CS520 Web Programming
Recommendation Systems

Chengyu Sun
California State University, Los Angeles

Recommendation Systems

- *Predict* items a user may be interested in based on information about the user and the items
- An effective way to help people cope with information overload
- Examples: Amazon, Netflix, Tivo, …

So How Can We Do It?

- The content based approach
  - E.g. full text search results
- The user feedback based approach
  - E.g. rating
- *Which one is better?? Any room for improvement??*

Collaborative Filtering

- Rate items based on the ratings of other users *who have similar taste as you*

Problem Definitions

- Prediction
  - Given: a user and k items
  - Return: predicted rating for each item
- Recommendation
  - Given: a user
  - Return: k items from the database with the highest predicted rating

Basic Assumptions

- Items are evaluated by users explicitly or implicitly
  - Ratings, reviews
  - Purchases, browsing behaviors
  - …
- We may map explicit and implicit evaluations to a rating scale, e.g. 1-5.

Heuristic

- People who agreed in the past are likely to agree in the future

Problem Formulation

- User-Item Matrix

Item	Ken	Lee	Meg	Nan
1	1	4	2	2
2	5	2	4	4
3			3	
4	2	5		5
5	4	1		1
6	??	2	5	

So what would be Ken's rating for Item 6??

Pearson Correlation Coefficient

- Let x and y be two users, and $r_{x,j}$ be the rating of item i by user x

$$w_{x,y} = \frac{\text{cov}(x,y)}{\sigma_x \sigma_u} = \frac{\Sigma_i (r_{x,i} - \bar{r}_x)(r_{y,i} - \bar{r}_y)}{\sqrt{\sum_i (r_{x,i} - \bar{r}_x)^2} \sqrt{\sum_i (r_{y,i} - \bar{r}_y)^2}}$$

So what is $w_{ken,lee}$?? what's the range of $w_{i,j}$?

Predicted Rating

- $p_{x,i}$ is the predicted rating of item i by user x

$$p_{x,i} = \bar{r}_x + \frac{\sum_u (r_{u,i} - \bar{r}_u) \times w_{x,u}}{\sum_u |w_{x,u}|}$$

So what is $p_{ken,6}$??

Variations and Optimizations

- Similarity measure
- Significance weighting
- Item rating variance
- Neighborhood selection
- Combine neighborhood ratings

Similarity Measures ...

- Pearson Correlation
- Spearman Correlation
  - Uses ranks instead of raw rating scores
- Cosine similarity
- Mean squared difference
- Entropy-based
- ...

... Similarity Measures

Cosine similarity: $\cos(\mathbf{X},\mathbf{Y}) = \frac{\mathbf{X}\bullet\mathbf{Y}}{\|\mathbf{X}\|\|\mathbf{Y}\|} = \frac{\sum x_i y_i}{\sqrt{\sum x_i^2}\sqrt{\sum y_i^2}}$

Mean squared difference: $msd(\mathbf{X},\mathbf{Y}) = \frac{\sum (x_i - y_i)^2}{N}$

Entropy-based association: $h(X,Y) = -\sum p_{i,j} \ln p_{i,j}$

Significance Weighting

- Weight users in additional to the similarity measure

$$w = \begin{cases} 1 & n \geq 50 \\ n/50 & n < 50 \end{cases}$$

where n is the number of items rated by both users.

Item Rating Variance

- Some items are more telling about tastes than others
  - E.g. "Sleepless in Seattle" is more telling about taste than "Titanic"
  - Give more weight to items with high variance in ratings

Neighborhood Selection

- Select a subset of users for better performance and *accuracy*
  - Correlation threshold
  - Best n neighbors

Combine Neighborhood Ratings

- Deviation from mean
- Weighted average
- Weighted average of z-scores

Mean absolute deviation: $s = \frac{1}{n}\sum_{i=1}^{n} |r_i - \bar{r}|$

Standardized measurement (*z-score*): $z_i = \frac{r_i - \bar{r}}{s}$

Algorithm Quality Metrics

- Coverage – percentage of items for which the system can produce a prediction
- Accuracy
  - Statistical metrics
    - Mean Absolute Error (MAE)
  - Decision-support metrics
- Efficiency
  - Throughput – number of recommendations per second

And The Winners Are

- Similarity measure
  - Pearson Correlation
  - Spearman Correlation*
- Significance weighting
- Neighborhood selection
  - Best n neighbors with $n \approx 20$
- Combine neighborhood ratings
  - Deviation from mean

Other Recommendation Algorithms

- Combine collaborative and content-based filtering
- Item-item collaborative filtering
- Bayesian networks
- ...

Some Libraries

- Taste – http://taste.sourceforge.net/
- COFE – http://eecs.oregonstate.edu/iis/CoFE/
- And more – http://en.wikipedia.org/wiki/Collaborative_filtering#Software_libraries

Non-personalized Recommendation

- What if the user is new to the site?
- What if the site itself is new, i.e. no previous user transactions?

Sales Transactions

t1:	Beef, Chicken, Milk
t2:	Beef, Cheese
t3:	Cheese, Boots
t4:	Beef, Chicken, Cheese
t5:	Beef, Chicken, Clothes, Cheese, Milk
t6:	Chicken, Clothes, Milk
t7:	Chicken, Clothes, Milk
t8:	Beef, Milk

Amazon-like recommendation:

Users who purchased milk also purchased the following items:

- *Clothes*
- *Chicken*

Support Count

- The support count, or frequency, of a itemset is the number of the transactions that contain the itemset
  - Item, Itemset, and Transaction
- Examples:
  - `support_count({beef})=5`
  - `support_count({beef,chicken,milk})=??`

Frequent Itemset

- An itemset is frequent if its support count is greater than or equals to a minimum support count threshold
  - `support_count(X)≥min_sup`
- Frequent itemset mining

The Apriori Property

- All nonempty subsets of a frequent itemset must also be frequent
- Or, if an itemset is not frequent, its supersets cannot be frequent either

Finding Frequent Itemsets – The Apriori Algorithm

- Given `min_sup`
- Find the frequent 1-itemsets L_1
- Find the the frequent k-itemsets L_k by joining the itemsets in L_{k-1}
- Stop when L_k is empty

Apriori Algorithm Example

beef	1
chicken	2
milk	3
cheese	4
boots	5
clothes	6

TID	Transactions
1	1, 2, 3
2	1, 4
3	4, 5
4	1, 2, 4
5	1, 2, 6, 4, 3
6	2, 6, 3
7	2, 6, 3
8	1, 3

- Support 25%

L_1

- Scan the data once to get the count of each item
- Remove the items that do not meet `min_sup`

C_1	support_count	L_1
{1}	5	{1}
{2}	5	{2}
{3}	5	{3}
{4}	4	{4}
{5}	1	
{6}	3	{6}

L_2

- $C_2=L_1 \times L_1$
- Scan the dataset again for the support_count of C_2, then remove non-frequent itemsets from C_2, i.e. $C_2 \rightarrow L_2$

C_2	support_count	L_2
{1,2}	3	{1,2}
{1,3}	3	{1,3}
{1,4}	3	{1,4}
{1,6}	1	
{2,3}	4	{2,3}
{2,4}	2	{2,4}
{2,6}	3	{2,6}
{3,4}	1	
{3,6}	3	{3,6}
{4,6}	1	

L_3

- ??

From L_{k-1} to C_k

- Let l_i be an itemset in L_{k-1}, and $l_i[j]$ be the jth item in l_i
- Items in an itemset are sorted, i.e. $l_i[1]<l_i[2]<\ldots<l_i[k-1]$
- l_1 and l_2 are joinable if
  - Their first k-2 items are the same, and
  - $l_1[k-1]<l_2[k-2]$

From C_k to L_k

- Reduce the size of C_k using the Apriori property
  - any (k-1)-subset of an candidate must be frequent, i.e. in L_{k-1}
- Scan the dataset to get the support counts

References

- *GroupLens: An Open Architecture for Collaborative Filtering of Netnews* by P. Resnick et. al, 1994.
- *An Algorithmic Framework for Performing Collaborative Filtering* by J. Herlocker et. Al, 1999.
- *E-Commerce Recommendation Applications* by J. B. Schafer et. al, 2001.
- *Data Mining: Concepts and Techniques* by Jiawei Han and Micheline Kamber