Automatically Extracting Features for Concept Learning from the Web

William W. Cohen WCOHEN@RESEARCH.ATT.COM
AT&T Labs–Research, Shannon Laboratory, 180 Park Avenue, Florham Park, NJ, 07932 USA

Abstract

We propose a simple, general-purpose method that takes as input a set of instances and a collection of web pages, and produces a set of new features, defined over the given instances. Each generated feature corresponds either to a word from an HTML header that appears to modify an instance, or else to a "position" in an HTML document that appears to contain an instance. In learning to classify musical artists, for example, the generated feature $g_{\text{classical}}$ might be true for all instances that appear in a web page below a header element containing the word "classical". Experiments show that the generated features reduce the error rate of learned classifiers in a wide variety of situations.

1. Introduction

Concept learning requires having appropriate features to describe the instances to be classified. Since appropriate features are not always available, there is a substantial literature on constructing new ones—*e.g.*, by combining old features in some interesting way (Pagallo & Haussler, 1990; Matheus & Rendell, 1989).

An alternative way of generating new features is to use external information sources, such as databases found on the web. This approach cannot always be used—*e.g.*, one is not likely to find additional information about individual Pima Indian patients on the web—but when it is applicable, it is often useful. For example, Basu et al. (1998) formalize movie recommendation as a classification problem, and show that classification performance can be improved using features extracted from the web. Unfortunately, writing programs that extract data from the web remains time-consuming, despite recent progress (Kushmerick et al., 1997; Muslea et al., 1998; Hammer et al., 1997; Ashish & Knoblock, 1997).

This paper is based on the conjecture that extracting data for learning systems might be qualitatively different from extracting data for other purposes. In particular, if web data is being extracted and then inserted into a conventional database system (for instance), then the extracted data must be relatively clean. This requirement often means that a relatively complex extraction procedure must be used, and that this procedure must be written (or verified) by a person. However, learning systems are quite tolerant of noisy data. This suggests that other approaches to data extraction might be useful for feature construction—approaches which extract lots of noisy data quickly, with little human cost.

We propose a simple general-purpose method that takes as input a collection of web pages and a set of instances, and produces a set of new features, defined over the given instances. For example, consider a learning problem in which the instances are the names of musical artists. The generated feature $g_{\text{classical}}$ might be true for all instances that appear in a web page below a header element containing the word "classical". Other generated features might be true for all instances that appear on particular web pages, or that appear in particular table or lists. When this "expansion" process is successful, adding the new features to the original dataset makes concept learning easier: *i.e.*, running a learning system on the augmented dataset will yield a lower error rate than running the same learning system on the original dataset.

To a first approximation, one can think of the expansion method as generating features based on a large number of automatically-generated extraction programs. Most of the features proposed will be meaningless, but a few might be useful, and if even a few useful features are proposed the concept learning system may be able to improve the error rate.

Below we describe the experimental data used to evaluate our expansion method, and then the expansion method itself. We finally present a series of experiments showing that the generated features reduce the error rate of learned classifiers in a wide variety of situations.

2. Experimental data

2.1 Choice of Benchmark Problems

We will assume that each example includes some commonly used natural-language "name" that identifies the instance—*e.g.*, the title of a movie, or the name of a person. We also assume that the associated set of web pages is well-chosen—*i.e.*, that it is a set of data-rich, well-structured HTML pages that are related in some way to the instances.

Four benchmark classification problems satisfying these conditions were used to evaluate our approach. In first benchmark problem, *music*, the goal is to classify into genres the musical artists appearing in a large on-line music collection. This dataset was collected as part of an earlier research project (Cohen & Fan, 2000) in which web data was used for collaborative filtering, and hence the dataset is associated with a set of data-rich web pages. The web pages were collected by an automatic system.[1]

Three additional suitable benchmark problems were taken from another earlier paper (Cohen & Hirsh, 1998). In *games* the name of a computer game is mapped to a broad category for that game (*e.g.*, *action*, *adventure*). In *birdcom* and *birdsci*, the name of a species of North American bird is mapped to its scientific order. In *birdcom* the species name is the common name only (*e.g.*, "American Robin"), and in *birdsci* the species names is the common name concatenated with the scientific name (*e.g.*, "American Robin—Turdus migratorius"). These three classification problems are associated with applications of WHIRL (Cohen, 1998), a database system that answers queries using information extracted from a set of web pages. Hence, each dataset is also associated with a collection of data-rich web pages. For the sake of reproducibility, we experimented with certain well-defined subcollections of these web pages, which are described elsewhere (Cohen & Fan, 1999; Cohen, 1999) and are available on the web.[2] Note that unlike the pages used for *music*, these page collections were assembled in large part manually.

In each benchmark problem, the initial representation for an instance is just the name of the instance, represented as a "bag of words" (or more precisely, as a set of stems produced by the Porter (1980) stemming algorithm). One can view this representation as a set of binary features, each feature f_t being "true" iff the term t appears in the instance's name. For example, the instance named "American Robin" might have the features f_{america} and f_{robin} true, and all other features false. The first columns of Table 1 summarize the four benchmark problems, listing for each problem the number of examples, classes, features, and associated web pages, and the total size of all web pages in megabytes.

2.2 Expansion Method

The method used for adding features to examples is motivated by a semi-automatic wrapper generation procedure, which is described elsewhere (Cohen, 1999). The expansion method takes as input a set of HTML pages $\mathcal{P}$, and a set of instances $\mathcal{X}$. (We will assume that $\mathcal{X}$ includes both the training and test instances, so the approach described here is really a method for *transduction* (Vapnik, 1998) rather than induction.) The result of the expansion process is to define a number of new features $g_1(x), \ldots, g_n(x)$ over the instances $x \in \mathcal{X}$.

The expansion method procedes as follows. First a set of pairs E is initialized to the empty set. Then, for each page $p \in \mathcal{P}$, the following steps are taken.

1. The HTML markup for p is parsed, generating an HTML parse tree T_p.[3] Each node of this parse tree corresponds either to an HTML element in p, or a string of text appearing in p. Note that the order of elements in p imposes an ordering $\prec$ on the HTML nodes of T_p. We use $text(n)$ to denote the concatenation (in order) of all strings appearing below the node n in T_p—that is, the text marked up by the HTML element corresponding to n. We use $tag(n)$ to denote the tag of the HTML element corresponding to n.

   Table 2 shows an example HTML page p and the

[1] In the terminology of (Cohen & Fan, 2000), the set of pages used here is a random 10% sample of all pages associated with non-empty "pseudo-users" that were collected by the "CF spider". Genre information was derived from the top-level categories that formed the "genre" pseudo-users, and artists associated with multiple genres were discarded. The web pages from which genre information was extracted are (of course) disjoint from the web pages used in the experiments.

[2] In http://www.isi.edu/~muslea/RISE/, a repository of extraction problems.

[3] Parsing HTML is actually rather difficult; because browers are quite forgiving of badly-formed HTML, many web pages are not syntactically correct. We used the HTML parser provided in the Perl *libwww* package for this step.

Table 1. Benchmark problems used in the experiments.

	#example	#class	#terms	#pages	(Mb)	#features added	%examples expanded
music	1010	20	1600	217	(11.7)	1890	68.7
games	791	6	1133	177	(2.5)	1169	53.0
birdcom	915	22	674	83	(2.2)	918	99.9
birdsci	915	22	1738	83	(2.2)	533	99.9

Table 2. A simple HTML page and the corresponding parse tree.

Sample HTML page p:

```
<html><head>...</head>
<body>
<h1>Editorial Board Members</h1>
<table> <tr>
    <td>Harry Q. Bovik, Cranberry U
    <td>G. R. Emlin, Lucent
  </tr><tr>
    <td>Bat Gangley, UC/Bovine
    <td>Pheobe L. Mind, Lough Tech
...
```

Parse tree T_p:

```
html(head(...),
       body(
n1:        h1("Editorial Board Members"),
           table(
               tr(td("Harry Q. Bovik, Cranberry U"),
n2:               td("G.R. Emlin, Lucent")),
               tr(td("Bat Gangley, UC/Bovine"),
                  td("Pheobe L. Mind, Lough Tech")),
               ...
```

corresponding parse tree T_p. The tree is shown in a functional notation, where the tag of a node n becomes the functor of a logical term, and the subtrees of n become the arguments.

2. For each node n that has K_{split} or more children corresponding to line-break (
) elements, new child nodes are introduced with the artificial tag *line* and with child nodes corresponding to elements between the
 elements. For example, the HTML

```
<strong>bright red<br>
            orange<br>
            ...
            violet
</strong>
```

would result in this tree, again using functional notation:

```
strong(line("bright red"),br(),line("orange"),br(),
        ...,line("violet"))
```

Conceptually, this operation groups items on the same line together in the tree T_p under a *line* node.

3. The *scope* of each header element in T_p is computed. The scope of a header is all HTML elements that appear to be below that header when the document is formatted.

Specifically, let n be a parse-tree node corresponding to a header element like <h1>,<h2>, etc. The *level* for n is defined to be 1 if n's HTML tag is <h1>, 2 if n's HTML tag is <h2>, and so on. Also, <title> nodes are considered to be headers, with a level of 0. If n is a header node at level i, the *scope of* n is defined to be all nodes n' such that $n \prec n' \prec n^*$, where n^* is the first header node such that $n \prec n^*$ and the level of n^* is not more than the level of n. (If no such n^* exists, then the scope is all nodes $n' : n \prec n'$.)

4. For each node $n \in T_p$ such that $|text(n)| < K_{text}$, each header node n_h such that n is in the scope of n_h, and each word w in $text(n_h)$, add the pair $(text(n), w)$ to $\mathcal{E}$. For example, in Table 2, the node n_2 is in the scope of n_1, so the pairs added to $\mathcal{E}$ would include ("G. R. Emlin, Lucent", "Editorial"), ("G. R. Emlin, Lucent", "Board"), and ("G. R. Emlin, Lucent", "Members"). Many less sensible pairs would also be added to $\mathcal{E}$, such as ("G. R. Emlin, Lucent Harry Q. Bovik, Cranberry U", "editorial").

5. For each node $n \in T_p$ such that $|text(n)| < K_{text}$, add to $\mathcal{E}$ the pair $(text(n), position(n))$. Here $position(n)$ is the string "$u(p)tag(a_0)\ldots tag(a_l)$" where $u(p)$ is the URL at which the page p was found, and $a_0 \ldots a_l$ are the nodes encountered in traversing the path from the root of T_p to n (inclusive).

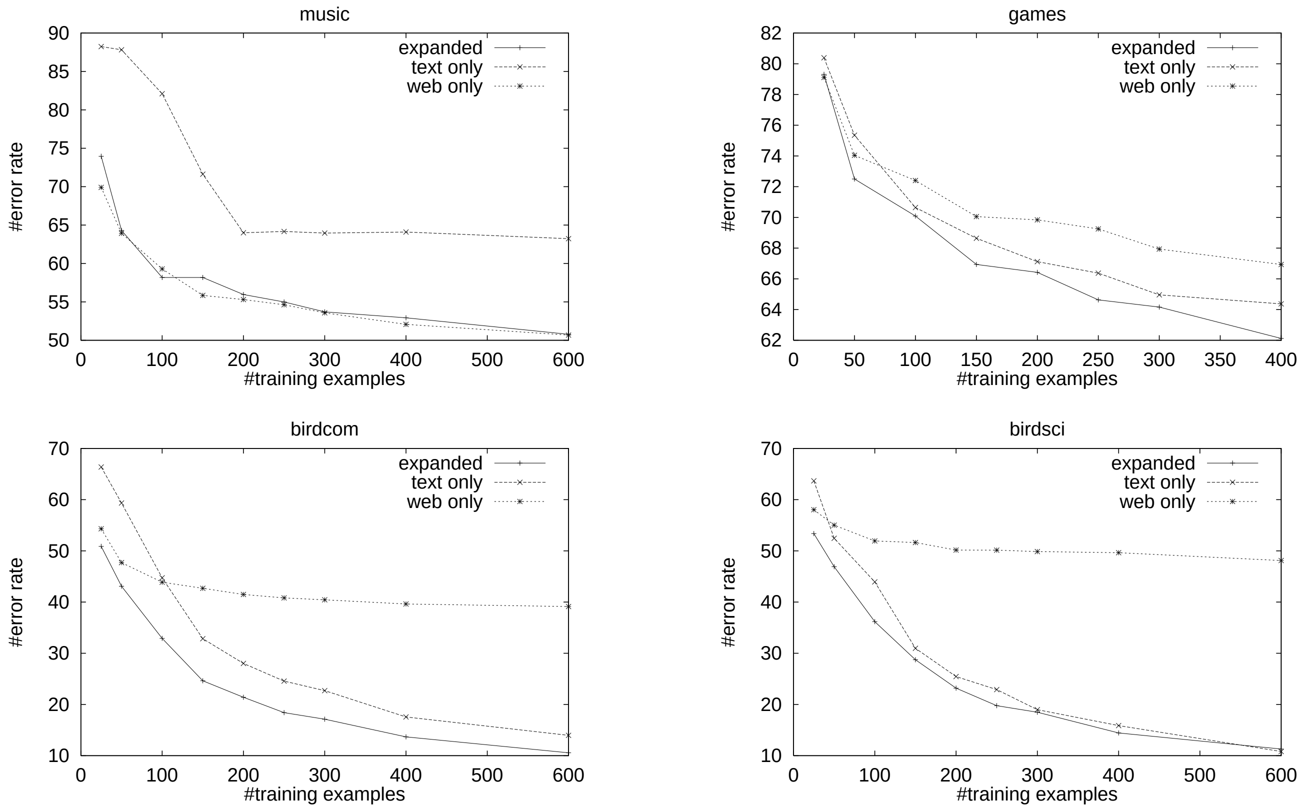

Figure 1. Error rate of RIPPER on the four benchmark problems as training set size is varied.

Again using Table 2 as an example, assume that u is the URL for p, and s is the string *html_body_table_tr_td*. Then this step would add to $\mathcal{E}$ pairs like ("G. R. Emlin, Lucent", us) and ("Bat Gangley, UC/Bovine", us). As before, this step would also add many less sensible pairs as well, such as ("Editorial Board Members",us'), where $s' =$ *html_body_h1*).

Finally, $\mathcal{E}$ is used to define a new set of features as follows. Let $sim(s,t)$ be the cosine similarity (Salton, 1989) of the strings s and t.[4] Let $\mathcal{T}$ be the set of positions and/or header words appearing in $\mathcal{E}$: that is, $\mathcal{T} = \{t : (y,t) \in \mathcal{E}\}$. For each $t \in \mathcal{T}$ a new feature g_t is defined as follows:

$$g_t(x) = 1 \text{ iff } \exists (y,t) \in \mathcal{E} : sim(name(x), y) \geq K_{sim}$$

Here $name(x)$ is the natural-language name for x. For example, if x is an instance with $name(x)$ ="G. R. Emlin", then the pairs computed from the sample page might lead to defining $g_{editorial}(x) = 1$, $g_{board}(x) = 1$, and $g_{us} = 1$.

[4]We follow the implementation used in WHIRL (Cohen, 1998).

Note that $g_t(x)$ will be true for x only if $name(x)$ is highly similar to some string y in a pair $(y,t) \in \mathcal{E}$. Thus if x's name (or some approximation of it) does not appear on any page p, the expansion process generates no features that are true for x, and hence no useful information about x. The final two columns of Table 1 indicate the number of new features created by this process for each of the four benchmark problems, and the percentage of examples x for which some generated feature is true. We used $K_{split} = 3$, $K_{text} = 40$, and $K_{sim} = 0.8$.

3. Experimental Results

3.1 Initial Experiments

The term-based features used in the initial representation are "sparse" binary features (that is, for most instances, only a few features are true), as are the features created by the expansion process. In our initial experiments, we used the rule-learning system RIPPER (Cohen, 1995), which handles sparse binary features efficiently (Cohen, 1996). We used RIPPER with the `-aunordered` option, which our experience sug-

Table 3. The expanded representation *vs* the text-only representation with 200 training examples.

		avg error		avg #rules	
	W-L-T	expand	text	expand	text
music	86-0-14	51.5	58.3	59.7	107.5
cdroms	29-7-64	65.8	67.2	21.4	31.9
birdcom	77-2-21	21.2	27.7	52.5	86.5
birdsci	35-8-57	23.6	26.4	61.2	80.0

gests is better for problems with many classes.[5]

Figure 1 shows the result of running RIPPER on the four problems. We used various sized training sets, testing on the remaining data, and averaged over 20 trials. Three representations were used for each dataset: the original representation, labeled *text only* in the figure; a representation including only the features g_t generated by the expansion process, labeled *web only*; and the union of all features, labeled *expanded*. To summarize, the generated features alone are usually less useful than the original features (the exceptions being *music*, or when very few examples are available). However, the generated features are useful when combined with the original features: average error rates are lower with the expanded representation than with the text-only representation.

To assess the statistical significance of the improvement in error rate, we trained RIPPER on 200 examples and tested on the remainder, using McNemar's test to determine if generalization performance was significantly different between the expanded representation and the text-only representation. McNemar's test makes minimal independence assumptions (Dietterich, 1998); however, it does depend on the training sample. We thus repeated this procedure 100 times for different randomly-chosen training sets, and recorded the number of *wins* for the expanded representation: that is, the number of times that McNemar's test indicated that, with a 95% or higher confidence, the expanded representation is superior to the text-only representation. *Losses* are defined analogously, and a *tie* indicates that neither representation was significantly better with 95% confidence.

Table 3 shows the won-loss-tied record on each of the four benchmark problems, and also the average error rate and the number of rules associated with each representation. To summarize, the expansion process frequently leads to a statistically significantly lower error rate, and seldom leads to a significantly higher error rate. The classifiers learned by RIPPER are also more compact with the expanded representation.

[5]This option causes RIPPER to learn a separate ruleset for each class. An instance x is then classified into the class associated with the "best" rule that fires on x, where rule quality is measured by the Bayes-Laplace corrected error on the training examples.

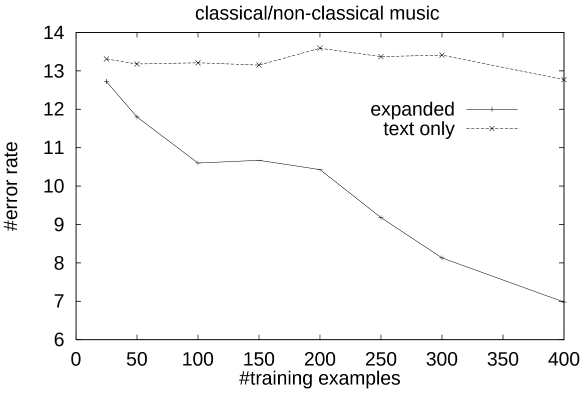

Figure 2. A two-class problem for which expansion provides a dramatic benefit.

The reduction in average error associated with the expanded representation ranges from 25% (on *birdcom*) to 2% (on *games*). We note that on these problems, the possible improvement is limited by many factors: in the *bird** benchmarks, the initial term-based representation is already quite informative; in the *games* and *music* benchmarks, many examples are not usefully expanded; and in all benchmarks, the large number of classes leads to a "small disjunct problem" (Holte et al., 1989) which limits the learning rate. Figure 2 shows the learning curve for a version of the *music* problem where the only classes are *classical* and *non-classical*, and where instances not represented in the set of web pages were discarded. For this problem the reduction in error rate is a more dramatic 50%.

3.2 Different Expansion Methods

The expansion method we propose adds two types of features: *header-word features*, which correspond to words in an HTML header element that appear to modify an example x, and *positional features*, corresponding to a place in an HTML document in which an instance x is named. We also tested several variants of this method, and compared them to the baseline expansion method described above. The *position only* and *header only* variants generate only one of the two types of features. The *coarse-position* variant uses header-word features plus a weaker sort of position feature, in which the "position" of an HTML parse-tree node n is simply the URL of the page containing n. Similarly, the *fine-position* variant uses header-word

Table 4. Left, the won-loss-tie record for the baseline expansion method *vs* four variants. Right, the won-loss-tie record *vs* the text-only representation for the baseline expansion method and two variants.

	baseline *vs* variants				variant *vs* text-only		
	posn	header	finepos	coarsepos	baseline	posn	header
music	92-0-8	81-0-19	89-0-11	85-0-15	86-0-14	77-0-23	78-0-22
games	29-4-67	32-1-67	26-5-69	28-3-69	29-7-64	11-13-76	42-7-51
birdcom	82-2-16	80-1-19	84-2-14	80-1-19	77-2-21	67-6-27	90-1-9
birdsci	39-5-56	44-6-50	42-5-53	33-10-57	35-8-57	37-10-53	45-7-48

features with a different, finer-grained notion of position (details omitted for brevity).

To evaluate these variants, we repeated the iterated McNemar's tests used above, except that we compared the baseline expansion method to each variant. (Thus a "win" indicates a situation in which the baseline is significantly better than some variant.) We also compared the position-only and header-only variants to the text-only representation. The results are shown in Table 4. To summarize, it appears that the baseline expansion method generally outperforms each of the four variants; however, both positional and header-word features are often useful independently.

3.3 Different Sets of Web Pages

Only for *music* were the web pages used in expansion automatically collected; for *games* and *bird*∗, the web pages were collected by a manual procedure. Thus it is not clear how easy it is to collect sets of web pages that will be useful for feature generation.

In order to explore this issue, we implemented a simple scheme for automatically collecting web pages in the *birds* and *games* domains. We found one current web page from each domain that appeared to point to many relevant data-rich pages,[6] and then ran, for each domain, a spider that crawled blindly from the appropriate starting point. Assuming that the automatically spidered pages would be, on average, less useful than the manually chosen ones, we halted the spiders when 174 bird-related pages and 266 game-related pages had been collected—somewhat more than were available in the original sets of pages. We then repeated the experiment of Figure 1 with the new page sets.

The pages collected for the *game* domain were not very useful: only about 75 instances (less than 10% of the total) appeared in the pages, and the error rate of RIPPER was not improved.[7] However, in the bird domain,

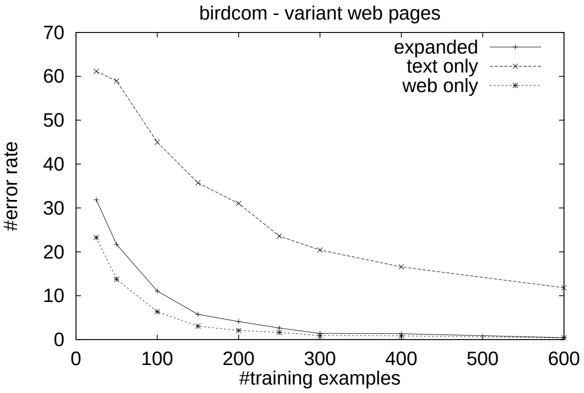

Figure 3. Performance on *birdcom* with automatically-collected web pages.

the results were excellent, as is shown in Figure 3. At 300 examples the error rate is reduced from 20% to less than 1.5%. The generated rulesets are also extremely compact and understandable.

3.4 Different Learning Systems

We also repeated the iterated McNemar's experiment for two other learning systems: C4.5 (Quinlan, 1994), a popular decision tree learner, and BoosTexter (Schapire & Singer, in press), a text classifier based on boosting simple decision rules.[8] The results are shown in Table 5. To summarize, the results are qualitatively the same as for RIPPER: expansion often leads to significantly lower error rates, and seldom leads to a significantly higher error rate. The lowest average error rates (shown in boldface) are always associated with an expanded representation.

[6] For birds, http://dmoz.org/Recreation/Birdwatching-/North_America/Checklists/ was the initial page, and, for *games*, http://www.childrenssoftware.com/links.html.

[7] The problem is probably that the entities concerned are no longer represented well on the web. The training set was assembled in mid 1997, and the experiments of this paper were conducted in early 2000. Few games are popular over this long a timespan. The manually-chosen collection of *games* web pages was assembled in early 1998.

[8] BoosTexter was used with a binary representation and run for 1000 rounds. For efficiency, we removed features that were true for two or fewer examples before running C4.5.

Table 5. Error rate of C4.5 and BoosTexter on the benchmark problems.

	RIPPER			C4.5			BoosTexter		
		avg %error			avg %error			avg %error	
	W-L-T	expand	text	W-L-T	expand	text	W-L-T	expand	text
music	86-0-14	51.5	58.3	100-0-0	49.3	59.6	100-0-0	**43.4**	58.1
games	29-7-64	65.8	67.2	13-6-81	68.2	68.6	16-1-83	**61.8**	63.1
birdcom	77-2-21	**21.2**	27.7	97-0-3	31.8	40.7	65-1-34	21.3	25.3
birdsci	35-8-57	**23.6**	26.4	46-4-50	37.3	39.0	35-6-59	25.4	26.7

4. Related Work

In this paper we have described a means for improving the performance of off-the-shelf supervised concept learners by constructing new features derived from web pages. More specifically, new features for an instance x are derived by looking for occurances of marked-up substrings similar to x in a set of pre-identified web pages. New features are then generated which pertain to these occurances.

There has been much prior work in "constructive induction", but this usually involves constructing new features by combining old ones (*e.g.*, (Pagallo & Haussler, 1990; Matheus & Rendell, 1989)) or by exploiting domain knowledge (*e.g.*, (Donoho & Rendell, 1995)). Here new features are found by exploiting unlabeled web pages from the same domain. There has also been prior work on learning methods that use unlabeled examples as well as labeled ones (*e.g.*, (Nigam et al., 1998)); in this paper, however, the additional input to the learning system is not a set of unlabeled instances, but a set of documents that may mention the labeled instances.

This paper is closely related to previous work of Collins and Singer (1999), who also consider constructing features based on occurances of labeled instances. However, in their experiments, instance occurances are found in free text, not in structured documents, and the constructed features are based on a natural-language parse of the text around an reference to an instance. Collins and Singer demonstrate that the extracted features can be exploited by a system that uses "co-training" (Blum & Mitchell, 1998) to exploit the new features.

This paper extends the results of Collins and Singer by showing the utility of features extracted from structured HTML documents, rather than parsed free text. We also show that conventional concept-learning methods can make use of these extracted features. We hope to conduct a more detailed comparision of this method with the Collins and Singer method in the future.

In another related paper, Cohen and Fan (2000) use similar heuristics to construct "pseudo-users" for a collaborative filtering system by analyzing web pages. Although "pseudo-users" are broadly similar to the features constructed by the expansion process, the details of feature construction are different (for instance, there is no analog of header-word features). We also note that in the experiments described by Cohen and Fan, adding pseudo-users to an existing dataset seldom improves collaborative-filtering performance. This contrasts with the experiments performed here with concept learning, in which performance is almost always improved by the new features.

5. Concluding Remarks

We have described a means for improving the performance of off-the-shelf supervised concept learners by constructing new features derived from web pages. More specifically, new features are derived by analyzing a set of unlabeled web pages, and looking for marked-up substrings similar to the name of some labeled instance x. New features for x are then generated, based on either header words that appear to modify this substring, or the position in the HTML page at which the substring appears. These new features improve performance on several benchmark problems in a wide variety of situations. Performance improvements are sometimes dramatic: on one problem, the error rate is decreased by a factor of ten, and on another, by half.

Acknowledgements

I wish to thank Michael Collins and Steve Abney for numerous helpful discussions on this topic.

References

Ashish, N., & Knoblock, C. (1997). Wrapper generation for semistructured Internet sources. *Proceedings of the Workshop on Management of Semistructured Data*. Tucson, Arizona: Available on-line from http://www.research.att.com/~suciu/workshop-papers.html.

Basu, C., Hirsh, H., & Cohen, W. (1998). Recommendation as classification: Combining social and content-based information in recommendation. *Proceedings of the Fifteenth National Conference on Artificial Intelligence*. Madison, WI.

Blum, A., & Mitchell, T. (1998). Combining labeled and unlabeled data with co-training. *Proceedings of the 1998 Conference on Computational Learning Theory*. Madison, WI.

Cohen, W. W. (1995). Fast effective rule induction. *Machine Learning: Proceedings of the Twelfth International Conference*. Lake Tahoe, California: Morgan Kaufmann.

Cohen, W. W. (1996). Learning with set-valued features. *Proceedings of the Thirteenth National Conference on Artificial Intelligence*. Portland, Oregon.

Cohen, W. W. (1998). Integration of heterogeneous databases without common domains using queries based on textual similarity. *Proceedings of ACM SIGMOD-98*. Seattle, WA.

Cohen, W. W. (1999). Recognizing structure in web pages using similarity queries. *Proceedings of the Sixteenth National Conference on Artificial Intelligence*. Orlando, FL.

Cohen, W. W., & Fan, W. (1999). Learning page-independent heuristics for extracting data from web pages. *Proceedings of The Eigth International World Wide Web Conference*. Toronto.

Cohen, W. W., & Fan, W. (2000). Web-collaborative filtering: Recommending music by crawling the web. *Proceedings of The Ninth International World Wide Web Conference*. Amsterdam.

Cohen, W. W., & Hirsh, H. (1998). Joins that generalize: Text categorization using WHIRL. *Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining* (pp. 169–173). New York, NY.

Collins, M., & Singer, Y. (1999). Unsupervised models for named entity classification. *Proceedings of the Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora*. College Park, MD.

Dietterich, T. G. (1998). Approximate statistical tests for comparing supervised classification learning algorithms. *Neural Computation, 10(7)*.

Donoho, S., & Rendell, L. (1995). Representing and restructuring domain theories: A constructive induction approach. *Journal of Artificial Intelligence Research*, *2*, 411–446.

Hammer, J., Garcia-Molina, H., Cho, J., & Crespo, A. (1997). Extracting semistructured information from the Web. *Proceedings of the Workshop on Management of Semistructured Data*. Tucson, Arizona: Available on-line from http://www.research.att.com/~suciu/workshop-papers.html.

Holte, R., Acker, L., & Porter, B. (1989). Concept learning and the problem of small disjuncts. *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*. Detroit, Michigan: Morgan Kaufmann.

Kushmerick, N., Weld, D. S., & Doorenbos, R. (1997). Wrapper induction for information extraction. *Proceedings of the 15th International Joint Conference on Artificial Intelligence*. Osaka, Japan.

Matheus, C. J., & Rendell, L. A. (1989). Constructive induction on decision trees. *Proceedings of the Eighth International Workshop on Machine Learning*. Evanston, Illinois: Morgan Kaufmann.

Muslea, I., Minton, S., & Knoblock, C. (1998). Wrapper induction for semistructured, web-based information sources. *Proceedings of the Conference on Automated Learning and Discovery*.

Nigam, K., McCallum, A., Thrun, S., & Mitchell, T. (1998). Learning to classify text from labeled and unlabeled documents. *Proceedings of the Fifteenth National Conference on Artificial Intelligence*. Madison, WI.

Pagallo, G., & Haussler, D. (1990). Boolean feature discovery in empirical learning. *Machine Learning, 5(1)*.

Porter, M. F. (1980). An algorithm for suffix stripping. *Program*, *14*, 130–137.

Quinlan, J. R. (1994). *C4.5: Programs for machine learning*. Morgan Kaufmann.

Salton, G. (Ed.). (1989). *Automatic text processing*. Reading, Massachusetts: Addison Welsley.

Schapire, R. E., & Singer, Y. (in press). BoosTexter: A boosting-based system for text categorization. To appear in *Machine Learning*.

Vapnik, V. (1998). *Statistical learning theory*. New York: Wiley and Sons.