Joining Collaborative and Content-based Filtering

Patrick Baudisch
Integrated Publication and Information Systems Institute IPSI
German National Research Center for Information Technology GMD
64293 Darmstadt, Germany
+49-6151-869-854
baudisch@gmd.de

ABSTRACT
Different authors have proposed combining content-based and collaborative attributes in a single table. In this article, we try out a different approach. We propose not to *merge* the two tables, but to *join* them as if they were tables in a relational database. As a result, we get several new application cases and a system architecture that supports the formulation of universal queries.

Keywords
Recommender system, collaborative filtering, content-based enhancements, relational database, join, SQL, user interface

RELATED WORK
During the past year, a number of authors and system designers have experimented with enhancing collaborative systems (also called recommender systems) with content-based extensions [1, 2]. While purely collaborative systems are based on a single *user × object* table, enhanced systems introduce formal descriptors into the system. These enhancements are based on the observation that the content-based relation *matches(descriptor, object)* is compatible with the collaborative relation *likes(user, object)*. Consequently, both relations can be stored in a single table as shown in Table 1. Content-based attributes are added as additional rows into the table, so that users and content-based descriptors are actually mixed. In other words, content-based descriptors are treated as additional users.

	Object1	**Object2**	**Λ**
User1	R_{U11}	R_{U12}	**Λ**
User2	R_{U21}	R_{U22}	**Λ**
M	**M**	**M**	O
Descriptor1	R_{D11}	R_{D12}	**Λ**
Descriptor2	R_{D21}	R_{D22}	**Λ**
M	**M**	**M**	O

Table 1: Introducing formal descriptors into collaborative systems

Which kinds of content-based descriptors are actually used depends on the application domain. The GroupLens *Filterbots* [1] help to filter newsgroup articles. Experiments have shown, for example, that the formal descriptor "high amount of cited text" is a good predictor for poor ratings in the alt.rec.humor newsgroup. Other Filterbots count the number of typographical errors or simply measure message lengths.

Greening [2] uses so-called *Archetypes* in his recommender system for music, videos, and books. Archetypes implement the same concept as Filterbots, i.e. they are "hypothetical users" that "like" all objects having a specific property. In this application domain, properties are product categories, artists, and listening types.

The primary purpose of Filterbots and Archetypes is to help overcome the so-called first-rater problem. First-rater problem means that objects newly introduced into the system have not been rated by any users and can therefore not be recommended. Due to the absence of recommendations, users tend to not be interested in regarding these new objects. This in turn has the consequence that the newly added objects remain in their state of not being recommendable. Whereas real users are affected by the first-rater problem, Filterbots and Archetypes are always "motivated" to rate new objects, i.e. to be early raters. The Filterbots/Archetypes approach is therefore a promising approach to solving the first-rater problem.

A DIFFERENT APPROACH
In this article, we will try out a different way of introducing content-based descriptors into collaborative systems. Similar to the systems described above, we will combine the two tables containing users and formal descriptors. However, we will not do this by *merging* the two tables but instead by *joining* them. We mean *joining* in the sense used in relational databases.

The purpose of joining tables is only indirectly related to the first-rater problem. It is a way of integrating collaborative systems with content-based information systems and of generating new functionality, which is then directly available to users.

Some remarks on notation
When discussing individual concepts, we will look at them on a rather high level. Although implementation and performance "tricks" *are* an important issue, especially in large-scale recommender systems (and recommender systems have to be large scale to work effectively), we will look at the involved data structures as simple relations. To implement the described concepts, however, it will be

necessary and crucial to consider an efficient implementation.

In general, we will borrow a number of notations from database systems, i.e. SQL-like syntax and entity relationship diagrams. We are aware of the fact that the actual calculations done in information retrieval/information filtering and in collaborative systems are more complicated then the ones suggested by the diagrams. When reading these notations, be aware that the calculations behind these notations are different from processing SQL-joins on tables.

Finally, a remark on terminology. In information retrieval literature, the objects retrieved from information systems are usually referred to as documents. However, one has to admit that these documents can be as well multimedia objects, hypertext and so on. In this article, we will be generally speaking of *objects* unless we explicitly describe them as text documents.

Before actually joining tables, we will start by looking at the two tables individually.

Content-based processing

Information retrieval and information filtering systems perform all calculations on the single relation *descriptor* × *object*. As an example, objects can be text documents and descriptors can be keywords. Figure 1 shows the two involved entities and the single relation between them.

Figure 1: Entity relationship model of content-based information retrieval and filtering

By using this relation in the one or the other direction and by joining it with itself, the four relations shown in Table 2 can be computed. The table contains the names of typical applications based on the individual cases.

Descriptors (e.g. Keywords)	Associative thesaurus	Information retrieval/ filtering
Object	Indexing	Query by example
	Descriptors (e.g. Keywords)	**Object**

Table 2: Applications of content-based systems. Table rows are input; table columns are output.

The primary goal of information systems is to retrieve or filter objects. Therefore, the right column of Table 2 is usually in the focus of interest. The two table cells in the left column, i.e. thesauri and indexing, provide necessary or helpful methods for enhancing search.

When retrieving objects, queries can be made up either of formal descriptors or of objects. In the latter case, the entered documents are considered as (usually positive) examples and when queried, the system returns a set of similar objects (query by example). Since entering objects or object references manually has proved to be difficult, object-based queries are usually handled as an extension of the keyword-based retrieval case. Users first have to query the system using a keyword-based query. Then users pick positive or negative examples from the returned set of objects to refine the query (relevance feedback). This approach is a union of the two right cells in Table 2. To combine the two input data types keywords and documents in a single query, documents are replaced with the most expressive keywords used in these documents.

Collaborative processing

In collaborative systems, all calculations are based on the single relation *user* × *object*. Figure 2 shows the two involved entities and the single relation between them.

Figure 2: Entity relationship model of collaborative systems

Analogous to the content-based cases described above, this relation can be used to address the four different scenarios shown in Table 3.

User	Matchmaker	Active collaborative filtering
Object	Find experts	Automated collaborative filtering
	User	**Object**

Table 3: Applications of a collaborative system

Those cases that retrieve objects form the right column. The two fields represent two well-known versions of collaborative filtering. In passive or automated collaborative filtering (e.g. [4]), users fill their user profiles with ratings about objects, and then solely this profile is used to query the system. In the case of active collaborative filtering (e.g. [5]), users are addressed individually, either for sending them recommendations or for gathering recommendations from them.

The two fields in the left column return users instead of objects. They can therefore be interesting for their social aspects. A typical application is matchmaking.

Comparison

There are some basic differences between collaborative and content-based approaches. Users do know and under-

stand the meaning of keywords and other formal descriptors, while they do not necessarily know the meaning of collaborative descriptors (i.e. users and the interests they represent). Consequently, keywords can always be used to explain the objects they match. Names of users can only be used to describe the objects these users like if the respective users are known. This is usually the case if the community of users is very small or if the respective users are known for other reasons, e.g. because they are opinion leaders [3]. This restricts the applicability of (active) collaborative filtering.

JOINING THE TWO RELATIONS

If we provide a system with both content-based and collaborative information as separate relations, we get the entity-relationship model shown in Figure 3.

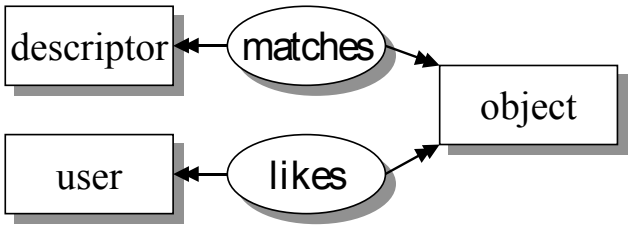


Figure 3: Entity-relationship model of combined system

Exploiting all possible joins of Figure 3 provides the functionality of a content-based system, as well as the functionality of a collaborative system as shown in Table 4. The thick lines show how uniting Table 2 and a 180°-rotated Table 3 generated this table.

There are new aspects about Table 4, compared to the two original tables forming it. Two new cells have been created. The *user→descriptor* cell turns user profiles into sets of formal descriptors. This function can be used to convert collaborative user profiles into textual descriptions. The opposite case, i.e. turning keywords into users, can be used to rapidly generate user profiles from formal descriptions or to find experts to a given topic.

Descriptor (e.g. Keyword)	Thesaurus	Information Retrieval/ Filtering	Profile import, find experts
Object	Indexing	Query by example	
		Automated collaborative filtering	Find experts
User	Profile export	Active collaborative filtering	Match Maker
	Descriptor (e.g. Keyword)	**Object**	**User**

Table 4: Possible queries in a combined system.

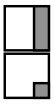
The fact that both sub-tables overlap at the center cell is not surprising. In the introduction, we already mentioned systems that make use of this fact by merging users and formal descriptors in a single table. In a combined system, there are actually three possibilities to infer object-object relationships. It can be done via descriptors (content-based), via users (collaborative), or both (e.g. using Filterbots).

USAGE SCENARIOS

Should we build a combined system and if so, which kind? Which user group would probably be interested in which functionality? We will look at some possible application scenarios. Icons on the left illustrate which cells of Table 4 are currently referred to.

Users of the actual recommender functionality are interested in retrieving objects. The retrieval process can be fed with content-based descriptors, a selection of relevant objects, a list of users known to have similar taste, as well as with a combination of all of these.

Marketing people are interested in obtaining information about users. They can query the system using keywords, objects (e.g. products), as well as with dummy users containing prototypical marketing-profiles.



As a special case of marketing, users might be interested in marketing themselves. Matchmaking has proven to be of interest in a number of web services, such as the Friendfinder (www.friendfinder.com), chat forums etc. Users can be retrieved as described in the marketing approaches above (describing the desired profile) or using their own user profile (seeking for users with similar interests). The same functions can be used to find experts to a given topic.

Within the system, objects and users are actually *references* to objects and users. Since these are only valid within the context of the system, they have to be translated into a textual description before they can be exported to another system using different references. Functions returning keywords and other formal descriptors allow such a translation. This way, object descriptions and user profiles can be transferred, e.g. to the query form of a WWW search engine, to search for additional information. As one special case, users can export their own user profile this way, e.g. to import it into another recommender system.

The corresponding functions allow importing the type of object and profile description generated by the export functions.

Cells returning the same type of data that was fed into them are useful for supporting browsing tasks. There are three possible thesauri to browse: a keyword thesaurus, an object thesaurus, and a user the-

saurus (left top to right bottom). Appropriate similarity measures have to be found here.

Table 5 summarizes some of the scenarios listed above by grouping cells according to the input data type. The center column is of special importance to the system. It is necessary to allow users to provide ratings and thereby to keep the collaborative part of the system running.

Descriptor	**Actual users** exchanging data and browsing	**Actual users** seeking recommendation	**Marketing people** seeking specific groups of users & **Users** interested in Matchmaking
Object			
User			
	Descriptor	**Object**	**User**

Table 5: Summary: Which user group is interested in which functionality?

COMPUTATIONS

When executing queries, all computation is done based on the two relations "likes" and "matches" from Figure 3. Table 6 shows how the individual cases can be computed. For the sake of simplicity, we used the following functional notation: *d* stands for a formal descriptor, *o* for an object, *u* for a user. L(), $L^T()$, M(), $M^T()$, LM() and $LM^T()$ stand for an application of the likes-, the matches-, or the combined ("Filterbots") relation respectively, in the one or the other direction (likes, is liked, ...).

Descriptor	$M^T(M(d))$	$M(d)$	$L^T(M(d))$
Object	$M^T(o)$	$M(M^T(o))$	$L^T(o)$
		$LM(LM^T(o))$	
		$L(L^T(o))$	
User	$M^T(L(u))$	$L(u)$	$L^T(L(u))$
	Descriptor	**Object**	**User**

Table 6: Computation of the individual queries

Extension: user-defined computation

We just covered queries such as "Give me all movies that Andrea likes". Answering that query would return the rather short list of positively rated objects from Andrea's user profile. However, we can also ask, "Give me all movies that users similar to Andrea like". Since we now include recommendations from many more users, this would result in a much longer list. In a similar fashion we could ask, "Give me all movies that people similar to those users that are similar to Andrea like", and so on.

Such query terms can be implemented by inserting $R^T(R())$ and $R(R^T())$ pairs into the queries in Table 6, with R() being any of the relations L(), M(), and LM(). The query "Give me all movies that Andrea likes" L(Andrea) can be turned into "Give me all movies that users similar to Andrea like" by inserting $L^T(L())$, $M^T(M())$, or $LM^T(LM())$ leading to $L(L^T(L(Andrea)))$ etc. There are three possibilities, because the "users similar to" operator can be executed in a content-based, collaborative, or combined way. Query by example, for example, can be implemented as $M(M^T(M(d)))$.

QUERY FORMULATION AND USER INTERFACES

When a system accepts two or more parameter types to formulate queries (i.e. the system implements two or more cells of a column from Table 4), then users should be allowed to combine these input parameters in a single query. An example for a successful implementation of this concept is relevance feedback. Consequently, the goal for implementing the center column of the above table is to implement a system that allows combining objects, formal descriptors, *and* users in a single query.

One possibility would be to combine two or more terms using the Boolean operators AND, OR, and NOT. An example from the movie domain could be "Give me all movies that user Lars likes and that have descriptions containing the words 'car' or 'race' which could be entered as "*userLars AND (car OR race)"*.

Entering parameters

Entering formal descriptors is generally uncritical, e.g. using a text-based interface. The keywords stored in the inverted files of the retrieval system are so simple that they can be identified uniquely from what user types. Object and user parameters are more difficult to enter. Identifying a user or a document from typed text is not always unambiguous. These entities could be identified using their internal references, but users might not know them or misspell them. As already mentioned in the case of relevance feedback, an appropriate solution is the following. Users describe the desired entities using formal descriptors. If the query returns a single entity only, this entity is selected automatically. If the query returns a number of entities, a selection dialog is started to select among them.

If input parameters of different types can be mixed, ambiguities about their type may occur. In a movie recommendation system, "Terminator" might be a search term, a movie, or the nickname of a user. If users enter queries via a textual interface it can therefore be necessary to resolve ambiguities by annotating parameters with their type.

If a system allows the retrieval of different types of entities using the same user interface (e.g. either objects or users), users have to select what kind of entity they want to retrieve, e.g. using a simple menu.

User interfaces for user-defined computation

When we permit users to apply extra relation pairs, as described above, the retrieval-like syntax just described

becomes insufficient. Users do not only have to provide parameters and combine them with Boolean operators; they also have to guide the computation, i.e. the application of relations. One possible syntax is the functional syntax used above, such as "$L(L^T(L(Andrea))$". Another possibility is an SQL-like syntax [6], using a simplified select-statement. The rather cryptical type of syntax is of course not desirable, especially when dealing with casual users. However, a number of graphical visualizations have been created to bring light into SQL syntax.

Figure 4 shows a screenshot of one of our experimental prototypes in the domain of movie recommendation. The query shown returns a list of keywords, describing boxing movies similar to "Rocky2" (i.e. $M^T(M(\text{“boxing”}) \text{ AND } LM^T(LM(\text{“Rocky 2”})))$). The square pieces on the left are used to enter parameters. Domino pieces and set operations (e.g. "AND") are appended to the right to direct computation. The T-icon symbolizes a keyword, the head icon a user, and the video-tape icon a movie. The double-size domino piece represents the $LM(LM^T())$ relation.

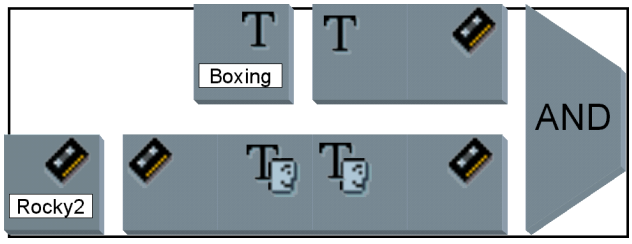


Figure 4: Prototypical user interface for entering queries of SQL-like syntax.

FUTURE WORK

We currently work on implementing the described techniques as part of our TV recommendation system TV Scout (http://www.tvscout.tvtoday.de). We experiment with different software platforms promising fast computation, as well as with user interfaces.

In this article, we assumed the existence of two explicitly represented relations *likes()* and *matches()* and experimented with a number of ways to apply them. Other systems provide different or more relations. Users might be allowed to store a hand-selected neighborhood of mentors in their user profiles (*user x user*), a set of keywords describing their favorite objects (*user x keywords*), or a set of favorite queries (*user x query*). Similar to the way we joined relations in this article, all of these relations could be joined and matched with each other, generating new functionality.

CONCLUSIONS

Mixing content-based and collaborative techniques was originally done to overcome the first-rater problem. However, a number of new possibilities can be derived from this basic idea, waiting to be exploited. This article is a first exploration of the possibilities. Next steps will be actual implementations and formal evaluations.

REFERENCES

1. Konstan, Joseph. A., Riedl, John, Borchers, Al and Herlocker, Jonathan L. *Recommender Systems: A GroupLens Perspective.* Recommender Systems. Papers from the 1998 Workshop. (pp. 60-64), Menlo Park, CA: AAAI Press, Technical Report WS-98-08.
2. Greening, Dan R. *Collaborative Filtering for Web Marketing Efforts.* Recommender Systems. Papers from the 1998 Workshop. (pp. 53-55), Menlo Park, CA: AAAI Press, Technical Report WS-98-08.
3. Baudisch, Patrick (1998). *Recommending TV Programs on the Web: How far can we get at zero user effort?* Recommender Systems. Papers from the 1998 Workshop. (pp. 16-18), Menlo Park, CA: AAAI Press, Technical Report WS-98-08.
4. Bradley N. Miller, John T. Riedl, Joseph A. Konstan, Experiences with GroupLens: Making Usenet Useful Again, *1997 Usenix Annual Technical Conference*, Anaheim, CA, 1997.
5. David Maltz and Kate Ehrlich: Pointing the way: Active collaborative filtering, CHI'95 Human Factors in Computing Systems, p. 202-209, 1995
6. Jerry Kiernan, Michael J. Carey: Extending SQL-92 for OODB Access: Design and Implementation Experience. OOPSLA 1995: 467-480