A Class-based Strategy to User Behavior Modeling in Recommender Systems*

Roberto Saia, Ludovico Boratto, and Salvatore Carta

Abstract A recommender system is a tool employed to filter the huge amounts of data that companies have to deal with, and produce effective suggestions to the users. The estimation of the interest of a user toward an item, however, is usually performed at the level of a single item, i.e., for each item not evaluated by a user, canonical approaches look for the rating given by similar users for that item, or for an item with similar content. Such approach leads toward the so-called *overspecialization/serendipity* problem, in which the recommended items are trivial and users do not come across surprising items. This work first shows that user preferences are actually distributed over a small set of classes of items, leading the recommended items to be too similar to the ones already evaluated, then we propose a novel model, named *Class Path Information (CPI)*, able to represent the current and future preferences of the users in terms of a ranked set of classes of items. The proposed approach is based on a semantic analysis of the items evaluated by the users, in order to extend the ground truth and infer their future preferences. The performed experiments show that our approach, by including in the *CPI* model the same classes predicted by a state-of-the-art recommender system, is able to accurately model the user preferences in terms of classes, instead of in terms of single items, allowing to recommend non trivial items.

Roberto Saia, Ludovico Boratto, and Salvatore Carta
Dipartimento di Matematica e Informatica, Università di Cagliari, Via Ospedale 72 - 09124 Cagliari, Italy, e-mail: {roberto.saia,ludovico.boratto,salvatore}@unica.it

* This work is partially funded by Regione Sardegna under project SocialGlue, through PIA - Pacchetti Integrati di Agevolazione "Industria Artigianato e Servizi" (annualità 2010), and by MIUR PRIN 2010-11 under project "Security Horizons".

1 Introduction

The goal of a recommender system is to produce meaningful suggestions for the users, related to items or products that might interest them [23] (e.g., goods on Amazon, movies on Netflix, and so on). In this context, the literature highlights that the rating prediction represents the core task of a recommender system [23, 4]. The importance of this aspect has been further evidenced by the Netflix prize [8], and recent studies showed its effectiveness also in improving classification tasks [2, 7, 33]. However, there are widely-known problems in the recommendation process.

Overspecialization/Serendipity. Independently from the approach used to build the predictions, recommender systems usually suggest items that have a strong similarity with the user profile, consequently the user always receives recommendations for items very similar to those that she/he already considered and never receives suggestions for unexpected, surprising, and novel items. This recommender systems limit, known in the literature as overspecialization/serendipity problem, worsens the user experience and does not give the users the opportunity to explore new items and to improve their knowledge [28]. It is known that this problem affects both the most used recommendation strategies, i.e., the content-based [16] and the collaborative filtering approaches [36]. In fact, on the one hand content-based recommender systems build their predictions by calculating the similarity between the items' content, while on the other hand collaborative filtering looks for items evaluated by the users similar to the target user who has to receive the recommendations. In the literature, several researches also highlight that the serendipity of a resource can be computed by measuring its distance from the items previously considered by the target user [16, 28, 14, 35].

Preference stability. To complicate the previous scenario, there are domains like movies in which the preferences tend to be stable over time [9] (i.e., users tend to watch movies of the same genres or by the same director/actor). This is useful to maintain high-quality knowledge sources, but does not allow a system to diversify the recommendations. Preference stability also leads to the fact that when users get in touch with diverse items, diversity is not valued [19]. On the one side, users tend to access to agreeable information (a phenomenon known as *filter bubble* [21]) and this leads to the overspecialization problem, while on the other side they do not want to face diversity.

Our contributions. In this paper we want to address the following research question: *can we exploit user preferences and represent them in a broader way. The goal is to suggest non trivial items, but not too diverse from those the user already evaluated?* In order to face this problem, we present a representation model, named *Class Path Information (CPI)*, built as a ranking of the classes of items that each user prefers. The *CPI* model is built with a novel approach that performs a semantic analysis of the items already evaluated by a user, in order to extend the ground truth and infer if the terms used to describe the items evaluated by a user that belong to a class (e.g., the movies of a specific genre) also characterize other classes of items, which the user may have or may have not evaluated. By modeling user preferences in terms of classes and by predicting where the future preferences of the users

will go, a recommender system can generate serendipity without recommending to the users something too far from their preferences. Moreover, by understanding the context in which recommendations should be produced in terms of classes, we avoid calculating the semantic distance between single items, which is a heavy process in terms of computational costs. Another advantage offered by this approach is that the generated models can be used to produce recommendations with any approach. Indeed, the *CPI* provides information of the classes of items the user prefers, which can be exploited by any recommendation technique.

The main contributions coming from our proposal are the following:

- we show that preference stability exists in terms of classes of items. An analysis performed on two real-world datasets shows that user preferences are distributed over a small set of classes;
- we characterize each class of items using a set of *Semantic Binary Sieves* (SBS), a novel type of filter able to weigh the relevance of each class for each user;
- we develop an algorithm able to evaluate a relevance score of each class of items for each user by using the *SBS* filters;
- we introduce the novel concept of *Class Path Information* (CPI) model, which builds a relevance score of the classes of items each user prefers, and define an algorithm to create it;
- we evaluate our approach on a real-world dataset and show that the classes available in the model have a large overlap with those of the items predicted by a state-of-the-art recommender system.

This paper extends the work presented in [25] in the following ways:

(i) we define the practical implementation of the *Semantic Binary Sieves*, by introducing the algorithm needed to build them, explaining in detail how each step works (Section 4.3.1);
(ii) we define the practical implementation of the *Class Path Information*, by introducing the algorithm needed to build it, explaining in detail how each step works (Section 4.4.1);
(iii) we formalize the concept of *CPI Valid Length*, a parameter that aims to determine the number of CPI elements (CPI length) to be taken into account, on the basis of the operating environment (Section 4.4.2);
(iv) we perform a series of additional experiments aimed to compare the computational load of our approach, with that of SVD, reporting the differences in terms of time needed to complete a recommendation process (Section 5.4.3).

Roadmap. The rest of the paper is organized as follows: Section 2 provides a background on the concepts handled by our proposal and the formal definition of our problem; Section 3 presents an analysis of preference stability on two real-world datasets; Section 4 describes the details of the proposed approach to model user preferences in terms of classes; Section 5 describes the experimental framework used to evaluate our proposal; Section 6 discusses related work; Section 7 contains conclusions and future work.

2 Preliminaries

Background. For many years the item descriptions were analyzed with a word vector space model, where all the terms of each item description are processed by TF-IDF [26] and stored in a weighted vector of terms. Due to the fact that this approach based on a simple *bag of words* is not able to perform a semantic disambiguation of the words in an item description, and motivated by the fact that exploiting a taxonomy for categorization purposes is an approach recognized in the literature [3] and by the fact that a semantic analysis is useful to improve the accuracy of a classification [5, 6], we decided to exploit the functionalities offered by the WordNet environment. Wordnet is a large lexical database of English, where *nouns*, *verbs*, *adjectives*, and *adverbs* are grouped into sets of cognitive synonyms (synsets), each expressing a distinct concept. Synsets are interlinked by means of conceptual-semantic and lexical relations.

Wordnet currently contains about 155,287 words, organized into 117,659 synsets for a total of 206,941 word-sense pairs [12]. In a short, the main relation among words in WordNet is the synonymy and the synsets are unordered sets of grouped words that denote the same concept and are interchangeable in many contexts. Each synset is linked to other synsets through a small number of *conceptual relations*. Word forms with several distinct meanings are represented in as many distinct synsets, in this way each form-meaning pair in WordNet will be unique (e.g., the *fly* noun and the *fly* verb belong to two distinct synsets). Most of the WordNet relations connect words that belong to the same part-of-speech (POS). There are four POS: *nouns*, *verbs*, *adjectives*, and *adverbs*. Both nouns and verbs are organized into precise hierarchies, defined by hypernym or *is-a* relationships. For example, the first sense of the word *radio* would have the following hypernym hierarchy, where the words at the same level are synonyms of each other: some sense of *radio* is synonymous with some other senses of *radiocommunication* or *wireless*, and so on. Each synset has a unique index and shares its properties, such as a gloss or dictionary definition. We use the synsets to perform both the definition of binary filters and the evaluation of the relevance scores of the classes in a user profile.

Notation. We are given a set of users $U = \{u_1, \ldots, u_N\}$, a set of items $I = \{i_1, \ldots, i_M\}$, and a set V of values used to express the user preferences (e.g., $V = [1,5]$ or $V = \{like, dislike\}$). The set of all possible preferences expressed by the users is a ternary relation $P \subseteq U \times I \times V$. We denote as $P_+ \subseteq P$ the subset of preferences with a positive value (i.e., $P_+ = \{(u,i,v) \in P | v \geq \overline{v} \vee v = like\}$), where $\overline{v}$ indicates the mean value (in the previous example, $\overline{v} = 3$). Moreover, we denote as $I_+ = \{i \in I | \exists (u,i,v) \in P_+\}$ the set of items for which there is a positive preference, and as $I_u = \{i \in I | \exists (u,i,v) \in P_+ \wedge u \in U\}$ the set of items a user u likes. Let $C = \{c_1, \ldots, c_K\}$ be a set of classes used to classify the items; we denote as $C_i \subseteq C$ the set of classes used to classify an item i (e.g., C_i might be the set of genres that a movie i was classified with), and with $C_u = \{c \in C | \exists (u,i,v) \in P_+ \wedge i \in C_i\}$ the classes associated to the items that a user likes.

Let $BoW = \{t_1, \ldots, t_W\}$ be the bag of words used to describe the items in I; we denote as d_i be the binary vector used to describe each item $i \in I$ (each vector is

such that $|d_i| = |BoW|$). We define as $S = \{s_1, \ldots, s_W\}$ the set of synsets associated to BoW (that is, for each term used to describe an item, we consider its associated synset), and as sd_i the semantic description of i. The set of semantic descriptions is denoted as $D = \{sd_1, \ldots, sd_M\}$ (note that we have a semantic description for each item, so $|D| = |I|$). The approach used to extract sd_i from d_i is described in detail in Section 4.

Problem Definition. Given a set of positive preferences P_+ that characterize the items each user likes, a set of classes C used to classify the items, and a set of semantic descriptions D, our goal is to assign a relevance score $r_u(c)$ for each user u and each class c, based on the semantic descriptions D. Each relevance score will be combined into a model CPI_u, defined as follows:

$$CPI_u = (r_u(c_1), \ldots, r_u(c_K)) \tag{1}$$

Each CPI_u must respect the following properties:

- $r_u(c_1) \geq ... \geq r_u(c_K)$
- $CPI_u \supseteq C_u$

So, each CPI model contains a list of classes ranked by relevance score and the classes available in the model are a superset of the classes for which a user expressed a preference (i.e., we are going to predict the future preferences of the users, based on the semantic analysis of the items she/he likes).

3 Characterizing Preference Stability

In order to understand if preference stability can be characterized in terms of the classes used to classify the items, in this section we are going to present the distribution of the classes C_u related to the items a user likes. For each user $u \in U$ and each class $c \in C_u$, we consider how many positive preferences the user expressed for that class. We call this value the *popularity* of the class for that user, and define it as the percentage of items that the user likes and belong to that class:

$$popularity(u,c) = \frac{|\{(u,i,v) \in P_+ | i \in c\}|}{|\{(u,i,v) \in P_+\}|} \tag{2}$$

Then, we ordered the *popularity* values of each user in decreasing order and average all the *popularity* values at the position j in the list of each user (i.e., if $j = 1$, we calculate the average amount of preferences each user expressed for the items in the most popular class).

The study has been performed on the following real-world datasets:

Yahoo! Webscope R4[2]**.** The dataset contains a large amount of data related to users preferences expressed by the Yahoo! Movies community that are rated on the base of two different scales, from 1 to 13 and from 1 to 5 (we have chosen to use

[2] http://webscope.sandbox.yahoo.com

the latter). The training data is composed by 7,642 users ($|U|$), 11,915 movies/items ($|I|$), and 211,231 ratings ($|R|$), and all users involved have rated at least 10 items and all items are rated by at least one user. The test data is composed by 2,309 users, 2,380 items, and 10,136 ratings. There are no test users/items that do not also appear in the training data. Each user in the training and test data is represented by a unique ID. As shown in Table 1, the items are classified by Yahoo in 20 different classes (movie genres), and it should be noted that each item may be classified in multiple classes.

Class	*Genre*	*Class*	*Genre*
1	Comedy	11	Reality
2	Drama	12	Kids/Family
3	Action/Adventure	13	Crime/Gangster
4	Miscellaneous	14	Romance
5	Suspense/Horror	15	Western
6	Sci-Fi/Fantasy	16	Musical/Arts
7	Thriller	17	Documentary
8	Art/Foreign	18	Special Interest
9	Animation	19	Adult Audience
10	Horror	20	Features

Table 1 Yahoo! Webscope R4 Genres

Movielens 10M[3]**.** This dataset contains 10,000,054 ratings and 95,580 tags related to 10,681 movies by 71,567 users that were selected at random from MovieLens (a movie recommendation website). All the users in the dataset had rated at least 20 movies, and each user is represented by a unique ID. The ratings of the items are based on a *5-star* scale, with *half-star* increments. As shown in Table 2, in this dataset the items are classified by Movielens in 18 different classes (movie genres), and it should be noted that also in this case each item may be classified in multiple classes.

Fig. 1 and Fig. 2 show the distribution of the popularities for the Yahoo! Webscope R4 and the Movielens 10M datasets. In Fig. 1, we can see that 41% of the preferences are all in a single class (in other words, nearly half of the positive ratings given by the users are for the same genre of movies) and, by considering as characterizing only the classes with $popularity \geq 1\%$, it is possible to observe that user preferences are distributed on 6 out of 20 classes. Fig. 2 shows that in the Movielens dataset preference stability has a lower impact. In fact, 26% of the ratings are in the most important class for each user, and 10 out of 18 classes are involved in the user preferences.

This analysis showed that preferences stability exists in terms of classes, and that user preferences are distributed between 30% and 55% of the classes. Based on

[3] http://grouplens.org/datasets/movielens/

Class	*Genre*	*Class*	*Genre*
1	Action	10	Film-Noir
2	Adventure	11	Horror
3	Animation	12	Musical
4	Children's	13	Mystery
5	Comedy	14	Romance
6	Crime	15	Sci-Fi
7	Documentary	16	Thriller
8	Drama	17	War
9	Fantasy	18	Western

Table 2 Movielens 10M Genres

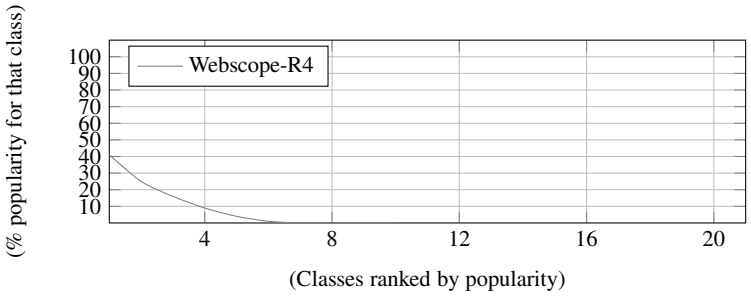

Fig. 1 Webscope-R4 - Involved Classes in the User Preferences

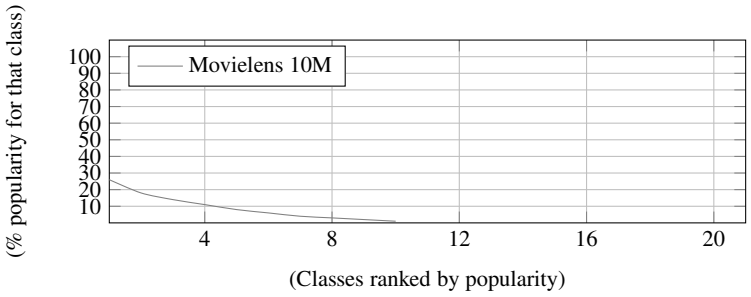

Fig. 2 Movielens 10M - Involved Classes in the User Preferences

these results, in the next section we are going to deepen our knowledge on the user preferences in terms of classes, in order to accurately model them.

4 Our Approach

In this section we present our approach, which performs a semantic analysis of the descriptions of the items the users like, in order to build a model that infers where the future preferences will go. The goal is to understand which terms used to describe an item that a user likes characterize other classes of items. Our modeling approach performs four steps:

1. **Text Preprocessing**: processing of the textual information (description, title, etc.) present in all items, in order to remove the useless elements for the subsequent operation of synset retrieving;
2. **User Modeling**: creation of a model that contains which synsets are present in the items a user likes;
3. **Semantic Binary Sieve Construction**: creation of a *binary vector* for each class of items, and subsequent definition of the *Semantic Binary Sieves* (SBS), a series of filters that we use to estimate which synsets are relevant for that class;
4. **Class Path Information Modeling**: definition of the *Class Path Information* (CPI) model that, based on the semantic analyses performed in the previous steps, infers the user preferences in terms of classes.

Note that all steps are based on the use of WordNet synsets, which allow us to consider the semantics of the content, without performing complex operations on it. In the following, we will describe in detail how each step works.

4.1 Text Preprocessing

Before extracting the WordNet synsets from the text that describes each item, we need to follow several preprocessing tasks. The first is to detect the correct *Part-Of-Speech* (POS) for each word in the text; in order to perform this task, we used the *Stanford Log-linear Part-Of-Speech Tagger* [32].Then, we remove punctuation marks and *stop-words*, which represent noise during the semantic analysis. Several *stop-words* lists can be found in the Internet, and in this work we used a list of 429 *stop-words* made available with the *Onix Text Retrieval Toolkit*[4]. Andfter we have determined the lemma of each word using the Java API implementation for WordNet Searching JAWS[5], we perform the so-called word sense disambiguation, a process where the correct sense of each word is determined. The best sense of each word in a sentence was found using the Java implementation of the adapted Lesk algorithm provided by the *Denmark Technical University* similarity application [27]. All the collected synsets form the set $S = \{s_1, \ldots, s_W\}$ defined in Section 2. The output of this step is the semantic disambiguation of the textual description of each item $i \in I$,

[4] http://www.lextek.com/manuals/onix/stopwords.html

[5] http://lyle.smu.edu/ tspell/jaws/index.html

which is stored in a binary vector ds_i; each element of the vector $ds_i[w]$ is 1 if the corresponding synset appears in the item description, and 0 otherwise.

4.2 User Modeling

For each user $u \in U$, this step considers the set of items I_u she/he likes, and builds a user model m_u that describes which synsets characterize the user profile (i.e., which synsets appear in the semantic description of these items). Each model m_u is a binary vector that contains an element for each synset $s_w \in S$.

In order to build the vector, we consider the semantic description ds_i of each item $i \in I_u$ for which the user expressed a positive preference. This step builds m_u, by performing the following operation on each element w:

$$m_u[w] = \begin{cases} 1, \; if \; ds_i[w] = 1 \\ m_u[w], \; otherwise \end{cases} \tag{3}$$

This means that if the semantic description of an item i contains the synset s_w, the synset becomes relevant for the user, and we set to 1 the bit at position w in the user model m_u; otherwise, its value remains unaltered. By performing this operation for all the items $i \in I_u$, we model which synsets are relevant for the user. The output of this step is a set $M = \{m_1, \ldots, m_N\}$ of user models (note that we have a model for each user, so $|M| = |U|$).

4.3 Semantic Binary Sieve Construction

For each class $c \in C$, we create a binary vector that will store which synsets are relevant for that class. These vectors, called *Semantic Binary Sieves*, will be stored in a set $B = \{b_1, \ldots, b_K\}$ (note that $|B| = |C|$, since we have a vector for each class). Each vector $b_k \in B$ contains an element for each synset $s_w \in S$ (i.e., $|b_k| = |S|$).

In order to build the vector, we consider the semantic description ds_i of each item $i \in I_+$ for which there is a positive preference, and each class c_k with whom i was classified. The binary vector b_k will store which synsets are relevant for a class c_k, by performing the following operation on each element $b_k[w]$ of the vector:

$$b_k[w] = \begin{cases} 1, \; if \; ds_i[w] = 1 \wedge i \in c_k \\ b_k[w], \; otherwise \end{cases} \tag{4}$$

In other words, if the semantic description of an item i contains the synset s_w, the synset becomes relevant for each class c_k that classifies i, and the semantic binary sieve b_k associated to c_k has the bit at position w set to 1; otherwise, its value remains unaltered. By performing this operation for all the items $i \in I_+$ that are classified with c_k, we know which synsets are relevant for the class.

4.3.1 Algorithm

Based on the above considerations, we can now define the Algorithm 1 used to create a *Semantic Binary Sieve* for each class (i.e., genre) of items. The algorithm takes as input a class $c \in C$ and the set of items I, and returns as output the SBS related with the class c.

Algorithm 1 Create SBS

Input: c=class, I= set of items
Output: SBS=Semantic Binary Sieves of c
1: $N = |GetDistinctSynsets(I)|$
2: $SBS = NewVector[N]$
3: $I_c = GetClassItems(c)$
4: **for** each item i in I_c **do**
5: $d = GetItemDescription(i)$
6: $S \leftarrow RetrieveAllSynsets(d)$
7: **for** each s in S **do**
8: $idx = GetSynsetIndex(s)$
9: $SBS[idx] = 1$
10: **end for**
11: **end for**
12: Return SBS

We start by storing in N the number of distinct synsets in the set I, which represents our synset ontology (step 1), then we create a SBS vector of size N (step 2). In the next step, we put in the set I_c all items that belong to a class c (step 3). For each of these items, we extract the description and the related synsets (steps 5 and 6), setting to 1 the element of the vector SBS that correspond with the index of these synsets (steps from 7 to 10). The algorithm ends by returning the SBS vector.

4.4 Class Path Information Modeling

This step compares the output of the two previous steps (i.e., the set B of binary vectors related to the *Semantic Binary Sieves*, and the set M of binary vectors related to the *user models*), in order to infer which classes are relevant for a user and where the future user preferences will go. The main idea is to consider which synsets are relevant for a user u (this information is stored in the user model m_u) and evaluate which classes are characterized by the synsets in m_u (this information is contained in each vector b_k, which contains the synsets that are relevant for the class c_k). The objective is to build a relevance score $r_u[k]$, which indicates the relevance of the class c_k for the user u.

The key concept behind this step is that *we do not consider the items a user evaluated anymore*. Each vector in B is used as a filter (for this reason the vectors are called *semantic binary sieves*), which allows us to estimate the relevance of

each class for that user. Therefore, the relevance score of a class for a user can be used to infer where the future preferences of the users will go, since *a user might be associated to classes of items she/he never expressed a preference for, but characterized by synsets that also characterize the user model*. By ordering the relevance scores in decreasing order (from the most to the least relevant), we can build a model, named *Class Path Information (CPI)*, which can be used to generate recommendations for the users. Indeed, a recommender system might use this model to know which classes are relevant for the user, and with which score.

By considering each semantic binary sieve $b_k \in B$ associated to the class c_k and the user model m_u, we define a matching criterion Θ between each synset $m_u[w]$ in the user model, and the corresponding synset $b_k[w]$ in the semantic binary sieve, by adding 1 to the relevance score of that class for the user (element $r_u[k]$) if the synset is set to 1 both in the semantic binary sieve and in the user model, and leaving the current value as it is otherwise. The semantics of the operator is shown in Equation (5).

$$b_k[w]\Theta m_u[w] = \begin{cases} r_u[k]++, \; if \; m_u[w] = 1 \wedge b_k[w] = 1 \\ r_u[k], \; otherwise \end{cases} \tag{5}$$

By comparing a user model m_u with each vector $b_k \in B$, we obtain a vector r_u that contains the relevance score of each class for the user (i.e., $|r_u| = |C|$). The relevance scores of each class for each user are sorted in decreasing order to build the *CPI* model for a user u (i.e., each model respects the following property: $r_u(c_1) \geq \ldots \geq r_u(c_K)$):

$$CPI_u = (r_u(c_1), \ldots, r_u(c_K)) \tag{6}$$

The output of this step is a Class Path Information model CPI_u for each user $u \in U$.

4.4.1 Algorithm

Algorithm 2 is used to create a *Class Path Information* for a user. It requires as input the set of items I, the user profile I_u (i.e., the set of items positively evaluated by the user u), and the set of classes C. It returns as output the CPI of a user u, which represents her/his preferences in terms of classes, sorted in decreasing order.

Algorithm 2 Create CPI

Input: I= set of items, I_u= User profile, C=Classes of items
Output: CPI = Class Path Information of u

1: $d = GetItemsDescription(I_u)$
2: $S \leftarrow RetrieveAllSynsets(d)$
3: $CPI = NewVector[|C|]$
4: **for** each class c in C **do**
5: $SBS = CreateSBS(c, I)$
6: **for** each s in S **do**
7: $x = GetSynsetIndex(s)$
8: $y = GetClassIndex(c)$
9: **if** $SBS[x] == 1$ **then**
10: $CPI[y] += 1$
11: **end if**
12: **end for**
13: **end for**
14: $CPI \leftarrow Sort(CPI, Descending)$
15: Return CPI

We start by appending in d the descriptions of the items in the user profile I_u, retrieving the related synsets (steps 1 and 2). In the step 3 we create a CPI vector of size $|C|$ (i.e., the number of classes), then for each class $c \in C$, we retrieve its SBS (step 5). In the steps from 6 to 12, we check each synset extracted from the user profile, and when the corresponding synset value in the SBS is 1, we increase by 1 unit the element of the vector CPI that corresponds with the class that we are processing. The algorithm ends by returning the CPI vector, sorted in decreasing order (steps 14 and 15).

4.4.2 CPI Valid Length

A possible way to determine the number of CPI elements to be taken into account, is by introducing the parameter θ, which denotes the number of CPI elements to consider. For instance, given $\theta = 2$, we take into account only the classes stored in the CPI with a $weight > 2$. We calculate the optimal value of θ as showed in Equation 7, where $Max(C)$ denotes the maximum value of weight assumed by a class, while $|U|$ and $|C|$ denote, respectively, the number of users and classes, and the $Zeros(C)$ denotes the number of classes with a value equal to 0.

$$\theta = \left\lfloor \frac{Max(C)}{(|U| \cdot |C|) - Zeros(C)} \cdot Classes \right\rfloor \tag{7}$$

The CVL (Check Valid Length) Algorithm 3 implements this operation. It requires as input the CPI of a user u and the θ value, and returns the CPI valid length for the user u, i.e., the part of CPI to be taken into account. For each class stored in the CPI value (step 2), it counts how many classes have a weight higher than θ (step

3), returning this value when there is a class with a weight not greater than θ (step 5).

Algorithm 3 Get CVL

Input: CPI_u= CPI value of the user u, θ=Weight level
Output: L = Valid length of CPI
1: L=0
2: **for** each class c in CPI_u **do**
3: **if** $Weight(c) > \theta$ **then** L++
4: **else**
5: Return L
6: **end if**
7: **end for**

5 Experimental Framework

The experimental framework was developed by using a machine with an Intel Pentium CPU P6100 Dual Core (2 GHz $\times$ 2) and a Linux 64-bit Operating System (Debian Wheezy) with 4 GBytes of RAM. The environment for this work is based on the Java language, with the support of Java API implementation for WordNet Searching (JAWS) to perform the semantic measures, and the support of Apache Mahout[6] Java framework to implement the state-of-the-art approach that we compare our CPI modeling approach with.

This section first describes the dataset and the preprocessing performed on the data, then we describe the strategy used to perform the evaluation, the metrics, and we conclude by presenting the experimental results.

5.1 Dataset and Data Preprocessing

We performed our experiments using the Yahoo! Webscope Movie dataset (R4) described in Section 3. Note that we had to limit our evaluation only to one of the two datasets previously considered, since the Movielens 10M dataset does not contain any textual description of the items.

In order to create a binary sieve for each class used to build a CPI model for every user (we take in account only the 2,309 users available in the test set), we need to define an ontology of synsets based on the descriptions of the items. To perform this operation we considered the description and title of each movie, and since the used algorithm takes into account only the items with a rating above the average, we selected only the movies with a rating ≥ 3.

[6] https://mahout.apache.org

5.2 Strategy

The objective of our approach is to create a model that infers the preferences of the users in terms of classes, not only relying on the ground truth. As stated in the motivation of our work, the main domain of application that could benefit of this modeling approach are the recommender systems, which build predictions for the items not yet evaluated. Therefore, we applied a state-of-the-art recommender system to our dataset, and evaluated for each user u the set of classes C_u for which a positive value was predicted, and compared them with the CPI_u model built for that user.

The system chosen for the comparison is SVD++ [34], the Koren's version of SVD [10] that has been proved to be one of the most accurate approaches. The SVD++ approach, which we implemented through the Mahout functionalities, in addition to the training dataset requires two additional parameters: the number of target features and the number of training steps to run. After a training of the parameters, the algorithm was run with the following setting: the first parameter would be equivalent to the number of involved genres, thus we have set this value to 20; about the second parameter, considering that larger values mean longer training time, and that we have not experienced significant improvements with higher values, we have chosen the value of 4.

We required the system to produce N recommendations for each user and tested different values of the parameter (more specifically, $N = \{20, 40, ..., 100\}$). The classes involved in the recommended items were almost identical in all the settings, therefore we chose $N = 100$ to perform an evaluation in which as much information as possible was available for the comparison.

We compare the results in relation about three different aspects:

- **Evaluation of preference stability.** We perform the same analysis performed in Section 3, and measure the average number of classes involved in our *CPI* models and their $popularity$ (i.e., how many positive preferences are associated to each class). The objective is to understand how capable our modeling approach is at reducing the effects of preference stability (which, as highlighted in the Introduction, introduces overspecialization problems), by extending the ground truth;
- **Evaluation of the classes included in the model.** In order to evaluate the significance of the produced models, we evaluate the overlap between the classes produced by the *CPI* model and the classes involved in the items recommended by SVD++, by measuring the Jaccard index between the sets C_u and CPI_u;
- **Evaluation of the computational load.** We evaluate the computational load of our approach, by comparing it with the SVD one, through a series of experiments aimed to measure the differences in terms of time needed to complete a recommendation process.

5.3 Metrics

The *popularity* metric, which allows to measure preference stability, was already introduced in Section 3 (Equation 2). The evaluation of the classes included in the model has been performed by measuring the *Jaccard* index, in order to measure the overlap between the classes included in our *CPI* model (denoted as $C_u(CPI)$), and those that classify the items recommended by SVD++ (denoted as $C_u(SDV++)$):

$$J(C_u(CPI), C_u(SDV++)) = \frac{|C_u(CPI) \cap C_u(SDV++)|}{|C_u(CPI) \cup C_u(SDV++)|} \quad (8)$$

During this operation we considered the three most relevant classes identified by the two approaches.

5.4 Experimental Results

This section presents the results obtained by the three evaluations previously presented.

5.4.1 Evaluation of preference stability

Fig. 3 shows preference stability for both approaches. The results show that the effect of preference stability is strongly reduced by our approach. In fact the number of classes involved in the model is now 10 (remember that in Section 3 we showed that user preferences were distributed over 6 classes). This shows an important first result, which is the capability of our approach to extend the ground truth and to be able to characterize user preferences over a larger set of classes, without considering the preferences of the other users. By enlarging the set of classes, we can also see that the *popularity* of each class (i.e., the number of preferences expressed for the items that belong to a class), is also strongly reduced; indeed, we move from 46% of preferences that characterize the main class of each user to a 27.2% value. We can also notice that the amount of classes inferred by our model that has a $popularity \geq 1\%$ is exactly the same available in the $C_u(SVD++)$ models. This means that our approach is able to characterize user preferences in terms of classes without producing any comparison with the preferences of the other users, thus strongly reducing the computational load necessary to infer what a user is going to like.

5.4.2 Evaluation of the classes included in the model

The previous analysis showed that the number of classes involved in our *CPI* models is the same of those produced by the $SVD++$ predictions. However, this result is not enough to validate our model, as the two sets of classes might be completely differ-

ent. For each user we measured the Jaccard index on the sets of classes and averaged the obtained values. The average value of all results is 0.8218, which demonstrates that the most popular classes recommended through our CPI model are almost the same of the SVD++ approach. This demonstrates that in spite its simplicity, the CPI approach operates within the same range of items of a canonical approach at the state-of-the-art.

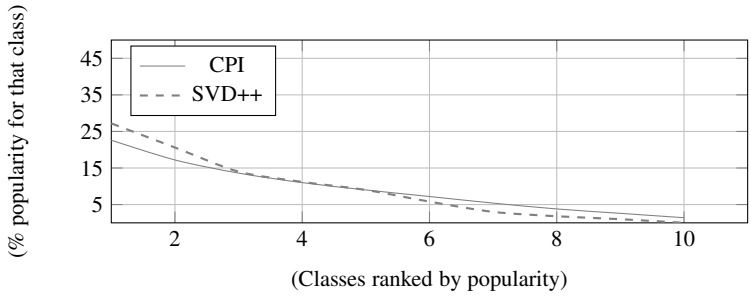

Fig. 3 Class Distribution by Popularity

5.4.3 Evaluation of the Computational Load

In order to compare the computational load of our approach, with that of SVD, we performed a series of experiments, whose results are presented in Figure 4. We show the differences, between our approach and SVD, in terms of time needed to complete a recommendation process. In more detail, we compare the performance by comparing the time needed to recommend N items, with $N = \{10, 20, ..., 100\}$. The value in the Y axis denotes the elapsed time needed to recommend the items, and the value on the X axis denotes the number of the recommendations. As we can observe, the results show a considerable difference in term of computational load, in favor of our approach.

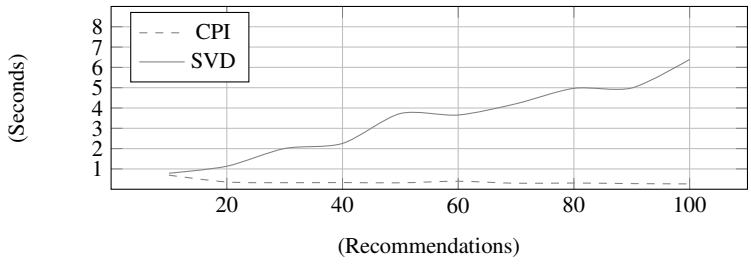

Fig. 4 Computational Load Comparison

6 Related Work

Likewise to other contexts, in recommender systems the preferences of the users about new choices (goods or services) tend to follow the behavior of the other users with similar tastes. This is a well-known phenomenon, called *homophily*, that in the recommender systems environment is embraced by the most common strategies used to recommend new items (e.g., *content-based* [16] and *collaborative-filtering* [11, 15] approaches). If on the one hand this approach leads toward items of likely interest to users, on the other hand it reduces the range of items of potential interest that a system could recommend, augmenting the serendipity problem, in a scenario known as the *filter bubble* [21]. The *serendipity* problem, i.e., the ability for a recommender system to suggest items of potential interest to the user that are not trivial, i.e., too similar to those in a user profile. In some works, such as [28], *serendipity* is briefly described as *a measure of how surprising the successful recommendations are*. The same work discusses the *serendipity* as the deviation from the *natural* prediction [20], and introduces the opportunity to estimate an optimal deviation value to use in order to make recommendations, underlining the risk related to an inappropriate measurement, which can lead toward a loss of user trust in regard to the recommendation system. A disadvantage that affects those recommender systems that take into account the diversity [1] is that they still operate within the classes of items for which users have expressed an explicit liking.

Another well-known problem is the so called *selective exposure*, i.e., the tendency of users to make their choices (goods or services) based only on their usual preferences, a typical way to proceed that excludes the possibility for the users to find new items that may be of interest to them [13]. The literature presents several approaches that try to reduce this problem, e.g., the *NewsCube* [22] that operates offering to the users several points of views, in order to stimulate them to make different and not usual choices.

In order to represent the user preferences and improve the effectiveness of the suggestions, in the literature we can find several approaches. For instance, users can be classified based on some explicit features (e.g., their demographic data) by extracting this information from sources such as Twitter [18], or based on other implicit features extracted through a more complex analysis of the same sources [17]. The problem of modeling semantically correlated items has been tackled in [29, 30, 31], by considering the temporal aspect. In [24], an approach to preprocess the users profiles in order to detect and remove the items that generate noise and could make the profiles not adherent to the real tastes of the users is also presented. The *serendipity* problem shows that a challenge in this area is then to identify a method able to make effective predictions, exploiting not only the information present in the user profiles. Our approach faces this problem by introducing a new modeling approach based on the class of items instead of considering the items. This is a new strategy of classification based on the classes that extends the set of possible items to recommend, taking also into account those distant from the previous choices of the users, although within their favorite classes of items. The

use of a *class-based* model is also able to reduce the entity of the *selective exposure* problem, because it selects the items within a broader set of classes.

7 Conclusions and Future Work

The approach proposed in this work represents a novel way to model the user preferences, in terms of classes instead of single items, with the goal to generate non trivial recommendations. It is based on a semantic process able to create models that characterize each class and each user in terms of preferences. Such process starts by generating a set of binary filters (called *Semantic Binary Sieves*) that characterize each class of items, using them to build a class relevance score for each user. The class relevance scores are combined in a ranked list, called *Class Path Information* model, which gives us information about the user preferences in terms of classes.

Experimental results showed the capability of our approach to extend the ground truth and infer where the future preferences of the users will go. Indeed, by comparing our models with the set of classes of the items predicted by a state-of-the-art recommender system, we highlighted a strong overlap between them. This means that our approach is able to accurately infer user preferences in terms of classes, and that the generated models can be employed by a recommender system to select items within the set of classes included in the model. The advantages are both on the computational load, since the system avoids calculating the semantic distance between the items, and on the possibility to recommend items that are not too dissimilar to those she/he already positively evaluated.

A possible follow up of this work could be the implementation of our modeling technique in a recommender system. The objective is to produce the recommendations considering the classes available in the model, which are semantically related to the items a user likes, but not the same. This kind of approach would allow us to reduce the negative consequences caused by diversity, producing serendipitous but effective item recommendations.

References

1. S. Abbar, S. Amer-Yahia, P. Indyk, and S. Mahabadi. Real-time recommendation of diverse related articles. In D. Schwabe, V. A. F. Almeida, H. Glaser, R. A. Baeza-Yates, and S. B. Moon, editors, *22nd International World Wide Web Conference, WWW '13, Rio de Janeiro, Brazil, May 13-17, 2013*, pages 1–12. International World Wide Web Conferences Steering Committee / ACM, 2013.
2. A. Addis, G. Armano, A. Giuliani, and E. Vargiu. A recommender system based on a generic contextual advertising approach. In *Proceedings of the 15th IEEE Symposium on Computers and Communications, ISCC 2010, Riccione, Italy, June 22-25, 2010*, pages 859–861. IEEE, 2010.
3. A. Addis, G. Armano, and E. Vargiu. Assessing progressive filtering to perform hierarchical text categorization in presence of input imbalance. In A. L. N. Fred and J. Filipe, editors, *KDIR*

2010 - Proceedings of the International Conference on Knowledge Discovery and Information Retrieval, Valencia, Spain, October 25-28, 2010, pages 14–23. SciTePress, 2010.
4. G. Adomavicius and A. Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE Trans. Knowl. Data Eng.*, 17(6):734–749, 2005.
5. G. Armano, A. Giuliani, and E. Vargiu. Semantic enrichment of contextual advertising by using concepts. In J. Filipe and A. L. N. Fred, editors, *KDIR 2011 - Proceedings of the International Conference on Knowledge Discovery and Information Retrieval, Paris, France, 26-29 October, 2011*, pages 232–237. SciTePress, 2011.
6. G. Armano, A. Giuliani, and E. Vargiu. Studying the impact of text summarization on contextual advertising. In F. Morvan, A. M. Tjoa, and R. Wagner, editors, *2011 Database and Expert Systems Applications, DEXA, International Workshops, Toulouse, France, August 29 - Sept. 2, 2011*, pages 172–176. IEEE Computer Society, 2011.
7. G. Armano and E. Vargiu. A unifying view of contextual advertising and recommender systems. In A. L. N. Fred and J. Filipe, editors, *KDIR 2010 - Proceedings of the International Conference on Knowledge Discovery and Information Retrieval, Valencia, Spain, October 25-28, 2010*, pages 463–466. SciTePress, 2010.
8. J. Bennett, C. Elkan, B. Liu, P. Smyth, and D. Tikk. Kdd cup and workshop 2007. *SIGKDD Explor. Newsl.*, 9(2):51–52, Dec. 2007.
9. R. D. Burke and M. Ramezani. Matching recommendation technologies and domains. In F. Ricci, L. Rokach, B. Shapira, and P. B. Kantor, editors, *Recommender Systems Handbook*, pages 367–386. Springer, 2011.
10. M. J. P. Daniel Billsus. Learning collaborative information filters. In J. W. Shavlik, editor, *Proceedings of the Fifteenth International Conference on Machine Learning (ICML 1998), Madison, Wisconsin, USA, July 24-27, 1998*, pages 46–54. Morgan Kaufmann, 1998.
11. C. Desrosiers and G. Karypis. A comprehensive survey of neighborhood-based recommendation methods. In F. Ricci, L. Rokach, B. Shapira, and P. B. Kantor, editors, *Recommender Systems Handbook*, pages 107–144. Springer, 2011.
12. C. Fellbaum. *WordNet: An Electronic Lexical Database*. Bradford Books, 1998.
13. L. Festinger. *A theory of cognitive dissonance*, volume 2. Stanford university press, 1962.
14. L. Iaquinta, M. de Gemmis, P. Lops, G. Semeraro, M. Filannino, and P. Molino. Introducing serendipity in a content-based recommender system. In *HIS*, pages 168–173. IEEE Computer Society, 2008.
15. Y. Koren and R. M. Bell. Advances in collaborative filtering. In F. Ricci, L. Rokach, B. Shapira, and P. B. Kantor, editors, *Recommender Systems Handbook*, pages 145–186. Springer, 2011.
16. P. Lops, M. de Gemmis, and G. Semeraro. Content-based recommender systems: State of the art and trends. In F. Ricci, L. Rokach, B. Shapira, and P. B. Kantor, editors, *Recommender Systems Handbook*, pages 73–105. Springer, 2011.
17. M. Michelson and S. A. Macskassy. Discovering users' topics of interest on twitter: a first look. In *Proceedings of the Fourth Workshop on Analytics for Noisy Unstructured Text Data, AND 2010, Toronto, Ontario, Canada, October 26th, 2010 (in conjunction with CIKM 2010)*. ACM, 2010.
18. A. Mislove, S. Lehmann, Y. Ahn, J. Onnela, and J. N. Rosenquist. Understanding the demographics of twitter users. In *Proceedings of the Fifth International Conference on Weblogs and Social Media, Barcelona, Catalonia, Spain, July 17-21, 2011*. The AAAI Press, 2011.
19. S. A. Munson and P. Resnick. Presenting diverse political opinions: How and how much. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '10, pages 1457–1466, New York, NY, USA, 2010. ACM.
20. T. Murakami, K. Mori, and R. Orihara. Metrics for evaluating the serendipity of recommendation lists. In K. Satoh, A. Inokuchi, K. Nagao, and T. Kawamura, editors, *New Frontiers in Artificial Intelligence, JSAI 2007 Conference and Workshops, Miyazaki, Japan, June 18-22, 2007, Revised Selected Papers*, volume 4914 of *Lecture Notes in Computer Science*, pages 40–46. Springer, 2008.

21. E. Pariser. *The Filter Bubble: What the Internet Is Hiding from You*. Penguin Group , The, 2011.
22. S. Park, S. Kang, S. Chung, and J. Song. Newscube: delivering multiple aspects of news to mitigate media bias. In *Proceedings of the 27th International Conference on Human Factors in Computing Systems, CHI 2009, Boston, MA, USA, April 4-9, 2009*. ACM, 2009.
23. F. Ricci, L. Rokach, and B. Shapira. Introduction to recommender systems handbook. In F. Ricci, L. Rokach, B. Shapira, and P. B. Kantor, editors, *Recommender Systems Handbook*, pages 1–35. Springer, 2011.
24. R. Saia, L. Boratto, and S. Carta. Semantic coherence-based user profile modeling in the recommender systems context. In *Proceedings of the 6th International Conference on Knowledge Discovery and Information Retrieval, KDIR 2014, Rome, Italy, October 21-24, 2014*, pages 154–161. SciTePress, 2014.
25. R. Saia, L. Boratto, and S. Carta. A new perspective on recommender systems: a class path information model. In *Science and Information Conference (SAI), 2015*, pages 578–585. IEEE, 2015.
26. G. Salton and C. Buckley. Term-weighting approaches in automatic text retrieval. *Inf. Process. Manage.*, 24(5):513–523, Aug. 1988.
27. G. Salton, A. Wong, and C. S. Yang. A vector space model for automatic indexing. *Commun. ACM*, 18(11):613–620, 1975.
28. G. Shani and A. Gunawardana. Evaluating recommendation systems. In F. Ricci, L. Rokach, B. Shapira, and P. B. Kantor, editors, *Recommender Systems Handbook*, pages 257–297. Springer, 2011.
29. G. Stilo and P. Velardi. Temporal semantics: Time-varying hashtag sense clustering. In *Knowledge Engineering and Knowledge Management*, volume 8876 of *Lecture Notes in Computer Science*, pages 563–578. Springer International Publishing, 2014.
30. G. Stilo and P. Velardi. Time makes sense: Event discovery in twitter using temporal similarity. In *Proceedings of the 2014 IEEE/WIC/ACM International Joint Conferences on Web Intelligence (WI) and Intelligent Agent Technologies (IAT) - Volume 02*, WI-IAT '14, pages 186–193, Washington, DC, USA, 2014. IEEE Computer Society.
31. G. Stilo and P. Velardi. Efficient temporal mining of micro-blog texts and its application to event discovery. *Data Mining and Knowledge Discovery*, 2015.
32. K. Toutanova, D. Klein, C. D. Manning, and Y. Singer. Feature-rich part-of-speech tagging with a cyclic dependency network. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology - Volume 1*, NAACL '03, pages 173–180, Stroudsburg, PA, USA, 2003. Association for Computational Linguistics.
33. E. Vargiu, A. Giuliani, and G. Armano. Improving contextual advertising by adopting collaborative filtering. *ACM Trans. Web*, 7(3):13:1–13:22, Sept. 2013.
34. C. V. Yehuda Koren, Robert M. Bell. Matrix factorization techniques for recommender systems. *IEEE Computer*, 42(8):30–37, 2009.
35. M. Zhang and N. Hurley. Avoiding monotony: improving the diversity of recommendation lists. In *RecSys*, pages 123–130. ACM, 2008.
36. C.-N. Ziegler, S. M. McNee, J. A. Konstan, and G. Lausen. Improving recommendation lists through topic diversification. In A. Ellis and T. Hagino, editors, *Proceedings of the 14th international conference on World Wide Web, WWW 2005, Chiba, Japan, May 10-14, 2005*, pages 22–32. ACM, 2005.