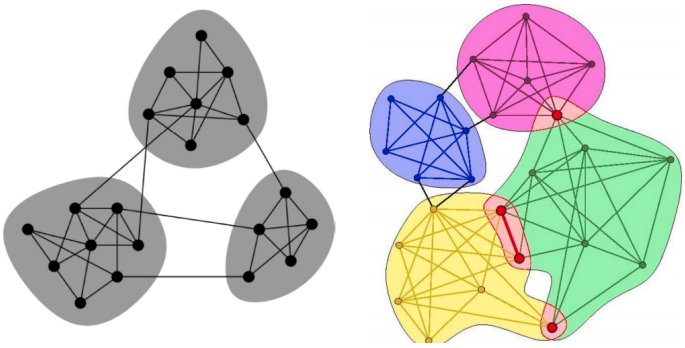
μ: Overall mean rating; b_x: Bias for user ***x***; b_i: Bias for movie ***i***; $q_i \cdot p_x$: User-Movie interaction

Chap 10 Mining Social-Network Graphs

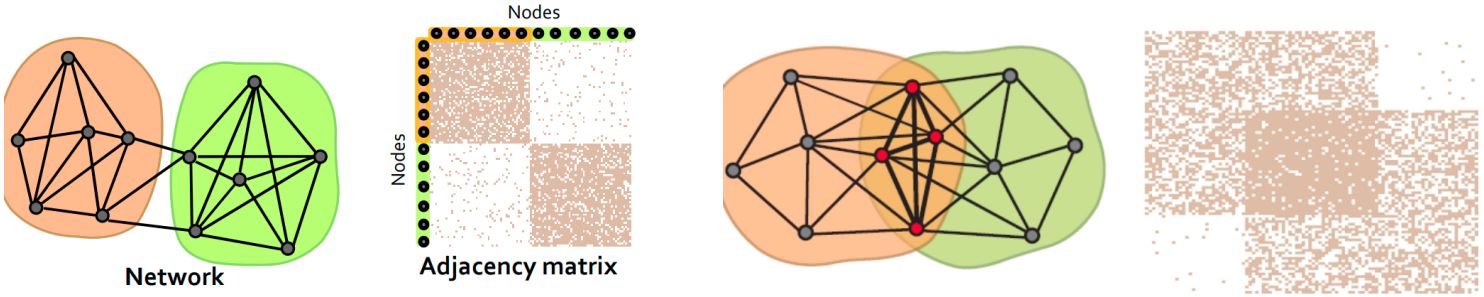
Community Detection in Graphs: Motivation

- Identify node groups in a large connected graphs

Non-overlapping vs. overlapping communities



Represented as adjacency matrix

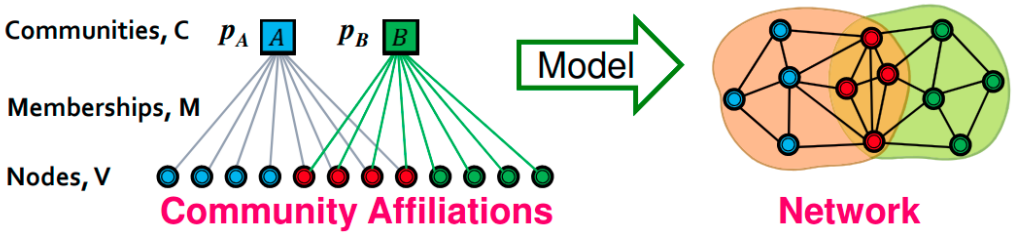


Model of networks

- Goal: define a model that can generate networks
  - the model will have a set of “param.” that we want to estimate (and detect communities)
- issue: given a set of nodes, how do communities “generate” edges of the network?

Community-Affiliation Graph Model (AGM)

- Generative model ***B(V, C, M, {p_c})*** for graphs
  - Nodes ***V***, Communities ***C***, Memberships ***M***
  - Each community ***c*** has a single probability p_c
  - Later we fit the model to networks to detect communities



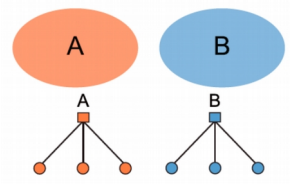
  -

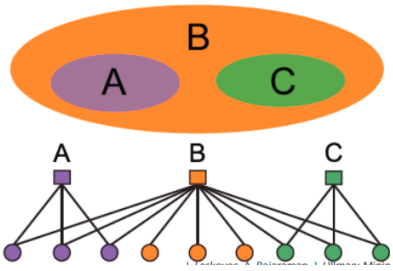
- AGM generates the links
  - for each pair of nodes in community ***A***, we connect them with prob. p_A
  - The overall edge prob. is:
    - $1 - p_c$ = u, v are not in common C; $\prod(1 - p_c)$ = u,v are not in all communities
    - $1 - \prod(1 - p_c)$ = u, v has at least one common community

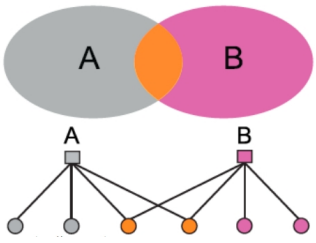
$$P(u,v) = 1 - \prod_{c \in M_u \cap M_v} (1 - p_c)$$

AGM: Flexibility

- AGM can express a variety of community structures
- Non-overlapping →
- Overlapping →
- Nested ↓







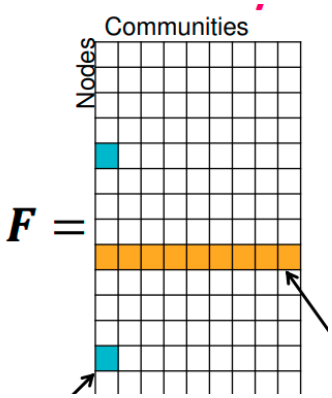
From AGM to BigCLAM (simplify)

- Issues: AGM's structure is too large and bulky, so we use **relaxation**
- Memberships have strengths
- F_{uA}: The membership strength of node u to community A
  - 0 means no membership; large value means active member
- Each community A links nodes independently:

$$P_A(u, v) = 1 - exp(-FuA \cdot FvA)$$

Factor Matrix F

- Recall
- Fu = vector of community membership strengths of u
- Recall Prob. that at least one common community C links the nodes:
  - $P(u, v) = 1 - \prod_C (1 - P_c(u, v))$



Example

$$P(u,v) = 1 - \prod_C \left(1 - P_C(u,v)\right)$$
$$= 1 - \exp\left(-\sum_C F_{uC} \cdot F_{vC}\right)$$
$$= 1 - \exp\left(-F_u \cdot F_v^T\right)$$

- Example F matrix:

F_u:	0	1.2	0	0.2
F_v:	0.5	0	0	0.8
F_w:	0	1.8	1	0

Then: $F_u \cdot F_v^T = 0.16$
And: $P(u,v) = 1 - exp(-0.16) = 0.14$
But: $P(u,w) = 0.88$
$P(v,w) = 0$

BigCLAM: How to find F

- Task: Given a network G(V, E), estimate F

Find F that maximizes the likelihood:

$$arg\ max_F \prod_{(u,v)\in E} P(u,v) \prod_{(u,v)\notin E} (1 - P(u,v))$$

- where: $P(u,v) = 1 - \exp(-F_u \cdot F_v^T)$
- Many times we take the logarithm of the likelihood, and call it log-likelihood: $l(F) = \log P(G|F)$

Goal: Find F that maximizes *l(F)*

$$l(F) = \sum_{(u,v)\in E} \log(1 - \exp(-F_u F_v^T)) - \sum_{(u,v)\notin E} F_u F_v^T$$

BigCLAM: V1.0

- **Compute gradient of a single row F_u of F:**

$$\nabla l(F_u) = \sum_{v\in\mathcal{N}(u)} F_v \frac{\exp(-F_u F_v^T)}{1 - \exp(-F_u F_v^T)} - \sum_{v\notin\mathcal{N}(u)} F_v$$

- **Coordinate gradient ascent:**
  - **Iterate over the rows of F:**
    - Compute gradient $\nabla l(F_u)$ of row u (while keeping others fixed)
    - Update the row F_u: $F_u \leftarrow F_u + \eta\, \nabla l(F_u)$
    - Project F_u back to a non-negative vector: If $F_{uC} < 0$: $F_{uC} = 0$
- **This is slow!** Computing $\nabla l(F_u)$ takes linear time!

$\mathcal{N}(u)$.. Set out outgoing neighbors

BigCLAM: V2.0

However, we notice:

$$\sum_{v\notin\mathcal{N}(u)} F_v = (\sum_{v} F_v - F_u - \sum_{v\in\mathcal{N}(u)} F_v)$$

- We cache $\sum_v F_v$
- So, computing $\sum_{v\notin\mathcal{N}(u)} F_v$ now takes **linear time** in the degree $|\mathcal{N}(u)|$ of u

Analysis of Large Graphs: Detecting Clusters

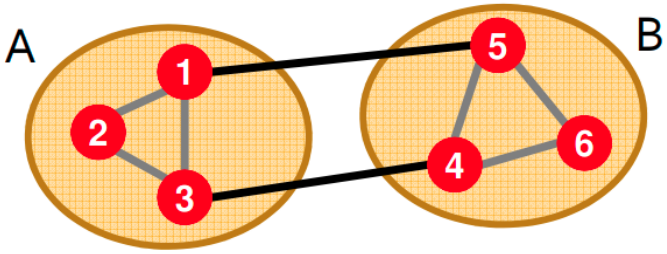
Graph Partitioning

- Criterion
  - Maximize # of within-group connections
  - Minimize # of between-group connections

Graph Cuts

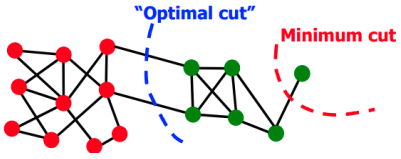
- Cut: set of edges with only one vertex in a group
- E.g cut(A, B) = 2

$$cut(A,B) = \sum_{i \in A, j \in B} w_{ij}$$



Group Cut: Criterion

- Minimum-cut
  - problem: does not consider internal cluster connectivity
- **Normalized-cut**
  - vol(A): total weight of the edges with at least one endpoint in A



$$ncut(A,B) = \frac{cut(A,B)}{vol(A)} + \frac{cut(A,B)}{vol(B)}$$

However, computing optimal cut is NP-hard

Spectral Graph Partitioning

- **A**: adjacency matrix of undirected **G**
  - $A_{ij} = 1$ if (i, j) is an edge, else 0
- **x** is a vector in R^n with components $(x_1, \ldots, x_n)$
- Entry y_i is a sum of labels x_j of neighbors of I

$$\begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & & \vdots \\ a_{n1} & \cdots & a_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix} \qquad y_i = \sum_{j=1}^{n} A_{ij} xj = \sum_{(i,j)\in E} x_j$$

Spectral Graph Theory

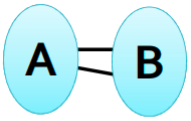
- Analyze the "spectrum" of matrix representing G
- Spectrum: Eigenvectors x_i of a graph, ordered by the magnitude (strength) of their corresponding eigenvalues λ_i
- *j-th* coord. Of $A \cdot x$ = sum of the x-values of neighbors of j

$$\begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & & \vdots \\ a_{n1} & \cdots & a_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} = \lambda \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}$$

d-regular graph

Connected case

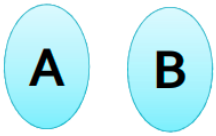
- Suppose all nodes in G have degree (#edges) d and G is connected
- Intuitively, we found that $\lambda = d$ if x = (1, 1, …, 1)



$\lambda_n - \lambda_{n-1} \approx 0$

Not-connected case

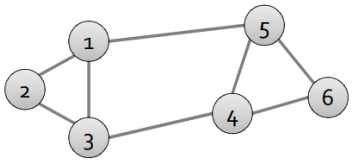
- Suppose G has 2 components
- Let put all 1s on A and 0s on B (or vice versa)
- We also found that $\lambda = d$



$\lambda_n = \lambda_{n-1}$

$$x' = (\underbrace{1, \ldots, 1}_{|A|}, \underbrace{0, \ldots, 0}_{|B|}) \text{ then } A \cdot x' = (d, \ldots, d, 0, \ldots, 0)$$
$$x'' = (0, \ldots, 0, 1, \ldots, 1) \text{ then } A \cdot x'' = (0, \ldots, 0, d, \ldots, d)$$

Matrix Representations



Adjacency matrix (*A*)

- n x n matrix
- ***A* = [a_{ij}], a_{ij} = *1*** if (i, j) is an edge
- A is symmetrix matrix
- Eigenvectors are real and orthogonal

	1	2	3	4	5	6
1	0	1	1	0	1	0
2	1	0	1	0	0	0
3	1	1	0	1	0	0
4	0	0	1	0	1	1
5	1	0	0	1	0	1
6	0	0	0	1	1	0

Degree matrix (*D*)

- n x n diagonal matrix
- ***D*=[d_{ii}], d_{ii}** = degree of node ***I***

	1	2	3	4	5	6
1	3	0	0	0	0	0
2	0	2	0	0	0	0
3	0	0	3	0	0	0
4	0	0	0	3	0	0
5	0	0	0	0	3	0
6	0	0	0	0	0	2

Laplacian matrix (*L*)

- n x n symmetric matrix
- L = D – A
  - -1 means a connected edge
- Eigenvalues are non-negative real numbers
- Eigenvectors are real and orthogonal

	1	2	3	4	5	6
1	3	-1	-1	0	-1	0
2	-1	2	-1	0	0	0
3	-1	-1	3	-1	0	0
4	0	0	-1	3	-1	-1
5	-1	0	0	-1	3	-1
6	0	0	0	-1	-1	2

Finding a Partition

Fact: For symmetric matrix M:

$$\lambda_2 = \min_x \frac{x^T M\ x}{x^T x}$$

What is the meaning of min $x^T L\, x$ on G?

- $x^T L\, x = \sum_{i,j=1}^{n} L_{ij}\, x_i x_j = \sum_{i,j=1}^{n} (D_{ij} - A_{ij})\, x_i x_j$
- $= \sum_i D_{ii} x_i^2 - \sum_{(i,j)\in E} 2 x_i x_j$
- $= \sum_{(i,j)\in E} (x_i^2 + x_j^2 - 2 x_i x_j) = \sum_{(i,j)\in E} (x_i - x_j)^2$

Node i has degree d_i. So, value x_i^2 needs to be summed up d_i times.
But each edge (i,j) has two endpoints so we need $x_i^2 + x_j^2$

- L = D - A
- $D_{ij}x_i x_j = D_{ii}x_i^2$ (because D is diagonal matrix)
- $A_{ij}x_i x_j = 2x_i x_j$ (because $\boldsymbol{A}\cdot\boldsymbol{x}$ = sum of values of neighbors; and we have x_i and x_j, so the twice)
- $D_{ii}x_i^2$ = degree * x_i^2 = summation over all the edges = $x_i^2 + x_j^2$

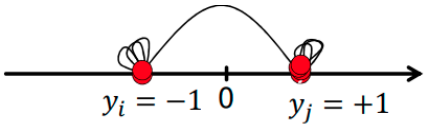
λ_2 as optimization problem

What else do we know about x?

- x is unit vector: $\sum_i x_i^2 = 1$
- x is orthogonal to 1^{st} eigenvector $(1, \ldots, 1)$ thus: $\sum_i x_i \cdot 1 = \sum_i x_i = 0$

Find Optimal Cut [Fiedler '73]

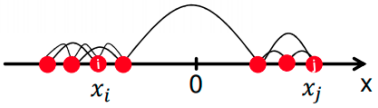
- Express partition (A, B) as a vector
  - y_i = +1 if i in A, -1 if i is B
- issue: cannot be solve exactly
- solution: relax *y* to use real value



Raleigh Theorem

- Instead of categorize into +1/-1, we use real value
- we want (1) small span, and (2) connected node on the same side

$$\min_{y \in \Re^n} f(y) = \sum_{(i,j) \in E} (y_i - y_j)^2 = y^T L y$$



- $\lambda_2 = \min_y f(y)$: The minimum value of $f(y)$ is given by the 2nd smallest eigenvalue λ_2 of the Laplacian matrix L
- $x = \arg\min_y f(y)$: The optimal solution for y is given by the corresponding eigenvector x, referred as the **Fiedler vector**

Spectral Clustering (3 stages)

1. Pre-processing
    1. construct a matrix representation of the graph
2. Decomposition
    1. Compute eigenvalues and eigenvectors of the matrix
    2. Map each point to a lower-dimensional representation based on one or more eigenvectors
        - recall λ_2 problem
3. Grouping
    1. Assign points to two or more clusters, based on the new representation

Spectral Partitioning

- Naive approaches
    - split at 0 or median value
- More expensive appaoaches
    - minimize normalized cut in 1-dimension

k-Way Spectral Clustering

- Recursive bi-partitioning
    - recursively apply bi-partitioning algorithm in a hierarchical divisive manner
    - con: inefficient, unstable
- Cluster multiple eigenvectors
    - build a reduced space from multiple eigenvectors
    - commonly used in recent papers