Generating Tagged Web Corpora for Supervised Topic Detection and Tracking

Chris Harman
Swarthmore College
500 College Ave
Swarthmore, PA 19081
charman1@cs.swarthmore.edu

Mustafa Paksoy
Swarthmore College
500 College Ave
Swarthmore, PA 19081
paksoy@cs.swarthmore.edu

Abstract

We implement a system for generating large tagged corpora from the World Wide Web. This system leverages the social bookmarking site del.icio.us to generate a topic space using only a handful seed topics and training set for each topic. We evaluate the effectiveness of this system by using a TDT algorithm based on Latent Semantic Analysis with text frequency- inverse document frequency coefficients. Our extremely simple TDT system, performs poorly, yielding maximum f-values of 0.055. Given a topic space of close to 1,000 tags this result is still significantly better than random selection. Our corpus obviously contains useful and relevant information that a more sophisticated TDT system can produce reasonable results with.

1 Introduction

Topic Detection and Tracking (TDT) is crucial to grouping information into categories so it can be dealt with effectively. There currently exists several supervised algorithms for performing TDT on a variety of corpora with moderate success. However, there is no good way to bootstrap these algorithms to generate reasonable size training corpora for an arbitrary size topic space.

Social bookmarking sites allow users to tag webpages and maintain records of these sites in a highly accessible fashion. We leverage a social bookmarking site, del.icio.us, to generate a large tagged corpus of webpages.

We implement an extremely simple supervised TDT system that does Latent Semantic Analysis (LSA) using text frequency-inverse document frequency (tf-idf) coefficients. In order to investigate the usability of our corpus, We train this system and evaluate its performance using our web generated corpus.

2 Previous Work

Dumais et al. (1988) describe the method of using Latent Semantic Analysis (LSA). This method involves creating vectors for each document in the space of all words, and calculating the cosines of these vectors in order to ascertain document similarity.

Schultz and Liberman (1999) use also LSA for TDT. However they use text frequency-inverse document frequency (tf-idf) values for their vector coefficients. They cluster their results using single-linkage clustering and cosine similarity. Their similarity metric proves highly usable. However, their clustering algorithm produces unusably large clusters due to chaining.

There is no previous work that we know of in using social bookmarking sites to generate topic detection training and testing corpora.

3 Methods

Our project first requires that we generate a large tagged corpus. This turns out to be non-trivial because of the unique method of corpus generation we chose for the project; we use the social bookmarking webpage del.icio.us to aid in our corpus generation.

Evaluating the performance of the system also relies heavily on del.icio.us. The tags we generate using tf-idf coefficients are meant to align with the user-defined tags from del.icio.us.

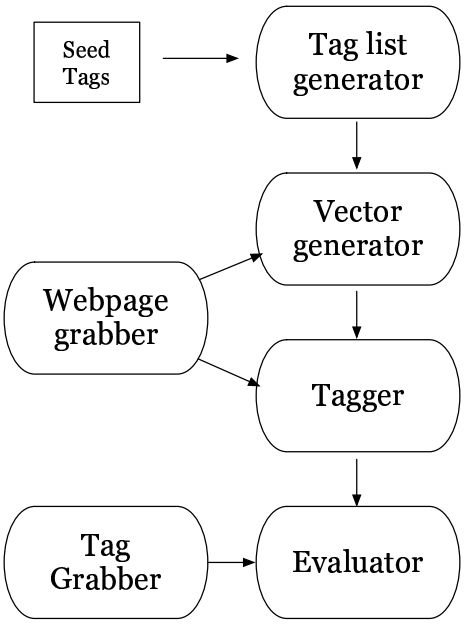

Figure 1: Overall corpus generation system and evaluation system structure. The user provides a list of seed tags. Tag list generator grabs list of all related tags for each tag. A vector is generated for each tag with 90% webpages belonging to that tag. The remaining sites are tagged using cosine similarity. The resulting guessed tags are compared with actual tags that have been assigned to each webpage by `del.icio.us`.

3.1 Generating Tagged Web Corpus

Dynamic corpus generation has the advantage of relevancy. Our method of corpus generation retrieves webpages. This allows for the effective tracking of the given tags; things like popularity and relevancy can be tracked using our method of corpus generation. Compared to archived corpora, our method of corpus generation has the advantage of serving as a "finger on the pulse" on the blogosphere.

3.1.1 Tag List Generation

`del.icio.us` has built-in functionality that allows one to check the RSS feed for a given tag, effectively yielding the most popular webpages. This functionality is accessed with HTTP: `http://del.icio.us/rss/tag/TAGTOCHECK`. The RSS feeds are parsed using the `Universal Feed Parser` (`http://feedparser.org`), an open source Python library for handling XML feeds such as RSS.

The initial step in the process of web corpus generation requires that one select several broad tags. We picked the most popular tags listed on `del.icio.us` and added a few that we thought would be interesting (about 100 tags combined). These broad tags identify the general subject of the corpus to be generated. Using with the broad topic tags, we begin to traverse the sites tagged by `del.icio.us` users and we record the previously unseen tags. This forms a large list of tags (approximately 1,000) seeded by the original broad tags. With the newly constructed large tag list, we again query for sites associated with the given tags to form our site list and obtain them with our web grabber. Given the size of `del.icio.us`'s user-base, this is the largest hand tagged corpus imaginable.

3.1.2 Web Grabbing

The process of web grabbing is non-trivial because of the highly non-standard nature of the HTML on the internet. `del.icio.us` does not limit itself to particular sites, which required that we write a general webpage parser that would output text only. We use a Python package called `BeautifulSoup` as the foundation of our general HTML parser.

After running across countless examples of malformed HTML code that would parse inconsistently,

we began searching for a parser with built-in malformed HTML handling rather than attempting to modify Python's `HTMLParser` library extensively with an overabundance of cases. `BeautifulSoup` has this capability, it is self described as:

1. Beautiful Soup won't choke if you give it bad markup. It yields a parse tree that makes approximately as much sense as your original document. This is usually good enough to collect the data you need and run away.

2. Beautiful Soup provides a few simple methods and Pythonic idioms for navigating, searching, and modifying a parse tree: a toolkit for dissecting a document and extracting what you need. You don't have to create a custom parser for each application.

3. Beautiful Soup automatically converts incoming documents to Unicode and outgoing documents to UTF-8. You don't have to think about encodings, unless the document doesn't specify an encoding and Beautiful Soup can't autodetect one. Then you just have to specify the original encoding. [1]

The "soup" that is returned from each site requires further line-by-line cleaning because of our tendency to grab more information from a given webpage; this is done to ensure that as much text is gleaned from a webpage as possible.

For each tag, a tenth of the webpages are excluded from the training corpus and preserved for testing. The rest are used to generate the tag vectors that make up our topic space.

In the end we downloaded and cleaned more than 19,000 documents. We distributed this task to 20 different hosts in the Swarthmore College Computer Science department network for performance sake.

3.1.3 Evaluation Tags

Once we have finished our tf-idf coefficient alignment process, we must have a method of evaluating the tag associations we have made. The obvious solution to this problem is to use the tags defined by `del.icio.us`'s user base for each particular webpage as an evaluation set. `del.icio.us` has built-in functionality specifically for this task. This is done over HTTP by querying the following address: `http://del.icio.us/rss/url?url=URLTOCHECK`. Again, results are returned in the form of RSS files and the `Universal Feed Parser` is used to obtain tags for each webpage. We include only those tags included in the long list of tags generated initially (because it would impossible for us to tag something with a tag unavailable to us initially) in the evaluation process and generate precision and recall values for each webpage.

[1] The descrption above was taken from `http://www.crummy.com/software/BeautifulSoup/`.

3.2 Topic Detection

Our topic detection system mainly serves to demonstrate the usefulness of our web generated corpus. It is intentionally kept as simple as possible, to shed light on the characteristics of the corpus we generate. The webpage groupings generated by our topic detection system provide significant information for searching, tracking and further categorizing web sites. The process involves first generating a semantic vector for each of the webpage's articles. The vectors are the compared using cosine similarity to test their semantic relatedness. Depending on the method of feature selection used, the comparison vector can be huge, the maximum size being the total number of unique words in the corpora. This alludes the problem of balancing sparse data; using too many feature words lends itself to too many false positives, overmatching and vice-versa. For simplicity's sake, we do not use any feature selection.

3.2.1 TF-IDF and Cosine Similarity

A semantic vector is generated for each document with tf-idf values as coefficients. These vectors have very high dimensionality and are very sparse. We judge the similarity of two documents based on the cosine of the angle between their vectors. Cosine is given as:

$$cos(\theta) = \frac{\vec{u} \cdot \vec{v}}{|\vec{u}||\vec{v}|}$$

The cosine quantifies the size of the angle between two vectors. Naturally, the closer the cosine value is to one, the more semantically related two documents are.

3.2.2 Semantic Vector

Generating the document frequency *df* vector is the first step in the process of generating a semantic vector for each document. The *df* vector is generated by stepping through every document in the corpus and creating a hash of every unique word seen thus far as a key. The contents of the hash is a count for the number of documents which contain the word; the count is incremented the first time a word is seen in the document only.

Every document in the corpus is parsed word-by-word to generate the term frequency *tf* vectors. We normalize our *tf* vectors by document length. This ensures that words are properly weighted within a given document; otherwise, longer documents with inherently larger word counts would have skewed tf-idf values compared to other documents. Dividing *tf* by *df* represents a balance between relevance and prevalence for every word in the document. The words with high tf-idf values are said to have high semantic value in the document. For this reason, the tf-idf vector for a document is reffered to as a semantic vector for the document.

We normalize webpage vectors by dividing each tf value by document length. Initially we were not doing this, however it proved necessary for allowing short documents to be assigned a reasonable number tags. Otherwise, longer documents push the average cosine similarity values up, and make it impossible for shorter documents to receive any tags.

3.2.3 Stop Words

There are particular words that have very little semantic value. Definite and indefinite articles are good examples of words that have no bearing on the relatedness of two webpages. The hope is that words with little semantic value are common enough that their idf would bring the tf-idf to very close to zero. We decided to create a stop word list that would eliminate the need to process those words deemed semantically insignificant. Part of our stop word list is directly acquired from the internet. This is a list of 700 words of very little semantic value for topic detection. [2]

[2]The lists we used can be found at: `http://www.pagex.com/webtools/stopwords.cfm` and `http://www.dcs.gla.ac.uk/idom/ir_resources/linguistic_utils/stop_words`

We also chose not to process words that have a *df* value of one. In general, these words are usually gibberish and they only cloud our data.

We do not shorten our semantic vectors in any other way. In other words, we don't apply feature selection.

3.2.4 Vector Comparison

The aim of our system is to use semantic vectors to align webpages with the tags defined by users of `del.icio.us`. This requires that we construct vectors for each of the tags in our initial long list of tags. We create a list of all webpages grouped by a specific tag and generate a cumulative *tf* vector for all of them; this is the tag's semantic vector.

Individual webpages' vectors are then compared with the vectors from each of the tags using cosine similarity. We use two very simple tag assignment methods: a constant threshold and a tag-variable threshold.

The constant threshold is constant for all tags and documents. If the cosine similarity between any webpage and a tag exceeds or equals the threshold, the website is assigned the tag.

The tag-variable threshold changes for each tag. If the cosine similarity of a webpage to tag exceeds the threshold for that specific tag, the website is assigned that tag. Obviously, this second method is more versatile and allows us to normalize for the varying behavior of different topic vectors.

3.2.5 Evaluation

We evaluate our performance relative to the `del.icio.us` users' tagging preferences. The `del.icio.us` user base seems to be relatively tech-savvy, as evidenced by the technical orientation of many of the tags seen. We place confidence in the fact that most technically-oriented people do not frivolously tag webpages. We use standard information metrics for evaluating our performance, namely precision, recall and f-value. Precision is defined as the number correct tags over the total number of tags guessed for a webpage. Recall is defined as the number of correct tags over the number of the total number of `del.icio.us` users' tags.

4 Results and Discussion

Our results are given in table 1.

Threshold Type	Recall	Precision	f-value
Constant	0.069	0.042	0.052
Variable	**0.072**	**0.045**	**0.055**

Table 1: Best f-values produced by variable and constant threshold tagging systems. Threshold is set to three times the average cosine value.

We have selected our optimal thresholds by scanning through multiples of the average cosine values (Figure 2). In the case where we use constant thresholds, this is the average cosine of all webpages and tags, which is approximately 3.5×10^{-4}. For the variable threshold algorithm a multiple of the average cosine value for each tag is used, these values range between 1.1×10^{-3} and 2.0×10^{-4}.

Overall, the TDT system performs poorly. Our best f-values do not exceed 0.05. Still, we do achieve recall values approaching 40%, which is remarkable given simplicity of our TDT system. Given a topic space approaching 1,000 tags, this performance is significantly better than random selection. This shows that our corpus highly usable.

Initially we intended to score tags differently based on how many users assign them. This would prevent our results from being adversely affected just because a single user picks a poor tag. The strength of social bookmarking sites comes from the sheer abundance of cross-validation. Unfortunately, `del.icio.us` does not provide information about what tags are more prevalent in its RSS tags for URLs. We believe outlier tags that we cannot, and should not, assign to websites are greatly hurting our f-values.

Looking at examples in Figure 3 we can see that tags with overlapping subject matters are easily confused. We believe this is greatly exacerbated by the fact that we do not do feature selection. Feature selection would reduce the breadth of each tag, preventing this behavior.

We also noticed that webpages that have less text, get assigned more tags by the guesser. This is because we normalize tf values by dividing by document length. As a result, most words in short websites get high tf values, thus appear to contain relevant keywords to many tags.

5 Conclusion

At a glance, the performance of our TDT system is unimpressive. However, it should be noted that this system is just inadequate given the size and heterogeneity of our corpus. There is no doubt a topic detection system better equipped handle noise and take advantage of the size of the corpus would perform much better.

With this in mind, our corpus appears very promising. With minimal human input it creates training and testing corpora in a remarkably large topic space.

6 Future Work

This paper mainly focuses on generating training corpora arbitrary size topic spaces using social bookmarking webpages. As such, our TDT system is exceedingly simple. One simple improvement to our TDT system is to add feature selection. Due to the size of our topic space, we have several tags that have very similar vectors. Feature selection would help amplify these differences.

Combining equivalent tags would also help. For example, our tag list contains separate tags christian and christianity; it would probably be a good idea to combine these two tags. This would orthogonalize our topic space and yield fewer yet more meaningful tags for each document. Megan Schuster and Anne Marie Frassica use a semantic graph to conflate words with similar semantic content, this method should also be usable in our case.

As previously states, we can improve our evaluation method by allowing tags to have differing values based on their popularity. This way we would not be penalizing our TDT system for not assigning superfluous tags.

Finally, an approach based on self-organizing maps, or neural networks could also be attempted. These methods are highly suitable for the kind of corpus that we generate because they require a lot of training data and handle noise extremely well.

References

Susan T. Dumais, George W. Furnas, Thomas K. Landauer, Scott Deerwester, and Richard Harshman. 1988. Using latent semantic analysis to improve access to textual information. In *Proceedings of the*

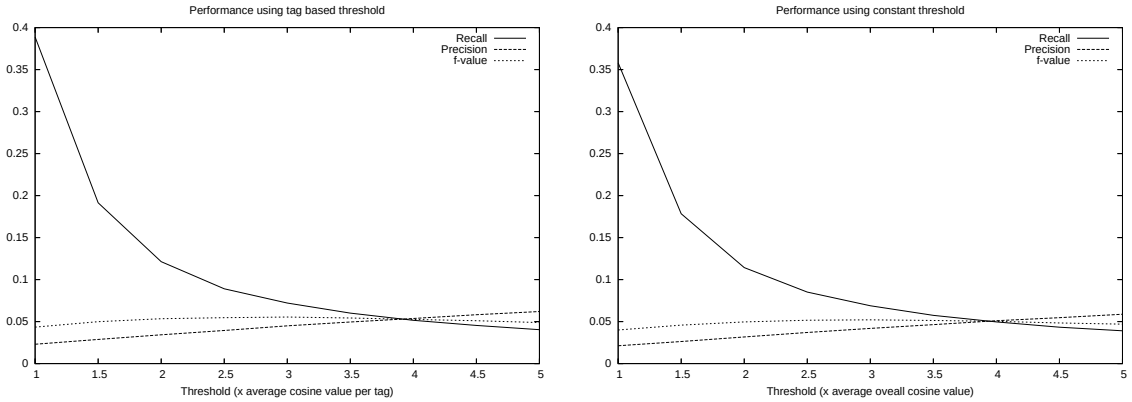

Figure 2: Threshold selection while using constant and variable thresholds. In both cases, optimal f-values are obtained when the threshold equals 3 average cosine.

- **USA Today article on the Iranian Holocaust denial conference**
  international christian iraq pseudoscience israel beautiful links united law hanukkah articles rhetoric typography war bush mysticism artist europe unitedstates cartoon world walks fabric islam us www.directories slang

- **Guide to performing the Muslim ritual prayer**
  ala egyptian tagging christian tags bloggers bornagain study fundamentalism interview philosophy catalog president hanukkah help grammar cartoons team scary microfinance dawkins typography etext yahoo! photo mysticism mythology buddhism cartoon parody religion christianity islam heat atheism

Figure 3: Example guessed tags.

Conference on Human Factors in Computing Systems CHI'88.

J. Schultz and M. Liberman. 1999. Topic detection and tracking using idfweighted cosine coefficient.