Synergy: A Workbench for Collaborative Filtering Algorithms on User Interaction Data

Babar Khan Tareen  
School of Computer Science and Engineering,  
Seoul National University,  
Seoul 151-742, Republic of Korea  
babartareen@europa.snu.ac.kr

Jae-won Lee  
School of Computer Science and Engineering,  
Seoul National University,  
Seoul 151-742, Republic of Korea  
lyonking@europa.snu.ac.kr

Sang-goo Lee  
School of Computer Science and Engineering,  
Seoul National University,  
Seoul 151-742, Republic of Korea  
sglee@europa.snu.ac.kr

ABSTRACT
Collaborative Filtering (CF) is a process used by many recommender systems for recommending content based on similarities between users. CF traditionally used item purchase history or rating information only, for finding out similar users. Nowadays users interact with the content in many different ways. Users can rate, tag, like, dislike or subscribe to the content and some information about relationship between users is also available usually in social network form. This additional interaction information can be used to filter out similar users more effectively. Various CF algorithms have been proposed to take into account this additional information and suggest better content. But it is very hard to figure out which algorithm will work best on a particular dataset. In this paper, we propose a generic data model, for storing user interaction data, which can easily support many collaborative filtering algorithms. We have also implemented a tool which assists in creating, running and comparing different collaborative filtering algorithms. The tool is based on the proposed data model and can easily be extended by using plugins. It is helpful in figuring out which algorithm works best on a particular domain.

Author Keywords
Collaborative filtering, Recommender systems, Information retrieval.

ACM Classification Keywords
H.3.3. Information Search and Retrieval: Information filtering

INTRODUCTION
With the rapid growth in online content, it is getting harder and harder for the users to find content that might be relevant to their needs. Recommender systems help users by suggesting content that may be of some interest to them. Recommender systems use historical data and preferences to recommend the content. Recommender systems act like search engines, the difference is that instead of using keywords, user’s preferences and historical data is used to find relevant content. Collaborative Filtering (CF) is a process of recommending content by finding similar users based on a user’s historical data and then suggesting the content accessed by the similar users. CF abstracts complexity of human liking and disliking behavior by clustering users who share similar history. As a result collaborative filtering provides good recommendations.

Traditional services only gave users the option of rating the content on numerical scale. This rating information was used to find out other users who are similar to the particular user. In case of e-commerce applications, purchase history was used instead of ratings. With the increase in user generated content in Web 2.0 landscape, now users are given many different ways to interact with the content. Users can tag, mark some content as favorite, like or dislike, review, give comments and the old rating option is also available. Every interaction option is a different dimension in itself and defines a part of user’s behavior. Some users may be more comfortable with using ‘Like’ or ‘Dislike’ type of options for expressing their interest in the content others may like the fine grained control in expressing the degree of interest provided by the numerical scale and yet others might use the Like & Dislike and numerical scale both. This provision of multiple ways to interact with the content has distributed the user’s expression of interest along multiple dimensions. This reduces traditional CF accuracy because those algorithms only use rating dimension for finding out similar users and recommending relevant content. Figure 1 depicts a scenario in which using any one interactivity dimension, rating, favorite or emailed does not correctly represent the level of similarity between two users. If we consider rating interaction then Alice and Bob share content A. In case of favorite and emailed interactions then Alice and Bob don’t share any content and thus are not similar. None of the interaction dimensions individually can completely model
the similarity between Alice and Bob, for that we need to consider all three interaction dimensions.

Using different interactivity dimensions opens up a window for improvement in CF algorithms. Some work has been done to use different interactivity options instead of just ratings in CF algorithms. In some domains, using tags resulted in better recommendations [1]. And sometime using tags together with previously used rating or purchase history resulted in better recommendations [2,3]. The problem with those approaches is that they highly rely on the characteristics of the data. We can’t take the tag based CF algorithm, and run it on any dataset with tags and expect to get similar results. It is interesting to note that interaction data is dependent on users and thus can change over time so an algorithm which was based on a particular interaction dimension may degrade over time because of the changes in the underlying dataset. This can be a serious limitation. This phenomenon is also affecting services like YouTube [4]. YouTube recommends videos based on previously played videos but recently YouTube has added a video recommendation feature based on the favorite videos.

Besides multiple interaction options more and more services are adding social networking features in one or another form. Social networking enriches services usually by providing users with the ability to share or discuss about the content with other users. This additional information be used to refine CF.

This paper introduces a data model for storing users’ interaction information with the content. The basic unit in the data model is an interaction. An interaction can be defined as any action performed by a user on the content. Some examples of interactions are playing video, rating an item, reviewing a product, ignoring recommendation about a product etc. Interactions can be considered as events as interactions are usually modeled as events in many systems. By handling all interactions in the same way this data model can help in generalizing CF algorithms by removing the dependency on any particular interaction dimension.

The social networking information is also stored for use by the CF algorithms.

We have also developed a tool, based on the proposed data model, which can easily model various CF algorithms. The objective of this tool is to remove the burden of computing similarity between users from the application and to help system administrators in running different CF algorithms on their datasets and see which algorithm performs best on that particular dataset. Furthermore this tool can also help in using different CF algorithms for different users in the dataset thus making it possible to select best CF algorithms at user level. Because of the flexibility provided by the underlying data model, this tool can use any interaction dimension in algorithms. Thus making it easy to use an algorithm originally defined for tags on pre-parsed comments. Researchers can also use this tool to easily model and run CF algorithms. The tool can generate recommendations using a part of data and verify the correctness of recommendations by checking against the remainder of the data. This way researchers and system administrators can quickly compare different algorithms.

RELATED WORK

Collaborative filtering is in use since early nineties when Goldberg, et al. [5] filtered emails and documents by sharing users’ reactions with one another. Resnick, et al. [6] developed GroupLens, a news recommendation system based on collective ratings from different users. The basic CF algorithms use explicit rating information and some similarity measure like cosine similarity to compute similarity between users. Then, top k similar users are selected and the content that is not yet accessed by the current user is recommended. There have been many modifications to basic collaborative filtering, usually additional information is added to improve recommendation quality.

Adomavicius and Tuzhilin in 2001 [7] presented a multidimensional recommendation model that used data warehousing techniques to process multidimensional data. Multidimensional model is suitable when user preferences are dependent on multiple factors. For example the decision of going to a tour on a particular resort may depend on season, price, distance and other factors. Multidimensional model can accommodate these factors and recommend content accordingly. Anderson, et al. in 2003 [8] used domain based rules along with multidimensional model to recommend music. The rules were based on the domain under consideration. For example, if a user has rated highly an album of Leonard Cohen then other albums from Leonard Cohen will be preferred. However, the problem highlighted in this work is about multiple interactions with the same content, which cannot be effectively modeled in a multidimensional system because generalizing interactions is difficult. Some users may tend to use all available interaction options other might be using only one option, in such cases it might not be appropriate to compare all
interaction dimensions to compute similarity between the users.

Zanardi and Capra [1] used tags to improve search results. Their model was based on a tagging system, where users associated tags with resources. They computed users’ similarity by comparing tags used by different users in other words the more tags two users have used in common, the more similar they are, regardless of what resources they used it on. They also computed tags’ similarity by comparing the tags associated with resources, the more resources are tagged with the same pair of tags, the more similar (related) those tags are, regardless of the user who used them. Then they used these similarities for query expansion and ranking the search results. Their approach can also be used in a CF recommender system. Nakamoto, et al. [2,3] used tags as context for recommending bookmarks. If two users tag a bookmark with the same tags that signifies that both the users are looking at the bookmark from the same perspective, thus both the users are similar and if the users annotate same bookmark with different tags that signifies that both the users are looking at the bookmark from different perspective hence the degree of similarity between the users is low.

Zheng, et al. [9,10] used social network information to only compute similarity between the users connected via social network. This reduces the number of comparisons required for finding out similar users and the resulting algorithm has similar accuracy with much lower runtime. This model is based on the assumption that the people who are socially connected have similar preferences. In an environment like Delicious [11], an online bookmarking service, where user can bookmark documents from multiple domains this assumption may become weak and result in poor recommendations.

In a white paper published by Cisco [12] use of socially relevant gestures in collaborative filtering has been emphasized. Socially relevant gestures are analogues to interactions in this paper. Authors also suggest the use of inverse relationships among users. For example if Alice highly rates a document but Bob gives poor rating to that document then it can be deduced that the documents which Alice like are not preferred by Bob.

These modifications in basic collaborative filtering algorithm generate better results but the improvements are for specific domains only and are hard to generalize. With the presence of multiple interaction dimensions, those algorithms which only use limited information about user interactions can only produce better results for a specific group of users but not all. Therefore it has become hard to figure out which interaction data and which algorithm should be used for recommending content in order to get good results on a particular dataset.

Duine [13] is a java based framework for building recommender applications. It can process user feedback in rating or logical (true / false) form only, thus difficult to customize for interactions which can have text values like tags and comments.

**DATA MODEL**

Databases design varies from application to application. Some applications focus on minimum disk space while others focus on performance and introduce redundancy. Similar different application have different feature thus different underlying data model. While designing the recommendation engine, application developers focus on a particular algorithm which uses a very fixed set of input parameters for recommendation and entire database is optimized based on those parameters. In such setup, users’ interaction information with content usually is distributed all over the database and different interactions are handled differently. This makes it difficult to use different interaction dimensions for example using number of times a video has been played for recommendations instead of ratings.

This difficulty can be overcome by treating all users’ interaction information uniformly. In this paper, we propose an interaction based data model for collaborative filtering systems. An interaction is the basic unit in the model. The model treats all interactions evenly. Collaborative filtering algorithms based on this data model will take some interaction dimensions as input and will return recommendations for some other dimension. As a result algorithms based on this data model are very flexible.

**Conceptual Model**

Interactions are very similar to events. An event is triggered by a source and affects some aspects of the system, similarly an interaction is caused by a user and happens on some content. This data model uses this same analogy and models interactions as events. This is because many existing systems record users’ interactions as events so it becomes easy to think of interactions as events. To formally define this model suppose that $U$ is the set of users, $C$ is the set of contents and $I$ is the set possible interaction which a user can do on some content then interaction event $E$ is set of all events that occurred and $V$ is the set of values associated to those interactions.

$$U = \{u_1, u_2, \ldots, u_n\}$$

$$C = \{c_1, c_2, \ldots, c_n\}$$

$$I = \{i_1, i_2, \ldots, i_n\}$$

$$E = \{(s, p, o, t) | s \in U, p \in I, o \in C, t \text{ is time}\}$$

$$V = \{(e, a) | e \in E, a \text{ is some value}\}$$
Every interaction event has associated attributes which describe that event. Every event must have a Subject, Object, Time and Event Type associated with it. Subject represents a user or an actor whose interactions caused this event to occur. Object is an entity, the content, with which user interacted and as a result this event occurred. Time is the time at which the interaction occurred and the event type identifies different interactions from one another. For example rating a video will have different event type then marking it as favorite. Other attributes can also be associated with the events. Events can have no values associated with them like in case of marking some content as favorite or can have values like annotating a document with tags, in this case tags will be considered as values for the Tag event. Social Network or explicit similarity information is stored separately. Figure 2 shows the conceptual data model.

**Database Schema**

Figure 3 shows the entity relationship diagram for the proposed data model. Event table stores the interaction information. Type table describes different types of interactions. Events like ‘Like / Dislike’ which do not have any associated values can be described by a single record in Event table. For events which have associated values, we put the values in Value table. Event and Value tables store event related information. Type table is for providing descriptive names to different interactions so that other tools can easily distinguish different event types. Two other tables are used for storing explicit relationship information between subject-subject and object-object. This explicit relationship information can be used to give more weight to some entities or only consider similar entities from those tables or in some other way by algorithms.

As already mentioned, database schemas are designed according to the application requirements. Therefore in this model we use the unique identifiers for subject, object and event values already being used in the database tables and these columns are foreign keys to those tables. This makes it easy to integrate this model with existing databases.

**FLEXIBILITY**

To show the flexibility of this data model we have taken two distinct datasets and modeled those in our proposed model. Two datasets used were, Delicious dataset and Last.fm dataset.

### Delicious

Delicious is an online bookmarking website. It supports bookmarking online documents and adding tags to the bookmarks. Different users are connected to each other by becoming fans of one another. A user can interact with content in two ways by bookmarking an online document and by associating tags to already bookmarked document. Thus the interaction of tagging a document depends on interaction of bookmarking the same document first. Figure 4 shows how different components are interrelated in delicious dataset. Table 1 shows how we mapped delicious data in our data model.

### Last.fm

Last.fm [14] is a music recommendation site. It uses users’ music listening events to recommend music. Whenever a user listens to some track using Last.fm software that event information is uploaded to the website. Last.fm also supports tagging so the users can tag the tracks. The dataset which we are using contains track listening events and tags associated with those tracks but lack the information on
who tagged a particular track with a particular tag. Therefore we assume that all users listening to a particular song have associated it with all the tags associated with that song. This assumption is made just to show that the proposed data model is flexible enough to store Last.fm data. Figure 5 shows relationships between different components in Last.fm dataset. Table 2 shows how we mapped Last.fm dataset into our data model.

**SYNERGY – COLLABORATIVE FILTERING WORKBENCH**

Application developers and administrators face the problem of deciding which algorithm to use for recommending content for an application. It is hard to know how application data will evolve once the application is deployed, especially in applications where content is generated by the users. As a result, over time the recommendations generated by the applied algorithm might degrade. Many collaborative filtering algorithms generate recommendations by dividing the process into two steps (i) computing similarity between users and (ii) ranking and recommending top ranked content from similar users. By integrating these two processes in the application it becomes difficult to change the CF algorithm being used without making changes in the underlying application code.

To reduce the impact of these hurdles we have developed Synergy, a collaborative filtering workbench. Synergy is built using Java programming language and is based on the proposed data model. Its objective is to help system administrators in running different collaborative filtering algorithms on their datasets and decide which algorithm best suits their needs. It also makes it easy for the system administrators to change the CF algorithm already used in the application by removing the need to compute users' similarity at application level. This tool can use any CF algorithm to compute the similarity among users and applications can access user similarity computed by this tool for recommending content. Thus administrators can use any algorithm to compute the similar users any time without modifying the application, even at runtime.

This tool works by dividing collaborative filtering algorithms into two steps (i) computing similarity between entities and (ii) finding content for recommendations. Some CF algorithms use user to user similarity while others use item to item similarity. Therefore we use the term entity to refer to both user and item. The tool is very customizable. Key components in the tool are implemented as plugins. Users can implement new plugins to add more functionality or get more control of the execution of the tool.

The interface is divided into two sections, sections 1 & 2 as highlighted in Figure 6. Section 1 focuses on configuring how the tool should run and section 2 displays the algorithms that will be executed for computing similarity between entities.

Synergy can run in two modes, first is the ‘Compute Similarity’ mode in which it only calculates the similarity values between different entities, that is subject-subject or object-to-object and stores the results in a database table for later use by the recommendation engines in the applications. Second is the ‘Compare Algorithms’ mode. In this mode it calculates and stores the similarity values just like in the previous mode but the difference is that it uses certain percentage of the actual data to compute the similarity and after computing the similarity it tests the algorithm by recommending content and using the rest of the data to check the accuracy of generated recommendations. By default F1 measure and normalized discounted cumulative gain (nDCCG) are implemented for computing the accuracy of generated results but other
accuracy measures can be added via plugins. Results of the tests are also stored in a database table. These results can be used to compare different algorithms and check which algorithms perform best on a particular dataset. Furthermore the tool can also write the similarity values and test results in comma separated values (CSV) files.

**Configuration**

For running this tool, first we need to configure the database connection information and map conceptual data model to physical tables. These settings can be configured via Tools menu.

**Defining an Algorithm**

After configuring the settings, we need to define algorithms for computing similarity between entities. Figure 7 shows the dialog used for defining an algorithm.

Defining an algorithm is easy. First enter the name for the algorithm. The ‘only use explicitly mapped objects …’ check box will force the algorithm to only compute similarity between entities which have explicit mapping given in a separate table. This can be used to simulate people connected via social network and checking the check box will be analogous to only using social network contacts as recommenders (people whose will be considered for recommending new content). Similarity measure combo box contains a list of similarity measures used to compute similarity between vectors. Currently two similarity measures are implemented (i) Cosine similarity and (ii) Pearson’s correlation. Similarity measures are implemented as plugins so other measures can also be included. The ‘Based on’ list consists of all interaction types available in the dataset. Note that in Figure 7 this list contains Bookmark and Tag only. This is because at the time of this snapshot delicious dataset was used which only supports two interactions, Bookmark and Tag. This list is populated dynamically based on interaction types present in the database. If a user selects multiple interaction types then the similarity measure is executed on all selected interaction types individual generating multiple similarity values, one similarity value per interaction type. To compute one similarity value from multiple similarity values, composition functions are used. The ‘Composition’ combo box displays list of composition functions available. It is also implemented as plugin so different implementations can also be added. By default three composition functions are implemented, (i) Normalized sum: it averages different similarity values computed with different interaction types, (ii) Multiply: it multiplies similarity values computed with different interaction types and (iii) Cascade: it is used for interaction types that are dependent. When two users share a primary interaction with an object then secondary interaction vector is used to compute the similarity among the users. For example if two users bookmark (primary interaction) same document then tags (secondary interaction) associated by each user to that document are compared to get a single similarity value. Figure 8 shows an example of using composition to aggregate similarities computed by bookmarks and tags.

**Plugins**

Synergy uses Java Simple Plugin Framework (jspf) [15] for supporting plugins. JSPF simplifies the plugin development process. To implement a plugin a user just has to implement an interface which defines the functionality of the plugin and annotate the implementation class with PluginImplementation annotation.
Synergy has four different types of plugins: (i) Similarity measure, used to compute similarity between two interaction vectors. (ii) Composition, for aggregating different similarity values (iii) Accuracy measure, for computing the accuracy of generated recommendations during testing and (iv) Recommendation notification, provides a notification to the plugin about the generated recommendations. Recommendation notification plugin can change notifications if needed. This can be used to filter out some content which for some reason should not be recommended or to re-rank the recommendations. Figure 9 shows the implementation of F1 plugin.

Execution
Running the tool in ‘Compute Similarity’ mode will generate similarity values between users or items and save those values in database table. In order to compare the algorithms, this tool can generate and check recommendations against the data available in the database. Section 1 in Figure 6 shows the settings related to tests. Percentage of data for computing similarity and testing can be configured. Users can select interaction type that should be recommended. Multiple accuracy measures can be selected. The user set, represents the number of similar users to consider for recommending content and item count is the number of recommended items.

Synergy stores the results in SIM_SIMILARITY and SIM_TESTRESULT tables. By selecting the ‘Write tables to files’ checkbox those tables can be written to CSV files.

EXPERIMENTATION
Tests were executed on two different datasets, Delicious and Last.fm.

Delicious
Delicious dataset consists of the web crawl between May 2nd, 2009 and May 8th, 2009. It consists of about 2,012 users, 1,910,143 documents, 2,913,452 bookmarks, 353,936 unique tags and 10,629,589 tags associates with bookmarks. To better understand the affect of data on algorithm performance, this dataset was divided into two sets, Dataset A which consists of documents with at least 2 bookmarks and the Dataset B which consists of documents which were bookmarked at least 10 times and users with at least 50 bookmarks. Table 4 compares these two datasets.

Four algorithms were compared against one another to compare the accuracy of those algorithms. Plain CF (labeled CF) algorithm used bookmarks for computing similarity between users and Social Network Collaborative Filtering (labeled SocialNet) [9,10] also used bookmarks for computing similar users but instead of computing similarity between the user and all other users, similarity was computed only between users who were part of user’s social network. Tags were used instead of bookmarks for computing similarity between users as suggested in [1]
filtering accuracy is increased. has improved and as a result tag based collaborative reduction in the bookmarks and thus the tags in dataset B. We have reduced the rarely shared bookmarks and recommending bookmarks works better. This is due to the reduction in the bookmarks and thus the tags in dataset B. We have reduced the rarely shared bookmarks and associated tags which make a user more unique but at the same time add noise. Thus in dataset B the quality of tags has improved and as a result tag based collaborative filtering accuracy is increased.

**Last.fm**

Last.fm dataset consists of 50 users, 169,721 tracks, 1,046,904 track listening records and 245,652 tags. We ran plain collaborative filtering and tag based collaborative filtering algorithms. We also executed CF (using bookmarks) + Tag based algorithm which first computes the similarity between users based on tracks listened and associated tags separately and then takes average of the two values, see Figure 8 for details. Figure 12 shows the results. X-axis represents users in no particular order and y-axis is the F1 measure for recommendations. Because we don’t have information about which tags were used by which user and to model the data properly we assumed that all users associate a track with all tags associated to that track. As a result plain CF and Tag based CF overlap, except for a few users. CF + Tag based approach performs better for a good number of users. It is clear from the results that using multiple interaction dimensions can produce better recommendations as compared to just using a single dimension.

**CONCLUSION**

In this paper, we have presented a generic data model for recommender applications. The model follows an event like approach where every interaction with the content is considered as an event and the details of the interaction are associated to the event. The model is flexible and two different datasets were modeled using it.

We have also implemented Synergy – a CF workbench, which uses the proposed data model to run different CF algorithms. The aim of this tool is to make it easy to run different algorithms on a particular dataset and help in figuring out which algorithm performs best for a dataset. This tool can also be used to compute similarities between users. Applications can then use the computed similarity values for recommendations. This makes it easy to change CF algorithms used by an application without making many changes to the code. And because the user similarities are computed outside of the application, different CF algorithms can be used for different users based on what algorithm perform best at user level. Due to the flexibility of underlying data model, traditional single dimension based collaborative filtering algorithms and many different variations of collaborative filtering algorithms can be defined and executed on a dataset using this tool. Researchers can use this tool to implement new algorithms quickly and compare those to existing algorithms without worrying about implementation details and focusing on actual algorithms. Furthermore, by using plugins we have made this tool customizable thus easy to implement new algorithms.

Test results on delicious and last.fm datasets show that recommendation results vary from dataset to dataset and that using multiple interaction dimensions can produce better results.

In Web 2.0 landscape, where services are providing different ways to interact with the content, this data model can be a basic building block for dynamic recommendation systems which can use combination of different algorithms or use different interaction information to generate better recommendations.

**ACKNOWLEDGMENTS**

- This research work is sponsored by ‘Higher Education Commission (HEC), Govt. Of Pakistan’ under the scholarship program titled: MS Level Training in Korean Universities/Industry, bearing GRE type Test Roll Number 71330202.

- This research was supported by the MKE(The Ministry of Knowledge Economy), Korea, under the
REFERENCES


