Tag Clustering with Self Organizing Maps

Marco Luca Sbodio, Edwin Simpson

HP Laboratories
HPL-2009-338

Keyword(s):
SOM, clustering, machine learning, folksonomy, tagging, web 2.0

Abstract:

Today, user-generated tags are a common way of navigating and organizing collections of resources. However, their value is limited by a lack of explicit semantics and differing use of tags between users. Clustering techniques that find groups of related tags could help to address these problems. In this paper, we show that a Self-Organizing Map (SOM) can be used to cluster tagged bookmarks. We present and test an iterative method for determining the optimal number of clusters. Finally, we show how the SOM can be used to intuitively classify new bookmarks into a set of clusters.

External Posting Date: October 6, 2009 [Fulltext] Approved for External Publication
Internal Posting Date: October 6, 2009 [Fulltext]

Tag Clustering with Self Organizing Maps

Marco Luca Sbodio
HP Italy Innovation Centre
Marco.Sbodio@hp.com

Edwin Simpson
Web Services and Systems Lab, HP Labs
edwin.simpson@gmail.com

October 5, 2009

Abstract

Today, user-generated tags are a common way of navigating and organizing collections of resources. However, their value is limited by a lack of explicit semantics and differing use of tags between users. Clustering techniques that find groups of related tags could help to address these problems. In this paper, we show that a Self-Organizing Map (SOM) can be used to cluster tagged bookmarks. We present and test an iterative method for determining the optimal number of clusters. Finally, we show how the SOM can be used to intuitively classify new bookmarks into a set of clusters.

1 Introduction

Recently, websites have introduced a range of innovative techniques known as Web 2.0 [7] [21]. These techniques have changed the way information is created, shared and organized, by encouraging the active involvement of end users in information production. One of these techniques, *tagging*, is used across many well-known websites, such as Youtube [9], Flickr [3] and Delicious [2]. Web sites supporting tagging allow users to associate one or more free-text labels with a content item, such as a video, image or web page. Tags are keywords used to describe an item in a way that is meaningful to the user who creates them, making them valuable for information retrieval. A set of tags is commonly referred to as *folksonomy*, a neologism coming from the fusion of the two words *folk* (people) and *taxonomy* [18] [26].

From a knowledge representation perspective, folksonomies lack formal consistency, and are not very precise: they do not constitute a shared vocabulary, and they do not have explicit semantics. For example, different users may use synonyms or generalizations of the same concept, and the folksonomy does not record the relationship between such tags [20] [25] [14]. However, this informality also makes tags relatively easy to add [24], and their abundance means they are a rich source of unstructured metadata linked with information published on the Internet.

To address the problem we describe, researchers have started to look for underlying structure in folksonomies. An interesting approach is based on clustering techniques [19] [11], which find groups or *clusters* of related tags. Clustering might suggest ways to identify implicit concept definitions inside a folksonomy, and might be a starting point for automating the extraction of formal vocabularies from unstructured folksonomies. Formal vocabularies could then be used to greatly improve retrieval or navigation of tagged items and assist users in choosing which tags to use.

In this work we use a Self Organizing Map (SOM) [17] to cluster tagged bookmarks taken from the website `Delicious` [2]. With `Delicious`, users save bookmarks to the website and can associate tags with each bookmark.

Self organizing maps are artificial neural networks that map high-dimensional data points to nodes in a low-dimensional grid, the *output layer*. Usually the grid is two-dimensional and can be viewed as a graphical map, where similar data points are placed close together or at the same point on the map.

We investigate the ability of SOMs to:

- cluster tags associated with bookmarks in meaningful ways;
- classify new tagged bookmarks in previously identified clusters.

The first feature would be useful in presenting user bookmarks in a graphical way, showing related bookmarks. It could also be used when a user enters a query or adds a tag to an item, to suggest related tags from within a cluster. The second feature would be useful in automatically suggesting additional tags for new bookmarks. Both features provide a way to identify bookmarks related to a given bookmark.

1.1 Related Work

A broad overview of approaches to extracting structure from folksonomies is given in [23]. Several pieces of related work explore explore tag clustering using graph-based methods. [10] creates a graph of tagged items, where edges are generated between items with shared tags. This graph is then divided to find clusters of items and associated tags. A similarity with our approach is that tags can appear in multiple clusters and items in one cluster. [10] suggests the clusters can be used to labeled disambiguate tags that could relate to several topics, and therefore appear in several clusters.

Graph-based clustering may also use graphs of tags, where weighted edges represent co-occurrence [11] [22] [13] or cosine similarity [16] between tags. First, tags are clustered so that tagged items may appear in multiple clusters. This reflects the idea that tags may identify several aspects of a single item, including multiple topics, dates, related people or locations. However, unlike our methods here, it does not allow ambiguous tags to be present in more than one cluster.

It is sometimes possible to use the text content of tagged items as input vectors for clustering [12], but this is unsuitable for items such as images with no text content.

An advantage of SOMs over other methods is that the clustering step itself produces a graphical map of the tagged items and folksonomy. Graph-based clustering methods such as [22] have also been used to produce a visual graph of tags: these graphs are often more complex, with many edges, and require

expensive layout algorithms. A simpler representation is a clustered tag cloud [15], where tags are shown textually with font sizes proportional to their frequency, with one line of text per cluster. However, SOMs place related clusters near to one another, so can convey more information than a clustered tag cloud.

2 Clustering Delicious Bookmarks using an SOM

To cluster bookmarks, bookmarks are assigned to nodes in the SOM so that several similar bookmarks are assigned to the same node, thus creating a cluster. Each bookmark has a feature vector used as input to the SOM, containing the number of times each tag was used with that bookmark. The SOM clusters bookmarks with the most similar feature vectors. The following sections describes the method used here in depth.

2.1 Software Architecture

Figure 1 shows the software architecture used, indicating the steps used to gather and process the data. The blocks in orange are software components, written in Java, whereas green blocks are exchanged data.

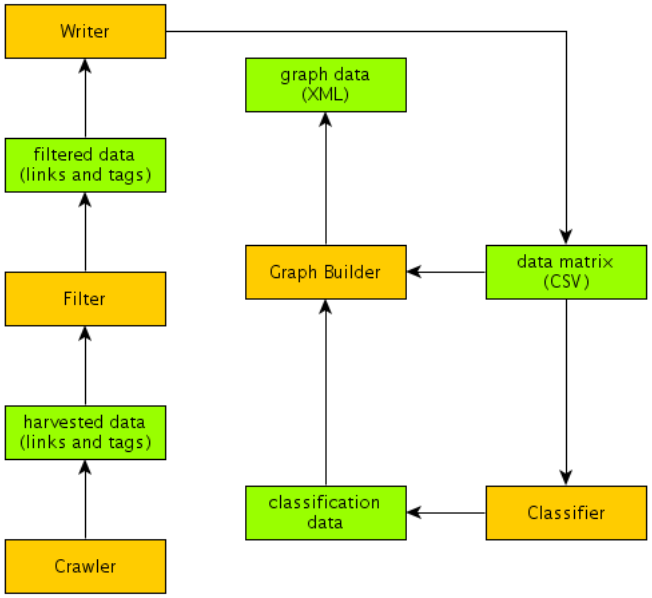

Figure 1: Software Architecture

More detail about each part of the architecture is given below.

2.2 Obtaining the Dataset

Before clustering with an SOM, data must be selected from Delicious and converted into suitable input for the SOM.

2.2.1 Crawler

The Crawler gathers data from `http://del.icio.us` and builds internal data structures. Gathered data is organized in two sets:

- $\mathcal{B}$: set of bookmarks
- $\mathcal{T}$: set of tags associated with the elements of $\mathcal{B}$.

We also define the following sets:

- $\mathcal{B}_t \subset \mathcal{B} = \{b \in \mathcal{B} \mid b \text{ is tagged with } t\}$
- $\mathcal{T}_b \subset \mathcal{T} = \{t \in \mathcal{T} \mid t \text{ is associated with } b\}$

Input to the crawler is a starting tag t_{start}. The crawler fetches bookmarks with this tag, $\mathcal{B}'_{t_{start}} \subset \mathcal{B}_{t_{start}}$. For each bookmark $b \in \mathcal{B}'_{t_{start}}$ the Crawler gathers the set of all tags for this bookmark, $\mathcal{T}_b = \{t_1, \ldots, t_n\}$. It then recursively repeats the process for each tag $\{t_1, \ldots, t_n\}$.

During this process, the Crawler builds a data structure Δ, which is essentially an associative map where each bookmark b is associated with its corresponding set of tags $\mathcal{T}_b$.

$$\begin{array}{ll} b_1 & \mathcal{T}_{b_1} \\ b_2 & \mathcal{T}_{b_2} \\ \ldots & \ldots \\ b_n & \mathcal{T}_{b_n} \end{array}$$

Recursively, the crawler explores $\mathcal{T}$ until Δ contains a set of bookmarks $\mathcal{B}_\Delta$ whose cardinality is greater or equal to a predefined threshold $\mathcal{C}_{crawler}$.

The Crawler uses Jena [4] to read and parse the RSS [6] data from `http://del.icio.us`. The elements of both $\mathcal{B}$ and $\mathcal{T}$ are represented as URLs (for example the tag *web* is identified by `http://feeds.del.icious.com/rss/tag/web`.

2.2.2 Filter

The Filter performs some cleaning of the data gathered by the Crawler. Experiments showed that Δ often contains many tags that are associated with very few bookmarks: such tags may not provide useful information for clustering of bookmarks, as too few bookmarks within a desired cluster would have this tag. They may also represent mistakes by users when entering the tag.

Let n_j be the number of times that tag t_j appears in Δ, and let $N = \sum_j n_j$; we define the frequency of tag t_j as $f_j = \dfrac{n_j}{N}$. The Filter works in two steps:

1. It removes from Δ all tags whose frequency is less than a predefined threshold f_{filter};
2. It removes from Δ all bookmarks b_i whose corresponding T_{b_i} is empty because of step 1.

We call Δ_f the filtered version of Δ.

2.2.3 Writer

The Writer simply transforms Δ_f into a matrix Ω, which is given as input to the Classifier. Ω has the following structure:

$$\begin{array}{cccccccc} & t_1 & t_2 & \dots & t_j & \dots & t_n \\ b_1 & \omega_{11} & \omega_{12} & \dots & \omega_{1j} & \dots & \omega_{1n} \\ b_2 & \omega_{21} & \omega_{22} & \dots & \omega_{2j} & \dots & \omega_{2n} \\ \vdots & \vdots & \vdots & & \vdots & & \vdots \\ b_i & \omega_{i1} & \omega_{i2} & \dots & \omega_{ij} & \dots & \omega_{in} \\ \vdots & \vdots & \vdots & & \vdots & & \vdots \\ b_m & \omega_{m1} & \omega_{m2} & \dots & \omega_{mj} & \dots & \omega_{mn} \end{array}$$

where

$$\omega_{ij} = \begin{cases} 1 & \text{if } t_j \in \mathcal{T}_{b_i} \\ 0 & \text{if } t_j \notin \mathcal{T}_{b_i} \end{cases}$$

The set $\{\omega_{i1}, \omega_{i2}, \dots, \omega_{in}\}$ is the *pattern* associated with b_i.

2.3 Classifier

The Classifier is a Self Organizing Map (SOM) which is trained in unsupervised mode using as input vectors the rows of Ω. the classifier maps bookmarks to nodes in the output layer of the SOM, each node forming a cluster of bookmarks. Unfortunately it is impossible to know in advance the expected number of clusters - and therefore the size of the output layer - so it is necessary to develop strategies for estimating it. We adopted the following initial strategy:

- let c_0 be the expected number of clusters; initially we have $c_0 = 0$
- we compute the frequency f_j of each tag $t_j \in \Delta_f$;
- let $f_{cluster}$ be a predefined threshold;
- for each tag t_j whose frequency $f_j > f_{cluster}$ we increment c_0, assuming that each of these tags potentially generates a cluster

Our initial strategy makes the assumption that the number of expected clusters is given by the number of tags in Δ_f whose frequency is greater than a predefined threshold. We set the size of the output layer of the SOM equal to the expected number of clusters c_0.

The SOM is created and trained using the Joone library [5]. The training phase is characterized by a learning rate, and lasts for a number of epochs; both parameters can be configured. When the training phase is complete, we again feed each row of Ω to the classifier, so that it is classified into a cluster.

2.4 Graph Builder

The Graph Builder produces a graphical representation of the clusters computed by the Classifier. To produce this representation the graph builder incorporates yEd - Java Graph Editor [8], and Aduna ClusterMap[1]. Both take in an XML file produced by specialized versions of the Graph Builder component.

3 Experiments

We conducted two kind of experiments:

Clustering Using Tags : creation and training of Self Organizing Maps to cluster bookmarks, and to produce a graphical representation of clusters;

New Bookmark Classification : classification of new tagged bookmarks into existing clusters using trained Self Organized Maps.

For the first kind of experiment, Clustering Using Tags, we tried different configurations of the system parameters:

- the starting tag $t_{start} \in \mathcal{T}$ used by the Crawler
- the minimum cardinality $\mathcal{C}_{crawler}$ of the set $\mathcal{B}_{\Delta}$
- the threshold f_{filter}
- the threshold f_{class}
- the learning rate and the number of epochs used for the SOM training.

We drew some empirical conclusions from experimenting with these system parameters:

- The value of f_{filter} must not be too high, otherwise the Filter cut off too many patterns. From our experiments we can conclude that if $\mathcal{C}_{crawler} \leq 1000$, then we can set $f_{filter} = \frac{2}{1000}$.
- If $\mathcal{C}_{crawler} \leq 1000$, then the SOM can be trained within 1000 epochs, and with a learning rate of 0.7.

The following section describes the details of one run of the first experiment. As explained below, we found that our initial strategy to compute the number of expected clusters c_0 usually underestimates the number of clusters. Hence we developed and tested an iterative process, which trains the SOM setting at every iteration an increasing value of c_0.

3.1 Clustering Using Tags: Results

In this experiment we used the following settings:

- $t_{start} =$ `http://feeds.del.icious.com/rss/tag/web`
- $\mathcal{C}_{crawler} = 500$
- $f_{filter} = 0.002$
- $f_{class} = 0.015$
- The SOM was trained in 1000 epochs with a learning rate of 0.7.

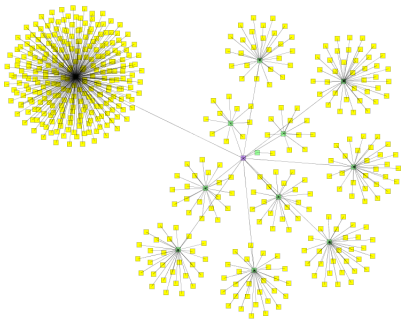

Figure 2: Classification of tag *web* in 12 cluster; picture generated with yEd

The matrix Ω^{web} has 498 rows (distinct bookmarks) and 67 columns (distinct tags). Our initial strategy computed $c_0 = 12$. Figure 2 shows a representation of the 12 clusters generated with the yEd editor. Yellow nodes represent the 498 bookmarks; they are connected to a central node, which represents the cluster. Each cluster is connected to a central node, which may be thought of as the common super-class.

Figure 2 contains two anomalous clusters: one containing a single element, and one containing many more elements than the other clusters.

Figure 3 shows the same results as Figure 2 using the tool Aduna Cluster Map. Clusters are shown as sets; they are connected to the common super-class, which has been labelled *Thing*. Figure 3 shows also the clusters cardinality, and gives a name to each cluster. The cluster name has been automatically generated by the Graph Builder component, and it is given by the concatenation of the most frequent tags associated with the bookmark in the cluster.

The presence of an over-populated cluster suggests that the clustering is not optimal, as we would prefer small clusters tat make it easier to select an individual item from a cluster and group more closely related items. The result is not affected by the number of epochs or the learning rate.

We repeated the experiment using an iterative process which progressively increases the number of clusters: at the i^{th} iteration, the value of expected clusters is $c_i = c_0 + 3i$ (where $c_0 = 12$). Comparing the results obtained in the various iteration we see that:

- increasing c_i the distribution of elements in the clusters is more uniform, and therefore over-populated clusters tend to disappear;
- some clusters are "persistent": increasing c_i such clusters remain almost identical, that is they contain the same bookmarks;
- some clusters disappear, and others emerge when c_i increases;
- the distribution of bookmarks in cluster remains almost stable when c_i reaches a value c_{best}, which is the optimal value of clusters for the dataset. In our case $c_{best} = 36$.

Figure 4 compares the distribution of bookmarks in clusters during the various iterations with $c_i = 12, 15, \ldots 36$. The *ClassID* is a unique identifier for each

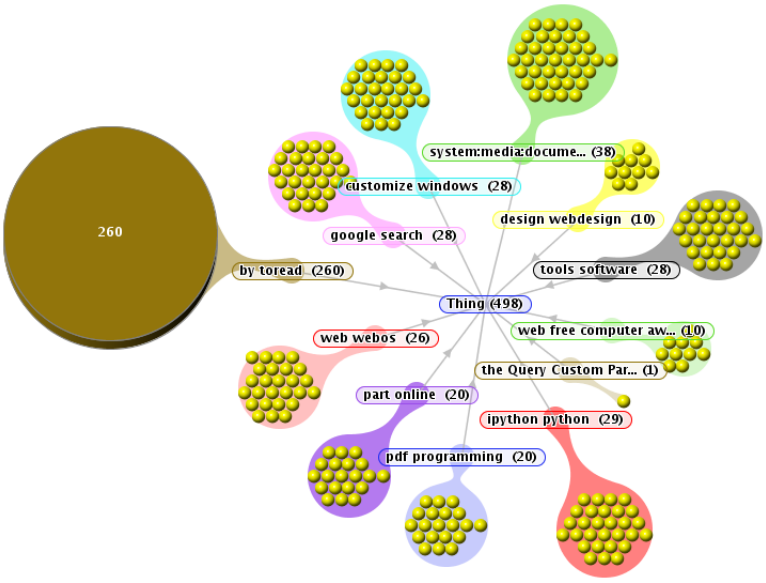

Figure 3: Classification of tag *web* in 12 cluster; picture generated with Aduna Cluster Map

cluster. A test is carried out at each iteration. Some *ClassID*s appear only in some tests. Tests are identified by $T12, T15, \ldots$, where $T12$ is the test with 12 clusters, $T15$ is the test with 15 clusters, etc. The vertical bars shows the number of bookmarks in each cluster. Figure 4 clearly shows that over-populated clusters progressively disappear, and that some clusters are persistent (for example, $C52$). It is also evident that some new clusters appear when c_i increases (for example $C57$ emerges when $c_i \geq 30$), and that some others disappear (for example $C61$ emerges when $c_i \geq 30$).

Table 1 summarizes the number of bookmarks in some persistent, emerging or disappearing clusters for the various tests with increasing values of c_i. Green columns (C11, C21, C35, C52) show data of persistent clusters; violet clusters (C19, C49, C61) show data of disappearing clusters; azure columns (C10, C20, C57) show data of emerging clusters.

Table 1: Persistent, emerging and disappearing clusters for the various $c_i = 12, 18, \ldots, 48$

	C10	C11	C19	C20	C21	C35	C49	C52	C57	C61
T12	0	28	0	0	29	38	0	10	0	26
T15	0	27	261	0	29	38	14	10	0	0
T18	0	28	0	0	30	37	9	10	0	24
T21	0	27	199	0	30	37	0	10	0	0
T24	0	27	0	0	29	37	9	10	0	24
T27	25	27	0	0	29	37	9	10	0	22
T30	0	26	0	28	29	37	0	10	3	0

Continued on next page

Table 1 – Continued from previous page

	C10	C11	C19	C20	C21	C35	C49	C52	C57	C61
T33	25	26	0	0	29	37	0	10	3	0
T36	25	26	0	28	29	37	0	10	3	0
T39	25	26	0	28	29	37	0	10	3	0
T42	25	0	0	28	29	37	0	10	3	0
T45	25	26	0	28	29	37	0	10	3	0
T48	25	0	0	28	24	37	0	10	3	0

C10 → custom design
C11 → customize windows
C19 → ingresos toread
C20 → ingresos web
C21 → ipython python
C35 → system:media:document system:filetype:pdf
C49 → web design webdesign
C52 → web free computer awsome freebie tech content pc blog
C57 → web tutorial webdesign design css
C61 → web webos

We call SOM_{36}^{web} the Self Organizing Map with $c_{best} = 36$ nodes in its output layer, trained using Ω^{web}. Table shows how data in Ω^{web} are classified using SOM_{36}^{web}: each row lists the most frequent tags in the cluster, the unique cluster ID, and its cardinality.

Table 2: Clusters generated by SOM_{36}^{web}

Most frequent tags	**ID**	$\mathcal{C}$
by of blog	C8	25
content writing	C1	16
custom design	C26	25
customize windows	C11	26
flash development documentation content video	C0	1
google search	C30	27
html development tips tutorial web2.0 webdesign article ajax programming css design howto reference	C32	1
ingresos web	C19	28
ipython python	C23	29
java query database search free security	C34	2
learning programming pdf reference tutorial	C29	3
part online	C7	20
pdf web2.0 tools	C33	17
query database	C4	21
search tools reference web online cool design	C17	2
security video	C5	22
software windows	C28	23

Continued on next page

Table 2 – Continued from previous page

Most frequent tags	ID	$\mathcal{C}$
system:media:document system:filetype:pdf	C6	37
The - Part	C21	28
the of	C27	22
the Query Custom Part Customize Web using by Content	C13	1
tools software	C9	20
tools video software	C2	3
toread blogs article programming tips security cool design blog howto reference	C35	27
using by	C18	24
web css design	C15	4
web firefox tools search programming cool custom customize	C20	2
web free computer awsome freebie tech content pc blog	C22	10
web security	C10	14
web system:filetype:pdf google toread pdf security system:media:document	C25	1
web tutorial webdesign design css	C16	3
web webdesign tips design reference	C14	3
web webdesign tools html development programming software	C3	1
web webos ajax	C12	6
web2.0 free content cool video design database	C31	1
webdesign design tools html	C24	3

Very common words such as “the” and “of” appear as the most frequent tags in some clusters. These tags are not subject-specific so do not indicate clearly whether these are useful clusters. It may be possible to improve the clusters by filtering out tags that appear in a list of common words (stopwords) before running the clustering algorithm. Table 2 also shows more clearly meaningful clusters, such as “ipython python” and “google search” where the most common tags are known to refer to closely-related topics. Using the iterative process to determine $c_{best} = 36$ has produced a largest cluster of cardinality 38, while 15 other clusters are at least half as big as this cluster; this suggests the iterative process for finding c_{best} has successfully avoided creating over-populated clusters.

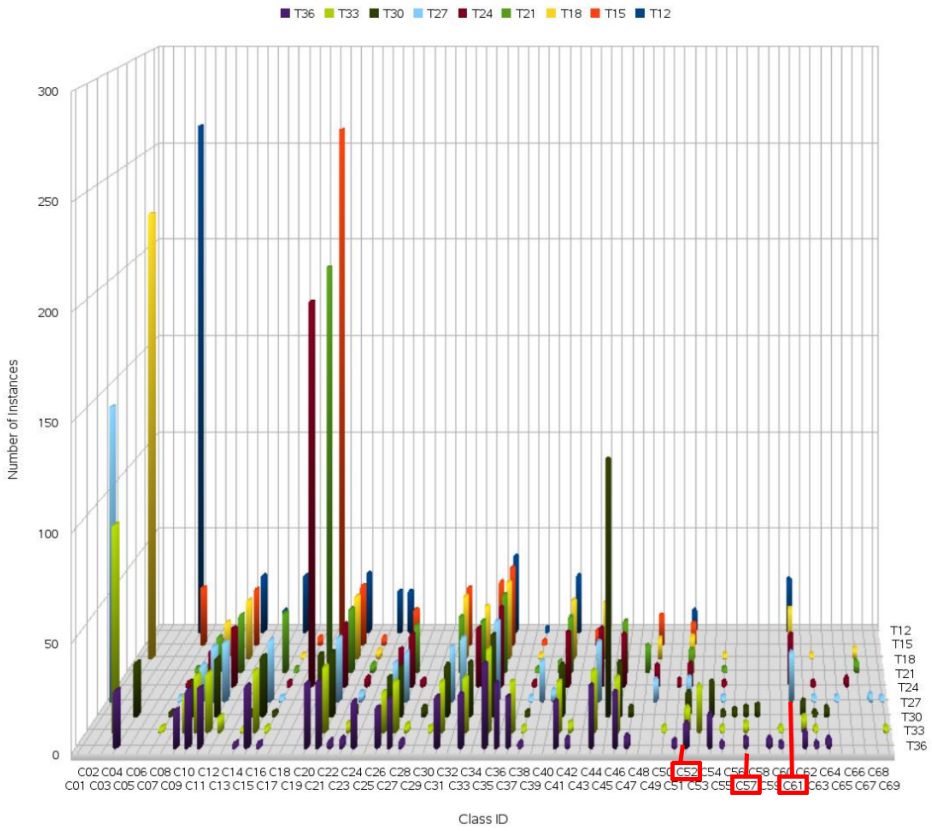

Figure 4: Clustering of tag *web* in 12, 15, . . . , 36 cluster

4 Classification of New Tagged Bookmarks

This experiment uses SOM_{36}^{web} described in section 3.1 to classify new tagged bookmarks. We performed some tests giving some fictitious bookmarks as input to SOM_{36}^{web}; the fictitious bookmark were tagged using some of the tags known by SOM_{36}^{web}. Table 3 shows some experimental results.

Table 3: Classification of new tagged bookmarks using SOM_{36}^{web}

Test ID	Tag	Resulting Class ID	Expected Class ID
01	ipython python programming reference	C23	C23
02	ipython python programming reference	C23	C23
Continued on next page			

Table 3 – Continued from previous page

Test ID	Tag	Resulting Class ID	Expected Class ID
	system:filetype:pdf		
03	ipython python programming reference system:filetype:pdf part	C23	C23
04	ipython python programming reference system:filetype:pdf part Customize	C23	C23
05	ipython python programming reference system:filetype:pdf part Customize windows	C23	C23 o C11
07	ipython python web tutorial webdesign design css	C16	C16 o C23
09	ipython python programming reference web tutorial webdesign design css	C16	C16 o C23
13	ipython python programming development web tutorial webdesign design	C23	C16 o C23

Continued on next page

Table 3 – Continued from previous page			
Test ID	**Tag**	**Resulting Class ID**	**Expected Class ID**
	css		

Data in table 3 shows that SOM_{36}^{web} is able to classify new bookmarks in a reasonable way, when the bookmarks have tags known by SOM_{36}^{web}:

- Test 01 shows the result obtained using a bookmark whose tags are the same as the most frequent tags of cluster C23. Tests 02, 03, and 04 add other tags, but the the result does not change. Test 05 add the most frequent tags of cluster C11, however the result remains the same.
- Test 07 and 09 (similarly to test 05) use the most frequent tags of clusters C16 and C23, and show that C16 has stronger influence on the result.
- Test 13 must be compared with test 09: they differ only for one tag (*development* instead of *reference*), and this yields a different result from the SOM.

5 Conclusion and Future Work

Our experiments show that an SOM can produce many reasonable clusters of bookmarks. Using an iterative process to find the optimal number of clusters was effective in producing clusters of a reasonable size with a reasonable distribution of bookmarks. Future work may further investigate how effective the clustering is for different types of dataset: the nature of investigation would depend on the intended application of the clusters, whether they are being used, for example, to recommend similar bookmarks or tags, or for users to browse tagged bookmark collections. While SOM clustering appears effective, a comparison of with graph-based hierarchical clustering would indicate more clearly when to select one algorithm over the other.

When classifying new tagged bookmarks in a real application, new bookmarks may have tags not known to the SOM; currently these would be discarded as infrequent tags. However, if a large number of new bookmarks have an unseen tag, these unseen tags have a higher frequency and may indicate that the new bookmarks are related in some way not represented by the SOM. Therefore, future work may investigate the best strategy for updating the SOM as new bookmarks and new tags appear.

References

[1] Aduna cluster map. http://www.aduna-software.com/technologies/clustermap/overview.view.

[2] Delicious. http://delicious.com/about/.

[3] Flickr. http://www.flickr.com.

[4] Jena semantic web framework. http://jena.sourceforge.net/.

[5] Joone - java object oriented neural engine. http://www.jooneworld.com/.

[6] Really simple syndication - rss. http://en.wikipedia.org/wiki/RSS_(file_format).

[7] Web 2.0. http://en.wikipedia.org/wiki/Web_2.

[8] yed - java graph editor. http://www.yworks.com/en/products_yed_about.html.

[9] Youtube. http://youtube.com.

[10] *Tag Meaning Disambiguation through Analysis of Tripartite Structure of Folksonomies*, 2007.

[11] G. Begelman, P. Keller, and F. Smadia. Automated tag clustering: Improving search and exploration in the tag space. volume Worldwide Web Conference 2006, 2006.

[12] C. H. Brooks and N. Montanez. Improved annotation of the blogosphere via autotagging and hierarchical clustering. In *15th World Wide Web Conference*, 2006.

[13] C. Van Damme, M. Hepp, and K. Siorpaes. Folksontology: An integrated approach for turning folksonomies into ontologies. In *European Semantic Web Conference*, 2007.

[14] Scott Golder and Bernado Huberman. Usage patterns of collaborative tagging systems. *Journal of Information Science*, 32(2):198–208, 2006.

[15] Y. Hassan-Montero and V. Herrero-Solana. Improving tag-clouds as visual information retrieval interfaces. In *International Conference on Multidisciplinary Information Sciences and Technologies*, 2006.

[16] P. Heymann and H. Garcia-Molina. Collaborative creation of communal hierarchical taxonomies in social tagging systems. Technical report, Dept. of Computer Science, Stanford University, 2006.

[17] Teuvo Kohonen. *Self-Organizing Maps*, volume 30 of *Series in Information Sciences*. Springer, 3rd ed. edition, 2001.

[18] A Mathes. Folksonomies: Cooperative classification and communication through shared metadata. http://www.adammathes.com/academic/computer-mediated-communication/folksonomies.html, 2004.

[19] Peter Mika. Ontologies are us: A unified model of social networks and semantics. *Journal of Web Semantics, Elsevier*, 5(1), 2007.

[20] Alireza Noruzi. (un)controlled vocabulary. *Knowledge Organization*, 33(4):199–203, 2006.

[21] Tim O'Reilly. What is web 2.0. http://www.oreillynet.com/lpt/a/6228.

[22] Edwin Simpson. Clustering tags in enterprise and web folksonomies. Technical report, Hewlett-Packard Labs, 2007.

[23] Edwin Simpson and Mark H. Butler. *Collaborative and Social Information Retrieval and Access: Techniques for Improved User Modeling*, chapter Analyzing Communal Tag Relationships for Enhanced Navigation and User Modeling. IGI Global, 2008.

[24] Rashmi Sinha. A cognitive analysis of tagging (or how the lower cognitive cost of tagging makes it popular). http://www.rashmisinha.com/2005/09/a-cognitive-analysis-of-tagging.

[25] E Speller. Collaborative tagging, folksonomies, distributed classification or ethnoclassification: a literature review. *Library Student Journal*, 2007.

[26] Thomas Vanderwal. Folksonomy coinage and definition. http://www.vanderwal.net/folksonomy.html, 2007.