Open access • Proceedings Article • DOI:10.1145/2396761.2398464

A simple approach to the design of site-level extractors using domain-centric principles — Source link

Chong Long, Xiubo Geng, Chang Xu, S. Sathiya Keerthi

Institutions: Yahoo!, Chinese Academy of Sciences, Microsoft

Published on: 29 Oct 2012 - Conference on Information and Knowledge Management

Topics: Domain (software engineering)

A Simple Approach to the Design of Site-Level Extractors Using Domain-Centric Principles

Chong Long
Yahoo! Labs, Beijing
chongl@yahoo-inc.com

Xiubo Geng
Yahoo! Labs, Beijing
gengxb@yahoo-inc.com

Chang Xu
Chinese Academy of Sciences
nuaaxc@gmail.com

Sathiya Keerthi
Cloud and Information
Services Lab, Microsoft
keerthi@microsoft.com

ABSTRACT

We consider the problem of extracting, in a domain-centric fashion, a given set of attributes from a large number of semi-structured websites. Previous approaches [7, 5] to solve this problem are based on page level inference. We propose a distinct new approach that directly chooses attribute extractors for a site using a scoring mechanism that is designed at the domain level via simple classification methods using a training set from a small number of sites. To keep the number of candidate extractors in each site manageably small we use two observations that hold in most domains: (a) imprecise annotators can be used to identify a small set of candidate extractors for a few attributes (anchors); and (b) non-anchor attributes lie in close proximity to the anchor attributes. Experiments on three domains (*Events*, *Books* and *Restaurants*) show that our approach is very effective in spite of its simplicity.

Categories and Subject Descriptors

I.2.6 [**Artificial Intelligence**]: Learning—*Knowledge Acquisition*; I.2.7 [**Artificial Intelligence**]: Natural Language Processing—*Text Analysis*

General Terms

Algorithms

Keywords

Information Extraction, Text Mining

1. INTRODUCTION

The web is a great source which can be tapped to extract useful data for driving applications. Nowadays even small websites (e.g. websites of schools, libraries, museums) automatically generate their pages via scripts that load data from underlying databases. Thus these pages are well formatted and in semi-structured form. Still, it is a challenge to extract useful information from them since that information resides in a small part of the pages, surrounded by a lot of extraneous elements.

We are interested in domain-centric web extraction where we choose a domain (e.g. *Events*, *Books* and *Restaurants*), define an attribute schema of interest (see Table 1 for examples), provide some domain knowledge (e.g. weak annotators for some attributes) and use these to extract the attributes of entities, from websites associated with the domain. If we have extracted a large number of entities from various sources, we can set up many useful web applications such as book/restaurant aggregators, hyper-local event recommendation service, etc. Consider the example of extracting attributes associated with events happening in all schools in the United States. More than 100K schools in the United States have websites. In domains such as this, it is not scalable to provide supervision at the individual site level. Though there is a large body of literature in information extraction [4], very few works address the large scale domain-centric extraction problem.

In this paper we propose a new and simple approach for large scale domain-centric extraction. Like Dalvi et al. [3] we design site-level extractors. However, instead of learning them simply and solely from annotations, we employ an extractor scoring model that uses various forms of domain knowledge and features to choose the extractors. Like [7, 5] the parameters of this model are learnt using domain level supervision. It is sufficient to have weak annotators only for some key attributes called the anchor attributes. We make the observation that, on web pages of the domain, the non-anchor attributes are located close to the anchor attributes; this is used to keep the number of candidate extractors small, thus making extractor inference efficient. In essence, our approach keeps the positive properties of the two classes of existing methods while discarding their negatives.

2. DOMAIN-CENTRIC EXTRACTION

We are interested in extraction approaches that are *domain-centric*, where, we fix a domain d and aim to extract a specified set of attributes (schema) about entities, from a large set of websites related to the domain, e.g., if d is *Events*, then: an entity is one event; the websites could be the set of all official websites of schools; the attributes could be *Date*, *Time*, *Title*, *Location* and *Description*.

A complete solution of the domain-centric extraction problem involves the following steps: discovering websites that contain information of interest, analyzing the websites to identify the correct subset of pages to extract from, developing the extraction rules, and integrating the gathered data from all websites into a single database. In this paper we are interested in the third problem of

Table 1: Examples of domains and their attributes. The anchor attributes (see Section 6) are marked in boldface.

Domain	Attributes
Events	*Title*, **Date**, **Time**, *Location*, *Description*
Books	*Name*, **Price**, **ISBN10** (10 digits), **ISBN13** (13 digits), **Date**, *NumberOfPages*, *Description*
Restaurants	*Name*, **Address**, **Phone**, *Timing*, *Review*, *Description*

developing the extraction rules. Since the number of websites is usually large (e.g. the number of *Schools* websites in the United States is more than 100K), supervision at the site level is not feasible. Thus, we focus on approaches that use supervision only at the domain level; this supervision consists of specifying weak/noisy annotators for a few (anchor) attributes (details in Section 3) and a small training set of correctly extracted data from a small number of websites.

Two good methods based on Hierarchical CRF [7] and Markov Logic Networks [5, 6] have been proposed in the literature for solving this extraction problem. Our approach is simpler and quite distinct from these. To clearly explain our approach and bring out the differences between these methods let us introduce some notations. Let A denote the set of attributes of interest in domain d. Let S be the set of websites for domain d from which extraction needs to be done. Since we will discuss extraction related to only one domain d, hereafter we will not explicitly mention the dependence of various terms on d. Let P_s be a representative set of web pages from a website $s \in S$.

To keep the discussion simple, we restrict ourselves only to extraction from *detail pages* in this paper. Here a *Detail Page* contains information about only one entity (e.g. an event). The related works mentioned above [7, 5] also deal with detail pages and so this restriction facilitates comparison. Also, the ideas developed for detail pages can be easily extended to the other formats. For example, unsupervised methods [1] can be used first to get records from a list; then each record can be viewed as a detail page and attributes can be extracted using the method developed for detail pages. Given that we work only with detail pages, we can make the following assumption.

Assumption 1. Each attribute occurs at most once on each page. For example, a detail page about a restaurant has only one name and address.

Each page $p \in P_s$ has an associated html dom tree. (We can also overlay a visual tree and use signals from it.) The leaf nodes of this tree contain the actual contents of the page, and they define a sequence. This sequence information allows us to define a distance measure between leaf nodes, which we will use later in Section 6. A second assumption that commonly holds is the following.

Assumption 2. Each attribute occurs within a single leaf node.

Sometimes this assumption is violated. For example, the *Description* attribute tends to occur in several consecutive paragraphs, which are different leaf nodes. But this can be easily handled by doing a pre-processing to analyze the dom tree and combining several such leaf nodes into a single leaf node, thus causing assumption 2 to hold. Let L_p be the set of leaf nodes after such a pre-processing. The extraction problem is to find, for each page p and each $a \in A$, a node in L_p that lodges attribute a.

3. GRAPHICAL MODELS APPROACH

The extraction problem stated above can be rewritten as a problem of labeling all nodes in L_p as follows. Let us include an extra attribute called *Irrelevant* in A to label nodes which have nothing to do with the attributes in A. Let $\tilde{A} = A \cup \{Irrelevant\}$. Let $\mathbf{y}$ denote a labeling of all nodes in page p with labels from $\tilde{A}$ while obeying assumptions 1 and 2. Let $\mathbf{Y}_p$ be the set of all possible $\mathbf{y}$ for page p.

Let $\mathbf{f}(\mathbf{y}, p)$ be a feature vector associated with $\mathbf{y}$ and p. This feature vector contains features based on content (word features, orthographic features, and dictionary-based features), proximity (e.g. a feature that checks if two attributes are located nearby in the node sequence), precedence (e.g. a feature that checks if an attribute occurs after another attribute in the node sequence) and alignment (e.g. a feature that checks if the same attribute is located in the same position in two different pages). A weight w_i is associated with feature $f_i(\mathbf{y}, p)$ to form the total score $\mathbf{w} \cdot \mathbf{f}(\mathbf{y}, p)$ where $\mathbf{w}$ is the weight vector containing the w_i's.

Inference is done using the score of $(\mathbf{y}, p)$:

$$\mathbf{y}^*(p) = \arg\max_{\mathbf{y} \in \mathbf{Y}_p} \mathbf{w} \cdot \mathbf{f}(\mathbf{y}, p) \qquad (1)$$

The features in $\mathbf{f}$ can be viewed as factors defined on a graphical model involving $\mathbf{y}$. Then, (1) becomes the Viterbi computation associated with that graphical model. Depending on the type of cliques that get formed (for example, whether the factors connect only local labels or not), this inference computation can be easy or hard. In Hierarchical CRF [7], complex factors that connect distant labels are avoided so as to keep the inference computationally tractable. But, Satpal et al. [5] show that using complex factors significantly lifts the performance of Hierarchical CRF. Note, however, that this gain in performance comes at the expense of much more complex inference in the method of Satpal et al. [5]. Thus the graphical model approach involves a balance between improved performance and complexity of inference.

Learning of $\mathbf{w}$ is done by using a training set $\{(p_i, \mathbf{y}_i)\}$ and minimizing the training objective function:

$$\min_{\mathbf{w}} R(\mathbf{w}) + \sum_i \mathcal{L}(\mathbf{w}, p_i, \mathbf{y}_i) \qquad (2)$$

where $R(\mathbf{w})$ is a regularizer (e.g. the L_2 regularizer, $\|\mathbf{w}\|^2/(2\sigma^2)$) and $\mathcal{L}(\mathbf{w}, p_i, \mathbf{y}_i)$ is either the negative log-likelihood of $\mathbf{y}_i$ given by the model, or a loss function based on the large margin principle. When the underlying graphical model is complex, the solution of (2) is also complex; Satpal et al. [5] employ a MIRA type algorithm [2].

4. OUR APPROACH

Our approach is rooted on the following three key observations.

(**O1**) There exist good site-level extractors for the attributes.

(**O2**) By utilizing domain knowledge it is possible to generate a manageably small set of candidate site-level extractors that contains good site-level extractors in it.

(**O3**) A domain-centric scoring model can be used to choose the best site-level extractor from the candidate extractors generated for each site.

O1 makes sense because, even in a small site such as a *School* website, web pages are automatically generated using a script that takes inputs from a database. Thus all detail pages tend to have the same format and so excellent site-level extractors (such as xpaths) exist. We will take up **O2** in Section 6. Like in the graphical model approach, **O3** is reasonable since signals based on content, proximity, precedence, alignment, etc. tend to be uniform across sites associated with a given domain.

Let $\mathbf{e}$ denote a site-level extractor; $\mathbf{e}$ consists of a collection of $\{e_a\}_{a \in A}$ where each e_a is the extractor associated with attribute a. It is possible that the extractors for the various attributes could be dependent on each other. We will define $\mathbf{e}$ precisely in Section 6; for now it suffices to say that $\mathbf{e}$ can be applied on all pages in a site to extract the attributes. Let $\mathbf{E}_s$ denote the set of candidate site-level extractors for site s. Given a site-level extractor $\mathbf{e}$, let $\mathbf{f}(\mathbf{e}, P_s)$ denote a vector of site-level features for $\mathbf{e}$ derived from the representative set of pages, P_s. Like in the graphical model approach, the features are based on content, proximity, precedence, alignment etc. The key difference is that, here we derive site-level features by aggregating statistics over P_s, whereas the graphical model approach uses features at the page level.

Inference (choosing a site-level extractor) can be defined by associating a weight vector $\mathbf{w}$ with $\mathbf{f}$ and using

$$e^*(s) = \arg\max_{\mathbf{e} \in \mathbf{E}_s} \psi(\mathbf{w}, \mathbf{e}, P_s) = \mathbf{w} \cdot \mathbf{f}(\mathbf{e}, P_s) \tag{3}$$

Learning of $\mathbf{w}$ can be done in several ways. It turns out that learning is much simpler for our approach. See Section 6 for details.

5. PARAMETER LEARNING

Let us now consider the learning of the parameter vector $\mathbf{w}$ used in (3). The training set for our approach consists of a set of triples, $\{(\mathbf{e}_j, s_j, P_{s_j})\}$ where $\mathbf{e}_j$ is the true extractor associated with site s_j. This training set is of a different type than the one used by the graphical model approach: the former specifies site-level extractors while the latter specifies page level extractions. In our approach, each site chosen for providing supervision seems to give just one training example. But note that, each training example also has a representative page set P_s, which is used for forming the feature vector $\mathbf{f}(\mathbf{e}, P_s)$. We can work with different representative subsets of pages P_s within a site s to generate more training examples. Apart from providing more training examples, such a generation also gives the ability for the extractor design to be robust to operation with a subset of the pages in a site instead of all pages in a site, during inference.

There are several possibilities for setting up the learning model to find $\mathbf{w}$.

The first idea is to use the scoring function $\psi(\mathbf{w}, \mathbf{e}, P_s)$ to define a multi-class model over the set of candidate extractors $\mathbf{E}_s$. This leads to the training problem

$$\min_{\mathbf{w}} R(\mathbf{w}) + \sum_j \mathcal{L}(\mathbf{w}, \mathbf{e}_j, P_{s_j}) \tag{4}$$

where $R(\mathbf{w})$ is the regularizer (like in (2)) and $\mathcal{L}$ is a suitable loss function. For example, in the case of the large margin (SVM) approach, $\mathcal{L}$ is given by

$$\mathcal{L}(\mathbf{w}, \mathbf{e}_j, P_{s_j}) = \max_{\mathbf{e} \in \mathbf{E}_{s_j}} \psi(\mathbf{w}, \mathbf{e}, P_{s_j}) - \psi(\mathbf{w}, \mathbf{e}_j, P_{s_j}) + \Delta(\mathbf{e}, \mathbf{e}_j)$$

where $\Delta(\mathbf{e}, \mathbf{e}_j)$ is a non-negative function that is zero only when $\mathbf{e} = \mathbf{e}_j$. One could also use a multinomial probabilistic loss instead.

A second and simpler idea is to set up a binary classification problem in which $\mathbf{e}_j$ is treated as a positive example and each $\mathbf{e} \in \mathbf{E}_{s_j}$, $\mathbf{e} \neq \mathbf{e}_j$ is treated as a negative example. This creates imbalance, but it can be handled easily by down-weighting the loss associated with each negative example by the factor $1/(|\mathbf{E}_{s_j}| - 1)$. We will refer to this method as the *Classifier Method*.

Another effective method is to use ranking/ preference learning in which the true extractor $\mathbf{e}_{s_j}$ is paired with every other extractor $\mathbf{e} \in \mathbf{E}_{s_j}$ and, in the training formulation $\psi(\mathbf{w}, \mathbf{e}_j, P_{s_j})$ is pressured to be higher than $\psi(\mathbf{w}, \mathbf{e}, P_{s_j})$. A variant of this method is to form groups of extractors (excellent, good, fair, bad, etc.) and use only pairs of extractors between groups. We will refer to this method as the *Ranking Method*.

6. GENERATION OF CANDIDATE EXTRACTORS

The feasibility of the approach that we described in section 4 depends on the ability to generate, for each site s, a manageably small set of candidate site-level extractors, $\mathbf{E}_s$ that contains good extractors in it. In this section we point out one such generation method which is effective on many domains. This method is founded on the observation that, *on many domains, there exist some easily recallable attributes (anchors) for which candidate extractors are easy to form and, the remaining attributes are located in close proximity to the anchors.* Thus, we divide the attribute set A into two parts: *Anchor attributes* are those for which it is possible to form annotators with decent precision ($\mathcal{P}$) and recall ($\mathcal{R}$); the remaining attributes are *Non-Anchor attributes*. Note that precision $\mathcal{P}$ can be less than 1, and so annotations are allowed to be noisy.

Annotators for the anchor attributes are typically based on one of the following: regexes (e.g. date, phone number), dictionaries (e.g. states), language models/classifiers (e.g. title, description, review). Most domains have a few anchor attributes that can be easily identifiable. Table 1 shows in boldface the anchor attributes associated with the three domains, *Events*, *Books* and *Restaurants*. As mentioned above, the annotators can be noisy; thus, for example, if we take a regex for date and apply it to all detail pages containing events, it will also mark date occurrences other than the true *event dates*.

Let us now see how the annotators can be used to form extractors for the anchor attributes. We represent the site-level extractors for anchor attributes as xpaths. For a given site and each anchor attribute we choose a candidate set of (site-level) xpaths as follows.

Algorithm 1 *Choosing candidate xpaths for anchor attribute a in site s*

```
1: Use the annotator for attribute $a$ to find all occurrences (leaf
   nodes) on pages in $P_s$.
2: Make a list of all xpaths that correspond to these nodes. Dalvi
   et al. [3] give an algorithm for getting this list of xpaths.
3: Choose only those xpaths with #occurrences $\geq \tau|P_s|$.
```

Here τ is a fraction threshold that is set at the domain level; it could be set differently for different attributes depending on the quality of the annotators. If τ is too big, good xpaths may fail to get selected. On the other hand, if τ is too small, many useless xpaths can get selected which will affect the efficiency of our approach. We now provide a rough analysis to help set τ and understand what is needed on the quality of annotators.

Take one anchor attribute. Let n_+ be the number of pages in P_s that are detail pages of interest. It can be written as $n_+ = \rho|P_s|$ where the fraction ρ typically varies from 0.2 to 1 depending on the site s and how P_s is formed.

Typically, each attribute has a unique xpath associated with the detail pages of site s; this is the true xpath of interest to us. By assumption 1 (see section 2), n_+ is the maximum number of occurrences of the attribute in site P_s; assuming that the attribute is rarely missing in a detail page we can take n_+ as the number of occurrences associated with the true xpath. The number of correct annotations, $n_{correct}$ is given by $\mathcal{R}n_+ = \mathcal{R}\rho|P_s|$. To make sure that the desired true xpath for the attribute gets chosen by algorithm 1, we require $n_{correct} \geq \tau|P_s|$. This translates to the condition $\tau \leq \mathcal{R}\rho$. Using domain knowledge on $\mathcal{R}$ and ρ we can set τ suitably.

Though precision $\mathcal{P}$ of the annotators does not enter in the above analysis of τ, it plays a role in keeping the number of xpath candidates chosen by algorithm 1 to be small. Even with a precision of just 0.5, we typically find that the number of xpath candidates chosen by algorithm 1 is just a few, say about 2. This is because, wrong annotations tend to occur non-systematically on different pages and so the number of occurrences associated with their xpaths tend to be very small, thus getting discarded by algorithm 1.

Let us say we have chosen extractors (xpath candidates) for each anchor attribute. If all the attributes are anchor attributes, then the set of candidate extractors is the cross product of the sets of xpaths of the attributes. In case there is one or more non-anchor attributes, we choose extractors for these attributes utilizing the property that they occur in close proximity to anchor attributes.

One way of setting up the extractor for a non-anchor attribute is by attaching it with one anchor attribute and requiring that the non-anchor attribute is located at an (integer) offset δ from the anchor attribute occurrence determined by its xpath. Here, offset is defined as the forward (positive) or backward (negative) distance from the anchor attribute node on the sequence formed by the html leaf nodes. Most times domain knowledge tells us which anchor attribute to base a non-anchor attribute on. For example, in the *Event* domain, *Title*, *Location* and *Description* tend to occur within an absolute offset of 5 from *Date*. Thus we can select a window size μ (set at the domain level, e.g. $\mu = 5$) and, for each non-anchor attribute, consider all offsets δ satisfying $-\mu \leq \delta \leq \mu$. If available, we may also use domain knowledge on attribute ordering (e.g. *Description* always occurs after *Date*) to consider only offsets on one side (+ or -) of the associated anchor attribute.

If we don't have clear domain knowledge as to which anchor attribute to base a non-anchor attribute on, then we can connect that non-anchor attribute to each anchor attribute and consider all resulting offsets as candidates. Each (*anchor_xpath, offset*) combination defines an extractor for a non-anchor attribute. If m_a (m_{na}) is the number of anchor (non-anchor) attributes and n_{xpath} is the maximum number of xpaths for each anchor attribute, then the size of the candidate extractor set can be bounded as follows:

$$|E_s| \leq \begin{cases} (n_{xpath})^{m_a}(2\mu+1)^{m_{na}} & \text{if base anchor is clear} \\ (n_{xpath})^{m_a}(m_a(2\mu+1))^{m_{na}} & \text{else} \end{cases} \quad (5)$$

For the event domain, $m_a = 2$, $m_{na} = 3$ and we find the following values to work well: $\mu = 7$, $n_{xpath} = 2$. The size of the candidate extractor set is at most 13,500 to 108,000. With precedence knowledge these numbers decrease by a factor of eight. Use of other forms of domain knowledge can bring down the numbers even further. Also, with code optimization the score computation associated with different extractors can be made very fast.

7. EXPERIMENTS

7.1 Datasets

We use datasets from three domains: *Events*, *Books* and *Restaurants*. The *Events* dataset was formed by collecting event detail

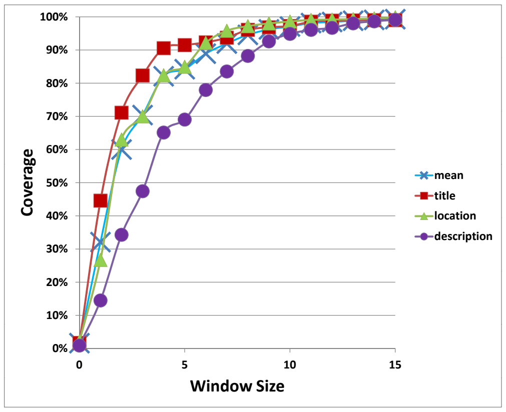

Figure 1: ***Events*****: Coverage of non-anchors for different window sizes**

pages from 247 sites having *Events* information such as schools, libraries, city halls and museums. Each site has at least 3 labeled detail pages, totally 1292 pages were labeled. This dataset is used to do basic experiments about our method. Ten fold cross validation (at the site level) is used to form the train/test data for comparing classifier and ranking models.

The *Books* and *Restaurants* datasets are exactly the ones used in Satpal et al. [5]; these datasets are used to compare our approach with methods based on graphical models. *Books* dataset has 5 websites and 3437 labeled pages; *Restaurants* dataset has 4 websites and 885 labeled pages. Given the small numbers of sites but large number of labeled pages for each site in these two datasets, we do the following two steps. (1) To increase training sites, we split each *Book* or *Restaurant* site into multiple sub-sites. Each *Book* sub-site has 46 detail pages on average and each *Restaurant* sub-site has 12 detail pages on average. After this process, 5 *Book* sites turn into 74 sub-sites and 4 *Restaurant* sites become 75 sub-sites; (2) the leave-one-out validation approach has been used for comparing methods; thus, each time one of the 5 original *Book* websites (or one of the 4 original *Restaurant* sites) is excluded as the test set.

7.2 Features

For the experiment on the comparison with other extraction methods involving *Books* and *Restaurants* datasets, we implemented exactly the same features as Satpal et al. [5] in *Books* and *Restaurants* domains. Details of these features can be found in that reference.

Let us now describe the features implemented in experiments on the *Events* domain. It is useful to describe these features since selection of good features is crucial for getting good performance in any domain.

The following features are based on page-level properties. Even though we only mention the page-level properties, when forming features at the site-level we aggregate these properties, e.g. the average value of a quantity over all pages in the site. The page-level features are similar to those page-level extraction approaches. Three of them are (1) Binary attributes that show whether a date segment matches with a date, time or location entity; (2) Average length range of a title or description; (3) Context feature. Some contextual words indicate the label of the segments, e.g. "Location: city hall".

The following properties are direct site-level aggregated features. (1) Pages extracted. The number of pages extracted. It is normal-

Table 2: Comparison of various learning algorithms

Method	Events	Books	Restaurants
Linear Regression	71.8%	91.3%	90.2%
Neural Network	67.3%	94.8%	92.7%
SVM Classification	57.0%	-	-
SVM Ranking	82.8%	93.0%	91.1%

Table 3: Comparison of various methods

Method	Books	Restaurants
MLN (MWS)	64.4%	68.7%
MLN (IHI)	88.4%	90.1%
Hierarchical CRF	60.0%	59.0%
Our Method	94.8%	92.7%

ized by the number of pages under this site; (2) Items extracted. The average number of items extracted per page under a site. It is normalized by the number of extracted pages; (3) Exceptions rate. Here "exceptions" refers to out-of-bound occurrences. For example, suppose *Date* exists on the top (the first segment) (call this position *DatePos*) and there is no segment in the position *DatePos-7*, then for offset -7 there is an exception; (4) Content variance. It takes low value whenever the text contents of the nodes are similar. This feature is useful since we expect non-static content across different records for a given attribute; (5) Xpath feature. It takes high value whenever the same types of attribute nodes share the same xpath, and low value otherwise.

7.3 Verification of two properties

A key property assumed by our approach is that the non-anchor attributes lie in close proximity to the xpaths chosen for the anchor attributes. We want to check if a chosen window size μ (see the discussion around (5)) covers all the non-anchor attributes. Figure 2 shows, for *Events*, the variation of the coverage of attributes as a function of the window size, μ. From this figure we can see that window size of 7 covers more than 90% of occurrences for attributes *Title* and *Location* and more than 80% for *Description*. In our implementation we use 7 as the default window size as a good balance between coverage and efficiency.

A necessary property for our approach is that the attributes are ordered consistently within each site. For *Events*, we evaluated the consistency with the majority order in websites. The *order consistency percentage* (*ocp*) for a site is computed as the average, over all pairs of attributes, of the percentage of web pages in the site consistent with the majority order for an attribute pair. 82.4% of websites have perfect attribute order and an additional 11.5% have $70\% \leq \text{ocp} < 100\%$; this is an indication of how well this property holds in domains.

7.4 Comparison of learning models

We compared different learning models on three domains: *Events*, *Books* and *Restaurants*. First consider "Events" column of Table 2 (from the second line to the end) for the results on the *Events* domain. Average F1-scores on features (see Table 1) are listed. There are the results of LR, NN, SVM classification and SVM ranking using full enumeration of the candidate wrappers. The SVM ranking model has the highest average F1 score. The SVM classification method gives relatively less accuracy. In general we found that, with SVMs, the classification method is not competitive to SVM ranking. Therefore, for the *Books* and *Restaurants* domains we do not provide the results of SVM classification. *Books* and *Restaurants* columns give the results on these two domains, respectively. Different from *Events* domain, LR, NN and SVM ranking all give similar good results. The results of MN are slightly better than the other two. As described in the description of datasets in Section 8.1, each *Events* website has (1292/247=) 5.2 detail pages on average, but each *Books* site has (3437/74=) 46.4 pages and each *Restaurants* site has (885/75=)11.8 pages. We can see from these numbers that there are much less annotated detail pages on each *Events* site. The SVM ranking model gives stable good performance irrespective of whether a site has enough annotated pages or not. On the other hand, LR and MN perform better when the number of annotated pages per site is sufficiently large.

7.5 Comparison with graphical model methods

We compare our approach (Neural Network model) with the graphical model based methods: the MLN method of Satpal et al. [5] and the Hierarchical CRF method of Zhu et al. [7]. We obtained *Books* and *Restaurants* datasets from the authors of Satpal et al. [5] and used exactly the same features as used by them, for our approach. Satpal et al. [5] implemented two inference schemes: a generic MaxWalkSat (MWS) scheme and an improved heuristic inference (IHI) scheme; they also implemented and compared their method with Hierarchical CRF on these datasets. Given the identical evaluation scenario, we directly borrow the results for the graphical model methods from Satpal et al. [5] and compare them with those of our method; see Table 3. Looking at the Average-F1 values it is clear that Hierarchical CRF does not do well; this is mainly due to the lack of complex factors connecting distant labels. With complex factors MLN performs very well; from a comparison of MLN (MWS) and MLN (IHI) it is clear that careful inference is crucial for MLN to perform well. Our method outperforms these state-of-the-art methods at the Average-F1 value.

8. CONCLUSION

In this paper we have given a new approach to large scale domain-centric extraction. This approach directly chooses attribute extractors for a site, using a scoring mechanism that is designed at the domain level via simple classification methods. Experiments on three domains show that our approach is competitive with complex, graphical models based approaches, while being a lot simpler.

9. REFERENCES

[1] M. Alvarez, A. Pan, J. Raposo, F. Bellas and F. Cacheda. Using clustering and edit distance techniques for automatic web data extraction. In WISE, 2007.

[2] K. Crammer, O. Dekel, J. Keshet, S. Shalev-Shwartz and Y. Singer, Online passive-aggressive algorithms. JMLR, vol 7, pp. 551-585, 2006.

[3] N. Dalvi, R. Kumar and M. Soliman, Automatic wrappers for large scale web extraction. Proceedings of the VLDB Endowment, vol 4, issue 4, pp. 219-230, January 2011.

[4] S. Sarawagi, Information extraction. Foundations and trends in databases, vol 1, pp. 261-377, 2008.

[5] S. Satpal, S. Bhadra, S. Sundararajan, R. Rastogi and P. Sen, Web information extraction using Markov logic networks. In KDD 2011.

[6] J. M. Yang, R. Cai, Y. Wang, J. Zhu, L. Zhang and W. Y. Ma, Incorporating site-level knowledge to extract structured data from web forums. In WWW, 2009.

[7] J. Zhu, Z. Nie, J. Wen, B. Zhang and J. Wen, Simultaneous record detection and attribute labeling in web data extraction. In KDD 2006.