Data-Driven Ontologies for Recommender Engines in Social Networks

Ingo Bax and János Moldvay

XING AG
Gänsemarkt 43
20354 Hamburg
Germany

Abstract. A crucial factor for success in today's web business is the ability to offer relevant, personalized content to users. To achieve this, most modern websites – especially e-commerce and social networking platforms – rely on data-driven recommender engines. In this contribution we present a general recommender scheme which uses an automatically generated ontology. The approach is based on a multistage procedure that involves the extraction of a large number of tags from user profiles of a popular social networking platform. From these tags an inverted index is computed which is often referred to as a "folksonomy". From this index a tag graph can be created which then allows to group together semantically related tags using a clustering method. We show how the resulting tag clusters can be linked to virtually any data entity on the web site in order to obtain a recommender engine for that entity. We provide two example applications for this: "user-to-user" recommendations and "user-to-group" recommendations. Approaches for evaluating the quality of the recommendations are discussed.

1 Introduction

One reason for the high popularity and the outstanding success of big internet players like Google, Amazon or Facebook is the way these websites present certain information to their users. These systems prove to be very efficient in mining their own terabytes of data in order to find the content that is most likely to be relevant to the user.

The enormous amount of content on these sites obeys the classical power law distribution, i.e. it adheres the well known long tail paradigm [2]. Most of the content on these platforms is relevant to only a very small number of users.

Therefore it is more efficient to *push* this content to users as recommendations rather than let the user *pull* it. Prominent examples for this are the "Customers Who Bought This Item Also Bought"-feature on Amazon or the friend suggestion mechanism on Facebook. Features like these make a website highly efficient and convenient for the users, but also play a critical role in many business models, especially in e-commerce applications.

A prerequisite for building such recommender engines is the availability of the right ontologies that allow us to relate different objects to one another, or

even to relate objects of different types to each other. Because of the massive amounts of data, manual procedures for building such ontologies are infeasible. Instead, automatic processes are needed for creating such ontologies. In this contribution we show for a popular social networking platform, that tag data provides an excellent data source which can be structured in multiple steps via automatic processes such that a general semantic recommender scheme can be obtained.

The organization of this manuscript follows the sketch of the system architecture as depicted in Fig. 1. In a first step, tags are extracted from a large database of user profiles and an inverted index – called "Tag Index" below – is created from the data (see Sect. 2). This tag index can be viewed as a special type of "folksonomy" [12]. From this index, in a second step, a "Tag Graph" is computed which links tags according to their co-occurrence in user profiles (see Sect. 3). Based on this graph, a clustering method is applied for grouping related tags (Sect. 4). Finally, these "Tag Clusters" are used to to build different recommender engines that are used by the web application to push personalized and relevant content to users.

We present two application examples of the proposed recommender scheme: "user-to-user" and "user-to-group" recommendations (see Sect. 5). In Sect. 6 we describe the experimental evaluation of the effectiveness of the recommender engines and we provide a summary and a conclusion in Sect. 7.

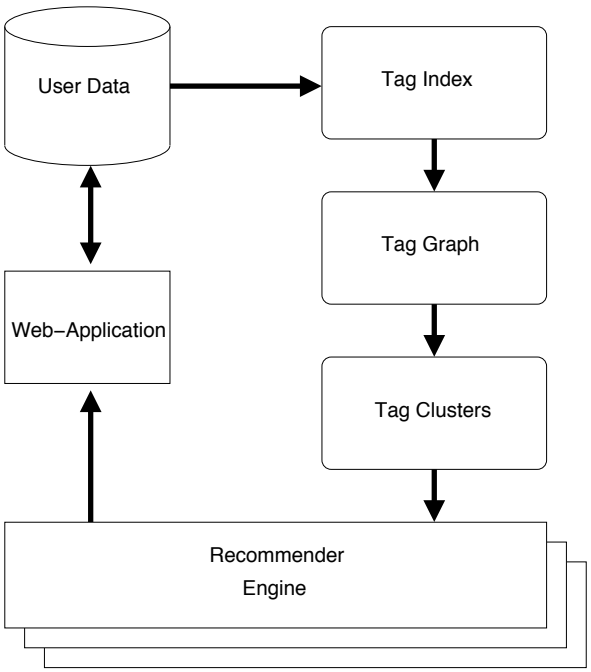


Fig. 1. System overview: Recommender engines for a web application are created by extracting a large number of tags from user profiles of a popular social networking website and structuring this data in multiple processing stages. These involve building a "Tag Index", a "Tag Graph" and finally, "Tag Clusters".

1.1 Related Work

Previous work regarding the creation of folksonomies from user generated meta-data on web sites was presented in [12]. The authors discuss the advantage of the approach over meta-data being created by dedicated experts or by the respective owners of the resources themselves, but possibilities for bringing structure to the folksonomie by further processing are not investigated.

One of the first works which used a clustering approach for analyzing the structure of a folksonomy was presented in [3], where the authors utilized the unveiled structures for improving search and exploration. The idea of creating personalized content by clustering a folksonomy was also brought up by [9], who also used the approach for improving search rather than creating recommendations.

Another approach for applying clustering techniques to uncover structures in folksonomies can be found in [14], where the author utilized the algorithm of Clauset, Newman and Moore [5] to find groups of tags in a tag graph. A modified version of this approach is also adopted for our method (see Sect. 4).

2 Tag Index: A Folksonomy of User Profile Data

At present, the XING platform holds more than eight million profiles of professionals from all over the world, covering all kinds of organization sizes, career levels and industries. On these profiles users provide rather structured information about their educational background and their employment history, but also unstructured data in a tag-like format about their personal *interests*, their professional skills (called *haves*), what they are seeking (called *wants*) and what *organizations* they belong to. This provides a rich data source for extracting a folksonomy [12] by processing and aggregating tags from these profiles and building an inverted index from this data.

In contrast to other popular web sites like flickr[1] or del.icio.us[2], where folksonomies arise from users collaboratively tagging a set of resources [9], we have a slightly different scenario here where there are no resources as such, but instead there are users who enter tags about themselves. Nevertheless – and as will be demonstrated below – this is completely sufficient for creating a useful special type of folksonomy $D = (U, T, A)$ which consists of a set of users U, a set of tags T and a set of annotations A of the form

$$A \subseteq \{\ (u, t) \mid u \in U, t \in T\ \}. \tag{1}$$

Besides the folksonomy D, we are also interested in two more things: a function

$$freq(t) = \|\{\ (u, t') \in A,\ t' = t\ \}\|, \tag{2}$$

which denotes the total number of occurrences of a tag t in A and a function

[1] http://www.flickr.com
[2] http://delicious.com

$$U(t) = \{\ u \mid (u, t') \in A,\ t' = t\ \}, \tag{3}$$

which denotes the set of users, that have the tag t in their profile.

In the following we describe from a technical point of view how tag data is extracted from user profiles and how an inverted index can be computed using the MapReduce paradigm [7]. This inverted index is a "materialization" of the set of tags T together with the above functions $freq(t)$ and $U(t)$.

2.1 Data Extraction

In order to extract tag data from user profiles a simple tokenizer is used which "visits" each profile and collects tags by converting the structured data into a flat list of terms. These terms are then passed through a normalization function which removes all whitespaces and special characters and capitalizes each character. Formally, we define the tokenizer as a function $T(u)$ which returns the set of all tags a user u has in their profile. This function will be used in the process described below.

2.2 MapReduce

While it is trivial to compute an inverted index given a small number of annotations (u, t), it turns out to be not as easy for a very large amount of input data like in the current application scenario: the total number of tags to be indexed is currently about 67 million. Also, the number of tags grows from day to day and the source data changes frequently, such that the index does not have to be build only once, but it must be automatically rebuilt on a regular basis. In order to achieve this, the computation has to be parallelized due to time and memory constraints. In fact, this is one of the standard application examples of the MapReduce paradigm as introduced in [7]. Parallelization is achieved by carrying out the computation in two phases, a "map-phase" and a "reduce-phase". In the "map-phase" the input data is split up into multiple blocks – each of which can be processed in parallel – and transformed into an intermediate representation. The intermediate representation has again the property that it can be split into blocks allowing for parallel processing in order to obtain the final representation.

Figure 2 shows the implementation of the MapReduce process for our task. In a first step, the user data is split into many blocks which are processed in parallel by a worker farm. When processing one block, a worker reads the profile data from the user database, applies the above described tokenizer to obtain a set of annotations (u, t) and writes them into an intermediate representation consisting of the two tables `tag`* and `user_tag`*. After the "map-phase" is finished, the "reduce-phase" starts and the table `tag`* is split into blocks which are again processed in parallel by another worker farm. Each worker processes blocks of tags by reading all corresponding annotations from the table `user_tag`*, determining the `count`, converting the string-typed tags to numerical ids for efficiency and writing the result into the two tables `tag` and `user_tag`.

This final representation serves as an efficient materialization of the set of tags T (columns `id` and `name` in table `tag`), the function $freq(t)$ (column `count` in table `tag`) and the function $U(t)$ (table `user_tag`). This will be the basis for the creation of a "Tag Graph" as described in the following Section.

3 Tag Graph: Deriving Semantic Relations

A tag graph $G = (T, E)$ is an undirected, weighted graph and consists of a set of tags T – the vertices of the graph – and a set of edges E given by

$$E = \{\ (t_i, t_j, w) \mid t_i, t_j \in T,\ t_i \neq t_j,\ w = lift(t_i, t_j),\ lift(t_i, t_j) > \theta\ \}, \tag{4}$$

where θ is a threshold parameter defining a minimum value for edge weights and $lift$ is a "relatedness" measure between the two tags t_i and t_j, defined as follows:

$$lift(t_i, t_j) = \frac{cooc(t_i, t_j) * \|U\|}{freq(t_i) * freq(t_j)}, \tag{5}$$

where $cooc(t_i, t_j)$ is the number of co-occurrences of the two tags t_i and t_j in one and the same user profile. The lift function as a measure of similarity between two items was introduced in [4], where it was originally called "interest". The function measures how many times more often the items t_i and t_j co-occur than expected if the two events where statistically independent.

This definition is a formalization of the idea that we can assume a certain semantic relationship between two tags if they often occur together in different users' profiles.

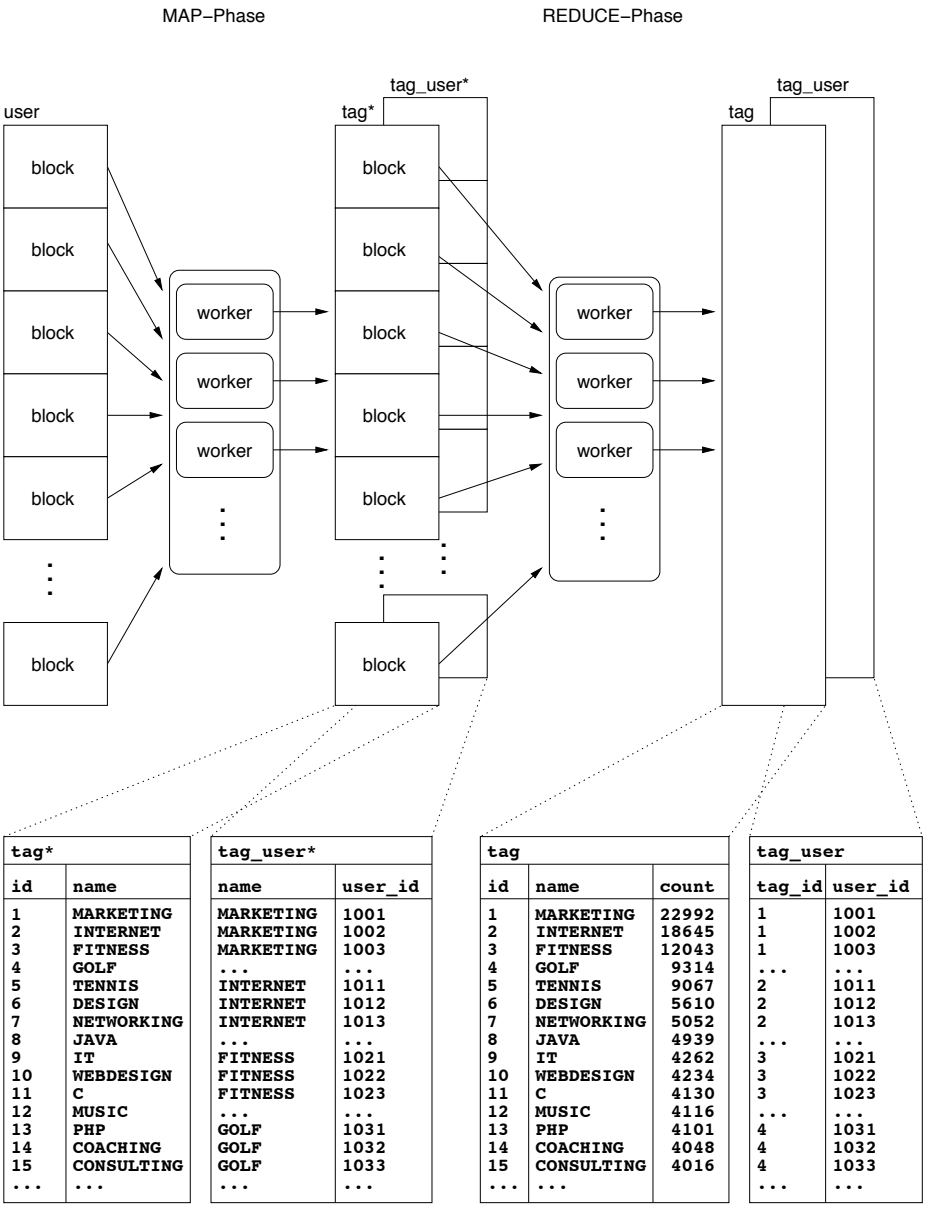
As also stated by other authors [3, 10, 6], a disadvantage of folksonomies is that they lack a kind of efficient organizational structure which is needed for many applications. The tag graph as introduced above can be seen as a first layer of such an organizational structure which already proves useful, e.g. for applications that recommend related tags [15, 11]. In the following, we use the tag graph as a basis for introducing even more structure by discovering groups of tags using a clustering technique.

4 Tag Clusters: Grouping Related Tags

The goal of creating tag clusters is to obtain an explicit organizational structure by finding groups of tags that are semantically related. The result is an ontology that can be used for building recommender engines as described in Sect. 5.

The input to the clustering method is the tag graph G as defined in Sect. 3 and as a result, we obtain a set C of clusters as well as a weight function

$$w_{\texttt{tag}} :\ T \times C \to \mathbb{R} \tag{6}$$



tag*

id	name
1	MARKETING
2	INTERNET
3	FITNESS
4	GOLF
5	TENNIS
6	DESIGN
7	NETWORKING
8	JAVA
9	IT
10	WEBDESIGN
11	C
12	MUSIC
13	PHP
14	COACHING
15	CONSULTING
...	...

tag_user*

name	user_id
MARKETING	1001
MARKETING	1002
MARKETING	1003
...	...
INTERNET	1011
INTERNET	1012
INTERNET	1013
...	...
FITNESS	1021
FITNESS	1022
FITNESS	1023
...	...
GOLF	1031
GOLF	1032
GOLF	1033
...	...

tag

id	name	count
1	MARKETING	22992
2	INTERNET	18645
3	FITNESS	12043
4	GOLF	9314
5	TENNIS	9067
6	DESIGN	5610
7	NETWORKING	5052
8	JAVA	4939
9	IT	4262
10	WEBDESIGN	4234
11	C	4130
12	MUSIC	4116
13	PHP	4101
14	COACHING	4048
15	CONSULTING	4016
...	...	

tag_user

tag_id	user_id
1	1001
1	1002
1	1003
...	...
2	1011
2	1012
2	1013
...	...
3	1021
3	1022
3	1023
...	...
4	1031
4	1032
4	1033
...	...

Fig. 2. Computing an inverted index of tags via MapReduce [7]: In a "map-phase" the user profile data is split up into multiple blocks which are processed in parallel by a worker farm and transformed into an intermediate representation given by the two tables `tag`* and `user_tag`*. In the "reduce-phase" the intermediate representation is again processed in blocks in order to obtain the final representation consisting of the two relations `tag` and `user_tag`. See text for further detail.

which returns – given a tag $t \in T$ and a cluster $c \in C$ – an "association strength" of the tag for this cluster.

For computing the cluster solution, we used a modified version of a graph based clustering method for community detection as introduced by Clauset, Newman and Moore [5]. Their algorithm optimizes a measure called "modularity" as defined by Newman and Girvan in [13]. The measure evaluates the quality of graph cluster solutions with respect to criterions originating from the discipline of network analysis. Based on a publicly available implementation in the igraph-library[3], we modified the method by adding constraints on the minimum and maximum number of tags per cluster. In our experiments, choosing 2 as a minimum and 30 as a maximum value for the number of tags per cluster yielded the best overall quality of the cluster solution, as determined by carrying out a series of experiments with different boundaries.

Another crucial factor for the quality of the cluster solution is the threshold θ from Eq. 4 which controls the number of edges in the input graph. It turned out, that the best results can be obtained by choosing rather high values for θ like 50 which effectively selects a subgraph only containing highly relevant connections.

Finally, for computing the value of the weight function, given a cluster and a tag, we utilize the "betweenness"-measure from [8] which describes how strong a tag's relation to a cluster is.

Note: The clustering method mentioned here produces solutions, where every tag belongs to exactly one cluster. This means, that for a given tag t the function $w_{\texttt{tag}}(t, c)$ has a positive value for only one cluster c^* and is 0 for all other clusters. However, our formalism would allow to use a different clustering algorithm which produces a fuzzy result in the sense, that each tag belongs to a cluster with a certain probability. This is useful for tags with ambiguous meanings [9].

5 Recommender Engines

In this section we show how the tag clusters as defined in the preceding section can be utilized in a general way for building recommender engines. The tag clusters can be seen as a kind of linkage between any two types of objects. Two examples that will be presented below are "user-to-user" and "user-to-group" recommendations. Other combinations like "user-to-jobposting", "user-to-event", or even "event-to-group" are thinkable as well.

Whichever case we choose, the key ingredient for creating a recommender engine between the two types of objects is to define weight functions which compute "projections" of the given objects to the cluster solution. E.g., for objects of type "user" a simple weight function can be defined as follows:

$$w_{\texttt{user}}(c, u) = \sum_{t \in T(u)} w_{\texttt{tag}}(t, c) \; \forall c \in C. \tag{7}$$

Recall, that $T(u)$ is the set of all tags that a user has in their profile (Sect. 2.1) and $w_{\texttt{tag}}(t, c)$ is the weight that tag t has in cluster c (Eq. 6). The function

[3] http://igraph.sourceforge.net/

simply takes all tags a user has in their profile and cumulates the weights of the tags cluster-wise.

Having available weight functions for both object types and given one object of the first type and a set of objects of the second type, we can generate a ranked list of recommendations by combining the outputs of the weight functions. Examples of how to do this will be given in the following two sections.

5.1 Example 1: User-to-user Recommendations

Using the weight function for users as already defined in the preceding section now "user-to-user" recommendations for a user u, given a set of other users U^*, can be computed by ranking each user $u^* \in U^*$ according to the following scoring function which cumulates the "projection"-weights for all clusters:

$$s_{\texttt{user}}(u, u^*) = \sum_{c \in C} w_{\texttt{user}}(u, c) * w_{\texttt{user}}(u^*, c). \tag{8}$$

Sorting the users in U^* according to the scoring function gives us a powerful recommender engine which can for instance be used to introduce users to others on the basis of semantic relations between the users' profiles, because this implies a high likelihood that the two persons would be interested in getting to know each other. This leverages the power of the social graph on the platform by increasing its density and generating new contacts between users.

A great advantage of the approach over search-based recommender engines is that by using the tag clusters, implicitly a "fuzzy" matching is done. For example, two users could get recommended to each other without having even one tag in common, but having tags in the same area of interest. For example, two users, one having the tags `DATAMINING` and `SAS` and the other the tags `MACHINELEARNING` and `R` can get recommended to each other at a high score, because all four tags have a high weight in the same cluster.

5.2 Example 2: Group Recommendations

Currently there are about 30.000 different groups on the platform, covering various topics ranging from "Amature Basketball Referees" to "SAS Users Germany". Accessability to these groups without appropriate recommender engines is suboptimal, a regular search engine approach is too inefficient. To develop a group recommender engine, i.e. a model predicting a user's membership in a specific group, we analyze the relationship between tag clusters and groups through association analysis [1].

Based on the association analysis we can construct a special weight function for groups which uses the co-occurrence of users in a given group and a given cluster as follows:

$$w_{\texttt{group}}(g, c) = \frac{cooc(g, c) * \|U\|}{freq(g) * freq(c)}, \tag{9}$$

where $cooc(g, c)$ denotes the number of users that are in group g and also have a non-zero weight $w_{\texttt{user}}(g, c)$ for cluster c. $freq(g)$ is the total number of users who are member of group g and $freq(c)$ is the total number of users for whom $w_{\texttt{user}}(g, c)$ is non-zero.

Finally, we can construct the recommender engine based on combining the user weight function (Eq. 7) and the group weight function to rank a set of groups G^* by computing a score for each group $g^* \in G^*$:

$$s_{\texttt{group}}(u, g^*) = \sum_{c \in C} w_{\texttt{user}}(u, c) * w_{\texttt{group}}(g^*, c). \tag{10}$$

Note: It would actually be more precise to conduct a "multiple item association analysis" [1] for every possible subset of clusters to every group. However, this is infeasible from a computational point of view and our experiments have shown that a simple cumulation of the weight values serves as a good approximation.

6 Evaluation

The method that we used for evaluating the quality of the recommendations produced by the approach proposed in this contribution was to send emails to a systematically selected test group of real users which contained some of the produced recommendations. Also, another group of users – a systematically selected control group – received emails with control content (see below). We then observed the behavior of the users in both groups on the platform with respect to a set of control variables. This allowed us to compare the two groups and thereby measure the success of the recommendations.

For evaluating the "user-to-user" recommender engine we sent emails containing the top four recommendations for each user to every one of 65.000 users in the test group. The users of the control group received an email with generic placeholder content. We then observed the users for one week and measured a set of test variables that were analyzed via hypothesis testing. Tab. 1 shows the resulting p-values. It can be seen that the variable "number of contacts" yields a significant improvement (assuming a significance level of 0.05) between the test and the control group. We value this as a proof for the usefulness of the "user-to-user" recommendations.

Regarding "user-to-group" recommendations, the results are also shown in Tab. 1. The test group also had a size of 65.000 users and each of them received the top four group recommendations. The test group received emails with control content. For this we chose to recommend a static list of ten very popular groups. From the results it can be seen that we obtained a highly significant improvement in the variable "number of groups joined" which proves the relevance of the group recommendations produced by our approach.

variable	user-to-user	user-to-group
number of profile visits	0,2788	N/A
number of contacts	**0,0222**	N/A
number of groups joined	0,5423	$<$ **0.0001**
number of invitations sent	0,0823	N/A
number of messages sent	0,3932	N/A
number of successful invitations	0,8624	N/A
number of logins	0,0917	N/A

Table 1. Results of the evaluation test with emails. Emails with recommendations were sent to real users of a test group. A set of test variables was observed and a hypothesis test was conducted against a control group. We obtained significant improvements (p-value below 0.05) in the variable "number of contacts" for "user-to-user" recommendations and in the variable "number of groups joined" for the "user-to-group" recommendations. (For the "user-to-group" recommendations, there are unfortunately no results available for the other variables.)

7 Summary and Conclusion

In this contribution we have presented an automatic multi-step procedure for creating a data-driven ontology for a popular business networking platform called XING. The ontology can be utilized as a general purpose recommender scheme.

For this we first created an inverted index of tags using the MapReduce paradigm and by extracting tag data from a large number of user profiles. The result of this is a special type of folksonomy. We then used association measures to derive a tag graph by discovering semantic relations between tags. This can be interpreted as bringing a first layer of organizational structure to the folksonomy.

Subsequently, we applied a graph clustering algorithm on the tag graph resulting in groups of tags which are semantically related to each other. The result of this was a second layer of organizational structure which can be seen as an ontology on top of the folksonomy. Based on tag clusters we showed how to build recommender engines by "projecting" different types of objects onto the tag clusters.

Finally, we presented evaluation results for two application scenarios for "user-to-user" recommendations and "user-to-group" recommendation, where we were able to prove the effectiveness of the proposed scheme. We sent emails to a test group of real users and obtained significant improvements in certain control variables.

Our future work will focus on utilizing the proposed scheme for building more recommender engines for other objects on the website in order to offer more relevant content to users. Promising candidates are for example "user-to-jobposting" and "user-to-events" recommender engines.

References

1. R. Agrawal, T. Imielinski, and A. Swami. Mining Association Rules between Sets of Items in Large Databases. In *Proc. SIGMOD Conference*, pages 207–216, 1993.
2. C. Anderson. *The Long Tail: Why the Future of Business Is Selling Less of More.* Hyperion, New York, 2006.
3. G. Begelman, P. Keller, and F. Smadja. Automated Tag Clustering: Improving search and exploration in the tag space. In *Proc. Collaborative Web Tagging Workshop at WWW 2006*, Edinburgh, Scotland, 2006.
4. S. Brin, R. Motwani, J. D. Ullman, and S. Tsur. Dynamic itemset counting and implication rules for market basket data. In *Proc. SIGMOD Conference*, pages 265–276, 1997.
5. A. Clauset, M. E. J. Newman, and C. Moore. Finding community structure in very large networks. *Phys. Rev. E*, 70(6):066111, Dec 2004.
6. C. Van Damme, M. Hepp, and K. Siorpaes. FolksOntology: An Integrated Approach for Turning Folksonomies into Ontologies. In *Proc. Workshop Bridging the Gap between Semantic Web and Web 2.0, ESWC 2007*, Innsbruck, Austria, 2007. LNCS Springer.
7. J. Dean and S. Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. In *Proc. 6th Symposium on Operating System Design and Implementation*, San Francisco, CA, 2004.
8. L. C. Freeman. A set of measures of centrality based on betweenness. *Sociometry*, 40(1):35–41, 1977.
9. J. Gemmell, A. Shepitsen, M. Mobasher, and R. Burke. Personalization in Folksonomies Based on Tag Clustering. In *Proc. 6th Workshop on Intelligent Techniques for Web Personalization and Recommender Systems*, pages 37–48, 2008.
10. P. Heymann and H. Garcia-Molina. Collaborative Creation of Communal Hierarchical Taxonomies in Social Tagging Systems. Technical Report 2006-10, Stanford InfoLab, April 2006.
11. M. Lipczak. Tag Recommendation for Folksonomies Oriented towards Individual Users. In *Proc. of the ECML/PKDD 2008 Discovery Challenge Workshop, part of the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases*, Antwerp, Belgium, 2008.
12. A. Mathes. Folksonomies - Cooperative Classification and Communication Through Shared Metadata. *Computer Mediated Communication - LIS590CMC*, 2004.
13. M. E. J. Newman and M. Girvan. Finding and evaluating community structure in networks. *Phys. Rev. E*, 69(2):026113, Feb 2004.
14. E. Simpson. Clustering Tags in Enterprise and Web Folksonomies. Technical report, HP Labs, 2008.
15. S. C. Sood, K. J. Hammond, S. H. Owsley, and L. Birnbaum. TagAssist: Automatic tag suggestion for blog posts. In *Proc. Int'l. Conf. Weblogs and Social Media (ICWSM 2007)*, 2007.