

SOMR — Self-Organizing Map Recommender

Joel Bennett

November 2008

submitted to the faculty of the Department of Computer Science, in partial fulfillment
of the requirements for the degree of Masters of Computer Science in the

Golisano College of Computing and Information Science

Rochester Institute of Technology

Rochester, New York

Jessica Bayliss Principal Adviser

Zack Butler

Leon Reznik

Abstract

This project comprises designing and implementing a hybrid recommender system for web-pages which uses data from a social tagging system to recommend interesting items to users. For this implementation, the tagging data comes from del.icio.us, the oldest and largest public social bookmarking or tagging system for web pages. The system clusters items using a pair of self-organizing maps (SOM) networks and allows users to see the system's evaluation of their region of interest, or set their own regions.

The focus of this project was the web-scraper for gathering tagged URLs from del.icio.us, and the recommender system. The SOM networks are built using the GHSOM implementation, and several recommenders were built to compare results: one using a single map for URLs *and* users and the other using separate maps to compare the relative quality of the recommendations.

Contents

1	Background	1
1.1	Recommender Systems	1
1.1.1	Content-based Systems	1
1.1.2	Collaborative Systems	2
1.2	Social Tagging	3
1.3	Self-Organizing Maps	4
1.3.1	Semantic Word Clustering	5
1.3.2	Code-book model	6
2	SOM Recommender System	7
2.1	Similar Work	7
2.2	Improvements	7
2.3	Choosing Map	8
2.4	PowerShell UI	9
2.5	Design Overview	10
2.5.1	Gathering Tagging Data	11
2.5.2	Generating 2D Maps	11
2.5.3	Making Recommendations	12
3	System Design	13
3.1	Gathering Tagging Data	13
3.2	SOM Networks	15
3.3	Recommender	16
3.3.1	Mapping the User's Interests	17
3.3.2	Making Recommendations	18
4	Project Outcomes	19
4.1	Good Results	19

4.2 Shortcomings	20
4.3 Future Work	21
4.4 Deliverables	22
Annotated Bibliography	23

1. Background

1.1 Recommender Systems

In the information age, one of the biggest problems facing users — and researchers in particular — is the need to find the useful and interesting items in the overwhelming flow of research papers, conference reports, web pages, emails, instant messages, etc. Information filtering (IF) is focused on analyzing documents and selecting the best documents for users automatically [Good et al. \[1999\]](#). Recommender systems (RS) can be thought of as information filters which focus on profiling individual users so that items can be picked out which are specifically relevant to them. In some sense they are therefore more specific, but their application is more general, because they are applicable not only to the field of information and documents, but to everything from on-line stores to music players [Burke \[2002\]](#).

The initial recommenders were passive systems like search engines which simply return lists of items matching keyword terms, but modern systems are expected to “learn” the users’ areas of interest as well, rather than relying on user-specified keywords. There are two major classes of recommenders, based on the differences in their algorithms, and additionally there are hybrid systems like FAB [Balabanović and Shoham \[1997\]](#) which use both. Content-based recommenders find items with similar content to other items that the user has previously rated highly. Community-based systems (also called collaborative filters) recommend items based on the ratings that similar users have given them.

1.1.1 Content-based Systems

Content-based filters are by far the more mature technology, dating back to work done by H. P. Luhn at IBM in the 1950’s. Luhn introduced the idea of creating profiles for individual users which could be used in an exact match system to produce reading recommendations, and the user’s actual reading habits (the system was designed for use in a physical library) would then be used to update the user’s profile [Douglas W. Oard \[1996\]](#). The main distinction of content-based systems is that they treat each user as though (s)he were working completely independently. Disregarding information from other users,

if present, a content-based system filters and makes recommendations based on the items themselves, and knowledge of the target individual.

Content-based systems are more closely related to the field of information filtering than collaborative filters, as they require the items to have intrinsic content that can be used in filtering. This becomes a major drawback to such systems, since some items we might wish to apply recommender systems to simply have no intrinsic content. Thus, content-based systems are primarily document classifiers, and don't generally work with other types of items like movies and restaurants. Another problem with content-based systems is that they may be unable to generalize sufficiently: since they're designed to return items similar to those already rated by the user, they may be too restrictive [Balabanović and Shoham \[1997\]](#), and the user may miss out on interesting items outside the range of documents they have already rated.

1.1.2 Collaborative Systems

The second wave of information filtering research was born of the idea expressed by the Information Lens team: that social factors could be effective in information filtering, including ratings from other users [Malone et al. \[1987\]](#). This idea was not actually implemented in Information Lens, so it was not until the Xerox Palo Alto Research Center (PARC) developed Tapestry that some level of social filtering was actually implemented [Goldberg et al. \[1992\]](#). Probably the most well known example of a collaborative recommender system is the GroupLens system from the University of Minnesota which was designed to recommend USENET articles based on ratings from all users.

The GroupLens project pushed collaborative filters forward in many ways, and the team working on it continues to do so today. They also identified the core problem which plagues collaborative filters. Variously known as the start-up problem, the ramp-up problem, or the cold start problem, it is basically a chicken-and-egg question of which comes first, users or items?

- **New users** have no correlation with existing users and thus get no recommendations. They must rate articles in order for the system to be able to recommend articles, but may abandon the system before they start receiving recommendations because they don't see the benefits.
- **New items** have no ratings and are therefore not recommended. This is particularly troublesome because it does not help to have these items rated if the users doing the rating are new users. That is, an early rating doesn't improve the item's chances of

being recommended, nor does it improve the user’s chance of being matched with other users.

The SOM recommender system we built in this project is essentially a collaborative system, but to some extent it behaves as a hybrid system because it is based not just on discovering similarities between users, but also on discovering similarities between documents based on the tags/categories assigned to them (which are presumed for the most part to represent the content).

1.2 Social Tagging

The SOMR system effectively side-steps the best known problems for both types of systems by using the data already collected from thousands of users by social tagging sites. As previously stated, our implementation works with data gathered from del.icio.us, but the core methods could be applied to any social tagging system, including language-specific bookmarking sites, specialized research sites, and presumably systems that aren’t even web page bookmarking sites. This data gives us a pre-existing set of users and items, and is basically all about items that not only have their own intrinsic characteristics, but have certain characteristics ascribed to them by the users in the form of “tags.”

Tagging is the act of assigning free-form keywords to items with no specific list of allowed keywords, no built-in taxonomy, and generally no limits on the length or number of keywords that can be assigned to an item. Social tagging allows users to assign keywords to items and share them with others. The key underlying assumption in social tagging is that, given this freedom, different users will sometimes use different words to describe the same item, and that the collection of keywords used by all users is helpful in both finding items and browsing similar content. The SOMR system is also based on this theory, and presumes that we can use this tagging data to build both a collaborative and a content-based recommender system (taking the tags to be a standin for the content).

Bookmarks have long been used by web surfers as a way of saving a local reference to specific pages they are interested in [Millen et al. \[2005\]](#). The desire to spread this solution from single users to social groups and enterprise teams has recently lead to a rise of web-based systems for bookmarking. These systems generally reject the traditional hierarchical “folder” hierarchy in favor of tagging systems, and so become social tagging systems where the *items* are web pages.

At first, these systems were sufficient on their own to allow users to explore other users’ bookmarks via the keywords and the ability to track users who bookmarked pages

you considered interesting. However, with some of these systems (like [del.icio.us](#)) having hundreds of thousands of users, and many users racking up tens of thousands of bookmarks, the task of finding interesting items or users is getting out of reach for the average user. The goal of our system, therefore, is to deduce the user's preferences and find items that would be of interest.

1.3 Self-Organizing Maps

The most important piece of any recommender system is the clustering algorithm. In order to make recommendations in either the content-based or collaborative models, the system must group like items or users together. To cluster the items in our system we require a clustering algorithm which can take into account both contents (as represented by categorizing tags) and user ratings, and which can also provide some sort of *measurement* of similarity. The clustering algorithm should be unsupervised to avoid the need for direct feedback, and should work well with high dimensional datasets, since the number of possible tags in our bookmarking system is essentially unlimited, as is the number of users.

One of the most popular unsupervised clustering algorithms in recent study is the Self-Organizing Map (SOM) neural network algorithm. It has attracted much attention, including over 5000 scientific papers that use or analyze the algorithm in the last 20 years, with hundreds of researchers using the algorithm for everything from pattern recognition to information theory and science [Oja et al. \[2003\]](#).

The reasons for the popularity of Self-Organizing Maps are simple: not only is it one of the most successful unsupervised clustering algorithms, it takes high-dimensional inputs and represents them in an easy to understand two-dimensional way. A SOM clusters items which are similar so they end up topologically near each other in the resulting map, thus exposing the existing relationships between items with high-dimensional descriptions even when there is so much data that the relationships are not otherwise discernible [Kohonen et al. \[1996\]](#). Since they work on high-dimensional vectors, they are naturally usable for our current task, and they create a simple distance metric that can be used to determine similarity.

One of the highlights of the SOM algorithm, and certainly one reason for choosing it for this task is that on top of providing reasonable unsupervised clustering, it provides an interesting *visual representation*, and a useful way to allow exploring of the results. The research group at Helsinki University of Technology has encouraged

this by providing some interesting interfaces for visualizing and navigating the resulting maps, and the best is probably the web-based interface to WEBSOM available at <http://websom.hut.fi/websom/> which provides several levels of detail in a zoom-like fashion, and the ability to pan page-by-page around the map at each level of detail. Other good examples of interfaces include the Growing Hierarchical SOM (GHSOM) [Rauber et al. \[2002\]](#), SOMLIB digital library, and LabelSOM algorithms developed at the Vienna University of Technology and available at <http://www.ifs.tuwien.ac.at/~andi/somlib/>, where two examples of it's use are with music in the [SOM enhanced jukebox](#) and countries (based on data from the CIA World Factbook) [using GHSOM](#). Originally we intended to create an improved visualization for SOM maps based on the interface for web maps such as Google maps, and to provide a more fluid zoomable UI which would expand details at certain levels of zoom and allow users to see their areas of interest visualized – and even modify them. This however, turns out to be a whole project's worth of work all on it's own, so our current implementation relies primarily on the HTML representation of the maps that is created by GHSOM, leaving an improved visualization for a future project.

1.3.1 Semantic Word Clustering

It is worth noting that in these statistical models, synonyms cannot be accounted for — that is, words which have similar meanings end up behaving the same as any other pair of words [Honkela et al. \[1998\]](#). One approach for dealing with this is semantic word clustering, which creates clusters of words with similar meanings. Using a clustering algorithm such as the SOM neural network algorithm already developed for document clustering, we can group similar words together with the goal of eliminating synonyms and even reducing the dimensionality of the classification space. The original WEBSOM system used this method simply to keep the dimensionality reasonable. They did an initial SOM word clustering based on context (the frequency of appearance of words in the context around a word was the input vector). The documents are then encoded by mapping them onto the word map, creating a histogram of the “hits” to each node in the map [Lagus et al. \[2004\]](#), [Honkela et al. \[1998\]](#).

This is one area where improvements could be made to SOMR. Given the amount of data available from del.icio.us, for instance, it should be possible to create an initial word map of the tags, based on tag co-occurrence on URLs, allowing synonyms to be clustered together and treated as a single dimension. It would risk marking true sub-category tags as synonyms of their super-set, but this tag map could then be used to encode the individual documents for placement into a document map. The current implementation

does not concern itself with this, preferring to simply map the documents based on the "raw" users and tags, and use both maps together to generate a weighted combined rating, rather than to generate a second-level document map.

1.3.2 Code–book model

Luo introduced a variation of the semantic word clustering that works at the level of letters. This method is simpler to apply to this project than to the general case, because the tags we're dealing with obviate the need for most of the word-stemming and other pre-processing used by Luo, since the tags are already a relatively small set of relevant words. The code–book system creates a SOM network trained on the frequency of letters appearing at a certain position in the words, essentially a code–book for the character patterns. It then creates a second intermediate SOM based on words for each document (similar to how WEBSOM creates its second SOM based on the first): each word is fed through the first level SOM and the inputs for the second level SOM are the histogram of "hits" —the neurons which fire in processing that word. Finally, the output–level SOM is trained on the documents based on a histogram of hits in the second level SOM Luo [2003]. Initially we believed this method would most likely not work directly because of the somewhat smaller set of words that our system has available in the form of tags, but Luo reports very good results in his experiments, and the number of words in our system turns out to be surprisingly large compared to the useful dimensions for the SOM. When compared to the random projection simplification of the vector space that was used by Kohonen's latest WEBSOM algorithm — an adaptation Kohonen found necessary for massive document sets — Luo reports this code–book model has nearly doubled the accuracy from 56% to 90% Luo [2003].

2. SOM Recommender System

The SOMR system is a recommender system based on Kohonen’s Self-Organizing Map algorithm. We theorized that we could use data from a social tagging system to recommend interesting web pages to users and in fact, to emulate the benefits of hybrid systems using only the data that was *already collected* by a social tagging system. We designed and implemented a recommender system which uses a pair of SOM neural networks to create two-dimensional maps of the users and the tags as they relate to the items, and conceived a method for mapping “new” items (URLs in our case) onto both of these maps. We provided the HTML-based visualization that allows users to see the system’s evaluation of their regions of interest. We created a recommender which ranks URLs based on these two maps by calculating their distance from the loci of the user’s interest, and combining these two “dimensions” to produce a single weighted score.

2.1 Similar Work

The essential idea behind this project is that a SOM can be created based on keywords which will map both items (web pages) and users, and that this will be sufficient to allow interesting recommendations to be made. Previous research has been done to create a recommender using SOM networks, but based on the scores assigned by users [Roh et al. \[2003\]](#), and recommender systems have been created in the past based on the tags in tagging systems, but they resulted in only 40 to 60% accuracy in recommendations [Mathes \[2004\]](#).

2.2 Improvements

The SOMR system sets out to improve upon existing recommenders in several ways. Since we simply use keyword tags assigned by users during bookmarking, the system will not require that users “rate” items, which means that most users can get “jump started” into the system by simply importing their browser bookmark file. This system only recommends “fresh” pages to users, based on pages that other users are *currently*

bookmarking, rather than on all pages ever bookmarked, and so will avoid the mistake of recycling old news. The system will take into account both the similarity of keywords in use, and of the users who've bookmarked the page to obtain a higher degree of certainty in recommendation.

Additionally, this system will not be forced to make a certain number of recommendations, rather, it will have an adaptable “quality” setting, as well as a “maximum” setting. This means that if there are no good pages, it will not be forced to recommend bad ones, but will allow the user to get more recommendations by relaxing the quality level. Users who's regions of interest are diverse, or extremely popular may get overwhelmed, so they can set the maximum number of recommendations they want, and the system will pick from within their quality limit, the best links that will fit in that count.

2.3 Choosing Map

The most time-consuming portion of the effort of creating the SOMR system was finding a way to feed the data from a tagging system into the neural network in such a way that it would produce reasonable results. We spent a lot of time experimenting with various implementations of SOM algorithms, and different ways of using the data from del.icio.us to generate maps: mapping tags to tags, tags to users, tags to urls, urls to tags, users to urls, urls to users, and so on. For each association class we were trying to find a way to represent the data as weighted input vectors for the neural networks in order to produce maps that appeared logical to the user and were well distributed and thus useful for our information filtering task.

We settled fairly quickly on the GHSOM algorithm, which has been well studied and documented in many publications by Rauber, Merkl, and Dittenbach et. al. [Rauber et al. \[2002\]](#), and we were able to use their existing C++ implementation with very little modification (except what was required to get it to compile in Visual C++ 9). We did spend a lot of time testing and tweaking the various settings available through its configuration files, and testing the different maps we could create based on our data, and polishing the way that we generate the appropriate text input files for GHSOM, but we didn't have to modify the core GHSOM algorithm much at all.

We originally intended to use the full hierarchical features of GHSOM, but found that creating a “score” for the items against the maps was overly complicated by the hierarchical dimension, since the hierarchy isn't an actual third dimension, and the “distance” attributable to layers of hierarchy varies for each place where an extra level is pushed out.

Essentially: we're using distance calculations for ranking tagged items, and the hierarchy makes these calculations much more complicated and does not yield any improvements in the rest of the system. As a result, although we used the "Growing Hierarchical" version of the Self-Organizing Map algorithm, we actually only grow the map in two-dimensions for each set of data.

2.4 PowerShell UI

All portions of the current SOMR system were written as command-line applications, and with the notable exception of the GHSOM project, they are all written as PowerShell cmdlets. A PowerShell Cmdlet is basically any .Net class which derives from `PSCmdlet` or `Cmdlet`. There were many benefits to using PowerShell for these cmdlets, but we found three main benefits of using the cmdlet framework over writing custom command-line applications:

- **Input Parsing** is handled by PowerShell, so you need only mark public properties of your cmdlet class with the `Parameter` attribute and they become parameters to your cmdlet which PowerShell will parse and convert to the correct type on the command-line.
- **Scripting and Pipelining** is possible with any cmdlet in PowerShell, so the commands and output objects you expose this way can be reused and mixed together at the command-line and in interpreted scripts with marginal speed loss, enabling greater flexibility when trying new ideas.
- **Object output** is what makes PowerShell unique, but because it expects objects to be output, there is essentially no need to format the output of your cmdlet: you just `WriteOutput` the object. In cases where some degree of customized output is desirable, you can specify the format in an XML file which can be modified on the fly (without recompiling) and even edited by the end user.

After creating PowerShell cmdlets for this project, we note that the ability to write a simple class, and then execute it in scripts and from the command line, together with the automatic argument parsing and type-casting, etc. makes PowerShell cmdlets quite useful for research and development. Because the overhead is so minimal, they reduce the distractions of creating a user interface to a minimum, and allow us to focus on the actual research goals. Additionally, because they can take any .Net objects as input parameters,

it is possible to create cmdlets which work in a pipeline, or accept the output of other cmdlets as input, allowing still greater experimentation. The overhead of the cmdlet class is minimal, and the classes are still usable for all other purposes — on top of being able to invoke them within PowerShell, they are still classes in a library, and can be reused within any .Net application, including graphical rich client or web applications.

2.5 Design Overview

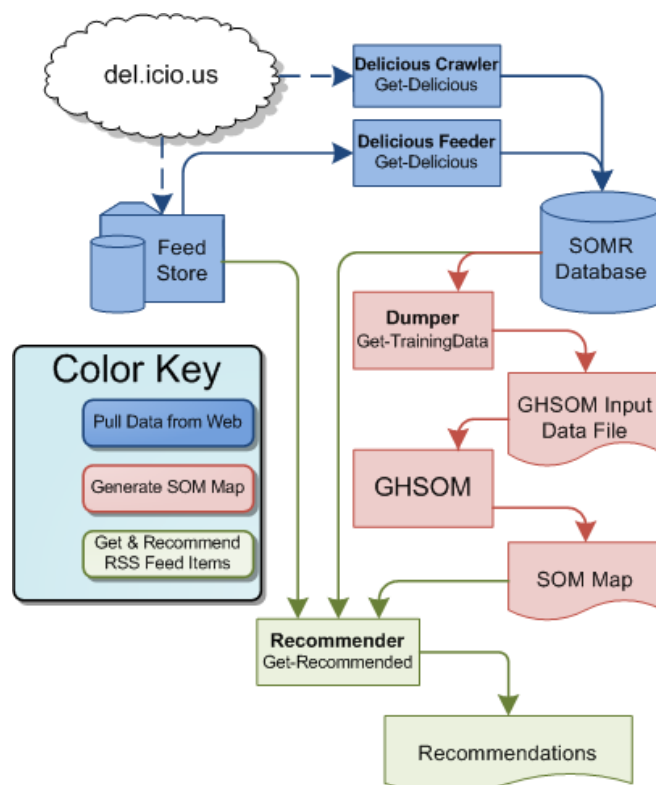


Figure 2.1: Data Flow

There are basically five parts to the design, as can be seen in 2.5: a crawler and a feed reader which fetch data from del.icio.us, a dumper which prepares the data for the GHSOM algorithm, the GHSOM binary itself, and the recommender. The output from each step is consumed by the next step, with only the final recommendations being output to the user (although the “map” output of the GHSOM is also viewable, of course). In the 2.5, we’ve color-coded these five parts into three tasks to highlight the way this could be implemented by del.icio.us or diigo or some other social tagging system.

2.5.1 Gathering Tagging Data Overview

To accomplish the first task of gathering the tagging data from our social tagging source, we created a web scraper. The scraper collects web pages from del.icio.us, parses them into valid XML (the del.icio.us site doesn't feed valid XHTML or even valid HTML), and then extracts user names, URLs, and tags. We created the SOMR database to store the data gathered by the scraper until it was needed for use in training the recommender or actually making recommendations.

Obviously this data-gathering portion of the system (color coded blue in 2.5) would be superfluous for any existing social tagging system — *they are* the source of the data. In our case, the crawler is designed to be able to retrieve all of the data about a single user and every page they have tagged (including the other users who have tagged them). It can branch out and crawl each of the discovered users, and so on. The same crawler is also used to gather the information about single pages, which is used on the "recent items" which are retrieved from the del.icio.us feed, to gather information about the pages in order to filter them.

2.5.2 Generating 2D Maps Overview

The bulk of the processing which the SOMR system does is performed in the second step: training of the SOM neural network, to create the maps which the recommendations are based on. In order to do this, we first have to get the data from the database into the vector format required by GHSOM. We created a data output adapter we referred to as the Dumper, which we have now implemented as the `Get-TrainingData` cmdlet. It is a series of SQL queries wrapped in loops that transform the data and filter it. From all of the available data we use only the top 2000 or so items, tags, and users which we have the most data about, because we found during testing that the full dataset takes far too long to process in our environment, so for training the neural networks we chose to use whichever items give us the most information based on what we have gathered from del.icio.us up to that point.

The red portion of 2.5 represents that step of dumping the data as well as the process of training the SOM neural networks. The resulting maps are then stored for use by the recommender. The actual processing by the GHSOM algorithm can take hours (as long as 11 hours running both maps concurrently on a dual-core E6400 Intel process), so we designed the system to do this infrequently, and rely on the generated maps for making recommendation for an extended period of time before re-training them. It would

obviously be helpful to be able to incrementally (re)train the neural net, but it is not actually necessary for the system to function.

2.5.3 Making Recommendations Overview

The final step is the actual Recommender. It also consists of two steps: first determining where the centers of interest for the user are, and then processing the available items. It takes the "recent items" from the del.icio.us feed, with all the information about them (previously fetched by the Delicious Feeder), and maps them onto the trained neural network, and then generates scores for them based on their distance from the user's areas of interest, and outputs them with the rankings.

3. System Design

3.1 Gathering Tagging Data Details

In an ideal world, a recommender system like this would have direct access to the tagging databases of a social tagging site, but since creating such a site was outside the reasonable scope of this project, and we weren't able to gain access to data from any of them directly, the first portion of the project became the task of downloading, scraping, storing and crawling the del.icio.us web feeds.

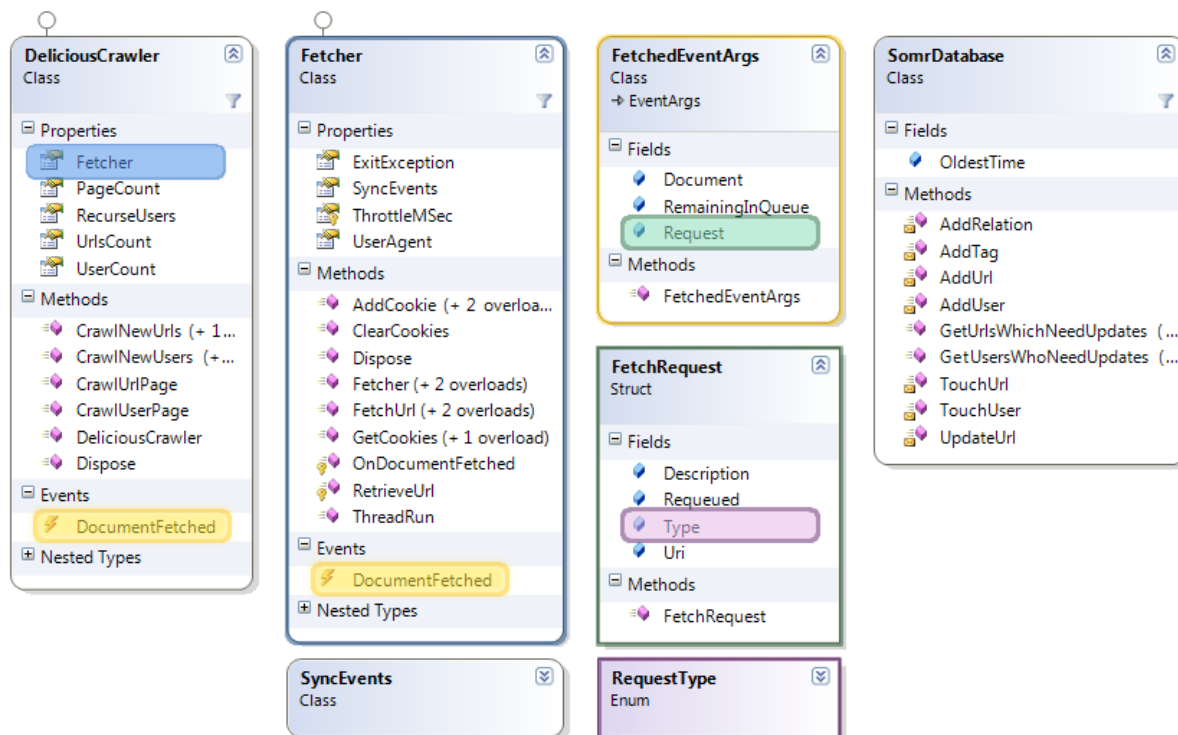


Figure 3.1: Scraper Class Diagram

The web scraper portion of this gathering system is in the `DeliciousScraper` assembly and namespace. We created a multi-threaded HTTP downloader class called `Fetcher` with the intent of speeding up the scraping process on multi-core systems. However, In the case of scraping del.icio.us we found that using multiple threads was basically useless because they strictly enforce a maximum rate of downloads per IP address which is much

slower than even a single thread could manage. Instead of using multiple threads, we had to add a throttle setting ($ThrottleMSec$) which specifies an artificial delay between requests.

The other major features of the `Fetcher` itself are that it supports HTTP and HTTPS, cookies, and basic HTTP authentication, and allows queuing multiple web pages for downloading at whatever rate the throttling allows. In addition, it retrieves web pages as XML documents by using an SGML parser. The `SGMLReader` class is a Microsoft sample library, Lovett [2008] which we used virtually unchanged to parse HTML documents (as well as invalid XHTML documents) and convert them into XML.

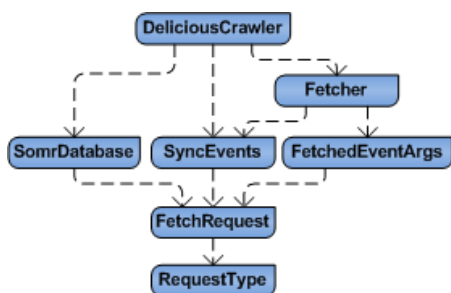


Figure 3.2: Scraper Dependency

The `Fetcher` creates a thread at construction which monitors the queue. The queue is thread-synchronized and the threads communicate using an `EventWaitHandle` to notify of new queued items, and a `DocumentFetched` event to notify of fetched items. The `FetchUrl` methods add requests to the queue, and the `Fetcher` thread(s) process items from it at whatever speed their throttling allows.

The `del.icio.us` crawler basically wraps this `fetcher` and adds actual parsing of the downloaded documents.

Since the documents are converted to XML documents, the crawler can do its parsing using XPath. The `DeliciousCrawler` class simply has separate parsing methods for the two main types of pages we need to scrape: pages which list the URLs bookmarked by a user, and pages which list the users who’ve bookmarked a URL – each has its own series of XPath lookups. The crawler also has to loop through a “next page” algorithm to make sure we have scraped all of the pages for the specified user or web page URL, because `del.icio.us` limits the number of records per page, so in some cases we have to fetch dozens or even hundreds of pages to get all of the information about a popular URL or a prolific user.

The `del.icio.us` crawler stores the data it retrieves in the SOMR database. This database has only four tables. `Users`, `Urls`, and `Tags` each map the text to an integer id and the `Users` and `Urls` track when the corresponding `del.icio.us` page was last crawled. The `Relationships` table maps a single URL, user, and tag together by their ids, along with a tag order (although we aren’t currently using in our analysis, the tag order is preserved by `del.icio.us` and has been shown to be significant in determining the importance of the tag to the user — this is one area where further work could be done to improve

the predictions of the system).

The del.icio.us crawler also has helper methods which can take a user name or a web URL and automatically generate and crawl the correct URL on the del.icio.us website. These methods are used by the PowerShell cmdlets and by a simple WPF user interface which we created in the initial stages of the project, and allow crawling a specific user or URL, as well as crawling URLs and/or users that are already in the database but haven't been crawled yet.

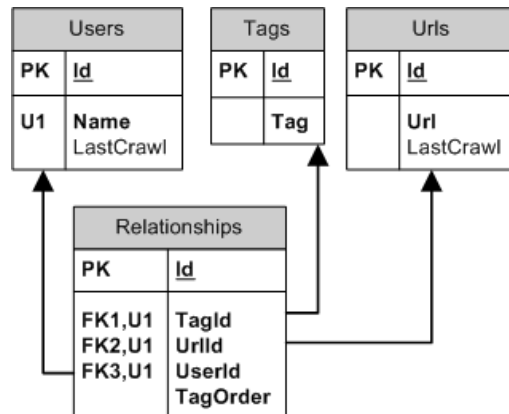


Figure 3.3: Database Design

3.2 SOM Networks

The GHSOM algorithm takes its input from the flat text files which are dumped by the `Get-TrainingData` cmdlet (all of the cmdlets are in the `DeliciousSnapin` assembly to make them easier to load), and it outputs to text files and XHTML, so the maps are serialized in an XML format which is easily readable by the recommender and by users.

Basically there are two maps created based on the data: users and tags. We actually wrote the code for generating a map of URL proximity based on users/tags, but it turned out to be confusing to visualize (if only because the "labels" are very long text URLs), and producing a score for new URLs against it using the clustering method we use with the other maps is conceptually troublesome, so we work exclusively with the other two.

The first map is the User map. In simple terms, it measures distance between users based on the number of URLs that users have in common in their bookmarks (with the number of tags used as a weight). We experimented with several attempts to adjust the weight based on the actual count of tags that the user has (so that users with few items tagged wouldn't be penalized), but that proved irrelevant when we made the decision to only use the most data-rich users.

The second map is the Tag map. It measures distance between tags based on the number of URLs the tags are both used on (with the number of users using it as the weight). Again, although we experimented with a few attempts to adjust the weights so that infrequently used tags that were used consistently could be more useful, in the end we removed most of those tags from consideration through our "dumper" filtering.

If the training of the SOM networks could be sped up significantly — or if it could be done “online” continuously as new data comes in — we would want to include more data in the training data, and it would therefore be useful to reexamine the question of weighting the training data in a way that made the weight be based on frequency rather than count.

3.3 Recommender

The third and final part of the SOMR system is the recommender. It steps through the items in the del.icio.us RSS feed and for each one, gets a collection of “hits” against the maps for users and tags. It then generates a rating for each of those maps based on the specific user, and generates a combined rating based on a weighted combination of the two maps. Finally, it sorts the items into order and outputs the best items. Because SOMR uses data which is stored on the del.icio.us web servers, this becomes a four part process: before recommendations can be made we must not only pull the new items from the web feed, but also download and parse the detail pages for each new item to gather the bookmarking and tagging information. Once that’s done, the actual recommender will run, but before it can process the items it must have the data about the user in question, and determine clusters for the user’s most common tags.

Of course, for the recommender to work in a timely manner, the items in the del.icio.us feed have to be pre-processed — their pages on del.icio.us have to be scraped so that they can be evaluated against the maps. The ideal way to do this would be to have it all running on the same server, of course. However, in our case it can be done in the background on the client.

The first step starts with the Windows feed store which continuously gathers new items from the del.icio.us feed. It is a Windows service provided as part of IE 7, and it continuously checks for updates (at a user-specified interval) and stores the new items until the recommender (or a user) retrieves them.

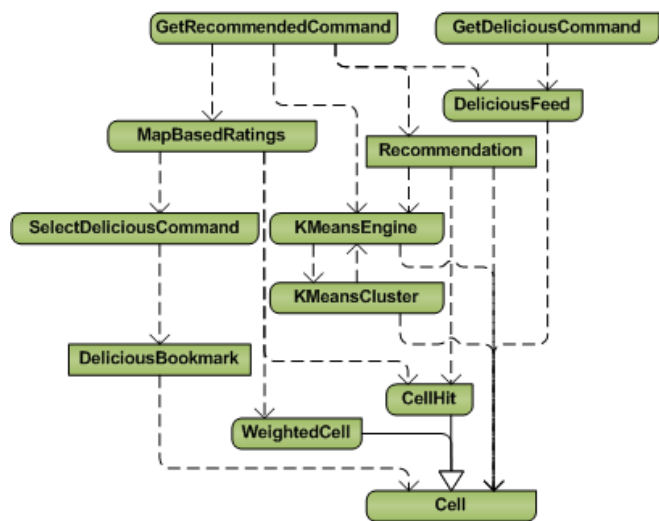


Figure 3.4: Snapin Dependency

The second step of getting a recommendation is the delicious feeder, as you can see in 2.5. Ideally this should run as a windows service or background process, pulling data from the feed store as it arrives, and processing each new URL. In our current implementation it is, instead, a PowerShell cmdlet which can be run by hand or scheduled using Windows scheduled tasks. This cmdlet invokes the `DeliciousScraper` to gather the data for each of the new items into the SOMR database, and then marks those items as "read" in the feed store.

When a user invokes the `Get-Recommended` cmdlet, the third and fourth steps are processed in order: the SOM maps are loaded, the user's points of interest are calculated, and then all of the "read" items in the feed store are processed, scored, and output.

3.3.1 Mapping the User's Interests

The users' point(s) of interest on the tag map are initially determined using a K-means clustering algorithm: the tags that they use the most are plotted on the map and clustered into groups. Our original plan was to present this mapping to them and allow them to visually inspect it and alter their interest points so they could, for instance, remove one of their observed areas of interest to avoid getting recommendations about them. However, as mentioned earlier, the extended visualization was removed, and the editing of interest loci went with it.

On the user map the situation is simpler: the users' point of interest is simply the location that the SOM algorithm maps them to. It would be interesting to see if the GHSOM algorithm could be tweaked to keep the user's node always at the center as it grows the map—thus producing a unique map for each user of how close people are to them—the current process sometimes results in the user (since they have bookmarked most of the URLs being used for reference) being placed somewhat randomly in a corner of the map, which (as we shall see later) seems to reduce the usefulness of that map.

The clustering and mapping can be re-calculated each time a request for recommendations is made (by running the `Get-Recommended` cmdlet), or can be calculated once and persisted to file for reuse. In our implementation there is a `Get-TagClusters` cmdlet which will return the clusters in a persistable form which can be passed to the `Get-Recommended` cmdlet. It also has an option to read the clusters from disk or re-calculate them and write them to disk. If `Get-TagClusters` is not called explicitly, and it's results passed to the call to `Get-Recommended`, then it is invoked implicitly, recalculating the clusters from the stored SOMR output maps.

3.3.2 Making Recommendations

The maps, the clusters, and the RSS items (with the scraped data) are then used in the `ProcessRecord` method to calculate scores for each RSS item.

The recommender system itself is actually a combination of two rating systems: one for tags, and one for users. It loops through all of the “read” items in the feed store (these are the ones which we’ve already fetched data for) and for each one, it finds the tags on the map and takes the `center` point of them and determines which of the user’s clusters is nearest, and assigns the “tag score” as the distance between them. The “user score” is calculated as the average of the distances from the user to the users who have already tagged the item.

The two ratings are then normalized to 100, so that they have similar absolute values, and they are combined using a *weight*, which generally favors the tag rating as being the more accurate of the two.

The output of the `Get-Recommended` cmdlet is a series of `Recommendation` objects which have all of this information embedded in them: the center point of the tags, the nearest tag cluster and it’s distance, the center point of the tagging users, and the user’s position, as well as the average distances for the users, and the three ratings. In addition, the recommender also includes on the output items a list of the top few tags that most closely matched the user’s interest, as well as a few of users who are closest to the user (which is frequently all the users, since we’re pulling the “new” items, they typically only have a few users who have bookmarked them).

The `DeliciousSnapin` includes a custom PowerShell format file which causes PowerShell to format the `Recommendations` in a table, and to display just the three ratings and the URL, and with a script which automatically sorts the output by the combination rating, and allows automatically opening items in a browser, etc., but all of the data is there in the objects for the expert end-user to play with (and for our user during testing).

4. Project Outcomes

The final recommender defaults to a weights which we judged to produce the most desirable results by putting the emphasis on the difference in score from the tags map over the score from the user map, primarily because the tag map scores on their own are clearly better predictors than the user map scores. We believe this difference could be improved by changing the algorithm for the users to assign weight to users logarithmically such that users who are “close” to the user carry more weight in the end score than users who are “far” from the user — if some of the people close to you have bookmarked it, we don’t want to throw it out just because people who are far from you have also bookmarked it.

The SOMR system does appear to improve on existing recommenders by taking two types of data into account, but it also performs surprisingly well considering that users are not required to explicitly “rate” items in order to get started. That is, although it will produce substantially better results with *tagged* bookmarks, the recommender can, in fact, be populated just with the user’s bookmark file, providing a workaround for the problems encountered by similar systems (although, as noted above, the recommendations based on *just* the user’s bookmark similarity are not as good). More importantly, we believe our results show that such a system could be implemented by any of the myriad of social tagging sites in existence today with good results, and accurate recommendations which would surpass the output of such popular recommender systems as “StumbleUpon”.

Simple experimentation shows that the SOMR system is substantially better with both tag and user data than with either one alone, and that it’s accuracy increases dramatically when there are more users who have bookmarked the page – in fact, in the final recommender a strict limit was set requiring at least 5 users to have bookmarked the page before it could be recommended.

4.1 Good Results

We tested the system by gathering data at two points in time, and running the recommender on the initial point and seeing how many of the recommended links have been bookmarked by the second run. Additionally once the system is in working order, we ran

Rating	Tag R#	User R#	Link
74.793	92.798	49.587	http://www.yammer.com/
66.250	86.335	38.131	http://www.os2world.com/
63.534	86.720	31.072	http://idlebackup.nl/
63.454	81.780	37.796	http://www.ovguide.com/
62.709	77.226	42.385	http://www.osdata.com/
62.246	86.720	27.981	http://umbrellatoday.com/
61.412	81.220	33.680	http://www.stormpulse.com/
61.174	83.896	29.363	http://savefile.com/
60.904	62.440	58.754	http://www.slate.com/id/2195892/
60.449	83.896	27.622	http://www.sendspace.com/
60.009	77.455	35.584	http://www.truecrypt.org/docs/
59.890	74.033	40.090	http://www.kidsastronomy.com/
59.582	81.220	29.289	http://www.americanrhetoric.com/top100speechesall.html
59.472	78.394	32.980	http://www.openwinforms.com/
59.245	86.720	20.779	http://www.sweetcron.com/
57.231	81.220	23.646	http://www.techbargains.com/
58.342	83.896	22.567	http://en.wikipedia.org/wiki/Antikythera_mechanism
57.597	83.896	20.779	http://www.rememberthemilk.com/
57.558	81.780	23.646	http://www.collectivex.com/
57.216	88.300	13.577	http://www.gyminee.com/

Table 4.1: Sample with “Top 20” Recommended

a live test and rated a series of recommendations (as was done in Mathes [2004]) to judge the precision of the recommender. Thus, the criteria for “good” recommendations in the first case is that the user had already bookmarked the page, and in the second, that they judged it an “interesting” page.

We did this test for two users, and the results were quite good, but because of the system resources and time necessary to scrape a user’s full bookmarks and create user-specific tag and user maps, we did not do any broader testing.

The accuracy of the recommendations and ratings is the primary concern in any information filter, and we found that the SOMR system was actually impressively accurate in our testing. In the typical test run shown in 4.1, out of the top 20 URLs the filter passed through, the user chose to tag (or had already tagged) 11 of the top 15 by rating, and was able to improve the accuracy of future recommendations by adding a requirement that the minimum rating be at or above 59%.

4.2 Shortcomings

There are some shortcomings in our proof-of-concept implementation which aren’t major problems, but are basically artifacts of the fact that we created the simplest implementation possible to just prove the system would work. The main problem of this sort is the fact that the SOM algorithm requires text files for input and output. If we were designing this as a “product,” we would create a custom SOM implementation which could read input directly from the database, and output a serialized multi-dimensional array along

with the XHTML. Of course we would also create the Feeder as a background service, as mentioned before.

The main downside of the SOMR system, however, is quite simply the performance of the GHSOM network. The GHSOM algorithm takes nearly 12 hours to process our limited data sets of the top 1000 URLs against 2000 tags and users on our dual-core development system. The only reason this is somewhat acceptable is because the maps don't have to be recalculated very frequently, but in order for this system to be deployable to end users it would have to be reduced astonishingly, or the system would need to be able to update the training as new URLs came in without recalculating from scratch the way we're doing it now.

4.3 Future Work

We have already mentioned several areas where the SOMR system could be improved, including the visualization system we had originally planned, and a few algorithm improvements such as using semantic tag clustering or a code-book model, and applying weighting to user distances. A visualization system that included automatically generated summary keywords for regions of the SOM maps would be a good improvement, and if it could be designed like a modern web map system, with different layers to represent the user's areas of interest, or to represent natural semantic clustering in the tags, etc., it would be a huge improvement over what's currently available. If it allowed the user to manipulate their areas of interest it would not only give them more control, but could result in allowing new users to be jump-started into the system without any bookmarks whatsoever by allowing them to chose a few areas of interest themselves.

The biggest improvements in the recommender system, however, would have to be speed-ups to the SOM training. The absolute biggest impact would be the integration of a SOM network that could maintain state and train incrementally. If such a system could be designed to train from a SQL query or stored procedure, skipping the extra step of generating text-files for input, that would be an added bonus, but in the end, any system which trained incrementally would be expected to virtually eliminate the problems we expressed related to the amount of time this training takes, and thus make this system viable for individual use, even without the explicit cooperation of a web-based tagging system.

Further improvements to the accuracy of the system would most likely involve improving the accuracy of the individual maps, or adding a third dimension (the missing

“url map” which would map new urls onto old ones based on *both* who had tagged them and what tags they used).

We believe that reducing the tag input space using semantic word clustering or a code-book model previously mentioned could also be useful. Since we are currently unable to process all the tags, any method which could map less frequent tags onto the frequent ones as synonyms would broaden the amount of tag data the system could process.

Finally, a weighting system for the user distance, might result in more varied and accurate scores for the user map as well, which would result in improving the final ratings. For instance, allowing users close to the target user to count for more than users further away, or calculating the user rating as a weighted distance to the closest users compared to the total count of users. We were unable to test any of the ideas we had for improving the calculation of the ratings from the distances on the user map, so there are undoubtedly other possibilities as well.

4.4 Deliverables

- The full project’s source code and Windows binaries, as well as a sample database (full of data) are included with the digital copy of this report that is available on the web at <http://JoelBennett.net/Recommender/>, and from the RIT CS department. In addition, we have included instructions and PowerShell scripts for compiling and running the software from source.
- A short user manual also accompanies the software, with instructions on how to use the various cmdlets to scrape del.icio.us and get recommendations, as well as how to tweak the weightings to experiment with the usefulness of the additional data.
- A bookmark file is also included with the source code which can be imported to del.icio.us so that you can use the same basic data that was used during this experiment: you can simply create a new account for yourself, and import the bookmarks provided.
- Code documentation can be produced at any time using SandCastle to generate class docs from the code comments, and a sample run has been included.

Annotated Bibliography

Arnt, A. and Zilberstein, S. (2003). Learning to perform moderation in online forums. In *Web Intelligence, 2003. WI 2003. Proceedings. IEEE/WIC International Conference on*, pages 637–641. Minimum description length (MDL) algorithm - probability of each word appearing in a document, given the documents length and the assumption that it belongs to a given category, trims non-discriminating words from the data. This paper discusses it's use in web forum moderation.

Balabanović, M. (1997). An adaptive web page recommendation service. In *AGENTS '97: Proceedings of the first international conference on Autonomous agents*, pages 378–385, New York, NY, USA. ACM Press. The introductory paper to the FAB hybrid system ... claims that pure collaborative recommendation solves all of the shortcomings of pure content-based systems, and content-based systems solve most of the problems inherent in collaborative systems. However, FAB doesn't fully implement a collaborative system, and doesn't overcome the content-based weaknesses.

Balabanović, M. and Shoham, Y. (1997). Fab: content-based, collaborative recommendation. *Communications of the ACM*, 40(3):66–72. The Stanford University digital library project has created "Fab," a partially hybrid recommendation system for web pages. It tracks the preferences of users (they're required to "rate" pages they are given to read) and based on *content* (word frequency) of the pages they rate, it builds profiles. Users are recommended pages which match their profiles, or which are rated highly by other users with similar profiles. [1.1](#), [1.1.1](#)

Bielenberg, K. and Zacher, M. (2005). Groups in social software: Utilizing tagging to integrate individual contexts for social navigation. Master of science in digital media, Universitt Bremen, Bremen. A description of Bayesian classifiers which represent users are decision trees which lead to recommendations, of clustering techniques for users to weight their influence on collaborative recomenders, and clustering items to use data from any item in the cluster to boost recommender quality.

- Breese, J. S., Heckerman, D., and Kadie, C. (1998). Empirical analysis of predictive algorithms for collaborative filtering. Technical Report MSR-TR-98-12, Microsoft, Redmond, WA 98052. Discusses the classification of recommenders as memory-based (weighted average of other users opinions) or model-based (correlation of items, like FAB, or vector similarity like NewsWeeder).
- Burke, R. (2002). Hybrid recommender systems: Survey and experiments. *User Modeling and User-Adapted Interaction*, 12(4):331–370. General overview survey of recommender systems and hybrid systems. [1.1](#)
- D. Keim, F. Mansmann, T. S. (2005). Mailsom - visual exploration of electronic mail archives using self-organizing maps. In *Second Conference on Email and Anti-Spam (CEAS 2005)*, Stanford University, Palo Alto, CA, USA, July 21-22, Palo Alto, CA, USA. Stanford University. Used a WebSOM-like algorithm to create SOMs, and then (using a spam filter in combination with manual classification) they rate each portion of the map based on the percentage of emails in that region that are considered "spam." Note that this is not a particularly good filter, since it necessarily leaves many emails on the fringes (there's no way to force a good separation), and doesn't grow. They have a couple other ideas for navigation and organization via the SOM.
- Delgado, J., Ishii, N., and Ura, T. (1998). Content-based collaborative information filtering: Actively learning to classify and recommend documents. In *CIA '98: Proceedings of the Second International Workshop on Cooperative Information Agents II, Learning, Mobility and Electronic Commerce for Information Discovery on the Internet*, pages 206–215, London, UK. Springer-Verlag. Content-based collaborative filtering. Tries to do collaborative filtering based on tf-idf: similarity of users is determined at the time of bookmarking based on the similarity of documents each user has bookmarked in the past (or rather, the keyword tfidf profile extracted from them), with greater weight applied to terms from documents in the class of the document being bookmarked...
- Douglas W. Oard, G. M. (1996). A conceptual framework for text filtering. Technical Report EE-TR-96-25 CAR-TR-830 CLIS-TR-96-02 CS-TR-3643, University of Maryland, College Park, MD 20742. Among other things, this paper contains a good description of H.P. Luhn's original manual content-based recommendation system from the 1950's. [1.1.1](#)
- Goldberg, D., Nichols, D., Oki, B. M., and Terry, D. (1992). Using collaborative filtering

to weave an information tapestry. *Commun. ACM*, 35(12):61–70. Tapestry. one of the first content filters which allowed a form of collaborative filtering. [1.1.2](#)

Good, N., Schafer, B. J., Konstan, J. A., Borchers, A., Sarwar, B., Herlocker, J., and Riedl, J. (1999). Combining collaborative filtering with personal agents for better recommendations. In *AAAI '99/IAAI '99: Proceedings of the sixteenth national conference on Artificial intelligence and the eleventh Innovative applications of artificial intelligence conference innovative applications of artificial intelligence*, pages 439–446, Menlo Park, CA, USA. American Association for Artificial Intelligence. Author's claim: The next generation of intelligent information systems will rely on cooperative agents to play the fundamental role of actively searching and finding relevant information on behalf of users ... They've created a system to support research by recommending bookmarks from one researcher to other researchers with similar research interests. It's a fairly simple system, but the paper is disproportionately interesting for it's assumptions and research background. [1.1](#)

Honkela, T., Kaski, S., Lagus, K., and Kohonen, T. (1998). Websom - self-organizing maps of document collections. In *Proceedings of WSOM'97, Workshop on Self-Organizing Maps, Espoo, Finland, June 4-6*, pages 310–315. Helsinki University of Technology, Neural Networks Research Centre, Espoo, Finland. A very slick visual representation of extremely large SOM maps for browsing and navigation. Multi-level zoom and links to actual documents. NOT an application of SOM to the web, but rather of the Web as a UI for SOMs. Of course, SOMs using full-text have already been applied to pretty much all types of documents you can imagine. [1.3.1](#)

Kohonen, T., Hynninen, J., Kangas, J., and Laaksonen, J. (1996). Som pak: The self-organizing map program package. Computer Science Report A31, Helsinki University of Technology, Rakentajanaukio 2 C, SF-02150 Espoo, FINLAND. The original SOM package from Kohonen available in source and binary format, with documentation. [1.3](#)

Kohonen, T., Kaski, S., Lagus, K., Salojarvi, J., Honkela, J., Paatero, V., and Saarela, A. (2000). Self organization of a massive document collection. *Neural Networks, IEEE Transactions on*, 11(3):574–585. A discussion of the special considerations needed for scaling SOM networks to map massive document collections. Notes semantic word clustering doesn't scale.

Konstan, J. A., Miller, B. N., Maltz, D., Herlocker, J. L., Gordon, L. R., and Riedl, J. (1997). GroupLens: Applying collaborative filtering to usenet news. *Communications*

- of the ACM*, 40(3):77–87. GroupLens uses only collaborative filters to filter USENET. It's partially a reaction to Tapestry being "closed." Identified the cold-start problem.
- Konstan, J. B. S. J. and Riedl, J. (1999). Recommender systems in e-commerce. In *1st ACM. Conf. on Electronic Commerce (EC99)*. A good sources for who's using collaborative recommenders in e-commerce.
- Lagus, K., Kaski, S., and Kohonen, T. (2004). Mining massive document collections by the websom method. *Inf. Sci.*, 163(1-3):135–156. Special methods for using SOM on massive document collections. Semantic Word Clustering, Singular value decomposition (SVD), etc. [1.3.1](#)
- Lang, K. (1995). Newsweeder: Learning to filter netnews. In *Proceedings of the 12th International Conference on Machine Learning*, pages 331–339, San Mateo, CA, USA. Morgan Kaufmann publishers Inc. Vector-based similarity using cosine of term frequency vectors.
- Lemire, D. and Maclachlan, A. (2005). Slope one predictors for online rating-based collaborative filtering. In *Proceedings of SIAM Data Mining (SDM'05)*. A discussion of correlation algorithms (a form of model-based algorithms) which derive models of the decision processes based on the distance between items, such as those used by Fab and RAAP.
- Li, Q. and Kim, B. M. (2003). Clustering approach for hybrid recommender system. In *Proceedings of the IEEE/WIC International Conference on Web Intelligence (WI03)*, pages 33–38. IEEE. This paper has particularly good background coverage of recommenders, from query-based systems through hybrid systems. The point of the paper is to introduce an enhanced collaborative recommender system using clustering techniques to bootstrap new items. This introduces a possibility to use content-analysis as a way of generating keyword tags for pages which haven't been bookmarked or tagged by users. This has an interesting application in allowing the introduction of new pages (such as advertiser's web pages) based on automated keyword assignment.
- Lovett, C. (2008). Sgmlreader. Online; accessed 03-November-2008. [3.1](#)
- Luo, X. Zincir-Heywood, A. (2003). A comparison of som based document categorization systems. In *Proceedings of the International Joint Conference on Neural Networks, 2003*, volume 3, 6050 University Avenue, Halifax, Nova Scotia. B3H 1W5. Dalhousie University. In information retrieval, a typical data representation phase uses the Vector

- Space Model (VSM), where the frequency of occurrence of each word in each document is recorded... this approach considers only term co-occurrences in documents and [ignores order and context]. The authors propose an alternative which they claim results in 90 vs 56 quality. I'm interested in this alternative, but I'm also very interested in how they determined the "quality." [1.3.2](#)
- Malone, T. W., Grant, K. R., Turbak, F. A., Brobst, S. A., and Cohen, M. D. (1987). Intelligent information-sharing systems. *Commun. ACM*, 30(5):390–402. Information Lens suggested the need for social factors to be taken into account, specifically, that social filtering rely “not just on the characteristics of the author, but also on the references and recommendations of other people.” [1.1.2](#)
- Mathes, A. (2004). Folksonomies - cooperative classification and communication through shared metadata. A graduate research paper. [2.1](#), [4.1](#)
- Millen, D., Feinberg, J., and Kerr, B. (2005). Social bookmarking in the enterprise. *Queue*, 3(9):28–35. Corporate use of Delicious etc. Why you should... Also citeulike-article-id = 703553. [1.2](#)
- Oard, D. W. (1997). The state of the art in text filtering. *User Modeling and User-Adapted Interaction*, 7(3):141–178. Statistics and information on the progress of the state of the art in text filtering or recommenders.
- Oja, M., Kaski, S., and Kohonen, T. (2003). Bibliography of self-organizing map (som) papers: 1998-2001 addendum. *Neural Computing Surveys*, 3:1–156. A bibliography of all research papers which have used SOM networks. [1.3](#)
- Ono, C., Motomura, Y., and Asoh, H. (2005). A study of probabilistic models for integrating collaborative and content-based recommendation. In *Nineteenth International Joint Conference on Artificial Intelligence (IJCAI05)*. Probabilistic Models for Hybrid Recommenders ... includes some references to specific implementations.
- Rauber, A., Merkl, D., and Dittenbach, M. (2002). The growing hierarchical self-organizing maps: exploratory analysis of high-dimensional data. [1.3](#), [2.3](#)
- Resnick, P., Iacovou, N., Suchak, M., Bergstorm, P., and Riedl, J. (1994). GroupLens: An open architecture for collaborative filtering of netnews. In *Proceedings of ACM 1994 Conference on Computer Supported Cooperative Work*, pages 175–186, Chapel Hill, North Carolina. ACM. Another paper on GroupLens, with specifics on the architecture and how it compares to other systems at the time.

Roh, T. H., Oh, K. J., and Han, I. (2003). The collaborative filtering recommendation based on som cluster-indexing cbr. *Expert Systems with Applications*, 25(3):413–423. This is probably as close to my actual project as anyone has ever written ... It uses "ratings" to map to other users "like me" but doesn't use keywords or contents to map to articles "like this one". That is, they worked exclusively with individual's profiles: the SOM (Self-Organizing Maps) creates clusters of like-minded users based on their ratings for each article. They worked from the GroupLens data which has users' ratings of items, whereas I'll be working from bookmarking data, so I have keywords instead of ratings. Given a specific user, they only recommend articles which the user-cluster rated highly but which the user hasn't rated at all. I'll be able to recommend based on similar users and similar keyword documents. They use CBR (Case Based Reasoning), which I'm not very familiar with, to select areas from the SOM. I'm thinking about trying that, to follow their example, and keep the differences of my research as few as possible, however, I *had* been thinking about using simple probability based on proximity to the clusters, or possibly extending it to some sort of bayesian network or use dempster-schafer theory. [2.1](#)

Shardanand, U. and Maes, P. (1995). Social information filtering: algorithms for automating "word of mouth". In *CHI '95: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 210–217, New York, NY, USA. ACM Press/Addison-Wesley Publishing Co. Ringo, another system like GroupLens which was one of the first to send predictions without requiring users to create queries.

Vucetic, S. and Obradovic, Z. (2005). Collaborative filtering using a regression-based approach. *Knowl. Inf. Syst.*, 7(1):1–22. Describes a rating system where the difference between a user's ratings for some items and several expert's ratings for those same items are quantitatively measured and weighted to predict how the the user would rate other items that the experts have already rated. Really good article, thorough and comprehensive coverage of both recommenders in general, as well as their specific approach.

Wikipedia (2006). Principal components analysis — wikipedia, the free encyclopedia. Online; accessed 22-August-2006. The wikipedia has a good article on PCA (principal components analysis): a statistical method for reducing dimensions or extracting features.