

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*
School of Information Technology
IIT Bombay
Powai 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 relaxes the usual Markov assumptions. This process is based on sequentially classifying *segments* of several adjacent words, rather than single words. In addition to allowing a natural way of coupling NER and high-performance record linkage methods, 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, relative to previously published methods for using external dictionaries in NER.

Keywords

Learning, information extraction, data integration, sequential learning

1. INTRODUCTION

Named entity recognition (NER) is the identification of names of entities in text. Well-studied cases of NER are identifying personal names and company names in newswire

*Both authors contributed equally to this research.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

KDD '04 Seattle

Copyright 200X ACM X-XXXXX-XX-X/XX/XX ...\$5.00.

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 a 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—that is, the 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 a particular 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 the 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

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, analyze its performance, 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)_{Person} please stop by (my office)_{Loc} (this afternoon)_{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, \dots, x_M , and a *tagger* associates with \mathbf{x} a parallel sequence of tags $\mathbf{y} = y_1, \dots, y_M$. 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 “Person” the probability that y_{i+1} is “Person” is increased, and the probability that y_{i+2} is “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

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

(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 $P(y_i|y_{i-1}, x_i)$ can be learned from derived triples (y_i, y_{i-1}, x_i) , 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 x are observed. Following work in semi-Markov decision processes [35, 18] we will call this a conditional semi-Markov model (CSMM).

For notation, let $\mathbf{S} = \langle S_1, \dots, S_K \rangle$ be a “segmentation” of \mathbf{x} . Also let $M = |\mathbf{x}|$. 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 here—or equivalently, we use t_j simply as a notational convenience. We will use \mathbf{x}_{S_j} to denote the subsequence $\langle x_{t_j} \dots x_{u_j} \rangle$.

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 $y_{t_j} \dots y_{u_j}$ corresponding to $x_{t_j} \dots x_{u_j}$ 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 $\ell_{J(\mathbf{S}, 1)}, \dots, \ell_{J(\mathbf{S}, M)}$.

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

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

$$\begin{aligned} P(\mathbf{S}|\mathbf{x}) &= \prod_j P(S_j | \ell_{j-1}, \mathbf{x}_{S_j}) \\ &= \prod_j P(u_j, \ell_j | \ell_{j-1}, \langle x_{u_{j-1}+1}, \dots, x_{u_j} \rangle) \end{aligned} \quad (1)$$

More generally, we will call any model in which each S_j is independent of all $S_{j'}$ other than S_{j-1} and S_{j+1} a *semi-Markov model* (SMM).

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 $O(LM|Y|)$, where M 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 $L \leq M$, 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 $P(S_j | \ell_{j-1}, \mathbf{x}_{S_j})$ 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: somewhat better performance can often be obtained by algorithms that learn a single global conditional model for $P(\mathbf{y}|\mathbf{x})$ [11, 24]. Below we will also present an extension of one such “global” conditional 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 HMM, as in an order- L HMM, 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 the experimental section we will show that SMMs improve over CMM-like extraction models that use substantial histories and token windows.

2.4 Discriminative Training for SMMs

The learning algorithm we use for training SMMs is based on 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

$$\mathbf{F}(\mathbf{x}, \mathbf{y}) = \sum_i^M \mathbf{f}(i, \mathbf{x}, \mathbf{y})$$

and let W be a weight vector over the components of \mathbf{F} . The global conditional probability $P(\mathbf{y}|\mathbf{x})$ can then be expressed as

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

During inference we need to compute $V(W, \mathbf{x})$ the Viterbi decoding of \mathbf{x} with W , i.e.

$$V(W, \mathbf{x}) = \operatorname{argmax}_{\mathbf{y}} \mathbf{F}(\mathbf{x}, \mathbf{y}) \cdot W$$

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:

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

¹The 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).

and then $V(W, \mathbf{x}) = \max_y V_{\mathbf{x}, \mathbf{f}, W}(|\mathbf{x}|, y)$.

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 $(\mathbf{x}_t, \mathbf{y}_t)$. This is analogous to finding a W that maximizes $Pr(\mathbf{y}|\mathbf{x})$ on the training data using stochastic gradient descent.

Specifically, Collins’ 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, and W_t is replaced with

$$\begin{aligned} W_{t+1} &= W_t + \mathbf{F}(\mathbf{x}_t, \mathbf{y}_t) - \mathbf{F}(\mathbf{x}_t, \hat{\mathbf{y}}_t) \\ &= W_t + \sum_{i=1}^M \mathbf{f}(i, \mathbf{x}_t, \mathbf{y}_t) - \mathbf{f}(i, \mathbf{x}_t, \hat{\mathbf{y}}_t) \end{aligned} \quad (3)$$

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 formally proved to converge under certain plausible assumptions [11].

We will consider a slight variant of Collins’ algorithm which is more amenable to extension to SMMs. Notice that in Equation 3, any indices i for which $\mathbf{f}(i, \mathbf{x}_t, \mathbf{y}_t) = \mathbf{f}(i, \mathbf{x}_t, \hat{\mathbf{y}}_t)$ will cancel and have no effect. Letting I_E be the set of indices i such that $\mathbf{f}(i, \mathbf{x}_t, \mathbf{y}_t) \neq \mathbf{f}(i, \mathbf{x}_t, \hat{\mathbf{y}}_t)$, the update of Equation 3 is the same as would be performed for a perceptron learning a concept over the space of outputs of \mathbf{f} , if for each I_E , $\mathbf{f}(i, \mathbf{x}_t, \mathbf{y}_t)$ was a misclassified positive example and $\mathbf{f}(i, \mathbf{x}_t, \hat{\mathbf{y}}_t)$ was a misclassified negative example.

This suggests the following variant of Collins’ algorithm. Define

$$T_{\text{TAG}} = \{(\mathbf{x}, \mathbf{y}) : \mathbf{y} \text{ is correct tagging of } \mathbf{x}\}$$

Consider now the set

$$T_{\text{TRANS}} = \{(i, \mathbf{x}, \mathbf{y}) : (\mathbf{x}, \mathbf{y}) \in T_{\text{TAG}} \text{ and } 0 \leq i \leq |\mathbf{x}|\}$$

which might be informally called the set of “transitions” in a correct tagging of \mathbf{x} .

Now consider the algorithm of Figure 1. It is clear that if \mathcal{L} can successfully learn the concept T_{TRANS} then Algorithm 1 will eventually converge to a H_t such that Step 2 will always produce the correct tagging.

Notice that if \mathcal{L} is a voted perceptron (and hence, H_t is a weight vector W_t) then Algorithm 1 is extremely similar to Collins’ algorithm.

It remains to show an implementation of Step 2. As a practical matter, it is probably better to use some “best” $\hat{\mathbf{y}}$ that satisfies the constraints, instead of an arbitrary one, so we will consider finding the sequence that maximizes² $\sum_i H_t(\mathbf{f}_i)$. This is the same maximization done by the Viterbi function $V(W, \mathbf{x})$ of Collins’ algorithm. Step 2 of Algorithm 1 can be implemented with the recurrence of Equation 3: it is only necessary to replace $W \cdot \mathbf{f}'(i, \mathbf{x}, y, y')$ with $H_t(\mathbf{f}'(i, \mathbf{x}, y, y'))$.

The natural extension of Algorithm 1 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

²Notice that if the online learner \mathcal{L} can be adjusted so that log-odds of membership in T_{TRANS} are produced (i.e., so that $H_t(\mathbf{f}) \approx \log Pr(\mathbf{f} \in T_{\text{TRANS}})$), then the sequence that maximizes $\sum_i H_t(\mathbf{f}_i)$ also maximizes the probability that $\forall i, \mathbf{f}(i, \mathbf{x}_t, \mathbf{y}_t) \in T_{\text{TRANS}}$.

Algorithm 1

Let \mathcal{L} be a on-line learning algorithm for the concept T_{TRANS} , and let $\mathbf{f}(i, \mathbf{x}, \mathbf{y})$ be a feature-vector representation of a single “transition”.

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

1. Let H_t be \mathcal{L} 's current hypothesis for T_{TRANS} .
2. Find some sequence $\hat{\mathbf{y}}_t$ such that

$$\forall i, \mathbf{f}(i, \mathbf{x}_t, \hat{\mathbf{y}}_t) \in H_t$$

3. For each i such that $\mathbf{f}(i, \mathbf{x}_t, \hat{\mathbf{y}}_t) \neq \mathbf{f}(i, \mathbf{x}_t, \mathbf{y}_t)$,
 - (a) submit $\mathbf{f}(i, \mathbf{x}_t, \mathbf{y}_t)$ to \mathcal{L} as a positive example,
 - (b) submit $\mathbf{f}(i, \mathbf{x}_t, \hat{\mathbf{y}}_t)$ to \mathcal{L} as a negative example.

Figure 1: A simple discriminative training algorithm for HMMs

can be computed for any segment S_j of a proposed segmentation \mathbf{S} , i.e. we assume a function $\mathbf{f}(j, \mathbf{x}, \mathbf{S})$. Define

$$T_{\text{SEGTRANS}} = \{ \langle j, \mathbf{x}, \mathbf{S} \rangle : \mathbf{S} \text{ is the correct segmentation of } \mathbf{x} \}$$

The natural extension of Algorithm 1 is then given in Figure 2. It is the same as Algorithm 1, except that the on-line learner \mathcal{L} learns the concept T_{SEGTRANS} , and that the Viterbi search is more complex. For SMM Viterbi search, we need to restrict each f^k to be of the form

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

As before, we let $\mathbf{f}'(t, u, \mathbf{x}, y, y')$ be the vector of f^k 's and define the recurrence:

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

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 $\ell_j = y$. By the semi-Markov property, this score depends only on the tag ℓ_{j-1} associated with the previous segment S_{j-1} (the y' in the max) and the start and end indices of S_j (the indices $i' + 1$ and i in the max).

For efficiency, if segment size is bounded by L then we can replace the $i' < i$ in the max term to be $i' : i - L \leq i' < i$. In our experiments, we do this. We also assume exactly two values for Y , an “extract this” tag ($y = 1$) and an “other” tag ($y = 0$). We limit the length of $y = 1$ segments to at most L , and limit the length of $y = 0$ segments to 1, which also makes the segmentation unique given a tagging.

Notice that $(j, \mathbf{x}, \mathbf{S})$ has to be part of a correct segmentation (of which there is only one). This leads to a different classification problem from the one used in Algorithm 1. The learner is no longer classifying tokens as to whether they're part of a correct tagging—it is classifying *segments* as to whether or not they correspond to *complete entity names*. In addition to having more context than before, this is a somewhat more natural and a more easily-grasped concept, and perhaps one for which it is easier to develop useful features.

Algorithm 2

Let \mathcal{L} be a on-line learning algorithm for the concept T_{SEGTRANS} , and let $\mathbf{f}(i, \mathbf{x}, \mathbf{S})$ be a feature-vector representation of a single “segment transition”.

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

1. Let H_t be \mathcal{L} 's current hypothesis for T_{SEGTRANS} .
2. Find some sequence $\hat{\mathbf{S}}_t$ such that

$$\forall j, \mathbf{f}(j, \mathbf{x}_t, \hat{\mathbf{S}}_t) \in H_t$$

3. For each j such that $\mathbf{f}(j, \mathbf{x}_t, \hat{\mathbf{S}}_t) \neq \mathbf{f}(j, \mathbf{x}_t, \mathbf{S}_t)$,
 - (a) submit $\mathbf{f}(j, \mathbf{x}_t, \mathbf{S}_t)$ to \mathcal{L} as a positive example,
 - (b) submit $\mathbf{f}(j, \mathbf{x}_t, \hat{\mathbf{S}}_t)$ to \mathcal{L} as a negative example.

Figure 2: A discriminative training algorithm for SMM's

2.5 Features for SMMs

Recall that $\mathbf{f}(j, \mathbf{x}, \mathbf{S})$ is a feature vector representation of the j -th segment of \mathbf{x} in the context of the segmentation \mathbf{S} , and $f^k(j, \mathbf{x}, \mathbf{S})$ is the k -th component of this vector. Recall that for efficiency we assumed each 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} . Often g^k will compute some hopefully-informative property of the proposed segment \mathbf{x}_{S_j} (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 traditional tagger-based 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 very informative for segments.

2.6 Dictionary 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 an external dictionary as a feature.

Letting D be a dictionary of entity names and d be a distance metric for entity names, we 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 $d(\langle \text{Fred} \rangle, \langle \text{frederick flintstone} \rangle) = 0.84$ and $d(\langle \text{Fred please} \rangle, \langle \text{frederick flintstone} \rangle) = 0.4$. A feature of the form $g_{D/d}$ can be trivially added to the SMM representation for any distance metric 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 charac-

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

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

Table 1: Possible feature functions g

ter n -grams appearing in dictionary entries, for $n = 3, 4, 5$, discarding any n -grams that are very common (i.e., 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 n -gram with \mathbf{e}' .

3. EXPERIMENTAL RESULTS

3.1 Baseline Algorithms

To evaluate Algorithm 2, we compared it with several variants of Collins’ perceptron-based algorithm for discriminatively training HMMs [11], on which Algorithm 1 is based. This is a strong baseline: in previous experimental studies [2, 11, 34], it 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. As features of the i -th token, all variants use a history of length one, plus the lower-cased value, letter cases, and letter-case pattern (as illustrated in Figure 1) for all tokens in a window of size three centered at the i -th token.

HMM-VP/1. The simplest variant, HMM-VP/1, uses only these baseline features, and predicts two labels y —one label for tokens inside an entity, and one label for tokens outside an entity. To provide dictionary information, 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.

HMM-VP/4. A more sophisticated scheme for NER tagging, and for use of dictionary information, was proposed by Borthwick *et al* [5]. In this scheme, 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 $\text{LOC}_{\text{unique}}$, $\text{LOC}_{\text{begin}}$, LOC_{end} , $\text{LOC}_{\text{continue}}$, and Other , and a tagged example like

$(\text{Fred})_{\text{Person}}$, please stop by the (fourth floor meet-
ing room) $_{\text{Loc}}$

is encoded (omitting for brevity the “Other” tags) as

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

Below we will call this variant HMM-VP/4. In our experiments (and those of others [6]) HMM-VP/4 generally performs better than HMM-VP/1.

To add dictionary information to HMM-VP/4, we again follow Borthwick *et al* [5] and use a set of four features, $f_{D.\text{unique}}$, $f_{D.\text{first}}$, $f_{D.\text{last}}$, and $f_{D.\text{continue}}$. 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 $x_i = \text{"flintstone"}$ will have feature values $f_{D.\text{unique}}(x_i) = 0$, $f_{D.\text{first}}(x_i) = 0$, $f_{D.\text{continue}}(x_i) = 0$, and $f_{D.\text{last}}(x_i) = 1$ (for the dictionary D used in Table 1).

Dictionary lookup. As an additional baseline, we evaluate rote matching against a dictionary—i.e., the simple NER that extracts all phrases exactly matching a dictionary entry.

This approach will clearly have low recall when the dictionary is incomplete. Also, it cannot handle even minor variations between the way names appear in the text and the dictionary (e.g., misspellings or abbreviations). However, these results do provide some indication of the quality of the dictionaries we used.

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.

3.2 The semi-Markov learner

In our implementation of Algorithm 2, we used a voted perceptron [17] as the on-line learner. This choice was made largely to facilitate comparison with the baseline algorithms: we suspect that other on-line learners (e.g., [26, 13]) would also be effective. Below we call this algorithm SMM-VP.

Like HMM-VP/1, SMM-VP predicts only two label values y , corresponding to segments inside and outside an entity. We used the same baseline set of features as 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 had that feature; an indicator function that is true iff the first token of the segment had that feature; and an indicator function that is true iff the last token of the segment had 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.

To provide dictionary information to SMM-VP, we used the features $f_{D/d}$ for the distance functions SoftTFIDF³, Jaro-Winkler, and Jaccard [10] from the SecondString open source software package [9]. The value L of the maximum segment was set to 3.

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 somewhat 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.

³SoftTFIDF corresponds to the JaroWinklerTFIDF class in the SecondString code.

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 along with various statistics like 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. An indication of the potential 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 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. We will later show results with other training set sizes.

We compared each of the above NER methods on the five different tasks, with and without an external dictionary. The first column of numbers show the recall, precision and F1 values⁴ without an external dictionary and the last column shows the same method with the external dictionary. All numbers reported are by averaging over 7 random selections of disjoint training and test examples. The test set is selected to be 30% of the available data. We measure accuracy in terms of the correctness of the entire extracted entity (i.e., partial extraction gets no credit).

We make the following observations concerning these results.

- HMM-VP/4 is always better than HMM-VP/1, both with and without a dictionary. Also, the external dictionary alone is never as good as the best NER system that makes use of dictionary information. Not surprisingly, this is mostly due to low recall.
- The addition of an external dictionary improves accuracy for all learners (with one exception for HMM-VP/1). The magnitude of improvement over the non-dictionary version seems to be higher in tasks where the dictionary lookup method works better: e.g., for Address_City, the dictionary boosts accuracy less than for Address_State. The exception to this rule is the Jobs_company dictionary, which does not improve extraction performance by much. This may be explained by the table below, where for each dataset we show the average distance of all extraction entities to the closest dictionary entry: according to this metric, the Jobs_company dictionary is much noisier than is suggested by the lookup method’s performance.

Task	Avg.distance
Address_state	0.825
Address_city	0.743
Email_persons	0.743
Jobs_company	0.406
Jobs_title	0.760

⁴F1 is defined as $2 * \text{precision} * \text{recall} / (\text{precision} + \text{recall})$.

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 title	288 463	1.61 2.63	97 156	2.16 2.64
Address	395	4226	state city	87 359	2.31 1.32	30 554	1.30 1.14

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

		Without dictionary			With dictionary		
		Recall	Precision	F1	Recall	Precision	F1
Address_state	Dictionary lookup				32.2	100.0	48.7
	HMM-VP/1	8.5	32.6	13.4	23.6	59.3	33.7
	HMM-VP/4	8.7	42.5	14.4	13.2	70.4	22.3
	SMM-VP	15.3	62.1	24.5	48.8	71.4	58.0
Address_city	Dictionary lookup				14.8	68.8	24.3
	HMM-VP/1	61.8	75.6	68.0	69.3	81.0	74.7
	HMM-VP/4	61.3	82.3	70.3	68.5	86.2	76.4
	SMM-VP	61.9	75.1	67.9	73.7	84.5	78.8
Email_persons	Dictionary lookup				38.7	82.6	57.3
	HMM-VP/1	52.3	69.4	59.7	62.9	74.2	68.1
	HMM-VP/4	52.0	82.6	63.8	64.4	84.7	73.1
	SMM-VP	57.8	66.6	61.9	71.5	78.0	74.6
Jobs_company	Dictionary lookup				14.1	54.8	22.3
	HMM-VP/1	33.0	40.5	36.4	29.7	39.7	34.0
	HMM-VP/4	34.1	40.5	37.0	35.5	42.9	38.8
	SMM-VP	37.8	70.0	49.0	41.7	67.6	51.6
Jobs_title	Dictionary lookup				29.4	29.5	29.4
	HMM-VP/1	9.8	8.2	8.9	13.8	8.2	10.3
	HMM-VP/4	15.3	10.9	12.7	19.9	10.6	13.8
	SMM-VP	22.1	38.9	28.2	33.0	44.9	38.0

Table 3: Comparing various methods on five IE tasks with and without an external dictionary

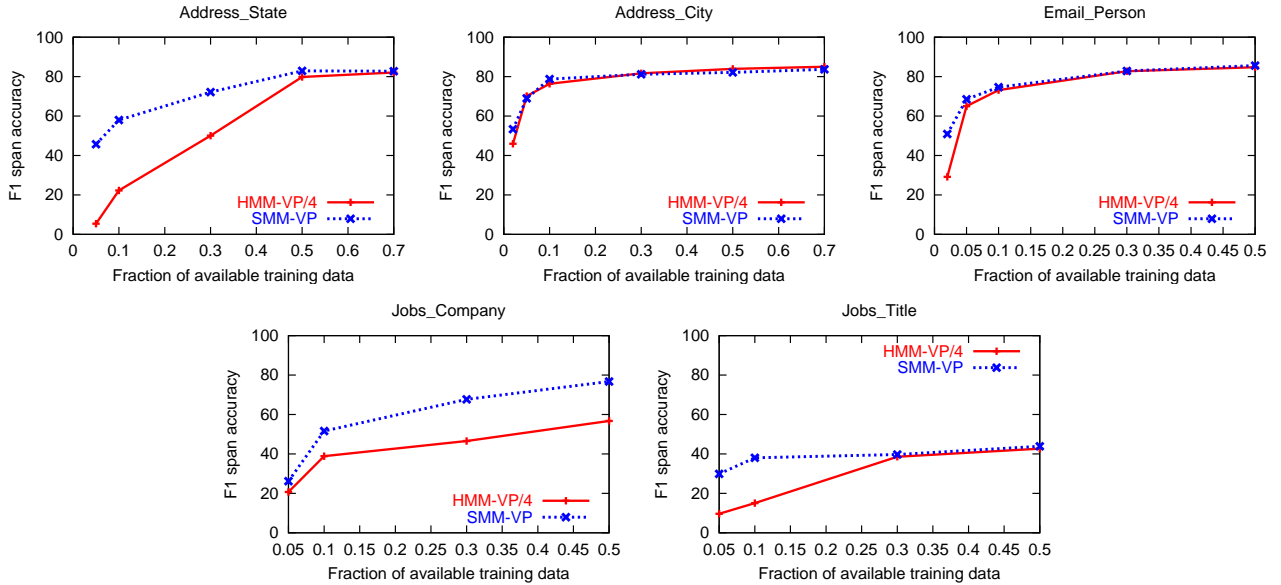


Figure 3: Comparing SMM-VP and HMM-VP/4 with changing training set size on five IE tasks. X-axis is the fraction of examples used for training and Y-axis is field-level F1.

- When dictionary information is used, SMM-VP is always better than both HMM-VP/1 and HMM-VP/4. Somewhat surprisingly, on several problems SMM-VP is *dramatically* better than HMM-VP/4, the previous state of the art in dictionary integration for NER. We will discuss the reasons for some of this difference in the next section.
- Even without a dictionary SMM-VP is better than HMM-VP/4 in many cases. Generally speaking, SMM-VP is never much worse than HMM-VP/4, and sometimes it is substantially better. This is because the SMM-VP allows more features, and more powerful features.

These experiments provide solid empirical evidence of the superiority of the SMM-VP technique over other state of the art methods. Below, we will perform a more detailed comparison of SMM-VP and HMM-VP/4 with changing training sizes, changing history lengths, and alternative dictionaries.

We will not present any detailed comparisons of running times of the two methods since our implementation is not yet optimized for running 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 Impact of changing training size

In Figure 3 we show the impact of increasing training set size on HMM-VP/4 and SMM-VP. In almost all cases, SMM-VP is much better than HMM-VP/4 when the training size is small. As the training size increases, the gap between the two methods generally narrows; this is because the dictionary features are less important when more training data is available (on these datasets).

The amount of data needed for convergence varies substantially. For Address.city and Emails, SMM-VP is much better than HMM-VP/4 for 2% training size or less, and then the two methods are more or less the same. In all other cases the gap is maintained up to 30% of the training data or more. SMM-VP on Address.city shows smaller improvement than Address.state because Address.city has almost four times as many more labeled entities.

3.6 Alternative dictionaries

We re-ran two of our extraction problems on alternative dictionaries to study the relative usefulness of the two methods as the dictionaries get more “distant”. 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 jobs_title extraction task, we obtained a dictionary of 159 job titles in California from a software jobs website (<http://www.softwarejobs.com>). 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 4 shows the result.

Not unexpectedly, the quality of extractions using the alternative dictionaries is lower for both methods. Interestingly, SMM-VP continues to maintain roughly the same lead over HMM-VP/4. For the email data, both methods continue to get a boost over the corresponding no-dictionary version. However, for the Jobs_title data HMM-VP/4 with the alternative dictionary is worse than the no-dictionary

	history	Recall	Precision	F1
Address.state				
HMM-VP/4	1	13.2	70.4	22.3
	2	9.2	70.9	16.3
	3	9.4	42.9	15.4
SMM-VP	1	48.8	71.4	58.0
Address.city				
HMM-VP/4	1	68.5	86.2	76.4
	2	66.4	83.0	73.8
	3	63.9	88.5	74.2
SMM-VP	1	73.7	84.5	78.8
Email.persons				
HMM-VP/4	1	64.4	84.7	73.1
	2	64.7	86.7	74.1
	3	62.0	86.1	72.1
SMM-VP	1	71.5	78.0	74.6

Table 5: Effect of increasing history size on F1

version whereas SMM-VP still gets a 5 points jump in F1.

We conjecture SMM-VP will improve most over HMM-VP/4 in situations where there is a very strong dictionary-independent correlation in the terms comprising an entry. For job titles that correlation is stronger than for names—while a job-title word like “DBA” is very likely to be associated with a word like “Oracle”, a person-name word like “William” is unlikely to have a strong association with “Cohen” across different dictionaries of person names.

3.7 Varying history size

Another reasonable question is how a SMM-VP compares to HMM-VP/4 when HMM-VP/4 is given a history length and window size equivalent to L (the maximum segment size). In the experiments above, history size was set to 1 and window size was set to 3. In Table 5 we show the result of increasing history size from 1 to 3. We find that the performance of HMM-VP/4 does not improve much with increasing history size, and hence SMM-VP still has an edge.

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.

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 has the potential of degrading performance by distorting the model learned from the true training data.

In concurrent work by one of these authors, a scheme is proposed for compiling a dictionary into a very large HMM

	Recall	Precision	F1	Recall	Precision	F1	Recall	Precision	F1
Email data	No dictionary			Original dictionary			Student names		
Dictionary lookup				43.11	85.3	57.3	0	0	0
HMM-VP/4	52.0	82.6	63.8	64.4	84.7	73.1	62.1	81.9	70.6
SMM-VP	57.8	66.6	61.9	71.5	78.0	74.6	67.9	75.0	71.3
Jobs_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	15.3	10.9	12.7	19.9	10.6	13.8	9.8	8.2	8.9
SMM-VP	22.1	38.9	28.2	33.0	44.9	38.0	29.4	39.3	33.6

Table 4: Results with changing dictionary

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 extremely 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 also exploit other types of distance metrics as well.

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 could easily be extended to make use of dictionary-based features. However, in prior experimental comparisons, sliding-window NER systems 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 starting positive examples, which are then used to learn extraction patterns that provide additional entries to the dictionary. These extraction systems are mostly rule-based (except [12, 14]), and appear to assume a relatively clean set of extracted entities, whereas our focus is probabilistic models and the incorporation of large, diverse, and noisy dictionaries.

To our knowledge, semi-Markov models have not been previously 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.

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 formalize a semi-Markov extraction process which relaxes the usual Markov assumptions. This process is based on sequentially classifying *segments* of several adjacent words, rather than single words. In addition to allowing a simple way of coupling NER 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 entities and words from the end of entities. In the semi-Markov formulation, one merely adds new features for entity-beginning words.

In order to learn semi-Markov NER systems, we first described a simple and easy-to-analyze discriminative learning method for HMMS, and then described an extension of this procedure to SMMs. We compared the new 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: it always outperforms the previous baseline, occasionally dramatically. The improvements seem to be largest when there is relatively little training data.

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. Tschantaris, 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 Barbara, 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. To appear.
- [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 Learning 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: evidence 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*.