Joint Unsupervised Structure Discovery and Information Extraction

Eli Cortez[1] Daniel Oliveira[1] Altigran S. da Silva[1]

Edleno S. de Moura[1] Alberto H. F. Laender[2]

[1]Departamento de Ciência da Computação
Universidade Federal do Amazonas
Manaus, AM, Brazil
{eccv,dpo,alti,edleno}@dcc.ufam.edu.br

[2]Departamento de Ciência da Computação
Universidade Federal de Minas Gerais
Belo Horizonte - MG - Brazil
laender@dcc.ufmg.br

ABSTRACT

In this paper we present JUDIE (Joint Unsupervised Structure Discovery and Information Extraction), a new method for automatically extracting semi-structured data records in the form of continuous text (e.g., bibliographic citations, postal addresses, classified ads, etc.) and having no explicit delimiters between them. While in state-of-the-art Information Extraction methods the structure of the data records is manually supplied the by user as a training step, JUDIE is capable of detecting the structure of each individual record being extracted without any user assistance. This is accomplished by a novel Structure Discovery algorithm that, given a sequence of labels representing attributes assigned to potential values, groups these labels into individual records by looking for frequent patterns of label repetitions among the given sequence. We also show how to integrate this algorithm in the information extraction process by means of successive refinement steps that alternate information extraction and structure discovery. Through an extensively experimental evaluation with different datasets in distinct domains, we compare JUDIE with state-of-the-art information extraction methods and conclude that, even without any user intervention, it is able to achieve high quality results on the tasks of discovering the structure of the records and extracting information from them.

Categories and Subject Descriptors

H.2 [**Database Management**]: Miscellaneous

General Terms

Algorithms, Performance, Experimentation

Keywords

Data Management, Information Extraction, Text Segmentation

SIGMOD'11, June 12–16, 2011, Athens, Greece.

1. INTRODUCTION

Information Extraction by Text Segmentation (IETS) is a widely applied technique to extract attribute values occurring in implicit semistructured data records in the form of continuous text, such as product descriptions, bibliographic citations, postal addresses, classified ads, etc. Current IETS methods rely on probabilistic graph-based models [10, 17] in which nodes (states) represent attributes and edges (transitions) represent the likely structures of the data records. When properly trained, such models are able to accurately predict a sequence of labels to be assigned to a sequence of text segments corresponding to attribute values.

The learning process thus consists in capturing content-related (or state) features, which characterize the domain of the attributes (e.g., typical values, terms composing them, their format, etc.), and structure-related (or transition) features (e.g., the positioning and sequencing of attribute values, etc.), which characterize the structure of the records within the source text. In recent years, the effort for training such models has been drastically reduced [1, 5, 14] or even eliminated [8]. This is mostly due to the use of pre-existing data sources such as reference tables, knowledge bases, etc., from which content-related features can be automatically learned. These features are, thus, source-independent. Structure-related features, however, are source-dependent. In most cases, they can be learned from user-provided training [1, 14] or can be automatically induced from the input texts in an on-demand way [8].

An important limitation in all previous IETS methods proposed in the literature is that they rely on the user to implicitly provide the likely structures of the records found on the source. This is true even for the most recent methods that apply some form of unsupervised learning [1, 5, 8, 14]. In most cases, the information on the likely structures is provided in the training phase, through sample records labeled by a user [1, 14]. The generated model is then able to extract information from one record at a time, what requires the user to separate each individual record prior to providing them as input for the extraction process. In other cases [5, 8], although the structure-related features can be automatically learned from the unlabeled input records, i.e., no explicit training is required, these records must still be provided one by one.

This requirement implies into several shortcomings for situations in which many implicit records are available in a single textual document (e.g., a list of references in a research article, or products in an inventory list) or a user is not available for separating the records (e.g., an extractor coupled with a crawler or when processing a stream of documents). Although straightforward methods could be applied to simple cases in which the set of attributes is fixed

for all records, dealing with semi-structured records such as heterogeneous bibliographic references, classified adds, etc., is much more complex. In the case of HTML pages, sometimes it is possible to automatically identify record boundaries and, thus, separate records by using heuristics based on the tags and paths inside the page [6, 7]. However, this is not the most common scenario on the Web and other on-line sources of textual documents, such as social networks or RSS messages.

1/2 cup butter 2 eggs 4 cups white sugar 1/2 cup milk 1 1/2 cups applesauce 2 tablespoons dark rum 2 cups all-purpose flour 1/4 cup cocoa powder 2 teaspoons baking soda ground cinnamon 1/8 teaspoon salt 1 cup raisins 6 chopped pecans 1/4 cup dark rum

Quantity	*Unit*	*Ingredient*
1/2	*cup*	*butter*
2		*eggs*
4	*cups*	*white sugar*
1/2	*cup*	*milk*
1 1/2	*cups*	*applesauce*
2	*tablespoons*	*dark rum*
2	*cups*	*all-purpose flour*
1/4	*cup*	*cocoa powder*
2	*teaspoons*	*baking soda*
		ground cinnamon
1/8	*teaspoon*	*salt*
1	*cup*	*raisins*
6		*chopped pecans*
1/4	*cup*	*dark rum*

Figure 1: Chocolate Cake recipe (top) and structured data extracted from it (bottom).

As an example, consider the Chocolate Cake recipe available in a pure text message illustrated in Figure 1. To provide a proper input to current IETS methods, a user would have to scan the message and manually separate each record containing the specification of an ingredient in the recipe. Notice the cases in which attributes "Quantity" and "Unit" are missing in the input message. Automatically processing several of such messages with current IETS methods is unfeasible, even if they come from the same source.

In this paper we present JUDIE (Joint Unsupervised structure Discovery and Information Extraction), a new method for IETS that addresses the problem of automatically extracting several implicit records and their attribute values from a single text input. Unlike previous methods in the literature, ours is capable of detecting the structure of each individual record being extracted without any user intervention. The table in Figure 1 illustrates the output of our method when the text on the top is given as input.

To uncover the structure of the input records, we use a novel algorithm that, given a sequence of labels representing attribute values, groups these labels into individual records by looking for frequent patterns of label repetitions, or *cycles*, among the given sequence. The *Structure Discover (SD)* algorithm is, thus, our first contribution in this paper.

Our second contribution is the way we integrate this algorithm in the information extraction process. This is accomplished by successive refinement steps that alternate information extraction and structure discovery. Following, we briefly describe how our method executes this process.

Given an input text with a set of implicit data records in textual format, such as the one illustrated in Figure 1, the first step of our method performs an initial labeling of the candidate values identified in this input with attribute names. As at this point there is no information on the structure of the data records, we resort only to content-related features for this labeling. Thus, this step, called *Structure-free Labeling*, generates a sequence of labels in which some candidate values may be missing or have received a wrong label. Despite being imprecise, this sequence of labels is accurate enough to allow the generation of an approximate description of the structure of the records in the input text (as demonstrated in our experiments). This is accomplished in the second step of our method, called *Structure Sketching*, by using the SD algorithm. The output of this step is a set of labeled values grouped into records that already bear a structure close to the correct one. Thus, from these records it is possible to learn structure-related source-dependent features. These features can now be used to revise the *Structure-free Labeling* from the first step. This *Structure-aware Labeling* is the third step of our method. As demonstrated by our experiments, the results produced by this step are more precise than those obtained by the *Structure-free Labeling*, since now content-related and structure-related features are taken into consideration. Our method then takes advantage of this more precise sequence of labels to revise the structure of the records. This new sequence is given as input to the SD algorithm. This is the fourth and final step of our method. It is called *Structure Refinement*. We notice that all of these steps are completely unsupervised.

The remainder of this paper is organized as follows. Section 2 discusses related work. Section 3 describes our method, while Section 4 focuses on the features used by our method. Section 5 presents a detailed description of the Structure Discovery algorithm. Section 6 reports the results of an empirical evaluation we have performed with several sources and different domains. Finally, Section 7 presents conclusions and discusses future work.

2. RELATED WORK

Since the pioneering work of Soderland [19], the problem of extracting information from textual inputs containing implicit data records has received considerable attention in the literature. This problem is also known as *Information Extraction by Text Segmentation (IETS)* and the most successful methods that deal with it rely on machine learning techniques. In general, these methods are supervised [10, 12, 13, 16], in the sense they require hand-labeled examples (training data) that are provided by an expert user. In order to alleviate this need, unsupervised methods have been proposed, for which training data is created from pre-existing data sources [1, 5, 8, 14].

However, existing IETS methods do not address the problem of automatically discovering the structure of implicit records. Usually, they consider that such structure is either inferred from given examples or provided in advance by the user. In this sense, JUDIE is a novel approach to automatically extracting several implicit records and their attribute values from a single text input, since it performs the extraction process while dealing with text inputs containing data records with different structures. This task is not accomplished by any previously mentioned method. For instance, ONDUX [8] and U-CRF [5] require direct user intervention to delimit each record within the text input as well as to define the structure of the records to be extracted. It is important to stress that the record structure plays a very important role in this context, since both U-CRF and ONDUX rely on structural features (positioning and sequencing) to perform the extraction task.

The task of structure discovery was studied earlier in the literature. For example, in [6] and [7], the authors propose methods to automatically find the structure of data records available in HTML pages. Notice that our context and the input texts we deal with are different from those treated by such methods. In our scenario, we do not rely on the existence of HTML tags and record delimiters, but only on the actual text that contains attribute values that might

compose a record. This provides more flexibility and robustness to our method, since HTML markup is not always available and, even when it is available, there is no guarantee that it is consistently used, which is an important requirement for those methods. This is particularly true on today's Web in which the extensive use of automatic layout generators and style-sheets may result in HTML structures that are hard to handle automatically.

Indeed, the development of unsupervised methods for information extraction is motivated by the increasing number of textual documents that are made available on the Web, not only in HTML format, but also as plain text. Thus, new information extraction methods should not only work at Web scale, but also be free, or at least highly independent, from any user assistance [4]. We believe that our method, JUDIE, is a step towards fulfilling these requirements, since it does not require any specific information from the user, neither to help discovering the structure of the target records, nor to extract any piece of information included in the input text.

3. METHOD DESCRIPTION

In this section, we present our method by describing the main four steps that comprise it. For that, we use a running example illustrated in Figures 2(a) to (f). We consider that the unstructured sequence of tokens corresponding to a list of items of a chocolate cake recipe, shown in Figure 2(a), is given as input. Our method then carries out the task of simultaneously extracting the components of each item, i.e., Quantity (**Q**), Unity (**U**) and Ingredient (**I**), and structuring them into records. The final output is illustrated in Figure 2(f). The four steps that comprise our method are described next.

3.1 Structure-free Labeling

Given an unstructured input text containing a set of implicit data records in textual format, such as the one illustrated in Figure 2(a), the first step of our method consists of initially labeling potential values identified in this input with attribute names. As at this point there is no information on the structure of the data records, we resort only to content-related features for this labeling. Thus, we call this step *Structure-free Labeling*.

All content-related features we use can be computed from a pre-existing dataset. Consider an attribute A and let v_A be a set of typical values for this attribute. Then, for any segment of tokens $x_i, \ldots, x_j$ from the input text, we can compute the value of a feature function $g^k(x_i, \ldots, x_j, A)$. Intuitively, g^k returns a real number that measures how well a hypothetical value formed by tokens in the text segment $x_i, \ldots, x_j$ follows some property of the values in the domain of A [17]. Obviously, the accuracy of such functions often depends on how representative v_A is with respect to the values in the domain of A.

A very important aspect we exploit in our method is that it is possible to compute g^k for a segment of tokens $x_i, \ldots, x_j$ independently of the input text in which it occurs. Thus, we say that g^k is *domain-dependent*, meaning that it is *source-independent*. Using domain-dependent features is a common trend in recent work on information extraction. While in most cases they are used only with textual attributes for computing string similarity [1,3,5,14] or vocabulary affinity [8], here we investigate other possibilities.

In Section 4.1 we present the details of the content-related features we use in our method. In the following, we describe how these features are used to process the structure-free labeling.

From now on, we assume that all datasets are representative of their attributes for the purpose of computing the features we use. Also, for a given extraction task that involves attributes $A_1, \ldots, A_n$, we say that datasets $v_{A_1}, \ldots, v_{A_n}$ form a knowledge base KB. Given a data source on a certain domain that includes values associated with attributes, building such a knowledge base is a simple process that consists in creating attribute-value pairs. Examples of such data sources are relations, reference tables, ontologies, etc.

Processing the Structure-free Labeling

The targets of the structure-free labeling are sequences of tokens in the input text that are likely to represent attribute values. We call them *candidate values* and they are defined as follows.

Let $I = t_1, t_2, \ldots, t_n$ be the set of tokens occurring in an input text, such that no token contains white space. Consider a knowledge base KB representing attributes $A_1, \ldots, A_m$. A *likely value* in I is the largest sequence of tokens $s = t_i, t_{i+1}, \ldots, t_{i+k}$ ($1 \leq i \leq n, k \geq 0$) from I that occurs as a value, or part of a value, of some attribute A_j. In the input text I, all likely values and all individual tokens that do not belong to any likely values are called *candidate values*.

Figure 2(b) illustrates the candidate values found in the input text of Figure 2(a). Notice that candidate values such as "raising flour" and "Melted butter" can only be likely values. In the knowledge base we use in our experiments for the Cooking Recipes domain, values such as "Milk" and "Salt" are represented. Thus, the corresponding candidate values, in spite of being formed by a single token, are also likely values. On the other hand, "Tbsp" is not present in that knowledge base. Thus, it is an isolated token taken as a candidate value.

Given a candidate value s, the decision on what label must be assigned to it takes into account different domain-dependent features g^k evaluated by feature functions of the form $g^k(s, A)$. To combine these features, we assume that they represent the probability of the candidate value s to occur as a value of the attribute A domain, according to KB. If we assume that these features exploit different properties of the attribute A domain, we can say they are independent, what allows us to combine them by means of the Bayesian disjunctive operator *or*, also known as *Noisy-OR-Gate* [15], which is defined as:

$$or(p_1, \ldots, p_n) = 1 - ((1 - p_1) \times \ldots \times (1 - p_n))$$

where each p_i is a probability.

Thus, our final equation is:

$$\ell(s, A) = 1 - \big(\big(1 - g^1(s, A)\big) \times \ldots \times (1 - g^n(s, A))\big) \quad (1)$$

Informally, by using the disjunctive operator we assume that any of the features is likely to determine the labeling (i.e., significantly increase its final probability), regardless of other factors [15]. By doing so, we avoid having to fine-tune relative weights for individual factors. As we shall see, this hypothesis is confirmed in our experiments.

Function $\ell(s, A)$ is computed for each candidate value s in the input text for all attributes A of the same data type (i.e., text or numeric). Thus, s is labeled with a label representing the attribute that yielded the highest score according to this function.

The results of applying the structure-free labeling over the input sequence of Figure 2(a) is illustrated in Figure 2(c), in which capital letters represent labels assigned to candidate values, each label representing an attribute as follows: **Q** for Quantity, **U** for Unity and **I** for Ingredient. Notice that one of the candidate values is marked with a "?", meaning that no label could be assigned to it. This occurs when no feature used in Equation 1 gives a value greater than zero. This exemplifies one of the anticipated limitations of the structure-free labeling, which we discuss below.

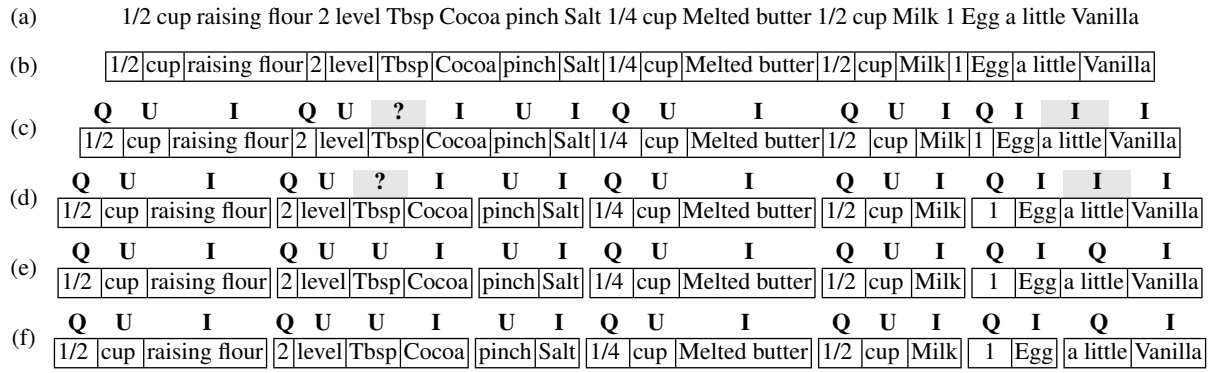


Figure 2: Running example with illustrations of the main steps that comprise our proposed method.

Limitations of the Structure-free Labeling

The use of very effective domain-dependent features yields a highly precise label assignment in the structure-free labeling step. This claim is supported by the results of extensive experiments we report in this paper, involving more than 30 distinct attributes on five distinct datasets.

In spite of that, using such features may represent a problem in two important cases: (1) two (or more) attributes in the same knowledge base are similar with respect to the property being evaluated by the feature function; (2) the property being evaluated is under-represented within the known values of some attribute in the knowledge base. In the first case, wrong labels can be assigned to some segment, i.e., a *label misassignment* occurs. In the second case, there is no support for "safely" assigning a label to that segment, i.e., a *label fault* occurs.

In Figure 2(a) we exemplify these two cases by shadowing the labels assigned to two of the candidate values. For the candidate value "Tbsp", the "?" indicates a label fault, while for the candidate value "a little" the shadowed "I" indicates a label misassignment. In this second case, the correct label would be "Q".

For dealing with such cases, state-of-the-art information extraction methods rely on features that also consider the context in which the segment being evaluated occurs within the input text. These features are derived from the structure of the record used as training data [1, 3, 5, 8, 10, 12, 14].

In our case, it is not possible to use these structure-related features simply because our input text bears no structure. However, imprecise as is, this sequence of labels generated by the structure-free labeling is accurate enough to allow the generation of an approximate description of the structure of the records in the input text. This is accomplished by the second step of our method, called *Structure Sketching*, which we describe next.

3.2 Structure Sketching

The goal of the structure sketching step is to organize the labeled candidate values into records, effectively inducing a structure on the unstructured text input. As this step takes as input the labels generated in the structure-free labeling step, in which imprecisions are expected, we consider this structure as a first approximation. The output of this step is a set of labeled values grouped into records that already bear a structure close to the correct one. In our method, this step plays an important role: with the structure of the input text uncovered, we can evaluate structural features and improve the initial labeling from the first step.

The structure sketching step uses a novel algorithm called *Structure Discover (SD)*, which is one of the main contributions of this paper. Let $\ell_1, \ell_2, \ldots, \ell_n$ be a sequence of labels generated by the structure-free labeling step, in which each label was assigned to a candidate value. The SD algorithm is used to identify in this sequence common subsequences of labels that are frequently repeated in the input text, which we call *cycles*. When a cycle that covers all the input text is found, it can be used to group labels in subsequences according to it. Each of these subsequences corresponds to a record grouping values from distinct attributes. These subsequences are called *candidate records*. We postpone a detailed discussion of the SD algorithm to Section 5.

The result of applying the SD algorithm on the labeled sequence of Figure 2(c) is shown in Figure 2(d). Notice that now candidate values are grouped into distinct subsequences, that is, into candidate records. In this example, the cycle found is a simple sequence of the attributes Quantity, Unit and Ingredient.

As this example illustrates, the algorithm is able to deal with irregularities in the candidate records, such as missing or repeated attribute values. Dealing with irregularities is important not only to address natural irregularities often found in real cases, but also to make the process robust to errors caused by the labeling process. In this particular example, while a candidate value of attribute Quantity is indeed missing in the third candidate record, the sequence of three candidate values for attribute Ingredient in the last candidate record is caused by an error in the structure-free labeling step.

As our experimental results indicate, in spite of these and others irregularities (e.g., candidate records with distinct orderings of attribute occurrence), the SD algorithm is able to discover a plausible structure for the input sequence of labels. Again, we refer the reader to Section 5 for details on the SD algorithm.

With a plausible structure already uncovered by the SD algorithm, it is now possible to compute structure-related features in conjunction with content-related features to improve the initial labeling of the candidate values. This procedure is explained next.

3.3 Structure-aware Labeling

Consider a candidate record $R = s_1, \ldots, s_r$, where each $s_i (1 \leq i \leq r)$ is a candidate value. Also, consider an attribute A and let ℓ_A be a label used for this attribute. Then, for any candidate value s_i, we can compute the value of a feature function $f^k(s_i, A, R)$. Function f^k returns a real number that measures the likelihood of a segment labeled ℓ_A to occur in the same place as s_i in R. Thus, the value of f^k is related to the structure of R.

Differently from the content-based features used so far, which are only domain-dependent, structure-based features such as f^k depend on the particular organization of the candidate values within the input text. This means that these features are *source-dependent*.

Like other information extraction methods (e.g., [5, 8, 14]), our method uses two structure-related features. The first considers the absolute position of the segment and the second considers its relative position, i.e., its occurrence between segment s_{i-1} (if any, i.e., when $i > 0$) and segment s_{i+1} (if any, i.e., when $i < r$).

For computing such features, it is common to build a graph model that represents the likelihood of attribute transitions within the input text (or any other input text from the same source). In this paper, we adopt the same approach as in [8]. More specifically, we built a probabilistic HMM-like graph model that we call PSM (Positioning and Sequencing Model). This model and the source-dependent features computed based on it are described in Section 4.2.

With the structure-related features in hand, we can use them to improve the initial structure-free labeling, as we describe next.

Processing the Structure-aware Labeling

Given a candidate value s, the decision on which label to assign to it can now consider the structure-related features f^j in addition to the content-related features g^k. As these features are also independent from the content-related ones, since they depend on the source, we again resort to the Bayesian Noisy-OR-Gate [15] to combine all features as follows:

$$\ell(s,R,A) = 1 - \big(\big(1 - g^1(s,A)\big) \times \ldots \times (1 - g^n(s,A)) \times \big(1 - f^1(s,A,R)\big) \times \ldots \times (1 - f^m(s,A,R))\big) \quad (2)$$

Function $\ell(s, R, A)$ is computed for each candidate segment s of all candidate records R in the input text for all attributes A of the same data type (i.e., text or numeric). Thus, s is labeled with a label representing the attribute that yielded the highest score according to ℓ.

The result of applying the structure-free labeling over the candidate records of Figure 2(d) is illustrated in Figure 2(e). Notice that with the addition of the structure-related features, the candidate value "Tbsp" is now correctly labeled as **U** for Unity (this term is indeed used in place of "tablespoon"). For the same reason, candidate value "a little" is now correctly labeled as **Q** for Quantity.

As this example suggests, in general, combining structure-related and content-related features produce more precise results than the initial structure-free labeling. This trend is clearly indicated by our experiments.

Our method then takes advantage of this more precise sequence of labels to also revise the structure of the records. This new sequence is given as input to the SD algorithm. This is the fourth and final step of our method.

3.4 Structure Refinement

This last step of our method simply consists in applying again the SD algorithm. This time, however, it takes as input the labels generated by the structure-aware labeling. As the labeling produced by this step is more precise, the result is a more accurate structure. This is also indicated by our experimental results.

To illustrate it, notice that in Figure 2(f) the last candidate record from Figure 2(g) has now been split in two different records by the SD algorithm. Again, we refer the reader to Section 5 for details on the SD algorithm.

4. FEATURES USED

In this section, we described the features we use in our method. We begin by presenting the content-related, domain-dependent features and then we present the structure-related, source-dependent features.

4.1 Content-related Features

Attribute Vocabulary. These features exploit the common vocabulary often shared by values of textual attributes (e.g., neighborhood and street names, author names, recipe ingredients, etc.). To capture this property, we resort to a function called *AF* (Attribute Frequency) [9], which estimates the similarity between a given value and the set of values of an attribute. In our case, the function AF is used to estimate the similarity between the content of a candidate value s and the values of an attribute A represented in the knowledge base KB. Function AF is defined as follows:

$$AF(s,A) = \frac{\sum_{t \in T(A) \cap T(s)} \mathit{fitness}(t,A)}{|T(s)|} \quad (3)$$

In Equation 3, $T(A)$ is the set of all terms found in the values of attribute A represented in KB and $T(s)$ is the set of terms found in the candidate value s. The function $fitness(t, A)$ evaluates how typical a term t is among the values of attribute A. It is computed as follows:

$$fitness(t,A) = \frac{f(t,A)}{N(t)} \times \frac{f(t,A)}{f_{max}(A)} \quad (4)$$

where $f(t, A)$ is the number of distinct values of A that contain the term t, $f_{max}(A)$ is the highest frequency of any term among the values of A, and $N(t)$ is the total number of occurrences of the term t in all attributes represented in KB.

The first fraction in Equation 4 expresses the likelihood of term t to be part of a value of A according to KB. This fraction is multiplied by a normalization factor in the second fraction. This prevents attributes with many values in KB from dominating and is also useful for making the term frequency comparable among all attributes.

We notice that although we could have used any other similarity function, for instance, based on the vector space model, experiments reported in the literature [9] have shown that AF is very effective for dealing with small portions of texts such as the ones typically found in candidate values.

It is also worth mentioning that we use inverted indexes over the knowledge base to speed up the computation of this feature.

Attribute Value Range. For the case of numeric candidate values (e.g., page numbers, year, phone number, price, quantity, etc.) textual similarity functions such as AF do not work properly. Thus, for dealing with these candidate values a proper feature function is needed. We assume, as proposed in [11], that the values of numeric attributes follow a Gaussian distribution. Based on this assumption, we measure the similarity between a numeric value v_s present in a candidate value s and the set of values v_A of an attribute A in KB, by evaluating how close v_s is from the mean value of v_A according to its probability density function. For that, we use the function *NM* (Numeric Matching), defined in Equation 5, normalized by the maximum probability density of v_A, which is reached when a given value is equal to the average[1].

$$NM(s,A) = e^{-\frac{v_s - \mu}{2\sigma^2}} \quad (5)$$

where σ and μ are, respectively, the standard deviation and the average of values in v_A, and v_s is the numeric value of s.

Notice that when v_s is close to the average of values in v_A, $NM(s, A)$ is close to 1. As v_s assumes values far from the average, the similarity tends to zero.

[1]The maximum probability density of v_A is $1/\sqrt{2\pi\sigma^2}$.

In many cases, numeric values in the input texts may include special characters (e.g., prices and phone numbers). Thus, prior to the application of the NM function, these characters are removed and the remaining numbers are concatenated. We call this process *Normalization*. For instance, the string "412-638-7273" is normalized to form a numeric value 4126387273 that can be applied to the function NM. Normalization is also performed over numeric values that occur in KB.

Attribute Value Format. Another property we exploit as a content-related feature is the common format often used to represent values of some attributes. Let v_A be the set of values available for an attribute A in KB. We automatically learn a sequence Markov model m_A that captures the format style of the values in v_A. This model is similar to the inner HMM used in [12] and is also applied to capture the format of values as a state feature.

For that, we first tokenize each value of v_A on white-spaces. Using a taxonomy proposed in [12], we encode this value as a sequence of *symbol masks* or simply *masks*. A mask is a character class identifier, possibly followed by a quantifier. For example, the value "Rivers inc." of the Company attribute is encoded as "[A-Z][a-z]+ [a-z]+.", where mask [A-Z] represents a string that starts with an uppercase letter, mask [a-z]+ represents a sequence of one or more lowercase letters, etc. This process is repeated for all known values of a given attribute.

Then, the model m_A is generated based on these masks, so that each node n corresponds to a mask that represents the values of v_A. An edge e between nodes n_i and n_j is built if n_i is followed by n_j in the masks. Thus, each value in v_A can be described by a path in m_A.

To express the likelihood of sequences of masks in the model, we define the *weight* of an edge $\langle n_x, n_y \rangle$ as:

$$w(n_x, n_y) = \frac{\text{\# of pairs } \langle n_x, n_y \rangle \text{ in } m_A}{\text{\# of pairs } \langle n_x, n_z \rangle, \forall n_z \in m_A}$$

Now, let s be a candidate value. We can encode s using the symbol taxonomy as above. This results in a sequence of masks. We evaluate how similar a candidate value s is to the values in v_A with respect to their formats by computing

$$format(s, A) = \frac{\sum_{\langle n_x, n_y \rangle \in path(s)} w(n_x, n_y)}{|path(s)|} \quad (6)$$

where $path(s)$ represents a path formed by the sequence of masks generated for s in m_A. Notice that, if no path matching for this sequence is found in m_a, $format(s, A) = 0$.

Intuitively, $format(s, A)$ evaluates how likely are the sequences of symbols forming a given candidate value s with respect to the sequences of symbols typically occurring as values of some attribute A. By using such feature, we capture specific formatting properties of URLs, e-mails, telephone numbers, etc.

Notice that the model m_A is learned from the set of values v_A only. Thus, differently from [12], no manual training is needed.

4.2 Structure-related Features

State-of-the-art information extraction methods (e.g., [5, 8, 14]) usually use two types of structure-related feature. The first type considers the absolute position of the text segment or token to be evaluated and the second one considers its relative position, i.e., its occurrence between other segments or tokens in the input text. For computing such features, it is common to build a graph model that represents the likelihood of transitions within the input text (or other input texts from the same source).

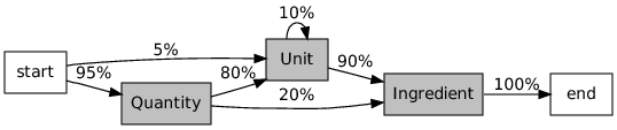


Figure 3: Example of a PSM

In most CRF-based methods, this model is built from training data, which consists of a set of delimited records manually labeled taken from the same input text [14]. In [5] and [8], the model is built in an unsupervised way during the extraction process itself. While in [5] a fixed order, learned from a sample, is assumed for the attributes in the input text, in [8] the model is built using all records available in the input text and no fixed order is assumed. Thus, in our work we adopt the same approach as in [8]. More specifically, we built a probabilistic HMM-like graph model called PSM (*Positioning and Sequencing Model*).

In our case, a PSM consists of: (1) a set of states $L = \{begin, \ell_1, \ell_2, \ldots, \ell_n, end\}$ where each state ℓ_i corresponds to a label assigned to a candidate value in the structure-free labeling step; (2) a matrix T that stores the probability of observing a transition from state ℓ_i to state ℓ_j; and (3) a matrix P that stores the probability of observing a label ℓ_i in the set of candidate labels that occupies the k-th position in a candidate record.

Matrix T, which stores the transition probabilities, is built using the ratio of the number of transitions made from state ℓ_i to state ℓ_j in a candidate record to the total number of transitions made from state ℓ_i in all known candidate records. Thus, each element $t_{i,j}$ in T is defined as:

$$t_{i,j} = \frac{\text{\# of transitions from } \ell_i \text{ to } \ell_j}{\text{Total \# of transitions out of } \ell_i} \quad (7)$$

Matrix P, which stores the position probabilities, is built using the ratio of the number of times a label ℓ_i is observed in position k in a candidate record to the total number of labels observed in candidate values that occupy position k in all known candidate records. Thus, each element $p_{i,k}$ in P is defined as:

$$p_{i,k} = \frac{\text{\# of observations of } \ell_i \text{ in } k}{\text{Total \# of candidate values in } k} \quad (8)$$

By using Equations 7 and 8, matrices T and P are built to maximize the probabilities of the sequencing and the positioning observed for the attribute values, according to the labeled blocks in the output of the matching step. This follows the Maximum Likelihood approach, commonly used for training graphical models [12, 17].

In practice, building matrices T and P involve performing a single pass over the input text which, at this point, has already been processed in the Structure Sketching step. Notice that candidate values left unmatched are discarded when building these matrices. Obviously, possible mismatched candidate values will be used to build the PSM, generating spurious transitions. However, as the number of mismatches resulting from the Structure-free Labeling step is rather small, as demonstrated in our experiments, they do not compromise the overall correctness of the model.

Figure 3 shows an example of the PSM built for an input text with cooking recipes. As we can see, the graph represents not only information on the sequencing of labels assigned to candidate values, but also on the positioning of candidate values in the input text. For instance, in this example, input texts are more likely to begin with blocks labeled Quantity than with blocks labeled Unit. Also, there is a high probability that blocks labeled Ingredient occur after blocks labeled Unit.

Let s_k be a candidate value in a candidate record $R = \ldots, s_k, \ldots$ for which a label ℓ_i corresponding to an attribute A_i is to be assigned. Also, suppose that in R the candidate value next to s_k is labeled with ℓ_j corresponding to an attribute A_j. Then, using Equations 7 an 8, we can compute the two structure-related features we consider, i.e, the *sequencing feature* and the *positioning feature*, respectively as:

$$seq(s_k, A_i, R) = t_{i,j} \text{ and } pos(s_k, A_i, R) = p_{i,k} \qquad (9)$$

5. THE SD ALGORITHM

In this section, we present a detailed description of the Structure Discovery (SD) algorithm. As discussed in Section 3, this algorithm plays an important role in our method: to uncover the structure of the implicit records from the input text. Besides the ultimate goal of structuring the data extracted in the final step (Structure Refinement), this algorithm is also used to induce structure-related features that improve the quality of the extraction in an intermediate step (Structure Sketching). In both steps, the SD algorithm takes as input a sequence of labels assigned to candidate values generated in the preceding steps and generates the structure of each record available in the input text. Following, we present preliminary concepts related to the algorithm and describe its main steps.

The main intuition behind the algorithm is that it is possible to identify patterns of sequences by looking for cycles into a graph that models the ordering of labels in the labeled input text. This graph, called *Adjacency Graph*, is defined below.

Adjacency Graph. Consider the sequence $s_1, s_2, \ldots, s_n$ of candidate values in the input text, such that s_i is labeled with ℓ_i. The ordered list $L = \langle \ell_1, \ell_2, \ldots, \ell_n \rangle$ is called an *Adjacency List*. An *Adjacency Graph* is a digraph $G = \langle V, E \rangle$ in which V is the set of all distinct labels in L, plus two special labels *begin* and *end*, and E is the set of all pairs $\langle \ell_i, \ell_j \rangle$ in E for all i, j such that $j = i + 1$ $(1 \leq i \leq n - 1)$, plus two special edges $\langle begin, \ell_1 \rangle$ and $\langle \ell_n, end \rangle$.

Title Conference Year Author Author Title Conference Year Author Title Conference Year ... Author Title Journal Issue Year Author Title Journal Issue Year Author Author Journal Issue Year Title Year ... Author Title Conference Year Author Author Author Title Journal Issue Year

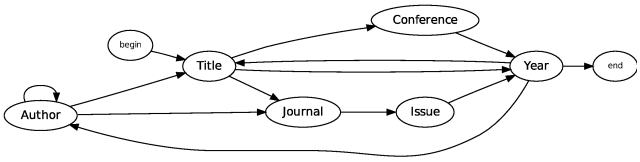


Figure 4: An Adjacency List and an Adjacency Graph for an input text with bibliographic data.

Figure 4 illustrates portions of an Adjacency List built from a sample unstructured text containing a number of implicit bibliographic data records. This sample is a simplified version of a real bibliographic data source such as *CORA* (a dataset we have used in our experiments) represented by some of the attributes involved (e.g., no volume or page information is represented). This sample, however, exemplifies some of the problems faced when processing real textual inputs.

Figure 4 also illustrates an Adjacency Graph built from this Adjacency List. In this graph, nodes corresponding to attributes are represented by ellipses identified by their respective labels in the adjacency list. Nodes *begin* and *end* are considered as if they occurred only once in this list, respectively, before and after the sequence of candidate values $s_1, \ldots, s_n$. Their role is simply to serve as references for the graph processing algorithms used by our method.

The two long paths ⟨Author,Title,Conference,Year⟩ and ⟨Author, Title,Journal,Issue,Year⟩ correspond, respectively, to publications in conferences and journals. Notice, however, that some edges indicate the occurrence of implicit records with missing attributes. This is the case of the edge ⟨Author,Journal⟩ that indicates a missing value for Title. Also notice that ⟨Year,Author⟩ and ⟨Year,Title⟩ intuitively indicate records ending with an Year candidate value leading to another record that may begin with either Author or Title. Indeed, the first implicit record in the input text begins with a Title candidate value.

The occurrence of implicit records with missing attributes is a very common issue in most real cases. This situation occurs either due to errors in the labeling process, specially in the case of the structure-free labeling, or because the implicit record indeed has no value for some attributes.

An important aspect the SD algorithm exploits in the adjacency graph is the occurrence of cycles. A cycle is a sequence of adjacent nodes $\langle \ell_i, \ldots, \ell_{i+k}, \ell_i \rangle$. For convenience, we use the notation $[\ell_i, \ldots, \ell_{i+k}]$, omitting the last node, which is always equal to the first one.

The different configurations of implicit records, i.e., the set of attributes composing them and the order in which their candidate values appear, can be detected by looking for cycles in the adjacency graph. This is the case of cycles [Author,Title,Conference,Year], [Author,Title,Journal,Issue,Year] and [Title,Conference,Year] in Figure 4.

Two important issues arise when using the adjacency graph to analyze the possible record structures in the input text: (1) in which order the labels in the cycle occur in the input text and (2) which cycles correspond to actual implicit records in the input text. To deal with both issues, we verify the correspondence between cycles and the sequence of labels in the adjacency list. For the definitions below, let G be an adjacency graph generated from an adjacency list L.

Coincident Cycles. Two cycles c_a and c_b are said to be *coincident*, meaning that they represent the same cycle in G, if they include the same edges in the same order, but beginning and ending at a different node in the cycle.

Cycle Instances and Viable Cycles. Let $c = [\ell_i, \ldots, \ell_{i+k}]$ be a cycle in G. Any sequence $\ell_i, \ldots, \ell_{i+k}$ in L is said to be an *instance* of c. The cycle c is said to be *viable* if there is at least one instance of c in L.

Dominant Cycles. Let $\{c_1, \ldots, c_n\}$ be a set of coincident cycles. The viable cycle c_i for which the order of labels is the most frequent in L is called the *dominant cycle*.

To exemplify these concepts, cycles c_a = [Author,Title,Conference,Year] and c_b = [Title,Conference,Year,Author] are coincident in the adjacency graph of Figure 4. By looking into the adjacency list, we find that c_b is the dominant cycle.

These concepts are used by the SD algorithm (Algorithm 1). This algorithm works by first identifying all dominant cycles in the adjacency graph and then processing each of these cycles in the order of their sizes, the largest cycles being processed first. Notice that nodes *begin* and *end* never participate in any cycle, since they are both connected to the graph by a single edge.

In Lines 2 and 3, the Adjacency List and the Adjacency Graph are created. Next, in Lines 4 and 5, the algorithm detects all single cycles in the graph in order to remove all sequences of a same label

Algorithm 1: Structure Discovery Algorithm

input : I: input text with labeled candidate values
output: L: sequence of records that structure I

```
1  begin
2      L ← adjlist(I);
3      G ← adjgraph(L);
4      foreach single cycle [ℓ, ℓ] in G do
5          Replace all sequences ℓ, ..., ℓ by one single element ℓ+ in L
6      G ← adjgraph(L);
7      C ← dominant_cycles(G);
8      i ← 0;
9      while C ≠ ∅ do
10         dc_i ← next(C);
11         for each instance ℓ_1, ..., ℓ_k of dc_i in L do
12             Replace ℓ_1, ..., ℓ_k by r_i in L;
13         i++;
14 end
```

from the adjacency list. Such sequences usually represent multivalued attributes (e.g., lists) that must be considered as a single component in the records being identified. Thus, these sequences are replaced by a single label $\ell+$ in the adjacency list.

In Line 6, a new Adjacency Graph is generated for reflecting the removal of theses sequences. If we consider the graph in Figure 4, the only effect will be the removal of the cycle involving Author.

In Line 7, the algorithm extracts all dominant cycles from G. Next, these dominant cycles are used to structure their instances in the input text. This is carried out by the loop in Lines 9 to 13. In Line 10, the function $next$ selects and removes the largest dominant cycles from C and, in Lines 11 and 12, the instances of the cycles in the adjacency list L are replaced by an indication that a record has been formed with each of these instances. Thus, in our algorithm, records are taken as cycle instances whose boundaries are determined by matching cycles derived from the graph to the adjacency list (Line 11).

We notice the importance of processing larger cycles first. Considering the graph in Figure 4, if the cycle c_b = [Title,Conference, Year] was processed before c_a = [Author,Title,Conference,Year], part of each instance of c_b would be taken as a instance of c_a. This process continues while there are cycles unprocessed in C.

For the Adjacency List and the Adjacency Graph of Figure 4, the sequence of dominant cycles that would be processed is the following: [Author,Title,Journal,Issue,Year], [Author,Title,Conference, Year], [Author,Journal,Issue,Year], [Title,Conference,Year] and [Title,Year].

6. EXPERIMENTAL EVALUATION

In this section, we describe the experiments we have performed to evaluate JUDIE using five distinct datasets. First, we describe the experimental setup and the metrics used to assess JUDIE's performance. Then, we report on the quality of the extraction results for each dataset.

6.1 Setup

The datasets employed in our experiments and the data sources used to generate the knowledge bases for JUDIE are summarized in Table 1. We notice that some of these datasets are the same employed in the evaluation of other information extraction methods. We also recall that our method takes as input sets of records without any explicit delimiters between them, as illustrated in Figure 1.

The dataset of the *Cooking Recipes* domain was previously used in [2]. In order to build the knowledge base for this domain, we have collected structured recipes from FreeBase[2]. For the *Product Offers* domain, the dataset is formed by unstructured strings containing lists of product offers from 25 Brazilian e-commerce stores. Data for building the respective knowledge base has been taken from Nhemu[3], a Brazilian price comparison website. For the *Postal Adresses* domain, both the dataset and the data source used to build the knowledge base have been obtained from *Bigbook*, a dataset available in the RISE repository[4] and that has been previously used in [5] and [8].

For the *Bibliography* domain, the dataset is part of the Cora Collection[5] and is composed of a large diversity of bibliographic citations in distinct styles and formats. It includes citations to journal articles, conference papers, books, technical reports, etc. The data source for building the knowledge base, PersonalBib, is also a dataset of bibliographic citations that has been used in [14]. Finally, for the *Classified Ads* domain we have taken the dataset previously used in [8]. This dataset is composed of unstructured strings containing ads from Brazilian newspaper websites. For building the knowledge base, we have collected data from a database available on the website of a major Brazilian newspaper.

For all performed experiments, we evaluated the extraction results for each individual attribute (attribute-level) and for each record type as whole (record-level). As evaluation metrics, we have used the well known precision, recall and F-measure as defined next.

Let B_i be a reference set and S_i be a test set to be compared with B_i. We define precision (P_i), recall (R_i) and F-measure (F_i) respectively as:

$$P_i = \frac{|B_i \cap S_i|}{|S_i|},\ R_i = \frac{|B_i \cap S_i|}{|B_i|} \text{ and } F_i = \frac{2(R_i.P_i)}{(R_i + P_i)}$$

In order to present attribute-level results, we calculate precision, recall and F-measure according to the above equations by considering B_i as the set of terms that compose the values of a given attribute a_i and S_i the set of terms assigned to a_i by our method. Likewise, for record-level results, we calculate precision, recall and F-measure by considering each record set B_i as the set of field values in a given structured record C_i and S_i the set of field values extracted for C_i by our method.

6.2 General Quality Results

In this section, we analyze the general quality of the extraction task performed by JUDIE on the datasets described in Table 1. For each domain, we have run the extraction task five times, each time selecting different data samples for the data extraction task and for building the respective knowledge bases. For all performed extractions, we report the average F-measure obtained for all runs. We also notice that there is no intersection between the knowledge bases and the corresponding datasets we use in our experiments.

Tables 2(a)–(e) present attribute-level F-measure values that assess the extraction quality in each dataset. Column "C1" refers to results obtained after the Structure-free Labeling and Structure Sketching steps, which correspond to what we call Phase 1, and Column "C2" refers to results obtained after the Structure-aware Labeling and Structure Discovery steps, which correspond to what we call Phase 2. Column "G" presents the gain in quality achieved from Phase 1 to Phase 2.

Each of these columns assesses a distinct aspect of our method. Results in column "C1" assess how well the content-related source-independent features alone have been able to assign correct labels

[2] http://www.freebase.com
[3] http://www.nhemu.com
[4] http://www.isi.edu/info-agents/RISE
[5] http://www.cs.umass.edu/~mccallum/data

Domain	Dataset	Text Inputs	Attributes	Source	Attributes	Records
Cooking Recipes	*Recipes*	500	3	*FreeBase.com*	3	100
Product Offers	*Products*	10000	3	*Nhemu.com*	3	5000
Postal Addresses	*BigBook*	2000	5	*BigBook*	5	2000
Bibliography	*CORA*	500	3 to 7	*PersonalBib*	7	395
Classified Ads	*WebAds*	500	5 to 18	*Folha On-line*	18	125

Table 1: Domains, datasets and KB data sources used in the experiments.

	Phase 1			Phase 2		
Attribute	FI/NM	FO	C1	S+P	C2	G %
Quantity	0.81	0.69	0.89	0.78	0.96	7.1
Unit	0.86	0.46	0.91	0.82	0.94	3.9
Ingredient	0.84	0.74	0.91	0.76	0.96	4.9
Average	0.84	0.63	0.90	0.79	0.95	5.3

(a) Recipes

	Phase 1			Phase 2		
Attribute	FI/NM	FO	C1	S+P	C2	G %
Name	0.77	0.37	0.85	0.69	0.90	5.3
Brand	0.74	0.52	0.83	0.71	0.92	10.5
Price	0.89	0.92	0.93	0.88	0.95	1.9
Average	0.80	0.60	0.87	0.76	0.92	5.8

(b) Products

	Phase 1			Phase 2		
Attribute	FI/NM	FO	C1	S+P	C2	G %
Name	0.79	0.48	0.94	0.63	0.97	2.6
Street	0.82	0.40	0.95	0.75	0.97	2.6
City	0.92	0.39	0.94	0.84	0.97	2.8
State	0.89	0.63	0.96	0.88	0.97	1.3
Phone	0.94	0.93	0.95	0.89	0.97	2.3
Average	0.87	0.57	0.95	0.80	0.97	2.3

(c) BigBook

	Phase 1			Phase 2		
Attribute	FI/NM	FO	C1	S+P	C2	G %
Author	0.79	0.60	0.83	0.65	0.88	5.9
Title	0.60	0.52	0.70	0.48	0.79	13.8
Booktitle	0.82	0.46	0.81	0.67	0.86	6.2
Journal	0.69	0.53	0.72	0.62	0.84	16.9
Volume	0.84	0.88	0.88	0.72	0.90	2.9
Pages	0.79	0.80	0.83	0.73	0.86	3.9
Date	0.72	0.76	0.79	0.69	0.87	9.5
Average	0.75	0.65	0.79	0.65	0.86	8.1

(d) CORA

	Phase 1			Phase 2		
Attribute	FI/NM	FO	C1	S+P	C2	G %
Bedroom	0.75	0.36	0.79	0.48	0.82	3.8
Living	0.81	0.46	0.85	0.69	0.89	5.6
Phone	0.79	0.84	0.80	0.62	0.87	8.8
Price	0.85	0.85	0.86	0.66	0.92	7.2
Kitchen	0.80	0.29	0.79	0.73	0.83	4.9
Bathroom	0.73	0.59	0.75	0.69	0.77	2.9
Suite	0.85	0.45	0.87	0.60	0.89	2.4
Pantry	0.79	0.50	0.77	0.66	0.80	3.7
Garage	0.78	0.52	0.79	0.73	0.84	6.6
Pool	0.77	0.63	0.78	0.78	0.82	5.2
Others	0.70	0.44	0.72	0.68	0.73	1.6
Average	0.78	0.54	0.80	0.67	0.84	4.8

(e) WebAds

Table 2: Attribute-level results for each dataset.

to the input text, while results in column "C2" also account for the use of structure-related source-dependent features learned from the input text itself.

To provide a perspective on the contribution of each feature to the overall extraction quality, we also present F-measure values obtained when each type of feature is individually used. The cases considered are: (1) either the *fitness* function for textual attributes (Eq. 4) or the *NM* function for numeric attributes (Eq. 5) is used (Column "FI/NM"); (2) only the *format* function (Eq. 6) is used (Column "FO"); and (3) only the *pos* and *seq* (Eq. 9) functions are used (Column "S+P'). We recall that "C1" results are obtained by combining in Phase 1 functions *fitness* (or *NM*) and *format* by using Eq. 1, and that "C2" results are obtained by combining in Phase 2 functions *fitness* (or *NM*), *format*, *pos* and *seq* by using Eq. 2.

As anticipated, we observe that the attribute-level results obtained in Phase 1 by combining features are already acceptable and, more importantly, are sufficient to yield a reasonable approximation of the records' structure. Furthermore, the *fitness* and *MN* functions are, in general, more accurate than the *format* function. However, their combination, as proposed in our method, leads to better results in all cases.

Phase 2 results are higher in all cases. While in most cases the gain is under 6%, there are interesting cases in which this gain is above 10%. For example, Title and Journal are attributes that present a large content overlap in the Bibliography dataset. Due to this problem, the percentage of labels incorrectly assigned to values of Title and Journal in Phase 1 was 25% and 16% respectively. In Phase 2, the majority of these misassignments were corrected. A large gain was also observed in the case of attribute Brand from the Products dataset. Because brand names are formed by terms usually not available in the knowledge base, more than 12% of values of this attribute were left unmatched in Phase 1. The structure-related features used in Phase 2 helped to recover these errors.

The behavior of our method when dealing with numeric attributes deserves specific comments. Considering the eight numeric attributes from the five distinct datasets used in our experiments, the *NM* function yielded an average attribute-level F-measure of 0.83 when used alone. This result is close to that obtained for the non-numeric attributes using the *fitness* function. For instance, with phone numbers, using only the *NM* function we obtained F-measure values of 0.94 and 0.79 for the BigBook and WebAds datasets, respectively. Moreover, the *format* function is also used with these attributes, but, in this case, unlikely to what happens with the *NM* function, values are not normalized (see Section 4.1). When used alone, this function also yielded an average F-measure of 0.83. Finally, the structure-based features helped to improve these results. When combined with the other two features, an average F-measure of 0.91 was obtained. As we can see, our method achieves equally good results with both numeric and textual attributes.

Dataset	Phase 1	Phase 2	Gain %
Recipes	0.79	0.90	13.2
Products	0.82	0.88	7.2
BigBook	0.86	0.93	8.8
CORA	0.69	0.83	19.3
WebAds	0.70	0.77	9.7

Table 3: General record-level results for each dataset.

Table 3 presents, for each dataset, record-level F-measure results obtained in Phase 1 and Phase 2. While results in Phase 1 are also acceptable (most of them above 0.7), improvements in labeling achieved in Phase 2 had a very positive effect. Indeed, in Phase 2 record-level F-measure has achieved results above 0.8 for four out of five datasets and, in all cases, gains have been above 7%. Notice, for instance, the case of the CORA dataset, in which the gain is higher than 19%, reflecting the improvements obtained by the structure-aware labeling step. As we can notice, adding the structure-related features (only possible in Phase 2) also leads to significant improvements regarding record-level results.

6.3 Impact of the Knowledge Base

In [8] the authors present an experiment to evaluate how dependent on the composition of the knowledge base is the quality of the extraction results.

In the case of JUDIE such study is even more important for the following reasons: (1) the extraction process entirely relies on the initial Structure-free Labeling step, which is solely based on content-related features learned from the knowledge base; (2) while in our closest competitor, ONDUX [8], the knowledge base is used only for matching, JUDIE also deploys a format feature based on its values. Thus, in JUDIE the knowledge base plays a crucial role, as we show in this experimental evaluation.

Here we compare JUDIE with ONDUX and U-CRF. These two methods are the current state-of-the-art unsupervised IETS methods. U-CRF was developed by adapting the publicly available implementation of CRF by Sunita Sarawagi[6] according to [5] and using additional features described in [10] (e.g., dictionary features, word score functions, transition features, etc.). As required by U-CRF, a batch of input strings is used to infer the order of the attribute values. Based on the information provided in [5], this batch is built using a sample of 10% of these strings.

As in [8], this experiment was performed using the *BigBook* dataset from the RISE repository. The knowledge base for ONDUX and JUDIE and the reference table for U-CRF were built by using sets of records already extracted. Once again, we notice that there is no intersection between these records and the corresponding datasets used in this experiment. Recall that while ONDUX and U-CRF received the input in a record-by-record basis, JUDIE received a single input text containing all 2000 records with no explicit delimiters between them.

The experiment consisted of varying the number of known terms common to the knowledge base (or reference table in the case of U-CRF) and the input test records from 50 to 1000 terms and evaluating the extraction quality in terms of average attribute-level F-measure. The results are presented in Figure 5(a).

The first important observation regarding this graph is that JUDIE is, as expected, more dependent on the knowledge base than ONDUX and U-CRF. Indeed, only when the number of shared terms approaches 1000, it reaches the same quality level as the baselines. This occurs because in both ONDUX and U-CRF the structure-related and content-related features are independent, while in JUDIE, as previously explained, content-related features are used to induce structured-based features through successive refinement steps.

Indeed, if content-based features are not enough, the induction of structure-based features fails. This can be observed in Figure 5(a), where the attribute-level F-measure values obtained with less than 250 common terms are very low. For this level of term intersection, the results of JUDIE's Phase 1, i.e., before any refinement, are better than the results of its Phase 2, in which structure-based features are also considered.

In spite of this limitation, JUDIE achieves results comparable to the state-of-the-art baselines for a task considerably harder, that is, extracting information while simultaneously uncovering its underlying structure. As already explained, this underlying structure is assumed as provided in the baseline methods. In Section 6.5, we present a detailed comparison between JUDIE and these baselines using other datasets.

6.4 Impact of Structure Diversity

In this section we study how our method deals with different types of structure observed in the implicit records found in the input text. For this we consider two different scenarios, structure diversity in different sources and within a single source. These two scenarios are discussed in the following.

6.4.1 Structure Diversity in Different Sources

To discuss the first scenario we use the Classified Ads domain, for which the knowledge base was build using data from one source and the input texts came from other five distinct sources. In the experiments reported below, each source corresponds to a different input text. Here, our goal is to demonstrate that the content-related features learned from data taken from one source can be used to induce the structure-related features for several related input texts from other distinct sources in the same domain.

In Figure 5(b) we show the attribute-level and record-level F-measure values obtained for each different source given as input to JUDIE. In all cases, the values are above 0.7 and for two cases they are above 0.8. This indicates that our method is source-independent, since it was able to correctly uncover the structure of implicit records in each source while also achieving good extraction level quality. This occurs despite of the differences in structure of the implicit records in each source.

6.4.2 Structure Diversity within a Single Source

For discussing the second scenario we use the Bibliography domain in which the knowledge base was built from the PersonalBib dataset [14] and single input texts came from the CORA collection. In this case we aim at showing how our method deals with a heterogeneous dataset in terms of structure.

By examining the distribution of citation styles among the 500 implicit records available in the CORA dataset, a total of 33 distinct styles were identified, but only six of them account for more than 90% of the citations[7].

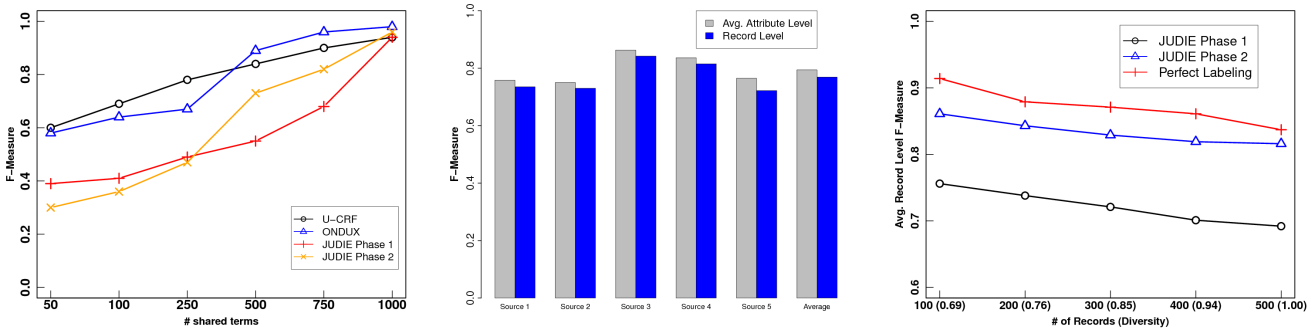
For these experiments, we generated different input texts containing 100 to 500 implicit records randomly selected from the CORA dataset. We then process each of these input texts separately with JUDIE using the knowledge base described above. The process was repeated 10 times for each input text size.

To characterize the diversity of each input text we have used the Shannon Index [18], which is frequently used to measure diversity in categorical datasets. This index is defined as: $H = -\sum_{i=1}^{S} p_i ln(p_i)$, where S is the total number of styles (33 in this case) and p_i is the relative frequency of each style i found in the input text. As the H index does not return values between 0 and 1, we normalize H values obtained for each input text by the maximum possible value for H. This value occurs when the input contains all citations available in the CORA dataset, that is, when all 33 different citation styles are present in the input text[8]. Thus, the closer the

[6] http://crf.sourceforge.net/

[7] A citation style characterized by the set of attributes composing the record and their ordering.

[8] $H = 2.23671$ for the input containing 500 records.



(a) Varying the number of shared terms. (b) Results for distinct Ads sources. (c) Results varying diversity.

Figure 5: Results obtained by our method varying distinct aspects in the input texts.

normalized H value is to 1, the greater is the diversity of the input text.

The results obtained are presented in Figure 5(c) in terms of the average record-level F-measure, considering Phases 1 and 2 of JUDIE. As a baseline, we use the record-level F-measure we would obtain if the labeling of attribute values was perfect in the input texts. In the X-axis, we present the number of records and the diversity of each input text in terms of the normalized H index.

This figure shows that our SD algorithm deals very well with structure diversity if the labels are correctly assigned, as it can be seen by comparing the curves representing JUDIE Phase 1 and the perfect labeling. As we can also observe, the improvements on the quality of the labeling provided by adding the structure-related features in Phase 2 impacts positively on the quality of the structure discovery. Indeed, record-level F-measure values obtained in Phase 2 are close to those obtained with the perfect labeling.

6.5 Comparison with Previous Work

In this section we present a comparison between the results obtained by JUDIE with those obtained by two state-of-the-art IETS methods, namely ONDUX [8] and U-CRF [5].

This comparison is made by reproducing in Tables 4(a) to (c) the attribute-level results obtained for three datasets, which were reported in [8] for the two methods, along with the results we obtain by running JUDIE over the same datasets.

While ONDUX was first presented and fully described in that paper, U-CRF was used there as a baseline. The details on its implementation are summarized in Section 6.3. In all cases, we have used the same sources for generating the knowledge bases and the input texts. We recall again that among the three methods, JUDIE is the only one that is able to both discover the structure and extract information automatically.

As a general observation, in spite of the fact the JUDIE faces a harder task, its performance was very close to that of ONDUX. In most cases, ONDUX outperformed JUDIE, but there are a few cases in which JUDIE performed better than ONDUX. These cases are explained mainly by the use of the format feature in JUDIE. Such feature is not considered in ONDUX.

In comparison with U-CRF, JUDIE performed worse on the BigBook dataset, but better on the CORA and WebAds datasets. This was expected, since these datasets are much more irregular in terms of structure than the first one.

6.6 Performance Issues

In Table 5 we present the running times of the experiments executed with JUDIE. For the datasets we used to run comparative ex-

Attribute	JUDIE	ONDUX	U-CRF
Name	0.967	0.996 (2.97%)	0.995 (2.86%)
Street	0.970	0.995 (2.58%)	0.993 (2.37%)
City	0.971	0.995 (2.43%)	0.990 (1.92%)
State	0.971	1.000 (2.95%)	0.999 (2.84%)
Phone	0.975	1.000 (2.57%)	0.988 (1.34%)
Average	0.971	0.997 (2.70%)	0.993 (2.27%)

(a) BigBook

Attribute	JUDIE	ONDUX	U-CRF
Author	0.881	0.922 (4.65%)	0.876 (-0.57%)
Title	0.794	0.792 (-0.25%)	0.694 (-12.59%)
Booktitle	0.855	0.892 (4.33%)	0.560 (-34.50%)
Journal	0.843	0.908 (7.71%)	0.553 (-34.40%)
Volume	0.901	0.958 (6.33%)	0.430 (-52.28%)
Pages	0.861	0.849 (-1.39%)	0.503 (-41.58%)
Date	0.865	0.895 (3.47%)	0.488 (-43.58%)
Average	0.857	0.888 (3.60%)	0.586 (-31.60%)

(b) CORA

Attribute	JUDIE	ONDUX	U-CRF
Bedroom	0.818	0.861 (5.25%)	0.791 (-3.30%)
Living	0.893	0.905 (1.34%)	0.724 (-18.93%)
Phone	0.873	0.926 (6.12%)	0.754 (-13.59%)
Price	0.923	0.936 (1.41%)	0.786 (-14.84%)
Kitchen	0.830	0.849 (2.29%)	0.788 (-5.06%)
Bathroom	0.773	0.792 (2.51%)	0.810 (4.84%)
Suite	0.894	0.881 (-1.50%)	0.900 (0.62%)
Pantry	0.800	0.796 (-0.55%)	0.687 (-14.17%)
Garage	0.844	0.816 (-3.28%)	0.714 (-15.37%)
Pool	0.818	0.780 (-4.66%)	0.683 (-16.52%)
Other	0.732	0.796 (8.68%)	0.719 (-1.84%)
Average	0.836	0.849 (1.52%)	0.760 (-9.16%)

(c) WebAds

Table 4: Comparison of results.

periments with our baselines ONDUX and U-CRF, we also include the running times of these systems. As the number of implicit input records is different for each dataset, we present both the total running time and the average running time by record.

Before discussing the results, we notice that JUDIE running times depend on two main factors: the number of implicit input records and the diversity in the structure of these records. Regarding the first factor, in all steps the input is scanned once. Thus, there is a linear influence of this factor. As for the second factor, having more

Datasets	Total (secs.)			Avg. per record (msecs.)		
	JUDIE	ONDUX	U-CRF	JUDIE	ONDUX	U-CRF
Recipes	37.5	-	-	75.1	-	-
Products	69.2	-	-	6.9	-	-
BigBook	50.2	14.1	297.1	25.1	7.1	148.5
CORA	74.4	10.7	185.9	148.8	21.4	371.8
WebAds	59.2	8.0	2701.9	118.5	16.0	5403.7

Table 5: JUDIE running times in comparison with baselines.

diverse records in terms of structure implies that a larger number of edges will occur in the Adjacency Graph and in the PSM. Thus, processing these graphs has higher costs for more heterogeneous structures. This explain why the average running times per record are higher for CORA and WebAds, which, as discussed in Section 6.4, are the more diverse datasets in our experiments.

Nevertheless, these running times are in the same order of magnitude as those of ONDUX and are, in general, smaller than those of U-CRF. ONDUX is faster since it executes fewer steps and does not include a structure discovery step. U-CRF has a worst performance due to costly inference steps, particularly when dealing with diverse structures, and due to the use of a larger number of features than ONDUX and JUDIE.

7. CONCLUSIONS AND FUTURE WORK

In this paper, we have presented JUDIE a novel method for extracting semi-structured data records in the form of continuous text (e.g., bibliographic citations, postal addresses, classified ads, etc.) with no explicit delimiters between them. Differently from state-of-the-art IETS methods, in which the structure of the data records is manually supplied by an expert user in a training step, JUDIE is capable of detecting the structure of each individual record being extracted without any user assistance. For this, we propose a novel Structure Discovery algorithm that, given a sequence of labels representing attributes assigned to potential values, groups these labels into individual records by looking for frequent patterns of label repetitions among the given sequence. We have also shown how to integrate this algorithm in the information extraction process by means of successive refinement steps that alternate information extraction and structure discovery.

By means of a thoroughly experimental evaluation, we have studied different aspects regarding our method and compared it with state-of-the-art IETS methods. Results indicate that our method performs quite well when compared with such methods, even without any user intervention.

As future work , we intend to investigate techniques for automating the generation of knowledge bases usually required by unsupervised IETS methods. In addition, since JUDIE currently does not handle nested structures, we also plan to address this issue by generalizing the SD algorithm to deal with nested cycles.

Acknowledgements

This work is partially supported by INWeb (MCT/CNPq grant 57.3871/2008-6), by project MinGroup (CNPq grant 575553/2008-1), by UOL Bolsa Pesquisa program (grant 20090213165000), and by the authors' individual grants and scholarships from CNPq, CAPES, FAPEMIG and FAPEAM.

8. REFERENCES

[1] E. Agichtein and V. Ganti. Mining Reference Tables for Automatic Text Segmentation. In *Proc. 10th ACM SIGKDD Intl. Conf. on Knowl. Discov. and Data Mining*, pages 20–29, 2004.

[2] L. Barbosa and J. Freire. Using Latent-structure to Detect Objects on the Web. In *Proc. of the 13th Intl. Workshop on the Web and Databases*, pages 1–6, 2010.

[3] W. Cohen and S. Sarawagi. Exploiting Dictionaries in Named Entity Extraction: Combining Semi-Markov Extraction Processes and Data Integration Methods. In *Proc. 10th ACM SIGKDD Intl. Conf. on Knowl. Discov. and Data Mining*, pages 89–98, 2004.

[4] A. Doan et. al. Information Extraction Challenges in Managing Unstructured Data. *SIGMOD Record*, 37(4):14–20, 2008.

[5] C. Zhao et. al. Exploiting Structured Reference Data for Unsupervised Text Segmentation with Conditional Random Fields. In *Proc. SIAM Intl. Conf. on Data Mining*, pages 420–431, 2008.

[6] D. Buttler et. al. A Fully Automated Object Extraction System for the World Wide Web. In *Proc. of the 21st Intl. Conf. on Dist. Comp. Syst.*, pages 361–370, 2001.

[7] D. W. Embley et. al. Conceptual-model-based data extraction from multiple-record web pages. *Data Knowl. Eng.*, 31(3):227–251, 1999.

[8] E. Cortez et. al. ONDUX: On-Demand Unsupervised Learning for Information Extraction. In *Proc. ACM SIGMOD Intl. Conf. on Management of Data*, pages 807–818, 2010.

[9] F. Mesquita et. al. LABRADOR: Efficiently publishing relational databases on the web by using keyword-based query interfaces. *Inform. Proc. and Management*, 43(4):983–1004, 2007.

[10] J. D. Lafferty et. al. Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data. In *Proc. 8th Intl. Conf. on Mach. Learning*, pages 282–289, 2001.

[11] S. Agrawal et. al. Automated Ranking of Database Query Results. In *Proc. of the First Biennial Conf. on Innov. Data Syst. Research*, 2003.

[12] V. Borkar et. al. Automatic Segmentation of Text into Structured Records. In *Proc. ACM SIGMOD Intl. Conf. on Management of Data*, pages 175–186, 2001.

[13] D. Freitag and A. McCallum. Information Extraction with HMM Structures Learned by Stochastic Optimization. In *Proc. of the 17th Nat. Conf. on Art. Intell. and 12th Conf. on Innov. Appl. of Art. Intell.*, pages 584–589, 2000.

[14] I. R. Mansuri and S. Sarawagi. Integrating Unstructured Data into Relational Databases. In *Proc. 22nd Intl. Conf. on Data Engineering*, page 29, 2006.

[15] J. Pearl and G. Shafer. *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Morgan Kaufmann, 1988.

[16] F. Peng and A. McCallum. Information extraction from research papers using conditional random fields. *Inform. Proc. and Management*, 42(4):963–979, 2006.

[17] S. Sarawagi. Information extraction. *Foundations and Trends in Databases*, 1(3):261–377, 2008.

[18] C. E. Shannon. A mathematical theory of communication. *ACM SIGMOBILE Mobile Comp. and Comm. Rev.*, 5(1):3–55, 2001.

[19] S. Soderland. Learning information extraction rules for semi-structured and free text. *Mach. Learn.*, 34:233–272, 1999.