International Journal of
Intelligent Systems and Applications in Engineering

ISSN:2147-6799 www.ijisae.org Original Research Paper

Reservoir Sampling Based Streaming Method for Large Scale Collaborative Filtering

Tevfik Aytekin*[1]

Accepted : 04/08/2018 Published: 29/09/2018

Abstract: Collaborative filtering algorithms work on user feedback data (such as purchases, clicks, or ratings) in order to build models of users and items. User feedback data in real life e-commerce sites can be very large which incurs high costs on maintenance and model building. Parallelization of computation might help but it results in additional costs for extra computing power and maintenance problems of very large datasets still persist. Sampling at this point can be an effective approach for reducing the amount of data. In this work we propose a novel sampling technique for collaborative filtering which can be used to reduce the amount of data considerably. Experimental results on three real life datasets show that the proposed method leads to a significant reduction in the amount of data with little harm to the accuracy of the models. The method works in a streaming fashion, which makes it suitable for being used in real time at large-scale e-commerce applications where there is a large flow of continuous user feedback.

Keywords: *Collaborative filtering, reservoir sampling, large-scale recommender systems.*

1. Introduction

Development of online markets has dramatically increased the number of products available for customers. This makes it difficult for people to search and find the products they are interested in. Recommender systems help people to find items they are looking for by analyzing their past interactions (such as purchases, clicks, and ratings). There are basically two main approaches for building recommender systems: content-based and collaborative filtering. In content-based systems the content of the items that users like in the past are analyzed and items with similar content are recommended to the users [1, 2]. In order to build a content-based recommender system item content information (which is generally in unstructured form) should be structured in order to be given as input to machine learning algorithms. For example if it is a movie recommender system, plot summaries need to be represented in vector space model and other content information about movies (such as genre, stars, director, etc.) should be represented in a suitable form. In collaborative filtering, instead of content, the interactions of users with items are used. There are two basic approaches in collaborative filtering: neighborhood-based (user or item based) and matrix factorization. In user-based collaborative filtering [3, 4] a user is recommended items, which are liked by similar users where similar users are defined as those users who have similar purchase histories. Whereas in item-based collaborative filtering [4, 5] users are recommended items which are similar to the items in their purchase history. Again, similarity between items is calculated by analyzing the purchase history of items. Since user/item based recommender algorithms need to calculate the similarities between all pairs of users/items the time for model building takes quadratic time with respect to users/items. This creates scalability problems for neighborhood-based algorithms when the number of users/items is large.

[1] *Computer Eng., Bahçeşehir University, İstanbul – 34353 TURKEY*
* *Corresponding Author Email: tevfik.aytekin@eng.bau.edu.tr*

In matrix factorization approaches [6, 7, 8, 9] which is another collaborative filtering method, latent representations of users and items are learned by making a low-rank approximation of user/item matrix. In order to build a matrix factorization model, first a cost function is designed and then this cost function is optimized, generally, with gradient descent. Matrix factorization approaches are considered to be the state-of-the-art recommender algorithms since their accuracy is generally superior to neighborhood-based approaches. There are also hybrids methods which try to combine the strength of various recommendation approaches [10, 11]. In the early stages of recommender systems research building an accurate model is the main objective. However, later it is recognized that there are other dimensions of recommender systems which need to be considered beyond accuracy. Context-aware recommendation [12], diversity [13], privacy [14], and robustness [15] can be given as examples of these other dimensions recent research has focused on. The work in recommender systems has produced very successful methods and today recommender systems are used by almost all e-commerce sites and large companies.

The data used to model user preferences in recommender systems can be very large. For example, in web scale an e-commerce system's users might leave millions of clicks or purchase information every minute. It becomes increasingly difficult to store and process this huge amount of data. Even transferring such data from one place to another can take a very long time. To deal with this problem researchers have developed a variety of techniques which can be grouped into two main categories. In the first category there are parallel processing techniques. Basically, these techniques distribute the computation to a cluster of machines. To this end various parallel or distributed recommendation algorithms are proposed [16, 17, 18]. Also in several open source projects (such as Apache Spark [19] and Apache Mahout [20]) distributed implementations of several collaborative filtering algorithms (e.g., alternating least squares, item-based collaborative filtering) can be found. The other direction of research for dealing with large amounts of data is to develop approximate solutions. These methods compromise accuracy in order to improve processing time. Here we can see recommender algorithms based on various clustering approaches [21, 22, 23] where similar users or items are clustered together in order to reduce the search time for similar users or items. To deal with large-scale data both approaches (namely parallel processing and approximate methods) can also be applied at the same time in a hybrid fashion.

Even though these techniques work well to a certain extent, they try to minimize the time for building recommender models. However, with the increasing amounts of data, even storing or transferring large amounts of data become a big burden. This naturally leads to another idea: instead of using all the available data, can't we sample a representative portion of it and ignore the rest. For sampling to work, we need to sample data intelligently in order to keep as much information as we can. Also, this sampling should assume a streaming environment. Given the large-scale data it is infeasible to first store all the data and then apply sampling on it. What should be done instead is to sample data as it arrives in a streaming fashion.

In this paper we propose a sampling mechanism tailored for recommender systems. In particular, the sampling method is designed for item-based collaborative filtering (IBCF) which is a well-known and widely used algorithm in the industry [24]. The proposed sampling method works in a streaming fashion which is important for utilizing it in a real life setting where data comes as a stream. We apply the method on three real life datasets and show that even if a small portion of the data is sampled using the proposed method, the sampled data can still be used to build models with little accuracy loss.

The paper is structured as follows: In Section 2, we will describe the proposed method in detail, in Section 3, we will give the experimental results and evaluate the success of the proposed method, and in Section 4, we will conclude the paper.

2. Proposed Approach

Let U be the set of users and I be the set of items. We assume that the data arrives as $<u, i>$ pairs where $u \in U$ and $i \in I$. An $<u, i>$ pair means user u has clicked or purchased item i. This type of feedback is known as implicit feedback in the literature to distinguish it from explicit feedback where users provide ratings for items. Since, compared to explicit feedback, implicit feedback is much more common in real life we assume this type of feedback in this work.

One simple sampling strategy, called Bernoulli sampling [25], for a stream of elements is to sample every ith element. There are two problems with this sampling strategy. First, since the size of the stream is not known (or unlimited) the size of the resulting sample cannot be limited to a fixed size. This might be a problem when the resources (such as memory) are limited and a certain maximum sample size must be guaranteed. The second difficulty is that this sampling strategy will sample the same ratio of preferences from each user and item. This strategy leads to a loss of information from users/items that have a small amount of feedback, i.e., users/items with small number of purchases or clicks. A better strategy should sample more from users/items with lots of feedback and use most (if not all) of the information from users/items with little feedback.

One well-known sampling method which guarantees a fixed sample size is called reservoir sampling [26]. Reservoir sampling produces a random sample of k elements from a stream of size n where the size of the stream S is unknown and the probability of an element to be in the sample is k/n. Algorithm 1 describes a reservoir sampling method for a stream of $<u, i>$ pairs.

Algorithm 1. Simple Reservoir Sampling (SIMR)
Input: S: A stream of preferences, k: reservoir size
Output: R: reservoir of preferences

```
(1)  $R \leftarrow []$;
(2)  for $p \leftarrow 1$ to $n$
(3)      if $p \leq k$
(4)          $R[p] \leftarrow S[p]$;
(5)      else
(6)          $j \leftarrow random(1, p)$;
(7)          if ($j \leq k$)
(8)              $R[j] \leftarrow S[p]$;
```

Reservoir sampling described in Algorithm 1 works as follows: initially reservoir R is set to be empty. The first k preferences in the stream are directly put to R without sampling. For each pth preference where $p > k$, first a random number j between 1 and p is generated, if j is less than or equal to k then the pth preference is replaced with the jth preference in the reservoir else the pth preference is discarded. This method can be applied to a stream of incoming preferences to select randomly k preferences where the probability of selecting each preference is k/n. The proof of this result is as follows: Consider pth preference, where $p > k$, the probability of its being placed in R is k/p. Now consider the next, $(p+1)$th preference. The probability of the pth preference to be kept in R is $(k/p)\times p/(p+1)$, the second term is the probability of not replacing the pth preference with the $(p+1)$th preference. In general then the probability of the pth ($p > k$) preference to be selected is:

$$\frac{k}{p}\times\frac{p}{p+1}\times\cdots\times\frac{n-2}{n-1}\times\frac{n-1}{n}=\frac{k}{n}$$

On the other hand if $p \leq k$ it will be directly placed in R. The probability of it being in R after the $(k+1)$th preference is $k/(k+1)$. Again in general the probability of the pth, where $p \leq k$, preference to be selected is:

$$\frac{k}{k+1}\times\frac{k+1}{k+2}\times\cdots\times\frac{n-2}{n-1}\times\frac{n-1}{n}=\frac{k}{n}$$

After selecting k preferences, a model can be build using the preferred collaborative filtering algorithm. We will use SIMR as a baseline for comparing our proposed sampling technique.

Even though simple reservoir sampling provides a fixed sample size, the second difficulty of Bernoulli sampling still persists. To alleviate this difficulty we propose a stratified reservoir sampling where for each user a separate reservoir is maintained. This method is described in Algorithm 2.

Algorithm 2. Stratified Reservoir Sampling (STR)
Input: S: A stream of preferences, k: reservoir size
Output: R_U: Reservoir of preferences for each user

```
(1)  foreach user $u$ in $U$
(2)      $R_u \leftarrow []$;
(3)  for $p \leftarrow 1$ to $n$
(4)      $u \leftarrow S[p].u$;
(5)      $R_u[j] \leftarrow S[p]$;
(6)  $R_U \leftarrow \bigcup_{u \in U} R_u$
```

where $S[p].u$ refers to the user u of the p^{th} preference in the stream. Initially all the reservoirs are initialized to be empty. Then for each incoming preference $p = <u, i>$, the preference is placed in the reservoir associated with user u using the simple reservoir algorithm described in Algorithm 1. In the final step all the reservoirs are merged.

In this method for every user a separate reservoir of size k is maintained. In real life environments while some users have little feedback some users provide a lot of feedback. By maintaining a separate reservoir for each user we aim to keep the preferences of users who have little feedback and prevent loosing this scarce and valuable information. In the next section we will experimentally show that this idea leads to improved results.

3. Experimental Results

In this section we will give the performance results of SIMR and STR on three real life datasets. The basic statistics of the datasets are given in Table 1. Because of memory limitations we reduced the original Amazon-books dataset by removing items and users which have less than 50 preferences.

Table 1. Basic statistics of the datasets.

Name	# of users	# of items	# of preferences	Sparsity
Movielens[1]	6040	3900	1000209	0.042
Amazon-books[2]	27529	8982	811962	0.0033
Jester[3]	59132	140	1761438	0.21

The evaluation methodology is as follows: we randomly put 10% of the preferences in the test set. For decreasing reservoir sizes (in STR we use the same reservoir size for each user) we sample preferences from the training set using one of the two sampling strategies and build a model with IBCF using the sampled preferences. Then we make a top-*N* recommendation to each user in the test set and calculate Mean Average Precision (MAP) and Normalized Discounted Cumulative Gain (NDCG). 10-fold cross validation is used for getting the final results. MAP is a widely used metric for evaluating recommender systems. MAP at *N* is defined as in (1):

$$MAP@N = \frac{1}{|U_t|}\sum_{u\in U_t}\frac{1}{|R_u|}\sum_{i=1}^{N}P(i)\times R(i) \qquad (1)$$

where $P(i)$ is the precision at the *i*th position of the top-*N* list for user *u*, U_t is the set of users in the test set, $R(i)$ is a binary indicator which returns 1 if the *i*th item is relevant or 0 otherwise, and $|R_u|$ is the number of items of user *u* in the test set.
NDCG is also a widely used performance metric used in recommender systems. NDCG at *N* is defined in (2):

$$NDCG@N = \frac{1}{|U_t|}\sum_{u\in U_t}\frac{1}{IDCG}\sum_{i=1}^{N}\frac{R(i)}{\log(i+1)} \qquad (2)$$

where U_t is the set of users in the test set, $R(i)$ is a binary indicator which returns 1 if the *i*th item is relevant or 0 otherwise, and IDCG (ideal discounted cumulative gain) is the maximum possible value DCG can get.
Fig. 1, Fig. 2, and Fig. 3 show the performance of the SIMR and STR methods on three datasets. These figures show MAP vs. the size of the sampled dataset used in the experiments. In all the figures STR gives better results than SIMR. These results show that the proposed method, STR, is effective in reducing the amount of data needed to make good recommendations. Now let us look at these figures in somewhat more detail. As seen in Fig. 1, it is possible to reduce the Movielens dataset with STR down to 10% of the original size with very little loss of accuracy. Fig. 3 shows that for the Amazon dataset down to 25% reduction is achieved with a small accuracy loss. Experimental results on Jester dataset, shown in Fig. 3, show that it is possible to reduce the dataset down to 6% of the original size without no decrease in accuracy. If we look at the sparsity values of the datasets given in Table 1 we can see that there is a correlation between the sparsity values and the amount of reduction (without hurting accuracy much) one can get. As the datasets get more denser there is a high potential for reducing the datasets. This is expected since a dense dataset means that users have lots of feedback and when there are lots of feedback we can ignore some of them and still get good results.

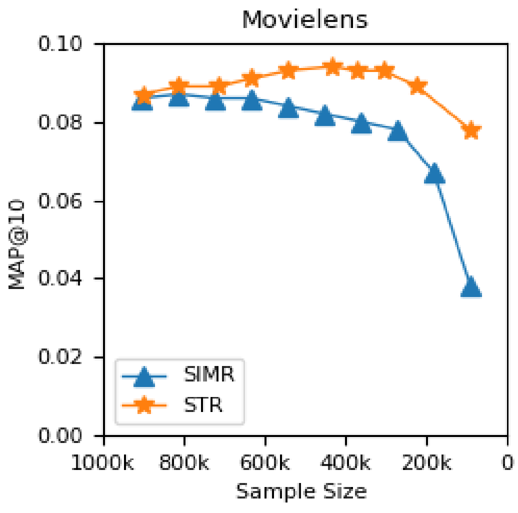

Fig. 1. MAP vs. reservoir size results for Movielens dataset.

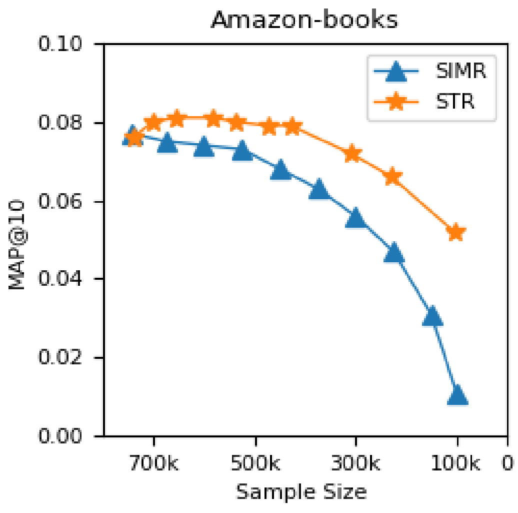

Fig. 2. MAP vs. reservoir size results for Amazon-books dataset.

[1]https://grouplens.org/datasets/movielens/
[2]http://jmcauley.ucsd.edu/data/amazon/
[3]http://goldberg.berkeley.edu/jester-data/

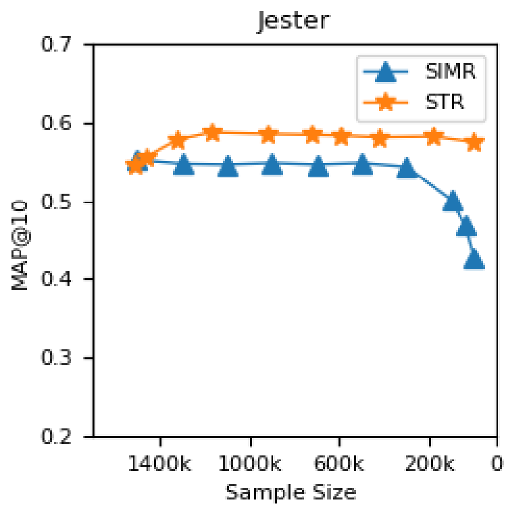

Fig. 3. MAP vs. reservoir size results for Jester dataset.

Fig. 4, Fig. 5, and Fig 6 show the performance results using the NDCG metric. NDGC results are in harmony with the MAP results which confirm that STR method leads to important improvements compared to SIMR.

Another interesting result which can be observed in the given figures is that for all datasets STR method leads to an initial improvement in the accuracy. The reason for this behavior might be the fact that STR method begins sampling initially from the users who have the highest number of preferences. Since initial reservoir sizes are large, all preferences of users who does not have many preferences are selected. Assuming that users who have many preferences tend to be more careless then users who have a low number of preferences, the initial steps in sampling increases the weight of the preferences of the letter type of users in the similarity calculations. Detailed experiments are required to fully explain this positive effect of STR. Since this is not the main contribution of this study we leave it as a future work.

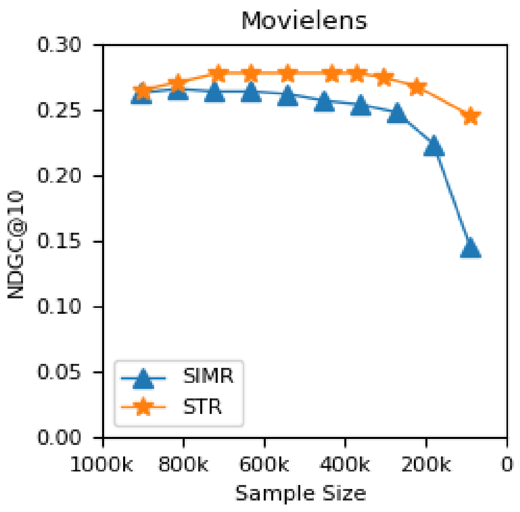

Fig. 4. NDGC vs. reservoir size results for Movielens dataset.

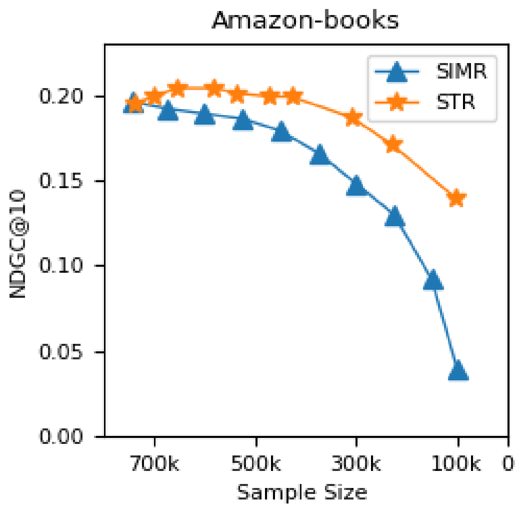

Fig. 5. MAP vs. reservoir size results for Amazon-books dataset.

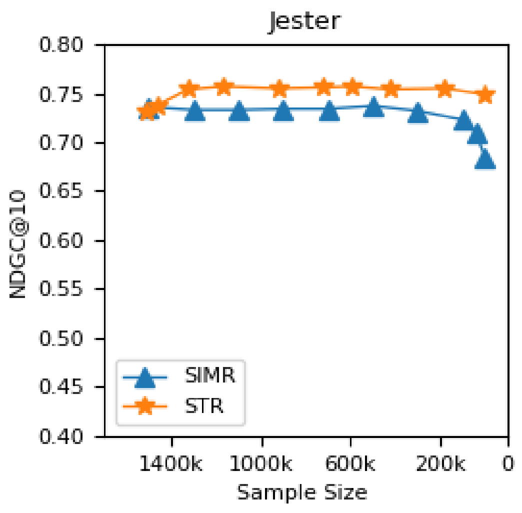

Fig. 6. MAP vs. reservoir size results for Jester dataset.

Recently aggregate diversity has become another dimension to evaluate the performance of a recommender system [26, 27]. Aggregate diversity refers to the number of unique items recommended across all recommendation lists which is defined in (2):

$$Aggregate\text{–}diversity = \left| \bigcup_{u \in U_t} L(u) \right| \tag{2}$$

where $L(u)$ is the top-N list of user u. Aggregate diversity is important because it is known that recommender algorithms tends to recommend popular items and do not recommend some items to anybody which is an undesired property from the business perspective. One drawback of the metric given in (2) is that it does not measure the distribution of recommended items. So, other metrics for measuring the distribution of recommended items are also used [28]. One such metric is given in (3):

$$Gini\text{–}diversity = 2\sum_{i=1}^{n}\left[\left(\frac{n+1-i}{n+1}\right)\times\left(\frac{rec(i)}{total}\right)\right] \tag{3}$$

where $rec(i)$ refers to the number of users that are recommended item $i \in I$ and $total$ refers to the total number of recommendations that are made to all users. For calculating Gini-diversity items are sorted in ascending order of *rec*(*i*). The value of this metric gets larger, in contrast to the traditional Gini metric, as the distribution becomes more uniform. We use this form of Gini since for all the metrics used in this work a larger value means a better result.

Even though in this work our focus is not on aggregate diversity, it is important to see the effect of our method on aggregate diversity. Table 2 and Table 3 show the diversity values for SIMR and STR on the three datasets.

Table 2. Aggregate diversity.

Movielens		Amazon Books		Jester	
SIMR	STR	SIMR	STR	SIMR	STR
1205	1197	7335	7314	128	128
1196	1170	7291	7369	127	127
1168	1175	7298	7310	127	126
1146	1172	7332	7347	128	127
1105	1140	7361	7356	127	126
1051	1116	7463	7375	126	126
1048	1090	7487	7323	128	127
955	1032	7659	7314	127	128
967	1003	7682	7344	128	127
966	1001	7616	7356	127	127

Table 3. Gini diversity.

Movielens		Amazon Books		Jester	
SIMR	STR	SIMR	STR	SIMR	STR
0,078	0,075	0,287	0,282	0,273	0,274
0,077	0,074	0,282	0,282	0,274	0,273
0,076	0,072	0,280	0,280	0,273	0,273
0,075	0,070	0,274	0,279	0,273	0,272
0,072	0,068	0,273	0,280	0,274	0,272
0,068	0,066	0,274	0,283	0,273	0,270
0,063	0,063	0,272	0,282	0,275	0,270
0,057	0,060	0,280	0,279	0,289	0,270
0,052	0,058	0,287	0,286	0,289	0,269
0.050	0.056	0.289	0.287	0.289	0.270

Rows in the tables correspond to decreasing reservoir sizes used in the experiments for generating the previous figures. As can be seen from the tables, for Amazon and Jester datasets there is no decrease in the diversity values. That is, after sampling the recommendation algorithm can still produce the same number and distribution of items in the recommendation lists. For the Movielens dataset there is a minor decrease in the diversity values. However, this is a result of the user item ratio of the Movielens dataset. As can be seen from Table 1, the number of users in Movielens dataset is close to the number of items. This makes it difficult to increase the number of uniquely recommended items. However, in real life settings the number of users is generally much more than the number of items, so even this minor decrease in diversity for the Movilens dataset will not likely occur in a real setting.

We also want to add that the proposed method can induce important savings not only for batch IBCF but also for incremental IBCF [29]. In incremental IBCF approaches as a new preference <*u*, *i*> arrives the similarities between item *i* and all other items need to be recomputed which is a very costly operation especially when the number of items is large. Sampling also helps to reduce this cost since re-computation of similarities will be made only for the sampled preferences instead of the entire set of preferences.

4. Conclusion

At web scale the amount of user feedback is huge and it becomes increasingly difficult to maintain and process this data. In this work we propose a novel sampling method in order to reduce the amount of data a recommender system needs. Compared to a simple sampling approach, we show that an important amount of reduction is possible without giving a significant harm to the accuracy of the recommender algorithm. This shows that it is possible to increase the efficiency of sampling by developing more intelligent and sophisticated sampling techniques. We are planning to investigate more effective sampling strategies tailored for IBCF. We also plan to work on developing new sampling strategies for matrix factorization methods.

Our method works in a streaming fashion and this makes it suitable for being used in real-time for real applications which receive large amounts of user data. We believe that sampling-based methods will be an important and necessary part of dealing with big data and there is a need for more research in developing new sampling-based strategies.

References

[1] M. J. Pazzani and D. Billsus, "Content-based recommendation systems," *The adaptive web*. Springer, Berlin, Heidelberg, pp. 325-341, 2007.

[2] M. de Gemmis, M. Pasquale Lops, C. Musto, F. Narducci, and G. Semeraro "Semantics-aware content-based recommender systems," *Recommender Systems Handbook*. Springer, Boston, MA, pp. 119-159, 2015

[3] Z. D. Zhao and M. S. Shang. "User-based collaborative-filtering recommendation algorithms on hadoop." In *Knowledge Discovery and Data Mining, 2010. WKDD'10. Third International Conference on*, pp. 478-481. IEEE, 2010.

[4] C. Desrosiers and G. Karypis, "A comprehensive survey of neighborhood-based recommendation methods," *Recommender systems handbook*. Springer, Boston, MA, pp. 107-144, 2011.

[5] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl. "Item-based collaborative filtering recommendation algorithms." In *Proceedings of the 10th international conference on World Wide Web*, pp. 285-295. ACM, 2001.

[6] Y. Koren and R. Bell, "Advances in collaborative filtering," *Recommender systems handbook*. Springer, Boston, MA, pp. 77-118, 2015.

[7] W. S. Chin, Y. Zhuang, Y. C. Juan, and C. J. Lin, "A fast parallel stochastic gradient method for matrix factorization in shared memory systems," *ACM Transactions on Intelligent Systems and Technology (TIST)* 6(1):2, 2015.

[8] L. Baltrunas, B. Ludwig, and F. Ricci. "Matrix factorization techniques for context aware recommendation." In *Proceedings of the fifth ACM conference on*

Recommender systems, pp. 301-304. ACM, 2011.

[9] J. Kawale, H. H. Bui, B. Kveton, L. Tran-Thanh, and S. Chawla. "Efficient Thompson Sampling for Online Matrix-Factorization Recommendation." In *Advances in neural information processing systems*, pp. 1297-1305, 2015.

[10] R. Burke. "Hybrid recommender systems: Survey and experiments." User modeling and user-adapted interaction 12,(4), pp. 331-370, 2002.

[11] J. Wang, A. P. De Vries, and M. JT Reinders. "Unifying user-based and item-based collaborative filtering approaches by similarity fusion." In *Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*, pp. 501-508. ACM, 2006.

[12] G. Adomavicius and A. Tuzhilin. "Context-aware recommender systems." In *Recommender systems handbook*, pp. 191-226. Springer, Boston, MA, 2015.

[13] P. Castells, N. J. Hurley, and S. Vargas. "Novelty and diversity in recommender systems." In *Recommender Systems Handbook*, pp. 881-918. Springer, Boston, MA, 2015.

[14] A. Friedman, S. Berkovsky, and M. A. Kaafar. "A differential privacy framework for matrix factorization recommender systems." *User Modeling and User-Adapted Interaction* 26(5), pp. 425-458, 2016.

[15] H. Yi and F. Zhang. "Robust recommendation method based on suspicious users measurement and multidimensional trust." *Journal of Intelligent Information Systems* 46(2), pp. 349-367, 2016.

[16] H. F. Yu, C. J. Hsieh, S. Si, and I. S. Dhillon, "Parallel matrix factorization for recommender systems," *Knowledge and Information Systems*, 41(3), pp. 793-819, 2014.

[17] H. Li, K. Li, A. Jiyao, and K. Li., "MSGD: A Novel Matrix Factorization Approach for Large-scale Collaborative Filtering Recommender Systems on GPUs," *IEEE Transactions on Parallel and Distributed Systems*, 2017.

[18] J. Jiang, J. Lu, G. Zhang, and G. Long. "Scaling-up item-based collaborative filtering recommendation algorithm based on hadoop." In *Services (SERVICES), 2011 IEEE World Congress on*, pp. 490-497. IEEE, 2011.

[19] Apache Spark. https://spark.apache.org/

[20] Apache Mahout. https://mahout.apache.org/

[21] C. L. Liao and S. J. Lee, "A clustering based approach to improving the efficiency of collaborative filtering recommendation," *Electronic Commerce Research and Applications*, *18*, pp. 1-9, 2016.

[22] S. K. L. Al Mamunur Rashid, G. Karypis, and J. Riedl, "ClustKNN: a highly scalable hybrid model-& memory-based CF algorithm," *Proceeding of WebKDD, 2006*.

[23] S. Gong. "A collaborative filtering recommendation algorithm based on user clustering and item clustering." *JSW*5, no. 7, pp. 745-752, 2010.

[24] B. Smith and G. Linden, "Two decades of recommender systems at Amazon.com," *IEEE Internet Computing*, *21*(3), pp. 12-18, 2017.

[25] P. J. Haas, "Data-stream sampling: basic techniques and results," In *Data Stream Management*, pp. 13-44, Springer, Berlin, Heidelberg, 2016.

[26] J. S. Vitter, "Random sampling with a reservoir," *ACM Transactions on Mathematical Software (TOMS)*, *11*(1), pp. 37-57, 1985.

[27] G. Adomavicius and Y. Kwon, "Optimization-based approaches for maximizing aggregate recommendation diversity," *INFORMS Journal on Computing*, *26*(2), pp. 351-369, 2015.

[28] G, Adomavicius and Y. Kwon, "Improving aggregate recommendation diversity using ranking-based techniques," *IEEE Transactions on Knowledge and Data Engineering*, *24*(5), pp. 896-911, 2012.

[29] C. Miranda and A. M. Jorge, "Item-based and user-based incremental collaborative filtering for web recommendations," In *Portuguese Conference on Artificial Intelligence*, pp. 673-684, Springer, Berlin, Heidelberg, 2009.