

# Exploiting Structured Reference Data for Unsupervised Text Segmentation with Conditional Random Fields

Chang Zhao    Jalal Mahmud    I.V. Ramakrishnan  
Computer Science Dept.  
Stony Brook University  
Stony Brook, NY, 11794, USA  
{changz,jmahmud,ram}@cs.sunysb.edu

## Abstract

Text segmentation is the process of converting information in unstructured text into structured records. This is an important problem since structured data is amenable to efficient query processing. CRFs are a class of discriminative probabilistic models that are gaining acceptance as an effective computing machinery for text segmentation. An important aspect of CRFs is learning model parameters from labeled training data. Labeling can be a labor intensive process. One can avoid the labeling step by using structured reference tables whose data domains and that of the input text data given for segmentation, coincide. In other words the labels in the training data drawn from reference tables “come for free”. Inspired by recent work on their use for training HMMs, we developed an unsupervised technique for text segmentation with CRFs using reference tables. Assuming text sequences to be segmented come in batches and sequences in a batch conform to the same attribute order, we build CRF models for each attribute in the reference table, use them to decide the attribute order of a batch of input sequences, derive labeled training data from the reference table according to that order, and train a global CRF model to segment the input sequences in the batch. Preliminary experimental results indicate that our technique works well in practice.

## 1 Introduction

Text segmentation is the process of converting information in plain text strings into structured records. Given a schema consisting of  $n$  attributes and an input string, the problem of segmenting the input string can be informally defined as partitioning the string into  $n$  contiguous sub-strings and assigning each sub-string a unique attribute from the  $n$  attributes. For example, given the address schema consisting of the five attributes  $\langle$ COMPANY, STREET, CITY, STATE, PHONE $\rangle$  and

the input string “1 2 3 Convenience Store (516)538-0854 144 Hempstead Tpke W Hempstead NY”, the task of text segmentation is to convert the string into the address record:  $\langle$ 1 2 3 Convenience Store, 144 Hempstead Tpke, W Hempstead, NY, (516)538-0854 $\rangle$ .

In the World Wide Web, data (such as product, bibliographic and address data) exists as unstructured text strings. They have to be segmented into structured records to facilitate efficient query processing and analysis. Therefore accurate text segmentation methods are important.

Extant techniques for text segmentation either use rules for identifying attributes in the text or employ statistical models. Rule-based approaches require domain experts to create and maintain a set of rules for each application domain. It is difficult to anticipate all possible variations in the text strings to be segmented and design rules accordingly. This difficulty is further compounded by the presence of noise in the data. Therefore rule-based approaches are neither scalable nor robust. In contrast statistical approaches automatically learn a statistical model for each application domain. The variability and noise in the input text data are elegantly dealt with by the statistical characteristics inherent in such approaches.

Hidden Markov Model (HMM) [18] is a dominant statistical model used in text segmentation. HMM is a generative model in the sense that it captures the probability distribution of observations (e.g. the input strings in the case of text segmentation). There are two main problems with HMM-based text segmentation. Firstly, in order to estimate the distributions of observations HMMs will need to enumerate all possible observations. This may not always be possible. Secondly, as articulated in [10], HMMs have to make strict independence assumptions to achieve computational tractability and hence cannot capture long-range dependences in the input data.

To address the above two shortcomings of HMMs, Conditional Random Fields (CRFs) were introduced in [10] for sequential labeling problems. Note that text segmentation is an application of this problem. There have been a number of recent works on CRFs (see [21, 23, 17, 15, 12]). CRF is a discriminative model that directly computes the conditional probability of a label sequence given an observation sequence. Therefore, it does not need to capture the probability distribution of observations. The other important aspect of CRFs is that they can capture long range dependence in data [10]. Implementations of CRFs have been shown to outperform generative models like HMMs [7, 20] for text segmentation, especially when the data exhibits long range dependencies.

Text segmentation using statistical models are typically supervised, that is, the model is supplied with manually labeled training data. This, in general, is a labor-intensive process. Unsupervised learning techniques eliminate the need for manually labeled data.

Recently a fully automatic, unsupervised text segmentation system is described in [4]. This system exploits structured reference tables consisting of clean tuples. In other words the attributes in these tables are already labeled.

Table 1 shows a fragment of a reference table. Reference tables can contain a large number of records thereby providing a rich source of labeled training data. Note that the order in which the attributes appear in the table might be different from that in which they appear in the input sequences. For example, the order in which attributes appear in bibliography data to be segmented may be [AUTHOR, TITLE, PUBLISHER, PAGE, YEAR] while in the reference table it may be [TITLE, AUTHOR, PUBLISHER, PAGE, YEAR]. So an unsupervised text segmentation system will have to deal with differences in the attribute order of the input sequences. Assuming that a batch of text sequences to be segmented share the same total attribute order (e.g. publications in a researcher’s home page), the technique in [4] first trains an HMM model for each attribute using the reference table data. Next it uses these trained HMM models to identify the best starting positions for every attribute in every input sequence. Then it uses these positions to infer the common total order. Finally the total order is used to construct a global HMM to segment the input text data sequences.

Although CRFs have recently been used for text segmentation, they are by and large supervised approaches. A recent work on CRF-based text segmentation [13] focuses on reducing the training data using reference tables but does not completely eliminate their use. In this paper, inspired by the the work in [4], we

propose a CRF-based unsupervised text segmentation technique using reference tables. Using CRFs for this problem poses some challenges. The main difficulty is inferring the total order. In HMMs this is not an issue. Being a generative model HMM can easily compute the marginal distribution of observations. In particular suppose  $P(\mathbf{o})$  is the marginal distribution of an observation sequence  $\mathbf{o}$ . Given two substrings  $\mathbf{s}_1$  and  $\mathbf{s}_2$  and the attribute HMM trained to recognize instances of *Attr*, we can readily determine which one is more likely to be an instance of *Attr* by simply comparing  $P(\mathbf{s}_1)$  and  $P(\mathbf{s}_2)$ .

Since CRFs do not model distributions of observations we will have to develop a new technique for inferring the total order. In this paper we present such a technique. We introduce negative labels and include negatively labeled examples in the training of attribute CRF models. An important aspect of our approach is the process underlying the generation of these examples so as to ensure that the attribute CRF will assign low likelihood scores to incorrect starting positions of an attribute in the input sequence.

The rest of the paper is organized as follows. Section 2 presents an overview of conditional random fields to set the context for understanding the rest of the paper. Section 3.1 describes how to build attribute CRFs from a reference table and Section 3.2 presents our solution to text segmentation with the attribute CRFs. Section 4 reports our experiment results on address, product and bibliographic datasets. Related work appears in Section 5 and Section 6 concludes the paper.

## 2 Preliminaries

In this section we provide an overview of Conditional Random Fields (CRFs) that underlies our technique described in the next section. We begin with notations that will be used in this paper.

**2.1 Notations** Conditional Random Field (CRF) [10] is a probabilistic framework that can be used to segment and label sequence data.

The input to the segmentation task consists of a sequence of tokens where what constitutes a token is application-specific. For example, in bioinformatics application a token is usually a letter whereas in the address segmentation problem tokens are words that are delimited by whitespaces. We will therefore assume that there is an application-specific *tok* function that maps any input to a *token sequence*. For example, *tok*(“1 2 3 convenience store 144 Hempstead Tpke W Hempstead NY”) is the sequence of 11 tokens: [1, 2, 3, convenience, store, 144, Hempstead, Tpke, W, Hempstead, NY].

For a token sequence  $[t_1, \dots, t_{len}]$ , a *token sub-*

| BUSINESS                     | STREET               | CITY      | STATE | PHONE         |
|------------------------------|----------------------|-----------|-------|---------------|
| 1 Hour Auto Glass Inc        | 403 West St          | New York  | NY    | (212)691-3344 |
| 1 Hundred 60 4th St Auto Svc | 8412 164th St        | Jamaica   | NY    | (718)523-9018 |
| 10 Minute Oil Change         | 1156 Hempstead Tpke  | Uniondale | NY    | (516)486-0060 |
| A Salerno Realty Crop        | 11 Mill              | Rhinebeck | NY    | (914)876-5551 |
| Circuit City                 | 111 E El Camino Real | Sunnyvale | CA    | (408)720-1043 |

Table 1: A Segment of a Reference Table

sequence  $sub(i, j)$  is defined to be  $[t_i, t_{i+1}, \dots, t_j]$  for any  $1 \leq i \leq j \leq len$ . Note the tokens in a token sub-sequence are contiguous. Therefore, for a token sequence of  $len$  tokens, the number of token sub-sequences is  $len + (len - 1) + \dots + 1 = len(len + 1)/2$ .

Segmenting a token sequence  $[t_1, \dots, t_{len}]$  with CRFs amounts to assigning a label sequence  $[l_1, \dots, l_{len}]$  where each  $l_i$  ( $1 \leq i \leq len$ ) is assigned to token  $t_i$  from a predefined label set. These labels correspond to attribute names. Therefore token sub-sequences might correspond to instances of attributes. For example, if the label sequence [STREET, STREET, STREET, CITY, CITY] is assigned to the input sequence [144, Hempstead, Tpke, New, York], then the token sub-sequence [144, Hempstead, Tpke] corresponds to an instance of the attribute STREET and [New, York] corresponds to an instance of the attribute CITY.

Throughout this paper, we use bold fonts to denote vectors, such as  $\mathbf{x}$  for an input token sequence, and  $\mathbf{y}$  for a label sequence. Normal fonts denote scalars, such as  $x$  and  $y$  for a single token and label respectively.

**2.2 Conditional Random Fields** CRF is a discriminative model in the sense that it directly computes the conditional probability distribution of label sequences  $\mathbf{y}$  given a particular input token sequence  $\mathbf{x}$ . In contrast generative models such as HMMs compute a joint distribution over both label and token sequences.

The conditional probability distribution of label sequences given an input token sequence is defined by a set of features capturing transitions between labels as well as relations between a label and the corresponding token it is assigned to, the other tokens in the neighborhood of this corresponding token, or even the entire token sequence.

We follow the notations in [13] in the rest of this section. CRF features are boolean functions in the form of  $f(y_{i-1}, y_i, \mathbf{x}, i) \mapsto \{0, 1\}$  where  $y_{i-1}$  is the  $(i - 1)$ -th label,  $y_i$  is the  $i$ -th label, and  $\mathbf{x}$  is the token sequence.

Let  $[[c]]$  be the indicator function whose value is 1 when the condition  $c$  is satisfied and 0 otherwise. Examples of how features are expressed using indicator functions are shown below. Feature (2.1) captures

the transition from STREET to CITY. Feature (2.2) captures the relation between the word “NY” and the label STATE.

$$(2.1) \quad f(y_{i-1}, y_i, \mathbf{x}, i) = [[y_i = CITY \text{ and } y_{i-1} = STREET]].$$

$$(2.2) \quad f(y_{i-1}, y_i, \mathbf{x}, i) = [[x_i \text{ is NY and } y_i = STATE]].$$

Let us denote the vector of all feature functions by  $\mathbf{f}$ . A vector  $\lambda$  of real numbers with the same length as  $\mathbf{f}$  defines the weights of the feature functions.

The feature function vector  $\mathbf{f}$  and the corresponding weight vector  $\lambda$  are used to define the conditional probability distribution over label sequences  $\mathbf{y}$  given an input token sequence  $\mathbf{x}$  as follows:

$$(2.3) \quad P(\mathbf{y}|\mathbf{x}) = \frac{1}{Z(\mathbf{x})} e^{\sum_{i=1}^{|\mathbf{x}|} \sum_{j=1}^{|\mathbf{f}|} \lambda_j f_j(y_{i-1}, y_i, \mathbf{x}, i)},$$

where  $|\mathbf{x}|$  is the length of  $\mathbf{x}$ ,  $|\mathbf{f}|$  is the number of feature functions, and  $Z(\mathbf{x})$  is a normalizing factor equal to

$$\sum_{\mathbf{y}'} e^{\sum_{i=1}^{|\mathbf{x}|} \sum_{j=1}^{|\mathbf{f}|} \lambda_j f_j(y'_{i-1}, y'_i, \mathbf{x}, i)}.$$

Therefore,

$$\sum_{\mathbf{y}'} P(\mathbf{y}'|\mathbf{x}) = 1$$

for all label sequence  $\mathbf{y}'$  of token sequence  $\mathbf{x}$ , which means the conditional probabilities of all possible label sequences for a given token sequence sum up to 1.

For a given token sequence  $\mathbf{x}$ , CRFs compute the label sequence  $\mathbf{y}$  with the highest conditional probability  $P(\mathbf{y}|\mathbf{x})$  as the best label sequence, i.e.,  $\arg \max_{\mathbf{y}} P(\mathbf{y}|\mathbf{x})$ . An important aspect of CRFs is to learn a CRF model from labeled token sequences, i.e. the set of paired token and label sequences  $\{\langle \mathbf{x}_1, \mathbf{y}_1 \rangle, \dots, \langle \mathbf{x}_m, \mathbf{y}_m \rangle\}$  (example see Figure 1). Usually feature functions are assumed to be given and therefore learning corresponds to estimating the weights of feature functions. Details of CRF inference and learning algorithms can be found in the seminal work of [10] as well as in [21, 16].

For an exposition of how CRFs segment token sequences, let us look at a simplified example. Suppose we have trained a CRF model from the training data in Figure 1. The feature functions and their weights are

```
{
<[Stony, Brook, NY], [CITY, CITY, STATE]>,
<[Port, Jefferson, NY], [CITY, CITY, STATE]>
}
```

Figure 1: A Small Set of Training Examples

listed in Table 2. For this CRF, the length of the feature function vector is 9, i.e.,  $|\mathbf{f}| = 9$ .

| Feature Function                                      | Weight |
|---|--------|
| $[[x_i \text{ is Stony and } y_i = \text{CITY}]]$     | 0.44   |
| $[[x_i \text{ is Brook and } y_i = \text{CITY}]]$     | 0.73   |
| $[[x_i \text{ is Port and } y_i = \text{CITY}]]$      | 0.44   |
| $[[x_i \text{ is Jefferson and } y_i = \text{CITY}]]$ | 0.73   |
| $[[x_i \text{ is NY and } y_i = \text{STATE}]]$       | 2.16   |
| $[[y_{i-1}=\text{CITY and } y_i = \text{CITY}]]$      | 0.29   |
| $[[y_{i-1}=\text{CITY and } y_i = \text{STATE}]]$     | 2.06   |
| $[[\text{start} = \text{CITY}]]$                      | 0.87   |
| $[[\text{end} = \text{STATE}]]$                       | 2.16   |

Table 2: Example CRF with Two Labels

There are four possible label sequences (see the 1st two columns in Table 3 for the token sequence  $\mathbf{x} = [\text{Huntington, NY}]$ ). The sum of their weighted feature functions,  $F = \sum_{i=1}^2 \sum_{j=1}^9 \lambda_j f_j(y_{i-1}, y_i, \mathbf{x}, i)$  and conditional probability  $P(\mathbf{y}|\mathbf{x})$  computed from Equation 2.3 are also listed in the 3rd and 5th columns respectively of the table. For example, the conditional probability of the label sequence  $[\text{CITY, CITY}]$  is  $3.2 / (3.2 + 1408.1 + 0 + 75.2) = 0.2\%$ . From the table we see that the best way of labeling  $[\text{Huntington NY}]$  is to assign the label CITY to token Huntington and STATE to NY.

| Huntington | NY    | $F$  | $e^F$  | $P(\mathbf{y} \mathbf{x})$ |
|------------|-------|------|--------|----------------------------|
| CITY       | CITY  | 1.16 | 3.2    | 0.2%                       |
| CITY       | STATE | 7.25 | 1408.1 | 94.7%                      |
| STATE      | CITY  | 0    | 0      | 0%                         |
| STATE      | STATE | 4.32 | 75.2   | 5.1%                       |

Table 3: Possible Ways of Labeling  $[\text{Huntington, NY}]$

### 3 Unsupervised CRF-based Text Segmentation with Reference Tables

A reference table is a relational table whose columns, particularly column names, correspond to labels that are to be assigned to tokens in test sequences. Let us assume  $\langle a_1, a_2, \dots, a_n \rangle$  are the columns. We use columns and column names interchangeably. Our method of exploiting the reference table for unsupervised text segmentation with CRF is based on the observation that test token sequences usually come in batches with an

1. **B & D Bar Restaurant Supply Company** <sup>®</sup>  
 (631) 689-0578 10 Seville Ln, Stony Brook, NY 0.99 mi  
[Map](#) | [Directions](#) | [Send to Phone](#) | [Save to Collection](#)

See all: [Agriculture Supplies & Equipment](#)

2. **Yin & Yang Restaurant** <sup>®</sup>  
 (631) 689-8585 2548 Nesconset Hwy, Stony Brook, NY 1.55 mi  
[Map](#) | [Directions](#) | [Send to Phone](#) | [Save to Collection](#)

...One of the better buffet chinese **restaurants** here on Long Island. And they do serve some... [more](#)  
 See all: [Restaurants](#)

3. **Friendly's Restaurant** <sup>®</sup>  
 (516) 751-3150 201 Hallock Rd, Stony Brook, NY 1.99 mi  
[Map](#) | [Directions](#) | [Send to Phone](#) | [Save to Collection](#)

...Friendlys - Delicious food, Premium Ice Cream and Magical Moments with Friends and Family  
 Friendlys... [more on web site](#)  
 See all: [Continental Restaurants](#) - [Salad Restaurants](#) - [Cafes](#) - [Restaurants](#) - [American Restaurants](#)  
[www.friendlys.com/](#)

4. **Country House** <sup>®</sup>  
 (631) 751-3332 Route 25a & Main St, Stony Brook, NY 0.56 mi  
[Map](#) | [Directions](#) | [Send to Phone](#) | [Save to Collection](#)

... in a tasteful manner, for both the food and the ambience, is what this **restaurant** is all... [more](#)  
 See all: [Restaurants](#)  
[www.countryhouserestaurant.com/](#)

Figure 2: Restaurant addresses presented in the same attribute order

attribute order common to all the sequences in a batch. Examples include product data description in an online vendor catalog such as Officemax and Staples.

Let  $\{\mathbf{s}_1, \mathbf{s}_2, \dots, \mathbf{s}_m\}$  denote a batch consisting of  $m$  test sequences with  $\mathbf{s}_i$  being the  $i$ 'th token sequence. Let us further assume that the common attribute order for this batch is:  $a'_1 \rightarrow a'_2 \rightarrow \dots \rightarrow a'_n$  where  $\langle a'_1, a'_2, \dots, a'_n \rangle$  is a permutation of  $\langle a_1, a_2, \dots, a_n \rangle$ . The overall idea of segmenting this batch is as follows:

1. We first build a CRF for each column  $a_i (1 \leq i \leq n)$  of the reference table. We will call it the *attribute CRF*. The training of attribute CRFs is *fully automatic*.
2. The attribute CRFs are used to compute the most likely starting position (in a probabilistic sense) of an attribute instance in each test sequence.
3. The results obtained in the previous step are combined to infer the common order  $a'_1 \rightarrow a'_2 \rightarrow \dots \rightarrow a'_n$  of the sequences in the batch.
4. In the final step we derive labeled training examples from the reference table according to the inferred common order and train a global CRF model for text segmentation using the known techniques [10, 21].

From now on, we will refer to the attribute CRF corresponding to the attribute Attr by Attr CRF.

The aforementioned process is illustrated by the following example. Figure 2 shows a fragment of the

```

{
[B, &, D, Bar, Restaurant, Supply, Company, 631,
689, 0578, 10, Seville, Ln, Stony, Brook, NY],
[Yin, &, Yang, Restaurant, 631, 689, 8585, 2548,
Nesconset, Hwy, Stony, Brook, NY],
[Friendly's, Restaurant, 516, 751, 3150, 201,
Hallock, Rd, Stony, Brook, NY],
[Country, House, 631, 751, 3332, Route, 25a, &,
Main, St, Stony, Brook, NY]
}

```

Figure 3: Example Test Token Sequences

search results returned by Yahoo Local in response to a query about restaurants in the neighborhood of zip code 11790.

The first entry in Figure 2 starts with the name of a BUSINESS “B & D Bar Restaurant Supply Company”, followed by its PHONE “(631)689-0578”, STREET “10 Seville Ln”, CITY “Stony Brook”, and STATE “NY”. The common order of the attributes for this search result fragment is: BUSINESS → PHONE → STREET → CITY → STATE.

We assume that the search result is preprocessed and tokenized. Our task now is to segment the four token sequences shown in Figure 3 and assign labels to the segments from the label set {BUSINESS, PHONE, STREET, CITY, STATE}.

An example reference table that can be used to segment token sequences like those in Figure 3 is shown in Table 1. We use the reference table to automatically train the 5 attribute CRFs: one each for BUSINESS, STREET, CITY, STATE, and PHONE. Those attribute CRFs will be deployed in tandem to infer the common attribute order in the batch consisting of the sequences in Figure 2, namely: BUSINESS → PHONE → STREET → CITY → STATE. Then labeled training examples are derived automatically from the reference table, Table 1, according to this inferred order. A global CRF is then trained from such examples and used to segment the test token sequences using standard CRF techniques [10, 21].

The idea of exploiting reference tables for unsupervised text segmentation was first explored in [4] using HMMs. Applying this idea to CRF is not entirely straight-forward. In the following section we first discuss the challenges and then present our solution.

### 3.1 Attribute CRF

**3.1.1 Issues in Training Attribute CRFs** Recall that the primary objective for building attribute CRFs is to infer the total order of attributes that is common to all the test sequences in a batch. For each test sequence,

we compute the most likely starting position for its attributes. These positions impose a local precedence relation  $\prec_{local}$  on pairs of attributes. Specifically, we say that  $a_i \prec_{local} a_j$  whenever  $p$  and  $q$  are the most likely starting positions for  $a_i$  and  $a_j$  respectively and  $p < q$ . If  $a_i \prec_{local} a_j$  in a majority of the test sequences then we say that  $a_i$  precedes  $a_j$  in the common total order, denoted  $\prec_{global}$ .

Thus, an attribute CRF for the attribute  $Attr$  should be able to identify the most likely token sub-sequence corresponding to  $Attr$ 's occurrence in the test sequence. However, training an attribute CRF to do such an identification is not as simple as taking all instances of  $Attr$  in the reference table and training the  $Attr$  CRF model using standard CRF training algorithms.

For an exposition of the underlying issues, let us revisit the example in Section 2 and examine the training of CITY CRF. If we train it only from the attribute instances of CITY, then no matter what the input token sequence is, the only way of labeling is to assign the label CITY to every token. Therefore, the conditional probability of labeling the token sub-sequence [Huntington] with CITY is the same as that of labeling [NY] with CITY, both of which are 1. It is hence impossible to decide whether [Huntington] or [NY] is the more likely instance of the attribute CITY.

In HMM-based attribute models as described in [4], this is not a problem. HMMs model the joint probability  $P(\mathbf{x}, \mathbf{y})$ ,  $\mathbf{x}$  being the token sequence and  $\mathbf{y}$  the label sequence. Note that a label sequence corresponds to a state sequence in HMM terminology. The marginal probability  $P(\mathbf{x}) = \sum_{\mathbf{y}} P(\mathbf{x}, \mathbf{y})$ , computed by the classic forward algorithm [18], can be used for the purpose of deciding the best starting position of an attribute instance. Specifically, given an attribute HMM trained from the attribute's instances in the reference table, deciding whether the token sub-sequence  $\mathbf{x}_1$  or  $\mathbf{x}_2$  fits the attribute better is simply done by comparing their marginal probabilities  $P(\mathbf{x}_1)$  and  $P(\mathbf{x}_2)$ , and picking the one with the higher probability value.

In contrast CRF is a discriminative model, i.e., given the token sequence  $\mathbf{x}$ , it directly models the conditional probability  $P(\mathbf{y}|\mathbf{x})$  of the label sequence  $\mathbf{y}$ . Since CRFs do not model probability distributions of observation (i.e. token) sequences, one cannot compute their marginal probabilities. In particular  $\sum_{\mathbf{y}} P(\mathbf{y}|\mathbf{x})$  is always 1 no matter what  $\mathbf{x}$  is and therefore it cannot serve as a criteria for deciding which token sub-sequence best fits an attribute.

### 3.1.2 Negative Labels: A Substitution for Marginalization

To solve the problem of using CRFs

for identifying the most likely token sub-sequence as an attribute instance, we introduce negative labels and include negatively labeled examples in the training of attribute CRF models.

We associate each attribute CRF with two labels – positive and negative. For example the labels associated with the CITY CRF are CITY and –CITY. Instances of an attribute in the reference table are assigned the positive label. So instances of CITY constitute the positive examples for the CITY CRF.

We assign the negative label to instances of all the other attributes in the reference table. So instances of BUSINESS, STREET, STATE and PHONE in Table 1 will be assigned the label –CITY.

However, these alone will not suffice as negatively labeled examples. Recall that an attribute CRF is required to choose from among all the token sub-sequences of a test sequence the most likely instance of the attribute. So for instance given the token sequence [B, &, D, Bar, Restaurant, Supply, Company, 631, 689, 0578, 10, Seville, Ln, Stony, Brook, NY], ideally CITY CRF should assign the label –CITY not only to tokens in the token sub-sequence [10, Seville, Ln] which is an instance of STREET, but also to the three token sub-sequences [Stony], [Stony, Brook, NY], and [Company, 631, 689, 0578, 10, Seville]. These are readily characterized by missing tokens (such as Brook in [Stony]) or with extra tokens (e.g. NY and Seville in the 2nd and 3rd token sub-sequences respectively). Therefore we should include different kinds of negative examples so that the attribute CRF can assign low attribute association likelihood to the token sub-sequences in the aforementioned cases.

Taking into consideration the discussion of the issues above, the process of generating negatively labeled examples for an attribute CRF is as follows: Suppose Attr is an attribute. A straightforward way to generate negatively labeled examples for Attr CRF is to simply form all possible combinations of token sub-sequences from the reference table and eliminate those that are instances of Attr. Clearly this is an expensive proposition. Instead we use the following simple but efficient heuristic rules:

- R1) token sequences resulting from deletion of the first token from any instance of Attr.
- R2) token sequences resulting from deletion of the last token from any instance of Attr.
- R3) token sequences obtained by prefixing any instance of Attr with the last token of any instance of any attribute other than Attr.
- R4) token sequences obtained by suffixing any instance

|       |   |
|-------|---|
| By R1 | [York]  |
| By R2 | [New]   |
| By R3 | [Inc, New, York], [St, New York],<br>[NY, New, York], [3344, New, York] |
| By R4 | [New, York, 1], [New, York, 403],<br>[New, York, NY], [New, York, 212]  |

Table 4: Negatively Labeled Examples

of Attr with the first token of any instance of any attribute other than Attr.

Table 4 shows examples assigned the label –CITY that are generated by the application of the above heuristic to the 1st tuple in Table 1. The 1st column in Table 4 indicates the rule used to generate the sequences in the row.

The positive and negative examples generated by the process described above is used to train an attribute CRF using well known CRF training algorithms [10, 21].

Armed with the trained CRF for an attribute say Attr, the probability of a token sequence  $\mathbf{s}$  being an instance of Attr is:

$$(3.4) \quad \frac{P(p|\mathbf{s})}{P(p|\mathbf{s}) + P(n|\mathbf{s})}$$

where  $P(p|\mathbf{s})$  is the conditional probability, computed by Attr CRF, for labeling all the tokens in  $\mathbf{s}$  with the positive label Attr and  $P(n|\mathbf{s})$  is the conditional probability for labeling all the tokens in  $\mathbf{s}$  with the negative label –Attr.

The idea of using negative examples in CRFs was first explored in [13]. But there are two differences with our approach. Firstly, CRFs with negative labels in [13] are used to define additional feature functions. Manually labeled examples, although fewer in number, are still required. In our case, we use such CRFs to infer a total order of attributes shared by a batch of test sequences, based on which we develop an unsupervised approach for training CRFs from reference tables. Secondly, as described previously in this section, the negative examples of our attribute CRFs are constructed subtly (see Rules 1, 2, 3 and 4) so as to ensure that incorrect token sub-sequences are given a very low score.

**3.2 Unsupervised Text Segmentation with Attribute CRF** Following [4], we also assume that the batch of test sequences share a common attribute order. The first step in inferring this order is to compute, with attribute CRFs, the most likely starting position (in a probabilistic sense) of an attribute instance in each test sequence.

### 3.2.1 Computing Most Likely Starting Positions of Attribute Instances

We associate two vectors  $\mathbf{v}_k$  and  $\mathbf{score}_k$  with the  $k$ -th test sequence in the batch  $[\mathbf{s}_1, \dots, \mathbf{s}_m]$ . The  $k$ -th test sequence  $\mathbf{s}_k$  is denoted by  $[t(k)_1, \dots, t(k)_{len_k}]$  where  $len_k$  is the number of tokens in  $\mathbf{s}_k$ . Given that  $a_1, \dots, a_n$  are the attributes in the reference table, both vectors have length  $n$ . The  $i$ -th element in  $\mathbf{v}_k$  is the most likely starting position in the test sequence for attribute  $a_i$ , and  $score_{k_i}$  denotes the probability of its likelihood. Algorithm *BestStartingPosition* is a sketch of how the most likely starting positions are computed.

**Algorithm *BestStartingPosition***

```

1. for  $i \leftarrow 1$  to  $n$ 
2.   do  $v_{k_i} \leftarrow -1$ ;
3.    $score_{k_i} \leftarrow 0$ 
4. for  $i \leftarrow 1$  to  $len_k$ 
5.   for  $j \leftarrow i$  to  $len_k$ 
6.     do  $\mathbf{s} \leftarrow [t(k)_i, \dots, t(k)_j]$ 
7.     for  $l \leftarrow 1$  to  $n$ 
8.       do  $p \leftarrow$  probability of  $\mathbf{s}$  being an instance
          of attribute  $a_l$  by Equation 3.4 using
           $a_l$ 's attribute CRF
9.       if  $p > score_{k_l}$ 
10.        then
11.           $v_{k_l} \leftarrow i$ ;
12.           $score_{k_l} \leftarrow p$ ;
```

Line 1 to 3 is initializing  $\mathbf{score}_k$  and  $\mathbf{v}_k$ . Line 4-5 forms a loop over all token sub-sequences of the test sequence. The token sub-sequence is assigned to  $\mathbf{s}$  in Line 6. Line 7 loops over all attributes. Line 8-12 updates the most likely starting position in  $\mathbf{v}_k$  and the associated probability in  $\mathbf{score}_k$  for each attribute and each token sub-sequence.

### 3.2.2 Inferring the Common Total Order

For a batch of  $m$  test sequences, we say that  $a_i \prec_{local} a_j$  in the  $k$ -th test sequence if  $v_{k_i} < v_{k_j}$ , otherwise  $a_j \prec_{local} a_i$ . Once the best starting positions are computed by Algorithm *BestStartingPosition*, we associate a global vote count  $votes_{i,j}$  for each pair of attributes  $a_i$  and  $a_j$  ( $1 \leq i, j \leq n$  and  $i \neq j$ ). This vote count corresponds to the number of test sequences in which  $a_i \prec_{local} a_j$ . If in a majority of test sequences  $a_i \prec_{local} a_j$  holds then we say that  $a_i \prec_{global} a_j$ . The rationale is that, in most cases the attribute CRFs can approximately recognize the correct starting position for an attribute among all possible token sub-sequences of a test sequence. Notice that for each pair of attributes  $a_i$  and  $a_j$ , we either have  $a_i \prec_{global} a_j$  or  $a_j \prec_{global} a_i$ . This  $\prec_{global}$  relation forms a directed graph. When there is no cycle in the graph, we can find a total order of the attributes. When a cycle exists, we break the cycle by removing an arbitrary edge in the cycle.

Algorithm *InferOrder* below sketches these ideas,

where  $R$  is the  $\prec_{global}$  relation,  $S$  is the attribute set, and  $LIST$  is a list representing the total order.

**Algorithm *InferOrder***

```

1. for  $i \leftarrow 1$  to  $n$ 
2.   for  $j \leftarrow 1$  to  $n$ 
3.      $votes_{i,j} \leftarrow 0$ 
4. for  $k \leftarrow 1$  to  $m$ 
5.   compute  $\mathbf{v}_k$  by Algorithm BestStartingPosition
6.   for  $i \leftarrow 1$  to  $n-1$ 
7.     for  $j \leftarrow i+1$  to  $n$ 
8.       if  $v_{k_i} < v_{k_j}$ 
9.         then  $votes_{i,j} ++$ 
10.        else  $votes_{j,i} ++$ 
11.  $R \leftarrow$  empty set
12. for  $i \leftarrow 1$  to  $n-1$ 
13.   for  $j \leftarrow i+1$  to  $n$ 
14.     if  $votes_{i,j} > votes_{j,i}$ 
15.       then add  $a_i \prec_{global} a_j$  to  $R$ 
16.       else add  $a_j \prec_{global} a_i$  to  $R$ 
17.  $S \leftarrow \{a_1, \dots, a_n\}$ 
18.  $LIST \leftarrow$  empty list
19. repeat
20.   if there is an attribute  $a$  so that there is no  $a' \prec_{global} a$ 
      for any attribute  $a'$  in  $R$ 
21.     then
22.       append  $a$  to  $LIST$ 
23.       delete  $a$  from  $S$ 
24.       delete all  $a \prec_{global} a'$  for any attribute  $a'$  from
           $R$ 
25.     else there is a cycle in the precedence relation; break
          the cycle by removing an arbitrary  $a \prec_{global} a'$ 
          that is part of the cycle
26. until  $S$  is empty
```

Line 1-3 initializes  $\mathbf{votes}$ . Line 4-10 computes  $votes_{i,j}$  for each pair of attributes  $a_i$  and  $a_j$ . Line 11 initializes  $R$ . Line 12-16 computes  $R$ , i.e., the  $\prec_{global}$  relation. Line 17 and Line 18 initializes  $S$  and  $LIST$ , respectively. Line 19-20 finds the total attribute order.

There are more sophisticated ways of inferring the total order. For example, the precedence relation is associated with probabilities in [4] and a brute-force search over all possible total orders is used to compute the most probable total order. A greedy algorithm for deciding the total order is described in [11]. However, as we will show in Section 4.3.2 on evaluation, our simple majority vote algorithm works well in practice. Note that cycles in the precedence relation are broken arbitrarily. Assuming there is an implicit global attribute order, accurate determination of local precedences will not produce cycles in attribute orders. We observed that the accuracy of precedence relationships (90%) in our experiments did not produce such cycles.

### 3.2.3 Putting it All Together

We use the the attribute order, determined in the previous step to generate labeled training examples from the reference table. Specifically, for each row in the reference table,

|    | BUSINESS | STREET | CITY | STATE | PHONE |
|----|----------|--------|------|-------|-------|
| #1 | 5        | 4      | 14   | 16    | 8     |
| #2 | 1        | 8      | 11   | 13    | 5     |
| #3 | 1        | 6      | 9    | 11    | 3     |
| #4 | 1        | 8      | 10   | 12    | 3     |

Table 5: Estimated Starting Positions

we concatenate attribute instances according to that order to form a token sequence. Each token is labeled with the attribute label associated with the column from where the token is drawn. These labeled sequences are used to train a global CRF for segmenting the test sequences.

### Unsupervised Text Segmentation

Once we get the labeled training examples, we use the standard training and inference techniques described in [10, 21, 1] to train a global CRF and use it to segment the test sequences in the batch (see Algorithm *UnsupervisedSeg* below).

**Algorithm *UnsupervisedSeg***

1. Infer total order by Algorithm *InferOrder* (Section 3.2.2)
2. Generate labeled examples
3. Train a global CRF
4. Segment test sequences in the batch

In the last two steps, we can just use any standard CRF training and inferencing techniques such as [10, 21, 16].

## 4 Evaluation

We implemented Algorithm *UnsupervisedSeg*, our structured-reference-table driven unsupervised text segmentation technique with CRFs. Our implementation extended the freely available CRF sourcecode [1]. In this section we report on its experimental performance. We begin by describing:

**4.1 The Experimental Setup** We discuss here the datasets used for the experiments including training and test data.

**Datasets:**

Table 6 lists the three different structured datasets that we used in our experiments, namely, address, product and bibliographic data. The address and bibliographic data were obtained from [2] and [3] respectively. The product data was extracted from the Web. The data schema corresponding to the three datasets is shown in the 2nd column of Table 6.

Each dataset represents a structured reference table where individual columns represent attributes. The entry in a particular column of the dataset represents an instance of the attribute corresponding to that column.

We use the attribute name to label its instances. Observe that these reference tables serve as labeled training data.

**Training and Test Datasets:** Each of the three datasets were divided into two mutually exclusive sets: a training set and a test set.

As noted above, the training data drawn from the structured reference table already comes in labeled form. We trained an attribute CRF model for each attribute in a dataset. Thus for the address dataset we build NAME CRF, STREET CRF, and so on. To construct an attribute CRF for say *Attr* we used two kinds of data - positive labeled data corresponding to instances of *Attr* and negatively labeled data generated as described in section 3.1.

We set aside a fraction of the reference table data not used for training, as the test dataset. We choose apriori a consistent attribute order for all the sequences in the test set and generate them from the reference table.

**4.2 Brute Force Segmentation** As is done in [4], Algorithm *UnsupervisedSeg* in section 3.1 also assumes that there is a consistent global total order among the attributes in the input (test) data sequences. To assess the impact of this assumption we compare *UnsupervisedSeg* with *BruteforceSeg*, a brute force text segmentation algorithm that makes no such assumptions.

Given a test sequence, *BruteforceSeg* tries all possible ways of labeling it in a brute-force way. Suppose we have  $n$  labels to label a test sequence of  $m$  tokens. First, *BruteforceSeg* divides the input sequence into  $i$  segments, with  $i$  from 1 to  $n$ . There are  $C_{m-1}^{i-1}$  ways of such segments. Then it chooses  $i$  labels from those  $n$  labels and permutes them. There are  $P_n^i$  permutations. Therefore in total there are  $\sum_{i=1}^n C_{i-1}^{m-1} P_i^n$  ways of labeling the input sequence. Each labeling corresponds to a particular segmentation of the test sequence. For each segmentation, the following computation is done: For each attribute CRF, *BruteforceSeg* computes the conditional probability of labeling the corresponding substring with only positive labels.

We illustrate this with an example. For the segmentation which labels “Stony Brook” as CITY and “NY” as STATE, *BruteforceSeg* computes the probability for “Stony Brook” with the CITY CRF and for “NY” with STATE CRF. After computation of individual conditional probabilities from each CRF, joint probability value is computed. Assuming independence among the attribute CRFs, an intuitive way of computing the joint probability of labeling “Stony Brook” with CITY and “NY” with STATE is to take the product. However, we have variable number of attributes and simply taking

| Dataset      | Schema  | Source                               | No. of Tuples |
|--------------|---|--------------------------------------|---------------|
| Address      | <i>Name, Street, City, State, Zip</i>                               | RISE repository[2]                   | 4300          |
| Product      | <i>Thickness, Manufacturer, Model, RingType, Category, Quantity</i> | Staples, OfficeMax, and Office Depot | 510           |
| Bibliography | <i>Author, Title, Publisher, Page, Year</i>                         | personal .bib file at [3]            | 80            |

Table 6: Datasets used for Experiments

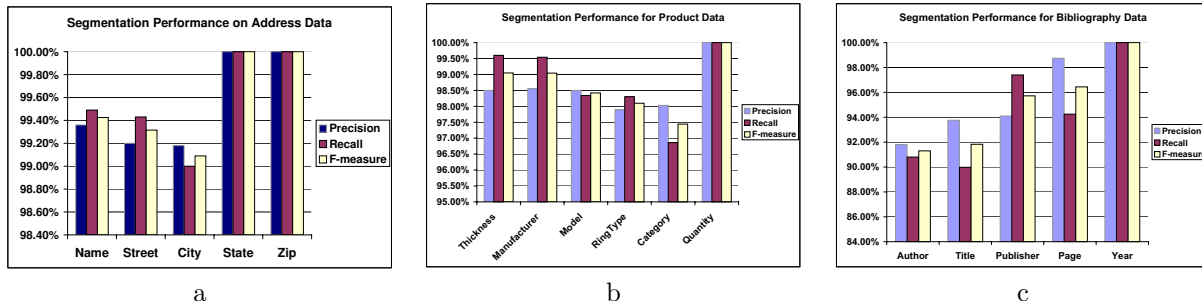


Figure 4: Performance for a) Address Dataset b) Product Dataset c) Bibliography Dataset

the product favors fewer attributes since each probability is between 0 and 1. So, *BruteforceSeg* computes the geometric mean of the probabilities returned by individual attribute CRFs as the joint probability. Once the joint probabilities of attribute CRFs of all possible segmentations are computed, the one with the highest joint probability is taken as the segmentation of the test sequence.

Note that, *BruteforceSeg* does not use the total order assumption. Therefore, it can be used as a baseline system to measure the goodness of *UnsupervisedSeg* as is shown in following subsection.

**4.3 Experimental Results** We evaluated the performance of our algorithm with respect to three performance metrics: recall, precision and f-measure<sup>1</sup>. We report on the performance of:

- i. *UnsupervisedSeg*
  - (a) with clean data
  - (b) with noisy data
  - (c) with varying training size
- ii. Algorithm *InferOrder*
- iii. Algorithm *BruteforceSeg* and compare it with *UnsupervisedSeg*.

<sup>1</sup>Recall value for an attribute *Attr* is defined as the ratio of tokens correctly assigned the label *Attr* over the total number of instances of *Attr* in the test set. For precision, the denominator is taken as the total number of tokens assigned the label *Attr* either correctly or incorrectly. The f-measure is calculated by taking the harmonic mean of recall and precision.

#### 4.3.1 Performance of UnsupervisedSeg: With Clean Data -

Figure 4a, 4b, and 4c illustrate precision, recall and f-measure performance for each of the attributes in address, product and bibliography dataset.

For address dataset, precision value ranges from 99.18% (for City) to 100% (for State and Zipcode). Recall performances range from 99% (for City) to 100% (for State and Zipcode). F-measure value ranges from 99.09% (for City) to 100% (for State and Zipcode).

For product dataset, precision value ranges from 94.50% (for Model) to 100% (Quantity). Recall performances range from 95% (for Model) to 100% (for Quantity) and f-measure performances range from 95.44% (for Category) to 100% (for Quantity).

For bibliography dataset, precision performances range from 91.80% (for Author) to 100% (for Year). Recall value ranges from 89.98% (for Title) to 100% (for Year). F-measure values range from 91.30% (for Author) to 100% (for Year).

These results suggest that unsupervised CRF-based text segmentation with reference tables works well in practice. Uniform performance is observed over all the three datasets. Our experimental results are comparable to the unsupervised HMM-based approach with reference tables in [4] and are considerably better than the work in [13] which exploits reference tables to reduce manually labeled data needed to train CRF models. We also note that attributes (such as Author in Figure 4c) with wide variability that is not captured in the training data are harder to find than others (such as Year in Figure 4c).

| Noise Type | Noise Description                           | Accuracy |
|------------|---|----------|
| Insertion  | Insert a random token                       | 80.81    |
| Deletion   | A randomly chosen token is deleted          | 92.63    |
| Spelling   | A randomly chosen token is corrupted        | 87.83    |
| Missing    | Replace randomly chosen attribute with null | 96.08    |

Table 7: Performance of UnsupervisedSeg with Noisy Data

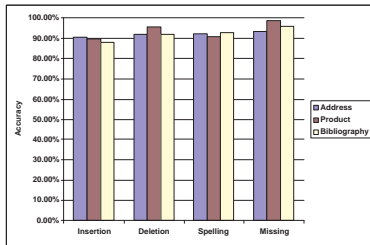


Figure 5: Accuracy of UnsupervisedSeg with Noisy Data

### With Noisy Data-

Noisy data refers to test sequences with missing attributes and errors due to insertion, deletion and misspellings. These errors were introduced randomly into the test sequences - one per sequence as was done in [4] - drawn from a uniform distribution.

Figure 5 shows accuracy of *UnsupervisedSeg* for address, product and bibliography dataset with noisy test data. The accuracy is defined as the fraction of correctly labeled tokens over all tokens in the test sequences.

Address and product datasets exhibit over 90% accuracy for any kind of noise present in the test data while for bibliography the accuracy is at least 88%.

Table 7 summarizes experimental results for each kind of noise.

These results suggest that *UnsupervisedSeg* works well even with noisy data and are comparable to those shown in [4].

### With Varying Training Size-

Figures 6, 7 and 8 show the relationship between training set size and performance on address, product and bibliography dataset respectively. 20% of the dataset was set aside for testing. From the remaining 80%, the size of the training data was increased from 25% to 100% in steps of 25%. Observe the longer training times and accuracy improvement with training size increase. The CRFs trained from 80% of data have an F-Measure of 99.1%, 99.3%, and 95.6% with training time of 25 minutes, 2 minutes, and 35 seconds for address, product, and bibliography respectively. Observe that, we can trade training time for high

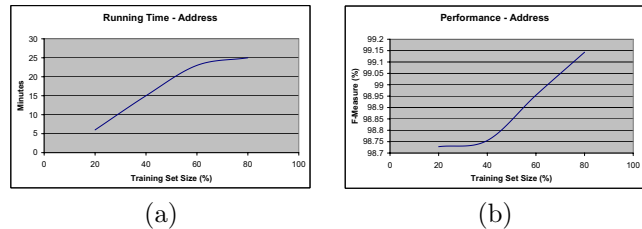


Figure 6: Varying Training Set Size of Address Dataset (a) Running Time (b) F-Measure

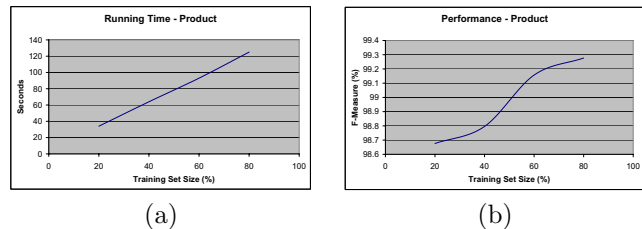


Figure 7: Varying Training Set Size of Product Dataset (a) Running Time (b) F-Measure

accuracy.

**4.3.2 Performance of InferOrder** To get a sense of how many training examples are needed to get a reasonable result, observe that 20% of training data amounts to 860, 102, and 16 examples for address, product and bibliography dataset respectively.

For the address dataset, the training time is 6 minutes and the accuracy of the resulting CRF in segmenting the test data is 98.7%. For the product dataset, the training time is 34 seconds and the accuracy

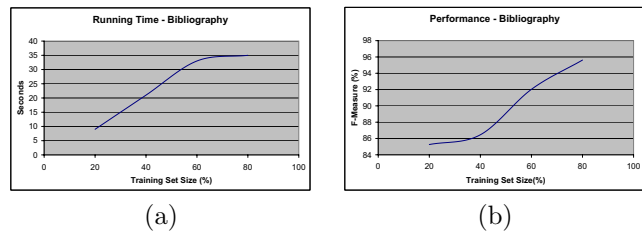


Figure 8: Varying Training Set Size of Bibliography Dataset (a) Running Time (b) F-Measure

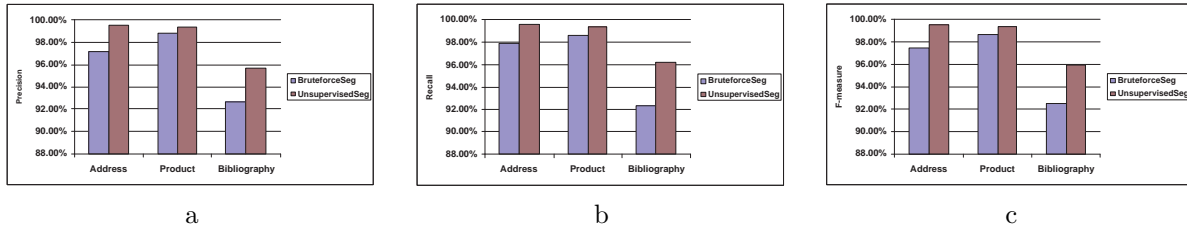


Figure 10: Comparative Performance a) Precision b) Recall c) F-Measure

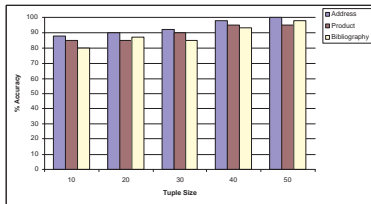


Figure 9: Accuracy of InferOrder

is also 98.7%. For the bibliography dataset, the numbers are 9 seconds and 94.6%. Therefore, we can achieve high accuracy even with a small number of training examples (which are not manually labeled). So these results seem to indicate that training a highly accurate CRF is not necessarily time consuming.

We evaluated the accuracy as defined in [4]<sup>2</sup> of InferOrder, for all three datasets. We took sets of sequences from each dataset. The size of each such set was varied from 10 to 50 tuples. Figure 9 shows the accuracy of our algorithm. Note that, our algorithm exhibits high accuracy even for small batch size.

### UnsupervisedSeg vs BruteforceSeg

Figure 10a, 10b, and 10c show their comparative performance. For all of the three datasets, we get higher precision, recall and f-measure values with the *UnsupervisedSeg*.

On the average, its precision, recall and f-measure values are 98.2%, 98.4% and 98.3% respectively. In contrast, *BruteforceSeg* achieves 96.20% precision, 96.3% recall and 96.2% in f-measure.

Observe that, *UnsupervisedSeg* performs better than *BruteforceSeg*. This is because the former can capture dependence between attributes actually present in the input data sequences assuming an attribute total order common to all of them.

<sup>2</sup>Given test sequences with a consistent total order, accuracy is defined as the fraction of attributes whose positions in the order were identified correctly

## 5 Related Work

The work described in this paper is broadly related to rule based, and supervised/unsupervised statistical text segmentation research.

### Rule-based Approaches:

Several rule based text segmentation works have been reported in the research literature (see [5, 22] for example). They are typically based on manually specified rules for identifying attribute instances in text. A limitation of such rule-based systems is that developers need to specify these rules and they may vary for different applications. With the use of reference tables our approach is fully automatic.

### Statistical Approaches:

Statistical based text segmentation approaches fall into two broad categories – supervised and unsupervised. The characteristic aspect of supervised methods, exemplified by a number of works [7, 14, 8, 6, 10], is that the segmentation models are trained with manually labeled training set. Among the dominant statistical models used for segmentation are HMMs (e.g. the DATAMOLD system [7]), Maximum Entropy Models (e.g. [14]) and Conditional Random Fields (e.g. [10]). A recent work on CRF-based text segmentation [13] focuses on reducing the training data using reference tables. There are two main differences of this work with our approach. First, CRFs with negative labels are used to define additional feature functions in [13]. Manually labeled examples, although fewer in number, are still required. In our case, we use such CRFs to infer a total order of attributes shared by a batch of test sequences, based on which we develop an unsupervised approach for training CRFs from reference tables. Second, the negative examples of our attribute CRFs are subtly constructed (as described in 3.1.2) to ensure that incorrect token sub-sequences are given a very low score.

Fully automatic unsupervised approaches to text segmentation is relatively less explored. Recently [4] describe CRAM, a fully automatic text segmentation system based on HMMs. The training data in this work is drawn from structured reference tables, obviating the need for manual labeling. Our unsupervised text

segmentation approach based on CRFs coupled with reference tables is inspired by this work.

In their work, reference tables are used to train HMM-based attribute recognition models (ARMs). The transition structures in ARMs require some knowledge of the application domain. So different application domains may require changes to this structure. In CRFs accommodating differences in application domain is accomplished by simply adding or removing features.

But the fundamental difference between CRAM and our approach is in the nature of the underlying statistical model. ARMs are based on HMMs in which probability distributions of tokens are stored in states. To accommodate tokens that were not seen in training CRAM uses a generalized dictionary, which is taxonomy of symbols such as words, numbers, delimiters, etc. Building a generalized dictionary is dependent on the application domain. In contrast CRFs are discriminative models, i.e. they do not capture the distribution of observations and hence dictionaries are not needed.

## 6 Conclusion

In this paper we described an unsupervised text segmentation algorithm using the discriminative CRF models in conjunction with reference tables. Our experimental results seem to suggest that the algorithm works well in practice. Furthermore they are comparable to the performance results reported in [4] and better than [13] which also uses a CRF-based approach in conjunction with reference tables and manually labeled examples. However we point out that [13] deals with multiple attribute occurrences using Semi-Markov CRFs [9, 19] that we have not addressed in this paper. Extending our approach to Semi-Markov CRFs remains to be done.

## References

- [1] <http://crf.sourceforge.net/>.
- [2] <http://www.isi.edu/info-agents/RISE/index.html>.
- [3] <http://www.it.iitb.ac.in/sunita/data/personalBib.tar.gz>.
- [4] E. Agichtein and V. Ganti. Mining reference tables for automatic text segmentation. In *KDD'04*, pages 20–29, August 2004.
- [5] B. Aldelberg. Nodose: A tool for semi-automatically extracting structured and semistructured data from text documents. In *SIGMOD*, 1998.
- [6] D. Beeferman, A. Berger, and J. Lafferty. Statistical models for text segmentation. *Machine Learning*, 34(1-3), 1999.
- [7] V. Borkar, K. Deshmukh, and S. Sarawagi. Automatic segmentation of text into structured records. In *ACM SIGMOD Conference*, pages 175–186, 2001.
- [8] M. E. Califf and R. J. Mooney. Relational learning of pattern-match rules for information extraction. In *Sixteenth National Conference on Artificial Intelligence*, 1999.
- [9] W. W. Cohen and S. Sarawagi. Exploiting dictionaries in named entity extraction: Combining semi-markov extraction processes and data integration methods. In *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Seattle, USA*, 2004.
- [10] J. Lafferty, A. McCallum, and F. Pereira. Conditional random fields: probabilistic models for segmenting and labeling sequence data. In *18th International Conference on Machine Learning*, pages 282–289, 2001.
- [11] M. Lapata. Probabilistic text structuring: Experiments with sentence ordering. In *Proceedings of annual meeting of ACL*, 2003.
- [12] W. Li and A. McCallum. Rapid development of hindi named entity recognition using conditional random fields and feature induction. 2:290–294, 2003.
- [13] I. R. Mansuri and S. Sarawagi. Integrating unstructured data into relational databases. In *ICDE*, page 29, 2006.
- [14] A. McCallum, D. Freitag, and F. C. N. Pereira. Maximum entropy markov models for information extraction and segmentation. In *ICML*, pages 591–598, 2000.
- [15] A. McCallum and W. Li. Early results for named entity recognition with conditional random fields. In *Proceedings of the seventh conference on Natural language learning (CoNLL-2003)*, 2003.
- [16] S. D. Pietra, V. D. Pietra, and J. Lafferty. Inducing features of random fields. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19:380–393.
- [17] D. Pinto, A. McCallum, X. Lee, and W. Croft. Table extraction using conditional random fields. In *Proceedings of the 26th ACM SIGIR*, 2003.
- [18] L. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of IEEE*, 77(2), 1989.
- [19] S. Sarawagi and W. W. Cohen. Semi-markov conditional random fields for information extraction. In *NIPS, 2004.*, 1989.
- [20] K. Seymore, A. McCallum, and R. Rosenfeld. Learning hidden markov model structure for information extraction. In *AAAI-99 Workshop on Machine Learning for Information Extraction*, 1999.
- [21] F. Sha and F. Pereira. Shallow parsing with conditional random fields. In *2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology*, pages 134–141, 2003.
- [22] S. Soderland. Learning information extraction rules for semi-structured and free text. *Machine Learning*, 34(1-3), 1999.
- [23] C. Sutton, K. Rohanimanesh, and A. McCallum. Dynamic conditional random fields: factorized probabilistic models for labeling and segmenting sequence data. In *ICML '04: Proceedings of the twenty-first international conference on Machine learning*, page 99, 2004.