CS6120: Intelligent Media Systems

Dr. Derek Bridge
School of Computer Science & Information Technology
UCC

Problems for User-Based Collaborative Recommenders

Efficiency

- The user-based kNN approach compares the user to every other user
  - finding the neighbours takes time and space that grows linearly with the number of users (and the number of items)
- Especially slow if many more users (e.g. millions) than items (e.g. thousands)

Coverage

- Sparsity: Typical users rate few items (e.g. hundreds)
  - difficult to find neighbours
  - so recommendations cannot always be made

Item-Based Collaborative Recommenders

- Amazon pioneered an alternative
  - item-based collaborative recommenders

Item-Item Similarity, $sim(i, j)$

- We can compute similarity between columns (item-item similarity) just as easily as we can compute similarity between rows (user-user similarity)

	Alien	Brazil	Crash	Dumbo	E.T.	Fargo
Ben		2	5	3	1	2
Clare	5	5		3	4	
Dan					3	
Edd	5	4	2	4	3	3
Flo	2	5	4	4		

- Item are likely to have more ratings than user's have
  - so easier to find neighbours

Collaborative Recommenders

User-Based

- Find users who are similar to u
  - user-user similarity
- Predict u's rating for i based on the neighbours' ratings for i

Item-Based

- Find items in u's profile that are similar to i
  - item-item similarity
- Predict u's rating for i based on u's ratings for these items

- Item-Based Collaborative Recommending uses an algorithm that is similar to the kNN algorithm we used for Content-Based Recommending
  - except that the similarity uses ratings, not descriptions

kNN for Item-Based Collaborative Recommending

for each candidate item, i
  for each item in the user's profile, j
    compute $sim(i, j)$
  let NN be the set of k nearest neighbours, i.e. the k items in the profile that are most similar to i and whose similarity to i is positive
  let the predicted rating for item i be the average of the ratings in NN
recommend the candidates in descending order of predicted rating

We saw nearly the same algorithm for content-based recommending

Notation

$\mathbb{U}$	the set of all users
$\mathbb{I}$	the set of all items
$\mathbb{U}_i$	the set of users who have rated item i
$\mathbb{U}_{i,j}$	the set of users who have rated both item i and item j
$\mathbb{I}_u$	the set of items that have been rated by user u
$\mathbb{I}_{u,v}$	the set of items that have been rated by both user u and user j
$r_{u,i}$	user u's *actual* rating for item i
$\widehat{r_{u,i}}$	user u's *predicted* rating for item i

Item-Item Similarity, $sim(i,j)$

- Many systems use an *adjusted cosine correlation*

$$\frac{\sum_{v\in\mathbb{U}_{i,j}}(r_{v,i}-\overline{r_v})(r_{v,j}-\overline{r_v})}{\sqrt{\sum_{v\in\mathbb{U}_{i,j}}(r_{v,i}-\overline{r_v})^2}\sqrt{\sum_{v\in\mathbb{U}_{i,j}}(r_{v,j}-\overline{r_v})^2}}$$

- Notice how it is computed over all users v who have rated both i and j, $\mathbb{U}_{i,j}$
  - $\overline{r_v}$ is the average over all of user v's ratings

Example: $sim(Alien, Brazil)$

$$\frac{\sum_{v\in\mathbb{U}_{i,j}}(r_{v,i}-\overline{r_v})(r_{v,j}-\overline{r_v})}{\sqrt{\sum_{v\in\mathbb{U}_{i,j}}(r_{v,i}-\overline{r_v})^2}\sqrt{\sum_{v\in\mathbb{U}_{i,j}}(r_{v,j}-\overline{r_v})^2}}$$

	Alien	Brazil	Crash	Dumbo	E.T.	Fargo
Ben		2	5	3	1	2
Clare	5	5		3	4	
Dan					3	
Edd	5	4	2	4	3	3
Flo	2	5	4	4		

Similarity: Niceties

- Again, could normalize all ratings in advance
- Again, small possibility that the divisor will be zero
  - in which case, take the similarity to be zero
- But we don't normally need any Significance Weighting

u's Predicted Rating for i, $\widehat{r_{u,i}}$

- Following the same reasoning as before, we take a weighted average:

$$\widehat{r_{u,i}} = \frac{\sum_{j \in NN} sim(i,j) \times r_{u,j}}{\sum_{j \in NN} sim(i,j)}$$

- (This time NN is not a set of the most similar users who have rated i; it is the set of most similar items rated by u)
- There is no need to adjust here for user's mean ratings because the only ratings used here come from the same user

Example, part I

	Alien	Brazil	Crash	Dumbo	E.T.	Fargo
Ben		2	5	3	1	2
Clare	5	5		3	4	
Dan					3	
Edd	5	4	2	4	3	3
Flo	2	5	4	4		
Ann	**2**			**4**	**3**	**5**

- We'll predict Ann's rating for Brazil
- Question: What similarities will we compute? Why?

Example, part II

- The similarities
  - $sim(Brazil, Alien)$ = -0.23
  - $sim(Brazil, Dumbo)$ = -0.26
  - $sim(Brazil, E.T.) = 0.28$
  - $sim(Brazil, Fargo) = 0.18$
- Suppose $k = 3$
- Brazil's 3 nearest neighbours are E.T, Fargo and Alien
- But we ignore Alien. Why?

Example, part III

	Alien	Brazil	Crash	Dumbo	E.T.	Fargo
Ann	2			4	3	5

$$\widehat{r_{u,i}} = \frac{\sum_{j \in NN} sim(i,j) \times r_{u,j}}{\sum_{j \in NN} sim(i,j)}$$

User-Based vs Item-Based

User-based

- User-based seems 'truer' to word-of-mouth recommendations
- Typically more accurate if there are fewer users than items, e.g. a research paper recommender
- People think
  - it is more personalized (probably not true)
  - more serendipity (possibly true)

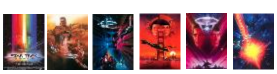
Item-based

- Item-based is often more efficient
  - you are only computing the similarity between i and each item that u has rated
  - more amenable to pre-computation because item-item similarity is likely to be more stable than user-user similarity – why?
- Typically more accurate if there are many more users than items

DIVERSITY

Diversity

- Suppose the items in a recommendation list are similar to each other
  - this reduces the chance that one of the recommendations will satisfy the user



- In many cases, we should seek to recommend items that
  - are similar to the user's preferences
  - but are *different from each other*
- *Diversity* is a property
  - not of a single item
  - but of the recommendation list as a whole

Bounded Greedy Selection

- Suppose we want to recommend n items
- Get a larger set of recommendations ($b \times n$) from your recommender
- Then select n from these, one by one, ensuring that each one we select has
  - high predicted rating
  - but also low total similarity to those already selected

Bounded Greedy Selection Algorithm

let *Recs* = $b \times n$ items with highest predicted ratings (found using any recommender you like)
let *Result* = [ ]
do the following n times
  let *best* = the member of *Recs* for which $Quality(i, Result)$ is highest
  insert *Best* into *Result*
  remove *Best* from *Recs*
recommend *Result*

Quality for Diversification

$$Quality(i, Result) = \widehat{r_{u,i}} + \alpha \times RelDiversity(i, Result)$$

$$RelDiversity(i, Result) = \begin{cases} 1 & \text{if } Result \text{ is empty} \\ \frac{\sum_{j \in Result}(1 - sim(i,j))}{size(Result)} & \text{otherwise} \end{cases}$$

- This require us to calculate $sim(i, j)$
  - can be done based on item descriptions, if available
  - or can be done as per item-based collaborative recommender
- What is the purpose of α?

HYBRIDS RECOMMENDERS

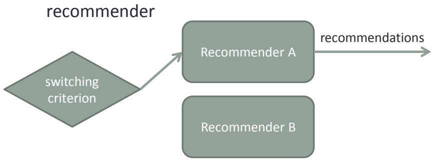
Hybrid Recommenders

- A hybrid combines different types of recommenders
- Main motivation
  - the recommenders compensate for each others' weaknesses

- E.g. a recommender that
  - tries to use kNN for a user-based collaborative recommendation
  - but when this can't be done (why?), it resorts to a non-personalised recommendation

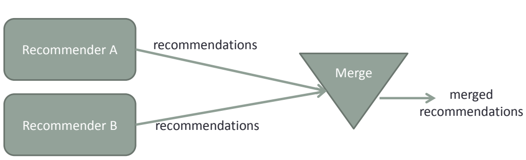
Switching

- Switching
  - decide on the best recommender
    - e.g. how many ratings the user has
  - run the selected recommender
- Issues
  - requires a reliable switching criterion



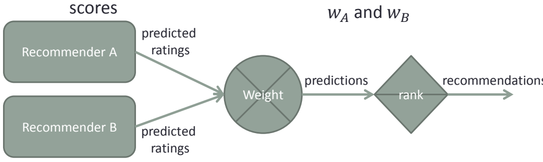
Mixing

- Mixing
  - run all recommenders
  - merge their recommendations
- Issues
  - how to merge ranked lists
    - e.g. Borda count



Weighting

- Weighting
  - run all recommenders
    - obtain predicted ratings or recommendations with confidence scores
  - combine the ratings or scores
- Issues
  - how to combine the scores
    - e.g. a linear formula $r_{u,i} = w_A r_{A,u,i} + w_B r_{B,u,i}$
  - how to assign weights, w_A and w_B



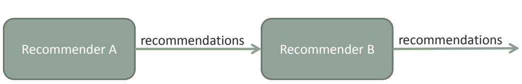
Netflix Prize Winners

- *BelKor's Pragmatic Chaos* won the Netflix prize

- Their solution uses weighting:
  - over 500 algorithms that predict ratings, including
    - user-based and item-based nearest-neighbour methods
    - matrix factorization
    - Restricted Boltzmann Machines
  - the weights for combining are learned from the data

Cascading

- Cascading
  - run a pipeline of recommenders
  - B filters or re-scores or re-ranks output of A
- Issues
  - B does not introduce new recommendations so A needs to be a strong recommender



Other 'Hybrids'

- There are other ways of combining recommenders
  - take knowledge used by one kind of recommender and make it available to another
  - so you only run one recommender

- E.g.
  - create pseudo-users, e.g. one per movie director or genre
  - the Woody Allen pseudo-user has high ratings for his films
  - use these pseudo-users in a normal collaborative recommender
    - Woody Allen lovers will be similar to the Woody Allen pseudo-user
  - we don't have to wait for real users to rate a new Woody Allen film