

LCBM: a fast and lightweight collaborative filtering algorithm for binary ratings[☆]

F. Petroni^{a,*}, L. Querzoni^{a,*}, R. Beraldi^a, M. Paolucci^b

^a*Department of Computer Control and Management Engineering Antonio Ruberti,
Sapienza University of Rome.*

^b*Institute of Cognitive Sciences and Technologies, CNR.*

Abstract

In the last ten years, *recommendation systems* evolved from novelties to powerful business tools, deeply changing the internet industry. Collaborative Filtering (CF) represents a widely adopted strategy today 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 (ii) provides results whose quality is on-par with state-of-the-art solutions (iii) at a fraction of the computational cost. These characteristics allow LCBM to efficiently handle large instances of the collaborative filtering problem on a single machine in short timeframes.

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

1. Introduction

Most of today's 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 companies. 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.

[☆]A preliminary version of this work appeared in the proceedings of the 17th International Conference on Business Information Systems.

*Corresponding author.

Email addresses: `petroni@dis.uniroma1.it` (F. Petroni), `querzoni@dis.uniroma1.it` (L. Querzoni), `beraldi@dis.uniroma1.it` (R. Beraldi), `mario.paolucci@istc.cnr.it` (M. Paolucci)

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].

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 fast and lightweight algorithm for collaborative filtering designed to work in systems with binary ratings. LCBM uses ratings collected on each item (i.e. products, news, tweets, movies, etc) to infer a probability density function shaped as a Beta distribution [9]; the Beta is a continuous family of probability functions normally used in the context of reputation systems; LCBM uses it to characterize 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. Differently from other solutions, LCBM builds a different profile for each users based on her previous voting history and on the other votes received by the items she voted. 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, in that the operations performed in the training procedure can be distributed and executed concurrently by multiple processors. Its performance has been extensively assessed through an experimental evaluation based on a large set of well-known public datasets (MovieLens 10M and 100K, Netflix, Tencent Weibo and Yahoo!) 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 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 [10, 11, 12, 13, 14].

It became popular in the late '90s with the spread of online services that use recommender systems, such as *Amazon.com*, *Yahoo!* and *Netflix*. The first work on the field of CF was the Tapestry system[15], developed at Xerox PARC, that used collaborative filtering to filter mails based on the opinion of other users, expressed with simple annotations (such as “useful survey” or “excellent”). Shortly after, the GroupLens system[16, 17] was developed, a pioneer application that gave users the opportunity to rate articles on a 1–5 scale and receive suggestions. CF solutions in the literature are often divided in two groups: *memory-based* and *model-based* [18, 10, 12].

Memory-based algorithms operate on the entire database of ratings to compute similarities between users or items. Such similarities constitute the “memory” of the collaborative filtering system, and are successively exploited to produce recommendations. Similar users or items are identified using a similarity metric, such as the Pearson correlation coefficient [16] and the cosine similarity [19, 20], that analyzes and compares the rating vectors of either users or items. The basic idea is to generate predictions by looking at the ratings of the most similar users or items; for this reason such techniques are called *neighborhood models*.

Neighborhood models are categorized as *user based* or *item based*. User based methods compute a similarity score between each pair of users, and then estimate unknown ratings based on recorded ratings of similar users [21, 22, 23, 24]. Item-oriented methods, instead, use the known ratings to compute similarities between items, and then provide recommendations by looking at similar items to those that an user has previously rated [25, 26, 27].

Memory-based methods are used in a lot of real-world systems because of their simple design and implementation. However, they impose several scalability limitations, since the computation of similarities between all pairs of users or items is expensive (i.e., quadratic time complexity with respect to the number of users or items), that makes their use impractical when dealing with large amounts of data. The slope one algorithms [28] were proposed to make faster prediction than memory-based algorithms, but they were unable to overcome the scalability issues of the latter.

LCBM differs from memory-based methods in that it does not require any similarity score to be computed for users or items. Our solution, in fact, only maintains a lightweight profile for each user and each item, and this allows LCBM to be much faster than neighborhood models, and to handle large-scale datasets in a reasonable amount of time with limited memory usage.

Model-based approaches have been investigated to overcome the shortcomings of memory-based algorithms. They use the collection of ratings to estimate or learn a model and then apply this model to make rating predictions. There have been several model-based CF approaches proposed in the literature, which use almost every existing machine learning technique. Noteworthy examples include cluster models and Bayesian networks[18], statistical model [29], linear regression [25],

MinHash [30]. In this multitude of algorithms, the most successful techniques are by far *latent factor* models. The goal of these approaches is to uncover latent features that explain observed ratings. For instance they can be thought of as representing user communities (like-minded users) or item communities (genres). Examples of latent factor algorithms include pLSA [31, 30], neural networks [32], and Latent Dirichlet Allocation [33].

The state-of-the-art on CF would not be complete without mentioning an event that impressively boosted the effort of the community in this field: the *Netflix prize* [34]. This open competition (2 October 2006 - 21 September 2009) had the aim to reward the CF algorithm that improved by 10% the Netflix one, with US\$1,000,000. The Netflix Prize definitively consecrated latent factor models, and a particular family of techniques proved to be superior to all other approaches: *matrix factorization* models.

Matrix factorization [35, 36, 37, 5, 6, 38, 39] have become a dominant methodology within collaborative filtering. It 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 approximation is performed by minimizing an application dependent error function $L(P, Q)$, that measures the quality of the reconstruction. Let be r_{ij} an observed entry in the global matrix of ratings R , p_i the i -th row of P and q_j the j -th column of Q . There exists a wide range of objective functions for matrix factorization. The most used error function is the regularized squared loss [35, 6, 40, 8, 41, 42, 39]:

$$L(P, Q) = \sum_{(i,j) \in P} (r_{ij} - p_i q_j)^2 + \lambda(\|P\|_F^2 + \|Q\|_F^2) \quad (1)$$

where $\|\cdot\|_F$ is the Frobenius norm and $\lambda \geq 0$ is a regularization coefficient used to avoid overfitting.

Concretely, a *matrix factorization* algorithm characterizes both items and users by vectors of factors inferred from item rating patterns. To make a prediction the system simply computes the dot product of the user vector p_i and the item vector q_j . The resulting value captures the interaction between the user and the item, and can be discretized to fit the actual scale of ratings. The most popular techniques to minimize the error function $L(P, Q)$ are *Alternating Least Squares (ALS)* and *Stochastic Gradient Descent (SGD)*. Both algorithms need several passes through the training set ratings to achieve convergence.

The *Alternating least square (ALS)* [5, 39] technique alternates between fixing P and Q . 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.

The *stochastic gradient descent (SGD)* [6, 8, 41, 39] technique 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. For each given training case r_{ij} , the system makes a prediction and computes

the associated error, as follows:

$$\varepsilon_{ij} = r_{ij} - p_i q_j \quad (2)$$

Then it modifies the item and user feature vectors by a magnitude proportional to μ (the *learning rate*) in the opposite direction of the gradient, as follows:

$$p_i \leftarrow p_i + \mu(\varepsilon_{ij} q_j - \lambda p_i) \quad (3)$$

$$q_j \leftarrow q_j + \mu(\varepsilon_{ij} p_i - \lambda q_j) \quad (4)$$

The λ parameter aims at preventing overfitting, and is called the *regularization factor*.

Some recent works aim at increasing the scalability of current MF solutions [7, 40, 8, 41, 39], 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.

LCBM differs from matrix factorization solutions in that it does not associate a latent factor vector with each user and each item, but just few values that constitute the profile of users and items. Moreover, the training procedure for our solution is extremely fast and light (it needs just one pass over the input data) while matrix factorization solutions requires a lengthy training phase, with several iterations over the input.

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* [43] 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 [44, 45] and reputation [9] adopted the same statistical method (i.e. Beta distribution) to combine feedback. Ungar and Foster [44] 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. [45] 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 $U = (u_1, \dots, u_N)$ users and $X = (x_1, \dots, x_M)$ items. Items represent a general abstraction that can be case by

symbol	description
U	users set
u_i	i-th user
N	number of users
X	items set
x_j	j-th item
M	number of items
R	rating matrix
r_{ij}	rating expressed by user u_i on item x_j

Table 1: Notation.

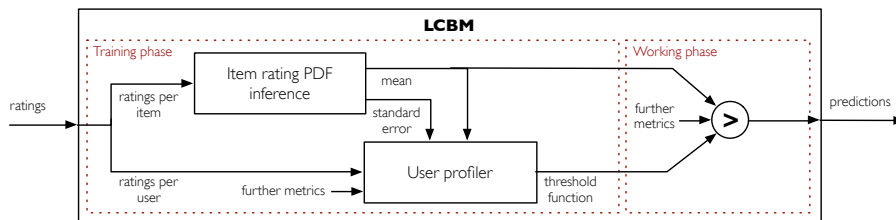


Figure 1: LCBM: algorithm block diagram.

case instantiated as news, tweets, shopping items, movies, songs, etc. Users can rate items with values from a predefined range. Rating values can be expressed in several different ways (depending on the specific system), however, in this paper we will consider only binary ratings.

By collecting user ratings it is possible to build a $N \times M$ rating matrix R that is usually a sparse matrix as each user rates a small subset of the available items. Each rating $r_{ij} \in R$ (expressed by user u_i on item x_j) is binary. Without loss of generality consider $r_{ij} \in \{1, -1\}$, where the two values can be considered as corresponding to *OK* (i.e., positive) and *KO* (i.e., negative) feedback respectively. The goal of a *collaborative filtering* system is to predict missing entries in this matrix (i.e., when $r_{ij} = ?$) using the known ratings.

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 and possible other metrics extracted from the input data. Such metrics are typically domain dependent; they could refer, for instance, to user and item attributes (e.g., movie release date), contextual information (e.g., rating timestamp) or other statistical indicators (e.g., item degree centrality in the rating graph, where vertices represent users and items and edges represent ratings). The output of this block for each user $u_i \in U$ is a threshold function $f_i(x)$, that takes in input an item $x_j \in I$ and outputs a prediction for the rating of user u_i for such item. The PDF mean values and the user threshold function represent the final output of this phase.

Working phase — The second phase is in charge of producing the rating predictions. For each couple $(i, j), u_i \in U, x_j \in X$ such that the user u_i has not rated the item x_j (i.e., $r_{ij} = ?$) the phase outputs the result of applying the threshold function on the given object, i.e. $f_i(x_j)$. This function, depending on its structure, may also take as input further metrics, as for the user profiler block (for instance, the rating timestamp).

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. In particular, different threads/machines can concurrently compute the profile associated with different users/items. LCBM can be easily implemented as a multithreaded application, for single-machine applications, or as a map-reduce process for large-scale applications. Section 4.5 provides an overview on the latter implementation.

4.2. Item rating PDF inference

Items are profiled by interpreting statistically the frequency of positive and negative votes. If there is no *a priori* evidence on the item, we consider positive and negative ratings equally probable. We assume that this process can be modeled as a random draw from a Bernoulli trial where the success probability p is unknown. When we start to collect votes, in the form of OKs and KOs, these affect the probability of subsequent ratings. What we need to know is the probability density for p , calculated *after* we see some extractions of OKs and KOs. This is known as the conjugate prior and is expressed by a Beta distribution [9] normally used in the context of reputation systems.

The Beta is a continuous family of probability functions on the interval $[0, 1]$, indexed by two parameters α and β . If both parameters are set to 1, the beta reduces to an uniform distribution, indicating complete ignorance. The profile of an item $x_j \in X$ is constituted by two values: $x_j.MEAN$ and $x_j.SE$. These two values are respectively the mean and the standard error of the Beta distribution obtained by setting α equal to the number of positive feedback item x_j received (plus one), and β equal to the number of negative feedback item x_j received (plus one). More formally:

$$\begin{aligned} x_j.OK &= |Y| : Y = \{r_{ij} \mid r_{ij} \in R \wedge r_{ij} = 1, \forall i \in [1, N]\} \\ x_j.KO &= |Y| : Y = \{r_{ij} \mid r_{ij} \in R \wedge r_{ij} = -1, \forall i \in [1, N]\} \\ \alpha &= x_j.OKs + 1 \\ \beta &= x_j.KOs + 1 \end{aligned}$$

The mean and standard error of the Beta distribution are as follow:

$$x_j.MEAN = \frac{\alpha}{\alpha + \beta} = \frac{x_j.OK + 1}{x_j.OK + x_j.KO + 2} \quad (5)$$

$$x_j.SE = \frac{1}{x_j.OK + x_j.KO + 2} \sqrt{\frac{(x_j.OK + 1)(x_j.KO + 1)}{(x_j.OK + x_j.KO)(x_j.OK + x_j.KO + 3)}} \quad (6)$$

The standard error $x_j.SE$ decreases with the number of observations (i.e., the number of ratings item x_j receives); thus, more observations bring a more precise estimate. Here we use the mean as the expected value for the relative frequency of positive ratings that item x_j will obtain in the future and is our main predictor. For a large number of observations, the mean approaches the intuitive value of $x_j.OK / (x_j.OK + x_j.KO)$. The corrections introduced by the Beta, however, produce better results for few observations, a significant improvement as early accuracy allows for a quick bootstrap phase. Intuitively, the more representative is the subset of voters, the lower the $Sx_j.E$ and the more accurate the $x_j.MEAN$ estimation.

As an example, Figure 2 shows the inferred PDF for an item that received so far 8 positive ratings and 3 negatives. This curve expresses the probability that the item will receive a relative fraction of μ positive ratings in the future. The mean of the distribution is approximately 0.7. This can be interpreted as the expected value for μ . For instance, we expect that 7 of the next 10 ratings for

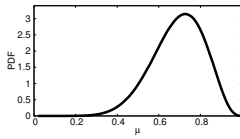


Figure 2: Item profiling. Beta function after 8 OKs and 3 KOs.

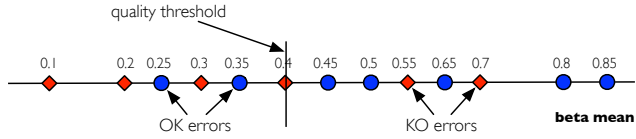


Figure 3: User profiling. Linear classification in one dimension. OK represented by blue circles and KO by red diamonds. In this example $QT = 0.4$.

the item will be positive. The standard error of the distribution is roughly 0.04, meaning that the true value should be between 0.62 and 0.78 with probability bigger than 0.75.

The idea of using the beta distribution to combine feedback has been used in the field since it was proposed in [9], with the goal of derive reputation scores. Collaborative filtering systems have similarities with reputation systems in that both collect ratings from members in a community [46]. However, there is a substantial difference between the two: reputation systems adopt a pessimistic world view, their final goal is to isolate untrusted parties from the community, and the ratings are therefore assumed insensitive to user taste; CF systems, instead, adopt an optimistic world view, where participants always report genuine opinion affected by their specific taste.

4.3. User Profiler

Goal of the *User profiler* is to identify for each user $u_i \in U$ a threshold function f_i by analyzing the votes he expressed. In general the function f_i takes an item with its characteristics as input and outputs either a positive (i.e. *OK*) or negative (i.e. *KO*) prediction. The output represents a classification of the item with respect to the user preferences.

User profiling, in general, may take as input both user characteristics inferred by its votes and item characteristics obtained from the item profiler. For these latter data to be available, user profiling starts after the item profiling procedure. We can therefore assume that $x_j.MEAN$ and $x_j.SE$ values are already defined for each item $x_j \in X$. These input data can be used to train a classifier. In this paper we consider two specific classifiers: a single-dimension and a two-dimensions linear classifiers.

4.3.1. Single dimension linear classifier

The single dimension linear classifier takes as input all user's ratings together with $x_j.MEAN$ and $x_j.SE$ values for all items the user rated. These values are fed to the classifier to identify a single *quality threshold* $u_i.QT$ for each user $u_i \in U$. Given the value of $u_i.QT$ for user u_i , the discriminant function f_i is defined as:

$$f_i(x_j) = \begin{cases} 1 \text{ (OK)}, & \text{if } x_j.MEAN \geq u_i.QT \\ -1 \text{ (KO)}, & \text{otherwise} \end{cases}$$

where x_j is an item and $x_j.MEAN$ is its mean value as calculated by the item rating PDF inference block.

User ratings are collected in a sorted data structure. Without loss of generality, let this structure be a sorted set of points. Every rating is represented by a unique point p with 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 $2 \cdot x.SE$ to the $x.MEAN$ of the item profile, if negative (*KO*) by subtracting $2 \cdot x.SE$ from the $x.MEAN$. More formally, to each rating $r_{ij} \in R$ user u_i expressed on item x_j we associate a point p with:

$$p.key = \begin{cases} x_j.MEAN + 2 \cdot x_j.SE, & \text{if } r_{ij} = 1 \\ x_j.MEAN - 2 \cdot x_j.SE, & \text{if } r_{ij} = -1 \end{cases}$$

$$p.value = r_{ij}$$

A simple *linear classifier* is then used to find the quality threshold $u_i.QT$ for user u_i 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 $p_e.key > u_i.QT$ or $p_e.value = OK$ and $p_e.key \leq u_i.QT$ (see Figure 3 for a graphical representation of such errors).

Concretely, the $u_i.QT$ is set to the key of a specific point p^* , that we call *discriminant point*. The discriminant point p^* is chosen by computing a score $E(p)$ for each point p , defined as follows:

$$E(p) = |L_{OK}(p)| + |R_{KO}(p)|$$

$$L_{OK}(p) = |Y| : Y = \{p_e | p_e.value = OK \wedge p_e.key \leq p.key\}$$

$$R_{KO}(p) = |Y| : Y = \{p_e | p_e.value = KO \wedge p_e.key > p.key\}$$

Intuitively, the score $E(p)$ reports the number of errors the algorithm makes by selecting p as a discriminant point. This latter is therefore chosen among the points with the minimum value for $E(p)$ (that could be more than one), in particular by selecting the point p^* with smallest key among the ones with the smallest absolute difference between $|L_{OK}(p)|$ and $|R_{KO}(p)|$ (i.e., the algorithm prefers balanced errors between left and right). The user quality threshold $u_i.QT$ is set to $p^*.key$:

$$u_i.QT = p^*.key$$

The solution can be found in polynomial time. The simplest approach is to pass three times over the points: one to compute $|L_{OK}(p)|$ for each point; one to compute $|R_{KO}(p)|$ for each point p ; one to find the discriminative point p^* .

The pseudocode for the single-dimension linear classifier is shown as Algorithm 1. The code takes as input the set of points L , where each point $p \in L$ is characterized by a key $p.key$ and a value $p.value$, and is made up of three

main steps. In the first step (lines 1-6) the algorithm scans in ascending order the set of points L and calculates for each $p \in L$ the value $p.leftOK$ (i.e. $p.leftOK = |L_{OK}(p)|$), representing the number of left errors the algorithm makes by picking that point as discriminative. The second step (lines 7-12) the same procedure is used to build for each point $p \in L$ the dual value $p.rightKO$ (i.e. $p.rightKO = |R_{KO}(p)|$). Finally, in the third step (lines 13-28) the value of QT is calculated following the procedure outlined above: the algorithm selects a discriminative point with (1) minimum value for $E(p)$, then (2) minimum difference between left and right errors, then (3) minimum key.

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. In fact, no other choice will deliver less than 4 errors (perfectly balanced between left and right) 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 QT s (OK votes with under-estimate item $MEAN$ values would lead to over-permissive QT s respectively).

4.3.2. Two-dimensions linear classifier

The two-dimension linear classifier extends the single-dimension one by collecting one further metric Y from the input data and uses this metric to improve the classifier output quality. This metric is, in most of the cases, domain dependent. It can represent, for instance, a movie release date or the rating timestamp. The metric Y is used to extend the sorted data structure described before with one further axis (hence its two-dimensional characteristic), thus placing user ratings on a plane where their x -axis coordinates are defined as before, while their y -axis coordinates are given by the corresponding Y 's value. Figure 4 shows an example of how votes expressed from a user may appear when depicted on this two-dimensional plane. The training phase consists in the identification of a discriminant function $t(x) = ax + b$ that cut the plane in two areas. Also in this case the cut should aim at minimizing the number of “misplaced” points, i.e. points with value OK lying in an area containing a majority of points with value KO (and vice-versa).

Linear support vector machine[47] provides a well-known solution to this problem; Section 5 reports the results of experiments run using LIBSVM as an algorithm to define $t(x)$. Given the definition of $t(x)$ the threshold function is defined as follows:

$$f_i(x_j, y) = \begin{cases} OK, & \text{if } y \cdot t(x_j.MEAN) \geq 0 \\ KO, & \text{otherwise} \end{cases}$$

where y is the value of metric Y , considered for the second axis (e.g., the rating timestamp or an attribute of the item). Its worth to be noticed that the

Algorithm 1 One-dimension linear classifier

Input: L : set of points, one for each rating**Output:** QT : user quality threshold

```
1:  $c \leftarrow 0$  ▷ Step 1 - compute left OK
2:  $L \leftarrow \text{SortAscending}(L)$ 
3: for all  $p \in L$  do
4:   if  $p.value$  then
5:      $c \leftarrow c + 1$ 
6:    $p.leftOK \leftarrow c$ 
7:  $c \leftarrow 0$  ▷ Step 2 - compute right KO
8:  $L \leftarrow \text{SortDescending}(L)$ 
9: for all  $p \in L$  do
10:   $p.rightKO \leftarrow c$ 
11:  if  $\neg p.value$  then
12:     $c \leftarrow c + 1$ 
13:  $E_{min} \leftarrow \infty$  ▷ Step 3 - search quality threshold
14:  $d_{min} \leftarrow \infty$ 
15: for all  $p \in L$  do
16:   $E(p) \leftarrow |p.leftOK + p.rightKO|$ 
17:  if  $E(p) < E_{min}$  then
18:     $E_{min} \leftarrow E(p)$ 
19:     $QT \leftarrow p.key$ 
20:  if  $E(p) = E_{min}$  then
21:     $d_p \leftarrow |p.leftOK - p.rightKO|$ 
22:    if  $d_p < d_{min}$  then
23:       $d_{min} \leftarrow d_p$ 
24:       $QT \leftarrow p.key$ 
25:    if  $d_p = d_{min}$  then
26:      if  $p.key < QT$  then
27:         $QT \leftarrow p.key$ 
28: return  $QT$ 
```

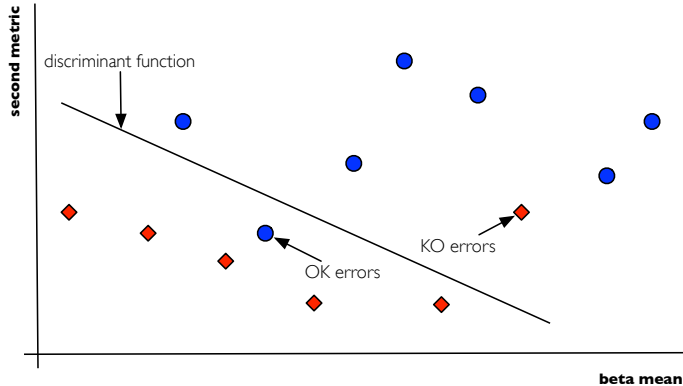


Figure 4: User profiling. Linear classification in two dimensions. OK represented by blue circles and KO by red diamonds.

previous definition is equivalent to the one provided for the single-dimension linear classifier if y is a positive constant value and $t(x) = x - QT_j$.

4.4. Algorithm analysis

Table 2 reports the cost of the LCBM algorithm compared with costs from other state-of-the-art solutions. In the table K is the number of hidden features [35] and E is the number of iterations for matrix factorization algorithms (i.e., SGD and ALS). We remark that $O(\cdot)$ is an upper bound, while $\Omega(\cdot)$ is a lower bound for the computational complexity; furthermore, to provide a fair comparison we considered as input the maximum possible number of ratings, i.e. $N \cdot M$; in real settings the number of ratings is usually a small fraction of this upper bound.

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; while the first block (item rating PDF inference) introduces a linear cost proportional to the number of available ratings, the second one must train a classifier for each user; its cost is therefore given by the number of users times the number of items times the cost for building a sorted list of points in the classifier algorithm (single-dimension linear classifier), i.e. $O(N \cdot M \cdot \log(M))$. The cost incurred with a two-dimension linear classifier would see the addition of the SVM cost. Once the model is built it will be constituted by a value for each item (its *MEAN*) and a threshold function for each user, thus the occupied memory will be $O(N + M)$. Finally, calculating the prediction for a single couple (u_i, x_j) requires a single comparison operation over two values, and thus incurs a constant cost.

4.5. MapReduce algorithm implementation

In this section we report the basic ideas behind the implementation of LCBM on MapReduce. The two-stage nature of LCBM suggests an implementation

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

Table 2: Algorithm cost compared with state-of-the-art solutions.

based on two iterations, one for each functional block (see Figure 1).

The first iteration implements the the item rating PDF inference block. The map function reads the input data and splits it in $\langle \text{key}, \text{value} \rangle$ pairs, where the key is the item id and the value is the boolean rating (OK or KO). The reduce function receives the pairs sorted by the key (item id), and computes the profile of each item by calculating the MEAN and the SE of the corresponding Beta function (Equations (5) and (6)). Then it stores on the distributed file system all the item profiles. This latter step is not expected to be excessively time-consuming as, in general, the number of items in a dataset is a small fraction of the number of users.

The second iteration implements the user profiler block. The map function reads again the input data and the item profiles stored in the previous iteration on the distributed file system, then it outputs a series of $\langle \text{key}, \text{value} \rangle$ pairs, one for each rating. The key this time is the user id while the value is a point of the classification space (single- or two-dimensional, depending on the specific classifier that we want to adopt). To define the key of the point the map function reads the profile of the rated item and operates as described in Section 4.3. The value of the point is the boolean rating (OK or KO). The Reduce function receives all the pairs (sorted by the user id), and, for each user, implements the linear classifier to find the quality threshold of the user. Then it stores on the distributed file system all the user profiles that can be later used to implement the working phase.

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 implemented¹ our LCBM algorithm, and evaluated it against open-source implementations of batch based CF algorithms provided by the *Apache*

¹Our prototype is available at <https://github.com/fabiopetroni/LCBM>

Dataset	MovieLens 100k	MovieLens 1M	Netflix	Tencent Weibo	Yahoo!
N	943	69878	480189	1392873	1000990
M	1682	10677	17770	4710	624961
$ratings$	100000	10000054	100480507	140046992	252800275

Table 3: Datasets description.

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 [11], and our LCBM algorithm outperformed them both in prediction accuracy and computational cost. We limited our comparative evaluation to *matrix factorization* solutions (currently considered the best approach to collaborative filtering [39]), 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 these last two parameters by noting that: (i) 30 was the lowest number of iterations needed for the prediction accuracy score to converge on the considered datasets and (ii) by increasing further the number of hidden features (i.e., for $K > 32$) the prediction accuracy score achieved by the algorithms didn't improve. 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, x_j) . 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 five 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 [48] (www.netflixprize.com). All the ratings in these datasets were on a scale from 1 to 5. The fourth dataset is The Yahoo! Music Dataset [49], used in the KDD-Cup 2011, consisting of ratings on musical items with scores between 0 to 100. In

order to “binarize” these four dataset we adopt the strategy proposed in [30]: 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 last dataset is a real binary dataset, used in the KDD-Cup 2012, consisting of a real trace from the Tencent Weibo social network [50]. In this dataset an item correspond to a user in the social network (person, organization, or group), and the rating scores represents the fact that the user accepts the recommendation of an item (*OK*), or rejects it (*KO*). Table 3 reports the number of users N , items M and ratings in the considered datasets.

The experiments were conducted on an *Intel Core i7 2,4GHz* quad-core machine with *32GB* of memory, using a GNU/Linux 64-bit operating system. The results of LCBM derive from the single dimension linear classifier, unless otherwise specified. All the considered algorithms (i.e., LCBM, SGD and ALS) execute the training procedure in a multithread fashion, using all the available cores. Moreover, we implemented a map-reduce version of LCBM². However, we used the parallel implementation of LCBM in our empirically study, since we noticed only a marginal improvement in execution time with the map-reduce implementation (in a 8 machines cluster). In fact, the LCBM algorithm can handle large instances of the collaborative filtering problem in very reasonable time on just a machine.

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 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**)[51] 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:

$$MCC = \frac{TP \times TN - FP \times FN}{\sqrt{(TP + FP) \times (TP + FN) \times (TN + FP) \times (TN + FN)}} \quad (7)$$

²Source code is available at <https://github.com/fabiopetroni/LCBM>

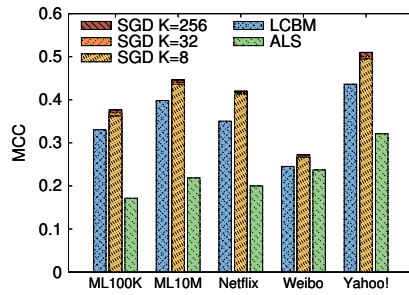
Starting from the confusion matrix we can further define the algorithm *Sensitivity* (or True Positive Rate (TPR)), i.e. the ratio between TP and $TP + FN$, that measures the proportion of actual OK ratings which are correctly identified as such. In the same way we can define the algorithm *Specificity* (or True Negative Rate (TNR)), i.e. the ratio between TN and $TN + FP$, that measures the proportion of KO ratings which are correctly identified as such. Their complementary values are represented by the False Positive Rate (FPR), i.e. $1 - TNR$, and False Negative Rate (FNR), i.e. $1 - TPR$. The *Receiver operating characteristic* (**ROC**) curve [52] visually illustrates the performance of a binary classifier by plotting different TPR (Y-axis) and FPR (X-axis) values. The diagonal line of this graph (the so-called *line of no-discrimination*) represents a completely random guess: classifiers represented by this line are no better than a random binary number generator. A perfect classifier would be represented by a point in the upper left corner of the ROC space (i.e. $FPR = 0$ and $TPR = 1$). Real classifiers are represented by curves lying in the space between the diagonal line and this ideal point.

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 2.

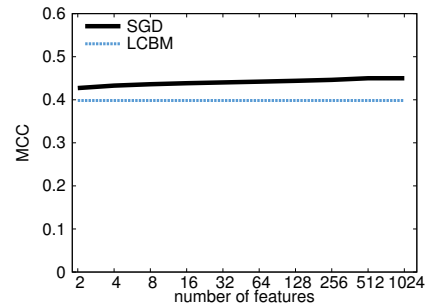
5.3. Overall performance of LCBMs

In this section we report the results obtained using a single dimension linear classifier in the training procedure. Figure 5 summarizes the performance of the CF algorithms over the considered datasets, in terms of achieved prediction accuracy, time required for the prediction and memory occupation. From Figure 5a 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.

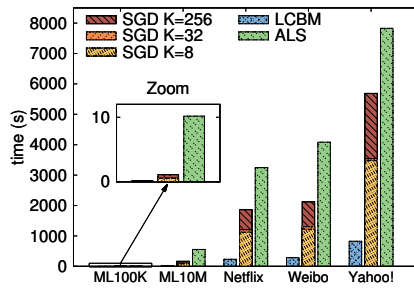
Figure 5b shows the time required to conclude both the training and the test phases. Tests run with LCBM terminate much earlier than those run with SGD and ALS. This was an expected result as the time complexity of SGD is equivalent to the LCBM one only if we consider a single feature ($K = 1$) and a single iteration ($E = 1$) (cfr. Section 4.4). 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.



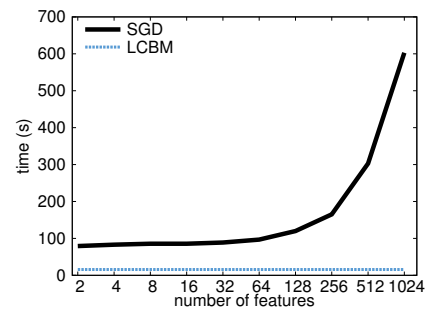
(a) MCC



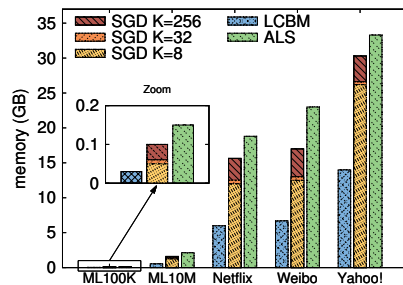
(a) MCC



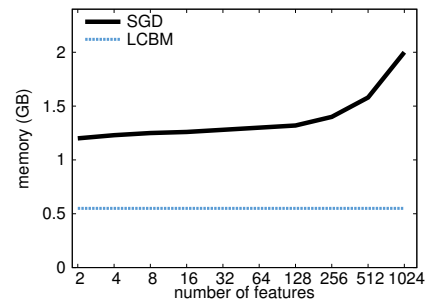
(b) Time



(b) Time



(c) Memory



(c) Memory

Figure 5: 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.

Figure 6: 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.

The peak memory occupation is reported in Figure 5c. Also in this plot the gap between LCBM and MF techniques is evident. To summarize, LCBM is competitive with existing state-of-the-art MF solutions in terms of accuracy,

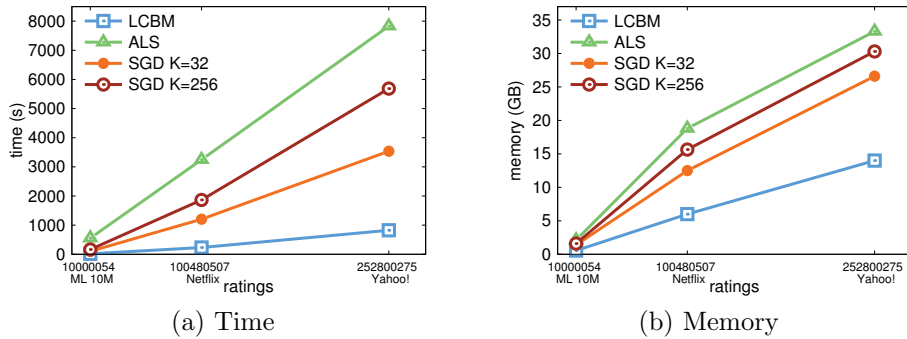


Figure 7: Algorithms performance, in terms of time and memory occupation, by increasing the dataset size.

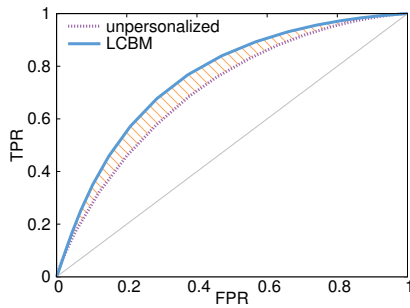


Figure 8: ROC curves comparing a *fixed* approach (i.e., same constant QT value for all users) with the LCBM solution on the Netflix dataset. The plot shows the gain in using a personalized approach with respect to a collective one.

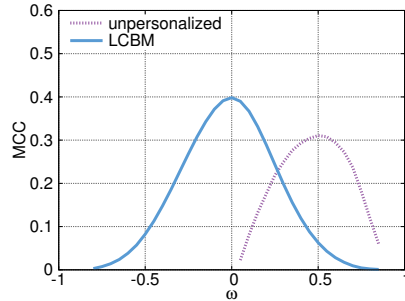


Figure 9: Algorithm accuracy (MCC) obtained by varying a constant ω value to the QT (LCBM curve) in comparison with a *fixed* approach (i.e., same constant QT value for all users) on the Netflix dataset.

yet 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 6 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 (this parameter is peculiar of matrix factorization solutions). Figures 6b and 6c 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 lower CPU time and memory usage of LCBM is highlighted by the considerable gap between its curves and the SGD ones. Figure 6a 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.

Figure 7 shows the trend of the time needed to build the model (Figure 7a) and the memory required to store it (Figure 7b) by increasing the dataset size. LCBM is, consistently with the previous analysis, the most efficient technique, both in terms of timelessness and memory usage. Moreover, the Figures highlight the growth of the gap between LCBM and the two MF solutions by increasing the number of ratings. In other words, the benefits of LCBM, in time and memory, grow proportionally with the size of the input, making our solution very appealing when the system has to deal with huge amount of data.

5.4. Benefit of personalization

The graph in Figure 8 reports the ROC curve for LCBM on the Netflix dataset. Actually, the output of the LCBM algorithm represents a single (FPR, TPR) point. In order to obtain a curve for LCBM (blue line in Figure 8), we performed several run of the algorithm, each time shifting the quality threshold of all the users (i.e., the output of the user profile block) by a constant value ω , that we varied from run to run. In particular, we considered 100 values from ω , uniformly distributed in the range $[-1, 1]$, and we executed a single LCBM run for each considered value. We obtained the ROC curve by connecting the corresponding 100 (FPR, TPR) points.

The goal of this experiment was to compare our LCBM algorithm, that associates a personalized quality threshold with all the users, with an unpersonalized solution, that simply uses a constant value as quality threshold for all the users (i.e., all users have the same QT value, they are assumed to share the same tastes). To obtain this unpersonalized curve (dashed purple line in Figure 8) we considered again 100 values from ω , uniformly distributed in the range $[-1, 1]$, and for each value we executed a modified version of the LCBM algorithm where all the QT values of the users were set to the ω value. We obtained the ROC curve by connecting the corresponding 100 (FPR, TPR) points.

Clearly, the latter and simpler solution, offers lower performance with respect to LCBM. However, the interesting point is that the area between the two curves (dashed in the graph) clearly shows the added value given by the independent profiling of users performed by LCBM: calculating and using a personalized QT value for each user pays back in terms of prediction accuracy. Furthermore, Figure 9 shows how LCBM performs at its best (in terms of MCC) when the constant ω value added to the QT is set to 0, thus confirming that the mechanism used by the user profiler to calculate QT is appropriately designed.

5.5. Two-dimensions linear classifier

Finally, figure 10 shows the improvements obtainable by adopting a two-dimensions linear classifier that uses different metrics for its second dimension. In particular, we tested this classifier on the MovieLens 100k dataset and fed the classifier with three different information: the movie degree centrality, or the movie release date and the rating timestamp. The rationale behind these

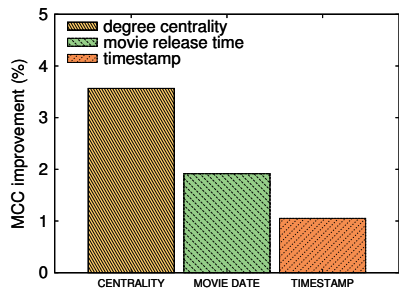


Figure 10: MCC improvement using a two-dimension linear classifier. Results obtained on the MovieLens 100k dataset considering three different metrics.

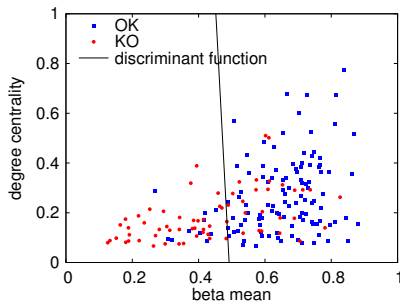


Figure 11: Example of a two-dimension classifier internal state for a specific user using the item degree centrality as the second metric (MovieLens 100k dataset).

choices depend obviously on the specific metric we consider. The movie degree centrality was considered an important characteristic of the items to be voted as it can deeply influence the perspective voter behavior: extremely popular movies tend to polarize the audience opinions (most of the people express the same vote, be it positive or negative), while “niche” movie ratings strongly depend on the user peculiar tastes. The movie release date was considered as a way to distinguish user groups: there are people that are mostly interested in recent movies, while other people are more used to watch “classics” and tend to prefer older movies. Finally, the vote timestamp was considered as a variable able to track the changing user preferences over time. In this latter case the classifier was trained to identify the changing preferences of the user as time passes by. The graph shows that, depending on the specific metric we consider, the two-dimensions linear classifier provides a 1% (rating timestamp) to 3.5% (movie degree centrality) improvement over the single-dimension classifier. The marginal improvement tells us that (i) the quality threshold identified for each user by the single dimension linear classifier is a good estimation of how the user tend to rank items and that (ii) trying to further enhance this classification by looking at how the quality threshold varies with respect to a second metric shows not provide significant advantages. Figure 11 shows a scatter plot representing the internal state of the two-dimension classifier for a specific user using the item degree centrality as the Y axis; the line in the middle represents the output of the classifier: the fact that this line is only slightly deviating from being perfectly vertical confirms that the degree centrality metric does not add significant information to the classifier whose output, as a result, is very close to the one obtainable from a single-dimension classifier. Similar results were observed with other metrics (results omitted from this document). What radically changes by adopting the two-dimension variant of our solution is the time needed to compute the results. In our test we observed a steady 100% increase in computation time, independently from the metric considered for the second

dimension. In general, we can conclude that the improvements provided by the two-dimension linear classifier with respect to the single-dimension one are not worth the extra cost.

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 profile each user with a personalized threshold function. 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] J. Mangalindan, Amazon's recommendation secret, CNN Money <http://tech.fortune.cnn.com/2012/07/30/amazon-5/> (2012).
- [2] R. Krikorian, New tweets per second record, and how!, Twitter blog (<https://blog.twitter.com/2013/new-tweets-per-second-record-and-how>) (2013).
- [3] A. Halevy, P. Norvig, F. Pereira, The unreasonable effectiveness of data, *Intelligent Systems*, IEEE 24 (2) (2009) 8–12.
- [4] A. Narang, R. Gupta, A. Joshi, V. Garg, Highly scalable parallel collaborative filtering algorithm, in: *High Performance Computing (HiPC)*, 2010 International Conference on, 2010, pp. 1–10. doi:10.1109/HIPC.2010.5713175.
- [5] Y. Zhou, D. Wilkinson, R. Schreiber, R. Pan, Large-scale parallel collaborative filtering for the netflix prize, in: *Algorithmic Aspects in Information and Management*, Springer, 2008, pp. 337–348.
- [6] G. Takács, I. Pilászy, B. Németh, D. Tikk, Scalable collaborative filtering approaches for large recommender systems, *The Journal of Machine Learning Research* 10 (2009) 623–656.
- [7] R. Gemulla, E. Nijkamp, P. J. Haas, Y. Sismanis, 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] Y. Zhuang, W.-S. Chin, Y.-C. Juan, C.-J. Lin, A fast parallel sgd for matrix factorization in shared memory systems, in: Proceedings of the 7th ACM conference on Recommender systems, ACM, 2013, pp. 249–256.
- [9] A. Jsang, R. Ismail, The beta reputation system, in: Proceedings of the 15th bled electronic commerce conference, 2002, pp. 41–55.
- [10] G. Adomavicius, A. Tuzhilin, 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] X. Su, T. M. Khoshgoftaar, A survey of collaborative filtering techniques, Advances in Artificial Intelligence (2009) 4.
- [12] M. D. Ekstrand, J. T. Riedl, J. A. Konstan, Collaborative filtering recommender systems, Foundations and Trends in Human-Computer Interaction 4 (2) (2011) 81–173.
- [13] F. Ricci, L. Rokach, B. Shapira, Introduction to recommender systems handbook, Springer, 2011.
- [14] J. A. Konstan, J. Riedl, Recommender systems: from algorithms to user experience, User Modeling and User-Adapted Interaction 22 (1-2) (2012) 101–123.
- [15] D. Goldberg, D. Nichols, B. M. Oki, D. Terry, Using collaborative filtering to weave an information tapestry, Communications of the ACM 35 (12) (1992) 61–70.
- [16] P. Resnick, N. Iacovou, M. Suchak, P. Bergstrom, J. Riedl, Grouplens: an open architecture for collaborative filtering of netnews, in: Proceedings of the 1994 ACM conference on Computer supported cooperative work, ACM, 1994, pp. 175–186.
- [17] J. A. Konstan, B. N. Miller, D. Maltz, J. L. Herlocker, L. R. Gordon, J. Riedl, Grouplens: applying collaborative filtering to usenet news, Communications of the ACM 40 (3) (1997) 77–87.
- [18] J. S. Breese, D. Heckerman, 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., 1998, pp. 43–52.
- [19] R. Baeza-Yates, B. Ribeiro-Neto, et al., Modern information retrieval, Vol. 463, ACM press New York, 1999.
- [20] A. Singhal, Modern information retrieval: A brief overview, IEEE Data Eng. Bull. 24 (4) (2001) 35–43.

- [21] J. L. Herlocker, J. A. Konstan, A. Borchers, J. Riedl, An algorithmic framework for performing collaborative filtering, in: Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval, ACM, 1999, pp. 230–237.
- [22] J. Zhang, P. Pu, A recursive prediction algorithm for collaborative filtering recommender systems, in: Proceedings of the 2007 ACM conference on Recommender systems, ACM, 2007, pp. 57–64.
- [23] Y. Shi, M. Larson, A. Hanjalic, Exploiting user similarity based on rated-item pools for improved user-based collaborative filtering, in: Proceedings of the third ACM conference on Recommender systems, ACM, 2009, pp. 125–132.
- [24] R. Xu, S. Wang, X. Zheng, Y. Chen, Distributed collaborative filtering with singular ratings for large scale recommendation, *Journal of Systems and Software* 95 (2014) 231–241.
- [25] B. Sarwar, G. Karypis, J. Konstan, J. Riedl, Item-based collaborative filtering recommendation algorithms, in: Proceedings of the 10th international conference on World Wide Web, ACM, 2001, pp. 285–295.
- [26] G. Linden, B. Smith, J. York, Amazon. com recommendations: Item-to-item collaborative filtering, *Internet Computing, IEEE* 7 (1) (2003) 76–80.
- [27] M. Deshpande, G. Karypis, Item-based top-n recommendation algorithms, *ACM Transactions on Information Systems (TOIS)* 22 (1) (2004) 143–177.
- [28] D. Lemire, A. Maclachlan, Slope one predictors for online rating-based collaborative filtering, *Society for Industrial Mathematics* 5 (2005) 471–480.
- [29] L. H. Ungar, D. P. Foster, Clustering methods for collaborative filtering, in: *AAAI Workshop on Recommendation Systems*, no. 1, 1998.
- [30] A. S. Das, M. Datar, A. Garg, S. Rajaram, Google news personalization: scalable online collaborative filtering, in: Proceedings of the 16th international conference on World Wide Web, ACM, 2007, pp. 271–280.
- [31] T. Hofmann, Latent semantic models for collaborative filtering, *ACM Transactions on Information Systems (TOIS)* 22 (1) (2004) 89–115.
- [32] R. Salakhutdinov, A. Mnih, G. Hinton, Restricted boltzmann machines for collaborative filtering, in: Proceedings of the 24th international conference on Machine learning, ACM, 2007, pp. 791–798.
- [33] D. M. Blei, A. Y. Ng, M. I. Jordan, Latent dirichlet allocation, *the Journal of machine Learning research* 3 (2003) 993–1022.
- [34] The netflix prize <http://www.netflixprize.com>.

- [35] Y. Koren, R. Bell, C. Volinsky, Matrix factorization techniques for recommender systems, *Computer* 42 (8) (2009) 30–37.
- [36] Y. Hu, Y. Koren, C. Volinsky, Collaborative filtering for implicit feedback datasets, in: *Data Mining, 2008. ICDM'08. Eighth IEEE International Conference on*, IEEE, 2008, pp. 263–272.
- [37] Y. Koren, 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, pp. 426–434.
- [38] X. Luo, M. Zhou, Y. Xia, Q. Zhu, An efficient non-negative matrix-factorization-based approach to collaborative filtering for recommender systems, *Industrial Informatics, IEEE Transactions on* 10 (2) (2014) 1273–1284.
- [39] F. Makari, C. Teflioudi, R. Gemulla, P. Haas, Y. Sismanis, Shared-memory and shared-nothing stochastic gradient descent algorithms for matrix completion, *Knowledge and Information Systems* 42 (3) (2014) 493–523.
- [40] C. Teflioudi, F. Makari, R. Gemulla, Distributed matrix completion., in: *ICDM, 2012*, pp. 655–664.
- [41] F. Petroni, L. Querzoni, Gasgd: stochastic gradient descent for distributed asynchronous matrix completion via graph partitioning., in: *Proceedings of the 8th ACM Conference on Recommender systems*, ACM, 2014, pp. 241–248.
- [42] S. Kabbur, G. Karypis, Nlmf: Nonlinear matrix factorization methods for top-n recommender systems, in: *Data Mining Workshop (ICDMW), 2014 IEEE International Conference on*, IEEE, 2014, pp. 167–174.
- [43] R. Pan, Y. Zhou, B. Cao, N. Liu, R. Lukose, M. Scholz, Q. Yang, One-class collaborative filtering, in: *Data Mining, 2008. ICDM '08. Eighth IEEE International Conference on*, 2008, pp. 502–511. doi:10.1109/ICDM.2008.16.
- [44] L. Ungar, D. P. Foster, A formal statistical approach to collaborative filtering, *CONALD'98*.
- [45] J. Wang, S. Robertson, A. P. de Vries, M. J. Reinders, Probabilistic relevance ranking for collaborative filtering, *Information Retrieval* 11 (6) (2008) 477–497.
- [46] A. Jøsang, R. Ismail, C. Boyd, A survey of trust and reputation systems for online service provision, *Decision support systems* 43 (2) (2007) 618–644.

- [47] C.-C. Chang, C.-J. Lin, Libsvm: A library for support vector machines, *ACM Trans. Intell. Syst. Technol.* 2 (3) (2011) 27:1–27:27. doi:10.1145/1961189.1961199.
URL <http://doi.acm.org/10.1145/1961189.1961199>
- [48] J. Bennett, S. Lanning, The netflix prize, in: *Proceedings of KDD cup and workshop*, 2007.
- [49] G. Dror, N. Koenigstein, Y. Koren, M. Weimer, The yahoo! music dataset and kdd-cup'11., *Journal of Machine Learning Research-Proceedings Track* 18 (2012) 8–18.
- [50] Y. Niu, Y. Wang, G. Sun, A. Yue, B. Dalessandro, C. Perlich, B. Hamner, The tencent dataset and kdd-cup'12, in: *KDD-Cup Workshop*, Vol. 170, 2012.
- [51] B. W. Matthews, Comparison of the predicted and observed secondary structure of t4 phage lysozyme, *Biochimica et Biophysica Acta (BBA)-Protein Structure* 405 (2) (1975) 442–451.
- [52] J. L. Herlocker, J. A. Konstan, L. G. Terveen, J. T. Riedl, Evaluating collaborative filtering recommender systems, *ACM Transactions on Information Systems (TOIS)* 22 (1) (2004) 5–53.