



IST PROJECT 35111

Tightening knowledge sharing in distributed software communities by applying semantic technologies

Project Number:	35111
Project Acronym:	TEAM
Project Title:	Tightening knowledge sharing in distributed software communities by applying semantic technologies
Instrument:	STREP
Thematic Priority:	Information Society Technologies (IST)
Start date of the project:	September 1 st , 2006
Duration:	30 months

D15: Report describing state-of-the art in Semantic Recommendation

Lead contractor:	FZI
Editor(s):	Hans-Jörg Happel
Author(s):	Ljiljana Stojanovic, Hans-Jörg Happel
Submission date:	October 2007
Dissemination level:	Public

Abstract: This report describes the state of research and practice in the topic of pro-active knowledge delivery. It focuses on the topic of recommender systems, their extension with ontologies and their application in the domain of Software Engineering.

Versioning and Contribution History

Version	Date	Modification reason	Modified by
0.1	02/07/07	Document structure and initial content	Hans-Jörg Happel
0.2	17/07/07	First version of Chapter 4	Hans-Jörg Happel
0.3	22/08/07	Update on Chapter 4	Hans-Jörg Happel
0.5	17/09/07	Chapters 1+2	Ljiljana Stojanovic
0.9	25/09/07	Chapter 3	Ljiljana Stojanovic
1.0	28/09/07	Integration and Summary	Hans-Jörg Happel
1.1	04/10/07	Minor corrections and finalization	Hans-Jörg Happel

This document has been produced in the context of the TEAM Project. The TEAM project is part of the European Community's Sixth Framework Program for research and development and is as such funded by the European Commission. All information in this document is provided "as is" and no guarantee or warranty is given that the information is fit for any particular purpose. The user thereof uses the information at its sole risk and liability. For the avoidance of all doubts, the European Commission has no liability in respect of this document, which is merely representing the authors view.

Table of Contents

EXECUTIVE SUMMARY	6
1 INTRODUCTION.....	7
2 FOUNDATIONS OF RECOMMENDER SYSTEMS.....	9
2.1 HISTORY AND DEFINITION	9
2.2 CLASSIFICATION/TYPES	10
2.2.1 <i>RATING-BASED RECOMMENDATION</i>	11
2.2.2 <i>PREFERENCE-BASED FILTERING</i>	16
2.2.3 <i>IMPLICIT FEEDBACK</i>	16
2.3 RESEARCH ISSUES	20
2.3.1 <i>COMPREHENSIVE UNDERSTANDING OF USERS AND ITEMS</i>	21
2.3.2 <i>MULTIDIMENSIONALITY OF RECOMMENDATIONS</i>	21
2.3.3 <i>MULTICRITERIA RATINGS</i>	22
2.3.4 <i>NONINTRUSIVENESS</i>	22
2.3.5 <i>FLEXIBILITY</i>	22
2.3.6 <i>EFFECTIVENESS OF RECOMMENDATIONS</i>	23
3 ONTOLOGY-BASED RECOMMENDER SYSTEMS	24
3.1 THE ROLE OF ONTOLOGIES	24
3.1.1 <i>ONTOLOGY-BASED RETRIEVAL</i>	24
3.1.2 <i>PRESENTATION OF RESULTS</i>	26
3.1.3 <i>IMPLICIT FEEDBACK</i>	29
3.2 EXAMPLE SYSTEMS	31
3.2.1 <i>ONTOLOGY-BASED PROFILING</i>	31
3.2.2 <i>DECENTRALIZED RECOMMENDER</i>	34
4 RECOMMENDER SYSTEMS IN SOFTWARE ENGINEERING.....	36
4.1 CODEBROKER.....	36
4.2 DHRUV.....	36
4.3 HIPIKAT	37
4.4 MYLYN	37
4.5 RASCAL.....	38
4.6 STRATHCONA.....	38
4.7 SUMMARY.....	39
5 CONCLUSION	41
REFERENCES.....	42

List of Figures

Figure 1: The general structure of the information push process	7
Figure 2: Challenges in the information push process.....	8
Figure 3: Three classification types of recommender systems and their role regarding information push process	10
Figure 4: Rating estimation strategy	20
Figure 5: Predicted observations strategy	20
Figure 6: The Quickstep system	32
Figure 7: Section of the Quickstep research paper ontology	35

List of Tables

Table 1: Observable behaviour for implicit feedback.....	18
Table 2: The increase in retrieval's precision by introducing more structure in the representation mechanism.....	26
Table 3: Overview of Software Engineering recommender systems.....	39

EXECUTIVE SUMMARY

The TEAM system aims to provide a distributed knowledge sharing toolkit to software developers. By using P2P technology, the TEAM systems of individual users will allow to access large amounts of data from other peers. However, accessing and evaluating this data is a costly process, especially in a dynamic environment with new peers coming and going. Thus, automated support for exploring the available peers for relevant information is a key feature to support users. In recent years, the area of recommender systems has explored these kinds of issues.

Recommender systems are based on the general idea of recommending items (which can be arbitrary entities such as products, movies, documents or source code) to users, which they might find useful in their given context. Typically, recommendations in these systems involve suggesting items selected by users with a similar profile (collaborative filtering) or items which are similar to other items the current user selected in the past (context-based filtering). User preferences are either collected explicitly (e.g. with ratings) or implicitly by any kind of user observation.

Recently, ontology-based representations for user profiles and item descriptions have been proposed to compensate for the common problems of the limited availability of meaningful user preferences and ratings. Also, a number of recommender systems have been proposed for Software Engineering applications. These systems either focus on suggesting relevant artifacts for solving software maintenance tasks, or on predicting source code methods for reuse.

1 INTRODUCTION

Due to an explosion in the content production in knowledge-intensive processes (like software development), the traditional access to information (i.e., the so-called information pull) in which the user goes out of its business context and makes a (keyword-based) query can no longer be considered as an efficient one.

Based principally on the same information retrieval paradigm (query-answer), an opposite approach (the so-called information push) has emerged in the last ten years. Briefly, this approach tries to proactively deliver relevant information to users, by “guessing” what can be of interest for them and in which situation and by automatically starting the information retrieval process.

Figure 1 illustrates the information push scenario: A user is working in a process context and the system generates an information need out of this context that is serialized in a query, posted against an information repository. The results are presented to the user, and the initial request can be changed according to the user’s (explicit and/or implicit) feedback. The loop can be repeated unless the user is satisfied with the provided results.

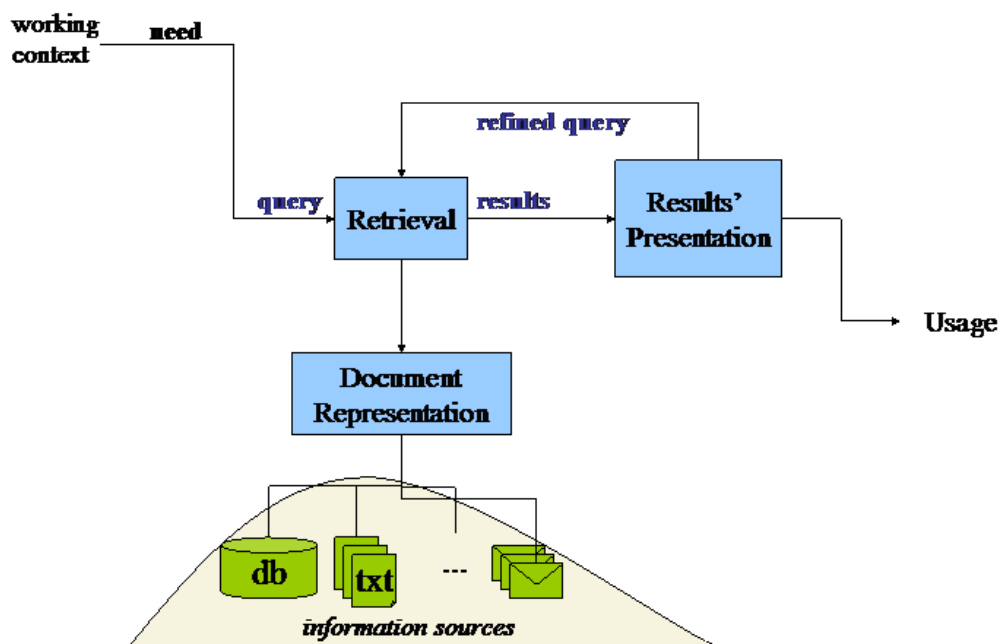


Figure 1: The general structure of the information push process

The information push process is realised in a class of systems called Recommender systems. Examples of such applications include recommending books, CDs, and other products at Amazon.com.

Recommender systems have become an important research area since the appearance of the first papers on collaborative filtering in the mid-1990s. There has been much work done both by the industry and academia on developing new approaches to recommender systems over the last decade. The interest in this area still remains high because: (a) it constitutes a problem-rich research area, and (b) there is abundance of practical applications that help users to deal with information overload and provide personalized recommendations, content, and services to them. However, in spite of all these advancements, the current generation of

recommender systems requires further improvements to make recommendation methods more effective and applicable to an even broader range of real-life applications. Figure 2 gives an overview of these challenges:

1. Keyword queries are ambiguous, i.e. the interpretation of what a user needs to find is ambiguous.
2. Documents are represented ambiguously, i.e. the used indexing methods usually process a document only statistically (syntactically).
3. Users consider only top ranked documents, i.e. if the ranking mechanism does not perform well, most relevant documents will be not found by a user.
4. Users do not know how to refine a query, i.e. due to a very large search space, a user is lost in the refinement process.

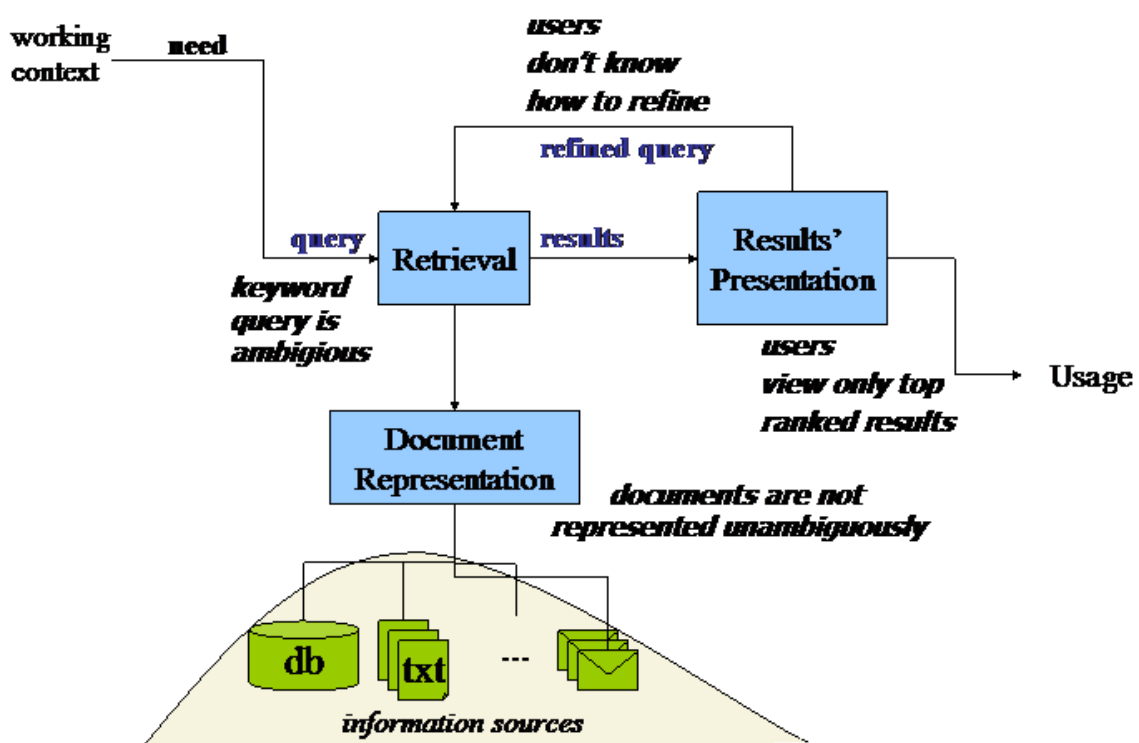


Figure 2: Challenges in the information push process

Since much of these challenges can be resolved by using more powerful (e.g. formal) modelling approaches, ontologies seems to be a good means for supporting this task and in this deliverable we will discuss this issue in details.

The deliverable is organised in the following way:

In section 2, a detailed introduction in the recommender systems is given, including their classification and research challenges.

In section 3, the role that an ontology can have in resolving the above mentioned challenges is outlined and an overview of existing approaches is given.

In section 4, an analysis of existing recommender systems in the software engineering domain is provided.

In section 5, some concluding remarks are presented.

2 FOUNDATIONS OF RECOMMENDER SYSTEMS

2.1 History and Definition

Although the roots of recommender systems can be traced back to the extensive work in cognitive science [Ric, 79], approximation theory [Pow, 81], information retrieval [Sal, 89], forecasting theories [Arm, 01], management science [Mur, 03] and to consumer choice modelling in marketing [Lil et al, 92], recommender systems emerged as an independent research area in the mid-1990s. At that time researchers started focusing on recommendation problems that explicitly rely on the ratings structure. In its most common formulation, the recommendation problem is reduced to the problem of estimating ratings for the items that have not been seen by a user. Intuitively, this estimation is usually based on the ratings given by this user to other items and on some other information that will be formally described below. Once we can estimate ratings for the yet unrated items, we can recommend to the user the item(s) with the highest estimated rating(s).

More formally, the recommendation problem can be formulated as follows: Let C be the set of all users and let S be the set of all possible items that can be recommended, such as books, movies, or restaurants. The space S of possible items can be very large, ranging in hundreds of thousands or even millions of items in some applications. Similarly, the user space can also be very large – millions in some cases. Let u be a utility function that measures the usefulness of item s to user c , i.e., $u : C \times S \rightarrow R$, where R is a totally ordered set (e.g., nonnegative integers or real numbers within a certain range). Then, for each user $c \in C$, we want to choose such item $s' \in S$ that maximizes the user's utility. More formally:

$$\forall c \in C, s'_c = \arg \max_{s \in S} u(c, s) \quad (1)$$

In recommender systems, the utility of an item is usually represented by a rating, which indicates how a particular user liked a particular item, e.g., John Doe gave the movie “Harry Potter” the rating of 7 (out of 10). However, as indicated earlier, in general, utility can be an arbitrary function, including a profit function. Depending on the application, utility u can either be specified by the user, as is often done for the user-defined ratings, or is computed by the application, as can be the case for a profit-based utility function.

Each element of the user space C can be defined with a *profile* that includes various user characteristics, such as age, gender, income, marital status, etc. In the simplest case, the profile can contain only a single (unique) element, such as User ID. Similarly, each element of the item space S is defined with a set of characteristics. For example, in a movie recommendation application, where S is a collection of movies, each movie can be represented not only by its ID, but also by its title, genre, director, year of release, leading actors, etc.

The central problem of recommender systems lies in that utility u is usually not defined on the whole $C \times S$ space, but only on some subset of it. This means u needs to be extrapolated to the whole space $C \times S$. In recommender systems, utility is typically represented by ratings and is initially defined only on the items previously rated by the users. For example, in a movie recommendation application (such as the one at MovieLens.org), users initially rate some subset of movies that they have already seen, i.e. there are users who have not rated a movie and vice versa. Therefore, the recommendation engine should be able to estimate

(predict) the ratings of the nonrated movie/user combinations and issue appropriate recommendations based on these predictions.

Extrapolations from known to unknown ratings are usually done by: 1) specifying *heuristics* that define the utility function and empirically validating its performance, and 2) *estimating* the utility function that optimizes certain performance criterion, such as the mean square error.

Once the unknown ratings are estimated, actual recommendations of an item to a user are made by selecting the highest rating among all the estimated ratings for that user, according to (1). Alternatively, we can recommend the N best items to a user or a set of users to an item.

The new ratings of the not-yet-rated items can be estimated in many different ways using methods from machine learning, approximation theory, and various heuristics.

In the next subsection we will discuss principal approaches for resolving problems in the recommender systems considering the whole information push process.

2.2 Classification/Types

As already mentioned in the first section, there are several challenges for the information push process. Figure 3 illustrates their reinterpretation from the recommender systems point of view, showing the three principle classification types for recommender systems:

- rating-based recommendation: regarding the methods for the retrieval of ratings;
- preference-based filtering: regarding the prediction of the relative preferences of users regarding retrieved results, i.e. the ranking of results; and
- implicit feedback: regarding the methods to capture user's feedback about calculated result.

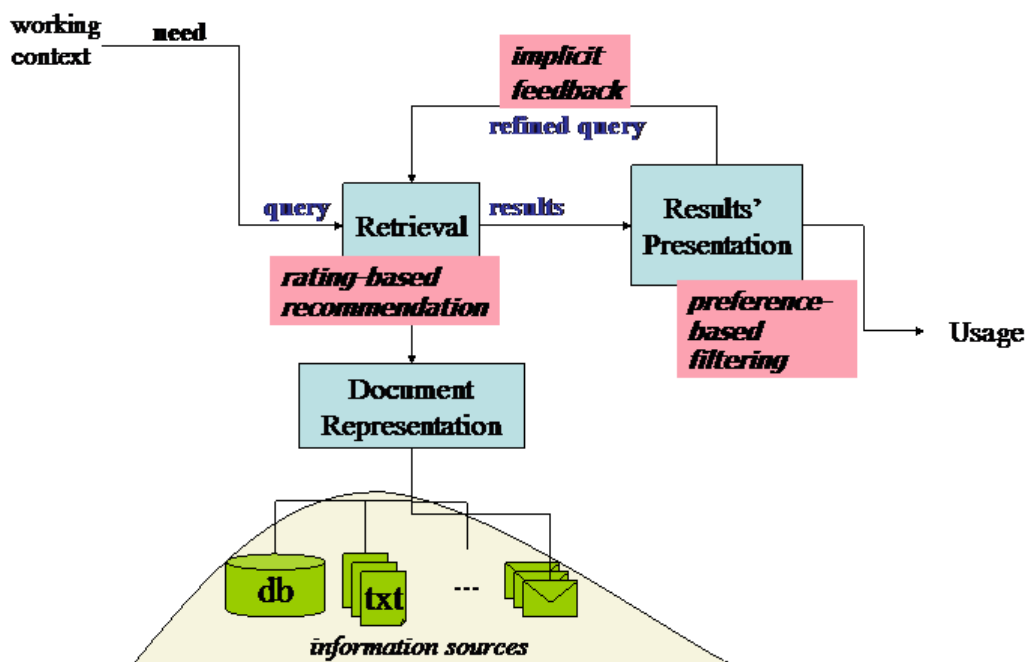


Figure 3: Three classification types of recommender systems and their role regarding information push process

2.2.1 Rating-based recommendation

Recommender systems are usually classified according to their approach to rating estimation [Bal, 97]:

- Content-based recommendations: The user will be recommended with items similar to the ones the user preferred in the past.
- Collaborative recommendations: The user will be recommended with items that people with similar tastes and preferences liked in the past.
- Hybrid approaches: These methods combine collaborative and content-based methods.

These approaches are subsequently described.

2.2.1.1 Content-based recommendations

In content-based recommendation methods, the utility $u(c, s)$ of item s for user c is estimated based on the utilities $u(c, s_i)$ assigned by user c to items $s_i \in S$ that are “similar” to item s . For example, in a movie recommendation application, in order to recommend movies to user c , the content-based recommender system tries to understand the commonalities among the movies user c has rated highly in the past (specific actors, directors, genres, subject matter, etc.). Then, only the movies that have a high degree of similarity to whatever the user’s preferences are would be recommended.

The content-based approach to recommendation has its roots in information retrieval [Sal, 89] and information filtering [Bel, 92] research. Because of the significant and early advancements made by the information retrieval and filtering communities and because of the importance of several text-based applications, many current content-based systems focus on recommending items containing textual information, such as documents, Web sites (URLs), and Usenet news messages. The improvement over the traditional information retrieval approaches comes from the use of user profiles that contain information about users’ tastes, preferences, and needs. The profiling information can be elicited from users explicitly, e.g., through questionnaires, or implicitly—learned from their transactional behaviour over time.

More formally, let $\text{Content}(s)$ be an item profile, i.e., a set of attributes characterizing item s . It is usually computed by extracting a set of features from item s (its content) and is used to determine the appropriateness of the item for recommendation purposes. Since, as mentioned earlier, content-based systems are designed mostly to recommend text-based items, the content in these systems is usually described with keywords. For example, a content-based component of the Fab system [Bal, 97], which recommends Web pages to users, represents Web page content with the 100 most important words. The “importance” (or “informativeness”) of word k_j in document d_j is determined with some weighting measure w_{ij} that can be defined in several different ways.

One of the best-known measures for specifying keyword weights in Information Retrieval is the term frequency/inverse document frequency (TF-IDF) measure [Sal, 89] that is defined as follows: Assume that N is the total number of documents that can be recommended to users and that keyword k_j appears in n_j of them. Moreover, assume that $f_{i,j}$ is the number of times keyword k_i appears in document d_j . Then, $\text{TF}_{i,j}$, the term frequency (or normalized frequency) of keyword k_i in document d_j , is defined as

$$TF_{ij} = \frac{f_{i,j}}{\max_z f_{z,j}}$$

where the maximum is computed over the frequencies $f_{z,j}$ of all keywords k_z that appear in the document d_j . However, keywords that appear in many documents are not useful in distinguishing between a relevant document and a nonrelevant one. Therefore, the measure of inverse document frequency (IDF_i) is often used in combination with simple term frequency ($TF_{i,j}$). The inverse document frequency for keyword k_i is usually defined as:

$$IDF_i = \log \frac{N}{n_i}.$$

Then, the TF-IDF weight for keyword k_i in document d_j is defined as

$$w_{i,j} = TF_{ij} \times IDF_i$$

and the content of document d_j is defined as

$$\text{Content}(d_j) = (w_{1j}, \dots, w_{kj}).$$

As stated earlier, content-based systems recommend items similar to those that a user liked in the past. In particular, various candidate items are compared with items previously rated by the user and the bestmatching item(s) are recommended. More formally, let $\text{ContentBasedProfile}(c)$ be the profile of user c containing tastes and preferences of this user. These profiles are obtained by analyzing the content of the items previously seen and rated by the user and are usually constructed using keyword analysis techniques from information retrieval. For example, $\text{ContentBasedProfile}(c)$ can be defined as a vector of weights (w_{c1}, \dots, w_{ck}) , where each weight w_{ci} denotes the importance of keyword k_i to user c and can be computed from individually rated content vectors using a variety of techniques. For example, some averaging approach, such as Rocchio algorithm [Roc, 71], can be used to compute $\text{ContentBasedProfile}(c)$ as an “average” vector from an individual content vectors. On the other hand, [Paz, 97] uses a Bayesian classifier in order to estimate the probability that a document is liked.

In content-based systems, the utility function $u(c, s)$ is usually defined as:

$$u(c, s) = \text{score}(\text{ContentBasedProfile}(c), \text{Content}(s)).$$

Content-based recommender systems have several limitations that are described in following text.

Limited Content Analysis

Content-based techniques are limited by the features that are explicitly associated with the objects that these systems recommend. Therefore, in order to have a sufficient set of features, the content must either be in a form that can be parsed automatically by a computer (e.g., text) or the features should be assigned to items manually. While information retrieval techniques work well in extracting features from text documents, some other domains have an inherent problem with automatic feature extraction. For example, automatic feature extraction methods are much harder to apply to multimedia data, e.g., graphical images, audio streams, and video streams. Moreover, it is often not practical to assign attributes by hand due to limitations of resources [Sha, 95].

Another problem with limited content analysis is that, if two different items are represented by the same set of features, they are indistinguishable. Therefore, since text based documents are usually represented by their most important keywords, content-based systems cannot distinguish between a well-written article and a badly written one, if they happen to use the same terms.

Overspecialization

When the system can only recommend items that score highly against a user's profile, the user is limited to being recommended items that are similar to those already rated. For example, a person with no experience with Greek cuisine would never receive a recommendation for even the greatest Greek restaurant in town. This problem, which has also been studied in other domains, is often addressed by introducing some randomness. For example, the use of genetic algorithms has been proposed as a possible solution in the context of information filtering [She, 93]. In addition, the problem with overspecialization is not only that the content-based systems cannot recommend items that are different from anything the user has seen before. In certain cases, items should not be recommended if they are too similar to something the user has already seen, such as a different news article describing the same event.

Therefore, some content-based recommender systems, such as Daily-Learner [Bil, 00], filter out items not only if they are too different from the user's preferences, but also if they are too similar to something the user has seen before. Furthermore, Zhang et al. [Zha, 02] provide a set of five redundancy measures to evaluate whether a document that is deemed to be relevant contains some novel information as well. In summary, the diversity of recommendations is often a desirable feature in recommender systems. Ideally, the user should be presented with a range of options and not with a homogeneous set of alternatives. For example, it is not necessarily a good idea to recommend all movies by Woody Allen to a user who liked one of them.

New User Problem

The user has to rate a sufficient number of items before a content-based recommender system can really understand the user's preferences and present the user with reliable recommendations. Therefore, a new user, having very few ratings, would not be able to get accurate recommendations.

2.2.1.2 Collaborative Methods

Unlike content-based recommendation methods, collaborative recommender systems (or collaborative filtering systems) try to predict the utility of items for a particular user based on the items previously rated by other users. More formally, the utility $u(c, s)$ of item s for user c is estimated based on the utilities $u(c_j, s)$ assigned to item s by those users $c_j \in C$ who are "similar" to user c . For example, in a movie recommendation application, in order to recommend movies to user c , the collaborative recommender system tries to find the "peers" of user c , i.e., other users that have similar tastes in movies (rate the same movies similarly). Then, only the movies that are most liked by the "peers" of user c would be recommended.

Algorithms for collaborative recommendations can be grouped into two general classes: memory-based (or heuristic-based) and model-based.

Memory-based algorithms (e.g. [Sha, 95] essentially are heuristics that make rating predictions based on the entire collection of previously rated items by the users. That is, the

value of the unknown rating $r_{c,s}$ for user c and item s is usually computed as an aggregate of the ratings of some other (usually, the N most similar) users for the same item s :

$$r_{c,s} = \text{aggr}(r_{c's})$$

where C' denotes the set of N users that are the most similar to user c and who have rated item s (N can range anywhere from 1 to the number of all users).

In contrast to memory-based methods, model-based algorithms (e.g. [Bre, 98]) use the collection of ratings to learn a model, which is then used to make rating predictions. For example, [Bre, 98] proposes a probabilistic approach to collaborative filtering.

As in the case of content-based techniques, the main difference between collaborative model-based techniques and heuristic-based approaches is that the model-based techniques calculate utility (rating) predictions based not on some ad hoc heuristic rules, but, rather, based on a model learned from the underlying data using statistical and machine learning techniques. A method combining both memory-based and model-based approaches was proposed in [Pen, 99], where it was empirically demonstrated that the use of this combined approach can provide better recommendations than pure memory-based and model-based collaborative approaches.

A different approach to improving the performance of existing collaborative filtering algorithms was taken in [Yu, 02], where the input set of user-specified ratings is carefully selected using several techniques that exclude noise, redundancy, and exploit the sparsity of the ratings' data. The empirical results demonstrate the increase in accuracy and efficiency for model-based collaborative filtering algorithms. It is also suggested that the proposed input selection techniques may help the model-based algorithms to address the problem of learning from large databases [Yu, 02].

The pure collaborative recommender systems do not have some of the shortcomings that content-based systems have. In particular, since collaborative systems use other users' recommendations (ratings), they can deal with any kind of content and recommend any items, even the ones that are dissimilar to those seen in the past. However, collaborative systems have their own limitations as described below.

New User Problem

It is the same problem as with content-based systems. In order to make accurate recommendations, the system must first learn the user's preferences from the ratings that the user gives. Several techniques have been proposed to address this problem. Most of them use the hybrid recommendation approach, which combines content-based and collaborative techniques. The next section describes hybrid recommender systems in more detail. An alternative approach is presented in [Ras, 02] where various techniques are explored for determining the best (i.e., most informative to a recommender system) items for a new user to rate. These techniques use strategies that are based on item popularity, item entropy, user personalization, and combinations of the above.

New Item Problem

New items are added regularly to recommender systems. Collaborative systems rely solely on users' preferences to make recommendations. Therefore, until the new item is rated by a substantial number of users, the recommender system would not be able to recommend it. This problem can also be addressed using hybrid recommendation approaches, described in the next section.

Sparsity

In any recommender system, the number of ratings already obtained is usually very small compared to the number of ratings that need to be predicted. Effective prediction of ratings from a small number of examples is important. Also, the success of the collaborative recommender system depends on the availability of a critical mass of users. For example, in the movie recommendation system, there may be many movies that have been rated by only few people and these movies would be recommended very rarely, even if those few users gave high ratings to them. Also, for the user whose tastes are unusual compared to the rest of the population, there will not be any other users who are particularly similar, leading to poor recommendations.

One way to overcome the problem of rating sparsity is to use user profile information when calculating user similarity. That is, two users could be considered similar not only if they rated the same movies similarly, but also if they belong to the same demographic segment. For example, [Paz, 99] uses the gender, age, area code, education, and employment information of users in the restaurant recommendation application. This extension of traditional collaborative filtering techniques is sometimes called “demographic filtering” [Paz, 99].

2.2.1.3 Hybrid Methods

Several recommendation systems use a hybrid approach by combining collaborative and content-based methods, which helps to avoid certain limitations of content-based and collaborative systems.

Different ways to combine collaborative and content-based methods into a hybrid recommender system can be classified as follows:

1. implementing collaborative and content-based methods separately and combining their predictions;
2. incorporating some content-based characteristics into a collaborative approach;
3. incorporating some collaborative characteristics into a content-based approach, and
4. constructing a general unifying model that incorporates both content-based and collaborative characteristics.

All of the above approaches have been used by recommender systems researchers, as described below.

Combining Separate Recommenders

One way to build hybrid recommender systems is to implement separate collaborative and content-based systems by combining the outputs (ratings) obtained from individual recommender systems into one final recommendation using either a linear combination of ratings [Cla 99] or a voting scheme [Paz, 99].

Adding Content-Based Characteristics to Collaborative Models

Several hybrid recommender systems, including Fab [Bal, 97] and the “collaboration via content” approach, described in [Paz, 99], are based on traditional collaborative techniques but also maintain the content-based profiles for each user. These content-based profiles, and not the commonly rated items, are then used to calculate the similarity between two users. As

mentioned in [76], this allows to overcome some sparsity-related problems of a purely collaborative approach since, typically, not many pairs of users will have a significant number of commonly rated items. Another benefit of this approach is that users can be recommended an item not only when this item is rated highly by users with similar profiles, but also directly, i.e., when this item scores highly against the user's profile [Bal, 97].

Adding Collaborative Characteristics to Content-Based Models

The most popular approach in this category is to use some dimensionality reduction technique on a group of content-based profiles. For example, [Sob, 99] uses latent semantic indexing (LSI) to create a collaborative view of a collection of user profiles, where user profiles are represented by term vectors, resulting in a performance improvement compared to the pure content based approach.

Developing a Single Unifying Recommendation Model

Many researchers have followed this approach in recent years. For instance, [Bas, 98] proposes using content-based and collaborative characteristics (e.g., the age or gender of users or the genre of movies) in a single rule-based classifier.

2.2.2 Preference-based filtering

In addition to recommender systems that predict the absolute values of ratings that individual users would give to the yet unseen items (as discussed above), there has been work done on *preference-based filtering*, i.e., predicting the relative preferences of users [Jin, 03]. For example, in a movie recommendation application, preference-based filtering techniques would focus on predicting the correct relative order of the movies, rather than their individual ratings.

2.2.3 Implicit feedback

2.2.3.1 Introduction

Recommender systems exploit ratings provided by an entire user population to reshape an information space for the benefit of one or more individuals [Oar, 97]. In research systems, these ratings are often provided explicitly by each user using one or more ordinal or qualitative scales.

Moreover, even in e-commerce applications, where the user's profile can be well defined, it is possible that the "current" search requires more specialized information than provided by the long-term user profile. It means that his/her information need in the current search session has to be discovered (short-term personalization) [Cro, 01]. Moreover, in some product areas (such as appliances or home electronic equipment) buyer interests typically only relate to a single buying session, and profile information is typically not re-used in later sessions. This is called *ephemeral personalization* [Sch, 01].

In the information retrieval community a class of so-called relevance feedback [Rut, 03] method is used for determining a user's profile from his feedback. This is an iterative process where the user assesses the relevance of a number of documents returned in response to an initial query. The user peruses the content of each document in this set, assesses it for relevance and mark those that best meet his information need. The limitations in providing increasingly better ranked results based solely on the initial query, and the resultant need for

query modification have already been identified. Relevance feedback systems automatically resubmit the initial query, expanding it by using terms taken from the documents marked relevant by the user. In practice, relevance feedback can be very effective but it relies on the user's assessing the relevance of documents and indicating to the system which documents contain relevant information. In a real scenario, the user may be unwilling to browse to documents web pages to gauge their relevance. Such a task imposes an increased burden and increased cognitive load. Documents may be lengthy or complex, the user may have time restrictions or the initial query may have retrieved a poor set of results.

Implicit relevance feedback [Roc, 71] is a method to discover a user's preferences from the information that can be derived from his *usual* interaction with the retrieval system. It means that this information suggests implicitly what the users prefers, i.e. what he is exactly searching for.

Therefore, implicit feedback techniques seek to avoid bottleneck by inferring something similar to the ratings that a user would assign from observations that are available to the system. Such an approach could greatly extend the range of applications for which recommender systems would be useful.

2.2.3.2 Sources of implicit feedback

Nichols [Nic, 97] surveyed the state of the art in implicit feedback techniques with an eye toward their potential use for information filtering. Table 1 presents the sources identified by Nichols and some others that we believe will also be useful. In addition to explicit ratings we have identified three broad categories of potentially useful observations: examination, retention and reference.

Information systems often provide brief summaries of several promising documents using some sort of selection interface display, and selection of individual objects for further examination can thus provide the first cue about a user's interests. USENET newsreader software typically records the identifiers of messages that users have seen, and Karlgren [Kar, 94] explored the design of a recommender system using such lists. Konstan [Kon, 97] found a positive correlation between reading time and explicit ratings in USENET news applications, and we have generalized that source of observations as "examination duration" to accommodate other modalities such as audio and video. [Hil, 92] have developed this idea further, defining "edit wear" as an analogue to the useful effects of uneven wear that physical materials accumulate over time that provide other users with cues that help discover useful materials and useful items of those items. In text browsing, for example, edit wear might be measured by using dwell times at specific locations in the text to characterize scrolling behavior. Examination may extend beyond more than a single interaction between user and system, and we seek to capture that source of observations by characterizing the repetition of the foregoing user behaviors. Finally, when information access is priced on a per-item basis, purchase decisions offer extremely strong evidence of the value ascribed to an object. Similar information would be available at a somewhat coarser scale when users purchase subscription access to certain types of content (e.g., subscription to a separately priced cable television channel).

Category	Observable Behavior
Examination	Selection Duration Edit wear Repetition Purchase (object or subscription)
Retention	Save a reference or save an object (with or without annotation) (with or without organization) Print Delete
Reference	Object->Object (forward, reply, post follow up) Portion->Object (hypertext link, citation) Object->Portion (cut & paste, quotation)

Table 1: Observable behaviour for implicit feedback

Our “retention” category is intended to group those behaviors that suggest some degree of intention to make future use of an object. Bookmarking a web page is a simple example of such a behavior, and we have generalized that idea as “save a reference” to accommodate a wider range of actions such as construction of symbolic links within a file system. Rucker & Polanco [Ruk, 97], for example, constructed a recommender system using bookmark lists. Saving the object itself is the obvious alternative, something Stevens [Ste, 93] used as implicit feedback for content-based filtering. In either case, the object may be saved with or without some form of annotation. For example, web browsers typically default to using the page title in the bookmark list, but users may optionally provide a more meaningful entry if they desire. Although numerous confounding factors would likely be present, it may be possible to infer something about the value a user places on an individual page by whether or not they go to the trouble of constructing an informative bookmark entry. Similarly, users may choose to save a reference or an object in an explicitly organized fashion or in the default manner. For example, storing electronic mail about this workshop in a new folder might provide greater support for an inference that the user ascribes particular value to the message than would the use of some default scheme such as placing it in the folder routinely used for mail from the message’s originator. The salient issue in this case is not the act of organizing, but rather the way in which the organization given to an individual object distinguishes it from the way in which similar forms of organization are assigned to other objects. This difference may not be easy to characterize, but it may be worth thinking about how to do it. We have chosen to group printing with retention because of the permanence of the printed page, but users may also print document or images to facilitate examination because paper still has some decided advantages over electronic displays in many applications. Printing overlaps with the next category (reference) as well, since users may print a document or image with the intention of forwarding them to another individual or including portions in another document. Nevertheless, printing is often associated with a

desire for retention, so we find this grouping useful. As with examination, it may be possible to infer something about the portions of a document that the user finds most valuable from the portions which he or she chooses to print. Finally, the retention category is distinguished by the possibility of directly observing evidence of negative evaluations as well. When retention is a default condition, as in some electronic mail systems, a decision by the user to delete an object might support to an inference that the deleted object is less valued than other objects that are retained.

The “refer to” category may appear at first glance to contain a fairly eclectic group of observable activities, but each has the effect of establishing some form of link between two objects. Forwarding a message, for example, establishes a link between the new message and the original. Similarly, replying individually or posting a follow up message to some form of group venue such as a mailing list establishes the same sort of link. Goldberg et al. [Gol, 92] described a simple example of this in which users could construct an electronic mail filter to display messages that their colleagues had taken the time to reply to. Hypertext links from one web page to another and bibliographic citations in academic papers create links from a portion of an object (characterized, perhaps, by some neighborhood around the link itself) to another object, although the refinement to a portion of a document has not been exploited often. Brin & Page [Bri, 98] provide an example of how hypertext links might be used, although their focus is on a population statistics rather than individual preferences. Garfield [Gar, 79] describes the design of retrieval systems that are based on bibliographic citations. Alternatively, selective inclusion of another document, using either cut-and-paste or a quotation, creates a link from an information object to a portion of another.

2.2.3.3 Using Implicit Feedback

The goal of a recommender system is to help users find desirable information objects. That task combines inference and prediction, and Figures 4 and 5 show alternative strategies for accomplishing this. Figure 4 depicts a modular strategy in which the inference stage seeks to produce ratings similar to those that a user would have explicitly assigned, and then the prediction stage uses those estimated ratings to predict future ratings. Konstan et al. [Kon, 97] adopted this perspective when evaluating how well observed reading time predicted explicit ratings for individual articles. Figure 5 shows an alternative strategy in which past observations are used to predict user behavior in response to new information, and then the inference stage seeks to estimate the value of the information based on the predicted behavior. We are not aware of any implementations of this second approach, but Stevens [Stev, 93] implemented a simplified version of the strategy. He predicted the examination duration for a new USENET news article based on the examination durations for similar articles in the past and then constructed content-based queries that would select articles with long predicted examination durations. This essentially amounts to a degenerate inference stage in which desirability is assumed to increase monotonically with examination duration.

The distinction between the two strategies is quite subtle in the case of content-based filtering. In a recommender system, by contrast, the strategy shown in Figure 4 would characterize each article using the examination durations reported by other users, while the strategy shown in Figure 5 would characterize each article using the predicted ratings for other users. Recommender systems based on the second strategy might be more flexible, since participating users might draw different inferences from the same observations if they did not share a common set of objectives. On the other hand, recommender systems using the first strategy would likely have more context available locally for interpreting observations

than would be available at other points in the network. It might thus be worth considering hybrid approaches in which some preliminary interpretation is performed locally when the observation is made and then additional inferences are drawn at other points in the network.

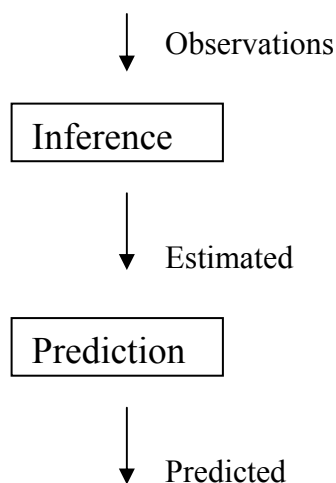


Figure 4: Rating estimation strategy

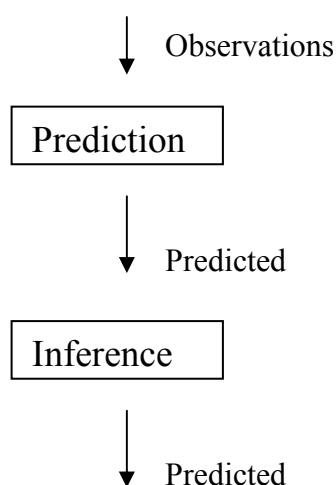


Figure 5: Predicted observations strategy

2.3 Research issues

Recommender systems can be extended in several ways that include improving the understanding of users and items, incorporating the contextual information into the recommendation process, supporting multicriteria ratings, and providing more flexible and less intrusive types of recommendations. Such more comprehensive models of recommender systems can provide better recommendation capabilities. In the remainder of this section, we

describe the proposed extensions and also identify various research opportunities for developing them.

2.3.1 Comprehensive Understanding of Users and Items

As was pointed out already, most of the recommendation methods produce ratings that are based on a limited understanding of users and items as captured by user and item profiles and do not take full advantage of the information in the user's transactional histories and other available data. For example, classical collaborative filtering methods do not use user and item profiles at all for recommendation purposes and rely exclusively on the ratings information to make recommendations. Although there has been some progress made on incorporating user and item profiles into some of the methods since the earlier days of recommender systems these profiles still tend to be quite simple and do not utilize some of the more advanced profiling techniques.

In addition to using traditional profile features, such as keywords and simple user demographics [Moo, 99], more advanced profiling techniques based on data mining rules [Ado, 01], sequences [Man, 95], and signatures [Cor, 00] that describe a user's interests can be used to build user profiles. Also, in addition to using the traditional item profile features, such as keywords [Paz, 99], similar advanced profiling techniques can also be used to build comprehensive item profiles. With respect to recommender systems, advanced profiling techniques that are based on data mining have been used mainly in the context of Web usage analysis [Li, 04], i.e., to discover the navigational Web usage patterns (i.e., page view sequences) of users in order to provide better Web site recommendations; however, such techniques have not been widely adopted in rating-based recommender systems.

2.3.2 Multidimensionality of Recommendations

The current generation of recommender systems operates in the two-dimensional User \times Item space. That is, they make their recommendations based only on the user and item information and do not take into consideration additional contextual information that may be crucial in some applications. However, in many situations, the utility of a certain product to a user may depend significantly on time (e.g., the time of the year, such as season or month, or the day of the week). It may also depend on the person(s) with whom the product will be consumed or shared and under which circumstances. In such situations, it may not be sufficient to simply recommend items to users; the recommender system must take additional contextual information, such as time, place, and the company of a user, into consideration when recommending a product.

For example, when recommending a vacation package, the system should also consider the time of the year, with whom the user plans to travel, traveling conditions and restrictions at that time, and other contextual information. As another example, a user can have significantly different preferences for the types of movies she wants to see when she is going out to a movie theater with a boyfriend on a Saturday night as opposed to watching a rental movie at home with her parents on a Wednesday evening.

Therefore it is important to extend traditional two-dimensional User \times Item recommendation methods to multidimensional settings. In addition, [Her, 01] argued that the inclusion of the knowledge about the user's task into the recommendation algorithm in certain applications can lead to better recommendations.

2.3.3 *Multicriteria Ratings*

Most of the current recommender systems deal with single criterion ratings, such as ratings of movies and books.

However, in some applications, such as restaurant recommenders, it is crucial to incorporate multicriteria ratings into recommendation methods. For example, many restaurant guides, such as Zagat's Guide, provide three criteria for restaurant ratings: food, decor, and service. Although multicriteria ratings have not yet been examined in the recommender systems literature, they have been extensively studied in the Operations Research community [Sta, 95]. Typical solutions to the multicriteria optimization problems include:

1. finding Pareto optimal solutions;
2. taking a linear combination of multiple criteria and reducing the problem to a single-criterion optimization problem;
3. optimizing the most important criterion and converting other criteria to constraints, and
4. consecutively optimizing one criterion at a time, converting an optimal solution to constraint(s), and repeating the process for other criteria.

An example of the latter approach is the method of successive concessions [Sta, 95].

2.3.4 *Nonintrusiveness*

Many recommender systems are intrusive in the sense that they require explicit feedback from the user and often at a significant level of user involvement. For example, before recommending any newsgroup articles, the system needs to acquire the ratings of previously read articles and, often, many of them. Since it is impractical to elicit many ratings of these articles from the user, some recommender systems use nonintrusive rating determination methods where certain proxies are used to estimate real ratings. For example, the amount of time a user spends reading a newsgroup article can serve as a proxy of the article's rating given by this user. However, nonintrusive ratings (such as time spent reading an article) are often inaccurate and cannot fully replace explicit ratings provided by the user. Therefore, the problem of minimizing intrusiveness while maintaining certain levels of accuracy of recommendations needs to be addressed by the recommender systems researchers.

One way to explore the intrusiveness problem is to determine an optimal number of ratings the system should ask from a new user. For example, before recommending any movies, MovieLens.org first asks the user to rate a predefined number of movies (e.g., 20). This request incurs certain costs on the end-user that can be modeled in various ways, the simplest model being a fixed-cost model (i.e., the cost of rating each movie is C and the cost of rating n movies is $C*n$).

2.3.5 *Flexibility*

Most of the recommendation methods are inflexible in the sense that they are "hard-wired" into the systems by the vendors and, therefore, support only a predefined and fixed set of recommendations. Therefore, the end-user cannot customize recommendations according to his or her needs in real time.

2.3.6 Effectiveness of Recommendations

The problem of developing good metrics to measure the effectiveness of recommendations has been extensively addressed in the recommender systems literature. In most of the recommender systems literature, the performance evaluation of recommendation algorithms is usually done in terms of coverage and accuracy metrics. Coverage measures the percentage of items for which a recommender system is capable of making predictions [Her, 99]. Accuracy measures can be either statistical or decision-support. Statistical accuracy metrics mainly compare the estimated ratings against the actual ratings R in the User \times Item matrix and include Mean Absolute Error (MAE), root mean squared error, and correlation between predictions and ratings. Decision-support measures determine how well a recommender system can make predictions of high-relevance items (i.e., items that would be rated highly by the user). They include classical IR measures of precision (the percentage of truly “high” ratings among those that were predicted to be “high” by the recommender system), recall (the percentage of correctly predicted “high” ratings among all the ratings known to be “high”), Fmeasure (a harmonic mean of precision and recall), and Receiver Operating Characteristic (ROC) measure demonstrating the trade-off between true positive and false positive rates in recommender systems [Her, 99].

Although popular, these empirical evaluation measures have certain limitations. One limitation is that these measures are typically performed on test data that the users chose to rate. However, items that users choose to rate are likely to constitute a skewed sample, e.g., users may rate mostly the items that they like. In other words, the empirical evaluation results typically only show how accurate the system is on items the user decided to rate, whereas the ability of the system to properly evaluate a random item (which it should be able to do during its normal real-life use) is not tested. Understandably, it is expensive and time-consuming to conduct controlled experiments with users in the recommender systems settings, therefore, the experiments that test recommendation quality on an unbiased random sample are rare. However, high-quality experiments are necessary in order to truly understand the benefits and limitations of the proposed recommendation techniques.

In addition, although crucial for measuring the accuracy of recommendations, the technical measures mentioned earlier often do not adequately capture “usefulness” and “quality” of recommendations. For example, as [Yan, 01] observes for a supermarket application, recommending obvious items (such as milk or bread) that the consumer will buy anyway will produce high accuracy rates; however, it will not be very helpful to the consumer. Therefore, it is also important to develop economics-oriented measures that capture the business value of recommendations, such as return on investments (ROI) and customer lifetime value (LTV) measures [Dwy, 89]. Developing and studying the measures that would remedy the limitations described in this section constitutes an interesting and important research topic.

3 ONTOLOGY-BASED RECOMMENDER SYSTEMS

3.1 The role of ontologies

As already mentioned in the introduction (see Figure 2), there are several challenges that should be resolved in order to make recommender systems more efficient. In this subsection we give an overview how ontologies can help in resolving them.

3.1.1 *Ontology-based Retrieval*

As already explained in section 2, in the nutshell of a recommender system is a representation mechanism of a user's profile, i.e. interests of a user. Obviously, by using only syntactically-driven mechanisms (like mentioned TFIDF), the precision of the retrieval process decreases (what is already known from the traditional IR). Indeed, the main problem in such retrieval is the ambiguity in the representation, e.g. a keyword-based representation of a preference has lots of interpretation. For example, just by saying that a user likes jaguars is not enough to determine what does it prefer: cars or animals. By using ontologies such ambiguities can be resolved.

Principally, the terms defined in ontology are used as metadata to markup the preferences; these semantic markups are semantic index terms for the retrieval. By using description logic reasoner we can obtain the equivalent classes of semantic index terms. The logical views of documents/items and user information needs / preferences, generated in terms of the equivalent classes of semantic index terms, can represent documents and user information needs well, so the performance of information retrieval can be improved effectively when suitable ranking function is chosen.

There are two general approaches:

1. disambiguation of a keyword-based representation using ontology-based interpretation of keywords; and
2. formalisation of the representation through logic expressions.

In the next subsection we give more details about these approaches.

Obviously, this approach is similar to the model-based recommendation mentioned in 2.2.1, but it uses much more powerful and formal model – ontologies.

Another point of view is the comparison with hybrid recommendation systems which can also be augmented by knowledge-based techniques [Bur, 00], such as case-based reasoning, in order to improve recommendation accuracy and to address some of the limitations (e.g., new user, new item problems) of traditional recommender systems. For example, knowledge-based recommender system *Entre'e* [Bur, 00] uses some domain knowledge about restaurants, cuisines, and foods (e.g., that “seafood” is not “vegetarian”) to recommend restaurants to its users.

The main drawback of knowledge-based systems is a need for knowledge acquisition - a well-known bottleneck for many artificial intelligence applications. However, knowledge-based recommendation systems have been developed for application domains where domain knowledge is readily available in some structured machine-readable form, e.g., as an ontology. For example, the Quickstep and Foxtrot systems use research paper topic ontology to recommend online research articles to the users.

Ontology-based interpretation of keywords

This approach is based on the usage of ontology concepts/relations for the disambiguation of the meaning of a keyword, or even the whole query. For example, by adding information that the preference “jaguar” belongs to the class of animals, the ambiguity in the preferences’ interpretation (jaguar-car vs. jaguar-animal) disappears.

There are several possibilities to add “meaning” to keywords:

1. by considering concept hierarchy, e.g. that a car is a vehicle:

In the traditional IR literature can be found that using concept hierarchy precision increased from 76.4% to 82.2% [Wang et al 03];

2. by indexing with the meaning defined in existing glossaries:

[Gonzalo et al 98] shows that indexing documents with wordnet synsets improves text retrieval up to 29%. [Navigli, Velardi 03] shows that expanding with WordNet produces a 26,83% improvement over the plain query words (in terms of precision);

3. by formalizing named entities:

Named entities are concepts from a text that belong into predefined categories such as the names of persons, organizations, locations, expressions of times, quantities, monetary values, percentages. Named entities modeled separately perform better precision 0.847 vs. 0.743 [Li et al 05].

Various types of ontologies can be used, like spatial ontologies [Jones et al 02].

In the traditional IR this problem would be classified into word sense disambiguation [Navigli, Velardi 2004].

Use logic expression (formalization)

The idea behind this approach is to formalize the meaning of a keyword based structure, like a query is. For example, for a query “process” and “knowledge”, there is a large space of possible interpretations what the user meant:

- 1) “process knowledge” as a type of knowledge;
- 2) “knowledge process” as a type of process, or
- 3) “process that is somehow, but not directly related to knowledge”.

By introducing an ontology interpretation a type of relation can be added to the query, like using `in(process, knowledge)`, that determines what is a query about. A similar approach can be found in [Sto, 2005].

Another approach is to translate NL queries in the logic form, as realised in the system Aqualog [Lopez, Motta, 04]. In [Van Bakel 98] a system called Condorcat for transforming Boolean queries in the logic form is presented. The results are promising (improvement in precision), but the main problems are scalability and cost-effectiveness. Finally, query can be formalized by a user which means that a user has to enter the query in the formal form, which is usually a difficult task. Usually, in order to hide the complexity of the formalisation, a user-friendly graphical interface is provided.

Finally, Table 2 [Finin et al 05] shows the difference in the mean average precision in various types of the retrieval. Obviously, using more structure increases the precision of the retrieval.

Representation formalism	Unstructured data (e.g., free text)	Structured data (RDF, DAML+ OIL, OWL)	Structured data plus free text
Mean average precision	25.9%	66.2%	85.5%

Table 2: The increase in retrieval’s precision by introducing more structure in the representation mechanism

3.1.2 Presentation of results

Due to information overload, it is to be expected that lots of relevant recommendation can be generated, so that the presentation of these results can be one of crucial issues for the acceptance of the system/results. In fact the problem/goal is how to facilitate the exploration/understanding of results. It can be considered as a solution for enriching the results. It is in fact complementary to the search process and is also a way to increase the result “relevance” for the user. If the result quality remains a major concern, the quality of the result restitution (organization and visualization) must be taken into account too. Without an effective organization of the results, the user has to process manually the huge amount of results or refine the query in order to limit the results.

Therefore, the solutions must enable a more relevant result organization, a richer visualization interface and an intuitive navigation in the result space.

Due to its generalisation possibilities, ontologies are very powerful means for supporting visualisation/understanding of results and in the following we are discussing some of approaches. The discussion is structured according to the role an ontology plays in the presentation:

1. ontology-based visualization: to visualise complex relation between results;
2. ontology-based clustering of results: to discover commonalities between results;
3. ontology-based browsing/navigation: to support better exploration of results;
4. ontology-based ranking: to put the most relevant results on the top of results’ list.

Ontology-based visualization

Since an ontology represents a conceptual model of a domain, it can be used as a very suitable means for structuring results. In other words, if the results of a search can be mapped into related domain ontology, then the visualization of results so that they are placed in the corresponding concepts (they belong to) can be very useful. For example, all relations that exist between concepts in the ontology can be used for connecting results that are visualized.

The domain knowledge is usually described in the form of taxonomies. Taxonomies are frequently used in several domains (e.g. biology, chemistry, libraries) as classification systems. Information architects consider taxonomies as basic building blocks, representing the backbone of most web sites. Non-formal taxonomies are already widely used in web applications for product classification (e.g. Amazon) or web-directories (e.g. Yahoo, Open Directory Project (ODP)).

A very interesting approach is Aduna's Cluster Map, described in [Fluit, Sabou, Harmelen, 05]. The Cluster Map is used to visualize light-weight ontologies that describe a domain through a set of classes (concepts) and their hierarchical relationships. The authors report lots of advantages of this type of visualisation, like the possibilities to analyse the results (e.g. in an e-commerce scenario to show the distribution of products among domain concepts, which can be useful to discover concepts with low/large coverage with products), or the support for query reformulation/refinement (see below ontology browsing part)

Ontology-based clustering of results

Clustering can be used to improve the results of retrieval, which has been investigated in many previous works [Hea, 99]. The two main possibilities are static clustering on the entire corpus and on-the-fly clustering which can be considered as a post-retrieval results browsing technique (e.g. the clustering engine vivisimo.com). However, the second approach is more relevant for this deliverable, since the recommendation space is difficult to model in advance.

The main advantage of using ontologies is the addition of the domain knowledge in the clustering process, i.e. an extension of the results with the background information so that the similarity between results, required for deciding which cluster a results belong to, can be calculated in a more efficient way. An interesting example is given in [Hotho et al 03], where WordNet has been used as the domain knowledge. The authors reported an improvement of 8.4 % in cluster purity, tested in clustering text documents (i.e. news texts).

Ontology-based browsing/navigation

Besides visualizing results, an ontology can support exploration of results in terms of helping the user to select a part of the results he/she would like to inspect.

Indeed, users are sometimes interested in going through list of results in order to get an impression of what can be found in an information repository. One of the reasons for this can be the unfamiliarity with the underlying information repository or the used vocabulary/taxonomy. Second, a user can change his need when he considers the alternatives (new products or new features). In other words, users are not aware of all preferences until they see them violated. For example, the user does not think of stating a preference for intermediate airport until a solution proposes to change airplane in a place that he dislikes. Finally, a user wants to be sure that the offer he found is the best possible solution for his need. For example, if a user is willing to buy a notebook with the Intel Pentium 4 2.4GHz processor that costs up to 1500 EURO and he finds that there is a notebook with the Intel Pentium 4 2.6GHz processor equipped with a DVD almost for the same price, then he might be very likely to decide to buy it. It is clear that in all these cases a user is "forced" to change/extend/adapt his query in order to find the most suitable product for his need. By using ontologies, the relations between results can be easily represented so that the browsing process is alleviated. For example, someone would not browse a class of results that has intersection with other classes, i.e. there are no elements (e.g. products) that belong only to this class. In [Sto, 05] an approach for this kind of the results exploration, called Query by Navigation, is presented. This approach is based on the (ontology-based) calculation of the neighbourhood of a query that indicates what will happen if the query is changed. The approach can be very efficiently applied in the case of the so-called cooperative answering, a

kind of recommendation how to refine a query to get more relevant results. For example, cooperative answering can be used for:

a) failing query resolution:

When a query fails, a system could be more cooperative by helping to trace the reason for the query's failure, or at least to pinpoint to the failure

b) alternative results:

An alternative result for a query is a result that does not fulfil all the constraints from the query perfectly (i.e. fulfils some of them), but has (many) suitable features from the user's point of view (e.g. usability).

Another interesting approach is given in [Aussenac, Mothe 04], where a multi-dimensional (OLAP-like) view on documents is enabled. Moreover hierarchies used in these views are inferred from an ontology based on a chosen relation (not only is-a relation).

Ontology-based ranking

Ranking of results is the most common support for the better presentation of results that can be found in the literature (e.g. the competitive advantages of Google was/is based on its ranking algorithm). An efficient ranking is usually based on the addition of some contextual information (e.g., which business process is the user currently performing, or which Office application is open) or some background information about the user (like what she/he did search in the past, or which documents are already stored on her/his computer) in order to define the relevance/preference for retrieved results (note that retrieval is usually based only on the content of documents and ranking is performed as a post-processing step).

Ontologies can support this process

- by helping in the definition of the relevance of a results:

For example, by using an ontology it can be clear that, in general, a document about ontology-based visualisation is more relevant than ontology-based 3D visualisation for someone who is interested in visualisation, since the distance between the concept visualisation and ontology-based visualisation is smaller than the distance between the concept visualisation and ontology-based 3D visualisation (in an corresponding ontology) [Mae, 03];

- the explanation of the resulting ranking:

Usually the ranking process generates a list of results without any explanation why/how it is produced, which introduces additional problem in judging about the relevance of results. An ontology can be used for expressing such explanation. This feature can be very efficiently combined with the formalisation of queries (described in 3.1.1), since in that case the retrieval algorithm is based on the evaluation of ontology rules (axioms) and the derivation path can be used for the explanation [Sto, 03].

3.1.3 *Implicit feedback*

As already explained in section 2.2.3, implicit relevance feedback is a method to discover a user's preferences from the information that can be derived from his *usual* interaction with the retrieval system. It means that this information suggests implicitly what the users prefer. In order to get better prediction of user's preferences these methods can be expanded and extended by using ontologies.

The theory about the implicit relevance feedback postulates that if the user selects a resource from the list of retrieved results, then this entry corresponds, to some extent, to his information need. As a list of results we consider a list of resources or a list of query refinements. However, a click on a particular result in the list cannot be treated as an absolute relevance judgement, since the user typically scans only the top l ranked ($l \approx 10$) results. For example, maybe a result ranked much lower in the list was much more relevant, but the user spotted it sooner. It appears that the user clicks on the (relatively) most promising results in the top l , independently of their absolute relevance. However, if we assume that the user scans the list of results from top to bottom, the relative relevance is evident: all non-clicked-on results placed above a clicked-on result are less relevant than the clicked-on result. If a result is an information resource, the relevance is related to some features that are contained in the clicked-on resource and not contained in non-clicked-on resources. It means that by analysing the commonalities in the attributes of results the user clicked/not clicked, we can infer more information about the intension of the user in the current query session. This is exactly where ontologies come in the game: to support the interpretation of the results, i.e. to find their commonalities.

Indeed, just by selecting several results from the results list, a user does not define precisely what he wants, but gives hints about what his goal may be. In the case that these results can be analysed from various perspectives, the probability that the user's interaction is interpreted in the correct way increases. For example, if a user selected two cars from the list, all possible common characteristics of these cars have to be detected and analysed, including characteristics that can be inferred as common, like that both cars are produced in Asia. Indeed, in order to deduce this fact, the knowledge about which country belongs to which continent is needed. This is exactly knowledge that can be coded in an ontology.

A comprehensive ontology-based approach for defining implicit feedback can be found in [Sto, 05]. It takes into account the implicit irrelevance (when an item has been not selected, but it is top ranked) and incremental refinement (a user selects results in several iterative recommendation phases, i.e. after each selection a new recommendation is generated, like an iterative query refinement)

In order to achieve this, the relation *preferred ranking* [Sto, 05] is defined as

$$R_i <_{r_Q} R_j \text{ for all pairs } 1 \leq j < i, \text{ with } i \in C \text{ and } j \notin C,$$

where (R_1, R_2, R_3, \dots) is a ranked list of results,

set C contains the ranks of the clicked-on results and

Q is the posted query.

ImplicitRelevance By analysing the difference between the features (attributes, constraints) of the clicked-on and non-clicked-on resources we get a set of so called *Preferred* constraints for query Q in the following manner:

$$Preferred(Q) = \{con \mid con \in \cup_j Preferred_j(Q)\},$$

where

$$Preferred_j(Q) = \{el \mid el \in Attr(R_j) \setminus \cup_i Attr(R_i), \forall i R_i <_R Q^* R_j\},$$

$Attr(R_x)$ = set of constraints (attributes) that are defined for R_x

Therefore, the set of constraints that seems to be relevant for the user in query Q_s can be calculated as:

$$ImplRel(c, Q_s) = \begin{cases} 0, & c \notin Preferred(Q_s) \\ \frac{1}{n} \sum_{i=1, n} \frac{1}{num_sessions(c, Q_s, i)}, & c \in Preferred(Q_s) \end{cases} \quad (2)$$

where $num_sessions(c, Q_s, i)$ is the number of refinement steps which constraint c is involved as a preferred constraint in, whereas n is the total number of times constraint c is treated as a preferred constraint in current session Q_s .

In this way we decrease the likelihood that a preferred constraint is still relevant if it was suggested as relevant in a previous refinement step, but was not selected by the user as relevant in the subsequent refinement step (see (2)). Therefore, our approach has a self-improvement characteristic – it learns from its failures.

ImplicitIrrelevance. Since the recommended refinements are presented to the user in the decreasing order of relevance, one can assume that if the user has selected n -th ranked results, then the first $n-1$ ranked results (constraints) are wrongly ranked on the top of the list of the refinements. We call these constraints *implicit irrelevance*. They are calculated in a similar manner as implicit relevant constraints:

$$ImplIrrel(c, Q_s) = \begin{cases} 0, & c \notin NonPreferred(Q_s) \\ \sum_{i=1, n} \frac{1}{num_sessions(c, Q_s, i)}, & c \in NonPreferred(Q_s) \end{cases} \quad (3)$$

where

$$NonPreferred(Q) = \{con \mid con \in \cup_j NonPreferred_j(Q)\}$$

$$NonPreferred_j(Q) = \{el \mid el \in \cup_i Attr(R_i) \setminus Attr(R_j), \forall i R_i <_R Q^* R_j\}.$$

The definition of $num_sessions(c, Q_s, i)$ is analogue to above, but regarding implicit irrelevance.

Similar to (2), formula (3) enables the correction of false assumptions (regarding the preferences of the current user) made in the ranking process.

Formulas (2) and (3) ensures the self-adaptivity of the ranking system, e.g. they do not allow that the system repeatedly ranks a non-interesting refinement highly. Finally, the calculated implicit relevance of refinement c looks like:

$$Relevance(c, Q_s) = \frac{ImplRel(c, Q_s) + 1}{ImplIrrel(c, Q_s) + 1}$$

3.2 Example systems

In this section we provide short overviews of two approaches for the semantic-based recommendation:

1. Ontology-based profiling in recommender systems [Middleton, 03]
2. Decentralised recommender for Semantic Web [Ziegler, 05]

3.2.1 *Ontology-based profiling*

Capturing user preferences is a problematic task. As we already explained, simply asking the users what they want is too intrusive and prone to error, yet monitoring behaviour unobtrusively and finding meaningful patterns is both difficult and computationally time consuming. [Middleton, 03] represents an approach that uses an ontology to represent user profiles and shows advantages over traditional profile representations in the context of recommender systems.

The approach is implemented in two experimental recommender systems: Quickstep and Foxtrot. We give here an overview only of the basic variant, i.e. Quickstep system.

The Quickstep system

Introduction

As the trend to publish research papers on-line increases, researchers are increasingly using the web as their primary source of papers. Typically, researchers need to know about new papers in their field of interest, and older papers relating to their current work. In addition, a researcher's time is limited, so browsing competes with other tasks in the work place. The Quickstep recommender system addresses this problem by recommending interesting publications.

Since researchers have their usual work to perform, unobtrusive monitoring methods are preferred because a researcher will be reluctant to use the system if it significantly interrupts normal workflow. Also, high recommendation accuracy is not critical as long as the system is deemed useful to them.

Quickstep unobtrusively monitors user browsing behaviour via a web proxy server, logging each URL browsed during normal work activity. A machine-learning algorithm classifies browsed URLs overnight, and saves each classified paper in a central paper store. Explicit feedback and browsed topics form the basis of the interest profile for each user. Figure 6 shows an overview diagram of the Quickstep system.

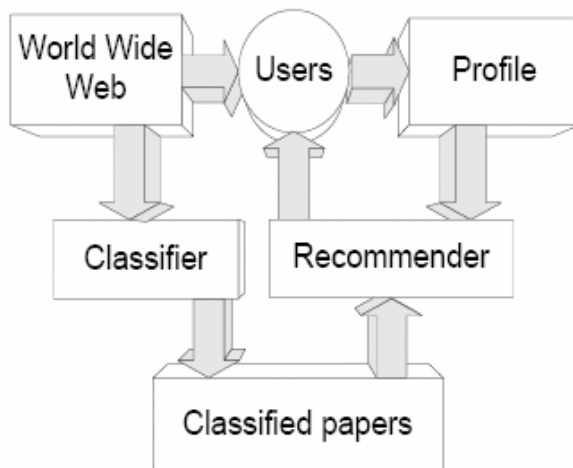


Figure 6: The Quickstep system

Each day a set of recommendations is computed, based on correlations between user interest profiles and classified paper topics. Any feedback offered on these recommendations is recorded when the user looks at them.

Users can provide new examples of topics and correct paper classifications where appropriate. In this way the training set improves over time as more feedback is elicited from the users.

Model

Quickstep is a hybrid recommendation system, combining both content-based and collaborative filtering techniques. Since both web pages and user interests are dynamic in nature, catalogues, rule-bases and static user profiles would quickly become out of date. The recommender system approach is well suited to the problem, since it works with observed dynamic behaviour and will adjust to the changing interests of its users on a daily basis.

Asking users to provide explicit feedback on browsed papers would be very intrusive and interfere with the normal workflow of the researchers. Unobtrusive monitoring of web browsing was thus chosen to acquire positive examples of user interest. Optional explicit feedback is elicited when recommendations are made, both on topic interest and paper classification accuracy. However, since users choose when to review their recommendations, asking for explicit feedback at this point will not interrupt their normal workflow as they have chosen to spend time with the system anyway. Since many users will be using the system at once it is sensible to share user feedback and maintain a common pool of example papers provided by the users.

An initial training set of example papers is provided for each topic in the ontology to bootstrap the classifier. This training set is then allowed to grow over time as users provide examples of their own or correct the bootstrap examples. This labelled training set is well suited to supervised learning techniques, which require a prior set of classes on which to base a classification. A term vector representation is used to represent research papers, a common approach in machine-learning. A term vector is a list of word weights, derived in this case from the frequency that words appear within the research paper main text.

Research papers are represented as term vectors, with term frequency / total number of terms used for a terms weight.

The profiling algorithm performs correlation between the paper topic classifications and user browsing logs. Whenever a research paper is browsed that has a classified topic, it accumulates an interest score for that topic.

Recommendations are formulated from a correlation between the user's current topics of interest and the papers classified as belonging to those topics. A paper is only recommended if it does not appear in the user's browsed URL log, ensuring that all recommendations have not been seen before. For each user, the top three interesting topics are selected with 10 recommendations made in total (making a 4/3/3 split of recommendations). The top three interesting topics can come from the ontologies non-leaf nodes as well as leaf nodes. Papers are ranked in order of the recommendation confidence before being presented to the user. Recommendation confidence is calculated in the following way:

$$\text{Recommendation confidence} = \text{classification confidence} * \text{topic interest value}$$

Ontology

The research paper topic ontology is based on the dmoz [dmoz] taxonomy of computer science topics, and was chosen since it is freely available, has had a significant amount of development and is regularly maintained. It is an is-a hierarchy of paper topics, up to 4 levels deep (e.g. an "interface agents" paper is-a "agents" paper). Pre-trial interviews generated some additional topics that were added to the dmoz taxonomy, and a review by two domain experts validated the ontology for correctness before use in the trials. Figure 7 shows a section of the research paper topic ontology.

A perfect ontology would tailor the granularity of topic size exactly to each user. If taken to an extreme this would require some sort of personal ontology per user and a mapping to shared example papers. To create such an ontology a full survey would be needed to elicit the types of research each user was interested in, and obtain some example papers for it.

Advantages

Quickstep and Foxtrot use a novel, ontological approach to user profiling within recommender systems. It is novel since most other recommender systems use a binary class approach, using "interesting" and "not interesting" classes specific to each user.

The ontological profile representation used by both Foxtrot and Quickstep represents each profile interest type as an ontological class. The relationships within the ontology are used to infer new knowledge about interests that could not be observed directly. Profiles are also visualized, using the ontological terms that are understandable to users, and profile feedback elicited.

Lastly, the recommender cold-start problem is addressed by applying a knowledge-base, based on publications and personnel data, which uses the same ontology to bootstrap new user profiles.

3.2.2 *Decentralized recommender*

Recently, novel distributed infrastructures are emerging, e.g., peer-to-peer and ad-hoc networks, the Semantic Web, the Grid, etc., and supersede classical client/server approaches in many respects. These infrastructures could likewise benefit from recommender system services, leading to a paradigm shift towards decentralized recommender systems.

[Ziegler, 05] investigates the challenges that decentralized recommender systems bring up and proposes diverse techniques in order to cope with those particular issues. The spectrum of methods proposed ranges from the employment of product classification taxonomies as powerful background knowledge, alleviating the sparsity problem, to trust propagation mechanisms designed to address the scalability issue. Empirical investigations on the correlation of interpersonal trust and interest similarity provide the component glue that melds these results together and renders the eventual creation of a decentralized recommender framework feasible.

Problem

One of the primary issues that recommender systems are facing is rating sparsity, particularly pronounced for decentralized scenarios. Hence, high quality product suggestions are only feasible when information density is high, i.e. large numbers of users voting for small numbers of items and issuing large numbers of explicit ratings each. Small-sized, decentralized and open Web communities, where ratings are mainly derived implicitly from user behavior and interaction patterns, poorly qualify for blessings provided by recommender systems.

Approach

The approach intends to alleviate the information sparsity issue by exploiting product classification taxonomies as powerful background knowledge. Semantic product classification corpora for diverse fields are becoming increasingly popular, facilitating smooth interaction across company boundaries and fostering meaningful information exchange.

The proposed taxonomy-based similarity metric, making inferences from hierarchical relationships between classification topics, represents the core of the hybrid filtering framework to compensate for sparsity. Quality recommendations become feasible in communities suffering from low information density, too.

Following the “collaboration via content” paradigm, the approach computes content-based user profiles which are then used to discover like-minded peers. Once the active agent’s neighborhood of most similar peers has been formed, the recommender focuses on products rated by those neighbors and generates top-N recommendation lists. The rank assigned to a product b depends on two factors. First, the similarity weight of neighbors voting for b , and, second, b ’s content description with respect to the active user’s interest profile. Hence the hybrid nature of the approach.

Another very important characteristic of the approach is social-network based trust propagation model. Trust metrics compute quantitative estimates of how much trust an agent a_i should accord to his peer a_j , taking into account trust ratings from other persons on the network. These metrics should also act “deliberately”, not overly awarding trust to persons or agents whose trustworthiness is questionable.

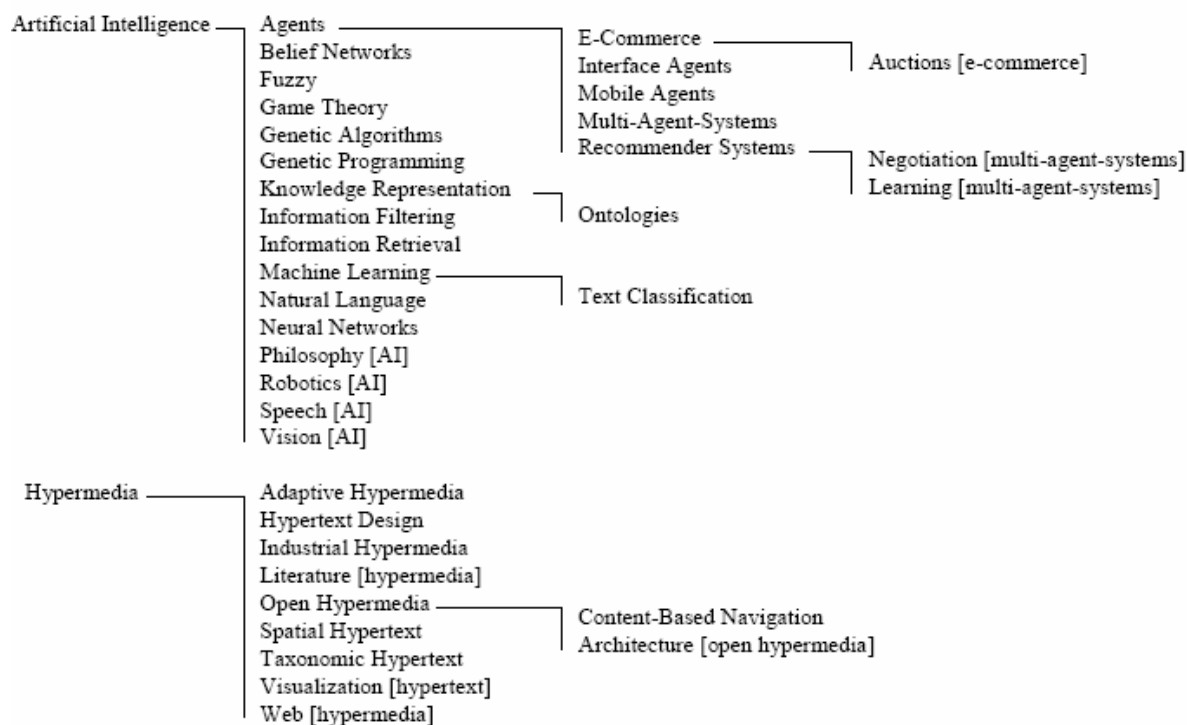


Figure 7: Section of the Quickstep research paper ontology

Advantages

Taxonomy-driven filtering

Taxonomy-driven filtering relies upon very large product classification taxonomies as powerful background knowledge to render recommendation computations feasible in environments where sparsity prevails. The approach features two important contributions:

- Interest profile assembly and similarity measurement.

The method for assembling profiles based on topic interests rather than product ratings constitutes the nucleus of taxonomy-driven filtering. Hereby, hierarchical relationships between topics provide an essential inference mechanism. The resulting profiles serve as means for computing user-user, user-product, and product-product similarity.

- Recommender framework for sparse data.

The approach embedded the taxonomy-driven profiling and similarity measuring technique into an adaptive framework for computing recommendations. The respective system, devised for centralized data storage and computation, also introduces a new product-user relevance predictor. Empirical evidence, featuring both online and offline evaluations for two different datasets, shows our approach's superior performance over common benchmark systems for sparse product-rating matrices.

Trust propagation based on spreading activation models

Trust metrics are nowadays gaining momentum through the emergence of decentralized infrastructures such as the Semantic Web. The approach proposes a scalable, attack-resistant trust metric based upon spreading activation models which is able to compute neighbourhoods of most-trustworthy peers.

4 RECOMMENDER SYSTEMS IN SOFTWARE ENGINEERING

As a knowledge intensive domain, the idea of supporting Software Engineering tasks with expert and recommendation systems is quite obvious. Given the existence of large amount of reusable source code in both corporate repositories and the internet, there is a large potential to improve the efficiency by assisting developers.

Current approaches are especially focussing on two scenarios:

- Recommending source code/methods to reuse from a given corpus of existing sources
- Recommending related information from the project memory to speed up maintenance tasks.

In this section, we will describe a number of systems that support those scenarios. At the end, we will provide an overview and address issues for further research.

4.1 CodeBroker

CodeBroker [Ye, 02/05] is a system that aims to foster software reuse by actively recommending source code (i.e. methods) that is suitable in the given context. The system consists of a client (“interface agent”) and a backend. The client, which is prototypically implemented inside the Emacs editor, queries the backend and displays suitable results in a special area of the editor (“reusable component information”).

The user context is composed of three parts. One part is the immediate programming task, which is the basis for getting results from the backend. It is extracted implicitly from the comments and the signature of the method the developer is writing currently. Additionally, CodeBroker maintains a “discourse model” of the developers’ interaction with the system. This model stores methods which were explicitly ruled out by the user. It is maintained for one session only. In opposite, the so-called “user model” is maintained permanently. It captures such methods, which the developer already knows and thus does not need to be recommended. The user model can be created implicitly (e.g. by analyzing source code a developer has written) but may also changed explicitly. Both, the discourse model and the user model are used by the interface agent, to filter out results that were returned from the backend.

Recommendations in CodeBroker are triggered continuously as soon as the interface agent is activated. As described, queries are based upon comments and the signature of the current method. Method comments are matched by information retrieval algorithms (LSI) against an index created from JavaDoc comments. Method signatures are matched by an algorithm based on the work of [Zar, 95]. Before presentation, the interface agent filters the results based on the discourse model and the user model. In the interface, users may refine the query by excluding components from certain packages.

4.2 Dhruv

The Dhruv system [Ank, 05/06] aims to assist the software maintenance / bug resolution process, by recommending relevant information during bug inspection. Therefore, Dhruv is integrated in a web-based bug tracking system and displays recommendations in a special

sidebar. Such recommendations may involve source code files, mailinglist discussions or similar bug reports.

Dhruv does not operate on a special user profile. The context for recommendation is always the bug report for which related information is retrieved. This information is included automatically when creating the report page. The recommendation corpus data is created in two steps. First, metadata is extracted from source code artifacts, mailinglists and existing bug reports. Afterwards, algorithms are employed in order to infer relationships among metadata items. Based on the identification of named entities, several heuristics are employed to infer the actual role of the entity in the context of the analyzed file. The metadata as well as possible relations are modeled in ontologies. Thus after analysis, the various metadata entities form an interconnected semantically described graph structure.

Based on this graph structure, recommendations are drawn by relational similarity. The relations in the ontology have weights assigned, which are used as part of the similarity calculation.

4.3 Hipikat

Hipikat [Cub, 05] strives to support developers (especially newcomers to a project) working on maintenance tasks. Therefore, Hipikat builds a group memory consisting of four types of artifacts (source code, E-Mail discussions, change tasks and documents). A developer may use the Hipikat client, which is implemented as an Eclipse plug-in to query artifacts inside Eclipse for related items. The Hipikat backend then returns a list of source code/E-Mail discussions or bug reports which is related to the developers query.

Hipikat does not maintain any kind of user profile. Instead, recommendations are based on explicit queries, which must include an artifact reference, for which recommendations are requested. The queries can be further constrained to certain pre-defined relations. For example, a bug may be queried for other similar bugs, or for code which fixes the bug. The selection of appropriate results is based on similarity calculations which operate on the relations between the artifacts.

The queried project memory is built automatically from existing artifacts. The relations are inferred by five different, manually implemented heuristics. Examples are a log matcher, which tries to identify bug report IDs in source code documentation using regular expressions, or an activity matcher, which compares check-in times of source code changes with the closing time of bug reports. If the appropriate values exceed pre-defined thresholds, a relation between the artifacts is created automatically.

4.4 Mylyn

While most other systems in this section strive to filter relevant source code from large repositories, mylyn (formerly called “mylar”) [Ker, 05] targets to restrict the user interface of an IDE. The core idea is that not all classes in large software projects are relevant for working on a given (maintenance) task. Thus, mylyn tries to identify classes which are relevant during a task, and to hide or blur those classes deemed not relevant (“task-focused UI”). Mylyn is prototypically implemented as a plugin for the Eclipse IDE.

The notion of a task is fundamental for mylyn. It is based on the assumption that developers are sequentially working on limited tasks (e.g. fixing a bug) which affect only a subset of

source code files. To support this notion of “task-oriented programming”, mylyn provides an integration of various issue tracking systems. Developers have to indicate the task they are working on. Thus, a task is also the primary context information for recommendations.

During each task, a “degree-of-interest” model is maintained for each source code file. The degree-of-interest is a real value, which is influenced by the developers’ interaction with the file and can decay over time. This is complemented by a “degree-of-separation” model, which represents relations among the source code files. The combined information yields a real value which is interpreted to visually indicate task-related files inside the IDE. Therefore, the display color of the filenames in the Eclipse workspace view is changed from light grey to the default black. The more black the filename, the more recommended is the file for the given task.

4.5 RASCAL

The overall approach of RASCAL [McC, 05] is quite similar to the CodeBroker system. It is also motivated by the availability of large repositories of code, which can not be overseen by developers. A special motivation is drawn from the rising popularity of agile software development practices, which favour the production of source code against extensive documentation. The RASCAL system consists of a client for the Eclipse IDE and a server backend.

The client tries to predict the next method the developer would use, by analyzing the current class and comparing it to similar classes. Thus, the user context in RASCAL solely consists of information implicitly extracted from the current class a developer is editing. More concretely, RASCAL extracts the total number of calls to (external) methods inside a class. Following the same principle, the backend corpus is built by analyzing existing source code.

Recommendations in RASCAL must be triggered manually. Upon that, the current class is matched against the information in the backend. In a first step, the k -nearest neighbours of the current class are identified. Afterwards, an average order of the methods is created. Finally, those methods are ranked high, which most often occur after the last method in the queries class. Experimentation shows, that RASCAL is able to deliver meaningful results, if half of a component is written and at least five suitable elements are on the recommendation list.

4.6 Strathcona

Strathcona [Hol, 05] is another Eclipse plugin, which aims to recommend source code examples that are relevant for the current development task. The main application scenario is the usage of third-party libraries/APIs. At the backend, Strathcona extracts facts from given source code and stores it into a relational database.

A query has to be triggered by the user by selecting a code fragment. Additionally, the Strathcona client automatically extracts what the authors call “structural context”. This includes the method signature, declaring types (plus super-types), field names, referenced types and fields of the current method. This information is sent to the server, where four different structural similarity heuristics (based on inheritance, method calls, usage and field references) are used to match relevant recommendations. All heuristics are implemented as SQL queries on top of the database. After executing the queries, the results from the four heuristics are merged and the top 20 are selected and sent back to the client.

4.7 Summary

Tool	Goal	Architecture	Recommended item	User profile/ Context	Trigger / Need inference	Corpus	Matching algorithm
Hipikat	Assist newcomers and maintainers in software projects	Client (Eclipse), Server	Files, messages, issues or documents	Artifact which is subject of the query (explicit)	Manual query	CVS, Bug reports, E-Mails, Documents	Relational similarity (content-based)
CodeBroker	Foster source code reuse by suggesting methods	Client (Emacs), Server	Source code (method)	Current method comments and signature (implicit), discourse model (explicit), user model (Implicit, allowing explicit changes)	Automatic query	JavaDoc and source code	Conceptual similarity (LSI) for textual comments and constraint similarity (signature matching) (Content-based)
Dhruv	Speed up bug resolution by recommending related software objects and artifacts	Integrated in server-side web interface	Code files, Discussions, Bug reports	Current bug report (implicit)	Automatically	Community data (Code, E-Mails, Bug reports)	Relational similarity (weighted) (Content-based)
mylyn	Provide task-focussed UI	Eclipse plugin	Source files	Task-based user interaction on files (explicit/implicit)	Automatically	Files in project workspace	Based on clicks and class relations (Content-based)
RASCAL	Predict next method a user might insert	Client (Eclipse), Server	Source code (method)	Analysis of current class (implicit)	Manual query	Swing-based applications from SourceForge	Hybrid
Strathcona	Recommend example code for third-party APIs, based on "structural context"	Client (Eclipse), Server	Example code	"Structural context" (Implicit)	Manual query	Source code	SQL-queries (Content-based)

Table 3: Overview of Software Engineering recommender systems

Table 3 shows an extensive overview of this chapter. The presented systems indicate that there is tremendous interest in recommendation functionality for Software Engineering tasks. However, with the exception of mylyn system, which is part of the Eclipse Europa distribution, such functionality has not yet been included into state-of-the-art development environments. Additionally, several points for improvement can be identified.

Architecture

Most of the presented systems are (1) using a client/server architectural style and are (2) closed systems which operate on one server exclusively. Thus, the amount of included data is limited by capacity and management effort on the server side. A P2P-based approach, like the one envisioned for the TEAM system, could make more data available without introducing performance problems. Furthermore, a decentralized approach could make additional knowledge accessible, which developers would not contribute to a central repository. None of the presented approaches uses a true collaborative filtering approach,

which leverages the experience of the developer community. Only mylyn is anticipating knowledge exchange across developers, by allowing exchanging the degree-of-interest model for a given task.

Flexibility

With the exception of the Dhruv system, all the described solutions are working with traditional knowledge representations and hard-coded heuristics. However, a more flexible knowledge representation, e.g. based on Semantic Web technologies, might not only improve the possibilities to integrate and share additional data, but might also help to make system behaviour more transparent – e.g. by providing explanations for recommended items.

Subject of recommendation

The presented approaches are either focussing on recommending methods to use (CodeBroker, RASCAL, Strathcona) or development artifacts (Dhruv, Hipikat). Thus, the approaches are limited to explicit knowledge, which already exists. In TEAM, the context observer component strives to capture problem solving patterns, which are usually not explicitly documented by developers. We consider this kind of information very useful for developers.

Context/Trigger

When compared to traditional recommender systems, the user profiles created by the described systems are rather simplistic. Thus, recommendations are either triggered automatically in a continuous way, or have to be requested by users. For the TEAM system, we strive to create a richer user context which allows identifying certain problem situations (e.g. a compiler error) that allow more focused recommendations.

5 CONCLUSION

The creation of large-scale information repositories, such as the internet or corporate intranets has brought a large amount of information in the reach of users. However, due to constraints in time and mental capacity, it is hard for humans to find information suitable for solving a given task. Thus, recommendation systems, providing an intelligent “information push” functionality which suggests information for a given task context of users is highly desirable.

In section 2, we have shown that the field of recommender systems has made considerable progress in recent years and were especially successful in large scale commercial applications such as product or movie recommendation. Applications in other domains have been limited by the sparsity of available relevant information. Ontology-based recommender systems, as described in section 3, can be helpful in such scenarios – e.g. by categorizing users and/or items into taxonomies.

Especially the field of Software Engineering is well-suited for the application of recommender systems. Software Engineering is a highly information-intensive activity and large repositories are available which include highly structured artifacts such as source code or bug reports. Several research efforts as presented in section 4 indicate a high interest for recommendation tools. However, our analysis has shown that current systems have a number of limitations such as their architecture, flexibility and the usage of implicit context information. Several components of the TEAM architecture, such as P2P network communication, ontology-based metadata storage and context information can be leveraged to address improvements on these issues.

REFERENCES

- [Ado, 01] G. Adomavicius and A. Tuzhilin, "Expert-Driven Validation of Rule-Based User Models in Personalization Applications," *Data Mining and Knowledge Discovery*, vol. 5, nos. 1 and 2, pp. 33-58, 2001
- [Ank, 05] Ankolekar, A.: *Towards a Semantic Web of Community, Content and Interactions*. Ph.D. Thesis September, CMU-HCII-05-103 (2005)
- [Ank, 06] Ankolekar, A., Sycara, K., Herbsleb, J., Kraut, R., Welty, C.: *Supporting Online Problemsolving Communities with the Semantic Web*. In *Proc. of the 15th Int. Conference on World Wide Web*, Edinburgh, Scotland, (2006) 575-584.
- [Arm, 01] J.S. Armstrong, *Principles of Forecasting—A Handbook for Researchers and Practitioners*. Kluwer Academic, 2001.
- [Bal, 97] M. Balabanovic and Y. Shoham, "Fab: Content-Based, Collaborative Recommendation," *Comm. ACM*, vol. 40, no. 3, pp. 66-72, 1997.
- [Aussenac et al, 04] N., Aussenac-Gilles, J., Mothe, *Ontologies as Background Knowledge to Explore Document Collections*, *RIAO*, pp 129-142, 2004.
- [Bas, 98] C. Basu, H. Hirsh, and W. Cohen, "Recommendation as Classification: Using Social and Content-Based Information in Recommendation," *Recommender Systems. Papers from 1998 Workshop*, Technical Report WS-98-08, AAAI Press 1998
- [Bel, 92] N. Belkin and B. Croft, "Information Filtering and Information Retrieval," *Comm. ACM*, vol. 35, no. 12, pp. 29-37, 1992.
- [Bil, 00] D. Billsus and M. Pazzani, "User Modeling for Adaptive News Access," *User Modeling and User-Adapted Interaction*, vol. 10, nos. 2-3, pp. 147-180, 2000.
- [Bre, 98] J.S. Breese, D. Heckerman, and C. Kadie, "Empirical Analysis of Predictive Algorithms for Collaborative Filtering," *Proc. 14th Conf. Uncertainty in Artificial Intelligence*, July 1998.
- [Bri, 98] Brin, S. and Page, L. 1998. *The Anatomy of a Large-Scale Hypertextual Web Search Engine*. Dept. of Computer Science, Stanford Univ.
- [Bur, 00] R. Burke, "Knowledge-Based Recommender Systems," *Encyclopedia of Library and Information Systems*, A. Kent, ed., vol. 69, Supplement 32, Marcel Dekker, 2000.
- [Cla 99] M. Claypool, A. Gokhale, T. Miranda, P. Murnikov, D. Netes, and M. Sartin, "Combining Content-Based and Collaborative Filters in an Online Newspaper," *Proc. ACM SIGIR '99 Workshop Recommender Systems: Algorithms and Evaluation*, Aug. 1999.
- [Cor, 00] C. Cortes, K. Fisher, D. Pregibon, A. Rogers, and F. Smith, "Hancock: A Language for Extracting Signatures from Data Streams," *Proc. Sixth ACM SIGKDD Int'l Conf. Knowledge Discovery and Data Mining*, 2000.
- [Cro, 01] W.B. Croft, S. Cronen-Townsend, V. Lavrenko, *Relevance Feedback and Personalization: A Language Modeling Perspective*, *DELOS Workshop: Personalisation and Recommender Systems in Digital Libraries*, 2001.

- [Cub, 05] Cubranic, D.; Murphy, G.C.; Singer, J.; Booth, K.S., Hipikat: a project memory for software development, *Software Engineering, IEEE Transactions on*, vol.31, no.6, pp. 446-465, June 2005
- [Dwy, 89] F.R. Dwyer, "Customer Lifetime Valuation to Support Marketing Decision Making," *J. Direct Marketing*, vol. 3, no. 4, 1989.
- [Fin et al, 05] Finin, T., Mayfield, J., Joshi, A., Cost, R. S., and Fink, C. 2005. Information Retrieval and the Semantic Web. In *Proceedings of the Proceedings of the 38th Annual Hawaii international Conference on System Sciences (Hicss'05) - Track 4 - Volume 04 (January 03 - 06, 2005)*. HICSS. IEEE Computer Society, Washington, DC, 113.1. DOI=<http://dx.doi.org/10.1109/HICSS.2005.319>
- [Fluit et al, 05] Fluit C, Sabou M, Van Harmelen F (2002) *Ontology-based information visualisation, Visualising the Semantic Web*, Springer-Verlag.
- [Gar, 79] Garfield, E. 1979. *Citation Indexing: Its Theory and Application in Science, Technology, and Humanities*. New York: Wiley-Interscience.
- [Gol, 92] Goldberg, D., Nichols, D., Oki, B. M, and Terry, D. 1992. Using Collaborative Filtering to Weave an Information Tapestry. *Communication of the ACM*, December, 35(12): 61-70.
- [Hea, 96] M.A. Hearst and J.O. Pedersen. Reexamining the Cluster Hypothesis: Scatter/Gather on Retrieval Results. In *Proc. of Int. ACM SIGIR Conf. on Research and Development in Information Retrieval*, pages 76–84. ACM, 1996.
- [Her, 01] J.L. Herlocker and J.A. Konstan, "Content-Independent Task- Focused Recommendation," *IEEE Internet Computing*, vol. 5, no. 6, pp. 40-47, Nov./Dec. 2001
- [Her, 99] J.L. Herlocker, J.A. Konstan, A. Borchers, and J. Riedl, "An Algorithmic Framework for Performing Collaborative Filtering," *Proc. 22nd Ann. Int'l ACM SIGIR Conf. Research and Development in Information Retrieval (SIGIR '99)*, 1999.
- [Hil, 92] Hill, W.C., Hollan, J. D., Wrobelwski, D. and McCandless, T. 1992. Read Wear and Edit Wear. In: *Proceedings of ACM Conference on Human Factors in Computing Systems, CHI '92*: 3-9.
- [Hol, 05] R. Holmes & G. C. Murphy, Using structural context to recommend source code examples, *ICSE 2005*, pp:117-125, 2005
- [Hotho et al 03] A. Hotho, S. Staab, G. Stumme: WordNet improves text document clustering In *Proc. of the SIGIR 2003 Semantic Web Workshop*. 2003.
- [Jin, 03] R. Jin, L. Si, and C. Zhai, "Preference-Based Graphic Models for Collaborative Filtering," *Proc. 19th Conf. Uncertainty in Artificial Intelligence (UAI 2003)*, Aug. 2003
- [Kar, 94] Karlgren, J. 1994. *Newsgroup Clustering Based on User Behavior: A Recommendation Algebra*. Technical and Research Reports from SICS, T94-01.
- [Ker, 05] M. Kersten & G. C. Murphy: Mylar: a degree-of-interest model for IDEs. *AOSD 2005*: 159-168
- [Kon, 97] Konstan, J. A., Miller, B. N., Maltz, D., Herlocker, J. L., Gordon, L. R., and Riedl, J. 1997. GroupLens: Applying Collaborative Filtering to Usenet News. *Communications of the ACM*, March, 40(3): 77-87.

- [Li, 04] J. Li and O.R. Zaïane, "Combining Usage, Content, and Structure Data to Improve Web Site Recommendation," Proc. Fifth Int'l Conf. Electronic Commerce and Web Technologies (EC-Web '04), pp. 305-315, 2004.
- [Lilien et al, 92] G.L. Lilien, P. Kotler, and K.S. Moorthy, Marketing Models. Prentice Hall, 1992.
- [Lopez et al, 05] V. Lopez, M. Pasin, E. Motta: AquaLog: An Ontology-Portable Question Answering System for the Semantic Web. ESWC 2005: 546-562
- [Man, 95] H. Mannila, H. Toivonen, and A.I. Verkamo, "Discovering Frequent Episodes in Sequences," Proc. First Int'l Conf. Knowledge Discovery and Data Mining (KDD-95), 1995.
- [Mae, 03] A. Maedche, S. Staab, N. Stojanovic, R. Studer, Y. Sure, SEMantic portAL: The SEAL Approach, Spinning the Semantic Web: Bringing the World Wide Web to Its Full Potential, MIT Press 2003, ISBN 0-262-06232-1, pp. 317-359, 2003.
- [McC, 05] McCarey, F., Cinnéide, M. Ó., and Kushmerick, N. 2005. Rascal: A Recommender Agent for Agile Reuse. Artif. Intell. Rev. 24, 3-4 (Nov. 2005), 253-276.
- [Middleton, 03] S. E. Middleton, (2003) Capturing knowledge of user preferences with recommender systems. PhD, ECS, University of Southampton.
- [Moo, 99] R.J. Mooney and L. Roy, "Content-Based Book Recommending Using Learning for Text Categorization," Proc. ACM SIGIR '99 Workshop Recommender Systems: Algorithms and Evaluation, 1999.
- [Mur et al, 03] B.P.S. Murthi and S. Sarkar, "The Role of the Management Sciences in Research on Personalization," Management Science, vol. 49, no. 10, pp. 1344-1362, 2003.
- [Navigli et al 05] Navigli, R. and Velardi, P. 2005. Structural Semantic Interconnections: A Knowledge- Based Approach to Word Sense Disambiguation. IEEE Trans. Pattern Anal. Mach. Intell. 27, 7 (Jul. 2005), 1075-1086. DOI= <http://dx.doi.org/10.1109/TPAMI.2005.149>
- [Nic, 97] Nichols, D. M. 1997. Implicit Ratings and Filtering. In Proceedings of the 5th DELOS Workshop on Filtering and Collaborative Filtering, 10-12. Budapest, Hungary, ERCIM.
- [Oard, 97] Oard, D. W. 1997. The State of the Art in Text Filtering. User Modeling and User-Adapted Interaction, 7(3): 141-178
- [Paz, 97] M. Pazzani and D. Billsus, "Learning and Revising User Profiles: The Identification of Interesting Web Sites," Machine Learning, vol. 27, pp. 313-331, 1997.
- [Paz, 99] M. Pazzani, "A Framework for Collaborative, Content-Based, and Demographic Filtering, Artificial Intelligence Rev., pp. 393-408, Dec. 1999.
- [Pen, 99] D.M. Pennock and E. Horvitz, "Collaborative Filtering by Personality Diagnosis: A Hybrid Memory And Model-Based Approach," Proc. Int'l Joint Conf. Artificial Intelligence Machine Learning for Information Filtering, Aug. 1999.
- [Pow, 81] M.J.D. Powell, Approximation Theory and Methods. Cambridge Univ. Press, 1981.
- [Ras, 02] A.M. Rashid, I. Albert, D. Cosley, S.K. Lam, S.M. McNee, J.A. Konstan, and J. Riedl, "Getting to Know You: Learning New User Preferences in Recommender Systems," Proc. Int'l Conf. Intelligent User Interfaces, 2002.

- [Ric, 79] E. Rich, "User Modeling via Stereotypes," *Cognitive Science*, vol. 3, no. 4, pp. 329-354, 1979.
- [Roc, 71] J.J. Rocchio, "Relevance Feedback in Information Retrieval," *SMART Retrieval System—Experiments in Automatic Document Processing*, G. Salton, ed., chapter 14, Prentice Hall, 1971.
- [Roo, 71] Rocchio, J.J. Relevance feedback in information retrieval, in the *SMART Retrieval System -- Experiments in Automatic Document Processing*, Englewood Cliffs, NJ, 1971. Prentice Hall, Inc. pp. 313-323
- [Ruk, 97] Rucker, J. and Polanco, M. J. 1997. Personalized Navigation for the Web. *Communications of the ACM*, March, 40(3): 73-89.
- [Rut, 03] I.Ruthven, M.Lalmas, C.J. van Rijsbergen, Incorporating user search behaviour into relevance feedback. *Journal of the American Society for Information Science and Technology*. 54. 6. pp528-548. 2003
- [Sal, 89]G. Salton, *Automatic Text Processing*. Addison-Wesley, 1989.
- [Sch, 01] J.B. Schafer, J.A. Konstan and J. Riedl (2001). *E-Commerce Recommendation Applications*. *Data Mining and Knowledge Discovery* 5(1/2):115-153.
- [Sha, 95] U. Shardanand and P. Maes, "Social Information Filtering: Algorithms for Automating 'Word of Mouth'," *Proc. Conf. Human Factors in Computing Systems*, 1995.
- [She, 93] B. Sheth and P. Maes, "Evolving Agents for Personalized Information Filtering," *Proc. Ninth IEEE Conf. Artificial Intelligence for Applications*, 1993
- [Sob, 99] I. Soboroff and C. Nicholas, "Combining Content and Collaboration in Text Filtering," *Proc. Int'l Joint Conf. Artificial Intelligence Workshop: Machine Learning for Information Filtering*, Aug. 1999.
- [Sta, 95] R.B. Statnikov and J.B. Matusov, *Multicriteria Optimization and Engineering*. Chapman & Hall, 1995.
- [Ste, 93] Stevens, C. 1993. *Knowledge-Based Assistance for Accessing Large, Poorly Structured Information Spaces*. Ph.D. dissertation, Dept. of Computer Science, Univ. of Colorado, Boulder.
- [Sto, 03] N. Stojanovic, R. Studer, L. Stojanovic, An Approach for the Ranking of Query Results in the Semantic Web, In *Proceedings of the International Semantic Web Conference (ISWC 2003)*, Sanibel Island, FL, USA, pp. 500-516, 2003.
- [Sto, 05] Stojanovic, N., "Method and Tools for Ontology-based Cooperative Query Answering", PhD thesis, University of Karlsruhe, Germany, 2005
- [Yan, 01] Y. Yang and B. Padmanabhan, "On Evaluating Online Personalization," *Proc. Workshop Information Technology and Systems*, pp. 35-41, Dec. 2001.
- [Ye, 02] Y. Ye & G. Fischer, Supporting reuse by delivering task-relevant and personalized information, *Proceedings of the 24th International Conference on Software Engineering (ICSE'02)*, Orlando, FL, 2002.
- [Ye, 05] Y. Ye & G. Fischer, Reuse-Conducive Development Environments. *Autom. Softw. Eng.*, Vol. 12, No. 2, pp. 199-235, 2005

- [Yu, 02] K. Yu, X. Xu, J. Tao, M. Ester, and H.-P. Kriegel, "Instance Selection Techniques for Memory-Based Collaborative Filtering," Proc. Second SIAM Int'l Conf. Data Mining (SDM '02), 2002.
- [Ziegler, 05] C.N. Ziegler, Towards decentralized recommender systems, University of Freiburg, 2005
- [Zha, 02] Y. Zhang, J. Callan, and T. Minka, "Novelty and Redundancy Detection in Adaptive Filtering," Proc. 25th Ann. Int'l ACM SIGIR Conf., pp. 81-88, 2002.
- [Van der Vet, 98] P.E. van der Vet and B. van Bakel. Combining Linguistic and Knowledge-based Engineering for Information Retrieval. In Proc, of TWLT14. <http://citeseer.ist.psu.edu/vandervet98combining.html>
- [Wang, 03] B. Wang, R.I. McKay, H.A. Abbass M. Barlow, (2003): A Comparative Study for Domain Ontology Guided Feature Extraction. In: Proceedings of ACSC-2003. Australian Computer Society.
- [Zar, 95] Zaremski, A. M. and Wing, J. M. 1995. Signature matching: a tool for using software libraries. ACM Trans. Softw. Eng. Methodol. 4, 2 (Apr. 1995), 146-170.