Integrating user modeling approaches into a framework for recommender agents

Abstract
Despite the various and different domains in which recommender agents are used and the variety of approaches they use to represent user interests and make recommendations, there is some functionality that is common to all of them, such as user model management and recommendation of interesting items. We have generalized these common behaviors into a framework that enables developers to reuse recommender agents’ main characteristics in their own developments. Our proposal provides support to the whole recommendation making process: management of different types of user models, memory-based recommendation, model-based recommendation, and social analysis of virtual communities. In addition to the different implementations provided for these components, new algorithms and user model representations can be easily added to our proposed approach. We describe three different recommender agents built by materializing the proposed framework: a movie recommender agent, a tourism recommender agent, and a Web page recommender agent. We also discuss the benefits of our proposal as well as its main limitations.

1 Introduction
Recommender agents are used to make recommendations of interesting items in a wide variety of application domains, such as Web page recommendation (Lieberman et al., 2001), music (Yoshii et al., 2008; Yapriady and Uitdenbogerd, 2005), e-commerce (Schafer et al., 2000), movie recommendation (Miller et al., 2003), tourism (Srisuwan and Srivihok, 2008), restaurant recommendation (Burke et al., 1996) among others.

User modeling approaches used by these agents differ not only because of domain-dependent characteristics, but also because of the recommendation strategy they adopt. Although there have been some attempts to abstract common behavior of recommender systems into general frameworks, these approaches fail at integrating multiple types of user models. For example, the framework CoFE1 (Collaborative Filtering Engine) (Herlocker et al., 1999), provides algorithms for collaborative filtering. However, it only supports user preferences represented by a list of rated items. The framework is not designed to support collaboration based on content-based and demographic user models. Similarly, Taste2 is a coollaborative filtering engine for Java, which takes users’ preferences for items (“tastes”) and returns estimated preferences for other items. Taste supports both memory-based and item-based recommender systems, but it does not currently support model-based recommenders. Pazzani (Pazzani, 1999) presents a theoretical framework to integrate the three types of user models rather than a software design of reusable components for recommender agents.

1http://eeecs.oregonstate.edu/iis/CoFE
2 http://taste.sourceforge.net/
In this work we present a framework for recommendation that provides the control structures, the data structures and a set of algorithms and metrics for different recommendation methods. The proposed framework acts as the base design for recommender agents or applications that want to add the already modeled and implemented capabilities to their own functionality. In contrast to other proposals, this framework is designed to enable the integration of diverse user models, such as demographic, content-based and item-based models. Thus, personal agents originally designed to assist a single user can reuse the behavior implemented in the framework to expand their recommendation strategies.

We describe three recommender agents that have been built and enhanced by reusing the functionality implemented in the framework as examples of its instantiation. Each agent uses a different recommendation approach. PersonalSearcher (Godoy and Amandi, 2000), an agent originally designed to suggest interesting Web pages to a user, was extended to collaboratively assist a group of users using content-based algorithms. MovieRecommender recommends interesting movies using an item-based approach and Traveller suggests holiday packages using demographic user models.

The article is organized as follows. Section 2 describes the different recommendations approaches we can find in the literature, mainly from the user modeling point of view. Section 3 presents our proposed framework, describing its main characteristics. Section 4 describes three recommender agents that were built using the proposed framework. Finally, Section 6 presents our conclusions and future work.

2 User modeling in recommender agents

A variety of approaches have been used by agents to perform recommendations, including content-based, collaborative, demographic, knowledge-based and others (Montaner et al., 2003; Adomavicius and Tuzhilin, 2005). To improve performance, these methods have sometimes been combined in hybrid recommenders (Yoshii et al., 2008). In spite of their common goal, these approaches differ in the form they represent user interests or preferences into user models. Figure 1 shows the integration of these user models into the framework for collaborative recommender agents.

The content-based approach is based on the intuition that each user exhibits a particular behavior under a given set of circumstances, and that this behavior is repeated under similar circumstances (Zukerman and Al-brecht, 2001). A content-based recommender learns a model of the user interests based on the features present in items the user rated as interesting either by implicit or explicit feedback. Thus, a user model contains those features which characterize a user interests, enabling agents to categorize items for recommendation based on the features they exhibit. For example, text recommendation in agents like NewsDude (Billsus and Pazzani, 1999) or Letizia (Lieberman et al., 2001) use the words appearing in documents as features.

The user models derived by content-based recommenders depend on the learning methods employed. In existing agents, user models range from: (a) a simple set of words weighted according to their importance at describing the user interests or the output format of a particular learning algorithm such as a decision tree or a probabilistic network, (b) to more sophisticated models keeping track of both long-term and short-term interests.
In contrast to the content-based approach in which the behavior of users is predicted from their past behavior, collaborative filtering (CF) is based on the intuition that people within a particular group tend to behave alike under similar circumstances. In the collaborative filtering approach the behavior of a user is predicted from the behavior of other like-minded people (Zukerman and Albrecht, 2001).

In a collaborative filtering system, there is a database of m users \( U = \{u_1, u_2, \ldots, u_m\} \) and a mapping between user-item pairs. The latter mapping is represented as an \( m \times n \) matrix \( M \). In the pure CF approach the matrix \( M \) usually represents ratings of items given either explicitly or implicitly by users, thus the entry \( m_{ui} \) represents a user \( u \) rating on item \( i \). Thus, the preferences of users are explicitly stated by the matrix \( M \) and a user model in this approach comprises a vector of item ratings, with the ratings being binary or real-valued.

The aim of collaborative filtering for the active user is to predict the score for an item which has not been rated yet by in order to recommend this item. By comparing the ratings of the active user to those of other users using some similarity measure, the system determines users who are most similar to the active one, and makes predictions or recommendations based on items that similar users have previously rated highly.

It is possible to identify two major classes of collaborative filtering, memory-based and model-based (Sarwar et al., 2001). Memory-based collaborative filtering uses nearest-neighbor algorithms that determine a set of neighboring users who have rated items similarly, and combine the neighbor’s preferences to obtain a prediction for the active user. Model-based collaborative recommenders do not use the user-item matrix directly to make recommendations, they generalize a model of user ratings using some machine learning approach and use this model to make predictions. Memory-based is the most popular prediction technique in CF applications since it is
more efficient in medium-size matrices, some examples are (Resnick et al., 1994; Shardanand and Maes, 1995; Terveen et al., 1997). However, if the user-item matrix is large the nearest neighbor computation becomes expensive. Then, model-based recommenders like (Basu et al., 1998; Zhang and Iyengar, 2002; Heckerman et al., 2000; Lee, 2001; Lin et al., 2002) are a suitable alternative.

Demographic recommenders aim at categorizing users based on their personal attributes as belonging to stereotypical classes. Instead of applying learning techniques for acquiring user models, these agents are based on stereotype reasoning (Kobsa et al., 2001). In this case, a user model is a list of demographic features which represent a class of users. This representation of demographic information in a user model can vary greatly. For example, Pazzani et al. (Pazzani, 1999) extract features from home pages to predict the preferences for certain restaurants, and Krulwich et al. (Krulwich, 1997) use demographic groups from marketing research to suggest a range of products and services.

In knowledge-based approaches, recommendation is based on inferences about a user needs and preferences which are performed using some functional knowledge, that is, there is knowledge about how a particular item meets a particular user need, and can therefore reason about the relationship between a need and a possible recommendation (Burke, 2002). The user models in knowledge-based recommenders can also take many forms, since they can consist of any knowledge structure that supports inference. The restaurant recommender Entree (Burke et al., 1996) makes recommendations by finding restaurants in a new city similar to restaurants the user knows and likes based on the knowledge of cuisines to infer similarity between restaurants. Ontology-based user profiling is also an example of knowledge-based recommendation. For example, Quickstep (Middleton et al., 2004) is a recommender system addressing the problem of recommending on-line research papers to researchers, which bases user interest models on an ontology of research paper topics.

In spite of these different types of user models, recommender agents still share similar behavior as regards collaboration. Most of these agents have to compare user models to find similar users for exchanging information about potential interesting items and, at the same time, they can be benefited from the knowledge that can be extracted from the implicit behavior of the whole community. The following section presents our proposal to abstract and model into a framework this common functionality.

3 Abstracting the common behavior of recommender agents

If we analyze different recommender agents, we can observe that despite the domain-dependent characteristics and user modeling approaches, these agents behave quite similarly. Then, the characteristics and behaviors that are common to most recommender agents can be abstracted to build a framework to facilitate the implementation of this kind of systems. A software framework is a reusable design for a software system (or subsystem) that is expressed as a set of abstract classes and the way the framework components collaborate for a specific type of software (Johnson and Foote, 1988).

In this paper we present a framework for recommendation that abstracts a number of content-based, collaborative and social filtering methods commonly used by recommender agents. This design acts as the skeleton for recommender agents or applications that want to add the modeled behavior to their own functionality. For example, a personal agent originally designed to assist a
single user can reuse the behavior implemented in the framework to start acting collaboratively in a community.

![Layered view of the framework and component interactions](image)

Figure 2: Layered view of the framework and component interactions

Figure 2 depicts a general view of this framework and its main components. Our proposed framework provides the control structure, the data structures, and a set of algorithms and metrics for different recommendation methods. The functionality provided by the framework is basically the following:

- Management of User Models (layer 1)
- Memory-Based Recommendation (layer 2)
- Model-Based Recommendation (layer 3)
- Social Analysis of virtual communities (layer 4)
- Recommendation Engine

The framework is implemented in Java, using a client-server architecture. We can see it as a service-provider, that is a server that provides recommender system capabilities to various remote clients, namely recommender agents. The client application or agent communicates with the server part of the framework to, for example, send user ratings and receive collaborative recommendations. The algorithms, techniques and data structures used to make recommendations are on the server side. The user model is on the client side, since it is usually application dependent, and it is sent to the recommendation engine (previous some transformation if it is necessary) on the server side so that it can be used to make suggestions.

The framework allows the creation of recommender agents from scratch as well as the integration of more complex recommender agents with the purpose of enriching their functionality. In this direction, as shown in layer 1 in Figure 2, the framework supports a number
of standard user models that can be used to create agents without further implementation efforts. More specific, domain-dependent user models can be added by specializing the supported models and providing a means to assess their similarity with both items to be recommended as well as other models. In Section 4 the integration of different types of user models is exemplified by three instantiations of the framework.

The comparison of user models enables agents to add collaborative recommendation by finding a set of users that have similar characteristics or have a history of agreeing with the active user (that is, they rate items similarly). Multiple algorithms and metrics are implemented in the framework for establishing the neighborhood of users and combine the preferences of neighbors for prediction. Thus, by only specifying the mechanism of comparison of user models, already developed personal agents can take advantage of collaborative recommendations using some memory-based algorithm. This functionality is provided by layer 2.

In the next framework layer we can find the model-based collaborative filtering algorithms, which provide item recommendation by first extracting a model of users. The model inference is performed by machine learning algorithms such as clustering, Bayesian networks, or rule-based approaches. The algorithms in this layer can be used in combination with the mentioned memory-based algorithms. For example, user clustering can be used to narrow the search of neighbors in a collaborative algorithm.

The recommendation engine is in charge of dealing with the recommendations generated by using the different recommendation approaches or a combination of them (e.g. content-based and collaborative recommendation). This engine enables the development of agents that pro-actively recommend users interesting information, generate recommendations under demand, or both. In addition, the recommendation engine collects the feedback from users, which is used to update the user models, and records the activity of users in the system.

As shown in layer 4 in the figure, the knowledge about the activities of users registered by the recommendation engine can be analyzed from a social point of view. Thus, in the last layer of the framework it is possible to find algorithms and techniques for social data analysis such as those included in the Social Data Mining (Amento et al., 2003) and Social Network Analysis (Sabater and Sierra, 2002) areas. Furthermore, the data about user activities serve as a source for generating diverse visualizations to explore and interpret the behavior of the community.

Each component of the proposed framework is detailed in the following subsections.

### 3.1 Management of User Models

As we have said before, a user model is a representation of a user interests, habits and preferences in a given domain. The representation formalism of a user model varies from one application to another. Our approach provides a number of standard representations for user models that tries to capture the approaches most widely used by recommender agents.

We consider three main categories of user models within recommender agents: content-based user models, item-based (or collaborative) user models, and demographic user models. Each type of user model has its own representation and requires a different method to compare it against other models or against items to recommend. The following sections describe the representations modeled in the framework.
3.1.1 Content-based user models

Content-based user models are built from the observation of the interaction of a user with an underlying application. Depending on the domain, different representations for the user model can be found. In our framework we consider three main representation formalisms for this kind of user model: a feature vector; a classifier denoting the relation between a set of features and a set of classes or categories; and a hierarchy of classifiers. In addition, new representations for user models can be easily added.

One of the most popular representations of items is describing them through their main characteristics. This representation is known as feature vector. For example, a scientific paper can be described by the authors, an abstract, the publication date, the journal or conference where it was published, a set of keywords, among others. In the case of Web pages or text documents, they are represented as a set of relevant words, each having a frequency value. The user model is then a vector of relevant words representing the user interests.

In some domains, the items are classified or categorized according to the attributes or features describing them by using a classifier inferred from a set of examples. Thus, our approach also provides the representation for those user models in which the user models holds the structure of a classifier that categorizes examples in a set of classes (e.g. a decision tree). In turn, the classifiers may form a hierarchy to distinguish hierarchical classes and categories. For example, the topics of interest of a user may be organized into a hierarchy where different levels of abstraction in the user preferences can be modeled.

3.1.2 Item-based user models

The idea underlying collaborative filtering is giving recommendations of items that were interesting to other users that are similar to the user the agent is assisting. The goal is obtaining the utility or rate a user would give for an item given information of the ratings provided by similar users.

Thus, the user models in collaborative filtering do not model the contents of the items a user is interested in, namely documents, movies, or books, but the evaluation or rating the user has assigned to these items. Thus, such a user model is composed of a set of name-value pairs in which the name represents an item under consideration and the value a rating provided for the item.

3.1.3 Demographic user models

Demographic data about users can be also used to make them recommendations of potentially interesting items. Demographic information may include attributes such as a sex, age, city, nationality, job, hobbies, among other features that may be relevant to the application domain. A demographic user model is generally obtained from the information explicitly given by the user through a user interface provided for that purpose.

Figure 3 shows the different user models proposed by our approach. A recommender agent that wants to define its own user model should implement a class inheriting from one of the classes shown in Figure 3 (HierarchicalUM for example). Similarly, new algorithms to build the content-based user models can be defined. Our framework provides a set of well-known Machine Learning (decision trees, naïve Bayes, etc.) and Information Retrieval (Rochio, tf-idf, etc.) algorithms that agent developers can use.
3.2 Memory-Based Recommendation

In order to make collaborative recommendations, a subset of users out of the whole population have to be chosen based on their similarity with the active user and a weighted combination of their ratings is used to generate predictions. Neighborhood-based or user-based collaborative filtering is performed in three steps: weighting all users according to their similarity with the active user, selecting a subset of these users and computing predictions based on the ratings given by the group of users. For each of these steps different algorithms and techniques are implemented in the framework.

In the first step, memory-based algorithms utilize either a metric of comparison for content-based user models or the entire user-item matrix to estimate a neighborhood of users which resemble the active one. This process results in a user-user matrix of similarities in which a row represents a user and columns hold the distance/similarity with the remaining users.

In the first case, content-based models represented as feature vectors are compared by comparing the values for the different attributes representing each user model. For vectors of normalized numerical attributes several common distance/similarity functions are provided by the framework, including the Euclidean distance, Manhattan distance, and cosine similarity. Thus, an agent representing the interest of a user by a single vector of keywords, such as Letizia, can be straightforwardly integrated in the framework by using the cosine similarity to compare user models and gain the ability of recommending the information discovered by other users. We can observe these components in Figure 4. More specific methods can be defined to compare more complex, specialized user models. For instance, demographic user models can be compared to another demographic user model by using the similarity functions used for feature vectors or defining a similarity measure, possibly combining numerical and nominal attributes weighted according to their importance.

In the second case, that is in item-based models, neighbors are identified by comparing the ratings of all users with the ratings given by the active user to items. The most common metrics are implemented in the framework for this purpose, including the mean squared difference, the Pearson correlation coefficient and the Spearman rank correlation. Also, significance...
weighting (Herlocker et al., 1999) is implemented to add a certain level of trust to neighbor correlations. Further correlations measures can be defined by extending the framework in this point.

The information available in the user-user similarity matrix allows the selection of the most alike users to use their opinions for prediction. The selection of neighbors can be achieved by using correlation-thresholding (Shardanand and Maes, 1995), which selects all users whose correlation is above a certain absolute threshold, and best-n-neighbors, which select the best \( n \) correlates for a given \( n \) (Herlocker et al., 1999).

Once a neighborhood of users is formed, different algorithms can be used to combine the preferences of neighbors to produce a prediction or top-\( N \) recommendations for the active user. The provided methods to combine the ratings of the neighbor users are the weighted average of the ratings used in Ringo (Shardanand and Maes, 1995), which uses the correlations as weights, the deviation-from-mean approach of GroupLens (Resnick et al., 1994) and the regression technique, which uses an approximation of the ratings based on regression model instead of the real ratings.

### 3.3 Model-Based Recommendation

Model-based recommendation consists of extracting a model from user preferences and using this model for prediction. The model building process can be performed by different machine learning algorithms. Clustering approaches work by identifying groups of users who appear to have similar preferences or ratings. Then, predictions for a candidate user are made by averaging the opinions of the other users in the cluster the user belongs to.

Our proposed framework provides different clustering algorithms that recommender agents can use to group similar users, including \( k \)-Means, PAM (Kaufman and Rousseeuw, 1987), and agglomerative hierarchical algorithm (HAC). However, agent developers have to be aware of the fact that not all algorithms can be applied to every type of user model. For example, the \( k \)-Means...
algorithm cannot be directly applied to content-based user models, since it considers vectors of numerical attributes. For this kind of models a means to compute an average value of user models has to be provided by the agent developers in order to calculate the cluster centroids. A variation of the $k$-Means for nominal data is the $k$-Modes algorithm which can be used for demographic user models. Figure 4 shows the class hierarchy for this part of the framework. Other clustering algorithms as well as specializations of the existing ones can be added by extending this group of classes.

In addition to clustering algorithms, a rule-based approach is included in the framework for model extraction. Usually, association rules discovery algorithms are used in e-commerce recommender systems to find associations between co-purchased items. The system then generates item recommendations based on the strength of the association between items. *Apriori* algorithm (Agrawal and Srikant, 1994), one of the most popular algorithms for mining of association rules, is included in the framework. Thus, a recommendation model based on association rules corresponds to the set of association rules generated from the user preferences.

The model-based algorithms can be used in combination with memory-based algorithms. For example, the results of user clustering can reduce the number of users against whom the agent compares the active user for neighborhood formation. Enhancing the accuracy of finding neighbor users can result in a more effective collaborative filtering.

### 3.4 Recommendation Engine

Using the framework agents can recommend items that are potentially interesting for their users and, in turn, to receive suggestions made by other users. In addition, they can request a prediction of the expected interest value for a specific item. Users can also provide feedback for the suggestions and recommendations they receive. In order to support these tasks, our framework provides storage capabilities in persistent media for the following components: user models, the recommendations made by users, user similarity structures, and statistic data about users and their interactions.

The main functionality of the recommendation engine can be divided into two main tasks: (a) the construction and administration of a set of data structures such as the user-user matrix, the database of candidate recommendations (e.g. a set of Web pages gathered by an agent from diverse sources), and (b) the usage of these data structures to make useful recommendations to users. The algorithms specified in other layers of the framework are employed by the recommendation engine in both tasks.

To build the data structures, the engine receives the user models sent by client applications, namely recommender agents. These user models are the input for memory-based and model-based algorithms for neighborhood formation or/and model extraction according to a predefined recommendation strategy. Then, the recommendation engine is in charge of exchanging recommendations produced by these collaborative approaches. If an item is discovered by a user and it is found potentially relevant for other user, it is sent to the corresponding agent so that it can display it.

In addition, our framework provides the means to handle user feedback, which can be provided by users for the suggested items. This information is recorded and it is used to compute the reputation values for the users that made the suggestions as well as to adapt the corresponding user models. Other uses of the received feedback can be specified by agent developers.
Finally, the recommendation engine generates the logs of the social activity. In the logs it is registered the user that discovered a certain item, the user that received this item as a recommendation and the pattern of feedback received. The resulting logs are used in the last layer of the framework for social analysis.

### 3.5 Social analysis of virtual communities

An important feature of the proposed framework is its support to virtual communities, that is, the provision of a set of mechanisms to enable the explicit exchange of tacit and explicit knowledge between members of the community. Users can interact by exchanging ratings and recommendation of items. These interactions can be used to discover different types of relationships between these users. In order to analyze these relationships, we need methods to extract useful information from the logs of social activities and techniques to visualize this information. The following sections describe the visualization methods that the framework provides. Further visualizations can be defined in this layer for exploring the available information about the community and the emerged social network. Although not specified in this framework, also Social Data Mining techniques can be used in this layer to extract models of social interaction.

#### 3.5.1 Visualizing user associations

This visualization make it possible to observe which community members exchange (now) or have exchanged (in the past) recommendations between each other and the volume of the information exchanged. In this way, we can study the formation of the relationships between different members of a community and its evolution over time. Figure 5(a) shows a set of users that are related due to their exchange of information. Each edge connecting two nodes indicates that the users represented by these nodes have exchanged a recommendation and the different colors of the edges denotes the strength of the association.

In addition, we can obtain information about each particular user, such as the trust or reputation value for each of the users he has interacted with, and the average trust or reputation value the other users have assigned to him. The reputation value is calculated by considering the feedback provided by the users to the recommendations he has received. Figure 5(b) shows the information recorded in the system for a user named Nicolás. Other statistic information such as the number of suggestions received from and submitted to other users is also available.

#### 3.5.2 Visualization of user similarity

This visualization enables us to view the different user grouped in clusters according to their similarity, as shown in Figure 6. Each cluster groups a set of users which user models are alike. The distance between the center of the cluster and the other users represents the distance between each pair of users. An ego-centric view of the cluster can be obtain by choosing a given user. Then, this user is placed in the center of the scene surrounded by the other users in the cluster it belong to. Also, the reputation and similarity values of the remaining users in the cluster are displayed.
Figure 5: Visualization of user information within a community

4 Materializations of the proposed approach

The following subsections describe three examples of recommender agents in different domains: movie recommendation, Web pages recommendation, and holiday packages recommendation. These agents use the functionality provided by our framework by acting as its clients. That is, the agents send the user models to the recommendation engine and they get the recommendations for the different users. In some cases, extensions to the original functionality provided by the framework were implemented.

The materializations presented show how some agents can be implemented without much effort, by simply reusing the components provided by the framework. This is the case of both the MovieRecommender and the Traveller agents. In turn, agents based on more sophisticated user models can not only be easily integrated with the framework, but also be enriched with collaborative capabilities.
4.1 Example of a collaborative-based recommender

*MovieRecommender* is an agent that assists users of a movie Web site by recommending them a set of movies that the agent believes the user would enjoy, according to the user’s preferences. When the user registers to the Web site, he informs his preferences regarding movies, that is, genres, actors, actresses, directors the user likes most. In its original version, the agent made recommendations to users considering only the user’s explicit preferences and the feedback the user gave to suggested movies. To enhance the agent’s capabilities with collaborative-based recommendations, we used the functionality provided by our framework.

To make collaborative recommendations, the user model must contain the movies the user rated with the corresponding ratings. Thus, we had to transform the original user model composed of the user’s explicit preferences and the feedback provided for recommended movies into an item-based model, as required by the framework. Thus, the feedback provided for the recommendations made by the agent were considered as ratings for the suggested movies. Figure 7 shows this transformation.

Assuming that a user would recommend those movies he has enjoyed, each time a user rates a movie as "very good" (rating of 5) or "excellent" (rating of 6), the agent considers this movie, so that it can recommend it to users having a model similar to the user who rated the movie. We had to add this functionality to *MovieRecommender*, that is, the ability of sending a potential recommendation to the recommendation engine.
In order to allow a user to view the recommendations made by other users, these recommendations must be available when the user enters into Web site. To provide this functionality, MovieRecommender sends a message to the recommendation engine, the engine sends the recommendations for this user (those made by similar users), and the agent presents them to the user. Figure 8 shows a screenshot where the agent gives a set of collaborative recommendations for the active user.

When a user gives his opinion about a movie that was suggested by another user, this feedback is used both as a potential suggestion for other users (if the feedback is highly positive) and as a rating for the user who made the suggestion (reputation). This information is used by the framework to determine future recommendations. Figure 8 shows a snapshot of MovieRecommender in which the agent requests the user to rate a movie.
Summarizing, to add collaborative-based recommendations to MovieRecommender using the proposed framework we had to: adapt the user model; enable the connection with the recommendation engine to send user ratings and to retrieve collaborative recommendations; implement a new user-interface to show the collaborative recommendations.

We evaluated the precision of the recommendations made by MovieRecommender through different experiments. For these experiments, we used the Movielens dataset. In the first experiment, we measured the precision of the collaborative recommendations made by the agent for various algorithms and different numbers of clusters for these algorithms. In the second experiment, we measured the precision of the collaborative recommendations for different similarity thresholds.

Figure 9 shows the results obtained for the first experiment. We can observe the percentage of correct recommendations made when the agent used the following clustering algorithms: k-Means, Max Min, PAM, and One cluster algorithm. For each algorithm, the graph plots the precision obtained with 3 and 4 clusters. In these experiments, the similarity threshold was set to 0.55.

Figure 10 shows the precision of MovieRecommender for different similarity thresholds. The algorithm selected was k-Means and the number of clusters was set to three. We can observe in the figure that as the similarity threshold increases, so it does the precision of the recommendations.

---

3http://movielens.umn.edu
4.2 Example of a demographic-based recommender

Recommender agents that make recommendations to users about travel and tourism can use a demographic user model in addition to the content-based and collaborative models. In this domain, the recommender agent *Traveller* suggests the user a holiday package. A holiday package is composed of a set of tours the user is supposed to like. Other characteristics of the package are the name, the modality of the trip (e.g. honeymoon), the departure date, the price per person, and the duration (in number of nights and/or number of days). The characteristics of a tour are: category, place or destination, price, means of transport, transport company (e.g. name of the airline), accommodation (e.g. name of the hotel), type of accommodation (hostel, 5 star hotel, 4 star hotel, etc.), room type, duration, meals included.

Initially *Traveller* used to suggest tours according to the user’s preferences in a content-based fashion, that is, according to tours the user had taken in the past for which he provided a positive feedback. Attributes such as destination (city, province, country), characteristics of the destination (e.g. beach, tropical climate, outdoor activities), means of transport, type of accommodation, cost, and duration, were used to suggest tours and holiday packages.

Then, we decided to enhance this functionality by recommending tours according to a demographic model. As an example of demographic-based recommendation, travel agencies often offer tours for retired people. In this situation, the agent cannot recommend these tours to teenagers or big families, for example. Similarly, travel agencies try to place in the same tours couples travelling as honeymooners and young couples travelling without their kids. In this approach, the recommender agent compares the characteristics of the tour against the demographic user model. To add this functionality to *Traveller*, we used the demographic user model provided by the framework and we added some new attributes to it, which we considered were useful for the tourism domain. Figure 11 shows an example of a demographic user model for *Traveller*. This model comprises: age, sex, marital state, number of children, city, and the type of people the user generally travels with.
To compare the user model against a potentially interesting tour, the agent compares each attribute in the user model against the corresponding attribute in the tour. Different similarity functions may be available for different categories of tours. The differences are given as weights assigned to the attributes in the calculus of the similitude. Different metrics were implemented as extensions to the framework.

Figure 12 shows a set of recommendations proposed by Traveller for a set of users. The recommendation comprises a user, the name of the holiday package and an estimated value of interest for the user. As shown in the figure, the agent also prepares an email to send the recommendation to the user. These recommendations were obtained using a hybrid approach, that is, combining the content-based and the demographic user model.
4.3 Example of content-based recommendation

In this section we introduce an example of a user modeling approach that was initially designed for content-based recommendation and later extended for collaborative recommendation using the framework functionality.

4.3.1 Representation of User Models

PersonalSearcher (Godoy and Amandi, 2000; Godoy et al., 2004) is an interface agent that helps users who are searching the Web for relevant information by filtering a set of documents retrieved from several search engines according to users’ interests. To acquire information about users’ interests, PersonalSearcher observes users’ behavior while they are reading Web documents, recording the main features characterizing these experiences. Based on these experiences, a PersonalSearcher agent builds a representation of the user interests or user model.

For personal information agents, a user model has to represent user information preferences in order to satisfy long-term information needs. Most users have diverse information interests regarding, for example, their work (e.g. computer software), their hobbies (e.g. sports) or current issues (e.g. presidential debates); which should be modeled into separate categories. In addition to modeling multiple interests in several domains, user models also require to represent the different abstraction levels of such interests.

PersonalSearcher obtains a hierarchical representation of user interests across different abstraction levels. At the top level of the hierarchy the most general topics are placed, those representing broader interests, while lower in the hierarchy we can find more specific interests, those referring to particular aspects of general topics. Figure 13 shows an example of a user interest hierarchy representing the user model of a hypothetical user. This user is interested in two general categories, sports and politics, and more specific categories such as basketball, football, political science and others. In each of these categories, the user has even more specific interests such as some basketball team or player.

Hierarchical views of user interests not only enhance the semantic of user models as it is much closer to the human conception of a set of interests, but also enable agents to have a temporal view of such interests. Even though user interests are expected to change over time, users frequently show a certain persistence in certain interests, so that user models have to gradually converge to the part of user needs that is predictable and consistent over time. In a hierarchical organization, interests at the top levels of the hierarchy can be seen as long-term interests, while more specific ones can be seen as shorter-term interests.

![User Interest Hierarchy Diagram]

User Interest Hierarchy

User Interest Categories

User Interest Experiences
Figure 13: Example of a hierarchical organization of interests

PersonalSearcher learns a user interest hierarchy based on observing experiences of the user interests. In an information domain these experiences are documents considered relevant to the user, such as Web pages the user browsed or a piece of news the user read in an on-line newspaper, which agents can capture by implementing explicit and/or implicit feedback mechanisms. The former mechanism consists in asking the user for explicit judgments about documents in an ordinal scale; whereas the later calculates an estimated user interest in a given document by observing a number of implicit interest indicators (such as the time spent reading, the amount of scrolling, etc.).

In addition to identifying categories to track the user interest in each of them, the agent characterizes categories in order to assess comprehensible user models. Individual user interests in a hierarchy are characterized by a set of descriptive terms. For example, the category sports in the previous figure can be described by the terms sport, team and cup; whereas sub-categories can define themselves by more specific terms, for example basketball, NBA and league.

In order to describe user interests, hierarchical models are built using a clustering algorithm, named WebDCC (Web Document Conceptual Clustering) (Godoy and Amandi, 2006), with structures and procedures specifically designed for user profiling. The unsupervised learning process provided by this clustering algorithm offers to the user modeling task the advantage that a priori knowledge of categories is not needed. The categories a user is interested in are incrementally discovered and characterized as agents interact with their users over time.

The advantage of using an algorithm belonging to the conceptual clustering paradigm is twofold. First, it is an incremental approach which allows agents interacting with users over time to acquire and maintain user interest hierarchies as well as to deal with unpredictable subject areas. Second, unlike most user profiling approaches, using this algorithm agents can gather a readable description of the user interests as a means of explaining agent actions, verifying the model correctness and communicating with other agents at a conceptual level.

Initially, PersonalSearcher was designed to run as a stand-alone application learning the interests of individual users. However, agents assisting several users can take advantage of existing knowledge in the community they are immersed in through cooperation. A cooperative approach fosters knowledge sharing and, consequently, potentially enriches the achievable results of individual agents by accessing other agent experience. For a given user model, a set of similar models have to be found by comparing the information contained in them about the interests of multiple users. Then, information items that were interesting for one user can be recommended to those users having similar models.

In order to take advantage of the functionality provided by the framework, the hierarchical representations of user interests modeled in the framework within the content-based user model was used. More importantly, it was needed to define a method to compare two user models in order to enable collaborative recommendation. In the previously described agents, user models were quite standard and, therefore, we were able to provide the framework with different measures to compare these user models. In contrast, user models in PersonalSearcher are specifically designed for this agent and, consequently, it was also necessary to define a method to compare this kind of models as it is explained in the following section.
4.3.2 Comparison of User Models

A first aspect to consider when extending PersonalSearcher for collaborative behavior is how two agents determine whether they know the same concepts. If both agents agree on the meaning of two different semantic concepts via the comparison of the corresponding interest hierarchies, these concepts translate to each other in the future for information exchange. By locating similar semantic concepts existing in distinctive user models, agents are able to overcome their lack of shared meaning and gain the ability of exchanging knowledge.

The Triple Matching-Model (MD3) (Rodríguez, 2000; Rodríguez and Egenhofer, 2003) was adopted to compare conceptual hierarchies built by PersonalSearcher agents (Giménez-Lugo et al., 2002). This model allows the evaluation the similarity among concepts across multiple ontologies or, in this case, conceptual views of the world externalized by user models.

MD3 aims at finding quantitative values of similarity among concepts across two user models by comparing concept descriptors as well as concept interrelationships. The similarity relationships make it possible to establish anchors between models while keeping each model autonomous. This is considered a weak form of integration of ontologies, because it cannot be used for making inferences about relationships of other concepts in the ontology, but it is particularly useful in dynamic user modeling since it provides a systematic way to detect which set of terms are the most similar and, therefore, which set of terms are the best candidates for establishing a link across the user models.

Figure 14 shows the part of the framework that was extended to redefine the method of comparison of user models.
4.3.3 Content-based and Collaborative Recommendation

Once extended the framework to support PersonalSearcher content-based user models and to compare them, collaborative recommendations can be obtained in addition to content-based ones. In this section, we first explain how the agent recommend Web pages based on the user model, the initial method of recommendation, and how Web pages are recommended using a collaboration approach provided by the framework. Both content-based and collaborative recommendations in PersonalSearcher are depicted in Figure 15.

Incoming Web pages in PersonalSearcher are analyzed by computing their relevance degree regarding to the user interest hierarchy in order to determine the convenience of presenting them to the user. Those pages that exceed a user relevance threshold as regards to the user model are sent back to the user as a result of the query. PersonalSearcher allows users to customize the desired level of assistance at any moment by adjusting the relevance pages have to exhibit in order to be recommended.

In order to evaluate individual recommendations, the algorithm should be able to find the most relevant experiences in the user model. To accomplish this goal, the category the Web page to be recommended belongs to has to be first established by classifying the page in the hierarchy of user interests. Then, the recommendation algorithm evaluates the confidence in recommending the Web page based on the similarity of the page to the past experiences belonging to the category is was classified into and the relevance of these experiences.

To sum up, the agent retrieves similar experiences in the user model to obtain a set of the best matching past experiences for evaluating a recommendation. Hierarchical categories in the user model bias the search toward the most relevant experiences. Once the global similarities between the new Web page and the experiences in the category the page belong to are calculated, the confidence in the recommendation is calculated.

The functionality implemented in the framework allows agents to provide collaborative recommendations using different algorithms and, at the same time, to provide a means to compare
and evaluate different recommendation strategies. In order to illustrate this new functionality in PersonalSearcher we show the experimental results of comparing collaboration approaches for a number of users.

Experiments were run using a Web page collection named Syskill&Webert. This collection consists of 332 Web pages belonging to four categories: Bands (recording artists), Goats, Sheep and BioMedical. Each domain contains from 50 to 100 pages. We learned the user models of five users using overlapping subsets of this collection, so that user models can show some shared interests among the users.

Figure Error: Reference source not found shows the precision of recommendations that PersonalSearcher was able to deliver to the five users using different clustering algorithms. In the figure, the results are shown considering 3 and 4 clusters of users. Others parameters that can also be varied for experimentation in the framework are the similarity and correlation measures among users. Finally, the framework allows also to explore the relationships among users. It can be observed in Figure 17 shows the interactions among users, this is the exchange of recommendations (some members of a cluster may not interact with others), when users were clustered using different number of clusters.

![Figure 16: Precision of recommendations in PersonalSearcher for different algorithms](image)

5 Discussion

The three recommenders presented above demonstrate not only the utility of the framework to develop simple collaborative filtering agents like the MovieRecommender and more advanced recommender agents like PersonalSearcher involving complex user models, but also the capacity of abstracting their different user modeling mechanisms to enable the analysis of agent communities. Regardless the representation of user models and the method used to assess their similarity, the layered structure of the framework allows interest groups and interaction among individual agents to be detected and analyzed through the implementation of different social networks as well as social data mining techniques.

http://kdd.ics.uci.edu/databases/syskill/webert/
Other related frameworks are CoFE, Taste and Duine\textsuperscript{5}. CoFE engine analyzes ratings, determines neighborhoods and compute recommendations. It implements a high-performance nearest-neighbor algorithm with similarities computed by Pearson correlation, although other algorithms can be easily integrated. Taste is a collaborative filtering engine for Java that implements several variations of user-based and item-based collaborative filtering. This framework also provides some model-based algorithms such as applications of clustering and Bayesian networks for recommendation. Duine is a collection of software libraries that allows developers to create prediction engines for their own applications. It implements different collaborative filtering (such as user-based similarity) as well as content-based (such as case-based reasoning) techniques for predicting the interest of items.

Our framework models collaborative filtering and content-based approaches in the first and second layers and then model-based approaches in the third, covering all of the methods separately offered in these three different frameworks. In addition, we added a fourth layer that simplifies the implementation of algorithms for analyzing the social behavior of recommenders independently of the user model employed.

\textsuperscript{5} http://www.duineframework.org/
6 Conclusions

We have presented a framework for recommender agents that abstracts the behavior of different user modeling and collaboration approaches commonly used for recommendation. We would like to highlight two contributions of the proposed framework. On the one hand, the functionality provided by the framework enables the development of recommender agents without the need of implementing its whole set of capabilities from scratch. The main processes and data structures of recommender agents are already implemented. On the other hand, already existing agents can be enhanced by incorporating the functionality provided by the recommendation framework in order to act collaboratively.

We have also presented three successful experiences in materializing the proposed approach to build agents that take advantage of the functionality provided by the framework. These agents make recommendations in different application domains and use different types of user models. Our framework has some limitations. As regards social analysis, as the number of users increases the performance of the clustering algorithms used to build the networks decreases, given the techniques used to compare the similarity between users. With respect to user models, some developers might find it difficult to adapt profiles built using techniques such as decision trees or association rules to the structures provided by our framework. Developers should extend the framework to enable the incorporation of these types of profiles and also carefully analyze whether they should use the similarity metrics provided or they should define their own.

As a future work, we are planning to enhance the proposed framework by adding a layer that will enable agents to unify heterogeneous user models. For example, an agent making recommendations of movies can use the information contained in the user model built by a book recommendation agent. In this case, the interests about the type of books a user reads can be used to suggest him and/or similar users potentially interesting movies. We are also considering extending the framework to support group recommendation approaches (Chen et al., 2008; McCarthy et al., 2006). To achieve this, the framework will provide different aggregation techniques such as merging recommendations made for individuals; aggregation of individuals’ ratings for particular items; construction of group preference models; maximizing average satisfaction; minimizing misery; among others.

References


