Integrating unstructured data into relational databases

Imran R. Mansuri
imran@it.iitb.ac.in
IIT Bombay

Sunita Sarawagi
sunita@it.iitb.ac.in
IIT Bombay

Abstract

In this paper we present a system for automatically integrating unstructured text into a multi-relational database using state-of-the-art statistical models for structure extraction and matching. We show how to extend current high-performing models, Conditional Random Fields and their semi-markov counterparts, to effectively exploit a variety of recognition clues available in a database of entities, thereby significantly reducing the dependence on manually labeled training data. Our system is designed to load unstructured records into columns spread across multiple tables in the database while resolving the relationship of the extracted text with existing column values, and preserving the cardinality and link constraints of the database. We show how to combine the inference algorithms of statistical models with the database imposed constraints for optimal data integration.

1 Introduction

Database systems are islands of structure in a sea of unstructured data sources. Several real world applications now need to create bridges for smooth integration of semi-structured sources with existing structured databases for seamless querying. Although a lot of research has gone in the NLP [4, 14], machine learning [8, 18] and web mining community [9] on extracting structured data from unstructured sources, most of the proposed methods depend on tediously labeled unstructured data. The database is at best treated as a store for the structured data.

In this paper we attempt to bridge this gap by more centrally exploiting existing large databases of the multi-relational entities to aid the extraction process on one hand, and on the other hand, augmenting the database itself with the extracted information from unstructured text in a way that allows multiple noisy variants of entities to co-exist and aid future extraction tasks.

1.1 Applications

There are a number of applications where unstructured data needs to be integrated with structured databases on an ongoing basis so that at the time of extraction a large database is available.

Consider the example of publications portals like Citeseer and Google Scholar. When integrating publication data from personal homepages, it does not make sense to perform the extraction task in isolation. Structured databases from, say ACM digital library or DBLP are readily available and should be exploited for better integration.

Another example is resume databases in HR departments of large organizations. Resumes are often stored with several structured fields like experience, education and references. When a new resume arrives in unstructured format, for example, via email text, we might wish to integrate it into the existing database and the extraction task can be made easier by consulting the database.

Another interesting example is from personal information management (PIM) systems where the goal is to organize personal data like documents, emails, projects and people in a structured inter-linked format. The success of such systems will depend on being able to automatically extract structure from the existing predominantly file-based unstructured sources. Thus, for example we should be able to automatically extract from a PowerPoint file, the author of a talk and link the person to the presenter of a talk announced in an email. Again, this is a scenario where we will already have plenty of existing structured data into which we have to integrate a new unstructured file.

1.2 Challenges and Contribution

A number of challenges need to be addressed in designing an effective solution to the data integration task. Useful clues are scattered in various ways across the structured database and the unstructured text records. Exploiting them effectively is a challenge.

First, the existing structured databases of entities are organized very differently from labeled unstructured text.

Most prior work has been on extracting information from unstructured text which can be modeled as a sequence of tokens. The input labeled data is thus a collection of token sequences. In contrast, the database is a graph of entity relationships and there is no information in the database about inter-entity sequencing. A good extraction system should exploit both the inter-entity sequencing information from the labeled unstructured text and the semantic relationship between entities in the database.

Second, there is significant format variation in the names of entities in the database and the unstructured text. The entities in a structured database provide useful pattern-level information that could help in recognizing entities in the unstructured text.

Finally, in most cases the database will be large whereas labeled text data will be small. Features designed from the databases should be efficient to apply and should not dominate features that capture contextual words and positional information from the limited labeled data.

We design a data integration system that builds upon state-of-the-art semi-Markov models [7, 16] for information extraction to exploit useful information in both structured data and labeled unstructured data in spite of their format, structure and size variations. Our experiments show that our proposed model significantly improves the accuracy of data integration tasks by exploiting databases more effectively than has been done before.

Outline In Section 2, we describe our problem setup and present the overall architecture of the system. In Section 3, we present background on statistical models for extraction and matching. In Section 4 we show how we augment these models to exploit features from the database, and integrate data so as to meet the cardinality and link constraints in the database. In Section 5, we present integration performance on real-life datasets and present related work in Section 6.

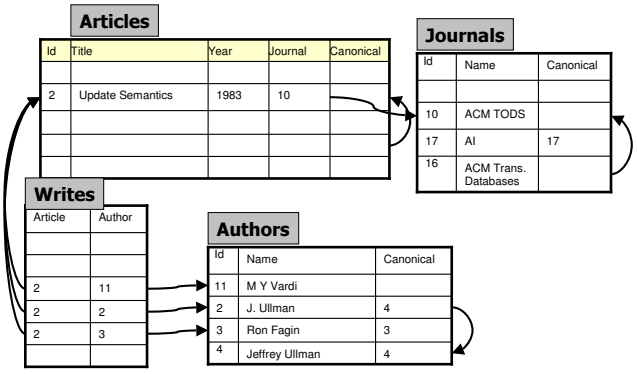


Figure 1. A sample normalized database for storing journal bibitems

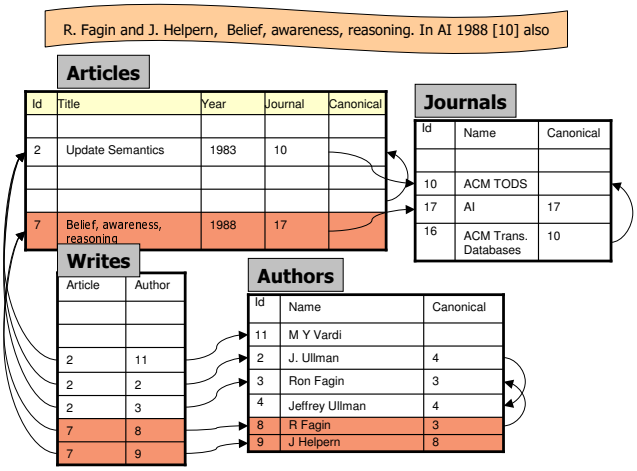


Figure 2. Integrating an unstructured citation into the database of Figure 1

2 Setup and Architecture

2.1 Relational data representation

A relational database represents a normalized set of entities with the relationships between them expressed through foreign keys. An example of such a multi-relational database is shown in Figure 1 representing bibtex entries for journal articles. The database consists of three different entities: articles, journals and authors. The foreign keys from the *article* to the *journal* tables shows how journals name have been normalized across different articles. The author name is a set-valued attribute of articles and this relationship is stored in the *writes* table.

In real-life the same physical entity is often represented by multiple names that can be thought of as noisy variants of each other. The database allows these variants to co-exist and **canonical** links are placed from a noisy variant of an entry to its canonical form. In the sample database of Figure 1, we show such canonical links between two journal entries "ACM TODS" and "ACM Trans. Databases" and between entries "Jeffrey Ullman" and "J. Ullman" in the *authors* table. Each variant of an entity has a unique identifier and thus the canonical link is a foreign key to this unique identifier. These foreign keys can be within the same table as in the examples in Figure 1 or they could be across tables. This data representation allows a user to preserve an entity in a format that is different from its canonical form. For example, a user might want author names in a citation to be abbreviated although the canonical entry might contain the full name.

In this paper we do not address the issue of which entry to pick as the canonical entry or how to construct a merged

canonical entry. These can be handled as part of follow-up operations once the canonical links are created.

A bit of notation to summarize the above: The database consists of a set of entities (each of which map to a database table) $E_1 \ldots E_e$ where each entity has a set of attributes or column values (these exclude columns like primary or foreign keys). Let $A_1 \ldots A_m$ refer to the union of all attributes over all entities. In Figure 1, m is 4 and includes title, year, journal name and author name. Each entity in the database has a unique identifier id and optionally a canonical identifier c that is a foreign key to the id of another entity of which it is assumed to be a noisy variant.

2.2 Labeled unstructured text

The unstructured data is a text record representing a single top-level entity like articles, but embedded in a background of irrelevant text. A labeled set L is provided by someone manually labeling portions of the text with the name of the entity to which it belongs. We will show in the experimental section that the size of L can be very small (5 to 10).

2.3 The task

We start with an existing multi-entity, multi-variant database D and a small labeled set L. We use this to train models for information extraction and matching so that given an unstructured text string $\mathbf{x}$, we can perform the following:

- Extract attributes from $\mathbf{x}$ corresponding to all attributes in the entity database. We call this the information extraction problem.
- Map the extracted entities to existing entries in the database if they match, otherwise, create new entries. Assign relationships through foreign keys when attributes span multiple linked tables. We call this the matching problem.

As a result of these operations the entry $\mathbf{x}$ is integrated across the multiple tables of the database with the canonical links resolved correctly. In actual deployment a user might also wish to inspect the extracted entries and correct for any errors of the automated integration step.

We continue with our sample database of Figure 1 and show how to integrate into it a noisy unstructured citation "R. Fagin and J. Helpern, Belief, awareness and reasoning. In AI 1988 [10] also" (see in Figure 2). We extracted from this two authors, a title, a journal name and a year field and integrated them with the database as shown in Figure 2. On matching with the database, we find the journal "AI" to be already present and the author "R. Fagin" to be a variant of an existing author name "Ron Fagin". This leads to the canonical link from the new entry and the existing author name.

2.4 Available clues

The database D with its entities and canonical links, and the labeled data L can be exploited in a variety of ways for learning how to integrate an incoming unstructured entry into the database. We go over these first and later show how these are exploited by the statistical models.

2.4.1 Clues in the database

As a database gets large, in most cases the entity to be extracted from unstructured text will exist in the database. We therefore need models that can effectively exploit this valuable piece of information. The entities in the database could be useful even when unstructured text does not match a database entity. Different entity attributes have very distinctive patterns that can be generalized to recognize these entities in unstructured text. For example, by inspecting entries in a publications database we can detect that author names often follow a pattern of the form "[A-Z]. [A-Z]. [A-Z][a-z]+".

The schema of the database provides information on the number of entities that are allowed in each record. In Figure 1, the table structure makes it clear that an article can have at most one year, journal and title field but it can have multiple author fields. Also, a foreign key in the database provides information on what entity is allowed to go with what. For example, if we label a text segment as a city name and the segment matches the entry "San Francisco" in the cities column, then any text segment that is labeled as a "state" needs to match entry "CA". In data integration tasks, respecting cardinality and link constraints could be a requirement even if it does not improve accuracy.

2.4.2 Information in the labeled unstructured data

Labeled unstructured data is the classical source of information for extraction model. Traditionally, three kinds of information have been derived from unstructured records. First, the labeled entities hold the same kind of pattern information as the database of entities as discussed above. Second, the context in which an entity appears is a valuable clue that is present solely in the labeled unstructured data. Context is typically captured via few words before or after the labeled entity. For example, the word "In" often appears before a journal name. Finally, the likely ordering of entity labels within a sequence is also useful. For example, author names usually appear before or after a title.

3 Background

In this section we review state-of-the-art statistical models for extraction and matching. These models are designed to exploit clues only in labeled unstructured data and in Section 4 we will discuss how to extend these to include clues from the database.

The problem of extracting structured entities from unstructured data is an extensively researched topic. A number of models have been proposed ranging from the earliest rule-learning models to probabilistic approaches based on generative models like HMMs [17, 3] and conditional models like maxent taggers [14]. Almost all the probabilistic models treat extraction as a sequence labeling problem. The input unstructured text is treated as a sequence of tokens $\mathbf{x} = x_1 \dots x_n$ which needs to be assigned a corresponding sequence of labels $\mathbf{y} = y_1 \dots y_n$ from the set of labels $A = \{A_1 \dots A_m\}$. A promising recently proposed model for information extraction is Conditional Random Fields (CRFs). CRFs have been shown [8, 18] to out-perform previous generative models like Hidden Markov Models and local conditional models like maxent taggers [14].

3.1 Conditional Random Fields

CRFs [8] model a conditional probability distribution over label sequence $\mathbf{y}$ given the input sequence $\mathbf{x}$. The label assigned to the i-th word can be a function of any property of the i-th word or the property of the word defining its context and the label of the word before it. These properties are represented as a vector of feature functions $\mathbf{f} = f^1 \dots f^K$ where each f^k is a scalar function of the form $f(y_i, y_{i-1}, \mathbf{x}, i) \mapsto \mathbf{R}$. Examples of such features are:

$$\begin{aligned} f^7(y_i, y_{i-1}, \mathbf{x}, i) &= [\![x_i \text{ is capitalized}]\!] \\ &\quad \cdot [\![y_i = \text{Author\_name}]\!] \\ f^{11}(y_i, y_{i-1}, \mathbf{x}, i) &= [\![y_i \text{ is journal}]\!] \cdot [\![y_{i-1} = \text{title}]\!] \end{aligned}$$

where the indicator function $[\![c]\!] = 1$ if c if true and zero otherwise;

The CRF model associates a weight to each of these features, thus there is a weight vector $\mathbf{W} = W_1 \dots W_K$ corresponding to the feature vector. These are used to design a global conditional model that assigns a probability for each possible label sequence given (the features of) the input $\mathbf{x}$ sequence as follows:

$$\Pr(\mathbf{y}|\mathbf{x}) = \frac{1}{Z(\mathbf{x})} e^{\mathbf{W} \cdot \mathbf{F}(\mathbf{x}, \mathbf{y})} \tag{1}$$

where $\mathbf{F}(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^{|\mathbf{x}|} \mathbf{f}(y_i, y_{i-1}, \mathbf{x}, i)$ and $Z(\mathbf{x})$ is a normalizing factor equal to $\sum_{\mathbf{y}'} e^{\mathbf{W} \cdot \mathbf{F}(\mathbf{x}, \mathbf{y}')}$.

When deployed for extraction of entities from a text sequence $\mathbf{x}$, CRFs find the best label sequence as:

$$\begin{aligned} \arg\max_{\mathbf{y}} \Pr(\mathbf{y}|\mathbf{x}) &= \arg\max_{\mathbf{y}} \mathbf{W} \cdot \mathbf{F}(\mathbf{x}, \mathbf{y}) \\ &= \arg\max_{\mathbf{y}} \mathbf{W} \cdot \sum_j \mathbf{f}(y_j, y_{j-1}, \mathbf{x}, j) \end{aligned}$$

An efficient inference algorithm is possible because all features depend on just the previous label. Let $\mathbf{y}_{i:y}$ denote the set of all partial labels starting from 1 (the first index of the sequence) to i, such that the i-th label is y. Let $V(i, y)$ denote the largest value of $\mathbf{W} \cdot \mathbf{F}(\mathbf{x}, \mathbf{y}')$ for any $\mathbf{y}' \in \mathbf{y}_{i:y}$. The following recursive calculation (called the Viterbi algorithm) finds the best sequence:

$$V(i, y) = \begin{cases} \max_{y'} V(i-1, y') + \mathbf{W} \cdot \mathbf{f}(y, y', \mathbf{x}, i) & \text{if } i > 0 \\ 0 & \text{if } i = 0 \end{cases} \tag{2}$$

The best label sequence corresponds to the path traced by $\max_y V(|\mathbf{x}|, y)$. This algorithm is linear in the length of $\mathbf{x}$.

Learning is performed by setting parameters $\mathbf{W}$ to maximize the likelihood of the training set $L = \{(\mathbf{x}_\ell, \mathbf{y}_\ell)\}_{\ell=1}^N$ expressed in logarithmic terms as

$$\mathcal{T}(\mathbf{W}) = \sum_\ell \log \Pr(\mathbf{y}_\ell|\mathbf{x}_\ell, \mathbf{W}) \tag{3}$$

$$= \sum_\ell (\mathbf{W} \cdot \mathbf{F}(\mathbf{x}_\ell, \mathbf{y}_\ell) - \log Z_{\mathbf{W}}(\mathbf{x}_\ell)) \tag{4}$$

We wish to find a $\mathbf{W}$ that maximizes $\mathcal{T}(\mathbf{W})$. The above equation is convex and can thus be maximized by gradient ascent or one of many related methods. (In our implementation, we have used a limited-memory quasi-Newton method[10, 11].) More computational details of training can be found in [18]. The training algorithm makes multiple linear scans through the data sequences and for each sequence computes the value of the likelihood function and its gradient in two scans over the features of each sequence.

3.2 Semi-Markov CRFs (Semi-CRFs)

Semi-CRFs [16, 7] extend CRFs so that features can be defined over multi-word segments of text instead of individual words. This provides for a more natural encoding of the information extraction problem, which is now defined as segmenting a text sequence $\mathbf{x}$ such that each segment corresponds to a whole entity. In CRFs, all contiguous words which get assigned the same label implicitly represent an entity. This is not very intuitive since entities mostly consist of multiple words and higher accuracy is achieved if features could be defined over the entire proposed entity as shown in [16, 7].

More formally, a segmentation $\mathbf{s}$ of an input sequence $\mathbf{x}$ is a sequence of segments $s_1 \dots s_p$ such that the last segment ends at n, the first segment starts at 1, and segment

s_{j+1} begins right after segment s_j ends. Each segment s_j consists of a *start position* t_j, an *end position* u_j, and a *label* $y_j \in Y$. A Semi-CRF models the conditional probability distribution over segmentation $\mathbf{s}$ for a given input sequence $\mathbf{x}$ as follows:

$$\Pr(\mathbf{s}|\mathbf{x}, \mathbf{W}) = \frac{1}{Z(\mathbf{x})} e^{\mathbf{W}\cdot\mathbf{F}(\mathbf{x},\mathbf{s})} \quad (5)$$

where again $\mathbf{W}$ is a weight vector for $\mathbf{F}$ and $Z(\mathbf{x}) = \sum_{\mathbf{s}'} e^{\mathbf{W}\cdot\mathbf{F}(\mathbf{x},\mathbf{s}')}$ and $\mathbf{F}(\mathbf{x},\mathbf{s}) = \sum_{j=1}^{|\mathbf{s}|} \mathbf{f}(j,\mathbf{x},\mathbf{s})$.

Like in the case of normal CRFs, the label of a segment depends on the label of the previous segment and the properties of the tokens comprising this segment. Thus a feature for segment $s_j = (y_j, t_j, u_j)$ is of the form $f(y_j, y_{j-1}, \mathbf{x}, t_j, u_j)$. An example of such features for a segment starting at position 3 and ending at 5 is:

$$f^8(y_i, y_{i-1}, \mathbf{x}, 3, 5) = [\![x_3x_4x_5 \text{ appears in a journal list}]\!] \cdot [\![y_i = \text{journal}]\!]$$

During *inference*, the goal is to find a segmentation $\mathbf{s} = s_1 \ldots s_p$ of the input sequence $\mathbf{x} = x_1 \ldots x_n$ such that $\Pr(\mathbf{s}|\mathbf{x}, \mathbf{W})$ (as defined by Equation 5) is maximized.

$$argmax_{\mathbf{s}} \Pr(\mathbf{s}|\mathbf{x}, \mathbf{W}) = argmax_{\mathbf{s}} \mathbf{W} \cdot \sum_j \mathbf{f}(y_j, y_{j-1}, \mathbf{x}, t_j, u_j)$$

Let U be an upper bound on segment length. Let $\mathbf{s}_{i:y}$ denote set of all partial segmentation starting from 1 (the first index of the sequence) to i, such that the last segment has the label y and ending position i. Let $V_{\mathbf{x},\mathbf{f},\mathbf{W}}(i, y)$ denote the largest value of $\mathbf{W} \cdot \mathbf{F}(\mathbf{x}, \mathbf{s}')$ for any $\mathbf{s}' \in \mathbf{s}_{i:y}$. Omitting the subscripts, the following recursive calculation implements a semi-Markov analog of the usual Viterbi algorithm:

$$V(i,y) = \begin{cases} \max_{y', i'=i-U\ldots i-1} V(i', y') & \\ \quad + \mathbf{W} \cdot \mathbf{f}(y, y', \mathbf{x}, i'+1, i) & \text{if } i > 0 \\ 0 & \text{if } i = 0 \\ -\infty & \text{if } i < 0 \end{cases} \quad (6)$$

The best segmentation then corresponds to the path traced by $\max_y V(|\mathbf{x}|, y)$. The running time of semi-Markov Viterbi (Equation 6) is just U times the running time of Markov Viterbi. U is usually a small number.

The *training* of semi-CRFs generalizes the training algorithm of CRFs in a manner analogous to the generalization of the Viterbi algorithm. We refer for details to [16].

3.3 Matching

Matching or deduplicating entities using statistical learning models is also a popular research topic [12, 2, 15]. At the core of most of these models is a binary classifier on various similarity measures between a pair of records where the prediction is "1" if the records match and "0" otherwise. This is a straight-forward binary classification problem where the features typically denote various kinds of attribute level similarity functions like Edit distance, Soundex, N-grams overlap, Jaccard, Jaro-Winkler and Subset match [6]. Thus, we can use any binary classifier like SVM, decision trees, and logistic regression. We use a CRF with a single variable for extensibility. Thus, given a text segment pair (s_1, s_2), the CRF predicts a r that can take values 0 or 1 as

$$\Pr(r|s_1, s_2) = \frac{e^(\mathbf{W}.\mathbf{F}(r, s_1, s_2))}{Z(s_1, s_2)} \quad (7)$$

The feature vector $\mathbf{F}(r, s_1, s_2)$ corresponds to various similarity measures between the text segments when $r = 1$, i.e., when s_1 and s_2 match.

4 Our models for data integration

In this section we describe how all the clues available across the labeled data and the multi-relational database can be combined to solve our data integration task. Let $\mathbf{x} = x_1 \ldots x_n$ denote the input unstructured string with n tokens obtained from the text using any tokenization method. Typically, text is tokenized based on white space and a limited set of delimiters like "," and ".". Our goal is to extract from $\mathbf{x}$, segments of text belonging to one of the entity attributes in the database $A = \{A_1, \ldots A_m\}$. We also need to assign to each segment labeled with an attribute a_j a canonical id c_j of an entry in the A_j column of the database with which it matches. If it matches none of the ids, then c_j is a special value "0" denoting "none-of-the-above". Also, canonical ids connected through foreign links need to respect the foreign keys in the database and the labels need to follow the cardinality constraints imposed by the schema.

4.1 Model for extraction

We deploy Semi-CRFs since they provide high-accuracy and an elegant mechanism for extracting entities in the presence of databases. The main challenge in their usage is how to incorporate clues obtained from the existing entities in the database. A straightforward mechanism would be to add the existing list of entity values as additional labeled training examples. However, in practice we found this approach to be very sensitive to the training set and size and often leads to a drop in accuracy.

We therefore designed models that separate out the roles of database entities and unstructured records. We use the database to add three types of entity-level features in the sequential model trained over unstructured text.

4.1.1 Similarity to a Word-level dictionary

Very often an entity in an unstructured database might already be existing in the database albeit in a slightly different form. A number of similarity functions have been proposed in the record-linkage literature for defining efficient and effective similarity functions which are tolerant to commonly occurring variations in named entities. We can exploit these to perform better extraction by adding similarity features as proposed in [7]. An example feature based on this is: $f^9(y_i, y_{i-1}, \mathbf{x}, 3, 5) =$

$\max_{e \in \text{Authors in } D} cosine(x_3x_4x_5, \text{e}) \cdot [\![y_i = \text{Author}]\!]$

This assigns as feature value the maximum cosine similarity that segment $x_3x_4x_5$ has to the Author_name column of the database.

4.1.2 Similarity to Pattern-level dictionary

Each entity in the database contains valuable information at the pattern-level that can be exploited even if the entity in the unstructured text is not the same. We design a succinct set of features to exploit such patterns as follows. We first choose a sequence of regular expression patterns. Then for each entry e in the database, we generalize each of the tokens of e to the first pattern it matches, concatenate the pattern-ids over all tokens in e and insert the concatenated pattern-ids of e in the index. For example, suppose our regular expression pattern sequence consisted of the following three patterns:

id	1	2	3
pattern	$[A-Z].$	$[A-Z][a-z]+$	$[A-Z]\{Punct\}$

Then the text string "R. L, Stevenson"' will get the pattern-id sequence: 1,3,2 assuming the text is tokenized on space.

We find the pattern representation of all entities in the database and index them separately for each entity type. We then design *boolean* features for segments that check for the presence of the segment's pattern in each entity's pattern dictionary.

4.1.3 Entity classifier

The database contains a variety of other clues including typical length of entities, characteristics words and multiple patterns per tokens, that is not captured by the above two dictionaries. We capture all these by building an entity classifier where an entity is characterized by as many such features as required and a label that denotes its type. In addition, we also include subsets of this entity with a negative label. This defines a multi-class classification problem. The CRF model (or any other classifier like SVM) can be used to train a multi-class classifier. Let C_D denote such a classifier.

When C_D is applied on a text segment s, it returns a vector of scores that denotes the probability that s is of each of the given entity types. Let E_L denote the extraction model that we are constructing from the unstructured examples. We use classifier C_D to define a new set of semi-markov features for model E_L. If the number of distinct entity types is m, we will have m^2 features where the (i, j) feature captures the strength of the jth score obtained from C_D in identifying the ith entity in E_L.

4.2 Model for matching

Given a segment s_j of $\mathbf{x}$ for which the attribute label a_j resolves to one of the m attributes $A_1 \ldots A_m$, we need to find a canonical id c_j that denotes which existing values of the attribute it matches in the database and c_j = "0" if it matches none of the existing entities.

We adopt the pair-wise match model of Section 3.3 where for each canonical labeled segment in the training data we form pairs with the database entities. The entities with the same canonical id get a label of "1" and others get a label of "0". In addition to examples in the training data we can easily also generate pairs of records from the database and assign a label of "1" to those with the same canonical-id and zero otherwise.

An important concern about the pair-wise matching process is efficiency. We cannot afford to attempt potential match with each and every possible canonical id for which there is an entry in the database. We solve this problem by using a top-K index for each database column. Given a text segment s_j, we probe the index to get a candidate set of segments $\{s_{j1}, s_{j2}, \ldots, s_{jK}\}$ and evaluate $\Pr(c_i|s_j, s_{ji})$ for each candidate s_{ji} with canonical label c_i. If the predicted value r is "0" for each candidate, we claim that the segment resolves to none-of-the-existing entities.

When presented with a record with multiple extracted attribute segments $s_1 \ldots s_p$, we need to simultaneously assign canonical-ids to each of these so as to not violate any of the integrity constraints. We first search the attribute index to find the top-K candidate canonical id for each attribute segment. We then pose database queries to find all canonical ids c'_j which have direct or indirect foreign key link(s) from each of the top-K canonical ids c_j associated with the attribute segment s_k. We denote the set of link constraints as $\mathcal{L}$, which is a set of tuples of the form (a_j, c_j, a'_j, c'_j), where a_j is the attribute type of an attribute segment s_k, c_j is one of the top-K canonical ids for the segment, a'_j is an attribute having a foreign key relationship with a_j, and c'_j is the canonical id of the entry with the attribute a'_j and is being referred by the entry c_j.

Assuming that a segment sequence has two attribute segment s_k and s'_k such that they are of attribute types a_j and a'_j respectively, then a link constraint is said to be violated

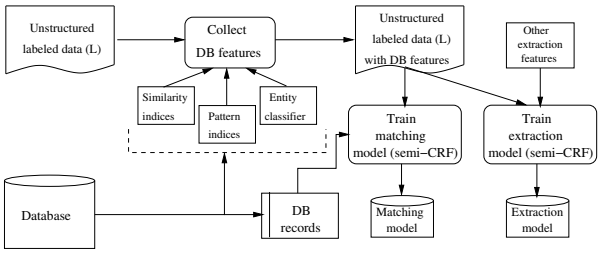


Figure 3. Training process

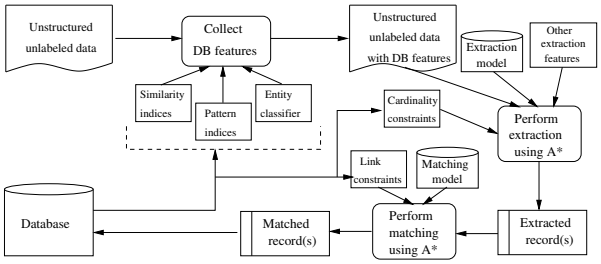


Figure 4. Deploy process

if any of the following is true:

- Attribute a_j has a foreign link to the attribute $a_{j'}$ in the database schema, and canonical id c_j assigned to the segment s_k points to a canonical id other than c'_j assigned to s'_k, i.e. $(a_j, c_j, a'_j, c'_j) \notin \mathcal{L}$.
- same as above with the roles of j and j' reversed.

4.3 Overall training and deployment process

The overall training and deployment process for extraction and matching appears in Figures 3 and 4.

4.3.1 Training process

During the offline training phase, we first train the entity classifier out of the entities in the database. The entity classifier could be trained incrementally but we do not support that currently. For the similarity and pattern features, we maintain an inverted index over each of the entity columns that is always kept upto-date with the entries in the database.

Next, for each labeled document we create its normal intrinsic features and then probe the entity classifier and indices to create the three kinds of database derived features. The index probes for computing the various dictionary features is performed using a Top-K search on the inverted index. Also, since this requires multiple index probes on overlapping segments of the input, we have designed a batch top-K index search algorithm [5] where instead of independently finding top-K matches for all candidate segments, we batch the top-K queries over overlapping segments. This

1: Perform backward Viterbi scan and cache all the $n.|Y|$ possible suffix upper bound scores $V'(i, y)$ (See eq: 8)
2: Initialize priority queue P with the start state $S = (0, null)$.
3: Initialize global-LB= $-\infty$.
4: **while** P is not empty **do**
5: Retrieve a state τ with the maximum sequence upper bound estimate $\gamma(\tau) = \alpha(\tau) + \beta(\tau)$ from the queue P.
6: **if** τ is a goal state **then**
7: Return the solution
8: **end if**
9: Generate successors of the state τ, and add valid successors τ' to P provided $\gamma(\tau') \geq$ global-LB.
10: Periodically update global-LB via beam search from τ.
11: **end while**
12: Return failure

Figure 5. Constrained inference using A*

significantly reduces the time required for finding matches. We do not consider such optimizations in this paper because of lack of space and because our main focus here is accuracy of extraction via skillful feature design. We cache the index lookups but compute all the other features on the fly for each training iteration.

We then train a semi-CRF model for extraction that outputs the weight values for each feature type. We train the match model by forming pairs both out of top-K matches of the training segments with the database entities and also with pairs formed purely out of database entities. The trained models are stored on disk for the deployment stage.

4.3.2 Deployment process

While making inference on the labels of an unstructured text, we also need to follow the cardinality constraints imposed by the database schema. The Viterbi algorithm presented earlier cannot easily meet these constraints because this upsets the Markovian property that Viterbi depends on. Section 4.4 presents an A* based search algorithm that can efficiently find the optimal segmentation even with constraints. After extracting the attribute segments, we assign canonical ids while maintaining the link constraints using another run of the same A* algorithm.

4.4 The A* inference algorithm

A simple mechanism to extend Viterbi for constrained labeling is via a beam search which maintains not just the best path but the best B paths at any position. When we extend the path by one more segment, we check for conflicts and drop invalid extensions. This is guaranteed to find the optimal solution when B is very large and the time and space requirement is multiplied by B. In practice, conflicts

are rare and the beam search algorithm might be unnecessarily maintaining alternatives that are never used.

We propose a better optimized inference algorithm based on the A* search paradigm. This algorithm is guaranteed to find the optimal solution given unbounded space but because of the non-Markovian constraints we cannot guarantee optimality in polynomial time. The algorithm allows us to tradeoff optimality and efficiency by controlling the number of search nodes expanded. Even when it is made to terminate early the solution will be no worse than a normal Viterbi run with a fixed beam size.

A* is a branch and bound search algorithm that depends on upper and lower bounds guarantees associated with states in the search space to reach the optimal solution via the minimum possible number of states. An outline of the inference algorithm based on the A* technique is given in Figure 5.

Our state space consists of the set of partial segmentation starting from the beginning of the sequence to an intermittent position i. Thus, each state τ in the state space can be expressed as a sequence of segments $\tau(s) = s_1 \ldots s_k$ such that the first one starts at 1 and the last one ends at i, i.e., using the segment notation from Section 3.2, $s_1.l = 1$ and $s_k.u = i$.

We denote a state as $\tau(i, s_1 \ldots s_k)$, the start state as $S(0, null)$ and a goal state as $G(n, s_1 \ldots s_p)$, where $s_p.u = n$. Each state τ has associated with it an upper bound score $\gamma(\tau)$ based on which it is ordered in a priority queue P. Initially, P has only the start state. In each iteration we dequeue the state with the largest upper bound and generate its successors as follows:

Generating successors The successors of a given state τ are all the states which extend the segmentation $\tau(s) = s_1 \ldots s_k$ with a segment s_{k+1} which starts at $i+1$, i.e., $s_{k+1}.t = s_k.u + 1 = i + 1$, ends at a position no greater than $i + U$ and has a label that does not violate any constraint given the sequence of labels in the segmentation of τ. The constraints imposed can be any arbitrary constraints on label assignments. In our case, cardinality constraints derived from the database schema are imposed during extraction. While for matching, we enforce link constraints derived from the database entities as discussed in Section 4.2.

Calculating upper bound We now discuss how to compute the upper bound score $\gamma(\tau)$ associated with a state τ. This score consists of two parts: the first part $\alpha(\tau)$ is the actual score over the partial segment sequence $\tau(\mathbf{s})$; thus $\alpha(\tau) = \mathbf{W} \cdot \mathbf{F}(\mathbf{x}, \tau(\mathbf{s}))$. This value can be computed incrementally when a state τ' is generated from its parent τ with the extension of a single segment. The second part $\beta(\tau)$ is an upper bound on the maximum score possible if $\tau(\mathbf{s})$ is extended with segments that take it to a goal state. The sum $\gamma(\tau) = \alpha(\tau) + \beta(\tau)$ thus is an upper bound on the maximum possible score of the entire sequence from the start position 1 to an ending position n with the prefix of the path constrained to be $\tau(\mathbf{s})$.

The value of $\beta(\tau)$ is computed for all states in a single backward Viterbi pass that ignores all constraints, and thus provides an upper bound on the maximum possible score. Formally, let $\mathbf{s}'_{j:y}$ denote the set of all partial segmentations starting from the position $j+1$ to n, such that the segment ending at position j has the label y. Let $V'_{\mathbf{x},\mathbf{f},\mathbf{W}}(j, y)$ denote the largest value of $\mathbf{W} \cdot \mathbf{F}(\mathbf{x}, \mathbf{s}')$ for any $\mathbf{s}' \in \mathbf{s}'_{j:y}$. Omitting the subscripts, the following recursive calculation implements a semi-Markov, backward Viterbi algorithm:

$$V'(j, y) = \begin{cases} \max_{y', j'=j+1\ldots j+U} V'(j', y') \\ \quad + \mathbf{W} \cdot \mathbf{f}(y, y', \mathbf{x}, j+1, j') & \text{if } j < n \\ 0 & \text{if } j = n \\ -\infty & \text{if } j > n \end{cases} \quad (8)$$

The value $V'(j, y)$ is the suffix upper bound score $\beta(\tau)$ for any partial segmentation $\tau(j, s_1 \ldots s_k)$ such that s_k ends at j and has label y.

4.4.1 Pruning on lower bounds

For most of the inference tasks, the number of state expansions before reaching a goal state is small. But sometimes, with many label constraints, the number of expansions can be large. For such cases, we limit the number of expansions and the size of the priority queue to some predefined maximum value. Another idea is to prune nodes which will never be part of an optimal solution. We can determine a lower bound solution using beam search Viterbi for constrained inference described earlier. Here, the beam size need not be very large, actually the value of 3-5 suffices in practice. The solution can be used as a lower bound, and the states with the sequence upper bound less than the lower bound can be pruned. The lower bound can be dynamically updated by extending the partial solution of the current best state using the beam search Viterbi technique. The frequency of updates of lower bound is a function of the number of expansions and queue size to optimize the overall performance. We skip details due to lack of space.

5 Experiments

We evaluated our system on the following datasets.

Personal_Bibtex In this case our goal was to simulate the scale of databases as would arise in a PIM. One of the authors of this paper had a set of 400 journal entries over her .bib files collected over 10 years. This was loaded on a normalized relational database following a schema similar to that shown in Figure 1 with one intermediate table called

Journal_issues to further normalize journals at the level of individual issues. The *articles* table points to *Journal_issues* which in turn points to *Journals*. The set of attributes consisted of title and page numbers from articles, year and volume from journal-issues, author names, and journal names. Of these, year, page-numbers and volume are fairly easy to detect and provide close to 95% accuracy with or without a DB. We therefore concentrate only on the remaining three fields in reporting accuracy values.

The unstructured records consisted of 100 citations obtained by searching Citeseer for citations of a random subset of authors appearing in the bibtex database. These were manually tagged with attribute and match ids. Of these 45% titles, 78% journals and 75% authors already appeared in the database. These were not exact matches but noisy inexact matches that needed manual resolution. We have published the dataset[1] for others to use.

Address data The Address dataset consists of 395 home addresses of students in IIT Bombay India. These addresses are less regular than US addresses, and extracting even fields like city names is challenging [3]. We purchased a postal database that contains a hierarchy of India pin-codes, cities and states arranged over three tables linked via foreign keys. All state names and 95% of the city names appeared in the database but the database was clean and contained no canonical variants whereas in the unstructured text there were several noisy variants. The labeled data had other fields also like Road names and building names totaling to a total of 16. We trained the extraction model over all sixteen labels but report accuracy over only the three labels for which we had columns in the database.

Platform We used Postgres to store the structured database and Lucene to index the text columns of the database. Our extraction and matching code was in Java. The learner was the open source CRF and semi-CRF package that we have contributed to sourceforge (http://crf.sf.net).

Features Features followed the template in Section 5.2. Each token contributed two types of features: (1) the token itself if it was encountered in the training set, otherwise a special feature called "Unknown" and, (2) the set of regular expressions like digit or not, capitalized or not that the token matches. In the semi-Markov model for each segment we invoke features on the first, last and middle tokens of the segment. The context features consist of features on the tokens one left and one right of the segment. These are all Markov features that can be handled by any Begin, Continue, End state encoding [4]. In addition, edge features captured the dependency between labels assigned to adjacent segments. The semi-Markov non-database features were segment length and counts for each possible regular expression that fires. The semi-Markov database features were of three types as described in Section 4.1. We used two similarity functions JaroWinkler and SoftTFIDF [6] in addition to Lucene scores.

Experiment settings Since our system is targeted to settings when labeled data (L) is limited, by default we used roughly 5% of the available data for training. The rest was used for testing. We also report numbers with increasing fraction of training data. All numbers are averaged over five random selections of training and test data. We measure accuracy in terms of the correctness of the entire extracted entity (*i.e.*, partial extraction gets no credit). We report for each label recall, precision and F1 values[2].

Our main focus in this paper is accuracy and we will study that in Sections 5.1 to 5.4. However, to establish practical feasibility of using a learning-based system, we will also report summarized results on training and test time in Section 5.5.

5.1 Overall data integration performance

In this section, we first present accuracy of the full integration task (extraction + matching) by more centrally exploiting the database.

We summarize the results in Table 1 where we report precision, recall and F1 values of integration for each text field for our datasets introduced earlier. The first set of numbers is where the database does not contribute to the learning models at all and the models are trained using the unstructured data alone. The second set of numbers is due to our model trained using both database specific features and features from unstructured text for both extraction and matching. The table shows that for all labels there is a significant improvement in F1 accuracy. For some fields like journals of the PersonalBib dataset and state names of address dataset, F1 improves by more than 50%.

The table also shows separate extraction and matching accuracies. For all labels there is a significant improvement ranging from F1 differences of 7 to 25 of extraction performance. For example, for journal we were able to increase F1 from 35 to 51 whereas for state we got a jump from 23 to 49. The overall average F1 went up by 10 points which is a 40% reduction in error.

In case of matching with databases, we added approximately 100 examples from the database as training instances. Matching results for IITB database with and without databases are almost the same. For personal bibtex

[1] http://www.it.iitb.ac.in/šunita/data/personalBib.tar.gz

[2] F1 is defined as 2*precision*recall/(precision+recall).

		Integration						Extraction						Match					
		Only-L			L + DB			Only-L			L + DB			Only-L			L + DB		
Dataset	Label	P	R	F1	P	R	F1	P	R	F1	P	R	F1	P	R	F1	P	R	F1
PersonalBib	author	63	76	69	79	84	81	67	81	73	74	85	79	94	98	96	94	98	96
	journal	34	26	29	64	50	56	40	31	35	56	48	51	88	93	90	88	94	91
	title	58	50	54	80	53	63	68	59	63	74	68	71	76	63	69	94	76	86
Address	city	67	69	68	75	76	77	70	72	71	78	78	78	97	99	98	97	99	98
	state	30	14	19	50	37	51	36	17	23	59	42	49	96	85	89	96	85	89
	pincode	89	87	88	92	91	90	94	92	93	96	94	95	94	92	93	96	94	95

Table 1. Accuracy of overall integration, extraction and matching with and without an external database (P=precision, R=recall)

database, there is a significant improvement in the accuracy of title when the database tuples are added in the training, while for other labels (journal and author) the accuracies are comparable.

We now present a finer grained analysis of the results and study the effects of various factors like data sizes, feature sets and databases on extraction since that is what contributes to most of the difference.

5.2 Effect of various features

We assess the usefulness of various kinds of clues for aiding the extraction tasks by evaluating performance with various feature combinations for two different train sizes of PersonalBib. In Figure 6, we show F1 numbers, starting with an extraction model trained only on features of L and then we add four different database features in sequence: schema derived cardinality information, similarity features, entity classifier and regex-level match features. Accuracy improves with each additional feature type. The first significant improvement was on using the similarity features of the database, followed by the improvement due to regex features. Next we evaluate the importance of features derived purely from L by dropping features of three types: entity, context and label_order while keeping all the database features. Dropping the entity features of L raises accuracy slightly, probably because given the small training set these tend to overfit the data. The context and edge (label dependency) features are important as seen by the drop in accuracy. Interestingly, if context and edge features are removed in parallel before removing entity features (not shown in figure), accuracy does not drop much. Thus, one of context and edge features is important and the two together are not needed.

5.3 Effect of increasing training data

We show how the relative accuracy of extraction with and without a database is affected by increasing labeled training data. For these experiments we fixed the test set to 50% of available and selected different random subsets for training. Figure 7 shows average F1 with increasing training sizes. Accuracy increases as expected in both cases and the database continues to be useful even for larger training set.

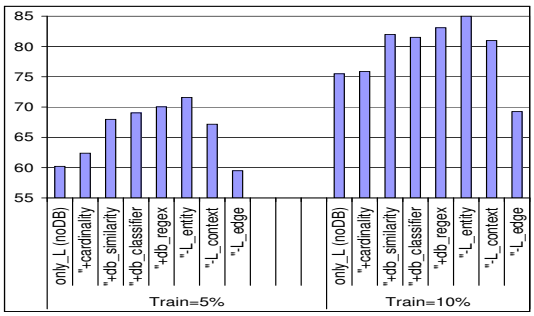


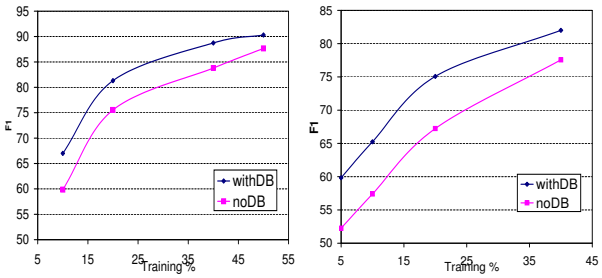
Figure 6. Effect of various features from the database (db) and labeled data (L) on F1 accuracy

5.4 Effect of changing database

In order to evaluate the sensitivity of our results to particular databases, we report numbers where our personal bibtex database is changed to the publicly available DBLP database while keeping L unchanged. In Table 2, we report performance on the two databases. We notice that title performance remains unchanged whereas author and journal accuracy drop by 4 and 3 F1 points respectively. However, there is still an overall 14% reduction in extraction error due to use of the database.

5.5 Running time

As the time required to train an extraction model dominates the integration process, we report performance number for extraction only. We compare the time required for extraction with and without databases.



(a) Personal Bibtex (b) Address data

Figure 7. Increasing fraction of labeled data (L) versus F1 with and without a database

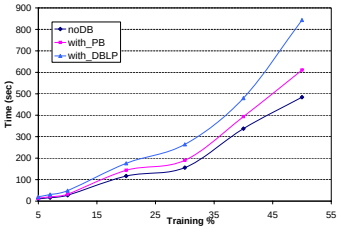


Figure 8. Increasing fraction of labeled data (L) versus running time (seconds) with and without databases

	Only-L			L + PersonalBib			L + DBLP		
Label	P	R	F1	P	R	F1	P	R	F1
author	67	81	73	74	85	79	70	82	75
journal	40	31	35	56	48	51	53	44	48
title	68	59	63	74	68	71	74	68	71
overall	63	66	64	71	74	72	68	71	69

Table 2. Accuracy of extraction with changing databases (P=precision, R=recall)

We report training times for extraction models in three settings: without a database, with the smaller personal bibtex and the larger DBLP database. The personal bibtex database has 400 article tuples, 1800 author entries and 200 journals. The DBLP database is relatively large and contains approximately 60000 articles, 70000 authors, and 200 journals. Figure 8 shows the average time required to train an extraction model for various training set sizes under these three settings. Even with 50 training records and the larger DBLP database, the time required is no more than 15 minutes, and less than double the time required by the no-database setting. It is possible to reduce the gap further by more efficient index probes as discussed in [5].

Once an extraction models is trained, it is applied on unstructured instances using the A* algorithm discussed in Section 4.4. For extraction without databases, inference on a single bibtex entry took on average 800 ms. For extraction with personal bibtex database average inference time is around 1800 ms, while with DBLP database it is 4600 ms. This time can also be reduced considerably via efficient index lookups [5].

6 Related work

External dictionaries have been exploited to improve accuracy of NER tasks based on conditional models. A common scheme is to define boolean features based on exact match of a word with terms that appear in an entity dictionary [4]. This is extended in [7] to handle noisy dictionaries through features that capture various forms of similarity measures with dictionary entries. In both these cases the goal is to exploit entity lists in isolation whereas our goal is to handle multiple inter-linked entity lists. Also, we exploit entity lists more effectively by building an entity classifier and pattern dictionaries.

Another mechanism is to treat the dictionary as a collection of training examples and this has been explored for the case of generative models in [17, 3] This method suffers from a number of drawbacks: there is no obvious way to apply it in a conditional setting; it is highly sensitive to misspellings within a token; and when the dictionary is too large or too different from the training text, it may degrade performance. In [1], some of these drawbacks are addressed by more carefully training a HMM so as to allow small variations in the extracted entities, but this approach being generative is still restricted in its scope of the variety of features that it can handle compared to recent conditional models. Also, it cannot effectively combine information available in both labeled unstructured data and external databases.

Recently there has also been work on exploiting links for matching entities in a relational database, including conditional graphical models for grouped matching of multi-attribute records [13] and CRFs for matching a group of records enforcing the transitivity constraint [12]. These are techniques for batched deduplication of a group of records whereas we are addressing an online scenario where unstructured records get inserted one-at-a-time in the database.

7 Conclusions

In this paper we showed how to integrate unstructured text records into existing multi-relational databases using

models that combine clues from both existing entities in the database and labeled unstructured text. We extend state-of-the-art semi-Markov CRFs with a succinct set of features to capture pattern-level and entity-level information in the database, use these to extract entities and integrate them in a database while respecting its key constraints. Experiments show that our proposed set of techniques lead to significant improvement in accuracy on real-life datasets.

This is one of the first papers on statistical models for integrating unstructured records into existing databases and there is lot of opportunity for future work. We plan to develop other ways of exploiting a database of entities particularly inter-entity correlation expressed via soft-constraints. Another important aspect is creating standardized version of entities either by choosing the best of the existing variants or merging several noisy variants.

Acknowledgements This work was supported by Microsoft Research and an IBM faculty award. The last author thanks them for their generous support.

References

[1] E. Agichtein and V. Ganti. Mining reference tables for automatic text segmentation. In *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Seattle, USA*, 2004.

[2] M. Bilenko, R. Mooney, W. Cohen, P. Ravikumar, and S. Fienberg. Adaptive name-matching in information integration. *IEEE Intelligent Systems*, 2003.

[3] V. R. Borkar, K. Deshmukh, and S. Sarawagi. Automatic text segmentation for extracting structured records. In *Proc. ACM SIGMOD International Conf. on Management of Data*, Santa Barabara,USA, 2001.

[4] A. Borthwick, J. Sterling, E. Agichtein, and R. Grishman. Exploiting diverse knowledge sources via maximum entropy in named entity recognition. In *Sixth Workshop on Very Large Corpora New Brunswick, New Jersey. Association for Computational Linguistics.*, 1998.

[5] A. Chandel, P. Nagesh, and S. Sarawagi. Efficient batch top-k search for dictionary-based entity recognition. In *Proc. of the 22nd IEEE Int'l Conference on Data Engineering (ICDE)*, 2006.

[6] W. W. Cohen, P. Ravikumar, and S. E. Fienberg. A comparison of string distance metrics for name-matching tasks. In *Proceedings of the IJCAI-2003 Workshop on Information Integration on the Web (IIWeb-03)*, 2003. To appear.

[7] W. W. Cohen and S. Sarawagi. Exploiting dictionaries in named entity extraction: Combining semi-markov extraction processes and data integration methods. In *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2004.

[8] J. Lafferty, A. McCallum, and F. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the International Conference on Machine Learning (ICML-2001)*, Williams, MA, 2001.

[9] S. Lawrence, C. L. Giles, and K. Bollacker. Digital libraries and autonomous citation indexing. *IEEE Computer*, 32(6):67–71, 1999.

[10] D. C. Liu and J. Nocedal. On the limited memory bfgs method for large-scale optimization. *Mathematic Programming*, 45:503–528, 1989.

[11] R. Malouf. A comparison of algorithms for maximum entropy parameter estimation. In *Proceedings of The Sixth Conference on Natural Language Learning (CoNLL-2002)*, pages 49–55, 2002.

[12] A. McCallum and B. Wellner. Toward conditional models of identity uncertainty with application to proper noun coreference. In *Proceedings of the IJCAI-2003 Workshop on Information Integration on the Web*, pages 79–86, Acapulco, Mexico, Aug. 2003.

[13] Parag and P. Domingos. Multi-relational record linkage. In *Proceedings of 3rd Workshop on Multi-Relational Data Mining at ACM SIGKDD*, Seattle, WA, Aug. 2004.

[14] A. Ratnaparkhi. Learning to parse natural language with maximum entropy models. *Machine Learning*, 34, 1999.

[15] S. Sarawagi and A. Bhamidipaty. Interactive deduplication using active learning. In *Proc. of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining(KDD-2002)*, Edmonton, Canada, July 2002.

[16] S. Sarawagi and W. W. Cohen. Semi-markov conditional random fields for information extraction. In *NIPs*, 2004.

[17] K. Seymore, A. McCallum, and R. Rosenfeld. Learning Hidden Markov Model structure for information extraction. In *Papers from the AAAI-99 Workshop on Machine Learning for Information Extraction*, pages 37–42, 1999.

[18] F. Sha and F. Pereira. Shallow parsing with conditional random fields. In *In Proceedings of HLT-NAACL*, 2003.