Open in app

Following

568K Followers

Photo by Glenn Carstens-Peters on Unsplash

Introduction To Recommender Systems- 1: Content-Based Filtering And Collaborative Filtering

How services like Netflix, Amazon, and Youtube recommend items to the users?

Abhijit Roy Jul 29, 2020 · 11 min read

We all have used services like Netflix, Amazon, and Youtube. These services use very sophisticated systems to recommend the best items to their users to make their

Component Procedures of a Recommender:

Recommenders mostly have 3 components:

1. **Candidate Generations:** This method is responsible for generating smaller subsets of candidates to recommend to a user, given a huge pool of thousands of items.

2. **Scoring Systems**: Candidate Generations can be done by different Generators, so, we need to standardize everything and try to assign a score to each of the items in the subsets. This is done by the Scoring system.

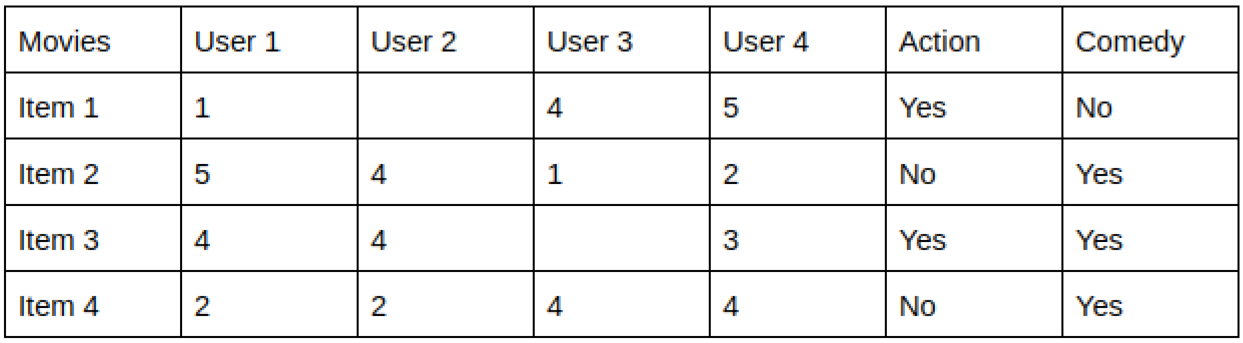
3. **Re-Ranking Systems:** After the scoring is done, along with it the system takes into account other additional constraints to produce the final rankings.

Types of Candidate Generation Systems:

1. Content-based filtering System

2. Collaborative filtering System

Content-based filtering system: Content-Based recommender system tries to guess the features or behavior of a user given the item's features, he/she reacts positively to.



Movies	User 1	User 2	User 3	User 4	Action	Comedy
Item 1	1		4	5	Yes	No
Item 2	5	4	1	2	No	Yes
Item 3	4	4		3	Yes	Yes
Item 4	2	2	4	4	No	Yes

The last two columns Action and Comedy Describe the Genres of the movies. Now, given these genres, we can know which users like which genre, as a result, we can obtain features corresponding to that particular user, depending on how he/she reacts to movies of that genre.

Once, we know the likings of the user we can embed him/her in an embedding space using the feature vector generated and recommend him/her according to his/her from his/her previous records. Then, the top few are recommended.

Content-based filtering does not require other users' data during recommendations to one user.

Collaborative filtering System: Collaborative does not need the features of the items to be given. Every user and item is described by a feature vector or embedding.

It creates embedding for both users and items on its own. It embeds both users and items in the same embedding space.

It considers other users' reactions while recommending a particular user. It notes which items a particular user likes and also the items that the users with behavior and likings like him/her likes, to recommend items to that user.

It collects user feedbacks on different items and uses them for recommendations.

Sources of user-item interactions

Implicit Feedback: The user's likes and dislikes are noted and recorded on the basis of his/her actions like clicks, searches, and purchases. They are found in abundance but negative feedback is not found.

Explicit Feedback: The user specifies his/her likes or dislikes by actions like reacting to an item or rating it. It has both positive and negative feedback but less in number

Types of collaborative Recommender Systems:

Memory-based collaborative filtering: Done mainly remembering the user-item interaction matrix, and how a user reacts to it, i.e, the rating that a user gives to an item. There is no dimensionality reduction or model fitting as such. Mainly two sections:

User-User filtering: In this kind, if a user A's characteristics are similar to some other user B then, the products that B liked are recommended to A. As a statement, we can say, "the users who like products similar to you also liked those products". So here we recommend using the similarities between two users.

$$R_{xu}=(\Sigma_{i=0}^{n}R_i) / n$$

Where Rxu is the rating given to x by user u and i=0 to n are the users who have shown behavior similar to u. Now, all the n users are not an equal amount similar to the user u. So, we find a weighted sum to provide the rank.

$$R_{xu}=(\Sigma_{i=0}^{n}R_iW_i) / \Sigma_{i=0}^{n}W_i$$

The weights here are the similarity metrics used.

Now, users show some differences in behaviors while rating. Some are generous raters, others are not, i.e, maybe one user rates in range 3 to 5, while other user rates 1 to 3. So, we calculate the average of all the ratings that the user has provided, and subtract the value from Ri in order to normalize the ratings by each user.

Item-Item filtering: Here, if user A likes an item x, then, the items y and z which are similar to x in property, then y and z are recommended to the user. As a statement, it can be said, “Because you liked this, you may also like those”.

The same equations are used here also

$$R_{xu}=(\Sigma_{i=0}^{n}R_i) / n$$

Where R is the rating user u gives to the product x, and it is the average of the ratings u gave to products like x. Here also, we take a weighted average

$$R_{xu}=(\Sigma_{i=0}^{n}R_iW_i) / \Sigma_{i=0}^{n}W_i$$

Where the Weight is the similarity between the products.

Similarity Metrics

They are mathematical measures which are used to determine how similar is a vector to a given vector.

1. Cosine Similarity: The Cosine angle between the vectors.
2. Dot Product: The cosine angle and magnitude of the vectors also matters.
3. Euclidian Distance: The elementwise squared distance between two vectors
4. Pearson Similarity: It is a coefficient given by:

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

Model-based collaborative filtering: Remembering the matrix is not required here. From the matrix, we try to learn how a specific user or an item behaves. We compress the large interaction matrix using dimensional Reduction or using clustering algorithms. In this type, We fit machine learning models and try to predict how many ratings will a user give a product. There are several methods:

1. **Clustering algorithms**
2. **Matrix Factorization based algorithm**
3. **Deep Learning methods**

Clustering Algorithms: They normally use simple clustering Algorithms like K-Nearest Neighbours to find the K closest neighbors or embeddings given a user or an item embedding based on the similarity metrics used.

Matrix Factorization based algorithms:

Idea: Like any big number can be factorized into smaller numbers, the user-item interaction table or matrix can also be factorized into two smaller matrices, and these two matrices can also be used to generate back the interaction matrix.

So, we generate the factor matrices as feature matrices for users and items. These feature matrices serve as embeddings for each user and item. To create the feature matrices we need dimensional reduction.

There are 4 users and 5 items, the users and items are placed according to a domain D1 say, genre if items are movies. We can say they are not very well separable and the whole thing looks very generalized.

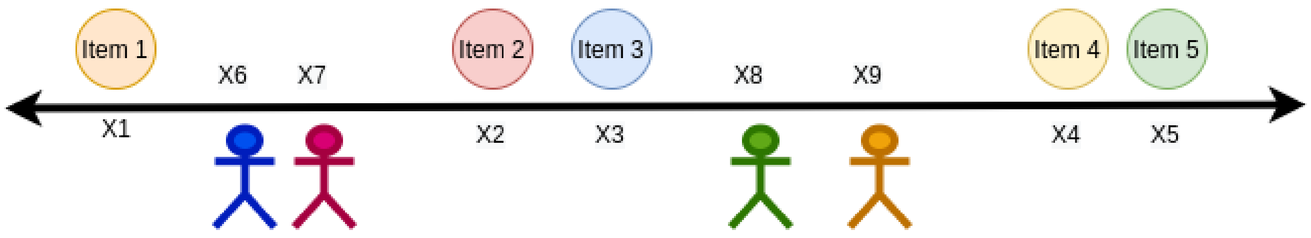
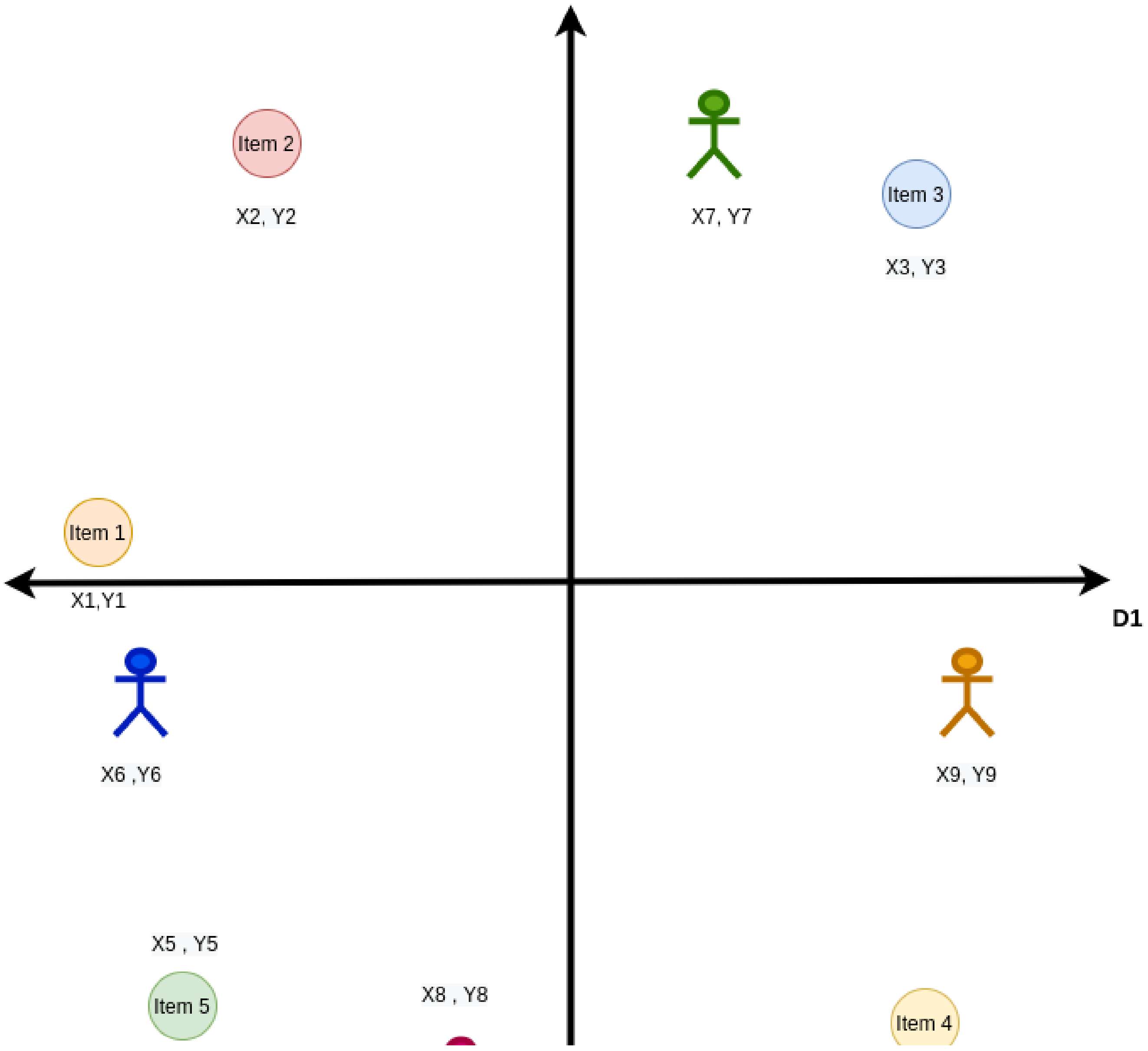


Image by Author

So, we increase the number of domains or add a domain, on whose basis we can classify the users and the items.



Open in app

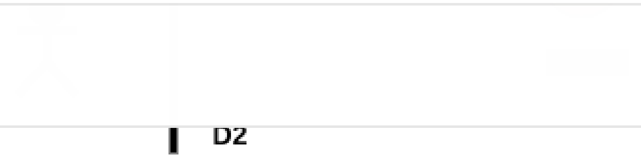
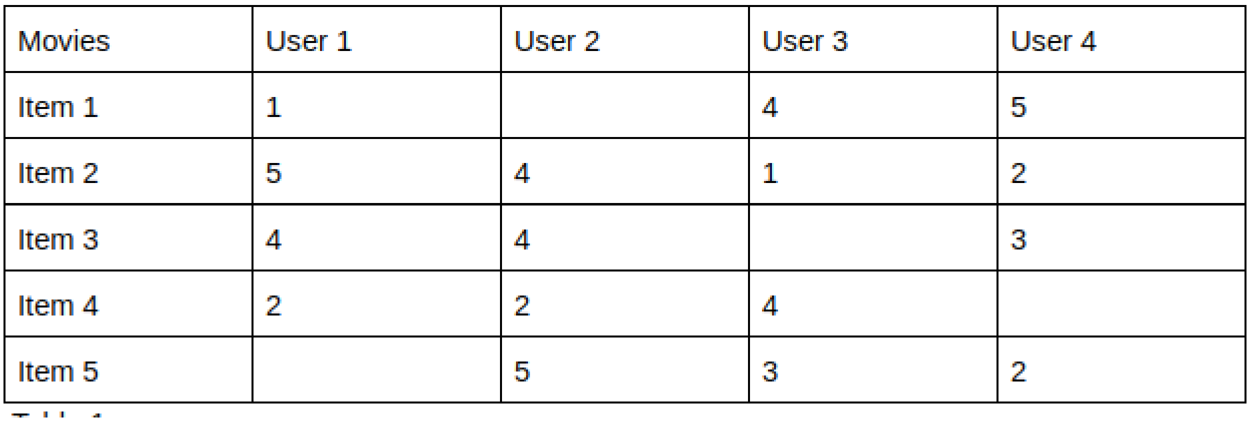


Image By Author

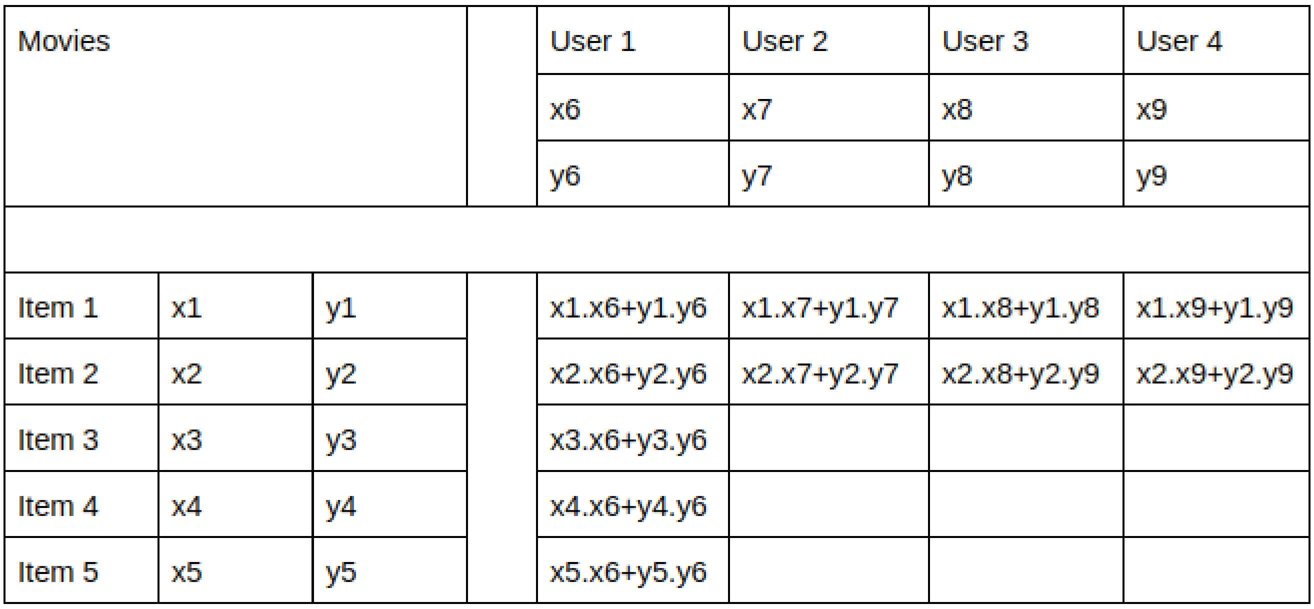
Now using these two domains we can easily classify the items and users properly, so, the (x,y) pairs can be used as their feature vectors or embeddings. Thus, our matrix is factorized.



Movies	User 1	User 2	User 3	User 4
Item 1	1		4	5
Item 2	5	4	1	2
Item 3	4	4		3
Item 4	2	2	4	
Item 5		5	3	2

Table 1

The above interaction table is converted into:



Movies				User 1	User 2	User 3	User 4
				x6	x7	x8	x9
				y6	y7	y8	y9
Item 1	x1	y1		x1.x6+y1.y6	x1.x7+y1.y7	x1.x8+y1.y8	x1.x9+y1.y9
Item 2	x2	y2		x2.x6+y2.y6	x2.x7+y2.y7	x2.x8+y2.y9	x2.x9+y2.y9
Item 3	x3	y3		x3.x6+y3.y6			
Item 4	x4	y4		x4.x6+y4.y6			
Item 5	x5	y5		x5.x6+y5.y6			

Table 2

So, our task is to find the (x,y) values in such a way that the numbers generated in table 2 are as close as the actual interaction matrix. Once we find all the (x,y) values we can to the user. Here 2 domains only x and y are shown actually there can be a very large number of domains. More the number of domain bigger the feature vector, bigger the embedding space.

Now, the number of features in the feature vectors depends on how many domains or features (a feature represented in a domain), we need to consider to distinctly represent the users and the items. So, we basically need to find the principal components of the user and items distributions. Finding principal components implies dimensionality reduction, i.e, representing a distribution distinctly using the least number of features possible.

The dimensionality reduction can be done by several methods:

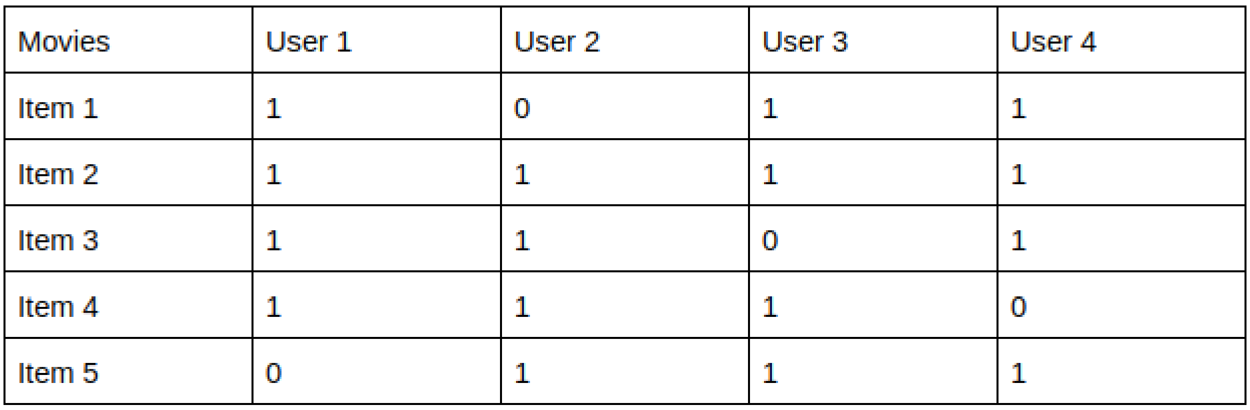
1. **SVD: Singular Value Decomposition**
2. **PMF: Probability Matrix Factorization**
3. **NMF: Non-Negative Matrix Factorization**

If we observe table 2, x1.x6+y1.y6, is the dot product of item 1 embedding vector multiplied by [x6, y6] i.e, transpose of the embedding vector of user 1.

So, each cell in table 2,

Rating to item u by user v= U.transpose(V)

Dimensionality Reduction: Creating Feature Vectors



Movies	User 1	User 2	User 3	User 4
Item 1	1	0	1	1
Item 2	1	1	1	1
Item 3	1	1	0	1
Item 4	1	1	1	0
Item 5	0	1	1	1

Table 3

of 0's much greater than the number of 1s. So, mostly they are sparse matrices.

Our objective function, i,e the one we need to minimize here is:

$$\text{Loss}= \Sigma_i\Sigma_j(A_{ij} - u_iv_j^T)^2 \quad \text{for } r(i,j)=1$$

r(i,j)=1 can be found from table 3. It implies to the users j who have reacted or rated items i. If r(i,j)=0 user j have not rated item i.

So, it means we are trying to minimize the difference between the original rating given A(i,j) from table 1 and the values obtained from table 2, multiplying the user and item's feature vectors. This helps to optimize the feature vectors.

For SVD,

$$\text{Loss}= \Sigma_i\Sigma_j(A_{ij} - u_iv_j^T)^2$$

where Aij is the cell wise rated value in table 1 and u, v are the values generated by the algorithm. Now, due to the sparsity of the matrix SVD does not perform that well here.

SVD is given by R=UΣV, where U is the item matrix of dimension n x d, i.e, total n items and V has dimension m x d i.e, total m users, Σ is d x d diagonal matrix for multiplication Compatability. D is the size of the feature vector for each user and item also.

In SVD, the matrix is decomposed as

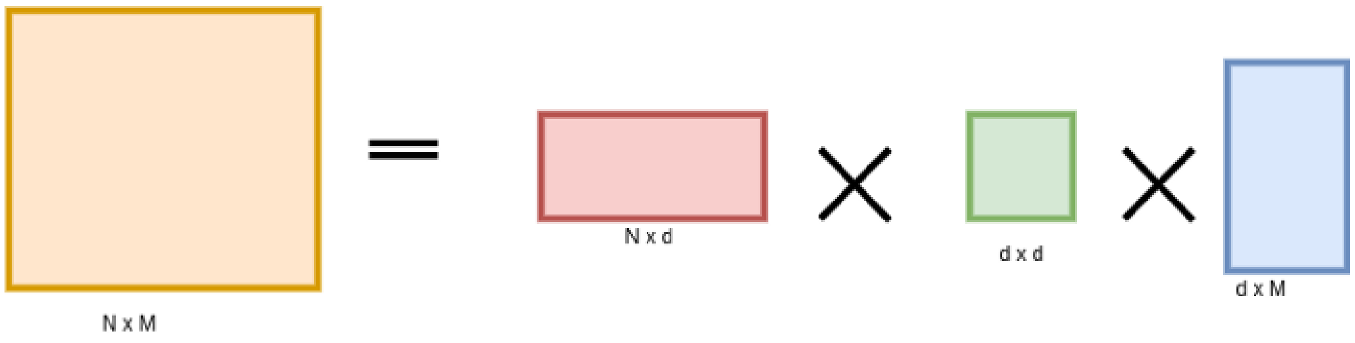


Image by author

The middle one is the diagonal vector.

NMF: Non-negative matrix factorization is so-called because the matrix here has no negative components, i.e, ratings can never be negative. The ones not rated is considered as 0's

Movies	User 1	User 2	User 3	User 4
Item 1	1	0	4	5
Item 2	5	4	1	2
Item 3	4	4	0	3
Item 4	2	2	4	0
Item 5	0	5	3	2

Table 4

So, NMF uses only the observed or rated ones. So, it modifies the function as:

$$\text{Loss}= \Sigma_i\Sigma_j(A_{ij} - U_iV_j^T)^2 \quad \text{for } r(i,j)=1 \text{ or } (i,j) \in (\text{obs/rated})$$

This performs better with sparse matrices.

It breaks the vector as:

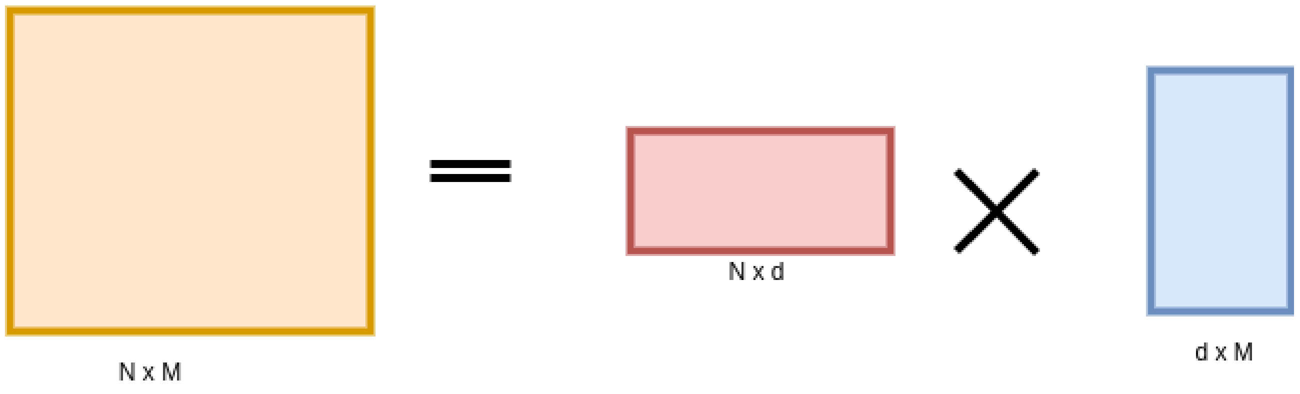


Image By author

Where N is the number of users, M is the number of items and d is the dimension or size of the feature vector.

$$\text{Loss}=W_0\ \Sigma_i\Sigma_j(A_{ij}-U_iV_j^T)^2 \text{ for } r(i,j)=1 \text{ or } (i,j)\in(\text{rated}) + W_1\ \Sigma_i\Sigma_j(U_iV_j^T)^2 \text{ for } r(i,j)=0 \text{ or } (i,j)\notin(\text{rated})$$

Here, we include two rated terms w0 and w1 and try to optimize the rated as well as the non-rated ones. The non-rated ones are considered to be zero, so, ((Ui.Vj)-0)2 is used to optimize the non-rated or zero ones. W0 and W1 are hyperparameters, we need to choose carefully.

Minimizing the Objective function

The most common algorithm used to minimize the include:

Weighted Alternating Least Squares: If we concentrate on the problem, We can find two separate problems: FInding the optimal embeddings that best describe the items and also the optimal embeddings that best describe the users.

Now, let’s decompose and see the problems individually,

1. If we have the feature matrices or vectors for the items as we did for the Content-Based systems, we can easily find the user embeddings or feature sets by noting how a user reacts or rates items having different feature vectors.

So, our target function becomes:

$$\Sigma_i^m\Sigma^n_j(A_{ij}-U_iV_j^T)^2 \text{ for } r(i,j)=1 \text{ or } (i,j)\in(\text{rated}) +\lambda/2\ \Sigma^n_j\ (V_j)^2$$

Where Vj is the user’s feature vector and the lambda term is the regularization term for optimizing the user embeddings, given the item embedding or feature vector Ui for each item i.

2. Now, if we have the user features or know how a user behaves or reacts, we can optimally find the item’s feature or embedding vectors, from the way each user reacts or rates the item.

Our target function for this:

$$\Sigma^n_j\Sigma_i^m(A_{ij}-U_iV_j^T)^2 \text{ for } r(i,j)=1 \text{ or } (i,j)\in(\text{rated}) +\lambda/2\ \Sigma_i^m(U_i)^2$$

Now, In matrix factorization, we need to find both U and V. So, the algorithm WALS works by alternating between the above two equations.

- Fixing U and solving for V.
- Fixing V and solving for U.

Now, the problem is these two equations are not convex at the same time. Either equation 1 is convex or equation 2 but not combined. As a result, we can't reach a global minimum here, but it has been observed that reaching a local minimum close to the global minimum gives us a good approximation of the optimized results at the global minimum. So, this algorithm gives us an approximated result.

Challenges:

1. The prediction of the model for a given (user, item) pair is the dot product of the corresponding embeddings. So, if an item is not seen during training, the system can't create an embedding for it and can't query the model with this item. This issue is often called the **cold-start problem**.

This problem is generally solved using two methods:

1. Projection in WALS
2. Heuristics to generate embeddings for fresh items

2. The side features are hard to include. SIde features are the ones that may affect the recommendations like for a movie, the U/PG ratings can be side features or the country.

Conclusion

In this article, we have taken a look at the two basic types of filtering mechanisms. In my next articles, I will talk about deep learning-based collaborative filtering and try to go through some applications.

Happy learning!!!.

Recommendation System Matrix Factorization Collaborative Filtering Basics