TencentRec: Real-time Stream Recommendation in Practice

Main Algorithm Design

Similarity Definition

Assume we have n users and m items in the recommender system, the ratings of users for items can be expressed as an $n * m$ matrix R, where each $r_{p,q}$ represents the user u_p's rating for item i_q.

For a user u, the co-rating of two items is defined as:

$$\text{co-rating}(i_p, i_q) = \min(r_{u,p}, r_{u,q})$$

The similarity of two items is calculated as follows:

$$sim(i_p, i_q) = \frac{\sum_{u \in U} \min(r_{u,p}, r_{u,q})}{\sqrt{\sum r_{u,p}}\sqrt{\sum r_{u,q}}}$$

For predicting a user's rating for an item, this formula is used:

$$\hat{r}_{u,p} = \frac{\sum_{i_q \in N^k(i_p)} sim(i_p, i_q) r_{u,i_q}}{\sum_{i_q \in N^k(i_p)} sim(i_p, i_q)}$$

where $N^k(i_p)$ is the set of k neighbors, i.e., the k items most similar to i_p.

Implicit Feedback

User's rating to an item is assigned based on their actions (e.g., viewed, clicked, liked, saved, bought)

Handling Incremental Updates

We consider changes of user rating to items to be incremental (which is true in implicit feedback case). Based on above definition for similarity function, we can write it as follows:

$$sim(i_p, i_q) = \frac{\text{pairCount}(i_p, i_q)}{\sqrt{\text{itemCount}(i_p)}\sqrt{\text{itemCount}(i_q)}}$$

where

$$\text{itemCount}(i_p) = \sum r_{u,p}$$

$$\text{pairCount}(i_p, i_q) = \sum_{u \in U} \text{co-rating}(i_p, i_q)$$

Now when the value of $r_{u,p}$ is increased, the similarity scores are updated in this way:

$$sim(i_p, i_q)' = \frac{\text{pairCount}(i_p, i_q)'}{\sqrt{\text{itemCount}(i_p)'}\sqrt{\text{itemCount}(i_q)'}}$$

$$= \frac{\text{pairCount}(i_p, i_q) + \Delta\text{co-rating}(i_p, i_q)}{\sqrt{\text{itemCount}(i_p) + \Delta r_{u_p}}\sqrt{\text{itemCount}(i_q) + \Delta r_{u_q}}}$$

The below figure shows how this algorithm can be implemented for real-time purposes and in parallel.

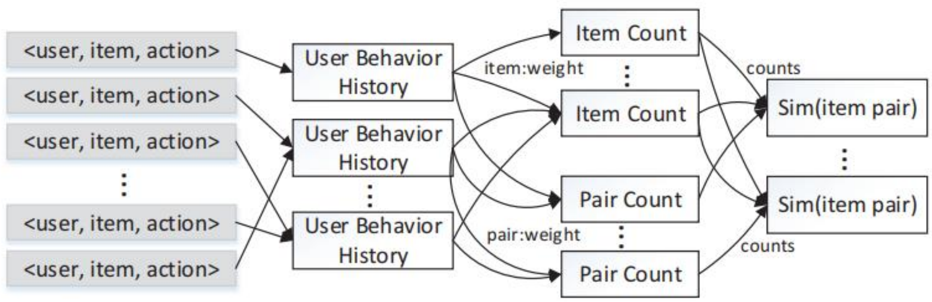

Figure 4: The Multi-layer Item-based CF

Real-time Pruning

We set a linked time for the item pairs that two items are considered as related only if users rate them together within a certain period. For recommendations in most situations such as e-commerce websites, the linked time is usually set to be three days or seven days, with nearly one hundred item pairs generated for each user action.

Not all of these item pairs are useful because a large portion of items are not in k-neighborhood ($N^k(i_p)$) of each other.

To solve this problem, we utilize the Hoeffding bound theory and develop a real-time pruning technique. It can be expressed as follows: let x be a real-valued random variable whose range is R (for the similarity score the range is one). Suppose we have made n independent observations of this variable, and computed their mean $\hat{x}$. The Hoeffding bound states that, with probability $1 - \delta$, the true mean of the variable is at most $\hat{x} + \epsilon$, where:

$$\epsilon = \sqrt{\frac{R^2 ln(1/\delta)}{2n}}$$

Final Algorithm

Algorithm 1: Item-based CF Algorithm with Real-time Pruning

Input: user rating action recording user u and item i

Get L_i
for *each item j rated by user u* **do**
  if j *in* L_i **then**
    Continue
  end
  Update pairCount(i, j)
  Get itemCount(i) and itemCount(j)
  Compute $sim(i, j)$ using Equation 5
  Increment n_{ij}
  Get threshold t_1 of i's similar-items list
  Get threshold t_2 of j's similar-items list
  $t = min(t_1, t_2)$
  Compute ϵ using Equation 9
  if $\epsilon < t - sim(i, j)$ **then**
    Add j to L_i
    Add i to L_j
  end
end

Other Techniques

Real-time Filtering Mechanism

A sliding window is used to forget the older rating data. In TencentRec, we split the time window into several sessions and we just consider the W most recent sessions in the recommendation computation. For example, implementing the sliding window, the $r_{u,p}$s used to compute the itemCounts and pairCounts in real-time item-based CF will refer to the ratings given by user u in recent W sessions, as follows:

$$sim(i_p, i_q) = \frac{\sum_{w \in W} \text{pairCount}_w(i_p, i_q)}{\sqrt{\sum_{w \in W} \text{itemCount}_w(i_p)}\sqrt{\sum_{w \in W} \text{itemCount}_w(i_q)}}$$

Beside the sliding window, for each user, we record the recent k items that he is interested in. Based on the perception that a user's interests fade away as time goes on, we believe only the recent k items are effective for the user's recommendation computation.

Data Sparsity Solution

#TODO

Implementational Details

#TODO