Collaborative Filtering Recommendation for MOOC Application

YANXIA PANG,[1,2] YUANYUAN JIN,[1] YING ZHANG,[1] TAO ZHU[1]

[1]*School of Computer Science and Software Engineering, East China Normal University, No 3363, Road North Zhongshan, Shanghai, China*

[2]*Academy of Computer and Information, Shanghai Polytechnic University, Shanghai, China*

Received 17 August 2016; accepted 28 November 2016

ABSTRACT: With the fast development of MOOC in recent years, one MOOC platform has more than millions of learners and thousands of courses. Personality course recommendation is necessary to help people discover potentially interested courses by analyzing user behaviors. Collaborative Filtering (CF) has been shown to be effective on recommending items according to users in same interests on same items. In this paper, considering the effectiveness and efficiency of CF, we proposed a method called Multi-Layer Bucketing Recommendation (MLBR) to recommend courses on MOOC. MLBR changes learner vectors into same length dimension and scatters them into buckets which contain similar learners with more courses in common. At the same time, MLBR reduces the time cost of online, offline and update computation in CF recommendation. Furthermore, we extend MLBR with map-reduce technique to improve the efficiency. Extensive experiments on real-world MOOC datasets demonstrate the effectiveness and efficiency of the proposed model. © 2017 Wiley Periodicals, Inc. Comput Appl Eng Educ 25:120–128, 2017; View this article online at wileyonlinelibrary.com/journal/cae; DOI 10.1002/cae.21785

Keywords: MOOC; multi-layer bucketing recommendation; collaborative filtering; map-reduce

INTRODUCTION

MOOC platforms were built for learners in about 2012. Because MOOC is usually free and credit-less, anyone with an Internet connection can enroll in. By 2013, Coursera (http://www.coursera.org) as a MOOC supplier had got millions of users registered [1]. On most MOOC platforms popular courses are usually recommended. Coursetalk (http://www.coursetalk.com) learners can click on same interest learners to view their course history. Coursera (http://www.coursera.org) mails learners to recommend courses of same area that the learner is interested in. Khan Academy (https://www.khanacademy.org) gives learners record points to encourage more attendance or further learning along their past learning path in which courses usually belong to the same field. Similar interest is an important factor for course recommendation on platforms. Interest can be valued by favorite scores for the course from learners.

Current recommendation research on MOOC analyses more learning data to enrich the learner profiles solving the problem of sparsity [2–5]. That does not solve the problems of recommendation on favorite scores. This paper focuses on favorite score analysis for better recommendation efficiency and effectiveness keeping.

Considering on the sparse data, Multi-Layer Bucketing Recommendation (MLBR) uses leaner-course vectors instead of learner-course matrix for lower space cost. MLBR functions scatters learners into buckets for more chances meeting similar learners. With similarity buckets the time cost of similarity calculation and similar learner searching are both cut down. Both space cost and time cost are saved with MLBR. The function indexing method facilitates MLBR's application in map-reduce.

This paper contributes as following:

(1) It proposes a method MLBR for Collaborating filtering (CF) recommendation on MOOC.
(2) MLBR saves the time cost of CF recommendation on favorite scores of learners to courses considering on the

Correspondence to Y. Pang (yxpang@sspu.edu.cn).

offline similarity calculation, online recommendation and similarity update of recommendation.
(3) MLBR saves the space cost of learners' scores on courses by learner-course vectors.
(4) Experiments on real world datasets are conducted showing a good efficiency with effectiveness keeping.

Next the paper introduces related work in section Related Work about collaborative learning, CF recommendation in MOOC area and the technical details about Locality Sensitive Hashing (LSH) and MinHash. Section Solution of Multi-Layer Bucketing Recommendation is about the proposed solution MLBR including algorithms. Update part of MLBR is discussed showing its advantage on time cost in section Update of Multi-Layer Bucketing Recommendation. In section Multi-Layer Bucketing Recommendation on Map-Reduce MLBR and CF comparison under map-reduce shows the advantage of MLBR with less cross-node communication. Section Experiment is about experiments on the real data. The experiments compare MLBR with CF recommendation, and test the influence of parameters. Finally section Summary gives summary providing conclusion and future work.

RELATED WORK

Collaborative Learning

Collaborative learning is a learning mode reducing the loneliness of MOOCers, which is criticized as a weakness compared with traditional campus studying. It is one of the main reasons contributing to the high drop rate of MOOC. Recently some work on team formation tries to find similar learners for collaborative learning [2–4]. In the application, CF is conducted for recommendation in many methods [6]. CF in this paper is performed by k-NearestNeighbor (KNN).

CF Recommendation

Recommender systems have been applied in several virtual domain intelligent systems [7–10]. There are two commonly used recommender strategies. Content-based recommender strategy comprises the description of the course contents or items to be recommended, which are compared to the learners preferences [11]. CF strategy provides solution on learner profiles, past learning or rating histories of the previous, and current learners [9,12]. Collaborative system offers personalized recommendation to learners based on their interest according to the rating history of different learners [10,13–15]. Their recommendation scores concern about learner similarity and favorite of the similar learners for the course. Top k highest scored courses will be recommended [16].

CF recommendation is improved in recent work. As an attender in KDD cup Toscher analyzed CF for recommendation in different methods [6]. But it did not solve the problem of sparsity. Sun calculates learner similarity by meta path. MOOC learning paths conversion enriches the data of learner portfolios. Chen adopted path similarity for heterogeneous relationship between user and item portfolios [2]. Wu improved the model for predicting scores with more factors considered including mistake and slide probability [5]. That improves CF with more factors considered. But CF recommendation on favorite scores is not considered by them. This paper proposes a solution for it.

To recommend for a learner, CF recommender system needs calculating the learners' similarity values to all other learners to find out most similar learners. Similarity calculation must be executed repeatedly to refresh the similarity values in accordance to any activity of MOOC learners [16]. Number of MOOC learners on one platform is millions and even more. n is used as the number of learners. All learner pairs number n^2 reaches billions. In the offline part similarity values among every learner pairs are calculated as Equation (1) shows. To calculate n^2 similarity values Equation (1) must be executed n^2 times. The time cost of offline similarity calculation reaches $O(n^2)$. The paper aims to cut down the time cost of offline part of similarity calculation, online part of query among big data set and the update part of related similarity refreshment.

$$sim(l_1, l_2) = (|l_1| \cap |l_2|)/(|l_1| \cup |l_2|) \quad (1)$$

MOOC recommendation is conducted online requiring immediate response, so the most time-consuming job is left offline. Taking user CF recommendation as an example, it can be divided into two parts. One is to calculate the similarity between all learner pairs offline. Top similar learners were selected out. The other part calculates and ranks online for top scored courses among courses enrolled in by top similar learners.

LSH and MinHash

LSH is a method for indexing. Hash indexing can classify learners into different buckets with hash functions. With functions, we can search a learner in its corresponding buckets instead of all the list. LSH works with special functions. The function keeps the feature of similarity. Similar learners are more possible to be in the same buckets after the function calculation. The distance between learners is kept after function transmission. With LSH the system just needs to project the learner into its buckets and check its neighbors in the buckets to find similar learners. A learner may be projected into more than one bucket to meet more neighbors.

The team uses Jaccard distance to describe the similarity between two learners. MinHash functions keep the property of Jaccard distance and the similarity of learners [17]. That also helps for reducing the high dimensions of a learner-course vector and transmitting vectors into same dimension [18]. Next MLBR uses LSH functions to scatter similar learners into same buckets. Thus, any learner's similar neighbors can be found out faster without reading all the similarity pairs.

SOLUTION OF MULTI-LAYER BUCKETING RECOMMENDATION

The author aims to cut down the time cost of course recommendation and find a recommendation solution suitable for map-reduce named as MLBR. MLBR scatters learners into different buckets. The bucketing keeps the distance feature of learners so that similar learners have more chance to be in same buckets. Similar learners can be found only among buckets they belong to. Recommendation does not have to take all the learners into account about their similarity.

Notation

We define following notations for later description.

Symbols	Description
m	The number of courses
m	The number of learners
L	A collection of learner-course vectors l_i with courses (c_{i1}, c_{i2},...) enrolled in *valued* ($c_{i1,}$ c_{i2}, ...)
k_1	The number of top similar learners
k_2	The number of top courses for recommendation.
d	The number of MinHash functions
b	The number of LSH functions
r	A learner vector to be recommended to
c_r	Means courses to be recommended to
S	The learner collection for top $k_1$most similar learner
$score_{ic}$	The favorite score to course c by learner i. Usually it is at most 10 points representing five stars.
sim_{rs}	The similarity between learner r and learner s

Structure of Multi-Layer Bucketing Recommendation

Figure 1 shows the process of bucketing. l_i is the course list of learner i which is finally indexed to some buckets by the Multi-Layer calculation of Figure 1. First l_i is dealt with d MinHash functions. The functions transact the vector l_i into values $m_1(l_i)$, $m_2(l_i)$,... $m_d(l_i)$. The number of MinHash functions is defined as d and the number of location hash functions as b. Every d/b MinHash values of l_i is grouped as a band for b LSH function transaction. Values generated by LSH functions are used as the bucket index values of learner l_i.

To improve the response time of online recommendation MLBR uses buckets to store similar learners. Number of courses that one learner learned varies from zero to hundreds. The learner-course matrix is usually huge and sparse. In view of this case, learner-course enrollment is saved with inverse list $l_i = (c_{i1}, c_{i2},...)$. c_{i1} is one of the courses enrolled in by learner l_i. Lengths of learner vectors are different. MinHash functions turn vector l_i into d values to solve this problem. The dimension of vector l_i varies with the course number of learner l_i enrolled in. MLBR deals with the vectors l_i by MinHash functions which can decrease the dimension number and change them into same dimension numbers. And MinHash functions keep the Jaccard distance in accordance to the distance before function calculation [18].

With MinHash MLBR can transmit the vector l_i into MinHash values. That controls the vector dimensions.

Then with location sensitive functions the MinHash values are transacted into index of LSH buckets. Functions must keep the similarity distance between two learners. Hash functions spread learners into buckets and keep similar learners in the same buckets with higher probability. Given a vector of learner l_i with courses (c_{i1}, c_{i2},...) enrolled in, MLBR changes the vector l_i into values indicating the index of buckets. Similar learners have more probability to scatter into same buckets. The algorithm adopts more functions to increase the opportunity that similar learners meet in same buckets [18].

Algorithm of Multi-Layer Bucketing Recommendation

When recommending for learner l_r (c_{r1}, c_{r2},...), MLBR calculates with LSH functions on values from MinHash functions on the vector r and finds the buckets it belongs to, then compare the learners in same buckets to find top k_1 similar learners. The similarity is defined as Equation (1). For the top k_1 similar learners, the algorithm will compare all courses they enrolled in. Recommendation scores of the courses are summed up by similarity and favorite score of all learners enrolling in the course as Equation (2) according to CF recommendation. The scores consider on the favorite scores of courses from their enrollers and their similarities with the learner to be recommended. Top k_1 scored courses are recommended to the learner. The time complexity is $O(k_1 * k_2)$.

The algorithm is discussed in next section. Its time complexity is $O(e * n)$ ($e \leq 1$) [19]. Learners in same buckets

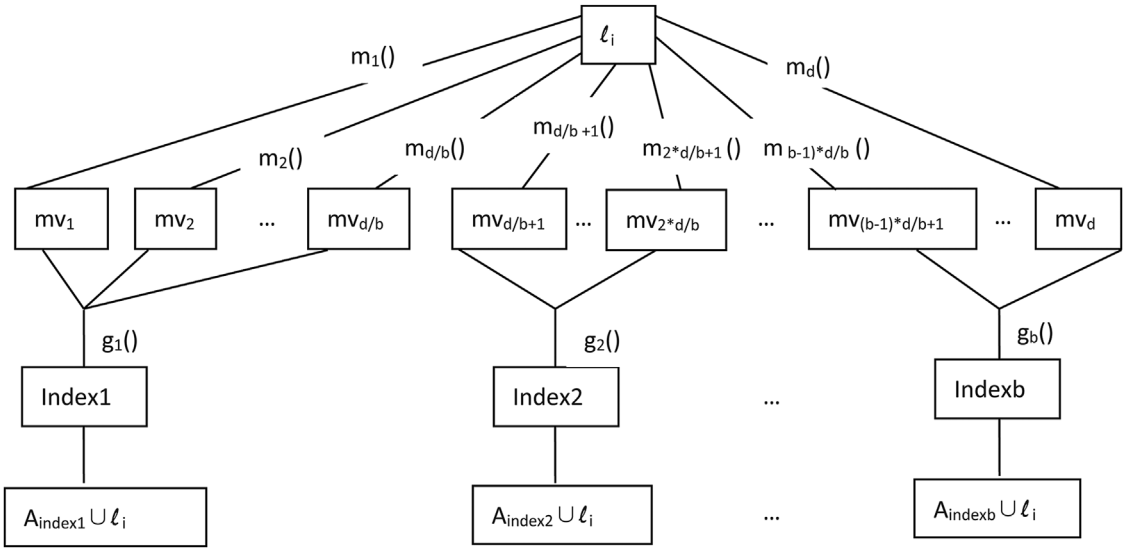

Figure 1 Structure of MLBR.

can be stored in same data node [20].

$$R_c = \sum_{s=1}^{M} sim_{rs} \times score_{sc} \quad (2)$$

Algorithm 1 is for similarity bucketing of MLBR. In the algorithm L is collection of learners l_i ($0 < i < n$). (g_1, ..., g_b) are LSH functions and (m_1, ..., m_d) are MinHash functions. Each learner vector l_i is calculated by d MinHash functions (m_1, ..., m_d) into d values among which every d/b values are combined together for LSH functions (g_1, ..., g_b). The values generated by LSH functions are the index values of buckets that learner l_i belongs to. Then learner l_i is inserted into corresponding buckets. As Algorithm 1 shows, MinHash functions turn vector l_i into many d/b dimension vectors that are hashed by functions (g_1, ..., g_b) into b hash buckets. The buckets are stored in LSH tables A.

Algorithm 2 describes the recommendation part on buckets generated in Algorithm 1. It calculates the similarity scores of learners in same buckets and ranks out the top k_1 similar learners. Then as candidates all courses of top k_1 similar learners are calculated for recommendation scores. The top k_2 courses are returned as the recommendation result. In Algorithm 2, lines 1–4 find buckets of learner r to be recommended. Lines 5–7 work on each neighbor in r's buckets, calculate the similarity and rank out the top k_1 similar learners. Line 8 ranks for top k_1 similar learners of r. Lines 9–13 calculate the recommendation values of courses enrolled in by top k_1 learners, then line 14 ranks the values and line 15 returns the top k_2 courses as the recommendation result for learner r.

ALGORITHM 1: Similarity bucketing of MLBR

Input: Learner collection $L = \{l_1, l_2, \ldots, l_n\}$, hash function $g_1, g_2 \ldots, g_b$, MinHash functions $m_1, m_2, \ldots, m_d$;
Output: LSH table $A_1, A_2 \ldots, A_b$;

```
1   for all l in L do
2       for i = 1 to b do
3           index = g_i(m_{(i-1)*d/b+1}(l), ..., m_{i*d/b}(l));
4           A_i(index) = A_i(index) ∪ l
5       end
6   end
7   Return A_1, A_2, ..., A_b
```

ALGORITHM 2: Recommendation on buckets

Input: A learner vector to be recommended r, hash functions $g_1, g_2, \ldots, g_b$, LSH table $A_1, A_2, \ldots, A_b$;
Output: Courses to be recommended c_r

```
1   for i = 1 to b do
2       index = g_i(m_{(i-1)×d/b+1}(r), ..., m_{i×d/b}(r));
3       A_i(index) = A_i(index) ∪ r
4   end
5   for each learner l in buckets learner r belonging to do
6       rank learner l in r 's buckets
7   end
8   get top k_1 similar learners S
9   for each learner s of S do
10      for each course c of learner s do
11          Rc+ = sim_rs × score_sc
12      end
13  end
14  get top k_2 high valued courses c_r
15  return c_r
```

UPDATE OF MULTI-LAYER BUCKETING RECOMMENDATION

Whenever a new learner activity like a new course enrollment occurs, the similarity will change correspondingly. As a consequence the offline part must be refreshed repeatedly.

With the method of CF recommendation similar learners are selected from all learners. Any learner-course activity may change the values of similarities between related pairs of the learner. CF recommendation must refresh all the similarity pairs related to the learner. The similarity matrix must be recalculated every short period. As an offline task, the similarity refreshment can be executed repeatedly. Similarity refreshment's time cost is O(n).

With LSH buckets similar learners are among those in same buckets. The algorithm just recalculates on learner vectors that changed and refreshes the related buckets. When a learner enrolls in more courses, the course vector of the learner changes. To update the similarity buckets MLBR only re-buckets the vector and changes its buckets as line 1–4 do in Algorithm 2. The re-bucket work costs only O(1).

Equation (1) defines similarity by the percentage of common courses enrolled in. It uses Jaccard similarity between two learners which is the proportion of their courses in common among total courses. The numerator is the number of course enrolled in by both l_1 and l_2. The denominator is total number of courses participated by both learners l_1 and l_2. When learner l_i enrolls in another course, the similarity between learner l_i and other learners will change according to Equation (1). The numerator in Equation (1) of common courses and the denominator will both change. So with CF recommendation every learning activity of learner l_i result in similarity refreshment of all similarity pairs leaner l_i related to.

MLBR can recalculate the changed learner vectors and refresh their buckets for updating more easily. When a learner enrolls in more courses its vector will also change. To update the similarity buckets the algorithm just needs re-bucketing the vector and change its buckets. As Algorithm 1 shows, the changed learner vector l_i is used for index calculation with MinHash functions and LSH bucket functions. Then MLBR can refresh learner l_i 's index values of buckets. The experiment part lists time cost comparison in different parts of recommendation.

MULTI-LAYER BUCKETING RECOMMENDATION ON MAP-REDUCE

Map-reduce is widely used in big data processing for time saving. MLBR has its advantage in map-reduce. The application of CF and MLBR in map-reduce are discussed as following in comparison.

Map-Reduce on CF

For a learner to be recommended with CF recommendation, his course enrollments will be compared with others one by one until finding learners who have most courses in common. As Figure 2

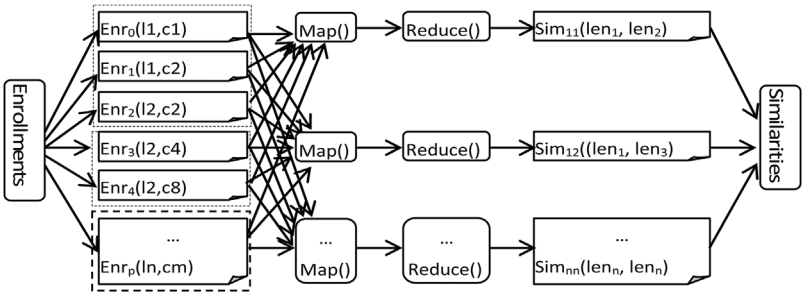

Figure 2 Map-reduce of similarity calculation in CF recommendation.

shows, (learner, course) pairs are mapped to different data nodes. To find the most similar learners clients check all data nodes. Thus, communication between data nodes is very dense which costs more time. Top similar learners are searched among n^2 data. The time cost reaches $O(n^2)$ [21].

Map-Reduce on MLBR

Figure 3 shows the similarity buckets generation in map-reduce of MLBR. All the operation is executed in lines without cross communication between data nodes. Slaves do not have to read from different data nodes. With similar neighbor buckets MLBR will be faster. The learner to be recommended is hashed to find its buckets in which the learners are selected out as candidates for top similar learners. The time cost is $O(q * n)$. q is less than one. Next job is to calculate on top similar learners' courses and find the most recommendable courses which can also be shortened with bucket index. So MLBR can be better applied for map-reduce than CF recommendation.

EXPERIMENT

Data

We use reviewing data crawled from Coursetalk (http://www.coursetalk.com). Coursetalk is a website providing courses from different suppliers. It also provides course reviews and scores from learners. Learner ratings on courses are used as the main feature for similarity consideration. The team of this paper can only crawl data of courses with reviews and scores. Information of reviewers are more reliable. So this paper adopts only learners and courses with reviews [22]. In fact, most learners do not review. So the data are only little part of factual data on Coursetalk. To specify the result comparison, the team expands the data according to their distribution [23]. Finally, the data set includes 4,612 learners, 304 courses, and 100,000 enrollments as Table 1 lists. Figure 4 gives the final few record of learner rating record. The fields include UserId, Star, StarTime, and CourseId. Field Star is the value of stars given by the user for the course. Field StarTime is the time when the user gave the star remark. Learners rate courses with 0–5 stars representing 0–10 points. The measurement uses precision, recall, *F*-score, and time cost to show the effectiveness and efficiency.

The experiments ran on a computer with 4 GB RAM and CPU 1.6 GHz. The algorithms are in Java. The data are stored in sql files. For fast access and less storage data of sql files is selected out and written into text files. Without loss of generation 4,000 learners are recommended for test. The time values are in second which is expressed by "*s*" in Tables 2 and 3.

Experiment Results

CF Recommendation and Multi-Layer Bucketing Recommendation Comparison. We run CF recommendation and MLBR on the Coursetalk data set. The result compares CF recommendation and MLBR showing as Table 2.

Table 2 shows CF recommendation works better in recall and worse in precision and *F*-score. CF recommendation finds more correct learners in ground truth. But it tries to recommend more which

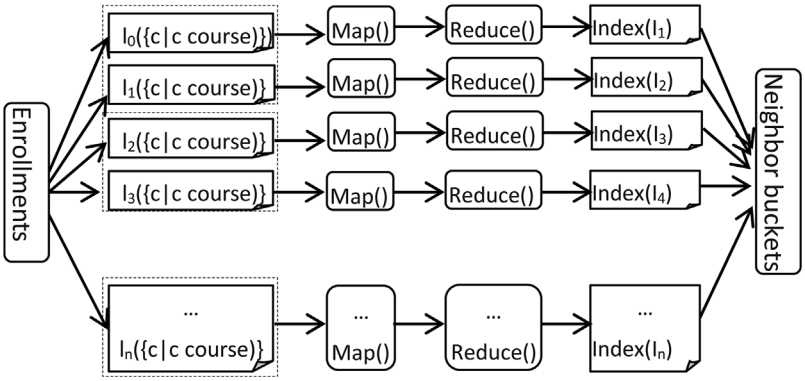

Figure 3 Map-reduce of similarity calculation in MLBR.

Table 1 Data Numbers

Data	Learners	Courses	Enrollments
Numbers	4612	304	100000

decreases the precision. *F*-score indicates the balance of precision and recall [24]. In general CF recommendation does not keep such a good balance as MLBR with high recall and low precision.

Values of Table 2 are of time cost for 4,000 learners' recommendation needs less than 1 s. The index work of neighbor bucketing shows its advantage in time cost. That demonstrates that MLBR works better in recommendation.

1	userID	star	stardate	courseID
999980	23770	10	2013/3/3	18638
999981	23770	10	2013/11/23	16331
999982	23770	10	2014/6/8	756
999983	23770	10	2013/4/3	14644
999984	23770	10	2014/6/7	13372
999985	23770	10	2013/2/19	14644
999986	23770	10	2012/12/7	18638
999987	23770	10	2012/8/22	8955
999988	23770	10	2014/6/8	8955
999989	23770	10	2014/3/18	13372
999990	23770	10	2013/2/26	756
999991	23770	10	2014/2/20	5824
999992	23770	10	2014/3/1	16331
999993	23770	10	2013/7/4	18638
999994	23770	10	2014/3/5	13372
999995	23770	10	2013/6/19	13372
999996	23770	10	2014/2/28	16331
999997	23770	10	2014/4/21	5824
999998	23770	10	2014/3/5	13372
999999	23770	10	2014/8/4	756
1000000	23770	10	2014/2/16	756
1000001	23770	10	2013/11/15	5824

Figure 4 Expanded data sample of Coursetalk.

recommendation. Time cost of similarity generation increases a little with 212 s in MLBR and 104 s in CF recommendation. It spends more time on function calculation and vector regeneration. The data are not so big. The learner vector does not have so many courses enrolled in. Bucketing wastes a little on vector regeneration which does not seem so necessary. With bigger data set the hash functions may show its advantages in dimension reduction. The total time cost of CF recommendation is 8,198 s about 7× that of MLBR 1,152 s. The reduction of recommendation time contributes to that. It decreases from 8,094 s in CF recommendation to 940 s in MLBR. One learner's

Table 3 lists the time cost of different parts of the recommendation and make a comparison on that between CF and MLBR. Columns of CF and MLBR are the time cost expectation. Columns of CF experiment and MLBR experiment are time cost values in the experiment. Offline similarity is time cost about the offline task of similarity calculation. In CF recommendation, the offline similarity is to calculate the similarity values between every two learners. Time cost of CF's offline similarity is $O(n^2)$. And n is the number of learners. In MLBR offline similarity is to calculate the bucket numbers of every learner to make sure similar learners in same buckets. The time cost is almost only on hash calculation $O(n)$. The part of "Online similar learners" is to find top k_1 similar learners in online part. To CF recommendation online similar learners is to search for the top k_1 similar learners among n^2 learner similarity values. The time cost is $O(n^2)$. To MLBR "Online similar learners" is to find top similar learners among those in same buckets. The time cost is $O(q * p)$. Here, p is the number of buckets of recommended learner and q is the number of learners in one bucket.

"Update on user activity" means the update of similarity when learner-course vectors change. Update of CF recommendation needs recalculating the similarity values between the active learner and all other learners, so the time cost is $O(n)$. Update of MLBR is executed only by calculation for the bucket numbers of the active learner, so the time cost is $O(1)$.

According to the tables the time reduction is obvious. Time cost on online and update part are both cut down. Especially on update part the improvement is obvious. In similarity calculation, the task does not pick the user vector calculation out, so it does not increase with the learner amount. The time cost is cut down in both offline part and update part.

Parameters of Candidate Numbers. Multi-Layer Bucketing helps to find similar learners quickly. k_1 is the number of top similar users and k_2 is the number of courses to be recommended. Values of k_1 and k_2 determine the numbers of candidates. More candidates expand the coverage of recommendation. But Figure 5 shows when the candidate number increases too much it may make the precision down and increase the time cost. The author changes the value k_1 and k_2 to find a best combination as Figure 5 shows. It can be found when k_1 and k_2 increase the precision and recall balance better and *F*-score increases too. When (k_1, k_2) is set (25, 15), the values go down. With the increase of k_1 and k_2, the time cost increases too. So (20, 20) is adopted as the best parameters value for further experiments.

Parameters of Function Numbers. Next test is on parameters d and b in Multi-Layer bucketing. d is the number of MinHash functions and b is the number of LSH functions. Those two values change the bucket numbers and also influence the metrics

Table 2 Comparison Between CF and MLBR

Recommendation	Precision	Recall	*F*-score	Total-time	Similarity-time (s)	Recommend-time (s)
CF	0.15	0.42	0.11052	8198	104	8094
MLBR	0.34	0.20	0.25	632	212	940

Table 3 CF and MLBR Comparison on Time Cost

Task	CF	CF experiment (s)	MLBR	MLBR experiment (s)
Offline similarity	$O(n^2)$	70.2	$O(n)$	1.76
Online similar learners	$O(n^2)$	43	$O(d * p)$	0.562 (d = 16)
Update on user activity	$O(n)$	10.8	O(1)	0.00038161 (1 LSH)

s, second; d = 16, the parameter d selects 16; 1 LSH, the time value is of one time LSH calculation.

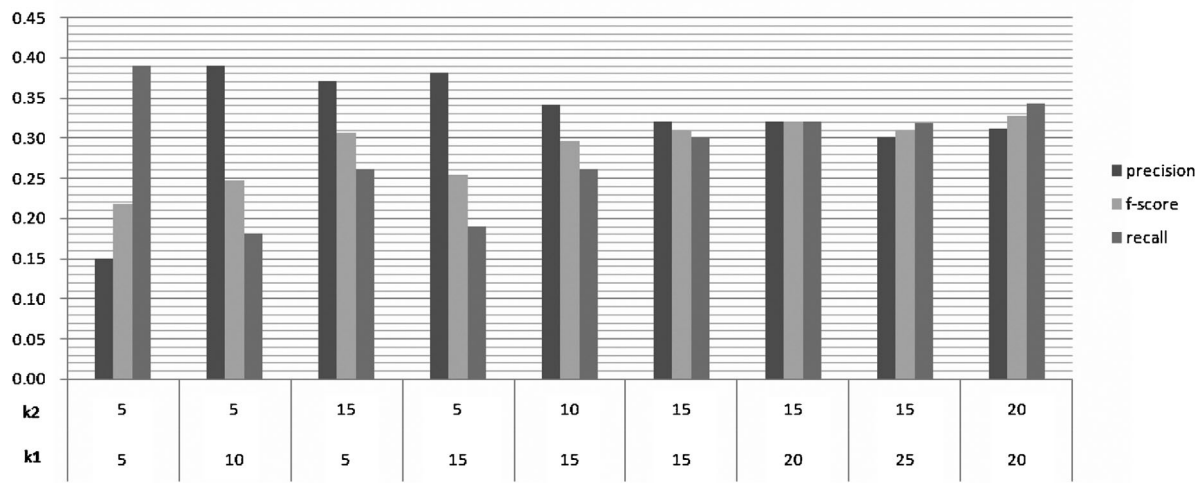

Figure 5 Candidate numbers' influence on result of recommendation.

including quality and time cost. As Figure 6 shows, the value of d/b influences the result too. When (d, b) valued as (16, 8) and (32, 16), similar learners have more possibility to meet in same buckets. That helps to increase the quality of result. The result shows the data are still sparse. Though the result of (32, 16) is a little better than (16, 8), but the time cost increases 10×. The vectors generated by MinHash functions are scattered into more buckets by more LSH functions. That increases the workload rapidly. The values of functions influence the workload quite a lot. Values of (d, b) should be adapted when the data size changes. The relationship between (d, b) and data size is for future research.

Numbers of Functions Influence on Time. The numbers of functions influence a lot on recommendation including bucket numbers and recommendation numbers. As a result, they influence the running time of similarity bucket scattering and

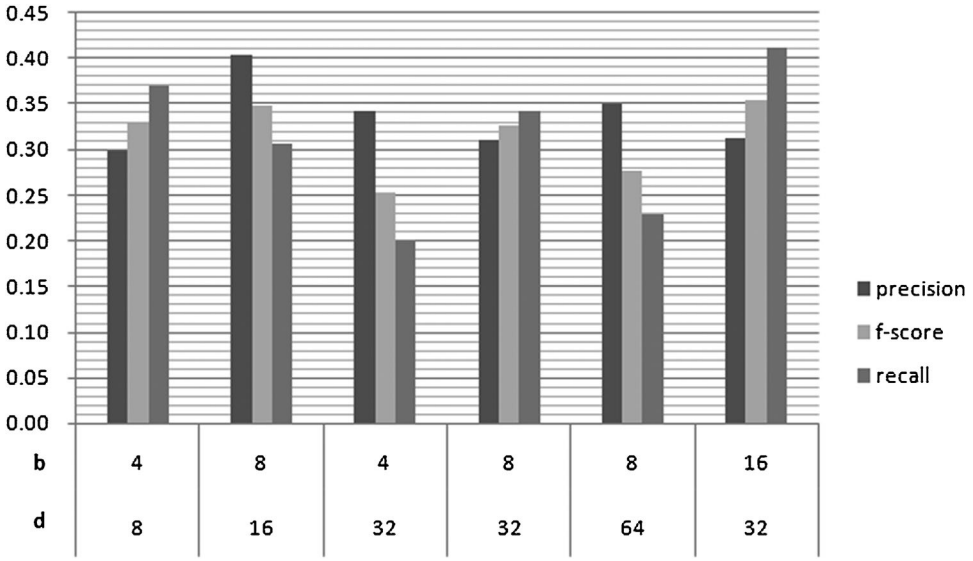

Figure 6 Function numbers' influence on recommendation result.

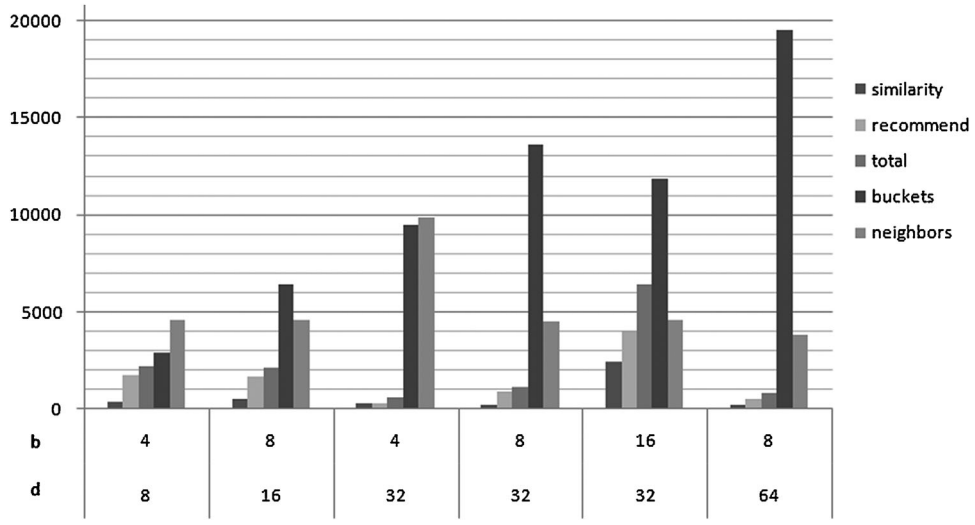

Figure 7 Function numbers' influence on time cost.

recommendation as Figure 7 shows. It is obvious the bucket numbers and neighbors both increase rapidly with the increase of d and b. Here similarity and recommendation are the time cost of the operations. The value of d/b seems to influence the similarity and recommendation time more. The less d/b is, the longer the running time lasts. But that is not the only determinative reason. The increase of b results in more values to be handled. when d/b decreases, the bucket number falls down and there are more similar learners in a bucket. The values of d, b and hash functions determine the running time and workload.

SUMMARY

MOOC recommendation is conducted on interest for better collaborative learning on platforms of MOOC suppliers. CF recommendation considers on interest by items of similar users, which fits well for collaborative learning in group. Learners' interest can be represented by favorite scores. This paper proposes an improved CF named as MLBR to fix data sparsity and cutting down the time cost on recommendation including offline similarity calculation, online similarity research, and update of similarity. The experiments show the improvement on time cost with quality keeping. MLBR shows advantage with less cross-node communication under map-reduce. It can be found in experiments parameters of MLBR influence the results. Best values of parameters vary with the data size. That is a problem worth further research in future.

REFERENCES

[1] L. Pappano, *The year of the mooc*, The New York Times 2 (2012), no. 12, 2012.

[2] Y. Chen, X. Zhao, J. Gan, J. Ren, and Y. Hu, Content-based top-n recommendation using heterogeneous relations, Databases Theory and Applications: 27th Australasian Database Conference, ADC 2016, Sydney, NSW, September 28-29, 2016 Proceedings, M. A. Cheema, W. Zhang, and L. Chang (Eds.), Springer International Publishing, Cham, Switzerland, 2016, pp 308–320.

[3] Y.-T. Lin, Y.-M. Huang, and S.-C. Cheng, An automatic group composition system for composing collaborative learning groups using enhanced particle swarm optimization, Compu Educ 55 (2010), 1483–1493.

[4] Y. Liu, Q. Liu, R. Wu, E. Chen, Y. Su, Z. Chen, and G. Hu, Collaborative learning team formation: A cognitive modeling perspective, Database systems for advanced applications: 21st international conference, DASFAA 2016, Dallas, TX, April 16-19, 2016, Proceedings, Part II, S. B. Navathe, W. Wu, S. Shekhar, X. Du, S. X. Wang, and H. Xiong (Eds.), Springer International Publishing, Cham, Switzerland, 2016, pp 383–400.

[5] R. Wu, Q. Liu, Y. Liu, E. Chen, Y. Su, Z. Chen and G. Hu. Cognitive modelling for predicting examinee performance, Proceedings of the 24th International Conference on Artificial Intelligence, AAAI Press, 2015, pp 1017–1024.

[6] A. Toscher and M. Jahrer, Collaborative filtering applied to educational data mining, KDD cup (2010).

[7] J. O'Donovan and B. Smyth, Trust in recommender systems. In: Proceedings of the 10th International Conference on Intelligent User Interfaces, ACM, San Diego, CA, 2005, pp 167–174.

[8] D. O'Sullivan, D. Wilson, and B. Smyth, Improving case-based recommendation, Advances in case-based reasoning: 6th European Conference, ECCBR 2002 Aberdeen, Scotland, UK, September 4–7, 2002 Proceedings, S. Craw and A. Preece (Eds.), Springer Berlin Heidelberg, Berlin, Heidelberg, 2002, pp 278–291.

[9] J. S. Breese, D. Heckerman, and C. Kadie, Empirical analysis of predictive algorithms for collaborative filtering. In: Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence, Morgan Kaufmann Publishers Inc., Madison, WI, 1998, pp 43–52.

[10] P. Resnick, N. Iacovou, M. Suchak, P. Bergstrom, and J. Riedl, Grouplens: An open architecture for collaborative filtering of netnews. In: Proceedings of the 1994 ACM conference on Computer supported cooperative work, ACM, Chapel Hill, NC, 1994, pp 175–186.

[11] P. Melville, R. J. Mooney, and R. Nagarajan, *Content-boosted collaborative filtering for improved recommendations*, Aaai/iaai, 2002, pp 187–192.

[12] J. O'Donovan and J. Dunnion, A framework for evaluation of information filtering techniques in an adaptive recommender system,

Computational linguistics and intelligent text processing: 5th International Conference, CICLing 2004 Seoul, Korea, February 15-21, 2004 Proceedings, A. Gelbukh (Ed.), Springer Berlin Heidelberg, Berlin, Heidelberg, 2004, pp 502–506.
[13] D. F. O. Onah and J. Sinclair, Collaborative filtering recommendation system: a framework in massive open online courses. 9th International Technology, Education and Development Conference, Madrid, Spain, Mar 2-4 2015 Proceedings. INTED Madrid, Spain, 2015, pp 1249–1257.
[14] U. Shardanand and P. Maes, Social information filtering: Algorithms for automating "word of mouth". In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, ACM Press/ Addison-Wesley Publishing Co., Denver, CO, 1995, pp 210–217.
[15] M. Balabanović and Y. Shoham, Fab: Content-based, collaborative recommendation, Commun ACM 40 (1997), 66–72.
[16] L. Xiang, Recommendation system practice, Beijing Youdian Publication House, BeiJing, 2012, 6.
[17] J. D. Ullman, J. Leskovec, and A. Rajaraman, Mining of massive datasets, Cambridge University Press, 2011.
[18] M. Datar, N. Immorlica, P. Indyk, and V. S. Mirrokni, Locality-sensitive hashing scheme based on *p*-stable distributions. In: Proceedings of the Twentieth Annual Symposium on Computational Geometry, ACM, Brooklyn, NY, 2004, pp 253–262.
[19] H. Chen and D. R. Karger, Less is more: Probabilistic models for retrieving fewer relevant documents. In: Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, ACM, Seattle, WA, 2006, pp 429–436.
[20] T. Zhu, Y. Lin, J. Cheng, and X. Wang, Efficient diverse rank of hot-topics-discussion on social network. Web-age information management: 15th International Conference, WAIM 2014, Macau, China, June 16-18, 2014 Proceedings, F. Li, G. Li, S.-W. Hwang, B. Yao, and Z. Zhang (Eds.), Springer International Publishing, Cham, Switzerland, 2014, pp 522–534.
[21] Z. Rasheed and H. Rangwala, A map-reduce framework for clustering metagenomes, Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW), 2013 IEEE 27th International, IEEE, Piscataway, NJ, 2013, pp 549–558.
[22] R. O'Toole, Pedagogical, strategies and technologies for peer assessment in massively open online courses (MOOCS). University of Warwick, Academic Technology Service. Discussion Paper. University of Warwick, Coventry, UK: University of Warwick. (Unpublished), (2013). http://wrap.warwick.ac.uk/54602/.
[23] L. Gu, M. Zhou, Z. Zhang, M.-C. Shan, A. Zhou, and M. Winslett, Chronos: An elastic parallel framework for stream benchmark generation and simulation, 2015 IEEE 31st International Conference on Data Engineering, IEEE, Piscataway, NJ, 2015, pp 101–112.
[24] N. Barbieri, G. Manco, and E. Ritacco, Probabilistic approaches to recommendations, Synthesis Lectures on Data Mining and Knowledge Discovery 5 (2014), 1–197.

BIOGRAPHIES

Yanxia Pang was born in Shandong, China, in 1980. She is studying in East China Normal University for PhD degree. At the same time she is also a teacher in Shanghai Polytechnic University. Her research interest focuses on educational data analysis and recommendation. She follows the development of MOOC and online education for many years. She combines learning rules and application data analysis for better online education research.

Yuanyuan Jin is currently a graduate in Computer Science at East China Normal University of Science and Technology. She received her Bachelor in Computer Science from East China Normal University in Shanghai, P. R. China in 2016. Her current research interests include location privacy protection and data mining.

Ying Zhang, Master degree candidate, East China Normal University. Main research directions are as follows: Recommender system, and user behavior analyzing.

Tao Zhu is currently a PhD candidate at School of Computer and Software, East China Normal University. He received his BSc in information security, from Nanjing University of Posts and Telecommunications, in 2012. His research interests include in-memory database, transaction processing, and distributed system.