Expert Systems with Applications xxx (2013) xxx–xxx

Contents lists available at SciVerse ScienceDirect

Expert Systems with Applications

journal homepage: www.elsevier.com/locate/eswa

Integrating similarity and dissimilarity notions in recommenders

Christos Zigkolis *, Savvas Karagiannidis, Ioannis Koumarelas, Athena Vakali

Department of Informatics, Aristotle University of Thessaloniki, 54124 Thessaloniki, Greece

ARTICLE INFO

Keywords:
Recommender systems
Dissimilarity recommender
Distributed framework

ABSTRACT

Collaborative recommenders rely on the assumption that *similar* users may exhibit *similar* tastes while content-based ones favour items that found to be *similar* with the items a user likes. Weak related entities, which are often considered to be useful, are neglected by those similarity-driven recommenders. To take advantage of this neglected information, we introduce a novel dissimilarity-based recommender that bases its estimations on degrees of dissimilarities among items' attributes. However, instead of using the proposed recommender as a stand-alone method, we combine it with similarity-based ones to maintain the selective nature of the latter while detecting, through our recommender, information that may have been overlooked. Such combinations are established by IANOS, a proposed framework through which we increase the accuracy of two popular similarity-based recommenders (Naive Bayes and Slope-One) after their combination with our algorithm. Improved accuracy results in experimentation on two datasets (Yahoo! Movies and Movielens) enhance our reasoning. However, the proposed recommender comes with an additional computational complexity when combined with other techniques. By using Hadoop technology, we developed a distributed version of IANOS through which execution time was reduced. Evaluation on IANOS procedures in terms of time performance endorses the use of distributed implementations.

1. Introduction

Recommenders search in collections of items (e.g. products, services or people) and suggest to users the most relevant ones (Resnick and Varian, 1997). The level of relevance of an item for a user is usually expressed by a rating, i.e. a degree of user's appreciation on it. By utilizing users' ratings, a recommender aims at estimating relevance of items towards extracting suggestions. In this direction, recommenders act as predictors which, based on their approach of estimating ratings, can be classified into the three following types: (a) content-based methods (Lops et al., 2011) where a user receives items similar to the ones preferred in the past, (b) collaborative methods (Su and Khoshgoftaar, 2009) where a user gets recommendations extracted by other users with similar preferences, and (c) hybrid methods (Burke, 2002) where content-based and collaborative types are combined.

A way to assess the predictive power of a recommender is to use a set of pre-rated items for which the recommender will provide estimations. In such a case, the recommender's success is based on how close its estimations are to the actual ratings the users gave—the more close, the more accurate the estimations are. Therefore, we can evaluate a recommender by calculating the *accuracy* (i.e. closeness) of the provided estimations. Towards increasing accuracy, many recommenders exhibit several limitations that negatively affect their estimations (Adomavicius and Tuzhilin, 2005). One limitation that is common to all the three aforementioned types is the *new-user problem* (a special case of cold-start problem) where a user has rated a limited number of items. In this case, a recommender is incapable of building models that represent users' real preferences which may result into poor (inaccurate) recommendations. Although accuracy has been criticized for its role in evaluation purposes (Bobadilla et al., 2011; McNee et al., 2006), it still remains a significant metric that has been extensively used in other studies to estimate recommender's predictive power. In this work, we provide solutions that deal with the accuracy and also alleviate the new-user limitation that negatively affects it.

Accuracy is reached typically by looking at similarities among entities (e.g. users, items) and the notion of similarity is highly used in all types of recommenders for providing suggestions. For instance, in collaborative methods, *Pearson Correlator* and *Cosine Distance* are two well-known functions which consider users' ratings to assess similarities either between users (Resnick et al., 1994) or items (Linden et al., 2003). Similarity is also apparent in content-based methods where similarities between items are estimated by taking into account their attributes (e.g. genres in movies, categories in articles, authors in books). Similarity-driven

* Corresponding author. Tel.: +30 2310991865.
E-mail addresses: chzigkol@csd.auth.gr (C. Zigkolis), sakaragi@csd.auth.gr (S. Karagiannidis), koumarel@csd.auth.gr (I. Koumarelas), avakali@csd.auth.gr (A. Vakali).

0957-4174/$ - see front matter © 2013 Elsevier Ltd. All rights reserved.
http://dx.doi.org/10.1016/j.eswa.2013.03.018

recommenders neglect weak related entities such as users of different tastes or items with different attributes, which may often offer useful information.

To take advantage of this overlooked information, we present a novel recommender that incorporates the notion of *dissimilarity* as a metric which takes into account users' ratings and calculates pairwise degrees of difference between items' attributes. For instance, from users' rating actions we may observe that movie genres *Romance* and *Adventure* are quite dissimilar (i.e. users who like "Romance" movies do not show the same appreciation to the "Adventure" ones). Those degrees of difference indicate characteristic, consistent and representative negative relations among attributes utilized by the proposed recommender towards rating estimations.

Instead of using our dissimilarity recommender as a standalone predictor, in this work, we combine it with similarity-based techniques. With that, we both maintain the selective nature of the similarity-based methods and utilize our recommender to detect information that they may have overlooked. To achieve that, we present a new framework, called *IANOS*,[1] which integrates the two "faces" of *similarity* and *dissimilarity* by providing a post-hybridization way of combining rating estimations from different recommenders.

1.1. Motivation for dissimilarity

The rationale behind the combination of similarity and dissimilarity is driven by studies of cognitive psychology which state that in comparison processes (judgements) both concepts of similarity and dissimilarity share equal importance. Mussweiler's study (Mussweiler, 2003) explains the significance of putting dissimilarity along with similarity when it comes to judging. Mussweiler supports that every judgement can be an outcome of a comparison process which involves the selection of an example to be compared with a target. He also suggests that while some people tend to select examples *similar* to the target, others may choose quite *dissimilar* ones. These two different comparison processes (similarity and dissimilarity) can be seen as important sources of knowledge when it comes to reaching a final decision. In the context of recommendations, by assuming that a rating from a user is actually a judgement on an item, it might be interesting to treat similarity and dissimilarity as two supplementary concepts rather than as two contradictory sources of knowledge that can be utilized independently.

Dissimilarities relevance with a recommender's accuracy improvement is obvious in real-world cases such as in movie recommenders. In such cases, users could be divided into two distinct groups: the *open-scope users* that rate movies of various genres without being definitively in favour of a particular ones, and the *narrow-scope users* that their ratings show favour to a subgroup of genres. In this context, we consider two scenarios. In a scenario with a collaborative recommender, we consider an open-scope user and a narrow-scope one with common ratings to each other which means that rated items from the former are candidates to be suggestions to the latter. In such a case, it is likely a movie to be recommended to narrow-scope user who show either no preference or negative preference to the genre the movie belongs to. In another scenario with a content-based recommender, a narrow-scoped user could give a positive rating on a movie with a genre which is not in line with her preferences. If there are none prior negative ratings for this genre, the recommender would be affected by this "outlier" rating and suggest movies with this genre to the user.

It is likely that accuracy to be reduced in both of the aforementioned indicative scenarios, because both recommenders will suggest movies with genres that are not positively preferred by the users. This fact may result in poor suggestions with negative effects on accuracy. This work is motivated by the fact that such poor suggestions can be prevented, once we identify items that belong to genres that are not in line with the users preferences. By not providing such suggestions, we can positively affect the accuracy.

1.2. Main contributions

The main contributions of this work are summarized into three major tasks: (i) we introduce a metric for calculating pairwise dissimilarity values among items' attributes, (ii) we propose a novel dissimilarity-based recommender, and (iii) we develop the IANOS framework through which we can combine existing similarity-based recommenders with the proposed one. Each task is highlighted in more detail in the next paragraphs.

1.2.1. A new metric for dissimilarity values between attributes

In this work, we search for underlying relations that indicate characteristic and consistent pairwise degrees of dissimilarities between items' attributes. In order to quantify these relations, we introduce a metric which takes users' preferences as input and extracts numeric values that express degrees of dissimilarity. To calculate a dissimilarity value between two attributes, we take the items that include them and by analyzing users' preferences on these items we conclude to a single value which indicates a summarized difference of preference between the two attributes. All those pairwise values form an attribute-based pairwise dissimilarity matrix.

1.2.2. A novel dissimilarity-based recommender

We introduce a new recommender in which, with the use of the aforementioned dissimilarity matrix, we build a training model characterized by a set of attribute-based vector spaces. Each space is intended for one attribute and the rated items with that attribute are mapped into the space through the use of *two new features* that we propose. One feature is extracted from the dissimilarity matrix and the other from user's atomic preferences. The rationale behind these features is that when it comes to estimating a rating for an item for a single user, our recommender discovers the relation of the item's attributes with both the user's preferences (local view) and all users' preferences (global view) in those attributes. These vector spaces act as input for our recommender's process of rating estimations which involves search operations for nearest rated items and make use of their rating information in order to provide estimations.

1.2.3. The IANOS framework

IANOS combines our recommender with existing similarity-based recommenders. It follows a post-hybridization way of combining results, a common process in the weighted hybrid recommenders (Burke, 2002). In such cases, each recommender provides its own rating estimations which are then combined into one final estimation for each unrated item. This virtual-based combinations of recommenders (Karagiannidis et al., 2010) enables us to evaluate our recommender when combined with different similarity-based ones. Although IANOS aims to improve the accuracy of the suggested estimations, we do not intent to achieve that at the expense of computational complexity. Since our recommender adds additional computational cost to the whole process, we have implemented a distributed version of IANOS through which we split the load into many work stations and reduce the overall time the framework needs to complete a run.

[1] IANOS is the name of a Greek God with two opposite looking faces.

Please cite this article in press as: Zigkolis, C., et al. Integrating similarity and dissimilarity notions in recommenders. *Expert Systems with Applications* (2013), http://dx.doi.org/10.1016/j.eswa.2013.03.018

To test the effectiveness of IANOS, we make use of two movie datasets (Yahoo! Movies[2] and Movielens[3]) on which we apply the proposed recommender along with two popular similarity-based techniques, Naive Bayes (Duda and Hart, 1973) and Slope-One (Lemire and Maclachlan, 2007). To assess the predictive power of the recommenders, we make use of the RMSE (Root Mean Square Error) metric (Gunawardana and Meek, 2009) which enables us to measure the differences between their rating estimations and the actual ratings the users gave. With our combined approach we intent to reduce the overall error in rating estimations and provide more accurate recommendations. Moreover, we evaluate IANOS distributed implementation in terms of time performance in order to test its scalability in situations where different sizes of data are applied.

The rest of the paper is structured as follows. Section 2 briefly discusses several related studies. In Section 3, we present the data modeling process of our recommender while in Section 4 our new metric for the extraction of dissimilarity matrix is described. Section 5 presents both the training model and the process of estimating ratings of the proposed recommender. Section 6 describes the design of IANOS framework and the post-hybridization way of combining the estimations from different recommenders. Section 7 shows our experimental results and in Section 8 the conclusions and future work are highlighted.

2. Related work

This paper leverages existing work from recommenders which deal with the concept of dissimilarity and it is moreover relevant with research efforts which utilize the weighted-hybrid techniques.

2.1. Dissimilarity in recommenders

The dissimilarity concept has previously been used in recommenders typically via a *metric* which calculates degrees of dissimilarity among entities such as users and/or items. Those dissimilarities are used as helpful hints towards providing more suitable recommendations.

Dissimilarities among users are apparent in group recommenders (McCarthy et al., 2006) in which suggestions are extracted for a group of users instead of individuals. For example, in Gartrell et al. (2010), the authors introduce a novel group consensus function that integrates social, expertise and interest dissimilarity among the members of the group in order to enhance group recommendations. Moreover, in Amer-Yahia et al. (2009) the incorporation of disagreements on the same items among group members is considered to be critical for the effectiveness of group recommendations.

Dissimilarities among items have also been utilized in recommenders. For instance, in Kagie et al. (2008) the authors introduce a weighted dissimilarity measure to find item-to-item differences with attribute weights being determined by the clickstream data of an e-commerce site. In addition, in De et al. (2012) the authors approach the problem of finding similar items for the recommendation task by creating a graph network of items where edges indicate the existence of common ratings for two items by a certain number of users. In the process of finding a degree of difference between two items, the authors propose a learning dissimilarity function that takes the different attributes of the two items and assigns weights on them. The less dissimilar items to a target one are presented as suggestions.

In another study (Bezerra and de Carvalho, 2004), the authors propose a recommender which creates user profiles with symbolic descriptions (Leite Dantas Bezerra and Tenorio de Carvalho, 2011) that summarize information from previously rated items. A dissimilarity metric is used in order to calculate degrees of difference between user profiles and items. Eventually, the recommender suggests items that have low degree of dissimilarity.

In our approach, relations between items' attributes play a significant role towards estimating a rating for an item for a single user where this estimation is carried out by discovering the relation of the item's attributes with all users' preferences in them. Therefore, our new metric, contrary to the aforementioned studies where users and items are the main entities, focuses on attributes and their dissimilarities with the objective to find informative relations that would help in the rating estimation process.

2.2. Weighted hybrid recommenders

Hybrid recommenders are further categorized into three types, the weighted, the switching and the mixed ones (Burke, 2002). A weighted hybrid recommender uses a weighted function to combine the estimations coming from the different recommenders, a switching hybrid recommender uses criteria to switch between recommendation techniques and in a mixed hybrid recommender the list of suggestions contains items coming from several different recommenders. Here, we focus on weighted hybrid recommenders, because their post-hybridization approach in combining results is tailored to the IANOS approach in combining the two "faces" of similarity and dissimilarity. The next indicative research efforts summarize relevant work in weighted-hybrid recommenders towards solving the accuracy problem.

The simplest weighted hybrid recommender uses a linear combination over rating estimations from different recommenders. Claypool et al. (1999) presented P-Tango, a system that initially gives collaborative and content-based recommenders equal weights, but gradually adjusts the weighting as estimations about ratings are confirmed. A linear-weighted hybrid method is also proposed by Gemmell et al. (2012) in online resources from social annotation systems, where multiple complementary components are combined into a single integrated model that provides flexibility while it takes into consideration the characteristics of items across different social annotation systems. Furthermore, in Kunaver et al. (2007), a weighted hybrid system is proposed based on three different collaborative recommenders which although they can separately provide adequate results, their combination into a unified system shows greater stability as precision and recall results indicate.

In most cases, the weights assigned to different recommenders are based either on their previously recorded accuracy (i.e. accuracy when they are applied separately) or they are dynamically defined as in Bellogin et al. (2011) study. These post-hybridization ways of combining results aim to finally achieve better rating estimations by giving higher weights to recommenders that are more accurate than the others. In our case, we make use of predefined weights in order to examine how the accuracy is affected when more emphasis is given to either the similarity-based or the dissimilarity-based method. With this way, we test whether the combination of the similarity and dissimilarity recommenders has a practical use which means that the accuracy is improved after such a combination.

3. Data modeling

We consider a set of items $I = \{item_1, item_2, \ldots, item_K\}$ along with their attributes identified by a set of categories $C = \{c_1, c_2, \ldots, c_L\}$.

[2] Yahoo! Movies R4—http://webscope.sandbox.yahoo.com/
[3] Movielens 10 M—http://www.grouplens.org/node/73

Please cite this article in press as: Zigkolis, C., et al. Integrating similarity and dissimilarity notions in recommenders. *Expert Systems with Applications* (2013), http://dx.doi.org/10.1016/j.eswa.2013.03.018

Each item is described by a subset of categories, thus $item_i = \{c_1, c_2, \ldots, c_m\} \subseteq C$ where $i = [1,2,\ldots,K]$ and $m \leqslant L$. Here, we consider categories as the only set of attributes to facilitate discussion on the proposed approach, but any other distinct valued set of attributes (e.g. actors, directors, tags) can be supported by our work. Moreover, we maintain both a set of users $U = \{u_1, u_2, \ldots, u_N\}$ and their preferences $P = \{p_1, p_2, \ldots, p_M\}$. Each preference is a triplet with a rating of a user on an item, thus $p_i = \{u_a, item_b, r_i\}$ where $i = [1,2,\ldots,M]$, $u_a \in U$, $item_b \in I$ and $r_i \in \Re$.

A recommender-predictor can be abstractly defined by three parts: the data modeling, the training model and the rating estimation part. The first two parts make use of a subset of rated items to build the algorithm's training model utilized by the last one to provide rating estimations for the rest of the items (i.e. the ones which were not used in the training phase). This data splitting is inspired by the leave-n-out approach (Breese et al., 1998) where a percentage of the dataset is withheld from a recommender and used as test data. Here, we follow a user-centric leave-n-out approach where we partition the preferences of each user into two complementary parts the size of which is determined by a predefined percentage of preferences we want to keep for training. Therefore, we have the training set P_{train} and the test set P_{test} of preferences where $P_{train} \cup P_{test} = P$ and $P_{train} \cap P_{test} = \emptyset$.

Regarding the data modeling, we present two processes, one for users and one for categories. Modeling users' preferences involves users' level of interest in the categories, while categories modeling embeds grouping of users and preferences. Both processes facilitate the extraction of dissimilarity matrix and the training model of our recommender as described in Sections 4 and 5 respectively.

3.1. Users modeling

The primary objective here is to determine for each user the degrees of interest in the available categories. To extract those degrees, we initially focus on a single user's preferences and then proceed to their category-based grouping:

- **User's Preferences in training data**: The preferences for a single user u_a in P_{train} are represented by the subset $P_{train}(u_a) = \{\forall p_i \in P_{train} | p_i = \{u_a, item, r_i\}\}$ where $item \in I$.
- **User's Preferences on a category in training data**: Here, along with the user u_a, we filter the P_{train} set with a predefined category (i.e. $c_l \in C$). Thus, we have $P_{train}(u_a, c_l) = \{\forall p_i \in P_{train} | p_i = \{u_a, c_l \in item, r_i\}\}$ where $item \in I$.

From the last set of preferences we can extract a user's average rating in a category. For the category $c_l \in C$, we have:

$$A(u_a, c_l) = \frac{\sum_{i=1}^{|P_{train}(u_a,c_l)|} \{d(u_a) * r_i\}}{|P_{train}(u_a, c_l)|} \quad (1)$$

$\forall p_i \in P_{train}(u_a, c_l) | p_i = \{u_a, item, r_i\}$.

In Eq. (1), the $d(u_a)$ factor characterizes a user's level of pessimism (or optimism) (Schafer et al., 2007) by comparing her ratings with all the ratings in training data. It is calculated by the average value of all ratings from all users in training data divided by the average value of user's ratings:

$$d(u_a) = \frac{\sum_{p_j \in P_{train}} \{r_j\} / |P_{train}|}{\sum_{p_i \in P_{train}(u_a)} \{r_i\} / |P_{train}(u_a)|} \quad (2)$$

$\forall p_j \in P_{train} | p_j = \{u, item, r_j\}$ and $\forall p_i \in P_{train}(u_a) | p_i = \{u_a, item, r_i\}$.

Before utilizing a rating contained in preferences of P_{train} or P_{test} sets, we multiple it first with this factor of the user who gave the rating. This is done due to the fact that a rating might have different meanings for each user. For instance, whilst a 3 out of 5 rating might mean "good" for a pessimistic user whose ratings are generally low, it could mean "average" for a optimistic user who usually gives high ratings. These multiplications are normalizations which eliminate the variance of ratings by bringing them closer to the average one in order to provide a common base for all users (Lemire and Maclachlan, 2007).

Inspired by the work of Braak et al. (2009), we capture users' interest in categories with the combination of two factors, the popularity and the likeness. Popularity is determined by the percentage of items with a particular category in all user's rated items:

$$Pop(u_a, c_l) = \frac{|P_{train}(u_a, c_l)|}{|P_{train}(u_a)|} \quad (3)$$

Likeness is extracted by user's average rating value in a category divided by the summarized average rating values in all categories:

$$Lik(u_a, c_l) = \frac{A(u_a, c_l)}{\sum_{i=1}^{L} A(u_a, c_i)} \quad (4)$$

Contrary to the combination presented in Braak et al. (2009) where both factors share the same importance, we favoured an *f*-measure equation in order to calculate the overall category interest value of a user. We define an integrating function through which we can put more emphasis on either popularity or likeness:

Definition 3.1 (*The Category-Interest Integrator*).

$$CI(u_a, c_l) = \frac{(1 + \beta_1^2) * Lik(u_a, c_l) * Pop(u_a, c_l)}{\beta_1^2 * Lik(u_a, c_l) + Pop(u_a, c_l)}$$

where $\beta_1 > 0$ parameter depicts to which factor we put more emphasis on (i.e. as β_1 increases, popularity is emphasized against likeness). The Category-Interest integrator indicates not only how well the category is rated, but also how often items of that category are rated by the user. Its flexible way of favouring factors enables us to test which one leads to improved accuracy results. Finally, we proceed to a normalization step on category interest values:

$$nCI(u_a, c_l) = \frac{CI(u_a, c_l)}{\sum_{i=1}^{L} CI(u_a, c_i)}$$

After the calculation of normalized category interest values for a user, we extract the predominant category (i.e. the one with the highest value):

$$pC(u_a) = argmax_{c_l \in C}(nCI(u_a, c_l))$$

Table 1 contains all the users modeling variables.

3.2. Categories modeling

Aiming to extract degrees of dissimilarity between pairs of categories, our primary goal in this part is to group users that shares preferences regarding such pairs. Considering a pair of categories c_{l_1} and c_{l_2}, we first create two sets of users, the one contains users who have rated items with c_{l_1} category, while the second consists of users for c_{l_2} category. Then, by taking the intersection of these two sets, we gather all the users that have expressed preference in both categories. In particular, we have:

- **Category's set of users**: This set holds a subset of users that have rated items with the category in question. Thus, we have $U(c_l) = \{\forall u_a \in U | p_i = \{u_a, c_l \in item, r_i\}\}$ where $p_i \in P_{train}$ and $item \in I$.
- **Set of users for a pair of categories**: This is a set of those users that have rated items of both of the pair's categories. Thus, we have the intersection set $U(c_{l_1}, c_{l_2}) = U(c_{l_1}) \cap U(c_{l_2}) \subseteq U$.

We consider those sets of users as containers from which we start the procedure of extracting the values of the dissimilarity

Table 1
Users modeling variables.

Variables	Definition
$P_{train}(u_a) \subseteq P_{train}$	Subset of preferences for the u_a
$P_{train}(u_a, c_l) \subseteq P_{train}$	Subset of preferences for the u_a for the category c_l
$A(u_a, c_l) \in \Re$	Average value of the user u_a for the category c_l
$d(u_a) \in \Re$	Ratio value for the user u_a
$nCI(u_a, c_l) \in [0,1]$	Normalized category interest value for category c_l of user u_a
$pC(u_a) \in C$	The predominant category of user u_a

matrix (see Section 4). Finally, we also group users' preferences per category for a later stage when the category-based vector spaces of our recommender's training model are formed (see Section 5.1). More specifically, for each category we define a set of preferences: $P_{train}(c_l) = \{\forall \quad p_i \in P_{train} | p_i = \{u, c_l \in item, r_i\}\}$ where $item \in I$ and $u \in U$. All variables of categories modeling are included in the Table 2.

Table 2
Categories modeling variables.

Variables	Definition
$U(c_l) \subseteq U$	Subset of users that have rated at least one item with the category c_l
$U(c_{l_1}, c_{l_2}) \subseteq U$	Subset of users that have rated items that contain either the one c_{l_1} or the other category c_{l_2}
$P_{train}(c_l) \subseteq P_{train}$	Subset of preferences on items with the category c_l

4. Estimating pairwise category-based dissimilarities

Here, we introduce a new metric which discovers pairwise negative relations in terms of preferences between items' categories. To quantify these relations, we take into account users' normalized category interest values from which summarized differences in interest for pairs of categories can be calculated. The proposed metric is applied to each pair and populates a dissimilarity matrix $D_{[L \times L]}$.

Considering two arbitrary categories c_i and c_j, we start the process of calculating their dissimilarity value starts with the set of users $U(c_i, c_j)$. Then, we take into account only those users who have shown greater interest in the category c_i than the category c_j. Thus, we end up with a smaller set of users $U_{nCI}(c_i, c_j) \subseteq U(c_i, c_j)$. Note that, $U_{nCI}(c_i, c_j) \neq U_{nCI}(c_j, c_i)$ which indicates that D matrix is not symmetric.

For each user $u_a \in U_{nCI}(c_i, c_j)$ we calculate the difference between the category interest values (i.e $nCI(u_a, c_i) - nCI(u_a, c_j) > 0$). The greater the difference, the more dissimilar those two categories are for the user u_a. At the end, a summarized difference between the two categories is calculated and eventually normalized by the number of users in $U_{nCI}(c_i, c_j)$. The proposed metric is defined by the following formula:

$$D(c_i, c_j) = \frac{\sum_{u_a \in U_{nCI}(c_i, c_j)} (nCI(u_a, c_i) - nCI(u_a, c_j))}{|U_{nCI}(c_i, c_j)|} \in [0, 1] \quad (5)$$

The main idea behind this group-based approach of calculating dissimilarity values is that we want to see users' opinion on one category (i.e. c_j) when they show a stronger interest on another category (i.e. c_i). If, on the whole, our users show an analogous interest on the category in question then the two categories are similar (i.e. low value in D matrix). On the other hand, if their opinion differs, we have just discovered a significant difference between the two categories. We describe the whole algorithmic procedure below (Algorithm 1) where all the values of D matrix are calculated.

Algorithm 1. Dissimilarity Values Estimation

Require: $U(c_l, c_m)$ where $c_l, c_m \in C$ and $c_l \neq c_m$
Ensure D matrix
  for all $c_i \in C$ **do**
    for all $c_j \in C$ **do**
      if $c_i \neq c_j$ **then**
        $summarizedCIdiff = 0$
        $userCounter = 0$
        for all $u_a \in U(c_i, c_j)$ **do**
          if $nCI(u_a, c_i) > nCI(u_a, c_j)$ **then**
            $summarizedCIdiff \mathrel{+}= (nCI(u_a, c_i) - nCI(u_a, c_j))$
            $userCounter{+}{+}$
          end if
        end for
        $D(c_i, c_j) = \left(\frac{summarizedCIdiff}{userCounter}\right)$
      end if
    end for
  end for

The computational complexity of the extraction of D matrix is affected by both the number of categories and the number of users included in the intersected groups for category pairs. In particular, we have:

$$O((L \times L - L) \times users_{avg} \times e)$$

where $L \times L - L$ are the number of pairs of categories and $users_{avg}$ is the average number of elements of the intersected groups of users belonged to those pairs (i.e. $|U(c_i, c_j)|$ where $i, j = 1, 2, \ldots, L$, $i \neq j$). The e cost represents the time for the calculation of the dissimilarity value of a pair. In terms of memory usage, we need space for $L \times L - L$ values. The algorithm's scalability is evaluated in terms of time performance in Section 7.5.

5. Dissimilarity recommender: training model & rating estimation process

When it comes to suggesting a new item to a user, our recommender discovers the relation of the item's categories with both the user's interest and all users' interest in them. To support this, the training model of our recommender follows a category-based approach represented by a set of vector spaces, one for each category. Our process of estimating ratings utilizes these spaces by searching for nearest rated items where their rating information is taken into account.

5.1. Category vector spaces

Inspired by Naive Bayes's strong independence assumptions among attributes, we create one space for each category instead of forming spaces for all the possible category combinations (i.e. power set of C). Each vector space contains information from training data characterized by users' preferences for the category in question (i.e. $P_{train}(c_l)$). Each object in this space depicts a preference (i.e. rated item) defined by two features: (a) the global feature extracted from values of D matrix and (b) the local feature extracted by using the normalized category interest values of the user who gave the rating. By using these two features together, we combine information regarding local and global degrees of interest for categories.

In this part, our main goal is to populate L vector spaces $S = \{S_{c_1}, S_{c_2}, \ldots, S_{c_L}\}$ in order to make use of them for rating estimations in a later stage. For each $p_i \in P_{train}$ we proceed to the following four tasks:

1. **Estimate the global feature**: This feature depicts the relation between a rated item's categories and the predominant category of the user who gave the rating. Its "global" meaning comes from the fact that we utilize the predominant category to find the opinion of users who have also shown greater interest in this category than the categories of the item in question. Such opinions are expressed by the values of the dissimilarity matrix. Therefore, we utilize these values to quantify the aforementioned global-scale relation. In particular, we summarize the dissimilarity values between item's categories and the predominant category and proceed their normalization:

$$global_i = \frac{\sum_{c_j \in item_b} D(pC(u_a), c_j)}{\sum_{c_l \in C} D(pC(u_a), c_l)} \in [0,1] \quad (6)$$

where $\exists p_i | p_i = (u_a, item_b, r_i)$.

2. **Estimate the local feature**: Here, we search for the relation between a rated item's categories and the user who gave the rating. By summing the user's normalized interest values in the categories of the item in question we can calculate an overall user's interest in those categories. Note that, we take into account only the categories that the user showed activity in the training data (i.e. $nCI(u_a, c_l) > 0$ where $c_l \in C$). The local feature value for p_i is defined as follows:

$$local_i = \frac{\sum_{j=1}^{m'} nCI(u_a, c_j)}{m' * nCI(u_a, pC(u_a))} \in [0,1] \quad (7)$$

where $\exists p_i | p_i = (u_a, item_b, r_i)$, $nCI(u_a, c_j) > 0$, $j = 1, 2, \ldots, m'$, $m' <= m$ and m is the number of $item_b$'s categories.

3. **Create Object**: We utilize the two aforementioned feature values to create a two-dimensional representation of a preference p_i. Apart from that, we also keep the rating included in p_i in order to be used in the rating estimation process. Therefore, an object in a vector space is defined as follows:

$$O_i = \{r_i, [global_i, local_i]\}$$

where $\exists p_i | p_i = (u_a, item_b, r_i)$.

4. **Employ Object Mapping**: Finally, we make use of the two-dimensional representation of the O_i object to map it into the vector spaces defined by the categories of the item. Note that, the rating r_i is not used as a feature of O_i in its mapping into the vector spaces. All in all, we have:

$$\forall c_j \in item_b \rightarrow S_{c_j} \uplus O_i$$

where $\exists p_i | p_i = (u_a, item_b, r_i)$.

Fig. 1 shows the process of both creating and mapping of O_i object derived from p_i preference. Its left part depicts the process of calculating the global and the local feature values through Eqs. (6) and (7) as long as the creation of O_i with the addition of the rating of p_i. Fig. 1 right part represents O_i's mapping into the vector spaces defined by the categories of p_i's item.

Intuitively, the two features act as two different sources of opinion when it comes to suggesting an item to a user. As described earlier, the local feature captures the personal opinion of the user on item's categories, while the global feature captures the opinion on these categories from all the users that show similar interests with the user in question. Consider the following three scenarios where there is a consensus in both types of opinion:

- **Weak Interest**: The categories of an item are not preferred by both the user and by those who have similar tastes with her. This item will be represented by low local and high global feature values in the vector spaces defined by its categories (see Fig. 2(a)). Note that, such cases indicate strong dissimilarity observed by both views.
- **Moderate Interest**: Both the user and the ones with similar tastes have shown moderate interest in an item's categories. This leads to moderate values in both features (see Fig. 2(b)).
- **Strong Interest**: The item's categories are preferred by both the user and the ones with similar tastes. Such opinions are characterized by high local and low global values (see Fig. 2(c)).

Although these two features may seem contradictory, they could be considered complementary and helpful in our case due to the fact that with their combination we try to discover the right balance between the global and the local view. The algorithmic version (Algorithm 2) of the whole procedure of forming the dissimilarity spaces is presented below.

Algorithm 2. Populate vector spaces

Require: D matrix
Ensure: $S = \{S_{c_1}, S_{c_2}, \ldots, S_{c_L}\}$
  for all $p_i = \{u_a, item_b, r_i\} \in P_{train}$ **do**
    $user_{nCI} = nCI(u_a, c_l)$, $\forall c_l \in C$
    $O_i = createObject(p_i, pC(u_a), D, user_{nCI})$
    for all $c_j \in item_b$ **do**
      $S_{c_j} = S_{c_j} \cup O_i$
    end for
  end for

Algorithm 3. Create Object

Require: $p_i = \{u_a, item_b, r_i\}$, D matrix, $pC(u_a)$ and $nCI(u_a, c_l)$ $\forall c_l \in C$
Ensure: O_i
  $totalGlobalSum = \sum_{c_l \in C} D(pC(u_a), c_j)$
  $counter = 0$
  $globalSum, localSum = 0$
  for all $c_j \in item_b$ **do**
    $globalSum$ += $D(pC(u_a), c_j)$
    $localSum$ += $nCI(u_a, c_j)$
    $counter$++
  end for
  $global_i = globalSum / totalGlobalSum$
  $local_i = localSum / counter * nCI(u_a, pC(u_a))$
  $O_i = \{r_i \in p_i, [global_i, local_i]\}$

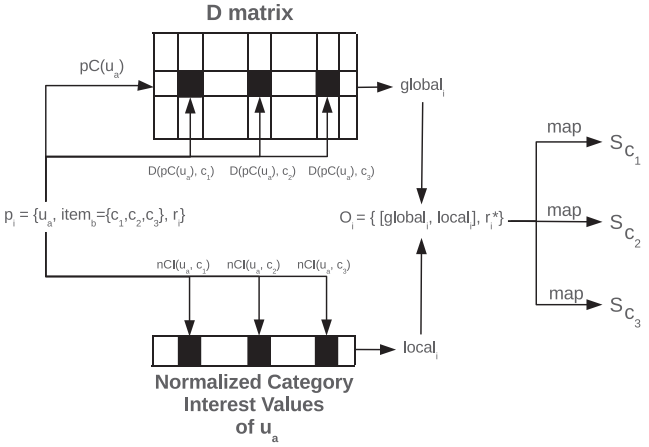
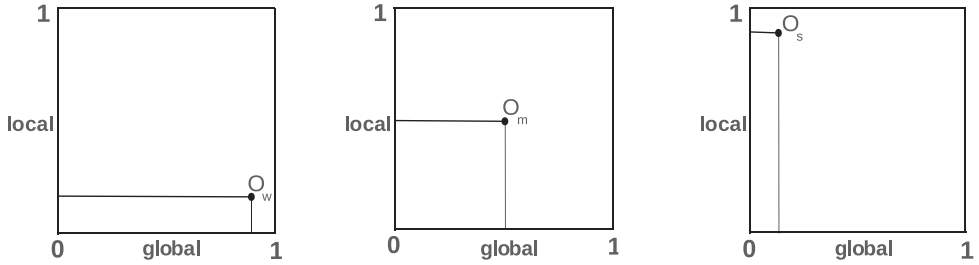


Fig. 1. Object creation and its mapping to vector spaces.



(a) Object with weak interest (b) Object with moderate interest (c) Object with strong interest

Fig. 2. Mapped objects with different types of interest in local and global views.

As we present in Section 5.2, the rating estimation process of our recommender involves searches for nearest objects in vector spaces in order to take into account their rating information towards providing estimations. We need an efficient method to get the nearest objects, especially when their number in these spaces increases. Instead of using the simple implementation of the *k*-nearest neighbor algorithm, we choose to create *kd*-trees structure[4] (Bentley, 1975). For each vector space we insert the two-dimensional representations of its objects into a *kd*-tree structure and we end up with an indexed space $indexedS_{c_l} = kdtree(S_{c_l})$ where $S_{c_l} \in S$.

In the process of creating those indexed spaces, the computational complexity is affected by the number of preferences in P_{train} as long as the number of categories. In particular,

$$O(|P_{train}| \times (e_1 + m_{avg}) + L \times e_2)$$

where e_1 cost is for calculating the global and local feature values for each $p_i \in P_{train}$ and m_{avg} denotes the mean number of categories of the items in training data. The $L \times e_2$ cost indicates the time for the creating a *kd*-tree structure for each category. For memory usage we have,

$$O(|P_{train}| \times m_{avg} \times e)$$

where e cost depicts the size of an object determined by the two feature values along with the rating of the preference from which the object came. The process of creating the indexed vector spaces is evaluated in Section 7.5 where we record changes in execution time when applying different sizes of training data.

5.2. Rating estimation process

We make use of the indexed category-based vector spaces to estimate ratings for items. For each $p_i \in P_{test}$ an object $O_i = \{r_i, [global_i, local_i]\}$ is created and mapped in those vector spaces defined by the categories of $item_b$ by using the two feature values. Each space gives one estimation which is the average of the actual ratings retrieved from the k nearest objects of O_i. The process of extracting the k nearest objects in a vector space involves the utilization of the *kd*-tree for this space created in the training phase (see Section 5.1).

After all the category-based rating estimations are made, we proceed to their combination into a single one by calculating their average value. This averaged value denotes the rating estimation of our recommender for the $item_b$—the more close to r_i, the more accurate the estimation is. At the end, we conclude with a set of estimations $E_{Dis} = \{est_1^{Dis}, est_2^{Dis}, \ldots, est_{|P_{test}|}^{Dis}\}$ where $est_i^{Dis} \in \Re$. Below, we describe the algorithm procedure (Algorithm 4):

Algorithm 4. Rating estimations based on dissimilarity recommender

Require: P_{test}, D matrix, k:#neighbors, $indexedS_{c_l}, \forall c_l \in C$
Ensure: E_{Dis}
  for all $p_i = \{u_a, item_b, r_i\} \in P_{test}$ **do**
    $user_{nCI} = nCI(u_a, c_l),\ \forall c_l \in C$
    $O_i = createObject(p_i, pC(u_a), D, user_{nCI})$
    $counter = 0$
    $sum = 0$
    for all $c_j \in item_b$ **do**
      sum += $nearestObjectsAvgRatingValue(indexedS_{c_j}, O_i, k)$
      $counter$++
    end for
    $est_i^{Dis} = sum/counter$
    $E_{Dis} \uplus est_i^{Dis}$
  end for

Regarding the rating estimation process, the computational complexity is mainly affected by the number of nearest objects. In particular, we have,

$$O(|P_{test}| \times (e_1 + m'_{avg} \times (NN(k) + e_2) + e_3))$$

where the e_1 cost denotes the time needed for the calculation of the two feature values for each $p_i \in P_{test}$ and m'_{avg} is the mean number of categories of items in test data. For each one of these categories, we need $NN(k)$ time to find the k nearest objects in its indexed vector space, while the cost e_2 denotes the time for the calculation of a category-based rating estimation. The cost e_3 is for the calculation of one item's overall rating estimation. For memory usage we have,

$$O(|P_{test}| \times m'_{avg} \times e)$$

where e cost denotes a category-based rating estimation value. The time performance of this process is affected by the number of nearest objects as long as different sizes of test data. Process's evaluation in terms of time is presented in Section 7.5.

6. The IANOS framework

Aiming to approach the recommendation problem from both similarity and dissimilarity perspective, we propose IANOS framework where two separate recommenders, a similarity-based recommender and our dissimilarity-based one, are applied. As Fig. 3 shows, IANOS starts with a data splitting process where data are separated into a training and a test part. Then, we have two components, the "Training Models Component" and the "Ratings Estimation Component".

[4] *Kd*-tree implementation for Java language, https://bitbucket.org/rednaxela/knn-benchmark/overview

6.1. Training models component

In this component, we make use of the training data to build the training models for the two recommenders. Therefore, we split the component into two segments, one for each recommender.

Regarding similarity-based recommenders, IANOS is not restricted to a specific type and it is flexible in utilizing existing ones as long as they provide rating estimations. We consider both the training model ("Build Model" in Fig. 3) and the process of rating estimations ("Similarity-based Recommender Estimator" in Fig. 3) of similarity-based techniques as black-box procedures. However, there might be a need to prepare the input data ("Data Modeling" in Fig. 3) in order to be used from the algorithm. For instance, in this work, we incorporated two recommenders, Naive Bayes and Slope-One (see Section 7.2). Those recommenders follow different data modeling processes which was the only part that we implement.

For the remainder of this work, we will refer to the training model of the similarity-based recommenders that we make use of as *SimModel*. Note that, the ratings in data used by those recommenders are affected by the user-based normalization factors ($d(u_a)$, see Section 3) just as in our recommender. This offers a common starting point for both recommenders with regard to input data.

As for our recommender, this component includes the data modeling process, the extraction of dissimilarity patterns and the formation of the indexed category-based vector spaces all of which were described in Sections 3, 4 and 5.1 respectively.

6.2. Ratings estimation component

IANOS uses the test data coming from the splitting process (i.e. P_{test}) and proceeds to two separate rating estimation processes. The first estimator comes from the similarity-based recommender that we trained earlier. Based on the *SimModel*, a rating for each item in test data is estimated and a set of estimations $E_{Sim} = \{est_1^{Sim}, est_2^{Sim}, \ldots, est_{|P_{test}|}^{Sim}\}$ is created where $est_i^{Sim} \in \Re$. The second estimator is from our recommender's training model the procedure of which was described in Section 5.2. From this estimator, we get another set of estimations $E_{Dis} = \{est_1^{Dis}, est_2^{Dis}, \ldots, est_{|P_{test}|}^{Dis}\}$.

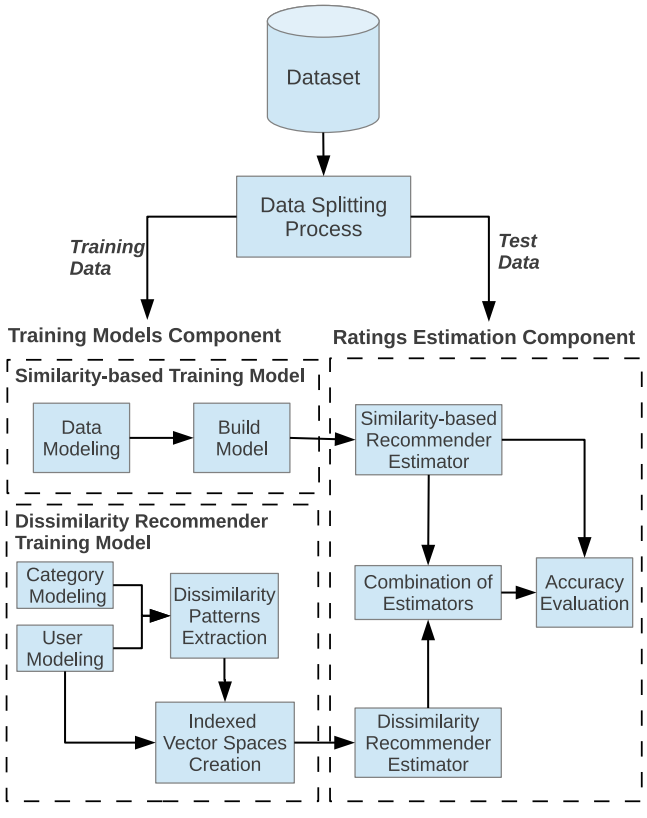


Fig. 3. Framework's architecture.

Having extracted the two lists of estimations (i.e. E_{Sim} and E_{Dis}), we form a third estimator by combining the estimations that correspond to the same items and produce one final estimation for each one of them. We make use of an f-measure equation due to the fact that it enables us to effortlessly put more emphasis on either the one or the other estimation through a $\beta_2 > 0$ parameter (as β_2 increases, estimations from our recommender are emphasized).

$$est_i^{Dis\&Sim} = \frac{(1+\beta_2^2) * est_i^{Dis} * est_i^{Sim}}{\beta_2^2 * est_i^{Dis} + est_i^{Sim}} \tag{8}$$

where $i = 1, 2, \ldots, |P_{test}|$. Therefore, we end up with a set of the combined rating estimations $E_{Dis\&Sim} = \{est_1^{Dis\&Sim}, est_2^{Dis\&Sim}, \ldots, est_{|P_{test}|}^{Dis\&Sim}\}$. To evaluate the performance of IANOS we compare the estimations contained in E_{Sim} and $E_{Dis\&Sim}$ in terms of accuracy through the use of RMSE metric (Gunawardana and Meek, 2009). Thus, we have two RMSE calculations, one that measures the error between real values and estimations coming from the similarity-based recommender and one with the estimations of our recommender combined with the similarity-based ones. More specifically, we have:

$$RMSE_{Sim} = \sqrt{\frac{\sum_{i=1}^{|P_{test}|}(est_i^{Sim} - r_i)}{|P_{test}|}} \tag{9}$$

$$RMSE_{Dis\&Sim} = \sqrt{\frac{\sum_{i=1}^{|P_{test}|}(est_i^{Dis\&Sim} - r_i)}{|P_{test}|}} \tag{10}$$

where r_i is the normalized rating that belongs to the item for which the two predictions were made. By comparing the two aforementioned RMSEs, we can conclude whether the combined estimations are more accurate than the estimations extracted from the similarity-based recommender (i.e. $RMSE_{Dis\&Sim} < RMSE_{Sim}$).

7. Experimentation

We carried out experiments on real-world datasets to ascertain the usability of the proposed dissimilarity recommender when it is combined with similarity-based ones. For the experiments, we utilized three different 64-bit Linux-based work stations on which we installed Hadoop 1.1.0 version. The master node, which was also a slave, has *8* cores with 12 GB physical memory. The second slave has *4* cores with 4 GB memory while the last one has the same amount of memory but *2* CPUs. Here, we present the datasets and the similarity-based recommenders we made use of. Moreover, we describe our experimentation steps and comment upon the results with regard to accuracy and time performance.

7.1. Datasets description

To test IANOS, we made use of two datasets, Yahoo! Movies and Movielens, which contain user ratings on movies accompanied by genres as their attributes. Although the nature of data in both datasets is the same, they greatly differ in term of size (i.e. Yahoo! Movies number of ratings are only the 2% of Movielens ratings). By applying our method to both a small and a big dataset, we can evaluate the scalability of our method as long as monitor how the learning curve of our recommender's training model is affected when we apply different sizes of training data. Table 3 contains

Please cite this article in press as: Zigkolis, C., et al. Integrating similarity and dissimilarity notions in recommenders. *Expert Systems with Applications* (2013), http://dx.doi.org/10.1016/j.eswa.2013.03.018

the description for the two datasets which, for the rest of the paper, will be referred as D_1 and D_2 respectively.

7.2. Similarity-driven recommenders

For the purpose of evaluating our recommender's usability, two very commonly used recommenders were selected to be combined, Naive Bayes (Duda and Hart, 1973) and Slope-One (Lemire and Maclachlan, 2007). They represent the two basic categories of recommenders, the content-based and the collaborative ones, and they were chosen because they incorporate the concept of similarity towards providing rating estimations. In the case of Naive Bayes, probabilistic models are built for each user based on her previous ratings that provide high scores for items that have *similar* attributes to the items favoured by each user. In the case of Slope-One, higher scores are given to items that were rated similarly high by the user in question and the other users. In other words, Slope-One searches for *similar* preferences between users to provide estimations for unrated items. Each algorithm provides rating estimations separately which are then combined with the ones coming from our recommender. Accuracy-based comparisons between the stand-alone and the combined rating estimations will indicate the usefulness of such combinations as well as how well our recommender co-operates with two of the most popular recommenders.

7.3. Evaluation metrics

IANOS performance is measured by two distinct metrics, RMSE improvement and execution time. We first calculate the RMSE produced by the stand-alone execution of either Naive Bayes or Slope-One. Then, by calculating the RMSE from the combined estimator, we end up with a positive or negative percentage improvement which is defined as follows:

$$improvement_{\%} = 100 - \frac{100 * RMSE_{Dis\&Sim}}{RMSE_{Sim}} \quad (11)$$

The greater the improvement in Eq. (11), the more accurate the combinations are as opposed to the similarity-based recommender.

Time of execution, on the other hand, shows the amount of time required for the system to produce the required recommendations. This metric will be used to present the additional computational complexity introduced by our recommender as well as show the impact the hadoop framework and its distributed nature have on execution time.

7.4. Accuracy-based experimental steps

IANOS framework has three parameters that need to be adjusted: (i) *the popularity/likeness weight* β_1 in Definition 3.1, (ii) *the k nearest objects* in Section 5.2 and (iii) *the intra-recommender weight* β_2 in Eq. (8). Our accuracy-based experiments are divided into three experimental steps where each one of them applies different values for one of the aforementioned parameters, and the value with the best accuracy results is used in the subsequent steps. Although this is a "depth-first"-liked search that leads to local optima, an exhaustive search is found to be unnecessary for the scope of this work. As said earlier, we do not intend to find the global optima, but to test the practicality of IANOS combinations. Moreover, an exhaustive search is computational prohibitive. Prior to the three steps, one more will be incorporated showing the results of IANOS using predefined parameters that will be used as a baseline for comparison. Table 4 contains the four experimental steps along with the parameters and their values they make use of.

We make use of predefined values for the three parameters (see Table 4). Regarding the first parameter β_1 we choose different values in order to favour either likeness or popularity towards extracting the degrees of users' interest in categories. Different values in this parameter will result in changes in both the dissimilarity values and the objects mapped in our training model's vector spaces. As for the k nearest objects, we go from small to large values in order to find an upper bound. Decline in accuracy results when using values above that bound may indicate that when estimating a rating (see Section 5.2) information from "noisy" objects is taken into account. In term of the last parameter β_2, we apply different values to put more emphasis on the estimations of either the similarity-based recommender or the proposed one. Changes here affect only the final combined estimations. All in all, changes in all parameters will affect the accuracy rates.

7.4.1. Accuracy results

As noted earlier, the first experimental step's accuracy results act as a baseline for comparison and every subsequent step monitors changes in accuracy when applying different values in the three parameters of our framework. Regarding the splitting of data, we traverse from small to large-scale size of training data by creating the following five splits for both datasets: 20–80%, 40–60%, 60–40%, 80–20%. We also create a 5–95% split in order to test our method under conditions that usually appear in new-user problem scenarios. The accuracy results for all the five experimental steps are presented below:

- **Control Set of Parameters:** In both datasets, our recommender's combination with Naive Bayes provides improved results of up to 18.09% and 18.74% in the D_1 and D_2 datasets respectively. However, Slope-Ones combination does not show, in most of the splits, corresponding improvement as in both datasets most results are negative with D_1 results being worse than D_2. The only positive improvements are found in D_2 in splits 20–80% and 40–60%. Fig. 4a shows the results for D_1 dataset while Fig. 4b illustrates the ones for the D_2 dataset for the five splits.
- **Popularity versus Likeness:** By Fig. 5a we can see that results show little change for Naive Bayes where improvements were provided regardless of the weight used. Nonetheless, we get the best results with $\beta_1 = 2$ indicating the greater importance

Table 3
Datasets description.

Dataset	$\|U\|$	$\|I\|$	$\|P\|$	$\|C\|$
Yahoo! Movies (D_1)	7637	9215	196,025	16
MovieLens (D_2)	69,878	10,675	10M	19

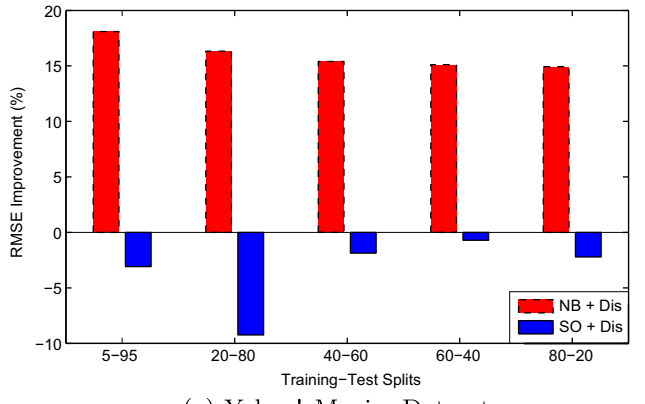
Table 4
IANOS parameters and their values in the experimental steps (NB: Naive Bayes, SO: Slope-One and ALL: all values tested).

Parameter	Values
β_1: weight on popularity/likeness	0.25, 0.5, 1, 2, 4
k: number of nearest objects	5, 10, 15, 20, 25, 30, 35, 40
β_2: weight on rating estimations	0.25, 0.5, 1, 2, 4
Experimental step	Parameters chosen values
Control set of parameters	NB: $\beta_1 = 1$, $k = 10$ and $\beta_2 = 1$ SO: $\beta_1 = 1$, $k = 10$ and $\beta_2 = 1$
Popularity VS Likeness	NB: $\beta_1 = ALL$, $k = 10$ and $\beta_2 = 1$ SO: $\beta_1 = ALL$, $k = 10$ and $\beta_2 = 1$
Different neighborhoods	NB: $\beta_1 = 2$, $k = ALL$ and $\beta_2 = 1$ SO: $\beta_1 = 4$, $k = ALL$ and $\beta_2 = 1$
Weight on two predictors	NB: $\beta_1 = 2$, $K = 40(D_1$ & $D_2)$ and $\beta_2 = ALL$ SO: $\beta_1 = 4$, $K = 40(D_1$ &$D_2)$ and $\beta_2 = ALL$

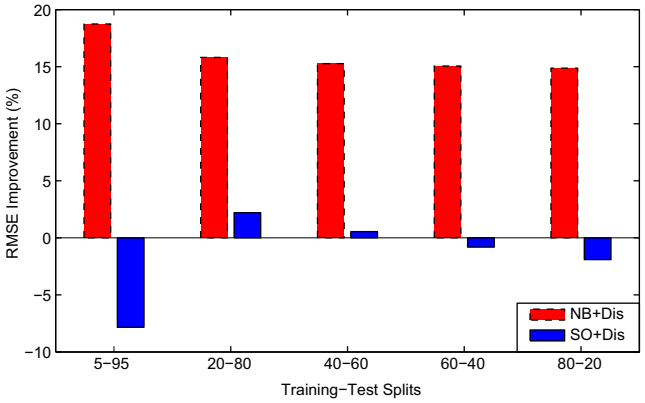
Please cite this article in press as: Zigkolis, C., et al. Integrating similarity and dissimilarity notions in recommenders. *Expert Systems with Applications* (2013), http://dx.doi.org/10.1016/j.eswa.2013.03.018

of popularity against the likeness without ignoring the contribution of the latter. In particular, for D_1 we have up to 18.22% improvement while for D_2 the accuracy was improved up to 19.28%. In Slope-Ones case, as Fig. 5b shows, putting more emphasis on likeness produces most of the time negative results as opposed to the improved results in both datasets when we greatly favour popularity ($\beta_1 = 4$). In particular, we get up to 8.74% improvement in D_1 and 6.56% in D_2. However, as the training set increases in size, this improvement lowers and gets negative in D_2, but remains slightly above zero in D_1. The improvement on Slope-One's estimations when popularity is greatly emphasized can be attributed to the algorithms own rationale. Slope-One takes into consideration how much an item was liked by users who have common ratings to the user in question. However, it does not take into account how many times the item was rated, meaning an item with a relatively high average rating that was rated by many users may be shunned in favour of an item that was rated only once with the maximum rating. By giving greater weight to popularity, this inherent weakness of the standard Slope-One is compensated. In case of Naive Bayes, the small changes in improvements when applying the different values of β_1 indicates that whether we favour popularity or likeness seems to be insignificant for our recommender when it is combined with Naive Bayes.

- **Different neighborhoods:** Results in both datasets do not present any significant changes for either Naive Bayes or Slope-One when we use different number for nearest objects. In Naive Bayes case, we achieved improvements up to 18.49% and 19.49% while in Slope-One we got 8.63% and 7.11% respectively for the two datasets. In both algorithms, best results came from $k = 40$ for both D_1 and D_2. However, results coming from all different numbers of k are so close with each other as to show us that there is no clear evidence that even larger values of k will produce better results. Fig. 6a and Fig. 6b present the results for this experimental step.
- **Weight on two predictors:** The results of the final set show increased performance in both datasets for Naive Bayes that reach 37.18% in D_1 and 38.8% in D_2 respectively. The results seem to remain stable as the training set increases hinting that the learning curve of the Dissimilarity-Naive Bayes combination is analogous with the one of Naive Bayes when we run it as a stand-alone recommender.
  In the case of Slope-One, a single best value for β_2 was not found. However, we observed that in D_1, the best results were taken when there was a balance between Slope-One's estimations and ours ($\beta_2 = 1$) especially in the medium and large training sets. The same phenomenon was observed in the small training sets of D_2, but when the training data increase more emphasis on Slope-One's estimations produces better accuracy results ($\beta_2 = 2$). This clearly indicates Slope-One has a better learning curve than our recommender in the presence of large sizes of training data. In particular, in D_1 we got, most of the time, positive results of up to 9,9%. In D_2 improved results reached up to 7%, an improvement that was declined as we passed to larger training sets. Fig. 7a and b present the results for this final step.



(a) Yahoo! Movies Dataset



(b) MovieLens Dataset

Fig. 4. Accuracy for control set of parameters.

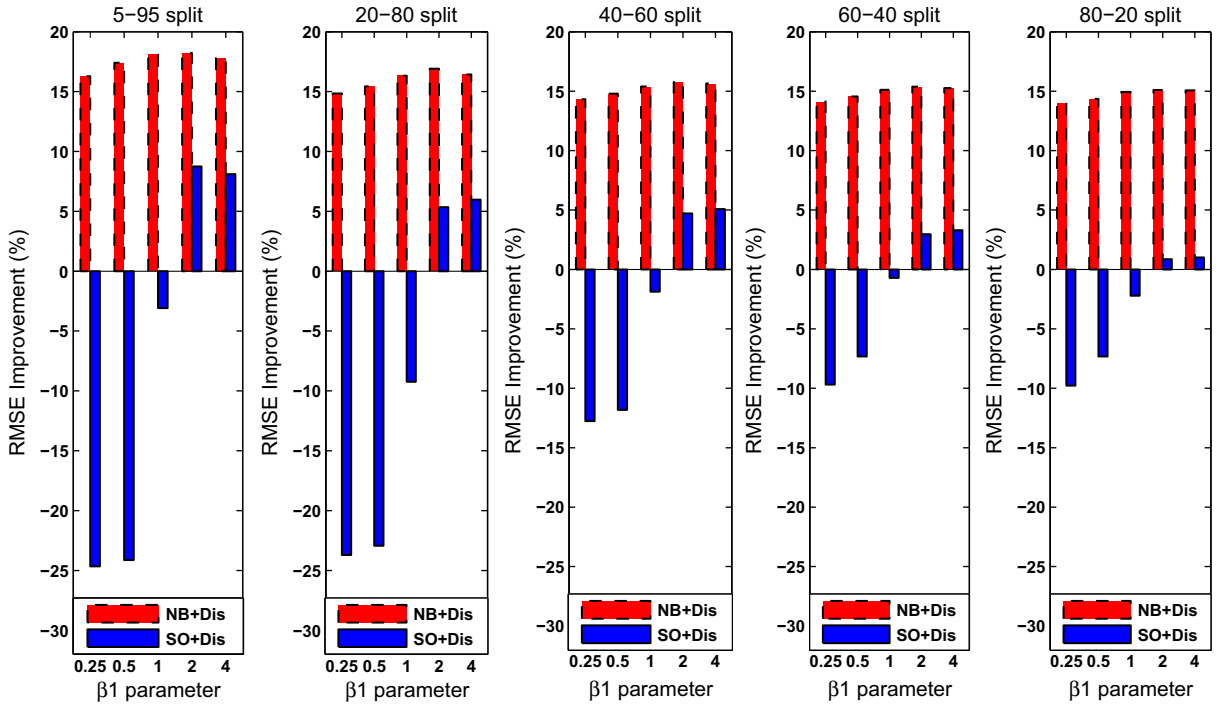
7.5. Time performance

Apart from the four steps discussed above, two more parameters will be introduced presenting time performances. We add and tune two parameters related to the hadoop framework, *the node number* and *the replication factor*. The former denotes the number of work stations we make use of in an experiment while the latter indicates the number of replications for hadoop data blocks in the different nodes. The minimum replication factor is *1* and the maximum is the number of all the available nodes. By giving the minimum value, we reduce the need for disk space (each data block in one node at the time), but we increase the likelihood for a node to request a block for the one that has it. This will lead to additional data block exchanges through the network. On the other hand, by giving the maximum value, we surely increase the need for disk space (i.e. all blocks in all nodes), but we ensure that when a node needs a block file, it will find it immediately by accessing its part in the hadoop distributed file system.

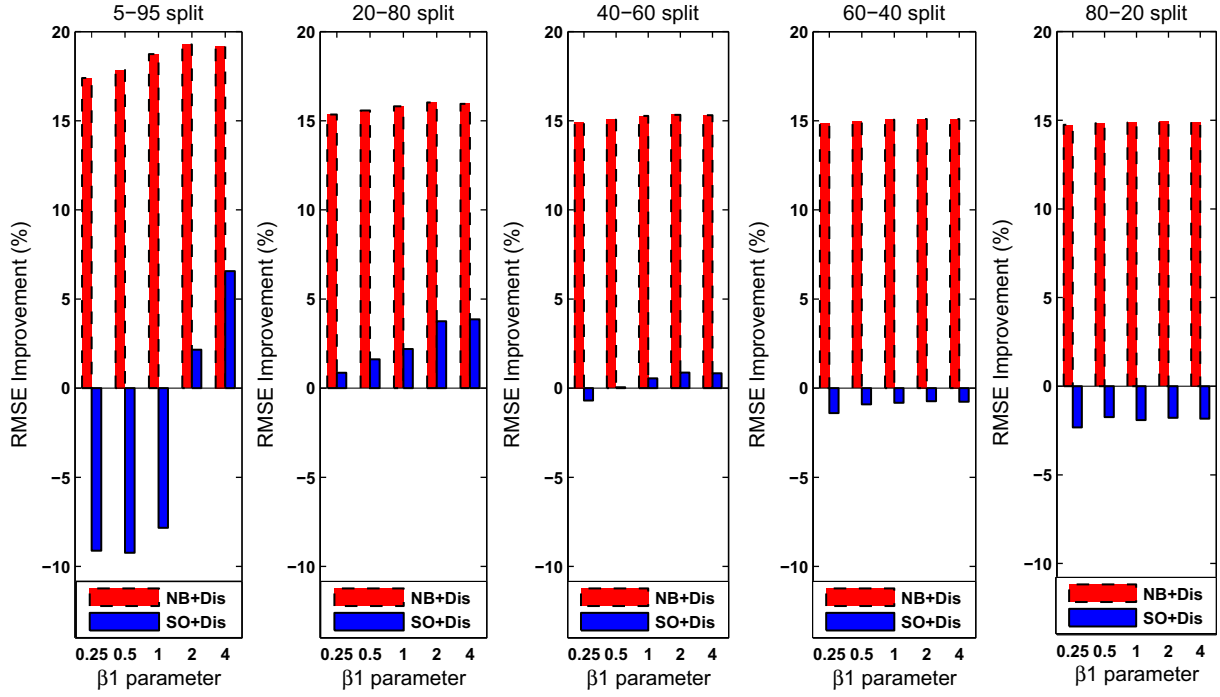
For the purpose of this section we evaluate in terms of time performance the following five processes from IANOS: (a) the data modeling process of our recommender, (b) the process of calculating the dissimilarity matrix, (c) the process of creating the indexed vector spaces, (d) the rating estimations process and (e) the combination of estimations process. These processes represent the most resource-demanding functions of the application and the purpose in this section is to show how time performance can be improved using a distributed implementation. The time of those five functions/elements has been measured from the beginning of the mapping procedure to the end of the reduction procedure as noted in the hadoop framework. Therefore, apart from the actual time complexity of the algorithms upon which those functions were created, overhead will also be included. Each of the aforementioned processes have been run in six different setups included in Table 5. These setups are all possible combinations for the two parameters when three work stations are available just as in our case.

Note that, only the framework's time performance of the Movielens dataset is presented here. The fact that both datasets have to

Please cite this article in press as: Zigkolis, C., et al. Integrating similarity and dissimilarity notions in recommenders. *Expert Systems with Applications* (2013), http://dx.doi.org/10.1016/j.eswa.2013.03.018



(a) Yahoo! Movies Dataset



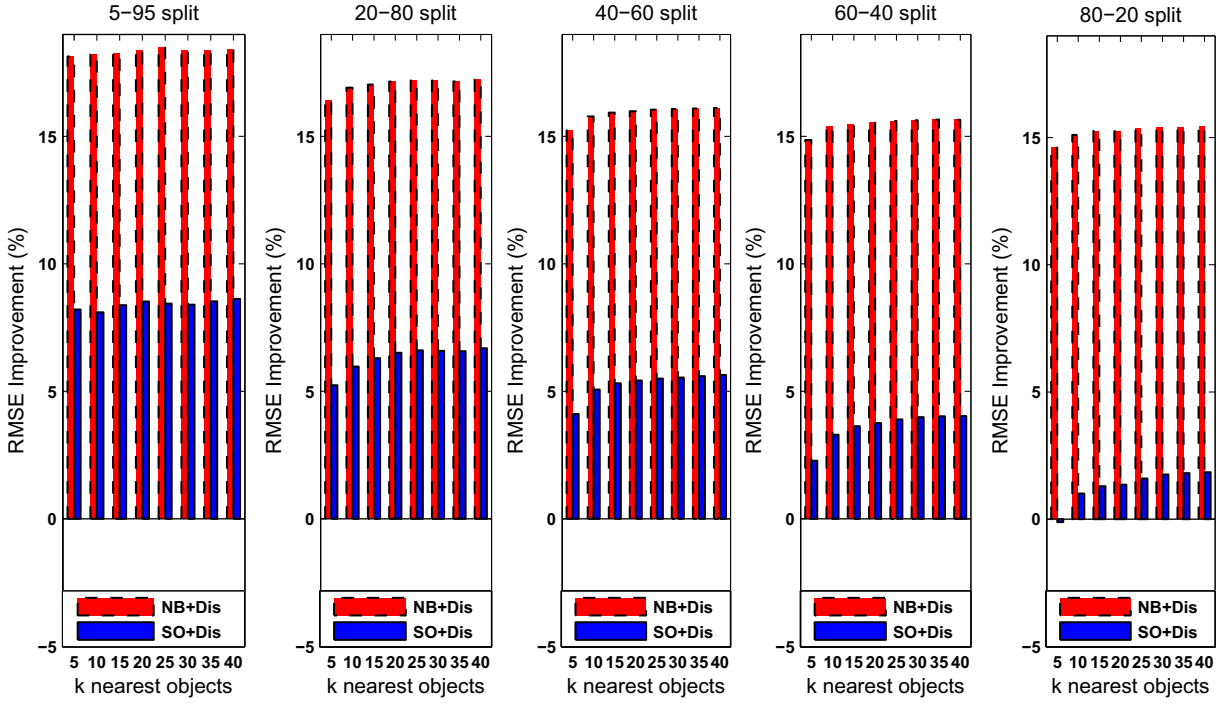
(b) MovieLens Dataset

Fig. 5. Accuracy for different values of β_1.

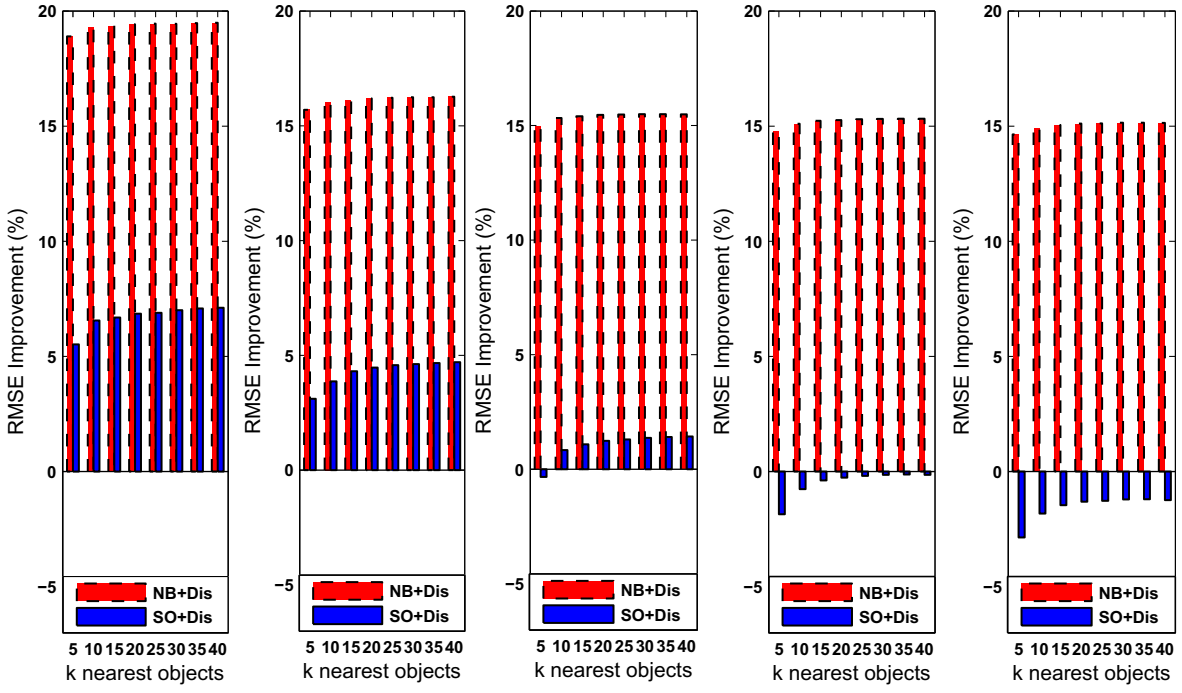
do with movies and that Movielens is larger than Yahoo! Movies make the presentation of time performance for Yahoo! Movies dataset redundant. With the data volume of Movielens dataset we can present IANOS behaviour when data scales. Fig. 8 accumulates the results from all processes comments upon which are presented below.

- **Data Modeling**: As Fig. 8a shows, time performance is declined as our training data increase. As we increase both the number of nodes and the replication factor, time decreases since there are more nodes processing the input data and not much intercommunication is required between the nodes during the process as data is available to all nodes locally. However, results are not

Please cite this article in press as: Zigkolis, C., et al. Integrating similarity and dissimilarity notions in recommenders. *Expert Systems with Applications* (2013), http://dx.doi.org/10.1016/j.eswa.2013.03.018



(a) Yahoo! Movies Dataset



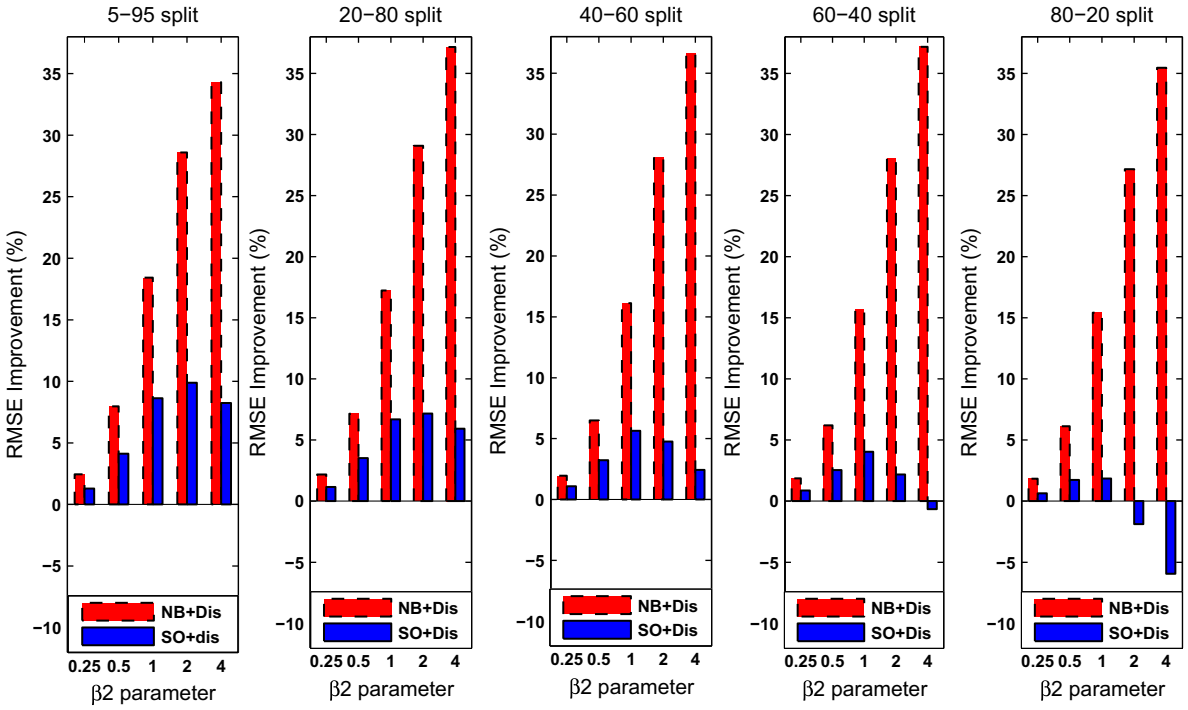
(b) MovieLens Dataset

Fig. 6. Accuracy for different number of neighbors.

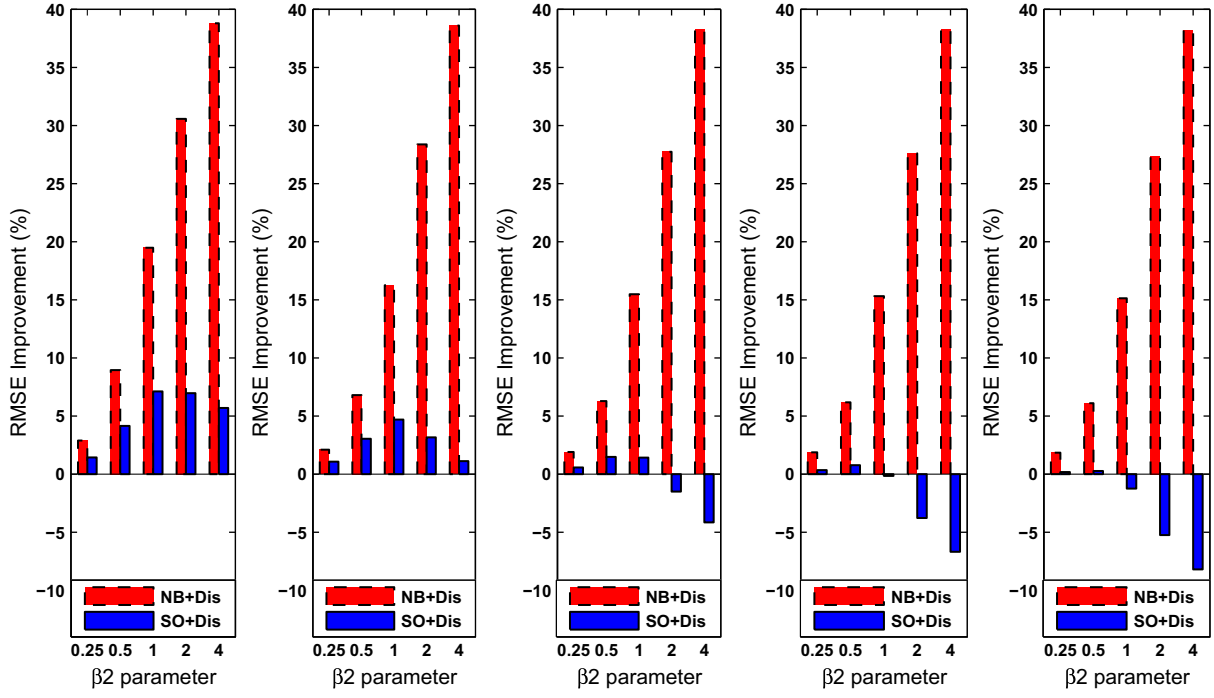
linearly improved. That can be attributed to the overhead of transferring the data blocks to the nodes, a phenomenon that is enhanced by the replication factor, and the additional complexity of merging the results as nodes increase. All in all, we improve time up to 54% between the 1st and the 6th setup which greatly improves the time required for the data modeling.

- **Extraction of dissimilarity matrix**: As before, time performance is declined as we move to a greater training-test split. In this process, increasing node number and replication factor also has a positive effect in time requirements reaching an improvement of up to 67% between the 1st and 6th setup (see Fig. 8b). We can also notice that by increasing the nodes and the replication factor the results are not uniformly improved.

Please cite this article in press as: Zigkolis, C., et al. Integrating similarity and dissimilarity notions in recommenders. *Expert Systems with Applications* (2013), http://dx.doi.org/10.1016/j.eswa.2013.03.018



(a) Yahoo! Movies Dataset



(b) MovieLens Dataset

Fig. 7. Accuracy for different values of β_2.

In some cases, using smaller replication factors seem to provide better results as noted in the 60–40% split where 4th setup yields better results compared to all the others. The reasons behind this, aside from overhead, is random noise in the medium, collisions and so forth which can affect results of relatively small size. However, increasing the nodes and the replication factor seems to on average improve the performance.

- **Creation of Vector Spaces**: This particular process is a relatively light procedure. While the plot shows that there is an improvement using multiple nodes, results are not very impressive with a 27% improvement at maximum (see Fig. 8c). Furthermore, performance is very similar between the various multinode modes. This can be attributed to the aforementioned reasons of collisions and overhead that in such low time com-

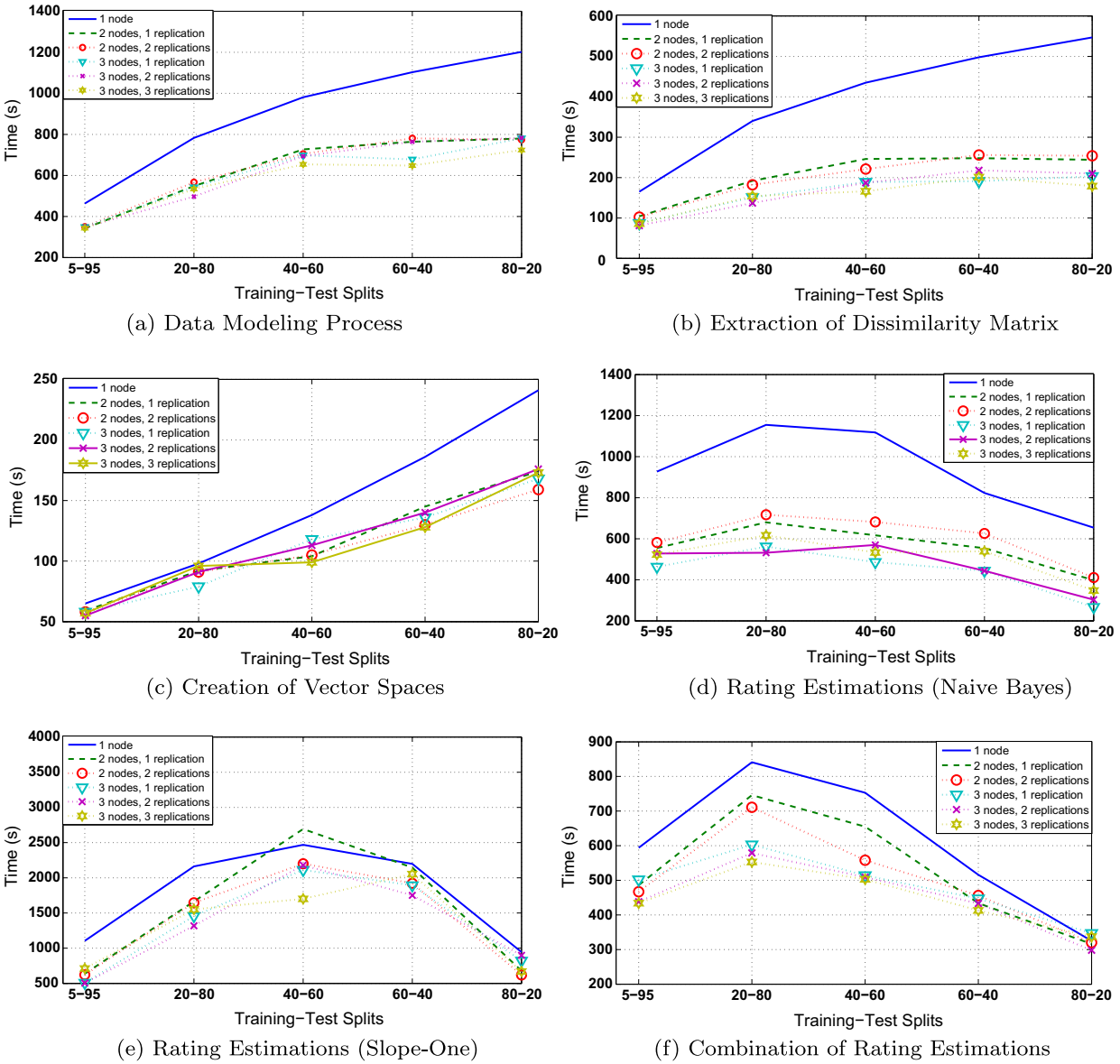
Please cite this article in press as: Zigkolis, C., et al. Integrating similarity and dissimilarity notions in recommenders. *Expert Systems with Applications* (2013), http://dx.doi.org/10.1016/j.eswa.2013.03.018

Table 5
Experimental setups for measuring time performance.

No.	Setup description
1	The master node in parallel mode (more than one threads)
2	Two nodes with replication factor 1
3	Two nodes with replication factor 2
4	Three nodes with replication factor 1
5	Three nodes with replication factor 2
6	Three nodes with replication factor 3

plexities they greatly limit the performance improvement. Nonetheless, while improvement is low for small sizes of training data, the difference becomes greater for large ones.

- **Rating Estimations**: In this phase, time performance did not only depend on the volume of our training data but also on that of the testing data. The greater the training data were, the bigger our category-based spaces would become and, thus, the k nearest objects searches would be more expensive in terms of time. Nonetheless, when the testing data increase there are more k nearest objects searches being executed. Since these two are inversely proportional, we observe a bell curve in the results in the plots. In addition to the estimations from our recommender, estimations are calculated by the two similarity-based recommender, Naive Bayes and Slope-One. As a result, two figures have been included, one when Naive Bayes is executed (Fig. 8d) and one when Slope-One is executed (Fig. 8e) with the latter showing clearly greater execution times than the former due to the more time-complex nature of Slope-One. In Fig. 8d, we also observe that the 4th setup performs slightly better than all the other modes. In addition, the 2nd setup is also faster than the 3rd. Similarly, in Fig. 8e, the 4th and 5th setup perform better most of the time than the 6th. The above can be attributed to the fact that while time complexity is high, since the models of the recommenders are passed to the nodes, there is a large transfer of data that takes



(a) Data Modeling Process

(b) Extraction of Dissimilarity Matrix

(c) Creation of Vector Spaces

(d) Rating Estimations (Naive Bayes)

(e) Rating Estimations (Slope-One)

(f) Combination of Rating Estimations

Fig. 8. Time performance.

Please cite this article in press as: Zigkolis, C., et al. Integrating similarity and dissimilarity notions in recommenders. *Expert Systems with Applications* (2013), http://dx.doi.org/10.1016/j.eswa.2013.03.018

place which doubles and triples for 5th and 6th setup respectively. This additional overhead outweighs the advantages of higher replication factors.

- **Combination of Estimations**: In the last phase, time complexity also depends on both the size of our training and test data. Nonetheless, it is apparent that the size of our test data affects time complexity on a much greater level which is evident by the fact that for splits equal to or greater than 20–80% the time required for the operation to end falls steadily. With the 6th setup we achieve the best results being faster by up to 35%. However, the improvement diminishes with our testing data and in the case of the 80–20% split the mode becomes slower than the one node due to overhead (see Fig. 8f).

7.6. Results discussion

Experiments carried out in the two datasets have shown that the proposed recommender can improve the accuracy when combined with the two popular Slope-One and Naive Bayes algorithms. In the case of Naive Bayes, experiments have shown that the combination consistently outperforms the stand-alone content-based filter by up to 37.18% in Yahoo! Movies dataset and up to 38.8% in Movielens. As the training set increases in size, the combination keeps its accuracy at high levels which shows that our recommender's learning curve is as good as Naive Bayes's. The stability in these results may be contributed to the fact that our recommender follows a content-based logic as it is attribute-oriented. Therefore, it seems to co-operate better when combined with recommender of this type. Experimentation with other content-based recommenders will enhance this conclusion.

In terms of Slope-One, results have varied based on the training set size. In most of the splits, improved results were observed, up to 9.9% in Yahoo! Movies and 7% in Movielens. However, this improvement was declined as we utilized larger training sets. In fact, in large splits, best results came when we put more emphasis on Slope-One's estimations. This indicates that Slope-One learns faster than our recommender and eventually provides more accurate recommendations as seen in the Movielens dataset. Therefore, it can be concluded that our recommender, when it is combined with collaborative recommenders, works better in small training sets. In cases with large training data, we can rely on estimations from the similarity-based algorithm without ignoring the contribution of our recommender.

As for the new-user problem, in both datasets, the combination of the our recommender with both the two similarity-based algorithms has provided improved results for a small number of training data as shown by Fig. 7. Therefore, our work can be used to alleviate the new-user problem that affects many contemporary recommenders.

Regarding time performance through the use of distributed processes, experiments have shown that by using Hadoop we can greatly reduce the additional costs of the proposed algorithm by up to 67% in the more time consuming processes of the algorithm. However, it was also shown that performance improvement varies based on the nature of the functions and training and test sets' sizes. As can be seen by the time performance results, distributability greatly decreases the time requirements of many of the procedures described in this paper, thus it is considered significant in this work.

All in all, IANOS, and subsequently the proposed recommender, seem to be practical in scenarios where we have small sizes of training data. In addition, IANOS can be seen as a benchmark tool because it enables us to incorporate several similarity-based recommender in order to test their sole performance in various datasets as long as to evaluate their combinations with our recommender and check to see how our recommender co-operates with them.

8. Conclusions & future work

In this work, a novel recommender has been presented based on underlying dissimilarity values that exist between attributes on a set of items. We combined this algorithm with two well-known recommenders, Naive Bayes and Slope-One, in order to evaluate our intuition that approaching the problem of recommendations from two different perspectives—similarity and dissimilarity—may lead to more accurate predictions. Experimentation that has been carried out in two separate datasets showed, most of the time, that similarity techniques provided less accurate results when they were used as stand-alone procedures rather than when they were combined with our dissimilarity technique. However, such combinations come with an additional cost in terms of computational complexity. In order to address this issue, we implemented IANOS distributed framework with Hadoop technology in order to allocate the work load to many work stations. Results, regarding time performance of IANOS, indicate the usefulness of its distributed implementation. With the proposed framework we can run different recommenders which gives us the opportunity both to evaluate their general performance and check to see how our recommender co-operates with them.

Future work will focus on exploiting our dissimilarity logic in different ways. For instance, it would be interesting to properly represent our attribute-based dissimilarity values through a distinct characteristic—a new feature—in content-based techniques. This additional feature would lead a content-based recommender to improved accuracy results. Moreover, considering the recent research interest in diversity issues (Zhang and Hurley, 2008) (i.e. increasing the diversity among suggestions), we plan to study possible applications of our dissimilarity logic to confront such issues. Dissimilarity and diversity can be seen close as concepts, so it might be worthwhile to expand our research to this direction.

Improvements in the implementation of IANOS in terms of time and memory usage are also planned for future work. In terms of time complexity, we could identify possible bottlenecks in the procedures and improve the execution time through the use of different modeling of data. For instance, the profiling of categories in our framework may demand great amount of memory when a node handles a category with a great number of ratings. Therefore, a different modeling policy could be used to alleviate this problem. On the other hand, the memory usage complexity could be confronted with the use of a file compression policy on the files saved on hadoop distributed file system. Last but not least, we plan to test other indexing techniques such as QuadTrees (Finkel et al., 1974) or use of a distributed (Trad et al., 2012) or parallel (Sismanis et al., 2012) logic in order to make the nearest objects search operations more efficient.

Acknowledgements

Christos Zigkolis's work has been co-financed by the European Union (European Social Fund ESF) and Greek national funds through the Operational Program "Education and Lifelong Learning" of the National Strategic Reference Framework (NSRF)–Research Funding Program: Heracleitus II. Investing in knowledge society through the European Social Fund. Furthermore, we would like to acknowledge Yahoo! Labs for providing this research endeavour with the dataset of Yahoo! Movies R4 via Webscope.

Please cite this article in press as: Zigkolis, C., et al. Integrating similarity and dissimilarity notions in recommenders. *Expert Systems with Applications* (2013), http://dx.doi.org/10.1016/j.eswa.2013.03.018

References

Adomavicius, G., & Tuzhilin, A. (2005). Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE Transactions on Knowledge and Data Engineering, 17*, 734–749. http://dx.doi.org/10.1109/TKDE.2005.99.

Amer-Yahia, S., Roy, S. B., Chawlat, A., Das, G., & Yu, C. (2009). Group recommendation: Semantics and efficiency. *Proceedings of the VLDB Endowment, 2*, 754–765. http://dl.acm.org/citation.cfm?id=1687627.1687713.

Bellogin, A., Castells, P., & Cantador, I. (2011). Self-adjusting hybrid recommenders based on social network analysis. In *Proceedings of the 34th international ACM SIGIR conference on research and development in information retrieval* (pp. 1147–1148). New York, NY, USA: ACM. http://dx.doi.org/10.1145/2009916.2010092. http://doi.acm.org/10.1145/2009916.2010092.

Bentley, J. L. (1975). Multidimensional binary search trees used for associative searching. *Communications of the ACM, 18*, 509–517. http://dx.doi.org/10.1145/361002.361007. http://doi.acm.org/10.1145/361002.361007.

Bezerra, B. L. D., & de Carvalho, F. d. A. T. (2004). A symbolic approach for content-based information filtering. *Information Processing Letters, 92*, 45–52. http://dx.doi.org/10.1016/j.ipl.2004.06.003. http://dx.doi.org/10.1016/j.ipl.2004.06.003.

Bobadilla, J., Hernando, A., Ortega, F., & Bernal, J. (2011). A framework for collaborative filtering recommender systems. *Expert Systems with Applications, 38*, 14609–14623. http://dx.doi.org/10.1016/j.eswa.2011.05.021. http://www.sciencedirect.com/science/article/pii/S0957417411008049.

Braak, P. t., Abdullah, N., & Xu, Y. (2009). Improving the performance of collaborative filtering recommender systems through user profile clustering. *Proceedings of the 2009 IEEE/WIC/ACM international joint conference on web intelligence and intelligent agent technology* (vol. 03, pp. 147–150). Washington, DC, USA: IEEE Computer Society. http://dx.doi.org/10.1109/WI-IAT.2009.422. http://dx.doi.org/10.1109/WI-IAT.2009.422.

Breese, J. S., Heckerman, D., & Kadie, C. (1998). Empirical analysis of predictive algorithms for collaborative filtering. In *Proceedings of the fourteenth conference on uncertainty in artificial intelligence* (pp. 43–52). San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.. http://dl.acm.org/citation.cfm?id=2074094.2074100.

Burke, R. (2002). Hybrid recommender systems: Survey and experiments. *User Modeling and User-Adapted Interaction, 12*, 331–370. http://dx.doi.org/10.1023/A:1021240730564. http://dl.acm.org/citation.cfm?id=586321.586352.

Claypool, M., Gokhale, A., Miranda, T., Murnikov, P., Netes, D., & Sartin, M. (1999). Combining content-based and collaborative filters in an online newspaper.

De, A., Desarkar, M. S., Ganguly, N., & Mitra, P. (2012). Local learning of item dissimilarity using content and link structure. In *Proceedings of the sixth ACM conference on recommender systems* (pp. 221–224). New York, NY, USA: ACM. http://dx.doi.org/10.1145/2365952.2365999. http://doi.acm.org/10.1145/2365952.2365999.

Duda, R. O., & Hart, P. E. (1973). *Pattern classification and scene analysis*. New Yotk: John Willey & Sons.

Finkel, R. A., Bentley, J. L., Finkel, R. A., & Bentley, J. L. (1974). Quad trees a data structure for retrieval on composite keys. *Acta Informatica, 4*, 1–9. http://dx.doi.org/10.1007/BF00288933. http://dx.doi.org/10.1007/BF00288933.

Gartrell, M., Xing, X., Lv, Q., Beach, A., Han, R., Mishra, S., et al. (2010). Enhancing group recommendation by incorporating social relationship interactions. In *Proceedings of the 16th ACM international conference on supporting group work* (pp. 97–106). New York, NY, USA: ACM. http://dx.doi.org/10.1145/1880071.1880087. http://doi.acm.org/10.1145/1880071.1880087.

Gemmell, J., Schimoler, T., Mobasher, B., & Burke, R. (2012). Resource recommendation in social annotation systems: A linear-weighted hybrid approach. *Journal of Computer and System Sciences, 78*, 1160–1174. http://dx.doi.org/10.1016/j.jcss.2011.10.006. http://www.sciencedirect.com/science/article/pii/S0022000011001127.

Gunawardana, A., & Meek, C. (2009). A unified approach to building hybrid recommender systems. In *Proceedings of the third ACM conference on recommender systems* (pp. 117–124). New York, NY, USA: ACM. http://dx.doi.org/10.1145/1639714.1639735. http://doi.acm.org/10.1145/1639714.1639735.

Kagie, M., van Wezel, M., & Groenen, P. J. (2008). Choosing attribute weights for item dissimilarity using clikstream data with an application to a product catalog map. In *Proceedings of the 2008 ACM conference on recommender systems* (pp. 195–202). New York, NY, USA: ACM. http://dx.doi.org/10.1145/1454008.1454040. http://doi.acm.org/10.1145/1454008.1454040.

Karagiannidis, S., Antaris, S., Zigkolis, C., & Vakali, A. (2010). Hydra: an open framework for virtual-fusion of recommendation filters. In *Proceedings of the fourth ACM conference on recommender systems* (pp. 229–232). New York, NY, USA: ACM. http://dx.doi.org/10.1145/1864708.1864754. http://doi.acm.org/10.1145/1864708.1864754.

Kunaver, M., Porl, T., Poganik, M., & Tasi, J. (2007). Optimisation of combined collaborative recommender systems. *AEU – International Journal of Electronics and Communications, 61*, 433–443. http://dx.doi.org/10.1016/j.aeue.2007.04.003. http://www.sciencedirect.com/science/article/pii/S1434841107001070.

Leite Dantas Bezerra, B., & Tenorio de Carvalho, F. d. A. (2011). Symbolic data analysis tools for recommendation systems. *Knowledge and Information Systems, 26*, 385–418.

Lemire, D., & Maclachlan, A. (2007). Slope one predictors for online rating-based collaborative filtering. CoRR abs/cs/0702144.

Linden, G., Smith, B., & York, J. (2003). Amazon.com recommendations: Item-to-item collaborative filtering. *IEEE Internet Computing, 7*, 76–80.

Lops, P., Gemmis, M., & Semeraro, G. (2011). Content-based recommender systems: State of the art and trends. In F. Ricci, L. Rokach, B. Shapira, & P. B. Kantor (Eds.), *Recommender systems handbook* (pp. 73–105). US: Springer. 10.1007/978-0-387-85820-3_3.

McCarthy, K., Salamó, M., Coyle, L., McGinty, L., Smyth, B., & Nixon, P. (2006). Group recommender systems: A critiquing based approach. In *Proceedings of the 11th international conference on Intelligent user interfaces* (pp. 267–269). New York, NY, USA: ACM. http://dx.doi.org/10.1145/1111449.1111506. http://doi.acm.org/10.1145/1111449.1111506.

McNee, S. M., Riedl, J., & Konstan, J. A. (2006). Being accurate is not enough: How accuracy metrics have hurt recommender systems. In *CHI '06 extended abstracts on human factors in computing systems* (pp. 1097–1101). New York, NY, USA: ACM. http://dx.doi.org/10.1145/1125451.1125659. http://doi.acm.org/10.1145/1125451.1125659.

Mussweiler, T. (2003). Comparison processes in social judgment: mechanisms and consequences. In: Psychological review. (pp. 472–489).

Resnick, P., Iacovou, N., Suchak, M., Bergstrom, P., & Riedl, J. (1994). Grouplens: an open architecture for collaborative filtering of netnews. In *Proceedings of the 1994 ACM conference on Computer supported cooperative work* (pp. 175–186). New York, NY, USA: ACM. http://dx.doi.org/10.1145/192844.192905. http://doi.acm.org/10.1145/192844.192905.

Resnick, P., & Varian, H. R. (1997). Recommender systems. *Communications of the ACM, 40*, 56–58. http://dx.doi.org/10.1145/245108.245121. http://doi.acm.org/10.1145/245108.245121.

Schafer, J. B., Frankowski, D., Herlocker, J., & Sen, S. (2007). *The adaptive web*. Berlin, Heidelberg.: Springer-Verlag. chapter Collaborative filtering recommender systems, pp. 291–324. http://dl.acm.org/citation.cfm?id=1768197.1768208..

Sismanis, N., Pitsianis, N., & Sun, X. (2012). Parallel search of *k*-nearest neighbors with synchronous operations, in: IEEE conference on high performance extreme computing (HPEC) (pp. 1–6) doi:http://dx.doi.org/10.1109/HPEC.2012.6408667.

Su, X., & Khoshgoftaar, T. M. (2009). A survey of collaborative filtering techniques. *Advances in Artificial Intelligence, 2009*, 4:2. http://dx.doi.org/10.1155/2009/421425.

Trad, M. R., Joly, A., & Boujemaa, N. (2012). Distributed *knn*-graph approximation via hashing. In *Proceedings of the 2nd ACM international conference on multimedia retrieval*. http://dx.doi.org/10.1145/2324796.2324847, pp. 43:1–43:8. http://doi.acm.org/10.1145/2324796.2324847.

Zhang, M., & Hurley, N. (2008). Avoiding monotony: Improving the diversity of recommendation lists. In *Proceedings of the 2008 ACM conference on recommender systems* (pp. 123–130). New York, NY, USA: ACM. http://dx.doi.org/10.1145/1454008.1454030. http://doi.acm.org/10.1145/1454008.1454030.

Please cite this article in press as: Zigkolis, C., et al. Integrating similarity and dissimilarity notions in recommenders. *Expert Systems with Applications* (2013), http://dx.doi.org/10.1016/j.eswa.2013.03.018