Enhancing Rating Prediction Quality Through Improving the Accuracy of Detection of Shifts in Rating Practices

Dionisis Margaris[1] and Costas Vassilakis[2](✉)

[1] Department of Informatics and Telecommunications, University of Athens, Athens, Greece
margaris@di.uoa.gr

[2] Department of Informatics and Telecommunications, University of the Peloponnese, Tripoli, Greece
costas@uop.gr

Abstract. The most widely used similarity metrics in collaborative filtering, namely the Pearson Correlation and the Adjusted Cosine Similarity, adjust each individual rating by the mean of the ratings entered by the specific user, when computing similarities, due to the fact that users follow different rating practices, in the sense that some are stricter when rating items, while others are more lenient. However, a user's rating practices change over time, i.e. a user could start as lenient and subsequently become stricter or vice versa; hence by relying on a single mean value per user, we fail to follow such shifts in users' rating practices, leading to decreased rating prediction accuracy. In this work, we present a novel algorithm for calculating dynamic user averages, i.e. time-in-point averages that follow shifts in users' rating practices, and exploit them in both user-user and item-item collaborative filtering implementations. The proposed algorithm has been found to introduce significant gains in rating prediction accuracy, and outperforms other dynamic average computation approaches that are presented in the literature.

Keywords: Recommender systems · Collaborative filtering
User-user similarity · Item-item similarity · Dynamic average
Prediction accuracy · Ratings' timestamps

1 Introduction

Collaborative filtering (CF) computes personalized recommendations, by taking into account users' past likings and tastes, in the form of ratings entered in the CF rating database. User-user CF algorithms firstly identify people having similar tastes, by examining the resemblance of already entered ratings; for each user *u*, other users having highly similar tastes with *u* are designated as *u*'s nearest neighbors (NNs). Afterwards, in order to predict the rating that *u* would give to an item *i* that *u* has not reviewed yet, the ratings assigned to item *i* by *u*'s NNs are combined [1], under the assumption that users are highly likely to exhibit similar tastes in the future, if they have done so in the past as well [26, 30]. Analogous practices are followed in item-item

A. Hameurlain and R. Wagner (Eds.): TLDKS XXXVII, LNCS 10940, pp. 151–191, 2018.
https://doi.org/10.1007/978-3-662-57932-9_5

CF algorithms, where the first step is to locate items that are similarly rated by users. CF is the most successful and most applied technique in the design of recommender systems [3]. In order to measure similarity between users or items, the Pearson Correlation Coefficient and the Adjusted Cosine Similarity [4] (see also Sect. 3) are the most commonly used formulas in CF recommender systems. In this context, both the Pearson correlation coefficient and the Adjusted Cosine Similarity adjust the ratings of a user *u* by the mean value of all ratings entered by *u*, and the ratings of an item *i* by the mean value of all ratings for this item, respectively, so as to tackle the issue that some users may rate items higher than others or that some items may be rated higher than others. However, relying on a single, global mean value presumes that the users' marking practices remain constant over time; in practice though, it is possible that a user's marking practices change over time, i.e. a user could start off being strict and subsequently change to being lenient, or vice versa; similarly, an item could start as being high rated and subsequently begin receiving lower marks, due to general shift of interest (music or clothing trends, decline of interest for blockbuster movies or books etc.). For instance, according to the MovieLens 20M dataset [9, 10], the Titanic movie started off with an average of 4.30/5 in 1997, dropping to 3.06 in 2005, and finally climbing back to 3.24 in 2015.

Similar situations arise for users: for example, consider that a user initially grades *Tudors* (http://www.imdb.com/title/tt0758790/), which is the first historic period drama series of high quality that she rates; being enthusiastic with the series, she enters a rating of 10. Subsequently, the same user rates *Game of Thrones* (http://www.imdb.com/title/tt0944947/), which she finds superb and better than *Tudors*, giving it the highest available mark, i.e. 10. Finally, the user watches a few episodes from the show *Vikings* (http://www.imdb.com/title/tt2306299), and grades this series with an 8. While both *Tudors* and *Game of Thrones* have been equally rated by the user, this does not necessarily reflect the fact that she considers them of equal quality; similarly, the fact that *Vikings* got a lower grade than *Tudors*, does not necessarily mean that she considers it of inferior quality: the user's rating criteria and practices have simply evolved, along with her experiences on historic period drama series.

Insofar, while many efforts have been made to improve the CF prediction accuracy, and the aspect of changes in users' interests has been extensively studied (Gama et al. [24] provide a comprehensive review), the issue of shifts in rating practices has not received adequate attention. Margaris and Vassilakis [33] introduce the concept of dynamic user rating averages which follow the users' marking practices shifts and present two alternative algorithms for computing a user's dynamic averages. These algorithms are validated in the context of user-user CF, and have been found to achieve better rating prediction accuracy than the plain CF algorithm, using the Pearson correlation similarity metric.

In this paper, we extend the work in [33] by introducing a new dynamic average computation algorithm, namely DA_{next}, which is capable of better following the users' marking practices shifts, leading to improved prediction accuracy, as compared to the two dynamic average algorithms presented in [33]. This improvement is consistent under both user-user CF implementations and item-item CF implementations, where similarities are measured using the Pearson Correlation and the Adjusted Cosine Similarity respectively. To validate our approach, we present an extensive comparative evaluation among (i) the proposed algorithm, (ii) the two approaches proposed in [33] and (iii) the classic static average (unique mean value), considering both the user-user and the item-item CF implementations.

The proposed algorithm is based on the exploitation of timestamp information which is associated with ratings; hence in this work, we use the Amazon datasets [7, 8], the MovieLens datasets [9, 10], the Netflix dataset [11] and the Ciao dataset [51], which contain timestamps. It is worth noting that the proposed algorithm can be combined with other techniques that have been proposed for either improving prediction accuracy in CF-based systems, including consideration of social network data (e.g. [14, 25, 29]), location data (e.g. [34, 35]) and pruning of old user ratings (e.g. [12, 38]), or techniques for speeding up prediction computation time, such as clustering (e.g. [36, 37, 40]).

The rest of the paper is structured as follows: Sect. 2 overviews related work, while Sect. 3 presents the proposed algorithm, together with the algorithms presented in [33], for self-containment purposes. Section 4 evaluates the proposed algorithm using the aforementioned datasets and finally, Sect. 5 concludes the paper and outlines future work.

2 Related Work

The accuracy of CF-based systems is a topic that has attracted considerable research efforts. Koren [15] proposes a new neighborhood-based model, which is based on formally optimizing a global cost function and leads to improved prediction accuracy, while maintaining merits of the neighborhood approach such as explainability of predictions and ability to handle new ratings (or new users) without retraining the model. In addition, he suggests a factorized version of the neighborhood model, which improves its computational complexity while retaining prediction accuracy. Liu et al. [18] present a new user similarity model to improve the recommendation performance when only few ratings are available to calculate the similarities for each user. The model considers the local context information of user ratings, as well as the global preference of user behavior. Ramezani et al. [39] propose a method to find the neighbor users based on the users' interest patterns in order to overcome challenges like sparsity and computational issues, following the idea that users who are interested in the same set of items share similar interest patterns, therefore, the non-redundant item subspaces are extracted to indicate the different patterns of interest and then, a user's tree structure is created based on the patterns he has in common with the active user.

Research has shown that exploiting time in the rating prediction computation can improve prediction accuracy, due to concept drift; concept drift is the phenomenon when the relation between the input data and the target variable changes over time [24]. Change of interests [5, 24] is a typical example of concept drift. To this end, Zliobaite et al. [22] develop an intelligent approach for sales prediction, which uses a mechanism for model switching, depending on the sales behavior of a product. This research presents an intelligent two level sales prediction approach that switches the predictors depending on the properties of the historical sales. This approach is shown to achieve better results as compared to both a baseline predictor and an ensemble of predictors.

Ang et al. [21] address the problem of adaptation when external changes are asynchronous, by developing an ensemble approach, called PINE, which combines reactive adaptation via drift detection, and proactive handling of upcoming changes via early warning and adaptation across the peers. In addition, PINE is parameter-insensitive and incurs less communication cost while achieving better accuracy.

Elwell and Polikar [20] tackle the issue of concept drift in the context of online learning, introducing a batch-based ensemble of classifiers, called Learn++.NSE, where NSE stands for Non-Stationary Environments. Learn++.NSE learns from consecutive batches of data without making any assumptions on the nature or rate of drift; it can learn from such environments that experience constant or variable rate of drift, addition or deletion of concept classes, as well as cyclical drift. The algorithm learns incrementally, as do other members of the Learn++ family of algorithms, that is, without requiring access to previously seen data. Learn++.NSE trains one new classifier for each batch of data it receives, and combines these classifiers using a dynamically weighted majority voting. The algorithm is evaluated on several synthetic datasets designed to simulate a variety of nonstationary environments, as well as a real-world weather prediction dataset. Minku et al. [19] present a new categorization for concept drift, separating drifts according to different criteria into mutually exclusive and non-heterogeneous categories. Moreover, they present a diversity analysis in the presence of different types of drifts and it shows that, before the drift, ensembles with less diversity obtain lower test errors. Nishida and Yamauchi [17] have developed a detection method that includes an online classifier and monitors its prediction errors during the learning process, which uses a statistical test of equal proportions. Experimental results showed that this method performed well in detecting the concept drift in five synthetic datasets that contained various types of concept drift.

Vaz et al. [16] propose an adaptation of the item-based CF algorithm to incorporate rating age influence in predictions. It considers ratings in two dimensions: the active user ratings and the community ratings, and it inserts a time weight, which gives more relevance to more recent ratings than to older ones, both in the similarity calculation and in the rating prediction equation.

Dror et al. [2] consider the temporal dimension in the context of recommender systems by capturing different temporal dynamics of music ratings, along with information from the taxonomy of music-related items; both these dimensions are exploited by a rich bias model. The method proposed in this work is applied on a sparse, large-scale dataset, and the particular characteristics of the dataset are extracted and utilized. Liu et al. [13] present a social temporal collaborative ranking model that can simultaneously achieve three objectives: (1) the combination of both explicit and implicit user feedback, (2) support for time awareness using an expressive sequential matrix factorization model and a temporal smoothness regularization function to tackle overfitting, and (3) support for social network awareness by incorporating a network regularization term. Dias and Fonseca [31] explore the usage of temporal context and session diversity in session-based CF techniques for music recommendation. They compare two techniques to capture the users' listening patterns over time: one explicitly extracts temporal properties and session diversity, to group and compare the similarity of sessions, the other uses a generative topic modeling algorithm, which is able to implicitly model temporal patterns. Results reveal that the inclusion of temporal information, either explicitly or implicitly, significantly increases the accuracy of the recommendation, as compared to the traditional session-based CF.

Li et al. [41] study the problem of predicting the popularity of social multimedia content embedded in short microblog messages, exploiting the idea of concept drift to capture the phenomenon that through the social networks' "re-share" feature, the popularity of a multimedia item may be revived or evolve. They model the social multimedia item popularity prediction problem using a classification-based approach which is used for two sub-tasks, namely re-share classification and popularity score classification. Furthermore, they develop a concept drift-based popularity predictor by ensembling multiple trained classifiers from social multimedia instances in different time intervals.

Lu et al. [42] present a novel evolutionary view of user's profile by proposing a Collaborative Evolution (CE) model, which learns the evolution of user's profiles through the sparse historical data in recommender systems and outputs the prospective user profile of the future. Kangasrääsiö et al. [43] formulate a Bayesian regression model for predicting the accuracy of each individual user feedback and thus find outliers in the feedback data set. Additionally, they introduce a timeline interface that visualizes the feedback history to the user and provides her with suggestions on which past feedback is likely in need of adjustment. This interface also allows the user to adjust the feedback accuracy inferences made by the model. The proposed modeling technique, combined with the timeline interface, makes it easier for the users to notice and correct mistakes in their feedback, and to discover new items.

Lo et al. [52] address the issue of tracking concept drift in individual user preferences; in this context they develop a Temporal Matrix Factorization approach (TMF) for tracking concept drift in each individual user latent vector. To this end, a time series of rating matrices is initially constructed from the ratings database; subsequently this time series is used to capture the concept drift dynamics for each individual user; and finally, the captured dynamics are taken into account in the rating prediction computation phase. Cheng et al. [53] propose the ISCF (interest sequences CF), a recommendation method based on users' interest sequences; interest sequences are first detected from the ratings, and are subsequently used to refine similarity metrics between users, thus taking into account dynamic evolution patterns of users' preferences.

However, none of the above mentioned works considers the issue of shifts in the users' rating practices. This issue has only recently received some attention: Margaris and Vassilakis [33] introduce and exploit the concept of dynamic user rating averages which follow the users' marking practices shifts. Furthermore, they present two alternative algorithms for computing a user's dynamic averages and perform a comparative evaluation in the context of a user-user CF implementation. The results of this evaluation show that the dynamic average-based algorithms exhibit better performance than the plain CF algorithm in terms of rating prediction accuracy, at the expense of a small to tolerable drop in coverage.

Interestingly, fuzzy recommender systems (FRS) [50] introduce the concept of fuzzy user context in the process of rating prediction and recommendation formulation. Under this approach, each rating entered by a user is associated with a particular context element through a fuzzy membership function. The FRS approach could be exploited for accommodating user rating practices as a specific rating criterion; subsequently the criterion would be used in the calculation of fuzzy similarity degree between users and finally be incorporated in the final rating prediction. To implement this approach would necessitate however a concrete, automated method for assigning strictness labels to individual user ratings, the definition of an appropriate membership & utility functions, and the evaluation of the overall system performance. To our knowledge, no FRS has been reported in the literature to accommodate these features.

This paper extends the work presented in [33] by (1) introducing a novel algorithm for dynamic average computation, which is able to follow the users' shifts in rating practices more accurately and (2) validating its performance against widely used datasets with diverse characteristics, exploring both the user-user and item-item CF implementations. The DA_{next} algorithm introduced in this paper is based on the rationale that the rating practices of a user at some time point is formulated according to the experiences she has amassed up to that time point and therefore it is more accurately reflected by her subsequent ratings, which are influenced by the same (and some additional) experiences. On the other hand, the $DA_{vicinity}$ algorithm introduced in [33] assumes that the user's ratings are mostly affected by temporally constrained factors, such as user mood, while the $DA_{previous}$ algorithm, also presented in [33], assumes that the rating-related behavior of a user at a certain time point is better estimated by considering the user's behavior up to that time point. While it is also possible that the ratings entered by a user during some period are affected by her mood [47], which would favor the $DA_{vicinity}$ algorithm, the results indicate that the effect of the amassed experiences is stronger than the effect of mood. Further qualitative evaluation on this subject is required, and this is envisioned as part of our future work.

The newly introduced algorithm has been found to provide more accurate rating predictions, by better capturing the shifts in users' rating practices.

3 Exploiting Ratings' Timestamps in Users Dynamic Average Configuration

In CF, predictions for a user *X* are computed based on a set of users which have rated items similarly with *X*; this set of users is termed "near neighbors of *X*" (X's NNs). The predominant similarity metric used in CF-based systems is the Pearson correlation metric [3], where the similarity between two users X and Y is expressed as:

$$Pearson\_sim(X,Y) = \frac{\sum_{i \in I_X \cap I_Y} \left(R_{X,i} - \overline{R_X}\right) * \left(R_{Y,i} - \overline{R_Y}\right)}{\sqrt{\sum_{i \in I_X \cap I_Y} \left(R_{X,i} - \overline{R_X}\right)^2} * \sqrt{\sum_{i \in I_X \cap I_Y} \left(R_{Y,i} - \overline{R_Y}\right)^2}} \quad (1)$$

where *i* ranges over items that have been rated by both *X* and *Y* and $\overline{R_X}$ (resp. $\overline{R_Y}$) is the mean value of the ratings entered by X (resp. Y); as noted above, the Pearson correlation formula uses a "global" mean value. The algorithms presented in this section target the computation of $\overline{R_X}$ (resp. $\overline{R_Y}$), aiming to substitute the global average, which is insensitive to shifts in rating practices, by an average that is tailored to the time period that $R_{X,i}$ (resp. $R_{Y,i}$) was entered. When a dynamic average computation algorithm DA_{Alg} is employed, the above formula is modified as:

$$Pearson\_sim(X,Y) = \frac{\sum_{i \in I_X \cap I_Y} \left(R_{X,i} - DA_{Alg}\left(R_{X,i}\right)\right) * \left(R_{Y,i} - DA_{Alg}\left(R_{Y,i}\right)\right)}{\sqrt{\sum_{i \in I_X \cap I_Y} \left(R_{X,i} - DA_{Alg}\left(R_{X,i}\right)\right)^2} * \sqrt{\sum_{i \in I_X \cap I_Y} \left(R_{Y,i} - DA_{Alg}\left(R_{Y,i}\right)\right)^2}} \tag{2}$$

Similarly, for item-item CF, the Adjusted Cosine Similarity (which is preferred against the basic cosine similarity metric, since it takes into account the differences in rating scale between different users [46]) is modified as:

$$Adj\_\cos\_sim(i,j) = \frac{\sum_{u \in U} \left(R_{u,i} - DA_{Alg}\left(R_{u,i}\right)\right) * \left(R_{u,j} - DA_{Alg}\left(R_{u,j}\right)\right)}{\sqrt{\sum_{u \in U} \left(R_{u,i} - DA_{Alg}\left(R_{u,i}\right)\right)^2} * \sqrt{\sum_{u \in U} \left(R_{u,j} - DA_{Alg}\left(R_{u,j}\right)\right)^2}} \tag{3}$$

where u ranges over the users that have rated both i and j; again $DA_{Alg}\left(R_{u,i}\right)$ denotes the dynamic average of user u at the time that $DA_{Alg}\left(R_{u,i}\right)$ was submitted.

While other similarity metrics, such as Euclidian distance [49], Manhattan Distance [49], Spearman's coefficient [4], Kendall's Tau [4] etc. have been used in recommender systems, in this paper we will confine ourselves to examining the Pearson similarity metric for the user-user strategy and the adjusted cosine similarity metric for the item-item strategy. This is due to the fact that the relevant formulas readily use the average values of the users' and items' ratings, hence the substitution of the global user's or item's average by the rating-specific dynamic average is a natural extension, while other similarity metrics do not adjust ratings according to the global average. This is also the case with the very promising matrix factorization technique [32]. In our future work, we plan to investigate how dynamic averages can be integrated into the above mentioned methods.

In the rest of this section we present the proposed technique for computing the dynamic user averages. For completeness purposes, we will also describe the relevant techniques described in [33], which are also used as yardsticks in the performance evaluation section.

3.1 The Proposed Algorithm

Under the proposed approach for computing dynamic averages, a separate average for each rating is calculated and stored. The algorithm for computing the dynamic average proposed in this paper takes into account only the ratings that have been submitted *after* the rating for which the dynamic average is submitted. Effectively, this algorithm is based on the assumption that, when considering a particular rating r, ratings that have been entered after r reflect more accurately the user's strictness at the time point that r was entered. Under this approach for computing dynamic averages, each user rating $r_{u,x}$ is coupled with its own dynamic average $DA_{next}(r_{u,x})$ which is computed as shown in Eq. 4:

$$DA_{next}\left(r_{u,x}\right) = \frac{\sum_{r \in Ratings(u) \, \wedge \, t(r) > t\left(r_{u,x}\right)} r}{\left| r : r \in Ratings(u) \, \wedge \, t(r) > t\left(r_{u,x}\right) \right|} \tag{4}$$

The pseudocode for the computation of the dynamic averages under the proposed algorithm is illustrated in Listing 1.

```
PROCEDURE bootstrapDynamicAverages(ratingsDB)
// Input: ratings database containing all users' ratings
// Output: the ratings database is complemented with
//   the dynamic averages of the ratings. For each rating,
//   the number of subsequent ratings is also stored to
//   facilitate the update of dynamic averages

FOREACH user u ∈ users(ratingsDB)
  ru = retrieveAllUserRatings(u, ratingsDB)
  ru = sort ru by rating timestamp in asc order
  calculateDAnext(ru)
END FOR
END PROCEDURE

PROCEDURE calculateDAnext(ratingList)
// Input: a list of ratings in ascending temporal order.
// Output: the dynamic averages of all the ratings in the
// input list have been caclulated

// the last rating has no next, so its dynamic average
// defaults to the rating itself
numRatings = count(ratingList)
ratingList[numRatings].dynamicAVG = ratingList[numRatings].rating

sumOfNextRatings = 0
FOR i = numRatings -1 DOWNTO 1 STEP -1
  sumOfNextRatings += ratingList[i + 1].rating
  ratingList[i].dynamicAVG = (sumOfNextRatings / (numRatings - i))
END FOR
END PROCEDURE
```

Listing 1. Pseudocode for the computation of the dynamic averages in the ratings database

After the dynamic averages have been computed as illustrated in Listing 1, the Pearson similarity between two users X and Y can be computed as shown in Listing 2, which implements the formula given in Eq. 2. The computation between each pair of users can be done while the algorithm bootstraps, and the cached similarities can be used thereafter for rating prediction, as in the typical case of user-user CF-based systems.

```
FUNCTION DA_BasedPearsonSimilarity(RatingsDB, X, Y)
// Input: ratings database containing all users' ratings and the
// identities of the users
// Output: Pearson similarity metric

ratingsX = retrieveAllUserRatings(X, RatingsDB)
ratingsY = retrieveAllUserRatings(Y, RatingsDB)
// find ids of items rated by both users
commonItemIDs = commonItems(ratingsX, ratingsY)
pearsonNominator = 0
pearsonDenomX = 0
pearsonDenomY = 0
FOREACH itemID in commonItemIDs
  ratingX = getRatingByItemID(ratingsX, itemId)
  ratingY = getRatingByItemID(ratingsX, itemId)
  pearsonNominator += (ratingX.rating - ratingX.dynamicAVG) *
                      (ratingY.rating - ratingY.dynamicAVG)
  pearsonDenomX += pow((ratingX.rating - ratingX.dynamicAVG), 2)
  pearsonDenomY += pow((ratingY.rating - ratingY.dynamicAVG), 2)
END FOR
RETURN pearsonNominator / (sqrt(pearsonDenomX) *
                            sqrt(pearsonDenomY))
END FUNCTION
```

Listing 2. Pseudocode for the computation of the Pearson similarity between two users, considering the dynamic averages

When a new rating is entered in the database by some user *u*, the denominator of Eq. 4 changes for all ratings *r* that have been entered by the particular user, therefore the dynamic averages for all ratings entered by *u* must be recalculated. This will in turn trigger the recalculation of all similarities between *u* and other users in a user-user CF implementation (cf. Eq. 2) or the recalculation of all similarities between items that *u* has rated and other items in an item-item CF implementation (cf. Eq. 3). The relevant performance implications are discussed in Subsect. 4.10, together with the memory and secondary storage requirements of the algorithm.

```
PROCEDURE addRating(ratingsDB, user, item, rating)
// add a new rating in the ratings database

// INPUT: rating database, the user that gave the rating,
// the item on which the rating was given and the rating
// value
// Output: updated rating database with the user's new
// dynamic averages
// append rating to ratings db; this is positioned last, main-
taining
// the temporal sort order within the ratings of each user
// that has been established in the algorithm bootstrap
appendRating(ratingsDB, new Rating(user, item, rating))
ru = retrieveAllUserRatings(user, ratingsDB)
// recalculate the user's dynamic averages
calculateDAnext(ru)
END PROCEDURE
```

Listing 3. Pseudocode for inserting a new rating into the ratings database

One issue that is worth discussing in the dynamic average computation procedure described above is that recent ratings have only few next ones, therefore the dynamic average computed for these ratings may be skewed. While this is true, it has to be noted that the most recent ratings of each user *u*, where this skew appears, are only a small fraction of the items that *u* has in common with her near neighbors (in our experiments, less than 3.6% of the computations for evaluating similarities between users involved the last four ratings of either *u* or *u*'s near neighbors), with the rest of the computations being based on previous ratings that have at least four next ratings. In this respect, the effect of this skew is small. In order to further improve the effectiveness of the algorithm and minimize skew, variations of the algorithm may be introduced, which would e.g. consider a number of *past* ratings in the dynamic average computation or use the global average when an adequate number of more recent ratings is not available. Further elaboration and experimentation on this aspect is required, and this is considered part of our future work.

3.2 Existing Dynamic Average Algorithms

In [33], two algorithms for computing dynamic user averages were proposed, namely (a) the dynamic average based on the temporal vicinity of the ratings, which will be denoted as $DA_{vicinity}$ and (b) the dynamic average based only on previous ratings, which will be denoted as $DA_{previous}$. While Margaris and Vassilakis [33] describe their application only in a user-user CF scenario, these algorithms can be also directly applied in an item-item CF implementation, by using the corresponding dynamic averages in Eq. 3. In the next subsections, we briefly present these algorithms for completeness purposes.

Computing the Dynamic Average Based on the Temporal Vicinity of the Ratings. According to the $DA_{vicinity}$ algorithm, when computing the dynamic average $DA_{vicinity}(r)$ for a rating *r*, each user rating *r'* posted by the same user is assigned a weight on the basis of its temporal vicinity to *r*: ratings that have been entered temporally close to *r*are assigned higher weights, and as temporal distance increases, the weights decrease. This approach is based on the rationale that user ratings that are temporally distant to *R* may not accurately reflect the user's strictness at that particular time, while ratings that are temporally close to *r* form a better basis for deriving user strictness for the time period that *r* was entered.

In more detail, for rating $r_{u,x}$ of item *x* by user *u*, which has been entered at $t(r_{u,x})$, the weight $w_{u,x}(r_{u,i})$ of a rating $r_{u,i}$ is computed using the standard normalization function presented in [27]:

$$w_{u,x}\left(r_{u,i}\right) = 1 - \frac{\left|t\left(r_{u,x}\right) - t\left(r_{u,i}\right)\right|}{\max\limits_{i \in Ratings(u)}\left(\mathrm{t}\left(r_{u,i}\right)\right) - \min\limits_{i \in Ratings(u)}\left(\mathrm{t}\left(r_{u,i}\right)\right)} \tag{5}$$

where $t(r_{u,i})$ is the timestamp of rating $r_{u,i}$, whereas $\min\limits_{i \in Ratings(u)}\left(\mathrm{t}\left(r_{u,i}\right)\right)$ and $\max\limits_{i \in Ratings(u)}\left(\mathrm{t}\left(r_{u,i}\right)\right)$ denote the minimum and the maximum timestamp in the database among ratings entered by user *u*, respectively.

Finally, the dynamic average associated to rating $r_{u,x}$ is computed using the formula:

$$DA_{vicinity}\left(r_{u,x}\right) = \frac{\sum_{r \in Ratings(u)} w_{u,x}(r) * r}{\sum_{r \in Ratings(u)} w_{u,x}(r)} \tag{6}$$

Computing the Dynamic Average Based only on Previous Ratings. Under this approach for computing dynamic averages, again each user rating $r_{u,x}$ is coupled with its own average $DA_{previous}(r_{u,x})$. When computing this average, only ratings entered by the same user (*u*) prior to $r_{u,x}$ are taken into account; formally this approach is expressed by Eq. 7:

$$DA_{previous}\left(r_{u,x}\right) = \frac{\sum_{r \in Ratings(u) \bigwedge t(r) < t\left(r_{u,x}\right)} r}{\left|r : r \in Ratings(u) \bigwedge t(r) \langle t\left(r_{u,x}\right)\right|} \tag{7}$$

In Eq. 7, the denominator corresponds to the number of ratings that have been entered by user *u* prior to rating $r_{u,x}$, i.e. the rating for which the dynamic average $DA_{previous}\left(r_{u,x}\right)$ is calculated. This approach is based on the rationale that all past behaviour of the user is equally important in estimating her rating practices.

4 Performance Evaluation

In this section, we report on our experiments through which we compared the proposed algorithm with the $DA_{vicinity}$ and $DA_{previous}$ algorithms presented in [33], as well as the plain CF algorithm. We decided to consider in our evaluation both algorithms presented in [33] due to the following reasons:

1. the evaluation presented in [33] targets only the user-user CF approach, while in this paper we examine both the user-user and item-item CF approaches; hence, both the $DA_{vicinity}$ and the $DA_{previous}$ algorithms should be tested in order to evaluate their effectiveness in the item-item CF approach.
2. the comparison performed in [33] did not designate a clear winner between the $DA_{vicinity}$ and the $DA_{previous}$ algorithms; even though the $DA_{previous}$ algorithm outperforms the $DA_{vicinity}$ algorithm regarding prediction accuracy in most cases, there is a tie between the algorithms when they are applied in the MovieLens 100K dataset, while in the Netflix dataset the $DA_{vicinity}$ algorithm has been found to produce more accurate recommendations than the $DA_{previous}$ algorithm.

In this comparison we consider the following aspects:

1. prediction accuracy; for this comparison, we used two well-established error metrics, namely the mean absolute error (MAE) metric, as well as the Root Mean Squared Error (RMSE) that 'punishes' big mistakes more severely. RMSE was used in the Netflix competition [11],
2. the coverage of the algorithm, i.e. the percentage of the cases for which a prediction can be computed and
3. the probability that an algorithm computes the correct user rating. Since user ratings are typically integer numbers, while predictions are calculated as real numbers, for comparing the prediction to the actual user rating we round the prediction to the nearest integer. This is analogous to the practice used in the Netflix Competition [11].

To compute the MAE, the RMSE and the probability to compute the correct prediction, we employed the following techniques:

1. the standard "hide one" technique [30], which is extensively used in recommender systems research; each time, we hid a random rating in the database and then predicted its value based on the ratings of other non-hidden items. For each user, this procedure was executed for 10 randomly selected ratings entered by that particular user.
2. each time, we hid the last rating only from each user, and then predicted its value based on the ratings of other non-hidden items. One prediction for each user was formulated.
3. dropping the last rating from every user, and then applying the technique listed in item 2 above in the remaining dataset.

In all cases, the computation of the MAE, the RMSE and the correct prediction probability was performed considering all users in the database. All results were in close agreement (MAE: ±.0.005; RMSE: ±.0.008; correct predictions: ±.0.2%; % coverage: ±.0.4%), therefore in the rest of the paper we present only the results obtained from the standard "hide one" technique.

Regarding the algorithms presented in the related work section, except for those presented in [33], these are not directly comparable to our approach because they are designed to handle different phenomena, and more specifically concept drift (i.e. the change in users' interests), significance decay of old ratings and session identification. Nevertheless, to provide some insight on the magnitude of the improvement that can be achieved by different approaches that take into account temporal dynamics, we compare the performance of the $\mathrm{DA}_{\mathrm{next}}$ algorithm against the following algorithms sourced from the literature:

1. from the category of forgetting algorithms, i.e. algorithms that decay the importance of old-aged ratings, we compare DA_{next} with the work of Vaz et al. [16];
2. from the set of algorithms targeting interest shift detection and exploitation, we compare DA_{next} against the ISCF algorithm [53];
3. from the domain of temporally-aware session-based algorithms, we compare DA_{next} with the work of Dror et al. [2], which examines the influence of the drifting effect in short-lived music listening sessions; and
4. from the category of temporally-aware matrix factorization algorithms, we compare DA_{next} against the algorithm proposed by Lo et al. [52], which tracks and exploits concept drift in each individual user latent vector.

The algorithms employing dynamic averages may exhibit different coverage, since the introduction of dynamic averages modifies the user-to-user and item-to-item similarity metrics, and henceforth users or items that are deemed "similar" when using the plain CF algorithm (i.e. when their standard Pearson or Adjusted Cosine similarity surpasses a threshold) may be deemed "not similar" when using the dynamic average-aware Pearson similarity or Adjusted Cosine Similarity, or vice versa. Under this condition, some users that are characterized as "grey sheep" [6] when using the plain CF algorithm (i.e. do not have enough near neighbours for a recommendation to be computed) may gain enough neighbours when using a dynamic average-based algorithm, thus increasing coverage; conversely some users for which a recommendation was computed using the plain CF algorithm may become "grey sheep" when using a dynamic average-based algorithm, in which case coverage decreases. An analogous phenomenon also appears in an item-item CF implementation.

For our experiments we used a machine equipped with six Intel Xeon E7 - 4830 @ 2.13 GHz CPUs, 256 GB of RAM and one 900 GB HDD with a transfer rate of 200 MBps, which hosted the datasets and ran the recommendation algorithms.

In the following paragraphs, we report on our experiments regarding ten datasets. Five of these datasets are obtained from Amazon [7, 8], three from MovieLens [9, 10] one from Netflix [11], while the last dataset is sourced from Ciao, a product review site, where users can post their experiences with products or services (the site, dvd.ciao.co.uk, has ceased its operations, however the datasets crawled from it still exist and are used in CF research). These ten datasets used in our experiments (a) contain reliable timestamps (most of the ratings within each dataset have been entered in real rating time and not in a batch mode), (b) are up to date (published between 1998 and 2016),

(c) are widely used as benchmarking datasets in CF research and (d) vary with respect to type of dataset (movies, music, books, videogames and automotive) and size (from 1 MB to 4.7 GB). The basic properties of these datasets are summarized in Table 1.

In each dataset, users initially having less than 10 ratings were dropped, since users with few ratings are known to exhibit low accuracy in predictions computed for them [26]. This procedure did not affect the MovieLens and the NetFlix datasets, because these datasets contain only users that have rated 20 items or more. Furthermore, we detected cases where for a particular user all her ratings' timestamps were almost identical (i.e. the difference between the minimum and maximum timestamp was less than 5 s). These users were dropped as well, since this timestamp distribution indicated that the ratings were entered in a batch mode, hence the assigned timestamps are not representative of the actual time that these ratings were given by the users.

Table 1. Datasets summary

Dataset name	#users	#ratings	#items	Avg. #ratings/user	DB size (in text format)
Amazon "Video-games" [7, 8]	8.1K	157K	50K	19.6	3.8 MB
Amazon "CDs and Vinyl" [7, 8]	41K	1,300K	486K	31.5	32 MB
Amazon "Movies and TV" [7, 8]	46K	1,300K	134K	29.0	31 MB
Amazon "Books" [7, 8]	295K	8,700K	2,330K	29.4	227 MB
Amazon "Automotive" [7, 8]	7.3K	113K	65K	15.5	2.6 MB
MovieLens "Old 100K Dataset" [9, 10]	0.94K	100K	1.68K	106.0	2.04 MB
MovieLens "Latest-20M","recommended for new research" [9, 10]	138K	20,000K	27K	145	486 MB
MovieLens "Latest 100K", "Recommended for education and development" (small) [9, 10]	0.7K	100K	9K	143	2.19 MB
NetFlix competition [11]	480K	96,000K	17.7K	200	4,700 MB
Ciao [51]	1.1K	40K	16K	36.3	1 MB

In the following paragraphs, we report on our findings regarding the experiments described above, using both a user-user CF implementation, which employs the standard Pearson correlation coefficient for measuring user similarity, and an item-item CF implementation, which employs the Adjusted Cosine Similarity for measuring item similarity.

4.1 The Amazon "Videogames" Dataset

The results obtained from the Amazon "Videogames" dataset, are depicted in Table 2. Column "% coverage" corresponds to the percentage of cases for which the algorithm could compute predictions, or –equivalently– when the number of near neighbors computed using the algorithm's similarity metric was adequate [28] to formulate a rating prediction.

We can observe that under both CF implementations (user-user and item-item), the DA_{next} algorithm achieves the best results regarding prediction accuracy. In more detail, under the user-user CF implementation, the DA_{next} algorithm achieves an improvement in MAE of 1.6% against the runner up, which is the $DA_{previous}$ algorithm and a 7.3% improvement against the plain CF algorithm. Considering the RMSE metric, the respective improvements are 1.3% and 5.7%. The DA_{next} algorithm also achieves the highest percentage of correct predictions, with its performance edge on this metric ranging from 0.5% to 0.7%. These improvements are achieved at the expense of a coverage drop of 2.2% against the plain CF algorithm, which is deemed to be tolerable; it is notable, however, that the DA_{next} algorithm achieves a better coverage percentage than the $DA_{previous}$ algorithm, which was the winner of the corresponding test in [33].

Table 2. Amazon "Videogames" dataset results

Method	User-user CF (Pearson correlation)				Item-item CF (adjusted cosine similarity)			
	MAE (out of 4)	RMSE	% correct predictions	% coverage	MAE (out of 4)	RMSE	% correct predictions	% coverage
Plain CF	0.777	1.082	32.04	**72.14**	0.383	0.596	66.61	**94.41**
$DA_{vicinity}$	0.752	1.048	32.15	70.86	0.377	0.582	66.89	93.96
$DA_{previous}$	0.732	1.033	32.26	69.76	0.375	0.579	67.22	93.97
DA_{next}	**0.720**	**1.020**	**32.76**	69.95	**0.368**	**0.561**	**67.86**	93.85

Regarding the item-item CF implementation, the DA_{next} algorithm has a performance edge of 1.9% on the MAE metric against the $DA_{previous}$ algorithm, which is the runner up, while the relevant improvement against the plain CF algorithm is 3.9%. Considering the RMSE metric, the respective improvements are 3.1% and 5.9%. With respect to the correct predictions metric, the DA_{next} algorithm is ranked first, having a performance lead of 0.6% against the $DA_{previous}$ algorithm which is ranked second, and a 1.2% performance lead compared to the plain CF algorithm. With respect to the coverage metric, the performance of the DA_{next} algorithm is almost equal to the other two dynamic averages approaches, lagging behind them by 0.1%, while the coverage deterioration of the DA_{next} algorithm as compared to the plain CF algorithm is 0.6%, which is deemed to be tolerable.

Overall, in this dataset the DA_{next} algorithm achieves considerable gains in rating prediction accuracy, under both the user-user and item-item CF implementations, at the expense of small to tolerable deteriorations in coverage.

4.2 The Amazon "CDs and Vinyl" Dataset

Table 3 illustrates the results obtained from the Amazon "CDs and Vinyl" dataset. In this dataset, the user-user CF implementation could formulate a prediction for 59.3% of the cases, while for the item-item CF implementation coverage increases to 86.6%.

Table 3. Amazon “CDs & Vinyl” dataset results

Method	User-user CF (Pearson correlation)				Item-item CF (adjusted cosine similarity)			
	MAE (out of 4)	RMSE	% correct predictions	% coverage	MAE (out of 4)	RMSE	% correct predictions	% coverage
Plain CF	0.702	1.010	29.15	**59.30**	0.335	0.557	64.87	**86.55**
$DA_{vicinity}$	0.682	0.984	29.36	58.66	0.332	0.544	65.94	86.27
$DA_{previous}$	0.669	0.969	29.28	57.71	0.331	0.539	66.73	86.19
$\mathbf{DA_{next}}$	**0.663**	**0.957**	**29.74**	57.95	**0.324**	**0.525**	**67.32**	86.21

Regarding the rating prediction quality, the DA_{next} algorithm again outperforms all other algorithms, under both the user-user and the item-item CF implementations. In more detail, when considering the user-user CF implementation the DA_{next} algorithm achieves the lowest MAE metric (0.663), having a performance lead of 0.9% against the $DA_{previous}$ algorithm, which is ranked second, and a performance lead of 5.6% compared to the plain CF algorithm. For the RMSE metric, the respective performance edges are 1.2% and 5.2%. The DA_{next} algorithm is also ranked first with regards to the percentage of correct predictions metric, with its performance lead ranging from 0.4% over the performance of the $DA_{vicinity}$ algorithm which is ranked second regarding this metric, to 0.6% as compared to the plain CF algorithm. These benefits are achieved at the expense of a coverage drop of 1.3% as compared to the plain CF algorithm, which is considered to be tolerable. It is worth noting that coverage-wise, the DA_{next} algorithm achieves a slightly better performance than the $DA_{previous}$ algorithm, which is the runner up with respect to prediction accuracy.

With respect to the item-item CF implementation, the DA_{next} algorithm attains the lowest value for the MAE metric, which is 2.1% better than the MAE of the runner up algorithm ($DA_{previous}$) and 3.3% better than the value of the plain CF algorithm. Considering the RMSE metric, the respective improvements are 2.6% and 5.7%. The DA_{next} algorithm also computes the highest percentage of correct predictions, outperforming the DA_{next} algorithm, which is ranked second, by 0.6% and the plain CF algorithm by 2.5%. The DA_{next} algorithm exhibits a coverage drop of 0.3% against the plain CF algorithm, which is very small, while coverage-wise it is almost equivalent to the other two dynamic average-based algorithms.

Overall, in this dataset the DA_{next} algorithm achieves considerable gains in rating prediction accuracy, under both the user-user and item-item CF implementations, at the expense of very small to small deteriorations in coverage.

4.3 The Amazon “Movies & TV” Dataset

Table 4 illustrates the results obtained from the Amazon “Movies & TV” dataset. We can observe that, again, the proposed dynamic average-based algorithm DA_{next} achieves the best results under both user-user and item-item CF implementations. Considering user-user CF, the DA_{next} algorithm reduces the MAE by 7.0% as compared to the plain CF algorithm, while it also achieves a MAE reduction of 1.3%, compared to the $DA_{previous}$ algorithm which is the runner up. The respective improvements for the RMSE metric are 5.2% and 0.4%. The DA_{next} algorithm is also ranked first regarding the percentage of correct predictions, with its performance edge ranging from 0.5% (against

DA_{previous}) to 1% (against plain CF). The coverage of the DA_{next} algorithm, however, lags behind that of the plain CF algorithm by 1.6%, a drop which is deemed tolerable.

Considering the item-item CF implementation, the DA_{next} algorithm decreases the MAE by 3.1% against plain CF and by 1.8% against the runner up, which is the $DA_{previous}$ algorithm. The respective reductions in RMSE are higher (4.5% against the plain CF algorithm and 2.8% against the $DA_{previous}$ algorithm), indicating that the DA_{next} algorithm manages to correct some predictions with high errors (recall that the RMSE metric penalizes predictions with high errors). The DA_{next} algorithm is ranked first with respect to the correct prediction percentage by a margin ranging from 0.5% (against the $DA_{previous}$ algorithm) to 0.8% (against the plain CF algorithm). Finally, the coverage drop inflicted by the use of the DA_{next} algorithm is 0.6% as compared to the plain CF algorithm, which is deemed small.

Overall, in this dataset the DA_{next} algorithm achieves noteworthy improvements in rating prediction accuracy, under both the user-user and item-item CF implementations, while the losses in coverage imposed by the algorithm are rated from small to tolerable.

Table 4. Amazon "Movies & TV" dataset results

Method	User-user CF (Pearson correlation)				Item-item CF (adjusted cosine similarity)			
	MAE (out of 4)	RMSE	% correct predictions	% coverage	MAE (out of 4)	RMSE	% correct predictions	% coverage
Plain CF	0.738	1.046	37.00	**78.50**	0.393	0.617	66.74	**95.90**
$DA_{vicinity}$	0.713	1.014	37.37	77.26	0.390	0.611	66.81	95.60
$DA_{previous}$	0.695	0.996	37.55	76.91	0.388	0.606	67.09	95.30
$\mathbf{DA_{next}}$	**0.686**	**0.992**	**38.01**	76.90	**0.381**	**0.589**	**67.56**	95.35

4.4 The Amazon "Books" Dataset

Table 5 illustrates the results obtained from the Amazon "Books" dataset. For the user-user CF, we can observe that again the DA_{next} algorithm is ranked first regarding prediction accuracy, its MAE being 2.6% less than the MAE of the plain CF algorithm and 0.5% smaller than the MAE of the $DA_{previous}$ algorithm, which is ranked second. The improvements regarding the RMSE metric are very similar to those of the MAE (2.5% and 0.7% respectively). The DA_{next} algorithm is also ranked first regarding the correct predictions percentage, by a narrow margin that ranges from 0.2% to 0.6%. The coverage achieved by the DA_{next} algorithm is 1% inferior to that achieved by the plain CF and the $DA_{vicinity}$ algorithms, which are tied for the first place; this drop, however, is deemed small to tolerable.

Table 5. Amazon "Books" dataset results

Method	User-user CF (Pearson correlation)				Item-item CF (adjusted cosine similarity)			
	MAE (out of 4)	RMSE	% correct predictions	% coverage	MAE (out of 4)	RMSE	% correct predictions	% coverage
Plain CF	0.625	0.883	43.67	**53.75**	0.301	0.466	71.97	**90.87**
$DA_{vicinity}$	0.619	0.876	43.96	**53.75**	0.297	0.461	72.02	90.24
$DA_{previous}$	0.612	0.867	44.04	53.36	0.288	0.450	72.21	89.46
$\mathbf{DA_{next}}$	**0.609**	**0.861**	**44.26**	52.73	**0.278**	**0.441**	**72.71**	89.77

Regarding the item-item CF implementation, the DA_{next} algorithm achieves more substantial improvements than in the user-user setting: it achieves a reduction of 7.6% in the MAE, as compared to the plain CF algorithm and a reduction of 3.5% in the MAE against the runner up, which is the $DA_{previous}$ algorithm. The respective improvements regarding the RMSE are 5.4% and 2%. The DA_{next} algorithm also exhibits the best performance regarding the correct predictions percentage, surpassing the $DA_{previous}$ algorithm by 0.5% and the plain CF algorithm by 0.7%. These gains are achieved at the expense of a coverage drop, which is quantified to 1.1% against the plain CF algorithm; notably however, the DA_{next} algorithm attains better coverage than the runner up algorithm in terms of performance ($DA_{previous}$), by a small margin of 0.3%.

Overall, in this dataset the DA_{next} algorithm achieves substantial gains in rating prediction accuracy, under both the user-user and item-item CF implementations, while the losses in coverage imposed by the algorithm are rated from small to tolerable.

4.5 The Amazon "Automotive" Dataset

Table 6 illustrates the results obtained from the Amazon "Automotive" dataset. For the user-user CF, we can observe that the DA_{next} algorithm is ranked first regarding prediction accuracy, reducing the MAE by 7.6% in comparison to the plain CF algorithm; the runner-up algorithm regarding the MAE metric is $DA_{previous}$, which achieves a MAE 4.0% smaller than the plain CF algorithm, lagging behind DA_{next} by 3.6%. The ranking is the same regarding the RMSE metric, with DA_{next} being the top-performing algorithm, achieving an improvement in the RMSE by 8.0% in comparison to the plain CF algorithm, which surpasses the performance of $DA_{previous}$ –which is ranked second– by 3.8%. The DA_{next} algorithm is also ranked first regarding the correct predictions percentage, by a margin that ranges from 2.7% (against the $DA_{previous}$ algorithm) to 5.4% (against the $DA_{vicinity}$ algorithm). The coverage achieved by the DA_{next} algorithm is 1.4% inferior to that achieved by the plain CF algorithm, which is ranked first regarding this metric; this drop, however, is deemed small to tolerable.

Table 6. Amazon "Automotive" dataset results

Method	User-user CF (Pearson correlation)				Item-item CF (adjusted cosine similarity)			
	MAE (out of 4)	RMSE	% correct predictions	% coverage	MAE (out of 4)	RMSE	% correct predictions	% coverage
Plain CF	0.645	1.022	56.12	**53.16**	0.310	0.582	60.34	**91.77**
$DA_{vicinity}$	0.626	1.015	54.25	52.99	0.302	0.581	61.61	91.30
$DA_{previous}$	0.619	0.979	57.01	51.96	0.290	0.566	65.23	90.71
DA_{next}	**0.596**	**0.940**	**59.68**	51.75	**0.282**	**0.533**	**66.28**	90.37

Regarding the item-item CF implementation, the DA_{next} algorithm achieves higher performance improvements than in the user-user setting: it achieves a reduction of 9.0% in the MAE, as compared to the plain CF algorithm, with this improvement being better by 2.6%, as compared to that achieved by the runner up, which is the $DA_{previous}$ algorithm. The respective performance edges regarding the RMSE are 8.4% and 5.7%. The DA_{next} algorithm also attains the best performance regarding the correct predictions percentage, surpassing the $DA_{previous}$ algorithm by 1.1% and the plain CF algorithm by 5.9%. These gains are achieved at the expense of a coverage drop, which is quantified to 1.4% against the plain CF algorithm.

Overall, in this dataset the DA_{next} algorithm achieves substantial gains in rating prediction accuracy, under both the user-user and item-item CF implementations, while the losses in coverage imposed by the algorithm are rated from small to tolerable.

4.6 The MovieLens "Old 100K" Dataset

Table 7 depicts the results obtained from the MovieLens "Old 100K" dataset. In this dataset, which is a relatively dense one, we can observe that in both the user-user and the item-item CF implementations, practically no coverage drop is incurred by the introduction of the dynamic average-based algorithms, and coverage is close to 100% in all cases, with negligible variations. As shown in the next subsections, this behavior is consistent across all dense datasets (i.e. all MovieLens datasets and the Netflix dataset).

Regarding rating prediction quality, under the user-user CF implementation the DA_{next} algorithm has a marginal performance edge over the runner up, $DA_{previous}$, since it exhibits smaller values for both the MAE and the RMSE metric by 0.4%. In comparison to the plain CF algorithm the performance lead of the DA_{next} algorithm is considerably higher (MAE: 3.9%; RMSE: 3.5%). With respect to the correct predictions percentage criterion, the DA_{next} algorithm surpasses the performance of all other algorithms, having a lead of 1.1% against the $DA_{vicinity}$ algorithm which is ranked second and a lead of 2.2% against the plain CF algorithm.

Table 7. MovieLens "Old 100K" dataset results

Method	User-user CF (Pearson correlation)				Item-Item CF (adjusted cosine similarity)			
	MAE (out of 4)	RMSE	% correct predictions	% coverage	MAE (out of 4)	RMSE	% correct predictions	% coverage
Plain CF	0.735	0.939	42.34	99.82	0.622	0.797	48.99	**99.90**
$DA_{vicinity}$	0.715	0.916	43.38	99.83	0.618	0.790	49.37	**99.90**
$DA_{previous}$	0.709	0.910	43.32	**99.84**	0.611	0.778	50.22	**99.90**
$\mathbf{DA_{next}}$	**0.706**	**0.906**	**44.49**	99.81	**0.603**	**0.760**	**50.82**	**99.90**

Considering the item-item CF implementation, the DA_{next} algorithm is again ranked first in all accuracy-related metrics. Regarding the MAE, the DA_{next} algorithm outscores the runner up (which is the $DA_{previous}$ algorithm) by 1.3% and the plain CF algorithm by 3.1%; in relation to the RMSE, the performance edge of the DA_{next} algorithm against the $DA_{previous}$ and the plain CF algorithms is 2.3% and 4.6%, respectively. Finally, the DA_{next} algorithm produces correct results in 0.6% more cases than the $DA_{previous}$ algorithm does, and in 1.8% more cases than the plain CF algorithm does.

Overall, in this dataset the DA_{next} algorithm achieves considerable gains in rating prediction accuracy, under both the user-user and item-item CF implementations, while practically no loss in coverage is sustained.

4.7 The MovieLens "Latest-20M - Recommended for New Research" Dataset

Table 8 depicts the results obtained from the MovieLens "Latest-20M, Recommended for New Research" dataset. As noted in the previous subsection, the coverage in this dataset is near 100% under both CF implementation strategies and remains practically unaltered by the introduction of dynamic average-based algorithms.

With respect to rating prediction quality, under the user-user CF implementation the DA_{next} algorithm is again ranked first, improving the MAE by 3.6% as compared to the plain CF algorithm and by 1.2% as compared to the $DA_{previous}$ algorithm, which is ranked second. The respective improvements considering the RMSE metric are 4.8% and 1.6%, respectively. Finally, the DA_{next} algorithm formulates the most correct predictions, surpassing the performance of the $DA_{previous}$ algorithm by 0.4% and that of the plain CF algorithm by 1.8%.

Table 8. MovieLens "Latest-20M - recommended for New Research" dataset results

Method	User-user CF (Pearson correlation)				Item-item CF (adjusted cosine similarity)			
	MAE (out of 9)	RMSE	% correct predictions	% coverage	MAE (out of 9)	RMSE	% correct predictions	% coverage
Plain CF	1.352	1.771	24.26	**99.96**	1.548	1.984	20.50	**99.99**
$DA_{vicinity}$	1.326	1.740	24.98	99.90	1.512	1.921	22.87	99.97
$DA_{previous}$	1.319	1.714	25.60	99.96	1.484	1.853	24.81	99.96
DA_{next}	**1.303**	**1.686**	**26.02**	99.94	**1.429**	**1.795**	**25.87**	99.96

Considering the item-item CF implementation, the DA_{next} algorithm has been found to produce the most accurate recommendations, exhibiting improvements in the MAE that range from 3.7% (against the $DA_{previous}$ algorithm) to 7.7% (against the plain CF algorithm); the corresponding improvements in RMSE range from 3.1% (against the $DA_{previous}$ algorithm) to 9.5% (against the plain CF algorithm). Finally, the DA_{next} algorithm produces the most correct predictions, having a performance lead of 1.1% in comparison to the $DA_{previous}$ algorithm, and a lead of 5.4% against the plain CF algorithm.

Overall, in this dataset the DA_{next} algorithm achieves considerable gains in rating prediction accuracy under both the user-user CF implementation and more substantial gains under the item-item CF implementation, while practically no loss in coverage is sustained.

4.8 The MovieLens "Latest 100K - Recommended for Education and Development" Dataset

Table 9 depicts the results obtained from the MovieLens "Latest 100K- Recommended for education and development" dataset. We can again notice that no coverage loss is introduced by the dynamic average-based algorithms, with coverage being near-100% in all cases.

With respect to rating prediction quality, under the user-user CF implementation scenario the DA_{next} algorithm is ranked first, achieving a reduction in the MAE of 3.2% in comparison to the plain CF and a reduction of 0.5% in comparison to the $DA_{previous}$ algorithm, which is ranked second. The respective gains for the RMSE metric are 4.3% and 1%, being higher than those of the MAE metric, indicating that the DA_{next} algorithm improves predictions with high errors. Finally, the DA_{next} algorithm computes approximately 1.7% more correct predictions than both the plain CF and $DA_{previous}$ algorithms, while it also exceeds the performance of the $DA_{vicinity}$ algorithm (which is the runner up for this metric) by 1.4%.

With regards to the item-item CF implementation scenario, the DA_{next} algorithm is again ranked first, achieving a 6.2% reduction in the MAE and 5.2% reduction in the RMSE, as compared to the plain CF algorithm. The $DA_{previous}$ algorithm is ranked second, lagging behind the DA_{next} algorithm by 2.0% regarding both the MAE and the RMSE metrics. Finally, the DA_{next} algorithm manages to produce the most correct predictions, computing 5.1% more correct predictions than the plain CF and 1.5% more correct predictions than the $DA_{previous}$ algorithm, which is ranked second.

Table 9. MovieLens "Latest 100K - recommended for education and development" dataset results

Method	User-user CF (Pearson correlation)				Item-item CF (adjusted cosine similarity)			
	MAE (out of 9)	RMSE	% correct predictions	% coverage	MAE (out of 9)	RMSE	% correct predictions	% coverage
Plain CF	1.404	1.858	24.16	99.57	0.999	1.374	34.34	**99.70**
$DA_{vicinity}$	1.376	1.816	24.44	**99.60**	0.987	1.368	35.08	99.68
$DA_{previous}$	1.366	1.796	24.07	99.40	0.956	1.330	37.99	99.65
DA_{next}	**1.359**	**1.778**	**25.83**	99.49	**0.937**	**1.303**	**39.46**	99.69

Overall, in this dataset the DA_{next} algorithm achieves considerable gains in rating prediction accuracy under the user-user CF implementation scenario and more substantial gains under item-item CF; in terms of coverage, practically no loss in coverage is sustained.

4.9 The “Netflix Competition” Dataset

Table 10 depicts the results obtained from the “Netflix Competition” dataset. Again, the coverage is close to 100% in all cases, and the losses sustained by the introduction of the dynamic average-based algorithms are negligible.

Regarding rating prediction quality, under the user-user CF scenario the DA_{next} algorithm is ranked first; it achieves a MAE improved by 3.7% against the plain CF algorithm and by 1.5% against the $DA_{vicinity}$ algorithm, which is ranked second; for the RMSE metric, the respective improvements are 4.4% and 1.4%. Finally, the DA_{next} algorithm computes 2.2% more correct predictions that the plain CF algorithm and 1.4% more correct predictions than the $DA_{vicinity}$ algorithm, which is ranked second for this metric.

Table 10. “Netflix Competition” dataset results

Method	User-user CF (Pearson correlation)				Item-item CF (adjusted cosine similarity)			
	MAE (out of 4)	RMSE	% correct predictions	% coverage	MAE (out of 4)	RMSE	% correct predictions	% coverage
Plain CF	0.758	0.960	41.42	**99.10**	0.857	1.061	33.61	**99.40**
$DA_{vicinity}$	0.741	0.931	42.25	99.03	0.811	0.956	39.98	99.30
$DA_{previous}$	0.752	0.936	42.12	99.00	0.808	0.954	40.08	99.23
DA_{next}	**0.730**	**0.918**	**43.60**	99.02	**0.761**	**0.892**	**44.30**	99.25

Considering the item-item CF implementation, the DA_{next} algorithm outperforms all other algorithms by a wider margin. For the MAE criterion, it achieves an improvement of 11.2% against the plain CF algorithm, and 5.8% against the $DA_{previous}$ algorithm, which is ranked second; the respective improvements for the RMSE metric are 15.9% and 6.5%. Finally, the DA_{next} algorithm computes the most correct predictions, having a performance edge of 4.2% against the $DA_{previous}$ algorithm, which is the runner up, and an edge of 10.7% against the plain CF algorithm.

Overall, in this dataset the DA_{next} algorithm achieves considerable gains in rating prediction accuracy under the user-user CF implementation and more substantial gains under the item-item CF implementation, while practically no loss in coverage is sustained.

4.10 The “Ciao” Dataset

Table 11 illustrates the results obtained from the “Ciao” dataset. For the user-user CF, we can observe that the DA_{next} algorithm is ranked first regarding prediction accuracy, reducing the MAE by 6.1% in comparison to the plain CF algorithm; the runner-up algorithm regarding the MAE metric is $DA_{previous}$, which achieves a MAE 3.4% smaller than the plain CF algorithm, lagging thus behind DA_{next} by 2.7%. The ranking is the same regarding the RMSE metric, with DA_{next} being the top-performing algorithm, achieving an improvement in the RMSE by 9.7% against the plain CF algorithm, which surpasses the performance of $DA_{previous}$ –which is ranked second– by 3.0%. The DA_{next}

algorithm is also ranked first regarding the correct predictions percentage, by a margin that ranges from 1.7% (against the $DA_{previous}$ algorithm) to 3.2% (against the plain CF algorithm). The coverage achieved by the DA_{next} algorithm is 1.4% inferior to that achieved by the plain CF algorithm, which is ranked first regarding this metric; this drop, however, is deemed small to tolerable.

Table 11. "Ciao" dataset results

Method	User-user CF (Pearson correlation)				Item-item CF (adjusted cosine similarity)			
	MAE (out of 4)	RMSE	% correct predictions	% coverage	MAE (out of 4)	RMSE	% correct predictions	% coverage
Plain CF	0.853	1.089	37.40	**76.86**	0.378	0.567	68.06	**99.26**
$\mathrm{DA}_{\mathrm{vicinity}}$	0.844	1.064	37.86	76.21	0.373	0.560	69.01	**99.26**
$\mathrm{DA}_{\mathrm{previous}}$	0.824	1.016	38.93	75.96	0.366	0.550	69.99	99.13
$\mathbf{DA}_{\mathbf{next}}$	**0.801**	**0.983**	**40.64**	75.47	**0.359**	**0.534**	**71.60**	99.11

Regarding the item-item CF implementation, the DA_{next} algorithm achieves a reduction of 5.0% in the MAE, as compared to the plain CF algorithm, with this improvement being better by 1.9%, as compared to that achieved by the runner up, which is the $DA_{previous}$ algorithm. The respective performance edges regarding the RMSE are 5.8% and 2.8%. The DA_{next} algorithm also attains the best performance regarding the correct predictions percentage, surpassing the $DA_{previous}$ algorithm by 1.6% and the plain CF algorithm by 3.5%. These gains are achieved at the expense of a practically negligible coverage drop, which is quantified to 0.15%.

Overall, in this dataset the DA_{next} algorithm achieves substantial gains in rating prediction accuracy, under both the user-user and item-item CF implementations, while the losses in coverage imposed by the algorithm are rated from negligible to tolerable.

4.11 Algorithms Comparison

In this subsection, we consolidate our findings from all datasets, to provide a comprehensive overview of the algorithms' performance regarding the prediction accuracy metrics. In all comparisons, the performance of the plain CF algorithm is taken as a baseline. We also compare the proposed algorithm against other approaches that have been published and evaluated in the literature.

Figure 1 depicts the improvement in the MAE achieved by all dynamic average-based algorithms under the user-user CF implementation scenario. Clearly, the DA_{next} algorithm achieves the best results, with its performance lead being on average approximately 1.5% against the $DA_{previous}$ algorithm which is the runner up; the respective reduction in the MAE against the baseline algorithm is 5.1% on average. It is worth noting that the DA_{next} algorithm surpasses the performance of both other algorithms in all datasets, while the $DA_{previous}$ algorithm is ranked second in 9 datasets and third in one dataset (Netflix). From the detailed examination of our results, the DA_{next} algorithm formulated the prediction with the lowest error in 94.3% of the prediction formulation requests across all datasets (this percentage includes ties for the first place, i.e. cases where the DA_{next} algorithm and some other algorithm(s) produced the same prediction, and this was the closest prediction to the actual rating).

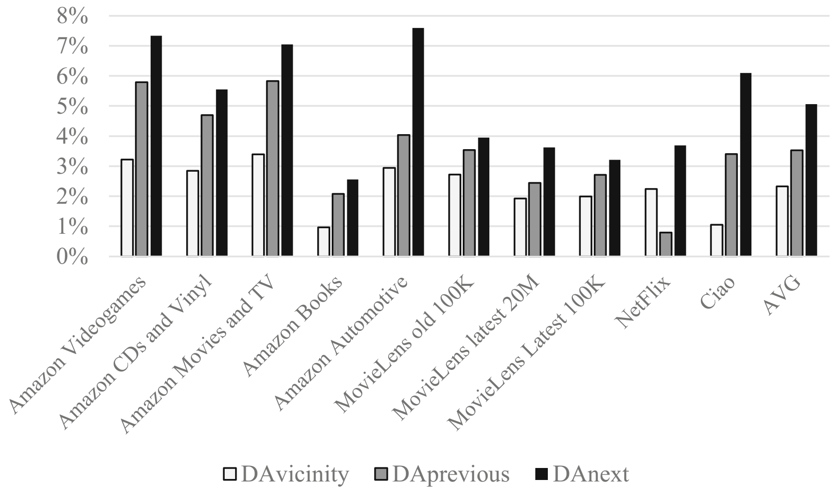

Fig. 1. MAE improvement achieved by the dynamic average-based algorithms under the user-user CF implementation

Figure 2 presents the respective improvements regarding the RMSE metric. In five datasets (and on average), the improvements are very similar to those of the MAE metric shown in Fig. 1, indicating that prediction improvements are spread uniformly among predictions with high and low errors. In three datasets (MovieLens latest 20M; MovieLens Latest 100K; and Ciao), the improvement in the RMSE metric is higher than the improvement in the MAE metric by a margin ranging from 1.1% to 3.6%, indicating that the DA_{next} algorithm manages to eliminate some high errors in predictions, while in two other datasets (Amazon Videogames and Amazon Movies and TV), the improvements in the MAE metric surpass those in the RMSE metric by 1.6% and 1.9% respectively, indicating that the DA_{next} algorithm mostly adjusts predictions with low errors. Again, the DA_{next} algorithm is consistently ranked first across all datasets, with its average performance lead against the runner up algorithm ($DA_{previous}$) being 1.5%, and the respective performance lead against the plain CF algorithm being 5.3%.

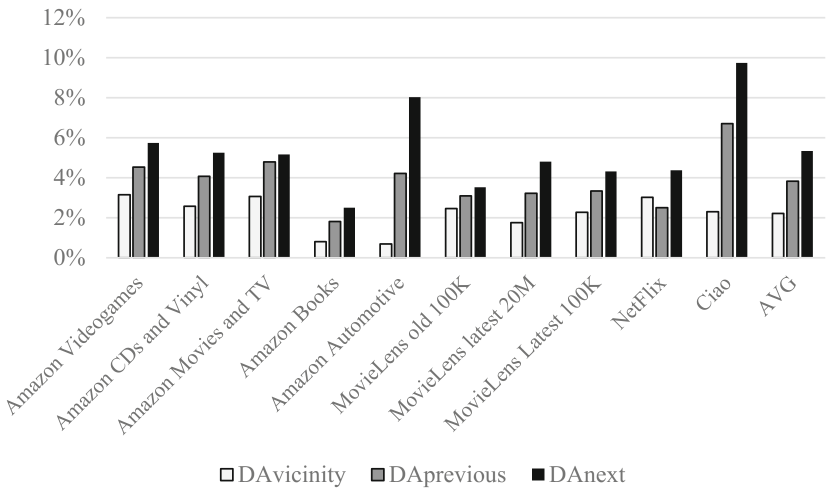

Fig. 2. RMSE improvement achieved by the dynamic average-based algorithms under the user-user CF implementation

Figure 3 illustrates the improvement in the MAE achieved by the dynamic average-based algorithms under the item-item CF implementation scenario. Again, the DA_{next} algorithm is consistently ranked first across all datasets, achieving a reduction of 6.0% on average against the baseline algorithm, and surpassing the performance of the runner up algorithm ($DA_{previous}$) by 2.6% on average. Performance gains in this case exhibit higher variations than those under the user-user implementation scenario, mainly owing to the Netflix dataset in which the DA_{next} algorithm achieves very substantial improvements in the MAE metric (11.2% against the plain CF algorithm and 5.5% against the runner up, which is the $DA_{previous}$ algorithm). From the detailed examination of our results, the DA_{next} algorithm formulated the prediction with the lowest error in 91.4% of the prediction formulation requests across all datasets.

Figure 4 presents the relevant improvements regarding the RMSE metric. Again, DA_{next} achieves the best results, with its performance lead against the $DA_{previous}$ algorithm, which is the runner up, being equal to 3.0% on average; the respective RMSE reduction against the baseline algorithm is 7.1% on average. In relation to the baseline algorithm, the average RMSE metric improvement is higher than that of the MAE metric by approximately 1.1%, indicating that the DA_{next} algorithm achieves to eliminate some high prediction errors. Considering individual datasets, the RMSE metric improvement is higher than the improvement of MAE in seven of the datasets (Amazon Videogames, Amazon CDs and Vinyl, Amazon Movies and TV, MovieLens old 100K, MovieLens latest 20M, Netflix and Ciao); in the remaining four three (Amazon Books, Amazon Automotive and MovieLens Latest 100K), the improvement in the RMSE metric lags behind the improvement of the MAE metric by a margin ranging from 0.6% to 2.2%, indicating that in these datasets the DA_{next} algorithm mostly improves predictions with low errors.

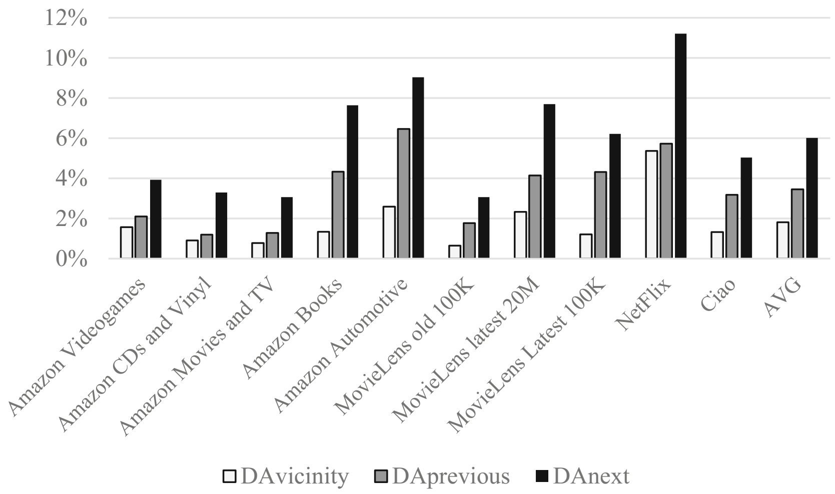

Fig. 3. MAE improvement achieved by the dynamic average-based algorithms under the item-item CF implementation

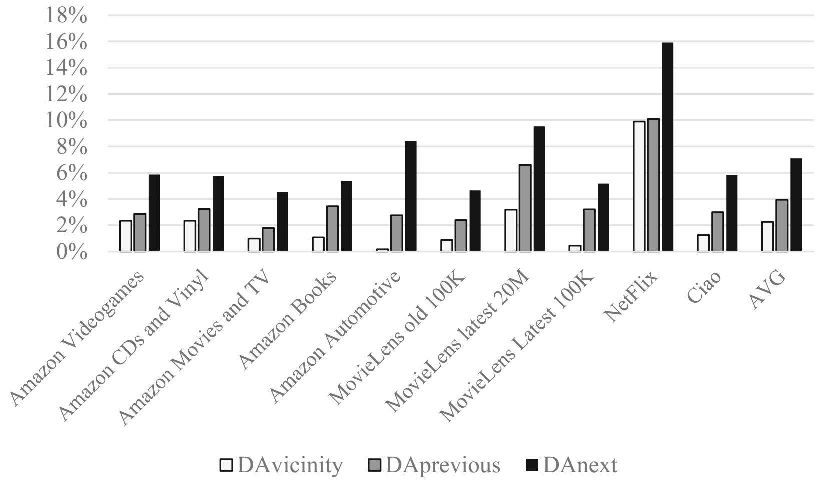

Fig. 4. RMSE improvement achieved by the dynamic average-based algorithms under the item-item CF implementation

Figure 5 illustrates the improvements regarding the correct prediction percentage for all dynamic average-based algorithms under the user-user CF implementation scenario. The DA_{next} algorithm achieves improvements ranging from 0.4% to 3.6% against the baseline algorithm. Regarding the dynamic average-based algorithms, we can observe that the DA_{next} algorithm is consistently ranked first across all datasets; the $DA_{previous}$ algorithm is ranked second in six of the datasets, lagging behind the DA_{next}

algorithm by a margin ranging between 0.22% and 2.67% (1.1% on average), while the $DA_{vicinity}$ algorithm is ranked second in the remaining four datasets, falling behind the performance of DA_{next} by a margin ranging between 0.3% and 5.43% (1.50% on average).

Figure 6 depicts the corresponding findings for the item-item CF implementation. Again, the DA_{next} algorithm is ranked first across all datasets, with the performance gains being considerably higher: the improvement against the baseline algorithm is 3.8% on average, ranging from 0.7% to 10.7%, while in comparison to the runner up, $DA_{previous}$, the performance edge of the DA_{next} algorithm is 1.2% on average, ranging from 0.5% to 4.2%. A significant part of this performance edge is owing to the results of the Netflix dataset, where the DA_{next} algorithm has the widest performance gap from the other algorithms. Besides the Netflix dataset, we can observe that the DA_{next} algorithm achieves its highest performance improvements in the latest MovieLens datasets (MovieLens latest 20M and MovieLens Latest 100K) and the Amazon "Automotive" dataset. The Netflix and both the Movielens datasets share the property of being denser than other datasets (and being the only dense datasets in the experiment), with their $\frac{\#ratings}{\#users * \#items}$ ratio exceeding 1%, while in the rest of the datasets this ratio ranges from 0.001% (Amazon Books) to 0.537% (MovieLens old 100K). However, further investigation is required to determine whether this behavior is owing solely to the density of the datasets, or to other properties as well. Interestingly, the DA_{next} algorithm achieves substantial improvements in the Amazon "Automotive" dataset too, which is relatively sparse.

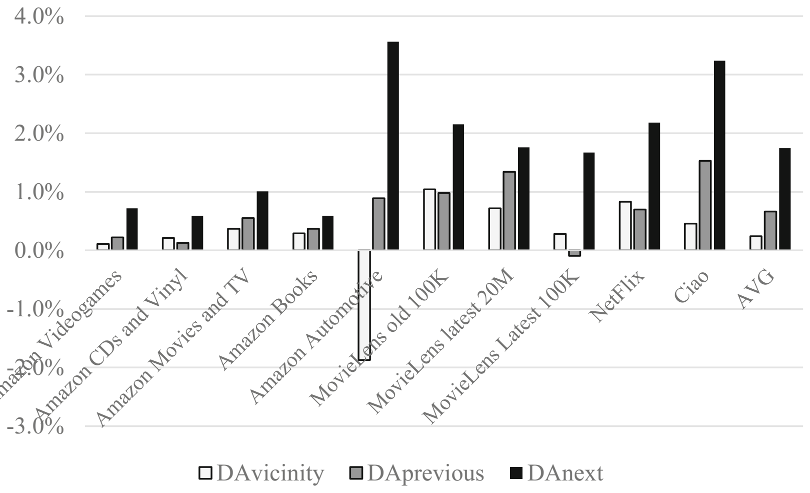

Fig. 5. Correct predictions percentage improvement achieved by the dynamic average-based algorithms under the user-user CF implementation

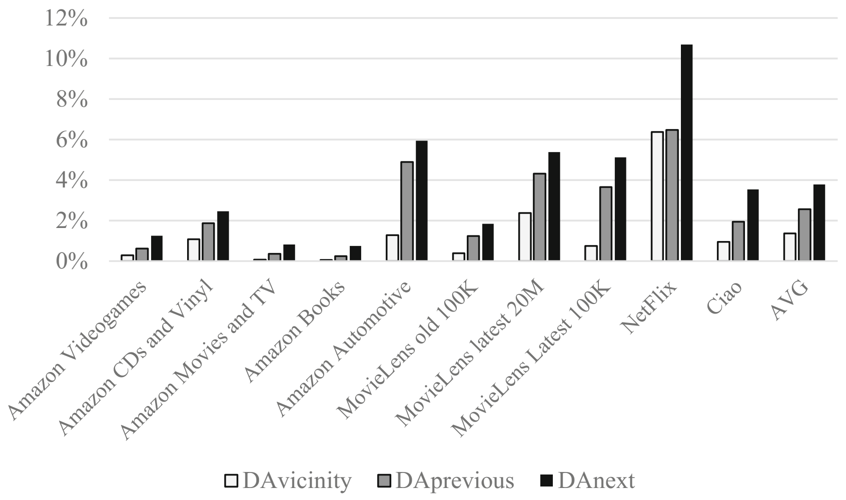

Fig. 6. Correct predictions percentage improvement achieved by the dynamic average-based algorithms under the item-item CF implementation

Summarizing, we can clearly see that in all datasets and under both CF implementation scenarios, the proposed algorithm outperforms all other algorithms achieving (1) the highest MAE reduction, (2) the highest RMSE reduction and (3) the highest correct predictions' percentage.

In relation to other approaches reported in the literature, Vaz et al. [16] exploit temporal dynamics by considering the age of user ratings and community ratings in an item-item CF scenario. The method presented therein achieves a MAE improvement of 0.2% when the a parameter (which controls the way that the age of user ratings is handled) ranges from 0.1 to 0.6 and the b parameter (which controls the way that the age of community ratings is handled) is set to 0, i.e. when the age of community ratings is disregarded. Under the item-item CF scenario, the presented algorithm achieves MAE reductions ranging from 3.05% to 11.20%, clearly thus achieving substantially higher improvements than the one presented in [16].

The ISCF (Interest Sequence CF) algorithm proposed by Cheng et al. [53] accommodates temporal dynamics in a user-user CF scenario, by considering user interest sequences, and is evaluated against the four real-world datasets (Ciao, Flixter, MovieLens old 100K, MovieLens latest 100K). The ISCF algorithm achieves an average reduction on the MAE (considering the four aforementioned datasets) by 2.41% (with improvements ranging from 0.21% to 4.33%), while the average reduction on the RMSE is 3.0% (with improvements ranging from 0.59% to 7.81%). The DA_{next} algorithm proposed in this paper achieves higher improvements regarding the MAE and the RMSE than ISCF on average, both in individual datasets and on overall average, as illustrated in Table 12.

Table 12. Accuracy improvements achieved by the DA$_{next}$ and the ISCF [53] algorithms

	Ciao		MovieLens old 100K		MovieLens latest 100K		Average (over all tested datasets)	
	MAE	RMSE	MAE	RMSE	MAE	RMSE	MAE	RMSE
ISCF	0.21%	0.59%	3.23%	0.21%	0.59%	2.24%	2.41%	3.0%
DA$_{next}$	6.10%	9.73%	3.95%	3.51%	3.20%	4.3%	5.1%	5.3%

Dror et al. [2] propose an algorithm for identifying and exploiting drifts in short-lived music listening sessions. Their algorithm is evaluated against the Yahoo Music! dataset; the algorithm presented in [2] encompasses two steps that exploit temporal dynamics, namely the *user session bias* and the *items temporal dynamics bias*. These two steps achieve a cumulative improvement in RMSE equal to 3.98%; on the other hand, the proposed algorithm has been found to improve RMSE by 6.52% under the user-user scenario and by 4.89% under the item-item scenario. Therefore, in this case the DA_{next} algorithm is found to achieve higher improvement levels, under both the user-user and the item-item CF scenarios, than the provisions for exploitation of temporal dynamics proposed by Dror et al. [2].

Finally, Lo et al. [52] develop a temporal matrix factorization approach for tracking concept drift in each individual user latent vector. The method proposed therein is applied on four real-world datasets, achieving reductions in the RMSE metric ranging from 0.24% (when applied on the MovieLens 20M dataset) to 5.04% (when applied on the Ciao dataset); the average RMSE metric improvement achieved by the algorithm presented in [52] considering the four real-world datasets is 1.73%. The respective improvements regarding the RMSE metric achieved by the algorithm proposed in this paper are as follows: regarding the MovieLens 20M dataset, the RMSE is decreased by 4.8% under the user-user scenario and by 9.5% under the item-item scenario; in regards to the Ciao dataset, the RMSE is decreased by 9.7% under the user-user scenario and by 5.8% under the item-item scenario; finally, the average RMSE reduction achieved by the proposed algorithm across all ten examined datasets is 5.3% under the user-user scenario and 7.1% under the item-item scenario. Recapitulating, the algorithm proposed in this paper achieves more substantial improvements in rating prediction accuracy than the one proposed in [52], both considering individual datasets and average performance, and this performance edge is achieved under the user-user CF scenario as well as the item-item CF scenario.

4.12 Algorithm Complexity and Scalability

In this subsection, we investigate the complexity and the scalability of the proposed algorithm, and compare them with the complexity and scalability of the other algorithms examined in our experiments [33]. In our investigation, we consider all phases of the algorithms, i.e. (i) bootstrap (initial computations of dynamic averages and Pearson similarities), (ii) computation of recommendations and (iii) update of dynamic averages and Pearson similarities.

Bootstrap Phase. Within the bootstrap phase, the dynamic average-based algorithms ($\mathrm{DA}_{\mathrm{next}}$, $\mathrm{DA}_{\mathrm{previous}}$ and $\mathrm{DA}_{\mathrm{vicinity}}$) need in order to compute the dynamic averages for each rating of the users as well as the Pearson similarities between users. The plain CF algorithm needs to compute the global average of each user's ratings and the Pearson similarities between users. For each of the algorithms, the relevant complexities are presented in the following paragraphs.

Computation of Needed Averages. For the DA_{next} algorithm, Eq. 4 indicates that for each rating of a user all ratings subsequently entered by the same user need to be examined to compute the rating's dynamic average; under this view, the complexity of the calculation of the dynamic averages for each user is $O(ru^2)$, where *ru* is the number of ratings of the user. However, Listing 1 shows an optimization for the procedure of calculating the dynamic averages, according to which ratings are sorted in ascending timestamp order and then the sorted list is traversed from the end to the beginning, computing at each step the relevant rating's dynamic average by only considering the previous rating's value and the results of the computations made in the previous steps. Thus the complexity of computing the dynamic averages for each user under the DA_{next} algorithm is $O(ru * log(ru))$, i.e. the complexity of the sorting phase, which dominates the complexity of the whole operation. The complexity of computing the dynamic averages for all users is $O(N * \bar{r} * \log(\bar{r}))$, where N is the number of users and $\bar{r}$ is the average number of ratings per user.

Regarding $DA_{previous}$ algorithm, the same technique can be employed for computing dynamic averages (with the sorted list being traversed from the oldest rating to the newest one), hence the complexity of the computing the dynamic averages for all users is again $O(N * \bar{r} * \log(\bar{r}))$.

In the case of the temporal vicinity algorithm, $DA_{vicinity}$, Eq. 5 indicates that, when computing the dynamic average for a specific rating $r_{u,i}$, every rating $r_{u,i'}$ entered by the same user u is assigned a weight, based on its temporal vicinity to $r_{u,i}$ and subsequently its value is multiplied by that weight to compute the dynamic average of $r_{u,i}$. Therefore the complexity of calculating the dynamic average for a specific rating is $O(ru)$ and consequently the complexity of computing all dynamic averages for a specific user's ratings is $O(ru^2)$ and the complexity of calculating all users' dynamic averages is $O(N * \bar{r}^2)$. An optimization is possible in this procedure: when computing the dynamic average for a rating $r_{u,i}$ the initial and the final part of the temporally sorted rating list for which $w_{u,x}(r_{u,i}) < \varepsilon$, where ε is a small value (e.g. 10^{-2}), which will have a minimal impact at the computation of the dynamic average of $r_{u,i}$ could be excluded from the computation. However, due to the fact that the $DA_{vicinity}$ algorithm achieved the smallest improvements out of all the dynamic average algorithms considered, such optimizations were not considered further.

Finally, the plain CF algorithm computes each user's global average with a single pass along the user's ratings, therefore complexity of calculating of the global average for a user u is $O(ru)$, and the complexity of computing all users' global averages is $O(N * \bar{r})$.

Computation of Pearson Similarities. The method for computing the Pearson similarities between users is common to all four algorithms; the only difference between the plain CF algorithm on the one hand and the dynamic average-based algorithms on the other is that the plain CF algorithm uses the single global average per user for adjusting each rating ($\overline{R_X}$ *and* $\overline{R_Y}$, c.f. Eq. 1), while the dynamic average-based algorithms use a precomputed, rating-specific average ($DA_{Alg}(R_{X,i})$ and $DA_{Alg}(R_{Y,i})$, c.f. Eq. 2). This does not affect the complexity of the operations, since the same amount of computations takes place. In practice, performance differences may occur due to the fact that global averages may be stored in registers for fast access, while dynamic averages should be fetched from main memory, however this difference has been quantified to be small (measurements are presented below). In terms of complexity, for computing the Pearson similarity between two users X and Y the items rated in common by both users need to be identified and for each of these pair of item ratings simple computations are performed. The identification of items commonly rated among two users can be achieved by creating a hash set for one of the user's ratings and then iterating over the other user's ratings and examining whether they exist in the hash set. With a sufficiently large key space value for the hash function (which is always possible since users have at most a few thousand ratings and a hash set with tens of millions of distinct keys can be easily accommodated in memory), insertion and lookup in the hash set is O(1), therefore the complexity of the creation of the hash set is $O(r_x)$ and the complexity of the lookup is $O(r_Y)$, where r_X and r_Y is the number of ratings entered by users *X* and *Y*, respectively. Consequently, the complexity of the computation of the Pearson similarity between two users X and Y is $O(r_x) + O(r_Y)$, or in the general case $\mathrm{O}(2 * \bar{\mathrm{r}})$. Since the Pearson similarity between any pair of users needs to be computed, the overall complexity for the computation of the Pearson similarity is $\mathrm{O}\left(2 * \mathrm{N}^2 * \bar{\mathrm{r}}\right)$; since however the Pearson similarity is symmetric (i.e. *Pearson_sim(X, Y) = Pearson_sim(Y, X)*), the number of computations can be reduced to the half, yielding an overall complexity of $\mathrm{O}\left(\mathrm{N}^2 * \bar{\mathrm{r}}\right)$.

Table 13 summarizes the result of the complexity analysis for the bootstrap phase of the four algorithms. Note that in the case of the plain CF algorithm, the complexity of the Pearson similarity computation phase dominates the complexity of the dynamic average computation phase, hence in the overall complexity only the former appears.

Regarding the disk storage space needed, the plain CF algorithm needs to store only triples of the form *(user, item, rating)*, while all dynamic average-based algorithms need to extend the triple to accommodate the rating timestamp. In many cases, the timestamp is stored as seconds since the epoch (1/1/1970), for which 8 bytes are sufficient. Even with datasets with billions (10^9) of ratings, this extension can be accommodated, since even commodity hardware supports storages at the TB level (10^{12}).

Table 13. Complexity analysis for the algorithms; bootstrap phase

Method	*Dynamic average computation complexity*	*Pearson similarity computation complexity*	*Overall complexity*
Plain CF	$O(N * \bar{r})$	$O(N^2 * \bar{r})$	$O(N^2 * \bar{r})$
$\mathrm{DA}_{\mathrm{vicinity}}$	$O(N * \bar{r}^2)$	$O(N^2 * \bar{r})$	$O(N * \bar{r}^2) + O(N^2 * \bar{r})$
$\mathrm{DA}_{\mathrm{previous}}$	$O(N * \bar{r} * \log(\bar{r}))$	$O(N^2 * \bar{r})$	$O(N * \bar{r} * \log(\bar{r})) + O(N^2 * \bar{r})$
$\mathrm{DA}_{\mathrm{next}}$	$O(N * \bar{r} * \log(\bar{r}))$	$O(N^2 * \bar{r})$	$O(N * \bar{r} * \log(\bar{r})) + O(N^2 * \bar{r})$

Finally, in terms of memory storage, the dynamic average-based algorithms require the storage of the dynamic average along with each rating. Dynamic averages are represented with a float-typed value, requiring a four additional bytes. Taking into account that the memory capacity of contemporary high-end servers has substantially increased at the TB level (e.g. [48] can accommodate more than 6 TB of memory), this extension is not expected to be a problem. It is worth noting that the $DA_{previous}$ and the DA_{next} algorithms may totally drop the timestamps from a user's ratings after the user's dynamic averages have been computed, hence it is not necessary at any time point to accommodate both all ratings' timestamps and dynamic averages (the $DA_{vicinity}$ algorithm may need to retain the timestamp in memory, in order to recompute the dynamic averages when new ratings are entered).

In the case that the above quantified increases in storage space requirements is a consideration, storage space needs may decrease by computing dynamic averages per time window (e.g. week; month; year), in a fashion similar to the one described in [23]. The study of the effect that such an approach would have on the quality of formulated predictions is part of our future work.

Figure 7 illustrates the time needed to compute all ratings' (dynamic) averages for the various datasets under each of the four examined algorithms.

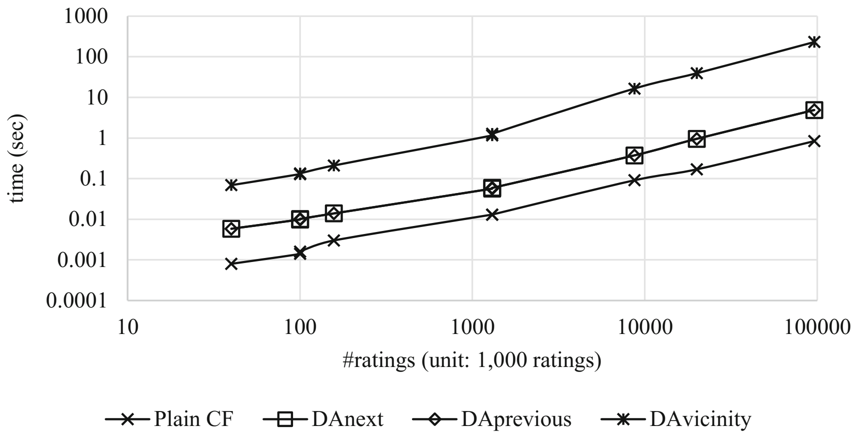

Fig. 7. Time needed for (dynamic) average computation

In Fig. 7 (n.b. both axis are in logarithmic scale) we can notice that the time needed by the plain CF algorithm to compute all global averages is less than 1 s (0.837 s. for the Netflix dataset which contains 100M ratings), while the respective time needed by the $\mathrm{DA}_{\mathrm{next}}$ and $\mathrm{DA}_{\mathrm{previous}}$ algorithms (whose lines fully coincide) is always less than 5 s (4.879 s for the Netflix dataset); this increment is definitely considered manageable. On the other hand, the $\mathrm{DA}_{\mathrm{vicinity}}$ algorithm needs significantly more time, up to 230 s for the Netflix dataset.

Figure 8 illustrates the time needed to compute the Pearson similarities for all user pairs. The time needed by all algorithms has been found to be almost identical, with the dynamic average-based algorithms needing approximately 0.98% more time than the plain CF algorithm, which –as stated above– is attributed to the fact that global averages may be stored in registers for fast access, while dynamic averages must be fetched from main memory. While generally the time needed appears to scale linearly with the number of ratings, we can notice three data points where scaling is not linear:

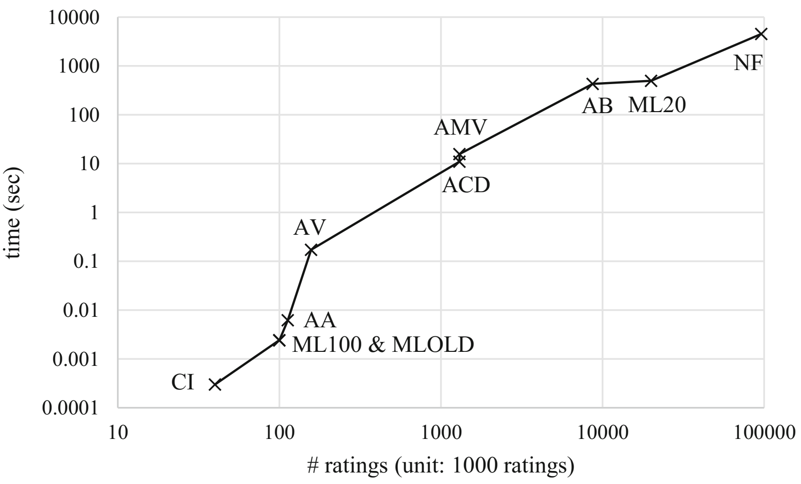

Fig. 8. Time needed for computing the pairwise Pearson similarities between users

1. The time needed for the Amazon Videogames dataset (denoted as AV in Fig. 8) is 72 times higher than the time needed for the MovieLens “Old 100K Dataset” (denoted as MLOLD), although the number of ratings is only 1.5 times higher. This is attributed to the fact that the number of users in the Amazon Videogames dataset is 8 times higher than the respective number of users in the MovieLens “Old 100K Dataset” and, as indicated by the complexity formulas in Table 13, the time needed is proportional to the square of the number of users, while scaling linearly with the average number of ratings.
2. The time needed for the Amazon Movies dataset (denoted as AMV in Fig. 8) is 1.3 times higher than the time needed for the Amazon CDs and Vinyl dataset (denoted as ACD), despite the fact that both datasets contain the same number of ratings. Again, this is attributed to the fact that the Amazon Movies dataset contains a higher number of users than the Amazon CDs and Vinyl dataset (approximately 11% higher).
3. Finally, the time needed for the Amazon Books dataset (denoted as AB) is almost equal to the time needed for the MovieLens “Latest 20M” dataset (denoted as ML20), although the latter contains 2.3 times more ratings than the former. This is

again owing to the fact that the Amazon Books dataset involves a significantly higher number of users than the MovieLens "Latest 20M" (295K users vs. 138K or 2.13 times more).

In all cases we can observe that the computation of all Pearson similarities requires from 0.3 ms (for the Ciao dataset, labeled as CI) to 75 min for a dataset containing 100M ratings (the Netflix dataset, labeled as NF), hence it is considered feasible. Since computation of Pearson similarities is clearly parallelizable (computations for any pair of users can proceed parallelly with the computation of any other pair), adding more execution cores can further reduce the time needed.

Rating Prediction Computation Phase. Rating prediction computation in both the plain CF and the dynamic average-based algorithms is performed using the following prediction formula [3]:

$$\widehat{r_{u,i}} = \overline{r_u} + \frac{\sum_{v \in NN(u)} Pearson\_sim(u,v) * r_{v,i}}{\sum_{v \in NN(u)} |Pearson\_sim(u,v)|} \tag{8}$$

where $\widehat{r_{u,i}}$ is the prediction for user *u*'s rating for item *i* and *NN(u)* is the set of nearest neighbors for user *u*. The complexity of this formula is equal to $O(|NN|)$, with $|NN|$ denoting the cardinality of the nearest neighbor set.

Figure 9 presents the time needed to compute a rating prediction, in relation to the number of ratings in the database.

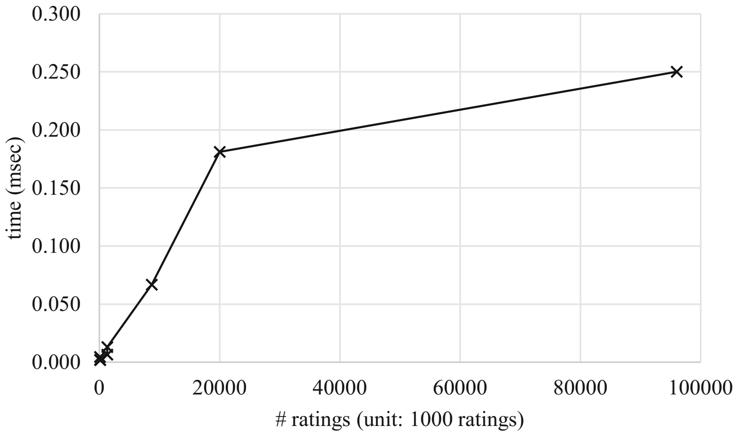

Fig. 9. Time needed for computing a rating prediction for a user

We can observe that the recommendation formulation time is under 1 ms in all cases. The time needed to compute a rating prediction increases with the number of ratings, owing to the fact that in our experiments the set of nearest neighbors was allowed to contain all users having a positive Pearson correlation with the user that the prediction was computed for. Pruning the nearest neighbor set to contain the top *N* similar users or users having a similarity above a higher threshold would render the computation even more efficient.

Database Update Phase. The ratings database is updated when users enter new ratings. When a rating is entered, the average(s) related to the user's ratings are modified, and additionally the user's Pearson similarity to other users changes. In the following subsections, we elaborate on the complexity and scalability of the database update phase, under the four algorithms considered in the evaluation section.

Updating a user's averages.
Under the plain CF algorithm, when a new rating r_{new} is entered in the database, the new global average can be computed using Eq. 9:

$$avg_{u,new} = \frac{avg_{u,current} * |ratings_u| + r_{new}}{|ratings_u| + 1} \tag{9}$$

where $ratings_u$ is the set of ratings that have been entered by user *u* before r_{new} was entered and $avg_{u,current}$ is the current global average of user *u*. Therefore, the complexity of updating a user's global average upon insertion of a new rating is O(1).

When the $DA_{previous}$ algorithm is employed, the arrival of a new rating r_{new} entered by user *u* necessitates only the computation of the dynamic average for the newly entered rating: indeed, since the dynamic average of any rating *r* is based only on ratings entered by the same user *before r*, the dynamic averages of ratings previously entered by the same user are not affected. If r_{last} is the last rating entered by user *u* before r_{new}, then the dynamic average for r_{new} can be computed using Eq. 10:

$$DA_{previous}(r_{new}) = \frac{DA_{previous}(r_{last}) * |ratings_u| + r_{new}}{|ratings_u| + 1} \tag{10}$$

where $ratings_u$ is the set of ratings that have been entered by user *u* before r_{new} was entered. Consequently, the complexity of updating a user's dynamic averages upon insertion of a new rating is O(1).

When the DA_{next} algorithm is used, the arrival of a new rating r_{new} entered by user *u* necessitates the computation of the dynamic averages for *all* ratings entered by *u*: this is due to the fact that the newly entered rating r_{new} has a greater timestamp than all existing ratings *r* entered by *u*, and therefore by virtue of Eq. 4 it affects the respective dynamic averages. To recalculate the dynamic averages of all ratings, the algorithm shown in Listing 1 can be employed. However, since the ratings are already ordered in ascending timestamp order, the sorting operation can be skipped, and therefore the complexity is reduced from $O(r_u * log(r_u))$ to $O(r_u)$, accounting for a single traversal of the ratings in descending timestamp order.

Under the $DA_{vicinity}$ algorithm, when a new rating r_{new} is entered by user *u*, the dynamic averages for *all* ratings entered by *u* must be calculated anew. This is because (a) since the newly entered rating will have a timestamp greater than all other ratings in the database, the denominator of Eq. 5 changes, and consequently the weight of every rating is modified and (b) the newly entered rating should be considered in the computation of the dynamic average of every rating entered by user *u*, according to Eq. 6. As shown in subsection "Bootstrap phase", the complexity of computing the dynamic averages for all ratings entered by some user *u* is $O(r_u^2)$, where r_u is the number of ratings entered by *u*.

Figure 10 illustrates the time needed for recomputing a user's dynamic averages in the context of the datasets used in this paper. We observe that the time needed for DA_{next} algorithm ranges between 0.54 and 10 μsec, while the time needed by the $DA_{previous}$ and the plain CF algorithm remains constant. Finally, the $DA_{vicinity}$ algorithm exhibits the worst performance among the examined algorithms.

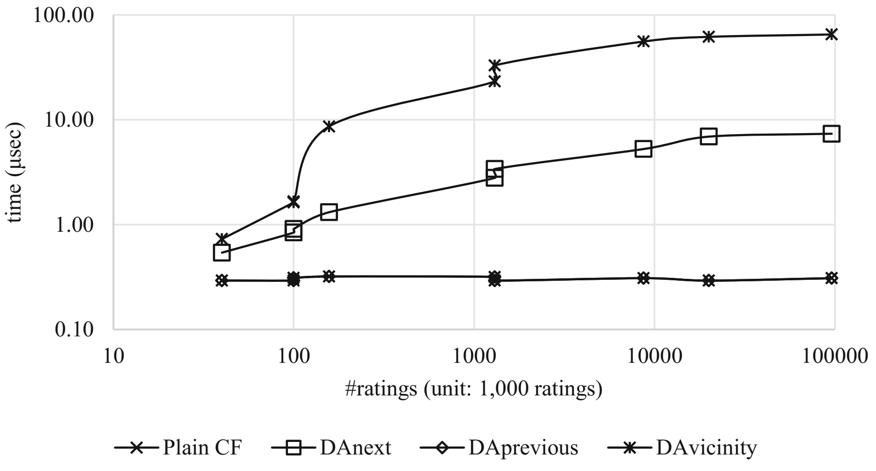

Fig. 10. Time needed for recomputing a user's average(s)

Recomputing a User's Pearson Similarities. When a new rating is entered, the Pearson similarities between a user and other users need to be recomputed. For the plain CF and the $DA_{previous}$ algorithm, where the global average and the dynamic averages respectively are not affected, this needs to be performed only for users that have rated the item referenced in the newly entered rating. For the DA_{next} and $DA_{vicinity}$ algorithms, similarities with all users need to be computed afresh, because the dynamic averages of all ratings change, and this affects the outcome of the dynamic average-aware Pearson similarity (c.f. Eq. 2).

Figure 11 depicts the performance of the procedure for recomputing the Pearson similarity between the user for which a new rating was entered and all other users in the dataset. For the Netflix dataset, which contains 480K users, the time needed is approximately 30 ms. It has to be noted that parallelism is not as efficient in this case as it has proven in the case of computing the pairwise similarities between all users, because the time needed to actually perform the calculations is now much smaller, therefore the overhead of thread creation is now a considerable portion in the overall execution time.

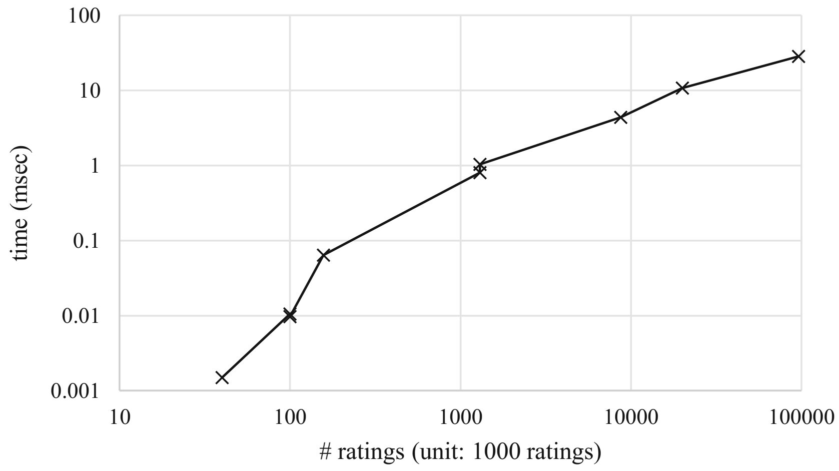

Fig. 11. Time needed to recompute the Pearson similarities between a user and all other users

Besides having a high computation cost, recomputing the Pearson similarities upon the arrival of each new rating is of low utility, because the results of some of these computations will be overwritten almost instantly, as new ratings by other users enter the database. For these reasons, the recomputation of the Pearson similarities can be performed in an offline fashion, either periodically or when a substantial number of new ratings has been amassed.

5 Conclusion and Future Work

In this paper we have introduced a novel dynamic average-based algorithm, DA_{next}, which is able to better follow the variations of user rating practices and, consequently, is able to produce more accurate rating predictions. The proposed algorithm has been experimentally verified using ten datasets and compared to other dynamic average-based algorithms presented in the literature [33], under both user-user and item-item CF implementations. The DA_{next} algorithm has been found to consistently outperform all other algorithms in all tested datasets, reducing prediction errors, as reflected through the MAE and RMSE metrics, and also achieving the highest correct prediction percentages. In particular, the average MAE reduction compared to the plain CF algorithm is 5.1% under the user-user CF implementation and 6.0% under the item-item CF implementation; the respective gains regarding the RMSE metric are 5.3% and 7.1%. In comparison to the runner up algorithm, i.e. the $DA_{previous}$ algorithm [33], the improvements in the MAE are 1.5% under the user-user CF implementation scenario and 2.6% under the item-item implementation scenario, while the respective gains in the RMSE are quantified to 1.5% and 3.0%. Considering the correct prediction metric, the proposed algorithm outperforms the runner up algorithm ($DA_{previous}$) by 1.09% under the user-user CF implementation scenario and by 1.2% under the item-item implementation scenario. These benefits are realized at the expense of drops in coverage, which range from negligible to tolerable; coverage drops by 0.7% on average, being less than 2.2% in all cases. For datasets with higher density, in particular, practically no coverage loss occurs. We have also compared the improvements in accuracy achieved by the proposed algorithm against the corresponding improvements achieved by other algorithms exploiting temporal dynamics, and DA_{next} has been found to achieve the most substantial improvements, in all cases.

Our future work will focus on investigating alternative methods for computing the dynamic averages, as well as employing different dynamic average techniques for different users, depending on the timestamp distribution of their rating history. The adaptation of other similarity metrics, such as the Euclidian distance and the Manhattan distance [49], to exploit information regarding identified shifts in rating practices will be investigated.

The matrix factorization technique [32] and the fuzzy recommender systems approach [50] are also particularly interesting areas for further research on how shifts in rating practices can be accommodated in these approaches. For the matrix factorization technique, in particular, the approach of time-aware matrix factorization models [44, 45] will be studied.

Since the DA_{next} algorithm introduced in this paper targets shifts in user rating practices, which is a distinct aspect than those addressed in other approaches (e.g. interest shifts; decay of old-aged ratings; etc.), opportunities exist for combining the algorithm presented in our paper with algorithms from other categories, so that even higher accuracy improvements can be harvested; such combinations will be investigated in our future work.

Finally, exploring methods for decreasing the space overhead for the implementation of dynamic averages, considering the maintenance of dynamic averages at a coarser granularity than the individual rating, such as monthly or yearly, as well as decreasing the need for recomputing dynamic averages due to the arrival of new ratings (e.g. periodic recomputation or the consideration of ratings only in a specific temporal vicinity, when computing dynamic averages), will be explored.

References

1. Balabanovic, M., Shoham, Y.: Fab: content-based, collaborative recommendation. Commun. ACM **40**(3), 66–72 (1997)
2. Dror, G., Koenigstein, N., Koren, Y.: Yahoo! music recommendations: modeling music ratings with temporal dynamics and item taxonomy. In: Proceedings of the 5th ACM Conference on Recommender Systems (RecSys 2011), New York, NY, USA, pp. 165–172 (2011)
3. Schafer, J.B., Frankowski, D., Herlocker, J., Sen, S.: Collaborative filtering recommender systems. In: Brusilovsky, P., Kobsa, A., Nejdl, W. (eds.) The Adaptive Web. LNCS, vol. 4321, pp. 291–324. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-72079-9_9

4. Herlocker, J.L., Konstan, J.A., Terveen, L.G., Riedl, J.T.: Evaluating collaborative filtering recommender systems. ACM Trans. Inf. Syst. **22**(1), 5–53 (2004)
5. Li, L., Zheng, L., Yang, F., Li, T.: Modeling and broadening temporal user interest in personalized news recommendation. Expert Syst. Appl. **41**(7), 3168–3177 (2014)
6. Burke, R.: Hybrid recommender systems: Survey and experiments. User Model. User-Adap. Interact. **12**(4), 331–370 (2002)
7. McAuley, J.J., Pandey, R., Leskovec, J.: Inferring networks of substitutable and complementary products. In: Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Sydney, NSW, Australia, pp. 785–794 (2015)
8. McAuley, J., Targett, C., Shi, J., van den Hengel, A.: Image-based recommendations on styles and substitutes. In: Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval, Santiago, Chile, pp. 43–52 (2015)
9. MovieLens datasets. http://grouplens.org/datasets/movielens/. Accessed 22 Sept 2017
10. Harper, F.M., Konstan, J.A.: The MovieLens datasets: history and context. ACM Trans. Interact. Intell. Syst. (TiiS) **5**(4), 19 (2015). Article No. 19
11. Zhou, Y., Wilkinson, D., Schreiber, R., Pan, R.: Large-scale parallel collaborative filtering for the netflix prize. In: Fleischer, R., Xu, J. (eds.) AAIM 2008. LNCS, vol. 5034, pp. 337–348. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-68880-8_32
12. Margaris, D., Vassilakis, C.: Pruning and aging for user histories in collaborative filtering. In: Proceedings of the 2016 IEEE Symposium Series on Computational Intelligence, Athens, Greece, pp. 1–8 (2016)
13. Liu, N.N., He, L., Zhao, M.: Social temporal collaborative ranking for context aware movie recommendation. ACM Trans. Intell. Syst. Technol. (TIST) **4**(1) (2013). Article No. 15
14. Bakshy, E., Rosenn, I., Marlow, C., Adamic, L.: The role of social networks in information diffusion. In: Proceedings of the 21st International Conference on World Wide Web, Lyon, France, pp. 519–528 (2012)
15. Koren, Y.: Factor in the neighbors: scalable and accurate collaborative filtering. ACM Trans. Knowl. Discov. Data (TKDD) **4**(1) (2010). Article No. 1
16. Vaz, P.C., Ribeiro, R., de Matos, D.M.: Understanding temporal dynamics of ratings in the book recommendation scenario. In: Proceedings of the 2013 International Conference on Information Systems and Design of Communication, ACM ISDOC 2013, New York, NY, USA, pp. 11–15 (2013)
17. Nishida, K., Yamauchi, K.: Detecting concept drift using statistical testing. In: Corruble, V., Takeda, M., Suzuki, E. (eds.) DS 2007. LNCS (LNAI), vol. 4755, pp. 264–269. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-75488-6_27
18. Liu, H., Hu, Z., Mian, A., Tian, H., Zhu, X.: A new user similarity model to improve the accuracy of collaborative filtering. Knowl.-Based Syst. **56**, 156–166 (2014)
19. Minku, L.L., White, A.P., Yao, X.: The impact of diversity on online ensemble learning in the presence of concept drift. IEEE Trans. Knowl. Data Eng. **22**(5), 730–742 (2010)
20. Elwell, R., Polikar, R.: Incremental learning of concept drift in nonstationary environments. IEEE Trans. Neural Netw. **22**(10), 1517–1531 (2011)
21. Ang, H.H., Gopalkrishnan, V., Zliobaite, I., Pechenizkiy, M., Hoi, S.C.H.: Predictive handling of asynchronous concept drifts in distributed environments. IEEE Trans. Knowl. Data Eng. **25**(10), 2343–2355 (2013)
22. Zliobaite, I., Bakker, J., Pechenizkiy, M.: Beating the baseline prediction in food sales: how intelligent an intelligent predictor is? Expert Syst. Appl. **39**(1), 806–815 (2012)

23. Vaz, P.C., Ribeiro, R., DeMatos, D.M.: Understanding temporal dynamics of ratings in the book recommendation scenario. In: Proceedings of the 2013 International Conference on Information Systems and Design of Communication (ISDOC 2013), Lisbon, Portugal, pp. 11–15 (2013)
24. Gama, J., Zliobaite, I., Bifet, A., Pechenizkiy, M., Bouchachia, A.: A survey on concept drift adaptation. ACM Comput. Surv. **1**(1) (2013). Article No. 1
25. Margaris, D., Vassilakis, C., Georgiadis, P.: Recommendation information diffusion in social networks considering user influence and semantics. Soc. Netw. Anal. Mining **6**(108), 1–22 (2016)
26. Ekstrand, M.D., Riedl, J.T., Konstan, J.A.: Collaborative filtering recommender systems. Found. Trends Hum.-Comput. Interact. **4**(2), 81–173 (2011)
27. He, D., Wu, D.: Toward a robust data fusion for document retrieval. In: Proceedings of the 4th IEEE International Conference on Natural Language Processing and Knowledge Engineering (NLP-KE), Beijing, China, pp. 1–8 (2008)
28. Shardanand, U., Maes, P.: Social information filtering: algorithms for automating "word of mouth". In: Proceedings of the 1995 SIGCHI Conference on Human Factors in Computing Systems, Denver, Colorado, USA, pp. 210–217 (1995)
29. Margaris, D., Vassilakis, C., Georgiadis, P.: Knowledge-based leisure time recommendations in social networks. In: Alor-Hernández, G., Valencia-García, R. (eds.) Current Trends on Knowledge-Based Systems. Intelligent Systems Reference Library, vol. 120, pp. 23–48. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-51905-0_2
30. Yu, K., Schwaighofer, A., Tresp, V., Xu, X., Kriegel, H.P.: Probabilistic memory-based collaborative filtering. IEEE Trans. Knowl. Data Eng. **16**(1), 56–69 (2004)
31. Dias, R., Fonseca, M.J.: Improving music recommendation in session-based collaborative filtering by using temporal context. In: Proceedings of the 25th IEEE International Conference on Tools with Artificial Intelligence, Herndon, VA, pp. 783–788 (2013)
32. Koren, Y., Bell, R., Volinsky, C.: Matrix factorization techniques for recommender systems. IEEE Comput. **42**(8), 42–49 (2009)
33. Margaris, D., Vassilakis, C.: Improving collaborative filtering's rating prediction quality by considering shifts in rating practices. In: Proceedings of the 19th IEEE International Conference on Business Informatics, Thessaloniki, Greece, vol. 01, pp. 158–166 (2017)
34. Bao, J., Zheng, Y., Mokbel, M.: Location-based and preference-aware recommendation using sparse geo-social networking data. In: Proceedings of the 20th International Conferences on Advances in Geographic Information Systems (SIGSPATIAL 2012), Redondo Beach, California, pp. 199–208 (2012)
35. Zheng, Y., Xie, X.: Learning travel recommendations from user-generated GPS traces. ACM Trans. Intell. Syst. Technol. (TIST) **2**(1), 29 (2011). Article No. 2
36. Gong, S.: A collaborative filtering recommendation algorithm based on user clustering and item clustering. J. Softw. **5**(7), 745–752 (2010)
37. Das, A., Datar, M., Garg, A., Rajaram, S.: Google news personalization: scalable online collaborative filtering. In: Proceedings of the 16th International Conference on World Wide Web, Banff, Alberta, Canada, pp. 271–280 (2007)
38. Margaris, D., Vassilakis, C.: Enhancing user rating database consistency through pruning. Trans. Large-Scale Data Knowl.-Centered Syst. **XXXIV**, 33–64 (2017)
39. Ramezani, M., Moradi, P., Akhlaghian, F.: A pattern mining approach to enhance the accuracy of collaborative filtering in sparse data domains. Phys. A: Stat. Mech. Appl. **408**, 72–84 (2014)
40. Najafabadi, M.K., Mahrin, M.N., Chuprat, S., Sarkan, H.M.: Improving the accuracy of collaborative filtering recommendations using clustering and association rules mining on implicit data. Comput. Hum. Behav. **67**, 113–128 (2017)

41. Li, C., Shan, M., Jheng, S., Chou, K.: Exploiting concept drift to predict popularity of social multimedia in microblogs. Inf. Sci. **339**, 310–331 (2016)
42. Lu, Z., Pan, S.J., Li, Y., Jiang, J., Yang, Q.: Collaborative evolution for user profiling in recommender systems. In: Proceedings of the 25th International Joint Conference on Artificial Intelligence (IJCAI 2016), pp. 3804–3810 (2016)
43. Kangasrääsiö, A., Chen, Y., Głowacka, D., Kaski, S.: Interactive modeling of concept drift and errors in relevance feedback. In: Proceedings of the 2016 Conference on User Modeling Adaptation and Personalization (ACM UMAP 2016), New York, NY, USA, pp. 185–193 (2016)
44. Gantner, Z., Rendle, S., Schmidt-Thieme, L.: Factorization models for context-/time-aware movie recommendations. In: Proceedings of the Workshop on Context-Aware Movie Recommendation (ACM CAMRa 2010), New York, NY, USA, pp. 14–19 (2010)
45. Zhang, J.D., Chow, C.Y.: TICRec: a probabilistic framework to utilize temporal influence correlations for time-aware location recommendations. IEEE Trans. Serv. Comput. **9**(4), 633–646 (2016)
46. Sarwar, B., Karypis, G., Konstan, J., Riedl, J.: Item-based collaborative filtering recommendation algorithms. In: Proceedings of the 10th International Conference on World Wide Web (WWW10), Hong Kong, pp. 285–295 (2001)
47. Winoto, P., Tang, T.Y.: The role of user mood in movie recommendations. Expert Syst. Appl. **37**, 6086–6092 (2010)
48. DELL: PowerEdge R930 Rack Server specs. http://www.dell.com/us/business/p/poweredge-r930/pd?ref=PD_OC. Accessed 22 Feb 2018
49. Cha, S.-H.: Comprehensive survey on distance/similarity measures between probability density functions. Int. J. Math. Models Methods Appl. Sci. **1**(4), 300–307 (2007)
50. Son, L.H.: HU-FCF: a hybrid user-based fuzzy collaborative filtering method in recommender systems. Expert Syst. Appl. **41**, 6861–6870 (2014)
51. Guo, G., Zhang, J., Thalmann, D., Yorke-Smith, N.: ETAF: an extended trust antecedents framework for trust prediction. In: Proceedings of the 2014 International Conference on Advances in Social Networks Analysis and Mining ASONAM 2014, Beijing, China, pp. 540–547 (2014)
52. Lo, Y.-Y., Liao, W., Chang, C.-S., Lee, Y.-C.: Temporal matrix factorization for tracking concept drift in individual user preferences. IEEE Trans. Comput. Soc. Syst. **5**(1), 156–168 (2018)
53. Cheng, W., Yin, G., Dong, Y., Dong, H., Zhang, W.: Collaborative filtering recommendation on users' interest sequences. PLoS One **11**(5), e0155739 (2016)