In Proceedings of the 2004 International Conference on Machine Learning and Applications (ICMLA '04), pages 10-17.

LASSO: A Learning Architecture for Semantic Web Ontologies

Christopher N. Hammack
Raytheon Company
1620 Wilshire Drive, Suite 301
Bellevue, NE 68005
chammack@fadedsky.com

Stephen D. Scott
Dept. of Computer Science & Engineering
University of Nebraska
Lincoln, NE 68588-0115
sscott@cse.unl.edu

Abstract

Expressing web page content in a way that computers can understand is the key to a semantic web. Generating ontological information from the web automatically using machine learning shows great promise towards this goal. We present LASSO, an architecture that combines distributed components for training web page classifiers via machine learning and information extraction, and then labels new pages with the classifiers. LASSO's results are semantic models of web pages stored in a database back end, and the models are defined with respect to whatever ontology the user chooses. LASSO can be used to build a wide variety of applications or can be used as a collaborative experimentation workbench. We give as part of our proof-of-concept prototype an application of an enhanced ontological search engine. We also describe how LASSO can be used to compare machine learning algorithms and analyze our system with a code reuse metric.

1. Introduction

Recently, much attention has been focused on producing a *semantic web*, which would allow computers to semantically parse web content. A semantic web would enable many applications that are presently difficult or require specialized solutions such as enhanced searches that treat the web as a database. Conventionally, information on the semantic web is represented in *ontologies*, which are the encapsulation of information. Ontologies have traditionally been published on the web using a variety of languages, including XML. The process of attaching semantic ontological information to web pages is referred to as *annotating*. Manually annotating pages is tedious.

Tools have been created to annotate web pages which allow the user to be abstracted from the relatively complex syntax that these languages have. Unfortunately, it is clear that these tools have not yet caught on—the tools are primarily used in academia, and are not yet integrated in any widely-used commercial web development software. The number of known ontology annotated pages is significantly smaller than the number of pages on the web. DAML Crawler[1] indexes pages that have been annotated using DAML[2], which is the most pervasive ontological standard. DAML Crawler has indexed approximately 21,000 annotated pages. In comparison, there are billions of unannotated pages indexed by Google.

To automate annotation, it is natural to turn to machine learning and information extraction. One of the first steps towards building ontologies is to determine the general character of a page. That is, what type of information is represented. Similarly, we can continue to make such classifications using hierarchical classification schemes. In each step, we can become more specific as to the type of information represented on a page. This sort of categorization is possible using machine learning. Similarly, once one has determined the most specific nature of a page possible according to the categorization scheme, one can perform certain types of manipulations that can retrieve certain properties of the page using information extraction. These ideas are not new: there are several projects that have similar goals. However, most are *ad hoc* implementations for academia, concentrate on a single learning or information extraction algorithm, or do not provide a complete set of tools to perform all necessary tasks.

LASSO is a complete web classification and extraction architecture that provides all the components necessary to harvest pages, extract and select features, train classifiers, and use classifiers to label pages with respect to any ontology. It is modular and allows custom components to be developed by anyone. These components can be locally or remotely located, as LASSO utilizes a cross-platform/cross-programming language remote execution protocol. LASSO provides services using a workflow-based model to allow components to be plugged together in many different ways,

1 `http://www.daml.org/crawler`
2 `http://www.daml.org/2000/10/daml-oil`

unlike many systems that require a strict, inflexible data path. Because LASSO stores all ontological results in a general database, it is not tied to any specific ontology markup language. Thus, LASSO provides an interactive, multi-user workbench for users interested in experimenting with web classification and machine learning. LASSO also provides a base that can be used to build applications.

More details on LASSO's design are in [5]. A prototype of LASSO is at `http://lasso.unl.edu/`. APIs are available for Java and C++ (experienced users can use any language using SOAP). We used our prototype in an experimental and a classroom setting as an experimentation workbench for comparing machine learning algorithms on web-based applications. We also developed numerous prototype applications with our architecture, including an enhanced search engine utilizing Google that uses a combination of Google index and semantic results in LASSO to produce and rank results. Similarly, a prototype annotation tool which is capable of exporting RDF [7] annotation from LASSO's database has been developed. This application can be used to add RDF annotation to pages which have been semantically marked up by LASSO. Finally, we evaluated our system by the level of code reuse it facilitates.

2. Features

LASSO has a number of key features. It provides a flexible architecture to plug in parsers, feature extractors, learning algorithms and trained classifiers. Since the components are actually web services, this provides a number of other advantages. First, the core and the components are truly independent. There are no code intricacies that tie the two together. Components can be replaced in their entirety with zero impact on the system. Second, the services can be remote, which allows users to add components without needing permission on a core LASSO server or needing to set up their own LASSO server. Finally, remote components allow for distributed computing.

Not only are LASSO's components distributable, but so are many of its core components. Multiple web servers and core deployment servers are possible using the same data. This allows for load distribution across the entire suite of facilities. In Figure 1, we see an overview of the various LASSO components. Each component is shown to be running on an individual computer. Alternatively, all components could run on a single computer or each component could run on multiple computers.

In addition to providing a componentized, distributed system, LASSO also offers flexible data manipulation. Data are in a logical ontological-style datastore that can be used to build powerful applications. Since the datastore layer is removable, it is possible to replace the current datastore with ontological datastores that provide more ontology-based features when they become available.

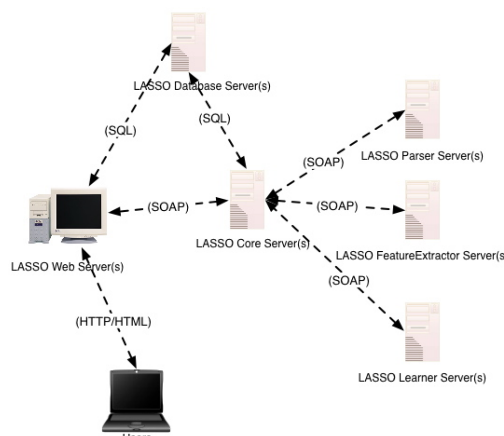

Figure 1. A systems overview of LASSO.

LASSO is built on a workflow model. This allows components to be plugged together in a flexible way. The only requirements to plugging two components together is that they share a common data exchange format. LASSO allows the user to use any built-in services or any services shared by others.

LASSO also provides an interactive environment that encourages sharing. LASSO is a user-based system. By default, all objects created in the system belong only to the user who created them. However, LASSO allows users to share components with other users. In traditional systems, this would require distributing the code or a binary of the application that one wishes to make available. In LASSO, one simply specifies that the resource should be public.

LASSO has been implemented as a significant (over 40,000 lines of code) prototype. In addition to the flexible core architecture that allows many types of components to be integrated into the system without modifying the core system, a number of prototype components have been developed. These include HTML and PDF parsing capabilities, general feature transforms, mutual information feature selection and extraction, and learning using a variety of learning algorithms such as decision trees, naïve Bayes, support vector machines, etc.

3. Related Work

Projects such as AutoSHOE [8] and MnM [12] already attempt to build software that provides automatic annotation services. The primary difference between LASSO and these projects is that LASSO allows the utilization of any type of algorithm through its componentized architecture and workflow design. Another major difference is that LASSO provides a complete architecture that can be used to build applications that can leverage machine learning and ontolo-

gies or it can be used collaboratively for research or instruction. In contrast, both AutoSHOE and MnM provide annotation services for a specific ontology (SHOE and DAML respectively), and are intended for producing markup only.

The popular web portal Yahoo![3] provides an ontological-like hierarchical classification of web pages. Web pages belong to an increasingly specific classification. However, this set of classifications is performed manually by a set of "experts" by hand, a daunting task.

4. Tasks LASSO Performs

LASSO utilizes a workflow design in which components are put together in a pipeline to accomplish a certain task. LASSO allows these components to be put together in any way that the user desires as long as integrity is maintained. The tasks are harvesting, parsing, feature extraction, learning, classification and information extraction. In Figure 2 is an example of combining tasks to learn a classifier. In Figure 3 is an example of using this classifer to label new pages.

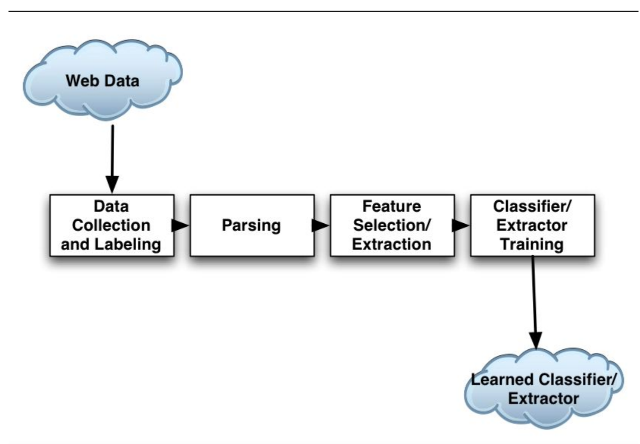

Figure 2. Learning a classifier in LASSO.

Datasets are collections of URLs. They may be used for either training or annotating. Labeled URLs in a dataset may be added in one of two ways. The first way is simply a batch introduction: the user provides a list of URLs be added and then labels them by hand or instructs a classifier to label them. The other way is to have LASSO crawl a URL and collect examples from a site. These URLs can be assigned a default label that can later be changed.

Parsing is generally the first component that is used on collected data. Since HTML is difficult to handle directly, we transform HTML into a more strict XML (XHTML)

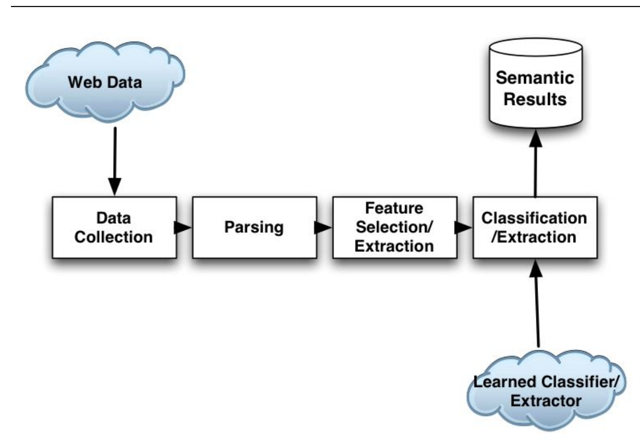

Figure 3. Labeling in LASSO.

format, which can be handled by a variety of parsers. We can also build parsers to transform formats such as PDF or Postscript into other intermediate forms.

Most tasks involve *feature extraction*. In LASSO, we define this as transforming the representation of the web data to a form that is usable by a particular learning algorithm. Feature extraction generally also includes *feature selection*, which is the process of choosing the relevant information.

One of LASSO's feature extractor modules converts HTML into text since many web classification algorithms are text classification algorithms. Additionally, LASSO feature extraction components may offer services such as using only user-presentable data. It can also constrain the tags from which HTML may be extracted. Feature extraction may also provide standard information extraction services such as stop-word removal and stemming.

Typical representation of features in text classification problems involves using a numeric feature for each subset of the words in a set of documents. Some feature selectors which have been implemented in LASSO include term frequency selection and mutual information selection [1]. Users may add any sort of feature extraction or selection algorithm they wish by building their own component.

The next important task in LASSO is the learning/annotating step. There are three distinct machine learning ontological tasks in the LASSO system. These tasks are ontology membership, class membership and expressing field values. These tasks apply to both learning (which labeled data is available), and annotating unlabeled data with trained classifiers.

Ontology Membership The ontology membership task determines whether or not a given page can be adequately described by an ontology. Ontology membership can act as the first layer of semantic categorization. Simply determin-

3 http://www.yahoo.com

ing which ontologies a page can be described by is very powerful on its own. Membership is also important because the other capabilities of the ontology are not useful for describing pages outside of the domain for which the ontology was intended. An example of determining ontological membership of a page is shown in Figure 4. In this example, the classification task is to determine whether the page is describable by the ShoppingPage ontology.

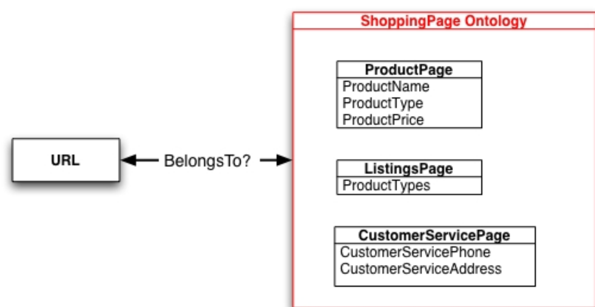

Figure 4. LASSO's ontology membership task.

Machine learning is easily applied to the problem of ontology membership. When building a set of tools to classify and annotate according to an ontology, the first step is to provide examples of pages that can be described by the ontology and examples of pages that cannot. Classifier modules have been built that utilize the Rainbow [9] text classification suite and the WEKA[4] suite of classifier algorithms. Naïve Bayes and support vector machines trained by SMO [10] have been particularly effective.

Class Membership The class membership task in LASSO is the process in which pages are determined to belong to one of several *classes* within an ontology. There are several approaches to this task in LASSO, each with benefits and drawbacks. The approach used should be based upon analysis of the particular ontology. In Figure 5 we can see a page being analyzed for membership in a number of classes in the ShoppingPage ontology. The three classes defined in this ontology are ProductPage (page describes a single product), ListingsPage (page lists several products) and CustomerServicePage. Presumably, these classes are mutually exclusive.

LASSO allows the user to choose between binary classification and multi-class classification. With binary classification, a classifier is trained for each class. With multi-class classification, only one classifier is trained per ontology. This single classifier would choose the most appropriate class for each page.

4 http://www.cs.waikato.ac.nz/ml/weka/

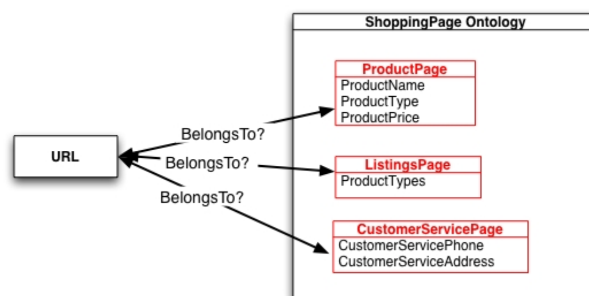

Figure 5. LASSO's class membership task.

Field Values The final major ontological learning task in LASSO is determining field values. There are two major approaches to determining the value of fields in LASSO. The first is *field enumeration* and the other is *field extraction*.

Field enumeration is useful when the field has a discrete set of values. For example in Figure 6, in an ontology that describes shopping sites, there may be a class that describes product pages. In the ProductPage class, we have a field that describes the type of the product displayed on this page, e.g. books, electronics, software and personal products. Allowing free-form product characterization would probably not be very accurate in many cases and precision would likely be an issue when trying to use free-form results.

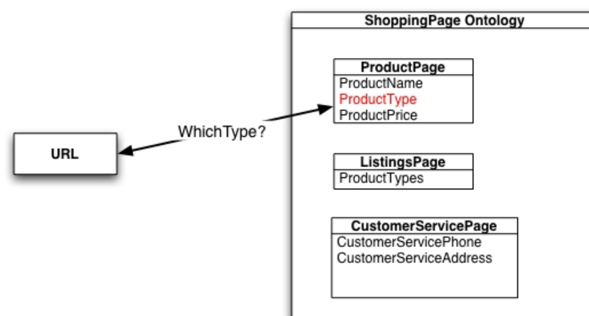

Figure 6. LASSO's field enumeration task.

In field enumeration, we train a classifier to determine which of the possible values is a *best-fit* match. The potential values are specified in the definitions of most ontologies, so getting a list of possible values is not difficult.

Field extraction is useful when the field has a continuous (free-form) set of values. Continuing our example in Figure 7, an example of this is "price of an object". A traditional classifier is unable to determine this sort of information from a page. However, using a machine learning algorithm that is capable of learning information extraction rules is a prime candidate for this task.

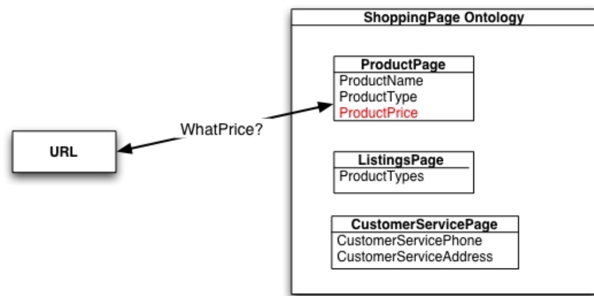

Figure 7. LASSO's field extraction task.

In field extraction, sample documents that belong to the ontology and the particular class are supplied. These documents are then marked up. The mark-up procedure consists of explicitly stating what the field values are in the document and where they are found in the training page. Given a number of examples, the learning algorithm will determine a set of extraction rules which can be applied on new unmarked pages. When provided a new instance, the field will be extracted, and that value will be used for that field in LASSO. Hidden Markov model approaches [6] and learnable information extractors (e.g. Rapier [2], LP^2 [3], and BWI [4]) are good candidates for algorithms here.

5. System Design

LASSO consists of many discrete components that fit together in a tightly-coupled way. The LASSO core is the most basic set of components. They perform most of the connectivity service and management of the entire LASSO system. Two major database components also make up the lower-most level of LASSO (the persistent filesystem datastore is simply an extension of the Core Relational Database). Through LASSO API and Data Access Objects, it is possible to build both applications and a user interface layer. The user interface layer is built using Struts/JSP.

5.1. Core

LASSO's core consists of the main components in LASSO that perform the coordination and management of tasks in LASSO. The core components interact heavily with the database, and are managed by the web interface. The core primarily deals with data management, remote execution through SOAP, and resource management. Core also provides a number of basic system functions to many of the other components.

SOAP[5] is used extensively in LASSO. SOAP allows many of LASSO's components to be independent and provides for remote and distributed execution. By providing an API based on SOAP, dependencies between modules are greatly reduced and components can be built by anyone who has the specification. Any component can be located anywhere in the world, which allows anyone to participate in building components for LASSO. Another major benefit is that users do not have to maintain their own LASSO server—they merely build their component and link it into the system. SOAP allows the user to participate in the system without having access to the LASSO core server code.

In order to maintain maximum flexibility in LASSO and to support as many types of each component possible, LASSO allows dynamic data formats as input and output of any remote component. These data formats are defined in LASSO and described in a database without making any changes to the core and can consist of custom data structures or general MIME types. The system allows combinations of traditional data types and custom order structures constructed using a serialized XML format. Because building SOAP components with complex data structures is complex, we provide a set of APIs in C++ and Java that use some of the most common data formats and require no SOAP knowledge. This allows users to concentrate on the algorithmic sections of their code.

5.1.1. Pipeline-based Tasks LASSO is designed to be a workflow model-based system. This means that the overall task is broken up into a series of discrete steps called a pipeline. The system must ensure that the purpose of each individual step is well-defined, the connection between steps is well-defined and the overall goal is accomplished. These goals include the learning and classification tasks we defined earlier and some convenience tasks that can store intermediate results.

5.2. Databases

The database in LASSO consists of three major types of datastore components. The first component is the core relational database (CRD) which is used to maintain the data

5 http://www.w3.org/TR/SOAP/

associated with the workflow, permissions and data tracking features in LASSO. The second is the filesystem datastore that is used to store intermediate results. The final component is the ontological datastore which is used to store long term ontological results from LASSO.

The core relational database (CRD) stores the data associated with LASSO workflow and supports the web interface. A large portion of the CRD is dedicated to storing information about LASSO's components. For each component, the name of the component, the web service URL and permissions must be stored. Each component also has flexible Access Control List (ACL) style permissions. The input and output interface DataFormat of each component must also be maintained, in addition to the definition of any custom DataFormats.

A significant portion of the CRD is data associated with pipeline construction and execution. A description of each stage of the pipeline (and any options) and a description of the overall pipeline is maintained in the database. Additionally, all job scheduling is also stored in the database, including mutual exclusion information. In the LASSO prototype, the CRD is implemented using a MySQL[6] database, managed using a combination of custom queries using JDBC and using the JDAO model using autogenerated code produced by FireStorm[7].

The final component is the ontological datastore, which is the storage of the results of classifications and extractions performed by LASSO. LASSO is designed so that the ontological datastore can be replaced. In the LASSO prototype, the ontological datastore maintains a representation of ontological structure using MySQL. For more advanced ontological manipulation and representation, LASSO could be adapted to use an ontology server or using a true persistent ontological datastore using a project like HP's JENA[8].

5.3. Web Interface

LASSO has an HTML-based web interface for managing learning tasks. The prototype is written in Java using the Caucho Resin application server and using the Jakarta Struts MVC framework for presentation. LASSO is modularly designed so that it can also be configured using command line utilities or custom interfaces.

The web interface provides a front end to several tasks. One task is component registration. The component name, location (web service URL) and data format of the input and output of the component must be provided. Another task is pipeline construction. Users specify the name and the task to accomplish and then visually "build" the flow of data into components. After providing this information, a validation step is performed to check the validity of the pipeline. The user interface also provides a job manager that can manage the currently running jobs and submit new jobs for processing. Finally, the interface provides a number of analysis tools that allow the user to graphically view results.

6. Applications and Analysis

We now discuss applications of LASSO and an analysis in code reuse[9]. First, we describe our prototyped enhanced web search engine. We then examine the use of LASSO as a tool for comparing machine learning algorithms on web data. Finally, we analyze our system based on the amount of code reuse it provides.

6.1. Enhanced Google Search

LASSO can be used to build a search engine technology. Web searches that are restricted by ontology could prove useful. For example, perhaps we might want to search for pages that contain products related to motorcycle helmets. A traditional web search on "motor helmets shopping" does produce some results, but only sites which explicitly use the word "shopping" are likely to appear in the results. If instead we could search for motorcycle helmets but only highly rank sites that were actual shopping sites, we could produce better results.

We created a prototype in LASSO to do exactly that. Users can provide traditional search terms in addition to an ontology that best describes the pages they seek. The search is completed using Google and results are then narrowed. Since we do not have direct access to Google's index, we have to settle for only examining the top n results returned by Google. A real implementation would be more tightly coupled with the crawler and database. Additionally, a real implementation may intelligently match certain key words to ontologies.

In Figure 8, we see an example of the prototype search engine. In this example, the user searched for "motor helmets" in the shopping ontology. The first page returned (which was not the first page returned in a regular Google search) matches the shopping ontology (indicated by the green check) and the second page does not (indicated by the red X). Visiting the first site validates that it is a shopping site for motor helmets. The second site (which was the first hit in Google) is actually a motor helmet decals enthusiast site, and is not really a shopping site.

This application is currently integrated into the LASSO prototype using the shopping, news and personal web page ontologies. In the prototype, we do only ontology matching.

6 http://www.mysql.org/
7 http://www.codefutures.com/
8 http://www.hpl.hp.com/semweb/index.html

9 Some applications, such as XML annotation of web pages with RDF [7], are omitted to save space.

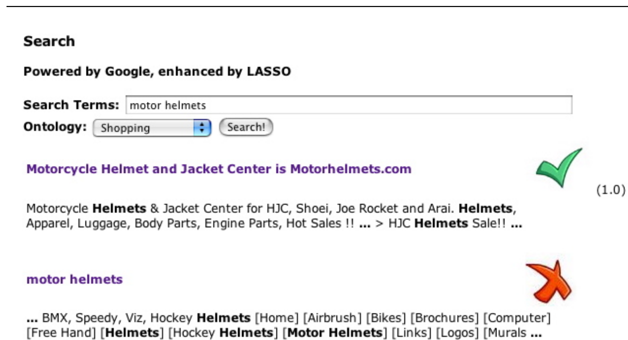

Figure 8. A prototype search engine using LASSO.

It would also be possible to add class and subclass matching capabilities to implement Yahoo!-style hierarchical categories, and to restrict pages that match certain fields. The training set consisted of 2190 examples, containing examples representing all of the ontologies. Using WEKA's SMO and naïve Bayes algorithms, binary classifiers were built for establishing ontology membership.

The search engine was designed to work in two different ways. First is a "cached" mode, which is similar to a real search engine implementation. In this mode, the results came directly from LASSO's database; no classifiers were invoked at search time. In the prototype, the first fifty results were retrieved from Google that match the search string. Then each URL was checked for membership in the ontology in the LASSO ontology datastore. Those matching the ontology were ranked higher than those that did not. To break ties, the original Google rank was used. If a page was not in LASSO's ontological datastore, it was ranked low.

In the other mode of operation, referred to as "live" mode, the search engine retrieves from the web the top fifty results from Google, and then runs a classifier on them. While live mode is much more useful than cached mode (because the LASSO database is miniscule in comparison to Google's), it is very slow compared to cached mode. The cached mode has similar performance (on the order of seconds) to a regular Google search, but the "live" mode may take many times longer (approximately a minute or so) but it is only intended to demonstrate the capabilities of the system. A full implementation of a LASSO-enhanced search engine would run in cached mode since the database would be integrated with the search engine's.

Generally, our enhanced search engine shows improvements over a traditional Google search. For example, in our "motor helmets" query, a Google search returns a large number of irrelevant documents in the top 50 results. Our enhanced search engine reorders the list so that the best results are first. If the query size increased beyond 50, we expect even better results as the classifier would further lower the rankings of irrelevant results. Similarly, if we search for "motor helmets shopping" we only get results which have the word shopping (or its word stems) in it and lose a great deal of the results. Sites that contain the word shopping are artificially ranked higher because they contain that word, even though they are no more a shopping site than many others that do not contain the term.

6.2. Research and Educational Tool

LASSO can be used for both research and educational purposes to analyze machine learning algorithms. During the Fall 2003 semester at the University of Nebraska, students in the machine learning course used LASSO to build classifiers such as decision trees, neural networks, naïve Bayes, boosting and support vector machines for use in LASSO classification.

By using LASSO, the students had exposure to building real world applications using machine learning and have gained more experience in modular, component-based programming for users to evaluate the performance of their pipeline jobs. The first tool is an overall classification summary in the form of a confusion matrix. Since it is sometimes desirable to get more detailed analysis than the performance analysis provided by the confusion matrix, LASSO also allows the user to evaluate the performance of a learner by showing the label the learner produced and compares it to the human "expert" label.

6.3. Analysis: Code Reuse

Although it is difficult to quantify LASSO's success as an educational tool and a basis for building applications, we can evaluate LASSO from an architectural standpoint. An often-used metric from software engineering for measuring the advantage of using a framework over writing code from scratch is code reuse [11]. The central issue in code reuse is choosing applicable code in the calculation. We address this by making several calculations with varying quantities of the code considered. Specifically, we consider the case where a user is constructing a learning algorithm from scratch and wishes to compare writing their own parsing, extraction and learning algorithms to implementing only the learning algorithm and using LASSO. In this example, we consider constructing a small (approximately 800 line) decision tree learner. We provide statistics in Table 1, including only the components themselves and using the entire LASSO system (the most fair comparison), which includes the database system, the analysis tools and front end. We further split each sample by giving statistics that include the

auxiliary code and those that do not. The auxiliary code includes the database management code and SOAP glue code.

Subset	Code Reuse
Components	62.2%
Components+Auxilary	64.9%
Complete LASSO	87.3%
Complete LASSO+Auxilary	94.8%

Table 1. Reuse statistics for creating a learner component.

As would be expected from a framework, LASSO provides a great deal of functionality that can be leveraged with no effort from developers. The learner component contributes a small amount of new code to the system—approximately one third of the component code, and a fraction (depending on whether auxiliary code is considered) of the total system. This shows that the system has been designed to be truly reusable. A developer needs no knowledge of (or access to) the code of the other components, or even the binaries. Although most users appreciate this ability, the option of the developer to provide 100% of the component code is also maintained for maximum flexibility.

7. Conclusions and Future Work

The goal of LASSO was to provide a flexible and powerful framework that could be used as a workbench for machine learning experimentation and be used to build applications that utilize machine learning and ontological information on the web. LASSO provides a componentized and distributed architecture that can be used for every step of the process of web page classification. LASSO can be extended to support a wide variety of new parsers, feature selectors/extractors, classification algorithms and information extraction algorithms without requiring code changes. LASSO has been used as an online machine learning workbench, and as a basis for applications using machine learning. LASSO proves that a modular, distributed, web machine learning framework is possible.

LASSO is a prototype and under considerable development. There are a number of extensions and items left for future work. While the core supports information extractors, advanced information extraction learning algorithms need to be adapted to be LASSO components. HMMs, RAPIER, BWI, etc. would all be good candidates for this.

LASSO's internal ontological representation should be upgraded to support some of the modern ontology standard features. HP's JENA would be a good candidate to replace the present ontological datastore. However, currently JENA's persistent datastore is experimental.

Expressing the relationships between pages could potentially be useful for extracting information. For example, consider that a student's web page may link to pages describing a class for which he is participating. The IE component in LASSO would likely be able to determine that he is a student in the class but it may not be able to determine a relationship between that student and the teacher of the course who may also have links to the page.

Acknowledgments

This project was supported by NIH Grant Number RR-P20RR17675 from the IDeA program of the National Center for Research Resources. It was also supported in part by NSF grant CCR-0092761. Christopher Hammack performed this work at the University of Nebraska.

References

[1] R. Battiti. Using mutual information for selecting features in supervised neural net learning. *IEEE Transactions on Neural Networks*, 5(4), 1994.

[2] M. E. Califf and R. J. Mooney. Relational learning of pattern-match rules for information extraction. In *Working Notes of AAAI Spring Symposium on Applying Machine Learning to Discourse Processing*, pages 6–11, Menlo Park, CA, 1998.

[3] F. Ciravegna. $(LP)^2$, an adaptive algorithm for information extraction from web-related texts. In *IJCAI Workshop on Adaptive Text Extraction and Mining*, 2001.

[4] D. Freitag and N. Kushmerick. Boosted wrapper induction. In *ECAI2000 Workshop on Machine Learning for IE*, pages 577–583, 2000.

[5] C. N. Hammack. LASSO: A learning architecture for semantic web ontologies. Master's thesis, Dept. of Comp. Science, University of Nebraska, 2003.

[6] A. McCallum K. Seymore and R. Rosenfeld. Learning hidden Markov model structure for information extraction. In *AAAI Workshop on Machine Learning for Information Extraction*, 1999.

[7] O. Lassila and R. Swick. Resource description framework (RDF): Model and syntax specification., 1999. `http://www.w3.org/TR/REC-rdf-syntax/`.

[8] Q. F. Lin, S. Scott, and S. C. Seth. A machine learning framework for automatically annotating web pages with simple HTML ontology extension (SHOE). In *Proc. of the International Conf. on Intelligent Agents, Web Technology and Internet Commerce*, pages 303–310, 2001.

[9] A. K. McCallum. Bow: A toolkit for statistical language modeling, text retrieval, classification and clustering, 1996. `http://www.cs.cmu.edu/~mccallum/bow`.

[10] J. Platt. Fast training of support vector machines using sequential minimal optimization. In *Advances in Kernel Methods: Support Vector Machines*, 1998.

[11] J. S. Poulin. *Measuring Software Reuse: Principles, Practices, and Economic Models.* Addison-Wesley, 1997.

[12] M. Vargas-Vera, E. Motta, J. Domingue, M. Lanzoni, Arthur Stutt, and Fabio Ciravegna. MnM: Ontology driven semi-automatic and automatic support for semantic markup, 2002.