# Automatic segmentation of text into structured records

Vinayak Borkar[*] Kaustubh Deshmukh[†] Sunita Sarawagi[‡]
Indian Institute of Technology, Bombay

## ABSTRACT

In this paper we present a method for automatically segmenting unformatted text records into structured elements. Several useful data sources today are human-generated as continuous text whereas convenient usage requires the data to be organized as structured records. A prime motivation is the warehouse address cleaning problem of transforming dirty addresses stored in large corporate databases as a single text field into subfields like "City" and "Street". Existing tools rely on hand-tuned, domain-specific rule-based systems.

We describe a tool DATAMOLD that learns to automatically extract structure when seeded with a small number of training examples. The tool enhances on Hidden Markov Models (HMM) to build a powerful probabilistic model that corroborates multiple sources of information including, the sequence of elements, their length distribution, distinguishing words from the vocabulary and an optional external data dictionary. Experiments on real-life datasets yielded accuracy of 90% on Asian addresses and 99% on US addresses. In contrast, existing information extraction methods based on rule-learning techniques yielded considerably lower accuracy.

## 1. INTRODUCTION

Several useful structured data sources exist today as continuous text primarily because humans find it easier to create them that way. Examples are postal addresses, bibliography records, classified ads and phone lists. All these applications have the property that the data has an implicit schema consisting of a set of attributes — for example postal addresses comprise of elements like "street", "city" and "zip" and bibliography records comprise of elements like "author-names", "title" and "page-numbers". However, the text string itself is generated by concatenating values of the different attributes without any explicit separator amongst them. The order of attributes is not fixed and not all attributes are present in all instances.

[*]Supported partially by the IBM Research Fellowship

[†]Author's current affiliation: University of Washington Seattle, USA

[‡]Contact author: sunita@it.iitb.ernet.in

These properties make the problem of extracting structure from such text different from both the general problem of information extraction from natural language documents and the popular problem of generating wrappers to extract structure from HTML documents. In the first case, the goal is to extract semantic entities from natural language documents based on linguistic constraints [14, 29]. Often only a small fraction of the text forms part of the structured schema. In the second case of wrapper generation, syntactic cues present as HTML tags are used extensively to define rules for structure extraction [17, 6, 11, 20, 25, 7, 22, 24]. The HTML tags in pages tend to be highly regular because the pages are often machine-generated [17, 7]. In contrast, in our case most data is human generated and hence highly irregular.

We elaborate on two applications, where text segmentation is useful.

### 1.1 International postal addresses

Large customer-oriented organizations like banks, telephone companies and universities collect millions of unformatted address records. Each address record is typically provided by a different person and thus subject to the variation in style that occurs from person to person. During warehouse construction, all these addresses are cleaned and converted to a standard consistent format with duplicates removed. This is a multi-step process. The first step, called *Address Elementization* [15], is where addresses are segmented into a fixed set of structured elements. For example an address string `''18100 New Hamshire Ave. Silver Spring, MD 20861''` can be segmented into five structured elements as follows:

| | | |
|---|---|---|
| House Number | : | 18100 |
| Street Name | : | New Hamshire Ave. |
| City | : | Silver Spring |
| State | : | MD |
| Zip | : | 20861 |

The second step called *Address Standardization* is where abbreviations (like "Ave.") get converted to a canonical format and spelling mistakes get corrected. This is followed by the *Deduplication* or *Householding* phase where all addresses belonging to the same household are brought together. The quality of both these phases can be considerably enhanced by first elementizing the addresses correctly.

| # | Address text<br>[Segmented address] |
|---|---|
| 0 | M. J Muller, 71, route de Longwy L-4750 PETANGE<br>[recipient: M. J Muller] [House#: 71,]<br>[Street: route de Longwy] [Zip: L-4750] [city:PETANGE] |
| 1 | Viale Europa, 22 00144-ROMA RM<br>[Street: Viale Europa,] [House#: 22] [City: ROMA]<br>[Province: RM] [Zip: 00144-] |
| 2 | 7D-Brijdham Bangur Nagar Goregaon (W) Bombay 400 090<br>[House#: 7D-] [Building: Brijdham]<br>[Colony: Bangur Nagar] [Area: Goregaon (W)]<br>[City: Bombay] [Zip: 400 090] |
| 3 | 18100 New Hamshire Ave. Silver Spring, MD 20861<br>[House#: 18100], [Street: New Hamshire Ave.],<br>[City: Silver Spring,], [State: MD], [Zip: 20861] |

**Table 1: Sample addresses from different countries. The first line shows the unformatted address and the second line shows the address broken into its elements**

Existing commercial tools [10] rely on hand-written rules coupled with a massive database of cities, states and zip codes. These methods work for the region they are developed and do not extend to other domains. A lot of manual work has to be performed in rewriting these rules when shifting domains. Postal addresses in different parts of the world often have drastically different structures. In some countries zip codes are five digit numbers whereas in others they are allowed to have strings. In Table 1 we show some example addresses[1] along with their elementized forms from different regions of the world. Not only do address formats differ from country to country, even within a city, addresses can have widely different formats. The problem is more challenging in large developing countries that do not follow the templatized western address schemes. There is no uniform numbering of buildings, the reliance on ad hoc descriptive landmarks is common, state abbreviations are not standardized, spelling mistakes are rampant and zip codes optional. Given the wide variety of patterns, it is clear that manual rule-based tools will not scale with the expanding globalization of business.

In spite of the commercial importance of this problem and the challenges it offers, research in the area has been limited because researchers view this as a largely labor-intensive task. One exception is the merge/purge problem for deduplicating address records [12, 23]. We believe and show in the paper that address elementization is another problem that can benefit from principled research.

## 1.2 Bibliography records

A second motivating example is cleaning bibliographic records for the construction of citation indices like Citeseer[2] [19]. This requires extracting from the end of each research paper, the list of references and matching each reference to a database of entries. References appear in different formats in different documents. As an example, we show in the table below the various forms in which the classical "Selinger, Query optimization" paper is referred [3].

1. McGraw-Hill. Selinger, P.; Astrahan, M.; Chamberlin, D.; Lorie, R.; and Price, T. 1979. Access path selection in a relational database management system. In SIGMOD '79.
2. In VLDB-96, 251–262. Selinger, P.; Astrahan, M.; Chamberlin, D.; Lorie, R.; and Price, T. 1979. Access path selection in a relational database management system. In SIGMOD '79.
3. Patricia G. Selinger, et al. Access path selection in a relational database management system. In Proceedings of the ACM SIGMOD Conference, pages 23–34, 1979.
4. Access path selection in a relational database management system. In Proc. of the ACM SIGMOD Conf. on Management of Data, pages 23#34, Boston, USA, May 1979.
5. Price #1979#. #Access path selection in a relational database management system," ACM SIGMOD International Conf. on Management of Data, pp. 23#34., 1979.
6. SIAM Journal of Computing, 17(6):1253–1262. Selinger, G., Astrahan, M., Chamberlin, D., Lorie, R., and Price, T. (1979). Access Path Selection in a Relational Database Management System. In Proceedings of the ACM SIGMOD International Conference on Management of Data, pages 23–34.
7. Selinger et al. Access Path Selection in a Relational Database System." In Proceedings of the ACM SIGMOD International Conference on the Management of Data, 1979.

Existing methods for matching records rely on rule-based, manual heuristics. The matching algorithm can be made more robust after individual fields are extracted as observed by the authors of Citeseer in [19]:

> "While CiteSeer's current algorithm is sufficient for practical use, it could be improved in many ways. For example, the use of machine learning techniques and probabilistic estimation based on training sets of known bibliographic data may boost performance."

The Selinger paper Citeseer had 283 total references. Citeseer wrongly classified them as thirteen different papers, seven of which are shown in the above list. They indeed appear very different overall. However, a closer examination reveals that in all references the "Title" and "Year" fields are the same and in most at least one author is common. Hence a field level match on Title, Year and Author instead of an approximate record level match would be more accurate. The extracted fields could also enable a better structured search instead of the current record-level search.

The field extraction problem is non-trivial in this case because of the high variance in the structure of the record. Author is not the first field in all cases. The year field appears within parenthesis for some, at the end for others. Comma separates some fields but comma is also used to separate last names from first names. A dot usually appears at the end of title but also appears after abbreviations. Several fields like location, month and page-numbers are optional.

## 1.3 Our Approach

We have developed a tool DATAMOLD for automatically segmenting such data starting from a small seed set of example segmented records. The core of DATAMOLD is a powerful statistical technique called Hidden Markov Modeling (HMM).

[1]obtained from `http://bitboost.com/ref/international-address-formats.html`, November 2000
[2]`http://citeseer.nj.nec.com/cs`, Nov 2000
[3]Obtained from Citeseer by searching for "Access and Path and Selection and Relational" (October 2000)

### 1.3.1 Hidden Markov Models

A Hidden Markov Model (HMM) is a probabilistic finite state automaton [26, 27] comprising of a set of states, a finite dictionary of discrete output symbols, and edges denoting transitions from one state to another. Each edge is associated with a transition probability value. Each state emits one symbol in the dictionary from a probability distribution for that state. There are two special states: a start state and an end state. Beginning from the start state, a HMM generates an output sequence $O = o_1, o_2, \ldots, o_k$ by making $k$ transitions from one state to the next until the end state is reached. The $i^{th}$ symbol $o_i$ is generated by the $i^{th}$ state based on that state's probability distribution of the dictionary symbols. In general, an output sequence can be generated through multiple paths each with some probability. The sum of these probabilities is the total probability of generating the output sequence. The HMM thus induces a probability distribution on sequences of symbols chosen from a discrete dictionary. The training data helps learn this distribution. During testing, the HMM outputs the most probable state transitions that could have generated an output sequence.

HMMs, while relatively new to the structure extraction task, have been used with much success for speech and handwriting recognition tasks [27] and for natural language tasks like parts-of-speech tagging [16]. In spite of the general principles being known, applying it to our text segmentation problem required new enhancements and experimentation & validation on real-life data. We list our main enhancements.

- We developed a nested model for learning the structure of the HMM — the outer HMM captures the sequencing relationship amongst elements and the inner HMMs learn the finer structure within each element. An important design issue in deploying a HMM is choosing the right structure of the model. There is no established method for doing this optimally. Our nested model provides a practical method to search amongst the exponential number of possibilities.
- We introduce the concept of a taxonomy on the symbols (words, numbers, delimiters) appearing in the training sequence and show how to generalize the dictionary of each HMM state to a level that provides the highest accuracy.
- We show how to integrate an external database into the basic HMM model. We propose an *optimal* modification to the classical Viterbi algorithm (used for finding the most likely path in an HMM) to incorporate relationships imposed by an external database.

The final model after incorporating all the enhancements, provides a powerful segmentation tool that combines information about different aspects of the record, including,

- Characteristic words in each element: The dictionary along with the symbol hierarchy learns characteristic words in each element intuitively capturing patterns of the form "words like street appear in road-names" and "house-numbers usually consist of digits".
- Number of symbols in each element: Sometimes, records can be segmented based on typical lengths of different elements. For example, "title" fields are long whereas "location" names are small. The inner HMMs and the transition probabilities capture this information.

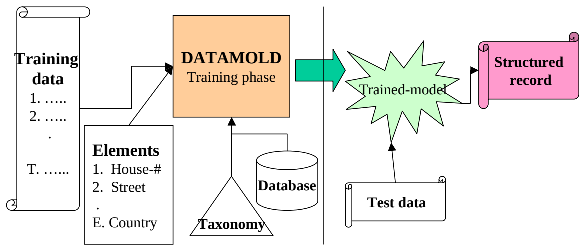

**Figure 1: An overview of the working of DATAMOLD.**

- Partial ordering amongst elements: Most often there is a partial ordering amongst elements: for example, "house number" appears earlier on in the address record than "zipcode". The transition probabilities and the structure of the outer HMM helps learn this information.
- Non-overlapping elements: Our approach attempts to simultaneously identify all the elements of a record. Thus, the different inner HMMs corroborate each other's finding to pick the segmentation that is globally optimal. This is in contrast to systems that extract each element in isolation, for example, as proposed in [5, 8].

Experiments on real-life address and bibliography datasets yield very encouraging results. We achieved accuracies of 99.2% on US addresses after training on just 50 instances and 89% on a considerably more complicated Asian address dataset. In contrast, a well-known information extraction method based on rule-learning yielded accuracy of 97% and 50% on the two datasets respectively. On a bibliography dataset the accuracy was 87% (with 100 training instances) with our method and 43% with the rule learning method.

The HMM-based approach thus provides a principled way to combine all the above information to segment data based on the maximization of a single objective function. Most tools proposed recently are rule-based systems [1, 5, 25, 17] that rely on heuristics to control the order in which rules are fired and extract each element in isolation exploiting only some of the above information. Other advantages with the HMM approach is that it can handle new data robustly, is computationally efficient and is easy for humans to interpret and tweak.

*Outline.* The rest of the paper is organized as follows. In Section 2 we present a detailed description of DATAMOLD. In Section 3 we present experimental results. In Section 4 we present related work and finally conclude in Section 5.

## 2. SEGMENTING TEXT USING DATAMOLD

In Figure 1 we present an overview of the working of DATAMOLD. The input to DATAMOLD is a fixed set of $E$ elements of the form "House #", "Street" and "City" and a collection of $T$ example addresses that have been segmented into one or more of these elements. We do not assume any fixed ordering amongst the elements nor are all elements required to be present in all addresses. Two other optional inputs to the training process are first, a taxonomy on the syntax of symbols in the training data and second, a database of relationship amongst symbols. The role of these additional information is discussed later in Sections 2.4 and 2.5. DATA-

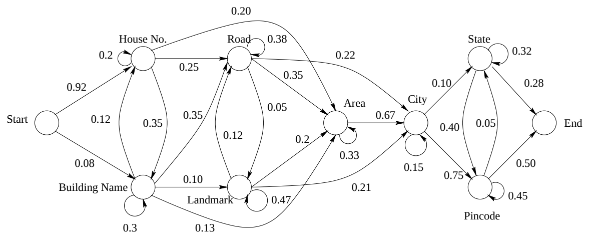

**Figure 2: Structure of a typical naive HMM**

MOLD uses the example segmented records to output a model that when presented with any unseen text segments it into one or more of its constituent elements.

## 2.1 HMMs for text segmentation

The basic HMM model as described in Section *1.3.1* consists of:

- a set of $n$ states,
- a dictionary of $m$ output symbols,
- an $n \times n$ transition matrix $A$ where the $ij^{th}$ element $a_{ij}$ is the probability of making a transition from state $i$ to state $j$, and
- a $n \times m$ emission matrix $B$ where entry $b_{jk}$ denotes the probability of emitting the $k$-th output symbol in state $j$.

This basic HMM model needs to be augmented for segmenting free text into the constituent elements. Let $E$ be the total number of elements into which the text has to be segmented. Each state of the HMM is marked with exactly one of these $E$ elements, although more than one state could be marked with the same element. The training data consists of a sequence of element-symbol pairs. This imposes the restriction that for each pair $\langle e, s \rangle$ the symbol $s$ can only be emitted from a state marked with element $e$.

In Figure 2 we show a typical HMM for address segmentation. The number of states $n$ is 10 and the edge labels depict the state transition probabilities ($A$ Matrix). For example, the probability of an address beginning with House Number is 0.92 and that of seeing a City after Road is 0.22. The dictionary and the emission probabilities are not shown for compactness.

*Training a HMM.* The parameters of the HMM comprising of the number of states $n$, the set of symbols in the dictionary $m$, the edge transition matrix $A$ and the emission probability matrix $B$ are learnt from training data. The training of an HMM has two phases. In the first phase we choose the structure of the HMM, that is, the number of states $n$ and edges amongst states and train the dictionary. Normally, training the dictionary is easy, it is just the union all words, digits and delimiters appearing in the training data. In Section 2.4 we present further refinements on the dictionary. In the second phase we learn the transition and emission probabilities assuming a fixed structure of the HMM. We first concentrate on this phase in Section 2.2 and discuss structure learning in Section 2.3.

*Using the HMM for testing.* Given an output symbol sequence $O = o_1, o_2, \ldots, o_k$, we want to associate each symbol with an element. Since each state in the HMM is associated with exactly one element, we associate each symbol with the state that emitted the symbol. Hence we need to find a path of length $k$ from the start state to the end state, such that the $i^{th}$ symbol $o_i$ is emitted by the $i^{th}$ state in the path. In general, an output sequence can be generated through multiple paths each with some probability. We assume the *Viterbi approximation* and say that the path having the highest probability is the one which generated the output sequence. Given $n$ states and a sequence of length $k$, there can be $O(k^n)$ possible paths that the sequence can go through. This exponential complexity is cut down to $O(kn^2)$ by the famous dynamic programming-based *Viterbi Algorithm* [27]. Readers familiar with the algorithm can skip the next section.

### 2.1.1 The Viterbi algorithm

Given an output sequence $O = o_1, o_2, \ldots, o_k$ of length $k$ and an HMM having $n$ states, we want to find out the most probable state sequence from the start state to the end state which generates $O$.

Let 0 and $n+1$ denote the special start and end states.

Let $v_s(i)$ be the probability of the most probable path for the prefix $o_1, o_2, \ldots o_i$ of $O$ that ends with state $s$.

We begin at the start state labeled 0. Thus, initially

$$v_0(0) = 1,\ v_k(0) = 0,\ k \neq 0$$

Subsequent values are found using the following recursive formulation:

$$v_s(i) = b_s(o_i) \max_{1 \le r \le n} \{a_{rs} v_r(i-1)\}, \quad 1 \le s \le n,\ 1 \le i \le k \quad (1)$$

where $b_s(o_i)$ is the probability of emitting the $i$-th symbol $o_i$ at state $s$ and $a_{rs}$ is the transition probability from state $r$ to state $s$. The maximum is taken over all states of the HMM.

The probability of the most probable path that generates the output sequence $O$ is given by

$$v_{n+1} = \max_{1 \le r \le n} a_{r(n+1)} v_r(k)$$

the actual path can be gotten by storing the argmax at each step. This formulation can be easily implemented as a dynamic programming algorithm running in $O(kn^2)$ time.

## 2.2 Learning transition and emission probabilities

The goal of the training process is to learn matrices $A$ and $B$ such that the probability of the HMM generating these training sequences is maximized. Each training sequence consists of a series of element-symbol pairs. The structure of the HMM is fixed and each state is marked with one of the $E$ elements. This restricts the states to which the symbols of a training sequence can be mapped. Consider two cases. In the first case, there is exactly one path from the start to the end state for all training sequences. The second case is where there is more than one valid path. All HMM structures we discuss in the paper satisfy the first condition of having a unique path. Hence we do not discuss this case further. In the first case, the transition probabilities can be calculated using the *Maximum Likelihood* approach on all training sequences. Accordingly, the probability of making a transition from state $i$ to state $j$ is the ratio of the number of transitions made from state $i$ to state $j$ in the training data to the total number of transitions made from state $i$. This can be written as:

$$a_{ij} = \frac{\text{Number of transitions from state } i \text{ to state } j}{\text{Total number of transitions out of state } i} \quad (2)$$

The emission probabilities are computed similarly. The probability of emitting symbol $k$ in state $j$ is the ratio of the number of times symbol $k$ was emitted in state $j$ to the total number of symbols emitted in the state. This can be written as:

$$b_{jk} = \frac{\text{Number of times the } k\text{-th symbol emitted at state } j}{\text{Total number of symbols emitted at state } j} \quad (3)$$

Computationally, training the $A$ and $B$ matrix involves making a single pass over all input training sequences, mapping each sequence to its unique path in the HMM and adding up the counts for each transition that it makes and output symbol it generates.

*Smoothing.* The above formula for emission probabilities needs to be refined when the training data is insufficient. Often during testing we encounter words that have not been seen during training. The above formula will assign a probability of zero for such symbols causing the final probability to be zero irrespective of the probability values elsewhere in the path. Hence assigning a correct probability to the unknown words is important. The traditional method for smoothing is Laplace smoothing [18] according to which Equation 3 will be modified to add one to the numerator and $m$ to the denominator. Thus, an unseen symbol $k$, in state $j$ will be assigned probability $\frac{1}{T_j+m}$ where $T_j$ is the denominator of Equation 3 and stands for the total number of tokens seen in state $j$. We found this smoothing method unsuitable in our case. An element like "road name", that during training has seen more distinct words than an element like "Country", is expected to also encounter unseen symbols more frequently during testing. Laplace smoothing does not capture this intuition. We use a method called absolute discounting. In this method we subtract a small value, say $x$ from the probability of all known $m_j$ distinct words seen in state $j$. We then distribute the accumulated probability equality amongst all unknown values. Thus, the probability of an unknown symbol is $\frac{m_j x}{m-m_j}$ and for a known symbol $k$ is $b_{jk} - x$ where $b_{jk}$ is as calculated in Equation 3. There is no theory about how to choose the best value of $x$. We pick $x$ as $\frac{1}{T_j+m}$.

## 2.3 Learning Structure

In general, it is difficult to get the optimal number of states in the HMM. We first present a naive model and later present a more elaborate nested model that is used in DATAMOLD.

### 2.3.1 Naive Model

A naive way to model the HMM is to have as many states as the number of elements $E$, and completely connect these $E$ states. To this we add a start state with transitions from it to every other state, and an end state with transitions to it, from every other state.

In Figure 2 we presented an example HMM for a typical address dataset of an Asian metropolis. For simplicity, only the important states have been shown. As mentioned earlier, the numbers on the edges represent the corresponding transition probabilities. The self loops are used to capture elements with more than one token.

This model captures the ordering relationship amongst elements. However, because it has just one state per element, it ignores any sequential relationship amongst words in the same element. For example, most road names end with words like "`Road`", "`Street`", or "`Avenue`". Treating an element as a single state does not capture this structure. Also for country names like "`New Zealand`" both "`New`" and "`Zealand`" will be outputs of the same state. This state will accept "`Zealand New`" with the same probability as "`New Zealand`". The other problem is that it learns only a limited kind of distribution on the number of words per element. For example, if 50% elements have one word and the rest 50% have three words each, then the naive model will create a single state with a self loop of probability 0.5. This accepts elements of length $1, 2, 3 \ldots k$ with probability $\frac{1}{2}, \frac{1}{4}, \frac{1}{8} \cdots \frac{1}{2^k}$ respectively. In contrast for the training data the corresponding probabilities are $\frac{1}{2}, 0, \frac{1}{2}, \ldots, 0$. We overcome these drawbacks in the next model.

### 2.3.2 Nested model

In this model, we have a nested structure of the HMM. Each element has its own inner HMM which captures its internal structure. An outer HMM captures the sequencing relationship amongst elements treating each inner HMM as a single state.

The HMM is learnt in a hierarchical manner in two stages. In the first stage we learn the outer HMM. In this stage, the training data is treated as a sequence of elements ignoring all details of the length of each element and the words it contains. These sequences are used to train the outer HMM.

In the second stage we learn the structure of the inner HMMs. The training data for each element is the sequence of all distinct tokens (word, delimiter, digit) in the element.

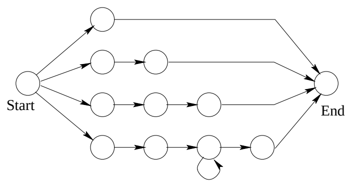

**Figure 3: A four length Parallel Path structure**

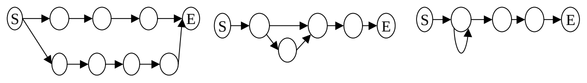

**Figure 4: Merging a four state path with a three state path**

An element typically has a variable number of tokens. For example, city-names most frequently have one token but sometimes have two or more tokens. We handle such variability automatically by choosing a parallel path structure (Figure 3) for each inner HMMs. In the Figure, the start and end states are dummy nodes to mark the two end points of an element. All records of length one will pass through the first path, length two will go through the second path and so on. The last path captures all records with four or more tokens. We next describe how the structure of the

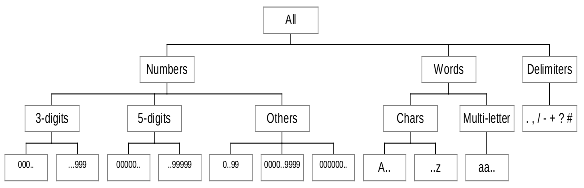

**Figure 5: An example taxonomy on symbols.**

HMM in terms of the number of paths and the position of the state with the self loop is chosen from the training data.

Initially, we create as many paths as the number of distinct token counts. This might leave some paths with insufficient training examples resulting in a poorly trained dictionary. We merge such paths to its neighboring path as follows. Starting with the longest path, we merge each path to the next smaller path as long as it improves the objective function described in the next paragraph. Merge of a $k$ state path to a $k-1$ state path as shown in the example in Figure 4 requires us to pick the one state that will not be merged. We try out all possible $k$ choices of this state and choose the one that gives the best value of the objective function. At the end we pick the largest path and replace any parallel path within it with a self-loop (as shown in the last diagram in Figure 4) so that paths longer than the longest path in the training data can be accepted.

*Objective function* **.** An inner HMM is good if it accepts the part of the symbol sequence belonging to itself and rejects the part not belonging to itself. Therefore, the best inner HMM cannot be found independently for each element. Another subtlety in choosing a good structure for the inner HMM is that, an inner HMM does not need to learn to reject all tokens – only the tokens that belong to an adjacent element.

We therefore learn each inner HMM in conjunction with others that are adjacent to it. Initially, all inner HMMs are unpruned. Starting from one end, we pick the HMM $h$ of the first element and HMMs of all other elements that transition to and from this element. We first attempt to prune $h$. For this, we truncate the training data to the part that is relevant to all the selected elements. Two paths of HMM $h$ are merged only if the accuracy of segmenting the training data does not decrease. After pruning $h$ to the right size, we prune the next inner unpruned HMM that it points to in the same manner and so on.

## 2.4 Hierarchical feature selection

One issue during the training phase is what constitutes the symbols in the dictionary. A reasonable first approach is to treat each distinct word, number or delimiter in the training data as a token. Thus, in the address `18100 New Hamshire Ave. Silver Spring, MD 20861` we have 10 tokens: six words, two numbers and two delimiters "," and ".". Intuitively, though we expect the specific number "18100" to be unimportant as far as we know that it is a number and not a word. Similarly, for the zip code field the specific value "20816" is not important; what matters perhaps is that it is a 5-digit number. How do we automatically make such decisions?

We propose that the features be arranged in a hierarchy. An example taxonomy is shown in Figure 5 where at the top-most level there is no distinction amongst symbols; at the next level they are divided into "Numbers", "Words" and "Delimiters"; "Numbers" are divided based on their length as "3-digit", "5-digit" or any other length numbers; and so on. Such taxonomy is part of a one-time static information input to DATAMOLD. The training data is used to automatically choose the optimum frontier of the taxonomy. Higher levels loose distinctiveness and lower levels are hard to generalize and require more training data. We need a middle ground.

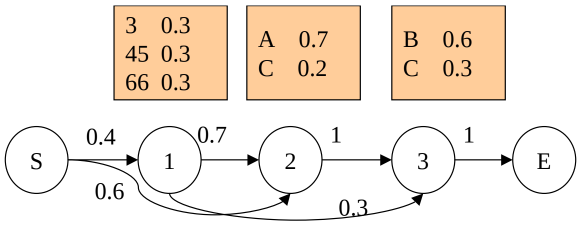

**Figure 6: An example HMM to motivate the need for feature selection.**

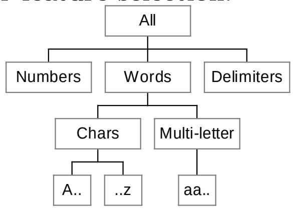

**Figure 7: Taxonomy of Figure 5 after being pruned for best performance.**

We motivate the need for feature selection through a small HMM in Figure 6 with three states and a start state and an end state. The dictionary of each state and the associated emission probability is shown in the box above each state. Thus, the dictionary of state 2 has two symbols A and C with non-zero probability 0.7 and 0.2 respectively whereas for state 1 only the three numbers 3, 45 and 66 had nonzero probability.

Each state assigns the same probability of $\alpha = 0.1$ to unknown symbols. Suppose, if we get a test sequence of the form (90, D). Then, this HMM will assign it to states (2,3) because the emission probability of symbols 90 and D that are unknown to all states will be the same and transition probability of the (2,3) is highest. Intuitively, however we expect digits to be assigned to state 1. Suppose instead through feature selection we transformed all individual numbers to a special token "#" that stood for all digits. Then dictionary of state 1 contains just one symbol "#" with probability 0.9. Then the probability of the path (1,3) becomes $0.4 \times 0.9 \times 0.3 \times 0.1$ that is higher than that of path (2,3) at $0.6 \times 0.1 \times 1.0 \times 0.1$. In contrast, if we attempt to convert the symbols $A, B, C$ to a single symbol "@" denoting all letters, then the dictionary of both states 2 and 3 become the same containing a single symbol "@" with probability 0.9. In this case, we have lost the distinction that in state 2 letter "A" is more likely and in state "3" letter "B" is more likely. In the extreme, if we replace all numbers and letters by the single symbol "All" at the topmost level, then it is like not having a dictionary at all. The best path is chosen based simply on the structure of the HMM and the edge transition probabilities. In this case all two and three token sequences will be mapped to states $(2, 3)$ and $(1, 2, 3)$ respectively.

The above example showed that it helps to generalize symbols in the training data to a higher level in some cases but not always. DATAMOLD uses the following method to choose the right level. The available segmented data is divided into two parts: training and validation. Normally, we set aside one-third of the total data for validation. First we train the dictionary with all symbols at their original detailed level as seen in the training part. Next we use the validation dataset to choose the right level of the taxonomy. The procedure is similar to how decision trees are pruned using cross-validation to avoid overfitting. Starting from the bottom we prune the tree at various levels in turn and check the accuracy on the validation data. The highest accuracy subtree is chosen. This process does not require training data to be scanned again since the higher level dictionaries can be formed from lower level symbols. Thus, each validation step is fast.

In Figure 7 we show an example frontier that yielded highest accuracy on one of our real-life address datasets for the feature tree of Figure 5. Thus all individual numbers are converted to a single special token representing numbers. All delimiters are converted to another special symbol and everything else is left as it is. This conversion is done during a preprocessing step on input data sequences.

The feature taxonomy is also used to modify the smoothing method described in Section 2.2. For unknown symbols, instead of doing absolute discounting over all the symbols in the dictionary, we find the first ancestor with non-zero probability and do absolute discounting over children of that feature. Thus, if we find an unknown symbol say “Fairyland” then we do absolute discounting only over multi-letter words in that state instead of all symbols. This leads to a closer approximation of the unknown probability.

## 2.5 Integrating a Partial Database

Sometimes, in addition to the training data we might have a database of richer semantic relationship amongst symbols of different elements. For address data we might have a hierarchical database of countries, the states in each country and the cities in each state. Similarly, for bibliography data, conference names, location and year could be related. Such information constraints the combination of values that are allowed in different elements and could be useful in finding the right assignment of symbols to states. For example, some address formats allow both the state name and country name to be optional. Thus, a city name could be followed by either a state or a country name. Suppose we get an address ending with (C,X) where C has been established to be a city name. If X occurs both in the state and country dictionary, the HMM may not be able to correctly pick the right element. However, this confusion would not arise if we had access to a database that established that country X has a city called C and that state X does not have any city called C.

Incorporating dependency information as implied by hierarchies is hard in HMMs because the output and transition probabilities of a state depends only on that state and is independent of the output symbols of the previous state. The Viterbi algorithm, used for efficiently finding the optimum path of a test sequence through a HMM, crucially relies on this property in its dynamic programming formulation. In this section, we present a modification of the algorithm for handling the above forms of dependencies.

This part requires an understanding of the Viterbi algorithm as explained in Section 2.1.1. We present our modification in the following section.

### 2.5.1 *Our modification to Viterbi*

As mentioned in Section 2.1.1, Viterbi finds the most probable path for a prefix $o_1, o_2, \ldots, o_i$ of $O$ ending at state $s$ using the following recursive formula.

$$v_s(i) = b_{s(o_i)} \max_{1 \le r \le n} \{a_{rs} v_r(i-1)\}, \quad 1 \le s \le n, \; 1 \le i \le k$$

where $v_s(i)$ is the probability of the most probable path for the prefix ending at state $s$. We modify the above formulation to restrict exploring paths that are invalid given the database of semantic relationships amongst symbols of different elements. In the modified algorithm, we model these semantic constraints as a pair of symbol-state assignments that are invalid. Suppose at the $i$th step we find that the assignment $\langle s, o_i \rangle$ conflicts with an earlier assignment $\langle s', o_j \rangle$ (for some $j < i$ ) in the best path from the start state to $s$. We change Equation 1 so that instead of taking a max over all states we disallow those that are invalid. We reduce the number of lookups by doing the checks only for the winning value. After finding the max on all states we check if the corresponding path is valid. If not, we take the second highest value, check for its validity and so on until a valid path is found. If all paths to $s$ are invalid than $v_s(i)$ is 0.

This modification ensures that the path output by Viterbi is valid. Unfortunately, the optimality of the solution is no longer guaranteed. The reason Viterbi works is that the best path for generating the $i$th output symbol is independent of the exact path from the first to $(i-1)$th symbol. We violate this basic assumption — an earlier assignment is affecting later assignments. We propose a second adjustment to ensure optimality in spite of such violations.

If in the Viterbi equation above if the best path to some state $r$ involved an assignment $\langle s', o_j \rangle$ (for some $j < i$ ) that conflicted with the $i^{th}$ assignment $\langle s, o_i \rangle$, we *re-evaluate* $v_r(i-1)$ while disallowing state $s'$ in the $j^{th}$ time step. Call this new value $v_r(i-1|\text{no } c_{rs}(i-1))$ where $c_{rs}(i-1)$ denotes the state in the best path $v_r(i-1)$ that conflicts with $\langle s, o_i \rangle$.

Thus, the value $v_s(i)$ is evaluated by taking the maximum over all states whose best paths do not conflict with assignment $\langle s, o_i \rangle$. If they do conflict for some state $r$, we reevaluate a different value $v_r(i-1|\text{no } c_{rs}(i-1))$ and evaluate max over that. Now, equation 1 changes to

$$v_s(i) = b_{s(o_i)} \max_{1 \le r \le n} \begin{cases} \{a_{rs} v_r(i-1) & \text{if no conflict with } \langle s, o_i \rangle \\ a_{rs} v_r(i-1|\{\text{no } c_{rs}(i-1)\}) & \text{otherwise}\} \end{cases} \quad (4)$$

In evaluating $v_r(i-1|\{\text{no } c_{rs}(i-1)\})$, some other assignment might be found to be illegal. That is appended to the set of disallowed assignments forming a list of size greater than one.

This modification returns the optimal valid path. Thus, any form of semantic constraint amongst symbols can be enforced. The constraints are closely integrated with the search for the best path in the HMM. This is considerably superior to a decoupled approach where the database lookup to check for validity is performed after the best assignment has been found by the HMM.

| Dataset | Number of elements ($E$) | Number of training instances | Number of test instances |
|---|---|---|---|
| US addresses | 6 | 250 | 490 |
| Student address | 16 | 650 | 1738 |
| Company address | 6 | 250 | 519 |

**Table 2: Datasets used for the experiments**

## 3. EXPERIMENTAL RESULTS

We measure the efficacy of the proposed techniques on real-life address datasets and bibliography databases. We compare our results with prior work on information extraction using rule-learning methods. We quantify the benefits of the nested HMM structure and the hierarchical feature selection steps and measure the sensitivity of our results to the number of training instances.

We do not concern much with running time issues. Our Nested HMM contains less than hundred states, so classification time for addresses using Viterbi is very small. The training time is also within practical limits — our largest dataset completed within an hour.

### 3.1 Datasets

We consider three different real-life address sources as summarized in Table 2.

*US addresses.* The US address dataset consisted of 740 addresses downloaded from an internet yellow-page directory[4]. The addresses were segmented into six elements: `House No, Box No. Road Name, City, State, Zip` as shown in Figure 8.

*Student address.* This dataset consisted of 2388 home addresses of students in the author's University . These addresses were partitioned into 16 elements (described in Figure 10) based on the postal format of the country. The addresses in this set do not have the kind of regularity found in US addresses.

*Company address.* This dataset consisted of 769 addresses of customers of a major national bank in a large Asian metropolis. The address was segmented into six elements: `Care Of, House Name, Road Name, Area, City, Zipcode` as shown in Figure 9.

For the experiments all the data instances were first manually segmented into its constituent elements. In each set, one-third of the dataset was used for training and the remaining two-thirds used for testing as summarized in Table 2.

All tokens were converted to lower case. Each word, digit and delimiter in the address formed a separate token to the HMM. Each record was preprocessed into its corresponding higher-level representation using the automatic preprocessing technique described in Section 2.4. The tree in Figure 5 was used for this preprocessing.

### 3.2 Overall accuracy measures

We obtained accuracy of 99%, 88.9% and 83.7% on the US, Student and Company dataset respectively. The Asian addresses have a much higher complexity compared to the US addresses. The company dataset had lower accuracy because of several errors in the segmentation of data that was handed to us. We dig further into the element-wise accuracy figures to better understand the behavior of DATAMOLD.

In Tables 8, 10, 9 we show the precision and recall values for individual elements. The second column is the total number of tokens over all test data for that element. The precision column shows what percentage of the tokens tagged as that element actually belong to the element. The recall column shows the number of tokens correctly tagged as a percentage of the actual number of tokens for that element.

The table shows that there is a wide variance in the precision of each element. Fortunately, elements like "Designation", "Building Names" and "Landmarks" in Table 10, on which accuracy is low also happen to be less important and occur infrequently in both the training and test sequences. The scarcity of data prevents them from getting trained properly. Also these elements tend to get confused with each other. Elements like Building names, Landmarks and Society are often hard to distinguish even for a human being. For landmarks, some help is provided by the first word which usually is "`Near`" or "`Opp.`" but it is difficult to judge where the landmark ends, unless it has been seen before.

For the company dataset most of the loss in accuracy happened because of the confusion amongst the first three elements arising out of the errors in the training data. Elements that were clearly road-names were wrongly tagged as house-names in the training data

### 3.3 Comparing different automatic approaches

We compare the performance of DATAMOLD with the following three automated approaches.

*Naive-HMM.* This is the HMM model described in Section *2.3.1* with just one state per element. Otherwise it includes all the other optimizations of DATAMOLD including feature selection and smoothing. The purpose here is to evaluate the benefit of the nested HMM model.

*Independent-HMM.* In this approach, for each element we train a separate HMM to extract just its part from a text record, independent of all other elements. Each independent HMM has a prefix and suffix state to absorb the text before and after its own segment. Otherwise the structure of the HMM is similar to what we used in the inner HMMs. Unlike the nested-model there is no outer HMM to capture the dependency amongst elements. The independent HMMs learn the relative location in the address where their element appears through the self-loop transition probabilities of the prefix and suffix states. This is similar to the approach used in [8] for extracting location and timings from talk announcements.

Feature selection and smoothing is done exactly as in DATAMOLD. The main idea here is to evaluate the benefit of simultaneously tagging all the elements of a record exploiting the sequential relationship amongst the elements using the outer HMM.

*Rule-learner.* We compare HMM-based approaches with a rule learner, Rapier [5]. (The code is freely download-

[4]http://www.superpages.com/

| Element | Tokens present | Precision (%) | Recall (%) |
|---|---|---|---|
| House No. | 427 | 99.3 | 99.765 |
| Po Box | 10 | 57.142 | 40.0 |
| Road Name | 1268 | 99.449 | 99.763 |
| City | 611 | 100.0 | 99.509 |
| State | 490 | 100.0 | 100.0 |
| Zipcode | 490 | 100.0 | 100.0 |
| Overall | 3296 | 99.605 | 99.605 |

**Figure 8: US addresses**

| Element | Tokens present | Precision (%) | Recall (%) |
|---|---|---|---|
| CareOf | 278 | 90.604 | 48.561 |
| House Address | 2145 | 77.343 | 88.484 |
| Road Name | 1646 | 78.153 | 73.025 |
| Area | 808 | 91.7 | 83.415 |
| City | 527 | 99.81 | 100.0 |
| Zip Code | 519 | 100.0 | 100.0 |
| Overall | 5923 | 83.656 | 83.656 |

**Figure 9: Company addresses**

| Element | Tokens present | Precision (%) | Recall (%) |
|---|---|---|---|
| Care Of | 281 | 98.091 | 91.459 |
| Department | 99 | 100.0 | 20.202 |
| Designation | 98 | 45.098 | 23.469 |
| House No. | 3306 | 95.681 | 90.471 |
| Bldg. Name | 1702 | 73.204 | 77.849 |
| Society | 3012 | 76.554 | 85.856 |
| Road Name | 2454 | 84.815 | 88.997 |
| Landmark | 443 | 88.338 | 68.397 |
| Area | 2364 | 93.277 | 89.805 |
| P O | 231 | 86.473 | 77.489 |
| City | 1785 | 96.561 | 97.535 |
| Village | 40 | 100.0 | 13.333 |
| District | 138 | 93.333 | 81.159 |
| State | 231 | 96.38 | 92.207 |
| Country | 20 | 100.0 | 45.0 |
| Zipcode | 3042 | 99.967 | 99.934 |
| Overall | 19246 | 88.901 | 88.901 |

**Figure 10: Student addresses**

**Figure 11: Precision and recall values for different datasets shown broken down into the constituent elements.**

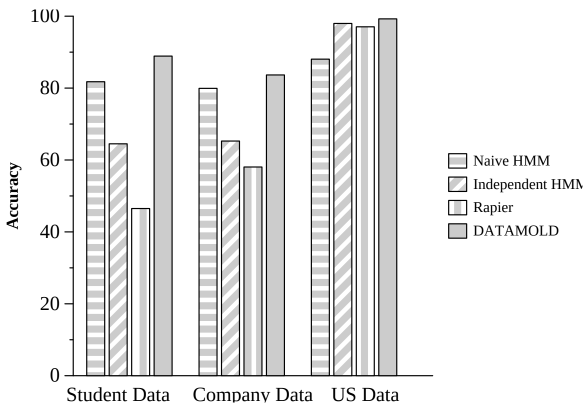

**Figure 12: Comparison of four different methods of text segmentation**

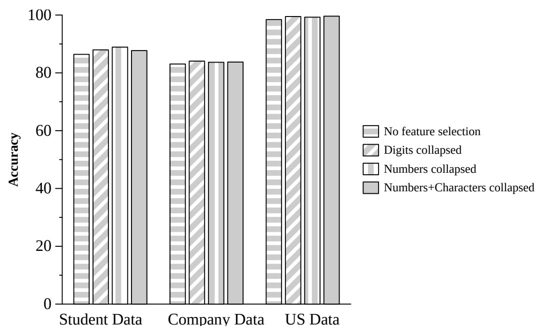

**Figure 13: Accuracy by generalizing features at various levels of the feature taxonomy of Figure 5**

able on the internet [5]). Rapier is a bottom-up inductive learning system for finding information extract rules. It has been tested on several domains and found to be competitive. It uses techniques from inductive logic programming and finds patterns that include constraints on the words, part-of-speech tags, and semantic classes present in the text in and around the tag. Like the Independent-HMM approach it also extracts each tag in isolation of the rest.

Figure 12 shows a comparison of the accuracy of the four methods Naive-HMM, Independent-HMM, Rule-learner and DATAMOLD. Accuracy is defined as the number of tokens correctly assigned to its element as a fraction of the total number of tokens in the test instance. The number of training and test instance for each dataset is as shown in Table 2. We can make the following observations from these results.

- The Independent-HMM approach is significantly worse than DATAMOLD because of the loss of valuable sequence information. For example, in the former case there is no restriction that tags cannot overlap — thus the same part of the address could be tagged as being part of two different elements. With a single HMM the different tags corroborate each other's finding to pick the segmentation that is globally optimal.
- Naive-HMM gives 3% to 10% lower accuracy than the nested-HMM approach of DATAMOLD. This shows the benefit of a detailed HMM for learning the finer structure of each element.
- The accuracy of Rapier is considerably lower than DATAMOLD. Rapier leaves many tokens untagged by not assigning them to any of the elements. Thus it has low recall. However, the precision of Rapier was found to be competitive to our method — 89.2%, 88%, and 98.3% for Student, Company and US datasets respectively. The overall accuracy is acceptable only for US addresses where the address format is regular enough to be amenable to rule-based processing. For the complicated sixteen-element Student dataset such rule-based processing could not successfully tag all elements.

[5]`http://www.cs.utexas.edu/users/ml/rapier.html`, November 2000

## 3.4 Effect of feature selection

In the graphs in Figure 13 we study the effect of choosing features at different levels of the taxonomy tree shown in Figure 5. For each dataset, the first bar shows accuracy without any feature selection, i.e, all features are at the lowest level of the taxonomy tree. The second bar is where individual numbers are grouped based on the number of digits it contains. The third bar is where all numbers are represented as a single symbol irrespective of their length. Finally, the fourth bar is where all individual letters are also aggregated to a single special token. For the Student dataset, the highest accuracy is achieved by the third option that gives a 4% boast in accuracy over not doing any feature selection. The other two datasets also show a slight boast in accuracy with feature selection.

### 3.5 Effect of training dataset size on accuracy

The size of the training data is an important concern in all extraction tasks that require manual effort in tagging the instances. In most such information extraction problems, untagged data is plentiful but tagged data to serve as training records is scarce and requires human effort. We study the amount of training effort needed to achieve peak performance.

In Figure 17 we show accuracies for different sizes of training data with DATAMOLD. The test data for each point in the graph is the total available data as show in Figure 2 minus the data used for training. The results show that HMMs are fast learners. For US addresses (Figure 14), just 50 addresses achieved the peak accuracy of 99.5% on 690 test instances and just 10 addresses yielded an accuracy of 91%. For the Student dataset (Figure 15) with 150 training addresses we get 85% accuracy on 2238 addresses and with 300 addresses reach 89% accuracy on the remaining 2088 addresses. Further increasing the training size only slightly boasts the accuracy. Similar trend is observed for the Company dataset in Figure 16.

### 3.6 Experiments on Bibliography data

The results so far were based on postal addresses. We verify the generality of DATAMOLD by using it for segmenting bibliography references.

The bibliography entries were obtained from two sources. The first source was a collection of PDF files whose references were known to have been generated using bibtex. We extracted the text part of the references in the PDF file (using screen cut and paste), so that each entry was a text string without any of the latex formating tags. The second source was bibliographic references from Citeseer [6]. Citeseer data was chosen to also include examples of references not necessarily generated by bibtex.

The training set had 100 references and the test set had 205 references and the number of elements was 10 as shown in Table 3. The results on DATAMOLD and Rapier are shown in Table 3. DATAMOLD yields an accuracy of 87.3% whereas Rapier although provides high precision suffers on the Recall. DATAMOLD tags all tokens — therefore the overall recall is equal to overall precision. In contrast Rapier leaves many tokens untagged causing the overall accuracy to drop.

The experiments showed that DATAMOLD is an effective tool for accurate segmentation of real-life datasets chosen from three very different domains: US addresses, irregular Asian addresses and bibliographic records. The results are considerably better than state of the art rule learning systems especially for the more complicated domains. The experiments also established the usefulness of the nested HMM model and feature selection. An analysis of the effect of training data size on accuracy established that HMMs are fast learners and in most cases 10% of the available segmented examples was enough to reach within 1% of the peak accuracy.

[6]http://citeseer.nj.nec.com/

| | | DATAMOLD | | Rapier | |
|---|---|---|---|---|---|
| Element | Tokens present | Prec. (%) | Recall (%) | Prec. (%) | Recall (%) |
| Author | 1174 | 88.07 | 86.20 | 0.00 | 0.00 |
| Title | 1591 | 90.26 | 97.86 | 92.60 | 51.31 |
| Conference | 1248 | 81.94 | 90.87 | 92.87 | 70.32 |
| Volume | 158 | 96.43 | 85.44 | 0.00 | 0.00 |
| Pages | 336 | 91.52 | 93.15 | 0.00 | 0.00 |
| Year | 191 | 99.46 | 96.86 | 98.31 | 93.09 |
| Month | 66 | 100.00 | 95.45 | 100.00 | 100.00 |
| Organization | 80 | 86.21 | 31.25 | 0.00 | 0.00 |
| Publisher | 127 | 66.97 | 57.48 | 0.00 | 0.00 |
| Address | 93 | 75.56 | 36.56 | 0.00 | 0.00 |
| Type | 35 | 81.82 | 25.71 | 0.00 | 0.00 |
| School | 13 | 42.86 | 23.08 | 0.00 | 0.00 |
| Note | 24 | 100.00 | 37.50 | 0.00 | 0.00 |
| Editor | 168 | 77.14 | 48.21 | 0.00 | 0.00 |
| Overall | 5304 | 87.35 | 87.35 | 93.46 | 36.48 |

**Table 3: Accuracy of DATAMOLD and Rapier on bibtex data**

## 4. RELATED WORK

The problem of extracting structure from unstructured documents can be addressed at various levels of complexity. On one extreme is the classical semantics extraction problem from free text using sophisticated natural language processing [5]. The other more approachable problem is structure extraction based on syntactic patterns. A popular subproblem in this space is extracting structured fields from HTML documents. Wrappers, as these are popularly called, do shallow information extraction based on syntactic cues present as HTML tags. Except for a few initial systems like TSIMMIS [11] and ARANEUS [22] that are based on manual approaches, most of the recent ones follow the same learn-from-example approach. Example systems of these kind are WEIN [17], SoftMealy [13], Stalker [25, 3], W4F [2], XWrap [20] and [7].

Many of these systems assume that the HTML is a regular list of multi-attribute data as would be machine generated from a database. WEIN [17], one of the early wrappers, deploys a rule-based system to find the left and right HTML tags to separate attributes. Stalker [25] follows the same rule-based technique but hierarchically learns to identify finer and finer structure of the document. The extraction for members within the same hierarchy are learnt independently like in WEIN. Softmealy [13] learns to simultaneously extract multiple elements like we do but they use a deterministic finite state automata (DFAs). DFAs are simpler than HMMs since transitions and output symbols do not have probabilities associated with them. When there is ambiguity in the rules of two out-going edges, they rely on external heuristics to decide which one to pick. In contrast an HMM will explore both possibilities and pick the one that gives highest total probability.

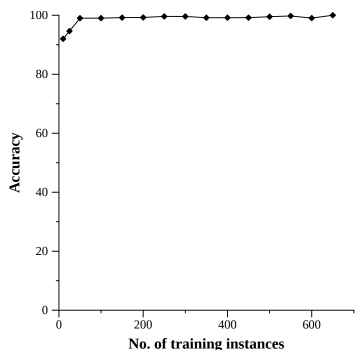

Figure 14: US addresses

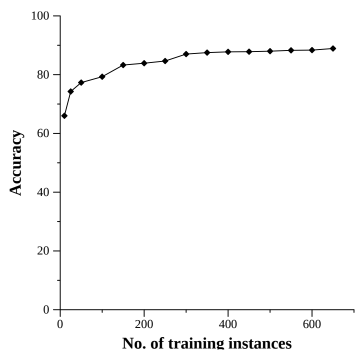

Figure 15: Student Addresses

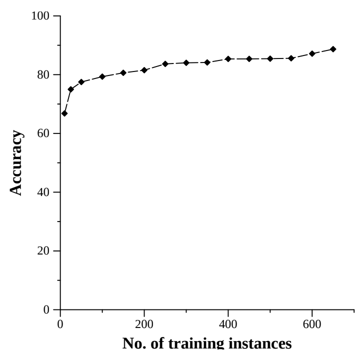

Figure 16: Company Addresses

**Figure 17: Effect of training data size on accuracy for different datasets**

Our problem instances are far more irregular — order of fields is not fixed, not all fields are present in all examples, demarcation between fields is often not present. Also wrappers rely extensively on HTML separator tag and give only secondary importance to the words in the element itself and the length distribution of the words.

We next discuss related work in the more challenging area of information extraction from free text. We discussed in Section 3 Rapier [5], a bottom-up rule learning algorithm. Another rule-based algorithm is Whisk presented in [29]. Nodose [1] is a semi-automated free-text wrapper where the user manually specifies a set of candidate rules for pattern extraction and the system follows a simple generate and verify hypothesis model to find the best pattern.

Hidden Markov Models [8, 9, 28, 21, 4] have been deployed for a few information extraction tasks like extracting "Dates" and "Locations" from talk announcements [9], extracting names and numeric entities like price from free text documents [4], extracting tags like "Title", "Author" and "Affiliation"from headers of computer science research papers. Of these all except [28] extract independent HMMs for each tag in isolation of the others like in Rapier and Whisk [29]. In our experiments we found the independent HMM approach to be inferior to the single HMM approach used in DATAMOLD. Also, DATAMOLD has a number of enhancements over the the basic HMM model used in [28] including, feature selection on a concept hierarchy of the input tokens, incorporating database dependencies through a modified Viterbi algorithm and the nested practical structure learning algorithm.

## 5. CONCLUSION

In this paper we presented an automated approach for segmenting unformatted text into a set of structured elements. This has applications in the crucial address cleaning phase of warehouse construction, reference matching phase of automated citation graph construction and generically in constructing structured queryable databases from unformatted records.

Recently there has been a lot of interest in extracting structure from HTML documents. The text segmentation problem is different and more challenging in that separators between elements is rarely present and data is highly irregular because most of it is human generated — often by different people at different times.

We propose a practical method based on the powerful Hidden Markov Modeling technique. The basic HMM method had to be extended in various ways to solve practical information extraction tasks. We proposed a nested two-level model for learning the structure of the HMM. We introduce a concept hierarchy on the input features for robust smoothing and automatic feature selection. We provide means for tightly integrating information derived from external databases into the basic HMM-based optimizations through a modified optimal Viterbi algorithm. The result is a unified learning model that can simultaneously tag elements exploiting cues from several sources, including, frequently occurring words within an element, partial sequential relationship amongst elements, length distribution of elements, and external databases of relationship amongst symbols. This global optimization driven approach is a departure from existing rule-based systems for information extraction that rely on heuristics to control the order in which rules are fired and extract each element in isolation exploiting only a subset of the information that an HMM can exploit.

Experiments on real-life address and bibliography datasets yield accuracies ranging from 84% and 89% on two complicated datasets of Asian addresses to 99.6% on templatized western addresses and 87.3% on bibliography records. These results were found to be considerably better than a state-of-the-art rule-learning algorithm for information extraction in free text documents. Further experiments proved that HMMs are fast learners and with just 50 to 100 training instances we get close to the maximum accuracy.

Given the encouraging results and the intuitive, human-interpretable nature of the model we believe that the HMM approach is an effective method for several practical information extraction problems.

Future work in the area include correcting and allowing for spelling mistakes in the data, automatically supplying missing fields for some records and exploiting active learning methods to reduce the amount of training data that needs to be manually tagged.

*Acknowledgements.* We would like to acknowledge the contribution of Vijay R. Borkar, Satya Sundar Sahoo and Mamata Desai who painstakingly hand-tagged the test data. We are also indebted to Arvind R. Hulgeri for his constructive suggestions.

## 6. REFERENCES

[1] B. Aldelberg. Nodose: A tool for semi-automatically extracting structured and semistructured data from text documents. In *SIGMOD*, 1998.

[2] Arnaud Sahuguet and Fabien Azavant. Building light-weight wrappers for legacy Web data-sources using W4F. In *International Conference on Very Large Databases (VLDB)*, 1999.

[3] G. Barish, Y.-S. Chen, D. DiPasquo, C. A. Knoblock, S. Minton, I. Muslea, and C. Shahabi. Theaterloc: Using information integration technology to rapidly build virtual applications. In *Intl. Conf. on Data Engineering ICDE*, pages 681–682, 2000.

[4] D. M. Bikel, S. Miller, R. Schwartz, and R. Weischedel. Nymble: a high-performance learning name-finder. In *Proceedings of ANLP-97*, pages 194–201, 1997.

[5] M. E. Califf and R. J. Mooney. Relational learning of pattern-match rules for information extraction. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence (AAAI-99)*, pages 328–334, July 1999.

[6] A. Crespo, J. Jannink, E. Neuhold, M. Rys, and R. Studer. A survey of semi-automatic extraction and transformation. http://www-db.stanford.edu/ crespo/publications/.

[7] D. W. Embley, Y. S. Jiang, and Y.-K. Ng. Record-boundary discovery in web documents. In *SIGMOD 1999, Proceedings ACM SIGMOD International Conference on Management of Data, June 1-3, 1999, Philadephia, Pennsylvania, USA*, pages 467–478, 1999.

[8] D. Freitag and A. McCallum. Information extraction using HMMs and shrinkage. In *Papers from the AAAI-99 Workshop on Machine Learning for Information Extraction*, pages 31–36, 1999.

[9] D. Freitag and A. McCallum. Information extraction with HMM structures learned by stochastic optimization. In *Proceedings of the AAAI 2000*, 2000.

[10] H. Galhardas. *http://caravel.inria.fr/ galharda/cleaning.html*.

[11] J. Hammer, H. Garcia-Molina, J. Cho, R. Aranha, and A. Crespo. Extracting semistructure information from the web. In *Workshop on mangement of semistructured data*, 1997.

[12] M. A. Hernandez and S. J. Stolfo. The merge/purge problem for large databases. In *Proceedings of the ACM SIGMOD*, 1995.

[13] C.-N. Hsu and M.-T. Dung. Generating finite-state transducers for semistructured data extraction from the web. *Information Systems Special Issue on Semistructured Data*, 23(8), 1998.

[14] S. Huffman. Learning information extraction patterns from examples. In S. Wermter, G. Scheler, and E. Riloff, editors, *Proceedings of the 1995 IJCAI Workshop on New Approaches to Learning for Natural Language Processing.*, 1995.

[15] R. Kimball. Dealing with dirty data. *Intelligent Enterprise*, September 1996. http://www.intelligententerprise.com/.

[16] J. Kupiec. Robust part of speech tagging using a hidden Markov model. *Computer Speech and Language*, 6:225–242, 1992.

[17] N. Kushmerick, D. Weld, and R. Doorenbos. Wrapper induction for information extraction. In *Proceedings of IJCAI*, 1997.

[18] P.-S. Laplace. *Philosophical Essays on Probabilities.* Springer-Verlag, New York, 1995. Translated by A. I. Dale from the 5th French edition of 1825.

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

[20] L. Liu, C. Pu, and W. Han. Xwrap: An xml-enabled wrapper construction system for web information sources. In *International Conference on Data Engineering (ICDE)*, pages 611–621, 2000.

[21] A. McCallum, D. Freitag, and F. Pereira". Maximum entropy markov models for information extraction and segmentation. In *In proceedings of ICML-2000*, 2000.

[22] G. Mecca, P. Merialdo, and P. Atzeni. Araneus in the era of xml. In *IEEE Data Engineering Bullettin, Special Issue on XML*. IEEE, September 1999.

[23] A. E. Monge and C. P. Elkan. The field matching problem: Algorithms and applications. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD-96)*, 1996.

[24] I. Muslea. Extraction patterns for information extraction tasks: A survey. In *The AAAI-99 Workshop on Machine Learning for Information Extraction*, 1999.

[25] I. Muslea, S. Minton, and C. A. Knoblock. A hierarchical approach to wrapper induction. In *Proceedings of the Third International Conference on Autonomous Agents*, Seattle, WA, 1999.

[26] L. Rabiner. A tutorial on Hidden Markov Models and selected applications in speech recognition. In *Proceedings of the IEEE, 77(2)*, 1989.

[27] L. Rabiner and B.-H. Juang. *Fundamentals of Speech Recognition*, chapter 6. Prentice-Hall, 1993.

[28] 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.

[29] S. Soderland. Learning information extraction rules for semi-structured and free text. *Machine Learning*, 34, 1999.