RAIRO-Oper. Res. 59 (2025) 3891–3911
https://doi.org/10.1051/ro/2025151

RAIRO Operations Research
www.rairo-ro.org

ADDRESSING THE COLD START PROBLEM IN PRIVACY PRESERVING CONTENT-BASED RECOMMENDER SYSTEMS USING HYPERCUBE GRAPHS

Noa Tuval[1], Alain Hertz[2,*] and Tsvi Kuflik[1]

Abstract. The initial interaction of a user with a recommender system is problematic because, in such a so-called cold start situation, the recommender system has very little information about the user, if any. Moreover, in collaborative filtering, users need to share their preferences with the service provider by rating items while in content-based filtering there is no need for such information sharing. A content-based model using hypercube graphs has recently been proposed and appears to be able to estimate user profiles based on a very limited number of ratings while preserving user privacy. In this paper, we confirm these findings on the basis of experiments with more than 1000 users in the restaurant and movie domains. We show that the proposed method outperforms standard machine learning algorithms when the number of available ratings is at most 10, which often happens, and is competitive with larger training sets. In addition, training is simple and doesn't require large computational efforts.

Mathematics Subject Classification. 68T05, 68Q32 .

Received February 4, 2025. Accepted November 8, 2025.

1. Introduction

The growing involvement of recommender systems in our daily life raises the need to develop reasoning mechanisms to recommend relevant items to users based on their past interactions with the system as well as additional external sources that may be available. In order to personalize the recommendations, these systems create a model for each user on the basis of information previously collected [3,24,40]. The two prevailing classical user modeling methods are *collaborative filtering* (CF) and *content-based filtering* (CB). Recommender systems based on CF build such models based on the similarities of preferences between users: the items recommended for a user are those that received a high score from a group of users who show similar ratings. Conversely, items recommended to a user by CB systems have attributes similar to those of items that have received a high rating from the user in the past.

CF systems need a lot of data to infer the preferences of their users and have difficulty in producing good recommendations when there are very few users compared to the number of items. Moreover, CF systems suffer from *the new item problem*: new products with no rating history are unlikely to be recommended [40, 54]. This problem doesn't exist in CB systems because recommending an item depends solely on its attributes. On the

Keywords. Recommender systems, cold start problem, hypercube graphs.

[1] Information Systems Department, The University of Haifa, Haifa, Israel.

[2] Department of Mathematics and Industrial Engineering, Polytechnique Montréal – Gerad, Montréal, Canada.
*Corresponding author: alain.hertz@gerad.ca

other hand, CB methods have problems that do not exist in CF systems: describing the items using a limited number of attributes makes it difficult to represent all aspects of the items, which may impair the accuracy of the recommendations. In addition, CB systems have the problem of over-specialization since they only recommend items with similar attributes to items the user has liked in the past [3].

With the huge amount of data flowing in the network, awareness of user privacy is increasing, due to the sensitivity and vulnerability of user data [22]. Recommender systems require users to reveal their preferences, in order for the recommendations produced by the system to match their tastes [3]. However, most users want the system to maintain their privacy [17]. Since CF methods rely on data provided by a group of users to generate recommendations for a particular user, this approach requires many users to be identified and share their preferences with the system in the form of ratings [3, 17, 25]. Conversely, CB methods need to know the characteristics of each item and user preferences about these characteristics to recommend to users items that match their tastes [3, 8]. These preferences can be stored locally on the user's personal device and not shared [26]. In this aspect, CB systems preserve the privacy of users better than CF systems.

An improvement to the classic recommendation methods are the *hybrid* methods [11, 40] and *graph-based* methods [3, 40, 49]. Hybrid recommender systems combine different methods, where usually the main method used is CF in combination with another strategy, while exploiting the advantages of the participating methods to overcome some difficulties such as the new item problem and data sparsity [11, 40]. One of the challenges facing hybrid systems is the need for data from different sources in order to create good personalized recommendations. However, in addition to the additional space required, the use of large amounts of data also requires more intensive calculations.

Graph-based methods provide versatile structures for representing the relationships among users and items [3, 31, 43]. In recent years, *graph learning* (GL) approaches have been developed for graph-based user modeling. Unlike the CB and CF methods, GL techniques can extract knowledge from various graph representations even where the entities are implicitly connected [40, 49]. A major challenge of graph-based recommender systems is that the data is usually very large and requires a lot of time and complex algorithms to be processed [49].

1.1. The cold start problem

It's natural to question what happens when a recommender system doesn't have enough information to define an adequate model for each user. This situation occurs when a user is accessing the system for the first time or when their interaction with it is only superficial, limited to evaluating a very small number of items. In such a case, the system hasn't collected enough information to understand the user preferences and therefore cannot create a reliable model of the user. This problem, which prevents new users from taking full advantage of the power and relevance of recommender systems, is known as the *cold start problem*. It prevents CF methods from producing reliable recommendations since there is no (or too little) prior information about the user of the system which therefore cannot determine users with similar tastes to produce recommendations [45]. Moreover, since CF systems derive user preferences based on previous ratings, they must have enough ratings to create reliable user models [2, 32].

As stated in [42, 53], the cold start problem is one of the research challenges of recommender systems. The problem has caught the attention of many researchers [7, 18, 35, 51] because the ability of a recommender system to produce relevant recommendations often comes up against the long-tail distribution of the number of ratings per user. As an example, we show in Figure 1 the number of ratings that the users of the yelp website (`https://www.yelp.com/dataset`) have given. It is clear that the vast majority of users of the system have provided very few evaluations. For example, there are 1 069 244 users with less than 10 ratings, but only 13 351 users with at least 20 ratings.

1.2. Objectives and practical implications

A new approach to create user models with CB systems is described in [21]. Roughly speaking, the users of the system and the items that can be recommended to them are vertices of a hypercube Q_n of dimension n

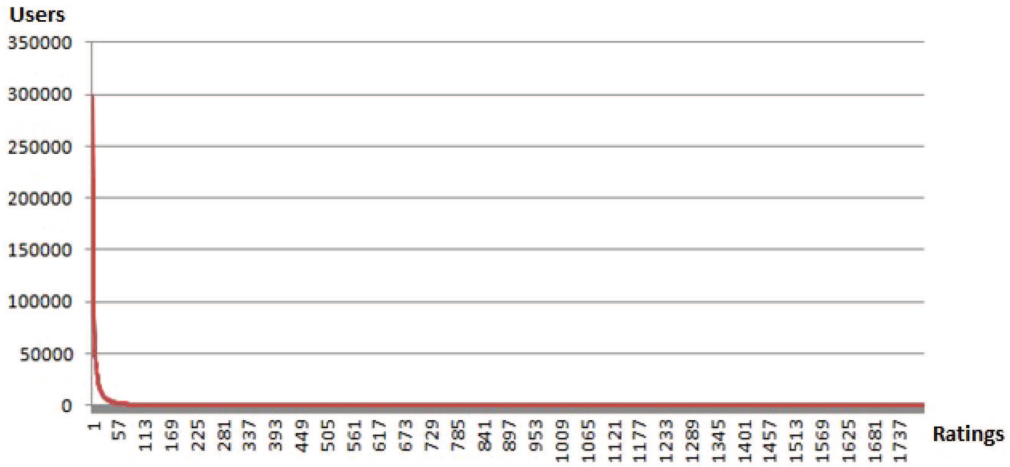

FIGURE 1. The long tail distribution of the number of ratings per user (taken from the yelp website https://www.yelp.com/dataset).

(see Fig. 2 in Sect. 3 for an example with $n = 3$), where n is the number of attributes of the items. Given a set R of items evaluated by a user, the estimated user model is the vertex u of the hypercube such that the distances between u and the vertices of R correspond as closely as possible to the scores assigned by the user to these items. Preliminary results reported in [21] show that this estimate is generally very accurate when using it to predict user preferences for items. In this paper, we confirm these observations on the basis of larger scale experiments.

Our main objective is to demonstrate that the content-based recommender system based on hypercube graphs described in [21] is an interesting approach to solve the cold start problem. We will show that it outperforms standard machine learning algorithms when the number of available ratings is at most 10. Also, an advantage of this approch over CF methods is that it allows preserving the confidential data of users because the information necessary to identify the preferences of each user can be kept locally on a laptop or a smartphone.

In summary, we show that we can learn user preferences with very few ratings. The comparison between the results obtained by our method with those produced by current standard techniques confirms the ability of our approach to solve the cold start problem while preserving user privacy.

The paper is structured as follows: A brief literature review related to user modeling, to the cold start problem and to graph-based models for recommender systems is given in the next section. Our approach based on hypercube graphs is described in Section 3, and computational experiments are reported in Section 4 where we compare our algorithms with five commonly used machine learning techniques. Concluding remarks and future work are given in Section 5.

2. Background and related work

We start this section with an overview of models and techniques used in recommender systems, and standard techniques used to build such models. We then show how graph theory is used in such systems, and we conclude the section with a review of recent studies for solving the cold start problem in recommender systems.

2.1. User models and machine learning techniques

Efficient user modeling is a major challenge in recommender systems that aim at providing personalized service to their users. These systems leverage information collected about users from various sources such as

web pages, social networks, and e-commerce sites. The information collected can then be structured in different ways, depending on the chosen recommendation technique [40].

Recommender systems usually make use of either or both collaborative filtering (CF) and content-based filtering (CB) approaches, as well as other techniques such as community-based methods and knowledge-based methods [40]. CF approaches recommend to the user items that users with similar taste like. These methods often use latent factor models like matrix factorization methods to improve data representation by transforming users and items to the same latent factor space [3,25]. Other methods used by CF are various machine learning techniques which build a summarized model of the data and predict items that the user may have an interest in [37,52]. CB approaches utilize user preferences over a series of item attributes to recommend additional items with similar properties [32,40]. CF and CB recommender systems need a sufficient amount of previous user ratings to generate accurate recommendations. However, these systems are not suitable for highly customized products which are not purchased on a regular basis, such as houses, cars or even smart electronic products [3]. Moreover, these systems tend to recommend items that are most similar to the ones they users have already selected [3].

Current recommender systems may also combine one or more approaches into a hybrid system which tries to exploit the benefits of each of the combined methods [8,40]. One of the hybrid techniques is the cross-representation mediation of user models from CF to CB recommender systems [5,6]. This mediation technique is effective in cases where the lack of data in a CB recommender system makes it difficult to generate recommendations. Using the mediation process, the CB system can leverage the data of the same users that has been collected by a CF system and transform these data into weighted item attributes.

In recent years, modeling user behavior using classical machine learning techniques has become a challenge with the large amount of information available for recommender systems and the complexity of user interactions with the system [53]. Thus, deep learning techniques have been increasingly used in recommender systems as they enable processing unstructured data and inferring hidden patterns of user preferences and item representation [32,53]. Also, deep learning techniques are able to consider descriptive information (such as text, images, audio, and video) about users and items available from various sources, to create a more reliable and accurate user model [53]. Deep learning has been successfully applied in fields as diverse as movie, music, image or quote recommendation [16,34,52].

As the architecture of a neural network has a decisive influence on the learning model performance, improving the construction of the neural system, for example by adding hidden layers to the neural network, may help the multitask learning. Also, implementing recommendation algorithms through deep neural networks involves high computational cost that should be reduced using more efficient algorithms [13,53]. The big amount of data required for the learning process is another disadvantage of deep learning techniques, since it is not possible to produce accurate recommendations when there is not enough data available [53].

2.2. Graph-based user modeling

Graphs and network are increasingly applied for user modeling. The main advantage of these models is that they allow combining entities and their mutual links in simple structures, without loss of information [43]. At the same time, graph learning methods, such as random walk methods, graph neural networks methods and graph embedding methods, capable of learning complex relations between entities and extracting knowledge from graphs, were developed [49].

In most graph-based approaches, user and item entities are represented by vertices, and an edge connects a user u with an item i when u has rated i, which leads to a bipartite graph [53]. An edge connecting a user to an item he has evaluated can be weighted with the value of the corresponding score, and the amount of information that can propagate along such an edge is then proportional to this weight [33]. Graph models are used to detect similarities between users or items, these similarities being measured using random walks [23] or path-based approaches [33].

The bipartite graph described above can be extended by adding additional attributes as new graph vertices and linking them to the existing vertices [49]. The resulting graph is called *heterogeneous* and enables integrating

collaborative ratings with domain knowledge and content aspects for better personalized recommendations [31,49]. User modeling with heterogeneous graphs is used, for example, in [50] for movie recommendation and in [31] for on-site guidance of museum visitors.

Graph Neural Networks (GNN) is a deep learning approach designed to help understanding all graph components and hidden data, which is essential for recommender systems. GNN techniques have proved their potential to learn from graph structured data, for example for session-based and social recommendation [15,38]. However, a major challenge of this model is the low computation efficiency due to the big amount of data required for the learning process [53].

A new approach for efficient analysis of graph structured data is graph embedding which converts graph data into low dimension vectors while preserving the graph structure and connectivity [10]. For example, an edge embedding technique is proposed in [44] for friend recommendation and in [12] for time-aware smart object recommendation.

To conclude this section, let us mention that graphs are also used to produce explainable recommendations, which is an aspect of recommender systems that has received recent attention [29].

2.3. The cold start problem in recommender systems

A common way to learn about user preferences and overcome the cold start problem is to ask users to rate a few items [39]. Based on this limited information, the system should create initial user models. It is therefore desirable that the items offered for evaluation can help the system to understand user preferences. Several techniques have been proposed for determining the most informative items to be rated by a user [39]. Recommending popular items is one of the solutions in cases where there is not enough prior information about user preferences. The obvious disadvantage of such a solution is that the user gets a recommendation for a product that has a high chance that he is already familiar with [3].

Sometimes users may wish to explore new items offered by a recommender system. In this case, they will cooperate with the system and willingly rate the suggested items. However,users may not have the time or desire to rate a large number of items. They want to get recommendations without being too involved in the process. Requiring users to rate items as a precondition for creating recommendations for them can be considered intrusive. It would therefore be interesting to be able to measure the benefit of getting additional ratings in terms of the increased accuracy of predictions based on these ratings [2].

Another way proposed to address the cold start problem is to use hybrid recommender systems that combine different types of recommendation techniques in order to overcome the weakness of one component through the use of another component of the system. A hybrid approach is described in [5] that uses a modeling mediation process which integrates partial user models produced by different types of recommender systems. When a new user enters a CB recommender system that has no information about his preferences, this cold start situation is overcome through a mediation process that leverages previous user data collected in a CF system in the same domain. The user model produced by the CF method, which is represented by a vector of ratings, is converted in the mediation process into a vector of weighted attributes which is then used by the CB system to produce recommendations [5].

A cross-domain recommender system is proposed in [47] where data from an online shopping domain is combined with information taken from an advertising platform. A neural CF model is trained on users who appear in both domains, allowing the system to produce recommendations to users who are active on the advertisement platform but not in the online shopping domain.

A deep learning CF approach that is suitable for both warm start and cold start is proposed in [46]. This method considers the cold start problem as a problem of missing data, so that the training of the model is done *via* input dropout, *i.e.*, some nodes along with their connections are dropped from the neural network during training to estimate how each rating contributes to improving the accuracy of user models.

Few-shot learning (FSL) is a machine learning method that aims to classify new data when only a few training samples are available. It has been used to solve the cold start problem in recommender systems [19,48]. Zero-shot learning (ZSL) is a variant of FSL that tries to deal with the situation where a learner observes samples from

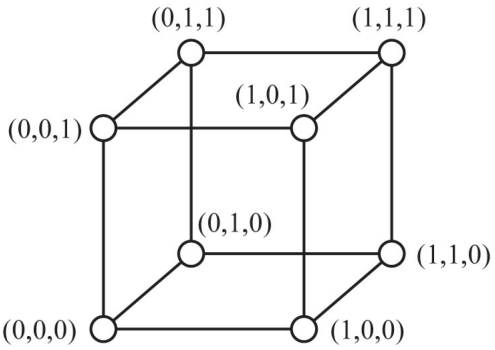

FIGURE 2. The hypercube Q_3.

classes which were not observed during training, and needs to predict the class that they belong to. A method for solving the cold start problem from a ZSL perspective is proposed in [28]. Also, collaborative filtering algorithms based on meta-learning, such as MAML (model-agnostic meta-learning) help the recommender system to initialize new users, while overcoming the bias that may be caused by global sharing initialization parameters common to all users [14, 27]. All the above techniques are based on learning many user profiles based on their personal data. For comparison, the method studied in this article handles the cold start issue without requiring users' personal data.

In the next section, we describe a technique we proposed in [21] for building user models through hypercube graphs in a CB recommender system. It differs greatly from approaches based on deep learning and graph embedding.

3. Hypercube graphs for CB recommender systems

In this section we briefly provide a theoretical background about using hypercube graphs for CB representation of items and users, and we then describe two user modeling techniques proposed in [21] based on hypercube graph representations in CB recommender systems.

Let U be a set of users of a recommender system, let A be an ordered set of n Boolean attributes, and let I be a set of items. Let Q_n be the n-dimensional hypercube with vertex set $\{0,1\}^n$, and where two vertices $\boldsymbol{x}$ and $\boldsymbol{y}$ are linked with an edge if and only if the Hamming distance $d(\boldsymbol{x},\boldsymbol{y})$ between $\boldsymbol{x}$ and $\boldsymbol{y}$ (*i.e.*, the number of indices $i \in \{1,\dots,n\}$ such that $x_i \neq y_i$) equals 1. Items and user models are represented as vertices in Q_n (see Fig. 2). More precisely:

- A vertex $\boldsymbol{v}^i = (v_1^i,\dots,v_n^i)$ of Q_n is associated with every item $i \in I$ so that $v_j^i = 1$ if i has the jth attribute in A, and $v_j^i = 0$ otherwise;
- A vertex $\boldsymbol{w}^u = (w_1^u,\dots,w_n^u)$ of Q_n is associated with every user $u \in U$ so that $w_j^u = 1$ if and only if u "likes" the jth attribute in A.

Note that two items with the same attributes and two users with the same preferences are associated with the same vertex in Q_n. We can therefore consider every vertex of Q_n as an *item type* and a *user type*.

Assume that a user $u \in U$ of a recommender system has indicated the number of attributes $a \in A$ he doesn't like in every items of a subset $I' \subseteq I$ of rated items. This means that we know the Hamming distance between $\boldsymbol{w}^u$ and all vertices $\boldsymbol{v}^i$ with $i \in I'$. To identify the vertex $\boldsymbol{w}^u$ which corresponds to the model of u, it is therefore sufficient to determine a vertex $\boldsymbol{x}$ of the hypercube such that, for all $i \in I'$, the Hamming distance $d(\boldsymbol{x},\boldsymbol{v}^i)$ is equal to the number of attributes that user u doesn't like in item i.

The problem we are facing with this approach is that users never indicate this precise information (*i.e.*, the Hamming distance) which would allow us to fully understand their tastes. Rather, they give a score to the items

they rate, and we therefore have to convert this score into a Hamming distance in Q_n. We suppose here that the users rate items according to an s-star scale, where the highest score of s stars is given by users to items that perfectly match their preferences, and the lowest score of 1 star when they did not like any of the attribute values of the rated item. So let r_i be the rating given by u to an item $i \in I$, using an s-star scale. This rating can be translated into a distance δ_i, called d-rating using function $\tau : \{1, \ldots, s\} \longrightarrow [0, n]$ which is defined as follows:

$$\delta_i = \tau(r_i) = n - \frac{n(r_i - 1)}{s - 1}.$$

The worst rating r_i with 1 star is thus translated into distance n, and the best rating with $r_i = s$ stars into distance 0. The d-rating can be considered as an approximation of the Hamming distance. Indeed, if a user $u \in U$ likes all attributes of an item $i \in I$, his rating will be $r_i = 5$ stars, which we translate into $\delta_i = 0$, with the meaning that our estimate of the Hamming distance $d(\boldsymbol{w}^u, \boldsymbol{v}^i)$ is $\delta_i = 0$. On the contrary, if u does not like any of the attributes of an item $i \in I$, his rating will be $r_i = 1$ star, which we translate into $\delta_i = n$, with the meaning that our estimate of the Hamming distance $d(\boldsymbol{w}^u, \boldsymbol{v}^i)$ is $\delta_i = n$.

The models described in [21] for recommending items to users of the system are based on this concept. They aim to determine a vertex $\boldsymbol{x}$ that fits the ratings of a user $u \in U$. More precicely, assume that u has rated a subset $I' \subseteq I$ of items, let r_i be the rating assigned to $i \in I'$ by u, and let δ_i be the associated d-ratings. The task is to determine the vector $\boldsymbol{w}^u$ associated with u in Q_n. Hence, the output $\boldsymbol{x}$ of the algorithms proposed in [21] should ideally be equal to $\boldsymbol{w}^u$. If $\boldsymbol{x} = \boldsymbol{w}^u$ then, for all $i \in I'$, the Hamming distance $d(\boldsymbol{v}^i, \boldsymbol{x})$ should be the number of attributes $a \in A$ such that $w_a^u \neq v_a^i$. Since we estimate $d(\boldsymbol{v}^i, \boldsymbol{w}^u)$ as being equal to δ_i, the cumulative error made by assuming that $\boldsymbol{x} = \boldsymbol{w}^u$ is estimated as $f_1(\boldsymbol{x})$, where

$$f_1(\boldsymbol{x}) = \sum_{i \in I'} \left| d\left(\boldsymbol{v}^i, \boldsymbol{x}\right) - \delta_i \right|.$$

In summary, the first algorithm in [21], called Algo1, generates a vector $\boldsymbol{x}$ in Q_n with minimum value $f_1(\boldsymbol{x})$. It can be described as follows.

Algo1

Input : a set I' of item rated by a user u, as well as the ratings r_i assigned to $i \in I'$ by u.
Output : a vector $\boldsymbol{x}$ in Q_n.

1. Build the vectors $\boldsymbol{v}^i$ of Q_n associated with each $i \in I'$;
2. Compute the d-ratings $\delta_i = \tau(r_i)$ for each $i \in I'$;
3. Determine the optimal solution of the following problem:
   minimize $f_1(\boldsymbol{x}) = \sum_{i \in I'} |d(\boldsymbol{v}^i, \boldsymbol{x}) - \delta_i|$
   subject to $\boldsymbol{x} \in Q_n$.

Variations of this algorithm are proposed in [21]. We consider here the one where "don't care" values are allowed. In other words, it is assumed that it may happen that a user doesn't care about some item attributes. A user $u \in U$ is then represented by a vector $\boldsymbol{w}^u$ in $\{-1, 0, 1\}^n$ (instead of $\{0, 1\}^n$), so that $w_j^u = -1$ if u doesn't like the jth attribute, $w_j^u = 0$ if u doesn't care about it, and $w_j^u = 1$ if u likes it. If $w_j^u = 0$, the ratings of u do not depend on the value of the jth attribute. To take this into account, the Hamming distance d is replaced by a new distance $d' : \{0, 1\}^n \times \{-1, 0, 1\}^n \to \{0, \ldots, n\}$ which, given a vertex $\boldsymbol{v} \in \{0, 1\}^n$ and a vertex $\boldsymbol{x} \in \{-1, 0, 1\}^n$, counts the number of components j with $v_j = 1$ and $x_j = -1$, or $v_j = 0$ and $x_j = 1$. The task of the second algorithm that we will test in the next section is to determine a vertex $\boldsymbol{x}$ in $\{-1, 0, 1\}^n$ that minimizes $f_2(\boldsymbol{x})$, where f_2 is obtained from f_1 by replacing d with d'. This second algorithm, called Algo2, can be described as follows.

Algo2

Input : a set I' of item rated by a user u, as well as the ratings r_i assigned to $i \in I'$ by u.
Output : a vector $\boldsymbol{x}$ in $\{-1, 0, 1\}^n$.

TABLE 1. Description of the data sets used in our first experiments.

		Number of rated items					Average	Standard
u_i	n_i	1⋆	2⋆	3⋆	4⋆	5⋆	rating	deviation
u_1	49	5	68	237	169	21	3.27	0.78
u_2	50	12	60	246	177	5	3.21	0.75
u_3	50	7	4	278	158	9	3.23	0.70
u_4	48	9	50	141	229	71	3.61	0.91
u_5	51	7	43	125	166	159	3.85	1.01
u_6	48	4	9	41	312	134	4.13	0.69
u_7	50	17	40	116	189	138	3.78	1.04
u_8	50	2	15	195	214	74	3.69	0.77
u_9	49	4	21	115	279	81	3.82	0.78
u_{10}	45	4	47	153	230	66	3.61	0.86

1. Build the vectors $\boldsymbol{v}^i$ of Q_n associated with each $i \in I'$;
2. Compute the d-ratings $\delta_i = \tau(r_i)$ for each $i \in I'$;
3. Determine the optimal solution of the following problem:
   minimize $f_2(\boldsymbol{x}) = \sum_{i \in I'} |d'(\boldsymbol{v}^i, \boldsymbol{x}) - \delta_i|$
   subject to $\boldsymbol{x} \in \{-1, 0, 1\}^n$.

As shown in [21], determining $\boldsymbol{x}$ that minimizes $f_1(\boldsymbol{x})$ or $f_2(\boldsymbol{x})$ is a problem that can be formulated as an integer program. Preliminary experiments are reported in [21] which show that few ratings are sufficient to generate a vertex $\boldsymbol{x}$ which is close to the user model. The aim of the next section is to confirm these findings on the basis of larger scale experiments.

4. Computational experiments

In this section, we report results obtained by executing the two algorithms mentioned in Section 3. We first focus on datasets from the restaurant domain which we used in [21] and compare the results produced by the two algorithms with those obtained using classical machine learning techniques widely used in recommender systems. We will then focus on datasets taken from the movie domain.

4.1. Experiments with restaurant datasets

In a previous study [21] we extracted ten datasets from the yelp website, each containing 500 restaurants that were rated by a single user on a 5-star scale. The number n of attributes varies from 45 to 51, referring to the restaurant genre (Italian, Chinese etc.), the price (low, medium, high, very high), and some other facilities (reservations, outdoor seating etc.). A restaurant in the dataset corresponding to user u_i, is characterized by n_i attributes from the full list of attributes, where an attribute is included in the dataset only if it exists in at least one of the restaurants rated by the user. For example, since no restaurant that user u_7 has rated serves Greek/Turkish food, the attribute corresponding to this type of food doesn't appear in u_7's dataset and therefore $n_7 = 50$ (but not 51). More details on these 10 datasets are shown in Table 1. We indicate the number of attributes, the number of rated items with a score of 1, 2, 3, 4, and 5 stars, the average rating, and the standard deviation.

Note that user ratings may seem inconsistent. For example, it is not rare that a user gives two different ratings to two items having the same attribute values. This can be due to a missing attribute in the system or to human inconsistency as user preferences may be impacted by various contextual aspects or simply changed with time [41]. The considered datasets are no exception. For example, three restaurants rated by u_4 have the

TABLE 2. Average prediction errors (in "stars") produced by each method for training sets with 450 items and test sets with 50 items.

u_i	Algo1	Algo2	NB	RF	SVR	DT	NN
u_1	0.554	0.567	0.550	0.563	0.496	0.678	0.648
u_2	0.542	0.548	0.552	0.541	0.500	0.682	0.605
u_3	0.482	0.476	0.418	0.440	0.380	0.588	0.484
u_4	0.674	0.678	0.612	0.665	0.604	0.740	0.705
u_5	0.778	0.782	0.876	0.757	0.710	0.888	0.851
u_6	0.407	0.410	0.406	0.451	0.388	0.612	0.493
u_7	0.770	0.768	0.772	0.767	0.716	0.940	0.936
u_8	0.614	0.608	0.580	0.533	0.570	0.622	0.653
u_9	0.500	0.500	0.498	0.525	0.470	0.616	0.646
u_{10}	0.644	0.615	0.584	0.580	0.508	0.684	0.665
Average	0.5964	0.5953	0.5848	0.5821	0.5342	0.7050	0.6686

same attribute values while the user has assigned a score of 3, 4 and 5 stars, respectively. Table 1 shows that most of the user ratings are of 3 and 4 stars, which is a known phenomenon [30].

For each user, we performed a 10-fold cross-validation process with 450 restaurants as a training set I' and the remaining 50 items of $I \setminus I'$ as a test set. As mentioned in Section 3, the algorithm that minimizes f_1 is called Algo1, while Algo2 is for the minimization of f_2. These two algorithms are called Model1 and Model3 in [21], respectively. As mentioned in Section 3, the output of Algo1 is a vector $\boldsymbol{x} \in \{0,1\}^n$. We have to explain how this output is used to predict the ratings given by the user to the 50 restaurants in the test set.

While function τ translates an s-star rating in $\{1,\ldots,s\}$ into a d-rating in $[0,n]$, we consider the following inverse function $\tau^{-1} : [0,n] \to \{1,\ldots,s\}$:

$$\tau^{-1}(\delta) = s - \left\lfloor \frac{\delta(s-1)}{n} \right\rceil,$$

where $\lfloor \rceil$ if the nearest integer function. For example, for $n = 20$ and $s = 5$, a d-rating $\delta = 7$ is transformed into an 5-star rating $\tau^{-1}(7) = 5 - \left\lfloor \frac{28}{20} \right\rceil = 4$.

Consider an item $i \in I \setminus I'$. The Hamming distance $d(\boldsymbol{v}^i, \boldsymbol{x})$ between the vertex $\boldsymbol{v}^i$ of the hypercube representing the item i and the vertex $\boldsymbol{x}$ representing the user model is our guess of the number of attribute values in $\boldsymbol{v}^i$ that do not match the user preferences. We therefore transform this distance into an s-star rating $\tau^{-1}(d(\boldsymbol{v}^i, \boldsymbol{x}))$ and compare it with the actual rating r_i. The average error $F_1(\boldsymbol{x})$ induced by $\boldsymbol{x}$ when predicting the ratings given by the user to the restaurants in $I \setminus I'$ is therefore defined as follows:

$$F_1(\boldsymbol{x}) = \frac{1}{|I| - |I'|} \sum_{i \in I \setminus I'} |\tau^{-1}(d(\boldsymbol{v}^i, \boldsymbol{x})) - \rho_i|.$$

For Algo2, the average prediction error is calculated in exactly the same way, except that distance d' is used instead of the Hamming distance d. The average prediction errors of Algo1 and Algo2 are reported in Table 2. We have run our algorithms on a 3 GHz Intel Xeon X5675 machine with 8 GB of RAM, and all integer programs were solved using CPLEX (v12.2), with a time limit of 1 s. When CPLEX was interrupted before completing the optimization, we report the best solution found. Experiments have shown that no noticeable improvement is obtained with longer computing times, of the order of a minute or an hour.

Since the datasets are very small and concern only a very limited number of users, deep learning techniques cannot be used. Hence, for comparing the above results with other techniques, we have run five machine learning methods widely used in recommender systems, namely Nave Bayes (NB), Random Forest (RF), Support Vector

TABLE 3. Results of the statistical tests: + indicates a statistically significant difference between the methods, while − indicates that the differences between the two methods are not statistically significant.

	SVR	NB	RF	Algo2	Algo1	NN
NB	+					
RF	+	−				
Algo2	+	−	−			
Algo1	+	−	−	−		
NN	+	+	+	+	+	
DT	+	+	+	+	+	+

Regression (SVR), Decision Tree (DT) and Neural Network (NN), using the Scikit machine learning tool [36]. The hyper parameters for these machine learning methods were chosen according to the recommendations given by the scikit-learn website (https://scikit-learn.org/). We also did some experiments to make sure we were getting the best results using the chosen hyper parameters.

As for Algo1 and Algo2, we performed a 10-fold cross validation, using 450 items as a training set and 50 items as a test set. The results are also given in Table 2. We observe that SVR is the best method with the smallest average prediction error of 0.5342, while the DT is the worst with an error of 0.7050. The errors for Algo1 and Algo2 are 0.5964 and 0.5953, respectively, and NB and RF have slightly better results.

To evaluate whether there is a statistically significant difference between the performance of two methods, we used the standard f-tests and t-tests, with significance level 0.05. If the average prediction error of a method m_1 is statistically significantly smaller than that of a method m_2, we say, for short, that m_1 is significantly better than m_2, and that m_2 is significantly worse than m_1. We also say that the error produced by m_1 is significantly smaller than that of m_2, and the one produced by m_2 is significantly bigger than that of m_1.

As reported in Table 3, the average prediction errors of Algo1, Algo2, NB and RF are not statistically significantly different, while SVR is significantly better than all other methods. This also means that although Algo1 and Algo2 give results that are a bit worse than those produced by NB and RF, there doesn't seem to be any advantage to using NB or RF over our two algorithms. It should be noted, however, that the sets I' of rated items are large, which means that the users considered in the above experiments have many interactions with the recommender system. When a user is not very active in the system, the results produced by the various methods can be very different. This is what we want to evaluate now, showing the advantage of Algo1 and Algo2 over other techniques for this cold start situation.

In the next experiment, we investigate the impact of the size of the training set I' on the accuracy of the recommendations. For each user dataset, we have set aside 10 test sets, each containing 50 restaurants taken from a different part of the dataset. For each test set, we considered 450 training sets of increasing size, containing $\ell = 1, \ldots, 450$ items, randomly selected from the 450 restaurants not included in the test set. We again performed a 10-folds cross validation process, applying Algo1, Algo2 and the five other machine learning techniques. The average results over the ten users are shown in Figure 3 and are also reported in Table 4, for $\ell = 1$ to 22, and for $\ell = 132$ to 150.

Our first observation is that Algo1 and Algo2 produce similar results. Hence, "don't care" values do not seem to have any impact in this experiment. We therefore only compare the output of Algo1 with those of the five machine learning algorithms. When a method is significantly better or worse than Algo1, we indicate it with a gray cell in Table 4. Smallest average errors are shown with bold characters.

The results shown in Table 4 can be divided into six main parts. For $\ell = 1$, Algo1 and Algo2 are significantly better than all other methods. For $\ell \in \{2, \ldots, 16\}$, the average error of Algo1 is the smallest and significantly smaller than that of NB, DT and NN. However, even if the results look better, they are not statistically significantly different from the results of RF (with 2 exceptions) or SVR. For $\ell \in \{17, \ldots, 450\}$, SVR got the

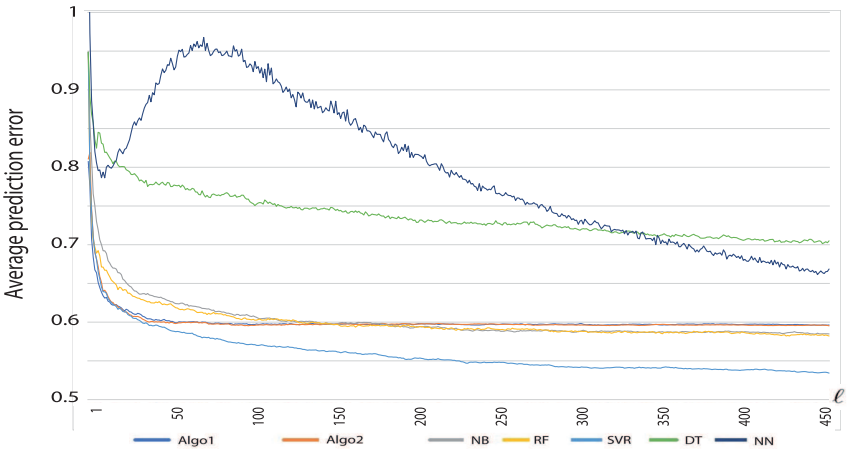

FIGURE 3. Impact of the size ℓ of the training set on the accuracy of each model.

smallest average error. Nevertheless, for rating sets containing less than 133 ratings, the error produced by SVR is not significantly smaller than that of Algo1 which produced the second best result. We also observe that the prediction errors produced with DT and NN are significantly bigger than those of Algo1 for all values $\ell = 1, \ldots, 450$. For $\ell > 148$, RF becomes the method with the second best result, slightly better than that of Algo1 and Algo2. This is summarized in Table 5.

In summary, with respect to the cold start problem, Algo1 and Algo2 produce relatively small prediction errors for small values of ℓ. As illustrated in Figure 3, the prediction error of Algo1 and Algo2 reaches its lowest value around $\ell = 50$, without a significant improvement for larger training sets. The SVR method, on the other hand, reaches the lowest error value for larger training sets, with improved prediction accuracy as ℓ increases. A zoom on the same curves for $\ell \leq 14$ is shown in Figure 4.

4.2. Experiments with movie datasets

As next experiment, we consider movie datasets taken from the MovieLens website (https://movielens.org) which allows to characterize each movie with a large set of attributes. We have extracted ten datasets, each containing 50 movies that were rated by a single user on a 5-star scale, with half-star increments (0.5 stars – 5.0 stars). Each movie is characterized using 1128 attributes which exhibit particular properties like movie's genre (action, crime, drama etc.), movie's theme (political corruption, midlife crisis, racism, memory loss etc.), awards given to the movie (Oscar awards in different categories, Saturn award for best special effects etc.), famous directors (Spielberg, Tarantino etc.), and other characteristics (true story, allegory, twist ending, thought provoking etc.) as well as viewers personal impressions (too long, unfunny, scary etc.).

As for restaurants, the dataset corresponding to user u_i, is characterized by n_i attributes from the full attributes list where, an attribute is included in the dataset only if it exists in at least one of the movies rated by the user. As a result, the numbers n_i of attributes vary from 627 to 796. More details on the 10 datasets are shown in Table 6 where we indicate, for each user, the number of attributes, the number of rated items with each score, the average rating, and the standard deviation. We notice that most ratings distribution are between 3.0 and 4.0.

For each user dataset, we have set aside 10 test sets, each containing 5 movies taken from a different part of the dataset. For each test set, we considered 45 training sets of increasing size, containing $\ell = 1, \ldots, 45$ items, randomly selected from the 45 movies not included in the test set. We again performed a 10-folds cross

TABLE 4. Average prediction error obtained for $\ell = 1$ to 22 and for $\ell = 132$ to 150. Bold numbers indicate the best results. Gray cells indicate a statistically significant difference with Algo1.

ℓ	Algo1	Algo2	NB	RF	SVR	DT	NN
1	**0.8076**	0.8105	0.9487	0.9487	0.9487	0.9487	1.1647
2	**0.7756**	0.8191	0.8591	0.7896	0.7918	0.8913	0.9730
3	**0.7076**	0.7464	0.8109	0.7356	0.7427	0.8684	0.8865
4	**0.6840**	0.7119	0.7658	0.7053	0.7097	0.8577	0.8571
5	**0.6676**	0.6897	0.7504	0.6970	0.6916	0.8361	0.8205
6	**0.6637**	0.6815	0.7297	0.6891	0.6810	0.8247	0.8057
7	**0.6515**	0.6720	0.7192	0.6924	0.6652	0.8452	0.7957
8	**0.6446**	0.6589	0.7063	0.6856	0.6559	0.8442	0.7966
9	**0.6387**	0.6514	0.7001	0.6712	0.6453	0.8312	0.7865
10	**0.6341**	0.6397	0.6922	0.6711	0.6394	0.8298	0.7937
11	**0.6311**	0.6413	0.6922	0.6670	0.6391	0.8191	0.7863
12	**0.6319**	0.6381	0.6888	0.6671	0.6371	0.8195	0.8002
13	**0.6271**	0.6331	0.6794	0.6625	0.6305	0.8160	0.8026
14	**0.6260**	0.6291	0.6764	0.6587	0.6279	0.8123	0.8000
15	**0.6230**	0.6261	0.6723	0.6544	0.6253	0.8090	0.7987
16	**0.6220**	0.6254	0.6698	0.6523	0.6231	0.8047	0.8016
17	0.6225	0.6224	0.6697	0.6504	**0.6217**	0.8081	0.8059
18	0.6226	0.6219	0.6685	0.6480	**0.6202**	0.8075	0.8089
19	0.6207	0.6196	0.6639	0.6421	**0.6165**	0.8029	0.8190
20	0.6199	0.6182	0.6637	0.6452	**0.6180**	0.8006	0.8238
21	0.6185	0.6174	0.6615	0.6441	**0.6163**	0.8009	0.8204
22	0.6162	0.6146	0.6554	0.6409	**0.6138**	0.8009	0.8174
				⋮			
132	0.5973	0.5968	0.6007	0.5994	**0.5648**	0.7471	0.8861
133	0.5980	0.5967	0.6011	0.5985	**0.5642**	0.7472	0.8841
134	0.5981	0.5969	0.6009	0.5984	**0.5637**	0.7470	0.8841
135	0.5982	0.5970	0.6012	0.5990	**0.5641**	0.7458	0.8879
136	0.5980	0.5971	0.6010	0.5998	**0.5639**	0.7466	0.8771
137	0.5983	0.5966	0.6009	0.5999	**0.5640**	0.7476	0.8869
138	0.5977	0.5971	0.5999	0.5996	**0.5640**	0.7473	0.8752
139	0.5978	0.5974	0.6003	0.5992	**0.5642**	0.7477	0.8849
140	0.5974	0.5974	0.5994	0.5991	**0.5632**	0.7451	0.8764
141	0.5971	0.5966	0.5987	0.5993	**0.5628**	0.7453	0.8776
142	0.5971	0.5967	0.5982	0.5983	**0.5631**	0.7464	0.8788
143	0.5973	0.5969	0.5973	0.5979	**0.5632**	0.7464	0.8704
144	0.5974	0.5973	0.5978	0.5983	**0.5632**	0.7473	0.8726
145	0.5974	0.5974	0.5991	0.5991	**0.5621**	0.7462	0.8769
146	0.5972	0.5976	0.5987	0.5976	**0.5621**	0.7460	0.8740
147	0.5974	0.5977	0.5977	0.5980	**0.5626**	0.7447	0.8727
148	0.5976	0.5974	0.5977	0.5976	**0.5625**	0.7482	0.8856
149	0.5970	0.5972	0.5978	0.5969	**0.5626**	0.7478	0.8787
150	0.5969	0.5974	0.5975	0.5968	**0.5625**	0.7462	0.8701

TABLE 5. Categories of results according to the size ℓ of the training set.

$\ell = 1$	Similar error as Algo1 Algo2	Smallest error Algo1	Second best error Algo2
$\in \{2, \ldots, 16\}$	Algo2 SVR RF (2 exceptions)	Algo1	SVR (1 exception)
$\in \{17, \ldots, 21\}$	Algo2 SVR RF	SVR	Algo2
$\in \{22, \ldots, 132\}$	Algo2 SVR RF NB	SVR	Algo2 (4 exceptions)
$\in \{133, \ldots, 148\}$	Algo2 RF NB	SVR	Algo2
$\in \{149, \ldots, 450\}$	Algo2 RF NB	SVR	RF

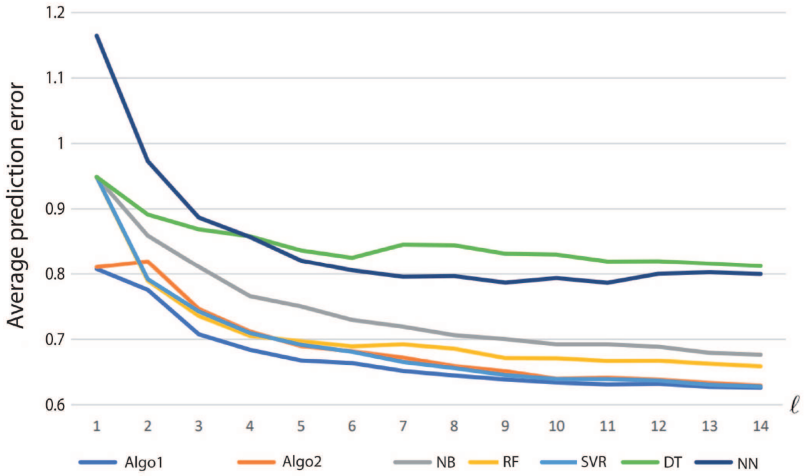

FIGURE 4. Impact of the size of the training set on the accuracy of each model.

validation process, applying Algo1, Algo2 and the five other machine learning techniques. As shown in [21], the numbers of constraints and variables in the integer programs of Algo1 and Algo2 increase linearly with the number of attributes, and we have therefore set the time limit for CPLEX at two seconds, to ensure that at least one feasible solution can be generated. Here again, no noticeable improvement can be obtained with longer computing times.

The average results over the ten users are shown in Figure 5 and are also reported in Table 7. The results of the statistical tests for $\ell \in \{1, \ldots, 45\}$, which were performed using the standard f-tests and t-tests, are represented in Table 7 as follows: when a method performs significantly better or worse than Algo2, we indicate

TABLE 6. Description of the datasets used in our experiments with movies.

		Number of rated items										Average	Standard
u_i	n_i	0.5⋆	1.0⋆	1.5⋆	2.0⋆	2.5⋆	3.0⋆	3.5⋆	4.0⋆	4.5⋆	5.0⋆	rating	deviation
u_1	753	1	0	0	1	5	13	10	15	3	2	3.45	0.79
u_2	696	0	4	0	9	0	20	0	12	0	5	3.10	1.07
u_3	735	0	0	1	4	6	11	15	11	2	0	3.26	0.69
u_4	748	0	2	0	3	0	25	0	12	0	8	3.42	0.97
u_5	627	0	0	0	1	2	25	6	12	0	4	3.42	0.67
u_6	649	0	1	0	2	0	17	0	26	0	4	3.60	0.78
u_7	745	0	0	0	1	0	18	0	26	0	5	3.70	0.68
u_8	712	0	6	0	5	0	30	0	9	0	0	2.84	0.87
u_9	796	6	4	0	6	5	5	7	6	4	7	2.96	1.47
u_{10}	755	1	1	3	2	6	11	10	8	4	4	3.26	1.04

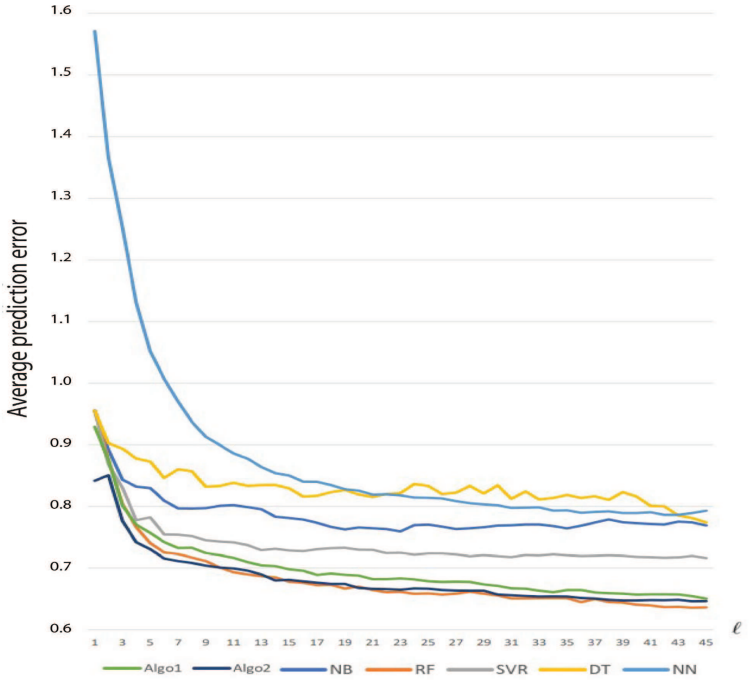

FIGURE 5. Impact of the size ℓ of the training set on the accuracy of each model for 10 movie datasets.

it with a gray cell. Best average errors are shown with bold characters. Note that the results generated by each of the methods are compared to those of Algo2 (instead of Algo1) which got better results, though not much different than Algo1, and also the smallest average error of all the methods for $\ell \in \{1, \ldots, 10\}$. A summary of the results appears in Table 8.

We can observe that the performance of Algo2 is significantly better than that of NB, DT and NN for $\ell \geq 1$, and of SVR for $\ell \neq 2$. Also, Algo1 is significantly better than NB, SVR, DT and NN for $\ell \geq 7$. Nevertheless, for $\ell \geq 11$ (with 2 exceptions), the RF method got the best results which are however not significantly different from those of Algo1 and Algo2.

TABLE 7. Average prediction error obtained for $\ell = 1$ to 45 for 10 movie datasets. Bold numbers indicate the best results. Gray cells indicate a statistically significant difference with Algo2.

ℓ	Algo1	Algo2	NB	RF	SVR	DT	NN
1	0.9292	**0.8415**	0.9555	0.9555	0.9555	0.9555	1.5712
2	0.8754	**0.8505**	0.8919	0.8757	0.8682	0.9026	1.3658
3	0.8012	**0.7762**	0.8433	0.8037	0.8303	0.8934	1.2524
4	0.7704	**0.7419**	0.8322	0.7663	0.7776	0.8777	1.1306
5	0.7570	**0.7307**	0.8298	0.7404	0.7822	0.8727	1.0515
6	0.7422	**0.7153**	0.8093	0.7255	0.7546	0.8461	1.0072
7	0.7327	**0.7111**	0.7968	0.7228	0.7541	0.8602	0.9699
8	0.7331	**0.7082**	0.7965	0.7169	0.7518	0.8568	0.9370
9	0.7244	**0.7040**	0.7972	0.7111	0.7450	0.8319	0.9131
10	0.7208	**0.7007**	0.8010	0.7008	0.7430	0.8331	0.8996
11	0.7164	0.6993	0.8019	**0.6931**	0.7416	0.8382	0.8859
12	0.7095	0.6958	0.7988	**0.6895**	0.7373	0.8334	0.8775
13	0.7044	0.6895	0.7954	**0.6867**	0.7292	0.8344	0.8639
14	0.7031	**0.6797**	0.7832	0.6848	0.7313	0.8347	0.8538
15	0.6979	0.6807	0.7812	**0.6776**	0.7287	0.8293	0.8500
16	0.6954	0.6783	0.7786	**0.6762**	0.7276	0.8159	0.8402
17	0.6887	0.6762	0.7732	**0.6722**	0.7308	0.8172	0.8400
18	0.6911	0.6742	0.7667	**0.6727**	0.7323	0.8231	0.8346
19	0.6890	0.6742	0.7625	**0.6667**	0.7330	0.8267	0.8280
20	0.6876	**0.6677**	0.7657	0.6697	0.7299	0.8194	0.8256
21	0.6817	0.6661	0.7645	**0.6644**	0.7296	0.8152	0.8190
22	0.6819	0.6657	0.7631	**0.6608**	0.7243	0.8202	0.8197
23	0.6830	0.6647	0.7594	**0.6613**	0.7249	0.8216	0.8180
24	0.6814	0.6669	0.7694	**0.6581**	0.7218	0.8363	0.8144
25	0.6785	0.6665	0.7704	**0.6588**	0.7237	0.8332	0.8140
26	0.6774	0.6642	0.7672	**0.6569**	0.7237	0.8200	0.8130
27	0.6777	0.6634	0.7632	**0.6584**	0.7222	0.8223	0.8085
28	0.6773	0.6633	0.7644	**0.6616**	0.7190	0.8333	0.8051
29	0.6734	0.6632	0.7661	**0.6584**	0.7209	0.8209	0.8033
30	0.6710	0.6569	0.7689	**0.6554**	0.7191	0.8342	0.8017
31	0.6670	0.6558	0.7693	**0.6509**	0.7177	0.8123	0.7976
32	0.6665	0.6547	0.7705	**0.6508**	0.7210	0.8244	0.7980
33	0.6632	0.6538	0.7706	**0.6513**	0.7203	0.8113	0.7983
34	0.6608	0.6540	0.7681	**0.6512**	0.7224	0.8137	0.7932
35	0.6643	0.6537	0.7642	**0.6512**	0.7205	0.8187	0.7936
36	0.6643	0.6514	0.7686	**0.6447**	0.7194	0.8137	0.7896
37	0.6604	0.6506	0.7737	**0.6497**	0.7197	0.8166	0.7910
38	0.6593	0.6484	0.7788	**0.6449**	0.7206	0.8107	0.7918
39	0.6584	0.6475	0.7742	**0.6441**	0.7197	0.8232	0.7891
40	0.6570	0.6474	0.7726	**0.6407**	0.7181	0.8161	0.7892
41	0.6573	0.6481	0.7715	**0.6394**	0.7174	0.8006	0.7903
42	0.6574	0.6475	0.7705	**0.6366**	0.7165	0.8001	0.7863
43	0.6572	0.6483	0.7751	**0.6371**	0.7173	0.7850	0.7863
44	0.6544	0.6426	0.7739	**0.6358**	0.7196	0.7811	0.7891
45	0.6504	0.6422	0.7690	**0.6363**	0.7160	0.7740	0.7930

TABLE 8. Categories of results according to the size ℓ of the training set.

ℓ	Similar error as Algo2	Smallest error	Second best error
1	–	Algo2	Algo1
2	Algo1 RF SVR	Algo2	SVR
3	Algo1	Algo2	Algo1
$\in \{4, 5, 6\}$	RF	Algo2	RF
$\in \{7, 8, 9, 10, 14, 20\}$	Algo1 RF	Algo2	RF
$\in \{11, \ldots, 45\} \setminus \{14, 20\}$	Algo1 RF	RF	Algo2

To test Algo1 and Algo2 on a larger number of instances, we have extracted 1000 datasets from the MovieLens website (https://movielens.org), each containing 50 movies that were rated by a single user on a 5-star scale, with half-star increments (0.5 stars – 5.0 stars). As a result, the numbers n_i of attributes vary from 440 to 844. Note that collaborative filtering methods are usually evaluated using more users than content based methods which in turn, need a lot of data to describe item attributes. As above, for each user dataset, we have set aside 10 test sets, each containing 5 movies taken from a different part of the dataset. For each test set, we considered 45 training sets of increasing size, containing $\ell = 1, \ldots, 20$ items, randomly selected from the 45 movies not included in the test set. Larger sizes ℓ for the training set are less relevant to the cold start problem. We again performed a 10-folds cross validation process, applying Algo1, Algo2 and the two best standard machine learning techniques for the cold start problem, namely RF and SVR.

The results are shown in Figure 6 and are also reported in Table 9. When a method performs significantly better or worse than Algo2, we indicate it with a gray cell. Best average errors are shown with bold characters.

Algo2 gives the best average error for training sets of size $\ell = 1$ and for $3 \leq \ell \leq 7$, while RF performs better for the other values of ℓ. However, as the statistical tests indicate, for all sizes of the training sets except for $\ell = 1$, there is no statistically significant difference between the results produced by Algo2 and RF. For $\ell = 1$ Algo2 is significantly better than the other three methods. Algo1 and SVR are significantly worse than Algo2 for $\ell = 1$ and also for $\ell > 3$. In summary, when there is only a single previous user rating, Algo2 is the preferable method. There is no statistically significant difference between the four methods for $\ell = 2$ and 3. For training sets of size $\ell = 5$ to 20, Algo2 and RF produce the best recommendations among all the methods, and the second-best methods are Algo1 and SVR.

5. Conclusions and future work

Nowadays, state of the art recommender systems are based on deep learning and graph embedding techniques, which are used in various platforms and domains [53]. These systems need a lot of data and therefore are not suitable for small datasets where classical machine learning techniques may perform well. Also, the cold start problem is still a challenge, as even classical machine learning techniques need enough examples for training.

To address the cold start problem, we have proposed in [21] a CB recommender system that builds user models on the basis of hypercube graphs. Limited experiments have shown that user preferences can be determined with a very limited number of ratings. In this paper, we have confirmed these findings on the basis of larger scale experiments. We have shown that our method allows building user models with a very small number of ratings and therefore constitutes a particularly interesting approach to solve the cold start problem. In particular,

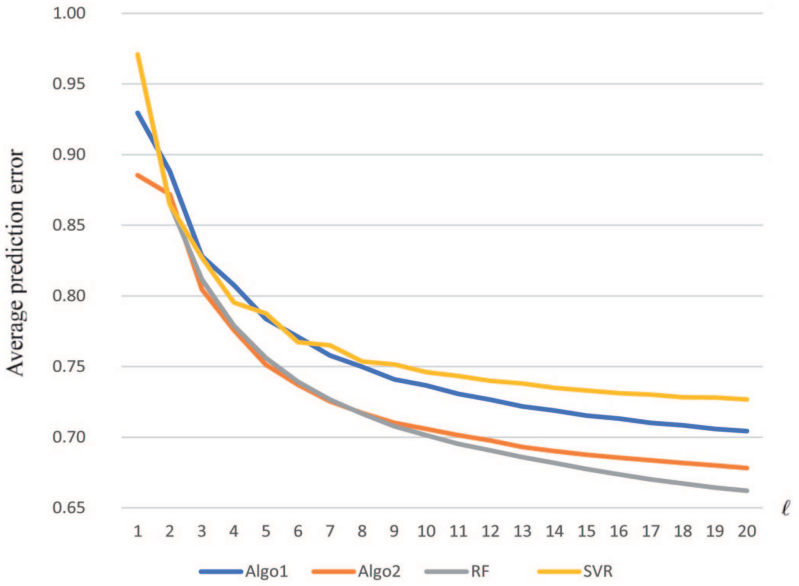

FIGURE 6. Impact of the size ℓ of the training set on the accuracy of each model for 1000 movie datasets.

TABLE 9. Average prediction error obtained for $\ell = 1$ to 20 for 1000 movie datasets. Bold numbers indicate the best results. Gray cells indicate a statistically significant difference with Algo2.

ℓ	Algo1	Algo2	RF	SVR
1	0.9294	**0.8854**	0.9708	0.9708
2	0.888251	0.8719	**0.8653**	0.8646
3	0.828108	**0.8047**	0.8116	0.827
4	0.8076	**0.7758**	0.7788	0.7953
5	0.7839	**0.7514**	0.7561	0.7876
6	0.7710	**0.7372**	0.7392	0.7672
7	0.7579	**0.7254**	0.7266	0.7651
8	0.7499	0.7171	**0.7166**	0.7535
9	0.7409	0.7102	**0.7077**	0.7515
10	0.7366	0.7059	**0.7014**	0.7461
11	0.7307	0.7015	**0.6953**	0.7434
12	0.7266	0.6976	**0.6906**	0.7399
13	0.7218	0.6931	**0.6859**	0.738
14	0.7189	0.6901	**0.6818**	0.735
15	0.7153	0.6876	**0.6775**	0.7331
16	0.7133	0.6856	**0.6737**	0.7311
17	0.7101	0.6837	**0.6702**	0.7301
18	0.7084	0.6818	**0.6673**	0.7282
19	0.7059	0.6801	**0.6644**	0.7281
20	0.7043	0.6782	**0.6621**	0.7267

experiments have shown that Algo1 and Algo2 outperform standard machine learning algorithms when the number of available ratings is at most 10, which often happens (see Fig. 1).

In all the experiments reported in Section 4, the four competing methods which consistently produced the smallest average prediction error were Algo1, Algo2, RF and SVR. In the restaurants domain where we conducted a small-scale experiment, Algo1 gave the best predictions for small-sized training sets, with no statistically significant difference between the average prediction error obtained by the four methods. In the large-scale experiment conducted in the movie domain, Algo2 and RF were superior, with no statistically significant difference between them, where Algo1 and SVR got the second-best results.

In summary, Tables 7 and 9 show that Algo2 produces significantly smaller prediction errors (with very minor exceptions) than four out of the five tested machine learning methods. Note also that while these two tables contain results for 10 and 1000 users respectively, we see a similar trend that demonstrates the benefit of using Algo2:

- SVR is significantly worse than Algo2;
- For small-sized training sets, Algo2 produces the smallest prediction error, but with no statistically significant difference when compared to RF;
- For large-sized training sets, the results obtained by RF are the best but they are not statistically significantly different from the results obtained by Algo2.

In order to get accurate predictions for any size of user ratings datasets, the methods can be combined. That is, we propose, as part of further research, to add to our model a hybrid mode which uses Algo1 and Algo2 for small size datasets, and when the number of user ratings is sufficient, the system switches to using RF or SVR for producing recommendations. However, the following should be considered when building a hybrid system. A major drawback of RF is that it can be slow because this method averages predictions obtained from a series of regression trees. To improve the precision of the predictions produced by RF, it is necessary to increase the number of trees which is computationally expensive [9]. The SVR method, on the other hand, suffers from high training complexity [1].

A limitation of the research described in this paper is that the data for the experiments was extracted from the restaurant and movie domains only. Although the movie domain is commonly used for evaluating recommender systems algorithms [13, 48], we would like to expand the experiments to other domains. Another limitation of this study is the use of a star scale in our models. However, the proposed algorithms can easily be adapted to deal with other rating scales. Other notable issues for future work are as follows.

- *Privacy.* Preserving user privacy is a challenge that needs to be addressed since datasets may include sensitive and vulnerable information [22]. The proposed recommendation models Algo1 and Algo2 allow preserving the confidential data of the users because the information necessary to identify the preferences of each user can be kept locally and doesn't require the knowledge of the ratings of other users. In future research we intend to implement Algo1 and Algo2 techniques on a laptop or a smartphone while using a mechanism for preserving user privacy, as proposed in [26].
- *Resolving sets.* A subset $I' \subseteq I$ of items rated by a user $u \in U$ is a *resolving set* of the hypercube Q_n if, given any two vertices $\boldsymbol{x}, \boldsymbol{y}$ in Q_n, there is at least one item $i \in I'$ such that $d(\boldsymbol{x}, \boldsymbol{v}^i) \neq d(\boldsymbol{y}, \boldsymbol{v}^i)$ [4]. In our context, this means that the knowledge of the Hamming distances between the vertex $\boldsymbol{w}^u$ associated with user u and the vertices of a resolving set of Q_n are sufficient to unequivocally determine the opinion of u on each attribute $a \in A$. As the resolving sets of hypercube graphs are of very small size [20], this approach can be used to solve the cold start problem. In future work, we will study how to encourage users of a recommender system to evaluate items that constitute a resolving set in order to know their preferences based on very few ratings.

In conclusion, the implications of this study are twofold: theoretical and practical. From a theoretical perspective, we have demonstrated that a simple and innovative method can estimate the profiles of users of a recommender system based on their previous ratings. This method makes it possible to infer, for each attribute

of an item, whether the user likes it, doesn't care about it, or dislikes it. From a practical perspective, we propose a computationally effective mechanism that doesn't require a lot of data or space for generating recommendations. It can be installed and used locally on the user's device without sharing data with a service provider, thus helping to preserve their privacy. Moreover, it allows users to obtain predictions for items (*e.g.*, restaurants) offline, without an internet connection, with all computations performed locally.

Funding

This research was partially supported by the Israeli Science foundation, grant number 216/23.

Data availability statement

The research data associated with this article are included in the article.

References

[1] A. Abdiansah and R. Wardoyo, Time complexity analysis of support vector machines (SVM) in LibSVM. *Int. J. Comput. App.* **128** (2015) 28–34.

[2] G. Adomavicius and A. Tuzhilin, Toward the next generation of recommender systems: a survey of the state-of-the-art and possible extensions. *IEEE Trans. Knowl. Data Eng.* **17** (2005) 734–749.

[3] C.C. Aggarwal, Recommender Systems – The Textbook. Springer (2016) 1–166.

[4] R. Belmonte, F.V. Fomin, P.A. Golovach and M.S. Ramanujan, Metric dimension of bounded width graphs, in Mathematical Foundations of Computer Science 2015, edited by G.F. Italiano, G. Pighizzini and D.T. Sannella. Springer Berlin Heidelberg, Berlin, Heidelberg (2015) 115–126.

[5] S. Berkovsky, T. Kuflik and F. Ricci, Mediation of user models for enhanced personalization in recommender systems. *User Modell. User-Adapted Interact.* **18** (2008) 245–286.

[6] S. Berkovsky, T. Kuflik and F. Ricci, Cross-representation mediation of user models. *User Modell. User-Adapted Interact.* **19** (2009) 35–63.

[7] L. Bernardi, J. Kamps, Y. Kiseleva and M. Mueller, The continuous cold start problem in e-commerce recommender systems, in CEUR Workshop Proceedings. Vol. 1448 (2015) 30–33.

[8] R. Burke, Hybrid web recommender systems, in The Adaptive Web: Methods and Strategies of Web Personalization. Springer Berlin Heidelberg, Berlin, Heidelberg (2007) 377–408.

[9] T. Buskirk, Surveying the forests and sampling the trees: an overview of classification and regression trees and random forests with applications in survey research. *Surv. Pract.* **11** (2018) 1–13.

[10] H. Cai, V.W. Zheng and K.C.-C. Chang, A comprehensive survey of graph embedding: problems, techniques, and applications. *IEEE Trans. Knowl. Data Eng.* **30** (2018) 1616–1637.

[11] E. Çano and M. Morisio, Hybrid recommender systems: a systematic literature review. *Intell. Data Anal.* **21** (2017) 1487–1524.

[12] Y. Chen, M. Zhou, Z. Zheng and D. Chen, Time-aware smart object recommendation in social internet of things. *IEEE Int. Things J.* **7** (2020) 2014–2027.

[13] A. Da'u and N. Salim, Recommendation system based on deep learning methods: a systematic review and new directions. *Artif. Intell. Rev.* **53** (2020) 2709–2748.

[14] M. Dong, F. Yuan, L. Yao, X. Xu and L. Zhu, Mamo: memory-augmented meta-optimization for cold-start recommendation, in Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '20. Association for Computing Machinery, New York, NY, USA (2020) 688–697.

[15] W. Fan, Y. Ma, Q. Li, Y. He, E. Zhao, J. Tang and D. Yin, Graph neural networks for social recommendation, in The World Wide Web Conference, WWW '19 (2019) 417–426.

[16] X. Guan, Z. Cheng, X. He, Y. Zhang, Z. Zhu, Q. Peng and T.-S. Chua, Attentive aspect modeling for review-aware recommendation. *ACM Trans. Inf. Syst.* **37** (2019) 1–27.

[17] A. Gunawardana, G. Shani and S. Yogev, Evaluating recommender systems, in Recommender Systems Handbook. Springer US, New York, NY (2022) 547–601.

[18] S. Gupta and S. Goel, Handling user cold start problem in recommender systems using fuzzy clustering, in Information and Communication Technology for Sustainable Development, edited by D.K. Mishra, M.K. Nayak and A. Joshi. Springer Singapore, Singapore (2018) 143–151.

[19] B. Hao, J. Zhang, C. Li and H. Chen, Few-shot representation learning for cold-start users and items, in Web and Big Data, edited by X. Wang, R. Zhang, Y.-K. Lee, L. Sun and Y.-S. Moon. Springer International Publishing (2020) 363–377.

[20] A. Hertz, An IP-based swapping algorithm for the metric dimension and minimal doubly resolving set problems in hypercubes. *Optim. Lett.* **14** (2020) 355–367.

[21] A. Hertz, T. Kuflik and N. Tuval, Resolving sets and integer programs for recommender systems. *J. Global Optim.* **81** (2021) 153–178.

[22] Y. Himeur, S.S. Sohail, F. Bensaali, A. Amira and M. Alazab, Latest trends of security and privacy in recommender systems: a comprehensive review and future perspectives. *Comput. Secur.* **118** (2022) 102746.

[23] J.C. Jackson and K. Wimmer, New results for random walk learning. *J. Mach. Learn. Res.* **15** (2014) 3655–3666.

[24] H. Ko, S. Lee, Y. Park and A. Choi, A survey of recommendation systems: recommendation models, techniques, and application fields. *Electronics* **11** (2022) 141.

[25] Y. Koren, S. Rendle and R. Bell, Advances in collaborative filtering, in Recommender Systems Handbook. Springer US, New York, NY (2022) 91–142.

[26] T. Kuflik and K. Poteriaykina, Integrating user modeling server with user modeling mediator on a personal device, in Proceedings of the 7th International Workshop on Ubiquitous User Modelling (2009) 11–15.

[27] H. Lee, J. Im, S. Jang, H. Cho and S. Chung, Melu: meta-learned user preference estimator for cold-start recommendation, in Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD '19. Association for Computing Machinery, New York, NY, USA (2019) 1073–1082.

[28] J. Li, M. Jing, K. Lu, L. Zhu, Y. Yang and Z. Huang, From zero-shot learning to cold-start recommendation, in Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence and Thirty-First Innovative Applications of Artificial Intelligence Conference and Ninth AAAI Symposium on Educational Advances in Artificial Intelligence. AAAI'19/IAAI'19/EAAI'19, AAAI Press (2019) 4189–4196.

[29] W. Ma, M. Zhang, Y. Cao, W. Jin, C. Wang, Y. Liu, S. Ma and X. Ren, Jointly learning explainable rules for recommendation with knowledge graph, in The World Wide Web Conference, WWW '19 (2019) 1210–1221.

[30] M. Mansoury, R. Burke and B. Mobasher, Flatter is better: percentile transformations for recommender systems. *ACM Trans. Intell. Syst. Technol.* **12** (2021) 1–16.

[31] E. Minkov, K. Kahanov and T. Kuflik, Graph-based recommendation integrating rating history and domain knowledge: application to on-site guidance of museum visitors. *J. Assoc. Inf. Sci. Technol.* **68** (2017) 1911–1924.

[32] C. Musto, M.D. Gemmis, P. Lops, F. Narducci and G. Semeraro, Semantics and content-based recommendations, in Recommender Systems Handbook. Springer US, New York, NY (2022) 251–298.

[33] A.N. Nikolakopoulos, X. Ning, C. Desrosiers and G. Karypis, Trust Your Neighbors: A Comprehensive Survey of Neighborhood-Based Methods for Recommender Systems. Springer US, New York, NY (2022) 39–89.

[34] W. Niu, J. Caverlee and H. Lu, Neural personalized ranking for image recommendation, in Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining, WSDM '18 (2018) 423–431.

[35] D.K. Panda and S. Ray, Approaches and algorithms to mitigate cold start problems in recommender systems: a systematic literature review. *J. Intell. Inf. Syst.* **59** (2022) 341–366.

[36] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot and E. Duchesnay, Scikit-learn: machine learning in python. *J. Mach. Learn. Res.* **12** (2011) 2825–2830.

[37] I. Portugal, P. Alencar and D. Cowan, The use of machine learning algorithms in recommender systems: a systematic review. *Expert Syst. App.* **97** (2018) 205–227.

[38] R. Qiu, Z. Huang, J. Li and H. Yin, Exploiting cross-session information for session-based recommendation with graph neural networks. *ACM Trans. Inf. Syst.* **38** (2020) 1–23.

[39] A.M. Rashid, I. Albert, D. Cosley, S.K. Lam, S.M. McNee, J.A. Konstan and J. Riedl, Getting to know you: learning new user preferences in recommender systems, in Proceedings of the 7th International Conference on Intelligent User Interfaces, IUI '02 (2002) 127–134.

[40] F. Ricci, L. Rokach and B. Shapira, Recommender systems: techniques, applications, and challenges, in Recommender Systems Handbook. Springer US, New York, NY (2022) 1–35.

[41] A. Said and A. Bellogín, Coherence and inconsistencies in rating behavior: estimating the magic barrier of recommender systems. *User Modell. User-Adapted Interact.* **28** (2018) 97–125.

[42] B.B. Sinha and R. Dhanalakshmi, Evolution of recommender paradigm optimization over time. *J. King Saud Univ. Comput. Inf. Sci.* **34** (2022) 1047–1059.

[43] A. Tiroshi, T. Kuflik, S. Berkovsky and M. Kaafar, Graph-based recommendations: from data representation to feature extraction and application, in Big Data Recommender Systems – Volume 2: Application Paradigms, Computing. Institution of Engineering and Technology (2019) 407–454.

[44] J. Verma, S. Gupta, D. Mukherjee and T. Chakraborty, Heterogeneous edge embedding for friend recommendation, in Advances in Information Retrieval, edited by L. Azzopardi, B. Stein, N. Fuhr, P. Mayr, C. Hauff and D. Hiemstra. Springer International Publishing, Cham (2019) 172–179.

[45] J. Vinagre, A. Jorge and J. Gama, An overview on the exploitation of time in collaborative filtering. *Wiley Interdiscipl. Rev.: Data Mining Knowl. Discovery* **5** (2015) 195–215.

[46] M. Volkovs, G. Yu and T. Poutanen, Dropoutnet: addressing cold start in recommender systems, in Proceedings of the 31st International Conference on Neural Information Processing Systems, NIPS'17 (2017) 4964–4973.

[47] H. Wang, D. Amagata, T. Makeawa, T. Hara, N. Hao, K. Yonekawa and M. Kurokawa, A DNN-based cross-domain recommender system for alleviating cold-start problem in e-commerce. *IEEE Open J. Ind. Electron. Soc.* **1** (2020) 194–206.

[48] Y. Wang, Q. Yao, J.T. Kwok and L.M. Ni, Generalizing from a few examples: a survey on few-shot learning. *ACM Comput. Surv.* **53** (2020) 1–34.

[49] S. Wang, L. Hu, Y. Wang, X. He, Q. Sheng, M. Orgun, L. Cao, F. Ricci and P. Yu, Graph learning based recommender systems: a review, in Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI 2021, edited by Z.-H. Zhou (2021) 4644–4652.

[50] E. Wasserman Pritsker, T. Kuflik and E. Minkov, Assessing the contribution of twitter's textual information to graph-based recommendation, in Proceedings of the 22nd International Conference on Intelligent User Interfaces (2017) 511–516.

[51] H. Yuan and A.A. Hernandez, User cold start problem in recommendation systems: a systematic review. *IEEE Access* **11** (2023) 136958–136977.

[52] Q. Zhang, J. Lu and Y. Jin, Artificial intelligence in recommender systems. *Complex Intell. Syst.* **7** (2021) 439–457.

[53] S. Zhang, Y. Tay, L. Yao, A. Sun and C. Zhang, Deep learning for recommender systems, in Recommender Systems Handbook. Springer US, New York, NY (2022) 173–210.

[54] Z. Zhu, J. Kim, T. Nguyen, A. Fenton and J. Caverlee, Fairness among new items in cold start recommender systems, in Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR'21 (2021) 767–776.