Political Blog Analysis Using Bootstrapping Techniques

Fritz Heckel, Nick Ward
Department of Computer Science
Swarthmore College
Swarthmore, Pennsylvania, USA
{fwph,nward}@sccs.swarthmore.edu

Abstract

In the past few years, the form of Internet media known as "blogging" or weblogging has exploded, especially in the realm of politics. We propose and implement a system for performing qualitative text analysis of political blogs, with the ultimate goal of placing them on a map to categorize them according to political bias. Our system performed surprisingly well on the task of categorizing entire blogs, though the success is not entirely unqualified, and the system is not suitable for categorizing individual articles.

1 Introduction

Political bloggers are some of the most prolific writers of today's new media, generating thousands, if not millions, of articles a day. We can harness the sheer mass of blog articles to create a large and very dense corpus for use with machine learning techniques. Furthermore, as blogs are generally quite easy for a human to classify by political bent, it is effectively a self-labeled corpus.

Because of the sheer size of "the Blogosphere", it can be extremely difficult to navigate; information overload takes on a new meaning after spending several hours traversing the blog nets. We propose a system to help ease this task, by performing qualitative text analysis on blog articles. We hope to develop a system that not only categorizes blogs discretely, but also places them along a spectrum so that it is easy to compare different blogs with a glance.

We do not make any hypotheses as to the nature of the final blog classification results. We seek only some sort of document classification that reveals some interesting patterns in blogs. Whether those patterns manifest themselves as political affiliation, authorship, or some more subtle content-based qualifier, they will give some meta-information about a given blog.

2 Related Work

There has been effectively no work in the area of automatically classifying blogs based on their content. Most proposals focus on creating a pre-defined taxonomy of blog subject matter that would be integrated directly into RSS stream files as metadata; the current incarnation of such a system is a <taxo> XML tag that can be included in a blog's RSS feed (Beged-Dov et al., 2000). Note that this system requires manual entry of blog metadata by each blog author for every document that they create.

One example of a set of classifications that could be used with a taxonomy-based system has been proposed for blogs produced not by individuals but by corporations, businesses, or other organizations (Wackå, 2004). The author suggests that the primary division in this blog subdomain is between "External Blogs", which are used by the company or organization to promote their image, and "Internal Blogs", which are used by employees to collaborate and disseminate knowledge and company culture. The author mentions that any sort of top-down classification proposed for blogs, even within a constrained subdomain of "the Blogosphere", is doomed to failure simply because no one will agree

Appeared in: *Proceedings of the Class of 2005 Senior Conference*, pages 20–27
Computer Science Department, Swarthmore College

on what the fixed taxonomical classification standard will be. This is why automated classification systems are necessary.

Other extensions to existing blog metadata have been proposed, such as one to add Semantic Web-compatible content classifiers to existing RSS feeds (Karger and Quan, 2004). None of these methods appear to address the fact that every blogger would be required to conform to some sort of metadata standard in order to make their entries classifiable, nor that some bloggers might choose to intentionally mislabel their entries for some reason.

3 System Architecture

Our system is composed of three major parts: a data harvester, a training system for discovering domain-specific lexicons, and a categorization component. The first component uses Perl's XML::RSS and LWP::RobotUA modules to create a simple RSS aggregator which feeds entries into a MySQL database. The use of MySQL lets us continuously harvest blog articles from RSS feeds while simultaneously training our system, avoiding problems of file-locking and other race conditions, while the Perl modules give us pre-existing code to help us avoid reinventing the wheel.

The second component is based on the BASILISK system (Riloff and Thelen, 2002) created by Ellen Riloff. BASILISK first utilizes the AutoSlog (Riloff, 1996) system to generate extraction patterns from the training data, then takes a seed lexicon to discover a larger dictionary of words relating to a number of categories in the political domain. We re-used the top extraction patterns as templates to match in new documents; these combined with the words matching in the pattern will be used to create feature vectors for the categorization step.

The third component is based on a part of the SOMLib Digital Library system (Merkl and Rauber, 2000) created by the Department of Software Technology team at the Vienna University of Technology. Their unsupervised document classifier consists of a hierarchical feature map (HFM), a tree-like arrangement of several independent self-organizing maps (SOMs). The feature vectors derived from the second component will be used as input to these HFMs, which will be implemented using an existing SOM software module in the Python Robotics Project (Blank et al., 2002) (Blank et al., 2005). Developing in this pre-defined Python environment, with which both of us are experienced, sped the development of this component.

4 RSS Aggregation

We chose as sources for training data a number of high profile, high volume blogs for which we knew the political slant. The full list is shown in Table 1.

Table 1: Political Weblogs

Political Weblogs	
Category	Blog
Conservative	GOP Bloggers
	The Museum of Left Wing Lunacy
	Secure Liberty
	The Blue State
Liberal	Blog vs. Blog
	Pandagon
	Eschaton
	TalkLeft
	Kicking Ass

Most of these blogs are fairly high volume, and each one is clearly partisan. Over the course of 2 1/2 weeks, 962 blog articles were aggregated, totaling around 25,000 words. This is not a very large training corpus, but the unique nature of blogs would likely cause a larger corpus aggregated over a much longer time period to be far less useful: issues in blog-space may change rapidly, requiring retraining of the feature map on a regular basis.

5 Feature Training

We tried two different approaches to building features:

- Caseframe features (Sec. 5.1)
- Lexical features (Sec. 5.2)

We expected lexical features to be far more successful, as caseframes would tend to be more general,

Appeared in: *Proceedings of the Class of 2005 Senior Conference*, pages 20–27
Computer Science Department, Swarthmore College

and not necessarily even strongly related to the domain (ie, *he said*). Lexical features, on the other hand, would be words which tended to show up frequently throughout the corpus.

AutoSlog builds caseframes based on a number of simple heuristic patterns such as <**subj**> **passive-verb**, which will generate caseframes like <**person**> **gave**. When using AutoSlog-TS, these extraction patterns are generated for each noun phrase in each set of texts. Based on the number of times a pattern is found in the relevant texts and not in the irrelevant texts, each pattern is assigned frequency values. These can then be used to provide a probability that a caseframe is found in the text of an interesting domain. This serves as a score, and a number of patterns can be chosen based on their high scores. AutoSlog extracts patterns for each word specified in a target dictionary, rather than for every word in the corpus as AutoSlog-TS does. Frequencies are not calculated for these caseframes; instead they are used for extracting additional words, and the words are scored based on frequency.

5.1 Caseframe Features

To build caseframe features, or *extraction patterns*, we used AutoSlog-TS. AutoSlog-TS is capable of generating extraction patterns from text with no supervision. Our text corpora were composed of about 900 entries from the blogs mentioned above, totaling about 25,000 words, and an unrelated text composed of samples from the Corpus of Professional Spoken American English[1]. The samples from CPSA totaled about 75,000 words; ideally, we would have used a more balanced corpus.

Caseframes are extraction patterns: generally, they are composed of a verb phrase (or partial verb phrase) with one or more slots for an associated noun phrase. From these corpora, AutoSlog generated 855 caseframes which we used as individual buckets in a feature vector. By calling Sundance on each blog entry, we were able to find the number of times each extraction pattern occurred in the text; these values were used to build the actual feature vector which could then be fed into the SOM for training.

[1] http://www.athel.com/cpsa.html

5.2 Lexical Features

To build a lexicon for the domain, we started with a number of seed noun phrases in several categories relating to politics. Table 2 shows examples of categories and seed noun phrases for each. Ultimately, we found just seven seed noun phrases to be sufficient: *Bush, President, security, terrorism, terror, Congress*, and *Senate*.

Building the full lexicon then followed a simplified variant on the BASILISK method(Riloff and Thelen, 2002).

- Run the training corpus through AutoSlog, using the seed lexicon to generate extraction patterns
- Use Sundance with the new extraction patterns to discover additional lexicon candidates
- Choose some number of candidate noun phrases to add to the lexicon
- Remove common noun phrases from the list (such as he, she, this, etc.)
- Repeat until the lexicon size stabilizes.

We simplified the method by placing a threshold on the number of occurrences necessary for noun phrases to be added to the lexicon, rather than using a full probabilistic method. Using a threshold of five occurrences, we found that after six iterations of the algorithm, the lexicon had stabilized at 254 noun phrases for our training set. Some of the top noun phrases were, unsurprisingly, *Congress, Senate, and George Bush*. Other, more loaded in context, were *Pro-Choice President* or *Pro-Life President*.

Once again, the feature vector was created by counting the number of times the features– this time, the members of the lexicon– occurred in each blog entry. This vector's buckets represented number of occurrences of words instead of extraction patterns, and the feature vectors were fed into the map as with the extraction pattern features.

6 Categorization

6.1 Self-Organizing Maps

A self-organizing map (SOM) consists of a two-dimensional grid of nodes, each of which is initialized with a random vector in the feature space.

Appeared in: *Proceedings of the Class of 2005 Senior Conference*, pages 20–27
Computer Science Department, Swarthmore College

Table 2: Categories and Seed Words

Categories and Seed words	
Category	Seeds
Issues	Social Security
	Iraq War
	Election Reform
	Gay Rights
Parties	Democrats
	Republicans
	GOP
Figures	George W. Bush
	John Kerry
	Paul Wolfowitz

The training set for a single SOM consists of a large number of vectors distributed throughout the feature space; therefore the SOM will effectively "learn" a simple clustering of that space. A conceptual depiction of a newly initialized 4x4 SOM can be seen in Figure 1. Note how the mapping of each SOM unit into the feature space is arbitrary.

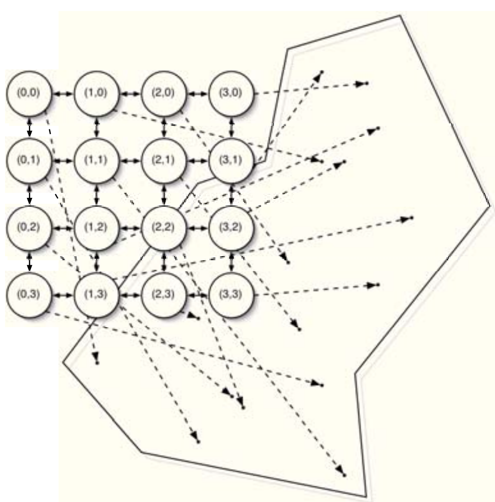

Figure 1: The units of a self-organizing map (SOM) in the standard two-dimensional grid configuration and their initial mapping into a feature space.

As each vector from the training set is input into the SOM, the unit whose state is closest to the input in the feature space by Euclidean distance "wins". The unit, and with some fall-off its neighbors, has its state adjusted to be closer to the input vector. After training, the nodes or units in the SOM will have clustered the feature space. The neighborhood function h_{ci} is given in Equation 1, where $\|r_c - r_i\|$ is the distance in feature space between two units' vectors, and $\sigma(t)$ is the width of the Gaussian. $\|r_c - r_i\|$ is merely a representation of whatever arbitrary distance metric is selected for a particular SOM implementation; no vector subtraction necessarily occurs. In our system, we chose to use standard Euclidean distance in the feature space, since we could guarantee that all of our input vectors would be the same length. $\sigma(t)$ decreases with time, so after many training iterations only the winning unit's position in feature space is updated (Merkl and Rauber, 2000).

$$h_{ci}(t) = e^{-\frac{\|r_c - r_i\|^2}{2\sigma^2(t)}} \tag{1}$$

By adjusting not just the best-match unit but also its neighbors, and by decreasing the influence on neighbors over time, the SOM units tend to detect clusters in the feature space. Hopefully, the end positions of the unit vectors subdivide that space in an interesting way.

The output of each unit in an SOM is associated with at most one cluster, although multiple units may represent a single informative cluster. The interpretation of the result of the training is entirely up to the user; the SOM can only give a clustering in terms of the feature space, as it does not "know" anything about the problem domain. In our case, feature vectors represent individual blog entries, so the clusters are interpreted as representing some abstract grouping of entries.

6.2 Hierarchical Feature Maps

The number of clusters is highly dependent on the architecture of an individual SOM. It can only cluster the feature space into at most as many clusters as it has units. This is where a hierarchical feature map (HFM) becomes useful: by using one SOM to divide the feature space into smaller sub-problems, each of those sub-problems can in turn be clustered by an SOM. The result is a highly specific clustering of the feature space.

Appeared in: *Proceedings of the Class of 2005 Senior Conference*, pages 20–27
Computer Science Department, Swarthmore College

A typical HFM consists of a tree structure in several layers, as shown in Figure 2. The topmost layer of the HFM, and the root of the tree, consists of a single SOM. The individual units of an SOM at each layer pass feature vectors onto an entire SOM in the next layer down, all the way down to the base of the HFM.

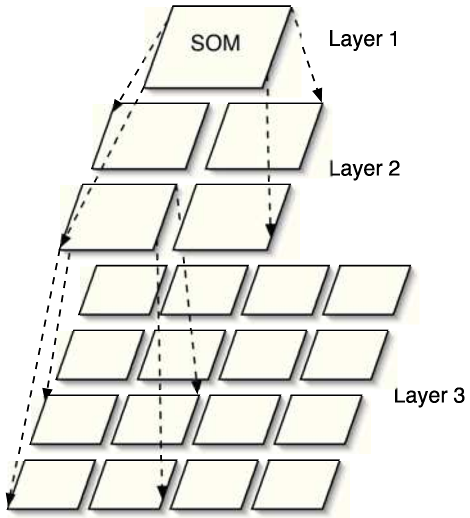

Figure 2: A hierarchical feature map (HFM) consists of multiple SOMs in a tree-like structure

To speed training and improve the accuracy of the results, the dimensionality of the feature space can be reduced between each layer on a unit-by-unit basis. If all of the input vectors that match have similar values along one feature space dimension, that dimension can be eliminated before passing the training subset onto the next SOM layer.

Note that it is possible for the dimensionality of the feature subspace being handled by different SOMs on the same HFM layer to vary. Some SOM units higher up in the HFM tree may be trained to make big clustering decisions that significantly reduce the dimensionality, while others may make no change to the dimensionality and pass vectors directly to their child SOMs.

Once the HFM has been trained, using it is simply a matter of inputting a feature vector. At each layer, the "winning" unit of the SOM will pass the vector down to its child SOM, until the bottom of the HFM tree is reached. The bottom-most units of an HFM are each interpreted as having some cluster-related meaning, based on the feature vectors.

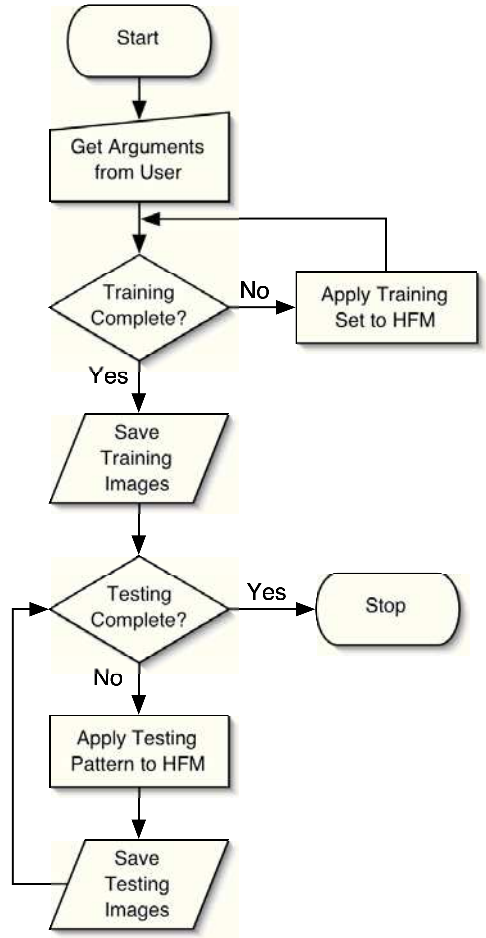

Figure 3: The training and testing process for an HFM

The complete process for training and testing a single HFM can take many iterations, depending on the size of the training and testing feature vector sets. The flowchart in Figure 3 demonstrates this procedurally.

6.3 Implementation

The blog aggregating software was written in Perl, using a MySQL database for storing information. Our spider used the XML::RSS and LWP::RobotUA Perl modules to fetch and parse blog RSS feeds. Some slight modifications to portions of the Sundance package were necessary to fix out of date code, and the process of running the two feature training algorithms was automated with a number of

Appeared in: *Proceedings of the Class of 2005 Senior Conference*, pages 20–27
Computer Science Department, Swarthmore College

Bash and Perl scripts, while feature vector information was generated using Perl.

Our implementation of HFMs was written entirely in the Python programming language. We chose to use Python so that we could use the existing SOM implementation written by Daniel Sproul '03 that is included as part of PyRo, the Python Robotics package (Blank et al., 2002).

The main functional unit of the system is the HFMNode class, which simply contains an SOM instance and a 2-D list of child HFMNodes. Because of the way in which the outputs of the SOM units at each HFM layer are passed on to the next layer, as described above in Section 6.2, the SOM and the list of child nodes must be the same size.

The HFMNodes are contained with the HFM class, which is merely a convenience class that holds a single HFMNode as the root of the HFM's tree. The HFM class also contains file I/O functionality, for reading input feature vectors generated during the lexical training steps described in Section 5.2.

6.4 Visualization

In order to examine the training and testing process, we needed some intuitive way of displaying the output of an HFM. Pyro's SOM implementation did have some visualization capability, but it was for live observation only, and suffered from data overload. In addition, our HFM implementation abstracted away from the SOM class to a large extent, so it was necessary to develop our own visualizer.

Each layer of a given HFM training or testing run can be output as a grid. This allows us to observe both the final categorization and the initial clustering decisions made by the SOMs in the lower-resolution layers of the HFM. Each grid cell in the output image represents one of the SOMs in that layer. The points drawn within cells are color-coded by blog, allowing us to distinguish the output. Each point represents a single tested or trained feature vector.

7 Results

To obtain our final results, we trained a 3-layer HFM consisting of 2x2 SOMs. The HFM was trained for 150 iterations using the entire corpus of blog entries from the nine blogs listed in Table 1. We then examined the testing results for each of the blogs individually.

The results from the caseframe features were unremarkable at best– the caseframe features did not extract a sufficient amount of information to cluster the blog entries in any manner. It is not necessary to cover that method any further, so the remainder of this section refers to our results using lexical features.

7.1 Training

Figures 4 and 5 contain examples of the output from the deepest layer of the HFM after training on the entire blog entry corpus. The primary feature of note is that the vast majority of the entries clustered along one of the diagonal axes of the HFM. The diagonal axis is inconsistent between training runs because the SOMs in the HFM have their initial positions in the feature space set randomly at runtime.

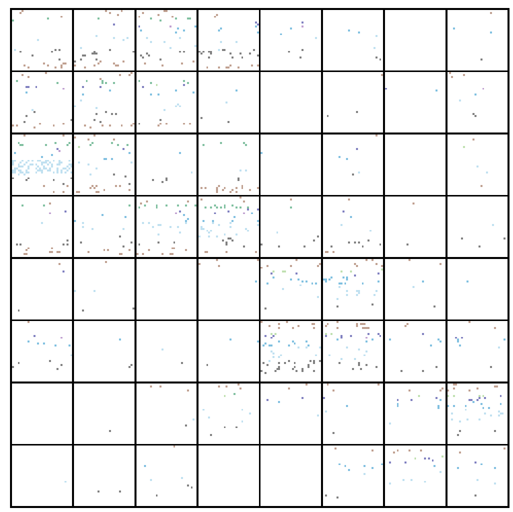

Figure 4: HFM training output for Run 1

The distributions of the training set are very similar between the two separate training runs, which implies that our HFM is probably learning the same distinguishing features. Note for example the strong cluster from a single blog that appears in cell (0, 2) of Figure 4 and in cell (7, 3) of Figure 5 as a denser stripe of points. This cluster from a single blog was assigned its own bucket between runs. The fact that we have consistent training of a randomly-initialized neural network structure is a very good sign that our testing results have some non-trivial meaning.

We wish to reiterate the somewhat black-box na-

Appeared in: *Proceedings of the Class of 2005 Senior Conference*, pages 20–27
Computer Science Department, Swarthmore College

ture of network-based learning methods. These results are very much open to interpretation, although we believe that the patterns that can be seen in our results are not just mere chance.

7.2 Testing

The distribution of blogs shown in Table 3 is associated with the second training run shown in Figure 4. With the exception of Pandagon and The Museum of Left Wing Lunacy, the lower-left contains conservative blogs and the upper-right contains liberal blogs. It should be noted that this table reflects the primary concentration of each blog's testing output. Instapundit and Talking Points Memo are two blogs that were exclusively in our testing data set.

If we examine the results more closely, it becomes clear that our HFM did succeed in performing the basic liberal/conservative classification task. Pandagon is a particularly unique blog, and had the most diffuse results from the HFM. It is only barely concentrated with the conservative blogs, even though it is a liberal blog. There is not a strong explanation for why Pandagon emerged differently, though it is worth noting that the tone and nature of the articles on Pandagon are rather unique.

The Museum of Left Wing Lunacy, as an unashamedly conservative blog, was classified with the liberal blogs. However, a quick examination of their entries shows that they primarily quote other blogs, and mostly liberal blogs at that. This means

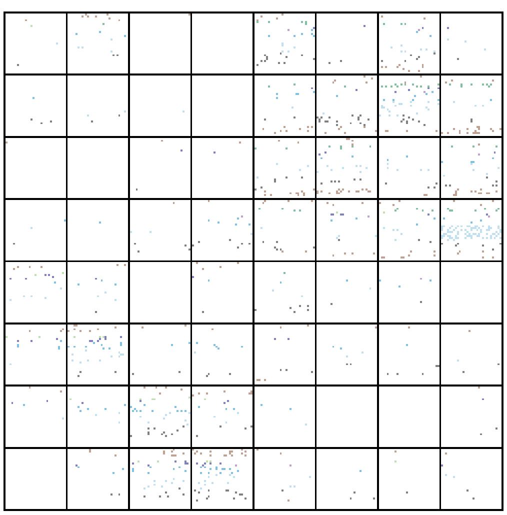

Figure 5: HFM training output for Run 2

Table 3: HFM output

HFM Output	
	Left Wing Lunacy (C) Blog vs. Blog (L) Eschaton (L) TalkLeft (L) Kicking Ass (L) Talking Points Memo
GOP Bloggers (C) Secure Liberty (C) Blue State Conservatives (C) Pandagon (L) Instapundit	

that the plain text version which we analyzed contained mostly liberal-leaning text. This result might be avoided if we removed all quotations from the input entries.

8 Future Work

Our implementation of BASILISK was somewhat less sophisticated than the original method, as we took a simpler approach to choosing words to be placed in the lexicon. Because many blog articles are very short– though may still contain a great deal of information– it seemed more important to avoid creating an overly small lexicon than one that was too large.

The original design of this project would have used both lexicon and extraction pattern data to generate the features, but we had difficulty in finding a feature representation which could concisely represent all of this data in a form which could provide useful results from the SOM. Feature maps seem to have served well in this task, though we do believe that our method could be refined significantly. We are not entirely satisfied with the feature vectors as they stand.

In addition, before calling this an unqualified success, further testing with a larger corpus must be performed.

Appeared in: *Proceedings of the Class of 2005 Senior Conference*, pages 20–27
Computer Science Department, Swarthmore College

9 Conclusion

The combination of BASILISK and Self-Organizing Maps worked surprisingly well for this project. Given the ultimate sparsity of our feature vectors, we did not expect to achieve the level of performance that we did. Further exploration of this combination would certainly be worthwhile in the future.

References

Beged-Dov, G., Brickley, D., Dornfest, R., Davis, I., Dodds, L., Eisenzopf, J., Galbraith, D., Guha, R.V., MacLeod, K., Miller, E., Swartz, A., and van der Vlist, E. (2000) "RSS 1.0 Modules: Taxonomy", RDF Site Summary 1.0 Modules, 20 Mar 2001, RSS-DEV Working Group, 01 Apr 2005, <http://web.resource.org/rss/1.0/modules/taxonomy/>.

Blank, D.S., Kumar, D., and Meeden, L. (2002) "Python robotics: An Environment for Exploring Robotics Beyond LEGOs", ACM Special Interest Group: Computer Science Education Conference, Reno, NV (SIGCSE 2003).

Blank, D.S., Kumar, D., Meeden, L. and Yanco, H. (2005) "Pyro: A Python-based Versatile Programming Environment for Teaching Robotics". To appear in the ACM Journal on Educational Resources in Computing (JERIC).

Karger, D. and Quan, D. (2004) "What Would It Mean to Blog on the Semantic Web?", International Semantic Web Conference 2004.

Merkl, D. and Rauber, A. (2000) "Document Classification with Unsupervised Neural Networks", Soft Computing in Information Retrieval, 2000, pp. 102-121.

Thelen, M. and Riloff, E. (2002) "A Bootstrapping Method for Learning Semantic Lexicons using Extraction Pattern Contexts", Proceedings of the 2002 Conference on Empirical Methods in Natural Language Processing (EMNLP 2002).

Riloff, E. (1996) "Automatically Generating Extraction Patterns from Untagged Text", Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI-96) , 1996, pp. 1044-1049.

Wackå, Fredrik, "Six Types Of Business Blogs - A Classification" [Weblog entry], CorporateBloggingBlog, 10 Aug 2004, <http://www.corporateblogging.info/2004/08/six-types-of-business-blogs.asp>, 01 Apr 2005.

Kicking Ass, http://www.democrats.org/blog/

TalkLeft, http://www.talkleft.com

Eschaton, http://atrios.blogspot.com/

Pandagon, http://www.pandagon.net/

Blog vs. Blog, http://blog.battletothedeath.net

The Blue State Conservatives, http://www.radiobs.net/thebluestateconservatives/

Secure Liberty, http://secureliberty.org/

The Museum of Left Wing Lunacy, http://www.museumofleftwinglunacy.com

GOP Bloggers, http://www.gopbloggers.org/

Appeared in: *Proceedings of the Class of 2005 Senior Conference*, pages 20–27
Computer Science Department, Swarthmore College