Music Recommender System

Prakyath Madadi

[1]Net ID: pm3140

Abstract. *With ever-growing data being generated by the internet, it is important to construct algorithms that can extract useful information from it. Recommender system is one such algorithm that uses this information to recommend content to the users. Many companies in the age of Internet, like Spotify, Netflix, Amazon, etc, are heavily dependent on the recommender systems that keep the users engaged. From music recommendations to target advertising, everything revolves around recommender systems. In a way, it is also acting as a proxy for the recommendations which we usually get from social interactions. So, it is very important for the companies and also us to know how such a system works.*

In this project, I'm going to discuss about various recommender systems that are being used and specifically focus on the the Alternating least square method of collaborative filtering to recommend music to users.

1 Introduction

A recommender system is a filtering system that predicts the content with the highest chance of user engagement. It takes advantage of the interactions between the users and the content. These interactions can be something that users directly provide (an explicit source), like ratings, or can be extracted from the history (an implicit source), like plays, listening history, etc.

2 Types

There are different types of recommender systems. There are 3 main types namely Content based filtering, Collaborative filtering and Hybrid filtering, which is a mix of content and collaborative filtering.

I'll now discuss about the different types and try to justify why the ALS based Collaborative filtering algorithm works better for music recommendations compared to Content based and Memory based Filtering methods.

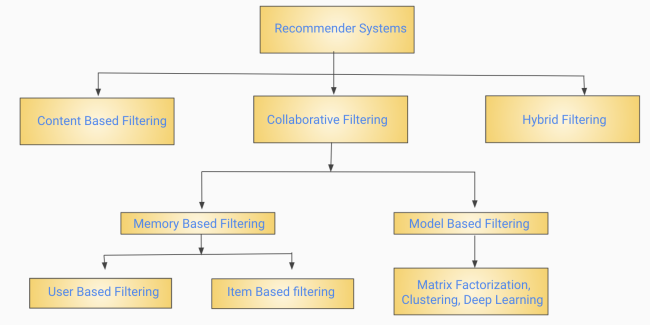


Figure 1: Types of Recommender systems

2.1 Content Based Filtering

The content based filtering algorithm tries to recommend songs/artists that are similar to what the user likes. It tries to predict the features for the user, considering we have the features for the music items, based on the user interactions. Based on the predicted features for the users, the systems tries to find the music items that are nearest to the features of the users. It doesn't consider the content that the users with similar features have interacted with, to recommend new content.

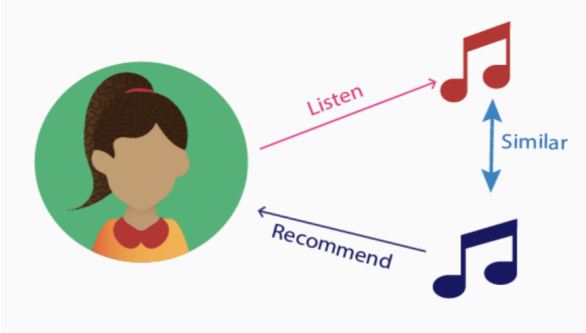


Figure 2: Content Based Filtering

2.2 Collaborative filtering

Collaborative filtering recommends songs/artists based on what the user previously liked and also users with similar taste. There are mainly two types: Memory based and Model based Collaborative filtering.

Memory based filtering stores the user interaction matrix and recommends based on it. There are two kinds of Memory based filtering methods: User Based Filtering and Item based filtering.

2.2.1 User Based Filtering

In user based filtering, users who are similar are recommended content which was liked by the other similar user. If we consider users w, x, y and z who are similar and x, y, z like an music item 'a', then the rating given by user w can be found as follows:

$$rating_{aw} = \frac{\sum_{i \epsilon x,y,z} rating_{ai} W_i}{\sum_{i \epsilon x,y,z} W_i} \quad (1)$$

where W is the weight which can be calculated using similarity between the user features. Similarity can be found using dot product, euclidean distance, etc. Rating_{ai} is the rating given to the item 'a' by user 'i'.

2.2.2 Item Based Filtering

In item based filtering, music items which are similar to the ones which the user liked are recommended to the users. It

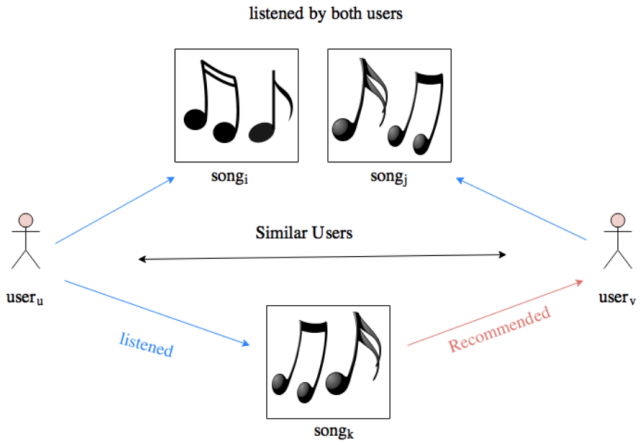


Figure 3: User Based Filtering

is similar to content based filtering in recommending similar items. If we consider user 'a' who likes item w and items x, y and z are similar to w, then rating given by user 'a' to item w can be found as follows:

$$rating_{wa} = \frac{\sum_{i \epsilon x,y,z} rating_i W_i}{\sum_{i \epsilon x,y,z} W_i} \tag{2}$$

where W is the weight which can be calculated using similarity between the item features. Rating$_i$ is the average rating given to the item 'i'

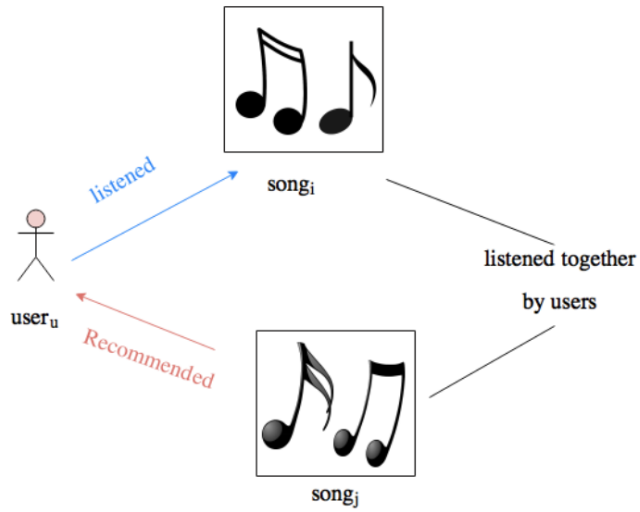


Figure 4: User Based Filtering

2.3 Model based Filtering

In model based filtering methods, the user interaction matrix is projected into two smaller dimensions using techniques like clustering and dimensionality reduction. There are many ways to do this: clustering based filtering, matrix factorization and neural network based recommendations.

The user interaction matrix is very sparse, especially for the music recommender case. There are millions of songs and millions of users, but a specific user would have listened or rated only a few hundred songs/artists. Most of the entries of the matrix are empty and it therefore useful to project the matrices into smaller dimensions.

There are other problems like the cold start problem, scalability and popularity bias which can be solved by using the model based filtering methods. These methods can be performed in a distributed sense dealing with the scalabilty issue. For new user/artist entries, the other algorithms tend to recommend the most popular choices which no longer exists as we are projecting the data to reduce the dimensions. The cold start problem can still be an issue but we can alter the algorithms slightly to deal with it.

3 Matrix factorization based Filtering

In matrix based filtering, the user interaction matrix is broken down into two low dimensional user and item feature matrices. Each row of the user matrix represents the features of a user and each column of the item matrix represents the features of the item. The new ratings given by a user to an item can be found by using the dot product between the embeddings. If 'U' is the item matrix and 'V' is the user matrix, then rating is defined as follows:

$$Rating_{ui} = \sum_{j=1}^{n} V_{uj} U_{ji} \tag{3}$$

where n is the number of dimensions for the user/item matrix. Rating$_{ui}$ is the rating given by user 'u' to item 'i'.

If 'R' is the initial ratings matrix and $\hat{R}$ be the ratings matrix formed from the dot product of user and item features. Let 'm' be the number of users and 'n' be the number of items. The Algorithm tries to minimize the difference between the ratings matrices.

$$\arg\min_{U,V} J : J = \sum_{i=1}^{m} \sum_{j=1}^{n} (R_{ij} - \hat{R}_{ij})^2 + \lambda ||U||_2 + \lambda ||V||_2 \tag{4}$$

$$\arg\min_{U,V} J : J = \sum_{i=1}^{m} \sum_{j=1}^{n} (R_{ij} - u_i v_j^T) + \lambda ||U||_2 + \lambda ||V||_2 \tag{5}$$

where λ is the regularization strength. The objective function 'J' can be solved using iterative optimization methods like gradient descent.

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

Figure 5: Ratings Matrix

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			

Figure 6: Ratings Matrix after Factorization

3.1 Alternating Least Squares

Alternating Least Sqaure (ALS) algorithm is an matrix factorization method with a slightly different objective function. The algorithm splits the objective function into two parts:

Part 1: Minimizing the objective to find the item feature matrix assuming that the user feature matrix is constant/known.

$$\arg\min_{U} J : J = \sum_{i=1}^{m}\sum_{j=1}^{n}(R_{ij} - u_i v_j^T) + \lambda \sum_{i=1}^{m} u_i^2 \quad (6)$$

Part 2: Minimizing the objective to find the user feature matrix assuming that the item feature matrix is constant/known.

$$\arg\min_{V} J : J = \sum_{i=1}^{m}\sum_{j=1}^{n}(R_{ij} - u_i v_j^T) + \lambda \sum_{j=1}^{n} v_i^2 \quad (7)$$

The algorithm keeps switching between the two objectives. This algorithm doesn't reach a global minimum but the local minimum achieved from optimizing them alternatively is very close to the global minimum.

The gradient descent can be preformed in a described sense and therefore makes it easier to scale the algorithm to large inputs. This is suitable for a music recommmender system as the input data can be very large.

To deal with the cold start problem using weighted version of the ALS algorithm. For every new user 'k', we can find an embedding for the user based on the few interactions with the items as follows:

$$\min_{v_k} ||R_k - v_k U|| \quad (8)$$

where v_k is the embedding for the user. The initial ratings can be the average embedding of similar users.

4 Data and Implementation

I've used a small subset of the million song dataset by Last.Fm. The data consists of list of users, artists, albums, artist ratings and album ratings. There are about 316,065 artist ratings for 10,281 artists. There are also about 175,413 album ratings for 6117 albums.

I'm using python and pyspark jupyter notebook to implement the ALS algorithm. The data is in 'json' format. We can use mongoDb to import and export data into the jupyter notebook if the size of the input is large.

The ALS has hyperparamters like 'rank', 'regParam' and 'maxIter' representing the number of features in the new matrix, the regularization strength and the max number of iterations. I fixed the number of iterations to '5' as the algorithm usually onverges before 5 iterations. I've tuned the other hyperparameters using cross validation. I search The range [2,4,8,12,16,24,30] for 'rank' and [0.01,0.05,0.1,0.25,0.5,1,2.5] for 'regParam'.

5 Results

I have found that the optimal 'rank' and 'regParam' are 4 and 0.25. I have used these optimal parameters to fit the model again and find the top 5 artists and albums for all users. Also, I found the probable top 5 users for the artists and albums.

user_idx	recommendations
1	[{7544, 3.4648542}, {8089, 3.4069488}, {8918, 3.389954}, {4243, 3.374133}, {1819, 3.3651853}]
2	[{1819, 3.4734209}, {6788, 3.4438586}, {10046, 3.4365757}, {8089, 3.4164448}, {3136, 3.4134703}]
3	[{7544, 3.572998}, {8918, 3.5232759}, {8089, 3.4610038}, {4243, 3.429198}, {7303, 3.4211671}]
4	[{4243, 3.460855}, {7544, 3.4414763}, {8089, 3.4352136}, {6788, 3.4141283}, {1819, 3.412143}]
5	[{1819, 3.4749312}, {6788, 3.4640107}, {8089, 3.4639509}, {7544, 3.4615295}, {7140, 3.3880632}]
6	[{1819, 3.4821591}, {6788, 3.4166138}, {8089, 3.4141784}, {10046, 3.412616}, {3136, 3.3967154}]
7	[{1819, 3.451817}, {6788, 3.3686082}, {8089, 3.3638687}, {7396, 3.3417702}, {7140, 3.316318}]
8	[{1819, 3.307413}, {7140, 3.2820246}, {3281, 3.2656436}, {7396, 3.241385}, {9058, 3.2397218}]
9	[{1819, 3.5749135}, {6788, 3.5678859}, {8089, 3.563372}, {7544, 3.5494065}, {4243, 3.5216446}]
10	[{6788, 3.3855686}, {1819, 3.3791702}, {8089, 3.3229852}, {3136, 3.2989779}, {10046, 3.2980933}]
11	[{6788, 3.5100856}, {8089, 3.3529837}, {7544, 3.3292365}, {1819, 3.3289943}, {4243, 3.327856}]
12	[{4243, 3.466939}, {7544, 3.4622812}, {8089, 3.4159293}, {6788, 3.409471}, {3136, 3.372313}]
13	[{7544, 3.321055}, {8089, 3.2933087}, {7694, 3.289783}, {1819, 3.2871528}, {6788, 3.2811153}]
14	[{6788, 3.4836507}, {8089, 3.4181724}, {1819, 3.4081907}, {7544, 3.4033754}, {4243, 3.3972945}]
15	[{6788, 3.7318196}, {7544, 3.6090531}, {8089, 3.605756}, {1819, 3.565579}, {4243, 3.557609}]
16	[{7544, 3.5039945}, {8089, 3.437168}, {8918, 3.4336588}, {7140, 3.3966186}, {1819, 3.3965044}]
17	[{1819, 3.4780521}, {8089, 3.46336}, {7544, 3.4569733}, {6788, 3.4369898}, {4243, 3.4125302}]
18	[{1819, 3.409272}, {8089, 3.380494}, {7544, 3.3656988}, {4243, 3.3528113}, {7317, 3.349691}]
19	[{6788, 3.4364657}, {7544, 3.4156857}, {8089, 3.4005911}, {4243, 3.3934436}, {1819, 3.3613064}]
20	[{4243, 3.4773178}, {7544, 3.470709}, {8089, 3.4408097}, {3136, 3.4128938}, {7317, 3.405713}]

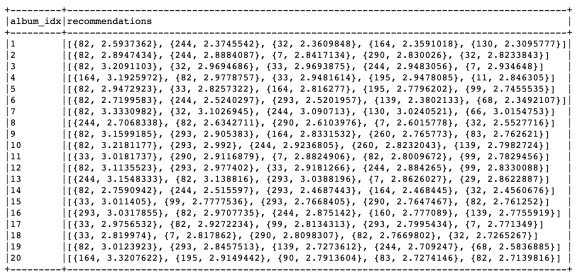
Figure 7: Top five artist recommendations to all users

artist_idx	recommendations
1	[{64, 2.884092}, {167, 2.8634398}, {189, 2.8480952}, {143, 2.8216105}, {272, 2.8073788}]
2	[{264, 2.7811399}, {189, 2.714577}, {144, 2.66073}, {71, 2.5928388}, {30, 2.5895445}]
3	[{189, 2.697942}, {264, 2.526031}, {144, 2.524995}, {131, 2.5183878}, {143, 2.4762745}]
4	[{64, 2.9404852}, {167, 2.923203}, {189, 2.9186113}, {143, 2.8987393}, {272, 2.851272}]
5	[{189, 3.1480453}, {264, 3.077175}, {144, 3.0213897}, {131, 2.972484}, {71, 2.938232}]
6	[{264, 3.763568}, {144, 3.3275576}, {30, 3.241963}, {151, 3.2176673}, {71, 3.2032735}]
7	[{189, 3.1044228}, {143, 2.8891954}, {131, 2.8633952}, {64, 2.8444715}, {43, 2.8209248}]
8	[{264, 3.5237117}, {144, 3.0860772}, {143, 3.06461}, {64, 3.0304457}, {151, 3.0045516}]
9	[{143, 3.0613923}, {64, 2.9367957}, {167, 2.84088816}, {294, 2.79311166}, {269, 2.765064}]
10	[{264, 3.4908469}, {151, 3.1962962}, {144, 3.1732826}, {189, 3.1644626}, {143, 3.152787}]
11	[{189, 2.5658092}, {264, 2.5379772}, {143, 2.4576797}, {144, 2.4489255}, {151, 2.4389088}]
12	[{143, 3.2258387}, {64, 3.1485415}, {167, 3.0361145}, {264, 2.9771037}, {189, 2.9705563}]
13	[{143, 2.851377}, {189, 2.817823}, {64, 2.8137417}, {264, 2.7377827}, {167, 2.7138677}]
14	[{189, 3.1127043}, {64, 3.0391598}, {167, 3.01114129}, {272, 2.9813826}, {143, 2.975492}]
15	[{264, 2.6821935}, {189, 2.4896502}, {143, 2.4782264}, {144, 2.4730852}, {64, 2.4636908}]
16	[{189, 3.3240476}, {143, 3.3008766}, {64, 3.295181}, {264, 3.2745268}, {167, 3.1791563}]
17	[{189, 2.4175165}, {131, 2.2400663}, {80, 2.2350767}, {64, 2.2104702}, {214, 2.1906555}]
18	[{264, 2.8485634}, {64, 2.7145987}, {143, 2.6723071}, {249, 2.634594}, {180, 2.6139188}]
19	[{264, 3.621532}, {143, 3.2480478}, {144, 3.2114146}, {151, 3.18923}, {189, 3.1394775}]
20	[{264, 3.296989}, {144, 2.7776203}, {30, 2.711888}, {143, 2.6899176}, {28, 2.6811585}]

Figure 8: Top five users prediction for all artists

user_idx	recommendations
1	[{3552, 3.4766335}, {5662, 3.4522738}, {4441, 3.3930807}, {3576, 3.3058467}, {546, 3.282145}]
2	[{3552, 3.7208512}, {2572, 3.4706404}, {1799, 3.3244174}, {5662, 3.3223758}, {4480, 3.3162722}]
3	[{3552, 3.4937115}, {5662, 3.3197756}, {229, 3.2247553}, {1522, 3.2152987}, {4480, 3.180952}]
4	[{3552, 3.6456823}, {5662, 3.3895822}, {229, 3.3698626}, {4491, 3.3664784}, {1522, 3.3604074}]
5	[{3552, 3.6469088}, {5662, 3.4438152}, {4441, 3.4265494}, {1548, 3.3235533}, {1382, 3.3176494}]
6	[{3552, 3.6184654}, {2572, 3.3113327}, {4441, 3.2970376}, {4435, 3.258345}, {1548, 3.2432368}]
7	[{3552, 3.824087}, {1382, 3.785201}, {4441, 3.7541711}, {4018, 3.7403776}, {1495, 3.7282395}]
8	[{3552, 3.7381964}, {2572, 3.5504458}, {4480, 3.4063163}, {501, 3.386284}, {2789, 3.334941}]
9	[{3552, 3.6470704}, {5662, 3.3300326}, {2572, 3.28463}, {1799, 3.275526}, {1548, 3.2710557}]
10	[{3552, 3.6892438}, {4018, 3.354984}, {2572, 3.3415124}, {4261, 3.313704}, {1495, 3.3128772}]
11	[{4441, 3.6657763}, {3552, 3.5442967}, {1127, 3.5015936}, {3860, 3.4173522}, {5662, 3.3879075}]
12	[{3552, 3.567127}, {5662, 3.4071715}, {229, 3.3473172}, {1522, 3.3402543}, {5333, 3.3103113}]
13	[{3552, 3.6670837}, {4441, 3.5063539}, {5662, 3.493185}, {4480, 3.4290495}, {1127, 3.3735294}]
14	[{3552, 3.7056987}, {5662, 3.5585682}, {229, 3.4604504}, {1522, 3.4520266}, {4480, 3.3939943}]
15	[{3552, 3.562554}, {5662, 3.363091}, {1382, 3.284894}, {1495, 3.2786744}, {4441, 3.2631855}]
16	[{3552, 3.6410174}, {4441, 3.4430418}, {5662, 3.4359245}, {4480, 3.357992}, {1548, 3.3185163}]
17	[{3552, 3.738447}, {2572, 3.4199002}, {5662, 3.3953695}, {4441, 3.3627248}, {1548, 3.354663}]
18	[{3552, 3.5945516}, {5662, 3.4072886}, {4480, 3.3286974}, {4441, 3.2887397}, {229, 3.285813}]
19	[{5662, 3.5696313}, {3552, 3.5689688}, {229, 3.4441211}, {1522, 3.4400396}, {3797, 3.4367657}]
20	[{3552, 3.615436}, {5662, 3.3589065}, {229, 3.3182864}, {1522, 3.307881}, {2572, 3.304473}]

Figure 9: Top five album recommendations to all users



album_idx	recommendations
1	[{82, 2.5937362}, {244, 2.3745542}, {32, 2.3609848}, {164, 2.3591018}, {130, 2.3095777}]
2	[{82, 2.8947434}, {244, 2.8884087}, {7, 2.8417134}, {290, 2.830026}, {32, 2.8233843}]
3	[{82, 3.2091103}, {32, 2.9694686}, {33, 2.9693875}, {244, 2.9483056}, {7, 2.934648}]
4	[{164, 3.1925972}, {82, 2.9778757}, {33, 2.9481614}, {195, 2.9478085}, {11, 2.846305}]
5	[{82, 2.9472923}, {33, 2.8257322}, {164, 2.816277}, {195, 2.7796202}, {99, 2.7455535}]
6	[{82, 2.7199583}, {244, 2.5240297}, {293, 2.5201957}, {139, 2.3802133}, {68, 2.3492107}]
7	[{82, 3.3330982}, {32, 3.1026945}, {244, 3.090713}, {130, 3.0240521}, {66, 3.0154753}]
8	[{244, 2.7068338}, {82, 2.6342711}, {290, 2.6103976}, {7, 2.6015778}, {32, 2.5527716}]
9	[{82, 3.1599185}, {293, 2.905383}, {164, 2.8331532}, {260, 2.765773}, {83, 2.762621}]
10	[{82, 3.2181177}, {293, 2.992}, {244, 2.9236805}, {260, 2.8232043}, {139, 2.7982724}]
11	[{33, 3.0181737}, {290, 2.9116879}, {7, 2.8824906}, {82, 2.8009672}, {99, 2.7829456}]
12	[{82, 3.1135523}, {293, 2.977402}, {33, 2.9181266}, {244, 2.884265}, {99, 2.8330088}]
13	[{244, 3.1548333}, {82, 3.138816}, {293, 3.0388196}, {7, 2.8626027}, {29, 2.8622887}]
14	[{82, 2.7590942}, {244, 2.515597}, {293, 2.4687443}, {164, 2.468445}, {32, 2.4560676}]
15	[{33, 3.011405}, {99, 2.7777536}, {293, 2.7668405}, {290, 2.7647467}, {82, 2.7612552}]
16	[{293, 3.0317855}, {82, 2.9707735}, {244, 2.875142}, {160, 2.777089}, {139, 2.7755919}]
17	[{33, 2.9756532}, {82, 2.9272234}, {99, 2.8134313}, {293, 2.7995434}, {7, 2.771349}]
18	[{33, 2.819974}, {7, 2.817862}, {290, 2.8098307}, {82, 2.7669802}, {32, 2.7265267}]
19	[{82, 3.0123923}, {293, 2.8457513}, {139, 2.7273612}, {244, 2.709247}, {68, 2.5836885}]
20	[{164, 3.3207622}, {195, 2.9149442}, {90, 2.7913604}, {83, 2.7274146}, {82, 2.7139816}]

Figure 10: Top five users prediction for all albums

6 Future scope

It is difficult to include side features like explicit ratings in this algorithm. Also, the feature extraction is purely based on ratings. We can improve our model by extracting the embeddings using neural networks. The networks are capable of extracting complex useful information from the interactions. We can accommodate side features and deal with the cold start problem if we use the neural based filtering.

References

[1] https://towardsdatascience.com/introduction-to-recommender-systems-1-971bd274f421

[2] https://towardsdatascience.com/prototyping-a-recommender-system-step-by-step-part-2-alternating-least-square-als-matrix-4a76c58714a1