Exploiting Dictionaries in Named Entity Extraction: Combining Semi-Markov Extraction Processes and Data Integration Methods

William W. Cohen
Center for Automated Learning and Discovery
Carnegie Mellon University
Pittsburgh, PA 15213
wcohen@cs.cmu.edu

Sunita Sarawagi[*]
IIT Bombay
Mumbai-400076 India
sunita@iitb.ac.in

ABSTRACT

We consider the problem of improving named entity recognition (NER) systems by using external dictionaries—more specifically, the problem of extending state-of-the-art NER systems by incorporating information about the similarity of extracted entities to entities in an external dictionary. This is difficult because most high-performance named entity recognition systems operate by sequentially classifying words as to whether or not they participate in an entity name; however, the most useful similarity measures score entire candidate names. To correct this mismatch we formalize a semi-Markov extraction process, which is based on sequentially classifying *segments* of several adjacent words, rather than single words. In addition to allowing a natural way of coupling high-performance NER methods and high-performance similarity functions, this formalism also allows the direct use of other useful entity-level features, and provides a more natural formulation of the NER problem than sequential word classification. Experiments in multiple domains show that the new model can substantially improve extraction performance over previous methods for using external dictionaries in NER.

Categories and Subject Descriptors: H.3.1[Information Storage and Retrieval]: Content Analysis and Indexing—*Dictionaries*; I.2.6[Artificial Intelligence]: Learning

General Terms: Algorithms, Experimentation.

Keywords: Learning, information extraction, named entity recognition, data integration, sequential learning.

1. INTRODUCTION

Named entity recognition (NER) is finding the names of entities in unstructured text. Well-studied cases of NER are identifying personal names and company names in newswire text (*e.g.*, [5]), identifying gene and protein names in biomedical publications (*e.g.*, [7, 20]), and identifying titles and authors in on-line publications (*e.g.*, [25, 29]). Named entity recognition is an important step in deriving structured database records from text.

In many cases, the ultimate goal of this information extraction process is to answer queries which *combine* information from structured and unstructured sources. For example, a biologist might want to look for publications about proteins from a particular superfamily, where the superfamily is defined in a structured database of biomedical information; a business analyst might want to find articles concerning companies in a particular industry sector; or an intelligence analyst might wish to look for documents that "link" persons previously known to have engaged in suspicious activity. In each of these applications, NER is successful only to the extent that it finds entity names that can be matched to something in a pre-existing database.

When NER methods are used as the first step of such a query process, it is natural to want to optimize them so that they perform best on the most important entities—*i.e.*, entities that appear in the external databases that will be used in these structured queries. Moreover, it is reasonable to expect that a large collection of names of known entities (such as the collection associated with some type in a structured database) would improve NER performance.

This paper investigates this problem—*i.e.*, the task of improving NER systems using external dictionaries. This problem is surprisingly subtle. Naively, one might expect that given a large dictionary, simply looking for exact matches to some dictionary entry would be a reasonable NER method. In fact, this is seldom the case. The surface form of a name in free text can vary substantially from its dictionary version, leading to issues analogous to those that arise in linking or "de-duping" heterogeneous database records [10, 32]. This problem is compounded in extracting from text which is informal or otherwise prone to noise and errors, such as the email corpus and the address corpus we consider in the experiments in this paper. Thus taking a good external dictionary and transforming it to a useful NER system is often difficult.

Conversely, taking a state-of-the-art NER system and incorporating information about possible linkage to an external dictionary is also non-trivial. The primary issue here

[*]Both authors contributed equally to this research.

is that most high-performance NER systems operate by sequentially classifying *words* as to whether or not they participate in an entity name, while record-linkage systems operate by scoring *entire candidate names* by similarity to an existing dictionary entry. This fundamental mismatch in representation means that incorporating dictionary information is awkward, at best.

In this paper we will discuss a new formalism for NER that corrects this mismatch. We describe a semi-Markov extraction process which relaxes the usual Markov assumptions made in NER systems. This process is based on sequentially classifying *segments* of adjacent words, rather than single words. In addition to allowing a natural way of linking NER and high-performance record linkage methods, this formalism also allows the direct use of other useful entity-level features, such as the length of an entity. It is also arguably a more natural formulation of the NER problem than sequential word classification, in that it eliminates certain decisions about problem encoding.

Below we will present the new model and describe a learning algorithm for it. We then present experimental results for the new algorithm, discuss related work, and conclude.

2. ALGORITHMS FOR NAME-FINDING

2.1 Name-Finding as Word Tagging

Named entity recognition (NER) is the process of annotating sections of a document that correspond to "entities" such as people, places, times and amounts. As an example, the output of NER on the email document

Fred, please stop by my office this afternoon.

might be

(Fred)$_{\text{Person}}$ please stop by (my office)$_{\text{Loc}}$ (this afternoon)$_{\text{Time}}$

A common approach to NER is to convert name-finding to a *tagging* task. A document is encoded as a sequence $\mathbf{x}$ of tokens $x_1, \ldots, x_N$, and a *tagger* associates with $\mathbf{x}$ a parallel sequence of tags $\mathbf{y} = y_1, \ldots, y_N$, where each y_i is in some tag set Y. If these tags are appropriately defined, the name segments can be derived from them. For instance, one might associate one tag with each entity type above, and also add a special "other" tag for words not part of any entity name, so that the tagged version of the sentence would be

Fred	please	stop	by	my	office	this	afternoon
Person	Oth	Oth	Oth	Loc	Loc	Time	Time

A common way of constructing such a tagging system is to learn a mapping from $\mathbf{x}$ to $\mathbf{y}$ from data [3, 5, 27]. Typically this data is in the form of annotated documents, which can be readily converted to $(\mathbf{x}, \mathbf{y})$ pairs.

Most methods for learning taggers exploit, in some way, the sequential nature of the classification process. In general, each tag depends on the tags around it: for instance, if person names are usually two tokens long, then if y_i is tagged "Person" the probability that y_{i+1} is a "Person" is increased, and the probability that y_{i+2} is a "Person" is decreased. Hence the most common learning-based approaches to NER learn a sequential model of the data, generally some variant of a hidden Markov model (HMM) [15].

It will be convenient to describe our framework in the context of one of these HMM variants, in which the conditional distribution of $\mathbf{y}$ given $\mathbf{x}$ is defined as

$$P(\mathbf{y}|\mathbf{x}) = \prod_{i=1}^{|\mathbf{x}|} P(y_i|i, \mathbf{x}, y_{i-1})$$

(Here we assume a distinguished start tag y_0 which begins every observation.) This is the formalism used in maximum entropy taggers [30], and it has been variously called a maximum entropy Markov model (MEMM) [28] and a conditional Markov model (CMM) [21]. Inference in this model can be performed with a variant of the Viterbi algorithm used for HMMs. Given training data in the form of pairs $(\mathbf{x}, \mathbf{y})$, the "local" conditional distribution $P(y_i|i, \mathbf{x}, y_{i-1})$ can be learned from derived triples $(y_i, i, \mathbf{x}, y_{i-1})$, for example by using maximum entropy methods.

2.2 Semi-Markovian NER

We will relax this model by assuming that tags y_i do not change at every position i; instead, tags change only at certain selected positions, and after each tag change, some number of tokens are observed. Following work in semi-Markov decision processes [35, 18] we will call this a conditional semi-Markov model (CSMM).

For notation, let $\mathbf{S} = \langle S_1, \ldots, S_M \rangle$ be a "segmentation" of $\mathbf{x}$. Each *segment* S_j consists of a *start position* t_j, which is an index between 1 and M, an *end position* u_j, and a *label* $\ell_j \in Y$. A segmentation $\mathbf{S}$ is *valid* if $\forall j$, $t_j = u_{j-1} + 1$. We will consider only valid segmentations.

Conceptually, a segmentation means that the tag ℓ_j is given to all x_i's between $i = t_j$ and $i = u_j$, inclusive: alternatively, it means that the tags $y_{t_j} \ldots y_{u_j}$ corresponding to $x_{t_j} \ldots x_{u_j}$ are all equal to ℓ_j. Formally, let $J(\mathbf{S}, i)$ be the index j such that $t_j \leq i \leq u_j$, and define the *tag sequence* $\mathbf{y}$ *derived from* $\mathbf{S}$ to be $\ell_{J(\mathbf{S},1)}, \ldots, \ell_{J(\mathbf{S},|\mathbf{x}|)}$.

For instance, a segmentation for the sample sentence above might be $\mathbf{S} = \{(1, 1, \text{Person}), (2, 4, \text{Oth}), (5, 6, \text{Loc}), (7, 8, \text{Time})\}$, which could be written as

(Fred)$_{\text{Person}}$ (please stop by)$_{\text{Oth}}$ (my office)$_{\text{Loc}}$ (this afternoon)$_{\text{Time}}$

A CSMM is defined by a distribution over pairs $(\mathbf{x}, \mathbf{S})$ of the form

$$P(\mathbf{S}|\mathbf{x}) = \prod_j P(S_j|t_j, \mathbf{x}, \ell_{j-1}) \qquad (1)$$

More generally, we use the term *semi-Markov model* (SMM). for any model in which each S_j depends only on the label ℓ_{j-1} associated with S_{j-1}, and is independent of $S_{j'}$ for all $j' \neq j$, $j' \neq j-1$.

2.3 Discussion

Two issues need to be addressed: inference for CSMMs, and learning algorithms for CSMMs. For inference, we will present below a version of Viterbi for CSMMs that finds the most probable $\mathbf{S}$ given $\mathbf{x}$ in time $O(NL|Y|)$, where N is the length of $\mathbf{x}$ and L is an upper bound on segment length—that is, $\forall j$, $L \geq u_j - t_j$. Since $L \leq N$, this inference procedure is always polynomial. (In our experiments, however, it is sufficient to limit L to rather small values.)

For learning, inspection of Equation 1 shows that given training in the form of $(\mathbf{x}, \mathbf{S})$ pairs, learning the "local" distribution $P(S_j|t_j, \mathbf{x}, \ell_{j-1})$ is not much more complex than for a CMM.[1] However, conditionally-structured models like the CMM are not the ideal model for NER systems: better performance can often be obtained by algorithms that learn a single global model for $P(\mathbf{y}|\mathbf{x})$[11, 24]. Below we will also present an extension of one such "global" learning algorithm to a semi-Markov distribution.

We emphasize that an SMM with segment length bounded by L is quite different from an order-L CMM, as in an order-L CMM, the next label depends on the previous L *labels*, but not the corresponding tokens. A SMM is also different from a CMM which uses a window of the previous L tokens to predict y_i, since the SMM makes a *single* labeling decision for a segment, rather than making series of interacting decisions incrementally. In Section 3.5.3 we will experimentally compare SMM's and high-order CMMs.

2.4 Discriminative Training for SMMs

2.4.1 Perceptron-based Training

The learning algorithm we use for training SMMs is derived from Collins' perceptron-based algorithm for discriminatively training HMMs [11], which can be summarized as follows. Assume a *local feature function* $\mathbf{f}$ which maps a pair $(\mathbf{x}, \mathbf{y})$ and an index i to a vector of features $\mathbf{f}(i, \mathbf{x}, \mathbf{y})$. Define

$$\mathbf{F}(\mathbf{x}, \mathbf{y}) = \sum_{i}^{|\mathbf{x}|} \mathbf{f}(i, \mathbf{x}, \mathbf{y})$$

and let W be a weight vector over the components of $\mathbf{F}$. During inference we need to compute $V(W, \mathbf{x})$, the Viterbi decoding of $\mathbf{x}$ with W, *i.e.*,

$$V(W, \mathbf{x}) = argmax_{\mathbf{y}} \mathbf{F}(\mathbf{x}, \mathbf{y}) \cdot W$$

For completeness, we will outline how $V(W, \mathbf{x})$ is computed. To make Viterbi search tractable, we must restrict $\mathbf{f}(i, \mathbf{x}, \mathbf{y})$ to make limited use of $\mathbf{y}$. To simplify discussion here, we assume that $\mathbf{f}$ is strictly Markovian, *i.e.*, that for each component f^k of $\mathbf{f}$,

$$f^k(i, \mathbf{x}, \mathbf{y}) = f^k(g^k(i, \mathbf{x}), y_i, y_{i-1})$$

For fixed y and y', we denote the vector of $f^k(g^k(i, \mathbf{x}), y, y')$ for all k as $\mathbf{f}'(i, \mathbf{x}, y, y')$.

Viterbi inference can now be defined by this recurrence, where y_0 is the designated start state:

$$V_{\mathbf{x},W}(i, y) = \quad (2)$$
$$\begin{cases} 0 & \text{if } i = 0 \text{ and } y = y_0 \\ -\infty & \text{if } i = 0 \text{ and } y \neq y_0 \\ \max_{y'} \; V_{\mathbf{x},W}(i-1, y') \\ \quad + W \cdot \mathbf{f}'(i, \mathbf{x}, y, y') & \text{if } i > 0 \end{cases}$$

and then $V(W, \mathbf{x}) = \max'_y V_{\mathbf{x},W}(|\mathbf{x}|, y)$.

The goal of learning is to find a W that leads to the globally best overall performance. This "best" W is found by repeatedly updating W to improve the quality of the Viterbi decoding on a particular example $(\mathbf{x}_t, \mathbf{y}_t)$. Specifically, Collin's algorithm starts with $W_0 = \mathbf{0}$. After the t-th example $(\mathbf{x}_t, \mathbf{y}_t)$, the Viterbi sequence $\hat{\mathbf{y}}_t = V(W_t, \mathbf{x}_t)$ is computed. If $\hat{\mathbf{y}}_t = \mathbf{y}_t$, W_{t+1} is set to W_t, and otherwise W_t is replaced with

$$W_{t+1} = W_t + \mathbf{F}(\mathbf{x}_t, \mathbf{y}_t) - \mathbf{F}(\mathbf{x}_t, \hat{\mathbf{y}}_t)$$

After training, one takes as the final learned weight vector W the average value of W_t over all time steps t.

This simple algorithm has performed well on a number of important sequential learning tasks [11, 2, 34], including NER. It can also be proved to converge under certain plausible assumptions [11].

The natural extension of this algorithm to SMM's assumes training data in the form of pairs $(\mathbf{x}, \mathbf{S})$, where $\mathbf{S}$ is a segmentation. We will assume a feature-vector representation can be computed for any segment S_j of a proposed segmentation $\mathbf{S}$, *i.e.*, we assume a function $\mathbf{f}(j, \mathbf{x}, \mathbf{S})$. Again defining $\mathbf{F}(\mathbf{x}, \mathbf{S}) = \sum_{j=1}^{|\mathbf{S}|} \mathbf{f}(j, \mathbf{x}, \mathbf{S})$, one can apply Collins' method immediately, as long as it is possible to perform a Viterbi search to find the best segmentation $\hat{\mathbf{S}}$ for an input $\mathbf{x}$.

For SMM Viterbi search, we need to restrict each f^k to be of the form

$$f^k(j, \mathbf{x}, \mathbf{S}) = f^k(g^k(t_j, u_j, \mathbf{x}), \ell_j, \ell_{j-1})$$

and as before, we let $\mathbf{f}'(t, u, \mathbf{x}, y, y')$ be the vector of f^k's. To implement Viterbi, we use the recurrence:

$$V_{\mathbf{x},W}(i, y) = \quad (3)$$
$$\begin{cases} 0 & \text{if } i = 0 \text{ and } y = y_0 \\ -\infty & \text{if } i = 0 \text{ and } y \neq y_0 \\ \max_{y', i' < i} \; V_{\mathbf{x},W}(i', y') \\ \quad + W \cdot \mathbf{f}'(i'+1, i, \mathbf{x}, y, y') & \text{if } i > 0 \end{cases}$$

Conceptually, $V(i, y)$ is the score of the best segmentation of the first i tokens in $\mathbf{x}$ that concludes with a segment S_j such that $u_j = i$ and $\ell_j = y$.

2.4.2 Refinements to the Learning Algorithm

The SMM Viterbi search can be made more efficient if the segment size is bounded by some number L. In this case we can replace the $i' < i$ in the max term of Equation 3 to be $i' : i - L \leq i' < i$.

Perceptron-Based SMM Learning

Let $\mathbf{f}(j, \mathbf{x}, \mathbf{S})$ be a feature-vector representation of segment S_j, and let $\mathbf{F}(\mathbf{x}, \mathbf{S}) = \sum_{j=1}^{|\mathbf{S}|} \mathbf{f}(j, \mathbf{x}, \mathbf{S})$.

Let $SCORE(\mathbf{x}, W; \mathbf{S}) = W \cdot \mathbf{F}(\mathbf{x}, \mathbf{S})$.

For each each example $\mathbf{x}_t, \mathbf{S}_t$:

1. Use a modified version of Equation 3 to find the K segmentations $\hat{\mathbf{S}}_1, \ldots, \hat{\mathbf{S}}_K$ that have the highest $SCORE(\mathbf{x}_t, W_t; \hat{\mathbf{S}}_i)$.
2. Let $W_{t+1} = W_t$.
3. For each i such that $SCORE(\mathbf{x}_t, W_t; \hat{\mathbf{S}}_i)$ is greater than $(1 - \beta) \cdot SCORE(\mathbf{x}_t, W_t; \mathbf{S}_t)$, update W_{t+1} as follows:

$$W_{t+1} \leftarrow W_{t+1} + \mathbf{F}(\mathbf{x}_t, \mathbf{S}_t) - \mathbf{F}(\mathbf{x}_t, \hat{\mathbf{S}}_i)$$

As the final output of learning, return $\overline{W}$, the average of the W_t's. To segment $\mathbf{x}$ with $\overline{W}$, use Equation 3 to find the best segmentation.

Figure 1: Discriminative training for SMM's.

[1]The additional complexity is that we must learn to predict not only a tag type ℓ_j, but also the end position u_j of each segment (or equivalently, its length).

We also experimentally evaluated a number of variants of Collins' method, and obtained somewhat better performance with the following extension. As described above, the algorithm finds the single top-scoring label sequence $\hat{\mathbf{y}}$, and updates W if the score of $\hat{\mathbf{y}}$ is greater than the score of the correct sequence $\mathbf{y}$ (where the "score" of $\mathbf{y}'$ is $W \cdot \mathbf{F}(\mathbf{x}, \mathbf{y}')$). In our extension, we modified the Viterbi method to find the top K sequences $\hat{\mathbf{y}}_1, \ldots, \hat{\mathbf{y}}_K$, and then update W for all $\hat{\mathbf{y}}_i$'s with a score higher than $(1-\beta)$ times the score of $\mathbf{y}$.

The complete algorithm is shown in Figure 1. The same technique can also be used to learn HMMs, by replacing $\mathbf{S}$ with $\mathbf{y}$ and Equation 3 with Equation 2.

Like Collins' algorithm, our method works best if it makes several passes over the data. There are thus four parameters for the method: K, β, L, and E, where E is the number of "epochs" or iterations through the examples.

2.5 Features for SMMs

In a semi-Markov learner, features no longer apply to individual words, but instead are applied to hypothesized entity names. This makes it somewhat more natural to define new features, as well as providing more context.

In the notation of this paper, recall that we assumed each SMM feature function f^k can be written as $f^k(j, \mathbf{x}, \mathbf{S}) = f^k(g^k(t_j, u_j, \mathbf{x}), \ell_j, \ell_{j-1})$, where g^k is any function of t_j, u_j, and the sequence $\mathbf{x}$. Typically, g^k will compute some property of the proposed segment $\langle x_{t_j} \ldots x_{u_j} \rangle$ (or possibly of the tokens around it), and f^k will be an indicator function that couples this property with the label ℓ_j. Some concrete examples of possible g^k's are given in Table 1.

Since any of these features can be applied to one-word segments (*i.e.*, ordinary tokens), they can also be used for a HMM-like, word-tagging NER system. However, some of the features are much more powerful when applied to multi-word segments. For instance, the pattern "X+ X+" (two capitalized words in sequence) is more indicative of a person name than the pattern "X+". As another example, the "length" feature is often informative for segments.

2.6 Distance Features

Since we are no longer classifying tokens, but are instead classifying *segments* as to whether or not they correspond to complete entity names, it is straightforward to make use of similarity to words in an external dictionary as a feature.

Let D be a dictionary of entity names and d be a distance metric for entity names. Define $g_{D/d}(\mathbf{e}')$ to be the minimum distance between $\mathbf{e}'$ and any entity name $\mathbf{e}$ in D:

$$g_{D/d}(\mathbf{e}') = \min_{\mathbf{e} \in D} d(\mathbf{e}, \mathbf{e}')$$

For instance, if D contains the two strings "frederick flintstone" and "barney rubble", and d is the Jaro-Winkler distance metric [37], then $g_{D/d}(\ \langle\text{Fred}\rangle\ ) = 0.84$, and $g_{D/d}(\ \langle\text{Fred,please}\rangle\ ) = 0.4$, since $d(\text{"Fred"}, \text{"frederick flintstone"}) = 0.84$ and $d(\text{"Fred please"}, \text{"frederick flintstone"}) = 0.4$. A feature of the form $g_{D/d}$ can be trivially added to the SMM representation for any pair D and d.

One problem with distance features is that they can be relatively expensive to compute, particularly for a large dictionary. In the experiments below, we pre-processed each dictionary by building an inverted index over the character n-grams appearing in dictionary entries, for $n = 3, 4, 5$, discarding any "frequent" n-grams that appear in more than 80% of the dictionary entries. We then approximate $g_{D/d}(\mathbf{e}')$ by finding a minimum over only those dictionary entries that share a common non-frequent n-gram with $\mathbf{e}'$.

3. EXPERIMENTAL RESULTS

3.1 Baseline Algorithms

To evaluate our proposed method for learning SMMs, we compared it with the HMM-based version of the same algorithm. This is a strong baseline. In previous experimental studies, Collins' method has proved to be superior to maximum entropy CMM-based tagging methods for NER and shallow parsing, and a close competitor to conditional random fields for POS tagging and shallow parsing [2, 11, 34]. Our extension to the method performs better on four of the five NER tasks we use (and also usually gives comparable improvements to both the SMM and HMM version of the algorithm—see Section 3.5.1 below). In the experiments, we used $\beta = 0.05$, $K = 2$, and $E = 20$.

As features for the i-th token, we used a history of length one, plus the lower-cased value of the token, letter cases, and letter-case patterns (as illustrated in Figure 1) for all tokens in a window of size three centered at the i-th token. Additional dictionary-based features are described below.

We experimented with two ways of encoding NER as a word-tagging problem. The simplest method, HMM-VP$_{(1)}$, predicts two labels y: one label for tokens inside an entity, and one label for tokens outside an entity.

The second encoding scheme we used is due to Borthwick *et al* [5]. Here four tags y are associated for each entity type, corresponding to (1) a one-token entity, (2) the first token of a multi-token entity, (3) the last word of a multi-token entity, or (4) any other token of a multi-token entity. There is also a fifth tag indicating tokens that are not part of any entity. For example, locations would be tagged with the five labels Loc$_{\text{unique}}$, Loc$_{\text{begin}}$, Loc$_{\text{end}}$, Loc$_{\text{continue}}$, and Other, and a tagged example like

> (Fred)$_{\text{Person}}$, please stop by the (fourth floor meeting room)$_{\text{Loc}}$

is encoded (omitting for brevity the "Other" tags) as

> (Fred)$_{\text{Person}_{\text{unique}}}$, please stop by the (fourth)$_{\text{Loc}_{\text{begin}}}$ (floor meeting)$_{\text{Loc}_{\text{continue}}}$ (room)$_{\text{Loc}_{\text{end}}}$

We will call this scheme HMM-VP$_{(4)}$.

To add dictionary information to HMM-VP$_{(1)}$, we simply add one additional binary feature f_D which is true for every token that appears in the dictionary: *i.e.*, for any token x_i, $f_D(x_i) = 1$ if x_i matches any word of the dictionary D and $f_D(x_i) = 0$ otherwise. This feature is then treated like any other binary feature, and the training procedure assigns an appropriate weighting to it relative to the other features.

To add dictionary information to HMM-VP$_{(4)}$, we again follow Borthwick *et al* [5], who proposed using a set of four features, $f_{D.unique}$, $f_{D.first}$, $f_{D.last}$, and $f_{D.continue}$. These features are analogous to the four entity labels: for each token x_i the four binary dictionary features denote, respectively: (1) a match with a one-word dictionary entry, (2) a match with the first word of a multi-word entry, (3) a match with the last word of a multi-word entry, or, (4) a match with any other word of an entry. For example, the token x_i="flintstone" will have feature values $f_{D.unique}(x_i) = 0$, $f_{D.first}(x_i) = 0$, $f_{D.continue}(x_i) = 0$, and $f_{D.last}(x_i) = 1$ (for the dictionary D used in Table 1).

Function $g(t,u,\mathbf{x})$	Explanation	Examples
$g = \langle x_t, \ldots, x_u \rangle$	value of segment	$g(1,1,\mathbf{x})$ = “Fred” $g(2,4,\mathbf{x})$ = “please stop by”
$g = \text{lowerCase}(\langle x_t, \ldots, x_u \rangle)$	lower-cased value	$g(1,1,\mathbf{x})$ = “fred” $g(2,4,\mathbf{x})$ = “please stop by”
$g = u - t$	length of segment	$g(1,1,\mathbf{x}) = 1$ $g(2,4,\mathbf{x}) = 3$
$g = x_{t-1}$	value of left window (size 1)	$g(1,1,\mathbf{x})$ = none $g(2,4,\mathbf{x})$ = “Fred”
$g = \langle x_{u+1}, x_{u+2} \rangle$	value of right window (size 2)	$g(1,1,\mathbf{x})$ = “please stop” $g(2,4,\mathbf{x})$ = “my office”
$g = \text{translate}(\text{A-Za-z,Xx}, \langle x_t, \ldots, x_u \rangle)$	letter cases for segment	$g(1,1,\mathbf{x})$ = “Xxxx” $g(2,4,\mathbf{x})$ = “xxxxxx xxxx xx”
$g = \text{translateCompressed}(\text{A-Za-z,Xx}, \langle x_t, \ldots, x_u \rangle)$	letter-case pattern for segment	$g(1,1,\mathbf{x})$ = “X+” $g(2,4,\mathbf{x})$ = “x+ x+ x+”
$g_{D/\text{JaroWinkler}}$	Jaro-Winkler distance to dictionary	$g(1,1,\mathbf{x}) = 0.88$ $g(2,4,\mathbf{x}) = 0.45$

In examples above, $\mathbf{x} = \langle$Fred,please,stop,by,my,office,this,afternoon$\rangle$ and $D = \{$“frederick flintstone”, “barney rubble$\}$

Table 1: Possible feature functions g.

As an additional baseline NER method, we evaluated rote matching against a dictionary (*i.e.*, extracting all phrases that exactly match a dictionary entry). This approach will have low recall when the dictionary is incomplete, and cannot handle variations between the way names appear in the text and the dictionary (*e.g.*, misspellings or abbreviations). However, these results do provide an indication of the quality of the dictionary.

We note that in some cases better performance might be obtained by carefully normalizing dictionary entries. One simple normalization scheme might be to eliminate case and punctuation; more complex ones have also been used in NER [6, 7, 19]. However, just as in record linkage problems, normalization is not always desirable (*e.g.*, “Will” is more likely to be a name than “will”, and “AT-6” is more likely to be a chemical than “at 6”) and proper normalization is certainly problem-dependent. In the experiments below we do not normalize dictionary entries, except for making the match case insensitive.

As a final “baseline” use of dictionary information for HMM-VP$_{(1)}$ and HMM-VP$_{(4)}$, we extended the distance features described to tokens—*i.e.*, for each distance d, we compute as a feature of token x_i the minimum distance between x_i and an entity in the dictionary D. These features are less natural for tokens than for segments, but turned out to be surprisingly useful, perhaps because weak partial matches to entity names are informative.

To our knowledge features of this sort have not been used previously in NER tasks. We used the dictionaries described below, and three distance functions from the SecondString open source software package [9, 10]: Jaccard, Jaro-Winkler, and SoftTFIDF.

Briefly, the *Jaccard distance* between two sets S and S' is $|S \cap S'|/|S \cup S'|$: in SecondString, this is applied to strings by treating them as sets of words. The *Jaro-Winkler distance* is a character-based distance, rather than a word-based distance: it is based on the number of characters which appear in approximately the same position in both strings. TFIDF is another word-based measure. As with Jaccard distance, TFIDF scores are based on the number of words in common between two strings; however, rare words are weighted more heavily than common words. *SoftTFIDF* is a hybrid measure, which modifies TFIDF by considering words with small Jaro-Winkler distance to be common to both strings.[2]

3.2 The semi-Markov learner

Below we will use SMM-VP to denote our implementation of the algorithm of Figure 1. The parameters β, K and E are set as for HMM-VP$_{(1)}$ and HMM-VP$_{(4)}$.

Like HMM-VP$_{(1)}$, SMM-VP predicts only two label values y, corresponding to segments inside and outside an entity. We limit the length of entity segments to at most L, and limit the length of non-entity segments to 1. The value of L was set separately for each dataset to a value between 4 and 6, based on observed entity lengths.

We used the same baseline set of features that were used by HMM-VP$_{(1)}$ and HMM-VP$_{(4)}$. Additionally, for each feature used by HMM-VP$_{(1)}$, there is an indicator function that is true iff any token of the segment has that feature; an indicator function that is true iff the first token of the segment has that feature; and an indicator function that is true iff the last token of the segment has that feature. For instance, suppose one of the baseline indicator-function features used by HMM-VP$_{(1)}$ was f^{office}, where $f^{\text{office}}(i, \mathbf{x}, \mathbf{y})$ was true iff x_i has the value “office” and $y_t = i$. Then SMM-VP would also use a function $f^{\text{office,any}}(t, u, \mathbf{x})$ which would be true if any $x_i : t \leq i \leq u$ has the value 'office'; a function $f^{\text{office,first}}(t, u, \mathbf{x})$, which would be true if x_t has the value 'office'; and an analogous $f^{\text{office,last}}$. Like the 4-state output encoding, these “first” and “last” features enable SMM-VP to model token distributions that are different for different parts of an entity.

As an alternative to the distance features as described in Section 2.6, we also provided binary dictionary information to SMM-VP by introducing a binary feature that is true for a segment iff it exactly matches some dictionary entity.

[2]SoftTFIDF corresponds to the JaroWinklerTFIDF class in the SecondString code.

Dataset	# instances	# words	entity	# entities in text	words/entity in text	#dictionary entries	words/entry in dictionary
Email	216	18121	person	661	1.70	844	2.33
Jobs	300	73330	company	288	1.61	97	2.16
			title	463	2.63	156	2.64
Address	395	4226	state	87	2.31	30	1.30
			city	359	1.32	554	1.14

Table 2: Description of data, tags and dictionary used in the experiments.

3.3 Datasets

We evaluated our systems on five information extraction problems derived from three different datasets.

Address data. The Address dataset consists of 395 home addresses of students in a major university in India. The addresses in this set are much less regular than US addresses, and therefore extracting even relatively structured fields like city names is challenging [4]. We found two external dictionaries, a list of cities in India and a list of state names in India, and defined two corresponding extraction tasks: to identify city names, and to identify state names.

Email data. This dataset consists of email messages from the CSpace email corpus, which contains approximately 15,000 email messages collected from a management game conducted at Carnegie Mellon University. In this game, 277 MBA students, organized in approximately 50 teams of four to six members, ran simulated companies in different market scenarios over a 14-week period [22]. All messages sent during a one-day time period were manually tagged for person names. Person names in email headers are more regular than names in email bodies; to reduce the effect of this in our testing, we used only two header fields, the "From" field and the "Subject" field. As a dictionary, we used a list of all students who participated in the game.

Jobs data. This is a set of 300 computer-related job postings posted in the 1990's to the Austin.jobs newsgroup. These postings were manually annotated for various entities by researchers from the University of Texas [8]. Two of the annotated entities are "company names" and "job titles". To construct a dictionaries for these entities, we manually extracted company names and job titles for current hi-tech job listings in the Austin area from a large job-listing board.

In Table 2 we give a summary of the five extraction tasks, listing the number of instances, entities, and dictionary entries; the average number of words in an entity; and the average number of words in dictionary entries. One indication of the difference between the dictionary entries and the entities to be extracted is seen in the difference in the number of tokens per entity in the two cases.

3.4 Results and Discussion

The results of our initial experiments are shown in Table 3. Since many of these NER tasks can be learned rather well regardless of the feature set used, given enough data, in the table the learners are trained with only 10% of the available data, with the remainder used for testing. (We will later show results with other training set sizes.) All results reported are averaged over 7 random selections of disjoint training and test examples, and we measure accuracy in terms of the correctness of the entire extracted entity (*i.e.*, partial extraction gets no credit).

We compared each of the above NER methods on the five different tasks, without an external dictionary (first column), with an external dictionary with binary features (second column), and with an external external dictionary with distance features (third column). For each we report recall, precision and F1 values[3]. We make the following observations concerning the results of Table 3.

- Generally speaking, SMM-VP is the best-performing method, and HMM-VP$_{(1)}$ is the worst. HMM-VP$_{(4)}$ outperforms or equals HMM-VP$_{(1)}$ on 13 of the 15 cases considered (five NER problems each with no dictionary, binary dictionary features, or distance features). The two exceptions are for address-state extraction with dictionary features. Likewise, SMM-VP outperforms HMM-VP$_{(4)}$ on 13 of the 15 cases.

- Binary dictionary features are helpful, but distance-based dictionary features are more helpful. The addition of binary dictionary features improves all three learners on all five problems. Replacing binary dictionary features with distance features also improves performance for all 15 cases.

- As expected, exact matches to the external dictionary generally give low recall. Precision is also often surprisingly poor (less than 30% for the job-title task). Dictionary lookup alone is never as good as SMM-VP with distance features.

For a more concise view of the results, Table 4 summarizes the impact on performance of the two novel techniques proposed in this paper—distance-based dictionary features and semi-Markov extraction methods—and compares them to the baseline method of HMM-VP$_{(4)}$ with binary dictionary features, which we take to be representative of the previous state of the art for using dictionary features in NER. F1 is improved on all five NER tasks if the baseline is modified by either using distance features rather than binary features (the line labeled binary→distance), or by using SMM-VP rather than HMM-VP$_{(4)}$ (the line labeled HMM→SMM). SMM-VP with distance features improves F1 scores over the baseline by an average of 44.5%.

3.5 Additional Experiments

Below, we will perform a more detailed comparison of SMM-VP and HMM-VP$_{(4)}$ under various conditions. We focus on comparing the F1 performance of SMM-VP and HMM-VP$_{(4)}$ with distance features. We will not present any detailed comparisons of running times of the two methods since our implementation is not yet optimized for running

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

		Without dictionary			With dictionary					
					Binary features			Distance features		
		Recall	Prec.	F1	Recall	Prec.	F1	Recall	Prec.	F1
Address-state	lookup				32.2	100.0	48.7			
	HMM-VP$_{(1)}$	5.2	56.8	9.5	19.3	82.6	**31.3**	41.5	87.3	56.3
	HMM-VP$_{(4)}$	8.9	90.7	**16.2**	13.0	97.3	23.0	25.7	100	40.9
	SMM-VP	8.2	62.2	14.6	16.4	82.0	27.3	39.7	97.7	**56.4**
Address-city	lookup				14.8	68.8	24.3			
	HMM-VP$_{(1)}$	60.1	79.3	68.3	68.0	84.2	75.2	70.8	84	76.8
	HMM-VP$_{(4)}$	59.1	87.3	70.5	64.1	91.2	75.2	68.1	90.6	77.7
	SMM-VP	62.8	87.5	**73.1**	70.7	90.0	**79.2**	72.2	89.4	**79.9**
Email-person	lookup				38.7	82.6	57.3			
	HMM-VP$_{(1)}$	60.4	74.9	66.8	73.4	83.7	78.2	79.1	84.6	81.8
	HMM-VP$_{(4)}$	60.9	80.2	69.3	71.1	87.6	78.5	77.1	89.2	82.7
	SMM-VP	64.1	80.3	**71.3**	77.7	88.1	**82.6**	78.9	88.5	**83.4**
Job-company	lookup				14.1	54.8	22.3			
	HMM-VP$_{(1)}$	1.3	34.7	2.5	2.0	28.1	3.8	8.9	79.8	16.1
	HMM-VP$_{(4)}$	3.6	59.8	6.8	11.5	80.6	20.2	18.6	93.4	**31.1**
	SMM-VP	5.2	55.3	**9.6**	13.8	85.4	**23.7**	17.8	95.9	30.0
Job-title	lookup				29.4	29.5	29.4			
	HMM-VP$_{(1)}$	18.4	43.7	25.9	23.9	43.2	30.8	30.9	44.2	36.4
	HMM-VP$_{(4)}$	17.3	51.5	25.9	27.9	48.4	35.4	30.9	45.7	36.8
	SMM-VP	20.9	52.0	**29.8**	34.9	48.8	**40.7**	36.2	47.9	**41.2**

Table 3: Performance of NER methods on five IE tasks under three conditions: with no external dictionary; with an external dictionary and binary features; with an external dictionary and distance features.

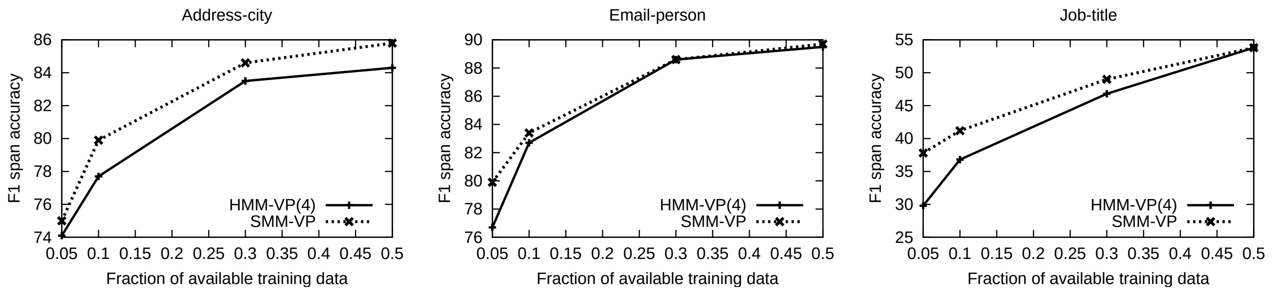

Figure 2: Comparing SMM-VP and HMM-VP$_{(4)}$ with changing training set size on IE tasks from three domains. The X-axis is the fraction of examples used for training and the Y-axis is field-level F1.

	Address		Email	Job	
	State	City	Person	Co.	Title
baseline method	23.0	75.2	78.5	20.2	35.4
+ binary→distance	40.9	77.7	82.7	31.1	36.8
+ HMM→SMM	27.3	79.2	82.6	23.7	40.7
+ both changes	56.4	79.9	83.4	30.0	41.2

Table 4: Summary of improvements in F1 measure over the baseline method of HMM-VP$_{(4)}$ with binary dictionary features.

		Address		Email	Job	
		State	City	Person	Co.	Title
HMM-VP$_{(4)}$	Collins	34.6	76.3	74.9	**56.1**	32.5
	C & S	**40.9**	**77.7**	**82.7**	31.1	**36.8**
SMM-VP	Collins	49.1	78.2	78.1	**53.0**	33.9
	C & S	**56.4**	**79.9**	**83.4**	30.0	**41.2**

Table 5: F1 performance of the voted perceptron variant considered here *vs* the method described by Collins.

time. (As implemented, the SMM-VP method is 3-5 times slower than HMM-VP$_{(4)}$, because of the more expensive distance features and the expanded Viterbi search.)

3.5.1 *Effect of Extensions to Collins' Method*

Table 5 compares the F1 performance of (our implementation of) Collins' original method (labeled Collins) to our variant (labeled C & S). We also compare the natural semi-Markov extension of Collins' method to SMM-VP. In both cases Collins' method performs much better on one of the five problems, but worse on the remaining four. The changes in performance associated with our extension seem to affect both the Markovian and semi-Markovian versions of the algorithm similarly. In none of the five tasks does the change from Collins' original method to our variant change the relative order of the two methods.

	History	Recall	Prec.	F1
Address-state				
HMM-VP$_{(4)}$	1	25.7	100	40.9
	2	23.2	100	37.7
	3	24.7	100	39.6
SMM-VP	1	39.7	97.7	**56.4**
Address-city				
HMM-VP$_{(4)}$	1	68.1	90.6	77.7
	2	68.5	90.8	78.1
	3	68.4	90.7	78.0
SMM-VP	1	72.2	89.4	**79.9**
Email-person				
HMM-VP$_{(4)}$	1	77.1	89.2	82.7
	2	77.0	88.6	82.4
	3	77.0	88.7	82.4
SMM-VP	1	78.9	88.5	**83.4**

Table 6: Effect of increasing history size of HMM-VP$_{(4)}$ on F1 performance, compared to F1 performance of SMM-VP.

3.5.2 Effect of training set size

In Figure 2 we show the impact of increasing training set size on HMM-VP$_{(4)}$ and SMM-VP on three representative NER tasks. Often when the training size is small, SMM-VP is much better than HMM-VP$_{(4)}$, but when the training size increases, the gap between the two methods narrows. This suggests that the semi-Markov features are less important when large amounts of training data are available. However, the amount of data needed for the two methods to converge varies substantially, as is illustrated by the curves for address-city and email-person.

3.5.3 Effect of history size

It is straightforward to extend the algorithms of this paper so that HMM-VP$_{(4)}$ can construct features that rely on the last several predicted classes, instead of only the last class. Table 6 we show the result of increasing the "history size" for HMM-VP$_{(4)}$ from one to three. We find that the performance of HMM-VP$_{(4)}$ does not improve much with increasing history size, and in particular, that increasing history size does not change the relative ordering of HMM-VP$_{(4)}$ and SMM-VP. This result supports the claim, made in Section 2.3, that a SMM-VP with segment length bounded by L is quite different from an order-L HMM.

3.5.4 Alternative dictionaries

We re-ran two of our extraction problems on alternative dictionaries to study sensitivity to dictionary quality. For emails we used a dictionary of 16623 student names, obtained from students at universities across the country as part of the RosterFinder project [36]. For the job-title extraction task, we obtained a dictionary of 159 job titles in California from a software jobs website[4]. Recall that the original email dictionary contained the names of the people who sent the emails, and the original dictionary for the Austin-area job postings was for jobs in the Austin area.

Table 7 shows the result, for HMM-VP$_{(4)}$ and SMM-VP with distance features. Both methods seem fairly robust to using dictionaries of less-related entities. Although the quality of extractions is lowered for both methods in three of the four cases, the performance changes are not large.

[4]`http://www.softwarejobs.com`

4. RELATED WORK

Besides the methods described in Section 3.1 for integrating a dictionary with NER systems [5, 7], a number of other techniques have been proposed for using dictionary information in extraction.

A method of incorporating an external dictionary for generative models like HMMs is proposed in [33, 4]. Here a dictionary was treated as a collection of training examples of emissions for the state which recognizes the corresponding entity: for instance, a dictionary of person names would be treated as example emissions of a "person name" state. 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 concurrent work by one of these authors, a scheme is proposed for compiling a dictionary into a very large HMM in which emission and transition probabilities are highly constrained, so that the HMM has very few free parameters. This approach suffers from many of the limitations described above, but may be useful when training data is limited.

Krauthammer *et al* [23] describe an edit-distance based scheme for finding partial matches to dictionary entries in text. Their scheme uses BLAST (a high-performance tool designed for DNA and protein sequence comparisons) to do the edit distance computations. However, there is no obvious way of combining edit-distance information with other informative features, as there is in our model. In experimental studies, pure edit-distance based metrics are often not the best performers in matching names [10]; this suggests that it may be advantageous in NER to be able to exploit other types of distance metrics as well as edit distance.

Some early NER systems used a "sliding windows" approach to extraction, in which all word n-grams were classified as "entities" or "non-entities" (for n of some bounded size) (*e.g.*, [16]). Such systems can easily be extended to make use of dictionary-based features. However, in prior experimental comparisons, sliding-window NER system have usually proved inferior to HMM-like NER systems. Sliding window approaches also have the disadvantage that they may extract entities that overlap.

Another mechanism of exploiting a dictionary is to use it to bootstrap a search for extraction patterns from unlabeled data [1, 12, 14, 31, 38]. In these systems, dictionary entries are matched on unlabeled instances to provide "seed" positive examples, which are then used to learn extraction patterns that provide additional entries to the dictionary. These extraction systems are mostly rule-based (with some exceptions [12, 14]), and appear to assume a relatively clean set of extracted entities. In contrast our focus is probabilistic models and the incorporation of large noisy dictionaries.

To our knowledge, semi-Markov models have not been previously been used for information extraction, although they have been used in other domains [18, 35]. To our knowledge, the SMM with dictionaries is also the first method that can combine arbitrary similarity measures on multi-word segments with a Markovian, HMM-like extraction-learning algorithm. In future work, we plan to explore adapting

	Recall	Prec.	F1	Recall	Prec.	F1	Recall	Prec.	F1
Email-person	No dictionary			Original dictionary			Student names		
Dictionary lookup				43.11	85.3	57.3	0	0	0
HMM-VP$_{(4)}$	60.9	80.2	69.3	77.1	89.2	82.7	73.5	86.3	**79.4**
SMM-VP	64.1	80.3	**71.3**	78.9	88.5	**83.4**	74.8	84.6	**79.4**
Job-title	No dictionary			Original (Austin job titles)			California job titles		
Dictionary lookup				29.4	29.5	29.4	4.3	27.0	7.2
HMM-VP$_{(4)}$	17.3	51.5	25.9	30.9	45.7	36.8	26.8	47.5	34.3
SMM-VP	20.9	52.0	**29.8**	36.2	47.9	**41.2**	36.0	51.9	**42.5**

Table 7: Results with changing dictionary.

the semi-Markov NER learning algorithms discussed here to conditional random fields [24, 34].

5. CONCLUSIONS

In many cases, the ultimate goal of an information extraction process is to answer queries which combine information from structured and unstructured sources. In these applications, NER is successful only to the extent that it finds entity names that can be matched to something in a pre-existing database. However, extending state-of-the-art NER systems by incorporating an external dictionary is difficult. In particular, incorporating information about the similarity of extracted entities to dictionary entries is awkward, because the best NER systems operate by sequentially classifying *words* as to whether or not they participate in an entity name, while the best similarity measures score entire candidate names.

To correct this mismatch we relax the usual Markov assumptions, and formalize a semi-Markov extraction process. This process is based on sequentially classifying *segments* of several adjacent words, rather than single words. In addition to allowing a way of coupling high-performance NER methods and high-performance record linkage metrics, this formalism also allows the direct use of other useful entity-level features (such as the length of entity). It also provides an arguably more natural formulation of the NER problem than sequential word classification. For instance, in the usual formulation, one must design a new set of output tags (and make a corresponding change in the tag-to-entity decoding scheme) to account for distributional differences between words from the beginning of an entity and words elsewhere in an entity. In the semi-Markov formulation, one merely adds new features for entity-beginning words.

We compared our proposed algorithm to a strong baseline NER, which uses Collins' perceptron-based algorithm for training an HMM and a state-of-the art, multi-label encoding for dictionary information. The new algorithm is surprisingly effective: on our datasets, it always outperforms the previous baseline, sometimes dramatically.

Acknowledgments

The authors thank the reviewers for this paper for a number of substantive comments which greatly improved the final version. The preparation of this paper was supported in part funded by grants from the Information Processing Technology Office (IPTO) of the Defense Advanced Research Projects Agency (DARPA), as well as National Science Foundation Grant No. EIA-0131884 to the National Institute of Statistical Sciences, and a contract from the Army Research Office to the Center for Computer and Communications Security (CyLab) at Carnegie Mellon University.

6. REFERENCES

[1] E. Agichtein and L. Gravano. Snowball: Extracting relations from large plaintext collections. In *Proceedings of the 5th ACM International Conference on Digital Libraries*, 2000.

[2] Y. Altun, I. Tsochantaridis, and T. Hofmann. Hidden markov support vector machines. In *Proceedings of the 20th International Conference on Machine Learning (ICML)*, 2003.

[3] D. M. Bikel, R. L. Schwartz, and R. M. Weischedel. An algorithm that learns what's in a name. *Machine Learning*, 34:211–231, 1999.

[4] 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.

[5] 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.

[6] R. Bunescu, R. Ge, R. J. Kate, E. M. Marcotte, R. J. Mooney, A. K. Ramani, and Y. W. Wong. Learning to extract proteins and their interactions from medline abstracts. Available from http://www.cs.utexas.edu/users/ml/publication/ie.html, 2002.

[7] R. Bunescu, R. Ge, R. J. Mooney, E. Marcotte, and A. K. Ramani. Extracting gene and protein names from biomedical abstracts. Unpublished Technical Note, Available from http://www.cs.utexas.edu/users/ml/publication/ie.html, 2002.

[8] M. E. Califf and R. J. Mooney. Bottom-up relational learning of pattern matching rules for information extraction. *Journal of Machine Learning Research*, 4:177–210, 2003.

[9] W. W. Cohen and P. Ravikumar. Secondstring: An open-source Java toolkit of approximate string-matching techniques. Project web page, http://secondstring.sourceforge.net, 2003.

[10] 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.

[11] M. Collins. Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms. In *Empirical Methods in Natural Language Processing (EMNLP)*, 2002.

[12] M. Collins and Y. Singer. Unsupervised models for named entity classification. In *Proceedings of the Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora (EMNLP99)*, College Park, MD, 1999.

[13] K. Crammer and Y. Singer. Ultraconservative online algorithms for multiclass problems. *J. Mach. Learn. Res.*, 3:951–991, 2003.

[14] M. Craven and J. Kumlien. Constructing biological knowledge bases by extracting information from text sources. In *Proceedings of the 7th International Conference on Intelligent Systems for Molecular Biology (ISMB-99)*, pages 77–86. AAAI Press, 1999.

[15] R. Durban, S. R. Eddy, A. Krogh, and G. Mitchison. *Biological sequence analysis - Probabilistic models of proteins and nucleic acids*. Cambridge University Press, Cambridge, 1998.

[16] D. Freitag. Multistrategy learning for information extraction. In *Proceedings of the Fifteenth International Conference on Machine Learning*. Morgan Kaufmann, 1998.

[17] Y. Freund and R. E. Schapire. Large margin classification using the perceptron algorithm. In *Computational Learing Theory*, pages 209–217, 1998.

[18] X. Ge. *Segmental Semi-Markov Models and Applications to Sequence Analysis*. PhD thesis, University of California, Irvine, December 2002.

[19] D. Hanisch, J. Fluck, H. Mevissen, and R. Zimmer. Playing biology's name game: identifying protein names in scientific text. In *Pac Symp Biocomput*, pages 403–14, 2003.

[20] K. Humphreys, G. Demetriou, and R. Gaizauskas. Two applications of information extraction to biological science journal articles: Enzyme interactions and protein structures. In *Proceedings of 2000 the Pacific Symposium on Biocomputing (PSB-2000)*, pages 502–513, 2000.

[21] D. Klein and C. D. Manning. Conditional structure versus conditional estimation in nlp models. In *Workshop on Empirical Methods in Natural Language Processing (EMNLP)*, 2002.

[22] R. E. Kraut, S. R. Fussell, F. J. Lerch, and J. A. Espinosa. Coordination in teams: evi-dence from a simulated management game. To appear in the Journal of Organizational Behavior, 2004.

[23] M. Krauthammer, A. Rzhetsky, P. Morozov, and C. Friedman. Using blast for identifying gene and protein names in journal articles. *Gene*, 259(1-2):245–52, 2000.

[24] 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.

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

[26] N. Littlestone. Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. *Machine Learning*, 2(4), 1988.

[27] R. Malouf. Markov models for language-independent named entity recognition. In *Proceedings of the Sixth Conference on Natural Language Learning (CoNLL-2002)*, 2002.

[28] A. McCallum, D. Freitag, and F. Pereira. Maximum entropy markov models for information extraction and segmentation. In *Proceedings of the International Conference on Machine Learning (ICML-2000)*, pages 591–598, Palo Alto, CA, 2000.

[29] A. K. McCallum, K. Nigam, J. Rennie, , and K. Seymore. Automating the construction of internet portals with machine learning. *Information Retrieval Journal*, 3:127–163, 2000.

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

[31] E. Riloff and R. Jones. Learning Dictionaries for Information Extraction by Multi-level Boot-strapping. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence*, pages 1044–1049, 1999.

[32] S. Sarawagi and A. Bhamidipaty. Interactive deduplication using active learning. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, July 23-26, 2002, Edmonton, Alberta, Canada*. ACM, 2002.

[33] 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.

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

[35] R. S. Sutton. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *Proceedings of the Seventh International Conference on Machine Learning*, Austin, Texas, 1990. Morgan Kaufmann.

[36] L. Sweeney. Finding lists of people on the web. Technical Report CMU-CS-03-168, CMU-ISRI-03-104, Carnegie Mellon University School of Computer Science, 2003. Available from http://privacy.cs.cmu.edu/dataprivacy/projects/rosterfinder/.

[37] W. E. Winkler. Matching and record linkage. In *Business Survey methods*. Wiley, 1995.

[38] R. Y. Winston Lin and R. Grishman. Bootstrapped learning of semantic classes from positive and negative examples. In *Proceedings of the ICML Workshop on The Continuum from Labeled to Unlabeled Data, Washington, D.C*, August 2003.