Automatically Extracting Structure from Free Text Addresses

Vinayak R. Borkar Kaustubh Deshmukh Sunita Sarawagi
Indian Institute of Technology, Bombay
sunita@it.iitb.ernet.in

Abstract

In this paper we present a novel way to automatically elementize postal addresses seen as a plain text string into atomic structured elements like "City" and "Street name". This is an essential step in all warehouse data cleaning activities. In spite of the practical importance of the problem and the technical challenges it offers, research effort on the topic has been limited. Existing commercial approaches are based on hand-tuned, rule-based approaches that are brittle and require extensive manual effort when moved to a different postal system. We present a Hidden Markov Model based approach that can work with just about any address domain when seeded with a small training data set. Experiments on real-life datasets yield accuracy of 89% on a heterogeneous nationwide database of Indian postal addresses and 99.6% on US addresses that tend to be more templatized.

1 Introduction

Address elementization [7] is the process of extracting structured elements like "Street name", "Company" and "City name" from an address record occurring as a free text string. For example consider the following address. `18100 New Hamshire Ave. Silver Spring, MD 20861`. This address can be elementized as: `House Number : 18100, Street : New Hamshire Ave., City : Silver Spring, State : MD, Zip : 20861.`

Address Elementatization is one of the key steps in the warehouse data cleaning process. Large customer-oriented organizations like banks, telephone companies and universities store millions of addresses. In the original form, these addresses have little explicit structure. Often for the same person, there are different address records stored in different databases. During warehouse construction, it is necessary to put all these addresses in a standard canonical format where all the different fields are identified and duplicates removed. An address record broken into its structured fields not only enables better querying, it also provides a more robust way of doing deduplication and householding — a process that identifies all addresses belonging to the same household. Previous approaches of deduplication without elementization relied on hand tuned rules to define similarity treating the address as a text string [6, 9].

Existing commercial approaches [5] rely on hand-coded rule-based methods coupled with a database of cities, states and zip codes. This solution is not practical and general because postal addresses in different parts of the world have drastically different structures. In some countries zip codes are five digit numbers whereas in others they are allowed to have strings. The problem is more challenging in older countries like India because most street names do not follow a uniform building numbering scheme, the reliance on ad hoc descriptive

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

landmarks is common, city names keep changing, state abbreviations are not standardized, spelling mistakes are rampant and zip codes optional. Further each region has evolved its own style of writing addresses that differs significantly from those of the other region. Consider for instance the following two valid addresses from two different Indian cities:

`7D-Brijdham 16-B Bangur Nagar Goregaon (West) Bombay 400 090`

`13 Shopping Centre Kota (Raj) 324 007`

The first address consists of seven elements: house number: `'7D'`, building name: `'Brijdham'`, building number: `'16-B'`, colony name: `'Bangur Nagar'`, area: `'Goregaon (West)'`, city: `'Bombay'` and zip code: `'400 090'`. The second address consists of the following five elements: house number: `'13'`, Colony name: `Shopping centre`, city: `'Kota'`, State: `(Raj)` and zip code: `'324 007'`. In the first address, 'East' was enclosed in parentheses and depicted direction while in the second the string 'Raj' within parentheses is the name of a geographical State. This element is missing in the first address. Whereas, in the second address building name, colony name and area elements are missing.

We propose an automated method for elementizing addresses based on Hidden Markov Models [12] (HMM). Hidden Markov Modeling is a powerful statistical machine learning technique that can handle new data robustly, is computationally efficient and easy for humans to interpret and tweak. The last property makes this technique particularly suitable for the address tagging problem.

An HMM combines information about multiple different aspects of the record in segmenting it. One source is the characteristic words in each element, for example the word "street" appears in road-names. A second source is the limited partial ordering between its elements. Often the first element is a house number, then a possible building name and so on and the last few elements are zip code and state-name. A third source is the typical number of words in each element. For example, state names usually have one or two words whereas road names are longer. Finally, the HMM simultaneously extracts each element from the address to optimize some global objective function. This is in contrast to existing rule learners used in traditional information tasks [4, 10, 8, 1, 11] that treat each element in isolation.

2 Our Approach

The input to the address elementization problem is a fixed set of E tags of the form "House number", "Road name" and "City" and a collection of T example addresses that have been segmented into one or more of these tags. We do not assume any fixed ordering between the elements nor are all elements required to be present in all addresses. The output of the training phase is a model that when presented with a address record segments it into one or more of its constituent elements.

2.1 Hidden Markov Models

A Hidden Markov Model(HMM) is a probabilistic finite state automaton [12, 13] where each state probabilistically outputs a symbol from a dictionary. Each state transition in the automaton is associated with a probability value. This probability is independent of the output generated by the current state or the next state, and also of the time at which this state is reached. Mathematically a HMM is completely described by two parameters, m and n and two probability distributions A and B. The description of these parameters is as follows.

- n, the number of states in the model.
- m, the number of distinct observation symbols, the discrete dictionary size.
- The state transition probability distribution $A = a_{ij}$ where a_{ij} is the probability of transiting from state i to state j.
- The observation symbol probability distribution, $B = b_j(k)$, where $b_j(k)$is the probability of emitting the k^{th} dictionary symbol in state j

We have shown a typical Hidden Markov Model used for Address Elementization in Figure 1. In this figure, the value of n is 10. The edge labels depict the state transition probabilities (A Matrix).

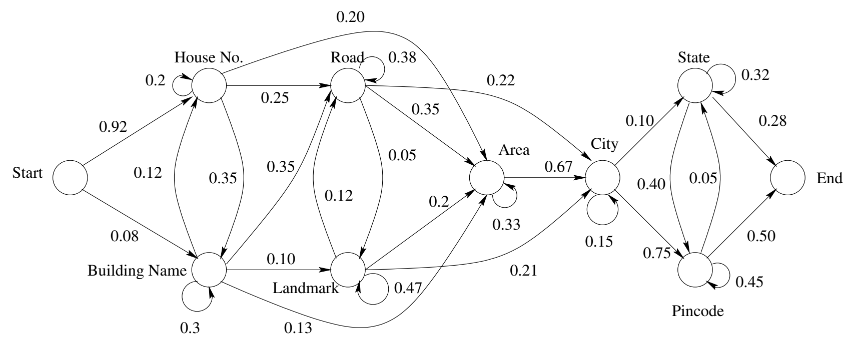

Figure 1: Structure of a typical naive HMM

2.2 HMMs for Address Elementization

This basic HMM model needs to be augmented for segmenting free text into the constituent elements. Let E be the total number of elements. 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 — for each pair $\langle e, s \rangle$ the symbol s can only be emitted from a state marked with element e.

The training process consists of two main tasks. The first task is choosing the structure of the HMM, that is, the number of states in the HMM and edges amongst states. In general, it is difficult to get the optimal number of states in the HMM. We will describe our approach in Section 2.3. The second task is learning the transition and output probabilities of the dictionary symbols. The dictionary consists of all words, digits and delimiters appearing the training data. In [2] we present further refinements on the dictionary where 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 highest accuracy.

Training the HMM The goal during training is finding the A and B matrices such that the probability of the HMM generating these training sequences is maximized. Each training sequence makes transitions from the start state to the end state through a series of intermediate states. When the transitions made by each sequence is known, transition probabilities can be calculated using a *Maximum Likelihood* approach. The probability of transition from state i to state j is computed as,

$$a_{ij} = \frac{\text{Number of transitions from state } i \text{ to state } j}{\text{Total number of transitions out of state } i} \tag{1}$$

The emission probabilities are computed similarly. The probability of emitting symbol k in state j is computed as,

$$b_{jk} = \frac{\text{Number of times the } k\text{-th symbol was emitted at state } j}{\text{Total number of symbols emitted at state } j} \tag{2}$$

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 having some probability. We assume the *Viterbi approximation* and say that the path having the highest probability is the path which generated the given output sequence. Given n states and a sequence of length k, there can be $O(k^n)$ possible paths that can generate the given sequence. This exponential complexity in finding the most probable path is cut down to $O(kn^2)$ by the famous dynamic programming-based *Viterbi Algorithm* [13].

2.3 Learning the structure of the HMM

We impose a nested hierarchical model on the structure of the HMM. An outer HMM captures the sequencing relationship amongst elements treating each element as a single state. Each element's state is expanded to an inner HMM that captures its internal structure.

Accordingly, training of the HMM is done 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 tokens it contains. These sequences are used to train the outer HMM. In the second stage, the inner HMMs are learnt using the procedure described next.

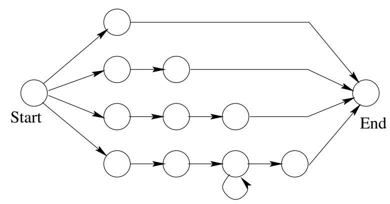

Figure 2: A four length Parallel Path structure

In general, the number of tokens in each element is variable. We handle such variability automatically by choosing a parallel path structure (Figure 2) for each nested HMMs. In the Figure, the start and end states are dummy nodes to mark the two end points of a tag. They do not output any token. 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 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 create some paths that have few training examples leading to a poorly trained dictionary. We merge such paths to its neighboring path as follows. Suppose a k state path needs to be merged with a $k-1$ state path to produce a single merged path. In the k-path, $k-1$ of its state will be merged with those of the $(k-1)$-path. We need to decide which of the k states will not be merged. We try out all possible k choices of this state and choose the one that gives the gives best result on a separate validation dataset. Such merging of paths is continued until a merge does not cause any improvement in accuracy.

3 Results

We measured the efficacy of the proposed techniques on real-life address datasets. We consider three different address sources:

US addresses: A set of 740 US addresses downloaded from an internet directory partitioned into 6 elements as shown in Table 1.

Student address: 2388 home addresses of students in the author's university campus partitioned into 16 elements as described in Figure 3.

Company address: 769 addresses of customers of a major national bank in a large Asian metropolitan broken into 6 elements as shown in Table 2.

In each case one-third of the data was used for training and the remaining two-thirds used for testing. We obtained accuracies of 99.6%,89.3% and 83% on the US, student and company dataset respectively. The non-US addresses have a much higher complexity compared to the US addresses. In Table 4 we show the accuracy partitioned into precision and recall values for individual elements. The table shows that there is a wide variance on the precision of each tag. Fortunately, the tags on which accuracy is low also happen to be the less important tags and occur infrequently in both the training and test instances.

We compared the performance of our approach with a rule learner, Rapier [3]. Rapier is a bottom-up induc-

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
Zip code	490	100.0	100.0
Overall	3296	99.605	99.605

Table 1: US addresses

Element	Tokens present	Precision (%)	Recall (%)
Care Of	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

Table 2: 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
Zip code	3042	99.967	99.934
Overall	19246	88.901	88.901

Table 3: Student addresses

Table 4: Precision and recall values for different datasets shown broken down into the constituent elements.

tive learning system for finding information extraction rules. It has been tested on several domains and found to be competitive. It extracts each tag in isolation of the rest. The accuracy of Rapier was found to be considerably lower than our approach. Rapier leaves many tokens untagged by not assigning them to any of the elements. Thus it has low recall — 98%, 50%, 60% on the US, Student and Company datasets respectively. However, the precision of Rapier was found to be competitive to our method. 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.

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 therefore study the amount of training effort needed to achieve peak performance in our approach. The results show that HMMs are fast learners. For US addresses (Figure 3), 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 4) 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 boosts the accuracy. A similar trend was observed for the Company dataset.

4 Conclusion

Address elementization is a key step of the warehouse data cleaning process and consequently of great practical importance. Surprisingly, research effort on the topic has been limited in spite of the many technical challenges that arise because of the multitudes of ways in which addresses are written. Existing commercial approaches rely on hand-coded rule-based systems that are hard to set up and evolve. In this paper we presented an automated approach for elementizing postal addresses using the powerful technique of Hidden Markov Modeling. We developed practical methods of choosing the best structure of the HMM and ways of incorporating a partial

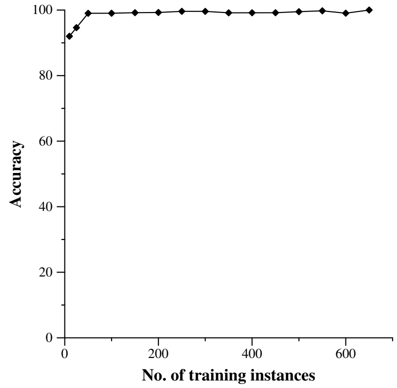

Figure 3: US addresses

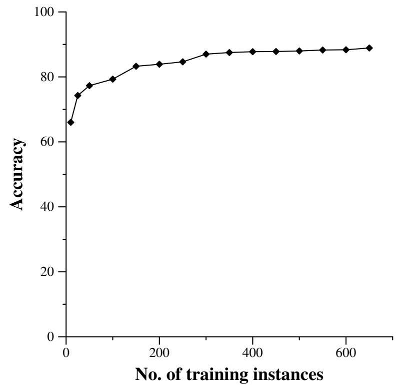

Figure 4: Student Addresses

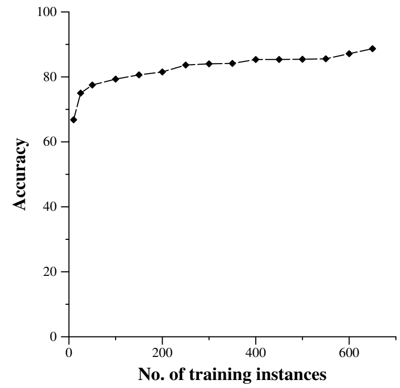

Figure 5: Company Addresses

Figure 6: Effect of training data size on accuracy for different datasets

hierarchical database into the HMM. Experiments on real-life address datasets yield very encouraging results. Given the encouraging experimental results and the intuitive, human-interpretable nature of the model we believe that the HMM approach is an excellent choice for the address elementization problem.

We are further refining our approach by adding support for automated feature selection, incorporating a partial database and reducing amount of training data needed by active learning. Other future work on the topic include correcting and allowing for spelling mistakes in the data and automatically segmenting a large heterogeneous training dataset into subsets to be trained on separate HMMs.

References

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

[2] Vinayak R. Borkar, Kaustubh Deshmukh, and Sunita Sarawagi. Automatic text segmentation for extracting structured records. Submitted for publication, 2000.

[3] Mary Elaine Califf and Raymond 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.

[4] Arturo Crespo, Jan Jannink, Erich Neuhold, Michael Rys, and Rudi Studer. A survey of semi-automatic extraction and transformation. http://www-db.stanford.edu/ crespo/publications/.

[5] Helena Galhardas. *http://caravel.inria.fr/ galharda/cleaning.html*.

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

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

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

[9] Alvaro E. Monge and Charles 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.

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

[11] Ion Muslea, Steve Minton, and Craig A. Knoblock. A hierarchical approach to wrapper induction. In *Proceedings of the Third International Conference on Autonomous Agents*, Seattle, WA, 1999.

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

[13] Lawrence Rabiner and Biing-Hwang Juang. *Fundamentals of Speech Recognition*, chapter 6. Prentice-Hall, 1993.