大数据管理系统与大规模数据分析 - June 12'rd, 2018

Data Analysis Concepts

https://github.com/rh01

3V / 4V

- 1-Scale (**Volume**)
- 2-Complexity (**Variety**)
- 3-Speed(**Velocity**)
- 4-Doubt(**Veracity**)

Distribute Hash Table

- Hash table spread over many nodes
  - Distributed over a wide area
- Main design goals
  - **Decentralization**
- no central coordinator
  - **Scalability**
- efficient even with large # of nodes
  - **Fault tolerance**
- tolerate nodes joining/leaving

Napster vs. Gnutella

- Napster $\longrightarrow$ Centralized Lookup
- Gnutella $\longrightarrow$ Flooded queries

Centralized vs. Flooded queries

- Centralized
  - Table size $^-O(n)$
  - Number of hops $^-O(1)$
- Flooded queries
  - Table size $^-O(1)$
  - Number of hops $^-O(n)$

DHTs: Problems

- Problem 1 (**dynamicity**): adding or removing nodes
  - With hash mod N, virtually every key will change its location!
    - $h(k) \bmod N \neq h(k) \bmod (N+1) \neq h(k) \bmod (N-1)$
- **Solution**: use consistent hashing (一致性哈希算法)

  - Define a fixed hash space
  - All hash values fall within that space and do not depend on the number of peers (hash bucket)
  - Each key goes to peer closest to its ID in hash space (according to some proximity metric)
- Problem 2 **(size):** all nodes must be known to insert or lookup data
  - Works with small and static server populations
- **Solution:** each peer knows of only a few “neighbors”
  - Messages are routed through neighbors via multiple hops (overlay routing)

Good DHT Design Principle

- For each object, the node(s) responsible for that object should be reachable via a “short”path (**small diameter**) (能夠響應請求找到最短路徑)
  - The different DHTs differ fundamentally only in the routing approach
- The number of neighbors for each node should remain “reasonable”(**small degree**) （節點的鄰居節點的數量是合理的，出/入度小）
- DHT routing mechanisms should be decentralized (**no single point of failure or bottleneck**)（保证服务可用，无单点故障或者性能瓶颈）
- Should gracefully handle nodes joining and leaving (优雅的处理节点加入或者节点离开)
  - Repartition the affected keys over existing nodes
  - Reorganize the neighbor sets
  - Bootstrap mechanisms to connect new nodes into the DHT
- To achieve good performance, DHT must provide low stretch
  - Minimize ratio of DHT routing vs. unicast latency

Service Discover

- CAN(内容分發網絡)
- Chord(和弦算法)
- Pastry
- Tapestry(地毯算法)
- .....

*Example: CAN-Content Addressable NetWork

- 定义 $\Longrightarrow$ A hash-based P2P file indexing and lookup scheme
- 特点 $\Longrightarrow$ Scalability,Fault tolerance, Self-organization，Decentralization
- 架构 $\Longrightarrow$ Cartesian coordinator space(笛卡尔空间建立的)

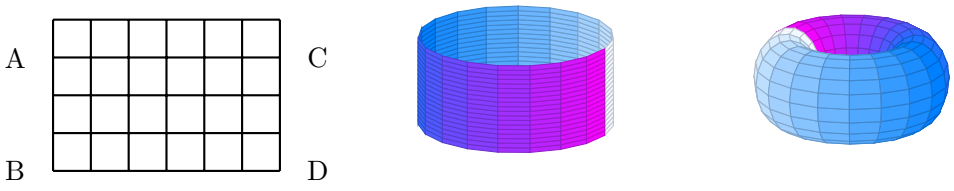

图 1：torus

- Routing in CAN?

  - CAN 的每个节点都会维护一张路由表，路由表保存了邻居节点的 IP 地址和相应的虚拟地址区域，节点最终会把消息路由到正确的节点上.
  - If d-torus is partitioned into n equal zones, an average routing path goes through $\frac{d}{4}n^{\frac{1}{d}}$ hops, or $O(n^{\frac{1}{d}})$
  - Steps:
    * 先找到那一块区域距离目标节点比较近
    * 通过路由表查找该区域的节点 IP 地址
    * 路由到该节点上，最终路由到目标区域。

- 如何 Join CAN?
  - Find a node already in the overlay network.
  - Identify a zone that can be split
  - Update the routing tables of nodes neighboring the newly split zone.

  具体流程：如果找到已经存的经典，通过引导节点，通知节点加入网络（分配 IP），该节点收到加入网络的请求该节点表示自己的区域空间（随机选择坐标空间任意一点，并将请求转发给空间该点），不能够将 IP 地址指向空间该点，通过 zone-To-IP 路由表，将请求转发给正确的设备上，一旦目标区域的管理节点收到了 Join 请求，会将自己的区域分为两份，一份空间留给自己，另一份给新加入节点，如果管理节点不接受该请求，那么节点将会重新选择空间区域的某点，并且继续上述操作，直至加入成功为止
- 如何 Departing CAN?
  - identify a node is departing
  - have the departing node's zone merged or taken over by a neighboring node
  - update the routing tables across the network.

  Detecting a node's departure can be done, for instance, via heartbeat messages that periodically broadcast routing table information between neighbors. After a predetermined period of silence from a neighbor, that neighboring node is determined as failed and is considered a departing node. Alternatively, a node that is willingly departing may broadcast such a notice to its neighbors.
  After a departing node is identified, its zone must be either merged or taken over. First the departed node's zone is analyzed to determine whether a neighboring node's zone can merge with the departed node's zone to form a valid zone. For example, a zone in a 2d coordinate space must be either a square or rectangle and cannot be L-shaped. The validation test may cycle through all neighboring zones to determine if a successful merge can occur. If one of the potential merges is deemed a valid merge, the zones are then merged. If none of the potential merges are deemed valid, then the neighboring node with the smallest zone takes over control of the departing node's zone.[1] After a take-over, the take-over node may periodically attempt to merge its additionally controlled zones with respective neighboring zones.
  If the merge is successful, routing tables of neighboring zones' nodes are updated to reflect the merge. The network will see the subsection of the overlay network as one, single zone after a merge and treat all routing processing with this mindset. To effectuate a take-over, the take-over node updates neighboring zones' nodes' routing tables, so that requests to either zone resolve to the take-over node. And, as such, the network still sees the subsection of the overlay network as two separate zones and treats all routing processing with this mindset.

  - 在两种情况下，Node 可以被标示 Departing NODE.

    * 心跳检测机制，在给定的时间里，邻居节点将会把自己的状态信息上报
    * 自愿离开
  - 一旦被标示为 DEPARTING NODE，将会有两种动作：**ZONE Merge** 或者 **Zone Take Over** $\Longrightarrow$ **ZONE Merge**
    * departing node 将会分析是否可以合并
    * 如果潜在的合并可以成灰合法的合并，那么就可以正常合并
    * 否则将会由最小区域的相邻节点接管（take over）
    * 接管后，接管节点会周期地将其控制的区域与相邻区域合并，如果合并成功，全网将会更新路由表

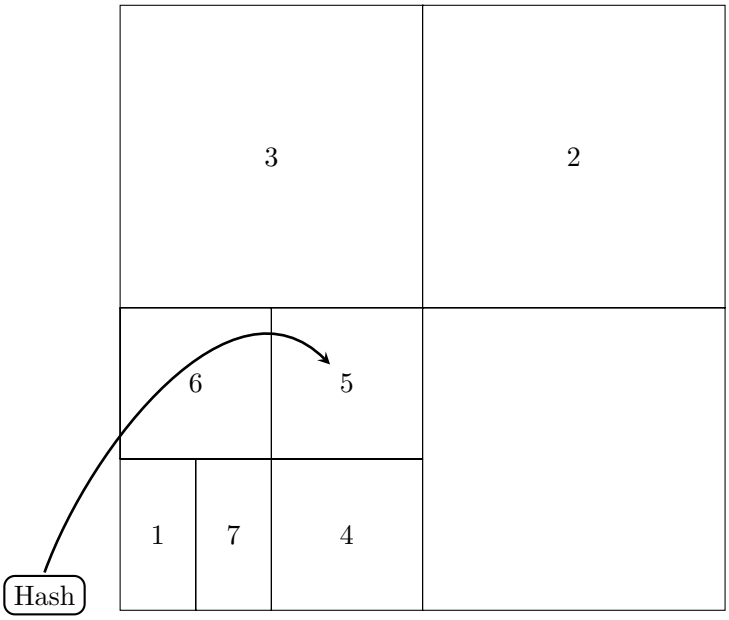

图 2. Hash

Finding Similar Items

Distance Measures

Ex. Jaccard distance/similarity

- The **Jaccard similarity** of two sets is the size of their intersection divided by the size of their union: $sim(C_1, C_2) = \frac{|C_1 \cup C_2|}{|C1 \cup C_2|}$
- **Jaccard distance:** $d(C_1, C_2) = 1 - \frac{|C_1 \cup C_2|}{|C1 \cup C_2|}$

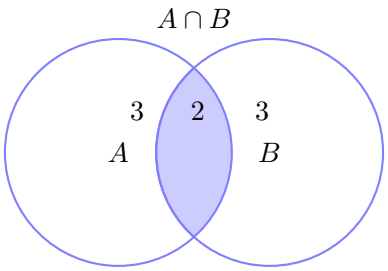

图 3：Distance Measures

3 Essential Steps for Similar Docs

- **Shingling**

  - Convert documents to sets
- **Min-Hashing**
  - Convert large sets to short signatures, while preserving similarity
- **Locality-Sensitive Hashing**
  - Focus on pairs of signatures likely to be from similar documents

Shingling

- A **k-shingle (or k-gram)** for a document is a sequence of k tokens that appears in the doc
- Example: k=2; document D1 = abcab
  - Set of 2-shingles: S(D1) = {ab, bc, ca}
  - Option: Shingles as a bag (multiset), count ab twice: S'(D1) = {ab, bc, ca, ab}

  Compressing Shingles
  - Represent a document by the set of hash values of its k-shingles
    * k=2; document D1= abcab
      · Set of 2-shingles: S(D1) = ab, bc, ca
      · Hash the singles: h(D1) = 1, 5, 7

Minhash/LSH

Ex. Naïvely, we would have to compute pairwise
  - Jaccard similarities for every pair of docs $\frac{N(N-1)}{2} \simeq 51011$comparisons
  - At 105 secs/day and 106 comparisons/sec, it would take 5 days

Minhash

- Encode sets using 0/1 (bit, boolean) vectors
  - One dimension per element in the universal set
- intersection $\Longrightarrow$ **AND**, and union $\Longrightarrow$ **OR**.
- Example: C1 = 10111; C2 = 10011
  - $C1 \cup C2 = \{10111\} \Longrightarrow |C1 \cup C2| = 4$
  - $C1 \cap C2 = \{10011\} \Longrightarrow |C1 \cup C2| = 3$

    $\therefore$ Jaccard similarity (not distance) = 3/4, Distance: d(C1,C2) = 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 sparse!
- Example:
  - sim(C1 ,C2) = ?

  - Size of intersection = 3; size of union = 6, Jaccard similarity (not distance) = 3/6
  - d(C1,C2) = 1 −(Jaccard similarity) = 3/6
- Documents $\Longrightarrow$ Sets of shingles
- Represent sets as boolean vectors in a matrix

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

表 1: document vs Shingles

Finding Similar Columns

- Naïve approach:
  - Signatures of columns: small summaries of columns
  - Examine pairs of signatures to find similar columns
    - Essential: Similarities of signatures and columns are related
  - Optional: Check that columns with similar signatures are really similar

***Hashing Columns (Signatures)**

- Key idea: "hash" each column C to a small signature h(C), such that:
  - h(C) is small enough that the signature fits in RAM
  - sim(C1, C2) is the same as the "similarity" of signatures h(C1) and h(C2)
- Goal: Find a hash function $h(\cdot)$ such that:
  - If sim(C1,C2) is high, then with high prob. h(C1) = h(C2)
  - If sim(C1,C2) is low, then with high prob. h(C1) $\neq$ h(C2)

Min-Hashing

Not all similarity metrics have a suitable **hash function**! There is a suitable hash function for the **Jaccard similarity**: It is called **Min-Hashing**

- Imagine the rows of the boolean matrix permuted under random permutation
- Define a "hash" function $h_\pi(C)$ = the index of the first (in the permuted order π) row in which column C has value 1:

$$h_\pi(C) = min_\pi \pi(C)$$

- **Use several (e.g., 100) independent hash functions (that is, permutations) to create a signature of a column**.

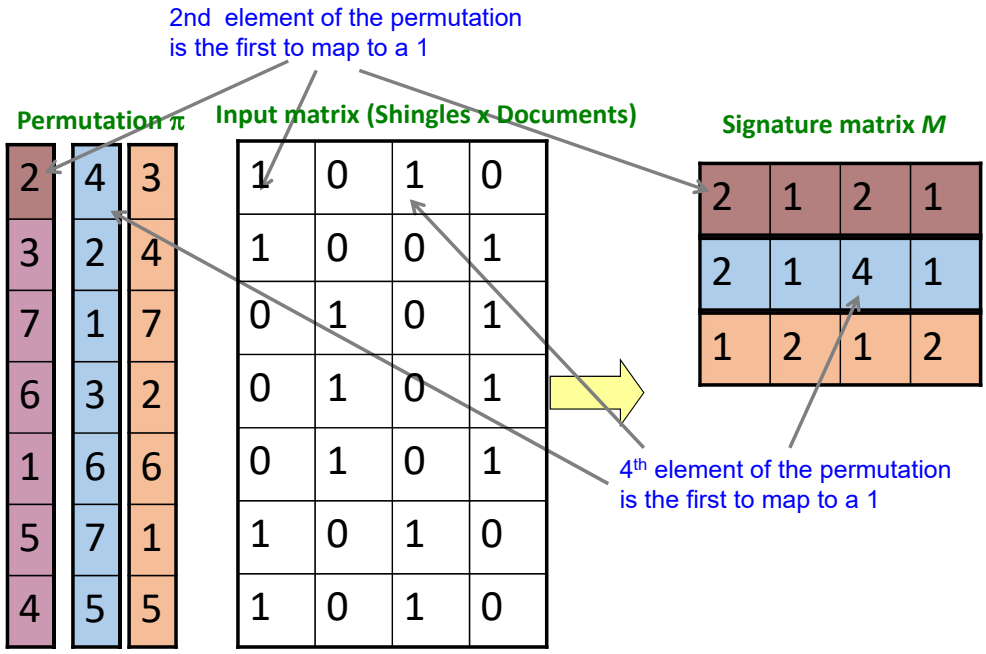

图 4：Min-Hashing Example

$\mathbf{Pr}[h_\pi(\mathbf{C1}) = h_\pi(\mathbf{C2})] = sim(\mathbf{C1}, \mathbf{C2})$

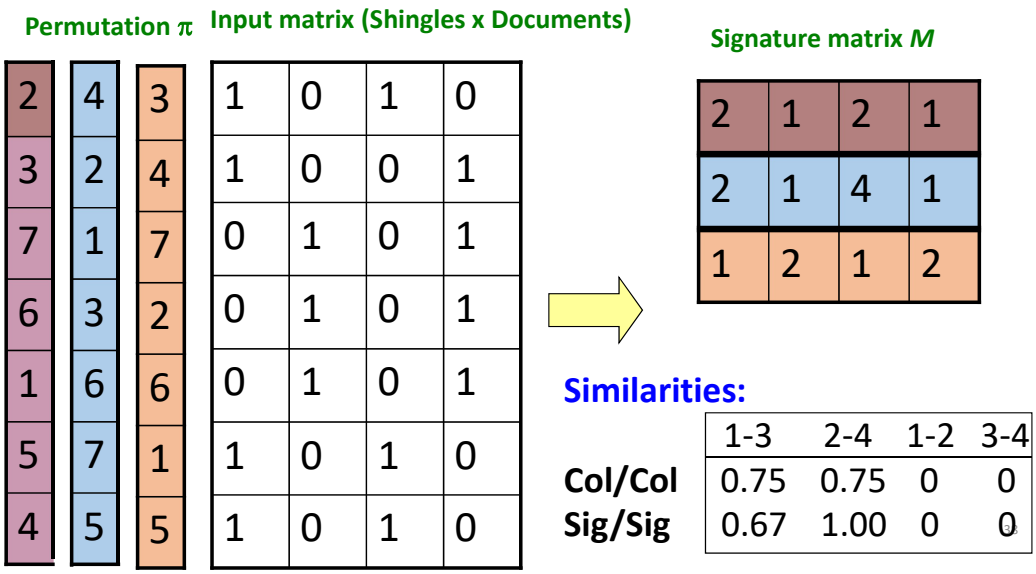

图 5：Min-Hashing Example

Locality-Sensitive Hashing

- 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

Why Reduce Dimensions?

- Discover hidden correlations/topics
  - Words that occur commonly together
  - Remove redundant and noisy features $\Longrightarrow$Not all words are useful
  - Interpretation and visualization
  - Easier storage and processing of the data

SVD-Definition

$$A_{[m\times n]} = U_{[m\times r]}\Sigma_{[r\times r]}(V_{[n\times r]})^{\mathrm{T}}$$

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

$$A = U\Sigma V^T = \underbrace{\begin{bmatrix}\mathbf{u}_1 & \mathbf{u}_2 & \dots & \mathbf{u}_r\end{bmatrix}}_{\mathrm{Col}\,A}\ \underbrace{\begin{bmatrix}\mathbf{u}_{r+1} & \dots & \mathbf{u}_m\end{bmatrix}}_{\mathrm{Nul}\,A^T}\begin{bmatrix}\sigma_1 & 0 & \dots & 0 & 0 & \dots & 0\\ 0 & \sigma_2 & \dots & 0 & 0 & \dots & 0\\ \dots & & & & & & \\ 0 & 0 & \dots & \sigma_r & 0 & \dots & 0\\ 0 & 0 & \dots & 0 & 0 & \dots & 0\\ \dots & & & & & & \\ 0 & 0 & \dots & 0 & 0 & \dots & 0\end{bmatrix}\begin{bmatrix}\mathbf{v}_1^T\\ \mathbf{v}_2^T\\ \dots\\ \mathbf{v}_r^T\\ \mathbf{v}_{r+1}^T\\ \dots\\ \mathbf{v}_n^T\end{bmatrix}\begin{matrix}\}\,\mathrm{Row}\,A\\ \\ \\ \\ \}\,\mathrm{Nul}\,A\\ \\ \\ \end{matrix}$$

$$A = U\Sigma V^T$$

$$= \underbrace{\begin{pmatrix} u_1 & \cdots & u_r & u_{r+1} & \cdots & u_m \end{pmatrix}}_{\begin{matrix}\mathrm{col}(A) & \mathrm{null}(A)\end{matrix}} \begin{pmatrix} \sigma_1 & & & & & \\ & \ddots & & & & \\ & & \sigma_r & & & \\ & & & 0 & & \\ & & & & \ddots & \\ & & & & & 0 \end{pmatrix} \begin{pmatrix} v_1^T \\ \\ v_r^T \\ v_{r+1}^T \\ \\ v_n^T \end{pmatrix} \begin{matrix}\}\,\mathrm{row}(A)\\ \\ \\ \}\,\mathrm{null}(A)\\ \\ \end{matrix}$$

图 6: SVD

SVD-Eigenvalue & Eigenvector

Given a n × n matrix A^TA, for any σ and v, if

$$A^TAv_j = \sigma_j v_j$$

Then σ is called eigenvalue, and w is called eigenvector.

The objective of the rotation transformation is to find the maximal variance. We Projection of data along v is Av. Variance:

$$\sigma^2 = (Av)^TAv = v^TA^TAv$$

SVD vs. PCA

For symmetric A, SVD is closely related to PCA

- PCA: $A = U\sigma U^T$
  - U and σ are eigenvectors and eigenvalues.
- SVD: $A = U\sigma V^T$
  - U is left(column) eigenvectors
  - V is right(row) eigenvectors
  - σ is the same eigenvalues
- Note the difference of A in PCA and SVD
- SVD: A is directly the data, e.g. word-by-document matrix
- PCA: A is covariance matrix, $A = X^TX$, each row in X is a sample

CUR Matrix Approximation

$$A_{[m\times n]} = C_{[m\ timesk]}U_{[k\times k]}R_{[k\times n]}$$

- C,R 源于某些列和行，而不像 SVD 那样（奇异矩阵）
- 优点
  - lower asymptotic time
  - 更易于理解，分解过后的矩阵的行列含义与原矩阵含义相同
- 缺点
  - 近似度没有 SVD 高
  - Rank-k 近似

A CUR matrix approximation is a set of three matrices that, when multiplied together, closely approximate a given matrix .A CUR approximation can be used in the same way as the low-rank approximation of the Singular value decomposition (SVD). CUR approximations are less accurate than the SVD, but they offer two key advantages, both stemming from the fact that the rows columns come from the original matrix (rather than left and right singular vectors):

There are methods to calculate it with lower asymptotic time complexity versus the SVD.

The matrices are more interpretable; The meanings of rows and columns in the decomposed matrix are essentially the same as their meanings in the original matrix.

Formally, a CUR matrix approximation of a matrix A is three matrices C, U, and R such that C is made from columns of A, R is made from rows of A, and that the product CUR closely approximates A. Usually the CUR is selected to be a rank-k approximation, which means that C contains k columns of A, R contains k rows of A, and U is a k-by-k matrix. There are many possible CUR matrix approximations, and many CUR matrix approximations for a given rank.

The CUR matrix approximation is often used in place of the low-rank approximation of the SVD in Principal components analysis. The CUR is **less accurate**, but **the columns of the matrix C are taken from A and the rows of R are taken from A**. In PCA, each column of A contains a data sample; thus, the matrix C is made of a subset of data samples. This is much easier to interpret than the SVD's left singular vectors, which represent the data in a rotated space. Similarly, the matrix R is made of a subset of variables measured for each data sample. This is easier to comprehend than the SVD's right singular vectors, which are another rotations of the data in space.

$$A = CUR = \underbrace{\begin{bmatrix}\mathbf{u}_1 & \mathbf{u}_2 & \dots & \mathbf{u}_k\end{bmatrix}}_{\text{Col } A} \underbrace{\begin{bmatrix}\mathbf{u}_{k+1} & \dots & \mathbf{u}_m\end{bmatrix}}_{\text{Nul } A^T} \begin{bmatrix}\sigma_1 & 0 & \dots & 0 & 0 & \dots & 0 \\ 0 & \sigma_2 & \dots & 0 & 0 & \dots & 0 \\ \dots & & & & & & \\ 0 & 0 & \dots & \sigma_k & 0 & \dots & 0 \\ 0 & 0 & \dots & 0 & 0 & \dots & 0 \\ \dots & & & & & & \\ 0 & 0 & \dots & 0 & 0 & \dots & 0\end{bmatrix} \begin{bmatrix}\mathbf{v}_1^T \\ \mathbf{v}_2^T \\ \dots \\ \mathbf{v}_k^T \\ \mathbf{v}_{k+1}^T \\ \dots \\ \mathbf{v}_n^T\end{bmatrix} \begin{matrix}\left.\vphantom{\begin{matrix}a\\a\\a\\a\end{matrix}}\right\} \text{Row } A \\ \left.\vphantom{\begin{matrix}a\\a\\a\end{matrix}}\right\} \text{Nul } A\end{matrix}$$

图 7：CUR.

Recommend Systems

Formal Model

- X = set of Customers
- S = set of Items
- R = set of ratings
  - Utility function u: X × S ⟶ R
- R is a totally ordered set
- e.g., 0-5 stars, real number in [0,1]

Example:

	Avatar	LOTR	Matrix	Pirates
Alice	1		0.2	
Bob		0.5		0.3
Carol	0.2		1	
David				0.4

表 2: Utility Matrix

问题

- **Gathering “known” ratings for matrix**
  - How to collect the data in the utility matrix
- **Extrapolate(推断) unknown ratings from the known ones**
  - Mainly interested in high unknown ratings
  - We are not interested in knowing what you don’t like but what you like
- **Evaluating extrapolation methods**
  - How to measure success/performance of recommendation methods

Gathering ratings

- Explicit
  - Ask people to rate items
  - Doesn’t work well in practice –people can’t be bothered
- Implicit
  - Learn ratings from user actions

Extrapolating Utilities

- **Key problem: Utility matrix U is sparse**
  - Most people have not rated most items
  - Cold start:(冷启动问题)
    - New items have no ratings
    - New users have no history
- Three approaches to recommender systems:
  - Content-based
  - Collaborative
    - User-User CF
    - Item-Item CF
  - Latent factor based

Content-based RS

Ex. Main idea: Recommend items to customer x similar to previous items rated highly by x

Item Profiles

- For each item, create an item profile
- Profile is a set (vector) of features
  - **Movies:** author, title, actor, director,⋯
  - **Text:** Set of “important” words in document
- How to pick important features?

  - Usual heuristic from text mining is TF-IDF(Term frequency * Inverse Doc Frequency)

TF-IDF

- f_{ij} = frequency of term (feature) i in doc (item) j
- $TF_{ij} = \frac{f_{i,j}}{max_k f_{kj}}$
- n_i = number of docs that mention term i
- N = total number of docs
- TF-IDF score: $w_{ij} = TF_{ij} \times IDF_i$
- Doc profile = set of words with highest TF-IDF scores, together with their scores

User Profile

- Prediction heuristic:
  - Given user profile x and item profile i, estimate

$$u(x, i) = cos(x, i) = \frac{x \cdot i}{||x|| \cdot ||i||}$$

Pros/Cons Of Content based RS

- Pros.
  - No need for data on other users
    - No cold-start or sparsity problems
  - Able to recommend to users with unique tastes
  - Able to recommend new & unpopular items
  - Able to provide explanations
- Cons.
  - Finding the appropriate features is hard
  - Recommendations for new users
    - How to build a user profile?
  - Overspecialization
    - Never recommends items outside user’ s content profile
    - People might have multiple interests
    - Unable to exploit quality judgments of other users

Collaborative Filtering

User-User

- 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

Finding ”Similar” Users

- Let r x be the vector of user x’ s ratings

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

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

Example: Similarity Metric

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

表 3: Similarity Metric

- 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

表 4: subtract the (row) mean

Rating Prediction

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:
  - $r_{xi} = \frac{1}{k}\Sigma_{y \in N} r_{yi}$
  - $r_{xi} = \frac{\Sigma_{y \in N} s_{xy} \cdot r_{yi}}{\Sigma_{y \in N} s_{xy}}$

Item-Item CF

- 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

$$r_{xi} = \frac{\Sigma_{y \in N} s_{xy} \cdot r_{yi}}{\Sigma_{y \in N} s_{xy}}$$

Example: Item-Item CF

	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	

表 5: Example:

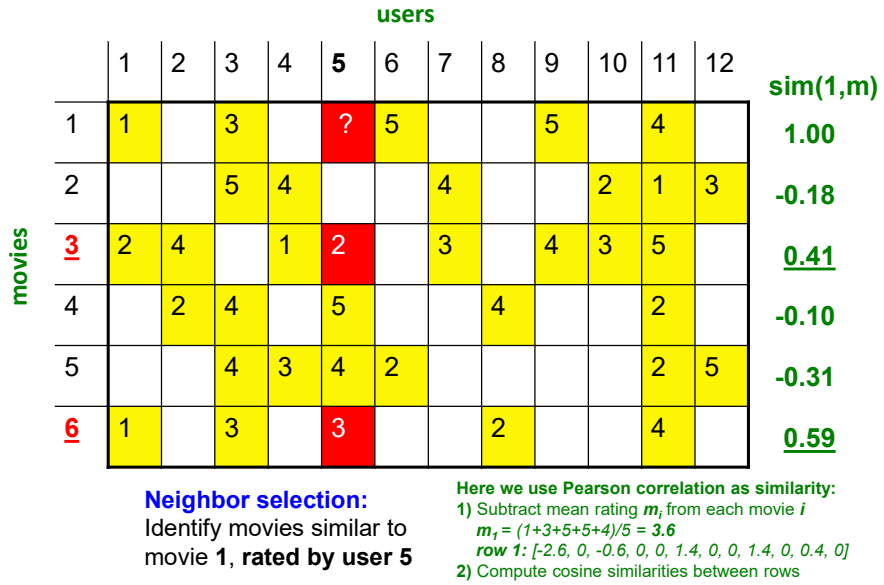

图 8：Item-Item CF

CF: Common Practice

- Define similarity sij of items i and j
- Select k nearest neighbors N(i; x)
- Items most similar to i, that were rated by x
- Estimate rating rxi as the weighted average:

$$\hat{r}_{xi} = \frac{\Sigma_{y \in N(i;x)} S_{xy} \cdot r_{xj}}{\Sigma_{j \in N(i;x)} S_{ij}}$$

Item-Item vs. User-User

- 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
- - 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

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

Evaluation

- Compare predictions with known ratings
  - Root-mean-square error (RMSE)
    - $\sqrt{\Sigma_{xi}(r_{xi} - r^{\star}_{xi})^2}$,where r_{xi} is predicted, $r^{*}_{xi}$is the true rating of x on i
  - Precision at top 10:
    - % of those in top 10
  - Rank Correlation:
    - Spearman's correlation between system's and user's complete rankings
  - Another approach: 0/1 model
    - Coverage:
      - Number of items/users for which system can make predictions
    - Precision:

- Accuracy of predictions
- Receiver operating characteristic (ROC)
- Tradeoff curve between false positives and false negatives

Latent Factor Models

In practice we get better estimates if we model deviations:

$$\hat{r}_{xi} = b_{xj} + \frac{\Sigma_{y \in N(i;x) S_{xy} \cdot (r_{xj} - b_{xj})}}{\Sigma_{j \in N(i;x)} S_{ij}}$$
$$b_{xj} = \mu + b_x + b_i$$

- μ = overall mean rating
- bx = rating deviation of user x = (avg. rating of user x) $^-\mu$
- b_i = (avg. rating of movie i) $^-\mu$

Problems/Issues:

- Similarity measures are "arbitrary"
- Pairwise similarities neglect interdependencies among users
- Taking a weighted average can be restricting
- Solution: Instead of sij use wij that we estimate directly from data
- Use a weighted sum rather than weighted avg.:

$$\hat{r}_{xi} = b_{xi} + \Sigma_{j \in N(i;x)} w_{ij} (r_{xj} - b_{xj})$$

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

Recommendations via Optimization

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

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

Latent Factor Models

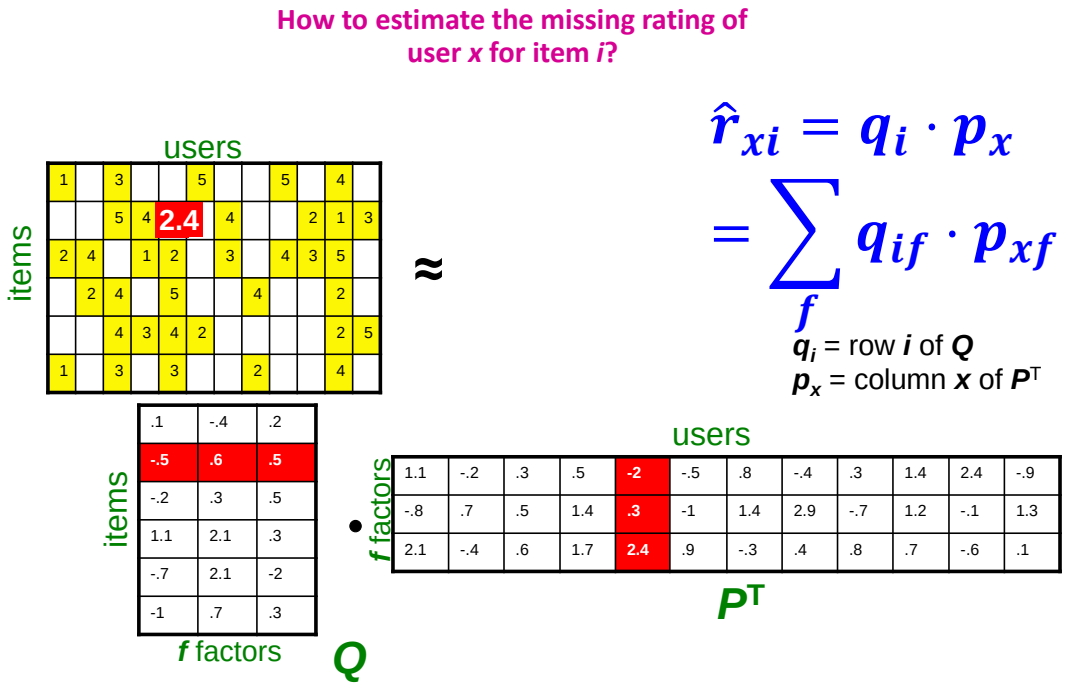

图 9：Ratings as Products of Factors

Dealing with Missing Entries

- To solve overfitting we introduce **regularization**:
- Allow rich model where there are sufficient data
- Shrink aggressively where data are scarce

$$min_{p,q}\Sigma_{training}(r_{xi} - q_i p_x)^2 + [\lambda_1 \Sigma_x ||p_x||^2 + \lambda_2 \Sigma_i ||q_i||^2]$$

- λ_1, λ_2 ⋯user set regularization parameters
- Note: We do not care about the "raw" value of the objective function,but we care in P,Q that achieve the minimum of the objective

Stochastic Gradient Descent

- Want to find matrices P and Q:

$$min_{p,q}\Sigma_{training}(r_{xi} - q_i p_x)^2 + [\lambda_1 \Sigma_x ||p_x||^2 + \lambda_2 \Sigma_i ||q_i||^2]$$

  - Gradient decent:
  - Initialize P and Q (using SVD, pretend missing ratings are 0)
  - Do gradient descent:
    * $P \leftarrow P - \eta \cdot \nabla P$
    * $Q \leftarrow Q - \eta \cdot \nabla Q$
    * where ∇Q is gradient/derivative of matrix
    * Here q_{if} is entry f of row q_i of matrix Q
    * Observation: Computing gradients is slow

Gradient Descent (GD) vs. Stochastic GD

- GD: $Q \leftarrow Q - \eta[\Sigma_{r_{xi}} \nabla Q(r_x i)]$

- SGD: $Q \leftarrow Q - \mu \nabla Q(r_{xi})$
  - Faster convergence!
    * Need more steps but each step is computed much faster
- 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!