Extracting Web Data Using Instance-Based Learning

Yanhong Zhai and Bing Liu

Department of Computer Science
University of Illinois at Chicago
851 S. Morgan Street, Chicago, IL 60607
yzhai,liub@cs.uic.edu

Abstract. This paper studies structured data extraction from Web pages, e.g., online product description pages. Existing approaches to data extraction include wrapper induction and automatic methods. In this paper, we propose an instance-based learning method, which performs extraction by comparing each new instance (or page) to be extracted with labeled instances (or pages). The key advantage of our method is that it does not need an initial set of labeled pages to learn extraction rules as in wrapper induction. Instead, the algorithm is able to start extraction from a single labeled instance (or page). Only when a new page cannot be extracted does the page need labeling. This avoids unnecessary page labeling, which solves a major problem with inductive learning (or wrapper induction), i.e., the set of labeled pages may not be representative of all other pages. The instance-based approach is very natural because structured data on the Web usually follow some fixed templates and pages of the same template usually can be extracted using a single page instance of the template. The key issue is the similarity or distance measure. Traditional measures based on the Euclidean distance or text similarity are not easily applicable in this context because items to be extracted from different pages can be entirely different. This paper proposes a novel similarity measure for the purpose, which is suitable for templated Web pages. Experimental results with product data extraction from 1200 pages in 24 diverse Web sites show that the approach is surprisingly effective. It outperforms the state-of-the-art existing systems significantly.

1 INTRODUCTION

Web data extraction is the problem of identifying and extracting target items from Web pages. It is important in practice because it allows one to integrate information or data from multiple sources (Web sites and pages) to provide value-added services, e.g., customizable Web information gathering, comparative shopping, meta-search, etc.

In this paper, we focus on regularly structured data which are produced by computer programs following some fixed templates. The contents of these data records are usually retrieved from backend databases, and then converted to

HTML documents by programs and published on the Web. One such example is the product description pages. Each merchant who sells products on the Web needs to provide a detailed description of each product. Fig.1 shows an example page. One may want to extract four pieces of information from this page, product name, product image, product description, and product price, for comparative shopping. Note that each of the items is marked with a dash-lined box in Fig.1. We call each required piece of information a *target item* (or simply *item*).

Fig. 1. An example product description page

Existing research on Web data extraction has produced a number of techniques ([1–10]). The current dominate technique is wrapper induction based on inductive machine learning. In this approach, the user first labels or marks the target items in a set of training pages or a list of data records in one page. The system then learns extraction rules from these training pages. The learned rules are then applied to extract target items from other pages. An extraction rule for a target item usually contains two patterns [11, 8]: a prefix pattern for detecting the beginning of a target item, and a suffix pattern for detecting the ending of a target item. Although there are also automatic approaches to extraction based on pattern finding, they are usually less accurate and also need manual post-processing to identify the items of interest. In Sec.2, we discuss these and other existing approaches further.

A major problem with inductive learning is that the initial set of labeled training pages may not be fully representative of the templates of all other pages. For pages that follow templates not covered by the labeled pages, learnt rules will perform poorly. The usual solution to this problem is to label more pages because more pages should cover more templates. However, manual labeling is labor intensive and time consuming (still no guarantee that all possible templates

will be covered). For a company that is interested in extracting all product information from most (if not all) merchant sites on the Web for comparative shopping, this represents a substantial work. Although active learning helps [12], it needs sophisticated mechanisms.

In this paper, we propose an instance-based learning approach to data extraction that is able to deal with this problem effectively. In classic instance-based learning, a set of labeled instances (more than 1) is stored first (no induction learning is performed). When a new instance is presented, it is compared with the stored instances to produce the results. The approach is commonly used in classification. The most popular instance-based learning methods are *k-nearest neighbor* and *case-based reasoning* [13]. However, we cannot directly apply these classic approaches because we will still need an initial set of many labeled instances, and thus will have the same problem as inductive learning.

We propose a different instance-based method that is more suitable to data extraction from Web pages. It does not need an initial set of labeled pages. Instead, the algorithm can begin extraction from a single labeled page. Only when a new page cannot be extracted does the page need labeling. This avoids unnecessary labeling and also ensures that all different templates are covered.

We believe that instance-based learning is very suitable for structured data extraction because such Web data are presented by following some fixed layout templates (see Fig.1). Pages from the same template can be extracted using a single page instance of the template.

The key to our instance based learning method is the similarity or distance measure. In our context, it is the problem of how to measure the similarity between the corresponding target items in a labeled page and a new page. Traditional measures based on the Euclidean distance or text similarity are not easily applicable in this context because target items from different pages can be entirely different. We propose a natural measure that exploits the HTML tag context of the items. Instead of comparing items themselves, we compare the tag strings before and after each target item to determine the extraction. This method is appropriate for templated pages because a template is essentially reflected by its sequence of formatting tags. Our technique works as follows:

1. A random page is selected for labeling.
2. The user labels/marks the items of interest in the page.
3. A sequence of consecutive tags (also called *tokens* later) before each labeled item (called the *prefix string* of the item) and a sequence of consecutive tags after the labeled item (called the *suffix string* of the item) are stored.
4. The system then starts to extract items from new pages. For a new page d, the system compares the stored prefix and suffix strings with the tag stream of page d to extract each item (this step is involved and will be clear later). If some target items from d cannot be identified (i.e., this page may follow a different template), page d is passed to step 2 for labeling.

We have implemented an extraction system, called IDE (*Instance-based Data Extraction*) based on the proposed approach. The system has been tested using 1200 product pages from 24 Web sites. Our results show that IDE is highly

effective. Out of the 1200 pages, only 6 pages were not extracted correctly, and only one of the items in each page was extracted incorrectly. For most Web sites, the user only needs to label 2-3 pages. We also compared IDE with the FETCH [14] system, which is the commercial version of the state-of-the-art research system Stalker [11, 8, 15, 12]. Our results show that IDE outperforms FETCH in our experiments. Our proposed approach is also efficient.

2 RELATED WORK

The closely related works to ours are in the area of wrapper generation. A wrapper is a program that extracts data items from a Web site/page and put them in a database. There are two main approaches to wrapper generation. The first approach is wrapper induction (or learning), which is the main technique presently. The second approach is automatic extraction.

As mentioned earlier, wrapper learning works as follows: The user first manually labels a set of training pages or data records in a list. A learning system then generates rules from the training pages. These rules can then be applied to extract target items from new pages. Example wrapper induction systems include WIEN [6], Softmealy [5], Stalker [8, 15, 12], BWI [3], WL^2 [1], and etc [9]. A theoretical study on wrapper learning is also done in [16]. It gives a family of PAC-learnable wrapper classes and their induction algorithms and complexities.

WIEN [6] and Softmealy [5] are earlier wrapper learning systems, which were improved by Stalker [11, 8, 15, 12]. Stalker learns rules for each item and uses more expressive representation of rules. It does not consider ordering of items but treat them separately. This is more flexible but also makes learning harder for complex pages because local information is not fully exploited. Recent research on Stalker has added various active learning capabilities to the system to reduce the number of pages to be labeled by the user. The idea of active learning is to let the system select the most useful pages to be labeled by the user and thus reduces some manual effort.

Existing systems essentially learn extraction rules. The rules are then used directly to extract each item in new pages. Our work is different. Our technique does not perform inductive learning. Instead, it uses an instance-based approach. It can start extraction from a single labeled page. Although [17] can learn from one page (two for single-record pages), it requires more manual work because if the system does not perform well the user who monitors the system needs to change some system thresholds.

In recent years, researchers also studied automatic extraction, i.e., no user labeling is involved. [18] proposes a method for finding repetitive patterns from a Web page, and then uses the patterns to extract items from each object in the page. [19] shows that this technique performs unsatisfactory in extraction. [20, 21, 7] propose two other automatic extraction methods. However, these automatic methods are less accurate than the systems that ask the user to label training pages. Manual post-processing is also needed for the user to identify what he/she is interested in. In [10], a more accurate technique is proposed

based on tree matching. However, it is only for list pages (each page contains multiple data records). [22, 19] propose some techniques for finding data objects or data records. However, they do not perform data extraction from the records.

Another related research is information extraction from text documents [23, 24, 2–4, 25, 26]. Our work is different as we mainly exploit structural information in a Web page for extraction, which requires different techniques. Finally, a number of toolkits to facilitate users to build wrappers are reported in [27–29].

3 INSTANCE-BASED EXTRACTION

We now present the proposed approach. As mentioned earlier, given a set of pages from a Web site, the proposed technique first (randomly) selects a page to ask the user to label the items that need to be extracted. The system then stores a certain number of consecutive prefix and suffix tokens (tags) of each item. After that, it starts to extract target items from each new page. During extraction, if the algorithm is unable to locate an item, this page is given to the user to label. This process goes on until all the pages from the given site have been processed. Below, Sec.3.1 presents the overall algorithm and the page labeling procedure. Sec.3.2 presents the similarity measure used for extraction. Sec.3.3 presents an efficient algorithm for implementation.

3.1 The Overall Algorithm

Let S be the set of pages from a Web site that the user wants to extract target items from. Let k be the number of tokens in the prefix or suffix string to be saved for each target item from a labeled page. In practice, we give k a large number, say 20. The setting of this value is not important because if it is too small, the system can always go back to the labeled page to get more tokens. Fig.2 gives the overall algorithm.

```
Algorithm(S,k)                    // S is the set of pages.
1. p = randomSelect(S);           // Randomly select a page p from S
2. Templates = <>;                // initialization
3. labelPage(Templates, p,k);     // the user labels the page p
4. for each remaining page d in S do
5.    if ¬(extract(Templates,d)) then
6.       labelPage(Templates,d,k)
7.    end-if
8. end-for
```

Fig. 2. The overall algorithm

In line 1, the algorithm randomly selects a page p from S. This page is given to the user for labeling (line 3). A user-interface has been implemented for the user to label target items easily. Variable *Templates* (line 2) stores the templates of all labeled pages so far. For example, in the page of Fig.1, we are

interested in extracting four items from a product page, namely, *name*, *image*, *description* and *price*. The template (T) for a labeled page is represented as follows: $T =< pat_{name}, pat_{img}, pat_{description}, pat_{price} >$

Each pat_i in T consists of a prefix string and a suffix string of the item i(also called the *prefix-suffix pattern* of i). For example, if the product image is embedded in the following HTML source:

...`<table><tr><td> <img> </td><td></td>`...

then we have:
$pat_{img}.prefix$ = `<<table ><tr><td>>` $pat_{img}.suffix$ = `<</td><td></td>>`
Here, we use $k = 3$ (in our experiments, we used $k = 20$, which is sufficient).

In this work, we treat each page to be labeled as a sequence of tokens. A token can be any HTML element, a HTML tag, a word, a punctuation mark, etc. Not all kinds of tokens before or after a target item will be saved as a part of the prefix or suffix string. All tags and at most one word or punctuation mark right before (or after) the target item are regarded as part of the prefix (or suffix) string. Basically, we mainly rely on HTML tags to locate each target item.

After page p is labeled by the user, the algorithm can start extraction from the rest of the pages (lines 4-8). The extraction procedure, extract() (line 5), extracts all the items from page *d*.

To label a page, the user marks the items to be extracted in the page. This procedure is given in Fig3. A user-interface makes this process very easy. Basically, only mouse clicks on each target item are needed.

A requirement for the first page p is that it must contain all target items. The reason for this requirement is that if there is one or more missing items in this page, the extraction system will not know that additional items are needed. Selecting such a page is not difficult as most pages have all target items. Another important issue is the handling of missing items (see below).

Procedure labelPage(*Templates, p, k*)
1. The user labels all the required items in page *p*;
2. *T* = <>; // initialization
3. **for** each required item *i* **do**
4. **if** item *i* does not exist in page *p* **then**
5. insert ∅ into *T* at the right end;
6. **else**
7. prefix = extract *k* prefix tokens before item *i* in *p*;
8. suffix = extract *k* suffix tokens after item *i* in *p*;
9. *T* = insert <*prefix, suffix*> into *T* at the right end;
10. **end - if**
11. **end - for**
12. **if** *p* has missing item(*s*) **then**
13. *Templates* = insert *T* into *Templates* at the end;
14. **else**
15. *Templates* = insert *T* into *Templates* before any template with missing item(*s*);
16. **end - if**
17. output all the labeled items in *p*;

Fig. 3. Labeling a page p

T is a new template that stores the prefix and suffix strings of every item in page p. In lines 12-13, if page p has missing items, we put T at the end of *Templates*, which stores all the templates of labeled pages. This is to ensure that it will not be used before any other template is used to extract a page.

3.2 The Similarity Measure

The key to instance-based learning is the similarity or distance measure. In our context, it is the problem of measuring whether an item in the new page (to be extracted) is similar to or is of the same *type* as a target item in a labeled page. As indicated earlier, we do not compare the items themselves. Instead, we compare their prefix and suffix strings. The score is the number of marches.

Definition 1. *(prefix match score): Let* $P =< p_1, ..., p_k >$ *be the prefix string of an item in a labeled page and* $\mathcal{A}$ *be the token string of page d (to be extracted). A sub-string of* $\mathcal{A}(=< a_1, ...a_i, a_{i+1}, ..., a_{i+h}, ..., a_n >)$ *matches P with a match score of* $h(h \leq k)$ *, if* $p_k = a_{i+h}, p_{k-1} = a_{i+h-1}, ..., p_{k-h-1} = a_{i+1}$*, and* $(p_{k-h} \neq a_i$ *or* $h = k$ *)*

Definition 2. *(suffix match score): Let* $P =< p_1, ..., p_k >$ *be the suffix string of an item in a labeled page and* $\mathcal{A}$ *be the token string of page d (to be extracted). A sub-string of* $\mathcal{A}(=< a_1, ...a_i, a_{i+1}, ..., a_{i+h}, ..., a_n >)$ *matches P with a match score of* $h(h \leq k)$ *, if* $p_1 = a_{i+1}, p_2 = a_{i+2}, ..., p_h = a_{i+h}$*, and* $(p_{h+1} \neq a_{i+h+1}$ *or* $h = k$ *)*

Note that the match starts from the right for the prefix match, and the left for the suffix match. Fig.4 shows an example. Assume that we saved 5 tokens `<table><tr><td><i><b>` in the prefix string of item *price* from a labeled page. The HTML source of a new page d to be extracted is shown in the box of the figure. From the figure, we see that there are 4 sub-strings in d that have matches with the prefix string of *price*. These are shown in four rows below the prefix string. The number within () is the sequence id of the token in page d. "-" means no match. The highest number of matches is 5, which is the best *match score* for this prefix string. The best score can also be computed for the suffix string.

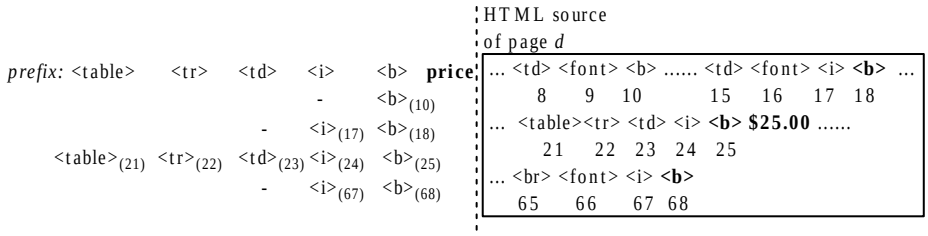

Fig. 4. Example 1 - Prefix matching

3.3 The Extraction Algorithm

We now discuss the extraction algorithm based on *match scores*. Note that in this paper, we are interested in extracting target items from pages that focus on a single object per page, not a list of objects per page due to one of our practical applications. A similar algorithm can be designed for list pages (we plan to do this in our future work).

The basic idea of the algorithm is as follows: For a new page d to be extracted, we try to use each labeled page (represented as a template of prefix and suffix strings) to extract the required items from d. Using the prefix and suffix strings of each item i, we can compute the prefix and suffix match scores of every item in page d. If a particular item j in d has a unique best match score (≥ 1) for both the prefix and suffix strings of a target item i in a template, item j is regarded as i's corresponding item in d and is extracted. After item j is extracted from d, we use the token strings of d before j and after j to identify and extract the remaining items. This process continues recursively until all items are extracted or an item cannot be extracted (which indicates that page d needs labeling). The detailed algorithm is more involved for efficiency reasons. This algorithm has the following characteristics:

1. In determining which item to extract, the system does not choose the item with the highest (prefix or suffix) match score among all items. Instead, it chooses the item with the unique best match score for the item in d.
2. There is no pre-specified sequence of items to be extracted. For example, the user is interested in 4 items from each page. The ordering of items in the HTML source is: *image*, *name*, *price*, and *description*. If at the beginning we are able to find item *price* uniquely in the page, we then start from price and search forward to find item *description* and search backward to find item *image* and *name*. In searching for the remaining items, the same approach is used. The final extraction sequence of items may be the one in Fig.5.

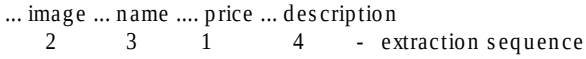

Fig. 5. An example item extraction sequence

   This method has a major advantage. That is, we can exploit local contexts. It may be the case that from the whole page we are unable to identify a particular item. However, within a local area, it is easy to identify it.
3. Ordering of items is exploited in extraction as shown above.

Fig.6 gives the extraction control procedure, which basically tries each saved template T in *Templates*. T contains the prefix and suffix strings of each item in a previously labeled page. d is the page to be extracted. If using a template T, all the items can be extracted with the procedure $extractItems()$ (line 2), it returns true (line 4) to indicate that page d is successfully extracted. If none of

the template in *Templates* can be used to extract page d, the procedure returns false to indicate that page d cannot be extracted using any previous T. "1" is the sequence id number of the first token of page d, and *end_id* is the sequence id of the last token of page d. These two id's tell extractItems() where to locate the target items. Note that each token has a sequence id, which enables the algorithm to find every token quickly.

```
Procedure extract(Templates, d)
1. for each template T in Templates do
2.    if extractItems(T, d, 1, end_id) then
3.       output the extracted items from d;
4.       return true
5.    end-if
6. end-for
7. return false;
```

Fig. 6. The extraction control procedure

The extractItems() procedure is given in Fig. 7. It takes 4 parameters, which have the same meanings as those in procedure extract(). *start* and *end* are the start and end token id's, which defines a region in d to look for target items.

```
Procedure extractItems(T, d, start, end)
1.  for each token t of d in sequence from start to end do
2.     for each pat_i in T do
3.        if pat_i ≠ ∅ and t and its predecessors match some prefix tokens in pati.prefix then
4.           record the string of id's of the matching tokens
5.        end-if
6.  end-for
7.  end-for
8.  if an item i's beginning can be uniquely identified then
9.     idB_i = the id of the immediate token on the right of the prefix string of item i;
10.    if idE_i = find the ending of item i between idB_i and end then
11.       extract and store item i;
12.       if before(T, i) is not empty then
13.          if (extractItems(before(T, i), d, start, idB_i)) then
14.             return false
15.          end-if
16.       end-if
17.       if after(T, i) is not empty then
18.          if (extractItems(after(T, i), d, idE_i, end)) then
19.             return false
20.          end-if
21.       end-if
22.    else return false
23.    end-if
24. elseif every target item may be missing then //indicated by ∅ in every element of T
25.    do nothing    // items are not in the page
26. else return false
27. end-if
28. return true
```

Fig. 7. The extractItems procedure

In line 1 of Fig.7, we scan the input stream of page d. From lines 2-6, we try the prefix string in pat_i of each item i to identify the beginning of item i. If token t and its predecessors match some tokens in the prefix string (line 3), we record the sequence id's of the matching tokens (line 4). $pat_i \neq \emptyset$ means that item i is not missing. Let us use an example in Fig.4 to illustrate this part. 5 tokens `<table><tr><td><i><b>` are saved in the prefix string of item *price* from a labeled page (assume we have only one labeled page). After going through lines 1 to 7 (Fig.7), i.e., scanning through the new page, we find four `<b>`'s, two `<i><b>` together, but only one `<table><tr><td><i><b>` together. We see that the beginning of *price* can be uniquely identified (which is done in line 8 of Fig. 7) because the longest match is unique.

The algorithm for finding the ending location of an item (line 10 in Fig. 7) is similar to finding the beginning. This process will not be discussed further.

After item i is extracted and stored (line 11), if there are still items to be extracted before or after item i (line 12 and 17), a recursive called is made to extractItems() (line 13 and 18). idB_i and idE_i are the sequence id's of the beginning and the ending tokens of item i.

Extraction failures are reported in lines 14, 19, 22 and 26. Lines 24 and 25 say that if all target items to be extracted from *start* to *end* may be missing, we do nothing (i.e., we accept that the page does not have these items).

The functions before() and after() obtain the prefix-suffix string patterns for items before item i and after item i respectively. For example, currently T contains prefix-suffix string patterns for items 1 to 4. If item 3 has just been extracted, then before(T) should give the saved prefix-suffix string patterns for items 1 and 2, and after(T) gives the saved prefix-suffix patterns of item 4.

Finally, we note that in lines 13 and 18 of Fig.7, both the *start* and *end* id's should extend further because the prefix and suffix strings of the next item could extend beyond the current item. We omit this detail in the algorithm to simplify the presentation.

4 EMPIRICAL EVALUATION

Based on the proposed approach, we built a data extraction system called IDE. We now evaluate IDE, and compare it with the state-of-the-art system FETCH [14], which is the commercial version of Stalker (the research version Stalker is not publicly available). Stalker improved the earlier systems such as WIEN, Softmealy, etc. The experimental results are given in Tab.1. Below, we first describe some experimental settings and then discuss the results.

Web sites used to build IDE: We used pages from 3 Web sites in building our system, i.e., designing algorithms and debugging the system. None of these sites is used in testing IDE.

Test Web sites and pages: 24 e-commerce Web sites are used in our experiments[1]. From each Web site, 50 product description pages are downloaded. All

[1] We did not use the archived data from the RISE repository (http://www.isi.edu/info-agents/RISE/) in our experiments because the data in RISE are mainly Web pages

the sites and pages are selected and downloaded by a MS student who is not involved in this project. See Tab.1 for our test Web sites.

From each product page, we extract the name, image, description and price for the product as they are important for many applications in Internet commerce, e.g., comparative shopping, product categorization and clustering.

Evaluation measures: We use the standard precision and recall measures to evaluate the results of each system.

Experiments: We performed two types of experiments on FETCH:

1. The training pages of FETCH are the pages being labeled by the user using IDE. These pages are likely to follow different templates and have distinctive features. Thus, they are the best pages for learning. However, they give FETCH an unfair boost because without IDE such pages will not be found. Tab.1 shows the results of FETCH and IDE in this setting.
2. Use the same number of training pages as used by IDE. However, the training pages are randomly selected (this is the common situation for inductive learning). In this case, FETCH's results are much worse (see Tab.2).

Tab.1 shows the results for experiment 1. Before discussing the results, we first explain the problem descriptions used in the table:

- miss: The page contains the target item, but it is not found.
- found-no: The page does not contain the target item, but the system finds one, which is wrong.
- wrong: The page has the target item, but a wrong one is found.
- partial err.: The page contains the target item, but the system finds only part of it (incomplete).
- page err.: It is the number of pages with extraction errors (any of the 4 types above).

We now summarize the results in Tab 1.

1. IDE is able to find all the correct items from every page of each Web site except for Web site 4. In site 4, IDE finds wrong product images in 6 pages. This problem is caused by irregular tags used before the image in many pages. This site also requires a high number of labeled pages which shows that this site has many irregularities. The FETCH system also made many mistakes in this site.
2. For 15 out of 24 Web sites, IDE only needs to label one or two pages and find all the items correctly.
3. We compute precision and recall in term of the number of items extracted. We also give a page accuracy value, which is computed based on the number of pages extracted correctly, i.e., every target item in these pages is extracted correctly.

---

that contain a list of objects per page. Our current technique is designed to extract data from pages that focus on a single object per page, which are important for comparative shopping applications.

Table 1. Experiment 1 results

	Site	No.of labeled pages	IDE					FETCH				
			miss	found-no	wrong	partial err.	page err.	miss	found-no	wrong	partial err.	page err.
1	alight	2	0	0	0	0	0	1	0	0	0	1
2	amazon	5	0	0	0	0	0	13	2	1	0	15
3	avenue	2	0	0	0	0	0	1	0	0	0	1
4	bargainoutfitters	8	0	0	6	0	6	0	0	9	0	9
5	circuitcity	2	0	0	0	0	0	1	0	0	3	4
6	computer4sure	4	0	0	0	0	0	4	0	0	0	4
7	computersurplusoutlet	1	0	0	0	0	0	0	0	0	0	0
8	dell	7	0	0	0	0	0	3	0	0	0	3
9	gap	2	0	0	0	0	0	0	0	0	1	1
10	hp	2	0	0	0	0	0	1	0	7	13	21
11	kmart	3	0	0	0	0	0	0	0	0	0	0
12	kohls	5	0	0	0	0	0	1	0	0	0	1
13	nike	1	0	0	0	0	0	0	0	0	0	0
14	officemax	3	0	0	0	0	0	24	0	7	9	38
15	oldnavy	2	0	0	0	0	0	0	0	0	5	5
16	paul	2	0	0	0	0	0	0	0	0	0	0
17	reebok	2	0	0	0	0	0	11	0	0	0	11
18	sony	5	0	0	0	0	0	0	0	1	7	8
19	shoebuy	1	0	0	0	0	0	5	0	0	26	31
20	shoes	2	0	0	0	0	0	0	0	0	1	1
21	staples	5	0	0	0	0	0	0	0	10	0	10
22	target	2	0	0	0	0	0	24	0	0	1	25
23	victoriasecret	2	0	0	0	0	0	0	0	0	14	14
24	walmart	2	0	0	0	0	0	9	0	0	0	9
	Total	72	0	0	6	0	6	98	2	25	80	212

	Recall	Precision	Page accuracy
IDE:	99.9%	99.9%	99.5%
FETCH:	95.7%	97.8%	82.5%

Tab. 2 summarizes the results of experiment 2 with FETCH, which is the normal use of FETCH. A set of pages is randomly selected and labeled (the same number of pages as above). They are then used to train FETCH. The recall value of FETCH drops significantly, and so does the page accuracy. IDE's results are copied from above.

Table 2. Experiment 2 results

	Recall	Precision	Page accuracy
IDE	99.9%	99.9%	99.5%
FETCH	85.9%	96.8%	46.0%

Time complexity: The proposed technique does not have the learning step as in FETCH and thus saves the learning time. The extraction step is also very efficient because the algorithm is only linear in the number of tokens in a page.

5 CONCLUSIONS

This paper proposed an instance-based learning approach to data extraction from structured Web pages. Unlike existing methods, the proposed method does

not perform inductive learning to generate extraction rules based on a set of user-labeled training pages. It thus does not commit itself pre-maturely. Our algorithm can start extraction from a single labeled page. Only when a new page cannot be extracted does the page need labeling. This avoids unnecessary page labeling, and thus solves a major problem with inductive learning, i.e., the set of labeled pages is not fully representative of all other pages. For the instance-based approach is to work, we proposed a novel similarity measure. Experimental results with product data extraction from 24 diverse Web sites show that the approach is highly effective.

6 ACKNOWLEDGMENTS

This work was partially supported by National Science Foundation (NSF) under the grant IIS-0307239. We would like to thank Steve Minton of Fetch Technologies for making the FETCH system available for our research.

References

1. Cohen, W., Hurst, M., Jensen, L.: A flexible learning system for wrapping tables and lists in html documents. In: The Eleventh International World Wide Web Conference WWW-2002. (2002)
2. Feldman, R., Aumann, Y., Finkelstein-Landau, M., Hurvitz, E., Regev, Y., Yaroshevich, A.: A comparative study of information extraction strategies. In: CICLing '02: Proceedings of the Third International Conference on Computational Linguistics and Intelligent Text Processing. (2002) 349–359
3. Freitag, D., Kushmerick, N.: Boosted wrapper induction. In: Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on Innovative Applications of Artificial Intelligence. (2000) 577–583
4. Freitag, D., McCallum, A.K.: Information extraction with hmms and shrinkage. In: Proceedings of the AAAI-99 Workshop on Machine Learning for Informatino Extraction. (1999)
5. Hsu, C.N., Dung, M.T.: Generating finite-state transducers for semi-structured data extraction from the web. Information Systems **23** (1998) 521–538
6. Kushmerick, N.: Wrapper induction for information extraction. PhD thesis (1997) Chairperson-Daniel S. Weld.
7. Lerman, K., Getoor, L., Minton, S., Knoblock, C.: Using the structure of web sites for automatic segmentation of tables. In: SIGMOD '04: Proceedings of the 2004 ACM SIGMOD international conference on Management of data. (2004) 119–130
8. Muslea, I., Minton, S., Knoblock, C.: A hierarchical approach to wrapper induction. In: AGENTS '99: Proceedings of the third annual conference on Autonomous Agents. (1999) 190–197
9. Pinto, D., McCallum, A., Wei, X., Croft, W.B.: Table extraction using conditional random fields. In: SIGIR '03: Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval. (2003) 235–242
10. Zhai, Y., Liu, B.: Web data extraction based on partial tree alignment. In: WWW '05: Proceedings of the 14th international conference on World Wide Web. (2005) 76–85

11. Knoblock, C.A., Lerman, K., Minton, S., Muslea, I.: Accurately and reliably extracting data from the web: a machine learning approach. (2003) 275–287
12. Muslea, I., Minton, S., Knoblock, C.: Active learning with strong and weak views: A case study on wrapper induction. In: Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI-2003). (2003)
13. Mitchell, T.: Machine Learning. McGraw-Hill (1997)
14. : (Fetch technologies, http://www.fetch.com/)
15. Muslea, I., Minton, S., Knoblock, C.: Adaptive view validation: A first step towards automatic view detection. In: Proceedings of ICML2002. (2002) 443–450
16. Kushmerick, N.: Wrapper induction: efficiency and expressiveness. Artif. Intell. (2000) 15–68
17. Chang, C.H., Kuo, S.C.: Olera: Semi-supervised web-data extraction with visual support. In: IEEE Intelligent systems. (2004)
18. Chang, C.H., Lui, S.C.: Iepad: information extraction based on pattern discovery. In: WWW '01: Proceedings of the 10th international conference on World Wide Web. (2001) 681–688
19. Lerman, K., Minton, S.: Learning the common structure of data. In: Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on Innovative Applications of Artificial Intelligence. (2000) 609–614
20. Arasu, A., Garcia-Molina, H.: Extracting structured data from web pages. In: SIGMOD '03: Proceedings of the 2003 ACM SIGMOD international conference on Management of data. (2003)
21. Crescenzi, V., Mecca, G., Merialdo, P.: Roadrunner: Towards automatic data extraction from large web sites. In: VLDB '01: Proceedings of the 27th International Conference on Very Large Data Bases. (2001) 109–118
22. Embley, D.W., Jiang, Y., Ng, Y.K.: Record-boundary discovery in web documents. In: SIGMOD. (1999)
23. Bunescu, R., Ge, R., Kate, R.J., Mooney, R.J., Wong, Y.W., Marcotte, E.M., Ramani, A.: Learning to extract proteins and their interactions from medline abstracts. In: ICML-2003 Workshop on Machine Learning in Bioinformatics. (2003)
24. Califf, M.E., Mooney, R.J.: Relational learning of pattern-match rules for information extraction. In: AAAI '99/IAAI '99: Proceedings of the sixteenth national conference on Artificial intelligence and the eleventh Innovative applications of artificial intelligence conference innovative applications of artificial intelligence. (1999) 328–334
25. McCallum, A., Freitag, D., Pereira, F.C.N.: Maximum entropy markov models for information extraction and segmentation. In: ICML '00: Proceedings of the Seventeenth International Conference on Machine Learning. (2000) 591–598
26. Nahm, U.Y., Mooney, R.J.: A mutually beneficial integration of data mining and information extraction. In: Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on Innovative Applications of Artificial Intelligence. (2000) 627–632
27. Hammer, J., Garcia-Molina, H., Cho, J., Crespo, A., Aranha, R.: Extracting semistructured information from the web. In: Proceedings of the Workshop on Management fo Semistructured Data. (1997)
28. Liu, L., Pu, C., Han, W.: Xwrap: An xml-enabled wrapper construction system for web information sources. In: ICDE '00: Proceedings of the 16th International Conference on Data Engineering. (2000) 611
29. Sahuguet, A., Azavant, F.: Wysiwyg web wrapper factory (w4f). In: WWW8. (1999)