Les Cahiers du GERAD ISSN: 0711–2440

Addressing the cold start problem in privacy preserving content-based recommender systems using hypercube graphs

N. Tuval, A. Hertz, T. Kuflik

G–2022–55

December 2022

La collection *Les Cahiers du GERAD* est constituée des travaux de recherche menés par nos membres. La plupart de ces documents de travail a été soumis à des revues avec comité de révision. Lorsqu'un document est accepté et publié, le pdf original est retiré si c'est nécessaire et un lien vers l'article publié est ajouté.

Citation suggérée : N. Tuval, A. Hertz, T. Kuflik (Décembre 2022). Addressing the cold start problem in privacy preserving content-based recommender systems using hypercube graphs, Rapport technique, Les Cahiers du GERAD G– 2022–55, GERAD, HEC Montréal, Canada.

Avant de citer ce rapport technique, veuillez visiter notre site Web (https://www.gerad.ca/fr/papers/G-2022-55) afin de mettre à jour vos données de référence, s'il a été publié dans une revue scientifique.

The series *Les Cahiers du GERAD* consists of working papers carried out by our members. Most of these pre-prints have been submitted to peer-reviewed journals. When accepted and published, if necessary, the original pdf is removed and a link to the published article is added.

Suggested citation: N. Tuval, A. Hertz, T. Kuflik (December 2022). Addressing the cold start problem in privacy preserving content-based recommender systems using hypercube graphs, Technical report, Les Cahiers du GERAD G–2022–55, GERAD, HEC Montréal, Canada.

Before citing this technical report, please visit our website (https://www.gerad.ca/en/papers/G-2022-55) to update your reference data, if it has been published in a scientific journal.

La publication de ces rapports de recherche est rendue possible grâce au soutien de HEC Montréal, Polytechnique Montréal, Université McGill, Université du Québec à Montréal, ainsi que du Fonds de recherche du Québec – Nature et technologies.

Dépôt légal – Bibliothèque et Archives nationales du Québec, 2022
– Bibliothèque et Archives Canada, 2022

The publication of these research reports is made possible thanks to the support of HEC Montréal, Polytechnique Montréal, McGill University, Université du Québec à Montréal, as well as the Fonds de recherche du Québec – Nature et technologies.

Legal deposit – Bibliothèque et Archives nationales du Québec, 2022
– Library and Archives Canada, 2022

GERAD HEC Montréal
3000, chemin de la Côte-Sainte-Catherine
Montréal (Québec) Canada H3T 2A7

Tél. : 514 340-6053
Téléc. : 514 340-5665
info@gerad.ca
www.gerad.ca

Addressing the cold start problem in privacy preserving content-based recommender systems using hypercube graphs

Noa Tuval [a]

Alain Hertz [b]

Tsvika Kuflik [a]

[a] *Information Systems Department, The University of Haifa, Mount Carmel, Haifa, Israel, 3498838*

[b] *Department of Mathematics and Industrial Engineering, Polytechnique Montréal & GERAD, Montréal (Qc), Canada, H3T 1J4*

alain.hertz@gerad.ca

December 2022
Les Cahiers du GERAD
G–2022–55

Copyright © 2022 GERAD, Tuval, Hertz, Kuflik

Les textes publiés dans la série des rapports de recherche *Les Cahiers du GERAD* n'engagent que la responsabilité de leurs auteurs. Les auteurs conservent leur droit d'auteur et leurs droits moraux sur leurs publications et les utilisateurs s'engagent à reconnaître et respecter les exigences légales associées à ces droits. Ainsi, les utilisateurs:

- Peuvent télécharger et imprimer une copie de toute publication du portail public aux fins d'étude ou de recherche privée;
- Ne peuvent pas distribuer le matériel ou l'utiliser pour une activité à but lucratif ou pour un gain commercial;
- Peuvent distribuer gratuitement l'URL identifiant la publication.

Si vous pensez que ce document enfreint le droit d'auteur, contactez-nous en fournissant des détails. Nous supprimerons immédiatement l'accès au travail et enquêterons sur votre demande.

The authors are exclusively responsible for the content of their research papers published in the series *Les Cahiers du GERAD*. Copyright and moral rights for the publications are retained by the authors and the users must commit themselves to recognize and abide the legal requirements associated with these rights. Thus, users:

- May download and print one copy of any publication from the public portal for the purpose of private study or research;
- May not further distribute the material or use it for any profit-making activity or commercial gain;
- May freely distribute the URL identifying the publication.

If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

Abstract : Recommender systems provide recommendations to their users for items and services by creating a model tailored to each user to infer their preferences based on previous interactions they have made with the system, as well as possibly using additional information sources. 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. Graph-based user modeling has now become common practice and we have recently shown that a content-based model that uses hypercube graphs can determine user preferences with a very limited number of ratings while better preserving user privacy. In this paper, we confirm these findings on the basis of larger scale experiments. We thus address the cold start problem by building user models based on a small number of ratings. We also show that the proposed method outperforms standard machine learning algorithms when the number of available ratings is very limited.

Keywords : Recommender systems, cold start problem, hypercube graphs

1 Introduction

The growing involvement of recommender systems in our daily life, including the online purchase of products and services, 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]. 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 him in the past. [48].

CF systems need a lot of data to infer the preferences of their users and the quality of CF recommendations is highly dependent on the sparsity of available data [3]. CF systems also have difficulty in producing good recommendations when there are very few users compared to the number of items, since the predictions are calculated on the basis of the observed interactions [29]. Moreover, CF systems suffer from *the new item problem*: new products with no rating history are unlikely to be recommended [48, 64]. This problem does not exist in CB systems because recommending an item depends solely on its attributes. Also, CB systems do not need data from other users to generate recommendations for a particular user, which means they can maintain user privacy better than CF systems. On the 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. For example, the system will not recommend a movie from a different genre than the genres the user usually watches, because it only recommends items with similar attributes to items the user has liked in the past [10].

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 [28]. 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 [23]. 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, 23, 34]. 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, 11]. These preferences can be stored locally on the user's personal device and not shared [35, 36]. In this aspect, CB systems preserve the privacy of users better than CF systems. Maintaining privacy is also a major challenge in cross-domain recommender systems where user data is shared between different systems. However, most studies in this area deal with privacy preservation in single-domain recommender systems [28].

An improvement to the classic recommendation methods are the *hybrid* methods [7, 8, 10, 14, 48] and *graph-based* methods [3, 48, 60]. 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 [14, 48]. 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. Other challenges in hybrid systems are creating context-based recommendations that take into account a variety of user contexts as well as developing cross-domain recommender systems [14].

Graph-based methods provide convenient and versatile structures for representing the relationships among users and items [3]. 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 [48, 60]. 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 [60].

It is natural to question what happens when a recommender system does not have enough information to define an adequate model for each 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* [11, 62]. It is well known that this cold start problem 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 [57]. Moreover, since CF systems derive user preferences based on previous ratings, they must have enough ratings to create reliable user models [2, 10, 20, 40].

As stated in [14, 51, 62], the cold start problem is one of the research challenges of recommender systems. The problem has caught the attention of many researchers [9, 24, 49] 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. A zoom on the part with less than 100 ratings appears in Figure 2. It is clear that the vast majority of users of the system have provided very few evaluations. For example, there are 774,639 users who provided less than 5 ratings (including 298,463 users with only one rating), 1,069,244 users with less than 10 ratings, but only 13,351 users with at least 20 ratings.

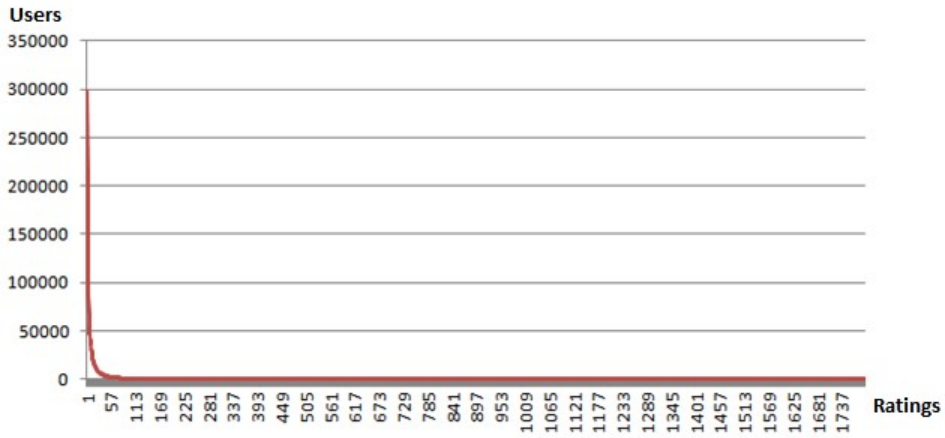

Figure 1: The long tail distribution of the number of ratings per user (taken from the yelp website https://www.yelp.com/dataset).

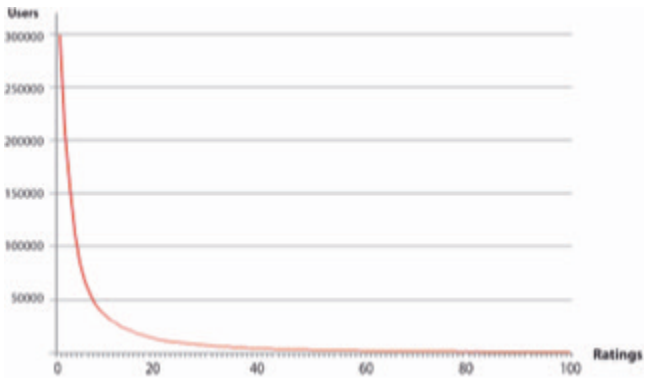

Figure 2: Zoom of Figure 1 for users with at most 100 ratings.

We have recently suggested a new approach to create user models with CB systems [27]. 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 (see Figure 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 [27] 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. 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 machine learning 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

A user model is a representation of the relevant characteristics of a user who interacts with a recommender system [33]. 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 [48].

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 [48]. 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, 34]. 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 [3, 45].

CB approaches utilize user preferences over a series of item attributes to recommend additional items with similar properties. They try to match a user model that describes his preferences with the item attributes to find the most appropriate items to recommend [40, 48]. 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, 38].

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 [10, 48]. One of the hybrid techniques is the cross-representation mediation of user models from CF to CB recommender systems [7, 8]. 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 [62]. 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 [40, 62]. Unlike existing linear models such as matrix factorization, which have a limited understanding of complex interaction patterns between users and items, deep learning methods are able to perform non-linear transformations of data for a better understanding of user preferences [63]. Another advantage of deep learning techniques is that they 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 [62, 63]. For example, in the movie domain, 'for adults only' and 'over 18' are two aspects referring to the same attribute, (where an aspect is a set of words used by a user in his reviews to describe an item attribute). To overcome this problem, a technique is proposed in [22] that models the interactions between similar aspects and then uses a deep neural network that simultaneously considers users, items, and aspect information to capture the importance a user gives to each aspect when looking at different items.

Another example of the benefit of using deep learning methods is the study in [52] where a neural network is proposed for a quote recommendation task. The network exploits abundant quote repositories to learn the meaning of quotes and help people make better use of quotes while writing. Extracting item attributes is also challenging in music recommendation, since music attributes are melody, rhythm and timbre [62]. A deep neural network is used in [55] to predict latent factors which describe user tastes from music audio content. Image recommender systems also use deep learning techniques for better user modeling. For example, in social imaging sharing communities (like Instagram and Pinterest) only implicit feedback is available. A neural personalized ranking model is proposed in [43] for image recommendation, which exploits user contextual preference clues from implicit feedback datasets and extracts visual attributes using a deep neural network model.

Finally, deep learning techniques are also increasingly used in session-based recommender systems which recommend an item based on the most recent preference of the user in the current session, where a session means a sequence of interactions between a user and items within a period [31, 46]. For example, a method is proposed in [54] that combines session clicks and content attributes to produce recommendations, while using a deep neural network to model the different types of data.

The main challenges of deep learning techniques are architecture improvement and system optimization [19]. 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 [19, 62]. 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 [62].

2.2 Graph-based user modeling

The rapid growth in the use of social networks, which provide free personal information about users, has led to the development and use of new tools for user modeling such as graphs and network data structures. The main advantage of these models is that they allow combining entities and their mutual links in simple structures, without loss of information [53]. 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 [60].

Graph models are used, for example, to define similarities in neighborhood-based methods when rating data is sparse in comparison to the number of users [3, 48]. In this approach, the data is represented by a graph where nodes are users, items or both, and edges encode interactions or similarities between users and items. 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 [42]. Graph models are also used to detect similarities between users or items, these similarities being measured as a function of the probability of reaching one vertex from another in a random walk [30, 41]. The similarity between two users or two items can also be measured using the path-based approach where the distance between two vertices of a graph is computed based on the lengths of the paths connecting them and the number of such paths. [42].

As already noted, 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 [62]. However, these graphs can be extended by adding additional attributes as new graph vertices and linking them to the existing vertices [60]. The resulting graph is called *heterogeneous* and enables integrating collaborative ratings with domain knowledge and content aspects for better personalized recommendations [39, 60]. A graph-based user model is proposed in [61] for movie recommendation. It uses textual information extracted from the social network to improve user profiling. The vertices of the heterogeneous graph represent entities like users, movies, topics, linguistic variables and personal traits which are derived from the data (e.g. gender, age), while the graph edges correspond to inter-relations between the entities (e.g. a user watched a movie). Random walk techniques, which are widely used in different types of graphs [60], are then used for discovering relationships between the vertices of the graph and producing recommendations [61]. Another example of the use of heterogeneous graphs is the recommender system for museum visitors described in [39]. It combines rating history and domain knowledge. Also, a recommender system for heterogeneous graphs is described in [32] which learns the weights of the edges using a Bayesian personalized ranking-based machine learning method and then uses a random walk with restart to model user preferences. Although random walk methods are widely used, their main disadvantage is that they are heuristic-based, lacking model parameters that can contribute to improve the recommendation process [60].

Graph Neural Networks (GNN) is a deep learning approach designed to learn graph-structured data from vertices and edges through message propagation and aggregation. This technique aims to help understanding all graph components and hidden data, which is essential for recommender systems. In the repeated learning process carried out by GNN, information from the neighboring vertices is pushed, passed and added to each vertex in the graph. This deep learning process allows information to be propagated across all graph elements [62]. When considering session-based recommender systems a session graph is a convenient way to represent click sequences and transition patterns from sessions [62]. For example, a technique is proposed in [46] that converts click sequences into a weighted directed session graph and learns click representation and item dependencies using a GNN approach. GNN is also used for combining graph-related data such as social graphs that represent user-item connections and user-user links which describe the mutual influence between the linked users [62]. For example, a GNN framework is proposed in [21] for social recommendations. Its objective is to capture interactions and opinions in the user-item graph by using aggregation techniques and an attention mechanism to measure social relations. Hence, GNN techniques have proved their potential to learn from graph structured data. However, a major challenge of this model is the low computation efficiency due to the big amount of data required for the learning process [13, 62].

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 [13]. For example, an edge embedding technique is proposed in [56] for friend recommendation. Also, a time-aware smart object recommendation model is described in [17] which uses a latent probabilistic

model to learn user preferences over time based on items usage events. The social relationships of smart objects (e.g. smart watch, smart kitchen) are embedded into a shared lower dimensional space to infer their social similarities, and an item-based CF method is then used for recommendation.

Graph representation is also adopted for producing explainable recommendations, which is an aspect of recommender systems that has received recent attention. For example, an approach is proposed in [37] for recommending items from explainable rules. An heterogeneous knowledge graphs is first built where the items are the vertices and different types of item associations (e.g. 'buy together') are edges. A rule learning process is then applied based on a random walk algorithm that computes for each pair of items the probabilities of finding paths that follow particular rules between the pair. The induced rules are then used to infer associations between new items and to produce more explainable recommendations.

2.3 The cold start problem in recommender systems

A recommender system collects the ratings that a user has assigned to items in order to produce recommendations for the user, according to his preferences and tastes. The more the user interacts with the system, the more information can be accumulated that can help the system to accurately determine user preferences [2, 48]. Thus, when someone starts using a recommender system, or when his interaction with it is superficial, there is not enough information available for the system to create a reliable model of the user. This situation where the system has not yet gathered sufficient information to understand the user preferences is known as the cold start problem [2, 20, 48, 50]. It refers to the situation where a user enters the system for the first time as well as to the situation where the user has little interaction with the system and therefore has only evaluated a few items. A common way to learn about user preferences and overcome the cold start problem is to ask users to rate a few items [47, 49, 50]. 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 [47].

Several techniques have been proposed for determining the most informative items to be rated by a user. These techniques use strategies that are based on item entropy, item popularity, user personalization, and combinations of the above [47]. Recommending popular items is one of the solutions in cases where there is not enough prior information about user preferences [47]. The obvious disadvantage of such a solution is that the user gets the most expected recommendation for a product that has a high chance that he is already familiar with.

Sometimes a user may wish to explore new items offered by a recommender system. In this case, the user will cooperate with the system and willingly rate the suggested items. However, a user may not have the time or desire to rate a large number of items. Such users want to get recommendations without being too involved in the recommendation process. Requiring the user to rate items as a precondition for creating recommendations for him 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 [10, 11, 48]. A hybrid approach is described in [7] that uses a modeling mediation process which integrates partial user models produced by different types of recommender systems. This method uses four types of mediation: cross-user, cross-item, cross-context, and cross-representation.

A cross-technique mediation process between two recommender systems from the same application domain (movies) is studied in [6, 7]. 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 [6, 7].

A cross-domain recommender system is proposed in [59] where data from an online shopping domain is combined with information taken from an advertising platform using deep learning techniques. The users of the system are represented by their browsing records taken from the advertisement platform, while the items are represented through textual description. 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 [58]. 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, for example, how each rating contributes to improving the accuracy of user models.

In the next section, we describe a technique we proposed in [27] for building user models through hypercube graphs in a CB recommender system. It differs greatly from approaches based on deep learning and graph embedding. Roughly speaking, an iterative process builds a user model from scratch, when there are no previous ratings from that user, and then refines it with each additional rating made by that user. Experiments described in [27] indicate that this method produces good approximations of user preferences with very few ratings, and thus offers an interesting solution to the cold start problem.

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 [27] 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 $i \in \{1,\ldots,n\}$ such that $x_i \neq y_i$) equals 1. Items and user profiles are represented as vertices in Q_n (see Figure 3). More precisely:

- A vertex $\boldsymbol{v}^i = (v_1^i,\ldots,v_n^i)$ 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 $\boldsymbol{w}^u = (w_1^u,\ldots,w_n^u)$ 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.

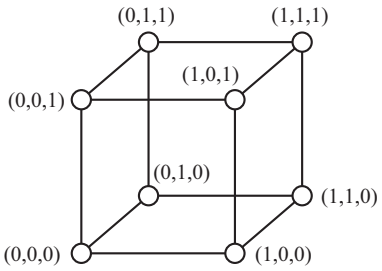

Figure 3: The hypercube Q_3.

For example, suppose that I is a restaurant dataset characterized by the following 12 attributes: Mexican food, Asian food, Traditional-American food, Southern food, Seafood, Café, Low Price,

Medium Price, High Price, Order Delivery, Order Takeout and Outdoor Seating. Then $\boldsymbol{v}^i = (010000010110)$ means that item $i \in I$ is a medium priced Asian restaurant which allows order delivery and order takeout and has no outdoor seating. If a user $u \in U$ likes low priced Mexican restaurants that allow outdoor seating, then $\boldsymbol{w}^u = (100000100001)$ describes the user preferences.

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*. For the above example, (100000100001) is the vertex that represents all low priced Mexican restaurants that allow outdoor seating, as well as all users of the recommender system that like low priced Mexican restaurants that allow outdoor seating.

Assume that a user $u \in U$ of a recommender system has indicated the number of attributes $a \in A$ he does not 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 profile 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 does not 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 [27] 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 $\delta_1, \ldots, \delta_{|I'|}$ 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 [27] 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 $w^u_a \neq v^i_a$. 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

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

In summary, the first algorithm described in [27] generates a vector $\boldsymbol{x}$ in Q_n with minimum value $f_1(\boldsymbol{x})$.

Variations of this algorithm are proposed in [27]. 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 does not 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 $w^u_j = -1$ if u does not like the jth attribute, $w^u_j = 0$ if u does not care about it, and $w^u_j = 1$ if u likes it. If $w^u_j = 0$, 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 Q_n that minimizes $f_2(\boldsymbol{x})$, where f_2 is obtained from f_1 by replacing d with d'.

As shown in [27], 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 [27] which show that few ratings are sufficient to generate a vertex $\boldsymbol{x}$ which is close to the user profile. 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 running the two algorithms mentioned in Section 3. We first focus on datasets from the restaurant domain which we used in [27] 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 [27] we extracted ten datasets from the yelp website (https://www.yelp.com/dataset), 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 does not 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.

Table 1: Description of the data sets used in our first experiments

		Number of rated items					average rating	standard deviation
u_i	n_i	1⋆	2⋆	3⋆	4⋆	5⋆		
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

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 [4]. 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 [1, 48]. The considered datasets are no exception. For

example, three restaurants rated by u_4 have the 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 [18].

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. The algorithm that minimizes f_1 will be called Algo1, while Algo2 is for the minimization of f_2. These two algorithms are called Model1 and Model3 in [27], 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 $\tau^{-1}(7) = 5 - \lfloor \frac{28}{20} \rceil = 4$.

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 second. When CPLEX was interrupted before completing the optimization, we report the best solution found. Experiments have shown that no noticeable improvement can be obtained with longer computing times, of the order of a minute or an hour.

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

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 Naïve Bayes (NB), Random Forest (RF), Support Vector Regression (SVR), Decision Tree (DT) and Neural Network

(NN), using the Scikit machine learning tool [44]. 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 the SVR method is the best with the smallest average prediction error of 0.5342, while the DT method is the worst with an error of 0.7050. The errors for Algo1 and Algo2 are 0.5964 and 0.5953, respectively, and the NB and RF methods have slightly better results.

As reported in Table 3, f-tests and t-tests indicate that the average prediction errors of Algo1, Algo2, NB and RF are not statistically different, while SVR is statistically 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 significant 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.

Table 3: Results of the statistical tests: + indicates a significant difference between the methods, while − indicates that the two methods are not significantly different.

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

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 4 and are also reported in Table 3, for $\ell = 1$ to 24, and for $\ell = 130$ to 150.

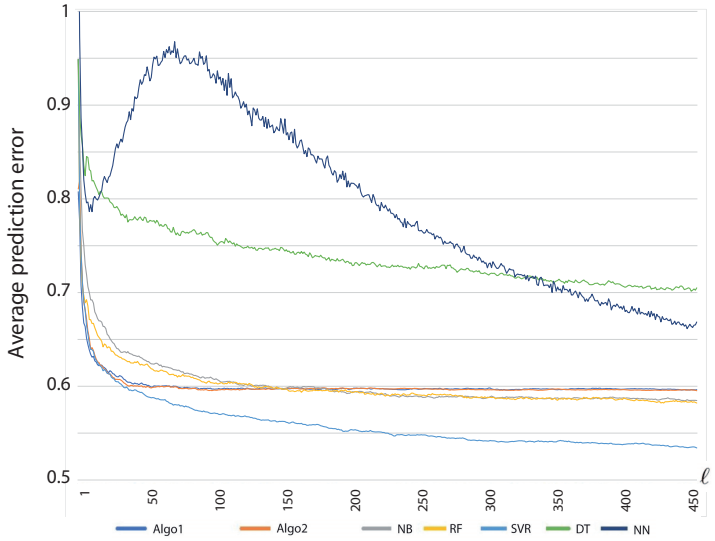

Figure 4: The impact of the size ℓ of the training set on the accuracy of each model, for $\ell = 1$ to 450.

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. To evaluate whether there is a significant difference between two methods, we used the standard f-tests and t-tests. When a method performs significantly better or worse than Algo1, we indicate it with a gray cell in Table 4. Best average errors are shown with bold characters.

Table 4: The average prediction error obtained for $\ell = 1$ to 24 and for $\ell = 130$ to 150. Bold numbers indicate the best results. Gray cells indicate a significant difference with the average prediction error produced by 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
23	0.6145	0.6133	0.6539	0.6421	**0.6133**	0.8006	0.8230
24	0.6167	0.6132	0.6510	0.6414	**0.6130**	0.7968	0.8245
				⋮			
130	0.5972	0.5970	0.6012	0.5998	**0.5656**	0.7454	0.8894
131	0.5970	0.5972	0.6010	0.5986	**0.5648**	0.7450	0.8961
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

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 significantly different from the results of RF (with 2 exceptions) or SVR. For $\ell \in \{17, \ldots, 450\}$, the SVR method got the 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.

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

ℓ	Similar error as Algo1	Smallest error	Second best error
$= 1$	Algo2	Algo1	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

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 4, 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 5.

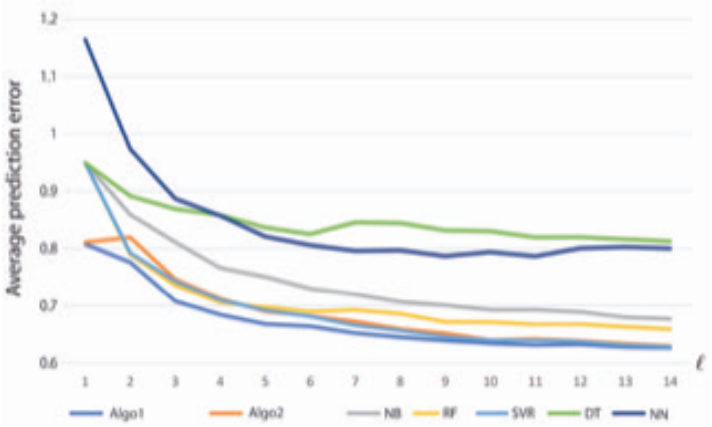

Figure 5: The impact of the size of the training set on the accuracy of each model, for $\ell = 1$ to 14.

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 [25]. 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 1,128 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.

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

		Number of rated items										average rating	standard deviation
u_i	n_i	0.5⋆	1.0⋆	1.5⋆	2.0⋆	2.5⋆	3.0⋆	3.5⋆	4.0⋆	4.5⋆	5.0⋆		
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

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 validation process, applying Algo1, Algo2 and the five other machine learning techniques. As shown in [27], 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 6 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 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.

To test Algo1 and Algo2 on a larger number of instances, we have extracted 1,000 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. 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

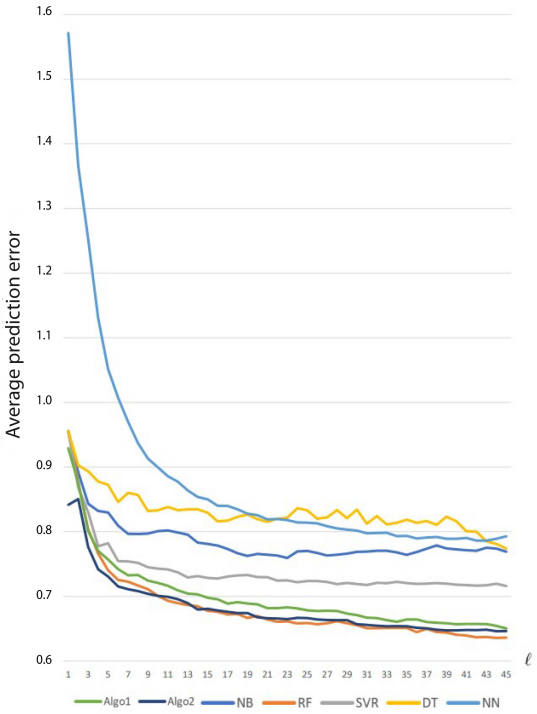

Figure 6: The impact of the size ℓ of the training set on the accuracy of each model, for $\ell = 1$ to 45 for 10 datasets taken from the MovieLens website.

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 7 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 significant difference between the results produced by Algo2 and RF. For $\ell = 1$ Algo2 is significantly better than the other 3 methods. Algo1 and SVR are statistically 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 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 [62]. However, these systems need a lot of data and therefore are not suitable for small datasets where classical machine learning techniques may perform well. This is especially true when considering personal CB recommender systems. Nevertheless, the cold start problem is still a challenge, as even classical machine learning techniques need enough examples for training. This becomes quite clear when looking at Figure 1 that shows that most

Table 7: The average prediction error obtained for $\ell = 1$ to 45 for 10 datasets taken from the MovieLens website. Bold numbers indicate the best results. Gray cells indicate a significant difference with the average prediction error produced by 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

users of a recommender system typically provide very few ratings, sometimes too few for enabling a system to build an accurate user model.

To address the cold start problem, we have proposed in [27] 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, 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 Figure 2).

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

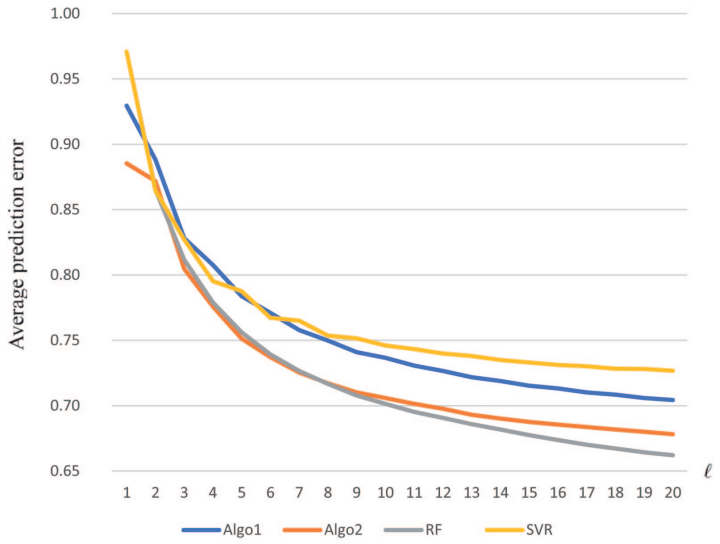

Figure 7: The impact of the size ℓ of the training set on the accuracy of each model, for $\ell = 1$ to 20 for 1,000 datasets taken from the MovieLens website.

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 statistical 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 statistical significant difference between them, where Algo1 and SVR got the second-best results.

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 [12]. The SVR method, on the other hand, suffers from high training complexity because finding a separating hyperplane requires solving a quadratic programming problem which is computationally expensive [15].

Table 9: The average prediction error obtained for $\ell = 1$ to 20 for 1,000 datasets taken from the MovieLens website. Bold numbers indicate the best results. Gray cells indicate a significant difference with the average prediction error produced by 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

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 [19, 59], 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.* Recommender systems collect and use large amounts of user data that may include sensitive and vulnerable information. Therefore, preserving user privacy is a challenge that needs to be addressed [28]. 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 does not require the knowledge of the ratings of other users. Many algorithms of recommender systems can be accessed through mobile devices, and therefore are highly available to the user [48]. A personal recommender system is proposed in [35, 36] which can be implemented on a personal device. It combines a user modeling server and a mediation module that enables the user to select the level of data sharing. 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.
- *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 $d(\boldsymbol{x}, \boldsymbol{v}^i) \neq d(\boldsymbol{y}, \boldsymbol{v}^i)$ [5]. As shown in [16], the vector of distances between a vertex $\boldsymbol{x}$ and the vertices of a resolving set unequivocally determines $\boldsymbol{x}$ in Q_n 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 $a \in A$. In other words, when a user of the system has rated a set of items which constitutes a resolving set of Q_n (where n is the number of attributes), it is possible to deduce his preferences [27]. As the resolving sets of hypercube graphs are of very small size [26], this approach can be used to solve the cold start problem. The size of a resolving set increases however with the number n of attributes, and it is therefore interesting to try to keep only those attributes which have the greatest influence on the ratings made by a user. This is what we intend to do in a future work. We will then encourage the user of a recommender system to

evaluate items that constitute a resolving set in order to know his preferences based on very few ratings.

References

[1] Adomavicius, G., Bauman, K., Tuzhilin, A., and Unger, M. Context-Aware Recommender Systems: From Foundations to Recent Developments Context-aware recommender systems. Springer US, New York, NY, 2022, pp. 211–250.

[2] Adomavicius, G., and Tuzhilin, A. Toward the next generation of recommender systems: a survey of the state-of-the-art and possible extensions. IEEE Transactions on Knowledge and Data Engineering 17, 6 (2005), 734–749.

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

[4] Amatriain, X., Pujol, J. M., and Oliver, N. I like it... i like it not: Evaluating user ratings noise in recommender systems. In User Modeling, Adaptation, and Personalization (Berlin, Heidelberg, 2009), G.-J. Houben, G. McCalla, F. Pianesi, and M. Zancanaro, Eds., Springer Berlin Heidelberg, pp. 247–258.

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

[6] Berkovsky, S., Kuflik, T., and Ricci, F. Cross-technique mediation of user models. In Adaptive Hypermedia and Adaptive Web-Based Systems (Berlin, Heidelberg, 2006), V. P. Wade, H. Ashman, and B. Smyth, Eds., Springer Berlin Heidelberg, pp. 21–30.

[7] Berkovsky, S., Kuflik, T., and Ricci, F. Mediation of user models for enhanced personalization in recommender systems. User Modelling and User-Adapted Interaction (2008), 245–286.

[8] Berkovsky, S., Kuflik, T., and Ricci, F. Cross-representation mediation of user models. User Modelling and User-Adapted Interaction 19, 1–2 (2009), 35–63.

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

[10] Burke, R. Hybrid recommender systems: Survey and experiments. User Modeling and User-Adapted Interaction 12 (2002), 331–370.

[11] Burke, R. Hybrid web recommender systems. In The Adaptive Web: Methods and Strategies of Web Personalization. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007, pp. 377–408.

[12] Buskirk, T. Surveying the forests and sampling the trees: An overview of classification and regression trees and random forests with applications in survey research. Survey Practice 11 (01 2018), 1–13.

[13] Cai, H., Zheng, V. W., and Chang, K. C.-C. A comprehensive survey of graph embedding: Problems, techniques, and applications. IEEE Transactions on Knowledge and Data Engineering 30 (2018), 1616–1637.

[14] Çano, E., and Morisio, M. Hybrid recommender systems: A systematic literature review. Intell. Data Anal. 21, 6 (jan 2017), 1487–1524.

[15] Cervantes, J., Li, X., and Yu, W. Svm classification for large data sets by considering models of classes distribution. In 2007 Sixth Mexican International Conference on Artificial Intelligence, Special Session (MICAI) (2007), pp. 51–60.

[16] Chartrand, G., Eroh, L., Johnson, M. A., and Oellermann, O. R. Resolvability in graphs and the metric dimension of a graph. Discrete Applied Mathematics 105, 1 (2000), 99–113.

[17] Chen, Y., Zhou, M., Zheng, Z., and Chen, D. Time-aware smart object recommendation in social internet of things. IEEE Internet of Things Journal 7, 3 (2020), 2014–2027.

[18] Dalvi, N., Kumar, R., and Pang, B. Para 'normal' activity: On the distribution of average ratings. Proceedings of the International AAAI Conference on Web and Social Media 7, 1 (2013), 110–119.

[19] Da'u, A., and Salim, N. Recommendation system based on deep learning methods: a systematic review and new directions. Artificial Intelligence Review 53 (2020), 2709–2748.

[20] de Gemmis, M., Lops, P., Musto, C., Narducci, F., and Semeraro, G. Semantics-aware content-based recommender systems. In Recommender Systems Handbook. Springer US, Boston, MA, 2015, pp. 119–159.

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

[22] Guan, X., Cheng, Z., He, X., Zhang, Y., Zhu, Z., Peng, Q., and Chua, T.-S. Attentive aspect modeling for review-aware recommendation. ACM Trans. Inf. Syst. 37, 3 (2019), 1–27. article no. 28.

[23] Gunawardana, A., Shani, G., and Yogev, S. Evaluating recommender systems. In Recommender Systems Handbook. Springer US, New York, NY, 2022, pp. 547–601.

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

[25] Harper, F. M., and Konstan, J. A. The movielens datasets: History and context. ACM Trans. Interact. Intell. Syst. 5, 4 (2016), 1–19.

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

[27] Hertz, A., Kuflik, T., and Tuval, N. Resolving sets and integer programs for recommender systems. Journal of Global Optimization 81 (2021), 153–178.

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

[29] Huang, Z., Chung, W., and Chen, H. A graph model for e-commerce recommender systems. J. Am. Soc. Inf. Sci. Technol. 55, 3 (2004), 259–274.

[30] Jackson, J. C., and Wimmer, K. New results for random walk learning. J. Mach. Learn. Res. 15, 1 (2014), 3655–3666.

[31] Jannach, D., Quadrana, M., and Cremonesi, P. Session-based recommender systems. In Recommender Systems Handbook. Springer US, New York, NY, 2022, pp. 301–334.

[32] Jiang, Z., Liu, H., Fu, B., Wu, Z., and Zhang, T. Recommendation in heterogeneous information networks based on generalized random walk model and bayesian personalized ranking. In Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining, WSDM '18 (2018), pp. 288–296.

[33] Kobsa, A. User modeling in dialog systems: Potentials and hazards. AI & Society 4, 3 (2001), 214–231.

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

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

[36] Kuflik, T., and Poteriaykina, K. User model on a key. In Proceedings of the 20th ACM Conference on Hypertext and Hypermedia, HT '09 (2009), pp. 371–372.

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

[38] Martínez, L., Barranco, M. J., Pérez, L. G., and Espinilla, M. A knowledge based recommender system with multigranular linguistic information. International Journal of Computational Intelligence Systems 1, 3 (2008), 225–236.

[39] Minkov, E., Kahanov, K., and Kuflik, T. Graph-based recommendation integrating rating history and domain knowledge: Application to on-site guidance of museum visitors. Journal of the Association for Information Science and Technology 68, 8 (2017), 1911–1924.

[40] Musto, C., Gemmis, M. d., Lops, P., Narducci, F., and Semeraro, G. Semantics and content-based recommendations. In Recommender Systems Handbook. Springer US, New York, NY, 2022, pp. 251–298.

[41] Neumann, M., Garnett, R., and Kersting, K. Coinciding walk kernels: Parallel absorbing random walks for learning with graphs and few labels. Journal of Machine Learning Research 29 (2013), 357–372.

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

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

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

[45] Portugal, I., Alencar, P., and Cowan, D. The use of machine learning algorithms in recommender systems: A systematic review. Expert Systems with Applications 97 (2018), 205–227.

[46] Qiu, R., Huang, Z., Li, J., and Yin, H. Exploiting cross-session information for session-based recommendation with graph neural networks. ACM Trans. Inf. Syst. 38, 3 (2020), 1–23.

[47] Rashid, A. M., Albert, I., Cosley, D., Lam, S. K., McNee, S. M., Konstan, J. A., and Riedl, J. 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), pp. 127 – 134.

[48] Ricci, F., Rokach, L., and Shapira, B. Recommender systems: Techniques, applications, and challenges. In Recommender Systems Handbook. Springer US, New York, NY, 2022, pp. 1–35.

[49] Rubens, N., Elahi, M., Sugiyama, M., and Kaplan, D. Active learning in recommender systems. In Recommender Systems Handbook. Springer US, Boston, MA, 2015, pp. 809–846.

[50] Schein, A. I., Popescul, A., Ungar, L. H., and Pennock, D. M. Methods and metrics for cold-start recommendations. In Proceedings of the 25th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR'02 (2002), pp. 253–260.

[51] Sinha, B. B., and Dhanalakshmi, R. Evolution of recommender paradigm optimization over time. J. King Saud Univ. Comput. Inf. Sci. 34, 4 (2022), 1047–1059.

[52] Tan, J., Wan, X., and Xiao, J. A neural network approach to quote recommendation in writings. In Proceedings of the 25th ACM International on Conference on Information and Knowledge Management, CIKM '16 (2016), pp. 65–74.

[53] Tiroshi, A., Kuflik, T., Berkovsky, S., and Kaafar, M. 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, pp. 407–454.

[54] Tuan, T. X., and Phuong, T. M. 3d convolutional networks for session-based recommendation with content features. In Proceedings of the Eleventh ACM Conference on Recommender Systems, RecSys '17 (2017), pp. 138–146.

[55] van den Oord, A., Dieleman, S., and Schrauwen, B. Deep content-based music recommendation. In Advances in Neural Information Processing Systems (2013), C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Weinberger, Eds., vol. 26, pp. 2643–2651.

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

[57] Vinagre, J., Jorge, A., and Gama, J. An overview on the exploitation of time in collaborative filtering. Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery 5 (2015), 195–215.

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

[59] Wang, H., Amagata, D., Makeawa, T., Hara, T., Hao, N., Yonekawa, K., and Kurokawa, M. A dnn-based cross-domain recommender system for alleviating cold-start problem in e-commerce. IEEE Open Journal of the Industrial Electronics Society 1 (2020), 194–206.

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

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

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

[63] Zhang, S., Yao, L., Sun, A., and Tay, Y. Deep learning based recommender system: A survey and new perspectives. ACM Comput. Surv. 52, 1 (2019), 1–38. article no. 5.

[64] Zhu, Z., Kim, J., Nguyen, T., Fenton, A., and Caverlee, J. 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), pp. 767–776.