

The Orellana Project: An Adaptive Recommendation System for Amazon.com

James Creel, Alexey Maslov, Adam Mikeal, Colin Speight

Computer Science Department

Texas A&M University

3112 TAMU

College Station, TX USA

ABSTRACT

The Amazon E-Commerce Service allows developers to access Amazon.com's extensive and exhaustive database of product information to power their own E-Commerce applications. Product information (such as titles, author, and publication date) and content generated by Amazon customers (such as Listmania! product listings, reviews, and wish lists) are available using queries to the AES API. By systematically extracting and recording this metadata into an external database, exploring and consolidating the various relationships that exist between products, performing statistical clustering on the items, and providing a simple interface that adapts to a user's browsing habits, we propose that the existing Amazon recommendation system can be enhanced and augmented by combining various relationships to help users discover new products that are related to their interests.

Keywords

Amazon E-Commerce Service, recommendation systems, E-Commerce, data mining, intelligent user interfaces, term vectors, clustering

INTRODUCTION

E-Commerce and Recommendation Systems

While the rapid growth of E-commerce has allowed companies to offer more product options and customization choices to consumers, this often carries the extra burdens of information processing when making product selections. One widely utilized solution to this problem is to offer a recommender system that uses metrics such as an analysis of past buying habits, comparisons between customers and the products that they have purchased, or reviews from other customers to make predictions of future shopping behavior and give a degree of personalization for each

customer who uses the site. Schafer, Konstan, and Riedl argue that recommender systems enhance E-commerce sales in three ways:

1. Since visitors to E-Commerce sites frequently look over items without purchasing anything, recommender systems help turn browsers in to shoppers by assisting customers find what they want easily.
2. Improving cross-selling by suggesting additional products to customers to purchase. An example would be suggesting additional items during the checkout process based on items in the shopping cart.
3. Encourage customer loyalty by creating a value-added relationship between the site and the customer. The more a customer uses a recommendation system and teaches it what they want, the more likely they are to return to the web site rather than buy from a competitor [1].

These systems however, suffer from limited applicability – that is (as implied by 3. above) they tend to apply only to product available from a single provider. If product information were represented with widely recognizable semantics, recommendation systems would be able to make sense of products from a variety of sources.

Amazon's Item-to-Item Recommendations

The recommendation system available through Amazon.com is a major marketing tool for the company and is used extensively to tailor and personalize the web site to the habits and tastes of each individual customer. After logging in, a customer may click on a "My Recommendations" link that leads them to an area where they may rate the products that have been recommended and those they have previously purchased, filter their recommendations by product line and subject area, and see how items were recommended to them (in other words, they can see which product they rated that triggered the recommendation). In addition to this personalized area, customers also receive recommended products based on the items that are currently in their shopping cart [1].

This work is licensed under the Creative Commons Attribution-NonCommercial-ShareAlike License.

To view a copy of this license, visit <http://creativecommons.org/licenses/by-nc-sa/2.0/> or send a letter to Creative Commons, 559 Nathan Abbott Way, Stanford, California 94305, USA

The algorithm used by Amazon to generate recommendations matches each of the user's purchased and rated items to other similar items, then compiles all similar items into a recommendation list. Items are considered similar by scanning past sales records and determining which items are frequently bought in pairs by other customers. The more frequently the product pairs occur, the higher the similarity metric that is used to gauge how related two items are [1]. Items that have the highest similarity rating are added to the customer's recommendation list if they can be paired with products that have been purchased or rated by the user. For example, if a customer has purchased or given a high rating to "The Mystery Science Theater 3000 Collection, Volume 1", then the recommendation system will find other items that have been purchased by other customers who bought this DVD set. The more customers who buy the products in pairs, the higher the probability that combination will be used as a recommendation.

The user interface to such a system poses an interesting challenge: how do you communicate a large quantity of information to the user without overloading them with similar-looking data? The easiest option, bulleted lists of the items the recommender system produces, was rejected for two reasons: first, this delivery is markedly similar to the existing Amazon interface, and a goal of our project is to distinguish our results from those provided by the company.

Second, long lists of similar data are not efficient mechanisms for communicating the information our data contains, which exists on several planes. For instance, while a set of 50 products will all be related in some manner to the original selection, some relationships are "tighter" and some "looser" than others; some of the relationships are derived from customer wishlists, and others from purchasing patterns, etc. Trying to represent these various bits of information textually in a long list would make it difficult for a user to visually appraise the search/recommendation results, or to see the make-up of the results at a glance.

PROJECT ABSTRACT

With the introduction of the Amazon E-Commerce Service, it is now possible for developers to access Amazon's extensive and exhaustive database of product information to power their own E-Commerce applications. This service is provided free of charge once a user has registered for an account and permits an unlimited number of queries to the Amazon API. These queries can reveal a wealth of information for every product available in the Amazon catalog, including product data (such as title, pricing, and images) and content generated from and contributed by Amazon customers (such as product reviews, wishlists, and Listmania! lists). We intend to make better use of this data than the recommendation system provided on Amazon.com.

Collection and Analysis of Textual Data

The data exposed by the Amazon API includes a wealth of textual data, such as names and titles of products and editorial reviews written about the products. Such data provides information that could be used to gauge the similarity of products in a more subtle (and less commercially oriented) way than the relationships exposed by Amazon's API. This approach requires a metric of comparison for textual data.

Part of the foundation for our recommendation system is the aggregation of several existing relationships between Amazon.com products. By tapping into the Amazon.com API as a source for this information, we will record basic metadata about each product that is discovered as well as the following relationships:

- **Similar Items** - this section is roughly analogous to the "customers who bought this also bought" section of an Amazon product listing. It is reasonable to expect that users who buy the same products can have similar interests.
- **Listmania! Listings** - these are user-defined associations that usually have an overarching theme that binds all the products on the list, such as "My Favorite Movies (The Ones I Own)" or "Why Zombie Flicks are Better than Real People v 1.0."
- **Wishlists** - any user may create a wishlist on Amazon's website to tell others about products they would like to have bought for them.
- **Reviews** - Amazon customers may submit reviews and ratings for any item that is listed. By extracting reviewer IDs and querying the API, it is possible to build associations between products that have been reviewed by the same person and given a particular rating. For instance, if a particular reviewer gives a product a favorable rating, it is possible to find other products that the same reviewer also has also given a high rating and establish a relationship between the two.
- **Reviewer/Wishlist** - if a reviewer gives an item a high rating, items that appear on their wishlist can be considered related
- **Browse Node** - browse nodes are used to classify Amazon products within their catalog. The intersection of these browse nodes determines the degree of this relationship.
- **Accessory** - Amazon classifies certain items as accessories of others.
- **Book Subject** - since books are categorized by subject. Those within the same subject category can be considered related.
- **Author/Creator** - Items written, composed, or created by the same individual.

By combining information gleaned from the Listmania! associations, wish lists, and user reviews with the current recommendation system, it may be possible to not only create a more robust recommendation system that incorporates more human-generated relationships into the system, but also generates detailed product recommendations for novice and casual users, based on pre-determined relationship types.

Visualization and User Model

Furthermore, we will investigate the inclusion of an intelligent user interface as a front-end to the relationships and metadata that are recorded from the Amazon API. The interface will present the recommendations in an organized manner to facilitate browsing. This organization will be expressed through clusters of related items, calculated by our clustering algorithm. The interface also includes a user profile that serves as a repository for structured data about a user's expertise, preferences, and tastes. This data will provide the basis for adaptiveness in the interface through filtering of search results.

Previous Work and Implementation

The Orellana Project is based on ideas and basic concepts that were developed for our Hypertext project, "A Hypertextual Augmentation of the Amazon E-Commerce Service." The primary concept that was adopted for this project was the metadata harvesting to build a local version of the Amazon product database. However, we decided to expand this concept to gather more detailed product information. Therefore, all code implementations, including the harvester scripts and database component, were completely re-written for this project.

POTENTIAL USERS

The current recommendation system used by Amazon favors customers who not only frequently purchase items from Amazon, but also take the time to rate individual products on the website. The more time spent rating items and money spent purchasing items and building a sales history, the more accurate and precise the item recommendations become. Therefore, it would be worthwhile to investigate alternate methods of constructing product recommendations for the casual or new user to the Amazon.com website.

By augmenting the textual link analysis with statistical analysis of product metadata, we anticipate the ability to provide better recommendations to first-time browsers. While the recommendations produced by these mechanisms will not be as heavily influenced by the user's past actions or interests, they will still be significantly different from the simple product links currently provided this class of users by Amazon.com.

PROJECT IMPLEMENTATION

Ingestion Component

In a production environment, we envision our product being used by Amazon to provide refined product recommendations based upon multiple relationships. Within this context, we would have direct access the Amazon backend database of product information. However, since we do not have such access (and to our knowledge, nor does such a representation of Amazon's data exist), we must first construct a database using the Amazon API.

To construct this database, we have written PHP scripts to automatically query the API, parse the XML REST data, and store the results in a MySQL database. We will provide some initial seed data in the form of an Amazon-specific unique product identification number (ASIN) to begin construction of this data source. For example, we have selected the DVD version of "Manos: The Hands of Fate" as the initial seed, resolved its ASIN (B0000AGWMF), and passed that number as the starting point for the population script. Full metadata for this seed item is recorded, including item attributes, browse node categories, URLs to product pictures, and editorial reviews. To find items that are related to the seed, we record the similar items that have been determined by Amazon, the Listmania! lists on which the item appears, the Amazon customers that have submitted a review for the item, Wishlists of those reviewers, and any items that are designated as accessories to the seed. As new items are discovered, their full metadata is recorded and they are added to the database. After completing this iteration for the seed, the script moves to the first related item that was discovered and explores its direct relationships. This cycle of discovering new items, recording their metadata, and tracing their relationships will continue until the script is interrupted or no new items can be found.

While the Amazon.com public API provides unlimited queries to developers, they restrict the request rate to one transaction per second. So, we begin construction of our local database by providing an initial seed, such as a movie or book title, and recording its corresponding metadata. Included within that metadata will be a at most five accessory items, a maximum of ten recommended titles, a maximum of ten Listmania! product listings in which that title appears, and up to 5,000 usernames gathered from customer reviews. By sending additional searches to the Amazon API, we can gather additional titles to store in the database by resolving reviewer names to their wish lists (which may hold up to 300 items), a maximum of 100 additional items that each reviewer has reviewed, and the Listmania! names to their title listing. Therefore, from a single initial product query, we can generate a maximum of roughly 2,000,265 additional related titles. However, since these entries will only have their metadata, we must then generate additional queries to trace the relationships for

each of these new titles to complete its description in the database.

Being restricted to one query per second, we began building our database early in the semester. After retrieving information from Amazon for about two months, we accumulated descriptions of about 1,000,000 products after exploring roughly 4,000 entries.

In reality, we rarely hit the maximum number of related titles on a single ASIN, as there was usually overlap between the various products and their related items. For example, if product A reports a relationship with products B and C, there is a good chance that at least one of the two products, B and C, will also report a relationship with A (and more likely, both will report such a relationship). Thus, a significant amount of “pruning” will occur during the building of the data source; since we are placing the results into a RDBMS as they are gathered, we will be able to eliminate unnecessary work by determining if the particular ASIN currently in the “queue” already has full metadata.

An interesting side effect of this phenomenon will be the ability to collect statistics on the rate of overlap between products, and then compare these rates across different criteria. Perhaps products categorized as literature have a very high rate of overlap between products, while technical and scientific writings have very little. We believe that interesting conclusions can be drawn from an analysis of these relationships, and the ways in which they change across various genres, subjects, etc.

Currently, this phase is implemented using the PHP scripting language and a MySQL database. The database schema created to store the product information is available in Appendix 1. The script pulls the XML data from the Amazon REST server, parsing the result, and stores individual records with full metadata. Since our database schema is enforcing a strict taxonomy of product and attribute types, the script is also handling the task of creating the types in the auxiliary tables if necessary (as happens the first time the script encounters a product with a particular relationship it has not yet seen).

Formulating relationships between products

Once we have built a sizeable local database to work with, we can begin generating aggregate relationships by querying product metadata stored within the database. While it will be a fraction of the entire body of Amazon product information, we should have sufficient entries in the database to extract a large number of relationships for examination.

In our schema, there are at least nine ways that an individual product entry can be considered related to another. First, any products appearing in the “Related Items” section are considered to be directly related to an entry (recall from the previous discussion of Amazon’s API that this section is roughly analogous to the “customers

who bought this also bought” section on their website). It is reasonable to expect that users who buy the same products can have similar interests, so this relationship allows a user to serendipitously discover products that they might enjoy.

Several relationships can be uncovered by extracting user names and ratings from the product reviews returned by the API. For instance, by retrieving the user IDs for customers that submitted reviews for a particular product, we can make additional calls to the API and find other items that they also reviewed. If the two items have a similar rating, the match is recorded as a reviewer relationship. Also, a user ID may also have an associated Wishlist. By extracting the user ID from the XML REST data, we can query the API for the corresponding user name, and then request the titles from the user’s wish list, if they have defined one on the website. This grouping yields a third relationship which relates items that appear on the same listing. Finally, if a user took the time to post a review on the web site and give a high rating, then we assume that they enjoyed the product and may be a good source for related items. This connection is called a Reviewer/Wishlist relationship. We consider this type to be “loose” – that is, there may be likely be a less obvious relationship between the products since people tend to create widely diverse lists of “wish” items, from books to movies to electronics to home and garden equipment. As a result, these relationships also have to potential to be the most interesting.

Two products may be considered related if both appear on the same Listmania! product listing. These associations are particularly valuable since the lists are defined by users and usually have an overarching theme that binds all the products on the list, such as “My Favorite Movies (The Ones I Own)” or “Why Zombie Flicks are Better than Real People v 1.0.” This relationship can be generated by extracting the list number from the XML REST data, then issuing a query to the API for each list. These relationships would be considered fairly “tight”, as the lists do tend to follow common themes or rules that would group products together (although it will be difficult, if not impossible for a computer to discover the metrics used by the human author of the list).

An additional explicit relationship between products exists if one item is an accessory of another. This relationship is determined by Amazon.com and is unidirectional only. For instance, a camera bag may be listed as an accessory for a digital camera, but the reverse relationship is not present and cannot be assumed.

The final three relationship types may be derived from the attribute and browse node data returned from Amazon. Attribute data consists of metadata that describes characteristics of the item, such as title, product category, and physical or conceptual attributes. Browse nodes are polyarchical category names that Amazon uses to classify its catalog. Examples of these category names includes director, genre type, and target audience. Using these two

sets of data, we can derive an Author/Creator relationship which associates products that have the same creator, which induces director or music composer. Also, books that share the same subject categories may be considered related. Finally, since browse nodes provide a good source of terms that describe an item, two items may be considered associated if they have a high degree of intersection between their assigned browse node categories.

Since there are several different categories of relationships between products, each representing a different type of relationship, we have decided to represent the relationship type by assigning to each a different weight, depending on the closeness of the relationships the type is describing. Additionally, if a single product appears in more than one relationship type from another single product, the weights of those relationships will be summed to create the total weight of the “closeness” of the two items. So if product A appears on product B’s “Related Items” list as well as two of B’s Listmania! lists, the total weight of the relationship between product A and B would be (related items weight) + (Listmania! weight) + (Listmania! weight).

For instance, if items on the same Listmania! product listing carry a weight of five and items on a “Related Items” list carry a weight of seven, then the cumulative weight of the aforementioned example relationship between A and B is seventeen. This total weight allows us to give a degree of relevancy between products, ultimately indicating to the end user which items are the most closely related to their target product, and which ones exist on the periphery. A possible implementation for a client that uses this data might allow the user to assign their own weights to the different types of relationships, thereby affecting which products appear most tightly linked.

First-stage link analysis

In order to produce the results described in the project overview, there are two stages of computation necessary: the first stage involves numerical analysis of the link types and quantities collected from the Amazon API; this data was not likely to change throughout the lifetime of the project, and thus was a candidate for batch processing and caching.. The second stage involves the statistical analysis of the textual metadata for each item (and the related items as determined by the first stage above); these calculations must be performed in real-time as they relied on user input and the user model maintained in the UI.

To process the link analysis and cache the results, we added a summary table to the database schema that pairs two products together in a undirected link relationship, and calculates the weights of the nine potential relationship types described in the project overview section. Once created, this summary table for our 1 million records contained over 250 million product pairs. The unanticipated scale of this single table presented unique difficulties for the platforms chosen for this project (MySQL on Windows). Ultimately, the sheer size of the table proved

impossible to manage in the scope of this semester-based project, and we were forced to calculate a small subset (~1 million records) to represent the cache functionality.

Statistical Analysis Component

The end result of the ingestion mechanics is a vast database consisting of thousands of items related to each other in a variety of ways. Since items can be related many criteria, several of them fairly weak, displaying all the relationships a given item has is unfeasible. Additionally, simply throwing those results into a flat list is both unreadable (due to the number of relationships) and completely fails to account for the strength of similarity. For this reason we apply filtering to reduce the number of relationships, and clustering to organize the remaining relationships in a meaningful way.

The process begins with a seed item that is currently being viewed. The interface back-end selects the set of related items from our database, and filters these items based on the user profile. The user profile contains lists of items the user has expressed interest in and a list of items the user has expressed disinterest in. The “interesting” related items are automatically included in the return set (space allowing) and the “uninteresting” items are excluded. Thus, the vast collection of related items to the seed item is pared down and made to fit the user profile. While not a statistical method, filtering is necessary for the next step in the process: clustering.

Statistical clustering is performed on the filtered set of related items. The few things necessary for implementing hierarchical clustering of any kind are a set of items to cluster and a way to compare those items with each other. In our case, the set of items is the aforementioned filtered set, and their comparison is done by considering the item’s attributes.

Each attribute of an item, like title or director, is considered to be a feature. In order to be able to compare items based on their features, those features are turned into term vectors based on the item’s attributes versus the attributes of all other items in our dataset. If a particular attribute is missing, its term vector is empty. This assures that all items have the same number of features (even if some are null) and that those features are term vectors of the same dimension. This allows for a direct comparison of any two items.

We chose the dot product as the distance metric for comparing items together, as suggested by Sulton. [6] Once the distance metric has been established, the application of a clustering algorithm is fairly straightforward. Once both the filtering and the subsequent clustering are complete, the recommendations are presented in a hierarchy of related items.

The clustering algorithm operates as follows. Similarity between two items is calculated as follows:

$$S_{ij} = \sqrt{\sum_{attributes} (TV_i \cdot TV_j)^2}$$

For a given seed item, based on the ‘Branching Factor’ b , most similar ‘ b ’ items are connected and are populated as cluster centers. This process is done recursively with the cluster centers found in previous phase as new seeds.

Visualization Component

The representation mechanism we chose for the display of recommendations is a user-controllable directed graph, where each node represents a product under consideration, and the edges represent the relationships (and associated relationship properties) between the products. To avoid the low-level work of creating classes to handle graph generation and manipulation, we have decided to use the open source graph visualization library Prefuse.

The Prefuse library includes classes to represent aggregate nodes, which are meant to encapsulate groups of nodes. These aggregate nodes will represent the clusters of items formed by our HAC algorithm.

In addition, users are able to influence their profiles from this graph view. By right clicking on an item node and selecting “Add to favorites (B000BW1ESO)” a user may add the item to the *interest* list in their profile. The item will then influence future recommendations, thereby allowing customization of the system that is very specific to the user. Alternately, the right-click context menu will also show an option labeled “I don’t like this item (B000BW1ESO)”, that will remove the item from the user’s *interest* list if it is there, and add the item to the user’s *exclusion* list (see Appendix B).

The interface also provides several adjustable properties of the graph, including the graph size (exposed on the interface control panel as a slider bar), an adjustable level of clustering (ranging from 1 cluster containing all related items to 1 cluster for every single related item), and checkboxes which control the use of full text metadata (such as editorials and reviews) in the clustering algorithm and disabling the clustering completely. The default levels of filtering and clustering will be specified in the user profile (see Appendix B).

To access more information about a particular product displayed in the graph without re-centering the graph and encumbering the expense of calculating the clustering relationships of the products, hovering the mouse briefly above a graph node will induce a popup box that contains basic metadata information, as well as a visual surrogate of the product, drawn from the Amazon API (usually a scanned image of the book or album cover). Additionally, full metadata (including a link to the Amazon record and the Amazon-provided editorial review) is provided in an “info panel” directly underneath the control panel on the main interface. This info panel always contains the metadata for the current focus item.

A text field allows direct entry of a product ASIN to serve as the focus of the graph. A single button is provided to re-visualize based on a new ASIN or new filtering and clustering levels. Alternately, the user can traverse the product links entirely within the graph, by left-clicking on an item to re-focus the graph around the new selection. The filtering and clustering levels used for any new graph re-visualization will be the currently-assigned values in the widgets on the control panel.

User Profile Component

In order to maximize the sharing of knowledge between the system and user, our goal was to fully integrate the user model into both the underlying system, and the UI. Therefore, the user profile had to be completely visible and editable throughout the browsing experience. The ability to add or remove items from their interests and exclusion lists—which comprises the bulk of the applications knowledge of the user—is directly accessible from the graph display, and not shunted off to a hidden menu. The current lists are easily available from a tab always visible at the top-left of the interface, and the user can review and modify the machine’s current understanding of their interests, which will allow them to understand how their profile determines the recommendations they receive. By revealing to the user the machine’s knowledge and design rationale, we hope to overcome the lack of mutual understanding described by Suchman [4].

ANALYSIS OF RESULTS

In observing the resulting clusters and item relationships, we noticed many obvious similarities between items within clusters. For example, when looking at the B-rate horror genre of cinema, we noticed that our items tended to be clustered by director and by plot. Movies by Ed Wood, a near legend in this genre, would form their own cluster, with the rest roughly breaking up by subject. In that regard our clustering mechanism works well.

We did, however, also notice occasional results that are clustered oddly. For example, a comedy would get clustered with a horror flick because of a similarity in title or reviews. This is addressed in the future work section below.

FUTURE WORK

Since all of the explicit and derived relationships between Amazon products are summarized for all possible product pairs, a future refinement to the project better utilized the relationship “weights” to generate product recommendations. Our current prototype retrieves item pairs from the summary table and sends the results to the clustering algorithm, regardless of the calculated weights. However, better recommendations may be returned by the clustering algorithm if the item pairs used are considered more “weighted”, or have higher numbers of relationships

between them. Since the sheer size of the summary table makes this impossible using MySQL, we will move the back-end database to a more robust program that can quickly handle queries on large tables.

Additionally, we would like to allow the user to search using a title or product name rather than an ASIN. Our current prototype requires a product ASIN in order to generate a new graph. Since this is not a practical way to search for items, we plan to integrate a search mechanism that resolves a title or name to the necessary product number.

Finally, an overhaul of the statistical clustering component may yield improvements in the runtime of the algorithm and the quality of the returned results. We intend to improve time performance of the clustering algorithm [$O(n^2)$] and use term vectors based on “atomic text units.” For instance, “Ed Wood” is currently parsed to “Ed” and “Wood.” Therefore, movies directed by Ed Wood will match others that starred “Ed Asner”, which may or may not be accurate. Modifying this to consider the entire name as an atomic unit may generate more accurate results

CONCLUSIONS

We believe that the implicit relationships derived from our information harvesting, combined with the basic clustering on term vectors provides a significant improvement over the recommendations provided by Amazon. With additional refinements, product recommendations can be generated that do not rely on a single dimension, such as product sales. By aggregating the relationships harvested from the API, we can generate a set of product recommendations that take a variety of sources into consideration, such as human generated associations based on themes, preferences, and keywords.

Additionally, the graph-based interaction provides significant UI advantages over the traditional bulleted list representation currently used on Amazon’s webpage. From our point of view, the results generated by our techniques demonstrate some very interesting groupings that would have been difficult to discover on Amazon.

The inclusion of the user profile introduces an intelligent component to the project. Users are able to indicate which items they do and do not like. Based on this set, the system can build a profile that describes user preferences and modify its recommendation calculations accordingly.

REFERENCES

1. Linden, G., Smith, B., York, J. 2003. Amazon.com Recommendations: Item-to-Item Collaborative Filtering in *IEEE Internet Computing*, pp. 76-79.
2. Amazon Web Services.
<http://www.amazon.com/gp/aws/landing.html>
3. Creel, J., Lu, J., Mikeal, A., Speight, C. 2005. A Hypertextual Augmentation of the Amazon E-Commerce Service (ECS) 4.0.
4. Suchman, L. Plans and Situated Actions: The Problem of Human-Machine Communication. Cambridge, 1984.
5. M. Sahami, S. Dumais, D. Heckerman, and E. Horvitz. A Bayesian approach to filtering junk email., AAAI Workshop on Learning for Text Categorization, July 1998, Madison, Wisconsin. AAAI Technical Report WS-98-05.
6. Salton, G, Wong, A. Generation and search of clustered files. *ACM Transactions on Database Systems (TODS) Volume 3, Issue 4*. ACM Press, New York, 1978.
7. Yaari, Y. Segmentation of Expository Texts by Hierarchical Agglomerative Clustering. *Proceedings of RANLP'97*. Bulgaria, 1997.
8. Fasulo, D. An Analysis of Recent Work on Clustering Algorithms. Technical Report: 01-03-02, Department of Computer Science and Engineering, University of Washington, 1999
9. Prefuse: interactive information visualization. <http://prefuse.sourceforge.net/>.

Appendix A: Database schema

accessories				
field	type	Key	default	extra
initial_item	lint(10) unsigned	PRI	0	
related_item	lint(10) unsigned	PRI	0	

attributes				
field	type	Key	default	extra
id	int(10) unsigned	PRI	0	auto_increment
value	varchar(50)	MUL	NULL	

banned_browse				
field	type	Key	default	extra
value	varchar(50)	PRI		

browse_category				
field	type	Key	default	extra
id	int(10) unsigned	PRI	NULL	auto_increment
value	varchar(50)	UNI		

editorial				
field	type	key	default	extra
id	int(10) unsigned	PRI	0	
review_num	tinyint(3) unsigned	PRI	0	
review	text		NULL	

item_data				
field	Type	key	default	extra
item_id	int(10) unsigned	PRI	0	
attr_id	int(10) unsigned	PRI	0	
value	Text	PRI		

images				
field	Type	key	default	extra
id	int(10) unsigned	PRI	0	
size	enum('small','medium','large')	PRI	NULL	
link	varchar(90)			

item				
field	Type	key	default	extra
updated	Timestamp			
id	int(10) unsigned	PRI	NULL	auto_increment
asin	varchar(10)	MUL	NULL	
amazon_link	Text		NULL	
started	enum('y', 'n')		n	
completed	enum('y', 'n')		n	

item_attributes				
field	type	Key	default	extra
item_id	int(10) unsigned	PRI	0	
attr_id	int(10) unsigned	PRI	0	
value	text	PRI		

item_browse				
field	type	key	default	extra
item_id	int(10) unsigned	PRI	0	
browse_id	int(10) unsigned	PRI	0	

listmania				
field	type	Key	default	extra
id	int(10) unsigned	PRI	NULL	auto_increment
value	varchar(20)	MUL	NULL	

listmania_items				
field	type	key	default	extra
list_id	int(10) unsigned	PRI	0	
item_id	int(10) unsigned	PRI	0	

log				
field	type	key	default	extra
number	int(15) unsigned	PRI	NULL	auto_increment
time	timestamp		current_timestamp	
message	text		NULL	

reviewer				
field	type	Key	default	extra
id	int(10) unsigned	PRI	NULL	auto_increment
value	varchar(20)	MUL	NULL	

reviewer_items				
field	type	key	default	extra
list_id	int(10) unsigned	PRI	0	
item_id	int(10) unsigned	PRI	0	
rating	int(2) unsigned	PRI	0	

similar				
field	type	Key	default	extra
id	int(10) unsigned	PRI	NULL	auto_increment
value	varchar(20)	MUL	NULL	

similar_items				
field	type	key	default	extra
list_id	int(10) unsigned	PRI	0	
item_id	int(10) unsigned	PRI	0	

summary				
field	type	Key	default	extra
id1	int(10) unsigned	PRI	0	
id2	int(10) unsigned	PRI	0	
listmania	smallint(5) unsigned	MUL	0	
similar	smallint(5) unsigned	MUL	0	
wishlist	smallint(5) unsigned	MUL	0	
reviewer	smallint(5) unsigned	MUL	0	
browsenode	smallint(5) unsigned	MUL	NULL	
revwish	smallint(5) unsigned	MUL	0	
creator	smallint(5) unsigned	MUL	0	
subject	smallint(5) unsigned	MUL	0	

wishlist				
field	type	Key	default	extra
id	int(10) unsigned	PRI	NULL	auto_increment
user_id	int(10) unsigned		0	
value	varchar(20)	MUL	NULL	

wishlist_items				
field	type	key	default	extra
list_id	int(10) unsigned	PRI	0	
item_id	int(10) unsigned	PRI	0	

Item Info Manage Profile

My Interests

-  The Wasp Woman
-  Manos, the Hands of Fate
-  The Killer Shrews

Excluded Items

-  The Alligator People
-  Death Curse of Tartu / Sting of Death

