Mining of Massive Datasets

Martin Takáč

ISE 395/495, Fall 2016, Finding Similar Items; Locality-Sensitive Hashing

September 26, 2016

LSH!

Outline

Outline

Finding similar text documents

- Consider a large collections of articles (news, wikipedia, books, thesis, emails, spam)
- we want to find a pair of items which are very similar why?
- assume we have some functions, which measure the similarity between two items x_i, x_j:

$$sim(x_i, x_j) \in [0, 1]$$

whereas small value implies that they are NOT very similar.

Some easy calculations

- we have $N = 1,000,000$ articles
- we have 1000 computers
- computing similarity between to articles takes 0.01 seconds

How long does it take to find all pairs with similarity measure ≥ 0.95?

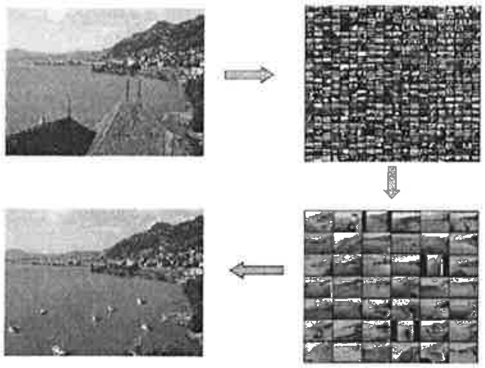
Answer - Naïve approach:

- we have to compute the similarity between each pair
- we have $\approx \frac{n^2}{2} = 5 \times 10^{11}$ pairs
- in 1 second we can check $100 \times 1000 = 10^5$ pairs
- we need $5 \times 10^{11} \times 10^{-5} = 5 \times 10^6$ seconds, which is 58 days!
- Changing N to $10M$ would lead to computation time of 5800 days $\approx$ 15 years

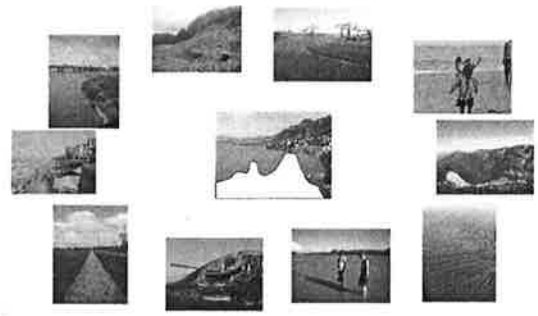
This lecture

- the main issue with the previous algorithm is that we need $O(n^2)$ comparisons
- in this lecture we show how it can be done in $O(n)$! yes, there is no n^2!
- but.....
  - the result is probabilistic LSH
  - it means that we will find "almost" all pairs which we want (and maybe few which we do not want – but we can exclude them)
  - however, we can process 10M articles in just few minutes!

Solution [Hays and Efros, SIGGRAPH 2007]

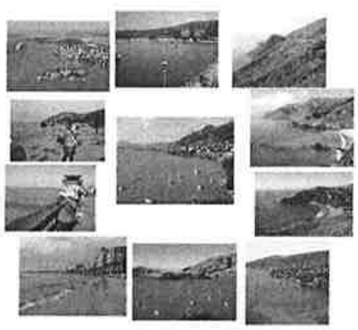


Scene Compl. Problem [Hays and Efros, SIGGRAPH 2007]



10 nearest neighbours from a collection of 20,000 images

Scene Compl. Problem [Hays and Efros, SIGGRAPH 2007]



10 nearest neighbours from a collection of 2,000,000 images

Outline

Euclidean vs. NonEuclidean spaces

Euclidean space

If X is Euclidean space (R^n), then for any two elements $x, y \in R^n$ we can define the distance

$$d(x,y) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$$

There are more measures, e.g. cosine distance (angle between vectors), Manhattan distance ($\|\cdot\|_1$) and many others

Euclidean vs. NonEuclidean spaces

NonEuclidean space

Now, imagine that we have set of words D (D=Dictionary).
And we have two sets $A, B \subseteq D$.
How to define a "distance" measure between A and B?

Hash Table

- Hash functions are primarily used in hash tables, to quickly locate a data record (e.g., a dictionary definition) given its search key (the headword).
- Specifically, the hash function is used to map the search key to an index;
- the index gives the place in the hash table where the corresponding record should be stored.

Python dictionary is basically a hash table!
The **key** is hashed and based on the value of hash is stored at a specific location. Therefore **look-up** in hash tables are **fast!**

Easy example

- Imagine that we want to hash positive integers.
- A natural and simple hash function is

$$h(x) = x \bmod B$$

(i.e. reminder when x is divided by B).
- If x are completely random values, then $h(x)$ will be equally likely to be any number in $\{0, \dots, B-1\}$.
- However, if our numbers would be just **even** numbers and $B = 10$ then this hash function would have very poor performance
- easy fix, choose $B = 11$ (very often we are choosing primes) B.

in case x is not uniform distribution.

What if data are not integers?

- In a sense, all data types have values that are composed of bits, and sequences of bits can always be interpreted as integers.
- However, there are some simple rules that enable us to convert common types to integers.
- For example, if x are strings, convert each character to its ASCII or Unicode equivalent, which can be interpreted as a small integer.
- Sum the integers (maybe with some weights) before dividing by B.

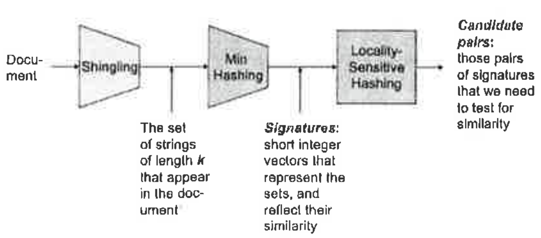
Simple example

- Assume that our input is always a string of length $n = 10$.
- We choose n large random primes $p_1, \dots, p_n$.
- we represent every character c_i $i \in \{1, \dots, n\}$ by its ASCII code **use command ord('a') in python** to find out ASCII code of letter 'a'
- define h as follows

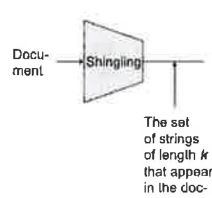
$$h(s) = \sum_{i=1}^{n} c_i p_i \bmod B$$

where B is some reasonable big prime number (but not bigger then maximal possible value of $\sum_{i=1}^{n} c_i p_i$)!

The Big Picture



Step 1: Shingling



Encoding Sets as Bit Vectors

- Many similarity problems can be formalized as finding subsets that have significant intersection
- Encode sets using 0/1 (bit, boolean) vectors
- One dimension per element in the universal set (e.g. set of hashes of k-shingle)
- Interpret set intersection as bitwise AND, and set union as bitwise OR
- **Example:** $C_1 = 10111$; $C_2 = 10011$
- Size of intersection = 3; size of union = 4, Jaccard similarity (not distance) = 3/4 Distance: d(C_1,C_2) = 1 - (Jaccard similarity) = 1/4

From Sets to Boolean Matrices

- **Rows** = elements (shingles)
- **Columns** = sets (documents)
- 1 in row e and column s if and only if e is a member of s
- Column similarity is the Jaccard similarity of the corresponding sets (rows with value 1)
- **Typical matrix is very sparse!**
- Each document is a column:
- **Example:** sim(C_1 ,C_2) = ?
  Size of intersection = 3;
  size of union = 6,
  Jaccard similarity = 3/6
  Jaccard dis: d(C_1,C_2) = 3/6

Documents (columns) / Shingles (rows)

1	1	1	0
1	1	0	1
0	1	0	1
0	0	0	1
1	0	0	1
1	1	1	0
1	0	1	0

Outline: Finding Similar Columns

- **So far:**
  - Documents ⇒ Sets of shingles
  - Represent sets as boolean vectors in a matrix
- **Next goal:** Find similar columns while computing small signatures **why do we need that???**
- **We wish that it will hold that** :
  Similarity of columns == similarity of signatures

Outline: Finding Similar Columns

- **Next goal:** Find similar columns; small signatures
- **Naïve approach**
  1. Signatures of columns: small summaries of columns
  2. Examine pairs of signatures to find similar columns
     Essential: Similarities of signatures and columns are related
  3. Optional: Check that columns with similar signatures are really similar
- **Warning!!!** :
  - Comparing all pairs may take too much time:
  - Job for LSH
  - These methods can produce **false negatives** , and even **false positives** (if the optional check is not made)

Hashing Columns (obtain Signatures)

- **Key idea:** "hash" each column C to a small signature $h(C)$, such that:
  1. $h(C)$ is small enough that the signature fits in RAM
  2. sim(C_1, C_2) is the same as the "similarity" of signatures $h(C_1)$ and $h(C_2)$

Goal: Find a hash function h() such that

- If sim(C_1,C_2) is high, then with high prob. $h(C_1) = h(C_2)$
- If $sim(C_1, C_2)$ is low, then with high prob. $h(C_1) \neq h(C_2)$

Hash docs into buckets. Expect that "most" pairs of near duplicate docs hash into the same bucket!

Min-Hashing

Goal: Find a hash function h() such that

- If sim(C_1,C_2) is high, then with high prob. $h(C_1) = h(C_2)$
- If $sim(C_1, C_2)$ is low, then with high prob. $h(C_1) \neq h(C_2)$

- **Observation:** the hash function has to depend on the similarity metric!
- Not all similarity metrics have a suitable hash function
- **There is a suitable hash function for the Jaccard similarity:** It is called Min-Hashing

Min-Hash signatures

- Pick **K=100 random permutations of the rows**
- Think of $sig(C)$ as a columns vector
- i-th coordinate of $sig(C)$ is the index of the first row that has 1 in column C according of i-th permutation
- Note: the sketch (signature) of a document C is small $100 numbers$!
- **We have achieved our goal!** We have "compressed" very long bit vectors into short signatures!

Min-Hash signatures

A bit of calculations

- Input data: 1M articles, each having maybe 100kB, in total 97GBs
- We find 10-shingles. However, we hash them into $B = 10^6$ buckets. We assume 0.01% of sparsity. Total storage needed

$$B \times 0.0001 \times 10^6 \times 33 bits = 0.38GB$$

- we choose 100 hash functions. The signature matrix has

$$100 \times 10^6 bytes = 0.093GBs \approx \mathbf{95MBs}$$

(more realistic: each number needs 4 bytes)

$$100 \times 10^6 \times 4 bytes = 0.372GBs$$

Implementation Trick

- **Permuting rows even once is prohibitive!**
- we can use **row hashing!**
  - Pick $K = 100$ has functions k_i
  - Ordering under k_i gives a random row permutation!
- One-pass over the data implementation
  - For each column C and hash-function k_i, keep a "slot" for the min-hash value
  - initialize all $sig[C](i) = \infty$
  - Scan rows of C looking for 1s
    - suppose row j has 1 in column C
    - then for each k_i: if $k_i(j) < sig(C)[i]$ then $sig(C)[i] = k_i(j)$.

How to choose random hash function?

Universal hashing:

$$h_{a,b}(x) = ((ax + b) \ mod \ p) \ mod \ N$$

where a,b are random integers, p is some prime $p > N$

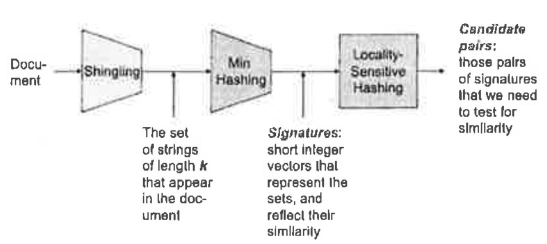
Exercise

- in this "lab" we will implement first 2 steps of LHS
- we will be using smaller dataset in a class

you should modify file

`LSH_Part1_Signatures.py`

Step 3: Locality Sensitive Hashing



So far, we still would have to process all N^2 pairs. We would like to explore only those which should be similar to each other!

LSH: First Cut

- **Goal:** Find documents with Jaccard similarity at least s (for some similarity threshold, e.g. $s = 0.8$)
- LSH - **General idea:** Use a function $f(x, y)$ that tells whether x and y is a **candidate pair** : a pair of elements whose similarity must be evaluated
- **For Min-Hash matrices:**
  - hash columns of signature matrix M to many buckets
  - each pair of documents that hashes into the same bucket is a **candidate pair**

Example of Bands

Assume the following case:

- Suppose 100,000 columns of M (100k docs)
- Signatures of 100 integers (rows)
- Therefore, signatures take

$$10^5 \cdot 100 \cdot 4b \approx 4MBs$$

- Choose b = 20 bands of r = 5 integers/band

Goal: Find pairs of documents that are at least s = 0.8 similar

C_1, C_2 are 80% similar

- Find pairs of $\geq s = 0.8$ similarity, set $b = 20, r = 5$
- Assume $sim(C_1, C_2) = 0.8$
- Since $sim(C_1, C_2) \geq s$ we want C_1, C_2 to be a **candidate pair**
  We want them to hash to at **least 1 common bucket**
- Probability C_1, C_2 identical in one particular band: $(0.8)^5 = 0.328$
- What is the probability that C_1, C_2 are **NOT** similar in all of the 20 bands?: **Answer:** $(1 - 0.328)^{20} = 0.00035$
  i.e., about 1/3000th of the 80%-similar column pairs are **false negatives** (we miss them)
  We would find 99.965% pairs of truly similar documents

C_1, C_2 are 30% similar

- Find pairs of $\geq s = 0.8$ similarity, set $b = 20, r = 5$
- Assume $sim(C_1, C_2) = 0.3$
- Since $sim(C_1, C_2) < s$ we want C_1, C_2 to hash to **NO common buckets** (all bands should be different)
- Probability C_1, C_2 identical in one particular band: $(0.3)^5 = 0.00243$
- What is the probability that C_1, C_2 are identical in at least 1 of 20 bands?: **Answer:** $1 - (1 - 0.00243)^{20} = 0.0474$
  In other words, approximately 4.74% pairs of docs with similarity 30% end up becoming **candidate pairs**
  they are **false positives** since we will have to examine them (they are candidate pairs) but then it will turn out their similarity is below threshold s

LSH Involves a Trade-off

Pick

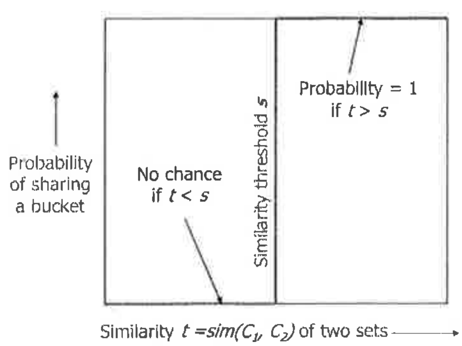
- The number of Min-Hashes (rows of M)
- The number of bands b, and
- The number of rows r per band

to balance false positives/negatives

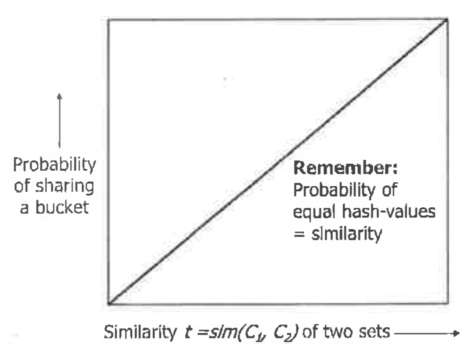
Example

If we had only 15 bands of 5 rows, the number of false positives would go down, but the number of false negatives would go up

Analysis of LSH - What we want



What 1 Band of 1 Row Gives You



Exercise

- we will finish the 3rd step

you should modify file

`LSH_Part2_Hashing.py`

Resources

- `http://spark.apache.org/`
- `https://en.wikipedia.org/wiki/MapReduce`
- `http://mmds.org/`

Mining of Massive Datasets

Martin Takáč

ISE 395/495, Fall 2016, Recommender Systems: Content-based Systems & Collaborative Filtering

October 10, 2016

Recommender System

Outline

1 Motivation
(1) 2 Content-based Recommender System
(2) 3 Collaborative Filtering
4 Remarks & Practical Tips

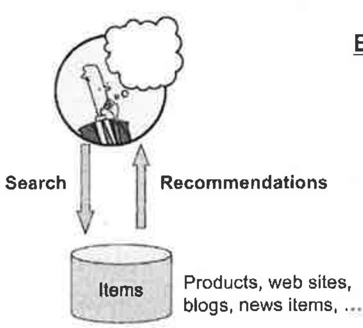
Outline

1 Motivation
2 Content-based Recommender System
3 Collaborative Filtering
4 Remarks & Practical Tips

Example: Recommender System

- **Customer X**
  - Buys Metallica CD
  - Buys Megadeth CD
- **Customer Y**
  - Does search on Metallica
  - Recommender system suggests Megadeth from data collected about customer **X**

Recommendations



Examples:
amazon.com, Pandora, StumbleUpon, del.icio.us, Netflix, movielens (helping you find the right movies), last.fm, Google News, YouTube, XBOX LIVE

From Scarcity to Abundance

- Shelf space is a **scarce commodity** for traditional retailers
- Also: TV networks, movie theaters,...
- Web enables **near-zero-cost dissemination of information about products**
  From scarcity to abundance
- **More choice necessitates better filters** Why?
  - Recommendation engines
  - How **Into Thin Air** made **Touching the Void** a bestseller: http://www.wired.com/wired/archive/12.10/tail.html

1. Gathering Ratings

- **Explicit**
  - Ask people to rate items
  - Doesn't work well in practice Why? – people can't be bothered
- **Implicit**
  - Learn ratings from user actions
  - E.g., purchase implies high rating
  - What about low ratings? how we can implicitly get it?

Extrapolating Utilities

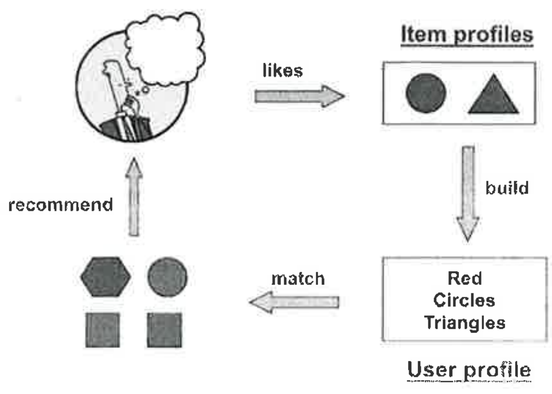
- **Key problem:** Utility matrix U is sparse
  - Most people have not rated most items
  - **Cold start:**
    New items have no ratings
    New users have no history **How does Netflix handle it?**
- **Three approaches to recommender systems:**
  - Content-based (Today!)
  - Collaborative (Today!)
  - Latent factor based

Outline

Content-based Recommender System

- **Main idea:** Recommend items to customer x similar to previous items rated highly by x
- **Example**
  - **Movie recommendations**
    - Recommend movies with same actor(s), director, genre, ...
  - **Websites, blogs, news**
    - Recommend other sites with "similar" content

Plan of Action



Item Profiles

- **For each item, create an item profile**
- **Profile is a set (vector) of features** of item
  - Movies: author, title, actor, director,...
  - Text: Set of "important" words in document

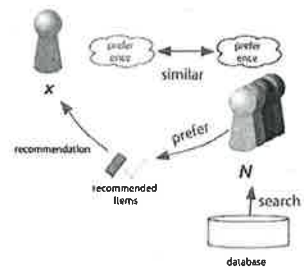
  How do we represent a profile? usually a binary vector (example: movies)
- **How to pick important features?**
  - Usual heuristic from text mining is TF-IDF (Term frequency * Inverse Doc Frequency)
    Term ... Feature
    Document ... Item

Collaborative Filtering

Harnessing quality judgements of other users

Collaborative Filtering

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



Finding "Similar" Users

- Let r_x be the vector of user x's ratings
- **Jaccard similarity measure**
  - Problem: Ignores the value of rating
- **Cosine similarity measure**
  - $$sim(x, y) = cos(r_x, r_y) = \frac{\langle r_x, r_y \rangle}{\|r_x\|\|r_y\|}$$
  - Problem: Treats missing ratings as "negative"
- **Pearson correlation coefficient**
  - $S_{x,y}$ - items rated by both users x and y

$$sim(x, y) = \frac{\sum_{s \in S_{x,y}} (r_{x,s} - \bar{r}_x)(r_{y,s} - \bar{r}_y)}{\sqrt{\sum_{s \in S_{x,y}} (r_{x,s} - \bar{r}_x)^2} \sqrt{\sum_{s \in S_{x,y}} (r_{y,s} - \bar{r}_y)^2}}$$

$\bar{r}_x$, $\bar{r}_y$ - average ratings

Finding "Similar" Users: Example

- $r_x = [*, ?, ?, *, ***]$
  $r_y = [*, ?, **, **, ?]$
- **Jaccard similarity measure** r_x, r_y as sets:
  $r_x = \{1, 4, 5\}$
  $r_y = \{1, 3, 4\}$
  sim(x,y)=?
- **Cosine similarity measure** r_x, r_y as points:
  $r_x = [1, 0, 0, 1, 3]$
  $r_y = [1, 0, 2, 2, 0]$
  sim(x,y)=?
- **Pearson correlation coefficient**

$$sim(x, y) = \frac{\sum_{s \in S_{x,y}} (r_{x,s} - \bar{r}_x)(r_{y,s} - \bar{r}_y)}{\sqrt{\sum_{s \in S_{x,y}} (r_{x,s} - \bar{r}_x)^2} \sqrt{\sum_{s \in S_{x,y}} (r_{y,s} - \bar{r}_y)^2}}$$

sim(x,y)=?

Similarity Metric

	HP1	HP2	HP3	TW	SW1	SW2	SW3
A	4			5	1		
B	5	5	4				
C				2	4	5	
D		3					3

Which user is more similar to A?

- **Intuitively we want:** $sim(A, B) > sim(A, C)$
- **Jaccard similarity:** $1/5 < 2/4$
- Cosine similarity: $0.386 > 0.322$
  - Considers missing ratings as "negative"
  - solution: subtract the row mean

	HP1	HP2	HP3	TW	SW1	SW2	SW3
A	2/3			5/3	−7/3		
B	1/3	1/3	−2/3				
C				−5/3	1/3	4/3	
D		0					0

$0.092 > -0.559$

Notice that cosine similarity is correlation when data is centred at 0

Rating Predictions

From similarity metric to recommendations:

- Let r_x be the vector of user x's ratings
- Let N be the set of k users most similar to x who have rated item i
- **Prediction for item s of user x:**
  - $$r_{x,i} = \frac{1}{k} \sum_{y \in N} r_{y,i}$$
  - $$r_{x,i} = \frac{\sum_{y \in N} s_{x,y} r_{y,i}}{\sum_{y \in N} s_{x,y}}$$

    where $s_{x,y} = sim(x, y)$
  - Other options?
- **Many other tricks possible...**

CF: Common Practise

- Define similarity $s_{i,j}$ of items i and j
- select k nearest neighbors $N(i;x)$
  - items most similar to i, that were rated by x
- **Estimate rating $r_{x,i}$ as the weighted average**

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

where $b_{x,i} = \mu + b_x + b_i$ is the **baseline estimate**
- μ = overall mean movie rating
- b_x = rating deviation of user x = (average rating of user x) - μ
- b_i = rating deviation of movie i

Why this makes sense?

Item-Item vs. User-User

	Avatar	LOTR	Matrix	Pirates
Alice	1		0.8	
Bob		0.5		0.3
Carol	0.9		1	0.8
David			1	0.4

- **In practice, it has been observed that item-item often works better than user-user**
- Why?
- Items are simpler, users have multiple tastes

Pros/Cons of Collaborative Filtering

- **+ Works for any kind of item**
  - No feature selection needed
- - Cold Start:
  - Need enough users in the system to find a match
- - Sparsity:
  - The user/ratings matrix is sparse
  - Hard to find users that have rated the same items
    What can we do about that?
    Clustering
- - First rater:
  - Cannot recommend an item that has not been previously rated
  - New items, Esoteric items
- - Popularity bias:
  - Cannot recommend items to someone with unique taste
  - Tends to recommend popular items

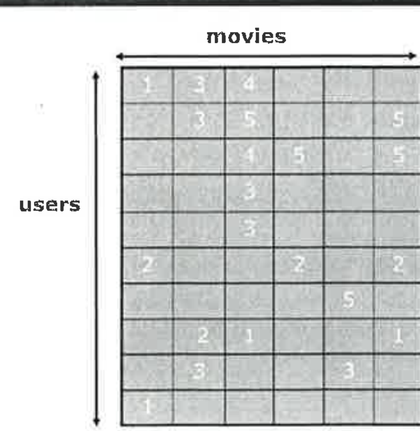
Solution: Hybrid Methods

- **Implement two or more different recommenders and combine predictions**
  - Perhaps using a linear model
- Add content-based methods to collaborative filtering
  - Item profiles for new item problem
  - Demographics to deal with new user problem

Outline

1 Motivation
2 Content-based Recommender System
3 Collaborative Filtering
4 Remarks & Practical Tips
- Evaluation
- Error metrics
- Complexity/Speed

Evaluation



Mining of Massive Datasets

Martin Takáč

ISE 395/495, Fall 2016, Recommender Systems: Latent Factor Models

October 10, 2016

Outline

Projects

- counts 20% of your final grade
- create groups of 4 (tell to my TA by the end of this week, better asap)
- each group has to have a **group leader**
- try to think about the topic, why you are interested in this topic, what kind of questions you want to answer
- if you dare, try `https://www.kaggle.com/competitions`
- it is better to "fail" when doing something hard, then do an easy project perfectly

Outline

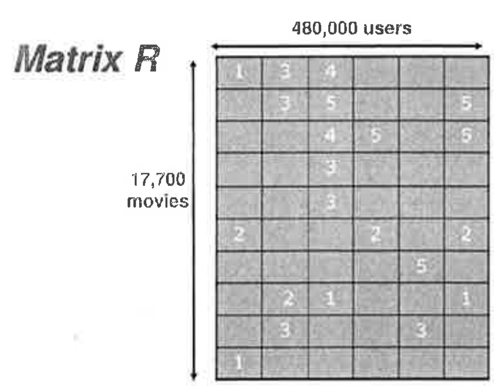
The Netflix Prize

- Training data
  - 100 million ratings, 480,000 users, 17,770 movies
  - 6 years of data: 2000-2005
- Test data
  - Last few ratings of each user (2.8 million)
  - Evaluation criterion: Root Mean Square Error (RMSE)

$$\frac{1}{|R|}\sqrt{\sum_{(i,x)\in R}(\hat{r}_{x,i}-r_{x,i})^2}$$

  - Netflix's system RMSE: 0.9514
- **Competition:**
  - 2,700+ teams
  - \$1 million prize for 10% improvement on Netflix

The Netflix Utility Matrix R



Idea: Interpolation Weights $w_{i,j}$

- Use a weighted sum rather than weighted avg.:

$$\hat{r}_{x,i} = b_{x,i} + \sum_{j \in N(i,x)} w_{i,j}(r_{x,j} - b_{x,j})$$

- A few notes:
  - $N(i, x)$ is set of movies rated by user x that are similar to movie i
  - $w_{i,j}$ is the interpolation weight (some real number)
    We allow: $\sum_{j \in N(i,x)} w_{i,j} \neq 1$
  - $w_{i,j}$ models **interaction between pairs of movies** (it does not depend on user x)

Idea: Interpolation Weights $w_{i,j}$

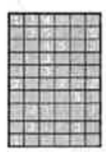
$$\hat{r}_{x,i} = b_{x,i} + \sum_{j \in N(i,x)} w_{i,j}(r_{x,j} - b_{x,j})$$

- How to set $w_{i,j}$?
  - Remember, error metric is $\frac{1}{|R|}\sqrt{\sum_{(i,x) \in R}(\hat{r}_{x,i} - r_{x,i})^2}$
  - or equivalently $SSE = \sum_{(i,x) \in R}(\hat{r}_{x,i} - r_{x,i})^2$
  - Find $w_{i,j}$ that minimize SSE on **training data** !
  - $w_{i,j}$ 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
  - **Why is this a good idea?**

Recommendations via Optimization

Goal: Make good recommendations

- Quantify goodness using RMSE:
- **Lower RMSE → better recommendations**
- Want to make good recommendations on items that user has not yet seen.
- Can't really do this! **Why?**
- **Let's build a system such that it works well on known (user, item) ratings**
- And **hope** the system will also predict well the **unknown ratings**



Outline

Recommendations via Optimization

- Idea: **Let's set values w such that they work well on known (user, item) ratings**
- **How to find such values w?**
- Idea: **Define an objective function and solve the optimization problem**
- Find w_{ij} that minimize SSE on training data!

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

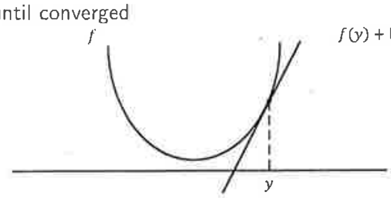
- think of w as a vector of numbers

Detour: Minimizing a function

- Assume we have a function $f : R^n \to R$
- **How can we minimize it?**
- Start at some point y
- Compute the derivative $\nabla f(y)$
- Make a step in the reverse direction of the gradient

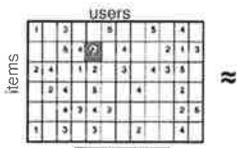
$$y = y - \nabla f(y)$$

- repeat until converged



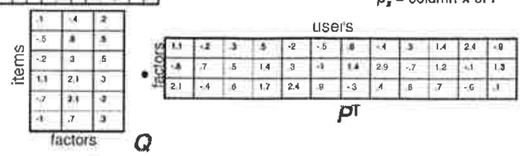
Ratings as Products of Factors

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



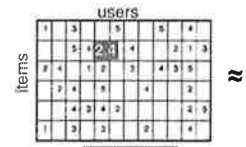
$$\hat{r}_{xi} = q_i \cdot p_x$$
$$= \sum_f q_{if} \cdot p_{xf}$$

q_i = row i of Q
p_x = column x of P^T



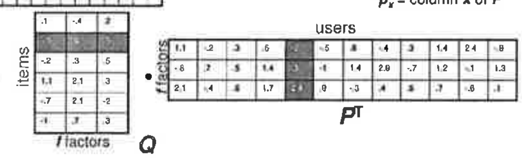
Ratings as Products of Factors

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

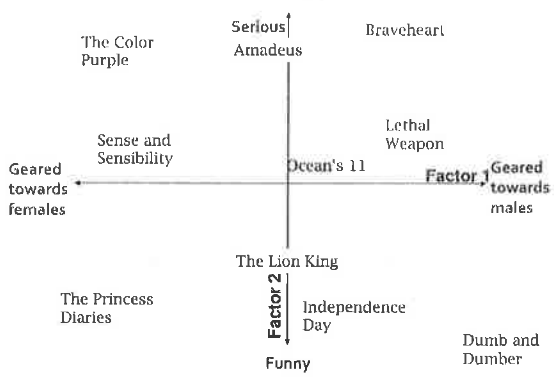


$$\hat{r}_{xi} = q_i \cdot p_x$$
$$= \sum_f q_{if} \cdot p_{xf}$$

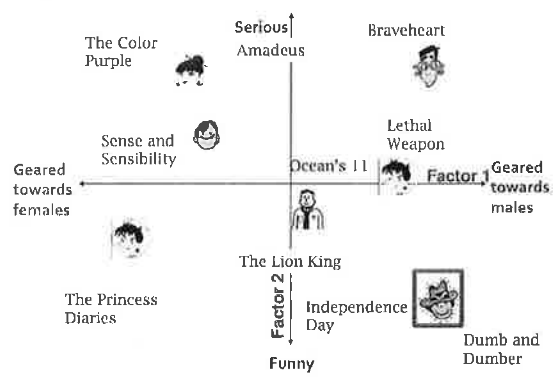
q_i = row i of Q
p_x = column x of P^T



Latent Factor Models



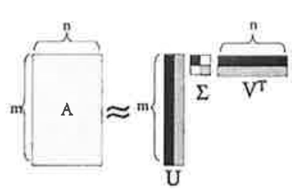
Latent Factor Models



Recap: SVD

Remember SVD:

- A: Input data matrix
- U: Left singular vecs
- V: Right singular vecs
- Σ: Singular values



- **So in our case:**
- SVD on Netflix data: $R \approx Q \cdot P^T$
- $A = R$, $Q = U$, $P^T = \Sigma V^T$

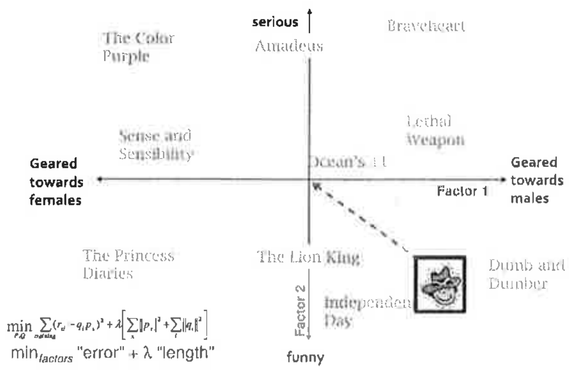
SVD: More good stuff

- We already know that SVD gives minimum reconstruction error (Sum of Squared Errors):

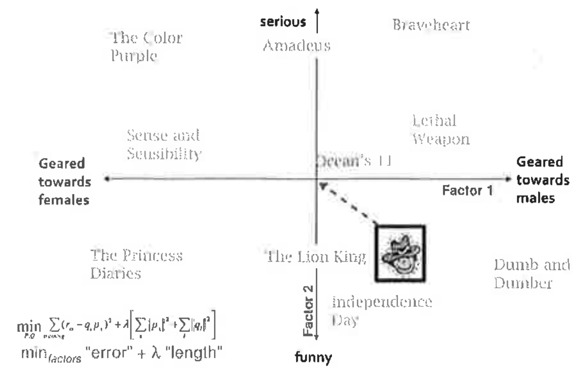
$$\min_{U,V,\Sigma} \sum_{ij} (A_{ij} - [U \Sigma V^T]_{ij})^2$$

- Note two things
  - SSE and RMSE are monotonically related:
  - $RMSE = \frac{1}{c}\sqrt{SSE}$ - Great news: **SVD is minimizing RMSE are we done?**
  - Complication: The sum in SVD error term is over all entries (no-rating in interpreted as zero-rating).
  - **But our R has missing entries!**

The effect of regularization



The effect of regularization



Outline

1. From History: The Netflix Prize
2. Local and Global Effects
3. Optimization
4. Latent Factor Models
5. **Stochastic Gradient Descent**
6. Extending Latent Factor Model to Include Biases
7. Spark

Stochastic Gradient Descent

- **Want to find matrices P and Q**

$$\min_{P,Q} \sum (r_{x,i} - q_i \cdot p_x)^2 + \left[\lambda_1 \sum_x \|p_x\|^2 + \lambda_2 \sum_i \|q_i\|^2\right]$$

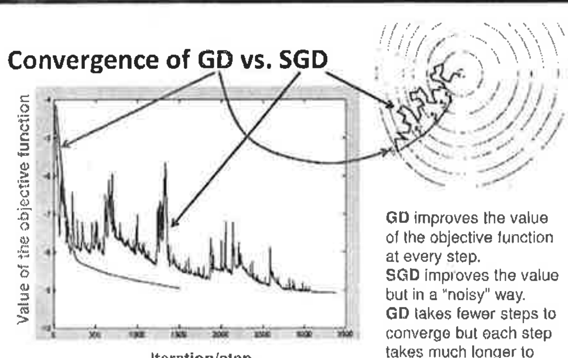
- Gradient Descent
  - Initialize P and Q (using SVD, pretend missing ratings are 0)
  - Do gradient descent
    - $P = P - \eta \nabla P$
    - $Q = Q - \eta \nabla Q$
    - where ∇Q is gradient/derivative of matrix Q
    - $\nabla Q = [\nabla q_{i,f}]$, where $\nabla q_{i,f} = \sum_{x,i} -2(r_{x,i} - q_i p_x) p_{x,f} + 2\lambda_2 q_{i,f}$
  - Observation: **Computing gradients is slow!**

Stochastic Gradient Descent

- **Gradient Descent** (GD) vs. **Stochastic** GD
- Observation $\nabla Q = [\nabla q_{i,f}]$,
- where $\nabla q_{i,f} = \sum_{x,i} -2(r_{x,i} - q_i p_x) p_{x,f} + 2\lambda_2 q_{i,f} = \sum_{x,i} \nabla Q(r_{x,i})$
- Gradient Descent: $Q = Q - \eta \nabla Q = Q - \eta \sum_{x,i} \nabla Q(r_{x,i})$
- **Idea:** Instead of evaluating gradient over all ratings evaluate it for each individual rating and make a step
  - *GD*: $Q = Q - \eta \sum_{x,i} \nabla Q(r_{x,i})$
  - *SGD*: $Q = Q - \mu \nabla Q(r_{x,i})$
    - Faster convergence!
    - Need more steps but each iteration is much much faster!

SGD vs. GD

- **Convergence of GD vs. SGD**



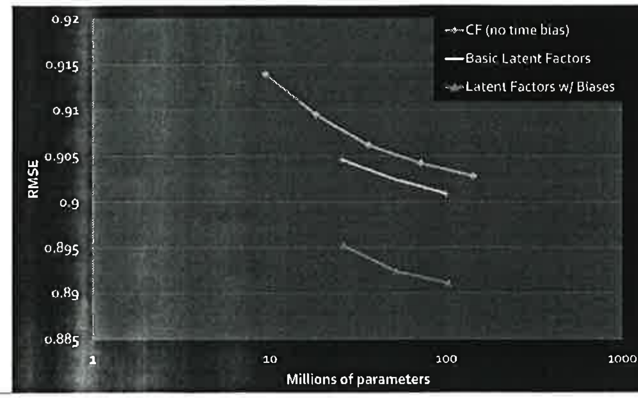
GD improves the value of the objective function at every step.
SGD improves the value but in a "noisy" way.
GD takes fewer steps to converge but each step takes much longer to compute.
In practice, **SGD** is much faster!

Fitting the New Model

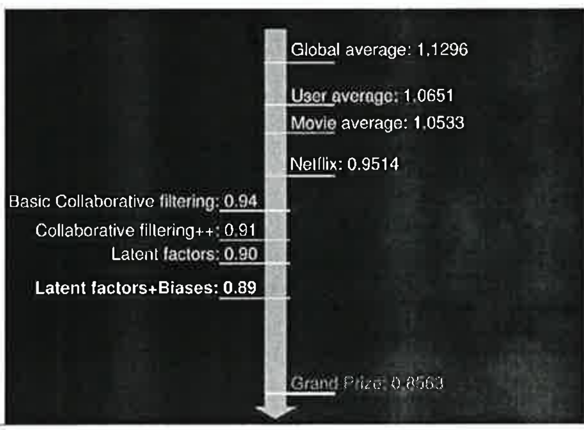
Solve

- $\min_{Q,P,b_x,b_i} \sum_{(x,i)\in R} (r_{x,i} - (\mu + b_x + b_i + q_i p_x))$
  $+\lambda_1 \sum_i \|q_i\|^2 + \lambda_2 \sum_x \|p_x\|^2 + \lambda_3 \sum_x \|b_x\|^2 + \lambda_4 \sum_i \|b_i\|^2$
- We can use **Stochastic Gradient Descent** to find parameters
  - Note: Both biases b_x, b_i as well as interactions q_i, p_x are treated as parameters (we estimate them)

Performance of Various Methods

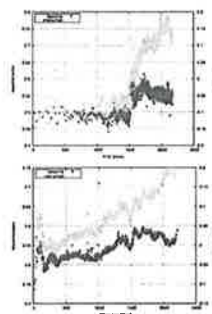


Performance of Various Methods



Temporal Biases of Users

- **Sudden rise in the average movie rating** (early 2004)
  - Improvements in Netflix
  - GUI improvements
  - Meaning of rating changed
- Movie age
  - Users prefer new movies without any reasons
  - Older movies are just inherently better than newer ones



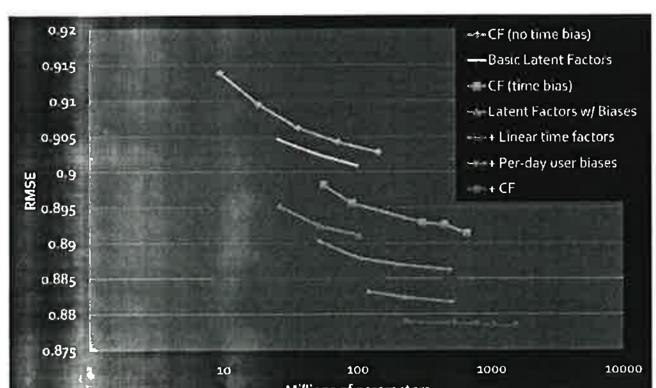
Y. Koren, Collaborative filtering with temporal dynamics, KDD '09

Temporal Biases & Factors

- **Original model:**
  $r_{xi} = \mu + b_x + b_i + q_i \cdot p_x$
- **Add time dependence to biases:**
  $r_{xi} = \mu + b_x(t) + b_i(t) + q_i \cdot p_x$
  - Make parameters b_x and b_i to depend on time
  - **(1)** Parameterize time-dependence by linear trends
    (2) Each bin corresponds to 10 consecutive weeks
    $b_i(t) = b_i + b_{i,\mathrm{Bin}(t)}$
- **Add temporal dependence to factors**
  - $p_x(t)$... user preference vector on day t

Y. Koren, Collaborative filtering with temporal dynamics, KDD '09

Adding Temporal Effects



Million $ Awarded Sept 21st 2009

Outline

1. [illegible]
2. [illegible]
3. [illegible]
4. [illegible]
5. Stochastic Gradient Descent
6. Extending Latent Factor Model to Include Biases
7. Spark

Hands on Exercise

See instructions on course-site

Resources

- `http://spark.apache.org`
- `http://mmds.org/`

Mining of Massive Datasets

Martin Takáč

ISE 395/495, Fall 2016, Analysis of Large Graphs: Link Analysis, PageRank

October 26, 2016

PageRank

Outline

1. Introduction and Motivation
2. PageRank - The "Flow" Formulation
3. PageRank: The Google Formulation
4. How do we actually compute the PageRank?

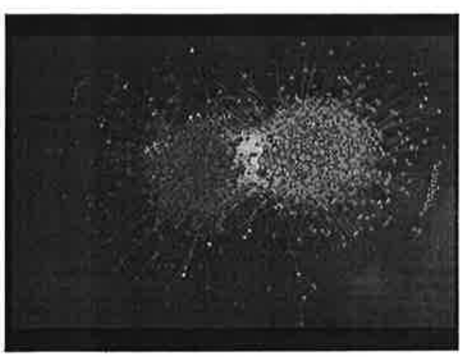
Outline

1. Introduction and Motivation
2. PageRank - The "Flow" Formulation
3. PageRank: The Google Formulation
4. How do we actually compute the PageRank?

Graph Data: Social Networks

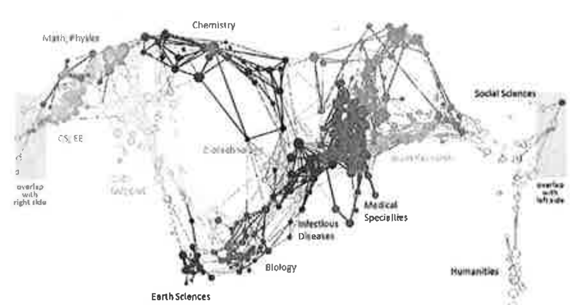
Facebook social graph
4-degrees of separation [Backstrom-Boldi-Rosa-Ugander-Vigna, 2011]

Graph Data: Media Networks



Connections between political blogs
Polarization of the network [Adamic-Glance, 2005]

Graph Data: Information Nets



Citation networks and Maps of science
[Börner et al., 2012]

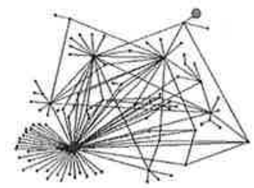
Web Search: 2 Challenges

2 challenges of web search:

- **(1) Web contains many sources of information**
  Who to "trust"?
  - Trick: Trustworthy pages may point to each other!
- **(2) What is the "best" answer to query "newspaper"?**
  - No single right answer
  - Trick: 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" Do you agree?**
  `www.joe-schmoe.com` vs. `www.wikipedia.org`
- There is large diversity in the web-graph node connectivity.
  Let's rank the pages by the link structure!



Link Analysis Algorithms

- We will cover the following **Link Analysis approaches** for computing **importances of nodes in a graph:**
  - Page Rank (today)
  - Topic-Specific (Personalized) Page Rank
  - Web Spam Detection Algorithms

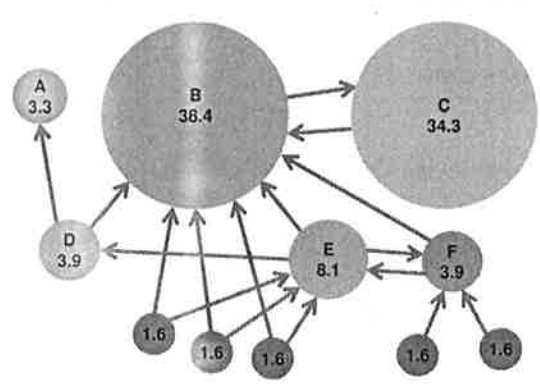
Outline

1. Introduction and Motivation
2. PageRank - The "Flow" Formulation
   - Power Iteration Method
3. PageRank - The Google Formulation
4. How do we actually compute the PageRank?

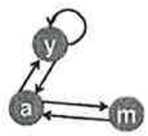
Links as Votes

- **Idea: Links as votes**
  Page ~~is more important~~ if it has more links
  - In-coming links? Out-going links?
- **Think of in-links as votes:**
  - `www.wikipedia.org` has 43,400,946 in-links
  - `www.joe-schmoe.com` has 1 in-link
- **Are all in-links are equal?**
  - Links from important pages count more
  - Recursive question!

Example: PageRank Scores



Example: Flow Equations & M



	y	a	m
y	½	½	0
a	½	0	1
m	0	½	0

$r = M \cdot r$

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

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

Power Iteration Method

- **Given a web graph with N nodes, where the nodes are pages and edges are hyperlinks**
- **Power iteration:** a simple iterative scheme
  - Suppose there are N web pages
  - Initialize $r^0 = [1/N, \ldots, 1/N]^T$
  - Iterate $r^{t+1} = Mr^t$
  - Stop when $\|r^{t+1} - r^t\|_1 \leq \epsilon$

PS: $\|x\|_1 = \sum_i |x_i|$ is the ℓ_1 norm. We can use any other vector norm, e.g. Euclidean

Page Rank: How to solve?

- **Power Iteration:**
  - Set $r_j = 1/N$
  - **1:** $r'_j = \sum_{i \to j} \frac{r_i}{d_i}$
  - **2:** $r = r'$
  - Goto **1**
- **Example:**

	y	a	m
y	½	½	0
a	½	0	1
m	0	½	0

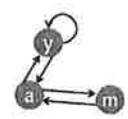
$r_y = r_y/2 + r_a/2$
$r_a = r_y/2 + r_m$
$r_m = r_a/2$

$$\begin{pmatrix} r_y \\ r_a \\ r_m \end{pmatrix} = \quad \begin{matrix} 1/3 \\ 1/3 \\ 1/3 \end{matrix}$$

Iteration 0, 1, 2, ...

Page Rank: How to solve?

- **Power Iteration:**
  - Set $r_j = 1/N$
  - **1:** $r'_j = \sum_{i \to j} \frac{r_i}{d_i}$
  - **2:** $r = r'$
  - Goto **1**
- **Example:**



	y	a	m
y	½	½	0
a	½	0	1
m	0	½	0

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

$$\begin{pmatrix} r_y \\ r_a \\ r_m \end{pmatrix} = \quad \begin{matrix} 1/3 & 1/3 & 5/12 & 9/24 & & 6/15 \\ 1/3 & 3/6 & 1/3 & 11/24 & \ldots & 6/15 \\ 1/3 & 1/6 & 3/12 & 1/6 & & 3/15 \end{matrix}$$

Iteration 0, 1, 2, ...

Why Power Iterations Works? (1)

- **Power iteration:**
  A method for finding dominant eigenvector (the vector corresponding to the largest eigenvalue)
  - $r^{(1)} = M \cdot r^{(0)}$
  - $r^{(2)} = M \cdot r^{(1)} = M(Mr^{(1)}) = M^2 \cdot r^{(0)}$
  - $r^{(3)} = M \cdot r^{(2)} = M(M^2 r^{(0)}) = M^3 \cdot r^{(0)}$
- **Claim:**
  Sequence $M \cdot r^{(0)}, M^2 \cdot r^{(0)}, \ldots M^k \cdot r^{(0)}, \ldots$ approaches the dominant eigenvector of M

Why Power Iterations Works? (2)

- Claim: Sequence $M \cdot r^{(0)}, M^2 \cdot r^{(0)}, \ldots M^k \cdot r^{(0)}, \ldots$ approaches the dominant eigenvector of M
- Proof:
  - Assume M has n linearly independent eigenvectors, $x_1, x_2, \ldots, x_n$ with corresponding eigenvalues $\lambda_1, \lambda_2, \ldots, \lambda_n$, where $\lambda_1 > \lambda_2 > \cdots > \lambda_n$
  - Vectors $x_1, x_2, \ldots, x_n$ form a basis and thus we can write: $r^{(0)} = c_1\, x_1 + c_2\, x_2 + \cdots + c_n\, x_n$
  - $Mr^{(0)} = M(c_1\, x_1 + c_2\, x_2 + \cdots + c_n\, x_n)$
    $= c_1(Mx_1) + c_2(Mx_2) + \cdots + c_n(Mx_n)$
    $= c_1(\lambda_1 x_1) + c_2(\lambda_2 x_2) + \cdots + c_n(\lambda_n x_n)$
  - **Repeated multiplication on both sides produces**
    $M^k r^{(0)} = c_1(\lambda_1^k x_1) + c_2(\lambda_2^k x_2) + \cdots + c_n(\lambda_n^k x_n)$

Does this converge?

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

- **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, ...

Does it converge to what we want?

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

- **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, ...

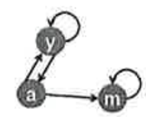
PageRank: Problems

2 problems:

1. Some pages are dead ends (have no out-links)
   - Random walk has "nowhere" to go to
   - Such pages cause importance to "leak out"
2. Spider traps: (all out-links are within the group)
   - random walked gets "stuck" in a trap
   - and eventually spider traps absorb all importance

Problem: Spider Traps

- **Power Iteration:**
  - Set $r_j = 1$
  - $r_j = \sum_{i \to j} \frac{r_i}{d_i}$
    - And iterate



m is a spider trap

	y	a	m
y	½	½	0
a	½	0	0
m	0	½	1

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

- **Example:**

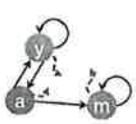
$$\begin{bmatrix} r_y \\ r_a \\ r_m \end{bmatrix} = \begin{matrix} 1/3 & 2/6 & 3/12 & 5/24 & & 0 \\ 1/3 & 1/6 & 2/12 & 3/24 & \dots & 0 \\ 1/3 & 3/6 & 7/12 & 16/24 & & 1 \end{matrix}$$

Iteration 0, 1, 2, ...

All the PageRank score gets "trapped" in node m.

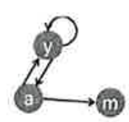
Solution: Teleports!

- **The Google solution for spider traps: At each time step, the random surfer has two options**
  - With prob. β, follow a link at random
  - With prob. $1-\beta$, jump to some random page
  - Common values for β are in the range 0.8 to 0.9
- **Surfer will teleport out of spider trap within a few time steps**



Problem: Dead Ends

- **Power Iteration:**
  - Set $r_j = 1$
  - $r_j = \sum_{i \to j} \frac{r_i}{d_i}$
    - And iterate



	y	a	m
y	½	½	0
a	½	0	0
m	0	½	0

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

- **Example:**

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

Iteration 0, 1, 2, ...

Here the PageRank "leaks" out since the matrix is not stochastic.

Computing Page Rank

- **Key step is matrix-vector multiplication** $r^{new} = Ar^{old}$
- Easy if we have enough main memory to hold A, r^{old}, r^{new}
- Say $N = 1$ billion pages
  - we need 4 bytes for each entry (say)
  - 2 billion entries for vectors, approx 8GB
  - Matrix A has N^2 entries
    - 10^{18} is a large number! (3,637,979 TB)

$$A = \beta \cdot M + (1-\beta)\,[1/N]_{N \times N}$$

$$A = 0.8 \begin{bmatrix} 1/2 & 1/2 & 0 \\ 1/2 & 0 & 0 \\ 0 & 1/2 & 1 \end{bmatrix} + 0.2 \begin{bmatrix} 1/3 & 1/3 & 1/3 \\ 1/3 & 1/3 & 1/3 \\ 1/3 & 1/3 & 1/3 \end{bmatrix}$$

$$= \begin{bmatrix} 7/15 & 7/15 & 1/15 \\ 7/15 & 1/15 & 1/15 \\ 1/15 & 7/15 & 13/15 \end{bmatrix}$$

Matrix Formulation

- Suppose there are N pages
- Consider page i, with d_i out-links
- We have $M_{ji} = 1/d_i$ when $i \to j$ and $M_{ji} = 0$ otherwise
- **The random teleport is equivalent to:**
  - Adding a **teleport link** from i to every other page and setting transition probability to $(1-\beta)/N$
  - Reducing the probability of following each out-link from $1/d_i$ to β/d_i
  - **Equivalent:** Tax each page a fraction $(1-\beta)$ of its score and redistribute evenly

Rearranging the Equation

- $\boldsymbol{r} = \boldsymbol{A} \cdot \boldsymbol{r}$, **where** $A_{ji} = \beta\, M_{ji} + \frac{1-\beta}{N}$
- $r_j = \sum_{i=1}^{N} A_{ji} \cdot r_i$
- $r_j = \sum_{i=1}^{N} \left[\beta\, M_{ji} + \frac{1-\beta}{N}\right] \cdot r_i$

$$= \sum_{i=1}^{N} \beta\, M_{ji} \cdot r_i + \frac{1-\beta}{N} \sum_{i=1}^{N} r_i$$

$$= \sum_{i=1}^{N} \beta\, M_{ji} \cdot r_i + \frac{1-\beta}{N} \qquad \text{since } \sum r_i = 1$$

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

Note: we assumed that M has no dead-ends

Sparse Matrix Formulation

- We just rearranged the **PageRank equation**

$$\boldsymbol{r} = \boldsymbol{\beta M} \cdot \boldsymbol{r} + \left[\frac{\boldsymbol{1-\beta}}{\boldsymbol{N}}\right]_{\boldsymbol{N}}$$

  - where $[(1-\beta)/N]_N$ is a vector with all N entries $(1-\beta)/N$

- M is a **sparse matrix!** (with no dead-ends)
  - 10 links per node, approx 10N entries
- So in each iteration, we need to:
  - Compute $r^{new} = \beta M \cdot r^{old}$
  - Add a constant value **(1-β)/N** to each entry in r^{new}
    - **Note if M contains dead-ends then $\sum_j r_j^{new} < 1$ 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 (can have **spider traps** and **dead ends**)
  - Parameter β
- **Output: PageRank vector r^{new}**
  - **Set:** $r_j^{old} = \frac{1}{N}$
  - **repeat until convergence:** $\sum_j \left|r_j^{new} - r_j^{old}\right| > \varepsilon$
    - $\forall j$: $r'^{new}_j = \sum_{i \to j} \beta \frac{r_i^{old}}{d_i}$
      $r'^{new}_j = 0$ if in-degree of j is 0
    - Now re-insert the leaked PageRank:
      $\forall j$: $r_j^{new} = r'^{new}_j + \frac{1-S}{N}$ where: $S = \sum_j r'^{new}_j$
    - $r^{old} = r^{new}$

If the graph has no dead-ends then the amount of leaked PageRank is 1-β. But since we have dead-ends the amount of leaked PageRank may be larger. We have to explicitly account for it by computing S.

Sparse Matrix Encoding

Encode sparse matrix using only nonzero entries

- Space proportional roughly to number of links
- Say 10N, or $4 \cdot 10 \cdot 1$ billion = 40GB
- Still won't fit in memory, but will fit on disk

source node	degree	destination nodes
0	3	1, 5, 7
1	5	17, 64, 113, 117, 245
2	2	13, 23

Some Problems with Page Rank

- Measures generic popularity of a page
  - Biased against topic-specific authorities
  - Solution: Topic-Specific PageRank (next)
- Uses a single measure of importance
  - Other models of importance
  - Solution: Hubs-and-Authorities
- **Susceptible to Link spam**
  - Artificial link topographies created in order to boost page rank
  - Solution: TrustRank

Mining of Massive Datasets

Martin Takáč

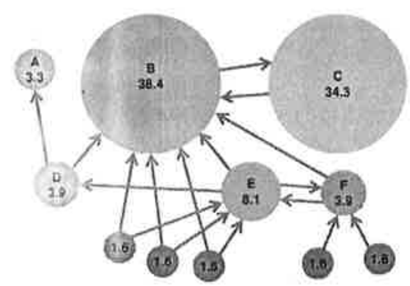
ISE 395/495, Fall 2016, Analysis of Large Graphs: TrustRank and WebSpam

November 2, 2016

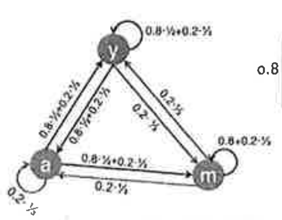
Outline

Outline

Example: PageRank Scores



Random Teleports ($\beta = 0.8$)



$$
\begin{array}{c}M\\ 0.8\begin{bmatrix}1/2 & 1/2 & 0\\ 1/2 & 0 & 0\\ 0 & 1/2 & 1\end{bmatrix}\end{array}
+0.2
\begin{array}{c}[1/N]_{N\times N}\\ \begin{bmatrix}1/3 & 1/3 & 1/3\\ 1/3 & 1/3 & 1/3\\ 1/3 & 1/3 & 1/3\end{bmatrix}\end{array}
$$

$$
\begin{array}{c|ccc}
y & 7/15 & 7/15 & 1/15\\
a & 7/15 & 1/15 & 1/15\\
m & 1/15 & 7/15 & 13/15
\end{array}
$$

A

y	1/3	0.33	0.24	0.26		7/33
a =	1/3	0.20	0.20	0.18	...	5/33
m	1/3	0.46	0.52	0.56		21/33

r = A r

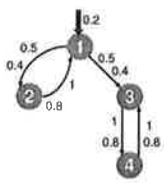
Equivalently: $r = \beta\, M \cdot r + \left[\frac{1-\beta}{N}\right]_N$

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: $r_j^{(0)} = \frac{1}{N}$, $t = 1$
  - do:
    - $\forall j$: $r'^{(t)}_j = \sum_{i \to j} \beta \frac{r_i^{(t-1)}}{d_i}$
      $r'^{(t)}_j = 0$ if in-degree 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 $\sum_j \left| r_j^{(t)} - r_j^{(t-1)} \right| > \varepsilon$

If the graph has no dead-ends then the amount of leaked PageRank is 1-β. But since we have dead-ends the amount of leaked PageRank may be larger. We have to explicitly account for it by computing S.

Example: Topic-Specific PageRank



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]

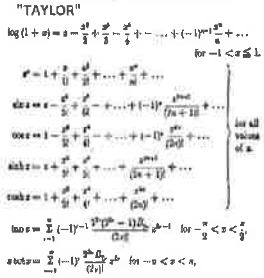
Discovering the Topic Vector S

- 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
  - Can 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, ...

Searching for "Taylor"...

WHAT OTHER PEOPLE THINK WHEN THEY HEAR "TAYLOR"

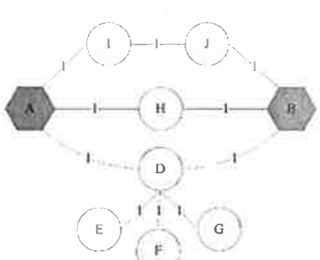
WHAT WE THINK WHEN WE HEAR "TAYLOR"



Outline

1. Recap from Last Lecture
2. Topic-Specific PageRank
3. Application to Measuring Proximity in Graphs
4. Spark Hands-on
5. TrustRank: Combating the Web Spam
6. HITS: Hubs and Authorities

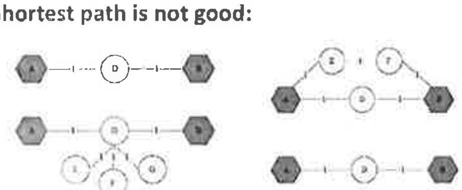
Proximity on Graphs [Tong-Faloutsos, '06]



a.k.a.: Relevance, Closeness, 'Similarity'...

Good proximity measure?

- **Shortest path is not good:**



- **No effect of degree-1 nodes (E, F, G)!**
- Multi-faceted relationships

Outline

1. Recap from Last Lecture
2. Topic Specific PageRank
3. Application to Measuring Proximity in Graphs
4. **Spark Hands on**
5. TrustRank: Combating the Web Spam
6. HITS: Hubs and Authorities

See Lab #3

Outline

1. Recap from Last Lecture
2. Topic Specific PageRank
3. Application to Measuring Proximity in Graphs
4. Spark Hands on
5. **TrustRank: Combating the Web Spam**
6. HITS: Hubs and Authorities

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
- This is a very broad definition
  - SEO industry might disagree!
  - SEO = search engine optimization
- 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:**
    - (1) Number of times query words appeared
    - (2) Prominence of word position, e.g. title, header

First Spammer

- As people began to use search engines to find things on the Web, those with commercial interests tried to **exploit search engines** to bring people to their own site whether they wanted to be there or not
- **Example:**
  - Shirt-seller might pretend to be about "movies"
- **Techniques for achieving high relevance/importance for a web page**

Google vs. Spammers: Round 2!

- Once Google became the dominant search engine, spammers began to work out ways to fool Google
- **Spam farms** were developed to concentrate PageRank on a single page
- **Link spam:** Creating link structures that boost PageRank of a particular page

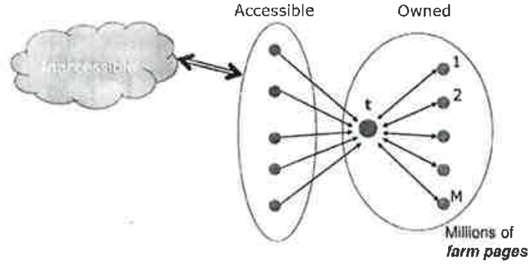
Link Spamming

- Three kinds of web pages from a spammer's point of view
  - Inaccessible pages
  - Accessible pages
    - e.g., blog comments pages
    - spammer can post links to his pages
  - Owned pages
    - Completely controlled by spammer
    - May span multiple domain names

Link Farms

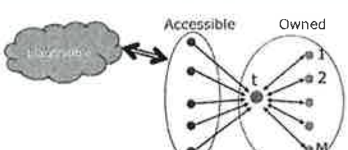
- **Spammer's goal:**
  - Maximize the PageRank of target page t
- **Technique:**
  - Get as many links from accessible pages as possible to target page t
  - Construct "link farm" to get PageRank multiplier effect

Link Farms



One of the most common and effective organizations for a link farm

Analysis



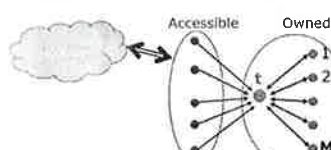
N...# pages on the web
M...# of pages spammer owns

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

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

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

Analysis



N...# pages on the web
M...# of pages spammer owns

- $y = \frac{x}{1-\beta^2} + c\frac{M}{N}$ where $c = \frac{\beta}{1+\beta}$
- For β = 0.85, 1/(1-β²)= 3.6

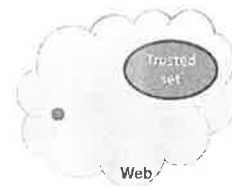
- Multiplier effect for acquired PageRank
- By making ***M*** large, we can make ***y*** as large as we want

Approaches to Picking Seed Set

- Suppose we want to pick a seed set of k pages
- **How to do that?**
  1. PageRank:
     - Pick the top k pages by PageRank
     - Theory is that you can't get a bad page's rank really high
  2. **Use trusted domains** whose membership is controlled, like .edu, .mil, .gov

Spam Mass

- In the TrustRank model, we start with good pages and propagate trust
- **Complementary view:**
  What fraction of a page's PageRank comes from spam pages?
- In practice, we don't know all the spam pages, so we need to estimate



Spam Mass Estimation

Solution 2

- r_p PageRank of page p from PageRank
- r_p^+ PageRank of p with teleport into **trusted** pages only from ~~TrustPage~~ TrustRank
- Then: **What fraction of a page's PageRank comes from spam pages?**

$$r_p^- = r_p - r_p^+$$

- **Spam mass of** $p = \frac{r_p^-}{r_p}$
  - Pages with high spam mass are spam

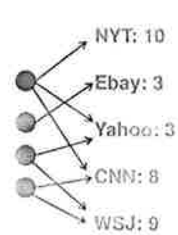
Outline

1. Recap from Last Lecture
2. Topic Specific PageRank
3. Application to Measuring Proximity in Graphs
4. [illegible]
5. TrustRank: Combating the Web Spam
6. HITS: Hubs and Authorities

Hubs And Authorities

- **HITS (Hypertext-Induced Topic Selection)**
  - **Is a measure of importance of pages or documents, similar to PageRank**
  - Proposed at around same time as PageRank ('98)
- **Goal:** Say we want to find good newspapers
  - Don't just find newspapers. Find "experts" – people who link in a coordinated way to good newspapers
- **Idea:** Links as votes
  - Page is more important if it has more links
    In-coming links? Out-going links?

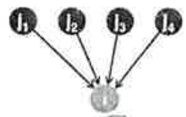
Finding Newspapers



- **Hubs** and **Authorities**
  Each page has 2 scores:
  - Quality as an expert (**hub**):
    - Total sum of votes of authorities pointed to
  - Quality as a content (**authority**):
    - Total sum of votes coming from experts
- **Principle of repeated improvement**

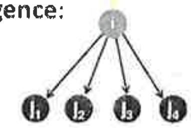
Hubs and Authorities [Kleinbers '98]

- **Each page i has 2 scores:**
  - Authority score: a_i
  - Hub score: h_i



$a_i = \sum_{j \to i} h_j$

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



$h_i = \sum_{i \to j} a_j$

Hubs and Authorities [Kleinbers '98]

- HITS converges to a single stable point
- Notation:
  - Vector $a = (a_1, \ldots, a_n)$, $h = (h_1, \ldots, h_n)$
  - Adjacency matrix $A (N \times N)$: $A_{i,j} = 1$ if $i \to j$, 0 otherwise
- Then $h_i = \sum_{i \to j} a_j$ can be rewritten as $h_i = \sum_j A_{i,j} a_j$
- so $h = Aa$
- similarly, $a_i = \sum_{j \to i} h_j$ can be rewritten as $a_i = \sum_j A_{j,i} h_j = A^T h$

$a = A^T h$

Hubs and Authorities

- **HITS algorithm in vector notation:**
  - Set: $a_i = h_i = \frac{1}{\sqrt{n}}$

  Repeat until convergence:
  - $h = A \cdot a$
  - $a = A^T \cdot h$
  - Normalize a and h
- **Then:** $a = \underbrace{A^T \cdot \underbrace{(A \cdot a)}_{\text{new } h}}_{\text{new } a}$

Convergence criterion:

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

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

a is updated (in 2 steps):
$a = A^T(A\,a) = (A^T A)\,a$

h is updated (in 2 steps):
$h = A\,(A^T h) = (A\,A^T)\,h$

Repeated matrix powering

Existence and Uniqueness

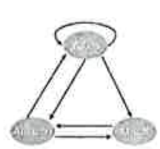
- $h = \lambda\, A\, a$
- $a = \mu\, A^T\, h$
- $h = \lambda\, \mu\, A\, A^T\, h$
- $a = \lambda\, \mu\, A^T A\, a$

$\lambda = 1 / \sum h_i$
$\mu = 1 / \sum a_i$

- Under reasonable assumptions about **A**, HITS **converges to vectors h^* and a^*:**
  - h^* is the principal eigenvector of matrix $A\,A^T$
  - a^* is the principal eigenvector of matrix $A^T A$

Example of HITS

$A = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}$ $A^T = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix}$



h(yahoo)	=	.58	.80	.80	.79	...	.788
h(amazon)	=	.58	.53	.53	.57	...	.577
h(m'soft)	=	.58	.27	.27	.23	...	.211
a(yahoo)	=	.58	.58	.62	.62	...	.628
a(amazon)	=	.58	.58	.49	.49	...	.459
a(m'soft)	=	.58	.58	.62	.62	...	.628

PageRank and HITS

- PageRank and HITS are two solutions to the same problem:
  - What is the value of an in-link from u to v?
  - In the PageRank model, the value of the link depends on the links into u
  - In the HITS model, it depends on the value of the other links out of u
- The destinies of PageRank and HITS post-1998 were very different