Learning Semantic Models of Data Sources Using Probabilistic Graphical Models

Binh Vu
USC Information Sciences Institute
Marina del Rey, CA
binhvu@isi.edu

Craig A. Knoblock
USC Information Sciences Institute
Marina del Rey, CA
knoblock@isi.edu

Jay Pujara
USC Information Sciences Institute
Marina del Rey, CA
jpujara@isi.edu

ABSTRACT

A semantic model of a data source is a representation of the concepts and relationships contained in the data. Building semantic models is a prerequisite to automatically publishing data to a knowledge graph. However, creating these semantic models is a complex process requiring considerable manual effort and can be error-prone. In this paper, we present a novel approach that efficiently searches over the combinatorial space of possible semantic models, and applies a probabilistic graphical model to identify the most probable semantic model for a data source. Probabilistic graphical models offer many advantages over existing methods: they are robust to noisy inputs and provide a straightforward approach for exploiting relationships within the data. Our solution uses a conditional random field (CRF) to encode structural patterns and enforce conceptual consistency within the semantic model. In an empirical evaluation, our approach outperforms state of the art systems by an average 8.4% of F_1 score, even with noisy input data.

CCS CONCEPTS

• **Information systems → Information integration**; • **Computing methodologies → Learning in probabilistic graphical models**.

KEYWORDS

Semantic models, knowledge graph, probabilistic graphical models, semantic web, linked data, ontology

ACM Reference Format:
Binh Vu, Craig A. Knoblock, and Jay Pujara. 2019. Learning Semantic Models of Data Sources Using Probabilistic Graphical Models. In *Proceedings of the 2019 World Wide Web Conference (WWW '19), May 13–17, 2019, San Francisco, CA, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3308558.3313711

1 INTRODUCTION

There is an enormous number of data sources available on the web. Integrating information from multiple data sources enables interesting applications that can have a large social impact [27]. However, because data publishers usually publish data in different data formats and using different conventions, harvesting information can be challenging. The task of semantic modeling addresses this challenge by building a model that maps the information found in data sources to semantically meaningful structures.

A semantic model is a graph where ontology classes and source attributes are nodes and ontology properties are edges. Figure 1 depicts a semantic model for a sample data source about artworks, which contain information about paintings, their creators and sitters. By explicitly encoding the semantic type and relationship between source attributes in the graph, the model precisely describes the intended meaning of the data source. Moreover, it can be used to automatically generate R2RML rules to transform the data source to RDF triples for publishing to knowledge graphs [7, 25].

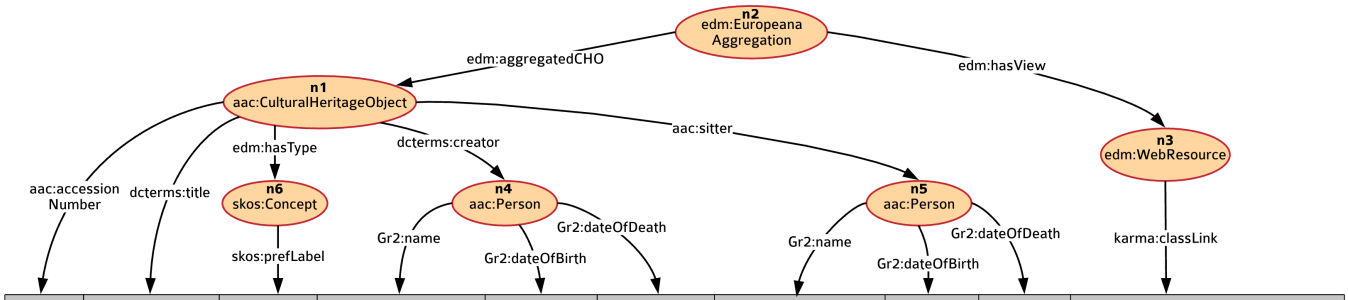
Since creating semantic models, or semantic modeling, requires significant effort and expertise, it is desirable to have an automated or semi-automated system to support users. There is previous work in predicting the semantic types of source attributes [18, 20, 22], called semantic labeling. A semantic type is a pair of ontology class and predicate. For instance, in Figure 1, the semantic type of attribute *name* is ⟨*aac:Person*, *Gr2:name*⟩. However, semantic labeling alone is insufficient for describing and publishing data as RDF because it fails to capture the relationships between attributes. Other work focuses directly on semantic modeling [29, 30]. The state-of-the-art methods use predicted semantic types of source attributes from semantic labeling systems, then build a minimum-weighted tree (Steiner Tree) that connects the attributes together. Heuristic functions and other machine learning methods, such as frequent pattern mining, are used to assign cost to edges of the tree.

Generating semantic models automatically still remains challenging for several reasons: the data of attributes may be very similar, and there is more than one possible relationship between entities. For example, it is easy to confuse a *date* attribute with created date of paintings or creators' birth date. If we simply look at one signal from the cost of the edge, it would be hard to decide which one is correct because they are similar. However, there are also multiple weak signals from its relationships with other remaining attributes that can assist our decision making. For instance, if we group values of *date* attribute by painting id and find that one painting appears to have been painted at two different points in time, then it is likely that this *date* attribute is not the created date.

To address these issues, we present a new approach for semantic modeling that uses a probabilistic graphical model (PGM). A PGM allows us to express and capture the weak signals within the data. From a set of known semantic models, we auto-generate positive and negative examples representing many possible descriptions of sources. Using this training set, the PGM is trained to distinguish between good and bad models. To predict the most probable semantic model for a data source, we use beam search to explore the

WWW '19, May 13–17, 2019, San Francisco, CA, USA

ACM ISBN 978-1-4503-6674-8/19/05.
https://doi.org/10.1145/3308558.3313711



ref	title	classification	artist	ArtistBornDate	ArtistDiedDate	name	bornDate	diedDate	image
NY270002	Chester Alan Arthur	Painting	Eastman Johnson	29 Jul 1824	5 Apr 1906	Chester Alan Arthur	5 Oct 1829	18 Nov 1886	http://npgportraits.si.edu/.../ny270002.jpg
NPG.70.36	Apollo 11 Crew	Painting	Ronald B. Anderson	1929	-	Neil Alden Armstrong	5 Aug 1930	25 Aug 2012	http://npgportraits.si.edu/.../7000062a.jpg
						Edwin Eugene Aldrin, Jr.	20 Jan 1930	-	
						Michael Collins	31 Oct 1930	-	
NPG.67.62	Chester A. Arthur	Painting	Ole Peter Hansen Balling	13 Apr 1823	1 May 1906	Chester Alan Arthur	5 Oct 1829	18 Nov 1886	http://npgportraits.si.edu/.../6700114a.jpg

Figure 1: NPG data source and its semantic model

space of possible models and evaluate the likelihood of the model using the trained PGM as a scoring function.

The contribution of this paper is a novel way to explore relationships within data sources and semantic models using a PGM, as well as a flexible framework that can easily be extended to incorporate new features. We present experimental results on two museum datasets published in Taheriyan et al. [29]. In the experiments, we control different levels of noise by using four different semantic labeling methods. These experiments show that our approach outperforms state-of-the-art systems by an average of 8.4% F_1 score, even with noisy input data.

2 MOTIVATING EXAMPLE

In this section, we provide an example to describe the typical process of building a semantic model for a sample data source of the National Portrait Gallery (NPG[1]) museum in Figure 1. On top of the table is a semantic model of the NPG. The source attributes are data nodes or leaf nodes. The ontology classes are class nodes, and links between nodes are ontology predicates. We also say that links between two class nodes are class links. Additionally, data links are links between class nodes and data nodes. For example, *aac:Person* and *aac:CulturalHeritageObject* are class nodes, while the source attributes *title* and *classification* are data nodes. The ontology classes and predicates are from DCTerms, SKOS, ElementsGr2, AAC, EDM ontologies[2].

The first step in building a semantic model is semantic labeling. A user or an automated semantic labeling system annotates each source attribute with semantic types. For example, the attributes *title* and *image* are labeled as ⟨*aac:CulturalHeritageObject*, *dcterms:title*⟩, ⟨*edm:WebResource*, *karma:classLink*⟩, respectively. Note that we use a special predicate *karma:classLink* to indicate that the data node *image* contains URIs of *edm:WebResource*.

The second step is specifying relationships between nodes in the model. This is a complicated problem, as there are several paths to connect two nodes in the domain ontology. For instance, *aac:Person* can be *dcterms:creator* or *aac:sitter* of *aac:CulturalHeritageObject*. Another example is that *bornDate* and *artistBornDate* do not belong to same person. If we only know the scores of semantic types of attributes and how frequent a predicate is used to connect two class nodes, we may not know which path is correct.

When users model a data source, they use various information to make decisions. For example, in Figure 1, we have three people who are listed under attribute *name* and are in the "Apollo 11 Crew" painting. As a painting is usually painted by one individual, a person would conclude that these are the names of the sitters rather than the names of the creators. Attributes *artist* and *ArtistBornDate* are two columns next to each other. This is weak evidence indicating that the two columns contain the name and birth date of a person. If the *bornDate* is mislabeled as death date of *aac:Person*, we would be skeptical about having Eastman Johnson draw a painting of Chester A. Arthur when he was 5 years old. Although these signals are useful, they are not always strong and correct. Can we learn to combine and apply the signals to build a correct semantic model?

3 PROBABILISTIC GRAPHICAL MODELS FOR SEMANTIC MODELING

Our problem of learning semantic models is defined as follow: suppose we have a set of domain ontologies O, a set of sources $\{s_1, s_2, ..., s_n\}$ and their semantic models $\{sm(s_1), sm(s_2), ..., sm(s_n)\}$. Given a target source $s(a_1, a_2, ..., a_n)$, in which a_i is a source attribute, we want to automatically build its corresponding semantic model $sm(s)$.

Our approach for building semantic models is similar to the procedure humans use. We model source attributes one at a time. For every attribute, we rank its modeling options based on the likelihood of the overall semantic model. Then, we select the attribute that has the highest rank. The process ends when there is no attribute left.

As we have discussed, users can exploit multiple signals to assist their decisions. For example, in Figure 1, given two options for relation ⟨n_1_*CulturalHeritageObject*, n_4_*Person*⟩: *dcterms:creator*

[1] http://npg.si.edu/home/national-portrait-gallery
[2] These ontologies are used in the evaluation datasets

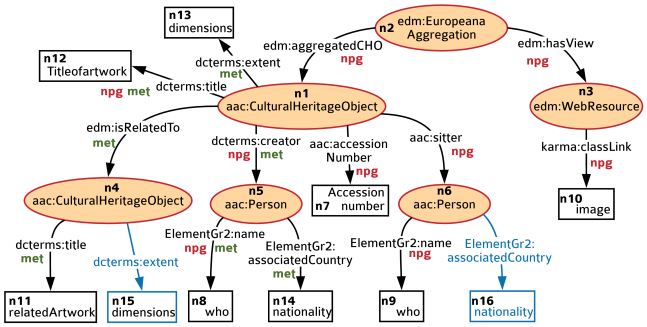


Figure 2: A example of integration graph[3]**.** This is built from sm(npg) and sm(met). Some nodes and properties are hidden for readability.

and *aac:sitter*, users can use the fact that name of n_4_*Person* is linked to the attribute *artist* to select predicate *dcterms:creator* over *aac:sitter*. Those signals are often collective and structural. They can be captured and represented naturally using a PGM, which is a very powerful framework that captures dependencies between relations and is robust to noise.

In Section 3.1, we describe the procedure to build a semantic model using search. We then explain how we apply and train a PGM as a ranking function in Section 3.2 and 3.3.

3.1 Searching for semantic models

Let $D_0 = sm(\{\})$ be a start state which is an empty model, $f : D_0 \rightarrow D_1$ be a transition function to move from one state to another state by adding one source attribute. Then, in general, constructing a semantic model can be solved using search algorithms by repeatedly invoking the transition function f until we reach the goal state, a tree connecting all source attributes.

Algorithm 1: TransitionFunction

Input: A search state: s_t
A set of source attributes: $A = \{a_0, ..., a_n\}$
An integration graph: G_{int}
Output: List of next search states

```
1 nextStates ← []
2 A' ← A \ s_t.leafNodes
3 for a_i ← A' do
4     trees ← MergeIntoTree (s_t, G_int, a_i)
5     for s_{t+1} ← trees do
6         nextStates.append(s_{t+1})
7 return nextStates
```

The transition function we use is defined in algorithm 1. We start with each unmerged attribute (line 2 - 3) and generate next states by merging them into current state (line 4). Algorithm 2 is used to combine an attribute with the current semantic model. The intuition of algorithm 2 is that we find all possible connecting paths between source attribute and leaves or root of the tree. Each path with a current tree forms a new state we want to generate.

[3]This example is based on an example from [29]

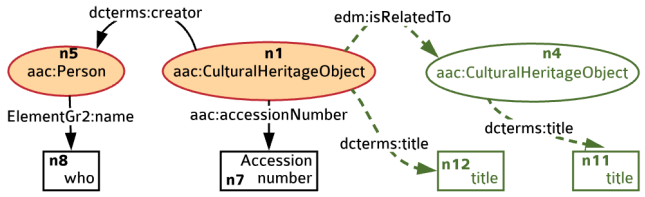


Figure 3: Two possible paths to add *title* attribute into current tree

To find the paths connecting source attribute and the existing tree, we use an integration graph from Taheriyan's work [29]. An integration graph is a directed weighted graph built from known semantic models, and represents a space of plausible semantic models we have seen in the training data. In Figure 2, we show an integration graph built from two data sources *npg* and *met*. Each link is annotated with its source name to specify where it comes from. When the graph is used in prediction, it is expanded by adding semantic types depicted in blue color. For example, in a new data source table, we have a column called *dimensions* tagged with a semantic type ⟨*aac:CulturalHeritageObject*, *dcterms:extent*⟩, we add one data node n_{15} and one link *dcterms:extent* from n_4 to n_{15}. However, we do not need to update node n_1 because we have already had link (n_1, *dcterms:extent*, n_{13}).

Now given an integration graph $G_{int} = \{V_{int}, E_{int}\}$, suppose that we are in the state s_t that we have four nodes $\{n_1, n_5, n_7, n_8\}$ as in Figure 3 and we want to merge attribute a_t = *title*. First, we need to align a_t to nodes in V_{int} that have same semantic type (line 2 in algorithm 2). The two nodes that match are n_{12} and n_{11}. For each node, we find all paths that connect the node to s_t, each path generates different tree. For example, there are two paths ($n_1 \rightarrow n_{12}$) and ($n_1 \rightarrow n_4 \rightarrow n_{11}$). Therefore, we have two next states. Each next state will be a tree that combines s_t and a merge path.

Algorithm 2: MergeIntoTree

Input: A search state: s_t
An integration graph: G_{int}
An attribute: a_t
Output: List of trees which contain the given attribute a_t

```
1 newTrees ← []
2 mntPts ← G_int.findAttribute(a_t.semanticType)
3 for mntPt ← mntPts do
4     connectedPaths ← all paths that connect leaves or root of
       s_t with mntPt
5     for path ← connectedPaths do
6         newTree ← merge mntPt to s_t using path
7         newTrees.append(newTree)
8 return newTrees
```

In addition to using the transition function in the beam search, we can also use it in interactive semantic modeling systems such as Karma [11]. In particular, from a current curated semantic model, the transition function is invoked to generate all semantic model

candidates. Then, the candidates are ranked using a scoring function to produce the top-k candidates for users to choose from. A user can also make corrections if there is no suitable option. In our approach, the scoring function directly affects the quality of our predicted semantic models. In the next section, we will describe how we use PGM to learn a scoring function.

3.2 Using Graphical Models as the score function

A probabilistic graphical model (PGM) is a well-known and effective framework to represent a probability distribution of random variables [13]. In order to apply PGMs in semantic modeling, we choose two kinds of random variables: input variables $\boldsymbol{x}$ that are links between nodes in a semantic model, and output variables $\boldsymbol{y}$ that are labels of the links. A label of a link has possible values in $\mathcal{V} = \{true, false\}$. A link with label *true* means it is correct and present in the gold semantic model. Since we are interested in predicting labels of links: $P(\boldsymbol{y}|\boldsymbol{x})$, it is natural to use a conditional random field (CRF). In particular, our standard CRF has following form:

$$P(\boldsymbol{y}|\boldsymbol{x}) = \frac{1}{Z(\boldsymbol{x}_c)} \prod_{C_p \in C'} \prod_{\Psi_c \in C_p} \Psi_c(\boldsymbol{y}_c, \boldsymbol{x}_c; \theta_p)$$

$$Z(\boldsymbol{x}_c) = \sum_{\boldsymbol{y}} \prod_{c \in C'} \prod_{\Psi_c \in C_p} \Psi_c(\boldsymbol{y}_c, \boldsymbol{x}_c; \theta_p)$$

where $\Psi_c(\boldsymbol{y}_c, \boldsymbol{x}_c; \theta_p)$ is a factor function that takes two set of variables $\boldsymbol{y}_c$ and $\boldsymbol{x}_c$ with parameters θ_p, and $Z(\boldsymbol{x}_c)$ is a partition function acted as a normalization factor to ensure the distribution $P(\boldsymbol{y}|\boldsymbol{x})$ sum to 1. The factors that have same input structure are grouped into $C' = \{C_1, C_2, ..., C_n\}$, where each C_i is a set of factors called *clique template* [26]. Factors in a clique template C_p share same parameters θ_p. A common choice of factor functions is exponential function, and each factor is log-linear with a set of feature functions f as follows:

$$\Psi_c(\boldsymbol{y}_c, \boldsymbol{x}_c, \theta_p) = \exp\left\{ \sum_k \theta_{pk} f_{pk}(\boldsymbol{y}_c, \boldsymbol{x}_c) \right\}$$

To simplify the notation, for every equation, x or y refers to one random variable, and $\boldsymbol{x}$ or $\boldsymbol{y}$ (bolded) refers to sets of random variables. Additionally, x and y symbols are used to denote input and output of a same link, respectively. For a complete guide about CRF, readers can refer to Sutton et al. [26].

Our ultimate goal is to construct a semantic model where all links are true. So the likelihood of the model being correct is $P(\forall y \in \boldsymbol{y}; y = true|\boldsymbol{x})$, and can be used as a score function for ranking. With the CRF framework above, in the following subsections, we will describe different clique templates and their features, each capture different relations between data sources and semantic models.

3.2.1 Link between nodes. These are factors over one link representing the likelihood of the link being correct or not. In particular, the factors have this form: $\Psi(y, x)$. The first feature of the factor is the confidence score of a semantic type, which is one of the outputs from the semantic labeling step. The features are defined as:

$$f_{ij}^{SS}(y, x) = \mathbf{1}_{\{y=i\}} \mathbf{1}_{\{st(x)=j\}} \text{st\_score}(x); \forall i \in \mathcal{V}, j \in \mathcal{ST}$$

$$f_{ij}^{\overline{SS}}(y, x) = \mathbf{1}_{\{y=i\}} \mathbf{1}_{\{st(x)=j\}} (1 - \text{st\_score}(x)); \forall i \in \mathcal{V}, j \in \mathcal{ST}$$

where $\mathbf{1}_{\{*\}}$ is an indicator function, $\mathcal{ST}$ is set of all possible semantic types, $st(x)$, $st\_score(x)$ are the semantic type and confidence score of link x, respectively. These are typical features used in graphical models, where each feature has its own weight and is non-zero only when a particular condition is met.

Note that if we only use $f_{ij}^{SS}(y, x)$, then $\Psi(y = true, x) - \Psi(y = false, x) = \text{st\_score}(x)(\theta_{true,j} - \theta_{false,j})$, then $\Psi(y = true, x) > \Psi(y = false, x)$ is independent of $\text{st\_score}(x)$ but only weights of the features. By introducing $f_{ij}^{\overline{SS}}(y, x)$, we are able to assign higher energy to $\Psi(y = true, x)$ when confidence score is high, and higher energy $\Psi(y = false, x)$ when confidence score is low (assuming that the higher confidence score, the more chance y is true). The simple and yet efficient technique above is applied to many other signals which have continuous value in this paper.

The confidence score signal is only applied for data links. For class links, we estimate the probability of a link being correct as follows:

$$P(s, x, o) = P(s) * P(o) * P(x|s, o)$$

$$P(x|s, o) = \frac{count(s, x, o)}{count(s, o)}$$

where s, o are source node and target node of a link x in the semantic model, respectively. In the equation, the probability of nodes $P(s)$ and $P(o)$ are estimated using a logistic regression (LR) model, which is trained on the same training set of our graphical model. The intuition of two features used in the LR model is described below:

- Total confidence score of semantic types of all child data nodes of n: a class node is likely to be present in the model if it links to many data nodes.
- The merging cost of node n with another node n' that has the same label in the graph: two class nodes with disjoint sets of properties should be merged together to simplify the model.

Features of class links are defined similar to f_{ij}^{SS} as follow.

$$f_{ij}^{CL}(y, x) = \mathbf{1}_{\{y=i\}} \mathbf{1}_{\{\text{triple}(x)=j\}} P(\text{triple}(x)); \forall i \in \mathcal{V}, j \in \mathcal{T}$$

$$f_{ij}^{\overline{CL}}(y, x) = \mathbf{1}_{\{y=i\}} \mathbf{1}_{\{\text{triple}(x)=j\}} (1 - P(\text{triple}(x))); \forall i \in \mathcal{V}, j \in \mathcal{T}$$

where $\text{triple}(x) = (s, x, o)$, $\mathcal{T}$ is set of all triples.

Beyond the two features above, we also use other features which depend mostly on x. To define these features, we define a series of observation functions $o_t(x)$. The corresponding features have the form:

$$f_{it}^{LO}(y, x) = \mathbf{1}_{\{y=i\}} o_t(x); \forall i \in \mathcal{V}$$

The set of observation functions we used are listed in table 1.

3.2.2 Cardinality relationships. The cardinality constraint between two source attributes can be used to discover an incorrect model. For example, a painting should have only one primary title, i.e the id and primary title of a painting have a one-to-one relationship. If we observe that the relationship is one-to-many, then one of the properties is linked to the wrong attribute. We express the heuristic

Table 1: Observation functions used in factor of link of nodes

Features	Explanation
Delta_SS(x=j) $\forall j \in \mathcal{ST}$	Difference between score of current semantic type and the remained best semantic type
SS_rank(x)	Rank of a predicted semantic type (determined by semantic labeling method
P(x|s)	Frequency of a link x given its source node s
NoDataChild	The source node has only child class nodes
NoSiblings	The source node has only one child node
Prior	Prior probability of a link

above using factors between two outgoing links of class nodes x_1 and x_2: $\Psi(y_1, y_2, x_1, x_2)$.

Cardinality relationship between attribute a_2 and attribute a_1 is computed by averaging the number of unique values of a_2 grouped by a_1. If the average number is greater than a threshold $\alpha = 1 + |\epsilon|, \epsilon \gtrapprox 0$, we consider this to be either a one-to-many or a many-to-many relationship, otherwise, it is a one-to-one relationship. The threshold is chosen to be slightly greater than 1 in order to make this feature robust to noise in the data source (e.g., missing values). In particular, α is set to 1.05 based on experimental results. For example, in Figure 1, the cardinality of attribute *name* and attribute *ref* is CRD(ref, name) = 1-to-many because $\frac{1+3+1}{3} = 1.67 > \alpha$. If two attributes belong to two separate nested lists, which occurs infrequently, we do not compute its cardinality relationship and set it to NULL. The cardinality features of this clique template are defined as:

$$f^{CRD}_{ijkmn}(y_1, y_2, x_1, x_2) = \mathbf{1}_{\{y_1=i\}}\mathbf{1}_{\{y_2=j\}}\mathbf{1}_{\{st(x_1)=k\}}\mathbf{1}_{\{st(x_2)=m\}} \mathbf{1}_{\{CRD(x_1,x_2)=n\}} ; \forall i, j \in \mathcal{V}, k, m \in \mathcal{ST}, n \in \mathcal{CRD}$$

where $\mathcal{CRD}$ = {many-to-many, 1-to-many, 1-to-1, NULL}.

3.2.3 Properties co-occurrence. This clique template gives a boost to properties that are likely to occur together. For example, birth date and death date of an artist have higher co-occurrence than birth date and creation date of a painting. If we have two dates attributes, the chance of their being birth date and creation date will be lower than the chance that they are birth date and death date.

A co-occurrence frequency matrix is estimated from known semantic models and are used as features in this clique template:

$$f^{OCC}_{ijmn}(y_1, y_2, x_1, x_2) = \mathbf{1}_{\{y_1=i\}}\mathbf{1}_{\{y_2=j\}}\mathbf{1}_{\{st(x_1)=m\}}\mathbf{1}_{\{st(x_2)=n\}} \text{co\_occurrence}(x_1, x_2) ; \forall i, j \in \mathcal{V}, m, n \in \mathcal{ST}$$

$$f^{\overline{OCC}}_{ijk}(y_1, y_2, x_1, x_2) = \mathbf{1}_{\{y_1=i\}}\mathbf{1}_{\{y_2=j\}}\mathbf{1}_{\{st(x_1)=m\}}\mathbf{1}_{\{st(x_2)=n\}} (1 - \text{co\_occurrence}(x_1, x_2)) ; \forall i, j \in \mathcal{V}, m, n \in \mathcal{ST}$$

where x_1 and x_2 are sibling links of a class node.

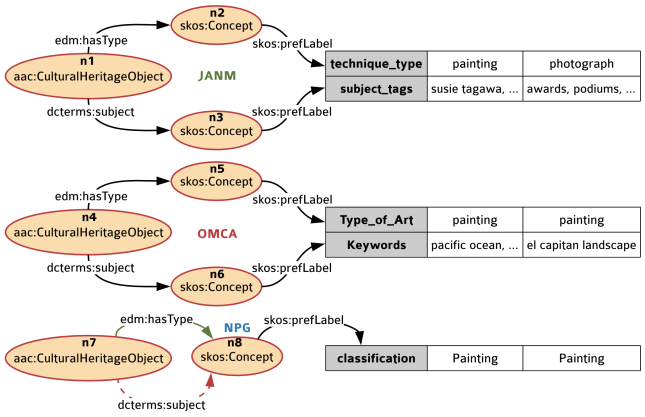


Figure 4: Excerpt from semantic models of the JANM, OMCA[4] and NPG data sources

3.2.4 Duplicated properties. An entity usually has many properties, but a few of them are duplicated. An example of a duplicated property is a class node *Person* with two links *worksFor* to two data nodes *organization1* and *organization2*. As semantic labeling systems often output the same semantic types for similar attributes, knowing which properties could be duplicated helps us identify incorrect links.

Factors in this clique template take a set of same label outgoing links of a class node and have the following features:

$$f^{DP}_{i0}(\boldsymbol{y_t}, \boldsymbol{x_t}) = \mathbf{1}_{\{|\{\forall y_i \in \boldsymbol{y_t} | y_i = true\}|=1\}}\mathbf{1}_{\{st(\boldsymbol{x_t})=i\}}; \forall i \in \mathcal{ST}$$

$$f^{DP}_{i1}(\boldsymbol{y_t}, \boldsymbol{x_t}) = \mathbf{1}_{\{|\{\forall y_i \in \boldsymbol{y_t} | y_i = true\}|>1\}}\mathbf{1}_{\{st(\boldsymbol{x_t})=i\}}; \forall i \in \mathcal{ST}$$

3.2.5 Grouping properties. In hierarchical data sources, properties of an entity are usually grouped under the same nested object. For example, birth date and name of person are properties of the *artist* object and *artist* is in a record object in a data source: {"artist": {"birth_date": "..", "name": ".."}, "title": ".."}.

Let SameScope(x_1, x_2) be a function of two sibling links that output *true* if their target data nodes are under the same nested object and *false* otherwise. Then, the features of this clique template are defined as:

$$f^{GP}_{ij}(y_1, y_2, x_1, x_2) = \mathbf{1}_{\{y_1=i\}}\mathbf{1}_{\{y_2=i\}}\text{SameScope}(x_1, x_2); \forall i, j \in \mathcal{V}$$

$$f^{\overline{GP}}_{ij}(y_1, y_2, x_1, x_2) = \mathbf{1}_{\{y_1=i\}}\mathbf{1}_{\{y_2=i\}}\neg\text{SameScope}(x_1, x_2); \forall i, j \in \mathcal{V}$$

3.2.6 Structural similarity. There are many possible ways to link two class nodes. Class nodes involved in the same relationship often connect to a similar set of data nodes. For example, in Figure 4, *Concept* nodes n_2, n_5 and n_8 link with *CulturalHeritageObject* nodes via the predicate *hasType* have similar values, which are types of artworks. While *Concept* nodes n_3 and n_6 involved in different relations (*dcterms:subject*) contain values about subjects in artworks. This heuristic can be used to predict the likelihood of links between class nodes. In particular, we define factor functions taking two links x_1 and x_2, which are class link and data link, respectively. The input x_1 and x_2 form a substructure. We say two substructures are partially matched if they both have the same link x_2. For example, in Figure 4,

[4]http://www.janm.org/ and http://museumca.org/

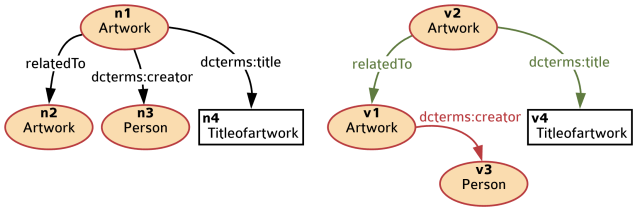


Figure 5: Example of a gold model and predicted model. The gold model is on the left side and the other is the predicted model. The green lines in the predicted model show correct links and red ones are incorrect links.

we have two substructures: $s_1 = \langle CulturalHeritageObject, hasType, Concept, prefLabel, classification\rangle$ and $s_2 = \langle CulturalHeritageObject, subject, Concept, prefLabel, classification\rangle$, in which $s2$ and $s1$ are partially matched.

The likelihood of a substructure is the maximum similarity score between its source attribute with other source attributes of other identical substructures in the training set. The reward of the substructure, Reward(x_1, x_2), is defined to be the difference between the likelihood of current substructure with the highest likelihood of other partial matched substructures. For example, in Figure 4, assuming the similarity score between attribute *classification* and attributes *technique_type*, *subject_tags*, *type_of_art*, *keywords* are 0.7, 0.5, 0.8, 0.2, respectively. Then, the likelihood of substructure $s_1 = max(0.7, 0.8) = 0.8$, and the likelihood of substructure $s_2 = 0.5$. The reward of the substructure $s_1 = 0.8 - 0.8 = 0$ while reward of $s_2 = 0.5 - 0.8 = -0.3$. Hence, our CRF will prefer s_1 to s_2.

Using Reward(x, y) function, the features of this clique template are defined as follow:

$$f_{ij}^{SrS}(y_1, y_2, x_1, x_2) = \mathbf{1}_{\{y_1=i\}}\mathbf{1}_{\{y_2=i\}}\text{Reward}(x_1, x_2); \forall i, j \in \mathcal{V}$$

$$f_{ij}^{\overline{SrS}}(y_1, y_2, x_1, x_2) = \mathbf{1}_{\{y_1=i\}}\mathbf{1}_{\{y_2=i\}} - \text{Reward}(x_1, x_2); \forall i, j \in \mathcal{V}$$

3.2.7 Error propagations. The factors of this clique template are used to remove class nodes in which all of their outgoing links are labeled *false* because they are redundant. In particular, the factors need to output high energy when all links of a class node are incorrect, and low energy when all of the outgoing links have label *false* but the incoming link is labeled *true*. Features of this clique template are defined as follow:

$$f_i^{EP}(y_0, \boldsymbol{y}_c, x_0, \boldsymbol{x}_c) = \mathbf{1}_{\{y_0=i\}}\mathbf{1}_{\{y=\text{false}; \forall y \in \boldsymbol{y}_c\}}; \forall i \in \mathcal{V}$$

where x_0 is the incoming link, $\boldsymbol{x}_c$ are the outgoing links.

3.3 Training the Graphical Model

To train the graphical model, beside positive examples obtained from set of known semantic models, we also need negative examples. A typical approach to generate more training data is collecting generated semantic models from the search procedure. Then, every link in the generated models is labeled *true* or *false* using an automated labeling procedure.

3.3.1 Auto-label examples. Define rel(*sm*) is a set of triples (u, e, v) of semantic model *sm*, in which u, v is a source node and target node of a link e, respectively. The labeling procedure works as follow. First, we find the best mapping from node in the predicted model sm' to node in known model *sm* such that it maximizes the number of overlapped triples between rel(*sm*) and rel(sm'). For example, in Figure 5, if the mapping is $\{n_1 \rightarrow v_2, n_2 \rightarrow v_1, n_3 \rightarrow v_3, n4 \rightarrow v_4\}$, we have 1 overlapping triple $\langle Artwork, dcterms{:}title, Titleofartwork\rangle$. If we switch the mapping between two nodes n_1 and n_2, we have 2 overlapping triples and this is the best mapping we can have. After the alignment step, each link e in predicted model sm' is assigned a true label if its triple (u, e, v) is in rel(*sm*); otherwise, it is assigned *false*.

3.3.2 Generate training examples.

Algorithm 3: EliminateDataNodes

Input: A gold semantic model: *sm*
Output: List of new semantic models in which some data nodes have been removed

```
startStates ← []
sm_new ← remove data nodes in sm, which semantic labeling doesn't predict their semantic types correctly
for u ← sm_new.dataNodes do
    for stype ← PredictedSemanticTypes (u) do
        V_match ← all the data nodes have semantic type stype
        if |V_match | = 0 then
            state ← remove u out of sm_new
            add state to startStates
        for v ← V_match do
            if u is not v then
                state ← remove u and v out of sm_new
                add state to startStates
return startStates
```

To make the process of generating training examples efficient, instead of starting from an empty semantic model, we start from a known semantic model, remove some data nodes, then invoke the search algorithm in Section 3.1 to add back the removed attributes. The algorithm 3 describes how we remove data nodes. The goal is instead of removing random nodes, we remove the nodes that have same semantic types, so that when we add back a data node, there is always more than one possible way we can merge the data node into the model, hence generate a hard example for our graphical model to learn. For example, in Figure 1, suppose that we are removing the attribute *artist*, and *artist* is tagged with two possible semantic types: $\langle CulturalHeritageObject, title\rangle$ and $\langle Person, name\rangle$. After removing, we have two started states: the first one is the graph missing the links $\langle n_1\_CulturalHeritageObject, title\rangle$ and $\langle n_4\_Person, name\rangle$, the second state is omitting the links $\langle n_4\_Person, name\rangle$ and $\langle n_5\_Person, name\rangle$.

Note that as there are many possible examples that can be generated, each example is associated with a score computed using Taheriyan et al. [29] method, then we randomly sample the examples using the score to reduce the number of training examples. In our experiments, the number of examples is set to 300 per gold semantic model.

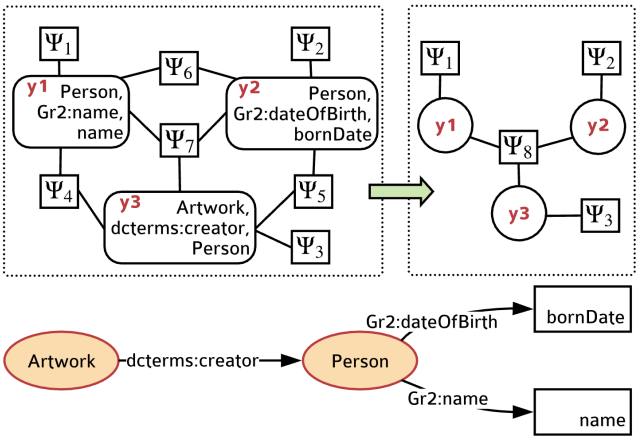


Figure 6: Factor graphs before and after eliminating cycles. In the factor graphs, we omit random variables $\boldsymbol{x}$ because their values are fixed.

3.3.3 Inference and Parameter Learning. We perform exact inference on CRF using Belief Propagation [26]. Undirected graphical models are often represented as factor graphs in which random variables and factor functions are nodes. An edge is drawn between a random variable and a factor if the variable is input of the factor function. Since we can only apply Belief Propagation to acyclic factor graphs, we eliminate cycles in our factor graphs by merging cliques which form cycles into single cliques. With the choices of factors in Section 3.2, we only need to merge factors, whose inputs are incoming links and/or outgoing links of same class nodes. For example, in the factor graph in Figure 6, the graph becomes acyclic after the four factors Ψ_4 to Ψ_7, which form cycles, are merged and replaced by a new factor:

$$\Psi_8(y_1, y_2, y_3, x_1, x_2, x_3) = \Psi_4(y_1, y_3, x_1, x_3)\Psi_5(y_2, y_3, x_2, x_3)$$
$$\Psi_6(y_1, y_2, x_1, x_2)\Psi_7(y_1, y_2, y_3, x_1, x_2, x_3)$$

The problem with this approach is that the merged factors may have many variables, which results in the number of values of $\boldsymbol{y}$ we need to enumerate increasing exponentially with n: 2^n, where $n = |\boldsymbol{y}|$. Fortunately, it is not common to have a class node containing too many properties. In order to ensure the inference algorithm has reasonable runtime, we could control the number of variables in the factor by keeping top $n' < n$ most frequently used predicates. In the experiments, we set $n = 11$ so there are only maximum 2048 combinations of $\boldsymbol{y}$ values.

Our CRF is trained using the standard maximum log-likelihood and Belief Propagation. For more details about Belief Propagation and training CRF, readers can refer to the tutorial in Sutton et al. [26].

4 EXPERIMENTAL EVALUATION

4.1 Dataset and experimental setup

Our objective is to assess the ability of our method to infer correct semantic models with and without noisy data. We evaluate our approach, which is called PGM-SM, on two different datasets: ds_{edm} and ds_{crm} from Taheriyan et al. [29]. Both datasets contain data sources from different art museums in US, and in different data formats (CSV, XML, JSON, etc). However, domain experts use two different well-known data models in the museum domain: Europeana Data Model (EDM), and CIDOC Conceptual Reference Model (CIDOC-CRM) for knowledge representation. The details of two datasets are listed in Table 2. Note that we updated the semantic models in the ground-truth datasets to fix some mistakes based on the recommendation[5] from domain experts [12]. The updated datasets, experimental results and our code are available on Github[6].

Table 2: The evaluation datasets ds_{edm} and ds_{crm}

	ds_{edm}	ds_{crm}
#data sources	28	28
#classes in domain ontologies	119	147
#properties in domain ontologies	351	409
#nodes in the gold-standard models	444	970
#data nodes in the gold-standard models	310	552
#class nodes in the gold-standard models	134	418
#links in the gold-standard models	416	942

Table 3: MRR scores of three semantic labeling methods

Datasets	DSL	Serene	SemTyper
ds_{edm}	0.886	0.912	0.830
ds_{crm}	0.896	0.910	0.628

Semantic labeling methods are evaluated using standard mean reciprocal rank (MRR) [5]. We assess the quality of predicted semantic models by comparing with gold semantic models in term of precision and recall as in Taheriyan et al. [29]:

$$\text{precision} = \frac{|rel(sm) \cap rel(f^*(sm'))|}{|rel(f^*(sm'))|}$$
$$\text{recall} = \frac{|rel(sm) \cap rel(f^*(sm'))|}{|rel(sm)|}$$
$$f^* = \underset{f}{\text{argmax}}\, |rel(sm) \cap rel(f(sm'))|$$

where $rel(sm)$ is a set of triples (u, e, v) of a semantic model sm, f is a mapping function that maps nodes in sm' to nodes in sm (Section 3.3.1).

In our experiments, we use half of the dataset for training and the other half for testing. We repeat the process by three times (3-folds) and average the results. Parameters of our CRF are learned using the ADAM-AMSGRAD optimization method [21] for 60 epoches with batch size 200 and learning rate 0.05. Our experiments are run on a single machine with Intel Xeon E5-2620 v4 and 32GB RAM.

4.2 Automatic semantic modeling

In this experiment, we evaluate our method on four different semantic labelers: SemTyper [20], DSL [18], Serene [22] and Oracle.

[5]The recommendation documentation can be found at: http://review.americanartcollaborative.org/
[6]https://github.com/binh-vu/semantic-modeling

Table 4: Performances of semantic modeling systems with respect to different semantic labelers

Datasets	Labelers	Precision			Recall			F1		
		Taheriyan	Serene	PGM-SM	Taheriyan	Serene	PGM-SM	Taheriyan	Serene	PGM-SM
ds_{edm}	SemTyper	0.726	0.689	0.772	0.702	0.701	0.767	0.712	0.693	0.768
	DSL	0.656	0.708	0.812	0.618	0.734	0.820	0.635	0.719	0.815
	Serene	0.808	0.777	0.822	0.800	0.803	0.837	0.803	0.789	0.829
	Oracle	0.883	0.876	0.945	0.892	0.896	0.927	0.887	0.885	0.935
ds_{crm}	SemTyper	0.695	0.710	0.809	0.559	0.624	0.660	0.618	0.663	0.725
	DSL	0.699	0.714	0.851	0.693	0.687	0.839	0.695	0.698	0.844
	Serene	0.779	0.736	0.886	0.770	0.773	0.875	0.774	0.753	0.880
	Oracle	0.869	0.838	0.965	0.847	0.846	0.928	0.857	0.840	0.944

Table 5: Average F_1 score of PGM-SM with respect to different training size

Datasets	25% of dataset	50% of dataset
ds_{edm}	0.808	0.837
ds_{crm}	0.823	0.848

Table 6: Average running time of PGM-SM on two different datasets.

Datasets	Training time	Testing time
ds_{edm}	162.47s	9.54s
ds_{crm}	315.74s	59.37s

Each semantic labeler has different performance and characteristic, thus provides different levels of noise to our system. Of the three methods, SemTyper has the lowest MRR score indicating that its output includes many incorrect semantic types. While DSL has higher MRR score, thus provides fewer incorrect semantic types. However, the confidence score between predicted semantic types is similar since DSL predicts semantic labels based on a similarity notion. Serene has the best MRR performance, and the confidence score between correct and incorrect semantic types is very different. The Oracle semantic labeler is a labeler that always outputs the correct semantic type, thus provides no noise to the semantic modeling system. The performance of these semantic labelers is reported in Table 3.

We compare our method with two state-of-the-art semantic modeling systems: Taheriyan et al. [29] and Uña et al. [30] (Serene). The experiment results are reported in Table 4. Generally, the F_1 score increases with less noise for three systems. Among them, our approach outperforms the two baseline methods with the average increase by 6.53% and 10.92% on ds_{edm} and ds_{crm}, respectively. As both baseline methods predict semantic models by finding the minimum-weighted tree, they would be affected if the weights of edges are similar. Therefore, they are more sensitive to the noise of DSL. Our approach, however, uses the collective signals and achieves significant improvement, which indicates it is more robust to noise.

Table 5 shows the average F_1 score of PGM-SM when the number of known semantic models is 25% and 50% of the datasets. As the training size increases, the performance of PGM-SM increases. The average percentage of performance gain is 2.7%, which suggests that the model achieves reasonable performance even when the training size is small (25%). Comparing to the two baseline methods Taheriyan et al. [29] and Serene in the same scenario, PGM-SM improves the F_1 score on average 7.68% and 9.20%, respectively.

In addition, we measure the running time of our approach in terms of training and testing time listed in Table 6. The training time starts from generating training examples to finishing training PGM-SM. The testing time is total time to predict semantic models of the testing data sources. Since the size of semantic models in ds_{crm} are twice as big as the models in ds_{edm} (Table 2), we can see that the training time of ds_{crm} is nearly double ds_{edm}. The testing time does not increase correspondingly because it depends not only on the runtime of PGM but also the runtime of the search algorithm, which increases linearly with the number of attributes in the data sources.

4.3 Feature analysis

To understand the impact of each feature on the result, we perform ablation analysis by removing the factors described in Section 3.2. As the results in Table 7 have shown, dropping some factors sometimes slightly increases the F_1 score. In particular, STRUCT-SIM and ERROR-PROP are important in dataset ds_{crm}, while they are not in dataset ds_{edm}. The reason of this phenomenon is that the semantic models in ds_{crm} are very complex and structural while semantic models in ds_{edm} are not. For example, in ds_{edm}, a creation date of an artwork is modeled as a literal date time value. While in ds_{crm}, the date is represented as a class *E52_Time-Span* to capture the fact that it could take an artist many years to create an artwork. As those factors have little effect in ds_{edm}, dropping them actually makes the CRF model to learn better parameters of the remaining factors. On the other hand, CO-OCCUR and GROUPING factors

Table 7: Ablation analysis of the factors in PGM-SM. Each cell value is a difference in F_1 score upon dropping one factor.

Factor	Factor name	ds_{edm}				ds_{crm}			
		SemTyper	DSL	Serene	Oracle	SemTyper	DSL	Serene	Oracle
Structural Similarity	STRUCT-SIM	-0.009	0.004	0.020	0.008	-0.006	-0.013	-0.001	-0.015
Error Propagation	ERROR-PROP	0.005	0.005	0.004	0.000	-0.006	-0.016	-0.001	0.003
Co-occurrence	CO-OCCUR	-0.006	-0.014	-0.008	-0.005	0.024	-0.008	0.010	0.004
Grouping properties	GROUPING	0.008	-0.011	-0.003	0.008	-0.007	-0.012	0.004	0.003
Duplicated properties	DUP-PROP	-0.003	-0.009	-0.005	0.000	-0.007	-0.008	-0.003	-0.007
Cardinality relationships	CARDINALITY	-0.006	-0.020	-0.013	-0.023	-0.014	-0.049	-0.022	-0.032

show the opposite trend. DUP-PROP and CARDINALITY have positive impact on the F_1 score for all datasets and CARDINALTIY is the most important feature in our CRF model.

5 RELATED WORK

There are many studies to automatically integrate data from multiple sources. Although their goals are similar, they are different in terms of inputs and assumptions and can be clustered into three groups.

The first group focuses on problems of schema matching and mapping [3]. Schema matching generates correspondences between elements of two schemas [4, 6, 19]. For instance, Madhavan et al. [15] combine information from schema constraints (data types, foreign keys), structure and values in data sources to find one to one correspondence between elements in source schema and target schema. Schema mapping generates mapping formulas from schema matches to transform data from source schema to target schema [1, 2, 8, 9, 16, 23]. In some recent studies, Kimmig et al. [10] use probabilistic soft logic to explore information from both data examples and schema constraints to generate the mappings. The semantic modeling problem, however, is different from schema matching and mapping [29]. First, in addition to finding mappings from attributes in a data source to ontology classes and properties, we also generate relationships between the attributes. Moreover, as data sources on the Web are often lack of schema constraints and have different conventions, semantic modeling provides a straightforward approach to represent and publish the data to knowledge graph.

The second group focuses on describing semantics of web tables. Limaye et al. [14] use a probabilistic graphical model and an external knowledge base YAGO to annotate tables with entities for cells, semantic types for columns and binary relationships between columns. Mulwad et al. [17] share similar ideas with Limaye et al., in which they leverage Linked Open Data (LOD) such as DBpedia, YAGO and Wikitology to link table cells with LOD entities. After generating initial lists of candidates for cell values, they perform inference to compute the assigned types of column, cell linkage and relations between columns. Instead of using available knowledge graphs, Venetis et al. [31] build two databases of class of entities and relationships between the entities from text documents available on the Web. The column labels are assigned to maximize the probability of column values given class labels of entities. All of these approaches rely on external knowledge bases which provide possible links between cell values and known entities. Hence, performance of these approaches highly depends on the quality of the linkages. Furthermore, as not all data attributes have corresponding entities in knowledge base, e.g: birth date, phone, age; these approaches may not be applicable to a broad range of data sources and be restricted to domains where online data is widely available.

Our problem is in the final group, in which semantic models are used to precisely describe meanings of data sources and publish data to knowledge graph. The semantic models can represent a wide range of data sources and in various formats such as CSV, XML, JSON, etc. To address the semantic modeling problem, Taheriyan et al. [28, 29] build an integration graph that represents space of plausible semantic models for a new data source. Each link in the integration graph is associated with a weight representing the frequency of use of the link in previous models. First, attributes of a new data source are associated with data nodes in the integration graph. Then, the semantic model is built by finding the Steiner Tree that connects the data nodes. Uña et al. [30] also predict semantic models by finding Steiner Tree. However, to solve the Steiner Tree problem, instead of using an approximation algorithm as in Taheriyan et al. [29], they use constraint programming, an exact algorithm, which allows it to incorporate more features as additional constraints. Instead of solving the Steiner Tree problem, our approach uses a PGM to exploit collective signals from a data source. The PGM allows us to express complex dependencies between the features of good and bad semantic models. Therefore, we are able to obtain better performance than the previous state-of-the-art methods.

6 CONCLUSION AND FUTURE WORK

In this paper, we presented a novel approach for semantic modeling that uses a probabilistic graphical model to exploit relationship within data sources and semantic models. From a set of data sources and their semantic models, we train a conditional random field (CRF) to distinguish between good and bad semantic models. Then, we use the CRF to efficiently search over the combinatorial space of possible semantic models to identify the most probable model for a data source. The evaluation shows that by exploring the relationships within data, our approach generates better semantic models and is more robust to noise than previous approaches.

Because labeled data is costly to obtain, a future direction of our work is to exploit transfer learning to leverage a vast amount of available linked datasets in the Linked Open Data. For example, we can train a CRF on linked datasets, which are represented using the same domain ontologies, and reuse them as a pre-trained model. We can also enrich our training set by applying ontology matching techniques to align linked data from different ontologies to our ontologies.

Another direction for future work is to integrate semantic modeling with web data extraction. Unsupervised web data extraction methods such as Trinity [24] can extract structural data from a set of web pages. However, the output of these methods often includes many redundant data attributes. By leveraging more information from the extraction step such as the position of extracted attributes in the web pages and collective information between data attributes, we can potentially improve performance of systems that automatically extract and semantically annotate web data.

ACKNOWLEDGMENTS

This material is based upon the work supported by the Defense Advanced Research Projects Agency (DARPA) under award W911NF-18-1-0027. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of DARPA. We thank the anonymous reviewers for their helpful comments on the paper.

REFERENCES

[1] Bogdan Alexe, Balder Ten Cate, Phokion G Kolaitis, and Wang-Chiew Tan. 2011. Designing and refining schema mappings via data examples. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*. ACM, 133–144.

[2] David Aumueller, Hong-Hai Do, Sabine Massmann, and Erhard Rahm. 2005. Schema and ontology matching with COMA++. In *Proceedings of the 2005 ACM SIGMOD international conference on Management of data*. Acm, 906–908.

[3] Zohra Bellahsène, Angela Bonifati, and Erhard Rahm. 2011. *Schema matching and mapping*. Springer.

[4] Philip A Bernstein, Jayant Madhavan, and Erhard Rahm. 2011. Generic schema matching, ten years later. *Proceedings of the VLDB Endowment* 4, 11 (2011), 695–701.

[5] Nick Craswell. 2009. Mean reciprocal rank. In *Encyclopedia of Database Systems*. Springer, 1703–1703.

[6] Robin Dhamankar, Yoonkyong Lee, AnHai Doan, Alon Halevy, and Pedro Domingos. 2004. iMAP: discovering complex semantic matches between database schemas. In *Proceedings of the 2004 ACM SIGMOD international conference on Management of data*. ACM, 383–394.

[7] Anastasia Dimou, Miel Vander Sande, Jason Slepicka, Pedro Szekely, Erik Mannens, Craig A. Knoblock, and Rik Van de Walle. 2014. Mapping hierarchical sources into RDF using the RML mapping language. In *IEEE International Conference on Semantic Computing (ICSC)*. IEEE, 151–158.

[8] Ronald Fagin, Laura M Haas, Mauricio Hernández, Renée J Miller, Lucian Popa, and Yannis Velegrakis. 2009. Clio: Schema mapping creation and data exchange. In *Conceptual modeling: foundations and applications*. Springer, 198–236.

[9] Ronald Fagin, Phokion G Kolaitis, Renée J Miller, and Lucian Popa. 2005. Data exchange: semantics and query answering. *Theoretical Computer Science* 336, 1 (2005), 89–124.

[10] Angelika Kimmig, Alexander Memory, Renee J Miller, and Lise Getoor. 2018. A Collective, Probabilistic Approach to Schema Mapping Using Diverse Noisy Evidence. *IEEE Transactions on Knowledge and Data Engineering* (2018).

[11] Craig A. Knoblock, Pedro Szekely, Jose Luis Ambite, Shubham Gupta, Aman Goel, Maria Muslea, Kristina Lerman, Mohsen Taheriyan, and Parag Mallick. 2012. Semi-Automatically Mapping Structured Sources into the Semantic Web. In *Proceedings of the Extended Semantic Web Conference*. Crete, Greece.

[12] Craig A. Knoblock, Pedro Szekely, Eleanor Fink, David Newbury Duane Degler, Robert Sanderson, Kate Blanch, Sara Snyder, Nilay Chheda, Nimesh Jain, Ravi Raju Krishna, Nikhila Begur Sreekanth, and Yixiang Yao. 2017. Lessons Learned in Building Linked Data for the American Art Collaborative. In *ISWC 2017 - 16th International Semantic Web Conference*.

[13] Daphne Koller and Nir Friedman. 2009. *Probabilistic Graphical Models: Principles and Techniques*.

[14] Girija Limaye, Sunita Sarawagi, and Soumen Chakrabarti. 2010. Annotating and Searching Web Tables Using Entities, Types and Relationships. *Proc. VLDB Endow.* 3, 1-2 (Sept. 2010), 1338–1347. https://doi.org/10.14778/1920841.1921005

[15] Jayant Madhavan, Philip A Bernstein, and Erhard Rahm. 2001. Generic schema matching with cupid. In *vldb*, Vol. 1. 49–58.

[16] Bruno Marnette, Giansalvatore Mecca, Paolo Papotti, Salvatore Raunich, Donatello Santoro, et al. 2011. ++ Spicy: an Open-Source Tool for Second-Generation Schema Mapping and Data Exchange. *Clio* 19 (2011), 21.

[17] Varish Mulwad, Tim Finin, and Anupam Joshi. 2013. Semantic message passing for generating linked data from tables. In *International Semantic Web Conference*. Springer, 363–378.

[18] Minh Pham, Suresh Alse, Craig Knoblock, and Pedro Szekely. 2016. Semantic labeling: A domain-independent approach. In *ISWC 2016 - 15th International Semantic Web Conference*.

[19] Erhard Rahm and Philip A Bernstein. 2001. A survey of approaches to automatic schema matching. *the VLDB Journal* 10, 4 (2001), 334–350.

[20] S.K. Ramnandan, Amol Mittal, Craig A. Knoblock, and Pedro Szekely. 2015. Assigning Semantic Labels to Data Sources. In *Proceedings of the 12th ESWC 2015*.

[21] Sashank J. Reddi, Satyen Kale, and Sanjiv Kumar. 2018. On the Convergence of Adam and Beyond. In *ICLR 2018 : International Conference on Learning Representations 2018*.

[22] Natalia Rümmele, Yuriy Tyshetskiy, and Alex Collins. 2018. Evaluating approaches for supervised semantic labeling. (2018).

[23] Satya S. Sahoo, Juan Sequeda, and Ahmed Ezzat. 2009. A Survey of Current Approaches for Mapping of Relational Databases to RDF. (2009).

[24] Hassan A Sleiman and Rafael Corchuelo. 2014. Trinity: on using trinary trees for unsupervised web data extraction. *IEEE Transactions on Knowledge and Data Engineering* 26, 6 (2014), 1544–1556.

[25] Jason Slepicka, Chengye Yin, Pedro Szekely, and Craig A. Knoblock. 2015. KR2RML: An Alternative Interpretation of R2RML for Heterogenous Sources. In *Proceedings of the 6th International Workshop on Consuming Linked Data (COLD 2015)*.

[26] Charles A. Sutton and Andrew McCallum. 2012. An Introduction to Conditional Random Fields. *arXiv preprint arXiv:1011.4088* 4, 4 (2012), 267–373.

[27] Pedro Szekely, Craig A. Knoblock, Jason Slepicka, Andrew Philpot, Amandeep Singh, Chengye Yin, Dipsy Kapoor, Prem Natarajan, Daniel Marcu, Kevin Knight, David Stallard, Subessware S. Karunamoorthy, Rajagopal Bojanapalli, Steven Minton, Brian Amanatullah, Todd Hughes, Mike Tamayo, David Flynt, Rachel Artiss, Shih-Fu Chang, Tao Chen, Gerald Hiebel, and Lidia Ferreira. 2015. Building and Using a Knowledge Graph to Combat Human Trafficking. In *Proceedings of the 14th International Semantic Web Conference (ISWC 2015)*.

[28] Mohsen Taheriyan, Craig Knoblock, Pedro Szekely, and Jose Luis Ambite. 2016. Leveraging Linked Data to Discover Semantic Relations within Data Sources. In *ISWC 2016 - 15th International Semantic Web Conference*.

[29] Mohsen Taheriyan, Craig A. Knoblock, Pedro Szekely, and Jose Luis Ambite. 2016. Learning the semantics of structured data sources. *Journal of Web Semantics* (2016).

[30] Diego De Uña, Nataliia Rümmele, Graeme Gange, Peter Schachte, Peter J. Stuckey, and Peter J. Stuckey. 2018. Machine Learning and Constraint Programming for Relational-To-Ontology Schema Mapping. In *IJCAI 2018: 27th International Joint Conference on Artificial Intelligence*. 1277–1283.

[31] Petros Venetis, Alon Halevy, Jayant Madhavan, Marius Paşca, Warren Shen, Fei Wu, Gengxin Miao, and Chung Wu. 2011. Recovering semantics of tables on the web. *Proceedings of the VLDB Endowment* 4, 9 (2011), 528–538.