Bootstrapping Information Extraction from Semi-structured Web Pages

Andrew Carlson[1] and Charles Schafer[2]

[1] Machine Learning Department, Carnegie Mellon University, Pittsburgh PA 15213, USA
[2] Google, Inc., 4720 Forbes Avenue, Pittsburgh PA 15213, USA

Abstract. We consider the problem of extracting structured records from semi-structured web pages with no human supervision required for each target web site. Previous work on this problem has either required significant human effort for each target site or used brittle heuristics to identify semantic data types. Our method only requires annotation for a few pages from a few sites in the target domain. Thus, after a tiny investment of human effort, our method allows automatic extraction from potentially thousands of other sites within the same domain. Our approach extends previous methods for detecting data fields in semi-structured web pages by matching those fields to domain schema columns using robust models of data values and contexts. Annotating 2–5 pages for 4–6 web sites yields an extraction accuracy of 83.8% on job offer sites and 91.1% on vacation rental sites. These results significantly outperform a baseline approach.

1 Introduction

This work addresses the problem of extracting structured records from semi-structured web pages with no human supervision required for the target web site. Semi-structured web pages are human-readable renderings of database entries. Familiar examples of semi-structured web page domains include books for sale, properties for rent, or job offers. We develop techniques that learn extraction models applicable to an entire domain of web sites from just 2–5 annotated pages from each of 4–6 web sites within that domain. The end result is a high-accuracy system that can be applied to many web sites within a domain without any human annotation of those sites.

In this work, we extract data from *detail pages* of web sites. These are pages which correspond to a single data entity, and which render various attributes of that entity in a human-readable form. An example detail page is shown in Fig. 1. While we focus on the setting where a page contains one record in this work, our methods could be easily adapted to the case where multiple data records exist on one page through use of existing data record detection algorithms (*e.g.* [1]).

Extracting structured records from semi-structured web pages is an important problem because a great deal of information on the internet is presented in this form. Moreover, the resulting structured records are particularly valuable to

W. Daelemans et al. (Eds.): ECML PKDD 2008, Part I, LNAI 5211, pp. 195–210, 2008.

Fig. 1. Part of an example detail page from the vacation rental domain. The page is about a particular house and displays attributes of that house such as its location, occupancy limit, and number of bathrooms.

downstream learning or querying systems because of their attribute/value structure and conformance to a fixed domain schema. Solving this problem without requiring human supervision for each target web site has been an understudied topic. Most previous work on this problem has required human supervision by requiring a person to either annotate example web pages from each target web site [2,3,4] or map the extracted data fields for each target site to columns in a domain schema [5,6,7]. The work that did not require supervision for each target site was not robust to typical variations on semi-structured web sites, such as cases where data fields do not have a label and do not match some unique pattern [8]. Our method only requires human supervision for a few web pages from a few sites in the target domain, allowing minimally supervised extraction from potentially hundreds or thousands of other sites within the same domain.

Our basic approach is to generalize extraction on a per-domain basis (e.g., for vacation rental sites). In taking on a new domain, first a human decides what the domain schema should be: that is, what schema columns are interesting and are present on many sites in the domain. Next, we annotate a small number of pages (2–5) for each of a few (4–6) web sites in the domain. These are provided as training data to an existing supervised extraction system[1], which learns a site-specific extraction model for each of the 4–6 sites; each model is then applied to all of the detail pages on the corresponding web site, yielding 4–6 machine-annotated sets of detail pages to be used as training data for our model. Finally, via the method presented in Sect. 3, we model both page contexts and values for each column in the domain schema, learning how to perform extraction for

[1] A supervised information extraction system making use of the partial tree alignment algorithm from DEPTA [7].

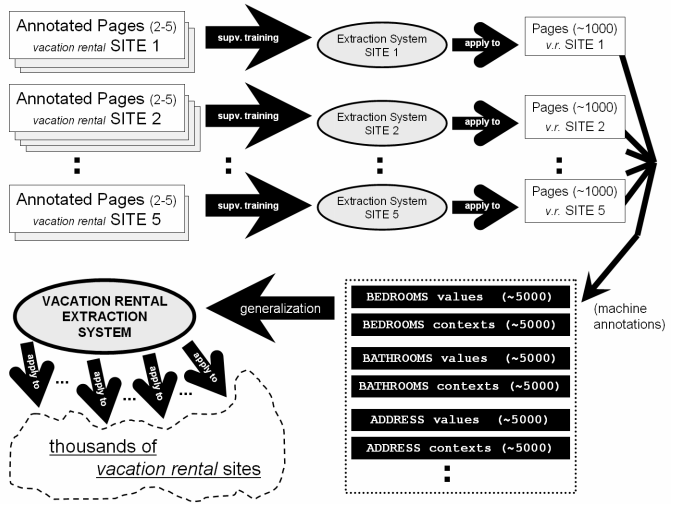

Fig. 2. A high-level view of learning under our system. By annotating 2–5 pages for 5 vacation rental sites, we train an extraction system that can be applied to thousands of other vacation rentals sites with no additional supervision required.

new vacation rental sites[2] with no additional human labeling effort. Figure 2 illustrates this learning scheme.

The experiments described in Sect. 4 measure an extraction accuracy of 83.8% on a previously unseen domain (job offer sites, which were all completely held out until evaluation time, providing a fair evaluation of how the system would perform in practice on other domains) with no direct supervision on the target sites being labeled. For previously unseen web sites in the vacation rentals domain (other vacation rental sites were used in algorithm development) we observed an accuracy of 91.1%. These results show a strong improvement over baseline accuracies of 61.8% for job sites and 65.8% for vacation rental sites, which were obtained using a logistic regression model. Other performance measures, also discussed in Sect. 4, indicate that use of our system for aiding human annotators is extremely promising, allowing a very small amount of effort to further increase accuracies.

2 Related Work

There has been much previous work on supervised learning of models for extraction of structured records from semi-structured web sites [2,3,4]. Such work

[2] Sites amenable to document object model tree alignment (implying that site-internal templates are used for automatic page generation), and making only trivial use of Javascript. We have observed that roughly 50% of vacation rental sites can be expected to meet these criteria.

requires a user to annotate training pages on a target web site, and then learns to extract data from the rest of the site. This process is labor-intensive, as it requires a new, separate labeling for every site.

Other previous work does not require a user to fully label example pages on the target web site, but does require manual labeling of the output. These approaches extract data fields by learning a common template for a page or site. Then, the user typically selects a subset of the output data fields and labels them according to the domain schema. Examples of such work are IEPAD [5], OLERA [6], and DEPTA [7]. This can be a labor-intensive process, because web sites tend to have many irrelevant fields, and thus the user must select from many more data fields than there are schema columns (web sites in our evaluation had an average of 20 data fields and 7 schema columns). Our work extends these methods, providing an automatic labeling of the extracted data by automatically mapping data fields to schema columns. This paper uses the partial tree alignment method of DEPTA to detect data fields, but many other template-finding algorithms, such as the repetitive pattern finding method of IEPAD, or the string editing distance method of OLERA, could be used in its place. Thus, this paper fills an important gap by allowing use of any one of a number of previous methods for detecting data fields on a web site, and providing minimally supervised selection of relevant fields and mapping to schema columns.

An exception to the requirement of user labeling of detected data fields is the DeLa system [8], which uses heuristics to automatically label extracted data fields. The heuristics depend on matches between the name of a schema column and labels present on a web page, or on data conforming to specific formats (e.g. dates or prices). There are many common cases where the DeLa system cannot label data fields. For example, the DeLa heuristic misses a common case where a data field does not have a label (for example, the job title field on most job sites is a heading with no prompt or label). Also, the DeLa heuristic would be confused by sites with multiple fields of the same data type (for example, the *bedrooms*, *bathrooms*, and *maximum occupancy* fields in vacation rentals). Our methods are much more robust to the variations typical on semi-structured web sites, and can handle these cases.

We label extracted data fields by comparing them to data fields on other annotated sites from the same domain. The only other previous work that we found that uses this approach to label semi-structured data is that of Golgher et al. [9]. This work bootstrapped onto new sites using examples of structured records from other web sites. These example records discarded the page context in which values appeared on the training web pages, and thus their system could not generalize across web sites based on context (our results in Table 1 demonstrate the high value of contextual features for the vacation rentals domain). Also, their model matched data values only at the token level, discarding useful textual features such as matches of short subsequences of characters.

Our method is partly inspired by techniques for schema matching in the database community. Specifically, the authors in [10] match elements in different database schemas by learning a model of schema columns which combines the

output of several base classifiers using a logistic regression meta-classifier. Our work applies a similar technique to a different problem. We match data fields on different web sites to a domain schema, rather than working with database fields. We also use different types of features, and we take advantage of the availability of distributions of data values by comparing frequencies of different values.

Freitag used a multi-strategy learning method where regression models were learned to map confidence scores to probabilities for various information extraction strategies. The probabilities were then combined in a Bayesian manner to make final predictions [11]. Our work uses a simpler, more direct method of combining models— we use a regression model for each schema column to combine confidence scores from our strategies (classifiers for each schema column that use some feature type to classify data fields). Additionally, our method classifies site-level data fields rather than individual data values, allowing better decisions to be made because more information is available at the site level.

3 Methods

In this paper, a *domain schema* is a set of attributes usually present in a record for the selected domain, while a *schema column* is a single attribute within that schema. A *detail page* of a web site is a page that corresponds to a single data record. A *data field* on a web site is a location within a template for that site (in this work, a node within a DOM tree template for that site), and a *data value* is an instance of that data field on a single detail page from that site. Data values appear in *contexts* which in this work are the text preceding a data value.

The central idea of our method is that we can bootstrap from a few training sites by building a model of data values and contexts for each schema column in our target domain schema, and then apply that model to many more sites with no further human effort. As training data, we require a collection of detail pages from a small number of web sites within the target domain that have been automatically labeled using a supervised wrapper induction algorithm according to some domain schema[3]. At test time, our method takes a collection of detail pages from a previously unseen target site[4] in the same domain. The goal is to annotate these pages according to the domain schema, identifying where schema columns are expressed in spans of text on the pages. We do this by first detecting potential data fields on the pages, and then classifying the data fields using a model learned from the training data.

[3] As mentioned in Sect. 1 acquiring this training data requires annotating 2–5 pages from 4–6 web sites to train a supervised system for each site, and then extracting records from the rest of the pages from the training sites.

[4] Our work assumes the availability of automatic methods for identifying such pages. To implement this ability we could use an approach that clusters and classifies pages, such as [12].

More specifically, we use the following method:

1. On the target site, create a template for the detail pages that identifies aligned regions across pages that look like potential data values. We consider these regions to be potential data fields. Our method for identifying these data fields is described in Sect. 3.1.
2. Based on the machine-annotated training data from a few training sites, label the detected data fields on the target site with a score that indicates how likely they are to be instances of each schema column in a domain schema. This is described in Sect. 3.2 below.
3. Using these scores, either automatically annotate the target site, or else give recommendations to aid a human annotator (these are alternate uses of our output). Details of this procedure are given in Sect. 3.3.

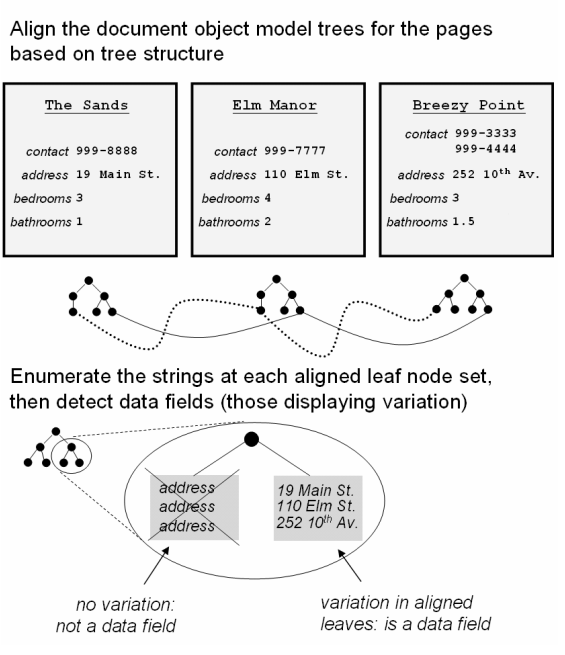

Fig. 3. Alignment of document object model trees and subsequent detection of data fields

3.1 Detecting Data Fields

We detect data fields across the pages on the target site by using the Partial Tree Alignment algorithm, which is part of the DEPTA system [7]. The data field detection is done in an unsupervised manner as follows. First, document object model (DOM) trees for the detail pages on a site are aligned. Next, the

strings occurring at aligned nodes are enumerated and inserted into a set. Any of these sets containing more than one element, then, corresponds to a region in the "template" for detail pages that exhibits variability across pages. Such regions (aligned DOM tree nodes exhibiting variability) are taken as the candidate data fields for the site. This process is illustrated in Fig. 3. Data fields that aligned to fewer than half of the pages on a site were filtered out, because these were typically not interesting data fields. Such filtered data fields tended to be different paragraphs of free text which the tree alignment algorithm occasionally decided to align. In effect, this filtering step ignores free text data fields which do not occur on all pages of the site and are not interesting data fields.

3.2 Classifying Data Fields

For each data field on the target site, we assign a score representing its correspondence to each schema column in the domain schema. A high score indicates a high degree of confidence that a data field should be mapped to that schema column. Informally, we find data fields on the target site that have data values that are similar to the values observed in the training data for a schema column, and we also want to find data fields that appear in page contexts that are similar to contexts from the training data.

To compute the score for a data field f and a schema column c, an obvious method to try is to extract a set of features for the data field, and use a classifier to map data fields to schema columns, where we train the classifier with our training sites. However, we discuss below in the 'Motivation' subsection that this method is impractical given the number of different textual features we must use (tens of thousands) and the number of training examples we have (roughly one hundred), so that a single classifier will tend to overfit and give irrelevant features high weights. Instead, we use a model that computes, for K different *feature types*, how similar the feature values observed for that data field are to the feature values observed in the training data for that schema column. This yields K different subscores, one for each feature type. We then combine those scores using a regression model that outputs a final score. The intuition is that for each schema column, different types of features matter, and that comparing distributions of observed features is much less noisy than singling out individual features in our model.

In the rest of this subsection, we describe the types of features that we use to represent data fields and schema columns. We then give details about our method for comparing different distributions of values of these features, describe the regression model that combines the similarity scores for each feature type, and give details on how we train the model. We end the subsection with further discussion of the motivation behind our model.

Feature Types. Our method uses four different *feature types*. A feature type is a family of features, such as 'lowercase tokens occurring in data values of a

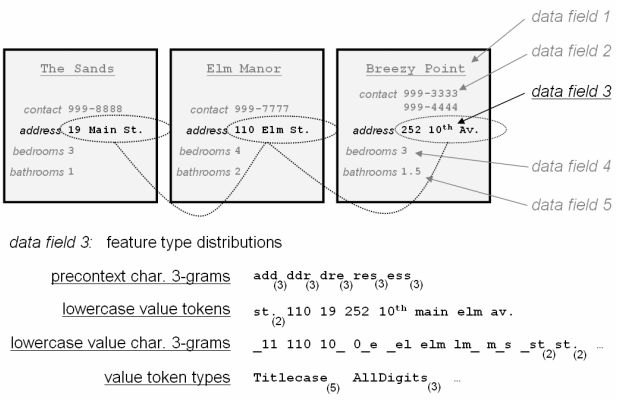

Fig. 4. Feature type distributions are created for data fields in aligned document object model trees. This is done for both training and test sites, so that the resulting distributions can be compared. In the figure, counts exceeding 1 are indicated by subscripts.

data field.' Our method combines different types of features because there can be different indicators of the correct label for a data field. We might sometimes have strong clues from the text before a data field (e.g. it might always say 'Bedrooms:', indicating that the data field holds the value for *bedrooms*). We might also have strong clues from the text inside a data field (e.g. 'Software Engineer Level I' indicating a *job title*). See Fig. 4 for an illustration of the feature type distributions for an example data field. We use the following feature types:

- **Precontext character 3-grams**: We extract character 3-grams from the non-tag text preceding a data field in HTML. Web sites often abbreviate or use different forms of labels, such as "Bedrooms", "Beds", "Bedrms". Character 3-grams can capture these variations.
- **Lowercase value tokens**: We tokenize data values by whitespace, and convert them to lowercase. Many of the schema columns that occur in semi-structured data have values that come from a limited vocabulary.
- **Lowercase value character 3-grams**: We extract character 3-grams of the data values. Many types of data can be abbreviated, or use similar but not identical words, or use special symbols or codes, all of which can be better captured using character 3-grams.
- **Value token types**: We categorize tokens in data values into general types (e.g. all caps, all digits, title case). These features are helpful for addresses, unique IDs, or other schema column types with a mix of token types.

Comparing Distributions of Feature Values. We compare distributions of features observed in training data to distributions observed in data fields to be labeled on the target site. This approach uses all of the data, including relative frequencies of features, to make its predictions. This is an advantage in cases such as the vacation rental domain, which includes schema columns *bedrooms*

and *bathrooms* which have very similar data values but, typically, different distributions of these values. Additionally, comparing distributions of features helps us avoid overfitting when dealing with high-dimensional feature spaces and small numbers of training examples.

A common method of comparing two distributions is Kullback-Leibler Divergence, or KL-Divergence. For the kth feature type, we have a distribution P_{kc} for the schema column c in the training data, and P_{kf} for the data field f in the target site. The KL-Divergence from the training data distribution to the data field distribution for feature type k is:

$$KL_k(c||f) = \sum_i P_{kc}(i) \log \frac{P_{kc}(i)}{P_{kf}(i)} \tag{1}$$

An issue with KL-Divergence is that when $P_{kc}(i) > 0$ and $P_{kf}(i) = 0$ for some feature value i, the KL-Divergence is undefined. To counter this, we use Skew Divergence, a smoothed version of KL-Divergence [13]:

$$SD_k(c||f) = KL_k(c||\alpha * f + (1 - \alpha) * c) \tag{2}$$

Note that $\alpha = 1$ gives the original KL-divergence. A value close to 1 gives a slightly smoothed KL-Divergence. We use $\alpha = 0.8$ in this work.

We alter the Skew Divergence with a simple transformation to create the Skew Similarity score, which has value 0 for the data field f most dissimilar from the schema column c, and highest value for the data field which is most similar to the schema column in the training data.

$$SS_k(c, f) = [\max_f SD_k(c||f)] - SD_k(c||f) \tag{3}$$

Our choice of Skew Divergence as the method of comparing distributions is one of many reasonable choices. We chose to use a smoothed version of the KL-Divergence. Other measures of the distance between two distributions, such as the Jensen-Shannon divergence, would also be reasonable.

Combining Skew Similarity Scores. Skew Similarity scores for different feature types will have different scales, and in some cases might actually be misleading (for instance, if contextual features are not helpful for some schema column in a domain, we want to ignore the Skew Similarity score for these features). Thus, we cannot simply average or sum the scores. Instead, we combine Skew Similarity scores for the different feature types using a linear regression model. A data field f on a target site is given a score $LR_c(f)$ for a schema column c using a linear regression model which combines the K different similarity scores (one for each feature type). The final score is a weighted combination of the Skew Similarity scores plus a constant:

$$LR_c(f) = \beta_{0c} + \sum_{k=1}^{K} \beta_{kc} SS_k(c, f) \tag{4}$$

If we view the Skew Similarity scores as different classifiers, each making predictions based on models over features learned from training data, then this overall model can be viewed as a stacked classifier model. Stacking, which generally refers to using the outputs of classifiers as inputs to another classifier, has been shown to often be an effective way of combining the predictions of multiple classifiers [14]. Our choice of linear regression as the 'top-level' classifier was motivated by the good empirical performance observed across a wide variety of data sets in previous work on stacking [14], and by ease of implementation. A number of other choices could be made for the top-level classifier, including logistic regression if posterior probability estimates were desired.

Training the Model. Training the stacked classifier regression model involves learning the weights β. This is done by holding out each training site, and generating data points for the linear regression for each data field on the held-out site with Skew Similarity scores computed relative to the other training sites. Using such a held-out approach to training the model is important, because otherwise the training examples will have very biased estimates of the accuracies of the Skew Similarity scores (they will appear to be much better than one would expect on held-out data). For every data field f on a training site and every schema column c in the domain, we generate an example: $(\delta(c,f), SS_1(c,f), SS_2(c,f), \ldots, SS_K(c,f))$ where $\delta(c,f) = 1$ if the data field f is annotated with schema column c in the training data and 0 otherwise. The coefficient β_{kc} controls how strong an indicator feature type k is for schema column c. The coefficients allow our method to learn which feature types are reliable indicators on a per-domain, per-schema column basis, which is essential for robust performance across a wide variety of domains and sites.

Motivation for the Model. One might question why we did not use a model for each schema column that learned a classifier over a feature vector that held all of the features that describe a data field. The key reason is that each annotated training site yields only as many training examples as there are data fields on that site. For example, with 5 training sites and an average of 20 data fields detected per site (typical values in our setting), we would have 100 examples, most of which would be negative examples. With the various character n-gram features and token features that we need in order to robustly recognize variations of data values and contexts, the dimensionality of the feature vectors would reach tens of thousands of features. It is difficult to train a reliable classifier in such a setting. Even with appropriate feature selection and regularization methods, with so few training examples and so many features, overfitting seems inevitable.

We instead chose a stacked model that combines a small number of similarity scores. Our choice was motivated by several reasons. First, a similar model design that combined a set of base learners with a regression-based classifier has been shown to be useful for a related problem, matching database schemas [10]. Second, such a model has to learn only $k + 1$ parameters for each schema column, where k is the number of base learners to combine. In our setting, where $k = 4$, we can expect to obtain sensible estimates of these parameters. Third,

similarity measures like Skew Divergence are effective ways to compare distributions of values and summarize them with a score. Finally, we desired a model that could be easily extended with additional sources of information, which our model facilitates. The new information can be simply added to the regression model.

In our evaluation, we compare these two approaches, using a regularized logistic regression classifier as a baseline approach that uses all features at once. Our results suggest that overfitting is a significant issue for the baseline method, and that our model yields superior performance.

3.3 Labeling the Target Site

After we have computed a score for each possible mapping of a data field on the target site to a schema column, we must output a labeling of the target site. The output of our method depends on the application setting. If we are operating in a fully-automated setting, the system chooses the data field f which maximizes the score $LR_c(f)$ for each schema column c, and annotates all pages on which those data fields occur with the appropriate labels (a data field does not always occur on every page). Not all schema columns occur on every target site, so we choose no data field if the maximum linear regression score is below a threshold θ for a schema column[5].

If we are aiding a human annotator, we recommend the top N data fields on the target site for each schema column. This dramatically reduces the annotation effort required, as the annotator only has to look at a handful of data fields.

We consider both scenarios in our evaluation.

4 Evaluation

The evaluation of our method was designed to measure the accuracy of automatically labeling new sites, and also to measure how well we could make recommendations to human annotators. Given a collection of annotated sites for a domain, we performed our evaluation in a cross-validated fashion, training the system on all but one of the sites, and testing on the held-out site. We used as our gold standard the nearly-perfect machine annotations from our supervised extraction system (hand-checking a large sample of annotations failed to find any errors in many dozens of pages, implying that they were over 99% accurate). In the results below, we refer to our method as the 'Stacked Skews Model.'

4.1 Baseline Method

As discussed in Sect. 3.2, an obvious approach to the problem we consider is to extract a feature vector describing each schema column from each training site, and train a multiclass classifier to classify data fields with which schema column they match (and to decide which data fields are irrelevant). To evaluate the

[5] We used $\theta = 0.1$ in this work, selected based on the development data.

performance of our system relative to this baseline, we implemented a regularized logistic regression classifier as our baseline method. The output of the classifier is used just as the output of the linear regression model is in our system, with the highest scoring data field for each schema column being mapped to that schema column. We hand-tuned the threshold below which the baseline decides that a schema column is not present on a target web site (we used a value of 0.2, which optimized the test results for the baseline). In the remainder of this section, we refer to this baseline method as 'Logistic Regression.'

4.2 Metrics

The metrics we used to evaluate our system are:

- Accuracy: On each page of the test site, we create an annotation for each schema column in the domain schema, or assert that the schema column is not present on the page. We measure the accuracy of these decisions[6].
- Recall in Top 3: For a test site, we select the top 3 data fields for each schema column. On each page of that test site, we check to see if each correct annotation on that page is in the top 3 ranked data fields for its label.

4.3 Domains

We evaluated our system using two different domains: vacation rentals and job sites. Vacation rental sites list properties offered for rent, with attributes such as number of bedrooms, maximum occupancy, and a text description of the property. Job listing sites describe open job positions, and include attributes like the company offering the job, the date it was posted, and the location of the job. Refer to Fig. 6 for the complete list of schema columns for each domain schema. We developed our methods using other sites in the vacation rentals domain than the ones we ultimately trained and tested on. The vacation rental sites used in our evaluation were not seen until the system development was finished, nor were any sites in the jobs domain.

4.4 Web Sites

We selected web sites for each domain that had most of the schema columns listed in Fig. 6 present and that were amenable to tree alignment. We were unable to find publicly available data suitable for our evaluation; while there was limited annotated data available publicly for job sites, our method needed at least 5 different sites in a domain annotated with a similar schema. Job sites had 240 pages and 17 detected data fields on average. Vacation rental sites had 151 pages and 25 detected data fields on average.

[6] Some schema columns appear on pages in multiple locations. Any correct location is acceptable.

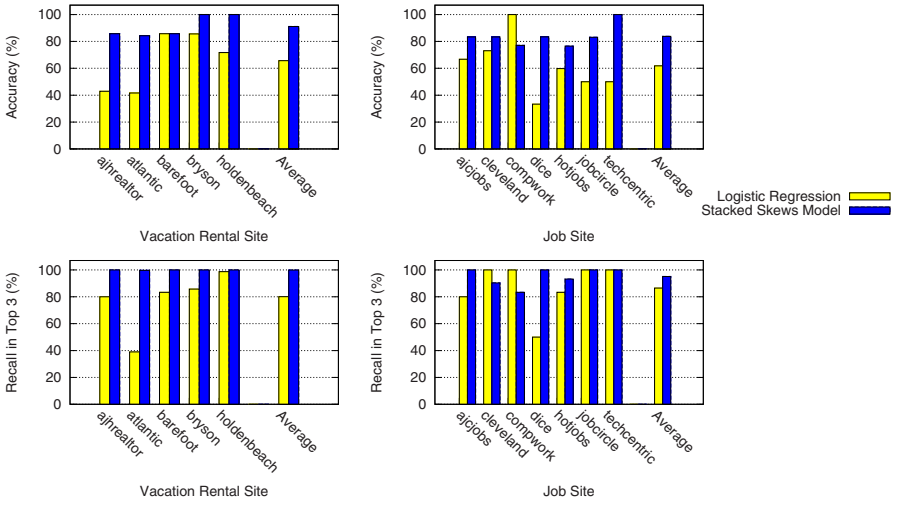

Fig. 5. Results for Logistic Regression and Stacked Skews Model when holding out each site from the training data and testing on it, averaged across schema columns. The 'Average' columns give results averaged across sites and columns. *(Top and Bottom Left)* Accuracy and Recall in Top 3, respectively, for each site in the jobs domain. *(Top and Bottom Right)* Accuracy and Recall in Top 3 for the vacation rentals domain.

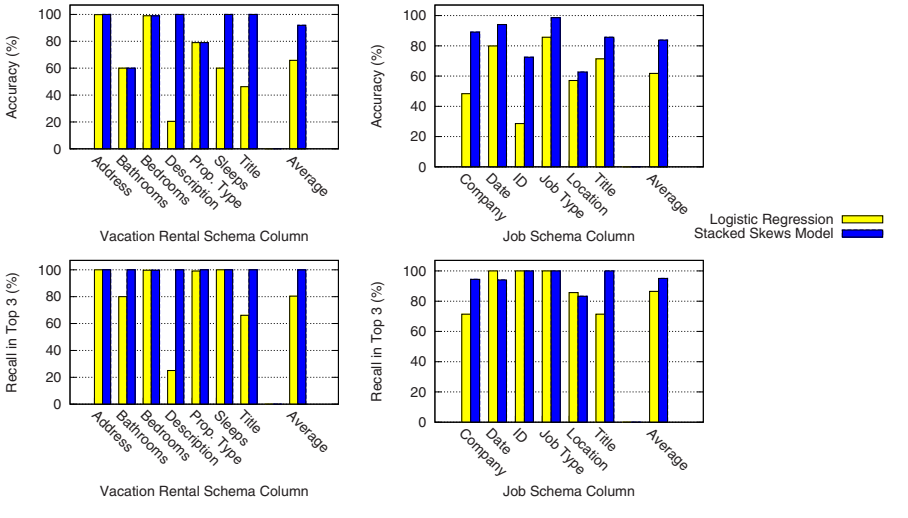

Fig. 6. Results for Logistic Regression and Stacked Skews Model by schema column, averaged across using each held-out web site as a test set. The 'Average' columns give results averaged across sites and schema columns. *(Top and Bottom Left)* Accuracy and Recall in Top 3, respectively, for each schema column in the jobs domain. *(Top and Bottom Right)* Accuracy and Recall in Top 3 for the vacation rentals domain.

Table 1. Results from feature type ablation experiments with the Stacked Skews Model. The top 4 rows give results for leaving out one feature type, and the rows below those give results with just one feature type.

	Jobs		Vacation Rentals	
Features	**Accuracy**	**Top 3**	**Accuracy**	**Top 3**
Excl. Context n-grams	85.3	95.1	54.4	86.5
Excl. Value n-grams	75.1	89.7	91.1	99.9
Excl. Value Token Types	83.8	95.1	91.1	99.8
Excl. Value Tokens	73.7	89.7	85.4	99.9
Only Context n-grams	41.8	61.1	77.0	89.9
Only Value n-grams	68.0	97.8	62.8	99.8
Only Value Token Types	30.2	48.2	31.3	46.5
Only Value Tokens	71.4	81.6	54.3	89.9
All	**83.8**	**95.1**	**91.1**	**99.9**

4.5 Results

Results by Site. Figure 5 shows results for each site, averaged across the different schema columns in the domain, for both the Logistic Regression baseline and our Stacked Skews Model. Averaged across all sites and schema columns (indicated by the 'All' column in the figure), our method achieved an accuracy of 91.1% for vacation rental sites, and 83.8% for job sites, significantly higher than the baseline accuracies of 65.8% and 61.8%. The results are reasonably consistent across different sites. Additionally, the correct data fields for vacation rental sites were present in the top 3 recommendations 99.9% of the time, and 95.1% for job sites. The baseline classifications of data fields had the correct answers in the top 3 only 80.3% and 86.5% of the time. As an additional comparison, random assignment of data fields to schema columns would have an expected accuracy of 5.9% for job sites, and 4.0% for vacation rentals, and an expected top 3 performance of 17.6% for job sites, and 12.0% for vacation rentals.

Results by Schema Column. Figure 6 gives results for each schema column for our methods, averaged across web sites. We see that our method is generally accurate for a wide range of data types, and nearly always exceeds the baseline results. Our method is particularly better suited to schema columns which tend to have free text (for example, the *description* and *title* schema columns for vacation rentals, and the *company* and *ID* schema columns for jobs). We believe that this is due to the logistic regression's tendency to overfit when many different features can perfectly classify the training data. The baseline method fares much better for schema columns with more limited vocabularies.

Referring to the Stacked Skews Model results for individual schema columns, most of the schema columns have high accuracy. The cases where performance suffered were the *job type* schema column, which has high variation in both contexts and values (some sites even use special codes as values for this schema

column), *bathrooms*, which had trouble when there was also a 'Half Bathrooms' item on a site, and *property type*, where one site incorrectly labeled a data field as *property type* because it was preceded by the text 'Property Type' on the site, but it was a different sense of property type from the intended meaning from our domain schema[7].

Identifying Missing Schema Columns. Most of the schema columns in each domain were present on our evaluation sites. In the case where a column was not present on a site, the accuracy metric required us to correctly identify that column as missing, or else it was considered an incorrect answer. Identification of such missing columns was described in Sect. 3.3. We evaluated the accuracy of our model for these cases, to see if they were a significant source of error in our evaluation. For each domain, there were 5 cases where a schema column was missing from a site. The Stacked Skews Model identified missing schema columns for vacation rentals with an accuracy of 80.0%, and a lower accuracy of 49.3% for job sites. This is because most job listing sites had some unstructured text describing the job, in addition to well-formatted sections which typically held our extraction targets. Often when a schema column was not present in the semi-structured text of a site, one of the data fields corresponding to the free text was chosen.

Feature Type Ablation Study. To assess the contributions and relative importance of each of the feature types, we ran ablation experiments where we used subsets of the four feature types. We considered combinations where we held out each feature type, and also where we used each feature type alone. Table 1 gives results for these experiments. We see that context features are very informative for the vacation rentals domain, but not informative for the jobs domain (in fact, excluding them improves average accuracy). The value token type features do not appear to be useful for either domain. In general, we see that using multiple feature types in combination allows the system to achieve higher accuracies than any single feature type, indicating that the different feature types provide complementary information, and that our stacking method effectively combines these sources of information.

5 Conclusions

This work addressed the problem of extracting structured records from semi-structured web pages. The system we described demonstrated a way to learn high-quality automated extraction systems for large numbers of semi-structured web sites, by exploiting a tiny, fixed amount of human effort per domain of interest. Starting from manual annotation of 2–5 pages each on 4–6 sites in a domain, we bootstrapped a system that achieves high accuracies even on a domain, jobs, that was not considered during development of our model. This

[7] Our domain schema included a notion of property type as a house, townhouse, condominium, etc. The site's notion of property type was beachfront vs. not.

performance is encouraging for use either as a standalone extraction system for certain applications, or as an aid to human annotators. The performance significantly exceeds the performance of a competitive baseline method.

Acknowledgments. Most of this work was done during an internship at Google in Pittsburgh. The authors wish to acknowledge Kamal Nigam, William Cohen, Gideon Mann, and the anonymous reviewers for their helpful comments, William Morris and Dominic Widdows for assistance with annotating data, and Haakan Younes for his reimplementation of the Partial Tree Alignment algorithm.

References

1. Liu, B., Grossman, R.L., Zhai, Y.: Mining data records in web pages. In: KDD, pp. 601–606 (2003)
2. Soderland, S.: Learning information extraction rules for semi-structured and free text. Machine Learning 34(1-3), 233–272 (1999)
3. Kushmerick, N., Weld, D.S., Doorenbos, R.B.: Wrapper induction for information extraction. In: IJCAI, pp. 729–737 (1997)
4. Muslea, I., Minton, S., Knoblock, C.: A hierarchical approach to wrapper induction. In: AGENTS, pp. 190–197 (1999)
5. Chang, C.H., Lui, S.C.: IEPAD: information extraction based on pattern discovery. In: WWW, pp. 681–688 (2001)
6. Chang, C.-H., Kuo, S.-C.: OLERA: Semisupervised web-data extraction with visual support. IEEE Intelligent Systems 19(6), 56–64 (2004)
7. Zhai, Y., Liu, B.: Web data extraction based on partial tree alignment. In: WWW, pp. 76–85 (2005)
8. Wang, J., Lochovsky, F.H.: Data extraction and label assignment for web databases. In: WWW, pp. 187–196 (2003)
9. Golgher, P.B., da Silva, A.S., Laender, A.H.F., Ribeiro-Neto, B.A.: Bootstrapping for example-based data extraction. In: CIKM, pp. 371–378 (2001)
10. Madhavan, J., Bernstein, P.A., Doan, A., Halevy, A.: Corpus-based schema matching. In: ICDE, pp. 57–68 (2005)
11. Freitag, D.: Multistrategy learning for information extraction. In: ICML, pp. 161–169 (1998)
12. Crescenzi, V., Mecca, G., Merialdo, P.: Wrapping-oriented classification of web pages. In: Nyberg, K., Heys, H.M. (eds.) SAC 2002. LNCS, vol. 2595, pp. 1108–1112. Springer, Heidelberg (2003)
13. Lee, L.: Measures of distributional similarity. In: ACL, pp. 25–32 (1999)
14. Ting, K.M., Witten, I.H.: Issues in stacked generalization. Journal of Artificial Intelligence Research 10, 271–289 (1999)