Exploiting Structure within Data for Accurate Labeling using Conditional Random Fields

Aman Goel[1], **Craig A. Knoblock**[2], **and Kristina Lerman**[3]
[1]amangoel@usc.edu, [2]knoblock@isi.edu, [3]lerman@isi.edu
Information Sciences Institute and Department of Computer Science
University of Southern California
Marina del Rey, CA 90292

Abstract—*Automatically assigning semantic class labels such as WindSpeed, Flight Number and Address to data obtained from structured sources including databases or web pages is an important problem in data integration since it enables the researchers to identify the contents of these sources. Automatic semantic annotation is difficult because of the variety of formats used for each semantic type (e.g., Date) as well as the similarity between different semantic types (e.g., Humidity and Chance of Precipitation). In this paper, we show that by exploiting different kinds of latent structure within data we can perform this task accurately. We show that this improvement happens in spite of higher complexity in terms of both the inference procedure and the increased number of labels. We study how increasing the amount of structure taken into account by the model improves accuracy of semantic labeling. Finally, we show that when exploiting all the relationships, we obtain a significant improvement in field labeling accuracy over the regular-expression-based approach, while still keeping the complexity low.*

Keywords: Supervised learning, Graphical models, Semantic web

1. Introduction

Automatic semantic annotation of structured data elements is an important problem in information integration. It helps in identifying the contents of various sources so that further operations can be performed on them. For instance, once the semantic type of the data in different sources is known, these sources can be joined together. Each domain has some common semantic types defined in them. For example, the weather domain presents information about *Temperature*, *Humidity*, and *Visibility*, the flights domain presents information about *Flight Number*, *Airport*, and *Time of Arrival* and the spatial domain presents information about *Address*, *Latitude*, and *Longitude*. The first column of Table 1 shows a sequence of fields extracted from a weather forecast website and its second column shows their semantic types or labels. Formally defined, the task of semantic annotation is to assign these semantic labels to the sequence of fields when their identity is unknown.

Automatic semantic annotation is difficult to perform accurately. There are two main reasons for this. First, the semantic types have many different formats in which they can be written. Table 2 shows some of the various ways of writing the value of 9 $miles$ for *Visibility*. There are variations in the choice of the unit, its abbreviation, the precision of the numeric value, and even in the representation of a particular edge case value as a non-numeric term (for example, *Clear*). Therefore, even if the model has seen examples of a few formats, a new format may not be recognized by the model. Second, several semantic types can look similar to each other. *Humidity* and *ChanceOfPrecipitation* are two such semantic classes. Both of them are percentage values, written as a numeric value followed by a percent symbol (e.g., *40%*). This makes it hard for the model to discriminate between these semantic types.

Researchers have used CRFs to perform semantic annotation in the past. Zhu et al. [17] used CRFs to labels objects on a webpage by exploiting the relationship between adjacent items. Tang et al. [15] used tree-structured CRFs to identify different information elements such as *Telephone Number* and *Secretary Name* in official reports. Both of these approaches exploit the relationships among the field labels. The problem with this approach is that in the presence of many semantic types and weak dependencies between adjacent fields, these approaches do not perform as well. Instead, we propose to exploit the structure within the fields by modeling the relationship between the labels of the field and the labels of their tokens as well as the relationships among the adjacent tokens to achieve high field labeling accuracy. In this paper, we show that as we exploit more and more of the latent structure within the data, we can achieve higher field labeling accuracy. We show that this holds true even when the complexity of the inference increases and we add many more labels that the model has to choose from. We also show that our graph structure keeps a low limit on the complexity of the inference algorithm which is proportional to $L_f \times L_t^2$, where L_f is the number of field labels and L_t

Table 1: A sample data tuple extracted from a weather forecast website. The table shows the fields, their corresponding semantic classes, their tokens (generate by our lexer) and their semantic classes.

Fields	**Field labels**	**Tokens**	**Token labels**
90292	Zip	90292	ZipValue
$76°F$	TempF	76	TempFValue
		$\circ$	DegreeSymbol
		F	TempFUnit
50%	Humidity	50	HumidityValue
		%	PercentSymbol
$5mph$	WindSpeed	5	WindSpeedValue
		mph	WindSpeedUnit
Los $Angeles,$ CA	Place	Los	CityName
		$Angeles$	CityName
		,	Symbol
		CA	StateAbbr
$4:05\ pm$ EDT	Time	4	HourValue
		:	TimeSeparator
		05	MinuteValue
		pm	AmPm
		EDT	TimeZone

Table 2: The various formats of writing the same value of Visibility in the weather domain.

Numeric Value	**Unit**
9	miles
	mi
	mi.
	MI.
	Mi.
	-no unit-
9.0	miles
9.00	miles
14.5	kilometers
	km
clear	

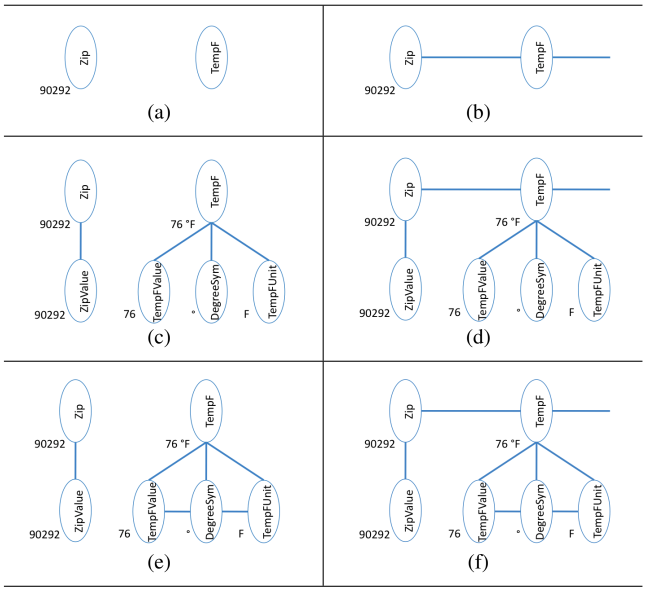
is the number of token labels.

The rest of the paper is structured as follows: In section 2, we give a brief overview of the CRF models. In section 3, we first explain our method of generating tokens from the fields and extracting features from them, which is common to all graph structures. Then we present different graph structures that we build incrementally to exploit more structure within the data. We describe how we generate these graphs, how we train the CRF models from them and our method of performing prediction on unlabeled graphs. We also discuss the complexity of each graph structure. In section 4, we present the results of our experiments and in section 5, we discuss the related work. We conclude in section 6.

2. Conditional random field models for labeling data

Conditional Random Field-based models [4] learn the probability distribution of labels conditioned on the evidence. The variables and their mutual probabilistic dependence is presented as nodes and edges respectively. There-

Fig. 1: Six graph structures that capture the various probabilistic dependencies between the data elements.



fore, the training examples are in the form of labeled graphs and prediction involves assigning values to the nodes of an unlabeled graph.

CRF models represent the relationships between the labels and the evidence and the relationships between mutually dependent variables (those that are connected via edges) in terms of feature functions. A feature function applies on a fully connected subgraph, called a clique. A feature function that applies on only one node is called a one-node feature function and that which applies on two nodes is called a two-node feature function. A feature function when applied on a clique, takes as input one label value for each node in the clique and checks if those values are the ones that it expects. It also considers the features of the evidence around those nodes. It returns a value of 1 or 0 depending on whether the required conditions are satisfied or not, respectively. Learning the model entails learning appropriate weights for these feature functions.

More formally, let the features of the evidence be represented by a feature vector, X, the labels be presented by the feature vector, $Y = \{y_i\}$, the feature functions be represented by $F = \{f_i\}$, and their weights by $W = \{w_i\}$. The potential for a clique is defined as:

$$\phi(clique) = exp\big(\sum_i f_i(y_{clique}|x)\big) \tag{1}$$

The potential of the whole graph is the product of the potentials of all the cliques. The sum of the graph potential for all possible label assignments is called the partition function and represented by $Z(X)$. The likelihood of a particular label assignment to the random variables $P(Y = y|X = x)$ is defined as the ratio of the graph potential for that label assignment to the partition function as follows:

$$1/Z(x) * exp\Big(\sum_c \big(\sum_k w_k f_k(y_c, x)\big)\Big) \qquad (2)$$

CRFs do not have a closed form solution for the weights. The weights are found by numerical optimization techniques such as gradient based approaches [8]. The gradient with respect to the weight w_k is given below. The term $p(y_c = y|x)$ is the marginal probability of the clique variables. These marginals are found using inference algorithms such as belief propagation [10]. The labels are predicted for an unlabeled graph using potential maximization algorithms, such as the Viterbi algorithm [16].

$$\sum_c f_k(y_c = y_k, x) - \sum_c \sum_y p(y_c = y|x) f_k(y_c, x) \qquad (3)$$

3. Exploiting structure within data for accurate labeling

There are three kinds of probabilistic dependencies that are generally seen among various data elements in structured data. They are as follows: 1) The dependency between the labels of neighboring fields, 2) The dependency between field labels and their token labels, and 3) The dependency between neighboring tokens within a field.

We can exploit all of these dependencies through different graph structures used for training the CRF models. Since a CRF graph represents the relationship between two random variables by means of edges between their nodes, it means that the more relationships one exploits, the more complex the graph structure becomes. A more complex graph structure also requires higher computational complexity inference techniques. Therefore, there is a trade-off between the numbers of relationships one exploits and the complexity and hence the time required to perform inference on these graphs.

In this work, we present six different graph structures constructed from the structured data by incrementally exploiting more structure within the data. These graphs are shown in figure 1. The simplest graph is one where there is only one node corresponding to each field. The purpose of the CRF model that uses this graph is to learn to predict the label for each field independently. This graph structure is show in figure 2(a) and is discussed in section 3.2. There is only one kind of label-to-label dependency that can be exploited here, the one between adjacent field nodes. The corresponding graph structure is shown in figure 2(b).

In addition to fields, the model can also take tokens into account for each field. These graphs are shown in figure 2(c) and 2(d) and are discussed in section 3.3.

Finally, we can also take into account the relationships between tokens. These graphs are show in figure 2(e) and 2(f) and are discussed in section 3.4.

In the following subsection, we discuss the process of tokenization of these fields and the extraction of features from the tokens.

3.1 Tokenization and Extraction of features

We generate the tokens from the fields using our own lexical analyzer. This lexical analyzer splits the field string at whitespaces and then splits the resultant parts in such a way that each token is either purely alphabetic (e.g., Cloudy), purely numeric (e.g., 76, -4.5), or a single symbol character (e.g., $^\circ$, %).

Once the tokens have been generated, we assign features to each token. There are three different classes of features, the ones that apply to purely alphabetic tokens, the ones that apply to purely numeric tokens, and the ones that apply to symbol tokens. Table 3 lists all the features that we use. Some of the features listed are generic features, whose particular instance is generated based on the token from which it is generated. For example, the feature *Starts_With_Alpha_*$\langle X \rangle$ is a generic feature and its particular instance for the token *Cloudy* will be *Starts_With_Alpha_C*. One of the important features is the identity of the token itself. Since CRF models assign weights to features based on their frequency of appearance, this feature is useful for those semantic types that consist of a small lexicon of terms (e.g., *Country Names*). The features used by us are generic and can apply to any domain. We have picked features that capture the basic properties of any token. It is very easy to add new domain specific features to this set of features, if it is so required. However, all our experiments use only these features.

Table 3: Features used to characterize the tokens.

Features	Description
Alphabetic features	
$Alpha\_Length\_\langle N \rangle$	Length. N = 1,2, ...
$Starts\_With\_Alpha\_\langle X \rangle$	First character. X = A, a, B, ...
$Capitalized\_Token$	Token is capitalized.
$All\_Uppercase\_Token$	Whole token is uppercase.
$Alpha\_Id\_\langle Token \rangle$	Token itself is feature. $Token$ = California, NW
Numeric features	
$Num\_Length\_\langle N \rangle$	Length. N = 1,2, ...
$Before\_Decimal\_Len\_\langle N \rangle$	# of digits before decimal.
$After\_Decimal\_Len\_\langle N \rangle$	# of digits after decimal.
$Negative\_Num$	Number is negative.
$Starting\_Digit\_\langle N \rangle$	First digit. N = 0, 1,2, ...
$Unit\_Place\_Digit\_\langle N \rangle$	Units place digit. N = 0, 1,2, ...
$Tenth\_Place\_Digit\_\langle N \rangle$	Tenth place digit. N = 0, 1,2, ...
Symbol features	
$Symbol\_\langle Sym \rangle$	Symbol itself is feature. Sym = %, :, $^\circ$

3.2 Graphs with field nodes only

Figure 2(a) shows the simplest graph structure generated from the data. Each field has a separate graph which contains only one node. This node represents the random variable associated with the field label. We tokenize the field into tokens, extract features for each token and then combine them into one set of features. These features are assigned to the field node. For example, the field node for '$76°F$' will have the features, Num_Length_2, $Starting\_Digit\_7$, $Symbol\_°$, $Alpha\_Length\_1$ and others, where the first two features have been extracted from token '76', the third feature has been extracted from '°' and the fourth feature has been extracted from the token 'F'.

We use only one class of feature functions for this graph structure. These feature functions are of the form:

$$f(field\_node, field\_label) \tag{4}$$

These feature functions return the value 1 if the field node has a feature p_m and input value $field\_label$ is some l_j. We find the marginals for a field node by finding the potential of the node for each value of label l_j and then dividing the potentials by the sum of the potentials, $Z(x)$. Once the marginals are calculated, the gradient can be calculated from them. We train our model in this way.

We predict the most likely label for a new field node by finding the label l_j for which the potential of the node and therefore of the graph (since the graph consists of only one node) is the highest. The complexity of the inference method as well as the labeling method is $O(L_f)$, where L_f is the number of field labels.

In every domain, it is generally the case that there is some order to the fields. This means that a field is immediately followed by one of a small set of fields. For example, the $Temperature$ field type is generally followed by $SkyCondition$ and a $WindSpeed$ field is generally followed by $WindDirection$. We exploit this relationship in the graph structure shown in 2(b). This graph is formed by connecting the fields in a linear chain. We capture this relationship using the following feature function:

$$f(field\_node1, field\_node2, field\_label1, field\_label2) \tag{5}$$

where it returns 1 if the input $field\_label1$ is some label l_j and input $field\_label2$ is some label l_k. Otherwise, it returns the value 0. Therefore, this graph uses the two classes of feature functions shown in equation 4 and 5.

We perform inference on this graph using the belief propagation algorithm [10]. We predict the most likely label assignment to the field nodes using the Viterbi algorithm [16]. The complexity of performing inference on this graph is again $O(L_f)$. Yet, the model uses more feature functions for these graphs. This graph structure better represents the field labels since they consider both the features of the fields as well as the neighboring fields. Our experiments show that these graph structures indeed give higher field labeling acurracy. This happens because the neighborhood information helps disambiguate between similar fields. For example, the fact that *Humidity* and *Sky Condition* are generally reported together prevented the model from mislabeling some of the *Humidity* values as *Chance Of Precipitation* because these values appeared close to a *Sky Condition* value. The model based on the graph structure in figure 2(a) did make some of these mistakes.

3.3 Adding token nodes to the field nodes

The features used to represent the fields in the models described above are derived from the tokens of the fields. It seems more appropriate to predict the identity of the tokens based on their own features and derive the identity of the field based on the tokens. For example, the field type $Temperature$ can have token types *TemperatureValue*, *DegreeSymbol*, and *TemperatureUnit* and the field type $Time$ can have the constituent token types $Hours$, $TimeSeparator$, $Minutes$ and $Seconds$. We create a new graph structure, where each field node will have token nodes as it children.

The graph structures shown in figures 2(c) and 2(d) show how these graphs look. We do not exploit the field-label-to-field-label relationship in the left figure, while we do so in the right figure.

The graph shown in figure 2(c) does not have any features belonging to the field node itself. Instead the features belong to the token nodes. We use two new classes of feature functions to represent relationships in these graphs. The first class of feature functions are written as:

$$f(token\_node, token\_label) \tag{6}$$

These feature functions take a $token\_label$ and a $token\_node$ as inputs and returns 1 if the $token\_node$ has a feature p_m and the $token\_label$ is some label l_j. These feature functions represent the dependency of the labels of token nodes on their features.

The second class of feature functions are written as:

$$f(field\_node, token\_node, field\_label, token\_label) \tag{7}$$

These feature functions take a $field\_label$ for a $field\_node$ and a $token\_label$ for a $token\_node$ and return 1 if the $field\_label$ is some label l_j and the $token\_label$ is some label l_k. These feature functions represent the co-occurence of the field labels with their token labels.

We use belief propagation to perform inference on these graphs. The graphs are already in the form of a tree, as required by the algorithm. The prediction for a new graph is made using a modified Viterbi algorithm called the max-sum algorithm [3]. This algorithm maximizes the sum of the log of the potentials of the various cliques in the graph.

Compared with the graph structure in figure 2(a), this graph is more natural because it finds the labels of each token based on its own features and then finds the label of the field based on the labels assigned to its tokens. Yet, this graph introduces more nodes to be labeled and also introduces a large set of token labels to chose from. This increases the ambiguity while training as well as during prediction. As we show in our experiments, this graph structure gives higher field labeling accuracy than the graph structure that has only one node in the graph. This happens because when the features of the tokens are combined into one set (as in figure 2(a) and 2(b)), most of the features from different semantic types are similar. For example, if we look at the semantic type $Temperature$ and $Humidity$, both these types have numeric values, and the values are between 0 and 100. The main difference appears in the fact that $Temperature$ values end with a F or a C, but $Humidity$ ends with a percentage (%) symbol. But, when these features are distributed over their own tokens, the most distinguishing tokens namely the F or C in one and the % symbol in the other have very distinct features. The CRF model can differentiate between these much better. With high confidence on these tokens, the field labels tend to be correctly assigned due to their co-occurence relationship with these tokens labels (e.g., the strong co-occurence of $Temperature$ with $TemperatureUnit$).

In addition, this graph structure also solves an important problem of identifying the semantic types of the various components within a field. For example, in order to compare two $Date$ fields that are written in different formats, (e.g., 07/21/2011 and 21 July, 2011), it is required that the semantic meaning of the various components of each date string are known so that they can be compared effectively. Therefore, the graph structure described here improves the accuracy of labeling while solving a larger problem.

The graph shown in figure 2(d) is an extension on the graph described above. In this graph, we also join the field nodes via edges. We exploit the neighboring relationships among the field nodes in the same manner as we did in the graph in figure 2(b) via the feature function presented in equation 5.

In order to perform inference in these graphs, we first need to convert them into a tree structure. We perform this conversion as follows: we make the first (leftmost) field node the root of the whole graph. All its token nodes are already its children. We also make the field node to its right, its child. This is repeated at each field node. This gives us a tree, where each node has one parent and each node can have zero or more children. We then apply belief propagation algorithm to this graph to perform inference.

Prediction on a new unlabeled graph is performed by first converting it into a tree structure as described above and then applying the max-sum algorithm [3] starting from the leaf nodes to find the most likely joint assignment of labels to all the field and token nodes. This graph structure is more complex than the one described above. We show in our experiments that exploiting the field-node-to-field-node label dependency helps improve the field labeling accuracy over the simpler model that does not exploit these relationships, similar to the results obtained for the graph structures in figure 2(a) and 2(b).

3.4 Connecting the tokens with each other

There is an important relationship between neighboring tokens within a field. In general the order of the token types within a field remains roughly constant. For example, $Minutes$ always follows $Hours$ in every format of $Time$, *Temperature Unit* (‘F’ or ‘C’) always appears after the *Temperature Value* token within a $Temperature$ field and *State Name* always written after $CityName$ within an $Address$ field. To exploit these strong relationships, we connect the tokens within the fields together. Figures 2(e) and 2(f) show how the graphs look in this case. Figure 2(e) shows the graphs where we do not exploit the neighborhood relationships between fields, whereas figure 2(f) shows the graph structure where we do.

We represent the relationships between labels of neighboring token nodes by a new class of feature functions. These feature functions are written as:

$$f(token\_node1, token\_node2, token\_label1, token\_label2) \tag{8}$$

This function returns the value 1 only when $token\_label1$ is some token label l_j and $token\_label2$ is some token label l_k. Therefore, this graph has three classes of feature functions. Two of these classes are the same as we used in the graph structure shown in figure 2(c).

We cannot use belief propagation to perform inference on these graphs since they have cycles in them. Therefore, we convert each graph into a different graph called a junction tree [5]. A junction tree (JT) is formed by replacing all the three-node cycles in the original graph by a single junction tree node (JT node) and connecting the JT nodes that share at least one node in the original graph. See [5] for the detailed general algorithm for converting any graph into a junction tree. We convert our graphs into junction trees as follows: if the field has only one or two token nodes, then it is replaced by just one JT node. Otherwise, for each field, we take two token nodes at a time starting from the left and replace the three-node cycle formed by them and the field node with one JT node. Figure 2 shows the junction tree constructed from the CRF graph in figure 2(e). Since, the adjacent JT nodes share the field node and a token node, they are connected with each other. This always gives us a linear chain of $(N - 1)$ JT nodes for N token nodes. Since this is a linear chain, we apply belief propagation to this graph. The possible states that a JT node can take is a power set of the labels that each node within the clique that they represent can take. Therefore the complexity of the belief propagation algorithm is equal to:

$$L_f \times L_t^2 \tag{9}$$

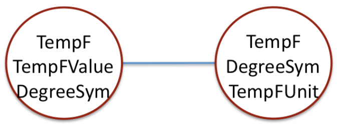


Fig. 2: The junction tree for one field and its tokens.

where L_f is the number of field labels and L_t is the number of token labels. This complexity is low enough to allow us to apply the belief propagation algorithm on graphs of practical sizes.

To label a new graph, we convert it into a junction tree, as explained above. We then use the Viterbi algorithm to find the most likely states for all the JT nodes. The Viterbi algorithm is constrained so that the states on the nodes that are shared by two adjacent JT nodes are the same. For example, since the two JT nodes in figure 2 share the field node and the middle token node, both JT nodes can only take values so that the labels values for the two shared nodes in the same.

Figure 2(f) is the most complex graph structure that we use. It exploits all the three dependencies among the field and the tokens. It uses the three feature functions used for the graph structure explained above. In addition it also uses the field label to field label feature function that we used in the graphs in figure 2(b) and 2(d) to exploit the neighborhood relationships among field labels.

Like in the previous graph, this graph also contains cycles. As a result we again convert these graphs into junction trees. Since the field nodes are connected to each other in a line, the two-node-cliques formed by consecutive field nodes are also converted into JT nodes for these graphs. These JT nodes share the field nodes with the JT nodes formed by token nodes under these field nodes (as explained for the previous graph). This results in a linear chain junction tree as shown in figure 3. Due to the linear structure, belief propagation algorithm can be applied to this graph. Since, the new JT nodes represent two field nodes, the number of states that they can take is:

$$L_f^2 \tag{10}$$

where the terms mean the same as before. Generally, this term will always be smaller than $L_f \times L_t^2$ since the total number of tokens is generally larger than the total number of fields. Therefore the complexity of performing inference on this algorithm remains bounded as before.

We similarly generate a CRF graph from unlabeled data, convert it into a linear chain junction tree and then use the Viterbi algorithm on it to predict the labels for fields and tokens.

4. Experiments on real world data

We tested the six graphical structures in three different domains: weather forecast, flight status and geocoding. For

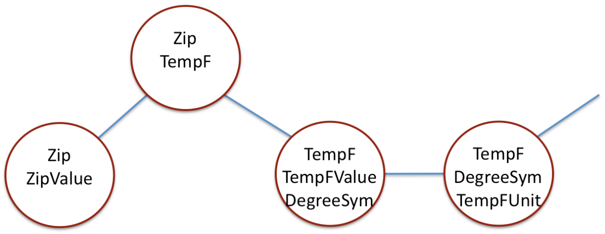


Fig. 3: The junction tree for the fully connected graph in figure 2(f). We have drawn the JT node formed from the field nodes' clique slightly elevated to differentiate it from the JT nodes formed from field and token nodes. Yet, this is a linear chain.

each domain, we scraped data[1] from 100 different web pages of four different web sites to obtain 400 tuples of data in each domain. We also defined a set of field level and token level semantic classes that we wanted to identify in each domain. We tokenized the fields in each tuple using our lexical analyzer. We then labeled all the fields and their tokens using the labels for the semantic classes already defined. We then extracted the features for each token. The average number of fields per tuple in the weather forecast domain was 31, in flight status domain it was 14 and in the geocoding domain it was 3.5. Table 4 shows the web sources used in each domain and also mentions the number of field and token semantic types in each domain.

Table 4: Experimental setup

Domain	Details	**Data sources**
weather forecast	#field types = 15 #token types = 37	wunderground.com
		unisys.com
		weather.com
		noaa.com
flight status	#field types = 8 #token types = 17	flytecomm.com
		flightview.com
		continental.com
		delta.com
geocoding	#field types = 5 #token types = 12	geocoderus.com
		geocoderca.com
		geonames.com
		worldkit.com

We ran four experiments on every graph structure in each of the three domains. In each experiment, we trained a CRF model on the graphs created from 300 tuples from three sources and tested it by labeling the 100 tuples from the fourth source. We then averaged the field and token labeling accuracy for each graph structure. Table 5 shows the average field labeling accuracy for each graph structure in each domain. The geocoding domain is a relatively simple domain with only five field types. Therefore, even the simplest graph structure gives high field labeling accuracy and there is little scope for improvement as we take advantage of more

[1]We used a tool called AgentBuilder, from Fetch Technologies.

structure within the data. However, there is an improvement of 1% from the simplest graph structure to the most complex graph structure in the field labeling accuracy.

All the differences in the table for the other two domains are statistically significant for $p = 0.05$. The results for the weather forecast and the flight status domains show that the accuracy increases each time we exploit additional relationship among the labels. For example, the accuracy for graph structures where the relationship between the neighboring field labels is exploited (right column) is always more than the accuracy for the graph structures where it is not (left column), except in only one case for the flight status domain where the source *flytecomm.com* has dates that are in a very different format from other sources and also appear out of the normal order so that all of them get consistently mislabeled. The accuracy also increases when we exploit the relationship between the field labels and the token labels. Further, when we connect the adjacent token nodes to exploit their neighborhood relationships, we get a significant improvement in labeling accuracy. There is an improvement of 10% in the weather domain and 7% in the flight domain from the graph structure that has only one field node to the graph structure that has both the field and token nodes and all of them are connected. These results clearly show that exploiting each new relationship gives us a better model and hence higher field labeling accuracy. These experiments represent the real accuracy that one might achieve while using these graph structures since we never test our models on the same data that we trained it on. Yet, despite the fact that each new source uses its own formats and representations, the model is able to successfully identify the semantic labels for a high percentage of fields.

The following is an example of how the model exploits the inter-label relationships to correctly label a field that uses a format not seen before by it: The weather source, *unisys.com*, represents temperature in fahrenheit (*TemperatureF*) using only a numeric value and the unit, and without a degree symbol. (for example, '$62\ F$'). None of the other three sources, on which the model was trained had such an example of this class. Yet, the model was able to assign the correct semantic label to these fields by taking advantage of the strong relationship between the field level label *TemperatureF* and the token level labels *TemperatureFValue* and *TemperatureFUnit*, and a strong correlation between the *TemperatureFUnit* token label and the feature *Alpha_Id_F*. In contrast, in the graph structure in figure 2(a), all such fields that had integer values were labeled as *Humidity* due to the lack of the differentiating feature $Symbol\_^{\circ}$ and a strong correlation between the label *Humidity* and the feature *After_Decimal_Len_0*.

Table 6 shows the token labeling accuracy results for the four graph structures in all three domains. There is an improvement in token labeling accuracy every time we add one more relationship to the model for use. There is an average improvement of 5% in the weather domain and 6% in the flights and geocoding domain between the simplest graph structure that involves tokens (figure 2(c)) and the graph structure that exploits all relationships (figure 2(f)). The average token labeling accuracy is around 85% across all the different graph structures and domains. This means that the model is able to correctly identify the semantic types of the tokens within the fields with high accuracy. Some of the tokens types are so similar to each other that it is difficult to differentiate between them. For example, the token types, $TemperatureValue$, $HumidityValue$, $PressureValue$, and $VisibilityValue$ all look very similar to each other and yet our models are able to use the identity of the neighboring fields and tokens to correctly identify them.

Table 5: Result of experiments on all six graph structures. The average field labeling accuracy improves as we exploit more structure within the tuple and the fields.

Domains		**Fields not connected**	**Fields connected**
Weather forecast	**Fields only**	0.79	0.83
	Field and tokens	0.80	0.85
	Tokens connected	0.85	**0.89**
Flight status	**Fields only**	0.90	0.88
	Field and tokens	0.90	0.93
	Tokens connected	0.93	**0.97**
Geo-coding	**Fields only**	0.97	0.97
	Field and tokens	0.98	0.98
	Tokens connected	0.98	**0.98**

Table 6: The average token labeling accuracy for experiments on all six graph structures.

Domains		**Fields not connected**	**Fields connected**
Weather forecast	**Field and tokens**	0.81	0.85
	Tokens connected	0.83	**0.86**
Flights status	**Field and tokens**	0.81	0.84
	Tokens connected	0.85	**0.87**
Geo-coding	**Field and tokens**	0.84	0.85
	Tokens connected	0.89	**0.90**

An alternate approach to assigning semantic labels to fields is based on matching unlabeled fields with regular expressions that are generally known to represent a particular semantic class. We compare a CRF model utilizing the graph structure in figure 2(f) with a sophisticated regular expression based pattern matching model proposed by us in [7]. In this work we generate regular expressions of varying generality to match the given labeled examples. For example, patterns such as 25, *2-digit*, and *numeric* are generated for a temperature value 25. The labels are assigned to unlabeled fields based on how well they match the regular expressions for each semantic type. We used the same experimental set up for running experiments on our previous approach. The average field labeling accuracy of both the approaches is presented in table 7.

In all the three domains, the labeling accuracy of the CRF model is much better than the accuracy for the regular expression based approach. This is because the regular expression based approach it is unable to generalize the structure of a semantic class from the examples that it has seen. Therefore, even a slight change in the structure, such as the introduction of a dot after an abbreviated unit can mislead it. On the other hand, since our model learns many different types of dependencies among labels, even if some such relationships do not hold true due to changes in the structure, other dependencies such as the relationship with neighboring fields can still help the model make the right prediction.

Table 7: Comparison of our graph structure that exploits all the three relationships with a regular expression based approach. The new model performs much better on all domains.

Domains	Regular expression based model	CRF model
Weather	0.65	0.89
Flight status	0.42	0.97
Geocoding	0.36	0.98

5. Related Work

Semantic annotation is a problem that occurs in various domains and on various kinds of documents. Named entity extraction [9] is a form of semantic annotation problem where the task is to assign named entity tags to various words or phrases in a text document such as a news report. Similarly, semantic annotation can also be applied to reports, where the task is to identify the various pieces of information such as telephone numbers, etc. An example of semantic annotation in semi-structured sources is the problem of identifying the various information fields on webpages [6]. Finally, the semantic annotation of structured sources such as databases is performed under the name of schema matching [2]. In this paper, we solve the problem of assigning semantic labels to structured data obtained from databases or web pages. Unlike schema matching, we do not label entire columns of data. Instead we assign labels to all fields in a tuple, which corresponds to a row in a database table. The advantage of our approach is that it is applicable to sources with missing and optional fields since we do not need the data to be available in a regular tabular format.

CRF models were proposed by Lafferty et al. [4], where they presented experiments on linear chain graph structures for performing part-of-speech tagging of natural text. Researchers then applied the same structure to other problems such as noun phrase chunking [13] and extracting tables from documents [11]. Later researchers demonstrated the use of other graphical structures such as factorial graphs [14], two-dimensional grid structures [17] and tree-structured graphs [15]. In this paper, we use various kinds of graph structures, which include linear chain graphs, hierarchical tree structured graphs and also cyclic graphs. We compare their capability in exploiting the various patterns within data and also examine the computational complexity of performing inference on them. In addition, we also show that the cyclic hierarchical tree structure graph (Figure 2(f)) is well suited to accomplish the task of semantic annotation and achieves high accuracy both in field and token labeling.

There have been two main works using CRF models to perform semantic annotation. Zhu et al. [17] used the CRF model to assign semantic labels to the image, description, price and title of a product on a commercial product webpage. They mapped the web objects to two-dimensional grids and exploited the spatial proximity and relationship to assign labels to them. In the other work, Tang et al. [15] used tree-structured graphs to represent the layout of the information elements on semi-structured reports. In both these cases, the researchers exploit the dependence of the field labels on their structures and the relationship between adjacent field labels. In this paper, we present six different graph structures that exploit different combinations of inter-label relationships including the graph that only exploits the relationship between adjacent field labels (figure 2(b)). We show that we can achieve a significant improvement in the labeling accuracy if we exploit the relationships between the field labels and the token labels as well as the relationships among neighboring token labels.

One of the other advantages of our graph structures is that we also assign semantic labels to the tokens of the fields. This is essential in comparing data values from different sources which might be using different formats. Borkar et al. [1] used HMMs [12] to assign semantic labels to the tokens of US addresses. They trained a linear chain HMM model that learns the transition probabilities beween the various labels and the emission probability of the various kinds of tokens from these labels. The semantic types used by them are: *HouseNumber, StreetName, CityName, StateName, ZipCode*. In contrast to their approach, we use a more powerful model than HMMs, which allows the use of overlapping features. Also, we do not need to train a separate model for each field level semantic type. Instead our model can simultaneously identify the field level type of an item of data as well as the semantic types of its tokens. For example, in the geocoding domain, our model can identify the semantic type of a field as being *Latitude*, *Longitude* or *Address* and also identify their token labels such as *LatitudeDegree*, *LatitudeMinutes*, *HouseNumber*, and *StateName*.

An alternate approach to graphical models, such as CRFs and HMMs, that can be applied to the problem of semantic annotation involves learning the syntactic rules that describe the syntax of the semantic classes using regular expressions. We proposed this approach in [7]. In this approach, we generate regular expressions of varying generality from the examples of the semantic types and then match these with new unlabeled examples to predict their label. We presented results comparing the approach proposed in this paper with our previous work and show that the new approach performs much better as compared to the previous approach. The previous approach could not handle variations in the formats easily and fails as soon as the format changes even slightly. In addition, it cannot take advantage of the information available about the neighboring fields and tokens. Since our model exploits many different classes of relationships and combines them in a probabilistic way, it can handle such variations because many of these relationships still hold true for the new format.

6. Conclusion

In this paper we have shown that we can get high labeling accuracy from CRF models by exploiting the latent structure within data. We showed that in spite of increased complexity and higher number of labels, exploiting more structure improved the labeling accuracy for fields as well as the tokens. We also showed that the complexity of the graph structure that exploits all three relationships is bounded by $L_f \times L_t^2$, where L_f is the total number of field labels and L_t is the total number of token labels.

There are many domains where entities have such hierarchical structure within them. For example, this approach can also be applied to assign part-of-speech labels to phrases within sentences, where the phrases can be split into words and the relationships between the labels of adjacent words can be used to achieve higher labeling accuracy for phrases. Similarly this approach can also be used for named entity recognition to identify top level entities such as *Address* as well as low level entities within them, such as, *City Name*, *State Name*, or *Country Name*.

One of the factors that impact the speed of training is the number of feature functions used in the model. Presently, we use all feature functions that can be generated from the training examples. In future, we will explore techniques of pruning this space to reduce the training time. We will also experiment with more complex feature functions that can be formed from conjunctions or disjunctions of elementary features functions. For example, a feature function that says that an *Hour* token can either have one or two digits is a better descriptor of this semantic type than two separate feature functions, one of which associates the feature of having one digit with the semantic type and the other associates the feature of having two digits with it. Both of the above mentioned directions can reduce the total number of feature functions used in the CRF model, while building more expressive CRF models.

7. Acknowledgments

This research is based upon work supported in part by the Intelligence Advanced Research Projects Activity (IARPA) via Air Force Research Laboratory (AFRL) contract number FA8650-10-C-7058. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon.

The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of IARPA, AFRL, or the U.S. Government.

References

[1] V. Borkar, K. Deshmukh, and S. Sarawagi. Automatic segmentation of text into structured records. In *Proceedings of the 2001 ACM SIGMOD International Conference on Management of Data*, 2001.

[2] A. Doan, P. Domingos, and A. Y. Levy. Learning source descriptions for data integration. *WebDB*, 81-86, 2000.

[3] F. R. Kschischang, B. J. Frey, and H. A. Loeliger. Factor graphs and the sum-product algorithm. *IEEE Transactions on Information Theory*, 47(2), February 2001.

[4] J. Lafferty, A. McCallum, and F. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the Eighteenth International Conference on Machine Learning*, pages 282–289, 2001.

[5] S. L. Lauritzen and D. J. Spiegelhalter. Local computations with probabilities on graphical structures and their application to expert systems. *Journal of the Royal Statistical Society*, 50(2):157–224, 1988.

[6] K. Lerman, L. Getoor, S. Minton, and C. A. Knoblock. Using the structure of web sites for automatic segmentation of tables. In *Proceedings of ACM SIG on Management of Data*, 2004.

[7] K. Lerman, A. Plangrasopchok, and C. A. Knoblock. Semantic labeling of online information sources. *IJSWIS, special issue on Ontology Matching*, 2006.

[8] D. C. Liu and J. Nocedal. On the limited memory method for large scale optimization. *Mathematical Programming*, 45(3):503–528, 1989.

[9] D. Nadeau and S. Sekine. A survey of named entity recognition and classification. *Linguisticae Investigationes*, 30(1):3–26, 2007.

[10] J. Pearl. *Probabilistic Reasoning in Intelligent Systems*. Morgan Kaufmann, 2nd edition, 1988.

[11] D. Pinto, A. McCallum, X. Wei, and W. B. Croft. Table extraction using conditional random fields. In *Proceedings of 26th annual international ACM SIGIR conference*, 2003.

[12] L. R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, pages 257–286, 1989.

[13] F. Sha and F. Pereira. Shallow parsing with conditional random fields. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology*, 2003.

[14] C. Sutton, A. McCallum, and K. Rohanimanesh. Dynamic conditional random fields: Factorized probabilistic models for labeling and segmenting sequence data. *Journal of Machine Learning Research*, 2007.

[15] J. Tang, M. Hong, J. Li, and B. Liang. Tree-structured conditional random fields for semantic annotation. In *Proceedings of 5th International Conference of Semantic Web*, 2006.

[16] A. J. Viterbi. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Transactions on Information Theory*, 13(2):260–269, 1967.

[17] J. Zhu, Z. Nie, J. Wen, B. Zhang, and W. Ma. 2d conditional random fields for web information extraction. In *Proceedings of the 22nd International Conference on Machine learning*, 2005.