LCBM: Statistics-based Parallel Collaborative Filtering*

Fabio Petroni[1], Leonardo Querzoni[1], Roberto Beraldi[1], and Mario Paolucci[2]

[1] Department of Computer Control and Management Engineering Antonio Ruberti, Sapienza University of Rome - petroni|querzoni|beraldi@dis.uniroma1.it
[2] Institute of Cognitive Sciences and Technologies, CNR - mario.paolucci@istc.cnr.it

Abstract. In the last ten years, *recommendation systems* evolved from novelties to powerful business tools, deeply changing the internet industry. Collaborative Filtering (CF) represents today's a widely adopted strategy to build recommendation engines. The most advanced CF techniques (i.e. those based on matrix factorization) provide high quality results, but may incur prohibitive computational costs when applied to very large data sets. In this paper we present Linear Classifier of Beta distributions Means (LCBM), a novel collaborative filtering algorithm for binary ratings that is (i) inherently parallelizable and (ii) provides results whose quality is on-par with state-of-the-art solutions (iii) at a fraction of the computational cost.

Keywords: Collaborative Filtering, Big Data, Personalization, Recommendation Systems.

1 Introduction

Most of todays internet businesses deeply root their success in the ability to provide users with strongly personalized experiences. This trend, pioneered by e-commerce companies like Amazon [1], has spread in the last years to possibly every kind of internet-based industries. As of today, successful players like Pandora or StumbleUpon provide user personalized access to services like a core business, rather than an add-on feature.

The fuel used by these companies to feed their recommendation engines and build personalized user experiences is constituted by huge amounts of user-provided data (ratings, feedback, purchases, comments, clicks, etc.) collected through their web systems or on social platforms. For instance, the Twitter micro-blogging service has surpassed 200 million active users, generating more than 500 million tweets (micro-blog posts) per day at rates that recently (Aug 2013) peaked at 143199 tweets per second [2]. The amount of data available to be fed to a recommendation engine is a key factor for its effectiveness [3]. A further key factor in this context is represented by timeliness: the ability to timely provide users with recommendations that fit their preferences constitutes a potentially enormous business advantage [4].

* This work has been partially supported by the TENACE PRIN Project (n. 20103P34XC) funded by the Italian Ministry of Education, University and Research.

A widely adopted approach to build recommendation engines able to cope with these two requirements is represented by *Collaborative filtering* (CF) algorithms. The essence of CF lies in analyzing the known preferences of a group of users to make predictions about the unknown preferences of other users. Research efforts spent in the last ten years on this topic yield several solutions [5,6,7,8] that, as of today, provide accurate rating predictions, but may incur prohibitive computational costs and large time-to-prediction intervals when applied on large data sets. This lack of efficiency is going to quickly limit the applicability of these solutions at the current rates of data production growth, and this motivates the need for further research in this field.

In this paper we introduce *Linear Classifier of Beta distributions Means* (LCBM), a novel algorithm for collaborative filtering designed to work in systems with binary ratings. The algorithm uses ratings collected on each item (i.e. products, news, tweets, movies, etc) to infer a probability density function shaped as a Beta distribution; this function characterizes the probability of observing positive or negative ratings for the item. A linear classifier is then used to build user profiles that capture the aptitude of each user to rate items positively or negatively. These profiles are leveraged to predict ratings users would express on items they did not rate. Our algorithm is able to provide predictions whose quality is on-par with current state-of-the-art solutions (based on matrix factorization techniques), but in shorter time and using less computational resources (memory occupation). Moreover, it is inherently parallelizable. Its performance has been extensively assessed through an experimental evaluation based on well-known public datasets (MovieLens and Netflix) and compared with those offered by open source implementations of state-of-the-art solutions.

The rest of this paper is organized as follows: Section 2 presents related works; Section 3 defines the system model and states the problem; Section 4 presents our solution, evaluated in Section 5; finally, Section 6 concludes the paper.

2 Related Work

Collaborative Filtering (CF) is a thriving subfield of machine learning, and several surveys expose the achievements in this fields [9,10]. CF solutions in the literature are often divided in two groups: *memory-based* and *model-based* [11].

Memory-based methods [12,13] are used in a lot of real-world systems because of their simple design and implementation. However, they impose several scalability limitations that make their use impractical when dealing with large amounts of data. The slope one algorithms [14] were proposed to make faster prediction than memory-based algorithms, but they were unable to overcome the scalability issues of the latter.

Model-based approaches have been investigated to overcome the shortcomings of memory-based algorithms. The most successful Model-based techniques are by far those based on low-dimensional factor models, as the Netflix Prize (www.netflixprize.com) established, in particular those based on *matrix factorization* (MF) [15,16,5,6]. These methods aims at obtaining two lower rank matrices P and Q, for users and items respectively, from the global matrix of ratings

R, with minimal loss of information. The most popular MF solutions are *Alternating Least Squares* (**ALS**) and *Stochastic Gradient Descent* (**SGD**). Both algorithms need several passes through the set of ratings to achieve this goal. ALS [5] alternates between keeping P and Q fixed. The idea is that, although both these values are unknown, when the item vectors are fixed, the system can recompute the user vectors by solving a *least-squares* problem (that can be solved optimally), and vice versa. SGD [6] works by taking steps proportional to the negative of the gradient of the error function. The term *stochastic* means that P and Q are updated, at each iteration, for each given training case by a small step, toward the average gradient descent. Some recent works aimed at increasing the scalability of current MF solutions [7,17,8], however the asymptotic cost of these techniques makes it difficult to fit the timeliness requirements of real-world applications, especially when applied on large data sets. Furthermore, each update leads to non-local changes (e.g. for each observation the user vector increment in SGD is proportional to the item vector, and vice versa) which increase the difficulty (i.e. the communication costs) of distributed implementations.

The binary-rating scenario we consider in this work can be considered as a special case of the more general multi dimensional rating scenario. However it is worth noticing that it fundamentally differs from *one-class collaborative filtering* [18] where only positive feedback are assumed to be available while negative feedback are treated as absent. Contrarily, in our work negative feedback is always considered at the same level of importance as positive feedback, but with an opposite meaning.

Some earlier works on collaborative filtering [19,20] and reputation [21] adopted the same statistical method (i.e. Beta distribution) to combine feedback. Ungar and Foster [19] proposed a clustering CF approach in which the connection probabilities between user and item clusters are given by a Beta distribution. The solution is computationally expensive, as Gibbs sampling is used for model fitting. Wang et al. [20] applied information retrieval theory to build probabilistic relevance CF models from implicit preferences (e.g. frequency count). They use the Beta distribution to model the probability of presence or absence of items in user profiles.

3 System Model and Problem Definition

We consider a system constituted by $U = (u_1, \cdots, u_N)$ users and $I = (i_1, \cdots, i_M)$ items. Items represent a general abstraction that can be case by case instantiated as news, tweets, shopping items, movies, songs, etc. Users can rate items with values from a predefined range. Rating values X can be expressed in several different ways (depending on the specific system), however, in this paper we will consider only binary ratings, thus $X = \{-1, 1\}$ where the two values can be considered as corresponding to *KO* and *OK* ratings respectively.

By collecting user ratings it is possible to build a $N \times M$ rating matrix that is usually a sparse matrix as each user rates a small subset of the available items. The goal of a *collaborative filtering* system is to predict missing entries in this matrix using the known ratings.

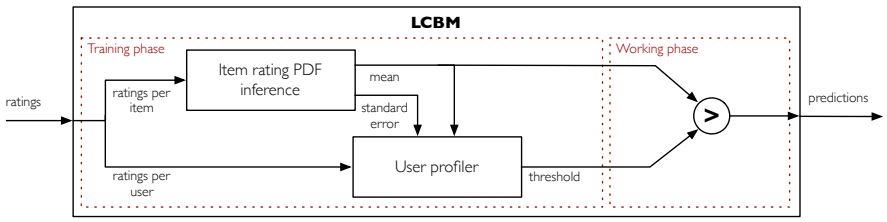


Fig. 1. LCBM: algorithm block diagram.

4 The LCBM algorithm

This section introduces the LCBM algorithm for collaborative filtering and analyzes its asymptotic behavior. First it describes the general structure of the algorithm and its internal functional blocks detailing their interactions; then the blocks are described in the following subsections.

4.1 Algorithm structure

Our solution departs from existing approaches to CF by considering items as elements whose tendency to be rated positively/negatively can be statistically characterized using an appropriate probability density function. Moreover, it also considers users as entities with different tastes that rate the same items using different criteria and that must thus be profiled. Information on items and users represents the basic knowledge needed to predict future user's ratings. LCBM is a two-stage algorithm constituted by a training phase, where the model is built, and a working phase, where the model is used to make predictions. Figure 1 shows a block diagram of LCBM that highlights its two-stage structure, its inputs and outputs, its main functional blocks and the interactions among them.

Training phase in this first phase collected ratings are fed to both an *Item rating PDF inference* block and a *User profiler* block. In the former case ratings are grouped by item and the block performs statistical operations on them to infer for each item the probability density function (PDF) of positive/negative rating ratios. Each inferred PDF is described by two measures: the *mean* and the *standard error*. In the latter case ratings are grouped by user and the block uses them to profile each user's rating behavior. It is important to note that in order to build accurate profiles this block is also fed with the data produced by the item's rating PDF inference block. The output of this block for each user is a single threshold value in the $[0, 1]$ range. The PDF mean values and the user thresholds represent the final output of this phase.

Working phase the second phase is in charge of producing the rating predictions. For each couple $(u, i), u \in U, i \in I$ such that the user u has not rated the item i a comparator is used to check the PDF mean value for that item against the user threshold and predict if that user will express a positive or negative rating.

It is important to notice that, while the flow of data between blocks in the algorithm architecture forces a sequential execution, operations performed within each block can be easily parallelized favoring a scalable implementation of the algorithm. Currently, we realized a prototype implementation of LCBM where the two blocks are implemented through multithreaded concurrent processes. More efficient and scalable implementations are part of our future work.

4.2 Item rating PDF inference

Items are profiled through a PDF of a single *random variable*, that represents the relative frequency of positive votes that the item will obtain in the future, given the observed ratings. We use the Beta distribution, a continuous family of probability functions indexed by two parameters α and β, to model this PDF. Given a number of received ratings, the unknown relative frequency of OKs an item will receive in the future has a probability distribution expressed by a Beta function with parameters α and β set to the number of OKs and KOs incremented by one respectively: $\alpha = OK + 1$ and $\beta = KO + 1$.

The profile of an item consists of two values: a measure of the beta distribution central tendency, the mean ($MEAN$), and a measure of the distribution variability, the standard error (SE):

$$MEAN = \frac{\alpha}{\alpha + \beta} = \frac{OK + 1}{OK + KO + 2} \tag{1}$$

$$SE = \frac{1}{OK + KO + 2}\sqrt{\frac{(OK + 1)(KO + 1)}{(OK + KO)(OK + KO + 3)}} \tag{2}$$

The $MEAN$ can be interpreted as the expected value for the relative frequency of OK votes that the item will obtain in the future. The SE is an estimate of the standard deviation of the $MEAN$. This value is important to indicate the reliability of an estimation. Intuitively, the more representative is the subset of voters, the lower the SE and the more accurate the $MEAN$ estimation.

Figure 2 shows the inferred PDF for an item that received so far 8 positive votes and 3 negatives. This curve expresses the probability that the item will receive a relative fraction of x positive ratings in the future. The mean of the distribution is 0.7. This can be interpreted as the expected value for x. For instance, the system expects that 7 of the next 10 ratings for the item will be positive. The standard error of the distribution is roughly 0.04. Using the Chebyshev's inequality [22], the SE can be interpreted as saying that the system expects that in the next 100 ratings for the item between 62 and 78 will be positive, with probability bigger than 0.75.

4.3 User Profiler

The goal of the *User profiler* is to identify a single value for each user using the votes that user expressed. We call this value the *quality threshold* (QT) of the user. Users profiling starts after the items profiling procedure. Therefore in this phase $MEAN$ and SE values are already defined for each item.

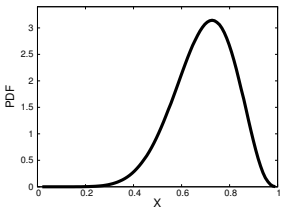


Fig. 2. Item profiling. Beta function after 8 OK and 3 KO.

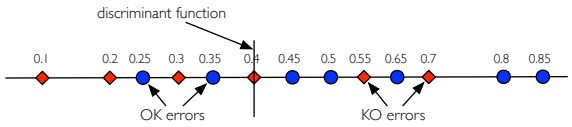


Fig. 3. User profiling. Linear classification in one dimension. OK represented by blue circles and KO by red diamonds. In this example $QT = 0.4$.

The algorithm uses a sorted data structure to collect all user's ratings. Without loss of generality, let this structure be a sorted set of points. Every rating is represented by a unique point p composed by two attributes: a key $p.key$, that gives the point's rank in the order, and a boolean value $p.value$ containing the rating. Each key lies on a $[0, 1]$ scale and its value is determined by the item's PDF. In particular, we adopt a *worst case estimation* approach: if the rating is positive (OK) the key is obtained by summing $2SE$ to the $MEAN$ of the item profile, if negative (KO) by subtracting $2SE$ from the $MEAN$.

A simple *linear classifier* is then used to find a *discriminant function* (i.e. a point p^*) for each user that separates the data with a minimal number of errors. We consider an error a point p_e with either $p_e.value = KO$ and $p.key > p^*.key$ or $p_e.value = OK$ and $p.key \leq p^*.key$.

Therefore, the discriminant point is chosen between those that minimize the following function:

$$L_{OK}(x) = \{p | p.value = OK \wedge p.key \leq x.key\} \tag{3}$$

$$R_{KO}(x) = \{p | p.value = KO \wedge p.key > x.key\} \tag{4}$$

$$f(x) = |L_{OK}(x)| + |R_{KO}(x)| \tag{5}$$

From all the points that minimize the function $f(x)$ the ones with the smaller absolute difference between $|L_{OK}(x)|$ and $|R_{KO}(x)|$ are selected, so that the errors are balanced between OKs and KOs. The user QT value is the smallest key in this set. The solution can be found in polynomial time. The simplest approach is to pass three times over the points: one to compute $|L_{OK}(x)|$ for each point; one to compute $|R_{KO}(x)|$ for each point; one to find the point that minimizes $f(x)$.

Figure 3 shows an example where an user expressed 13 votes, 7 OK (blue circles) and 6 KO (red diamonds). The user QT value is 0.4, key of the discriminant point p^*. In fact, no other choice will deliver less than 4 errors (perfectly balanced) in the classification task (two with smaller keys and OK values and two with bigger keys and KO values).

The *worst case estimation* approach prevents inaccurate item profiles from corrupting the classification task. Indeed, without this mechanism KO votes with over-estimate item $MEAN$ would lead to over-strict QTs (OK votes with under-estimate item $MEAN$ values would lead to over-permissive QTs respectively).

	LCBM	**SGD** [6]	**ALS** [5]
time to model	$O(X)$	$O(X \cdot K \cdot E)$	$\Omega(K^3(N+M) + K^2 \cdot X)$
time to prediction	$O(1)$	$O(K)$	$O(K)$
memory usage	$O(N+M)$	$O(K(N+M))$	$O(M^2 + X)$

Table 1. Algorithm cost compared with state-of-the-art solutions.

4.4 Working phase

In order to produce a prediction the algorithm simply compares QT values from the user profiler with $MEAN$ values from the item's rating PDF inference block. In particular, for each couple (u, i) without a rating it checks if the item $MEAN$ value is bigger than the user QT value. If this is the case the predicted user's rating for the item is OK, KO otherwise.

4.5 Algorithm analysis

Table 1 reports the cost of the LCBM algorithm compared with costs from other state-of-the-art solutions. In the table X is the number of collected ratings, K is the number of hidden features [23] and E is the number of iterations. We remark that $O(\cdot)$ is an upper bound, while $\Omega(\cdot)$ is a lower bound for the computational complexity.

If we consider the time needed to calculate the model, our solution performs two passes over the set of available ratings, one for each functional block in the training phase, thus its linear asymptotical cost. Note that this is the lowest asymptotical cost possible to build the model (as any solution should read each available rating at least once to build a model). Once the model is built it will be constituted by a value for each item (its $MEAN$) and a value for each user (its QT), thus the occupied memory will be $O(N + M)$. Finally, calculating the prediction for a single couple (u, i) requires a single comparison operation over two values, and thus incurs a constant cost.

5 Experimental Evaluation

In this section we report the results of the experimental evaluation we conducted on a prototype implementation of our solution. The goal of this evaluation was to assess how much our solution is effective in predicting ratings and the cost it incurs in doing so.

5.1 Experimental Setting and Test Datasets

We implemented[3] our LCBM algorithm, and evaluated it against open-source implementations of batch based CF algorithms provided by the *Apache Mahout* project (mahout.apache.org). We compared LCBM against both *memory-based* and *matrix factorization* solutions, however, this section only reports results from the latter as memory-based solutions have well-known scalability issues

[3] Our prototype is available at `http://www.dis.uniroma1.it/~midlab/LCBM/`

[9], and our LCBM algorithm outperformed them both in prediction accuracy and computational cost[4]. We limited our comparative evaluation to *matrix factorization* solutions, focusing on the two factorization techniques presented in Section 2: SGD and ALS. More precisely, we considered a lock-free and parallel implementation of the SGD factorizer based on [6] (the source code can be found in the *ParallelSGDFactorizer* class of the *Apache Mahout* library); the algorithm makes use of user and item biases for the prediction task. These two values indicate how much the ratings deviate from the average. This intuitively captures both users tendencies to give higher or lower ratings than others and items tendencies to receive higher or lower ratings than others. We also considered a parallel implementation of ALS with *Weighted-λ-Regularization* based on [5] (the source code can be found in the *ALSWRFactorizer* class of the *Apache Mahout* library).

If not differently specified, we set the following parameters for the above algorithms: *regularization factor* $\lambda = 0.065$ (as suggested in [5]), $K = 32$ hidden features and $E = 30$ iterations. We defined this last parameter by noting that 30 was the lowest number of iterations needed for the prediction accuracy score to converge on the considered datasets. Note that *Apache Mahout* allows you to define additional optional parameters for the two algorithms. In our experiments we used the default values for these variables, embedded in the corresponding source code. The algorithms return a real value (between -1 and 1) as a preference estimation for a couple (u, i). To discretize the prediction we adopted the most natural strategy: if the result is positive or zero the algorithm predicts an OK, if negative a KO.

We used three test datasets for our comparative study. The first two datasets were made available by the *GroupLens research lab* (grouplens.org) and consist of movie rating data collected through the *MovieLens* recommendation website (movielens.org). The third one is the *Netflix prize* dataset (`http://www.netflixprize.com`). All the ratings in these datasets were on a scale from 1 to 5, and we "binarized" them as follows [24]: if the rating for an item, by a user, is larger than the average rating by that user (average computed over his entire set of ratings) we assigned it a binary rating of 1 (OK), -1 (KO) otherwise.

The experiments were conducted on an *Intel Core i*7 $2,4GHz$ quad-core machine with $20GB$ of memory, using a GNU/Linux 64-bit operating system.

5.2 Evaluation methodology and Performance Metrics

Similar to most machine learning evaluation methodologies, we adopted a *k-fold cross-validation* approach. This technique divides the dataset in several folds and then uses in turn one of the folds as *test set* and the remaining ones as *training set*. The training set is used to build the model. The model is used to predict ratings that are then compared with those from the test set to compute the algorithm accuracy score. We randomly split the datasets in 5 folds, so that

[4] We also ran comparative tests with slope one algorithms. The results are not reported here as those algorithms showed worse performance than LCBM on all metrics.

each fold contained 20% of the ratings for each item. The reported results are the average of 5 independent runs, one for each possible fold chosen as test set.

In general, in order to evaluate the results of a binary CF algorithm we can identify four possible cases: either (i) *correct predictions*, both for OKs (TP true positives), and KOs (TN true negatives) or (ii) *wrong predictions*, both if OK is predicted for an observed KO (FP false positives) or if KO is predicted for an observed OK (FN false negatives). These four values constitute the so called *confusion matrix* of the classifier.
The *Matthews correlation coefficient* (**MCC**)[25] measures the quality of binary classifications. It returns a value between -1 and $+1$ where $+1$ represents a perfect prediction, 0 no better than random prediction and -1 indicates total disagreement between prediction and observation. The MCC can be calculated on the basis of the confusion matrix with the following formula:

$$MCC = \frac{TP \times TN - FP \times FN}{\sqrt{(TP + FP) \times (TP + FN) \times (TN + FP) \times (TN + FN)}} \tag{6}$$

To assess the load incurred by the system to run the algorithms we also calculated the *time* needed to run the test (from the starting point until all the possible predictions have been made) and the peak *memory* load during the test. It is important to remark that running times depend strongly on the specific implementation and platform, so they must be considered as relative indicators, whose final scope is to reflect the asymptotic costs already presented in Table 1.

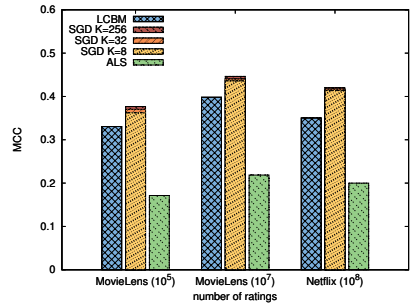
5.3 Evaluation Results

Figure 4 summarizes the performance of the CF algorithms over the three considered datasets, in terms of achieved prediction accuracy, time required for the prediction and memory occupation.

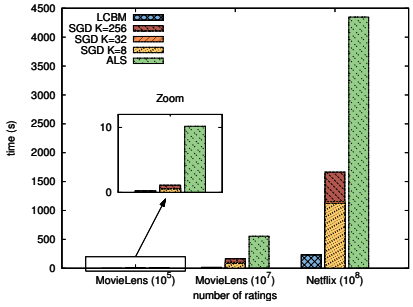
From Figure 4a it is possible to observe that LCBM consistently outperforms ALS by a large margin for all the considered datasets. Conversely, SGD outperforms LCBM in all datasets by a small margin whatever the value chosen for the number of features K is. By looking at this graph we can consider LCBM as a solution whose accuracy is very close to the accuracy offered by the best solution available in the state-of-the-art. However, the real advantages of LCBM come to light by looking at the load it imposes on the system.

Figure 4b shows the time required to conclude both the training and the test phases. Tests run with LCBM terminate much earlier that those run with SGD and ALS. This was an expected result as the time complexity of SGD is equivalent to the LCBM one only if we consider a single feature ($K = 1$) and a single iteration ($E = 1$) (cfr. Section 4.5). Note, however, that with this peculiar configuration SGD running time is still slightly larger than LCBM while its prediction accuracy, in terms of MCC, drops below the LCBM one (not shown in the graphs). The running time of ALS, as reported in the Figure, is always larger than LCBM.

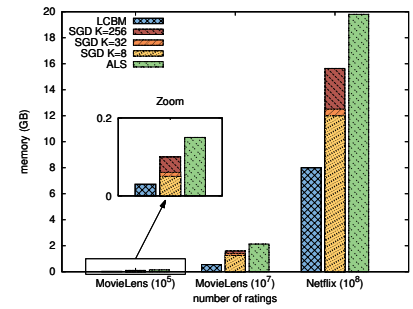
The peak memory occupation is reported in Figure 4c. Also in this plot the gap between LCBM and MF techniques is evident. To summarize, LCBM is



(a) MCC

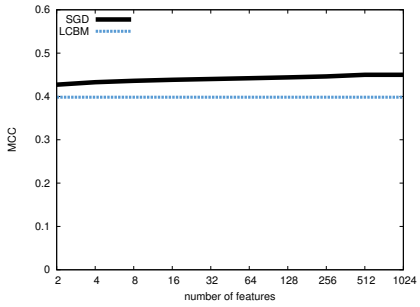


(b) Time

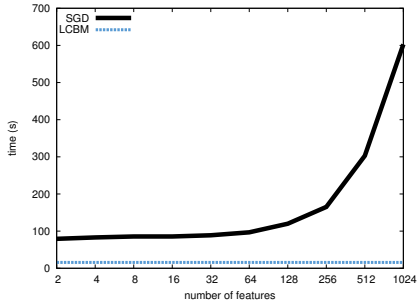


(c) Memory

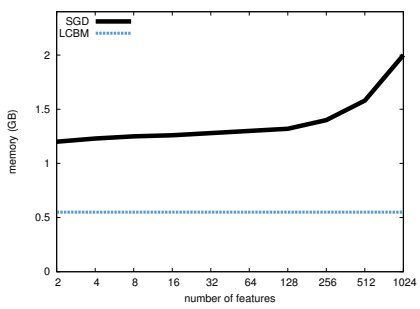
Fig. 4. Collaborative filtering algorithms performance, in terms of achieved accuracy, computational time required and memory occupation. The number of iterations for the matrix factorization models is set to 30. The SGD algorithm is trained with 8, 32 and 256 features. The number of features for the ALS algorithm is set to 32.



(a) MCC



(b) Time



(c) Memory

Fig. 5. LCBM vs. SGD performance varying the number K of hidden features, in terms of achieved accuracy, computational time required and memory occupation. The dataset used for the experiments is MovieLens with 10^7 ratings. The number of iterations was set to 30. The LCBM algorithm is agnostic to the number of features.

competitive with existing state-of-the-art MF solutions in terms of accuracy, but it runs faster while using less resources (in terms of memory).

The previous experiments have shown that the most performant matrix factorization solution is SGD. ALS, in facts, always showed the worst performance in our tests for all the considered metrics. Figure 5 reports the results of an experiment conducted on the MovieLens (10^7) dataset varying the number of hidden features K for the SGD factorizer. The LCBM performance are reported for comparison, and the corresponding curves are always constant because our solution is agnostic to K. Figures 5b and 5c show graphically what the asymptotic analysis has already revealed: time and space grow linearly with the number of features (note that the X-axis in the graphs has a logarithmic scale). The higher timeliness and memory usage thriftiness of LCBM is highlighted by the considerable gap between its curves and the SGD ones. Figure 5a reports the MCC values. As shown before SGD provides slightly better results than LCBM, and the gap tends to widen as the number of features grows. This, however, comes at the cost of a longer and more space consuming training procedure.

6 Conclusions

This paper introduced LCBM, a novel algorithm for collaborative filtering with binary ratings. LCBM works by analyzing collected ratings to (i) infer a probability density function of the relative frequency of positive votes that the item will receive and (ii) to compute a personalized quality threshold for each user. These two pieces of information are then used to predict missing ratings. Thanks to its internal modular nature LCBM is inherently parallelizable and can thus be adopted in demanding scenarios where large datasets must be analyzed. The paper presented a comparative analysis and experimental evaluation among LCBM and current solutions in the state-of-the-art that shows how LCBM is able to provide rating predictions whose accuracy is close to that offered by the best available solutions, but in a shorter time and using less resources (memory).

References

1. Mangalindan, J.: Amazon's recommendation secret. CNN Money `http://tech.fortune.cnn.com/2012/07/30/amazon-5/` (2012)
2. Krikorian, R.: New tweets per second record, and how! Twitter blog (`https://blog.twitter.com/2013/new-tweets-per-second-record-and-how`) (2013)
3. Halevy, A., Norvig, P., Pereira, F.: The unreasonable effectiveness of data. Intelligent Systems, IEEE **24**(2) (2009) 8–12
4. Narang, A., Gupta, R., Joshi, A., Garg, V.: Highly scalable parallel collaborative filtering algorithm. In: High Performance Computing (HiPC), 2010 International Conference on. (2010) 1–10
5. Zhou, Y., Wilkinson, D., Schreiber, R., Pan, R.: Large-scale parallel collaborative filtering for the netflix prize. In: Algorithmic Aspects in Information and Management. Springer (2008) 337–348
6. Takács, G., Pilászy, I., Németh, B., Tikk, D.: Scalable collaborative filtering approaches for large recommender systems. The Journal of Machine Learning Research **10** (2009) 623–656

7. Gemulla, R., Nijkamp, E., Haas, P.J., Sismanis, Y.: Large-scale matrix factorization with distributed stochastic gradient descent. In: Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining. (2011)
8. Zhuang, Y., Chin, W.S., Juan, Y.C., Lin, C.J.: A fast parallel sgd for matrix factorization in shared memory systems. In: Proceedings of the 7th ACM conference on Recommender systems, ACM (2013) 249–256
9. Su, X., Khoshgoftaar, T.M.: A survey of collaborative filtering techniques. Advances in Artificial Intelligence **2009** (2009) 4
10. Adomavicius, G., Tuzhilin, A.: Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. Knowledge and Data Engineering, IEEE Transactions on **17**(6) (2005) 734–749
11. Breese, J.S., Heckerman, D., Kadie, C.: Empirical analysis of predictive algorithms for collaborative filtering. In: Proceedings of the Fourteenth conference on Uncertainty in artificial intelligence, Morgan Kaufmann Publishers Inc. (1998) 43–52
12. Resnick, P., Iacovou, N., Suchak, M., Bergstrom, P., Riedl, J.: Grouplens: an open architecture for collaborative filtering of netnews. In: Proceedings of the 1994 ACM conference on Computer supported cooperative work, ACM (1994) 175–186
13. Linden, G., Smith, B., York, J.: Amazon. com recommendations: Item-to-item collaborative filtering. Internet Computing, IEEE **7**(1) (2003) 76–80
14. Lemire, D., Maclachlan, A.: Slope one predictors for online rating-based collaborative filtering. Society for Industrial Mathematics **5** (2005) 471–480
15. Hu, Y., Koren, Y., Volinsky, C.: Collaborative filtering for implicit feedback datasets. In: Data Mining, 2008. ICDM'08. Eighth IEEE International Conference on, IEEE (2008) 263–272
16. Koren, Y.: Factorization meets the neighborhood: a multifaceted collaborative filtering model. In: Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining, ACM (2008) 426–434
17. Teflioudi, C., Makari, F., Gemulla, R.: Distributed matrix completion. In: ICDM. (2012) 655–664
18. Pan, R., Zhou, Y., Cao, B., Liu, N., Lukose, R., Scholz, M., Yang, Q.: One-class collaborative filtering. In: Data Mining, 2008. ICDM '08. Eighth IEEE International Conference on. (2008) 502–511
19. Ungar, L., Foster, D.P.: A formal statistical approach to collaborative filtering. CONALD'98 (1998)
20. Wang, J., Robertson, S., de Vries, A.P., Reinders, M.J.: Probabilistic relevance ranking for collaborative filtering. Information Retrieval **11**(6) (2008) 477–497
21. Jsang, A., Ismail, R.: The beta reputation system. In: Proceedings of the 15th bled electronic commerce conference. (2002) 41–55
22. Huber, P.J.: The behavior of maximum likelihood estimates under nonstandard conditions. In: Proceedings of the fifth Berkeley symposium on mathematical statistics and probability. Volume 1. (1967) 221–233
23. Koren, Y., Bell, R., Volinsky, C.: Matrix factorization techniques for recommender systems. Computer **42**(8) (2009) 30–37
24. Das, A.S., Datar, M., Garg, A., Rajaram, S.: Google news personalization: scalable online collaborative filtering. In: Proceedings of the 16th international conference on World Wide Web, ACM (2007) 271–280
25. Matthews, B.W.: Comparison of the predicted and observed secondary structure of t4 phage lysozyme. Biochimica et Biophysica Acta (BBA)-Protein Structure **405**(2) (1975) 442–451