TOP-N RECOMMENDER SYSTEM USING BIG DATA ITEM-TO-ITEM COLLABORATIVE FILTERING

Conference Paper · April 2018

CITATIONS
0

READS
81

3 authors, including:

Gautam Pal
University of Liverpool
10 PUBLICATIONS **9** CITATIONS
SEE PROFILE

Gangmin Li
Xi'an Jiaotong-Liverpool University
61 PUBLICATIONS **522** CITATIONS
SEE PROFILE

TOP-N RECOMMENDER SYSTEM USING BIG DATA ITEM-TO-ITEM COLLABORATIVE FILTERING

Gautam Pal, Gangmin Li, Katie Atkinson
Department of Computer Science,
University of Liverpool, UK

ABSTRACT

Recommender system in e-commerce provides a prominent way to enhance overall shopping experience by providing personalized or contextual advice through mining and discovering the interests and analyzing patterns of customers. This has made e-commerce more personal. No longer are products marketed to a mass audience, but to individuals based on their unique needs. It also has a significant role in generating revenue for the website by the fact that user tend to purchase if a recommended product is relevant to his need.

In this paper we introduce contextual item to item Collaborative Filtering an improved version popularized by Amazon [1], based on the concept of items also viewed under the same browsing session. Users' location is considered as the context for each user and the algorithm computes on users' implicit rating (by clicks). The proposed system designed to provide recommendation for non-authenticated (not logged in) user. Different Big Data ecosystem tools are considered as multi agent system in collaboration and final recommendation is calculated based on a proposed equation by weighted sum between near real time and historical batch data.

Keyword: Recommender system, Item to item Collaborative Filtering, Big Data, Analytics, Multi Agent System.

1. INTRODUCTION

RECOmender systems are a useful alternative to search algorithms since they help users discover items users might not have found by themselves. Recommenders adopts three types of approaches: (a) Content based (b) Collaborative (c) Hybrid. Content-based filtering methods are based on a description of the item and a profile of the user's preferences[2]. The system creates a content-based profile of users based on a weighted vector of item features. These are often combined with user's feedback to assign higher or lower weight to order the top n recommendation. For example, when a user views a mobile phone in the shopping portal, few other similar mobile phones is recommended for him. On the other hand, Collaborative Filtering methods [3-6] are based on collecting and analyzing a large amount of information on all users' behaviors, activities or preferences and predicting what users will like based on their similarity to other similar users or items. Underlying philosophy is on the assumption that people who agreed in the past will agree in the future. Hybrid approach makes content-based and collaborative-based predictions separately and blends them with weightage.

Recommender systems gets input from different sources to make recommendations. Most common way of collecting input is through user's feedback. Shopping portals for example collects ratings provided by the users. But explicit feedback may not be always

available. Thus number of attempts were taken to build recommendations through users implicit feedback[7].

In this paper, we propose a top n recommendation system(TNRS) by using a hybrid approach on users' implicit feedback in the form of item views. The contribution of the paper is as follows: (a) it interprets item to item collaboration on one browsing session by a user which in contrast to traditional user to user similarity approach; (b) to make recommendation relevant, TNRS extracts user's location of browsing as a latent factor. context aware recommendation tends to be more accurate; (c) this paper redefines the item to item relationship by putting more relevance to current trend than the historical data. (d) we use a hybrid approach to mix results from Collaborative Filtering with content based filtering on item similarity (e) finally based on above studies the paper presents a novel architecture of an end to end recommender system with host of online and offline Big Data eco system tools and their correlation as multi agent interaction model.

2. BACKGROUND AND MOTIVATION

In real-world applications it is common have access to implicit feedback in the form of views, clicks, purchases, likes, shares, comments etc.[8]. Still overwhelming majority of existing research just focus on users' explicit feedback (rating). The paper [7] proposed implicit feedback based recommender systems to improve customer experience through personalized recommendations depending on prior implicit feedback. In our work, we do not access rating provided by the user and do not access user's preferences specified during registration with the portal. We purely count on the implicit feedback in terms of the item views.

Majority of the existing approaches to recommender systems focus on recommending the most relevant items to individual users and do not take into consideration of any contextual information such as time, place etc. For example, in case of online shopping portals, buying pattern largely influenced by user's geographic location such as colder states vs wormer states, remote areas vs cities, browsing from desktop vs mobile etc. On the other words recommender systems deals with two types of entities: users and items but do not put them into a context when providing recommendation. Context here is a multifaceted concept that has been studied across various research disciplines including Computer Science, Cognitive Science, Linguistics, Philosophy, Psychology etc.[9-13]. This paper takes an approach to consider context information as a latent factor to provide more meaningful recommendations as depicted below:

Contextual Recommendation System: Users×Items×Context

⇨ Top n Recommender System (TNRS).

Existing Item-to-user rating based recommender system suffers from issues like user cold start, item cold start, sparsity, scalability etc. [14-16]. In user cold start, rating can't be predicted for a new item until the similar users has rated the same item. In case of sparsity, very few items are rated by users leading to a sparse item-user matrix and hence finding the similar users is difficult. For the scalability issue, as the dataset grows with time, filling the sparse matrix becomes gradually extremely compute intensive and each batch run takes longer time [14-16].

Proposed TNRS model overcomes the above said issues easily by adapting item-to-item collaborative filtering approach as it doesn't attempt to find similar users based on rating.

3. RELATED WORK

3.1 Event Capture Framework

In a context aware recommendation system, we design a Big Data event capture framework which ingests every end user click from a customer touch point. Customer's location and time is derived and appended in each session object which is associated with each time user newly opens the e-commerce website. Each time it's a new session and new context to be captured.

The UI layer JavaScript assigns a context ID derived from unique session ID returned by Java session object. Context ID would then be appended with each user Click Stream data which is the URL generated from each user's click on an item. An example of click data is as follows: http://xxx.com:8080/electronics/products?userID=id3&productName=iPhone8&price=500&location=london.

Recommender systems gets input from different sources to make recommendations. Most common way of collecting input is through user's feedback. Shopping portals for example collects rating provided by the users. But explicit feedback may not be available always. Thus number of techniques were adopted to build recommendation through implicit feedback [7].

Click stream contains user's location data which is derived from client IP address. Country, region, state and city is the part of location data. Each user click on an item generates an event and clickstream which is captured by real time Big Data ingestion agent Flume and data is moved from source system to the analytics processing system on a near real-time basis.

Spark Streaming framework process the data and stores into a MongoDB datastore.

This context aware real-time data is used to show 'recently viewed' or 'recently purchased' items to enhance overall shopping experience. Data that is stored in MongoDB is processed using

A batch job which runs periodically at an interval of 6 hours. Result of the batch job is an item recommendation table which is again stored back in MongoDB. Recommendation table is accessed from UI layer through RESTful API written in Java Spring framework. The end-to-end Big Data ingestion framework is shown in Fig 1.

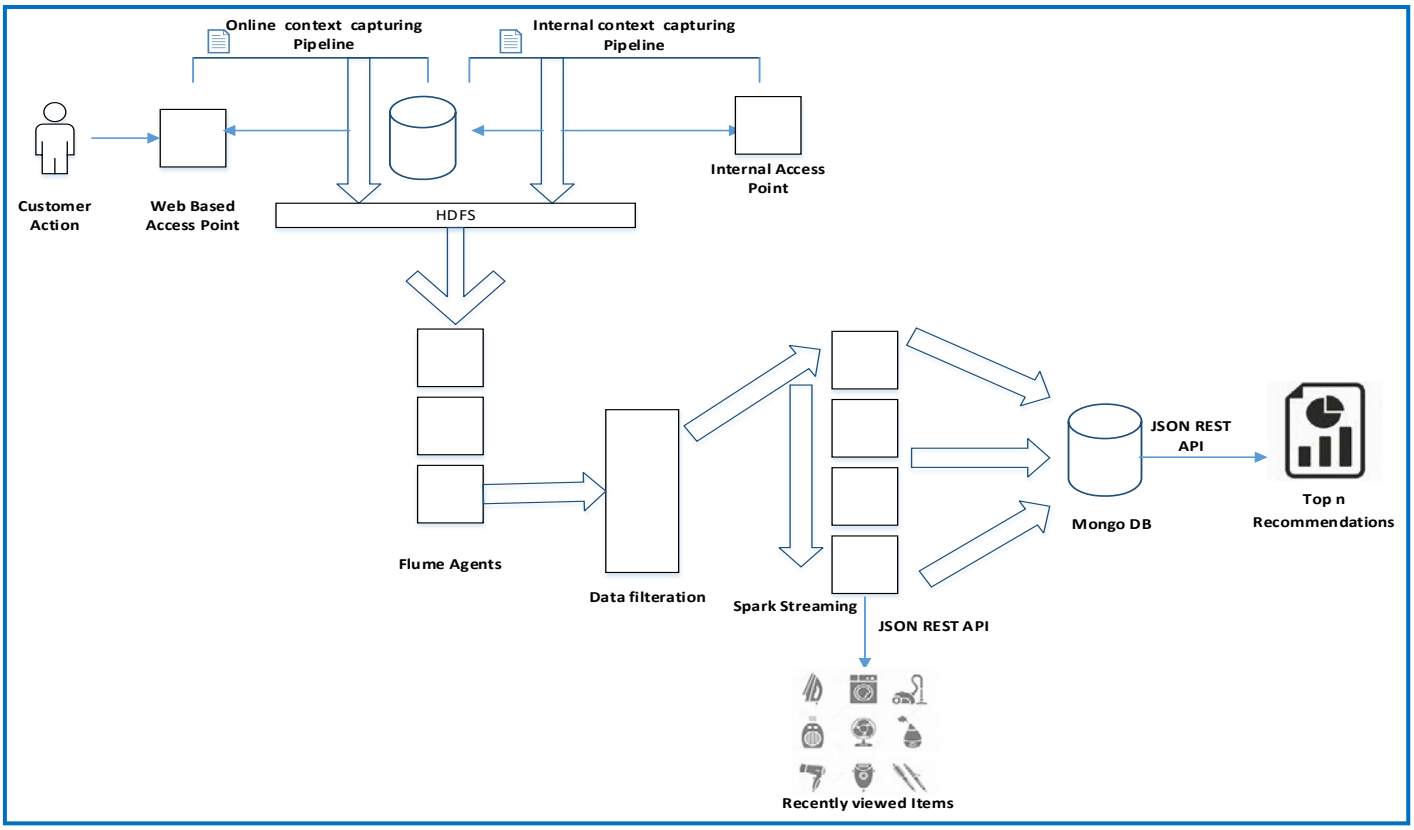

Figure 1

3.2 Big Data Solution that enables rapid aggregation to create co-occurrence matrix

After the events are captured and stored into MongoDB, at a periodic interval, a co-occurrence job (Spark SQL/Pig) is initiated by a scheduling component(Ozzie) to read events out of MongoDB store to generate co-occurrence matrix in the following sequence:

a) Start with an empty co-occurrence table. See Fig 2.

item 1	item 2	count

Figure 2

b) Session 1, user views/purchase items A, B, C

A-B (and B-A)

A-C (and C-A)

B-C (and C-B)

We initialize the co-occurrence count to 1. See Fig 3

item 1	item 2	count
A	B	1
A	C	1
B	C	1
B	A	1
C	A	1
C	B	1

Figure 3

c) Session 2, user views item B, C, D. Create or increment following association by count 1:

B-C (and C-B)
B-D (and D-B)
C-D (and D-C)
Count field is updated with each iteration. See Fig 4.

Method used to build this matrix is described in Algorithm 1.

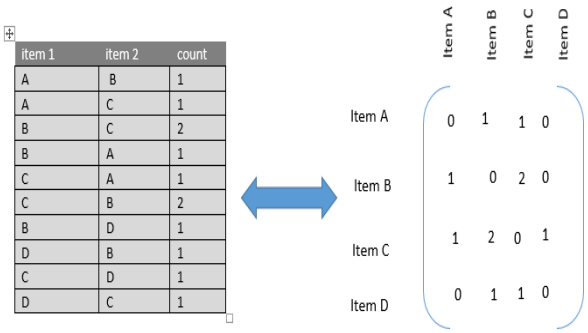

Figure 4

3.3 Item Similarity Computation

Once the co-occurrence matrix is built, the list is sorted in descending order based on the co-occurrence count. To filter out unrelated item pairs we compute cosine similarity between all item pairs by comparing common features.
For example, a television (product A) may fall in the following category: Appliances (category id 1) – LCD
(category id 4) – 3D LCD (Category id 6). And a camera (product B) is in the following category: Electronics (category id 1) – camera (category id 35) – DSLR camera (category id 95).
We calculate the item similarity strength of product A and B at a scale of 0 to 1 by the following formula:

$$SA,B=\cos(\theta)=\frac{(A,B)}{(|A||B|)}=\frac{\sum_{i=1}^{n} W_iA \cdot W_iB}{\sum_{i=1}^{n}\sqrt{W_iA^2}\sqrt{W_iB^2}} \quad (1)$$

Using the equation (1) we can calculate similarity strength between a television and a camera:
A= (1,4,6)
B = (1,35,95)
Cosine Similarity $SA,B = \frac{1\times1+4\times25+6\times27}{\sqrt{1^2+4^2+6^2}\times\sqrt{1^2+35^2+95^2}} = 0.964.$

3.4 Computing Recommended items

A weighted hybridization strategy combines the recommendations of two or more factors by computing weighted sums of their scores. This step is performed once the items are filtered out based on similarity count. The overall recommendation score R can be calculated for a user by the following equation:

$R=\prod_{k=1}^{n} \bar{\omega}_K$ (2)

$\bar{\omega}_K$ is the normalized from for each latent factor weight ω_k

Normalization is done by the following equation: $\bar{\omega}_K = \frac{\omega_k}{max_k(\omega_k)}$

where ω_k is the weight for each recommendation factor, $max_k(\omega_k)$ is the weight of the same factor which is maximum among all of the also viewed items.

Different potential latent factors are: (i) co-occurrence count, (ii) user's location and (iii) timestamp of the data in the recently viewed table. Co-occurrence count is obtained from step A. Location and timestamp of the data are discussed below:

3.5 User's Location

For an active user for whom recommendation is to be computed, we first retrieve the user's physical location from click data by parsing clickstream URL. Then to calculate the recommendation score, more weight is applied for the same location where item is viewed, and item is also viewed by users under the same shopping session.

For example, if active user is from location l_1 and recommendation has to be shown for item i_1 then we calculate also viewed items for i_1 as i_2, i_3, i_4 and i_5. If we consider i_1 and i_2 are viewed in the same location, we put more weight to score of i_2. If i_1 and i_2 are viewed together n times and both are from same location then the recommendation score for $i_2 = n \times l_1$ where l_1 is the weight applied for similar location. If i_1 and i_3 are from two diffrent location and the occurance count between them is m then we calculate recommendation score for $i_3 = m \times l_2$.

3.6 Time variant data

One of the primary motivations behind item-based recommendation algorithms depends on the fact that the two items viewed by the same user are likely be related to each other. But in e-commerce scenario age of the data can play an important factor for relevancy of related items [17]. Age of the data and can be an influencing factor ranking the items for top n recommendations. Current algorithm prioritized recent data over the old historical data. To achieve this, we first fetch all recently viewed item from MongoDB and order them reverse chronologically so that most recent item is on the top. We then use Big Data solution that enables rapid aggregation to create co-occurrence matrix for each recently viewed item that is discussed in the previous section. Since the recently viewed items are reverse chronologically ordered, when weight is applied in descending order-most recent item gets highest weight. Real time and historical data is represented as autonomous Multi Agent System in collaboration. The entire algorithms is expressed in Algorithm 1:

Algorithm 1: Top n recommendation generation algorithm

Input: Customer click stream data, time_decay_factor
Output: top n recommended product list
Begin
item_timestamp_weight=100
For each customer C who viewed Ii do
Record all the items recently viewed I_i (i=1 to n) in reverse chronological order
For each recently viewed item I_i do
Record customer C also viewed related item RI_i
For each item pair I_i, RI_i
Filter pair I_i, RI_i based on cosine similarity Compute the co-occurrence count between pair I_i, RI_i
Decide user's location
Apply Equation (2) to decide overall score
End for
Item_timestamp_weight=item_time_stamp_weight - time_decay_factor
Decide and the final recommendation order of I_i
End for
End

4. ILLUSTRATIVE EXAMPLE

Suppose a user $u_1$is in a location loc-1, opens e-commerce portal and views 3 items: A, B, C at time t_1, t_2, t_3. where $t_1 < t_2 < t_3$. We need to build up recommendations for user u_1.

First, consider historical data that's already loaded into DB capturing users' click data. Create an also-viewed table as shown in Table 1. It captures items also viewed across all user base. A periodic batch job aggregates this data by adding the also-viewed count and similarity strength between two products through Cosine Similarity on item category and subcategory. See Table 1.

TABLE I
Computing TOP n RECOMMENDATION: STEP -1

Timestamp (t)	Item viewed	Item also viewed	Similarity strength	Also viewed count
t1	A	D	0.98	10
t1	A	E	0.95	9
t1	B	F	0.88	11
t1	B	G	0.97	12
t2	B	H	0.77	15
t3	C	I	0.95	8
t3	C	J	0.94	6
t3	C	K	0.60	13

Next, recommendation score is computed based on
Algorithm 1. A weighted hybridization strategy computes the final score putting different weight scheme on location and time. If item viewed and item also viewed are from same location more weight is applied to also viewed item. Similarly, weight is applied for time stamp field in a reverse chronological way. So item with latest timestamp will get highest weight. Item K is not considered for further processing as the Cosine Similarity count is low. See Table 2.

TABLE 2
Computing Top n Recommendation: STEP 2

Item also viewed	Also viewed Count (normalized)	Item viewed location	Also viewed location	Recommendation Score (t×l×c)
D	1	Loc1	Loc1	1×1×1=1
E	0.9	Loc1	Loc2	1×0.8×0.9=0.72
F	0.73	Loc3	Loc3	0.9×1×0.73=0.65
G	0.8	Loc3	Loc4	0.9×0.8×0.8=0.57
H	1	Loc3	Loc5	0.9×0.8×1=0.72
I	0.61	Loc6	Loc6	0.8×1×0.61=0.48
J	0.46	Loc6	Loc7	0.8×0.8×0.46=0.29

Base on the recommendation score, final order for the top n recommendations are:
D>E>H>F>G>I>J

5. EXPREMENTS

In this section, we examine TNRS on real world 20 million dataset from movielens.org [18] and compare its performance with Spark MLLib ALS API [19]. Movielns 20M data contains upto 1.38 million users and 20 million ratings. Since the TNRS doesn't predict the rating but rather ranks the recommended products on overall score, we validate the ranks against the rating predicted by Apache Spark ALS API. Deployment model is depicted in fig 3. We used Google Cloud Compute Engine medium level server configuration as follows:
1 vCPU
6.5 GB RAM
Centos 7 OS

Recommender application was developed as Java Spring REST Webservice which is deployed in Tomcat application server. Interface between MongoDB datastore and Recommender app is done through Jongo driver which is a popular MongoDB Java client. The deployment model is shown in fig 5.

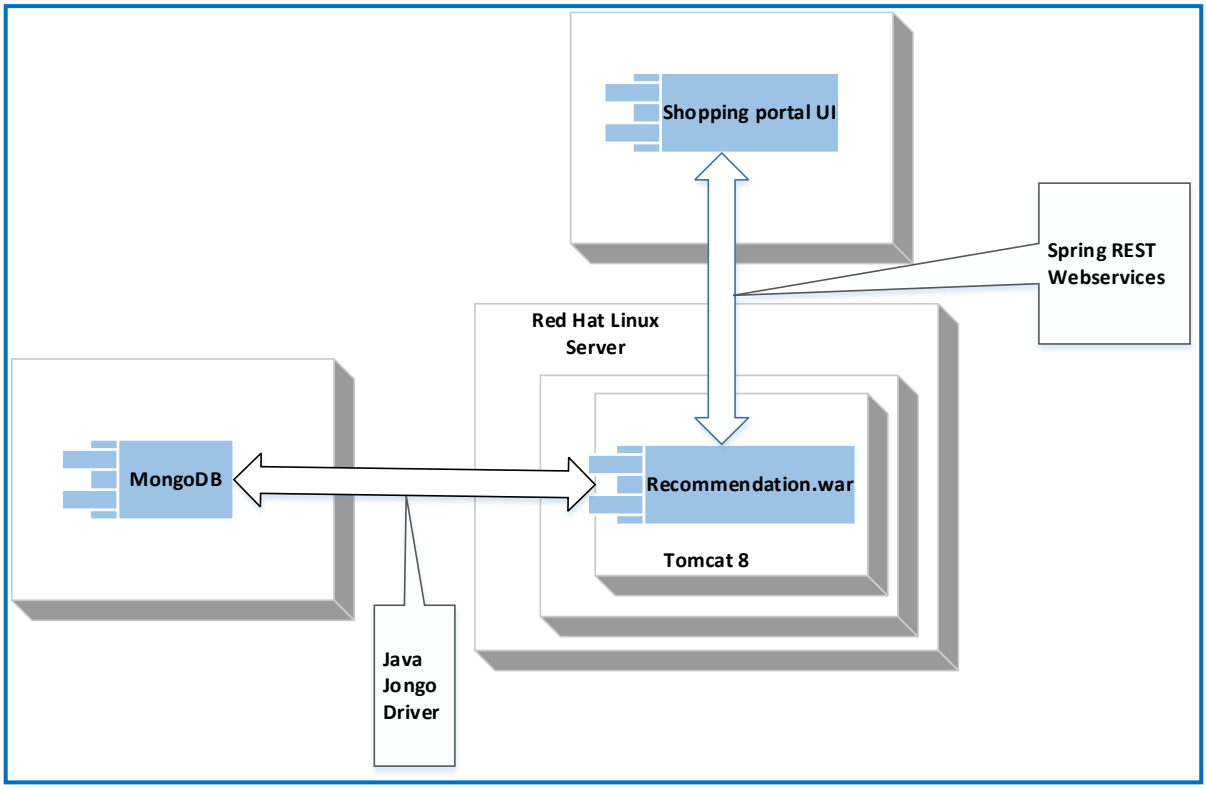

Figure 5

We start with analyzing the movielens data. Movielens DB has three tables with following schema:

TABLE 3
Movielens Database

Table	Column Names
rating	UserID::MovieID::Rating::Timestamp
users	UserID::Gender::Age::Occupation::Zip-code
movies	MovieID::Title::Genres

We use these tables to create a derived table users-movie table which contains the following column: UserID:, MovieID and Zip-code.
Hadoop Pig scripts are used to transform this data which will contain movies also viewed across all user base. However, in current model we can't determine from movielens database if also viewed products by a single user occurred in a single session or multiple sessions. As the algorithm is based on the assumption that items are considered also viewed if viewed under same browsing session. This is a given limitation which can be overcome by availability of live click-stream data from e-commerce portals.
In step two, we determine user's location (particular state user is located) from zip code.
In step three, produce a 'also-viewed' product count on the following two scenarios:
Count also-viewed products where product viewed, and product also viewed are same location (state).

Count also-viewed products where product viewed, and product also viewed are in different location.
In step three, apply the algorithm illustrated in Algorithm 1 to compute top n recommended products.
Step four, now validate the result. For the entire movielens 20 M database, take 80% of the data for prediction and match the result with remaining 20% of the data.
We used Spark MLLib ALS API [19] to predict the future rating. Written in Scala Recommendation parameters are set as follows:s

```
val als = new ALS()
   .setMaxIter(5)
   .setRegParam(0.01)
   .setUserCol("UserID")
   .setItemCol("MovieID")
   .setRatingCol("Rating")
val alsModel = als.fit(training)
val predictions = alsModel.transform(test)
```

We validate the predicted ration from Spark ALS with ranking we make. More specifically, if we predict a rating with Spark ALS and if the prediction matches with actual rating in 20% test database, we compare the rating with ranking made by TNRS. If an item with rating five is ranked among top three then the raking algorithm is considered successful.
Findings
TNRS is more accurate than Spark ALS API when we consider the context parameters (e.g. location) and apply time decay function. The algorithm's approach to put more weight on current data over historical data makes the recommendation more relevant and hence brings more chances of higher Click Through Rate (CTR).

TABLE 4
Experimental Results

Item Number	Actual Rating	Spark ALS Prediction	Top n recommendation algorithm order
2987	4	4.1	2nd
1250	5	4.3	1st
3791	4	3.9	4th
858	3	2.6	5th
1304	3	3.2	5th
3791	2	2.5	22nd
2746	4	4.3	2nd
260	5	4.6	1st
150	4	3.7	4th
2987	1	2.5	29th
3448	3	3.8	4th

6. DISCUSSION

TNRS model doesn't suffer user/item cold start, sparsity or scalability problems. However, if absolutely no historical data is available for a product (may be a newly launched product), user can't see any recommendation since there are no similar items viewed together across user base. In this this scenario, TNRS can be extended to a hybridization strategy to recommend similar items based on the item feature. Two strategies can be adopted – first one is just to recommend same category of products. That is, if user views a smartphone, recommendation would be few other smartphones ordered by popularity. Second approach to this problem can be to create an user feature vector x_i and item feature vector x_j. Then a common strategy is to find the similarity between these vectors. There are several ways to measure similarity between two vectors. We can start with a simple setting where x_i and x_j are points in the same vector space; that is, both users and items are represented using the same set of features. We can then find the cosine similarity between two vectors.

For validation of recommendation accuracy, we have adopted an offline strategy where 80% data was used for recommendation and 20% for validation. However arguably a more effective to verify recommendation accuracy is through actual recommendation click through rate(CTR) and purchase rate(PR). Which determines user's preferences on viewing or buying from recommended items vs non-recommended items. As this would require a production deployment with large set of real user base this is not a feasible option for us at this moment.

7. CONCLUSION

In this paper we discussed a novel approach for top n recommendation system. Novelty of this approach is in the use of contextual parameter with collaboration of multi agent system for historical batch data and near real time data to predict the recommendation accurately. Other important distinction in the algorithm is that, it doesn't rely on explicit rating provided by the user, rather it computes on user's passive endorsement by click data and doesn't suffer from sparsity problem of user similarity approach. Finally, algorithm doesn't enforce user to login to provide recommendations and provides accurate recommendations for non-logged in users.

8. REFERENCES

[1] B. Smith and G. Linden, "Two Decades of Recommender Systems at Amazon.com," IEEE Internet Computing, vol. 21, no. 3, pp. 12-18, 2017.

[2] L. Yao, Q. Z. Sheng, A. H. H. Ngu, J. Yu, and A. Segev, "Unified Collaborative and Content-Based Web Service Recommendation," IEEE Transactions on Services Computing, vol. 8, no. 3, pp. 453-466, 2015.

[3] Y. Yang, Y. Xu, E. Wang, J. Han, and Z. Yu, "Improving Existing Collaborative Filtering Recommendations via Serendipity-based Algorithm," IEEE Transactions on Multimedia, vol. PP, no. 99, pp. 1-1, 2017.

[4] M. Kobayashi, K. Mikawa, M. Goto, and S. Hirasawa, "Collaborative filtering analysis of consumption behavior based on the latent class model," in 2017 IEEE International Conference on Systems, Man, and Cybernetics (SMC), 2017, pp. 1926-1931.

[5] B. Guo, S. Xu, D. Liu, L. Niu, F. Tan, and Y. Zhang, "Collaborative filtering recommendation model with user similarity filling," in 2017 IEEE 3rd Information

Technology and Mechatronics Engineering Conference (ITOEC), 2017, pp. 1151-1154.
[6] M. Alain and A. Smolic, "Light field denoising by sparse 5D transform domain collaborative filtering," in 2017 IEEE 19th International Workshop on Multimedia Signal Processing (MMSP), 2017, pp. 1-6.
[7] Y. Hu, Y. Koren, and C. Volinsky, "Collaborative Filtering for Implicit Feedback Datasets," in 2008 Eighth IEEE International Conference on Data Mining, 2008, pp. 263-272.
[8] Collaborative Filtering - RDD-based API. Available: https://spark.apache.org/docs/2.2.0/mllib-collaborative-filtering.html
[9] J. Wang, X. Peng, Z. Xing, K. Fu, and W. Zhao, "Contextual Recommendation of Relevant Program Elements in an Interactive Feature Location Process," in 2017 IEEE 17th International Working Conference on Source Code Analysis and Manipulation (SCAM), 2017, pp. 61-70.
[10] Y. Ren, M. Tomko, F. D. Salim, J. Chan, C. Clarke, and M. Sanderson, "A Location-Query-Browse Graph for Contextual Recommendation," IEEE Transactions on Knowledge and Data Engineering, vol. PP, no. 99, pp. 1-1, 2017.
[11] M. M. Rahman, "Contextual recommendation system," in 2013 International Conference on Informatics, Electronics and Vision (ICIEV), 2013, pp. 1-6.
[12] F. B. Kharrat, A. Elkhleifi, and R. Faiz, "Recommendation system based contextual analysis of Facebook comment," in 2016 IEEE/ACS 13th International Conference of Computer Systems and Applications (AICCSA), 2016, pp. 1-6.
[13] M. A. Domingues, C. V. Sundermann, M. G. Manzato, R. M. Marcacini, and S. O. Rezende, "Exploiting Text Mining Techniques for Contextual Recommendations," in 2014 IEEE/WIC/ACM International Joint Conferences on Web Intelligence (WI) and Intelligent Agent Technologies (IAT), 2014, vol. 2, pp. 210-217.
[14] F. Xie, M. Xu, and Z. Chen, "RBRA: A Simple and Efficient Rating-Based Recommender Algorithm to Cope with Sparsity in Recommender Systems," in 2012 26th International Conference on Advanced Information Networking and Applications Workshops, 2012, pp. 306-311.
[15] Z. Sharifi, M. Rezghi, and M. Nasiri, "A new algorithm for solving data sparsity problem based-on Non negative matrix factorization in recommender systems," in 2014 4th International Conference on Computer and Knowledge Engineering (ICCKE), 2014, pp. 56-61.
[16] R. Reshma, G. Ambikesh, and P. S. Thilagam, "Alleviating data sparsity and cold start in recommender systems using social behaviour," in 2016 International Conference on Recent Trends in Information Technology (ICRTIT), 2016, pp. 1-8.
[17] C. Xia, X. Jiang, L. Sen, L. Zhaobo, and Y. Zhang, "Dynamic item-based recommendation algorithm with time decay," in 2010 Sixth International Conference on Natural Computation, 2010, vol. 1, pp. 242-247.
[18] MovieLens 20M Dataset. Available: https://grouplens.org/datasets/movielens/20m/
[19] M. Winlaw, M. B. Hynes, A. Caterini, and H. D. Sterck, "Algorithmic Acceleration of Parallel ALS for Collaborative Filtering: Speeding up Distributed Big Data Recommendation in Spark," in 2015 IEEE 21st International Conference on Parallel and Distributed Systems (ICPADS), 2015, pp. 682-691.