Corrective Feedback and Persistent Learning for Information Extraction

Aron Culotta [a], Trausti Kristjansson [b], Andrew McCallum [a], Paul Viola [c]

[a] *Dept. of Computer Science, University of Massachusetts, Amherst, MA 01003*

[b] *IBM T.J. Watson Research Center, Yorktown Heights, NY 10598*

[c] *Microsoft Research, Redmond, WA 98052*

Abstract

To successfully embed statistical machine learning models in real world applications, two post-deployment capabilities must be provided: (1) the ability to solicit user corrections and (2) the ability to update the model from these corrections. We refer to the former capability as *corrective feedback* and the latter as *persistent learning*. While these capabilities have a natural implementation for simple classification tasks such as spam filtering, we argue that a more careful design is required for *structured classification* tasks.

One example of a structured classification task is *information extraction*, in which raw text is analyzed to automatically populate a database. In this work, we augment a probabilistic information extraction system with corrective feedback and persistent learning components to assist the user in building, correcting, and updating the extraction model. We describe methods of guiding the user to incorrect predictions, suggesting the most *informative* fields to correct, and incorporating corrections into the inference algorithm. We also present an active learning framework that minimizes not only how many examples a user must label, but also how difficult each example is to label. We empirically validate each of the technical components in simulation and quantify the user effort saved. We conclude that more efficient corrective feedback mechanisms lead to more effective persistent learning.

Key words: information extraction, active learning, graphical models

Email addresses: culotta@cs.umass.edu (Aron Culotta), tkristj@us.ibm.com (Trausti Kristjansson), mccallum@cs.umass.edu (Andrew McCallum), viola@microsoft.com (Paul Viola).

1 Introduction

Machine learning algorithms are rarely perfect. To be successfully deployed, they must compensate for their imperfections by interacting intelligently with the user and the environment. We define two broad categories of such interaction: *corrective feedback* and *persistent learning*.

Corrective feedback is the ability to solicit corrections from the user. For example, corrective feedback may be required when spam filters incorrectly classify email messages, when speech recognizers incorrectly transcribe words, or when automated assembly systems incorrectly join product components. The main difficulty in corrective feedback is designing the corrective action to be as effortless as possible for the user. The amount of effort per correction becomes increasingly important in domains requiring high accuracy, for example where each prediction must be manually inspected for errors.

If after being corrected the system repeats its errors, the user will be justifiably disappointed. This is the motivation behind the second capability, *persistent learning*. *Persistent learning* is the ability of the system to continually update its prediction model *after* deployment. Given corrected data examples, the system should re-estimate its parameters to improve future performance. For example, given enough corrective feedback, a spam filter should become personalized to the type of mail each user receives, and a speech recognizer should become personalized to the speech idiosyncrasies of each user.

Persistent learning and corrective feedback have been successfully implemented for simple classification tasks such as spam filtering. However, such a simple interaction model is not possible for algorithms that operate over more complex domains. In particular, we are interested in algorithms designed for *structured prediction*: classification tasks where the output has multiple interacting labels. Examples of structured prediction tasks include *speech recognition*, where the input is a spoken utterance and the output is a sequence of words, and *information extraction*, where the input is a sequence of text and the output is a relational database of the entities in the text.

Soliciting corrective feedback is often more difficult for structured prediction tasks than for simple prediction tasks. For example, correcting a spam filter can be as simple as a single mouse click, whereas correcting a speech recognizer may require retyping entire words and phrases, and correcting an information extraction system may require re-labeling and re-segmenting extracted entities. The more difficult it is for the user to correct the system, the less feedback the system will receive. This in turn leads to a brittle system incapable of adapting to its environment.

In this paper, we argue that by designing more efficient corrective feedback mechanisms, we can enable more effective persistent learning.

We examine this hypothesis on one common instance of structured classification: information extraction. In particular, we consider the task of discovering contact information (e.g. name, address, phone number) from on-line sources such as email messages and web pages. This is an example of *named-entity recognition* — the task of identifying a set of interesting entity types in text.

As we will show, an extraction system based on linear-chain conditional random fields (CRFs) (Lafferty et al., 2001; Sutton and McCallum, 2006) can extract over 90% of these fields correctly from a diverse set of noisy sources. However, this accuracy is only attainable given hand-labeled data. Efficiently acquiring this data is the goal of this work. We present an *interactive* information extraction system that makes correcting the predictions of a partially-trained extractor as effortless as possible, ensuring data integrity and fast training of a high-accuracy extractor.

There are four main contributions of this paper. The first is an algorithm to incorporate corrective feedback into CRFs (Section 3.1). By constraining the prediction procedure to respect user corrections, we enable what we refer to as *correction propagation*: the correction to one part of the output automatically corrects other parts of the output. We demonstrate empirically that correction propagation can lead to more efficient corrective feedback (Section 3.6.1).

The second contribution is a set of algorithms to determine the order in which predictions should be corrected by the user. For each example, we may want to correct the least confident prediction first, as described in Section 3.2, or we may want to correct the prediction that will maximize the amount of correction propagation, as described in Section 3.3.

Third is the introduction of an interactive information extraction interface (Section 3.4). This interface highlights the label assigned to each field in the unstructured document while flagging labels that should be corrected. The interface also allows for rapid correction using "drag and drop," and supports the correction propagation capability described above.

Finally, relying on these corrective feedback mechanisms, we advocate a cost-sensitive active learning paradigm for information extraction that reduces not only *how many* examples the annotator must label, but also *how difficult* each example is to annotate (Section 4). That is, whereas traditional active learning approaches minimize the number of examples that must be manually labeled, we minimize the number of corrective actions. We show that more efficient corrective feedback mechanisms decrease the amount of effort required to train an accurate extractor.

The remainder of this paper first reviews CRFs for information extraction, then describes each of our four contributions in turn. We perform experiments simulating an interactive information extraction environment and demonstrate the amount of user effort saved through corrective feedback and persistent learning.

2 Information Extraction with Conditional Random Fields

Information extraction (IE) is the task of automatically populating a relational database with facts discovered from natural language text. A common subtask of IE is *named-entity recognition* (NER), the task of annotating text with shallow semantic information, such as the names of people, places, or organizations. For example, in this paper we are concerned with annotating free-text contact records with field labels, such as *name*, *company*, *city*, *phone number*, etc.

More formally, we represent a document D by a sequence of word *tokens* $\mathbf{x} = \langle x_1 \dots x_n \rangle$. The goal of NER is to extract from D a set of fields $\mathbf{F} = \{F_1 \dots F_k\}$, where each field is an attribute-value pair, $F_i = \langle a, v \rangle$ (for example $F_i = \langle$*City*, *San Francisco*$\rangle$). Note that a field value may span multiple word tokens.

For example, consider the input string *John was born in San Francisco, California.* From this sequence of tokens, the NER system should extract the fields $F_1 = \langle$*Name*, *John*$\rangle$, $F_2 = \langle$*City*, *San Francisco*$\rangle$, and $F_3 = \langle$*State*, *California*$\rangle$. We will often refer to the attribute as a *label* of a token; e.g. in this example *California* is labeled as a *State*.

There have been numerous NER systems proposed in the literature. We desire a system that not only has accurate performance, but also facilitates intelligent and efficient interaction with the user.

A simple, but often effective, NER system can be built simply using hand-crafted regular expressions. For example, the pattern "born in [CAPS]" could be used to label as a city any capitalized token that directly follows the phrase "born in." Unfortunately, the infinite variability of human language makes this approach error prone. We categorize NER errors into two types: (1) *precision* errors, e.g. erroneously labeling *Charity Hospital* as a city in the phrase *born in Charity Hospital*, and (2) *recall* errors, e.g. failing to label *San Francisco* as a city in the phrase *raised in San Francisco.* Many wrapper induction techniques have been proposed to learn regular expressions that can reduce some of these errors (Kushmerick et al., 1997); however, they are still constrained by the brittleness of pattern matching.

A popular alternative to pattern matching is statistical machine learning. In this approach, a number of *features* are computed for each token to provide evidence of its label. Example features include information about capitalization, syntax, context words, presence in name lists, and even the regular expressions used in pattern matching techniques. Given some training examples in which tokens are annotated with their true labels, these systems learn correlations between features and labels, thereby inducing a distribution over possible labels for each token.

In addition to often being more accurate and robust than pattern matching techniques, statistical machine learning approaches frequently have the capability of

reliably estimating the confidence of each labeling decision. This becomes important in an interactive system, where we would like to direct the user to fields most likely in need of correction.

Maximum entropy classification (Jaynes, 1979) is a potentially quite powerful machine learning approach to NER, since it allows arbitrary, potentially dependent, features of the input and can also naturally estimate the confidence of its decisions. However, because maximum entropy classification extracts each field independently of related fields, there is no potential for correction propagation.

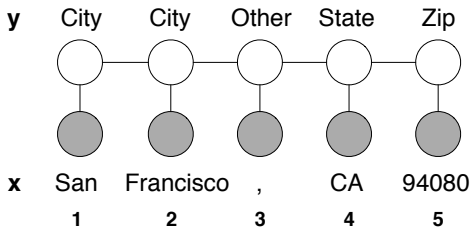
Conditional random fields (CRFs) are a generalization both of maximum entropy models and hidden Markov models that have been shown to perform well on information extraction tasks (Lafferty et al., 2001; Sutton and McCallum, 2006; McCallum and Li, 2003; Pinto et al., 2003; McCallum, 2003; Sha and Pereira, 2003). Like maximum entropy classifiers, they allow for the introduction of arbitrary non-local features; additionally, they capture the dependencies between neighboring labels. CRFs are well-suited for interactive information extraction since the confidence of the labels can be estimated, and there is a natural scheme for optimally propagating user corrections. We now give a brief overview of CRFs.

CRFs are undirected graphical models that encode the conditional probability of values on designated output nodes given values on designated input nodes. In the special case in which the designated output nodes of the graphical model are linked by edges in a *linear chain*, CRFs make a first-order Markov independence assumption among output nodes, and thus correspond to finite state machines (FSMs). In this case CRFs can be roughly understood as conditionally-trained hidden Markov models, with additional flexibility to take advantage of complex, overlapping features.

Let $\mathbf{x} = \langle x_1, x_2, ...x_T \rangle$ be an observed input data sequence, such as a sequence of word tokens in a document (the values on T input nodes of the graphical model). Let $\mathbf{L}$ be a set of FSM states, each of which is associated with a label (such as *LastName* or *PhoneNumber*). Let $\mathbf{y} = \langle y_1, y_2, ...y_T \rangle$ be some sequence of states, (the values on T output nodes). CRFs define the conditional probability of a state sequence given an input sequence as

$$p_\Lambda(\mathbf{y}|\mathbf{x}) = \frac{1}{Z_\mathbf{x}} \exp\left(\sum_{t=1}^{T} \sum_k \lambda_k f_k(y_{t-1}, y_t, \mathbf{x}, t)\right) \quad (1)$$

where $Z_\mathbf{x}$ is a normalization factor over all state sequences, $f_k(y_{t-1}, y_t, \mathbf{x}, t)$ is an arbitrary feature function over its arguments, and $\lambda_k \in \Lambda$ is a learned weight for each feature function. The normalization factor, $Z_\mathbf{x}$, involves a sum over an exponential number of different possible state sequences, but because these nodes with unknown values are connected in a graph without cycles (a linear chain in this case), it can be efficiently calculated via belief propagation using dynamic programming. Inference to find the most likely state sequence is also a dynamic program, in this



F1	F2	F3
City	**State**	**Zip**
San Francisco	CA	94080

Fig. 1. A graphical model of a CRF for a named-entity recognition example. The predicted label sequence $\mathbf{y}$ corresponds to the three extracted fields $F1, F2, F3$.

case very similar to the Viterbi algorithm of hidden Markov models.

The Λ parameters can be determined using supervised machine learning. Given a set of N *training sequences* $\mathcal{D} = \{\mathbf{x}^{(i)}, \mathbf{y}^{(i)}\}$, where $\mathbf{y}^{(i)}$ is the true labeling of token sequence $\mathbf{x}^{(i)}$, the Λ weights of the CRF can be set to maximize the *conditional log likelihood* of the true labels of $\mathcal{D}$. To mitigate over-fitting, the conditional log likelihood is often *regularized* by a Gaussian prior over parameters, with mean 0 and variance σ^2. The resulting function we wish to maximize is

$$\mathcal{L}(\Lambda; \mathcal{D}) = \sum_{i=1}^{N} \log p_\Lambda(\mathbf{y}^{(i)}|\mathbf{x}^{(i)}) - \sum_k \frac{\lambda_k^2}{2\sigma^2}$$

This maximization can be formulated as a convex optimization problem, solved efficiently using hill-climbing methods such as conjugate gradient or its improved second-order cousin, limited-memory BFGS (Liu and Nocedal, 1989). BFGS can simply be treated as a black-box optimization procedure, requiring only that one provide the first-derivative of the function to be optimized. The first-derivative of the regularized conditional log-likelihood is

$$\frac{\delta \mathcal{L}}{\delta \lambda_k} = \left(\sum_{i=1}^{N} C_k(\mathbf{y}^{(i)}, \mathbf{x}^{(i)}) \right) - \left(\sum_{i=1}^{N} \sum_{\mathbf{s}} p_\Lambda(\mathbf{y}|\mathbf{x}^{(i)}) C_k(\mathbf{y}, \mathbf{x}^{(i)}) \right) - \frac{\lambda_k}{\sigma}$$

where $C_k(\mathbf{y}, \mathbf{x})$ is the "count" for feature k given $\mathbf{y}^{(i)}$ and $\mathbf{x}^{(i)}$, equal to the sum of all of the $f_k(y_{t-1}, y_t, \mathbf{x}^{(i)}, t)$ values for each position in the sequence $\mathbf{y}^{(i)}$. The last term, λ_k/σ, is the derivative of the Gaussian prior.

Figure 1 shows an example of the graphical model for a linear-chain CRF. In graphical modeling notation, circles represent random variables, shaded nodes indicated observed random variables, and edges indicate probabilistic dependence. Each edge is parameterized by a set of weighted feature functions representing contextual evidence of a label, such as capitalization, word identity, or presence in a lexicon. The features are presented in more detail in Section 3.5.3.

For illustrative purposes, we will now step through a concrete example of how to

calculate the probability of the label sequence in Figure 1, according to Equation 1. Assume that we have only one type of feature $f_1(y_{t-1}, y_t, \mathbf{x}, t)$, which is equal to 1 if token t is capitalized, and is 0 otherwise. Assume further that the weight associated with this feature is 0.8 if $y_i \in \{\textit{City}, \textit{State}\}$, and is -0.2 otherwise. Then, the probability of the label sequence given in Figure 1 is calculated as

$$\begin{aligned} p_\Lambda(\mathbf{y}|\mathbf{x}) \propto \exp\Big(&0.8 \cdot f_1(\text{null}, \textit{City}, \textit{San}, 1) + 0.8 \cdot f_1(\textit{City}, \textit{City}, \textit{Francisco}, 2) \\ &-0.2 \cdot f_1(\textit{City}, \textit{Other}, \text{“,”}, 3) + 0.8 \cdot f_1(\textit{Other}, \textit{State}, \textit{CA}, 4) \\ &-0.2 \cdot f_1(\textit{State}, \textit{Zip}, \textit{94080}, 5)\Big) \\ \propto &\, 0.8 \cdot 3 = 2.4 \end{aligned}$$

To convert this unnormalized score into a probability, we must divide by $Z_\mathbf{x}$, the sum of the scores for every other possible label sequence for the given input sequence. There exists a well-known dynamic programming solution to calculate this sum in time $O(T\mathbf{L}^2)$, where T is the length of the sequence, and $\mathbf{L}$ is the number of different output labels (see Section 3.1).

Note that in this example the feature only computes evidence over the current token x_t. In general, features can gather evidence from any element of the input sequence, for example a feature that indicates the identity of the previous token, or whether the next token contains only digits. These contextual features are extremely informative for NER tasks.

In the next sections we discuss ways to extend CRFs to support corrective feedback and persistent learning.

3 Corrective Feedback

Although CRFs have been quite successful on many information extraction task, their output will still inevitably contain errors. The goal of this section is to present extensions to CRFs that allow the user to verify and correct system predictions with as little effort as possible.

The first way we reduce effort is by interactively updating system predictions as the user makes corrections (Section 3.1). When a correction is made, the constraints imposed upon the inference algorithm often lead to other errors being automatically corrected with no additional input from the user. We call this capability *correction propagation*.

The second way we reduce effort is by focusing the user's attention to certain fields that should be corrected. The user is directed to fields either when the system has

low confidence in its prediction (Section 3.2) or when correcting that field is expected to lead to correction propagation (Section 3.3).

3.1 Correction Propagation with the Constrained Viterbi Algorithm

When the user corrects the label for one extracted field, we would like the model to re-perform inference in case this correction affects the predicted labels of other fields.

For example, given the name "Charles Stanley," it is likely that the first name is Charles and the last name is "Stanley." But, the opposite is possible as well. Given the error that the two names have been switched, naïve correction systems require two corrective actions. In the *interactive information extraction* system described below, when the user corrects the first name field to be "Stanley," the system then automatically changes the last name field to be "Charles," because this is the most likely interpretation given the correction.

The inference algorithm for CRFs has a natural extension that essentially "clamps" some hidden $\mathbf{y}$ nodes to their corrected value, often resulting in new predictions for other fields. We first briefly describe the traditional inference algorithm, then its constrained counterpart.

In hidden Markov models, the Viterbi algorithm (Rabiner, 1989) (also known as the *max-product* algorithm) is an efficient dynamic programming solution to the problem of finding the state sequence most likely to have generated the observation sequence (i.e. the *most probable explanation* (MPE) inference problem). CRFs employ a conditional analog of Viterbi that returns the most likely state sequence given an observation sequence, i.e. the solution to

$$\mathbf{y}^* = \underset{\mathbf{y}}{\operatorname{argmax}}\ p_\Lambda(\mathbf{y}|\mathbf{x}).$$

To avoid an exponential-time search over all possible settings of $\mathbf{y}$, Viterbi stores the probability of the most likely path at time t which accounts for the first t observations and ends in state y_i. Following the notation of Rabiner (1989), we define this probability to be $\delta_t(y_i)$, where $\delta_0(y_i)$ is the probability of starting in each state y_i, and the induction step is given by:

$$\delta_{t+1}(y_i) = \max_{y'} \left[\delta_t(y') \exp\left(\sum_k \lambda_k f_k(y', y_i, \mathbf{x}, t) \right) \right]. \tag{2}$$

The recursion terminates in

$$y_T^* = \underset{i}{\operatorname{argmax}}[\delta_T(y_i)]$$

We can backtrack through the dynamic programming table to recover $\mathbf{y}^*$.

We now describe how to modify Viterbi to respect a user correction. By a user correction, we mean that a user has fixed the labels for some set of tokens, either by correcting a field label, or adjusting the start or end boundaries of a field.

When a user enters a correction to a field, we represent this by fixing the $\mathbf{y}$ labels for that field to the labels specified by the user. These are encoded as constraints in the Viterbi algorithm, resulting in the *constrained Viterbi* algorithm. Constrained Viterbi alters Eq. 2 such that y^* is constrained to pass through some sub-path $C = \langle y_t, y_{t+1} \ldots \rangle$, corresponding to a user correction. These constraints C now define the new induction

$$\delta_{t+1}(y_i) = \begin{cases} \max_{y'} \left[\delta_t(y') \exp\left(\sum_k \lambda_k f_k(y', y_i, \mathbf{x}, t) \right) \right] & \text{if } y_i = y_{t+1} \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

for all $y_{t+1} \in C$. For time steps not constrained by C, Eq. 2 is used instead. Thus, constrained Viterbi restricts Viterbi search to only consider paths that respect constraints C.

Because CRFs model the dependence between adjacent labels, a change to the prediction for label y_i can change the MPE estimate for label y_{i+1}, which can in turn change the estimate for y_{i+2}, etc. In this way, a single user correction can be propagated throughout the entire sequence.

In an interactive setting, when the user corrects one field, these corrections are propagated in real-time to the rest of the fields, allowing the user to fix multiple errors with a single action.

We refer to a CRF augmented with constrained Viterbi as a *constrained conditional random field* (CCRF).

3.2 Confidence Estimation with the Constrained Forward-Backward Algorithm

Manually inspecting each automatically labeled field can be tedious for the user. One way to mitigate this effort is to direct the user to fields that are most likely to be incorrect. In this section, we describe how a CRF can estimate the confidence of each field it extracts.

The conditional probability of the label for one *token* $p(y_i|\mathbf{x})$ is calculated by a variant of the Viterbi algorithm called *forward-backward* (also known as the *sum-product* algorithm). This algorithm is similar to the Viterbi algorithm; but instead of choosing the most probable state sequence, forward-backward evaluates all possible state sequences given the observation sequence.

The *forward values* $\alpha_{t+1}(y_i)$ are recursively defined similarly to Eq. 2, except the $\max$ is replaced by a summation. Thus we have

$$\alpha_{t+1}(y_i) = \sum_{y'} \left[\alpha_t(y') \exp\left(\sum_k \lambda_k f_k(y', y_i, \mathbf{x}, t) \right) \right]. \quad (4)$$

The recursion terminates to define $Z_{\mathbf{x}}$ in Eq. 1:

$$Z_{\mathbf{x}} = \sum_i \alpha_T(y_i) \quad (5)$$

Although the probability of the label for one *token* $p(y_i|\mathbf{x})$ is easily obtained by the CRF inference algorithm, the label for an entire *field* requires calculating the probability of a sequence of tokens $p(y_i \dots y_k|\mathbf{x})$, where the field contains tokens $x_i \dots x_k$.

To estimate the confidence the CRF has in an extracted field, we employ a technique we term *constrained forward-backward* (Culotta and McCallum, 2004), which calculates the probability of any state sequence matching the labeling of the field under consideration. The constrained forward-backward algorithm calculates the probability of any sequence passing through a set of constraints $C = \langle y_q \dots y_r \rangle$, where now $y_q \in C$ can be either a positive constraint or a *negative* constraint. A negative constraint constrains the forward value calculation *not* to pass through state y_q.

The calculations of the forward values can be made to conform to C in a manner similar to the constrained Viterbi algorithm. If $\alpha'_{t+1}(y_i)$ is the constrained forward value, then $Z'_{\mathbf{x}} = \sum_i \alpha'_T(y_i)$ is the value of the *constrained lattice*. Our confidence estimate is equal to the normalized value of the constrained lattice: $Z'_{\mathbf{x}}/Z_{\mathbf{x}}$. For predicted value f for field F_i, this confidence estimate is equivalent to $P(F_i = f|\mathbf{x})$.

In the context of interactive form filling, the constraints C correspond to an automatically extracted field. The positive constraints specify the observation tokens labeled inside the field, and the negative constraints specify the boundary of the field. For example, if state names *B-Title* and *I-JobTitle* represent label tokens that begin and continue a *JobTitle* field, and the system labels observation sequence $\langle x_2 \dots x_5 \rangle$ as a *JobTitle* field, then $C = \langle y_2 = \textit{B-JobTitle}, y_3 = y_4 = y_5 = \textit{I-JobTitle}, y_6 \neq \textit{I-JobTitle} \rangle$. Thus, the confidence estimate corresponds to the probability of any state sequence predicting these constrained *JobTitle* labels.

3.3 Maximizing Correction Propagation

While highlighting the least confident field is likely to direct the user to incorrectly labeled fields, an alternative objective is to solicit user actions that maximize the number of fields automatically fixed by correction propagation. The motivation for

this objective is to maximize the number of "free" corrections enabled by correction propagation. Because of the dependencies among predicted labels, knowing the true label of one field may reduce the uncertainty of the predictions for other fields.

We define two scoring functions that rank fields to be labeled based on the expected amount of correction propagation that will follow their correction.

The first scoring function prefers fields that have high mutual information with the rest of the sequence. Let $\mathbf{y}^{-i}$ be the set of label variables excluding those for field F_i. The score for field F_i is the mutual information between $\mathbf{y}^{-i}$ and F_i:

$$\begin{aligned} I(\mathbf{y}^{-i}|F_i) &= H(F_i) - H(F_i|\mathbf{y}^{-i}) \\ &= -\sum_f P(F_i = f) \log P(F_i = f) \\ &+ \sum_j \sum_f P(\mathbf{y}^{-i} = \mathbf{y}^{(j)}, F_i = f) \log P(\mathbf{y}^{-i} = \mathbf{y}^{(j)}|F_i = f) \end{aligned} \quad (6)$$

In the last term, the sum over j requires iterating over all possible labelings of $\mathbf{y}$. We approximate this exponential calculation by restricting the sum to the top T most probable paths (e.g. $T = 30$). Similarly, when field F_i contains many tokens, summing over all competing predictions can also become intractable. In this case, we sample from the top most probable predictions for F_i.

The intuition behind this scoring function is that if the distribution over one field conveys a large amount of information about the distribution over other fields, then correcting this field may lead to the automatic correction of other fields.

The second scoring function attempts to maximize the expected number of automatic corrections directly. Let $\mathbf{y}^*_{F_i=f}$ be the constrained Viterbi path where field F_i is clamped to the setting f. Let $\#(F_i = f)$ be the number of labels in $\mathbf{y}^*_{F_i=f}$ that are changed from the original Viterbi output when the labeling for field F_i is set to f. Then the expected number of tokens automatically corrected by having the user correct field F_i is estimated as

$$EC(F_i) = \sum_f P(\mathbf{y}^*_{F_i=f}|\mathbf{x})\#(F_i = f) \quad (7)$$

The intuition behind this measure is to weight the number of label changes effected by setting F_i to f by the probability that those changes are correct.

We compare the effectiveness of these scoring functions empirically in Section 3.6.2.

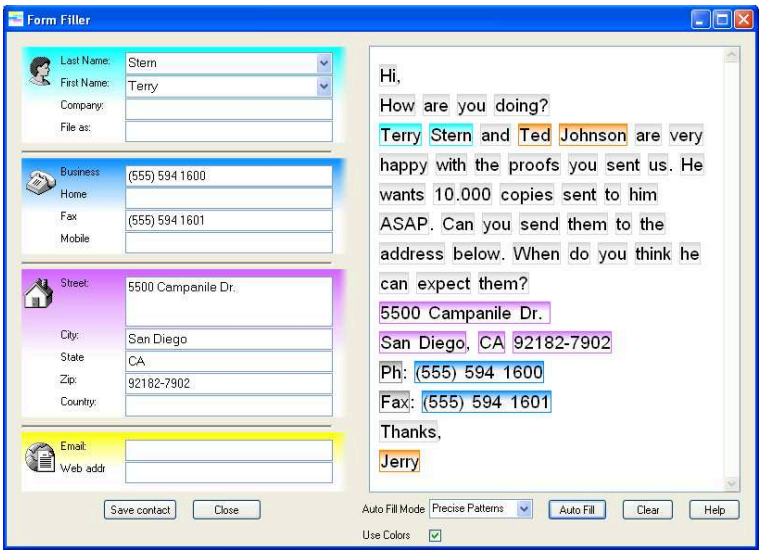


Fig. 2. A user interface for entry of contact information. The user interface relies on interactive information extraction. If a user makes a correction, the interactive parser can update other fields. Notice that there are 3 possible names associated with the address. The user is alerted to the ambiguity by the color coding.

3.4 User Interface

From the perspective of user interface design, there are a number of goals, including reducing cognitive load, reducing the number of user actions (clicks and keystrokes), and speeding up the data acquisition process. An important element that is often overlooked is the confidence the user has in the integrity of the data. This is crucial to the usability of the application, as users are not tolerant of (surprising) errors, and will discontinue the use of an automatic semi-intelligent application if it has corrupted or misclassified information. Unfortunately such factors are often hard to quantify. We describe an interface that enables efficient corrective feedback to ensure data integrity.

3.4.1 User Interfaces for Information Extraction

Figure 2 shows a user interface that facilitates interactive information extraction. The fields to be populated are on the left side, and the source text was pasted by the user into the right side. The information extraction system extracts text segments from the unstructured text and populates the corresponding fields in the contact record. This user interface is designed with the strengths and weaknesses of the information extraction technology in mind. Some important aspects are:

- The UI displays visual aids that allow the user to quickly verify the correctness of the extracted fields. In this case color-coded correspondence is used (e.g. blue for all phone information, and yellow for email addresses). Other options include

arrows or floating overlayed tags.

- The UI allows for rapid correction. For example, text segments can easily be grouped into blocks to allow for a single click-drag-drop. In the contact record at the left, fields have drop down menus with other candidates for the field. Alternatively the interface could include "try again" buttons next to the fields that cycle through possible alternative extractions for the field until the correct value is found.
- By integrating the original text in the interface, the system addresses the common "recall" errors of extractors. That is, if a token is incorrectly labeled as not being part of the record, the user can correct this error by dragging the token to the correct field box.
- The UI immediately propagates all corrections and additions by the constrained Viterbi algorithm.
- The UI visually alerts the user to fields that have low confidence based on the constrained forward-backward algorithm. Furthermore, in the unstructured text box, possible alternatives may be highlighted (e.g. alternate names are indicated in orange).

Confidence scores can be incorporated in a UI in a number of ways. Field assignments with relatively low confidence can be visually marked. If a field assignment has very low confidence, and is likely to be incorrect, we may choose not to fill in the field at all. The text that is most likely to be assigned to the field can then be highlighted in the text-box (e.g. in orange).

Another related case is when there are multiple text segments that are all equally likely to be classified as e.g. a name, then this could also be visually indicated (as is done in Figure 2).

3.5 Experimental Setup

Below we simulate an interactive information extraction environment and show that correction propagation and confidence estimation can decrease the expected amount of user effort.

3.5.1 User Interaction Models

For the purposes of quantitative evaluation we will simulate the behavior of a user performing contact record entry, verification, and correction. This allows for a simpler experimental paradigm that can more clearly distinguish the values of the various technical components.

A large number of user interaction models are possible given the particulars of the interface and information extraction engine. Here we outline the basic models that

will be evaluated in the experimental section.

UIM1: The simplest case. The user is presented with the results of automatic field assignment and has to correct all errors (i.e. no correction-propagation).

UIM2: Under this model, we assume an initial automatic field assignment, followed by a single randomly-chosen manual correction by the user. We then perform correction-propagation, and the user has to correct all remaining errors manually.

UIM3: This model is similar to UIM2. We assume an initial automatic field assignment. Next the user is asked to correct the *least confident incorrect field*. The user is visually alerted to the fields in order of confidence, until an error is found. We then perform correction-propagation and the user then has to correct all remaining errors manually.

UIMm: The user has to fill in all fields manually.

3.5.2 The Expected Number of User Actions:

The goal in designing a new application technology is that users see an immediate benefit in using the technology. Assuming that perfect accuracy is required, benefit is realized if the technology increases the time efficiency of users, or if it reduces the cognitive load, or both. Here we introduce an efficiency measure, called the Expected Number of User Actions, which will be used in addition to standard IE performance measures.

The *Expected Number of User Actions* (ENUA) measure is defined as the number of user actions (e.g. clicks) required to correctly enter all fields of a record. For these experiments, we define an action to be the correction of one field, either by entering a field, changing its label or adjusting its boundaries. The Expected Number of User Actions will depend on the user interaction model. To express the Expected Number of User Actions, we introduce the following notation: $P_i(j)$ is the probability distribution over the number of errors j after i manual corrections. This distribution is represented by the histogram in Figure 3.

Under UIM1, which does not involve correction propagation, the Expected Number of User Actions is:

$$\mathrm{ENUA} = \sum_{n=0}^{\infty} nP_0(n) \tag{8}$$

where $P_0(n)$ is the distribution over the number of incorrect fields (see Figure 3).

In models UIM2 and UIM3 the Expected Number of User Actions is

$$\mathrm{ENUA}_1 = (1 - P_0(0)) + \sum_n nP_1(n). \tag{9}$$

where $P_0(0)$ is the probability that all fields are correctly assigned initially and $P_1(n)$ is the distribution over the number of incorrect fields in a record after one

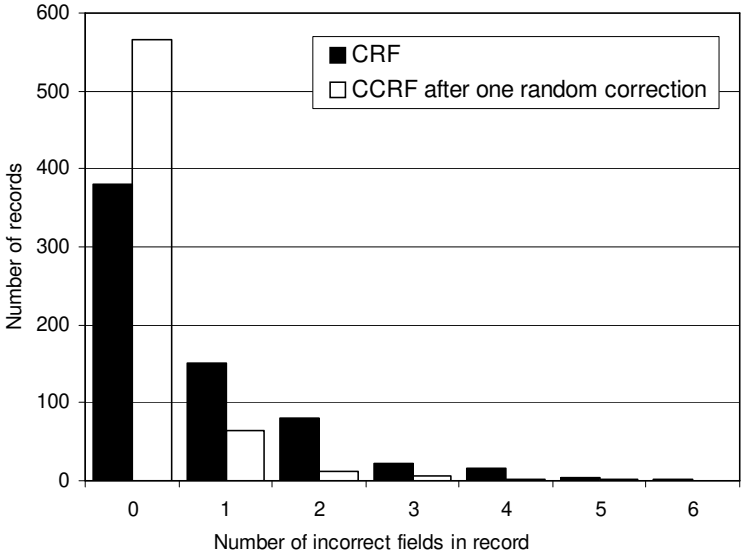


Fig. 3. Histogram, where records fall into bins depending on how many fields in a record are in error. Solid bars are for CRF before any corrections. The shaded bars show the distribution after one random incorrect field has been corrected. These can be used to estimate $P_0(j)$ and $P_1(j)$, respectively.

field has been corrected. The distribution P_1 will depend on which incorrect field is corrected, e.g. a random incorrect field is corrected under UIM2, whereas the least confident incorrect field is corrected under UIM3. The subscript 1 on $ENUA_1$ indicates that correction-propagation is performed once.

3.5.3 Data

For training and testing we collected 2187 documents (27,560 words) from web pages and email and hand-labeled 25 fields. [1] Each document example consists of one contact record that must be labeled with the correct field names, and may contain tokens that are not part of the record (e.g. email text). Some data comes from pages containing lists of addresses, and about half come from disparate web pages found by searching for valid pairs of city name and zip code. For each experiment, we sampled three random splits of the data, reserving 70% for training and 30% for testing.

The features consist of capitalization features, 24 regular expressions over the token text (e.g. *ConstainsHyphen*, *ContainsDigits*, etc.), character n-grams of length 2-4,

[1] The 25 fields are: *FirstName, MiddleName, LastName, NickName, Suffix, Title, JobTitle, CompanyName, Department, AddressLine, City1, City2, State, Country, PostalCode, HomePhone, Fax, CompanyPhone, DirectCompanyPhone, Mobile, Pager, VoiceMail, URL, Email, InstantMessage*

	Token Acc.	F1	Prec	Rec
CRF	**92.30**	**88.47**	**89.03**	**87.93**
MaxEnt	89.80	82.48	82.48	82.47

Table 1
Token accuracy and field performance for the Conditional Random Field based field extractor, and the Maximum Entropy based field extractor. All differences are statistically significant ($p = 0.01$).

and offsets of these features within a window of size 5. We also used 19 lexicons, including "US Last Names," "US First Names," "State names," "Titles/Suffixes," "Job titles," and "Road endings." Feature induction was not used in these experiments.

3.6 Results

We implement two machine learning methods to automatically annotate the text of each contact record. *CRF* is the conditional random field described in Section 2. *MaxEnt* is a maximum entropy classifier with the same set of features as the CRF. However, *MaxEnt* does not model the dependence between adjacent labels. Table 1 shows the performance for the two methods averaged over three random trials. Column 1 lists the token accuracy (the proportion of tokens labeled correctly), and columns 3-4 list the precision and recall at the field level; that is, all the tokens in a field must be extracted correctly to be considered correct. F1 is the harmonic mean of recall and precision. These experiments do not include any user feedback. Notice that the token error rate of the CRF system is about 25% lower than that of the MaxEnt system. These results are statistically significant according to a paired-t test with $p = 0.01$.

In the following sections, we start by discussing results in terms of the Expected Number of User Actions. Then we discuss results that highlight the effectiveness of correction-propagation and confidence estimation.

3.6.1 User Interaction Experiments

Table 2 shows the Expected Number of User Actions for the different algorithms and User Interaction Models. In addition to the *CRF* and *MaxEnt* algorithms, Table 2 shows results for *CCRF*, which is the constrained conditional random field classifier presented in this paper.

The baseline user interaction model (UIM1) is expected to require 0.73 user actions per record. Notice that manual entry of records is expected to require on average 6.31 user actions to enter all fields, about 8.6 times more actions than UIM1. This

	ENUA	Change
CRF – (UIM1)	0.73	baseline
CCRF – (UIM2)	0.63	**-13.9%**
CCRF – (UIM3)	0.64	-11.3%
MaxEnt – (UIM1)	0.94	+29.0%
Manual – (UIMm)	6.31	+770.0%

Table 2
The Expected Number of User Actions (ENUA) to completely enter a contact record. Notice that Constrained CRF with a random corrected field reduces the Expected Number of User Actions by 13.9%.

difference confirms that correcting the CRF requires much less effort than entering fields manually.

The improvement of UIM2 over UIM1 is due to correction propagation. In UIM2, correction propagation occurs between the user's first and second correction, often reducing the number of actions. The ENUA drops to 0.63, which is a relative drop in ENUA of 13.9%. In comparison, manual entry requires over 10 times more user actions.

Confidence estimation is used in UIM3. Recall that in this user interaction model the system assigns confidence scores to the fields, and the user is asked to correct the least confident *incorrect* field.

Interestingly, correcting a random field (ENUA = 0.63) seems to be slightly more informative for correction-propagation than correcting the least confident erroneous field (ENUA = 0.64). While this may seem surprising, recall that a field will have low confidence if the posterior probability of the competing labels is close to the score for the chosen class. Hence, it only requires a small amount of extra information to boost the posterior for one of the other labels and "flip" the classification. We can imagine a contrived example containing two adjacent incorrect fields. In this case, we should correct the *more* confident of the two to maximize correction propagation. This is because the field with lower confidence requires a smaller amount of extra information to correct its classification, all else being equal.

To better understand this phenomenon, in the next section we compare different methods of estimating the amount of correction propagation.

3.6.2 Correction Propagation Experiments

In this section, we describe experiments that directly measure the amount of correction propagation enabled by different methods of ordering field corrections.

	CFB	*EC*	*MI*
%OPT	.536	.571	**.875**

Table 3
The percentage of optimal correction propagation for competing scoring functions.

We compare the scoring functions described in Section 3.3 to determine which best estimates the amount of correction propagation. For each record, each field is given a score by the scoring function, and the incorrect field with the highest score is corrected. We then measure the number of fields automatically corrected by this one manual correction.

For comparison, we also implement two boundary scoring functions, *OPT* and *NONOPT*. Given a record with errors in multiple fields, *OPT* gives the highest score to the incorrect field that will result in the maximum amount of correction propagation; *NONOPT* results in the least amount of correction propagation. We note that *OPT* is not a strict upper-bound, as there may be combinations of corrections that result in greater propagation than choosing a single correction greedily.

The three other scoring functions are *CFB*, which uses constrained forward-backward to score each field with the negative of its confidence value; *EC*, the expected number of correction (Equation 7); and *MI*, the mutual information criterion (Equation 6).

The values in Table 3 are normalized to be a percentage of optimal performance. If $N(X)$ is the number of field errors that remain under scoring function X, then

$$\%\text{OPT}(X) = \frac{N(\text{NONOPT}) - N(X)}{N(\text{NONOPT}) - N(\text{OPT})}$$

Thus, *%OPT(NONOPT)*$= 0$ and *%OPT(OPT)*$= 1$.

These results suggest that the mutual information criterion (*MI*) is the best estimate of the expected amount of correction propagation. *MI* outperforms *EC* most likely because *EC* only considers the optimal path for each possible correction of a field, whereas *MI* considers the full distribution of state sequences (up to the T-best approximation).

If the system knows which fields are incorrectly labeled, it can maximize correction propagation by soliciting corrections in the order determined by *MI*. Of course, the system does not know which fields are incorrect until the user corrects them. Because a field with a high *MI* score is not necessarily incorrect, *MI* will often direct the user to fields needing no correction. This incurs the additional user effort of verifying correct fields.

To reduce this burden, in the next section we evaluate how accurately the CRF can

predict whether a field is correct.

3.6.3 Confidence Estimation Experiments

A simple way of assessing the effectiveness of the confidence measure is to ask how effective is it at directing the user to an incorrect field. In our experiments with CCRFs, the number of records that contained one or more incorrect fields was 276. Using the constrained forward-backward algorithm, the least confident field was truly incorrect in 226 out of those 276 records. Hence, confidence estimation correctly predicts an erroneous fields 81.9% of the time. If we instead choose a token at random, then we will choose an incorrect token in 80 out of the 276 records, or 29.0%. In practice, the user does not initially know where the errors are, so confidence estimates can be used effectively to direct the user to incorrect fields.

We perform a more thorough evaluation under a different user scenario, in which we wish to reduce the labeling error rate of a large amount of data, but we do not need the labeling to be error free. If we have limited man-power, we would like to maximize the efficiency of the human labeler.

This user interaction model assumes that we allow the human labeler to *verify* **or** *correct* a single field in each record, before going on to the next record.

As before the constrained conditional random field model is used, where constrained forward-backward predicts the least confident extracted field. If this field is *incorrect*, then *CCRF* is supplied with the correct labeling, and correction propagation is performed using constrained Viterbi. If this field is *correct*, then no changes are made, and we go on to the next record.

The experiments compare the effectiveness of verifying or correcting the least confident field i.e. *CCRF - (L.Conf)*, to verifying or correcting an arbitrary field i.e. *CCRF - (Random)*.

Finally, *CMaxEnt* is a Maximum Entropy classifier that estimates the confidence of each field by averaging the posterior probabilities of the labels assigned to each token in the field. As in *CCRF*, the least confident field is corrected if necessary. Note that *CMaxEnt* does not perform correction propagation, since each field is predicted independently.

Table 4 shows results after a single field in each record has been verified or corrected. Notice that if a random field is chosen to be verified or corrected, then the token accuracy increases to 93.82%, only a 20.6% reduction in error rate. If however, we verify or correct only the least confident field, the error rate is reduced by 47.8%. These results are statistically significant according to a paired-t test ($p = 0.01$).

Method	Δ Error	Token Acc	F1	Precision	Recall
CCRF - (L. Conf.)	**-47.8%**	**95.69**	**93.98**	**94.46**	**93.52**
CCRF - (Random)	-20.6%	93.82	90.85	91.58	90.13
CMaxEnt	-30.1%	92.46	87.75	88.39	87.11

Table 4
Token accuracy and field performance for interactive field labeling. *CCRF - (L. Conf.)* obtains a 47.8% reduction in F1 error over *CRF*. These reduction results are relative to Table 1, where no user corrections are given. The improvements of *CCRF - (L. Conf.)* over *CCRF - (Random)* and *CMaxEnt* are statistically significant (paired-t test, p = 0.01).

	Pearson's r	Avg. Precision
CFB	**0.530**	**97.8**
Random	0.003	88.93
WorstCase	-	72.8

Table 5
The correlation coefficient and average precision evaluations of the constrained forward-backward confidence estimate.

This difference illustrates that reliable confidence prediction can increase the effectiveness of a human labeler. Also note that the 47.8% error reduction *CCRF* achieves over *CRF* is substantially greater than the 30.1% error reduction between *CMaxEnt* and *MaxEnt*. This difference is due both to the correction propagation and more accurate confidence estimation of CRFs.

To explicitly measure the effectiveness of the constrained forward-backward algorithm for confidence estimation, Table 5 displays two evaluation measures: Pearson's *r* and average precision. Pearson's *r* is a correlation coefficient ranging from -1 to 1 which measures the correlation between a confidence score of a field and whether or not it is correct.

Given a list of extracted fields ordered by their confidence scores, average precision measures the quality of this ordering. We calculate the precision at each point in the ranked list where a correct field is found and then average these values. *WorstCase* is the average precision obtained by ranking all incorrect fields above all correct fields. Both Pearson's r and average precision results demonstrate the effectiveness of constrained forward-backward for estimating the confidence of extracted fields.

We summarize the empirical results thus far as follows:

- Correction propagation reduces the expected number of actions to correct an automatically extracted database.
- Mutual information is the most reliable estimator of correction propagation, among the three estimators compared.
- Confidence estimation with constrained forward-backward can accelerate data

cleaning by directing the user to fields most likely needing correction.

4 Persistent Learning

Thus far, we have discussed extensions to CRFs to enable rapid correction of system errors. However, we have not yet described how to use these corrections to improve the prediction model of the CRF. In this section, we will discuss persistent learning for CRFs. The techniques presented here can be used either to create a new CRF for a novel domain, or to improve an existing CRF with new training data.

Below, we discuss a cost-sensitive active learning framework to train a CRF interactively while minimizing the amount of time spent labeling data. The efficient corrective feedback techniques discussed in the previous sections are incorporated into this active learning system to improve learning rates.

4.1 Active Learning for Information Extraction

Training a CRF extractor requires labeling a training set with the true labels of each token. This is particularly expensive to obtain for structured prediction tasks, since each training example may have multiple, interacting labels, all of which must be correctly annotated for the example to be of use to the learner. To give the user the flexibility to use these techniques on customized tasks, we would like to make this labeling process as painless as possible.

Active learning is a machine learning technique designed to address this problem. The idea is to optimize the order in which the training examples are labeled to increase learning efficiency (Cohn et al., 1995; Lewis and Catlett, 1994). Most active learners are evaluated by plotting a learning curve that displays the learner's performance on a held-out data set as the number of labeled examples increases. An active learner is considered successful if it obtains better performance than a traditional learner given the same number of labeled examples. Thus, active learning expedites annotation by reducing the *number* of labeled examples required to train an accurate model.

However, this paradigm assumes each example is equally difficult to annotate. While this assumption may hold in traditional classification tasks, in structured classification tasks it does not. For example, consider the following labeled example:

```
<name> Jane Smith </name>
   <title> CEO </title>
<company> Unicorp, LLC </company>
```

```
Phone: <phone> (555)555-5555 </phone>
```

To label this example, the user must not only specify which type of field each token belongs to, but also must determine the start and end boundaries of each field. Clearly, the amount of work required to label an example such as this will vary between examples, based on the number of fields. Additionally, unlike in traditional classification tasks, a structured prediction system may be able to *partially* label an example, which can simplify annotation. In the above example, the partially-trained system might correctly segment the title field, but mislabel it as a company name. These partial predictions can reduce labeling effort.

This greater variety of labeling effort is not reflected by the standard evaluation metrics from active learning. Since our goal is to reduce annotation effort, it is desirable to design a labeling framework that considers not only *how many* examples the annotator must label, but also *how difficult* each example is to annotate.

In the next section, we propose a framework to address these shortcomings for a CRF-based extraction system. We then provide a fine-grained extension of the Expected Number of User Actions measure defined in Section 3.5.2 that distinguishes between boundary and classification annotations. Finally, we demonstrate an interactive information extraction system that aims to minimize the amount of effort required to train an accurate extractor.

4.2 Annotation framework

To expedite annotation for information extraction, we first note that the main difference between labeling IE examples and labeling traditional classification examples is the problem of boundary annotation (or *segmentation*). Given a sequence of text that is correctly segmented, choosing the correct label for each field is simply a classification task: the annotator must choose among a finite set of labels for each field. However, determining the boundaries of each field is an intrinsically distinct task, since the number of ways to segment a sequence is exponential in the sequence length. Additionally, from a human-computer interaction perspective, the "clicking and dragging" involved in boundary annotation generally requires more hand-eye coordination from the user than does classification annotation.

With this distinction in mind, our system reduces annotation effort in two ways. First, many segmentation decisions are converted into classification decisions by presenting the user with multiple predicted segmentations to choose from. Thus, instead of hand segmenting each field, the user may select the correct segmentation from the given choices.

Second, the system uses the effort-saving techniques discussed in Section 3 to allow the user to efficiently correct examples to be added to the training set.

The resulting system allows the user to constrain the predictions of the learner without manually annotating the boundaries of incorrect segments. Very often, these constraints will allow the user to simply select the correct annotation from among the provided choices. Thus, the annotator can frequently label a record without explicitly annotating the boundaries.

We validate this active learning framework in an interactive information extraction system, reducing the total number of annotation actions by 21% and the number of boundary annotations by 42%, as compared with competing methods.

We first provide a brief overview of the annotation framework applied to IE. Given an IE learning algorithm L and a set of unlabeled data U, the task is to iteratively solicit annotations from the user and retrain the extractor. Each example is a sequence of tokens (e.g. a paragraph), and a labeled example provides the true field label for each token.

At iteration t, the system operates as follows:

(1) Rank the set of unlabeled examples U by the priority to be labeled.
(2) Select the top-ranked example $u \in U$ to be labeled.
(3) Present to the user the top k labelings of u predicted by L^t (the model at time t).
(4) If the correct labeling exists in the top k choices, allow the user to select that labeling, and add u to the labeled data set.
(5) Otherwise, for *any* field in these k predictions that is *segmented* correctly but *classified* incorrectly, allow the user to provide the correct label for this field.
(6) Based on these corrections, generate a new set of k predictions, propagating these corrections to possibly fix other errors.
(7) If the correct labeling exists in these new top k choices, allow the user to select that labeling and add u to the labeled dataset.
(8) Otherwise, if the correct labeling still does not exist in these k predictions, allow the user to manually correct one of these incorrect k predictions with the true labeling.

Notice that the only step in which the user must manually segment fields is step 8. Steps 4 and 7 allow the user to label the sequence by making a choice among k predictions. Step 5 allows the user to provide correct field labels to the learner, without manually segmenting fields. In step 6, the system performs constrained inference to generate a new set of predictions that conform to the user's corrections. It is in this step that the system often automatically corrects segmentation errors present in the first k choices.

This framework allows the user to rapidly and easily annotate examples and correct the system's predictions, while reducing the amount of effort spent labeling boundaries.

In the remaining sections, we describe in more detail the components of this system.

4.3 Ranking function

In step (1), the system ranks the unlabeled examples by the order in which they should be labeled. The ranking function should order examples to create the steepest learning curve (i.e. achieve the highest accuracy with the fewest number of labeled examples).

This ranking function is the subject of much of the work in active learning (Cohn et al., 1995; Lewis and Catlett, 1994; Muslea et al., 2003). However, our proposed framework is not directly concerned with this ranking function, but rather with the user interaction after the examples have been ranked. Hence, any of the popular active learning ranking functions can be used in step (1). We experiment with two common methods:

- **LeastConfidence:** This uncertainty-based approach ranks each example by the probability of the top prediction, i.e. $p(\mathbf{y}^*|\mathbf{x})$.
- **Query-by-committee (QBC)**: This committee-based approach trains a pool of learners and ranks examples by the amount of disagreement among the pool. In particular, we split the set of labeled examples into m sets, and train a CRF on each set. To score an unlabeled example, we generate m labelings, one from each CRF. We calculate the *normalized vote entropy* (Argamon-Engelson and Dagan, 1999) for a labeled token t as follows:

$$D(t) = -\frac{1}{\log \min(m, |L|)} \sum_l \frac{V(l,t)}{m} \log \frac{V(l,t)}{m}$$

  where $V(l,t)$ is the number of labelings assigning label l to token t, and $|L|$ is the number of possible labels. To obtain the score for an entire sequence, we average the vote entropies of each labeled token. This is a measure of how much disagreement exists among the m CRFs.

4.4 Presenting multiple predictions

To present the user with the top k predictions, we must extend the CRF inference algorithm to return k predictions, instead of simply the top prediction. There are well-established, efficient modifications to the Viterbi algorithm that can calculate the top k optimal predictions, often called *k-best Viterbi* (Schwartz and Chow, 1990). This algorithm can be viewed as a beam search through the space of possible predictions. We apply this algorithm to inference in CRFs to generate the k most probable predictions.

In step 5, the annotator provides the true label for fields that have been correctly segmented but incorrectly classified. The system must then produce the top k predictions that conform to these new annotations.

In Section 3.1 we described the *constrained Viterbi* algorithm, which modifies the traditional Viterbi algorithm to prune from the search space those labelings that do not agree with the given annotations. We extend this to our current task using an algorithm we call *k-best constrained Viterbi*, which, as its name suggests, combines *k-best Viterbi* with *constrained Viterbi*. This extension can be straight-forwardly implemented by constraining the *k-best Viterbi* algorithm to prune predictions that do not agree with the annotations.

Using this algorithm, we enable the system to solicit corrections for the *classification* of fields, which are then propagated to correct both the *classification* and *segmentation* of other fields. In this way, we can reduce the amount of effort expended on segmentation labeling.

4.5 Measuring annotation effort

We refine the Expected Number of User Actions metric from Section 3.5.2 to construct a more fine-grained estimate of the number of actions required to label each example. Whereas ENUA assumes it takes one action to enter, relabel, or adjust the boundaries of a field, we wish to distinguish among these actions. We define three atomic labeling actions: *start*, *end*, and *type*, corresponding to labeling the start boundary, end boundary, and type of an field.

Thus, labeling the input

```
<name> Jane Smith </name>
   <title> CEO </title>
```

requires 2 *start*, 2 *end*, and 2 *type* actions. The goal of our annotation framework is to reduce the total number of annotation actions.

We can see that a partially labeled example can require fewer annotation actions. For example, consider the following partially labeled record:

```
<name> Jane </name> Smith
 <company> CEO </company>
```

This requires one *end* action to fix the ending boundary of "Jane," and one *type* action to change "CEO" from a company to a title. Thus, using the partial labeling has reduced the total number of required actions from 6 to 2.

By presenting the user with k predictions, we introduce another action: If one of

Action Name	Action Description
type	User corrects the field label for a token or set of token.
start	User adjusts the beginning boundary of a field.
end	User adjusts the ending boundary of a field.
choice	User selects the correct record labeling from a choice of k labelings.

Table 6
The set of measured user actions.

the k predictions is correct, the user must choose this prediction. We call this action *choice*. A summary of user actions is given in Table 6.

To simulate corrections, we accumulate the number of times each action is performed. In the first round, when the user corrects the labels of correctly segmented fields, the only action incurred is the *type* action. If none of the k constrained predictions are correct, then (and only then) the user must perform the segmentation actions *start* and *end*.

It will generally be the case that some actions are more expensive than others. For example, as mentioned earlier, *start* and *end* actions may require more hand-eye coordination than *type* actions. A cost-sensitive approach could take this into account; however, in these experiments, we assume each action has unit cost.

4.6 Experiments

Using the same fully annotated collection of extracted contact records from Section 3.5, we simulate our annotation framework and measure the performance of the CRF with respect to the number of actions required to train it.

We use 150 examples to train an initial CRF, 1018 to simulate user annotation, and 1019 to evaluate performance. Results are averaged over three random samples.

We first show that traditional active learning is beneficial in this domain. Figure 4 plots the average field F1 versus training set size, where the order in which examples are labeled is either random (*Random*), by order of least confidence (*LeastConfidence*), or by the query-by-committee method (*QBC*, with three committee members). Results are averaged over three random trials, with standard error bars as indicated. This figure demonstrates that the order in which examples are labeled can affect learning efficiency. For example, *LeastConfidence* requires approximately 300 fewer training examples to achieve the same F1 performance as *Random*.

Interestingly, the more sophisticated and computationally expensive *QBC* method is outperformed by the straight-forward confidence measure. A likely reason for this result is that because CRFs require a substantial number of labeled examples

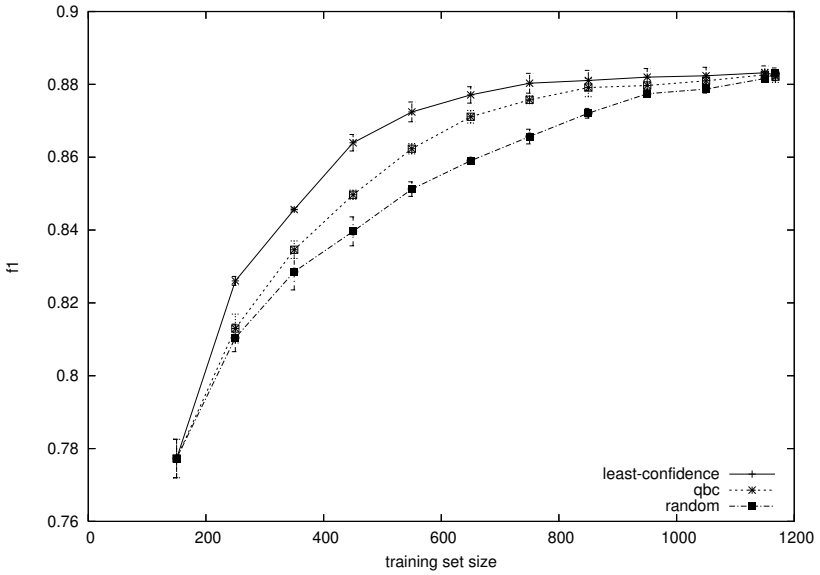


Fig. 4. Testing label F1 as a function of training set size, with standard error bars over three trials. *LeastConfidence* outperforms both query-by-committee (*qbc*) and the random baseline.

to perform well, dividing the small training set among three CRFs results in high-variance models that do not give reliable estimates of overall performance. Because of its simplicity and superior performance, *LeastConfidence* is the only ranking function used in the following experiments.

Note that in Figure 4 each labeled example (e.g. a paragraph) must be manually labeled by the annotator. This figure assumes each example requires the same amount of user effort to label (namely, one unit of cost). However, because each example is a *variable-length* sequence of labels, different examples will incur different labeling costs. Moreover, using the corrective feedback methods we have presented, even examples of the same length may have different labeling costs. Thus, we desire a more explicit measure of labeling effort. In the next experiments, we examine how F1 varies with the number of annotation actions.

We compare two competing methods. *Baseline* presents the user with the top prediction, and the user must hand annotate all corrections. The other method is the learning framework advocated in this paper, which presents the user with k possible segmentation, and interactively solicits label corrections. We vary k from 1 to 4. Constrained inference and correction propagation are used in one round of interactive labeling. *Note that all of these methods use LeastConfidence to rank the unlabeled examples*. The difference is the *interaction* that takes place when labeling each example.

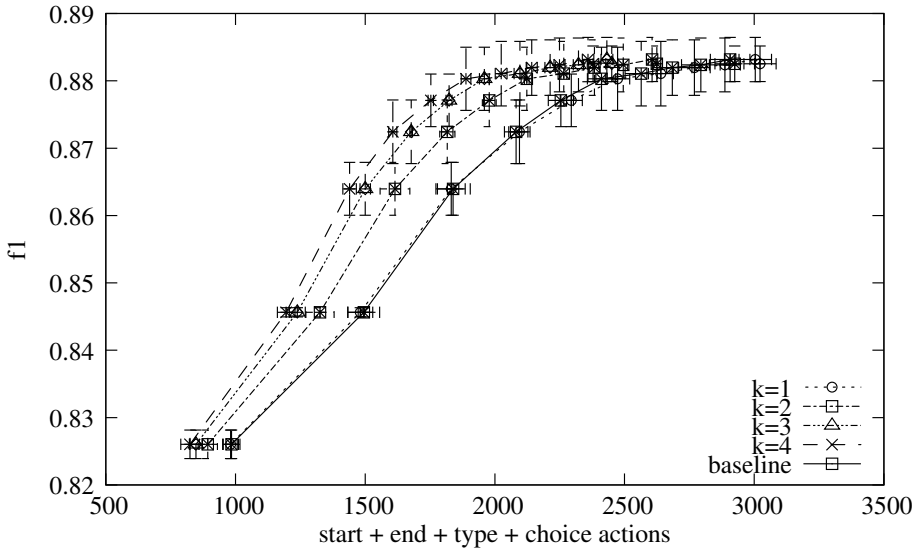


Fig. 5. Testing label F1 as a function of the total number of annotation actions. At $k = 4$, performance plateaus with roughly 800 fewer actions than the baseline.

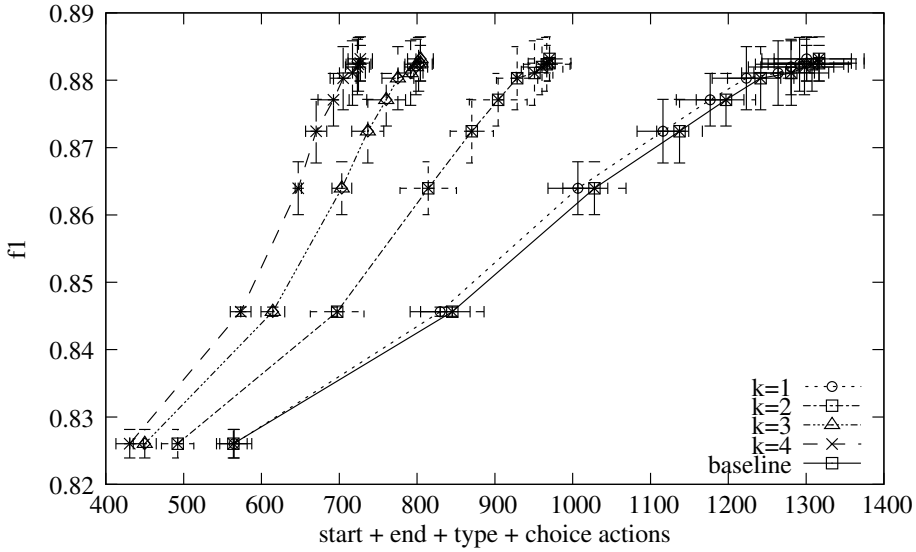


Fig. 6. Testing label F1 as a function of the total number of segmentation actions. The interactive system with $k = 4$ requires just over half the number of segmentation actions of the baseline.

Figure 5 compares these methods, measuring the total number of annotation actions required for each F1 value. The interactive framework outperforms the baseline consistently. On average, interactive labeling with $k = 4$ requires 21% fewer actions than *baseline*.

Note that $k = 1$ closely tracks the *baseline* performance. This suggests that when we restrict the user corrections to *type* actions only, there are not enough errors fixed by correction propagation to overcome the additional cost of a round of user interaction. This is further confirmed by the fact that performance increases with k.

	start + end	*type*	*choice*	*start + end + type*	*total*
$k = 1$	1.28	1.07	**0.62**	2.35	2.97
$k = 2$	0.95	0.92	0.70	1.87	2.58
$k = 3$	0.79	0.88	0.74	1.67	2.41
$k = 4$	**0.71**	0.86	0.76	**1.58**	**2.34**

Table 7
The average number of actions to label a single record, averaged over 1018 records, where each record has on average 6.2 fields. As k increases, we convert more and more segmentation actions into type and choice actions. Results for $k = 4$ show a 22% reduction in actions over $k = 1$. Boldfaced results are significantly lower than all other column values (paired-t test, $p = 0.05$).

However, as k increases, we notice diminishing marginal returns in savings. This suggests a trade-off between the difficulty of examining k labelings and the savings realized by reducing the user action to a simple multiple-choice selection.

To demonstrate the reduction in segmentation labeling, Figure 6 displays the number of segmentation actions (*start* or *end*) needed to achieve a particular F1 value. On average across the sampled F1 values, interactive labeling with $k = 4$ requires 42% fewer segmentation actions.

Note the steep learning curve of the interactive method. This suggests that the CRF's poor segmentation performance early in training is quickly overcome. The result is that after a small number of actions, the annotator can reduce the number of segmentation actions needed to train the CRF, and instead mostly provide *type* annotation.

Table 7 displays the average number of actions required to label each record for different values of k. These results are compatible with the trends in Figures 5 and 6. Note that the increase in *choice* actions as k increases is expected, since there are many examples where the correct labeling is in the top k choices. The advantage of this framework is that the increase in the number of *choice* actions is outweighed by the reduction in other actions. Note also that this reduction in effort is manifest even assuming all actions incur the same cost. If we assume that boundary annotation is more costly than *type* annotation, these difference will be even more pronounced.

These experiments have demonstrated that by providing the user with efficient corrective feedback mechanisms, we can decrease the labeling effort required to train a high-accuracy extractor.

5 Related Work

This paper unifies previous work introducing interactive information extraction and cost-sensitive active learning (Kristjannson et al., 2004; Culotta and McCallum, 2005). In addition, we provide novel scoring functions to maximize correction propagation (Section 3.3), provide more thorough evaluation of their effectiveness (Section 3.6.2), and compare against additional active learning methods (Section 4.3).

Others have studied efficient ways to interactively train an extraction system (Cardie and Pierce, 1998; Caruana et al., 2000); however, these methods do not use partially labeled examples to reduce labeling effort and do not use the corrective feedback methods we propose here. Instead, partially correct annotations are instead simply marked as incorrect.

There has also been work in the human-computer interaction literature on designing interfaces to support error correction for speech and handwriting recognition systems (Mankoff and Abowd, 1999; Suhm et al., 1999). They motivate the importance of confidence estimation and user feedback, but consider neither correction propagation nor active learning systems.

Active learning in general is a widely-researched area that mainly investigates different forms of scoring functions to rank unlabeled examples, as we discussed in Section 4.3. These functions can be coarsely divided into *uncertainty-based* methods, *committee-based* methods, and *error minimization* methods. Uncertainty-based methods rank examples according to how much confidence the learner has in its prediction (Lewis and Catlett, 1994; Scheffer et al., 2001). The *LeastConfidence* method discussed in Section 4.3 is an example of this approach. Notably, recent work by Schein (2005) has shown that simple uncertainty-based methods can be competitive or superior to committee-based methods. Committee-based methods construct a committee of learners and rank examples by the amount of disagreement among them (Freund et al., 1997; McCallum and Nigam, 1998; Argamon-Engelson and Dagan, 1999). The *QBC* method presented in Section 4.3 is an example of this approach. Query-by-committee methods have also been extended to multi-view and co-testing domains (Muslea et al., 2003; Ghani et al., 2003). Finally, error minimization methods attempt to directly minimize the expected classification error on future test examples (Cohn et al., 1995; Roy and McCallum, 2001). While this approach provides nice theoretical guarantees, it is computationally expensive and requires a number of approximations to be effective in practice.

To the best of our knowledge, the active learning framework we propose is the first that (1) is sensitive to the *difficulty* of labeling each training example (2) uses partially labeled examples to reduce this labeling difficulty, and (3) uses efficient corrective feedback mechanisms (such as correction propagation) to reduce user

effort. Below we discuss other active learning methods that are more closely related to our approach.

Thompson et al. (1999) present an active learning system for information extraction and parsing, which are instances of structured learning tasks. While they demonstrate the advantage of active learning for these tasks, they require the annotator to fully label each training example, which is precisely what this paper aims to avoid.

Vlachos (2006) has recently presented an active learning method that employs an unsupervised learning algorithm to partially label examples, which are then corrected by the user. This can be seen as an unsupervised analog to our previous work (Kristjannson et al., 2004; Culotta and McCallum, 2005). Three central distinctions in our work is that we consider confidence at the *field* rather than token level, we allow *correction propagation* to reduce labeling effort, and we provide estimates of user effort per annotation.

Baldridge and Osborne (2004) consider active learning to annotate sentences with parse trees. The annotator is presented with the top n predicted parse trees for a sentence. If the true parse is in the top n predicted parses, the user may select that parse. Otherwise, the user must manually annotate the sentence. This differs from our work in that they do not leverage information from partially correct parses, and correction propagation is not considered.

Additionally, Anderson and Moore (2005) evaluate various objective functions for active learning for HMMs. Our mutual information scoring function (Equation 6) used to maximize correction propagation can be seen as a variant of the entropy loss function used in their work.

Confidence prediction itself is also an under-studied aspect of information extraction, although it has been investigated in document classification (Bennett, 2000), speech recognition (Gunawardana et al., 1998), and machine translation (Gandrabur and Foster, 2003). Much of the previous work in confidence estimation for information extraction comes from the active learning literature. Scheffer et al. (2001) derive confidence estimates using hidden Markov models in an information extraction system; however, they do not estimate the confidence of entire fields, only singleton tokens. The token confidence is estimated by the difference between the probabilities of its first and second most likely labels, whereas our constrained forward-backward (Culotta and McCallum, 2004) considers multi-token fields, and the full distribution of all suboptimal paths. Scheffer et al. (2001) also explore an idea similar to constrained forward-backward to perform Baum-Welch training with partially labeled data, where a limited number of labels provide constraints. However, these constraints are again for singleton tokens only. Analogs of constrained Viterbi have been used in bioinformatics to find sub-optimal alignments of RNA sequences (Zuker, 1991), and in speech recognition to train HMMs when the word sequence is known but the sub-phone sequence is not (Franzini et al., 1990).

6 Conclusions and Future Work

We have presented a framework for corrective feedback and persistent learning for information extraction and have demonstrated its value empirically through the simulated correction of real data.

While these simulations are effective for evaluation, the best evaluation is a complete user study. Performance could be measured by the actual time it takes users to train an accurate system and correct its predictions. This more costly evaluation method also requires addressing issues of user interface design and annotator variability, but it is ultimately the most direct evaluation procedure.

Additionally, while our proposed active learning system makes no assumptions about the underlying ranking function, there may exist ways to construct a ranking function that has a preference for examples that are easy to label.

While we have restricted our empirical study to information extraction, the ideas presented here can be applied to other structured learning domains. The main principles we have advocated in this paper are the following:

- Active learning systems should leverage partially labeled examples to reduce annotation effort.
- When a user correction is provided, predictions should be updated in real-time to enable *correction propagation* savings.
- Users should be directed to fields that are either most likely to need correction, or most likely to lead to correction propagation.
- To better reflect user effort, evaluation of active learning systems should consider the number of user actions, not simply the number of labeled examples.

This work can be seen as a way to facilitate the wide-spread use of machine learning algorithms for structured prediction. These algorithms often require additional training examples to be personalized to each user. Therefore, the easier it is for a system to be trained and corrected by the user, more likely it is that the system will be well-received, frequently used, and accurate.

7 Acknowledgments

We thank the anonymous reviewers for helpful suggestions on previous drafts of this article. We would also like to thank David Parkinson for providing labeling and parsing tools and valuable discussions. This work was supported in part by the Center for Intelligent Information Retrieval, in part by U.S. Government contract #NBCH040171 through a subcontract with BBNT Solutions LLC, in part by The

Central Intelligence Agency, the National Security Agency and National Science Foundation under NSF grant #IIS-0326249, and in part by the Defense Advanced Research Projects Agency (DARPA), through the Department of the Interior, NBC, Acquisition Services Division, under contract number NBCHD030010. Any opinions, findings and conclusions or recommendations expressed in this material are the author(s) and do not necessarily reflect those of the sponsor.

References

Anderson, B., Moore, A., 2005. Active learning for hidden markov models: Objective functions and algorithms. In: ICML.

Argamon-Engelson, S., Dagan, I., 1999. Committee-based sample selection for probabilistic classifiers. Journal of Artificial Intelligence, 335–360.

Baldridge, J., Osborne, M., 2004. Active learning and the total cost of annotation. In: In Proceedings of the Conference on Empirical Methods in Natural Language Processing. Barcelona, Spain.

Bennett, P. N., September 2000. Assessing the calibration of naive bayes' posterior estimates. Tech. Rep. CMU-CS-00-155, Computer Science Department, School of Computer Science, Carnegie Mellon University.

Cardie, C., Pierce, D., 1998. Proposal for an interactive environment for information extraction. Tech. Rep. TR98-1702, Cornell University.

Caruana, R., Hodor, P., Rosenberg, J., August 2000. High precision information extraction. In: KDD-2000 Workshop on Text Mining.

Cohn, D. A., Ghahramani, Z., Jordan, M. I., 1995. Active learning with statistical models. In: Tesauro, G., Touretzky, D., Leen, T. (Eds.), Advances in Neural Information Processing Systems. Vol. 7. The MIT Press, pp. 705–712.

Culotta, A., McCallum, A., 2004. Confidence estimation for information extraction. In: Human Langauge Technology Conference (HLT 2004). Boston, MA.

Culotta, A., McCallum, A., 2005. Reducing labeling effort for stuctured prediction tasks. In: Twentieth National Conference on Artificial Intelligence. Pittsburgh, PA, pp. 746–751.

Franzini, M., Lee, K. F., Waibel, A., 1990. Connectionist viterbi training: a new hybrid method for continuous speech recognition. In: Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). Vol. volume 1.

Freund, Y., Seung, S., Shamir, E., Tishby, N., 1997. Selective sampling using the query by committee algorithm. Machine Learning 28, 133–168.

Gandrabur, S., Foster, G., 2003. Confidence estimation for text prediction. In: Proceedings of the Conference on Natural Language Learning. Edmonton, Canada.

Ghani, R., Jones, R., Mitchell, T., Riloff, E., 2003. Active learning for information extraction with multiple view feature sets. In: ICML.

Gunawardana, A., Hon, H., Jiang, L., 1998. Word-based acoustic confidence measures for large-vocabulary speech recognition. In: Proc. ICSLP-98. Sydney, Aus-

tralia, pp. 791–794.

Jaynes, E. T., 1979. Where do we stand on maximum entropy? In: Levine, R. D., Tribus, M. (Eds.), The Maximum Entropy Formalism. MIT Press, Cambridge, MA, pp. 15–118.

Kristjannson, T., Culotta, A., Viola, P., McCallum, A., 2004. Interactive information extraction with conditional random fields. Nineteenth National Conference on Artificial Intelligence.

Kushmerick, N., Weld, D. S., Doorenbos, R. B., 1997. Wrapper induction for information extraction. In: IJCAI. p. 729.

Lafferty, J., McCallum, A., Pereira, F., 2001. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In: Proc. 18th International Conf. on Machine Learning. Morgan Kaufmann, San Francisco, CA, pp. 282–289.

Lewis, D. D., Catlett, J., 1994. Heterogeneous uncertainty sampling for supervised learning. In: Cohen, W. W., Hirsh, H. (Eds.), Proceedings of ICML-94, 11th International Conference on Machine Learning. Morgan Kaufmann Publishers, San Francisco, US, New Brunswick, US, pp. 148–156.

Liu, D. C., Nocedal, J., 1989. On the limited memory BFGS method for large scale optimization. Math. Programming 45 (3, (Ser. B)), 503–528.

Mankoff, J., Abowd, G. D., 1999. Error correction techniques for handwriting, speech, and other ambiguous or error prone systems. Tech. rep., GVU Center and College of Computing Georgia Institute of Technology.

McCallum, A., 2003. Efficiently inducing features of conditional random fields. In: Nineteenth Conference on Uncertainty in Artificial Intelligence.

McCallum, A., Li, W., 2003. Early results for named entity recognition with conditional random fields, feature induction and web-enhanced lexicons. In: Hearst, M., Ostendorf, M. (Eds.), HLT-NAACL. Association for Computational Linguistics, Edmonton, Alberta, Canada.

McCallum, A., Nigam, K., 1998. Employing em in pool-based active learning for text classification. In: Proceedings of the 15th International Conference on Machine Learning. pp. 359–367.

Muslea, I., Minton, S., Knoblock, C., 2003. Active learning with strong and weak views: a case study on wrapper induction. In: Proceedings of International Joint Conference on Atificial Intelligence. pp. 415–420.

Pinto, D., McCallum, A., Wei, X., Croft, W. B., 2003. Table extraction using conditional random fields. In: SIGIR '03: Proceedings of the Twenty-sixth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval.

Rabiner, L., 1989. A tutorial on hidden Markov models. In: IEEE. Vol. 77. pp. 257–286.

Roy, N., McCallum, A., 2001. Toward optimal active learning through sampling estimation of error reduction. In: Proceedings of the 18th International Conference on Machine Learning. pp. 441–448.

Scheffer, T., Decomain, C., Wrobel, S., 2001. Active hidden markov models for information extraction. In: Advances in Intelligent Data Analysis, 4th International

Conference, IDA 2001.
Schein, A. I., 2005. Active learning for logistic regression. Ph.D. thesis, University of Pennsylvania.
Schwartz, R., Chow, Y.-L., 1990. The n-best algorithms: an efficient and exact procedure for finding the n most likely sentence hypotheses. In: International Conference on Acoustics, Speech, and Signal Processing (ICASSP-90).
Sha, F., Pereira, F., 2003. Shallow parsing with conditional random fields. In: Hearst, M., Ostendorf, M. (Eds.), HLT-NAACL: Main Proceedings. Association for Computational Linguistics, Edmonton, Alberta, Canada, pp. 213–220.
Suhm, B., Myers, B. A., Waibel, A., 1999. Model-based and empirical evaluation of multimodal interactive error correction. In: CHI.
Sutton, C., McCallum, A., 2006. An introduction to conditional random fields for relational learning. In: Getoor, L., Taskar, B. (Eds.), Introduction to Statistical Relational Learning. MIT Press, to appear.
Thompson, C. A., Califf, M. E., Mooney, R. J., 1999. Active learning for natural language parsing and information extraction. In: Proc. 16th International Conf. on Machine Learning. Morgan Kaufmann, San Francisco, CA, pp. 406–414.
Vlachos, A., 2006. Active annotation. In: Proceedings of the EACL 2006 Workshop on Adaptive Text Extraction.
Zuker, M., 1991. Suboptimal sequence alignment in molecular biology: Alignment with error analysis. Journal of Molecular Biology 221, 403–420.