Utilizing Folksonomy:
Similarity Metadata from the Del.icio.us System
CS6125 Project

Blake Shaw <bs2018@columbia.edu>

December 9th, 2005

1 Proposal

1.1 Abstract

Traditionally, metadata is thought of simply as keywords that describe some content, and while the primary aim of folksonomic systems like the Del.icio.us bookmarking tool is to produce these keywords, a richer set of metadata is also produced. Because these keywords are now contributed from many different individuals and aggregated, useful information comes not only from the keyword itself but also from the information about who contributed to labeling the content with that keyword. This idea can be broadened to a general framework for producing a new layer of metadata: similarity between concepts. By analyzing the distributions of how users apply tags, how tags are applied to links, and how users pick content, we should be able to calculate the "distance" between tags, users, and content. This "distance" metric could then be used to construct a more powerful tool for browsing content, allowing the user to specify a query made up of keywords, content, or even other users. Furthermore, this metadata can be condensed into a lower dimensional space and visualized in order to gain better insight into the relationships between the concepts themselves.

1.2 Introduction

1.2.1 Del.icio.us

Del.icio.us is a web-based bookmarking system. Each user adds bookmarks, and then labels them with keywords. For example a user might bookmark *apple.com* and tag it with *computer, tech, design*. Del.icio.us is a perfect example of a folksonomy (see figure 1). Users tag small parts of the web, and then when all of these tags are aggregated, a large comprehensive set of metadata is produced. Del.icio.us uses this metadata to allow users to query for items given a certain tag, or to browse through users or tags.

1.2.2 Collecting Data from Del.icio.us

The data that Del.icio.us collects opens up the possibility for more interesting analysis, and visualization using algorithms and techniques borrowed from machine learning, however only if this data can be collected and formatted appropriately. I propose building a tool which downloads RSS feeds from Del.icio.us for a set of users, parses the feeds using an XML SAX parser, and loads this data into a mysql database. The data can then be analyzed for simple statistics and similarity information, as well as be fed into a visualization algorithm such as Locally Linear Embedding or Semidefinite embedding.

1.3 Functionality

- Collect data from a sample of Del.icio.us users by means of RSS aggregation

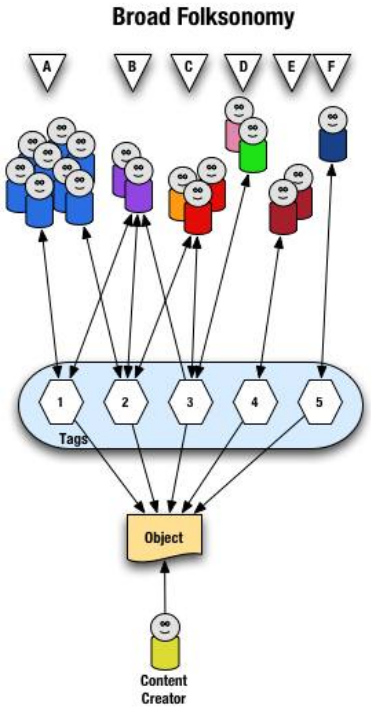

Figure 1: Structure of Del.icio.us. Users tag objects (in this case links) with a vocabulary of tags. [1]

- Load this data into a mysql database
- Compute a "distance" between tags, users, and links
- Package this information to be used in matlab for visualization algorithms
- Provide a web-based proof-of-concept browser that enables the user to browse for content by specifying a query of keywords while providing the user with suggestions for related keywords and showing the relationship between them on a 2-dimensional map

1.4 Technology

1.4.1 Languages

- **Java** – Used for connecting to del.icio.us, downloading the RSS feeds and parsing them using a custom Java XML SAX parser, as well as loading data into the mysql database and a sparse matrix format, and computing the "distance" metric (KL-Divergence).
- **Matlab** – Used for producing the data needed for proper visualization.
- **Python Server Pages, Javascript, Flash** – For a flexible web-based front end that can dynamically update using AJAH (Asynchronous Javascript and HTML) and is capable of displaying interactive maps

1.4.2 Algorithms and Metrics

- **KL-Divergence** A way to calculate the distance between two probability distributions by looking at cross-entropy.

- **Semidefinite Embedding (SDE)** An algorithm for reducing high-dimensional data so it can be visualized in 2 dimensions.

1.4.3 Existing packages

- The standard JAVA SAX XML parser will need to be customized for parsing Del.icio.us RSS feeds.
- A Java Sparse Matrix implementation
- A Java mysql interface
- A framework for doing SDE which I wrote for another project

1.5 Deliverables

There are two main deliverables for this project: 1) a simple proof-of-concept browser which takes advantage of similarity metadata 2) maps of tags which show these keywords projected onto a 2-dimensional plane arranged in terms of similarity, highlighting clusters and relationships among tags.

2 Architecture and Design

2.1 Introduction

There are 4 high-level components that need to be built for this system, each of which contains a variety of sub-components. These 4 components are highlighted in grey in Figures 3 and 5, and 6, and will be discussed in depth in the following sections:

- A tool for downloading RSS content from Del.icio.us, and parsing that information into a mysql database.
- An analysis engine which computes the "distances" between items
- A package for computing a low-dimensional embedding of the metadata for visualization
- A web-based front end which leverages this similarity metadata to allow the user to browse through tags and explore the relationships visually on a map

2.2 RSS Collector

2.2.1 Collecting RSS data

Aggregation of the RSS feeds will be handled by the class Collector.java (see Figure 2 for information about class stucture). The first objective of this component is to download RSS feeds from Del.icio.us.
Five different kinds of feeds are made available by Del.icio.us as follows:

- http://del.icio.us/rss/url?url=
  How users tagged a specific URL.
- http://del.icio.us/rss/tag/tagname
  Which links were tagged with a specific tag.
- http://del.icio.us/rss/username
  What links did a user tag, and with what tags.
- http://del.icio.us/rss/popular
  What links are popular.
- http://del.icio.us/rss/popular/tagname
  What links are popular pertaining to a certain tag.

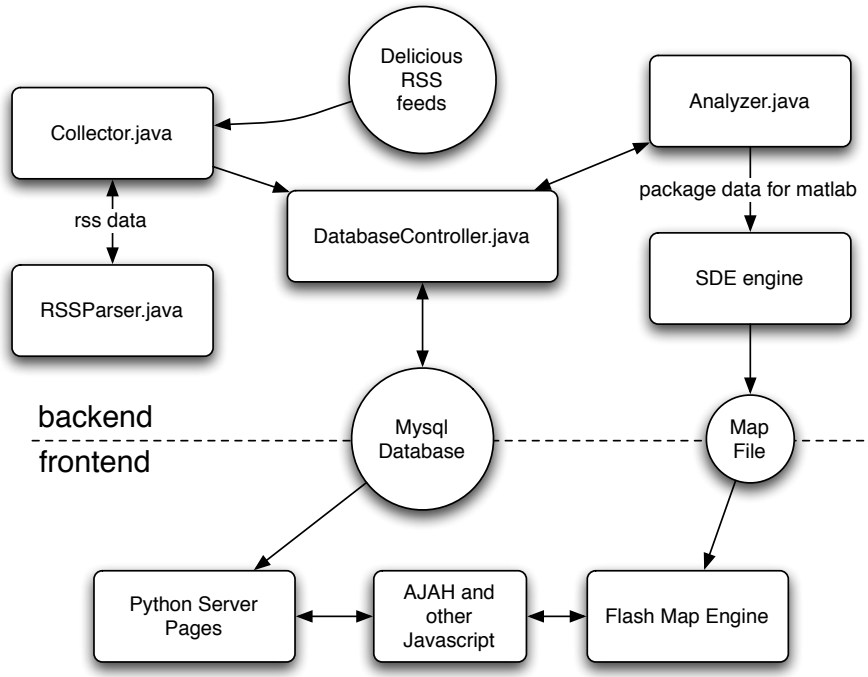

Figure 2: An overview of class structure of the Del.icio.us analysis system.

The collector will begin by downloading the popular feed and a few other feeds (either specific links, tags, or popular tags) simply to collect a sample list of users. Once this seed of users is created, the collector will then download the RSS feed for each user (see Figure 3).

2.2.2 Parsing RSS data

Parsing of the RSS feeds will be handled by the class RSSParser.java which extends the default handler for the org.xml.sax parser. Here is a snippet of a typical RSS feed:

```
<item rdf:about="http://slashdot.org/">
  <title>Slashdot</title>
  <link>http://slashdot.org/</link>
  <dc:creator>metablake</dc:creator>
  <dc:date>2005-09-13T23:12:03Z</dc:date>
  <dc:subject>apple nerd tech</dc:subject>
  <taxo:topics>
    <rdf:Bag>
      ...
    </rdf:Bag>
  </taxo:topics>
</item>
```

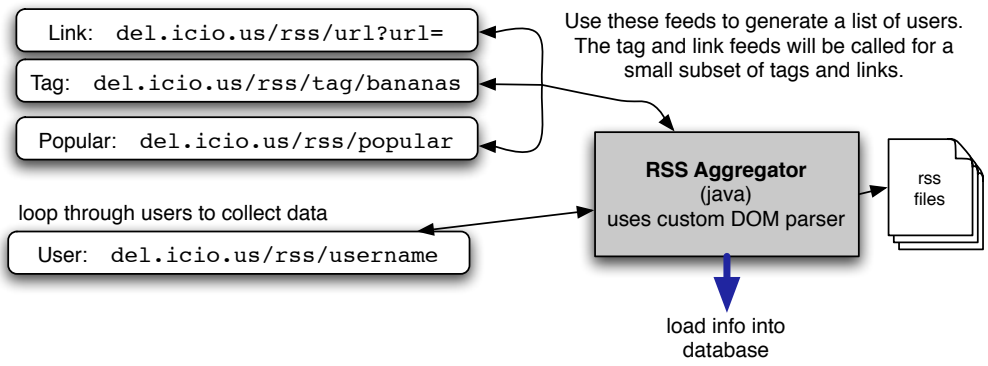

Figure 3: The system for aggregating RSS feeds.

For each of these items, the RSS parser will extract the items title, link, user, and tags (taken from the subject field). These items will then be passed back to the collector to be stored.

2.2.3 Storing and transforming the data

After the data is collected, it will be loaded into a mysql tables. Below is an outline of the schema.

- **user** – id, username (id is a hash of username)
- **tag** – id, tagname (id is a hash of tagname)
- **link** – id, link title, link url (id is a hash of link title)
- **usertaglink** – userid, tagid, linkid

The data will then be collected from the database and formatted for use in matlab (Sparse Matrix Format implemented with the MTJ package: http://rs.cipr.uib.no/mtj/). Matlab will then produce a map file to be read by the Flash map engine.

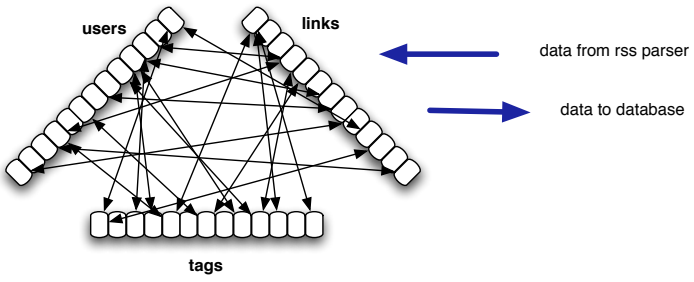

Figure 4: Intermediate data structures

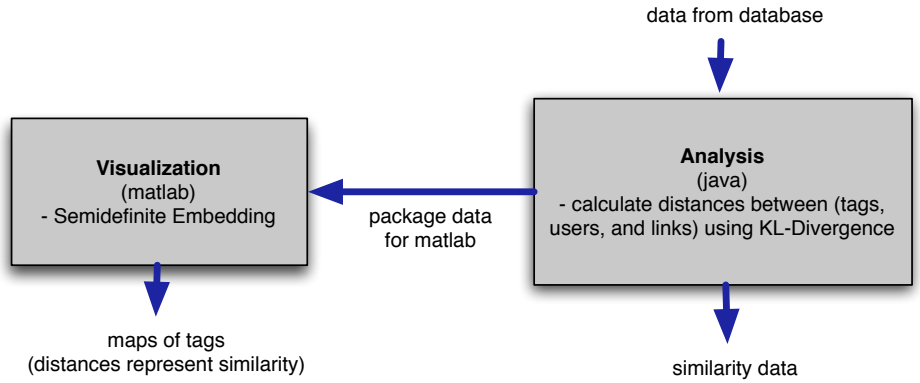

Figure 5: Structure of the analysis section

2.3 Analysis

Analysis will be handled by the Analyzer.java class, and a set of matlab scripts (see Figure 5). The analyzer will collect data from the database and use the KL-divergence metric to compute the distances between tags. This can be easily expanded to compute the distances between users, and links as well; however, for this project we focus mainly on tags. These distances will be entered back into the database, and will allow for a simple query function to retrieve tags which are close to other tags. Furthermore, the analyzer will package the data to be sent to matlab for further analysis of simple statistics, and visualization.

2.3.1 Computing distance matrices

Kullback-Leibler divergence

$$distance(P,Q) = \sum_{i=0}^{D} P_i \log \frac{P_i}{Q_i} \tag{1}$$

Where P and Q are probability distributions, (such as the probability distribution that a user will use a given tag)

From my experience, KL-Divergence seems to be the optimal distance metric for this problem; however, I may experiment with variants of it as well.

The analyzer will use this metric on every pair of tags (or users, or links), to compute the distances between each pair, and store the results back in the database.

2.3.2 Simple statistics

There are a number of simple statistics that can be generated:

- Number of users, tags, and links collected.
- Average tags per user, tags per link, and links per user.
- Variance of tags per user, tags per link, and links per user.

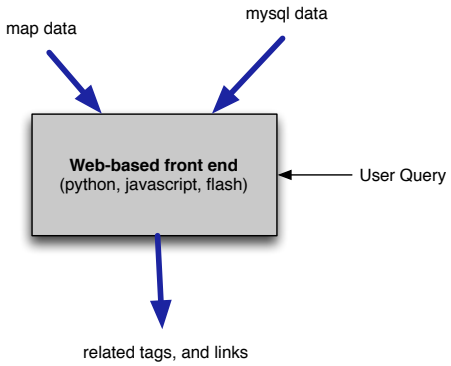

Figure 6: Structure of web-based front-end

2.3.3 Semidefinite Embedding

Semidefinite Embedding is an algorithm for non-linear dimensionality reduction. It tries to find a low-dimensional manifold which best approximates data in a high-dimensional space by means of transforming the problem into a semidefinite programming problem. This algorithm will allow for the visualization of tags in a 2 dimensional space. For more information about the details of the algorithm, please consult [4].

2.4 Web-based front end

The web-based front end will use the Python Server Pages to query the mysql database as well as format the data to be displayed in HTML. A simple AJAH (Asynchronous Javascript and HTML) package will be used to allow for dynamic updates of content without reloading the whole page. Furthermore, there will be a flash-based interactive map which communicates with the rest of the site via javascript, and reads map data from the map file created by matlab which is placed on the webserver. See figure 6.

3 Results

3.1 Collection and Analysis

Collector.java turned out to be very efficient, allowing me to collect the bookmarks for 1494 users relatively quickly. The dataset consisted of 27726 unique links, and 14839 tags. The entire collection process took under an hour.

3.2 Maps

The maps, as intended, show interesting features about the relationships between tags (see figure 7). One interesting feature is that one can show a meaningful correspondence between the axes of the map and attributes of the tags. The x-axis, for example, is a good measure of how much the tag corresponds to a technical idea vs. a less technical one, and the y-axis corresponds to how much the tag is related to reference materials vs. actual tools. Furthermore, we can see many other features that make sense, such as the proximity of synonyms and the existence of distinct regions which accurately represent a general concept. See figure 8.

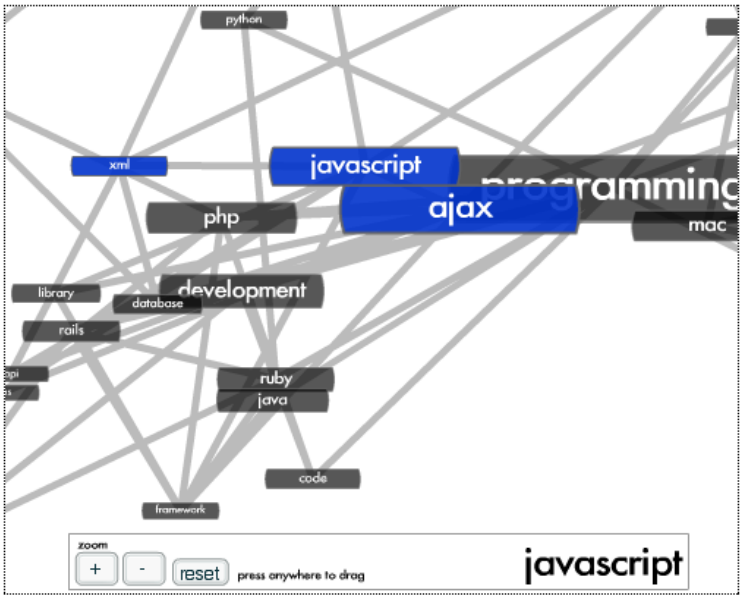

Figure 7: A close-up view of the map. This area consists primarily of programming related tags.

3.3 Proof-of-concept Browser

The browser, as shown in figure 9, is an excellent example of what is possible using this new metadata. The user builds a search query by either entering tags or clicking on tags in the map. As the user is doing this, he is presented with tags that are similar to the tags he is selecting as well. In fact, all queries are compounded by their similar tags. For example, when a user searches for apple, he also searches for mac, osx, etc. (although the weights of these similar tags in the query are significantly lower). The links are then found which best match the weights of the search query, giving bonuses to the links that contain many different tags in the search query. The result is a folksonomy-based browser that allows the user to find interesting content that is loosely related to a certain topic.

3.4 Future Directions

Currently, the browser only allows for the search terms to be tags; however, it would require little effort to extend the system to allow the user to enter any combination of tags, users, or links as the set of search terms. This would make it very easy to filter content to fit a specific user, for example. Furthermore, the browser currently automatically weights all of the terms specified for a search. Again, because of the way the system is built, it would be easy to allow the user to manually adjust how much each term contributes to the search. Another key limitation of the system at the moment is that it does not incorporate temporal data from Del.icio.us. Adding the ability to filter links by when they are posted would be a valuable feature and easy to add.

Furthemore, a way to speed up database queries is needed. Currently, the database access is very slow; that is because the indexing scheme does not incorporate this additional knowledge about the similarity between items. One could imagine a more complex indexing scheme, giving every item a set of 10 numbers for example, specifying its position in a high-dimensional space. Querying for relative content would then simply mean specifying a position and retrieving items which are within a given range for each dimension.

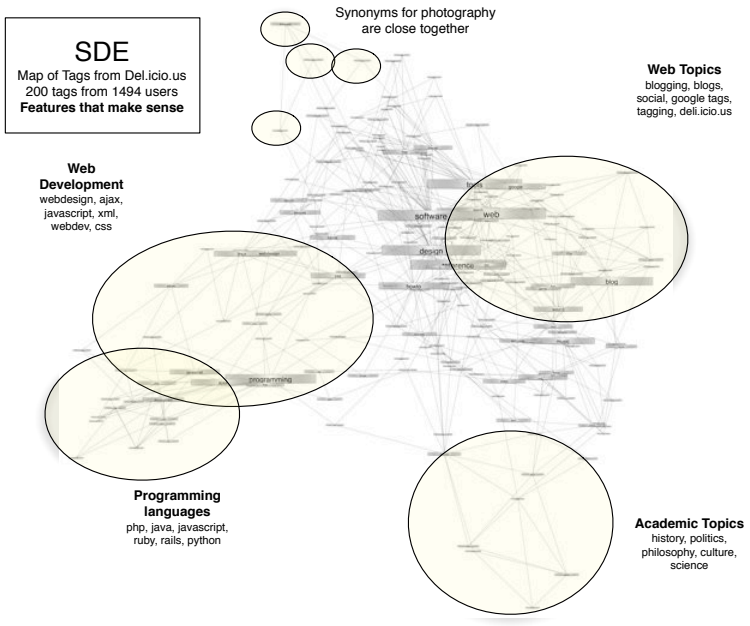

Figure 8: Map of tags. Features that make sense.

3.5 Remarks

I very much enjoyed working on this project. I learned a lot about the concept of folksonomy. Furthermore, I found it very exciting to apply machine-learning algorithms to this interesting real-world data in order to build a new kind of tool for information retrieval. Thanks so much, Professor Biliris.

References

[1] Thomas Vander Wal. Explaining and showing broad and narrow folksonomies.

[2] Wikipedia. Definition: Folksonomy.

[3] Wikipedia. Definition: rss.

[4] Wikipedia. Definition: Semidefinite embedding.

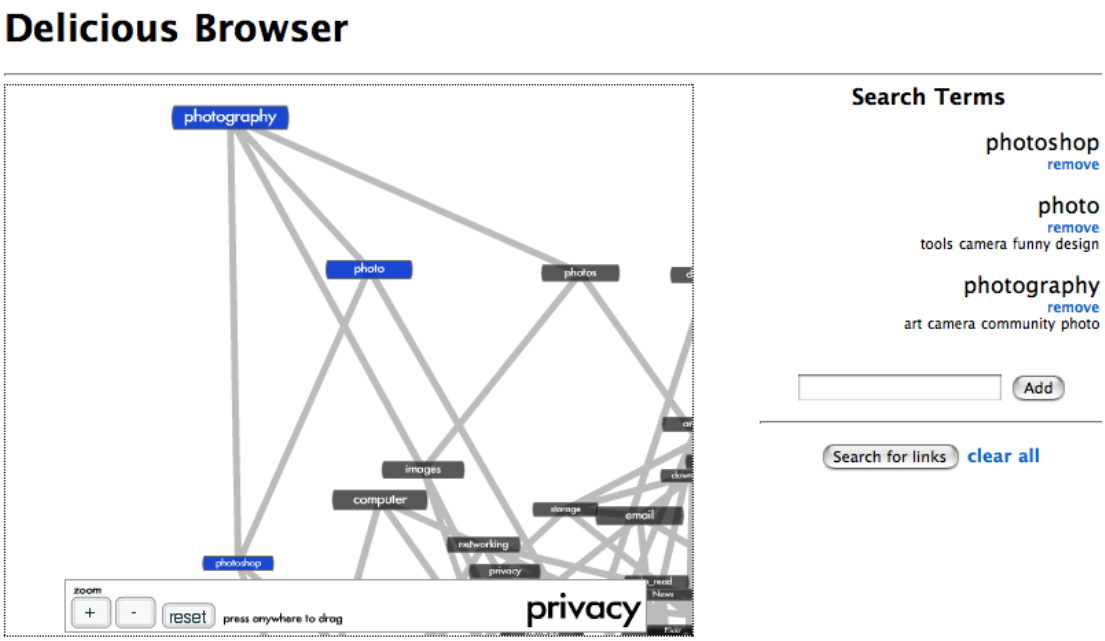

Figure 9: The proof-of-concept browser.