To appear in the Tenth International Conference on Machine Learning and Applications, 2011

Improving Classifier Performance by Autonomously Collecting Background Knowledge from the Web

Steven N. Minton
InferLink Corp
El Segundo, CA

Matthew Michelson
Fetch Technologies
El Segundo, CA

Kane See
InferLink Corp
El Segundo, CA

Sofus Macskassy
Fetch Technologies
El Segundo, CA

Bora C. Gazen
Google
Mountain View, CA

Lise Getoor
Univ. Maryland
College Park, MD

Abstract— **Many websites allow users to tag data items to make them easier to find. In this paper we consider the problem of classifying tagged data according to user-specified interests. We present an approach for aggregating background knowledge from the Web to improve the performance of a classier. In previous work, researchers have developed technology for extracting knowledge, in the form of relational tables, from semi-structured websites. In this paper we integrate this extraction technology with generic machine learning algorithms, showing that knowledge extracted from the Web can significantly benefit the learning process. Specifically, the knowledge can lead to better generalizations, reduce the number of samples required for supervised learning, and eliminate the need to retrain the system when the environment changes. We validate the approach with an application that classifies tagged Fickr data.**

Keywords: Information Extraction, Web Harvesting, Ontologies, Classifiers, Background Knowledge

I. INTRODUCTION

A common way that websites organize crowd-sourced data is by asking users to *tag* the data. A tag is a keyword or "folksonomic term" assigned as metadata to an information object, such as a picture on Flickr, a video on YouTube, or a document on DocStoc. In this paper, we describe an application that *monitors* sites with tagged data items, such as pictures on Flickr, to identify items that match user-specified interests, such as pictures of NBA basketball games. While this classification problem could be addressed using standard information retrieval techniques, we can take a more interesting approach here because we are monitoring data sources over time, as opposed to searching in real-time

In particular, we describe a system that can, in response to a particular classification task, extract domain knowledge from the Web and autonomously "educate itself" to improve its performance. The approach is interesting in part because we use a very general, unsupervised extraction system that can capture relational data from semi-structured web sites. This relational data can be expressed in the form of standard first-order "domain theories" and which can be directly utilized by generic machine-learning classifiers.

In contrast to many other systems that learn from the Web, our system starts with a (domain-independent) classification task, rather having the goal of simply building an ontology. It is distinguished from other task-oriented systems in that it employs generic extraction, representation and learning methods (rather than methods designed for the specific task).

II. THE APPLICATION

Our research was motivated by a "Web Intelligence" portal builder that we are developing for situation awareness in niche domains. This application allows a domain expert to integrate and monitor Web data from heterogeneous sources. The collected data can then be displayed in a "vertical portal", which end-users can easily browse on a regular basis to find out what's happening in that domain. For an example, we built a portal that tracks wildfires in the U.S., aggregating statistics on each fire from the U.S. Forest Service, news stories from online newspapers, fire warnings from state agencies, pictures of wildfires from Flickr, videos from YouTube, etc. The information is integrated so that users can easily see what's new in a region or find out about a particular fire.

The portal infrastructure is domain-independent and applicable for a wide-variety of domains, including musical groups, political events and sports. Our challenge is enabling a domain expert to aggregate information about a particular domain, without requiring any programming.

In particular, one type of valuable data is tagged data from sites like Flickr, YouTube, Del.i.cious, etc. In these sites, users collaboratively annotate data with informal tags to help categorize the data. This makes it possible, in theory, to search a site such as Flickr for pictures of wildfires for the wildfire portal, or pictures of professional basketball games for an NBA portal, and so forth. Unfortunately, however, entering the search term "wildfire" into Flickr returns a wide variety of photos, only a minority of which are relevant, because the tags are informal, and each tag may have a plethora of "meanings". For instance, on Flickr the tag "wildfire" is associated with photos of roses (the "Wildfire" variety), girls with red hair, sunsets, horses, etc., along with the wildfire incident photos relevant to this portal.

To filter out irrelevant data from tagged sources, the portal employs a classifier which can be trained by a domain expert. The expert first provides a few search keywords such as "wildfire" and "fire" (or "basketball", "NBA", etc.) which system uses to disjunctively query a tagged data source such as Flickr, YouTube, Technorati, etc.[1] We refer to the set of key

[1] To query the source, we can either rely on an API provided by the source, such as Flickr's API, or use a web agent to harvest data from the source. In

words used to query the source as Q, and the union of the tagged data objects returned by the source in response to Q is $\Theta = \{O_1, O_2, \ldots O_n\}$. Each of these objects O_x is associated with a set of tags $\{\tau_1^x, \tau_2^x, \ldots \tau_k^x\}$.

The expert then labels a random sample of the returned data Θ as positive/negative, and the system induces a classifier to identify instances of the target concept based on their tags. The classifier not only is useful for automatically filtering the data set Θ but can be re-applied when the data source is re-queried later, i.e., using the same query set Q to generate a new set of objects $\Theta^{\square}$.

This approach eliminates the need to manually filter the data. Even so, it suffers from two well-known problems with machine learning. First, to induce a good classifier, the expert may be required to train the system on many examples, which can be tedious. Second, the classification rules may grow stale over time since the distribution of tags associated with the target concept may change.

III. WELLGROUND

In order to address these short-comings, we developed the WebGround system, which is designed to collect background knowledge from the Web to aid classification. As we will show, background knowledge can significantly improve the accuracy of classification and reduce the number of training examples required, saving experts considerable time and effort. Consider, for instance, the task of identifying tagged photos of NBA games. For this task, knowing the names of NBA players, teams, and arenas can be helpful. In particular, this knowledge is particularly useful for identifying the less popular players, since their names rarely occur in the training set.

Background knowledge can also be useful if the target concept changes over time. For instance, new NBA players are drafted each year. Rather than retraining the system, one can simply monitor the appropriate knowledge sources and update the background knowledge appropriately.

The WebGround system, in response to a classification problem, "educates itself" by aggregating background knowledge from the Web. WebGround's process begins by querying the source with the search keywords in Q to return tagged data that the expert can label, as described earlier. However, to reduce the number of examples required to achieve a given level of performance, the system also searches for and extracts relational data on the Web that mention the tags. This relational data is potentially useful as background knowledge for classification. In the next sections we describe the process of extracting potentially relevant data from Web sites, and how the data is encoded to augment standard classifiers.

IV. SITE SELECTION AND DATA EXTRACTION

WebGround identifies sites with potentially useful domain knowledge by employing commercial search engines, such as Google, to return a list of URLs based on the user-provided search keyword set, Q. This simple approach has tended to work well in our experiments, because the search keywords tend to be general, such as "NBA" or "wildfire", and Google is proficient at returning a mix of sites that are rich with data. Our approach does not require all the sites to be relevant; as long as relevant data is found in at least one site, the approach can be useful.

WebGround analyzes each site and extracts relations (i.e., tables of data). Our extraction approach is based on previous work by Gazen & Minton (2005), who developed an unsupervised learning system that automatically extracts semi-structured (and structured) data from a website. Semi-structured data consists of data where the formatting of the data can be described by a (reasonably simple) formal grammar; however, that grammar must be induced (rather than being explicit, as in HTML tables explicitly specified by a <table> tag). For instance, much of the data on Amazon.com is semi-structured, such as the product titles, prices, feature lists, etc, all of which are formatted in a regular fashion throughout the site.

Before explaining the extraction process, let us consider how websites are built. Consider the schematic for a (extremely simplified) weather website "Forecast.com" shown in Figure 1. From the homepage the user can choose a U.S. state, each of which is associated with a URL. On each state page there is a list of cities and URLs. At the next level, each city page includes the weather outlook and a high and low temperature. This information can be described by three database tables. Note that there are three types of pages, each type being similarly formatted and containing the same data fields.

The *site extraction* problem is to extract all the data from the site, essentially to reconstruct the database tables. Our approach works as follows. First, starting from the URL provided by the search engine, the system spiders the site (to a specified depth). Next, a set of "expert" modules analyze the pages. Each expert is an algorithm that makes similarity judgments about the pages, focusing on a particular type of structure. For example, we have experts that identify pages with similar HTML sections, experts that identify pages with similar visual layout components, experts that identify pages with similar semantic elements, and so forth. Based on the similarity judgments the system clusters the pages, following the approach of Gazen and Minton. The goal is to cluster the pages into *page types*, so that each cluster contains similarly formatted pages.

After the pages are clustered, the system identifies which strings on the pages represent "data". Essentially, this involves inducing a grammar that describes the organization of the data on the page. Previous authors have developed a variety of methods for this (e.g.., Crescenzi, 2001). Once the pages have been clustered into page types, WebGround searches for template components that are common to all the pages in the cluster, as discussed by Lerman et al. (2003). To make the process efficient, WebGround relies on a restricted class of grammars based on the Embedded Catalog formalism (Muslea et al., 2001). Specifically, a page type must consist of a sequence of fields separated by template components. Each field is either an *atomic field*, containing one data item, such as

either case, it is straightforward to regularly retrieve data from an internet source that matches at least one of our keywords.

the city's high temperature, or a *list field* containing of a repeated sequence of atomic fields $[a_1, a_2,...a_n]^*$ separated by template components, such as a list of US states and their URLs.

For each page type, WebGround constructs a set of relational tables. All the non-list fields are included in one table, and each list field results in its own table. Thus each atomic field corresponds to a distinct column in these tables. Note that the current version of WebGround does not generate meaningful names for the columns. The system has no understanding of the semantics of the data.

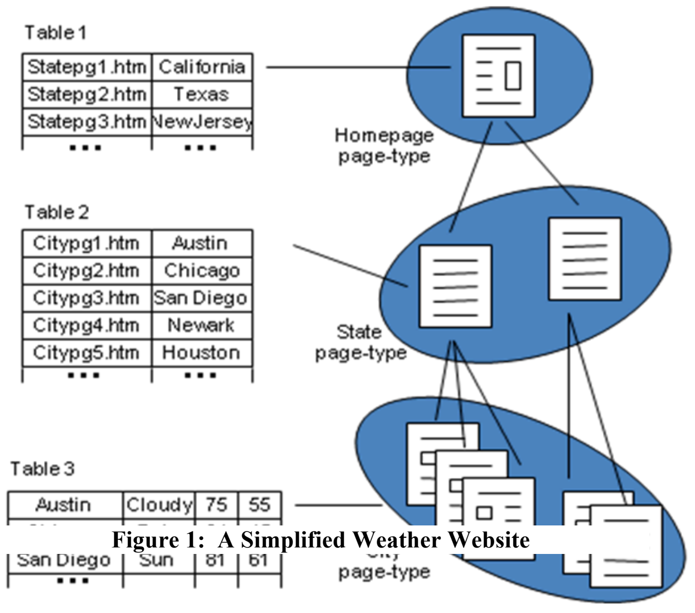

Table 1

Statepg1.htm	California
Statepg2.htm	Texas
Statepg3.htm	NewJersey
...	...

Table 2

Citypg1.htm	Austin
Citypg2.htm	Chicago
Citypg3.htm	San Diego
Citypg4.htm	Newark
Citypg5.htm	Houston
...	...

Table 3

Austin	Cloudy	75	55
San Diego	Sun	81	61
...			

Figure 1: A Simplified Weather Website

As shown in Figure 1, each page type corresponds to a cluster of one or more pages, and each table normally includes data from each of the pages in the cluster. Of course, for a given web site, there may be *many* tables generated, with many columns per table, because the system attempts to extract *all* the data on the site, including URLS, scripts, HTML, as well as text fields and numbers

V. FEATURE CREATION

The next step is to convert the tables into features that can be used for classification. Our current approach is to create a new predicate for each column in each table, using the projection operator. That is, for each column *c* in relational table *r* of site *s* we create a predicate $P_{s,r,c}$ such that for each data element *x* in column *c*, $P_{s,r,c}(x)$ holds. Then, for every data element in each table, we identify matching tags, effectively enabling a classifier to use the predicates as generalized tags. For instance, suppose we have a data object -- a photo -- named O_{27} that has a tag "San Diego", which we might represent as HasTag(O_{27}, "San Diego"), or some equivalent representation depending the classification algorithm used. If WebGround extracts the tables from the Forecast.com site shown in Figure 1, then column 2 of table 2, which lists U.S. cities, will generate the background fact $P_{\text{Forecast.com},2,2}$("San Diego"). Essentially, $P_{\text{Forecast.com},2,2}$ functions as an "Is-City" predicate. This enables the classifier to learn to classify photos about cities in general, as opposed to having to learn individual rules that mention each city.[2]

One issue that arises with employing background knowledge for classifying tagged data is that tags may not precisely match the acquired knowledge. In part, this arises because the tags are very informal. For instance, Flickr tags may not include spaces, even in long phrases, so a single tag might be "SanDiegoROCKS". To address this, we employ a tag cleaning process which rewrites each tag and heuristically inserts spaces to separate words. A tag *t* then *matches* background fact $P_{s,r,c}(f)$ iff the words in *f* are a subsequence of the words in *t*. This insures that verbose tags match the relevant background facts, while tags that only match part of a fact do not (they often have other meanings). So, for instance, the tag "IloveSanDiego" matches the city "San Diego". However, the tag "Diego" does not match the city "San Diego". For the purposes of this paper, we also refer to a predicate $P_{s,r,c}$ matching a tag *t*, by which we mean that there exists a background fact $P_{s,r,c}(f)$ such that *f* matches *t*.

Even though we create only unary predicates, as we noted earlier, there are potentially a large number of predicates created by WebGround. In our example, all of the columns in Figure 1 become predicates, including the list of cities, the list of weather conditions, the temperatures, etc. However, only a small minority of the columns in most tables are likely to be useful. Because of the cost of testing the predicates during learning, WebGround eliminates predicates that appear irrelevant. Specifically, WebGround includes the following criteria that potential predicates must satisfy:

- Each predicate must match more than k_1 distinct tags within the positive training examples. In our experiments, k_1 was set to 1, so that predicates have to convey more information than any single existing tag.
- The ratio of positive training examples that match the predicate to negative examples matched by the predicate must be greater than k_2. In our experiments, K_2 was set to 1, so that predicates must be at least minimally informative with respect to the target concept.
- Each predicate must have at least k_3 members in the corresponding column. In our experiments k_3 was set to 5, so that very short lists were eliminated from contention.

In our experiments we integrated several different learners with WebGround, including Weka's SMO SVM and J48 decision tree implementations (Hall, et al., 2009), Naïve Bayes (our own implementation), and the Aleph first-order logic ILP system (Srinivasan, 2001). In the case of SVMs and decision trees, the WebGround knowledge is encoded as propositional features of the data objects. For instance, when classifying Flickr photos, the tags associated with each picture are encoded as binary features, and the matching predicates are also encoded as binary features. That is, if a photo has a tag *t*, and $P_{s,r,c}$ is a predicate that matches *t*, we add the tag $P_{s,r,c}$ to the photo.

[2] Note that our method only takes advantage of a portion of the background knowledge, because only unary predicates are created. Later, we discuss creation of n-ary predicates.

For Naïve Bayes, we use a slightly different scheme to avoid adding many potentially-similar features to the photos (which violates the conditional independence assumption underlying Naïve Bayes). For each tag that matches one or more predicates, we replace the tag with the most informative matching predicate as the feature.[3] So each photo is associated with a "bag-of-tags" that includes the predicate names as part of this bag. Finally, for the ILP system we can encode the background knowledge directly as facts, as was described previously.

VI. EXPERIMENTAL RESULS

In this section, we report on experiments in classifying tagged NBA images and wildfire images, two very different domains. As noted above, we have previously built portals for both domains using a more manual, labor-intensive approach. The experiments here were conducted separately to evaluate WebGround's performance under controlled conditions.

A. NBA Experiments

Our first experiments focus on the NBA domain. To create our dataset of NBA photos we sampled Flickr using the search keywords "NBA" and "basketball", as described previously, and then manually identified pictures with current NBA players and/or pictures of NBA games (the same criteria we use for our portal) to create a labeled dataset.[4] This full data set contains 640 images, of which 204 were labeled as positive examples. For our classification task, we created 10 experimental folds, each consisting of these 204 positive samples and 204 randomly sampled negative examples. For each fold, we then broke the data into a test set composed of a random sample of 40% of the fold's data, setting aside the remaining 60% of the data for training. This results in 10 distinct folds, each with a set of positive and negative examples for testing. We kept the number of positive and negative examples equal in both the training and test data to simplify thresholding (i.e., we assume equal priors).

We then used WebGround to collect background knowledge using the same search keywords "NBA" and "basketball". As described above, the system queried Google with these terms, and retrieved the URLs returned on the first page of results. WebGround extracted a large number of relevant tables from sites such as NBA.com and Sportsillustrated.com, including tables listing players, teams, as well as other information.

The system then classified the images as NBA photos (or not) using our four different classifiers (Naïve Bayes, decision trees, SVM, and Aleph). Figure 2 shows the average F-measure of each classifier as the amount of training data increases from 10% to 60%. (Initially we set aside the full 60% of training data for each fold such that when we train on 20% of the data, for instance, we are including the 10% of training data it subsumes.)

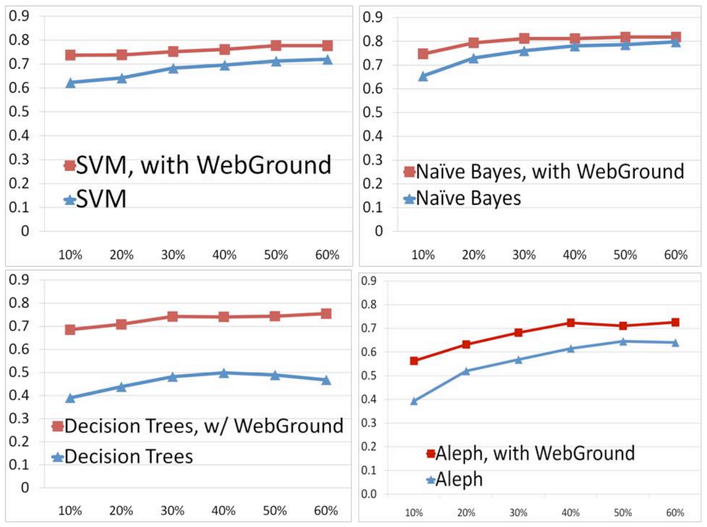

Figure 2: Experimental Results, NBA Domain

As the graphs illustrate, in all cases, the additional background knowledge resulted in accuracy increases due to better generalization and faster learning (in that fewer examples were required to achieve any given level of accuracy). We note that with one exception (Naïve Bayes at 60%) these F-measures are statistically significant at 95% confidence, using a two-tailed t-test.

Digging deeper into these results, we found that in fact, it is a boost in recall that results in the increased F-measure. In almost all cases, the differences in precision are not statistically significant, while the large boost in recall is. Therefore, the background knowledge allows the classifier to correctly identify more correct cases, without hindering its precision.

The key point is that the classifiers clearly make use of the background knowledge. For example, analysis of the decision trees show that acquired predicates are often used near the root of the trees, indicating that they are often more informative than the individual tags. If we look at the specific predicates acquired by WebGround that were incorporated in the classifiers, we find that lists of NBA players, teams, and locations were among those selected by the learners, as one might expect. As we noted earlier, the background knowledge appears particularly useful for helping the classifiers "recognize" infrequently occurring tags. For example, the tag "KobeBryant" occurred frequently enough in training so that the classifiers learned it was a strongly predictive of a positive instance. However, tags associated with less popular players, such as "Earl Watson" may not even show up in the training set. Thus, the learned predicate that matches NBA player names is particularly helpful for classifying pictures of less popular players.

One potential disadvantage of our current approach is that, in many cases, the lists included spurious elements that were extracted by overly aggressive heuristics. For instance, one list of NBA teams included terms such as "NHL" and "NFL" in

[3] Using the training data, we define a predicate's "informativeness" as a combination of its ratio of positive to negative samples in the training data, and its size (i.e., the number of elements in the corresponding column), under the assumption that larger predicates are more likely to match unseen tag samples in test data.

[4] To keep the experiments simple, the learning task was designed to focus solely on analyzing user tags. We ignored the photo's title and description. We also eliminated the tags "NBA" and "basketball" because they were used as the query terms. So in some respects this experimental task is harder than the actual application requires. (The same methodology was used for both the NBA and Wildfire dataset described later.)

addition to the team names. This may occur for several reasons. For example, WebGround may incorrectly cluster pages, incorporating too many pages into a cluster. In this case, the system may then create a single field from different HTML structures (e.g., two different lists) on multiple pages, and the result will be a noisy list. Alternatively, the syntactic structure of the pages may not precisely reflect the semantics of the target domain. For example, there may be a list of URLS for navigating the site that not only includes the NBA teams but also includes the terms "contact us" and "help".

Interesting, the learning algorithms achieve significantly higher accuracy than people when classifying the photos based on the tags alone (an artificial task for people, so perhaps not surprising). We recruited three volunteers, all NBA fans, who were given all the training data (60%) for a given fold, and asked to manually classify the photos based on the tags alone. Table 1 shows the average recall, precision, and F1-mesaure for the three human volunteers, compared to the four classifiers using WebGround.

We found that the humans had low recall and high precision, because they focused on tagged photos that they are confident about. Interestingly, we computed the Kappa agreement statistics between all pairs of volunteers, and found only moderate-to-fair agreement between the pairs. This means that each human user was able to accurately classify only a subset of the photos, and these subsets did not have high overlap. By contrast, the machine learners have better overall coverage.

Table 1: Comparison with Human Subjects on NBA domain

	Recall	Precision	F1
Humans	51.22%	85.6%	64.1%
SVM with Webground	75.61%	80.0%	77.7%
NaïveBayes with WebGround	89.76%	75.3%	81.8%
Decision Tree with WebGround	70.85%	81.1%	75.4%
Aleph ILP with WebGround	66.6%	82.5%	73.5%

B. *Wildfire Experiments*

Our experiments with the Wildfire domain were designed to validate our claim that WebGround can learn background knowledge to reduce the need for retraining when the environment changes. We used the same methodology to construct an initial data set, identifying Flickr pictures that showed wildfire incidents. This corpus contained 402 images, of which 100 were positive examples. Again, we broke the data into 10 folds, consisting of 60% of the positive samples (and an equal number of random negative samples), and 40% for testing.

In our evaluation, we trained the system, and then, to simulate the occurrence of new fires over time, we discarded the original testing data for each fold, and replaced it with a modified test set containing 20 new positive samples and 20 random negative samples. To generate this special test set we identified some wildfire incidents (listed by the National Forest Service) that were *not* included in our initially gathered dataset of 402 images, due to the nature of our sampling process and the fact that not all fires receive equal attention from Flickr users. We then searched Flickr, specifically looking for pictures of these fires to create the special test set.

We invoked WebGround to collect background knowledge using the search keywords "wildfire" and "fire", and used Yahoo as our search engine to find relevant sites (we used Yahoo rather than Google to demonstrate that the system is search engine agnostic). All three classifiers we tested did significantly better with the background knowledge extracted from these sites. (Aleph was not tested due to time constraints.) As with the NBA data, the best overall performance was achieved by Naïve Bayes. Naïve Bayes alone had 62.5% recall and 69.7% precision, whereas with Webground the algorithm achieved 85.5% recall and 71.4% precision. Thus, F1 improved from 65.58 to 77.52., an 18% boost. These results validate our claim that WebGround knowledge (which generalizes fire instances into predicate concepts) can produce results that are less brittle in the face of a changing environment.

Our review of the extracted concepts and the rules learned by the system shows a few concepts were particularly important. For example, relevant concepts included a list of recent fires from Inciweb.org, a government sponsored site that publishes information about U.S. wildfire incidents, including a list of wildfires during the last three months. As with the NBA results, the extracted lists often included a variety of spurious terms, however this did not significantly impact our results.

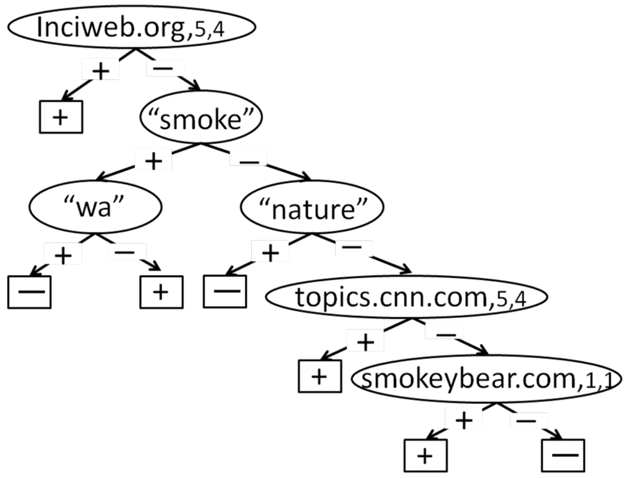

Figure 3: Example of a Learned Decision Tree

Figure 3 shows an illustrative decision tree learned by the system. The list of InciWeb fires is the root node in the tree. (Inciweb.org,5,4, refers to the 5th table, 4th column of the Inciweb.org site, which lists the names of fires.) Other nodes in the tree refer to a table from CNN (which happens to include terms that are fire-related) and a table from Smokeybear (which has some forest-related terms).[5] This tree also includes nodes that test for some basic tags, such as the tag "Nature".

[5] Note that once the system identifies which extracted concepts are included in a classifier, it is straightforward to build a Web agent that is specifically designed to monitor the site and update the data, as in (Lerman et al., 2003).

VII. RELATED WORK AND DISCUSSION

Our work focuses on *monitoring* folksonomy-oriented sites for tagged objects. Previous work has primarily considered *searching* such sites, the difference being that in monitoring applications the query Q is fixed and the set of data objects ϴ are (slowly) changing, whereas in searching, query Q changes on each invocation. While the work on search is generally quite different from our focus, some researchers (e.g. Passant, 2007, Specia & Motta, 2007) have considered how an ontology can be used to enrich or disambiguate tags, which is similar to our goal. However, as Limpens et al. (2009) point out, "the main limitation of such an approach is the limited coverage of currently available ontologies". We are not aware of any research on enhancing folksomy search where background theories are extracted from web sites.

More closely related to WebGround are systems that harvest Web data for constructing knowledgebases or ontologies. Most of this research focuses on unstructured text sources (e.g, Schoenmackers et al, 2010; Kozareva et al., 2008). Several researchers have focused on automously extracting ontological data from structured sites, such as Wikipedia (e.g., Suchanek et al, 2008) However, as useful as it is, Wikipedia is not as comprehensive as the Web itself (e.g., Wikipedia does not currently include a list of recent wildfires.) There has been comparatively little work on unsupervised extraction of relational data from semi-structured sites. One reason, as we described, is that the harvested tables can be noisy, and much of the data is not suitable for inclusion in an ontology. Because WebGround has a concrete classification task, it has a clear measure of the utility of the harvested data.

Other task-oriented Web harvesting systems exist, of course, but many of these are customized to extract and process data using task-specific methods. For instance, Ern et al. (2005) describe a crossword puzzle solver that extracts potential answers from web pages by looking specifically for words/terms of given length. In contrast, WebGround's extractor (which harvests arbitrary relational data) and classifiers are completely generic.

One direction for future work is to create more complex predicates from the harvested relational data. Currently, WebGround creates only unary predicates. In effect, although the system is harvesting tables, we are considering each column as an isolated list. We could theoretically take advantage of the rows of the table to create binary, or even n-ary predicates, so that relations could be used by the classifier. In the NBA domain, this would allow the creation of predicates capturing "teammate-of" or "plays for" relations. To avoid overwhelming the classifier with too many additional predicates, we believe that more sophisticated predicate selection criteria could be developed.

VIII. CONCLUSION

A distinguishing feature of our work is that the knowledge acquisition process is both autonomous and driven by the classification problem. This is both a strength and a weakness. On the positive side, the harvesting process is both highly directed and there is a clear goal – to improve the performance of the classifier. On the other hand, this leads to potentially myopic behavior. The system collects information that is relevant to the problem, but does not create a clean and complete domain theory. As a result, the acquired knowledge often contains spurious data items, as we pointed out.

Our work takes a step towards more autonomous classifier systems that can learn about a domain. Specifically, we have shown in two domains that using rich features extracted by WebGround resulted in significant improvements in classification performance for all the classifiers we tested. We believe this approach is a promising direction for future research.

REFERENCES

[1] Crescenzi, V., Mecca, G., Merialdo, P. 2001 RoadRunner: Towards Automatic Data Extraction from Large Web Sites. VLDB: 109-118.

[2] Ern, M., Angelini, G., and Gori, M. 2005.Webcrow: A web-based system for crossword solving, Proc. AAAI .

[3] Kozareva, Z., E. Riloff, and E.H. Hovy. 2008. Semantic Class Learning from the Web with Hyponym Pattern Linkage Graphs. Proc. ACL-08.

[4] Gazen, B. & Minton, S. 2005 AutoFeed: an unsupervised learning system for generating webfeeds., K-CAP : 3-10.

[5] Hall, M., Frank, E.,Holmes, G., Pfahringer, B., Reutemann, P., and Witten, I.H., 2009, The WEKA Data Mining Software: An Update. SIGKDD Explorations 11(1).

[6] Lerman, K., Minton, S., Knoblock, C. A., 2003. Wrapper Maintenance: A Machine Learning Approach. JAIR 18.

[7] Limpens, F., Gandon, F., Buffa, M., 2009. Linking Folksonomiesand Ontologies for Supporting Knowledge Sharing: a State of the Art. Tech. Report, ISICIL.

[8] Muslea, I., Minton, S. and Knoblock, C.A. 2001 Hierarchical Wrapper Induction for Semistructured Information Sources. Autonomous Agents and Multi-Agent Systems 4(1/2): 93-114.

[9] Passant, A. 2007. Using Ontologies to Strengthen Folksonomies and Enrich Information Retrieval in Weblogs, ICWSM.

[10] Schoenmackers, S., Davis, J., Etzioni O. and Weld, D.S. 2010. Learning First-Order Horn Clauses from Web Text. Empirical Methods in NLP.

[11] Specia, L. & Motta, E. 2007. Integrating folksonomies with the semantic web. 4th European Semantic Web Conference..

[12] Suchanek, F.M., Kasneci, G., Weikum, G. 2008. YAGO: A Large Ontology from Wikipedia and WordNet. J. Web Sem. 6(3): 203-217.

[13] Srinivasan, A. The Aleph Manual, 2001.

[14] Weld, D.S., Hoffmann, R., and Wu, F., 2009, Using Wikipedia to Bootstrap Open Information Extraction, ACM SIGMOD Record, 37 (4)